69
2ej.

FORMATOS GRAFICOS

Universidad Nacional Autónoma de México

Escuela Nacional de Estudios Profesionales

Campus Aragón

PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMPUTACIÓN

P R E S E N T A :

TREJO BONAGA MARILU

A Dios

Por acompañarme en cada día de mi vida.

A mis padres

Por su amor, aliento y confianza.

A mis hermanos

Por su apoyo y comprensión.

Mary.

A Dios

Por darme vida , salud y permitirme terminar esta meta de mi vida.

A mi Madre.

Por su amor , dedicación y esfuerzo que me acompañaron hasta el final de mi carrera. y a quien dedico este logro alcanzado. Esto es para ti. mi Madre.

El que habita al abrigo del altísimo morara bajo la sombra del omnipotente.

Sal. 91.1

A mi Padre.

Por su amor y apoyo moral.

A mis Hermanos.

Que gracias a su esfuerzo. apoyo moral y económico me permitieron alcanzar todos mis objetivos marcados en esta meta de mi vida.

A la Universidad Nacional Autónoma de México. Maestros y Compañeros.

Quienes creyeron en mi. y siempre me apoyaron.

Carlos C. R.

Agradecimiento especial A:

Ismael Manzo Salazar de quien tuvimos gran ayuda para realizar esta tesis.

INDICE

OBJETIVOS

Formatos gráficos pretende introducir al lector en los principales conceptos de la programación gráfica, enfocada principalmente al manejo de imágenes, los objetivos primordiales se mencionan a continuación.

- Conocer los componentes de una PC (tanto hardware como software) para la programación gráfica.
- Adquirir los conceptos básicos de la programación gráfica, en dos lenguajes de programación (Pascal y C).
- Analizar la estructura básica de los formatos gráficos (bitmap).
- Analizar los diferentes métodos de compresión utilizados por los formatos gráficos.
- Aplicar los conocimientos adquiridos de la estructura de los formatos para realizar conversiones entre ellos.
- Realizar un sistema el cual aplique los objetivos planteados.

INTRODUCCIÓN

Programación Gráfica (Imágenes)

Recordemos la frase: "Una imagen vale mas que mil palabras", y estaremos de acuerdo en que la imagen gráfica en el dibujo asistido por computadora ha adquirido gran importancia, ya que si una computadora dispone de funciones gráficas ricas en color y resolución, los medios audiovisuales son susceptibles de enriquecerse.

Desde el momento en que se domina a la programación gráfica y se disponen de elementos para generar imágenes (programas de creación o digitalizadores), el desarrollo de dibujos animados se considera interactivo ya que el espectador pasa de ser pasivo a activo y profundiza el contenido de la presentación. La realización de dibujos o presentaciones de proyectos animados constituye un medio eficaz para mejorar una imagen o de comunicar mensajes. Quien tenga una computadora y tenga el gusto o la necesidad de hacerlo, puede realizar dichas aplicaciones

Modo Texto y Modo Gráfico

Una computadora tiene dos modos básicos de visualización: modo texto y modo gráfico. El modo texto se utiliza para presentar cada uno de los 256 caracteres predefinidos en el código ASCII, es decir, letras mayúsculas o minúsculas, números, símbolos de puntuación o aritméticos y gráficos elementales, la manera de presentar dichos caracteres en pantalla es en zonas rectangulares formadas por un conjunto de puntos. El modo texto más utilizado es aquel que soporta 25 líneas de 80 de cada una e implica que en este modo no podemos visualizar un punto cualquiera en pantalla

El modo gráfico puede visualizar cualquier punto en pantalla por lo que requiere de una tarjeta especial denominada interfaz gráfica, la cual

ofrece algunas ventajas sobre el modo texto. Dentro el ambiente existen diferentes tipos de resoluciones que dependen de la tarjeta gráfica y el monitor conque se disponga, siendo la más baja la correspondiente a 320 puntos de ancho por 200 puntos de alto, en esta resolución un carácter estándar (letra, número, signo o elemento gráfico) ocupara en la pantalla un cuadrado de 8 puntos de largo por 8 puntos de alto, ofrece la posibilidad de tratar un carácter como una imagen además de crear nuevos juegos de caracteres grandes, pequeños, bonitos, iluminados, coloreados y sombreados e incluirlos en dibujos de imágenes digitalizadas, lo que no es posible en un modo texto.

Paralelamente hoy en día los más populares software (Aldus Page Maker, Microsoft Excel, Ventura Publisher, Corel Draw, entre otros) dependen de interfaces de usuarios gráficas (GUI, Graphical User Interfaces) para interactuar eficientemente con el usuario. La creciente popularidad de GUI ha hecho posible el rápido avance en tecnología de gráficos.

Compresión de Imágenes

Para guardar una imagen se requiere de cierto espacio en disco (duro o flexible), que en ocasiones resulta excesivo, lo cual puede resultar un gran problema, por lo que se opta por comprimir la imagen. La compresión de imágenes consiste en la eliminación de información redundante de una imagen mediante métodos de codificación eficientes que derivan en los diferentes formatos que existen actualmente. La operación de compresión y descompresión toma tiempo, la ganancia de espacio resultante depende de los datos de la propia imagen y del método utilizado. El presente trabajo analiza las principales características de algunos formatos así como sus métodos de descompresión y compresión para poder realizar conversiones entre uno y otro de los formatos estudiados.

CAPITULO

I

CAPITULO I

MODOS GRÁFICOS

I.1 Arquitectura de la PC y Sistemas Aritméticos

El mundo de la computación se desarrolla en forma constante y rápida ofreciendo un extenso campo de posibilidades para realizar un sin fin de actividades.

Desde un principio, elementos como el procesador, tarjetas de video y monitores han sido mejorados con el paso del tiempo. Hablaremos de la evolución y características de algunos de los componentes de la computadora que han ayudado al avance de la programación en ambiente gráfico.

I.1.1 Componentes de la PC

Anteriormente las primeras PC´ s contenían una serie de ranuras de expansión de 8 bits (posteriormente se ampliaron a 16 con la llegada de las *AT*), conocidas como *bus ISA* (Industry Standard Architecture). Este bus trabaja a 8 Mhz. de frecuencia y una velocidad de transferencia máxima de 20 Mb/seg. Actualmente se siguen utilizando, por su sencillez y bajo costo, además del gran número de tarjetas existentes para dicho formato.

IBM incluyó a sus computadoras *PS* una arquitectura a la que llamó Micro Channel (o *MCA*). Sus ranuras trabajaban tarjetas hasta de 32 bits, ampliando su velocidad a 10 Mhz., logrando una transferencia máxima de 40 Mb/seg. Aun con éstas características no fue aceptada como se esperaba.

En 1987 se intento mejorar el bus, apareciendo el bus de expansión *EISA* (Extended Industry Standard Architecture). No es más que una extensión del bus ISA, que aumenta las conexiones de 16 a 32 bits. Conservando la velocidad de 8 Mhz. para mantener la compatibilidad, ambos tipos de bus tienen técnicas de gestión avanzada de bus (*bus mastering*). Esta técnica permite a aquellos *dispositivos inteligentes* de expansión tomar el control del bus. Descargando al CPU de las tareas de transferencia, permitiéndole se dedique a trabajos de mayor prioridad.

Algunos de los problemas de las arquitecturas EISA y MCA son: el límite de velocidad del bus, y que el usuario debía adquirir las tarjetas exclusivas de su sistema; olvidándose de aquellas de las que no existiesen versiones EISA o MCA. Esto restringía la adquisición de periféricos.

Aparición del Bus Local

Los sistemas de *bus local* aparece como respuesta al aumento de la velocidad en las computadoras. Dejan a un lado el bus de expansión y se conectan directamente al *bus de datos* del microprocesador y memoria, lo que hace que la velocidad de trabajo de éste tipo de bus dependa de la frecuencia de reloj del procesador. De este modo el bus ISA paso de los 16 Mb/seg. a transferencias de 132 Mb/seg. Sin embargo, el bus local tiene como desventaja el límite de periféricos conectados, generalmente 2.

VESA Local Bus

La Asociación *VESA* (Vídeo Electronic Standard Asociation) creó una patente que aprovechara todas las ventajas del bus local y además fuera accesible al público, naciendo así el bus local VESA (*VL bus*).

El diseño del bus no incluye ningún tipo de memoria adicional, y funciona a la velocidad del procesador. En principio puede parecer una ventaja, sin embargo puede volverse en contra en sistemas de mas de 33 Mhz. por problemas de tolerancia de tiempo. Como los datos a esas velocidades viajan muy rápido el bus necesita añadir *estados de espera* (wait states), para que el periférico tenga la oportunidad de asimilarlos, originando cierta lentitud en la ejecución de procesos. Otra desventaja es la forma en la que los 32 bits se conectan al bus del procesador con un máximo de tres periféricos conectados vía VL bus. Existen dos tipos de tarjetas de las que la tecnología de bus local obtiene un verdadero provecho: las tarjetas de video y las tarjetas controladoras de unidades de disco duro.

PCI

El laboratorio de arquitectura Intel y otras industrias líderes en el mercado, comenzaron en 1991 ha desarrollar un alternativa de alto nivel al bus local, diseñada con miras a los periféricos del futuro. Provocando la estandarización por parte de la *Asociación SIG* (Special Interest Group) de un nuevo sistema de bus local: el *PCI* (Peripherical Component Interconect).

El PCI ofrece una arquitectura más compleja que la VL bus. Las transferencias de datos están controladas por un sistema de gestión entre periféricos y CPU, asegurando un correcto funcionamiento a altas velocidades, sin pérdida alguna de información. Esto se consigue por medio de dos factores: primero el bus local PCI trabaja de forma síncrona a una velocidad máxima de 33 Mhz., incluso con computaras más rápidas; segundo, posee una serie de buffers (zonas de memoria intermedia) que adecuan la velocidad de las tarjetas a la del microprocesador, y permiten la interconexión de un máximo de 10 periféricos (5 en tarjeta y 5 integrados en placa base). Los buffers ayudan a aislar a los periféricos de la unidad central, lo que permite una disminución de ruidos exteriores y un aumento general de la fiabilidad en la transferencia. Además, ofrece la posibilidad de mandar y recibir datos a través de ráfagas lineales (*modo burst*). Esto consiste en trasladar a través del bus grandes cantidades de datos en un solo impulso, lo cual resulta útil, por ejemplo, al grabar archivos de gran tamaño o al visualizar imágenes de alta definición.

El bus PCI ofrece al usuario una forma sencilla y fácil de manejo, a pesar su complejidad interior. No será necesario seleccionar interrupciones ni abrir la computadora para cambiar un puente (*jumper*) de la tarjeta, sino que los registros de configuración y los programas ya implementados se encargan de seleccionar automáticamente las interrupciones. Existen además funciones tale como el soporte de múltiples procesadores, el enfoque de diferentes niveles de bus, o la posibilidad de operar a 64 bits en computadoras Pentium. Sus características lo hacen una opción completa en lo que avances se refiere, sin embargo su precio es algo elevado y hay pocas tarjetas que emplean éste estándar.

Memorias

En la figura I.1[1] se observa como en una PC 286 en adelante se pueden localizar al menos 4 tipos de memoria, descritas a continuación:

- La memoria *RAM*. Almacena programas y datos durante su ejecución. Es de lectura y escritura.
- La *memoria ROM*. Almacena código de arranque de la PC y BIOS, así como los *drivers* de algunos dispositivos.

[1] PC Media, Año I No. 5, "Como funciona", figura 1, p.55.

- La *memoria caché*. Almacena datos que se cree van a hacer falta durante las siguientes instrucciones, evitando así el acceso a la RAM, que es más lenta. Es pequeña y rápida.
- La *memoria CMOS*. Almacena parámetros de configuración de la PC, tales como el número de unidades, el tamaño y tipo de disco duro, la fecha y hora del sistema, etc. Puede leerse y escribirse. Como su información no debe perderse cuando se apaga la PC, la información se mantiene mediante una pequeña pila.

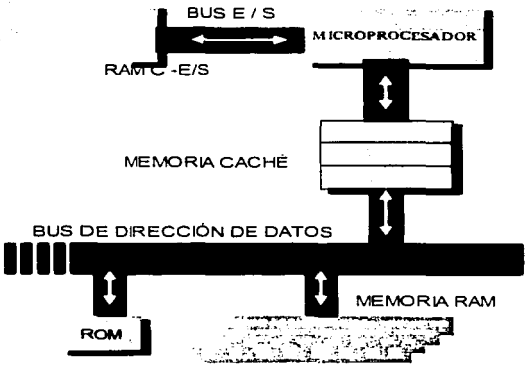


Figura I.1 Tipos de memoria.

Organización de la Memoria RAM

La figura I.2 [2] muestra el mapa de memoria de una PC. Se organiza de la manera siguiente:

- La *memoria convencional*. Son los 640 Kb más bajos del mapa. En ellos se almacena el código y los datos del programa en ejecución, junto al sistema operativo y vectores de interrupción.
- La memoria de video. Se encuentra por encima de la convencional, hasta 128 Kb. La capacidad de ésta memoria varía en función de la placa gráfica que puede aplicarla hasta varios Mb.

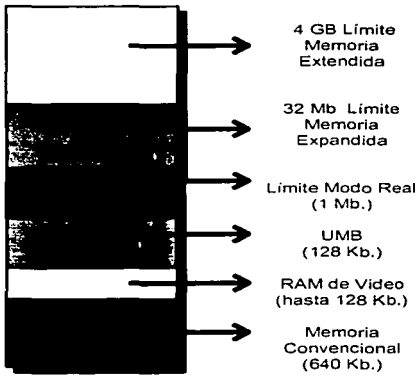


Figura I.2 Esquema de la memoria.

[2] PC Media, Año I No. 5, "Como funciona", figura 2, p.56.

- La memoria *UMB*. Entre la RAM de video y la ROM hay 128 Kb, llamado UMB (Bloques de memoria alta). En el se pueden cargar dirvers, fragmentos del DOS, páginas para acceso de memoria expandida.
- La *memoria extendida*. En modo real (compatible con el 8088), el procesador no puede acceder más allá del primer Mb. La memoria a partir de este primer Mb recibe el nombre de memoria extendida que puede llegar hasta los 4 Gb, sólo se puede acceder con procesadores 286 o posteriores.
- La *memoria expandida*. Acceder a la RAM por encima del primer Mb y puede llegar solo hasta 32 Mb), y acceder a ella mediante un ingenioso mecanismo SW/HW de paginación, tal y como se muestra en la figura I.3 [3].

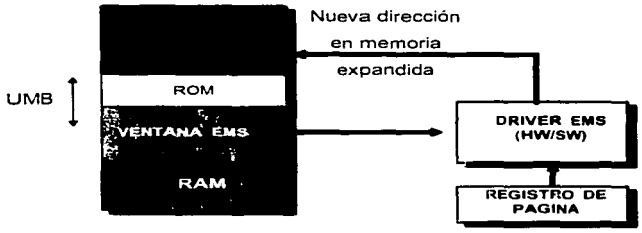


Figura I.3 Mapa de Memoria EMS/XMS.

Memoria Caché

Para evitar que la RAM imponga excesivos estados de espera, se intercala entre ella y el procesador una pequeña cantidad (entre 8 y 512 Kb) de RAM muy rápida, llamada caché. Tal y como se muestra en la figura I.4[4] cuando el microprocesador necesita acceder a memoria, lo primero que hace es acudir a la memoria caché. Si el dato buscado está ahí, lo recupera más rápidamente. De lo contrario, efectúa un acceso a la RAM.

La caché puede admitir diferentes configuraciones, puede estar "intercalada" entre el microprocesador y el bus (look through caché), interviniendo en todos los accesos a memoria, o estar de lado" (look aside caché) sondeando lo que pasa por el bus e interviniendo sólo cuando disponga del dato buscado.

[3] PC Media. Año I No. 5, "Como funciona", figura 3, p.56

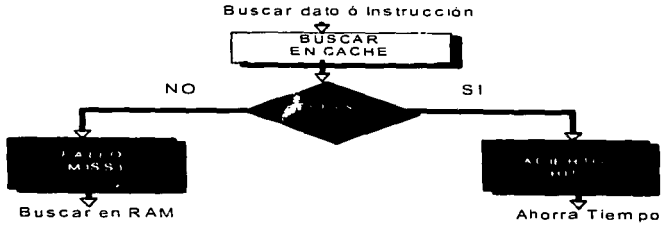


Figura 1.4 Comprobación de datos en la memoria caché.

Tipos de Monitores

Al evaluar los monitores, se debe prestar atención a la frecuencia vertical y a la distancia entre puntos. Éstas cifras ayudarán a evitar gráficos granulados y el pestañeo en la pantalla. Los monitores y adaptadores gráficos se comunican a base de frecuencias. Una tarjeta gráfica envía señales de sincronización vertical y horizontal al monitor. Los cañones de electrones del monitor dibujan las imágenes línea por línea, de izquierda a derecha. Los cañones deben hacer varios cientos de bases horizontales en cada cuadro (320 para CGA, 640 para VGA, 800 para SVGA, etc.). El número de cuadros que el monitor dibuja por segundo se conoce como la frecuencia de cuadros, velocidad de refresco vertical o frecuencia de barrido vertical. Como hay que completar muchos ciclos horizontales antes de completar un ciclo vertical, la frecuencia horizontal se mide en kilohertz (khz), mientras que la vertical se expresa en hertz (Hz).

La distancia entre puntos se refiere al espacio entre los puntos individuales o elementos de pantalla (*pixeles*) que componen la pantalla. Una distancia grande entre puntos produce una imagen granulada, de menos enfoque, mientras que una distancia entre puntos más pequeña produce un imagen más nítida. La distancia preferible no debe superar los 0.25 mm.

La tabla I.1 describe las características de los monitores existentes en el mercado:

MDA. Monochrome Display Adapter
CGA. Color Graphics Adapter
HGC. Hércules Graphics Adapter o MGA Monoichrome Graphic Adapter
EGA. Enhanced Graphics Adapter

[4] PC Media. Año II No. 6."Como funciona". figura 4. p.56.

VGA. Video Graphics Array
SVGA. Super Video Graphics Array
UVGA. Ultra Video Graphics Array

Tabla I.1 Características de los monitores.

Características	MDA	CGA	HGC	EGA	VGA	SVGA	UVGA
Año de aparición en el mercado	1981	1981	1982	1984	1987	1989	1991
Resolución	NO	320 x 200	720 x 348	640 x 360	640 x 480	1024 x 768	1280 x 1024
Número de colores	2	4	2	26	16	256	16M
Memoria de video	4 Kb	16 Kb	64 Kb	256 Kb	> 256 Kb	> 512 Kb	> 516 Kb

I.1.2 Tarjetas de Video

El principal objetivo de una tarjeta de video o adaptador de video consiste en convertir los datos digitales suministrados por el procesador, en señales de diferente intensidad que se transmiten al monitor.

El monitor interpreta dichas señales para formar las imágenes que pueden verse en su pantalla. Cualquier imagen mostrada en un monitor esta compuesta por pequeños puntos, llamados pixeles. Un *pixel* es la unidad más pequeña que puede observarse en pantalla. Para codificar una imagen en formato digital, es necesario almacenar el valor que toma el color de todos y cada uno de los pixeles mediante series de bits. Este valor nace como resultado de medir la cantidad de componentes Rojo, Verde y Azul (los tres colores básicos) que existe en cada pixel, figura I.5[5]. El número de bits dedicado a cuantificar ésta información, también denominado "paleta de colores", determinará la cantidad de colores disponibles en una imagen. Utilizando 8 bits por pixel se consiguen 256 colores (2^8). Mientras que con 24 bits se tienen más de 16 millones (2^{24} = 16,777,216), que son más que los que el ojo humano puede percibir.

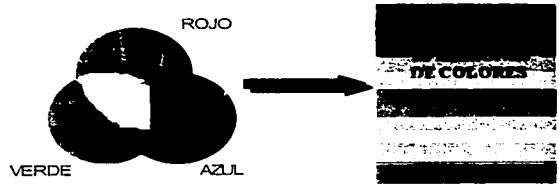


Figura I.5 Composición de colores.

[5] PC Media, Año II No. 6."Como funciona". p.55.

Componentes de la Tarjeta

Los tres elementos claves de una tarjeta de video son la memoria, el chip de video y el bus (figura I.6[9]). La memoria almacena las imágenes. Pueden encontrarse dos tipos de memoria en las tarjetas de video: VRAM y DRAM. La diferencia estriba en el canal de distribución (medio a través del cual se envían y reciben datos). La DRAM sólo posee un sistema dual que le permite enviar y recibir al mismo tiempo. Supone un mayor rendimiento, aunque su precio aumenta con diferencia.

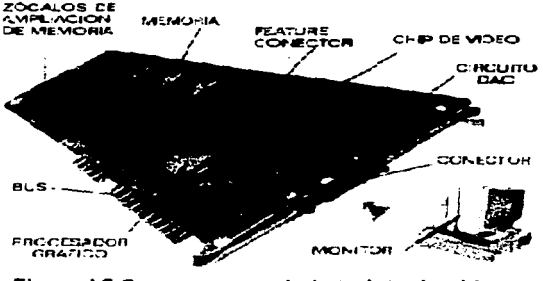


Figura I.6 Componentes de la tarjeta de video.

El chip de video traslada la información de la memoria al monitor, controlando el haz de luz que ilumina a los pixeles. De esta operación se deriva el concepto de velocidad de refresco o número de veces que se "redibuja" una imagen en la pantalla del monitor. Se mide en hertz (hz), y de su valor dependerá que la imagen parpadee o permanezca nítida. Las señales digitales enviadas por el sistema operativo o el software del usuario llegan al chip. Posteriormente éste las redirecciona a un circuito *DAC* (Convertidor Digital Analógico) que convierte lo valores numéricos en voltajes de mayor o menor intensidad enviados al monitor. En el circuito se dan cita tres convertidores DAC, uno para cada color primario (Rojo, Verde y Azul), que generalmente, estarán contenidos en el mismo chip. El último factor determinante de la eficiencia de la tarjeta de video es el bus, anteriormente los bus eran de 8 bits hoy pueden encontrarse bus de 16, 32 y hasta 64 bits.

Comunicación con la Tarjeta de Video

Existen tres formas de comunicarse con la tarjeta de vídeo: a través del BIOS, mediante puertos y por direcciones de memoria. A los servicios de video del BIOS se accede por medio de la interrupción 10h. El valor que contengan ciertos registros determinará el comando y los parámetros que se deseen efectuar. El BIOS ahorra trabajo a los programadores, ya que se encarga realmente de interactuar con la tarjeta. Sin embargo su lentitud lo hace poco recomendable para aplicaciones

[9] PC Media. Año II No. 6."Como funciona". p.54

que requieran un uso intensivo de gráficos. Por ello sólo puede emplearse para realizar cosas complicadas y no muy continuas, como cambiar el modo gráfico. La tarjeta de video se conecta a la computadora a través de una serie de puertos que posee para comunicarse con el exterior. Para ello, se escriben y leen valores a los puertos que tienen asignados la tarjeta. Desafortunadamente éste número resulta insuficiente, y debido al tiempo de utilización de puertos, éste método no es muy usado.

El sistema más común de trabajar con la tarjeta gráfica consiste en hacer responder directamente los pixeles de la imagen con direcciones de memoria. La zona de memoria dedicada a comunicación con la tarjeta ocupa un segmento, esto es, 64 Kb. De ésta forma, si se trabaja en un modo de 320 x 200 x 256 (que ocupa 64 bytes, menos de 64 Kb), para dibujar un punto sólo habrá que modificar el color del pixel correspondiente escribiendo en dicha zona de memoria. Sin embargo cuando el modo gráfico requiere más memoria, como muchos modos VGA y todos los SVGA resulta más complicado. La solución está en dividir la pantalla en trozos o bancos que quepan en esos 64 Kb e ir cambiando de banco dependiendo de donde se dibuje. Otro método consiste en tratar la imagen como si estuviera compuesta por planos superpuestos, cada uno encargado de almacenar la intensidad de un cierto color. En cualquiera de los dos casos y como conclusión a mayor resolución y uso de la paleta de colores, los cálculos para trabajar con los modos de vídeo de más calidad aumentan considerablemente.

Aceleradores Gráficos

Para agilizar la visualización surgen las tarjetas aceleradoras de video. La diferencia con las tarjetas de video es un procesador capaz de ejecutar independientemente del procesador funciones gráficas. Para dibujar un rectángulo había antes que calcular todos los puntos que los componían y dar las órdenes para pintar cada uno de los pixeles, ahora sólo debe darse una única orden en la que se dice que lo que se quiere dibujar es un rectángulo con unas ciertas dimensiones, siendo posible que los programas,

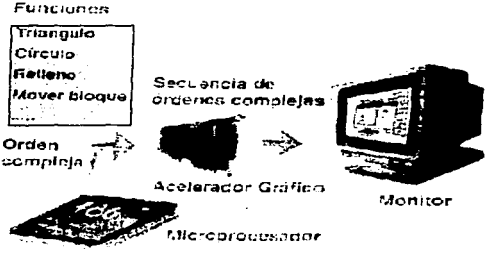


Figura I.7 Aceleradores Gráficos.

en especial, entornos gráficos tipo Windows u OS/2 aumenten su rendimiento, ya que muchos de sus gráficos se derivan a partir de primitivas geométricas, figura I.7[7]. La forma en que se implementan éstas funciones gráficas en los programas es a través de una secuencia de "escape". Cuando el procesador de la computadora encuentra dicha secuencia durante la ejecución de un programa, envía el conjunto de las instrucciones que siguen al bus de datos directamente, y no las ejecuta, las pasa a la tarjeta de video quién estará pendiente para ejecutar dicho conjunto.

Existen además otro tipo de tarjetas de video, como los *framebuffers* o los coprocesadores programables de video.

I.1.3 Aritmética Binaria, Decimal y Hexadecimal

Una PC manipula elementos discretos de información, circuito cerrado y circuito abierto (ausencia o paso de corriente), por lo tanto la computadora trabajará exclusivamente en aritmética binaria (operadores, dígitos decimales, letras del alfabeto, etc.), al igual que el procesamiento de datos se realiza mediante elementos lógicos binarios, usando señales binarias.

Un número decimal tal como 7,392 representa:

7 millares	7000 +
3 centenas	0300 +
9 decenas	0090 +
2 unidades	0002
	7392

Los millares, las centenas, etc. son potencias de 10 indicadas por la posición de los coeficientes; por ejemplo, la cantidad antes mencionada puede ser escrita de la siguiente manera:

$$7\times10^3 + 3\times10^2 + 9\times10^1 + 2\times10^0 = 7000 + 300 + 90 + 2 = 7392$$

Se dice que el sistema de números decimales tiene la base o raíz 10 debido a que usa diez dígitos (0,1,2,3,4,5,6,7,8,9) y que los coeficientes son multiplicados por potencias de diez.

El sistema binario es un sistema numérico diferente. Los coeficientes del sistema de números binarios tienen dos valores posibles: 0 y 1 a los cuales en el ámbito de las computadoras se les conoce como bits (representa la mínima cantidad de información que se puede almacenar), por consiguiente su base o raíz es 2, por ejemplo, la cantidad antes mencionada puede ser escrita de la siguiente manera: 1110011100000 = 7392

[7] PC Media, Año II No. 6,"Como funciona", p.56

$$1x2^{12} + 1x2^{11} + 1x2^{10} + 0x2^{9} + 0x2^{8} + 1x2^{7} + 1x2^{6} + 1x2^{5} + 0x2^{4} + 0x2^{3} + 0x2^{2} + 0x2^{1} + 0x2^{0} = 7392$$
$$4096 + 2048 + 1024 + 0 + 0 + 128 + 64 + 32 + 0 + 0 + 0 + 0 + 0 = 7392$$

Como se observa en el ejemplo anterior un número binario puede ser convertido a decimal formando la suma de las potencias de base 2 de aquellos coeficientes cuyo valor sea 1.

Es costumbre presentar los n dígitos necesarios para los coeficientes de un sistema que contenga menos de 10 dígitos, para los sistemas que utilizan más de diez dígitos se utilizan como complemento las letras del alfabeto. Por ejemplo, en el sistema hexadecimal (base 16) se representan los primeros 10 dígitos del sistema decimal, las letras A, B C ,D E y F se usan para representar los números 10, 11, 12, 13 , 14 y 15 respectivamente, así como en el ejemplo anterior la cantidad antes mencionada puede ser escrita de la siguiente manera:

$$1CE0 = 7392$$
$$1 \times 16^{3} + C \times 16^{2} + E \times 16^{1} + 0 \times 16^{0} = 7392$$
$$1 \times 16^{3} + 12 \times 16^{2} + 14 \times 16^{1} + 0 \times 16^{0} = 7392$$
$$4096 + 3072 + 224 + 0 = 7392$$

Como puede observar en un sistema hexadecimal la representación de información puede verse en un menor número de dígitos que en los dos sistemas anteriores.

Así por ejemplo en el número 65,535 en decimal, en binario sería 1111111111111111 y en hexadecimal sería solamente FFFF, cuatro dígitos.

En la Tabla I.2 se muestra una comparación de los sistemas numéricos mencionados.

Tabla I.2 Comparación de sistemas numéricos.

BASE 10 (DECIMAL)	BASE 2 (BINARIO)	BASE 16 (HEXADECIMAL)
0	0	0
1	1	1
2	01	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Tabla I.2 Comparación de sistemas numéricos. (continuación)		
BASE 10 (DECIMAL)	BASE 2 (BINARIO)	BASE 16 (HEXADECIMAL)
16	1 0000	10
17	1 0001	11
18	1 0010	12
19	1 0011	13
.....	.....	.....
255	1111 1111	FF
.....	.....	.....
65 536	1111 1111 1111 1111	FFFF

I.2 CGA, EGA, VGA, SVGA

Existen diversos modos de video, cada uno de ellos dependen del equipo (*hardware*) con que se cuente, es decir, tarjeta de video y monitor.

Entre las tarjetas más comunes están: la CGA, EGA, VGA y SVGA, en cada una de éstas tarjetas su principal diferencia es la capacidad para soportar determinados modos de video.

I.2.1 Modo texto y Modo Gráfico

La computadora tiene dos modos de video: de texto y gráfico. En modo texto, la computadora puede visualizar cualquiera de los 256 caracteres ASCII que soporta. Estos caracteres están permanentemente en la memoria de la computadora, por lo que sabe como dibujarlos.

La Tarjeta de Video y la Memoria de Visualización

Un monitor puede mostrar 80 caracteres horizontalmente y 25 verticalmente (un total de 2,000 caracteres en toda la pantalla), los monitores EGA tiene un modo de 43 líneas y los monitores VGA tienen otro de 50 líneas o más. Cada caracter tiene su propio color de fondo (*background*) y su color de primer plano (*foreground*). El color de primer plano es el del propio caracter; el color de fondo es el color de la zona sobre la que está el caracter.

La computadora guarda la información de los caracteres y del color en una zona especial de memoria conocida como memoria de visualización. Esta memoria es la que le indica a la computadora los caracteres y colores a visualizar. Aunque la memoria de visualización se encuentra en la tarjeta de video, se considera como una parte de la RAM. El primer byte de la memoria de

visualización contiene el primer caracter del monitor, que es el que aparece en la esquina superior izquierda.

De este modo, el primer byte de la memoria de visualización contiene el valor 41h (el valor ASCII en hexadecimal de la letra "A"), el monitor visualiza la letra "A" en la esquina superior izquierda del monitor. El segundo byte de visualización es el byte de atributos del primer caracter; los bytes de atributos controlan el color y otros aspectos de la visualización de cada caracter. El patrón -caracter, atributo, caracter, atributo- se repite por lo menos 2,000 a 4,000 veces para caracteres que aparecen en el monitor.

El Byte de Atributos

Una tarjeta de video a color con monitor en color es capaz de mostrar colores diferentes, cada uno de los cuales puede estar formado hasta por cuatro elementos: azul, rojo, verde y atributo. El color que muestra la computadora depende de la combinación particular de éstos elementos. Por ejemplo el color negro no combina ningún elemento, mientras que el cían claro combina el azul, el verde y el atributo. La tarjeta de video también soporta un elemento de parpadeo que hace que el caracter aparezca y desaparezca rápidamente y que no tienen ningún efecto sobre el color.

Un byte de atributos guarda el color de fondo y de primer plano del byte de caracter que le precede. La figura I.8 muestra como contribuye cada bit del byte de atributos en el color del caracter y su fondo.

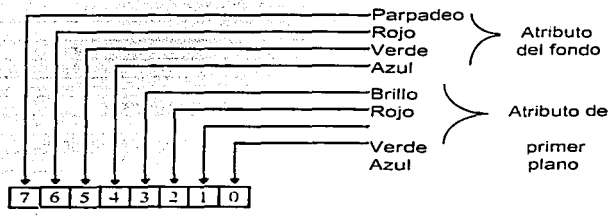


Figura I.8 Correspondencias del byte de atributos.

Los tres primeros bits (0,1 y 2) del byte de atributos controlan el color de primer plano del caracter, mientras que el cuarto bit (bit 3) le da brillo al color. Se pueden combinar estos cuatro elementos para crear 16 colores de primer plano diferentes. Los bits 4, 5 y 6 determinan el color de fondo del caracter, y el bit 7, cuando esta activado, hace que el caracter parpadee. Solo hay disponibles 8 colores para el fondo, puesto que el bit de parpadeo reemplaza al bit de brillo.

En el monitor monocromo, el byte de atributo sólo puede crear cinco formatos de visualización: oculto, normal, brillo, subrayado e inverso. Igual que con la tarjeta de video en color, el brillo lo controla el bit 3 y el parpadeo lo controla el bit 7. Cuando el color de primer plano y el color de fondo son iguales, los caracteres son ocultos, los bits de brillo y parpadeo no tienen efecto sobre los caracteres ocultos.

Modo Texto

Las computadoras soportan varios modos texto. Algunos de estos modos se listan en la Tabla I.3, indican el tamaño de los caracteres y los colores de visualización. El modo 0, por ejemplo, muestra caracteres grandes (40 por línea) en tonos de gris.

Los cuatro primeros modos de visualización, del 0 al 3, sólo funcionan con adaptadores de gráficos en color, la mayoría de los cuales pueden cambiar entre los cuatro modos. El modo 7, por otro lado, solo lo utilizan los adaptadores monocromáticos. El modo 256 se utiliza junto con los modos del 0 al 3 (mediante el operador OR) para proporcionan visualizaciones de 43 o 50 líneas.

Tabla I.3 Constantes de modo texto

Modo	Tamaño	Colores/Adaptador
0	40x25	Blanco y Negro
1	40x25	Color
2	80x25	Blanco y Negro
3	80x25	Color
7	80x25	Adaptador Monocromo
256	80x43/50	Solo EGA/VGA

Comparándolo con el modo gráfico, el modo texto es fácil de usar. Mostrar información en la pantalla es tan sencillo como colocar caracteres ASCII en posiciones de memoria específicas. La pantalla en texto está dividida en 80 columnas y 25 filas y la computadora misma ya sabe como trazar los caracteres ahorrando muchas molestias.

Modo Gráfico

El modo gráfico en lugar de bytes de caracteres y atributos, se tienen pixeles, los elementos de dibujo más pequeños del monitor. Un caracter en la pantalla de la computadora está formado por muchos pixeles organizados en un patrón. En el modo gráfico se pueden activar pixeles en cualquier parte del monitor ordenados en líneas horizontales y verticales. Esto es válido para todos los modos gráficos de la computadora; la diferencia esta en el tamaño del pixel. En el modo de baja resolución del CGA los pixeles son bastante grandes por lo que solo caben 320x200 , en el adaptador VGA se aprecia una alta resolución con pixeles que abarcan de 640x480 hasta en un SVGA de 1024x768 y en un UVGA de 1280x1024.

I.2.2 Historia de los Modos Gráficos

IBM introdujo el adaptador VGA (Vídeo Graphics Array) en 1985 como un adaptador standard para sus computadoras PS/2, el VGA ha sido muy aceptado como el adaptador de tecnología de hoy y mañana.

A diferencia de la mayoría de los adaptadores anteriores cuyos drivers despliegan con un TTL, interface digital, el adaptador VGA maneja un despliegue análogo. Esto hace posible para el adaptador VGA desplegar algunos colores más que otros adaptadores (incluyendo el EGA). Además significa que el adaptador VGA es compatible con otros adaptadores existentes.

A medida que el VGA y su monitor análogo se han generalizado, algunos programas escritos para otros adaptadores de despliegue de color no pueden operar con adaptadores monocromáticos, y viceversa. En VGA si un adaptador monocromático es conectado, la información de color es automáticamente convertida a sombras de grises. La información monocromática puede también ser mostrada en una pantalla a color.

El VGA es un dispositivo de despliegue no inteligente, no tiene capacidad de procesamiento. El procesador debe ejecutar todas las funciones de dibujo por escritura directamente a la memoria de despliegue. La escritura de un bit en la memoria de despliegue es equivalente a iluminar un pixel en pantalla. La mayor parte de la arquitectura del VGA está dedicada a la tarea de transferir el dato de memoria de despliegue a la pantalla de despliegue. Este proceso llamado despliegue de refresco, se ejecuta entre 50 y 70 por segundo.

El número de colores que pueden ser desplegados en pantalla en un tiempo está determinado por el número de bits de memoria de despliegue que están dedicados a la información de color por cada pixel. Si "n" bits por pixel son usados, 2^n colores pueden ser generados. VGA usa desde uno a 8 bits por pixel, permitiendo subir a 256 (2^8) colores para ser desplegados en pantalla al mismo tiempo.

IBM definió un grupo standard de modos de despliegue para el VGA. Vendedores de SVGA han añadido algunos a la lista de modos estándar por la creación de nuevos modos de despliegue de alta resolución. Estos modos no representan todas las configuraciones en las cuales la tarjeta puede operar. La Tabla I.4 lista algunos modos de despliegue:

Tabla I.4 Modos de video.

Modo	Tipo	Resolución	Colores
0,1	texto	40 columnas x 25 líneas (320x200,8x8 caracteres por celda)	16
0	texto	40 columnas x 25 líneas (320x350,8x14caracteres por celda)	16
0+	texto	40 columnas x 25 líneas (320x400,9x16 caracteres por celda)	16
2,3	texto	80 columnas x 25 líneas (640x200,8x8 caracteres por celda)	16
2	texto	80 columnas x 25 líneas (640x350,8x14 caracteres por celda)	16
2+,3+	texto	80 columnas x 25 líneas (640x400,9x16 caracteres por celda)	16
4,5	gráfico	320 horizontal 200 vertical	4
6	gráfico	640 horizontal 200 vertical	2
7	texto	80 columnas x 25 líneas (720x350,8x14 caracteres por celda)	Mono
7+	texto	80 columnas x 25 líneas (720x400,9x16 caracteres por celda)	Mono
D	gráfico	320 horizontal 200 vertical	16
E	gráfico	640 horizontal 200 vertical	16
F	gráfico	640 horizontal 350 vertical	Mono
10h	gráfico	640 horizontal 250 vertical	16
11h	gráfico	640 horizontal 480 vertical	2
12h	gráfico	640 horizontal 480 vertical	16
13h	gráfico	320 horizontal 200 vertical	256
1B	texto	132 horizontal 25 vertical	16
26	gráfico	640 horizontal 480 vertical	256
27	gráfico	800 horizontal 600 vertical	256

Desde el adaptador CGA, todos los adaptadores han incluido 40 columnas en modo texto. Estos modos fueron creados para permitir que texto pueda ser desplegado en aparatos de televisión caseros los cuales tienen más pobre resolución que los despliegues de computadoras y no pueden desplegar 80 columnas de texto. Otro pequeño número de juegos de computadora quienes han sido escritos usando 40 columnas de texto, estos modos no son usados comúnmente.

Un diseño de circuito especial de tarjeta es requerida para conectar una computadora a un aparato de televisión (a menos que el aparato de TV pueda aceptar un compuesto de entrada de video).

Modos de Despliegue

- Modos 0 y 1 (40 columnas de texto a color). En el VGA, no hay una diferencia funcional entre el modo 0 y 1. Estos modos fueron traídos con el CGA. Tienen una resolución de texto a color de 40 x 25. La compatibilidad CGA no es completa, y no todo el *software* CGA puede correr correctamente en estos modos. En general, el software que hace uso del servicio de video BIOS y evita algún acceso directo para registros de entrada/salida en la tarjeta de video correrá sin problemas. Los procesos de acceso directo a la memoria de despliegue no causan problemas de compatibilidad.

- Modos 2 y 3 (80 columnas de texto a color).Los modos 2 y 3 son los homólogos de 80 columnas de los modos 0 y 1 de 40 columnas. En el VGA, no hay una diferencia funcional entre el modo 2 y 3. Así como los modos 0 y 1, estos dos modos se presentan en la tarjeta de video CGA. Poseen una resolución de texto a color de 80 x 25 .

  Doble Scaneo.Cuando opera en modo gráfico compatible CGA, la tarjeta VGA usa una técnica conocída como " Doble escaneo " para desplegar la baja resolución (200 líneas) de CGA despliega en la alta resolución (400 líneas scan) de despliegue VGA. Cada una de las 200 líneas scan horizontales es desplegada dos veces, incrementando la resolución vertical de la pantalla de 200 líneas a 400. Esto mejora la calidad del despliegue y ayuda a compensar el aspecto de razón de diferencia para el despliegue VGA. El doble escaneo es usado para los modos 4,5,6,D y E.

- Modos 4 y 5 (Gráficos de 4 colores 320 x 200). Los modos 4 y 5 son modos gráficos CGA muy populares que fueron también traídos a EGA y VGA. La resolución de despliegue es de 320x200 pixeles. El VGA usa doble escaneo para incrementar estas a 400 líneas verticalmente. Cuatro datos de color de pixel están almacenados en un formato de paquete de pixel con dos bits por pixel. Así como todos los modos estándares CGA, la compatibilidad no es completa. El software que escribe directamente al registro I/O del CGA pueden no funcionar correctamente en VGA. El

Software que usa llamadas de BIOS par configurar los registros usualmente operaran correctamente.

Modos Gráficos CGA

Estos modos presentan un inusual grupo de desafíos para los programadores gráficos porque la memoria de despliegue no es linealmente mapeada. Es requerido un cálculo para transformar de una posición de pixel en pantalla a una dirección en la memoria de despliegue.

- Modo 6 (Gráficos de 2 colores 640 x 200).El modo 6 es el modo gráfico de mas alta resolución del CGA, transportado a VGA. Una resolución de pantalla de 640x 200 líneas es soportada, pero solo en dos colores. El VGA usa doble escaneo para incrementar estas a 400 líneas verticalmente. Así como todos los modos estándares CGA, la compatibilidad no es completa. Los software que escriben directamente al registro I/O del CGA pueden no funcionar correctamente en EGA. Los software que hacen uso de llamadas del BIOS para configurar los registros usualmente operaran correctamente. Como se explicó para los modos 4 y 5, la memoria de despliegue no es linealmente mapeada. Un cálculo es requerido para traducir desde la posición de un pixel en pantalla a una dirección en la memoria de despliegue.

- Modo 7 (Texto monocromático). En el modo 7 el VGA es software parcialmente compatible con el adaptador MDA. El despliegue es de formato 80x 25 caracteres por fila. En modo texto monocromático, los atributos de caracter no controla el color de caracter pero representa otras características de despliegue. Los atributos de texto monocromático incluye caracteres de parpadeo, intensidad, subrayado y vídeo reversa.

- Modo D (Gráficos de 16 colores 320 x 200). A diferencia de los modos descritos anteriormente, este modo no es compatible con los anteriores modos CGA o MDA, existe solo para EGA y VGA. Es aproximadamente un patrón después del modo 4 (Gráficos CGA de 4 colores) pero ofrece más colores. La resolución limitada del modo D (320 por 20) lo hace indeseable para nuevas aplicaciones de software, peor aun no es software compatible con algunas aplicaciones viejas. El resultado es que el modo D raramente es usado. El VGA usa doble escaneo para incrementar el tamaño de pantalla a 400 líneas verticalmente. El modo D no sufre de el mapeo de

memoria no lineal que el modo gráfico CGA compatible hace, la transformación de una posición de pixel en pantalla a la dirección de memoria de despliegue es relativamente sencilla.

- Modo E (Gráficos de 16 colores 640 x 200). Parecido al modo D, el modo E existe solo para le EGA y VGA. Aparece después del modo 6 CGA (Gráficos de 2 colores), pero ofrece más colores. Su resolución es limitada (640x200) lo hace impopular para nuevas desarrollos de software y no es compatible con algunos software más viejos existentes. El resultado es que el modo E raramente es usado. El VGA usa doble escaneo para incrementar el tamaño de pantalla a 400 líneas verticales. El modo E no sufre de el mapeo de memoria no lineal que el modo gráfico CGA compatible hace, la transformación de una posición de pixel en pantalla a la dirección de memoria de despliegue es relativamente simple.

- Modo F (Gráficos monocromáticos 640 x 350). El modo gráfico F es único para el EGA y VGA. La resolución en el modo F es de 640 por 350 que es menor que la resolución de 720 por 348 del adaptador gráfico monocromático Hércules. El modo D no sufre del mapeo de direccionamiento de memoria de despliegue no lineal del adaptador Hércules. La memoria de despliegue es linealmente mapeada. Dos planos de color de memoria de despliegue son usados dando a cada pixel monocromático cuatro atributos. Estos atributos son:

  00 - negro
  01 - blanco
  10 - parpadeo
  11 - intensidad

Los planos de memoria pueden ser habilitados o deshabilitados independientemente por escritura al registro de plano habilitado, indexa dos en la secuencia.

- Modo 10 (Gráficos de 16 colores 640 x 350).El modo 10 que es único para el EGA y VGA, es el modo más popular para nuevas aplicaciones de gráficos a color. Soporta una resolución de 640x350. Cuatro planos de colores son usados, con rendimiento de 16 colores simultáneos. Los planos de colores son habilitados y desahibilitados por escritura al registro de plano habilitado en la secuencia.

- Modo 11 (Gráficos de 2 colores 640x480).Modo 11 soporta el IBM VGA en su más alta resolución estándar (640x480), pero soporta solo dos colores simultáneos. Este modo puede ser usado para desplegar 30 líneas de 80 columnas de texto.

- Modo 12 (Gráficos de 16 colores 640x480). El modo 12 soporta el VGA en su más alta resolución (640x480), con 16 colores simultáneos. Este es un modo popular para nuevas aplicaciones gráficas a color. Cuatro planos de colores son usados, produciendo 16 colores simultáneos. Los planos de colores son habilitados y deshabilitados por escritura al registro de habilitación de plano, indexa 2, en el secuenciador.

- Modo 13 (Gráficos de 256 colores 320 x 200). Este modo es único para el VGA, es el modo de 256 colores del VGA estándar. Su resolución está limitada a solo 320x200, que tiene doble escaneo para incrementar la altura a 400 líneas.

I.3 VGA

El VGA es una de las tarjetas de video más comunes (incluyendo sus mejoras: SVGA y UVGA) actualmente, cada equipo nuevo ya incluye este tipo de hardware, las tarjetas CGA y EGA ya son obsoletas para aplicaciones actuales e incluso ya es difícil conseguirlas en el mercado, es por esto que el VGA es y será una magnífica herramienta de trabajo en el ámbito computacional.

I.3.1 Arquitectura del VGA

Con la excepción de la salida de vídeo DAC (Digital to Analog Converters), la arquitectura del VGA en detalle es parecida a la del EGA. El VGA incluye unos pocos registros adicionales, faltándole el lápiz óptico soportado por el EGA. A diferencia de algunos de los registros del EGA, la mayoría de los registros VGA incluyen la capacidad de lectura-hacia atrás, la falta de la habilidad de lectura-hacia atrás fue un inconveniente en el original EGA que fue añadida después para la mayor parte de chip de manufactura EGA.

Empaquetamiento de Pixeles Vs. Planos de Colores

Dos técnicas comunes para almacenamiento de información de color son el método de empaquetamiento de pixeles y el método de plano de color. El EGA original está orientado al plano

de color, excepto por el modo CGA-compatible, los modos 4 hasta 5 usan empaquetamiento de pixeles. VGA tiene un modo añadido, el modo de empaquetamiento de pixeles de 256 colores.

Con el empaquetamiento de pixeles, toda la información de color para un pixel es empaquetado dentro de una palabra de dato de memoria. Para un sistema con pocos colores, este empaquetamiento de pixeles puede requerir solo una parte de un byte de memoria, para sistemas muy elaborados, un empaquetamiento de pixel puede estar en varios bytes de largos. Usando 8 bits por pixel, un empaquetamiento de pixeles se ve como se muestra en la figura I.9.

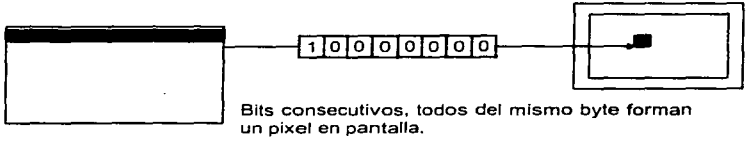


Figura I.9 Empaquetamiento de Pixeles.

Con el plano de color aproximado, la memoria de despliegue es separada dentro de varios planos de memoria independientes, con cada plano dedicado a controlar un componente de color (tal como rojo, verde azul). Cada pixel del despliegue ocupa una posición de bit en cada plano. Esta aproximación se muestra en la figura I.10.

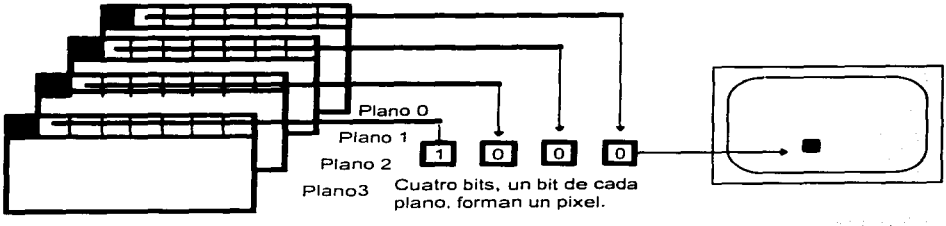


Figura I.10 Plano de color.

Modo Texto Vs. Modo Gráfico

Dos tipos básicos de modos de operación existen para el VGA: modo texto y modo gráfico. En el modo gráfico (tal como IBM refiere frecuentemente como modo de todos los puntos direccionables) un juego de bits en la memoria de despliegue representa un pixel sencillo en el despliegue en pantalla. En modo texto, sin embargo, un byte sencillo de código de caracter ASCII es colocado en la memoria de despliegue causa un caracter de texto completo para ser desplegado en pantalla. El modo texto requiere mucho menos memoria de despliegue y coloca menos carga en el sistema procesador pero es muy limitado en que solo texto y bloques de objetos gráficos toscos pueden ser desplegados. La figura I.11a ilustra la operación básica de un modo texto y la figura I.11b muestra la operación de un modo gráfico.

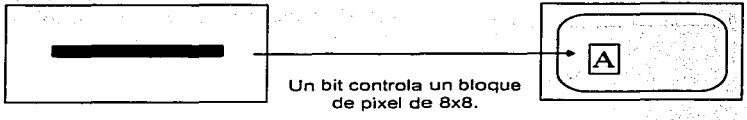


Figura I.11a
Modo de operación texto

Figura I.11b
Modo de operación gráfico

Figura I.11 Modo texto Vs. Modo gráfico.

La figura I.12 ilustra la arquitectura básica del VGA, la cual consiste principalmente de seis bloques funcionales:

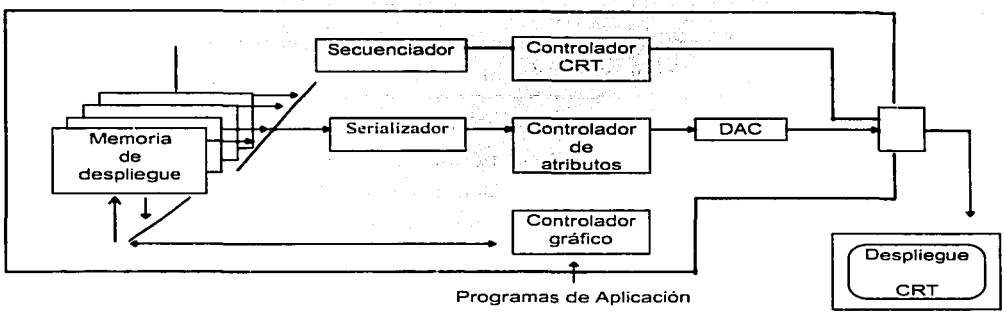


Figura I.12 Diagrama a bloques del VGA.

- Memoria de despliegue. Es un banco de 256 Kb. o más de memoria de acceso aleatorio dinámico (*DRAM o VRAM*), dividida en cuatro planos, los cuales contienen los datos de despliegue en pantalla.
- Controlador Gráfico. Reside en la dirección de datos entre el procesador y la memoria de despliegue. Puede ser programado para ejecutar funciones lógicas (tales como AND, OR, XOR o ROTATE) en datos siendo escritos en la memoria de despliegue. Estas funciones lógicas pueden proporcionar una hardware de ayuda para simplificar operaciones de dibujo.
- Controlador *CRT*. Genera señales de cronometraje para controlar la operación del despliegue CRT y desplegar cronometrajes de refresco.
- Serializador de datos. Captura la información de despliegue la cual es tomada de la memoria de despliegue uno o más bytes a la vez, y los convierte a una cadena de bits serial para ser enviados al despliegue CRT, Algunas tarjetas usan VRAM para datos seriales.
- Controlador de atributos. Contiene uno o dos tablas *lookup* de color (LUTs) que convierten la información de color de la memoria de despliegue a información de color para el despliegue CRT. La primera tabla lookup esta controlada vía registro de paleta del controlador de atributo, y la segunda tabla esta contenida en el video (DACs). Para programar una tabla lookup de color en el adaptador de despliegue, un programador puede seleccionar cual subjuego del despliegue de colores será soportado para su hardware.

- Video DACs. (Convertidos Digital-Analógico) convierte datos de color digital a una señal analógica. También contiene la segunda tabla lookup de color.
- Secuenciador. Controla el cronometraje de todas las funciones en la tarjeta. También contiene lógica para habilitar y deshabilitar lo planos de colores.

Memoria de Despliegue

El VGA contiene 256 Kb o más de despliegue de memoria, dividida en cuatro secciones independientes de memoria de 64 Kb (o 128 Kb) llamadas planos de color. todos estos planos de memoria residen en el mismo espacio de memoria del procesador. Cada plano de color esta siendo escrito o leído en algún tiempo determinado por la instalación de algunos registros I/O.

Con los cuatro planos de memoria residiendo en la misma dirección de espacio, el procesador puede escribir para los cuatro planos (o alguna combinación de ella) con un simple ciclo de escritura de memoria. Esta capacidad puede ser muy útil para algunas operaciones de dibujo, tales como el llenado rápido de pantalla. En otras operaciones de dibujo puede ser deseable para deshabilitar la escritura a todo pero a un plano de memoria simple. Los planos de colores son habilitados y deshabilitados por vía escritura al registro de habilitación de escritura del plano de color del secuenciador.

Después podría no ser significativo para el procesador el intento de leer datos de más de una fuente a la vez, solo un plano de memoria puede ser habilitado para lectura. Un plano de color es habilitado por vía del controlador gráfico. Un modo especial para leer datos de un plano de color múltiple es comparar un dato de referencia y regresar una información al procesador declarando si los colores hacen juego. La función de comparación de color es útil para encontrar ciertos patrones en la memoria de despliegue durante operaciones tales como el llenado de área. Este modo es controlado por el registro de comparación de colores del controlador gráfico.

En algunas operaciones de modos, la organización de la memoria de despliegue será alterada. El mejor ejemplo de esto esta en el modo texto, donde direcciones de memoria par (contienen datos *ASCII*) son un plano de memoria 0, direcciones de memoria impar (contienen atributos de texto) están en un plano de memoria 1, el plano de memoria 2 esta reservado para el generador de caracteres, y el plano de memoria 3 no es usado.

Para algunas operaciones de modo, el espacio de dirección de 64 Kb del EGA esta dividido en varias paginas de despliegue el software de aplicación tiene entonces el control de cual página

está activa (estando examinada) en un tiempo, y las operaciones de dibujo pueden tomar lugar fuera del despliegue en pantalla.

El procesador de dirección de espacio usado por el EGA y VGA depende en el modo de operación. Esta dirección de espacio puede iniciar en la dirección A000:0000, B000:0000 o B800:0000, dependiendo del modo.

Memoria de Despliegue en Modo Texto

El despliegue de modo texto ha sido en común uso mucho más largo que el despliegue gráfico, y son aún muy útiles en aplicaciones las cuales no requieren gráficos (o en las cuales un simple bloque gráfico será suficiente). El modo texto coloca mucha menor carga en el sistema procesador, el cual solo ha de manipular código de caracteres ASCII mejor que pixeles individuales.

En el modo texto estándar, el despliegue de pantalla está dividido en 25 líneas de texto, con 40 u 80 columnas de texto por línea. En el modo de 40 columnas, 1000 caracteres pueden ser desplegados en pantalla, en el modo de 80 columnas, 2000 caracteres pueden ser desplegados en pantalla (ver figura I.13). Dos bytes de memoria de despliegue son usados para definir cada caracter, el primer byte, mapeado en una dirección de memoria par, contiene el código de caracter ASCII, y el segundo byte, mapeado en una dirección de memoria impar, contienen la información de color llamada el *caracter de atributo*. Una página de memoria de despliegue es de 4096 bytes de largo, dejando 96 bytes sin usar al final de cada página.

Protegiendo el Modo de Despliegue Durante la Selección de Modo

La selección de funciones en modo BIOS opcionalmente protegerá el contenido de la memoria de despliegue si el número de modo deseado es ORED con el valor 80h antes que la llamada al BIOS sea hecha. Esta capacidad es limitada en el modo texto, sin embargo, hasta estos modos utilizan planos de memoria de despliegue para almacenar el generador de caracteres. Por lo tanto, no es posible introducir y sacar un modo texto sin corromper al menos parte de la memoria de despliegue.

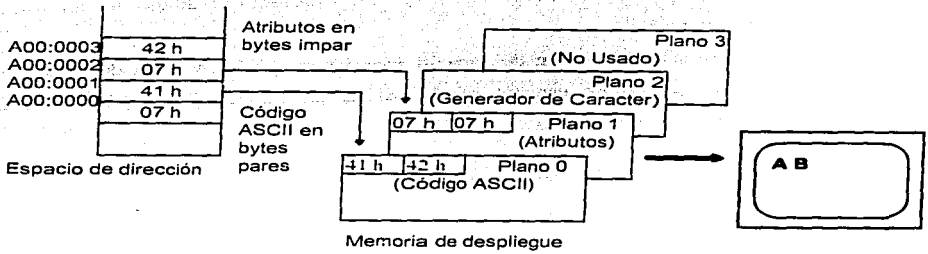


Figura I.13 Formato de memoria de despliegue en modo texto.

Generador de Caracteres

Para convertir un código de caracter ASCII en un arreglo de pixeles en pantalla, una tabla de conversión o *Generador de caracteres* es usada. En la más vieja tarjeta de video tal como el MDA o CGA, el generador de caracteres está localizado en la ROM. el VGA no es un generador de caracteres ROM, en cambio, datos de generador de caracteres están cargados en el plano 2 del despliegue RAM. Está característica lo hace fácil para juegos de caracteres acostumbrados al ser cargados. Juegos de caracteres múltiples (arriba de 8) pueden residir en la RAM simultáneamente. Un juego de servicios BIOS están disponibles para facilitar la carga de juegos de caracteres. La Figura I.14 ilustra como los códigos de caracteres son usados como un índice en un generador de caracteres.

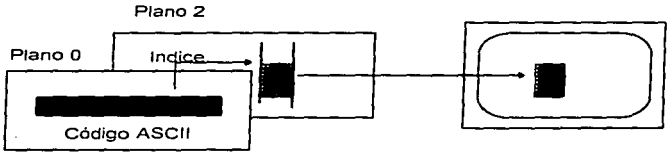


Figura I.14 Código de caracteres como índice en un generador de caracteres.

Cada uno o dos generador de pueden caracteres estar activos, dando al VGA la capacidad para desplegar arriba de 512 diferentes caracteres en la pantalla simultáneamente. Cuando dos generadores de caracteres están activos, un bit en cada byte de caracter de atributo selecciona cual juego de caracter será usado para ese caracter. Un registro en el secuenciador es usado para seleccionar

los dos generadores de caracter activos. El ancho de caracter esta fijo en 8 pixeles. La altura de caracter se selecciona entre 1 a 32 pixeles hasta un registro de salida. La figura I.15 ilustra como un generador de caracter es designado.

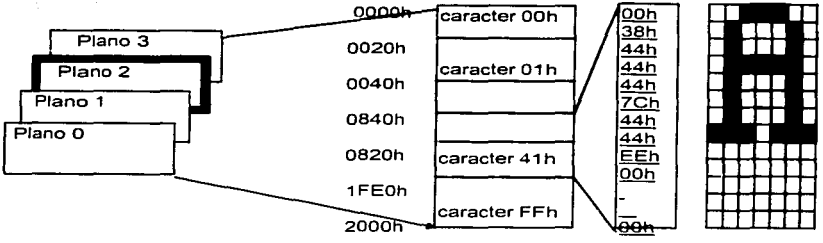


Figura I.15 Generador de caracteres.

La localización de generadores de caracter en memoria es mostrado en la Tabla I.5:

Tabla I.5 Localización de generadores de caracter residentes en RAM.	
Caracter mapa A	**Caracter mapa E**
0000h a 001Fh- caracter 0	2000h a 201Fh- caracter 0
0020h a 003Fh- caracter 1	2020h a 203Fh- caracter 1
0040h a 005Fh- caracter 2	2040h a 205Fh- caracter 2
...	...
1FE0h A 1FFFh- caracter 255	3FE0h A 3FFFh- caracter 255
Caracter mapa B	**Caracter mapa F**
4000h a 401Fh- caracter 0	6000h a 601Fh- caracter 0
5FE0h a 5FFFh- caracter 255	7FE0h a 7FFFh- caracter 255
Caracter mapa C	**Caracter mapa G**
8000h a 801Fh- caracter 0	A000h a A01Fh- caracter 0
9FE0h a 9FFFh- caracter 255	BFE0h a BFFFh- caracter 255
Caracter mapa D	**Caracter mapa E**
C000h a C01Fh- caracter 0	E000h a E01Fh- caracter 0
DFE0h a DFFFh- caracter 255	FFE0h a FFFFh- caracter 255

Atributos de Texto

Cada caracter ASCII desplegado en pantalla tiene su correspondiente byte de atributo para definir los colores y otros atributos que el caracter puede tener. La interpretación de los atributos de texto depende del modo de operación.

Atributos de Texto de Color Estándar

La figura I.16 muestra la definición de cada bit para un byte de atributo de texto cuando se opera en un modo de texto de color estándar. Los bits D0-D2, colores de primer plano, selecciona el color para el cuerpo del caracter. Bits D4-D6, color de fondo, selecciona el color para el resto de la celda del caracter.

El bit de atributo D3 puede ser usado como un control de intensidad de primer plano efectivamente, doblando el número de primer plano de 8 a 16 colores.

Bit7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Intensidad o parpadeo de fondo	color de fondo			Intensidad de color de texto	color de texto		

Figura I.16 Byte de atributo texto.

Si dos juegos de caracteres están siendo usados simultáneamente (definida por el registro selector generador de caracteres del secuenciador) el bit D3 selecciona cual juego de caracteres serán usados. En este caso, el registro de la paleta de colores del controlador de atributo podrá ser modificado para deshabilitar el bit D3 afectando el color.

El bit de atributo D7 puede ser usado de dos maneras como un habilitador de primer plano blink (parpadeo), provocando el caracter a blink o como un control de intensidad de fondo, doblando el número de primer plano de 8 colores a 16. La función del bit D7 está definida en el registro de modo del controlador de atributo. La instalación por default habilita el blink.

La Tabla I.6 muestra los colores estándar que son usados para el fondo y primer plano.

Tabla I.6 Atributos de color estándar.

Atributo	Color estándar	Color intensificado
000	Negro	Gris oscuro
001	Azul	Azul Luminoso
010	Verde	Verde Luminoso
011	Cyan	Cyan Luminoso
100	Rojo	Rojo Luminoso
101	Magenta	Magenta Luminoso
110	Café	Amarillo
111	Gris Luminoso	Blanco

Atributos de Texto Monocromático

La tabla I.7 muestra el bit de definición para un byte de atributo de texto monocromático, el cual es similar en función al byte de atributo de texto de color. Los bits D0-D2 controlan los atributos

de primer plano, los cuales pueden ser normal. blanqueado o subrayado. El bit D3 podrá intensificar el color de primer plano. Los bis D4-D6 pueden seleccionar un caracter de reversa de video. El bit D7 puede ser usado de ambas maneras, para habilitar el blink o control de intensidad del primer plano, está función está controlada en el registro de Control de Modo del controlador de atributos. La instalación por default habilita el parpadeo del primer plano.

Tabla I.7 Atributos de texto monocromático.	
00000000	Blanco
00000111	Caracter Normal
10000111	Caracter Parpadeo
00001111	Caracter Intensificado
10001111	Caracter Intensificado Parpadeo
00000001	Caracter Subrayado
10000001	Caracter Subrayado Parpadeo
00001001	Caracter Subrayado Intensificado
10001001	Caracter Subrayado Intensificado Parpadeo
01110000	Video Reversa
11110000	Video Reversa Parpadeo

El bit D3 puede ser usado para seleccionar entre dos juegos de caracteres activos.

Como se observa en la tabla I.7, hay sólo un pequeño número de atributos de texto válidos en el modo monocromático. Todos los valores de atributos no mostrados en la tabla anterior pueden ser considerados como inválidos. El uso de atributos inválidos puede crear problemas de compatibilidad cuando el software es ejecutado en diferentes tipos de tarjeta de video (MDA, EGA, VGA y Hércules). Si un caracter es video reversa no puede ser subrayado o intensificado.

Memoria de Despliegue en Modos Gráficos

- Modo 6 (Gráficos CGA en dos colores). En 640x200, el modo 6 es el más alto modo de resolución del adaptador CGA. Usa solo un bit por pixel (8 pixeles por byte). Un pixel con valor 0 despliega negro y un pixel con valor 1 despliega un blanco. El dato del pixel es almacenado en un plano de color 0. El despliegue de datos es serial el bit más significativo primero, de éste modo la primera posición de bit en la esquina superior izquierda de la pantalla despliega el dato en el bit D7 del byte 0 de la memoria de despliegue.

  Las limitaciones del controlador CRT 6845 el cual fue usado en el CGA resultó en un mapeado no lineal de la dirección de espacio de la memoria de despliegue para todos los modos gráficos. En éste complicado algoritmo de dibujo, después es requerido un cálculo para transformar

entre una posición de pixel en el despliegue de pantalla y una posición de bit en la memoria de despliegue.

La figura I.17 ilustra la forma de la transformación que ocurre entre la memoria de despliegue y el despliegue de pantalla. La primera mitad de memoria de despliegue contiene los datos para todos las líneas escaneadas numeradas par. La segunda mitad de la memoria de despliegue contiene los datos para todas las líneas escaneadas numeradas impar. Para realizar la transformación de una posición de pixel (x, y) en el despliegue de pantalla, donde "x" es la coordenada horizontal en el rango 0-639 y "y" es la coordenada vertical en el rango 0-199 para una posición de bit en la memoria de despliegue, use la siguiente fórmula:

Dirección de byte = 80*(y/2) + (x+8) si "y" es par
Dirección de byte = 8192 +80*((y-1)/2)+ (x/8) si "y" es impar
Posición de bit (0-7) = 7 - (x modulo 8)

El operador modulo es equivalente a tomar el sobrante de x/8.

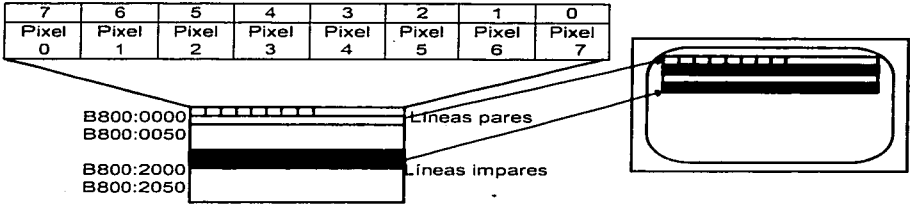


Figura I.17 Mapeo de memoria. Modo Gráfico 6.

- Modos 4 y 5 (Gráficos CGA en 4 colores). Estos son los más coloridos, también los más populares en la tarjeta CGA. La resolución es baja (320 x 200). El mapa de memoria de despliegue usa paquetes de pixeles, 2 bits por pixel, empaquetados 4 pixeles por byte. El dato de pixel es almacenado en un plano (plano 0 y 1 encadenados para formar un plano). El despliegue de datos es serial el bit más significativo primero, de éste modo la primera posición de bit en la esquina superior izquierda de la pantalla despliega el dato en los bits D7 y D6 del byte 0 de la memoria de despliegue.

Ya que con todos los modos gráficos CGA, la memoria de despliegue es mapeada no linealmente es requerido un cálculo para transformar de una localización de pixel en el despliegue

de pantalla a una localización de bit en memoria de despliegue. La figura I.18 muestra el mapeo de memoria para los modos 4 y 5. La primera mitad de memoria de despliegue contiene los datos para todos las líneas escaneadas numeradas par. La segunda mitad de la memoria de despliegue contiene los datos para todas las líneas escaneadas numeradas impar.

Para transformar de una posición de pixel (x,y) en el despliegue de pantalla para una posición de bit en la memoria de despliegue, donde "x" es la coordenada horizontal en el rango 0-319 y "y" es la coordenada vertical en el rango 0-199, use la siguiente fórmula:

Dirección de byte = 80*(y/2) + (x+4) si "y " es par
Dirección de byte = 8192 +80*((y-1)/2)+ (x/4) si "y" es impar
Posición de bit (0,2,4,6) = (x modulo 4) * 2

Dos juegos de colores estándar son soportados en los modos 4 y 5. Una llamada al BIOS (función BIOS 0Bh) es usada para seleccionar los colores. El estándar de colores para los modos 4 y 5 son mostrados en la tabla I.8:

Tabla I.8 Atributos de color estándar. Modo 4 y 5.

Valor del pixel	Color Estándar	Color Alternado
00	Negro	Negro
01	Cyan Luminoso	Verde
10	Magenta Luminoso	Rojo
11	Blanco Intenso	Café

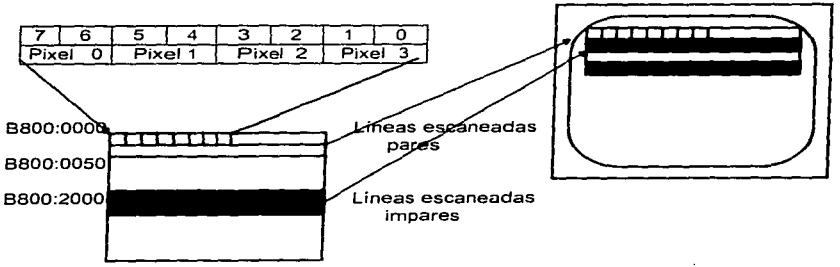


Figura I.18 Mapeo de memoria-Modos Gráficos 4 y 5.

- Modo F-Gráficos Monocromáticos. El modo F, el cuál es único para el EGA y VGA, no sufre de problemas de direccionamiento no lineal de los modos gráficos CGA. Su resolución es de 640 x 350. Son usados dos planos de color (plano 1 y 0). Cada pixel ocupa un bit en cada plano color. los cuatro colores soportados por éstos pixeles "2-bits" son negro, blanco, blanco intensificado y parpadeo. Los dos planos de colores son habilitados y deshabilitados independientemente por escritura desde el registro de habilitación de escritura del plano de color del secuenciador.

  La organización de la memoria se muestra en la figura I.19, excepto que solo los planos 0 y 1 son usados. Para transformar de un pixel (x,y) en pantalla a la localización de un bit en la memoria de despliegue, donde "x" es la coordenada horizontal en el rango 0-639 y "y" es la coordenada vertical en el rango 0-349, use la siguiente fórmula:

  Dirección de byte = y *80 + x/8
  Posición de bit (0,7) = 7 - (x modulo 8)

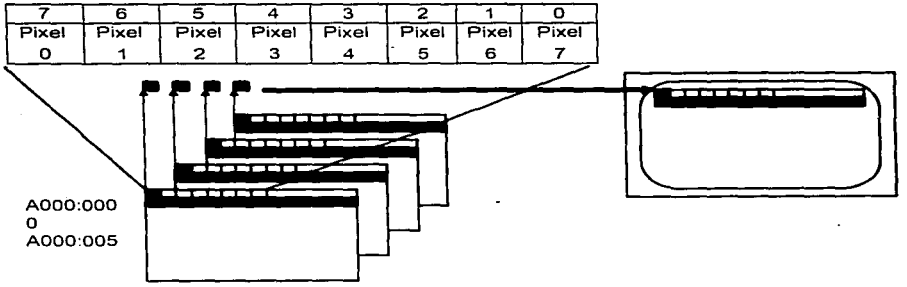


Figura I.19 Mapeo de memoria-Modo Planar (Dh, Eh, Fh, 10h, 11h, 12h).

- Modo 10. Gráficos a color mejorados. El modo 10, el cual es único para EGA y VGA, es el modo más popular para nuevas aplicaciones gráficas a color. su resolución es 640 x 350. Los cuatro planos de color son usados Los planos de color son habilitados y deshabilitados independientemente por escritura desde el registro de habilitación de escritura del plano de color del secuenciador. Cada pixel ocupa un bit en cada plano de color. Estos pixeles 4-bits permiten 16 colores simultáneos para ser desplegados. La figura I.19 muestra el mapa planar de memoria para el modo 10 h. Para transformar de un pixel (x,y) en pantalla a la localización de un bit en la memoria de

despliegue, donde "x" es la coordenada horizontal en el rango 0-639 y "y" es la coordenada vertical en el rango 0-349, use la siguiente fórmula:

Dirección de byte = y *80 + x/8
Posición de bit (0,7) = 7 - (x modulo 8)

- Modo D y E (Gráficos de 16 colores). Los modos D y E son muy similares en operación al modo 10, difieren solamente en la resolución de la pantalla. El modo D opera en una resolución de 320 x 200. El modo E opera en una resolución de 640 x 200. Estos dos modos no han sido muy populares por la resolución limitada que ofrecen.

- Modo 11 (Gráficos de 2 colores). EL modo 11 es único para el adaptador VGA. Su resolución es de 640 x 480, pero solo soporta 2 colores. El despliegue de datos es almacenado en el plano 0, y los otros planos no son usados. Cada pixel ocupa un bit en la memoria de despliegue.

  La memoria de despliegue es similar al de la figura I.19, excepto que solo un plano es usado. Para transformar de un pixel (x, y) en pantalla a la localización de un bit en la memoria de despliegue, donde "x" es la coordenada horizontal en el rango 0-639 y "y" es la coordenada vertical en el rango 0-479, use la siguiente fórmula:

  Dirección de byte = (y *80) + (x/8)
  Posición de bit (0,7) = 7 - (x modulo 8)

- Modo 12 (Gráficos de 16 colores). El modo 12 es único para el VGA, es similar al modo 10h excepto que la resolución vertical es expandida de 350 a 480 líneas. Los 4 planos de colores son usados, y 16 colores simultáneos son soportados. La organización de la memoria es la misma que la mostrada en la figura I.19.

- Modo 13 (Gráficos de 356 colores). El modo 13, el cuál también es único para el VGA, permite 256 colores simultáneos para ser usados en baja resolución (320x200). La memoria es linealmente mapeada como se muestra en la figura I.20. Para transformar de un pixel (x,y) en pantalla a la localización de un bit en la memoria de despliegue, donde "x" es la coordenada horizontal en el rango 0-319 y "y" es la coordenada vertical en el rango 0-199, use la siguiente fórmula:

  Dirección de byte = (y *320) + x

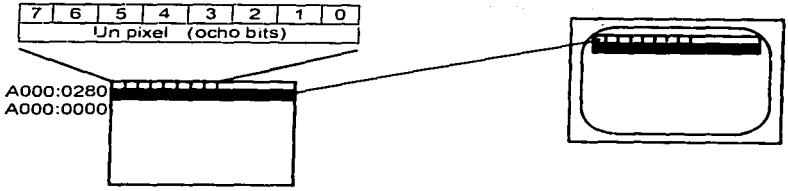


Figura I.20 Mapeo de memoria. Modo Gráficos 13.

Controlador Gráfico

El controlador gráfico reside en la dirección de datos entre el procesador y la memoria de despliegue. En su estado de default, el controlador gráfico es transparente. Pueden ser escritos y leídos datos desde la memoria de despliegue sin alteraciones. El controlador gráfico puede, sin embargo, ser programado para asistir en operaciones de dibujo por ejecución de tareas que podrían de otra manera han de ser ejecutadas por el procesador principal.

Código Procesador de Lectura

Cada vez que el sistema procesador lee datos desde la memoria de despliegue, el dato también es codificado por el código de lectura de la tarjeta VGA. Durante los ciclos de escritura, el dato en este código de lectura puede ser lógicamente combinado con datos de escritura desde el procesador. Si es usado correctamente, esta función puede asistir al procesador en la ejecución de operaciones de dibujo. Mientras el procesador puede solo leer datos desde un plano a la vez, el código de lectura codifica datos desde los cuatro planos simultáneamente. Esto puede ser usado para agilizar la copia de datos desde una región del plano de memoria a otra.

Unidad Lógica

Durante los ciclos de escritura de la memoria de despliegue, el controlador gráfico puede ejecutar alguna de las siguientes funciones en la escritura de datos:

- Escritura de datos sin modificar.
- Lógica *OR* en escritura de datos con datos en el código de lectura.
- Lógica *AND* en escritura de datos con datos en el código de lectura
- Lógica *XOR* en escritura de datos con datos en el código de lectura

- Rotación de escritura de datos.

Las funciones lógicas AND, OR, XOR son útiles para añadir y remover elementos de despliegue del primer plano sobre el fondo (como el *cursor* gráfico y sprites). La rotación de datos es útil cuando se ejecutan bloques de transferencia de un dato byte-no-alineado.

La función del controlador gráfico durante la operación de escritura esta ilustrada en la figura I.21.

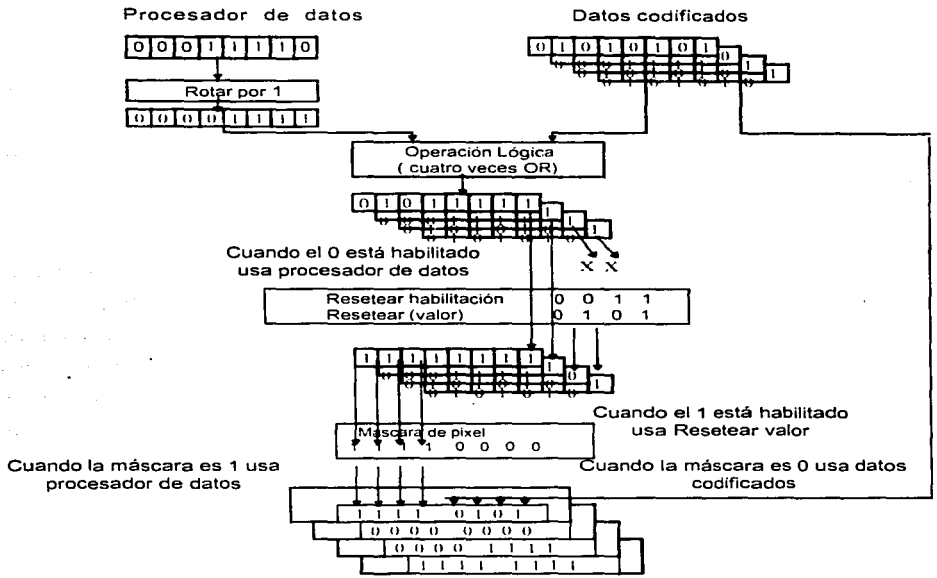


Figura I.21 Operación de escritura del Controlador Gráfico.

Comparación de Color

Durante los ciclos de procesos de lectura, el controlador gráfico puede ejecutar una función llamada Comparación de Color, la cuál es útil para algoritmos de dibujo tales como flood fill (algo así como llenado por inundación) donde una pantalla de color específico o cambio en color debe ser

detectada. Usando ciclos de lectura de memoria de despliegue normal, el procesador puede solo preguntar un plano de color a la vez. Con la comparación de color, sin embargo, el procesador introduce una referencia de color dentro de un registro del controlador gráfico. Durante un ciclo de lectura, el controlador gráfico compara el dato en los cuatro planos (o algún subjuego seleccionado de los cuatro planos) junto al color de referencia e indica si un color igual fue encontrado.

La comparación de color provee la habilidad para buscar memoria de despliegue para un objeto específico de un color específico.

Serializador de Datos

El serializador de datos captura el dato de lectura desde la memoria de despliegue durante el despliegue de ciclos de refresco y los convierte en una cadena de bits serial al drive CRT de despliegue. El despliegue de datos es serial el bit más significativo primero. Algunas tarjetas usan VRAM para su memoria de despliegue, y en tales casos VRAM es usada para serializar datos.

Controlador de Atributos y DACs

El controlador de atributos y registros DACs determina que colores podrán ser desplegados para ambos modos, texto y gráfico. El fondo de un controlador de atributos es una tabla lookup de color *(LUT)* que en modos planares transforma códigos de 4 bits de color de la memoria de despliegue en códigos de 6 bits de color. Estos códigos de colores son combinados con un valor de registro de selección de color para formar códigos de 8 bits que alimentan al DAC de video. Una segunda tabla lookup de color, interna en el DAC, convierte este código de 8 bits en un código de 16 colores de la memoria de despliegue directamente al video DAC, sin transformación.

Como parte de una operación de selección del modo BIOS, las tablas de color lookup son inicializadas con datos apropiados para ese modo. Para modos monocromáticos, las tablas son inicializadas al despliegue de 2 colores. Para modos CGA, las tablas son inicializadas para soportar el límite de colores disponibles con ése adaptador. Para modos EGA y VGA, las tablas son inicializadas para soportar los más ricos colores de esos adaptadores. Aplicaciones de software pueden algunas veces redefinir la paleta de colores por reprogramación de controlador de atributos y/o el registro DAC.

La figura I.22 ilustra la función de la tabla lookup de color durante un ciclo de refresco de pantalla por un modo típico planar. En el diagrama un valor de color de pixel de 0101 (en decimal 5) ha sido leído de los planos de color. Este valor de color es usado como una dirección para

seleccionar un registro en la tabla lookup de color. El registro 5 en la tabla lookup contiene el dato binario 000101, lo cual resulta en un pixel magenta en pantalla (asumiendo por default los valores del registro DAC).

Note que el color (atributo) es representado en forma diferente en modo texto que en modo gráfico.

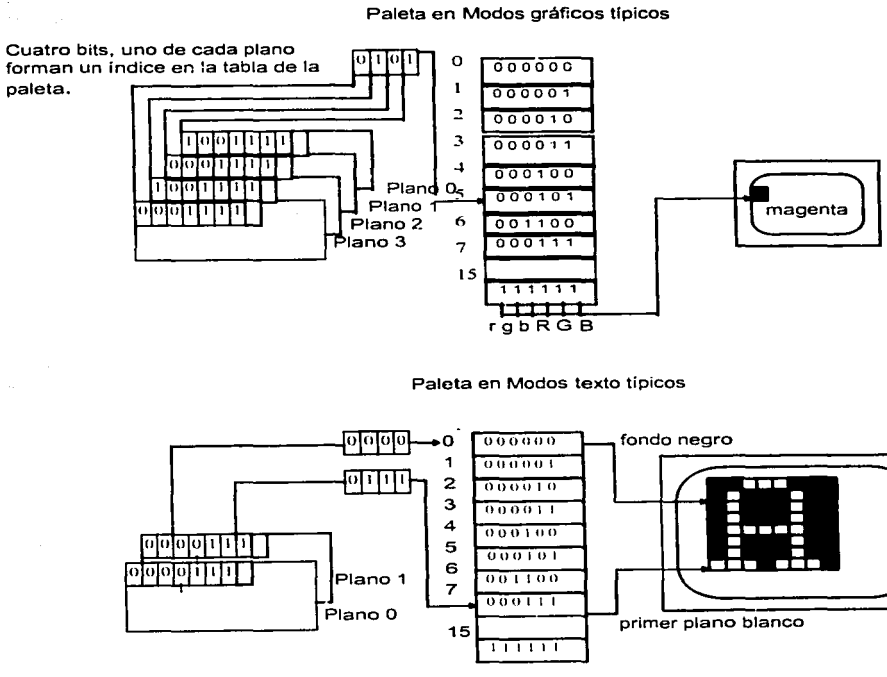


Figura I.22 Tabla look up.

Controlador CRT

La mayoría de los registros del controlador CRT son puestos por el BIOS para definir cronometrajes y no deben ser modificados. Otros registros de controlador CRT definen formas y posiciones de cursor y ejecutan *scrolling* vertical.

El Secuenciador

El secuenciador genera el punto y caracter de reloj que controla el cronometraje de despliegue de refresco. Controla el cronometraje de ciclos de memoria de despliegue de lectura y escritura, y genera estados de espera para el procesador cuando son necesarios.

El secuenciador también contiene lógica para habilitar y deshabilitar acceso al procesador para planos de color específicos. Es ésta función la que hace que el secuenciador sea interesante para lo programadores. Figura I.23.

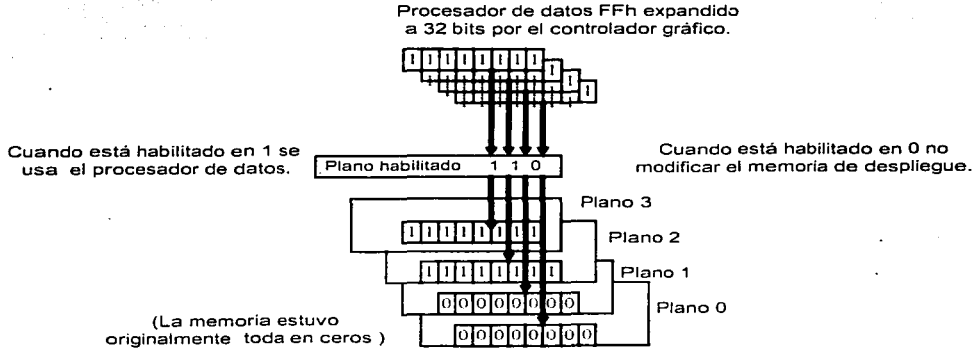


Figura I.23 Función de habilitación de escritura de plano del Secuenciador.

I.4 SVGA

I.4.1 Arquitectura del SVGA

La arquitectura básica de un SVGA incluye 5 bloques funcionales principales: controlador CRT, secuenciador, controlador de atributos, controlador gráfico y memoria de despliegue.

La principal característica significativa adicional de todos los SVGA es la capacidad de desplegar altas resoluciones y mas colores que el estándar VGA.

Una llave necesaria para esta capacidad extendida es la habilidad de direccionar mayor cantidad de despliegue de memoria que el VGA puede alojar, lo cual es usualmente realizado con algún tipo de mecanismo de paginamiento de memoria. Mientras esquemas de paginamiento de memoria varían entre manufacturas, el principio básico permanece igual.

Los SVGA usualmente incluyen otros soportes de software en la forma de upgrade de BIOS y drivers de aplicaciones de software para soportar modos de despliegue extendidos.

Mapeo de la Memoria de Despliegue

Espacio de Dirección Host / Host Window

Las operaciones de dibujo VGA son ejecutadas por el sistema procesador, el cual lee datos de la base de datos escrita en la memoria de despliegue. Para cumplir esto, la memoria de despliegue es mapeada a un segmento específico (o segmentos) del espacio de dirección de memoria del host procesador. Esto es referido a veces como el host Window para la memoria de despliegue.

La tabla I.9 muestra la organización para el primer megabyte del sistema de memoria direccionable en una computadora compatible IBM.

Tabla I.9 Mapeo de memoria de la PC.

Dirección	Contenido
F000:FFFF	
F000:0000	ROM BIOS
E000:0000	LAN, Cinta de respaldo, EMS,...
CC00:0000	Otras tarjetas añadidas de BIOS
C800:0000	BIOS de disco XT
C000:0000	Memoria de despliegue CGA/VGA/EGA (segunda página de RAM Hércules)
B800:0000	Memoria de despliegue MDA/VGA/EGA (primera página de RAM Hércules)
B000:0000	VGA/EGA Memoria. de despliegue Programa transiente de área (usuario de memoria). Parte residente del command.com. Buffers de disco, drivers instalables,... Kernel del DOS.
0000:0400	Área de datos BIOS.
0000:0000	Vectores de interrupción.

El host Window usado por el VGA varia dependiendo del modo de operación. La tabla I.10 contiene el estándar host Window y muestras de modos usando cada ventana.

Tabla I.10 Host Window VGA.	
Dirección Host	**Contenido**
C000:0000h - C000:5FFFh	ROM BIOS VGA IBM
B800:0000h - B800:7FFFh	Texto a Color (Modo 3)
B000:0000h - B000:7FFFh	Texto Monocromático (Modo 7)
A000:0000h - B000:FFFFh	Gráficos VGA (modos D,E,F, 10,..)
A000:0000h - B000:FFFFh	Gráficos mejorados (A000:FFFF para mayores modos)

Para modos texto, quienes requieren relativamente pocos datos para ser movidos, es usado un espacio de 32K. En modos gráficos, donde mayor cantidad de datos es requerida, es usado un espacio de 64K. Cuando los cuatro planos de color VGA son usados, esto da al procesador acceso a 256K de memoria de despliegue (4 x 64k).

Si la resolución de pantalla y numero de colores aumenta, la cantidad de memoria de despliegue que debe ser accesada por el procesador también aumenta. En algunos modos de alta resolución, más de 256 Kb de memoria deben ser accesados.

Una manera simple de ganar acceso a más memoria de despliegue es incrementar el tamaño de memoria host usado por el VGA de 64 Kb a 128 Kb, usando el espacio de dirección de memoria de A000:0 a B000:FFFF. Esta opción estándar VGA, la cual es seleccionada vía Miscelánea de registro del Controlador Gráfico, esto es conveniente y eficiente pero interfiere con otros adaptadores de despliegue co-residentes tal como el MDA, CGA o Hércules. Un modo de despliegue estándar IBM no usa éste host Window de 128 Kb. Un modo alternativo para accesar mas de 64 Kb es usar un esquema de paginamiento de memoria.

Planos de Memoria Vs. Páginas de Memoria

Los modos EGA y VGA incluyen 256 Kb. de memoria de despliegue. Para permitir al procesador acceso de todo la memoria de despliegue a través de un host Window de 64 Kb, la memoria de despliegue esta dividida en cuatro planos de memoria (4 planos x 64 Kb por plano = 256 Kb). Los planos de memoria son ilustrados en la figura I.24:

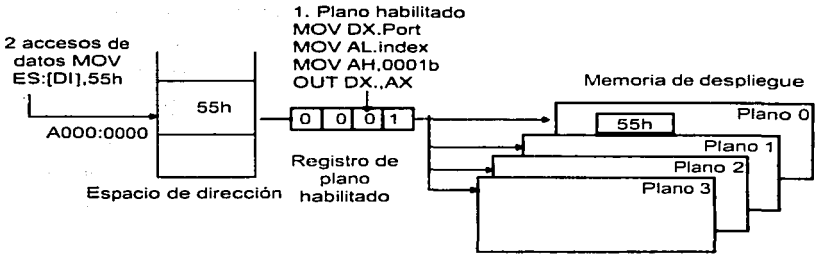


Figura I.24 Selección de plano.

Los SVGA pueden ser configurados con 256 Kb, 512 Kb o 1024 Kb de memoria de despliegue. Para permitir al procesador acceso a esta cantidad de memoria de despliegue a través de 64 Kb del host window, el SVGA debe incluir un mecanismo de paginamiento de memoria añadido para permitir a una sección de memoria de despliegue ser mapeada al procesador.

Paginamiento de la Memoria de Despliegue

El paginamiento de memoria de despliegue opera de una manera similar al sistema de paginamiento usado con las tarjetas de memoria expandida (también llamadas EMS o memoria LIM/EMS después de las especificaciones de memoria expandida de Lotus Intel Microsoft). Antes de transferir datos a o desde la memoria de despliegue, un programa de aplicación debe primero seleccionar la página de memoria propia para cargar el número de página en el registro de selección de página. Este proceso es ilustrado en la figura I.25:

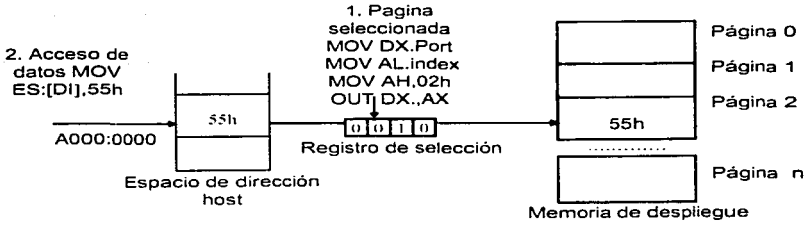


Figura I.25 Selección de página.

El mecanismo de paginamiento de memoria de despliegue varía entre las manufacturas del SVGA. El tamaño de páginas de memoria de despliegue varía entre diferentes productos VGA, o entre modos del mismo producto, 32 y 64 Kb. son tamaños de página comunes. La granularidad de paginas que inicia direcciones (el mínimo incremento con el cual inicia la dirección de memoria para una página puede ser especificado), también varía y puede ser más pequeña que el tamaño de página actual. Un tamaño de página grande con una granularidad pequeña es deseable. La figura I.26 muestra los efectos de granularidad y tamaño de página. Una granularidad fina requiere mas bits en el registro de paginamiento, como se muestra en la tabla I.11

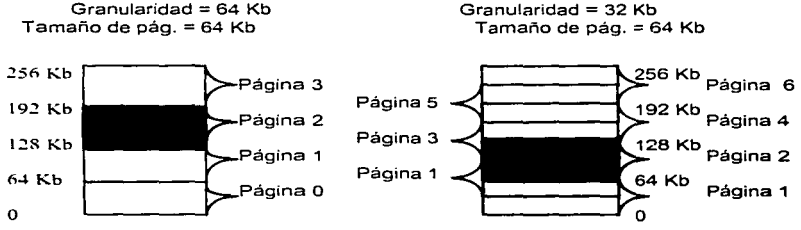


Figura I.26 Granularidad y tamaño de página.

Tabla I.11 Granularidad Vs. bits necesarios para registro selector de página.		
	Bits necesarios	
Tamaño Granulado	RAM 512 Kb	RAM 1024 Kb
1 Kb	9 bits	10 bits
4 Kb	7 bits	8 bits
32 Kb	4 bits	5 bits
64 Kb	3 bits	4 bits

El esquema de paginamiento de memoria de despliegue cae dentro de las tres categorías acordadas para el número de páginas simultáneas soportadas, y los tipos de acceso soportados (lectura, escritura y ambos) para cada página.

Una Página de Memoria de Despliegue

En la más simple implementación de paginamiento de memoria de despliegue, solo una página de memoria puede ser seleccionada a la vez. Las funciones que requieren datos para ser

transferidos de una área de memoria de despliegue a otra, tal como las operaciones *BITBLT*, pueden dificultar la ejecución usando éste esquema, si deben ser transferidos datos entre páginas. Tal transferencia consiste en cuatro pasos: seleccionar la página fuente, leer los datos, seleccionar la página destino, escribir los datos.

Para minimizar la cantidad de páginas a transferir requeridas en tales casos y para permitir el uso de bloques mover (REP MOVSB), los datos deben ser transferidos usando un buffer temporal en el huésped de memoria. Una operación BITBLT usa está aproximación como se ilustra en la figura I.27.

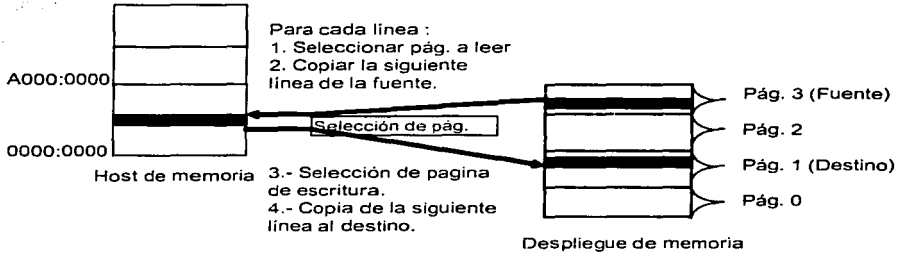


Figura I.27 BITBLT con una página.

Dos Páginas de Memoria Simultáneas, Una de Solo Lectura y Otra de Solo Escritura

Una segunda aproximación para el paginamiento de memoria de despliegue permite dos páginas separadas de memoria de despliegue para ser seleccionadas simultáneamente, siendo una página de solo lectura y la otra de solo escritura. Ambas páginas están mapeadas en el mismo dirección del huésped de memoria. Esto se ilustra en la figura I.28.

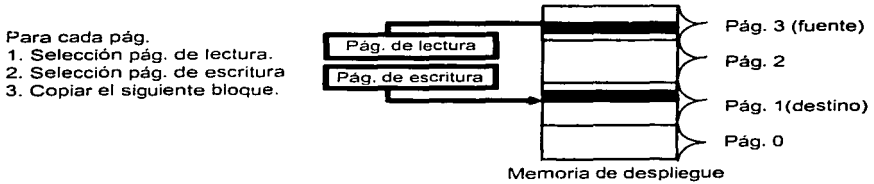


Figura I.28 BITBLT con páginas de escritura y lectura separadas.

Esta implementación permite transferir datos rápido de una página de memoria de despliegue a otra, hasta la instrucción mover bloque (REP MOVSB) puede ser usada para mover datos directamente de una página a otra. Esta aproximación tiene sus limitaciones, sin embargo, no ofrece ventajas para operaciones BITBLT que ejecutan operaciones lógicas (AND, OR, XOR, etc.) entre fuentes de datos y destinos de datos, si la página destino es de solo escritura. Un buffer intermedio en el host de memoria es requerido.

Dos Páginas de Memoria Totalmente Independientes.

Una aproximación más flexible para paginamiento de memoria es la que permite dos páginas de memoria totalmente independientes para ser seleccionadas simultáneamente en diferentes direcciones de memoria. Usando este esquema las operaciones BITBLT con las funciones lógicas son ejecutadas con un mínimo de intercambio entre páginas, y sin un buffer intermedio en el host de memoria. Esta aproximación también tiene sus desventajas, sin embargo, hasta las dos páginas pueden residir en direcciones de huésped de memoria diferentes, el host Window debe ser dos veces su largo, o el tamaño de página debe ser cortada a la mitad. Mejorar el host window de 64 Kb a 128 Kb causa conflictos con adaptadores de despliegue secundarios. Reducir el tamaño de página abajo de 64 Kb complica el algoritmo que debe detectar los limites de página.

Programación de Gráficos con Paginación de Memoria de Despliegue

El acceso de memoria de despliegue a través de un mecanismo de paginamiento de memoria causa una inevitable degradación en la velocidad de dibujo. Esta degradación puede tomar rangos de no legible hasta severo, dependiendo de como están escritas las rutinas de dibujo. Así como una rutina de dibujo avanza a través de la memoria de despliegue, es importante:

1. Minimizar la frecuencia con la cual se debe checa el limite de página.
2. Minimizar el número de instrucciones usadas en chequeos de limites de página.
3. Ejecutar selección de página solo cuando sea requerido.

Para algunos algoritmos de dibujo es posible calcular donde el limite de página será cruzada, entonces divide la operación de dibujo en dos o más pasos, (un paso para cada página). El límite de página checa entonces si no son requeridas durante el ciclo interno repetitivo de la función de dibujo.

Detección del Límite de Página

Por eficiencia los algoritmos de dibujo que se mueven a través de un rango de coordenadas "x" y "y" usualmente no trasladaran repetidamente coordenadas "x,y" dentro de las direcciones de memoria de despliegue. Esta traslación es solo ejecutada una vez al inicio de la rutina de dibujo, más tarde el algoritmo de dibujo avanza en la dirección "x" o "y" simplemente por incremento o decremento de direcciones de memoria de despliegue por un valor constante. Tales algoritmos son conocidos como DDA*s* (Digital Differential Analyzers).

Modos Mejorados

Modos de despliegue mejorados con altas resoluciones y más colores son las más importantes características del SVGA. Altas resoluciones en modo texto que permiten columnas, así como alta resolución en modos gráficos con capacidad de 256 colores simultáneos. No todas las tarjetas SVGA las mismas resoluciones de despliegue mejorados, pro ciertas resoluciones se han convertido en estándares. Esto ha sido determinado principalmente por las capacidad de despliegue que están disponibles. Resoluciones populares incluyen 640x480, 800x600, y 1024x768 pixeles. Hasta estos modos han sido desarrollados como extensiones básicas para el VGA, hay usualmente un alto grado de similitud en la manera en que están implementados.

Modos Texto Mejorados

Por variación en el despliegue y tamaño de una celda de caracter, algunos modos texto pueden ser soportados.

Los modos que despliegan una pantalla mas ancha (más caracteres por línea) son útiles para aplicaciones tales como hojas de calculo donde algunas columnas de datos deben ser desplegadas, las 132 columnas de texto son populares desde que representaron un ancho estándar para salidas de impresión de las computadoras, por cada una de las más altas resoluciones de caracteres son pequeños y dificultan para leer en este formato, los formatos de 100 columnas y 120 columnas son también populares.

Los modos texto de 132 columnas usualmente requieren más de 1000 pixeles de resolución horizontal en el despliegue. Mientras eso excede las especificaciones publicadas para virtualmente todas las clases de despliegue VGA, los caracteres permaneces legibles en la mayoría de los más populares despliegues VGA, sin embargo con menor calidad.

La organización de la memoria de despliegue para texto mejorados es la misma que para los modos estándar VGA. Las características mejoradas incluyen más altas resoluciones y bits de atributos adicionales para selección de fuentes (figura I.29).

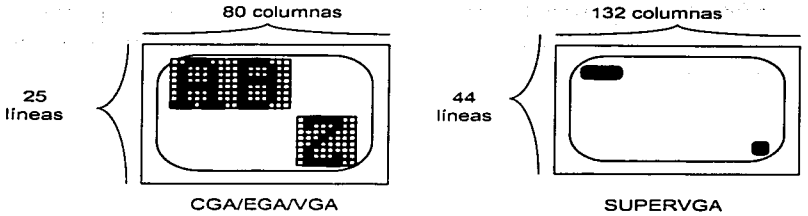


Figura I.29 Modo texto mejorado.

Modos Gráficos Mejorados

Los modos que ofrecen 256 colores simultáneos en resoluciones más grandes que 320x200 pixeles pueden ser usados para presentar imágenes fotográficas a todo color con una impresionante fidelidad. Los modos que ofrecen 16 colores en resoluciones mayores al VGA son populares para aplicaciones que envuelven detalles visuales finos, tales como CAD/CAM y publicaciones de escritorio. Los modos que ofrecen altas resoluciones con solo 2 o 4 colores son populares para despliegues WYSIWYG (what You See Is What You Get, lo que ves es lo que obtienes) en publicaciones de escritorio donde la resolución es lo importante pero los colores usualmente no lo son. La figura I.30 ilustra las resoluciones de modos gráficos mejorados:

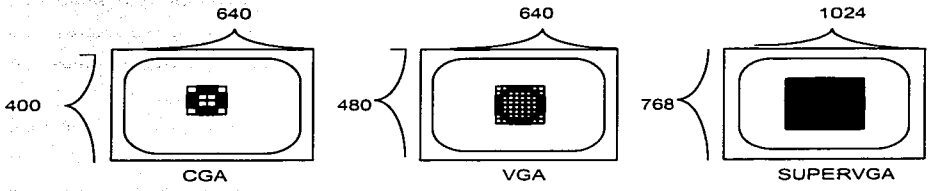


Figura I.30 Modos Gráficos Mejorados.

Gráficos a 256 Colores 640 x 400

Es una resolución lógica para algunos adaptadores porque requiere 256 bytes de memoria de despliegue, la cual es la cantidad de memoria de despliegue incluida en los productos VGA (262,144 bytes). La resolución es también un múltiplo exacto del modo estándar VGA 13h (320x200 a 256 colores). Este es un modo común para los modos gráficos que no requieren pixeles cuadrados (un aspecto de radio es requerido para alcanzar pixeles cuadrados en la industria de despliegue estándar).

Gráficos a 256 Colores 640 x 480

Es una resolución igual a la más alta resolución VGA, con capacidad para 256 colores, lo hace un modo lógico para el SVGA. Este es soportado solo por VGAs que incluyen al menos 512 Kb de memoria de despliegue.

Alguna forma de paginamiento de memoria es requerido para hacer más grande el memoria de despliegue disponible para la organización del procesador de la memoria de despliegue.

Gráficos a 256 Colores 800 x 600

Es la más alta resolución que está disponible al más bajo costo de los despliegue de multifrecuencia. También es la más alta resolución disponible en adaptadores con 512 Kb de memoria de despliegue. Imágenes fotográficas a todo color pueden ser desplegadas con fidelidad remarcable en esta resolución. Este modo requiere de 480 Kb de memoria de despliegue.

Gráficos a 256 Colores 1024 x 768

Es la más alta resolución encontrada en tarjetas SVGA sin embargo algunas manufacturas de chips suben la capacidad de resolución arriba de 1280x1024, o 16 bits por pixel (65536 colores), como de esta escritura no hay adaptadores SVGA disponibles con capacidades abajo de 1024x768 con 256 colores. Este modo requiere 768 Kb de memoria de despliegue.

Gráficos a 16 Colores 800 x 600

Este modo requiere 240 Kb de memoria de despliegue y es la más alta resolución a 16 colores que puede ser soportada usando solo 256 Kb de memoria de despliegue. También es la más alta resolución a 16 colores que puede ser soportada sin utilizar alguna forma de paginamiento de memoria para permitir al procesador accesar a la memoria de despliegue completo. Es el más alto límite de resolución en los originales despliegues de multifrecuencia.

Gráficos a 16 Colores 1024 x 768

Es la más alta resolución que es comúnmente encontrada en los productos VGA. solo el mejor despliegue VGA puede soportar esta resolución.

En orden para el adaptador VGA para soportar esta resolución, su diseño debe incluir dos elementos llave: debe incluir al menos 512 Kb de memoria de despliegue para acomodar la pantalla más larga y debe tener la capacidad para operar en radio de video alrededor de 65 Mhz (o 45 Mhz para despliegues entrelazados).

En esta resolución la pantalla contiene 786,432 pixeles los cuales exceden el número de pixeles (524,288) que pueden ser accesados en un segmento de 64 Kb de memoria de despliegue. Para hacer este largo memoria de despliegue accesible al procesador alguna forma de paginamiento de memoria debe ser empleada. La implementación exacta varia dependiendo de la manufactura.

Algunos despliegues entrelazan para reducir el ancho de banda requerido en esta resolución. Algunas tarjetas VGA soportan tanto despliegues entrelazados como no entrelazados, algunos soportan solo despliegues entrelazados y otros solo despliegues no entrelazados.

Gráficos a 4 Colores 1024 x 768

Gráficos a cuatro colores son populares para publicaciones de escritorio, donde el color usualmente no es tan importante como la resolución. La limitación del número de colores permite altas resoluciones para ser soportadas sin incrementar el tamaño de memoria de despliegue. Esta resolución es la resolución más alta posible en tarjetas con 256 Kb de memoria de despliegue. Algunos colores para cada pixel pueden también resultar en ejecuciones improvisadas si el procesador tiene algunos bits para escribir durante operaciones de dibujo.

El BIOS

Idealmente todas las funciones estándar del BIOS estarán disponibles en todos los modos mejorados del SVGA. Esto desafortunadamente no es el caso para las tarjetas VGA. Virtualmente todos los SVGA soportan la función Seleccionar Modo BIOS en todos los modos de despliegue mejorados, otros soportan diferentes BIOS dependiendo de la manufactura.

Algunas manufacturas tienen funciones mejoradas de texto para trabajar en sus modos de texto mejorados, pero algunos VGA no soportan la habilidad para usar funciones de texto del BIOS

mientras en modo gráfico mejorado si. Estas funciones son especialmente difíciles para soportar en modos gráficos de alta resolución quienes requieren paginamiento de memoria de despliegue.

La complejidad del VGA fue incrementada, el tamaño del ROM BIOS también. El BIOS EGA usa 16 KB de memoria del sistema en el rango de dirección de C000:0000 a C000:3FFF. El VGA usa 24 KB de memoria de sistema en el rango de dirección de C000:0000 a C000:5FFF. Algunas manufacturas han tomado algunos espacios largos de su BIOS, otros han desarrollado métodos de expansión del BIOS sin incrementar su espacio de localización ROM.

Algunas tarjetas SVGA usan un paginamiento de ROM BIOS. Así el ROM BIOS no es afectado por operaciones de escritura de memoria, una operación de escritura al espacio ROM BIOS es usado para seleccionar la página de memoria deseada.

Otras tarjetas localizan alguno de su código BIOS en sistema de memoria. Este método elevara la velocidad de ejecución, si el sistema de memoria es normalmente más rápido que la memoria ROM BIOS. Este puede proveer una medida de ejecución improvisada durante operaciones de texto.

El VESA ha definido un nuevo juego de funciones BIOS las cuales pueden ser usadas para implementar compatibilidad entre diferentes productos VGA.

Otras Características

Algunos productos VGA ofrecen otras características útiles tales como zoom de hardware (la capacidad para alargar una sección de la pantalla), o hardware de soporte para un cursor gráfico. Este tipo de soporte adicional puede proveer la ejecución global de la tarjeta por reducción de sobre carga impuesta en el sistema procesador.

Bus de Datos de 16 bits

Las tarjetas EGA y VGA usan un bus de datos de 8 bits para comunicarse con el procesador. Es una práctica común cuando se diseñan versiones mejoradas de productos para añadir la capacidad de operar completamente en bus de datos de 16 bits, en teoría la transferencia de datos será más rápida con un bus mas ancho. En la mayoría de las aplicaciones el incremento en la transferencia de datos para la memoria de despliegue usualmente tiene un pequeño o no medible efecto en la ejecución del sistema o despliegue, y en algunos sistemas puede haber conflicto con otras tarjetas instaladas.

Para algunas operaciones de dibujo, la mayoría de programadores gráficos VGA elegirán la transferencia de memoria restringida a solo 89 bits para mantener la compatibilidad con algunas tarjetas VGA en los más posible.

I.5 Utilizando 256 Colores

I.5.1 Paleta de Colores (RGB)

El utilizar colores en el modo gráfico implica la combinación de bits, la cual esta ligada a la forma de trabajo con el monitor a color, ya que este a su vez cuenta con tres cañones de electrones que envían ráfagas correspondientes a las señales rojas, verdes y azules (RGB, Red-Green-Blue). El manejo del sistema binario determina el número de colores que se visualizan en pantalla. Evidentemente, son necesarios al menos dos colores para ver un contraste en la pantalla, por ejemplo, blanco y negro (color de primer plano y color de fondo respectivamente). En la pantalla el color se encenderá (bit con valor 1) o bien se apagará (bit con valor 0). Si utilizamos 2 bits para codificar un pixel, como lo hace el CGA, solo encontraremos 4 posibilidades :

00	Color de fondo
01	
10	
11	Color de primer

Una computadora puede tomar 16 colores (4 bits), pero el CGA por sus características puede tomar de esos 16 solo 4, haciendo una combinación de bits en una dirección de memoria, a esta combinación se le conoce como "paleta de colores" .

La paleta de colores es un área de memoria, en la cual se almacenan los 3 colores básicos que la computadora puede utilizar (rojo verde azul), el numero de combinaciones o tones de color que se tengan va a depender del Hardware que este disponible (tarjetas gráficas y monitores).

Por ejemplo:

Un pintor que no cuenta con la inmensa gama de colores que el podría utilizar para pintar un cuadro multicolor, no seria accesible para él tener todos los colores que requiere, es por eso que le seria mas factible contar con los colores primarios (rojo verde y azul) y el color negro, en su paleta, así al mezclar estos colores pueda formar un color o tono de color que necesite.

La computadora puede, dependiendo de su hardware, tener una combinación de varios colores y de esta manera se formara un numero determinado de colores que la computadora puede utilizar. En una tarjeta CGA, solo puede contar con 4 colores de los 16 que puede tomar, los cuales son:

Rojo	Verde	Azul	Color	Paleta	modo CGA
0	0	*	negro	0	4
0	1	*	verde	0	4
1	0	*	rojo	0	4
Rojo	**Verde**	**Azul**	**Color**	**Paleta**	**Modo CGA**
1	1	*	amarillo	0	4
*	0	0	negro	1	5
*	1	1	cian	1	5
*	0	1	magenta	1	5
*	1	1	blanco	1	5

Generalmente se dice que esta tarjeta cuenta con 4 colores, utilizando otro modo del CGA se pueden obtener otras combinaciones de Rojo, Verde y Azul.

En una tarjeta EGA o VGA, que utilice 16 colores, sería una combinación de 3 colores más el de intensidad (o sea 4 bits), esto se logra tomando la operación de combinación, es decir:

$$\text{No. de Combinaciones} = (\text{No. colores o bits})^2 = (4)^2 = 16$$

Estos colores son:

Intensidad	Rojo	Verde	Azul	Numero	Color
0	0	0	0	0	Negro
0	0	0	1	1	Azul oscuro
0	0	1	0	2	Verde oscuro
0	0	1	1	3	Turquesa (verde + azul)
0	1	0	0	4	Rojo oscuro
0	1	0	1	5	Violeta (azul +rojo)
0	1	1	0	6	Marrón (verde +rojo)
0	1	1	1	7	Gris claro (rojo + verde + azul)
1	0	0	0	8	Gris obscuro
1	0	0	1	9	Azul
1	0	1	0	10	Verde
1	0	1	1	11	Cian
1	1	0	0	12	Rojo
1	1	0	1	13	Magenta
1	1	1	0	14	Amarillo
1	1	1	1	15	Blanco

En una tarjeta VGA o superior, que utilice 256 colores, se tendrían disponibles 64 tonos de los colores primarios es decir:

64 tonos de Rojo (de 0 a 63 "6 bits")

64 tonos de Verde (de 0 a 63 "6 bits")

64 tonos de Azul (de 0 a 63 "6 bits")

pero tómese en cuenta que si se utiliza el mismo criterio que en la tarjeta anterior se tendría:

No. De Combinaciones = (# colores) ^{3}
= (64)
= 262144 colores posibles, pero de los cuales solo se pueden tomar 256

Con algunos archivos emuladores se pueden alcanzar hasta 16 millones de colores (3 bytes por cada color $(256)^3$"16,777,216 "256 Tonos de Rojo, Verde y Azul").

CAPITULO

I

CAPITULO II

LENGUAJES DE PROGRAMACIÓN

II.1 Tipos de Lenguaje

El hombre se vale de diferentes formas de lenguajes (oral, escrito, etc.) para comunicarse e interactuar con su exterior, las computadoras realizan lo mismo con dispositivos, periféricos, computadoras incluyendo a quién la maneja (el hombre). Actualmente hay gran variedad de "lenguajes de computadora" o *lenguajes de programación*. Dichos lenguajes ejecutan *programas* los cuales realizan ciertas funciones útiles para usuarios de computadoras, un programa es un conjunto de instrucciones que se ejecutan y proporcionan un resultado, a estos programas se les denomina también software.

Existen *lenguajes de alto y bajo nivel*, los primeros permiten que la función o procedimiento a ejecutar se exprese en un nivel más alto que el lenguaje mismo de la computadora, cada lenguaje lleva consigo un *compilador o interprete*, ya que pueden compilarse o traducirse al lenguaje de la computadora, entre ellos están Ada, ALGOL, BASIC, C, C++, Cobol, Dbase, Pascal, por mencionar algunos. Los lenguajes de programación de alto nivel consisten en caracteres ASCII: letras, números, símbolos, retornos de carro y espacios.

Los lenguajes de bajo nivel usualmente generan una instrucción de máquina, anteriormente existía un lenguaje ensamblador para cada tipo de máquina, siendo diferentes y difíciles de convertir de uno a otro, con la estandarización de máquinas se ha reducido este problema.

A continuación se describen algunas características de cuatro lenguajes, de los cuales dos se utilizarán para realizar la programación del presente trabajo.

II.1.1 Lenguaje BASIC

Es un lenguaje fácil de aprender y de uso general. Ideado para principiantes por John Kemeny y Thomas Kurtz en el Darmont College. En sus inicios aparecieron las siguientes versiones: BASIC para cassette, cuya versión venía en las computadoras personales IBM y se encontraba almacenada

en la memoria ROM. Si la máquina tenía una o más unidades de disco (floppy o drive) se utilizaba la versión denominada BASIC para disquete, modalidad que incluía todos los comandos de BASIC para cassette y otros adicionales que permitían utilizar las unidades de disquete. Otra versión conocida como BÁSICA (abreviatura de Advanced BASIC) con todos los comandos de BASIC para disquete agregando comandos para realizar funciones avanzadas de gráficas, para tocar música y controlar varios dispositivos opcionales, entre ellos control de juegos, pluma electrónica, etc.

BASIC opera en dos modos: modo de comando y modo de ejecución. En el modo comando acepta que se tecleen líneas de programas y comandos tecleados que manipulan los programas, la computadora identifica una línea de programa por su número de línea. Las líneas se ejecutan al momento que el operador de la computadora se lo indique mediante un comando (Run), los comandos si se ejecutan de inmediato. En el modo de ejecución la computadora corre o ejecuta un programa, la pantalla se encuentra bajo el control del programa. Una característica sobresaliente en este lenguaje son los famosos GOTO que se eliminan en la programación estructurada (conocida como programación no GOTO). BASIC es un lenguaje de programación lineal.

BASIC permite un poco de libertad de expresión, sus comandos pueden teclearse en mayúsculas, minúsculas o una mezcla de ambas, los espacios adicionales no se tienen en cuenta. Cada instrucción en un programa BASIC debe ir numerada, ya que la computadora ejecutará las instrucciones en orden ascendente, la instrucción END indica el fin del programa, maneja constantes numéricas y de cadena, variables y palabras reservadas o comandos. Además todas las variables pueden declararse en cualquier parte del programa.

Surgieron nuevas versiones del BASIC, en donde se podría decir que su forma de trabajo ya es estructurada pero no completamente ya que el programador en cualquier momento que desee, puede regresar al método de programación del BASIC anterior, versiones recientes del BASIC son GWBASIC, Turbo BASIC, QBASIC y una versión bajo Windows que proporciona *MicroSoft*, "VISUAL BASIC".

II.1.2 Lenguaje ASM

La forma específica de un lenguaje ensamblador depende de la arquitectura y configuración de la máquina en la que se esté trabajando, el componente que determina el diseño del ensamblador

es la unidad central de procesamiento CPU ya que este mantiene el control de todos los componentes (Figura II.1).

El CPU contiene el microprocesador que posee la habilidad de manipular todo lo que entra, sale y se procesa en la computadora a través de los dispositivos que interactuan con ella (aunque actualmente existen dispositivos que realizan procesos que anteriormente hacía el microprocesador, para agilizar el trabajo). Cada uno de éstos CPU's deben ser programados en lenguaje de máquina (es decir, por medio de cadenas de unos y ceros); sin embargo, su fabricante establece un grupo de mnemónicos para representar los distintos patrones asociados con cada instrucción implantada en el CPU. Estos mnemónicos sirven para desarrollar el lenguaje ensamblador asociado con determinada arquitectura.

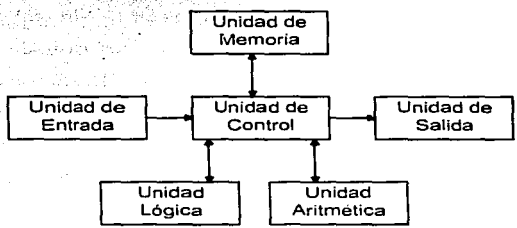


Figura II.1 Arquitectura básica de una computadora.

Los programadores que emplean lenguajes de alto nivel para desarrollar aplicaciones donde el tiempo no es un factor crítico o hacen uso de dispositivos estándar de entrada/salida (I/O), no necesitan ejecutar rutinas que no sean parte de la librería del compilador, es decir, no necesitan programar en lenguaje ensamblador. Sin embargo, alguien debe escribir las rutinas de librería para obtener una interfaz estándar, ya que estas rutinas forman la parte no transportable del lenguaje que se este utilizando y están escritas en lenguaje ensamblador. Si parte del código de un programa escrito en lenguaje de alto nivel tarda en ejecutarse es recomendable escribir una rutina en lenguaje ensamblador que haga lo mismo que el código en cuestión. Programadores experimentados pueden eliminar los loops y otras situaciones parecidas, o por lo menos minimizarlos, mejorando el código generado.

Si existe la necesidad de instalar un dispositivo de propósito especial, el programador debe escribir un manejador o driver para este dispositivo, siendo más fácil escribir y depurar tal manejador en lenguaje ensamblador que en algún lenguaje de alto nivel.

Las técnicas de inteligencia artificial (IA) pueden disminuir los requerimientos de programación de hardware.

II.1.3 Lenguaje C

El lenguaje C proporciona un entorno de programación de propósito general estructurado y modular.

C comienza con un estudio sobre el sistema operativo *UNIX*, ya que tanto el sistema como la mayoría de los programas que se ejecutan en él están escritos en C. Sin embargo, C no esta atado a UNIX, ni a ningún otro sistema operativo o máquina. UNIX fue desarrollado originalmente en 1969 en la DEC PDP-7 en los Laboratorios Bellen New Jersey. UNIX fue escrito totalmente en el lenguaje ensamblador del PDP-7. Algunos descendientes de C serían:

- Algol 60 Diseñado por un Comité internacional a principios del año 1960.
- CPL Lenguaje de programación combinado desarrollado tanto en Cambridge como en la Universidad de Londres en 1963.
- BCPL Lenguaje de programación combinado básico desarrollado en Cambridge por Martín Richars, en 1967.
- B Desarrollado por Ken Thompson, laboratorios Bell, en 1970.
- C Desarrollado por Denis Ritchie, laboratorios Bell, en 1972.
- CLIPPER Lenguaje manejador de Bases de Datos.

Dennis Ritchie destacó debido a su trabajo en software de sistemas: lenguajes de computadora, sistemas operativos y generadores de programas. C tiene la reputación de ser un lenguaje de programación de sistemas por que es útil para escribir compiladores y sistemas operativos, es un lenguaje relativamente de bajo nivel, que permite especificar cada detalle en la lógica de un algoritmo para lograr la máxima eficiencia de la computadora. También es un lenguaje de alto nivel que puede ocultar los detalles de la arquitectura de la computadora, incrementando la eficiencia de la programación.

C++ tiene un origen similar al de C. Aunque C++ tiene algo en común con BCPL y programación orientada a objetos (*POO*). Los constructores de clase son el vehículo fundamental para la programación orientada a objetos. Los métodos constructores y destructores se utilizan para garantizar la inicialización de los datos definidos en un objeto de una clase específica. El concepto de ocultación y encapsulamiento de datos implica la denegación del acceso de la estructura interna de un objeto. Existen ciertas diferencias al escribir funciones de C, dependiendo si los programadores tratan (o no) de ajustarse a la norma *ANSI*. El C++ de *Borland* reconoce esta y otras normativas anteriores de C.

II.1.4 Lenguaje Pascal

El lenguaje Pascal fue desarrollado en Zurich por Niklaus Wirth, a finales de los años sesenta. Surgió como respuesta a la insatisfacción que se tenía con los lenguajes modulares (orientados a rutinas) y se diseño para ser enfocado a la enseñanza de la programación como una disciplina y aplicar los métodos de programación estructurada definidos por Dijkstra y Hoare. La programación en Pascal con la metodología de diseño y programación estructurada da por resultado sistemas flexibles, modulares, comprensibles, fáciles de modificar, corregir y dar mantenimiento.

Pascal es un lenguaje ordenado que consiste en secciones bien definidas, cada una de las cuales sirve para un propósito específico. Dichas secciones son conocidas como módulos, un conjunto de instrucciones identificado con un nombre determinado que hacen una y solo una función específica, dependiente o independientemente de otros y puede conformarse o ser parte de otro(s) módulos.

El compilador de Pascal ofrece diversas opciones que se pueden utilizar para construir programas y depurarlos más fácilmente. Dichas opciones, que realizan tareas tales como la comprobación de errores, se llaman directivas de compilación, ya que dirigen al compilador en su trabajo.

Pascal soporta múltiples segmentos de código, un segmento de código para cada unidad y otro adicional para el programa principal. Las llamadas a funciones y procedimientos que se encuentran dentro de un archivo de unidad o de programa se denominan llamadas intrasegmento o cercanas, ya que tanto el código que ejecuta la llamada como el procedimiento llamado residen en el mismo segmento de código. Sin embargo, cuando una sentencia en una unidad llama a un procedimiento en otra unidad, dicha llamada se denomina llamada intersegmentos o lejana, ya que la llamada cruza las fronteras del segmento de código. Las llamadas cercanas requieren menos trabajo que las lejanas

porque el segmento de código no cambia. Las llamadas lejanas, por el hecho de implicar a más de un segmento de código, requieren más trabajo y se ejecutan más despacio. Afortunadamente, Pascal es suficientemente inteligente como para saber cuando debe utilizar una llamada cercana y cuándo utilizar una lejana. No obstante, hay veces en que es deseable ignorar los juicios de Pascal y forzar a que un procedimiento sea llamado con una llamada lejana, aunque normalmente la llamada se considerara como una llamada cercana.

Es posible escribir rutinas en ensamblador para utilizarlas en programas en Pascal enlazando los archivos objeto de ensamblador con el programa en Pascal.

II.2 Porque C y Pascal

"Los problemas de programación diferentes requieren de soluciones diferentes. La elección del mejor lenguaje para un proyecto corresponde al ingeniero de software. En cualquier proyecto esta es una de las primeras decisiones que necesitamos tomar y es casi irrevocable una vez que se empieza a codificar. La elección de un lenguaje de programación puede marcar la diferencia entre el éxito y el fracaso del proyecto."

Tomando en cuenta el párrafo anterior el haber elegido los lenguajes C y Pascal reside en las características de cada uno mencionadas con anterioridad, además de la experiencia de programación en ellos y, salvo la opinión del lector, ya que cada persona (en este caso programador) tiene un punto de vista particular, esperamos haya sido la decisión correcta, a continuación se mencionan algunas ventajas y desventajas del los lenguajes C, Pascal y Ensamblador.

II.2.1 Ventajas y Desventajas

Ventajas de C

C incluye todas las estructuras de control de un lenguaje moderno. Aunque se considera que C es anterior a la programación formal estructurada. Incorpora bucles for, construcciones if-else, instrucciones case y bucles while, permite separar código y datos controlando su ámbito. Proporciona el concepto de compilación y enlace separados que permite volver a compilar solo las partes de un programa que se han cambiado durante el desarrollo, importante cuando estamos desarrollando programas grandes, o incluso programas medianos en sistemas lentos.

C tiene menos reglas de sintaxis que muchos otros lenguajes y podemos escribir un compilador de C de alta calidad que trabaje solamente con 256K de memoria total. En C realmente hay mas operadores y combinaciones de operadores que palabras clave (aproximadamente 77).

Lenguaje pequeño, sistema de ejecución pequeño y un lenguaje cercano al hardware hace que la velocidad de ejecución de muchos programas en C se aproxime a la de sus equivalentes en lenguaje ensamblador.

Proporciona un amplio conjunto de operadores de manipulación de bits.

C se destaca por su capacidad de realizar aritmética de punteros para poder direccionar áreas específicas de memoria. Esta capacidad aumenta la velocidad de ejecución de un programa.

El lenguaje ensamblador es específico para cada maquina (microprocesador) y, por tanto, no es tan portable como el código en C. Borland tiene un ensamblador que permite desarrollar código para la familia de microprocesadores Intel (8086, 8088, 80186, 80286, 80386, 80486 y Pentium). Podemos escribir programas autónomos en lenguaje ensamblador con el editor de Borland y entonces ensamblar los y enlazarlos. También podemos integrar rutinas de C en programas en lenguaje de ensamblador.

Su interfaz de hardware-software, su sistema de programación de bajo nivel, su eficiencia, economía y sus poderosas expresiones. Prácticamente, una aplicación en C++ puede incorporar tanto el modelo de programación estructurada como el modelo nuevo orientado a objetos.

El C++ de Borland nos permite escribir programas que pueden llamar fácilmente a otras rutinas escritas en diferentes lenguajes (Pascal, por ejemplo).

Desventajas de C

C no realiza una verificación fuerte de tipos, la tipificación es una medida al usar tipos de variable (por ejemplo, entero y punto flotante son dos tipos diferentes de numero). En ocasiones no se puede asignar un tipo de dato a otro sin llamar a una función de conversión, evitando la modificación de ellos con redondeos inesperados.

La falta de verificación en el sistema del tiempo de ejecución de C origina problemas misteriosos y transitorios que pueden pasar desapercibidos tal como el exceso de los limites de un arreglo.

Ventajas de Pascal

Pascal incluye todas las estructuras de control de un lenguaje moderno. Tales como bucles for y while, construcciones if-then-else, repeat-until, instrucciones case, permite separar código y datos controlando su ámbito. Proporciona el concepto de compilación y enlace separados que permite volver a compilar solo las partes de un programa que se han cambiado durante el desarrollo.

La base de la programación modular es dividir un programa en pequeñas partes, conocidas como procedimientos y funciones. Siendo programas más fáciles de escribir y mantener ya que cada procedimiento o función puede ser diseñado y ejecutado por separado para posteriormente enlazarlo al programa principal, sin riesgo de error, además permite la transmisión de variables al procedimiento o función.-

Desde la versión 5.5 Borland introdujo a Pascal el soporte para POO, quienes dependen de tres conceptos fundamentales:

1. Combinación de códigos y datos.
2. Herencia
3. Encapsulación.

Disciplina a los programadores con buenos hábitos de programación, proporcionando un vasto rango de herramientas de programación para escribir un código claro y sencillo comparado con otros lenguajes (como BASIC).

Es un lenguaje con fuerte comprobación de tipos. No se pueden mezclar variables de tipos diferentes. En las sentencias de asignación, los valores de la derecha deben ser de tipos compatibles con el de la variable de la izquierda.

Proporciona un amplio conjunto de operadores de manipulación de bits.

Posee la técnica de recursividad, donde un procedimiento de llama así mismo al momento de su ejecución.

Siendo un lenguaje de alto nivel es transportable de una computadora a otra sin mayores modificaciones.

Es posible escribir un programa en Pascal que llame a una rutina en C++. Se debe considerar dos puntos muy importantes: los nombres de los identificadores y el modo en que se pasan los parámetros.

Desventajas de Pascal

Aunque una fuerte comprobación de tipos tiene ciertas ventajas, es preferible tener un menor grado de comprobación de tipos.

Ventajas de Ensamblador

Control de hardware.

Desarrollo de fragmentos de programas de rápida ejecución.

Acceso de forma óptima y eficiente al procesador.

Compresión de métodos que realizan una sintaxis asociada con lenguajes de alto nivel.

Disciplina para programar en forma estructurada.

Ayuda a comprender la manera en que el procesador manipula las diversas estructuras de datos.

Desventajas de Ensamblador

Los lenguajes ensambladores proporcionan a los programadores una alternativa para trabajar directamente con el juego de instrucciones incorporadas por la computadora, fuerzan a pensar en términos del hardware, hay que especificar cada operación en términos de la máquina. Se deben mover bits hacia o desde los registros, sumándolos, desplazando el contenido de un registro a otro y almacenando los resultados en memoria, representando un esfuerzo tedioso, propenso a errores además de que el programador debe poseer un gran dominio de la arquitectura de su equipo tanto física (Hardware) como lógica (Software). La programación del manejador del hardware está íntimamente ligada a la arquitectura de la computadora: la dirección de los periféricos (código contenido en el BIOS), así como, el acceso e inicialización de cada circuito inteligente de soporte que se ejecuten.

II.2.2 BASIC (¿Olvidado...?)

Algunas características por las que no se utilizó BASIC para la programación de este trabajo se mencionan a continuación.

Surge como herramienta de enseñanza, siendo un lenguaje de aprendizaje fácil pero para no programadores expertos, da acceso a la gente inexperta para aprender sencillos programas, sin embargo crea hábitos de programación pobres y genera código ilegible, difícil de comprender. Sin

embargo, es significativo el hecho que versiones más actuales de BASIC contengan algunas características de Pascal.

Todas sus variables son globales, se puede hacer referencia a cualquier variable en cualquier momento dentro del programa, pudiendo crear confusión al manipularlas.

Aquellas personas que mantienen un contacto constante con el mundo de la programación, en especial del desarrollo de la programación estructurada, recuerda el uso de la sentencia GOTO, la cual permitía un salto a cualquier punto del programa sin importar la consistencia en el flujo del programa. La depuración y mantenimiento de programas que utilizan esta sentencia es difícil y compleja.

Pero BASIC a pesar de todo esto es de gran utilidad para programadores que quieren realizar funciones de manera rápida, como la comunicación de dos computadoras o manipulación de los puertos paralelo y serie, aunque se puede realizar en cualquier lenguaje en BASIC es mas sencillo, nosotros en lo personal no dejamos aun el BASIC incluso le llamamos "El genial BASIC".

II.3 Lenguaje de Programación Pascal

II.3.1 Estructura de un Programa

Un programa en Pascal distingue tres secciones principales: la cabecera, las declaraciones y el bloque. La cabecera se encuentra al principio del programa. La sección de declaraciones define las constantes, los tipos de datos, las variables, las etiquetas, los procedimientos y las funciones; el bloque contiene las estructuras de control.

En general un programa de Pascal contiene la siguiente estructura:

- Un encabezado PROGRAM. Indica el nombre del programa, este encabezado es opcional desde la versión 5.0.
- Una instrucción USES. Especifica las unidades que se incluirán en la compilación del programa.
- Un grupo de instrucciones opcionales de declaración. Indica las etiquetas, constantes, tipos y variables para el programa.
    - LABEL define los identificadores de etiqueta para las instrucciones GOTO (ya que la gran mayoría de los programadores no utilizan los GOTO, LABEL rara vez aparece en un programa).

- CONST define las constantes simbólicas que se utilizarán en el programa.
- TYPE especifica los tipos de datos definidos por el usuario que se utilizarán en el programa.
- VAR define la variables que se utilizarán en el programa.

- Una sección de procedimientos y funciones. Contienen las instrucciones a desarrollar para realizar tareas definidas por el programa. Un procedimiento ejecuta una acción definida, y una función devuelve un solo valor como resultado.
- Una sección principal del programa. Controla el programa haciendo llamadas a los procedimientos y funciones.

El diagrama de bloque de la estructura general de un programa en Pascal se muestra en la figura II.2.

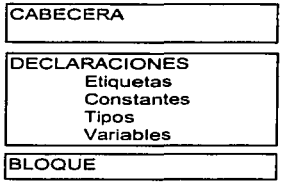


Figura II.2 Estructura general de un programa en Pascal.

Cada procedimiento y función, al igual que el programa principal, contienen la misma estructura general (cabecera, declaraciones, bloque), inician con la sentencia Begin del bloque principal y continúa hasta que alcanza la sentencia End, es posible encontrar begin y end entre el begin y end principal (un programa puede terminar también por encontrar la orden Halt o debido a un error fatal, pero estos casos son excepciones). Cada línea de instrucción en Pascal debe terminar con un punto y coma (;). Como se muestra en el ejemplo de la figura II.3.

```
CABECERAS
  Program  Nombre;
DECLARACIONES
   USES unidad_1,...,unidad_n;
   LABEL eti_1,...;eti_n;
   CONST const_1=valor_1;
                  .......
             const_n=valor_n;
   TYPE    var_1=RECORD
                 { estructura del tipo}
           END;
                  ........
            var_n=RECORD
END;
   VAR   var_1:tipo;
          ......
          var_n:tipo;

   FUNCTION nombre_funcion (parámetros): valor_retorno;
                        { estructura de la función}

   PROCEDURE nombre_procedimiento (parámetros);
                        { estructura del procedimiento}
BLOQUES
  BEGIN
        { estructuras de control}
  END.
```

Figura II.3 Estructura ejemplo de un programa en Pascal.

Tipos de Variables

Pascal posee cuatro clases fundamentales: palabras reservadas, identificadores, constantes y símbolos. Las palabras reservadas son utilizadas exclusivamente por el lenguaje; no se pueden usar para ningún otro propósito. Un identificador es una palabra que se usa para referenciar una posición de memoria, es el nombre de una variable, un tipo, un subprograma, un parámetro o algo similar, deben comenzar con una letra o un símbolo de subrayado y siguen con letras, números y subrayados. Sólo se comprueban los primeros 63 caracteres para determinar las diferencias. Ni las palabras reservadas ni los identificadores diferencian entre mayúsculas o minúsculas, de modo que, no importa que las letras estén en minúsculas o mayúsculas.

Una constante es un valor que nunca cambia. En Pascal las constantes son números o cadenas de caracteres, Pascal permite definir constantes propias. Un símbolo es un caracter, o una

secuencia de caracteres, que tiene un significado especial en el lenguaje de alto nivel. La sintaxis del lenguaje define cómo se combinan los símbolos, las constantes, los identificadores y las palabras reservadas.

Cabeceras

Consiste de la palabra Program seguida del nombre del programa. En Pascal estándar se trata de una línea obligatoria (en versiones posteriores se puede omitir), identifica el nombre del programa e indica si se realiza entrada, salida o ambas.

La cabecera de un subprograma incluye, como mínimo, el tipo de subprograma (procedimiento o función) y su nombre. Las funciones también requieren que se especifique el tipo que devuelven. También puede incluir una lista de parámetros en la cabecera de un subprograma, las unidades también tienen cabeceras, que consisten en la palabra Unit seguida del nombre de la unidad.

Declaraciones

Incluye las definiciones de variables, constantes, etiquetas, tipos y subprogramas. Ya que los subprogramas son también entidades permite anidar subprogramas dentro de otros subprogramas. En Pascal estándar, la sección de declaraciones tienen el siguiente orden: primero las constantes, luego los tipos y después las variables. Versiones actuales de Pascal pueden seguir el orden que se desee. La única restricción es que se debe declarar un identificador antes de poder usarlo.

- Constantes. Pascal no inicia las variables al principio de la ejecución del programa. Por lo tanto, no hay forma de saber qué valor contiene una variable antes de que se le asigne uno. Sin embargo, cuando el programa comienza, las constantes tienen asignados valores especificados por el programador. La palabra reservada const indica el área de definición de constantes. Hay dos tipos de constantes en Pascal: con tipo y sin tipo. Una constante sin tipo se declara con la sintaxis siguiente: un identificador seguido por un signo igual, un valor literal (numérico o de texto) y un punto y coma, y no se especifica un tipo en su definición, se puede decir que son las verdaderas constantes, ya que no se puede cambiar su valor en ningún momento de la ejecución del programa. Las constantes con tipo se definen de forma similar, exceptuando que se inserta la definición de tipo entre el identificador y el signo igual.
- Tipos. En el área de definiciones de tipos, que comienza por la palabra reservada de Pascal Type, se pueden definir tipos de datos propios que se utilicen después para declarar variables. El

formato general para las definiciones de tipos es de un identificador seguido por un signo igual, el tipo de datos y terminado en un punto y coma.

- Variables. Las variables son áreas de memoria a las que se les da un nombre. Para comenzar un bloque de declaración de variables se coloca la palabra reservada Var, seguida del identificador de la variable (el nombre que se le da a la variable), dos puntos y el tipo de dato. Se pueden declarar variables utilizando los tipos de datos estándar de Pascal (por ejemplo, boolean, integer, real) o tipos de datos definidos por el usuario establecidos en la sección type. El formato para la declaración de variables es prácticamente igual que el que se utiliza para las definiciones de tipos, pero en este caso el identificador no está seguido de un signo igual sino de dos puntos.

A medida que un programa aumenta de tamaño se hace necesario dividirlo en partes pequeñas y manejables. Pascal proporciona tres mecanismos para hacerlo: unidades, inclusión de archivos y solapamientos.

Unidades

Una unidad es una agrupación lógica de declaraciones que se compila por separado y que es accesible para otros programas y unidades. Las unidades pueden contener tipos, constantes, variables y subprogramas. Una unidad tiene tres ventajas principales:

- Modularidad. Las unidades se compilan por separado, parten un programa en trozos manejables.
- Reutilización. Las declaraciones están disponibles para otro programa o unidad, por lo que las unidades constituyen bibliotecas de componentes reutilizables.
- Ocultamiento de información. Están estructuradas de forma que un programa que las utilice no tenga acceso a los detalles de implementación.

Si se necesita acceder a alguna declaración o algún subprograma de otra unidad estándar, se debe incluir una cláusula Uses. Esta cláusula es simplemente una lista de las unidades en las que se debe buscar durante el proceso de compilación. La cláusula uses es parte de la cabecera del programa y debe aparecer antes que cualquier declaración del programa.

Inclusión de archivos

El editor de Pascal no alberga más de 62K de texto en un programa. Si se excede éste el límite, hay que dividir el programa en múltiples archivos. Cuando se compila el programa, la directiva de inclusión de archivo coloca todas las piezas juntas a partir de los distintos archivos. La

directiva de inclusión de archivo resulta también útil cuando se tienen bibliotecas de rutinas estándar que se usan con frecuencia.

Para incluir un archivo en un programa de Pascal hay que colocar un comentario con la directiva de compilación **$I** y el nombre del archivo que se va a utilizar.

Solapamientos

Pascal proporciona una unidad estándar llamada **Overlay** (Solapamiento). Al incluir esta unidad en la cláusula Uses de un programa se pueden solapar las unidades del programa para minimizar la cantidad de memoria que requiere el programa. El uso de los solapamientos es bastante fácil siempre que se sigan las siguientes reglas:

- Activar la directiva de compilación de Forzamientos de llamadas lejanas ({$F+}). Para el programa y para todas las unidades.
- Nombrar la unidad Overlay como primera unidad de la cláusula uses del programa.
- Compilar las unidades solapables con la directiva de compilación {$O+}.
- Especificar las unidades solapables utilizando la directiva de compilación {$O Nombre_archivo}.
- Asegurarse de que el programa inicia con **OvrInit** el mecanismo de solapamiento antes de ejecutar cualquier sentencia, incluyendo las de las secciones de inicialización de unidades solapables.

II.3.2 Manejo de Punteros

Pascal utiliza diferentes partes o segmentos de la memoria de la computadora para distintos propósitos. Algunos segmentos guardan las instrucciones que ejecuta la computadora mientras otros almacenan los datos. Aunque la mayor parte de la gestión de memoria de Pascal se hace " entre bastidores", comprendiendo los segmentos y sus papeles se consigue un mejor uso de la memoria.

Pascal divide la memoria de la computadora en cuatro partes: el segmento de código, el segmento de datos, el segmento de la pila y el montón. Los programas que utilizan unidades tienen un segmento de código para cada unidad así como uno para el programa principal. Sin embargo, todos los programas tienen un único segmento de datos, que contiene las constantes con tipo y las variables globales. Aunque el segmento de datos está dedicado específicamente al almacenamiento de los datos, éstos también se pueden almacenar en otras posiciones. La pila y el montón contienen

los datos dinámicos, asignando memoria según se vaya necesitando. Debido a que la pila es de vital importancia, su operación es controlada automáticamente por Pascal, el usuario por sí mismo no puede hacer gran cosa con la pila. Por el contrario, el montón tiene gran importancia para las técnicas avanzadas de programación.

Cuando un programa arranca, abre un segmento para las instrucciones del programa (el segmento o segmentos de código), un segmento para albergar los datos del programa (el segmento de datos) y un segmento que guarda los datos temporalmente (el segmento de la pila). A medida que se van ejecutando las instrucciones del segmento de código, éstas manipulan los datos del segmento de datos y del segmento de la pila.

Mediante el uso de direcciones un programa sabe en qué byte comienzan los segmentos o localiza un byte en particular. Cada byte tiene una dirección, un valor de 20 bits que identifica a esa posición única. Cuando un programa necesita acceder a un byte en particular, utiliza la dirección para encontrar la posición del byte en memoria.

Si la dirección de una computadora consistiese en una palabra simple (dos bytes), no podría direccionar más de 64K (65536 bytes) de RAM. Este era el caso de los primeros microprocesadores de 8 bits. La llegada de los procesadores de 16 bits, en particular de la familia Intel 8086/88, trajo consigo un nuevo esquema de direccionamiento de memoria, conocido como direccionamiento segmentado. El direccionamiento segmentado combina dos valores de palabra, un segmento y un desplazamiento, para formar una dirección de 20 bits. Piense en los segmentos como los bloques de una calle y en los desplazamientos como las casas de cada bloque. Cada segmento alberga 64 Kb. de RAM. Los procesadores 8086/88 tienen 16 segmentos, dando como resultado 1.048.560 bytes (1Mb.) de memoria direccionable. No obstante, el DOS limita a 640 Kb. la cantidad de memoria que puede utilizar la computadora. En el modo protegido sólo se está limitado por la cantidad total de memoria RAM disponible en la máquina.

Segmentos y Desplazamientos

Pascal proporciona dos funciones estándar, Seg y Ofs, que facilitan la exploración del direccionamiento de memoria del PC. Seg proporciona el segmento en el que resiste la variable y Ofs proporciona su desplazamiento.

Las variables que son globales tienen el mismo segmento. (Las variables globales residen en el segmento de datos.) Además, la distancia entre desplazamientos coincide exactamente con el

número de bytes que se necesitan para almacenar cada tipo de variable. Por ejemplo, una palabra (word) comienza en el desplazamiento 5Ch y necesita dos bytes de almacenamiento; la variable de la siguiente línea (de tipo string) comienza en el desplazamiento 5Eh. Las constantes con tipo de Pascal se guardan en el segmento de datos, mientras que las constantes sin tipo lo hacen en el segmento de código. En realidad, las constantes sin tipo no tienen direcciones. Las variables que se declaran en las funciones y los procedimientos se guardan en la pila, un área de almacenamiento dinámico de datos. Cuando un programa llama a un procedimiento Pascal va añadiendo variables a la pila, la pila crece hacia abajo en la memoria. Cuando termina el procedimiento, Pascal desecha esas variables y libera la memoria para que sea utilizada de nuevo. El cuarto segmento de la memoria en Pascal, el montón, es un área dinámica de datos que controla el programador. El montón permite un uso eficiente de la memoria ya que elimina la necesidad de mantener todas las estructuras de datos a lo largo del programa; en lugar de ello, se puede crear una variable en el montón en un momento dado, borrarla del montón más adelante y reutilizar el espacio para alguna otra variable.

La figura II.4 [1] proporciona un diagrama esquemático de la memoria de Pascal. El segmento de código ocupa la parte más baja de la memoria seguido de los segmentos de datos y de la pila. El montón ocupa toda la parte alta que queda de la memoria, hasta el máximo que se haya fijado con la directiva de compilación SM. El diagrama muestra también que la pila crece hacia abajo y el montón crece hacia arriba.

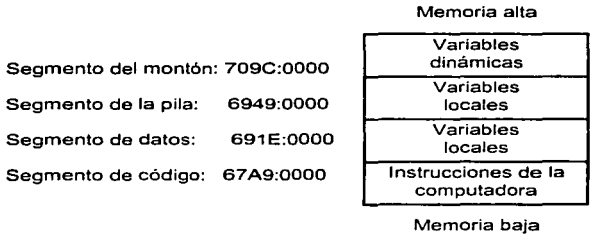


Figura II.4 Manejo de memoria en Pascal.

[1] Stephen K. O'Brien,Steve Nameroff. Turbo Pascal 7. Manual de referencia.. cap. 6 p. 158., México D.F., Ed. Mc Graw-Hill. 1995.

Punteros

La mayoría de las variables que se declaran en Pascal son estáticas, es decir, se les asigna memoria en el momento en que comienza el programa y la mantienen hasta que termina. Por el contrario, el montón utiliza tipos de datos dinámicos conocidos como punteros. Las variables puntero pueden utilizar y reutilizar la memoria del montón.

El uso de variables puntero del montón ofrece dos ventajas principales:

- Aumenta la cantidad total de espacio de datos disponible para un programa. El segmento de datos está limitado a 64 Kb., pero el montón sólo está limitado por la cantidad de RAM de la computadora.
- El uso de variables puntero del montón permite que el programa se ejecute con menos memoria. Por ejemplo, un programa podría tener dos estructuras de datos muy largas, pero usando sólo una de ellas cada vez. Si esas estructuras de datos se declaran globalmente, residirán en el segmento de datos y ocuparán memoria todo el tiempo. Sin embargo, si esas estructuras de datos se definen como punteros, se pueden colocar en el montón y utilizarlas cuando sean necesarias, con lo que se reducen los requisitos de memoria del programa .

Variable Puntero

Una variable puntero no contiene datos de la misma forma que lo hacen las demás variables, ya que guarda la dirección que apunta a una variable que se encuentra en el montón. El símbolo ^ se coloca delante del tipo de datos en la definición e indica a Pascal la definición de una variable puntero.

Cuando se realizan operaciones de direccionamiento, a menudo es necesario asignar a un puntero la dirección de una variable o de un procedimiento. Esta operación se lleva a cabo por medio del operador @ que devuelve la dirección del identificador al que precede.

Una de las reglas de Pascal es que todos los tipos puntero son compatibles en asignación. Esto significa que se puede pasar a un subprograma de un puntero a cualquier variable. Si se pasa a un método un puntero a una variable objeto, ese método puede a su vez llamar a un método virtual, lo que proporciona un polimorfismo extremadamente potente.

Asignación Dinámica de Memoria

Cuando comienza el programa, el montón es una pizarra en blanco. Antes de que se pueda usar el puntero se debe utilizar la sentencia New(var) para indicar a Pascal que asigne a la variable puntero una dirección del montón. Dispose(var); que aparece cerca del final del programa, es lo contrario de New. Dispose "saca" una variable del montón, liberando la memoria para que otras variables la utilicen.

Una vez que se coloca en el montón, la variable se puede utilizar en sentencias aritméticas y de asignación, colocando el símbolo ^ delante del identificador. El símbolo ^ indica que se está refiriendo a la variable del montón y no al puntero en sí.

Pascal ofrece una alternativa al uso de New y Dispose para asignar de forma dinámica la memoria: Mark y Release. En lugar de dejar huecos en el montón como hacen New y Dispose, Mark y Release gestionan una parte final del montón marcada a partir de un determinado punto.

Otro método de asignación dinámica de memoria se basa en el uso de GetMem y FreMem. Son similares a New y Dispose porque asignan y liberan memoria variable a variable. El valor especial de GetMem y FreMem es que se puede especificar la cantidad de memoria que se quiere asignar independientemente del tipo de variable que se esté utilizando. El número de bytes que se especifique en la sentencia FreMem debe coincidir con el de la sentencia GetMem. No utilice Dispose en lugar de FreMem; si lo hace, el montón se desincronizará sin remedio.

II.3.3 Manejo de Archivos

Pascal soporta tres tipos de archivos: de texto, con tipo y sin tipo. Estos tres tipos se utilizan como entrada y salida de información, deben tener un nombre entre 1 u 8 caracteres, y una extensión (.TXT, .DAT, .BMP).

II.3.3.1 Creación y Apertura de Archivos

Archivos de Texto

Consiste en una serie de líneas que termina con un retorno de carro y un salto de línea (juntos de conocen como delimitador). Primero se debe declarar un identificador de archivo, igual que los identificadores de variables, antes de utilizar dicho archivo para entrada o salida se debe

asignar aun archivo de disco, con la sentencia Assign. Posteriormente se prepara al archivo de disco con alguna de las siguientes órdenes: Reset, Rewrite o Append.

Reset abre el archivo de disco y lo prepara como archivo de entrada, sólo se pueden utilizar órdenes de entrada, es decir, no se puede escribir en él. Además Reset coloca al principio del archivo el puntero de archivo, el sigue la posición del archivo en un programa haciendo que las entradas comiencen al principio del archivo y se dirijan hacia adelante.

Rewrite y Append preparan un archivo de texto para salida, si el archivo existe se borra su contenido al utilizarlo con Rewrite, si no existe lo crea. Append conserva el contenido del archivo y coloca el puntero de archivo al final del archivo, es decir, va añadiendo datos.

Al término del uso de un archivo, éste se debe cerrar, mediante la orden close, asegurando que los datos que localizados en los buffers temporales se almacenen, además libera el gestor de archivos; mecanismo que proporciona DOS a los programas para gestionar las operaciones de archivo. Close también actualiza el directorio de archivos con las modificaciones hechas en tamaño, hora y fecha del archivo.

Al abrir un archivo con Reset se puede obtener información de él con los procedimientos Read y Readln y colocarlos en alguna variable, con características para manipular la información que se tome del archivo. Con las órdenes Write y Writeln se puede mandar la salida al archivo y enviar sólo la información que se desea.

Archivos con Tipo

Contienen datos de un tipo concreto, haciendo la programación más sencilla y eficiente, proporcionando una entrada y/o salida más rápida que los archivos de texto. Tienen una estructura rígida dada por el tipo de datos que posee. Su declaración es la siguiente: var : file of type;

Este tipo de archivo se encuentra organizado en registros, siendo un registro por elemento, la longitud del registro corresponde con el número de bytes requeridos por el tipo de datos. Utiliza la mismas ordenes que los archivos de texto excepto por Witeln y Readln ya que no están hechos por líneas.

Los datos que se almacenan en un archivo con tipo tienen la misma forma de como se almacenan en la memoria RAM y se pueden leer directamente de ella, por ello son más rápidos.

Archivos sin Tipo

Asumen datos sin algún tipo y su transferencia es inmediata, se utilizan en aplicaciones que requieren rapidez en su entrada y/o salida. Su declaración es la siguiente: var : file; donde la palabra reservada file no es seguida de una especificación de tipo.

Las órdenes Reset y Rewrite pueden llevar más parámetros, como su tamaño de registro. Otras órdenes especiales son BlockRead y BlockWrite. Los archivos con tipo y sin tipo también se conocen como archivos de acceso aleatorio, ya que se puede acceder a sus registros en orden no secuencial.

II.3.3.2 Lectura y Escritura de un Byte desde un Archivo

Para poder leer o escribir un byte desde un archivo se utilizan las sentencias BlockRead y BlockWrite.

Para BlockRead tenemos los siguientes parámetros:

BlockRead(Archivo_Fuente, Buffer, Sizeof(Buffer), Bytes_Leidos);

Archivo_Fuente. Identifica el archivo donde se va a leer.

Buffer. Indica la estructura donde se van a colocar los datos leídos.

SizeOf. Especifica el tamaño en bytes a leer.

Bytes_Leídos. Identifica cuantos bytes se han leído con la sentencia BlockRead.

De forma similar para BlockWrite se tiene:

BlockWrite(Archivo_Destino, Buffer, Bytes_Escritos);

Archivo_Destino. Identifica el archivo donde se va a escribir.

Buffer. Indica la estructura donde se encuentran los datos a escribir.

Bytes_Escritos. Identifica el numero de bytes que se escribieron con éxito.

Operadores de Bits

Los operadores shl (desplazamiento hacia la izquierda, en inglés, shift-left) y shr (desplazamiento hacia la derecha, en inglés, shift-right) desplazan los bits de un byte a la izquierda o a la derecha. Un byte se puede desplazar a la izquierda o a la derecha un máximo de ocho veces, en cuyo caso todos los bits quedan a cero. Cuando un byte se desplaza una posición hacia la izquierda, cada bit del byte se traslada una posición hacia la izquierda. El bit del extremo izquierdo se pierde y aparece un cero

en la posición del extremo derecho. El desplazamiento a la derecha (shr) opera de igual forma que el desplazamiento a la izquierda, pero en dirección contraria. Cuando se desplaza un byte hacia la derecha , el bit de más a la derecha se pierde y el bit de la izquierda se pone a cero.

II.4 Lenguaje de Programación C

II.4.1 Tipos de variables

C proporciona las siguientes clases fundamentales: palabras reservadas, identificadores, constantes y símbolos. Las palabras reservadas son identificadores predefinidos que tienen un significado especial para el compilador C. Un identificador es una palabra que se usa para referenciar una posición de memoria, es el nombre de una variable, un tipo, un subprograma, o un parámetro. Los identificadores son los nombres que utilizamos para representar las variables, constantes, tipos, funciones y etiquetas de nuestro programa. Se crea un identificador especificándolo en la declaración de una variable tipo o función. C y C++ distinguen entre letras mayúsculas y minúsculas, es decir, el compilador las considera como caracteres distintos. El nombre de un identificador de un programa no puede tener la misma ortografía y letras en mayúsculas o en minúsculas que una palabra clave de C.

Los modificadores const y volatile son nuevos en C y C++, fueron añadidos por el C estándar de ANSI para ayudar a identificar a las variables que nunca cambian (const) y a la variable que puede cambiar inesperadamente (volatile).

En la ejecución de un programa el compilador primero lleva acabo las directivas include y define. Cuando el procesador encuentra una directiva define, reemplaza en el programa cada aparición en la primera cadena de caracteres para la segunda cadena de caracteres.

Las declaraciones de funciones en C y C++ comienzan en el prototipo de estas. El prototipo de la función es sencillo, y esta incluido al principio del código del programa para notificar al compilador el tipo y número de argumentos que va usar una función. Esto forza a una verificación de tipo estricta.

Estructuras

Podemos pensar en una estructura como un grupo de variables que pueden ser de diferentes tipos tratadas todas juntas como una unidad. Al igual que sucede con otros tipos de variables

normales C y C++ asignan toda la memoria para los miembros de la estructura. Podemos hacer referencia a los miembros de una estructura usando el operador punto (.). La sintaxis es **nombre_estructura.miembro_estructura** Donde nombre_estructura es una variable asociada con el tipo de estructura y miembro_estructura es cualquier miembro de la misma.

Clases

Las clases son unas de las mayores contribuciones de C++ a la programación. Una definición de clase puede encapsular todas las declaraciones de datos, los valores iniciales y el conjunto de operaciones (llamados métodos) para la abstracción de datos. Los objetos pueden ser declarados para ser de una clase dada y se pueden enviar mensajes a objetos. Cada objeto de una clase específica puede contener su propio conjunto privado y público de datos representativos de esa clase. Una clase puede tener como miembros tanto datos como funciones. Es más, una clase puede incluir partes publicas, privadas y protegidas. Este concepto permiten a determinadas partes, generalmente las variables de la clase, estar ocultas a todas las funciones aquellas que son miembros de la clase.

La definición de una clase comienza con la palabra clave **class**. El nombre de clase o etiqueta de tipo sigue inmediatamente a la palabra clave. Las variables miembro siguen inmediatamente a la declaración de clase. Estas variables, por omisión son privadas para la clase y solo pueden ser accedidas por una función miembro de las que vienen a continuación. Las funciones siguen generalmente a una declaración **public** que permite el acceso a la clase desde funciones externas. Toda función miembro de una clase tiene acceso tanto a la parte publica como a la privada de dicha clase.

Constructores y Destructores

Los métodos constructores y destructores se utilizan para garantizar la inicialización de los datos definidos en un objeto de una clase específica. Al declarar un objeto el constructor de inicialización especificado se aplica. Los destructores desasignan automáticamente el almacenamiento para el objeto asociado cuando se sale del alcance en el que se esta declarado el objeto.

El concepto de ocultación y encapsulamiento de datos implica la denegación del acceso de la estructura interna de un objeto. La sección privada de una clase esta normalmente fuera del alcance de cualquier función externa a la clase. C++ permite declarar otras funciones, aparte de funciones o

clases, que son afines de una clase especificada. La afinidad rompe esta pared, normalmente impenetrable, y permite el acceso de datos y funciones privadas de la clase.

II.4.2 Manejo de Punteros

Cuando se compila un programa en C, la memoria en la computadora se divide en cuatro zonas que contienen el código de programa, toda la información global, la pila y el heap (montón). El montón es un área de memoria libre que se manipula con las funciones de asignación dinámica malloc y free.

Con malloc se asigna un bloque contiguo de almacenamiento al objeto especificado, devolviendo un puntero al comienzo del bloque. El argumento pasado a malloc es un entero sin signo que representa el número de bytes de almacenamiento requeridos. Free libera memoria ya asignada por el montón para que sea reasignada.

Variable Puntero

Para acceder a una variable se puede usar una segunda variable que contenga la dirección de la variable que se desea accesar. Una variable que contiene una dirección, se denomina puntero o variable puntero. Si una variable viene precedida por el operador de dirección &, se accede a la dirección de dicha variable, para acceder al contenido de una dirección la variable debe ser precedida por el operador *.

Un puntero mantiene la dirección de un tipo particular de variables, pero no es en sí mismo uno de los tipos de dato primitivos int, float u otros. Un sistema particular puede permitir a un puntero ser copiado en un variable int, y una variable int podría ser copiada en un puntero. Sin embargo, C no garantiza que los punteros puedan ser almacenados en variables int. Para garantizar la portabilidad del código se debe evitar esta practica. Además, no todas las operaciones aritméticas están permitidas sobre punteros.

El tamaño real de una variable puntero depende de dos cosas: del tamaño del modelo de memoria que se haya elegido para la aplicación, o del uso de las implementaciones especificas y no portables de las palabras clave near, far y huge. Estos modificadores cambian los tamaños de los punteros a los datos. Near tiene una longitud de 2 bytes, far y huge 4 bytes.

Punteros a Punteros

En C podemos definir punteros que apuntan a otras variables punteros, que a su vez, apunta a información. La aparición de los entornos de programación de OS/2, Windows, Windows NT inicia el desarrollo de entornos de operación *multitarea* diseñados para maximizar el uso de memoria. Para compactar el uso de memoria, el *Sistema Operativo* tiene la capacidad de poder mover objetos en memoria siempre que sea necesario. Si un programa apuntara directamente a la celda de memoria física donde esta almacenado el objeto, y el sistema operativo lo mueve, el resultado seria desastroso. En lugar de eso, la aplicación apunta a una dirección de celda de memoria que no cambiará mientras su programa esta ejecutándose (una dirección virtual), y la celda de dirección virtual mientras la dirección física actual del objeto de información, el sistema operativo puede actualizar sin problemas la dirección física actual almacenada en la celda de dirección virtual. En todo lo concerniente a la aplicación, se utiliza la dirección virtual (que en ningún caso es alterada) para apuntar a la dirección física actualizada. Para definir un puntero a un puntero en C, se incrementa simplemente el número de asteriscos que preceden al identificador.

C posee la facilidad para trabajar con el hardware logrando manipular variables puntero, permite realizar solamente dos operaciones aritméticas sobre la dirección de un puntero: suma y resta. Sin embargo, se pueden utilizar otras operaciones. Algunas de estas incluyen:

- Sustracción de un entero a un puntero.
- Sustracción de dos punteros (señalando generalmente al mismo objeto).
- El comparar punteros utilizando un operador relacional como <=, = o >=.

Punteros a Funciones

Podemos también utilizar punteros a porciones de código utilizando un puntero a una función. Los punteros a funciones sirven para el mismo propósito que los puntero o datos, permiten referirse a la función indirectamente, del mismo modo que un puntero a un elemento de datos permite referirse indirectamente a dicho elemento. Un puntero void es un puntero a cualquier tipo de información.

II.4.3 Manejo de Archivos

En la mayoría de los casos una aplicación requiere para su ejecución entrada(s) y/o salida(s), específicamente archivos y no el teclado o el monitor.

II.4.3.1 Creación y Apertura de Archivos

En un programa en C un archivo tiene que estar asociado con un puntero de archivo. El puntero de archivo apunta a información que define varios aspectos de un archivo, incluyendo la vía de acceso al archivo, su nombre y su estado. Un puntero de archivo es una variable de puntero de file, y esta definido en la librería stdio.h.

Al terminar una aplicación, C cierra los archivos automáticamente, sin embargo. La entrada y salida de archivos de disco es ligeramente diferente en C++ y en C. Las facilidades de entrada y salida actúan como un componente de una biblioteca estándar denominada <iostream>. Para realizar operaciones de entrada y salida de archivos, se tiene que utilizar las clases derivadas ifstream y ofstream.

Flujos

Para utilizar las funciones de flujo, la aplicación tiene que incluir stdio.h. Este archivo contiene definiciones para las constantes, tipos y las estructuras utilizadas en las funciones de flujo, así como declaraciones de funciones y definiciones de macros para las rutinas de flujo. Por mencionar algunas: eof esta definido como el valor devuelto al final del archivo y NULL es el puntero nulo. Además file define la estructura utilizada para mantener información acerca de un flujo y bufsize define el tamaño por omisión en bytes, de las memorias intermedias de los flujos.

Podemos utilizar una de las tres siguientes funciones para abrir un flujo antes de que se pueda realizar la entrada y salida con: fopen, fdopen o fropen. El modo y forma del archivo se establece en el momento en que se abre el flujo. El archivo de flujo puede abrirse para leer, escribir, o ambas cosas y puede abrirse ya sea en modo texto o el modo binario. Las tres funciones (fopen, fdopen y freopen) devuelven un puntero de archivo (file), que es utilizado para referirse al flujo. Cuando una aplicación comienza con su ejecución, se abren automáticamente cinco flujos estos flujos son la entrada estándar (stdin), la salida estándar (stdout), el error estándar (stderr), la impresora estándar (stdprn) y el estándar auxiliar (stdaux).Por omisión, la entrada estándar, y salida

estándar y error estándar se refieren a la consola del usuario. Esto significa que siempre que un programa espera algo de la entrada estándar, la recibe desde la consola. Así mismo, un programa que escribe en la salida estándar imprime su información en la consola. Cualquier mensaje de error generado par las rutinas de la biblioteca se envía al flujo de error estándar, es decir, aparecen en la consola del usuario. Los flujos estándar auxiliar y estándar de impresión se refieren generalmente a un puerto auxiliar y a una impresora. Se puede utilizar los cinco punteros de archivo (file) en cualquier función que requiera un puntero de flujo como argumento. Algunas funciones, como getchar y putchar, están diseñadas para utilizar automáticamente stdin o stdeout como los punteros stdin, stdout, stderr, stdprn y stdaux son constantes y no variables, no debe reasignarle un nuevo valor de puntero de flujo.

Las funciones fclose y fcloseall cierra uno o varios flujos. La función fclose cierra un único archivo, mientras que fcloseall cierra todos los flujos abiertos excepto stdin, stdout, stderr, stdprn y stdaux.

II.4.3.2 Lectura y Escritura de un Byte desde un Archivo

Para poder leer o escribir un byte desde un archivo se utilizan las sentencias fread y fwrite siempre y cuando el archivo sea abierto en modo "rb+" (lectura y escritura especificando modo binario).

Para fread tenemos los siguientes parámetros

```
fread(void *Buffer, Tamaño_a_leer, Bytes_leidos, FILE *Archivo_Fuente);
```

Buffer. Indica la estructura donde se van a colocar los datos leídos.

Tamaño_a_Leer. Específica el tamaño en bytes a leer.

Bytes_Leídos. Identifica cuantos bytes se leyeron.

Archivo_Fuente. Identifica el archivo donde se va a leer.

De forma similar para fwrite se tiene:

```
fread(void *Buffer, Tamaño_a_Escribir, Bytes_Escritos, FILE *Archivo_Destino);
```

Buffer. Indica la estructura donde se encuentran los datos a escribir.

Tamaño_a_Escribir. Específica el tamaño en bytes a escribir.

Bytes_Escritos. Identifica cuantos bytes se escribieron con éxito.

Archivo_Destino. Identifica el archivo donde se va a escribir.

Operadores de Bits

Los operadores de bit tratan a las variables como conjuntos de bits más que como números. Son útiles para acceder a los bits individuales en memoria, así como a la memoria de la pantalla para visualizaciones gráficas. Pueden operar solamente sobre tipos de datos discretos, no en números en coma flotante. Tres de los operadores de bit actúan igual que los operadores lógicos, pero sobre cada uno de los bits de un entero: AND, OR y XOR. Un operador adicional es el complemento a uno, que solamente invierte cada bit. AND La operación lógica de AND compara dos bits. Si ambos bits son 1, el resultado es 1. La operación exclusive OR (XOR) compara dos bits y devuelve como resultado 1 solamente cuando los dos bits son complementarios. Esta operación lógica puede ser útil cuando necesitamos complementar posiciones de bits específicas, como ocurre con las aplicaciones gráficas de la computadora.

Existen muchas otras órdenes para manipular bits, por mencionar algunas: a sentencia filelenght devuelve el tamaño del flujo en bytes. Seek se posiciona en el byte especificado de un archivo.

C posee dos operadores de desplazamiento: el desplazamiento a la izquierda, << y el desplazamiento a la derecha, >>. El desplazamiento a la izquierda mueve los bits hacia la izquierda y coloca en el bit más a la derecha (bit menos significativo) un 0. El bit situado más a la izquierda (bit más significativo) se desecha.

II.5 Programación Gráfica en Pascal y C

Uno de los principales rasgos de una computadora son sus posibilidades gráficas. Mediante los gráficos se pueden crear dibujos, diagramas, texto multifuente o cualquier cosa que se pueda dibujar además de manejar una gran cantidad de colores (dependiendo claro, del hardware con que se cuente). Como ya se hizo mención el uso de modos gráficos requiere más trabajo que el uso de modos texto. Se tienen que desarrollar métodos para dibujar líneas y caracteres, tomando en cuenta que deben ser capaces de dibujarlos a escala en la perspectiva adecuada. Afortunadamente, Pascal y C proporcionan un amplio conjunto de rutinas de gráficos que pueden hacer que la programación de gráficos sea fácil, sin dejar de lado la aportación del programador .

II.5.1 Componentes Adicionales para la Programación Gráfica

La programación gráfica, bajo un leguaje de programación (en este caso Pascal o C) requiere varios elementos, si se quiere tomar todas las funciones que proporcione el lenguaje, es posible que éste soporte el manejo de gráficos directamente, es decir, un manejo de gráficos mediante interrupciones directas a memoria, no hay necesidad de utilizar los archivos especiales que el lenguaje tiene en sus versiones de Turbo Pascal y las variantes del lenguaje C (Turbo C y C++), empero este tipo de manejo de gráficos puede, en algún momento, si el programador no tiene los conocimientos suficientes, ser bastante pobre, ya que todas las funciones para manejo de puntos en pantalla, colores, líneas, relleno de áreas, etc. las tiene que realizar el mismo programador, sin embargo si un programador decide tomar este camino, esta programación puede hacer de él un programador experto en gráficos, y desarrollar una lógica de programación mejor e incluso hacer que otros lenguajes que no son gráficos (Como Clipper), puedan serlo (La versión 5.3 de Clipper ya maneja gráficos). En la programación hay un dicho muy común entre los programadores, "Si ya esta hecho , para que volverlo a ser.", parece obvio y aceptable, pero si se conoce el método utilizado se puede sacar más provecho, se hace mención a esto, ya que el presente trabajo utiliza los componentes adicionales de Borland, sin olvidar completamente la programación de acceso directo.

El soporte de Borland para gráficos tiene dos variantes:

- EL *BGI* (Borland Graphics Interface, interfaz gráfico de Borland). Un conjunto completo de herramientas para los gráficos en el entorno DOS.
- EL *GDI* (Graphics Device Interface, interfaz de dispositivo gráfico). La parte del IPA (Interfaz de Programación de Aplicaciones) de Windows que visualiza gráficos.

El programa que se presenta trabaja en ambiente DOS, los archivos de soporte utilizados son del tipo BGI, dichos archivos de soporte contiene en su interior infinidad de funciones para la manipulación de gráficos, los archivos que se obtienen cuando se adquiere algún producto de Borland (Pascal o C), son aun pobres para un manejo de gráficos en color a grande escala.

Los archivos de gráficos BGI que proporciona Borland se muestran en la tabla II.1:

Tabla II. 1 Gráficos BGI de Borland en versiones de Turbo Pascal 7 y C++ v3.1	
Nombre	Modos y No. de colores
ATT.BGI	320x200 CGA PO color que proporciona el CGA 320x200 CGA P1 color que proporciona el CGA 320x200 CGA P2 color que proporciona el CGA 320x200 CGA P3 color que proporciona el CGA 320x200 CGA color que proporciona el CGA 320x400 AT&T
CGA.BGI	320x200x4 CGA CO 320x200x4 CGA C1 320x200x4 CGA C2 320x200x4 CGA C3 640x200x2 CGA 640x480 MCGA
EGAVGA.BGI	EGA 320x200x16 EGA 640x200x16 EGA 640x350x16 VGA 640x480x16 VGA 640x350x16
HERC.BGI	720 x 348 x 2
IBM8514.BGI	640 x 480 x 16 1024 x 768 x 16 en modelos de IBM 8514
PC3270.BGI	720 x 350 x 2 en modelo 3270 PC
VESA16.BGI	800x600 x 16 1024x768 x 16 1280x1024 x 16

Existen archivos complementarios que se pueden adquirir contactando directamente a Borland o mediante *Internet* en la siguiente dirección www.borland.com y se listan en la Tabla II.2.

Tabla II.2 Archivos complementarios proporcionados por Borland.	
Nombre	Modos y No. de Colores
SVGA16.BGI	0) Standard EGA/VGA 320x200x16 1) Standard EGA/VGA 640x200x16 2) Standard EGA/VGA 640x350x16 3) Standard VGA 640x480x16 4) SuperVGA/VESA 800x600x16 5) SuperVGA/VESA 1024x768x16 6) SuperVGA/VESA 1280x1024x16
SVGA256.BGI	0) Standard VGA/MCGA 320x200x256 1) 256k Svga/VESA 640x400x256 2) 512k Svga/VESA 640x480x256 3) 512k Svga/VESA 800x600x256 4) 1024k Svga/VESA 1024x768x256 5) 256k Svga 640x350x256 6) 1280k+ VESA 1280x1024x256

Tabla II.2 (continuación) Archivos complementarios proporcionados por Borland.	
Nombre	Modos y No. de Colores
SVGA32K.BGI	0) 320x200x32768 1) 640x350x32768 2) 640x400x32768 3) 640x480x32768 4) 800x600x32768 5) 1024x768x32768 6) 1280x1024x32768
SVGA64K.BGI	0) 320x200x65536 1) 640x350x65536 2) 640x400x65536 3) 640x480x65536 4) 800x600x65536 5) 1024x768x65536 6) 1280x1024x65536
SVGA16M.BGI	0) 320x200x24-bit 1) 640x350x24-bit 2) 640x400x24-bit 3) 640x480x24-bit 4) 800x600x24-bit 5) 1024x768x24-bit 6) 1280x1024x24-bit

II.5.2 Inicialización del Modo Gráfico

Cuando se quiere activar el modo gráfico es necesario especificar el tipo de adaptador y el modo gráfico y después llamar a initgraph (Función que inicia el modo gráfico). También hay que indicarle a initgraph dónde tiene que buscar el archivo .BGI que corresponde al adaptador. Un archivo .BGI contiene información de un adaptador de gráficos especificado:

initgraph(adaptador, modo, dirección)

La pantalla de gráficos está formada por pixeles ordenados en líneas horizontales y verticales. Esto es válido para todos los modos gráficos de la computadora; la principal diferencia reside en el tamaño del pixel. En el modo de baja resolución de CGA, los pixeles son grandes, por lo que sólo caben 320 horizontalmente y 200 verticalmente. La tarjeta VGA tiene un modo de alta resolución con pixeles tan pequeños que caben 640 horizontalmente y 480 verticalmente, cuanto más pequeño es el pixel, más pixeles hay por imagen y mayor es la calidad de la imagen.

Cada tarjeta tiene uno o varios modos gráficos, y cada modo tiene un sistema de coordenadas asociado. El sistema de coordenadas para los gráficos en baja resolución de CGA es de 320 por 200.

Las coordenadas gráficas comienzan en la posición (0,0), que está situada en la esquina superior izquierda de la pantalla. El pixel de más a la derecha es el número 319 y el de más abajo el 199.

Los sistemas de coordenadas de la pantalla varían de tarjeta en tarjeta y de modo en modo. Por ejemplo, el adaptador CGA soporta modos de alta y baja resolución, y cada uno tiene diferentes sistemas de coordenadas uno es de 320x200 con 4 colores y el otro es de 640x200 con solo 2 colores.

El manejo de imágenes, requiere de una resolución aceptable además del manejo de muchos colores, los archivos que originalmente proporciona Borland ofrecen resoluciones elevadas como el caso de VESA16.BGI que alcanza resoluciones de hasta 1280x 1024, pero con una cantidad pobre de colores, sin embargo los archivos complementarios ofrecen además de buenas resoluciones una gran cantidad de colores, SVGA256.BGI ofrece la misma resolución que el VESA16.BGI, pero con 256 colores, el SVGA32K.BGI con 32k de colores . Etc.

En la Tabla II.3 se proporciona una lista de rutinas de inicialización de modos gráficos:

Tabla II.3 Rutinas de inicialización .

Rutina	Descripción
cleardevice	Limpia la pantalla de gráficos
clearviewport	Limpia la ventana gráfica actual
closegraph	Restaura el monitor de Video al modo que se encontraba antes de entrar en el modo gráfico. También libera la memoria usada por el sistema de gráficos.
detectgraph	Devuelve el controlador de gráficos detectado y el modo gráfico de la tarjeta de gráficos de la computadora.
getdrivername	Devuelve el nombre del controlador de gráficos actual.
getgraphmode	Devuelve el modo gráfico actual. El valor numérico del modo gráfico se tiene que interpretar conjuntamente con la información sobre el controlador de gráficos usado.
getmaxx	Devuelve la máxima coordenada horizontal del modo gráfico actual.
getmaxy	Devuelve la máxima coordenada vertical del modo gráfico actual.
grapherromsg	Devuelve el mensaje de error correspondiente a la condición de error indicada por Código.
graphresult	Devuelve un código de error relativo al último procedimiento de gráficos utilizado.
initgraph	Inicializa el entorno de gráficos para el controlador gráfico. El procedimiento busca los archivos. BGI.
installuserdriver	Instala un controlador gráfico que no sea de los que proporciona Borland en sus productos.
registerbgidriver	Permite al usuario cargar un archivo controlador. BGI (leído desde el disco al montón o enlazado con el programa mediante BINOBJ) y registrar el controlador con el sistema de gráficos.
restorecrtmode	Restaura el monitor de vídeo al modo que tenía antes de iniciar los gráficos.

II.5.3 Lectura y Escritura de un Punto

Para escribir un punto en pantalla utilizando una programación directa, es necesario hacer una referencia a la dirección de memoria donde se quiere colocar el punto, se debe tomar en cuenta que la memoria no es una matriz de direcciones, es decir, sus direcciones no son de tipo matriz, son lineales y cuando se hace referencia a alguna dirección se toma un segmento y un desplazamiento, por lo que una coordenada (x,y) que se desee graficar se debe convertir a una dirección lineal, y no se debe manejar como matriz, para esto es necesario conocer la resolución a la cual se está trabajando. Por ejemplo :

Colocar un punto en las coordenadas (160,100) de la pantalla gráfica de resolución de (320,200) (MCGA a 256 colores), tome en cuenta que cuando se refiera a esta resolución, se hablará de una coordenada máxima de (319,199) ya que para efectos de conteo la coordenada (0,0) también se toma en cuenta. Primero es necesario conocer la resolución a la cual se trabajara en este caso (320,200), después localizar la dirección de memoria en donde comienza el modo MCGA (320x200x256), esta dirección se encuentra en (A000:0000). Donde A000 es el segmento de Memoria y 0000 es el desplazamiento y por ultimo asignarle a esta dirección un color (Byte).

Si ponemos un punto en la dirección A000:0000, en coordenadas cartesianas seria (0,0), para la dirección A000:0001 la coordenada seria (1,0), para A000:0002 seria (2,0) y para A000:F9FF seria (319,199), fue un salto grande eh!, pero como es que sale esta coordenada, observe que solo hacemos referencia al desplazamiento y no al segmento, entonces si F9FF es el desplazamiento la operación para obtener las coordenadas cartesianas se hace con la siguiente operación:

$$(y * 320) + x = \text{Desplazamiento}$$

donde $x = \text{Desplazamiento} - (y * 320)$

y $y = (\text{Desplazamiento} - x) / 320$

Para Desplazamiento = F9FF (Hexadecimal) = 63999 (Decimal)

$$(199 * 320) + 319 = 63999$$

$x = 63999 - (y * 320)$ para cualquier valor de y;

$y = (63999 - x) / 320$ para cualquier valor de x .

Por esto se dice que esta resolución tiene 64k de memoria de video, es decir, desde la dirección 0000 hasta F9FF (0 a 63999) existen 64000 posibles direcciones de memoria (Figura II.5).

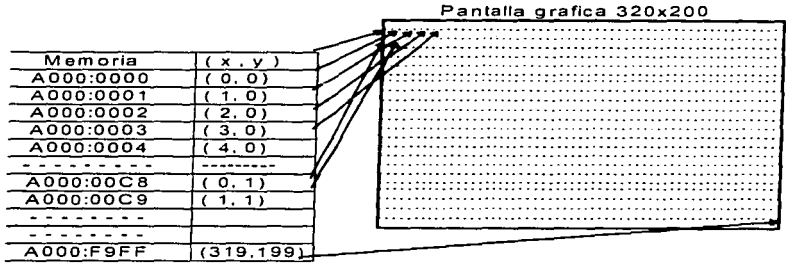


Figura II.5 Pantalla gráfica y su relación con la memoria de video.

En el ejemplo; si x=160, y=100 entonces (y*320)+x=32160 =7DA0, la formula es sencilla pero puede causar confusión.

Una vez explicado esto, falta mencionar como se envía el byte a la dirección de memoria.

Utilizaremos la palabra MEM(Segmento: Desplazamiento) para hacer referencia a una dirección en memoria ya que Pascal y C utilizan funciones diferentes ("mem" y "char far dirección" respectivamente).

Si asignamos un byte (color) a una dirección, se indica la dirección, el desplazamiento y el valor del color que se desee:

MEM(A000:7DA0)=1

donde el color deseado es el azul, o

MEM(A000:(y*320)+x)=1

donde x=160, y=100

La instrucción es sencilla pero el entender y armar esta función, se lleva un poco de tiempo, en cambio, si deseamos poner un punto en pantalla con las funciones de los archivos BGI, pondríamos la siguiente instrucción

putpixel (x,y,color);

Donde x=160, y=100 y color=1;

Es mas sencillo, ya que la instrucción se encarga de direccionar el punto a la memoria y no el programador como en el caso anterior.

Al igual que en escritura, para la lectura, es necesario conocer en que dirección se encuentra el punto a leer, la formula que se analizo anteriormente servirá para realizar la operación de lectura, nuevamente utilizaremos MEM(), pero ahora el valor que este en la dirección especificada será asignado a una variable, por ejemplo :

Leer el byte que esta en la coordenada 160,100; primero se calcula la conversión a forma lineal y tenemos:

Variable = MEM (A000 : (y * 320) + x)

donde x=160, y=100

Utilizando las funciones :

Variable = getpixel(y,x)

donde x=160, y=100

El escribir o leer un punto en pantalla sin utilizar las funciones de los BGI implica, como se pudo observar, conocer mas acerca de nuestro equipo y tener mas tiempo para realizar un programa ya que no es solo utilizar una dirección de memoria, debemos también usar algunas funciones de interrupción al bios como en el caso de la inicialización, operaciones matemáticas complicadas, etc., siendo una de las razones principales por las que se decidió utilizar las funciones que proporciona Borland y no utilizar las de acceso directo, aunque no las olvidaremos del todo.

La siguiente lista de rutinas, son para poder dibujar líneas, puntos, etc. en pantalla:

Rutina	Descripción
bar	Dibuja en la pantalla un área rectangular rellena, según el color de relleno actual.
circle	Dibuja un círculo con un determinado Radio alrededor de unas coordenadas X, Y.
drawpoly	Dibuja un polígono. Un array contiene las coordenadas de los vértices del polígono.
getpixel	Devuelve el color del pixel de unas coordenadas X,Y.
getx	Devuelve la coordenada horizontal de la posición actual.
gety	Devuelve la coordenada vertical de la posición actual.
line	Dibuja una línea de un punto a otro.
pieslice	Dibuja un trozo de diagrama de tarta (de pastel).
putpixel	Dibuja un punto en una posición en pantalla.
rectangle	Dibuja en la pantalla un área rectangular sin relleno.
Setwritemode	Selecciona uno de los dos modos de dibujo de líneas (MOV y XOR).

II.5.4 Almacenar Una Imagen

Cuando trabajamos con imágenes, se requiere del acceso a dos posibles arreas: el área de memoria y el área de disco, con la primera se tiene un manejo mas rápido con pequeños bloques de información, la segunda por otra parte tiene la capacidad para almacenar estos pequeños bloques e incluso bloques mas grandes que en memoria no son tan manejables como se desearía. Si el programador domina estas dos arreas y además utiliza métodos de computación, podrá hacer de la programación gráfica una poderosa herramienta para realizar aplicaciones que no le piden nada a las que actualmente están en el mercado.

Para guardar una imagen en disco se debe saber, como la queremos guardar, es decir, con algún tipo de formato gráfico, byte por byte, o con funciones de los BGI. La primera opción será analizada con detalle en los capítulos III y IV, las otras se analizan a continuación.

Byte por Byte

Si deseamos guardar una imagen byte por byte, debemos considerar algo importante, el tamaño de la imagen, el cual se debe conocer y tener presente siempre. Al igual que la memoria, el acceso a disco es de forma lineal (Si hablamos de archivos, sin tipo en pascal y binarios en C), si no se conocen las dimensiones de la imagen los puntos en pantalla serán colocados linealmente, si el programador conoce este tamaño, él mismo se encarga de controlar la colocación de esos puntos.

Dibujar líneas y circulos aleatorios y guardar en disco el arrea cuyas coordenadas son (0,0)-(160,100).

- Iniciar el modo Gráfico.
- Generar la rutina que realiza el dibujo.
- Crear un archivo donde se guardara la imagen.
- Leer de pantalla cada uno de los puntos y al mismo tiempo almacenarlos en el archivo.
- Si deseamos que la paleta de colores actual también se guarde, debemos generar una rutina para obtener dicha paleta de la memoria en un arreglo y guardarlo en el archivo.

Con Funciones de los BGI

- Iniciar el modo Gráfico.
- Generar la rutina que realiza el dibujo.

- Crear un archivo donde se guardara la imagen.
- Utilizar las funciones para almacenar esa imagen en un puntero (imagesize, getmem, getimage).

  Determinar cuánta memoria se va a necesitar para almacenar la Imagen gráfica.

  Reservar un buffer de ese tamaño.

  Salvar la imagen en el buffer.
- Escribir en el archivo el puntero (Como un bloque de bytes).
- Cerrar el archivo.

II.5.5 Lectura y Visualización de Una Imagen

Tomando como base los métodos de almacenamiento mencionados anteriormente, así mismo haremos una lectura byte a byte o con funciones que proporcionan los archivos BGI de Borland.

Byte por Byte

Si queremos leer una imagen byte a byte debemos de tomar en cuenta varios puntos:

- Tamaño de la imagen (horizontal y vertical)
- Que tamaño en color ocupa la imagen (2, 4, 256, 32k,64K,16M de colores)
- Si en el interior fue almacenada la paleta de colores y si es así conocer el tamaño de la misma.
- Si no fue almacenada la paleta de colores la imagen tomará los colores de la paleta actual.

Si algunos de estos datos falta conocer es probable que la imagen que se visualice no sea la esperada.

- Iniciar el modo Gráfico.
- Abrir el archivo donde se leerá la imagen.
- Leer del archivo la paleta de colores, si es que esta almacenada en el archivo, y cargarla en la memoria
- Leer del archivo byte por byte los puntos y al mismo tiempo colocarlos en pantalla.
- Cerrar el archivo.

Con Funciones de los BGI

- Iniciar el modo Gráfico.

- Abrir el archivo donde se encuentra la imagen.
- Preparar el puntero para almacenarle un bloque de información (imagesize, getmem).

  Tamaño de la imagen

  Reservar el tamaño de memoria a ocupar
- Leer del archivo un bloque de bytes y almacenarlos en el puntero.
- Posicionar en pantalla la imagen (Putimage)

  Indicando las coordenadas donde se quiere colocar y método de visualización (AND, OR, XOR).
- Cerrar el archivo

CAPITULO

II

CAPITULO III

ANÁLISIS DESCRIPTIVO

III.1 Formatos de Archivos Gráficos

Los archivos gráficos (Graphics File) son estructurados de acuerdo a las convenciones de un formato específico, los cuales son (o deben ser) documentados con las especificaciones del formato, escritos y revisados por el creador del formato. Sin embargo, no todos los formatos gráficos son documentados, por diversas razones: el creador del formato ha emigrado, el formato ha sido vendido a otra organización, la organización que lo creó ya no lo distribuye, etc.

Un formato de archivo gráfico (*Graphic File Format*) es aquel formato en el cual los datos gráficos, es decir, los datos que describen una imagen, son almacenados en un archivo. Los formatos de archivos gráficos surgen como una necesidad de almacenar, organizar y recuperar dichos datos gráficos de una manera eficiente y lógica.

III.1.1 Antecedentes

Un formato de archivo gráfico puede ser complejo, claro que nunca parece complejo hasta que se trata de implementarlo en un determinado Software. La forma en que se almacenan los datos gráficos indican la velocidad con que serán leídos, el espacio que ocupará y la facilidad con que pueden ser accesados por una aplicación. Un programa simple debe guardar sus datos en un formato razonable. De otro modo, corre el riesgo de ser considerado como inútil.

Prácticamente cada aplicación crea y almacena de alguna forma sus datos. Igual que los más simples editores de texto crean archivos que contienen dibujos de línea hechos de caracteres ASCII o secuencias de escape. Las aplicaciones basadas en GUI (*Graphical User Interface*), el cual ha proliferado recientemente, ahora requieren el soporte para formatos híbridos, que permiten la incorporación de datos bitmap en documentos de texto. Los programas de bases de datos (*Database*) con extensiones de imágenes también permiten almacenar texto junto con datos bitmap en un archivo. Por lo cual, los

archivos gráficos son un importante mecanismo de transporte que permite el intercambio de datos visuales entre aplicaciones y computadoras.

Un gráfico indica la producción de una representación visual de un objeto real o imaginario creado por métodos conocidos como escritura, dibujo, impresión y grabado. El resultado final de un proceso de producción de gráficos tradicional eventualmente aparece en una superficie de 2 dimensiones, tal como papel o lona. Los gráficos de computadora, sin embargo, han expandido el significado de gráficos para incluir algunos datos para su despliegue en algún dispositivo de salida (pantalla, impresora, *plotter*, cinta de filmar o video cinta, por mencionar algunos).

Los gráficos se refieren a una salida actual como "alguna cosa" que puede verse, ahora solo significa "algo" intencional para desplegar o ser puesto en una salida. Esta distinción puede parecer tonta para usuarios experimentados, sin embargo, hay artistas que han tenido problema con ello.

En la práctica los gráficos de computadora y la creación de un trabajo están a veces por separado de su representación. Una forma de ponerlo, es que un proceso de gráfico de computadora se produce una "salida virtual" en memoria, o una salida en un archivo permanente, por ejemplo disco o cinta. En otras palabras, aunque un programa ha escrito un archivo de "algo", la salida no existe aún desde un punto de vista tradicional ya que nada ha sido desplegado en algún dispositivo. Así que, decimos que los datos gráficos son la "salida virtual" de un programa, porque una representación puede ser construida y/o reconstruida desde datos gráficos guardados a un archivo, posiblemente por el mismo programa.

La mayoría de la gente hace la distinción entre la creación y la interpretación (*rendering*). Tradicionalmente un imagen es una representación visual de un objeto del mundo real, capturado por un artista por medio de alguna clase de proceso mecánico, electrónico o fotográfico. En un gráfico de computadora, el significado de una imagen ha sido ampliado para referir a un objeto que aparece en algún dispositivo de salida. Los datos gráficos son interpretados cuando un programa "dibuja" una imagen en algún dispositivo de salida. Se ha oído hablar de los gráficos de computadora "*production pipeline*" que, consiste en una serie de pasos referentes a la definición y creación de datos gráficos y dar la imagen. La figura III.1 ilustra este proceso.

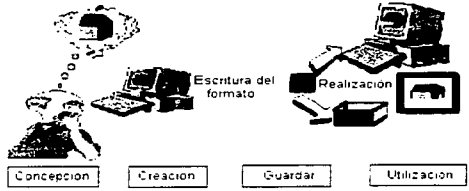


Figura III.1 Producción Pipeline.

III.1.2 Archivos Gráficos

Un archivo gráfico es aquel archivo que almacena algún tipo de dato gráfico (el opuesto a texto, hoja de cálculo, datos numéricos, por ejemplo) para su realización y despliegue. Los diversos modos en que estos archivos están estructurados son conocidos como Formatos de Archivos Gráficos. El estudio de algunos de ellos es el objetivo del presente trabajo.

Cuando una imagen es enviada a un archivo, el contenido de ese archivo se convierte en datos gráficos, tales datos son una vez más enviados al proceso de interpretación del archivo para poder visualizarla, aunque, siguen siendo meramente datos. De hecho, el dato puede ser de un tipo diferente, esto es lo que sucede en las operaciones de conversión de archivos: un archivo de formato tipo 1 es enviada (por un programa de conversión) a un segundo archivo de formato tipo 2.

III.1.3 Tipos de Datos Gráficos

Un dato gráfico tradicionalmente es dividido en tres clases: *Vector*, Bitmap y *Objeto*. Los cuales se explican a continuación:

Vector

En gráficos de computadora un vector significa la representación de líneas, polígonos o curvas (o algún objeto que pueda ser fácilmente dibujado con líneas) por puntos clave (*key points*) especificados numéricamente. El trabajo de un programa es enviar estos datos de puntos clave para regenerar las líneas por conexión de los puntos o dibujo usando los puntos como guía. Asociado con el vector de puntos viene información de los atributos (tales como color y ancho de la línea de información) y un juego de convenciones o reglas que permiten al programa dibujar el objeto deseado. Estas convenciones pueden ser implícitas o explícitas, y aunque diseñadas para cumplir los mismos objetivos, son generalmente diferentes de un programa a otro. La figura III.2 muestra ejemplos de datos vector.

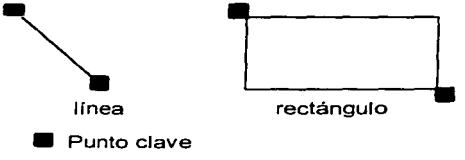


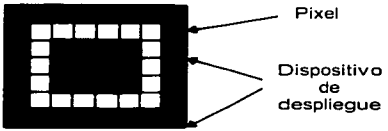
Figura III.2 Vector.

En ciencias y matemáticas un vector es un línea recta que tiene magnitud y dirección. En gráficos de computadora vector es un término que toma de todo, puede ser casi un tipo de línea o

segmento de línea, usualmente es especificado como un juego puntos extremos, excepto en el caso de líneas curvas y figuras geométricas más complicadas, las cuales requieren otros puntos clave para ser completamente especificadas.

Bitmap

Un bitmap está formado de un juego de valores numéricos especificando los colores de pixel individuales o *pels* (picture elements). Un bitmap es un arreglo de pixeles (ver figura III.3), aunque, un bitmap técnicamente consiste de un arreglo de valores numéricos usados para colocar o "encender" el pixel correspondiente en un dispositivo de salida donde el bitmap es enviado. Se utiliza el término valor de pixel pata referir al valor numérico en el dato bitmap correspondiente al color de pixel en la imagen en el dispositivo de despliegue.



Dato bitmap en archivo:
0, 0, 0, 0, 0, 0, 0, 0,0, 255,255, 255, 255,0,...

Figura III.3 Bitmap.

Anteriormente el término bitmap era usado en ocasiones para referir a un arreglo (o mapa) de bits sencillos, cada bit era correspondiente a un pixel, mientras el término *pixelmap*, *graymap* y *pixmap* estaban reservados para arreglos de pixeles *multibit*. Tomaremos el término bitmap para referir un arreglo de pixeles, de cualquier tipo que especifica el ancho de bit (*bit depth)* o ancho de pixel (*pixel depth)*, que es el tamaño del pixel en bits o en alguna otra unidad conveniente (como bytes). El ancho de bit determina el número de colores que un valor de pixel puede representar. Un pixel de 1 bit puede ser uno de 2 colores, un pixel de 4 bits, uno de 16 colores, y así sucesivamente. Los valores más comunes son: 1, 2, 4, 8, 16, 24 y 32 bits.

Fuente del Bitmap: Dispositivos de Rastreo

El término rastreo *(raster)* ha sido asociado con la tecnología del tubo de rayos catódicos (CRT, Cathode Ray Tube) referido al patrón de líneas que el dispositivo hace cuando despliega una imagen. Las imágenes con formato de rastreo son por lo tanto una colección de pixeles organizados en series de líneas escaneadas (*scan line*). Ya que los dispositivos de despliegue por rastreo son el tipo más popular disponibles actualmente, el despliegue de imágenes con patrones de pixeles, los valores de pixeles son generalmente organizados para su fácil despliegue como datos de rastreo por lo que también son conocidos con este nombre.

Otras fuentes de datos bitmap son los dispositivos de rastreo usados para trabajar con imágenes los cuales son: scanners, vídeo cámaras y otros dispositivos digitalizadores. Cuando se habla de datos gráficos capturados de un dispositivos de rastreo se habla de una imagen bitmap.

Objeto

Anteriormente un objeto implicaba un método para diseñar figuras complejas, tales como polígonos anidados, por medio de un método de notación tipo taquigrafía para enviar dichas formas con un mínimo de datos. Actualmente un objeto indica un dato almacenado y acompañado del código de programa o información del algoritmo requerido para enviarlo. Esta distinción se hace clara en la programación orientada a objetos. La figura III.4 muestra un ejemplo de objeto.

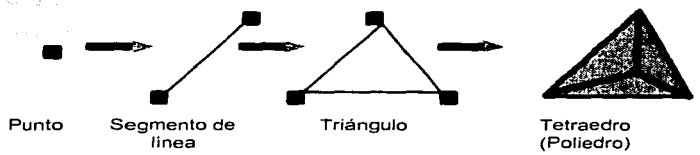


Figura III.4 Objeto.

En los archivos gráficos se incluyen datos que contienen la estructura, color y alguna otra información descriptiva. Esta información está incluida principalmente como una ayuda para el envío, reconstrucción y despliegue de la imagen.

Hace 25 años, los gráficos de computadora eran guardados en vectores y desplegados en dispositivos basados en vector Random-scan, actualmente son almacenados en bitmap y los dispositivos de despliegues son basados en rastreo. Los gráficos bitmap son importantes en aplicaciones CAD (Computer Asisted Design, Diseño Asistido por Computadora), envío en 3D, modelado en 2 y 3 dimensiones, Arte por computadora, animación, interface de usuario gráfica, vídeo juegos, procesamiento de imágenes en documentos electrónicos (Electronic Document Image Processing, EDIP), procesamiento y análisis de imágenes.

El crecimiento del World Wide Web ha ayudado al incremento del uso de bitmap ya que casi toda página Web tienen uno o más archivo bitmap asociados a el.

III.1.4 Tipos de Formatos de Archivos Gráficos

Existen gran variedad de tipos de formatos de archivos gráficos. Cada tipo almacena datos gráficos de diferente manera. Formatos bitmap, vector y *metafile* son los más utilizados, sin embargo hay otras clases tales como escena, animación, multimedia, híbrido, hypertexto, hypermedia, 3D, VRML, audio, fuente y PDL. El incremento de la popularidad del Web ha hecho de algunos de estos formatos los más comunes. A continuación se hace un breve bosquejo de ellos.

Formato Bitmap (Bitmap Format)

Son utilizados para almacenar datos de un bitmap. Este tipo de archivos están hechos el uno para el otro en el almacenamiento de imágenes reales, como de fotografías y vídeo. Son también conocidos como archivos de rastreo (raster files), esencialmente contienen un mapeo exacto pixel por pixel de una imagen. Una aplicación de envío consecuentemente puede reconstruir dicha imagen en un dispositivo de salida.

Microsoft BMP, PCX, TIFF y TGA son ejemplos de formatos bitmap.

Formato Vector (Vector Format)

Son útiles para almacenar elementos basados en líneas, como líneas y polígonos, o que pueden ser descompuestos en objetos geométricos simples. Contienen descripciones matemáticas de elementos de una imagen. Una aplicación de envío usa estas descripciones de formas gráficas para construir una imagen final. En general están estructurados de forma más simple que los archivos bitmap y están organizados como cadenas de datos.

AutoCAD DCF y Microsoft SYLK son ejemplos de formatos Vector.

Formato Metafile (Metafile Format)

Pueden guardar tanto datos de un bitmap como de un vector en un solo archivo. Los archivos metafile más simples son parecidos a los archivos vector; mejoran un lenguaje o gramática que puede ser usada para definir elementos de un vector, pero también almacena una representación de una imagen bitmap. Se utilizan para transportar datos de un vector y un bitmap entre diferentes plataformas de hardware, o mover datos de una imagen entre plataformas de Software.

WPG, Macintosh PICT y CGM son ejemplos de formatos metafile.

Formato Escena (Scene Format)

Conocidos también como archivos de descripción de escena (scene description files), están diseñados para almacenar una representación condensada de una imagen o escena, que es usada por un programa para reconstruir la imagen actual. Su diferencia con el formato vector es que éstos últimos proporcionan la descripción de parte de una imagen y los archivos escena contienen instrucciones que el programa de envío usa para construir la imagen.

Autodesk 3D Studio, DKB y NFF son ejemplos de formatos Escena.

Formato de Animación (Animation Format)

La idea básica de éste formato es hojear un libro como un niño, con estos libros, se despliega rápidamente una imagen superpuesta sobre otra dando la impresión que los objetos en la imagen están en movimiento. Los formatos de animación muy primitivos almacenaban imágenes enteras que eran desplegadas en secuencia, usualmente en un ciclo. Los formatos un poco más avanzados almacenan una sola imagen con múltiples paletas de colores para la imagen. Cargando una nueva paleta de colores, los colores en la imagen cambian y los objetos parecen moverse. Los formatos de animación avanzada almacenan solo las diferencias entre dos dibujos adyacentes (llamados *frames*) y actualiza solo los pixeles que han de ser cambiados en cada frame desplegado. Un radio de despliegue de entre 10 a 15 frames por segundo es común en una animación de caricatura. Una animación de vídeo usualmente requiere desplegar en un radio de 20 frames por segundo o más para un movimiento más real.

TDDD y TTDD son ejemplos de formatos de animación.

Formato de Multimedia (Multimedia Format)

Son relativamente nuevos pero han ido tomando gran importancia. Almacenan datos de diferentes tipos en un mismo archivo. Usualmente permiten la inclusión de información de gráficos, audio y vídeo.

Microsoft´s RIFF, Apple´s QuickTime, Autodesk´s FLI y MPEG son ejemplos de formatos de multimedia.

Formato Híbrido (Hybrid Format)

Ha conducido la búsqueda en la integración de texto no estructurado y bitmap ("*hybrid text*") y la integración de información basada en récords y bitmap ("*hybrid database*").

Formato de Hypertexto e Hypermedia (Animation Format)

El hypertexto es una estrategia para permitir acceso no lineal a información. En contraste, la mayoría de los libros son lineales, teniendo un inicio y un fin, y un patrón de progreso a través del texto definido. El hypertexto, sin embargo, habilita documentos para ser construidos con uno o más inicios o múltiples finales; y con algunos vínculos de hypertexto que permiten a los usuarios saltar a algún lugar disponible en el documento al que ellos quieren ir.

Los lenguajes de hypertexto no son archivos de formato gráfico, como los formatos GIF o DXF. En cambio lenguajes de programación de hypertexto, como Postcript o C, están diseñados específicamente para la transmisión de datos seriales. Se puede iniciar codificando una cadena de información de hypertexto como reciben los datos, no necesita esperar el documento completo de hypertexto para empezar a cargarlo.

El término hypermedia se refiere a la unión de hypertexto y multimedia. Lenguajes de hypertexto modernos y protocolos de red soportan una amplia variedad de media, incluyendo texto y fuentes, datos de gráficos animados, audio, vídeo y 3D. El hypertexto permite la creación de una estructura que habilita datos multimedia para ser organizados, desplegados e interlazados siendo navegado por un usuario de computadora.

GIF, JPEG, MPEG, AVI y Postcript son ejemplos de formatos Hypertexto e Hypermedia, utilizados principalmente en las páginas Web.

Formato 3D (3D Format)

Almacenan descripciones de una forma y color de modelos 3D de objetos del mundo real o imaginario. Los modelos 3D están construidos de polígonos y superficies lisas, combinadas con descripciones de elementos relacionados, como color, textura, reflexiones, entre otras, que una aplicación de envío (generalmente programas de modelado y animación, NewTek´s Lightwave y Autodesk´s 3DStudio) pueden usar para reconstruir un objeto. Los modelos son colocados en escenas con luces y cámaras, por lo que también son conocidos como elementos de escena.

Formato VRLM (VRLM (Virtual Reality Modeling Language) Format)

Pronunciado como "vermel" se considera como un híbrido de gráficos 3D y HTML. VRML v1.0 es esencialmente un subjuego del archivo de formato Silicon Graphics añadiendo soporte para entrelazar a URLs (Uniform Resource Locators) en un Web.

VRML codifica datos 3D en un formato conveniente para intercambio a través de Internet usando el protocolo de transferencia de hypertexto (HTTP, Hypertext Transfer Protocol). Los datos VRLM recibidos de un servidor Web son desplegados en un browser que soporta la interpretación del lenguaje VRLM.

Formato de Audio (Audio Format)

Generalmente son almacenados en cintas magnéticas como datos analógicos. Un dato de audio para ser almacenado en CD-ROM o disco duro debe ser primero codificado usando un proceso de muestreo digital empleado para almacenar datos de vídeo digital. En cada codificación el dato de audio puede entonces ser escrito al disco como una cadena de datos de audio digital, o almacenado utilizando un formato de audio.

Los archivos de formato de audio son idénticos en concepto a los archivos de formato gráfico, excepto que los datos almacenados son enviados al oído. La mayoría de los formatos contienen una simple cabecera (header) que describe al dato de audio que contiene. La información almacenada en la cabecera de un formato de archivo de audio incluye muestras por segundo, número de canales y número de bits por muestra.

Los métodos de compresión que usan difieren en relación a los datos gráficos. Utilizan el Huffman de 8 bits. Los datos de 16 bits requieren algoritmos adaptados especialmente para la compresión de audio que incluyen las recomendaciones G.711(uLAW), G.721 (ADPCM 32), G.723 (ADPCM 24) del *CCITT* (International Telegraph Consultative Committee) y FIPS-1016 (CELP), FIPS-1015 (LCP-10E) del estándar de U.S.

Formato de Fuente (Font Format)

Contienen la descripción de juegos de caracteres alfanuméricos y símbolos compactados en un formato de fácil acceso. Están diseñados para facilitar el acceso aleatorio a los datos asociados con caracteres individuales. En este sentido son bases de datos de información de caracteres o símbolos, y por esta razón en ocasiones son usados para almacenar datos gráficos que no son alfanuméricos o simbólicos en esencia.

Algunos archivos de fuente soportan compresión y otros *encriptación* de datos de caracter. Existen tres tipos principales de archivos fuente: bitmap, *stroke* y contornos *spline-based*, los cuales se describen brevemente a continuación.

- Fuentes Bitmap. Consiste serie de imágenes de caracter enviados en pequeños bitmap rectangulares y almacenados secuencialmente en un archivo. Puede o no tener cabecera, la mayoría son monocromáticos y almacenan fuentes en tamaños rectangulares uniformes para facilitar la velocidad de acceso; haciendo de ello una ventaja además de su fácil manejo.
- Fuentes Stroke. Son bases de datos de caracteres almacenados en forma de vectores. Los caracteres pueden consistir de un solo stroke. Un caracter stroke generalmente consiste de una lista de fin de puntos para ser dibujados secuencialmente. Tienen como ventaja que pueden ser escalados y rotados fácilmente, y están compuestos por primitivas (líneas y arcos). No es posible representar caracteres de alta calidad.
- Fuentes contornos spline-based. La descripción de caracteres están compuestas de un control de puntos que permiten la reconstrucción de primitivas geométricas conocidas como splines acompañadas de información usada en la reconstrucción de los caracteres. son usadas para crear representaciones de caracteres de alta calidad, que pueden ser rotados, escalados y otras manipulaciones.

Formato PDL (PDL (Page Description Language) Format)

Son lenguajes de computadora actuales usados para describir la composición, información de fuente y gráficos de páginas impresas y desplegadas. PDL es usado como lenguaje de interpretación para comunicar información a dispositivos de impresión (impresoras) o despliegue (GUI). Es un dispositivo dependiente.

Pueden contener información gráfica, aunque no es considerado como un archivo de formato gráfico, se puede decir que es un modulo de código C que contiene un arreglo de información gráfica para ser un archivo de formato gráfico. Son lenguajes de programación completos, que requieren el uso de intérpretes sofisticados para leer sus datos; son bastante diferentes de muchos de los más simples *parse* usados para leer archivos de formato gráfico.

III.1.5 Elementos de un Archivo Gráfico

Las diferentes especificaciones de los archivos de gráficos usan una diversa terminología. En realidad, es posible tener un significado común a través de los formatos que en este trabajo se van a analizar. La clasificación que utilizaremos es la siguiente:

- Campo. Un campo es una estructura de datos con un tamaño fijo en un archivo gráfico. Un campo fijo tiene no solo un tamaño fijo si no una posición fija dentro del archivo. La localización del campo es especificada con un desplazamiento desde un lugar en el archivo, tal como el inicio o fin, o un desplazamiento de algún otro dato. El tamaño del campo está situado en la especificación del formato o puede ser deducida desde otra información.
- Etiquetas. Una etiqueta es una estructura de datos que puede variar en tamaño y posición de archivo en archivo. La posición de una etiqueta, tal como un campo, es especificada por un desplazamiento desde un lugar en el archivo, o un desplazamiento relativo desde otro elemento del archivo. Las etiquetas por sí mismas pueden contener otras etiquetas o colección de campos relacionados.
- Cadenas. Los campos y etiquetas son una ayuda de acceso aleatorio, están diseñadas para ayudar al programa a un acceso rápido a datos conocidos. Una vez que una posición en un archivo es conocida, un programa puede accesar a esa posición directamente sin tener que leer datos intermedios. Un archivo organiza los datos como una cadena, de otra manera, pierde la estructura de un archivo organizado por campos y etiquetas para crear paquetes, que pueden variar en tamaño, son subelementos de la cadena, y son significativos para la lectura del programa del archivo. El inicio y fin de una cadena puede ser o no conocidos y especificados.

III.1.6 Conversión de Formatos

En ocasiones es necesario convertir archivos gráficos de un formato a otro, para imprimir o manipular con un Software específico, por ejemplo. Aunque la conversión puede ser sencilla o complicada: existen conversiones realmente problemáticas si necesita convertir entre tipos de formato básicos, bitmap a vector, por ejemplo.

Afortunadamente existen excelentes productos que pueden manipular la conversión entre formatos, por mencionar algunos tenemos: pbmplus (Protable Bitmap Utilities) en ambiente UNIX, HiJaak de Inset Systems (ahora Quarterdeck) para ambiente MS-DOS y Windows, por último para ambiente Macintosh DeBabelizer.

Compresión de Datos

La compresión de datos (o codificación de datos) es el proceso usado para la reducción de espacio físico de un bloque de información. En la compresión de datos gráficos, podemos guardar

más información en un espacio de almacenamiento físico. Ya que las imágenes gráficas usualmente requieren una gran cantidad de espacio para ser almacenadas, la compresión es una importante consideración para los archivos de formato gráfico. Casi cada formato de archivo gráfico emplea algún tipo de método de compresión.

Hay diversos modos de ver la compresión, podemos hablar de las diferencias entre la compresión física y lógica, simétrica y asimétrica, por mencionar algunas. A continuación se mencionan brevemente los métodos o algoritmos más comunes para la compresión, algunos de los cuales se detallarán más adelante en el análisis de los formatos que el presente trabajo estudia.

- Pixel packing (Empaquetamiento de Pixeles). No es un método de compresión en sí, pero es una forma eficiente de almacenar datos en bytes de memoria contiguos. Es usado por el formato PICT de Macintosh y otros formatos capaces de almacenamiento múltiple 1, 2 ó 4 bits por pixel por byte de memoria o espacio en disco.
- Run-Length Encoding (RLE). Es un algoritmo muy común de compresión usado por formatos bitmap tales como BMP, TGA y PCX para reducir la cantidad de datos gráficos redundantes.
- Lempel-Ziv-Welch (LZW). Es usado por GIF y TIFF, este algoritmo es también parte de la compresión estándar de módem v.42 bits y de Postcript Nivel 2.
- Codificación CCITT. Una forma de compresión de datos para transmisión de *facsimil* estandarizada por CCITT. Un estándar particular basado un esquema de compresión por llaves introducido por David Huffman conocido como codificación Huffman.
- Joint Photographic Experts Group (JPEG). Una herramienta de método de compresión utilizada particularmente para imágenes de datos con tonos continuos y multimedia. La línea base de implementación del JPEG usa un esquema de codificación basado en el algoritmo Discrete Cosine Transform (*DCT*).
- Joint Bi-level Image Experts Group (JBIG). Un método de compresión de datos de imagen bi-level (2 colores), que está destinado para reemplazar los algoritmos de compresión Modified Read (MR) y Modified Modified Read (MMR) usados por CCITT Grupo 3 y Grupo 4.
- ART. Un algoritmo de compresión desarrollado por Johnson-Grace que puede ser adaptado para soportar audio, animación, y movimiento de vídeo completo en un futuro.

Orden de Byte

Generalmente pensamos que la información en memoria o disco está siendo organizada en series de bytes individuales de datos. El dato es leído secuencialmente en el orden en que son almacenados. Este tipo de datos es llamado dato orientado a byte, generalmente usado para almacenar cadenas de caracteres y datos creados por computadoras de 8 bits. Sin embargo, por eficiencia, existen computadoras de 16, 32 y 64 bits, con bytes organizados en celdas de 16, 32 y 64, generando datos orientados a una *palabra*, palabra doble y palabra cuádruple cuyo orden de byte no siempre es el mismo, depende de la computadora en que hayan sido creados.

Un dato orientado a byte no tiene un orden particular y por lo tanto es leído de la misma manera en todas las computadoras. En una dato orientado a una palabra existe un gran problema, que radica principalmente cuando un dato binario es escrito a un archivo en una computadora con un orden de byte y entonces es leída con otro orden de byte, es decir, es leído incorrectamente.

Es el orden de byte en cada palabra y palabra doble de dato que determina el "endiannes" del dato. Las dos principales categorías de esquemas de ordenamiento de byte son: *big-endian* y *little-endian*. Las computadoras con big-endian almacenan el byte más significativo (Most Significative Byte, MSB) en la dirección más arriba de una palabra, usualmente referido como '0'. Este tipo de computadoras incluyen los siguientes tipos de CPU: Motorola series MC68000A, 68000,68020, 68030, 68040, ..., incluyendo la Commodore Amiga, Macintosh y algunas UNIX.

Las computadoras con little-endian almacenan el byte menos significativo (Least Significative Byte, LSB) en la dirección más abajo de una palabra. Es decir, un valor de palabra doble 1234h, escrito en un archivo con formato little-endian, sería leído como 3412h en un sistema big-endian. Este tipo de computadoras incluyen los siguientes tipos de CPU: Intel iAPX96 series 8088, 80286, 80386, 80486, ..., incluyendo IMB PC y clones.

III.2 Archivos Bitmap

Los archivos bitmap varian grandemente en sus detalles, pero muestran la misma estructura general.

Los archivos bitmap consisten de una cabecera, datos bitmap, y alguna otra información, que puede incluir una paleta de colores y otros datos.

Advertencia: inexplicablemente, se continúan diseñando aplicaciones que usan en ocasiones formatos conocidos como *raw* format (formato bruto). Los archivos raw, consisten solamente de datos de la imagen y omite algunas indicaciones en su estructura. El creador de tales archivos y las aplicaciones rendering deben conocer alguna manera, antes de la hora, como están estructurados los archivos. Ya que usualmente no podemos decir si un raw viene de otro (excepto quizás, examinan sus tamaños relativos), se hablará de los archivos bitmap que al menos contienen cabecera.

III.2.1 Estructura

Los componentes básicos de un archivo bitmap son:

Cabecera
Datos Bitmap

Si un archivo no contiene datos de la imagen, solo la cabecera estará presente. Si es requerida información adicional que no está fija en la cabecera, entonces estará presente un pie de archivo como se muestra a continuación:

Cabecera
Datos Bitmap
Pie de Archivo

Un archivo de imagen que almacena una paleta de colores, aparecerá inmediatamente después de la cabecera:

Cabecera
Paleta de colores
Datos Bitmap
Pie de Archivo

Una paleta también puede aparecer inmediatamente después de los datos de la imagen, como un pie de archivo, o ser almacenada en el mismo pie de archivo:

Cabecera
Datos Bitmap
Paleta de colores

Tablas de líneas escaneadas (*scan-line)* y corrección de color también pueden aparecer después de la cabecera y antes o después de los datos de la imagen:

Cabecera
Paleta de colores
Tabla de líneas escaneadas
Tabla de corrección de color (aquí)
Datos Bitmap
Tabla de corrección de color (ó aquí)
Pie de Archivo

Si un formato de archivo de imagen es capaz de almacenar imágenes múltiples, entonces debe aparecer un índice de archivo de imagen, que contiene los valores del desplazamiento de la posición de inicio de la imágenes en el archivo:

Cabecera
Paleta de colores
Indices Bitmap
Dato Bitmap 2
.....
Datos Bitmap "n"
Pie de Archivo

Si la definición del formato permite a cada imagen tener su propia paleta, la paleta, en la mayoría de los casos aparece antes de los datos de la imagen a los que están asociados:

Cabecera
Paleta de colores
Indices Bitmap
Paleta de colores 1
Dato Bitmap
Paleta de colores 2
Dato Bitmap 2
.....
Paleta de colores "n"
Datos Bitmap "n"
Pie de Archivo

III.2.2 Cabecera

La cabecera es una sección de datos binarios o de formato ASCII normalmente encontrados al inicio del archivo, conteniendo información acerca de los datos bitmap encontrados en cualquier parte del archivo. Todos los archivos bitmap tienen alguna forma de cabecera, aunque el formato de la cabecera y la información almacenada en ella varia considerablemente de formato a formato. Generalmente, una cabecera bitmap esta compuesta de campos fijos. Ninguno de estos campos son absolutamente necesarios, no se encuentran en todos los formatos, pero esta lista es típica de aquellos formatos de uso general actualmente.

Cabecera
Paleta de colores
Indices Bitmap
Paleta de colores 1
Identificador del archivo
Versión del archivo
Número de líneas por imagen
Número de pixeles por línea
Número de bits por pixel
Número de planos de color
Tipo de compresión
Origen X de la imagen
Origen Y de la imagen
Espacio sin usar

Identificador del Archivo

Una cabecera usualmente inicia con alguna forma de valor de identificación único llamado identificador del archivo, ID del archivo, o valor ID. Su fin es permitir al software de aplicación determinar el formato para el archivo gráfico particular que esta siendo leído.

Los valores ID son en ocasiones "valores mágicos" en el sentido de que están designados arbitrariamente por el creador del archivo de formato. Pueden ser una serie de caracteres ASCII,

tales como "BM" o "GIF", o valores de 2 ó 4 bytes, tales como 42h42h ó 59h6AhA6h95h, o algún otro patrón que tiene sentido para el creador del formato. El patrón generalmente es asumido como único, aunque atraviesa plataformas, pero éste no es siempre el caso. Generalmente, si un valor en el lugar correcto en un archivo iguala al valor de identificación esperado, la aplicación que lee la cabecera del archivo puede asumir que el formato de la imagen es conocido.

Tres circunstancias surgen, sin embargo, que hacen esto menor que una regla dura y rápida. Algunos formatos omiten el identificador del archivo, iniciando con datos que pueden cambiar de archivo a archivo. En este caso, hay una pequeña probabilidad de que los datos sean accidentalmente duplicados de uno de los "valores mágicos" de otro archivo de formato conocido por la aplicación. Afortunadamente, la opción de ésta ocurrencia es remota.

La segunda circunstancia puede venir cuando un nuevo formato es creado y el creador del formato inadvertidamente duplica, en conjunto o en parte, el "valor mágico" de otro formato, en este caso parece, aunque, a diferencia que la duplicación sea accidental, esto ya ha sucedido varias veces. Probablemente la principal causa es que, históricamente, los programadores han tomado prestadas ideas de otras plataformas, seguramente en la creencia de que sus esfuerzos serán aislados detrás de la muralla china de incompatibilidad binaria.

La tercera circunstancia viene cuando el proveedor cambia, intencionalmente o no, la especificación del formato, mientras mantiene el valor ID especificado en la documentación del formato. En este caso una aplicación puede reconocer el formato, pero es incapaz de leer algunos o todos los datos.

Versión del Archivo

A continuación del valor de identificación en la cabecera, generalmente está un campo que contienen la versión del archivo. Naturalmente, versiones sucesivas de bitmap pueden diferir en características como tamaño de cabecera, datos bitmap soportados y capacidad de colores. Una vez verificado el formato del archivo con el identificador, una aplicación generalmente examinará la versión para determinar si puede manejar los datos de imagen contenidos en el archivo.

Información de la Descripción de la Imagen

Siguen una serie de campos que describen la imagen en sí misma. Como se verá, los bitmap están generalmente organizados, físicamente o lógicamente, en líneas de pixeles. El campo Número

de líneas por imagen, también conocido como longitud de la imagen, largo de imagen o número de líneas escaneadas, mantiene un valor correspondiente al número de líneas hechas al dato bitmap real. El campo Número de pixeles por línea, también llamado ancho de imagen o ancho de línea escaneada, indica el número de pixeles almacenados en cada línea.

Número de bits por pixel indica el tamaño del dato requerido para describir cada pixel por plano de color. También puede ser almacenado como número de bytes por pixel, es más adecuado llamarlo ancho del pixel. Olvidar la interpretación exacta de este campo cuando se codifican lectores de formato es la principal fuente de errores. Si el dato bitmap es almacenado en una serie de planos, el número de planos de color indica el número de planos usados. Con frecuencia el valor por default es 1. Hay una creciente tendencia a almacenar bitmap en formato de plano sencillo, pero los planos múltiples continúan para ser utilizados en el soporte de hardware especial y modelos de color alternativos.

El número de bits en una línea de la imagen es calculado multiplicando el valor de número de bits por pixel, número de pixeles por línea y número de planos de color. Podemos determinar el número de bytes por línea escaneada dividiendo el valor resultante entre 8. Note que no hay requerimientos de número de bits por pixel para ser un número entero de bytes de 8 bits.

Tipo de Compresión

Si el formato soporta algún tipo de codificación diseñada para reducir el tamaño de los datos bitmap, entonces el campo tipo de compresión será encontrado en la cabecera. Algunos formatos soportan múltiples tipos de compresión, incluyendo datos raw o sin comprimir. Algunas revisiones de formato consisten principalmente de adiciones o cambios al esquema de compresión utilizado. Los datos de compresión son un campo activo y, nuevos tipos de compresión que alojan avances en tecnología aparecen con regularidad.

Origen X y Y

El origen X y Y de la imagen especifican un par de coordenadas que indican donde inicia la imagen en el dispositivo de despliegue. El origen más común es la coordenada (0,0), que pone una esquina de la imagen en el punto de origen del dispositivo. Cambiar estos valores generalmente causa que la imagen sea desplegada en una localización diferente cuando es enviada.

La mayoría de los formatos bitmap fueron diseñados con ciertas presunciones acerca del dispositivo en mente y, por lo tanto, puede ser dicho si un modelo de dispositivo real o virtual tiene una característica llamada superficie de dibujo (*drawing surface*). La superficie de dibujo tiene un origen implicado, que define el punto de inicio de la imagen y, una orientación implicada, que define la orientación en la cual líneas sucesivas serán dibujadas cuando la imagen de salida sea enviada. Varios formatos y dispositivos de despliegue varían en la posición del punto de origen y dirección de orientación. algunos colocan el origen en la esquina superior izquierda, en el centro o en la esquina inferior izquierda. Otros, aunque raras ocasiones lo ponen en la esquina inferior o superior derecha.

Los modelos de orientación con el origen en la esquina superior izquierda son en ocasiones dichos o han sido creados en soporte de hardware y puede haber alguna justificación histórica y del mundo real para ello. La gente con conocimientos en matemáticas y ciencias naturales, sin embargo, utilizan la esquina inferior izquierda o el centro de la superficie de dibujo.

Una imagen desplegada por una aplicación utilizando un punto de origen u orientación incorrecto puede aparecer al revés o puede ser desplazada horizontalmente alguna fracción del ancho de la superficie de dibujo del dispositivo de salida.

Algunas veces la cabecera contendrá un campo de descripción de texto, que es una sección de comentarios consistentes de datos ASCII describiendo el nombre de la imagen, nombre del archivo, el nombre de la persona quien creo la imagen, o el software de aplicación utilizado para crear la imagen. Este campo puede contener datos ASCII de 7 bits, por portabilidad de la información de la cabecera a través de plataformas.

Espacio sin Usar

Al final de la cabecera puede haber un campo sin utilizar, algunas veces referido como relleno (*padding o filler*), espacio reservado o campos reservados. Los campos reservados no contienen datos, están sin documentar, sin estructurar y esencialmente actúan como "place holders". Todo lo que se sabe de ellos es su tamaño y posición en la cabecera. Por lo tanto, si el formato es alterado en un futuro para incorporar nuevos datos, el espacio reservado puede ser usado para describir el formato o localización de éstos datos mientras se mantiene la compatibilidad con programas que soportan versiones anteriores del formato. Este es un método común empleado para minimizar los problemas de versión, creando una versión inicial basada en una cabecera fija

substancialmente más larga de lo necesario. Los nuevos campos pueden ser añadidos en las áreas reservadas en revisiones subsecuentes del formato sin alterar el tamaño de la cabecera.

En ocasiones las cabeceras de formatos son intencionalmente rellenadas usando el método de 128, 256 ó 512 bytes. Esto tienen algunas implicaciones para su ejecución, particularmente en sistemas más viejos y está diseñado para adecuar la lectura y escritura común de tamaños de buffer. El relleno puede aparecer después de los campos documentados al final de la cabecera y, esto es en ocasiones una indicación de que el creador del formato tenía desempeños y extensiones en mente al formato que ha creado.

Optimando Lectores de Cabeceras

La velocidad de un lector de cabecera puede ser optimada viendo la forma en que la aplicación utiliza los datos, ya que el lector de datos de la cabecera pueden generalmente ser llevadas a cabo de diferentes maneras. Si solo se seleccionan los valores de la cabecera que se necesitan, la aplicación puede calcular el desplazamiento de datos desde alguna marca al inicio del archivo. La aplicación puede también buscar directamente los datos de valores requeridos y leerlos.

La mayoría de los datos contenidos en la cabecera son requeridos por la aplicación, entonces puede ser más conveniente leer la cabecera completa en un buffer o estructura de datos asignada. Esto puede ser realizado rápidamente, tomando ventaja de algunas eficiencias dadas por una lectura de bloques enteros de potencias de 2. Todos los datos de la cabecera estarán disponibles en memoria para que sean tomados para su uso cuando sean requeridos. Un problema, si embargo, ocurre cuando el orden de byte del archivo es diferente al orden nativo del byte del sistema en que el archivo esta siendo leído. La mayoría de las funciones que leen bloques, por ejemplo, no están diseñadas para rellenar por el compilador o ambiente de tiempo de ejecución para propósitos de alineamiento de datos.

Paleta de Colores

Una paleta de colores, también conocida como mapa de colores (color map), mapa de índices (index map), tabla de colores (color table) o tabla look up (Look up table, LUT) es un arreglo unidimensional de valores de colores. Dichos colores pueden ser representados por 3 sistemas de colores que se muestran en la tabla III.1:

Tabla III.1. Equivalencia de valores RGB, CMY y HSV.			
	RGB	CMY	HSV
Rojo	255,0,0	0,255,255	0,240,120
Amarillo	255,255,0	0,0,255	40,240,120
Verde	0,255,0	255,0,255	80,240,120
Cían	0,255,255	255,0,0	120,240,120
Azul	0,0,255	255,255,0	160,240,120
Magenta	255,0,255	0,255,0	200,240,120
Negro	0,0,0	255,255,255	160,0,0
Sombra de grises	63,63,63	191,191,191	160,0,59
Gris	191,191,191	63,63,63	160,0,180
Blanco	255,255,255	0,0,0	160,0,240

RGB (Red Green Blue). Componentes de Rojo Verde y Azul.
CMY (Cyan Magenta Yellow). Componentes de Cían Magenta y Amarillo.
HVS (Hue , Saturation Value). Componentes de Matiz, Saturación y Importancia.

III.2.3 Datos Bitmap

Los datos bitmap reales generalmente hacen el volumen de un archivo de formato bitmap.

En cualquier archivo de formato bitmap los datos bitmap reales son encontrados inmediatamente después del fin de la cabecera. Pueden ser encontrados en cualquier parte del archivo, para alojar una paleta o alguna otra estructura de datos que pueda estar presente. Si este es el caso, un valor de desplazamiento aparecerá en la cabecera o en la documentación indicando donde encontrar el inicio de los datos de imagen en el archivo.

Afortunadamente, la estructura de los datos bitmap en la mayoría de los archivos es sencilla y fácilmente deducida. Como se ha mencionado, un dato bitmap está compuesto de valores de pixel. Los pixeles en el dispositivo de salida están generalmente dibujados en líneas escaneadas correspondientes a filas abarcando el ancho de la superficie de despliegue. Esto está reflejado en la organización de los datos de imagen.. El ejercicio de deducir la organización exacta de datos en el archivo, en ocasiones ayuda a tener una idea de los dispositivos de despliegue que el creador del formato tenía en mente.

Uno o más líneas escaneadas combinadas forman un *azulejo* de datos de pixel, aunque podemos pensar de cada pixel en el bitmap como en una coordenada especifica. Un bitmap también puede ser pensado como una secuencia de valores que lógicamente mapea datos bitmap en un archivo para una imagen en la superficie de despliegue de un dispositivo de salida. Los datos bitmap reales generalmente son la parte más larga de cualquier archivo de formato bitmap.

Organización de Datos Bitmap

Antes de que una aplicación escriba una imagen en un archivo, los datos de imagen generalmente primero son reunidos en uno o más bloques de memoria. Estos bloques pueden ser localizados en el espacio de memoria principal de la computadora o en parte de un dispositivo de colección de datos auxiliar. La forma exacta en que los datos son reunidos depende de varios factores, incluyendo la cantidad de memoria instalada, la cantidad disponible para la aplicación y la adquisición especifica de datos u operación de escritura del archivo en uso. Cuando un dato bitmap es finalmente escrito a un archivo, solo uno de los siguientes métodos de organización es utilizada: dato de líneas escaneadas o dato planar.

Dato de Líneas Escaneadas

Es el primer método más simple de organización de valores de pixel en filas o líneas escaneadas. Si consideramos que cada imagen pueda ser hecha de una o más líneas escaneadas, el dato del pixel en el archivo describe que la imagen será una serie de valores de colecciones, cada colección correspondiente a una fila de la imagen. Filas múltiples son representadas por múltiples colecciones escritas de inicio a fin en el archivo. Este es el método más común para almacenar datos de imagen organizados en filas.

Si conocemos el tamaño de cada pixel en la imagen, y el número de pixeles por fila, podemos calcular el desplazamiento del inicio de cada fila en el archivo. Por ejemplo, en una imagen de 8 bits cada valor de pixel es de un byte de longitud. Si la imagen es de 21 pixeles de ancho, las filas en el archivo están representadas por colecciones de valores de pixel de 21 bytes de ancho. En este caso, las filas en el archivo inician en los bytes de desplazamiento 0, 21, 42, 63, etc. del inicio de los datos bitmap.

En algunas máquinas y algunos formatos las filas de los datos de imagen pueden estar en ciertos múltiplos de bytes pares en longitud. Un ejemplo es la regla común requerida en datos de fila bitmap al final de límites de palabra larga, que es de 4 bytes de longitud. En el ejemplo mencionado, una imagen con un ancho de 21 pixeles entonces será almacenado en el archivo como colecciones de valores de pixel de 24 bytes de longitud y, las filas iniciarán en los desplazamientos 0, 24, 48, 64. Los 3 bytes extras por fila son el relleno. En este caso particular, los 3 bytes en el archivo son desperdiciados para cada fila y, de hecho, la imagen de 21 pixeles de ancho toma el mismo espacio que una de 24 pixeles de ancho. En la práctica este almacenamiento ineficiente es usualmente,

pero no siempre, compensado por un incremento en la velocidad ganada alimentando las peculiaridades de la máquina en consideración a su habilidad de manipular rápidamente 2 ó 4 bytes a la vez.

En una imagen de 24 bits, cada pixel corresponde a una longitud de valores de pixel de 3 bytes en el archivo. En el ejemplo revisado, una imagen de 21 pixeles de ancho requerirá un mínimo de 21*3 = 63 bytes de almacenamiento. Si el formato requiere que la fila inicie alineada en palabras largas, serán requeridos 24 bytes para almacenar el valor del pixel para cada línea. Ocasionalmente, las imágenes de 24 bits almacenan sus datos en 4 bytes, y cada fila de la imagen es almacenada en una serie de 21*4 = 84 bytes. Almacenar datos de imagen de 24 bits como valores de 4 bytes tiene la ventaja de estar alineada al límite de la palabra larga.

En una imagen de 4 bits, cada pixel corresponde a la mitad de un byte, el dato generalmente es almacenado 2 pixeles por byte, sin embargo, almacenar los datos como valores de pixel de 1 byte hará al dato más fácil de leer y, de hecho, no es inaudito.

La figura III.5 ilustra la organización de datos de pixel en líneas escaneadas.

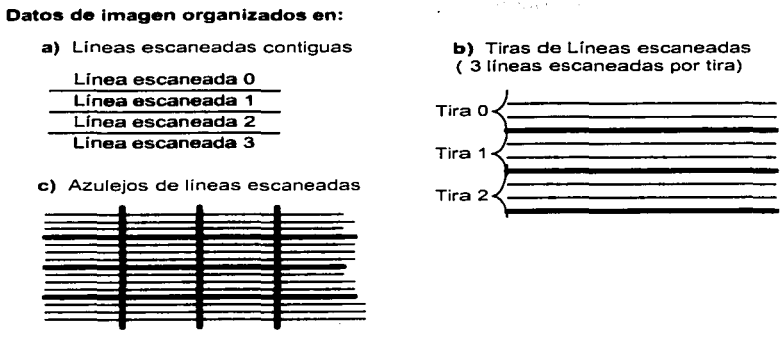


Figura III.5 Organización de datos de pixel en líneas escaneadas.

Datos de Planos de Colores

El segundo método de organización de valores de pixel envuelve la separación de datos de imagen en 2 o más planos. Los archivos en que los datos bitmap están organizados de este modo son llamados archivos de planos de colores. El término imagen compuesta hace referencia a una

imagen con varios colores (no es monocromática, ni escala de grises, ni de un solo color). Bajo este término la mayoría de las imágenes a color son imágenes compuestas.

Una imagen compuesta puede estar representada por tres bloques de datos bitmap, cada bloque contiene solo uno de los componentes de colores que hacen la imagen. Construyendo cada bloque es parecido al proceso fotográfico para hacer una separación, utilizando filtros para separar un color en un juego de componentes de colores, usualmente tres de ellos. La fotografía original puede ser reconstruida combinando las tres separaciones. Cada bloque está compuesto de filas colocadas fin a fin, como en el más simple método de almacenamiento explicado anteriormente; en éste caso, más de un bloque es requerido ahora para reconstruir la imagen. Los bloques pueden ser almacenados consecutivamente o pueden estar separados físicamente uno del otro en el archivo.

El formato de planos de colores generalmente, es una señal de que el formato está diseñado con algún tipo despliegue en particular en mente, uno que construido de pixeles de color compuesto de componentes mapeados a través del hardware diseñado para manejar un color a la vez. Por razones de eficiencia, el formato de plano de colores es leído en bloques de un plano a la vez, ya que la aplicación puede elegir para ensamblar los pixeles compuestos leyendo datos del lugar adecuado en cada plano secuencialmente.

Como un ejemplo, una imagen de 24 bits de dos filas por tres columnas de ancho puede estar representado en un formato RGB con 6 valores de pixeles:

(00, 01, 02) (03, 04, 05) (06, 07, 08)
(09, 10, 11) (12, 13, 14) (15, 16, 17)

Serán escritos en formato planar como:

plano Rojo	plano Verde	plano Azul
(00) (03) (06)	(01) (04) (07)	(02) (05) (08)
(09) (12) (15)	(10) (13) (16)	(11) (14) (17)

Note que son los mismos datos pero organizados de diferente manera. En el primer caso, una imagen consistente de pixeles de 24 bits es almacenada como 6 valores de pixel de 3 bytes organizados en un plano sencillo. En el segundo caso, método planar, la misma imagen es almacenada con 18 valores de pixel de 1 byte organizados en 3 planos, cada plano correspondiente a la información de rojo, verde y azul, respectivamente. Cada método toma exactamente la misma cantidad de espacio, 18 bytes, al menos en este ejemplo. La figura III.6 ilustra la organización de un dato de pixel en planos de colores.

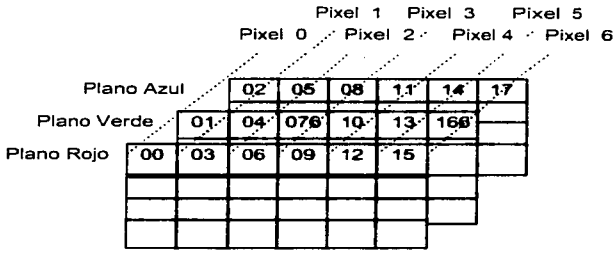


Figura III.6 Organización de datos de pixel en planos de colores.

Normalmente, consideramos realizar una imagen en determinado número de filas, cada fila con un cierto número de ancho de pixel. El dato del pixel presentando en la imagen puede ser almacenado en el archivo de tres formas: como datos contiguos, como tiras o como azulejos. La figura III.7 ilustra estas tres representaciones.

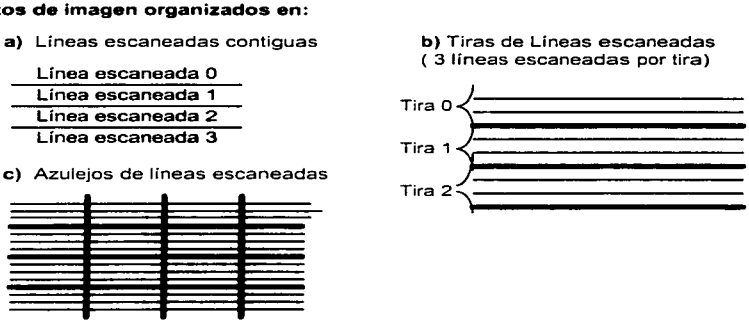


Figura III.7 Almacenamiento de datos bitmap.

Datos Contiguos

El método más simple de organización de filas es donde todos los datos de imagen están almacenados contiguamente en el archivo, una fila sigue a la última. Para recuperar los datos lea las filas en el orden del archivo, que entrega las filas en el orden en que fueron escritas. El dato en éste

esquema organizacional es almacenado en el archivo de forma equivalente a un arreglo bidimensional. Puede indexar en el dato en el archivo conociendo el ancho de la fila en pixeles y tamaño del valor de pixel. Los datos almacenados contiguamente en este manera pueden ser leídos rápidamente, en trozos largos y, ensamblados en memoria de manera rápida.

Tiras

La imagen es almacenada en tiras, que contienen filas almacenadas contiguamente. La imagen total, sin embargo, es representada por más de una tira, y la tira individual puede muy estar separada en el archivo. Las tiras dividen a la imagen en un número de segmentos, que son siempre de un ancho justo para la imagen original.

Las tiras hacen más fácil el manejo de datos de imagen en máquinas con memoria limitada. Una imagen de 1024 filas de longitud, puede ser almacenada en el archivo como 8 tiras de 128 filas cada una.

Las tiras vienen en juegos cuando los datos de pixel están almacenados en un formato codificado o comprimido. En este caso, una aplicación debe primero leer el dato compreso en un buffer y entonces decodificar el dato en otro buffer del mismo tamaño, o más largo que el primero. Organizando la compresión en una tira por base facilita la tarea del lector de formato, que necesita manejar solo un tira a la vez.

Las tiras son en ocasiones evidencias de que el creador del formato ha pensado acerca de las limitaciones de las posibles plataformas que están siendo soportadas y ha querido no limitar el tamaño de la imagen que puede ser manejada por el formato. Por lo regular, la cabecera proporciona el numero de tiras de datos, tamaño de cada tira y la posición de desplazamiento de cada tira en el archivo.

Azulejos

Los azulejos son similares a las tiras en que cada una es una delimitación de un área rectangular de una imagen. Sin embargo, a diferencia de las tiras, que están siempre en el ancho de la imagen, los azulejos pueden tener cualquier ancho, desde un pixel sencillo hasta la imagen completa. En este sentido, una imagen contigua es realmente un azulejo largo. En la práctica, sin embargo, los azulejos están organizados de tal manera que el dato de pixel correspondiente a cada

uno está entre 4 Kb. y 64 Kb. en tamaño y usualmente es de un ancho y largo divisible entre 16. Estos límites ayudan a incrementar la eficiencia con la que el dato puede ser "buffereado" y decodificado.

Cuando una imagen esta en azulejos, generalmente: todos los azulejos son del mismo tamaño, no se traslapan y están todos codificados utilizando el mismo esquema de compresión. Una excepción es el formato CALS Raster Tipo II, que permite a los datos de imagen estar compuestos de azulejos codificados y no codificados. Los azulejos se dejan sin codificar cuando tal codificación podría causar que el azulejo incremente en tamaño (compresión negativa) o cuando cantidades no razonables de tiempo serían requeridos para codificar el azulejo.

III.2.4 Pie de Archivo

El pie de archivo, a veces llamado *trailer*, es una estructura de datos similar a la cabecera y es en ocasiones una adición a la cabecera original, pero al final del archivo. Se añade cuando el formato ha sido actualizado para adecuar nuevos tipos de datos y no es conveniente modificar o agregar la información de la cabecera. Es principalmente un resultado de un deseo de mantener la compatibilidad con versiones anteriores del formato. Un ejemplo es el formato TGA, su versión actual contiene un pie de archivo que habilita aplicaciones para identificar las diferentes versiones de este formato y accesar a características especiales disponibles solo en algunas versiones del formato.

Ya que por definición aparece después de los datos de imagen, que generalmente varía en tamaño, un pie de archivo nunca es encontrado en un desplazamiento fijo desde el inicio del archivo a menos que los datos de imagen sean siempre del mismo tamaño. Sin embargo, generalmente es localizado con un desplazamiento específico desde el fin del archivo. como las cabeceras, generalmente tienen un tamaño fijo. El valor del desplazamiento del pie de archivo puede estar presente en la información de la cabecera, proveyéndolo en un espacio reservado o rellenando un espacio disponible en la cabecera. Puede contener un campo de identificación o "número mágico" que puede ser usado para enviar a la aplicación para diferenciarlo de otra estructura de datos en el archivo.

Otras Características

Un formato de archivo que permite más de una imagen en el archivo necesita dar algún método de identificación del inicio de cada imagen. Por lo tanto, una tabla de desplazamiento de

imagen, también conocida como índice de archivo de imagen o página de tabla, debe ser usado para almacenar el valor del desplazamiento de inicio de cada imagen desde el inicio del archivo.

Una tabla de líneas escaneadas debe ser provista para localizar el inicio de cada línea escaneada de datos de imagen. Esto puede ser útil si los datos de imagen están comprimidos y un dato de pixel correspondiente a una línea escaneada individual debe ser accesada aleatoriamente; el pixel en los datos de imagen no puede ser incluido hasta que el dato de imagen es codificado. La tabla de líneas escaneadas contienen una entrada por línea escaneada de imagen. Variantes de esta idea incluyen tablas de localización de tiras (una entrada por grupo de líneas escaneadas) y tablas de localización de azulejos (una entrada por cada área rectangular de la imagen).

Algunos formatos incorporan estructuras de datos inusuales o únicas en su diseño. Usualmente cumplen los propósitos de especificación del formato o lo crean tan generalmente como sea posible. Como TIFF que se verá más adelante.

Pros y Contras de Formatos de Archivos Bitmap

Ventajas:

- Los archivos bitmap están estructurados especialmente para almacenar imágenes del mundo real; imágenes complejas rastreadas con video, scanners y fotografías.
- Pueden ser fácilmente creados de datos de pixel existentes almacenados en arreglos de memoria.
- Recuperar datos de pixel almacenados en un archivo bitmap puede en ocasiones ser cumplido utilizando un juego de coordenadas que permite a los datos ser vistos como un azulejo.
- Los valores de pixel pueden ser modificados individualmente o como grupos largos alterando una paleta de colores presente.
- Los archivos bitmap se pueden interpretar bien en dispositivos de salida de formato de puntos tales como CRT e impresoras.

Desventajas:

- Pueden ser muy extensos, particularmente si la imagen contiene un número grande de colores. La compresión de datos puede reducir el tamaño de datos de pixel, pero el dato debe ser decodificado antes de poder utilizarlo, esto puede hacer lento el proceso de lectura y envío considerablemente. El dato bitmap es más complejo como menos eficiente sea el método de compresión.

- No escalan muy bien. Reducir una imagen por *decimation* (tirar pixeles) puede cambiar la imagen de una manera inaceptable, así como expandir la imagen por réplicas de pixeles. Razón por la cual los archivos bitmap deben generalmente ser impresos en la resolución en que fueron almacenados originalmente.

III.3 Formato BMP

Nombre	Microsoft Windows Bitmap.
Conocido como	BMP, Windows BMP, Windows DIB, Bitmap Compatible.
Tipo	Bitmap.
Colores	1,4,8,16,24 y 32 Bits.
Compresión	RLE, Sin comprimir.
Tamaño Máximo de Imagen	32 Kb x 32 KB y 2 Gb x 2 Gb pixeles.
Imágenes Múltiples por Archivo	No.
Orden de Byte	Little-endian.
Creador	Microsoft Corp.
Plataforma	MS-DOS, Windows, Windows NT, Windows 95 y Os/2.
Aplicaciones que soporta	Demasiado numerosas para listarlas.

III.3.1 Antecedentes

El archivo de formato bitmap de Windows es uno de los archivos gráficos que soporta el ambiente Windows. BMP es el formato nativo bitmap de Windows y OS/2, es utilizado para almacenar virtualmente cualquier tipo de dato bitmap. La mayoría de los gráficos y aplicaciones que corren bajo ambiente Windows soportan la creación y despliegue de archivos de formato BMP. También es muy popular en el Sistema Operativo MS-DOS.

El formato original bitmap creado para Windows v1.0 fue muy simple, tenía una paleta de color fija, no soportaba compresión de datos y fue diseñado para soportar las más populares tarjetas de video disponibles en ese tiempo (CGA, EGA, Hércules y otras). Este formato ya caducado es referido a veces como el original Device Dependent Bitmap (DDB) de Windows.

Windows v2.0 soportaba una paleta de color programable permitiendo datos de color definidos por el usuario para ser almacenados junto con el dato bitmap usado. Cuando es almacenado en memoria, esta colección de información es conocida como Microsoft Device Independent Bitmap (DIB). Cuando ésta información es escrita en un archivo, es conocido como Microsoft Bitmap Format (BMP). Cuando se hace referencia al archivo de formato DIB, es el BMP que esta siendo referido.

Durante el desarrollo de BMP, Microsoft compartió responsabilidad con IBM para el desarrollo de versiones posteriores del Sistema Operativo OS/2. Con el Presentation Manager, el OS/2 necesitó un formato bitmap, el archivo de formato BMP de Windows fue usado. Por lo tanto, el archivo de formato BMP de Windows v2.0 y OS/2 v1.x son idénticos.

El formato BMP fue modificado para Windows v3.0 difiere ligeramente del bitmap del Presentation Manager de OS/2 que le precedía. Las versiones posteriores diseñadas para soportar el bitmap del Presentation Manager de OS/2 v2.x han resultado en divergencia entre los archivos BMP de Windows e IBM OS/2. La versión actual de Windows v4.0 (Windows 95) contiene todas las características e historia del formato BMP de Windows v2.x, v3.x y Windows NT.

La estructura del archivo de formato BMP está vinculada con detalle al *API* (Advanced Programming Interface) de Windows y OS/2. BMP no fue un formato portable o usado para datos Bitmap intercambiables entre diferentes Sistemas Operativos. Cada uno de los APIs de estos Sistemas Operativos han cambiado y el formato BMP ha cambiado con ellos.

Actualmente hay tres versiones de BMP bajo Windows (v2.x, v3.x y v4.x [Windows 95]), dos versiones bajo OS/2 (v1.x y v2.x con 6 posibles variaciones) y una versión para Windows NT.

El formato BMP actual es independiente del hardware y puede adecuar imágenes arriba de 32 bits de color. Su diseño general lo hace un formato necesario que puede ser usado para almacenar imágenes a color o en blanco y negro, si el tamaño del archivo no es un factor. Sus principales virtudes son su simplicidad y soporte extendido.

El método de compresión usado es un tipo de codificación RLE (Run Length Encoding), sin embargo, la mayoría de los archivos BMP han sido almacenados sin comprimir. Una excepción notable es el Windows v3.1 distribuido con todas las copias del producto. El esquema RLE de BMP es pequeño, fácil y rápido de descomprimir, no es considerado como un método de compresión sofisticado.

III.3.2 Estructura

III.3.2.1 Microsoft Windows v1.x Device Dependent Bitmap

El archivo DDB contiene una cabecera del archivo seguida de los datos bitmap sin comprimir.

Cabecera

Contiene 6 campos con una longitud de 10 bytes en total.

Campo 1. Tipo (2 bytes, bytes 0-1). Especifica el tipo de archivo. Debe ser '0'.

Campo 2. Ancho (2 bytes, bytes 2-3). Especifica el ancho en pixeles de la imagen bitmap.

Campo 3. Largo (2 bytes, bytes 4-5). Especifica el largo en pixeles de la imagen bitmap.

Campo 4. Ancho en byte (2 bytes, bytes 6-7). Especifica el ancho del bitmap en bytes.

Campo 5. Planos (1 byte, byte 8). Especifican el número de planos de color utilizados para almacenar la imagen. Este valor siempre es '1'.

Campo 6. Bits por pixel (1 byte, byte 9). Especifica el tamaño en bits de cada pixel. Generalmente toma los valores 1, 4 u 8.

Los datos de la imagen le siguen a la cabecera y son almacenados en un formato sin comprimir. Cada pixel almacena un índice del valor de paleta de color fija del sistema usado por Windows v1.0

III.3.2.2 Microsoft Windows v2.x

Los archivos BMP de Windows 2.0 posee una cabecera del archivo, seguida de una cabecera del bitmap y, la paleta de colores.

Cabecera del archivo

Posee información general del archivo, tiene una longitud de 14 bytes.

Campo 1. Tipo de archivo (2 bytes, bytes 0-1). Especifica el tipo de archivo. Debe ser 4D42h o "BM" en ASCII.

Campo 2. Tamaño del archivo (4 bytes, bytes 2-5). Especifica el tamaño del archivo.

Campo 3. Reservado 1 (2 bytes, bytes 6-7). Debe estar puesto en '0'.

Campo 4. Reservado 2 (2 bytes, bytes 8-9). Debe estar puesto en '0'. Estos dos campos reservados son usados por una aplicación cuando la cabecera es leída en memoria.

Campo 5. Bits de desplazamiento (4 bytes, bytes 10-13). Especifican el número de bytes del desplazamiento desde la cabecera del archivo al bitmap en el archivo, es decir, la posición de inicio de los datos bitmap.

Cabecera del Bitmap

Posee información específica de los datos bitmap, tiene una longitud de 12 bytes.

Campo 6. Tamaño de la cabecera del bitmap (4 bytes, bytes 14-17). Especifica el tamaño de la cabecera del bitmap. Para Windows v2.0 este valor debe ser 12.

Campo 7. Ancho (2 bytes, bytes 18-19). Especifica el ancho en pixeles de la imagen

Campo 8. Largo (2 bytes, bytes 20-21). Especifica el largo en pixeles de la imagen. Este campo es con signo. Si el valor es positivo la imagen es de abajo hacia arriba con origen en la esquina inferior izquierda. Si es negativo entonces la imagen es de arriba hacia abajo con origen en la esquina superior izquierda.

Campo 9. Planos (2 bytes, bytes 22-23). Número de planos de color. Los archivos BMP solo contienen un plano de color, así que este valor debe estar puesto en 1.

Campo 10. Bits por pixel (2 bytes, bytes 24-25). Especifican el número de bits en cada pixel. Este valor puede ser 1, 4, 8 ó 24.

Paleta de colores

Una paleta de colores está siempre presente en un archivo BMP si el bitmap contiene datos de 1, 4 u 8 bits. Bitmap de 24 bits nunca utiliza una paleta de colores. Cada elemento de la paleta tiene 3 bytes de longitud, un byte para cada componente de rojo, verde y azul, cada uno de los cuales puede tener un valor entre 0 y 255. El tamaño de la paleta es calculado del campo 5, bits por pixel. La paleta tiene 2, 16, 256 ó 0 entradas para un valor de bits por pixel de 1, 4, 8 y 24 respectivamente.

III.3.2.3 Microsoft Windows v3.x

Los archivos BMP de Windows 3.x poseen cuatro secciones que se muestran en la figura III.8

De estas cuatro secciones solo la paleta de colores es opcional, dependiendo del ancho de bit del bitmap. La cabecera del archivo posee 14 bytes y es idéntica a la cabecera de un archivo Windows v2.x. Es seguida de una segunda cabecera conocida como cabecera del bitmap, una paleta de colores de tamaño variable y por último los datos bitmap.

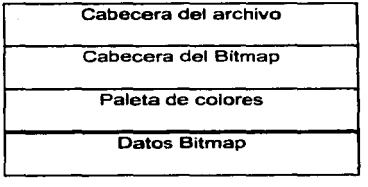


Figura III. 8 Estructura del Archivo BMP.

Cabecera del archivo

Campo 1. Tipo de archivo (2 bytes, bytes 0-1). Especifica el tipo de archivo. Debe ser "BM" o 4D42h.

Campo 2. Tamaño del archivo (4 bytes, bytes 2-5). Especifica el tamaño del archivo en bytes.

Campo 3. Reservado 1 (2 bytes, bytes 6-7). Debe estar puesto en '0'.

Campo 4. Reservado 2 (2 bytes, bytes 8-9). Debe estar puesto en '0'.

Campo 5. Bits de desplazamiento (4 bytes, bytes 10-13). Especifican el número de bytes del desplazamiento desde la cabecera del archivo al bitmap en el archivo, es decir, la posición de inicio de los datos bitmap.

Cabecera del Bitmap

Campo 6. Tamaño de la cabecera del bitmap (4 bytes, bytes 14-17). Especifica el tamaño de la cabecera del bitmap. Para Windows v3.x este valor debe ser 40.

Campo 7. Ancho (4 bytes, bytes 18-21). Especifica el ancho en pixeles de la imagen

Campo 8. Largo (4 bytes, bytes 22-25). Especifica el largo en pixeles de la imagen. Este campo es con signo. Si el valor es positivo la imagen es de abajo hacia arriba con origen en la esquina inferior izquierda. Si es negativo entonces la imagen es de arriba hacia abajo con origen en la esquina superior izquierda.

Campo 9. Planos (2 bytes, bytes 26-27). Número de planos de color. Los archivos BMP solo contienen un plano de color, así que este valor debe estar puesto en 1.

Campo 10. Bits por pixel (2 bytes, bytes 28-29). Especifican el número de bits en cada pixel. Este valor puede ser 1, 4, 8 ó 24.

Campo 11. Contiene los siguientes seis campos adicionales.

Campo 12. Tipo de compresión (4 bytes, bytes 30-33). Especifica el método de compresión para bitmap comprimidos. Puede tomar alguno de los valores que se muestran en la tabla III.2

Tabla III.2 Tipo de compresión en Windows v3.x.

Tipo de compresión	Descripción
0 (BI_RGB)	El bitmap no está compreso
1 (BI_RLE8)	Compresión Run Length para bitmap de 8 bits por pixel
2 (BI_RLE4)	Compresión Run Length para bitmap de 4 bits por pixel

Campo 12. Tamaño del bitmap (4 bytes, bytes 34-37). Especifica el tamaño del bitmap en bytes. Este valor generalmente es '0' si los datos bitmap están sin comprimir; en este caso el decodificador calcula el tamaño mediante las dimensiones de la imagen.

Campo 13. Resolución Horizontal (4 bytes, bytes 38-41). Indica la resolución horizontal en pixeles por metro.

Campo 14. Resolución vertical (4 bytes, bytes 42-45). Indica la resolución horizontal en pixeles por metro

Campo 15. Colores utilizados (4 bytes, bytes 46-49). Número de colores en la paleta de la imagen. Si este valor es cero, y el valor de bits por pixel es menor a 16, entonces el número de entradas es igual al máximo tamaño posible para la paleta. En los archivos BMP con un valor de bits por pixel de 16 o más no tendrá paleta de colores incluida. Este valor es calculado usando la siguiente fórmula: Colores utilizados = 1 con corrimiento de bits por pixel hacia la izquierda.

Campo 16. Colores importantes (4 bytes, bytes 50-53). Número mínimo de colores significativos en la imagen, determinados por su frecuencia de aparición en los datos bitmap, la mayor frecuencia de ocurrencia de un color es el más importante. Este campo es usado para mejorar la exactitud en un despliegue tanto como sea posible cuando se usa un hardware gráfico que soporta menos colores que los definidos por la imagen.

Paleta de colores

La paleta de colores (si es que la hay) contiene 1, 4 u 8 bits de datos. Datos bitmap de 24 bits nunca utilizan una paleta de colores (no es requerido). Cada elemento de la paleta está en 3 bytes de longitud, un byte para cada componente de rojo, verde y azul.

Rojo, verde y azul contienen el valor componente para un pixel, cada uno en un rango de 0 a 255. El tamaño de la paleta de colores es calculada del valor de bits por pixel. La paleta tiene 2, 16, 256 ó 0 entradas para 1, 4, 8 y 24 bits por pixel.

Para detectar la presencia de una paleta en un archivo BMP, se puede calcular el número de bytes entre la cabecera del bitmap y los datos bitmap y dividir este número entre el tamaño de elemento simple de la paleta. si el número de entradas es cero, no hay paleta; de otro modo, obtiene el número de elementos en la paleta.

III.3.2.4 Microsoft Windows NT

Windows NT usa una variación del formato BMP de Windows v3.x para almacenar datos de 16 y 32 bits en un archivo BMP. Esta variación añade 3 campos adicionales que siguen a la cabecera del bitmap en lugar de la paleta de colores. La cabecera del bitmap tiene 40 bytes de longitud con los siguientes campos:

Cabecera del Bitmap

Campo 12. Tamaño de la cabecera del bitmap (4 bytes, bytes 14-17). Especifica el tamaño de la cabecera del bitmap.

Campo 13. Ancho (4 bytes, bytes 18-21). Especifica el ancho en pixeles de la imagen

Campo 14. Largo (4 bytes, bytes 22-25). Especifica el largo en pixeles de la imagen. Este campo es con signo. Si el valor es positivo la imagen es de abajo hacia arriba con origen en la esquina inferior izquierda. Si es negativo entonces la imagen es de arriba hacia abajo con origen en la esquina superior izquierda.

Campo 15. Planos (2 bytes, bytes 26-27). Número de planos de color. Los archivos BMP solo contienen un plano de color, así que este valor debe estar puesto en 1.

Campo 16. Bits por pixel (2 bytes, bytes 28-29). Especifican el número de bits en cada pixel. Este valor puede ser 1,4,8 ó 24.

Campo 17. Tipo de compresión (4 bytes, bytes 30-33). Especifica el método de compresión para bitmap comprimidos. Puede ser uno de los valores mostrados en la tabla III.3 .

Tabla III.3 Tipo de compresión en Windows NT.

Tipo de compresión	Descripción
0 (BI_RGB)	El bitmap no está compreso
1 (BI_RLE8)	Compresión Run Length para bitmap de 8 bits por pixel
2 (BI_RLE4)	Compresión Run Length para bitmap de 4 bits por pixel
3	Compresión Bitfield

Si el bitmap contiene datos de 16 ó 32 bits, entonces tipo de compresión solo soporta el valor de 3 y los campos Máscara rojo, Máscara verde y Máscara azul estarán presentes a continuación de la cabecera del bitmap en lugar de la paleta de colores.

Campo 12. Tamaño del bitmap (4 bytes, bytes 34-37). Especifica el tamaño del bitmap en bytes. Este valor generalmente es '0' si los datos bitmap están sin comprimir; en este caso el decodificador calcula el tamaño mediante las dimensiones de la imagen.

Campo 13. Resolución Horizontal (4 bytes, bytes 38-41). Indica la resolución horizontal en pixeles por metro

Campo 14. Resolución vertical (4 bytes, bytes 42-45). Indica la resolución horizontal en pixeles por metro

Campo 15. Colores utilizados (4 bytes, bytes 46-49). Número de colores en la paleta de la imagen. Si este valor es cero, y el valor de bits por pixel es menor a 16, entonces el número de entradas es igual al máximo tamaño posible para la paleta. En los archivos BMP con un valor de bits por pixel de 16 o mayor no tendrá paleta de color incluida. Este valor es calculado usando la siguiente fórmula: Colores utilizados = 1 con corrimiento de bits por pixel hacia la izquierda.

Campo 16. Colores importantes (4 bytes, bytes 50-53). Número mínimo de colores significativos en la imagen, determinados por su frecuencia de aparición en los datos bitmap, la mayor frecuencia de ocurrencia de un color es el más importante. Este campo es usado para mejorar la exactitud en un despliegue tanto como sea posible cuando se usan hardware gráfico que soportan menos colores que los definidos por la imagen.

Como ya se hizo mención si el campo 11, tipo de compresión es 3, aparecen los siguientes tres campos, de lo contrario el archivo es idéntico a un archivo BMP De Windows v3.x.

Campo 17. Máscara Rojo (4 bytes, bytes 54-57). Máscara de identificación de bits para rojo.

Campo 18. Máscara Verde (4 bytes, bytes 58-61). Máscara de identificación de bits para verde.

Campo 19. Máscara Azul (4 bytes, bytes 62-65). Máscara de identificación de bits para azul.

Las máscaras especifican que bits en un valor de pixel corresponden a un color especifico en bitmap de 16 ó 32 bits. Los bits en los valores de las máscaras deben ser contiguos y no deben contener campos traslapados. Los bits en el pixel están ordenados del bit más significativo al menos significativo. Para bitmap de 16 bits, el formato RGB565 es en ocasiones usado para especificar 5 bits para rojo y azul y 6 para verde:

Máscara rojo	= 0xF8000000	1111	1000	0000	0000	0000	0000	0000	0000
Máscara verde	= 0x07E00000	0000	0111	1110	0000	0000	0000	0000	0000

Máscara azul	= 0x0001F000	0000	0000	0001	1111	0000	0000	0000	0000

Para 32 bits, el formato RGB101010 es en ocasiones usado para especificar 10 bits para rojo, 10 para verde y 10 para azul:

Máscara rojo	= 0xFF00000	1111	1111	1100	0000	0000	0000	0000	0000
Máscara verde	= 0x003FF000	0000	0000	0011	1111	1111	0000	0000	0000
Máscara azul	= 0x00000FFC	0000	0000	0000	0000	0000	1111	1111	1100

III.3.2.5 Microsoft Windows 95

Inicia con los mismos 14 bytes de la cabecera del archivo como Windows v2.x y v3.x. También le sigue una cabecera del bitmap, que es una versión extendida de la cabecera de Windows v3.x, incorporando los campos de máscara de Windows NT. Tiene un tamaño de 108 bytes con 17 campos adicionales.

Campos añadidos para Windows v4.x:

Campo 17. Máscara rojo (4 bytes, bytes 54-57). Máscara de identificación de bits para rojo.

Campo 18. Máscara verde (4 bytes, bytes 58-61). Máscara de identificación de bits para el verde.

Campo 19. Máscara azul (4 bytes, bytes 62-65). Máscara de identificación de bits para el azul.

Campo 20. Máscara alfa (4 bytes, bytes 66-69). Máscara de identificación de bits para alfa.

Los campos 17-20 especifican que bits en un valor de pixel corresponden a un color especifico o canal alfa en bitmap de 16 ó 32 bits. Los bits en los valores de las máscaras deben ser contiguos y no deben contener campos traslapados. Los bits en el pixel están ordenados del bit más significativo al menos significativo. Para bitmap de 16 bits, el formato RGB555 es en ocasiones usado para especificar 5 bits para rojo, verde y azul:

Máscara alfa	= 0xF8000000	1111	1000	0000	0000	0000	0000	0000	0000
Máscara rojo	= 0x07C00000	0000	0111	1100	0000	0000	0000	0000	0000
Máscara verde	= 0x003E0000	0000	0000	0011	1110	0000	0000	0000	0000
Máscara azul	= 0x0001F000	0000	0000	0000	0001	1111	0000	0000	0000

Para 32 bits, el formato RGB888 especifica 8 bits para rojo, 8 para verde y 8 para azul:

Máscara alfa	= 0xFF000000	1111	1111	0000	0000	0000	0000	0000	0000
Máscara rojo	= 0x0FF00000	0000	0000	1111	1111	0000	0000	0000	0000

Máscara verde	= 0x0000FF00	0000	0000	0000	0000	1111	1111	0000	0000
Máscara azul	= 0x000000FF	0000	0000	0000	0000	0000	0000	1111	1111

Note que Windows 95 solo soporta las máscara RGB555 y RGB565 para bitmap de 16 bits y RGB888 para bitmap de 32 bits.

Campo 21. Tipo de Color de Espacio (4 bytes, bytes 70-73). Especifica el tipo de color de espacio usado por el bitmap los valores puede tomar son:

00h	Calibrado RGB
01h	Dispositivo dependiente RGB
02h	Dispositivo dependiente CMYK

El valor por default es Dispositivo dependiente RGB. Calibrado RGB está definido por el estándar CIE XYZ.

Campo 22. Rojo X (4 bytes, bytes 74-77). Coordenada X de fin de punto para rojo.

Campo 23. Rojo Y (4 bytes, bytes 78-81). Coordenada Y de fin de punto para rojo.

Campo 24. Rojo Z (4 bytes, bytes 82-85). Coordenada Z de fin de punto para rojo.

Los campos 22-24 especifican las coordenadas CIE XYZ del fin de punto del componente rojo de un color de espacio lógico especificado. Estos campos son usados cuando el campo 21, Tipo de color de espacio es 00h (calibrado RGB).

Campo 25. Verde X (4 bytes, bytes 86-89). Coordenada X de fin de punto para verde.

Campo 26. Verde Y (4 bytes, bytes 90-93). Coordenada Y de fin de punto para verde.

Campo 27. Verde Z (4 bytes, bytes 94-97). Coordenada Z de fin de punto para verde.

Los campos 25-27 especifican las coordenadas CIE XYZ del fin de punto del componente Verde de un color de espacio lógico especificado. Estos campos son usados cuando el campo 21, Tipo de color de espacio es 00h (calibrado RGB).

Campo 28. Azul X (4 bytes, bytes 98-101). Coordenada X de fin de punto para Azul.

Campo 29. Azul Y (4 bytes, bytes 102-105). Coordenada Y de fin de punto para Azul.

Campo 30. Azul Z (4 bytes, bytes 106-109). Coordenada Z de fin de punto para Azul.

Los campos 28-30 especifican las coordenadas CIE XYZ del fin de punto del componente Azul de un color de espacio lógico especificado. Estos campos son usados cuando el campo 21, Tipo de color de espacio es 00h (calibrado RGB).

Campo 31. Gamma Rojo (4 bytes, bytes 110-113). Coordenada Gamma para rojo.

Campo 32. Gamma Verde (4 bytes, bytes 114-117). Coordenada Gamma para verde.

Campo 33. Gamma Azul (4 bytes, bytes 118-121). Coordenada Gamma para azul.

Los campos 31-33 especifican las coordenadas gamma de rojo, verde y azul para el bitmap.

Todos los campos añadidos a la cabecera bitmap de Windows 4.x son usados para soportar bitmap de 16 y 32 bits y caracterización de color del bitmap. El procesamiento del color puede ser ejecutado en una imagen y la información ICM (Image Color Matching) almacenada en un archivo BMP. Estos datos son usados para mejorar el proceso de "emparejamiento" de color cuando el bitmap es impreso o desplegado.

Paleta de Colores

Como ya se ha visto una paleta de color BMP es arreglo de una estructura que especifica los valores de intensidad de rojo, verde y azul de cada color en la paleta para un dispositivo de despliegue. Cada pixel en el dato bitmap almacena un valor usado como un índice en la paleta de color. La información de color almacenada en el elemento en ese índice especifica el color de ese pixel. A continuación se presenta la tabla III.4 con las características de los datos en un archivo BMP de Windows.

Tabla III.4 Características de datos en un archivo BMP.

Ancho pixel (Bits/ pixel)	Tamaño del pixel	Tipo de Compresión	Paleta de colores	Máscaras de colores
1 bit	1 bit	0	Sí	No
4 bits	4 bits	0,2	Sí	No
8 bits	1 byte	0,1	Sí	No
16 bits	4 bytes	3	No	Sí
24 bits	3 bytes	0	No	No
32 bits	4 bytes	3	No	Sí

III.3.3 Tipos de Imagen

Cada nueva versión de BMP ha añadido nueva información a la cabecera del bitmap. En algunos casos, las versiones más recientes han cambiado el tamaño de la paleta y añadido características al formato por sí mismo. Desafortunadamente, no fue incluido un campo en la cabe[illegible] para identificar fácilmente la versión del formato de archivo o el tipo de Sistema Operativo en que fue creado el archivo BMP.

Está claro que no podemos conocer el formato interno de un archivo BMP basados solo en su extensión. Afortunadamente, podemos usar un pequeño algoritmo para determinar el formato interno de los archivos BMP.

Iniciamos en el campo 1, tipo de archivo de la cabecera del archivo; consiste de 2 bytes con los valores 424Dh ("BM"), entonces tenemos una imagen simple de archivo BMP que pudo haber sido creada bajo Windows u OS/2. Si el tipo de archivo es 4142h ("BA"), entonces tiene un archivo de arreglo bitmap OS/2. Otras variaciones de BMP de OS/2 son las extensiones .ICO y .PTR.

Si su tipo de archivo es "BM", entonces debe leer el campo 6, tamaño de la cabecera del bitmap, dependiendo del valor de este campo se puede determinar la versión del archivo conforme a la tabla III.5.

Tabla III.5 Tipos de archivos BMP.

Valor	Tipo de archivo
12	Windows v2.x OS/2 v1.x
40	Windows v3.x Windows NT
12-64	OS/2 v2.x
108	Windows v4.x (Windows 95)

Un archivo BMP de Windows NT siempre tiene un valor de compresión de 3; de otro modo, se lee como un archivo BMP de Windows v3.x.

Los bitmap monocromáticos contienen un bit por pixel, 8 pixeles por byte (siendo el bit más significativo el pixel más a la izquierda), y tienen dos elementos de color en la paleta. Si un lector BMP elige ignorar la paleta de colores, todos los bits '1' son puestos al color del primer plano del dispositivo de despliegue y todos los bits '0' son puestos al color de fondo.

Cuatro bits por pixel son empaquetados 2 por byte con el más significativo a la izquierda. Ocho bits por pixel son almacenados 1 por byte. Los valores de 4 y 8 son índices en la paleta de colores con tamaño de 16 y 256 elementos respectivamente.

Dieciséis bits por pixel en un formato de Windows NT son de 2 bytes en tamaño y son almacenados en orden big-endian. En otras palabras, en máquinas con orden little-endian estos bytes deben ser leídos y volteados en orden little-endian antes de ser usados. La organización de los campos de 16 bits por pixel está definida por los valores de los campos 17,18 y 19 (Máscara rojo, verde y azul) en la cabecera del bitmap.

III.3.4 Compresión

El formato BMP de Windows soporta un esquema de compresión simple: RLE (Run Length Encoded) para comprimir datos bitmap de 4 y 8 bits. Ya que es un esquema RLE de 1 byte, los bitmap de 1, 16, 24 y 32 no lo utilizan, debido a la falta de largas series de byte con valores idénticos en este tipo de datos.

BMP usa un esquema RLE de 2 valores. El primer valor contiene una cuenta del número de pixeles en la serie, y el segundo valor contiene el valor del pixel. RLE toma ventaja del hecho de que algunos tipos de imágenes tienen largas secciones en donde los valores del pixel son los mismos, es decir, series que contienen pixeles del mismo valor, RLE puede reducir grandemente el tamaño de la imagen almacenada. Series de 255 pixeles de valores idénticos pueden ser codificadas tan solo en 2 bytes de datos. En adición a la series de códigos, hay series no codificadas, marcadores delta, marcadores de fin de línea escaneada, y un marcador de fin de dato RLE.

RLE8

El algoritmo RLE de 8 bits (RLE8) almacena valores de pixel repetidos como una serie de códigos. El primer byte de una serie codificada estará en el rango de 1 a 255. El segundo byte es el valor de un pixel de 8 bits en la serie. Por ejemplo, una serie codificada de 05 18 sería decodificada en 5 pixeles cada uno con el valor 18, ó 18 18 18 18 18.

Cuando una línea escaneada no contiene suficientes series de pixeles para realizar una cantidad significante de compresión, valores contiguos de pixel pueden ser almacenados literalmente o en series no codificadas. Una serie no codificada puede contener de 3 a 255 valores de pixel. El primer byte de una serie no codificada siempre será '00'. Esto hace posible distinguir la diferencia entre el inicio de un valor de pixel codificado y el inicio de una serie no codificada. El segundo byte es el número de pixeles no codificados que siguen. Si el número de pixeles no codificados es impar, entonces un valor de relleno 00 le sigue. Este valor de relleno no es parte del dato del pixel original y no debe ser escrito a la cadena de datos decodificados. Por ejemplo:

Bytes codificados	Bytes decodificados	Descripción
05 10	10 10 10 10 10	Cinco bytes con valor 10h (16)
00 05 23 65 34 56 45 00	23 65 34 56 45	Cinco bytes de datos no codificados
01 03 05 0A	03 0A 0A 0A 0A 0A	Un byte 03 y 5 bytes con valor 0Ah (10)
01 03 01 04 05 0B	03 04 0B 0B 0B 0B 0B	Un byte 03, un 04 y 5 bytes con valor 0Bh (11)
00 04 46 57 68 79	46 57 68 79	Cuatro bytes de datos no codificados
01 03 05 0A	03 0A 0A 0A 0A 0A	Un byte 03 y 5 bytes con valor 0Ah (10)

Existen 3 marcadores que pueden ser encontrados en datos RLE. Cada uno de estos marcadores también inician con un byte de valor 00. El segundo byte indica el tipo de marcador. Estos marcadores especifican información de posiciones relacionadas al dato bitmap decodificado y no genera ningún dato por sí mismos, se muestran en la tabla III.6 .

Tabla III.6 Marcadores de datos RLE	
Valor	**Especificación**
00 00	Fin de línea escaneada
00 01	Fin de dato bitmap
00 02 XX YY	Desplazamiento de serie (delta)

El primer marcador es el marcador de fin de línea escaneada y es identificado por dos bytes con los valores 00 y 00. Este marcador es una indicación de que el fin de datos para la línea escaneada actual ha sido alcanzada. Los datos codificados ocurridos después de éste marcador son decodificados como el inicio de la siguiente línea escaneada.

Este marcador es usado solo cuando se desea forzar la decodificación de un línea escaneada a terminar en un lugar en particular. Si el marcador de fin de línea escaneada ocurre a la mitad de la línea escaneada, los pixeles sobrantes en el bitmap decodificados para la línea son ignorados. Esta técnica de "línea escaneada corta" es utilizada para omitir porciones no requeridas de las líneas escaneadas. La mayoría de los casos, es encontrada en archivos BMP de iconos y punteros.

El siguiente marcador es el marcador de fin de dato RLE, es identificado por dos bytes con los valores 00 y 01. Este marcador aparece en los últimos 2 bytes de los datos RLE. Indica que el lector debe detener la decodificación de datos.

El último marcador es el marcador de desplazamiento de serie, también conocido como delta o código de vector. Tiene 4 bytes de longitud, los dos primeros con los valores 00 y 02, y los últimos 2 indican una dirección de pixel usando valores sin signo para X y Y como un desplazamiento desde la actual posición del bitmap. El valor X es el número de pixeles a través de la línea escaneada, y el valor Y es el número de líneas adelante en el bitmap.

Este desplazamiento indica la localización en el bitmap donde deben estar escritos la siguiente serie de pixeles decodificados. Por ejemplo, Un marcador de desplazamiento de serie con el valor: 00 02 05 03 indicará que el desplazamiento del bitmap debe moverse 5 pixeles abajo de la línea escaneada, y 3 líneas hacia adelante, y escribir la siguiente serie.

Este marcador es útil cuando un bitmap contiene grandes cantidades de pixeles no importantes. Por ejemplo, si el archivo BMP contiene bitmap usados como máscara (tales como los usados en iconos y punteros), algunos de los pixeles en el bitmap rectangular no deben ser usados. Se prefiere almacenar estos pixeles no usados en el archivo BMP, solo los pixeles significativos están almacenados, y los marcadores delta son usados como "saltos" para evitar las partes del bitmap no usada en la máscara.

A continuación se presenta un algoritmo del método de compresión RLE en un formato BMP:

```
Escribir los datos de la cabecera y paleta de colores
Leer los datos bitmap desde la esquina inferior izquierda (Campo 8 con valor positivo)
Repetir
  Repetir
    Leer datos de la imagen (1 dato = 1 pixel).
    Si hay por lo menos 3 datos que no se repitan
        Contar de hasta 255 datos posibles que no se repitan
         Escribir 0, la cuenta  y los datos que no se repiten
         Si en total de la cuanta es valor impar
             Escribir 0
          Fin del si
        Si no
          Si hay 2 datos que no se repitan
              Escribir 1, el primer dato ,1 y el segundo dato
          Si no
              Contar de entre 1 a 255 datos posibles que se repitan.
              Escribir la cuenta y el dato que se repite.
             Fin del si
          Fin del si
  Hasta que sea un fin de línea
Escribir 0 0
Hasta leer todos los datos de la imagen
Escribir 0 1
```

Codifiquemos la siguiente línea de datos utilizando el algoritmo de compresión que utiliza BMP

$Pixel_1$	$Pixel_2$	$Pixel_3$	$Pixel_4$	$Pixel_5$	$Pixel_6$	$Pixel_7$	$Pixel_8$	Pixel
254	10	10	10	10	10	8	9	192

Línea de pixeles sin codificar

Utilizando el algoritmo:

```
Escribir los datos de la cabecera y paleta de colores
Leer los datos bitmap desde la esquina inferior izquierda (Campo 8 con valor positivo)
Repetir
  Repetir
    Leer datos de la imagen (1 dato = 1 pixel). (En este caso solo se codificará el 254)
    Si hay por lo menos 3 datos que no se repitan (no las hay )
       Contar de hasta 255 datos posibles que no se repitan
       Escribir 0, la cuenta y los datos que no se repiten
       Si en total de la cuenta es valor impar
           Escribir 0
     Fin del si
    Si no
     Si hay 2 datos que no se repitan (no los hay)
          Escribir 1, el primer dato ,1 y el segundo dato
     Si no (solo hay un dato que no se repite)
          Contar de entre 1 a 255 datos posibles que se repitan.
          Escribir la cuenta y el dato que se repite (01 254)
      Fin del si
    Fin del si
    Leer datos de la imagen (1 dato = 1 pixel). (en este caso se codificarán los 5 valores de 10)
    Si hay por lo menos 3 datos que no se repitan (no los hay)
          Contar hasta 255 datos posibles que no se repitan
          Escribir 0, la cuenta y los datos que no se repiten
          Si en total de la cuenta es valor impar
             Escribir 0
          Fin del si
     Si no
          Si hay 2 datos que no se repitan (no los hay)
              Escribir 1, el primer dato ,1 y el segundo dato
           Si no (hay 5 datos repetidos)
              Contar de entre 1 a 255 datos posibles que se repitan.
              Escribir la cuenta y el dato que se repite (05 10)
          Fin del si
      Fin del si
    Leer datos de la imagen (1 dato = 1 pixel). (En este caso solo se codificarán el 8, 9, 192)
    Si hay por lo menos 3 datos que no se repitan
```

```
            Contar de hasta 255 datos posibles que no se repitan (Solo hay 3: 8, 9, 192)
            Escribir 0, la cuenta  y los datos que no se repiten  (0 3 8 9 192)
            Si en total de la cuanta es valor impar (3 es impar)
                Escribir 0  ( 0 )
             Fin del si
            Si no
                Si hay 2 datos que no se repitan
                    Escribir 1, el primer dato ,1 y el segundo dato
                Si no
                    Contar de entre 1 a 255 datos posibles que se repitan.
                    Escribir la cuenta y el dato que se repite
                 Fin del si
            Fin del si
    Hasta que sea un fin de línea
  Escribir 00 00
  Hasta leer todos los datos de la imagen
  Escribir 00 01
```

La codificación quedaría de la siguiente manera:

$Pixel_0$	$Pixel_1$	$Pixel_2$	$Pixel_3$	$Pixel_4$	$Pixel_5$	$Pixel_6$	$Pixel_7$	$Pixel_8$
254	10	10	10	10	10	8	9	192

Línea de pixeles sin codificar

Dato0	Dato1	Dato2	Dato3	Dato4	Dato5	Dato6	Dato7	Dato8	Dato9	Dato10
1	254	5	10	0	3	8	9	192	0	...

Línea de datos codificados

Como se puede observar la codificación del BMP no es lo bastante buena, si la imagen no tiene muchas repeticiones consecutivas.

Decodificación de un archivo BMP :

La decodificación de una archivo BMP es de manera semejante, se leen los datos de la cabecera para reconocer de que tipo de imagen vamos a tratar, cual es la longitud de esta y la lectura de la paleta. Una vez hecho esto nos preparamos para la decodificación. Tomemos el siguiente algoritmo.

```
Leer cabecera y  paleta de colores para  activarla
```

```
Repetir
  Leer dos bytes del archivo
  Si el primer valor es 0
    Si el segundo valor es 0
            Hay un fin de linea
    Si  el segundo valor es 1
            Hay un fin de archivo
    Si el segundo valor es 2
  Hay  un desplazamiento Delta
    Si el segundo valor es mayor o igual a 3
            Leer  tantos valores como indique el segundo valor y desplegarlos
            Si el segundo valor fue impar
                    Leer otro dato sin desplegarlo
            Fin del si
  Si no
    Desplegar el segundo valor tantas veces como indique el primer valor
  Fin del si
Hasta encontrar el termino de archivo ( 0 1)
```

RLE4

El algoritmo RLE de 4 bits (RLE4) almacena valores de pixel repetidos de una manera similar al RLE8. Los marcadores son los mismos. La única diferencia real es que dos valores de pixel son almacenados por 1 byte, y estos valores de pixel alternan cuando son decodificados. Por ejemplo, una cadena de datos codificada con RLE4 con valor 07 48 será decodificada a 7 pixeles, alternando en los valores 04 08 04 08 04 08 04.

Esto parece estrafalario, ya que raramente se ven series alternativas de valores de pixel en bitmap de 8 bits o más de ancho. Bitmap de 4 bits (16 colores), sin embargo, usualmente las contienen.

Una serie de 12 pixeles todas del mismo valor, 9, en RLE4 codifica como 0C 99 y serán codificada en la siguiente serie 09 09 09 09 09 09 09 09 09 09 09 09.

Un relleno es añadido para series de pixeles no codificados que están en número impar de bytes, de longitud. Por ejemplo, 6 pixeles con los valores 1 3 5 7 9 0 será almacenada como una serie no codificada 00 06 13 57 90 00, mientras que 5 pixeles con valores de 1 3 5 7 9 deben ser almacenados como una serie no codificada 00 05 13 57 90 00. Por ejemplo:

Bytes codificados	Bytes decodificados	Descripción
04 16	Cuatro valores alternativos de 1 y 6	01 06 01 06
08 44	Ocho valores alternativos de 4 y 4	04 04 04 04 04 04 04 04
03 E4	Tres valores alternativos de E y 4	0E 04 0E
00 06 12 A4 46 00	Seis valores de datos no codificados	01 02 0A 04 04 06

III.4 Formato PCX

Nombre	PCX
Conocido como	PC Paintbrush File Format (Formato de Archivo Paintbrush), DCX, PCC.
Tipo	Bitmap
Colores	Mono, 4 , 8 y 24 bits.
Compresión	RLE, Sin comprimir.
Tamaño Máximo de Imagen	64 Kb x 64 Kb pixeles.
Imágenes Múltiples por Archivo	No
Orden de Byte	Little-endian
Creador	ZSoft, Microsoft.
Plataforma	MS-DOS, Windows, UNIX y otros.
Aplicaciones que soporta	Demasiado numerosas para listarlas.

III.4.1 Antecedentes

PCX es uno de los formatos ampliamente usados. Creado para PC PaintBrush para MS-DOS por ZSoft, por lo cual, PCX se conoce en ocasiones como el formato de PC PaintBrush. ZSoft introdujo un dispositivo *OEM* con Microsoft, que permitió a Microsoft incluir PC Paintbrush con varios productos, incluyendo una versión llamada Microsoft Paintbrush para Windows, este producto fue distribuido con cada copia de Microsoft Windows que fue vendida. Esta distribución estableció la importancia de PCX.

El formato PCX original, versión 2.5 de PC Paintbrush, almacenaba gráficos e imágenes con no más de 16 colores, limitación de la tecnología de despliegue EGA producida por IBM. Cuando se introdujo la tarjeta de vídeo VGA, el formato PCX fue revisado y corregido para almacenar imágenes arriba de 256 colores.

La más reciente revisión de PCX hasta ahora incluye la habilidad para almacenar imágenes con 24 bits de color, permitiendo a PCX ser utilizado para almacenar imágenes creadas por la mayoría de las tecnologías gráficas avanzadas disponibles actualmente.

PCX es dependiente del hardware en el sentido de que fue originalmente diseñado para almacenar en un tipo de despliegue especifico. Sin embargo, los datos pueden ser almacenados por planos orientados (para la tarjeta EGA) o pixeles orientados para adecuar a la tarjeta de vídeo VGA.

Los datos de la imagen están codificados usando una variante del RLE el cual es simple y en ocasiones rápido en su operación. Como con otros esquemas del RLE, es difícil decir que tanto se reduce el tamaño de una imagen dada con el esquema de compresión PCX, ya que el factor de reducción depende del contenido de la imagen, es decir, que tan "saturada" está, y cuantos colores son utilizados. Generalmente si una imagen usa 16 o menos colores será reducida entre un 40 y 70% de los datos originales. En una imagen de 64 a 256 colores (de un scanner o vídeo) pueden ser reducidos solo entre un 10 y 30%. Es posible que una imagen sea tan compleja que el esquema de compresión PCX cause que el tamaño de los datos incremente después de la compresión.

III.4.2 Estructura

La figura III.9 muestra un diagrama a bloques de la estructura en un archivo con formato PCX.

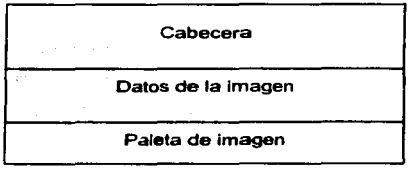


Figura III.9 Estructura del archivo PCX.

El archivo PCX esta organizado en tres partes: La cabecera, los datos de la imagen y la paleta de colores. La paleta de colores esta asociada con imágenes que contienen 256 colores para tarjetas VGA.

Estos datos de paleta se encuentran solo en versiones recientes del PCX (como la versión 3.0 plus).

Cabecera

La cabecera de el PCX se compone de 128 bytes distribuidos en los siguientes campos.

Campo 1. Identificador (1 byte, byte 0). Especifica un identificador del archivo. Su valor es de 10 (0Ah). Este valor no tiene significado real solo indica que el archivo es un PCX de Zsoft. Para los programadores puede ser útil ya que si no tiene este identificador, aunque el archivo tenga extensión PCX, se puede decidir que este no es archivo PCX y no procede su lectura.

Campo 2. Versión (1 byte, byte 1). Indica la versión del archivo. Zsoft pone en cada archivo una versión según las características de despliegue de cada PCX. En la versión 2.5 la información era considerada propiedad de ZSoft Corporación. En la tabla III.7 Se muestran los valores posibles de este campo.

Tabla III. 7 Versiones de archivos PCX.

Valor del Campo	Versión	Descripción.
0	Versión 2.5	Información de la paleta de colores EGA.
2	Versión 2.8	Información dentro del archivo de la paleta de colores EGA que se podía modificar.
3	Versión 2.8	Sin información de la paleta de colores.
4	Versión 3.0	Versión de PaintBrush para Windows, PC Paintbrush, PC Paintbrush Plus.
5	Versión 3.0 Plus	Versión plus de Paintbrush Plus para Windows, Publisher's Paintbrush, y todas las imágenes de 24 bits.

Campo 3. Tipo de Compresión (1 byte, byte 2). Indica el tipo de compresión utilizado en la imagen. El valor de este campo siempre es 1, Run-Length (RLE).

Campo 4. Bits Por Pixel (1 byte, byte 3).Indica el número de bits por cada pixel de la imagen. Los valores posibles son 1, 2, 4, 8 y 24 para 2, 4, 16, 256 y 16 millones de colores respectivamente.

Campo 5. Coordenada Mínima en X (2 bytes, bytes 4-5). Especifica la coordenada X de la esquina superior izquierda de la imagen.

Campo 6. Coordenada Mínima en Y (2 bytes, bytes 6-7). Especifica la coordenada Y de la esquina superior izquierda de la imagen.

Campo 7. Coordenada Máxima en X (2 bytes, bytes 8-9). Especifica la coordenada X de la esquina inferior derecha de la imagen.

Campo 8. Coordenada Máxima en Y (2 bytes, bytes 10-11). Especifica la coordenada Y de la esquina inferior derecha de la imagen.

Estos cuatro valores son las dimensiones rectangulares de la parte visible de la imagen (a veces llamada ventana de dimensión de la imagen). Se considera que la esquina superior izquierda de la pantalla es la coordenada (0,0), por lo tanto, la imagen debe contener como coordenadas mínimas en "x" y "y" un valor de '0', si estas coordenadas no son cero, entonces el despliegue de la imagen debe de comenzar en las coordenadas que se indiquen. Sin embargo, éste es un rasgo raramente apoyado por aplicaciones que utilicen PCX.

Campo 9. Resolución Horizontal (2 bytes, bytes 12-13). Contiene la resolución horizontal de donde se guardo la imagen.

Campo 10. Resolución Vertical (2 bytes, bytes 14-15). Contiene la resolución vertical de donde se guardo la imagen.

Las resoluciones Horizontal y vertical deben estar en dpi (dots per inches, puntos por pulgada).

Campo 11. Arreglo de paleta EGA (48 bytes, bytes 16-63). Especifica los elementos de la paleta EGA. Los 48 elementos para la versión que puede modificar su paleta (Solo en EGA de 16 colores se utiliza).Estos elementos constan de 8 bits cada uno y representan los 16 colores de 64 que puede tener el EGA (16 tonos de rojo, verde y azul).

3 tonos x 16 colores = 48 datos almacenados.

Campo 12. Reservado (1 byte, byte 64). En versiones anteriores este campo fue creado, al igual que el identificador, un campo que indique que el archivo es PCX, en algunas aplicaciones que utilizan PCX determinan que si este campo no es 0 el archivo no es PCX.

Campo 13. Bits por plano (1 byte, byte 65). Especifica el número de planos de color que contiene la imagen. El número de planos generalmente es 1, 3 ó 4 y se utilizan en conjunción con el campo 4 (bits por pixel), determina el modo de video para desplegar la imagen. Los modos de despliegue que utiliza se muestran en la tabla III.8.

Tabla III.8 Modos de video utilizados por PCX.			
Plano de colores	**Bits por pixel**	**Número de colores**	**Modo de video**
1	1	2	Monocromático
1	2	4	CGA
3	1	8	EGA
4	1	16	EGA y VGA
1	8	256	VGA extendido
3	8	16,777,216	VGA extendido y XGA

Número de colores = Plano de colores * (Bits por pixel) ^{2}

Campo 14. Bytes por línea (2 bytes, bytes 66-67). Indica el tamaño en bytes de un línea sin comprimir en un plano de color. Este valor es multiplicado por el numero de planos y se encuentra la longitud en bytes de una línea sin comprimir.

Línea = (Bytes Por Línea* Número de planos)

Campo 15. Tipo de Paleta (2 bytes,bytes 68-69). Contiene información sobre la paleta de colores. Un valor de '1' indica color o información monocromática, mientras un valor de '2' indica información en escala de grises. Este valor es realmente un indicador de si se debe desplegar la imagen en color o en escala de grises. (Sólo VGA es capaz de desplegar imágenes en escala de grises) PC PaintBrush y algunos otros programas que utilizan PCX ignoran este campo.

Campo 16. Tamaño de pantalla Horizontal (2 bytes, bytes 70-71). Indica el tamaño de la pantalla en sentido horizontal.

Campo 17. Tamaño de pantalla Vertical (2 byte, bytes 72-73). Indica el tamaño de la pantalla en sentido vertical.

Se agregaron los campos tamaño de pantalla horizontal y vertical en la estructura del PCX, para PC PaintBrush 4.0 y 4.0 Plus. Estos tamaños representan la resolución de la pantalla en donde se creo la imagen. Esto permite al gráfico que se despliegue y ajuste a su modo de vídeo. Ya que estos campos se agregaron después de las versión 3.0 de PanitBrush no hay manera de saber si estos campos contienen información válida.

Campo 18. Reservado (54 bytes, bytes 74-127 ó 58 bytes, bytes 70-127). Reservados, debe ser '0'. Su longitud depende de que existan los campos tamaño de pantalla vertical y horizontal. Se utiliza para rellenar la cabecera a 128 bytes para en futuras revisiones adicionar campos sin modificar la estructura.

Datos de la imagen

Campo 19. Datos de la imagen (Variable). Contiene la información de la imagen comprimida.

Paleta de colores

Campo 20. Separador (1 byte, variable). Especifica el separador entre los datos de la imagen y la paleta. Su valor es 12 (0Ch). Si hay paleta de colores integrada y esta imagen no es para modos EGA entonces hay un separador entre los datos de la imagen y la paleta de colores (Usado generalmente por el VGA en el modo de 256 colores).

Campo 21. Paleta de colores (768 bytes, últimos 768 bytes). Contiene la paleta de colores. Generalmente para la versión 5, estos datos contienen 256 tonos de cada uno de los 3 colores primarios (rojo, verde y azul), los colores se encuentran distribuidos de la siguiente manera R_1 V_1 A_1, R_2 V_2 A_2 ... R_n, V_n, A_n.

III.4.3 Tipos de imagen

Cada versión de PCX agregó nueva información al archivo, específicamente en la paleta de colores.

La compresión sigue siendo la misma, sus tipos de imagen radican esencialmente en los adaptadores de vídeo CGA, EGA y VGA y se diferencian por las versiones.

Versión 2.5.

La primera versión de PCX nunca utilizo una paleta de colores incluida en su archivo, la paleta de colores la tomaba de la paleta EGA actual de la computadora. Si se modificaba de alguna manera esta información de la maquina, el archivo PCX también modificaba sus colores, es decir, se modificaban ya que, los valores seguían siendo los mismos lo que cambiaba eran los tonos de colores que utilizaban cada valor. Por ejemplo, si el color numero 1 (Azul) tenia como valores en rojo, verde y azul (0,0,1) respectivamente, y el usuario modifica este color y pone en el los colores (1,1,1) el color 1 tendría como nuevo color el gris.

Versión 2.8.

Versiones que aparecieron más tarde de PCX podían trabajar con o sin una paleta de colores, esta versión era la 2.8 con paleta modificable y la versión 2.8 sin paleta modificable, aunque,

aparecieron con la misma versión el campo versión dentro de la cabecera hacia la diferencia, 2 para paleta incluida y 3 para paleta no incluida.

El campo 3 incluye dicha paleta en un arreglo de 48 elementos de un byte cada uno y su orden esta de la forma 16 RGB, o sea tripletas de los colores rojo, verde y azul, cada uno de estos en un rango de 0-255. Esta paleta puede tener 2, 4, 8 ó 16 colores en la imagen y los elementos restantes en los arreglos deben de estar en cero (00h). Como los adaptadores EGA cuentan con solo 4 colores para cada uno de los colores rojo, verde y azul (0 al 3), es necesario que estos valores se obtengan y al mismo tiempo cargarlos en la paleta EGA de la siguiente manera:

EgaColor0Rojo = PaletaEga[0] desplazando a la derecha 6 posiciones
(Ejemplo 255 desplazando 6 posiciones en bits es 11111111=255 Desplazando 6 seria 00000011=3)
EgaColor0Verde = PaletaEga[1] desplazando a la derecha 6 posiciones
EgaColor0Azul = PaletaEga[2] desplazando a la derecha 6 posiciones
EgaColor1Rojo = PaletaEga[3] desplazando a la derecha 6 posiciones
EgaColor1Verde = PaletaEga[4] desplazando a la derecha 6 posiciones
EgaColor1Azul = PaletaEga[5] desplazando a la derecha 6 posiciones
y así sucesivamente.

Paleta CGA 4-colores

Si se utiliza el mismo arreglo EGA contenido en la imagen para desplegar a una imagen CGA que utiliza en cada color RGB uno de 8 los posibles, desplegándose imágenes de cuatro colores 3 de primer plano y uno de fondo. Los cuatro bits mas significativos del primer byte del EGA contiene el color de fondo y esta en el rango de 0-15.Los 3 bits mas significativos del cuarto byte del EGA contiene el color de primer plano. Estos tres bits corresponden al tipo de color, la paleta y la intensidad como se muestra abajo:

Tipo de color Bit(7)	Paleta Bit(6)	Intensidad Bit (5)
Color	Amarillo	Normal
0	0	0
Monocromático	Blanco	Luminoso
1	1	1

Es necesario que estos valores se obtengan y al mismo tiempo cargarlos en la paleta CGA de la siguiente manera:

/* se toma primero el color de fondo */
CgaColorFondo = PaletaEga[0] desplazando hacia la derecha 4 posiciones (del 0 al 15)
/* Color de primer plano CGA*/
CgaTipoColor = PaletaEga[3] tomar el bit mas significativo posición 8 (0 ó 1)
CgaValorPaleta = PaletaEga[3] tomar el bit que esta en la posición 7 (0 ó 1)
CgaIntensidad = PaletaEga[3] tomar el bit que esta en la posición 6 (0 ó 1)

Versión 3.0 y 3.0 plus.

Paleta de 256-colores

Cuando PCX ya era un formato utilizado por muchos, comenzaron a ser muy pobres los colores que proporcionaba el EGA, ya que solo se podían desplegar 16 colores de una paleta de 64.

La aparición de la tecnología EGA en 16 colores en 1984, dio pie a la nueva tecnología VGA de 256 colores que apareció en 1987. PCX ahora podía contener hasta 256 colores de una paleta de 262, 144, pero se tendría un inconveniente, el cual consistía en tener que aumentar la estructura del archivo para el nuevo formato de imágenes VGA. Los diseñadores optaron por añadir la paleta de colores al final del archivo, pero, ¿como saber en donde debía de comenzar?, ya que no todas la imágenes eran del mismo tamaño una vez comprimidas, esto ocasiono confusión pero se soluciono de una manera muy sencilla: la lectura de esta paleta ya no seria del principio del archivo sino del final del mismo. Como existían imágenes que no llevaban incluidas la paleta de colores aun siendo de la versión 3.0, tuvo que ser necesario grabar un byte que identificaba el inicio de la paleta y entonces en vez de leer 768 bytes (256 colores de cada elemento RGB), se leerían 769 en donde el primer valor de esos 769 seria 12 (0Ch) y los siguientes serian los valores de los colores RGB. PCX especifica que si se trata de la versión 3.0 ó 3.0 plus (Campo 2=5) puede tener una paleta incluida al final del archivo. Normalmente, un archivo PCX debe tener una paleta VGA si hay mas de 16 colores en la imagen, sin embargo, hay muchos programas que trabajan con archivos PCX de la versión 3.0 sin incluir una paleta de colores, mientras que otros programas siempre incluyen una paleta VGA al archivo aunque este utilice solo 2 colores en la imagen, esto confunde las cosas, y aun más cuando se habla de imágenes de 24 bits que nunca tendrán una paleta incluida, aunque estas sean de la versión 3.0 plus.

Un archivo PCX de la versión 3.0 ó 3.0 plus no necesita una paleta incluida, aun cuando coincida el número 12 (0Ch) en la posición 769 antes de finalizar el archivo, si es así, el lector PCX

interpretaría los últimos 768 bytes de la imagen como datos de una paleta VGA, y seria más raro aun, si esta imagen se desplegará bien. Una solución a este problema seria leer primero todos los datos de la imagen y notar si el indicador del archivo se detuvo en la posición 769 antes del final del archivo. Si es así, entonces el archivo PCX contiene una Paleta VGA de 256 colores. Otro método seria si, además de leer los datos de la cabecera en el campo versión también verificar si el valor 769 antes del final del archivo es 12 (0Ch).

III.4.4 Compresión

El algoritmo de compresión usado en el PCX es un esquema simple de RLE (Run-Length-Encoding) de 1 byte y/o 2 bytes. Aunque este tipo de compresión no es lo mas eficaz en la reducción de datos, es muy rápido en su funcionamiento y bastante fácil de llevar a cabo.

Una imagen normalmente contiene una serie de líneas de pixeles, en donde algunos datos están repetidos (dos o mas pixeles con el mismo valor). Al utilizar el RLE es posible poner en tan solo dos bytes la representación de toda una línea, un byte contiene el numero de pixeles que se repetirán mientras que el otro contiene el pixel que se repite.

$Pixel_0$	$Pixel_1$	$Pixel_2$	$Pixel_3$	$Pixel_4$	$Pixel_5$	$Pixel_6$	$Pixel_7$	Pixel
254	10	10	10	10	10	10	8	.....

Línea de pixeles sin codificar

PCX utiliza RLE pero con una pequeña variación, esta variación consiste en utilizar dos o un byte para codificación. PCX utiliza los dos bits más significativos de un byte para representar una repetición de datos o solo un dato, es decir, PCX solo puede hacer una compresión de datos pero sin que esta rebase los 63 datos consecutivos. Por ejemplo:

Bytes codificados	Bytes decodificados	Descripción
198 16	16 16 16 16 16 16	Seis bytes con valor 16
23 65 34 56 193 200	23 65 34 56 200	Cuatro bytes no codificados y uno codificado
3 197 10	3 10 10 10 10 10	Un byte 3 y 5 bytes con valor 10
3 4 194 11	3 4 11 11	Un byte 3, un 4 y 2 bytes con valor 11

A continuación se presenta un algoritmo que muestra los pasos a seguir para comprimir una imagen en formato PCX:

- Escribir los datos de la cabecera

```
Comenzar a leer desde el principio de la imagen
Repetir
   Leer un dato la imagen (dato = pixel).
   Contar hasta un máximo de 63 datos posibles que se repitan.
            Si no hay Repetición
                     Si el dato leído es mayor o igual a 192
                              Escribir  193 y el dato leído
                     Si no
                              Escribir el dato leído
                     fin del Si
            Si no
                     Escribir  (192 + las veces que se repite)  y  el dato leído
            Fin del si
Hasta leer todos los datos de la imagen
```

Nota: Los datos de la imagen deben ser leídos desde la esquina superior izquierda hasta la esquina inferior derecha, y en orden de izquierda a derecha.

Codifiquemos la siguiente línea de datos utilizando el algoritmo de compresión que utiliza PCX

$Pixel_1$	$Pixel_2$	$Pixel_3$	$Pixel_4$	$Pixel_5$	$Pixel_6$	$Pixel_7$	$Pixel_8$	Pixel
254	10	10	10	10	10	10	192	192

Línea de pixeles sin codificar

Utilizando el algoritmo:

```
Comenzar desde el inicio de la línea
Repetir
Leer un dato de la imagen (254 ).
   Contar hasta un máximo de 63 datos posibles que se repitan (no hay repetición ).
            Si no hay Repetición
                     Si el dato leído es mayor o igual a 192  (254 es mayor que 192)
                              Escribir  193 y el dato leído  (Escribir 193 y  254 )
                     Si no
                              Escribir el dato leído
                     fin del Si
            Si no
                     Escribir  (192 + las veces que se repite)  y  el dato leído
            Fin del si
```

```
Leer el siguiente dato de la imagen (10).
  Contar hasta un máximo de 63 datos posibles que se repitan (hay 6 repeticiones ).
        Si no hay Repetición
                Si el dato leído es mayor o igual a 192
                        Escribir  193 y el dato leído
                Si no
                        Escribir el dato leído
                fin del Si
        Si no
      Escribir (192 + las veces que se repite)  y  el dato leído  (Escribir 192+6=198 y 10)
        Fin del si
Leer el siguiente dato de la imagen (192).
  Contar hasta un máximo de 63 datos posibles que se repitan (hay 2 repeticiones ).
        Si no hay Repetición
                Si el dato leído es mayor o igual a 192
                        Escribir  193 y el dato leído
                Si no
                        Escribir el dato leído
                fin del Si
        Si no
        Escribir  (192 + las veces que se repite)  y  el dato leído  (192+2=194 y 192)
        Fin del si
Hasta leer toda los datos de la línea
```

La codificación quedaría de la siguiente manera:

$Pixel_0$	$Pixel_1$	$Pixel_2$	$Pixel_3$	$Pixel_4$	$Pixel_5$	$Pixel_6$	$Pixel_7$	$Pixel_8$
254	10	10	10	10	10	10	192	192

Línea de pixeles sin codificar

Dato0	Dato1	Dato2	Dato3	Dato4	Dato5	Dato6	Dato7	Dato8
193	254	198	10	194	192			

Línea de datos codificados

Como se puede observar la codificación del PCX no es lo bastante buena pero si la imagen tiene muchas repeticiones consecutivas el método de compresión puede ser bastante aceptable.

Decodificación de un archivo PCX :

La decodificación de un archivo PCX es de manera semejante, primeramente leeremos los datos de la cabecera para reconocer de que tipo de imagen vamos a tratar, cual es la longitud de esta y la lectura de la paleta. Una vez hecho esto nos preparamos para la decodificación. Tomemos el siguiente algoritmo.

```
Leer cabecera
Calculemos el ancho y alto de la imagen  como sigue:
                    Ancho = (Campo 7) - (Campo 5)+ 1 ( Ancho de la imagen en pixeles )
                    Alto    = (Campo 8) - (Campo 6)+ 1  (Alto de la imagen)
Leer paleta de colores y activarla
Hacer cont_pixel=0
Colocarnos en la posición 128 del archivo e iniciar la lectura.
Repetir
 Leer un byte del archivo
 Si el byte es mayor de 192
          Hacer Byte_cont = byte - 192
          leer siguiente byte del archivo
          Poner byte_cont veces el byte leído (pixel)
 Si no
          Poner el byte leído (pixel)
 Fin del si
Hasta que se hayan puesto (Ancho * Alto) pixeles
```

Existe otra forma de codificación en la cual se toman en cuenta los bits por línea (Campo 14) y planos de colores (Campo 13), además de la codificación de archivos PCX de 24 bits utilizando 3 bytes por cada color.

Formatos de archivo relacionados con PCX

Varios formatos utilizan la misma codificación que utiliza PCX..

Archivo de formato PCC

Archivo que era un PCX pero tenia la capacidad de copiar una área del PCX y usar esta copia dentro del archivo mismo.

Archivo de formato DCX

Archivo PCX con múltiples imágenes incluidas. El archivo DCX almacena hasta 1023 imágenes PCX dentro de un solo archivo. Cada imagen en el DCX es un PCX completo incluyendo la cabecera y la información de la cabecera. El formato DCX es bastante conveniente y muy fácil usar; sin embargo, éste formato padece un inconveniente. Cuando una serie de archivos PCX son encadenados en un DCX, toda la información de los PCX se guarda, pero los nombres reales del PCX se pierden.

III.5 Formato GIF

Nombre	GIF
Conocido como	Graphics Interchange Format (Formato de intercambio de gráficos).
Tipo	Bitmap
Colores	1 a 8 bits.
Compresión	LZW.
Tamaño Máximo de Imagen	64 Kb x 64 Kb pixeles.
Imágenes Múltiples por Archivo	Sí.
Orden de Byte	Little-endian
Creador	CompuServe, Inc.
Plataforma	MS-DOS, Macintosh, UNIX, Amiga, y otros.
Aplicaciones que soporta	Demasiado numerosas para listarlas.

III.5.1 Antecedentes

GIF es una creación de CompuServe usado para almacenar imágenes bitmap múltiples en un solo archivo para intercambiarse entre plataformas y sistemas. En términos de número de archivos en existencia, GIF es quizás el más usado para almacenamiento de gráficos multibit y datos de imagen, por ejemplo, archivos gráficos de los *BBSs*. Algunos de estos archivos son imágenes de alta calidad de gentes, carros, astros, entre otros. Las librerías *Shareware* y BBSs son archivos con imágenes GIF.

La mayoría de los archivos GIF contienen 16 ó 256 colores con calidad de imagen casi de fotografía. Imágenes en escala de grises, producidas por scanners, también son comúnmente almacenadas utilizando el formato GIF, sin embargo, gráficos monocromáticos tales como archivos Clip Art e imágenes de documentos, raramente lo utilizan.

Sin embargo, la mayoría de los archivos GIF los encontramos en el ambiente de MS-DOS, empero, GIF no está asociado con alguna aplicación en particular ni fue creado para alguna en especial, pero la mayoría de las aplicaciones que leen y escriben datos de imágenes, tales como programas de dibujo, scanners y Software de video, los principales programas de despliegue y conversión de imágenes, soportan GIF. GIF fue introducido para permitir un fácil intercambio y despliegue de datos de imágenes almacenados en computadoras locales o remotas.

El formato GIF es diferente de algunos otros formatos bitmap comunes en el sentido que es *stream-based* (basado en cadenas). Consiste de una serie de paquetes de datos, llamados bloques, acompañados de información adicional de protocolo. A causa de este acuerdo, los archivos GIF deben ser leídos como si fueran una cadena continua de datos. Varios bloques y sub-bloques de datos definidos por GIF pueden ser encontrados al menos en cualquier parte dentro del archivo. Esta incertidumbre hace imposible encapsular cada posible arreglo de datos GIF en la forma de estructuras de C.

Hay un número diferente de categorías de bloques de datos, y cada uno de los varios bloques definidos cae en alguna de estas categorías. En la terminología de GIF, un bloque de Control de Extensión de Gráficos es un tipo de bloque de Control Gráfico. De forma semejante, un bloque de Extensión de Texto sencillo y un Descriptor de Imagen Local son tipos de bloques de Extensión de Comentarios e Interpretación (rendering) de Gráficos. Los datos bitmap están en un bloque de Datos de Imagen. Los bloques de Extensión de Comentarios y Extensión de Aplicaciones son tipos de bloques de Propósito Especial.

Los bloques, además de almacenar campos de información, pueden contener sub-bloques. Cada dato del sub-bloque inicia con un byte simple de cuenta, que puede estar en el rango de 1 a 255 e indica el número de bytes de datos que le siguen al byte de cuenta. Múltiples sub-bloques pueden aparecer en un grupo contiguo (byte de cuenta, bytes de datos, byte de cuenta, byte de datos, y así sucesivamente). Una secuencia de uno o más sub-bloques es terminada por una byte de cuenta con un valor de '0'.

El formato GIF es capaz de almacenar datos bitmap con anchos de pixel de 1 a 8 bits. Las imágenes son siempre almacenadas usando un modelo de color RGB y datos de la paleta de colores. GIF es también capaz de almacenar imágenes múltiples por archivo, pero esta habilidad es raramente utilizada, y la vasta mayoría de archivos GIF contienen solo una imagen. La mayor parte de los lectores de archivos GIF, en realidad, no soportan el despliegue de archivos GIF con imágenes múltiples o pueden desplegar solo la primera imagen almacenada en el archivo.

Los datos de imagen almacenados en archivos GIF siempre están comprimidos con LZW. Este algoritmo reduce cadenas de bytes con valores idénticos en un código de palabra y es capaz de reducir el tamaño de datos de 8 bits por pixel en un 40% o más. La habilidad de almacenar datos sin

comprimir, o datos codificados usando un algoritmo de compresión diferente, no es soportada en la versión actual del formato GIF.

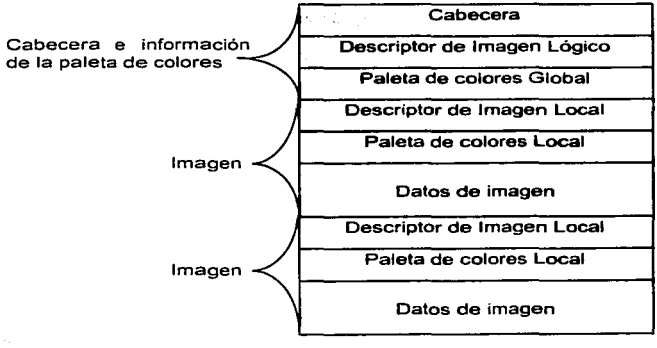
Existen dos revisiones de la especificación GIF, que han sido ampliamente distribuidas. La revisión original fue GIF87a, y algunas imágenes fueron creadas en éste formato. La revisión actual, GIF89a, adiciona varias capacidades, incluyendo la habilidad para almacenar datos de texto y gráficos en el mismo archivo.

III.5.2 Estructura

III.5.2.1 GIF87a

Esta versión fue introducida en Mayo de 1987 y es leída por la mayoría del Software que soporta el formato GIF.

La figura III.10 ilustra el esquema básico de un archivo GIF87a. Cada archivo inicia con una cabecera y un Descriptor de Imagen Lógico. Una paleta de colores Global puede aparecer opcionalmente después del descriptor de imagen. Cada una de estas 3 secciones es siempre encontrada en el mismo desplazamiento desde el inicio del archivo. Cada imagen almacenada en el archivo contiene un descriptor de imagen local, una paleta de colores local opcional, y un bloque de datos de imagen. El último campo en un archivo GIF es un caracter terminador, que indica el fin de la cadena de datos GIF.



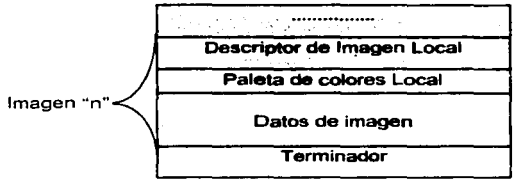


Figura III.10 Estructura del archivo GIF87a.

Cabecera

Contiene 2 campos con una longitud de 6 bytes en total. Identifica al archivo como tipo GIF.

Campo 1. Signatura (Firma) (3 bytes, bytes 0-2). Especifica el tipo de archivo. Debe ser "GIF" o 71 73 70. Todos los archivos GIF inician con estos 3 bytes, quien no lo tenga no será leído como un archivo GIF.

Campo 2. Versión (3 bytes, bytes 3-5). Especifica la versión del formato GIF, "87a" ó "89a".

Descriptor de imagen lógico

Contiene información de la pantalla y color usada para crear y desplegar la imagen.

Campo 3. Ancho de pantalla (2 bytes, bytes 6-7). Especifica el ancho en pixeles del despliegue en pantalla.

Campo 4. Largo de pantalla (2 bytes, bytes 8-9). Especifica el largo en pixeles del despliegue en pantalla.

Los campos 3 y 4 indican la mínima resolución requerida para desplegar los datos de la imagen. Si el dispositivo de despliegue no es capaz de soportar la resolución especificada, algún tipo de escala será necesaria para desplegar la imagen adecuadamente.

Campo 5. Paquete (1 byte, byte 10). Especifica la información de la pantalla y paleta de colores. Contiene los siguientes subcampos de datos, el bit 0 es el menos significativo.

Campo 5.1 Bits 0-2. Tamaño de la paleta de colores global. Número de bits en cada entrada de la paleta menos 1. Por ejemplo, si una imagen contiene 8 bits por pixel, el valor de este campo es 7. El número total de elementos en la paleta de colores global es calculado de la siguiente forma:

No entradas paleta global = 1 desplazando (paleta global + 1) veces hacia la izquierda

El tamaño de la paleta esta siempre puesto al tamaño apropiado incluso si no hay paleta de colores global, es decir, si tipo de bandera está puesto en ´0´.

Campo 5.2 Bit 3. Tipo de bandera de la paleta de colores. Si este subcampo está puesto en ´1´, entonces las entradas de la paleta de colores están almacenadas del más significativo al menos significativo. Clasificar los colores en la paleta ayuda a la aplicación a elegir los colores a usar con el hardware de despliegue que tiene menor número de colores disponibles que los datos de la imagen. Esta bandera es solo válida para la versión "87a", es decir, para GIF "89a" este subcampo está reservado y, por lo tanto, puesto en ´0´.

Campo 5.3 Bits 4-6. Resolución del color. Indica el número de bits en una entrada de la paleta de colores original menos 1. Este valor iguala al máximo tamaño de la paleta de colores original. Por ejemplo, si una imagen originalmente contenía 8 bits por color primario, el valor de éste campo deberá ser 7.

Campo 5.4 Bit 7 . Tipo de bandera de la paleta de colores global. Si este valor es ´1´ hay paleta de colores global, por el contrario, si es ´0´ no hay paleta de colores global presente en el archivo GIF. Los datos de la paleta de colores global siempre le siguen al descriptor de pantalla lógico en un archivo GIF.

Campo 6. Color de fondo (1 byte, byte 11). Especifica el índice del color de la paleta de colores global a usar para los bordes y fondo de la imagen. El fondo es considerado como el área en pantalla no cubierto por la imagen. Si no hay paleta de colores global, éste campo no se usa y es ignorado.

Campo 7. *Aspect Ratio* (1 byte, byte 12). Especifica el valor del aspect ratio de los pixeles en la imagen. El aspect ratio es el ancho del pixel dividido por el alto del pixel. Este valor cae en un rango de 1 a 255 y es utilizado en el siguiente cálculo:

$$\text{Aspect ratio del pixel} = (\text{aspect ratio} + 15) / 64$$

Si el campo está puesto en ´0´, entonces no hay aspect ratio especificado.

Paleta de colores global

El descriptor de imagen lógico puede ser seguido por una paleta de colores global opcional. Esta paleta de colores, si está presente, es la paleta de colores utilizada para los índices de color de pixel de los datos contenidos en los datos de imagen. Si no hay paleta de colores global, cada imagen almacenada en el archivo contiene una paleta de colores local que se utiliza en lugar de la

paleta de colores global. Si cada imagen en el archivo usa su propia paleta, entonces no hay paleta de colores global. Si no existe ni paleta de colores global ni local, asegúrese de que su aplicación facilite un paleta de colores por default para ser utilizada. Se sugiere que la primera entrada de paleta por default sea el color negro y la segunda entrada el color blanco.

Los datos de la paleta de colores global siempre le siguen a la información del descriptor de imagen lógico y varia en tamaño dependiendo del número de entradas en la paleta. La paleta de colores global es una serie de tripletas de 3 bytes para cada elemento en la paleta. Cada tripleta contiene el valor del color primario rojo, verde y azul de cada elemento en paleta.

El número de entradas en la paleta es siempre una potencia de 2 (2, 4, 8, 16,...) hasta un máximo de 256 entradas. Como ya se hizo mención, el tamaño en bytes de la paleta global es calculado usando los bits 0, 1 y 2 del campo 5, paquete, de la siguiente manera:

Tamaño paleta = 3 * (1 desplazando (paleta global + 1) veces hacia la izquierda)

La cabecera, el descriptor de imagen lógico y los datos de la paleta de colores global son seguidos de una o más secciones de datos de imagen. Cada imagen en un archivo GIF es almacenada separadamente, con un descriptor de imagen y posiblemente una paleta local. El descriptor de imagen es similar a una cabecera y contiene información solo acerca de los datos de imagen que le siguen inmediatamente. La paleta de colores local contiene la información de color especifica solo para los datos de esa imagen y puede o no estar presente.

Descriptor de imagen local

El descriptor de imagen local aparece antes de cada sección de datos de imagen y tiene la siguiente estructura:

Campo 8. Separador (1 byte, variable). Especifica un identificador de descriptor de imagen local. Contiene el valor 2Ch "," (coma) e indica el inicio de un bloque de datos de descriptor de imagen local.

Campo 9. Izquierda (2 bytes, variable). Posición en X de la imagen es su dispositivo de despliegue.

Campo 10. Arriba (2 bytes, variable). Posición en Y de la imagen es su dispositivo de despliegue. Los campos 9 y 10 son las coordenadas en pixeles de la esquina superior izquierda de la imagen en la pantalla lógica. La esquina superior izquierda es considerada como la coordenada (0,0).

Campo 11. Ancho (2 bytes, variable). Especifica el ancho de la imagen en pixeles.

Campo 12. Largo (2 bytes, variable). Especifica el largo de la imagen en pixeles.

Los campos 11 y 12 son el tamaño de la imagen en pixeles.

Campo 13. Paquete (1 byte, variable). Contiene los siguientes subcampos de datos, el bit 0 es el menos significativo.

Campo 13.1 Bit 0. Bandera de la paleta de colores local. Si es '1' existe una paleta de colores local asociada con esta imagen, en caso contrario, no la hay y, la paleta de colores global deberá ser utilizada.

Campo 13.2 Bit 1. Bandera de Interlazado. Si este subcampo está puesto en '1', entonces la imagen es interlazada, de lo contrario, si esta en '0', es no interlazada.

Campo 13.3 Bit 2. Bandera de clasificación. Indica si las entradas en la paleta de colores han sido clasificadas por su orden de importancia. Su importancia es usualmente decidida por la frecuencia de ocurrencia del color en los datos de la imagen. Un valor de '1' significa que la paleta almacenada está clasificada, un valor de '0' aparece si paleta no está clasificada. Este campo es válido solamente para la versión "89a". En GIF "87a", este campo es reservado y debe estar puesto en '0'.

Campo 13.4 Bits 3-4. Reservados. Deben estar puestos en '0'.

Campo 13.5 Bits 5-7. Tamaño de entradas en la paleta de colores local. Indica el número de bits por entrada en la paleta local. Si el valor del subcampo 5.1 bandera de la paleta de colores local es '0', entonces este subcampo es también '0'.

Paleta de colores local

Si una paleta de colores local está presente, le sigue inmediatamente al descriptor de imagen local y le precede a los datos de la imagen con que está asociada. El formato de todas las paletas locales es idéntico al de la paleta de colores global. Cada elemento es una serie de tripletas de 3 bytes que contienen el valor del color primario rojo, verde y azul de cada elemento en la paleta local.

El número de entradas y el tamaño en bytes de la paleta local es calculado del mismo modo que la paleta global:

Tamaño paleta = 3 * (1 desplazando (paleta global + 1) veces hacia la izquierda)

No. entradas paleta global = 1 desplazando (paleta global + 1) veces hacia la izquierda

Una paleta de colores local solo afecta a la imagen con quien está asociada y, si está presente, sus datos suplantan a los de la paleta global.

Datos de imagen

Los datos de imagen encontrados en cada archivo GIF están siempre comprimidos usando el esquema de codificación LZW (Lempel-Ziv-Welch), que generalmente se utiliza en compactadores (pkzip y zoo). Comprimir un archivo GIF mediante tales compactadores es una operación redundante, que raramente resulta en archivos más pequeños y usualmente no vale la pena el tiempo y esfuerzo envuelto en el intento.

Usualmente cuando son almacenados datos codificados con LZW en un archivo de formato gráfico, es organizado como una cadena continua de datos que son leídos desde el inicio al fin. El formato GIF, sin embargo, almacena datos de imágenes codificadas como una serie de sub-bloques de datos.

Cada sub-bloque de datos inicia con un byte de cuenta. El valor del byte de cuenta toma el rango de 1 a 255 e indica el número de bytes de datos en el sub-bloque. El bloque de datos le sigue inmediatamente al byte de cuenta. Un grupo contiguo de bloque de datos es terminado por un byte con valor '0'. Este puede ser visto como un valor terminador o como un sub-bloque con un byte de cuenta de '0', en ambos casos, incida que no le siguen bytes de datos.

Ya que los archivos GIF no contienen cadenas contiguas de datos codificados con LZW, cada sub-bloque debe ser leído y enviado al decodificador LZW. La mayoría de los sub-bloques almacenan datos de imagen con 255 bytes de longitud, este es un excelente tamaño máximo para el uso del buffer que retendrá el dato de imagen codificado. También, el proceso de codificación mantiene rastros de cuando cada línea escaneada inicia y termina. Por lo tanto, es probable que una línea escaneada terminará y otra inicia a mitad del sub-bloque de datos de imagen.

El formato de decodificación de datos GIF no es sencillo. Cada pixel en una línea escaneada decodificada siempre es de un byte y contiene un valor del índice en la paleta local o global. Aunque la estructura del formato GIF es completamente capaz de almacenar información de color directamente en los datos de imagen (de este modo evita la necesidad de una paleta de colores), la especificación GIF no indica este hecho como una posible opción. Por lo tanto, incluso un dato de imagen de 1 bit debe usar valores de índice de 8 bits y una paleta de 2 entradas.

Los datos de imagen GIF están siempre almacenados por línea escaneada y por pixel. GIF no tiene la capacidad para almacenar datos de imagen como planos, así que cuando archivos GIF son desplegadas usando dispositivos de despliegue por planos orientados, máscaras de datos de imagen deben ocurrir primero antes de que la imagen pueda ser desplegada.

Las líneas escaneadas compensan los datos de imagen bitmap GIF que normalmente son almacenados en orden consecutivo, iniciando con la primera línea y terminando en la última. El formato GIF también soporta una forma alternativa de almacenar líneas de datos bitmap en un orden interlazado. Imágenes interlazadas son almacenadas como líneas alternativas de datos bitmap. Si alguna vez ha visto un archivo GIF que aparece en pantalla como una serie de 4 "limpiadas", que saltan a través de la pantalla, cuando la imagen fue desplegada, estuvo viendo un archivo GIF interlazado.

La figura III.11 compara el orden de las líneas almacenadas en un formato interlazado y no interlazado. En el formato no interlazado, las líneas de los datos bitmap son almacenados iniciando con la primera líneas y continuando secuencialmente hasta la última línea. Este es el formato típico de almacenamiento para la mayoría de los archivos de formato bitmap. El formato interlazado, sin embargo, almacena las líneas fuera de la secuencia normal. Todas las líneas pares son almacenadas primero y todas las líneas impares son almacenadas al final. Se puede observar que cada paso sucesivo usualmente codifica más líneas que el paso previo.

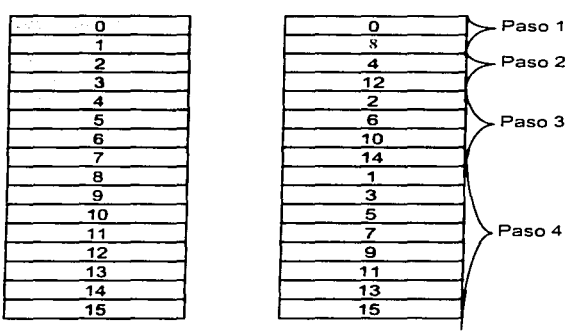


Figura III. 11 Organización de líneas escaneadas interlazadas y no interlazadas.

GIF utiliza un esquema de interlazado de 4 pasos. El primer paso inicia en la línea 0 y lee cada 8 líneas de datos bitmap. El segundo paso inicia en la cuarta línea y lee cada 8 líneas de datos. El tercer paso inicia en la segunda línea y lee cada 4 líneas. El último paso inicia en la primera línea y lee cada 2 líneas. Usando este esquema, todas las líneas de los datos bitmap son leídas y almacenadas.

¿Por qué interlazar una imagen GIF ? Interlazar puede parecer hacer la lectura, escritura y despliegue de una imagen más difícil, y claro que lo es. ¿Este arreglo hace de alguna manera más fácil el despliegue en monitores interlazados?

GIF fue diseñado como un protocolo de comunicación de imágenes usado para ver interactivamente imágenes en línea. Un usuario conectado a un servicio de información vía módem puede no solo "bajar" una imagen GIF, también puede ver como aparece en su pantalla como fuera siendo "bajada". Si una imagen GIF fue almacenada en un formato no interlazado, la imagen GIF podrá desplegarse en forma progresiva iniciando en la parte superior de la pantalla y terminando en la parte inferior. Después de que un 50% del "bajado" de la imagen se ha completado, solo la mitad de la parte superior de la imagen será visible. Una imagen interlazada, sin embargo, podrá desplegar iniciando con cada octava línea, cada cuarta línea, cada segunda línea, y así sucesivamente. Cuando se ha completado un 50%, el contenido de la imagen completa será discernido aunque solo la mitad de la imagen ha sido desplegada. La otra mitad la llenará el cerebro con el vistazo hecho.

El interlazado presenta problemas cuando convierte de una imagen GIF a algún otro formato. Puede ser creada una tabla de líneas escaneadas para escribir las líneas escaneadas en su propio orden no interlazado. Utilizamos 2 tablas, la primera contiene el mapeo de líneas escaneadas no interlazadas, con los valores de la primera columna de la figura III.11 (del 0 al 15) en orden consecutivo. La segunda tabla contiene el código interlazado, segunda columna de la figura III.11.

Podemos almacenar la imagen no interlazada mediante los valores almacenados en la segunda tabla. La línea 0 de la imagen no interlazada es la línea 0 de la imagen interlazada. La primera línea de la imagen no interlazada es la octava línea de la imagen interlazada. La segunda línea de la imagen no interlazada es la cuarta línea de la imagen interlazada, y así sucesivamente.

Terminador

El terminador (avance) es un byte sencillo de datos que aparece como el último caracter en el archivo. Este valor siempre es 3Bh "55" e indica el fin de la cadena de datos. Un terminador debe aparecer en cada archivo GIF.

III.5.2.2 GIF89a

La versión "89a" es la más reciente de GIF y fue introducida en Julio de 1989. Aunque el formato GIF89a es muy similar al GIF87a, contiene algunos bloques adicionales de información no definida en la especificación "87a". Por ésta razón los archivos GIF89a no pueden ser leídos y desplegados adecuadamente por aplicaciones que solo leen archivos GIF87a. Algunos de estos programas no contemplan desplegar imágenes de archivo "89a", por lo que ésta versión no será reconocida. Aunque cambiar el número de versión de "87a" a "89a" resolverá este problema, los datos de imagen pueden continuar sin ser desplegados correctamente, por razones que se verán más adelante.

La figura III.12 ilustra el esquema básico del archivo GIF89a. Tal como el "87a" también inicia con una cabecera, un descriptor de pantalla lógico y, una paleta de colores global. Cada imagen también contiene un descriptor de imagen local, una paleta local opcional y un bloque de datos de imagen. El terminador en cada archivo "89a" contiene el mismo valor del terminador del "87a".

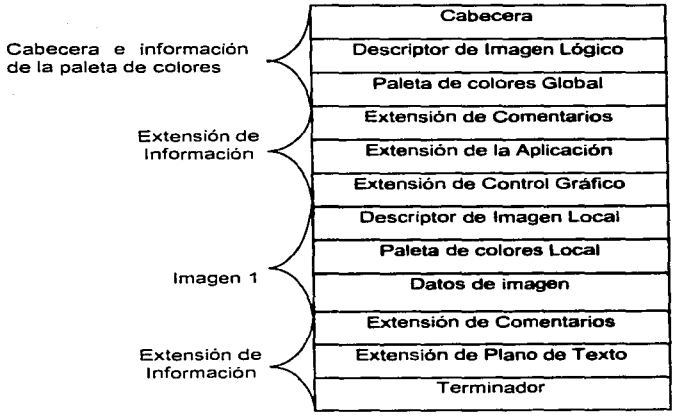


Figura III. 12 Estructura del archivo GIF89a.

La versión "89a" añade una nueva característica al formato llamada Extensiones de Control. Estas extensiones a la versión "87a" son bloques especializados de información utilizada para el control de la interpretación de los datos gráficos almacenados en el archivo. El diseño del "87a" solo permitía el despliegue de imágenes una a la vez en una forma "slide show" (presentación de diapositiva). Aunque la interpretación y uso de los datos de la extensión de control, permite al GIF89 que los datos gráficos basados en bitmap y textuales sean desplegados, sobrepuestos y borrados como una presentación de multimedia animada.

Las cuatro extensiones de control introducidas por GIF89a son: Extensión de Control Gráfico, Extensión de Plano de Texto, Extensión de comentarios y Extensión de la Aplicación.

El bloque de extensión de control gráfico controla como el dato bitmap y el texto encontrados en un bloque de interpretación del gráfico son desplegados. Puede incluir si el gráfico va a ser sobrepuesto en una transparencia o forma opaca sobre otro gráfico, si el gráfico es para ser restaurado o borrado, y si una entrada del usuario es esperada antes de continuar con el despliegue del dato.

El bloque de extensión de plano de texto permite mezclar gráficos ASCII plano de texto con datos de imagen bitmap. Algunas imágenes GIF contienen texto legible al humano que es actualmente parte del dato bitmap en sí mismo. Utilizando la extensión plano de texto, las leyendas que no son actualmente parte de la imagen bitmap pueden ser sobrepuestas en la imagen. Esto es invaluable cuando es necesario desplegar datos textuales sobre una imagen, pero es inconveniente alternar el bitmap para incluir esta información. Incluso es posible construir un archivo "89a" que contenga solo datos plano de texto y no solo datos de imagen bitmap.

El bloque de extensión de comentarios contiene texto ASCII incrustado en la cadena de datos GIF en una manera similar a los comentarios en un programa en lenguaje C.

El bloque de extensión de la aplicación permite almacenar datos que son entendidos solo por el Software de aplicación que lee el archivo GIF. Estos datos podrán ser información adicional usada para auxiliar el despliegue de los datos o coordinar la forma en que los datos de imagen son desplegados con otros archivos GIF.

Con solo unas pocas restricciones, algún número de bloques de extensión de control pueden aparecer casi en cualquier parte en una cadena de datos GIF siguiendo a la paleta de colores global. Todos los bloques de extensión inician con un valor de introducción de extensión 21h, que identifica

el bloque de datos como un bloque de extensión. Este valor es seguido por una etiqueta de bloque, que identifica el tipo de información de la extensión contenida en el bloque. Los valores de identificación de las etiquetas de bloque están en el rango de 00h a FFh. Los bloques de extensión de plano de texto, aplicación y comentarios pueden también contener uno o más sub-bloques de datos.

Las características añadidas al "89a" son opcionales y no son requeridas para aparecer en una cadena de datos GIF. Otra diferencia entre "87a" y "89a" es que al menos uno de los campos descriptor de imagen local y descriptor de pantalla lógico, que están reservados en "87a", es usado en "89a". En realidad, algunos archivos GIF que están escritos bajo la versión "89a", pero no usan las características de éste, podrían usar la versión "87a".

Bloque de Extensión de Control Gráfico

La información encontrada en este bloque es utilizada para modificar los datos en el bloque de interpretación del gráfico que le sigue inmediatamente. Un bloque de control gráfico puede modificar tanto datos bitmap como texto sencillo. Debe también aparecer en la cadena GIF antes de los datos a modificar y solo un bloque de control gráfico debe aparecer por bloque de interpretación del gráfico.

Este bloque tiene una longitud de 8 bytes con la siguiente estructura.

Campo 1. Introductor (1 byte). Identifica el inicio del bloque de extensión, debe ser 21h.

Campo 2. Etiqueta (1 byte). Etiqueta que identifica al bloque de extensión, en éste caso deber ser F9h.

Campo 3. Tamaño del bloque (1 byte). Longitud de los campos restantes, debe ser 04h. Este valor es el número de bytes de los campos paquete (1), tiempo de espera (2) e índice de color (1).

Campo 4. Paquete (1 byte). Método de colocación de gráficos a usar. Contiene los siguientes subcampos:

Campo 4.1 Bit 0. Bandera de color transparente. Si tiene un valor de '1', el campo 6, índice de color, contiene un índice de color de transparencia. En caso contrario, si tienen un valor '0', no existe tal índice de color de transparencia.

Campo 4.2 Bit 1. Bandera de entrada del usuario. Tiene un valor de '1' si una entrada de usuario (presionar una tecla, un click del mouse, o algo parecido) es esperado antes de poder continuar con la siguiente secuencia gráfica, de otro modo, el bit es '0'.

Campo 4.3 Bits 2-4. Método de colocación. Su valor indica como será dispuesto el gráfico una vez que haya sido desplegado. Los valores para este campo son:

00h	Método de disposición no especificado.
01h	No disponer del gráfico.
02h	Sobreescribir el gráfico con color de fondo negro
04h	Sobrescribir el gráfico con un gráfico previo

Campo 4.4 Bits 5-7. Reservados. no se utilizan en ésta versión y están puestos en '0'.

Campo 5. Tiempo de espera (1 byte). Centenas de segundos a esperar. Contiene un valor igual al número de centenas de un segundo que deben transcurrir antes que la presentación de gráficos continúe. si este campo tiene un valor de '0', entonces no hay tiempo de espera utilizado. Si tanto el campo tiempo de espera como el de entrada del usuario tienen un valor diferente de '0', los gráficos continúan si ha expirado el tiempo de espera como si ha sido recibida la entrada del usuario.

Campo 6. Índice de color (1 byte). Índice de color transparente. Contiene un valor solo si el subcampo 4.1, bandera de color transparente es '1'.

Campo 7. Terminador (1 byte).Terminador del bloque, debe ser '0'. Este valor de '0' marca el fin del bloque de extensión de control gráfico.

Bloque de Extensión de Plano de Texto

Un archivo GIF87a solo puede contener datos bitmap en la interpretación del gráfico. GIF89a añade la habilidad de almacenar información textual que puede ser enviada como una imagen gráfica.

Algún número de bloques de extensión de plano de texto puede aparecer en un archivo GIF. Para desplegar datos de texto sencillo, una rejilla es la que contiene los datos. El alto, ancho y posición de ésta rejilla en la pantalla de despliegue son especificados. El tamaño de cada celda en la rejilla también son descritos, y es desplegado un caracter por celda. El color de fondo y de primer plano del texto son tomados de la paleta de colores global y están descritos en el bloque de extensión de plano de texto. El dato de plano de texto real es una simple cadena de caracteres ASCII.

Este bloque tiene una longitud de 17 bytes, el byte 16 tiene una longitud variable, poseen la siguiente estructura.

Campo 1. Introductor (1 byte). Identifica el inicio del bloque de extensión, debe ser 21h.

Campo 2. Etiqueta (1 byte). Etiqueta que identifica al bloque de extensión, en éste caso deber ser 01h.

Campo 3. Tamaño del bloque (1 byte). Longitud de los campos 4 a 11, debe ser 0Ch. Este valor es el número de bytes contenidos en los campos siguientes a tamaño de bloque.

Campo 4. Rejilla texto izquierda (2 bytes). Posición en pixeles de X de la rejilla.

Campo 5. Rejilla texto superior (2 bytes). Posición en pixeles de Y de la rejilla.

Los campos 4 y 5 contienen las coordenadas X y Y (es decir, la posición) de la rejilla de texto con respecto a la esquina superior izquierda de la pantalla de despliegue (coordenada 0,0).

Campo 6. Rejilla texto ancho (2 bytes). Ancho en pixeles de la rejilla.

Campo 7. Rejilla texto largo (2 bytes). Largo en pixeles de la rejilla.

Los campos 6 y 7 contienen el tamaño en pixeles de la rejilla de texto.

Campo 8. Ancho celda (1 byte). Ancho en pixeles de la celda en la rejilla.

Campo 9. Largo celda (1 byte). Largo en pixeles de la celda en la rejilla.

Los campos 8 y 9 contienen el tamaño en pixeles de cada celda de caracter en la rejilla de texto.

Campo 10. Índice de color primer plano (1 byte). Índice de color de primer plano. Contiene de la paleta de colores global para recuperar el color del texto.

Campo 11. Índice de color fondo (1 byte). Índice de color de fondo. Contiene un índice de la paleta de colores global para ser usado como el color de fondo del texto.

Campo 12. Datos del plano de texto (1 byte). Cadena de datos del plano de texto. Contiene la información real del texto que va a ser enviada como un gráfico. El campo contienen uno o más sub-bloques de datos. Cada sub-bloque inicia con un byte que indica el número de bytes de datos que le siguen. De 1 a 255 bytes de datos le pueden seguir a éste byte.

Campo 13. Terminador (1 byte).Terminador del bloque, debe ser '0'. Este valor de '0' marca el fin del bloque de extensión de plano de texto.

Bloque de Extensión de la Aplicación

Este bloque contienen la información especifica de la aplicación de una forma similar a las etiquetas utilizada en los archivos de formato TIFF y TGA. La información no encontrada normalmente en un archivo GIF puede ser almacenada en el bloque de extensión de la aplicación que entiende como interpretar los datos.

Los datos de la extensión de la aplicación son legibles solo a la aplicación. Todos los datos almacenados en esta extensión están diseñados para ser procesados por el software de aplicación que esta leyendo y procesando la cadena de datos GIF. Para almacenar datos legibles se usa el bloque extensión de comentarios.

Algunos de los datos que puede almacenar este bloque son instrucciones de cambio de modos de video, aplicaciones especiales de procesamiento para desplegar datos de imagen, almacenar paletas de colores adicionales, información empleada para el control de plataformas de computadoras, información de como manipular archivos, como accesar a dispositivos periféricos tales como módem e impresoras y como enviar señales audibles a la bocina.

Este bloque tiene una longitud de 16 bytes, el byte 15 tiene una longitud variable, poseen la siguiente estructura.

Campo 1. Introductor (1 byte). Identifica el inicio del bloque de extensión, debe ser 21h.

Campo 2. Etiqueta (1 byte). Etiqueta que identifica al bloque de extensión, en éste caso deber ser FFh.

Campo 3. Tamaño del bloque (1 byte). Longitud de los campos 4 y 5, debe ser 0Bh. Este valor es el número de bytes contenidos en los campos identificador y código auténtico.

Campo 4. Identificador (8 bytes). Identificación de la aplicación. Puede contener hasta 8 caracteres ASCII de 7 bits. Estos caracteres son usados para identificar la aplicación por la que fue escrito el bloque de extensión de la aplicación. Si este valor de identificador es reconocido, la porción restante del bloque es leída y procesada. De lo contrario dicha porción es leída y descartada.

Campo 5. Código auténtico (3 bytes). Código de autenticidad de la aplicación. Contiene un valor que es utilizado como identificador único de un software de aplicación quién creo la extensión de la aplicación. Este campo puede contener un número de serie, número de versión, o un código ASCII o binario único para identificar el software o plataforma de

computadora. Este campo puede ser empleado para permitir solo copias especificas o revisiones de un software en particular para accesar a los datos en cierto bloque de extensión de la aplicación.

Campo 6. Datos de la aplicación (1 byte). Cadena de datos de la aplicación. Contiene la información que es utilizada por el software. Este campo esta estructurado es un serie de sub-bloques idénticos a los datos encontrados en un bloque de plano de texto.

Campo 7. Terminador (1 byte).Terminador del bloque, debe ser '0'. Este valor de '0' marca el fin del bloque de extensión de la aplicación.

Para entender como un lector GIF podría interpretar la información de un bloque de extensión de la aplicación, consideremos el siguiente ejemplo:

El identificador contienen los siguientes caracteres "CHKDATE". Este identificador es reconocido por la aplicación que esta leyendo el archivo GIF. Código auténtico contienen el valor "UNX", que es una indicación que solo las versiones de este software corriendo bajo el S. O. UNIX podrán usar los datos de este bloque. Todas las versiones del programa que no corren bajo UNIX deberán ignorar este bloque.

Bloque de Extensión de Comentarios

Este bloque es utilizado para insertar una cadena de texto legible en un archivo GIF o cadena de datos. Cada comentario puede contener hasta 255 caracteres ASCII de 7 bits, incluyendo todos los códigos de control ASCII. Este bloque puede aparecer en cualquier lugar después de la paleta de colores global. Se sugiere, sin embargo, que los comentarios deben aparecer antes o después de los datos de imagen.

Todos los datos almacenados en este bloque están diseñados para ser leídos solo por el usuario humano examinando el archivo. Todos los datos de los comentarios deben ser ignorados por la aplicación que esta leyendo el archivo. Para almacenar datos legibles a la computadora utilícese el bloque comentarios de la aplicación.

Los comentarios generalmente son usados para identificar la fuente de la imagen GIF, su autor, Software de creación, hora y fecha de la creación, la notificación de los derechos reservados, entre otros. Algunos programas de despliegue de imágenes que adecuan la versión "89a" también tienen la capacidad de desplegar los comentarios almacenados en un archivo GIF.

El bloque de extensión de comentarios debe permanecer independiente de todos los demás datos del archivo. No son modificados por la información de algún otro bloque, y los comentario no deben contener datos que intenten ser leídos e interpretados como instrucciones por el software de aplicación.

Este bloque tiene una longitud variable de entre 5 a 259 bytes con la siguiente estructura.

Campo 1. Introductor (1 byte). Identifica el inicio del bloque de extensión, debe ser 21h.

Campo 2. Etiqueta (1 byte). Etiqueta que identifica al bloque de extensión, en éste caso deber ser FEh.

Campo 3. Datos de los comentarios (1 byte). Cadena de datos de los comentarios. Contiene uno o más sub-bloques de cadenas de datos ASCII. Las cadenas de caracteres almacenados en uno de estos sub-bloques no requieren un terminador nulo.

Campo 4. Terminador (1 byte).Terminador del bloque, debe ser '0'. Este valor de '0' marca el fin del bloque de extensión de la aplicación. el valor del terminador puede ser usado como un terminador nulo si "tamaño + 1" bytes de datos de comentario es leído del bloque.

III.5.3 Tipos de imagen

El formato GIF utiliza una paleta de colores para enviar gráficos basados en rastreo de color.

Rastreo se refiere a datos gráficos representados por valores de color en puntos, que tomados juntos describen el despliegue en un dispositivo de salida.

III.5.4 Compresión

El algoritmo del LZW está basado en una tabla de traslación o tabla de cadenas, que mapea cadenas de caracteres de entrada en código. Esta tabla de cadenas no necesita estar fija para la descompresión. El truco es hacer la descompresión construyendo automáticamente la misma tabla como es construida cuando se comprimen los datos. Se utiliza el siguiente algoritmo:

! LZW NO esta libre !

Si está creando o modificando Software que utiliza el algoritmo LZW, sea cuidadoso bajo ciertas circunstancias, deberá pagar los derechos de licencia por el uso de LZW.

La Corporación UNISYS posee la patente para el LZW codec (encoding/decoding algorithm) y requiere que un derecho de licencia sea pagado para cada programa que utilice el algoritmo LZW.

```
Escribir cabecera y  paleta de colores
Inicializa tabla de cadenas
   Escribe código (código de limpieza)
   Hacer Omega = cadena vacía
   Hacer cuenta = 0
   Repetir
            Hacer k = Obtén siguiente dato
            Si (Omega + k) está en la tabla de cadenas
                     Hacer Omega = Omega + K
            Si no
                     Escribe código (Omega) (se arma un código de cadena de salida)
                     Incrementa cuenta
                     Añade (Omega +K) a la tabla de cadenas
                     Hacer Omega = k
            Fin del si
            Si cuenta = 255 (Puede ser cualquier valor de entre 1-255)
                     Escribir 255
                     Hacer cuenta = 0
            Fin del si
   Hasta leer todos los datos de la imagen
Escribe código (Omega)
Escribe código (fin de información)
```

El esquema es simple aunque es un reto implementarlo eficientemente. Posee 2 códigos clave, los cuales son: código de limpieza y código de fin de información.

Los datos que toman las cadenas de LZW son bytes sin comprimir de la imagen. Debido a la adaptabilidad de LZW, no se paga un radio de castigo significante de compresión por combinar varios pixeles en un byte antes de comprimir.

Inicializa tabla de cadenas, inicializa la tabla de cadenas para contener todas las cadenas posibles de cadenas de datos sencillos. Hay 256 de ellos, numerados de 0 a 255.

Escribe código, escribe el valor de **omega** como un código en el código de cadenas, que es la cadena de salida. El primer código escrito es el **código de limpieza**, que está definido como el código #256.

Omega es una **cadena prefijo**. El símbolo más (+) indica la concatenación de cadenas.

Obtén siguiente dato, recupera el siguiente valor de dato de la imagen.

Añade entrada a tabla, añade una entrada a la **tabla de cadenas**. **Inicializa tabla de cadenas** tiene ya puestas 256 entradas en la tabla, que corresponden a la paleta de colores. Cada entrada consiste de una cadena de caracter sencillo y esta asociada con un valor de código, que es, en nuestra aplicación, idéntico al caracter en sí mismo. Es decir, la entrada 0 en la tabla consiste de la cadena '0', con su correspondiente valor de '0', la primera entrada consiste de la cadena '1', con su correspondiente valor de código '1',..., y las entrada 255 en la tabla consiste de la cadena '255', con su correspondiente valor de código '255'. El código 256 es reservado para el **código de** limpieza y el código 257 para un **código de fin de información**. Así, tenemos que la primera entrada de caracteres múltiples añadida a la tabla estará en la posición de código #258.

El código fin de información será escrito al final de la codificación.

Supongamos que tenemos los siguientes datos de entrada:

$Pixel_0$	$Pixel_1$	$Pixel_2$	$Pixel_3$	$Pixel_4$	$Pixel_5$	$Pixel_6$	$Pixel_7$	$Pixel_8$	$Pixel_9$
7	7	7	8	8	7	7	6	6	5

Línea de pixeles sin codificar

Por la complejidad del método se ha optado por realizar la codificación mediante una tabla, en lo personal creemos que será más comprensible de esta forma en ésta ocasión, de no ser así, en el capitulo V se analiza con más detalle.

Ya escrita la cabecera de datos y paleta de colores, se realiza la inicialización de tabla de cadenas, procedemos ha hacer nuestra tabla con las variables que necesitamos, el subíndice representa el número de paso en un ciclo especifico:

No. ciclo	Entradas Tabla de cadenas	k	Omega	Omega + K	¿ está ?	Código de cadena (salida)	Cuenta
			$Null_2$			256_1	0_3
1		7_1	7_4	$Null+7=7_2$	$Sí_3$		
2	$258 = 7+7_6$	7_1	7_7	$7+7_2$	No_3	7_4	1_5
3		7_1	$7+7_4$	$7+7_2$	$Sí_3$		
4	$259 = 7+7+8_6$	8_1	8_7	$(7+7)+8_2$	No_3	258_4	2_5
5	$260 = 8+8_6$	8_1	8_7	$8+8_2$	No_3	8_4	3_5
6	$261 = 8+7_6$	7_1	7_7	$8+7_2$	No_3	8_4	4_5
7		7_1	$7+7_7$	$7+7_2$	$Sí_3$		
8	$262 = 7+7+6_6$	6_1	6_7	$(7+7)+6_2$	No_3	258_4	5_5
9	$263 = 6+6_6$	6_1	6_7	$6+6_2$	No_3	6_4	6_5
10	$263 = 6+5_6$	5_1	6_7	$6+5_2$	No_3	6_4	6_5

Los cadena de salida es la siguiente: 256 7 258 8 8 258 6 6... Como se puede apreciar cada dato necesita estar escrito utilizando 9 bits. Cuando estamos iniciando, se utilizan códigos de 9 bits, ya que las nuevas entradas a la tabla de cadenas son mayores a 255 pero menores que 512, para sacar los códigos de 8 bits se realiza una concatenación de éstos datos y se toman de 8 en 8 datos, para sacar 9 bytes, como se muestra a continuación:

Tomando los datos de 9 bits

6	6	258	8	8	258	7	256
000000110	000000110	100000010	000001000	000001000	100000010	000000111	100000000

Tomando 8 bits

3	1	160	64	128	68	8	15	0
00000011	00000001	10100000	01000000	10000000	01000100	00001000	00001111	00000000

$Pixel_0$	$Pixel_1$	$Pixel_2$	$Pixel_3$	$Pixel_4$	$Pixel_5$	$Pixel_6$	$Pixel_7$	$Pixel_8$	$Pixel_9$
7	7	7	8	8	7	7	6	6	5

Linea de pixeles sin codificar

Dato0	Dato1	Dato2	Dato3	Dato4	Dato5	Dato6	Dato7	Dato8
0	15	8	68	128	64	160	1	3	...

Linea de datos codificados

Cuando añadamos la entrada 512, ajustamos a datos de 10 bits. Se realiza la misma operación para sacar códigos de 8 bits, tomando datos de 8 en 8, para sacar 10 bytes. Igualmente ajustamos a códigos de 11 bits en 1024, códigos de 12 bits en 2048. Se limitará al manejo de código hasta 12 bits, ya que la tabla puede tener a los sumo 4096 entradas. Si se tomarán más bits la tabla tenderá a ser demasiado larga.

Tan pronto como se utiliza la entrada 4094, escribimos un **código de limpieza** de 12 bits. Si esperamos algo más largo para escribir el **código de limpieza**, el decodificador puede tratar de interpretarlo como un código de limpieza de 13 bits. En éste punto el codificador vuelve a iniciar **la tabla de cadenas** escribiendo de nuevo datos de 9 bits.

Note que cada vez que se escribe un código se añade una entrada a la tabla, **Omega** no se queda vacía. Contiene exactamente un dato. Tenga cuidado de no perderlo cuando escriba un código de limpieza al final de la tabla. Puede escribir **omega** como un código de 12 bits antes de escribir el código de limpieza, en tal caso, deberá hacerlo después de añadir la entrada 4093 a la tabla o después del código de limpieza como un código de 9 bits. La decodificación da el mismo resultado en tal caso. El código de compresión LZW está almacenado en bytes de alto a bajo orden. El código comprimido está escrito como bytes, no como palabras (16 bits).

Note que la **tabla de cadenas** de LZW es continuamente actualizada de cadenas que han sido encontradas en los datos. Por lo tanto, refleja las características de los datos, mejorando en un alto grado la adaptabilidad.

El procedimiento para la decodificación es quizás un poco más difícil conceptualmente, pero no es tan malo.

De nuevo tenemos que inicializar una tabla de cadenas. Esta tabla viene del conocimiento que tenemos acerca de cadena de datos que obtendremos eventualmente, semejante a los valores posibles de datos que puede tomar. En archivos GIF, está información está en la cabecera como el número de valores posibles de pixel. La belleza de LZW, sin embargo, es que, esto es todo lo que necesitamos saber. Construiremos el resto de la tabla de cadenas tan pronto se descomprima la cadena de códigos. La compresión está hecha de tal modo que nunca encontraremos un código en la cadena de códigos que no pueda ser traducido a una cadena.

Puesto que es necesario un análisis minucioso para poder comprender el algoritmo de descompresión, lo dejaremos pendiente por ahora, para explicarlo a fondo en el capítulo IV.

III.6 Formato TIFF

Nombre	TIFF
Conocido como	Tag Image File Format.
Tipo	Bitmap
Colores	1 a 24 Bits.
Compresión	RLE, Sin comprimir, LZW, CCITT grupo 3 y Grupo 4, JPEG.
Tamaño Máximo de Imagen	$2^{32}-1$
Imágenes Múltiples por Archivo	Sí.
Orden de Byte	En discusión
Creador	Aldus, Corp.
Plataforma	MS-DOS, Macintosh, UNIX y otros.
Aplicaciones que soporta	La mayoría de dibujo y algunos programas de publicaciones.

III.6.1 Antecedentes

La especificación TIFF fue originalmente publicada en 1986 por Aldus Corporation como un método estándar de almacenamiento de imágenes en blanco y negro creadas por scanners y aplicaciones de publicaciones. La primera publicación de TIFF, fue la tercera revisión principal del formato TIFF y, sin embargo, no fue asignada a un número de versión específica, se le puede referir como TIFF revisión 3.0.

TIFF revisión 4.0 publicada en Abril de 1987 ofrecía soporte para imágenes a color RGB sin comprimir y fue rápidamente seguida por el TIFF revisión 5.0 en Agosto de 1988, la cual en su primera revisión añadía la capacidad para almacenar imágenes con paleta de color y soporte para el algoritmo de compresión LZW. TIFF v6.0 fue publicada en Junio de 1992 con soporte para imágenes de color CMYK y YCbCr así como el método de compresión JPEG.

Actualmente TIFF es un archivo de formato que se utiliza en la mayoría de los programas y Software de dibujo y publicaciones, es un formato nativo del GUI de Microsoft Windows. La extensión de TIFF permite el almacenamiento de imágenes múltiples bitmap de cualquier tamaño de pixel, lo que lo hace ideal para la mayoría de los requerimientos de almacenamiento.

TIFF fue creando una reputación de poder y flexibilidad, pero es considerado complicado y misterioso. En su diseño, TIFF contempla ser extensible además de mejorar algunas características

que un programador necesita en un formato de archivo. Como TIFF es demasiado extensible y tiene algunas capacidades extras sobre los demás formatos de archivos de imagen, este formato es probablemente el formato más confuso de entender y usar.

Los archivos TIFF no son muy portables entre aplicaciones, considerando que TIFF es ampliamente usado como un formato de intercambio de imágenes. Si se bajan imágenes de un BBSs es posible que un programa de dibujo o procesador de palabras pueda desplegarlos pero no todas la imágenes que se hayan bajado, ya que puede ocurrir algunos errores tales como: "Unknown Tag Type" (Tipo de Etiqueta Desconocida) o "Unsupported Compression Type" (Tipo de Compresión no Soportado) o si crea un archivo TIFF en una aplicación y otra aplicación de la misma computadora no la puede leer o desplegar, esto hace que el formato sea considerado como malo, pero no es malo, sino que la aplicación no supo leer o escribir apropiadamente el formato.

Una fuente de problemas en el TIFF es la incapacidad para leer datos indiferentes de clasificación de bytes, little-endian (como el Intel iAPX86) y el big-endian (como el Motorola MC68000A). Si la lectura es en big-endian y usamos el little-endian da por resultado basura. Otra fuente de problemas es que algunos lectores de imágenes no utilizan el algoritmo de la codificación de los datos de la imagen. La mayoría de lectores apoyan a los archivos sin comprimir y el RLE pero no leen la compresión CCITT T.4 y T.6. e incluso también los archivos que utilizan colores pero no utilizan la decodificación LZW. Algunos otros problemas consisten en que el lector de imágenes no interpreta correctamente los datos que la cabecera contiene. El programa que acompaña este trabajo esta adecuado solo para leer los archivos que utilizan RLE y sin comprimir.

III.6.2 Estructura

La figura III.13 muestra un diagrama a bloques de la estructura de un formato de archivo TIFF.

Figura III. 13 Estructura del archivo TIFF.

El archivo TIFF esta estructurado básicamente en tres secciones: La cabecera del archivo, el directorio de la imagen, y los datos de la imagen. Un archivo TIFF que contiene imágenes múltiples, tiene por cada imagen un directorio de imagen y sus respectivos datos de imagen.

TIFF tiene la reputación de ser un formato complicado, en parte por la situación de que cada directorio de imagen y sus datos no están precisamente juntos o fijos en algún lugar del archivo como lo maneja GIF. De hecho, sólo hay una parte fija dentro del archivo TIFF, esta es la cabecera del archivo que está siempre en los primeros ocho bytes del archivo TIFF. Todos los datos de cada imagen se encuentran en cada directorio de imagen e incluso si hay más imágenes, el mismo directorio de imagen nos dice donde esta el siguiente directorio de imagen, además de que cada directorio de imagen puede utilizar un método de compresión diferente al de los demás. No hay ningún límite para el número de imágenes TIFF incluidos en un archivo.

Cada directorio de la imagen contiene una o más estructuras de datos que se conocen como etiquetas o índices (TAG). Cada etiqueta es una estructura de 12 bytes y contiene una parte de toda la información necesaria para desplegar una imagen. Una etiqueta contiene cualquier tipo de datos, hay más de 70 etiquetas que sirven para representar en conjunto una imagen, casi siempre se encuentran en grupos inmediatos dentro de cada directorio de la imagen. Las etiquetas que son definidas por TIFF se llaman etiquetas públicas y no se modifican. Las etiquetas que son definidas por los usuarios, se llaman privadas, algunas de estas son definidas por el propietario de TIFF, Aldus y, solo son utilizadas por la revisión 6.0 del TIFF.

La figura III.14 muestra las posibles formas de encontrar un archivo TIFF con sus respectivos directorios de imagen.

En cada ejemplo la cabecera aparece primero. En el primer ejemplo cada uno de los directorios de imagen se escriben primero y después cada uno de los datos de las imágenes, este arreglo es el mas eficaz para leer un TIFF. En el segundo ejemplo, se escribe cada directorio de imagen, seguido por su propio conjunto de datos de imagen, es muy común encontrar este tipo cuando hay imágenes múltiples; en el último ejemplo vemos que los datos de la imagen se han escrito en los primeros bytes del archivo y son seguidos por los directorios de imagen, este, aparentemente es el más raro de los tres. Sin embargo, por experiencia propia es muy común cuando solo hay una imagen en el TIFF.

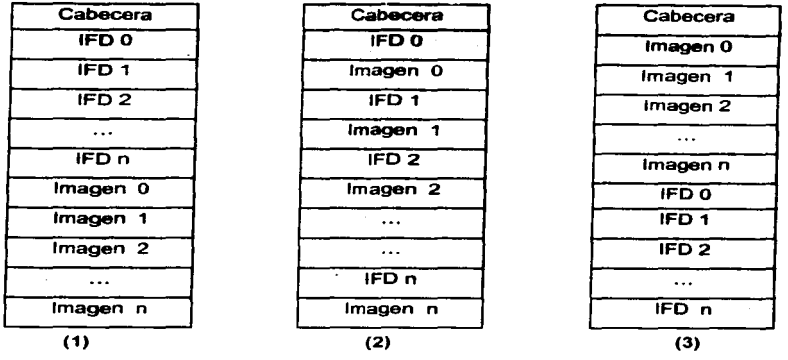


Figura III.14. Posibles arreglos de una estructura TIFF.

Aunque mostramos tres posibles formas de encontrar un TIFF, no son las únicas, aunque parezca raro, estos datos pueden grabarse de cualquier forma, esto depende del que escribe el TIFF, y de su conocimiento sobre lo que ofrece el formato, por lo tanto, debemos estar preparados y dominar en un porçentaje elevado el formato TIFF.

La figura III.15 muestra la forma lógica de un archivo con formato TIFF.

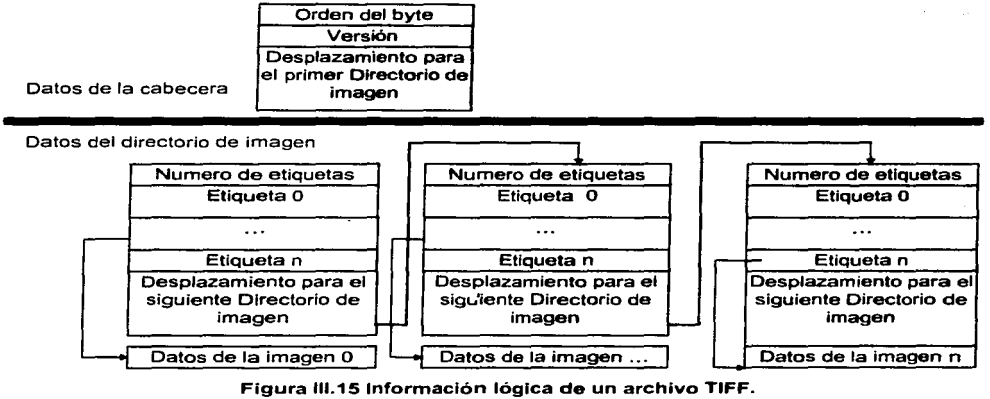


Figura III.15 Información lógica de un archivo TIFF.

TIFF a pesar de su complejidad, tiene la cabecera más simple de todos los formatos. La cabecera del TIFF se compone de los siguientes campos de solo ocho bytes de longitud en total.

Cabecera

Campo 1. Identificador. (2 bytes,bytes 0-1). Especifica el orden de byte utilizado. Contiene el valor de 4949h (II) o 4D4Dh (MM). Estos valores indican si los datos en el TIFF se escriben en orden little-endian o big-endian, respectivamente. Todo los datos que se encuentren después de estos datos se encontraran según la clasificación del orden del byte que especifica este campo.

Campo 2. Versión (2 bytes, bytes 2-4). Indica la versión del TIFF, debe ser 42 (2Ah) sin importar la revisión.

Campo 3. Desplazamiento directorio de imagen (4 bytes,bytes 5-7). Especifica el desplazamiento del primer directorio de imagen. Es un valor de 32-bits. Si los datos del directorio de imagen le siguen a la cabecera, entonces este campo tendrá el valor 00 00 00 08h indicando que la posición del primer directorio de imagen esta en la posición 8 del archivo.

Una manera rápida de verificar si un archivo es TIFF, es leer los primeros 4 bytes y ver si contienen los datos: 49h 49h 2Ah 00h ó 4Dh 4Dh 00h 2Ah, entonces es un archivo TIFF correcto.

Directorio de imagen

Un directorio de imagen es una colección de información similar a la de la cabecera y es utilizada para describir los datos de imagen asociados al directorio de imagen. Cada directorio de imagen contiene información del alto, ancho, profundidad de la imagen, número de planos de color y tipo de compresión. A diferencia de una cabecera clásica que esta fija, el directorio de imagen es dinámico y puede variar su tamaño y se puede encontrar en cualquier parte dentro del TIFF. Puede haber más de un directorio de imagen dentro de un archivo TIFF. La figura III.16 muestra la estructura del directorio de imagen.

Un concepto erróneo sobre TIFF es cuando se piensa que un directorio de imagen es parte de la cabecera del TIFF, la cabecera del TIFF no contiene la información de un directorio de la imagen. Es posible pensar que los directorios de imagen de un TIFF son extensiones de la cabecera del TIFF.

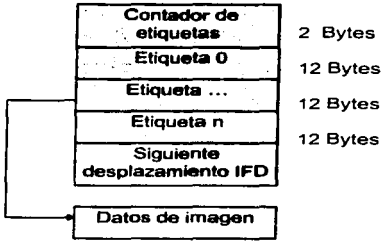


Figura III.16 Estructura del directorio de imagen.

Un directorio de imagen variara en tamaño, porque contendrá un número inconstante de datos o etiquetas. Cada etiqueta contiene un pedazo único de información de la imagen, lo mismo que hacen los campos dentro de una cabecera. Sin embargo, hay una diferencia, se pueden agregan etiquetas y se pueden modificar el numero de ellas por cada directorio de imagen; en cambio, en una cabecera el campo se debe considerar inmovible y de tamaño fijo.

El directorio de imagen muestra en su estructura 3 campos, la posición de estos campos varia de acuerdo al desplazamiento en donde se encuentre el directorio de la imagen, estos campos son :

Campo 1. Número de Etiquetas (2 bytes,variable). Especifica el número de etiquetas que contiene.

Campo 2. Arreglo de etiquetas (# de bytes = numero de etiquetas*12). Contiene un arreglo bidimensional en donde el tamaño depende del numero de etiquetas que se encontraron por 4.

Campo 3. Desplazamiento del directorio de imagen (4 bytes,variable). Contiene el desplazamiento del siguiente directorio de imagen, si no hay más directorio de imagen, este campo tiene un valor de 0.

Etiquetas

Como se mencionó en la sección anterior, se puede pensar en una etiqueta como los datos de una cabecera de archivo. Sin embargo, considerando que un campo de la cabecera puede contener datos de un tamaño fijo y normalmente se localiza en una posición fija dentro del archivo, una etiqueta contendrá, datos de un tamaño variable y además se puede encontrar en cualquier parte del archivo.

Cada etiqueta tiene un tamaño de 12 bytes (ver figura III.17), y se encuentran en el orden que especifica TIFF. La Tabla III.9 lista las etiquetas públicas y privadas incluidas en las revisiones 4.0, 5.0 y 6.0 de TIFF, por orden de aparición alfabético (en inglés), sin embargo, el orden al que se hace referencia lo especifica el # de etiqueta, en orden ascendente. Algunas etiquetas se han vuelto obsoletas y no se encuentran en la revisión actual de TIFF; sin embargo, proporcionamos las etiquetas de las versiones 4,0 y 5,0 del TIFF ya que aun se encuentran en uso. También, note que varias etiquetas apoyarían a más de una revisión.

Identificador
Tipo de dato
Contador
Desplazamiento

Figura III.17 Estructura de una etiqueta TIFF.

Tabla III.9 Etiquetas públicas y privadas.

Nombre (Original)	Nombre (Usado en el presente trabajo)	# de Etiqueta	Tipo	Revisión soportada
Artist	Artista	315	02	--- R5 R6
BadFaxLines[1]	Linea fax mala	326	04	--- --- ---
BitsPerSample	Bits_muestra	258	03	R4 R5 R6
CellLength	Largo de Celda	265	03	R4 R5 R6
CellWidth	Ancho de Celda	264	03	R4 R5 R6
CleanFaxData[1]	Dato_fax_Nvo	327	03	--- --- ---
ColorMap	Paleta	320	03	--- R5 R6
ColorResponseCurve	Curv_resp_col	301	03	R4 R5 ---
Nombre (Original)	Nombre (Usado en el presente trabajo)	# de Etiqueta	Tipo	Revisión soportada
ColorResponseUnit	Uni_resp_col	300	03	R4 --- ---
Compression	Compresión	259	03	R4 R5 R6
Uncompressed	Sin compresión	1		R4 R5 R6
CCITT 1D	CCITT 1D	2		R4 R5 R6
CCITT Group 3	CCITT Grupo 3	3		R4 R5 R6
CCITT Group 4	CCITT Grupo 4	4		R4 R5 R6
LZW	LZW	5		--- R5 R6
JPEG	JPEG	6		--- --- R6
Uncompressed	Sin compresión	32771		R4 O5 O6
Packbits	Bits Empaquetados RLE	32773		R4 R5 R6
ConsecutiveBadFaxLines	Lin_fax_mal_c	328	04	--- --- ---
Copyright	Derechos	33432	02	--- --- R6
DateTime	Fecha y hora	306	02	--- R5 R6
DocumentName	Nombre_doc	269	02	R4 R5 R6
DotRange	Rango_punto	336	03	--- --- R6

Tabla III.9 Etiquetas públicas y privadas. (continuación)				
Nombre (Original)	**Nombre (Usado en el presente trabajo)**	**# de Etiqueta**	**Tipo**	**Revisión soportada**
ExtraSamples	Muestras_extra	338	01	--- --- R6
FillOrder	Orden_relleno	266	03	R4 R5 R6
FreeByteCounts	Cta_byte_libre	289	04	R4 R5 R6
FreeOffsets	Desplaza_libre	288	04	R4 R5 R6
GrayResponseCurve	Resp_curv_gris	291	03	R4 R5 R6
GrayResponseUnit	Uni_resp_gris	290	03	R4 R5 R6
HalftoneHints	Hints_med_tono	321	03	--- --- R6
HostComputer	Host_PC	316	02	--- R5 R6
ImageDescription	Descrip_imagen	270	02	R4 R5 R6
ImageHeight	Largo_imagen	257	03	R4 R5 R6
ImageWidth	Ancho_imagen	256	03	R4 R5 R6
InkNames	Nombres_ink	333	02	--- --- R6
InkSet	Puesta_ink	332	03	--- --- R6
JPEGACTTables	Tabla_JPEGACT	521	04	--- --- R6
JPEGDCTTables	Tabla_JPEGDCT	520	04	--- --- R6
JPEGInterchangeForm	Form_Int_JPEG	513	04	--- --- R6
JPEGInterchangeFormatLength	Largo_FIJPEG	514	04	--- --- R6
JPEGLosslessPredictors	Pred_Com_JPEG	517	03	--- --- R6
JPEGPointTransforms	Trans_pun_JPEG	518	03	--- --- R6
JPEGProc	Proceso_JPEG	512	03	--- --- R6
JPEGRestartInterval	Int_reini_JPEG	515	03	--- --- R6
JPEGQTables	Tabla_JPEGQ	519	04	--- --- R6
Make	Realizar	271	02	R4 R5 R6
MaxSampleValue	Valmax_muestra	281	03	R4 R5 R6
MinSampleValue	Valmin_muestra	280	03	R4 R5 R6
Model	Modelo	272	02	R4 R5 R6
NewSubFileType	Nvoti_Subarchi	254	04	--- R5 R6
NumberOfInks	Numero_ink	334	03	--- --- R6
Orientation	Orientacion	274	03	R4 R5 R6
PageName	Nombre_pagina	285	02	R4 R5 R6
PageNumber	Numero_pagina	297	03	R4 R5 R6
PhotometricInterpretation	Inter_fotome	262	03	R4 R5 R6
WhiteIsZero	Blanco si es Cero	0		R4 R5 R6
BlackIsZero	Negro si es cero	1		R4 R5 R6
RGB	RGB	2		R4 R5 R6
RGB Palette	Paleta RGB	3		R4 R5 R6
Tranparency Mask	Máscara de	4		R4 R5 R6
CMYK	transparencia	5		R4 R5 R6
YCbCr	Color de tipo CMYK	6		R4 R5 R6
CIELab	Color de tipo YCbCr	8		R4 R5 R6
	Color de tipo CIELab			
PlanarConfiguration	Config_plano	284	03	R4 R5 R6
Predictor	Predictor	317	03	--- R5 R6
PrimaryChromaticities	Croma_prim	319	05	--- R5 R6
ReferenceBlackWhite	Referencia_B/N	532	04	--- --- R6
ResolutionUnit	Uni_resolucion	296	03	R4 R5 R6
RowsPerStrip	Lineas_x_Tiras	278	04	R4 R5 R6
SampleFormat	Mues_formato	339	03	--- --- R6
SamplesPerPixel	Mues_x_pixel	277	03	R4 R5 R6

Tabla III.9 Etiquetas públicas y privadas. (continuación)				
Nombre (Original)	**Nombre (Usado en el presente trabajo)**	**# de Etiqueta**	**Tipo**	**Revisión soportada**
SMaxSampleValue	Val_smax_mues	341	04	--- --- R6
SMinSampleValue	Val_smin_mues	340	04	--- --- R6
Software 305 ASCII	Software	305	02	--- R5 R6
StripByteCounts	cta_byte_Tira	279	04	R4 R5 R6
StripOffsets	Desplaza_Tira	273	04	R4 R5 R6
SubFileType	Tipo_subarchi	255	03	R4 R5 R6
T4Options[2]	Opcion_t40	292	04	R4 R5 R6
T6Options[3]	Opcion_t60	293	04	R4 R5 R6
TargetPrinter	Tipo_impresora	337	02	--- --- R6
Thresholding	Thresholding	263	03	R4 R5 R6
TileByteCounts	Cta_byte_Azulejo	325	04	--- --- R6
TileLength	Largo_Azulejo	323	04	--- --- R6
TileOffsets	Desplaza_Azulejo	324	04	--- --- R6
TileWidth	Ancho_Azulejo	322	04	--- --- R6
TransferFunction[4]	Funcion_trans	301	03	--- --- R6
TransferRange	Rango_trans	342	03	--- --- R6
XPosition	Posicion_x	286	05	R4 R5 R6
Xresolution	Resolucion_x	282	05	R4 R5 R6
YCbCrCoefficients	Coefi_YCbCr	529	05	--- --- R6
YCbCrPositioning	Posicion_YCbCr	531	03	--- --- R6
YCbCrSubSampling	Submues_YCbCr	530	03	--- --- R6
YPosition	Posicion_y	287	05	R4 R5 R6
YResolution	Resolucion_y	283	05	R4 R5 R6
WhitePoint	Punto_blanco	318	05	-- R5 R6

En la columna revisión soportada el prefijo R indica la revisión soportada, --- indica que no esta disponible y el prefijo O indica que esta obsoleta en esa revisión.

Notas:

[1] Etiquetas BadFaxLines, CleanFaxData, y ConsecutiveBadFaxLines son parte de TIFF Clasificación F que Aldus mantuvo no definiéndose realmente en el TIFF 6,0.

[2] Etiqueta No. 292 de Group3Options se cambio por el nombre T4Options en TIFF 6.0.

[3] Etiqueta No. 293 de Group3Options se cambio por el nombre T6Options en TIFF 6.0.

[4] Etiqueta 301 de ColorResponseCurve se cambio por TransferFunction en el TIFF 6.0.

La versatilidad de las etiquetas usadas por TIFF aumenta el tamaño de datos para cada imagen , esto se puede considerar como una desventaja en cuanto a los datos de una cabecera que son fijos. Una etiqueta puede solo representar un byte de información, sin embargo, esta etiqueta siempre debe tener un tamaño de 12 bytes. El número de etiquetas por directorio de imagen se limita a 65,535.

Una etiqueta TIFF contiene los siguientes campos:

Campo 1. Identificador (2 bytes, variable). Especifica el identificador de etiqueta (Tabla III.9).

Campo 2. Tipo de dato (2 bytes,variable). Indica el tipo de dato que usa la etiqueta (Tabla III.10).

Campo 3. Contador de datos (4 bytes, variable). Indica un número de referencia sobre el tamaño que especifica el tipo de dato de cada etiqueta. Por ejemplo, un contador de datos que tenga valor de 28h y un tipo de dato con valor 02h indica una cadena con caracteres ASCII de 40 bytes en longitud, incluso el NULL (Carácter de terminación).

Campo 4. Desplazamiento (4 bytes, variable). Especifica el desplazamiento de los datos reales dentro del TIFF según la etiqueta. Si los datos de la etiqueta son de cuatro bytes o menos en tamaño, estos datos se encuentran en este campo. Si son más mayores a cuatro bytes, entonces este campo contiene un desplazamiento a la posición de los datos. La mayoría de los datos de cada etiqueta se guardan típicamente fuera de los datos de etiqueta, ya sea antes o después de la misma (ver Figura III.14). La etiqueta graba datos en este campo cuando guarda la resolución de la imagen, el método de compresión utilizado, y algunos otros, cuando no almacena en este campo hace una referencia a los datos de imagen.

Tabla III. 10 Tipos de datos utilizados por las etiquetas.

Tipo	Nombre	Bits Utilizados
01	BYTE	8-bit entero sin signo
02	ASCII	8-bit, Cadena con Terminación NULL
03	SHORT	16-bit Entero si signo
04	LONG	32-bit Entero sin signo
05	RATIONAL	32-bit Dos Enteros sin signo
Nuevos tipos a partir de la revisión 6.0		
06	SBYTE	8-bit Entero con signo
07	UNDEFINE	8-bit byte
08	SSHORT	16-bit Entero con signo
09	SLONG	32-bit Entero con signo
10	SRATIONAL	32-bit Dos Enteros con signo
11	FLOAT	4 bytes simple precisión IEEE Valor tipo Real
12	DOUBLE	8 bytes doble precisión IEEE Valor tipo Real

III.6.3 Tipos de imagen

Aquellos que utilicen el TIFF deben escribir solo etiquetas útiles al archivo. TIFF define un concepto llamado *baseline* para sus tipos de datos (Colores) en sus imágenes. Se definen estos baseline por el tipo de color que utiliza cada imagen: 2 Niveles (Blanco y Negro), Escala de grises,

Paleta de colores (RGB) y Color Total (True Color). Cada baseline tiene un número minimo de etiquetas que requiere TIFF.

En el TIFF 5,0 estos baselines eran llamados clases de TIFF. Cada archivo TIFF consta de un baseline común (Clase X) el cual se modifico por una clase adicional que depende de los tipos de datos que requiere la imagen. Las clases adicionales fueron Clase B (2 niveles), Clase G (escala de grises), Clase P (paleta de colores), y Clase R (Color Total).

En el TIFF 6,0 se redefinen estas clases y se forman 4 clases por separado. La clase X que se combina con cada una de las cuatro clases para formar: 2 niveles, escala de grises, Paleta de colores, y el baseline de Color total. Aunque TIFF 6,0 esta muy lejos de predominar con sus clases, probablemente porque las que actualmente están dominando son de TIFF 5.0, sin embargo ,TIFF en algunos años se referirá a las clases usadas por el TIFF 6.0.

La clase F existe específicamente para el almacenamiento de imágenes de facsímil con la estructura TIFF. Esta clase de TIFF se creó por Tecnologías Cygnet, y es utilizada por productos Everex que emplea el facsímil. Aunque Tecnologías Cygnet ya no esta activo, TIFF Clase F se usa y se considera por algunos un excelente almacenamiento para datos de facsímil.

La Tabla III.11 muestra el mínimo de etiquetas que deben de aparecer en el directorio de la imagen según el baseline del TIFF 6.0.

Tabla III.11 Etiquetas mínimas requeridas.

Nombre de la clase	Tipo de etiqueta	Nombre de la etiqueta
2 niveles y escala de grises	254	Nvoti_Subarchi
	256	Ancho_imagen
	257	Largo_imagen
	258	Bits_muestra
	259	Compresión
	262	Inter_fotome
	273	Desplaza_tira
	277	Mues_x_pixel
	278	Lineas_x_tiras
	279	cta_byte_Tira
	282	Resolución_x
	283	Resolución_y
	296	Uni_resolucion
Paleta de colores	320	Paleta
RGB	284	Config_plano
YCbCr	529	Coefi_YCbCr
	530	Submues_YCbCr

Tabla III.11 Etiquetas mínimas requeridas. (continuación)		
Nombre de la clase	**Tipo de etiqueta**	**Nombre de la etiqueta**
	531	Posicion_YCbCr
	532	Referencia_B/N
Clase F	326	Linea fax mala
	327	Dato_fax_Nvo
	328	Lin_fax_mal_c

2-Niveles (anteriormente Clase B) y escala de grises (anteriormente Clase G). Deben contener las trece etiquetas que se listan. Estas etiquetas deben aparecer en todas la revisiones 5,0 y 6,0 sin importar el tipo de imagen que se almacene.

Paleta de colores (anteriormente Clase P). Agrega una etiqueta más que describe el tipo de información de la paleta que se encuentra dentro del TIFF.

RGB Color Total (anteriormente Clase R). Utiliza las 13 etiquetas de 2 niveles y agrega una 14ª . la cual describe el formato de los datos del bitmap de la imagen.

Color tipo YCbCr . Utiliza las 13 etiquetas de 2 niveles y agrega cuatro más.

La clase F del TIFF agrega tres etiquetas además de las 13.

Todas las etiquetas que se encuentren de más en el archivo, están disponibles para que el diseñador las utilice de acuerdo a sus necesidades. Mientras que un lector del TIFF debe poder apoyar la interpretación de todas las etiquetas, un escritor no debe de incluir tantas etiquetas dentro del archivo excepto las que se requieran.

Organización de datos

Como hemos visto, los datos de la imagen en un TIFF no aparecen siempre inmediatamente después de la cabecera de la imagen, como en otros formatos. Por el contrario, podrán aparecer casi en cualquier parte dentro del archivo. Para un lector o escritor de TIFF es necesario que se conozca y se entienda el concepto de tiras (*strips*).

Nota: TIFF 6,0 contendrá azulejos (*Tiles*) en lugar de tiras.

Tiras

Hay lectores o escritores que dentro de su código fuente o en alguna parte de su pantalla de visualización dicen "Este lector TIFF no apoya las imágenes a base de tiras ". Hay un gran número de lectores de archivos TIFF que dejan de leer ciertos archivos porque el autor del lector no

entendía realmente el concepto de cómo se pueden organizar los datos de la imagen dentro de un TIFF mediante tiras.

Una tira es una colección individual de una o más líneas inmediatas de datos de la imagen. Dividir los datos de la imagen en tiras (pequeños buffers) hace que se pueda acceder a ellas más fácilmente. Este concepto existe en varios formatos gráficos, teniendo como nombres bloques, bandas, pedazos cortos y largos. TIFF en tiras difiere de los otros conceptos en varias aspectos importantes debido a su estructura.

Existen 3 etiquetas que son necesarias para definir tiras en una imagen dentro de un archivo TIFF. Estas etiquetas son :

Lineas_x_Tiras (278): Indica el numero de líneas de datos de la imagen que se encuentran comprimidas, el número máximo de líneas por tira es de 2^{32}-1, que es el tamaño máximo posible de una imagen TIFF. Todas las tiras deben de tener el mismo proceso de codificación, esquema de color, tamaño de pixel, etc. Si deseamos saber cuantas tiras se encuentran en un archivo TIFF utilizando esta etiqueta el calculo seria de la siguiente manera:

Tiras_en_la_imagen=((longitud de imagen * (Renglon_por_tira-1)) / Renglo_por_tira)

Desplaza_tira (273): Esta etiqueta contiene un orden de valores de desplazamiento, uno por tira, que indica la posición del primer byte de cada tira en del archivo. Los valores de desplazamiento están en orden, así, el primer desplazamiento indica los datos de la primera tira, el segundo indica la segunda tira de datos, y así sucesivamente. Si los datos de la imagen están por planos separados (Config_plano(284) = 2), Desplaza_tira contiene un arreglo de 2D de valores. Todos los componentes del plano de color 0 se guardan en los primeros bytes, seguidos por las componentes del plano de color 1 y así sucesivamente. Las tiras del plano de color se escriben al archivo en cualquier orden pero típicamente están por el plano (RRRRGGGGBBBB) o por el componente de color (RGBRGBRGBRGB).

Desplaza_tira deja que cada tira en un TIFF tenga una colocación completamente independiente de todas las otras tiras en el mismo subarchivo[1]. Esto indica que las tiras pueden aparecer en cualquier orden dentro del archivo, si

[1] Se dice subarchivo ya que puede haber más de una imagen en el mismo archivo TIFF.

un lector de TIFF lee la imagen, sin utilizar el concepto de tiras, y esa imagen la lee de manera correcta, puede ser que las tiras de grabaron en orden en el que aparece la imagen y por consiguiente el hecho de utilizar tiras o no ya sale sobrando, incluso cuando hay un método de compresión. Esta técnica es eficaz cuando todas las tiras en el TIFF son continuas, si estas se guardan sin una continuidad y se trata de leer esta imagen sin el concepto de tiras, la imagen de desplegara probablemente en rebanadas.

La etiqueta Lineas_por_tiras y el tamaño de cada elemento en el orden de Desplaza_tira son generalmente de tipo LONG (32 bits). TIFF 5.0 tuvo la habilidad de usar en estas etiquetas el tipo SHORT (16 bits). Pero algunos lectores antiguos tendrían problemas al leer datos de tipo SHORT siendo estos de tipo LONG, por consiguiente, el desplazamiento puede ser impropio. Esto se arreglo cuando dichos lectores hacían la modificación de Desplaza_tira en memoria antes de usarlos. El TIFF 6.0 sugiere que el desplazamiento no debe requerir más de 64K en tamaño.

Cta_byte_Tira (279): La tercera etiqueta, Cta_byte_Tira, mantiene un orden de valores que indica el tamaño de cada tira en bytes. Al igual que la etiqueta Desplaza_tira, esta etiqueta también requiere un orden de valores, uno por tira, y de 2D si se habla de formato de planos, cada uno indica el número de bytes en cada tira.

Esta etiqueta es necesaria porque hay varios casos en el que las tiras de una imagen son de tamaño diferente en bytes. Esto ocurre cuando se usan datos de la imagen ya comprimidos. El valor de Cta_byte_tira es el tamaño de los datos de la imagen después de la compresión. Aunque hay un número fijo de bytes en un fila de un archivo sin comprimir, el tamaño de una fila comprimida varía dependiendo de los datos que se comprimen, esto porque siempre guardamos un número fijo de líneas, no de bytes por tira, esto es probable en la mayoría de tiras, ya que el tamaño de cada fila en la tira variara de longitud una vez comprimidas. Sólo cuando la imagen no sea comprimida el numero de bytes por tira será el mismo.

Algunos escritores de TIFF intentan crear tiras de manera que cada tira en un TIFF tendrá el mismo número de líneas. Por ejemplo, una imagen con una longitud de 2200, puede dividirse en

líneas de 22 cada una y cada una contendrá 100 datos de la imagen. Sin embargo, no siempre es conveniente dividir una imagen en el mismo numero de líneas. Si nosotros queremos dividir una imagen que contiene una longitud de 482 datos y queremos dividirla en cinco tiras, teniendo cinco líneas por cada tira, tendríamos que tener 25 líneas de un mismo tamaño, haciendo la operación tendríamos en cada línea 19 datos excepto en la ultima que tendrá 26 datos, la etiqueta Lineas_x_Tiras tendrá un valor de 5 que estaría correcto para todas las tiras salvo en la última. Aunque un lector de TIFF no necesita saber el número de líneas en cada tira si no el número de bytes por cada tira.

Existen varias ventajas para organizar los datos de una imagen en tiras.

Primero, no todas las aplicaciones pueden leer un archivo entero en memoria. Muchas máquinas actualmente tienen 1 megabyte o menos de memoria disponible. Y aun cuando un sistema tenga mucha memoria, no hay garantía de que el lector pueda usarla. Si se usa la memoria como medio de lectura más rápido se debe tener un buffer en el cual se almacenaran las tiras, este buffer debe tener un tamaño fijo, así que, cuando se manejen datos comprimidos habrá tiras que no son del mismo tamaño y entonces se tendrá que leer cada valor de Desplaza_cad y el valor más grande se le asigne a ese buffer. Por consiguiente, se recomienda que cada tira se limite aproximadamente a 8Kb. de tamaño. Si un lector puede asignar un buffer más grande que 8Kb. pero menos que 64kb. (recordando que una tira pueden tener hasta 64k de datos), entonces se podrán leer tiras múltiples.

Segundo, si se quiere un acceso aleatorio en datos donde se manejen tiras será más fácil.

Azulejos

Las tiras no son el único método de organizar los datos de una imagen TIFF. TIFF 6.0 introdujo el concepto de *Azulejos*[2] en lugar de tiras. Una tira es un objeto unidimensional (una longitud). En cambio, un azulejo puede ser un objeto de 2D. De hecho, se puede pensar en cada azulejo como en una imagen pequeña de la imagen total.

Dividir una imagen en azulejos rectangulares en lugar de tiras horizontales tiene más beneficio en imágenes grandes y de alta resolución. El concepto de azulejo es como un rompecabezas ya que son pequeños trozos rectangulares de la imagen y se pueden tomar de cualquier parte de la imagen, lo que en las tiras no es posible ya que se deben leer las tiras sucesivas hasta completar el trozo deseado.

[2] El concepto de azulejo que maneja TIFF, se puede decir que es como los azulejos de un baño.

Muchos algoritmos de compresión, tal como JPEG, hacen la compresión en azulejos no en tiras como lo hace TIFF. Los datos comprimidos en azulejos perfeccionan la descompresión de los datos por pedazos. De hecho, el apoyo de los algoritmos de 2D es quizás la razón de porque la capacidad de compresión aumenta cuando se utilizan azulejos.

Cuando se usan azulejos en lugar de tiras, las tres etiquetas, Lineas_x_tira, Cta_byte_tira y Desplaza_tira, se reemplazan por las etiquetas Ancho_azulejo (322), Largo_azulejo (323), Desplaza_ azulejo (324) y Cta_byte_azulejo (325).

Ancho_azulejo y Largo_azulejo describen el tamaño de los azulejos en la imagen. Los valores de Ancho_azulejo y Largo_azulejo deben ser de un múltiplo de 16, todos los azulejos de un subarchivo debeb ser siempre del mismo tamaño. El TIFF 6.0 recomienda que estos azulejos deben contener de 4Kb. a 32Kb. de imagen. No es necesario que los datos estén en forma cuadrada, la compresión es igual tanto para azulejos cuadrados como rectangulares. Las etiquetas Ancho_azulejo y Largo_azulejo se pueden usar para determinar el número de azulejos en una imagen que no utilice el tipo de color YCbCr:

Azulejo_Alto= (Ancho_Imagen + (Ancho_azulejo - 1)) / Ancho_azulejo

Azulejo_Bajo= (Largo_Imagen + (Largo_azulejo - 1)) / Largo_azulejo

Azulejos_en_imagen = Azulejo_Alto * Azulejo_Bajo

Si se utilizan en imágenes de 2 planos, el calculo de azulejos será como sigue:

Azulejos_en_imagen = Azulejo_Alto * Azulejo_Bajo * Mues_x_pixel

Desplaza_azulejo contiene un orden de desplazamientos desde el primer byte de cada azulejo. No se guardan azulejos necesariamente en una sucesión inmediata. Cada azulejo tiene una situación separada y está independiente de todos los otros azulejos. Los desplazamientos en esta etiqueta se mandan en forma izquierda a derecha y de arriba a abajo. Si los datos de la imagen son separados en planos, todos los desplazamientos del primer plano se guardan primero, seguidos de los desplazamientos del segundo plano, y así sucesivamente.

Finalmente la etiqueta, Cta_byte_azulejo, contiene el número de bytes en cada azulejo comprimido. El número de valores en esta etiqueta es igual a el número de etiquetas en la imagen, y se mandan de la misma manera como los valores de la etiqueta Desplaza_azulejo.

Normalmente, un azulejo se clasifica según su tamaño. Una imagen de 6400 pixeles de ancho por 4800 pixeles de largo se divide uniformemente en 150 azulejos, cada uno de 640 pixeles de ancho por

320 pixeles de largo. Sin embargo, no todas imágenes son divisibles por 16. Una imagen de 2200 pixeles de ancho por 2850 de largo no se puede dividir uniformemente en azulejos cuyos tamaño debe ser múltiplos de 16. La solución es escoger un buen tamaño del azulejo y con los datos faltantes de un azulejo hacer un relleno.

Para encontrar un buen tamaño de azulejo, debemos escoger un azulejo, clasificándolo según el tamaño mínimo que normalmente se encuentra en una imagen. En este ejemplo usaremos azulejos de 256 pixeles de ancho por 320 pixeles de largo. Usar azulejos de este tamaño requiere de 104 pixeles extra de relleno que se aumente en cada línea a lo ancho y 30 líneas adicionales a lo largo. El tamaño de los datos de la imagen relleno es ahora 2304 pixeles de ancho por 2880 pixeles largo y se puede dividir uniformemente entre 81 azulejos de 256 por 320 pixeles.

La figura III.18 se ilustra las caracteristicas tanto de las tiras como de los azulejos.

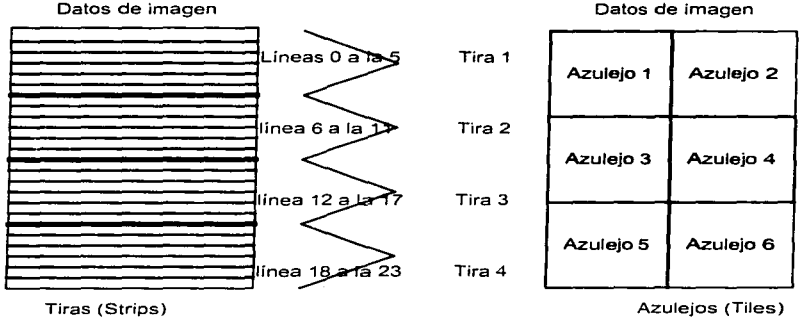


Figura III. 18 Almacenamiento de datos en tiras y azulejos.

III.6.4 Compresión

TIFF es un formato que utiliza varios métodos de compresión. quizás es el que más utiliza a diferencia de los otros tipos de formatos de archivo. TIFF 4.0 solo utilizo el método de compresión PackBits (una variación de RLE) y CCITT T.4 y T.6. Estos esquemas de compresión son sólo para usarse en 8 bits de color, en escala de grises y 1 bit de color (imágenes en blanco y negro), respectivamente. TIFF 5.0 agregó el LZW en forma de planos para imágenes en color, y TIFF 6.0 agregó el método de compresión JPEG para imágenes de color total y escala de grises.

Packbits

TIFF usa la compresión PackBits RLE de Macintosh Toolbox. PackBits es un algoritmo simple, eficaz y fácil de llevar a cabo. PackBits es un RLE eficaz que utiliza la codificación de corrimiento de bytes.

PackBits realmente tiene tres tipos de paquetes. El primero es un paquete de dos bytes. El primer byte (byte contador de corrimiento) indica el número de bytes que están empaquetados, y el segundo byte indica lo que esta empaquetado por cada valor del primer byte. El contador de corrimiento está en el rango de 0 a 127 y representa los valores 1 a 128 (cuenta de corrimiento +1).

Otro tipo de paquete, es el paquete de corrimiento literal[3], reserva de 2 a 128 bytes literalmente en datos sin comprimir. Corrimiento Literal guarda datos de paquetes con pocos corrimientos, cuando se encuentran imágenes complejas, donde no existan bytes consecutivos iguales . El corrimiento literal cuenta un corrimiento en el rango de -127 a -1, indica valores de 2 a 128 (-(contador de cuneta)+ 1), los cuales indican que no se pueden comprimir.

El último tipo de paquete es el paquete de No-op. No-op es de un byte en longitud y tiene un valor de 128. El paquete de No-op no es útil en la compresión PackBits, por consiguiente, nunca se encontrara en una compresión de PackBits.

Usemos un algoritmo para representar el método de compresión PackBits que utiliza TIFF.

```
Escribir los datos de la cabecera y el conjunto de etiquetas usadas
Comenzar a leer desde el principio de la imagen
Repetir
   Leer un dato de la imagen (dato = pixel).
   Contar hasta un máximo de 128 datos posibles que se repitan.
            Si no hay Repetición
             Contar cuantos datos de los siguientes no se pueden comprimir (no mas de 128)
                    al resultado obtenido restarle 1 y multiplicarlo por -1
             Escribir  el resultado de la operación y a continuación los datos que no se comprimieron
            Si no
     Escribir el resultado de la cuenta-1 y a continuación el byte que se contó
            Fin del si
Hasta  leer todos los datos de la imagen
```

[3] Este tipo de paquete es como el que utiliza TGA al cual le llama Raw Packets.

Nota: Los datos de la imagen deben ser contados como si fuera una tira, recuerde que la tira se compone de un número determinado de líneas. En TIFF cuando se utilizan tiras, la compresión se hace para cada tira como si fuera una imagen individual.

Por ejemplo, hagamos la compresión de los siguientes datos utilizando el algoritmo:

$Pixel_1$	$Pixel_2$	$Pixel_3$	Pixel4	Pixel5	Pixel6	Pixel7	Pixel8	Pixel...
254	5	10	10	10	10	10	192	192

Línea de pixeles sin codificar

Escribir los datos de la cabecera y el conjunto de etiquetas usadas
Comenzar a leer desde el principio de la imagen
Repetir
  Leer un dato la imagen (**254**).
  Contar hasta un máximo de 128 datos posibles que se repitan. (**no hay repetición**)
    Si no hay Repetición
      Contar cuantos datos de los siguientes no se pueden comprimir (en este caso 2 (254,5))
      al resultado obtenido restarle 1 y multiplicarlo por -1 (**((2-1) * (-1)) =-1**)
    Escribir el resultado de la operación y a continuación los datos que no se comprimieron **(-1 , 254 , 5** si se utilizan bytes sin signo el valor de -1 representado en bits seria 111111111[4] = 511 y en realidad los bytes que se verán en el archivo físicamente hablando serán **255, 254, 5)**
    Si no
  Escribir el resultado de la cuenta-1 y a continuación el byte que se contó
    Fin del si
Leer el siguiente dato de la imagen (**10**).
  Contar hasta un máximo de 128 datos posibles que se repitan. (**5 repeticiones**)
    Si no hay Repetición
      Contar cuantos datos de los siguientes no se pueden comprimir
      al resultado obtenido restarle 1 y multiplicarlo por -1
    Escribir el resultado de la operación y a continuación los datos que no se comprimieron
    Si no
  Escribir el resultado de la cuenta-1 y a continuación el byte que se contó (4, **10**)
    Fin del si
Leer un dato la imagen (**192**).
  Contar hasta un máximo de 128 datos posibles que se repitan. (**2 repeticiones**)

[4] Cuando se manejan bytes con signo el S. O. utiliza un noveno bit para el signo. "1" cuando son valores negativos y "0" cuando son positivos (de un repaso a el sistema binario y la forma de manejo del S. O.).

Si no hay Repetición
Contar cuantos datos de los siguientes no se pueden comprimir
al resultado obtenido restarle 1 y multiplicarlo por -1
Escribir el resultado de la operación y a continuación los datos que no se comprimieron
Si no
Escribir el resultado de la cuenta-1 y a continuación el byte que se contó **(1, 192)**
Fin del si
Hasta leer todos los datos de la imagen

Hecha la compresión los datos quedan como sigue:

$Pixel_0$	$Pixel_1$	$Pixel_2$	$Pixel_3$	$Pixel_4$	$Pixel_5$	$Pixel_6$	$Pixel_7$	$Pixel_8$
254	5	10	10	10	10	10	192	192

Línea de pixeles sin codificar

$Dato_0$	$Dato_1$	$Dato_2$	$Dato_3$	$Dato_4$	$Dato_5$	$Dato_6$	$Dato_7$	$Dato_8$
255	254	5	4	10	1	192	...	...

Línea de datos codificados

Nota: Hay que tomar en La descompresión de PackBits se hace de manera simple, ya que solo es una lectura de paquetes de bytes que se convierten en un corrimiento de bytes. Nuevamente tomemos un algoritmo para ejemplificar el método de descompresión.

cuenta el tipo de dato que tomaran los datos que leeremos del archivo, es decir, bytes con o sin signo, ya que se debe hacer una operación para obtener los bytes necesarios para la descompresión o para los datos no comprimidos.

Cuando manejamos un lenguaje como pascal existe un tipo de dato llamado SHORTINT (-128 a 127) si utilizamos este tipo de dato los datos negativos que se leen son negativos, pero si utilizamos el tipo de dato BYTE, también de pascal, los datos negativos que leamos serán representados como valores regresivos de 256 que es 0, 255 será el valor de -1, 254 el valor -2, etc. , así que, si utilizamos este tipo de dato solo tendremos que hacer la siguiente operación (dato leído -256) o si queremos de una vez el valor positivo, para evitarnos la multiplicación por -1 (256 - dato leido).

Leer los datos de la cabecera y el conjunto de etiquetas usadas
Localizamos el inicio de la imagen (Recuerde que va a utilizar Tiras o Azulejos)
Repetir
Leer un dato la imagen (Suponemos que utilizaremos el tipo SHORTINT como lo maneja pascal).

Hacer Cuantos = dato
Si Cuantos es mayor que cero
Hacer Cuantos = Cuantos+1
Leer el siguiente dato
Desplegar este dato tantas veces como tenga "Cuantos"
Si no
Hacer cuantos = (cuantos * (-1))+1
Leer tantos datos como veces tenga "Cuantos" y desplegarlos uno a uno
Fin del si
Hasta leer todos los datos de la imagen

Utilizando el algoritmo de descompresión para el ejemplo mencionado:

Leer los datos de la cabecera y el conjunto de etiquetas usadas
Localizamos el inicio de la imagen (Recuerde que va a utilizar Tiras o Azulejos)
Repetir
Leer un dato la imagen (Suponemos que utilizaremos el tipo SHORTINT como lo maneja pascal) **(1)**.
Hacer Cuantos = dato (**Cuantos=-1**)
Si es mayor que cero
Hacer Cuantos = Cuantos+1
Leer el siguiente dato
Desplegar este dato tantas veces como tenga "Cuantos"
Si no
Hacer cuantos = (cuantos * (-1))+1 (**Cuantos = (Cuantos*-1)+1= 2**)
Leer tantos datos como veces tenga "Cuantos" y desplegarlos uno a uno (**254,5**)
Fin del si
Leer el siguiente dato de la imagen
Almacenar este numero en "Cuantos" (**Cuantos=4**)
Si es mayor que cero
Hacer Cuantos = Cuantos+1 (**Cuantos=cuantos+1=5**)
Leer el siguiente dato (**10**)
Desplegar este dato tantas veces como tenga "Cuantos" (**10,10,10,10,10**)
Si no
Hacer cuantos = (cuantos * (-1))+1
Leer tantos datos como veces tenga "Cuantos" y desplegarlos uno a uno
Fin del si
Leer el siguiente dato de la imagen

```
Almacenar este numero en "Cuantos" (Cuantos=1)
        Si es mayor que cero
                Hacer Cuantos = Cuantos+1 (Cuantos=cuantos+1=2)
                Leer el siguiente dato  (192)
                Desplegar este dato tantas veces como tenga "Cuantos" ( 192, 192 )
        Si no
                Hacer cuantos = (cuantos * ( -1))+1
        Leer tantos datos como veces tenga "Cuantos" y desplegarlos uno a uno
        Fin del si
Hasta  leer todos los datos de la imagen
```

CCITT T.4 y T.6.

El CCITT (International Telegraph and Telephone Consultative Committee "Comité Internacional Consultativo de Telégrafo y Teléfono") es un organización de normas que ha desarrollado una serie de protocolos de comunicaciones para la transmisión de imágenes de facsímil en blanco y negro mediante líneas telefónicas y redes de computadoras. Oficialmente estos protocolos se llaman normas CCITT T.4 y T.6 pero normalmente se le llaman compresión CCITT grupo 3 y grupo 4 respectivamente. El CCITT grupo 3 y grupo 4 (T.4 y T.6 respectivamente) tienen las características técnicas sobre los protocolos de comunicaciones. La mayor parte del método de compresión se copia directamente de las características técnicas del CCITT.

A veces CCITT se refiere a una compresión no con mucha precisión como lo hace HUFFMAN. La compresión de Huffman es un algoritmo de compresión que fue realizado por David Huffman en 1952. CCITT es de una dimensión de un tipo específico de Huffman. Los otros tipos de CCITT no lo son, sin embargo, utilizan aplicaciones del Huffman en forma de planos.

Compresión en Forma de Planos

Una línea (fila) de datos se compone de una serie de palabras de compresión inconstantes. Cada palabra de codificación representa una longitud de datos en blanco o de datos en negro.

Para asegurar que el receptor (descompresor) mantenga una sincronización de colores, todas las líneas de los datos empezarán con una código de corrimiento fijo blanco o negro.

Las palabras de la codificación son de dos tipos: Palabra "código de terminación" y Palabra "Código de Make-up". Cada código de corrimiento es representado por ceros o varios Make-up´s

seguidos por un código de terminación .

Las longitudes de cada corrimiento están en el rango de 0 a 63 pixeles cada uno con su propio código de Terminación. Las longitudes de corrimiento que están en el rango de 64 a 2623 (2560+63) se pone en un código Make-up seguido por el código de terminación.

Las longitudes de corrimiento que son mayores o iguales a 2624 son codificados por make-up´s de 2560. Si la parte restante del corrimiento (después de la primera codificación del Make-up 2560) está en 2560 o más, el (los) Make-up´s adicional(es) de 2560 se emiten hasta que la parte restante del corrimiento vuelva a menos de 2560. Entonces se pone en código la parte restante del corrimiento y el código de terminación.

Observe las tablas III.11 ,III.12 ,III.13 que muestran los códigos antes explicados.

Tabla III.11 Tabla de códigos de terminación del CCITT.			
Blanco		Negro	
Código de corrimiento	Palabra	Código de corrimiento	Palabra
0	00110101	0	0000110111
1	000111	1	010
2	0111	2	11
3	1000	3	10
4	1011	4	011
5	1100	5	0011
6	1110	6	0010
7	1111	7	00011
8	10011	8	000101
9	10100	9	000100
10	00111	10	0000100
11	01000	11	0000101
12	001000	12	0000111
13	000011	13	00000100
14	110100	14	00000111
15	110101	15	000011000
16	101010	16	0000010111
20	0001000	20	00001101000
21	0010111	21	00001101100
22	0000011	22	00000110111
23	0000100	23	00000101000
24	0101000	24	00000010111
25	0101011	25	00000011000
26	0010011	26	000011001010
27	0100100	27	000011001011
28	0011000	28	000011001100
29	00000010	29	000011001101
30	00000011	30	000001101000
31	00011010	31	000001101001

Tabla III.11 Tabla de códigos de terminación del CCITT. (continuación)			
Blanco		Negro	
Código de corrimiento	Palabra	Código de corrimiento	Palabra
32	00011011	32	000001101010
33	00010010	33	000001101011
34	00010011	34	000011010010
35	00010100	35	000011010011
36	00010101	36	000011010100
37	00010110	37	000011010101
38	00010111	38	000011010110
39	00101000	39	000011010111
40	00101001	40	000001101100
41	00101010	41	000001101101
42	00101011	42	000011011010
43	00101100	43	000011011011
44	00101101	44	000001010100
45	00000100	45	000001010101
46	00000101	46	000001010110
47	00001010	47	000001010111
48	00001011	48	000001100100
49	01010010	49	000001100101
50	01010011	50	000001010010
51	01010100	51	000001010011
52	01010101	52	000000100100
53	00100100	53	000000110111
54	00100101	54	000000111000
55	01011000	55	000000100111
56	01011001	56	000000101000
57	01011010	57	000001011000
58	01011011	58	000001011001
59	01001010	59	000000101011
60	01001011	60	000000101100
61	00110010	61	000001011010
62	00110011	62	000001100110
63	00110100	63	000001100111
64	11011	64	0000001111
128	10010	128	000011001000
192	010111	192	000011001001
256	0110111	256	000001011011
320	00110110	320	000000110011
384	00110111	384	000000110100
448	01100100	448	00000011010 1
512	01100101	512	0000001101100
576	01101000	576	0000001101101
640	01100111	640	0000001001010
704	011001100	704	0000001001011
768	011001101	768	0000001001100
832	011010010	832	0000001001101
896	011010011	896	0000001110010

Tabla III.11 Tabla de códigos de terminación del CCITT. (continuación)			
Blanco		Negro	
Código de corrimiento	Palabra	Código de corrimiento	Palabra
960	011010100	960	0000001110011
1024	011010101	1024	0000001110100
1088	011010110	1088	0000001110101
1152	011010111	1152	0000001110110
1216	011011000	1216	0000001110111
1280	011011001	1280	0000001010010
1344	011011010	1344	0000001010011
1408	011011011	1408	0000001010100
1472	010011000	1472	0000001010101
1536	010011001	1536	0000001011010
1600	010011010	1600	0000001011011
1664	011000	1664	0000001100100
1728	010011011	1728	0000001100101
EOL	000000000001	EOL	000000000001

Tabla III.13 Tabla de códigos adicionales Make-Up del CCITT.	
Blanco y Negro	
Código de corrimiento	Palabra
1792	00000001000
1856	00000001100
1920	00000001101
1984	000000010010
2048	000000010011
2112	000000010100
2176	000000010101
2240	000000010110
2304	000000010111
2368	000000011100
2432	000000011101
2496	000000011110
2560	000000011111

LZW

TIFF utiliza el mismo método de compresión que GIF (LZW[5]), además del LZW emplea otra compresión, y así los datos comprimidos se codifican nuevamente reduciendo aun más la cantidad de datos. esta compresión no se encuentra muy distribuida.

[5] Vea el método de compresión de GIF.

JPEG

Otro método de compresión utilizado es el de JPEG este método solo lo utiliza TIFF 6.0 y no es muy común actualmente, si se encuentran imágenes con compresión JPEG puede tener algunos problemas, según el Dr. Tom Lane miembro de un grupo independiente del JPEG y del comité asesor del TIFF.

Problemas con TIFF 6.0 JPEG

TIFF 6.0 añadió JPEG a la lista de esquemas de compresión de TIFF. Desafortunadamente, el camino tomado en la especificación 6.0 es un diseño muy pobre. Un nuevo diseño ha sido desarrollado por el Comité Consultivo TIFF. Si está considerando implementar JPEG en TIFF, cheque la revisión descrita en TIFF Tech Note #2 de la especificación 6.0.

El problema fundamental con el diseño TIFF 6.0 JPEG es que las tablas y parámetros variables de JPEG estallan como campos separados, que el control lógico de TIFF debe manipular. Es una ingeniería de Software mala ya que la información debe ser privada al compresor/descompresor JPEG. Lo peor, los campos por sí mismos son especificados sin haber pensado en futuras extensiones y sin mirar las convenciones establecidas por TIFF. Aquí están algunos de los problemas más significantivos:

- Los campos de tablas de JPEG utilizan un esquema sumamente no estandarizado: prefieren que los datos estén contenidos directamente en la estructura del campo; los campos mantienen punteros de información en otra parte en el archivo. Esto requiere códigos de propósito especial para ser añadidos a cada aplicación que leen TIFF. Incluso un editor trivial TIFF (por ejemplo, un programa para añadir un campo descriptor de imagen a un archivo TIFF) debe estar explícitamente consciente de la estructura interna de las tablas JPEG relacionadas, o probablemente terminará el archivo. Cada otro campo auxiliar en TIFF sigue las reglas normales de TIFF y puede ser copiado o relocalizado por un código estándar.
- La especificación requiere el control lógico TIFF para conocer un estupendo convenio acerca de los detalles JPEG, por ejemplo, calcular la longitud de una tabla de código Huffman. La longitud no está implicada en la estructura del campo y puede ser encontrada solo inspeccionando la tabla de contenidos.
- El diseño especifica tablas separadas Huffman para cada componente de color. Esto falta al hecho de que el codec baseline JPEG puede soportar solo 2 juegos de tablas Huffman. Por los

tanto, un decodificador debe, o gastar el espacio o violar la convención de TIFF que prohibe duplicar punteros. Además, decodificadores baseline deben revisar para ver que tablas son idénticas, un gasto de tiempo y espacio de código.

- El campo Formato de Intercambio JPEG violo de nuevo la descripción en contra de duplicar punteros; concibe tener el puntero normal de tira o azulejo apuntando a áreas de datos largas apuntadas para el Formato de Intercambio JPEG. Aplicaciones TIFF deben estar conscientes de estas relaciones, ya que deben mantenerlas ó, si no pueden, deben borrar el campo Formato de Intercambio JPEG.
- El campo Tabla JPEG es fijo en 1 byte por entrada en tabla; no hay modo para soportar valores de cuantización de 16 bits. Esto es un serio impedimento para extender TIFF para usar JPEG de 12 bits.
- El diseño no puede soportar utilizar diferentes tablas de cuantización en diferentes tiras o azulejos de una imagen (así como codificar algunas áreas de más alta calidad que otras). Además, ya que las tablas de cuantización son vinculadas una por una a los componentes de color, el diseño no puede soportar opción de intercambio entre tablas que son parecidas para ser añadidas en futuras extensiones JPEG.

En adición a estos principales errores de diseño, la especificación TIFF 6.0 JPEG es seriamente ambigua. En particular, algunas interpretaciones incompatibilidades son posibles para su manejo de marcadores de reinicio JPEG, la compresión JPEG contradice las imagen en azulejos en lo que concerniente a las restricciones de tamaño.

Finalmente, el diseño de 6.0 crea problemas para las implementaciones que necesitan mantener el codec JPEG separado del control lógico TIFF, considere usar un chip JPEG que no fue diseñado especificamente para TIFF. Codec JPEG generalmente quiere producir o consumir una cadena de datos estándar JPEG, no solo datos brutos. (Si manejan datos brutos, un mecanismo separado fuera de banda debe ser provisto para cargar las tablas en el codec). Con tal codec, el control lógico TIFF debe estar preparado para traducir marcadores JPEG para crear los campos de tablas TIFF (cuando codifica) o sintetizar marcadores JPEG de los campos (cuando decodifica). Claro, esto significa que el control lógico debe conocer más acerca de JPEG de lo que le gustaría. La traducción y reconstrucción de los marcadores representa una cantidad de trabajo innecesario.

Debido a todos estos problemas, el Comité consultivo TIFF ha desarrollado un reemplazo del esquema JPEG en TIFF. El preliminar es como sigue:

1. Cada segmento de imagen (tira o azulejo) en un imagen TIFF compresa en JPEG contiene una cadena de datos JPEG, completa con todos los marcadores. Estos datos forman una imagen independiente de dimensiones propias para la tira o azulejo.
2. Evita duplicar tablas en un archivo multisegmentos, los segmentos pueden usar estructura de cadena de datos "datos de imagen abreviados", en que las tablas DQT y DHT son omitidas. Las tablas comunes son suplidas en una cadena de datos "datos de imagen abreviados", que está contenida en un campo nuevo Tablas JPEG. Ya que las tablas en cuestión tipicamente suman 550 bytes.
3. Todas las definiciones de campos en la compresión JPEG son eliminados. En práctica, esta etiquetas de campos permanecen reservado indefinidamente, y este esquema usará un nuevo código de compresión, (Compresión = 7).

III.7 Formato TGA

Nombre	TGA
Conocido como	Truevision Graphics Adapter, Targa Graphics Adapter Image File, VST, VDA, ICB, TPIC.
Tipo	Bitmap
Colores	8, 16, 24 y 32 Bits.
Compresión	RLE, Sin comprimir.
Tamaño Máximo de Imagen	No
Imágenes Múltiples por Archivo	No
Orden de Byte	Little-endian
Creador	Truevison, Inc.
Plataforma	MS-DOS, Windows, UNIX, Atari, Amiga, y otros.
Aplicaciones que soporta	Demasiado numerosas para listarlas.

III.7.1 Antecedentes

El éxito del formato TGA se debe a que almacena imágenes de color que pueden ser manejadas fácilmente, además utiliza una pequeña cantidad de memoria para almacenar el archivo, de hecho fue el primer archivo de formato de color total (True color) disponible.

Truevision definió el formato TGA en 1984 para ser usado con sus primeros productos vídeo gráficos. Desde entonces se estima que actualmente el 80 % de las imágenes a color almacenadas emplean alguna variación del formato TGA. El formato puede ser usado por software desarrollado por Truevison en combinación con algunas otras aplicaciones para una mayor flexibilidad y logro de soluciones adecuadas a cada necesidad.

El formato Truevision TGA original ha sido aceptado ampliamente por la industria de los gráficos. Sin embargo, tecnologías y técnicas recientes han creado la necesidad de almacenar información adicional de la imagen en el archivo. En 1989 Truevision introdujo extensiones al TGA para satisfacer las demandas hechas por la industria de gráficos y futuras necesidades de color. Las extensiones son opcionales y no tendrán impacto en paquetes existentes (si los paquetes siguieron las pautas del Formato el TGA original). En particular el nuevo TGA está enfocado a las siguiente necesidades:

- La inclusión de una copia escalada-abajo "postage stamp" de la imagen.
- Fecha y hora de creación del archivo de la imagen.
- Nombre del autor.
- Comentarios del autor.
- Nombre del trabajo.
- Tiempo de trabajo acumulado.
- Valor de gamma
- Corrección de color LUT (LookUp Table)
- Aspect ratio del pixel.
- Tabla scan line offset.
- Color llave.
- Nombre del software y número de versión.
- Diseño de áreas definidas.
- Tipo del canal de atributo (Alfa).
- La habilidad para expansión simple.

III.7.2 Estructura

La figura III.18 muestra un diagrama a bloques de la estructura en un archivo con formato TGA.

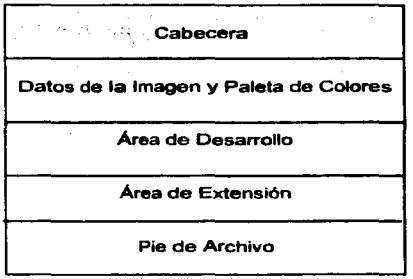


Figura III. 13 Estructura del archivo TIFF.

Los tres últimos bloques Área de desarrollo, Área de Extensión y Pie de Archivo son incluidos en la especificación desde Septiembre de 1989.

Cabecera

Campo 1. Tamaño del campo de Identificación (1 byte, byte 0). Especifica el número de bytes contenidos en el campo 6, campo Identificación de Imagen. El número máximo de caracteres es 255. Un valor cero indica que no hay Identificación de la imagen incluida en el archivo.

Campo 2. Tipo de paleta (1 byte, byte 1). Indica el tipo de paleta de colores (si es que está) incluida con la imagen. Generalmente hay dos valores definidos para este campo:

0 Indica que no hay paleta incluida en la imagen
1 Indica que hay paleta incluida en la imagen

Los primeros 128 códigos del tipo de paleta están reservados para uso de Truevision, mientras que el segundo juego de 128 códigos (128 a 255) pueden ser usados por las aplicaciones del diseñador.

Imágenes con color total generalmente no usan el campo de paleta de colores, pero algunas aplicaciones actuales almacenan la información de la paleta o la información del desarrollo definido en este campo. Es mejor checar el campo 3, Tipo de imagen, para asegurar que se tiene un archivo que puede usar los datos almacenados en el campo Tipo de Paleta de Colores. De otro modo, ignore la información. Cuando se guardan o crean imágenes con color total no se usa el campo y se coloca un '0' para asegurar la compatibilidad. Por favor vea la especificación del área de desarrollo para la información de métodos de desarrollo de almacenamiento definidos.

Campo 3. Tipo de imágen (1 byte, byte 2). El formato TGA puede ser usado para almacenar imágenes Pseudo-color, True Color (Color total) y Direct-color (Color directo), de varios anchos de pixel. Truevision tiene actualmente definidos los tipos de imagen mostrados en la tabla III.14

Los códigos 0 a 127 del dato tipo de imagen están reservados para uso de Truevision para aplicaciones generales. Los códigos de 128 a 255 pueden ser usados para aplicaciones del diseñador.

Tabla III.14 Tipo de imágenes de TGA.	
Tipo de imagen	**Descripción**
0	Datos de imagen no incluidos
1	Sin compresión, Color-Mapped
2	Sin compresión, True Color
3	Sin compresión, Blanco y Negro
9	Compresión RLE, Color-Mapped
10	Compresión RLE, True Color
11	Compresión RLE, Blanco y Negro
32	Compresión RLE, Delta, Huffman, Color-Mapped
33	Compresión RLE, Delta, Huffman, Color-Mapped, cuatro tipo de procesos

Campo 4. Especificación de la paleta (5 bytes, bytes 3-7). Este campo y sus subcampos describen la paleta de colores (si la hay) usada para la imagen. Si el campo tipo de paleta está puesta en cero, indica que no hay paleta, entonces estos 5 bytes deben ser '0', estos bytes siempre deben estar escrito en el archivo.

Campo 4.1. Indice de la primera entrada (2 bytes, bytes 3-4). El índice se refiere a la entrada de inicio al cargar la paleta. Por ejemplo, si tiene 1024 entradas en la paleta completa pero solo necesita almacenar 72 de estas entradas, este campo le permite iniciar en la mitad de la paleta.

Campo 4.2. Largo de la paleta (2 bytes, bytes 5-6). Número total de entradas incluidas en la paleta.

Campo 4.3. Tamaño de las entradas en la paleta (1 byte, byte 7). Establece el número de bits por entrada. Comúnmente valores de 15, 16, 24 ó 32 bits son usados.

Campo 5. Especificación de la imagen (10 bytes, byte 8-17). Este campo y sus subcampos describen ancho de pixel, tamaño y localización en pantalla de la imagen.

Campo 5.1. Origen X (2 bytes, bytes 8-9). Indica la coordenada horizontal X para la esquina superior izquierda de la imagen y su posición en el dispositivo de despliegue, teniendo como origen la esquina superior izquierda de la pantalla.

Campo 5.2. Origen Y (2 bytes, bytes 10-11). Indica la coordenada vertical Y para la esquina superior izquierda de la imagen y su posición en el dispositivo de despliegue, teniendo como origen la esquina superior izquierda de la pantalla

Campo 5.3. Ancho de la Imagen (2 bytes, bytes 12-13). Indica el ancho de la imagen en pixeles.

Campo 5.4. Largo de la Imagen (2 bytes, byte 14-15). Indica el largo de la imagen en pixeles.

Campo 5.5. Ancho del pixel (1 byte, byte 16). Indica el número de bits por pixel. Este número incluye bits de atributo o canal alfa. Los valores comunes son 8, 16, 24 y 32 pero pueden ser usados otros anchos de pixel.

Campo 5.6. Descriptor de imagen (1 byte, byte 17).

Bits 3-0. Especifican el número de atributos por pixel (canal alfa).

Bits 5-4. Indican el orden en que el dato del pixel es transferido desde el archivo a la pantalla. El bit 4 es para el orden de izquierda a derecha y el bit 5 para el orden de arriba a abajo, como se muestra en la tabla III.15.

Tabla III.15 Origen de la Imagen.

Destino en pantalla del primer pixel	Bit 5	Bit 4
izquierda inferior	0	0
derecha inferior	0	1
izquierda superior	1	0
derecha superior	1	1

Bits 7-6. Deben ser 0 para asegurar futuras compatibilidades.

Datos y Paleta de la Imagen

Campo 6. Identificación de imagen (variable). Este campo es opcional, contiene información sobre la imagen. El tamaño máximo de este campo es 255 bytes. Cheque el campo 1 para verificar el tamaño de este campo. Si el campo 1 está puesto en '0' indica que la identificación de imagen no existe entonces este byte no son escritos, es decir, no existen en el archivo.

Campo 7. Datos de la paleta de colores (variable). Si el campo 2, tipo de paleta, está puesto en '0' indica que no hay paleta de color por lo tanto éste campo no existe.

La longitud de este campo varia ya que contiene la información de la paleta actual (datos LUT). El campo 4.3 específica el tamaño en bits de cada entrada en la paleta; el campo 4.2 indica el número de entradas en la paleta para el campo 7. Por lo tanto, los campos 4.2 y 4.3 determinan el número de bytes que contiene el campo 7.

Cada entrada en la paleta está almacenada utilizando un número entero de bytes. La especificación RGB para cada entrada en la paleta es almacenada en campos de bits

sucesivos en la entrada multibyte. Si el campo 4.3 contiene un 24, entonces cada especificación de color es de 8 bits de largo; si contiene un 32, cada color es también de 8 bits (32/3 da 10, pero 8 es más pequeño). Los bits sin usar se especifican como bits de atributos. El bit de atributo en ocasiones es conocido como canal alfa, bit(s) de *overlay*, o bit(s) de interrupción.

ATVista y NuVista, el número de bits en la paleta es 24 ó 32. Los componentes de rojo, verde y azul son representados cada uno por un byte.

Campo 8. Datos de imagen (variable). Indica el ancho y largo en pixeles de la imagen. Cada pixel especifica un dato de imagen en alguno de los siguientes formatos: un índice de paleta sencillo para pseudo-color, atributo, datos en orden rojo, verde, azul para color total, e índices independientes de paleta para Color Directo. Los valores para el ancho y largo están definidos en los campos 5.3 y 5.4 respectivamente.

El número de bits de atributo y definición de color para cada pixel están definidos en los campos 5.5 y 5.6 respectivamente. Cada pixel es almacenado como un número entero de bytes.

Área de Desarrollo

Campo 9. Datos de desarrollo (variable). Truevision creó el área de desarrollo para satisfacer las necesidades de los desarrolladores que ya han creado áreas similares, en un intento de ser compatibles con estos métodos de desarrollo.

El área de desarrollo es un área que puede ser usada para almacenar algún tipo de información, sin embargo, se recomienda sea usada solo para información especifica de la aplicación, es información que no será aplicable a otros productos y que no ha sido cubierta por el formato TGA.

El tamaño y desarrollo de éste campo es totalmente dependiente del desarrollador. Los lectores de este formato deben ignorar este campo a menos que tengan conocimiento de su organización y contenido.

Ya que un archivo puede contener más de un área de desarrollo, fue creado un directorio de desarrollo que contienen una localización de los campos incluidos en ésta área. Es localizado usando el puntero de desplazamiento contenido en el pie del archivo TGA. El contenido de este campo esta en un byte de desplazamiento desde el inicio del

archivo al inicio del directorio de desarrollo. Si el desplazamiento es '0' (cero binario, claro) no existe ni directorio ni área de desarrollo.

Área de Extensión

El área de extensión fue creada para satisfacer los requerimientos de los desarrolladores para información adicional del archivo. Es localizada por el desplazamiento del área de desarrollo en el pie de archivo TGA. Si el desplazamiento es cero, entonces no existe área de extensión. De otro modo , este valor es el desplazamiento desde el inicio del archivo al inicio del área de extensión.

La combinación del área de extensión y de desarrollo deben servir a Truevision y a su comunidad de desarrolladores para bienes futuros. Sin embargo la tecnología puede dictar cambios a la especificación deberán ser completamente fáciles de implementar. El primer campo del área de extensión es el campo tamaño del área de extensión. Este campo actualmente contiene el valor 495, indicando que hay 495 bytes en una longitud de porción fija del área de extensión. Si futuros desarrolladores ordenan campos adicionales, entonces este valor puede cambiar para indicar que han sido añadidos nuevos campos. Algún cambio en el número de bytes en el área de extensión será hecho por Truevision y serán retransmitidos a la comunidad de desarrolladores por vía de una revisión de la especificación del formato.

Campo 10. Tamaño de extensión (2 bytes, variable). Especifica el número de bytes del área de extensión. Para el formato TGA v2.0, este número deberá ser 495. Si el número encontrado no es 495, entonces se asume que la versión es mayor a la v2.0, el cambió será controlado por Truevision y será acompañado por una revisión del Formato acompañada de un cambio en el número de versión.

Campo 11. Nombre del autor (41 bytes, variable). Es un campo ASCII de 41 bytes donde el último byte debe ser un nulo (cero binario), dando un total de 40 caracteres ASCII para el nombre. Si el campo es usado, debe contener el nombre de la persona que creó la imagen (autor). Si el campo no es usado, se podrá llenar de nulos o series de blancos (espacios) terminados por un nulo. El 41° byte siempre debe ser un nulo.

Campo 12. Comentarios del autor (324 bytes, variable). Es un campo ASCII que consiste de 324 bytes organizados en 4 líneas de 80 caracteres, cada una seguida de un terminador nulo. Este campo está incluido, en forma adicional al campo original Identificación de imagen (en el formato TGA original) ya que se determino que algunos desarrolladores utilizaban

el campo Identificación de Imagen para sus propios requerimientos. Este campo proporciona al desarrollador cuatro líneas de 80 caracteres cada una para usar como un área de comentarios del autor. Si no se utilizan los 80 caracteres en la línea, coloque un caracter nulo después de su último caracter y blancos o nulos para el resto de la línea.

Campo 13. Fecha y hora marca (12 bytes, variable). Especifica una serie de valores que definen un valor entero para la fecha y hora en que la imagen fue salvada. Los datos tienen la siguiente estructura:

1 (bytes 0-1)	Mes	1-12
2 (bytes 2-3)	Día	1-31
3 (bytes 4-5)	Año	4 dígitos (1997)
4 (bytes 6-7)	Hora	0-23
5 (bytes 8-9)	Minuto	0-59
6 (bytes 10-11)	Segundos	0-59

Este campo se proporciona ya que el Sistema Operativo suele cambiar la fecha y hora del archivo si éste es copiado. El uso de éste campo le garantiza no modificar la fecha y hora ya grabada. Si no usa estos campos, los debe llenar con ceros.

Campo 14. Nombre identificación del trabajo (41 bytes, variable). Es un campo ASCII de 41 bytes donde el último byte debe ser un cero. Dando un total de 40 caracteres ASCII para el nombre del trabajo o identificación. Si el campo es usado, debe contener una etiqueta del nombre o identificación que refiera al trabajo con el que fue asociada la imagen. si el campo no se usa lo puede llenar de caracteres nulos terminado por blancos (espacios) o nulo. En cualquier caso el 41° byte debe ser un nulo.

Campo 15. Tiempo de trabajo (6 bytes, variable). Especifica 3 valores que definen un valor entero para el tiempo transcurrido del trabajo cuando la imagen fue guardada. Estos tienen la siguiente estructura:

1 (bytes 0-1)	Horas	0-65535
2 (bytes 2-3)	Minutos	0-59
3 (bytes 4-5)	Segundos	0-59

El propósito de éste campo es permitir a las compañías de producción (y otras) mantener un control total del tiempo invertido en el trabajo de una imagen particular. Este campo es muy útil para tiempo estimado, costos y gastos. Si el campo no es usado, lo debe llenar de ceros.

Campo 16. Identificación del Software (41 bytes, variable). Es un campo ASCII de 41 bytes donde el último byte debe ser un cero (nulo). Dando un total de 40 caracteres ASCII para la identificación del Software. El propósito de este campo es permitir al Software determinar y guardar en que programa fue creada una imagen en particular. Si el campo no es usado, se debe llenar por una serie de nulos o blancos (espacios) terminado por un nulo. El byte 41° siempre debe ser un nulo.

Campo 17. Versión del Software (3 bytes, variable). Consiste de dos subcampos, uno de dos bytes y otro de uno. El propósito de este campo es definir la versión del Software definido en el campo Identificación de Software. Los dos bytes contienen el número de versión como un número entero binario 100 veces. Es decir, si la versión es 4.17 el número entero debe ser 417. Permitiendo las dos posiciones decimales de las subversiones. El tercer byte es de tipo ASCII y contiene la letra de la versión. Así que su estructura es la siguiente:

1 (bytes 467-468)	Numero de versión * 100 byte
2 (byte 469)	Letra de la versión

Si no se usan estos campos los dos primeros bytes deben ser ceros binarios y el tercer byte un espacio (" ").

Campo 18. Color llave (4 bytes, variable). Indica el valor del color llave real cuando la imagen ha sido guardada. El formato es A:R:G:B donde 'A' (byte más significante) es el color llave del canal alfa (si no tiene un canal alfa en su aplicación, ponga este byte en cero).

El color llave puede ser tomado como el color de fondo o "color transparente". Es decir, el color del área donde no hay imagen en pantalla, y el mismo color que la pantalla limpiará si es borrada la aplicación. Si no usa este campo se debe poner en ceros. Poner todo en cero es lo mismo que seleccionar el color negro como color llave.

Un buen ejemplo del color llave es el color transparente usado en TIPS para cargar o guardar una ventana.

Campo 19. Aspect Ratio (Aspecto del Radio del pixel) (4 bytes.variable). Especifica dos subcampos que indican el tamaño del radio. Su estructura es la siguiente:

1 (bytes 0-1)	Numerador del Radio del pixel (ancho del pixel)
2 (bytes 2-3)	Denominador del Radio del pixel (largo del pixel)

Estos valores se usan para determinar el aspect ratio del pixel. Es muy útil cuando es importante preservar el aspect ratio adecuado de la imagen guardada. Si los dos valores son iguales diferentes de cero, entonces la imagen esta compuesta de pixeles cuadrados. Un cero en el segundo subcampo indica que no hay especificación del aspect ratio.

Campo 20. Valor Gama (4 bytes, variable). Especifica dos subcampos que indican el valor gamma. Su estructura es la siguiente:

1. (bytes 0-1) Numerador Gama
2. (bytes 2-3) Denominador Gama

El valor resultante debe estar en el rango de 0.0 a 10.0 con un solo decimal de precisión. Una imagen no corregida (que no tiene valor gama) deberá tener un valor 1.0. Esto se cumple colocando los mismo valores diferentes de cero en ambos posiciones. Si decide ignorar este campo, debe poner el denominador (segundo campo) en cero. Esto indica que el campo de valor gama no esta siendo usado.

Campo 21. Desplazamiento del corrección de color (4 bytes, variable). Especifica un valor de desplazamiento simple. Es un desplazamiento desde el inicio del archivo al inicio de la tabla de corrección de color. Esta tabla puede estar escrita en cualquier parte entre el fin del campo de datos de la imagen (campo 8) y el inicio del Pie de Archivo. Si la imagen no tiene tabla de corrección de color o si el valor de gama es suficiente, se debe poner este valor en cero y no escribir la tabla de corrección.

Campo 22. Desplazamiento franquicia marca (4 bytes, variable). Especifica un valor de desplazamiento simple. Es un desplazamiento desde el inicio del archivo al inicio de la franquicia de marca. La franquicia de marca debe ser escrita después del campo 25 (tabla de líneas escaneadas) y antes del inicio de Pie del archivo. Si no hay franquicia de imagen, este campo debe ser puesto en cero.

Campo 23. Desplazamiento de línea escaneada (4 bytes, variable). Especifica un valor de desplazamiento simple. Es un desplazamiento desde el inicio del archivo al inicio de la tabla de líneas escaneadas.

Campo 24. Tipo de atributos (1 byte, variable). Especifica un valor que indica el tipo de datos del canal alfa que contiene el archivo. Con los siguientes valores:

Valor	Significado
0	Datos alfa no incluidos (los bits 3-0 del campo 5.6 deben ser ceros también)
1	Datos no definidos en el campo alfa, puede ser ignorado
2	Datos no definidos en el campo alfa, pero ser guardados
3	Datos de canal alfa útiles están presentes
4	Alfa premultiplicados (ver descripción abajo)
5-127	Reservado
128-255	No asignados

Alfa premultiplicados. Suponga que los datos del canal alfa están siendo utilizados para especificar la opacidad de cada pixel (para usar cuando la imagen está por capas en otra imagen), donde '0' indica que el pixel es completamente transparente y un valor de '1' indica que el pixel es completamente opaco (asume que todos los valores de los componentes han sido normalizados). Una cuadrupla (a,r,g,b) de (0.5,1,0,0) podría indicar que el pixel es rojo puro con una media transparencia. Por numerosas razones (incluyendo imágenes compuestas) es mejor premultiplicar los componentes de color individual con el valor del canal alfa. Una premultiplicación por encima podría producir una cuadrupla (0.5,0.5,0,0).

Un valor de 3 en el campo tipo de atributos indica que los componentes de color del pixel ya han sido escalados or el valor en canal alfa.

Campo 25. Tabla de líneas escaneadas (variable). Esta información es proporcionada por los siguientes requerimientos:

- Hacer un acceso aleatorio fácil de imágenes compresas
- Permitir acceso de "giant imagen" (imagen gigante) en pequeños pedazos.

Esta tabla puede contener un serie de desplazamientos de 4 bytes. En cada desplazamiento puede escribir puntos al inicio dela siguiente línea escaneada, en el orden que la imagen fue salvada (arriba-abajo, abajo-arriba). el desplazamiento debe iniciar desde el inicio del archivo. Sin embargo podrá tener cuatro valores de byte para cada línea escaneada en su imagen. Esto significa que si su imagen es de 768 pixel de largo, usted tendrá 768, punteros de 4 byte de desplazamiento (para un total de 3072 bytes). Este tamaño no es muy extremo, y por lo tanto esta tabla puede ser construida y mantenida en memoria, y usada a su debido tiempo.

Campo 26. Franquicia de marca de la imagen (variable). El área de franquicia de marca es una pequeña representación de la imagen original. Esto es útil para "browsing" (hojear) una

colección de archivos de imágenes, se recomienda que se cree una utilizando técnicas de submuestreo para crear la mejor representación posible. La franquicia de marca de imagen debe estar almacenada en el mismo formato que la imagen normal especificada en el archivo, pero sin compresión alguna. El primer byte de éste campo especifica el tamaño en X en pixeles, y el segundo byte el tamaño en Y también en pixeles. Truevision no recomienda franquicias más largas de 64 x 64 pixeles, y sugiere que las que sean más largas sean almacenadas en forma recortada. Obviamente el almacenamiento de franquicia ayuda en gran medida en el almacenamiento de la imagen. Dos imágenes almacenadas usando el mismo formato bajo la asunción que si puede leer la imagen entonces puede leer la franquicia de la imagen.

Campo 27. Tabla de corrección de color (2 Kb., variable) La tabla de corrección de color es un bloque de 256 x 4 valores (de 2 bytes cada uno), donde cada puesta de 4 valores contiguos son la corrección de A:R:G:B para esa entrada. Esto permite al usuario almacenar una tabla de corrección para el remapeo de la imagen o manejo de la tabla LUT. Ya que cada color en el bloque es de 2 byte, el valor máximo para un cañón de color es 65536, y el valor mínimo es cero. Por lo tanto, el mapeo para el blanco es 0,0,0,0 y para el blanco es 65535,65535,65535,65535.

Pie del Archivo

1. Un lector TGA debe iniciar por determinar si el archivo es un formato original TGA o un nuevo formato TGA. Esto se cumple examinando loa últimos 26 bytes del archivo. Leyendo los últimos 26 bytes del archivo se recuperan los últimos 26 byte de datos de imagen (si el archivo es un original TGA), o recupera el Pie del archivo (si es un nuevo TGA).
2. Para determinar si el dato adquirido es Pie del Archivo TGA legal, lea los bytes 8 a 23 del Pie como caracteres ASCII y determine si concuerdan con la siguiente rúbrica: TRUEVISION-XFILE

Si esta rúbrica contiene exactamente 16 bytes y es igual a la rúbrica de arriba, se asume que el archivo es un nuevo TGA y puede, por lo tanto, contener área de desarrollo y/o área de extensión. Si no se encuentra esta rúbrica es archivo es un original TGA y sólo contendrá la cabecera y datos de la paleta y: por consiguiente, el formato de byte del Pie de Archivo está definido como sigue:

Bytes 0-3	Desplazamiento de área de extensión
Bytes 4-7	Desplazamiento del directorio de desarrollo
Bytes 8-23	Rúbrica TRUEVISION-XFILE
Byte 24	Caracter ASCII ". " (Código 46)
Byte 25	Cero binario terminador (0x00)

Nota: Para las áreas de desarrollo y extensión no es requerido leer, escribir o utilizarlas, son opcionales. Aun cuando estas áreas no son usadas, se recomienda que el pie de archivo TGA continúe incluido en el archivo.

Campo 28. Desplazamiento área de extensión (bytes 0-3). Los primeros cuatro bytes del Pie de Archivo contienen el desplazamiento desde el inicio del archivo al inicio del área de extensión. Si el desplazamiento es cero, no existe área de extensión en el archivo.

Campo 29. Desplazamiento directorio de desarrollo (bytes 4-7). Los siguientes cuatro bytes contienen el desplazamiento desde el inicio del archivo al inicio del directorio de desarrollo. Si el desplazamiento es cero, no existe directorio de desarrollo.

Campo 30. Rúbrica (bytes 8-23). Tiene exactamente 16 bytes con la siguiente leyenda TRUEVISION -XFILE. Si esta rúbrica es detectada es un archivo nuevo TGA y, por consiguiente, puede tener los campos área de desarrollo y/o área de extensión. En caso contrario es un archivo original TGA.

Campo 31. Reservado (byte 24). Es el caracter ASCII ","(coma). Este caracter debe ser una coma o el archivo no es un auténtico archivo TGA.

Campo 32. Terminador de cadena binaria Cero (byte 25). Es un cero binario que actúa como terminador final y permite que el Pie de Archivo TGA sea leído completamente y utilizado como una cadena.

III.7.3 Tipos de imagen

Imagen Color-Mapped

Estos tipos de datos son usados para almacenar imágenes Color Mapped (imágenes VDA/D, TARGA M8 o Color-Mapped TrueVista). Las imágenes almacenadas en éste formato pueden ser recuperadas y desplegadas rápidamente; sin embargo, una pantalla completa VDA (256 x 200) requiere 511 Kb. de almacenamiento. Este formato es deseable cuando el tiempo de almacenamiento y despliegue son críticos pero el tamaño del archivo no lo es.

Este formato de archivo también proporciona una manera compacta para almacenar imágenes TARGA True Color y Limmited-Color. Los programas pueden fácilmente desplegar imágenes Color Mapped en modos de despliegue en color real por mapeo de pixeles cuando los copia al despliegue de memoria.

Imagen Color Mapped con codificación RLE

Este tipo de datos es usado para almacenar imágenes donde cada pixel puede ser representado por un índice de paleta. La información es codificada, así que, los pixeles sucesivos con el mismo valor de color son codificados con RLE. Este formato está diseñado para usar con computadoras que generan y capturan imágenes que tienen largas secciones contiguas de los mismos colores.

Imagen True Color

Este tipo de datos es usado para almacenar imágenes donde cada pixel es representado por su valor correspondiente de rojo, verde y azul (imágenes ICB, TARGA 16, TARGA 24, TARGA 32 y TrueVista). Este formato es útil donde el tiempo de almacenamiento y despliegue son criticos pero el tamaño del archivo no lo es.

Imagen True Color con codificación RLE

Este tipo de datos es usado para almacenar imágenes por rastreo donde cada pixel es representado por un su componente de rojo, verde y azul. La información es codificada, así que, los pixeles sucesivos con el mismo valor de color son codificados con RLE.

Se usa este formato para almacenar imágenes ICB, TARGA (16, 24 y 32) y TrueVista (modos de 16 y 32 bits RGB). Esta diseñado principalmente para computadoras que generan y capturan imágenes que tienen largas secciones contiguas de los mismos colores

Imagen B/N (sin mapeo)

Este tipo de datos es usado para almacenar imágenes por rastreo donde cada pixel puede ser representado por un valor en escala de grises (imágenes TARGA 8, TARGA M8, y algunos modos TrueVista). Este formato es usado para programas de demostración distribuidos con el TARGA 8.

Imagen B/N con codificación RLE

Este tipo de datos es usado para almacenar imágenes por rastreo donde cada pixel puede ser representado por un nivel de gris. La información es codificada, así que, los pixeles sucesivos con el mismo valor de color son codificados con RLE.

Se usa este formato para almacenar imágenes en B/N TARGA 8 y TARGA M8. Esta diseñado principalmente para computadoras que generan y capturan imágenes que tienen largas secciones contiguas de los mismos colores.

III.7.4 Compresión

Codificación Run Length

Algunos de los tipos de imágenes descritos están almacenados usando el algoritmo de compresión conocido como RLE. Este tipo de codificación es usado es usado con los tipos de imagen 9 (Run Length Encoded Color-Mapped), 10 (Run Length Encoded True-Color) y 11 (Run Length Encoded B/N).

RLE toma ventaja del hecho de que algunos tipos de imágenes tienen largas secciones en donde los valores del pixel son los mismos. Esta situación ocurre más ocasiones en imágenes de vídeo. En esta imágenes donde largas áreas contienen pixeles del mismo valor, RLE puede reducir grandemente el tamaño de la imagen almacenada.

RLE comprime dos tipos de elementos de dato: Run Length Packets y Raw Packets.

El primer campo (1 byte) de cada paquete es llamado el campo de cuenta de repetición. El segundo campo es conocido como campo del valor del pixel. Para RLE packets, el campo valor del pixel contiene un valor simple de pixel. Para Raw packets, el campo es un número variable de valor del pixel.

El Orden de bit más alto de la cuenta de repetición indica si el paquete es RLE packets o Raw packets. Si el bit 7 de la Cuenta de Repetición es un '1', entonces el paquete es RLE packets. Si es '0', entonces es Raw packets.

Los 7 bits más bajos de la Cuenta de Repetición especifica cuantos valores de pixel están representados en el paquete. En el caso de RLE Packets esta cuenta indica cuantos pixeles sucesivos tiene el valor de pixel especificado por el campo Valor del pixel. Para Raw Packets, la Cuenta de Repetición especifica cuantos valores de pixel están contenidos actualmente en el siguiente campo.

Estos valores de los 7 bits están codificados en realidad como 1 menos que el número de pixeles en el paquete (un valor de cero implica 1 pixel mientras que un valor de 07h implica 128 pixeles).

Como un ejemplo de la diferencia entre estos dos tipos, considere una sección simple de línea escaneada con 128 pixeles de 24 bits (3 bytes), todos con el mismo valor (de color). Raw Packets requeriría 1 byte para la Cuenta de Repetición y 128 valores de pixeles cada uno siendo de 3 (bytes) de longitud (384 bytes). Por lo tanto el número total de bytes requeridos sería 385. RLE Packets requiere 1 byte para la Cuenta de Repetición y uno (3 byte para el valor del pixel). Es decir , un total de solo 4 bytes.

Run Length Packets

Run Length Packets está compuesto de dos partes. La primera es una Cuenta de Repetición y la segunda el Valor del pixel a repetir. Ver la tabla III.16.

Tabla III. 16 Run Length Packets.

Nombre del Campo	Tipo de Paquete debe ser 1 para RLE	Cuenta del Pixel (número de pixeles codificados por este paquete menos 1)	Dato del pixel. Valor común del pixel para ser usado por el ancho del pixel
Tamaño del Campo	1 bit	7 bits	Ancho del pixel (campo 5.5)

1. Cuenta del pixel. La Cuenta del pixel toma el rango de 0 a 127 (1-128). Ya que Run Length Packets nunca codifica pixeles en cero, y su orden hace uso de todos los valores disponibles con 7 bits, con un valor de '0' Cuenta del pixel especifica que un pixel está codificado por el paquete. Un valor de '1' especifica que 2 pixeles están en el paquete. En otras palabras, el valor de Cuenta del pixel es siempre uno menos que el número actual de pixeles codificados en el paquete. De este modo, se pueden codificar entre 1 y 128 pixeles con un solo paquete. Si son mayores a 128 bytes, debe codificarlos usando múltiples paquetes.Nota: El bit de alto orden de este subcampo está siempre puesto en '1' para indicar que el tipo de paquete es Run Length.

2. Dato del pixel. Indica un valor de pixel simple para ser repetido. El número de bits en este campo está especificado en el campo 5.5 de la cabecera del archivo.

Si 19 pixeles contiguos en una línea escaneada tienen el valor 0x36 h (tomando 8 bits por pixel) entonces Run Length Packets lo codificaría como sigue:

Tipo de paquete	Cuenta del pixel	Dato del pixel
1	0010010 (0x12 h)	001100110 (0x36 h)

Recuerde que la Cuenta de repetición contiene 1 menos que el número actual de pixeles que están siendo codificados. Ya que estabamos codificando 19 (13h) pixeles, un valor de 18 (18h) fue colocado en la Cuenta de repetición. También, ya que es un Run Length Packets, el bit de más alto orden es puesto en '1'. Esto cambia el byte de Cuenta de repetición de 12h a 92h. El paquete resultante es, por lo tanto, 9236h.

Raw Packets

Raw Packets está compuesto de dos partes. La primera es una Cuenta de Repetición y la segunda el dato del pixel. Ver la tabla III.17.

Tabla III. 7 Raw Packets.

Nombre del Campo	Tipo de Paquete debe ser 0 para Raw Packet	Cuenta del Pixel (número de pixeles codificados por este paquete menos 1	Dato del pixel.
Tamaño del Campo	1 bit	7 bits	Ancho del pixel * Cuenta de pixel + 1

1. Cuenta del pixel. La Cuenta del pixel toma el rango de 0 a 127 (1-128). Ya que Raw Packets nunca codifica pixeles en cero, y su orden hace uso de todos los valores disponibles con 7 bits, con un valor de '0' Cuenta del pixel especifica que un pixel está incluido en el paquete. Un valor de '1' especifica que 2 pixeles están en el paquete. En otras palabras, el valor de Cuenta del pixel es siempre uno menos que el número actual de pixeles incluidos en el paquete. De este modo, se pueden codificar entre 1 y 128 pixeles con un solo paquete. Si son mayores a 128 bytes, debe codificarlos usando múltiples paquetes.

   Nota: El bit de alto orden de este subcampo está siempre puesto en '0' para indicar que el tipo de paquete es Raw Packet.

2. Dato del pixel. Indica un dato de pixel no codificado. Los pixeles serán desplegados en el orden que sean almacenados.

Usemos un algoritmo para representar el método de compresión que utiliza TGA.

Escribir los datos de la cabecera y paleta si la hay
Comenzar a leer desde el principio de la imagen
Repetir
    Leer un dato de la imagen (dato = pixel).
    Contar hasta un máximo de 128 datos posibles que se repitan.
        Si no hay Repetición

{Raw Packets}
Contar cuantos datos consecutivos no se pueden comprimir (no mas de 128)
Escribir el resultado de la operación y a los datos que no se comprimieron
Si no
{RLE Packets}
Escribir el resultado de la (cuenta-1)+128 y el byte que se contó
Fin del si
Hasta leer todos los datos de la imagen

Por ejemplo, hagamos la compresión de los siguientes datos utilizando el algoritmo:

$Pixel_1$	$Pixel_2$	$Pixel_3$	$Pixel_4$	$Pixel_5$	$Pixel_6$	$Pixel_7$	$Pixel_8$	Pixel
254	5	10	10	10	10	10	192	192

Línea de pixeles sin codificar

Escribir los datos de la cabecera y paleta si la hay
Comenzar a leer desde el principio de la imagen
Repetir
Leer un dato de la imagen (dato = pixel) (254).
Contar hasta un máximo de 128 datos posibles que se repitan.
Si no hay Repetición **(no hay repetición)**
{Raw Packets}
Contar cuantos datos consecutivos no se pueden comprimir (no mas de 128) (2 ,**(254,5)**)
Escribir el resultado de la operación y los datos que no se comprimieron **(2 , 254,5)**
Si no
{RLE Packets}
Escribir el resultado de la (cuenta-1)+128 y el byte que se contó
Fin del si
Leer siguiente dato de la imagen (dato = pixel). **(10)**
Contar hasta un máximo de 128 datos posibles que se repitan. (**hay 5 repeticiones**)
Si no hay Repetición
{Raw Packets}
Contar cuantos datos consecutivos no se pueden comprimir (no mas de 128)
Escribir el resultado de la operación y los datos que no se comprimieron
Si no
{RLE Packets}
Escribir el resultado de la (cuenta-1)+128 y el byte que se contó **(132,10)**
Fin del si
Leer siguiente dato de la imagen (dato = pixel). **(192)**

Contar hasta un máximo de 128 datos posibles que se repitan. (**hay 2 repeticiones**)
Si no hay Repetición
{Raw Packets}
Contar cuantos datos consecutivos no se pueden comprimir (no mas de 128)
Escribir el resultado de la operación y los datos que no se comprimieron
Si no
{RLE Packets}
Escribir el resultado de la (cuenta-1)+128 y el byte que se contó (129,192)
Fin del si
Hasta leer todos los datos de la imagen

Hecha la compresión los datos quedan como sigue:

$Pixel_0$	$Pixel_1$	$Pixel_2$	$Pixel_3$	$Pixel_4$	$Pixel_5$	$Pixel_6$	$Pixel_7$	$Pixel_8$
254	5	10	10	10	10	10	192	192

Línea de pixeles sin codificar

Dato0	Dato1	Dato2	Dato3	Dato4	Dato5	Dato6	Dato7
2	254	5	132	10	129	192	...	...

Línea de datos codificados

La descompresión se hace de manera simple, ya que solo es una lectura de paquetes de bytes que se convierten en un corrimiento de bytes. Nuevamente tomemos un algoritmo para ejemplificar el método de descompresión.

Leer los datos de la cabecera y paleta de colores si la hay
Localizamos el inicio de la imagen
Repetir
Leer un dato de la imagen
Si Dato >=128
{RLE packets}
Hacer Cuantos = (dato -128)+1
Leer el siguiente dato
Desplegar este dato tantas veces como tenga "Cuantos"
Si no
{Raw packets}
Hacer Cuantos = dato
Leer tantos datos como veces tenga "Cuantos" y desplegarlos uno a uno
Fin del si
Hasta leer todos los datos de la imagen

Utilizando el algoritmo de descompresión para el ejemplo mencionado:

```
Leer los datos de la cabecera y paleta de colores si la hay
Localizamos el inicio de la imagen
Repetir
  Leer un dato de la imagen  (2)
        Si Dato >=128
              {RLE packets}
          Hacer Cuantos = (dato -128)+1
          Leer el siguiente dato
          Desplegar este dato tantas veces como tenga "Cuantos"
        Si no
              {Raw packets}
          Hacer Cuantos = dato
          Leer tantos datos como veces tenga "Cuantos" y desplegarlos uno a uno (254 ,5)
        Fin del si
Leer siguiente dato  de la imagen  (132)
        Si Dato >=128
              {RLE packets}
          Hacer Cuantos = (dato -128)+1 (132-128+1=5)
          Leer el siguiente dato (10)
          Desplegar este dato tantas veces como tenga "Cuantos" (10,10,10,10,10)
        Si no
              {Raw packets}
          Hacer Cuantos = dato
          Leer tantos datos como veces tenga "Cuantos" y desplegarlos uno a uno
        Fin del si
Leer siguiente dato de la imagen (129)
        Si Dato >=128
              {RLE packets}
          Hacer Cuantos = (dato -128)+1 (129-128+1=2)
          Leer el siguiente dato  (192)
          Desplegar este dato tantas veces como tenga "Cuantos" (192,192)
        Si no
              {Raw packets}
          Hacer Cuantos = dato
          Leer tantos datos como veces tenga "Cuantos" y desplegarlos uno a uno
        Fin del si
Hasta  leer todos los datos de la imagen
```

CAPITULO IV

CAPITULO IV

ANÁLISIS EXPERIMENTAL

IV.1 Acerca de los Formatos de Archivos Gráficos

¿ Dónde encontrar información acerca de los formatos de archivos gráficos ?

La mayor parte de la información referente a este tema existe en Internet, ya que no se cuenta con una amplia bibliografía de información especifica de los formatos.

Sin embargo, esta información la podemos encontrar ya sea en forma descriptiva o experimental; la primera nos proporciona el entorno y teoría del formato de archivo gráfico (generalmente historia, versiones del formato, estructura y método de compresión), la segunda, en este caso es un programa con el cual es posible leer y desplegar la información contenida en el formato de archivo gráfico, la mayoría de las veces la información de ambas no coincide, es decir, se refiere a una misma idea con diferentes conceptos, lo cual ocasiona en mayor o menor grado cierta confusión sobre el manejo de los formatos.

Esta es la razón principal por la cual se decidió incluir los capítulos análisis descriptivo y análisis experimental en este trabajo, para dar seguimiento a la teoría proporcionada en el capítulo III, ya no en teoría si no en la práctica, es decir, programando.

Tener ambos tipos de información es suficiente, pero ¿cuál es el más importante?, puesto que no siempre se encuentran los dos. La teoría es la base para poder comprender fácilmente lo que esta realizando el programa y el porque de ello.

Si solo se cuenta con la información descriptiva, posiblemente no existirá una programación lo suficientemente rápida y eficaz, claro, en concordancia con la teoría explicada y como haya sido entendida.

De otra forma teniendo sólo el programa, depende de la capacidad y habilidad del programador para comprender lo que hace el programa para poder aplicarlo y/ó adaptarlo según su criterio y necesidades.

Generalmente si uno se encuentra en este tipo de situación, recurre al camino más sencillo "si ya esta hecho, ¿para qué volverlo a hacer?", esta frase es conocida por la mayoría de los programadores; en una opinión muy particular, ciertamente no es correcta; implica varios aspectos: el tiempo disponible y los requerimientos del trabajo. En última instancia, si un programador tiene este tipo de pensamiento, es probable que en él no exista la capacidad de desarrollo suficiente; el hecho de volver a realizar algo implica, que posiblemente se mejore y de no ser así, por lo menos obtener un conocimiento nuevo, el cual servirá en proyectos futuros.

En este capitulo se presentan los programas de lectura de los formatos de archivos gráficos desglosados en el capítulo anterior, utilizando la información proporcionada en él. Como los formatos que se estudian ya están creados, analizaremos primero la lectura (descompresión o decodificación de los datos de imagen) de cada uno de ellos. Posteriormente, en el Capitulo V se explica la escritura (compresión o codificación de datos de imagen) y conversión entre ellos. No se pretende que sea un curso de programación, tan solo se aplica en un lenguaje de programación (en este caso C y Pascal) la teoría de cada uno de ellos en lo que apreciamos es una forma comprensible.

IV.2 Estructura de Cada Programa

Un aspecto que es importante resaltar, son las funciones y/ó procedimientos (depende si el lenguaje es Pascal o C, en éste último hay funciones que no regresan valores) más utilizados (as) en estos programas que se le harán familiares al lector, para no perderse entre conceptos, de aqui en adelante nos referiremos a ellas como funciones. Por experiencia propia, la mayoría de los programas no hechos por uno mismo, no son lo suficientemente fiables y causan confusión, o simplemente son muy complejos.

La estructura general de los programas realizados en este trabajo en lenguaje C y Pascal se muestra en la figura IV.1. Los signos XXX representan cada una de las iniciales de los formatos estudiados BMP, PCX, GIF, etc.

Abre_lectura_XXX . Prepara el archivo para su lectura.

Lee_Cabecera_XXX. Lee los datos de la cabecera y los despliega en pantalla.

Lee_Paleta_XXX . Lee y almacena en un arreglo la paleta de colores de la imagen.

Inicializa. Inicializa el modo gráfico con una rutina en ensamblador.

Pon_Paleta_XXX. Activa la paleta de colores leída del archivo y almacenada en el arreglo.

Lee_Imagen_XXX. Realiza el método de descompresión y visualiza en pantalla la imagen.

Abre_lectura_XXX
Lee_Cabecera_XXX
Lee_Paleta _XXX
Pon_Paleta_ _XXX
Lee_imagen _XXX

Figura IV.1 Estructura de cada programa.

Las funciones listadas trabajan en forma independiente excepto Lee_imagen_XXX, ya que necesita los datos que proporcionan las demás funciones. estas funciones serán de utilidad posteriormente en el capitulo V, así que, le recomendamos entender la forma de trabajo de cada una de ellas.

El diagrama de flujo de la estructura general de los programas se muestra en la figura IV.2.

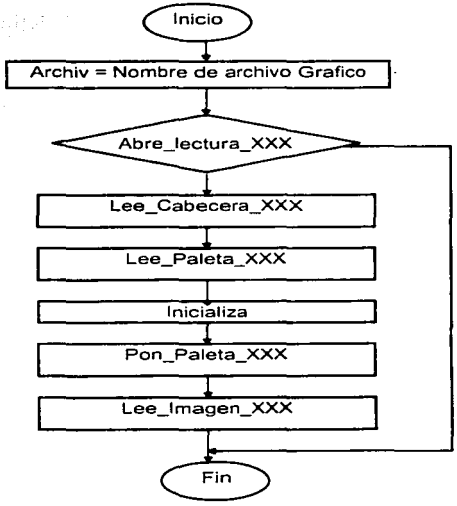


Figura IV.2 Diagrama de flujo.

IV.3 Funciones Comunes

Si se quiere hacer programas que sean compatibles y transportables es necesario hacer funciones que sean comunes, el concepto de cajas negras ejemplifica dichas funciones, una caja negra es un objeto al cual se le pide algo y devuelve un resultado, no interesa como lo hace.

En cada programa existen funciones o procedimientos comunes, estos procedimientos solo hacen lo que se les pide y están adecuados para que cualquier programa las pueda tomar, al conjunto de este tipo de funciones y/o procedimientos se les llama librerías, pueden estar juntas en un mismo archivo.

Las funciones comunes serán explicadas solo una vez y en cada parte correspondiente de los programas, se incluirá el comentario correspondiente.

Las funciones comunes que se utilizan son:

Inicializa (Driver, Modo). Función que inicializa el modo gráfico, utiliza el driver "SVGA256.BGI" que proporciona BORLAND. Driver y Modo serán especificados según el equipo con que se cuente, o resolución que se desee. Ver Tabla II.2.

En lenguaje Pascal, es una función con el siguiente código, la letra curva entre llaves indica que es un comentario:

```
Function Inicializa(Driver,Modo:integer):boolean;        {Inicializa gráficos y reporta los errores que ocurran}
var
 ErrorG:integer;                                          {Variable usada para detectar el código de error}
        x,y:word;          {Variables Utilizadas para detectar la máxima resolución del modo gráfico elegido}
Begin
if (driver=16) Then                                  {Si utiliza SVGA256.BGI debe enviar el Driver con valor 16}
  driver := InstallUserDriver('Svga256',nil);                    {Da de alta el driver Svga256.BGI, devuelve}
                                                              {un código de error si no puede realizar la alta}
if driver=grError Then  halt(1);                                   {Corta el Programa si hubo algún error}
InitGraph(Driver, Modo ,'d:\bp\programa');                                   {Inicializa el modo gráfico}
asm                                                   {El siguiente bloque es una rutina en ensamblador que}
   mov ax,000fh                                      {activa el tamaño del área del Ratón en resoluciones altas}
   mov cx,8
   mov dx,8
   int 33h
end;
x:=getmaxx;                                                   {Toma el máximo valor de X en la pantalla gráfica}
y:=getmaxy;                                                   {Toma el máximo valor de Y en la pantalla gráfica}
asm                                                      {El siguiente bloque es iuna rutina en ensamblador que}
   mov ax,x                                        {activa con "x" y "y" el área de pantalla en resoluciones altas.}
   mov dx,ax                                     {para que el ratón pueda utilizar toda la pantalla en sus movimientos}
   mov ax,0007h
   mov cx,0h
   int 33h
```

```
    mov ax,y
    mov dx,ax
    mov ax,0008h
    mov cx,0h
    int 33h
  end;
 ErrorG:=graphResult;                                  {Almacena el error ocurrido en una variable}
 If ErrorG<>grok Then                                  {Verifica si ha ocuurido algún error}
  Begin                                                {Bloque de código si ha ocurrido algún error}
   Writeln('Error en graficos : ',GraphErrormsg(ErrorG));
   Inicializa:=False;                                  {Devuelve un valor de falso para la función inicializa}
   Restorecrtmode;                                     {Regresa al modo gráfico anterior}
  End
 Else
  Begin                                                {Bloque de código si no ha ocuurido algún error}
   cleardevice;
   Inicializa:=True;                                   {Devuelve un valor de verdadero para la función}
  End;
 End;
```

En resoluciones altas, que manejan arriba de 256 colores, no es posible contar con todas las ventajas del ratón: no se dispone del área total de la pantalla, no existe un puntero que indique en que coordenada se encuentra el ratón, no se conoce el área que dispone el puntero. Las rutinas hechas en ensamblador proveen estas características excepto, la de visualización del puntero, este problema lo debe resolver el programador por sí solo (se puede hacer utilizando una pequeña máscara sobre la pantalla). Sin embargo, estas rutinas están de más para los programas presentados, ya que no es necesario utilizar el ratón, el ratón solo es utilizado en el programa ejecutable que acompaña a este trabajo (fuentes y librerías que se proporcionan).

En lenguaje C, la función Inicializa devuelve un valor entero, contiene el siguiente código, la letra curva entre los signos // indica que es un comentario:

```
int Inicializa(int Driver, int Modo)                  // Inicializa Gráficos y reporta los errores que ocurran
{
 int ErrorG;                                          // Variable utilizada para almacenar el código de error
 unsigned int x,y;                                    // Variables utilizadas para detectar la máxima resolución
                                                      // del modo gráfico elegido
 union REGS regin,regout;                             // Variables de registro para activar el área del ratón
 if (Driver==16)                                      // Si utiliza SVGA256.BGI debe enviar el Driver con valor 16
 Driver = installuserdriver("Svga256",NULL);          // Da de alta el driver Svga256.BGI, devuelve
                                                      // un código de error si no puede realizar la alta
 if (Driver==grError)                                 // Corta el programa si hubo algún erroral dar de alta el driver
     exit;
initgraph( &Driver, &Modo,"d:\borlandc\programa\exeobj");   // Inicializa el modo gráfico
regin.x.ax=0x000f;                                    // El siguiente bloque es una rutina que activa el tamaño
regin.x.cx=0x8;                                       // del área del ratón en resoluciones altas
regin.x.dx=0x8;
int86(0x33,&regin,&regout);
```

```
 x=getmaxx();                                   // Toma el máximo valor de X en la pantalla gráfica
 y=getmaxy();                                   // Toma el máximo valor de Y en la pantalla gráfica
 regin.x.ax=x;                                  // El siguiente bloque es una rutina que activa con "x" y
 regin.x.dx=regin.x.ax;                         // "y" el área de pantalla en resoluciones altas, para que el ratón
 regin.x.ax=0x0007;                             // pueda utilizar toda la pantalla en sus movimientos
 regin.x.cx=0x0;
 int86(0x33,&regin,&regout);
 regin.x.ax=y;
 regin.x.dx=regin.x.ax;
 regin.x.ax=0x0008;
 regin.x.cx=0x0;
 int86(0x33,&regin,&regout);
 ErrorG=graphresult();                          // Almacena el error ocurrido en una variable
 if (ErrorG!=grOk)                              // Verifica si ha ocuurido algún error
 {                                              // Bloque de código si ha ocurrido algún error
   printf("\n Error en graficos : %s",grapherrormsg(ErrorG));
   restorecrtmode;                              // Regresa al modo gráfico anterior
   return 0;                                    // Devuelve un valor de ´0´ para la función inicializa
 }
 else                                           // Bloque de código si no ha ocuurido algún error
 {
   cleardevice;
   return 1;                                    // Devuelve un valor de ´1´ para la función inicializa
 }
}
```

Cambia_Color (Color, rojo, verde, azul). Función que modifica la paleta de colores activa en el S.O.. Los parámetros Color, rojo, verde y azul indican el número de color que se desea modificar, y los componentes en rojo verde y azul respectivamente. Utiliza la dirección del puerto donde se encuentran los datos de la paleta de colores (3C8h), localiza el color y modifica sus componentes en rojo, verde y azul.

En lenguaje Pascal es un procedimiento con el siguiente código.

```
Procedure Cambia_color(color,r,v,a:Byte)
Begin                    {Procedimiento que cambia un color de la paleta activa del sistema operativo}
asm
  mov dx,3C8h                                   {Localiza el puerto}
  mov al,color                                  {Localiza número de color}
  out dx,al
  inc dx
  mov al,r                                      {Modifica el componente en rojo}
  out dx,al
  mov al,v                                      {Modifica el componente en verde}
  out dx,al
  mov al,a                                      {Modifica el componente en azul}
  out dx,al
 End;
End;
```

En lenguaje C es una función que no regresa valor alguno, contiene el siguiente código.

```
                                                    //BYTE = #define BYTE unsigned char
void Cambia_color(BYTE color,BYTE r,BYTE v,BYTE a)
                    // Procedimiento que cambia un color de la paleta activa del sistema operativo
 asm mov dx,3C8h                                                    // Localiza el puerto
 asm mov al,color                                                   // Localiza # de color
 asm out dx,al
 asm inc dx
 asm mov al,r                                                       // Modifica el componente en rojo
 asm out dx,al
 asm mov al,v                                                       // Modifica el componente en verde
 asm out dx,al
 asm mov al,a                                                       // Modifica el componente en azul
 asm out dx,al
}
```

Abre_lectura_XXX (Archivo_imagen). Función que abre el archivo de la imagen para verificar si es posible leerlo y que el programa no aborte. El parámetro Archivo_imagen especifica el nombre del archivo a leer. XXX indica las siglas del archivo correspondiente: BMP, PCX,....,etc.

En lenguaje Pascal es un procedimiento con el siguiente código.

```
Function Abre_lectura_XXX(Arch_imagen:String):Boolean;          {Prepara al archivo para su lectura}
Begin
 Assign(Lee_Arch,arch_imagen);                                  Enlaza Lee_Arch con el archivo arch_imagen}
  {Sy-}                                                          Desactiva errores de entrada y salida}
  reset(Lee_arch,1);                                   Prepara Lee_Arch para ser leído, el numero 1 indica que se}
                                                                {abrira para leer registros con 1 byte de longitud}
   {Sy+}                                                          {Activa errores de entrada y salida}
if ioresult = 0 Then                                          {Verifica si hay errores, ioresult es cero si no hay}
   Abre_lectura_xxx:=True;                                 {Devuelve un valor de verdadero para Abre_lecutra_xxx}
 else
  Begin
   Writeln('Error de apertura de archivo');
   Abre_lectura_xxx:=False;                                {Devuelve un valor de falso para Abre_lecutra_xxx}
  End;
End;
```

En lenguaje C es una función que regresa un valor entero, con el siguiente código.

```
int Abre_lectura_bmp(char *arch_imagen)                          // Prepara al archivo para su lectura
{
lee_arch=fopen(arch_imagen,"rb");                                          // Abre y lee el archivo
 if (lee_arch==NULL)                                                   // Si encuentra un valor nulo
  {return 0;}                                                                   // Devuelve un '0'
 else                                                          // Si encuentra un caracter diferente al nulo
{return 1;}                                                                     // Devuelve un '1'
}
```

IV.4 Formato BMP

El formato BMP es uno de los formatos de archivos gráficos más sencillos de programar. Debe contener la estructura del formato, se utilizan 5 estructuras para manipular su información, las cuales se describen a continuación:

Cabeza_BMP. Estructura de la cabecera del archivo BMP, es la cabecera del archivo.

Informe_bmp. Especifica la estructura de la cabecera bitmap para archivos de Windows 3.x.

Centro_bmp. Especifica la estructura de la cabecera bitmap para archivos de Windows v2.0 o la versión para OS/2.

Paleta_bmpi. Especifican el tipo de paleta que se leerá para la versión 2.

Paleta_bmpc. Especifican el tipo de paleta que se leerá para la versión 3.

Los datos de la cabecera serán utilizados de acuerdo al tipo de BMP que se este leyendo, este programa soporta las versiones 2.0 y 3.0 de archivos BMP de Microsoft Windows y la versión 1.x de OS/2 de IBM con 256 colores. Visualiza imágenes con tipo de compresión 0 (sin codificar) y 1 (RLE de 8 bits).

Los programas de cada uno de los formatos analizados tienen un estándar de visualización de imágen con 256 colores, 8 bits por pixel, excepto TGA como se podrá apreciar más adelante. Razón por la cual en el formato BMP no se descomprimen datos codificados con RLE4 ni con imagenes de 16 millones de colores.

IV.4.1 En Lenguaje Pascal

En Pascal el programa se codifica de la siguiente manera.

```
Program Lee_bmp;          {Programa que lee el formato de archivo BMP
                           Realizado para el trabajo de tesis titulado:FORMATOS GRÁFICOS
                           Programadores :
                                        Cantero Ramírez Carlos
                                        Trejo Bonaga Marilu
                          Asesores :
                                        Manzo Salazar Itsmael
                                       Monterrosa Escobar Amilcar A. }
Uses crt,dos,graph;
type                                                    {Definición de las 5 estructuras}
Cabeza_BMP=record                                       {Cabecera del archivo}
  tipo: word;                                           {Tipo de archivo letras " BM ", identificador}
  tamano: longint;                                      {Tamaño del archivo en bytes}
  reservado1,reservado2: word;                          {Reservados  con valor cero}
  tam_cabeza: longint;                                  {Tamaño cabecera, incluye paleta ;bytes de desplazamiento}
```

```
  tipo_bmp: longint;                    {Determina el tipo de BMP 40 ó 12, tamaño de la cabecera del bitmap}
end;

Informe_bmp=record                                                  {Cabecera del bitmap  para Windows v3.0}
  xmaximo: longint;                                                              {X máximo de la imagen}
  ymaximo:longint;                                                               {Y máximo de la imagen}
  planos: word;                                                                        {Planos por pixel}
  contador: word;                                                                {Contador, bits por pixel}
  tipo_compresion: longint;                                                        {Tipo de compresión}
  tamano_imagen: longint;                                                        {Tamaño de la imagen}
  bitxmetrox: longint;                                                               {Bits en x por metro}
  bitxmetroy: longint;                                                               {Bits en y por metro}
  biclrusado: longint;                                                                {Colores usados}
  biclrimport: longint;                                                            {Colores Importantes}
end;

Centro_bmp=Record                                          {Cabecera del bitmap de Windows v2.0 u OS/2 v1.x}
  xmaximo:word;                                                                  {Y máximo de la imagen}
  ymaximo: word;                                                                 {X máximo de la imagen}
  planos:word;                                                                         {Planos por pixel}
  contador:word;                                                                                {Contador}
End;

Paleta_bmpi=record                                         {Estructura de paleta de colores para Windows v3.0}
  azul,verde,rojo,reservado: byte;                                                          {En este orden}
end;
Paleta_bmpc=record                                         {Estructura de paleta de colores para Windows v2.0}
  azul,verde,rojo: byte;                                                                    {En este orden}
end;

const
        bi_rgb=0;                                          {Constantes para cada tipo de compresión}
        bi_rle8=1;
        bi_rle4=2;

Var
 k,i,j,leyo:integer;                                                                  {Variables auxiliares}
 archiv:string;                                                          {Variable para el nombre del archivo a leer}
 Lee_Arch:File;                                                                          {Archivo de lectura}
 bmp:Cabeza_bmp;                                                {Variable para la lectura de datos cabecera del archivo}
 bmpi:informe_bmp;                                              {Variable para la lectura de datos cabecera bitmap v3.0}
 bmpc:Centro_bmp;                                               {Variable para la lectura de datos cabecera bitmap v2.0}
 bmp_pali:Array [0..255] of paleta_bmpi;                                   {Variable para almacenar paleta v3.0}
 bmp_palc:Array [0..255] of paleta_bmpc;                                   {Variable para almacenar paleta v2.0}

Function Inicializa(Driver,Modo:integer):boolean;                {Inicializa modo gráfico y reportar los errores}
var                                                                                              {que ocurran}
 ErrorG:integer;                                                        {Esta función no necesita explicación alguna}
 x,y:word;
Begin
 if (Driver=16) Then
  Driver := InstallUserDriver('Svga256',nil);
 if Driver=grError Then
  Begin
```

```
    halt(1);
   End;
  InitGraph(Driver, Modo,'d:\bp\programa');
    asm
      mov ax,000fh
      mov cx,8
      mov dx,8
      int 33h
    end;
     x:=getmaxx; y:=getmaxy;
    asm
      mov ax,x
      mov dx,ax
      mov ax,0007h
      mov cx,0h
      int 33h
      mov ax,y
      mov dx,ax
      mov ax,0008h
      mov cx,0h
      int 33h
    end;
  ErrorG:=graphResult;
  If ErrorG<>grok Then
   Begin
    Writeln('Error en graficos : ',GraphErrormsg(ErrorG));
    Inicializa:=False;
    Restorecrtmode;
   End
  Else
   Begin
    cleardevice;
    Inicializa:=True;
   End;
 End;


 Procedure Cambia_color(color,r,v,a:Byte);          {Modifica la paleta de colores activa en el S.O}
 Begin                                              {Esta función no necesita explicación alguna}
  asm
   mov dx,3C8h
   mov al,color
   out dx,al
   inc dx
   mov al,r
   out dx,al
   mov al,v
   out dx,al
   mov al,a
   out dx,al
  End;
 End;
```

```
Function Abre_lectura_bmp(Arch_imagen:String):Boolean;          {Prepara al archivo para su lectura}
Begin                                                            {Esta función no necesita explicación}
 Assign(Lee_Arch,arch_imagen);
   {Sy-}
  reset(Lee_arch,1);
   {Sy+}
 if ioresult = 0 Then Abre_lectura_bmp:=True
 else
   Begin
    Writeln("Error de apertura de archivo");
    Abre_lectura_bmp:=False;
   end;
End;


Procedure Lee_Cabecera_BMP;                  {Lee los datos de la cabecera y los despliega en pantalla}
Var
 leyo:Integer;                                                                          {Variable auxiliar}
 posicion:LongInt;                                                                      {Variable auxiliar}
Begin
 blockread(Lee_Arch,bmp,sizeof(bmp),leyo);                          {Lee los datos del archivo y los almacena}
 posicion:=filepos(Lee_arch);                                  {Asigna la posición actual del puntero del archivo}
 With bmp do                                                {El siguiente bloque despliega los datos almacenados}
   Begin
    Gotoxy(01,01);Write('Cabecera del archivo BMP de MicroSoft Windows ',archiv);
    Gotoxy(01,02);Write('Tipo de bmp          :');
    Gotoxy(25,02);Writeln(chr(tipo),chr(tipo shr 8),'     ','Posición del puntero :',posicion);
    Gotoxy(01,03);Write('Tamano del arch      :');
    Gotoxy(25,03);Writeln(tamano);
    Gotoxy(01,04);Write('tam_cabeza           :');
    Gotoxy(25,04);Writeln(tam_cabeza,'  ',reservado1,'  ',reservado2);
    Gotoxy(01,05);Write('Tipo de bmp          :');Gotoxy(25,05);Writeln(tipo_bmp);
  End;
 If (bmp.tipo_bmp=40) Then                                                          {Si tipo es 40, Windows v3.0}
  Begin
    blockread(Lee_Arch,bmpi,sizeof(bmpi),leyo);                        {Lee los datos del archivo y los almacena}
    With bmpi do                                                {El siguiente bloque despliega los datos almcenados}
      Begin
       Gotoxy(01,06);Write('Xmaximo          :');Gotoxy(25,06);Writeln(xmaximo);
       Gotoxy(01,07);Write('Ymaximo          :');Gotoxy(25,07);Writeln(ymaximo);
       Gotoxy(01,08);Write('planos           :');Gotoxy(25,08);Writeln(planos);
       Gotoxy(01,09);Write('bits por pixel   :');Gotoxy(25,09);Writeln(contador);
       Gotoxy(01,10);Write('tipo de compresion  :');Gotoxy(25,10);Writeln(tipo_compresion);
       Gotoxy(01,11);Write('tamaño de imagen    :');Gotoxy(25,11);Writeln(tamano_imagen);
       Gotoxy(01,12);Write('bi x             :');Gotoxy(25,12);Writeln(bitxmetrox);
       Gotoxy(01,13);Write('bi y             :');Gotoxy(25,13);Writeln(bitxmetroy);
       Gotoxy(01,14);Write('bi clr used      :');Gotoxy(25,14);Writeln(biclrusado);
       Gotoxy(01,15);Write('bi clr import    :');Gotoxy(25,15);Writeln(biclrimport);
      End;
 End
Else                                                                          {Es  12, Windows v2.0 y Os/2 v1.0}
  Begin
    blockread(Lee_Arch,bmpc,sizeof(bmpc),leyo);                        {Lee los datos del archivo y los almacena}
    With bmpc do                                                {El siguiente bloque despliega los datos almcenados}
```

```
      Begin
        Gotoxy(01,06);Write('Xmaximo        :');Gotoxy(25,06);Writeln(xmaximo);
        Gotoxy(01,07);Write('Ymaximo        :');Gotoxy(25,07);Writeln(ymaximo);
        Gotoxy(01,08);Write('planos         :');Gotoxy(25,08);Writeln(planos);
        Gotoxy(01,09);Write('bits por pixel :');Gotoxy(25,09);Writeln(contador);
      End;
    End;
  End;


Procedure Lee_Paleta_BMP;                          {Lee la paleta de colores y la almacena en un arreglo}
Begin
 If (bmp.tipo_bmp=40) Then                                                    {Si es Windows v3.0}
 blockread(Lee_Arch,bmp_pali,sizeof(bmp_pali),leyo)                          {Lee y almacena la paleta}
  Else                                                                   {Es 12 Windows v2.0 y OS/2 v1.0}
   blockread(Lee_Arch,bmp_palc,sizeof(bmp_palc),leyo);                        {Lee y almacena la paleta}
End;


Procedure Pon_Paleta_BMP;                                          {Activa  la paleta de colores del archivo}
Var
 i,y,k:integer;
Begin
 If (bmp.tipo_bmp=40) Then                                                   {Si es paleta de Windows v3.0}
   For i:=0 to 255 do
     with bmp_pali[i] do
       Cambia_color(i,rojo shr 2,verde shr 2,azul shr 2);          {Cambia la paleta, ver nota al final del código}
 Else                                                                        {Es 12, Windows v2.0 y OS/2 v1.0}
   For i:=0 to 255 do
     with bmp_palc[i] do
       Cambia_color(i,rojo shr 2,verde shr 2,azul shr 2);                                   {Cambia la paleta}
 End;


Procedure Lee_imagen_BMP;                       {Descomprime los datos de imagen y los pone en pantalla}
Var
 ponx,pony,numcols:integer;                   {Variables para el almacenamiento de datos del pixel y auxiliar}
 i,j,k,cont,tam_imagen:Longint;                                                          {Variables auxiliares}
 unbyte,cuantos:byte;                                                   {Variables para los dos bytes de código}
 Salte:boolean;                                                                   {Bandera para continuar decodificando}
Begin                                                          {Este programa solo trabaja con  256colores; se puede sustituir}
 numcols:=256;                                                                   {por número de colores= 2^contador1 = 2^8=256}
 seek(lee_arch,bmp.tam_cabeza);                          {Posiciona el puntero del archivo al final de la cabecera " donde}
                                                                                   {comienzan los datos de imagen "}

 If (bmp.tipo_bmp=40) Then                                                                  {Para  Windows v3.0}
  Begin
    if bmpi.tipo_compresion=0 Then
       bmpi.xmaximo:=Trunc((bmp.tamano - bmp.tam_cabeza)/bmpi.ymaximo);
                          {Si  tipo  de  compresión  es  '0',  sin comprimir  se realiza  una "confirmación" del
                             {valor bmpi.xmaximo ya que en las pruebas realizadas pudo observarse que en
                              {este tipo de compresión solía suceder que  bmp.tamano  - bmp.tam_cabeza =
                                  {tam_imagen  era  diferente  a  bmp.xmaximo  *  bmpi.ymaximo,  siendo
                                                                                  {bmpi.xmaximo el dato erróneo. }
```

```
tam_imagen:=((bmpi.ymaximo)*bmpi.xmaximo);                    {Calcula el tamaño total de la imagen}
Case bmpi.tipo_compresion of                                  {Para cada tipo de compresión}

 0:                                                           {Datos de imagen sin comprimir}
  Begin
   for i:=0 to (tam_imagen-1) do                              {Para todos los datos de la imagen}
    begin
      if numcols=256 then                                     {Si número de colores  es 256}
        begin
          blockread(Lee_Arch,unbyte,1,leyo);                  {Lee y almacena un byte}
        end
      else                                                    {Número de colores  no es 256}
        if numcols=16 then                                    {Si número de colores  es 16}
          begin                                               {Insertar el proceso para 16 colores}
          end
        else
          if numcols=2 then                                   {Si número de colores es 2}
            begin                                             {Insertar el proceso para 2 colores}
            end;
      ponx:=(i mod (bmpi.xmaximo));                           {Calcula la posición X del pixel [1]}
      pony:=(i div (bmpi.xmaximo));                           {Calcula la posición Y del pixel}
      putpixel(ponx,bmpi.ymaximo-pony,Unbyte);                {Pone en pantalla el dato del pixel [2]}
    End;
  End;

 1:                                                           {Compresión de RLE8}
   Begin
    i:=0;j:=0;cont:=0;k:=0;salte:=false;                      {Inicialización de variables}
    Repeat                                                    {Repite el proceso hasta encontrar el fin de archivo}
  blockread(Lee_Arch,unbyte,sizeof(unbyte),leyo);             {Lectura del primer byte de código}
  blockread(Lee_Arch,cuantos,sizeof(cuantos),leyo);           {Lectura del segundo byte de código}
      If (unbyte=0) Then                                      {Marcadores del RLE}
       Begin
        case cuantos of                                       {Bloque de opciones para el segundo byte}

        0:                                                    {Es un código de Fin de linea}
         Begin
           inc(j);                                            {Incrementa la cuenta de lineas}
           i:=0;                                              {Inicializa la cuenta de datos por linea}
         End;

        1:                                                    {Es un código de fin de archivo}
         Begin
           salte:=True;                                       {Termina la decodificación}
         End;

        2:                                                    {Es un código Delta, no soportado}
         Begin
         End;
```

[1]Recuerde que los datos del archivo así como la memoria es lineal y debemos "cortar líneas" de acuerdo a la resolución de la pantalla.
[2]Se resta pony a bmp.ymáximo debido a que en bmp los datos de la imagen están almacenados de la esquina inferior izquierda a la esquina superior derecha.

```
          Else                                                    {Si no es algún marcador son datos sin codificar}
            Begin
              for k:=1 to cuantos do                              {Imprime tantos datos diferentes como sea el valor}
                Begin                                                                                   {de cuantos}
                  inc(i);                                         {Incrementa i, cuenta de datos decodificados por linea}
                  inc(cont);                                      {Incrementa cont, cuenta de datos decodificados}
                  blockread(Lee_Arch,unbyte,sizeof(unbyte),leyo);  {Lee el siguiente dato diferente}
                  ponx:=(cont mod (bmpi.xmaximo));                 {Calcula la posición X del pixel}
                  pony:=(cont div (bmpi.xmaximo));                 {Calcula la posicion Y del pixel}
                  putpixel(ponx,bmpi.ymaximo-pony,unbyte);         {Pone en pantalla el dato del pixel}
                End;
              If cuantos mod 2 <> 0 Then                          {Si el número de bytes sin codificadar es impar}
                blockread(Lee_Arch,unbyte,sizeof(unbyte),leyo);   {Lee otro byte para completar el par}
            End;
          End;                                                                                          {fin del case}
          End                                                     {fin del primer if del caso 1:compresión RLE8}
        Else                                                      {Si unbyte es diferente de '0', datos iguales codificados}
          Begin
            for k:=1 to unbyte do                                 {Imprime tantos datos como sea el valor de unbyte}
              Begin
                inc(i);                                           {Incrementa i, cuenta de datos decodificados por linea}
                inc(cont);                                        {Incrementa cont, cuenta de datos decodificados}
                ponx:=(cont mod (bmpi.xmaximo));                  {Calcula la posición X del pixel}
                pony:=(cont div (bmpi.xmaximo));                  {Calcula la posición Y del pixel}
                putpixel(ponx,bmpi.ymaximo-pony,cuantos);         {Pone en pantalla el dato del pixel}
              End;
          End;
      Until (Salte);
    End;

  2:                                                                                                    {RLE de 4 bits}
    Begin                                                         {Insertar el código para la decodificación de RLE4}
    End;
  End;                                                                                                  {Fin del primer case}
End                                                                                                     {Fin del primer if}

Else                                                                                                    {Es Windows v2.0, OS/2  v1.0}
  Begin
    bmpc.xmaximo:=Trunc((bmp.tamano-bmp.tam_cabeza)/bmpc.ymaximo);                                     {Para confirmar}
    tam_imagen:=bmpc.ymaximo;
    tam_imagen:=(tam_imagen*bmpc.xmaximo);                        {Calcula el  tamaño de la imagen}
    For i:=0 to (tam_imagen-1) do                                 {Para cada dato de la imagen}
      Begin
        if numcols=256 then                                       {Si número de colores  es 256}
          begin
            blockread(Lee_Arch,unbyte,1,leyo);                                                          {Lee un byte}
          end
        else                                                      {Número de colores  no es 256}
          if numcols=16 then                                      {Si número de colores  es 16}
            begin
            end
          else
            if numcols=2 then                                     {Si número de colores  es 2}
```

```
            begin
            end;
        ponx:=(i mod (bmpc.xmaximo));                    {Calcula la posición X del pixel}
        pony:=(i div (bmpc.xmaximo));                    {Calcula la posición Y del pixel}
        putpixel(ponx,bmpc.ymaximo-pony,Unbyte);         {Pone en pantalla el dato del pixel}
      end;
  End;
End;


Begin                                                    {Cuerpo del bloque principal del programa}
  clrscr;                                                {Limpia la pantalla}
  Archiv:='os2croc.bmp';                                 {Asigna a una variable el nombre del archivo a leer}
  If Abre_lectura_bmp(Archiv) Then                       {Si logra abrir el archivo continua con el programa}
    Begin
      Lee_Cabecera_BMP;                                  {Lee los datos de la cabecera y los coloca en pantalla}
      Readkey;                                           {Pausa para tomar nota de la cabecera}
      Lee_Paleta_BMP;                                    {Lee los datos de la paleta}
      Inicializa(16,4);                                  {Inicializa el modo gráfico, 1024x768x256}
      Pon_Paleta_BMP;                                    {Activa la paleta leida del archivo}
      Lee_imagen_BMP;                                    {Descomprime la imagen y la visualiza}
      Readkey;                                           {Pausa para ver la imagen}
      close(Lee_arch);                                   {Cierra el archivo}
      restorecrtmode;                                    {Recupera el modo texto}
      closegraph;                                        {Cierra la libreria gráficos}
      clrscr;                                            {Limpia pantalla}
    End;
End.
```

Nota: El procedimiento pon_paleta utilliza la función shr (en lenguaje C es >>), esta función sirve para utillizar los 63 tonos que se encuentran en cada byte de color Rojo Verde y Azul, simplemente hace un desplazamiento de bits hacia la derecha, por ejemplo, si el byte leido es 255 en el color rojo, desplazando hacia la derecha 2 bits el nuevo valor es 63, que en este caso es el máximo tono del color rojo.

Una de las imágenes procesadas con el programa es: os2croc.bmp, la cual es una imagen de tipo OS/2.

Os2croc.bmp

El programa da como resultado los siguientes datos:

```
Cabecera del archivo BMP de MicroSoft Windows os2croc.bmp
Tipo de bmp          :   BM      Posición del puntero :18
Tamaño del arch      :   307994
tam_cabeza           :   794  ,0  ,0
Tipo de bmp          :   12
xmaximo            :     640
ymaximo              :   480
planos               :   1
bits por pixel       :   8
-
```

Posteriormente se despliega la imagen.

IV.4.2 En Lenguaje C

El código en C del programa se muestra a continuación.

```
                    // Programa que lee el formato de archivo BMP
                     Realizado para el trabajo de tesis titulado:FORMATOS GRÁFICOS
                    Programadores :
                                Cantero Ramírez Carlos
                                Trejo Bonaga Marilu
                    Asesores :
                                Manzo Salazar Itsmael
                                Monterrosa Escobar Amilcar A.
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>
#include <dos.h>
#include <conio.h>
#define BYTE unsigned char          Definición de constantes simbólicas
#define True  1
#define False 0
#define LONGINT unsigned long
#define WORD    unsigned int
                                                        // Definición de las 5 estructuras
typedef struct{
   char  tipo[2];                                       // Tipo de Archivo letras " BM "
   LONGINT tamano;                                      // Tamaño del Archivo en bytes
   WORD    reservado1,reservado2;                       // Reservados  con valor cero
   LONGINT tam_cabeza;                                  // Desplazamiento de la cabeza del bitmap
   LONGINT tipo_bmp;                                    // Tipo de BMP 40 ó 12, tamaño  cabecera del bitmap
} Cabeza_BMP;                                           // Cabecera del archivo

typedef struct {
   LONGINT xmaximo;                                     // Determina X máximo de la imagen
   LONGINT ymaximo;                                     // Determina Y máximo de la imagen
   WORD planos;                                         // Planos por pixel
   WORD contador;                                       // Contador, bits por pixel
   LONGINT tipo_compresion,                             // Tipo de compresion
           tamano_imagen,                               // Tamaño de la imagen
           bitxmetrox,                                  // Bits en x por metro
           bitxmetroy,                                  // Bits en y por metro
           biclrusado,                                  // Colores usados
           biclrimport;                                 // Colores Importantes
}Informe_bmp;                                           // Cabecera del bitmap  para Windows v3.0

typedef struct{
 WORD xmaximo,                                          // Determina X máximo de la imagen
      ymaximo,                                          // Determina Y maximo de la imagen
        planos,                                         // Planos por pixel
        contador;                                       // Contador
}Centro_bmp;                                            // Cabecera del bitmap  para Windows v2.0 u OS2 v1.0

typedef struct {
   BYTE azul,verde,rojo,reservado;                      // En este orden
}Paleta_bmpi;                                           Estructura de paleta de colores para Windows v3.0
```

```
typedef struct {
   BYTE azul,verde,rojo;                                   // En este orden
}Paleta_bmpc;                                              // Estructura de paleta de colores para Windows v2.0

#define bi_rgb  0                                          // Constantes para cada tipo de compresión
#define bi_rle8 1
#define bi_rle4 2

                                                           // Bloque de declaraciones de las variables
int leyo;                                                  // Variable auxiliar
char *archiv;                                              // Variable para el nombre del archivo a leer
Cabeza_BMP bmp;                                            // Variable para la lectura de datos cabecera del archivo
Informe_bmp bmpi;                                          // Variable para la lectura de la cabecera bitmap v3.0
Centro_bmp bmpc;                                           // Variable para la lectura de la cabecera bitmap v2.0
Paleta_bmpi bmp_pali[256];                                 // Variable para almacenar paleta v3.0
Paleta_bmpc bmp_palc[256];                                 // Variable para almacenar paleta v2.0
FILE *lee_arch;                                            // Archivo de lectura
                                                           // Bloque de declaración de las funciones utilizadas
void Lee_Cabecera_BMP(void);
int Inicializa(int Driver, int Modo);
int Abre_lectura_bmp(char *arch_imagen);
void Lee_Paleta_BMP(void);
void Pon_Paleta_BMP();
void Lee_imagen_BMP();
void Cambia_color(BYTE color,BYTE r,BYTE v,BYTE a);

void main ()                                               // Cuerpo del bloque principal del programa
{
 clrscr();                                                 // Limpia la pantalla
 archiv="rlebruce.bmp";                                    // Asigna a una variable el nombre del archivo a leer
 if (Abre_lectura_bmp(archiv))                             // Si logra abrir el archivo continua con el programa
  {
   Lee_Cabecera_BMP();                                     // Lee los datos de la cabecera y los coloca en pantalla
   getch;                                                  // Pausa para tomar nota de la cabecera
   Lee_Paleta_BMP;                                         // Lee los datos de la paleta
   Inicializa(16,4);                                       // Inicializa el modo gráfico, 1024x768x256
   Pon_Paleta_BMP();                                       // Activa la paleta leída del archivo
   Lee_imagen_BMP();                                       // Descomprime la imagen y la visualiza
   getch();                                                // Pausa para ver la imagen
   fclose(lee_arch);                                       // Cierra archivo
   closegraph;                                             // Cierra libreria de gráficos
   restorecrtmode();                                       // Recupera el modo texto
   clrscr();                                               // Limpia pantalla
  }
}
                                                           // Cuerpo de cada una de las funciones declaradas
int Inicializa(int Driver, int Modo)                       // Inicializa modo gráfico y reporta los errores que ocurran
{                                                          // Esta fiunción ya no necesita explicación alguna
 int ErrorG;
 unsigned int x,y;
 union REGS regin,regout;
 if (Driver==16)
 Driver = installuserdriver("Svga256",NULL);
 if (Driver==grError) exit;
 initgraph( &Driver, &Modo,"d:\borlandc\programa\exeobj");
```

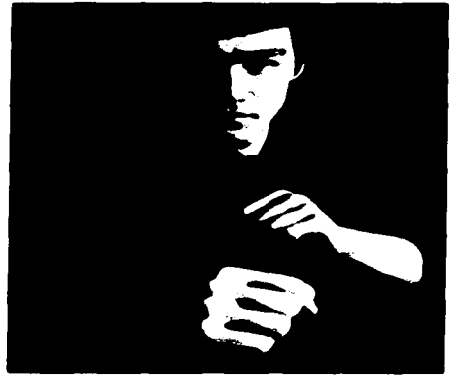
```
  regin.x.ax=0x000f;
  regin.x.cx=0x8;
  regin.x.dx=0x8;
  int86(0x33,&regin,&regout);
  x=getmaxx();
  y=getmaxy();
  regin.x.ax=x;
  regin.x.dx=regin.x.ax;
  regin.x.ax=0x0007;
  regin.x.cx=0x0;
  int86(0x33,&regin,&regout);
  regin.x.ax=y;
  regin.x.dx=regin.x.ax;
  regin.x.ax=0x0008;
  regin.x.cx=0x0;
  int86(0x33,&regin,&regout);
  ErrorG=graphresult();
 if (ErrorG!=grOk)
  {
   printf("\n Error en graficos : %s",grapherrormsg(ErrorG));
   restorecrtmode;
   return 0;
  }
 else
  {
   cleardevice;
   return 1;
  }
}

void Cambia_color(BYTE color,BYTE r,BYTE v,BYTE a)      // Modifica la paleta de colores activa en el S.O.
{                                                       // Esta función ya no necesita explicación alguna
 asm mov dx,3C8h
 asm mov al,color
 asm out dx,al
 asm inc dx
 asm mov al,r
 asm out dx,al
 asm mov al,v
 asm out dx,al
 asm mov al,a
 asm out dx,al
}

int Abre_lectura_bmp(char *arch_imagen)                 // Prepara al archivo para su lectura
{                                                       // Esta función no necesita explicación
 lee_arch=fopen(arch_imagen,"rb");
 if (lee_arch==NULL)
  { return 0; }
 else
  { return 1; }
}
```

```
 void Lee_Cabecera_BMP()                                   // Lee los dctos de la cabecera y los despliega en pantalla
 {
 int leyo;
 fpos_t posicion;
 fread((char *)&bmp,1,sizeof(cabeza_bmp),lee_arch);          // Lee los datos de la cabecera y los almacena
 fgetpos(lee_arch,&posicion);                                // Asigna la posición actual del puntero de archivo
 clrscr;                                                     // El siguiente bloque despliega los datos almacenados
 gotoxy(1,1);printf("Cabecera del archivo BMP de MicroSoft Windows %s",archiv);
 gotoxy(1,2);printf("Tipo de bmp:%c%c Posición del puntero :%d",bmp.tipo[0],bmp.tipo[1], posicion);
 gotoxy(1,3);printf("Tamaño del arch:%ld",bmp.tamano);
 gotoxy(1,4);printf("tam_cabeza: %ld %d %d ",bmp.tam_cabeza,bmp.reservado1,bmp.reservado2);
 gotoxy(1,5);printf("Tipo de bmp        : %ld",bmp.tipo_bmp);
 if (bmp.tipo_bmp==40)                                       // Si tipo es 40, Windows v3.0
   {                                                         // Lee los datos del archivo y los almacena
    fread((char *)&bmpi,1,sizeof(informe_bmp),lee_arch);     // Despliega los datos almacenados
    gotoxy(1,6) ;printf("Xmaximo          : %ld",bmpi.xmaximo);
    gotoxy(1,7) ;printf("Ymaximo          : %ld",bmpi.ymaximo);
    gotoxy(1,8) ;printf("planos           : %d",bmpi.planos);
    gotoxy(1,9) ;printf("bits por pixel   : %d",bmpi.contador);
    gotoxy(1,10);printf("tipo de compresion  : %ld",bmpi.tipo_compresion);
    gotoxy(1,11);printf("tamaño de imagen    : %ld",bmpi.tamano_imagen);
    gotoxy(1,12);printf("bi x             : %ld",bmpi.bitxmetrox);
    gotoxy(1,13);printf("bi y             : %ld",bmpi.bitxmetroy);
    gotoxy(1,14);printf("bi clr used      : %ld",bmpi.biclrusado);
    gotoxy(1,15);printf("bi clr import    : %ld",bmpi.biclrimport);
 }
else                                                         // Es 12, Windows v2.0 y Os /2 v1.0
 {                                                           // Lee los datos del archivo y los almacena
 fread((char *)&bmpc,1,sizeof(centro_bmp),lee_arch);
 gotoxy(1,6);printf("Xmaximo:%d",bmpc.xmaximo);            // Despliega los datos almcenados
 gotoxy(1,7);printf("Ymaximo          : %d",bmpc.ymaximo);
 gotoxy(1,8);printf("planos           : %d",bmpc.planos);
 gotoxy(1,9);printf("bits por pixel   : %d",bmpc.contador);
 }
}

void Lee_Paleta_BMP()                                        // Lee la paleta de colores y la almacena en un arreglo
{
 if (bmp.tipo_bmp==40)                                       // Si es Windows v3.0
   fread((BYTE *)&bmp_pali,256,sizeof(paleta_bmpi),lee_arch);  // Lee y Almacena la paleta
 else                                                        // Windows v2.0 y OS/2 v1.0
   fread((BYTE *)&bmp_palc,256,sizeof(paleta_bmpc),lee_arch);  // Lee y Almacena la paleta
}


void Pon_Paleta_BMP()                                        // Activa la paleta de colores del archivo
{ int i;
 if (bmp.tipo_bmp==40)                                       // Si es paleta de Windows v3.0
  for(i=0;i<=255;i++)                                        // Cambia la paleta
    Cambia_color(i,bmp_pali[i].rojo >> 2,bmp_pali[i].verde >> 2,bmp_pali[i].azul >> 2);
 else                                                        //Es 12, Windows v2.0 y OS/2 v1.0
  for (i=0;i<=255;i++)                                       // Cambia la paleta
    Cambia_color(i,bmp_palc[i].rojo >> 2,bmp_palc[i].verde >> 2,bmp_palc[i].azul >> 2);
}
```

```
void Lee_imagen_BMP()                          // Descomprime los datos de imagen y los pone en pantalla
{
 int ponx,pony,numcols;                        // Variables para el almacenamiento de datos del pixel y auxiliar
 LONGINT i,j,k,cont,tam_imagen;                                                    // Variables auxiliares
 BYTE unbyte,cuantos;                                                   // Variables para los dos bytes de código
 char salte;                                                            // Bandera para continuar decodificando
 numcols=256;                                  // Se puede sustituir por número de colores = 2^(contador) = 2^3 = 256
 fseek(lee_arch,bmp.tam_cabeza,SEEK_SET);                       // Posiciíona el puntero al final de la cabecera
 if (bmp.tipo_bmp==40)                                                                   // Si es Windows v3.0
 {
   if (bmpi.tipo_compresion==0)                 // Confirma el valor de bmpi.xmaximo, por lo que ya se explico
      bmpi.xmaximo=(bmp.tamano-bmp.tam_cabeza)/bmpi.ymaximo;
   tam_imagen=((bmpi.ymaximo)*bmpi.xmaximo);                                  // Calcula el tamaño total de la imagen
   switch (bmpi.tipo_compresion)                                                          // Para cada tipo de compresión
     {
        case 0:                                                                   // Datos de imagen sin comprimir
          for (i=0;i<=(tam_imagen-1);i++)                                                 // Para cada dato de la imagen
           {
             if (numcols==256)                                                            // Si número de colores es 256
                fread((unsigned char*)&unbyte,1,1,lee_arch);
            else
              if (numcols==16)                                                                        // Para 16 colores
               {  }
                else
               if (numcols==2)                                                                         // Para 2 colores
                 { }
               ponx=(i%(bmpi.xmaximo));                                                   Calcula la posición X del pixel
               pony=(i/(bmpi.xmaximo));                                                 // Calcula la posicion Y del pixel
               putpixel(ponx,bmpi.ymaximo-pony,unbyte);                              // Pone en pantalla el dato del pixel
           }
        break;
        case 1:                                                                                  // Compresión RLE8
        i=0;j=0;cont=0;k=0;salte=False;                                                   // Inicialización de variables
        do
         { fread((unsigned char *)&unbyte,1,1,lee_arch);                                  // Lectura del primer byte de código
          fread((unsigned char *)&cuantos,1,1,lee_arch);                                 // Lectura del segundo byte de código
            if(unbyte==0)                                                                                  // Marcadores del RLE
            {
               switch (cuantos)                                                          // Bloque de opciones para el segundo byte
               {
                 case 0:                                                                                 // Es un código de fin de línea
                   j++;                                                                                 // Incrementa la cuenta de líneas
                   i=0;                                                                          // Incrementa la cuenta de datos por línea
                 break;
                 case  1:                                                                                // Es un código de Fin de archivo
                   salte=True;                                                                            // Termina la decodificación
                 break;
                 case  2:                                                                              // Es un código Delta, no soportado
                 break;
                 default:                                                                  // Si no es algún marcador son datos sin codificar
                   for (k=1;k<=cuantos;k++)                                                   // Impime tantos datos diferentes como
                   {                                                                                          // sea el valor de cuantos
                    i++;                                                                  // Incrementa i, cuenta de datos decodificados por línea
                    cont++;                                                                  // Incrementa cont, cuenta de datos decodificados
```

```
                    fread((unsigned char *)&unbyte,1,1,lee_arch);          // Lee el sig. dato diferente
                    ponx=(cont % (bmpi.xmaximo));                          // Calcula la posición X del pixel
                    pony=(cont / (bmpi.xmaximo));                          // Calcula la posición Y del pixel
                    putpixel(ponx,bmpi.ymaximo-pony,unbyte);               // Pone en pantalla el dato del pixel
                   }
                 if ((cuantos % 2)!=0)                                     // Número de bytes sin codificar es impar
                    fread((unsigned char *)&unbyte,1,1,lee_arch);          // Completa el par
                 break;
                }
              }
              else                                                         // Unbyte es diferente de '0', datos iguales codificados
               { for (k=1;k<=unbyte;k++)                                   // Imprime tantos datos como sea el valor de unbyte
                 {
                   i++;                                                    // Incrementa i, cuenta de datos decodificados por línea
                   cont++;                                                 // Incrmenta cont, cuenta de datos decodificados
                   ponx=(cont % (bmpi.xmaximo));                           // Calcula la posición X del pixel
                   pony=(cont / (bmpi.xmaximo));                           // Calcula la posición Y del pixel
                   putpixel(ponx,bmpi.ymaximo-pony,cuantos);               // Pone en pantalla el dato del pixel
                  }
               }
            } while (salte==False);                                        // Mientras no encuentre el fin de archivo
          case 2:                                                          // RLE de 4 bits
           break;
       }
   }
   else                                                                    // Es windows v2.0
   {
    bmpc.xmaximo=((bmp.tamano-bmp.tam_cabeza)/bmpc.ymaximo);               // Para confirmar
    tam_imagen=bmpc.ymaximo;
    tam_imagen=(tam_imagen*bmpc.xmaximo);
     for (i=0;i<=tam_imagen;i++)
      {
        if (numcols==256)                                                  // Si número de colores es 256
           fread((char *)&unbyte,1,1,lee_arch);                            // Lee un byte y lo almacena en unbyte
        else                                                               // Número de colores no es 256
           if (numcols==16)                                                // Para 16 colores
              { }
           else
              if (numcols==2)                                              // Para 2 colores
                 { }
        ponx=(i%(bmpc.xmaximo));                                           // Calcula la posición X del pixel
        pony=(i/(bmpc.xmaximo));                                           // Calcula la posición Y del pixel
        putpixel(ponx,bmpc.ymaximo-pony,unbyte);                           // Pone en pantalla el dato del pixel
      }
   }
}
```

Una de las imágenes procesadas con el programa es: rlebruce.bmp, la cual es una imagen tipo Windows 3.x con codificación RLE.



Rlebruce.bmp

El programa da como resultado los siguientes datos:

```
Cabecera del archivo BMP de MicroSoft Windows rlebruce.bmp
Tipo de bmp            : BM    Posición del puntero :18
Tamaño del arch        : 31686
tam_cabeza             : 1078 0 0
Tipo de bmp            : 40
xmaximo                : 380
ymaximo                : 400
planos                 : 1
bits por pixel         : 8
tipo de compresion     : 1
tamaño de imagen       : 30528
bi x                   : 0
bi y                   : 0
bi clr used            : 256
bi clr import          : 256
```

Posteriormente se despliega la imagen.

Como se puede observar la diferencias al programar en C o Pascal el formato BMP son casi nulas y no tiene gran problema para la decodificación de los datos de imagen. La estructura del progamama es flexible y contempla el poder añadir los procesos para las opciones no analizadas por el programa (RLE4, 2 y 16 colores) .

IV.5 Formato PCX

El formato PCX también es uno de los formatos de archivos gráficos fáciles de programar. Por la estructura del formato solo se utiliza 1 estructura:

Cabecera. Estructura de la cabecera del archivo PCX.

IV.5.1 En Lenguaje Pascal

En Pascal el programa se codifica de la siguiente manera.

```
Program Lee_pcx;          { Programa que lee el formato de archivo PCX.
                           Realizado para el trabajo de tesis titulado:FORMATOS GRÁFICOS
                           Programadores :
                                        Cantero Ramírez Carlos
                                        Trejo Bonaga Marilu
                          Asesores :
                                        Manzo Salazar Itsmael
                                        Monterrosa Escobar Amilcar A.  }

Uses crt,dos,graph;

Type                                                          {Definición de la estructura}
 Cabecera=Record    .                                         {Cabecera del archivo}
     manufactura:byte;                                        {Identificador de PCX, valor 10}
     version:byte;                                            {Versión del archivo}
     decodifica:byte;                                         {Tipo de compresión }
     bits_por_pixel:byte;                                     {Bits por pixel}
     xminimo:Integer;                                         {Coordenada X mínima }
     yminimo:Integer;                                         {Coordenada Y mínima }
     xmaximo:Integer;                                         {Coordenada X máxima}
     ymaximo:Integer;                                         {Coordenada Y máxima}
     horiz_dpi:Integer;                                       {Resolución horizontal}
     vert_dpi:Integer;                                        {Resolución vertical}
     mapa_color:Array [1..48] of byte;                        {Arreglo de la paleta EGA}
     reservado:byte;                                          {Reservado, '0'}
     col_planos:byte;                                         {Planos de color}
     bit_linea_plano:Integer;                                 {Tamaño línea sin comprimir}
     tipo_paleta:Integer;                                     {Tipo de paleta}
     relleno:Array[1..58] of Byte;                            {Relleno de la cabecera PCX}
 End;

 Var
  Archivo:string;                                             {Variable para el nombre del archivo a leer}
  tam_imagen:Longint;                                         {Tamaño de la imagen}
  paleta:Array [0..767] of byte;                              {Arreglo para la paleta de  256 colores}
  i,j,k:longint;                                              {Variables auxiliares}
  Lee_arch:File;                                              {Archivo de lectura}
  posicion,leyo:integer;                                      {Variables auxiliares}
  pcx:Cabecera;                                               {Variable para almacenar la cabecera}
```

```
Function Inicializa(Driver,Modo:integer):boolean;        {Inicializar gráficos y reporta los errores que ocurran}
var
 ErrorG:integer;                                          {Esta función  no necesita explicación}
 x,y:word;
Begin
 if (Driver=16) Then
        Driver := InstallUserDriver('Svga256',nil);
 if Driver=grError Then
        halt(1);
 InitGraph(Driver, Modo,'d:\bp\programa');
  asm
    mov ax,000fh
    mov cx,8
    mov dx,8
    int 33h
  end;
  x:=getmaxx;
  y:=getmaxy;
  asm
    mov ax,x
    mov dx,ax
    mov ax,0007h
    mov cx,0h
    int 33h
    mov ax,y
    mov dx,ax
    mov ax,0008h
    mov cx,0h
    int 33h
  end;
 ErrorG:=graphResult;
 If ErrorG<>grok Then
  Begin
   Writeln('Error en graficos : ',GraphErrormsg(ErrorG));
   Inicializa:=False;
   Restorecrtmode;
  End
 Else
  Begin
   cleardevice;
   Inicializa:=True;
  End;
End;


Procedure Cambia_color(color,r,v,a:Byte);                {Modifica la paleta de colores activa n el S.O.}
Begin                                                    {Esta función ya no necesita explicación alguna}
 asm
  mov dx,3C8h
  mov al,color
  out dx,al
  inc dx
  mov al,r
  out dx,al
```

```
 mov al,v
 out dx,al
 mov al,a
 out dx,al
 End;
End;


Function Abre_lectura_PCX(Arch_imagen:String):Boolean;          {Prepara al archivo para su lectura}
Begin                                                            {Esta función ya no necesita explicación}
 Assign(Lee_Arch,arch_imagen);
{Sy-}
  reset(Lee_arch,1);
{Si+}
 if ioresult = 0 Then
     Abre_lectura_PCX:=True
 else
  Begin
   Writeln('Error de apertura de archivo');
   Abre_lectura_PCX:=False;
  End;
End;


Procedure Lee_Cabecera_PCX;                    {Lee los datos de la cabecera y los despliega en pantalla}
Var
 i,y,k:byte;
Begin
 Blockread(Lee_arch,pcx,sizeof(pcx),leyo);                       {Lee los datos del archivo y los almacena}
 posicion:=Filepos(Lee_arch);                                    {Asigna la posición actual del puntero del archivo}
 Gotoxy(01,01);Write('Manufactura    :');                        {Despliega los datos almacenados}
 Gotoxy(25,01);Writeln(pcx.manufactura,'    ','Posición del puntero :',posicion);
 Gotoxy(01,02);Write('Version        :');
 Gotoxy(25,02);Writeln(pcx.version);
 Gotoxy(01,03);Write('Decodifica     :');
 Gotoxy(25,03);Writeln(pcx.decodifica);
 Gotoxy(01,04);Write('Bits_por_pixel :');
 Gotoxy(25,04);Writeln(pcx.bits_por_pixel);
 Gotoxy(01,05);Write('Xminimo        :');
 Gotoxy(25,05);Writeln(pcx.xminimo);
 Gotoxy(01,06);Write('Yminimo        :');
 Gotoxy(25,06);Writeln(pcx.yminimo);
 Gotoxy(01,07);Write('Xmaximo        :');
 Gotoxy(25,07);Writeln(pcx.xmaximo);
 Gotoxy(01,08);Write('Ymaximo        :');
 Gotoxy(25,08);Writeln(pcx.ymaximo);
 Gotoxy(01,09);Write('Horiz_dpi      :');
 Gotoxy(25,09);Writeln(pcx.horiz_dpi);
 Gotoxy(01,10);Write('Vert_dpi       :');
 Gotoxy(25,10);Writeln(pcx.vert_dpi);
 Gotoxy(01,11);Writeln('mapa de colores:');                      {Despliega en pantalla los elementos EGA}
 y:=12;k:=1;                                                     {de la paleta}
 For i:=1 to 48 do
  Begin
```

```
  Gotoxy((k*4),y);Write(pcx.mapa_color[i]);
  if i mod 15 =0 Then
    Begin
      inc(y);
      k:=0;
    End;
  inc(k);
 End;                                                    {Continua desplegando los datos restantes}
Gotoxy(01,15);Write('reservado      :');
Gotoxy(25,15);Writeln(pcx.reservado);
Gotoxy(01,16);Write('Col_Planos     :');
Gotoxy(25,16);Writeln(pcx.col_planos);
Gotoxy(01,17);Write('Bit_linea_plano:');
Gotoxy(25,17);Writeln(pcx.bit_linea_plano);
Gotoxy(01,18);Write('Tipo_paleta    :');
Gotoxy(25,18);Writeln(pcx.tipo_paleta);
Gotoxy(01,19);Writeln('relleno      :');
y:=20;k:=1;                                          {Despliega en pantalla el relleno de la cabecera PCX}
For i:=1 to 58 do
 Begin
   Gotoxy((k*4),y);Writeln(pcx.relleno[i]);
   If i mod 15 =0 Then
     Begin
       inc(y);
       k:=0;
     End;
   inc(k);
 End;
End;


Procedure Lee_Paleta_PCX;                                {Lee la paleta de 256 colores y la almacena}
Var
 Pal256:byte;
Begin
 If (pcx.manufactura=10) and (pcx.version=5)  Then                {Para la version de 256 colores}
  Begin
    Seek(Lee_arch,filesize(Lee_arch)-769);                {Se coloca en la posición del separador}
    Blockread(Lee_arch,pal256,sizeof(pal256),leyo);
    If (pal256=12) Then                                  {Si es 12, separador entre datos y paleta}
      Begin
        Seek(Lee_arch,filesize(Lee_arch)-768);                {Se coloca en la  posicón de la paleta}
        Blockread(Lee_arch,paleta,768);                  {Lee los últimos 768 datos y los almacena}
        for i:=0 to 767 do
            paleta[i]:=paleta[i] shr 2;                       {Realiza un desplazamiento de 2 bits}
        Gotoxy(60,1);Write(' Tiene 256 Colores ');
     End
    Else
        Gotoxy(60,1);Writeln(' Error no es de 256 Colores ');
 End
Else
  Begin                                              {Para La version de 16 colores, insertar código}
  End;
End;
```

```
Procedure Pon_Paleta_PCX;                                          {Activa la paleta de colores del archivo}
Var
 i:integer;
Begin
 For i:=0 to 255 do
   Cambia_color(i,paleta[(i*3)],paleta[(i*3)+1],paleta[(i*3)+2]);                    {Cambia la paleta}
End;


Procedure Lee_imagen_PCX;                          {Descomprime los datos de imagen y los pone en pantalla}
Var
 modo,ponx,pony:integer;                          {Variables auxiliar y para el almacenamiento de datos del pixel}
 outbyte,bytecont,bytemodo,runmodo:byte;                          {Variables de bytes de códigos y auxiliares}
Begin
 seek(Lee_arch,127);                {Se coloca en  la posición de inicio de datos de imagen, final de la cabecera}
 bytemodo:=0;                                                                          {Indica datos sin codificar}
 runmodo:=1;                                                                           {Indica datos codificados}
 modo:=bytemodo;                                                                {Por default empieza con datos sin codificar}
 j:=(pcx.xmaximo+1);                                                                                {Asigna X máximo}
 k:=(pcx.ymaximo+1);                                                                                {Asigna Y máximo}
 tam_imagen:=j*k;                                                                          {Calcula el tamaño de la imagen}
 For i:=0 to (tam_imagen-1) do                                                                {Para cada dato de la imagen}
 Begin
   If (modo=bytemodo) Then                                                                {Si esta en modo datos sin codificar }
      Begin
        BlockRead(Lee_arch,outbyte,1,leyo);                                                                   {Lee un dato}
         if (outbyte>$C0) then                                                {Si el dato es mayor a 192, son datos codificados}
          Begin
            bytecont:=(outbyte and $3f);                                                {Calcula las repeticiones del siguiente dato}
            BlockRead(Lee_arch,outbyte,1,leyo);                                                          {Lee el dato a repetir}
            bytecont:=bytecont-1;                                                       {Decrementa la cuenta de repetición del dato}
            if (bytecont>0) Then                                                   {Si bytecont>0, cambia a modo datos codificados.}
                modo:=Runmodo;
          End;                                                                                       {Hay repetición del dato}
       End
      Else                                                                                {Modo datos codificados, repetición del dato}
       Begin
         bytecont:=bytecont-1;                                                          {Decrementa la cuenta de repetición del dato}
         If (bytecont)=0 Then                                                        {Si bytecont = 0, ya no hay repetición del  dato}
             modo:=bytemodo;
       End;
 ponx:=(i mod (pcx.xmaximo+1));                                                                   {Calcula la posición X del pixel}
 pony:=(i div (pcx.xmaximo+1));                                                                   {Calcula la posición Y del pixel}
 putpixel(ponx,pony,outbyte);                                                                {Pone en pantalla el dato del pixel}
 End;
End;


Begin                                                                                   {Cuerpo del bloque principal del programa}
 clrscr;                                                                                                   {Limpia la pantalla}
 Archivo:='arana.pcx';                                                                {Asigna a una variable el nombre del archivo a leer}
 If Abre_lectura_PCX(Archivo) Then                                                   {Si logra abrir el archivo continua con el programa}
```

```
  Begin
    Lee_Cabecera_PCX;          {Lee los datos de la cabecra y los coloca en pantalla}
    Readkey;                   {Pausa para tomar nota de la cabecera}
    Lee_Paleta_PCX;            {Lee los datos de la paleta}
    Inicializa(16,4);          {Inicializa el modo gráfico, 1024x768x256}
    Pon_Paleta_PCX;            {Activa la paleta leida del archivo}
    Lee_imagen_PCX;            {Descomprime la imagen y la visualiza}
    Readkey;                   {Pausa para ver la imagen}
    close(Lee_arch);           {Cierra el archivo}
    restorecrtmode;            {Recupera el modo texto}
    closegraph;                {Cierra libreria de gráficos}
    clrscr;                    {Limpia pantalla}
  End;
End.
```

Una de las imágenes procesadas con el programa es: arana.pcx.

Arana.pcx

El programa da como resultado los siguientes datos:

```
Nombre de archivo PC-Zsoft:  arana.pcx            Tiene 256 Colores
Manufactura       :          10      Posición del puntero :128
Version           :          5
Decodifica        :          1
Bits_por_pixel    :          8
Xminimo           :          0
Yminimo           :          0
Xmaximo           :          618
Ymaximo           :          978
Horiz_dpi         :          150
Vert_dpi          :          150
Mapa de colores:
   0   0   0   8   0   0   0   0   0   8   8   0   8   0   0   16
   8   0   16  0   0   24  0   8   0   0   16  0   0   24  0
   8   8   0   0   8   8   0   8   16  0   12  16  0   24  8
reservado         :          0
Col_Planos        :          1
Bit_linea_plano:             620
Tipo_paleta       :          1
relleno           :
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0
```

Posteriormente se despliega la imagen.

IV.5.2 En Lenguaje C

El código en C del programa se muestra a continuación.

```
// Programa que lee el formato de archivo PCX.
 Realizado para el trabajo de tesis titulado:FORMATOS GRÁFICOS
                    Programadores :
                              Cantero Ramírez Carlos
                              Trejo Bonaga Marilu
                    Asesores :
                              Manzo Salazar Itsmael
                              Monterrosa Escobar Amilcar A.
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>
#include <dos.h>
#include <conio.h>
#include <io.h>

#define BYTE    unsigned char                    // Definición de constantes simbólicas
#define True  1
#define False 0
#define LONGINT unsigned long
#define WORD    unsigned int

                                                 // Definición de la estructura principal
typedef struct                                   // Cabecera del archivo
 { BYTE manufactura,                             // Identificador de PCX, valor 10
        version,                                 // Versión del archivo
        decodifica,                              // Tipo de compresión
        bits_por_pixel;                          // Bits por pixel
   int  xminimo,                                 // Coordenada X mínima
        yminimo,                                 // Coordenada Y mínima
        xmaximo,                                 // Coordenada X máxima
        ymaximo,                                 // Coordenada Y máxima
        horiz_dpi,                               // Resolución horizontal
        vert_dpi;                                // Resolución vertical
  BYTE  mapa_color[48];                          // Arreglo de la paleta EGA
  BYTE  reservado,                               // Reservado, '0'
        col_planos;                              // Planos de color
   int  bit_linea_plano,                         // Tamaño línea sin comprimir
        tipo_paleta;                             // Tipo de paleta
  BYTE relleno[58];                              // Relleno de la cabecera PCX
 }Cabecera;

int leyo;                                        // Variable auxiliar
fpos_t posicion;                                 // Variable auxulia
char *archivo;                                   // Variable para el nombre del archivo a leer
LONGINT tam_imagen;                              // Tamaño de la imagen
int i,j,k;                                       // Variables auxiliares
BYTE paleta[768];                                // Arreglo para la paleta 256
cabecera pcx;                                    // Variable para almacenar la cabecera
FILE *lee_arch;                                  // Archivo de lectura
```

```
                                                    // Bloque de declaración de las funciones utilizadas
void Lee_Cabecera_PCX(void);
int Inicializa(int Driver, int Modo);
int Abre_lectura_PCX(char *arch_imagen);
void Lee_Paleta_PCX(void);
void Pon_Paleta_PCX();
void Lee_imagen_PCX();
void Cambia_color(BYTE color,BYTE r,BYTE v,BYTE a);


void main ()                                        // Cuerpo del bloque principal del programa
{
   clrscr();                                                                  // Limpia pantalla
   archivo="pcxleon.pcx";                          // Asigna  auna variable el nombre del archivo a leer
   if (Abre_lectura_PCX(archivo))                  // Si logra abrir el archivo continua con el programa
    {
       Lee_Cabecera_PCX();                         // Lee los datos de la cabecera y los coloca en pantalla
       getch();                                                   // Pausa para tomar nota de la cabecera
       Lee_Paleta_PCX();                                                  // Lee los datos de la paleta
       Inicializa(16,4);                           // Inicializa el modo gráfico, 1024x768x256
       Pon_Paleta_PCX();                                              // Activa la paleta leida del archivo
       Lee_imagen_PCX();                                          // Descomprime la imagen y la visualiza
       getch();                                                              // Pausa para ver la imagen
       fclose(lee_arch);                                                                // Cierra el archivo
       close graph;                                                          // Cierra librería de gráficos
       restorecrtmode();                                                      // Recupera el modo texto
       clrscr();                                                                        // Limpia pantalla
    }
}

                                                    // Cuerpo de cada una de las funciones declaradas
int Inicializa(int Driver, int Modo)                // Inicializar gráficos y reportar los errores que ocurran
{                                                                   // Esta función no necesita explicación
  int ErrorG;
  unsigned int x,y;
  union REGS regin,regout;
  if (Driver==16)
      Driver = installuserdriver("Svga256",NULL);
  if (Driver==grError) exit;
  initgraph( &Driver, &Modo,"d:\borlandc\programa\exeobj");
  regin.x.ax=0x000f;
  regin.x.cx=0x8;
  regin.x.dx=0x8;
  int86(0x33,&regin,&regout);
  x=getmaxx();
  y=getmaxy();
  regin.x.ax=x;
  regin.x.dx=regin.x.ax;
  regin.x.ax=0x0007;
  regin.x.cx=0x0;
  int86(0x33,&regin,&regout);
  regin.x.ax=y;
  regin.x.dx=regin.x.ax;
  regin.x.ax=0x0008;
```

```
 regin.x.cx=0x0;
 int86(0x33,&regin,&regout);
 ErrorG=graphresult();
 if (ErrorG!=grOk)
 { printf("\n Error en graficos : %s",grapherrormsg(ErrorG));
    restorecrtmode;
    return 0;
  }
 else
  { cleardevice;
    return 1;
  }
}


void Cambia_color(BYTE color,BYTE r,BYTE v,BYTE a)                    // Modifica la paleta activa
{                                                                    // Esta función no necesita explicación
 asm mov dx,3C8h
 asm mov al,color
 asm out dx,al
 asm inc dx
 asm mov al,r
 asm out dx,al
 asm mov al,v
 asm out dx,al
 asm mov al,a
 asm out dx,al
}


int Abre_lectura_PCX(char *arch_imagen)                              // Prepara al archivo para su lectura
{                                                                    // Esta función no necesita explicación
  lee_arch=fopen(arch_imagen,"rb");
   if (lee_arch==NULL)
     { return 0; }
   else
     { return 1; }
 }


 void Lee_Cabecera_PCX()                          // Lee los datos de la cabecera y los despliega en pantalla
 {
   BYTE i,y,k;                                                                        // Variables auxiliares
   fread((BYTE *)&pcx,1,sizeof(pcx),lee_arch);                          // Lee los datos del archivo y los almacena
   fgetpos(lee_arch,&posicion);                                         // Asigna la posición del puntero del archivo
   gotoxy( 1, 1);printf("Manufactura    :%d    Posición del puntero :%d",pcx.manufactura,posicion);
   gotoxy( 1, 2);printf("Version        :%d",pcx.version);                          // Despliega los datos almacenados
   gotoxy( 1, 3);printf("Decodifica     :%d",pcx.decodifica);
   gotoxy( 1, 4);printf("Bits_por_pixel :%d",pcx.bits_por_pixel);
   gotoxy( 1, 5);printf("Xminimo        :%d",pcx.xminimo);
   gotoxy( 1, 6);printf("Yminimo        :%d",pcx.yminimo);
   gotoxy( 1, 7);printf("Xmaximo        :%d",pcx.xmaximo);
   gotoxy( 1, 8);printf("Ymaximo        :%d",pcx.ymaximo);
   gotoxy( 1, 9);printf("Horiz_dpi      :%d",pcx.horiz_dpi);
```

```
   gotoxy( 1,10);printf("Vert_dpi      :%d",pcx.vert_dpi);
   gotoxy( 1,11);printf("Mapa de colores:");                    // Despliega los elementos de la paleta EGA
   y=12;
   k=1;
   for (i=1;i<=48;i++)
   {
       gotoxy((k*4),y);printf("%d",pcx.mapa_color[i-1]);
       if ((i % 15) ==0)
        {
            y++;
            k=0;
        }
       k++;
   }                                                          // Continua desplegando los datos restantes
   gotoxy( 1,15);printf("reservado      :%d",pcx.reservado);
   gotoxy( 1,16);printf("Col_Planos     :%d",pcx.col_planos);
   gotoxy( 1,17);printf("Bit_linea_plano:%d",pcx.bit_linea_plano);
   gotoxy( 1,18);printf("Tipo_paleta    :%d",pcx.tipo_paleta);
   gotoxy( 1,19);printf("relleno        :");                // Despliega en pantalla el relleno de la cabecera PCX
   y=20;k=1;
   for (i=1;i<=58;i++)
   { gotoxy((k*4),y);printf("%d",pcx.relleno[i-1]);
     if ((i % 15)==0)
      { y++;
        k=0;
      }
      k++;
   }
 }


 void Lee_Paleta_PCX()                                        // Lee la paleta de 256 colores y la almacena
 {
   BYTE pal256;
   if ((pcx.manufactura==10) && (pcx.version==5))             // Para la version de 256 colores
  { fseek(lee_arch,-769,SEEK_END);          // Se coloca en la posición donde puede estar el separador
    fread((BYTE *)&pal256,1,sizeof(pal256),lee_arch);
    if (pal256==12)                                          // Si es 12, separador entre datos y paleta
     { fseek(lee_arch,-768,SEEK_END);          // Se coloca en la posición 768 antes del fin del archivo
       fread((BYTE *)&paleta,1,sizeof(paleta),lee_arch);     // Lee la paleta y la almacena
       for (i=0;i<=767; i++)
        {  paleta[i]=paleta[i] >> 2;                         // Realiza el desplazamiento de bits
         }
       gotoxy(50,1);printf(" Tiene 256 Colores ");
     }
    else
     {
       gotoxy(50,1);printf(" Error no es de 256 Colores ");
        exit;
     }
  }
  else                                                       Para La version de 16 colores
     {  }
}
```

```
void Pon_Paleta_PCX()                                          // Activa la paleta de colores del archivo
{
  int y;
  for (i=0;i<=255;i++)                                         // Cambia la paleta
     Cambia_color(i,paleta[(i*3)],paleta[(i*3)+1],paleta[(i*3)+2]);
}


void Lee_imagen_PCX()                          // Descomprime los datos de image y los pone en pantalla
{
  int modo,ponx,pony,aux1;                 // Variables auxiliares y para almacenamiento de datos del pixel
  LONGINT i,j;                                                                  // Variables auxiliares
  BYTE outbyte,bytecont,bytemodo,runmodo;                   // Variables de bytes de códigos y auxiliares
  fseek(lee_arch,128,SEEK_SET);          // Busca el inicio de datos de imagen, al final de la cabecera
  bytemodo=0;                                                                  // Indica datos sin codificar
  runmodo=1;                                                                   // Indica datos codificados
  modo=bytemodo;                                                  // Por default empieza con datos sin codificar
  j=(pcx.xmaximo+1);                                                                   // Asigna X máximo
  k=(pcx.ymaximo+1);                                                                   // Asigna Y máximo
  tam_imagen=j*k;                                                          // Calcula el tamaño de la imagen
  for (i=0;i<=tam_imagen;i++)                                                  // Para cada dato de la imagen
  {
    if (modo==bytemodo)                                                  // Si está en modo datos sin codificar
      {
        fread((BYTE *)&outbyte,1,1,lee_arch);                                                  // Lee un dato
        aux1=outbyte;                                                          // Lo asigna a una variable entera
        if (aux1>0xC0)                                                                  // Si el dato es mayor a 192
         {
          bytecont=(aux1 & 0x3F);                                     // Calcula las repeticiones del siguiente dato
          fread((BYTE *)&outbyte,1,1,lee_arch);                                          // Lee el dato a repetir
          aux1=outbyte;                                                          // Lo asigna una variable entera
          bytecont=bytecont-1;                                       // Decrementa la cuenta de repetición del dato
          if (bytecont>0) modo=runmodo;                       // Si bytecont > 0,cambia a modo datos codificados,
        }                                                                          // ya no hay repetición del dato
    }
   else                                                                  // Modo datos codificados, repetición del dato
    {
       bytecont=bytecont-1;                                             // Decrementa la cuenta de repetición del dato
       if (bytecont==0) modo=bytemodo;                                 // Si bytecont = 0, ya no hay repetición del dato
    }
   ponx=(i % (pcx.xmaximo+1));                                                        // Calcula la posición X del pixel
   pony=(i /(pcx.xmaximo+1));                                                         // Calcula la posición Y del pixel
   putpixel(ponx,pony,aux1);                                                        // Pone en pantalla el dato del pixel
  }
}
```

Una de las imágenes procesadas con el programa es: pcxleon.pcx.

Pcxleon.pcx

El programa da como resultado los siguientes datos:

```
Manufactura      :10      Posición del puntero :128  Tiene 256 Colores
Version          :5
Decodifica       :1
Bits_por_pixel   :8
Xminimo          :0
Yminimo          :0
Xmaximo          :383
Ymaximo          :255
Horiz_dpi        :150
Vert_dpi         :150
Mapa de colores:
   0   11   8   4   12  20  6   24  21  16  16  16  26  19  5
   20  20  18  20  37  16  33  28  16  0   24  33  18  30  24
   5   33  44  17  36  41  28  33  24  24  45  24  24  37  33
reservado        :0
Col_Planos       :1
Bit_linea_plano:384
Tipo_paleta      :1
relleno          :
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0<128>
```

Posteriormente se despliega la imagen.

En realidad la programación de la estructura del formato PCX tanto en Pascal como en C es la más simple de todos los formatos, en comparación a los otros formatos analizados en el presente trabajo. El programa deja abierta la opción para imágenes de 16 colores. No olvide que el estándar para este trabajo es de 8 bits por pixel (256 colores).

IV.6 Formato GIF

El formato GIF no es tan sencillo de programar, particularmente el método de descompresión. Su complejidad radica, en que sus códigos pueden ser de 9, 10 11 ó 12 bits, el problema es tomar los 8 bits de cada código sin perder la pista de códigos anteriores o posteriores del código que se esta leyendo. La cabecera debe contener la estructura del formato, utiliza 3 estructuras principales:

Descriptor_de_pantalla. Estructura de la cabecera y descriptor de pantalla lógico.

Paleta_GIF. Estructura de la paleta global.

Descriptor_de_imagen. Estructura del descriptor de imagen local.

Contiene otras 4 estructuras que especificabn cada uno de los bloques de extensión para GIF89a:

Ext_Control. Bloque de extensión para el control y manipuleo de gráficos, en el caso de imágenes múltiples.

Ext_Texto. Bloque de extensión de plano de texto.

Ext_Aplica. Bloque de extensión para la datos de la aplicación.

Exte_Coment. Bloque de estensión de comentarios.

Las funciones especificas de estos bloques de extensión se explicaron con detalle en el capítulo III. En el caso de los bloques de extensión, solo notifica si es que existe alguno de ellos; no realiza ningún proceso con los datos encrontrados en cada bloque.

El programa es capaz de leer archivos de 256 colores con imagen múltiples, sin embargo, para este caso solo lee la primera imagen del archivo, no lee imágenes interlazadas, pero si despliega sus trozos.

IV.6.1 En Lenguaje Pascal

En Pascal el programa se codifica de la siguiente manera.

```
Program Lee_GIF;
                    { Programa que lee el formato de archivo GIF.
                    . Realizado para el trabajo de tesis titulado:FORMATOS GRÁFICOS
                            Programadores :
                                    Cantero Ramirez Carlos
                                    Trejo Bonaga Marilu
                            Asesores :
                                    Manzo Salazar Itsmael
                                    Monterrosa Escobar Amilcar A.  }
```

```
Uses crt,dos,graph;
                                                        {Definición de las estructuras principales}

Type
Descriptor_de_pantalla = Record                         {Cabecera y descriptor de pantalla lógico}
  GIF,ver:Array[0..2] of CHAR;                          {Identificador (GIF) y versión ("87a" ó "89a")}
  xtotal,ytotal:Word;                                   {Ancho y largo de la pantalla en pixeles}
  describe,fondo,aspect_ratio:Byte;                     {Paquete, color de fondo y aspect ratio}
end;

Paleta_GIF=record                                       {Paleta de colores global}
   rojo,verde,azul: byte;
end;

Descriptor_de_imagen = Record                           {Descriptor de imagen local}
  separador:Char;                                       {Siempre 44=","}
  xminimo,yminimo,xmaximo,ymaximo:Word;                 {Coordenadas de inicio y fin de la imagen}
  inter:Byte;                                           {Variable de Interzado}
end;

Ar4096=Array [0..4096] of Integer;                      {Arreglo para  prefijo y sufijo }
Ar1024=Array [0..1024] of Integer;                      {Arreglo para los códigos de salida}

Var

 GIFdp:Descriptor_de_pantalla;                          {Variable para almacenar el descriptor de pantalla}
 GIFdi:Descriptor_de_imagen;                            {Variable para almacenar el descriptor de imagen}
 GIF_pal:Array [0..255] of paleta_GIF;                  {Variable para almacenar los datos de la paleta global}
 Posicion:integer;                                      {Variable auxiliar}
 Archivo:string;                                        {Variable para el nombre del archivo a leer}
 cont_imagen,tam_imagen:Longint;                        {Variable de cuenta de datos y tamaño de imagen}
 i,j,k:integer;                                         {Variable auxiliar}
 Sololee:Word;                                          {Variable auxiliar}
 Lee_Arch:File;                                         {Archivo de lectura}


Function Potencia(base,exp:Integer):Longint;            {Función para potencias}
Var
  i:Integer;
  Res:longint;
Begin
  res:=1;
  For i:=1 to exp do
     res:=res*base;
  Potencia:=res;
End;


Función Inicializa(Driver,Modo:integer):boolean;        {Inicializa gráficos y reporta los}
var                                                     {errores que ocurran}
  ErrorG:integer;                                       {Esta función no necesita explicación}
  x,y:word;
Begin
  if (Driver=16) Then Driver := InstallUserDriver('Svga256',nil);
```

```
 if Driver=grError Then halt(1);
InitGraph(Driver, Modo,'d:\bp\programa');
 asm
    mov ax,000fh
    mov cx,8
    mov dx,8
    int 33h
 end;
 x:=getmaxx;
 y:=getmaxy;
 asm
    mov ax,x
    mov dx,ax
    mov ax,0007h
    mov cx,0h
    int 33h
    mov ax,y
    mov dx,ax
    mov ax,0008h
    mov cx,0h
    int 33h
 end;
 ErrorG:=graphResult;
 If ErrorG<>grok Then
  Begin
   Writeln('Error en graficos : ',GraphErrormsg(ErrorG));
   Inicializa:=False;
   Restorecrtmode;
  End
 Else
  Begin
   cleardevice;
   Inicializa:=True;
  End;
End;


Procedure Cambia_color(color,r,v,a:Byte);                    {Cambia la paleta activa en el S.O.}
Begin                                                        {Esta función no necesita explicación}
asm
  mov dx,3C8h
  mov al,color
  out dx,al
  inc dx
  mov al,r
  out dx,al
  mov al,v
  out dx,al
  mov al,a
  out dx,al
 End;
 End;
```

```
Function Abre_lectura_GIF(Arch_imagen:String):Boolean;          {Prepara al archivo para su lectura}
Begin
 Assign(Lee_Arch,arch_imagen);                                   {Esta función no necesita explicación}
{$i-}
  reset(Lee_arch,1);
{$i+}
 if ioresult = 0 Then  Abre_lectura_GIF:=True
 else
  Begin
    Writeln('Error de apertura de archivo');
    Abre_lectura_GIF:=False;
  End;
End;


Procedure Lee_Cabecera_GIF;                  {Lee los datos de la cabecera y los despliega en pantalla}
Type                                {Define cada una de las estructuras para los bloques de extensión para "89a"}
 Ext_Control=Record                                                          {Bloque de control de gráficos}
   Introduc,                                                                                  {Introductor, 21h}
   Etiqueta,                                                                            {Etiqueta del bloque, F9h}
   Tam_Bloque,                                                                                {Tamaño del bloque}
   Paquete:Byte;                                                                {Método de colocación de gráficos}
   Espera:Word;                                                                                {Tiempo de espera}
   Color_index,                                                                   {Indice de color transparente}
   Terminador :Byte;                                                                           {Terminador,0h}
End;

Ext_Texto=Record                                                          {Bloque de extensión de plano de texto}
   Introduc,                                                                                  {Introductor, 21h}
   Etiqueta,                                                                            {Etiqueta del bloque, 01h}
   Tam_Bloque:Byte;                                                                           {Tamaño del bloque}
   Textx,                                                                                {Posición en X del texto}
   Texty,                                                                                {Posición en Y del texto}
   TextAncho,                                                                                  {Ancho del texto}
   TextAlto:Word;                                                                               {Alto del texto}
   CeldaAncho,                                                                              {Ancho de la celda}
   CeldaAlto,                                                                                {Alto de la celda}
   ColorText,                                                                          {Color de primer plano}
   ColorFondo,aux:Byte;                                                                          {Color de fondo}
   Texto:String;                                                                       {Datos de plano de texto}
   Terminador:Byte;                                                                           {Terminador, 0h}
End;

Ext_Aplica=Record                                                         {Bloque de extensión de la aplicaión}
 Introduc,                                                                                    {Introductor, 21h}
 Etiqueta,                                                                              {Etiqueta del bloque, FFh}
 Tam_Bloque:Byte;                                                                             {Tamaño del bloque}
 Identifica:Array [1..8] of Char;                                                   {Identificador de la aplicación}
 Autenticidad:Array [1..3] of byte;                                                          {Código de autenticidad}
 aux:byte;                                                                                     {Variable auxiliar}
 Aplicac:String;                                                                            {Datos de la aplicación}
 Terminador:Byte;                                                                             {Terminador, 0h}
End;
```

```
 Ext_Coment=Record                                          {Bloque de extensión de comentarios}
  Introduc,                                                                        {Introductor, 21h}
  Etiqueta,aux:Byte;                                                      {Etiqueta del bloque, FEh}
  Texto:String;                                                           {Datos de los comentarios}
  Terminador:Byte;                                                                 {Terminador, 0h}
 End;

Var
  Salte:Boolean;                                                                     {Variable auxiliar}
  Cont_lecturas,Val1,Val2:Byte;                          {Cuenta de bloques, introductor y etiqueta del bloque}
  Control:Ext_Control;                                   {Variable para almacenar datos de control de gráficos}
  Texto:Ext_Texto;                                        {Variable para almacenar datos de  plano de texto}
  Aplica:Ext_Aplica;                                   {Variable para almacenar datos del bloque de la aplicación}
  Coment:Ext_Coment;                                   {Variable para almacenar datos del bloque de comentarios}

 Procedure Lee_Ext_Control;                          {Prodecimiento para leer los datos del bloque de control de}
 Begin                                                                                          {gráficos}
  With Control do
   Begin
    Introduc:=Val1;
    Etiqueta:=Val2;
    BlockRead(Lee_arch,Tam_Bloque ,SizeOf(Tam_Bloque) ,SoloLee);
    BlockRead(Lee_arch,Paquete    ,SizeOf(Paquete)    ,SoloLee);
    BlockRead(Lee_arch,Espera     ,SizeOf(Espera)     ,SoloLee);
    BlockRead(Lee_arch,Color_index,SizeOf(Color_index),SoloLee);
    BlockRead(Lee_arch,Terminador ,SizeOf(Terminador) ,SoloLee);
   End;
   Gotoxy(01,16);Write('Extensión de Control de graficos presente');        {Solo marca que existe este bloque}
 End;

 Procedure Lee_Ext_Texto;                          {Procedimiento para leer los datos del bloque de plano de texto}
 Begin
  With Texto do
   Begin
    Introduc:=Val1;
    Etiqueta:=Val2;
    BlockRead(Lee_arch,Tam_Bloque,SizeOf(Tam_Bloque),SoloLee);
    BlockRead(Lee_arch,Textx     ,SizeOf(Textx)     ,SoloLee);
    BlockRead(Lee_arch,Texty     ,SizeOf(Texty)     ,SoloLee);
    BlockRead(Lee_arch,TextAncho ,SizeOf(TextAncho) ,SoloLee);
    BlockRead(Lee_arch,TextAlto  ,SizeOf(TextAlto)  ,SoloLee);
    BlockRead(Lee_arch,CeldaAncho,SizeOf(CeldaAncho),SoloLee);
    BlockRead(Lee_arch,CeldaALto ,SizeOf(CeldaAlto) ,SoloLee);
    BlockRead(Lee_arch,Colortext ,SizeOf(Colortext) ,SoloLee);
    BlockRead(Lee_arch,ColorFondo,SizeOf(Colorfondo),SoloLee);
    BlockRead(Lee_arch,Aux       ,sizeof(Aux)       ,SoloLee);
    BlockRead(Lee_arch,Texto     ,aux               ,SoloLee);
    BlockRead(Lee_arch,Terminador,SizeOf(Terminador),SoloLee);
   End;
   Gotoxy(01,17);Write('Extensión de texto presente ');                    {Solo marca que existe este bloque}
   Gotoxy(40,17);Write(texto.texto);
  End;
```

```
Procedure Lee_Ext_Aplica;                    {Procedimiento para leer los datos del bloque de la aplicación}
Var
 Textostr:String;
Begin
 With Aplica do
  Begin
    Introduc:=Val1;
    Etiqueta:=Val2;
    BlockRead(Lee_arch,Tam_Bloque  ,SizeOf(Tam_Bloque)   ,SoloLee);
    BlockRead(Lee_arch,Identifica  ,SizeOf(Identifica)   ,SoloLee);
    BlockRead(Lee_arch,Autenticidad,SizeOf(Autenticidad) ,SoloLee);
    BlockRead(Lee_arch,Aux         ,sizeof(Aux)          ,SoloLee);
    BlockRead(Lee_arch,aplicac     ,Aux                  ,SoloLee);
    BlockRead(Lee_arch,Terminador  ,SizeOf(Terminador)   ,SoloLee);
    textostr:=Identifica[1]+Identifica[2]+Identifica[3]+Identifica[4]+
              Identifica[5]+Identifica[6]+ Identifica[7]+ Identifica[8];
  End;
  Gotoxy(01,18);Write('Extensión de aplicación presente :');      {Solo marca que existe este bloque}
  Gotoxy(40,18);Write(textostr);
End;

Procedure Lee_Ext_Coment;                    {Procedimiento para leer los datos del bloque de comentarios}
Begin
 With Coment do
  Begin
    Introduc:=Val1;
    Etiqueta:=Val2;
    BlockRead(Lee_arch,Aux      ,sizeof(Aux)      ,SoloLee);
    BlockRead(Lee_arch,Texto    ,aux              ,SoloLee);
    BlockRead(Lee_arch,Terminador,SizeOf(Terminador),SoloLee);
  End;
  Gotoxy(01,19);Write('Extensión de comentarios ');               {Solo marca que existe este bloque}
  Gotoxy(40,18);Write(coment.texto);
End;

Function Lee_hasta_44(Val1,Val2:byte):Boolean;                    {Verifica si existen bloques de extensión}
Var
 Bueno:Boolean;                                                   {Bandera de descriptor de imagen encontrado ".'}
Begin
 Bueno:=False;
 If Val1=33 Then                                                  {Introductor de bloques de extensión, 21h}
  Case Val2 of
   1:                                                             {Existe bloque de plano de texto "1"}
     Begin
       Lee_Ext_Texto;
     End;
   249:                                                           {Existe bloque de control de gráficos  "F9"}
     Begin
       Lee_Ext_Control;
      End;
   255:                                                           {Existe bloque de la aplicacion "FF"}
     Begin
        Lee_Ext_Aplica;
      End;
```

```
    254:                                                        {Existe bloque de comentarios "FE"}
      Begin
        Lee_Ext_coment;
      End;
    End
  Else                                                          {Separador de datos, 44 (2Ch ó ",")}
    Begin
      GIFdi.Separador:=CHR(Val1);                               {Convierte a val1 en GIFdi.Separador}
      Blockread(lee_arch,Val1,1,sololee);
      GIFdi.xminimo:=(Val1 shl 8) and Val2;                     {Convierte a Val1 y Val2 en GIFdi.xminimo}
      Blockread(lee_arch,GIFdi.yminimo  ,2,sololee);
      Blockread(lee_arch,GIFdi.xmaximo  ,2,sololee);
      Blockread(lee_arch,GIFdi.ymaximo  ,2,sololee);
      Blockread(lee_arch,GIFdi.inter    ,1,sololee);
      Posicion:=Filepos(Lee_arch);                              {Despliega los datos del descriptor de imagen}
      Gotoxy(01,09);Write('Separador      :');
      Gotoxy(25,09);Writeln(GIFdi.Separador,'    ','Posición del puntero :',posicion);
      Gotoxy(01,10);Write('xminimo        :');
      Gotoxy(25,10);Writeln(GIFdi.xminimo);
      Gotoxy(01,11);Write('yminimo        :');
      Gotoxy(25,11);Writeln(GIFdi.yminimo);
      Gotoxy(01,12);Write('xmaximo        :');
      Gotoxy(25,12);Writeln(GIFdi.xmaximo);
      Gotoxy(01,13);Write('ymaximo        :');
      Gotoxy(25,13);Writeln(GIFdi.ymaximo);
      Gotoxy(01,14);Write('Interlazado    :');
      Gotoxy(25,14);Writeln(ifs(GIFdi.inter and 64=64,'Si','No'));
      Gotoxy(01,15);Write('Mapa Color local');
      Gotoxy(25,15);Writeln(ifs(GIFdi.inter and 128=128,'Si','No'));
      Bueno:=True;
  End;
  Lee_hasta_44:=Bueno;
End;
                                                                {Cuerpo principal de Lee_Cabecera_GIF}
Begin                                   {Despiega los datos de la cabecera y descriptor de pantalla lógico}
  Blockread(lee_arch,GIFdp,sizeof(GIFdp),sololee);
  Posicion:=Filepos(Lee_arch);
  Gotoxy(01,01);Write('Nombre de Archivo :',archivo);
  Gotoxy(01,02);Write('Identificacion :');
  Gotoxy(25,02);Writeln(GIFdp.GIF[0],GIFdp.GIF[1],GIFdp.GIF[2],'    ','Posición del puntero :',posicion);
  Gotoxy(01,03);Write('Versión        :');
  Gotoxy(25,03);Writeln(GIFdp.ver[0],GIFdp.ver[1],GIFdp.ver[2]);
  Gotoxy(01,04);Write('Total  en x    :');
  Gotoxy(25,04);Writeln(GIFdp.xtotal);
  Gotoxy(01,05);Write('Total  en y    :');
  Gotoxy(25,05);Writeln(GIFdp.ytotal);
  Gotoxy(01,06);Write('Bits por pixel :');
  Gotoxy(25,06);Writeln((GIFdp.describe and 7)+1);
  Gotoxy(01,07);Write('Fondo          :');
  Gotoxy(25,07);Writeln(GIFdp.fondo);
  Gotoxy(01,08);Write('Aspect ratio   :');
  Gotoxy(25,08);Writeln(Ifs(GIFdp.Aspect_ratio=0,'Si','No'));
  seek(Lee_arch,posicion+768);                                  {Coloca el puntero al final de la paleta}
  Cont_lecturas:=0;
```

```
  Repeat                                          {Repite el ciclo hasta encontrar al descriptor de imagen}
    BlockRead (lee_arch,Val1,1,sololee);
    BlockRead (lee_arch,Val2,1,sololee);
    Salte:=Lee_hasta_44(Val1,Val2);               {Verifica si hay bloques de extensión incluídos}
    inc(Cont_lecturas);
  Until (Salte) or (Cont_lecturas>4);
  If Cont_Lecturas>4 Then Writeln('Malo',Readkey);  {Verifica que no existan más de 4 bloques}
End;
Procedure Lee_Paleta_GIF;                         {Lee la paleta de la imagen y la almacena en un arreglo}
Begin
  seek(lee_arch,13);                              {Coloca el puntero al final de la cabecera}
  BlockRead(lee_arch,GIF_pal,sizeof(GIF_pal),sololee);
End;


Procedure Pon_Paleta_GIF;                         {Activa la paleta de colores de la imagen}
Var
 i:integer;
Begin
  For i:=0 to 255 do
    Cambia_color(i,GIF_pal[i].rojo shr 2,GIF_pal[i].verde shr 2,GIF_pal[i].azul shr 2);
End;


Procedure  Lee_imagen_GIF;                        {Descomprime los datos de imagen y los pone en pantalla}
Var                                               {Declaración de variables}
 prefijo:^Ar4096;                                 {Puntero de códigos almacenados}
 sufijo :^Ar4096;                                 {Puntero de códigos}
 sal_cod:^Ar1024;                                 {Puntero de códigos que se imprimen}
 i,j,k,                                           {Variables auxiliares}
  tam_codigo,                                     {Tamaño de código,  9 a 12}
  cod_limpia,                                     {{Código de limpieza,  256}
  cod_final,                                      {Código de fin  de lectura, 257}
  cod_inicial,                                    {Código inicial de lectura, 258}
  cont_cod,                                       {Cuenta de cada código}
  ini_codigo,                                     {Código inicial,  9}
  maxcod,                                         {Máximo bloque codificado}
  mascbit,                                        {Máscara de escritura}
  long_bloque,                                    {Longitud del  bloque de lectura }
  bits_ent,                                       {Bits de entrada en cada código}
  sal_cont,                                       {Cuenta de bytes de salida}
  codigo,                                         {Código de lectura}
  cont_num,                                       {Cuenta de datos del bloque }
  cod_actual,                                     {Código actual usado por sufijo}
  cod_anter,                                      {Código anterior usado por Prefijo}
  incod,                                          {Código auxiliar}
  final: Integer;                                 {Byte que será impreso}
  lee_byte:byte;                                  {Byte de lectura}


 Procedure Var_Inicial;                           {Inicialización de variables}
 Begin
  BlockRead(lee_arch,Lee_byte,1,sololee);         {Lee byte para inicializacion, para 256 colores el}
                                                  {primer byte de los datos de imagen tiene un valor de 8}
```

```
  tam_codigo:=lee_byte+1;                                        {Tamaño de código, 9 a 12}
  cod_limpia:=potencia(2,lee_byte);                              {Código de limpieza, 256}
  cod_final:=cod_limpia+1;                                       {Código de fin de lectura, 257}
  cod_inicial:=cod_limpia+2;                                     {Código inicial de lectura, 258}
  cont_cod:=cod_inicial;                                         {Cuenta de cada código}
  ini_codigo:=tam_codigo;                                        {Código inicial, generalmente 9}
  maxcod:=potencia(2,tam_codigo);                                {Máximo bloque codificado}
  mascbit:=potencia(2,(GIFdp.describe and 7)+1)-1;               {Máscara, 255 para 8 bits por pixel}
  BlockRead(Lee_arch,Lee_byte,1,sololee);                        {Lee byte de cuenta, 0 a 255}
  long_bloque:=lee_byte+1;                                       {Longitud del bloque de datos codificados}
  bits_ent:=8;                                                   {Bits de entrada en cada código}
  sal_cont:=0;                                                   {Cuenta de bytes de salida}
  cont_num :=0;                                                  {Cuenta de datos del bloque}
  codigo:=0;                                                     {Código de lectura}
  cod_actual:=0;                                                 {Código actual usado por suffijo}
  cod_anter:=0;                                                  {Código anterior usado por Prefijo}
  incod:=0;                                                      {Código auxiliar}
  final:=0;                                                      {Byte que será impreso}
  cont_imagen:=0;                                                {Cuenta de datos de imagen}
 End;


Function Lee_bit_x_bit:Integer;          {Lee bit a bit cada uno de los datos codificados, tomando 8 bits}
Const
 bitsaux:Array [0..8] of byte=(0,1,2,4,8,16,32,64,128);
Begin
 Inc(bits_ent);
 If bits_ent=9 Then                                  {Si son 9 bits lee un nuevo código y regresa a 1}
  Begin
   BlockRead(Lee_arch,lee_byte,1,sololee);                       {Lee siguiente código}
   bits_ent:=1;                                                  {Inicializa bits de entrada}
   inc(cont_num);                                                {Incrementa cuenta de datos del bloque}
   If cont_num=long_bloque Then                                  {Si se leyó todo el bloque, lee el siguiente bloque}
    Begin
     long_bloque:=lee_byte;                                      {Lee siguiente byte de cuenta, 0-255}
     BlockRead(Lee_arch,Lee_byte,1,sololee);                     {Lee el primer código del bloque}
     cont_num:=0;                                                {Reinicia el contador para el nuevo bloque}
    End;
  End;
  If (lee_byte and bitsaux[bits_ent]) =0 Then                    {Verifica con una máscara que dato desglosa}
      Lee_bit_x_bit:=0
  Else
      Lee_bit_x_bit:=1;
 End;

Function Lee_codigo_LZW(tama_cod:integer):Integer;               {Desglosa un código utilizando LZW }
Var                                                              {de 9 a 12 bits}
  ii,codigo_aux:Integer;
Begin
   codigo_aux:=0;
   For ii:=0 to (tama_cod - 1) do                                {Obtiene el código por medio de cada bit}
     codigo_aux:=codigo_aux + (Lee_bit_x_bit * potencia(2,ii));
   Lee_codigo_LZW:=codigo_aux;                                   {Devuelve el dato decodificado}
 End;
```

```
Procedure  Pon_pixel(color:byte);                                   {Pone un pixel en pantalla}
Var
  ponx,pony:Integer;
Begin
  ponx:=(cont_imagen mod (GIFdi.xmaximo));                          {Calcula la posición X del pixel}
  pony:=(cont_imagen div (GIFdi.xmaximo));                          {Calcula la posición Y del pixel}
  If (con_imagen<=(Tam_imagen-1)) Then
    putpixel(ponx,pony,Color);                                      {Pone en pantalla el dato del pixel}
  inc(cont_imagen);                                                 {Incrementa cuenta de datos de imagen decodificados}
End;


Begin                                                               {Cuerpo principal de lee_imagen_GIF}
 Tam_imagen:=gifdp.xtotal*gifdp.ytotal;                             {Calcula tamaño de la imagen}
 new(sufijo);                                                       {Asignación de memoria}
 new(prefijo); new(sal_cod);
 Seek(lee_arch,posicion);                                           {Coloca el puntero en la posición de los datos de imagen}
 Var_inicial;                                                       {Inicializa variables}
 Repeat
   codigo:=Lee_codigo_LZW(tam_codigo);                              {Decodifica un código utilizando LZW}
   If codigo<>cod_final Then                                        {Si  no es código Final}
    Begin
     If codigo=cod_limpia Then                                      {Si es código de limpieza, 256}
      Begin                                                         {Reinicia  variables}
        tam_codigo:=ini_codigo;                                     {Vuelve al tamaño inicial, 9}
        maxcod:=potencia(2,tam_codigo);                             {Vuelve al tamaño inicial, 256}
        cont_cod:=cod_inicial;                                      {Vuelve a cuenta de códigos inicial}
        codigo:=lee_codigo_LZW(tam_codigo);                         {Decodifica un código utilizando LZW }
        cod_actual:=codigo;                                         {Asigna el valor del código}
        cod_anter:=codigo;                                          {Asigna el valor del código}
        final:=codigo and mascBit;                                  {Asigna el valor que será impreso, tomando solo 8 bits}
        Pon_pixel(final);                                           {Pone el pixel en pantalla}
     End
    Else                                                            {Si no es código de limpieza}
     Begin
      cod_actual:=codigo;                                           {Asigna el valor del código para la concatenación}
      incod:=codigo;                                                {Asigna el valor del código}
      If (codigo>=cont_cod) Then                                    {Si es mayor o igual a cuenta de cada código}
      Begin
        cod_actual:=cod_anter;                                      {Asigna el valor del dato decodificado}
        sal_cod^[sal_cont] := final;                                {Almacena el dato del pixel en los datos de salida}
        inc(sal_cont);                                              {Incrementa cuenta de codigos de salida}
      End;
     If (cod_actual>mascbit)  Then                                  {Si código es mayor a 255}
      Repeat                                                        {Realiza una regresión sobre los bytes  almacenados en el arreglo de}
        sal_cod^[sal_cont] := Sufijo^[cod_actual];                  {sufijo tomando los que están en prefijo}
         inc(sal_cont);                                             {En este bloque es donde se toman todos los }
        cod_actual:=Prefijo^[cod_actual];                           {datos de salida "decompresión"}
      Until (cod_actual<=mascbit);                                  {Hasta que código sea menor o igual a 255}
     final:=cod_actual and mascbit ;                                {Asigna el valor que será impreso, tomando solo 8 bits}
     sal_cod^[sal_cont]:=final;                                     {Almacena el dato del pixel en los datos de salida}
     inc(sal_cont);                                                 {Incrementa la cuenta de datos de salida}
     for i:= (Sal_cont-1) downto 0 do                               {Coloca los pixeles que se han almacenado}
       pon_pixel(sal_cod^[i]);
```

```
    sal_cont:=0;                                  {Reinicia la cuenta de datos de salida}
    Prefijo^[cont_cod]:=cod_anter;                {Actualiza prefijo}
    Sufijo ^[cont_cod]:=final;                    {Actualiza sufijo}
    cod_anter:=incod;                             {Actualiza código anterior}
    inc(cont_cod);                                {Incrementa cuenta de cada código}
    If (cont_cod>=maxcod) Then                    {Cuenta de cada código es mayor o igual al máximo bloque}
      If (tam_codigo <12) Then                    {Si los bits de codigo no rebasan los 12 bits }
      Begin
        inc(tam_codigo);                          {Incrementa tamaño de código}
        maxcod:=maxcod*2;                         {Actualiza máximo bloque para codificar}
      End;
   End;
  End;
 Until (codigo=cod_final);                        {Hasta encontrar el código de fin de archivo}
 Dispose(sufijo);                                 {Libera memoria}
 Dispose(prefijo);
 Dispose(Sal_cod);
End;


Begin                                             {Cuerpo del bloque principal del programa}
 clrscr;                                          {Limpia pantalla}
 archivo:='Anima1.GIF';                           {Asigna el nombre del archivo}
 If Abre_lectura_GIF(Archivo) Then                {Si puede abrir el archivo continua con el programa}
 Begin
    Lee_Cabecera_GIF;                             {Lee y poneen pantalla la informacion de la cabecera}
    Readkey;                                      {Pausa para tomar nota de la cabecera}
    Lee_Paleta_GIF;                               {Lee los datos de la paleta}
    Inicializa(16,4);                             {Inicializa el modo gráfico, 1024x768x256}
    Pon_Paleta_GIF;                               {Activa la paleta leida del archivo}
    Lee_imagen_GIF;                               {Descomprime la imagen y la visualiza}
    Readkey;                                      {Pausa para ver la imagen}
    close(Lee_arch);                              {Cierra el archivo}
    Restorecrtmode;                               {Recupera el modo texto}
    closegraph;                                   {Cierra libreria de gráficos}
    clrscr;                                       {Limpia pantalla}
  End;
 End.
```

Una de las imágenes procesadas con el programa es: anima1.gif que es un archivo con imágenes multiples (Animado).

Anima1.gif

El programa da como resultado los siguientes datos:

```
Nombre de Archivo :anima1.GIF
Identificacion :        GIF      Posición del puntero :13
Versión        :        89a
Total  en x    :        65
Total  en y    :        128
Bits por pixel :        8
Fondo          :        38
Color Global   :        Si
Separador      :        ,        Posición del puntero :818
xminimo        :        0
yminimo        :        0
xmaximo        :        65
ymaximo        :        128
Interlazado    :        No
Mapa de Color Local     No
Extensión de Control de graficos presente

Extensión de aplicación presente :      NETSCAPE
```

Posteriormente se despliega la imagen.

IV.6.2 En Lenguaje C

En C el programa se codifica de la siguiente manera.

```
//Programa que lee el formato de archivo GIF.
 Realizado para el trabajo de tesis titulado:FORMATOS GRÁFICOS
 Programadores :
                Cantero Ramírez Carlos
                Trejo Bonaga Marilu
 Asesores :
                Manzo Salazar Itsmael
                Monterrosa Escobar Amilcar A.

#include <stdio.h>                                  // Bloque de unidades que necesita C
#include <stdlib.h>
#include <graphics.h>
#include <dos.h>
#include <conio.h>
#include <io.h>
#include <math.h>

#define BYTE  unsigned char                         // Definición de constantes simbólicas
#define True  1
#define False 0
#define LONGINT unsigned long
#define WORD    unsigned int
                                                    // Definición de las estructuras principales
typedef struct                                      // Cabecera y descriptor de pantalla lógico
 {  char GIF[3],ver[3];                             // Identificador (GIF) y versión ("87a" ó "89a"
    WORD xtotal,ytotal;                             // Ancho y largo de la pantalla en píxeles
    BYTE describe,fondo,aspect_ratio;               // Paquete, color de fondo y aspect ratio
 }descriptor_de_pantalla;

typedef struct                                      // Paleta de colores global
 {
   BYTE  rojo,verde,azul;
 }paleta_GIF;

typedef struct                                      // Descriptor de pantalla local
{                                                   // Siempre 44=","
   char separador;
   WORD xminimo,yminimo,xmaximo,ymaximo;            // Coordenadas de inicio y fin de la imagen
   BYTE inter;                                      // Variable de interlazado
}descriptor_de_imagen;

fpos_t posicion;                                    // Variable auxiliar
char *archivo;                                      // Variable para elnombre del archivo a leer
LONGINT cont_imagen,tam_imagen;                     // Cuenta de datos de la imagen
int i=0,j,k;                                        // Variables auxiliares
paleta_GIF GIF_pal[256];                            // Variable para almacenar paleta global
descriptor_de_pantalla GIFdp;                       // Descriptor de la pantalla
descriptor_de_imagen GIFdi;                         // Descriptor de la Imagen
FILE *lee_arch;                                     // Archivo de lectura
```

```
void Lee_Cabecera_GIF(void);                                   // Funciones utilizadas en el programa
int Inicializa(int Driver, int Modo);
int Abre_lectura_GIF(char *arch_imagen);
void Lee_Paleta_GIF(void);
void Pon_Paleta_GIF();
void Lee_imagen_GIF();
void Cambia_color(BYTE color,BYTE r,BYTE v,BYTE a);


void main ()                                                   // Cuerpo del bloque principal del programa
{                                                              // Limpia pantalla
  clrscr();
 archivo="87amk.GIF";                                          // Asigna el nombre del archivo
  if (Abre_lectura_GIF(archivo))                               // Si puede abrir el archivo continua con el programa
  {                                                            // Lee y pone en pantalla la información de la cabecera
    Lee_Cabecera_GIF();
    getch();                                                   // Pausa para tomar nota de la cabecera
    Lee_Paleta_GIF();                                          // Lee los datos de la paleta
    Inicializa(16,4);                                          // Inicializa el modo gráfico. 1024x768x256
    Pon_Paleta_GIF();                                          // Activa la paleta leída del archivo
    Lee_imagen_GIF();                                          // Descomprime la imagen y la visualiza
    getch();                                                   // Pausa para ver la imagen
    fclose(lee_arch);                                          // Cierra el archivo
    restorecrtmode();                                          // Recupera el modo texto
    closegraph();                                              // Cierra librería de gráficos
    clrscr();                                                  // Limpia pantalla
  }
}

                                                               // Cuerpo de cada una de las funciones declaradas
int Inicializa(int Driver, int Modo)                           // Inicializar gráficos y reportar los errores que ocurran
{                                                              // Esta función no necesita explicación
  int ErrorG;
  unsigned int x,y;
  union REGS regin,regout;
  if (Driver==16)
    Driver = installuserdriver("Svga256",NULL);
  if (Driver==grError) exit;
  initgraph( &Driver, &Modo,"d:\borlandc\programa\exeobj");
  regin.x.ax=0x000f;
  regin.x.cx=0x8;
  regin.x.dx=0x8;
  int86(0x33,&regin,&regout);
  x=getmaxx();
  y=getmaxy();
  regin.x.ax=x;
  regin.x.dx=regin.x.ax;
  regin.x.ax=0x0007;
  regin.x.cx=0x0;
  int86(0x33,&regin,&regout);
  regin.x.ax=y;
  regin.x.dx=regin.x.ax;
  regin.x.ax=0x0008;
  regin.x.cx=0x0;
```

```
  int86(0x33,&regin,&regout);
  ErrorG=graphresult();
  if (ErrorG!=grOk)
 {
   printf("\n Error en graficos : %s",grapherrormsg(ErrorG));
   restorecrtmode;
   return 0;
  }
  else
 {
   cleardevice;
   return 1;
  }
}


void Cambia_color(BYTE color,BYTE r,BYTE v,BYTE a)          // Modifica la paleta de activa
{                                                          // Esta función no necesita explicación
 asm mov dx,3C8h
 asm mov al,color
 asm out dx,al
 asm inc dx
 asm mov al,r
 asm out dx,al
 asm mov al,v
 asm out dx,al
 asm mov al,a
 asm out dx,al
}


int Abre_lectura_GIF(char *arch_imagen)                    // Prepara al archivo para  su lectura
{                                                          // Esta función no necesita explicación
 lee_arch=fopen(arch_imagen,"rb");
 if (lee_arch==NULL)
  {return 0;}
 else
  {return 1;}
}

                    // Declaración de las estructuras y variables utilizadas en Lee_cabecera_GIF

typedef struct                                             // Bloque de control de gráficos
 {                                                         // Introductor, 21h
  BYTE introduc,
          etiqueta,                                        // Etiqueta del bloque
          tam_bloque,                                      // Tamaño del bloque
          paquete;                                         // Método de colocación de gráficos
  WORD  espera;                                            // Tiempo de espera
  BYTE   color_index,                                      // Indice de color transparente
          terminador;                                      // Terminador, 0h
 }ext_control;
```

```
typedef struct                                  // Bloque de extensión de plano de texto
 {                                                                  // Introductor, 21h
  BYTE  introduc,
        etiqueta,                                                   // Etiqueta del bloque, 01h
        tam_bloque;                                                  // Tamaño del bloque
  WORD textx,                                                       // Posición en X del texto
        texty,                                                      // Posición en Y del tecto
        textancho,                                                   // Ancho del texto
        textalto;                                                     // Alto del texto
  BYTE  celdaancho,                                                  // Ancho de la celda
        celdaalto,                                                    // Alto de la celda
        colortext,                                                  // Color de primer plano
        colorfondo,aux;                                              // Color de fondo
   char  *texto;                                                   // Datos de plano de texto
  BYTE  terminador;                                                  // Terminador, 0h
}ext_texto;


typedef struct                                  // Bloque de extensión de la aplicación
 {                                                                  // Introductor, 21h
  BYTE introduc,
        etiqueta,                                                   // Etiqueta del bloque, Ffh
        tam_bloque;                                                  // Tamaño del bloque
    char identifica[8];                                            // Identificador de la aplicación
  BYTE autenticidad[3];                                            // Código de autenticidad
  BYTE aux;                                                         // Variable auxiliar
  char *aplicac;                                                   // Datos de la aplicación
  BYTE terminador;                                                  // Terminador
}ext_aplica;


typedef struct                                  // Bloque de extensión de comentarios
 {                                                                  // Introductor
  BYTE introduc,
        etiqueta,aux;                                               // Etiqueta del bloque, FEh
    char *texto;                                                   // Datos de los comentarios
  BYTE terminador;                                                  // Terminador
}ext_coment;


BYTE salte,                                                         // Variable auxiliar
      cont_lecturas;                                                // Cuenta de bloques
int val1, val2;                                   // Variable para el introductor y etiqueta del bloque
ext_control control;                   // Variable para almacenar datos del bloque  control de gráficos
ext_texto  texto;                          // Variable para almacenar datos del bloque plano de texto
ext_aplica  aplica;                       // Variable para almacenar datos del bloque de la aplicación
ext_coment  coment;                       // Variable para almacenar datos del bloque de comentarios


void Lee_Cabecera_GIF()                           // Lee los datos de la cabecera y  descriptor depantalla
{                                                                   // Declaración de las funciones utilizadas
 int lee_hasta_44(BYTE val1,BYTE val2);
 void lee_ext_control();
 void lee_ext_texto();
```

```
  void lee_ext_aplica();
  void lee_ext_coment();
                                                    // Despliega los datos de la cabecera y descriptor de pantalla
 fread((BYTE *)&GIFdp,1,sizeof(GIFdp),lee_arch);
 fgetpos(lee_arch,&posicion);
 gotoxy( 1, 1);printf("Nombre de Archivo :%s",archivo);
 gotoxy( 1, 2);printf("Identificacion : %c%c%c    Posición del puntero :%d"          , GIFdp.GIF[0],
                                                                  GIFdp.GIF[1],GIFdp.GIF[2], posicion);
 gotoxy( 1, 3);printf("Versión        : %c%c%c "  ,GIFdp.ver[0], GIFdp.ver[1], GIFdp.ver[2]);
 gotoxy( 1, 4);printf("Total  en x    : %u ",GIFdp.xtotal);
 gotoxy( 1, 5);printf("Total  en y    : %u ",GIFdp.ytotal);
 gotoxy( 1, 6);printf("Bits por pixel : %d ",(GIFdp.describe & 7)+1);
 gotoxy( 1, 7);printf("Fondo          : %d ",GIFdp.fondo);
 gotoxy( 1, 8);printf("Aspect ratio (0=si, otro=no) : %d ",GIFdp.aspect_ratio);
 fseek(lee_arch,posicion+768,SEEK_SET);                          // Coloca el puntero al final de la paleta
 cont_lecturas=0;
 do
  {
    fread((BYTE *)&val1,1,1,lee_arch);
    fread((BYTE *)&val2,1,1,lee_arch);
    salte=lee_hasta_44(val1,val2);                          // Verifica si hay bloques de extensión incluidos
    cont_lecturas++;
  } while ((salte==0)&&(cont_lecturas<4));                  // Repite hasta encontrar al descriptor de imagen
 if (cont_lecturas>4) printf("\n GIF Erroneo ");                  // Verifica no existan más de 4 bloques
 fgetpos(lee_arch,&posicion);
}

                                                                    // Funciones utilizadas en Lee_hasta_44

void lee_ext_control()                                      // Lee los datos del bloque de control de gráficos
 {
  control.introduc=val1;
  control.etiqueta=val2;
  fread((BYTE *)&control.tam_bloque ,1,sizeof(control.tam_bloque ),lee_arch);
  fread((BYTE *)&control.paquete    ,1,sizeof(control.paquete)    ,lee_arch);
  fread((WORD *)&control.espera     ,1,sizeof(control.espera)     ,lee_arch);
  fread((BYTE *)&control.color_index,1,sizeof(control.color_index),lee_arch);
  fread((BYTE *)&control.terminador ,1,sizeof(control.terminador ),lee_arch);
  gotoxy(1,16);printf("Extensión de Control de graficos presente");
}


void lee_ext_texto()                                        // Lee los datos del bloque de plano de texto
{
  texto.introduc=val1;
  texto.etiqueta=val2;
  fread((BYTE *)&texto.tam_bloque,1,sizeof(texto.tam_bloque),lee_arch);
  fread((WORD *)&texto.textx     ,1,sizeof(texto.textx)     ,lee_arch);
  fread((WORD *)&texto.texty     ,1,sizeof(texto.texty)     ,lee_arch);
  fread((WORD *)&texto.textancho ,1,sizeof(texto.textancho) ,lee_arch);
  fread((WORD *)&texto.textalto  ,1,sizeof(texto.textalto)  ,lee_arch);
  fread((BYTE *)&texto.celdaancho,1,sizeof(texto.celdaancho),lee_arch);
  fread((BYTE *)&texto.celdaalto ,1,sizeof(texto.celdaalto) ,lee_arch);
  fread((BYTE *)&texto.colortext ,1,sizeof(texto.colortext) ,lee_arch);
```

```
        fread((BYTE *)&texto.colorfondo,1,sizeof(texto.colorfondo),lee_arch);
        fread((BYTE *)&texto.aux      ,1,sizeof(texto.aux)      ,lee_arch);
        fread((char *)&texto.texto    ,1,texto.aux              ,lee_arch);
        fread((BYTE *)&texto.terminador,1,sizeof(texto.terminador),lee_arch);
       gotoxy( 1,17);printf("Extensión de texto presente %s",texto.texto);
     }


    void lee_ext_aplica()                                          // Lee los datos del bloque de la apicación
    {  aplica.introduc=val1;
       aplica.etiqueta=val2;
       fread((BYTE *)&aplica.tam_bloque  ,1,sizeof(aplica.tam_bloque)  ,lee_arch);
       fread((char *)&aplica.identifica  ,1,sizeof(aplica.identifica)  ,lee_arch);
       fread((char *)&aplica.autenticidad,1,sizeof(aplica.autenticidad) ,lee_arch);
       fread((BYTE *)&aplica.aux         ,1,sizeof(aplica.aux)         ,lee_arch);
       fread((char *)&aplica.aplicac     ,1,aplica.aux                 ,lee_arch);
       fread((BYTE *)&aplica.terminador  ,1,sizeof(aplica.terminador)  ,lee_arch);
       gotoxy(1,18);printf("Extensión de aplicación presente : %c%c%c%c%c%c%c%c",
                           aplica.identifica[0],aplica.identifica[1], aplica.identifica[2],aplica.identifica[3],
                           aplica.identifica[4],aplica.identifica[5], aplica.identifica[6],aplica.identifica[7]);
    }


   void lee_ext_coment()                                           // Lee los datos del bloque de comentarios
   {
     coment.introduc=val1;
     coment.etiqueta=val2;
     fread((BYTE *)&coment.aux      ,1,sizeof(coment.aux)      ,lee_arch);
     fread((char *)&coment.texto    ,1,coment.aux              ,lee_arch);
     fread((BYTE *)&coment.terminador,1,sizeof(coment.terminador),lee_arch);
   gotoxy(01,19);printf("Extensión de comentarios %s",coment.texto);
  }


int lee_hasta_44(BYTE val1,BYTE val2)                              // Busca el introductor de datos de imagen
{                                                                                          // Variable auxiliar
  int bueno;
  bueno=False;                                                    // Bandera de datos de imagen encontrados
  if (val1==33)                                                    // Introductor de bloques de extensión, 21h
  {
    switch (val2)
    {
     case 1:                                                       // Existe bloque de Plano de  Texto  "1"
          lee_ext_texto();
          break;
    case 249:                                                     // Existe bloque de control de gráficos "F9"
          lee_ext_control();
          break;
   case 255:                                                      // Existe bloque de la aplicacion "FF"
          lee_ext_aplica();
          break;
   case 254:                                                      // Existe bloque de comentarios "FE"
          lee_ext_coment();
          break;
```

```
  fread((BYTE *)&texto.colorfondo,1,sizeof(texto.colorfondo),lee_arch);
  fread((BYTE *)&texto.aux      ,1,sizeof(texto.aux)      ,lee_arch);
  fread((char *)&texto.texto    ,1,texto.aux              ,lee_arch);
  fread((BYTE *)&texto.terminador,1,sizeof(texto.terminador),lee_arch);
 gotoxy( 1,17);printf("Extensión de texto presente %s",texto.texto);
}


void lee_ext_aplica()                                          // Lee los datos del bloque de la apicación
{ aplica.introduc=val1;
  aplica.etiqueta=val2;
  fread((BYTE *)&aplica.tam_bloque  ,1,sizeof(aplica.tam_bloque)  ,lee_arch);
  fread((char *)&aplica.identifica  ,1,sizeof(aplica.identifica)  ,lee_arch);
  fread((char *)&aplica.autenticidad,1,sizeof(aplica.autenticidad) ,lee_arch);
  fread((BYTE *)&aplica.aux         ,1,sizeof(aplica.aux)         ,lee_arch);
  fread((char *)&aplica.aplicac     ,1,aplica.aux                 ,lee_arch);
  fread((BYTE *)&aplica.terminador  ,1,sizeof(aplica.terminador)  ,lee_arch);
  gotoxy(1,18);printf("Extensión de aplicación presente : %c%c%c%c%c%c%c%c%c",
                      aplica.identifica[0],aplica.identifica[1], aplica.identifica[2],aplica.identifica[3],
                      aplica.identifica[4],aplica.identifica[5], aplica.identifica[6],aplica.identifica[7]);
}


void lee_ext_coment()                                          // Lee los datos del bloque de comentarios
{
  coment.introduc=val1;
  coment.etiqueta=val2;
  fread((BYTE *)&coment.aux      ,1,sizeof(coment.aux)      ,lee_arch);
  fread((char *)&coment.texto    ,1,coment.aux              ,lee_arch);
  fread((BYTE *)&coment.terminador,1,sizeof(coment.terminador),lee_arch);
 gotoxy(01,19);printf("Extensión de comentarios %s",coment.texto);
}


int lee_hasta_44(BYTE val1,BYTE val2)                          // Busca el introductor de datos de imagen
{                                                              // Variable auxiliar
  int bueno;
  bueno=False;                                                 // Bandera de datos de imagen encontrados
  if (val1==33)                                                // Introductor de bloques de extensión, 21h
  {
    switch (val2)
    {
     case 1:                                                   // Existe bloque de Plano de  Texto  "1"
          lee_ext_texto();
          break;
     case 249:                                                 // Existe bloque de control de gráficos "F9"
          lee_ext_control();
          break;
     case 255:                                                 // Existe bloque de la aplicacion "FF"
          lee_ext_aplica();
          break;
     case 254:                                                 // Existe bloque de comentarios "FE"
          lee_ext_coment();
          break;
```

```
    }
   }
  else                                                    // Encontro el separador de datos de imagen
   {                                                      // Despliega los datos del descriptor de imagen
     GIFdi.separador=val1;
     fread((BYTE *)&val1 ,1,sizeof(val1)   ,lee_arch);
     GIFdi.xminimo=(val1 << 8) & val2;
     fread((WORD *)&GIFdi.yminimo ,1,sizeof(GIFdi.yminimo)   ,lee_arch);
     fread((WORD *)&GIFdi.xmaximo ,1,sizeof(GIFdi.xmaximo)   ,lee_arch);
     fread((WORD *)&GIFdi.ymaximo ,1,sizeof(GIFdi.ymaximo)   ,lee_arch);
     fread((BYTE *)&GIFdi.inter   ,1,sizeof(GIFdi.inter)   ,lee_arch);
     fgetpos(lee_arch,&posicion);
     gotoxy(1,9);printf("Separador       : %c      Posición del puntero :%d",GIFdi.separador,posicion);
     gotoxy(1,10);printf("xminimo        : %d",GIFdi.xminimo);
     gotoxy(1,11);printf("yminimo        : %d",GIFdi.yminimo);
     gotoxy(1,12);printf("xmaximo        : %d",GIFdi.xmaximo);
     gotoxy(1,13);printf("ymaximo        : %d",GIFdi.ymaximo);
     gotoxy(1,14);printf("Interlazado    (1=si, 0=no): %d",GIFdi.inter & 64);
     gotoxy(1,15);printf("Mapa de Color Local (1=si, 0=no): %d",GIFdi.inter & 128);
     bueno=True;
   } return bueno;
}


void Lee_Paleta_GIF()                                     // Lee la paleta de la imagen y la almacena en un arreglo
{
  fseek(lee_arch,13,SEEK_SET);                            // Coloca el puntero al final de la cabecera
  fread((BYTE *)&GIF_pal ,1,sizeof(GIF_pal)   ,lee_arch);
}


void Pon_Paleta_GIF()                                     // Activa la paleta de colores de la imagen
{
 int i;
   for (i=0;i<=255;i++)
    Cambia_color(i,GIF_pal[i].rojo >> 2,GIF_pal[i].verde >> 2,GIF_pal[i].azul >> 2);
}

                                                          // Declaración de las variables utilizadas en Lee_imagen_GIF

int prefijo[4096];                                        // Arreglo de códigos almacenados
int sufijo [4096];                                        // Arreglo de códigos
int sal_cod[1024];                                        // Arreglo de códigos que se imprimen
int tam_codigo,                                           // Tamaño de código,  9 a 12
 cod_limpia,                                              // Código de limpieza, 256
 cod_final,                                               // Código de fin de lectura,  257
 cod_inicial,                                             // Código inicial de lectura, 258
 cont_cod,                                                // Cuenta de cada código
 ini_codigo,                                              // Código inicial, 9
 maxcod,                                                  // Máximo bloque codificado
 mascbit,                                                 // Máscara de escritura
 long_bloque,                                             // Longitud del  bloque de lectura
 bits_ent,                                                // Bits de entrada en cada código
 sal_cont,                                                // Cuenta de bytes de salida
```

```
 codigo,                                                    // Código de lectura
 cont_num,                                                  // Cuenta de datos del bloque
 cod_actual,                                                // Código actual usado por sufijo
 cod_anter,                                                 // Código anterior usado por prefijo
 incod,                                                     // Código auxiliar
 final;                                                     // Byte que será impreso
 lee_byte;                                                  // Byte de lectura

void Lee_imagen_GIF()                                       // Descomprime los datos de imagen y los pone en pantalla
{                                                           // Declaración de las funciones utilizadas en esta función
  void var_inicial();                                       // Inicializa variables
  int lee_bit_x_bit();                                      // Lee bit x bit cada uno de los códigos
  int lee_codigo_LZW(int tama_cod);                         // Desglosa un código utilizando LZW
  void pon_pixel(int color);                                // Pone el pixel en pantalla
  tam_imagen=gifdp.xtotal*gifdp.ytotal;                     // Calcula tamaño de la imagen
  fseek(lee_arch,posicion,SEEK_SET);                        // Coloca el puntero en la posición de los datos de imagen
  var_inicial();                                            // Inicializa variables
  do
  {
   codigo=lee_codigo_LZW(tam_codigo);                       // Decodifica un código utilizando LZW
   if (codigo!=cod_final)                                   // Si no es código Final
    {
      if (codigo==cod_limpia)                               // Si es código de limpieza, 256
        {                                                   // Reinicia variables
          tam_codigo=ini_codigo;                            // Vuelve al tamaño inicial, 9
          maxcod=pow(2,tam_codigo);                         // Vuelve al tamaño inicial, 256
          cont_cod=cod_inicial;                             // Vuelve a cuenta de códigos inicial
          codigo=lee_codigo_LZW(tam_codigo);                // Decodifica un código utilizando LZW
          cod_actual=codigo;                                // Asigna el valor del código
          cod_anter=codigo;                                 // Asigna el valor del código
          final=codigo & mascbit;                           // Asigna el valor que será impreso tomando solo 8 bits
          pon_pixel(final);                                 // Pone el pixel en pantalla
        }
      else                                                  // Si no es código de limpieza
        { cod_actual=codigo;                                // Asigna el valor del código para la concatenación
          incod    =codigo;                                 // Asigna el valor del código
          if (codigo>=cont_cod)                             // Si es mayor o igual a cuenta de cada código
           { cod_actual=cod_anter;                          // Asigna el valor del dato decodificado
             sal_cod[sal_cont] = final;                     // Asigna el valor del dato decodificado
             sal_cont++;                                    // Incrementa la cuenta de códigos de salida
           }
          if (cod_actual>mascbit)                           // Si código es mayor a 255
           do
           {                    // Realiza una regresion sobre los bytes que estan almacenados en el arreglo
            sal_cod[sal_cont] = sufijo[cod_actual];         // de Sufijo tomando los que estan en prefijo
            sal_cont++;                                     // En este bloque es donde se toman todos los
            cod_actual=prefijo[cod_actual];                 // datos de salida "decompresión"
           } while (cod_actual>mascbit);                    // Repite mientras cod_actual sea mayor a 255
          final=cod_actual & mascbit;                       // Asigna el valor que será impreso tomando solo 8 bits
          sal_cod[sal_cont]=final;                          // Almacena el dato del pixel en los datos de salida
          sal_cont++;                                       // Coloca los pixeles que se han almacenado
          for (i=sal_cont-1;i>=0;i--)
          {
```

```
            pon_pixel(sal_cod[i]);
           }
           sal_cont=0;                                   // Reinicia la cuenta de datos de salida
           prefijo[cont_cod]=cod_anter;                  // Actualiza prefijo
           sufijo [cont_cod]=final;                      // Actualiza sufijo
           cod_anter=incod;                              // Actualiza código anterior
           cont_cod++;                                   // Incrementa cuenta de cada código
          if (cont_cod>=maxcod)                          // Cuenta de cada código es mayo o igual al máximo bloque
             if (tam_codigo<12)                          // Si los bits de codigo no rebasan los 12 bits
             {
               tam_codigo++;                             // Incrementa tamaño de código
               maxcod=maxcod*2;                          // Actualiza máximo bloque codificado
             }
         }
     }
   } while(codigo!=cod_final);                           // Repite mientras no sea fin de archivo
}

                                                         // Cuerpo de las funciones utilizadas en Lee_imagen_GIF

void var_inicial()                                       // Inicializa variables
 { fread((BYTE *)&lee_byte,1,1,lee_arch);                // Lee byte para inicialización, para 256 colores el primer
                                                         // byte de los datos de imagen tiene un valor de 8
   tam_codigo =lee_byte+1;                               // Tamaño de código, 9 a 12
   cod_limpia =pow(2,lee_byte);                          // Código de limpieza,  256
   cod_final  =cod_limpia+1;                             // Código de fin de lectura, 257
   cod_inicial=cod_limpia+2;                             // Código inicial de lectura, 258
   cont_cod   =cod_inicial;                              // Contador de cada código
   ini_codigo =tam_codigo;                               // Código inicial, generalmente 9
   maxcod     =pow(2,tam_codigo);                        // Máximo bloque codificado
   mascbit    =pow(2,(GIFdp.describe & 7)+1) -1;         // Máscara, 255 para 8 bits por pixel
   fread((BYTE *)&lee_byte,1,1,lee_arch);                // Lee byte de cuenta, 0 a 255
   long_bloque=lee_byte+1;                               // Longitud de un bloque de datos codificados
   bits_ent   =8;                                        // Bits de entrada en cada código
   sal_cont   =0;                                        // Cuenta de bytes de salida
   cont_num   =0;                                        // Cuenta de datos del bloque
   codigo     =0;                                        // Código de lectura
   cod_actual =0;                                        // Código actual usado por suffijo
   cod_anter  =0;                                        // Código anterior usado por prefijo
   incod      =0;                                        // Código auxiliar
   final      =0;                                        // Byte que sera impreso
   cont_imagen=0;                                        // Cuenta de datos de imagen
 }


int lee_bit_x_bit()                                      // Lee bit a bit cada uno de los datos codificados, tomando 8 bits
   int  bitsaux[9]={0,1,2,4,8,16,32,64,128};
   int Pot;
   bits_ent++;
   if (bits_ent==9)                                      // Si son 9 bits lee un nuevo códigoy regresa a 1
    {
     fread((BYTE *)&lee_byte ,1,1,lee_arch);             // Lee sigueinte código
     bits_ent=1;                                         // Inicializa bits de entrada
     cont_num++;                                         // Incrementa cuenta de datos del bloque
```

```
   if (cont_num==long_bloque)                    // Si se leyó todo el bloque, lee el siguiente bloque de datos
     {
      long_bloque=lee_byte;                                                    // Lee siguiente byte de cuenta
           fread((BYTE *)&lee_byte ,1,1,lee_arch);                            // Lee el primer código del bloque
           cont_num=0;                                                     // Reinicia cuenta para el nuevo bloque
          }
     }
   if ((lee_byte & bitsaux[bits_ent]) ==0)                          // Verifica con una máscara que dato desglosa
     return 0;
   else return 1;
 }


int lee_codigo_LZW(int tama_cod)                          // Desglosa un código utilizando LZW de 9 a 12 bits
{  int ii,codigo_aux;
   codigo_aux=0;
   for (ii=0;ii<=(tama_cod - 1);ii++)                                   // Obtiene el código por medio de cada bit
     codigo_aux=codigo_aux + (lee_bit_x_bit() * pow(2,ii));                          // Devuelve el dato decodificado
   return codigo_aux;
 }


void  pon_pixel(int color)                                                    // Pone un dato de pixel en pantalla
{ int ponx,pony;
  ponx=(cont_imagen % (GIFdi.xmaximo));                                          // Calcula la posición X del pixel
  pony=(cont_imagen / (GIFdi.xmaximo));                                          // Calcula la posición Y del pixel
 if (con_imagen<=(tam_imagen-1))
    putpixel(ponx,pony,color);                                                  // Pone en pantalla el dato del
                                                                                pixel
  cont_imagen++;                                          // Incrementa la cuenta de datos de imagen decodificados
}
```

Una de las imágenes procesadas con el programa es: 87amk.gif que es un archivo realizado en la versión 87a.

87amk.gif

El programa da como resultado los siguientes datos:

```
Nombre de Archivo :87amk.GIF
Identificacion : GIF      Posición del puntero :13
Versión        : 87a
Total  en x    : 298
Total  en y    : 433
Bits por pixel : 8
Fondo          : 0
Color Global   (0=si, otro=no) : 0
Separador       : ,       Posición del puntero :791
xminimo        : 0
yminimo        : 0
xmaximo        : 298
ymaximo        : 433
Interlazado    (1=si, 0=no): 0
Mapa de Color Local (1=si, 0=no): 0_
```

Posteriormente se despliega la imagen.

Programar el método de decodificación, tanto en lenguaje C como en Pascal, puede parecer difícil, sin embargo, si se tiene claro el concepto de compresión con LZW (explicado en el capítulo anterior) será más sencillo entenderlo. Si se tiene alguna duda respecto a la forma en que se descomprimen los códigos basta con una prueba de escritorio para aclarar sus dudas.

IV.7 Formato TIFF

El formato TIFF es el más complicado de programar, respecto a su gran cantidad de etiquetas que maneja un archivo específico, así como el desplazamiento de datos para cada tira. La cabecera debe contener la estructura del formato, utiliza 5 estructuras principales:

IFH_TIFF. Cabecera del archivo.

Etiquetas_TIF. Especifica los datos de cada una de las etiquetas.

Direcc_del_archivo. Contienen los datos del directorio de imagen.

Etiquetas. Especifica las características de las etiquetas.

Paleta_TIF. Paleta de colores del archivo.

El programa lee archivos con paleta RGB (256), en escala de grises, con datos sin comprimir y empaquetados RLE (Llamados en TIF PackBits).

IV.7.1 En Lenguaje Pascal

En Pascal el programa se codifica de la siguiente manera.

```
Program Lee_TIF;        { Programa que lee el formato de archivo TIF
                          Realizado para el trabajo de tesis titulado:FORMATOS GRÁFICOS
                                    Programadores :
                                          Cantero Ramírez Carlos
                                          Trejo Bonaga Marilu
                                    Asesores :
                                          Manzo Salazar Itsmael
                                          Monterrosa Escobar Amilcar A. }

Uses crt,dos,graph;

Const                                                   {Tipos de datos utilizados por las etiquetas}

  tipos:Array [0..12] of STring[10]=('Sin Tipo ','Byte    ','Ascii   ','Short   ',
                                     'Long    ','Rational ','Sbyte   ','No defini',
                                     'Sshort  ','Slong   ','Srational','Float   ','Double  ');

Type                                                                  {Declaración de las 5 estructuras}
 IFH_TIFF = Record                                          {Cabecera del TIFF  (Image File Header IFH)}
   Iden :Array[0..1] of CHAR;                                          {Identificación, 4949h ó 4D4Dh}
   Ver:word;                                                                          {Version, 2Ah}
   Pos_dir:Longint;                                        {Posición del primer directorio de imagen (IFD)}
end;

Paleta_TIF=record                                                                   {Paleta de colores}
    rojo,verde,azul: word;
end;
```

```
Etiquetas_TIF=Record                                          {Campos de las etiquetas}
   etiqueta,                                                  {Descripción de la  etiqueta}
   tipo_dato :Word;                                           {Tipo del dato a utilizar}
   cont_dato,                                                 {Contador de dato}
   des_dato :Longint;                                         {Desplazamiento del dato}
End;

Direcc_del_archivo = Record                                   {Directorio de imagen}
   entradas:Word;                                             {Número de etiquetas en el TIF}
   etiqueta_tif:Array [1..80] of Etiquetas_TIF;               {Arreglo de etiquetas}
   siguiente:Longint;                                         {Desplazamiento del sig. directorio de imagen}
end;

Etiquetas=Record                                              {Datos de las etiquetas}
   nombre:String[15];                                         {Nombre}
   num:Word;                                                  {Número}
   tipo:Byte;                                                 {Tipo de etiqueta}
   ver:String[8];                                             {Versiones soportadas}
End;
Const                                                         {Arreglo para almacenar los datos de las etiquetas}
  Claves:Array[1..79] of Etiquetas=((nombre: 'Artista       ';num: 00315;tipo: 02;ver:'-- V5 V6'),
                         (nombre: 'Linea fax mala';num: 00326;tipo: 04;ver:'-- -- --'),
                         (nombre: 'Bits_muestra  ';num: 00258;tipo: 03;ver:'V4 V5 V6'),
                         (nombre: 'Largo de Celda';num: 00265;tipo: 03;ver:'V4 V5 V6'),
                         (nombre: 'Ancho de Celda';num: 00264;tipo: 03;ver:'V4 V5 V6'),
                         (nombre: 'Dato_fax_Nvo  ';num: 00327;tipo: 03;ver:'-- -- --'),
                         (nombre: 'Paleta        ';num: 00320;tipo: 03;ver:'-- V5 V6'),
                         (nombre: 'Curv_resp_col ';num: 00301;tipo: 03;ver:'V4 V5 --'),
                         (nombre: 'Uni_resp_col  ';num: 00300;tipo: 03;ver:'V4 -- --'),
                         (nombre: 'Compresión    ';num: 00259;tipo: 03;ver:'V4 V5 V6'),
                         (nombre: 'Lin_fax_mal_c ';num: 00328;tipo: 04;ver:'-- -- --'),
                         (nombre: 'Derechos      ';num: 33432;tipo: 02;ver:'-- -- V6'),
                         (nombre: 'Fecha y hora  ';num: 00306;tipo: 02;ver:'-- V5 V6'),
                         (nombre: 'Nombre_doc    ';num: 00269;tipo: 02;ver:'V4 V5 V6'),
                         (nombre: 'Rango_punto   ';num: 00336;tipo: 03;ver:'-- -- V6'),
                         (nombre: 'Muestras_extra';num: 00338;tipo: 01;ver:'-- -- V6'),
                         (nombre: 'Orden_relleno ';num: 00266;tipo: 03;ver:'V4 V5 V6'),
                         (nombre: 'Cta_byte_libre';num: 00289;tipo: 04;ver:'V4 V5 V6'),
                         (nombre: 'Desplaza_libre';num: 00288;tipo: 04;ver:'V4 V5 V6'),
                         (nombre: 'Resp_curv_gris';num: 00291;tipo: 03;ver:'V4 V5 V6'),
                         (nombre: 'Uni_resp_gris ';num: 00290;tipo: 03;ver:'V4 V5 V6'),
                         (nombre: 'Hints_med_tono';num: 00321;tipo: 03;ver:'-- -- V6'),
                         (nombre: 'Host_PC       ';num: 00316;tipo: 02;ver:'-- V5 V6'),
                         (nombre: 'Descrip_imagen';num: 00270;tipo: 02;ver:'V4 V5 V6'),
                         (nombre: 'Largo_imagen  ';num: 00257;tipo: 03;ver:'V4 V5 V6'),
                         (nombre: 'Ancho_imagen  ';num: 00256;tipo: 03;ver:'V4 V5 V6'),
                         (nombre: 'Nombres_ink   ';num: 00333;tipo: 02;ver:'-- -- V6'),
                         (nombre: 'Puesta_ink    ';num: 00332;tipo: 03;ver:'-- -- V6'),
                         (nombre: 'Tabla_JPEGACT ';num: 00521;tipo: 04;ver:'-- -- V6'),
                         (nombre: 'Tabla_JPEGDCT ';num: 00520;tipo: 04;ver:'-- -- V6'),
                         (nombre: 'Form_Int_JPEG ';num: 00513;tipo: 04;ver:'-- -- V6'),
                         (nombre: 'Largo_FIJPEG  ';num: 00514;tipo: 04;ver:'-- -- V6'),
                         (nombre: 'Pred_Com_JPEG ';num: 00517;tipo: 03;ver:'-- -- V6'),
                         (nombre: 'Trans_pun_JPEG';num: 00518;tipo: 03;ver:'-- -- V6'),
```

```
                    (nombre: 'Proceso_JPEG  ';num: 00512;tipo: 03;ver:'-- -- V6'),
                    (nombre: 'Int_reini_JPEG';num: 00515;tipo: 03;ver:'-- -- V6'),
                    (nombre: 'Tabla_JPEGQ   ';num: 00519;tipo: 04;ver:'-- -- V6'),
                    (nombre: 'Realizar      ';num: 00271;tipo: 02;ver:'V4 V5 V6'),
                    (nombre: 'Valmax_muestra';num: 00281;tipo: 03;ver:'V4 V5 V6'),
                    (nombre: 'Valmin_muestra';num: 00280;tipo: 03;ver:'V4 V5 V6'),
                    (nombre: 'Modelo        ';num: 00272;tipo: 02;ver:'V4 V5 V6'),
                    (nombre: 'Nvoti_Subarchi';num: 00254;tipo: 04;ver:'-- V5 V6'),
                    (nombre: 'Numero_ink    ';num: 00334;tipo: 03;ver:'-- -- V6'),
                    (nombre: 'Orientacion   ';num: 00274;tipo: 03;ver:'V4 V5 V6'),
                    (nombre: 'Nombre_pagina ';num: 00285;tipo: 02;ver:'V4 V5 V6'),
                    (nombre: 'Numero_pagina ';num: 00297;tipo: 03;ver:'V4 V5 V6'),
                    (nombre: 'Inter_fotome  ';num: 00262;tipo: 03;ver:'V4 V5 V6'),
                    (nombre: 'Config_planar ';num: 00284;tipo: 03;ver:'V4 V5 V6'),
                    (nombre: 'Predictor     ';num: 00317;tipo: 03;ver:'-- V5 V6'),
                    (nombre: 'Croma_prim    ';num: 00319;tipo: 05;ver:'-- V5 V6'),
                    (nombre: 'Referencia_B/N';num: 00532;tipo: 04;ver:'-- -- V6'),
                    (nombre: 'Uni_resolucion';num: 00296;tipo: 03;ver:'V4 V5 V6'),
                    (nombre: 'Lineas_x_Tiras';num: 00278;tipo: 04;ver:'V4 V5 V6'),
                    (nombre: 'Mues_formato  ';num: 00339;tipo: 03;ver:'-- -- V6'),
                    (nombre: 'Mues x pixel  ';num: 00277;tipo: 03;ver:'V4 V5 V6'),
                    (nombre: 'Val_smax_mues ';num: 00341;tipo: 04;ver:'-- -- V6'),
                    (nombre: 'Val_smin_mues ';num: 00340;tipo: 04;ver:'-- -- V6'),
                    (nombre: 'Software      ';num: 00305;tipo: 02;ver:'-- V5 V6'),
                    (nombre: 'cta_byte_Tira ';num: 00279;tipo: 04;ver:'V4 V5 V6'),
                    (nombre: 'Desplaza_Tlra ';num: 00273;tipo: 04;ver:'V4 V5 V6'),
                    (nombre: 'Tipo_subarchi ';num: 00255;tipo: 03;ver:'V4 V5 V6'),
                    (nombre: 'Opcion_t40    ';num: 00292;tipo: 04;ver:'V4 V5 V6'),
                    (nombre: 'Opcion_t60    ';num: 00293;tipo: 04;ver:'V4 V5 V6'),
                    (nombre: 'Tipo_impresora';num: 00337;tipo: 02;ver:'-- -- V6'),
                    (nombre: 'Thresholding  ';num: 00263;tipo: 03;ver:'V4 V5 V6'),
                    (nombre: 'Cta_byte_tile ';num: 00325;tipo: 04;ver:'-- -- V6'),
                    (nombre: 'Largo_tile    ';num: 00323;tipo: 04;ver:'-- -- V6'),
                    (nombre: 'Desplaza_tile ';num: 00324;tipo: 04;ver:'-- -- V6'),
                    (nombre: 'Ancho_tile    ';num: 00322;tipo: 04;ver:'-- -- V6'),
                    (nombre: 'Funcion_trans ';num: 00301;tipo: 03;ver:'-- -- V6'),
                    (nombre: 'Rango_trans   ';num: 00342;tipo: 03;ver:'-- -- V6'),
                    (nombre: 'Posicion_x    ';num: 00286;tipo: 05;ver:'V4 V5 V6'),
                    (nombre: 'Resolucion_y  ';num: 00282;tipo: 05;ver:'V4 V5 V6'),
                    (nombre: 'Coefi_YCbCr   ';num: 00529;tipo: 05;ver:'-- -- V6'),
                    (nombre: 'Posicion_YCbCr';num: 00531;tipo: 03;ver:'-- -- V6'),
                    (nombre: 'Submues_YCbCr ';num: 00530;tipo: 03;ver:'-- -- V6'),
                    (nombre: 'Posicion_y    ';num: 00287;tipo: 05;ver:'V4 V5 V6'),
                    (nombre: 'Resolucion_y  ';num: 00283;tipo: 05;ver:'V4 V5 V6'),
                    (nombre: 'Punto_blanco  ';num: 00318;tipo: 05;ver:'-- V5 V6'));
Var
 TIFFC:IFH_TIFF;                                    {Variable para almacenar la cabecera}
 TIFFD:direcc_del_archivo;                          {Variable para almacenar el directorio de imagen}
 Tif_Pal:Array [0..255] of Paleta_TIF;              {Variable para almacenar la paleta de colores}
 Archivo:string;                                    {Variable para almacenar el nombre del archivo}
 Posicion,Cont_imagen:Longint;                      {Variables auxiliares}
 i,j,k:integer;
 Sololee:Word;
 Lee_Arch:File;                                     {Archivo a leer}
```

```
Function Inicializa(Driver,Modo:integer):boolean;        {Inicializa gráficos y reporta los errores que ocurran}
var                                                                                {No necesita explicación}
  ErrorG:integer;
  x,y:word;
Begin
  if (Driver=16) Then
      Driver := InstallUserDriver('Svga256',nil);
  if Driver=grError Then
      halt(1);
  InitGraph(Driver, Modo,'d:\bp\programa');
  asm
    mov ax,000fh
    mov cx,8
    mov dx,8
    int 33h
  end;
  x:=getmaxx;
  y:=getmaxy;
  asm
     mov ax,x
     mov dx,ax
     mov ax,0007h
     mov cx,0h
     int 33h
     mov ax,y
     mov dx,ax
     mov ax,0008h
     mov cx,0h
     int 33h
   end;
 ErrorG:=graphResult;
 If ErrorG<>grok Then
  Begin
   Writeln('Error en graficos : ',GraphErrormsg(ErrorG,);
   Inicializa:=False;
   Restorecrtmode;
  End
 Else
  Begin
   cleardevice;
   Inicializa:=True;
  End;
End;

Procedure Cambia_color(color,r,v,a:Byte);                  {Cambia la paleta activa en el S.O.}
Begin                                                                              {No necesita explicación}
 asm
  mov dx,3C8h
  mov al,color
  out dx,al
  inc dx
  mov al,r
  out dx,al
  mov al,v
```

```
 out dx,al
 mov al,a
 out dx,al
 End;
End;

Procedure limpia(x,y,x1,y1,C:byte);                    {Deja limpia determinada área en pantalla}
Begin
 Window(x,y,x1,y1);
 Textbackground(c);
 clrscr;
 Window(1,1,80,25);
End;

Function ifi(cond:boolean;cond1,cond2:Longint):Longint;   {Función de condición If-else para enteros}
 Begin
  if cond Then ifi:=cond1 else ifi:=Cond2;
 End;

Function Abre_lectura_TIF(Arch_imagen:String):Boolean;        {Prepara al archivo para su lectura}
Begin                                                          {No necesita explicación}
  Assign(Lee_Arch,arch_imagen);
  {Si-}
  reset(Lee_arch,1);
  {Si+}
  if ioresult = 0 Then
    Abre_lectura_Tif:=True
  else
   Begin
     Writeln('Error de apertura de archivo');
      Abre_lectura_Tif:=False;
    End;
End;

Procedure Lee_cabecera_TIF(Archivo:String);                    {Despliega los datos de la cabecera}
Function busca(num:word):Byte;                        {Busca los datos de la etiqueta en el arreglo claves}
Var
   Cont:byte;
Begin
  cont:=0;
  Repeat
    inc(cont);
  Until (num=claves[cont].num) or (Cont>=79);
    Busca:=Cont;
  End;

Begin                                                          {Despliega los datos de la cabecera}
 Blockread(lee_arch,TiffC,sizeof(TiffC),sololee);
 Gotoxy(01,01);Write('Nombre de Archivo :',archivo);
 Gotoxy(01,02);Write('Identificacion   :');
 Gotoxy(25,02);Writeln(TIFFC.Iden[0],TIFFC.iden[1],'    ','Posición del puntero :',posicion);
 Gotoxy(01,03);Write('Versión          :');
 Gotoxy(25,03);Writeln(TIFFC.ver);
 Gotoxy(01,04);Write('OFFSET -         :');
```

```
Gotoxy(25,04);Writeln(TIFFC.pos_dir);
seek(Lee_Arch,TiffC.pos_dir);                    {Se coloca en la posición del primer directorio de imagen}
Blockread(lee_arch,TIFFD.entradas,sizeof(TIFFD.entradas),sololee);          {Lee cuantas etiquetas hay}
Gotoxy(01,05);Write('Numero de etiquetas:');
Gotoxy(25,05);Writeln(TIFFD.Entradas);
  With TIFFD do                                          {Lee los datos de cada etiqueta del archivo}
  For i:=1 to Entradas do
    With etiqueta_tif[i] do
     Begin
      Blockread(lee_arch,etiqueta,2,sololee);
      Blockread(lee_arch,Tipo_dato,2,sololee);
      Blockread(lee_arch,Cont_dato,4,sololee);
      Blockread(lee_arch,des_dato,4,sololee);
     End;
Posicion:=Filepos(Lee_arch);
Blockread(lee_arch,TIFFD.siguiente,1,sololee);       {Toma la posiscion de una siguiente imagen, si la hay}
Posicion:=Filepos(Lee_arch);
Gotoxy(01,06);Write('Siguiente imagen :');
Gotoxy(25,06);Writeln(TIFFD.siguiente);
Gotoxy(01,07);Writeln('Nombre        etiqueta    Tipo        Version  Datacount   Dataoffset');
Gotoxy(01,08);                                         Despliega en pantalla los datos de las etiquetas}
 k:=8; j:=1;
 With TIFFD do
  For i:=1 to Entradas do
   Begin
    With etiqueta_tif[i] do
     Begin
       Gotoxy(1,k);Write(claves[Busca(etiqueta)].nombre);
       Gotoxy(17,k);Write(claves[Busca(etiqueta)].num);
       Gotoxy(28,k);Write(tipos[claves[Busca(etiqueta)].tipo],'(',tipo_dato,')');
       Gotoxy(42,k);Write(claves[Busca(etiqueta)].ver);
       Gotoxy(52,k);Write(Cont_dato);
       Gotoxy(64,k);Write(des_dato);
     End;
     inc(k);
    if k>23 Then
      Begin
       k:=8;
       Readkey;
       limpia(1,8,79,24,0);        {Limpia cierta área de la pantalla cuando esta se haya llenado con etiquetas}
      End;
   End;
End;

Function BusTIF(num:word):Byte;                      {Busca una etiqueta y devuelve su posicion en el arreglo}
Var
  cont:byte;
Begin
  cont:=0;
  Repeat
    inc(Cont);
  Until (num=tiffd.etiqueta_tif[cont].etiqueta) or (Cont>tiffd.entradas);
  BusTif:=ifi(Cont>tiffd.entradas,0,cont);
 End;
```

```
Procedure Lee_Paleta_TIF(Archivo:String);                    {Lee la paleta de colores del archivo}
Var
  posis:Word;
  r1,v1,a1,r2,v2,a2,pal_sn: byte;
Begin
 pal_sn:=BusTif(262);
 For posis:=0 to 255 do
  Begin
   Tif_pal[posis].rojo :=0;
   Tif_pal[posis].verde:=0;
   Tif_pal[posis].azul :=0;
  End;
  IF pal_sn<>0 Then
  Case tiffd.etiqueta_tif[pal_SN].des_dato OF
   0:                                  {Paleta de colores en escala de grises, blanco si es cero}
    Begin                                                                          {No soportado}
    End;
   1:                                  {Paleta de colores en escala de grises, negro si es cero}
    Begin
     If (busTIF(290)<>0) and (BusTIF(291)<>0) Then          {Unidad y curva de respuesta en grises}
      For posis:=0 to 255 do
       Begin
         Tif_pal[posis].Rojo :=posis;
         Tif_pal[posis].Verde:=posis;
         Tif_pal[posis].Azul :=posis;
       End;
    End;
   2:                                                                   {RGB Color Total o True Color}
    Begin                                                                          {No soportado}
    End;
   3:                                                                                {Paleta RGB}
     Begin                                              {Busca, lee y almacena datos de la paleta RGB}
       seek(lee_arch,tiffd.etiqueta_tif[busTIF(320)].des_dato);
       posis:=tiffd.etiqueta_tif[busTIF(320)].des_dato;
       Posicion:=Filepos(Lee_arch);        {Como los datos de la paleta vienen en planos de color RGB, se lee}
                                                                          {cada plano de forma independiente}
       for posis:=0 to 255 do                                                        {Plano de color Rojo}
        Begin
          BlockRead(lee_arch,r1,1,sololee);
          BlockRead(lee_arch,r2,1,sololee);
          Tif_pal[posis].rojo :=r2 or (r1 shl 8);        {Realiza esta operación ya que los datos en el TIF estan}
        End;
       for posis:=0 to 255 do                                                       {Plano de color Verde}
       Begin
          BlockRead(lee_arch,v1,1,sololee);
          BlockRead(lee_arch,v2,1,sololee);
          Tif_pal[posis].verde :=v2 or (v1 shl 8);
        End;
       for posis:=0 to 255 do                                                        {Plano de color Azul}
        Begin
          BlockRead(lee_arch,a1,1,sololee);
          BlockRead(lee_arch,a2,1,sololee);
          Tif_pal[posis].azul :=a2 or (a1 shl 8);
       End;
```

```
    End;
   4:                                                          {Máscara de transparencia}
    Begin                                                                    {No soportado}
    End;
   5:                                                                      {Color tipo CMYK}
    Begin                                                                    {No soportado}
    End;
   6:                                                                      {Color tipo YCbCr}
    Begin                                                                    {No soportado}
    End;
   8:                                                                     {Color tipo CIELab}
    Begin                                                                    {No soportado}
    End;
  End;
End;

Procedure Pon_Paleta_TIF;                                          {Activa la paleta leída del archivo}
Var
 i:integer;
Begin
 For i:=0 to 255 do                                                   {Cambia la paleta de colores}
   Cambia_color(i, (Tif_pal[i].rojo and $FF) shr 2,(Tif_pal[i].verde and $ff) shr 2,
                                                                    (Tif_pal[i].azul and $ff) shr 2);
End;

Procedure  Lee_imagen_TIF;                                   {Lee y descomprime los datos de imagen}
Var
 No_dir,Cont_no_dir:Integer;
 xmaximo,ymaximo,ponx,pony:Word;
 tam_imagen,i,cont,localiza,cta_byte,aux:Longint;
 color,cuantos:byte;
 unbyte:ShortInt;
Begin
 No_dir:=tiffd.etiqueta_tif[BusTif(273)].cont_dato;                          {No. de tiras en la imagen }
 xmaximo:=tiffd.etiqueta_tif[BusTif(256)].des_dato;                                   {Ancho de la imagen}
 yMaximo:=tiffd.etiqueta_tif[BusTif(257)].des_dato;                                    {Largo de la imagen}
 tam_imagen:=xmaximo;
 tam_imagen:=tam_imagen*ymaximo;                                          {Calcula tamaño de la imagen}
 i:=0;
 cont:=0;
 Cont_no_dir:=0;                                                                      {Contador de tiras}
 Repeat                                   {Busca el cta_bytemiento de la tira y la cuenta de bytes por tira}
  Seek(lee_arch,tiffd.etiqueta_tif[BusTif(273)].des_dato+(Cont_no_dir*4));
  blockRead(lee_arch,localiza,4,sololee);                {Localiza cada uno de los desplazamientos por tira}
  Seek(lee_arch,tiffd.etiqueta_tif[BusTif(279)].des_dato+(Cont_no_dir*4));
  blockRead(lee_arch,cta_byte,4,sololee);                                {Cuantos bytes se leen por cada tira}
  seek(lee_arch,localiza);                                        {posiciona el puntero del archivo en la tira}
  inc(Cont_no_dir);
  If BusTif(259)<>0 Then                          {Verifica que el tipo de compresión sea de los utilizados por TIFF}
  Begin
   Case TIFFD.etiqueta_tif[BusTif(259)].des_dato of
    1:                                                                                   {Sin compresión}
    Begin
      aux:=TIFFD.etiqueta_tif[BusTif(262)].des_dato;
```

```
  If (aux=1) or (aux=3) Then                                  {Paleta en escala de grises o paleta RGB}
   Begin
     For cont:=1 to cta_byte do                                {Para cada dato de la tira}
      Begin
        ponx:=(i mod (xmaximo));                               {Calcula posición x del pixel}
        pony:=(i div (xmaximo));                               {Calcula posición y del pixel}
        blockread(Lee_Arch,color,sizeof(color),sololee);       {Lee el dato del pixel}
        putpixel(ponx,Pony,color);                             {Pone el pixel en pantalla}
        inc(i);                                                {Incrementa cuenta de datos para la tira}
      End;
   End;
 End;
2:                                                             {Compresión  CCITT ID}
  Begin                                                        {No soportado}
  End;
3:                                                             {Compresión CCITT Grupo 3}
  Begin                                                        {No soportado}
  End;
4:                                                             {Compresión CCITT Grupo 4}
  Begin                                                        {No soportado}
  End;
5:                                                             {Compresión LZW}
  Begin        {No soportado, ya que no solo se utiliza LZW si no otro metodo de compresión}
  End;
6:                                                             {Compresión JPG}
  Begin                                                        {No soportado}
  End;
Else                                                           {Ninguno de los anteriores}
  Begin
  If tiffd.etiqueta_tif[BusTif(259)].des_dato=32771 Then       {Sin compresion}
   Begin
     For cont:=1 to cta_byte do                                {Para cada dato de la tira}
      Begin
       ponx:=(i mod (xmaximo));                                {Calcula posición x del pixel}
       pony:=(i div (xmaximo));                                {Calcula posición y del pixel}
       blockread(Lee_Arch,color,sizeof(color),sololee);        {Lee el dato del pixel}
       putpixel(ponx,Pony,color);                              Pone el pixel en pantalla}
       inc(i);                                                 {Incrementa cuenta de datos para la tira}
      End;
   End
  Else                                                         {Bits Empaquetados RLE "PackBits"}
   IF  tiffd.etiqueta_tif[BusTif(259)].des_dato=32773 Then
    Begin
     aux:=0;
     Repeat
      BlockRead(lee_arch,unbyte,1,sololee);                    {Lee el primer dato}
      inc(aux);                                                {Incrementa cuenta de auxiliar}
      IF (unbyte>=0) and (unbyte<=127) Then                    {Sin empaquetar}
       Begin
        For i:=1 to unbyte+1 Do                                {Lee tantas veces más uno el valor de un byte}
         Begin
          BlockRead(lee_arch,color,1,sololee);
          inc(aux);                                            {Incrementa cuenta auxiliar, datos leidos de la tira}
          inc(cont);                                           {Incrementa cuenta de datos de imagen}
```

```
            ponx:=(cont mod (xmaximo));                    {Calcula posición x del pixel}
            pony:=(cont div (xmaximo));                    {Calcula posición y del pixel}
            putpixel(ponx,Pony,color);                     {Pone el pixel en pantalla}
           End;
          End
         Else                                             {Empaquetados}
          Begin
           Cuantos:=(unbyte*-1)+1;                        Ver la nota al final del programa}
           BlockRead(lee_arch,color,1,sololee);
           inc(aux);                                      {Incrementa cuenta auxiliar, datos  leídos de la tira}
           For i:=1 to cuantos do                         {Imprime el pixel tantas veces el valor de cuantos}
            Begin
             inc(cont);                                   {Incrementa cuenta de datos de imagen}
             ponx:=(cont mod (xmaximo));                  {Calcula la posición x del pixel}
             pony:=(cont div (xmaximo));                  {Calcula la posición y del pixel}
             putpixel(Ponx,Pony,color);                   {Pone el pixel en pantalla}
            End;
           End;
          Until (aux=cta_byte);                           {Hasta leer todos los datos de la tira}
        End;
      End;
    End;
  End;
 Until(Cont_no_dir=No_dir);                               {Hasta leer todas las tiras}
 Posicion:=Filepos(Lee_arch);
End;

Begin                                                     {Cuerpo del bloque principal del programa}
 clrscr;                                                  {Limpia pantalla}
 Archivo:='Pakmk256.TIF';                                 {Asigna el nombre del archivo}
 If Abre_lectura_Tif(Archivo) Then                        {Si puede abrir el archivo continua con el programa}
  Begin
    Lee_cabecera_Tif(Archivo);                            {Lee y pone en pantalla la informacion de la cabecera}
    Readkey;                                              {Pausa para tomar nota de la cabecera}
    Lee_Paleta_TIF(Archivo);                              {Lee los datos de la paleta}
    Inicializa(16,4);                                     {Inicializa el modo gráfico, 1024x768x256}
    Pon_Paleta_TIF;                                       {Activa la paleta leída del archivo}
    Lee_imagen_Tif;                                       {Descomprime la imagen y la visualiza}
    Readkey;                                              {Pausa para ver la imagen}
    Close(Lee_arch);                                      {Cierra el archivo}
    Restorecrtmode;                                       {Recupera el modo texto}
    closegraph;                                           {Cierra librería de  gráficos}
    clrscr;                                               {Limpia pantalla}
  End;
End.
```

Nota: La multiplicación por (-1) es por la razon que cuando se manejan bytes con signo el S. O. utiliza un noveno bit para el signo. "1" cuando son valores negativos y es "0" cuando son positivos (de un repaso a el sistema binario y la forma de manejo del S. O.). En TIF en vez de ser 3 datos compresos en dis seran un -3 cuando se utilize este tipo.

Una de las imágenes procesadas con el programa es: Pakmk256.Tif que es un archivo TIF con compresión packbits.

Pakmk256.Tif

El programa da como resultado los siguientes datos:

```
Nombre de Archivo :pakmk256.TIF
Identificacion    :      II      Posición del puntero :8
Versión           :      42
OFFSET -          :      88178
Numero de etiquetas:     13
Siguiente imagen  :      0
Nombre          etiqueta     Tipo          Version   Datacount    Dataoffset
Nuoti_Subarchi   254         Long     <4>  --  U5  U6  1            0
Ancho_imagen     256         Short    <4>  U4  U5  U6  1            400
Largo_imagen     257         Short    <4>  U4  U5  U6  1            254
Bits_muestra     258         Short    <3>  U4  U5  U6  1            8
Compresión       259         Short    <3>  U4  U5  U6  1            32773
Inter_fotome     262         Short    <3>  U4  U5  U6  1            3
Desplaza_Tira    273         Long     <4>  U4  U5  U6  15           88058
Orientacion      274         Short    <3>  U4  U5  U6  1            1
Mues x pixel     277         Short    <3>  U4  U5  U6  1            1
Lineas_x_Tiras   278         Long     <4>  U4  U5  U6  1            18
cta_byte_Tira    279         Long     <4>  U4  U5  U6  15           88118
Config_planar    284         Short    <3>  U4  U5  U6  1            1
Paleta           320         Short    <3>  --  U5  U6  768          86522_
```

Posteriormente se despliega la imagen.

IV.7.2 En Lenguaje C

En C el programa se codifica de la siguiente manera.

```
                    // Programa que lee el formato de archivo TIF
                    Realizado para el trabajo de tesis titulado:FORMATOS GRÁFICOS
                                Programadores :
                                        Cantero Ramirez Carlos
                                        Trejo Bonaga Marilu
                                Asesores :
                                        Manzo Salazar Itsmael
                                        Monterrosa Escobar Amilcar A.

#include <fcntl.H>
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>
#include <dos.h>
#include <conio.h>
#include <io.h>

#define BYTE    unsigned char                                   // Definición de constantes simbólicas
#define True  1
#define False 0
#define LONGINT unsigned long
#define WORD    unsigned int

char tipos[13][10]={"Sin Tipo ","Byte     ","Ascii    ",        // Tipos de datos utilizados por las etiquetas
                    "Short    ","Long     ","Rational ",
                    "Sbyte    ","No defini","Sshort   ",
                    "Slong    ","Srational","Float    ",
                    "Double   "};

                                                                // Declaración de las 5 estructuras
typedef struct                                                  // Cabecera del TIFF (Image File Header IFH)
{                                                               // Identificación, 4949h ó 4D4Dh
  char iden[2];
  WORD ver;                                                     // Version, 2Ah
  LONGINT pos_dir;                                              // Posicion del primer directorio de imagen (IFD)
 }IFH_TIFF;

typedef struct                                                  Paleta de colores
{
  WORD rojo,verde,azul;
}Paleta_TIF;

typedef struct                                                  // Campos de las etiquetas
{                                                               // Descripción de la  etiqueta
 WORD etiqueta,
          tipo_dato;                                            // Tipo del dato a utilizar
 LONGINT cont_dato,                                             // Contador de dato
              des_dato;                                         // Desplazamiento del dato
}etiquetas_tif;
```

```
typedef struct                                                    Directorio de imagen
{                                                                 Número de etiquetas en el TIF
  WORD entradas;
  etiquetas_tif etiqueta_tif[80];                                 // Arreglo de etiquetas
  LONGINT siguiente;                                              // Desplazamiento del sig. directorio de imagen
}direcc_del_archivo;

typedef struct                                                    // Datos de la etiquetas
{                                                                 // Nombre
  char *nombre;
  WORD num;                                                       // Número
  BYTE tipo;                                                      // Tipo de etiqueta
  char *ver;                                                      // Versiones soportadas
}etiquetas;
                                                                  // Arreglo para almacenar los datos de las etiquetas
etiquetas claves[80] ={{ "Artista       ", 315, 2,"-- V5 V6"},
                        { "Linea fax mala", 326, 4,"-- -- --"},
                        { "Bits_muestra  ", 258, 3,"V4 V5 V6"},
                        { "Largo de Celda", 265, 3,"V4 V5 V6"},
                        { "Ancho de Celda", 264, 3,"V4 V5 V6"},
                        { "Dato_fax_Nvo  ", 327, 3,"-- -- --"},
                        { "Paleta        ", 320, 3,"-- V5 V6"},
                        { "Curv_resp_col ", 301, 3,"V4 V5 --"},
                        { "Uni_resp_col  ", 300, 3,"V4 -- --"},
                        { "Compresión    ", 259, 3,"V4 V5 V6"},
                        { "Lin_fax_mal_c ", 328, 4,"-- -- --"},
                        { "Derechos      ", 33432,2,"-- -- V6"},
                        { "Fecha y hora  ", 306, 2,"-- V5 V6"},
                        { "Nombre_doc    ", 269, 2,"V4 V5 V6"},
                        { "Rango_punto   ", 336, 3,"-- -- V6"},
                        { "Muestras_extra", 338, 1,"-- -- V6"},
                        { "Orden_relleno ", 266, 3,"V4 V5 V6"},
                        { "Cta_byte_libre", 289, 4,"V4 V5 V6"},
                        { "Desplaza_libre", 288, 4,"V4 V5 V6"},
                        { "Resp_curv_gris", 291, 3,"V4 V5 V6"},
                        { "Uni_resp_gris ", 290, 3,"V4 V5 V6"},
                        { "Hints_med_tono", 321, 3,"-- -- V6"},
                        { "Host_PC       ", 316, 2,"-- V5 V6"},
                        { "Descrip_imagen", 270, 2,"V4 V5 V6"},
                        { "Largo_imagen  ", 257, 3,"V4 V5 V6"},
                        { "Ancho_imagen  ", 256, 3,"V4 V5 V6"},
                        { "Nombres_ink   ", 333, 2,"-- -- V6"},
                        { "Puesta_ink    ", 332, 3,"-- -- V6"},
                        { "Tabla_JPEGACT ", 521, 4,"-- -- V6"},
                        { "Tabla_JPEGDCT ", 520, 4,"-- -- V6"},
                        { "Form_Int_JPEG ", 513, 4,"-- -- V6"},
                        { "Largo_FIJPEG  ", 514, 4,"-- -- V6"},
                        { "Pred_Com_JPEG ", 517, 3,"-- -- V6"},
                        { "Trans_pun_JPEG", 518, 3,"-- -- V6"},
                        { "Proceso_JPEG  ", 512, 3,"-- -- V6"},
                        { "Int_reini_JPEG", 515, 3,"-- -- V6"},
                        { "Tabla_JPEGQ   ", 519, 4,"-- -- V6"},
                        { "Realizar      ", 271, 2,"V4 V5 V6"},
                        { "Valmax_muestra", 281, 3,"V4 V5 V6"},
```

```
{ "Valmin_muestra", 280, 3,"V4 V5 V6"},
{ "Modelo        ", 272, 2,"V4 V5 V6"},
{ "Nvoti_Subarchi", 254, 4,"-- V5 V6"},
{ "Numero_ink    ", 334, 3,"-- -- V6"},
{ "Orientacion   ", 274, 3,"V4 V5 V6"},
{ "Nombre_pagina ", 285, 2,"V4 V5 V6"},
{ "Numero_pagina ", 297, 3,"V4 V5 V6"},
{ "Inter_fotome  ", 262, 3,"V4 V5 V6"},
{ "Config_planar ", 284, 3,"V4 V5 V6"},
{ "Predictor     ", 317, 3,"-- V5 V6"},
{ "Croma_prim    ", 319, 5,"-- V5 V6"},
{ "Referencia_B/N", 532, 4,"-- -- V6"},
{ "Uni_resolucion", 296, 3,"V4 V5 V6"},
{ "Lineas_x_Tiras", 278, 4,"V4 V5 V6"},
{ "Mues_formato  ", 339, 3,"-- -- V6"},
{ "Mues x pixel  ", 277, 3,"V4 V5 V6"},
{ "Val_smax_mues ", 341, 4,"-- -- V6"},
{ "Val_smin_mues ", 340, 4,"-- -- V6"},
{ "Software     ", 305, 2,"-- V5 V6"},
{ "cta_byte_Tira ", 279, 4,"V4 V5 V6"},
{ "Desplaza_TIra ", 273, 4,"V4 V5 V6"},
{ "Tipo_subarchi ", 255, 3,"V4 V5 V6"},
{ "Opcion_t40    ", 292, 4,"V4 V5 V6"},
{ "Opcion_t60    ", 293, 4,"V4 V5 V6"},
{ "Tipo_impresora", 337, 2,"-- -- V6"},
{ "Thresholding  ", 263, 3,"V4 V5 V6"},
{ "Cta_byte_tile ", 325, 4,"-- -- V6"},
{ "Largo_tile    ", 323, 4,"-- -- V6"},
{ "Desplaza_tile ", 324, 4,"-- -- V6"},
{ "Ancho_tile    ", 322, 4,"-- -- V6"},
{ "Funcion_trans ", 301, 3,"-- -- V6"},
{ "Rango_trans   ", 342, 3,"-- -- V6"},
{ "Posicion_x    ", 286, 5,"V4 V5 V6"},
{ "Resolucion_y  ", 282, 5,"V4 V5 V6"},
{ "Coefi_YCbCr   ", 529, 5,"-- -- V6"},
{ "Posicion_YCbCr", 531, 3,"-- -- V6"},
{ "Submues_YCbCr ", 530, 3,"-- -- V6"},
{ "Posicion_y    ", 287, 5,"V4 V5 V6"},
{ "Resolucion_y  ", 283, 5,"V4 V5 V6"},
{ "Punto_blanco  ", 318, 5,"-- V5 V6"}};

char *archivo;                          // Variable para almacenar el nombre del archivo
IFH_TIFF tiffc;                         // Variable para almacenar la cabecera
direcc_del_archivo tiffd;               // Variable para almacenar el directorio de imagen
Paleta_TIF tif_pal[256];                // Variable para almacenar la paleta de colores
LONGINT cont_imagen;
int ban_pal;                            // Variables auxiliares
int i,j,k;
fpos_t posicion;
FILE *lee_arch;                         // Archivo a leer

void Lee_Cabecera_TIF(void);            // Declaración de las funciones utilizadas
int Inicializa(int Driver, int Modo);
int Abre_lectura_TIF(char *arch_imagen);
```

```
 void Lee_Paleta_TIF(void);
 void Pon_Paleta_TIF();
 void Lee_imagen_TIF();
 void Cambia_color(BYTE color,BYTE r,BYTE v,BYTE a);

 void main ()                                          // Cuerpo del bloque principal del programa
 {                                                     // Limpia pantalla
   clrscr();
   archivo="uncomp.tif";                               // Asigna el nombre del archivo
   if (Abre_lectura_TIF(archivo))                      // Si puede abrir el archivo continua con el programa
   {                                                   // Lee y pone en pantalla la información de la cabecera
    Lee_Cabecera_TIF();
    getch();                                           // Pausa para tomar nota de la cabecera
    Lee_Paleta_TIF();                                  // Lee los datos de la paleta
    Inicializa(16,4);                                  // Inicializa el modo gráfico
    Pon_Paleta_TIF();                                  // Activa la paleta leída del archivo
    Lee_imagen_TIF();                                  // Descomprime la imagen y la visualiza
    getch();                                           // Pausa para ver la imagen
    fclose(lee_arch);                                  // Cierra el archivo
    restorecrtmode();                                  // Recupera el modo texto
    closegraph();                                      // Cierra librería de gráficos
    clrscr();                                          // Limpia pantalla
   }
}

int Inicializa(int Driver, int Modo)                   // Inicializa gráficos y reporta los errores que ocurran
{                                                      // No necesita explicación
 int ErrorG;
 unsigned int x,y;
 union REGS regin,regout;
 if (Driver==16)
 Driver = installuserdriver("Svga256",NULL);
 if (Driver==grError) exit;
 initgraph( &Driver, &Modo,"d:\borlandc\programa\exeobj");
 regin.x.ax=0x000f;
 regin.x.cx=0x8;
 regin.x.dx=0x8;
 int86(0x33,&regin,&regout);
 x=getmaxx();
 y=getmaxy();
 regin.x.ax=x;
 regin.x.dx=regin.x.ax;
 regin.x.ax=0x0007;
 regin.x.cx=0x0;
 int86(0x33,&regin,&regout);
 regin.x.ax=y;
 regin.x.dx=regin.x.ax;
 regin.x.ax=0x0008;
 regin.x.cx=0x0;
 int86(0x33,&regin,&regout);
 ErrorG=graphresult();
if (ErrorG!=grOk)
{
 printf("\n Error en graficos : %s",grapherrormsg(ErrorG));
```

```
   restorecrtmode;
   return 0;
  }
 else
  {
    cleardevice;
    return 1;
   }
}

void Cambia_color(BYTE color,BYTE r,BYTE v,BYTE a)                    // Cambia la paleta activa en el S.O.
{                                                                     // No necesita explicación
 asm mov dx,3C8h
 asm mov al,color
 asm out dx,al
 asm inc dx
 asm mov al,r
 asm out dx,al
 asm mov al,v
 asm out dx,al
 asm mov al,a
 asm out dx,al
}

int Abre_lectura_TIF(char *arch_imagen)                               // Prepara al archivo para su lectura
{                                                                     // No necesita explicación
 lee_arch=fopen(arch_imagen,"rb");
 if (lee_arch==NULL)
   {return 0;}
 else
   {return 1;}
}

void Lee_Cabecera_TIF()                                               // Lee los datos de la cabecera y los despliega en pantalla
{                                                                     // Declara las funciones que utiliza
 BYTE busca(WORD num);                                                // Busca los datos de la etiqueta  en el arreglo
 void limpia(BYTE x,BYTE y,BYTE x1,BYTE y1,BYTE c);                   // Limpia área en pantalla
 fread((char *)&tiffc,1,sizeof(IFH_TIFF),lee_arch);                   // Despliega los datos de la cabecera
 fgetpos(lee_arch,&posicion);
 gotoxy( 1, 1);printf("Nombre de Archivo :%s",archivo);
 gotoxy( 1, 2);printf("Identificacion   :%c%c    Posición del puntero :%d", tiffc.iden[0], tiffc.iden[1],posicion);
 gotoxy( 1, 3);printf("Versión          :%d",tiffc.ver);
 gotoxy( 1, 4);printf("OFFSET -         :%ld",tiffc.pos_dir);
 fseek(lee_arch,tiffc.pos_dir,SEEK_SET);                              // Busca la posición del directorio de imagen
 fread((BYTE *)&tiffd,1,sizeof(direcc_del_archivo),lee_arch);         // Cuantas etiquetas hay
 fgetpos(lee_arch,&posicion);
 gotoxy( 1, 5);printf("Numero de etiquetas: %d",tiffd.entradas);
 for (i=0;i<tiffd.entradas;i++)                                       // Almacena los datos de las etiquetas
   {
     fread((WORD    *)&tiffd.etiqueta_tif[i].etiqueta ,1,sizeof(tiffd.etiqueta_tif[i].etiqueta) ,lee_arch);
     fread((WORD    *)&tiffd.etiqueta_tif[i].tipo_dato,1,sizeof(tiffd.etiqueta_tif[i].tipo_dato),lee_arch);
     fread((LONGINT *) &tiffd.etiqueta_tif[i].cont_dato, 1,sizeof(tiffd.etiqueta_tif[i].cont_dato), lee_arch);
     fread((LONGINT *)&tiffd.etiqueta_tif[i].des_dato ,1,sizeof(tiffd.etiqueta_tif[i].des_dato), lee_arch);
   }                                                                  // Checa si hay otro directorio de imagen
```

```
  fread((BYTE *)&tiffd.siguiente,1,sizeof(tiffd.siguiente),lee_arch);
  fgetpos(lee_arch,&posicion);
  gotoxy( 1, 6);printf("Siguiente imagen : %ld ",tiffd.siguiente);
  gotoxy( 1, 7);printf("Nombre        etiqueta    Tipo      Version  Datacount  Dataoffset");
  gotoxy( 1, 8);                                    // Despliega en pantalla los datos de las etiquetas
  k=8;
  j=1;
   for (i=0;i<tiffd.entradas;i++)
   {
    gotoxy(1, k);printf("%s",claves[busca(tiffd.etiqueta_tif[i].etiqueta)].nombre);
    gotoxy(17,k);printf("%d",claves[busca(tiffd.etiqueta_tif[i].etiqueta)].num);
    gotoxy(28,k);printf("%s(%d)", tipos[claves[busca(tiffd.etiqueta_tif[i].etiqueta)].tipo],
                                                                        tiffd.etiqueta_tif[i].tipo_dato);
    gotoxy(42,k);printf("%s",claves[busca(tiffd.etiqueta_tif[i].etiqueta)].ver);
    gotoxy(52,k);printf("%ld",tiffd.etiqueta_tif[i].cont_dato);
    gotoxy(64,k);printf("%ld",tiffd.etiqueta_tif[i].des_dato);
    k++;
    if (k>23)
     {
      k=8;
      getch();
      limpia(1,8,79,24,0);                          // Limpia cierta área de la pantalla
     }
   }
 }

 BYTE busca(WORD num)                               // Busca los datos de la etiqueta en el arreglo claves
 {
   BYTE  cont;
   cont=0;
   do
   {
     cont++;
   } while(!((num==claves[cont].num) || (cont>=79)));
  return cont;
}

void limpia(BYTE x,BYTE y,BYTE x1,BYTE y1,BYTE c)                    // Limpia un área en pantalla
 {
  window(x,y,x1,y1);
  textbackground(c);
  clrscr();
  window(1,1,80,25);
 }

void Lee_Paleta_TIF()                                   // Lee la paleta de colores del archivo
{
  WORD posis;
  BYTE r1,v1,a1,r2,v2,a2,pal_sn;
  BYTE bustif(WORD num);
  pal_sn=bustif(262);
  for (posis=0;posis<=255;posis++)
  {
   tif_pal[posis].rojo =0;
```

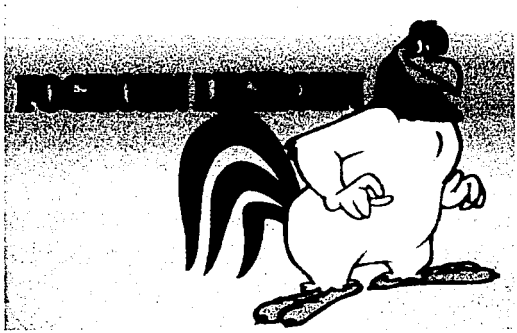
```
  tif_pal[posis].verde=0;
  tif_pal[posis].azul =0;
 }
 if (pal_sn!=0)
  {
   switch (tiffd.etiqueta_tif[pal_sn].des_dato)
    {
     case 0:                                          // Paleta de colores en escala de grises. blanco si es cero
          { break;};                                                                          // No soportado
     case 1:                                          // Paleta de colores en escala de grises. negro si es cero
          {                                                                                   // Revisa si es ecala de grises
           if ((bustif(290)!=0) && (bustif(291)!=0))                                          // Unidad y curva de respuesta grises
            for (posis=0;posis<=255;posis++)
             {
               tif_pal[posis].rojo =posis;
               tif_pal[posis].verde=posis;
               tif_pal[posis].azul =posis;
             }
            break;
           }
     case 2:                                                                                  // RGB Color Total o True Color
          {break;}                                                                            // No soportado
     case 3:                                                                                  // Paleta RGB
          {                                                                                   // Busca, lee y almacena datos de la paleta RGB
           fseek(lee_arch,tiffd.etiqueta_tif[bustif(320)].des_dato,SEEK_SET);
                                                                                              // Como los datos de la paleta vienen en planos de color RGB,
                                                                                              // se lee cada plano de forma independiente
            posis=tiffd.etiqueta_tif[bustif(320)].des_dato;
            for (posis=0;posis<=255;posis++)                                                  // Plano de color Rojo
            {
              fread((BYTE *)&r1,1,sizeof(r1),lee_arch);
              fread((BYTE *)&r2,1,sizeof(r2),lee_arch);
              tif_pal[posis].rojo =r2 | (r1 << 8);
            }
            for (posis=0;posis<=255;posis++)                                                  // Plano de color Verde
            {
              fread((BYTE *)&v1,1,sizeof(r1),lee_arch);
              fread((BYTE *)&v2,1,sizeof(r2),lee_arch);
              tif_pal[posis].verde =v2 | (v1 << 8);
            }
            for (posis=0;posis<=255;posis++)                                                  // Plano de color Azul
            {
              fread((BYTE *)&a1,1,sizeof(r1),lee_arch);
              fread((BYTE *)&a2,1,sizeof(r2),lee_arch);
              tif_pal[posis].azul =a2 | (a1 << 8);
            }
           break;
          }
     case 4:                                                                                  // Máscara de transparencia
          {break;}                                                                            // No soportado
     case 5:                                                                                  // Color tipo CMYK
          {break;}                                                                            // No soportado
     case 6:                                                                                  // Color tipo YCbCr
          {break;}                                                                            // No soportado
```

```
      case 8:                                                    // Color tipo CIELab
          {break;}                                               // No soportado
    }
  }
}

BYTE bustif(WORD num)                    // Busca una etiqueta y devuelve su posición en el arreglo
{
 BYTE cont=0;
 do
  {
   cont++;
  } while (!((num==tiffd.etiqueta_tif[cont].etiqueta) || (cont>tiffd.entradas)));
 if(cont>tiffd.entradas)
   return 0;
 else
   return cont;
}

void Pon_Paleta_TIF()                                        // Activa la paleta leída del archivo
{
  int i;
  for (i=0;i<=255;i++)                                       // Cambia la paleta de colores
    Cambia_color(i,tif_pal[i].rojo >> 2,tif_pal[i].verde >> 2,tif_pal[i].azul>>2);
}

void Lee_imagen_TIF()                                        // Lee y descomprime los datos de imagen
{
  int no_dir,cont_no_dir;
  WORD xmaximo,ymaximo,ponx,pony;
  LONGINT tam_imagen,i,cont,localiza,cta_byte,aux;
  BYTE color,cuantos;
  char unbyte;
  no_dir =tiffd.etiqueta_tif[bustif(273)].cont_dato;                 // No. de tiras en la imagen
  xmaximo=tiffd.etiqueta_tif[bustif(256)].des_dato;                  // Ancho de la imagen
  ymaximo=tiffd.etiqueta_tif[bustif(257)].des_dato;                  // Largo de la imagen
  tam_imagen=xmaximo;
  tam_imagen=tam_imagen*ymaximo;                             // Calcula tamaño de la imagen
  i=0;
  cont=0;
  cont_no_dir=0;                                                     // Contador de tiras
  do
  {                                        // Busca el desplazamiento de la tira y la cuenta de bytes por tira
    fseek(lee_arch,tiffd.etiqueta_tif[bustif(273)].des_dato+(cont_no_dir*4),SEEK_SET);
    fread((LONGINT *)&localiza,1,sizeof(localiza),lee_arch);
    fseek(lee_arch,tiffd.etiqueta_tif[bustif(279)].des_dato+(cont_no_dir*4),SEEK_SET);
    fread((LONGINT *)&cta_byte,1,sizeof(cta_byte),lee_arch);
    fseek(lee_arch,localiza,SEEK_SET);
    cont_no_dir++;
    if (bustif(259)!=0)                    // Verifica que el tipo de Compresion sea de los utilizados por TIFF
    {
      switch (tiffd.etiqueta_tif[bustif(259)].des_dato)
      {
```

```
case 1:                                                                    // Sin compresión
    aux=tiffd.etiqueta_tif[bustif(262)].des_dato;
    if ((aux==1) || (aux==3))                                   // Paleta en escala de grises o paleta RGB
     {
      for (cont=1;cont<=cta_byte;cont++)                                   // Para cada dato de la tira
       {
          ponx=(i % (xmaximo));                                            // Calcula posición x del pixel
          pony=(i / (xmaximo));                                            // Calcula posición y del pixel
          fread((BYTE *)&color,1,sizeof(color),lee_arch);                  // Lee el dato del pixel
          putpixel(ponx,pony,color);                                       // Pone el pixel en pantalla
          i++;                                                     // Incrementa cuenta de datos para la tira
       }
     }
case 2:                                                                    // Compresión CCITT ID
   {break;}                                                                // No soportado
case 3:                                                                    // Compresión CCITT Grupo 3
   {break;}                                                                // No soportado
case 4:                                                                    // Compresión CCITT Grupo 4
   {break;}                                                                // No soportado
case 5:                                                                    // Compresión LZW
   {break;}          // No soportado, ya que no solo se utiliza LZW si no otro metodo de compresión
case 6:                                                                    // Compresión JPG
   {break;}                                                                // No soportado
default:                                                                   // Ninguno de los anteriores
     if (tiffd.etiqueta_tif[bustif(259)].des_dato==32771)                  // Sin compresion
      {
       for (cont=1;cont<=cta_byte;cont++)                                  // Para cada dato de la tira
        {                                                                  // Calcula posición x del pixel
          ponx=(i % (xmaximo));
          pony=(i / (xmaximo));                                            // Calcula posición y del pixel
          fread((BYTE *)&color,1,sizeof(color),lee_arch);                  // Lee el dato del pixel
          putpixel(ponx,pony,color);                                       // Pone el pixel en pantalla
          i++;                                                     // Incrementa cuenta de datos por tira
        }
      }
     else                                                                  // Bits  emapquetados RLE
      if (tiffd.etiqueta_tif[bustif(259)].des_dato==32773)
       {
        aux=0;
        do
         {
          fread((BYTE *)&unbyte,1,sizeof(unbyte),lee_arch);                // Lee el primer dato
          aux++;                                                   // Incrementa cuenta de auxiliar
          if ((unbyte>=0) && (unbyte<=127))                                // Sin empaquetar
           {
            for (i=1;i<=unbyte+1;i++)                          // Lee tantas veces más uno el valor de un byte
             {
              fread((BYTE *)&color,1,sizeof(color),lee_arch);
              aux++;                                           // Incrementa cuenta auxiliar. datos leídos de la tira
              cont++;                                                  // Incrementa  cuenta de datos de imagen
              ponx=(cont % (xmaximo));                                 // Calcula posición x del pixel
              pony=(cont / (xmaximo));                                 // Calcula posición y del pixel
              putpixel(ponx,pony,color);                               // Pone el pixel en pantalla
             }
```

```
            }
            else                                                    // Empaqutados
             {
              cuantos=(unbyte*-1)+1;                                // Ver nota
              fread((BYTE *)&color,1,sizeof(color),lee_arch);
              aux++;                                  // Incrementa cuenta de datos leídos de la tira
              for (i=1;i<=cuantos;i++)                // Imprime el pixel tantas veces el valor de cuantos
               {
                cont++;                               // Incrementa cuenta de datos de imagen
                ponx=(cont % (xmaximo));              // Calcula posición x del pixel
                pony=(cont / (xmaximo));              // Calcula posición y del pixel
                putpixel(ponx,pony,color);            // Pone el pixel en pantalla
               }
             }
           } while (aux!=cta_byte);                   // Hasta leer todos los datos de la tira
          }
        break;
    }
  }
 } while (cont_no_dir!=no_dir);                        // Hasta leer todas las tiras
}
```

Una de las imágenes procesadas con el programa es: Uncomp.Tif que es un archivo TIF sin compresión.



uncomp.Tif

El programa da como resultado los siguientes datos:

```
Nombre de Archivo :uncomp.tif
Identificacion    :II      Posición del puntero :0
Versión           :42
OFFSET            :123442
Numero de etiquetas: 14
Siguiente imagen : 0
Nombre            etiqueta     Tipo          Version  Datacount    Dataoffset
Nvoti_Subarchi    254          Long     <4>  -- U5 U6  1            0
Ancho_imagen      256          Short    <3>  U4 U5 U6  1            365
Largo_imagen      257          Short    <3>  U4 U5 U6  1            333
Pues x pixel      277          Short    <3>  U4 U5 U6  1            1
Bits_muestra      258          Short    <3>  U4 U5 U6  1            8
Compresión        259          Short    <3>  U4 U5 U6  1            1
Inter_fotome      262          Short    <3>  U4 U5 U6  1            3
Desplaza_Tira     273          Long     <4>  U4 U5 U6  42           1560
Lineas_x_Tiras    278          Long     <4>  U4 U5 U6  1            0
cta_byte_Tira     279          Long     <4>  U4 U5 U6  42           1728
Resolucion_y      282          Rational <5>  U4 U5 U6  1            1544
Resolucion_y      283          Rational <5>  U4 U5 U6  1            1552
Uni_resolucion    296          Short    <3>  U4 U5 U6  1            2
Paleta            320          Short    <3>  -- U5 U6  768          0
```

Como puede apreciar las diferencias entre programar este formato en C o Pascal no son notorias, en éste último se utilizo una función de más (ifi) la cual no fue necesario estructurar en C.

Un detalle que no debe pasar desapercibido es la forma en que descomprime datos empaquetados en RLE, como se explicó en la nota al final del código en Pascal.

IV.8 Formato TGA

El formato TGA al igual que tif tiene varios tipos de imagenes, en TGA los que varia son sus colores que utiliza, el programa es capaz de leer imagenes en 16 millones de colores, pero, no lo despiegla en esos 16 millones de colores, en este formato se hace una función la cual reduce la cantidad de colores en solo 256 colores o menos, esto solo se hizo en tga, aunque esta función puede funcionar en cualquier otro programa de cualquier formato. La cabecera debe contener la estructura del formato, la cual utiliza 2 estructuras principales:

Cabecera_tga. Especifica los elementos de la cabecera del archivo.

Paleta_tga. Estructura para almacenar la paleta de colores.

El programa solo lee los tipos de imagen 0, 1, 9 y 10.

IV.8.1 En Lenguaje Pascal

En Pascal el programa se codifica de la siguiente manera.

```
Program Lee_TGA;          {Programa que lee el formato de archivo TGA
                           Realizado para el trabajo de tesis titulado:FORMATOS GRÁFICOS
                           Programadores :
                                          Cantero Ramirez Carlos
                                          Trejo Bonaga Marilu
                           Asesores :
                                          Manzo Salazar Itsmael
                                          Monterrosa Escobar Amilcar A.  }

Uses crt,dos,graph;

Type                                                    {Declaración de las dos estructuras}
 Cabecera_tga=Record                                    {Cabecera del archivo}
  identific,                                            {Tamaño del campo de identificación, 0 no incluye}
  color_map,                                            {Paleta de colores incluida,  si o no (1,0)}
  tipo_tga:byte;                                        {Tipo de imagen  TGA}
  color_orig,                                           {Indice de inicio de la paleta}
  color_long:Word;                                      {Longitud de la paleta}
  color_tam:Byte;                                       {Número de bits por entrada en la paleta, 16,24,32}
  Xminimo,yminimo,                                      {Coordenadas mínimas X y Y de la imagen}
  xmaximo,ymaximo:word;                                 {Coordenadas máximas X y Y de la imagen}
  NoColor,                                              {Número  de bits por pixel}
  describe:Byte;                                        {Descriptor de imagen}
 End;

 paleta_tga=record                                      {Paleta de colores}
   azul,verde,rojo: byte;                               {En  este orden}
 end;
```

```
Var
 tga:Cabecera_tga;                                   {Variable para almacenar la cabecera}
 tga_pal:Array [0..255] of paleta_tga;               {Varable para almacenar la paleta de colores}
 Ban_pal:Boolean;                                    {Variable para verificar si hay paleta o no}
 Archivo:string;                                     {Variable para almacenar el nombre del archivo}
 i,j,k,posicion:integer;                             {Variables auxiliares}
 Sololee:Word;
 Lee_Arch:File;                                      {Archivo a leer}

Function Inicializa(Driver,Modo:integer):boolean;    {Inicializa gráficos y reporta los errores que ocurran}
var
 ErrorG:integer;                                     No necesita explicación}
 x,y:word;
Begin
 if (Driver=16) Then
  Driver := InstallUserDriver('Svga256',nil);
 if Driver=grError Then
  Begin   halt(1);
  End;
 InitGraph(Driver, Modo,'d:\bp\programa');
   asm
     mov ax,000fh
     mov cx,8
     mov dx,8
     int 33h
    end;
   x:=getmaxx;
   y:=getmaxy;
   asm
     mov ax,x
     mov dx,ax
     mov ax,0007h
     mov cx,0h
     int 33h
     mov ax,y
     mov dx,ax
     mov ax,0008h
     mov cx,0h
     int 33h
    end;
  ErrorG:=graphResult;
  If ErrorG<>grok Then
   Begin
    Writeln('Error en graficos : ',GraphErrormsg(ErrorG));
    Inicializa:=False;
    Restorecrtmode;
   End
  Else
   Begin
    cleardevice;
    Inicializa:=True;
   End;
  End;
```

```
Procedure Cambia_color(color,r,v,a:Byte);                    {Cambia la paleta activa en el S.O.}
Begin                                                        {No necesita explicación}
 asm
  mov dx,3C8h
  mov al,color
  out dx,al
  inc dx
  mov al,r
  out dx,al
  mov al,v
  out dx,al
  mov al,a
  out dx,al
 End;
End;

Function Abre_lectura_TGA(Arch_imagen:String):Boolean;       {Prepara al archivo para su lectura}
Begin                                                        {No necesita explicación}
 Assign(Lee_Arch,arch_imagen);
{$i-}
  reset(Lee_arch,1);
{$i+}
 if ioresult = 0 Then
   Abre_lectura_TGA:=True
 else
  Begin
   Writeln('Error de apertura de archivo');
   Abre_lectura_TGA:=False;
  End;
 End;

 Procedure Lee_cabecera_TGA;                  {Lee los datos de la cabecera y los despliega en pantalla}
 Begin
   Blockread(lee_arch,tga,sizeof(tga),sololee);               {Almacena los datos de la cabecera}
   posicion:=Filepos(lee_arch);
   Gotoxy(01,01);Write('Nombre de Archivo :',archivo);
   Gotoxy(01,02);Write('Identificacion :');
   Gotoxy(25,02);Writeln(tga.Identific,'    ','Posición del puntero :',posicion);
   Gotoxy(01,03);Write('Mapa de color  :');
   Gotoxy(25,03);Writeln(tga.Color_map);
   Gotoxy(01,04);Write('Tipo de tga    :');
   Gotoxy(25,04);Writeln(tga.Tipo_tga);
   Gotoxy(01,05);Write('Color de origen:');
   Gotoxy(25,05);Writeln(tga.Color_orig);
   Gotoxy(01,06);Write('Color longitud :');
   Gotoxy(25,06);Writeln(tga.Color_Long);
   Gotoxy(01,07);Write('Tamaño de pixel:');
   Gotoxy(25,07);Writeln(tga.Color_tam);
   Gotoxy(01,08);Write('Xminimo        :');
   Gotoxy(25,08);Writeln(tga.xminimo);
   Gotoxy(01,09);Write('Yminimo        :');
   Gotoxy(25,09);Writeln(tga.yminimo);
   Gotoxy(01,10);Write('Xmaximo        :');
   Gotoxy(25,10);Writeln(tga.xmaximo);
```

```
  Gotoxy(01,11);Write('Ymaximo        :');
  Gotoxy(25,11);Writeln(tga.ymaximo);
  Gotoxy(01,12);Write('bits por pixel  :');
  Gotoxy(25,12);Writeln(tga.Nocolor);
  Gotoxy(01,13);Write('Descriptor     :');
  Gotoxy(25,13);Writeln(tga.Describe);
 End;

 Procedure Lee_Paleta_TGA;                                          {Lee la paleta del archivo}
 Begin
  seek(lee_arch,18);                                   {Busca la posición de la paleta, después de la cabecera}
  If tga.color_map=1 Then                                {Verifica que la paleta este incluida en el archivo}
   Begin                                                                    {Si hay paleta incluida, tipos 1 y 9}
     BlockRead(lee_arch,tga_pal,sizeof(tga_pal),sololee);                    {Almacena los datos de la paleta}
      ban_pal:=True;
   End
   else
     Begin                                                                  {No hay paleta incluida,tipos  2 y 10}
        ban_pal:=False;
     End;
 End;

 Procedure Pon_Paleta_TGA;                                          {Activa la paleta leida del archivo}
 Var
  i:integer;
 Begin
  If ban_pal Then
    For i:=0 to 255 do                                                      {Cambia la paleta de colores}
    Cambia_color(i,tga_pal[i].rojo shr 2,tga_pal[i].verde shr 2,tga_pal[i].azul shr 2);
 End;

 Procedure Lee_imagen_TGA;                                          {Lee y descomprime los datos de imagen}
 Var
   Cabeza,cuantos,unbyte:byte;
   Cont,Tam_imagen:longint;
   i,j,k,ponx,pony:integer;
   colores:Array[0..255,0..2] of Integer ;
   rgb:array[0..2] of byte;
   col116m,col216m:byte;

 Procedure Ini_pal;                          {Inicializa los valores para cada entrada en la paleta, tipos 2 y 10}
 Var
  i,j:Byte;
 Begin
  For i:=0 to 2 do                                                          {Para cada componente}
   For j:=0 to 255 do                                                          {Para cada entrada}
      Colores[j,i]:=-1;
 End;
```

{La siguiente funcion recibe los 3 colores primarios, realiza una operación con ellos y despues los almacena en un areglo de 256 elmentos de cada color, rojo verde y azul y asi mismo activa la paleta conforme entren nuevos datos a la paleta de colores provicional. se toman dos variables para hacer la aproximaxión, col116m y col216m supongamos que col116m=8 y col216m=4 estos valores seran modificados en el programa dependiendo del numero de bits utilizados en el tga.La operación consiste en aproximar cada uno de los tres

tonos a un numero cercano de color por ejemplo:Si se recibe la siguiente entrada en r , v , a =(10,15,30) la operación es la siguiente (se explica solo en rojo).

Si el residuo de la divicion de r con 8 es menor o igual a cuatro entonces
has a r=la parte entera de la divicion de r con 8 y multiplicala por 8
sino
has a r=la parte enter de la divicion de r con 5 sumale 1 y multiplicla por 8.

*Para el ejemplo r=10 residuo de r= 2 y su parte entera es 1 por consiguiente el residuo es menor o igual a cuatro, siendo ahora r= (r div 8)*8=(1)*8=8 , y para v y a es:*

*v= ((v div 8)+1)*8=(1+1)*8=16 hay residuo mayor a cuatro*
*a= ((a div 8)+1)*8=(3+1)*8=32 hay residuo mayor a cuatro*

Se observa que lo que hacemos es acotar colores. Supongamos ahora que hay una nueva entrada r, v, a = (9,19,29), si realizamos la operación tendremos:

*r= (r div 8)*8=(1)*8=8*
*v= ((v div 8))*8=(2)*8=16 con residuo igual a cuatro*
*a= ((a div 8)+1)*8=(3+1)*8=32 con residuo mayor a cuatro*

con estos resultados podemos obtener el color nuevo tomando el anterior que ya esta almacenado en el arreglo. Asi aseguramos un porcentaje elevado de colores semejenates en imagenes de 16 millones de colores reducidos a 256 colores}

```
Function Pos_color(r,v,a:byte):Byte;                    {Busca el índice de color, en la paleta provisional}
var
  i:byte;
  si_esta:boolean;
Begin
  i:=0;
  si_esta:=False;
  if (r+v+a)>0 Then
   Begin
     if (r mod col116m <= col216m ) then r:=(r div col116m)* col116m else r:=((r div col116m)+1) * col116m;
     if (v mod col116m <= col216m ) then v:=(v div col116m)* col116m else v:=((v div col116m)+1) * col116m;
     if (a mod col116m <= col216m ) then a:=(a div col116m)* col116m else a:=((a div col116m)+1) * col116m;
   End;
  Repeat
     if ((r=Colores[i,0]) and (v=Colores[i,1]) and (a=Colores[i,2])) Then Si_esta:=true;
      inc(i);
  Until (I>=255) OR ((colores[i-1,0]+colores[i-1,1]+colores[i-1,2])=-3) or (Si_esta);
  If (Si_esta) or (I>=255)  Then    { Si_esta determina que ya hay un color semejente en la paleta provicional}
      pos_color:=i-1
  Else
   Begin          {Almacena los nuevos colores en el arreglo provicional, ademas de activar al mismo tiempo}
      Colores[i-1,0]:=r;                                  {Almacena los nuevos colores en el arreglo provisional}
      Colores[i-1,1]:=v;
      Colores[i-1,2]:=a;
      Cambia_color(i-1,r shr 2,v shr 2,a shr 2);
     pos_color:=i-1;
   End;
  End;

 Begin                                                                          {Cuerpo principal de la función}
  tam_imagen:=tga.xmaximo;
  tam_imagen:=(tam_imagen*tga.ymaximo);                                         {Calcula tamaño de la imagen}
 col116m:=(tga.Nocolor*2)-8;
 col216m:=(tga.Nocolor*2)-8;
  Case tga.tipo_tga of
```

```
1:                                                          {Datos sin compresión, color mapped}
 Begin
   Seek(Lee_arch,18+sizeof(tga_pal));                       {Busca posición de datos de imagen}
   For cont:=0 to tam_imagen do                             {Hasta completar tamaño de la imagen}
   Begin
    ponx:=(cont mod (tga.xmaximo));                         {Calcula posición x del pixel}
    pony:=(cont div (tga.xmaximo));                         {Calcula posición y del pixel}
    blockread(Lee_Arch,unbyte,sizeof(unbyte),sololee)       {Lee dato del pixel}
    if (tga.Describe and 32)=0 Then                         {Verifica destino en pantalla del primer pixel}
     putpixel(ponx,tga.ymaximo-pony,unbyte);                {Pixel en esquina izquierda inferior de pantalla}
    Else
       putpixel(ponx,Pony,unbyte);                          {Pixel en esquina izquierda superior de pantalla}
    End;
 End;
2:                                                          {Sin compresión, True color 16 millones de colores}
 Begin
   Seek(Lee_arch,18);                                       {Se coloca en la posición de datos de imagen}
   Ini_pal;                                                 {inicializa paleta}
   For cont:=0 to tam_imagen do                             {Hasta completar tamaño de imagen}
    Begin
       ponx:=(cont mod (tga.xmaximo));                      {Calcula posición x del pixel}
       pony:=(cont div (tga.xmaximo));                      {Calcula posición y del pixel}
       blockread(Lee_Arch,rgb,sizeof(rgb),sololee);         {Lee dato del pixel}
       if (tga.Describe and 32)=0 Then
         putpixel(ponx,tga.ymaximo-pony,Pos_Color(rgb[2] ,rgb[1],rgb[0]))   { Esquina izquierda inferior}
       Else
          putpixel(ponx,pony,Pos_Color(rgb[2] ,rgb[1],rgb[0]));             {Esquina izquierda superior}
    End;
 End;
9:                                                          {Compresión RLE, color mapped}
 Begin
   Seek(Lee_arch,18+sizeof(tga_pal));                       {Se coloca en la posición de datos de imagen}
   i:=0;
   cont:=0;
   Repeat
      BlockRead(Lee_arch,Cabeza,1,sololee);                 {Lee primer dato}
      Case (Cabeza>=128) of                                 {Es Run-length}
       True:
        Begin
          Cuantos:=(Cabeza-128)+1;                          {Calcula las repeticiones}
           blockread(Lee_Arch,unbyte,sizeof(unbyte),sololee);  {Lee pixel a repetir}
          For k:=1 to cuantos do                            {Imprime tantas veces el valor de cuantos}
           Begin
             inc(i);                                        {incrementa cuenta de datos de imagen}
             inc(cont);
             ponx:=(cont mod (tga.xmaximo));                {Calcula posición x del pixel}
             pony:=(cont div (tga.xmaximo));                {Calcula posición y del pixel}
             if (tga.Describe and 32)=0 Then                {Verifica destino en pantalla del primer pixel}
               putpixel(ponx,tga.ymaximo-pony,unbyte)       {Pixel en esquina izquierda inferior}
              Else
               putpixel(ponx,pony,unbyte);                  {Pixel en esquina derecha superior }
         End;
       End;
```

```
False:                                                                  {Raw-Packet}
 Begin
   Cuantos:=Cabeza+1;                                                   {Calcula las repeticiones}
   For k:=1 to cuantos do                                       {Imprime tantas veces el valor de cuantos}
    Begin
       inc(i);
       inc(cont);
       ponx:=(cont mod (tga.xmaximo));                                  {Calcula posición x del pixel}
       pony:=(cont div (tga.xmaximo));                                  {Calcula posición y del pixel}
       blockread(Lee_Arch,unbyte,sizeof(unbyte),sololee);
       If (tga.Describe and 32)=0 Then                          {Verifica destino en pantalla del primer pixel}
          putpixel(ponx,tga.ymaximo-pony,unbyte)                        {Pixel en esquina izquierda inferior}
       Else
          putpixel(ponx,pony,unbyte);                                   {Pixel en esquina derecha  superior }
    End;
  End;
 End;
Until (Eof(Lee_arch) or (cont>=tga.xmaximo*tga.ymaximo));
End;
10:                                                         {Compresión RLE, True color (16 millones)}
 Begin
   Seek(Lee_arch,18);                                       {Se coloca en la posición de datos de imagen}
   Ini_pal;                                                                          {Inicializa paleta}
   i:=0;
   cont:=0;
   Repeat
    BlockRead(Lee_arch,Cabeza,1,sololee);
    Case (Cabeza>=128) of                                                          { Run-length}
    True:
     Begin
       Cuantos:=(Cabeza-128)+1;
        blockread(Lee_Arch,rgb,sizeof(rgb),sololee);
        For k:=1 to cuantos do
         Begin
           inc(i);
           inc(cont);
           ponx:=(cont mod (tga.xmaximo));
           pony:=(cont div (tga.xmaximo));
           If (tga.Describe and 32)=0 Then
            putpixel(ponx,tga.ymaximo-pony,Pos_Color(rgb[2] ,rgb[1],rgb[0]))      {Envia los 3 colores 16M}
           Else
             putpixel(ponx,pony,Pos_Color(rgb[2] ,rgb[1],rgb[0]));               {Envia los 3 colores 16M}
         End;
       End;
    False:                                                                  {Raw-Packet}
     Begin
       Cuantos:=Cabeza+1;
       For k:=1 to cuantos do
        Begin
          inc(i);
          inc(cont);
          ponx:=(cont mod (tga.xmaximo));
          pony:=(cont div (tga.xmaximo));
          blockread(Lee_Arch,rgb,sizeof(rgb),sololee);
```

```
            If (tga.Describe and 32)=0 Then
              putpixel(ponx,tga.ymaximo-pony,Pos_Color(rgb[2] ,rgb[1],rgb[0]))      {Envia los 3 colores 16M}
            Else
              putpixel(ponx,pony,Pos_Color(rgb[2] ,rgb[1],rgb[0]));                 {Envia los 3 colores 16M}
          End;
        End;
      End;
    Until (Eof(Lee_arch) or (cont>=tga.xmaximo*tga.ymaximo));          {Encuentra fin de archivo o se}
  End;                                                                 {iguala el tamaño de imagen}
 End;
End;


Begin                                                                  {Cuerpo del bloque principal del programa}
 clrscr;                                                               {Limpia pantalla}
 Archivo:='un24bfot.TGA';                                              {Asigna el nombre del archivo}
 If Abre_lectura_TGA(Archivo) Then                                     {Si puede abrir el archivo continua con el programa}
  Begin
   Lee_cabecera_TGA;                                                   {Lee y pone en pantalla la informacion de la cabecera}
   Lee_paleta_TGA;                                                     {Lee la paleta del archivo}
   Readkey;                                                            {Pausa para tomar nota de la cabecera}
   Inicializa(16,4);                                                   {Inicializa gráficos, 1024x768x256}
   Pon_paleta_TGAx                                                     {Activa la paleta leida del archivo}
   lee_imagen_TGA;                                                     {Descomprime la imagen y la visualiza en pantalla}
   Readkey;                                                            {Pausa para ver la imagen}
   close(Lee_arch);                                                    {Cierra el archivo}
   Restorecrtmode;                                                     {Recupera el modo texto}
   closegraph;                                                         {Cierra gráficos}
   clrscr;                                                             {Limpia pantalla}
  End;
End.
```

Una de las imágenes procesadas con el programa es: Un24bfot.Tga que es un archivo Tga sin compresión pero en 16 millones de colores.

Un24bfot.Tga

El programa da como resultado los siguientes datos:

```
Nombre de Archivo :un24bfot.TGA
Identificacion :          0      Posición del puntero :18
Mapa de color  :          0
Tipo de tga    :          2
Color de origen:          0
Color longitud :          0
Tamaño de pixel:          0
Xminimo        :          0
Yminimo        :          0
Xmaximo        :          640
Ymaximo        :          480
bit por pixel  :          24
Descriptor     :          0
```

Posteriormente se despliega la imagen.

IV.8.2 En Lenguaje C

En C el programa se codifica de la siguiente manera.

```
// Programa que lee el formato de archivo TGA
Realizado para el trabajo de tesis titulado:FORMATOS GRAFICOS
Programadores :
                    Cantero Ramirez Carlos
                    Trejo Bonaga Marilu
Asesores :
                    Manzo Salazar Itsmael
                    Monterrosa Escobar Amilcar A.

#include <fcntl.H>
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>
#include <dos.h>
#include <conio.h>
#include <io.h>

#define BYTE    unsigned char                    // Definición de constantes simbólicas
#define True  1
#define False 0
#define LONGINT unsigned long
#define WORD    unsigned int

                                                 // Definición de las estructuras
typedef struct                                   // Cabecera del archivo
 {                                               // Número de bytes que Identifican al archivo, 0 no incluye
   BYTE identific,
        color_map,                               // Paleta de colores incluido,  si o no (1,0)
        tipo_tga;                                // Tipo de imagen TGA
   WORD color_orig,                              // Indice de inicio de la paleta
        color_long;                              // Longitud de la paleta
   BYTE  color_tam;                              // Número de colores . 16,24,32
   WORD xminimo, yminimo,                        // Coordenadas mínimas X y Y de la imagen
        xmaximo, ymaximo;                        // Coordenadas máximas X y Y de la imagen
   BYTE   nocolor,                               // Número de bits por pixel
          describe;                              // Descriptor de imagen
}cabecera_tga;

typedef struct                                   // Paleta de colores
 {
 BYTE azul,verde,rojo;                           // En este orden
}paleta_tga;


char *archivo;                                   // Variable para almacenar el nombre del archivo
cabecera_tga tga;                                // Variable para almacenar la cabecera
paleta_tga tga_pal[256];                         // Variable para almacenar la paleta de colores
FILE *lee_arch;                                  // Archivo a leer
int ban_pal;                                     // Variable para verificar si hay paleta o no
int i,j,k;                                       // Variables auxiliares
fpos_t posicion;
```

```
 void Lee_Cabecera_TGA(void);                                   // Declaración de las funciones utilizadas
 int Inicializa(int Driver, int Modo);
 int Abre_lectura_TGA(char *arch_imagen);
 void Lee_Paleta_TGA(void);
 void Pon_Paleta_TGA();
 void Lee_imagen_TGA();
 void Cambia_color(BYTE color,BYTE r,BYTE v,BYTE a);

 void main ()                                                   // Cuerpo del bloque principal del  programa
 {                                                                                          // Limpia pantalla
   clrscr();
   archivo="rle8bciu.tga";                                                          // Asigna el nombre del archivo
   if (Abre_lectura_TGA(archivo))                          // Si puede abrir el archivo continua con el programa
   {
     Lee_Cabecera_TGA();                                  // Lee y despliega en pantalla los datos de la cabecera
     getch();                                                                  // Pausa para tomar nota de la cabecera
     Lee_Paleta_TGA();                                                                   // Lee la paleta del archivo
     Inicializa(16,4);                                                   // Inicializa el modo gráfico, 1024x768x256
     Pon_Paleta_TGA();                                                                      // Activa la paleta leída
     Lee_imagen_TGA();                                              // Descomprime y visualiza en pantalla la imagen
     getch();                                                                                // Pausa para ver la imagen
     fclose(archivo);                                                                              // Cierra el archivo
     restorecrtmode();                                                                   // Recupera el modo texto
     closegraph();                                                                                 // Cierra gráficos
     clrscr();                                                                                     // Limpia pantalla
  }
}

int Inicializa(int Driver, int Modo)                            // Inicializa gráficos y reporta los errores ocurridos
{                                                                                          // No necesita explicación
 int ErrorG;
 unsigned int x,y;
 union REGS regin,regout;
 if (Driver==16)  Driver = installuserdriver("Svga256",NULL);
 if (Driver==grError) exit;
 initgraph( &Driver, &Modo,"d:\borlandc\programa\exeobj");
  regin.x.ax=0x000f;
  regin.x.cx=0x8;
  regin.x.dx=0x8;
  int86(0x33,&regin,&regout);
  x=getmaxx();
  y=getmaxy();
  regin.x.ax=x;
  regin.x.dx=regin.x.ax;
  regin.x.ax=0x0007;
  regin.x.cx=0x0;
  int86(0x33,&regin,&regout);
  regin.x.ax=y;
  regin.x.dx=regin.x.ax;
  regin.x.ax=0x0008;
  regin.x.cx=0x0;
  int86(0x33,&regin,&regout);
  ErrorG=graphresult();
 if (ErrorG!=grOk)
```

```
 {
  printf("\n Error en graficos : %s",grapherrormsg(ErrorG));
  restorecrtmode;
  return 0;
 }
 else
 {
  cleardevice;
  return 1;
  }
}

void Cambia_color(BYTE color,BYTE r,BYTE v,BYTE a)          // Cambia la paleta activa en el S.O.
{                                                            // No necesita explicación
 asm mov dx,3C8h
 asm mov al,color
 asm out dx,al
 asm inc dx
 asm mov al,r
 asm out dx,al
 asm mov al,v
 asm out dx,al
 asm mov al,a
 asm out dx,al
}


int Abre_lectura_TGA(char *arch_imagen)                      // Prepara al archivo para su lectura
{                                                            // No necesita explicación
 lee_arch=fopen(arch_imagen,"rb");
 if (lee_arch==NULL)
  {return 0;}
 else
  {return 1;}
}

void Lee_Cabecera_TGA()                                      // Lee y almacena los datos de la cabecera
{
 fread((char *)&tga,1,sizeof(cabecera_tga),lee_arch);
 fgetpos(lee_arch,&posicion);                                // Despliega los datos de la cabecera en la pantalla
 gotoxy(1, 1);printf("Nombre de Archivo : %s ",archivo);
 gotoxy(1, 2);printf("Identificacion : %d  Posición del puntero : %d",tga.identific,posicion);
 gotoxy(1, 3);printf("Mapa de color  : %d",tga.color_map);
 gotoxy(1, 4);printf("Tipo de tga     : %d",tga.tipo_tga);
 gotoxy(1, 5);printf("Color de origen: %d",tga.color_orig);
 gotoxy(1, 6);printf("Color longitud : %d",tga.color_long);
 gotoxy(1, 7);printf("Tamaño de pixel: %d",tga.color_tam);
 gotoxy(1, 8);printf("Xminimo         : %d",tga.xminimo);
 gotoxy(1, 9);printf("Yminimo         : %d",tga.yminimo);
 gotoxy(1,10);printf("Xmaximo         : %d",tga.xmaximo);
 gotoxy(1,11);printf("Ymaximo         : %d",tga.ymaximo);
 gotoxy(1,12);printf("bits por pixel  : %d",tga.nocolor);
 gotoxy(1,13);printf("Descriptor      : %d",tga.describe);
}
```

```
 void Lee_Paleta_TGA()                                          // Lee los datos de la paleta
 {
  BYTE pal_sn;
  fseek(lee_arch,18,SEEK_SET);                  // Busca la posición de la paleta, después de la cabecera
  if (tga.color_map==1)                                  // Verifica que la paleta este incluida en el archivo
   {                                                                        // Si hay paleta incluida, tipos 1 y 9
    fread((char *)&tga_pal,1,sizeof(tga_pal),lee_arch);                // Lee y almacena los datos de la paleta
    ban_pal=True;
   }
  else
   ban_pal=False;                                                   // No hay paleta incluida, tipos 2 y 10
 }

 void Pon_Paleta_TGA()                                          // Activa la paleta leída del archivo
 {
  int i;
  if (ban_pal)
  for (i=0 ;i<=255;i++)                                                   // Cambia la paleta de colores
   Cambia_color(i,tga_pal[i].rojo >> 2,tga_pal[i].verde >> 2,tga_pal[i].azul >> 2);
}

int  colores[256][3];                                          // Lee y descomprime los datos de imagen

void Lee_imagen_TGA()
{
 int ponx,pony,numcols;
 LONGINT i,j,k,cont,tam_imagen;
 BYTE cabeza,unbyte,cuantos;
 char salte;
 BYTE rgb[3];
 BYTE col116m,col216m;

 void ini_pal();
 BYTE pos_color(BYTE r,BYTE v,BYTE a);
 int paletas;
 tam_imagen=tga.xmaximo;
 tam_imagen=(tam_imagen*tga.ymaximo);
 col116m:=(tga.Nocolor*2)-8;
 col216m:=(tga.Nocolor*2)-8;
 switch (tga.tipo_tga)
 {
  case 1:                                                   // Datos sin compresión, color mapped
   fseek(lee_arch,18+sizeof(tga_pal),SEEK_SET);                  // Busca posición de datos de imagen
   for (cont=0;cont<=tam_imagen;cont++)
    {
      ponx=(cont % (tga.xmaximo));                                          // Calcula posición x del pixel
      pony=(cont / (tga.xmaximo));                                          // Calcula posición y del pixel
      fread((char *)&unbyte,1,size of(unbyte),lee_arch);                          // Lee dato del pixel
      if ((tga.describe & 32)==0 )                              // Verifica destino en pantalla del primer pixel
       putpixel(ponx,tga.ymaxi mo-pony,unbyte);                 // Pixel en esquina izquierda inferior de pantalla
      else
       putpixel(ponx,pony,unbyte);                              // Pixel en esquina izquierda superior de pantalla
    }
 break;
```

```
case 2:                                          // Sin compresión, True color 16 millones de colores
   fseek(lee_arch,18,SEEK_SET);                  // Se coloca en la posición de datos de imagen
   ini_pal();                                    // Inicializa paleta
   for (cont=0;cont<=tam_imagen;cont++)          // Hasta completar tamaño de imagen
   {
     ponx=(cont % (tga.xmaximo));                // Calcula posisción x del pixel
     pony=(cont / (tga.xmaximo));                // Calcula posisción y del pixel
     fread((BYTE *)&rgb,1,3,lee_arch);           // Lee dato del pixel
     fgetpos(lee_arch,&posicion);
     if ((tga.describe & 32)==0)
       putpixel(ponx,tga.ymaximo-pony,pos_color(rgb[2] ,rgb[1],rgb[0]));   // Esquina izquierda inferior
     else                                        // Envia 3 colores en 16M
       putpixel(ponx,pony,pos_color(rgb[2] ,rgb[1],rgb[0]));               // Esquina izquierda superior
   }
  break;
 case 9:                                          // Compresión RLE, color mapped
   fseek(lee_arch,18+sizeof(tga_pal),SEEK_SET);   // Se coloca en la posición de datos de imagen
   i=0;
   cont=0;
  do
   {
     fread((char *)&cabeza,1,sizeof(cabeza),lee_arch);   // Lee el primer dato
      switch (cabeza>=128)
       {
        case True:                                // Run-length
          cuantos=(cabeza-128)+1;                 // Calcula repeticiones
          fread((char *)&unbyte,1,sizeof(unbyte),lee_arch);   // Lee pixel a repetir
          for (k=1;k<=cuantos;k++)                // Imprime tantas veces el valor de cuantos
          {
            i++;                                  // Incrementa cuenta de datos de imagen
            cont++;
            ponx=(cont % (tga.xmaximo));          // Calcula posisciones de los pixeles
            pony=(cont / (tga.xmaximo));
            if ((tga.describe & 32)==0)
              putpixel(ponx,tga.ymaximo-pony,unbyte);   // Pixel en esquina izquierda inferior
            else
              putpixel(ponx,pony,unbyte);         // Pixel en esquina izquierda superior
          }
          break;
          case False:                             // Raw-Packet
           cuantos=cabeza+1;                      // Calcula las repeticiones
           for (k=1;k<=cuantos;k++)               // Imprime tantas veces el valor de cuantos
           {
             i++;
             cont++;
             ponx=(cont % (tga.xmaximo));         // Calcula posisciones
             pony=(cont / (tga.xmaximo));
             fread((char *)&unbyte,1,sizeof(unbyte),lee_arch);
             if ((tga.describe & 32)==0)
               putpixel(ponx,tga.ymaximo-pony,unbyte);   // Pixel en esquina izquierda inferior
             else
               putpixel(ponx,pony,unbyte);        // Pixel en esquina izquierda superior
           }
        }
```

```
    }while (!feof(lee_arch)&&(cont<(tga.xmaximo*tga.ymaximo)));
    break;
  case 10:                                          // Compresión, RLE 16 millones de colores
    fseek(lee_arch,18,SEEK_SET);                    // Se coloca en la posición de datos de imagen
    ini_pal();                                      // Inicializa paleta
    i=0;
    cont=0;
    do
    {  fread((char *)&cabeza,1,sizeof(cabeza),lee_arch);
       switch (cabeza>=128)
       {
         case True:                                 // Run-length
          cuantos=(cabeza-128)+1;
          fread((BYTE *)&rgb,1,sizeof(rgb),lee_arch);
          for (k=1;k<=cuantos;k++)
          {
             i++;
             cont++;
             ponx=(cont % (tga.xmaximo));
             pony=(cont / (tga.xmaximo));
             if ((tga.describe && 32)==0)
                putpixel(ponx,tga.ymaximo-pony,pos_color(rgb[2] ,rgb[1],rgb[0]));   // Envia 3 colores 16M
             else
                putpixel(ponx,pony,pos_color(rgb[2],rgb[1],rgb[0]));                // Envia 3 colores 16M
          }
         break;
         case False:                                // Raw-Packet
           cuantos=cabeza+1;
           for (k=1;k<=cuantos;k++)
           {
              i++;
              cont++;
              ponx=(cont % (tga.xmaximo));
              pony=(cont / (tga.xmaximo));
              fread((BYTE *)&rgb,1,sizeof(rgb),lee_arch);
              if ((tga.describe & 32)==0)
                 putpixel(ponx,tga.ymaximo-pony,pos_color(rgb[2] ,rgb[1],rgb[0]));
              else
                 putpixel(ponx,pony,pos_color(rgb[2] ,rgb[1],rgb[0]));
           }
         break;
       };
    } while (!feof(lee_arch) && (cont<(tga.xmaximo*tga.ymaximo)));
   break;
  }
}

void ini_pal()                                      // Inicializa los valores para cad entrada en la paleta
{
 int i,j;
  for (i=0;i<=2;i++)
   for (j=0;j<=255;j++)
    colores[j][i]=-1;
}
```

```
BYTE pos_color(BYTE r,BYTE v,BYTE a)                                    // Busca el indice de color
{
  int i=0,suma=0;
  int si_esta=False;
  if ((r+v+a)>0)
  {
    if (r % col116m <= col216m ) r:=(r / col116m)* col116m; else r:=((r / col116m)+1)*col116m;
    if (v % col116m <= col216m ) v:=(v / col116m)* col116m; else v:=((v / col116m)+1)*col116m;
    if (a % col116m <= col216m ) a:=(a / col116m)* col116m; else a:=((a / col116m)+1)*col116m;
}
 do
  {
    if ((r==colores[i][0]) && (v==colores[i][1]) && (a==colores[i][2]))
        si_esta=True;
    i++;
    suma=(colores[i-1][0]+colores[i-1][1]+colores[i-1][2]);
  } while (!((i>=255) || (suma==-3) || (si_esta)));
 if ((si_esta) || (i>=255))
    return (i-1);
 else
 {                    // Almacena los nuevos colores en el arreglo provicional, ademas de activar al mismo tiempo
   colores[i-1][0]=r;                                    // Almacena los nuevos colores en el arreglo provisional
   colores[i-1][1]=v;
   colores[i-1][2]=a;
   Cambia_color(i-1,r >> 2,v >> 2,a >> 2);
   return (i-1);
   }
  }
```

Una de las imágenes procesadas con el programa es: RLE8bciu que es un archivo Tga con compresión RLE de 8 bits.

RLE8bciu.Tga

El programa da como resultado los siguientes datos:

```
Nombre de Archivo : RLE8bciu.tga
Identificacion : 0   Posición del puntero : 18
Mapa de color  : 1
Tipo de tga    : 9
Color de origen: 0
Color longitud : 256
Tamaño de pixel: 24
Xminimo        : 0
Yminimo        : 0
Xmaximo        : 640
Ymaximo        : 480
bit por pixel  : 8
Descriptor     : 0
```

El programa TGA puede leer imágenes de 16 millones pero las reduce a 256 para visualizarlas, si contamos con un driver que nos proporcione 16 millones de colores y además logramos accesar al puerto donde se localicen podremos leer imágenes en colores vivos no solo en TGA sino también todos los demás formatos.

CAPITULO

V

CAPITULO V

CONVERSIÓN

V.1 Conversión Entre Formatos de Archivos Gráficos

Como se explicó en el capítulo III los archivos gráficos son estructurados de acuerdo a las convenciones de un formato específico. La portabilidad que pueda ofrecer un formato determinado en una aplicación específica es muy importante. En ocasiones es más conveniente utilizar un formato en lugar de otro, ya sea por la facilidad con que pueda ser accesado por la aplicación, el espacio que ocupa en disco, la velocidad con que será leído, la flexibilidad de su cabecera, el manejo de colores, entre otras causas. De otro modo, se corre el riesgo de que el formato sea considerado como inútil u obsoleto para tal aplicación.

La mayoría de los formatos ofrecen características especiales que solo le conviene emplear a la aplicación que se este utilizando; por ejemplo, en Windows, en todas sus versiones hasta hoy conocidas, el papel tapiz sólo utiliza el formato BMP por dos razones, la primera de ellas por ser el formato propio de Windows, la segunda, porque es un formato que maneja sus datos casi en un bitmap plano (sin compresión), lo que hace fácil y rápido de leer y cargar en memoria.

La conversión entre formatos de archivos bitmap puede parecer fácil, pues poseen las características comunes necesarias para trasformar de un formato a otro. Se toman las características necesarias del formato fuente (el que se va a convertir), tamaño de la imagen, sin olvidar los datos de la imagen. El problema puede radicar al momento de programar el método de codificación del formato destino (al que se desea convertir).

La idea original de la conversión entre formatos, en este capitulo, era convertir de un formato específico a otro; por ejemplo del formato BMP al formato PCX, en el transcurso del desarrollo de este trabajo se pudo observar que era más factible y abierto tomar los datos generales de cada uno de los formatos para después emplearlos en la conversión a otro de ellos. Dichos datos se almacenan en un archivo temporal, el cual se manipula posteriormente para proceder a la conversión al formato destino.

Los datos indispensables para cualquier imagen son: sus máximas coordenadas en "x" y "y" (largo y ancho de la imagen), paleta de colores, datos de imagen, sin ningún tipo de compresión. El archivo temporal también guarda el nombre del archivo para asignarlo al archivo que se cree, así como la orientación de la imagen. El archivo temporal tendra la estructura mostrada en la figura V.1.

Datos de imagen en forma bitmap
Esctructura temporal

Figura V.1 Estructura del archivo temporal.

V.2 Estructura de Cada Programa

Con la teoría explicada en el capítulo III y la programación realizada en el capítulo IV tenemos las bases suficientes para poder realizar la conversión entre formatos. La estructura de los programas es sencilla y muy parecida a la del capítulo anterior, retomamos las funciones utilizadas anteriormente y le agregamos una más: Esc_imagen_XXX, que se encarga de codificar los datos de la imagen al formato destino.

La estructura general de los programas que realizan la compresión de imágenes se muestra en la figura V.2. Los signos XXX representan cada una de las iniciales de los formatos estudiados BMP, PCX, GIF, etc.

Abre_lectura_XXX. Prepara al archivo fuente para su lectura. Prepara al archivo temporal para su escritura.

Lee_Cabecera_XXX. Lee los datos de la cabecera y los despliega en pantalla. Almacena los datos de cabecera en la estructura temporal requeridos para la conversión del formato.

Lee_Paleta_XXX . Lee y almacena en un arreglo la paleta de colores de la imagen. Almacena la paleta de colores en la estructura temporal.

Inicializa. Inicializa el modo gráfico con una rutina en ensamblador.

Pon_Paleta_XXX. Activa la paleta de colores leída del archivo y almacenada en el arreglo.

Lee_Imagen_XXX. Realiza el método de descompresión, puede visualizar o no en pantalla la imagen. Guarda los datos de imagen (sin ningún tipo de compresión) y la estructura temporal en el archivo temporal.

Esc_Imagen_XXX. Realiza el método de compresión y guarda en un archivo de disco la imagen convertida en el formato destino.

Abre_lectura_XXX
Lee_Cabecera_XXX
Lee_Paleta _XXX
Pon_Paleta_ _XXX
Lee_imagen _XXX
Esc_imagen _XXX

Figura V.2 Estructura de cada programa .

Las funciones trabajan en forma independiente excepto Lee_imagen_XXX y Esc_imagen_XXX, la primera necesita los datos que proporcionan las demás funciones, y la segunda requiere de los datos que recauda Abre_Lectura_XXX, Lee_Paleta_XXX y Lee_imagen_XXX.

El diagrama de flujo de la estructura general de los programas se muestra en la figura IV.3.

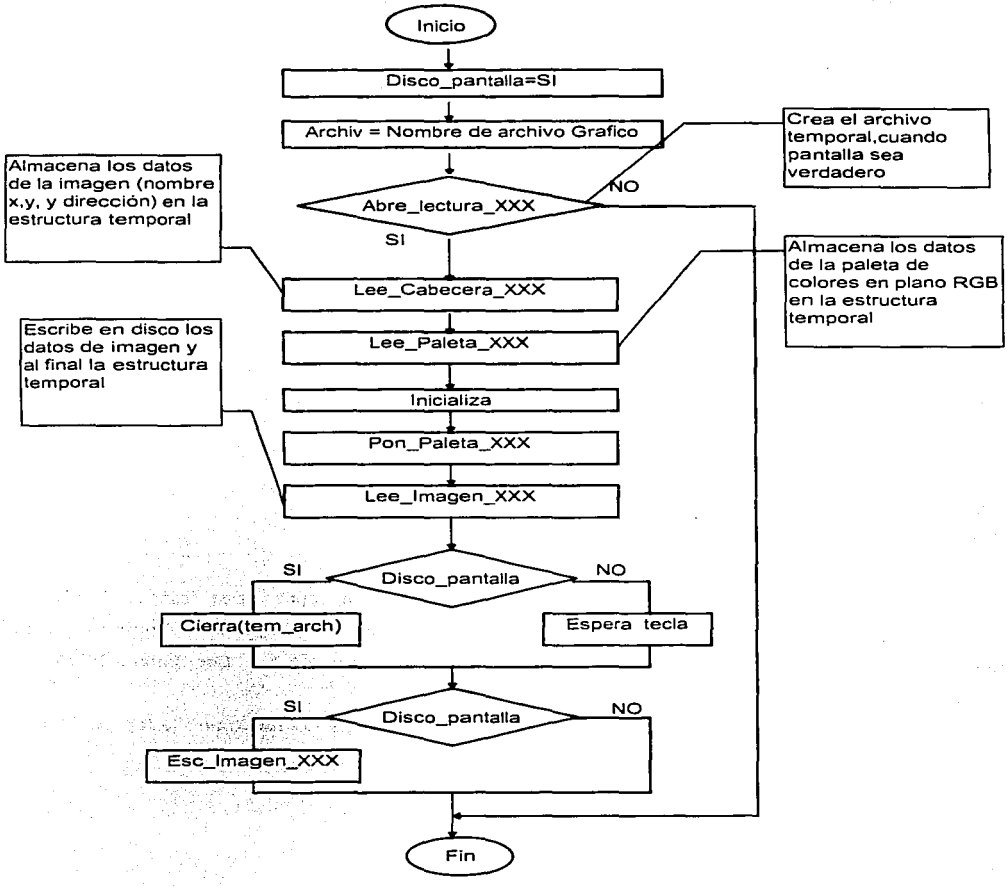


Figura V.3 Diagrama de flujo.

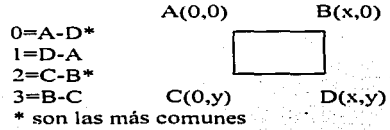
V.3 Funciones Comunes

Las funciones comunes utilizadas en la conversión de formatos son las mismas que en la lectura de los formatos por lo que solo las mencionaremos sin repetir el código tanto en Pascal como en C. Unicamente existe una nueva que se menciona más adelante. Ademas de estas funciones comunes existen algunas varibles que son necesarias para cada uno de los formatos. Estas variables son:

esc_arch ,tem_arch. Varibles utilizadas para crear el archvo temporal y el archivo donde se creara la nueva imagen.

- cab_aux. Varible que almacena la escrutura temporal la cual consiste en:
- nombre. Nombre del archivo sin extensión
- xmax,ymax. Largo y ancho de la imagen
- dir. Dirección de la imagen .

  Posibles direcciones



pal_rgb. Arreglo para la paleta, debe grabarse en plano RGB.

- disco_pantalla. Determina si se visualiza la imagen o se escribe en disco para proceder a su conversión, si disco pantalla es verdadero se escribe en disco si no es así se visualiza la imagen si crear en archivo temporal.

Y las funciones comunes son:

Inicializa (Driver, Modo). Función que inicializa el modo gráfico

Cambia_Color (Color, rojo, verde, azul). Función que modifica la paleta de colores activa.

Abre_lectura_XXX (Archivo_imagen). Función que abre el archivo de la imagen para verificar si es posible leerlo y que el programa no aborte.

Como ya se hizo mención, a ésta función se le agregó el código para crear el archivo temporal donde se almacenaran los datos del formato fuente, por lo que repetiremos su código.

En Pascal el código es el siguiente, la letra negrita indica la modificación:

```
Function Abre_lectura_bmp(arch_imagen:String):Boolean;
Var
 exito:Boolean;                                        {Determina si se puede leer y escribir en disco}

Begin
 Assign(lee_arch,arch_imagen);
{Si-}
 reset(lee_arch,1);
{Si+}
 if ioresult = 0 Then                                  {Si no hubo errores}
  Begin
   exito:=True;
   if (disco_pantalla) Then                            {Si disco pantalla es verdadero}
    Begin
     Assign(tem_arch,'T_e_m_p!.$$$');                  {Enlaza Tem_arch con el nombre T_e_m_p.$$$}
      {$i-}                                            {Desactiva errores de entrada y salida}
      rewrite(tem_arch,1);                             {Prepara Tem_arch para escribir, el número 1 indica
                                                       {que se abrirá para escribir registros con 1 byte de longitud}
      {$i+}                                            {Activa errores de entrada y salida}
     if (ioresult <>0) Then                            {Verifica si hay errores}
      exito:=False;                                    {Devuelve un valor de falso si existen errores}
    End;
   End
  else
   Begin
    Writeln('Error de apertura de archivo');
    exito:=false;
   End;
   Abre_lectura_bmp:=exito;
  End;
```

En C el código es el siguiente. la letra negrita indica la modificación:

```
int Abre_lectura_bmp(char *arch_imagen)
{
 int exito;                                            // Determina si se puede leer y escribir en disco
 lee_arch=fopen(arch_imagen,"rb");
 if (lee_arch==NULL)                                   // Si hay error
  {printf("Error de apertura de archivo \n");
   exito=False;
  }
 else
  {exito=True;
   if (disco_pantalla)                                 // Si se va a realizar la conversión
    {
     tem_arch=fopen("T_E_M_P!.$$$","wb");              // Abre el archivo para escritura en forma binaria
      if (tem_arch==NULL)                              // Si hay error
       exito=False;                                    // Devuelve un valor de falso si existen errores
    }
  }
  return exito;
 }
```

Solo_nombre (nombre_archivo). Función que separa el nombre del archivo de su extensión.

En lenguaje Pascal, es una función con el siguiente código:

```
Function Solo_nombre(archivo:string):String;                    {Almacena el nombre del archivo, sin extensión}
Var
 i:byte;
 nombre:string;
Begin
 i:=length(archivo);                                                          {Obtiene la longitud del nombre}
 nombre:='';                                                        {Inicializa la variable donde se almacena el nombre}
 repeat
    nombre:=concat(archivo[i],nombre);
    dec(i);
 until (archivo[i]=':') or (archivo[i]='\') or (i<=0);
 Solo_nombre:=copy(nombre,1,pos('.',nombre)-1);                  {Regresa el nombre del archivo a la función}
End;
```

En lenguaje C, la función Solo_nombre devuelve una cadena de caracteres, contiene el siguiente código:

```
char nom_arch[8];                                   // Toma una variable global para almacenar  el nombre
void Solo_nombre(char *archivo)                          // Almacena el nombre del archivo, sin extensión
{
 int i;
 i=strlen(archivo)-4;                                                    // Obtiene la longitud del nombre
 strncpy(nom_arch,archivo,i);                                            // Copia el nombre sin extensión
 nom_arch[i]=0;                                                                  // Asigna un nulo al final
}
```

Ya que los programas del capítulo anterior son la base para los programas de este capítulo y uno de los objetivos del trabajo no es saturar al lector de código fuente, solo por esta vez repetiremos todo el código fuente de BMP, resaltando las modificaciones hechas a algunos procesos que se realizan por igual en los formatos restantes. Por lo que en ellos solo se pondrá y explicará la función Esc_imagen_XXX; el lector debe ser capaz de realizar los cambios en cada uno de ellos basándose en lo explicado en BMP.

V.4 Compresión del Formato de Archivo BMP

La compresión de datos utilizando el método de codificación que emplea BMP (RLE) es fácil de programar, sin embargo no hay que olvidar sus características principales: solo puede codificar hasta 255 bytes iguales, usa pares de bytes para indicar que tipo de código se esta tratando (código delta, código de fin de línea, de fin de archivo o si está comprimido o no un byte), si el número de bytes comprimidos es impar se debe completar el par. Por último debe cuidar el termino de cada línea de bytes.

V.4.1 En Lenguaje Pascal

En Pascal el programa se codifica de la siguiente manera.

```
Program Lee_y_Esc_bmp;          {Programa que escribe el formato de archivo BMP.
                                 {Realizado para el trabajo de tesis titulado: FORMATOS GRÁFICOS
                                 Programadores :
                                                 Cantero Ramírez Carlos
                                                 Trejo Bonaga Marilu
                                 Asesores :
                                                 Manzo Salazar Itsmael
                                                 Monterrosa Escobar Amilcar A.   }
Uses crt,dos,graph;

type                                    {Estructura auxiliar para grabar en un archivo temporal}
 arch_temp=Record
 nombre:array[1..9] of char;                                    {Nombre del archivo}
 xmax,ymax:Longint;                                             {Largo y ancho de la imagen}
 dir  :Byte;                    {Dirección de la imagen          A(0,0)        B(x,0)
                                          0=A-D*
                                          1=D-A
                                          2=C-B*
                                          3=B-C          C(0,y)        D(x,y)
                                   * son las más comunes}
 pal_rgb:array [1..768] of byte;        {Arreglo para la paleta, debe grabarse en plano RGB}
End;

cabeza_bmp=record
  tipo: word;                                                   {Tipo de Archivo letras " BM "}
  tamano: longint;                                              {Tamaño del archivo en bytes}
  reservado1,reservado2: word;                                  {Reservados  con valor cero}
  tam_cabeza: longint;                                          {Desplazamiento de la cabeza del bitmap}
  tipo_bmp: longint;                                            {Determina tipo de BMP}
end;

paleta_bmpi=record
  azul,verde,rojo,reservado: byte;                              {En este orden}
end;
paleta_bmpc=record
  azul,verde,rojo: byte;                                        {En este orden}
end;

informe_bmp=record
 xmaximo: longint;                                              {Determina X máximo de la imagen}
 ymaximo:longint;                                               {Determina Y máximo de la imagen}
 planos: word;                                                  {Planos por pixel}
 contador: word;                                                {Contador}
 tipo_compresion: longint;                                      {Tipo de compresión}
 tamano_imagen: longint;                                        {Tamaño de la imagen}
 bitxmetrox: longint;                                           {Bits en x por metro}
 bitxmetroy: longint;                                           {Bits en y por metro}
 biclrusado: longint;                                           {No. de colores usados}
 biclrimport: longint;                                          {No. de colores más importantes}
end;
```

```
centro_bmp=Record
  xmaximo: word;                                  {Determina X máximo de la imagen}
  ymaximo: word;                                  {Determina Y máximo de la imagen}
  planos:  word;                                  {Planos por pixel}
  contador:word;                                  {Contador}
end;

const bi_rgb=0;                                   {Constantes para la compresión}
      bi_rle8=1;
      bi_rle4=2;

Var
 leyo,escribio:integer;
 archiv          :string;
 lee_arch,                                        {Archivo de lectura}
  esc_arch,                                       {Archivo de escritura}
   tem_arch   :File;                              {Archivo temporal}
 cab_aux      :arch_temp;                         {Cabecera auxiliar, almacena los datos de la imagen}
 bmp           :cabeza_bmp;
 bmpi          :informe_bmp;
 bmpc          :centro_bmp;
 bmp_palc     :Array [0..255] of paleta_bmpc;
 bmp_pali     :Array [0..255] of paleta_bmpi;
 disco_pantalla:Boolean;                          {Variable que indica si se convierte o no}
 posicion:LongInt;

 Function Inicializa(Driver,Modo:integer):boolean;   {Inicializa modo gráfico y reporta los errores}
 var                                                 {que ocurran}
  ErrorG:integer;                                    {Esta función no necesita explicación alguna}
  x,y:word;
 Begin
  if (Driver=16) Then
   Driver := InstallUserDriver('Svga256',nil);
  if Driver=grError Then
   Begin
    halt(1);
   End;
  InitGraph(Driver, Modo,'d:\bp\programa');
     asm
       mov ax,000fh
       mov cx,8
       mov dx,8
       int 33h
      end;
     x:=getmaxx;
     y:=getmaxy;
     asm
       mov ax,x
       mov dx,ax
       mov ax,0007h
       mov cx,0h
       int 33h
       mov ax,y
       mov dx,ax
```

```
      mov ax,0008h
      mov cx,0h
      int 33h
    end;
 ErrorG:=graphResult;
 If ErrorG<>grok Then
  Begin
   Writeln('Error en graficos : ',GraphErrormsg(ErrorG));
   Inicializa:=False;
   Restorecrtmode;
  End
 Else
  Begin
   cleardevice;
   Inicializa:=True;
  End;
End;

Procedure Cambia_color(color,r,v,a:Byte);          {Modifica la paleta de colores activa en el S.O.}
Begin                                                        {Esta función no necesita explicación}
 asm
  mov dx,3C8h
  mov al,color
  out dx,al
  inc dx
  mov al,r
  out dx,al
  mov al,v
  out dx,al
  mov al,a
  out dx,al
 End;
End;

Function Solo_nombre(archivo:string):String;       {Almacena el nombre sin extensión del archivo}
Var                                                        {Esta función no necesita explicación alguna}
 i:byte;
 nombre:string;
 Begin
 i:=length(archivo); nombre:='';
 repeat
  nombre:=concat(archivo[i],nombre);
  dec(i);
 until (archivo[i]=':') or (archivo[i]='\') or (i<=0);
 Solo_nombre:=copy(nombre,1,pos('.',nombre)-1);
 End;

Function Abre_lectura_bmp(arch_imagen:String):Boolean;          {Prepara al archivo para su lectura}
Var                                                        { y al archivo temporal para su escritura}
 exito:Boolean;                                            {Esta función no necesita explicación}
Begin
 Assign(lee_arch,arch_imagen);
  {Si-}
```

```
reset(lee_arch,1);
  {$i+}
if ioresult = 0 Then
 Begin
  exito:=True;
  If (disco_pantalla) Then
   Begin
    Assign(tem_arch,'T_e_m_p!.$$$');
      {$i-}
     rewrite(tem_arch,1);
      {$i+}
    if (ioresult <>0) Then
      exito:=False;
    End;
  End
 else
  Begin
   Writeln('Error de apertura de archivo');
   exito:=false;
  End;
   Abre_lectura_bmp:=exito;
 End;

 Procedure Lee_Cabecera_BMP;                {Lee los datos de la cabecera y los almacena en una estructura}
  Var
   i:byte;
   nom_arch:String[12];
  Begin
   blockread(lee_arch,bmp,sizeof(bmp),leyo);
   posicion:=filepos(lee_arch);
   nom_arch:=Solo_nombre(archiv)+chr(0);                {Asigna el nombre del archivo sin extensión}
   With bmp do
    Begin
     Gotoxy(01,01);Write('Cabecera del archivo BMP de MicroSoft Windows ',archiv);
     Gotoxy(01,02);Write('Tipo de bmp          :');
     Gotoxy(25,02);Writeln(chr(tipo),chr(tipo shr 8),'     ','Posición del puntero :',posicion);
     Gotoxy(01,03);Write('Tamaño del arch      :');
     Gotoxy(25,03);Writeln(tamano);
     Gotoxy(01,04);Write('Tam_cabeza           :');
     Gotoxy(25,04);Writeln(tam_cabeza,'  ',reservado1,'  ',reservado2);
     Gotoxy(01,05);Write('Tipo de bmp          :');Gotoxy(25,05);Writeln(tipo_bmp);
    End;
   If (bmp.tipo_bmp=40) Then
    Begin
     blockread(lee_arch,bmpi,sizeof(bmpi),leyo);
     With bmpi do
      Begin
       Gotoxy(01,06);Write('Xmáximo              :');Gotoxy(25,06);Writeln(xmaximo);
       Gotoxy(01,07);Write('Ymáximo              :');Gotoxy(25,07);Writeln(ymaximo);
       Gotoxy(01,08);Write('Planos               :');Gotoxy(25,08);Writeln(planos);
       Gotoxy(01,09);Write('Bits por pixel       :');Gotoxy(25,09);Writeln(contador);
       Gotoxy(01,10);Write('Tipo de compresión  :');Gotoxy(25,10);Writeln(tipo_compresion);
       Gotoxy(01,11);Write('Tamaño de imagen    :');Gotoxy(25,11);Writeln(tamano_imagen);
       Gotoxy(01,12);Write('bi x                :');Gotoxy(25,12);Writeln(bitxmetrox);
```

```
    Gotoxy(01,13);Write('bi y                :');Gotoxy(25,13);Writeln(bitxmetroy);
    Gotoxy(01,14);Write('bi clr used         :');Gotoxy(25,14);Writeln(biclrusado);
    Gotoxy(01,15);Write('bi clr import       :');Gotoxy(25,15);Writeln(biclrimport);
          {El siguiente bloque almacena los datos de la cabecera bmpi  requeridos en la estructura cab_aux}
    if (disco_pantalla) Then
     Begin
      for i:=1 to 9 do
       cab_aux.nombre[i]:=nom_arch[i];
       cab_aux.xmax:=xmaximo;
       cab_aux.ymax:=ymaximo;
       cab_aux.dir:=2;                                  {Una imagen BMP[1] se despliega de la parte inferior}
     End;                                                                  {izquierda a la superior derecha}
    End;
   End
  Else
   Begin
    blockread(lee_arch,bmpc,sizeof(bmpc),leyo);
    With bmpc do
     Begin
      Gotoxy(01,06);Write('xmaximo          :');Gotoxy(25,06);Writeln(xmaximo);
      Gotoxy(01,07);Write('ymaximo          :');Gotoxy(25,07);Writeln(ymaximo);
      Gotoxy(01,08);Write('planos           :');Gotoxy(25,08);Writeln(planos);
      Gotoxy(01,09);Write('bits por pixel   :');Gotoxy(25,09);Writeln(contador);
          {El siguiente bloque almacena los datos de la cabecera bmpc  requeridos en la estructura cab_aux}
  if (disco_pantalla) Then
      Begin
       for i:=1 to 9 do
        cab_aux.nombre[i]:=nom_arch[i];
        cab_aux.xmax:=xmaximo;
        cab_aux.ymax:=ymaximo;
        cab_aux.dir:=2;                                 {Una imagen BMP se despliega de la parte inferior}
      End;                                                                  {izquierda a la superior derecha}
     End;
   End;
 End;

Procedure Lee_Paleta_BMP;                          {Lee la paleta de colores y la almacena en un arreglo}
Var                                                                        así como en la estructura cab_aux}
 contp:Integer;
Begin
 If (bmp.tipo_bmp=40) Then
  Begin
   blockread(lee_arch,bmp_pali,sizeof(bmp_pali),leyo);
                          {El siguiente bloque almacena la paleta en el arreglo de la estructura cab_aux}
   if (disco_pantalla) Then
     for contp:=0 to 255 do
     begin
      cab_aux.pal_rgb[(contp*3)+1]:=bmp_pali[contp].rojo;
      cab_aux.pal_rgb[(contp*3)+2]:=bmp_pali[contp].verde;
      cab_aux.pal_rgb[(contp*3)+3]:=bmp_pali[contp].azul;[2]
```

[1] La colocacion de este bloque de codigo debe ser en el lugar donde ya se cuente con todos los datos necesarios para almacenarlos.

```
    end;
  End
 Else
  Begin
   blockread(lee_arch,bmp_palc,sizeof(bmp_palc),leyo);
                                    {El siguiente bloque almacena la paleta en el arreglo de la estructura cab_aux}
   if (disco_pantalla) Then
    for Contp:=0 to 255 do
     begin
      cab_aux.pal_rgb[(contp*3)+1]:=bmp_palc[contp].rojo;
      cab_aux.pal_rgb[(contp*3)+2]:=bmp_palc[contp].verde;
      cab_aux.pal_rgb[(contp*3)+3]:=bmp_palc[contp].azul;
     end;
  End;
End;

Procedure Pon_Paleta_BMP;                                   {Activa la paleta de colores del archivo}
Var
 i:integer;
Begin
 If (bmp.tipo_bmp=40) Then
  For i:=0 to 255 do
    with bmp_pali[i] do
     Cambia_color(i,rojo shr 2,verde shr 2,azul shr 2)
 Else
  For i:=0 to 255 do
    with bmp_palc[i] do
     Cambia_color(i,rojo shr 2,verde shr 2,azul shr 2);
End;

Procedure Lee_imagen_BMP;                          {Descomprime los datos de imagen y los almacena}
Var                                                                          {en el archivo temporal}
 ponx,pony,numcols,bit_linea:integer;
 i,j,k,cont,tam_imagen:Longint;
 unbyte,cuantos:byte;
 salte:boolean;
Begin
 numcols:=256;
 seek(lee_arch,bmp.tam_cabeza);
 If (bmp.tipo_bmp=40) Then
  Begin
      Case bmpi.tipo_compresion of
      0:
       Begin
        bmpi.xmaximo:=Trunc((bmp.tamano-bmp.tam_cabeza)/bmpi.ymaximo);
        cab_aux.xmax:=bmpi.xmaximo;
        tam_imagen:=((bmpi.ymaximo)*bmpi.xmaximo);
        for i:=0 to (tam_imagen-1) do
         begin
          if numcols=256 then
           begin
```

[2] Recuerde colocar el código en un lugar estratégico, en el caso de la paleta se debe almacenar una vez leída del archivo y se debe colocar en plano RGB independientemente en el plano que se encuentre.

```
        blockread(lee_arch,unbyte,1,leyo);
       end
      else
      if numcols=16 then
        begin
        end
      else
       if numcols=2 then
        begin
        end;
     ponx:=(i mod (bit_linea));
     pony:=(i div (bit_linea));
     if (disco_pantalla) then[3]
       blockwrite(tem_arch,unbyte,1,escribio)          {Almacena el dato en el archivo temporal}
     else
       putpixel(ponx,bmpi.ymaximo-pony,unbyte);
   End;
 End;
1:
 Begin
  i:=0;j:=0;cont:=0;k:=0;salte:=false;
  Repeat
    blockread(lee_arch,unbyte,sizeof(unbyte),leyo);
    blockread(lee_arch,cuantos,sizeof(cuantos),leyo);
    if( unbyte=0) Then
     Begin
      case cuantos of
        0:
         Begin                                                          {Fin de línea}
          inc(j);
          i:=0;
         End;
        1:
         Begin                                                          {Fin de archivo}
          salte:=True;
         End;
        2:
         Begin                                                          {Delta}
         End;
        else
         Begin                                                          {Sin codificar}
          for k:=1 to cuantos do
           Begin
            inc(i);
            inc(cont);
            blockread(lee_arch,unbyte,sizeof(unbyte),leyo);
            ponx:=(cont mod (bmpi.xmaximo));
            pony:=(cont div (bmpi.xmaximo));
            If (disco_pantalla) Then
              blockwrite(tem_arch,unbyte,1,escribio)        {Almacena el dato en el archivo temporal}
            else
```

[3] En este código simplemente nos basamos en la variable disco_pantalla para saber si mandamos el byte al disco ó a la pantalla.

```
            putpixel(ponx,bmpi.ymaximo-pony,unbyte);
          End;
          If cuantos mod 2 <> 0 Then
           blockread(lee_arch,unbyte,sizeof(unbyte),leyo);
         End;
       End;
      End
     Else
      Begin
        for k:=1 to unbyte do
         Begin
          inc(i);
          inc(cont);
          ponx:=(cont mod (bmpi.xmaximo));
          pony:=(cont div (bmpi.xmaximo));
          if (disco_pantalla) Then
           blockwrite(tem_arch,cuantos,1,escribio)          {Almacena el dato en el archivo temporal}
          else
           putpixel(ponx,bmpi.ymaximo-pony,cuantos);
         End;
       End;
     Until (Salte);
    End;
   2:                                                                          {RLE de 4 bits}
    Begin
    End;
   End;
 End
 Else                                                                          {Es Windows v2.0, OS/2  v1.0}
 Begin
   bmpc.xmaximo:=Trunc((bmp.tamano-bmp.tam_cabeza)/bmpc.ymaximo);
   cab_aux.xmax:=bmpc.xmaximo;
   tam_imagen:=bmpc.ymaximo;
   tam_imagen:=(tam_imagen*bmpc.xmaximo);
    For i:=0 to (Tam_imagen-1) do
     begin
      if numcols=256 then
         blockread(lee_arch,unbyte,1,leyo);
      else
       if numcols=16 then
         begin
         end
       else
         if numcols=2 then
           begin
            end;
       ponx:=(i mod (bmpc.xmaximo));
       pony:=(i div (bmpc.xmaximo));
       if (disco_pantalla) Then
        blockwrite(tem_arch,unbyte,1,escribio)              Almacena el dato en el archivo temporal}
       else  putpixel(ponx,bmpc.ymaximo-pony,Unbyte);
     end;
  End;
 If (disco_pantalla) Then
```

```
  blockwrite(tem_arch,cab_aux,sizeof(cab_aux),escribio);
                                    {Una vez almacenados los datos de imagen en el archivo temporal,
                                     { ahora al final se  almacena  la esctrucuta temporal cab_aux}
End;

Procedure Esc_imagen_BMP;                                   {Realiza la conversión al formato BMP}
Type
 buffer=Array [1..4096] of byte;                            {Arreglo para almacenar los datos}

Var
 ponx,pony,numcols:integer;
 i,j,k,cont,tam_imagen:Longint;
 aux,                                                       {Byte auxiliar}
  unbyte,                                                   {Byte último}
   cuantos,                                                 {Contador de datos comprimidos}
    byteAnt,                                                {Byte antepenúltimo}
     bytepen,                                               {Byte penúltimo}
      sin_comp,                                             {Contador de datos sin compresión}
       bandera:byte;                                        {Bandera tipo de compresión}
 salte:boolean;
 nom_bmp:String;                                            {Nombre del archivo}
 arr_bytes:Array [1..256] of byte;                          {Arreglo de bytes comprimidos}
 buf:^buffer;                                               {Variable para el arreglo de datos}


Procedure pos_sig_bloque(var j:longint);                    {Determina la posición[4] del siguiente bloque de datos}
Begin
 inc(j);                                                    {Incrementa cuenta de renglones}
 inc(k,cab_aux.xmax);                                       {Incrementa cuenta de bloques}
 if (cab_aux.dir=2) Then
   seek(tem_arch,j*cab_aux.xmax)                            {Asigna la siguiente posición a partir del inicio}
 else
   seek(tem_arch,filesize(tem_arch)-k);                     {Asigna la siguiente posición a partir del final}
 gotoxy(28,1);write((j*100/cab_aux.ymax):3:0);              {Calcula el porcentaje de compresión}
End;

Procedure RLE;                                              {Procedimiento para comprimir con RLE}
 Begin
  blockwrite(esc_arch,cuantos,1,escribio);                  {Escribe el número de bytes comprimidos}
  blockwrite(esc_arch,byteant,1,escribio);                  {Escribe el valor de dicho byte}
  cuantos:=0;                                               {Inicializa cuantos}
 End;


Procedure RAW;                                              {Procedimiento para comprimir con RAW}
Var
 cont:integer;
Begin
 if (sin_comp)>=3 Then                                      {Si datos sin comprimir es mayor o igual a 3}
  Begin
   aux:=0;
```

[4] Este procedimiento coloca el puntero del archivo calculando la posición por línea de acuerdo a las dimensiones de la imagen.

```
  blockwrite(esc_arch,aux,1,escribio);                    {Escribe el byte que indica datos sin comprimir}
  blockwrite(esc_arch,sin_comp,1,escribio);               {Escribe el número de bytes sin comprimir}
  for cont:=1 to sin_comp do
    blockwrite(esc_arch,arr_bytes[cont],1,escribio);      {Escribe cada uno de los bytes sin comprimir}
  if (sin_comp) mod 2=1 Then                              {Si el número de bytes sin comprimir es impar}
    blockwrite(esc_arch,aux,1,escribio);                  {Escribe otro byte para completar el par}
 End
 else                                                     {Datos sin comprimir es menor a 3}
  for cont:=1 to sin_comp do                              {Para cada dato sin comprimir}
   Begin
    aux:=1;
    blockwrite(esc_arch,aux,1,escribio);                  {Escribe byte que indica un byte compreso}
    blockwrite(esc_arch,arr_bytes[cont],1,escribio);      {Escribe el valor de dicho byte}
   End;
End;


Begin                                                     {El siguiente bloque abre el archivo temporal para su lectura}
 Assign(tem_arch,'T_e_m_p!.SSS');
 {Si-}
 reset(tem_arch,1);
  {Si+}
 if (ioresult=0) Then                                     {Si no hay errores continúa}
  Begin
  tam_imagen:=filesize(tem_arch)-786;                     {Calcula el tamaño de la imagen}
  Seek(tem_arch,filesize(tem_arch)-786);                  {Se coloca en el inicio de la cabecera auxiliar}
  blockRead(tem_arch,cab_aux,sizeof(cab_aux),leyo);       {Lee y almacena la cabecera auxiliar}
  nom_bmp:='';i:=1;                                       {Inicializa variables}
                                                          {El siguiente bloque asigna X_nombre al archivo de escritura}
  While (cab_aux.nombre[i]<>chr(0)) do
   Begin
    nom_bmp:=concat(nom_bmp,cab_aux.nombre[i]);
    inc(i);
   End;
  If Length(nom_BMP)>=5 Then
    nom_bmp:='X_'+copy(nom_bmp,1,6)+'.BMP'
  else
    nom_bmp:='X_'+nom_bmp+'.BMP';
  Assign(esc_arch,nom_bmp);                               {El siguiente bloque abre el archivo destino para su escritura}
   {Si-}
   rewrite(esc_arch,1);
   {Si+}
  if (ioresult=0) then                                    {Si no hay errores continúa con el proceso}
   Begin                                                  {El siguiente bloque asigna los datos de la cabecera}
    With bmp do
     Begin
      tipo:=19778;                                        {Asigna tipo de archivo, letras "BM", identificador}
      tamano:=0;                                          {Se asigna hasta tener el archivo completo}
      tam_cabeza:=1078;                                   {Paleta de colores incluida}
      reservado1:=0;                                      {Siempre cero}
      reservado2:=0;                                      {Siempre cero}
      tipo_bmp:=40;                                       {Tipo 40, Windows 3.x o NT}
     End;
    With bmpi do                                          {Cabecera informe BMP}
```

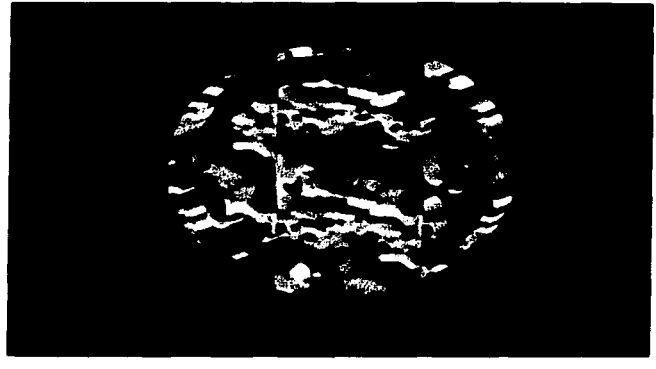
```
Begin
 xmaximo:=cab_aux.xmax;                                   {X máximo de la imagen}
 ymaximo:=cab_aux.ymax;                                   {Y máximo de la imagen}
 planos:=1;                                               {Solo un plano de color RGB}
 contador:=8;                                             {8 bits por píxel, 256 colores}
 tipo_compresion:=1;                                      {RLE, método de compresión a utilizar}
 tamano_imagen:=0;                                        {Se asigna hasta tener el archivo completo
                                                          {este valor es = tamano-tam_cabeza;}
 bitxmetrox:=0;                                           {Valor con cero}
 bitxmetroy:=0;                                           {Valor con cero}
 biclrusado:=256;                                         {Utiliza todos los colores}
 biclrimport:=256;                                        {Todos los colores son importantes}
End;
                                                          {Reserva lugares para completar cabecera}
blockwrite(esc_arch,bmp ,sizeof(bmp) ,escribio);
blockwrite(esc_arch,bmpi,sizeof(bmpi),escribio);
unbyte:=0;                                                {Escribe paleta de colores}
for i:=0 to 255 do
 Begin
  blockwrite(esc_arch,cab_aux.pal_RGB[(i*3)+3],1,escribio);   {Azul}
  blockwrite(esc_arch,cab_aux.pal_RGB[(i*3)+2],1,escribio);   {Verde}
  blockwrite(esc_arch,cab_aux.pal_RGB[(i*3)+1],1,escribio);   {Rojo}
  blockwrite(esc_arch,unbyte,1,escribio);                     {Reservado}
 End;                                                     {Inicia proceso de compresión}
 new(buf);                                                {Asigna memoria}
 aux:=0; i:=0; j:=0;                                      {Auxiliar, cuenta de columnas y cuenta de renglones}
 gotoxy(1,1);write('Porcentaje de conversión :    %');
 k:=(786+bmpi.xmaximo);                                   {Localiza la posición de acuerdo con la dirección de la imagen}
 if (cab_aux.dir=2) Then
    Seek(tem_arch,0)                                      {Se coloca al inicio del archivo}
 else
    Seek(tem_arch,filesize(tem_arch)-k);                  {Se coloca al final del archivo menos la posición anterior}
 BlockRead(tem_arch,buf^,cab_aux.xmax,leyo);              {Lee un bloque de datos de longitud xmáximo}
 Repeat                                                   {Repite el siguiente bloque}
  i:=1;                                                   {Inicializa variables}
  sin_comp:=0; cuantos:=0;
  bandera:=0; sin_comp:=1;
  Repeat                                                  {Repite el siguiente bloque}
   byteant:=buf^[i];                                      {Asigna byteant}
   bytepen:=buf^[i+1];                                    {Asigna bytepen}
   unbyte:=buf^[i+2];                                     {Asigna unbyte}
   If (byteant<>bytepen) or (bytepen<>unbyte) Then        {Si algún byte es diferente}
    Begin
     if bandera=1 Then                                    {Hay datos comprimidos}
      Begin
       cuantos:=cuantos+2;                                {Incrementa cuantos}
       i:=i+1;                                            {Incrementa columnas}
       RLE;                                               {Escribe RLE}
       bandera:=0;                                        {Inicializa bandera}
      End
    else                                                  {Almacena datos sin comprimir}
     Begin
      arr_bytes[sin_comp]:=byteant;
      inc(sin_comp);
```

```
        if sin_comp>=255 Then                                   {Verifica sean menor a 256}
         Begin
           dec(sin_comp);                                       {Decrementa sin comprimir}
           RAW;                                                 {Escribe RAW}
           sin_comp:=1;                                         {Inicializa sin comprimir}
         End;
       bandera:=2;                                              {Asigna bandera, sin comprimir}
      End;
     cuantos:=0;                                                {Iniciliza cuantos}
     End
    Else                                                        {Hay bytes iguales}
     Begin
      if bandera=2 Then                                         {Revisa si se envían bytes sin comprimir}
       Begin
         dec(sin_comp);                                         {Decrementa sin comprimir}
         RAW;                                                   {Escribe RAW}
       End;
      bandera:=1;                                               {Cuenta 1 byte compreso}
      inc(cuantos);                                             {Incrementa cuantos}
      if cuantos>=255 Then RLE;                                 {Verifica sea menor a 256}
      sin_comp:=1;                                              {Inicializa sin comprimir}
     End;
    inc(i);                                                     {Incrementa cuenta de columnas}
   Until (i>=cab_aux.xmax-2);                                   {Hasta terminar dicho renglón}
   if bandera=2 Then                                            {Revisa si un conteo quedo sin terminar}
    Begin
     dec(sin_comp);                                             {Decrementa sin comprimir}
     RAW;                                                       {Escribe RAW}
    End;
   if bandera=1 Then RLE;                                       {Escribe los últimos bytes}
   sin_comp:=cab_aux.xmax-i+1;                                  {Asigna posición del byte}
   arr_bytes[1]:=buf^[i];                                       {Asigna bytes a comparar}
   arr_bytes[2]:=buf^[i+1];
   arr_bytes[3]:=buf^[i+2];
   RAW;                                                         {Escribe RAW}
   aux:=0;                                                      {Escribe fin de línea " 00h 00h"}
   blockwrite(esc_arch,aux,1,escribio);
   blockwrite(esc_arch,aux,1,escribio);
   pos_sig_bloque(j);                                           {Calcula posición del siguiente bloque}
   BlockRead(tem_arch,buf^,cab_aux.xmax,leyo);                  {Lee otro bloque de datos}
   Until(j>cab_aux.ymax-1);                                     {Hasta terminar todos los renglones}
   dispose(buf);                                                {Descarga memoria}
   aux:=0;blockwrite(esc_arch,aux,1,escribio);                  {Escribe código de fin de información "00h 01h"}
   aux:=1;blockwrite( esc_arch,aux,1,escribio);
  seek(esc_arch,0);                                             {Se coloca al inicio del archivo de escritura}
  bmp.tamano:=filesize(esc_arch);          {El siguiente bloque calcula los datos de la cabecera faltantes}
  bmpi.tamano_imagen:=filesize(esc_arch)-bmp.tam_cabeza;
  blockwrite(esc_arch,bmp ,sizeof(bmp) ,escribio);              {Escribe la cabecera BMP}
  blockwrite(esc_arch,bmpi,sizeof(bmpi),escribio);
  close (esc_arch);
 End;
End;
End;
```

```
Begin                                                    {Cuerpo del programa princiapl}
 clrscr;
 disco_pantalla:=true;                                   {Disco=verdadero (conversión)  Pantalla=falso}
 archiv:='mk.bmp';                                       {Asigna  a una variable el nombre del archivo}
 If Abre_lectura_bmp(archiv) Then                        {Si logra abrir el archivo continua con el programa}
  Begin
   Lee_Cabecera_BMP;                                     {Lee los datos de la cabecera}
   Readkey;
   Lee_Paleta_BMP;                                       {Lee los datos de la paleta}
   Inicializa(16,4);                                     {Inicializa el modo gráfico, 1024x768x256}
   Pon_Paleta_BMP;                                       {Activa la paleta leída del archivo}
   Lee_imagen_BMP;                                       {Descomprime la imagen}
   close(lee_arch);                                      {Cierra archivo de lectura}
   close(tem_arch);                                      {Cierra archivo temporal}
   restorecrtmode;                                       {Restaura modo gráfico}
   closegraph;                                           {Cierra modo gráfico}
   clrscr;                                               {Limpia pantalla}
   if disco_pantalla Then                                {Si disco_pantalla es verdadero}
    Begin
     Esc_imagen_BMP;                                     {Escribe la imagen}
     gotoxy(1,2);write('La conversión se completo con éxito');
    End;
  End;
End.
```

Como puede apreciar la programación de éste método de compresión no es muy complejo. Note que al proceso de escribir datos sin comprimir se le denominó RAW (Raw packet).

Una de las imágenes procesadas con el programa es: mk.bmp que es un archivo BMP sin compresión.



MK.BMP

El programa da como resultado los siguientes datos:

```
Cabecera del archivo BMP de MicroSoft Windows mk.bmp
Tipo de bmp              :    BM      Posición del puntero :18
Tamaño del arch          :    308278
Tam_cabeza               :    1078   ,0   ,0
Tipo de bmp              :    40
Xmáximo                  :    640
Ymáximo                  :    480
Planos                   :    1
Bits por pixel           :    8
Tipo de compresión       :    1
Tamaño de imagen         :    307200
bi x                     :    0
bi y                     :    0
bi clr used              :    0
bi clr import            :    0
```

Espere un momento en el cual se descomprime la imagen y se crea el temporal enseguida aparece:

```
Porcentaje de conversión : 100 %
La conversión se completo con éxito
```

V.4.2 En Lenguaje C

El programa en C se codifica de la siguiente manera:

```
/*Programa que escribe el formato de BMP.
  Realizado para el trabajo de tesis titulado: FORMATOS GRÁFICOS
        Programadores :
                Cantero Ramírez Carlos
                Trejo Bonaga Marilu
        Asesores  :
                Manzo Salazar Itsmael
                Monterrosa Escobar Amilcar.   */

#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>
#include <dos.h>
#include <conio.h>
#include <string.h>
#include <io.h>
#include <fcntl.h>

#define BYTE    unsigned char                    // Definición de constantes simbólicas
#define True  1
#define False 0
#define LONGINT unsigned long
#define WORD    unsigned int

typedef struct{                          // Estructura auxiliar para grabar en un archivo temporal
 unsigned char nombre[9];
 LONGINT xmax,ymax;
 BYTE dir;                               /*Dirección de la imagen    A(0,0)      B(x,0)
                                                0=A-D*
                                                1=D-A
                                                2=C-B*
                                                3=B-C                C(0,y)      D(x,y)
                                          * son las más comunes    */

 BYTE pal_rgb[768];
}arch_temp;

typedef struct{
 WORD  tipo;                                                  // Tipo de archivo letras " BM "
 LONGINT tamano;                                              // Tamaño del archivo en bytes
 WORD    reservado1,reservado2;                               // Reservados con valor cero
 LONGINT tam_cabeza;                                          // Desplazamiento de la cabeza del bitmap
 LONGINT tipo_bmp;                                            // Determina tipo de BMP
}cabeza_bmp;

typedef struct{
 BYTE azul,verde,rojo,reservado;                              // En este orden
}paleta_bmpi;

typedef struct{
 BYTE azul,verde,rojo;                                        // En este orden
}paleta_bmpc;
```

```
typedef struct{
LONGINT xmaximo;                    // Determina Y máximo de la imagen
LONGINT ymaximo;                    // Determina X máximo de la imagen
WORD planos;                        // Planos por pixel
WORD contador;                      // Contador
LONGINT tipo_compresion,            // Tipo de compresión
        tamano_imagen,              // Tamaño de la imagen
        bitxmetrox,                 // Bits en x por metro
        bitxmetroy,                 // Bits en y por metro
        biclrusado,                 // No. de colores usados
        biclrimport;                // No. de colores importantes
}informe_bmp;

typedef struct{
WORD xmaximo,                       // Determina Y máximo de la imagen
     ymaximo,                       // Determina X máximo de la imagen
     planos,                        // Planos por pixel
     contador;                      // Contador
}centro_bmp;

#define bi_rgb 0                    // Constantes de compresión
#define bi_rle8 1
#define bi_rle4 2

char *archivo;
FILE *lee_arch,                     // Archivo de lectura
    *esc_arch,                      // Archivo de escritura
    *tem_arch;                      // Archivo de temporal
arch_temp cab_aux;                  // Cabecera auxiliar, almacena los datos de la imagen
cabeza_bmp bmp;
informe_bmp bmpi;
centro_bmp bmpc;
paleta_bmpi bmp_pali[256];
paleta_bmpc bmp_palc[256];
int disco_pantalla;                 // Variable que indica si se convierte o no
fpos_t posicion;
LONGINT i,j,k;

int Inicializa(int Driver, int Modo);
void Cambia_color(BYTE color,BYTE r,BYTE v,BYTE a);
void Solo_nombre(char *archivo);
int  Abre_lectura_bmp(char *arch_imagen);
void Lee_Cabecera_BMP(void);
void Lee_Paleta_BMP(void);
void Pon_Paleta_BMP();
void Lee_imagen_BMP();
void Esc_imagen_BMP();

void main ()
{
 clrscr();
 disco_pantalla=1;                  // Disco=1 (conversión)  pantalla=0
 archivo="mountn1.bmp";             // Asigna a una variable el nombre del archivo
 if (Abre_lectura_bmp(archivo))     // Si logra abrir el archivo continua con el programa
```

```
 {
  Lee_Cabecera_BMP();                                   // Lee los datos de la cabecera
  getch();
  Lee_Paleta_BMP();                                     // Lee los datos de la paleta
  Inicializa(16,4);                                     // Inicializa el modo gráfico, 1024x768x256
  Pon_Paleta_BMP();                                     // Activa la paleta leída
  Lee_imagen_BMP();                                     // Descomprime la imagen
  fclose(lee_arch);                                     // Cierra el archivo de lectura
  fclose(tem_arch);                                     // Cierra el archivo temporal
  restorecrtmode();                                     // Restaura modo gráfico
  closegraph();                                         // Cierra modo gráfico
  clrscr();                                             // Limpia pantalla
  if (disco_pantalla)                                   // Si disco_pantalla es verdadero
   { Esc_imagen_BMP();                                  // Escribe la imagen
     gotoxy(1,2);printf("La conversión se completo con éxito");
   }
 }
}

int Inicializa(int Driver, int Modo)                    // Inicializa modo gráfico y reporta los errores que ocurran
{ int ErrorG;                                           // Esta función no necesita explicación alguna
 unsigned int x,y;
 union REGS regin,regout;
 if (Driver==16) Driver = installuserdriver("Svga256",NULL);
 if (Driver==grError) exit;
 initgraph( &Driver, &Modo,"d:\borlandc\programa\exeobj");
  regin.x.ax=0x000f;
  regin.x.cx=0x8;
  regin.x.dx=0x8;
  int86(0x33,&regin,&regout);
  x=getmaxx();y=getmaxy();
  regin.x.ax=x;
  regin.x.dx=regin.x.ax;
  regin.x.ax=0x0007;
  regin.x.cx=0x0;
  int86(0x33,&regin,&regout);
  regin.x.ax=y;
  regin.x.dx=regin.x.ax;
  regin.x.ax=0x0008;
  regin.x.cx=0x0;
  int86(0x33,&regin,&regout);
  ErrorG=graphresult();
 if (ErrorG!=grOk)
  {
    printf("\n Error en graficos : %s",grapherrormsg(ErrorG));
    restorecrtmode;
    return 0;
  }
 else
  {
    cleardevice;
    return 1;
   }
 }
```

```
void Cambia_color(BYTE color,BYTE r,BYTE v,BYTE a)          // Modifica la paleta de colores activa
{                                                           // Esta función no necesita explicación
 asm mov dx,3C8h
 asm mov al,color
 asm out dx,al
 asm inc dx
 asm mov al,r
 asm out dx,al
 asm mov al,v
 asm out dx,al
 asm mov al,a
 asm out dx,al
}

char nom_arch[8];
void Solo_nombre(char *archivo)                             // Almacena el nombre sin extensión del archivo
{                                                           // Esta función no necesita explicación
 int i;
 i=strlen(archivo)-4;
 strncpy(nom_arch,archivo,i);
 nom_arch[i]=0;
}

int Abre_lectura_bmp(char *arch_imagen)                     // Prepara al archivo para su lectura
{                                                           // y al archivo temporal para su escritura
 int exito;                                                 // Esta función no necesita explicación
 lee_arch=fopen(arch_imagen,"rb");
 if (lee_arch==NULL)
  {
   printf("Error de apertura de archivo \n");
   exito=False;
  }
 else
  {
   exito=True;
   if (disco_pantalla==1)
    {
     tem_arch=fopen("T_E_M_P!.SSS","wb");
     if (tem_arch==NULL)
       exito=False;
    }
  }
 return exito;
}

void Lee_Cabecera_BMP()                     // Lee los datos de la cabecera y los almacena en una estructura
{
 int i=0;
 fread((char *)&bmp,1,sizeof(cabeza_bmp),lee_arch);
 fgetpos(lee_arch,&posicion);
 clrscr;
 Solo_nombre(archivo);                      // Asigna el nombre del archivo sin extensión
 gotoxy(1,1);printf("Cabecera del archivo BMP de MicroSoft Windows %s",archivo);
 gotoxy(12);printf("Tipo de bmp : %c%c Posición del puntero :%c %c"bmp.tipo & 0xffbmp.tipo >> 8,posicion);
```

```
gotoxy(1,3);printf("Tamaño del arch     : %ld",bmp.tamano);
gotoxy(1,4);printf("tam_cabeza          : %ld %d %d ",bmp.tam _cabeza,bmp.reservado1,bmp.reservado2);
gotoxy(1,5);printf("Tipo de bmp         : %ld",bmp.tipo_bmp);
if (bmp.tipo_bmp==40)
 {
  fread((char *)&bmpi,1,sizeof(informe_bmp),lee_arch);
  gotoxy(1,6) ;printf("xmaximo            : %ld",bmpi.xmaximo);
  gotoxy(1,7) ;printf("ymaximo            : %ld",bmpi.ymaximo);
  gotoxy(1,8) ;printf("planos             : %d",bmpi.planos);
  gotoxy(1,9) ;printf("bits por pixel     : %d",bmpi.contador);
  gotoxy(1,10);printf("tipo de compresion  : %ld",bmpi.tipo_compresion);
  gotoxy(1,11);printf("tamaño de imagen    : %ld",bmpi.tamano_imagen);
  gotoxy(1,12);printf("bi x               : %ld",bmpi.bitxmetrox);
  gotoxy(1,13);printf("bi y               : %ld",bmpi.bitxmetroy);
  gotoxy(1,14);printf("bi clr used        : %ld",bmpi.biclrusado);
  gotoxy(1,15);printf("bi clr import      : %ld",bmpi.biclrimport);
                                        // El siguiente bloque almacena los datos de la cabecera bmpi
                                           // requeridos para la compresión en la estructura cab_aux
  if (disco_pantalla)
   {
    for(i=0;i<=8;i++)
        cab_aux.nombre[i]=nom_arch[i];
    cab_aux.xmax=bmpi.xmaximo;
    cab_aux.ymax=bmpi.ymaximo;
    cab_aux.dir=2;                          // Una imagen BMP se despliega de la parte inferior
   }                                                          // izquierda a la superior derecha
 }
else
 {
  fread((char *)&bmpc,1,sizeof(centro_bmp),lee_arch);
  gotoxy(1,6);printf("xmaximo            : %d",bmpc.xmaximo);
  gotoxy(1,7);printf("ymaximo            : %d",bmpc.ymaximo);
  gotoxy(1,8);printf("planos             : %d",bmpc.planos);
  gotoxy(1,9);printf("bits por pixel     : %d",bmpc.contador);
        // El siguiente bloque almacena los datos de la cabecera bmpc requeridos en la estructura cab_aux
  if (disco_pantalla)
   {
    for(i=0;i<=8;i++)
        cab_aux.nombre[i]=nom_arch[i];
    cab_aux.xmax=bmpc.xmaximo;
    cab_aux.ymax=bmpc.ymaximo;
    cab_aux.dir=2;                          // Una imagen BMP se despliega de la parte inferior
   }                                                          // izquierda a la superior derecha
 }
}

void Lee_Paleta_BMP()
{                   // Lee la paleta de colores y la almacena en un arreglo  así como en la estructura cab_aux
 int contp;
 if (bmp.tipo_bmp==40)
  {
   fread((BYTE *)&bmp_pali,256,sizeof(paleta_bmpi),lee_arch);
                                  // El siguiente bloque almacena la paleta en el arreglo de la estructura cab_aux
   if (disco_pantalla)
```

```
    for (contp=0;contp<=255;contp++)
    {
     cab_aux.pal_rgb[(contp*3)+0]=bmp_pali[contp].rojo;
     cab_aux.pal_rgb[(contp*3)+1]=bmp_pali[contp].verde;
     cab_aux.pal_rgb[(contp*3)+2]=bmp_pali[contp].azul;
    }
  }
  else
  {
   fread((BYTE *)&bmp_palc,256,sizeof(paleta_bmpc),lee_arch);
                                    // El siguiente bloque almacena la paleta en el arreglo de la estructura cab_aux
 if (disco_pantalla)
    for (contp=0;contp<=255;contp++)
    {
     cab_aux.pal_rgb[(contp*3)+0]=bmp_palc[contp].rojo;
     cab_aux.pal_rgb[(contp*3)+1]=bmp_palc[contp].verde;
     cab_aux.pal_rgb[(contp*3)+2]=bmp_palc[contp].azul;
    }
  }
}

void Pon_Paleta_BMP()                                          // Activa la paleta de colores del archivo
{
 int i;
 if (bmp.tipo_bmp==40)
  for(i=0;i<=255;i++)
    Cambia_color(i,bmp_pali[i].rojo >> 2,bmp_pali[i].verde >> 2,bmp_pali[i].azul >> 2);
 else
  for (i=0;i<=255;i++)
    Cambia_color(i,bmp_palc[i].rojo >> 2,bmp_palc[i].verde >> 2,bmp_palc[i].azul >> 2);
}

void Lee_imagen_BMP()                          // Descomprime los datos de imagen y los almacena en
{                                                                                  // el archivo temporal
 int ponx,pony,numcols;
 LONGINT cont,tam_imagen;
 BYTE unbyte,cuantos;
 char salte;
 numcols=256;
 fseek(lee_arch,bmp.tam_cabeza,SEEK_SET);
 if (bmp.tipo_bmp==40)
  {
   if (bmpi.tipo_compresion==0)
    bmpi.xmaximo=(bmp.tamano-bmp.tam_cabeza)/bmpi.ymaximo;
    cab_aux.xmax=bmpi.xmaximo;
    tam_imagen=((bmpi.ymaximo)*bmpi.xmaximo);
    switch (bmpi.tipo_compresion)
    {
          case 0:
             for (i=0;i<=tam_imagen;i++)
              {
               if (numcols==256)
                 fread((unsigned char *)&unbyte,1,1,lee_arch);
               else
```

```
      if (numcols==16)                                          // Para 16 colores
       {}
      else
       if (numcols==2)                                          // Para 2 colores
        {}
       ponx=(i%(bmpi.xmaximo));
       pony=(i/(bmpi.xmaximo));
       if (disco_pantalla)                          // Almacena el dato en el archivo temporal
         fwrite((unsigned char *)&unbyte,1,1,tem_arch);
      else
          putpixel(ponx,bmpi.ymaximo-pony,unbyte);
     }
    break;
  case 1:
    i=0;j=0;cont=0;k=0;salte=False;
    do
     {
      fread((unsigned char *)&unbyte,1,1,lee_arch);
      fread((unsigned char *)&cuantos,1,1,lee_arch);
      if(unbyte==0)
       {
           switch (cuantos)
            {
             case 0:                                            // Fin de linea
              j++;
              i=0;
              break;
             case  1:                                           // Fin de archivo
              salte=True;
              break;
             case  2:                                           // Delta
              break;
             default:                                           // Sin codificar
              for (k=1;k<=cuantos;k++)
               {
                i++;
                cont++;
                fread((unsigned char *)&unbyte,1,1,lee_arch);
                ponx=(cont % (bmpi.xmaximo));
                pony=(cont / (bmpi.xmaximo));
                if (disco_pantalla)                     // Almacena el dato en el archivo temporal
                     fwrite((unsigned char *)&unbyte,1,1,tem_arch);
                else
                     putpixel(ponx,bmpi.ymaximo-pony,unbyte);
               }
              if ((cuantos % 2)!=0) fread((unsigned char *)&unbyte,1,1,lee_arch);
              break;
            }
       }
      else
       {
           for (k=1;k<=unbyte;k++)
            {
             i++;
```

```
                    cont++;
                    ponx=(cont % (bmpi.xmaximo));
                    pony=(cont /(bmpi.xmaximo));
                    if (disco_pantalla)                    // Almacena el dato en el archivo temporal
                      fwrite((unsigned char *)&cuantos,1,1,tem_arch);
                    else
                      putpixel(ponx,bmpi.ymaximo-pony,cuantos);
                  }
                }
              } while (salte==False);
            case 2:                                                          // RLE de 4 bits
              break;
      }
    }
    else                                                                 // Es windows v2.0, Os 2 v1.0
    {
      bmpc.xmaximo=((bmp.tamano-bmp.tam_cabeza)/bmpc.ymaximo);
      cab_aux.xmax=bmpc.xmaximo;
      tam_imagen=bmpc.ymaximo;
      tam_imagen=(tam_imagen*bmpc.xmaximo);
       for (i=0;i<=tam_imagen;i++)
       {
            if (numcols==256)
             fread((char *)&unbyte,1,1,lee_arch);
            else
             if (numcols==16)                                            // Para 16 colores
              {}
             else
              if (numcols==2)                                            // Para 2 colores
               {}
            ponx=(i%(bmpc.xmaximo));
            pony=(i/(bmpc.xmaximo));
            if (disco_pantalla)                    // Almacena el dato en el archivo temporal
              fwrite((unsigned char *)&unbyte,1,1,tem_arch);
            else
              putpixel(ponx,bmpc.ymaximo-pony,unbyte);
      }
    }
    if (disco_pantalla)                    // Una vez almacenados los datos de imagen en el archivo
                                           // temporal almacena al final la cabecera auxiliar cab_aux
      fwrite((unsigned char *)&cab_aux,1,786,tem_arch);
  }

  int ponx,pony,numcols;
  LONGINT cont,tam_imagen,tamano;
  int
   aux,                                                                  // Byte auxliar
    unbyte,                                                              // Byte de lectura
     cuantos,                                                            // Contador de datos con compresión
      byteant,                                                           // Byte antepenúltimo
       bytepen,                                                          // Byte penúltimo
        sin_comp,                                                        // Contador de datos sin compresión
         bandera;                                                        // Bandera tipo de compresión
  BYTE arr_bytes[256];                                                   // Arreglo de bytes comprimidos
```

```
void Esc_imagen_BMP()                                          // Realiza la conversión al formato BMP
{
 int salte;
 char *siglas=".bmp";                                          // Extensión del archivo
 unsigned char nom_bmp[12];                                    // Nombre del archivo
 BYTE buf[4096];                                               // Arreglo para almacenar los datos

 void pos_sig_bloque();                                        // Determina la posición del siguiente bloque de datos
 void RLE();                                                   // Procedimiento para comprimir con RLE
 void RAW();                                                   // Procedimiento para comprimir con RAW
 tem_arch=fopen("T_E_M_P!.$$$","rb");                          // Abre el archivo temporal para su lectura
  if (tem_arch!=NULL)                                          // Si no hay errores continúa
   {
    tam_imagen=filelength(open("T_E_M_P!.$$$",O_RDWR));        // Calcula el tamaño de la imagen
    fseek(tem_arch,tam_imagen-786,SEEK_SET);                   // Se coloca en el inicio de la cabecera auxiliar
   fread((unsigned char *)&cab_aux,1,sizeof(cab_aux),tem_arch);  // Lee y almacena la cabecera auxiliar
                                                               // El siguiente bloque asigna X_nombre al archivo de escritura
   for(i=0;i<=11;i++)
     nom_bmp[i]=0;
     nom_bmp[0]='x';
     nom_bmp[1]='_';
   i=0;
    do
     {
       nom_bmp[i+2]=cab_aux.nombre[i];
       i++;
     }while (cab_aux.nombre[i]!=0);
     strcat(nom_bmp,siglas);
     esc_arch=fopen(nom_bmp,"wb");                             // Abre el archivo destino para su escritura
     if (esc_arch!=NULL)                                       // Si no hay errores continúa
      {                                                        // El siguiente bloque asigna los datos de la cabecera
       bmp.tipo=19778;                                         // Asigna tipo de archivo, letras "BM", identificador
       bmp.tamano=0;                                           // Se asigna hasta tener el archivo completo
       bmp.tam_cabeza=1078;                                    // Paleta de colores incluída
       bmp.reservado1=0;                                       // Siempre cero
       bmp.reservado2=0;                                       // Siempre cero
       bmp.tipo_bmp=40;                                        // Tipo 40, Windows 3.x o NT
       bmpi.xmaximo=cab_aux.xmax;                              // X máximo de la imagen
       bmpi.ymaximo=cab_aux.ymax;                              // Y máximo de la imagen
       bmpi.planos=1;                                          // Solo un plano de color RGB
       bmpi.contador=8;                                        // 8 bits por pixel, 256 colores
       bmpi.tipo_compresion=1;                                 // RLE método de compresión a utilizar
       bmpi.tamano_imagen=0;                                   // Se asigna hasta tener el archivo completo
                                                               // este valor es = tamano-tam_cabeza
       bmpi.bitxmetrox=0;                                      // Valor con cero
       bmpi.bitxmetroy=0;                                      // Valor con cero
       bmpi.biclrusado=256;                                    // Utiliza todos los colores
       bmpi.biclrimport=256;                                   // Todos los colores son importantes
                                                               // Reserva lugares para completar cabecera
       fwrite((unsigned char *)&bmp,1,sizeof(bmp),esc_arch);
       fwrite((unsigned char *)&bmpi,1,sizeof(bmpi),esc_arch);
       unbyte=0;                                               // Escribe paleta de colores
       for (i=0;i<=255;i++)
           {
```

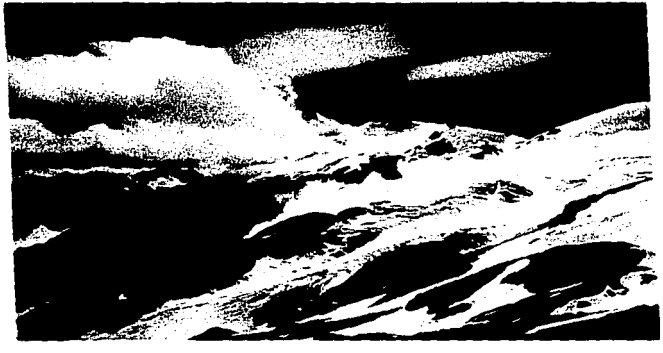
```
    fwrite((unsigned char *)&cab_aux.pal_rgb[(i*3)+2],1,1,esc_arch);          // Azul
    fwrite((unsigned char *)&cab_aux.pal_rgb[(i*3)+1],1,1,esc_arch);          // Verde
    fwrite((unsigned char *)&cab_aux.pal_rgb[(i*3)+0],1,1,esc_arch);          // Rojo
    fwrite((unsigned char *)&unbyte,1,1,esc_arch);                            // Reservado
   }                                                                          // Inicia poroceso de compresión
   aux=0;
   i=0;                                                                       // Cuenta los columnas
   j=0;                                                                       // Cuenta las renglones
   gotoxy(1,1);printf("Porcentaje de conversión :    \%");
   k=(786+bmpi.xmaximo);                // Localiza la posición de acuerdo con la dirección de la imagen
   if (cab_aux.dir==2)
    fseek(tem_arch,0,SEEK_SET);                                              // Se coloca al inicio del archivo
   else
    fseek(tem_arch,tam_imagen-k,SEEK_SET);      // Se coloca al final del archivo menos la posición
                                                // Lee un bloque de datos con longitud x.máximo
fread((unsigned char *)&buf,cab_aux.xmax,1,tem_arch);
do                                                                            // Haz
  {
   i=0;                                                                      // Inicializa variables
   cuantos=0;
   bandera=0;
   sin_comp=1;
   do                                                                         // Haz
   {
    byteant=buf[i];                                                          // Asigna byteant
    bytepen=buf[i+1];                                                        // Asigna bytepen
    unbyte=buf[i+2];                                                         // Asigna unbyte
    if ((byteant!=bytepen) || (bytepen!=unbyte))                             // Si algún byte es diferente
     {
      if (bandera==1)                                                        // Datos comprimidos
      {
       cuantos=cuantos+2;                                                    // Incrementa cuantos
       i++;                                                                  // Incrementa columnas
       RLE();                                                                // Escribe RLE
       bandera=0;                                                            // Inicializa bandera
      }
     else                                                                    // Almacena bytes sin comprimir
      {
       arr_bytes[sin_comp]=byteant;
       sin_comp++;
       if (sin_comp>=255)                                                    // Verifica sean menor a 256
        {
           sin_comp--;                                                       // Decrementa sin comprimir
           RAW();                                                            // Escribe RAW
           sin_comp=1;                                                       // Inicializa sin comprimir
        }
       bandera=2;                                                            // Asigna bandera sin comprimir
      }
     cuantos=0;                                                              // Inicializa cuantos
     }
    else                                                                     // Hay bytes iguales
     {
      if (bandera==2)                                                        // Revisa si se envian bytes sin comprimir
       {
```

```
            sin_comp--;                                          // Decrementa bytes sin comprimir
            RAW();                                               // Escribe RAW
           }
           bandera=1;                                            // Cuenta 1 byte compreso
           cuantos++;                                            // Incrementa cuantos
          if (cuantos>=255)  RLE();                              // Verifica sea menor a 256
          sin_comp=1;                                            // Inicializa sin comprimir
         }
         i++;                                                    // Incrementa cuanta de columnas
        }while (i<(cab_aux.xmax-3));                             // Mientras no se acomplete el renglón
        if (bandera==2)                                          // Revisa si un conteo quedo sin terminar
         {
          sin_comp--;                                            // Decrementa sin comprimir
          RAW();                                                 // Escribe RAW
         }
        if (bandera==1) RLE();                                   // Escribe los ultimos bytes
        sin_comp=cab_aux.xmax-i;                                 // Asigna posición del byte
        arr_bytes[1]=buf[i];                                     // Asigna bytes a comparar
        arr_bytes[2]=buf[i+1];
        arr_bytes[3]=buf[i+2];
        RAW();                                                   // Escribe RAW
        aux=0;                                                   // Escribe fin de linea " 00h 00h"
        fwrite((unsigned char *)&aux,1,1,esc_arch);
        fwrite((unsigned char *)&aux,1,1,esc_arch);
        pos_sig_bloque();                                        // Calcula posición del siguiente bloque
        fread((unsigned char *)&buf,cab_aux.xmax,1,tem_arch);    // Lee otro bloque de datos
    }while (j<=(cab_aux.ymax-1));                                // Mientras no se terminen los renglones
    aux=0;                                                       // Escribe codigo de fin de informacion "00h 01h"
    fwrite((unsigned char *)&aux,1,1,esc_arch);
    aux=1;
    fwrite((unsigned char *)&aux,1,1,esc_arch);
                                                                 // El siguiente bloque calcula los datos de la cabecera faltantes
    tamano=filelength(open(nom_bmp,O_RDWR));
    bmp.tamano=tamano;
    bmpi.tamano_imagen=tamano-bmp.tam_cabeza;
    fseek(esc_arch,0,SEEK_SET);                                  // Se coloca al inicio del archivo, escribe la cabecera
    fwrite((unsigned long *)&bmp,1,sizeof(bmp),esc_arch);
    fwrite((unsigned char *)&bmpi,1,sizeof(bmpi),esc_arch);
    fcloseall();
   }
  }
}

void pos_sig_bloque()                                            // Determina la posición del siguiente bloque de datos
{
 j++;                                                            // Incrementa cuenta de renglones
 k=k+cab_aux.xmax;                                               // Incrementa cuenta de bloques
 if (cab_aux.dir==2)
  seek(tem_arch,j*cab_aux.xmax,SEEK_SET);                        // Asigna la siguiente posición a partir del inicio
 else
  fseek(tem_arch,tam_imagen-k,SEEK_SET);                         // Asigna la siguiente posición a partir del final
 gotoxy(28,1);printf("%3d",(j*100/cab_aux.ymax));                // Calcula el porcentaje de compresión
}
```

```
void RLE()                                                    // Procedimiento para comprimir con RLE
{
  fwrite((unsigned char *)&cuantos,1,1,esc_arch);             // Escribe el n{umero de bytes comprimidos
  fwrite((unsigned char *)&byteant,1,1,esc_arch);             // Escribe el valor de dicho byte
  cuantos=0;                                                  // Inicializa cuantos
}

void RAW ()                                                   // Procedimiento para comprimir con RAW
{
 int cont;
 if (sin_comp>=3)                                             // Si datos sin comprimir es mayor o igual a 3
  {
   aux=0;
   fwrite((unsigned char *)&aux,1,1,esc_arch);                // Escribre el byte que indica datos sin comprimir
   fwrite((unsigned char *)&sin_comp,1,1,esc_arch);           // Escribe el número de bytes sin comprimir
   for (cont=1;cont<=sin_comp;cont++)
    fwrite((unsigned char*)&arr_bytes[cont],1,1,esc_arch);    // Escribe cada uno de los bytes sin comprimir
   if ((sin_comp%2==1))                                       // Si el número de bytes sin comprimir es inpar
    fwrite((unsigned char *)&aux,1,1,esc_arch);               // Escribe otro byte para completar el par
  }
 else                                                         // Datos sin comprimir es menor a 3
  {
   for (cont=1;cont<=sin_comp;cont++)                         // Para cada dato sin comprimir
    {
     aux=1;
     fwrite((unsigned char *)&aux,1,1,esc_arch);              // Escribe byte que indica un byte impreso
     fwrite((unsigned char *)&arr_bytes[cont],1,1,esc_arch);  // Escribe el valor de dicho byte
    }
  }
}
```

La programación tanto en Pascal como en C no varía mucho salvo en el manejo de cadenas en el momento de manipular el nombre del archivo y asignarlo al archivo destino.

Una de las imágenes procesadas con el programa es: mountn1.bmp que es un archivo BMP sin compresión.



MOUNTN1.BMP

El programa da como resultado los siguientes datos:

```
Cabecera del archivo BMP de MicroSoft Windows mountn1.bmp
Tipo de bmp              : BM    Posición del puntero :$
Tamaño del arch          : 308278
tam_cabeza               : 1078 0 0
Tipo de bmp              : 40
xmaximo                  : 640
ymaximo                  : 480
planos                   : 1
bits por pixel           : 8
tipo de compresion       : 0
tamaño de imagen         : 307200
bi x                     : 0
bi y                     : 0
bi clr used              : 0
bi clr import            : 0_
```

Espere un momento en el cual se descomprime la imagen y se crea el temporal enseguida aparece:

```
Porcentaje de conversión : 100 %
La conversión se completo con éxito
```

V.5 Compresión del Formato de Archivo PCX

Al igual que BMP, PCX utiliza la base del método de compresión RLE para codificar sus datos, a diferencia que no utiliza pares si no una llave, 192, además su límite es de 63, en general el proceso de codificación es el mismo como podrá observarse a continuación.

V.5.1 En Lenguaje Pascal

Una vez realizadas las modificaciones al código de la lectura de imagen en Pascal, la función que nos interesa ingresar al código es la siguiente:

```
Procedure Esc_imagen_PCX;
Var
 tam_imagen:Longint;
 byteant,                                                    {Byte anterior}
  outbyte,                                                   {Byte actual}
   bytecont,                                                 {Cuenta bytes comprimidos}
    aux,                                                     {Byte auxiliar}
     bandera:byte;                                           {Bandera tipo de compresión}
 nom_pcx:string;                                             {Nombre del archivo}

Procedure pos_sig_bloque(var j:longint);                     {Determina la posición del siguiente bloque}
Begin
 inc(j);                                                     {Incrementa cuenta de renglones}
 inc(k,cab_aux.xmax);                                        {Incrementa cuenta de bloques}
 if (cab_aux.dir=0) Then
   seek(tem_arch,j*cab_aux.xmax)                             {Asigna la siguiente posición a partir del inicio}
 else
   seek(tem_arch,filesize(tem_arch)-k);                      {Asigna la siguiente posición  a partie del final}
 gotoxy(28,1);write((j*100/cab_aux.ymax):3:0);               {Calcula el porcentaje de compresión}
End;

Begin                                                        {El siguiente bloque abre el archivo temporal para su lectura}
 Assign(tem_arch,'T_e_m_p!.SSS');
   {Si-}
   reset(tem_arch,1);
  {Si+}
 if (ioresult=0) Then                                        {Si no hay errores continúa con el programa}
  Begin
   tam_imagen:=filesize(tem_arch)-786;                       {Calcula tamaño de la imagen}
   Seek(tem_arch,filesize(tem_arch)-786);                    {Se coloca al inicio de la cabecera auxiliar}
   blockRead(tem_arch,cab_aux,sizeof(cab_aux),leyo);         {Lee y almacena la cabecera auxiliar}
   nom_pcx:='';i:=1;                                         {Inicializa variables}
   While (cab_aux.nombre[i]<>chr(0)) do
    Begin                                                    {El siguiente bloque asigna X_nombre_archivo al archivo de escritura }
     nom_pcx:=concat(nom_pcx,cab_aux.nombre[i]);
     inc(i);
    End
```

```
If Length(nom_pcx)>=5 Then
  nom_pcx:='X_'+copy(nom_pcx,1,6)+'.pcx'
else
  nom_pcx:='X_'+nom_pcx+'.pcx';
Assign(esc_arch,nom_pcx);                    {El siguiente bloque abre el archivo destino para su escritura}
  {Si-}
  rewrite(esc_arch,1);
  {Si+}
if (ioresult=0) Then                                  {Si no hay errores continúa con el programa}
 Begin                                        {El siguiente bloque asigna los datos de la cabecera PCX}
   pcx.manufactura:=10;                                        {Identificador de PCX. valor 10}
   pcx.version:=5;                                                          {Versión del archivo}
   pcx.decodifica:=1;                                                       {Tipo de compresión}
   pcx.bits_por_pixel:=8;                                                        {Bits por pixel}
   pcx.xminimo:=0;                                                         {Coordenada x mínima}
   pcx.yminimo:=0;                                                         {Coordenada y mínima}
   pcx.xmaximo:=cab_aux.xmax-1;                                            {Coordenada x máxima}
   pcx.ymaximo:=cab_aux.ymax-1;                                            {Coordenada y máxima}
   pcx.horiz_dpi:=0;                                                       {Resolución horizontal}
   pcx.vert_dpi:=0;                                                          {Resolución vertical}
   For i:=1 to 48 do
     pcx.mapa_color[i]:=cab_aux.pal_rgb[i];                                          {Paleta EGA}
   pcx.reservado:=0;                                                          {Siempre cero}
   pcx.col_planos:=1;                                                   {Un plano de color RGB}
   pcx.bit_linea_plano:=cab_aux.xmax;                              {Tamaño de línea sin comprimir}
   pcx.tipo_paleta:=1;                                                       {Tipo de paleta}
   For i:=1 to 58 do                                                {Relleno de la cabecera PCX}
     pcx.relleno[i]:=0;
   blockwrite(esc_arch,pcx,sizeof(pcx),escribio);                {Escribe la cabecera en el archivo destino}
   aux:=0;                                                              {Inicia proceso de compresión}
   gotoxy(1,1);write('Porcentaje de conversión :    %');
   k:=(786+pcx.bit_linea_plano);              {Localiza la posicion [5] de acuerdo con la dirección de la imagen}
   if (cab_aux.dir=0) Then
     Seek(tem_arch,0)                                                   {Se coloca al inicio del archivo}
   else
     Seek(tem_arch,filesize(tem_arch)-k);          {Se coloca al final del archivo menos la posición anterior}
   blockread(tem_arch,outbyte,1,leyo);                                                   {Lee un byte}
   bandera:=0;bytecont:=0;                                                {Inicializa variables}
   tam_imagen:=(pcx.ymaximo+1);
   tam_imagen:=tam_imagen*pcx.Bit_linea_plano;                          {Calcula tamaño de imagen}
   i:=1;                                                                {Inicia cuenta de columnas}
   j:=0;                                                                {Inicia cuenta de renglones}
   Repeat                                                                          {Repite el proceso}
     byteant:=outbyte;                                                          {Asigna el byte leído}
     blockread(tem_arch,outbyte,1,leyo);                                                   {Lee un byte}
     inc(i);                                                            {Incrementa cuenta de columnas}
     if (i mod cab_aux.xmax=0) Then                                          {Si se terminó el bloque}
       pos_sig_bloque(j);                                          {Calcula la posición del siguiente bloque}
     if byteant<>outbyte Then                                                {Si los bytes son diferentes}
      Begin
       if bandera=1 Then                                                        {Hay datos comprimidos}
```

[5] Anteriormente la dirección de la imagen se ha designado como 0 (cero) ya que las imágenes PCX se despliegan de la esquina superior izquierda a la esquina inferior derecha

```
      Begin
       bytecont:=bytecont+192+1;                          {Asigna valor de bytes comprimidos}
       blockwrite(esc_arch,bytecont,1,escribio);          {Escribe número de bytes comprimidos}
       blockwrite(esc_arch,byteant ,1,escribio);          {Escribe el valor de dicho byte}
       bytecont:=0;                                       {Inicializa cuenta bytes comprimidos}
      End
     Else                                                 {Datos sin comprimir}
      if byteant>=192 Then                                {Si el byte es mayor o igual a 192}
       Begin
        aux:=193;
        blockwrite(esc_arch,aux     ,1,escribio);         {Escribe número de byte comprimido}
        blockwrite(esc_arch,byteant ,1,escribio);         {Escribe dicho byte}
       End
      else                                                {Byte menor a 192}
       Begin
        blockwrite(esc_arch,byteant,1,escribio);          {Escribe dicho byte}
       End;
      bandera:=0;                                         {Inicializa bandera}
     End
    Else                                                  {Bytes iguales}
     Begin
      outbyte:=byteant;                                   {Asigna el byte leído}
      inc(bytecont);                                      {Incrementa cuenta bytes comprimidos}
      bandera:=1;                                         {Asigna bandera, bytes comprimidos}
      if bytecont>=62 Then                                {Verifica no rebase 255, 192+63}
       Begin
        bytecont:=bytecont+192+1;                         {Asigna valor de bytes comprimidos}
        blockwrite(esc_arch,bytecont,1,escribio);         {Escribe número de bytes comprimidos}
        blockwrite(esc_arch,byteant ,1,escribio);         {Escribe el valor de dicho byte}
        bytecont:=0;                                      {Inicia cuenta de bytes comprimidos}
        blockread(tem_arch,outbyte,1,leyo);               {Lee byte}
        inc(i);                                           {Incrementa cuenta de columnas}
        if (i mod cab_aux.xmax=0) Then                    {Si se terminó el bloque}
           pos_sig_bloque(j);                             {Calcula la posición del siguiente bloque}
        bandera:=0;                                       {Inicializa bandera}
       End;
     End;
   until (i>=tam_imagen);                                 {Hasta completar la imagen}
   aux:=12;                                               {Escribe separador de datos}
   blockwrite(esc_arch,aux     ,1,escribio);
   blockwrite(esc_arch,cab_aux.pal_rgb,sizeof(cab_aux.pal_rgb),escribio);   {Escribe paleta RGB}
   close (esc_arch);
  End;
 close (tem_arch);
 End;
End;
```

Una de las imágenes procesadas con el programa es: pcxbruce.pcx.

PCXBRUCE.PCX

El programa da como resultado los siguientes datos:

```
Nombre de archivo PC-Zsoft:  pcxbruce.pcx          Tiene 256 Colores _
Manufactura       :        10      Posición del puntero :128
Versión           :        5
Decodifica        :        1
Bits_por_pixel    :        8
Xmínimo           :        0
Ymínimo           :        0
Xmáximo           :        287
Ymáximo           :        429
Horiz_dpi         :        150
Vert_dpi          :        150
Mapa de colores:
   0   0   0   1   1   1   2   2   2   3   3   3   4   4   4
   5   5   5   6   6   6   7   7   7   8   8   8   9   9   9
   10  10  10  11  11  11  12  12  12  13  13  13  14  14  14
Reservado         :        0
Col_Planos        :        1
Bit_linea_plano:           288
Tipo_paleta       :        1
Relleno           :
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0
```

Espere un momento en el cual se descomprime la imagen y se crea el temporal enseguida aparece:

```
Porcentaje de conversión : 100 %
La conversión se completo con éxito
```

V.5.2 En Lenguaje C

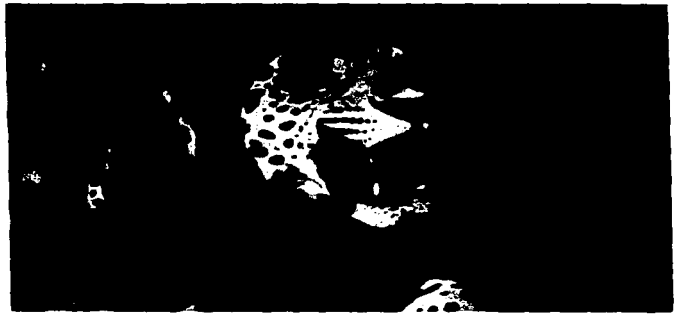
La función en C se codifica de la siguiente manera:

```
LONGINT dato,tam_imagen;
BYTE byteant,                                                    // Byte anterior
        outbyte,                                                 // Byte actual
          bytecont,                                              // Cuenta bytes comprimidos
            aux,                                                 // Byte auxiliar
             bandera;                                            // Bandera tipo de compresión

void Esc_imagen_PCX()                                            // Realiza la conversión a PCX
{
unsigned char nom_pcx[12];                                       // Nombre del archivo
char *siglas=".PCX";


void pos_sig_bloque();                                           // Determina la posición del siguiente bloque


 tem_arch= fopen("T_E_M_P!.SSS","rb");                           // Abre el archivo temporal para su lectura
 if (tem_arch!=NULL)                                             // Si no hay errores continúa con el programa
  {
   tam_imagen=filelength(open("T_E_M_P!.SSS",O_RDWR));           // Calcula tamaño de imagen
   fseek(tem_arch,tam_imagen-786, SEEK_SET);                     // Se coloca al inicio de la cabecera auxiliar
   fread((unsigned char *)&cab_aux,1,sizeof(cab_aux),tem_arch);  // Lee y almacena la cabecera aux.
                                                                 // El siguiente bloque asigna X_nombre_archivo al archivo de escritura
   for (i=0;i<=11;i++)
     nom_pcx[i]=0;
   nom_pcx[0]='x';
   nom_pcx[1]='_';
   i=0;
   do
    {
     nom_pcx[i+2]=cab_aux.nombre[i];
     i++;
    }while (cab_aux.nombre[i]!=0);
   strcat(nom_pcx,siglas);
   esc_arch=fopen(nom_pcx,"wb");                                 // Abre el archivo destino para su escritura
   if (esc_arch!=NULL)                                           // Si no hay errores continúa con el programa
   {                                                             // El siguiente bloque asigna los datos de la cabecera PCX
     pcx.manufactura=10;                                         // Identificador de PCX, valor 10
     pcx.version=5;                                              // Versión del archivo
     pcx.decodifica=1;                                           // Tipo de compresión
     pcx.bits_por_pixel=8;                                       // Bits por pixel
     pcx.xminimo=0;                                              // Coordenada x mínima
     pcx.yminimo=0;                                              // Coordenada y mínima
     pcx.xmaximo=cab_aux.xmax-1;                                 // Coordenada x máxima
     pcx.ymaximo=cab_aux.ymax-1;                                 // Coordenada y máxima
     pcx.horiz_dpi=0;                                            // Resolución horizontal
     pcx.vert_dpi=0;                                             // Resolución vertical
     for (i=1;i<=48;i++)
          pcx.mapa_color[i]=cab_aux.pal_rgb[i];                  // Paleta EGA
```

```
pcx.reservado=0;                                                    // Siempres cero
pcx.col_planos=1;                                                   // Un plano de color RGB
pcx.bit_linea_plano=cab_aux.xmax;                                   // Tamaño de lineas sin comprimir
pcx.tipo_paleta=1;                                                  // Tipo de paleta
for (i=1;i<=58;i++)
      pcx.relleno[i]=0;                                             // Relleno de la cabecera PCX
fwrite((unsigned char *)&pcx,1,sizeof(pcx),esc_arch);               // Escribe la cabecera en el archivo destino
aux=0;                                                              // Inicia proceso de compresión
k=(786+pcx.bit_linea_plano);          // Localiza la posicion de acuerdo con la dirección de la imagen
gotoxy(1,1);printf("Porcentaje de conversión :    \%");
if (cab_aux.dir==0)
      fseek(tem_arch,0,SEEK_SET);                                   // Se coloca al inicio del archivo
else
      fseek(tem_arch,tam_imagen-k,SEEK_SET);                        // Se coloca al final del archivo menos la
                                                                    // posición anterior
fread((unsigned char *)&outbyte,1,1,tem_arch);                      // Lee un byte
bandera=0;bytecont=0;                                               // Inicializa variables
tam_imagen=pcx.ymaximo+1;
tam_imagen=tam_imagen*pcx.bit_linea_plano;                          // Calcula tamaño de imagen
i=1;                                                                // Inicia cuenta de columnas
j=0;                                                                // Inicia cuenta de renglones
do                                                                  // Haz
{
     byteant=outbyte;                                               // Asigna el byte leído
     fread((unsigned char *)&outbyte,1,1,tem_arch);                 // Lee un byte
     i++;                                                           // Incrementa cuenta de columnas
     if (i % cab_aux.xmax==0)                                       // Si se teminó el bloque
          pos_sig_bloque();                                         // Calcula la posición del siguiente bloque
     if (byteant!=outbyte)                                          // Si los bytes son diferentes
      {
       if (bandera==1)                                              // Hay datos comprimidos
        {
         bytecont=bytecont+192+1;                                   // Asigna valor de bytes comprimidos
         fwrite((unsigned char *)&bytecont,1,1,esc_arch);           // Escribe número de bytes comprimidos
         fwrite((unsigned char *)&byteant,1,1,esc_arch);            // Escribe el valor de dicho byte
         bytecont=0;                                                // Inicia cuenta de bytes comprimidos
        }
       else                                                         // Datos sin comprimir
        if (byteant>=192)                                           // Si el byte es mayor o igual a 192
         {
          aux=193;
          fwrite((unsigned char *)&aux,1,1,esc_arch);               // Escribe número de byte comprimido
          fwrite((unsigned char *)&byteant,1,1,esc_arch);           // Escribe dicho byte
         }
        else                                                        // Byte menor a 192
          fwrite((unsigned char *)&byteant,1,1,esc_arch);           // Escribe dicho byte
        bandera=0;                                                  // Inicializa bandera
      }
     else                                                           // Bytes iguales
      {
       outbyte=byteant;                                             // Asigna byte leído
       bytecont++;                                                  // Incrementa cuenta bytes comprimidos
       bandera=1;                                                   // Asigna bandera, bytes comprimidos
       if (bytecont>=62)                                            // Verifica no rebase 255, 192+63
```

```
          {
           bytecont=bytecont+192+1;                          // Asigna el valor de bytes comprimidos
           fwrite((unsigned char *)&bytecont,1,1,esc_arch);  // Escribe número de bytes comprimidos
           fwrite((unsigned char *)&byteant,1,1,esc_arch);   Escribe el valor de dicho byte
           bytecont=0;                                       // Inicia cuenta de bytes comprimidos
           fread((unsigned char *)&outbyte,1,1,tem_arch);    // Lee byte
           i++;                                              // Incrementa cuenta de columnas
           if (i % cab_aux.xmax==0)                          // Si se terminó el bloque
             pos_sig_bloque();                               // Calcula la posición del siguiente cloque
           bandera=0;                                        // Inicializa bandera
          }
         }
    } while (i<tam_imagen);                                  // Hasta completar la imagen
    aux=12;                                                  // Escribe separador de datos
    fwrite((unsigned char *)&aux,1,1,esc_arch);
   fwrite((unsigned char )&cab_aux.pal_rgb,1,sizeof(cab_aux.pal_rgb),esc_arch);   // Escribe paleta RGB
    fclose (esc_arch);
   }
  fclose (tem_arch);
 }
}

void pos_sig_bloque()                                        // Determina la posición del siguiente bloque
{
 j++;                                                        // Incrementa cuenta de renglones
 k=k+cab_aux.xmax;                                           // Incrementa cuenta de bloques
 if (cab_aux.dir==0)
  fseek(tem_arch,j*cab_aux.xmax,SEEK_SET);                   // Asigna la siguiente posición a partir del inicio
 else
   fseek(tem_arch.tam_imagen-k,SEEK_SET);                    // Asigna la siguiente posición a partir del final
 gotoxy(28,1);printf("%3d",(j*100/cab_aux.ymax));            // Calcula el porcentaje de compresión
}
```

Una de las imágenes procesadas con el programa es pcxleop.pcx.



PCXLEOP.PCX

El programa da como resultado los siguientes datos:

```
Cabecera del archivo PCX: pcxleop.pcx                Tiene 256 Colores
Manufactura      :10      Posición del puntero :128
Version          :5
Decodifica       :1
Bits_por_pixel   :8
Xminimo          :0
Yminimo          :0
Xmaximo          :767
Ymaximo          :511
Horiz_dpi        :150
Vert_dpi         :150
Mapa de colores:
   0   0   0   0   0   85  0   0   170 0   0   255 0   36  0
   0   36  85  0   36  170 0   36  255 0   72  0   0   72  85
Reservado        :0  72  255 0   108 0   0   108 85  0   108 170
Col_Planos       :1
Bit_linea_plano:768
Tipo_paleta      :1
relleno          :
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0(128)
```

Espere un momento en el cual se descomprime la imagen y se crea el temporal enseguida aparece:

```
Porcentaje de conversión : 100 %
La conversión se completo con éxito
```

V.6 Compresión del Formato de Archivo GIF

El método de compresión LZW que utiliza GIF puede parecer complicado, pues va formando una tabla de códigos de cadenas con los bytes de entrada. No olvide que maneja códigos de 9, 10, 11 y 12 bits que posteriormente se convierten en bytes (8 bits), por lo que se emplean arreglos de cadenas donde se van desglosando tales bytes; el método empleado, tomar 8 códigos (sean de 9,10,11 o 12 bits) para sacar un determinado número de bytes puede parecer rústico, sin embargo, si el concepto es entendido, el lector puede adecuar otro método más rápido y eficaz.

V.6.1 En Lenguaje Pascal

El procedimiento en Pascal se codifica de la siguiene manera:

```
Procedure Esc_imagen_GIF;

Type
  Arca4096=Array [0..4096,1..2] of integer;

Var
 cadenas:^Arca4096;                                   {Arreglo de cadenas almacenadas}
 sal_cod:^Ar4096;                                     {Arreglo de códigos almacenados}
 omega:Array[1..2] of integer;                        {Prefijo}
 cont_datos,                                          {Contador de datos de archivo temporal}
  jj, kk:longInt;                                     {Contador de renglones, contador de bloques }
  i,j,k,cont_num_inicial,                             {Variables auxiliares, contador de código inicial}
  cod_limpia,                                         {Código de limpieza, 256}
   cod_final,                                         {Código de fin de lectura, 257}
    cont_cod,                                         {Contador de cada código en la tabla}
     maxcod,                                          {Máximo bloque codificado}
      posis_omega,                                    {Posición prefijo}
       cont_num,                                      {Contador de bloque}
        datos,                                        {Datos de salida}
         cuenta,                                      {Cuenta de códigos a desglosar}
          vueltas:Integer;                            {Vueltas por cada 4096 finalizado}
  lee_byte,                                           {Byte de lectura }
   aux,                                               {Auxiliar}
    resto,                                            {Resto del código}
     desplaza:byte;                                   {Desplazamiento de bytes}
  escribe_bytes:Array [0..253] of byte;               {Códigos de salida}
  nom_gif:string;                                     {Nombre del archivo}
  fin_de_datos:boolean;

  Procedure var_inicial;                              {Inicializa  variables}
  Var
   i:integer;
  Begin
    cod_limpia :=256;                                 {Código de limpieza siempre 256}
    cod_final  :=257;                                 {Código de fin de lectura 257}
```

```
 cont_cod   :=258;                                   {Contador de cada codigo en la tabla}
 maxcod     :=potencia(2,9);                         {Máximo bloque codificado}
 cont_num   :=1;                                     {Contador de bloque}
 posis_omega:=0;                                     {Posición prefijo}
                                                     {El siguiente bloque inicializa arreglos para eliminar basura}
 for i:=0 to 255 do
   Begin
    cadenas^[i,1]:=-1;
    cadenas^[i,2]:=i;
   End;
 cadenas^[256,1]:=-1;
 cadenas^[256,2]:=cod_limpia;
 cadenas^[257,1]:=-1;
 cadenas^[257,2]:=Cod_final;
 cadenas^[258,1]:=-1;
 cadenas^[258,2]:=-1;
 omega[1]:=-1;
 omega[2]:=-1;
End;

Function busca_pos_cod(omega,byte_omega:integer):Integer;   {Busca posición de omega}
Var
 k:Integer;
Begin
  k:=-1;                                             {Inicializa contador}
  if byte_omega<255 Then                             {Omega 2 menor a 255}
   Repeat                                            {Repite}
     inc(k);                                         {Incrementa k}
                                                     {Hasta encontrar las dos omegas o rebasar el contador de cadenas en la tabla}
   Until (k>cont_cod) or ((omega=cadenas^[k,1]) and (byte_omega=cadenas^[k,2]))
  else                                               {Es mayor a 255}
   k:=byte_omega;                                    {Asigna segunda omega}
 busca_pos_cod:=k;                                   {Devuelve k}
End;

Function esta_en_tabla(cod1,cod2,cod3:Integer):BOOLEAN;
Var
  j:integer;
Begin
 j:=-1;                                              {Inicializa contador}
 if cod1>=0 Then                                     {Omega 1 es mayor o igual a cero}
  Begin
   Repeat
     inc(j);                                         {Incrementa j}
                                                     {Hasta encontrar las dos omegas o rebasar el contador de cadenas en la tabla}
   Until (j>cont_cod) or ((cod1=cadenas^[j,1]) and (cod2=cadenas^[j,2]));
   cod2:=j;                                          {Asigna  omega 2}
   omega[1]:=-1;                                     {Reasigna omega 1}
   omega[2]:=j;                                      {Reasigna omega 2}
  End;
  Repeat
   inc(j);                                           {Incrementa j}
                                                     {Hasta encontrar la segunda  omega y el byte leido o rebasar el contador de cadenas en la tabla}
  Until (j>cont_cod) or ((cod2=cadenas^[j,1]) and (cod3=cadenas^[j,2]));
```

```
 If (j>cont_cod) Then                                   {Si j rebasa el contador de cadenas en la tabla}
   Esta_en_tabla:=False                                 {No está en la tabla }
 Else
   Esta_en_tabla:=True;                                 {Esta en la tabla }
 End;

Procedure escribe_datos(cont_num:integer;fin_si_no:boolean);   {Desglosa los códigos en bytes}
Var
 no_bits:Byte;                                          {Número de bits por código}
 ini,fin:longint;                                       {Inicio, fin de cadena}
 i,j,k:integer;
 cadena,aux:String;                                     {Códigos leídos desglosados en cadenas, cadena auxiliar}

 Procedure escribe_en_disco;                            {Escribe los bytes resultantes en el archivo destino}
  Begin
   blockwrite(Esc_arch,datos,1,escribio);               {Escribe cuantos bytes (254, excepto al final)}
   blockwrite(Esc_arch,escribe_bytes,datos,escribio);   {Escribe los bytes almacenados}
   datos:=0;                                            {Inicializa datos}
  End;

 Procedure saca_datos(cadena:string;no_bits:byte);      {Desglosa los bytes de la cadena}
 Var
  cont:Byte;                                            {Contador auxiliar}
  num_str:string;                                       {Cadena de 8 bits}
  numero,b:integer;                                     {Byte desglosado de num_str, auxiliar}
 Begin
  b:=0;
  For cont:=1 to no_bits do                             {Desde 1 hasta número de bits por código (9,10,11,12)}
   Begin                                                {La siguiente línea saca de la cadena 8 bits y los convierte a decimal}
    num_str:=sis_num(2,10,3,(copy(cadena,(length(cadena)+1)-(8*cont),8)));
    val(num_str,numero,b);                              {Convierte la cadena a su representación numérica}
    escribe_bytes[datos]:=numero;                       {Almacena el byte resultante}
    inc(datos);                                         {Incrementa cuenta de datos almacenados}
    If datos>=254 Then                                  {Si datos = 254}
     escribe_en_disco;                                  {Manda escribir al archivo destino}
   End;
  End;

  Begin
   no_bits:=9;                                          {Inicializa bits de entrada}
   maxcod:=512;                                         {Inicializa máximo bloque codificado}
    if cont_num<3839 Then                               {No es fin de bloque}
     Begin
      inc(cont_num);                                    {Incrementa contador de bloque}
      sal_cod^[cont_num]:=cod_final;                    {Asigna código final}
      inc(cont_num,2);                                  {Incrementa contador de bloque}
     End
    else                                                {Es fin de bloque}
      sal_cod^[cont_num]:=cod_limpia;                   {Asigna código de limpieza}
    ini:=cont_num_inicial;                              {Inicializa ini}
    fin:=ini+7;                                         {Inicializa fin}
    Repeat                                              {Repite el proceso}
    cadena:='';                                         {Inicializa cadena}
    cuenta:=0;                                          {Inicializa cuenta}
```

```
For i:=ini to fin do                                                    {Toma 8 códigos}
 Begin
  sal_cod^[i]:=sal_cod^[i] shl desplaza or resto;                       {Asigna  nuevo valor de código}
  resto:=0;                                                             {Inicializa resto}
cadena:=sis_num(10,2,no_bits+desplaza,int_str(sal_cod^[i]))+cadena;     {Convierte el código a binario}
  inc(cuenta);                                                          {Incrementa cuenta}
  if (cuenta*no_bits)<length(cadena) Then                               {Rebasa la longitud recomendable}
   Begin
    if (i=3839)  and (length(cadena)>=88) Then                          {Es fin de bloque y la longitud rebasa a 88}
     Begin
      aux:=copy(cadena,1,length(cadena)-88);                            {Asigna los bits restantes}
      cadena:=copy(cadena,length(cadena)-87,88);                        {Trunca la cadena}
     End
    else                                                                {No es fin de bloque}
     Begin
      aux:=copy(cadena,1,length(cadena)-(cuenta*no_bits));              {Asigna los bits restantes}
      if  i<>3839 Then
       cadena:=copy(cadena,length(cadena)-(cuenta*no_bits)+1,(cuenta*no_bits));   {Trunca la cadena}
     End;
    val(sis_num(2,10,1,aux),resto,k);                                   {Asigna los bits restantes[6] a resto}
   End;
 End;
 saca_datos(cadena,length(cadena) div 8);                               {Desglosa los bytes de la cadena}
 inc(ini,8);                                                            {Incrementa ini}
 fin:=ifi(fin+8>cont_num,cont_num,fin+8);                               {Incrementa fin}
 if fin+(257-cont_num_inicial)>maxcod Then                              {Es límite de bloque (512,1024,..)}
  Begin
   maxcod:=maxcod*2;                                                    {Incrementa máximo bloque}
   inc(no_bits);                                                        {Incrementa número de bits por código}
   If vueltas>0 Then inc(desplaza);                                     {Incrementa desplaza[7]}
   if desplaza>=8 Then                                                  {Desplaza es mayor o gual a 8}
    Begin
     desplaza:=0;                                                       {Inicializa desplaza}
     escribe_bytes[datos]:=resto;                                       {Asigna resto}
     inc(datos);                                                        {Incrementa datos}
     resto:=0;                                                          {Inicializa resto}
    End;
   End;
 Until (ini>=(cont_num));                                               {Hasta terminar el bloque}
 if fin_si_no Then                                                      {Se terminaron datos del temporal}
  Begin
   dec(datos);                                                          {Decrementa datos}
   blockwrite(Esc_arch,datos,1,escribio);                               {Escribe cuantos bytes, no necesariamente 254}
   blockwrite(Esc_arch,escribe_bytes,datos+1,escribio);                 {Escribe los bytes almacenados}
  End;
  vueltas:=ifi(vueltas+1=10,2,vueltas+1);                               {Asigna vueltas[8] }
  desplaza:=ifi(((9-vueltas)>=8) or ((9-vueltas)=0),0,9-vueltas);       {Asigna desplaza}
 if(cuenta<8) and (resto=0) Then                                        {Verifica bits restantes}
```

[6] Cuando desplaza se incrementa algunos de los códigos rebasan el número de bits por código correspondiente, a resto se le asignas tales bits y éste a su vez los suma al siguiente código.
[7] Después de la priemra vuelta se observó que era necesario realizar un cierto desplazamiento sobre cada código, tal desplazamiento depende del número de vueltas realizadas. El número máximo de desplazamientos es 8 (0-7).
[8] Vueltas también tiene un límite de 8 para efectos del programa no es necesario que empiece desde 0.

```
  Begin
   aux:=copy(cadena,1,length(cadena) mod 8);                    {Asigna bits restantes}
   val(sis_num(2,10,1,aux),resto,k);                            {Asigna los bits restantes a resto}
  End;
 End;

Procedure pos_sig_bloque(var jj:longint);                       {Determina la posición del siguiente bloque de datos}
 Begin
  inc(jj);                                                      {Incrementa cuenta de renglones}
  inc(kk,cab_aux.xmax);                                         {Incrementa cuenta de bloque}
  if (cab_aux.dir=0) Then
    seek(tem_arch,jj*cab_aux.xmax)                              {Asigna la posición a partir del inicio}
  else
    seek(tem_arch,filesize(tem_arch)-kk);                       {Asigna la posición a partir del final}
  gotoxy(28,1);write((jj*100/cab_aux.ymax):3:0)                 {Calcula porcentaje de conversión}
 End;

Begin
 new(cadenas);
 new(sal_cod);
 Assign(tem_arch,'T_e_m_p!.SSS');                               {El siguiente bloque abre el archivo temporal para su lectura}
   {Si-}
  reset(tem_arch,1);
   {Si+}
 if (ioresult=0) Then                                           {Si no hay error continúa}
  Begin
   tam_imagen:=filesize(tem_arch)-786;                          {Calcula tamaño de la imagen}
   Seek(Tem_arch,filesize(tem_arch)-786);                       {Se coloca en el inicio de la cabecera auxiliat}
   blockRead(tem_arch,cab_aux,sizeof(cab_aux),sololee);         {Lee y almacena la cabecera auxiliar}
   nom_gif:='';i:=1;                                            {Inicializa variables}
   While (cab_aux.nombre[i]<>chr(0)) do                         {El siguiente bloque asigna X_nombre al archivo de escritura}
    Begin
    nom_gif:=concat(nom_gif,cab_aux.nombre[i]);
     inc(i);
    End;
   If Length(nom_gif)>=5 Then
     nom_gif:='X_'+copy(nom_gif,1,6)+'.GIF'
   else
     nom_gif:='X_'+nom_gif+'.GIF';
   Assign(Esc_arch,nom_gif);                                    {El siguiente bloque abre el archivo destino para su escritura}
     {Si-}
    rewrite(Esc_arch,1);
     {Si+}
   if (ioresult=0) Then                                         {Si no hay errores continúa}
    Begin                                                       {El siguiente bloque asigna los datos de la cabecera}
      gifdp.GIF[0]:='G';                                        {Firma}
      gifdp.GIF[1]:='I';
      gifdp.GIF[2]:='F';
      gifdp.ver[0]:='8';                                        {Versión}
      gifdp.ver[1]:='9';
      gifdp.ver[2]:='a';
      gifdp.xtotal:=cab_aux.xmax;                               {Ancho de pantalla}
      gifdp.ytotal:=cab_aux.ymax;                               {Largo de pantalla}
      gifdp.describe:=215;                                      {Paquete}
```

```
gifdp.fondo:=0;                                                    {Color de fondo}
gifdp.aspect_ratio:=0;                                             {Aspect ratio}
For i:=0 to 255 do                                                 {Paleta de colores}
 Begin
  gif_pal[i].rojo :=cab_aux.pal_rgb[(i*3)+1];
  gif_pal[i].verde:=cab_aux.pal_rgb[(i*3)+2];
  Gif_pal[i].azul :=cab_aux.pal_rgb[(i*3)+3];
 End;
gifdi.separador:=',';                                              {Separador}
gifdi.xminimo:=0;                                                  {X mínima}
gifdi.yminimo:=0;                                                  {Y mínima}
gifdi.xmaximo:=cab_aux.xmax;                                       {X máxima}
gifdi.ymaximo:=cab_aux.ymax;                                       {Y máxima}
gifdi.inter:=0;                                                    {Paquete, interlazado}
blockwrite(Esc_arch,gifdp  ,sizeof(gifdp)  ,escribio);             {Escribe descriptor de pantalla}
blockwrite(Esc_arch,gif_pal,sizeof(gif_pal),escribio);             {Escribe paleta de colores}
blockwrite(Esc_arch,gifdi  ,sizeof(gifdi)  ,escribio);             {Escribe descriptor de imagen}
aux:=8;
blockwrite(Esc_arch,aux,1,escribio);                               {Byte de inicialización}
var_inicial;                                                       {Inicializa variables}
Seek(tem_arch,0);
cont_datos :=0;                                                    {Contador de datos archivo temporal}
cont_num   :=1;                                                    {Contador de bloque}
cont_num_inicial:=0;                                               {Contador de bloque inicial}
sal_cod^[cont_num_inicial]:=cod_limpia;                            {Inicializa primer código almacenado}
sal_cod^[1]:=0;                                                    {Limpia sig. posición de código}
datos:=0;                                                          {Inicializa variables}
cuenta:=0;
desplaza:=0;
vueltas:=0;
resto:=0;
gotoxy(1,1);write('Porcentaje de conversión :    %');
                                   {Localiza la posición [9] de acuerdo con la dirección de la imagen}
kk:=(786+gifdp.xtotal);
if (cab_aux.dir=0) Then
   Seek(tem_arch,0)                                                {Se coloca al inicio del archivo}
else
   Seek(tem_arch,filesize(tem_arch)-kk);                           {Se coloca al final del archivo}
 jj:=0;                                                            {Inicializa cuenta de renglones}
Repeat                                                             {Repite el siguiente bloque}
 Repeat                                                            {Repite el siguiente bloque}
  blockread(tem_arch,lee_byte,1,sololee);                          {Lee dato}
  inc(cont_datos);                                                 {Incrementa datos}
  if (cont_datos mod cab_aux.xmax=0) Then                          {Verifica si terminó la línea de datos}
      pos_sig_bloque(jj);                                          {Calcula la posición del sig. bloque}
  If esta_en_tabla(omega[1],omega[2],lee_byte) Then                {Si esta en la tabla de cadenas}
   Begin
    omega[1]:=omega[2];                                            {Reasigna omega 1}
    omega[2]:=Lee_byte;                                            {Reasigna omega 2}
   End
  Else                                                             {Si no esta en la tabla}
```

[9] Anteriormente la dirección de la imagen se ha designado como 0 (cero) ya que las imágenes GIF se despliegan de la esquina superior izquierda a la esquina inferior derecha.

```
      Begin
        posis_omega:=busca_pos_cod(omega[1],omega[2])        {Busca la posición de las dos omegas}
       sal_cod^[cont_num]:=posis_omega;                                          {Almacena código}
       sal_cod^[cont_num+1]:=0;                                          {Limpia siguiente posición}
       cadenas^[cont_cod,1]:=omega[2];                                {Asigna nueva entrada en la tabla}
       cadenas^[cont_cod,2]:=Lee_byte;
       omega[2]:=Lee_byte;                                                          {Reasigna omega 2}
       inc(cont_cod);                                                {Incrementa cuenta códigos en tabla}
       inc(cont_num);                                                          {Incrementa cuenta de bloque}

       if cont_num=3839 Then                                        {Si rebasa fin de bloque 3839+257=4096}
          posicion:=filepos(tem_arch)-1;                                          {Se regresa una posición}
       cadenas^[cont_cod,1]:=-1;                                                {Limpia siguiente posición}
       cadenas^[cont_cod,2]:=-1;
       End;
                                                          {Hasta 4096 códigos o terminar los datos del temporal}
     Until(cont_num=(3839)) or (cont_datos>(Filesize(tem_arch)-786));
     for j:=1 to 8 do                                        {Limpia posiciones de los 8 códigos a desglosar}
        sal_cod^[cont_num+j]:=0;
     seek (tem_arch,posicion);
     dec(cont_datos);                                                    {Decrementa cuenta datos temporal}
  posis_omega:=busca_pos_cod(omega[1],omega[2]);                {Busca la posición de las dos omegas}
     fin_de_datos:=False;                                                          {Asigna fin de datos}
     if (cont_datos>=(Filesize(tem_arch)-786)) Then                {Si se terminaron los datos del temporal}
        fin_de_datos:=true;                                                   {Fin de datos es verdadero}
     sal_cod^[cont_num]:=posis_omega;                                   {Almacena la posición de omega}
     escribe_datos(cont_num,fin_de_datos);          {Desglosa los códigos en bytes y los escribe en disco}
      var_inicial;                                                              {Inicializa variables}
     cont_num_inicial:=1;                                                {Inicializa cont_num_inicial}
     Until(cont_datos>=(Filesize(tem_arch)-786));                   {Hasta terminar los datos del temporal}
     aux:=59;                                                                {Escribe trailer (fin de archivo)}
     blockwrite(Esc_arch,aux,1,escribio);
   End;
 End;
dispose(cadenas);
dispose(sal_cod);
End;
```

Una de las imágenes procesadas con el programa es cat.gif.

CAT.GIF

El programa da como resultado los siguientes datos:

```
Nombre de Archivo :cat.gif
Identificación : GIF      Posición del puntero :13
Versión        : 89a
Total  en x    : 113
Total  en y    : 146
Bits por pixel : 6
Fondo          : 0
Aspect ratio   (0=si, otro=no) : 0
Separador      : e        Posición del puntero :791
xmínimo        : 0
ymínimo        : 11331
xmáximo        : -9153
ymáximo        : -27949
Interlazado    (1=si, 0=no): 0
Mapa de Color Local (1=si, 0=no): 0
```

Espere un momento en el cual se descomprime la imagen y se crea el temporal enseguida aparece:

```
Porcentaje de conversión : 100 %
La conversión se completo con éxito
```

V.6.2 En Lenguaje C

En C la función se codifica de la siguiente manera:

```
int cadenas[4097][2];                                   // Arreglo de cadenas almacenadas
LONGINT cod_sal[4097];                                  // Arreglo de códigos almacenados
int omega[2];                                           // Prefijo
LONGINT cont_datos,                                     // Contador de datos del archivo temporal
        jj,                                             // Cuenta de renglones
        kk;                                             // Cuenta de bloques
int cont_num_inicial,                                   // Contador de bloque inicial
    posis_omega,                                        // Posición prefijo
        datos,                                          // Datos de salida
        cuenta,                                         // Cuenta de códigos a desglosar
        vueltas;                                        // Vueltas por cada 4096 finalizado
BYTE lee_byte,aux,resto=0,desplaza;                     // Byte de lectura, auxiliar
BYTE escribe_bytes [255];                               // Códigos de salida

void Esc_imagen_GIF()
{
 void inicia_var();
 int busca_pos_cod(int omega,int byte_omega);
 int esta_en_tabla(int cod1,int cod2,int cod3);
 void escribe_datos(int cont_num,int fin_si_no);
 void pos_sig_bloque();

 char *nom_gif,*siglas=".GIF";                          // Nombre del archivo, extensón
 int fin_de_datos;

 tem_arch=fopen("T_E_M_P!.SSS","rb");                   // Abre el archivo temporal para su lectura
 if (tem_arch!=NULL)                                    // Si no hay errores continúa
  {
   tam_imagen=filelength(open("T_E_M_P!.SSS",O_RDWR));  // Calcula tamaño de la imagen
   fseek(tem_arch,tam_imagen-786,SEEK_SET);             // Se coloca al inicio de la cabecera auxiliar
   fread((unsigned char *)&cab_aux,1,sizeof(cab_aux),tem_arch);  // Lee y almacena la cabecera auxiliar
                                                        // El siguiente bloque asigna X_nombre al archivo de escritura
   for (i=0;i<=11;i++)
    nom_gif[i]=0;
   nom_gif[0]='x';
   nom_gif[1]='_';
   i=0;
   do
    {
     nom_gif[i+2]=cab_aux.nombre[i];
     i++;
    }while (cab_aux.nombre[i]!=0);
   strcat(nom_gif,siglas);
   esc_arch=fopen(nom_gif,"wb");                        // Abre el archivo destino para su escritura
   if (esc_arch!=NULL)                                  // Si no hay errores continúa
    {                                                   // El siguiente bloque asigna los datos de la cabecera
     gifdp.GIF[0]='G';                                  // Firma
     gifdp.GIF[1]='I';
     gifdp.GIF[2]='F';
```

```
 gifdp.ver[0]='8';                                                    // Versión
 gifdp.ver[1]='9';
 gifdp.ver[2]='a';
 gifdp.xtotal=cab_aux.xmax;                                           // Ancho de pantalla
 gifdp.ytotal=cab_aux.ymax;                                           // Largo de pantalla
 gifdp.describe=215;                                                  // Paquete
 gifdp.fondo=0;                                                       // Color de fondo
 gifdp.aspect_ratio=0;                                                // Aspect ratio
 for (i=0;i<=255;i++)                                                 // Paleta de colores
  {
    gif_pal[i].rojo =cab_aux.pal_rgb[(i*3)+0];
    gif_pal[i].verde=cab_aux.pal_rgb[(i*3)+1];
    gif_pal[i].azul =cab_aux.pal_rgb[(i*3)+2];
  }
 gifdi.separador=',';                                                 // Separador
 gifdi.xminimo=0;                                                     // X mínima
 gifdi.yminimo=0;                                                     // Y mínima
 gifdi.xmaximo=cab_aux.xmax;                                          // X máxima
 gifdi.ymaximo=cab_aux.ymax;                                          // Y máxima
 gifdi.inter=0;                                                       // Paquete, Interlazado
 fwrite((unsigned char *)&gifdp,1,sizeof(gifdp),esc_arch);            // Escribe descriptor de pantalla
 fwrite((unsigned char *)&gif_pal,1,sizeof(gif_pal),esc_arch);        // Escribe paleta de colores
 fwrite((unsigned char *)&gifdi,1,sizeof(gifdi),esc_arch);            // Escribe descriptor de imagen
 aux=8;
 fwrite((unsigned char *)&aux,1,1,esc_arch);                          // Byte de inicialización
 inicia_var();                                                        // Inicializa variables
 fseek(tem_arch,0,SEEK_SET);
 cont_datos=0;                                                        // Contador de datos de archivo temporal
 cont_num   =1;                                                       // Contador de bloque
 cont_num_inicial=0;                                                  // Contador de bloque inicial
 cod_sal[cont_num_inicial]=cod_limpia;                                // Inicializa primer código almacenado
 cod_sal[1]=0;                                                        // Limpia sig. psocición de código
 datos=0;                                                             // Inicializa variables
 cuenta=0;
 desplaza=0;
 vueltas=0;
 resto=0;
 gotoxy(1,1);printf("Porcentaje de conversión :     \%");
 kk=(786+gifdp.xtotal);                                               // Localiza la posición de acuerdo con la dirección
 if (cab_aux.dir==0)
      fseek(tem_arch,0,SEEK_SET);                                     // Se coloca al inicio del archivo
 else
      fseek(tem_arch,tam_imagen-kk,SEEK_SET);                         // Se coloca al final del archivo
 jj=0;                                                                // Inicializa cuenta de renglones
 do                                                                   // Haz
 {
  do                                                                  // Haz
   {
    fread((unsigned char *)&lee_byte,1,1,tem_arch);                   // Lee dato
    cont_datos++;                                                     // Incrementa datos
    if (cont_datos % cab_aux.xmax==0)                                 // Verifica si terminó la línea de datos
        pos_sig_bloque();                                             // Calcula la posición del siguiente bloque
    if (esta_en_tabla(omega[0],omega[1],lee_byte))                    // Si está en la tabla de cadenas
     {
```

```
        omega[0]=omega[1];                                          // Reasigna omega 1
        omega[1]=lee_byte;                                          // Reasigna omega 2
       }
      else                                                          // Si no está en la tabla
       {
        posis_omega=busca_pos_cod(omega[0],omega[1]);               // Busca la posición de las dos omegas
        cod_sal[cont_num]=posis_omega;                              // Almacena código
        cod_sal[cont_num+1]=0;                                      // Limpia siguiente posición
        cadenas[cont_cod][0]=omega[1];                              // Asigna nueva entrada en la tabla
        cadenas[cont_cod][1]=lee_byte;
        omega[1]=lee_byte;                                          // Reasigna omega 1
        cont_cod++;                                                 // Incrementa cuenta códigos en tabla
        cont_num++;                                                 // Incrementa cuenta de bloque
        if (cont_num==3839)                                         // Si rebasa el fin de bloque 3839+257=4096
         {
          fgetpos(tem_arch,&posicion);
          posicion--;                                               // Se regresa una posición
         }
        cadenas[cont_cod][0]=-1;                                    // Limpia siguiente posición
        cadenas[cont_cod][1]=-1;
       }
                                                                    // Mientras no sean 4096 códigos o no se terminen los datos del temporal
     } while ((cont_num!=3839) && (cont_datos<=(tam_imagen-786)));
     for (j=1;j<=8;j++)                                             // Limpia posiciones de los 8 códigos a desglosar
      cod_sal[cont_num+j]=0;
     fseek (tem_arch,posicion,SEEK_SET);
     cont_datos--;                                                  // Decrementa datos del temporal
     posis_omega=busca_pos_cod(omega[0],omega[1]);                  // Busca la posición de las dos omegas
     fin_de_datos=False;                                            // Asigna fin de datos
     if (cont_datos>=(tam_imagen-786))                              // Si se terminaron los datos del temporal
      fin_de_datos=True;                                            // Ffin de datos es veredadero
     cod_sal[cont_num]=posis_omega;                                 // Almacena la posición de omega
     escribe_datos(cont_num,fin_de_datos);                          // Desglosa los códigos en byte y los escribe en disco
     inicia_var();                                                  // Inicializa variables
     cont_num_inicial=1;                                            // Inicializa cont_num_inicial
   } while (cont_datos<(tam_imagen-786));                           // Mientras no se terminen los datos del temporal
   aux=59                                                           // Escribe trailer (fin de archivo)
   fwrite((unsigned char *)&aux,1,1,esc_arch);
  }
 }
 fcloseall();
}


void inicia_var()                                                   // Inicializa variables
{
 int i;
 cod_limpia=256;                                                    // Código de limpieza siempre 256
 cod_final =257;                                                    // Código final
 cont_cod  =258;                                                    // Contador de cada código en la tabla
 maxcod    =pow(2,9);                                               // Máximo bloque codificado
 cont_num  =1;                                                      // Contador de bloque
 posis_omega=0;                                                     // Posición de prefijo
 for (i=0;i<=255;i++)                                               // El siguiente bloqqe inicializa los arreglos para eliminar basura
```

```
 {
  cadenas[i][0]=-1;
  cadenas[i][1]=i;
 }
 cadenas[256][0]=-1;
 cadenas[256][1]=cod_limpia;
 cadenas[257][0]=-1;
 cadenas[257][1]=cod_final;
 cadenas[258][0]=-1;
 cadenas[258][1]=-1;
 omega[0]=-1;
 omega[1]=-1;
}


void pos_sig_bloque()                                   // Determina la posición del siguiente bloque de datos
{
 jj++;                                                  // Incrementa cuenta de renglones
 kk=kk+cab_aux.xmax;                                    // Incrementa cuenta de bloque
 if (cab_aux.dir==0)
   fseek(tem_arch,jj*cab_aux.xmax,SEEK_SET);            // Asigna la posición a partir del inicio
 else
   fseek(tem_arch,tam_imagen-kk,SEEK_SET);              // Asigna la posición a partir del final
 gotoxy(28,1);printf("%3d",(jj*100/cab_aux.ymax));      // Calcula porcentaje de conversión
}


int esta_en_tabla(int cod1,int cod2,int cod3)
{
 int j,esta;
 j=-1;                                                  // Inicializa contador
 if (cod1>=0)                                           // Omega 0 es mayor o igual a cero
  {
   do                                                   // Haz
   {
    j++;                                                // Incrementa j
                    // Mientras no encuentres las dos omegas o no rebase el ontador de cadenas en la tabla
   } while ((j<=cont_cod) && ((cod1!=cadenas[j][0]) || (cod2!=cadenas[j][1])));
   cod2=j;                                              // Asigna omega 1
   omega[0]=-1;                                         // Reasiga omega 0
   omega[1]=j;                                          // Reasigna omega 1
  }
  j=-1;                                                 // Inicializa j
  do                                                    // Haz
  {
   j++;                                                 // Incrementa j
                // Mientras no encuentre omega 1 y el dato leido o no rebase la cuenta de cadenas en la tabla
  } while ((j<=cont_cod) && ((cod2!=cadenas[j][0]) || (cod3!=cadenas[j][1])));
  if (j>cont_cod)                                       // Si j rebasa el contador de cadenas en la tabla
   esta=False;                                          // esta es falso
  else
   esta=True;                                           // esta es verdadero
 return(esta);                                          // Devuelve esta
}
```

```
int busca_pos_cod(int omega,int byte_omega)                    // Busca posición de omega
{
 int k,esta;
 k=-1;                                                         // Inicializa contador
 if (byte_omega<255)                                           // Omega 1 menor a 255
  do                                                           // Haz
   {
    k++;                                                       // Incrementa k
                    // Mientras no encuentre las dos omegas o rebase la cuenta de cadenas en la tabla
   } while ((k<=cont_cod) && ((omega!=cadenas[k][0]) || (byte_omega!=cadenas[k][1])));
 else                                                          // Es mayor a 255
   k=byte_omega;                                               // Asigna seguda omega
 esta=k;                                                       // Asigna k
 return(esta);                                                 // Devuelve k
}


 char cadena[200];
 LONGINT ini,fin,aux1;


 void escribe_datos(int cont_num,int fin_si_no)                // Desglosa los códigos en bytes
 {
  LONGINT cod_aux;                                             // Código auxiliar
  int no_bits;                                                 // Número de bits por código
  int cont_aux;                                                // Contador auxiliar
  void saca_datos(int no_bits);

  no_bits=9;                                                   // Inicializa bits de entrada
  maxcod=512;                                                  // Inicializa máximo bloque codificado
  if (cont_num<3839)                                           // No es fin de bloque
   {
    cont_num++;                                                // Incrementa contador de bloque
    cod_sal[cont_num]=cod_final;                               // Asigna código final
    cont_num=cont_num+2;                                       // Incrementa contador de bloque
   }
  else                                                         // Es fin de bloque
    cod_sal[cont_num]=cod_limpia;                              // Asigna código de limpieza
  ini=cont_num_inicial;                                        // Inicializa ini
  fin=ini+7;                                                   // Inicializa fin
  do                                                           // Haz
  {
   cuenta=0;                                                   // Inicializa cuenta
   cont_aux=0;                                                 // Inicializa cont_aux
   for (i=ini;i<=fin;i++)                                      // Toma 8 códigos
    {
     cod_sal[i]=(cod_sal[i] << desplaza) | resto;              // Asigna nuevo valor de código
     resto=0;                                                  // Inicializa resto
                                        // El siguiente bloque convierte el código a binario
     for (j=0,cod_aux=cod_sal[i];((j<(no_bits+desplaza)) || (cod_aux!=0));j++,cod_aux=cod_aux >> 1)
     {
      aux1=pow(2,j);
      if ((cod_sal[i] & aux1)==aux1)
       cadena[cont_aux+j]='1';
```

```
 else
  cadena[cont_aux+j]='0';
 }
 cont_aux=cont_aux+j;                                   // Asigna cont_aux
 cuenta++;                                              // Incrementa cuenta
 if ((cuenta*no_bits)<cont_aux)                         // Rebasa la longitud recomendable
 {
  if ((i==3839) && (cont_aux>=88))                      // Es fin de bloque y la longitud rebasa a 88
     {
      resto=0;                                          // Inicializa resto
      aux1=0;                                           // Inicializa aux1
      for (k=88;k<=cont_aux;k++)                        // Este bloque asigna los bits restantes y trunca la cadena
       {
        if (cadena[k]=='1')
         resto=resto+pow(2,aux1);
        aux1++;
        cadena[k]='\0';
       }
      cont_aux=88;
     }
  else                                                  // No es fin de bloque
   {
    resto=0;
    aux1=0;
    for (k=cuenta*no_bits;k<cont_aux;k++)               // Este bloque asigna los bits restantes y trunca la cadena
       {
        if (cadena[k]=='1')
         resto=resto+pow(2,aux1);
        aux1++;
        cadena[k]='\0';
       }
      if (i!=3839)
       cont_aux=cuenta*no_bits;
   }
  }
 }
 saca_datos(cont_aux/8);                                // Desglosa los bytes de la cadena
 ini=ini+8;                                             // Incrementa ini
 if ((fin+8)>cont_num)  fin=cont_num;                   // Incremente fin
 else  fin=fin+8;
 if (fin+(257-cont_num_inicial)>maxcod)                 // Es límite de bloque (512, 1024,..)
 {
  maxcod=maxcod*2;                                      // Incrementa máximo bloque
  no_bits++;                                            // Incrementa no_bits
  if  (vueltas>0) desplaza++;                           // Incrementa desplaza
  if (desplaza>=8)                                      // Desplaza es mayor o igual a 8
   {
    desplaza=0;                                         // Inicializa desplaza
    escribe_bytes[datos]=resto;                         // Asigna resto
    datos++;                                            // Incrementa datos
    resto=0;                                            // Inicializa resto
   }
 }
}while (ini<cont_num);                                  // Mientras no termine el bloque
```

```
 if (fin_si_no)                                                    // Se terminaron datos del temporal
  {
   datos--;                                                        // Decrementa datos
   fwrite((unsigned char *)&datos,1,1,esc_arch);          // Escribe cuantos bytes, no necesariamente 254
   fwrite((unsigned char *)&escribe_bytes,1,datos+1,esc_arch);     // Escribe los bytes almacenados
  }
 if ((vueltas+1)==10) vueltas=2;                                   // Asigna vueltas
 else  vueltas++;
 if (((9-vueltas)>=8) ||((9-vueltas)==0))  desplaza=0;             // Asigna desplaza
 else  desplaza=9-vueltas;
  if ((cuenta<8) && (resto==0))                                    // Verifica bits restantes
  {
   aux1=0;                                                         // El sig. bloque asigna bits restantes a resto
   for (k=((cont_aux / 8)*8);k<cont_aux;k++)
    {
     if (cadena[k]=='1')
      resto=resto+pow(2,aux1);
     aux1++;
     cadena[k]='\0';
    }
  }
}


int numero;                                                        // Byte desglosado de num_str


void saca_datos(int no_bits)                                       // Desglosa los bytes de la cadena
{
 char *num_str;                                                    // Cadena de 8 bits
 int j,cont;                                                       // Auxiliares
 void escribe_en_disco();

 for (cont=0;cont<no_bits;cont++)                                  // Desde 1 hasta no_bits (9,10,11,12)
 {
  numero=0;                                                        // Inicializa número
  escribe_bytes[datos]=0;                                          // Inicializa arreglo
                          // El siguiente  bloque proporciona la potencia a 2 de cada bi1 '1'de la cadena
  for (j=(cont*8);j<((cont+1)*8);j++);
   {
    if (cadena[j]=='1')
     escribe_bytes[datos]=escribe_bytes[datos]+pow(2,numero);      // Almacena el byte resultante
    numero++;
   }
  datos++;                                                         // Incrementa cuenta de datos almacenados
  if (datos>=254)                                                  // Si datos= 254
   {
    escribe_en_disco();                                            // Manda a escribir al archivo destino
   }
 }
}
```

```
void escribe_en_disco()                                   Escribe los bytes resultantes en el archivo destino
{
 fwrite((unsigned char *)&datos,1,1,esc_arch);            Escribe cuantos bytes (254, excepto al final)
 fwrite((unsigned char *)&escribe_bytes,1,datos,esc_arch);  Escribe los bytes almacenados
 datos=0;                                                 Inicializa datos
}
```

Una de las imágenes procesadas con el programa es amber.gif.

AMBER.GIF

El programa da como resultado los siguientes datos:

```
Nombre de Archivo :amber.gif
Identificación : GIF      Posición del puntero :13
Versión        : 89a
Total  en x    : 320
Total  en y    : 200
Bits por pixel : 8
Fondo          : 0
Aspect ratio  (0=si, otro=no) : 0
Separador      : ,        Posición del puntero :791
xmínimo        : 0
ymínimo        : 0
xmáximo        : 320
ymáximo        : 200
Interlazado   (1=si, 0=no): 0
Mapa de Color Local (1=si, 0=no): 0_
```

Espere un momento en el cual se descomprime la imagen y se crea el temporal enseguida aparece:

```
Porcentaje de conversión : 100 %
La conversión se completo con éxito
```

V.7 Compresión del Formato de Archivo TIF

El método de compresión que utiliza TIF al igual que TGA es RLE, recuerde que su límite es 127 y utiliza bytes con signo (números negativos). Cabe resaltar que TIF utiliza tiras de datos, cada tira tiene un número determinado de líneas, este número es calculado mediante una operación logaritmica. Ya que cada tira es independiente de las demás, el proceso de compresión se aplica a cada tira.

V.7.1 En Lenguaje Pascal

El procedimiento en Pascal se codifica de la siguiente manera:

```
Procedure  Esc_imagen_Tif;
Type
 buffer=Array [0..4096] of byte;                          {Arreglo para almacenar los bytes}

Var
 num_tiras,lineas_x_tira:Integer;                         {Número de tiras y líneas por tira}
 tam_imagen,j,k,i,c_tira:Longint;                         {Tamaño de imagen y auxiliares}
 color,                                                   {Color del byte}
  unbyte,                                                 {Byte actual}
   bytepen,                                               {Byte penúltimo}
    byteant,                                              {Byte antepenúltimo}
     sin_comp:byte;                                       {Sin comprimir}
 cuantos:ShortInt;                                        {Bytes comprimidos}
 nom_tif:string;                                          {Nombre del archivo}
 desplaza,                                                {Desplazamiento del bloque de bytes}
  cta_byte:Array [1..500] of Longint;                     {Cuenta de bytes por bloque}
 bandera:byte;                                            {Bandera tipo de compresión}
 arr_bytes:Array [0..260] of byte;                        {Arreglo para almacenar bytes de color}
 buf:^buffer;                                             {Arreglo para alamacenar bytes}


 Procedure pos_sig_bloque(var j:longint);                 {Determina la posisicón del siguiente bloque de datos}
 Begin
  inc(j);                                                 {Incrementa cuenta de renglones}
  inc(k,cab_aux.xmax);                                    {Incrementa cuenta de bloques}
  if (cab_aux.dir=0) Then
    seek(tem_arch,j*cab_aux.xmax)                         {Asigna la posición a partir del inicio}
  else
    seek(tem_arch,filesize(tem_arch)-k);                  {Asigna la posición a partir del final}
  gotoxy(28,1);write((j*100/cab_aux.ymax):3:0);           {Calcula el porcentaje de compresión}
 End;


 Procedure RLE;                                           {Procedimiento para comprimir con RLE}
 Begin
  inc(i);                                                 {Incrementa cuenta de columnas}
   if cuantos=127 Then                                    {Si bytes comprimidos igual a 127}
    cuantos:=cuantos*-1                                   {Multiplica por (menos 1)}
```

```
else
  cuantos:=(cuantos +1)*-1;                          {Incrementa cuantos y multiplica por (menos 1)}
blockwrite(esc_arch,cuantos,1,escribio);             {Escribe el número de bytes comprimidos}
blockwrite(esc_arch,byteant,1,escribio);             {Escribe el valor de dicho byte}
bytepen:=unbyte                                      {El siguiente bloque asigna nuevos valores e inicializa}
unbyte:=buf^[i+2];
bandera:=0;
cuantos:=0;
sin_comp:=0;
End;


Procedure RAW;                                       {Procedimiento para comprimir con RAW}
Var
 cont:integer;
Begin
 dec(sin_comp);                                      {Decrementa sin comprimir}
 blockwrite(esc_arch,sin_comp,1,escribio);           {Escribe cuantos datos sin comprimir}
 for cont:=0 to sin_comp do
  Begin
   blockwrite(esc_arch,arr_bytes[cont],1,escribio);  {Escribe cada uno de los datos sin comprimir}
  End;
 sin_comp:=0;                                        {El siguiente bloque inicializa variables}
 cuantos:=0;
 bandera:=0;
End;


Begin                                                {El siguiente bloque abre el archivo temporal para su lectura}
 new(buf);
 Assign(tem_arch,'T_e_m_p!.SSS');
  {Si-}
 Reset(tem_arch,1);
   {Si+}
 if (ioresult=0) Then                                {Si no hay errores continúa con el proceso}
  Begin
   tam_imagen:=filesize(tem_arch)-786;               {Calcula tamaño de la imagen}
   posicion:=filepos(tem_arch);
   Seek(tem_arch,filesize(tem_arch)-786);`           {Se coloca al inicio de la cabecera auxiliar}
   blockRead(tem_arch,cab_aux,sizeof(cab_aux),sololee);  {Lee y alamcena la cabecera auxiliar}
   nom_tif:='';i:=1;                                 {Inicializa variables}
                                                     {El siguiente bloque asigna X_nombre al archivo de escritura}
   While (cab_aux.nombre[i]<>chr(0)) do
    Begin
     nom_tif:=concat(nom_tif,cab_aux.nombre[i]);
     inc(i);
    End;
   If Length(nom_tif)>=5 Then
     nom_tif:='X_'+copy(nom_tif,1,6)+'.Tif'
   else
     nom_tif:='X_'+nom_tif+'.Tif';
   Assign(esc_arch,nom_tif);                         {El siguiente bloque abre el archivo destino para su escritura}
    {Si-}
```

```
  rewrite(esc_arch,1);
  {Si+}
 if (ioresult=0) Then                                          {Si no hay errores continúa con el proceso}
  Begin                                                        {El siguiente bloque asigna los datos de la cabecera}
   TIFFC.iden:='II'                                            {Identificación}
   TIFFC.ver:=42;                                              {Versión}
   TIFFC.pos_dir:=0;          {Posición del primer directorio (IFD), modificar despues de la compresión}
   lineas_x_tira:=(trunc(ln(cab_aux.ymax/10)/ln(2))*2)+2;      {Líneas por tira}
   num_tiras:=ifi(round(cab_aux.ymax/lineas_x_tira)>=(cab_aux.ymax/lineas_x_tira)
            ,round(cab_aux.ymax/lineas_x_tira)
            ,round(cab_aux.ymax/lineas_x_tira)+1);             {Número de tiras}
   blockwrite(esc_arch,TIFFC ,sizeof(TIFFC) ,escribio);       {Escribe cabecera}
   posicion:=Filesize(esc_arch);
   TIFFD.entradas:=13;                                        {Número de etiquetas en el TIF}
   gotoxy(1,1);write('Porcentaje de conversión :    %');
   k:=(786+cab_aux.xmax);          {Localiza la posición [10] de acuerdo con la dirección de la imagen}
   if (cab_aux.dir=0) Then Seek(tem_arch,0)                   {Se coloca al inicio del archivo}
      Seek(tem_arch,filesize(tem_arch)-k);      {Se coloca al final del archivo menos la posición anterior}
 BlockRead(tem_arch,buf^,cab_aux.xmax,sololee);        {Lee un bloque de datos de longitud xmaximo}
                                                             {El siguiente bloque inicializa variables}
   i:=0;                                                     {Cuenta de columnas}
   j:=0;                                                     {Cuenta de renglones}
   cuantos:=0;
   sin_comp:=0;
   bandera:=0;
   c_tira:=1;
    Repeat                                                   {Repite el siguiente bloque}
     Repeat                                                  {Repite el siguiente bloque}
      byteant:=buf^[i];                                      {Asigna byteant}
      bytepen:=buf^[i+1];                                    {Asigna bytepen}
      unbyte:=buf^[i+2];                                     {Asigna unbyte}
      If (byteant<>bytepen) or (bytepen<>unbyte) Then        {Si algún byte es diferente}
       Begin
        if bandera=1 Then                                    {Hay datos comprimidos}
          RLE                                                {Escribe RLE}
        else                                                 {No hay datos comprimidos}
         Begin                                               {Almacena datos sin comprimir}
          arr_bytes[sin_comp]:=byteant;
          inc(sin_comp);
          if sin_comp>=128 Then                              {Verifica sea menor a 128}
           RAW                                               {Escribe RAW}
          else                                               {Asigna bandera sin comprimir}
            bandera:=2;
         End;
       End
      Else                                                   {Hay bytes iguales}
       Begin
        if (bandera=2)  Then                                 {Revisa si se envian bytes sin comprimir}
          RAW;                                               {Escribe RAW}
        inc(cuantos);                                        {Incrementa cuantos}
        bandera:=1;                                          {Cuenta 1 byte compreso}
```

[10] Anteriormente la dirección de la imagen se ha designado como 0 (cero) ya que las imágenes TIF se despliegan de la esquina superior izquierda a la esquina inferior derecha.

```
      if cuantos>=127 Then                                        {Verifica sea menor a 128}
        RLE;                                                      {Escribe RLE}
     End;
    inc(i);                                                       {Incrementa cuenta de columnas}
   Until (i>=cab_aux.xmax);                                       {Hasta terminar dicho renglón}
   if bandera=2 Then RAW;                                         {Revisa si quedo un conteo sin terminar}
   dec(cuantos,2);                                                {Decrementa cuantos}
   if bandera=1 Then RLE;                                         {Escribe los últimos bytes}
   cuantos:=0;                                                    {Inicializa variables}
   bandera:=0;
   i:=0;
   pos_sig_bloque(j);                                             {Calcula la posición del siguiente bloque}
BlockRead(tem_arch,buf^,cab_aux.xmax,sololee);                    {Lee otro bloque de datos}
   if j mod lineas_x_tira =0  then
    Begin          {El siguiente bloque asigna y almacena el desplazamiento y cuenta de byte por bloque}
     desplaza[c_tira]:=posicion;
     cta_byte[c_tira]:=Filesize(esc_arch)-posicion;
     posicion:=Filesize(esc_arch);
     inc(c_tira);
    End;
   Until (j>=cab_aux.ymax);                                       {Hasta terminar todos los renglones}
  desplaza[c_tira]:=posicion;                                     {Asigna nuevos valores si hubo cambios}
  cta_byte[c_tira]:=Filesize(esc_arch)-posicion;
  posicion:=filesize(esc_arch);

  For j:=0 to 255 do                                              {El siguiente bloque escribe la paleta de colores}
   Begin                                                          {Rojo}
    color:=(cab_aux.pal_rgb[(j*3)+1] and SFF) shr 8;
    blockwrite(esc_arch,color,1,escribio);
    color:=(cab_aux.pal_rgb[(j*3)+1] and SFF);
    blockwrite(esc_arch,color,1,escribio);
   End;

  For j:=0 to 255 do                                              {Verde}
   Begin
    color:=(cab_aux.pal_rgb[(j*3)+2] and SFF) shr 8;
    blockwrite(esc_arch,color,1,escribio);
    color:=(cab_aux.pal_rgb[(j*3)+2] and SFF);
    blockwrite(esc_arch,color,1,escribio);
   End;

  For j:=0 to 255 do                                              {Azul}
   Begin
    color:=(cab_aux.pal_rgb[(j*3)+3] and SFF) shr 8;
    blockwrite(esc_arch,color,1,escribio);
    color:=(cab_aux.pal_rgb[(j*3)+3] and SFF);
    blockwrite(esc_arch,color,1,escribio);
   End;

  i:=Filesize(esc_arch);          {El siguiente bloque termina de asignar los valores de las etiquetas}
  blockwrite(esc_arch,desplaza ,(num_tiras)*4 ,escribio);
  j:=Filesize(esc_arch);
  blockwrite(esc_arch,cta_byte ,(num_tiras)*4,escribio);
```

```
TIFFD.etiqueta_tif[01].etiqueta:=254;                    {Nvoti_subarchivo}
TIFFD.etiqueta_tif[01].tipo_dato:=4;
TIFFD.etiqueta_tif[01].cont_dato:=1;
TIFFD.etiqueta_tif[01].des_dato:=0;
TIFFD.etiqueta_tif[02].etiqueta:=256;                    {Ancho_imagen}
TIFFD.etiqueta_tif[02].tipo_dato:=4;
TIFFD.etiqueta_tif[02].cont_dato:=1;
TIFFD.etiqueta_tif[02].des_dato:=cab_aux.xmax;
TIFFD.etiqueta_tif[03].etiqueta:=257;                    {Largo_imagen}
TIFFD.etiqueta_tif[03].tipo_dato:=4;
TIFFD.etiqueta_tif[03].cont_dato:=1;
TIFFD.etiqueta_tif[03].des_dato:=cab_aux.ymax;
TIFFD.etiqueta_tif[04].etiqueta:=258;                    {Bits_muestra}
TIFFD.etiqueta_tif[04].tipo_dato:=3;
TIFFD.etiqueta_tif[04].cont_dato:=1;
TIFFD.etiqueta_tif[04].des_dato:=8;
TIFFD.etiqueta_tif[05].etiqueta:=259;                    {Compresión RLE Packbits}
TIFFD.etiqueta_tif[05].tipo_dato:=3;
TIFFD.etiqueta_tif[05].cont_dato:=1;
TIFFD.etiqueta_tif[05].des_dato:=32773;
TIFFD.etiqueta_tif[06].etiqueta:=262;                    {Inter_fotome}
TIFFD.etiqueta_tif[06].tipo_dato:=3;
TIFFD.etiqueta_tif[06].cont_dato:=1;
TIFFD.etiqueta_tif[06].des_dato:=3;
TIFFD.etiqueta_tif[07].etiqueta:=273;                    {Desplaza_tira}
TIFFD.etiqueta_tif[07].tipo_dato:=4;
TIFFD.etiqueta_tif[07].cont_dato:=num_tiras;
TIFFD.etiqueta_tif[07].des_dato:=i;
TIFFD.etiqueta_tif[08].etiqueta:=274;                    {Orientación}
TIFFD.etiqueta_tif[08].tipo_dato:=3;
TIFFD.etiqueta_tif[08].cont_dato:=1;
TIFFD.etiqueta_tif[08].des_dato:=1;
TIFFD.etiqueta_tif[09].etiqueta:=277;                    {Mues_x_pixel}
TIFFD.etiqueta_tif[09].tipo_dato:=3;
TIFFD.etiqueta_tif[09].cont_dato:=1;
TIFFD.etiqueta_tif[09].des_dato:=1;
TIFFD.etiqueta_tif[10].etiqueta:=278;                    {Lineas_x_tiras}
TIFFD.etiqueta_tif[10].tipo_dato:=4;
TIFFD.etiqueta_tif[10].cont_dato:=1;
TIFFD.etiqueta_tif[10].des_dato:=lineas_x_tira;
TIFFD.etiqueta_tif[11].etiqueta:=279;                    {cta_byte_tira}
TIFFD.etiqueta_tif[11].tipo_dato:=4;
TIFFD.etiqueta_tif[11].cont_dato:=num_tiras;
TIFFD.etiqueta_tif[11].des_dato:=j;
TIFFD.etiqueta_tif[12].etiqueta:=284;                    {Config_planar}
TIFFD.etiqueta_tif[12].tipo_dato:=3;
TIFFD.etiqueta_tif[12].cont_dato:=1;
TIFFD.etiqueta_tif[12].des_dato:=1;
TIFFD.etiqueta_tif[13].etiqueta:=320;                    {Paleta}
TIFFD.etiqueta_tif[13].tipo_dato:=3;
TIFFD.etiqueta_tif[13].cont_dato:=768;
TIFFD.etiqueta_tif[13].des_dato:=posicion;               {La paleta se localiza al final de los datos}
TIFFD.Siguiente:=0;                                      {Siguiente desplazamiento del IFD,no hay = 0}
posicion:=filesize(esc_arch);
```

```
    TIFFC.Pos_dir:=Filesize(esc_arch);                          {Modificación del primer directorio}
    blockwrite(esc_arch,TIFFD.Entradas,sizeof(TIFFD.Entradas),escribio);
    blockwrite(esc_arch,TIFFD.etiqueta_tif,sizeof(etiquetas_tif)*13,escribio);
    blockwrite(esc_arch,TIFFD.siguiente,sizeof(TIFFD.siguiente),escribio);
    Seek(esc_arch,0);                                           {Escribe la cabecera}
    blockwrite(esc_arch,TIFFC        ,sizeof(TIFFC)        ,escribio);
    close(esc_arch);
   End;
  End;
 dispose(buf);
End;
```

Una de las imágenes procesadas con el programa es logomk3.tif.

LOGOMK3.TIF

El programa da como resultado los siguientes datos:

```
Nombre de Archivo :logomk3.tif
Identificacion    :II      Posición del puntero :8
Versión           :42
OFFSET -          :275440
Numero de etiquetas: 13
Siguiente imagen : 0
Nombre            etiqueta     Tipo        Version   Datacount    Dataoffset
Nvoti_Subarchi    254          Long    (4)  -- U5 U6  1            0
Ancho_imagen      256          Short   (4)  U4 U5 U6  1            570
Largo_imagen      257          Short   (4)  U4 U5 U6  1            480
Bits_muestra      258          Short   (3)  U4 U5 U6  1            8
Compresión        259          Short   (3)  U4 U5 U6  1            1
Inter_fotome      262          Short   (3)  U4 U5 U6  1            3
Desplaza_Tira     273          Long    (4)  U4 U5 U6  37           275144
Orientacion       274          Short   (3)  U4 U5 U6  1            1
Mues x pixel      277          Short   (3)  U4 U5 U6  1            1
Lineas_x_Tiras    278          Long    (4)  U4 U5 U6  1            13
cta_byte_Tira     279          Long    (4)  U4 U5 U6  37           275292
Config_planar     284          Short   (3)  U4 U5 U6  1            1
Paleta            320          Short   (3)  -- U5 U6  768          273600
```

Espere un momento en el cual se descomprime la imagen y se crea el temporal enseguida aparece:

```
Porcentaje de conversión : 100 %
La conversión se completo con éxito
```

V.7.2 En Lenguaje C

La función en C se codifica de la siguiente manera:

```
LONGINT tam_imagen,c_tira;
 int
  unbyte,                                                    // Byte actual
   byteant,                                                  // Byte anterior
    bytepen,                                                 // Byte penúltimo
     sin_comp,                                               // Sin comprimir
      bandera,                                               // Bandera tipo de compresión
        num_tiras,                                           // Número de tiras
          color,                                             // Color
           lineas_x_tira;                                    // Líneas por tira
 char cuantos;                                               // Bytes comprimidos
 BYTE arr_bytes[256];                                        // Arreglo para almacenar códigos
 LONGINT desplaza[501],                                      // Desplazamiento del bloque de bytes
             cta_byte[501];                                  // Cuenta de bytes por bloque
 char buf[4096];                                             // Arreglo para almacenar bytes

 void Esc_imagen_TIF()
 {

  char *nom_aux,*siglas=".tif";                              // Extensión del archivo
  unsigned char nom_TIF[12];                                 // Nombre del archivo

  void pos_sig_bloque();                                     // Determina la posición del siguiente bloque
  void RLE();                                                // Escribe RLE
  void RAW();                                                // Escribe RAW

  tem_arch=fopen("T_E_M_P!.$$$","rb");                       // Abre el archivo temporal para su lectura
   if (tem_arch!=NULL)                                       // Si no hay errores continúa
    {
     tam_imagen=filelength(open("T_E_M_P!.$$$",O_RDWR));     // Calcula tamaño de la imagen
     fseek(tem_arch,tam_imagen-786,SEEK_SET);                // Se coloca al inicio de la cabecera auxiliar
     fread((unsigned char *)&cab_aux,1,sizeof(cab_aux),tem_arch);   // Lee y almacena la cabecera auxiliar

                                                             // El siguiente bloque asigna X_nombre al archivo de escritura
     for (i=0;i<=11;i++)
       nom_TIF[i]=0;
     nom_TIF[0]='x';
     nom_TIF[1]='_';
    i=0;
     do
      {
       nom_TIF[i+2]=cab_aux.nombre[i];
       i++;
      }while (cab_aux.nombre[i]!=0);
      strcat(nom_TIF,siglas);
    esc_arch=fopen(nom_TIF,"wb");                            // Abre el archivo destino para su escritura
      if (esc_arch!=NULL)                                    // Si no hay errores continúa
       {                                                     // El siguiente bloque asigna los datos de la cabecera
```

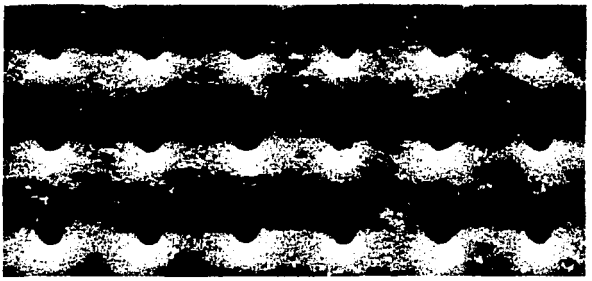
```
 tiffc.iden[0]=73;                                                    // Identificación
 tiffc.iden[1]=73;                                                    // Identificación
 tiffc.ver=42;                                                        // Versión
 tiffc.pos_dir=0;          // Posición del primer directorio (IFD) modificar después de la compresión
 lineas_x_tira=((log(cab_aux.ymax/10) log(2))*2)+2;                   // Líneas por tira
 if ((cab_aux.ymax % lineas_x_tira)==0)
   num_tiras=(cab_aux.ymax/lineas_x_tira);                            // Número de tiras
 else
   num_tiras=(cab_aux.ymax/lineas_x_tira)+1;
 fwrite((unsigned char *)&tiffc,1,sizeof(tiffc),esc_arch);            // Escribe cabecera
 fgetpos(esc_arch,&posicion);
 tiffd.entradas=13;                                                   // Número de etiquetas en el TIF
 gotoxy(1,1);printf("Porcentaje de conversión :    \%");             // Inicia método de compresión
 k=(786+cab_aux.xmax);          // Localiza la posición de acuerdo con la dirección de la imagen
 if (cab_aux.dir==0)
 fseek(tem_arch,0,SEEK_SET);                                          // Se coloca al inicio del archivo
 else
  fseek(tem_arch,tam_imagen-k,SEEK_SET);      // Se coloca al final del archivo menos la posición
 fread((char *)&buf,1,sizeof(buf),tem_arch);                          // Lee un bloque de datos de longitud xmaximo
                                                                      // El siguiente bloque inicializa variables
 i=0;                                                                 // Cuenta de columnas
 j=0;                                                                 // Cuenta de renglones
 cuantos=0;
 sin_comp=0;
 bandera=0;
 c_tira=0;
 do                                                                   // Haz
 {
   do                                                                 // Haz
   {
     byteant=buf[i];                                                  // Asigna byteant
     bytepen=buf[i+1];                                                // Asigna bytepen
     unbyte=buf[i+2];                                                 // Asigna unbyte
     if ((byteant!=bytepen) || (bytepen!=unbyte))                     // Si algún byte es diferente
     {
         if (bandera==1)                                              // Hay datos comprimidos
           RLE();                                                     // Escribe RLE
         else                                                         // No hay datos comprimidos
         {                                                            // Almacena datos sin comprimir
           arr_bytes[sin_comp]=byteant;
           sin_comp++;
           if (sin_comp>=128)                                         // Verifica sea menor a 128
            RAW();                                                    // Escribe RAW
           else
            bandera=2;                                                // Asigna bandera sin comprimir
         }
     }
    else                                                              // Hay bytes iguales
    {
      if (bandera==2)                                                 // Revisa si se envían bytes sin comprimir
         RAW();                                                       // Escribe RAW
      cuantos++;                                                      // Incrementa cuantos
      bandera=1;                                                      // Cuenta 1 byte compreso
      if (cuantos>=127)                                               // Verifica sea menor a 128
```

```
        RLE();                                                    // Escribe RLE
      }
      i++;                                                        // Incrementa cuenta de columnas
    } while(i<cab_aux.xmax);                                      // Mientras no se termine dicho renglón
  if (bandera==2)  RAW();                                         // Revisa si quedo un conteo sin terminar
  cuantos=cuantos-2;                                              // Decrementa cuantos
  if (bandera==1) RLE();                                          // Escribe los últimos bytes
  cuantos=0;                                                      // Inicializa variables
  bandera=0;
  i=0;
  pos_sig_bloque();                                               // Calcula la posición del siguiente bloque
  fread((char *)&buf,1,cab_aux.xmax,tem_arch);                    // Lee otro bloque de datos
  if ((j % lineas_x_tira) ==0)
   {          // El siguiente bloque asigna y almacena desplazamiento y cuenta de bytes por bloque
    desplaza[c_tira]=posicion;
    fgetpos(esc_arch,&posicion);
    cta_byte[c_tira]=posicion-desplaza[c_tira];
    c_tira++;
   }
}while(j<cab_aux.ymax);                                           // Mientras no termine todos los renglones
desplaza[c_tira]=posicion;                                        // Asigna nuevos valores si hubo cambios
fgetpos(esc_arch,&posicion);
cta_byte[c_tira]=posicion-desplaza[c_tira];

                                                                  // El siguiente bloque escribe la paleta de colores

 for (j=0;j<=255;j++)                                             {Rojo}
  {
   color=(cab_aux.pal_rgb[(j*3)] & 0xFF) >> 8;
   fwrite((unsigned char *)&color,1,1,esc_arch);
   color=(cab_aux.pal_rgb[(j*3)] & 0xFF);
   fwrite((unsigned char *)&color,1,1,esc_arch);
  }
                                                                  {Verde}
 for (j=0;j<=255;j++)
  {
   color=(cab_aux.pal_rgb[(j*3)+1] & 0xFF) >> 8;
   fwrite((unsigned char *)&color,1,1,esc_arch);
   color=(cab_aux.pal_rgb[(j*3)+1] & 0xFF);
   fwrite((unsigned char *)&color,1,1,esc_arch);
   }

 for (j=0;j<=255;j++)                                             {Azul}
  {
   color=(cab_aux.pal_rgb[(j*3)+2] & 0xFF) >> 8;
   fwrite((unsigned char *)&color,1,1,esc_arch);
   color=(cab_aux.pal_rgb[(j*3)+2] & 0xFF);
   fwrite((unsigned char *)&color,1,1,esc_arch);
  }
                                                                  // El siguiente bloque termina de asignar los valores de las etiquetas
 fgetpos(esc_arch,&pos_des);
 fwrite((LONGINT *)&desplaza,1,(num_tiras*4),esc_arch);
 fgetpos(esc_arch,&pos_cta);
 fwrite((LONGINT *)&cta_byte,1,(num_tiras*4),esc_arch);
```

```
tiffd.etiqueta_TIF[ 0].etiqueta=254;                    // Nvoti_subarchivo
tiffd.etiqueta_TIF[ 0].tipo_dato=4;
tiffd.etiqueta_TIF[ 0].cont_dato=1;
tiffd.etiqueta_TIF[ 0].des_dato=0;
tiffd.etiqueta_TIF[ 1].etiqueta=256;                    // Ancho_imagen
tiffd.etiqueta_TIF[ 1].tipo_dato=4;
tiffd.etiqueta_TIF[ 1].cont_dato=1;
tiffd.etiqueta_TIF[ 1].des_dato=cab_aux.xmax;
tiffd.etiqueta_TIF[ 2].etiqueta=257;                    // Largo_imagen
tiffd.etiqueta_TIF[ 2].tipo_dato=4;
tiffd.etiqueta_TIF[ 2].cont_dato=1;
tiffd.etiqueta_TIF[ 2].des_dato=cab_aux.ymax;
tiffd.etiqueta_TIF[ 3].etiqueta=258;                    // Bits_muestra
tiffd.etiqueta_TIF[ 3].tipo_dato=3;
tiffd.etiqueta_TIF[ 3].cont_dato=1;
tiffd.etiqueta_TIF[ 3].des_dato=8;
tiffd.etiqueta_TIF[ 4].etiqueta=259;                    // Compresión RLE Packbits
tiffd.etiqueta_TIF[ 4].tipo_dato=3;
tiffd.etiqueta_TIF[ 4].cont_dato=1;
tiffd.etiqueta_TIF[ 4].des_dato=32773;
tiffd.etiqueta_TIF[ 5].etiqueta=262;                    // Inter_fotome
tiffd.etiqueta_TIF[ 5].tipo_dato=3;
tiffd.etiqueta_TIF[ 5].cont_dato=1;
tiffd.etiqueta_TIF[ 5].des_dato=3;
tiffd.etiqueta_TIF[ 6].etiqueta=273;                    // Desplaza_tira
tiffd.etiqueta_TIF[ 6].tipo_dato=4;
tiffd.etiqueta_TIF[ 6].cont_dato=num_tiras;
tiffd.etiqueta_TIF[ 6].des_dato=pos_des;
tiffd.etiqueta_TIF[ 7].etiqueta=274;                    // Orientación
tiffd.etiqueta_TIF[ 7].tipo_dato=3;
tiffd.etiqueta_TIF[ 7].cont_dato=1;
tiffd.etiqueta_TIF[ 7].des_dato=1;
tiffd.etiqueta_TIF[ 8].etiqueta=277;                    // Mues_x_pixel
tiffd.etiqueta_TIF[ 8].tipo_dato=3;
tiffd.etiqueta_TIF[ 8].cont_dato=1;
tiffd.etiqueta_TIF[ 8].des_dato=1;
tiffd.etiqueta_TIF[ 9].etiqueta=278;                    // Lineas_x_tiras
tiffd.etiqueta_TIF[ 9].tipo_dato=4;
tiffd.etiqueta_TIF[ 9].cont_dato=1;
tiffd.etiqueta_TIF[ 9].des_dato=lineas_x_tira;
tiffd.etiqueta_TIF[10].etiqueta=279;                    // cta_byte_tira
tiffd.etiqueta_TIF[10].tipo_dato=4;
tiffd.etiqueta_TIF[10].cont_dato=num_tiras;
tiffd.etiqueta_TIF[10].des_dato=pos_cta;
tiffd.etiqueta_TIF[11].etiqueta=284;                    // Config_planar
tiffd.etiqueta_TIF[11].tipo_dato=3;
tiffd.etiqueta_TIF[11].cont_dato=1;
tiffd.etiqueta_TIF[11].des_dato=1;
tiffd.etiqueta_TIF[12].etiqueta=320;                    // Paleta
tiffd.etiqueta_TIF[12].tipo_dato=3;
tiffd.etiqueta_TIF[12].cont_dato=768;
tiffd.etiqueta_TIF[12].des_dato=posicion;               // La paleta se localiza al final de los datos
tiffd.siguiente=0;                                      // Siguiente desplazamiento del IFD,no hay = 0
fgetpos(esc_arch,&posicion);
```

```
        tiffc.pos_dir=posicion;                                   // Modificación del primer directorio
        fwrite((unsigned char *)&tiffd.entradas,1,sizeof(tiffd.entradas),esc_arch);
        fwrite((unsigned char *)&tiffd.etiqueta_TIF,1,sizeof(etiquetas_TIF)*13,esc_arch);
        fwrite((unsigned char *)&tiffd.siguiente,1,sizeof(tiffd.siguiente),esc_arch);
        fseek(esc_arch,0,SEEK_SET);                               // Escribe la cabecera
        fwrite((unsigned char *)&tiffc,1,sizeof(tiffc),esc_arch);
        fclose(esc_arch);
    }
  }
}


void pos_sig_bloque()                                   // Determina la posición del siguiente bloque de datos
{
 j++;                                                   // Incrementa cuenta de renglones
 k=k+cab_aux.xmax;                                      // Incrementa cuenta de bloque
 if (cab_aux.dir==0)
  fseek(tem_arch,j*cab_aux.xmax,SEEK_SET);              // Asigna la posición a partir del inicio
 else
  fseek(tem_arch,tam_imagen-k,SEEK_SET);                Asigna la posición a partir del final
 gotoxy(28,1);printf("%3d",(j*100/cab_aux.ymax));      // Calcula porcentaje de conversión
}


void RLE()                                              // Procedimiento para comprimir con RLE
{
 i++;                                                   // incrementa cuenta de columnas
 if (cuantos==127)                                      // Si bytes comprimidos es igual a 127
  cuantos=cuantos*-1;                                   // Multiplica por (menos 1)
 else
  cuantos=(cuantos +1)*-1;                              // Incrementa cuantos y multiplica por (menos 1)
 fwrite((unsigned char *)&cuantos,1,1,esc_arch);        // Escribe el número de bytes comprimidos
 fwrite((unsigned char *)&byteant,1,1,esc_arch);        // Escribe el valor de dicho byte

                                                        // El siguiente bloque asigna nuevos valores e inicializa
 bytepen=unbyte;
 unbyte=buf[i+2];
 bandera=0;
 cuantos=0;
 sin_comp=0;
}


void RAW ()                                             // Procedimiento para comprimir con RAW
{
 int cont;
 sin_comp--;                                            // Decrementa sin comprimir
 fwrite((unsigned char *)&sin_comp,1,1,esc_arch);       // Escribe cuantos datos sin comprimir
 for(cont=0;cont<=sin_comp;cont++)
  fwrite((BYTE *)&arr_bytes[cont],1,1,esc_arch);        // Escribe cada uno de los datos sin comprimir
 sin_comp=0;                                            // El siguiente bloque inicializa variables
 cuantos=0;
 bandera=0;
}
```

Una de las imágenes procesadas con el programa es crack.tif.



CRACK.TIF

El programa da como resultado los siguientes datos:

```
Nombre de Archivo :crack.tif
Identificacion    :II      Posición del puntero :8
Versión           :42
OFFSET -          :195048
Numero de etiquetas: 13
Siguiente imagen : 0
Nombre            etiqueta      Tipo          Version   Datacount    Dataoffset
Nvoti_Subarchi    254           Long     (4)  -- V5 V6  1            0
Ancho_imagen      256           Short    (4)  V4 V5 V6  1            604
Largo_imagen      257           Short    (4)  V4 V5 V6  1            320
Bits_muestra      258           Short    (3)  V4 V5 V6  1            8
Compresión        259           Short    (3)  V4 V5 V6  1            1
Inter_fotome      262           Short    (3)  V4 V5 V6  1            3
Desplaza_Tira     273           Long     (4)  V4 V5 V6  27           194824
Orientacion       274           Short    (3)  V4 V5 V6  1            1
Mues x pixel      277           Short    (3)  V4 V5 V6  1            1
Lineas_x_Tiras    278           Long     (4)  V4 V5 V6  1            12
cta_byte_Tira     279           Long     (4)  V4 V5 V6  27           194932
Config_planar     284           Short    (3)  V4 V5 V6  1            1
Paleta            320           Short    (3)  -- V5 V6  768          193288
```

Espere un momento en el cual se descomprime la imagen y se crea el temporal enseguida aparece:

```
Porcentaje de conversión : 100 %
La conversión se completo con éxito
```

V.8 Compresión del Formato de Archivo TGA

TGA utiliza el método de compresión RLE con límite de 128. Como recordará existen 4 tipos de imagénes analizados en este trabajo, siguiendo el estándar de 8 bits, 256 colores se comprimirá en tga tipo 9 RLE-RAW Packet; en el código de lectura de la imagen en los tipos 2 y 10 se crea una paleta como reducción de una imagen de 16 millones, esta paleta será la que el archivo temporal almacene, así que una vez leída la imagen se debe asignar los valores de la paleta creada a el arreglo de la estructura temporal.

V.8.1 En Lenguaje Pascal

```
Procedure Esc_imagen_TGA;                                   {Realiza la conversión al formato TGA}
Type
 buffer=Array [0..4096] of byte;                            {Arreglo para almacenar los datos}

Var
  cuantos,                                                  {Contador de datos comprimidos}
    unbyte,                                                 {Byte último}
      bytepen,                                              {Byte penúltimo}
       byteant,                                             {Byte antepenúltimo}
        sin_comp:byte;                                      {Sin comprimir}
  i,j,k,tam_imagen:longint;                                 {Variables auxiliares, tamaño de imagen}
  nom_tga:string;                                           {Nombre del archivo}
  bandera:byte;                                             {Bandera tipo de compresión}
  arr_bytes:Array [0..260] of byte;                         {Arreglo de bytes comprimidos}
  buf:^buffer;                                              {Variable para arreglo de datos}


 Procedure pos_sig_bloque(var j:longint);                   {Determina la posición del siguiente bloque de datos}
Begin
 inc(j);                                                    {Incrementa cuenta de renglones}
 inc(k,cab_aux.xmax);                                       {Incrementa cuenta de bloques}
 if (cab_aux.dir=2) Then
   seek(tem_arch,j*cab_aux.xmax)                            {Asigna la siguiente posición a partir del inicio}
 else
   seek(tem_arch,filesize(tem_arch)-k);                     {Asigna la siguiente posición a partir del final}
 gotoxy(28,1);write((j*100/cab_aux.ymax):3:0);              {Calcula el porcentaje de conversión}
End;

Procedure RLE;                                              {Procedimiento para comprimir con RLE}
 Begin
  inc(i);                                                   {incrementa cuenta de columnas}
  if cuantos=127 Then                                       {Si cuantos es igual a 127}
    cuantos:=(cuantos or 128)                               {Asigna bit de compresión}
  else
    cuantos:=(cuantos or 128)+1;                            {Asigna bits de compresión más 1}
  blockwrite(esc_arch,cuantos,1,escribio);                  {Escribe el número de bytes comprimidos}
```

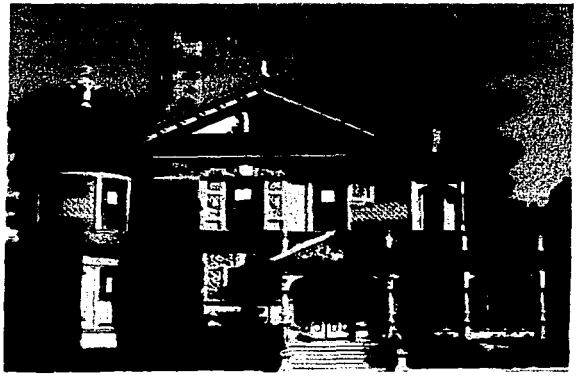
```
 blockwrite(esc_arch,byteant,1,escribio);                    {Escribe el valor de dicho byte}
 bytepen:=unbyte;                                  {El sig. bloque asigna nuevos valores e inicializ a}
 unbyte:=buf^[i+2];
 bandera:=0;
 cuantos:=0;
 sin_comp:=0;
End;

Procedure RAW;                                          {Porcedimiento para comprimir con RAW}
Var
 cont:integer;
Begin
 dec(sin_comp);                                                    {Decrementa sin comprimir}
 blockwrite(esc_arch,sin_comp,1,escribio);                  {Escribe cuantos bytes sin comprimir}
 for cont:=0 to sin_comp do
  Begin
   blockwrite(esc_arch,arr_bytes[cont],1,escribio);       {Escribe cada uno de los bytes sin comprimir}
  End;
 sin_comp:=0;                                                          {Inicializa variables}
 cuantos:=0;
 bandera:=0;
End;

Begin
 new(buf);
 Assign(tem_arch,'T_e_m_p!.SSS');          {El siguiente bloque abre el archivo temporal para su lectura}
   {Si-}
  reset(tem_arch,1);
  {Si+}
 if (ioresult=0) Then                                          {Si no hay errores continúa}
 Begin
  tam_imagen:=filesize(tem_arch)-786;                          {Calcula tamaño de imagen}
  Seek(tem_arch,filesize(tem_arch)-786);                {Se coloca al inicio de la cabecera auxiliar}
  blockRead(tem_arch,cab_aux,sizeof(cab_aux),sololee);        {Lee y almacena la cabecera auxiliar}
  nom_tga:='';i:=1;                                                   {Inicializa variables}
  While (cab_aux.nombre[i]<>chr(0)) do        {El siguiente bloque asigna x_nombre al archivo de escritura}
   Begin
    nom_tga:=concat(nom_tga,cab_aux.nombre[i]);
    inc(i);
   End;
  If Length(nom_tga)>=5 Then
   nom_tga:='X_'+copy(nom_tga,1,6)+'.tga'
  else
   nom_tga:='X_'+nom_tga+'.tga';
  Assign(esc_arch,nom_tga);                  {El siguiente bloque abre el archivo destino para su escritura}
  {Si-}
  Rewrite(esc_arch,1);
  {Si+}
 if (ioresult=0) Then                                          {Si no hay errores continúa}
Begin                                                  {El siguiente bloque asigna los datos de la cabecera}
   tga.identific:=0;                                                      {No hay datos incluidos}
   tga.color_map:=1;                                                      {Paleta RGB incluida}
   tga.tipo_tga:=9;                                                       {TGA tipo 9}
   tga.color_orig:=0;                                                     {Primer color}
```

```
  tga.color_Long:=256;                                                    {Último color}
  tga.color_tam:=24;                                                      {Tamaño de color}
  tga.xminimo:=0;                                                         {X mínima}
  tga.yminimo:=0;                                                         {Y mínima}
  tga.xmaximo:=cab_aux.xmax;                                              {X máxima}
  tga.ymaximo:=cab_aux.ymax;                                              {Y máxima}
  tga.nocolor:=8;                                                         {Bits por pixel}
  tga.describe:=0;                                                        {Tipo 2 en direccion de la imagen}
 End;
 blockwrite(esc_arch,tga,sizeof(tga),sololee);                            {Escribe la cabecera}
 for i:=0 to 255 do                                                       {Escribe paleta de colores azul,verde y rojo}
  Begin
   blockwrite(esc_arch,cab_aux.pal_rgb[(i*3)+3],1,sololee);               {Azul}
   blockwrite(esc_arch,cab_aux.pal_rgb[(i*3)+2],1,sololee);               {Verde}
   blockwrite(esc_arch,cab_aux.pal_rgb[(i*3)+1],1,sololee);               {Rojo}
 End;
 gotoxy(1,1);write('Porcentaje de conversión :     %');                  {Inicia método de compresión}
 k:=(786+tga.xmaximo);                                                    {Localiza la posición [11] de acuerdo con la dirección de la imagen}
 if (cab_aux.dir=2) Then
    Seek(tem_arch,0)                                                      {Se coloca al inicio del archivo}
 else
    Seek(tem_arch,filesize(tem_arch)-k);                                  {Se coloca al final del archivo menos la posiciòn anterior}
BlockRead(tem_arch,buf^,cab_aux.xmax,sololee);                            {Lee un bloque de datos de longitud xmàximo}
 i:=0;                                                                    {Inicializa variables}
 j:=0;
 cuantos:=0;
 sin_comp:=0;
 bandera:=0;
 Repeat                                                                   {Repite el siguiente bloque}
  Repeat                                                                  {Repite el siguiente bloque}
   byteant:=buf^[i];                                                      {Asigna byteant}
   bytepen:=buf^[i+1];                                                    {Asigna bytepen}
   unbyte:=buf^[i+2];                                                     {Asigna unbyte}
   If (byteant<>bytepen) or (bytepen<>unbyte) Then                        {Si algún byte es diferente}
    Begin
     if bandera=1 Then                                                    {Hay datos comprimidos}
       RLE                                                                {Escribe RLE}
     else                                                                 {Almacena datos sin comprimir}
      Begin
       arr_bytes[sin_comp]:=byteant;
       inc(sin_comp);
       if sin_comp>=127 Then                                              {Verifica no rebase 128}
        RAW                                                               {Escribe RAW}
       else
        Bandera:=2;                                                       {Asigna bandera sin comprimir}
      End;
    End
   Else                                                                   {Hay bytes iguales}
    Begin                                                                 {Cuenta 1 byte compreso}
     if (bandera=2)  Then                                                 {Revisa si se envian bytes sin comprimir}
        RAW;                                                              {Escribe RAW }
```

[11] Anteriormente la dirección de la imagen se ha designado como 2 (dos) ya que las imágenes TGA se despliegan de la esquina superior izquierda a la esquina inferior derecha.

```
        inc(cuantos);                                          {Incrementa cuantos}
        bandera:=1;                                            {Asigna bandera bytes comprimidos}
        if cuantos>=127 Then                                   {Verifica no rebase 128}
          RLE;                                                 {Escribe RLE}
      End;
      inc(i);                                                  {Incrementa cuenta de columnas}
    Until (i>=cab_aux.xmax);                                   {Hasta terminar dicho renglón}
    if bandera=2 Then RAW;                                     {Revisa si quedo un conteo sin terminar}
    dec(cuantos,2);                                            {Decrementa cuantos}
    if bandera=1 Then RLE;                                     {Escribe los últimos bytes}
    cuantos:=0;                                                {inicializa varables}
    Bandera:=0;
    i:=0;
    pos_sig_bloque(j);                                         {Calcula posición del sig. bloque}
  BlockRead(tem_arch,buf^,cab_aux.xmax,sololee);               {Lee otro bloque de datos}
    Until (j>=cab_aux.ymax);                                   {Hasta terminar todos los renglones}
   End;
 dispose(buf);
End;
```

Una de las imágenes procesadas con el programa es com24cas.tga



COM24CAS.TGA

El programa da como resultado los siguientes datos:

```
Nombre de Archivo  : com24cas.tga
Identificación  : 0   Posición del puntero  : 18
Mapa de color   : 0
Tipo de tga     : 10
Color de origen : 0
Color longitud  : 0
Tamaño de pixel : 0
Xminimo         : 0
Yminimo         : 0
Xmaximo         : 512
Ymaximo         : 480
Bits por pixel  : 24
Descriptor      : 0_
```

Espere un momento en el cual se descomprime la imagen y se crea el temporal enseguida aparece:

```
Porcentaje de conversión : 100 %
La conversión se completo con éxito
```

V.8.2 En Lenguaje C

```
BYTE cuantos,                                   // Contador de datos comprimidos
       unbyte,                                  // Byte último
        sin_comp,                               // Sin comprimir
         byteant,                               // Byte antepenúltimo
          bytepen,                              // Byte penúltimo
           bandera;                             // Bandera tipo de compresión
BYTE arr_bytes[260];                            // Arreglo de bytes comprimidos
BYTE buf[4096];                                 // Arreglo de datos

void Esc_imagen_TGA()                           // Realiza la conversión al formato TGA
{
 char *nom_tga;                                 // Nombre de archivo
 char *siglas=".TGA";                           // Extensión del formato

 void pos_sig_bloque();                         // Calcula posición del siguiente bloque
 void RLE();                                    // Escribe RLE
 void RAW();                                    //Escribe RAW

 tem_arch=fopen("T_E_M_P!.SSS","rb");           // Abre archivo temporal para su lectura
  if (tem_arch!=NULL)                           // Si no hay errores continúa
  {
   tam_imagen=filelength(open("T_E_M_P!.SSS",O_RDWR));   // Calcula tamaño de la imagen
   fseek(tem_arch,tam_imagen-786,SEEK_SET);     // Se coloca al inicio de la cabecera
   fread((unsigned char *)&cab_aux,1,sizeof(cab_aux),tem_arch);   // Lee y almacena la cabecera auxiliar
                                                // El siguiente bloque asigna X_nombre al archivo de escritura
   for (i=0;i<=11;i++)
     nom_tga[i]=0;
   nom_tga[0]='x';
   nom_tga[1]='_';
   i=0;
   do
    {
     nom_tga[i+2]=cab_aux.nombre[i];
     i++;
    }while (cab_aux.nombre[i]!=0);
    strcat(nom_tga,siglas);
    esc_arch=fopen(nom_tga,"wb");               Abre el archivo destino para su escritura
    if (esc_arch!=NULL)                         Si no hay errores continúa
    {                                           El siguiente bloque asigna los datos de la cabecera
     tga.identific=0;                           No hay datos incluidos
     tga.color_map=1;                           Paleta RGB incluida
     tga.tipo_tga=9;                            TGA tipo 9
     tga.color_orig=0;                          Primer color
     tga.color_long=256;                        Último color
     tga.color_tam=24;                          Tamaño de color
     tga.xminimo=0;                             X mínima
     tga.yminimo=0;                             Y mínima
     tga.xmaximo=cab_aux.xmax;                  X máxima
     tga.ymaximo=cab_aux.ymax;                  Y máxima
     tga.nocolor=8;                             Bits por pixel
     tga.describe=0;                            Tipo 2 en direccion de la imagen
    }
```

```
 fwrite((unsigned char *)&tga,1,sizeof(tga),esc_arch);                    // Escribe la cabecera
                                                                  // Escribe la paleta de colores azul,verde y rojo
 for (i=0;i<=255;i++)
  {
   fwrite((unsigned char *)&cab_aux.pal_rgb[(i*3)+2],1,1,esc_arch);                    // Azul
   fwrite((unsigned char *)&cab_aux.pal_rgb[(i*3)+1],1,1,esc_arch);                    // Verde
   fwrite((unsigned char *)&cab_aux.pal_rgb[(i*3)+0],1,1,esc_arch);                    // Rojo
  }
gotoxy(1,1);printf("Porcentaje de conversión :    \%");                    // Inicia método de compresión
k=(786+tga.xmaximo);                          // Localiza la posición de acuerdo con la dirección de la imagen
if (cab_aux.dir==2)
  fseek(tem_arch,0,SEEK_SET);                                             // Se coloca al inicio del archivo
else
  fseek(tem_arch,tam_imagen-k,SEEK_SET);                  // Se coloca al final del archivo menos la posición
fread((unsigned char *)&buf,cab_aux.xmax,1,tem_arch);     // Lee un bloque de datos de longitud xmáximo
i=0;                                                                                  // Inicializa variables
j=0;
cuantos=0;
sin_comp=0;
bandera=0;
do                                                                                                    // Haz
  {
    do                                                                                                // Haz
       {
        byteant=buf[i];                                                                   // Asigna byteant
        bytepen=buf[i+1];                                                                 // Asigna bytepen
        unbyte=buf[i+2];                                                                  // Asigna unbyte
        if ((byteant!=bytepen) || (bytepen!=unbyte))                              // Si algún byte es diferente
         {
          if (bandera==1)                                                         // Hay datos comprimidos
            RLE();                                                                                // Escribe RLE
          else                                                                   // Almacena datos sin comprimir
           {
            arr_bytes[sin_comp]=byteant;
            sin_comp++;
            if (sin_comp>=127)                                                    // Verifica no rebase 128
             RAW();                                                                               // Escribe RAW
            else
             bandera=2;                                                          // Asigna bandera sin comprimir
           }
         }
        else                                                                                // Hay bytes iguales
         {                                                                                 // Cuenta 1 byte compreso
          if (bandera==2)                                                  // Revisa si se envian bytes sin comprimir
            RAW();                                                                                // Escribe RAW
          cuantos++;                                                                      // Incrementa cuantos
          bandera=1;                                                           // Asigna bandera datos comprimidos
          if (cuantos>=127)                                                                Verifica no rebase 128
            RLE();                                                                                  Escribe  RLE
         }
        i++;                                                                              Incrementa cuenta de columnas
    }while (i<cab_aux.xmax);                                                          Mientras no termine dicho renglón
    if (bandera==2)                                                                 Revisa si quedo un conteo sin terminar
       RAW();                                                                                             // Escribe RAW
```

```
    cuantos=cuantos-2;                                          // Decrementa cuantos
    if (bandera==1) RLE();                                      // Escribe los últimos bytes
    cuantos=0;                                                  // Inicializa variables
    bandera=0;
    i=0;
    pos_sig_bloque();                                           // Calcula posición del siguiente bloque
    fread((unsigned char *)&buf,cab_aux.xmax,1,tem_arch);       // Lee otro bloque de datos
   } while (j<cab_aux.ymax);                                    // Mientras no termine todos los renglones
  }
}


void pos_sig_bloque()                                           // Determina la posición del siguiente bloque de datos
{
 j++;                                                           // Incrementa cuenta de renglones
 k=k+cab_aux.xmax;                                              // Incrementa cuenta de bloques
 if (cab_aux.dir==2)
  fseek(tem_arch,j*cab_aux.xmax,SEEK_SET);                      // Asigna la posición  a partir del inicio
 else
  fseek(tem_arch,tam_imagen-k,SEEK_SET);                        // Asigna la posición a paritr del final
 gotoxy(28,1);printf("%3d",(j*100/cab_aux.ymax));              // Calcula porcentaje de compresión
}


void RLE()                                                      // Compresión con RLE
{
 i++;                                                           // Incrementa cuenta de columnas
 if (cuantos==127)                                              // Si cuantos es igual a 127
  cuantos=(cuantos | 128);                                      // Asigna bit de compresión
 else
  cuantos=(cuantos | 128)+1;                                    // Asigna bit de compresión más 1
 fwrite((unsigned char *)&cuantos,1,1,esc_arch);                // Escribe el número de bytes comprimidos
 fwrite((unsigned char *)&byteant,1,1,esc_arch);                // Escribe el valor de dicho byte
 bytepen=unbyte;                                                // El sig. bloque asigna nuevos valores e inicializa
 unbyte=buf[i+2];
 bandera=0;
 cuantos=0;
 sin_comp=0;
}


void RAW()                                                      // Compresión con RAW
{
 int cont;
 sin_comp--;                                                    // Decrementa sin comprimir
 fwrite((unsigned char *)&sin_comp,1,1,esc_arch);               // Escribe cuantos bytes sin comprimir
 for (cont=0;cont<=sin_comp;cont++)                             // Escribe cada uno de los bytes sin comprimir
  fwrite((unsigned char *)&arr_bytes[cont],1,1,esc_arch);
 sin_comp=0;                                                    // Inicializa variabes
 cuantos=0;
 bandera=0;
}
```

Una de las imágenes procesadas con el programa es pumaunco.tga.

PUMAUNCO.TGA

El programa da como resultado los siguientes datos:

```
Nombre de Archivo : pumaunco.tga
Identificación : 0   Posición del puntero : 18
Mapa de color  : 0
Tipo de tga    : 2
Color de origen: 0
Color longitud : 0
Tamaño de pixel: 0
Xminimo        : 0
Yminimo        : 0
Xmaximo        : 384
Ymaximo        : 256
Bits por pixel : 24
Descriptor     : 0
```

Espere un momento en el cual se descomprime la imagen y se crea el temporal enseguida aparece:

```
Porcentaje de conversión : 100 %
La conversión se completo con éxito
```

CONCLUSIONS

CONCLUSIONES

Actualmente las imágenes gráficas tienen un gran avance en nuestra sociedad, temas como la realidad virtual, fractales y en campos como el cine, video y televisión se están basando en el manejo de las imágenes. Con los conocimientos planteados en el presente trabajo, el lector se puede introducir al ambiente gráfico.

Manipular una imagen no solo es desplegarla en pantalla, es también, crear, convertir, animar y jugar con sus colores para representar movimientos. Pero para lograr todo esto es necesario conocer todo los elementos que intervienen o pueden intervenir en un ambiente gráfico.

Tratamos de hacer una programación que fuera fácil de comprender, no quisimos utilizar funciones que fueran complicadas (manejo de memoria extendida por ejemplo), para que el lector sobre todo si es principiante sea capaz de crear su propia rutina para llamar una imagen. La programación es estándar, una función es independiente de las demás y puede ser llamada una a la vez, esto hace que el lector logre entender los conceptos mejor y probablemente hasta mejore los programas, esto desde nuestro punto de vista

Los objetivos marcados al inicio de la tesis se han cumplido, sin embargo, existen muchos otros formatos gráficos que actualmente están creciendo mucho, como el PNG (que algunos lo llaman el sustituto de GIF), CDR (de Corel Draw), 3DS (de 3D Studio), WMF (de Windows Metafile), estos dos últimos sin ser bitmap, por mencionar algunos. Consideramos que el gran ausente en el presente trabajo es JPG, ya que debido a la falta de

información (principalmente en el método de compresión) no fue posible incluirlo, pero aun así el trabajo no carece de conceptos realmente buenos y que junto con el lector (sea principiante o avanzado en la programación gráfica) se logren incrementar los conocimientos necesarios para desglosar cualquier otro formato, ¡ y pronto tener a JPG en nuestras manos...!.

GLOSARIO

GLOSARIO

and

Y lógico.

ansi (american national standards institute)

Coordina el desarrollo de estándares voluntarios a nivel nacional, que incluyen lenguajes de programación, EDI, telecomunicaciones y propiedades de medios de disco y cinta.

api (application program interface)

Interface de programa de aplicación. Lenguaje y formato de mensaje utilizados por un programa para activar e interactuar con las funciones de otro programa o de hardware.

ascii (american standard code for information interchange)

Código estándar de los E.U. para intercambio de información. Código binario de datos que se usa en comunicaciones, en las computadoras. Solo los primeros 128 caracteres (0-127) dentro de las 256 combinaciones de un byte constituyen el estándar ASCII. El resto se utiliza en forma diferente de acuerdo a la computadora.

asociación sig (special interest group)

Grupo de Interés Especial. Grupo de personas que se reúnen y comparten información acerca de determinado tema de interés. Usualmente es parte de un grupo ó Asociación mayor.

aspect ratio

La medida proporcional de una imagen o pixel basado en su tamaño horizontal y vertical. Por ejemplo, una imagen un aspect ratio de 4:3 tiene un tamaño horizontal que es 4/3 del tamaño vertical.

at

Advanced Technology. Tecnología avanzada. Primer computadora personal de IBM basada en el 80286, introducido en 1984. Era la máquina más avanzada de la línea de PC's e incluía un nuevo teclado, drive de 1.2 Mb. y bus de datos de 16 bits. Las máquinas del tipo AT funcionan mucho más rápido que las XT (PC basadas en el 8088.

backgroud

Color de base ó fondo, segundo plano de la pantalla.

basic

Beginners All purpose Symbolic Instruction Code

bbs

Bulletin Board System. Un programa de telecomunicaciones que permite ejecutándose en una computadora que permite a otras computadoras con módem comunicarse y accesar a archivos. Son la fuente principal de archivos de imágenes e información. Nombres más antiguos de BBS son CBBS (Computer Bulletin Board System) y EBBS (Electronic Bulletin Board System).

big-endian

Se refiere a los sistemas o máquinas que almacenan al byte más significante (MSB) en la posición más baja en una palabra (word), generalmente referido como el byte 0. Contrasta con little endian.

bit depth

El valor de tamaño usado para representar un pixel en un bitmap. Esta dado como el número de bits (ó número de bytes) que forman un valor de dato individual. El número 2 es la base para las potencias de bit depth especificadas como el número máximo de valores de pixel que puede asumir. También es conocido como pixel depth.

bitblt (bit block transfer)

Transferencia de bloques de bits. En aceleradores y máquinas gráficas, característica del hardware que mueve el bloque rectangular de bits de la memoria principal a la memoria de video. Acelera la visualización de objetos en movimiento (animación, desplazamiento) en pantalla.

bitmap

Un juego de valores numéricos que especifican los colores de pixel en un dispositivo de salida. Anteriormente, el término era empleado para referirse a los datos que se deseaban desplegar en los dispositivos de salida capaces de desplegar color en dos niveles. Se usa como un sinónimo de raster.

borland int'l, inc.

Lenguajes de programación (Turbo C, Turbo Pascal), Dbase, Paradox, Quattro Pro, Sidekick. Scotts Valley, CA.

bus

Canal o ruta común entre dispositivos del hardware. Un bus conecta el CPU con la memoria principal y a los bancos de memoria que residen en las unidades de control de mecanismos periféricos. Está compuesto de dos partes. Las direcciones se envían sobre el bus de direcciones para señalar una locación de memoria y los datos se transfieren sobre el bus de datos a ésta.

bus local

En una PC, canal de datos que va de l CPU a los periféricos, que corre a la velocidad más alta del reloj del CPU, en lugar de las velocidades más bajas de los bus de ISA, EISA y MCA. Las primeras implementaciones utilizaban diseños patentados; sin embargo, VESA estandarizó el bus VL, e Intel lanzó su especificación PCI en 1993.

CONCLUSIONES

Actualmente las imágenes gráficas tienen un gran avance en nuestra sociedad, temas como la realidad virtual, fractales y en campos como el cine, video y televisión se están basando en el manejo de las imágenes. Con los conocimientos planteados en el presente trabajo, el lector se puede introducir al ambiente gráfico.

Manipular una imagen no solo es desplegarla en pantalla, es también, crear, convertir, animar y jugar con sus colores para representar movimientos. Pero para lograr todo esto es necesario conocer todo los elementos que intervienen o pueden intervenir en un ambiente gráfico.

Tratamos de hacer una programación que fuera fácil de comprender, no quisimos utilizar funciones que fueran complicadas (manejo de memoria extendida por ejemplo), para que el lector sobre todo si es principiante sea capaz de crear su propia rutina para llamar una imagen. La programación es estándar, una función es independiente de las demás y puede ser llamada una a la vez, esto hace que el lector logre entender los conceptos mejor y probablemente hasta mejore los programas, esto desde nuestro punto de vista.

Los objetivos marcados al inicio de la tesis se han cumplido, sin embargo, existen muchos otros formatos gráficos que actualmente están creciendo mucho, como el PNG (que algunos lo llaman el sustituto de GIF), CDR (de Corel Draw), 3DS (de 3D Studio), WMF (de Windows Metafile), estos dos últimos sin ser bitmap, por mencionar algunos. Consideramos que el gran ausente en el presente trabajo es JPG, ya que debido a la falta de

información (principalmente en el método de compresión) no fue posible incluirlo, pero aun así el trabajo no carece de conceptos realmente buenos y que junto con el lector (sea principiante o avanzado en la programación gráfica) se logren incrementar los conocimientos necesarios para desglosar cualquier otro formato, ¡ y pronto tener a JPG en nuestras manos...!.

GLOSARIO

GLOSARIO

and
Y lógico.

ansi (american national standards institute)
Coordina el desarrollo de estándares voluntarios a nivel nacional, que incluyen lenguajes de programación, EDI, telecomunicaciones y propiedades de medios de disco y cinta.

api (application program interface)
Interface de programa de aplicación. Lenguaje y formato de mensaje utilizados por un programa para activar e interactuar con las funciones de otro programa o de hardware.

ascii (american standard code for information interchange)
Código estándar de los E.U. para intercambio de información. Código binario de datos que se usa en comunicaciones, en las computadoras. Solo los primeros 128 caracteres (0-127) dentro de las 256 combinaciones de un byte constituyen el estándar ASCII. El resto se utiliza en forma diferente de acuerdo a la computadora.

asociación sig (special interest group)
Grupo de Interés Especial. Grupo de personas que se reúnen y comparten información acerca de determinado tema de interés. Usualmente es parte de un grupo ó Asociación mayor.

aspect ratio
La medida proporcional de una imagen o pixel basado en su tamaño horizontal y vertical. Por ejemplo, una imagen un aspect ratio de 4:3 tiene un tamaño horizontal que es 4/3 del tamaño vertical.

at
Advanced Technology. Tecnología avanzada. Primer computadora personal de IBM basada en el 80286, introducido en 1984. Era la máquina más avanzada de la línea de PC's e incluía un nuevo teclado, drive de 1.2 Mb. y bus de datos de 16 bits. Las máquinas del tipo AT funcionan mucho más rápido que las XT (PC basadas en el 8088.

backgroud
Color de base ó fondo, segundo plano de la pantalla.

basic
Beginners All purpose Symbolic Instruction Code

bbs

Bulletin Board System. Un programa de telecomunicaciones que permite ejecutándose en una computadora que permite a otras computadoras con módem comunicarse y accesar a archivos. Son la fuente principal de archivos de imágenes e información. Nombres más antiguos de BBS son CBBS (Computer Bulletin Board System) y EBBS (Electronic Bulletin Board System).

big-endian

Se refiere a los sistemas o máquinas que almacenan al byte más significante (MSB) en la posición más baja en una palabra (word), generalmente referido como el byte 0. Contrasta con little endian.

bit depth

El valor de tamaño usado para representar un pixel en un bitmap. Esta dado como el número de bits (ó número de bytes) que forman un valor de dato individual. El número 2 es la base para las potencias de bit depth especificadas como el número máximo de valores de pixel que puede asumir. También es conocido como pixel depth.

bitblt (bit block transfer)

Transferencia de bloques de bits. En aceleradores y máquinas gráficas, característica del hardware que mueve el bloque rectangular de bits de la memoria principal a la memoria de video. Acelera la visualización de objetos en movimiento (animación, desplazamiento) en pantalla.

bitmap

Un juego de valores numéricos que especifican los colores de pixel en un dispositivo de salida. Anteriormente, el término era empleado para referirse a los datos que se deseaban desplegar en los dispositivos de salida capaces de desplegar color en dos niveles. Se usa como un sinónimo de raster.

borland int'l, inc.

Lenguajes de programación (Turbo C, Turbo Pascal), Dbase, Paradox, Quattro Pro, Sidekick. Scotts Valley, CA.

bus

Canal o ruta común entre dispositivos del hardware. Un bus conecta el CPU con la memoria principal y a los bancos de memoria que residen en las unidades de control de mecanismos periféricos. Está compuesto de dos partes. Las direcciones se envían sobre el bus de direcciones para señalar una locación de memoria y los datos se transfieren sobre el bus de datos a ésta.

bus local

En una PC, canal de datos que va de l CPU a los periféricos, que corre a la velocidad más alta del reloj del CPU, en lugar de las velocidades más bajas de los bus de ISA, EISA y MCA. Las primeras implementaciones utilizaban diseños patentados; sin embargo, VESA estandarizó el bus VL, e Intel lanzó su especificación PCI en 1993.

bus de datos

Trayecto interno mediante el cual los datos se transfieren hacia y desde el CPU. Las ranuras de expansión en las computadoras están conectadas al bus de datos.

bus mastering

Dominación del bus. Diseño del bus que permite que las tarjetas adicionales procesen independiente del CPU, y sean capaces de acceder a la memoria de la computadora y sus periféricos por su cuenta.

búsqueda booleana

Búsqueda de datos específicos. Implica que cualquier condición puede buscarse utilizando los operadores booleanos AND, OR, NOT.

caché

Sección reservada de memoria que se utiliza para mejorar el rendimiento. Un caché de disco es una sección reservada de la memoria normal, o memoria adicional en la tarjeta controladora del disco. Cuando el disco es leído, se copia un gran bloque de datos en el caché. Si las solicitudes de datos subsiguientes pueden ser satisfechas por el caché, no se necesita la utilización de un acceso de disco que es más lento. Si el caché es utilizado para escritura, los datos se alinean en la memoria y se graban en el disco en bloques más grandes.
La memoria caché son bancos de memoria de alta velocidad entre la memoria y el CPU. Los bloques de instrucciones y datos se copian en el caché, y la ejecución de las instrucciones y la actualización son llevados a cabo en la memoria de alta velocidad.

ccitt

Comité consultivo Internacional de Telégrafo y Teléfono.

cmos

Memoria hecha de chips CMOS. Banco de memoria respaldado por baterías en computadoras que se utiliza para mantener la hora, fecha e información de sistemas como tipo de unidades.

compilador

Software que traduce lenguajes de programación de alto nivel (COBOL, C, etc.) a lenguaje de máquina. Un compilador habitualmente genera en primer lugar un lenguaje ensamblador y a continuación traduce este último a uno de máquina.
Software que convierte un lenguaje de alto nivel en una representación de nivel más bajo. por ejemplo, un compilador se ayuda convierte un documento de texto en comandos apropiados a un sistema de ayuda en línea. Un compilador de tipo diccionario convierte términos y definiciones en un sistema de diccionario de búsqueda.

cluster

Racimo, grupo, conglomerado, agrupamiento. Cantidad de sectores de disco (por lo general de 2 a 16) tratados como unidad. Todo el disco se divide en sectores cluster, cada uno con un incremento mínimo de almacenamiento. Por consiguiente un archivo de 30 bytes puede ocupar hasta 2,048 bytes en disco si el cluster es de cuatro sectores de 512 bytes.

crt (cathode ray tube)

Tubo de rayos catódicos. Tubo de vacío utilizado como pantalla de representación en una terminal de video o TV. El término se utiliza con frecuencia para referirse a la terminal misma.

cursor

Símbolo móvil en una pantalla que sirve como punto de contacto entre el usuario y los datos. En los sistemas basados en texto, el cursor es un rectángulo o símbolo titilante, y se mueve mediante la activación del mouse o de las teclas Inicio, Fin, Repág, Avpág y las cuatro teclas marcadas con las flechas. En los sistemas gráficos, éste se denomina puntero y puede adoptar cualquier forma (flecha, cuadrado, pincel, etc.) y habitualmente cambia de forma cuando se desplaza a diferentes zonas de la pantalla.

dac (digital to analog converter, d/a)

Convertidor digital analógico. Dispositivo que convierte pulsaciones digitales en señales analógicas.

database

Conjunto de ficheros o registros relacionados lógicamente. Una base de datos comprende muchos registros que pueden haber sido almacenados previamente en ficheros separados, de forma que una agrupación común de registros de datos sirve como fichero central para diversas aplicaciones de procesamiento de datos.

dct

Discrete Cosine Transform. Transformación matemática empleada para convertir datos de 3D a formas de 2D. Utilizado por algunos métodos de compresión como JPEG y MPEG.

decimation

El proceso de sacar fuera porciones de una imagen bitmap cuando se reduce su tamaño.

dram (d-ram, dynamic ram)

RAM Dinámica. Tipo más común de memoria en computadoras. Generalmente utiliza un transistor y un condensador para representar un bit. Los condensadores deben ser energetizados cientos de veces por segundo para mantener sus cargas. A diferencia de los chips de firmware (ROM,PROM, etc.), las dos principales variedades de RAM (dinámica y estática) pierden su contenido cuando se corta el suministro de energía.

En la publicidad de memoria, RAM dinámica con frecuencia se menciona erróneamente como un tipo de paquete: por ejemplo: "DRAM, SIMM y SIP para la venta". Debería ser "DIP, SIMM y SIP" como tres paquetes que por lo general incluyen chips de RAM dinámica.

drawing surface

El área en un dispositivo de salida donde aparece la representación de una imagen.

driver

Controlador. Rutina de programa que conecta un dispositivo periférico o una función interna al sistema operativo. Contiene el lenguaje de máquina necesario para activar todas las

funciones del dispositivo e incluye la información detallada de sus características, como sectores por pista o la cantidad de pixeles de la resolución de la pantalla.

eeprom (electrically erasable progammable read only memory)
Memoria programable y borrable eléctricamente de solo lectura. Chip de memoria que retiene su contenido sin energía. Puede ser borrado, tanto dentro de la computadora como en el exterior de ella, y usualmente requiere más voltaje para el borrado que el común de +5v, utilizado en los circuitos lógicos.

eisa
Extenden ISA. Arquitectura estándar industrial extendida. Estándar de bus para PC que extiende el bus AT (bus ISA) a 32 bits y permite el control del bus. EISA fue anunciado en 1988 como una alternativa de 32 bits a MCA que preservaría la inversión en las tarjetas existentes. Las tarjetas de PC y AT (tarjetas ISA) pueden enchufarse a una ranura EISA.

encriptación
Cifrado, criptografiado, criptografiacción. Codificación de datos con propósito de seguridad, convirtiendo el código estándar de datos en un código propio.

eprom (erasable progammable read only memory)
Memoria programable y borrable de solo lectura. Chip PROM reutilizable que conserva su contenido hasta ser borrado bajo luz ultravioleta.

estados de espera (wait states)
Cantidad de tiempo empleado para esperar que se ejecute una operación. Puede referirse a una longitud variable de tiempo que debe esperar un programa antes de procesarse, o referirse a una duración específica de tiempo, como un ciclo de máquina.
Cuando la memoria es demasiado lenta para responder a la petición del CPU, se introducen estados de espera hasta que pueda alcanzarlo la memoria.

foregrund
Color de primer plano en la pantalla.

frame
Una imagen sencilla. Múltiples frames de imágenes ligeramente diferentes desplegadas en una secuencia rápida son utilizadas para crear animaciones.

frame buffer
Término antiguo para las tarjetas de video. Técnicamente es la porción de una tarjeta de video que contiene la memoria donde los datos de imagen son reunidos antes de ser enviados al monitor.

gui (graphical user interface)
Sistema Operativo que proporciona instrucciones indicándole al ordenar cómo aceptar sus entradas, generar las salidas y procesar datos. En lugar de presentar mensajes basados en texto, el interface proporciona ventanas, menús e iconos gráficos que representan comandos.

graphics file
Un archivo que contiene datos gráficos.

graphics file format
La definición de, y convenciones asociadas con, una estructura de archivo utilizada para el almacenamiento de datos gráficos.

graymap
Datos de rastreo compuestos de valores con no más de 2 niveles, empleados para ser desplegados en un dispositivo de salida capaz de desplegar solo sombras de grises.

hardware
Maquinaria y equipo (CPU, discos, cintas, módem, cables, etc.). En una operación, la computadora es tanto el hardware como el software. El uno no sirve sin el otro. El diseño del hardware especifica los comandos que puede seguir y las instrucciones que le dicen que debe hacer.

hybrid database
La habilidad de almacenar la información de bases de datos complejas en conjunción con datos gráficos.

hybrid text
El almacenamiento y despliegue de bitmap y datos de texto usando un formato de archivo simple. GIF89a es un ejemplo de un formato con una capacidad de hybrid text.

key points
Punto necesario para la reconstrucción de un objeto gráfico desde un vector. Usualmente son el mínimo requerido para especificar al objeto.

internet
Red extensa compuesta por una gran cantidad de redes más pequeñas.
Red orientada a la investigación que comprende más de 3,00 redes gubernamentales y académicas en más de 40 países.

interprete
Traductor de lenguajes de programación de alto nivel que traduce y ejecuta el programa al mismo tiempo. Traduce una sentencia de programa a lenguaje de máquina, la ejecuta y luego pasa a la sentencia siguiente.

isa
Industry Standard Architecture. Arquitectura industrial estándar. Se refiere a la arquitectura original del bus para PC, específicamente el bus AT de 16 bits.

jumper
Puente. Puente de metal que se usa para cerrar un circuito. Puede ser un tramo corto de alambre ó un bloque de metal cubierto de plástico que se presione sobre dos pernos (pins) en una tarjeta de circuito. A menudo se utiliza en lugar de conmutadores DIP.

lenguaje de alto nivel
Permites que el problema se exprese en un nivel más alto que el de la máquina. Se llaman lenguajes compiladores, y pueden compilarse (traducirse) al lenguaje de la máquina para una variedad de diferentes familias de computadoras.

lenguaje de bajo nivel
Existe un lenguaje ensamblador, o lenguaje de bajo nivel, para cada tipo de máquina, que usualmente genera una instrucción de máquina para cada instrucción del lenguaje ensamblador. Con frecuencia, los lenguajes ensambladores son muy diferentes y difíciles de convertir de uno a otro.

lenguajes de programación
Lenguaje que se utiliza para escribir instrucciones para la computadora. Permite que el programador exprese el procesamiento de datos en forma simbólica sin tener encuentra los detalles específicos de la máquina.

little-endian
Se refiere a los sistemas o máquinas que almacenan al byte menos significante (LSB) en la posición más baja en una palabra (word), generalmente referido como el byte 0. Contrasta con big-endian.

look up table
Una serie de parejas de valores numéricos con los cuales un programa puede igualar un valor significativo a uno que especifica un color en un dispositivo de salida.

mca (micro channel architecture)
Arquitectura de microcanal. Bus de 32 bits de IBM utilizado en la mayor parte de las PS/2, la serie RS/6000 y ciertos modelos de ES/9370. Las tarjetas MCA pueden diseñarse para el dominio del bus y también contienen una identificación incorporada que elimina las fijaciones manuales que, a menudo, se requieren en las tarjetas ISA. Las tarjetas MCA no son intercambiables con las tarjetas ISA ó EISA.

memoria convencional
En una computadora es el primer megabyte de memoria. El término también puede referirse solo a los primeros 640Kb. Los últimos 384Kb. Del primer megabyte se denominan memoria alta del DOS ó área de memoria más alta.

memoria expandida
Memoria más allá de los 32Mb. en una computadora.

memoria extendida
En las computadoras 286 o superiores, es la memoria por encima de un megabyte.

metafile
Un formato de archivo capaz de almacenar dos o más tipos de datos, generalmente vector y bitmap, en el mismo archivo.

microsoft corp.
DOS, Windows, lenguajes de programación, aplicaciones. Fundada en 1975 por Bill Gates y Paul Allen. Redmon, WA.

mnemotécnicos
Significa una ayuda a memoria. Nombre asignado a una función de máquina. Por ejemplo, en DOS, COM1 es el mnemotécnico asignado al puerto serial 1. Los lenguajes de programación son casi totalmente mnemotécnicos.

modo burst (burst mode)
Modo de estallido. Método alternativo para transmisión a alta velocidad en un canal de comunicaciones o de computadora. Bajo ciertas condiciones, el sistema puede enviar un "estallido" de datos a mayor velocidad por cierto periodo. Por ejemplo, un canal multiplexor puede suspender la transmisión de varias corrientes de datos y enviar transmisión de datos a alta velocidad utilizando todo el ancho de banda.

multitarea
Ejecución de dos o más programas en una computadora al mismo tiempo. La tarea múltiple se controla mediante el sistema operativo. La cantidad de programas que pueden realizar tareas múltiples efectivamente depende de la cantidad de memoria disponible, la velocidad del CPU, la capacidad y velocidad del disco duro, así como la eficiencia del sistema operativo.

oem (original equipment manufacturer)
Fabricante de equipo original. Fabricante que vende equipo a un distribuidor. El término también se refiere al distribuidor en sí. Los clientes de un OEM agregan valor al producto antes de revenderlo, le colocan etiquetas con sus propias marcas o lo venden en paquete junto con sus propios productos.

or
or lógico.

palabra
Word. Unidad interna de almacenamiento de la computadora. Se refiere a la cantidad de datos que puede contener en sus registros. Por ejemplo, a la misma velocidad del reloj, una computadora de 16 bits procesa dos bytes en el mismo tiempo que en una computadora de 8 bits procesa 1 byte.

padding
Porción de un archivo generalmente incluida para adecuar las dependencias de las computadoras o para incrementar la velocidad de lectura y/ó escritura.

parse
Análisis gramatical. Analizar una sentencia o instrucción de lenguaje. El análisis gramatical descompone las palabras en unidades funcionales que puede convertirse a lenguaje de máquina.

pels (picture element)
Vea pixel.

pixel
PIX (Picture) ELement. Elemento más pequeño en pantalla. Una pantalla se divide en miles de diminutos puntos, y un pixel es uno ó más puntos que se tratan como una unidad. Un pixel puede ser un punto en una pantalla monocromática, tres puntos (rojo, verde y azul) en pantallas de color,. ó una agrupación de tales puntos.

pixel depth
Vea bit depth.

pixel map
Un dato bitmap compuesto de valores con no más de 2 niveles, empleados para ser desplegados en un dispositivo de salida capaz de desplegar color.

pixmap
Vea pixel map.

plotter
La representación de un bitmap de vector o datos gráficos 3D utilizado para desplegar una aproximación de los datos gráficos. Vea thumbnail.

poo
Object Oriented Programming (OOP). Programación orientada a objetos, tecnología de programación que es más flexible que la estándar. Es una forma evolutiva de programación modular con reglas formales que permite con mayor facilidad que segmentos de software sean reutilizados e intercambiados entre diversos programas. Trata con módulos autónomos, u objetos, que contienen tanto los datos como las rutinas que actúan en éstos. Esto se denomina encapsulamiento y estos tipos de datos definidos por el usuario se llaman clase. Un ejemplo de clase es el objeto. Otra característica importante es la herencia. Las clases se crean en jerarquías que permiten que el conocimiento de una clase se pase a la jerarquía.
El Smalltalk de Xerox fue el primer lenguaje orientado a objetos. Actualmente, C++ es el lenguaje orientado a objetos más popular porque es una ampliación del C tradicional.

production pipeline
Serie de operaciones envueltas en la definición , creación y despliegue de una imagen, desde la concepción hasta su realización o grabación en un dispositivo de salida.

programa
Conjunto de instrucciones que indican qué debe hacer la computadora. Un programa se denomina software; por lo tanto, programa, software e instrucciones son sinónimos.

prom (programmable read only memory)
Memoria programable de solo lectura. Chip de memoria permanente que es programado ó llenado, por el cliente, en lugar del fabricante de chips. Observe la diferencia con ROM, que es programada en el momento de su fabricación.

ram (random acces memory)

Memoria de acceso aleatorio. Memoria de tipo "aleatorio" que significa que puede tener acceso directamente a los contenidos de cada byte sin hacer referencia a los bytes anteriores o posteriores a éste. Los chips de memoria RAM necesitan energía para mantener su contenido.

raw

Datos de imagen sin información de cabecera. En ocasiones referido como datos de imagen, especialmente en los bitmap, que están sin comprimir ó codificar.

raster

Se refiere a datos gráficos representados por valores de color en puntos, que se toman juntos para su despliegue en un dispositivo de salida. Actualmente se le conoce como bitmap.

rendering

La representación actual de una imagen de datos gráficos en un dispositivo de salida.

rom (read only memory)

Memoria de solo lectura. Chip de memoria que almacena en forma permanente instrucciones y datos. Su contenido se crea en el momento de la fabricación y no puede alterarse. SE utilizan ampliamente para almacenar rutinas de control en computadoras (ROM BIOS) y en controladores periféricos; también se emplean en cartuchos conectables para impresoras, video juegos y otros sistemas. Vea PROM, EPROM y EEPROM.

scan line

Una línea de pixeles. El término provienen de la acción de escanear del rastreo del CRT de los dispositivos de salida, que producen líneas sucesivas de salida en la superficie de despliegue.

scroll

Enrollar, desplazar. Moverse en forma continua hacia adelante, hacia atrás o hacia los lados a través de las imágenes en pantalla o dentro de una ventana. El enrollado implica un movimiento continuo o uniforme, de a una línea, carácter ó pixel a la vez, como si los datos estuvieran sobre un rollo de papel que está detrás de la pantalla.

shareware

Software compartido. Software distribuido sobre una base de ensayo a través de BBS, servicios en líneas, distribuidores de pedidos por correo y grupos de usuarios. Si se utiliza en forma regular, debe registrarse y pagarse por éste, con lo que se recibirá respaldo técnico y documentación adicional o el siguiente proceso de mejoramiento. Se requieren licencias de pago por distribución comercial.

sistema operativo

Programa maestro de control que opera la computadora. Es el primer programa que se carga cuando se enciende la computadora, y su parte central, llamada kernel (núcleo), reside en la memoria todo el tiempo. El sistema operativo puede ser desarrollado por el fabricante de la computadora o por terceros.

Es un componente importante de la computadora, porque establece los estándares para los programas de aplicación que se ejecutarán en éste. Todos los programas deben "dialogar" con el sistema operativo. También llamado ejecutivo o supervisor.

software
Instrucciones para la computadora. Una serie de instrucciones que realiza una tarea en particular se llama programa o programa de software. Las dos categorías principales son software de sistemas y software de aplicaciones.

strip
La colección de una o más líneas escaneadas en un bitmap. Las líneas escaneadas son en ocasiones agrupadas en strips para "bufferearlas" en memoria más eficientemente. También llamadas bandas en algunas especificaciones de formatos de archivos.

stroke
Información de caracter para ser representada dibujando líneas simples, usualmente por un plotter o algún otro dispositivo que responde al lápiz arriba, lápiz abajo y movimiento de comandos.

thumbnail
Una imagen pequeña derivada de una imagen grande usada para desplegar rápidamente una aproximación del contenido de una imagen.

tile
Una subsección 2D de un bitmap. Por ejemplo, un bitmap de 100 x 100 pixeles de tamaño puede dividirse en 4 tiles de 25 x 25 pixeles. Los pixeles en ocasiones son agrupados como tiles para conseguir un más eficiente uso de la memoria.

trailer
También conocido como *footer* (pie de archivo). Estructura de datos similar a la cabecera pero localizada al final del archivo.

uma (upper memory area)
Área de memoria superior. Memoria de la computadora entre 640Kb. y 1024 Kb.

umb (upper memory block)
Bloque de memoria superior. Bloques no utilizados en el UMA (640Kb-1MB.). Un proveedor de UMB es el software que puede cargar controladores y TSR en esta memoria.

unix
Sistema operativo multiusuario y multitarea de AT&T que corre en computadoras de micro a mainframes. UNIX está escrito en C (también desarrollador por AT&T), el cual puede estar compilado en muchos lenguajes diferentes de máquina; esto hace que UNIX corra en mayor variedad de hardware que cualquier otro programa de control. Así, UNIX se ha convertido en sinónimo de sistemas abiertos. Con los estándares que se han agregado con el paso del tiempo, UNIX ha evolucionado para convertirse en el prototipo para procesamiento distribuido e interoperabilidad.

vector

Se refiere a datos gráficos compuestos principalmente de representaciones de líneas y contornos de objetos, que pueden ser compactamente representados por juegos de puntos clave específicos. Un programa que despliegue de vector debe saber como dibujar líneas por puntos interpolados entre puntos clave.

vesa

Video Electronics Standards Association. Estandares de representación de video. San José, CA.

vl bus

VESA Local bus. Bus local de PC respaldado por VESA que provee una ruta de datos de 32 bits a velocidades hasta de 40 Mhz. (hasta 66 Mhz. para controladores establecidos en la tarjeta base). La ranura VL bus utiliza un conector Micro Channel de 32 bits adyacente a la ranura estándar ISA, EISA o MCA, permitiendo a los fabricantes diseñar tarjetas que utilizan sólo el bus local o ambos bus al mismo tiempo. El VL bus respalda hasta tres periféricos y la dominación del bus.

vram (video ram)

RAM de video. Circuitos de memoria especialmente diseñados en un panel de representación de video utilizado para retener la imagen que aparece en la pantalla.

APENDICE

A

APÉNDICE A

BIOS DEL VGA

El BIOS del VGA es un juego rutinas de bajo nivel que pueden ser accesadas ejecutando una interrupción (INT 10h) con parámetros especificados en los registros.

A continuación se lista las funciones más comunes para VGA.

Función 0h. Selección de Modo.

AH = 0
AL = Número del modo (0 a 13h)

Función 1h. Tamaño del cursor

AH = 1
CH = Inicio de línea escaneada (0-31)
CL = Fin de línea escaneada (0-31)

Función 2h. Posición del cursor

AH = 2
BH = Número de página de despliegue
DH = Renglón DL = Columna

Función 3h. Leer la posición del cursor

AH = 3
BH = Número de página de despliegue
Valores devueltos:
CH = Inicio de línea escaneada del cursor
CL = Fin de línea escaneada del cursor
DH = Renglón del cursor
DL = Columna del cursor

Función 5h. Selección de página activa

AH = 5
AL = Número de página de despliegue

Función 8h. Lectura de un caracter y sus atributos en la posición del cursor

AH = 8
BH = Número de la página de despliegue

AL = Código del caracter
AH = Atributo del caracter (solo en modo texto)

Función 9h. Escritura de un caracter y sus atributos en la posición del cursor

AH = 9
AL = Código del caracter
BH = Número de la página de despliegue
BL = Atributo (solo en modo texto) ó color (modos gráficos)
CX = Cuenta de repetición (de inicio a fin del renglón actual)

Función 0Ah. Escritura de un carácter sin atributos en la posición del cursor

AH = 0Ah
AL = Código del caracter
BH = Número de página de despliegue
BL = Color (modos gráficos)
CX = Cuenta de repetición (de inicio a fin del renglón actual)

Función 0Bh. Paleta de colores CGA (Modos 4,5 y 6)

AH = 0Bh
BH = 0:
BL = Color de fondo en gráficos ó color de borde en texto
BH = 1:
BL = Número de paleta (0 ó 1)

Función 0Ch. Escritura de un pixel

AH = 0Ch
AL = Valor del pixel
CX = Número de columna del pixel
DX = Número de renglón del pixel

Función 0Dh. Lectura de pixel

AH = 0Dh
CX = Número de columna del pixel
DX = Número de renglón del pixel
Valor devuelto:
AL = Valor del pixel

Función 0Eh. Escritura de un caracter con avance del cursor

AH = 0Eh
AL = Caracter a escribir
BH = Número de la página activa (solo modo texto)
BL = Color de carácter (solo modo gráfico)

Función 0Fh. Lectura del estado actual del video

AH = 0Fh

Valores devueltos:
AH = Número de caracteres por línea
AL = Modo de video actual
BH = Número de la página activa

Función 10h. Programación de registros de la paleta de colores EGA, MCGA, VGA

Subfunción 0h. Definir el registro de color

AH = 10h
AL = 00h
BL = Número de color a modificar (0-15)
BH = Color a escribir (0-63)

Subfunción 1h. Controla el registro del borde en alta resolución (sobrebarrido)

AH = 10h
AL = 01h
BL = Valor a escribir (0-255)

Subfunción 2h. Programa los 16 registros de la paleta y el borde de alta resolución

AH = 10h
AL = 02h
ES:DX = Apunta a una tabla de 17 bytes, siendo los primeros 16 los colores a cargar en los registros y el último el de sobrebarrido

Subfunción 3h. Selecciona intensidad ó parpadeo

AH = 10h
AL = 03h
Si BL = 1, el parpadeo se activará
Si BL = 0, el parpadeo se desactivará (la intensidad del primer plano será activada)

Subfunción 7h. Lectura de un registro de color

AH = 10h
AL = 07h
BL = Número de color a leer (0-15)
Valor devuelto:
BH = Valor leído

Subfunción 8h. Lectura del registro de color del borde

AH = 10h
AL = 08h
Valor devuelto:
BH = Valor leído

Subfunción 9h. Lectura de los registros de la paleta de colores y del registro del color del borde

AH = 10h
AL = 09h
ES:DX = Dirección del buffer de 17 bytes (16 valores de la paleta más 1 de sobrebarrido
Valor devuelto:
17 bytes almacenados en ES:DX

Subfunción 10h. Definir un registro de color

AH = 10h
AL = 10h
BX = Número del registro del color a programar
DH = Valor del componente rojo (0-63)
CH = Valor del componente verde (0-63)
CL = Valor del componente azul (0-63)

Subfunción 12h. Definir un bloque de registros de color

AH = 10h
AL = 12h
BX = Número del primer registro de color (0-255)
CX = Número de registros consecutivos
ES:DX = Apunta a una tabla que contiene los valores a programar, esto es rojo, verde, azul

Subfunción 13h. Seleccionar subjuego de color

AH = 10h
AL = 13h
Si BL = 0: Selecciona modo
BH = 0:4 subjuegos de 64 colores
BH = 1:16 subjuegos de 16 colores
Si BL = 1: Selecciona subjuego
BH = subjuegos (0-16)

Subfunción 15h. Lectura de un registro de color

AH = 10h
AL = 15h
BX = Número de registro del color a leer
Valor devuelto:
DH = Valor del componente rojo
CH = Valor del componente verde
CL = Valor del componente azul

Subfunción 17h. Lectura de un bloque de registros de color

AH = 10h

AL = 17h
BX = Número del primer registro de color (0-255)
CX = Número de registros (0-256)
ES:DX = Apunta a una tabla que recibirá los valores leídos en el orden rojo, verde, azul para cada registro

Subfunción 1Ah. Lectura de un subjuego

AH = 10h
AL = 1Ah
Valor devuelto:
BH = Número del subjuego de color actual
BL = 0 si 4 subjuegos están disponibles
BL = 1 si 16 subjuegos están disponibles

Subfunción 1Bh. Sustituye el color por niveles de grises

AH = 10h
AL = 1Bh
BX = Número del primer registro de color
CX = Número de registros consecutivos
BL = 1 si 16 subjuegos están disponibles

Función 1Bh. Devuelve el status de información del VGA

AH = 1Bh
BX = 0
ES:DI Apunta a un buffer de 64 bytes
Valor devuelto:
AL = 1B

El buffer devuelto contendrá la información como se muestra en la siguiente tabla:

Información del estado del VGA y funcionalidad.		
Número del byte	**Tamaño en bytes**	**Contenido**
0	4	Apunta a la tabla de funcionalidad estática
4	1	Modo de video actual
5	2	Número de caracteres por columna
7	2	Tamaño del área de video en bytes
9h	2	Desplazamiento actual del buffer REGEN
0Bh	16	Posición del cursor 2 palabras por página hasta 8 páginas
1Bh	1	Fin del cursor
1Ch	1	Inicio del cursor
1Dh	1	Página de despliegue actual
1Eh	2	Dirección del controlador CRT
20h	1	Valor del registro del modo CGA/MDA
21h	1	Valor del registro del color CGA/MDA
22h	1	Número de renglones en el texto
23h	1	Alto del caracter (en líneas escaneadas)
25h	1	Código de configuración del despliegue (despliegue activo)

Información del estado del VGA y funcionalidad. (continuación)		
Número del byte	**Tamaño en bytes**	**Contenido**
26h	1	Código de configuración del despliegue (despliegue inactivo)
27h	1	Número de colores en el modo actual (0 para modos mono)
29h	2	Número de páginas de despliegue en el modo actual
2Ah	1	Número de líneas escaneadas en el modo actual 0=200, 1=350, 2=400, 3=480
2Bh	1	Generador de caracter primario (0-7)
2Ch	1	Generador de caracter secundario (0-7)
2Dh	1	Miscelánea de información D5 = 1 Parpadeo habilitado D5 = 0 Intensidad del color de fondo habilitado D4 = 1 Emulación del cursor CGA habilitado D3 = 1 Inicialización de la paleta default habilitado D2 = 1 Despliegue monocromático enlazado D1 = 1 Conversión de escala de grises habilitado D0 = 1 Todos los modos habilitados en todos los monitores
2Eh	1	Reservado
2Fh	1	Reservado
30h	1	Reservado
31h	1	Tamaño de la memoria de despliegue: 0=64Kb, 1=128Kb, 2=192Kb, 3=256Kb
32h	1	Información del puntero almacenado D5 = 1 Extensión DCC está activa D4 = 1 Paleta disponible activa D3 = 1 Fuentes gráficas disponibles activa D2 = 1 Fuentes alfa disponibles activa D1 = 1 Área dinámica activa D0 = 1 Juego de caracteres activos
33h a 33F		Reservados

APENDICE B

APÉNDICE B

Código fuente de la aplicación

```
{Programa Principal: Programa Realizado para la  tesis titulada "Formatos
Gráficos".
        Realizado por :
        Cantero Ramírez Carlos
        Trejo Bonaga Marilu.
        Asesores :
        Manzo Salazar Itsmael
        Monterrosa Escobar Amilcar A.}
Program Formatos_graficos;

Uses crt,dos,graph,graficos,texto,raton,
   Imag_Bmp,Imag_Pcx,Imag_Tga,Imag_Gif,imag_Tif;
{ imag_xxx son unidades creadas a partir de los programas explicados en los capítulos III, IV y V, sin
embargo el código fuente se incluye en el disco que acompaña a este trabajo de tesis.}

CONST
Extensiones='BMP-PCX-TGA-GIF-TIF';
ESC=#27;
Enter=#13;
F1=#59;
F2=#60;
F3=#61;

Type
 Lista_archivos_imagen=Record
  nombre:String[10];
  Ext:String[3];
  Dir:Boolean;
  Info:String[30];
 End;
 arreglo_de_archivos=Array [1..1000] of Lista_archivos_imagen;

Var
 Ran_Uni_x1,Ran_Uni_x2,Rel_x,Rel_y,
 Rx,Ry,Cx,Cy,No_archivo,contd,porcent,numArch,desy:Word;
 Dir_imagen:^arreglo_de_archivos;
 Numero:Integer;
 I_ini,I_fin:Word;
 Maxcordx,maxcordy:longint;
 Que_imagen,Bot_IZ_DE:Byte;
 dir_Actual:dirarchv;
 opcion,opcion_conv:Char;
 path,archiv,Archivo,Uni_actual,unidades:string;
 todo_bien:Boolean;
```

```
 Procedure Presenta;                                                    {Procedimiento que arma la pantalla principal}
Var
 cont:byte;
 dir_temp:string;
 triangulo:Array [1..3] of pointType;
Begin
  Cleardevice;
  Getdir(0,dir_temp);
  Chdir(path);
   MaxCordX:=getmaxx;
   MaxCordy:=getmaxy;
   Botong(0,0,maxCordx,maxCordy,1,0,254);
   Botong(Rel_X*1,Rel_y*1,Rel_x*99,Rel_y*7,1,0,-1);                                  {Boton del Titulo}
   Escribeg(rel_x*31+2,rel_y*1,251,1,5,'FORMATOS GRAFICOS');                          {Titulo}
   Escribeg(rel_x*31+4,rel_y*1-1,252,1,5,'FORMATOS GRAFICOS');                        {Titulo}
   Escribeg(rel_x*31+1,rel_y*1+2,251,1,5,'FORMATOS GRAFICOS');                        {Titulo}
   Botong(Rel_X*55,Rel_y*43,Rel_x*99,Rel_y*47,1,0,-1);                        {Botón del nombre de la Imagen}
   Escribeg(rel_x*57  ,rel_y*43  ,250,2,7,'Nombre: ');                          {Nombre de la imagen}
   Escribeg(rel_x*57+1,rel_y*43+1,251,2,7,'Nombre: ');                          {Nombre de la imagen}
   Botong(Rel_X*55,Rel_y*49,Rel_x*99,Rel_y*90,1,0,-1);                   {Botón de la Informacion de la imagen}
   Botong(Rel_X*1,Rel_y*91,Rel_x*80,Rel_y*95,1,0,-1);                          {Botón de instrucciones}
                                                                                          {Instrucciones }
   Escribeg(Rel_x*2  ,rel_y*91  ,252,8,1,'Botón Izq=Previo    Botón Der=Ver     F1=Ayuda  F2=Conversión');
   Escribeg(Rel_x*2+1,rel_y*91+1,253,8,1,'Botón Izq=Previo    Botón Der=Ver     F1=Ayuda  F2=Conversión');
   Botong(Rel_X*81,Rel_y*91,Rel_x*99,Rel_y*95,1,0,-1);                                {Botón de Esc}
   Escribeg(Rel_x*82  ,rel_y*91  ,252,8,1,'Esc=Salir');                                {Salir}
   Escribeg(Rel_x*82+1,rel_y*91+1,253,8,1,'Esc=Salir');                                {Salir}
   Botong(Rel_X*1,Rel_y*14,Rel_x*50,Rel_y*18,1,1,254);                                 {Ruta}
   getdir(0,uni_actual);
   Escribeg(Rel_x*2  ,rel_y*14  ,250,2,6,Mayuscula(uni_actual));                       {Ruta}
   Escribeg(Rel_x*2+1,rel_y*14+1,253,2,6,Mayuscula(uni_actual));                       {Ruta}
   Escribeg(Rel_x*2  ,rel_y*9  ,252,7,1,'Ruta :');                                     {Ruta}
   Escribeg(Rel_x*2+1,rel_y*9+1,251,7,1,'Ruta :');                                     {Ruta}
   Botong(Rel_X*1,Rel_y*25,Rel_x*47,Rel_y*85,1,1,-1);                                  {Directorios}
   Botong(Rel_X*47+3,Rel_y*25,Rel_x*50,Rel_y*28,1,0,254);                      {Desplazamiento Arriba}
   Triangulo[1].x:=Rel_X*48+(rel_x div 2);
   Triangulo[1].y:=Rel_y*26;
   Triangulo[2].x:=Rel_X*48;
   Triangulo[2].y:=Rel_y*27;
   Triangulo[3].x:=Rel_X*49;
   Triangulo[3].y:=Rel_y*27;
   setfillpattern(patron,253);
   FillPoly(3,triangulo);                                                   {Puntas de desplazamiento}
   Triangulo[1].x:=Rel_X*48+(rel_x div 2);
   Triangulo[1].y:=Rel_y*84;
   Triangulo[2].x:=Rel_X*48;
   Triangulo[2].y:=Rel_y*83;
   Triangulo[3].x:=Rel_X*49;
   Triangulo[3].y:=Rel_y*83;
   FillPoly(3,triangulo);                                                   {Puntas de desplazamiento}
   Botong(Rel_X*47+3,Rel_y*29,Rel_x*50,Rel_y*81,1,1,-1);                       {Barra de Desplazamiento }
   Botong(Rel_X*47+5,desy+2,rel_x*50-2,desy+(Rel_y*3)-2,1,0,-1);        {Boton de barra de Desplazamiento }
   Botong(Rel_X*47+3,Rel_y*82,Rel_x*50,Rel_y*85,1,0,-1);                        {Desplazamiento Abajo }
   Escribeg(Rel_x*2  ,rel_y*21  ,252,7,1,'Directorio :');                              {Directorios }
```

```
  Escribeg(Rel_x*2+1,rel_y*21+1,251,7,1,'Directorio :');                    {Directorios }
  unidades:=Exis_uni('*');
  Ran_Uni_x1:=Rel_X*2;
  Ran_Uni_x2:=Rel_x*(2*length(unidades)+2);                  {Rango de las unidades de disco}
  For Cont:=1 to length(unidades) do
   Begin
    Botong(Rel_X*(2*cont)+1,Rel_y*88,Rel_x*(2*cont+2)-1,Rel_y*90,1,0,-1);     {Unidades de disco }
    Escribeg(Rel_X*(2*cont)+(rel_x div 2) ,Rel_y*88,251,2,5,unidades[cont]);          {Directorios }
    Escribeg(Rel_X*(2*cont)+(rel_x div 2)+1,Rel_y*88+1,251,2,5,unidades[cont]);       {Directorios }
   End;
  Escribeg(Rel_x*2  ,rel_y*85  ,252,7,1,'Unidades de disco :');                          {Discos }
  Escribeg(Rel_x*2+1,rel_y*85+1,253,7,1,'Unidades de disco :');                          {Discos }
  chdir(dir_temp);
End;


Procedure Prepara_archivos;                  { Llena un arreglo con una lista de archivos de imagen}
Var                                          {y su información, llamando al procedimiento directorio}
i,j:integer;
 Begin
  Directorio('*.*',$10,numero);
  j:=0;
  Botong(Rel_X*1,Rel_y*25,Rel_x*47,Rel_y*85,1,1,254);                                   {Directorios}
  Botong(Rel_X*47+3,Rel_y*29,Rel_x*50,Rel_y*81,1,1,254);                     {Barra de Desplazamiento}
  Botong(Rel_X*47+5,desy+2,rel_x*50-2,desy+(Rel_y*3)-2,1,0,254);     {Botón de barra de Desplazamiento}
  For i:=1 to numero do
   with lista_dir[i] do
    Begin
     if dir or ((Pos(Ext,Extensiones)<>0) and (length(ext)=3)) THEN
      Begin
       inc(j);
       Dir_imagen^[j].nombre:=nombre;
       Dir_imagen^[j].ext:=ext;
       Dir_imagen^[j].dir:=dir;
       Dir_imagen^[j].info:=ifs(dir,'Directorio '+int_str(j),'Informe de la imagen '+int_str(j));
       If Ext='BMP' Then Dir_imagen^[j].info:=Resolucion_Bmp(nombre+'.'+ext);
       If Ext='PCX' Then Dir_imagen^[j].info:=Resolucion_Pcx(nombre+'.'+ext);
       If Ext='GIF' Then Dir_imagen^[j].info:=Resolucion_Gif(nombre+'.'+ext);
       If Ext='TGA' Then Dir_imagen^[j].info:=Resolucion_Tga(nombre+'.'+ext);
       If Ext='TIF' Then Dir_imagen^[j].info:=Resolucion_Tif(nombre+'.'+ext);
      End;
    End;
   numero:=j;
  For i:=numero+1 to numero+18 do
   Begin
    Dir_imagen^[i].nombre:='';
    Dir_imagen^[i].ext:='';
    Dir_imagen^[i].dir:=False;
    Dir_imagen^[i].info:='';
   End;
  numero:=j;
 End;
```

```
Procedure Lista_archivos(ini,fin:integer;Arriba:boolean);      {Lista los archivos encontrados en la parte}
                                                               {de la pantalla destinada para su presentación}
Var
i,j,k:integer;
 dir_temp:string;
 Begin
  j:=0;k:=0;
  Getdir(0,dir_temp);
  Chdir(path);
  Botong(Rel_X*1,Rel_y*14,Rel_x*50,Rel_y*18,1,1,254);                                   {Ruta}
  Escribeg(Rel_x*2  ,rel_y*14  ,250,2,6,Mayuscula(Dir_temp));                           {Ruta}
  Escribeg(Rel_x*2+1,rel_y*14+1,253,2,6,Mayuscula(Dir_temp));                           {Ruta}
  if numero<=fin Then fin:=numero;
  For i:=ini to fin do
   Begin
    inc(j);
    If Arriba Then k:=i+1 else k:=i-1;
     with Dir_imagen^[k] do
       Begin
        setcolor(254);
        Escribeg(Rel_X*02,(Rel_y*21)+(j * (Rel_y*29-Rel_y*25)),254,ifi(dir,7,2),ifi(dir,1,6),nombre);
        Escribeg(Rel_X*16,(Rel_y*21)+(j * (Rel_y*29-Rel_y*25)),254,ifi(dir,7,2),ifi(dir,1,6),Ext);
        Escribeg(Rel_X*23,(Rel_y*21)+(j * (Rel_y*29-Rel_y*25)),254,ifi(dir,7,2),ifi(dir,1,6),Info);
       End;
    with Dir_imagen^[i] do
      Begin
       setcolor(254);
       Escribeg(Rel_X*02,(Rel_y*21)+(j * (Rel_y*29-Rel_y*25)),ifi(dir,251,253),ifi(dir,7,2),ifi(dir,1,6),nombre);
       Escribeg(Rel_X*16,(Rel_y*21)+(j * (Rel_y*29-Rel_y*25)),ifi(dir,251,253),ifi(dir,7,2),ifi(dir,1,6),Ext);
       Escribeg(Rel_X*23,(Rel_y*21)+(j * (Rel_y*29-Rel_y*25)),ifi(dir,251,253),ifi(dir,7,2),ifi(dir,1,6),Info);
      End;
   End;
  Chdir(dir_temp);
 End;


Procedure Limpia_variables;                                                              {Limpia Variabels }
 Begin
  presenta;
  For i:=1 to 90 do
   Begin
    Dir_imagen^[i].nombre:='';
    Dir_imagen^[i].ext:='';
    Dir_imagen^[i].dir:=False;
    Dir_imagen^[i].info:='';
    lista_dir[i].nombre:='';
    lista_dir[i].ext:='';
    lista_dir[i].dir:=False;
   End;
  Prepara_archivos;
 End;
```

```
Function Arch_imagen:Word;              {Procedimiento que retorna el archivo seleccionado con el ratón}
Var
x1,y1,x2,y2,i,cont:Word;
 Begin
  porcent:=Trunc((Rel_y*81-Rel_y*29)/numero);
   Opcion:=chr(0);
   bot_iz_de:=0;
   Repeat
    raton_botonG(Rx,Ry,4,2);
    if rango_ratonG(Rel_X*1,Rel_y*25,Rel_x*47,Rel_y*84,bot_iz_de) Then
     Begin
      setcolor(254);
      Rectangle(Rel_X*1+3,(Rel_y*25+(cont+2)),Rel_x*47-3,Rel_y*29+(cont-2));
      Cont:=((ry-Rel_y*25) div (Rel_y*29-Rel_y*25));
      cont:=cont * (Rel_y*29-Rel_y*25);
      numArch:=(cont div (Rel_y*29-Rel_y*25))+1;
      x1:=Rel_X*1+3;y1:=(Rel_y*25+(cont+2));x2:=Rel_x*47-3;y2:=Rel_y*29+(cont-2);
      if numArch<=numero Then
       Begin
        Botong(x1,y1,x2,y2,1,0,-1);                                  {Botón de selección de archivo}
        while rango_ratonG(x1,y1,x2,y2,bot_iz_de) and (pressboton=0) and not keypressed do
         Begin
          raton_botonG(Rx,Ry,4,2);
          Raton_pumas(Rx,Ry,False);
         End;
       End;
      Bot_IZ_DE:=Pressboton;
     End;
    if rango_ratonG(Rel_X*47+3,Rel_y*25,Rel_x*50,Rel_y*28,bot_iz_de) Then
     Begin
      if pressboton=1 Then
       Begin
        Botong(Rel_X*47+3,Rel_y*25,Rel_x*50,Rel_y*28,1,1,-1);        {Desplazamiento Arriba presionado}
        delay(10);
        setcolor(254);
         Rectangle(Rel_X*47+5,desy+2,Rel_x*50-2,desy+(Rel_y*3)-2);
          contd:=contd-ifi(contd<=1,0,1);
          desy :=desy-ifi(contd<=1,0,porcent);
          L_ini:=ifi((contd+13>=numero),L_ini,contd);
          l_fin:=ifi((contd+13>=numero),l_fin,contd+14);
          If not(contd+13>=numero) Then Lista_archivos(L_ini,L_fin,True);
       Botong(Rel_X*47+5,desy+2,rel_x*50-2,desy+(Rel_y*3)-2,1,0,-1);  {Botón de barra de Desplazamiento}
        Botong(Rel_X*47+3,Rel_y*25,Rel_x*50,Rel_y*28,1,0,-1);       {Desplazamiento Arriba sin presionar}
       End;
     End;
    if rango_ratonG(Rel_X*47+3,Rel_y*82,Rel_x*50,Rel_y*85,bot_iz_de) Then
     Begin
      if pressboton=1 Then
       Begin
        Botong(Rel_X*47+3,Rel_y*82,Rel_x*50,Rel_y*85,1,1,-1);        {Desplazamiento Abajo presionado}
        delay(10);
        setcolor(254);
         Rectangle(Rel_X*47+5,desy+2,Rel_x*50-2,desy+(Rel_y*3)-2);
          contd:=contd+ifi(contd>=numero,0,1);
```

```
      desy:=desy+ifi(contd>=numero,0,porcent);
      L_ini:=ifi((contd+13>=numero),L_ini,contd);
      l_fin:=ifi((contd+13>=numero),l_fin,contd+14);
      If not(contd+13>=numero) Then Lista_archivos(L_ini,L_fin,False);
     Botong(Rel_X*47+5,desy+2,rel_x*50-2,desy+(Rel_y*3)-2,1,0,-1);  {Botón de barra de Desplazamiento }
     Botong(Rel_X*47+3,Rel_y*82,Rel_x*50,Rel_y*85,1,0,-1);          {Desplazamiento Abajo sin precionar}
    End;
   End;
  If Rango_ratonG(Ran_uni_x1,Rel_y*88,Ran_uni_x2,Rel_y*90,Bot_iz_de) Then
   Begin
    if pressboton=1 Then
     Begin
      For Cont:=1 to length(unidades) do
       If Rango_ratong(Rel_X*(2*cont)+1,Rel_y*88,Rel_x*(2*cont+2)-1,Rel_y*90,Bot_iz_de) Then
        Begin
         Botong(Rel_X*(2*cont)+1,Rel_y*88,Rel_x*(2*cont+2)-1,Rel_y*90,1,1,-1);          {Unidades de disco }
          {Si-}
           Chdir(unidades[Cont]+':');
          {Si+}
          if ioresult<>0 Then Write(chr(7));
          desy:=Rel_y*29;
          Prepara_archivos;
          l_ini:=1;
          l_fin:=ifi(numero<15,numero,15);
          contd:=1;
          numArch:=1;
          Lista_archivos(l_ini,l_fin,true);
         Botong(Rel_X*(2*cont)+1,Rel_y*88,Rel_x*(2*cont+2)-1,Rel_y*90,1,0,-1);          {Unidades de disco }
        End;
     End;
    End;
   If Rx>=MaxCordx-33 Then
    Raton_posis(MaxCordx-33,Ry)
   Else
    Raton_pumas(Rx,Ry,False);
   if rango_ratonG(Rel_X*81,Rel_y*91,Rel_x*99,Rel_y*95,bot_iz_de) and (pressboton=1) Then opcion:=Esc;
if KEYPRESSED tHEN opcion:=READKEY;
   If Opcion=Enter Then Bot_iz_de:=1;
  Until (bot_iz_de=1) or (bot_iz_de=2) OR (opcion=Esc) or (opcion=F1) or (opcion=F2);
 FIJA_COLOR;
 setfillpattern(patron,254);
 setcolor(254);
 BAR(Rel_X*55,Rel_y*9,Rel_X*99,Rel_y*40);
 setcolor(254);
 Rectangle(Rel_X*1+3,(Rel_y*25+(cont+2)),Rel_x*47-3,Rel_y*29+(cont-2));
 Arch_imagen:=numArch+l_ini-1;
End;

Function Checa_Imagen(No_Arch:Word):Byte;          {Verifica si  el archivo seleccionado es un archivo}
const                                  {de imagen o un directorio, si es un directorio realiza un cambio de directorio}
Ext:Array [1..5] of string[3]=('BMP','PCX','TGA','GIF','TIF');
Var
 i:byte;
 strg:String[3];
```

```
Begin
 Archivo:=Mayuscula(Dir_imagen^[No_arch].nombre+'.'+Dir_imagen^[No_arch].EXT);
 i:=0;
 if (opcion=Esc) OR (Dir_imagen^[No_arch].DIR) Then
  Begin
   if (Dir_imagen^[No_arch].DIR) and (opcion<>Esc) and (opcion<>F2) Then
    Begin
     CHDIR(ifs(ARCHIVO[1]='.','..',Archivo));
     Botong(Rel_X*1,Rel_y*14,Rel_x*50,Rel_y*18,1,1,254);  { Ruta }
     getdir(0,uni_actual);
     Chdir(path);
     Escribeg(Rel_x*2  ,rel_y*14  ,250,2,6,Mayuscula(uni_actual));                    {Ruta }
     Escribeg(Rel_x*2+1,rel_y*14+1,253,2,6,Mayuscula(uni_actual));                    { Ruta }
     Chdir(Uni_actual);
     desy:=Rel_y*29;
     Prepara_archivos;
     l_ini:=1;
     l_fin:=ifi(numero<15,numero,15);
     contd:=1;
     numArch:=1;
     Lista_archivos(l_ini,l_fin,true);
    End;
   i:=7;
  End
 Else
  Repeat
   inc(i);
   strg:=Dir_imagen^[No_arch].EXT;
  Until(strg=ext[i]) or (i>=7);
 Checa_imagen:=i;
End;

Procedure convierte_a(opcion_conv:char);                                  {Procedimiento de conversión}
 Begin
  case opcion_conv of
   '1':Esc_imagen_Bmp;
   '2':Esc_imagen_Pcx;
   '3':Esc_imagen_Gif;
   '4':Esc_imagen_Tif;
   '5':Esc_imagen_Tga;
   '6'::
  End;
 Getdir(0,dir_temp);                                                      {Actualiza el directorio principal}
 Chdir(path);
 Chdir(dir_temp);
 End;

Begin                                                                     {Cuerpo principal del programa}
clrscr;
new(Dir_imagen);
If raton_inicial<>0 Then
  Begin
   getdir(0,path);
   Raton_pumas(Rx,Ry,True);
```

```
Inicializa(16,4,path);
MaxCordX:=getmaxx;
MaxCordy:=getmaxy;
Rel_x:=Round(MaxCordx/100);Rel_y:=Round(MaxCordy/100);
desy:=Rel_y*29;
Limpia_variables;
l_ini:=1;
l_fin:=ifi(numero<15,numero,15);
contd:=1;
numArch:=1;
Repeat
  Lista_archivos(l_ini,l_fin,true);
  No_Archivo:=Arch_imagen;
  Case Opcion Of
   F1:
    Begin
     Botong(Rel_x*20,rel_y*20,Rel_x*80,Rel_y*80,1,0,254);
     Escribeg(Rel_x*45,rel_y*22,252,2,7,'AYUDA');
     Escribeg(Rel_x*45+2,rel_y*22+2,253,2,7,'AYUDA');
     Escribeg(Rel_x*22+2,rel_y*26+2,253,2,7,'Movimientos con el ratón');
     Escribeg(Rel_x*22+2,rel_y*30+2,253,2,6,'Botón izquierdo selección ó previo');
     Escribeg(Rel_x*22+2,rel_y*32+2,253,2,6,'Botón derecho ver imagen');
     Escribeg(Rel_x*22+2,rel_y*36+2,253,2,7,'Movimientos con el Teclado');
     Escribeg(Rel_x*22+2,rel_y*40+2,253,2,6,'Enter = Ver Imagen');
     Escribeg(Rel_x*22+2,rel_y*42+2,253,2,6,'Esc = Salir');
     Escribeg(Rel_x*22+2,rel_y*44+2,253,2,6,'F1 = Ayuda');
     Escribeg(Rel_x*22+2,rel_y*46+2,253,2,6,'F2=Conversiones');
     Escribeg(Rel_x*22+2,rel_y*48+2,253,2,6,' Conversión trabaja sobre el archivo actual convirtiendolo');
     Escribeg(Rel_x*22+2,rel_y*50+2,253,2,6,' a cualquiera de los dem s formatos :');
     Escribeg(Rel_x*22+2,rel_y*54+2,253,2,6,' ( 1 ) Archivo BMP ');
     Escribeg(Rel_x*22+2,rel_y*57+2,253,2,6,' ( 2 ) Archivo PCX ');
     Escribeg(Rel_x*22+2,rel_y*60+2,253,2,6,' ( 3 ) Archivo GIF ');
     Escribeg(Rel_x*22+2,rel_y*63+2,253,2,6,' ( 4 ) Archivo TIF ');
     Escribeg(Rel_x*22+2,rel_y*66+2,253,2,6,' ( 5 ) Archivo TGA ');
     Escribeg(Rel_x*30,rel_y*77,252,2,6,' Presione una tecla para salir de ayuda');
     Escribeg(Rel_x*30+2,rel_y*77+2,253,2,6,' Presione una tecla para salir de ayuda');
     Readkey; Presenta;
    End;
  Else
    Begin
     Que_Imagen:=Checa_imagen(No_Archivo);
     opcion_conv:=' ';
     if (Que_imagen<>7) and (opcion=F2)Then
      Begin
       Botong(Rel_x*30,rel_y*30,Rel_x*70,Rel_y*66,1,0,254);
       Escribeg(Rel_x*39,rel_y*32;252,2,7,'C O N V E R S I O N');
       Escribeg(Rel_x*39+2,rel_y*32+2,253,2,7,'C O N V E R S I O N');
       Escribeg(Rel_x*32  ,rel_y*38  ,252,2,6,'( 1 ) Convertir al formato BMP ');
       Escribeg(Rel_x*32+2,rel_y*38+2,253,2,6,'( 1 ) Convertir al formato BMP ');
       Escribeg(Rel_x*32  ,rel_y*42  ,252,2,6,'( 2 ) Convertir al formato PCX ');
       Escribeg(Rel_x*32+2,rel_y*42+2,253,2,6,'( 2 ) Convertir al formato PCX ');
       Escribeg(Rel_x*32  ,rel_y*46  ,252,2,6,'( 3 ) Convertir al formato GIF ');
       Escribeg(Rel_x*32+2,rel_y*46+2,253,2,6,'( 3 ) Convertir al formato GIF ');
       Escribeg(Rel_x*32  ,rel_y*50  ,252,2,6,'( 4 ) Convertir al formato TIF ');
```

```
      Escribeg(Rel_x*32+2,rel_y*50+2,253,2,6,'( 4 ) Convertir al formato TIF ');
      Escribeg(Rel_x*32  ,rel_y*54  ,252,2,6,'( 5 ) Convertir al formato TGA ');
      Escribeg(Rel_x*32+2,rel_y*54+2,253,2,6,'( 5 ) Convertir al formato TGA ');
      Escribeg(Rel_x*32  ,rel_y*62  ,252,2,6,'«« Teclee el n£mero correspondiente »»');
      Escribeg(Rel_x*32+1,rel_y*62+1,251,2,6,'«« Teclee el n£mero correspondiente »»');
      repeat
       opcion_conv:=Readkey;
      until (opcion_conv=ESC) or (pos(opcion_conv,'12345')<>0);
      Bot_iz_de:=1; presenta;
     End;
     Todo_bien:=False;
     if (opcion_conv<>Esc) Then
      Case Que_imagen of
       1:                                                          {Para imágenes  con extensión BMP}
        todo_bien:=Analiza_Bmp(rel_x,rel_y,Bot_iz_de,Archivo,opcion);
       2:                                                          {Para imágenes con extensión PCX}
        todo_bien:=Analiza_Pcx(rel_x,rel_y,Bot_iz_de,Archivo,opcion);
       3:                                                          {Para imágenes con extensión TGA}
        todo_bien:=Analiza_Tga(rel_x,rel_y,Bot_iz_de,Archivo,opcion);
       4:                                                          {Para imágenes con extensión GIF}
        todo_bien:=Analiza_Gif(rel_x,rel_y,Bot_iz_de,Archivo,opcion);
       5:                                                          {Para imágenes con extensión TIF}
        todo_bien:=Analiza_Tif(rel_x,rel_y,Bot_iz_de,Archivo,opcion);
       else                                                        {Para Archivos de imagen erróneo}
        Begin
         If opcion=ESc Then
          Begin
           Botong(Rel_X*81,Rel_y*91,Rel_x*99,Rel_y*95,1,1,-1);                    {Botón de Esc }
           DELAY(30);
           Botong(Rel_X*81,Rel_y*91,Rel_x*99,Rel_y*95,1,0,-1);                    {Botón de Esc }
          End;
        End;
      End;
     IF Todo_bien Then
      Begin
       If bot_iz_de=2 Then  Presenta;
       if opcion=F2 Then
        Begin
         convierte_a(opcion_conv);
         Limpia_variables;
        End;
       End;
     End;
    End;
   UNTIL (opcion=ESC);
   Cleardevice;
   Restorecrtmode;
   clrscr;
  End
  Else
     Writeln('No hay raton instalado...');
dispose(Dir_imagen);
 CHDIR(path);                                        {Retorna al directorio donde se ejecuto el programa }
End.                                                                          {Fin del programa}
```

APENDICE C

APÉNDICE C

Unidades Complementarias en Pascal

Existen 3 unidades complementarias al programa principal, Grafico.TPU, Raton.TPU y Texto.TPU, las cuales contienen funciones comunes, que pueden ser utilizadas para cualquier otro programa, algunas otras son funciones exclusivas del programa principal, sin embargo, si son bien utilizadas pueden incluirse en otro programa de características similares. El código de estas unidades no se describirá a detalle, solo se dará una idea general de la función.

GRAFICOS.TPU

```
{Unidad Gráficos: Unidad Realizada para el programa de tesis titulado "Formatos Gráficos".
                    Realizado por :
                                    Cantero Ramirez Carlos
                                    Trejo Bonaga Marilu.
                     Asesores :
                                    Manzo Salazar Itsmael
                                    Monterrosa Escobar Amilcar A.}
Unit Graficos;
Interface
 Uses crt,dos,graph;

 Const
  patron:fillpatternType=($FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF);

 Function Inicializa(Driver,Modo:integer;path:string):boolean;
 Procedure fija_Color;
 Procedure EscribeG(x,y,color,tipo,tam:integer;mensaje:string);
 Procedure BotonG(x, y, x1, y1,grueso, pres_sn,color:Integer);
 Procedure Cambia_color(color,r,v,a:Byte);
 Procedure Carga_colores(color:Integer;Var r,v,a:Byte);

Implementation

var
  AutoDetectPointer : pointer;

Function Inicializa(Driver,Modo:integer;path:string):boolean;
{Descripción general: Función que activa el modo gráfico, que se desea utilizar.
        Variables de entrada:
            Driver: Es el numero de Driver de los graficos, (0..10)  si se cuentan con todos los drivers
                    de Borland y 16 si se cuenta con los drivers adicionales.
        Valor de retorno: Retorna falso si ocurre un error, si no hay errores retorna verdadero.}
```

```
var
 ErrorG:integer;
 x,y:word;

Begin
 if (Driver=16) Then
   Driver := InstallUserDriver('Svga256',nil);
 if Driver=grError Then
     halt(1);
InitGraph(Driver, Modo, path);
   asm
     mov ax,000fh
     mov cx,8
     mov dx,8
     int 33h
    end;
   x:=getmaxx;
   y:=getmaxy;
   asm
     mov ax,x
     mov dx,ax
     mov ax,0007h
     mov cx,0h
     int 33h
     mov ax,y
     mov dx,ax
     mov ax,0008h
     mov cx,0h
     int 33h
   end;
 ErrorG:=graphResult;
 If ErrorG<>grok Then
  Begin
   Writeln('Error en graficos : ',GraphErrormsg(ErrorG));
   Inicializa:=False;
   Restorecrtmode;
  End
 Else
  Begin
   cleardevice;
   Inicializa:=True;
  End;
End;

Procedure fija_Color;
{Descripción general:  Fija 5 colores los cuales son usados en el programa principal, azul, dorado,
                       blanco, gris y negro.
        Variables de entrada: Ninguna
        Valor de retorno:  Ninguno
    Toma 4 Colores para manipular la ventana
```

	r	v	a	Color
Dorado	181	173	016	250
Azul	000	000	255	251
Blanco	255	255	255	252
negro	000	000	000³	253

```
                                        gris        100   100   100   254
                                        auxiliar1   000   000   000   255 }
Var
 i,y,k:integer;
Begin
  cambia_color(250,181 ,173,016 );
  cambia_color(251,000 ,000,230 );
  cambia_color(252,255 ,255,255 );
  cambia_color(253,000 ,000,000 );
  cambia_color(254,170 ,170,170 );
End;

Procedure EscribeG(x,y,color,tipo,tam:integer;mensaje:string);
{Descripción general : Escribe un texto en el modo gráfico.
        Variables de entrada:
                x,y: Coordenadas gráficas donde se colocara el texto.
                Color: Es el color del texto.
                Tipo: es el tipo de fuente a utilizar.
                Tam: es el tamaño de la letra.
                Mesage: Es el texto a escribir.
        Valor de retorno: Despliega el texto en pantalla, no retorna valor.}
var
 anterior:word;
Begin
 anterior:=getcolor;
 settextstyle(tipo,0,tam);
 setcolor(color);
 outtextxy(x,y,mensaje);
End;

Procedure BotonG(x, y, x1, y1,grueso, pres_sn,color:Integer);
{Descripción general: Coloca un gráfico en forma de botón.
        Variables de entrada:
                x,y: Coordenadas de la esquina superior izquierda.
                x1,y1: Coordenadas de la esquina inferior derecha.
                Grueso: Grosor del la linea del gráfico.
                Pres__sn : Dos posibles valores para determinar si la apariencia del botón es
                presionado o no.
                Color: es el color del botón si el boton es presionado el color es el mismo pero mas
                claro.
        Valor de retorno: Ninguno.}
Begin
 setlinestyle(0,1,grueso);
 setcolor(15);
 fija_Color;
 Case (pres_sn) of
  0:
   Begin
    If color<>-1 Then
     Begin
      setfillpattern(patron,color);
      bar(x,y,x1,y1);
     End;
    setcolor(252);
```

```
  rectangle(x,y,x1,y1);
  setcolor(253);
  line (x,y1,x1,y1); line (x1,y,x1,y1);
 End;
1:
 Begin
  If color<>-1 Then
   Begin
    setfillpattern(patron,color);
    bar(x,y,x1,y1);
   End;
  setcolor(252);
  rectangle(x,y,x1,y1);
  setcolor(253);
  line (x,y,x1,y);
  line (x,y,x,y1);
 End;
End;

Procedure Cambia_color(color,r,v,a:Byte);
{Descripción general: Modifica un color en la paleta de colores activa.
        Variables de entrada:
                Color :Es el numero de color a modificar.
                r,v,a: Son las tonalidades de rojo, verde y azul para el color.
        Valor de retorno: Ninguno.}
BEGIN
 asm
  mov dx,3C8h
  mov al,color
  out dx,al
  inc dx
  mov al,r
  out dx,al
  mov al,v
  out dx,al
  mov al,a
  out dx,al
 End;
End;

Procedure Carga_colores(color:Integer;Var r,v,a:Byte);
{Descripción general: Lee los tonos de un color de la paleta activa.
        Variables de entrada:
                Color: Numero de color.
                r,v,a :Variables donde se almacenan los 3 tonos.
        Valor de retorno: La palabra var dentro de los paréntesis indica que las variables que entran
saldrán con un valor, que la función donde fue llamado respetara estos valores.}
 Var
  rr,vv,aa:byte;
 Begin
  asm
   mov ah,10h
   mov al,15h
```

```
  mov bx,color
  int 10h
  mov rr,dh
  mov vv,ch
  mov aa,cl
 end;
  r:=rr shl 2;
  v:=rr shl 2;
  a:=rr shl 2;
 End;
End.
```

RATON.TPU

```
{ Unidad Raton: Unidad Realizada para el programa de tesis titulado "Formatos Gráficos".
        Realizado por :
                Cantero Ramírez Carlos
                Trejo Bonaga Marilu.
        Asesores :
                Manzo Salazar Itsmael
                Monterrosa Escobar Amilcar}

Unit Raton;
Interface
 Uses crt,graph;
 Var
  Cordxy          :Array [1..33,1..60] of word;
  Anima_mouse:Integer;

 Function raton_inicial:Integer;                                  {Verifica si existe Raton Instalado}
 Function rango_raton(xs,ys,xi,yi:byte):boolean;
 Function rango_ratonG(xs,ys,xi,yi:Word;estado:Byte):boolean;
 Function PressBoton:byte;
 Procedure raton_Activa;                                          {Activa el Ratón}
 Procedure raton_desactiva;                                       { Desactiva el Ratón}
 Procedure raton_boton(Var x,y:Word;boton:Byte);                  {Da coordenadas y el Botón Presionado}
 Procedure raton_botonG(Var x,y:Word;boton,Estado:Byte);          {Da coordenadas y el Botón Presionado}
 Procedure raton_posis(x,y:Word);                                 {Posiciona el Ratón}
 Procedure raton_pumas(x,y:word;Inicial:Boolean);
 Procedure Pumas_Cierra(x,y:word);
 Procedure Pumas_abre(x,y:word);

Implementation

 Function raton_inicial:Integer;
{Descripción general: Detecta si hay ratón instalado.
        Variables de entrada:
                Ninguna.
        Valor de retorno: Valor 1 si hay ratón, en otro caso cualquier otro valor.}
 Var
  correcto:Integer;
 Begin
  asm
```

```
  mov ax,0000h;
  int 33h;
  mov correcto,ax;
 end;
 raton_inicial:=correcto;
End;

Function rango_raton(xs,ys,xi,yi:byte):boolean;
{Descripción general : Detecta si el ratón se encuentra en una área especifica y si se presiono un
botón del Mouse.
        Variables de entrada:
                xs,ys,xi,yi: Coordenadas de el área a revisar.
        Valor de retorno: verdadero si esta en el área, falso si no lo esta. Esta función es para modo
texto.-}
Var
 x,y:word;
 boton:byte;
Begin
 raton_boton(x,y,boton);
 if ((x>=xs)and(x<=xi)and(y>=ys)and(y<=yi)) Then
  rango_raton:=True
 else
  rango_raton:=False;
End;

Function rango_ratonG(xs,ys,xi,yi:Word;estado:byte):boolean;
{Descripción general : Detecta si el ratón se encuentra en una área especifica y si se presiono un
botón del Mouse.
        Variables de entrada:
                xs,ys,xi,yi: Coordenadas de el área a revisar.
        Valor de retorno: verdadero si esta en el área, falso si no lo esta. Esta función es para modo
gráfico.-}
Var
 x,y:word;
 boton:byte;
Begin
 raton_botonG(x,y,boton,Estado);
if ((x>=xs)and(x<=xi)and(y>=ys)and(y<=yi)) Then
  rango_ratonG:=True
 else
  rango_ratonG:=False;
End;


Function PressBoton:byte;
{Descripción general :Detecta que botón es presionado.
        Variables de entrada:
                Ninguna.
        Valor de retorno: Numero de botón que es presionado
                0: Si no hay presionado
                1: Si es el botón izquierdo
                2: Si es el botón derecho
                3: Si son los dos botones
                4:  Si es el botón del centro (en caso de Ratones con 3 botones)}
```

```
Var
 n_boton:byte;
Begin
  asm
   mov ax,0003h;
   int 33h;
    mov n_boton,bl;
  end;
 PressBoton:=n_boton;
End;

Procedure raton_Activa;
{Descripción general: Visualiza el cursor del ratón en la pantalla.
        Variables de entrada:
                Ninguna.
        Valor de retorno: Ninguno.}
 Begin
  asm
   mov ax,0001h;
   int 33h;
  end;
 End;

 Procedure raton_desactiva;
{Descripción general: Esconde el cursor del ratón.
        Variables de entrada: Ninguna.
        Valor de retorno: Ninguno.}
  Begin
   asm
    mov ax,0002h;
    int 33h;
   end;
  End;


 Procedure raton_boton(Var x,y:Word;boton:Byte);
{Descripción general : Localiza las coordenadas del ratón y el estado de los botones del Mouse.
        Variables de entrada:
                x,y : Variables en donde se almacenan las coordenadas del ratón.
                Botón :Estado del Botón
        Valor de retorno: Coordenadas del ratón y estado de los botones. Para modo Texto.}
 var
  xx,yy:Word;
  Begin
   asm
    mov ax,0003h;
    int 33h;
    mov boton,bl;
    mov xx,cx;
    mov yy,dx;
   End;
   x:=(xx div 8)+1;
   y:=(yy div 8)+1;
  End;
```

```
Procedure raton_botonG(Var x,y:Word;Boton,Estado:byte);
{Descripción general : Localiza las coordenadas del ratón y el estado de los botones del Mouse.
        Variables de entrada
                x,y : Variables en donde se almacenan las coordenadas del ratón.
                Boton,Estado :Estado de los Botones.
        Valor de retorno: Coordenadas del ratón y estado de los botones. Para modo Gráfico.}
 var
  xx,yy:Word;
  Begin
   asm
    mov ax,0003h;
    int 33h;
    mov boton,bl;
    mov xx,cx;
    mov yy,dx;
   End;
   if (Estado=3) Then
    x:=Trunc(xx/2)
   else
    x:=xx;
    y:=yy;
  End;

 Procedure Raton_Posis(x,y:Word);
{Descripción general: Coloca el cursor en una coordenada especifica.
        Variables de entrada
                x,y: Coordenadas de la posición.
        Valor de retorno: Ninguno.}
 var
  xx,yy:Word;
  Begin
   asm
    mov ax,0004h;
    mov cx,x;
    mov dx,y;
    int 33h;
   End;
  End;

 Procedure Pumas_abre(x,y:word);
{Descripción general: Animación del logo de los pumas que abre los ojos.
        Variables de entrada
                Coordenadas de aparición.
        Valor de retorno: Ninguno.}
 Var
  c1,c2:Word;
  Begin
   If x<=getmaxx-28 Then
    for c1:=1 to 28 do {y}
     Begin
      for c2:=1 to 33 do {x}
       Begin
        case cordxy[c2,c1] of
         0  :putpixel(x+c2,y+c1,253);
```

```
        4  :putpixel(x+c2,y+c1,251);
        7  ::
        10 :putpixel(x+c2,y+c1,250);
        255:putpixel(x+c2,y+c1,252);
       End;
      End;
     End;
   End;

  Procedure Pumas_Cierra(x,y:word);
{Descripción general: Animación del logo de los pumas que cierra los ojos.
        Variables de entrada
                Coordenadas de aparición.
        Valor de retorno: Ninguno.}
  Var
   c1,c2:Word;
   Begin
     for c1:=29 to 60 do {y}
      Begin
       for c2:=1 to 33 do {x}
        Begin
         case cordxy[c2,c1] of
          0  :putpixel(x+c2,y+c1-31,253);
          4  :putpixel(x+c2,y+c1-31,251);
          7  ::
          10 :putpixel(x+c2,y+c1-31,250);
          255:putpixel(x+c2,y+c1-31,252);
         End;
        End;
      End;
   End;

  Procedure raton_pumas(x,y:word;Inicial:Boolean);
{Descripción general:  Procedimiento que anima el ratón.
        Variables de entrada
                x,y: Coordenadas de aparición.
                Inicial :Indica si es llamada la función por primera vez.
        Valor de retorno
                Ninguno.}
  Var
   tam,ii,jj,pix,algo,Esc_bytes:word;
   tiempo:byte;
   mascara:pointer;
   puma:File;

  Procedure Graba_Raton;
{Descripción general: Función que graba una área de pantalla a un archivo, y este a su ves es el
gráfico que aparece como cursor del Mouse.
        Variables de entrada
                Ninguna.
        Valor de retorno
                Ninguno.}
  Var
   c1,c2:word;
```

```
 Begin
 Assign(puma,'Mouse.CCR');
 Rewrite(puma,1);
 for c1:=1 to 60 do {y}
  Begin
   for c2:=1 to 33 do {x}
    Begin
    pix:=Getpixel(55+c2,43+c1);
    cordxy[c2,c1]:=pix;
    End;
  End;
 blockwrite(puma,cordxy,sizeof(cordxy),Esc_bytes);
 close(Puma);
 Anima_mouse:=0;
End;

 Procedure Lee_Raton_pumas;
{Descripción general: Realiza la lectura del archivo Mouse.CCR generado en Graba_Raton.
          Variables de entrada
                    Ninguna.
          Valor de retorno: Ninguno.}
 Var
  c1,c2:word;
  Begin
  Assign(puma,'Mouse.CCR');
  Reset(puma,1);
  blockread(puma,cordxy,sizeof(cordxy),Esc_bytes);
  CLOSE(puma);
  Anima_mouse:=0;
 End;

 begin
  If Inicial Then
   Begin
    Lee_Raton_pumas;
   End
  Else
   Begin
     tam:=imagesize(56,45,90,75);
     GetMem(mascara, tam);
     tiempo:=20;
      GetImage(x,y,x+34,y+30,mascara^);
     inc(Anima_mouse);
     If Anima_mouse<tiempo Then
      Begin
       Pumas_abre(x,y);
       Delay(60);
      End
     else
      Begin
       If anima_mouse>=Tiempo*2 Then Anima_mouse:=0;
       Pumas_Cierra(x,y);
       Delay(60);
      End;
```

```
    putimage(x,y,mascara^,0);
   FREEMem(mascara,TAM);
  End;
 End;
End.                                                    {De la unidad }
```

TEXTO.TPU

```
{Unidad Texto: Unidad Realizada para el programa de tesis titulado "Formatos Gráficos".
        Realizado por :
                Cantero Ramirez Carlos
                 Trejo Bonaga Marilu.
        Asesores :
                Manzo Salazar Itsmael
                Monterrosa Escobar Amilcar}
Unit Texto;
Interface
 Uses crt,dos;
 Type
 String10=String[8];
 String3 =String[3];
 Dir_lista=Record
  Nombre:String10;
  Ext:String3;
  Dir:boolean;
 End;
 dirarchv=array [1..512] of string[14];
 Var
  lista_dir:Array [1..512] of Dir_lista;
  Atributo:Word;

  Function ifi(cond:boolean;cond1,cond2:Longint):Longint;
  Function ifs(cond:boolean;cond1,cond2:string):string;
  Function ifb(cond:boolean;cond1,cond2:Boolean):boolean;
  Function agrega_ceros(num:longint;campo:integer):string;
  Function Exis_uni(unidad:Char):String;
  Function quita_espa(cadena:string):string;
  Function int_str(entero:Longint):String;
  Function mayuscula (cadena:string):string;
  Function minuscula (cadena:string):string;
  Function sis_num(Base_orig,base_des,tam:byte;numero:string):string;
  Function Potencia(base,exp:Integer):Longint;
  Function Solo_nombre(archivo:string):String;
  Procedure dir(Var arreglo:dirarchv;filespec:string;atrib,apartir:byte;Var numero:integer);
  Procedure Info_arch(Nombre:String;Var tam:longint; Var Fecha,Hora,atrib:String10);
  Procedure Directorio(Tipo:String;Atributo:byte;var numero:integer);

Implementation

{Descripción general: Funciones if then de una sola línea.
        Ifi:  para valores  de retorno Numérico.
        Ifb: para valores  de retorno Boleados (falso, verdadero)
        Ifs:  para valores  de retorno Strings (cadenas)
```

```
        Variables de entrada:
                Cond:  Condición a evaluar
                Cond1:Valor que es retornado si es verdadero.
                cond2:Valor que es retornado si es falso.
        Valor de retorno: Expresión que fue evaluada según la condición.}
Function ifi(cond:boolean;cond1,cond2:Longint):Longint;
 Begin
  if cond Then ifi:=cond1 else ifi:=Cond2;
 End;

Function ifs(cond:boolean;cond1,cond2:String):string;
 Begin
  if cond Then ifs:=cond1 else ifs:=Cond2;
 End;

Function ifb(cond:boolean;cond1,cond2:Boolean):boolean;
 Begin
  if cond Then ifb:=cond1 else ifb:=Cond2;
 End;

Function agrega_ceros(num:longint;campo:integer):string;
{Descripción general: Agrega ceros convierte un valor numérico a cadena , agregándole ceros
dependiendo del tamaño que se debe ocupar.
        Variables de entrada:
                num: Numero a convertir a cadena
                Campo: el numero de dígitos que abarcara la cadena si el numero es menor a la
                longitud de la cadena los caracteres sobrantes se rellenan con ceros.
        Valor de retorno: Cadena convertida rellenada con ceros.}
 var
  con,cont:integer;
  cadena,cero:string;
 Begin
  cero:='0';
  str(num,cadena);
  con:=campo-length(cadena);
  for cont:=1 to con do
    insert(cero,cadena,1);
  agrega_ceros:=cadena;
 end;


Function Exis_uni(unidad:Char):String;
{Descripción general: Detecta  si se encuentra una unidad de disco.
        Variables de entrada:
                Unidad : Especifica la letra de la unidad.
        Valor de retorno:
                Retorna un '1' si se encuentra la unidad y un '0' so no se encuentra.
                Si Unidad='*' entonces se retorna una cadena con todas las letras de unidades
                existentes.}
Var
 i:byte;
 DriveA:Byte absolute $0040:$0090;
 DriveB:Byte absolute $0040:$0091;
 Existe,Dir_Aux:String;
```

```
Begin
 Existe:='';
 Case upcase(unidad) of
 'A':if (DriveA and 7<>0) Then Existe:='1' else Existe:='0';
 'B':if (DriveB and 7<>0) Then Existe:='1' else Existe:='0';
 'C'..'Z':
   Begin
   getdir(0,Dir_aux);
   {Si-}
    chdir(unidad+':\');
   {Si+}
    if ioresult=0 Then Existe:='1' else Existe:='0 ';
   chdir(Dir_aux);
   End;
 '*':
   Begin
    if (DriveA and 7<>0) Then Existe:='A';
    if (DriveB and 7<>0)Then Existe:=Concat(Existe,'B');
    getdir(0,Dir_aux);
    For i:=67 to 90 do
     Begin
      {Si-}
       chdir(chr(i)+':\');
      {Si+}
       if ioresult=0 Then Existe:=Concat(Existe,chr(i));
     End;
    chdir(Dir_aux);
   End;
  End;
  Exis_uni:=Existe;
 End;


 Function quita_espa(cadena:string):string;
 {Descripción general: Quita los espacios en blanco de una cadena de caracteres.
          Variables de entrada: Cadena: Es la cadena a tratar
          Valor de retorno: Retorna la cadena sin espacios en blanco.}
  var
   longi:byte;
  begin
    repeat
     if copy (cadena,1,1)=' ' then
      delete (cadena,1,1);
    until (copy (cadena,1,1)<>' ');
    longi:=length(cadena);
    repeat
     if copy (cadena,longi,1)=' ' then
      delete (cadena,longi,1);
     longi:=length(cadena);
    until (copy(cadena,longi,1)<>' ');
    quita_espa:=cadena;
   end;
```

```
Function int_str(entero:Longint):String;
{Descripción general: Convierte un numero a cadena.
        Variables de entrada: Entero: Es el  numero a tratar.
        Valor de retorno: Retorna el numero en una cadena.}
 Var
  cadena:String;
Begin
 str(entero,cadena);
 int_str:=quita_espa(cadena);
End;

Function minuscula(cadena:string):string;
{Descripción general: Convierte a minúsculas todas las letras de una cadena.
        Variables de entrada: Cadena: Es la cadena a tratar.
        Valor de retorno: La cadena convertida a minúsculas.}
  Var
  minus:string;
  destino,letra:char;
  letra_mayuscula:set of char;
  i:integer;
Begin
 letra_mayuscula:=['A'..'Z'];
  minus:=' ';
  For i:=1 TO Length(cadena) DO
    Begin
      destino:=cadena[i];
      if (destino in letra_mayuscula) then
        letra:=chr(ord(destino)+32)
         else
        letra:=destino;
      if cadena[i]='¥' then letra:='¤';
       minus:=minus+letra;
    End;
  minuscula:=quita_espa(minus);
End;

 Function mayuscula(cadena:string):string;
{Descripción general: Convierte a mayúsculas  todas las letras de una cadena.
        Variables de entrada: Cadena: Es la cadena a tratar.
        Valor de retorno: La cadena convertida a mayúsculas.}
  var
    mayus:string;
    i:integer;
  Begin
   mayus:=' ';
   For i:=1 TO Length(cadena) DO
    Begin
     if cadena[i]='¤' then cadena[i]:='¥';
     mayus:=mayus+UpCase(cadena[i]);
    End;
   mayuscula:=quita_espa(mayus);
  End;
```

```
Function sis_num(Base_orig,base_des,tam:byte;numero:string):string;
{Descripción general: Convierte un número de una base a otra.
        Variables de entrada:
                Base_orig: Base original del numero.
                Base_des: Base a la que se desea convertir.
                Tam: Tamaño de la cadena, si este tamaño no es igual al numero convertido este es
                rellenado de ceros.
                Numero: Es el número a convertir, este valor debe ser especificado en cadena.
        Valor de retorno: El numero convertido en cadena.}
 Var
  residuo:integer;
  i,j,k:word;
  num,num1:Longint;
  strnum,finalstr:String;
 Begin
  case Base_orig of
   2:
    Begin
     num:=0;num1:=0;k:=0;
     For i:=length(numero) downto 1 do
      begin
       num:=num+(Ifi(numero[i]='0',0,potencia(2,k)));
       inc(k);
      End;
     str(num,finalstr);
    End;
   10:
    Begin
     Val(numero,num,k);
     finalstr:='';
     Repeat
      Residuo:=abs(num mod 2);
      num:=num div 2;
      str(residuo,strnum);
      finalstr:=Concat(strnum,finalstr);
     Until(num=0);
     if tam>length(finalstr) Then
      Begin
       For i:=length(finalstr)+1 to tam do
        finalstr:=Concat('0',finalstr);
      End;
     if copy(numero,1,1)='-' Then finalstr:=Concat('1',finalstr);
    End;
 End;
 Sis_num:=Finalstr;
End;

Function Potencia(base,exp:Integer):Longint;
{Descripción general: Realiza la función de potenciación.
        Variables de entrada:
                Base: Es el numero a tratar.
                Exp : Es el exponente al cual será elevado.
        Valor de retorno: El resultado de la potenciación.}
```

```
 Var
  i:Integer;
  Res:longint;
 Begin
  res:=1;
  For i:=1 to exp do
   res:=res*base;
  Potencia:=res;
 End;

 Function Solo_nombre(archivo:string):String;
{Descripción general: Retorna solo el nombre de un archivo sin ruta ni extención.
        Variables de entrada: Archivo: Es el nombre del archivo incluyendo su ruta y extención.
        Valor de retorno: Solo el nombre del archivo.}
Var
  i:byte;
  nombre:string;
 Begin
  i:=length(archivo);
  nombre:='';
  repeat
   nombre:=concat(archivo[i],nombre);
   dec(i);
  until (archivo[i]=':') or (archivo[i]='\') or (i<=0);
  Solo_nombre:=copy(nombre,1,pos('.',nombre)-1);
 End;

 Procedure dir(Var arreglo:dirarchv;filespec:string;atrib,apartir:byte;Var numero:integer);
{Descripción general: Retorna un arreglo con la lista de archivos y sus caracteristicas.
        Variables de entrada:
                Arreglo: Es el arreglo donde se almacenan los archivos y caracteristicas.
                FileSpec: Es el comodin.
                Atrib: Atributos de búsqueda.
                Apartir: Apartir  de que archivo se empieza a contar.
                Numero: Es el numero de archivos que se encontraron.
        Valor de retorno: Arreglo con archivos.}
 Const
  columns=5;
 Var
  i,j,tamcol,contador:integer;
  aux:string[8];
  reg:registers;
  dta:array[1..43] of byte;

 Begin
  if (apartir<=1) Then apartir:=0;
  contador:=apartir+1;
  if (filespec='*.*') Then apartir:=0;
  contador:=apartir+1;
  tamcol:=80 div columns;
  reg.dx:=ofs(dta);
  reg.ds:=seg(dta);
  reg.ax:=$1A00;
  msdos(reg);
```

```
  filespec:=filespec+chr(0);
  reg.dx:=ofs(filespec[1]);
  reg.ds:=seg(filespec[1]);
  reg.cx:=atrib;
  reg.Ax:=S4E00;
  msdos(reg);
  if lo(reg.ax)<>0 Then
   Begin
    numero:=0;
   End
  Else
   Begin
    j:=31;
    while dta[j]<>0 do
     Begin
      arreglo[contador]:=arreglo[contador]+chr(dta[j]);
      j:=j+1;
     End;
    inc(contador);
   Repeat
   reg.dx:=ofs(dta);
   reg.ds:=seg(dta);
   reg.Ax:=S4F00;
   msdos(reg);
   if lo(reg.ax)=0 Then
    Begin
     j:=31;
     while dta[j]<>0 do
      Begin
       arreglo[contador]:=arreglo[contador]+chr(dta[j]);
       j:=j+1;
      End;
     inc(contador);
    End;
   Until (lo(reg.ax)<>0);
   numero:=contador-1;
   for i:=1 to numero do
    Begin
     aux:=' ';
     if (arreglo[i][1]<>'.')and(length(arreglo[i])>4) Then
      Begin
       for j:=1 to (13-length(arreglo[i])) do
        insert(aux,arreglo[i],pos('.',arreglo[i]));
      End;
    End;
  End;
 End;

Procedure Info_arch(Nombre:String;Var tam:longint; Var Fecha,Hora,atrib:String10);
{Descripción general: Información completa de un archivo.
        Variables de entrada:
                Nombre: Nombre de archivo.
                Tam: Su tamaño encontrado.
                Fecha: Su fecha de creación.
```

Hora: Su hora de creación.
Atrib: Los atributos del archivo.
Valor de retorno: Información del archivo.}

```
 Type
  DatosFich=Record
   Atrib:Byte;
   Hor,fech,bajotam,altotam:word;
   Atr:byte;
  End;
 Var
  reg:Registers;
  dta:Array[1..43] of byte;
  fd:DatosFich absolute dta;
  an,mes,dia,hr,min,sgd:byte;
  st:string[20];
  tama:longint;
 Begin
  reg.dx:=Ofs(dta);
  reg.ds:=Seg(dta);
  reg.ax:=$1A00;
  msdos(reg);
  nombre:=nombre+chr(0);
  reg.dx:=ofs(nombre[1]);
  reg.ds:=seg(nombre[1]);
  reg.cx:=$16;
  reg.ax:=$4E00;
  msdos(Reg);
  If lo(reg.ax)<>0 Then
   Begin
   End
  Else
   Begin
    Tam:=fd.bajotam+(fd.AltoTam SHL 16);
    An:=(hi(fd.fech) shr 1) + 80;
    if an>99 Then an:=an-100;
    mes:=lo(fd.fech) shr 5 +(hi(fd.fech) and $01 shl 3);
    dia:=lo(fd.fech) and $1F;
    fecha:=agrega_ceros(dia,2)+'-';
    fecha:=fecha+agrega_ceros(mes,2)+'-';
    fecha:=fecha+agrega_ceros(an,2);
    hr :=hi(fd.hor)  shr 3;
    min:=lo(fd.hor)  shr 5 +(hi(fd.hor) and $07 shl 3);
    sgd:=(lo(fd.Hor) and $1F) shl 1;
    hora:=agrega_ceros(hr,2)+':';
    hora:=hora+agrega_ceros(min,2)+':';
    hora:=hora+agrega_ceros(sgd,2);
    If fd.Atr and $20=$20 Then Atrib[1]:='A';
    If fd.Atr and $10=$10 Then Atrib[1]:='D';
    If fd.Atr and $02=$02 Then Atrib[1]:='H';
    If fd.Atr and $01=$01 Then Atrib[1]:='R';
    If fd.Atr and $04=$04 Then Atrib[1]:='S';
   End;
 End;
```

```
Procedure Directorio(Tipo:String;Atributo:byte;var numero:integer);
{Descripción general: Retorna y manipula un directorio.
        Variables de entrada:Tipo: Comodin.
                Atributo: Atributo de búsqueda.
                Numero: Numero de archivos encontrados.
        Valor de retorno: Manipula un arreglo.
        Atributo=$00 Archivos unicamente
                $10 Archivos ademas de subdirectorios
                $16 Archivos, subdirectorios además de los ocultos.}
Var
 reg:Registers;
 dta:Array[1..43] of byte;
 i,j:Integer;
 tam:Longint;
 nom:string;
 xdir1,xdir2:String[1];
xNombre Fecha,Hora,atrib:STRING10;
xExt:String3;
 xDir:boolean;
Procedure Limpia_lista;
 Begin
  lista_dir[i].nombre:='';
  lista_dir[i].ext:='';
  lista_dir[i].dir:=False;
 End;
Begin
 reg.dx:=Ofs(dta);
 reg.ds:=Seg(dta);
 reg.ax:=$1A00;
 msdos(reg);
 Tipo:=Tipo+chr(0);
 reg.dx:=ofs(tipo[1]);
 reg.ds:=seg(tipo[1]);
 reg.cx:=Atributo;
 reg.ax:=$4E00;
 msdos(Reg);
 i:=1;
 If lo(reg.ax)<>0 Then
  Begin
  End
 Else
  Begin
   limpia_lista;
   j:=31; nom:='';
   While dta[J]<>0 do
    Begin
     nom:=nom+(chr(dta[j]));
     inc(j);
    End;
   If dta[22] and $10<>0 Then lista_dir[i].dir:=True else lista_dir[i].dir:=False;
   with lista_dir[i] do
    Begin
     Nombre:=ifs((dta[22] and $10<>0),'.',copy(nom,1,pos('.',nom)-1));
     Ext   :=ifs((dta[22] and $10<>0),' ',copy(nom,pos('.',nom)+1,3));
```

```
      End;
      if (lista_dir[i].dir<>True) and (nom[1]<>'.') Then inc(i);
     Repeat
      reg.dx:=Ofs(dta);
      reg.ds:=Seg(dta);
      reg.ax:=$4f00;
      msdos(reg);
      If lo(reg.ax)=0 Then
       Begin
        limpia_lista;
        j:=31; nom:='';
        While dta[J]<>0 do
         Begin
           nom:=nom+(chr(dta[j]));
           inc(j);
         End;
        If dta[22] and $10<>0 Then lista_dir[i].dir:=True else lista_dir[i].dir:=False;
        if pos('.',nom)=0 Then
          nom:=nom+'.';
        with lista_dir[i] do
         Begin
          Nombre:=ifs(nom[1]='.','..',copy(nom,1,pos('.',nom)-1));
          Ext   :=ifs(nom[1]='.',' ',copy(nom,pos('.',nom)+1,3));
         End;
        inc(i);
      End;
    Until (lo(Reg.ax)<>0);
   End;
  numero:=i;
  for i:=1 to numero do
   with lista_dir[i] do
    info_arch(nombre+'.'+ext,tam,fecha,hora,atrib);
  for i:=numero to numero+20 do
   limpia_lista;
   for i:=1 to numero-1 do
    for j:=1 to numero-i do
     begin
      xdir1:=ifs(lista_dir[j].dir,'0','1');
      xdir2:=ifs(lista_dir[j+1].dir,'0','1');
      if (xdir1+lista_dir[j].ext+lista_dir[j].nombre)>=(xdir2+lista_dir[j+1].ext+lista_dir[j+1].nombre) then
       begin
        xnombre                 :=lista_dir[j].nombre;
        xext                    :=lista_dir[j].ext;
        xdir                    :=lista_dir[j].dir;
        lista_dir[j].nombre     :=lista_dir[j+1].nombre;
        lista_dir[j].ext        :=lista_dir[j+1].ext;
        lista_dir[j].dir        :=lista_dir[j+1].dir;
        lista_dir[j+1].nombre :=xnombre;
        lista_dir[j+1].ext      :=xext;
        lista_dir[j+1].dir      :=xdir;
       end;
     end;
 end;
End.
```

APENDICE

D

APÉNDICE D

GUÍA DEL USUARIO

Como Instalar el Programa

El contenido del disquette es el siguiente:

```
A:\>dir

 El volumen de la unidad A no tiene etiqueta
 Directorio de A:\

FORMATOS ARJ        877,090  08-30-97  1:49p FORMATOS.ARJ
INSTALAR EXE         28,656  09-06-97  5:09p INSTALAR.EXE
         2 archivo(s)            905,746 bytes
         0 directorio(s)         551,424 bytes libres

A:\>
```

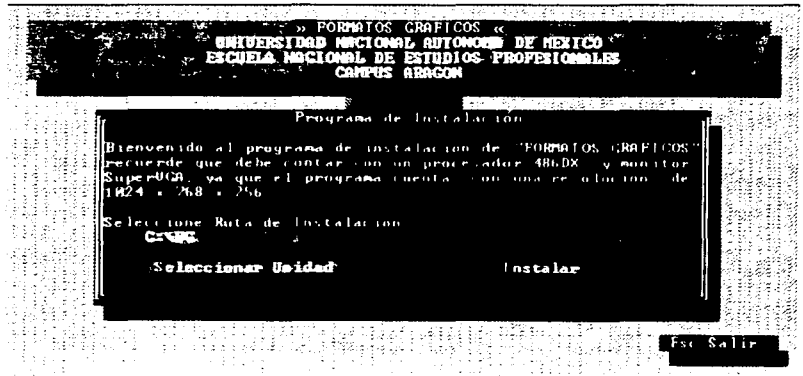
El archivo Instalar.exe es el programa ejecutable que realiza la instalación en el disco duro de los archivos necesarios para el programa Formatos.EXE, que se listan a continuación.

- El archivo mouse.ccr como su nombre lo indica manipula el mouse utilizado en el programa.
- El archivo formatos.exe es el programa ejecutable creado en Pascal.
- El archivo SVGA256.BGI es el para poner el modo gráfico SVGA en 256 colores.
- Los archivos con extensión CHR son los tipos de fuentes utilizadas en el programa.

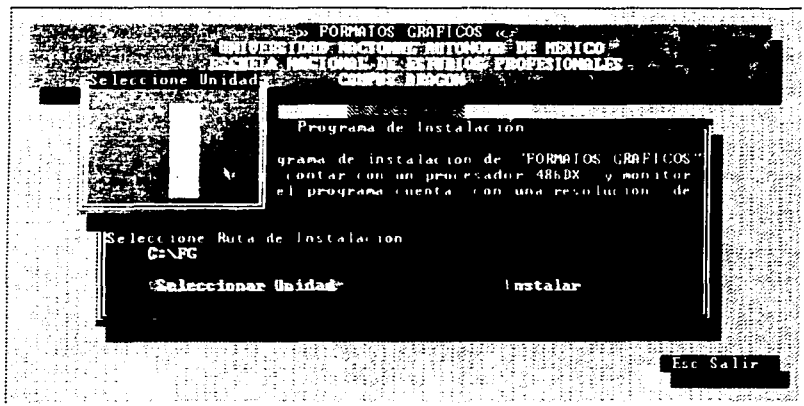
Instalación

La instalación del programa puede ser ejecutada desde la unidad de 3 ½ o desde el disco duro.

El programa de instalación presenta la siguiente pantalla. La cual indica en que directorio se copiaran los archivos.

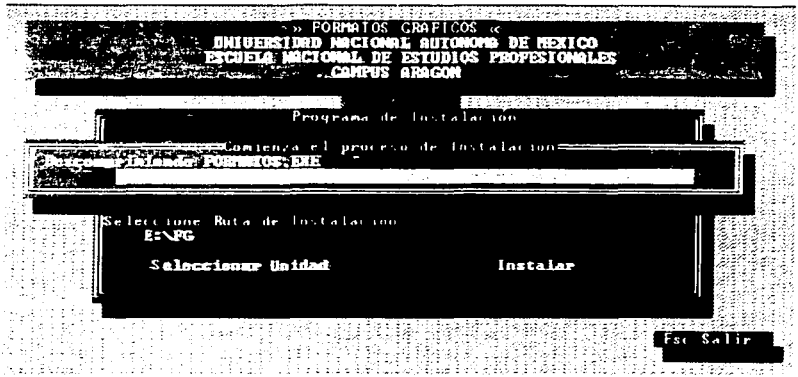


Con el botón Seleccionar Unidad se puede elegir otra unidad de disco duro.

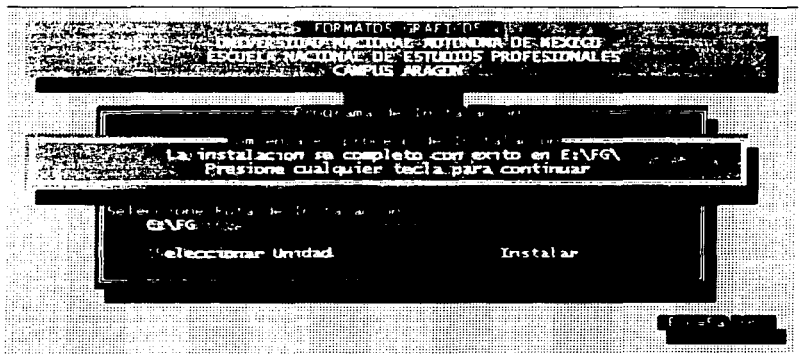


Si desea modificar el nombre de directorio solo de un click en la barra donde se muestra el directorio y presione Enter para confirmar.

Presione el botón instalar para iniciar el proceso de instalación.

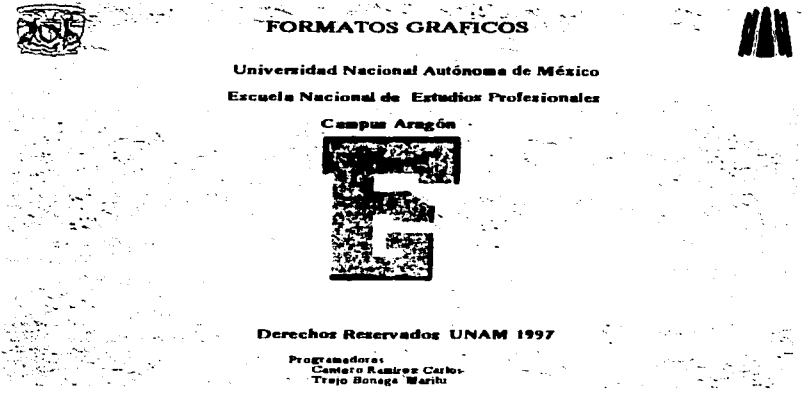


Una vez finalizado el proceso aparece la siguiente pantalla:

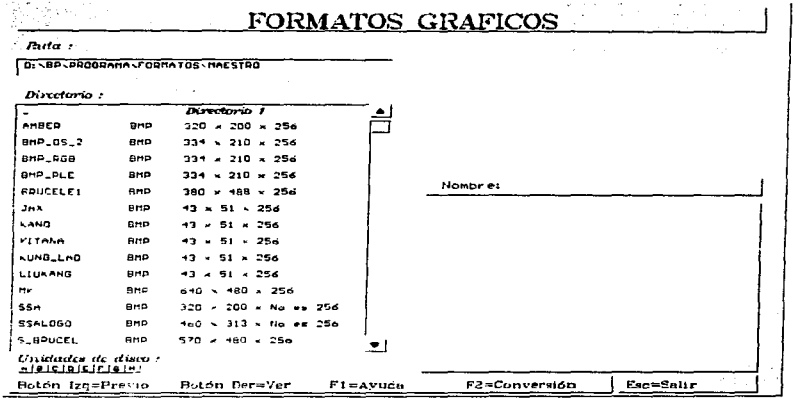


Desde el directorio donde se instalo, ejecute el programa Formatos.exe.

Al ejecutar este comando aparece la siguiente pantalla:



Presione cualquier tecla para continuar.

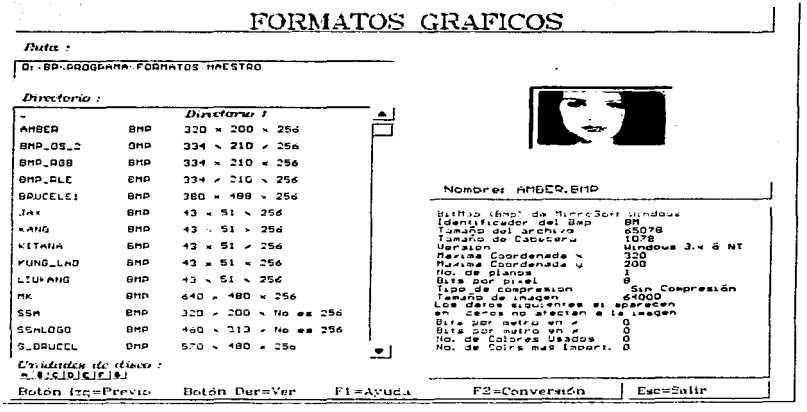


Los componentes que aparecen en la pantalla son:

- Ruta: Path del directorio en que se encuentra actualmente el programa.
- A la derecha del botón de Ruta aparece un recuadro en donde se despliega un previo de la imagen.
- Directorio: Despliega solo los archivos de formatos gráficos que se encuentran en dicho directorio además de los subdirectorios.
- Unidades de disco: el programa verifica las unidades que se encuentran en la computadora y las despliega en forma de botón, para poder accesar a ellas.
- Nombre: una vez elegida una imagen en este recuadro se despliega el nombre de la imagen y en el recuadro abajo de él las propiedades de dicha imagen.

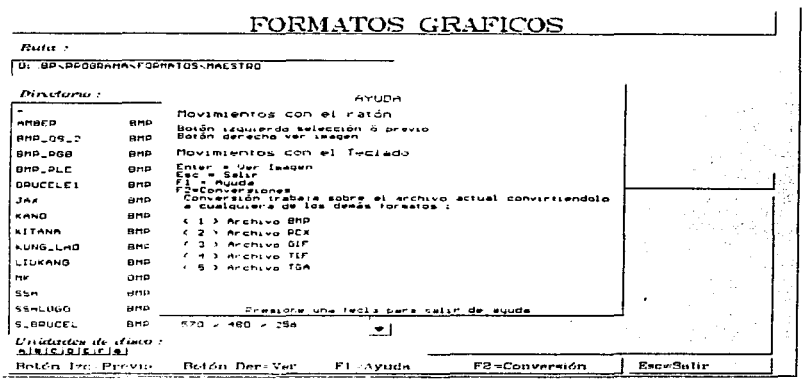
En la parte inferior de la pantalla aparecen dos botones, el más largo de ellos contiene los siguientes comandos, que solo se pueden ejecutar mediante el teclado:

Botón izq = Previo La imagen se despliega en la misma pantalla en la parte superior derecha

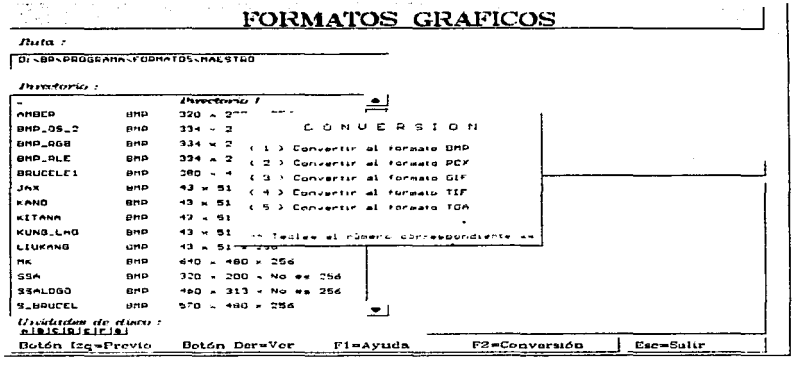


Botón Der = Ver La imagen se despliega en pantalla completa

F1 = Ayuda Despliga un recuadro coninformación de ayuda acerca del programa



F2 = Conversión Despliega un menú de las opciones de conversión entre formatos



En el segundo botón aparece el comando ESC = Salir el cual puede ser ejecutado tanto por el mouse (dando un click en dicho botón) o mediante el teclado para finalizar la ejecución del programa.

APENDICE

E

APÉNDICE E

Software de Apoyo

En el presente trabajo de tesis, fueron de gran utilidad algunos programas que manipulan imágenes, este software fue usado esencialmente en la comparación de resultados obtenidos en el transcurso del trabajo, referidos a la lectura y conversión de imágenes tales como el tamaño en bytes, número de colores, resolución de la imagen y calidad de la conversión. A continuación se lista el software utilizado en este trabajo.

Nombre	**Descripción**	**Marca registrada por:**	**Formatos Soportados (utilizados en la tesis)**
Story Board Live	Creador de historias, basado en imágenes consecutivas.	IBM.	PCX, BMP, TIF,
Sea	Visualizador y convertidor de imágenes. Convierte formatos entre si y es capaz de reducir o aumentar la cantidad de colores de una imagen. Su caracteristica es la velocidad.	Programación de : H. De Laat & B.Wakkee. Trabajo de Arte: R. Gortzen.	PCX, BMP, TIF, TGA.
Paint Shop Pro 4.1 y 4.2	Visualizador, convertidor y creador de imágenes. Es capaz de crear y procesar imágenes, posee herramientas para el manejo de colores entre otros.	JASC Incoporporated	PCX, TIF, GIF, TGA, BMP.
Graphics Work Shop (GWS) para DOS	Visualizador y convertidor de imágenes. Convierte formatos entre si y es capaz de reducir o aumentar la cantidad de colores de una imagen. Su caracteristica es hacer de una imagen un archivo ejecutable.	Graphic Workshop 7.0b Copyright (c) 1989, 1993 Alchemy Mindworks Inc.	PCX, TIF, GIF, TGA, BMP.
Graphics Work Shop (GWS) para Windows	Visualizador y Convertidor de imágenes. Convierte formatos entre si y es capaz de reducir o aumentar la cantidad de colores de una imagen.	Graphic Workshop 7.0b Copyright (c) 1991, 1995 Alchemy Mindworks Inc.	PCX, TIF, GIF, TGA, BMP.

3D Studio V. 4	Creador de objetos para animación.	AutoDesk	PCX, TIF, GIF, TGA, BMP.
Netscape Gold 3	Navegador de Internet. Fue utilizado para obtener algunas características del GIF animado.	Netscape	GIF.
Mosaic.	Navegador de Internet. Fue utilizado para obtener algunas características del GIF animado.	O'Reilly and Associates, Inc	GIF.
Explorer	Navegador de Internet. Fue utilizado para obtener algunas características del GIF animado	MicroSoft	GIF.
Paint de Windows 95	Visualizador y creador de imágenes BMP.	Microsoft	BMP, PCX..
Photo Stacker	Procesador digital de imágenes.	O'Rima Electronics Corp.	PCX, TIF, GIF, TGA, BMP.
Corel Draw 5.0	Animador, procesador y creador de imágenes con una calidad profesional.	Corel	PCX, TIF, GIF, TGA, BMP.

BIBLIOGRAFIA

BIBLIOGRAFIA

GERVAIS,F. ,

"Programación de las tarjetas gráficas CGA, EGA, VGA",
Ed. Addison-Wesley Iberoamericana,
1a. ed.,
España, 1991.

GODFREY, J. Terry,

"Lenguaje Ensamblador para Microcomputadoras IBM",
Ed. Prentice Hall,
1a. ed.,
México D.F., 1991.

HERGET, Douglas,

"Domine Turbo Pascal 5",
Ed. Ventura Ediciones,
2a, ed.,
México D.F., 1991.

KERNIGHAN, Brian W., Ritchie Dennis M.,

"El lenguaje de Programación C",
Ed. Prentice Hall,
2a. ed.,

MICROSOFT Corp.,
"Microsoft GW BASIC, Guía del usuario",
s/Ed.,
s/ed.,
México D.F., 1988.

MORRIS, M. Mano
"Lógica Digital y Diseño de Computadores",
Ed. Prentice Hall,
1a. ed.,
México. D.F., 1982.

MURRAY, James D., Van Ryper William.
"Encyclopedia of Graphics File Format",
De. O'Reilly & Associates, Inc.,
2a. ed.,
USA, 1996.

O'BRIEN, Stephen k., Nameroff Steve,
"Turbo Pascal 7, Manual de referencia",
De. Mc Graw-Hill,
1a. ed,
México, D.F., 1993.

PAPPAS, Chris H., Murray William H.
"Manual de Borland C++ 4.0",
Ed. Mc Graw-Hill,
1a. ed.,
España, 1994.

SUTTY, George, Blair Steve.
"Advanced programmer's guide to Super VGAs",
Ed. Braddy, distribuido por Prentice Hall.
s/ed.,
USA, 1990.