SISTEMA PARA EL PROCESAMIENTO DE IMAGENES DIGITALES IMPLANTADO EN UNA MICROCOMPUTADORA

Tesis que para obtener el grado de Maestro en Ciencias de la Computación, presenta el Licenciado en Ciencias Físico Matemáticas:

Alfredo Cortés Anaya

Unidad Académica de los Ciclos Profesional y de Posgrado del Colegio de Ciencias y Humanidades. Universidad Nacional Autónoma de México.

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

Contenido

Página

Algoritmos

Tablas

Introducción

Este trabajo de tesis presenta al *Sistema para el Procesamiento de Imágenes Digitales* (SPID) implantado en una microcomputadora. Los aspectos más importantes de este sistema son:

1. Efectuar operaciones sobre imágenes, transformando éstas de acuerdo a una función aplicada, (*e.g.* operaciones aritméticas, filtrajes, etc.). Existe una gran variedad de operaciones que se pueden realizar sobre las imágenes, sin embargo, en el SPID se han implantado sólo las más representativas, ya que la finalidad de este trabajo es dar la estructura básica necesaria para desarrollos posteriores, de acuerdo a las necesidades específicas de cada usuario.

2. Transportar datos entre los diferentes dispositivos de almacenamiento de la microcomputadora. El volumen de datos en un sistema de procesamiento de imágenes es considerable, por lo que se deben utilizar algoritmos eficientes para lograr una velocidad óptima en la transferencia. Básicamente, la operación más usual de traslado de datos en el SPID es de la memoria principal al disco y viceversa.

3. Obtener información referente a las imágenes en cuanto a la distribución de sus datos (*e.g.* histograma). Estas operaciones no alteran el contenido de una imagen, pero sirven como base a operaciones del tipo mencionado en el primer párrafo.

El *procesamiento de imágenes digitales* es un conjunto de métodos y técnicas enfocados hacia un objetivo general: Mejorar el aspecto visual (cualitativo) de una imagen, de tal manera que resulte más fácil detectar contrastes en la misma, para una mejor evaluación de los recursos naturales que la imagen represente.

Suponemos aquí que una imagen es un arreglo matricial que se puede considerar como una función de dos variables discretas:

$$\vartheta_{ij}, \quad \{1 \leq i \leq M,\ 1 \leq j \leq N\},$$

en donde M es el número de renglones y N el número de columnas de la imagen, respectivamente. La función anterior es el resultado de un muestreo bidimensional de una escena determinada; este muestreo se logra generando una malla rectangular y equiespaciada sobre la escena, en la que se determinan valores correspondientes a alguna característica física. Generalmente los valores de ϑ están asociados a tonos de gris que representan la

respuesta espectral de la escena, en un $\Delta\lambda$ de longitudes de onda dado. El intervalo de la función se define en el conjunto $Z_{[0...255]}$, (los enteros en el intervalo $[0...255]$). Este conjunto no es más que el intervalo de valores que un número entero de 8 dígitos binarios[1] puede representar. Son 8 *dígitos binarios*, llamados octetos (*bytes*), precisamente los que definen el valor de un elemento de imagen (*pixel*).

El advenimiento de las computadoras digitales y de los dispositivos electrónicos en los que se puede representar y desplegar una imagen han venido a contribuir al desarrollo y al análisis de su contenido, debido a su velocidad de procesamiento, fundamental en el tratamiento de imágenes en donde la cantidad de información a procesar puede ser muy grande.

Podemos mencionar algunos ejemplos del campo de acción del procesamiento de imágenes a la Geofísica y la Geología, en donde esta técnica puede jugar un papel de mucha importancia en el análisis, evaluación y explotación de los recursos naturales de nuestro planeta. Esta información se obtiene por medio de satélites artificiales utilizando mecanismos ópticos y electrónicos muy elaborados.

En general, las imágenes se pueden clasificar en cuatro clases representativas [Pavl82]:

Clase 1 Todas aquellas imágenes formadas por una escala completa de grises (256) y las imágenes de color. Las imágenes de satélite pertenecen a esta clase. Su almacenamiento se lleva a cabo en forma matricial, renglón a renglón como una cadena de longitud MXN.

Clase 2 Todas las imágenes formadas por dos colores (blanco y negro), por lo que se les llama *imágenes binarias*. Para su almacenamiento se utiliza la misma técnica que en el párrafo anterior, excepto que ahora son sólo dos valores.

Clase 3 Las imágenes compuestas de líneas y curvas continuas; por ejemplo un diagrama que muestre las curvas de nivel de una función bidimensional.

Clase 4 Estas imágenes constan sólo de puntos aislados y muy separados entre sí, respecto al espacio que ocupan. No llenan el plano que los contiene como lo hace una imagen de clase 1, por lo que para su almacenamiento se requiere el valor de las coordenadas de cada uno de sus puntos.

El sistema SPID utiliza imágenes de la clase 1, ya que el monitor puede desplegar 256 colores diferentes y las imágenes estan codificadas utilizando este mismo número de valores. Una de las características que hacen de SPID un sistema amigable es el uso de menús dado que éstos son de gran utilidad en la programación de sistemas por la interacción que debe tener el usuario con la computadora. Un *menú* es una manera cómoda de presentarle al usuario las alternativas de solución a su problema, con el mínimo esfuerzo de su parte y de una manera interactiva; la característica principal de todo manejador de menús es la de

[1] En inglés se les llama *bits* de su abreviación *binary digit*.

presentar al usuario un conjunto de alternativas de las cuales pueda elegir las necesarias para la solución de su problema. Un manejador de menús se implanta eficientemente con el uso de autómatas de estados finitos, cuyas características de ejecución e implantación se definen en el capítulo 1. El despliegue de imágenes es de importancia fundamental en un sistema interactivo, ya que éste debe ser tan rápido como sea posible.

Capítulo 1

Preliminares

Todo libro o artículo técnico requiere de un marco matemático de referencia con base en el cual se pueda desarrollar la teoría pertinente; esta tesis no es la excepción y es en este capítulo donde se definirán los antecedentes matemáticos relacionados con el diseño e implantación del SPID.

1.1 Requerimientos electrónicos

El equipo de cómputo básico que SPID requiere es una tarjeta gráfica de alta resolución, 256K octetos de memoria principal y un monitor a color. La impresora es opcional, pero si se cuenta con ella, entonces se podrán obtener copias de imágenes en papel.

Para el despliegue de una imagen se utiliza una tarjeta de alta resolución conocida como HLGE (de su nombre en inglés *High Level Graphics Engine*) que significa *Tarjeta gráfica de alta resolución.* Se considera que el despliegue de una imagen es una de las partes más importantes de un sistema interactivo, y la HLGE lo realiza de manera satisfactoria en cuanto a velocidad y resolución se refiere ya que el despliegue de una imagen de 480 × 640 elementos se lleva a cabo aproximadamente en 2 segundos.

La tarjeta de alta resolución permite el despliegue de imágenes haciendo un uso mínimo del microprocesador de la computadora. La resolución espacial que proporciona la tarjeta es de *480 renglones* × *640 columnas* mostrando 256 colores a la vez, que se pueden escoger de 4096 opciones. Tiene un emulador de gráficas a color que permite a la computadora ejecutar programas de gráficas como si se tratara de la tarjeta CGA (*Color Graphics Adaptor*), utilizada por la mayoría de las microcomputadoras IBM–AT con una resolución de 200 × 320 elementos a 4 colores. La HLGE tiene su propio microprocesador de 32 *bits* que procesa los comandos que se le envían desde el CPU de la microcomputadora.

El procesador de comandos utiliza una sección de memoria fija del sistema cuya longitud es de 1024 *octetos*, área de donde toma los comandos que el sistema le envía.

La HLGE tiene un área interna de memoria de 307200 *octetos* que mapea, a través de un filtro, a la pantalla de despliegue; este filtro define la intensidad en cada uno de 3 haces de luz (ROJO, VERDE Y AZUL), los cuales, al mezclarse, definen un color. La intensidad se le proporciona como un arreglo de 3 valores en el intervalo [0..15], que corresponden a cada color primario. Para el despliegue de imágenes en blanco y negro, se logran únicamente 16 tonos diferentes que varían entre el blanco y el negro, obtenidos de mezclar iguales intensidades de luz en los tres haces: ROJO, VERDE Y AZUL (el negro se obtiene con la tercia (0,0,0) y el blanco con (15,15,15)).

Es importante mencionar que SPID hace los despliegues por bloques de datos y que en ningún momento almacena en memoria principal del sistema más de 64K *octetos*. De la misma manera, para cualquier tipo de operación se trabaja con bloques pequeños de memoria que se pueden manejar fácilmente en la microcomputadora, pero sin causar trabajo excesivo a la unidad lectora de disco, ya que esto haría que el sistema fuese lento. Es decir, se escogen los bloques de datos que optimicen el espacio en memoria y la lectura en disco.

Hay algunas operaciones matemáticas que gastan mucho tiempo de microprocesador. En estos casos, es necesario el uso de un coprocesador matemático. El sistema funciona con un coprocesador matemático 8087/80287 para el procesamiento de imágenes, principalmente en las operaciones de área y vectoriales. Sin el coprocesador el SPID no funciona.

1.2 Obtención de imágenes

Existen al menos tres métodos para obtener una imagen digitizada: El primer método consiste en generar la imagen con dispositivos electrónicos integrados en los satélites del tipo LANDSAT y SPOT, equipo con el que generan imágenes digitizadas de escenas correspondientes a medios diferentes, utilizando medios ópticos.

El segundo método consta de un sistema digitizador que se compone de una cámara de televisión y de un rastreador. Este último genera un archivo de datos; con la imagen captada por la cámara, asignando valores de acuerdo a la intensidad luminosa en la escena.

Finalmente las *imágenes sintéticas* se forman a partir de funciones matemáticas. Por ejemplo una función de dos variables discretas para que genere un conjunto de puntos que definen, a su vez, una superficie. Este conjnto de puntos se selecciona de tal manera que forman una malla espaciada, finita, en cuyo intervalo se encuentra definida la función. Si los valores de la superficie se mapean en el conjunto $Z_{[0...255]}$, se obtiene una imagen.

1.3 Nomenclatura y notación matemática

Se define el conjunto $\Im$ cuyos elementos son todas aquellas imágenes que maneja SPID, de la forma I^{MN}, en donde MN es la dimensión de la imagen: M renglones × N columnas. Se utiliza la variable v_{ij} para identificar al elemento de una imagen que se encuentra en el renglón i y en la columna j.

Ventana es el área de una imagen que se puede ver, y *el "enrollamiento"*, tiene como función mover esta ventana a lo largo y ancho de la imagen permitiendo al usuario la visualización completa de la misma, cuando sus dimensiones son mayores de lo que un monitor puede mostrar en la pantalla.

La notación $\lfloor x \rfloor$ define al entero obtenido a partir de x y que se encuentre más cercano a 0, es decir, el entero máximo menor o igual a x si $x > 0$; mientras que si $x < 0$, representa el entero mínimo mayor o igual a x.

$Z_{[x_1 \ldots x_2]}$ es el conjunto de enteros $\{x_1, \ldots, x_2\}$. En este trabajo se hace continua referencia al conjunto $Z_{[0 \ldots 255]}$; el conjunto de enteros que se puede representar con un *octeto*. Se hace poca referencia a los conjuntos, únicamente en lo que se refiere a su uso en la representación de autómatas de estados finitos analizados en la siguiente sección. Un conjunto se escribirá con letras caligráficas, por ejemplo: $\mathcal{A}, \mathcal{B}, \mathcal{C}$, etc. La cardinalidad de un conjunto $\mathcal{A}$ se denotará como $|\mathcal{A}|$.

El *esfuerzo computacional* de un algoritmo se define como la cantidad de operaciones aritméticas (suma, resta, división y multiplicación), operaciones de acceso a disco (búsqueda de datos y direccionamiento del apuntador a los archivos) y operaciones de movimiento de datos en memoria principal y de ésta a la memoria integrada de la IILGE. Al esfuerzo computacional de un algoritmo F se denota como $C(n)$, en donde n es el tamaño del mismo. El *tamaño* n de un algoritmo se define de acuerdo al contexto del mismo, por ejemplo, si se trata de un producto escalar, el tamaño n es la cantidad de elementos en un vector; si se trata de una multiplicación matricial, entonces el tamaño n del problema es la dimensión de una matriz, etc.

El *orden de un algoritmo* se define de la siguiente manera: Sea un algoritmo F y sean $C(n)$ y $f(n)$ dos funciones de $N \rightarrow N$, donde n es el tamaño del problema que se quiere resolver y $C(n)$ es el esfuerzo computacional del algoritmo F; se dice que $C(n) = O(f(n))$ si existen constantes $c > 0$ y $k \geq 0$ tales que $|C(n)| \leq c|f(n)| \quad \forall \; n \geq k$. Esta definición indica que dada una función $C(n)$, existe otra función $f(n)$ que multiplicada por una constante acota a $C(n)$ y $O(f(n))$ es el orden del algoritmo F.

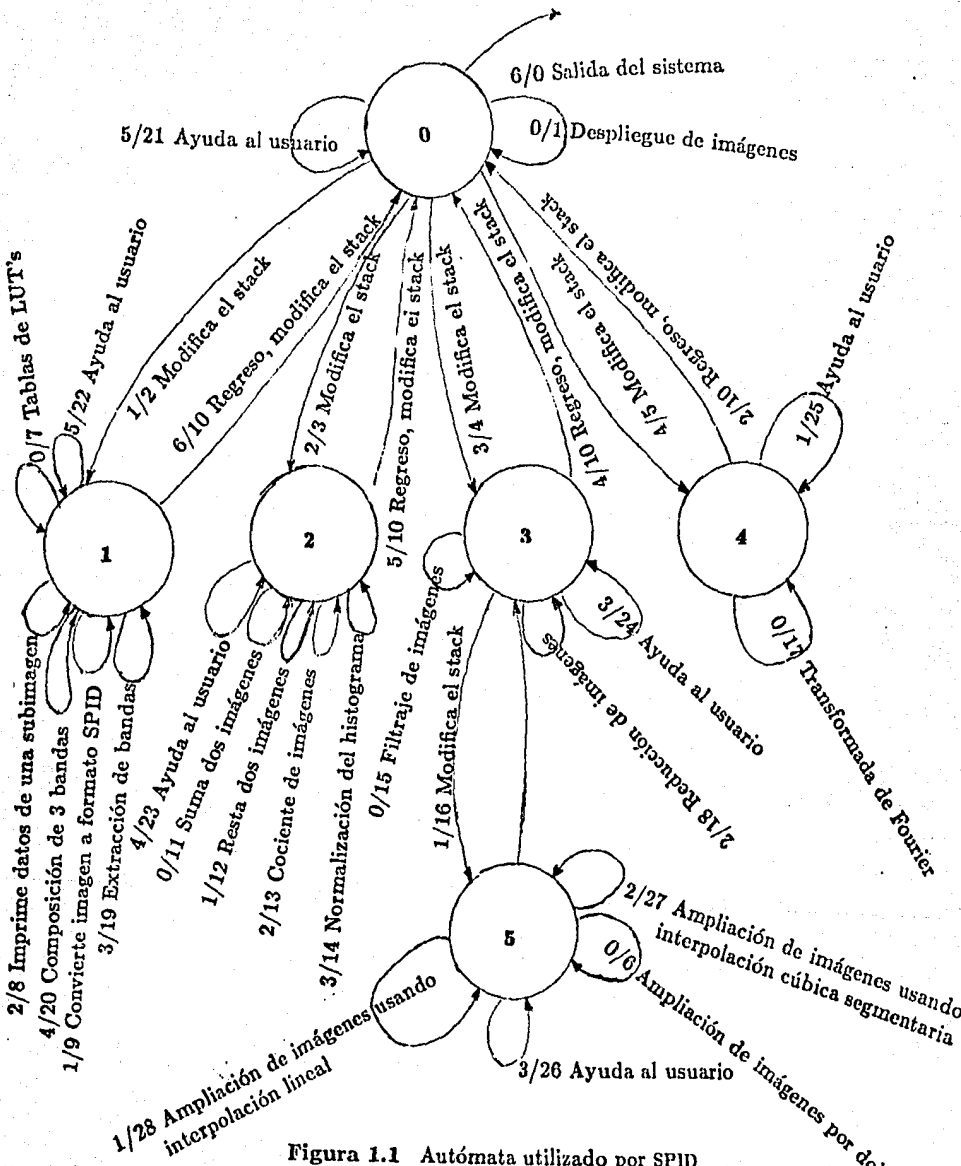


Figura 1.1 Autómata utilizado por SPID

1.4 Autómatas de estados finitos

Debido a que el núcleo de SPID está gobernado por un *autómata o máquina de estados finitos*, en esta sección se definirán algunos conceptos matemáticos referentes a estos conceptos. Al final de esta sección se presenta una manera de implantarlos en una computadora como parte integral de un sistema interactivo.

La teoría de autómatas se puede definir como el *"análisis en el comportamiento dinámico de los sistemas de información cuyos parámetros son discretos"* [Boot 67], estos parámetros se pueden dividir en tres categorías: parámetros de entrada, parámetros que indican el estado del sistema y parámetros de salida. Los *parámetros de entrada* consisten en un conjunto de elementos que definen el comportamiento del autómata y que el sistema recibe produciendo un cambio en su comportamiento. Los *parámetros de salida* representan el comportamiento *observable* del sistema y constan principalmente de funciones y procedimientos que el sistema ejecuta en respuesta a los parámetros de entrada.

Cabe señalar que el concepto de teoría de autómatas es una abstracción matemática relacionada con la invención y estudio de máquinas idealizadas a las que se les conoce con el nombre de *autómatas*. La abstracción está relacionada con el manejo de información en dispositivos electrónicos tales como una computadora, la cual es precisamente nuestra herramienta de trabajo en el desarrollo de este sistema. En esta sección se mencionará la importancia y la facilidad del uso de una abstracción matemática de este estilo.

Como se puede ver en la Fig. 1.1, el autómata utilizado por el sistema SPID consta de 6 estados, tiene 7 parámetros de entrada y genera 28 funciones de salida. Para la representación de este autómata en un sistema se requiere usar una estructura de datos conveniente que represente lo mejor posible dentro de la computadora el diagrama de la Fig. 1.1. Veremos en seguida que la mejor representación es a través de matrices.

La representación matricial de un autómata requiere de dos matrices: M^t y M^s; en la matriz M^t se almacena el estado siguiente mientras que en M^s se almacena un índice que apunta a una función de salida. El elemento M^t_{ij} es el estado al que debe pasar el autómata cuando se encuentre en el estado e_i y reciba un símbolo de entrada a_j y el elemento M^s_{ij} es el apuntador a una función de salida correspondiente al cambio de estado. Si el autómata que se representa consta de $|\mathcal{E}|$ estados y $|\mathcal{A}|$ símbolos de entrada, entonces la matriz de transición tendrá respectivamente $|\mathcal{E}|$ renglones y $|\mathcal{A}|$ columnas. En la Tabla 1.1 se muestran las matrices que utiliza SPID.

Matriz de transición M^t							Matriz de salidas M^s						
0	1	2	3	4	0	6	1	2	3	4	5	21	0
1	1	1	1	1	1	0	7	9	8	19	20	22	10
2	2	2	2	2	0	0	11	12	13	14	23	10	10
3	5	3	3	0	3	0	15	16	18	24	10	0	10
4	4	0	4	4	4	0	17	25	10	0	0	0	10
5	5	5	5	3	5	3	6	28	27	26	10	0	10

Tabla 1.1 Matriz de transición, M^t y matriz de salidas M^s que utiliza SPID.

Los valores numéricos que aparecen en la matriz de salidas M^s tienen asignada una función específica dada por la Tabla 1.2:

A manera de ejemplo y considerando el diagrama de la Fig. 1.1 y las Tablas 1.1 y 1.2, supóngase que SPID se encuentra en el estado 2, (renglón 2 de la matriz de transición M^t) y recibe de entrada un 2 (columna 2 de M^t), generará una salida igual al número que se encuentra en la misma posición en la matriz de salidas M^s (renglón 2, columna 2), que es 13 (los renglones y columnas de las matrices se numeran desde 0), viendo la Tabla 1.2, en la entrada 13 dice " Cociente de imágenes", que no es más que una rutina que se ejecutará enseguida.

Salida	Función realizada
1	Despliegue de imágenes
2	Modifica el stack
3	Modifica el stack
4	Modifica el stack
5	Modifica el stack
6	Ampliación de imágenes por doblamiento
7	Tabla de luts
8	Imprime una subimagen
9	Convierte una imagen a formato SPID
10	Modifica el stack
11	Suma dos imágenes
12	Resta dos imágenes
13	Cociente de imágenes
14	Normalización del histograma
15	Filtraje de imágenes
16	Modifica el stack
17	Transformada de Fourier
18	Reducción de imágenes
19	Extracción de bandas
20	Composición de 3 bandas
21	Ayuda al usuario
22	Ayuda al usuario
23	Ayuda al usuario
24	Ayuda al usuario
25	Ayuda al usuario
26	Ayuda al usuario
27	Ampliación de imágenes usando interpolación cúbica segmentaria
28	Ampliación de imágenes usando interpolación lineal

Tabla 1.2 Funciones de salida dadas por la matriz M^s de la tabla anterior.

Capítulo 2

Almacenamiento y despliegue de imágenes

Una de las partes clave en un sistema para el procesamiento de imágenes digitales es su preparación y manejo; básicamente su almacenamiento pero, más importante, su despliegue; éste debe ser lo más rápido posible debido a las características interactivas con las que debe trabajar cualquier sistema como SPID.

En este capítulo se desglosa la estructura interna de una imagen; se analiza su despliegue en un monitor de alta resolución y se revisan algunos aspectos de SPID que, propiamente, no son procesos sobre imágenes sino más bien de ayuda al usuario. Por ejemplo, el uso de ventanas para la visualización por secciones de una imagen y del enrollamiento; con lo que el usuario puede ver toda la imagen, por muy grande que ésta sea, utilizando una ventana que se puede deslizar a lo largo y ancho de la imagen almacenada en disco.

2.1 Formato y almacenamiento de imágenes

La estructura de datos que se emplee en la representación de una imagen depende mucho del dispositivo de almacenamiento utilizado, que puede ser cinta, disco o memoria principal. La estructura es importante en la implantación, eficiencia y tiempo de respuesta de los algoritmos y programas de acceso a estos datos, por lo que resulta de crucial importancia la selección de una estructura adecuada de almacenamiento de imágenes. En la representación de una imagen se debe contemplar la posibilidad de almacenar imágenes multiespectrales, facilidad de accesos a disco directo y aleatorio, ya que esto último permitirá trabajar con subimágenes.

La estructura de datos en disco utilizada por SPID para almacenar una imagen es muy sencilla y consta de dos partes fundamentales: en la primera parte se encuentran los

datos que definen la imagen, mientras que en la segunda parte se encuentran algunos datos relacionados con sus características.

Al principio, se encuentra el histograma de la imagen, seguida por dos datos muy importantes: *los valores mínimo y máximo dentro de la imagen y sus dimensiones—alto×ancho.*

El histograma necesita 1K *octetos* de espacio ya que cada valor se almacena en 4 octetos y son 256 valores por almacenar. En seguida vienen 6 octetos con la siguiente información: En el primero se encuentra el valor del elemento mínimo dentro de la imagen, mientras que en el segundo se almacena el valor máximo; los cuatro restantes almacenan las dimensiones de la imagen 2 para los renglones y 2 para las columnas, éstos representan números enteros sin signo, con lo que se pueden manejar (procesar) imágenes cuyas dimensiones sean de 65536 × 65536 elementos, lo cual es suficiente para una imagen común, ya que sólo para su almacenamiento se requeriría del orden de 429 *Mb*,[1] las imágenes de satélite LANDSAT 4 y 5 pueden ser de hasta 6000 × 6000. En la Fig. 2.1 se muestra gráficamente la estructura de una imagen en formato SPID.

MATRIZ DE DATOS DE LA IMAGEN				
Histograma de la imagen 1024 *octetos*	Elemento Mínimo	Elemento Máximo	Número de Renglones	Número de Columnas

Figura 2.1 Estructura de una imagen en el SPID

La estructura matricial que se le ha dado a las imágenes, obedece a que la mayoría de las operaciones que realiza SPID se simplifican con este esquema por tratarse de operaciones puntuales y de área.

Hasta ahora se ha hecho referencia a una imagen matricial, sin embargo, los datos en disco se encuentran almacenados en un arreglo lineal donde se han concatenado los renglones de la imagen, de arriba hacia abajo, uno a continuación del otro. El valor de un elemento $\vartheta_{i,j} \in I^{MN}$ (renglón i, columna j) se encuentra en el arreglo lineal en la posición:

$$\vartheta_{i,j} = (i-1) * M + j \qquad \forall \;\; 1 \leq i \leq M \;\; y \;\; \forall \;\; 1 \leq j \leq N. \tag{2.1}$$

La posicion inicial del histograma de una imagen se calculará con la fórmula:

[1] Una microcomputadora con disco duro maneja del orden de 40*Mb*.

$$Base_h = x - 1030 \tag{2.2}$$

en donde el valor 1030 surge de sumar 1024 (el tamaño del histograma) y 6 (la información que define la imagen).

La posición de los elementos mínimo y máximo de una imagen se calculan con las fórmulas siguientes:

$$\begin{aligned} \vartheta_{min} &= val(x-6) \\ \vartheta_{max} &= val(x-5), \end{aligned} \tag{2.3}$$

las dimensiones M y N se calculan con las fórmulas:

$$\begin{aligned} M &= val(x-4) * 256 + val(x-3) \\ N &= val(x-2) * 256 + val(x), \end{aligned} \tag{2.4}$$

en donde x es la longitud total del arreglo lineal que compone la imagen y la función $val(a)$ regresa el valor de la localidad a en el arreglo. Se definen M y N en función de x; x se calcula a partir de la longitud de un archivo en disco. La función LeeLongImg, explicada en el Capítulo 5, se encarga de actualizar el valor de x de las fórmulas anteriores.

La estructura de almacenamiento de los datos posibilita una visualización más clara de la imagen, permitiendo a la vez el acceso a una subimagen cualesquiera, dando las coordenadas de su esquina superior izquierda y las dimensiones de la misma, tarea que realiza la función LeeSubImg, analizada más a fondo en el Capítulo 5. Esta posibilidad de accesar subimágenes de manera rápida es una de las principales características del sistema SPID.

Las imágenes que SPID puede manejar no tienen restricciones severas, aun cuando sus dimensiones sean grandes. Para su manejo, se dividen en bloques cuyo tamaño no exceda el de la memoria principal, de tal manera que se puedan manejar en este espacio procesando los bloques uno a uno, permaneciendo el resto en disco, hasta procesar la imagen completa. Con este esquema se requiere que SPID sea capaz de leer y escribir de disco a memoria y viceversa, tan rápido como sea posible. Como se ha mencionado, los algoritmos que se presentan en este trabajo intentan tener un máximo de eficiencia En el Apéndice A se listan las subrutinas utilizadas en el manejo de entrada/salida de datos, ahí mismo se resalta la importancia de la implantación de las funciones de lectura y escritura de datos a disco, que es una de las tareas más frecuentes y que consumen más tiempo.

2.2 Despliegue rápido de una imagen

Con el uso de la tarjeta de alta resolución, el monitor puede desplegar la sección de una imagen de 480 renglones por 640 columnas. La HLGE contiene memoria interna dedicada exclusivamente al monitor y de la cual hace un mapeo directo al mismo. El despliegue de un punto se traduce a la escritura de datos en una localidad de esta memoria, cada punto ocupa un *octeto* de memoria, por lo que se pueden desplegar 256 colores a la vez (de un conjunto de 4096). El acceso a la memoria de la tarjeta no es directo, se puede hacer mediante un comando de control, sin embargo, no es con este método que conviene desplegar una imagen, ya que tarda del orden de 15 segundos en desplegar toda la pantalla, y en este tipo de programas es un tiempo demasiado largo.

En SPID se programa el DMA *(Direct Memory Access)* del sistema para hacer transferencias directas entre la memoria de la tarjeta y la memoria de la computadora y viceversa, con lo cual se logra el despliegue de una imagen en fracciones de segundo.

Cada vez que se desea ver una sección de imagen, ésta se lleva de disco a memoria principal y de ahí se despliega a través del DMA. Este esquema puede parecer tardado pero el trabajar con bloques de memoria menores o iguales a 64K *octetos* facilita mucho las transferencias, sobre todo si se considera que el canal del DMA transfiere información a una velocidad aproximada de 400 *Kb/s* por segundo.

En el algoritmo 2.1 se muestra cómo se lleva a cabo el despliegue de una imagen. En este algoritmo hay algunos pasos generales que requieren mayor explicación, tales como los que incluyen los programas: DESPLEGAR_IMAGEN, DESPLEGAR_HISTOGRAMA y AMPLIAR_IMAGEN. Los algoritmos 2.2 al 2.4 correspondientes a estos programas se muestran a continuación.

Debido a que en esta tesis se pretende también hacer una evaluación aproximada del *esfuerzo o costo computacional de los algoritmos*, que puede llamarse también *complejidad de un algoritmo* [2] se evaluará la complejidad de los algoritmos de despliegue. La complejidad de éstos es proporcional al tamaño de la imagen en el caso del despliegue de una imagen, mientras que el despliegue del histograma y del cambio de tablas LUT's es constante.

[2] Normalmente este parámetro se da en función de la cantidad de operaciones aritméticas que los algoritmos realizan, y en este trabajo así se hizo. Sin embargo, no se debe negar la posibilidad de realizar la evaluación de un algoritmo tomando en consideración otro tipo de operaciones, como puede ser el movimiento de datos entre memoria principal y secundaria.

```
1   Inicio del algoritmo
2     Leer un caracter, mientras éste no sea ESC
      y el estado actual no sea 0, hacer :
3       si el caracter leído es 'A' o 'a' hacer :
4         Crear el medio ambiente de AMPLIACION
5         Leer a través del DMA la subimagen a ampliar
6       fin_si
7       si el caracter leído es 'D' o 'd' hacer :
8         Crear el medio ambiente y el menú de DESPLIEGUE
9         Ejecutar el programa DESPLEGAR_IMAGEN
10      fin_si
11      si el caracter leído es 'H' o 'h' hacer :
12        Crear el medio ambiente del HISTOGRAMA
13        Ejecutar el programa DESPLEGAR_HISTOGRAMA
14      fin_si
15      si el caracter leído es 'L' o 'l' hacer :
16        Cambiar la tabla de LUT's y avanzar el índice
          en el archivo TBL-LUTS.DAT
17      fin_si
18      si el caracter leído es 'CR' hacer :
19        Ejecutar el programa AMPLIA_IMAGEN y desplegar
          la imagen ampliada, con opción a seguir ampliando
20      fin_si
21      si fue una función especial (↑, ↗, →, ↘, ↓, ↙, ←, ↖), entonces
        mover la ventana de ampliación en la dirección indicada
22      fin_si
23    fin_mientras
24  Fin del algoritmo
```

Algoritmo 2.1 Despliegue de imágenes

Cabe hacer notar que el sistema utiliza dos segmentos de memoria de 64K *octetos* para el almacenamiento de datos temporales. A estos segmentos se les ha dado el nombre de SEG_IMG1 y SEG_IMG2, respectivamente. Las transferencias de datos, utilizando el DMA sólo se pueden hacer en límites físicos de memoria que empiecen en segmentos de 64K, no se pueden hacer en otras particiones tal como lo hacen los segmentos de datos o de código. Esta restricción es algo impositiva en el uso del DMA, ya que con esto el sistema se ve restringido a utilizar conjuntamente los dos segmentos mencionados anteriormente.

La forma en que se determina la partición de 64K es tomando en consideración que los dos segmentos lógicos se encuentran contiguos en memoria y ambos límites cruzan una partición de 64K como se muestra en la figura 2.2.

Segmento $n-2$	Segmento	$n-1$	Segmento	n	Segmento	$n+1$	Segmento $n+2$
		SEG_IMG1		SEG_IMG2			

Figura 2.2 Uso físico y lógico de segmentos en memoria

La memoria física es con la que cuenta toda computadora como circuitos integrados, encima de la cual se crean particiones a las cuales se les denomina memoria lógica, ésta última es con la que se entienden los programas de aplicación como lo es SPID. La figura 2.2 se debe interpretar de la siguiente manera: Existen dos capas que se sobreponen: La que forman los segmentos físicos (Segmento $n-2$, ..., Segmento $n+2$) y la que forman los segmentos lógicos (SEG_IMG1 y SEG_IMG2) .

Se puede apreciar en la misma figura que los segmentos lógicos se fijan en tres segmentos físicos, a saber, $n-1, n, n+1$, de los cuales, sólo el segundo se cubre totalmente.

Cada segmento físico de 64K *octetos* se identifica con un número entre 0 y 15 (Fh, hexadecimal), para encontrar el número $n \in Z_{[0...15]}$ se lleva a cabo la siguiente operación:

$$n = \text{SEG\_IMG2}\&\&F000h >> Ch \tag{2.5}$$

en donde && denota el AND lógico entre bits de los números involucrados, mientras que >> denota el corrimiento de bits a la derecha. El AND hace ceros todos los bits del número SEG_IMG2, excepto los 4 más significativos, mientras que el corrimiento los mueve a la derecha ajustando n al conjunto $Z_{[0...15]}$.

La función BaseImg, explicada en el capítulo 5 utiliza la fórmula anterior para calcular la partición a 64K y de esta manera se puede utilizar en la transferencia de datos, vía el DMA.

El despliegue de una subimagen se lleva a cabo leyendo bloques de 64K *octetos* de disco a memoria a través del DMA, y de memoria se pasan a la tarjeta de despliegue, la cual a través de un mapeo a su memoria y utilizando la tabla de LUT's definida en el momento, despliega la imagen. El algoritmo de despliegue de imágenes es el 2.2.

En el paso 3 del Algoritmo 2.2 se utiliza la función LeeSubImg analizada en el apéndice A. En casi todos los algoritmos se utiliza esta función por la propiedad de SPID de hacer todas las operaciones por subimagenes.

```
1   Inicio del algoritmo
2     mientras falten líneas por desplegar, hacer :
3       Leer una subimagen a memoria
4       Escribir la subimagen a la tarjeta, vía el DMA
5       Actualizar el número de líneas leídas y desplegadas
6     fin_mientras
7   Fin del algoritmo
```

Algoritmo 2.2 Despliegue de una subimagen

La complejidad del algoritmo depende únicamente de la longitud de la imagen desplegada y esto se debe a la forma en que se leen los datos.

Analizaremos someramente el funcionamiento de LeeSubImg: Lee de disco el número de líneas que puedan almacenarse en un bloque de memorias de 64K, resulta claro que entre más ancha sea una imagen menos renglones se harán, la fórmula usada es:

$$\text{LineasALeer} = \frac{64K}{N} \quad \propto \quad \frac{1}{N} \tag{2.6}$$

en donde N es el ancho de la imagen. Para desplegar toda la subimagen se repite el proceso de lectura de bloques x veces, en donde:

$$x = \left\lceil \frac{\text{ALTOPAN}}{\text{LineasALeer}} \right\rceil \quad \propto \quad N \tag{2.7}$$

de donde se puede ver que el orden del algoritmo de despliegue es lineal ($O(N)$).

En el paso 13 del algoritmo 2.1 se hace mención al despliegue del histograma, este despliegue es bastante rápido ya que la información se encuentra disponible en cada imagen (ver la Fig. 2.1). La altura máxima de cada valor del histograma es igual a ALTOHISTO, que en el archivo SPID.DEF, mostrado en el apéndice B, tiene un valor de 415 puntos. Cada valor del histograma se escala a este máximo y se traza una línea vertical de la altura correspondiente al escalamiento dado por la fórmula:

$$\text{Altura de cada valor} = \frac{\text{Valor máximo en el histograma}}{\text{ALTOHISTO}} \text{ Valor a desplegar} \tag{2.8}$$

La *Altura de cada valor* está dado en puntos en la pantalla de despliegue, el *valor máximo del histograma* se calcula a partir de todos sus valores y el *valor a desplegar* varía desde el 0 hasta el 255, para cada imagen. El algoritmo se muestra a continuación:

```
1   Inicio del algoritmo
2     Leer los valores del histograma
3     Calcular el elemento máximo (Max_Histo)
4     Calcular el cociente: Cociente = Max_Histo / ALTOHISTO
5     para cada elemento en el histograma, hacer :
6       Calcular el número de puntos cuya altura se desplegará
        para este valor, con la fórmula 2.6 anterior,
        pero sin calcular cada vez el cociente del paso 4
7       Desplegar la línea con la altura calculada
8     fin_para
9   Fin del algoritmo
```

Algoritmo 2.3 Despliegue del histograma

En el paso 2 del algoritmo anterior se utiliza la función LeeHisto, esta función carga en memoria el histograma de la imagen que es un arreglo de 256 valores. En el paso 3 se encuentra el elemento máximo de estos valores y se hace un mapeo (paso 6) de cada valor del histograma para ser desplegada una línea proporcional al valor en cuestión. La complejidad en cada uno de los pasos de este algoritmo es constante, por lo que la complejidad de todo el algoritmo es también constante.

La ampliación de imágenes se lleva a cabo en el modo de despliegue o a traves de los menús. El primer caso se puede considerar que es *proceso en línea*, mientras que el segundo como *proceso por bloques*. Se le llama en línea porque al momento de elegir una ampliación se lleva a cabo en el momento y se despliega inmediatamente el resultado, sin almacenar el resultado en disco; mientras que en el proceso por bloques la imagen obtenida se almacena en disco, sin desplegarse.

En el caso de ampliación por bloques, existen tres maneras de hacerlo, como se verá en este capítulo: Por doblamiento de elementos, por interpolación lineal y por interpolación cúbica. Por otro lado la ampliación en línea se lleva a cabo utilizando interpolación lineal, se escogió ésta ya que es más rápida que la cúbica y da más resolución que el doblamiento de elementos.

La complejidad del algoritmo 2.4, de ampliación en línea, es constante ya que depende de constantes, el área que se amplía en línea es de (ALTOSUB×ANCHOSUB) y se amplia a 2·ALTOSUB×2·ANCHOSUB.

```
1   Inicio del algoritmo
2     Crear un archivo temporal TEMPO.BAK
3     Leer la subimagen a ampliar y escribirla en TEMPO.BAK
4     Crear un segundo archivo temporal TEMPO2.BAK
5     mientras falten líneas por ampliar, hacer
6       Leer una subimagen de TEMPO.BAK
7         para cada renglón de la subimagen, hacer
8           para cada par de elementos ϑ_k, ϑ_{k+1}, hacer
                ϑ_interpolado = (ϑ_k + ϑ_{k+1}) / 2
9         fin_para
10        Repetir los dos pasos anteriores, pero considerando
          que las columnas son renglones y los renglones
          columnas, es decir, interpolar por columnas
11      fin_para
12      Escribir la subimagen ampliada en el archivo TEMPO2.BAK
13    fin_mientras
14    Finalmente, desplegar el archivo TEMPO2.BAK
15    Borrar los archivos TEMPO.BAK y TEMPO2.BAK
16  Fin del algoritmo
```

Algoritmo 2.4 Ampliación de imágenes en línea

2.3 Ventaneo y enrollamiento

En la sección 2.1 se mencionó la estructura de una imagen. Se dijo que al final de la misma existe un par de datos que definen sus dimensiones (alto × ancho). La estrategia de guardar esta información permite que una imagen sea de cualesquiera dimensiones, es decir, no tiene restricciones[3] en cuanto a tamaño, más que la capacidad de almacenamiento en disco.

Sin embargo, la pantalla de despliegue es de dimensiones finitas (y en SPID se aparta un área de la pantalla para despliegue, de dimensiones 460 × 530. Esta ventana sirve para visualizar una imagen cuyas dimensiones sean mayores que las desplegadas. En el capítulo 1 ya se definió lo que significa una ventana y el enrollamiento. En el despliegue de una imagen cuyo tamaño es mayor que el de la ventana de visualización se utiliza el enrollamiento. Inicialmente, la ventana se posiciona en la parte superior izquierda de la

[3] Excepto la que ya se mencionó en el capítulo anterior. Ya que el alto y el ancho de una imagen varían en el conjunto $Z_{[0,\ldots,65535]}$, la restricción en el tamaño de una imagen es que no mida más de 65535 de ancho ni de alto. Dimensión suficiente en la mayoría de los casos.

imagen y a continuación la despliega, el usuario tiene las teclas ↑, ↗, →, ↘, ↓, ↙, ←, y ↖ a su disposición, con estas puede realizar el enrollamiento que le permitirá visualizar la imagen por secciones.

Capítulo 3

Funciones de utilería

Existe un conjunto de operaciones implantadas en SPID que no modifican el contenido de las imágenes, sin embargo, son importantes ya que ayudan al usuario en la preparación de imagénes para un proceso posterior. Por ejemplo, se requiere a veces imprimir en papel los datos numéricos de una subimagen, para analizarlos cuantitativamente. En este capítulo explicaremos cada una de las funciones de utilería.

3.1 Ayuda al usuario

Existe una ayuda en línea que proporciona SPID al usuario. Esta ayuda indica brevemente el nivel en que se encuentra el usuario dentro del sistema al momento de invocarla.

Para cada conjunto de operaciones se explica brevemente su significado, no se explica a detalle cada operación, sino más bien a nivel de su clasificación. En el caso de SPID ésta se hace de acuerdo a cada una de las secciones del capítulo 4: Operaciones puntuales, de área, vectoriales, funciones de utilería, etc.

Esta función considera el estado en que se encuentra el autómata, el cual hace las veces de índice de acceso a la tabla de ventanas, éstas, a su vez, contienen los textos instructivos al usuario.

3.2 Manejo de errores

En el uso de todo sistema existe la posibilidad de cometer errores en su uso; por lo tanto se debe tener un programa que los maneje de la manera más eficiente posible: Tal es el caso de SPID que puede recuperarse de todos los errores cometidos por el usuario. La Tabla

3.1 muestra los errores que se manejan en SPID.

Tabla 3.1 Errores posibles de ser detectados en el uso de SPID

Número de error	Descripción del error
0	SPID requiere un coprocesador matemático
1	Número de función inválida
2	No se encuentra la imagen
3	No se encuentra el camino
4	Ya no hay espacio para abrir imágenes
5	No se le permite el acceso a la imagen
6	Manejador de archivo inválido
7	Se destruyeron bloques de control en memoria
8	Espacio insuficiente en memoria
9	Direcciones en memoria inválidas
10	Medio ambiente inválido (ver SET)
11	Formato inválido
12	Código de acceso inválido
13	Datos inválidos
14	Error fatal: El espacio en disco es insuficiente
15	Especificación de *drive* inválida
16	Intento de borrar este directorio
17	No es el mismo dispositivo
18	No se encontraron más archivos
20	La imagen < > ya existe
21	El formato de la imagen es inválido
22	Error en el formato de un número real
23	Error en el formato de un número entero
24	Las dimensiones de las imágenes no son iguales

Los errores 1—18 son los que regresa el sistema operativo MS-DOS al ejecutar una acción errónea en cualquiera de sus funciones. Hay una excepción en este conjunto de errores: El número 14 no es de la versión 2.0 o menor del sistema operativo, este error se incorporó hasta versiones posteriores, pero con otro número, sin embargo, aquí se incluye en este número ya que no se utiliza normalmente. El resto de los errores son propios del sistema SPID.

A grandes rasgos, los errores se clasifican en dos grupos: Los no críticos y los que sí lo son. Un error crítico termina la ejecución del sistema ya que no puede continuar, mientras que los errores no críticos sólo avisan al usuario sobre una falta que haya cometido y se restituye el funcionamiento. A los errores dentro del sistema se les conoce con los identificadores CRITICO y NO_CRITICO. Los errores críticos son únicamente dos: La falta

de un coprocesador matemático en el equipo de cómputo utilizado (error 0) y espacio insuficiente en disco (error 14), éste se genera al ejecutar un programa que obtenga como resultado nuevas imágenes para las cuales ya no hay espacio en disco.

El despliegue de errores se hace en el momento en que se cometen y se utiliza una ventana del tamaño de despliegue, se muestra el tiempo necesario para que sea leído por el usuario. Si es crítico, se termina la ejecución del sistema, de otro modo se restablece su funcionamiento, desapareciendo la ventana.

3.3 Definición de tablas de LUT's

Este programa de utilería es dependiente de la tarjeta gráfica IILGE que se utilizó en el desarrollo de SPID. La pantalla puede mostrar simultáneamente 256 colores diferentes, sin embargo, éstos se pueden escoger de un total de 4096, definidos de la siguiente manera: Existen 3 colores primarios: rojo, verde y azul; cada uno con 16 intensidades que varían entre 0 y 15, de aquí que los colores diferentes que se pueden producir son: (16 rojos)×(16 verdes)×(16 azules)=4096.

Una tabla de 256 colores se construye usando las intensidades en cada uno de los colores primarios y formando tercias, a cada una de las cuales se le asigna un índice, este índice varia entre 0 y 255. Por ejemplo, la tercia (0,0,0) define el color negro; la tercia (0,0,15) define el color azul más intenso, y a cada una de estas tercias se le asigna un índice, cuando un elemento de la imagen a desplegar sea igual a éste, entonces el elemento aparecerá en la pantalla en el color dado por la tercia que lo define.

La definición de una tabla conlleva a la falsa coloración de una imagen, de tal forma que tablas diferentes colorean imágenes de manera diferente. En la Tabla 3.2 se muestra un ejemplo. Se definen todas las tercias asociadas a los 16 tonos de gris que se pueden obtener. la escala es creciente, de tal manera que el 0 corresponde al negro y el 255 al blanco más intenso. Se repite cada tercia para intervalos de 16 valores. En la Tabla 3.2 se muestra una definición de 256 colores.

En el sistema SPID se pueden definir hasta 15 tablas diferentes, las cuales se almacenan en una archivo llamado TBL-LUTS.DAT. La tarjeta de despliegue IILGE tiene 4 tablas predefinidas internamente. Cada imagen desplegada puede conmutar de tablas -lo cual cambia su coloración- utilizando una opción en el menú de despliegue llamada CAMBIA LUT.

Esta opción carga una a una las tablas definidas por el usuario, además de las que tiene integradas la tarjeta, pudiendo colorear una imagen de hasta 19 formas diferentes; será tarea del usuario definir tablas que se apeguen a la coloración más conveniente de una imagen.

Tabla 3.2 Ejemplo de una tabla de LUT's, que define 16 tonos de gris

Indice	Tercia	Indice	Tercia	Indice	Tercia	Indice	Tercia
0	(0,0,0)	64	(4,4,4)	128	(8,8,8)	192	(12,12,12)
1	(0,0,0)	65	(4,4,4)	129	(8,8,8)	193	(12,12,12)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
15	(0,0,0)	79	(4,4,4)	143	(8,8,8)	207	(12,12,12)
16	(1,1,1)	80	(5,5,5)	144	(9,9,9)	208	(13,13,13)
17	(1,1,1)	81	(5,5,5)	145	(9,9,9)	209	(13,13,13)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
31	(1,1,1)	95	(5,5,5)	159	(9,9,9)	223	(13,13,13)
32	(2,2,2)	96	(6,6,6)	160	(10,10,10)	224	(14,14,14)
33	(2,2,2)	97	(6,6,6)	161	(10,10,10)	225	(14,14,14)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
47	(2,2,2)	111	(6,6,6)	175	(10,10,10)	239	(14,14,14)
48	(3,3,3)	112	(7,7,7)	176	(11,11,11)	240	(15,15,15)
49	(3,3,3)	113	(7,7,7)	177	(11,11,11)	241	(15,15,15)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
63	(3,3,3)	127	(7,7,7)	191	(11,11,11)	255	(15,15,15)

El algoritmo 3.1 muestra a grandes rasgos el programa utilizado para la generación de tablas de LUT's.

3.4 Impresión de valores de una subimagen

Por ahora, el sistema no cuenta con una interfaz ni dispositivo para la impresión de imágenes, sin embargo, a veces es necesario conocer el valor numérico de sus elementos. Existe una opción en SPID en el menú de funciones de utilería que imprime el valor de los elementos de una subimagen. Estos valores se encuentran en el conjunto $Z_{[0...255]}$.

El usuario proporciona únicamente el nombre de la imagen, las coordenadas del punto superior izquierdo de la subimagen a imprimir, así como la altura y el ancho de la misma. Debe estar conectada una impresora para que esta función opere correctamente.

```
1   Inicio del algoritmo
2     Abrir el archivo TBL-LUTS.DAT
3     Muestra el medio ambiente de la generación de tablas de LUT's
4     Desplegar los valores de
      Indice, Intensidad del rojo, verde y azul módulo 16
5     Leer un caracter, mientras éste no sea ESC, hacer :
6       Colorear la region que muestra el color,
        de acuerdo a la tercia definida
7       si el caracter leído es '←', entonces mover el cursor a la izquierda,
8       si el cursor se encontraba en el primer cuadro,
        entonces moverlo al último
9       si el caracter leído es '→', entonces mover el cursor a la derecha,
10      si el cursor se encontraba en el último cuadro,
        entonces moverlo al primero
11      si el caracter leído es '↑', entonces incrementar el contador
        del cuadro en que se encuentre el cursor, si es el índice,
        entonces se hace mod 256 de otro modo se hace mod 16
12      si el caracter leído es '↓', entonces restar uno al contador
        del cuadro en que se encuentre el cursor, si es el índice,
        entonces se hace mod 256 de otro modo se hace mod 16
13    fin_mientras
14    Escribir en memoria la tabla
15    Al final de la definición, escribir la tabla en el archivo TBL-LUTS.DAT
      con un índice dado por el usuario, entre 1 y 14
16    Cerrar el archivo TBL-LUTS.DAT
17  Fin del algoritmo
```

Algoritmo 3.1 Definición de tablas de LUT's

3.5 Conversión de una matriz de datos a formato SPID

La Figura 2.1 muestra el formato de una imagen, tal como lo maneja SPID. La función que explicaremos en esta sección toma como datos de entrada la matriz que define la imagen; calcula los elementos mínimo y máximo dentro de la imagen; calcula el histograma de la misma y almacena toda esta información al final de la imagen, tal como se muestra en la Figura 2.1.

El usuario proporciona el número de renglones y columnas que definen la imagen, si éstos no coinciden con el archivo, la imagen obtenida no será la correspondiente a la matriz de datos original. Este fenómeno se reflejará en el despliegue de la imagen, ya que para esto SPID lee las dimensiones al final de la misma y, si no coinciden con la matriz, el despliegue no será correcto.

En el siguiente algoritmo se muestra a grandes rasgos la conversión de un archivo de datos al formato requerido por el sistema SPID.

```
1   Inicio del algoritmo
2     Inicializar el histograma a ceros
3     Actualizar las dimensiones de la imagen a crear
4     mientras falten líneas por procesar, hacer :
5       Leer un bloque de datos del archivo de entrada
6       Calcular los elementos mínimo y máximo de este bloque de datos
7       Actualizar el histograma con este bloque de datos
8       Escribir el bloque de datos en un archivo de salida
9     fin_mientras
10    Escribir en el archivo de salida los valores de los elementos
      mínimo y máximo encontrados además
      del histograma y las dimensiones de la imagen
11  Fin del algoritmo
```

Algoritmo 3.2 Conversión de imágenes a formato SPID

La complejidad del algoritmo anterior es fácil de estimar. En los pasos 2 y 3 se requiere de un tiempo constante. El paso 6 es el que más tiempo consume y es proporcional a N·*LíneasALeer*, [Horo78], en donde N es el ancho de la imagen y "*LíneasALeer*" es el número de líneas a leer en un segmento de memoria en el i–ésimo paso. Este proceso se repite x veces, en donde:

$$x = \left\lceil \frac{M}{LíneasALeer} \right\rceil \tag{3.1}$$

De las consideraciones anteriores, la complejidad de conversión de formatos de imágenes es de $O(MN)$. A lo largo del trabajo veremos que casi todos los algoritmos son de esta complejidad, la diferencia en tiempo que existe entre la ejecución de uno y otro está dado por una constante, lo cual quiere decir que aun cuando su orden de complejidad sea el mismo, no lo es su tiempo de ejecución.

3.6 Composición de tres bandas

Hasta ahora sólo se ha hablado y tratado con imágenes monoespectrales, sin embargo, existen 2 operaciones de movimiento de datos en SPID que manejan imágenes multiespectrales, como las provenientes de satélites de percepción remota.

La primera de ellas —analizada en esta sección— es la de composición de 3 bandas y la segunda —analizada en la siguiente sección— es la de extracción de una banda.

Las imágenes multiespectrales del satélite SPOT están compuestas de 3 bandas, la primera banda, con un intervalo espectral de $0.59\mu m$, se encuentra en el verde-amarillo (visible); la segunda banda, entre 0.61 y $0.68\mu m$, se encuentra en el naranja-rojo (visible) y finalmente; la tercera banda, entre 0.79 y de $0.89\mu m$, se encuentra en el infrarojo cercano (invisible).

Las imágenes multiespectrales LANDSAT se componen de 6 bandas con diferentes intervalos espectrales. Para cada tipo de imagen se requiere, en algunos casos, el despliegue simultáneo de 3 bandas. Antes del despliegue se crea una imagen compuesta por las tres bandas, en el caso de una imagen SPOT se ocupan las tres bandas, pero en el caso de una imagen LANDSAT existen $C_3^6 = 20$ combinaciones posibles de componer una imagen, considerando 3 bandas a la vez, será decisión del usuario cuales bandas considerar.

La composición se lleva a cabo de la siguienta manera: Cada banda tiene su intervalo de valores en el conjunto $Z_{[0...255]}$ por lo que se requieren $256 \times 256 \times 256 = 256^3 = 16777216$ colores diferentes para desplegar simultaneamente 3 bandas. Sin embargo, la tarjeta gráfica HLGE —y muchas otras en el mercado— no alcanza a mostrar esta variedad de colores simultáneamente. Para evitar esta limitación, se considera que el conjunto de valores que puede tomar cada elemento de cada banda se reduce drásticamente, de tal manera que la composición se ajuste a los 256 colores disponibles. Para la primera banda se considera que sus elementos pueden tomar sólo 4 valores, para la segunda y tercera bandas sus elementos pueden tomar sólo 8 valores. La razón de esto es la siguiente: Cada punto en la pantalla requiere de 8 *unidades*, éstas se deben distribuir entre las tres bandas a componer: 2 para la primera, 3 para la segunda y tercera. Con esta restricción en el intervalo de valores se obtienen $2^2 \times 2^3 \times 2^3 = 2^8 = 256$ colores diferentes.

La operación aplicada a cada banda a componer es la siguiente:

$$\text{Elemento obtenido} = \frac{\text{Elemento original}}{x} \tag{3.2}$$

en donde x es 64 para la primera banda y 32 para la segunda. Esta fórmula ajusta y clasifica la primera banda en 4 valores y la segunda y tercera en 8 valores. De acuerdo a lo anterior, el valor de cada elemento de la imagen compuesta es de la forma $1_r1_r2_v2_v2_v3_a3_a3_a$, en donde el número representa la posición de cada banda y el subíndice el color asignado (r, rojo; v, verde y a, azul).

```
1    Inicio del algoritmo
2      mientras faltan líneas por componer, hacer
3        Leer un bloque de datos de la imagen multiespectral
4        Inicializar tres apuntadores al inicio de cada renglón
         que se va a componer, segun la banda: Rojo, verde y azul
5        para cada elemento de los tres renglones hacer
6          Elemento generado = Rojo&&C0h|| Verde&&3Ch|| Azul&&07h
           en donde && denota el AND lógico,
           el caracter '||' denota el OR lógico
           y la 'h' denota números en notación hexadecimal
7          Avanzar los apuntadores
8        fin_para
9        Desechar los demás renglones del bloque de datos leídos
10       Escribir el bloque de datos compuesto
11     fin_mientras
12   Fin del algoritmo
```

Algoritmo 3.3 Composición de 3 bandas de una imagen multiespectral

El Algoritmo 3.3 muestra a grandes rasgos las operaciones llevadas a cabo en la composición de tres bandas.

3.7 Extracción de bandas

En el formato de las imágenes multiespectrales SPOT y LANDSAT se combinan todas la bandas renglón por renglón; como lo muestra la Figura 3.1.

renglón 1	banda 1
renglón 1	banda 2
⋮	⋮
renglón 1	banda *N*
renglón 2	banda 1
renglón 2	banda 2
⋮	⋮
renglón 2	banda *N*
⋮	⋮
renglón M	banda 1
renglón M	banda 2
⋮	⋮
renglón M	banda *N*

Figura 3.1 Formato de una imagen multiespectral

La función de extracción de una banda determinada lee todos los datos de la imagen y desecha aquéllos que no corresponden a esa banda. De esta manera se puede desplegar una imagen multiespectral banda por banda, como lo muestra el Algoritmo 3.4.

```
1    Inicio del algoritmo
2      mientras faltenlíneas por extraer, hacer
3        Leer un bloque de datos de la imagen multiespectral
4        Inicializar dos apuntadores: Uno al inicio del primer renglón
         a extraer y el otro al inicio del bloque a generar
5        para cada renglón a extraer hacer
6          Mover el renglón del bloque leído al bloque generado
7        fin_para
8        Desechar los demás renglones del bloque de datos leídos
9        Escribir el bloque de datos extraído
10     fin_mientras
11   Fin del algoritmo
```

Algoritmo 3.4 Extracción de una banda en una imagen multiespectral

Capítulo 4

Operaciones sobre imágenes

La parte central de un sistema para el procesamiento de imágenes digitales es el conjunto de algoritmos y programas que realizan transformaciones de las mismas. En este capítulo se proporcionan los aspectos teóricos y la implantación de algunos de estos algoritmos.

Estas operaciones pueden ser de una inmensa variedad que dependerá de las aplicaciones que se le pretenda dar al sistema. Inicialmente SPID se construyó con un núcleo básico de funciones, pero ha sido diseñado para agregarle todas aquellas que una aplicación específica requiera. La clasificación hecha en este capítulo se tomó de [Daws87] y agrupa en tres categorías todas las operaciones comunes: puntuales, de área y vectoriales.

4.1 Operaciones puntuales

La característica principal de las operaciones puntuales es que se llevan a cabo punto a punto. El proceso de transformación en una imagen para cada uno de sus elementos depende únicamente del valor mismo, es decir, las operaciones puntuales se definen de la siguiente manera:

$$\vartheta'_{ij} = f(\vartheta_{ij}), \qquad \begin{matrix} 1 \leq i \leq M \\ 1 \leq j \leq N \end{matrix} \tag{4.1}$$

en donde ϑ'_{ij} es el elemento obtenido después de aplicar la función f puntualmente a cada elemento de la imagen.

En esta sección se analizarán las operaciones aritméticas (suma, resta y cociente de imágenes o bandas) como parte fundamental de las operaciones puntuales. La normalización del histograma se considera como una operación puntual, aunque el cálculo del

histograma sea en sí una operación estadística.

4.1.1 Normalización de las operaciones aritméticas

Así como existen las operaciones algebraicas en los enteros, así tambien las hay en las operaciones entre imágenes. A continuación se definirán estas operaciones, las cuales se llevan a cabo de manera puntual, a nivel de elementos de imagen. Sean $\vartheta_{1_{ij}} \in I_1$, $\vartheta_{2_{ij}} \in I_2$, $\vartheta_{3_{ij}} \in I_3$ tres elementos de tres imágenes diferentes, se definen las operaciones aritméticas entre imágenes como:

$$\vartheta_{3_{i,j}} = \begin{cases} \vartheta_{1_{ij}} + \vartheta_{2_{ij}} & \text{SUMA} \\ \vartheta_{1_{ij}} - \vartheta_{2_{ij}} & \text{RESTA} \\ \vartheta_{1_{ij}} * \vartheta_{2_{ij}} & \text{PRODUCTO} \\ \vartheta_{1_{ij}} / (\vartheta_{2_{ij}} + 1) & \text{DIVISION} \end{cases} \quad \forall\, 1 \leq i \leq \text{M} \quad \forall\, 1 \leq j \leq \text{N} \tag{4.2}$$

Las operaciones anteriores pueden generar elementos de una nueva imagen cuyo resultado se salga del intervalo de valores permitido. Por ejemplo, la suma $\vartheta_{3_{i,j}} = \vartheta_{1_{ij}} + \vartheta_{2_{ij}}$ no estará en el conjunto $Z_{[0...255]}$, si $\vartheta_{3_{i,j}} > 255$.

Sean $\vartheta_{max} = \max(\vartheta_{3_{i,j}})$ y $\vartheta_{min} = \min(\vartheta_{3_{i,j}})$ $\{\ \forall\ 1 \leq i \leq \text{M}$ y $\forall\ 1 \leq j \leq \text{N}\}$ los valores definidos por la fórmula 4.1. Si existe un valor $\vartheta \in I_3$ que no se encuentre en el conjunto $Z_{[0...255]}$, entonces hay dos formas de ajustar ϑ al conjunto; la primera, haciendo:

$$\vartheta' = \begin{cases} 0 & \text{si } \vartheta < 0 \\ 255 & \text{si } \vartheta > 255, \end{cases} \tag{4.3}$$

y la segunda:

$$\vartheta' = \left\lfloor 255 \frac{\vartheta - \vartheta_{min}}{\vartheta_{max} - \vartheta_{min}} \right\rfloor, \tag{4.4}$$

en donde ϑ' es el valor ajustado. La primera transformación simplemente *corta* los valores fuera del alcance de su límite más cercano, esta operación realiza la función Ajusta, analizada en el capítulo 5. La segunda transformación distribuye de manera más uniforme los valores fuera de rango ya que mapea lincalmente el intervalo $[\vartheta_{min}, \vartheta_{max}]$ en el conjunto $Z_{[0...255]}$.

4.1.2 Programación de las operaciones aritméticas

Las operaciones aritméticas entre imágenes se llevan a cabo de igual manera que la mayor parte de operaciones programadas en el sistema: Procesando una subimagen que se pueda

manejar en memoria principal y repitiendo el proceso hasta que se termine con toda la imagen.

Para cada subimagen la operación se lleva a cabo renglón a renglón y elemento por elemento. Posicionalmente se opera con los dos elementos de las dos imágenes operando según la fórmula (4.2).

Si las dimensiones de las dos imágenes no coinciden, entonces genera un mensaje de error al usuario (ver la Tabla 3.1, error 24). El sistema continúa su ejecución preguntando nuevamente por otro par de imágenes. El ususario proporciona únicamente el nombre de las dos imágenes operandos y la imagen resultado. El algoritmo general para la suma de imágenes es el 4.1.

```
1   Inicio del algoritmo
2     Leer los nombres de las imágenes de entrada y de salida
3     Leer las dimensiones de las dos imágenes de entrada
4     si las dimensiones coinciden hacer
5       mientras falten líneas por procesar hacer
6         Leer secuencialmente un bloque de datos
7         para cada renglón hacer
8           para cada columna hacer
9             Aplicar la fórmula (4.2)
10          fin_para
11        fin_para
12        Escribir el bloque de datos obtenido
13        Actualizar el número de líneas procesadas
14      fin_mientras
15    fin_si
16    de otro modo Desplegar un mensaje de error (24)
        y volver a empezar en el paso 2
17  Fin del algoritmo
```

Algoritmo 4.1 Suma de imágenes

Nótese que para la suma no se requiere normalizar la operación si se calcula únicamente el promedio de la suma.

El resto de operaciones aritméticas se lleva cabo de la misma manera que la suma, utilizando el mismo algoritmo, excepto que en el paso 9 se aplica la operación correspondiente.

La complejidad del algoritmo anterior y de los tres algoritmos restantes es fácil de estimar. El ciclo principal (pasos 5–14) es muy parecido estructuralmente al del algoritmo 3.2 (pasos 4–9) por lo que se espera que sea de la misma complejidad. En efecto, en el paso 9 se requiere una operación [1], considerando el paso 8, se requieren N operaciones,

[1] Depende de qué operación se esté llevando a cabo entre imágenes: Suma, resta o cociente

dentro del ciclo 7–11 las operaciones son $N \times LineasALeer$, en donde $LineasALeer$ es un valor determinado por el tamaño de un bloque de memoria de $64K$ bytes y el ancho de las imágenes operandos:

$$LineasALeer = \left\lfloor \frac{64K}{N} \right\rfloor \tag{4.5}$$

Al realizar el ciclo 5–14 el valor de $LineasALeer$ tiende a M, la cantidad total de renglones de las imágenes de entrada. De lo anterior, se deduce que la complejidad del algoritmo es $O(MN)$.

4.1.3 Normalización del histograma

Las *operaciones estadísticas* actuan puntualmente sobre cada elemento de la imagen, *pero sin modificarlo.* Estas operaciones sirven únicamente al usuario para proporcionar información, de acuerdo a la cual se puede tomar alguna decisión respecto a la imagen.

Una de las operaciones de mayor importancia que se puede realizar con los elementos de una imagen es el cálculo de su *histograma*, operación que se puede clasificar dentro de las operaciones puntuales de una imagen ya que toma en consideración el valor de cada elemento de la misma, independientemente del resto de la imagen, pero a la vez es una operación que proporciona información estadística acerca de la imagen. Conociendo el histograma se pueden tomar varias medidas para el realce de las características de la imagen mediante una reclasificación de valores, como lo veremos enseguida.

Si ϑ_{ij} $\{i \in [1,\ldots,M],\ j \in [1,\ldots,N]\}$ es el valor de un elemento cualquiera de la imagen $I \in \Im$, el *histograma* de la misma, $h(i)$ $\{i = 0,\ldots,255\}$, se calcula utilizando el siguiente algoritmo:

$$\begin{aligned} &\text{Inicialmente } h(i) = 0 \qquad \forall\, i \in [0,\ldots,255] \\ &\qquad h(\vartheta_{ij}) = h(\vartheta_{ij}) + 1 \qquad \text{para cada } i = 1,\ldots,M \ \ j = 1,\ldots,N. \end{aligned} \tag{4.6}$$

El arreglo $h(i)$ $\{i = 0,\ldots,255\}$ define una función discreta cuyo intervalo está comprendido en el conjunto $Z_{[0\ldots255]}$ y su dominio, tambien acotado, varía en el conjunto de los enteros $Z_{[0\ldots MN]}$, el cual depende de las dimensiones de la imagen. El dominio define la distribución de datos contenidos en una imagen, ya que determina cuántos elementos de un valor dado existen dentro de la imagen.

En la sección 2.1 se mencionó que el histograma se encuentra almacenado al final de cada imagen. Consta de 256 valores de la forma $h(i)$, cada uno de los cuales representa

el número de veces que el elemento i se encuentra en la imagen. Cada uno de estos valores se almacena en 4 octetos con lo que cada elemento puede estar en una imagen 2^{32} veces.

El histograma muestra la distribución de datos dentro de la imagen que representa. Suele suceder que la distribución no sea uniforme y que el uso de las intensidades de grises no aproveche toda su escala. Si la distribución de datos en una imagen se encuentra alrededor de un solo número dado, la forma del histograma generado será un pulso.

La normalización del histograma no es más que la redistribución de los valores de grises en todo el intervalo $Z_{[0...255]}$ y el algoritmo utilizado para esta operación es el 4.2:

```
1   Inicio del algoritmo
2     Leer los nombres de la imágenes de entrada y la de salida
3     Leer el histograma H_0...255
4     Calcular el valor promedio del histograma Hp = MN/256
5     hacer Der = 0, Hint = Σ_{i=0}^{n} Hi = 0 (La integral del histograma)
6     para i = 0 hasta 255 hacer
7       hacer Izq = Der
8       hacer Hint = Hint + Hi
9       mientras Hint sea mayor que Hp hacer
10        Hint = Hint - Hp
11        Der = Der + 1
12      fin_mientras
13      hacer Hi_nuevo = (Izq + Der)/2
14    fin_para
15    mientras falten líneas por procesar hacer
16      Leer secuencialmente un bloque de datos
17      para cada renglón hacer
18        para cada columna hacer
19          v_ij = H_{v_ij_nuevo}
20        fin_para
21      fin_para
22      Escribir el bloque de datos obtenido
23    fin_mientras
24  Fin del algoritmo
```

Algoritmo 4.2 Normalización del histograma de una imagen

En el algoritmo anterior, la variable $H_{i_{nuevo}}$ es un arreglo de 256 elementos, el cual, a partir del paso 15, contiene el mapeo a los nuevos elementos de la imagen, es decir, el valor original de la imagen de entrada sirve de índice para localizar el nuevo elemento (el elemento normalizado) en el arreglo.

El mapeo se hace por bloques de datos y se lleva a cabo en el paso 19. El proceso se repite para cada bloque de datos de la imagen original, hasta que se termine con la misma. Al final, la imagen obtenida (normalizada) se almacena con un nuevo nombre, dado por el usuario, en el paso 2.

4.2 Operaciones de área

Las operaciones de área son aquéllas en las que el resultado obtenido sobre un elemento de una imagen, al aplicarle dicha operación, dependerá tanto de su valor original como de los valores vecinos al mismo.

La fórmula correspondiente a una operación de área se define de manera análoga a las operaciones puntuales:

$$\vartheta'_{ij} = f\begin{pmatrix} \vartheta_{i-c_1,j-c_2} & \cdots & \vartheta_{i-c_1,j} & \cdots & \vartheta_{i-c_1,j+c_2} \\ \vdots & & \vdots & & \vdots \\ \vartheta_{i,j-c_2} & \cdots & \vartheta_{i,j} & \cdots & \vartheta_{i,j+c_2} \\ \vdots & & \vdots & & \vdots \\ \vartheta_{i+c_1,j-c_2} & \cdots & \vartheta_{i+c_1,j} & \cdots & \vartheta_{i+c_1,j+c_2} \end{pmatrix} \quad \begin{array}{l} 1 \le i \le N \\ 1 \le j \le M \\ c_1 \ll N \\ c_2 \ll M \end{array} \tag{4.7}$$

en donde las constantes c_1 y c_2 definen el *alcance* de la operación de área. Normalmente estos valores son mucho más pequeños que las dimensiones de la imagen.

Sea un elemento $\vartheta_{ij} \in I$ cualquiera, se definen sus *4-vecinos inmediatos* como los elementos $\vartheta_{i+1,j}, \vartheta_{i-1,j}, \vartheta_{i,j+1}, \vartheta_{i,j-1}$ y sus *4-vecinos en diagonal* como los elementos $\vartheta_{i+1,j+1}, \vartheta_{i+1,j-1}, \vartheta_{i-1,j+1}, \vartheta_{i-1,j-1}$ [Pavl82]. Los ocho elementos forman el conjunto de *los 8-vecinos de* $\vartheta_{i,j}$. Estas definiciones se pueden ampliar tanto como se desee, siempre y cuando el elemento ϑ_{ij} se encuentre en el centro del área formada por los vecinos.

4.2.1 Filtros digitales lineales

Un filtro digital consiste en una operación que se realiza sobre una imagen y cuyo resultado es otra imagen con características diferentes a la original, y dependen del filtro que se use. A estos filtros se les conoce como *operadores locales*, ya que actúan transformando cada elemento de una imagen de acuerdo a su valor y al de sus vecinos inmediatos; se puede definir matemáticamente un filtro digital y la conexión que entabla entre la imagen original I y la imagen filtrada I', como:

$$\vartheta'_{ij} = \sum_{k=-m}^{m} \sum_{l=-n}^{n} h(i,j,k,l)\vartheta_{i+k,j+l} \qquad \begin{array}{l} \vartheta_{ij} \in I \quad \vartheta'_{ij} \in I' \\ \forall\, 1 \le i \le M \quad \text{y} \quad \forall\, 1 \le j \le N \end{array} \tag{4.8}$$

A este tipo de filtro se le conoce con el nombre de *filtraje lineal* si la función $h(i,j,k,l)$ es lineal en sus variables. Las constantes m,n definen las dimensiones del filtro que, en general, es de forma rectangular ($m \neq n$), sin embargo, los filtros más usuales definen los valores $m = n = 1$, con los cuales se construye un filtro cuadrado de tres por tres, en cuyo caso se utilizan los 8–vecinos del elemento en cuestión.

La función $h(i,j,k,l)$ depende de la posición del elemento a filtrar (*e.g.*, i,j), por lo que este filtro es *variante bajo el espacio*. Si se desea hacer un filtraje *invariante bajo el espacio*, independiente de la posición de los elementos a filtrar, entonces el filtro de la ecuación anterior se convierte en:

$$\vartheta'_{ij} = \sum_{k=-m}^{m} \sum_{l=-n}^{n} h(k,l)\,\vartheta_{i+k,j+l} \qquad \begin{array}{l} \vartheta_{ij} \in I \quad \vartheta'_{ij} \in I' \\ \forall\, 1 \leq i \leq M \quad \text{y} \quad \forall\, 1 \leq j \leq N \end{array} \tag{4.9}$$

en donde $h(k,l)$ está definida por una matriz a la que llamaremos *núcleo del filtro (i.e. kernel)* y cuyos elementos deberán definirse de acuerdo a la transformación que se debe llevar a cabo. Existen filtros lineales para una gran variedad de operaciones. Los que se muestran en la Figura 4.1, son los que están implantados en el SPID.

El algoritmo 4.3 utilizado para el filtraje de imágenes requiere únicamente los valores de m y n y los elementos $h(k,l)$ del *nucleo* del filtro. De esta manera el SPID intenta ser lo más eficiente posible dado que da la libertad total al usuario en la selección de las dimensiones y contenido del filtro. En el caso más simple en que $m = n = 1$, el núcleo estará formado por una matriz de elementos correspondientes, posicionalmente, al elemento mismo, y a sus 8–vecinos. Si se trata de un filtro en el que $m = n = 2$, el núcleo estará formado por una matriz de elementos correspondientes, posicionalmente, al elemento mismo, y a sus 24–vecinos, definidos éstos de manera análoga a los 8–vecinos.

Se evaluará el costo del filtraje lineal de una imagen de dimensiones M × N. De la fórmula (4.9) anterior, el cálculo de un elemento ϑ'_{ij} requiere de $(2m+1)(2n+1)$ sumas y productos, en donde $2m+1$ y $2n+1$ son el alto y ancho del kernel del filtro. Si se considera igual una suma a un producto en cuanto a tiempo de ejecución y sabiendo que existen M × N elementos en la imagen, el tiempo total en el filtraje de una imagen es proporcional a:

$$C(I') = 2MN(2m+1)(2n+1) \tag{4.10}$$

Si consideramos el caso comúnmente usado en que $m = n = cte \ll$ M,N, entonces $C(I') = O(MN)$. En la evaluación anterior se ha considerado que las imágenes I' e I se encuentran completamente en memoria principal, lo cual, en SPID no es siempre el caso, si la imagen es de dimensión pequeña, entonces (4.10) es válida, de otro modo se debe tomar en consideración el manejo de bloques de memoria secundaria a memoria principal

y viceversa, lo cual ocasiona un gasto computacional extra que va en función del tamaño del bloque que se pueda almacenar en un momento dado en memoria principal.

0	1	0
1	-4	1
0	1	0

Filtro laplaciano

0	0	0
0	2	-1
0	-1	0

Filtro gradiente

0	-1	0
-1	5	-1
0	-1	0

Filtro pasa altas

1	1	1
1	1	1
1	1	1

Filtro pasa bajas

-1	-2	-1
0	0	0
1	2	1

Filtro detector de bordes horizontales

1	0	-1
2	0	-2
1	0	-1

Filtro detector de bordes verticales

0	-1	2
-1	2	-1
2	-1	0

Filtro detector de bordes a 45°

2	-1	0
-1	2	-1
0	-1	2

Filtro detector de bordes a 135°

Figura 4.1 Filtros digitales implantados en SPID

Todas las operaciones realizadas en el Algoritmo 4.3 son muy generales en cada uno de los pasos mostrados. Los pasos 6, 7, 12, 13, 18 y 19 merecen una atención especial ya que son la parte central del filtraje de imágenes en el dominio espacial.

Los pasos 6, 12 y 18 indican una extrapolación de elementos de un bloque de datos. Esta operación es necesaria para llevar a cabo el filtraje de datos en las fronteras de la imagen. Para el filtraje en las fronteras de una subimagen se requiere tomar una decisión en la forma en que se llevará a cabo. Existen al menos dos formas de hacerlo:

```
1   Inicio del algoritmo
2     Leer los nombres de la imagen de entrada y la de salida
3     Leer las dimensiones del núcleo m × n
4     Leer los datos del filtro f(i,j)   1 ≤ i ≤ m     1 ≤ j ≤ n
5     Leer un bloque de datos de la imagen de entrada
6     Extrapolar el bloque leído en memoria para efectos de frontera
7     Filtrar el bloque de datos
8     Escribir el bloque de datos filtrado a la imagen de salida
9     hacer LineasEn64K = Número de líneas que caben en
      un segmento de memoria de 64K bytes
10    mientras las líneas que faltan por filtrar sean más que
      LineasEn64K hacer
11        Leer un bloque de datos de la imagen de entrada
12        Extrapolar el bloque leído en memoria para efectos de frontera
13        Filtrar el bloque de datos
14        Escribir el bloque de datos filtrado a la imagen de salida
15    fin_mientras
16    si faltan líneas por procesar entonces hacer
17        Leer el bloque de datos restante
18        Extrapolar el bloque leído en memoria para efectos de frontera
19        Filtrar el bloque de datos
20        Escribir el bloque de datos filtrado a la imagen de salida
21    fin_si
22  Fin del algoritmo
```

Algoritmo 4.3 Filtraje de imágenes

a) Colocando el filtro en las fronteras y despreciando aquellos elementos que geométricamente quedan fuera del filtraje.

b) Extrapolando los elementos de la frontera hasta donde cubre el filtro. La extrapolación se simplifica si se copian los elementos de las fronteras tantas veces como sea necesario.

En la implantación de este sistema se utilizó el esquema del inciso b), copiando elementos en las fronteras. El algoritmo 4.4 profundiza más sobre la extrapolación o repetición de elementos en una subimagen. De hecho esta parte prepara un bloque de datos para un filtraje posterior.

```
1   Inicio del algoritmo
2     La subimagen leída es
      (ϑ_ij     1 ≤ i ≤ Alto   1 ≤ j ≤ Ancho)
3     hacer AltoTotal = Alto + 2 * ⌊AltoFiltro / 2⌋
4     hacer AnchoTotal = Ancho + 2 * ⌊AnchoFiltro / 2⌋
5     hacer un corrimiento de elementos en el bloque de datos
      de tal manera que el elemento ϑ_11 se mueva a la posición de
      ϑ_{1+AltoFiltro,1+AnchoFiltro}
6     para i = AnchoFiltro/2 hasta i = AnchoTotal − AnchoFiltro/2
      hacer
7       para j = 1 hasta j = AltoFiltro/2 hacer
8         ϑ_ji = ϑ_{AltoFiltro/2+1,i}
9       fin_para
10    fin_para
11    para j = AnchoFiltro/2 hasta j = AnchoTotal − AnchoFiltro/2
      hacer
12      para i = AltoTotal − AltoFiltro/2 hasta i = AltoTotal hacer
13        hacer ϑ_ji = ϑ_{AltoTotal−AltoFiltro/2−1,i}
14      fin_para
15    fin_para
16    para j = 1 hasta j = AltoTotal hacer
17      para i = 1 hasta i = AnchoFiltro/2 hacer
18        hacer ϑ_ji = ϑ_{j,AnchoFiltro/2+1}
19      fin_para
20    fin_para
21    para j = 1 hasta j = AltoTotal hacer
22      para i = AnchoTotal − AnchoFiltro/2 hasta i = AnchoTotal
        hacer
23        hacer ϑ_ji = ϑ_{j,AnchoTotal−AnchoFiltro/2−1}
24      fin_para
25    fin_para
26  Fin del algoritmo
```

Algoritmo 4.4 Extrapolación de datos en una subimagen

Los pasos 7, 13 y 19 del algoritmo de filtraje de imágenes anterior (Algoritmo 4.3) requieren de un análisis más a fondo, ya que en ellos es donde se llevan a cabo las operaciones básicas del filtraje. El Algoritmo 4.5 ilustra más a fondo el mecanismo mediante el cual se filtra una subimagen.

1 Inicio del algoritmo

2 hacer $AltoTotal = Alto + 2 * \left\lfloor \frac{AltoFiltro}{2} \right\rfloor$

3 hacer $AnchoTotal = Ancho + 2 * \left\lfloor \frac{AnchoFiltro}{2} \right\rfloor$

4 *La subimagen a filtrar es*
$(\vartheta_{ij} \quad 1 \leq i \leq AltoTotal \quad 1 \leq j \leq AnchoTotal)$

5 para $i = AltoFiltro/2 + 1$ hasta $i = AltoTotal - AltoFiltro/2 - 1$ hacer

6 para $j = AnchoFiltro/2 + 1$
hasta $j = AnchoTotal - AnchoFiltro/2 - 1$ hacer

7 $$\vartheta'_{ij} = \sum_{k=-AltoFiltro}^{AltoFiltro} \sum_{l=-AnchoFiltro}^{AnchoFiltro} \vartheta_{i-k,j-l} Filtro_{kl}$$

8 *Normalizar el valor calculado:* $\vartheta'_{ij} = \dfrac{\vartheta'_{ij}}{AltoFiltro * AnchoFiltro}$

9 fin_para

10 fin_para

11 Fin del algoritmo

Algoritmo 4.5 Filtraje de subimágenes

En el paso 7 del Algoritmo 4.5, la variable $Filtro_{kl}$ se refiere al núcleo del filtro, el cual está dado en la fórmula 4.8 como la función h.

4.2.2 Ampliación de imágenes

Cuando se tiene información continua del comportamiento de una superficie, como la que genera una función de dos variables independientes en el espacio, esta superficie se puede analizar desde cualquier distancia a la misma ya que existe una fuente que genera la información necesaria para la creación de la superficie. De esta manera se crea la sensación de que se puede ampliar o reducir la superficie ampliando o reduciendo la distancia de observación a la misma.

En el caso de imágenes el problema no es tan trivial, ya que la información que se tiene es discreta y de tamaño fijo pues no se tiene a la disposición una función generadora. Sin embargo, es a veces necesario analizar una imagen desde diferentes puntos de vista, ampliando detalles de interés o reduciendo zonas muy extensas que no presentan cambios locales.

Se puede tomar la idea de una función generadora de imágenes la cual se discretiza en puntos equiespaciados a lo largo y ancho de una escena y en los cuales toma los valores que formarán la imagen. Suponiendo que esto es válido, se presenta a continuación una solución al problema de ampliación de imágenes.

Dada una imagen y un factor de escala, se podrá generar otra imagen cuya razón de crecimiento respecto a la original será igual al factor de escala. Si la escala es menor a la unidad, entonces la imagen se reducirá. Por el contrario, si la escala es mayor de la unidad, entonces la imagen se ampliará. En este sentido la ampliación y la reducción son el mismo problema.

Si la imagen original tiene dimensiones M × N y el factor de escala es e, entonces:

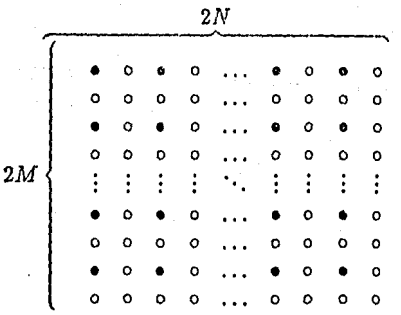
$$M' = \lfloor eM \rfloor ; \qquad N' = \lfloor eN \rfloor \tag{4.11}$$

en donde M' y N' son las dimensiones de la imagen ampliada. El factor de escala utilizado en el sistema SPID es 2 para la ampliación y de $\frac{1}{2}$ para la reducción, por lo que las fórmulas anteriores se convierten en:

$$\begin{aligned} M' &= \lfloor 2M \rfloor ; & N' &= \lfloor 2N \rfloor & &\text{Ampliación} \\ M' &= \left\lfloor \frac{M}{2} \right\rfloor ; & N' &= \left\lfloor \frac{N}{2} \right\rfloor & &\text{Reducción} \end{aligned} \tag{4.12}$$

A continuación se explicará el método utilizado para la operación de ampliación de imágenes. Se supondrá que una imagen es la representación de una escena digitizada en puntos discretos y cuya separación real es lo suficientemente cercana de tal manera que no se pierde la información contenida en la escena. Una imagen ampliada se verá como se muestra en la Fig. 4.2.

En un primer paso de la ampliación y en cada renglón de la imagen original, entre cada par de elementos se inserta uno nuevo, interpolado usando los vecinos en el renglón. Después de hacerlo con los renglones se obtiene una imagen ampliada a lo ancho; en un segundo paso, en cada columna de la imagen ampliada por renglones se inserta un nuevo elemento, interpolado usando los elementos vecinos en la columna. De esta manera, la ampliación de imágenes se ejecuta como una operación separable. Al final de estas dos etapas, se obtiene una imagen ampliada de dimensiones $2M \times 2N$, cuando el tamaño de la imagen original es $M \times N$.



• Elementos de la imagen original
o Elementos obtenidos con interpolación

Figura 4.2 Formato de una imagen ampliada.

Para fines de implantación de un algoritmo la función básica es la de ampliar un renglón. Esta ampliación se hará $M+N$ veces, M para los renglones y N para las columnas. Si vemos un renglón desde una perspectiva unidimensional se verá como se muestra en la Figura 4.3.

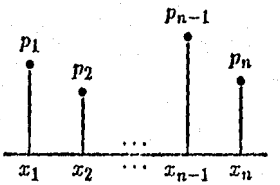


Figura 4.3 Un renglón cualquiera de una imagen.

en donde p_i $(i = 1,\ldots,n)$ es el valor, en un renglón, del elemento de la imagen en la posición x_i correspondiente. Para facilitar la notación utilizaremos p_i para señalar el i-ésimo elemento en un renglón cualquiera de una imagen, mientras que p_{ij} señalará el j-ésimo elemento del i-ésimo renglon de la misma, esto último sólo cuando el contexto sea bidimensional y se trate de señalar un renglón específico.

En la ampliación y reducción de imágenes se requiere de un determinado tipo de interpolación que llene los huecos dejados por la imagen al ser ampliada.

En este trabajo se manejarán tres tipos de ampliación. La decisión de cúal utilizar dependerá del usuario y de la imagen en particular que se esté analizando. El primer método que se utilizará será el de *doblamiento de elementos de una imagen*, éste consiste en repetir cada elemento de la imagen original 4 veces en la imagen ampliada, una vez en

su posición original, otra a la derecha y dos más abajo en las mismas columnas que las 2 anteriores. Con este esquema, la imagen ampliada se cuadruplica en número de elementos. El segundo método consiste en utilizar *interpolación lineal y cúbica segmentaria*, en este caso, se interpola cada elemento de acuerdo al valor de sus elementos vecinos sobre el mismo renglón o columna.

A continuación se muestran los algoritmos de ampliación de imágenes, utilizando repetición de elementos, interpolación lineal segmentaria e interpolación cúbica segmentaria y finalmente se presenta el algoritmo de reducción de imágenes.

```
1   Inicio del algoritmo
2     Dar el nombre de la imagen de entrada
3     Verificar que su formato sea correcto
4     mientras falten líneas por procesar hacer
5       Leer un bloque de datos:   ϑ_ij    1 ≤ i ≤ Alto   1 ≤ j ≤ Ancho
6       para i = 1 hasta i =Alto hacer
7         para j = 1 hasta j =Ancho hacer
8           ϑ'_{2i-1,2j-1} = ϑ'_{2i-1,2j} = ϑ_ij
9         fin_para
10      fin_para
11      para i = 1 hasta i =Alto hacer
12        para j = 1 hasta j = 2×Ancho hacer
13          ϑ'_{2i,j} = ϑ'_{2i-1,j}
14        fin_para
15      fin_para
16      Escribir el bloque de datos ampliado:
            ϑ'_ij    1 ≤ i ≤ 2 × Alto   1 ≤ j ≤ 2 × Ancho
17    fin_mientras
18  Fin del algoritmo
```

Algoritmo 4.6 Ampliación de imágenes por doblamiento de elementos

En los pasos 6-10 del Algoritmo 4.6 se hace la ampliación de una subimagen (ϑ_{ij}) por columnas, es decir, se dobla el ancho del bloque. En los pasos 11-15 se hace la ampliación por renglones, tomando en consideración la ampliación anterior. La subimagen ϑ_{ij} de dimensiones Alto×Ancho se amplía a la subimagen ϑ'_{ij} de dimensiones 2·Alto×2·Ancho.

4.2.3 Interpolación lineal segmentaria

La interpolación segmentaria deriva su nombre de la propiedad que tiene de llevarse a cabo *localmente*, es decir, interpola a partir de un conjunto de puntos comprendidos en una vecindad. Para el caso lineal, esta vecindad está compuesta de dos puntos.

La ecuación de una recta en el plano que pasa por los puntos (x_{k-1}, y_{k-1}) y (x_k, y_k) está dada por la fórmula:

$$y - y_{k-1} = \frac{y_k - y_{k-1}}{x_k - x_{k-1}} (x - x_{k-1}) \tag{4.13}$$

Esta ecuación se aplicará a la ampliación de imágenes, en donde éstas estan formadas de renglones y columnas. La interpolación se hará por renglones y después por columnas, pero para cada caso se aplicará la fórmula anterior a cada par de puntos. La interpolación entre los elementos $\vartheta_{c,k-1}, \vartheta_{c,k}$ de un renglon $c \in Z_{[1...M]}$ cualquiera de una imagen se llevará a cabo con la siguiente fórmula:

$$y - \vartheta_{c,k-1} = \frac{\vartheta_{c,k} - \vartheta_{c,k-1}}{x_k - x_{k-1}} (x - x_{k-1}) \tag{4.14}$$

La fórmula anterior se simplifica si se toman en consideración los siguientes hechos: Primero, la separación entre elementos de la imagen se considera unitaria, por lo que el denominador del lado derecho se hace unitario; segundo, el punto interpolado se encuentra en el punto medio de los puntos interpolantes, es decir, $x - x_{k-1} = x_k - x = x_k - x_{k-1}/2 = \frac{1}{2}$. Sin pérdida de generalidad, puede considerarse que la ecuación anterior se aplica sobre un renglon c cualquiera de la imagen, y por lo tanto, se puede ignorar la primera dimensión. Con las dos simplificaciones anteriores, la fórmula anterior se reduce a:

$$\vartheta'_k = \vartheta_{k-1} + \left(\frac{\vartheta_k - \vartheta_{k-1}}{2} \right) = \left(\frac{\vartheta_{k-1} + \vartheta_k}{2} \right) \tag{4.15}$$

la cual muestra que el punto a interpolar no es más que el promedio de los dos elementos de la imagen considerados.

En el Algoritmo 4.7 se implantan todas las fórmulas anteriores para la ampliación de imágenes utilizando interpolación lineal segmentaria. En principio, es casi el mismo algoritmo que el 4.6, lo único que cambia es la forma de obtener los nuevos elementos. En este caso el valor se obtiene promediando los valores de los dos elementos vecinos al elemento a obtener.

Para calcular la complejidad del Algoritmo 4.7 analicemos paso a paso su ejecución. El ciclo 6-11, que varía con i y j realiza una suma y un cociente por cada juego de índices, que en total son $Alto \times Ancho$. El ciclo 12-16, por el otro lado, realiza $2 \times Alto \times (2 \times Ancho)$ operaciones, totalizando $5 \times Alto \times Ancho$ por cada bloque de datos cargado a memoria. El número de bloques (b) que se cargan a memoria depende de las dimensiones de la imagen, y está dado por:

$$b = \frac{MN}{64K} \tag{4.16}$$

```
1   Inicio del algoritmo
2     Dar el nombre de la imagen de entrada
3     Verificar que su formato sea correcto
4     mientras falten líneas por procesar hacer
5       Leer un bloque de datos:   ϑ_ij   1 ≤ i ≤ Alto   1 ≤ j ≤ Ancho
6       para i = 1 hasta i = Alto hacer
7         para j = 1 hasta j = Ancho hacer
8           ϑ'_{2i-1,2j-1} = ϑ_ij
9           ϑ'_{2i-1,2j} = (ϑ_ij + ϑ_{i,j+1}) / 2
10        fin_para
11      fin_para
12      para i = 1 hasta i = Alto hacer
13        para j = 1 hasta j = 2×Ancho hacer
14          ϑ'_{2i,j} = (ϑ'_{2i-1,j} + ϑ'_{2i+1,j}) / 2
15        fin_para
16      fin_para
17      Escribir el bloque de datos ampliado:
          ϑ'_ij   1 ≤ i ≤ 2 × Alto   1 ≤ j ≤ 2 × Ancho
18    fin_mientras
19  Fin del algoritmo
```

Algoritmo 4.7 Ampliación de imágenes por interpolación lineal segmentaria

en donde el denominador es el tamaño de un segmento de memoria y M y N son el alto y ancho de una imagen, respectivamente. De aquí que el número total de operaciones en la ampliación lineal de imágenes sea de:

$$b \times 5 \times Alto \times Ancho = \frac{MN}{64K} \times 5 \times \frac{64K}{Ancho} \times Ancho = 5MN = O(MN) \qquad (4.17)$$

4.2.4 Interpolación cúbica segmentaria

Es conveniente, para el desarrollo del algoritmo general, considerar el caso unidimensional y posteriormente hacer una obvia extensión a dos dimensiones. Por cuatro puntos cualesquiera en el plano, se puede hacer pasar uno y sólo un polinomio cúbico (Boor, 1978), que cumple con las siguientes propiedades:

$$\begin{aligned} P_i(x_j) &= p_j & i &= 1,\ldots,n-1 \\ P_{i-1}(x_j) &= P_i(x_j) & j &= i-1, i, i+1 \end{aligned} \tag{4.18}$$

en donde $P_i(x_j), P_{i-1}(x_j)$ y $P_i(x_j)$ son polinomios cúbicos representados en la Figura 4.4.

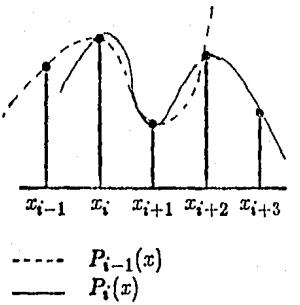


Figura 4.4 Dos polinomios adyacentes.

Es decir, el requisito de cada segmento polinomial es que pase por los cuatro puntos contiguos $(x_i, x_{i+1}, x_{i+2}, x_{i+3})$ y se traslapen uno con el otro en tres de esos cuatro puntos. En cada segmento polinomial se interpolará únicamente un punto, cuya posición se encuentra exactamente a la mitad de los cuatro interpolantes. Ya que la distribución de elementos en una imagen es homogénea consideraremos que la separación entre ellos es constante e igual a la unidad, es decir, $x_i - x_{i-1} = 1$, esto simplificará los desarrollos posteriores. Cada polinomio se puede referir a un sistema de coordenadas cartesiano, como lo ilustra la Figura 4.5.

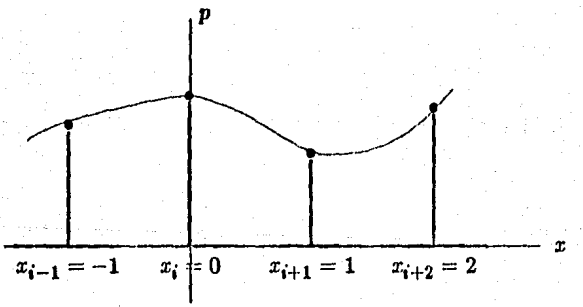


Figura 4.5 Superposición de un sistema de coordenadas.

De acuerdo a la Figura 4.5, la ecuación del polinomio cúbico interpolante se puede

determinar a partir de una tabla de diferencias divididas de Newton (Boor, 1978), de la siguiente manera:

$$
\begin{array}{llll}
-1 & p_i & & \\
 & & p_{i+1} - p_i & \\
0 & p_{i+1} & & \dfrac{p_{i+2} - 2p_{i+1} + p_i}{2} \\
 & & p_{i+2} - p_{i+1} & \\
1 & p_{i+2} & & \dfrac{p_{i+3} - 2p_{i+2} + p_{i+1}}{2} \\
 & & p_{i+3} - p_{i+2} & \\
2 & p_{i+3} & &
\end{array}
\qquad
\frac{p_{i+3} - 3p_{i+2} + 3p_{i+1} - p_i}{6}
$$

De donde se puede deducir el polinomio:

$$P(x) = p_i + (x+1)(p_{i+1} - p_i) + x(x+1)\frac{p_{i+2} - 2p_{i+1} + p_i}{2} + x(x^2-1)\frac{p_{i+3} - 3p_{i+2} + 3p_{i+1} - p_i}{6} \tag{4.19}$$

Como ya se dijo, el valor a interpolar se encuentra en el punto medio de los elementos $x_i = 0$ y $x_{i+1} = 1$, por lo que $x = (x_{i+1} - x_i)/2 = 1/2$, sustituyendo este valor en la ecuación anterior se obtiene:

$$P\left(\frac{1}{2}\right) = \frac{9(p_{i+1} + p_{i+2}) - (p_i + p_{i+3})}{16} \tag{4.20}$$

y esta es la fórmula que se aplica en la ampliación de imágenes. Aunque el cociente no necesariamente es entero, el valor esperado para $P\left(\frac{1}{2}\right)$ en un entero en el intervalo [0..255] por lo que el cociente se calcula con aritmética entera, truncando el resultado.

La fórmula (4.17) define un polinomio cúbico, y utilizando (4.18) se puede llevar a cabo la interpolación de todos aquellos elementos dentro del renglón que cuenten con dos elementos vecinos a cada lado, sin embargo, en los extremos de los mismos no se puede aplicar (4.18) ya que no existen los elementos suficientes para la interpolación, esto se ilustra en las Figuras 4.6a y 4.6b.

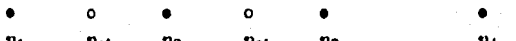


Figura 4.6a Interpolación en el extremo izquierdo.

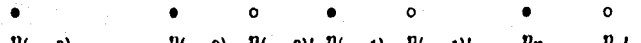


Figura 4.6b Interpolación y extrapolación en el extremo derecho.

En la Figura (4.6a) se muestran los cuatro elementos originales (•) y dos obtenidos con interpolación (o) del extremo izquierdo de un renglón en una imagen ampliada, mientras que en la Figura (4.6b) se muestran también cuatro elementos originales y tres obtenidos por interpolación y extrapolación. Para interpolar el elemento $p_{1'}$ (Figura 4.6a) utilizaremos el primer polinomio interpolante de cada renglón $P_1(x)$, de esta manera, según la Figura 4.6a y fijando el sistema de coordenadas que se muestra en la Figura 4.5, la interpolación será en el punto $x = -\frac{1}{2}$, sustituyendo este valor en el polinomio (4.17), tendremos:

$$P\left(-\frac{1}{2}\right) = p_{1'} = \frac{5(p_i + 3p_{i+1} - p_{i+2}) + p_{i+3}}{16} \tag{4.21}$$

De la misma manera se hace para el penúltimo elemento a interpolar ($p_{(n-1)'}$ en la Figura 4.6b), utilizando en este caso el último polinomio interpolante, $P_{n-3}(x)$, en el punto $x = \frac{3}{2}$, sustituyendo este valor en el polinomio (4.17), tendremos:

$$P\left(\frac{3}{2}\right) = p_{(n-1)'} = \frac{p_i - 5(p_{i+1} - 3p_{i+2} - p_{i+3})}{16} \tag{4.22}$$

Un caso especial se presenta en el extremo derecho, cuando se desea obtener el último elemento en la ampliación, $p_{n'}$ (Figura 4.6b). Este elemento no se encuentra dentro del renglón interpolante, como se puede ver en la figura. En este caso se hace una extrapolación utilizando el último polinomio interpolante, $P_{n-3}(x)$, de cada renglón. Para extrapolar en el punto $x = \frac{5}{2}$ nuevamente se utiliza el polinomio (4.17), según el cual se obtiene:

$$P\left(\frac{5}{2}\right) = p_{n'} = \frac{21p_{i+1} - 5(p_i + 7(p_{i+2} - 7p_{i+3}))}{16}. \tag{4.23}$$

Se puede considerar otra posibilidad cuando se amplía una imagen utilizando las fórmulas de esta sección: En el caso de las fronteras se puede descartar el elemento de las fronteras, en cuyo caso la reducción en las dimensiones de la imagen sería de dos renglones y dos columnas. Este caso especial no se maneja en SPID, sólo se menciona como posibilidad.

Utilizaremos las Fórmulas 4.18–4.21 para deducir la complejidad del algoritmo de ampliación de imágenes utilizando interpolación cúbica segmentaria. Se considerará que todas las operaciones aritméticas de punto fijo requieren del mismo tiempo de ejecución. La fórmula (4.18) requiere de dos sumas, una resta, un producto y un cociente; cinco operaciones en total. Las fórmulas (4.19), (4.20) y (4.21) requieren de seis operaciones aritméticas cada una. En cada renglon de la imagen original se insertan $N - 3$ elementos, que son interpolados usando la ecuación (4.18) y un elemento por cada ecuación (4.19), (4.20) y (4.21), de aquí que el número total de operaciones por renglón es de:

$$5(N-3)+18 = 5N+3. \tag{4.24}$$

Si la imagen tiene M renglones, entonces el número total de operaciones en el primer paso de la ampliación es $M(5N+3)$, en el segundo paso se realizan $N(5M+3)$, que se obtiene intercambiando M y N. De aquí que el número total de operaciones aritméticas es de:

$$10MN + 3(M+N) = O(MN). \tag{4.25}$$

4.2.5 Reducción de imágenes

El algoritmo para la reducción de imágenes lee bloques de datos a los cuales les extrae los elementos despreciando uno y reteniendo el otro para cada uno de los renglones y columnas de la subimagen. En el algoritmo 4.8 se muestra la reducción de imágenes.

Debido a que en este algoritmo no se lleva a cabo ninguna operación aritmética no se evalúa su complejidad, que está dada en función de los accesos a disco y del movimiento de datos en memoria.

```
1   Inicio del algoritmo
2     Dar el nombre de la imagen de entrada
3     Verificar que su formato sea correcto
4     mientras falten líneas por procesar hacer
5       Leer un bloque de datos:  ϑ_ij   1 ≤ i ≤ Alto   1 ≤ j ≤ Ancho
6       para i = 1 hasta i = ⌊Alto/2⌋ hacer
7         para j = 1 hasta j = ⌊Ancho/2⌋ hacer
8           ϑ'_ij = ϑ_{2i-1,2j-1}
9         fin_para
10      fin_para
11      Escribir el bloque de datos reducido:
          ϑ'_ij   1 ≤ i ≤ ⌊Alto/2⌋   1 ≤ j ≤ ⌊Ancho/2⌋
12    fin_mientras
13  Fin del algoritmo
```

Algoritmo 4.8 Reducción de imágenes por supresión de elementos

4.3 Operaciones vectoriales

Se han clasificado con el nombre de *operaciones vectoriales* aquéllas que modifican los elementos de una imagen dependiendo de los valores de todos los elementos restantes a lo largo del conjunto que forman una columna o un renglón de la imagen. En cierta forma este tipo de operaciones son un caso más general que el de operaciones de área, en las que el elemento modificado depende únicamente de un conjunto de elementos que forman una vecindad al mismo.

La fórmula utilizada para las operaciones vectoriales se define de manera análoga a las operaciones de área, de la siguiente manera:

$$v'_{ij} = f\begin{pmatrix} v_{11} & \cdots & v_{1j} & \cdots & v_{1N} \\ \vdots & & \vdots & & \vdots \\ v_{i1} & \cdots & v_{ij} & \cdots & v_{iN} \\ \vdots & & \vdots & & \vdots \\ v_{M1} & \cdots & v_{Mj} & \cdots & v_{MN} \end{pmatrix} \qquad \begin{matrix} 1 \leq i \leq N \\ 1 \leq j \leq M \end{matrix} \tag{4.26}$$

de donde se puede ver que las operaciones vectoriales dependen de toda la imagen para la actualización de cada uno de los elementos de la misma.

4.3.1 La Transformada de Fourier

El significado general de una transformación sobre una imagen es su descomposición en un conjunto de componentes que tienen algunas propiedades de las que se puede obtener algún provecho, propiedades tales como las variaciones espaciales en las componentes de una imagen. El concepto de filtrajes de la sección anterior vuelve a tener significado dentro del contexto de transformada de Fourier, si se toma en cuenta que una imagen puede descomponerse en una serie de frecuencias, se podrán, por ejemplo descartar las componentes de baja frecuencia y obtener un filtro pasa altas que sólo permita el paso de frecuencias medias y altas. Las componentes de baja frecuencia de la transformación contienen las variaciones muy lentas en la intensidad de la imagen, mientras que las componentes de alta frecuencia contienen las variaciones rápidas en la intensidad de la imagen.

La Transformada de Fourier de una función real de dos variables reales, $I(x,y)$ se define como:

$$I'(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I(x,y)e^{-jxu}e^{-jyv}\,dx\,dy. \tag{4.27}$$

Esta transformada se aplica únicamente a funciones de variables x, y continuas. En el procesamiento de imágenes digitales se requiere de una transformada de variables discretas.

La transformada discreta de Fourier (TDF) de una imagen I es la imagen I' definida con la fórmula siguiente:

$$I'(u,v) = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} I(k,l) e^{-j2\pi ku/M} e^{-j2\pi lv/N}. \tag{4.28}$$

Esta transformada se puede separar en las dos dimensiones que la componen, de tal manera que se pueda aplicar la transformada a la imagen por renglones y posteriormente por columnas, esta propiedad permite a SPID realizar la TDF por bloques.

La relación entre las fórmulas (4.27) y (4.28) es directa. Se puede demostrar, [Brac65], que si $I(k,l)$ se obtiene por muestreo de una función $I(x,y)$ continua de dos variables continuas x, y en los puntos $x = x_0 + k\Delta x$, $y = y_0 + l\Delta y$, en donde los valores de $I'(u,v)$ corresponden a los muestreos de la transformada continua de Fourier de $I(x,y)$ en los puntos $u/M\Delta x$, $v/N\Delta y$, la sustitución directa de estos valores en la ecuación 4.27, genera la ecuación 4.28 y la imagen $I'(u,v)$ obtenida con la TDF es periódica, con período M en u y período N en v, [Cool67].

En secciones posteriores haremos referencia a la TDF como una herramienta en el procesamiento de imágenes digitales. Una de sus aplicaciones más importantes es en la restauración de imágenes, utilizada muy a menudo ya que proporciona una mejora muy significativa en la calidad de la imagen.

Si se define $\alpha = e^{-j2\pi/N}$ y la matriz A como sigue:

$$A = \begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & \alpha & \dots & \alpha^{N-1} \\ & \vdots & \ddots & \\ 1 & \alpha^{N-1} & \dots & \alpha^{(N-1)(N-1)} \end{pmatrix}. \tag{4.29}$$

En donde el elemento $A_{k,l} = \alpha^{kl}$. La ecuación (4.28) se puede escribir como:

$$I' = AIA^T. \tag{4.30}$$

El cálculo directo de esta transformada implica $O(N^4)$ operaciones (sumas y productos). Para mostrar más facilmente como se programa la transformada discreta de Fourier, se define el caso unidimensional como:

$$I'(u) = \sum_{k=0}^{N-1} I(k) e^{-j2\pi ku/N} \qquad 0 \le u \le N-1 \tag{4.31}$$

en donde $I(k)$ es el valor de un elemento de un renglón de la imagen que se desea transformar y la cual está definida en los N puntos dados por la fórmula anterior. El valor de $I'(u)$ es también un elemento de un renglon de dimensión N ya que cada punto $I(k)$ se transforma en otro punto de aquélla. Hemos definido la transformada discreta ya que es ésta la que se programa en el procesamiento de imágenes.

En la implantación de la fórmula anterior se requiere hacer algunas simplificaciones para que se pueda aplicar el algoritmo conocido como *Transformada rápida de Fourier*. La fórmula (4.31) se puede expresar como:

$$I'(u) = \sum_{k=0}^{N-1} I(k) W_N^{ku} \qquad 0 \leq u \leq N-1 \tag{4.32}$$

en donde $W_N = e^{-j2\pi/N}$. Si se considera que $N = 2^n$, entonces $N = 2P$ para algún P entero positivo, de acuerdo con esto, la fórmula (4.32) se puede reescribir como:

$$I'(u) = \sum_{k=0}^{P-1} f(2k) W_{2P}^{2ku} + \sum_{k=0}^{P-1} f(2k+1) W_{2P}^{(2k+1)u}. \qquad 0 \leq u \leq N-1 \tag{4.33}$$

Se puede observar que $W_{2P}^{2ku} = W_P^{ku}$ y que $W_P^{u+P} = W_P^u$, $W_{2P}^{u+P} = -W_{2P}^u$. Si se define $I_1(u) = \sum_{k=0}^{P-1} f(2k) W_P^{ku}$ e $I_2(u) = \sum_{k=0}^{P-1} f(2k+1) W_P^{ku}$, la fórmula anterior se convierte en:

$$I'(u) = I_1(u) + I_2(u) W_{2P}^u \tag{4.34}$$

y se tiene además la siguiente propiedad:

$$F(u+P) = I'(u) - I_2(u) W_{2P}^u \tag{4.35}$$

4.3.2 Implantación de la transformada

```
1   Inicio del algoritmo
2     Dar el nombre de la imagen de entrada
3     Verificar que su formato sea correcto
4     Leer las dimensiones de la imagen de entrada (AltoImg×AnchoImg)
5     L2AnchoImg = log2(AnchoImg)
6     si ⌊L2AnchoImg⌋ no es igual a log2AnchoImg
      entonces L2AnchoImg = L2AnchoImg + 1
7     AnchoImgAmpliado = 2^L2AnchoImg
8     mientras falten líneas por procesar hacer
9       Leer un bloque de datos:  v_ij   1 ≤ i ≤ Alto  1 ≤ j ≤ Ancho
10      Expander el bloque leído a AnchoImgAmpliado en los renglones
        considerando que cada elemento de la imagen se transforma a un
        número complejo, con la parte imaginaria igual a cero
11      para i = 1 hasta i =Alto hacer
12        Transformar el i-ésimo renglon
13        Quitarle los elementos agregados en la expansión anterior
14      fin_para
15      Trasponer el bloque de datos transformado
16      Escribir el bloque de datos transformado
17    fin_mientras
18  Fin del algoritmo
```

Algoritmo 4.9 Cálculo de la semi-transformada

Con las últimas fórmulas de la sección anterior se hace la programación en la microcomputadora. Los detalles varían de acuerdo a las diferentes etapas del cálculo: Transformación en los renglones, en las columnas, cálculo del espectro de frecuencias y la normalización del mismo.

La imagen original se encuentra almacenada en disco, sus dimensiones son variables y limitadas únicamente por la capacidad de almacenamiento del mismo. Sin embargo, la memoria principal de la microcomputadora está limitada. Se hará la transformación de una imagen con un bloque de 64K *octetos* en memoria principal. La operación se llevará a cabo utilizando el esquema de subimágenes. Se carga una sección de imagen cuyas dimensiones no rebasen la capacidad de la memoria principal disponible y se calcula que quede espacio suficiente para almacenar la parte transformada.

La operación del párrafo anterior se repite tantas veces como sea necesario. Cada subimagen transformada se traspone en memoria principal y se almacena de esta forma en disco. Al finalizar la primera parte de la transformación (*e.g.* por renglones), se tiene una *semi-transformada transpuesta*. En el Algoritmo 4.9 se programa esta primera parte.

El paso 12 del Algoritmo 4.9 es la parte central del mismo, ya que en él se realiza la transformación de cada uno de los renglones de la imagen. En el algoritmo 4.10 se muestra la implantación de esta parte [Horo78, Aho74, Kron79].

Se inicia a continuación la segunda etapa que, consiste en calcular la transformada por columnas. Debido a que sería un problema serio la carga de una subimagen por columnas, se ha almacenado la semi-transformada transpuesta de tal manera que se puede leer de disco como una subimagen por renglones y que corresponden en realidad a columnas de la imagen original.

Se cargan subimagenes de la semi-transformada y se les aplica nuevamente la transformada de Fourier. Se trasponen nuevamente éstas y se almacenan en disco. Dos trasposiciones no afectan la posición original de los elementos. Se hace de tal forma que al terminar la segunda transformación se tiene ya la tarea terminada. El algoritmo es exactamente igual que el Algoritmo 4.9, excepto que en la expansion de elementos ya no se convierte a números complejos porque en la primera transformación se hizo así.

```
1    Inicio del algoritmo
2      Ancho = 2^m
3      j = 0
4      para i = 0 hasta i = Ancho − 1 hacer
5        si i < j entonces hacer
6          Intercambiar los valores de ϑ_i y ϑ_j:
7          Temporal = ϑ_i
8          ϑ_i = ϑ_j
9          ϑ_j = Temporal
10       fin_si
11       k = Ancho / 2
12       mientras k ≤ j hacer
13         j = j − k
14         k = k/2
15       fin_mientras
16       j = j + k
17     fin_para
18     para k = 1 hasta k = log2 Ancho hacer
19       Pot = 2^k
20       PotM1 = 2^(k−1)
21       r = (1,0)                    Constante compleja
22       s = (cos(π/PotM1),sen(pi/PotM1))
23       para k = 1 hasta PotM1 hacer
24         para i = k hasta i =Ancho sumando a i, Pot hacer
25           Indice = i+PotM1
26           Temporal = rϑ_Indice
27           ϑ_Indice = ϑ_i+ Temporal
28           ϑ_i = ϑ_i− Temporal
29         fin_para
30         r = rs
31       fin_para
32     fin_para
33   Fin del algoritmo
```

Algoritmo 4.10 Transformada rápida de Fourier de un renglón

La complejidad del Algoritmo 4.10 es $O(N \log N)$ [Horo78, Aho74, Kron79] y servirá para calcular la complejidad en el cálculo de la TRF de toda la imagen. El cálculo de la TRF se lleva cabo sobre M renglones y posteriormente sobre N columnas, de aquí que la complejidad total del algoritmo sea:

$$M(N \log N) + N(M \log M) = MN(\log N + \log M) = MN \log MN \tag{4.36}$$

Como parte complementaria al cálculo de la Transformada de Fourier de una imagen se mapea su *espectro de potencias de la imagen* al conjunto $Z_{[0...255]}$ y desplegarlo como una imagen.

$$e_{ij} = \sqrt{Re^2 + Im^2} \qquad \begin{matrix} 0 \leq i \leq M-1 \\ 0 \leq j \leq N-1 \end{matrix} \tag{4.37}$$

en donde Re e Im son la parte real e imaginaria de cada punto transformado. Este cálculo es más fácil que la transformada ya que se aplica únicamente a un punto. En el cálculo de la ecuación anterior se utilizará el mismo esquema de subimagen, es decir, la imagen se procesa por bloques.

Calculado el espectro, se procederá a normalizarlo, cuya finalidad es la siguiente: el espectro de frecuencias de una imagen se encuentra en el intervalo $[e_{min} \ldots e_{max}]$ y se requiere que éste sea mapeado al intervalo $[0 \ldots 255]$. El mapeo debe ser logarítmico debido a la distribución de los datos del espectro, y se calcula con la fórmula:

$$e'_{ij} = \log_2(1 + |e_{ij}|). \qquad \begin{matrix} 0 \leq i \leq M-1 \\ 0 \leq j \leq N-1 \end{matrix} \tag{4.38}$$

El algoritmo 4.11 es una implantación de las ecuaciones (4.37) y (4.38) anteriores, éste genera una imagen que corresponde al espectro de frecuencias de la imagen original, pero con un inconveniente que resolveremos a continuación. El espectro generado tiene los cuadrantes intercambiados: El cuadrante 1 va al cuadrante 3 y viceversa, el cuadrante 2 va al cuadrante 4 y viceversa. Los cuadrantes se numeran como en el plano cartesiano. Esta operación es la que se hace finalmente para que el espectro de frecuencias quede de manera correcta al desplegarlo.

```
1   Inicio del algoritmo
2     AmplitudMaxima = 1 × 10^38
3     AmplitudMinima = −1 × 10^38
4     mientras falten líneas por leer hacer
5       Leer un bloque de número complejos:
          C_ij   1 ≤ i ≤Alto   1 ≤ j ≤Ancho
6       para i =1 hasta i =Alto hacer
7         para j =1 hasta j =Ancho hacer
8           e_ij = log_2 ( 1 + sqrt( Re^2(C_ij) + Im^2(C_ij) ) )
9           si e_ij > AmplitudMaxima
                entonces AmplitudMaxima = e_ij
10            de otro modo
11              si e_ij < AmplitudMinima
                    entonces AmplitudMinima = e_ij
12          fin_si
13        fin_para
14      fin_para
15      Escribir el bloque de números reales que componen
        el espectro de frecuencias
16    fin_mientras
17    mientras falten líneas por normalizar hacer
18      Leer un bloque de datos: e_ij   1 ≤ i ≤Alto   1 ≤ j ≤Ancho
19      para i =1 hasta i =Alto hacer
20        para j =1 hasta j =Ancho hacer
21          ϑ_ij = ( (AmplitudMaxima − AmplitudMinima) / 255 ) e_ij
22        fin_para
23      fin_para
24      Escribir la subimagen normalizada:
          ϑ_ij   1 ≤ i ≤Alto   1 ≤ j ≤Ancho
25    finmientras
26  Fin del algoritmo
```

Algoritmo 4.11 Cálculo de las amplitudes máxima y mínima y normalización del espectro

Capítulo 5

Biblioteca de funciones del sistema SPID en C

Las operaciones básicas sobre las que se apoya SPID fueron escritas en lenguaje ensamblador debido a las características de velocidad y capacidad necesarias en el manejo de grandes bloques de memoria. Son varias las funciones básicas requeridas, por lo que se clasificarán de acuerdo a su objetivo. El lenguaje huesped fue C por sus características de direccionamiento y por la flexibilidad de programación. En este lenguaje se escribieron los programas principales de SPID que hacen uso de las funciones y subrutinas básicas.

A continuación se mostrarán los encabezados utilizados por el lenguaje C para cada una de las rutinas y funciones del sistema. Estos encabezados identifican, además del nombre, los parámetros utilizados por las funciones. En el apéndice A se lista el código fuente de las funciones de la biblioteca de entrada y salida del sistema.

Este capítulo se ha dividido en 6 secciones, cada una de las cuales identifica diferentes tipos de funciones y procedimientos de acuerdo a la tarea que realizan.

En la Sección 1, se muestran los encabezados de las funciones y procedimientos que utilizan la tarjeta gráfica HLGE. Se mencionan algunos aspectos teóricos sobre el funcionamiento e implantación de cada una de las funciones y en especial las que utilizan la tarjeta gráfica y cuya programación requiere algunos conocimientos sobre la misma.

En la Sección 2 se muestran los encabezados de funciones y procedimientos que utilizan la pantalla en modo gráfico normal, con la HLGE sirviendo de emulador de gráficas, estas rutinas utilizan la pantalla en modo gráfico de la IBM–AT, con 4 colores y a una resolución de 200 × 360 puntos. La emulación trata la HLGE como si fuese una CGA (*Color Graphics Adaptor*) normal.

En la Sección 3 se muestran los encabezados de funciones y procedimientos que realizan la entrada/salida de datos, se definen algunas funciones que son estándares de la biblioteca del lenguaje C y que se hizo únicamente para resaltar sus características específicas. En esta parte no se analizan las funciones desde un punto de vista técnico y

de implantación ya que esto se hace hasta el Apéndice A.

En la Sección 4 se muestran los encabezados de las funciones y procedimientos de propósito general que utiliza el sistema. Estas funciones tienen la característica de realizar una función específica, que por su naturaleza no se clasificaron dentro de las otras bibliotecas.

En la Sección 5 se muestran los encabezados de las funciones y procedimientos que utilizan el coprocesador matemático. Casi todas estas funciones son las que se utilizan en el proceso de la transformada de Fourier, sin embargo, se cree que esta biblioteca puede seguir creciendo a medida que se amplíe el sistema y se requieran más operaciones que utilicen el coprocesador.

Finalmente, en la Sección 6 se muestran los encabezados de un conjunto de funciones y procedimientos que fueron escritos en el siguiente nivel de programación. Cada una de estas funciones utiliza las funciones de las secciones anteriores y se escribieron en C.

5.1 Subrutinas de modo gráfico, alta resolución, spidtrj.h

Este grupo de funciones y procedimientos será el que deba de cambiarse en el caso de un cambio de tarjeta de alta resolución, como puede ser el caso debido al gran avance tecnológico de estos tiempos. Se han agrupado en esta sección para una mayor información al usuario y para el programador mismo.

Int IniGrafTrj( ).

Esta función inicializa la tarjeta de despliegue de alta resolucion. Verifica que ésta exista. Esta rutina debe ser la primera que se llama en el programa, para que éste pueda empezar. Regresa un valor de cero si no existe la tarjeta de alta resolución o si hay algún problema con la misma. Realiza cinco tareas: el modo de comunicación entre la tarjeta y el sistema es en ASCII (CA) y hexadecimal (CX), esta función inicializa a CX; inicializa todas las banderas de la tarjeta gráfica (para más información, ver [Vect86]); limpia la pantalla a un color llamado COLORFONDO (ver apéndice B); crea una ventana de toda la pantalla (0,639,0,479) y, finalmente, define todos los caracteres que aparecen en el menú de despliegue, tales como las flechas en el recuadro superior derecho.

void ModoGrafTrj( ).

Esta función inicializa la pantalla para despliegue de imágenes utilizando la tarjeta HLGE en alta resolución. No regresa ningún valor. Esta función actúa de la siguiente manera: El modo de despliegue de la HLGE lo lee de la localidad de memoria C600:030C; si encuentra un cero, entonces el estado es de alta resolución; si es uno, entonces su estado es de emulación de la CGA.

void ModoTextTrj( ).

Esta función conmuta del modo grafico de alta resolución a modo textual. No regresa ningún valor. El funcionamiento de este modo es igual al de la descripción hecha en el párrafo anterior para la emulación. Se alcanza este modo escribiendo un 1 en la localidad C600:030C de memoria.

void UsaTrjAsc(Cadena).

Esta subrutina envía una cadena de caracteres a la tarjeta de alta resolución para su procesamiento, la dirección en memoria de la cadena es Cadena:APTRCERCA. No regresa ningún valor. La cadena debe estar en formato ASCIIZ.[1] El envío se hace a través de una sección de memoria que empieza en la localidad apuntada por la localidad C600:0300.

void UsaTrjHex(Cadena, Numoctetos).

Esta subrutina envía Numoctetos:**int** a la tarjeta de alta resolución para su procesamiento, la dirección en memoria de la cadena es Cadena:APTRCERCA. No regresa ningún valor. Su funcionamiento es análogo a la función anterior, excepto que ahora la cadena de caracteres lleva un contador que indica su longitud.

void EscribeDMA(AbsInf, AbsDer, OrdInf, OrdSup).

Esta función muestra una sección de imagen en pantalla. Esta subrutina se utilizará en el ventaneo y enrollamiento de imágenes. Las coordenadas de la ventana son AbsInf:**int**, Abscisa en pantalla del margen izquierdo; AbsDer:**int**, Abscisa del margen derecho; OrdInf:**int**, Ordenada del margen inferior y OrdSup:**int** Ordenada del margen superior. No regresa ningún valor.

Para la transferencia se hace uso de los puertos 2,3,A,B,C y 83 para escritura y lectura. Para empezar escribe un 0 a la localidad de memoria C600:0310 lo cual indica a la tarjeta que se va a iniciar una transferencia de datos a través del canal 1 del DMA. A continuación se envían las coordenadas de los puntos en la pantalla que definen el área de despliegue y se inicia la transferencia, ésta termina cuando la localidad en memoria C600:0310 tiene un valor diferente de cero.

void LeeDMA(AbsInf, AbsDer, OrdInf, OrdSup).

Esta función lee una sección de imagen en pantalla. Esta subrutina se utilizara en el ventaneo y enrollamiento de imágenes. Las coordenadas de la ventana son AbsInf:**int**, Abscisa en pantalla del margen izquierdo; AbsDer:**int**, Abscisa del margen derecho; OrdInf:**int**, Ordenada del margen inferior y OrdSup:**int** Ordenada del margen superior. No regresa ningún valor. Al igual que la función anterior utiliza el canal del DMA, sólo que la transferencia es ahora de la tarjeta hacia la memoria del sistema.

---

[1] Este formato consta de una cadena de caracteres terminada con un cero.

void CargaTablaLUT(AptrLUT).

Esta subrutina actualiza la tarjeta gráfica HLGE con un conjunto de valores (256) definidos por una tabla cuyo índice es AptrLUT:**Int** y que determinan los colores que se van a utilizar en el despliegue de una imagen. La longitud de esta tabla es 2*256 = 512 =LNGREGLUT. Estos valores se leen del archivo TBL-LUTS.DAT, creado con la utilería vista en la Sección 3.3. No regresa ningún valor.

El proceso es muy sencillo: Se habre el archivo TBL-LUTS.DAT, se calcula la posición de la tabla pedida con la operación: Tabla pedida=256*IndLUT, y se leen 512 octetos que corresponden a la tabla requerida.

void SalvaTablaLUT(AptrLUT).

Esta subrutina guarda un registro de 256 valores de LUT's en el archivo TBL-LUTS.DAT. Se identifica con un índice, AptrLUT:**Int**, el cual se le pasa de parámetro. No regresa ningún valor. La longitud de este es $2 \times 256 = 512$ = LNGREGLUT. Opera igual que la función anterior excepto que en vez de leer ahora escribe.

void DefineLUT(IndColor, IntRojo, IntVerde, IntAzul).

Esta subrutina carga un valor en la tabla de LUT's, definiendo de esta manera un color dado por las intensidades de los indices: IndColor:octeto, índice que define el color; IntRojo:octeto, intensidad en el color rojo IntVerde:octeto, intensidad en el color verde; IntAzul:octeto, intensidad en el azul. Se utiliza en la defincón de tablas, creando entrada por entrada, las cuales define el usuario con el programa analizado en la sección 3.3.

void TablaLUTPred(IndLUT).

Este procedimiento carga una tabla de LUT's de las que ya están definidas en la tarjeta gráfica usando el índice IndLUT:octeto ($0 \leq$ IndLUT ≤ 3). No regresa ningún valor.

5.2 Rutinas de graficación, spidgraf.h

En el lenguaje C no existe un estándar de funciones gráficas, por lo que se creó un conjunto de funciones y subrutinas para tal fin, es en esta sección donde se muestran los encabezados de las mismas. Estas funciones se escribieron en lenguaje ensamblador.

void ModoGrafico().

Este procedimiento inicializa la tarjeta de despliegue a modo gráfico 4 colores y 320 columnas por 200 renglones. No regresa ningún valor. Usa la función 4 de la interrupción 10h del sistema operativo.

void ModoTextual(Atributo).

Este procedimiento inicializa la tarjeta de despliegue a modo textual normal de 80 columnas por 25 renglones. El parámetro que se le pasa es un atributo que despliega en toda la pantalla (color en que queda la pantalla). No regresa ningún valor. Utiliza la función 3 de la interrupción 10h del sistema operativo.

void MueveCursor(Columna, Renglón).

Este procedimiento posiciona el cursor en una columna (Columna:octeto) y renglon (Renglon:octeto) determinados. No regresa ningún valor. Usa la función 2 de la interrupción 10h.

void EscCarAtr(X, Y, Caracter, Atributo).

Este procedimiento escribe un Caracter:octeto en la pantalla junto con su Atributo:octeto, en las coordenadas X:octeto, Y:octeto. No regresa ningún valor. Esta función no utiliza el sistema operativo, escribe directamente en memoria el caracter.

void EscCadAtr(X, Y, Atributo, Cadena).

Este procedimiento escribe una Cadena:APTRCERCA de caracteres en formato ASCIIZ, con un Atributo:octeto, en las coordenadas (X:octeto, Y:octeto). No regresa ningún valor. Funciona de manera análoga a EscCarAtr.

APTRCERCA SalvaVent(DirBaseVent, AptrVent).

La función almacena una ventana en memoria para restablecerla posteriormente. Regresa la dirección de almacenamiento. DirBaseVent:APTRCERCA es la dirección base de la ventana a almacenar AptrVent:VENTANA * es un apuntador a una ventana que define las características de la misma.

void RestableceVent(DirBaseVent, AptrVent).

Este procedimiento despliega en la pantalla una ventana que previamente se había almacenado. Esto sucede cuando se regresa en un nivel en el manejador de menús. No regresa ningún valor. DirBaseVent:APTRCERCA es la dirección base de la ventana a desplegar y AptrVent:VENTANA * es un apuntador a una ventana que define las características de la misma.

5.3 Subrutinas de entrada/salida, spides.h

Aunque la biblioteca estándar de C contiene algunas de las funciones y procedimientos que se definen en esta sección, aquéllos son de propósito muy general, mientras que en SPID se requieren tareas más simples, reduciendo de esta manera el código generado. Por ejemplo, la función estándar de C, printf, SPID la utilizaría únicamente para desplegar una cadena de caracteres, en este caso, mejor se define una función más pequeña que lo haga. Todos los listados fuente de estas funciones se encuentran en el apéndice A, en donde se hará una explicación más detallada de cada una de éstas.

byte LeeCar().

La función lee un caracter del teclado sin desplegarlo en la pantalla. Regresa el código ASCII del caracter leído. Usa la función 7 de la interrupción 21h del sistema operativo.

byte LeeCarEco().

La función lee un caracter del teclado y lo despliega en la pantalla. Regresa el código ASCII del caracter leído. Usa la función 1 de la interrupción 21h del sistema operativo.

void LeeCadena(DirBaseCad).

La función lee una cadena de caracteres en el formato ASCIIZ. Se le pasa la dirección donde colocará la cadena, DirBaseCad:APTRCERCA. Utiliza la función anterior LeeCarEco, el fin de la cadena se detecta con el caracter CR.

int LeeEntero().

La función lee un número. Regresa su valor. Verifica que no haya error en el formato, usa la función anterior LeeCadena.

float LeeReal().

La función lee un número real en precisión simple. Regresa la dirección donde se encuentra el valor de este número. Usa la función LeeCadena. Verifica que el formato sea correcto.

void EscCar(Caracter).

La función escribe un caracter en la pantalla. Caracter:**char** es el caracter que se desea imprimir. Usa la función 2 de la interrupción 21h del sistema operativo.

IMAGEN AbreImg(NomImg).

La función abre una imagen para lectura y escritura. Si existe un error regresa un 0, de otro modo regresa el identificador de la misma. NomImg:APTRCERCA es un apuntador a una cadena ASCIIZ del nombre de la imagen por abrir. Usa la función 3Dh de la interrupción 21h del sistema operativo.

IMAGEN CreaImg(NomImg).

La función crea una imagen para lectura únicamente. Si existe un error regresa un 0, de otro modo regresa el identificador de la misma.
NomImg:APTRCERCA es un apuntador a una cadena ASCIIZ del nombre de la imagen por crear. Usa la función 3Ch de la interrupción 21h del sistema operativo.

void CierraImg(Img).

La función cierra una imagen. No regresa valor alguno. Img:IMAGEN es el identificador de la imagen. Usa la función 3Eh de la interrupción 21h del sistema operativo.

void BorraImg(NomImg).

La función borra una imagen. NomImg:APTRCERCA es una cadena ASCIIZ del nombre de la imagen a borrar. No regresa ningún valor. Usa la función 41h de la interrupción 21h del sistema operativo.

int ErrorLectura().

La función regresa un 1 si hubo en error en la lectura del dato anterior, esta función se aplica a la lectura de números enteros y reales, de otro modo regresa un 0.

int ErrorDOS().

La función regresa el número de error cometido en una operación del sistema operativo en disco (DOS). Ver la tabla 3.1 para una información más completa.

long LeeLongImg(Img).

La función lee la longitud de una imagen. Esta función sirve para detectar si el formato de una imagen es válido o no, comparando el tamaño físico de la imagen con las dimensiones que se encuentran al final del archivo que define la imagen. IdentImg:IMAGEN es el identificador de la imagen. Regresa la longitud física de una imagen.

void LeeDimension(Img, MMD DIMEN far *).

La subrutina lee los datos de una imagen. Img:IMAGEN es el identificador de la imagen; MMD:DIMEN far * es un apuntador a una estructura en la que se regresa la información referente a la imagen.

PALABRA LeeSubImg(Img, DirBase, LongX, RefY, RefX, NumCols).

La subrutina carga en memoria una sección de una imagen. Regresa el número de líneas leídas. Los parámetros son: Img:IMAGEN es el identificador de la imagen de que se van a leer los datos; DirBase:APTRLEJOS es la dirección en memoria a la que se transfieren los datos; LongX:**int** es el ancho de la imagen en disco; RefY:**int** es la ordenada de la esquina superior izquierda de la subimagen a leer; RefX:**int** es la abscisa de la coordenada formada por el punto (RefX,RefY); y, finalmente, el último parámetro es NumCols:**int**, el número de columnas de la subimagen a leer.

PALABRA LeeSeq(Img, BaseImg, Numbytes).

La función lee secuencialmente un archivo de datos a partir de la posición del apuntador de archivo hasta la longitud deseada. Regresa el número de octetos leídos. Los parámetros son: Img:IMAGEN es el identificador de la imagen que contiene los datos de que se va a leer; BaseImg:APTRLEJOS es un apuntador a la dirección de transferencia de los datos; Numbytes:PALABRA es la cantidad de datos pedida.

void EscSubImg(Img, BaseImg, LongX, RefY, RefX, NumRens, NumCols).

La subrutina escribe una subimagen de memoria a disco. No regresa ningún

valor. Los parámetros son: Img:IMAGEN es el identificador de la imagen a la que se va a escribir; BaseImg:APTRLEJOS es un apuntador a la dirección en memoria, de transferencia de la subimagen a escribir; LongX:**int** es el ancho de la imagen en disco; RefY:**int** es la ordenada de la esquina superior izquierda de la imagen en disco; RefX:**int** es la abscisa del mismo punto, formado por la pareja: (RefX,RefY); NumRens:**int** es el número de renglones de la subimagen a escribir y NumCols:**int** es el número de columnas de la subimagen.

PALABRA EscSeq(Img, BaseImg, Numbytes).

La función escribe secuencialmente en un archivo de datos a partir de la posición del apuntador de archivo hasta la longitud deseada. Los parámetros son: Img:IMAGEN es el identificador de la imagen a la que se escribirán los datos; BaseImg:APTRLEJOS es un apuntador en memoria a la dirección de transferencia de los datos y Numbytes:PALABRA es la cantidad de datos a escribir.

5.4 Subrutinas de propósito general, spidgral.h

En esta sección se muestran los encabezados de las funciones y procedimientos utilizados por SPID de propósito general. Existen también aquí, al igual que en las secciones anteriores, algunas funciones y procedimientos de la biblioteca estándar de C, pero se prefirió escribir una version propia.

void Retardo(Tiempo).

La función genera un ciclo nulo de tal forma que la computadora permanece inactiva en un período de tiempo dado por el parámetro Tiempo:**int**.

BYTE Ajusta(ValAAjustar).

La función convierte un número entero, ValAAjustar a BYTE con las siguientes características:

a) Si es menor de cero, lo ajusta a cero,

b) Si es mayor de 255, lo ajusta a 255,

c) De otro modo, toma sólo el BYTE de la parte baja.

void EntAAsc(NumEnt, CadSal).

La rutina convierte un número entero a cadena ASCIIZ. NumEnt:**int** es el valor que se desea convertir a cadena. CadSal:Cadena es la cadena resultante de convertir NumEnt a ASCIIZ, ésta debe ser un arreglo de caracteres definido global.

BYTE Checa8087( ).

La función verifica que exista el coprocesador matemático sin el cual SPID no puede funcionar. Regresa un 0 si hay un coprocesador, de otro modo, regresa

otro valor.

void Convierte(ArchEnt, Alto, Ancho).

La subrutina convierte un archivo de datos en una imagen. Calcula el histograma resultante y la información relacionada con la imagen, tal como su altura, el ancho y los elementos mínimo y máximo, según el formato mostrado en la figura 2.1. No regresa valor alguno. Los parámetros que se le pasan son los siguientes: ArchEnt:IMAGEN es el identificador del archivo de datos de entrada; Alto:Intes el número de renglones de la imagen; Ancho:Int es el ancho de la imagen o número de columnas.

APTRLEJOS BaseImg(Opción).

La función regresa un apuntador que corresponde a la dirección en donde inicia el almacenamiento de las subimágenes procesadas en SPID. La división es a segmentos, por lo que el número regresado corresponde a un segmento en memoria, con un offset de cero. El parámetro Opcion:BYTE indica el segmento de memoria que se desea direccionar:

1: Regresa un apuntador al primer segmento de memoria que utiliza SPID, SEG_IMG1.

2: Regresa un apuntador al segundo segmento de memoria que utiliza SPID, SEG_IMG2.

3: Regresa un apuntador al segmento que se encuentra entre el primero y el segundo y es división de 64K, ver la fórmula 2.5 y la figura 2.2.

void DoblaSubImg(DirOrigen, DirDestino, Renglones, Columnas.).

El procedimiento amplía una subimagen por doblamiento de sus elementos originales. No regresa ningún valor. Los parámetros que se le pasan son: DirOrigen:APTRLEJOS es un apuntador a la dirección del bloque a ampliar; DirDestino:APTRLEJOS es la dirección del bloque ampliado; Renglones:Int es la altura de la subimagen a ampliar y Columnas:Int es el ancho de la subimagen a ampliar. Ver el algoritmo 4.6.

void AmpliaLineal(DirOrigen, DirDestino, Renglones, Columnas).

El procedimiento amplía una subimagen utilizando interpolación lineal entre cada dos elementos y extrapolación en la frontera. No regresa ningún valor. Los parámetros que se le pasan son iguales al procedimiento anterior. Ver el algoritmo 4.7.

void AmpliaCubico(DirOrigen, DirDestino, Renglones, Columnas).

El procedimiento amplía una subimagen utilizando interpolación cúbica segmentaria en la parte central y lineal en los extremos utilizando extrapolación en las fronteras. No regresa ningún valor. Los parámetros son los mismos que los dos procedimientpos anteriores. Ver la fórmula 1.3 y el algoritmo 1.1.

void TraspBlqBytes(DirOrigen, DirDestino, Renglones, Columnas).

El procedimiento traspone un bloque de bytes. No regresa ningún valor. Los parámetros, nuevamente, son los mismos de los tres procedimientos anteriores.

void DespliegaHisto(Img).

El procedimiento muestra el histograma de una imagen dada. No regresa ningún valor. Img:IMAGEN es el identificador de la imagen. Ver el algoritmo 2.3.

void CreaMapeo(Img, DirArreglo, Renglones, Columnas).

El procedimiento genera un arreglo de 256 elementos que sirven para mapear una imagen de acuerdo al histograma normalizado. Img:IMAGEN es el identificador de la imagen de entrada; DirArreglo:APTRCERCA es la dirección del arreglo que forma el mapeo Renglones:PALABRA es el número de renglones de la imagen y Columnas:PALABRA es el ancho de la imagen.

void MapeaBloque(DirOrigen, DirMapeo, NumBytes).

El procedimiento es la parte complementaria del procedimiento anterior, veáse el algoritmo 4.2 en donde aparecen ambas funciones. En este procedimiento se transforma un bloque de bytes de la imagen original, de acuerdo al mapeo, en otro bloque. DirOrigen:APTRLEJOS es la dirección de transferencia de los datos; DirMapeo:APTRCERCA es la dirección en la que se encuentra el arreglo mapeador y NumBytes:PALABRA es la cantidad de bytes a transformar.

void SumaBloque(DirOrigen, DirDestino, NumBytes).

El procedimiento promedia dos subimágenes puntualmente y se utiliza en la suma de dos imagenes. No regresa ningún valor. DirOrigen:APTRLEJOS es la dirección del primer bloque; DirDestino:APTRLEJOS es la dirección del segundo bloque y lugar donde se almacena el resultado del promedio y NumBytes:**int** es el número de elementos en el bloque.

void RestaBloque(DirOrigen, Dirdestino, NumBytes).

El procedimiento encuentra la diferencia entre dos elementos en una subimagen. Normaliza el intervalo resultante al intervalo [0..255] linealmente. No regresa valor alguno. DirOrigen:APTRLEJOS es la dirección del primer bloque; DirDestino:APTRLEJOS es la dirección del segundo bloque y lugar donde se almacena el resultado de la diferencia y NumBytes:**int** es el número de elementos en el bloque.

void DivideBloque(DirOrigen, Dirdestino, NumBytes).

El procedimiento realiza el cociente de dos bloques elemento a elemento. No regresa ningún valor. DirOrigen:APTRLEJOS es la dirección del primer bloque; DirDestino:APTRLEJOS es la dirección del segundo bloque y lugar donde se almacena el resultado del promedio y NumBytes:**int** es el número de elementos en el bloque.

void FiltraSubImg(DirOrigen, DirFiltro, RensFiltrar, ColsFiltrar, AltoFiltro, AnchoFiltro).

El procedimiento filtra una subimagen. No regresa ningún valor. DirOrigen:APTRLEJOS **es la dirección en la que se encuentra el bloque;** DirFiltro:APTRCERCA **es la direccióne del filtro;** RensFiltrar:**int es la altura del bloque a filtrar;** ColsFiltrar:**int es el ancho del bloque a filtrar;** AltoFiltro:**int es la altura del filtro y** AnchoFiltro:**int es el ancho del mismo.**

void ExtrapolaBlq(DirOrigen, AltoBlq, AnchoBlq, RenExtra, ColExtra).

El procedimiento ejecuta dos tareas fundamentales:

1: Dado un bloque de datos lo traslada a otra posición en memoria y lo expande, es decir, genera renglones más largos que los originales, creando un bloque de datos mayor que el original.

2: Ampliado el bloque extrapola los datos de la frontera del mismo copiandolos repetidamente tantas veces como las dimensiones del filtro.

Los parámetros son los siguientes: DirOrigen:APTRLEJOS **es la dirección donde se encuentra el bloque originalmente;** AltoBlq:**int es el número de renglones del bloque;** AnchoBlq:**int es el ancho del bloque;** RenExtra:**int es el número de renglones del filtro / 2 y** ColExtra:**int es el número de columnas del filtro / 2.**

void ReduceSubImg(Dirorigen,Renglones,Columnas).

El procedimiento reduce una subimagen descartando elementos de la misma. Sobre cada renglón desprecia un elemento y toma el siguiente. De cada dos renglones toma únicamente uno. Los parámetros son los siguientes: DirOrigen:APTRLEJOS **es la dirección donde se encuentra la subimagen que se va a reducir;** Renglones:**int es el número de renglones del bloque a reducir y** Columnas:**int es el ancho del mismo.**

void ExtraeSubBanda(DirOrigen, RensBanda, ColsBanda, Banda, NumBandas).

El procedimiento extrae una banda de una subimagen multiespectral. Los parámetros son: DirOrigen:APTRLEJOS **es la dirección base de la subimagen multiespectral;** RensBanda:**int es el número de renglones de la subimagen** ColsBanda:**int es el ancho de un renglón;** Banda:**int es el número de banda que se desea extraer;** NumBandas:**int es el número de bandas de que consta la imagen multiespectral.**

void ComponeSubImg(DirOrigen, RensSubImg, ColsSubImg, NumBandas, BandaRojo, BandaVerde, BandaAzul).

El procedimiento crea una subimagen a partir de tres bandas, consideradas como RR VVV AAA, en donde la R es rojo, la V es verde y la A es azul. Los parámetros que se le pasan son: DirOrigen:APTRLEJOS **es la dirección base de la subimagen multiespectral** RensSubImg:**int es el número de renglones de la subimagen; ColsSubImg:int es el ancho de un renglón;** NumBandas:**int es el número de bandas de que consta la imagen multiespectral;** BandaRoja:**int**

es el número de banda a la que se le asignará el rojo; BandaVerde:**Int** es el número de banda, en la imagen multiespectral, a la que se le asignará el color verde y BandaAzul:**Int** es la banda que se considerará del azul.

5.5 Subrutinas de punto flotante, spidflot.h

Existe un conjunto de funciones y subrutinas que debe ejecutar SPID de punto flotante, en donde un emulador resulta muy ineficiente por lo complicado de las operaciones, por lo tanto, se hace necesario el uso de un coprocesador matemático que maneje todas las operaciones de punto flotante.

SPID utiliza el 8087 para soporte numérico, de esta manera se logra una mayor eficiencia en tareas que requieren de una gran cantidad de operaciones. En esta sección aparece la mayor parte de las funciones y procedimientos que utiliza el coprocesador.

PALABRA Completar(ValorACompletar).

La función genera un número potencia de 2 a partir del parámetro de entrada, ValorACompletar:PALABRA. Si el parámetro es potencia de 2, entonces el resultado es este mismo. Si es mayor que una potencia de 2, es decir, es de la forma: $2^z + a$, entonces el resultado es 2^{z+1}.

Int LogBase2(ValLog2).

La función calcula el logaritmo en base 2 de una potencia de 2, ValLog2 : PALABRA. Regresa este valor.

void Traspone(DirOrigen, DirDestino, Renglones, Columnas).

El procedimiento traspone un arreglo rectangular de datos complejos que se encuentra en memoria, en la dirección DirOrigen:APTRLEJOS a la dirección DirDestino:APTRLEJOS. Para trasponer los elementos se utiliza la formula:

$$(i,j) = in + j \Rightarrow (j,i) = jm + i.$$

En donde m es el número de renglones y n es la longitud de un renglón. No regresa ningún valor. renglónes:**Int** es el alto del área rectangular o número de renglónes y Columnas:**Int** es el ancho de la misma.

void Expande(DirOrigen, Renglones, Columnas, LongNueva).

El procedimiento se encarga de distribuir los datos de una imagen en un segmento de memoria, de tal manera que cada byte que ocupaba una localidad de memoria ahora ocupará 8. Se convierte en número complejo y ocupa 4 bytes la parte real más 4 de la parte imaginaria. Este procedimiento transforma un bloque de bytes a un bloque de números complejos. Todo lo hace en el mismo segmento de memoria. No regresa ningún valor. DirOrigen:APTRLEJOS es la dirección base del segmento a expander; Renglones:**Int** es el número de

renglones o alto de la subimagen NumCols:**Int**es el número de columnas o ancho de la subimagen LongNueva:**Int** es el ancho a que se desea ampliar cada renglón.

void TransfBloque(DirOrigen, Renglones, Columnas, L2Columnas).

El procedimiento utiliza el algoritmo de la transformada de Fourier para transformar una subimagen que se encuentra en memoria principal. Transforma renglón a renglón. No regresa ningún valor. DirOrigen:APTRLEJOS es la dirección base del segmento a transformar; Renglones:Int número de renglones de la subimagen a transformar; Columnas:Int es el número de columnas o ancho de la subimagen a transformar y L2Columnas:Int es el logaritmo en base 2 del ancho del renglón.

void AmplMaxMin(DirOrigen, Renglones, Columnas).

El procedimiento calcula la amplitud de la transformada de Fourier de una subimagen en memoria. Encuentra los valores máximo y mínimo de las amplitudes, valores que posteriormente utilizará en el mapeo de las amplitudes al intervalo [0,255]. No regresa valor alguno. DirOrigen:APTRLEJOS es la dirección base del segmento en el cual se encuentran los valores sobre los cuales calcula el máximo y el mínimo; Renglones:Int es el número de renglones de la subimagen y Columnas:Int es el número de columnas de la subimagen.

void EspectroImg(DirOrigen, Renglones, Columnas).

El procedimiento calcula el mapeo del espectro de un bloque correspondiente a una subimagen. DirOrigen:APTRLEJOS es la direccion base del bloque; Renglones:Int es el número de renglones de una subimagen Columnas:Int es el número de columnas de una subimagen.

5.6 Subrutinas de alto nivel, spidlib.h

Algunas funciones y procedimientos se escribieron en el lenguaje C ya que se componen de una o más funciones básicas, como las que hemos analizado hasta ahora. En esta sección se muestran los encabezados de estas funciones.

Int DameLetra( ).

La función lee del teclado un caracter y regresa un entero con las siguientes características: a) Si la tecla presionada es normal, ésta se regresa en la parte alta de la función y b) Si la tecla presionada es de código extendido, la parte alta es cero y la parte baja es el código correspondiente.

void LeeCadenaXY(X, Y, DirOrigen).

La función lee una cadena de caracteres. Regresa un apuntador a la cadena en DirOrigen:APTRCERCA. Se le pasan como parámetros las coordenadas (X:BYTE, Y:BYTE) del punto en la pantalla donde se lee la cadena.

char LeeCarSNXY(X,Y).

La función lee un caracter y sólo acepta que éste sea 'S', 's', 'N' o 'n'. Se le pasan de parámetros las coordenadas (X:BYTE, Y:BYTE) del punto en la pantalla donde se lee el caracter. Regresa el código ASCII del caracter leído.

float LeeRealXY(DirVentana, X, Y, Longitud).

La función lee un número real, verificando que su formato sea correcto. Regresa el número leído. Se le pasan como parámetros un apuntador DirVentana:APTRVENT a la ventana en la que se lee el número; las coordenadas (X:BYTE, Y:BYTE) del punto en la pantalla; y la cantidad máxima Longitud:BYTE de caracteres de que se puede componer el número.

Int LeeEnteroXY(DirVentana, X, Y, Longitud).

La función lee un número entero, verificando que su formato sea correcto. Regresa el número leído. Los parámetros que se le pasan son exactamente los mismos que para la lectura de un número real.

void MarcoTrj(X1, Y1, X2, Y2, ColorMarco).

El procedimiento despliega un rectángulo en la pantalla de alta resolución. No regresa ningún valor. Los parámetros son las coordenadas (X1:**Int**, Y1:**Int**) y (X2:**Int**, Y2:**Int**) de la esquina superior izquierda e inferior derecha del mismo, finalmente ColorMarco:BYTE es el color con que se desea desplegar el marco.

void MarcoRellenoTrj(X1, Y1, X2, Y2, ColorMarco, ColorFondo).

La subrutina traza un marco y lo rellena de un color dado. Los parámetros son los mismos que para un rectángulo normal —subrutina anterior—, excepto que ahora se le agrega el color del fondo ColorFondo:BYTE con que se rellena el marco.

void RellenaTrj(X, Y, ColorARellenar, ColorFrontera).

La subrutina rellena un área de la pantalla usando la tarjeta de alta resolución. Como parámetros se le pasan: Las coordenadas (X:**Int**, Y:**Int**) del punto inicial de partida para rellenar, este punto debe ser interior al área; el color con que se va a rellenar, ColorARellenar:BYTE y, finalmente, el color de la frontera del área, ColorFrontera:BYTE.

void EscTextTrj(Columna, renglón, ColorTex, TipoFont, Estilo, Tamano, AptrTexto).

La subrutina escribe una cadena de texto en la pantalla utilizando la tarjeta de alta resolución. No regresa ningún valor. Los parámetros que se le pasan son los siguientes: Columna:**Int** es la abscisa a partir de la cual se desplegará el texto; renglón:**Int** es el renglón a partir del cual se desplegará el texto; ColorTex:BYTE es el color del texto; TipoFont:**Int** es el tipo de font que se desea utilizar para el despliegue, si es 0, entonces es el font predefinido en la tarjeta de despliegue, si es 1, entonces el font desplegado es el definido por el usuario; Estilo:**Int** es la forma en que se despliega el texto, si es 0, el texto es delgado, si es 1, el texto es grueso; Tamano:**Int** es el tamaño del texto

desplegado, puede variar desde 1 hasta 20mm.; AptrTexto:APTRCERCA es la dirección en donde se encuentra el texto a desplegar.

void Marco(X, Y, Alto, Ancho, Atributo).

El procedimiento despliega un marco en pantalla, utiliza un octeto de Atributo : BYTE para el mismo. No regresa ningún valor. Se le pasan de parámetros las coordenadas de la esquina superior izquierda (X:BYTE, Y:BYTE) y el Alto:BYTE y Ancho:BYTE del mismo.

void DespVentana(AptrV, Normal).

El procedimiento despliega una ventana. Muestra el marco y despliega las opciones. AptrV:VENTANA * es un apuntador a la ventana que se va a desplegar, el parámetro Normal:**int** es un valor que determina el tipo de ventana: 0 si es una ventana de ayuda —en este caso no coloca la barra separadora entre título y opciones—, si es 1, entonces la ventana es normal.

BYTE LeeOpcion(AptrV).

El procedimiento lee la opción deseada dentro de una ventana. Regresa 0 para la primer opción, 1 para la segunda, etc. AptrV:VENTANA * es un apuntador a la ventana en cuestión.

IMAGEN AbreImgXY(NomImgEnt, X, Y, AptrVent).

La función abre una imagen colocando el cursor en la posición (X:BYTE, Y:BYTE) en pantalla para preguntar su nombre. Se le pasa como parámetro el nombre de la imagen NomImgEnt:APTRCERCA. Si la imagen no existe, despliega un mensaje de error y pregunta por otro nombre. Regresa el identificador de la imagen cuando logra abrirla. El último parámetro es un apuntador AptrVent:APTRCERCA a la ventana en que se lee el nombre de la imagen.

IMAGEN CreaImgXY(NomImgEnt, X, Y, AptrVent).

La función crea una imagen si ésta no existe; de otra manera pregunta al usuario si desea borrarla, si éste es el caso entonces la borra, de otro modo pregunta por una nueva imagen. Los parámetros son exactamente los mismos que la función anterior.

int LeeDimConError(AptrVent, Img, MMD).

La función lee las dimensiones de una imagen y las compara con la longitud declarada de la misma, reportando un error, si no corresponden ambas dimensiones. Los parámetros son los siguientes: AptrVent:APTRVENT es un apuntador a una ventana desde la cual se lee la dimensión de la imagen; Img:IMAGEN es el identificador de la imagen y MMD:DIMEN far * es un apuntador a una estructura donde se lee la información.

Capítulo 6

Programa fuente SPID.C

A continuación se muestra el listado del programa fuente SPID.C para referencia.

```
#include <stdio.h>
#include <spid.def>
#include <spides.h>
#include <spidflot.h>
#include <spidgraf.h>
#include <spidgral.h>
#include <spidtrj.h>
#include <spidlib.h>


/* Pasos a seguir cuando se modifique una ventana:
        1: Actualizar los mensajes
        2: Actualizar las dimensiones de 'Mensaje'
        3: Actualizar el número de ventana, si se incluye una más
        4: Actualizar en 'ArregloV' los apuntadores
        5: Actualizar 'MatSal' y 'MatTrans'
*/
```

A continuación se muestran todos los mensajes que se utilizan en el manejo de menús y que aparecen en las ventanas de despliegue. Los comentarios que aparecen a la derecha de cada ventana (cada ventana está separada por un espacio vertical en blanco)

indican el número de ventana, el ancho en caracteres de la misma y el índice en el arreglo Mensaje.

```
char Mensaje[NUM_MSG][ANCHO_MSG] =          /* NumVentana, Ancho, Indice */
    {   "   MENU PRINCIPAL                          ",      /* 0 42 0 */
        "   Despliegue e histograma de imágenes     ",
        "   Funciones de utilería al usuario        ",
        "   Operaciones puntuales                   ",
        "   Operaciones de área                     ",
        "   Operaciones vectoriales                 ",
        "   Ayuda al usuario                        ",
        "   Salida del sistema < Esc >              ",

        "   FUNCIONES DE UTILERIA AL USUARIO        ",      /* 1 49 8 */
        "   Definición de tablas de LUT's           ",
        "   Convertir archivo de datos a imagen     ",
        "   Imprimir datos de una subimagen         ",
        "   Extraer una banda de una imagen multiespectral  ",
        "   Composición de tres bandas              ",
        "   Ayuda al usuario                        ",
        "   Regreso al nivel anterior < Esc >       ",

        "   OPERACIONES PUNTUALES                   ",      /* 2 37 16 */
        "   Suma de imágenes                        ",
        "   Resta de imágenes                       ",
        "   División de imágenes                    ",
        "   Normalización del histograma            ",
        "   Ayuda al usuario                        ",
        "   Regreso al nivel anterior <Esc>         ",

        "   OPERACIONES DE AREA                     ",      /* 3 34 23 */
        "   Filtros digitales                       ",
        "   Ampliación de imágenes                  ",
        "   Reducción de imágenes                   ",
        "   Ayuda al usuario                        ",
        "   Regreso al nivel anterior <Esc>         ",

        "   OPERACIONES VECTORIALES                 ",      /* 4 33 29 */
```

```
" Transformada de Fourier                    ",
" Ayuda al usuario                           ",
" Regreso al nivel anterior<Esc>             ",

" AMPLIACION DE IMAGENES                     ",   /* 5 38 33 */
" Doblamiento de la imagen                   ",
" Interpolación lineal                       ",
" Interpolación cúbica segmentaria           ",
" Ayuda al usuario                           ",
" Regreso al nivel anterior < Esc >          ",

" Nombre de la imagen:                                ",   /* 6 38 39 */

" Nombre de la imagen de entrada:                     ",   /* 7 48 40 */
" Alto de la imagen:                                  ",
" Ancho de la imagen:                                 ",

" Nombre de la imagen de entrada:                     ",   /* 8 48 43 */
" Nombre de la imagen de salida:                      ",

" Indice (en el archivo) de la tabla definida:        ",   /* 9 50 45 */

" Nombre de la imagen (1) de entrada:                 ",   /* 10 50 46 */
" Nombre de la imagen (2) de entrada:                 ",
" Nombre de la imagen resultante:                     ",

" Nombre de la imagen a filtrar:                      ",   /* 11 49 49 */
" Nombre de la imagen de salida:                      ",
" Alto del filtro:                                    ",
" Ancho del filtro:                                   ",
"                                                     ",

" Nombre de la imagen a imprimir:                     ",   /* 12 49 54 */
" Abscisa de la esquina superior izquierda:           ",
```

```
" Ordenada de la esquina superior izquierda:                    ",
" Alto de la subimagen a imprimir:                              ",
" Ancho de la subimagen a imprimir:                             ",


" Nombre de la imagen multiespectral:                           ",  /* 13 50 59 */
" Nombre de la imagen compuesta:                                ",
" Renglones en la imagen multiespectral:                        ",
" Ancho de cada renglón en la imagen:                           ",
" De cuántas bandas se compone la imagen:                       ",
" Banda asignada al rojo:                                       ",
" Banda asignada al verde:                                      ",
" Banda asignada al azul:                                       ",
" Contiene encabezado (S/N) ? :                                 ",
"                                                               ",


" Nombre de la imagen multiespectral:                           ",  /* 14 50 69 */
" Nombre de la imagen extraída:                                 ",
" Renglones en la imagen multiespectral:                        ",
" Ancho de cada renglón en la imagen:                           ",
" De cuántas bandas se compone la imagen:                       ",
" Cual banda se va a extraer:                                   ",
" Contiene encabezado (S/N) ? :                                 ",
"                                                               ",


"                                                               ",  /* 15 50 77 */


"                                                               ",  /* 16 50 78 */
"                                                               ",


" AYUDA. MENU PRINCIPAL                                         ",  /* 17 50 80 */
" En todos los programas que utilizan la tarjeta,               ",
" se termina su ejecucion presionando la tecla                  ",
" <Esc>, con lo cual se mostrará el menú anterior.              ",
" Sin embargo, a nivel de menús, esta tecla                     ",
" terminará el programa.                                        ",
" ==========                                                    ",
" Presione < Retorno > para continuar                           ",
```

```
" AYUDA. FUNCIONES DE UTILERIA AL USUARIO ",  /* 18 50 88 */
"  A las operaciones subyacentes a SPID que no ",
"  alteran el contenido de las imágenes se les ",
"  considera de ayuda al usuario. Estas opciones ",
"  son: ",
"  Definición de tablas de LUT's. Con esta opción ",
"  es posible crear tablas de colores (256) para ",
"  desplegar imágenes con estos. ",
"  Convertir archivo de datos a imagen. Dado un ",
"  conjunto de datos, se crea su histograma y se ",
"  genera un archivo con características de img. ",
"  Imprimir datos de una subimagen. Envia a la ",
"  impresora un conjunto de valores de elementos ",
"  una subimagen, dada su posición en la imagen. ",
"  ═══════════════════════════════════ ",
"  Presione < Retorno > para continuar ",

" AYUDA. OPERACIONES PUNTUALES ",  /* 19 50 104 */
" Se consideran puntuales aquellas operaciones en ",
" que la modificación de un elemento depende ",
" únicamente del mismo. Entre otras se han ",
" definido las operaciones aritméticas de suma, ",
" resta y división, las cuales requieren de dos ",
" imágenes; también se encuentra implantada la ",
" operación de normalización del histograma, fun- ",
" ción con la cual se logra una mejor distribución ",
" de los datos en una imagen. ",
" ═══════════════════════════════════ ",
" Presione < Retorno > para continuar ",

" AYUDA. OPERACIONES DE AREA ",  /* 20 50 116 */
" Las operaciones de área son aquellas en las que ",
" cada uno de los elementos de una imagen se ",
" modifica en función de los elementos vecinos. ",
" ═══════════════════════════════════ ",
" Presione < Retorno > para continuar ",

" AYUDA. OPERACIONES VECTORIALES ",  /* 21 50 122 */
" Estas operaciones se caracterizan por modificar ",
```

```
    "  un elemento en la imagen de acuerdo al valor de      ",
    "  un conjunto de ellos, ordenados en forma de          ",
    "  renglón o columna.                                   ",
    "  ═══════════════════════════════════                  ",
    "  Presione < Retorno > para continuar                  ",


    "  AYUDA. AMPLIACION DE IMAGENES                        ",  /* 22 50 129 */
    "  Hay tres maneras de ampliar una imagen: Por          ",
    "  doblamiento, consistente en repetir cada             ",
    "  elemento 4 veces, a la derecha, abajo y en la        ",
    "  diagonal abajo.                                      ",
    "  Por interpolación lineal entre dos elementos de      ",
    "  la imagen original, por renglones y columnas.        ",
    "  Por interpolación cúbica segmentaria, análoga        ",
    "  a la lineal, sólo que utilizando polinomios          ",
    "  cúbicos.                                             ",
    "  ═══════════════════════════════════                  ",
    "  Presione < Retorno > para continuar                  "};

VENTANA ArregloV[NUM_VENTANAS] =
{   {10,9,   42, 11, 112,79, 31, 0,  0},        /* 0 Menú principal */
    {25,12,  50, 11, 113,79, 47, 0,  8},        /* 1 Funciones de utileria */
    {40,6,   37, 10, 83, 79, 83, 0,  16},       /* 2 Operaciones puntuales */
    {35,13,  34, 9,  83, 79, 95, 0,  23},       /* 3 Operaciones de area */
    {39,12,  33, 7,  31, 103,89, 0,  29},       /* 4 Operaciones vectoriales */
    {18,15,  38, 9,  31, 103,112,0,  33},       /* 5 Ampliación de imágenes */
    {5, 20,  38, 3,ATR_VAR,0,101,0,39},     /* 6 */
    {5, 17,  48, 5,ATR_VAR,0,110,0,40},     /* 7 */
    {5, 19,  48, 4,ATR_VAR,0,111,0,43},     /* 8 */
    {5, 20,  50, 3,ATR_VAR,0,112,0,45},     /* 9 */
    {5, 18,  50, 5,ATR_VAR,0,114,0,46},     /* 10 */
    {5, 17,  49, 7,ATR_VAR,0,115,0,49},     /* 11 */
    {5, 14,  49, 7,ATR_VAR,0,115,0,54},     /* 12 */
    {5, 12,  51, 12,ATR_VAR,0,115,0,59},    /* 13 */
    {5, 14,  51, 10,ATR_VAR,0,115,0,69},    /* 14 */
    {X_MSG_ER,Y_MSG_ER,51,3,ATR_ER,207,ATR_ER-128,0,77},
                                    /* 15 Mensaje de error VENT_ERROR */
    {X_MSG_LN,Y_MSG_LN,51,4,ATR_MSG,0,ATR_MSG,0,78},
                                    /* 16 Mensaje de edo. del proceso VENT_MSG */
    {20,14,  51, 10, 31, 0,  101,0,  80},/* 17 Ayuda. Menú principal VENT_AYUDA */
```

```
    {18,5,  51,  18,  31,  0,   101,0,   88},      /* 18   " Func. Utileria */
    {14,11, 51,  14,  31,  0,   101,0,   104},     /* 19   " Oper. Puntuales */
    {22,7,  51,  8,   31,  0,   101,0,   116},     /* 20   " Oper. de área */
    {11,10, 51,  9,   31,  0,   101,0,   122},     /* 21   " Oper. vectoriales */
    {7,  10, 51,  14,  31,  0,   101,0,   129}};   /* 22   " Ampl. de imágenes */

BYTE MatTrans[ESTADOS][TIPOSCAR] =
{   {0,  1,  2,  3,  4,  0,  6  },
    {1,  1,  1,  1,  1,  1,  0  },
    {2,  2,  2,  2,  2,  0,  0  },
    {3,  5,  3,  3,  0,  3,  0  },
    {4,  4,  0,  4,  4,  4,  0  },
    {5,  5,  5,  5,  3,  5,  3  }  };

BYTE MatSal[ESTADOS][TIPOSCAR] =
{   { 1, 2,  3,  4,  5,  21, 0  },
    { 7, 9,  8,  19, 20, 22, 10 },
    {11,12, 13,  14, 23, 10, 10 },
    {15,16, 18,  24, 10, 0,  10 },
    {17,25, 10,  0,  0,  0,  10 },
    { 6, 28, 27, 26, 10, 0,  10 }  };

BYTE BloqueVent[MAX_BLOQUE_VENT];

APTRCERCA StackVent[MAX_TOPE];

float Filtro[ALTO_MAX_FIL * ANCHO_MAX_FIL] = {0};

CADENA NomImgEnt ="                                    ";
CADENA NomImgEnt1 ="                                   ";
CADENA NomImgSal ="                                    ";
CADENA CadenaAux ="                                    ";
APTRCERCA DirBaseVent =&BloqueVent[0];
APTRCERCA VentMsg =&BloqueVent[0];

main ( )
```

```
{
    BYTE Estado;
    BYTE Indice;
    BYTE Opcion;
    BYTE Salida;
    BYTE NumVent;
    BYTE TopeStkVent;


    ModoTextual(3);
    if (Checa8087() ≠ CERO) Error(0,CRITICO,"");
    Marco(0,0,25,80,63);
    EscCadAtr(1,1,78,"SISTEMA PARA EL PROCESAMIENTO
                      DE IMAGENES DIGITALES");
    EscCadAtr(4,3,62,"[redacted]");
    EscCadAtr(4,4,62,"[redacted]");
    EscCadAtr(4,5,62,"[redacted]");
    EscCadAtr(4,6,62,"[redacted]");
    EscCadAtr(4,7,62,"[redacted]");
    EscCadAtr(31,7,49,"A. Cortés");
    NumVent = 0;
    Salida = 0;
    TopeStkVent = 0;
    Opcion = 0;
    Estado = 0;
    do
    {
        MueveCursor(0,30);
        DespVentana(&ArregloV[NumVent],1);
        Opcion = LeeOpcion(&ArregloV[NumVent]);
        Salida = MatSal[Estado][Opcion];
        switch (Salida)
        {
/* Cada uno de los casos que se muestran a continuación corresponde a las salidas mostradas en la tabla 1.2, cuyo comportamiento lo define la matriz de salidas. */
        case 1: NumVent = DespImg();
                break;
        case 2:
        case 3:
        case 4:
        case 5:
```

```
            StackVent[TopeStkVent] = DirBaseVent;
            TopeStkVent++;
            NumVent = Salida - 1;
            DirBaseVent = SalvaVent(DirBaseVent,&ArregloV[NumVent]);
            break;
    case 6: NumVent = Amplía(0);
            break;
    case 7: NumVent = TablaDeLuts();
            break;
    case 8: NumVent = ImprimeImg();
            break;
    case 9: NumVent = ConvierteImg();
            break;
    case 10:
            TopeStkVent--;
            DirBaseVent = StackVent[TopeStkVent];
            RestableceVent(DirBaseVent,&ArregloV[Estado]);
            if (Estado == 5) NumVent = 3;
                    else NumVent = 0;
            break;
    case 11: NumVent = OperaImg(0);
            break;
    case 12: NumVent = OperaImg(1);
            break;
    case 13: NumVent = OperaImg(2);
            break;
    case 14: NumVent = NormalizaHisto();
            break;
    case 15: NumVent = Filtros();
            break;
    case 16:
            StackVent[TopeStkVent] = DirBaseVent;
            TopeStkVent++;
            NumVent = 5;
            DirBaseVent = SalvaVent(DirBaseVent,&ArregloV[NumVent]);
            break;
    case 17: NumVent = Fourier();
            break;
    case 18: NumVent = Reduce();
            break;
    case 19: NumVent = Extrae();
```

```
                break;
        case 20: NumVent = ComponeBanda();
                break;
        case 21: NumVent = Ayuda(VENT_AYUDA);
                break;
        case 22: NumVent = Ayuda(VENT_AYUDA+1);
                break;
        case 23: NumVent = Ayuda(VENT_AYUDA+2);
                break;
        case 24: NumVent = Ayuda(VENT_AYUDA+3);
                break;
        case 25: NumVent = Ayuda(VENT_AYUDA+4);
                break;
        case 26: NumVent = Ayuda(VENT_AYUDA+5);
                break;
        case 27: NumVent = Amplía(2);
                break;
        case 28: NumVent = Amplía(1);
        }
        Estado = MatTrans[Estado][Opción];
    }while (Estado ≠ 6);
    ModoTextual(0);
}
```

Capítulo 7

Conclusiones

El sistema presentado en esta tesis es una de las las herramientas fundamentales en el análisis de imágenes digitales, con la ventaja de modularidad que permitirá en un futuro cercano ampliarse tanto como sea necesario y de acuerdo a las necesidades de cada usuario. Inicialmente el sistema se ha desarrollado con un conjunto básico de operaciones, pero sin ninguna restricción para que este conjunto sea ampliado tanto como se desee. Siendo un sistema desarrollado en una microcomputadora tiene la enorme ventaja de poderse implantar a bajo costo con lo que amplía sus perspectivas de uso por un gran número de usuarios, además que la tendencia en la computación es hacia el uso de microcomputadoras.

Los sistemas existentes hasta la fecha tienen dos desventajas: La primera es que, si se trata de un sistema que realice procesamiento de imágenes a nivel profesional, éste estará implantado en equipos de cómputo muy sofisticados y de propósito especial, resultando muy caros, por este sólo hecho. En segundo lugar, los sistemas en microcomputadora son también muy caros y se convierten en cajas negras en el momento de sus adquisición. La programación de SPID crea hasta cierto punto independencia tecnológica. Además algunos sistemas en el mercado son más ineficientes en algunas operaciones comparados con SPID en cuanto a tiempo de ejecución o las tareas que realizan no se pueden considerar como de aplicación profesional debido a las restricciones impuestas tales como: tamaño fijo, pocos valores para representar un elemento de la imagen y pocas operaciones disponibles.

Una de las ventajas de SPID es la de poder contar con una tarjeta de alta resolución, la cual define algunas pautas a seguir en la implantación de un sistema. Con la ayuda de esta tarjeta puede considerarse a SPID como un sistema con características profesionales de procesamiento. Se ha considerado que los elementos de una imagen pertenecen al conjunto $Z_{[0...255]}$ con lo cual se logran imágenes desplegadas a 256 colores.

Se ha implementado una gran variedad de operaciones aritméticas, puntuales, de

área, vectoriales y estadísticas, para que el usuario haga uso de las mismas con un esfuerzo mínimo de su parte, debido al uso de ventanas y de menús. El sistema es autocontenido ya que contiene información en línea de las operaciones y opciones a realizar en cualquier momento dado, únicamente presionando la tecla de ayuda.

Por ahora, el sistema SPID es dependiente de una tarjeta de alta resolución, HLGE, cuyas características se han analizado a lo largo del trabajo, sin embargo, se piensa expander el sistema a tal grado que no dependa de la tarjeta gráfica. La ampliación será compatible con las tarjetas gráficas: VGA, EGA y aun con la CGA.

El sistema funciona con 256K octetos de memoria principal. El disco duro debe ser de las dimensiones requeridas por la aplicación a que se vaya a enfocar SPID, esto depende de la cantidad de datos en imágenes que se vaya a generar en un momento dado. Finalmente, no se requiere forzosamente un coprocesador matemático, pero se recomienda para que la velocidad de operación de SPID mejore. Se requiere, además un monitor a color, de preferencia que soporte alta resolución por si se utiliza la tarjeta HLGE.

Apéndice A.

Programas de la biblioteca

SPIDES.ASM

Las funciones y rutinas que se listan en este apéndice son para el manejo de la entrada y salida de datos entre memoria y disco y viceversa del sistema SPID. Se lista todo el archivo que compone la biblioteca completa de entrada/salida de todos los encabezados mostrados en la sección 5.3. Todas estas funciones y procedimientos están escritos en lenguaje ensamblador con base en el microprocesador 8086/8088, utilizando en algunas funciones el coprocesador matemático 8087/80287.

El formato será como el que se encuentra originalmente en el programa, sin cambio en los comentarios. Si el tipo de letra es romana, entonces los comentarios fueron agregados posteriormente.

En este momento cabe hacer notar que el sistema SPID utiliza dos segmentos de memoria de 64K *bytes* cada uno para almecenar datos temporales. Estos segmentos se les ha llamado SEG_IMG1 y SEG_IMG2, respectivamente. Se utilizó este esquema de memoria fija ya que la mayoría de las funciones requieren de bloques grandes de memoria y el manejo dinámico de la misma hubiese redundado en un mayor trabajo.

En casi todos las funciones y procedimientos de este apéndice y del resto de la biblioteca de funciones, cuyos encabezados se muestran en el capítulo 5, se utilizan los dos segmentos como áreas temporales de memoria. Al leer una subimagen, por ejemplo, se lee en uno de estos segmentos, si se requiere un proceso, éste se lleva a cabo en el segundo segmento, o viceversa.

Existen algunos casos en que se requieren más de dos segmentos, pero no simultáneamente, se puede aprovechar esta circunstancia para utilizar únicamente los dos disponibles, liberando el espacio de cualquiera una vez que deje de utilizarse.

A continuación se muestra que todas las subrutinas de este archivo son de uso

público, lo cual permite que todo el sistema SPID tenga acceso a ellas.

```
public      _LeeCar
public      _LeeCarEco
public      _LeeCadena
public      _LeeEntero
public      _LeeReal
public      _ErrorLectura
public      _ErrorDOS
public      _EscCar
public      _AbreImg
public      _CreaImg
public      _CierraImg
public      _BorraImg
public      _LeeLongImg
public      _LeeDimension
public      _LeeSubImg
public      _LeeSeq
public      _EscSubImg
public      _EscSeq

public      _CadenaEnt          ; Ya que otras subrutinas lo utilizan
```

El segmento de código debe llamarse _TEXT por compatibilidad con el lenguaje C. En el modelo pequeño de memoria, este lenguaje utiliza únicamente 64K *bytes* para el código de un programa, de aquí la necesidad de que cada segmento sea llamado como se muestra a continuación.

Se creó un conjunto de segmentos de datos y fueron agrupados bajo el nombre de DGROUP. Inicialmente el registro de segmento de datos, ds, se posiciona al inicio del grupo. Despues veremos que no todas las funciones mantinen el registro de datos apuntando a este grupo.

```
_TEXT       SEGMENT BYTE PUBLIC 'CODE'
            assume cs:_TEXT, ds:DGROUP
```

La mayor parte de funciones y procedimientos listados en este apéndice utilizan la función 21h del sistema operativo MS-DOS, como lo muestran las dos funciones siguientes. La primera función, _LeeCar, lee un caracter del teclado, sin mostrarlo en la pantalla, mientras que la segunda función, _LeeCarEco, sí lo despliega.

```
_LeeCar     proc  near
            mov   ah,7
            int   21h
            ret
```

```
_LeeCar      endp

_LeeCarEco   proc  near
             mov   ah,1
             int   21h
             ret
_LeeCarEco   endp
```

Las tres funciones que siguen utilizan la función _LeeCarEco anterior para leer una cadena de caracteres —_LeeCadena—, un número entero —_LeeEntero— y un número real —_LeeReal—. Posteriormente, estas funciones se utilizan, a su vez, dentro de ventanas para verificar si el dato leído es correcto y corresponde al formato del mismo. La función _LeeCadena utiliza un área de memoria direccionada por el parámetro que se le pasa de entrada. El final de la cadena se detecta con un caracter CR

```
_LeeCadena   proc  near
CadenaEntrada equ word ptr [bp+4]
             push  bp
             mov   bp,sp
             push  di
             mov   di,CadenaEntrada
             mov   cx,di
SigLeeCad:   call  _LeeCarEco
             cmp   al,8                ; Borrar caracter anterior
             jne   AnalizaCar
             cmp   di,cx
             je    SigLeeCad
             dec   di
             mov   ax,' '
             push  ax
             call  _EscCar
             pop   ax
             mov   ax,8
             push  ax
             call  _EscCar
             pop   ax
             jmp   SigLeeCad
AnalizaCar:  cmp   al,CERO
             jne   CarNormal
             call  _LeeCarEco
CarNormal:   cmp   al,CR               ;'?  Es fin de la cadena ?
             je    FinLeeCad
             mov   [di],al
             inc   di
             jmp   SigLeeCad
FinLeeCad:   mov   byte ptr [di],0
             pop   di
             mov   sp,bp
```

```
            pop   bp
            ret
_LeeCadena  endp

_LeeEntero  proc  near
            push  si
            mov   word ptr DGROUP:ErrorLect,0
            mov   ax,offset DGROUP:_CadenaEnt
            push  ax
            call  _LeeCadena          ; Cadena que representa el número
            add   sp,2
            mov   si,offset DGROUP:_CadenaEnt
            cld
            mov   bx,0                ; bx será el número leído
            mov   dl,0                ; dl es el signo (0) positivo
            lodsb                     ; (1) negativo
            cmp   al,'+'
            je    SigCarLeeEnt
            cmp   al,'-'
            jne   EnteroPos
            mov   dl,1                ; Se trata de un numero negativo
SigCarLeeEnt: lodsb
EnteroPos:  cmp   al,CERO
            je    FinLeeEnt
            cmp   al,'0'
            jl    ErrorLeeEnt
            cmp   al,'9'
            jg    ErrorLeeEnt
            xor   ah,ah
            sub   al,'0'
            mov   cx,bx
            shl   bx,1
            shl   bx,1
            add   bx,cx
            shl   bx,1
            add   bx,ax
            jmp   SigCarLeeEnt
ErrorLeeEnt: mov  word ptr DGROUP:ErrorLect,1
FinLeeEnt:  mov   ax,bx
            or    dl,dl
            jz    FuePos
            neg   ax
FuePos:     pop   si
            ret
_LeeEntero  endp
```

La función _LeeReal es un poco más sofisticada que las dos anteriores ya que acepta cualquier formato válido en el número que se desea leer, incluyendo notación exponencial. El formato válido para un número real es:

[±][Número Entero][.]Número Entero[e][E] ± [Número Entero]

en donde los corchetes indican que es opcional el parámetro que encierran. Si el formato del número leído no se ajusta al anterior, entonces la función enciende una bandera de error escribiendo un 1 en la localidad de memoria ErrorLect, valor que posteriormente sirve para hacer notar el error al usuario.

```
_LeeReal    proc  near

; Variables locales

Digito      equ   word ptr [bp-2]
Signo       equ   word ptr [bp-4]
ExpReal     equ   word ptr [bp-6]
SignoExp    equ   word ptr [bp-8]
ExpCor      equ   word ptr [bp-10]
Diez        equ   word ptr [bp-12]

            push  bp
            mov   bp,sp
            sub   sp,12
            push  si
            cld
            mov   ax,offset DGROUP:_CadenaEnt
            push  ax
            call  _LeeCadena          ; Cadena que representa el número
            add   sp,2
            mov   si,offset DGROUP:_CadenaEnt
            mov   ax,10
            mov   Diez,ax
            mov   Signo,0
            mov   SignoExp,0
            mov   ExpCor,0
            mov   ExpReal,0
            FLDZ
            mov   word ptr DGROUP:ErrorLect,0
            lodsb
            cmp   al,'+'
            je    LeeSigDigito
            cmp   al,'-'
            jne   PrimerDigito
            mov   Signo,1             ; Se trata de un número negativo
LeeSigDigito: lodsb
PrimerDigito: cmp  al,'0'
            jl    NoDigito1
            cmp   al,'9'
            jg    NoDigito1
            xor   ah,ah
```

```
                sub   ax,'0'
                mov   Digito,ax
                FIMUL Diez
                FIADD Digito
                jmp   LeeSigDigito
NoDigito1:      cmp   al,'.'
                je    SigEntero
                jmp   NoDigito2
SigEntero:      lodsb
                cmp   al,'0'
                jl    NoDigito2
                cmp   al,'9'
                jg    NoDigito2
                xor   ah,ah
                sub   ax,'0'
                mov   Digito,ax
                FIMUL Diez
                FIADD Digito
                inc   ExpCor
                jmp   SigEntero
NoDigito2:      cmp   al,'e'
                je    LeeExp
                cmp   al,'E'
                je    LeeExp
                jmp   FinLeeReal
LeeExp:         lodsb
                cmp   al,'+'
                je    SigDigExp
                cmp   al,'-'
                jne   PrimerDigExp
                mov   SignoExp,1
SigDigExp:      lodsb
PrimerDigExp:   cmp al,'0'
                jl    FinLeeReal
                cmp   al,'9'
                jg    FinLeeReal
                xor   ah,ah
                sub   ax,'0'
                mov   bx,ExpReal
                shl   bx,1
                shl   bx,1
                add   bx,ExpReal
                shl   bx,1
                add   bx,ax
                mov   ExpReal,bx
                jmp   SigDigExp
FinLeeReal:     cmp   al,CERO
                je    NoErrorReal
                mov   word ptr DGROUP:ErrorLect,1
                jmp   RetLeeReal
```

```
NoErrorReal:  mov   cx,Expreal
              cmp   SignoExp,0
              je    ExpPosit
              neg   cx
ExpPosit:     sub   cx,ExpCor
              or    cx,cx
              jz    RetornoReal
              jg    ExpPos
              neg   cx
AjustaDiv:    FIDIV Diez
              loop  AjustaDiv
              jmp   RetornoReal
ExpPos:       FIMUL Diez
              loop  ExpPos
RetornoReal:  cmp   Signo,0
              je    RetLeeReal
              FCHS
RetLeeReal:   FSTP  qword ptr DGROUP:TempoGral
              mov   ax,offset DGROUP:TempoGral
              pop   si
              mov   sp,bp
              pop   bp
              ret
_LeeReal      endp
```

Las siguientes dos funciones detectan un posible error en una operación de entrada/salida. _ErrorLectura regresa un valor de 1 si hubo un error en la lectura del último número leído, ya sea entero o real. Se puede ver en las funciones anteriores que en caso de un error, se escribe un 1 a la localidad de memoria ErrorLect, mismo que regresa la función _ErrorLectura, por otro lado, si no hay error de lectura en el número, la función regresa 0. _ErrorDos, por el otro lado, regresa un código que corresponde al error cometido en la ejecución de una función del sistema operativo, la tabla de errores se encuentra en el capítulo 3, ahí se puede ver a qué error corresponde, si no hay error el valor regresado es 0.

```
_ErrorLectura  proc near
               mov   ax,word ptr DGROUP:ErrorLect
               ret
_ErrorLectura endp

_ErrorDOS      proc  near
               mov   ax,word ptr DGROUP:ErrorDOS
               ret
_ErrorDOS      endp
```

En las funciones que siguen se nota el predominio en el uso de la función 21h del sistema operativo. La siguiente función escribe un caracter en la pantalla.

```
_EscCar     proc   near
Caracter    equ    byte ptr [bp+4]    ; Caracter a imprimir

            push   bp
            mov    bp,sp
            mov    dl,Caracter
            mov    ah,2
            int    21h
            mov    sp,bp
            pop    bp
            ret
_EscCar     endp
```

El resto de funciones son todas las que manejan grandes volúmenes de datos, entre disco y memoria y viceversa. Las primeras cuatro funciones —_AbreImg, _CreaImg, _CierraImg y _BorraImg— utilizan directamente una función del sistema operativo, con las características necesarias para el manejo de un archivo que compone una imagen. La tarea que realiza cada una de estas funciones es explícita por sí misma. En cada una de estas funciones se escribe inicialmente un 0 en la localidad de memoria ErrorDOS, indicando con esto que de entrada no hay ningún error en la operación de archivos. Si al ejecutarse la función se detecta un error, entonces se escribe su código a la localidad de memoria ya mencionada. Posteriormente el sistema verifica que este valor sea cero, de otro modo se lo hace notar al usuario.

```
_AbreImg    proc   near
NombreImg   equ    word ptr [bp+4]    ; Apuntador a nombre de imagen

            push   bp
            mov    bp,sp
            mov    DGROUP:ErrorDOS,0  ; No hay error
            mov    dx,NombreImg       ; Offset del nombre de la img.
            mov    ax,3D02h           ; Se abre para lectura y esc.
            int    21h
            jnc    SiAbrio
            mov    DGROUP:ErrorDOS,ax
SiAbrio:    mov    sp,bp
            pop    bp
            ret
_AbreImg    endp

_CreaImg    proc   near
NombreImg   equ    word ptr [bp+4]    ; Nombre de la imagen a crear

            push   bp
            mov    bp,sp
            mov    DGROUP:ErrorDOS,0  ; No hay error
            mov    dx,NombreImg       ; Offset del nombre de la img.
            mov    cx,20h             ; se crea para lectura y esc.
```

```
                mov     ah,3Ch
                int     21h
                jnc     SiCreo
                mov     DGROUP:ErrorDOS,ax
SiCreo:         mov     sp,bp
                pop     bp
                ret
_CreaImg        endp

_CierraImg      proc    near
FileHandle      equ     word ptr [bp+4]     ; Identificar de la imagen

                push    bp
                mov     bp,sp
                mov     DGROUP:ErrorDOS,0 ; No hay error
                mov     bx,FileHandle       ; Identificador de la imagen
                mov     ah,3Eh
                int     21h
                jnc     SiCerro
                mov     DGROUP:ErrorDOS,ax
SiCerro:        mov     sp,bp
                pop     bp
                ret
_CierraImg      endp

_BorraImg       proc    near
NombreImg       equ     word ptr [bp+4]     ; Identificar de la imagen

                push    bp
                mov     bp,sp
                mov     DGROUP:ErrorDOS,0 ; No hay error
                mov     dx,NombreImg        ; Identificador de la imagen
                mov     ah,41h
                int     21h
                jnc     SiBorro
                mov     DGROUP:ErrorDOS,ax
SiBorro:        mov     sp,bp
                pop     bp
                ret
_BorraImg       endp
```

La siguiente función calcula la longitud de una imagen tomando en consideración la parte final de la misma. Esta información es la que define las dimensiones de la imagen, elementos mínimo y máximo de la imagen y su histograma.

```
_LeeLongImg     proc    near
FileHandle      equ     word ptr [bp+4]
                push    bp
```

```
                mov     bp,sp
                mov     bx,FileHandle       ; Identificador de la imagen
                xor     cx,cx
                xor     dx,dx
                mov     ax,4202h            ; Calcula la longitud de la imagen
                int     21h
                push    ax
                push    dx
                mov     cx,0
                mov     dx,0
                mov     ax,4200h            ; Se posiciona al inicio de
                int     21h                 ; la imagen
                pop     dx
                pop     ax
                mov     sp,bp
                pop     bp
                ret
_LeeLongImg     endp
```

La siguiente función lee la información relacionada con la imagen, ésta no lee el histograma, sólo las dimensiones y los elementos mínimo y máximo de la misma. Esta rutina y la anterior sirven para verificar el formato de una imagen de la siguiente manera: La función _LeeLongImg anterior calcula el tamaño físico del archivo que contiene la imagen, junto con su información, si a este valor se le resta el número de bytes (TAM_INF_IMG + TAM_HISTO) que ocupa esta información, se obtiene el tamaño neto de la matriz de datos que forma la imagen.

Por el otro lado, la función _LeeDimension lee las dimensiones de la imagen (Alto×Ancho). El producto de estas dimensiones debe coincidir con la obtenida por el método anterior, de otro modo, existe una anomalía en el formato de la imagen.

```
_LeeDimension   proc near
FileHandle      equ     word ptr [bp+4]     ; Identificador de la imagen
DimenImg        equ     dword ptr [bp+6]    ; Apuntador FAR a la estructura

                push    bp
                mov     bp,sp
                mov     bx,FileHandle       ; Identificador de la imagen
                xor     cx,cx
                xor     dx,dx
                mov     ax,4202h            ; Calcula la longitud de la imagen
                int     21h
                sub     ax,TAM_INF_IMG
                sbb     dx,0
                mov     cx,dx
                mov     dx,ax
                mov     ax,4200h            ; Se posiciona al inicio de la
                int     21h                 ; información de la imagen
```

```
            mov   cx,TAM_INF_IMG
            push  ds                    ; Segmento y offset de la
            lds   dx,DimenImg           ; dirección de transferencia
            mov   ah,3Fh
            int   21h
            pop   ds
            mov   cx,0
            mov   dx,0
            mov   ax,4200h              ; Se posiciona al inicio de
            int   21h                   ; la imagen
            mov   sp,bp
            pop   bp
            ret
_LeeDimension endp
```

La función siguiente es una de las más importantes de SPID ya que en todos los programas se utiliza, por lo que requiere de una explicación más detallada, la cual damos a continuación.

Con las coordenadas del punto superior izquierdo de la subimagen a leer, calcula la posición inicial en disco y lee un segmento de datos de 64K. El proceso de lectura es el siguiente: Al posicionarse el apuntador al inicio del bloque a ser leído, se lleva a cabo la lectura, llevando a memoria todas las columnas comprendidas en el bloque cuya altura es igual al número de renglones de la imagen que caben en el segmento de 64K. Si el ancho de la subimagen a leer es menor que el ancho de la imagen total, se desechan en memoria las columnas que sobran, compactando el bloque leído. Es tarea del programa que la invoca verificar si requiere más datos, en cuyo caso ejecuta nuevamente esta función, hasta completar el conjunto de datos requerida. Normalmente las subimágenes que se requieren son menores o iguales a esta cantidad.

```
_LeeSubImg  proc  near
Imagen      equ   word ptr [bp+4]    ; Identificador de la imagen
BaseMem     equ   dword ptr [bp+6]   ; Aptr FAR a donde se transfiere
Ancho       equ   word ptr [bp+10]   ; Ancho de la imagen
RefY        equ   word ptr [bp+12]   ; Ordenada de la esquina sup.
RefX        equ   word ptr [bp+14]   ; izq.  formada por (RefX,RefY)
NumCols     equ   word ptr [bp+16]   ; Ancho de la subimagen

; Variables locales

Diferencia  equ   word ptr [bp-2]    ; Diferencia = Ancho - NumCols

            assume ds:SEG_IMG2, es:SEG_IMG2
            push  bp
            mov   bp,sp
            sub   sp,2
            push  ds
            push  es
            push  si
```

```
                push    di
                mov     ax,Ancho            ; Calcula la diferencia entre el
                sub     ax,NumCols          ; ancho de la imagen en disco y
                mov     Diferencia,ax       ; el ancho de la subimagen.
                mov     bx,Imagen           ; Identificador de la imagen
                mov     ax,Ancho            ; Calcula la posición inicial
                mul     RefY                ; del apuntador a la imagen.
                add     ax,RefX             ; LongX*RefY -> dx:ax
                adc     dx,0                ; LongX*RefY+RefX -> dx:ax
                mov     cx,dx               ; Asigna la posición del
                mov     dx,ax               ; apuntador.
                mov     ax,4200h            ; función 42 y apuntador
                int     21h                 ; MUEVE EL APUNTADOR A LA IMG.
                mov     cx,0FFFFh           ; Bytes a leer.  La dirección de
                lds     dx,BaseMem          ; transferencia es DS:DX.
                push    ds
                pop     es
                mov     ah,3Fh              ; En AX regresara el numero de
                int     21h                 ; bytes realmente leídos.
                push    ax
                div     Ancho               ; AX = Renglones de la subimagen.
                mov     cx,ax
                jcxz    FinDeSubImg         ; Sólo si es fin de imagen.
                mov     dx,NumCols          ; Verifica si el ancho de la sub-
                cmp     dx,Ancho            ; imagen es igual a la imagen en
                je      FinDeSubImg         ; disco, en cuyo caso no mueve
                mov     si,0                ; Inicializa los apuntadores de
                mov     di,0                ; transferencia de datos.
                cld
MueveRen:       push    cx
                mov     cx,NumCols          ; Este ciclo mueve los datos, com-
                rep     movsb               ; pactando la subimagen leída.
                add     si,Diferencia
                pop     cx
                loop    MueveRen
FinDeSubImg:    pop     ax
                pop     di
                pop     si
                pop     es
                pop     ds
                mov     sp,bp
                pop     bp
                ret
_LeeSubImg      endp
```

La función _LeeSeq es muy parecida a la anterior, excepto que esta última lee a partir de la posición del apuntador al archivo. Normalmente se utiliza en procesos en los cuales se leerán todas las columnas de la imagen y cuando el proceso se lleva a cabo secuencialmente.

```
_LeeSeq     proc   near
Imagen      equ    word ptr [bp+4]    ; Identificador de la imagen
BaseMem     equ    dword ptr [bp+6]   ; Area de transferencia en mem.
NumBytes    equ    word ptr [bp+10]   ; Bytes a leer

            push   bp
            mov    bp,sp
            push   ds
            mov    bx,Imagen          ; Identificador de la imagen
            mov    cx,NumBytes        ; Bytes a leer.  La dirección de
            lds    dx,BaseMem         ; transferencia es DS:DX.
            mov    ah,3Fh             ; En AX regresara el numero de
            int    21h                ; bytes realmente leidos.
            pop    ds
            mov    sp,bp
            pop    bp
            ret
_LeeSeq     endp
```

La siguiente función es de importancia fundamentalen el sistema SPID, y con una dificultad mayor que la de lectura de datos, por lo que requiere una doble atención.

La escritura de una subimagen a disco tiene una característica dual: Por un lado debe ser lo más rápido posible, y por el otro, debe hacerse con el mínimo número de accesos a disco.

Inicialmente, esta función se habia programado de tal manera que por cada renglón de la subimagen que se escribía se movía el apuntador a la imagen, calculando la posición inicial de cada renglón se escribía éste. Sin embargo, este proceso es muy lento por dos razones: La primera, el movimiento del apuntador en la imagen es lento y la segunda, se escribía renglón por renglón, lo cual acrecentaba el acceso a disco enormemente con un gasto de tiempo excesivo.

La versión que se presenta aquí es un poco más sofisticada y se atacan los dos problemas anteriores al mismo tiempo. La descripción es la siguiente: Sólo una vez se posiciona el apuntador en el archivo, a continuación se lee un segmento de 64K *bytes* a memoria. Ya en memoria se actualizan las columnas de ese bloque de datos con los que se van a escribir a disco, hecho ésto, lo cual es mucho más rápido ya que se actualizan en memoria los renglones necesarios, se escribe el bloque de datos actualizado a disco. El proceso anterior se repite tantas veces como sea necesario hasta escribir toda la subimagen.

Cabe decir que este método no es lo suficientemente rápido en el caso especial en que se desee escribir una subimagen cuya altura es igual a la altura de la imagen. La razón es que para escribirla se requiere leer, actualizar y escribir toda la imagen.

```
_EscSubImg  proc   near
Imagen      equ    word ptr [bp+4]    ; Identificador de la imagen
BaseMem     equ    dword ptr [bp+6]   ; Aptr FAR de donde se transfiere
Ancho       equ    word ptr [bp+10]   ; Ancho de la imagen
RefY        equ    word ptr [bp+12]   ; Ordenada de la esquina sup.
```

```
RefX        equ   word ptr [bp+14]   ; izq. formada por (RefX,RefY)
NumRens     equ   word ptr [bp+16]   ; Renglones de la subimagen
NumCols     equ   word ptr [bp+18]   ; Ancho de la subimagen

; Variables locales

Diferencia  equ   word ptr [bp-2]
RensFaltan  equ   word ptr [bp-4]
RensLeer    equ   word ptr [bp-6]
Rens64K     equ   word ptr [bp-8]
BytesLeerEsc equ  word ptr [bp-10]
ParteAlta   equ   word ptr [bp-12]
ParteBaja   equ   word ptr [bp-14]
ErEsc       equ   word ptr [bp-16]

; Este procedimiento utiliza el segmento SEG_IMG2 como segmento auxiliar
; en la escritura de un bloque de datos a una imagen, por lo que se debe
; tener cuidado que no se traslape el área de datos que se desea escribir
; con este segmento.

            assume es:SEG_IMG2
            push  bp
            mov   bp,sp
            sub   sp,16
            push  di
            push  si
            push  ds
            push  es
            mov   ErEsc,0           ; No hay error, el disco tiene
            lds   si,BaseMem        ; espacio suficiente.
            mov   ax,SEG_IMG2
            mov   es,ax
            mov   bx,Imagen         ; En todo el subprograma BX guarda
            mov   ax,NumRens        ; el identificador de la imagen
            mov   RensFaltan,ax     ; RensFaltan = NumRens
            mov   dx,0              ; Calcula el numero de renglones
            mov   ax,0FFFFh         ; que caben en un segmento de 64K
            div   Ancho
            mov   Rens64K,ax
            mov   ax,Ancho          ; Calcula la diferencia entre el
            sub   ax,NumCols        ; ancho de la imagen en disco y
            mov   Diferencia,ax     ; la subimagen

; Posiciona el apuntador del archivo de entrada al inicio del primer
; renglón sobre el cual se escribirá la subimagen.

            mov   ax,RefY
            mul   Ancho
            mov   cx,dx
            mov   dx,ax
```

```
                mov   ParteAlta,cx        ; Apuntador a la imagen =
                mov   ParteBaja,dx        ; ParteAlta * 65536 + ParteBaja
                mov   ax,4200h
                int   21h

                cld
OtroSeg:        mov   ax,Rens64K          ; Calcula los renglones que faltan
                cmp   ax,RensFaltan       ; por escribir, con la formula
                jle   NoAcaba             ; RensLeer=MIN(Rens64K,RensFaltan)
                mov   ax,RensFaltan
NoAcaba:        mov   RensLeer,ax
                mul   Ancho
                mov   BytesLeerEsc,ax     ; Calcula los bytes a escribir

; Lee un segmento de 64K bytes.  La dirección de transferencia es a
; donde apunte el par de registros DS:DX.

                mov   cx,ax               ; CX = Bytes a leer
                mov   dx,0
                push  ds
                push  es
                pop   ds
                mov   ax,3F00h
                int   21h
                pop   ds
                or    ax,ax
                jz    LeeNada
                mov   cx,ParteAlta        ; Reposiciona el apuntador de la
                mov   dx,ParteBaja        ; imagen al inicio del primer dato
                mov   ax,4200h            ; CX:DX posición que se desea
                int   21h                 ; Regresa DX:AX posición final

; Se actualiza el segmento con los datos de la subimagen que se va
; a escribir.

LeeNada:        mov   di,RefX
                mov   dx,RensLeer
MueveOtroRen    mov   cx,NumCols
                rep   movsb
                add   di,Diferencia
                dec   dx
                or    dx,dx
                jz    FinMueveOtro
                jmp   MueveOtroRen

; Se procede a escribir el segmento actualizado

FinMueveOtro    mov   cx,BytesLeerEsc
                mov   ax,4000h
                push  ds
```

```
                push    es
                pop     ds
                int     21h
                pop     ds

; Si el espacio en disco es insuficiente, el valor regresado en AX no
; corresponderá al que se pidió en CX.

                cmp     ax,cx
                je      HayEspacio
                mov     ErEsc,1
                jmp     FinEscS
HayEspacio:     add     ParteBaja,ax        ; Apuntador a la imagen +=
                adc     ParteAlta,0         ; BytesLeerEsc
                mov     ax,RensLeer
                sub     RensFaltan,ax       ; RensFaltan -= RensLeer
                cmp     RensFaltan,0
                jle     FinEscS
                jmp     OtroSeg
FinEscS:        pop     es
                pop     ds
                assume  ds:DGROUP
                cmp     ErEsc,1
                jne     FinNormal
                mov     word ptr DGROUP:ErrorDOS,14 ; Error no usado por DOS
FinNormal:      pop     si
                pop     di
                mov     sp,bp
                pop     bp
                ret
_EscSubImg      endp
```

La función _EscSeq escribe secuencialmente una subimagen a disco. Como puede deducirse, el ancho de la subimagen a escribir debe ser del ancho de la imagen, de otra manera no se puede utilizar esta función y debe emplearse la anterior.

```
_EscSeq         proc    near
Imagen          equ     word ptr [bp+4]     ; Identificador de la imagen
BaseMem         equ     dword ptr [bp+6]    ; Area de transferencia en mem.
NumBytes        equ     word ptr [bp+10]    ; Bytes a leer

                push    bp
                mov     bp,sp
                push    ds
                mov     bx,Imagen           ; Identificador de la imagen
                mov     cx,NumBytes         ; Bytes a escribir. La dirección
                lds     dx,BaseMem          ; de transferencia es DS:DX.
                mov     ah,40h              ; En AX regresara el numero de
```

```
            int   21h                ; bytes realmente escritos.
            pop   ds

; Si el espacio en disco es insuficiente, el valor regresado en AX no
; corresponderá al que se pidió en CX.

            assume ds:DGROUP
            cmp   ax,cx
            je    HayEspacioE
            mov   word ptr DGROUP:ErrorDOS,14 ; Error no usado por DOS
HayEspacioE: mov  sp,bp
            pop   bp
            ret
_EscSeq     endp
_TEXT       ENDS
```

El segmento de datos se llama _DATA y forma parte del grupo DGROUP. El siguiente segmento de datos contiene 4 variables, como se muestra. Estas variables se utilizan en esta biblioteca de funciones.

```
_DATA       SEGMENT WORD PUBLIC 'DATA'
TempoGral   dq    ?
_CadenaEnt  db    50 dup(?)
ErrorLect   dw    0
ErrorDOS    dw    0
_DATA       ENDS
            END
```

Apéndice B.

Constantes y tipos de SPID

En este apéndice se darán las definiciones de constantes utilizadas por SPID y de los tipos de variables utilizadas, así como las variables globales a todo el sistema.

```
#ifndef  SPIDDEF
#define  SPIDDEF        1

                    /*  DEFINICION DE SIMBOLOS  */

#define  CERO           0
#define  CR             13
#define  ESC            27

            /*  DEFINICION DE CONSTANTES GENERALES  */

#define  LNGREGLUT      512     /*  Longitud de un registro de LUT          */
#define  TAM_INF_IMG    6       /*  Tamaño de los datos de una imagen       */
#define  TAM_HISTO      1024    /*  Tamaño del histograma en bytes          */
#define  LONGENCAB      128     /*  Long. del encabezado de una img. mult.  */
#define  SEGMENTO       0xFFFF  /*  64K - 1 de memoria                      */
#define  SEGMENTOS4     0x3FFF  /*  16K - 1 de memoria                      */
#define  SEGMENTOS8     0x1FFF  /*  8K - 1 de memoria                       */
#define  VENT_AYUDA     17      /*  Primera ventana de ayuda                */
#define  VENT_ERROR     15      /*  Ventana de despliegue de errores        */
#define  VENT_MSG       16      /*  Ventana de mensajes                     */
#define  NUM_MSG        141     /*  Mensajes de las ventanas                */
#define  ANCHO_MSG      51      /*  Ancho de cada mensaje                   */
#define  ATR_VAR        118     /*Atributo de las ventanas donde se leen datos */
#define  ATR_MSG        79      /*  Atributo de la vent. de mensajes        */
```

```
#define  ATR_ER          207     /*  Atributo del mensaje de error          */
#define  X_MSG_LN        20      /*  Posición de la ventana de mensajes     */
#define  Y_MSG_LN        18
#define  X_MSG_ER        3       /*  Posición de la ventana de errores      */
#define  Y_MSG_ER        5
```

/* DEFINICION DE CONSTANTES UTILIZADAS EN LA PANTALLA DE DESPLIEGUE */

```
#define  ALTOPAN         478     /*  Renglones desplegados de la imagen     */
#define  ANCHOPAN        540     /*  Columnas desplegadas de la imagen      */
#define  COLIZQ          1       /*  Abscisa de la esq. inf. izq. en pantalla */
#define  COLDER          540     /*  Abscisa de la esq. sup. der. en pantalla */
#define  RENINF          1       /*  Ordenada de la esq. inf. izq. en pantalla */
#define  RENSUP          478     /* Ordenada de la esq. sup. der. en pantalla */
#define  ALTOSUB         239     /* Altura de la subimagen a ampliar en linea */
#define  ANCHOSUB        270     /* Ancho de la subimagen a ampliar en linea */
#define  ANCHOC          18      /*  Ancho y alto de cada cuadrito de flechas */
#define  COLB            564     /*  Abscisa base del cuadrito de flechas   */
#define  COLC            582     /*  COLB + ANCHOC                          */
#define  COLD            545     /*  COLDER + 10                            */
#define  RENA            116     /*  Ordenada base de las opciones          */
#define  RENB            420     /*  Ordenada base del cuadrito de flechas  */
#define  RENC            75      /*  Ordenada del cuadro inferior           */
#define  ALTOHISTO       415     /*  Altura del histograma desplegado       */
#define  COLIZQHISTO     14
#define  COLDERHISTO     527
#define  RENIZQHISTO     27
#define  RENDERHISTO     447
#define  COLORMARCOS     254     /*  Color de los marcos desplegados        */
#define  COLORLETRAS     253     /*  Color de las letras en el despliegue   */
#define  COLOROPCION     5       /*  Color del cuadrito en la posición relativa */
#define  COLORFONDO      88      /*  Color del fondo de la pantalla de desp. */
#define  DELTA           100     /* Incremento de la ventana en enrollamiento */
#define  DELTASUB        10      /*  Incremento en la ventana de ampliación */
```

/* DEFINICION DE CONTANTES DE LAS VENTANAS */

```
#define  MAX_BLOQUE_VENT0x1000*     4K para almacenar ventanas             */
#define  MAX_TOPE        5       /*  Máximo no. de elementos en el stack de vent.
#define  NUM_VENTANAS    23      /*  Número de ventanas                     */
#define  ESQSUPIZQ       201
#define  ESQSUPDER       187
```

```
#define  ESQINFIZQ      200
#define  ESQINFDER      188
#define  UNIONIZQ       204
#define  UNIONDER       185
#define  UNIONVERT      186
#define  UNIONHOR       205

    /*  DEFINICION DE CONSTANTES UTILIZADAS POR EL AUTOMATA  */

#define  ESTADOS        6
#define  TIPOSCAR       7

    /*  DEFINICION DE ESTADOS DE ERROR Y CONSTANTES AFINES  */

#define  CRITICO        1
#define  NO_CRITICO     0
#define  ANCHO_ERR      50
#define  NUM_ERRS       30

            /*  DEFINICION DE CONSTANTES DEL FILTRO  */

#define  ANCHO_MAX_FIL  11
#define  ALTO_MAX_FIL   11

 /*  DEFINICION DE MACROS UTILIZADOS FRECUENTEMENTE POR SPID  */

#define  MAX(a,b)((a) < (b))?(b) : (a)/*  Elemento máximo entre a y b   */
#define  MIN(a,b)((a) < (b))?(a) : (b)/*  Elemento mínimo entre a y b   */

            /*  DEFINICION DE ESTRUCTURAS Y TIPOS  */

typedef unsigned int PALABRA;
typedef unsigned char BYTE;
typedef BYTE far * APTRLEJOS;
typedef BYTE * APTRCERCA;
typedef char * CADENA;
typedef int IMAGEN;
typedef struct
{                                /*  Información sobre la imagen. 6 bytes   */
  BYTE Emin,Emax;                /*  Elementos mínimo y máximo de una imagen */
  PALABRA Alto,Ancho;            /*  Dimensiones de la imagen en disco      */
}                                /*  DIMEN;                                 */
typedef struct
{
```

```
  BYTE PosCol, PosRen;          /* Posición de la esq. sup. izq. de la ventana   */
  BYTE NumCols, NumRens;        /* Dimensiones de la ventana                     */
  BYTE AtrTexNor;               /* Atributo del texto normal                     */
  BYTE AtrTexOpc;               /* Atributo del texto que marca la opción actual */
  BYTE AtrMarco;                /* Atributo del marco                            */
  BYTE NumOpc;                  /* Opción actual en la ventana                   */
  BYTE Encabezado;              /* Indice de encabezado en el arreglo 'Mensaje'  */
} VENTANA;
#endif
```

Bibliografía

[Aho74] *The design and analysis of computer algorithms.* Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman. *Addison–Wesley*, 1974.

[Boor78] *A Practical Guide to Splines.* Carl de Boor. *Springer–Verlag*, 1978.

[Boot67] *Sequential Machines and Automata Theory.* Taylor L. Booth. *John Wiley and Sons*, 1967.

[Brac65] *The Fourier Transform and its Applications.* Ron Bracewell. *Mc. Graw–Hill*, 1965.

[Cool67] *Applications of the Fast Fourier Transform to Computation of Fourier Integrals, Fourier Series and Convolution Integrals.* Lewis Cooley, J. W., P. A. W. and Welch. *IEEE Trans. on Audio and Electroacoustics*, AV-15,No. 2, 79, 1967.

[Conr87] *Data Dependent Filters for Edge Enhancement of Landsat Images.* Knut Conradsen and Gert Wilson. *Computer, Graphics and Image Processing*, 38, pp. 101-121, 1987.

[Daws87] *Introduction to Image Processing Algorithms.* Benjamin M. Dawson. *BYTE The Small Systems Journal*, Vol. 12, Num. 3, March, 1987.

[Ekst84] *Digital Image Processing Techniques.* Michael P. Ekstrom. *Academic Press, Inc*, 1984.

[Elli82] *Fast Transforms, Algorithms, Analyses, Applications.* Douglas F. Elliot, K. Ramamohan Rao. *Academic Press*, 1982.

[Gonz85] *Splines in Geophysics.* Pedro Gonzalez C. and Román Alvarez. *Geophysics, vol.* 50, *No.* 12, pp. 2831-2848, December 1985.

[Hahn84] *Simple Enhancement Techniques in Digital Image Processing.* Saúl Hahn and Eugenio E. Mendoza. *Computer Graphics and Image Processing*, 26, pp. 233-241, 1984.

[Hara78] *KANDIDATS: An Interactive Image Processing System.* Robert M. Haralick and Gary Minden. *Computer Graphics and Image Processing*, 8 pp. 1-15, 1978.

[Hord82] *Digital Image processing of Remotely Sensed Data.* R. Michael Hord. *Academic*

Press, Inc, 1982.

[Horo78] *Fundamentals of computer algorithms.* . Computer Science Press, 1978.

[Knut86] *The TEXbook.* Donald E. Knuth. *Addison Wesley Publishing Company*, 1986.

[Kron89] *Algorithms. Their complexity and efficiency.* Lydia Kronsjö. *John Wiley & Sons*, 1979.

[Land84] *HIPS: A Uniz-Based Image Processing System.* Michael Landy, Yoav Cohen and George Sperling. *Computer Vision, Graphics and Image Processing*, **25, pp. 101-121**, 1984.

[Lira87] *La percepción remota: Nuestros ojos desde el espacio.* Jorge Lira. *Fondo de cultura económica-CONACyT-SEP*, 1987.

[Math87] *Computer Processing of Remotely-Sensed Images.* Paul M. Mather. *John Wiley & Sons*, 1987.

[Mill86] *Assembly Language Techniques for the IBM-PC.* Allan R. Miller. SIBEX *Inc*, **1986.**

[Offe85] *VLSI Image Processing.* R. J. Offen. *Britsh Library Cataloguing in Publication* **Data**, 1985.

[Pavl82] *Algorithms for Graphics and Image Processing.* Theo Pavlidis. *Computer Science Press*, 1982.

[Rich86] *Remote Sensing Digital Image Analysis, An introduction.* John A. Richards. *Springer-Verlag*, 1986.

[Star85] *8087 Applications and programming.* Richard Startz. *Prentice Hall Press*, 1985.

[Vect86] *HP 82960 Graphics Controller Programmer's Reference Manual.* . *Hewlett* **Packard**, January, 1986.

[Whit87] *Compressing Image Data with Quadtrees.* Ronald G. White. *Dr. Dobb's* **Journal** *of Software Tools*, **Vol. 12, No. 3**, March, 1987.