UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

COMPRESION DE IMAGENES CON TECNICAS FRACTALES

T E S I S

QUE PARA OBTENER EL TITULO DE

INGENIERO EN COMPUTACION

P R E S E N T A :

EDUARDO CARRERA OLVERA

DIRECTOR DE TESIS:

ING. SERGIO AMBRIZ MAGUEY

MEXICO, D.F. 1995

Dedico esta tesis a:

Mis padres Juvencio y Honoria Sabina, por el apoyo incondicional, tanto moral como económico, que me brindaron durante todos mis estudios.

Mi esposa Gabriela y mi hijo Gustavo, por su paciencia y apoyo durante el desarrollo de la misma.

Mi asesor Sergio por su guía y consejos para la realización de este trabajo.

Mis hermanos.

Mi amigo de siempre Enrique.

A la UNAM, de la cual me siento orgulloso por pertenecer a ella.

Tabla de Contenido

Introducción

El objetivo principal de este trabajo es desarrollar un sistema que incorpore las nuevas técnicas fractales para la compresión de imágenes; éstas se encuentran almacenadas en archivos con formatos gráficos comerciales usados desde hace varios años y se convertirán a un nuevo formato propuesto: compresión con códigos IFS (Iterated Function Systems), nacidos de la teoría fractal.

El problema general que se pretende resolver es disminuir los requerimientos de espacio en disco que necesitan las imágenes para su almacenamiento, principalmente las imágenes de 24 bits; el sistema a desarrollar para la compresión de imágenes es genérico y puede utilizarse en cualquier campo que requiera el manejo y almacenamiento de imágenes.

El trabajo se divide en 3 partes; en la primera parte se presentan los fundamentos teóricos para el desarrollo del sistema propuesto. El capítulo 1 habla de la teoría del color y presenta los 3 esquemas utilizados para la representación del color; es necesario este conocimiento para comprender el manejo del color por medio de la computadora y además en la compresión de imágenes de 24 bits se utilizan transformaciones de un espacio de color a otro. El capítulo 2 es una descripción de las técnicas más utilizadas actualmente para la compresión de imágenes, se presenta como marco teórico para conocer las diferentes técnicas de compresión que se utilizan en los formatos gráficos comerciales (algoritmos Huffman, RLE y LZW principalmente) y para su comparación con las técnicas fractales. El capítulo 3 explica la teoría fractal y dentro de ésta se hace énfasis especial en los IFS, dado que son la base para la compresión de imágenes por métodos fractales. El capítulo 4 presenta la explicación de los árboles cuádruples los cuales son una estructura de datos idónea para la representación de imágenes y la compresión de éstas; sirve como conocimiento básico para la implementación de la compresión de imágenes con códigos IFS presentada en el capítulo 7.

En la segunda parte del trabajo se presentan las aplicaciones, resultado de la teoría explicada en la parte 1. En el capítulo 5 se detallan los formatos gráficos comerciales utilizados por el sistema propuesto. El capítulo 6 habla del método desarrollado para la compresión de imágenes por medio de códigos IFS, para esto se basa en el capítulo 1 de la teoría del color, en el capítulo 3 de la teoría fractal y en el capítulo 4 para el manejo de los árboles cuádruples.

Finalmente, en la tercera parte se explica el sistema propuesto el cual engloba todos los fundamentos teóricos y aplicaciones de los mismos, explicados en las partes 1 y 2; se detallan sus requerimientos de software y hardware, la interface con el usuario, la operación del sistema y los resultados obtenidos.

Parte I

Fundamentos Teóricos

Capítulo 1

Teoría del Color

Capítulo 1. Teoría del Color.

El sistema visual detecta, asimila e interpreta la información de su alrededor; esto podría parecer muy sencillo pero en realidad los mecanismos perceptuales son muy complejos; el aparato de detección es el ojo, el cual selectivamente detecta diferentes colores de luz, y los dos ojos situados en diferente posición proveen dos interpretaciones geométricas diferentes, permitiendo de esta manera que el sistema visual interprete la profundidad.

La generación y presentación de imágenes provee una alternativa a la riqueza de estimulaciones presentes en el mundo que nos rodea. La técnicas actuales de presentación de imágenes solamente pueden reproducir una pequeño conjunto de los estímulos disponibles en un ambiente real. Si se intenta una representación realística de las imágenes es necesario relacionar el procesamiento perceptual de la información visual con la tecnología del despliegue de imágenes de manera que dicha representación sea lo más fiel posible.

Este capítulo es una introducción a la teoría del color; se discute la percepción del color y su interrelación con el cálculo del color y su reproducción por medio de la tecnología.

1.1 Conceptos Básicos.

La luz es una forma de energía electromagnética que puede especificarse completamente por medio de su longitud de onda; no toda la radiación electromagnética es visible al ojo humano, de hecho la porción visible de la radiación ocupan una longitud de banda muy estrecha comprendida entre los 380 nm y los 780 nm. Esta radiación, cuando incide en el ojo, produce una variedad de sensaciones; la luz compuesta de muchas longitudes de onda produce una sensación visual muy importante llamada color, las diferentes distribuciones espectrales se perciben, aunque no generalmente, con un diferente color. De esta manera, el color es el aspecto de la energía radiante visible por el cual un observador puede distinguir distintas composiciones espectrales; el color es generalmente caracterizado por medio de nombres tales como blanco, negro, rojo, verde, azul, etc.

Los estímulos de color son generalmente más agradables que los estímulos blanco y negro, consecuentemente las imágenes en color son ampliamente utilizadas en televisión, fotografía e impresión; el color es también muy usado en la graficación por computadora para añadir mejores efectos a las imágenes sintetizadas, y la coloración de imágenes en blanco y negro ha sido utilizada extensamente por artistas y gente trabajando en el reconocimiento de patrones.

1.1.1 Descripción del Color.

El color se describe cualitativamente utilizando los términos de matiz (o tono), saturación (a veces llamado cromaticidad) y valor (también conocido como brillantez o intensidad). Estos términos caracterizan la tendencia hacia una organización natural del color. Un primer ordenamiento natural es agrupar en colores tales como rojo, azul y amarillo; este es un ordenamiento por matiz en el cual el color se asocia con su posición en el espectro luminoso. Dentro de cada matiz los colores se ordenan con base en su intensidad y saturación; la intensidad varía del negro al blanco, y en una determinada intensidad, los colores varían de un color neutral a un color puro; al color puro se le conoce como un color saturado mientras que un color neutral es insaturado. Este esquema de ordenamiento del color fue descrito por Meyer en 1983; el sistema de color de Munsell conceptualiza este esquema de clasificación usando un eje neutral cuya intensidad varía del negro al blanco, con la saturación representada por la distancia entre el punto de color y el eje neutral y el matiz tomado como la posición radial (**figura 1.1**).

Una variedad de representaciones conceptuales del color comunes en la graficación por computadora se basan en los colores primarios (rojo, verde y azul) de los monitores de despliegue de video. Algunas de las presentaciones más comunes (**figura 1.1**) son:

- El cubo de color RGB: Red, Green, Blue (en español Rojo, Verde, Azul).
  Éste representa los colores primarios rojo, verde y azul como ejes ortogonales. Los colores que se pueden desplegar en un monitor se encuentran dentro del cubo definido por las coordenadas del punto negro (0,0,0) al punto blanco (1,1,1) y el eje neutral es una línea del punto negro al punto blanco.

- El cono hexagonal HSV: Hue, Saturation, Value (en español Matiz, Saturación, Intensidad).
  El cono hexagonal HSV usa un eje neutral del negro al blanco, el punto blanco se encuentra en un hexágono cuyos vértices representan los colores de las caras del cubo de color RGB.

- El doble cono hexagonal HSL: Hue, Saturation, Lightness (en español Matiz, Saturación, Obscuridad).
  El doble cono hexagonal HSL es una representación similar al cono hexagonal HSV con la excepción de que los colores más intensos se representan con un valor de 0.5 en vez de tener una intensidad igual al blanco.

- **El doble cono HSL.**
  El doble cono HSL es muy similar al modelo del doble cono hexagonal excepto que la sección cruzada es circular.

- **El cilindro HSL.**
  El cilindro HSL expande la base y la parte superior del cono doble en círculos que representan al negro y al blanco respectivamente.

Estas representaciones del color utilizadas en la graficación por computadora están estrechamente relacionadas con los dispositivos de reproducción del color o con la selección artística del color. El cubo de color es un sistema de coordenadas natural porque los tres componentes del color se mapean en un sistema de coordenadas ortogonales utilizando técnicas normales de la geometría en tercera dimensión; las otras representaciones tienen el objetivo de facilitar el empleo de los colores debido a que éstos son fácilmente seleccionados, primeramente al escoger un matiz de una sección del eje neutral y después al tomar la saturación y la intensidad de una parte de dicha sección. Sin embargo estas representaciones son poco utilizadas en la síntesis de imágenes realísticas.

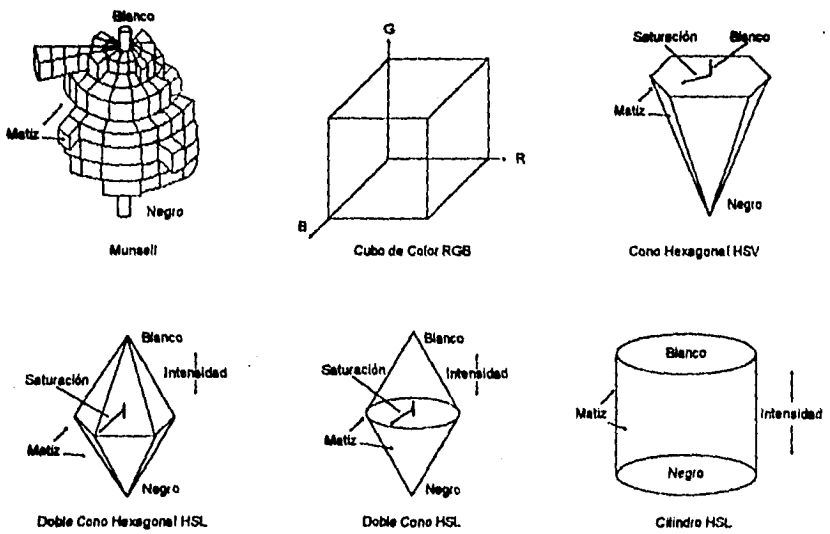


Figura 1.1 Representaciones del Color.

1.1.2 Colorimetría.

La colorimetría es una ciencia perceptual, es decir, estudia e intenta cuantificar la manera en que el sistema visual percibe el color. El estudio de la percepción del color ha dado como resultado un sistema de estándares derivados empírica y estadísticamente que se puede aplicar a la graficación por computadora con resultados predecibles y aceptables. El proceso de obtención de tales estándares se describe a continuación.

El sistema visual humano responde a una porción muy limitada del espectro electromagnético puesto que la luz visible al ojo humano varía de 380 nm a 770 nm, los colores son el resultado de la mezcla de las diferentes longitudes de onda de luz que alcanza a nuestros ojos. Los estudios que se han hecho del ojo humano han demostrado que existen tres tipos de receptores del color llamados conos; excitando selectivamente los tipos de conos es posible producir cualquier sensación de color.

Hacer coincidir un color de prueba con la combinación de las intensidades de tres controles de luz es muy útil si las intensidades requeridas se pueden predecir dada la curva espectral del color de prueba; dicha curva espectral es una gráfica de la intensidad del color en función de la longitud de onda. La sensación que produce en el ojo el color de prueba es una suma de las sensaciones individuales producidas por el color de prueba en cada una de las longitudes de onda del rango visible de luz.; si la misma suma de la sensación visual se produce por medio de alguna combinación de las intensidades de los controles de luz, entonces existe la coincidencia entre el color de prueba y el color generado con los controles. Se debe notar que esta coincidencia es la que se está buscando, no la coincidencia de las curvas espectrales, las cuales no necesariamente tienen que ser iguales en el color de prueba y en el color que coincide.

Suponiendo que se conoce la intensidad requerida de cada control de luz para generar cualquier longitud de onda, la sensación de color producida por la mezcla de las longitudes de onda se puede reproducir sumando las intensidades requeridas de los controles de luz para generar cada una de las longitudes de onda individuales. Las curvas coincidentes para el conjunto de luces de control son gráficas en las cuales el eje Y es la intensidad requerida de los controles de luz y el eje X es la longitud de onda. Predecir las intensidades de los tres controles de luz requeridas para coincidir con el color de prueba es simplemente multiplicar la curva espectral del color de prueba con las curvas coincidentes e integrar las tres curvas resultantes, tal como se muestra en la **figura 1.2**.

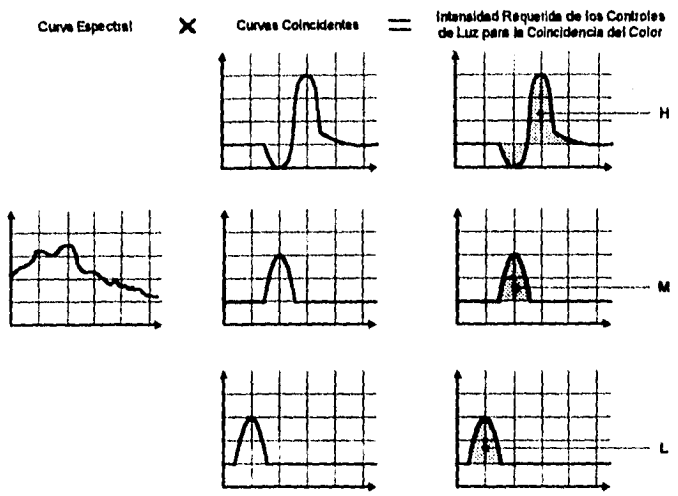


Figura 1.2 Esquema para Determinar las Intensidades de los Controles de Luces dadas la Curva Espectral del Color de Prueba y las Funciones Coincidentes.

Las curvas coincidentes se determinan experimentalmente escogiendo tres controles de luz y la única condición es que un control no debe ser alguna combinación de los otros dos. Se consideran controles de luz con longitudes de onda de 445 nm, 535 nm y 630 nm (escogidos arbitrariamente para demostrar el principio del experimento); se escogen líneas espectrales puras para generar las curvas coincidentes; el observador puede hacer coincidir el matiz de las líneas espectrales pero no la saturación. Se puede hacer coincidir el matiz de las luces de prueba con una combinación de dos controles de luz pero el observador necesita utilizar el tercer control de luz para hacer coincidir la saturación. El experimento de la coincidencia del color se encuentra configurado con luces de control tanto en el lado de control como en el lado del color de prueba, tal como se muestra en la **figura 1.3**; los resultados del experimento se presentan en la **figura 1.4**.

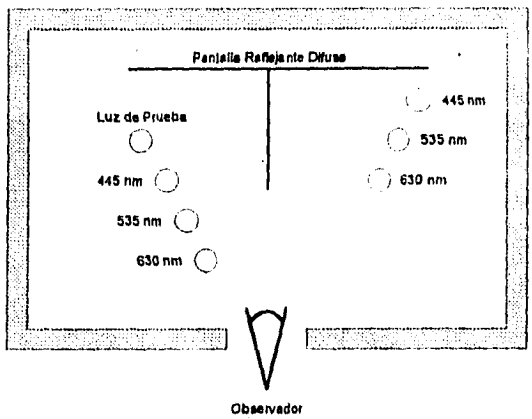


Figura 1.3 Coincidencia de Color con tres Fuentes de Control.

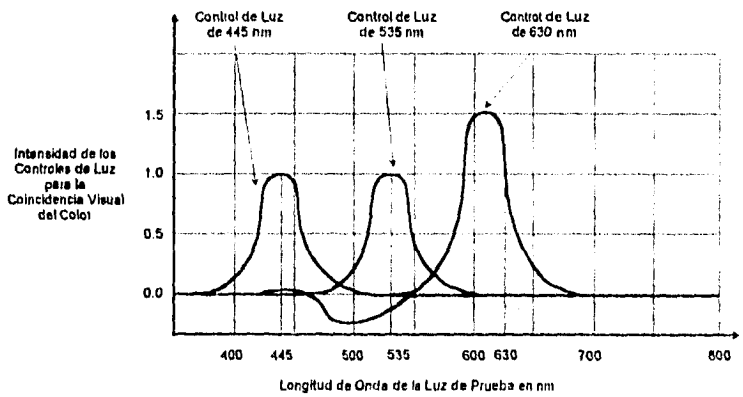


Figura 1.4 Gráficas de las Curvas Coincidentes para Fuentes de Control de 445 nm, 535 nm y 630 nm.

Diagrama de Cromaticidad.

Un diagrama de cromaticidad se crea a partir de una proyección en dos dimensiones de las curvas coincidentes encontradas con el experimento anteriormente descrito, graficadas en tres dimensiones. Cada eje ortogonal representa una de las luces de control, L (longitud de onda corta, azul), M (longitud de onda media, verde) y H (longitud de onda larga, rojo); para cada longitud de onda se dibuja el vector LMH y se extrapola hasta que intersecta el plano L+M+H=1; las coordenadas de esta intersección LMH' se encuentran dividiendo cada L, M y H entre la suma L+M+H, tal como se esquematiza en la **figura 1.5**. La curva definida por los puntos de intersección de los vectores y el plano se proyecta en dos dimensiones en la dirección del eje L **(figura 1.6)**.

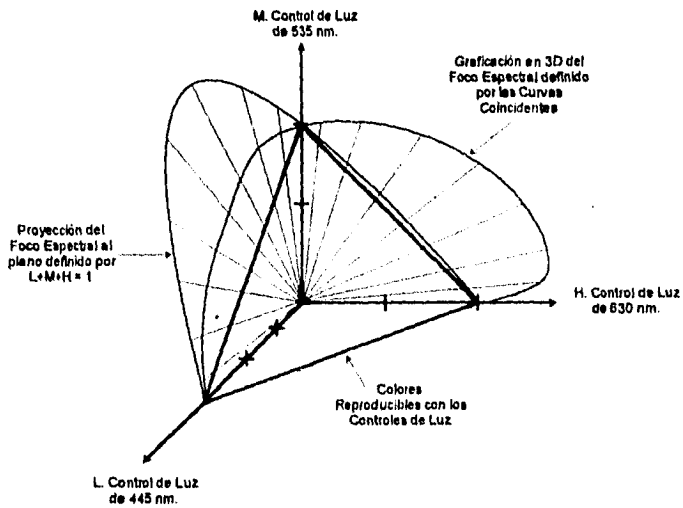


Figura 1.5 Proyección del Diagrama de Cromaticidad.

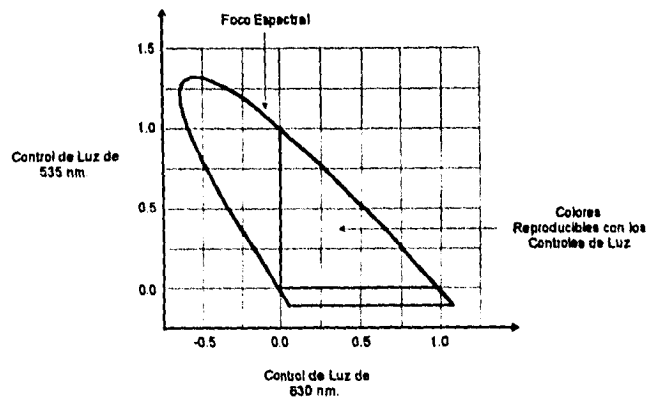


Figura 1.6 Diagrama de Cromaticidad para las Fuentes de Control de 445 nm, 535 nm y 630 nm.

El triángulo creado con las intersecciones de los tres ejes y el plano describe todos los colores que se pueden reproducir con los tres colores primarios. La intensidad del color no se representa en esta proyección debido a que la curva describe los colores espectrales puros. Todos los colores visibles se representan en el interior de la curva; la cromaticidad de un color (describiendo el matiz y la saturación) es su posición en la gráfica.

El resultado de estos experimentos se puede aplicar a cualquier conjunto de colores primarios utilizados para la reproducción de colores. Considérese un conjunto de colores primarios de un monitor RGB con curvas espectrales conocidas, utilizando la técnica previamente descrita, las intensidades de LMH requeridas para cada pixel se determinan con las ecuaciones:

$$R = aL + bM + cH$$
$$G = dL + eM + fH$$
$$B = gL + hM + iH$$

las cuales se representan en forma matricial como:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} L \\ M \\ H \end{bmatrix} = \begin{bmatrix} T \end{bmatrix} \begin{bmatrix} L \\ M \\ H \end{bmatrix}$$

El significado de la anterior representación es que un conjunto de colores primarios se puede expresar en términos de otro, entonces la transformación [T] para encontrar el segundo a partir del primero y $[T]^{-1}$ para encontrar el primero a partir del segundo se definen fácilmente. Específicamente, si las cromaticidades del monitor se conocen a partir de un estándar, entonces las transformaciones hacia y del estándar se pueden determinar; además, si se conocen las cromaticidades de un segundo monitor las transformaciones entre uno y otro también se pueden determinar con facilidad.

La CIE (Comission International d'Eclairage) establece un conjunto primario hipotético XYZ definiendo todos los colores visibles en el octante positivo, las funciones coincidentes iguales, y la función Y coincidiendo la función de sensitividad de la intensidad (función luminosa eficiente). La **figura 1.7** y la **figura 1.8** representan las curvas coincidentes y el diagrama de cromaticidad de la CIEXYZ; la implicación del área igual de las funciones coincidentes es que la curva espectral plana (igual a la intensidad de todas las longitudes de onda) se representa con valores iguales de XYZ.

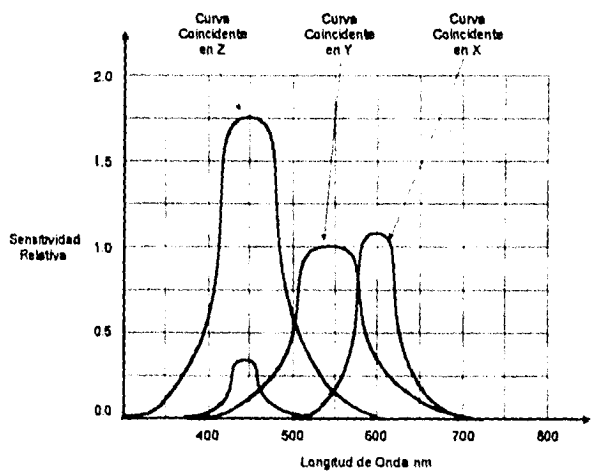


Figura 1.7 Curvas Coincidentes CIEXYZ.

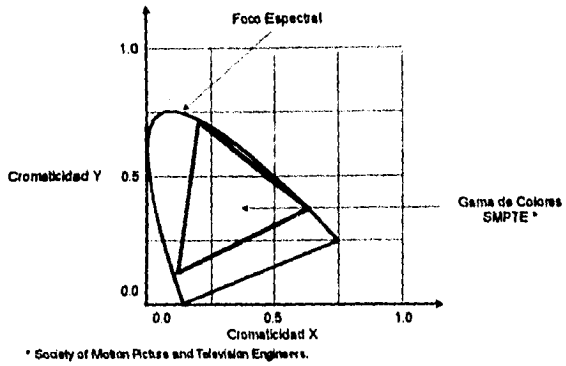


Figura 1.8 Diagrama de Cromaticidad CIEXYZ.

1.1.3 Colorimetría y el monitor RGB.

El despliegue adecuado del color en un monitor requiere en primer lugar conocer la curva espectral o la cromaticidad CIEXYZ del color que se desea desplegar, y en segundo lugar, la transformación de CIEXYZ al conjunto primario RGB. La curva espectral del color se muestrea a CIEXYZ multiplicando la curva por las funciones coincidentes CIEXYZ e integrando las curvas resultantes, después de lo cual el color CIEXYZ se transforma en valores RGB para su despliegue. Alternativamente las funciones coincidentes se transforman al espacio de colores del monitor; estas curvas se utilizan para muestrear la curva espectral directamente a los valores RGB para desplegarlos. La matriz CIEXYZ a RGB se genera utilizando los datos de cromaticidad de los monitores de fósforo, los cuales usualmente los proporciona el fabricante, si no, se pueden calcular utilizando un medidor de cromaticidad de la luz incidente. La matriz de transformación se encuentra a partir del fósforo del monitor y el punto blanco expresando las relaciones de cromaticidad en forma matricial de la siguiente manera:

Fósforo rojo:	r_x	r_y	$r_z = 1 - r_x - r_y$
Fósforo verde:	g_x	g_y	$g_z = 1 - g_x - g_y$
Fósforo azul:	b_x	b_y	$b_z = 1 - b_x - b_y$
Punto blanco:	w_x	w_y	$w_z = 1 - w_x - w_y$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} s_r(r_x) & s_g(g_x) & s_b(b_x) \\ s_r(r_y) & s_g(g_y) & s_b(b_y) \\ s_r(r_z) & s_g(g_z) & s_b(b_z) \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

En esta ecuación, s_r, s_g y s_b son factores de escala para el rojo, verde y azul que definen el punto blanco correcto para las transformaciones. Se seleccionan los valores RGB y XYZ correspondientes al punto blanco y se resuelve para s_r, s_g y s_b. Típicamente las transformaciones se normalizan de tal manera que Y sea 1 para el punto blanco, por lo tanto el punto blanco RGB es (1,1,1) y el punto blanco XYZ es (w_x/w_y, 1, w_z/w_y).

Las transformaciones entre las matrices RGB y CIEXYZ se requieren también para generar la transformación entre los monitores RGB con diferente cromaticidades primarias y/o diferentes puntos blancos. La transformación de los datos de la imagen de RGB para el primer monitor a un segundo monitor R'G'B' se expresa como:

$$\begin{bmatrix} R'G'B' \\ a \\ XYZ \end{bmatrix} \cdot^{-1} \begin{bmatrix} RGB \\ a \\ XYZ \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix}$$

1.1.4 Representaciones Alternas del Color

La especificación del color en el espacio CIEXYZ es ampliamente utilizada para la reproducción del color pero no es muy útil para evaluar cambios relativos en el color. Frecuentemente no es posible representar un color exactamente; se necesita de un medio para evaluar que tan cercana es la coincidencia de un color alternativo para seleccionar el más correcto; el espacio de color CIEXYZ no es perceptualmente lineal, es decir, no es posible predecir la cercanía perceptual de los colores utilizando su posición relativa en el espacio CIEXYZ.

Existen diversos espacios de colores que intentan representar los colores de una manera perceptualmente lineal; estos espacios se utilizan para la comparación de colores. Cada uno de dichos espacios es un sistema triaxial en los cuales la distancia perceptual entre colores es proporcional a la distancia geométrica entre los colores. Los dos espacios de colores más utilizados son el L*a*b*, el cual se basa en una aproximación de tercer orden del sistema de Munsell, y el L*u*v*, el cual evolucionó del estándar CIE U*V*W* de 1964. El CIE 1964(U*V*W*) intentó crear un espacio uniforme del color.

La transformación de XYZ a L*a*b* está dada por:

$$L^* = 25\left[\frac{100Y}{Y_o}\right]^{1/3} - 16 \qquad \text{Para } 1 \le Y \le 100$$

$$a^* = 500\left[\left[\frac{X}{X_o}\right]^{1/3} - \left[\frac{Y}{Y_o}\right]^{1/3}\right]$$

$$b^* = 200\left[\left[\frac{Y}{Y_o}\right]^{1/3} - \left[\frac{Z}{Z_o}\right]^{1/3}\right]$$

La transformación de XYZ a L*u*v* está dada por:

$$L^* = 25\left[\frac{100Y}{Y_o}\right]^{1/3} - 16 \qquad \text{Para } 1 \le Y \le 100$$

$$u^* = 13L^*(u' - u_o')$$
$$v^* = 13L^*(v' - v_o')$$

Donde

$$u' = \frac{4X}{X+15Y+3Z} \qquad v' = \frac{9Y}{X+15Y+3Z}$$

$$u_o' = \frac{4X_o}{X_o+15Y_o+3Z_o} \qquad v_o' = \frac{9Y_o}{X_o+15Y_o+3Z_o}$$

$X_oY_oZ_o$ definen el color del blanco nominal o el punto blanco del monitor. La **figura 1.9** muestra el cubo RGB transformado a los espacios de colores L*a*b* y L*u*v*.

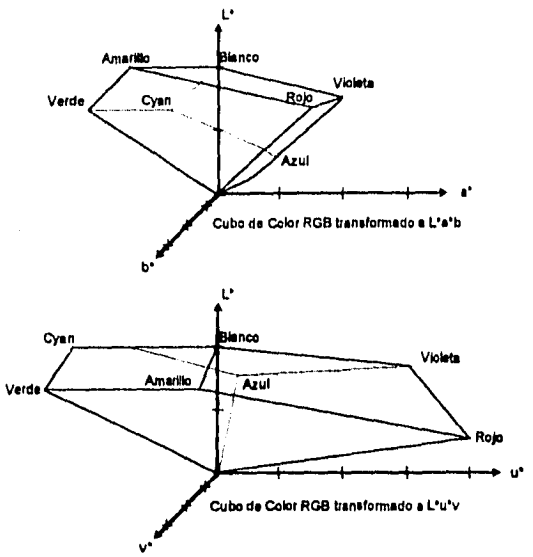


Figura 1.9 Cubo RGB de Color mapeado a $L^*a^*b^*$ y $L^*u^*v^*$.

1.1.5 Espacios de Color para el Cálculo del Color.

La práctica común de calcular colores utilizando valores RGB conduce a graves errores cuando se utilizan modelos complejos de iluminación. La exactitud de los modelos de iluminación y equipos de color calibrados no producirán imágenes realistas a menos que se proporcione la información exacta del material y fuentes de luz y el método de computación no distorsione estos datos.

Los modelos de iluminación se expresan generalmente sin las referencias específicas de la longitud de onda de la luz utilizada. Se asume implícitamente que el cálculo se repite para cada longitud de onda; la interpretación general es la repetición del cálculo del color utilizando los colores rojo, verde y azul para los materiales y las luces.

Los errores potenciales que se pueden presentar al calcular colores con valores RGB son los siguientes:

- La información se pierde cuando se muestrean las curvas espectrales y se representan con sólo tres valores.

- Las funciones RGB de muestreo no se seleccionan con base en la teoría del muestreo sino que se determinan de acuerdo a un criterio perceptual.
- Las funciones RGB de muestreo son específicas al conjunto RGB primario utilizado por lo tanto el cálculo en diferentes espacios RGB de colores puede no producir el mismo resultado.

El espacio de color RGB es una selección obvia para el cálculo del color porque frecuentemente los colores se escogen de una paleta desplegable en un monitor RGB; el usuario simplemente selecciona los colores para los materiales y la luz basándose en un juicio estético en vez de medir las curvas espectrales del material y fuentes de luz, como resultado los materiales y la luz son expresados en términos de valores RGB del monitor utilizado durante la selección. Las tendencias actuales hacia el realismo de las imágenes enfatizan la medición de las propiedades de los materiales y de las fuentes de luz; en lo que concierne al cálculo de los valores RGB se tiene el siguiente ejemplo. Considérese la luz reflejada en un espejo perfecto; la curva de reflectancia para el espejo tiene un valor de 1 en cualquier longitud de onda; si esta curva se muestrea utilizando las funciones coincidentes de la CIEXYZ los valores XYZ resultantes son (106.85, 106.86, 106.93), transformando estos valores a un monitor RGB basado en el estándar NTSC los valores RGB resultantes son (119.85, 102.15, 98.01). Claramente se requiere que los valores RGB sean (1, 1, 1) para que el material se comporte como un espejo perfecto.
Se puede incorporar una operación de escalamiento en las funciones coincidentes sin afectar la cromaticidad muestreada; por ejemplo, las curvas CIEXYZ se pueden escalar normalizando el área bajo la curva Y, esto resulta en un valor XYZ de (1, 1, 1) y en un valor RGB (1.12, 0.96, 0.92), nuevamente se nota que esto no es el resultado deseado
Algunas soluciones sugeridas a este problema son utilizar el espacio CIEXYZ para el cálculo del color, recorrer el punto blanco del monitor del tal manera que la curva espectral del espejo se muestrea a un valor RGB de (1, 1, 1) y la normalización de las funciones de muestreo RGB para que tengan igual área (lo cual recorre la cromaticidad del valor muestreado). El origen del problema es efectuar cálculos de las longitudes de onda después de que los colores ha sido transladados fuera del dominio de la longitud de onda hacia un dominio perceptual.

1.2 Despliegue de Imágenes.

El proceso de despliegue comienza con la información de la imagen la cual describe el color y la intensidad de todos los puntos en el plano de la imagen y se completa con un observador que percibe la imagen desplegada resultante. Los pasos para pasar de la información de la imagen a la percepción del observador comprenden la transformación de los datos de la imagen tal que pueda desplegarse dentro de las limitaciones impuestas por el dispositivo de despliegue, tradicionalmente parte de las transformaciones ocurren en la etapa de la generación de la imagen antes de escribirla al archivo y el resto de las transformaciones ocurren cuando la imagen del archivo se mapea a un dispositivo específico de despliegue.

1.2.1 Consideraciones en los Archivos de Imágenes.

Existen dos aspectos de la corrección de imágenes para su despliegue: el primero es mapear las imágenes de color al espacio de color del monitor utilizado para el despliegue y el segundo es corregir la intensidad no lineal o factor gama del monitor. Ambos, mapear al espacio de color y corregir el factor gama son esenciales para el despliegue adecuado de una imagen.

Los archivos de imágenes son un paso intermedio entre la computación de la imagen y su despliegue. Algo que se debe notar es la forma en que se almacena la imagen en el archivo; el proceso de corrección puede llevarse a cabo antes de almacenar la imagen o después, durante el proceso de despliegue; tradicionalmente la corrección de la imagen se retarda lo más posible hasta el proceso de despliegue.

La práctica actual en la mayoría de las aplicaciones es ignorar la corrección del color puesto que los colores se seleccionan de los valores de RGB utilizando el mismo monitor que se usó para el almacenamiento de la imagen. Las imágenes se calculan comúnmente utilizando aritmética de punto flotante con un espacio de color RGB normalizado en el cual la imagen se representa con valores entre 0 y 1. Los *frame buffers* utilizados para el despliegue de gráficas en alta calidad utilizan 24 bits de precisión, con 8 bits para el rojo, 8 para el verde y 8 para el azul; adicionalmente se utilizan 8 bits extras para el canal del pixel el cual contiene la información mate o alfa para definir la transparencia de la imagen. Los valores de la imagen se calculan de 0 a 1 y después se mapean de 0 a 255 para el rojo, verde y azul con el objeto de almacenar la imagen; cualquier valor de la imagen arriba de 1 se trunca o el color del pixel se escala antes de salvar la imagen puesto que el cálculo de la imagen en color genera valores de intensidad en una escala lineal. Estos valores de intensidad son típicamente salvados en el archivo de la imagen ya que los valores lineales y las tablas de *lookup* en el *frame buffer* se utilizan para la corrección del factor gama.

La resolución de las imágenes se encuentra generalmente en el rango de 512x486 para el video NTSC (640x486 es también un valor común) a 1024x1280 para la aplicaciones del cine, se espera que en un futuro existan resoluciones más altas. Sin compresión de información una imagen típica de video NTSC ocupa aproximadamente 3/4 MB para 24 bits por pixel (1 MB considerando el canal alfa); si la imagen fuera almacenada utilizando punto flotante el archivo se agrandaría hasta 3 MB. Los requerimientos del almacenamiento de datos generalmente dictan el uso de un archivo de imagen codificada con 24 bits por pixel y afortunadamente los esquemas de compresión de datos pueden producir de 50 a 50% de reducción en un archivo de imagen.

Para presentar una imagen lo mejor posible es necesario efectuar la corrección del color, el truncamiento de colores y la corrección del factor gama antes de la creación del archivo de imagen. Es evidente que la transformación del color, el truncamiento de colores fuera de rango y la corrección del factor gama son medidas importantes para el despliegue de imágenes de alta calidad con la apariencia correcta. El formato de imágenes con 24 bits por pixel salva información del color muy limitada de la imagen; cualquier corrección efectuada en el archivo de la imagen resulta en una degradación adicional de la imagen y debe prevenirse. La metodología ideal para el cálculo de la imagen, almacenamiento y despliegue en contraste con la práctica actual se muestra en la **figura 1.10**.

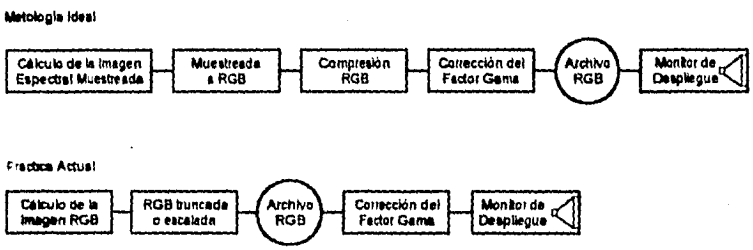


Figura 1.10 Metodología para el Almacenamiento y Despliegue de Imágenes.

1.2.2 Corrección de Color.

La corrección de color para el despliegue consiste en mapear la imagen del espacio de color usado para el cálculo del color al espacio de color utilizado para el despliegue; la corrección de color resuelve las diferencias entre el espacio del color calculado y el espacio de color del despliegue; también se aplica para mapear la imagen a un espacio de color diferente al que fue creada, esta situación se da cuando existen una gran variedad de monitores con diferentes fósforos primarios. Una imagen se debe ver igual no importa cual monitor o *frame buffer* se utiliza para mirarla; en aplicaciones de video la imagen debe ser la misma durante el diseño y después de la grabación.

El primer paso en la corrección del color es determinar a cual espacio de color se debe ajustar la imagen y verificar que la información que define tal espacio de color se encuentra almacenada en el archivo. Los valores RGB son útiles como especificaciones absolutas del color solamente cuando las cromaticidades del monitor y su punto blanco están bien definidos; si solamente hay un monitor en los lugares en que se desea desplegar la imagen, ésta se debe adecuar a dicho monitor.

La imagen debe ser almacenada en el espacio de color que es más crítico para el usuario final de la imagen; una imagen siempre se puede convertir a cualquier espacio de color, sin embargo la imagen se degrada en cada transformación[1]. Los archivos de imagen deben contener el espacio de color como parte de la información de la cabecera del archivo; sin esta información, los colores absolutos correctos para la imagen son desconocidos y no es posible la corrección de color para diferentes monitores.

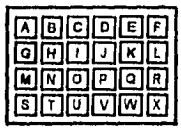
La configuración adecuada del video establece un punto blanco D_{6500}[2]; para aplicaciones de video el conjunto primario de la NTSC es la base para la transformación de la señal de video compuesta YIQ. La transformación en la circuitería descodificadora del monitor debe corregir el conjunto primario RGB_{NTSC} al conjunto primario del dicho monitor. El punto blanco D_{6500} se utiliza en vez de la iluminación recomendada por la NTSC porque el punto blanco del monitor se establece independientemente de la corrección de las cromaticidades del conjunto primario. De esta manera, el espacio de color apropiado para las aplicaciones de video es:

	x	y
Rojo	0.670	0.330
Verde	0.210	0.710
Azul	0.140	0.080
Blanco	0.313	0.329

Las cromaticidades del fósforo las debe proporcionar el fabricante del monitor; en la ausencia de datos de las cromaticidades éstas se pueden medir con un analizador espectral o con un medidor de cromaticidades. El diagrama *Macbeth ColorChecker* proporciona una referencia visual para escalas de grises y de color representativas para la fotografía. Este patrón se utiliza para determinar si la matriz de transformación de CIEXYZ a RGB para el monitor es correcta y para verificar la corrección RGB a RGB de dos monitores diferentes. Este patrón de prueba se presenta en la **figura 1.11**; los colores CIEXYZ se han normalizado para la transformación al espacio RGB del monitor, para esto se ha escalado el eje Y de los colores Macbeth de tal manera que el punto obscuro neutral es negro en el monitor y el punto neutral brillante es blanco.

[1] La degradación de la imagen se debe tanto a errores de cuantificación y redondeos generados por una baja resolución del color de la imagen así como al truncamiento de colores que se encuentran fuera del rango desplegable del monitor.

[2] La designación de D_{6500} se refiere al estándar de iluminación con una cromaticidad de 0.313, 0.329



Los valores de XYZ se normalizan para la transformación de XYZ a RGB utilizando una transformación normalizada con Y=1 para blanco.

Color	X	Y	Z
A. Texture obscura	0.092	0.081	0.058
B. Textura Clara	0.411	0.376	0.303
C. Azul Cielo	0.183	0.186	0.373
D. Follaje	0.094	0.117	0.067
E. Azul Flor	0.269	0.244	0.503
F. Verde Azulado	0.350	0.460	0.531
G. Naranja	0.386	0.311	0.066
H. Azul Púrpura	0.123	0.102	0.359
I. Rojo Moderado	0.284	0.192	0.151
J. Púrpura	0.059	0.040	0.102
K. Verde Amarillo	0.368	0.474	0.127
L. Amarillo Naranja	0.497	0.460	0.094
M. Azul	0.050	0.035	0.183
N. Verde	0.149	0.234	0.106
O. Rojo	0.176	0.102	0.048
P. Amarillo	0.614	0.644	0.112
Q. Violeta	0.300	0.192	0.332
R. Cyan	0.149	0.192	0.421
S. Blanco	0.981	1.000	1.184
T. Neutral 8	0.632	0.644	0.763
U. Neutral 6.5	0.374	0.381	0.451
V. Neutral 5	0.189	0.192	0.227
W. Neutral 3.5	0.067	0.068	0.080
X. Negro	0.000	0.000	0.000

Figura 1.11 Diagrama Macbeth ColorChecker.

1.2.3 Corrección del Factor Gama.

El cálculo del color se basa en los valores lineales de la intensidad; por ejemplo un color **RGB** de (0.5, 0.5, 0.5) tiene la mitad de intensidad que el color (1, 1, 1). Tradicionalmente las imágenes producidas en la computadora se escriben en archivos con bytes denotando la intensidad lineal, con un byte para el rojo, uno para el verde y uno para el azul, de esta manera un pixel con valor de (127, 127, 127) tiene la mitad de intensidad que (255, 255, 255). Desafortunadamente la respuesta típica de los monitores de video a color y el sistema visual no es lineal por lo cual el almacenamiento de las imágenes en formato lineal resulta en una cuantización de las intensidades a una resolución mucho más baja que las disponibles en una resolución de 256 por color. Cuando una imagen lineal se carga en un *frame buffer* también se carga un mapa de color (Tabla de *lookup*) para corregir la no linealidad del monitor. Una curva típica de la luminiscencia del monitor se muestra en la **figura 1.12**, la tabla de *lookup* correspondiente para corregir la no linealidad del monitor se presenta en la **figura 1.13**.

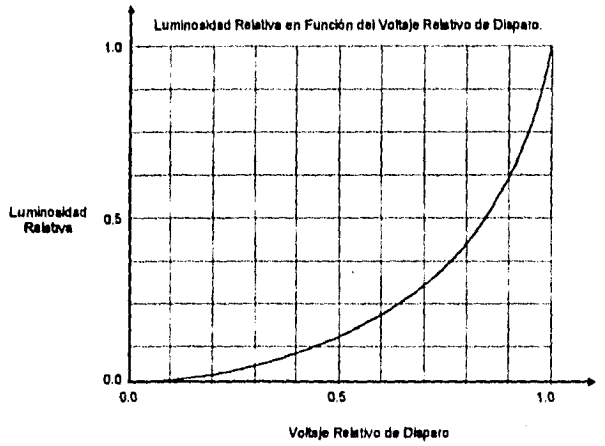


Figura 1.12 Curva de Respuesta de un Monitor Típico.

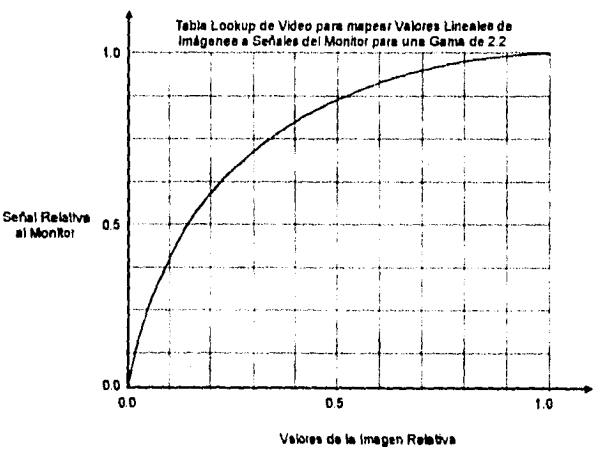


Figura 1.13 Curva de Corrección de la Tabla de Video Lookup típica.

La función de corrección del monitor es una función exponencial de la forma:

Valor lookup = intensidad$^{1.0/\gamma}$

Gama(γ) representa la no linealidad del monitor; los monitores generalmente tienen un valor gama en el rango de 2.0 a 3.0, una gama de 1 representa un dispositivo lineal; la señal estándar gama de la NTSC es 2.2. Utilizar un valor incorrecto de gama resulta en un corrimiento de la cromaticidad e intensidades de la imagen; una gama que es muy grande hace que las intensidades resultantes del mapeo sean más brillantes, una gama muy pequeña hace que las intensidades sean más débiles; las intensidades blancas y negras siempre se mapean correctamente independientemente del factor gama. Considérese un color de (0, 127, 255), si la gama es muy alta el componente verde será muy intenso haciendo que el color tienda al verde, si la gama es muy baja el componente verde es muy débil y el color tiende al azul; los colores rojo y azul no se afectan porque se representan con valores normalizados de cero y uno.

La gama para el monitor se establece desplegando un pequeño cuadro con diferentes valores de colores y midiendo su intensidad, después se establece una curva exponencial que ajuste los puntos medidos y por medio del método de los mínimos cuadrados se obtiene el valor de gama; es necesario normalizar todas las intensidades medidas antes de ajustar la curva. Un problema con esta técnica son los errores de medición con las intensidades bajas a menos que se cuente con aparatos muy sensibles; se requiere realizar medidas con prueba-error para determinar el valor de la intensidad baja requerida para la normalización antes del ajuste de la curva. Además de establecer la gama correcta del monitor, la decisión de corregir el factor gama antes del almacenamiento en un archivo o después en tiempo de despliegue es crítica para la buena apariencia de la imagen. La brillantez lineal percibida es logarítmica con respecto a la intensidad actual y es similar a la curva de respuesta del monitor (figura 1.14), de esta manera la respuesta típica de un monitor presenta características lineales aproximadas en término de la intensidad percibida.

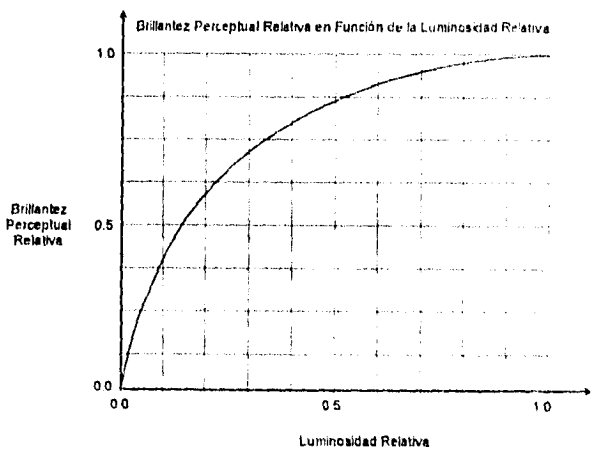


Figura 1.14 Brillantez Percibida en Función de la Intensidad.

Los valores que se cargan en la tabla *lookup* de video para la corrección de un monitor típico con gama de 2.2 se presentan en la **figura 1.15**. El resultado de desplegar la imagen a través de la tabla de video es que el 10% de las intensidades bajas en el archivo de imagen se ajustan dentro del 35% del rango de la resolución de despliegue; otra manera de ver esto es que un incremento de 1 en el archivo de imagen se mapea en un incremento mucho más grande en el despliegue. De esta manera la resolución disponible en el sistema de despliegue en las intensidades más bajas, en las cuales el sistema visual es más sensible, no se está usando; por ejemplo, utilizando la tabla 1.15 el valor 3 de la imagen se mapea a 33 y el valor 4 se mapea a 38 por lo tanto los valores entre 33 y 38 nunca se despliegan; el resultado observable de esto es que las imágenes tienden a presentar bandas en las regiones de baja intensidad debido a la cuantización de intensidades a una resolución menor que la manejada por el dispositivo de despliegue; además, hay que notar que los rangos de intensidades altas mapean diversos valores de la imagen hacia un solo valor de despliegue, de esta manera la información en el archivo de imagen está siendo ignorada.

La función gama se puede aplicar a los valores calculados del color en el modelo de la iluminación antes que de que se conviertan a valores de bytes para su almacenamiento; esto trae consigo que una imagen pueda desplegarse con tablas *lookup* de video lineales tomando ventaja de la resolución total del dispositivo de despliegue.

Valor de la Imagen	Valor de la Tabla	Valor de la Imagen	Valor de la Tabla	Valor de la Imagen	Valor de la Tabla	Valor de la Imagen	Valor de la Tabla
0	0	16	72	.	.	240	248
1	20	17	74	.	.	241	248
2	28	18	76	.	.	242	249
3	33	19	78	.	.	243	249
4	38	20	80	.	.	244	249
5	42	21	81	.	.	245	250
6	46	22	83	.	.	246	250
7	49	23	85	231	243	247	251
8	52	24	87	232	244	248	251
9	55	25	88	233	244	249	252
10	58	26	90	234	245	250	252
11	61	.	.	235	245	251	253
12	63	.	.	236	246	252	253
13	65	.	.	237	246	253	254
14	68	.	.	238	247	254	254
15	70	.	.	239	247	255	255

Figura 1.15 Extracto de la Tabla de Corrección para un Monitor con Gama 2.2.

1.2.4 Truncamiento de Color y Compresión.

Los monitores pueden desplegar solamente un pequeño subconjunto de los colores perceptibles por lo cual los colores de una imagen se deben truncar o comprimir completamente dentro de este subconjunto para su despliegue.

El conjunto de colores desplegables en un monitor está restringido por las cromaticidades del conjunto primario y por las intensidades máxima y mínima que pueden desplegarse; como se presentó en la figura 1.8, los colores desplegables son un subconjunto de los colores visibles. La gama de colores puede representarse como un cubo dibujado en el sistema axial RGB; cualquier color puede dibujarse en este sistema axial y cualquier color fuera del cubo no es desplegable. Los colores que no son desplegables se encuentran en las siguientes dos categorías:

- Colores cuyas cromaticidades están fuera del rango desplegable.
- Colores que tienen cromaticidades desplegables pero que exceden las intensidades desplegables.

En la primera categoría se tienen valores RGB negativos cuando el color se mapea al espacio de color del monitor, en la segunda categoría los valores RGB normalizados son mayores de 1. Cualquier color no desplegable debe ser truncado o comprimido en el espacio de color desplegable; el método de truncar o comprimir debe generar solo una distorsión mínima del color y no debe crear anomalías visibles tales como bandas o corrimiento de colores.

La primera categoría ha sido largamente ignorada en la graficación por computadora; tradicionalmente los colores se calculan como RGB y las curvas espectrales no se utilizan para tal efecto, por lo tanto los colores RGB resultantes no tienen componentes negativos y el resultado del truncamiento o compresión tampoco presenta resultados negativos. Debido a este proceso tradicional no se tenían valores negativos, fue hasta que se comenzaron a usar las curvas espectrales que se presentó dicho problema; una solución es truncar los colores negativos a cero, sin embargo se sugiere que este caso se maneja mejor si se mantiene el matiz del color o la longitud de onda dominante y se resta saturación al color (se añade blanco) hasta que se encuentre dentro del rango desplegable.

Existen varias soluciones para manejar el segundo caso; en una de ellas se escala la imagen hasta que no haya intensidades altas para el despliegue, una solución alterna mantiene la cromaticidad y escala la intensidad del color transgresor; una tercera solución mantiene el matiz y la intensidad dominantes del color y resta saturación al color, y una tercera solución trunca a 1 cualquier componente del color mayor que 1.

Todos estos métodos se practican actualmente, cada método tiene sus propias características de velocidad, facilidad de implememación y apariencia resultante; sin embargo cualquier método de truncamiento de color trae consigo un corrimiento entre el color correcto y el color desplegado; este corrimiento puede ser en matiz, saturación y/o intensidad.

Escalar toda la imagen es análogo a cerrar la apertura del diafragma fotográfico; la razón de contraste de la película es generalmente mucho más pequeña que la del fenómeno que se fotografía y la apertura se ajusta de tal manera que las intensidades del objeto de interés caigan dentro de la región linealmente sensitiva de la película para que se tenga el mejor contraste. Puesto que el rango de contraste del despliegue es muy pequeño comparado con el fenómeno que se modela, la escalación simplemente disminuye la razón de contraste y comprime la información más importante en el rango de intensidades más bajas de la imagen produciendo resultado inaceptables. Considérese un gradiente de color que comienza en algún valor bajo de intensidad de un color desplegable y se extiende a una intensidad alta de la gama desplegable pero con la misma cromaticidad. La **figura 1.16** muestra dicho gradiente mapeado a un cubo de color RGB desplegable, la solución de escalamiento selectivo trunca el gradiente en el punto donde abandona al cubo sin considerar la magnitud hasta este punto; esto se implementa simplemente escalando el valor del color del tal manera que el componente más grande sea reducido a 1, esta solución mantiene el matiz y la saturación pero modifica la cromaticidad; frecuentemente se observan bandas donde comienza el truncamiento de colores debido a que hay una discontinuidad en la primer derivada de la intensidad.

La solución que mantiene la intensidad y matiz pero modifica la saturación produce un ruta de truncamiento que va de la intersección y el cubo de color hasta el punto blanco; las ambigüedades en la implementación de esta técnica de truncamiento es la interpretación de mantener la intensidad o mantener el matiz. Una solución intuitiva alternativa es que el plano perpendicular al eje neutral sea igual al plano de la intensidad; la selección del mejor punto blanco es cuestión perceptual, nuestra percepción del blanco es dependiente del contexto, de esta forma el mejor punto blanco se escoge de acuerdo al ambiente de visión, al punto blanco del monitor y al contenido de la imagen; para facilidad de implementación se puede utilizar el eje neutral del monitor. Este método todavía exhibe una discontinuada en la primer derivada de la intensidad si la ruta de truncamiento alcanza el punto blanco.

El truncamiento puede resultar en un corrimiento de cualquiera de las combinaciones de matiz, saturación e intensidad, dependiendo de la orientación del gradiente; por ejemplo, truncando un valor **RGB** normalizado de (2, 1, 0) a (1, 1, 0) produce un color con corrimiento del naranja al amarillo . La **figura 1.16** describe las rutas de truncamiento para diversos gradientes utilizando diferentes técnicas de truncamiento.

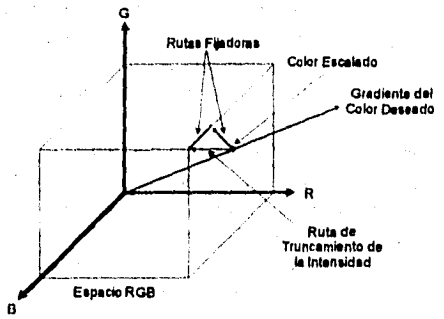


Figura 1.16 Ejemplos de Rutas de Truncamiento para Gradientes de Color.

1.2.6 Dither y Patrones.

La resolución de 8 bits por color primario utilizada comúnmente en el hardware actual de despliegue no es suficiente para eliminar las bandas en aplicaciones críticas del color. El *dither* y el uso de patrones son dos técnicas que pueden usarse para resolver este problema. Tradicionalmente estas técnicas se han aplicado a despliegues con baja resolución en color para aparentar un aumento en dicha resolución.

El uso de patrones sacrifica resolución espacial para incrementar la resolución en color; los pixeles se agrupan en celdas de 2x2 pixeles, 2x3 pixeles, 3x3 pixeles, etc. El color promedio de la imagen sobre el área cubierta por las celdas se genera prendiendo un cierto número de pixeles dentro de la celda. Por ejemplo, considérese un dispositivo de despliegue en blanco y negro de 1 bit, si se despliega la imagen como un conjunto de celdas cubriendo 4 pixeles cada una de ellas, entonces se pueden obtener 5 niveles de intensidad prendiendo o apagando los pixeles 0, 1, 2, 3 ó 4 de la celda (**figura 1.17**). El índice del patrón que se despliega para la celda se determina multiplicando el valor promedio del pixel por el número de intensidades en el patrón; la decisión de cual patrón de la celda se despliega se hace añadiendo los valores deseados del pixel dentro de la celda y multiplicando por un número ligeramente menor que 1.5 para obtener un índice al patrón de la celda. Se puede simular un gran número de niveles de intensidades si se utiliza un tamaño grande de la celda. En el contexto de 8 bits por color primario una celda de 2x2 pixeles permite 1021 niveles de intensidad en vez de los 256 usuales, sin embargo la resolución espacial se ve reducida a 1/2 si se aplican dichas celdas.

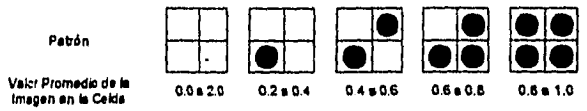


Figura 1.17 Patrones para medios tonos de bits utilizando una celda de 4x4 pixeles.

Aplicar el *dither* en una imagen introduce aleatoriedad en el redondeo tradicional de los colores de la imagen; lo que se intenta es producir valores aparentes intermedios de intensidad con un patrón sin que se presente la reducción espacial que se mencionó previamente.

Existe dos métodos de efectuar el *dither*; el primero acarrea el error en el despliegue de un solo pixel a los pixeles adyacentes; el segundo introduce un error aleatorio en todos los pixeles antes de redondearlos para su despliegue.

La primera técnica fue presentada por Floyd y Steinberg en 1975 y usa un algoritmo que distribuye el error para el despliegue de un pixel dado a los pixeles de la derecha y abajo de ese pixel; este método asume un cálculo secuencial de la imagen comenzado por la esquina superior izquierda de la imagen. Para el pixel actual el error es la diferencia entre el valor calculado y el valor real almacenado en el archivo, este error se distribuye añadiendo 3/8 del error calculado al pixel de la derecha, 3/8 del error al pixel debajo y 1/4 del error al pixel debajo a la derecha.

La segunda técnica añade un número aleatorio a cada valor calculado de la imagen antes de determinar el valor que se almacenará en el archivo. Considérese una gradación normalizada para un despliegue RGB de 24 bits que tiene un valor para uno de los componentes de color que varía de 5.5 en el lado izquierdo de la pantalla a 7.5 en el lado derecho, el despliegue sin *dither* resulta en valores redondeados de 5, 6 y 7. Idealmente la orilla izquierda de la pantalla debe tener un número igual de pixeles con valores de 5 y 6 y la orilla derecha debe tener valores de 7 y 8 para dar en promedio 5.5 y 7.5 respectivamente. Añadiendo un número aleatorio entre 1 y 0 a cada componente del color antes de redondear para el despliegue de color genera la distribución mencionada; añadiendo números completamente aleatorios no produce necesariamente mejores resultados, el problema que se presenta es que los números aleatorios tienen un costo alto para generarse y además la distribución par sobre una pequeña cantidad de muestras no es segura; una distribución non puede causar patrones visibles en la imagen.

Una solución alternativa, conocida como *dither* ordenado, construye una tabla de números pseudo aleatorios que están distribuidos en forma par; esto asegura una distribución par y elimina la necesidad de generar continuamente números aleatorios. Típicamente los números pseudo aleatorios se ordenan en una celda la cual se aplica a la imagen. El patrón de *dither* para una celda de 2x2 es :

1/8	5/8
7/8	3/8

El patrón de *dither* para una celda de 4x4 pixeles es:

1/32	17/32	5/32	21/32
25/32	9/32	29/32	13/32
7/32	23/32	3/32	19/32
31/32	15/32	27/32	11/32

Las celdas de los valores para el *dither* se añaden a los valores calculados de la imagen después de la normalización para la resolución de despliegue.

El algoritmo de Floyd-Steinberg es un proceso serial en el cual cada valor del pixel afecta todos los pixeles que se encuentran debajo a la derecha de dicho pixel, esto trae como consecuencia que para ciertas imágenes se presenten "fantasmas". El algoritmo del *dither* ordenado localiza más el efecto de un pixel sobre sus vecinos pero se presenta una disminución en la resolución espacial. Una solución alternativa es el algoritmo de Knuth, el cual es similar al algoritmo Floyd-Steinberg pero confinado a pequeñas celdas de la imagen; los pixeles en cada celda se ordenan y para cada pixel se determina un valor de salida y el error se distribuye en los pixeles que lo rodean y que todavía no han sido considerados.

Como se dijo previamente, las técnicas de *dither* fueron creadas para incrementar aparentemente el número de colores de una imagen sin sacrificar la resolución espacial. En las imágenes de 24 bits el problema es remover las líneas aparentes causadas por pequeños cambios en el color los cuales se deben a la aplicación de sombras. Debido a la alta resolución de color, añadir *dither* no introduce una degradación notable en la resolución espacial, además las técnicas tradicionales del *dither* ordenado producen resultados aceptables, una celda de 2x2 introduce suficiente aleatoriedad para disminuir las bandas indeseables de una imagen, y en combinación con la corrección del factor gama antes de que se almacene la imagen elimina efectivamente dichas bandas para imágenes de 24 bits; el *dither* se aplica después de que se han calculado los valores para la corrección del factor gama y normalizado a la resolución adecuada para su despliegue.

1.2.6 Video NTSC y RGB.

Frecuentemente existe una gran confusión entre lo que significan las señales de video NTSC y las señales de video RGB. El video RGB se refiere a la separación de las señales para los componentes rojo, verde y azul que se envían al monitor. Los estándares de la señal que se envía al monitor varían de acuerdo al tipo de monitor que se usa para el despliegue. El video NTSC se refiere normalmente al estándar de video compuesto adoptado por la National Television System Committee (NTSC). La señal de video NTSC codifica el rojo, verde, azul y la información del tiempo en una sola señal.

Conectar una señal RGB a un monitor RGB requiere cables separados para las señales rojo, verde y azul; una señal de sincronización indica al monitor cuándo inician los *frames* en la señal entrante; la señal de sincronía se incluye por lo regular en la señal verde (sincronía interna). En trabajos de video de alta calidad se manda una señal de sincronización calibrada en un cable separado (sincronía externa). Conectar una señal NTSC a un monitor NTSC requiere un solo cable; la señal puede convertirse a una señal RGB utilizando un descodificador NTSC; una señal RGB puede convertirse a NTSC utilizando un codificador NTSC.

El proceso de codificación y descodificación NTSC usa una representación intermedia del color llamada YIQ; la transformación del monitor RGB de la NTSC a YIQ está dada por:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.528 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

La transformación de YIQ a RGB_{NTSC} está dada por la inversa de la matriz:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0000 & 0.9557 & 0.6199 \\ 1.0000 & -0.2716 & -0.6469 \\ 1.0000 & -1.1082 & 1.7051 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

La señal YIQ se transforma posteriormente en una señal compuesta de video. La señal Y tiene un ancho de banda de 4.2 MHz, los componentes de color I y Q se modulan a 3.58 MHz defasada 90 grados y el resultado es superpuesto a la señal Y. Un esquema del proceso de codificación y descodificación se presenta en la **figura 1.18**. Se debe notar que en el proceso de descodificación para un monitor de video incluye descodificar a YIQ y después transformar a $RGB_{monitor}$, la transformación final incluye la transformación a RGB_{NTSC} y la transformación al espacio primario del fósforo del monitor.

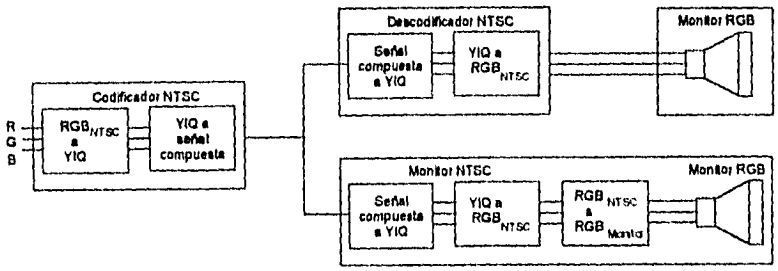


Figura 1.18 Esquema para la Codificación y Descodificación RGT-NTSC-RGB.

1.2.7 Despliegue de Imágenes en Blanco y Negro.

El uso del video en blanco y negro está disminuyendo en todas las áreas del mundo; sin embargo la tecnología en blanco y negro aún domina en muchas otras áreas. Por ejemplo se requiere una imagen desplegada en modo blanco y negro para evaluar las imágenes que pueden ser transmitido en blanco y negro por razones de economía. Este modo se logra simplemente pasando los colores RGB a través del componente Y de la transformación YIQ; el despliegue en blanco y negro usa la señal Y para controlar los cátodos RGB a la misma intensidad. De esta manera el valor $RGB_{monocromático}$ para el despliegue se determina como:

$$RGB_{monocromático} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

La importancia de utilizar vistas de las imágenes en blanco y negro resulta evidente si los colores que son muy diferentes entre sí se hacen indistinguibles cuando se despliegan en blanco y negro; además, el despliegue en blanco y negro impone diferentes condiciones que el despliegue en color cuando se están diseñando imágenes para un impacto visual. Un ejemplo de esto es el uso de chocolate para simular sangre en las películas creadas en el inicio del cine; el impacto de la imagen se puede generar solo si el diseñador puede ver la imagen tanto en color como en blanco y negro.

Capítulo 2

Compresión de Imágenes

Capítulo 2. Compresión de Imágenes.

Durante la historia de la computación, la transferencia de grandes volúmenes de información entre computadoras remotas y la creación de almacenamientos masivos de información y sistemas para consultarla han tenido un crecimiento enorme. Concurrentemente con esto ha habido también una serie de problemas aunada a dicho crecimiento; El principal problema es que el tamaño de las bases de datos utilizadas por las empresas para almacenar su información y los programas de consulta se han hecho cada vez más grandes, requiriendo espacio adicional en disco para los sistemas en línea y cartuchos de cinta para el procesamiento en lotes. Junto al crecimiento de las bases de datos ha habido también un incremento en el número de usuarios y en la duración de las sesiones en nodos remotos. Estos factores han contribuido a exista una cantidad tremenda de información transferida entre computadoras y terminales remotas. Para proveer facilidades para la transmisión de datos las empresas han actualizado continuamente las líneas de comunicación y dispositivos auxiliares tales como modems y multiplexores para permitir alta capacidad en la transferencia de información.

Aunque las soluciones obvias a este problema de almacenamiento y transferencia de información es instalar dispositivos de almacenamiento extra y aumentar las facilidades existentes en las líneas de comunicación, implantar dichas soluciones requiere un incremento adicional en el equipo de la compañía y consecuentemente en el costo. Un método que se puede emplear para disminuir el problema del almacenamiento y transmisión de la información es representar los datos de una manera más eficiente. Si se examina la base de datos de una organización, los archivos de imágenes o los monitores en la línea de transmisión es posible darse cuenta de que existe una gran cantidad de información que puede codificarse más efectivamente.

En este capítulo se presentarán los conceptos generales involucrados en la compresión de información, los tipos de compresión y los métodos actuales para llevarla a cabo. Aunque las técnicas de compresión descritas se aplican a cualquier tipo de compresión, en el desarrollo de las secciones se hará siempre énfasis en la compresión de imágenes, la cual es el objetivo principal de este capítulo.

2.1 Aspectos Básicos de la Compresión.

Para comprender los métodos de compresión de información primero es necesario conocer los conceptos más comunes involucrados en ella. En las secciones siguientes se describen los conceptos básicos involucrados en la compresión de la información.

2.1.1 Compresión de Información.

La compresión de información es el proceso de codificar un conjunto de datos para reducir los requerimientos de almacenamiento. Con la compresión sin pérdida de información los datos comprimidos pueden descomprimirse y ser idénticos a los datos originales, mientras que con la compresión con pérdida de información, los datos descomprimidos son una aproximación aceptable a los originales, de acuerdo a algún criterio de fidelidad. Por ejemplo, con video digitalizado puede ser solamente necesario que el video descomprimido se vea igual de bien que el video original para el ojo humano. Las dos funciones principales de la compresión de datos son:

- Disminuir los requerimientos de almacenamiento.

  La capacidad de los dispositivos de almacenamiento pueden incrementarse efectivamente con software o hardware que compriman los datos antes de que se almacenen y los descompriman cuando se recuperen.

- Aumentar las razones de transmisión.

  El ancho de banda de la línea de comunicación pueden incrementarse comprimiendo la información en el transmisor y descomprimiéndola en el lado del receptor. En estas aplicaciones puede ser crucial que la compresión y descompresión se efectúen en tiempo real.

Los tipos de datos que a los cuales se aplica actualmente la compresión incluye texto de lenguajes naturales, programas fuentes, código objeto, mapas de bits, datos numéricos, gráficas, datos de CAD, mapas, voz, música, datos científicos e instrumentales, facsímiles, imágenes a color o en tonos de grises, imágenes médicas, video, animación y datos espaciales, entre otros.

2.1.2 Compresión de Imágenes.

El objetivo básico de la compresión de imágenes es la conversión de una imagen original continua muestreada o una imagen digital con una razón de bits grande, en una imagen comprimida con códigos binarios que tienen R bits por pixel (bpp) de manera que la reproducción tenga la mejor fidelidad posible. La fidelidad se juzga con un criterio cuantitativo tal como el error promedio cuadrado, con un criterio subjetivo tal como la calidad de la imagen para un observador experimentado, o bien, con pruebas de aceptabilidad estadística para ciertas aplicaciones específicas como las imágenes médicas.

La compresión de imágenes mapea una imagen original *raster*[1] en una cadena de bits adecuada para transmitirla o para almacenarla de manera que el número de bits requeridos para representar la imagen codificada sea menor que el requerido para la imagen original. Si la imagen original está en forma analógica, se requeriría de una precisión infinita para representarla digitalmente pero idealmente lo que se busca es que la imagen codificada requiera los menos bits posibles para minimizar las necesidades de almacenamiento o el tiempo de transmisión. En algunas aplicaciones se puede requerir que la imagen original sea perfectamente recuperable de la imagen codificada; desafortunadamente esto no es siempre posible; por ejemplo, si la imagen es una fotografía analógica no es posible regenerarla a partir de la representación digital, no importa cuantos bits se hayan utilizado para la compresión. Para resolver este problema se podría esperar que, utilizando suficientes bits para la representación digital, se tendría una imagen que es perceptualmente indistinguible de la original, sin embargo, "suficientes bits" pueden ser demasiados para el espacio disponible en disco o el ancho de banda de la línea de comunicaciones. Además, se puede perder información que es importante aunque un ojo entrenado no detecte su ausencia, de esta manera la definición de la calidad depende en gran parte del tipo de aplicación. Es posible utilizar una medida matemática de la calidad para ayudar a la computadora a codificar más eficientemente, y utilizar una prueba subjetiva perceptual más compleja para validar la codificación en una aplicación en particular.

2.1.3 Usos de la Compresión.

Existe una gran variedad de aplicaciones en las cuales se puede utilizar la compresión de imágenes:

- La compresión conserva el espacio de almacenamiento lo cual permite grandes cantidades de imágenes almacenadas en el mismo dispositivo. Por ejemplo, en las aplicaciones médicas significa que los estudios pueden almacenarse en una base de datos local en línea en vez de almacenarse en una localidad remota fuera de línea.

[1] Una imagen *raster* está formada por líneas, cada una de ellas sigue físicamente a la otra, es decir, la hilera $i+1$ sigue a la hilera i.

- La compresión aumenta la razón de transferencia de datos. Por ejemplo, las imágenes comprimidas pueden enviarse en anchos de banda más pequeña, aumentando el número de canales de video en la línea de comunicaciones.

- Las buenas técnicas de compresión permiten una transmisión progresiva, produciendo una reconstrucción gradual de la imagen conforme los bits van arribando. Esto es especialmente útil en *telebrowsing*[2], en donde la selección de las partes importantes de la imagen en sensores remotos debe efectúarse rápidamente.

- Los sistemas de compresión reducen la complejidad del procesamiento de señales y de imágenes. Por ejemplo, los algoritmos de mejoramiento construidos con los códigos resultantes de la compresión de imágenes son mejores que los originales, subjetivamente hablando, con un menor tiempo de procesamiento. Con la compresión se pueden efectuar también diversos procesamientos de imágenes tales como detección de bordes, conversión a medios tonos, etc.

- Los sistemas de compresión pueden incorporar encriptamiento para la seguridad de los datos.

Aunque el ancho de banda se está incrementando gracias a la fibra óptica y el almacenamiento se ha abaratado gracias a la tecnología óptima y magnética, la compresión todavía es de mucho interés debido a:

- Siempre habrá capacidad relativamente pequeña de las líneas de comunicación para la gente que tiene necesidad de usarlas; por ejemplo, las líneas de comunicación vía satélite no pueden manejar los grandes volúmenes de datos científicos de sensores remotos, los dispositivos de comunicación manual (radios de onda corta, modems) tienen un espectro limitado, al igual que la radio marina en pequeños botes.

- Aún las líneas de fibra óptica se saturan rápidamente con televisión de alta definición (HDTV *High Definition Television*) e imágenes médicas o de cine.

- Para las bases de datos enormes se necesita que las consultas se efectúen en un tiempo razonable y los buenos algoritmos de compresión se pueden utilizar para proveer de estructuras de datos eficientes, rapidez en la comunicación y requerimientos bajos de memoria en línea para realizar esta tarea.

Además de todas estas aplicaciones, la compresión puede ayudar en el procesamiento subsecuentes de las imágenes y señales; una razón para digitalizar imágenes es permitir el procesamiento digital en ellas; algunos ejemplos comunes de procesamiento son la mejora, clasificación y análisis de escenas. Un esquema de compresión de imágenes frecuentemente sirve como base para dicho procesamiento digital.

[2] Sistemas de monitoreo remoto.

2.1.4 Beneficios de la Compresión.

Cuando la compresión de datos se utiliza para reducir los requerimientos de almacenamiento también se reduce el tiempo de ejecución de los programas; esto es debido a que la reducción en el almacenamiento trae consigo menos acceso al disco, mientras que los procesos de codificación y descodificación sólo aumentan instrucciones adicionales del programa que debe ejecutarse; puesto que el tiempo de ejecución de un conjunto de instrucciones es normalmente menos significativo que el tiempo requerido para accesar y transferir información a los dispositivos periféricos el tiempo de ejecución disminuye.

Para la comunicación de datos, la transferencia de información comprimida en un canal incrementa la razón efectiva de transmisión de información aún cuando la razón actual de dicha transferencia permanezca igual. La compresión de datos se puede implementar en la mayoría del hardware existente por medio de software o a través de dispositivos especiales de hardware que incorporen una o más técnicas de compresión.

En la **figura 2.1** se ilustra un diagrama de bloques del proceso de compresión-descompresión, el cual se representa como una caja negra.

Figura 2.1 Diagrama Básico de Compresión de Datos.

2.1.5 Terminología.

Como se presentó en la **figura 2.1** , la cadena original de datos se opera de acuerdo a un algoritmo particular para producir una cadena de datos comprimidos. Esta compresión de la cadena de datos original se llama proceso de codificación y al resultado se le conoce como cadena de datos comprimida o también como cadena de datos codificada. Reinvirtiendo el proceso, la cadena de datos comprimida se descomprime para reproducir la cadena original de datos; puesto que el proceso de descompresión descodifica la cadena comprimida, a la cadena resultante se le llama cadena de datos descodificada.

El grado de reducción obtenido como resultado del proceso de compresión se conoce como *razón de compresión*; esta razón mide la cantidad de datos comprimida en comparación con los datos originales, de tal manera que:

$$\text{Razón de Compresión} = \frac{\text{Longitud de la cadena original de datos}}{\text{Longitud de la cadena comprimida}}$$

De esta ecuación se deduce que entre más grande sea la razón de compresión más eficiente es la técnica de compresión empleada. Otro término utilizado cuando se habla de compresión es la *entropía*, la cual se denota como:

$$\text{Entropía} = \frac{\text{Longitud de la cadena comprimida}}{\text{Longitud de la cadena original de datos}}$$

La entropía es el recíproco de la razón de compresión y siempre debe ser menor que la unidad para que el proceso de compresión sea efectivo. La *fracción de reducción de datos* es la unidad menos la entropía (1-entropía). Por ejemplo, una técnica de compresión que tenga un byte de datos comprimido por cada 3 bytes en la cadena original de datos, tiene una razón de compresión de 3, una entropía de 0.33 y una fracción de reducción de datos de 0.66. En la **figura 2.2** se puede ver otro ejemplo de estos términos.

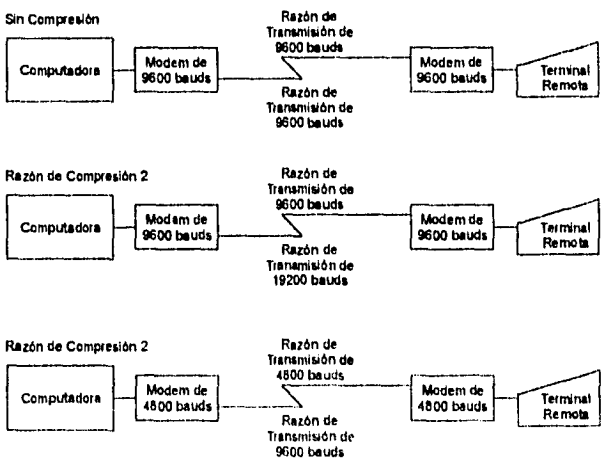


Figura 2.2 La Compresión de Datos afecta la Razón de Transferencia de Información.

2.2 Clasificación de los Esquemas de Compresión.

No es tarea fácil la clasificación de los esquemas de compresión puesto que las técnicas para implementarla son muy variadas, dependiendo de la aplicación. Una primera aproximación que se puede hacer es la división de la compresión en dos grandes tipos:

- **La compresión lógica.**

  Se utiliza principalmente en el diseño de estructuras de datos adecuadas para una cierta aplicación, por ejemplo, el uso de matrices esparcidas para sistemas de ingeniería, la sustitución de campos en las bases de datos, etc.

- **La compresión física.**

  Su objetivo principal es disminuir efectivamente la cantidad de bits requerida para almacenar o transmitir la información, utilizando para esto la correlación de los datos, la eliminación de información superflua, etc.

Este capítulo está enfocado a la descripción de las técnicas de compresión física de datos. En la figura 2.3 se presenta una división de los esquemas de compresión más utilizados, tanto en la transmisión de imágenes digitales como en el almacenamiento de las mismas.

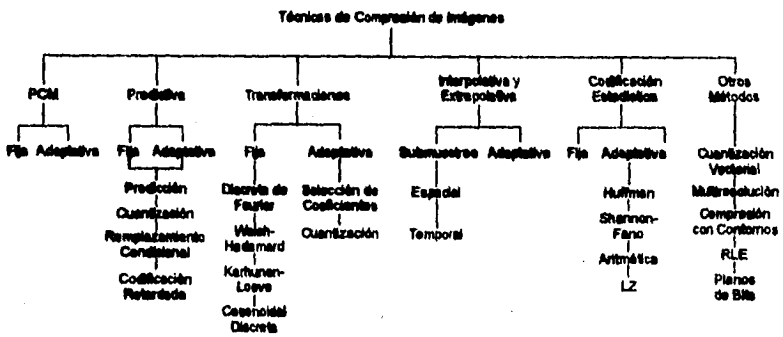


Figura 2.3 Clasificación de los Algoritmos de Compresión de Imágenes.

Como se puede apreciar en el diagrama anterior, existen 5 grandes divisiones en los algoritmos de compresión de información: codificación por modulación de pulsos, codificación predictiva, codificación con transformaciones, codificación extrapolativa e interpolativa, y codificación estadística. Existe otra división más en los cuales los esquemas de compresión no caen dentro de ninguna de las 5 categorías mencionadas pero que se utilizan en la compresión de ciertas imágenes; por ejemplo, los algoritmos RLE (*Run Lenght Encoded* o codificación de la longitud de corrida) son populares y eficientes en el almacenamiento o transmisión de facsímiles en blanco y negro. Adicionalmente, cada una de estas clases de compresiones se puede subdividir basándose en el hecho de que los parámetros sean fijos o cambien de acuerdo al tipo de datos que se están codificando (adaptativos).

2.2.1 Compresión con Pérdida y sin Pérdida de Información.

Otra división más que se aplica sobre los algoritmos de compresión es el hecho de la imagen original pueda ser o no recuperada perfectamente a partir de la imagen codificada; en este caso se está hablando de que existen técnicas con pérdida de información y técnicas sin pérdida de información. De la clasificación de los algoritmos de compresión presentada anteriormente, las siguientes categorías caen dentro de las técnicas con pérdida de información: codificación por modulación de pulsos, codificación predictiva, codificación con transformaciones, codificación extrapolativa e interpolativa. La categoría que se encuentra dentro de las técnicas sin pérdida de información es la codificación estadística. De los otros métodos de compresión, todos son con pérdida de información, excepto los algoritmos RLE.

En la práctica, un codificador puede usar una mezcla de estos algoritmos en una manera compatible para lograr una compresión que sea aceptable tanto en desempeño como en costo. Por ejemplo, es común utilizar PCM para la conversión analógica/digital, después utilizar códigos de Huffman para su transmisión, y finalmente almacenar la imagen con un algoritmo LZ.

En las secciones 2.3 y 2.4 se explica las técnicas más importantes, con pérdida de información y sin pérdida de información, respectivamente y a continuación se explica más detalladamente lo que es la compresión lógica de datos, lo que es la compresión física y las diferencias entre la compresión adaptativa y no adaptativa.

2.2.2 Compresión Lógica.

Cuando se diseña una base de datos uno de los primeros pasos es reducir la información lo más posible. Esta reducción es resultado de la eliminación de campos redundantes y la representación de los otros campos con indicadores lógicos tan pequeños como sea posible. Aunque la compresión lógica depende de los datos y los métodos que se emplean pueden variar dependiendo del análisis, los dos siguientes ejemplos ilustran la facilidad de implementación y los beneficios de esta técnica.

El primer ejemplo de la compresión lógica es el campo de la ocupación en una base de datos de personal. Supóngase que se apartan 30 caracteres para este campo, si el campo es fijo entonces una descripción tal como "barrendero" desperdiciaría 20 caracteres; por lo tanto, para un millón de trabajadores se necesitarían 30 millones de caracteres desperdiciando 20 millones de ellos. Supóngase ahora que se tienen 32, 768 ocupaciones distintas, se podría codificar la ocupación con un entero entre 0 y 32,767, el cual se puede representar en 2 bytes (2^{15}-1= 32, 767), de esta manera con un millón de trabajadores se ocuparían sólo 2 millones de bytes en vez de los 30 millones utilizados para la descripción de la ocupación y para ésta última se tendría un catálogo de máximo 32,767 descripciones con un campo para el índice y un campo para la descripción; el espacio requerido para el catálogo es 32,767 entradas* (2 bytes del índice * 30 bytes de la descripción) = 1,048,544, y el espacio total es:

1,048,544 del catálogo + 2,000,000 ocupación = 3,048,544

De esta manera la reducción en el requerimiento de espacio para este campo sería de 90% aproximadamente.

El segundo ejemplo de compresión lógica es la codificación de un campo de fecha; este tipo de campo se encuentra frecuentemente en las bases de datos. Normalmente se utiliza una representación numérica para una fecha de tal manera que 240768 representa al 24 de julio de 1968; aunque esta representación es ya una reducción utiliza 6 caracteres y puede ser comprimida aún más utilizando notación binaria. Puesto que el día del mes nunca excede a 31 son suficiente 5 bits para representarlo, similarmente se pueden usar 4 bits para representar al mes, mientras que 7 bits pueden representar 127 años permitiendo un rango de años de 1900 a 2027. De esta manera, la fecha se representa con 2 bytes.

Como se discutió anteriormente, pueden considerarse muchos métodos de compresión lógica durante el diseño de una base de datos y cada método tiene un distinto grado de reducción en el almacenamiento; consecuentemente, cuando se transmiten bases de datos comprimidas lógicamente, o parte de ellas, se requiere menos tiempo de transmisión puesto que se transmiten menos caracteres.

2.2.3 Compresión Física.

La compresión física es el proceso de reducir la cantidad de datos antes de su transmisión o de su almacenamiento y expanderlos en el canal receptor o al extraerlos del medio de almacenamiento. Aunque ambos tipos de compresión, lógica y física, traen consigo una reducción en el tiempo de transmisión y en el espacio requerido para el almacenamiento de la información, existen diferencias fundamentales entre ellas, por un lado la compresión lógica se utiliza normalmente para representar más eficientemente a las bases de datos y no considera la frecuencia de las ocurrencias de caracteres o grupos de caracteres, mientras que por otro, la compresión física toma ventaja del hecho de que, cuando los datos se codifican como una entidad distinta y separada, las probabilidades de cada ocurrencia de caracteres o grupos de caracteres es diferente; los caracteres que ocurren más frecuentemente se representan con códigos cortos mientras que los que ocurren raras veces utilizan códigos largos. Al igual que la compresión lógica existen diversas técnicas de compresión física (mostradas en el esquema de la **figura 2.3**), algunas técnicas remplazan las cadenas de caracteres repetidos con un caracter especial indicador de compresión y un carácter de conteo, otras técnicas reemplazan los caracteres que ocurren frecuentemente con un código binario pequeño y los caracteres que ocurren raras veces con un código binario largo. La mayoría de estas técnicas se discuten en este capítulo; la compresión lógica no es objeto de esta tesis, simplemente se ha presentado como una introducción general para definir los tipos de compresión existentes.

2.2.4 Compresión Adaptativa y Compresión no Adaptativa.

Las técnicas con una pasada son conocidas como estáticas y las que utilizan 2 pasadas son llamadas semi-adaptativas. Estas dos técnicas no son adecuadas para la compresión general de datos; las técnicas estáticas no pueden adaptarse a los datos inesperados, y las técnicas semi-adaptativas requieren 2 pasadas sobre el mensaje lo cual no es recomendable para las líneas de comunicaciones. Por ejemplo, las técnicas de compresión **RLE** y métodos de diccionario en general utilizan estadísticas fijas o efectúan 2 pasadas en el mensaje, la primera pasada es para determinar las estadísticas y la segunda para codificar el mensaje (utilizando las estadísticas previamente determinadas).

Las técnicas adaptativas combinan lo mejor de las técnicas estáticas y semi-adaptativas efectuando una sola pasada sobre el mensaje y adaptándose de acuerdo a éste. En cada paso la siguiente parte del mensaje se almacena utilizando un código construido del historial de la compresión; esto es posible debido a que tanto el codificador como el descodificador tienen acceso a dicha historia y pueden construir independientemente el código usado para almacenar la siguiente pieza del mensaje.

Un ejemplo de técnica adaptativa es una técnica en la cual cada instancia de datos se transmite utilizando un código de Huffman construido a partir del historial de transmisión. Una implementación sencilla es prohibitiva e ineficiente porque requeriría que se construyera un nuevo árbol de Huffman de la historia de cada instancia transmitida. El truco es diseñar una estructura de datos que se pueda actualizar incrementalmente con bajo costo; tal técnica ha sido ya implementada con los códigos de Huffman modificados (sección 2.4.1).

La ventaja de adaptación durante una sola pasada podría ser considerada razón suficiente para no considerar a las técnicas estáticas y semi-adaptativas. Una técnica estática siempre logrará mejor compresión que una técnica adaptativa en aquellos datos para los cuales se ha mejorada la técnica estática; esto se debe a que la técnica adaptativa debe gastar tiempo en reconocer lo que la técnica estática ya conoce de antemano, sin embargo, si la técnica estática se alimenta con datos no adecuados producirá resultados totalmente ineficientes. Por otro lado, se ha demostrado que una técnica adaptativa siempre es mejor que una técnica semiadaptativa, o en el peor de los casos es igual.

2.3 Técnicas con Pérdida de Información.

En las técnicas con pérdida de información la intensidad original de los pixeles no se puede recuperar perfectamente como es en el caso de la cuantización sencilla o la conversión analógica-digital; el objetivo de estas técnicas es minimizar la distorsión promedio para una razón de bits dada. Los rangos de compresión varían de 4:1 a 32:1 para *frames* de imágenes utilizando las técnicas comunes de compresión, y se han logrado razones de compresión mucho mejores en las técnicas más recientemente investigadas. Aunque estas técnicas se utilizan primordialmente para conversiones analógicas/digitales de imágenes y transmisión de las mismas, muchos de sus principios y conceptos han servido de base para el desarrollo de otras técnicas de compresión aplicadas al almacenamiento de imágenes.

2.3.1 Cuantización Escalar (PCM).

En la codificación por modulación de pulsos (PCM Pulse Code Modulation), también conocida como conversión analógica/digital, se genera una representación discreta en amplitud y tiempo de los pixeles pero sin eliminar la redundancia estadística o perceptual de la señal. El tiempo discreto se genera muestreando la señal a la razón de Nyquist, mientras que la amplitud discreta se crea utilizando un número suficientemente grande de niveles de cuantización de manera que la degradación debida a los errores de cuantización sea tolerable. La codificación PCM se ha utilizado en la digitalización de video para su transmisión o almacenamiento, o como técnica básica antes de aplicar técnicas más sofisticadas de codificación.

Como se muestra en la **figura 2.4**, el PCM básico consta del muestreo (usualmente a la razón de Nyquist) de una señal y cuantizando cada muestra utilizando 2^k niveles. Aunque no se muestra explícitamente en la figura, se utiliza un prefiltro adecuado antes del muestreo de manera que la razón de muestreo es cercana a la velocidad de Nyquist, y se evitan las distorsiones del efecto escalera (*aliasing*). Cada nivel se representa con una palabra binaria de K bits; usualmente la palabra binaria está relacionada con el nivel de manera que las operaciones aritméticas sean más fáciles. En el decodificador, estas palabras binarias se convierten en una secuencia de las amplitudes discretas de los niveles, los cuales se hacen pasar por un filtro paso-bajas. El PCM básico tiene una simplicidad conceptual que no se encuentra comúnmente en otros codificadores, sin embargo, sufre de ineficiencias puesto que no utiliza la redundancia presente en las imágenes.

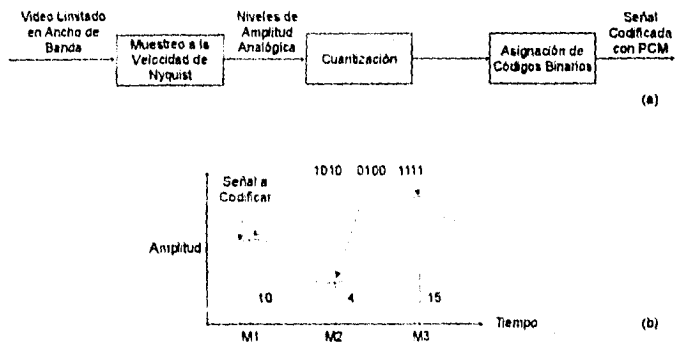


Figura 2.4 **Codificación PCM. (a) Componente de un Codificador PCM. (b) Representación Binaria en 4 bits para Niveles de Amplitud entre 0 y 15.**

La selección de los niveles de cuantización se realiza de manera que se reduzca el error en la cuantización; el ruido de la cuantización puede ser visible en la imagen debido a la separación entre los niveles de cuantización pero es posible reducirlo por varios métodos. Una técnica es utilizar pre y post filtros en el cuantizador mostrado en la **figura 2.4**, haciendo que el cuantizador se comporte como una fuente de ruido aditivo, se calculan los filtros más óptimos de acuerdo al criterio de error promedio cuadrado. Las simulaciones por computadora de estos filtros dan como resultado imágenes esencialmente libres de contornos artificiales; sin embargo, estos filtros reducen también la resolución de las imágenes reproducidas. Una de las técnicas utilizadas para el filtrado es el *dither*, explicado en la sección 1.2.5; con esta técnica se añade ruido pseudoaleatorio a la imagen antes de su cuantización, y después el decodificador substrae el mismo ruido de la imagen cuantizada; estas técnicas de *dither* también se aplican en la conversión de imágenes en tonos de grises a blanco y negro.

2.3.1.1 Codificación PCM para el Color.

Una cámara de color tiene usualmente disponibles los tres colores RGB para cada pixel; sin embargo, la codificación PCM no siempre utiliza estas tres señales. En vez de eso, estas señales se pueden transforman a otro espacio de color antes de su digitalización, por ejemplo, al espacio de color de la NTSC con los componentes YIQ.

Durante el proceso de digitalización de las señales RGB no es necesario cuantizar cada una de ellas con la misma exactitud puesto que el ruido de cuantización no es igualmente visible para cada uno de estos componentes; por ejemplo, los experimentos en imágenes demuestran que el ruido aditivo (el cual es muy similar al ruido de la cuantización) en la señal azul es 10 dB menor que en la señal roja, y 20 dB menor que en la señal verde, de esta manera, si los 3 componentes RGB tienen el mismo ancho de banda, se utilizan menos bits para cuantizar la señales azul y rojas en comparación con la verde. El problema con la codificación de los componentes RGB es que se requiere una resolución espacial relativamente alta para cada componente. Aunque algunos estudios indican que para la mayoría de las escenas naturales se puede aplicar un menor ancho de banda a las señales roja y azul con respecto a la señal verde, es posible lograr una mayor reducción cuando se utilizan otros componente del color (por ejemplo, luminiscencia y cromaticidad en el espacio de colores YIQ). Sin embargo, todavía es común en la práctica codificar los componente RGB utilizando un ancho de banda completa para cada uno de ellos; tales señales PCM así obtenidas se utilizan fácilmente para procesamientos adicionales tales como filtrado, compresión, almacenamiento, etc.

La señales de luminiscencia y cromaticidad YIQ se pueden utilizar para digitalizar directamente el color. En este caso se ajustan las razones de muestreo para que correspondan a los menores anchos de banda permitidos para las señales I y Q. Se han realizado diversos estudios para determinar la exactitud requerida para la cuantización de las señales I y Q de la NTSC y de las señales U y V de la PAL[3]; en estos estudios se ha intentado relacionar los niveles del cuantizador en el dominio de la cromaticidad a algún otro dominio que tenga más significado perceptual.

[1] *Phase Alternation Line* o Línea de Alternación de Fase; es el estándar europeo de televisión.

2.3.1.2 Mapas de Color.

En muchas aplicaciones, por razones de economía o debido a que las imágenes tienen inherentemente un número limitado de colores (por ejemplo, imágenes de circuitos electrónicos), es necesario mapear el espacio de color a un número más pequeño de colores representativos; tales mapas se llaman tablas de colores. Por ejemplo, si se utilizan 8 bits para cada componente de color antes del mapeo entonces el espacio de color tiene 2^{24} colores distintos[4]; sin embargo, si solo se utiliza un número pequeño de colores, por decir 8, es necesario mapear de los 2^{24} colores a los 8. Esto se hace usualmente dibujando un histograma en el espacio de color tridimensional para una imagen dada y escogiendo los colores más representativos que minimicen los errores; se puede utilizar el espacio de colores de la CIE L*u*v para minimizar los errores debido a que dicho espacio es perceptualmente uniforme. Los mapas o tablas de color derivados con este procedimiento dependen obviamente de las imágenes utilizadas para obtener el histograma.

2.3.2 Codificación Predictiva (DPCM).

Los sistemas PCM transmiten amplitudes cuantizadas de cada pixel; sin embargo, existe mucha correlación entre los pixeles que están, espacial y temporalmente, cerca entre sí. La codificación predictiva explota esta correlación. En los sistemas básicos de codificación predictiva se hace una predicción aproximada de la muestra que se va a codificar basándose en las muestras previamente transmitidas; el error o señal diferencial que resulta de la resta de la predicción con el valor actual del pixel se cuantiza en un conjunto de L niveles de amplitud discreta; cada nivel se representa con palabras binarias de longitud fija o variable, las cuales se envían por el canal de transmisión.

En la codificación predictiva, conocida también como codificación con modulación de pulsos diferenciales, se hace una predicción del pixel que se va a codificar; esta predicción se realiza utilizando valores codificados de los pixeles transmitidos previamente, y solamente el error de predicción (señal diferencial) se cuantiza para su transmisión. Esta técnica se puede hacer adaptativa cambiando la predicción de acuerdo con las estadísticas de la imagen, variando los niveles de cuantización con base en criterios perceptuales, o no transmitiendo el error cuando éste se encuentre debajo de un valor límite establecido. Otra posibilidad es retardar la codificación de un pixel hasta que sea posible observar la tendencia futura de la señal y de esta manera utilizar dicha tendencia para la predicción del pixel actual.

[4] Sin embargo, estos colores no son visualmente distintos.

De esta manera, el codificador predictivo tiene 3 componentes básicos: 1) Predictor, 2) Cuantizador, 3) Asignador de códigos. Dependiendo del número de niveles del cuantizador se tiene una división entre la modulación delta (DM Delta Modulation), en la cual el número de niveles de cuantización es $L=2$, y la codificación con modulación de pulsos diferenciales (DPCM Differential Pulse Code Modulation), la cual tiene más de 2 niveles de cuantización $L>2$. Puesto que en la modulación delta se utilizan únicamente dos niveles de cuantización, para obtener una calidad adecuada en la imagen es necesario muestrear a una velocidad mucho más alta que la razón de Nyquist. Aunque la modulación delta se ha utilizado extensivamente en otras formas de onda (por ejemplo, el habla), no ha tenido gran uso en la codificación de imágenes, debido quizá, a la gran velocidad de muestreo requerida; por otro lado, los codificadores DPCM si han tenido mucha utilidad en la codificación de imágenes, razón por la cual, en la siguiente presentación se hablará en detalle de los codificadores DPCM.

En la figura 2.5 se presenta el esquema general de la codificación DPCM; en ella, los $N-1$ pixeles previamente transmitidos se utilizan para calcular la predicción $\hat{b}_N$ del pixel b_N, tanto en el codificador como en el descodificador. La señal diferencial $b_N-\hat{b}_N$ se codifica usualmente con una palabra binaria de longitud variable.

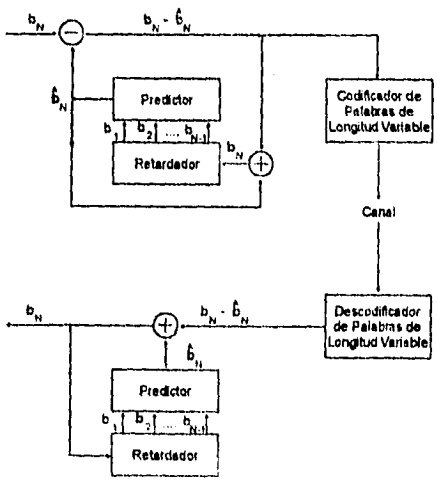


Figura 2.5 Esquema General de la Codificación DPCM.

En cuanto a la codificación DPCM del color y la creación de mapas de colores, se aplican las mismas técnicas descritas en las secciones 2.3.1.1 y 2.3.1.2 para la codificación PCM.

2.3.2.1 Predictores.

Los predictores para la codificación DPCM se pueden clasificar como lineales o no lineales dependiendo si el predictor es una función lineal o no lineal de los pixeles previamente transmitidos. Es posible hacer una división adicional dependiendo de la localización de los pixeles previamente transmitidos: a) Los predictores unidimensionales utilizan pixeles de la misma línea en la que se encuentra el pixel que se va a codificar; b) Los predictores bidimensionales utilizan pixeles en la o las líneas previas; c) Los predictores de *interframe* utilizan además los pixeles de los campos o *frames* previamente transmitidos. Adicionalmente sobre esto, los predictores adaptativos cambian sus características en función de los datos que se están codificando, mientras que los predictores fijos mantienen sus mismas características independientemente de los datos.

2.3.2.2 Cuantizadores.

Los esquemas de codificación DPCM logran la compresión de la imagen al cuantizar el error de predicción con menos niveles que la señal original en sí. Se han estudiado diversos métodos de optimización de los cuantizadores pero su diseño todavía no se ha generalizado y se efectúa *ad hoc*. La mayoría del trabajo que se ha realizado en los procedimientos sistemáticos para optimar la cuantización ha sido en los elementos previos al codificador DPCM, en los cuales se cuantiza la inclinación horizontal aproximada de la señal de entrada

Los cuantizadores se pueden diseñar basándose en aspectos estadísticos de la señal o utilizando ciertos criterios perceptuales, y pueden ser adaptativos o fijos. En los dos apartados siguientes se presentan las características generales de los cuantizadores no adaptativos y cuantizadores adaptativos.

Cuantizadores No Adaptativos.

Se han desarrollado un buen número de cuantizadores no adaptativos con base en las estadísticas de las imágenes a codificar, utilizando el criterio del error promedio cuadrado. Si x con la densidad de probabilidad $p(x)$, es la entrada al cuantizador DPCM, es posible obtener los parámetros del cuantizador para minimizar la siguiente medida del error de cuantización:

$$D = \sum_{k=1}^{L} \int_{t_{k-1}}^{t_k} f(x - l_k)\, p(x)dx$$

donde $t_0 < t_1 < \ldots < t_L$ y $l_1 < l_2 < \ldots < l_L$ son niveles de decisión y representativos, respectivamente, y f(.) es una función de error no negativa. Como se muestra en la **figura 2.6**, se asume que todas las entradas $t_{k-1} < x < t_k$ del cuantizador se representan como l_k. Las condiciones necesarias para optimización del cuantizador con respecto a t_k y l_k para un número fijo de L niveles, están dadas por:

$$f(t_{k-1} - l_{k-1}) = f(t_{k-1} - l_k), \qquad k = 2, 3, \ldots, L$$

y

$$\int_{t_{k-1}}^{t_k} (d/dx)\, f(x - l_k)\, p(x)dx = 0, \qquad k = 2, 3, \ldots, L$$

asumiendo que f(.) es diferenciable. Existen diversos algoritmos para la solución de estas ecuaciones; por ejemplo, la probabilidad de densidad *p(x)* en el caso de los pixeles previos transmitidos se puede aproximar con una densidad laplaciana.

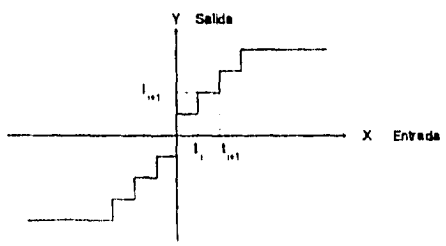


Figura 2.6 **Características de un Cuantizador. *x* es la entrada y *y* la salida. $\{t_i\}$ y $\{l_i\}$ son niveles de decisión y representativos, respectivamente. Las entradas entre t_i y t_{i+1} se representan por l_{i+1}.**

La minimización del error de cuantización, es decir, la distorsión, para un número fijo de niveles de cuantización es relevante para sistemas DPCM que codifican y transmiten la salida del cuantizador utilizando palabras binarias de longitud fija, puesto que este caso la razón de bits de salida depende solamente del logaritmo del número de niveles de cuantización; sin embargo, debido a que la probabilidad de ocurrencia de los diferentes niveles de cuantización es altamente variable, existe una ventaja considerable en el uso de palabras binarias de longitud variable para representar las salida del cuantizador.

Por largo tiempo se ha considerado que para tener una mejor calidad en las imágenes, los cuantizadores deben diseñarse con base en un criterios perceptuales; sin embargo, todavía no existe una uniformidad en cuanto a que criterio utilizar, y menos aún considerando que el sistema visual humano es muy complejo.

Cuantizadores Adaptativos.

Debido a la variación de las estadísticas de la imagen y la fidelidad de reproducción requerida en diferentes regiones de la imagen, los cuantizadores adaptativos DPCM tienen la ventaja de reducir las razones de bits. En general, la imagen se segmenta en varias subimágenes de tal manera que dentro de cada subimagen tanto el ruido de cuantización como las propiedades estadísticas sean relativamente fijas. Sin embargo, esta tarea es extremadamente difícil puesto que la percepción del ruido y las estadísticas pueden no estar suficientemente relacionadas una con otra; existen diversas aproximaciones a esta situación ideal, algunas son puramente estadísticas y otras basadas en criterios psicovisuales. Por ejemplo, es posible trabajar en el dominio de la frecuencia espacial y dividir la señal en dos bandas de frecuencia para explotar la sensibilidad del ojo a las variaciones en el detalle de la imagen; en este caso, la señal de baja frecuencia se muestrea a una razón baja pero se cuantiza más finamente debido a la alta sensibilidad del ojo al ruido en las bajas frecuencias; el componente de alta frecuencia se muestrea a una razón mayor pero se cuantiza con menos niveles debido a la reducida sensibilidad del ojo al ruido en las altas frecuencias. En la cuantización no adaptativa descrita previamente, el error de predicción se utiliza como una medida del error de cuantización; sin embargo, para la cuantización adaptativa se utilizan medidas más complejas no descritas en esta tesis.

2.3.2.3 Asignación de Códigos.

Se mencionó previamente que la distribución de probabilidad de los niveles de cuantización es altamente no uniforme; esto conduce naturalmente a la representación de éstos utilizando palabras binarias de longitud variable (códigos de Huffman, por ejemplo) en vez de usar longitudes fijas. La razón promedio de bits de tales códigos es usualmente cercana a la entropía de la señal de salida del cuantizador. Un código típico de longitud variable para un codificador DPCM con 16 niveles se muestra en la **figura 2.7**, los niveles más internos ocurren más frecuentemente por lo cual se representan con códigos más cortos.

Nivel No.	Longitud del Código	Código
1	12	100101010101
2	10	1001010100
3	8	10010100
4	6	100100
5	4	1000
6	4	1111
7	3	110
8	2	01
9	2	00
10	3	101
11	4	1110
12	5	10011
13	7	1001011
14	9	100101011
15	11	10010101011
16	12	100101010100

Figura 2.7 **Códigos Típicos de Longitud Variable para una Señal codificada con DPCM, con 16 Niveles de Cuantización.**

Para generar los códigos de Huffman (explicados en la sección 2.4.1) se requiere de un conocimiento fidedigno de la distribución de probabilidad de la salida cuantizada; sin embargo, existen otros procedimientos que trabajan con probabilidades aproximadas y tienen una eficiencia aceptable aún cuando las probabilidades reales difieran de los valores asumidos. En la práctica se ha encontrado que los códigos de Huffman no son muy sensitivos a los cambios en la distribución de probabilidades en diferentes imágenes, sin embargo, los códigos de Huffman basados en las estadísticas promedio de muchas imágenes aún son eficientes comparados con códigos diseñados con base en las estadísticas de una imagen individual. Para la mayoría de las imágenes, la entropía de la salida del cuantizador (y consecuentemente la razón promedio de bits utilizando la codificación de Huffman) es más o menos 1 bit/pixel menos que la razón de bits correspondiente para un código de longitud fija.

Uno de los problemas con el uso de los códigos de longitud variable es que la razón de bits de salida del codificador cambia con el contenido de la imagen. Para transmitir la señal digital con razón de bits variable en un canal de bits constante, la salida del codificador deben almacenarse temporalmente en un buffer primeras-entradas primeras-salidas que pueda aceptar una entrada a una razón no uniforme pero cuya salida sea a una razón constante. Puesto que en un sistema práctico se debe utilizar un buffer de tamaño finito, cualquier diseño de un codificador DPCM debe tomar en cuenta la posibilidad del desbordamiento o subdesbordamiento del buffer, y esto depende complejamente de varios factores: el tamaño del buffer, el código variable utilizado, las estadísticas de la imagen y la razón de bits del canal.

El subdesbordamiento del buffer usualmente no es un problema puesto que podrían insertarse valores PCM para incrementar la salida del codificador cuando se requiera: utilizando un canal cuya razón de bits sea mayor que la entropía de la salida del cuantizador reduce grandemente la posibilidad del desbordamiento del buffer; además, diseñando códigos que minimicen la probabilidad de que la longitud de los códigos exceda cierto número, reduce también el riesgo de desbordamiento. Sin embargo, puesto que el desbordamiento del buffer no se puede evitar con seguridad, se deben tener estrategias para reducir gradualmente la razón de bits de salida del codificador cuando el buffer comience a llenarse

2.3.3 Codificación con Transformaciones de Bloques.

En la codificación con transformaciones de bloques, en vez de codificar la imagen como valores de intensidad discreta de un conjunto de puntos muestreados, se realiza una representación alternativa primero transformando los bloques de pixeles en bloques de datos llamados coeficientes y después cuantizando los coeficientes seleccionados para la transmisión. Se han utilizado diversas transformaciones tales como la sencilla de Hadamard o la compleja de Karhunen-Loeve; la transformada cosenoidal se ha popularizado porque se adapta muy bien a las estadísticas de cualquier imagen. La adaptatividad de los codificadores se puede implementar cambiando la transformación para ajustarse a las estadísticas de una imagen en particular o cambiando el criterio de selección y cuantización de los coeficientes para ajustarse a criterios cualitativos y perceptuales.

Se ha encontrado experimentalmente que la codificación con transformaciones tiene un buen desempeño en la reducción de la razón de bits, y este desempeño proviene de 2 hechos: primero, no es necesario transmitir todos los coeficientes resultado de la transformación para obtener una imagen de buena calidad, y segundo, los coeficientes que son codificados no necesitan representarse con una exactitud total.

En las siguientes secciones se presentan las transformaciones lineales más comunes en la codificación con transformación de bloques. Estas técnicas se aplican a imágenes monocromáticas; sin embargo, para codificar imágenes a color la solución más común es codificar separadamente cada uno de los 3 componentes RGB del color[5], utilizando para esto las técnicas de codificación monocromática que se describirán a continuación.

[5] Al igual que en la codificación a color PCM y DPCM, el primer paso es convertir los componentes originales RGB a componentes de luminiscencia y cromaticidad YIQ de tal manera que estos tengan la mínima correlación posible unos con otros.

2.3.3.1 Transformaciones Discretas Ortonormales.

Con transformaciones lineales, cada bloque de pixeles a transformarse se arregla en un vector columna b de longitud N, al cual se aplica una transformada lineal ortonormal (a veces llamada unitaria).

$$c = Tb$$
$$B = T'c$$

donde T es una matriz de transformación de tamaño NxN, c es el vector columna de los coeficientes de transformación, y el apóstrofe indica conjugada transpuesta. Si la *m-ésima* columna de la matriz T' se denota con t_m entonces:

$$t'_m t_n = \delta_{mn}$$

donde $\delta_{mn} = 1$ si $m = n$, $\delta_{mn} = 0$ en cualquier otro caso. De esto proviene el nombre de ortonormal. Los vectores t_m son vectores bases ortonormales de la transformación unitaria T, y pueden escribirse como:

$$b = \sum_{m=1}^{N} c_m t_m$$

donde c_m es el elemento n-ésimo de c, dado por:

$$c_m = t'_m b$$

El primer vector base t_1 consiste de valores constantes para prácticamente todas las transformaciones de interés, es decir, corresponde a una frecuencia espacial de cero y está dada:

$$t'_1 = \frac{1}{\sqrt{N}} (1, 1, 1, \ldots 1)$$

De esta manera, si b_{max} es el valor más grande posible para un pixel, entonces $\sqrt{N}\, b_{max}$ es valor más grande posible para c_1.

El vector b puede construirse de N pixeles sucesivos en una línea *raster* y en tal caso la codificación con transformaciones explota la correlación unidimensional entre los pixeles de una misma línea. Alternativamente el vector b puede construirse considerando un arreglo bidimensional de LxL pixeles de la imagen de tal manera que se construya un vector de longitud $N=L^2$ en el cual se explote la correlación horizontal y vertical en la imagen. Otra tercera posibilidad es denotar un arreglo de LxL pixeles con la matriz cuadrada $B = [b_{ij}]$ y separar la transformación en 2 pasos; el primer paso es transformar las hileras de B con longitud L para aprovechar la correlación horizontal; después se transforman las columnas de B para explotar la correlación vertical. La operación combinada puede escribirse como:

$$c_{nm} = \sum_{i=1}^{L} \sum_{j=1}^{L} b_{ij} t_{nj} t_{mi}$$

donde las t's son los elementos de la matriz de transformación T, y $C = [c_{mn}]$ es la matriz de tamaño LxL de coeficientes de transformación.

En la **figura 2.8** se muestra el diagrama general de la codificación con transformaciones de bloques.

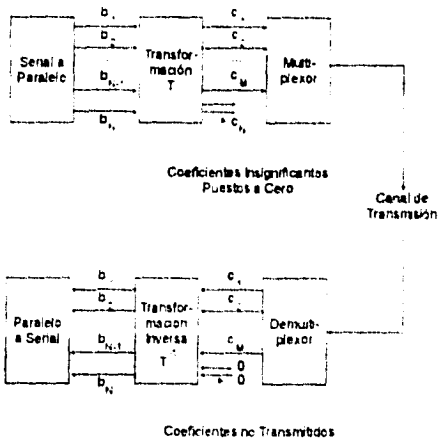


Figura 2.8 **Codificación con Transformaciones. Cada bloque de N pixeles se transforma (usualmente con una matriz lineal ortonormal) en un bloque de coeficientes de transformación y los coeficientes insignificantes se eliminan. Los restantes $M=pN$ coeficientes se codifican y transmiten al receptor, en el cual se efectúa la operación inversa.**

Para la matriz de transformación T se han usado diversos esquemas; los principales son: la transformada discreta de Fourier, la transformada de Walsh-Hadamard, la transformada de Karhunen-Loeve y la transformada cosenoidal discreta. En los siguientes apartados se definen y explican brevemente cada una de estas transformaciones.

2.3.3.2 Transformada Discreta de Fourier.

La matriz unitaria T para la transformada discreta de Fourier (DFT Discrete Fourier Transform) tienen los elementos:

$$t_{mi} = \frac{1}{\sqrt{N}} e^{wr} \left[-\frac{2\pi}{N}\sqrt{-1}(i-1)(m-1) \right] \qquad i, m = 1 \ldots N$$

Para $N = 8$ la matriz T se muestra en la **figura 2.9**; puesto que en este caso el vector b es real, los coeficientes complejos de c tienen una simetria conjugada, es decir:

$$c_{2+p} = c'_{N+p}; \quad 0 \le p < N-2$$

De esta manera, la reconstrucción de los N pixeles de b solamente requiere la transmisión de N valores reales.

Parte Real

1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.000	0.707	0.000	-0.707	-1.000	-0.707	0.000	0.707
1.000	0.000	-1.000	0.000	1.000	0.000	-1.000	0.000
1.000	-0.707	0.000	0.707	-1.000	0.707	0.000	-0.707
1.000	-1.000	1.000	-1.000	1.000	-1.000	1.000	-1.000
1.000	-0.707	0.000	0.707	-1.000	0.707	0.000	-0.707
1.000	0.000	-1.000	0.000	1.000	0.000	-1.000	0.000
1.000	0.707	0.000	-0.707	-1.000	-0.707	0.000	0.707

Parte Imaginaria

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	-0.707	-1.000	-0.707	0.000	0.707	1.000	0.707
0.000	-1.000	0.000	1.000	0.000	-1.000	0.000	1.000
0.000	-0.707	1.000	-0.707	0.000	0.707	-1.000	0.707
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.707	-1.000	0.707	0.000	-0.707	1.000	-0707
0.000	1.000	0.000	-1.000	0.000	1.000	0.000	-1.000
0.000	0.707	1.000	0.707	0.000	-0.707	-1.000	-0.707

Figura 2.9 Matriz para la Transformada Discreta de Fourier con N=8.

Una característica favorable de la DFT es la existencia de un algoritmo bien estudiado llamado *transformada rápida de Fourier* (FFT Fast Fourier Transform). Mientras que una multiplicación directa de la matriz DFT requiere aproximadamente N^2 sumas y multiplicaciones complejas, la FFT requiere cerca de $Nlog_2N$ operaciones si N es potencia de 2. Esta eficiencia se logra utilizando las periodicidades que existen en la matriz T de la DFT; también se han desarrollado algoritmos para la FFT en el caso de que N no sea una potencia de 2. Otras ventajas de la FFT en comparación con la DFT es la reducción de los requerimientos de almacenamiento y errores pequeños de redondeo. Sin embargo, la desventaja de esta transformada es que presenta efectos indeseables debido a las discontinuidades en la extensión periódica del bloque b de N pixeles.

2.3.3.3 Transformada Walsh-Hadamard.

La transformada Walsh-Hadamard (WHT) se describe más fácilmente con recursividad. Sea $H_1 = 1$, y para $N = 2^n$ se define:

$$H_{2N} = \begin{bmatrix} H_N & H_N \\ H_N & -H_N \end{bmatrix}$$

Entonces, la matriz de transformación WHT unitaria simétrica está dada por:

$$T = \frac{1}{\sqrt{N}} H_N = T$$

Para $N = 4$ y $N = 8$ las matrices T se muestran en la **figura 2.10**. La principal ventaja de la WHT es que aparte del factor $(1/\sqrt{N})$, el cálculo solamente requiere sumas y restas, en contraste con la mayoría de las otras transformaciones que requieren además multiplicaciones. Adicionalmente a esto, existe un algoritmo rápido que requiere aproximadamente $N\,Log_2\,N$ operaciones.

$$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$$\frac{1}{\sqrt{8}}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}$$

Figura 2.10 Matrices para la Transformada de Walsh-Hadamard con N = 4 y N = 8

2.3.3.4 Transformada Karhunen-Loeve.

La transformada Karhunen-Loeve (KLT) es una transformada ortonormal que elimina la correlación estadística (no la dependencia estadística) de bloques de pixeles. Es decir, los coeficientes de transformación de la transformada KTL satisfacen (para $m \neq n$):

$$\sum_{c_m c_n} c_m c_n P(c_m, c_n) = \sum_{c_m} c_m P(c_m) \cdot \sum_{c_n} c_n P(c_n)$$

donde la sumatoria se efectúa sobre todos los valores posibles. Esto se escribe usualmente utilizando el operador del promedio estadístico E, como:

$$E(c_m c_n) = Ec_m \cdot Ec_n; \quad m \neq n$$

o bien como:

$$E[(c_m - Ec_m)(c_n - Ec_n)] = \lambda_m \delta_{mn}; \qquad \forall\, m, n$$

donde λ_m es la varianza de c_m. Se debe notar que la independencia estadística implica no correlación pero lo inverso no es generalmente cierto.

La KLT se deriva ajustando primero los vectores de pixeles *b* de tal forma que tengan una media igual a cero, es decir:

Eb = 0, y por lo tanto Ec = 0.

La matriz *NxN* de correlación para los pixeles está dada por:

R = E(bb') = E(T'c c'T) = T'E(cc')T

y tomando en cuenta la condición de ortonormalidad:

RT' = T'E(cc')

E(cc') es una matriz diagonal de *NxN* con varianzas c_m en la diagonal principal; de esta manera, en términos de vectores columna de *T'*, la ecuación anterior se convierte en:

$Rt_m = \lambda_m t_m$; m = 1, ... , N

Además de la ventaja de eliminar la correlación de los coeficientes, la transformada KLT tiene la propiedad de maximizar el número de coeficientes que son lo suficientemente pequeños para no tomarlos en cuenta; estadísticamente, esto significa que la KLT minimiza el error promedio cuadrado.

2.3.3.6 Transformada Cosenoidal Discreta.

Los elementos de la transformada cosenoidal discreta (DCT Discrete Cosine Transform) están dados por:

$$t_{mi} = \sqrt{\frac{2 - \delta_{m-1}}{N}} \cos\left(\frac{\pi}{N}[i - 1/2][m - 1]\right);$$

$i,m = 1...N$

con $\delta_0 = 1$, $\delta_p = 0$, para $p > 0$

De esta manera, los vectores bases t_m de la DCT son senoides con la frecuencia indexada por *m*.

El uso de la DCT se ha incrementado en los últimos años debido a que bajo ciertas circunstancias su desempeño es muy cercano al de la KLT, y además al igual que la transformada discreta de Fourier, existe un algoritmo rápido para el cálculo de la TCD.

2.3.4 Cuantización Vectorial.

Como se vio en la sección 2.3.1, la cuantización escalar involucra básicamente dos operaciones: 1) Particionar el rango de posibles valores de entrada en un conjunto finito de subconjuntos, es decir, establecer niveles de cuantización; y 2) Para cada subconjunto, escoger un valor representativo de salida cuando la entrada se encuentra dentro de ese subconjunto. Con la cuantización vectorial, se efectúan las mismas dos operaciones básicas pero éstas ya no tienen lugar en un espacio unidimensional, sino en un espacio vectorial de dimensión N. De esta manera, si el espacio se particiona en 2^{2R} subconjuntos, cada uno de ellos con su correspondiente valor representativo o *código vectorial*, entonces los bloques b de N de pixeles se pueden codificar con RN bits por bloque o R bits por pixel.

Si no se asumen errores de otro tipo es posible definir una medición de distorsión $d(b - \hat{b})$ entre el vector de entrada b y el vector codificado $\hat{b}$. De esta manera, el problema de la cuantización vectorial es escoger el particionamiento y los códigos vectoriales de tal forma que minimicen la distorsión para el tipo de imágenes manejadas. La medición de distorsión más utilizada, al igual que en otras técnicas, es el error promedio cuadrado, el cual se define como:

$$d = \frac{1}{N}(b - \hat{b})^t (b - \hat{b})$$

$$d = \frac{1}{N}\sum_{i=1}^{N}(b_i, \hat{b}_i)^2$$

y la distorsión es simplemente el valor esperado:

$$D = E(d)$$

El diseño de cuantizadores vectoriales (VQ Vectorial Quantizers) generalmente requiere de imágenes de *entrenamiento* puesto que todavía no están disponibles estadísticas exactas. Los códigos vectoriales VQ y el particionamiento se determinan iterativamente repitiendo el procesamiento del conjunto de imágenes de entrenamiento; una vez que se ha diseñado el VQ, un bloque b de pixeles se puede codificar utilizando la regla del *vecino más cercano*, es decir, se debe escoger el código vectorial $\hat{b}$ que minimice $d(b - \hat{b})$ y se transmite un índice codificado de NR bits para indicar al receptor cual palabra de código debe utilizar; finalmente, el descodificador simplemente despliega el vector $\hat{b}$ como la representación codificada del vector b.

La principal ventaja de la VQ es la estructura sencilla en el receptor, la cual solamente consiste de una tabla de códigos que contiene 2^{NR} códigos vectoriales. La desventaja es la complejidad del codificador y el hecho de que las imágenes diferentes al conjunto de entrenamiento no se representan correctamente en la tabla de códigos.

2.3.4.1 Diseño de la Tabla de Códigos.

Un método propuesto recientemente y que se ha generalizado es el algoritmo de Lloyd, también conocido como algoritmo LGB o algoritmo de los k-promedios, el cual requiere de una tabla inicial de códigos y procede como sigue:

1) Mapea los bloques de entrenamiento en códigos utilizando la regla del vecino más cercano. Si la distorsión total de este mapeo o la cuantización es muy pequeña, terminar; si no, seguir con el paso 2.

2) Para cada código vectorial $\hat{b}$ se determina el subconjunto de bloques de entrenamiento que se mapearon en éste; después, se remplaza $\hat{b}$ con otro código que reduzca (idealmente minimice) la distorsión total para dicho subconjunto de bloques de entrenamiento. Continuar con el paso 1.

En el paso 2, para el error promedio cuadrado mínimo lo mejor es remplazar el código con el promedio del correspondiente subconjunto de bloques de entrenamiento; el algoritmo termina cuando la distorsión total se ha minimizado al máximo posible.Hay que notar que para una tabla inicial de códigos dada, el algoritmo LGB solamente produce una distorsión local mínima; otras tablas iniciales de códigos también pueden converger a la mejor tabla final de códigos. En realidad, escoger una tabla inicial de códigos es otro problema en sí; una selección adecuada es que la tabla inicial sea un subconjunto de vectores de entrenamiento y que cada uno de ellos sea lo menos similar entre sí; otra tabla inicial de códigos podría obtenerse simplemente utilizando un cuantizador con pocos niveles en cada uno de los componentes individuales de los vectores.

Un tercer método para la tabla inicial de códigos (el más utilizado), se llama técnica de particionamiento, la cual comienza con un sólo código igual al promedio de los vectores de entrenamiento, después se genera el siguiente código introduciendo un pequeño factor diferencial y se aplica el algoritmo LGB para optimar estos dos códigos. El proceso continúa de manera similar (**figura 2.11**), es decir, en un nivel K dado, a cada uno de los códigos vectoriales se le agrega el factor diferencial para obtener el conjunto de $2K$ códigos vectoriales, los cuales se optiman con el algoritmo LGB para produce la tabla de código al nivel $2K$; finalmente, se alcanzan 2^{RN} códigos vectoriales. Este árbol también se utiliza en el proceso de codificación; en cada nodo C_k, el vector de entrada se compara con los códigos vectoriales C_{2k-1} y C_{2k} y el más cercano se escoge como el siguiente código.

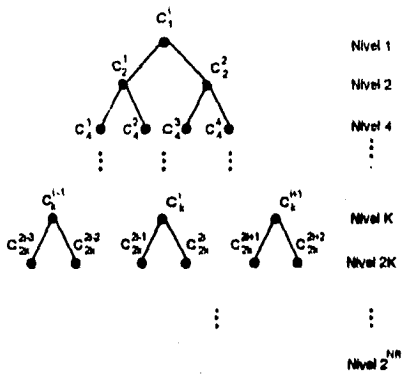


Figura 2.11 El método de particionamiento para generar la tabla de códigos VQ.

Un problema con el algoritmo LGB y sus variantes es su excesivo tiempo de convergencia en la construcción de la tabla de códigos; sin embargo, las técnicas para reducir este tiempo son casi siempre menos efectivas. Existe una técnica que reduce el tiempo de construcción de la tabla con un desempeño aceptable, conocida como *tabla de códigos en árbol*, la cual comienza con un código y procede al igual que la técnica de particionamiento; sin embargo, en cada etapa K el conjunto de vectores de entrenamiento se particiona en subconjuntos de vectores de entrenamiento que se mapean al mismo código vectorial; después, en la siguiente etapa, como cada código se multiplica por el factor diferencial para producir otro código vectorial, para optimar los dos vectores resultantes solamente se utilizan aquellos vectores de entrenamiento que se encuentran en el subconjunto correspondiente.

Otra técnica para la generación de la tabla de códigos se llama algoritmo del *vecino más cercano* o algoritmo NN (NN Nearest Neighbor). Esta técnica aparentemente no presenta degradaciones en su desempeño, es significativamente más rápida que el método de particionamiento, y tiene una complejidad computacional que crece linealmente con el tamaño del conjunto de entrenamiento. La idea básica de este método es comenzar con el conjunto completo de entrenamiento y, como primer paso, unir los dos vectores que se encuentran mas cerca uno de otro para generar otro que sea el promedio de ambos; la aplicación sucesiva de esta operación eventualmente reduce el conjunto de vectores al tamaño deseado. La principal ventaja computacional del algoritmo NN se logra con una operación de pre-proceso en la cual se particiona el conjunto de entrenamiento para generar un árbol *K-d*; esto no solo posibilita que la operación de búsqueda sea más rápida sino que se efectúan uniones múltiples en cada iteración. El algoritmo NN generalmente produce una mejor tabla de códigos que el algoritmo LGB, lo cual muestra la importancia de seleccionar adecuadamente una tabla inicial de códigos; por otro lado, se puede obtener una tabla inicial de códigos más óptima generando una tabla con el algoritmo NN la cual sirva de entrada al algoritmo LGB.

2.3.4.2 Desempeño de los códigos VQ.

La calidad de las imágenes que se obtienen con la codificación VQ depende de muchos factores, incluyendo el tamaño del bloque, la cantidad de correlación entre los pixeles, lo adecuado de la tabla de códigos para las imágenes que se codificarán, etc. La codificación VQ se utiliza bastante para codificar imágenes con una razón pequeña de bits puesto que las razones altas de bits entrañan un grado de complejidad impráctico. Por ejemplo, con bloques de tamaño 4x4 pixeles, la codificación a una razón de 1 bit/pixel implica una tabla de códigos con 2^{16} palabras de códigos; aunque esto no sería mucho problema para el descodificador, la búsqueda en la tabla de códigos durante el proceso de codificación sería prohibitivamente compleja y larga. Continuando con el ejemplo anterior, con imágenes de tamaño 256x256 pixeles en un conjunto de 16 imágenes se tienen solamente 2^{16} bloques, esto implica que se necesita un conjunto mucho más grande de imágenes de entrenamiento, lo cual incrementa la complejidad para generar la tabla de códigos.

La optimización de la calidad subjetiva de la imagen utilizando cuantización vectorial no está definida actualmente; el error promedio cuadrado es virtualmente el único criterio considerado en los trabajos actuales de codificación con cuantización vectorial. Sin embargo, se ha propuesto a la cuantización vectorial como un método ideal para la codificación y reproducción de imágenes con diferentes características de detalles, texturas, bordes, etc.

2.3.5 Otros Métodos de Compresión.

Existen otros métodos de compresión de imágenes con pérdida de información, aparte de los mencionados anteriormente. En las siguientes secciones se explicarán brevemente cada uno de estos métodos, junto con sus ventajas y desventajas.

2.3.5.1 Codificación en Multirresolución, Piramidal y Subbandas.

Codificación en Multirresolución.

En ésta, una imagen se codifica a diferentes intensidades y resoluciones espaciales, usualmente en una forma anidada. Quizás el ejemplo más conocido son los códigos piramidales de Burt y Adelson, los cuales forman una representación piramidal laplaciana de la imagen por medio de muestreo e interpolación, y arreglando los residuos en capas repetidas; este proceso, conocido como codificación piramidal, se explica más a detalle a continuación.

Codificación Piramidal.

En la codificación piramidal una imagen se representa como una serie de imágenes pasa-bandas, cada una muestreada con razones de bits sucesivamente menores. Una implementación de esto es construir en el primer paso una secuencia de imágenes filtradas con filtros paso-bajas, $\{ b_k(x_j, y_i) \}$, $k = 1 \ldots n$, tal que:

$$b_{k+1}(x_j, y_i) = \sum_{m=-p}^{+p} \sum_{n=-p}^{+p} h(m, n)\, b_k(x_{2j+m}, y_{2i+m})$$

donde las funciones contribuyentes $h(m, n)$ se escogen para ser aproximadamente igual a una función Gaussiana, y $b_0(x_j, y_i)$ es la imagen original. Por simplicidad $h(m, n)$ se calcula en forma separada como:

$$h(m, n) = h(m)\, h(n)$$

Para $p = 2$, las funciones contribuyentes que se toman son $h(0) = 0.4$, $h(1) = h(-1) = 0.25$ y $h(2) = h(-2) = 0.05$. En este caso $h(m, n)$ es un producto cartesiano de dos funciones triangulares idénticas y simétricas; la imagen $b_{k+1}(.)$ es una versión filtrada de $b_k(.)$, de esta manera:

$$L_k(x_j, y_i) = b_k(x_j, y_i) - b_{k+1,e}(x_j, y_i)$$

donde los pixeles interpolados de la versión expandida $b_{k+1,e}(.)$ están dados por:

$$b_{k+1,e}(x_j, y_i) = 4 \sum_{m=-2}^{+2} \sum_{n=-2}^{+2} h(m, n)\, b_{k+1}(x_{(j-n)/2}, y_{(i-m)/2})$$

En la ecuación anterior solamente se incluyen en la sumatoria los enteros subscritos de x y y.

De esta manera, se han creado una serie $L_k(.)$ de imágenes filtradas con paso-bajas; si cada una de estas imágenes se colocan apiladas unas sobre otras, el resultado es una estructura de datos piramidal. La compresión de la imagen se efectúa cuantizando $b_n(.)$ seguido de $L_k(.)$ y después codificar la salida cuantizada; de hecho, la imagen $b_n(.)$ puede consistir solamente de 1 pixel. En el receptor, la operación inversa se lleva a cabo fácilmente.

Las simulaciones muestran que el desempeño de 1 bit/pixel se puede lograr con una calidad razonable de la imagen; además de la buena razón de compresión, el esquema conduce por si mismo a una transmisión progresiva; en este caso, el nivel más alto de la pirámide se manda primero y es expandido por interpolación en el receptor para presentar una imagen inicial tosca, esto es seguido por el envío de los otros niveles, los cuales se añaden a la imagen inicial, incrementando de esta manera su calidad. La principal ventaja de este método es que los cálculos son simples, locales y pueden ser efectuados en paralelo; además, los mismos cálculos son iterados para construir la secuencia que constituye a la pirámide.

Codificación con Subbandas.

En el procedimiento descrito en la codificación piramidal, las $L_k(.)$ pueden calcularse como la salida de n filtros pasabandas paralelos, en vez de obtenerse iterativamente; en este caso, el algoritmo de compresión es conocido como codificación con subbandas. Al igual que en la codificación piramidal, las $L_k(.)$ se cuantizan y codifican separadamente, quizás utilizando una codificación RLE para las cadenas con valor de cero; la imagen descodificada se obtiene simplemente añadiendo las imágenes pasabandas recibidas en cada paso de la transmisión.

2.3.6.2 Codificación con Transformaciones Interpolativas no Unitarias.

En esta sección se presenta brevemente una técnica de compresión de imágenes llamada codificación con transformaciones interpolativas no unitarias, la cual se explica con un ejemplo. Supóngase que se particiona la imagen en bloques de 4x4 pixeles, etiquetados como sigue:

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

Para cada bloque se codifica y se transmite el primer pixel A sin modificaciones; después, se codifica y transmite el pixel P utilizando DPCM y el pixel $\hat{A}$[6] como predicción; posteriormente se transmiten los pixeles D y M con DPCM y utilizando $(\hat{A}+\hat{P})/2$ como predicción. Los pixeles restantes se codifican y transmiten con DPCM utilizando la interpolación de los vecinos más cercanos enviados previamente como predicciones. De esta manera, por ejemplo, el pixel B podría codificarse usando como predicción el valor interpolado $(2\hat{A} + \hat{D})/3$, y el pixel C usaría $(\hat{B} + \hat{D})/2$ como predicción; de esta manera, todas las predicciones utilizan solamente los pixeles que previamente se han cuantizado y transmitido, previniendo una posible acumulación del error de cuantización.

[6] El acento circunflejo denota el valor cuantizado

Si se arregla el bloque de vectores en un vector columna

$$b = [A\ B\ C\ D\ E\ F\ G\ H\ I\ J\ K\ L\ M\ N\ O\ P]'$$

entonces los valores diferenciales **DPCM** se pueden calcular con una matriz de transformación de tamaño 16x16. Por ejemplo, los primeros 6 renglones de la matriz de transformación corresponden a la codificación interpolativa **DPCM** de los pixeles *A*, *P*, *D*, *M*, *B* y *C*.

$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
-1/2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1/2 \\
-1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1/2 \\
-2/3 & 1 & 0 & -1/3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1/2 & 1 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\cdot & & & & & & & & & & & & & & & \cdot \\
\cdot & & & & & & & & & & & & & & & \cdot \\
\cdot & & & & & & & & & & & & & & & \cdot
\end{bmatrix}$$

La técnica descrita previamente difiere de la codificación con transformaciones sencillas en que ahora cada coeficiente de transformación se asocia con un pixel en particular del bloque *b*; además, después de que cada coeficiente se cuantiza, el pixel asociado de *b* se remplaza con un valor cuantizado de manera que las predicciones puedan calcularse exactamente en el receptor. Puesto que la matriz de transformación es una matriz esparcida (conteniendo muchos ceros), la complejidad de la transformación es menor que con las transformaciones ortonormales; en la mayoría de los casos la complejidad aumenta linealmente con el tamaño de los bloques.

2.3.5.3 Codificación con Truncamiento de Bloques.

El objetivo de este algoritmo es cuantizar los pixeles de un bloque de *LxL* de tal manera que se preserve el promedio y la varianza; en particular, un cuantizador de 2 niveles es atractivo debido a su simplicidad. El primer paso es calcular el promedio y la varianza del bloque:

$$\hat{b} = (1/L^2) \sum_{i,j=1}^{L} b_{ij}$$

$$\sigma^2 = (1/L^2) \sum_{i,j=1}^{L} (b_{ij} - \hat{b})^2$$

los cuales son cuantizados y transmitidos. Después, los pixeles del bloque se cuantizan a dos niveles utilizando $\hat{b}$ como valor de disparo, es decir, se envía 1 ó 0 dependiendo de que b_{ij} sea mayor o menor que $\hat{b}$, respectivamente.

En el receptor, el primer paso es evaluar q, el número de 1's recibidos para el bloque M; el número de 0's se calcular como $p = L^2 - q$. El siguiente paso es evaluar los dos niveles del cuantizador para preservar el promedio y la varianza; esto se hace fácilmente con :

$$L_0 = \hat{b} - \sigma \sqrt{\frac{q}{p}}$$

$$L_1 = \hat{b} - \sigma \sqrt{\frac{p}{q}}$$

Finalmente, cada pixel se descodifica a L_0 o L_1 dependiendo si se recibió un 0 ó 1, respectivamente.

Utilizando bloques de 4x4 pixeles y 8 bits por cada $\hat{b}$ y σ, la razón de bits para la codificación es 2 bits/pixel; si $\hat{b}$ y σ se codifican juntos con 10 bits, la razón disminuye a 1.625 bits/pixel. Las imágenes descodificadas con esta técnica reproducen bastante bien los bordes y los objetos grandes; además, la textura aleatoria, aunque no se reproduce con gran exactitud, frecuentemente es de buena calidad para la mayoría de las aplicaciones. Sin embargo, en áreas con poco detalle de la imagen donde la brillantez y/o color son pequeños, existen errores de discontinuidad y bordes fantasmas debido a los pocos niveles de cuantización.

2.3.5.4 Codificación Extrapolativa e Interpolativa.

Las técnicas de codificación extrapolativa e interpolativa trabajan con base en un principio diferente a las técnicas PCM y DPCM, pues lo que hacen es enviar un subconjunto de pixeles hacia el receptor, el cual extrapola o interpola para obtener los pixeles no transmitidos. Estas técnicas se han utilizado bastante para sistemas de *interframes* en conjunto con la codificación predictiva; la adaptación en estos sistemas consiste en variar el criterio de selección de las muestras que se envían y cambiar también la estrategia de extrapolación e interpolación de las muestras que aún no se han enviado.

En la codificación interpolativa y extrapolativa se escoge un subconjunto de pixeles para su transmisión, la cual se puede efectuar con cualquiera de las técnicas descritas previamente (PCM, DPCM, transformación de bloques, etc.); los pixeles que no son seleccionados para la transmisión se reproducen en el receptor por medio de interpolación utilizando la información de los pixeles transmitidos. En la codificación interpolativa, algunos pixeles presentes y algunos pixeles futuros se transmiten y el resto se interpolan, mientras que en la codificación extrapolativa, los pixeles futuros inmediatos se extrapolan de los pixeles pasados, uno por uno; el proceso de extrapolación se detiene cuando se encuentra un pixel cuyo error de extrapolación sobrepasa un valor de umbral predefinido, después de lo cual dicho pixel se transmite y los siguientes pixeles son extrapolados otra vez como se hizo anteriormente

Un ejemplo de codificación interpolativa se muestra en la figura 2.12.

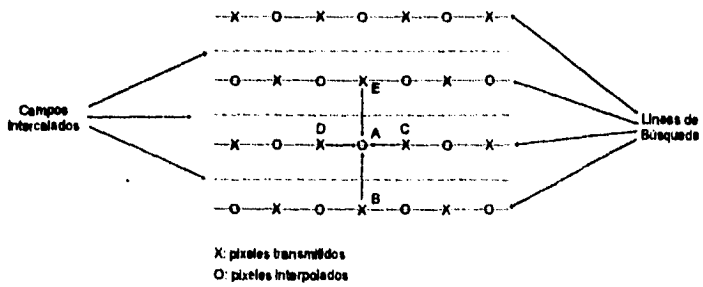


Figura 2.12 **Ejemplo de codificación interpolativa utilizando un muestreo 2:1, escalonado de una línea a la siguiente.**

2.4 Técnicas sin Pérdida de Información.

La compresión sin pérdida de información, conocida también como codificación sin ruido, compactación de datos, codificación con entropía o codificación invertible, es aquella donde la información original puede ser perfectamente recuperada de la representación digital. La compresión sin pérdida de información solamente es aplicable a imágenes ya digitalizadas y requiere técnicas de códigos con longitud variable. La idea básica de esto códigos es utilizar palabras largas de códigos para cadenas raras y palabras cortas para las cadenas más comunes; los códigos se diseñan explícitamente de tal manera que el número de bits para cada pixel sea lo más pequeño posible.

Las técnicas más populares para la compresión sin pérdida de información son los códigos de Huffman, los códigos de Huffman modificados, algoritmos LZ (Ziv-Lempel) y los códigos aritméticos. Las razones típicas de compresión para las técnicas sin pérdida de información varían desde 1.7:1 hasta 4:1.

Está difundido ampliamente que la compresión sin pérdida de información es necesaria en muchas aplicaciones y debido a esto ha habido un gran cantidad de trabajo dedicado al refinamiento de estas técnicas. La necesidad de no perder información es obvia en algunas aplicaciones tales como la compresión de programas de computadora o archivos binarios arbitrarios; sin embargo, en aplicaciones científicas y médicas esto ya no es cierto pues lo que se busca son aproximaciones de buena calidad, razón por la cual se utilizan las técnicas con pérdida de información descritas en la sección 2.2.

2.4.1 Códigos de Huffman.

Los códigos de Huffman es una técnica estadística de compresión de datos cuyo empleo reduce la longitud promedio de los códigos que se utilizan para representar los símbolos de un alfabeto. El alfabeto puede ser el alfabeto de cualquier idioma o un tipo de alfabeto de datos codificados tales como los conjuntos de caracteres ASCII o EBCDIC. Los códigos de Huffman son códigos óptimos debido a que tienen la longitud promedio más pequeña de todas las técnicas de codificación estadística; además, poseen la propiedad del prefijo, lo cual establece que un grupo de códigos pequeños no puede ser duplicado como principio de un grupo de códigos más largo, esto significa que si un carácter se representa con la combinación 100 entonces 10001 no puede ser el código de otra letra puesto que buscando en la cadena de bits de izquierda a derecha el algoritmo podría interpretar los 5 bits como el código 100 seguido de 01.

Los códigos de Huffman se obtienen con una estructura de árbol tal como se muestra en la **figura 2.13**. Primero, los símbolos se arreglan en orden descendente de frecuencia de ocurrencias, los grupos con las frecuencias más pequeñas (X_3 y X_4) se combinan en un nuevo nodo con ambas probabilidades sumadas, después este nodo se une con la siguiente probabilidad más baja de la ocurrencia de otro símbolo o par de símbolos. En la **figura 2.13** el par X_1X_4 se une con X_2 para producir un nodo con probabilidad de 0.4375; finalmente, el nodo que representa las probabilidades de X_2, X_1 y X_4 se une con X_1 dando como resultado un nodo cuya probabilidad es uno. Este nodo maestro representa la probabilidad de ocurrencia de todos los símbolos del conjunto, asignando valores binarios de 0's y 1's a cada segmento que parte de cada nodo es posible deducir el código de Huffman para cada carácter. El código se obtiene trazando una ruta desde el nodo con probabilidad uno a cada carácter escribiendo cada uno de los 0's o 1's encontrados.

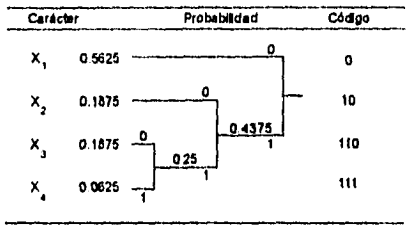


Figura 2.13 Desarrollo de los Códigos de Huffman utilizando una estructura de árbol.

El promedio de bits por símbolo se calcula multiplicando las longitudes del código de Huffman por sus probabilidades de ocurrencia. De esta manera los códigos usan:

$$1*0.5625 + 2*0.1875 + 3*0.1875 + 3*0.0625 = 1.63 \text{ bits/símbolo.}$$

Una propiedad clave de los códigos de Huffman es que son instantáneamente descodificados al momento en que los bits codificados se leen de la cadena de datos comprimidos. Un ejemplo de esta descodificación instantánea se muestra en la **figura 2.14**, aquí, la cadena de datos comprimidos se descodifica inmediatamente leyendo de izquierda a derecha sin esperar a que se presenta el fin de bloque.

Mensaje Codificado	0	10	0	111	10	110	0
Mensaje Descodificado	X_1	X_2	X_1	X_4	X_2	X_3	X_1

Figura 2.14 Propiedad de Descodificación Instantánea.

La sustitución de un número de bits que representa un carácter particular de los datos o grupo de caracteres es un proceso simple cuando el número de sustituciones es limitado. Conforme el número de sustituciones se incrementa, la complejidad del proceso de sustitución también aumenta. En la **figura 2.15** y en la **figura 2.16** se ilustra el desarrollo de los códigos de Huffman para el alfabeto inglés.

La estructura de árbol empleada para desarrollar los códigos mostrados en la **figura 2.15** se obtiene de la siguiente manera:

1. El conjunto de caracteres se arregla en una columna a la izquierda en orden decreciente de frecuencia de ocurrencia, con la frecuencia escrita al lado de cada carácter.

2. Comenzando en la parte inferior de la tabla se dibujan líneas horizontales para cada frecuencia del carácter. Las líneas con las dos frecuencias menores se unen y se suman sus frecuencias para obtener una frecuencia compuesta, se traza una línea horizontal a partir de esta frecuencia compuesta.

3. El proceso de combinar las dos frecuencias menores continúa hasta que se han unido todas las líneas.

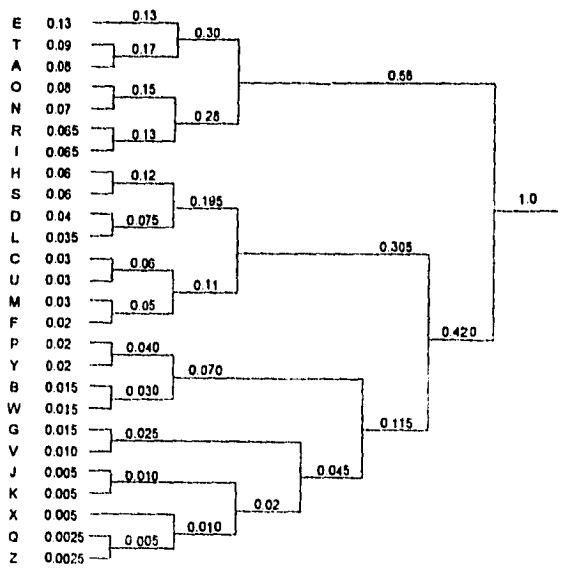


Figura 2.15 Desarrollo de una Estructura de Árbol para el Alfabeto Inglés.

Después de que se ha construido el árbol, el código de Huffman de cada carácter se asigna colocando un bit 0 en uno de los lados del nodo y un bit 1 en el otro[7]. La secuencia apropiada de bits para cada carácter se determina trazando la ruta del nodo maestro al nodo correspondiente al carácter, utilizando los 0's ó 1's que se encuentren en dicha ruta. La asignación de bits a las rutas del árbol y los códigos de Huffman resultantes se presentan en **la figura 2.16.**

El número de bits requerido para codificar una letra con la técnica de Huffman se determina con la fórmula:

$$b = f(-\log_2 P)$$

donde P = probabilidad de ocurrencia de la letra.
f(x) = el entero más cercano más grande o igual que x.

Puesto que la probabilidad de E es 0.13 y $-\log_2 0.13$ es 2.94 entonces el entero más grande o igual a 2.94 es 3, de esta manera se requieren 3 bits para codificar la letra E.

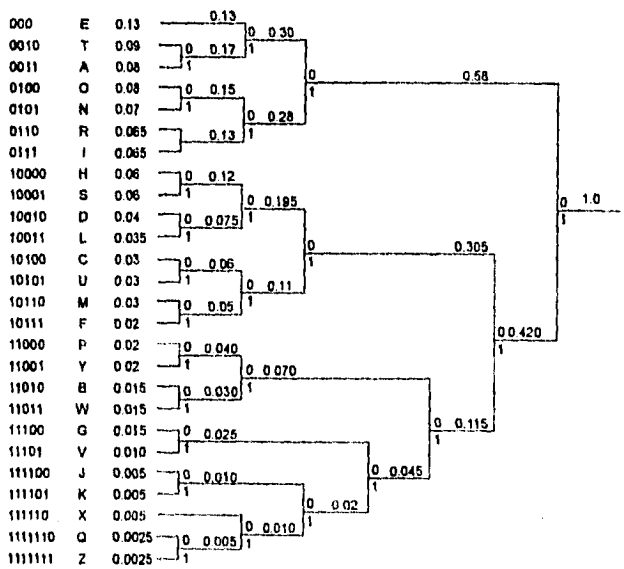


Figura 2.16 Asignación de los Códigos de Huffman.

[7] La asignación de que lado se colocan los 0's ó los 1's es arbitraria pero una vez que se escoge un orden de asignación se debe conservar para todos los nodos del árbol.

2.4.1.1 Distribuciones de Frecuencia.

Para desarrollar los códigos de Huffman con una longitud promedio del código que se aproxime a su entropía se requiere que la distribución de frecuencia de cada carácter o símbolo a codificarse se conozca de antemano. Puesto que la distribución de frecuencia de una cadena de datos es proporcional al uso final de la cadena, es factible utilizar una distribución preseleccionada de frecuencias para desarrollar los códigos de Huffman óptimos para ciertas secuencias de transmisión. Por ejemplo, la distribución de frecuencias en un archivo de un texto en español es diferente de un archivo con un programa en lenguaje C; en el primer caso la distribución de caracteres sigue la distribución normal del idioma español con las vocales presentando más ocurrencias y la letra ñ menos, mientras que en un programa en C los caracteres especiales tales como *, +, -, / tienen un alto grado de ocurrencias que normalmente no se encuentran en un texto de español.

Para compensar las diferencias de distribución de frecuencias se pueden considerar varios esquemas. Un primer método es considerar una codificación adaptativa de Huffman (explicada en la siguiente sección). Esta técnica requiere de un análisis de un bloque grande datos el cual puede ser codificado con base en su distribución. Antes de la transmisión de la información comprimida se debe almacenar una tabla de símbolos y códigos de Huffman desarrollados para cada símbolo para que los datos codificados puedan ser descodificados con éxito.

No es muy difícil darse cuenta de que las frecuencias cambiantes de las cadenas de datos traen consigo que se requiera el almacenamiento de numerosas tablas junto con la información codificada; estas tablas son una sobrecarga y hacen que la razón de compresión disminuya.

Un segundo método para compensar las diferencias en las distribuciones de frecuencia es el uso de un código de texto plano. El código de texto plano se utiliza para indicar que el carácter debe reproducirse tal como se recibe, esto permite a los caracteres que ocurren poco frecuentemente ser excluidos del proceso de codificación. En este esquema se puede agrupar a los caracteres de baja ocurrencia y asignar un código Huffman que representa la suma de las probabilidades individuales; éste podría ser el código de texto plano para indicar que los 8 bits siguientes representan un carácter no codificado. Sin el uso del código de texto plano se necesitarían cadenas largas de caracteres de 20 ó más bits para representar los caracteres con baja frecuencia de ocurrencia.

2.4.1.2 Códigos de Huffman Modificados.

La representación de caracteres y símbolos por medio de un código de Huffman apropiado es excelente en teoría si se desea tener un promedio de bits por símbolo aproximado a la entropía. En la práctica, sin embargo, se tienen varias dificultades cuando se aplican los códigos de Huffman a ciertas aplicaciones, particularmente en el área de transmisión de facsímiles. Cuando se aplican los códigos de Huffman a la transmisión de facsímiles cada línea de éstos se puede mirar como una serie de *corridas* de pixeles blancos o negros, cada corrida consiste de una serie de elementos similares de la imagen; si se conoce el tipo de la primera corrida de la imagen se conoce entonces el tipo de las sucesivas corridas puesto que éstas deben alternar entre blanco y negro. La probabilidad de ocurrencia de cada corrida de una cierta longitud de pixeles se puede calcular y se asignan códigos cortos para representar las corridas que poseen una alta frecuencia de ocurrencia. Las estadísticas de las probabilidades de la longitud de la corrida asociada con la línea procesada cambian de línea a línea y de documento a documento, de esta manera un código óptimo para una línea o documento en particular puede no ser el adecuado para una línea o documento diferente. Otro grave problema es el hecho de que la creación de los códigos de Huffman en tiempo real requiere un gran poder de procesamiento, normalmente fuera del alcance de las máquinas de facsímil, en las cuales el costo del *scanner*, transmisor/receptor, lógica central y fuente de poder debe estar dentro de unos miles de pesos para que las máquinas sean competitivas en el mercado.

Para disminuir los requerimientos de procesamiento en tiempo real se utiliza una tabla de *lookup*. Puesto que los estándares de la CCITT[8] requieren 1728 pixeles por línea, el uso de la técnica de la tabla de *lookup* requiere almacenamiento para 1728 localidades de longitud variables para cada máquina de facsímil, cada localidad contiene un código binario que corresponde a una longitud de corrida en particular. Los problemas de implementación asociados con el uso de los códigos de Huffman en las aplicaciones de facsímil traen consigo el desarrollo de los códigos de Huffman modificados, los cuales resultan más adecuados a las restricciones del costo de hardware en el mercado competitivo de las máquinas de facsímil.

En el desarrollo de un esquema con códigos de Huffman modificados para aplicaciones de facsímil se ha hecho un cambio, el cual aunque raramente permite que la longitud promedio para codificar un símbolo se acerque a su entropía, permite una compresión aceptable minimizando los requerimientos de hardware y de tiempo de procesamiento. En este esquema la probabilidad de ocurrencia de diferentes longitudes de corridas de pixeles fue calculada para todas las longitudes de corridas de blancos y negros con base en las estadísticas obtenidas del análisis de 11 documentos típicos recomendados por la CCITT.

[8] *Comité Consultatif International Télégraphique et Téléphonique* o Comité Consultor Internacional de Telegrafía y Telefonía, es un comité europeo para la estandarización de formatos de transmisión y comunicaciones, principalmente de facsímiles.

Para reducir el almacenamiento de la tabla de *lookup* los códigos de Huffman fueron truncados con la creación de una representación en base 64 para cada longitud de corrida y la utilización de dos tablas de códigos para reducir el tamaño total de la tabla en comparación con el tamaño que se hubiera requerido si solamente se usara una tabla.

Basándose en las probabilidades de la longitud de las corridas de 11 documentos típicos se han elaborado tablas de códigos con corridas que varían desde 0 a 63 pixeles. Puesto que la probabilidad de la ocurrencia de corridas de pixeles blancos difiere de la frecuencia de ocurrencia de corridas de pixeles negros, se ha desarrollado una tabla que contemple a ambos. Esta tabla se presenta en la **figura 2.17** para corridas de 0 a 63 pixeles de longitud; el código en esta tabla se conoce como código de terminación (TC Terminating Code) y representa el dígito menos significativo de la palabra de código. Para permitir la codificación de corridas que exceden de 63 pixeles se debe emplear una segunda tabla para manejar corridas que van de 64 pixeles a 1728 pixeles. Estos códigos se muestran en la **figura 2.18**, se conocen cómo códigos maestros y representan el dígito más significativo.

Longitud de la Corrida Blanca	Código de Terminación (TC)	Representación en Base 64	Longitud de la Corrida Negra	Código de Terminación (TC)
0	00110101	0	0	0000110111
1	000111	1	1	010
2	0111	2	2	11
3	1000	3	3	10
4	1011	4	4	011
5	1100	5	5	0011
6	1110	6	6	0010
7	1111	7	7	00011
8	10011	8	8	000101
9	10100	9	9	000100
10	00111	a	10	0000100
11	01000	b	11	0000101
12	001000	c	12	0000111
13	000011	d	13	00000100
14	110100	e	14	00000111
15	110101	f	15	000011000
16	101010	g	16	0000010111
17	101011	h	17	0000011000
18	0100111	i	18	0000001000
19	0001100	j	19	00001100111
20	0001000	k	20	00001101000
21	0010111	l	21	00001101100
22	0000011	m	22	00000110111
23	0000100	n	23	00000101000
24	0101000	o	24	00000010111
25	0101011	p	25	00000011000
26	0010011	q	26	000001001010
27	0100100	r	27	000001001011
28	0011000	s	28	000001001100
29	00000010	t	29	000001001101
30	00000011	u	30	000001101000
31	00011010	v	31	000001101001

Parte 1.

Figura 2.17a Códigos de los Dígitos menos Significativos para el Proceso de Huffman Modificado.

Longitud de la Corrida Blanca	Código de Terminación (TC)	Representación en Base 64	Longitud de la Corrida Negra	Código de Terminación (TC)
32	00011011	w	32	000001101010
33	00010010	x	33	000001101011
34	00010011	y	34	000011010010
35	00010100	z	35	000011010011
36	00010101	A	36	000011010100
37	00010110	B	37	000011010101
38	00010111	C	38	000011010110
39	00101000	D	39	000011010111
40	00101001	E	40	000001101100
41	00101010	F	41	000001101101
42	00101011	G	42	000011011010
43	00101100	H	43	000011011011
44	00101101	I	44	000001010100
45	00000100	J	45	000001010101
46	00000101	K	46	000001010110
47	00001010	L	47	000001010111
48	00001011	M	48	000001100100
49	01010010	N	49	000001100101
50	01010011	O	50	000001010010
51	01010100	P	51	000001010011
52	01010101	Q	52	000000100100
53	00100100	R	53	000000110111
54	00100101	S	54	000000111000
55	01011000	T	55	000000100111
56	01011001	U	56	000000101000
57	01011010	V	57	000001011000
58	01011011	W	58	000001011001
59	01001010	X	59	000000101011
60	01001011	Y	60	000000101100
61	00110010	Z	61	000001011010
62	00110011	*	62	000001100110
63	00110100	#	63	000001100111

Parte 2.

Figura 2.17b Códigos de los Dígitos menos Significativos para el Proceso de Huffman Modificado.

Longitud de la Corrida Blanca	Código Maestro	Representación en Base 64	Longitud de la Corrida Negra	Código Maestro
64	11011	1	64	0000001111
128	10010	2	128	000011001000
192	010111	3	192	000011001001
256	0110111	4	256	000001101011
320	00110110	5	320	000000110011
384	00110111	6	384	000000110100
448	01100100	7	448	000000110101
512	01100101	8	512	000000110110 0
576	011010000	9	576	000000110110 1
640	01100111	a	640	000000100101 0
704	011001100	b	704	000000100101 1
768	011001101	c	768	000000100110 0
832	011010010	d	832	000000100110 1
836	011010011	e	836	000000111001 0
960	011010100	f	960	000000111001 1
1024	011010101	g	1024	000000111010 0
1088	011010110	h	1088	000000111010 1
1152	011010111	i	1152	000000111011 0
1216	011011000	j	1216	000000111011 1
1280	011011001	k	1280	000000101001 0
1344	011011010	l	1344	000000101001 1
1408	011011011	m	1408	000000101010 0
1472	010011000	n	1472	000000101010 1
1536	010011001	o	1536	000000101101 0
1600	010011010	p	1600	000000101101 1
1664	011000	q	1664	000000110010 0
1728	010011011	r	1728	000000110010 1
EOL	0000000000		EOL	000000000001

Figura 2.18 Códigos de los Dígitos más Significativos para el Proceso de Huffman Modificado.

Cuando se encuentra una corrida con 63 o menos pixeles se accesa el tipo de código TC para obtener un código en base 64. Para codificar una corrida de 64 pixeles o más se deben usar dos códigos en base 64, para esto se sigue el siguiente procedimiento: primero, se obtiene el código maestro de tal manera que *N*64* (para *1≤N≤27*) no exceda la longitud de la corrida, después se calcula la diferencia entre la longitud de la corrida y *N*64*, y el dígito menos significativo se accesa de la tabla de códigos TC apropiada. La **figura 2.19** muestra un ejemplo de las operaciones en una tabla de *lookup* para una secuencia de corridas blancas y negras de varios tamaños; en la parte superior de esta ilustración se tabula la relación entre una serie datos de video original y su representación con códigos de Huffman.

Datos Originales de Video	Código de Huffman Modificado, en Base 64	Código de Huffman Modificado, en Base 2	
		Código Maestro	TC
5 pixeles negros	5 (negro)	NA	0011
17 pixeles blancos	h (blanco)	NA	101011
32 pixeles negros	w (negro)	NA	000001101010
32 pixeles blancos	w (blanco)	NA	00011011
728 pixeles negros	b0 (negro)	0000001001011	00000010111
1728 pixeles blancos	r0 (blanco)	010011011	00110101
64 pixeles negros	10 (negro)	0000001111	0000110111
55 pixeles blancos	T (blanco)	NA	01011000
1028 pixeles blancos	g2 (blanca)	011010101	0111

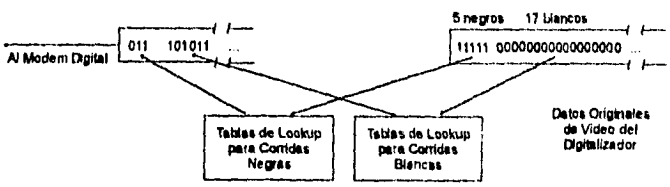


Figura 2.19 Codificación utilizando los Códigos de Huffman Modificados.

Para emplear satisfactoriamente el esquema de codificación modificada de Huffman se deben seguir ciertas reglas para eliminar las deficiencias inherentes en una técnica de codificación estadística. En estas técnicas los códigos no contienen información inherente de la posición la cual es necesaria para la sincronización; esto se compensa estableciendo la regla de que la primera corrida de todas las líneas debe ser blanca, aún si debe ser una corrida de longitud cero, de esta manera las corridas alternan entre blanco y negro; para denotar el principio de cada línea es necesario emplear un delimitador de fin de línea llamado código de fin de línea (EOL, End Of Line), una vez que la línea se codifica se puede rellenar con 0's, si fuera necesario, antes de colocar el carácter EOL para propósitos de sincronía. El resultado de aplicar estas reglas se presenta en la **figura 2.20**.

Con el empleo de los códigos de Huffman modificados el tiempo de transmisión de un documento típico de negocios se ha reducido a menos de 60 segundos con una razón de transmisión de 4800 bauds; el significado de esta reducción de tiempo es clara si se considera que la resolución de 1728 pixeles por línea y 96 líneas horizontales por pulgada son 1 401 048 pixeles en un documento de 8.5 x 11 pulgadas, sin compresión el tiempo aproximado de transmisión sería de 5 minutos sin considerar la transmisión de los códigos de fin de línea.

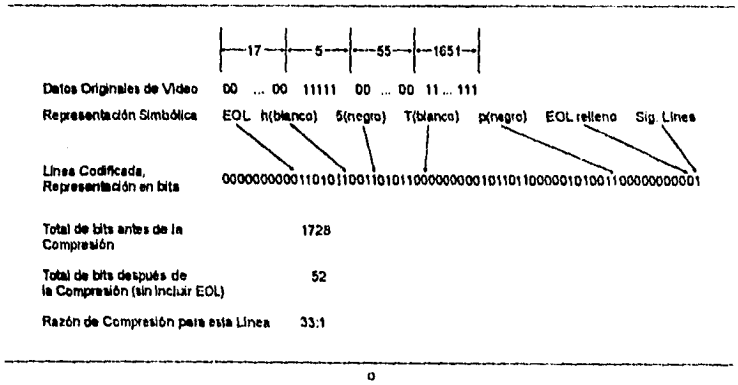


Figura 2.20 Reglas que definen el Formato de Línea.

2.4.2 Algoritmos RLE.

Los algoritmos RLE (Run Length Encoding o codificación de la longitud de corrida) son un método exacto de codificación; se utilizan normalmente para imágenes monocromáticas aunque su uso se ha extendido a imágenes de 16 ó 256 colores. La codificación RLE se utiliza en conjunto con el algoritmo modificado de Huffman así como en los esquemas de compresión de la CCITT.

En la codificación RLE, se combinan corridas consecutivas de pixeles del mismo color y se representan con un código para su transmisión o almacenamiento. Por ejemplo, en la figura 2.21 la longitud de cada corrida (pixeles negros o blancos) se representa con palabras binarias; puesto que las corridas de pixeles blancos alternan con corridas negras, excepto para el comienzo de cada línea, el color de la corrida no necesita almacenarse[9]. Las estadísticas de la longitud de corrida varían de documento a documento porque, en general, éstos no son uniformes.

[9] En el caso de una imagen a colores se debe almacenar explícitamente el color de la corrida.

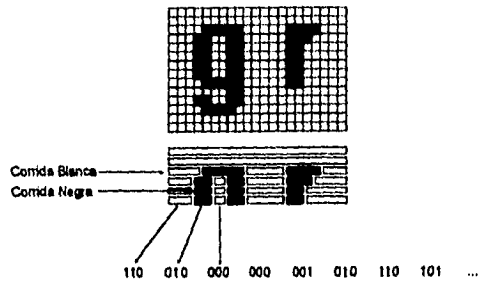


Figura 2.21 Una Imagen en Blanco y Negro digitalizada en una Cuadrícula de Pixeles y su Representación en Términos de Corridas Horizontales.

Los códigos de longitud variable utilizados para las longitudes de corrida hacen uso de la ventaja que representa la distribución de probabilidad de la no uniformidad de las corridas; además, puesto que las estadísticas de los 0's son diferentes que las de los 1's se utilizan diferentes tablas de códigos, una para las corridas de negros y otra para las corridas de blancos. La mayoría de los códigos de longitud variable se han obtenido con la codificación modificada de Huffman.

Puesto que la mayoría de los documentos tiene un poco más de mil pixeles por línea (los documentos estándares de la CCITT tienen 1728), las tablas de códigos podrían, en principio, contener una gran cantidad de palabras de códigos. Sin embargo, para prevenir esta complejidad generalmente se utiliza un código de Huffman modificado en los codificadores prácticos. En este código modificado de Huffman, cada longitud de corrida más grande que un cierto valor (por decir l) se divide en dos longitudes de corrida: una longitud inicial con valor de Nl (donde N es un entero) y una longitud con un valor entre 0 y l-1; esto reduce el número de códigos en la tabla y facilita la implementación del descodificador. Dicha conversión de las longitudes de corrida modifica las estadísticas de los códigos RLE y trae consigo una ligera pérdida en la eficiencia de la compresión. Una ventaja del código de Huffman modificado es su habilidad para manejar corridas de tamaño arbitrario con un tamaño adecuado.

La **figura 2.22** presenta algunas estadísticas relevantes que se pueden utilizar para juzgar la eficiencia de los códigos RLE unidimensionales para las imágenes estándar de la CCITT; es claro que para los documentos que tienen información negra sobre fondo blanco el promedio de la longitud de corrida blanca (r_w) es mucho más grande que el promedio de la corrida negra (r_b). La razón de compresión (1/entropía) varía también ampliamente dependiendo del documento, en promedio, la razón de compresión con esta técnica es 12.

Doc.	r_w	r_b	H_w	H_b	Q_{max}
1	156.3	6.793	5.451	3.592	18.03
2	257.1	14.31	8.163	4.513	21.41
3	89.81	8.515	5.688	3.572	10.62
4	39.00	5.674	4.698	3.124	5.712
5	79.16	6.986	5.740	3.328	9.500
6	138.5	8.038	6.204	3.641	14.89
7	45.32	4.442	5.894	3.068	5.553
8	85.68	70.87	6.862	5.761	12.40

Figura 2.22 **Distribución de Longitud de Corrida para los Documentos Estándares de la CCITT. r_w y r_b representan la longitud promedio de las corridas blancas y negras respectivamente. H_w y H_b son las entropías correspondientes en bits/corrida. Q_{max} es la razón máxima de compresión, es decir, el inverso de la entropía, donde la entropía se expresa en bits/pixel.**

2.4.3 Esquemas de la CCITT.

Existen dos esquemas estandarizados por la CCITT, el primer esquema usa un algoritmo RLE unidimensional mientras que el segundo explota la correlación bidimensional en los datos. Estos esquemas, llamados Grupo (G3) se utilizan principalmente para transmitir documentos de tamaño A4 (210 mm x 298 mm) en redes públicas telefónicas. Puesto que las redes telefónicas están propensas a errores de transmisión, se implementan dos formas de redundancia para prevenir la degradación de la imagen cuando ocurren errores; esto incluye códigos especiales al final de cada línea y la transmisión de un código comprimido unidimensionalmente cada dos o cuatro líneas.

La CCITT ha desarrollado también recomendaciones llamadas Grupo 4 (G4), aplicables a imágenes y líneas de comunicación más generales. Como ejemplo de esto, para las comunicaciones en redes digitales con control de errores, los esquemas G4 se obtienen a partir de los G3 removiendo las dos redundancias mencionadas previamente puesto que la red provee una transmisión libre de errores. Además, los esquemas G4 permiten imágenes más generales incorporando 4 resoluciones posibles: 200, 240, 300 y 400 pixeles por pulgada, y permiten también un modo de operación mezclada de símbolos y gráficas.

Las partes de la imagen que contienen caracteres alfanuméricos se transmiten utilizando formatos tales como el ASCII, mientras que las partes de la imagen que contienen información no-carácter tales como dibujos y escritura a mano se codifican utilizando los esquemas gráficos. En cualquier caso, los esquemas de codificación gráfica utilizados en el Grupo 4 son una variación menor de los esquemas básicos del Grupo 3.

Estos esquemas de compresión de la CCITT se desarrollaron principalmente para la transmisión de facsímiles, sin embargo, algunos formatos comerciales de almacenamiento de imágenes utilizan estos esquemas (por ejemplo, el formato TIFF, descrito en el capítulo 5).

2.4.3.1 Codificación Unidimensional de la CCITT.

Cada línea se codifica asignando palabras a las corridas de elementos blancos o negros que alternan a lo largo de la línea; se asume que estas líneas comienzan con una corrida blanca, si la primera corrida es negra se agrega una corrida blanca de longitud 0. Se utilizan tablas de códigos separadas basadas en los códigos modificados de Huffman y en estadísticas de las 8 imágenes estándares de la CCITT utilizadas para representar corridas blancas y negras.

La tabla de códigos que se utiliza en esta técnicas es la misma tabla de códigos de Huffman modificados descrita en la sección 2.4.1.2. La tabla de códigos contiene códigos para corridas de hasta 1728 pixeles por línea. Estos códigos son de dos tipos: códigos de terminación (TC Terminating Codes) y códigos maestros. Las corridas entre 0 y 63 pixeles se transmiten utilizando una sola palabra de código, las corridas entre 64 y 1728 se transmiten con un código maestro seguido de un TC; el código maestro representa una longitud de $64xN$ (donde N es un entero entre 1 y 27), la cual es igual o menor que la longitud de la corrida que se transmite. La diferencia entre el código maestro y la longitud actual de la corrida se especifica por el TC; a cada línea codificada se le agrega un EOL, el cual es una secuencia única que no puede ocurrir dentro de los datos codificados, el código utilizado es 000000000001, 11 0's seguido de 1. Si el número de bits codificados en una línea es menor que un cierto valor se agrega un relleno de 0's entre el fin de los datos codificados y el código de fin de línea; esto asegura que cada línea codificada requiera de un tiempo fijo de transmisión de manera que el transmisor y el receptor puedan sincronizarse; el estándar mínimo recomendado es 96 bits/línea a una razón de transmisión de 4800 bits/seg, con opciones para 48, 24 y 0 bits, de esta manera los bits de relleno son reconocidos y descartados fácilmente por el receptor. La **figura 2.23** muestra el formato de los datos para varias líneas codificadas; el fin del documento se indica con 6 EOL's consecutivos para formar la señal de retorno de control. El desempeño de la compresión para este esquema RLE se muestra en la **figura 2.24** para 8 documentos estándares de la CCITT, el promedio de compresión es de 9.7

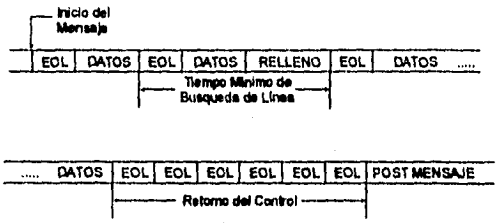


Figura 2.23 **Formato de Mensajes Transmitidos para varias Líneas. El Tiempo Mínimo de Búsqueda de Línea se obtiene utilizando bits de relleno al final de la línea. El final del documento se marca con 6 códigos de fin de línea consecutivos, lo cual indica retorno del control.**

Doc.	Promedio Corrida Blanca	Promedio Corrida Negra	Entropía Corrida Blanca	Entropía Corrida Negra	Razon Máx. Comp.	Razón Comp.
1	134.6	6.790	5.230	3.592	16.02	15.16
2	167.9	14.02	5.989	4.457	17.41	16.67
3	71.50	8.468	5.189	3.587	9.112	8.350
4	36.38	5.673	4.574	3.126	5.461	4.911
5	66.41	6.996	5.280	3.339	8.513	7.927
6	90.65	8.001	5.063	3.651	11.32	10.78
7	39.07	4.442	5.320	3.068	5.188	4.990
8	64.30	60.56	4.427	5.310	11.52	8.665

Figura 2.24 **Desempeño y Estadísticas Relevantes del Esquema Unidimensional para los 8 Documentos Estándares de la CCITT. La razón de compresión máxima se calcula empleando la medición de entropía de las corridas blancas y negras para cada documento. La razón de compresión es calculada utilizando los códigos de Huffman sin el fin de línea y sin los bits de relleno.**

2.4.3.2 Codificación Bidimensional de la CCITT.

Este esquema es conocido como *Código READ Modificado* (READ Relative Element Address Designate o Dirección Designada del Elemento Relativo), y evolucionó a través de varias propuestas emitidas por la CCITT hasta 1979. Este esquema es una esquema de línea por línea en el cual la posición de cada elemento cambiante en la línea presente se codifica con respecto a, ya sea la posición correspondiente de un elemento cambiante en la línea de referencia, la cual se encuentra inmediatamente arriba de la presente línea, o bien con respecto al elemento cambiante precedente en la misma línea presente; después de que la línea presente se ha codificado pasa a ser la línea de referencia para la siguiente línea. El elemento cambiante es un elemento de color diferente al elemento previo a lo largo de la misma línea.

El procedimiento de codificación usa 5 elementos cambiantes definidos como se presenta en la **figura 2.25**:

- a_0: El primer elemento cambiante de la línea presente.
- a_1: El siguiente elemento cambiante de la línea presente; de acuerdo a la definición, tiene un color opuesto a a_0.
- a_2: El elemento cambiante de la línea presente, siguiente a a_1.
- b_1: El elemento cambiante de la línea de referencia a la derecha de a_0 con el mismo color de a_1.
- b_2: El elemento cambiante de la línea de referencia, siguiente a b_1.

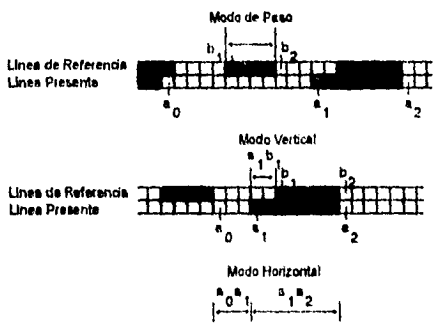


Figura 2.25 **Ejemplos de Modos de Codificación para el Esquema Bidimensional. La configuración superior muestra el modo de paso, la central y la del fondo muestran los modos vertical y horizontal respectivamente.**

Dependiendo de la posición relativa del elemento cambiante que está siendo codificado, el codificador opera en tres modos:

1) Modo de Paso.

En este modo el elemento b_2 se encuentra horizontalmente a la izquierda de a_1. Esto ocurre cuando la corrida blanca o negra en la línea de referencia no es adyacente a la correspondiente corrida blanca o negra en la línea presente. El modo de paso se representa con una palabra de código.

2) Modo Vertical.

En este modo el elemento a_1 está lo suficientemente cerca a b_1 y de esta manera se codifica relativamente con la posición de b_1. Se utiliza solamente si a_1 está a la izquierda o a la derecha de b_1 al menos 3 pixeles, y por lo tanto la distancia relativa a_1b_1 puede tomar cualquiera de los 7 valores V(0), $V_D(1)$, $V_D(2)$, $V_D(3)$, $V_I(1)$, $V_I(2)$ y $V_I(3)$; el subíndice D se utiliza si a_1 está a la derecha de b_1 e I si se encuentra a la izquierda; el número entre paréntesis indica la distancia a_1b_1 en pixeles.

3) Modo Horizontal.

Si a_1 no está lo suficientemente cerca a b_1 entonces su posición debe codificarse en modo horizontal. De esta manera, las corridas a_0a_1 y a_1a_2 se codifican utilizando la concatenación de las tres palabras de código H, $M(a_0a_1)$ y $M(a_1a_2)$. La palabra de código H, tomada como 001, sirve como prefijo o bandera, y $M(a_0a_1)$ y $M(a_1a_2)$ se toman de la tablas de código para representar colores y valores de las corridas a_0a_1 y a_1a_2, esta tabla se basa en los códigos de Huffman modificados igual que lo hacía el esquema unidimensional de la CCITT.

2.4.4 Algoritmos LZ.

Todas las técnicas que utilizan diccionarios tienen que encontrar la manera de almacenar el diccionario para que el descodificador pueda efectuar su función. Como se mencionó en la sección 2.2.3, las técnicas estáticas utilizan un diccionario que no necesita almacenarse porque ya se encuentra definido implícitamente en la técnica, y las técnicas semiadaptativas necesitan almacenar el diccionario antes de enviar el mensaje. Por otro lado, las técnicas adaptativas no necesitan almacenar el diccionario explícitamente; en vez de eso el codificador y el descodificador construyen incrementalmente el diccionario añadiendo a éste cada instancia (o grupo de instancias) que se ha almacenado. En cada punto durante la codificación, se usa el diccionario actual para codificar la siguiente porción del mensaje.

La técnica de Ziv y Lempel es una técnica adaptativa basada en diccionario. La codificación Ziv y Lempel (codificación LZ[10]) se refiere a dos técnicas distintas presentadas por Ziv y Lempel en 1977 y 1978; la idea fundamental detrás de los algoritmos LZ es que una subcadena del mensaje se reemplaza por una referencia a otra subcadena en alguna parte anterior del mensaje; esta referencia puede ser, por ejemplo, un tuplo (longitud, desplazamiento). A continuación se describen los dos algoritmos LZ llamados LZ77 y LZ78 debido a la fecha en la cual fueron dados a conocer.

2.4.4.1 Algoritmo LZ77.

El algoritmo LZ77 tiene dos parámetros fundamentales:

- $N (1 \leq N \leq \infty)$, la longitud del buffer utilizado en la compresión.
- $F (1 \leq F \leq N-1)$, la máxima longitud de la cadena coincidente, con $F << N$.

Los valores típicos que se utilizan en la práctica son 2^{13} para N y 2^4 para F; las potencias de dos, como es usual, son una implementación más eficiente.

La implementación del algoritmos LZ77 se basa en un buffer de desplazamiento de longitud N a través del cual se pasa el mensaje de derecha a izquierda **(figura 2.26)**. Los elementos del buffer se numeran consecutivamente con el 1 a la izquierda y N a la derecha, los N-F elementos de izquierda a derecha son las subcadenas que se van encontrando y definiendo y los F elementos más a la derecha son las posibles subcadenas. La sección A[11] contiene las N-F instancias transmitidas previamente y la sección B almacena las siguientes F instancias que se codificarán.

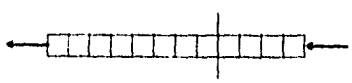


Figura 2.26 El Algoritmos LZ77.

[10] A esta técnica se le conoce como Ziv y Lempel, sin embargo debido a un error histórico, cuando se abrevia se invierten las iniciales de los nombres

[11] En la literatura, a la sección A se le conoce como *Mención* y a la sección B como *Innovación*.

Para comenzar, el algoritmo inicializa la sección A con una cadena predefinida y coloca el inicio del mensaje en la sección A, la codificación se lleva a cabo encontrando la subcadena más larga en el buffer cuyos elementos más a la izquierda se encuentran en la sección A y coincidan primero con cero o más instancias en la sección B y transmitiéndola (junto con la siguiente instancia) como un conjunto de triadas (s, l, a) donde s (en el rango $l \leq s \leq N\text{-}F$) es la posición en la sección A donde comienza la cadena coincidente[12], l (en el rango $0 \leq s \leq F$) es la longitud de la cadena coincidente y a es la instancia siguiente a la cadena coincidente. Después, el mensaje se desplaza en el buffer desde la derecha hasta que la siguiente instancia a ser codificada se encuentra en el elemento más izquierda de la sección B, es decir, el elemento $N\text{-}F+1$.

Se pueden hacer las siguientes observaciones acerca del algoritmo LZ77:

- La instancia extra a que se transmite prevee el caso en el cual no se encontró ninguna coincidencia, es decir cuando $l = 0$.
- La cadena coincidente puede comenzar cerca del fin de la sección A y extenderse a la sección B. Esto funciona debido a que el descodificador tendrá que reconstruir la parte de la cadena coincidente en la sección B en el momento en que la sección en si misma debe ser copiada. Esta característica significa que el algoritmo codifica eficientemente corridas de instancias idénticas.
- Para esta técnica el diccionario consiste en cada subcadena en la sección A; a pesar de esto nunca se almacena porque se actualiza incrementalmente tanto en el codificador como en el descodificador.
- El algoritmo es adaptativo localmente porque su modelo se basa únicamente en las $N\text{-}F$ instancias transmitidas previamente.
- Buscar en el buffer la cadena coincidente más larga es costoso pero limitado por N y F. El algoritmo codifica y descodifica en un tiempo lineal a la longitud del mensaje.
- Descodificar es extremadamente rápido. El descodificador utiliza un buffer idéntico al codificador y copia repetidamente las subcadenas, especificadas por el conjunto de triadas, de la sección A a la sección B.
- Debido a que N y F son finitos, s, l y a pueden empacarse en campos de bits de longitud fija.
- El buffer de desplazamiento se puede implementar utilizando aritmética de módulo N, lo cual elimina la necesidad de un buffer explícito
- Ziv y Lempel demostraron que el desempeño de este algoritmo es tan bueno como las técnicas semi-adaptativas de diccionario.

[12] Esta se indexa tradicionalmente de la posición más a la derecha de la Sección A con (N-F) considerado como la posición 1

2.4.4.2 Algoritmo LZ78.

El algoritmo LZ78 es muy similar al LZ77 excepto que la sección A se reemplaza con un diccionario creciente de d frases (d en el rango de 0 a ∞) numeradas de 0 a d-1, no se limita la longitud de la sección B y el algoritmo no tiene parámetros.

El algoritmo inicializa el diccionario a una sola frase que consiste de la cadena vacía e inicializa d a 1. En cada paso, el algoritmo transmite una nueva frase $p \in S$ en la sección B, la cual consiste de la frase coincidente más larga $m \in S$ en el diccionario, más la siguiente instancia a. De esta manera $p = ma$ y la sección B es igual a "p ..."; p se transmite como un índice (con $log_2 d$ bits) de entrada al diccionario de m's seguido por la instancia a. La nueva frase p se inserta en el diccionario, después, otra parte más del mensaje se desplaza a la sección B del buffer y el proceso continúa. Este proceso lleva a dividir la entrada en frases cada una de las cuales consiste de la frase previa más larga, más una instancia.

Como se muestra en la **figura 2.27**, el algoritmo LZ78 construye un diccionario de frases y repetidamente almacena el número de la entrada más larga al diccionario que coincide la sección B, seguido de una instancia. Después de que cada frase se almacena se añade una nueva frase al diccionario, la nueva frase es la frase almacenada más la instancia siguiente. En la **figura 2.27** se muestra el estado del algoritmo a mitad del procesamiento de la cadena "wooloomooloo".

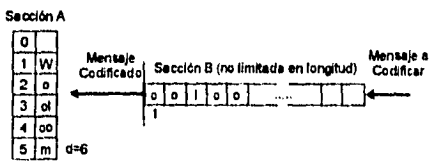


Figura 2.27 El Algoritmo LZ78.

Las características más importantes de este algoritmo son:

- En este algoritmo el diccionario se puede implementar eficientemente utilizando un árbol de búsqueda (**figura 2.28**). Cada paso del *parser* consiste en viajar de la raíz del árbol al nodo correspondiente a m y después añadiendo un nuevo nodo p a m utilizando una rama llamada a.

- Tal como se define el algoritmo, el tamaño del diccionario se incrementa infinitamente, sin embargo, en la práctica la memoria puede agotarse. Para evitar esto, una solución común es vaciar el árbol y continuar el proceso.

El diagrama de la **figura 2.28** muestra al algoritmo a mitad del procesamiento de la cadena "wooloomooloo".

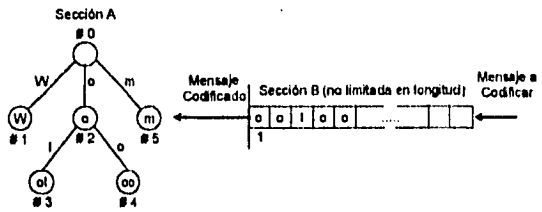


Figura 2.28 El Algoritmo LZ78 implementado con un Árbol.

2.4.4.3 Algoritmo LZW.

Se han sugerido diversos métodos de compresión para los casos en los cuales las estadísticas de los datos no son conocidas inicialmente por el codificador. Básicamente, estas técnicas, llamadas *Algoritmos de Codificación Universal* intentan medir las estadísticas durante el proceso de codificación y adaptarse para maximizar la compresión. Sin embargo, un codificador universal típico no explota la redundancia en los patrones repetitivos de pixeles ni es capaz de remover la correlación entre valores de pixeles en líneas adyacentes, campos o *frames*; aunque se pueden adaptar inicialmente a un número variado de estadísticas, usualmente no son capaces de manejar bien los cambios en las estadísticas dentro de los datos. Estos algoritmos trabajan bien cuando los pixeles se encuentran cuantizados a un número relativamente pequeño de niveles.

El algoritmo LZW (Lempel-Ziv-Welch) es una de los codificadores universales más utilizados. En esta técnica la codificación se efectúa por medio de una tabla de cadenas la cual contiene 2^j cadenas; esta tabla se inicializa simplemente con el conjunto de los 2^k valores posibles de los pixeles (número de colores), y se logra una mejor compresión si $k<<j$, dependiendo de la cantidad de redundancia en los datos.

La codificación comienza definiendo una cadena actual S con el primer pixel de la imagen; se debe notar que S es ya un miembro de la tabla de cadenas. La codificación continúa de la siguiente manera:

1. Si no existen más pixeles (fin de la imagen), escribe el código j-ésimo para *S* y termina. Si no continúa con el paso 2.
2. Se toma el siguiente pixel *P* y se agrega a *S* para formar la cadena *SP*.
3. Si *SP* se encuentra ya en la tabla de cadenas se hace *S* = *SP* y continúa con el paso 1. Si no continua con el paso 4.
4. Escribe el código j-ésimo para *S*.
5. Añade *SP* a la tabla de cadenas si aún existe espacio en ésta.
6. Se hace *S* = *P* y continúa con el paso 1.

Existen muchas variantes de este algoritmo que incrementan la razón de compresión de manera significativa. Por ejemplo, la tabla de cadenas se puede codificar utilizando longitudes de palabra variables, lo cual conduce a una mejor compresión durante la primera parte de la codificación. También se puede utilizar internamente una medida de compresión de tal manera que, cuando ésta cae debajo de cierto valor prefijado, la tabla de cadenas debe ser reinicializada y reconstruida considerando los cambios en las estadísticas.

2.4.5 Algoritmos con Códigos Aritméticos.

Los datos puede comprimirse siempre que los símbolos de los datos ocurran varias veces o tengan parecido entre sí. Se ha demostrado que para obtener los mejores códigos de compresión (en el sentido de la longitud mínima promedio de los códigos), la longitud de la salida debe tener una contribución de $-\log p$ bits de la codificación de cada símbolo cuya probabilidad de ocurrencia es p. Si se puede presentar un modelo exacto para la probabilidad de ocurrencia de cada símbolo en cualquier punto del archivo que se desea comprimir, es posible entonces utilizar la codificación aritmética para codificar los símbolos que se presenten; el número de bits utilizados por la codificación aritmética para codificar un símbolo con probabilidad p es muy cercana a $-\log p$, así que la codificación se encuentra muy cerca para una probabilidad estimada dada.

La idea de la codificación aritmética fue presentada por Shannon en un seminario de 1942 sobre la teoría de la información.

2.4.5.1 Implementación de la Codificación Aritmética.

El algoritmo básico para codificar un archivo utilizando codificación aritmética trabaja conceptualmente de la siguiente manera:

1. Se comienza con el intervalo actual [L, H) inicializado a [0, 1).

2. Para cada símbolo del archivo se ejecutan los 2 pasos siguientes **(figura 2.29)**:

   a) Se subdivide el intervalo actual en intervalos, cada uno de ellos corresponde a un posible símbolo del alfabeto. El tamaño del subintervalo para un símbolo es proporcional a la probabilidad estimada de que ese símbolo será la siguiente entrada, de acuerdo al modelo probabilístico utilizado.

   b) Se selecciona el subintervalo correspondiente al símbolo que se presenta realmente en la entrada, y se hace el intervalo actual.

3. Se escriben los bits necesarios para distinguir el intervalo actual final de los otros posible subintervalos finales.

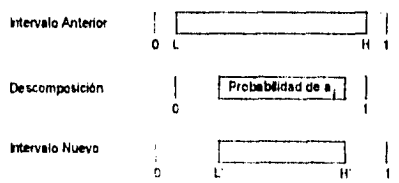


Figura 2.29 **Subdivisión del Intervalo Actual basada en la Probabilidad del siguiente Símbolo de Entrada.**

Claramente, la longitud del intervalo final es igual al producto de la probabilidades de los símbolos individuales, la cual es la probabilidad p de la secuencia particular de símbolos en el archivo. La subdivisión final utiliza casi exactamente $-\log p$ bits para distinguir al archivo de otros posibles archivos. Se necesita además un mecanismo adicional para indicar el fin de archivo, ya sea un símbolo especial de fin de archivo o alguna indicación externa de la longitud del archivo.

Para comprender la codificación aritmética se presenta a continuación un ejemplo con un modelo probabilístico no adaptativo para comprimir un archivo que contiene los símbolos *bbb*, utilizando probabilidades arbitrarias $p_a = 0.4$, $p_b = 0.5$ y $p_{EOF} = 0.1$. El proceso de codificación es el siguiente:

Intervalo Actual	Acción	Subintervalos a	b	EOF	Entrada
[0.000, 1.000)	Subdividir	[0.400, 0.900)	[0.400, 0.900)	[0.900, 1.000)	b
[0.400, 0.900)	Subdividir	[0.400, 0.600)	[0.600, 0.850)	[0.850, 0.900)	b
[0.600, 0.850)	Subdividir	[0.600, 0.700)	[0.700, 0.825)	[0.825, 0.850)	b
[0.700, 0.825)	Subdividir	[0.700, 0.750)	[0.750, 0.812)	[0.812, 0.825)	EOF
[0.812, 0.825)					

El intervalo final (sin redondeo) es [0.8125, 0.825), el cual en representación binaria es aproximadamente (0.11010 00000, 0.11010 01100); se puede identificar este intervalo en forma única asignando la cadena **1101000**. De acuerdo al modelo fijo, la probabilidad p de este archivo en particular es $(0.5)^3(0.1) = 0.0125$, exactamente el tamaño del intervalo final, y la longitud del código (en bits) es $-\log p = 6.322$; en la práctica se utilizan 7 bits.

La implementación básica de la codificación aritmética descrita en el párrafo anterior tiene dos grandes dificultades: disminuir el intervalo actual requiere usar una precisión aritmética alta, y no se produce la salida hasta que se ha leído todo el archivo. La solución más sencilla a estos problemas es escribir cada cadena de bits tan pronto como se genere, y después duplicar la longitud del intervalo actual de manera que refleje solamente la parte desconocida del intervalo final. A continuación se describe en detalle como funciona la codificación utilizando estas mejoras.

Se repiten los siguientes pasos (**figura 2.30**) tantas veces como sea posible:

1. Si el nuevo intervalo no está totalmente dentro de uno de los intervalos [0, 1/2), [1/4, 3/4) o [1/2, 1), terminar y regresar.

2. Si el nuevo intervalo está dentro de [0, 1/2), se escribe 0 y todos los 1's a la izquierda de los símbolos previos, después duplicar el tamaño del intervalo [0, 1/2) expandiéndolo hacia la derecha.

3. Si el nuevo intervalo está dentro de [1/2, 1), se escribe 1 y todos los 0's a la izquierda de los símbolos previos, después se duplica el tamaño del intervalo [1/2, 1) expandiéndolo hacia la izquierda.

4. Si el nuevo intervalo está dentro de [1/4, 3/4), se toma nota de esto hecho para las futuras salidas de 0's y 1's, después se duplica el tamaño del intervalo [1/4, 3/4) expandiéndolo en ambas direcciones a partir del punto medio.

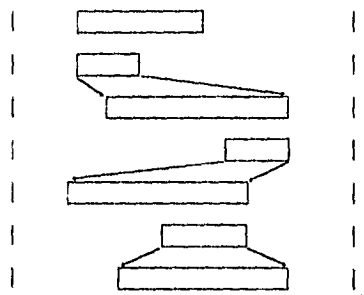


Figura 2.30 Proceso de Expansión de los Intervalos. (a) Si expansión. (b) Intervalo en [0, 1/2). (c) Intervalo en [1/2, 1). (d) Intervalo en [1/4, 3/4).

En el siguiente ejemplo se muestra la codificación del mismo archivo descrito en el ejemplo anterior.

Intervalo Actual	Acción	Subintervalos a	b	EOF	Entrada
[0.00, 1.00)	Subdividir	[0.00, 0.40)	[0.40, 0.90)	[0.90, 1.00)	b
[0.40, 0.90)	Subdividir	[0.40, 0.60)	[0.60, 0.85)	[0.85, 0.90)	b
[0.60, 0.85)	Escribir 1 Expandir [1/2, 1)				
[0.20, 0.70)	Subdividir	[0.20, 0.40)	[0.40, 0.65)	[0.65, 0.70)	b
[0.40, 0.65)	Continuar Expandir [1/4, 3/4)				
[0.30, 0.85)	Subdividir	[0.30, 0.50)	[0.50, 0.75)	[0.75, 0.80)	EOF

(Continuación ...)

Intervalo Actual	Acción	Subintervalos a	Subintervalos b	EOF	Entrada
[0.75, 0.80)	Escribir 10 Expandir [1/2, 1)				
[0.50, 0.60)	Escribir 1 Expandir [1/2, 1)				
[0.00, 0.20)	Escribir 0 Expandir [0, 1/2)				
[0.00, 0.40)	Escribir 0 Expandir [0, 1/2)				
[0.00, 0.80)	Escribir 0				

La salida etiquetada con *continuar* en la sexta línea indica que se está tomando en cuenta que la siguiente salida estará seguida por su opuesto; el bit opuesto es 0 en la novena línea. De esta manera, el archivo codificado es 1101000 como anteriormente se logró.

Es claro que el intervalo actual contiene información acerca de las entradas precedentes; esta información todavía no se ha escrito a la salida y por lo tanto se puede pensar que es el estado del codificador. Si *a* es la longitud del intervalo actual, entonces el estado del codificador contiene *-log a* bits que todavía no se han escrito a la salida; en el método básico presentado, el estado contiene toda la información de la salida puesto que nada se ha escrito hasta que termina el proceso. En el ejemplo ilustrado anteriormente, el estado siempre contiene menos de 2 bits de información escrita a la salida puesto que la longitud del intervalo actual siempre es menor que 1/4. El estado final en el ejemplo descrito es [0, 0.8), el cual contiene *-log 0.8 = 0.332* bits de información.

En la práctica, la aritmética se efectúa utilizando intervalos lo suficientemente grandes para utilizar enteros largos en vez de números en punto flotante. Por ejemplo, en vez de utilizar un intervalo [0, 1) se puede utilizar uno de [0, 100) omitiendo los puntos decimales; además se utilizan los enteros durante el conteo de frecuencia para estimar las probabilidades de un símbolo.

2.4.5.2 Codificación Aritmética Binaria.

En la sección anterior se ha hablado primordialmente de codificación con alfabetos multisímbolos, aunque el mismo principio se aplica también a alfabetos binarios. Es útil distinguir los dos casos puesto que ambos, el codificador y la interface con el modelo son más sencillos para un alfabeto binario. La codificación de imágenes monocromáticas, un problema importante con un alfabeto natural de 2 símbolos, frecuentemente produce probabilidades cercanas a 1, lo cual indica que el uso de la codificación aritmética produce una buena compresión.

En la mayoría de las aplicaciones de compresión de texto e imágenes se utiliza un alfabeto multisímbolo, sin embargo es posible mapear los posibles símbolos a un árbol binario y codificar un evento viajando a través del árbol y codificando la decisión en cada nodo interno. Si se hace esto, el modelo no necesita producir y mantener probabilidades acumulativas; una sola probabilidad es suficiente para codificar cada decisión; sin embargo, ahora se debe codificar más de un evento para cada símbolo de entrada y se tiene un nuevo problema con las estructuras de datos pues es necesario mantener los árboles de codificación sin usar espacio de memoria excesivo.

2.4.5.3 Ventajas y Desventajas.

La ventaja más importante de la codificación aritmética es su flexibilidad puesto que puede utilizarse en conjunto con otro modelo que pueda aportar una secuencia de probabilidades y eventos; esta ventaja es significativa porque se pueden logran grandes razones de compresión a través del uso de modelos probabilísticos sofisticados para los datos de entrada. Los modelos utilizados para la codificación aritmética pueden ser adaptativos, y en realidad se utilizan varios modelos independientes en sucesión para codificar un solo archivo; esta flexibilidad es resultado de la clara separación de los codificadores del proceso de modelado probabilístico. Existe un costo asociado a esta flexibilidad: la interface entre los modelos y el codificador, pues, aunque sea simple, requiere de espacio y tiempo para las estructuras de datos del modelo, especialmente en el caso de entradas con alfabetos multisímbolos.

La otra ventaja importante de la codificación aritmética es su optimización; esta técnica es óptima en teoría y muy cercanamente óptima en la práctica, en el sentido de codificar utilizando la longitud mínima promedio de los códigos. Esta optimización no ha cobrado la debida importancia como lo ha hecho, por ejemplo, los códigos de Huffman, aunque sean igual de eficientes; sin embargo, cuando la probabilidad de un símbolo es cercada a 1, la codificación aritmética logra razones de compresión mucho mayores que otras técnicas.

La desventaja principal de la codificación aritmética es que tiende a ser lenta. Como se vio en la sección anterior, la precisión aritmética requiere de al menos una multiplicación por evento y en algunas implementaciones hasta 2 multiplicaciones y 2 divisiones por evento. Otra desventaja de la codificación aritmética es que tiene una resistencia pobre a los errores, especialmente cuando se utilizan modelos adaptativos; un solo bit erróneo en la codificación trae consigo que el codificador entre en un error interno, haciendo que el resto del archivo se comprima en forma errónea; en realidad este comportamiento lo presentan todas las técnicas adaptativas, incluyendo los algoritmos LZ y de Huffman. En la práctica, la resistencia pobre a los errores se corrige aplicando los códigos adecuados para ello al archivo que está siendo comprimido.

Debido a su óptimo desempeño, se ha propuesto que la codificación aritmética se utilice en conjunto con otras técnicas de compresión tales como los algoritmos LZ. Los valores de salida producidos por estos algoritmos no se encuentran uniformemente distribuidos, lo cual ha sugerido que se puede utilizar un algoritmo con códigos aritméticos para comprimir aún más la salida. De esta manera se aumenta la razón de compresión, aunque también se incrementa la lentitud y complejidad de los algoritmos.

Capítulo 3

Teoría Fractal

Capítulo 3. Teoría Fractal.

La teoría fractal surgió como una alternativa a la geometría euclidiana para estudiar formas y objetos que no compartían las características geométricas de las formas euclidianas y cuya generación por medio de éstas técnicas era muy difícil. La teoría fractal se conoce en general como *geometría fractal* y es de esta geometría sobre la cual se hablará en el presente capítulo.

El capítulo se divide en dos partes. En la sección 3.1 se hace una introducción general a lo que son los fractales, sus bases matemáticas, características, usos y los diversos términos necesarios para comprender la teoría fractal. En la sección 3.2 se habla de los sistemas de funciones iteradas, las cuales son la herramienta fractal para la representación de imágenes del mundo real. Esta última parte es la de más interés debido a que se utiliza como base principal para cumplir el objetivo de esta tesis, el cual es la compresión de imágenes con sistemas de funciones iteradas (técnicas fractales).

3.1 Introducción General a los Fractales.

Los objetos artificiales (hechos por el hombre) tienen superficies planas y/o superficies con curvas suaves, las cuales se estudian y generan con técnicas normales de la geometría euclidiana y la geometría analítica. Sin embargo, los objetos que se encuentran en la naturaleza con frecuencia presentan rugosidad, desigualdades y bordes aleatorios; estas características no pueden generarse fácilmente con métodos geométricos normales; por ejemplo, si se intentara dibujar montañas, ríos, árboles o relámpagos con líneas o polígonos directamente, se requeriría gran cantidad de especificaciones geométricas; sin embargo, si se aplica la geometría fractal, creada por Benoit Mandelbrot, es más fácil y sencillo generar dichas formas.

En las subsecuentes secciones se hablará de las principales características de los fractales, sus diferencias con la geometría euclidiana, los conceptos involucrados en la teoría fractal y algunos ejemplos sencillos y complejos de formas fractales.

3.1.1 Geometría Euclidiana y Geometría Fractal.

La geometría fractal de Mandelbrot proporciona una descripción y un modelo matemático para muchos de los objetos complejos que se encuentran en la naturaleza. Las formas tales como costas, montañas y nubes no se pueden describir fácilmente por medio de la geometría euclidiana y sin embargo, dichas formas presentan una invarianza notable bajo cambios de escala. Esta semejanza estadística a si mismo es la cualidad esencial de los fractales en la naturaleza y puede cuantificarse por medio de una dimensión fractal, es decir, un número que concuerda con nuestra noción intuitiva de dimensión pero no es necesariamente un entero.

En la **figura 3.1** se resumen algunas de las mayores diferencias entre los fractales y las formas tradicionales euclidianas. En primer lugar, los fractales son definitivamente una invención moderna, han sido reconocidos por los científicos a partir de hace unos 10 años. En segundo lugar, mientras que las formas euclidianas tienen uno o pocos tamaños característicos y escalas de longitud (el radio de una esfera, el lado de un cubo, etc.), los fractales (como la costa presentada en la **figura 3.13**) no posee un tamaño característico; se dice que las formas fractales son semejantes a si mismas e independientes de la escala. En tercer lugar, la geometría euclidiana proporciona una descripción exacta de los objetos hechos por el hombre pero es inapropiada para las formas naturales. Finalmente, las formas euclidianas se generan normalmente por medio de fórmulas algebraicas tales como $r^2 = x^2 + y^2$ para definir a un círculo de radio r, los fractales, por otro lado, resultan de la construcción de un algoritmo o procedimiento que frecuentemente es recursivo y es ideal para implementar en las computadoras.

Geometría	
Euclidiana	**Fractal**
- Tradicional (más de 2000 años) - Basada en un tamaño o escala característica. - Adecuada para objetos hechos por el hombre. - Descrita por medio de fórmulas	- Moderna (aproximadamente 10 años). - No tiene una escala o tamaño específico. - Apropiada para formas naturales. - Algoritmo recursivo.

Figura 3.1 Comparación de las Geometrías Euclidiana y Fractal.

3.1.2 Características de los Fractales.

Los objetos fractales se caracterizan principalmente por 3 aspectos: tienen una dimensión no entera (fraccionaria), son semejantes a si mismos, o bien, son afines a si mismos. A continuación se discuten a detalle cada una de estas características.

3.1.2.1 Medición de la Dimensión Fractal.

La introducción a la teoría de los fractales es intuitiva. Supóngase que se tiene un segmento elástico; si este objeto puede deformarse para generar un segmento de línea, se dice que tiene una dimensión $D_t = 1$; si se deforma para generar un plano se dice que tiene una dimensión $D_t = 2$; y si se deforma para generar una esfera o un poliedro se dice que tiene una dimensión $D_t = 3$. A esta dimensión D_t se le conoce como dimensión topológica.

Supóngase ahora otra medida de la dimensión de un objeto. Se tiene un segmento de línea de longitud N y se divide en N partes idénticas, cada una con longitud $l = L/N$. Las partes se asemejan al original, solamente se han escalado con un factor $1/s = l/L$; para formar el segmento de línea original con los segmentos escalados por $1/s$, se deben juntar simplemente las N partes, es decir:

$$N = s^1$$

Considérese ahora un cuadro; si se escala por $1/s$, se obtendrá un cuadro más pequeño; ¿Cuántos cuadros pequeños se necesitan para reconstruir el original? En el caso de $s = 2$, se necesitan 4 cuadros, y para $s = 3$ se necesitan 9; en general se tiene:

$$N = s^2$$

Similarmente, para un cubo, si se escala $1/s$ se encuentra que el número de cubos pequeños necesarios para ensamblar un cubo grande es:

$$N = s^3$$

De esta manera, es posible encontrar una relación: de una línea a un cubo el exponente va de 1 a 3, por lo tanto, dicho exponente es una medida de la dimensión. Por lo tanto, se puede escribir la siguiente definición para la dimensión: si se escala un objeto por s y se deben utilizar N objetos escalados para formar el original, la dimensión D del objeto está dada por:

$$N = s^D$$

A esta dimensión se le conoce como *dimensión fractal* y resolviendo la ecuación anterior se tiene:

$$D = \log N / \log s$$

El proceso de deducción de la dimensión fractal se ilustra en la **figura 3.2**, en la cual se muestra la interpretación de las figuras estándares de dimensión entera en términos de semejanza exacta a si mismo y su extensión a dimensiones fractales no enteras.

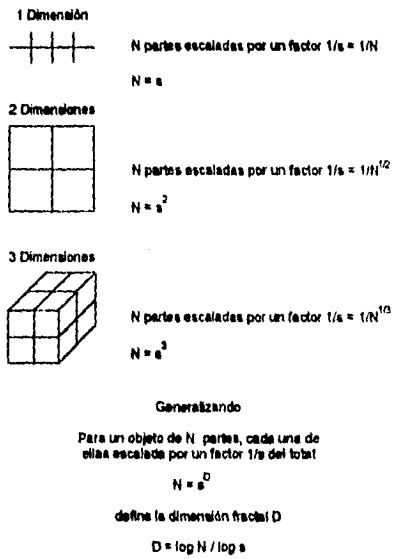


Figura 3.2 Deducción de la Dimensión Fractal.

3.1.2.2 Semejanza a si Mismo, Estadistica y Exacta.

La diferencia entre semejanza y afinidad es importante y se debe tomar en cuenta para la generación de formas fractales. Un objeto similar a si mismo está compuesto de N copias de si mismo (con posibles translaciones y rotaciones) cada una de ellas escalada por un factor r en todas las coordenadas E posibles. Más formalmente, considérese un conjunto S de puntos en las posiciones:

$$x = (x_1, \ldots, x_E)$$

en el espacio euclidiano de dimensión E. Bajo una transformación de semejanza con un factor de escalamiento real $0 < r < 1$, el conjunto S se convierte en rS con los puntos en:

$$rx = (rx_1, \ldots, rx_E)$$

Un conjunto cerrado S es semejante a si mismo cuando S es la unión de los N distintos (y no sobrepuestos) subconjuntos, cada uno de los cuales es *congruente* con rS. *Congruente* significa idéntico bajo la aplicación de translaciones y rotaciones.

3.1.2.3 Afinidad a si Mismo.

En las transformaciones afines, cada una de las coordenadas E de un vector x en un espacio euclidiano pueden escalarse por un diferente factor $(r_1, \ldots, r_E)$. De esta manera, el conjunto S se transforma a $r(S)$ con los puntos en $r(x) = (r_1x_1, \ldots, r_Ex_E)$. Un conjunto cerrado S es afín a si mismo cuando S es la unión de los N distintos (y no sobrepuestos) subconjuntos cada uno de los cuales es congruente con $r(S)$. Similarmente, S es afín estadísticamente a si mismo cuando S es la unión de los N distintos subconjuntos cada uno de los cuales es congruente con la distribución de $r(S)$.

3.1.3 Uso de los Fractales.

Los fractales (palabra creada por Benoit Mandelbrot en 1975) han tenido un crecimiento tremendo en los años anteriores y han ayudado a reconectar la investigación en matemáticas puras con las ciencias naturales y la computación. En los 5 a 10 años pasados la geometría fractal y sus conceptos han llegado a ser una herramienta importante en la mayoría de las ciencias naturales: física, química, biología, geología y metereología. Al mismo tiempo, los fractales han sido de interés para los diseñadores gráficos y cineastas por su capacidad para crear nuevas y excitantes formas y mundos artificiales pero realistas. Las imágenes fractales parecen complejas pero se generan a partir de reglas simples.

La graficación por computadora ha jugado un papel importante en el desarrollo y la aceptación de la geometría fractal como una nueva disciplina válida; el *render* de las imágenes fractales no deja duda de su relevancia para representar formas naturales; conversamente, la geometría fractal juega ahora un papel central en el *render* de imágenes realistas y en el modelado de los fenómenos naturales por medio de la graficación por computadora.

3.1.4 Algunas Formas Fractales Básicas.

Existen algunas construcciones fractales típicas que han sido estudiadas a fondo desde hace muchos años; estas construcciones son sencillas y bellas, y además sirven como un excelente ejemplo para la compresión de la esencia fractal. En las siguientes secciones se presenta como se pueden generar las curvas de Hilbert y Koch, y las líneas y superficies fractales

3.1.4.1 Curva de Hilbert.

Considérese una extraña construcción llamada *Curva de Peano* o *curva llena-espacio*. La curva de Peano de la que se hablará es la *curva de Hilbert*; esta curva se puede construir con las aproximaciones sucesivas descritas a continuación

Se comienza con un cuadro; la primera aproximación es dividir el cuadro en 4 cuadros pequeños y dibujar la curva que conecte el centro de cada punto (**figura 3.3**). La segunda aproximación es subdividir cada uno de estos 4 cuadros y conectar el centro de cada uno de estas divisiones más finas antes de moverse al siguiente cuadro mayor (**figura 3.4**). La tercera aproximación subdivide nuevamente; otra vez, se conectan los centros de las subdivisiones más finas antes de saltar al siguiente nivel de detalle (**figura 3.5**).

Este proceso se sigue indefinidamente; un hecho interesante es que en cada subdivisión, la curva llena un cuadrante pequeño pero nunca cruza una área que ya ha sido ocupada; otro hecho es que la curva se encuentra arbitrariamente cercana a cualquier punto en el cuadro porque la curva pasa a través de los puntos en la cuadrícula, la cual se hace más fina conforme se subdivide más y más. Otro hecho más es que la longitud de la curva es infinita; con cada subdivisión la longitud de la curva se incrementa por un factor de 4; puesto que no existe límite en las subdivisiones se puede decir que no existe límite en cuanto a la longitud.

De esta manera, se ha construido una línea que tiene una dimensión topológica $D_t = 1$; sin embargo, se ha duplicado y "doblado" de tal manera que llena exactamente al cuadro. Es posible encontrar la dimensión fractal para esta curva; en cada subdivisión, la escala cambia por 2 pero la longitud cambia por 4; al igual que para el cuadro, se necesitan 4 curvas a la mitad de la escala para construir el objeto original, de tal manera que la dimensión D está dada por:

$$4 = 2^D$$

$$D = 2$$

La curva de Hilbert tiene una dimensión topológica de 1 pero una dimensión fractal de 2, es decir, conforme la línea se dobla se comporta como un objeto bidimensional.

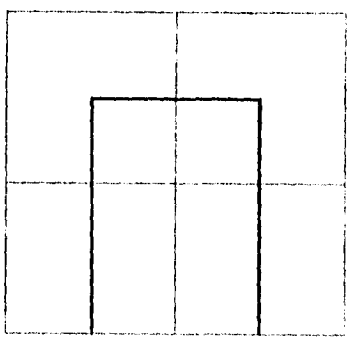


Figura 3.3 Primera Aproximación a la Curva de Hilbert.

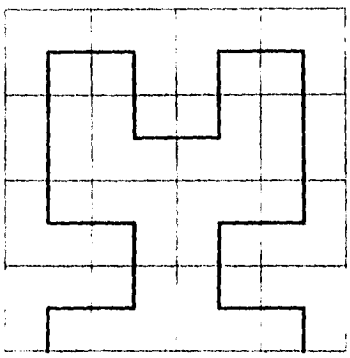


Figura 3.4 Segunda Aproximación a la Curva de Hilbert.

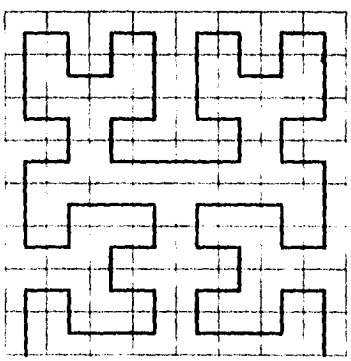


Figura 3.5 Tercera Aproximación a la Curva de Hilbert.

3.1.4.2 Curva de Koch.

La curva de Koch comienza con un segmento de línea; se divide en 3 parte y se remplaza la del centro con dos lados adyacentes para formar un triángulo equilátero (**figura 3.6**). Con esto se obtiene una curva que comienza y termina en el mismo lugar que el segmento de línea original pero construida con 4 segmentos, cada uno de ellos con una longitud de 1/3 del original; de esta manera, la nueva curva tiene una longitud de 4/3 del segmento inicial. Si se efectúa este proceso sobre los 4 segmentos resultantes (**figura 3.7**) la curva es ahora 16/9 de la original; si se repite este proceso indefinidamente se tiene que en cada repetición se incrementa la longitud por un factor de 4/3 y ésta aumenta infinitamente; sin embargo, a diferencia de la curva de Peano, la curva de Koch no llena un área. En realidad, no se desvía mucho de la forma original; tiene una dimensión topológica de 1 y una dimensión fractal de:

$$4 = 3^D$$

de donde:

$$D = \log_3 4 = \log 4 / \log 3$$

$$D = 1.2618$$

Como se puede observar, se tiene una dimensión no integral; la curva de Koch actúa como un objeto entre dimensión 1 y 2. Esto nos conduce a la definición de un fractal: los puntos, curvas y superficies que tengan una dimensión más grande que su dimensión topológica son llamados fractales. La curva de Hilbert y la curva de Koch son fractales porque su dimensión fractal (2 y 1.2628, respectivamente) es más grande que su dimensión topológica de 1.

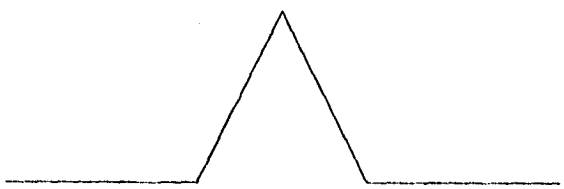


Figura 3.6 Remplazamiento de un Segmento de Línea por una Curva Triádica de Koch.

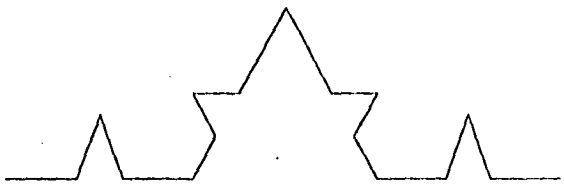


Figura 3.7 Segunda Aproximación a la Curva de Koch.

3.1.4.3 Líneas Fractales.

Se puede utilizar fácilmente la computadora para generar curvas fractales semejantes a si mismas. El dibujo de la curva similar a si misma se puede efectuar con un procedimiento recursivo; una curva compuesta por N partes similares, cada una de ellas escalada por $1/s$, se puede dibujar con una rutina que se llame N veces a si misma con el argumento escalado por $1/s$. Por supuesto, un procedimiento por computadora debe terminar mientras que un fractal no; en la rutina de computadora, cada llamada recursiva tiene un argumento cada vez más pequeño, cuando se alcanza un valor establecido previamente (el argumento es menor que el tamaño de un pixel), la rutina termina y la curva generada es la mejor aproximación a un fractal.

El poder hacer las curvas fractales por computadora significa que el usuario puede generar fácilmente costas realistas, montañas o relámpagos sin preocuparse por definir explícitamente cada una de las irregularidades; la computadora puede generar éstas proporcionándole solamente el punto inicial y el punto final. A la línea generada entre estos puntos se le conoce como línea fractal.

Para generar la línea se toma el punto medio de la línea inicial, se agrega un desplazamiento en ambas coordenadas y se dibujan las líneas resultantes del punto 1 al punto 1' y de éste al punto 2, tal como se muestra en la **figura 3.8**. Se repite este proceso hasta que se alcanza un límite establecido con anterioridad; la línea fractal resultante se presenta en la figura 3.9.

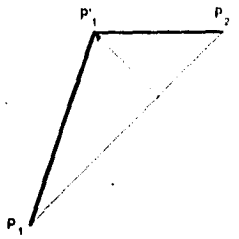


Figura 3.8 Construcción de una Línea Fractal.

Figura 3.9 Una Línea Fractal.

3.1.4.4 Superficies Fractales.

Una línea fractal puede usarse para representar un relámpago, por ejemplo, pero para representar algún objeto natural tridimensional tal como una montaña, es necesario utilizar una superficie fractal. Existen varias ideas para extender la idea fractal a una superficie; la que se presenta aquí está basada en triángulos. Dados 3 puntos en el espacio, se genera una superficie fractal para el área entre ellos; existen métodos para descomponer polígonos arbitrarios en triángulos, de manera que el método utilizado aquí se puede usar para formas más generales. El proceso se efectúa de la siguiente manera: considérese cada lado del triángulo, se genera una línea fractal para cada uno de ellos (**figura 3.10**) y se unen los puntos medios de dichas líneas con otros segmentos de línea (figura **3.11**); de esta manera el triángulo original se ha dividido en 4 subtriángulos. Se aplica el mismo método a cada una de estos triángulos pequeños; se puede continuar esta división hasta que los triángulos sean tan pequeños que ya no tenga caso continuar el proceso (**figura 3.12**)

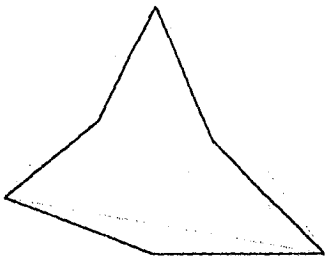


Figura 3.10 Se encuentra el punto medio y se aplica un desplazamiento para cada lado del triángulo.

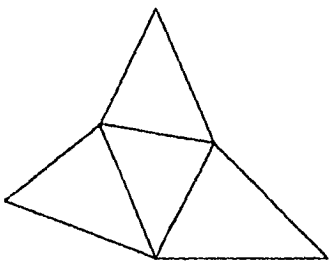


Figura 3.11 Se subdivide el triángulo en 4 triángulos más pequeños.

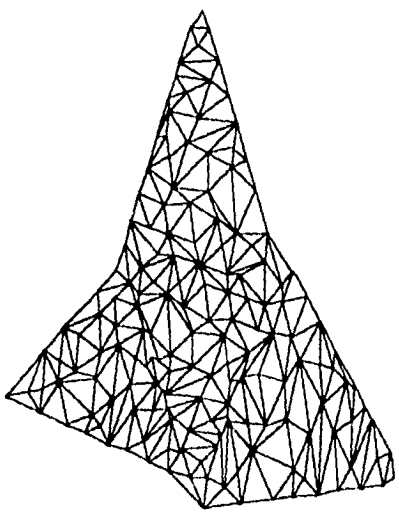


Figura 3.12 Una superficie fractal.

3.1.4.5 Discusión de las Características Fractales.

Las curvas de Hilbert y de Koch, y las líneas y superficies fractales demuestran que la iteración de una regla muy simple puede producir formas complejas con propiedades inusuales. A diferencia de las formas euclidianas, estas curvas tienen detalle en todas las escalas; realmente, mientras más cerca se miren más detalles se encuentran. Otra cuestión importante es que las curvas de Hilbert y de Koch poseen semejanza exacta con si mismas; cada parte pequeña, cuando se amplifica, puede reproducir exactamente una parte más grande, razón por la cual se dice que las curvas son invariantes bajo los cambios de escala. En cada etapa de la construcción de las curvas de Hilbert y de Koch, la curva se incrementa por un factor mayor de 1 (4/2 y 4/3, respectivamente), de esta manera la longitud de la curva se aumenta infinitamente sobre un área finita sin intersectarse a si misma; en las sucesivas iteraciones correspondiente a una amplificación es posible encontrar nuevos detalles. Las líneas y superficies fractales, por otro lado, son un ejemplo claro de semejanza estadística a si mismo, pues aunque cada parte amplificada es similar a la original, no son exactamente iguales.

Finalmente, aunque los algoritmos para generar la curva de Hilbert, la curva de Koch y las líneas y superficies fractales son concisos, simples de describir y fáciles de calcular, no existe una fórmula algebraica que los defina.

3.1.5 Formas Fractales Complejas: Costas, Montañas y Nubes.

La esencia de los fractales en la naturaleza se ilustra en la **figura 3.13** con vistas sucesivas de un planeta fractal ficticio desde una cápsula espacial. En esta figura la imagen inicial sin amplificación se escala por un factor de 4 para dar la siguiente imagen; similarmente, cada imagen sucesiva representa una amplificación de la porción seleccionada en la costa de la imagen previa, hasta que se alcanza una amplificación de 65,536. Esta vista amplificada puede compararse con la vista original sin amplificación, y aunque ambas vistas no son iguales, comparten muchas características similares que es difícil creer que no son diferentes vistas del mismo mapa con la misma amplificación. Esta propiedad de los objetos de que sus subconjuntos amplificados se vean similares o idénticos unos y otros, y al conjunto original se le conoce como *semejanza a si mismo* (definida en la sección 3.1.2.2) y es una característica de los fractales que los diferencia de los objetos euclidianos (los cuales en general, se ven más suavizados a medida que se amplifican).

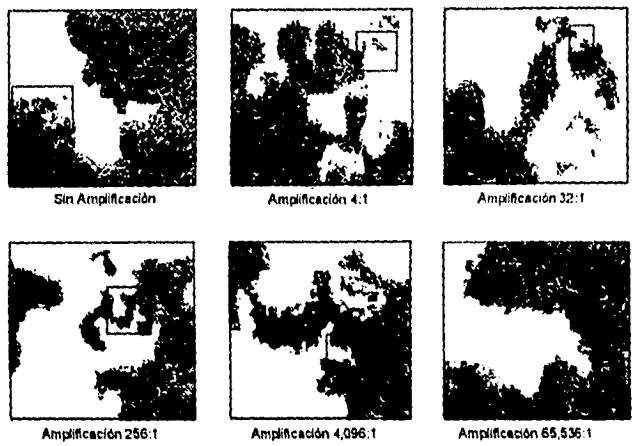


Figura 3.13 Secuencia de *Zooms* de una Costa en una Paisaje Fractal Semejante a si Mismo.

La semejanza exacta a si misma de la curva de Hilbert y de la Curva de Koch puede considerarse como un modelo crudo para la imagen de la costa, pero difiere de ésta en un aspecto. Amplificando partes de la costa se generan segmentos a diferente escala que son similares, pero no exactos a la costa original; sin embargo, el concepto de fractal se puede aplicar a estos objetos conocidos como *semejantes estadisticamente a si mismos*. Durante la medición de la longitud de la costa, si se tiene más cuidado en medir las irregularidades de ésta, la longitud es más grande (un paseo por la costa es más largo que manejar a lo largo de la carretera costera); además, cada sección pequeña de la costa se ve similar (pero no exactamente igual) a una sección grande. Si se utiliza una regla de tamaño r para medir la longitud de la costa, ésta es igual al tamaño r de la regla por el número de veces $N(r)$ que se utilizó la regla:

$$\text{Longitud} = r\,N(r).$$

y al igual que con el copo de nieve de Koch, $N(r)$ varía en promedio como $1/r^D$ y se tiene que:

$$\text{Longitud} \propto r\,\frac{1}{r^D} = \frac{1}{r^{D-1}}$$

con $D > 1$, es decir, mientras que el tamaño de la regla utilizada para medir la costa se decrementa, la longitud se incrementa. Las costas reales pueden, en realidad, caracterizarse con una dimensión fractal D de 1.15 a 1.25, muy cercana a la dimensión fractal *log(4)/log(3)* de la curva de Koch.

La propiedad de aquellos objetos de que se ven similares en diversas escalas de longitud pero se ven distintos en detalle, es la característica principal de los fractales en la naturaleza. La costa de la **figura 3.13** es aleatoria en el sentido de que (a diferencia de la curva de Koch), una vista en una escala muy grande no es suficiente para predecir los detalles exactos de la vista amplificada; además, la manera en la cual varían los detalles cuando se cambia la escala está caracterizada por una dimensión fractal. La superficie fractal mostrada en la **figura 3.13** tiene una $D = 2.2$ y la costa en la misma figura tienen una dimensión fractal menor que la superficie, $D = 1.2$.

3.2 Sistemas de Funciones Iteradas

En esta sección se presenta el problema de modelar geométricamente formas y texturas de imágenes bidimensionales utilizando sistemas de funciones iteradas. La solución a este problema está demostrado con la producción y procesamiento de imágenes sintéticas codificadas a partir fotografías en color. La solución que se ha logrado consiste de dos algoritmos: 1) Un algoritmo para encontrar códigos de funciones iteradas y 2) Un algoritmo de iteración aleatoria para calcular la geometría y textura de las imágenes definidas por los códigos de funciones iteradas. Las razones para desarrollar algoritmos de sistemas de funciones iteradas es su habilidad para producir imágenes complicadas y texturas a partir de pequeñas bases de datos, y su potencialidad para implementarse con métodos de procesamiento en paralelo.

3.2.1 Introducción a los IFS.

La teoría matemática de los sistemas de funciones iteradas (IFS Iterated Function Systems) tiene una gran ventaja para tratar una amplia clase de problemas de modelado, incluyendo el modelado de objetos naturales y escenas. La factibilidad de usar la teoría de los IFS para la producción de imágenes, incluyendo nubes, humo, paisajes terrestres y marinos, e inclusive rostros ha sido probada en varios trabajos de Michael F. Barnsley.

Los métodos presentados en este capítulo tienen sus orígenes en la geometría fractal, cuyas aplicaciones en graficación por computadora ha sido investigada por diversos autores incluyendo Mandelbrot, Kawaguchi, Oppenheimer y otros; en todos los casos el objetivo principal ha sido modelar objetos naturales y escenas y se han utilizado geometrías tanto determinísticas como aleatorias. Este capítulo trata sobre el uso de los IFS, los cuales proveen de una base para, al parecer, un número ilimitado de imágenes; se diferencia de las otras técnicas fractales en que se concentra en la teoría de la medición (*measure theory*) en lugar de la teoría geométrica.

3.2.2 Obtención de Imágenes a Partir de los Códigos IFS.

La obtención de imágenes con textura a partir de los códigos IFS es posible mediante dos procesos. En el primer proceso se obtienen los códigos IFS que definen las características de la imagen; en el segundo proceso se extrae la imagen del modelo asociado con los códigos IFS y se efectúa el proceso de *render*. Este segundo proceso es el que se discute en esta sección; la obtención de los códigos IFS se discute en la siguiente sección.

3.2.2.1 Códigos IFS.

Una transformación afín w: $R^2 \rightarrow R^2$ de un espacio bidimensional R^2 sobre si mismo se define como:

$$w\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}x_2 + b_1 \\ a_{21}x_1 + a_{22}x_2 + b_2 \end{bmatrix}$$

donde las a_{ij}'s y b_i's son constantes reales.

Sea A la matriz (a_{ij}), b el vector $(b_1, b_2)^t$, donde t representa el operador de transposición, y x el vector $(x_1, x_2)^t$, se puede escribir:

$$w(x) = Ax + b$$

De esta manera, una transformación afín se representa completamente por seis números reales.

Dada una transformación afín siempre es posible encontrar un número positivo s tal que:

$$|w(x) - w(y)| \leq s|x - y|$$

El número s más pequeño para el cual la expresión anterior es verdadera se llama *constante de Lipshitz* para w. En este escrito se asume la forma euclidiana para obtener el módulo de un vector:

$$|(x_1, x_2)^t| = \sqrt{x_1^2 + x_2^2}$$

Se dice que la transformación es *contractiva* si s<1 y se llama simétrica si

$$|w(x) - w(y)| = s|x - y| \text{ para toda } (x, y) \text{ en } R^2$$

Es *expansiva* si la constante de Lipshitz es mayor que uno.

Un código IFS bidimensional consiste de un conjunto de *N* transformaciones afines, donde *N* es un entero. El conjunto se denota como:

$$\{w_1, w_2, w_3, \ldots, w_N\}$$

donde cada w_n toma valores de R^2 hacia R^2; se requiere además de un conjunto de probabilidades:

$$\{p_1, p_2, p_3, \ldots, p_N\}$$

donde cada $p_n > 0$ y

$$\sum_{n=1}^{N} p_n = 1$$

Sea s_n la constante de Lipshitz para cada w_n con $n = 1, 2, 3, \ldots, N$, entonces tenemos que un IFS obedece la *condición de la contractividad promedio* si

$$s_1^{p1}, s_2^{p2}, s_3^{p3}, \ldots, s_N^{pN} < 1$$

Un código IFS es un conjunto $\{w_n, p_n : n = 1, 2, \ldots, N\}$ tal que la condición de la contractividad promedio se cumpla.

3.2.2.2 El Modelo Asociado con un Código IFS.

Sea $\{w_n, p_n : n = 1, 2, \ldots, N\}$ un código IFS, entonces existe un objeto geométrico único asociado, subconjunto de R^2, llamado el *atractor* del código IFS y denotado por *A*. *A* tiene la propiedad de ser invariante bajo la aplicación de *N* transformaciones afines $\{w_1, w_2, \ldots, w_n\}$.

En notación de conjuntos se tiene:

$$A = \bigcup_{n=1}^{N} w_n(A)$$

Existe también una *medida invariante* única, soportada por A y denotada como μ; esta medida asigna un número entero no negativo a cada subconjunto de R^2. Puede pensarse en esta medida como una distribución de arena infinitamente fina de la masa total, fija sobre A. La medida de un subconjunto β de A es el peso de la arena sobre β y se denota con $\mu(\beta)$. De esta manera, el modelo asociado con un código IFS consiste del atractor A, junto con una medida asociada μ, y es simbolizado como (A, μ).

La estructura de A es controlada por los mapas afines $\{w_1, w_2, \ldots, w_n\}$ en el código IFS; es decir, los 6 números en el mapa afín especifican la geometría del modelo asociado, el cual a su vez determina la geometría de la imagen asociada. La medida μ está gobernada por las probabilidades $\{p_1, p_2, \ldots, p_n\}$ del código IFS y es esta medida la que proporciona la información para el *render* de las imágenes.

El modelo asociado (A, μ) puede considerarse como un subconjunto de un espacio bidimensional, cuya geometría y textura (controladas por la medida μ) son definidas a la resolución más fina imaginable. La manera en la cual este modelo asociado define imágenes utilizando una proyección a pixeles en un puerto de visión en la computadora se describe en la siguiente sección. El algoritmo para el cálculo de estas imágenes se presenta en la sección 3.2.2.4.

3.2.2.3 Obtención de Imágenes a partir del Modelo Asociado.

Sea (A, μ) el modelo asociado con un código IFS, y sea V un puerto de visión definido por:

$$V = \{ (X, Y) : a \leq X \leq b, c \leq Y \leq d \}$$

Se asume que la intersección $V \cap A$ no es vacía y tiene una medida positiva $\mu(V) > 0$. Para generar la resolución de la vista, se especifica una partición de V en una cuadrícula de LxM rectángulos de la siguiente manera; el intervalo [a, b) se divide en L subintervalos $[X_{l-1}, X_l)$ para $l = 1, 2, \ldots, L$, donde:

$$X_l = a + (b-a)\, l/L$$

Similarmente [c, d) se divide en M subintervalos $[Y_{m-1}, Y_m)$ *para* $m = 1, 2, \ldots, M$, donde:

$$Y_m = c + (d-c)\, m/M$$

Si $V_{l,m}$ denota el rectángulo:

$$V_{l,m} = \{ (X, Y) : X_{l-1} \leq X \leq X_l, Y_{m-1} \leq Y \leq Y_m \}$$

Entonces, el *modelo discretizado* asociado con V a una resolución de LxM se denota como $I(A, \mu, V, L, M)$ y consiste de todos los rectángulos $V_{l,m}$ tales que $\mu(V_{l,m}) \neq 0$ (es decir, todos los rectángulos sobre los cuales reside una masa positiva de arena).

El *render* del modelo discretizado $I(A, \mu, V, L, M)$ asociado con V a una resolución LxM, se efectúa asignado índices RGB de color a cada uno de los rectángulos $V_{l,m}$. Para lograr esta asignación de color se especifica un mapa de color f el cual asocia índices de colores enteros con número reales en [0, 1]. Sea *num-cols* el número de diferentes colores que se usarán; se podría escoger, por ejemplo, 8 tonos de grises en un sistema RGB, entonces *num-cols* = 8 y el índice de color i se asocia con $12.5i$ % de rojo, $12.5i$ % de verde y $12.5i$ % de azul, para $i = 0, 1, \ldots, 7$. El intervalo [0, 1] se divide en subintervalos definidos por números reales C que satisfacen:

$$0 = C_0 < C_1 < C_2 < \ldots < C_{\text{num-cols}} = 1$$

El mapa de color está definido por:

$$f(x) = \begin{cases} i, & \text{si } C_{i-1} < x < C_i \text{ para } i = 0, 1, 2, \ldots, \text{num-cols-1}. \\ \text{num-cols-1}, & \text{si } x = 1. \end{cases}$$

A $I(A, \mu, V, L, M)$ se le aplica el *render* asignando índices de colores $f(\mu(V_{l,m})/\mu(V))$ al rectángulo $V_{l,m}$.

En resumen, el modelo asociado se convierte en una imagen, la cual corresponde a un puerto de visión V con resolución LxM, discretizando a la resolución LxM la parte del atractor que se encuentra dentro del puerto de visión. El *render* de estos valores de discretización son determinados por la medida relativa $\mu(V_{l,m})/\mu(V)$ (la cual corresponde a la proporción de arena que cae dentro del un pixel).

3.2.2.4 El Algoritmo de *Render* de la Medida.

El algoritmo del *render* de la medida comienza con un código IFS $\{w_n, p_n : n = 1, 2, ..., N\}$ junto con un puerto de visión V de resolución LxM, y calcula la imagen IFS asociada, tal como se definió en la sección anterior. Se genera un recorrido aleatorio en R^2 con el código IFS, y se obtiene la medida $\mu(V_{l,m})$ para el pixel a partir de la frecuencia con la cual se visitan los diferentes rectángulos $V_{l,m}$.

Se necesita un punto inicial $(x_0, x_1) \in R^2$ para iniciar el algoritmo. Por simplicidad se asume que la transformación afín $w_1(x) = Ax_1 + b_1$ es una contracción; entonces (x_0, x_0) se puede obtener resolviendo el siguiente sistema de ecuaciones lineales para el punto de w_1:

$$\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} - A_1 \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} b_{11} \\ b_{12} \end{bmatrix}$$

Un arreglo *I* de enteros, de tamaño *LxM* se asocia con el puerto de visión discretizado. También se necesita definir un número de iteraciones, *num-its*, largo comparado con *LxM*. El arreglo *I* se inicializa cero.

La parte del algoritmo para el recorrido aleatorio se efectúa de la siguiente manera:

```
Desde n = 0 hasta num-its haz
        aleatorio = número aleatorio entre [0, 1]
        total = p1; k = 1;
        mientras total < aleatorio haz
                k = k + 1;
                total = total + pk;
        Fin Mientras;

        [ x(n+1) ]   =   wk [ xn ]
        [ y(n+1) ]          [ yn ]

        n1 = int (x(n+1));
        n2 = int (y(n+1));
        I [n1][n2] = I [n1][n2] + 1;

Fin Desde;
```

Finalmente, los elementos del arreglo *I* se normalizan dividiéndolos entre la máxima entrada en el arreglo, *J = max(I [n1][n2])*. Los colores se asignan al arreglo de acuerdo a:

I [n1][n2] = f(I [n1][n2] / J)

Si el número de iteraciones es suficientemente largo, los valores de *render* asignados a un pixel se estabilizan a un valor único.

3.2.3 Determinación de los Códigos IFS.

En un código IFS $\{w_n, p_n : n = 1, 2, ..., N\}$ las w_n's determinan la geometría del modelo asociado, es decir, la estructura del atractor, y las p_n's proporcionan la información para el *render* a través de la medida μ. En esta sección se describe un algoritmo interactivo para determinar las w_n's correspondientes a un determinado modelo. Este algoritmo tiene sus bases en el *Teorema del Collage*, descrito más adelante.

3.2.3.1 El Algoritmo del *Collage*.

El algoritmo comienza con una imagen inicial T, la cual se encuentra dentro de un puerto de visión V, definido como [0, 1] x [0, 1]. T puede ser una imagen digitalizada (por ejemplo, una hoja blanca en un fondo negro), o una aproximación poligonal (por ejemplo, una frontera poligonal para una hoja). Se genera una transformación afín $w_1(x) = A^{(1)}x + b^{(1)}$ con los coeficientes inicializados a $a^{(1)}_{11} = a^{(1)}_{22} = 0.25$, $a^{(1)}_{12} = a^{(1)}_{21} = b^{(1)}_2 = 0$. La imagen $w_1(T)$ se despliega en el monitor en un color diferente a T. En realidad, $w_1(T)$ es una copia de un cuarto del tamaño T, cercana a (0, 0). Ahora, el usuario ajusta interactivamente las $a^{(1)}_{ij}$ utilizando un ratón o el teclado, para que la imagen $w_1(T)$ sea rotada y transladada en la pantalla. La meta del usuario es transformar $w_1(T)$ de tal manera que ocupe una parte la imagen original T, es decir, $w_1(T)$ es representada como un subconjunto de pixeles de T; es importante que la dimensión de $w_1(T)$ sea más pequeña que la dimensión de T para asegurar que w_1 es una contracción. Una vez que $w_1(T)$ se encuentra colocada de manera adecuada en T, se introduce una nueva copia $w_2(T)$, ahora, se ajusta $w_2(T)$ para que ocupe una porción de T, que no esté ocupada por $w_1(T)$; se permite que haya sobreposición de $w_1(T)$ y $w_2(T)$ pero por motivos de eficiencia debe ser lo mínimo posible. De esta manera el usuario determina un conjunto de transformaciones afines contractivas $\{w_1, w_2, ..., w_n\}$ con la propiedad de que la imagen original T y el conjunto:

$$\bar{T} = \bigcup_{n=1}^{N} w_n(T)$$

se encuentran visualmente cerca, siempre que N sea lo más pequeño posible.

El indicador matemático de la cercanía de T y $\tilde{T}$ es la distancia de Hausdorff $h(T, \tilde{T})$ definida más adelante; visualmente cercano significa que $h(T, \tilde{T})$ es pequeña. Los mapas que se calculan con este algoritmo se almacenan para su posterior uso. El teorema del *Collage*, definido a continuación, asegura que el atractor de cualquier código IFS que utilice esos mapas también será visualmente cercano a T. Además, si $T = \tilde{T}$, más exacto si $h(T, \tilde{T}) = 0$, entonces $A = T$. De esta manera, el algoritmo proporciona códigos IFS tales que la geometría del modelo asociado se asemeja al original.

El algoritmo se ilustra en la **figura 3.14**, en la cual se muestra una imagen poligonal de una hoja T, en la parte superior e inferior izquierda. En cada caso T se ha cubierto aproximadamente por 4 transformaciones afines de si misma; en la imagen superior esto se ha hecho adecuadamente, mientras que en la inferior se ha realizado pobremente. Los atractores correspondientes se muestran en la parte derecha de la figura, el superior es mucho más parecido a la imagen original que el inferior debido a que el *collage* (pegado) es mejor. El proceso también se ilustra en la **figura 3.15**, la cual muestra un *collage*, los códigos IFS que determina, y el atractor del IFS.

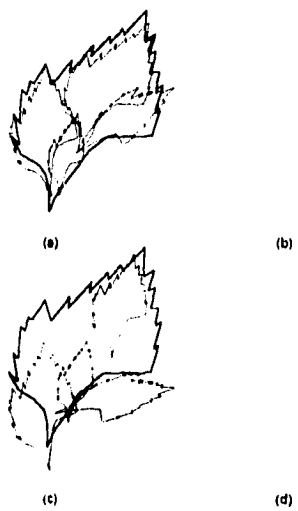
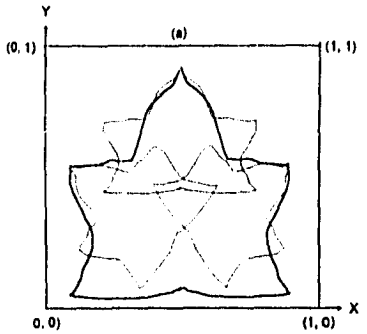


Figura 3.14 **Ilustración del Teorema del *Collage*. El *collage* superior (a) es mejor que el inferior (c), así que el atractor correspondiente, mostrado en la esquina superior derecha (b), se asemeja más a la imagen original que el atractor en la parte inferior (d).**



(b)

(c)

$w_i(z) = s_i Z + (1-s_i) a_i$	
$s_1 = 0.6$ $s_2 = 0.6$ $s_3 = 0.4 - 0.3i$ $s_4 = 0.4 + 0.3i$	$a_1 = 0.45 + 0.9i$ $a_2 = 0.45 + 0.3i$ $a_3 = 0.60 - 0.3i$ $a_4 = 0.30 + 0.3i$

Figura 3.15 **Ilustración de como se aplica el Teorema del *Collage*. El *collage* en (a) determina los códigos IFS de la tabla (c), los cuales fijan el atractor (b).**

3.2.3.2 La Distancia de Hausdorff y el Teorema del *Collage*.

Como se mencionó anteriormente, la distancia de Hausdorff mide el grado de error en el cálculo de la imagen IFS y la imagen original. La distancia de Hausdorff $h(A, B)$ entre dos subconjuntos cerrados de R^2 se define como:

$$h(A, B) = \max \{ \max_{x \in A} \min_{y \in B} | x - y| ; \max_{y \in B} \min_{x \in A} | y - x| \}$$

El teorema del *Collage* define una sentencia acerca de la distancia de Hausdorff entre el atractor A de un IFS y la imagen original T.

Teorema del Collage: Sea $\{w_n, p_n : n = 1, 2, ..., N\}$ un código IFS de mapas afines contractivos. Sea $s < 1$ la constante de Lipshitz más grande permitida para los mapas. Sea $\xi > 0$ cualquier número positivo. Sea T un subconjunto cerrado de R^2, y supóngase que los mapas w_n se han escogido de tal manera que:

$$h\left(T, \bigcup_{n=1}^{N} w_n(T)\right) < \xi$$

Entonces:

$$h(T, A) < \frac{\xi}{1-s}$$

donde A denota el atractor del IFS.

3.2.3.3 Obtención de las Transformaciones Afines.

Para entender como se aplica el Teorema del *collage* y su correspondiente algoritmo para obtener mapas afines, es muy importante ver como dos imágenes de un mismo objeto pueden utilizarse para determinar la transformación afín:

$$w: R^2 \rightarrow R^2$$

Para ilustrar esta idea se comienza con dos copias de hojas de hiedra, una pequeña y una grande, tal como se muestra en la **figura 3.16**. Lo que se desea es encontrar los números reales a, b, c, d, e, f, de tal manera que la transformación, escrita en forma matricial como:

$$w\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by + e \\ cx + dy + f \end{bmatrix}$$

tenga la propiedad de *w(hoja grande)* sea aproximadamente igual a *w(hoja pequeña)*.

El primero paso es colocar ejes cartesianos x y y, como ya está en la figura 3.16. Se marcan 3 puntos en la hoja grande (pueden ser 3 puntos cualesquiera) y se determinan sus coordenadas (α_1, α_2), (β_1, β_2) y (γ_1, γ_2). Se marcan los correspondientes puntos en la hoja pequeña y se determinan también sus coordenadas (α'_1, α'_2), (β'_1, β'_2) y (γ'_1, γ'_2).

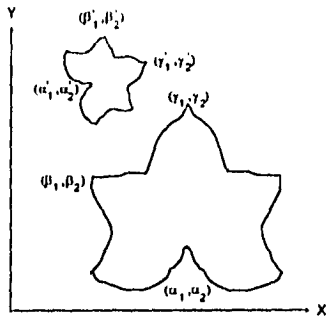


Figura 3.16 Dos hojas de hiedra para definir una transformación afín $w: R^2 \rightarrow R^2$.

Entonces *a*, *b* y *e* se obtienen resolviendo el siguiente sistema de ecuaciones lineales:

$$\alpha_1 a + \alpha_2 b + e = \alpha'_1$$
$$\beta_1 a + \beta_2 b + e = \beta'_1$$
$$\gamma_1 a + \gamma_2 b + e = \gamma'_1$$

mientras que *c*, *d* y *f* satisfacen:

$$\alpha_1 c + \alpha_2 d + f = \alpha'_2$$
$$\beta_1 c + \beta_2 d + f = \beta'_2$$
$$\gamma_1 c + \gamma_2 d + f = \gamma'_2$$

3.2.4 Estructuras Jerárquicas.

El modelado de escenas naturales es una meta difícil en la graficación por computadora; las fotografías de escenas naturales contienen información redundante en la forma de patrones sutiles y sus variaciones. El análisis de estos patrones es el foco de interés de la investigación actual en la graficación por computadora.

Dos características de las escenas naturales han sido esclarecidas; estas son a) La presencia de estructuras geométricas complejas en todas las escalas de observación, puesto que las texturas y fronteras naturales no se suavizan a medida que se amplifican sino que siempre guardan algo de rugosidad, y b) El esquema jerárquico de los objetos primitivos, puesto que se ha descubierto que las escenas naturales se encuentran organizadas jerárquicamente, es decir, los bosques están formados por árboles, éstos consisten de ramas y a su vez éstas se encuentran formadas por hojas, etc. Estos dos aspectos importantes pueden integrarse dentro del modelado con códigos IFS.

3.2.4.1 Conjuntos de Condensación e IFS's Jerárquicos.

En esta sección se presentan algunos de los aspectos de los códigos IFS con condensación, los cuales pueden utilizarse para un *render* con estructuras jerárquicas.

Como en las secciones previas, el espacio métrico es el plano euclidiano R^2. Un código IFS con condensación se denota como $\{w_0, w_1, w_2, \ldots, w_N\}$ y consiste de los mapas afines $\{w_1, w_2, \ldots, w_N\}$ más un mapa predefinido w_0. Este mapa se define con un subconjunto cerrado C en R^2, de acuerdo a:

$$w_0(x) = C; \quad \text{para toda } x \in R^2$$

C es conocido como el *conjunto de condensación* del IFS. Se puede demostrar que su atractor A_C es único y está definido por la ecuación:

$$A_C = \bigcup_{n=0}^{N} w_n(A_C)$$

Si C es el conjunto vacío esto se reduce a la situación ya conocida. Si C no es vacío se tiene que:

$$A_C = \text{cerradura } \{C \cup \bigcup_{n=0}^{\infty} W^{on}(C)\}$$

donde $W(C)$ denota la unión de las imágenes del conjunto C bajo las N transformaciones afines $w_1, w_2, \ldots, w_N$, es decir:

$$W(C) = \bigcup_{n=1}^{N} w_n(C)$$

Se debe notar que w_0 no es una transformación afín sino una contracción debido a que es un conjunto de mapas.

3.2.4.2 Teorema del *Collage* con Condensación.

Un código IFS con condensación puede usarse para proporcionar un modelo global de un conjunto estructurado de objetos geométricos tales como campos de flores, un árbol, una huerta o un bosque. Esto es factible debido a que el Teorema del *Collage* puede hacerse extensivo al caso de los códigos IFS con condensación. Cuando estos se implementan en software proporcionan al usuario un medio para definir una colección de transformaciones afines contractivas $\{w_0, w_1, w_2, \ldots, w_N\}$ junto con un conjunto de condensación C de tal manera que el atractor asociado con el código IFS con condensación es visualmente cercano a una imagen dada T. A continuación se presenta un enunciado formal del teorema.

Collage con Condensación: Sea $\{w_n: n = 1, 2, \ldots, N\}$ un código IFS de mapas afines contractivos. Sea C un subconjunto cerrado de R^2. Sea w_0 el conjunto de transformaciones asociadas con C, definido por $w_0(B) = C$ para todos los subconjuntos cerrados no vacíos $B \subset R^2$. Sea $s < 1$ la constante de Lipshitz más grande permitida para los mapas. Sea $\xi > 0$ cualquier número positivo. Sea T un subconjunto cerrado de R^2, y supóngase que los mapas w_n se han escogido de tal manera que:

$$h\left(T, \bigcup_{n=1}^{N} w_n(T)\right) < \xi$$

Entonces:

$$h(T, A_C) < \frac{\xi}{1-s}$$

donde A_C denota el atractor de $\{w_n: n = 1, 2, \ldots, N\}$.

En la **figura 3.17** se ilustra la aplicación del Teorema del *Collage* con condensación. En la parte izquierda se muestra una imagen sombreada en la cual se ha sobrepuesto en blanco un *collage* de condensación, consistiendo en cada caso de un conjunto de condensación y dos transformaciones afines en la imagen. En cada uno de los casos el *collage* determina un IFS con condensación diferente. En el lado derecho de cada imagen se muestra el atractor del IFS asociado. El teorema dice que mientras más cercana es la imagen blanca (es decir, la unión del conjunto de condensación junto con las copias afines de la imagen original) a la imagen sombreada, el atractor del IFS con condensación asociado estará más cercano a la imagen sombreada. Esta cercanía se mide objetivamente con la distancia de Hausdorff; el *collage* superior es mejor que el del fondo y esto se debe a que se hizo coincidir subjetivamente el atractor y la imagen original.

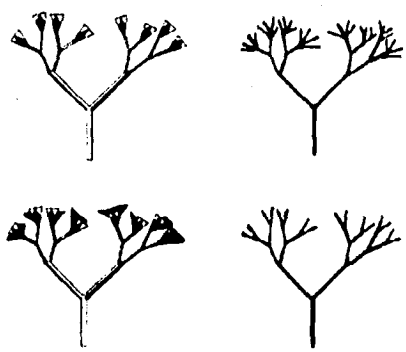


Figura 3.17 Ilustración del Teorema del *Collage* con Condensación.

Para obtener un código jerárquico IFS se escoge a C para que sea el atractor de un código IFS $\{v_m: m = 1, 2, ..., M\}$. De esta manera, el par ordenado del IFS $\{ \{v_m\}, \{w_n\} \}$ define un subconjunto único de R^2. Después, nuevamente se puede escoger $\{v_m\}$ para que sea un código IFS con condensación y el proceso se repite para obtener una triada ordenada del IFS, y así, sucesivamente; esto permite construir modelos más y más complicados.

Capítulo 4

Estructuras de Árboles Cuádruples

Capítulo 4. Estructuras de Árboles Cuádruples.

Los árboles cuádruples son estructuras jerárquicas para representar datos espaciales y están basados en el principio de descomposición recursiva, tal como se verá más adelante. Este tipo de estructura de datos se utiliza en aplicaciones de graficación por computadora, procesamiento digital de imágenes, sistemas de información geográfica, visión computarizada, modelado de sólidos, y en muchas otras aplicaciones.

Este capítulo se centra en el uso de los árboles cuádruples para representar regiones similares en imágenes; estas regiones se utilizan a su vez para la aplicación de los sistemas de funciones iteradas (explicadas en el capítulo 3), con el objeto de comprimir la imagen. La forma específica de hacerlo se detalla en el capítulo 6.

El capítulo se divide en tres partes, en la primera se dan los fundamentos teóricos generales de los árboles cuádruples, la segunda trata someramente de las técnicas de búsqueda, y en la tercera se habla específicamente de la obtención de árboles cuádruples para representar imágenes (o regiones de la misma).

4.1 Teoría General.

Existen diversas técnicas para representar datos con estructuras jerárquicas; una de las técnicas más comúnmente utilizada son los árboles cuádruples, basados en una descomposición recursiva de la imagen. Esta técnica ha evolucionado a partir del trabajo en diferentes campos de la investigación en ciencias de la computación, por lo tanto, es natural que hayan numerosas adaptaciones de los árboles cuádruples para cada tipo de datos. El desarrollo de estas estructuras ha sido motivado por el deseo de reducir los requerimientos de almacenamiento por medio del remplazamiento de valores idénticos de los datos y el uso estructuras de datos más eficientes para su representación.

Esta sección contiene una introducción a la teoría de los árboles cuádruples, cuyo objetivo es mostrar el uso de los árboles en la representación de regiones y definir las propiedades principales de estas estructuras; además, se mencionan las definiciones básicas para comprender la teoría de los árboles cuádruples, un resumen de su historia y la manera de implementarlos.

4.1.1 Definiciones Básicas.

Para comenzar, primero es necesario definir unos cuantos términos con respecto a datos bidimensionales (las imágenes son un buen ejemplo de ello y son precisamente en las cuales estamos interesados). Si se tiene un arreglo en dos dimensiones de pixeles, a este arreglo se le denota *imagen*; si los pixeles son negros y blancos se dice que la imagen es *binaria*, si existen tonos de grises se dice que la imagen es una imagen en *escala de grises*; el *borde* de la imagen son los pixeles que se encuentran fuera del arreglo que define a la imagen.

Se dice que dos pixeles son *tetra-adyacentes* si se encuentran situados uno al lado del otro, ya sea en posición vertical u horizontal; si el concepto de adyacencia también incluye los pixeles diagonales, se dice que los dos pixeles son *octa-adyacentes*. Un conjunto S de pixeles es *tetra-conectado* si para cualquiera de los pixeles p, q en S existe una secuencia de pixeles $p = p_0, p_1, \cdots, p_n = q$ en S, tal que p_{i+1} es tetra-adyacente a p_i para $0 \leq i \leq n$; se aplica el mismo concepto para un conjunto S *octa-conectado*.

Una región negra, o *componente negro tetra-conectado*, es el máximo conjunto tetra-conectado de pixeles negros; el proceso de asignar la misma etiqueta a todos los pixeles negros tetra-adyacentes es conocido como *Etiquetamiento del componente conectado*. Una región blanca es el conjunto máximo octa-conectado de pixeles blancos, de manera análoga a la región negra. El complemento de las regiones negras consiste de la unión de las regiones blancas octa-conectadas, esta unión forma el fondo blanco; las otras regiones blancas, si las hay, se llaman *hoyos* en la región negra. De esta manera, la región negra R está rodeada por una región blanca infinita, y R rodea a su vez a otras regiones blancas.

Se considera que un pixel tiene cuatro lados, cada uno de ellos de longitud 1; la *frontera* de la región blanca consiste del conjunto de lados de sus pixeles constituyentes que también sirven como lado para los pixeles negros. Se pueden formular definiciones similares con base en bloques rectangulares, cuyos pixeles tienen el mismo color; por ejemplo, dos bloques disjuntos P y Q son tetra-adyacentes si existe un pixel p en P y un pixel q en Q de tal manera que p y q son tetra-adyacentes.

Finalmente, aunque las siguientes secciones hablan de la aplicación de los árboles cuádruples a imágenes binarias, se debe señalar que esta técnica se aplica también a imágenes a color, aunque por supuesto, la complejidad de los algoritmos es mayor.

4.1.2 Propiedades de los Árboles Cuádruples.

El término árbol cuádruple se utiliza para describir una clase de estructura jerárquica de datos, cuya propiedad común es estar basada en el principio de descomposición recursiva del espacio. Los árboles cuádruples pueden diferenciarse de acuerdo con los siguientes criterios:

1. El tipo de datos para los cuales se utiliza.
2. El principio que guía al proceso de descomposición.
3. La resolución (variable o no).

Actualmente, los árboles cuádruples se utilizan para datos puntuales, áreas, curvas, superficies y volúmenes. La descomposición puede ser en partes iguales en cada nivel, llamada *descomposición regular*, o puede estar gobernada por la entrada; en la graficación por computadora, a esta distinción se le conoce como *jerarquías espacio-imagen* contra *jerarquías espacio-objeto*, respectivamente. La resolución de la descomposición, es decir, el número de veces que se aplica el proceso de descomposición, puede fijarse de antemano o definirse de acuerdo a las propiedades de los datos de entrada; para algunas aplicaciones se pueden diferenciar estructuras basándose en la especificación de fronteras (curvas y superficies) o en la organización de su interior (áreas y volúmenes).

El caso más usual para el cual se utiliza el árbol cuádruple es la representación de una región binaria; a este árbol se le conoce como *árbol cuádruple de región*, pero se abreviará de aquí en adelante solamente como árbol cuádruple, pues es el tipo de árbol que se estudiará en este capítulo. Este árbol está basado en la subdivisión sucesiva de un arreglo de una imagen binaria en 4 subcuadrantes de igual tamaño; si el arreglo no consiste enteramente de 1's ó 0's (la región no cubre a todo el arreglo), se subdivide en cuadrantes, subcuadrantes y así, hasta que los bloques que se obtengan contenga solamente 0's ó 1's; el árbol cuádruple puede caracterizarse como una estructura de datos de resolución variable.

Como ejemplo de árbol cuádruple considérese la región mostrada en la **figura 4.1a**, representada por el arreglo binario de $2^3 \times 2^3$ de la **figura 4.1b**. Se puede observar que los 1's corresponden a los pixeles dentro de la región y los 0's a pixeles fuera de la región; los bloques resultantes del arreglo de la **figura 4.1b** se presentan en la **figura 4.1c**. Este proceso se representa con un árbol de grado 4, es decir, cada nodo no terminal tiene 4 hijos.

En la representación en árbol, el nodo raíz corresponde a todo el arreglo; cada hijo de un nodo es un cuadrante de la región representada por ese nodo; este nodo se etiqueta en el orden NO, NE, SO, SE, denotando noroeste, noreste, suroeste y sureste, respectivamente. Los nodos terminales del árbol corresponden a aquellos bloques para los cuales ya no es necesaria la subdivisión; se dice que un nodo terminal es blanco o negro dependiendo de si un bloque está fuera de la región (contiene solamente 0's) o dentro de la región (contiene solamente 1's); todos los nodos no terminales son grises, es decir, sus bloques contienen 0's y 1's. En una imagen de $2^n \times 2^n$, el nodo raíz se encuentra en el nivel n, mientras que un nodo en el nivel 0 corresponde a un solo pixel de la imagen[1]; la representación del árbol cuádruple para la figura 4.1c se muestra en la figura 4.1d; los nodos terminales se etiquetan con números y los nodos no terminales se etiquetan con letras, también se muestran los niveles del árbol.

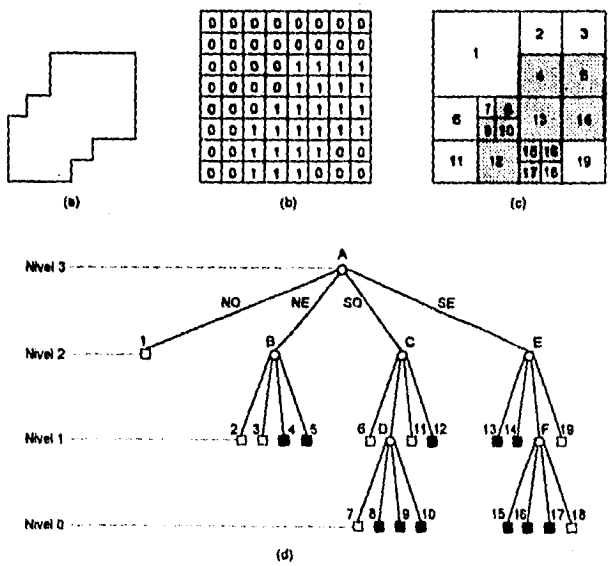


Figura 4.1 **Un ejemplo de (a) Una región, (b) Su arreglo binario, (c) Sus bloques, y (d) el árbol cuádruple correspondiente.**

[1] Alternativamente, se puede decir que el nodo raíz está a una profundidad 0 mientras que el nodo a la profundidad n corresponde a un solo pixel de la imagen. Ambos conceptos, nivel o profundidad, se utilizan dependiendo del contexto.

Se utilizan cuadrados en la descomposición del árbol cuádruple debido a que la descomposición resultante satisface las siguientes dos propiedades:

- Es una partición basada en un patrón repetitivo infinito y por lo tanto puede utilizarse con imágenes de cualquier tamaño.
- Es una partición que se puede descomponer infinitamente en patrones más finos, es decir, en mayor resolución.

Una descomposición en triángulos equiláteros, presentada en la **figura 4.2a**, también satisface estos criterios; sin embargo, a diferencia de la descomposición en cuadrados, no tiene una orientación uniforme, es decir, todos las subtriángulos con la misma orientación no puede mapearse a los que tienen diferente orientación utilizando simplemente translaciones y no rotaciones. En contraste, una descomposición en hexágonos (**figura 4.2b**) tiene una orientación uniforme pero no satisface la segunda propiedad.

La principal motivación para desarrollar los árboles cuádruples es el objetivo de reducir la cantidad de espacio necesario para almacenar datos a través del uso de agregación de bloques homogéneos. Sin embargo, una implementación con árboles cuádruples tiene una sobrecarga generada por los nodos no terminales; para una imagen con B bloque blancos y N bloques negros se requieren $4(B+W)/3$ nodos; en contraste, un arreglo binario con una imagen de 2^n x 2^n requiere solamente 2^{2n} bits.

El peor caso para un árbol cuádruple con una profundidad dada, en términos de requerimientos de almacenamiento, ocurre cuando la región corresponde a un patrón similar al tablero de ajedrez, tal como se muestra en la **figura 4.3**. La cantidad de espacio requerida es función de la resolución (es decir, el número de niveles en el árbol) y el tamaño de la imagen (es decir, su perímetro).

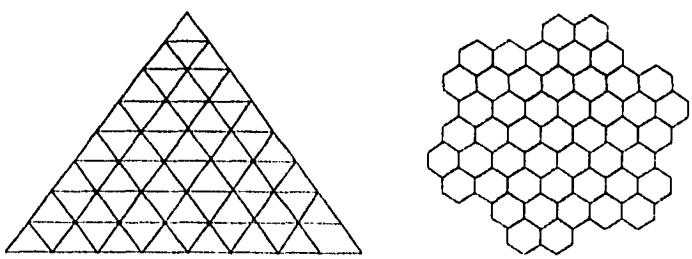


Figura 4.2 Ejemplos de particiones no cuadradas: (a) Triángulos equiláteros, y (b) Hexágonos.

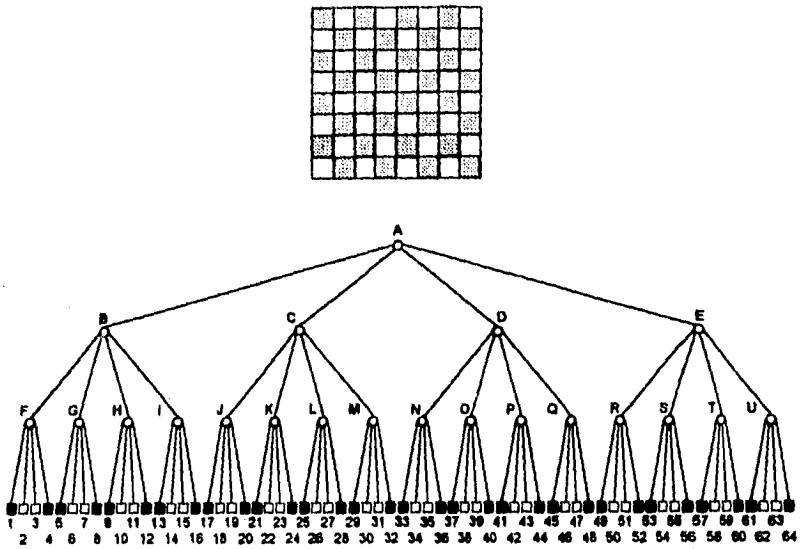


Figura 4.3 Un tablero de ajedrez y su árbol cuádruple.

4.1.3 Variantes de los Árboles Cuádruples.

La definición más general de una imagen es que ésta es una colección o conjunto de pixeles; este conjunto puede representarse de diversas maneras, incluyendo arreglos, listas ligadas y árboles. Además, los subconjuntos de pixeles similares puede agruparse en bloques y este conjunto de bloques puede otra vez representarse utilizando arreglos, listas, árboles, etc. Una vez que se ha escogido una representación, se debe decidir como se ordenan los pixeles, cada uno con respecto a los otros; por ejemplo, la implementación de operaciones para el procesamiento digital de imágenes se facilita mucho cuando se procesa información ordenada.

El arreglo es la representación de imágenes más frecuentemente utilizado; sin embargo, la cantidad de almacenamiento requerida es, con frecuencia, excesiva, razón por lo cual se utiliza mejor una representación *raster*, es decir, la imagen se procesa una hilera a la vez. La representación *raster* puede mejorarse descomponiendo las hileras en bloques unidimensionales de pixeles idénticos (al igual que en las corridas RLE), de esta manera, la imagen se representa como un conjunto de dichas corridas.

Un árbol cuádruple para una región es un miembro de una clase de representaciones caracterizadas por ser una colección de bloques máximos, cada uno de los cuales contiene una región dada y cuya unión es la imagen completa. La más simple de dichas representaciones es la de las corridas descrita anteriormente; en este caso, los bloques están restringidos a ser rectángulos de tamaño $l \times m$.

El árbol cuádruple para una región es una variante de la representación del máximo bloque; requiere que los bloques sean disjuntos y tengan tamaños estándares (los lados deben ser una potencia de 2) y una localización estándar. La motivación para su desarrollo fue el deseo de obtener un modo sistemático para representar partes homogéneas de la imagen; de esta manera, para transformar la imagen en un árbol cuádruple se debe escoger un criterio para decidir si la imagen es homogénea o no.

Uno de tales criterios es que la desviación estándar de los niveles de grises se encuentre debajo de cierto nivel de umbral t; utilizando este criterio, una imagen en un arreglo se subdivide sucesivamente en cuadrantes y subcuadrantes hasta que se obtengan bloques homogéneos; este proceso conduce a una descomposición regular. Si se asocia un nivel promedio de gris a cada nodo no terminal, el árbol cuádruple resultante especifica un aproximación a la imagen donde cada bloque homogéneo es representado por su valor promedio. El caso donde $t = 0$, es decir, el caso en el que el bloque no es homogéneo a menos que su nivel de gris sea constante, es de particular interés puesto que permite una reconstrucción exacta de la imagen a partir de su árbol cuádruple.

Hay que notar que los bloques de un árbol cuádruple no necesariamente corresponden a la región homogéneas máxima de la imagen, más bien pueden existir uniones de bloques que son homogéneas. Para obtener una segmentación de la imagen en regiones homogéneas máximas, se debe permitir la fusión de los bloques adyacentes del mismo color; esto se logra mediante un algoritmo de *partir y fusionar*, sin embargo, la partición resultante ya no se puede representar como un árbol cuádruple, sino como una gráfica de adyacencia. De esta manera, el árbol cuádruple se utiliza como un paso inicial para el proceso de segmentación.

Por ejemplo, la **figura 4.4** demuestra el resultado de aplicar dicho algoritmo; en la **figura 4.4a** la imagen inicial se divide en 16 bloques cuadrados de igual tamaño; después, el paso de fusionar trata de formar bloques más grandes uniendo los grupos de 4 bloques homogéneos (los 4 bloques en los cuadrantes del NO y SE de la **figura 4.4b**); el paso de partir descompone recursivamente los bloques que todavía no son homogéneos (los bloques NE y SO de la **figura 4.4c**) hasta que se satisfaga un criterio particular de homogeneidad; finalmente, el paso de agrupar agrega todos los bloques negros homogéneos tetra-adyacentes en una sola región negra, y de la misma manera, los bloques blancos octa-adyacentes se agregan a la región blanca (**figura 4.4d**).

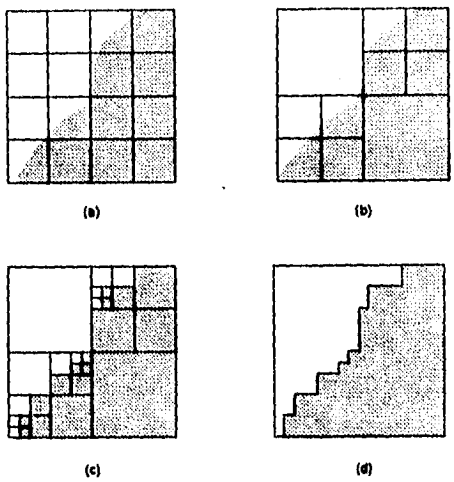


Figura 4.4 **Ejemplo para ilustrar el procedimiento de *partir y fusionar*: (a) Comienzo, (b) Fusionar, (c) Partir, y (d) Agrupar.**

Una alternativa a la representación de la imagen con un árbol cuádruple es utilizar un método de descomposición que no es regular; en este caso se utilizan rectángulos de tamaño arbitrario en vez de cuadrados. Esta alternativa tiene la ventaja de requerir menos espacio; la desventaja es que la determinación de una partición óptima puede ser computacionalmente intensiva.

La descomposición de un árbol cuádruple tiene la propiedad de que en cada subdivisión sucesiva la imagen se subdivide en cuatro partes de igual tamaño; cuando la imagen original es un cuadrado, el resultado es un conjunto de cuadrados, cada uno de los cuales tiene un lado con una longitud que es una potencia de 2. Un árbol binario es una descomposición alternativa definida de manera similar al árbol cuádruple, excepto que en cada paso de subdivisión la imagen se divide en 2 partes iguales; en los pasos nones se particiona la coordenada x y en los pasos pares se particiona la coordenada y.

El árbol binario es equivalente a un árbol cuádruple si se remplazan todos los nodos terminales en los pasos nones por dos hijos con el mismo color; por ejemplo, la **figura 4.5** muestra la representación en árbol binario de la imagen de la **figura 4.1**. Se asume que para la partición en x, el subárbol izquierdo corresponde a la mitad oeste de la imagen y el subárbol derecho a la mitad este; para la partición en y, el subárbol superior corresponde a la mitad norte y para el subárbol inferior corresponde la mitad sur. Nuevamente, los nodos terminales en la **figura 4.5** están etiquetados con números y los nodos no terminales se marcan con letras.

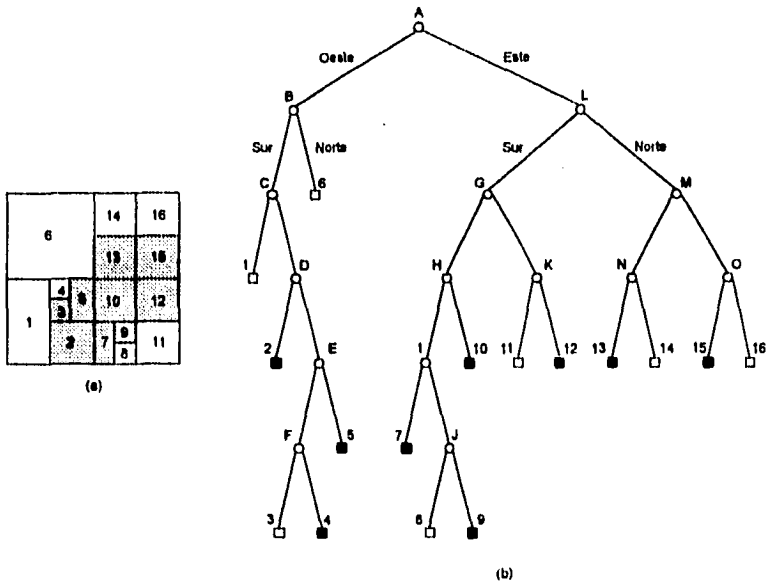


Figura 4.5 **La representación en árbol binario correspondiente a la figura 4.1: (a) Descomposición en bloques, y (b) Árbol binario.**

4.1.4 Historia y Uso de los Árboles Cuádruples.

El origen del principio de la descomposición recursiva, en la cual están basados los árboles cuádruples, es difícil de determinar; a continuación se darán algunas notas indicando el uso de los árboles cuádruples en algunas aplicaciones específicas.

Esta técnica se empezó a utilizar en la implementación de algoritmos de eliminación de líneas y superficies ocultas, en los cuales se descomponía recursivamente una imagen; el área de la imagen era subdividida repetidamente hasta que los rectángulos fueran los suficientemente pequeños para desplegarse. Esta idea también se aplicó en algoritmos de reconocimiento de patrones y en aplicaciones de animación por computadora. Actualmente, los árboles cuádruples se utilizan en la construcción de mallas para análisis de elementos finitos.

Paralelamente al desarrollo de las estructuras de árboles cuádruples, se han realizado trabajos en el campo del reconocimiento de patrones. En estos trabajos se introdujo el concepto de *plano*, el cual es una pequeña área cuyos pixeles representan el promedio de una escala de grises en bloques de 8 x 8 de una imagen más grande; con esta técnica se evita la necesidad de detectar los bordes porque primero se determinan los bordes en el *plano* y después se utilizan estos bordes para buscar selectivamente los bordes en la imagen más grande. La generalización de esta idea motivó el desarrollo de representaciones de imágenes en multirresolución; por ejemplo, el cono de reconocimiento de Uhr, el cono de preprocesamiento de Riseman y Arbib y la pirámide de Tanimoto y Pavlidis[2].

Las descomposiciones similares a los árboles cuádruples son muy utilizadas en los métodos de ordenamiento espacial, cuyo propósito es optimar el almacenamiento y el procesamiento de secuencias de datos bidimensionales, mapeándolos a una dimensión; el mapeo debe preservar la localidad espacial de la imagen bidimensional original en una sola dimensión; el resultado de este mapeo se conoce como *curva llena-espacio* porque pasa a través de cada punto de la imagen. Una discusión de estas curvas ya fue presentada en la sección 3.1.4.1.

4.1.5 Implementación.

La representación más común de un árbol cuádruple es en la forma de un árbol construido con apuntadores y para distinguirlo de otras implementaciones se le conoce como *árbol cuádruple explícito*. En este caso, cada nodo se representa como un registro de tipo *nodo*, el cual contiene 6 campos; los primeros 5 campos contienen apuntadores al nodo padre y a los 4 nodos hijos, los cuales corresponden a los 4 cuadrantes, resultado de la subdivisión; si el nodo es terminal, sus 4 apuntadores a los cuadrantes son nulos. Si P es un apuntador a un nodo e I es un cuadrante, estos campos se especifican como *Padre(P)* e *Hijo(P,I)*, respectivamente.

Se puede determinar el cuadrante específico en el cual se encuentra un nodo P, relativo a su padre, por medio de la función *TipoHijo(P)*, el cual tiene un valor de I si *Hijo(Padre(P), I)* = P. El sexto campo, *TipoNodo*, describe el contenido del bloque de la imagen que representa: negro, blanco o gris. El apuntador de un nodo a su padre no es necesario pero se requiere para moverse fácilmente entre nodos arbitrarios en un cuadrante; este apuntador se utiliza también en diversos algoritmos de operaciones básicas del procesamiento de imágenes.

[2] La codificación de una imagen en multirresolución fue explicada en la sección 2.3.5.1, capítulo 2

4.2 Técnicas de Búsqueda.

En esta sección se describen las técnicas básicas para la manipulación de los árboles cuádruples y algunos conceptos relacionados con dicha manipulación. En la sección se examinan los términos de *adyacencia* y *vecinos*, los cuales son básicos para comprender las operaciones que se pueden efectuar sobre los árboles cuádruples, y la notación utilizada para ello.

4.2.1 Adyacencia y Vecinos en los Árboles Cuádruples.

Cada nodo de un árbol cuádruple corresponde a un bloque de la imagen original; los términos *bloque* y *nodo* se utilizan intercambiablemente; el término utilizado depende del hecho de que la referencia sea a una representación en bloques o en árbol, respectivamente. Cada bloque tiene 4 *lados* o *fronteras*, y 4 vértices conocidos como *esquinas*; a estos lados y vértices se les denota con direcciones, es decir, los 4 lados de un bloque se denominan como N, S, E y O, y los 4 vértices se conocen como NO, NE, SO y SE; la **figura 4.6** ilustra este etiquetamiento.

Dados 2 nodos P y Q que corresponden a bloques que no se sobreponen, y una dirección l, se define el predicado de *adyacencia* tal que la función *Adyacente(P, Q, l)* es verdadero si existen 2 pixeles p y q, contenidos en P y Q, de tal forma que q sea adyacente a un lado de p en la dirección l, o q sea adyacente al vértice l de p; en ambos casos se dice que los nodos P y Q son *vecinos* (más específicamente, *vecinos por lado* o *vecinos por vértice*, respectivamente). Por ejemplo, los nodos 6 y 9 de la **figura 4.7**[1] son vecinos por lado puesto que 6 se encuentra al oeste de 9, mientras que los nodos 8 y 4 son vecinos por vértice puesto que 4 está al NE de 8. Dos bloques pueden ser adyacentes tanto a lo largo de un lado como de un vértice; por ejemplo, el bloque 1 se encuentra tanto al NE de 6 como al N, sin embargo, el bloque 9 está al este de 6 pero no al SE del 6. La relación de adyacencia es válida para nodos terminales y nodos no terminales.

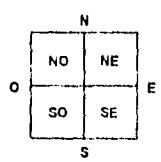


Figura 4.6 La relación entre los 4 cuadrantes de un bloque y sus fronteras.

[1] Se utiliza el mismo ejemplo de la figura 4.1, pero para mayor comodidad se repite aquí.

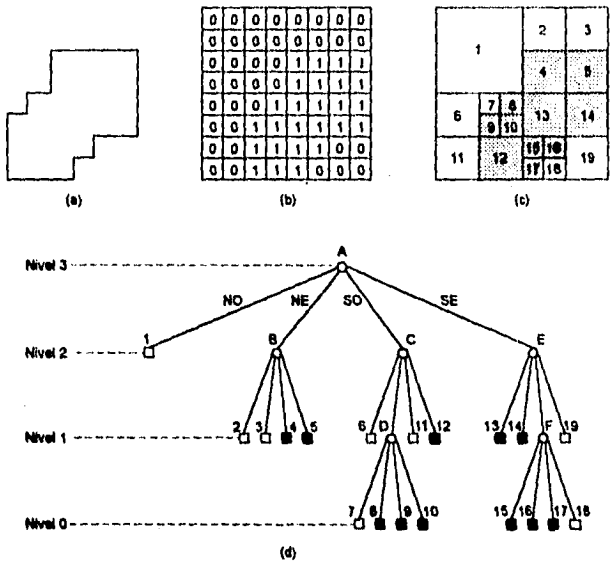


Figura 4.7 **Un ejemplo de (a) Una región, (b) Su arreglo binario, (c) Sus bloques, y (d) el árbol cuádruple correspondiente.**

Desgraciadamente, la relación de vecino no es un una función matemática en sentido estricto; el problema es que, dado un nodo P y una dirección I, existe más de un nodo Q que es adyacente en la dirección I. Por ejemplo, los nodos 8, 10, D y C son vecinos al oeste del nodo 13, similarmente, los nodos 10, D y C son vecinos al NO del nodo 15. Esto significa que para especificar a un vecino es necesario contar con información más precisa acerca de su naturaleza (terminal o no terminal), tamaño y localización.

En particular, es necesario distinguir entre vecinos que son adyacentes a lo largo de un lado entero de un nodo (por ejemplo, el bloque 1 es un vecino al NE del bloque 6) y de aquellos que son adyacentes solo a lo largo de un segmento del bloque (el bloque 7 es un vecino al E del bloque 6). En el primer caso, el interés se centra en determinar un nodo Q tal que su bloque correspondiente es el bloque más pequeño (posiblemente gris) de tamaño mayor o igual que el bloque correspondiente P; en el segundo caso se especifica el vértice de P al cual Q debe ser adyacente.

Abajo se definen las funciones que expresan estas relaciones; la construcción de los nombres sigue la siguiente convención: 'M' para mayor o igual que, 'V' para vértices, 'L' para lado y 'N' para vecino.

1. MLN(P, I) = Q. El nodo *Q* corresponde al bloque más pequeño (puede ser gris) que es adyacente al lado *I* del nodo *P* y su tamaño es mayor o igual que el bloque correspondiente a *P*.
2. VLN(P, I, V) = Q. El nodo *Q* corresponde al bloque más pequeño que es adyacente al lado *I* de la esquina formada por el vértice *V* y el nodo *P*.
3. MVN(P, V) = Q. El nodo *Q* corresponde al bloque más pequeño (puede ser gris) que es diagonalmente opuesto al vértice *V* del nodo *P* y su tamaño es mayor o igual que el bloque correspondiente a *P*.
4. VVN(P, V) = Q. El nodo *Q* corresponde al bloque más pequeño que es diagonalmente opuesto al vértice *V* del nodo *P* y su tamaño es menor que el bloque correspondiente a *P*.

Por ejemplo, *MLN(6, E) = D*, *MLN(6, S) = 11*, *VLN(6, E, SE) = 9*, *MVN(4 , NE) = 3*, *MVN(4, SO) = D*, y *VVN(4, SO) = 8*. De los ejemplos anteriores se puede deducir que *MVN* es el vértice contraparte de *MLN*, al igual que *VVN* para *VLN*; se tienen las siguientes observaciones al respecto: primera, ninguna de las funciones *MLN*, *VLN*, *MVN* o *VVN* define una correspondencia 1 a 1, es decir, un nodo pues ser un vecino en una dirección dada de varios nodos; por ejemplo, *MLN(6, N) = 1*, *MLN(7, N) = 1*, *MLN(8, N) = 1*. Segunda, ninguna de las funciones MLN, VLN, MVN, VNE son necesariamente simétricas; por ejemplo, *MLN(4, O) = 1* pero *MLN(1, E) = B*.

Cuando se utiliza el término *vecino* (*P* es vecino de *Q*), se quiere decir que *P* es un nodo de tamaño mayor o igual a *Q*[4]; por ejemplo, el nodo 10 en la **figura 4.7d** o su bloque equivalente en la **figura 4.7c** tiene como vecinos a los nodos 8, 13, 15, 12, 9 y 7. Un nodo que no es adyacente al borde de la imagen tiene un mínimo de 5 vecinos; esto sucede porque un nodo no puede ser adyacente a dos nodos de tamaño mayor en los lados opuestos (ver **figura 4.8a**) o a dos vértices opuestos en la misma diagonal (**figura 4.8b**).

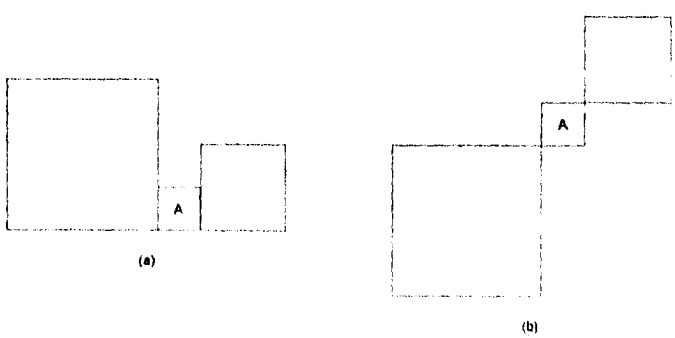


Figura 4.8 Configuraciones imposibles de nodos en un árbol cuádruple.

[4] Esta convención también se aplica a los términos *vecino por lado* y *vecino por vértice*.

4.2.2 Notación y Operaciones.

Existen diferentes maneras de localizar vecinos en un árbol cuádruple implementado con apuntadores; estas técnicas difieren de acuerdo al tipo de información utilizada para llevar a cabo el proceso. El método que se discute aquí es el más general; es independiente tanto de la posición (o coordenadas) como del tamaño del nodo cuyo vecino se está buscando; este método está basado en la localización del ancestro común más cercano y utiliza solamente la estructura del árbol cuádruple, es decir, usa los apuntadores a los 4 hijos de un nodo y el apuntador a su padre.

Para comprender el algoritmo de búsqueda del ancestro común más cercano se requieren algunas definiciones y la explicación de la notación. Se asume que cada nodo del árbol se almacena como un registro del tipo *nodo*, definido en la sección 4.1.5.

Las funciones *Ady*, *Refleja* y *LadoComun* ayudan en la expresión de operaciones con bloques, sus lados y sus vértices. La **figura 4.9** contiene las definiciones de *Ady*, *Refleja* y *LadoComun*; la notación *na* denota un valor no aplicable o indefinido.

La función *Ady(I, O)*, presentada en la **figura 4.9a**, es verdadera si, y solamente si, el cuadrante *O* es adyacente al lado o vértice de otro bloque señalado por la dirección *I*; por ejemplo, *Ady(O, SO)* es verdadero, al igual que *Ady(SO, SO)*. La relación también puede describirse como verdadera si *O* es de tipo *I*, o equivalentemente, que la etiqueta de las *I's* es un subconjunto de la etiqueta de las *O's*.

En la **figura 4.9b** se muestra la función *Refleja(I,O)*, la cual proporciona el tipo de hijo del bloque de igual tamaño (no necesariamente un hermano) que comparte el lado o vértice *I* de un bloque con un tipo de hijo *O*; por ejemplo, *Refleja(N, SO) = NO*, *Refleja(SO, SO) = NE* y *Refleja(NO, SO) = NE*.

LadoComun(I,O), mostrada en la **figura 4.9c** retorna el tipo del lado (etiqueta) de los bloques *O's* que contienen un bloque que es común al cuadrante *O* y a su vecino en la dirección *I*; por ejemplo, *LadoComun(SO, NO) = O*.

Para un árbol cuádruple correspondiente a una imagen de $2^n \times 2^n$, se dice que la raíz se encuentra en el nivel *n* y que un nodo al nivel *i* se encuentra a una distancia *n-i* de la raíz del árbol; en otras palabras, para un nodo de nivel *i*, se deben ascender *n-i* ligas al padre para alcanzar la raíz Un nodo de nivel 0 corresponde a un solo pixel en la imagen, mientras que un nodo de tamaño 2^s se encuentra en el nivel *s* del árbol.

I (Dirección)	O (Cuadrante)			
	NO	NE	SO	SE
N	T	T	F	F
E	F	T	F	T
S	F	F	T	T
O	T	F	T	F
NO	T	F	F	F
NE	F	T	F	F
SO	F	F	T	F
SE	F	F	F	T

(a) Adv(I, O)

I (Dirección)	O (Cuadrante)			
	NO	NE	SO	SE
N	SO	SE	NO	NE
E	NE	NO	SE	SO
S	SO	SE	NO	NE
O	NE	NO	SE	SO
NO	SE	SO	NE	NO
NE	SE	SO	NE	NO
SO	SE	SO	NE	NO
SE	SE	SO	NE	NO

(b) Refleja(I, O)

I (Vértice)	O (Cuadrante)			
	NO	NE	SO	SE
NO	na	N	O	na
NE	N	na	na	E
SO	O	na	na	S
SE	na	E	S	na

(C) LadoComun(I, O)

Figura 4.9 Operaciones básicas en un árbol cuádruple.

Localizar un *vecino por lado* es relativamente simple; primero se asume que se está buscando un vecino de tamaño igual al nodo P en la dirección I; la idea básica es ascender el árbol hasta que se encuentre el ancestro común más cercano y después descender hasta que se encuentre el nodo vecino. Es obvio que siempre se puede ascender hasta la raíz del árbol y después comenzar el descenso; sin embargo, la meta es encontrar el ancestro común más cercano porque esto minimiza el número de nodos visitados. Estos dos pasos se describen a continuación.

1. Localizar el ancestro común más cercano. Éste es el primer nodo ancestro alcanzado por un hijo de tipo O tal que *Adyl(I, O)* sea falsa; en otras palabras, la etiqueta de las *I's* no es un subconjunto de la etiqueta de las *O's*.

2. Re-trazar la ruta que fue seguida para localizar el ancestro común más cercano utilizando la función *Refleja* para hacer un espejo del movimiento sobre los lados compartidos por los nodos vecinos.

Localizar un *vecino por vértice* es considerablemente más complicado. Se asume que se está tratando de buscar un vecino de igual tamaño al nodo P en la dirección I; el paso inicial es localizar el ancestro común más cercano de P y su vecino, para esto se necesita ascender por el árbol y se debe tomar en cuenta la situación en la cual los ancestros de P y su vecino son adyacente en un lado. Sea N que denote el nodo que están siendo examinado en el ascenso, existen 3 casos descritos a continuación.

1. Mientras que *N* sea un hijo de tipo *O* tal que *Ady(I, O)* es verdadera, se continúa el ascenso en el árbol; en otras palabras, la etiqueta de las *I's* es un subconjunto de la etiqueta de las *O's*.

2. Si el padre de *N* y el ancestro del vecino deseado, por decir *A*, son adyacentes por un lado, entonces se calcula *A* por medio del procedimiento de *vecinos por lado*, descritos anteriormente. Esta situación y la dirección exacta de *A* se determinan con la función *LadoComun* aplicada a *I* y al hijo de tipo *N*; una vez que se obtienen *A*, el vecino deseado se localiza aplicando el paso 3.

3. De otra manera, *N* es un hijo de tipo *O* tal que ninguna de las etiquetas de los lados que formar al vértice *I* son un subconjunto de las etiquetas *O's*; su padre, por decir *T*, es el ancestro común más cercano. El vecino deseado se obtiene simplemente retrazando la ruta utilizada para localizar *T*, excepto que ahora se hace con un movimiento diagonalmente opuesto al movimiento sobre los vértices compartidos por los nodos vecinos; este procedimiento es facilitado por la función *Refleja*.

Las técnicas de búsqueda y operaciones en un árbol binario descritas en esta sección son básicas para la conversión de imágenes *raster* a árboles cuádruples y viceversa, tal como se detalla en las siguientes secciones.

4.3 Conversión de Imágenes tipo *Raster* a Árboles Cuádruples.

El árbol cuádruple se ha propuesto como una representación para imágenes binarias debido a que su naturaleza jerárquica facilita el desempeño de un gran número de operaciones; sin embargo, la mayoría de las imágenes se representan tradicionalmente con arreglos binarios, arreglos tipo *raster*, códigos encadenados o vectores, algunos de los cuales se escogen por razones de hardware; por ejemplo, los arreglos tipo *raster* son utilizados especialmente por dispositivos de despliegue que procesan la imagen hilera por hilera, tales como la televisión o las máquinas de facsímiles.

En esta sección se tratará la cuestión específica de convertir imágenes *raster* a árboles cuádruples y viceversa; no se tratan otras representaciones debido a que las imágenes usualmente se almacenan en archivos y en la memoria en forma *raster*, y no en forma de arreglo.

4.3.1 Representaciones *Raster*.

Una imagen usualmente existe como un archivo secuencial donde a la hilera *i* le sigue la hilera *i+1*; a esta representación se le conoce como representación *raster* y consiste de una lista de pixeles ordenados por hileras. Esta representación se utiliza principalmente cuando la imagen es grande, por lo cual no es práctico leerla en la memoria como un arreglo y después convertirlo a un árbol cuádruple; en vez de eso, la imagen se lee hilera por hilera al tiempo que se va construyendo el árbol.

Por otro lado, una imagen usualmente se despliega en un dispositivo *raster* y por lo tanto también existe la necesidad de convertir un árbol cuádruple a un representación *raster*. En las dos secciones siguientes se explica más detalladamente como se construye un árbol cuádruple a partir de una imagen *raster* y viceversa.

4.3.2 Construcción de un Árbol Cuádruple a partir de una Imagen *raster*.

La clave para el algoritmo de conversión de una imagen *raster* a un árbol cuádruple es que en cualquier instante de tiempo, es decir, después de que cada pixel en una hilera se ha procesado, existe un árbol cuádruple válido que contiene a todos los pixeles, tanto los que ya se procesaron como los que están todavía sin procesar; éstos últimos pixeles son presumiblemente blancos (aunque en realidad se conocen como *incoloros*). De esta manera, conforme se construye el árbol cuádruple, los nodos se fusionan para generar bloques máximos.

El procedimiento principal para la construcción del árbol cuádruple a partir de la imagen *raster* se invoca con un apuntador a la primera hilera y el ancho de la misma y se asume que la imagen tiene un número par de pixeles en cada hilera; el algoritmo trabaja bien para cualquier imagen que tenga al menos dos hileras con dos pixeles por hilera; se considera que la imagen contiene también un número par de hileras. Si la imagen contiene un número non de hileras, se agrega una hilera extra para lograr el número par; de esta manera, el árbol cuádruple resultante corresponde a una imagen cuadrada cuya longitud de sus lados es una potencia de 2; todos los pixeles que se añaden para que se cumpla esta condición se consideran blancos.

La imagen se representa como una lista de registros de tipo *ListaHilera*, cada uno de los cuales tiene dos campos, *Hilera* y *Siguiente*; *Hilera* corresponde a un arreglo de registros de tipo *Pixel*, y *Siguiente* es un apuntador al siguiente elemento de la lista; el elemento *Pixel* tiene un campo denotado *Color* para almacenar el color del pixel (blanco o negro).

Como ejemplo de la aplicación del algoritmo, considérese la región presentada en la **figura 4.10a**, su descomposición en hileras está dada en la **figura 4.10b**, los bloques correspondientes a la descomposición se muestran en la **figura 4.10c**, y en la **figura 4.10d** se presenta el árbol cuádruple correspondiente; este árbol es análogo al de la figura 4.1 excepto que los pixeles han sido numerados en el orden en el cual el algoritmo los procesa y todos los nodos que se han fusionado se etiquetan con las letras A-K; el orden alfabético corresponde al orden en el cual se unieron los nodos cuando fueron creados. La **figura 4.11** muestra los pasos en la construcción del árbol cuádruple correspondiente a los 4 primeros pixeles de la primera hilera (pixeles 1, 2, 3 y 4); la **figura 4.12** y la **figura 4.13** presentan el árbol resultante después de que se han procesado la primera y segunda hileras, respectivamente.

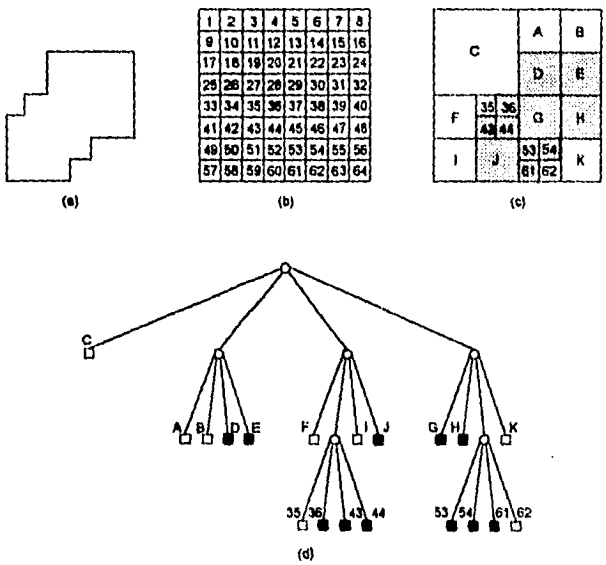


Figura 4.10 **Un ejemplo de (a) Una región, (b) Su representación en arreglo binario en la cual los pixeles están etiquetados de acuerdo al orden en que se visitan durante el proceso de construcción del árbol cuádruple a partir de la representación *raster*, (c) Su máxima descomposición en bloques, y (d) El árbol cuádruple correspondiente.**

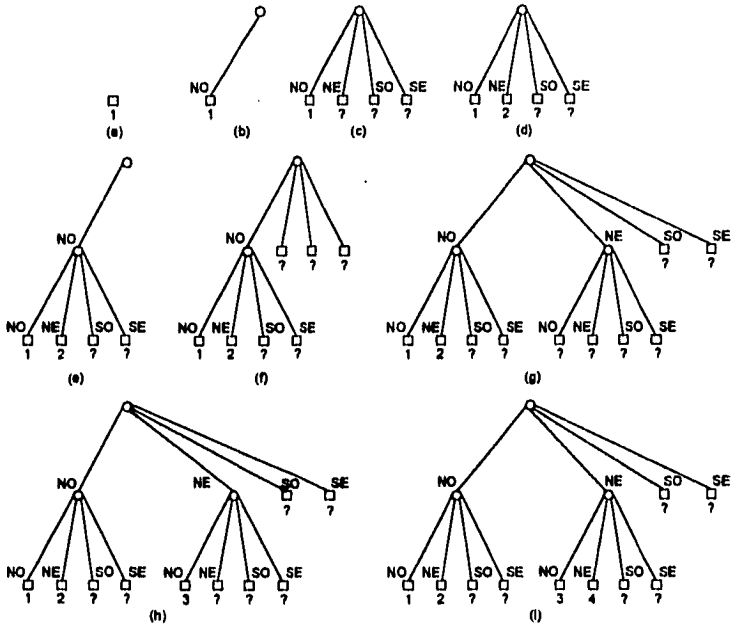


Figura 4.11 **Árboles cuádruples intermedios en el proceso de obtención de un árbol cuádruple para la primera mitad de la primera hilera de la figura 4.10**

En el algoritmo se utilizan dos procedimientos auxiliares para añadir hileras nones e hileras pares de la imagen al árbol; esto se logra localizando al nodo vecino y creando los nodos necesarios para los pixeles que no tienen un nodo en el árbol cuádruple; existe un procedimiento de fusión, responsable de remplazar cualquier nodo gris que tenga 4 hijos del mismo color por un nodo de ese color, y eliminar los hijos.

La cantidad de trabajo requerida depende del tipo de hilera procesada; las hileras nones requieren poco trabajo porque en ellas no se lleva a cabo ningún proceso de fusión; en cambio, en las hileras pares se tiene más dificultad debido a dicha fusión. Para las hileras nones, el árbol se construye procesando la hilera de izquierda a derecha para cada pixel; conforme el árbol se va construyendo se van añadiendo nodos no terminales. Puesto que lo que se desea es tener un árbol válido después de procesar cada pixel, siempre que se añada un nodo no terminal se deben añadir los otros 3 nodos, los cuales se consideran como incoloros o blancos hasta que todos su pixeles correspondientes se han procesado (en las figuras, estos nodos se etiquetan con el símbolo '?' indicando que todavía no se les ha asignado su color final).

Para buscar el vecino del pixel procesado, se localiza el ancestro común más cercano y una vez que se encuentra, se desciende a través de una ruta que es un reflejo sobre el eje formado por la frontera común entre los dos pixeles; cuando el ancestro común más cercano no existe, es decir, el pixel que está siendo procesado se encuentra a la extrema derecha de la hilera, se añade un nodo no terminal con sus tres hijos restantes incoloros (ver **figura 4.11c** y **figura 4.11f**).

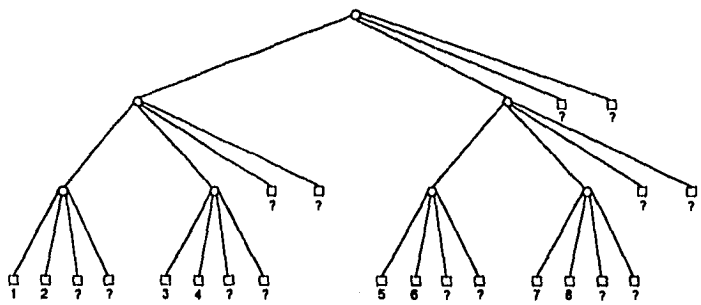


Figura 4.12 **El árbol cuádruple después de procesar la primera hilera de la figura 4.10b.**

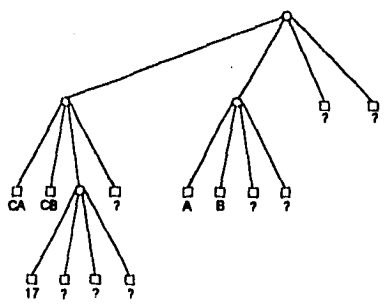


Figura 4.13 **El árbol cuádruple después de procesar la segunda hilera de la figura 4.10b.**

Una vez que se han añadido al árbol el ancestro común más cercano y sus 3 hijos restantes, se desciende a lo largo de la ruta reflejada sobre el eje formado por la frontera común entre el pixel y su vecino buscado; durante este descenso, los nodos blancos se convierten en nodos grises y se añaden 4 hijos incoloros (ver **figura 4.11g**); como paso final, el nodo terminal se colorea apropiadamente (**figura 4.11d** y **figura 4.11h**). En este ejemplo, los subárboles mostrados en la **figura 4.11** son piezas del árbol durante el proceso de construcción para los pixeles 1, 2 ,3 y 4; se puede notar que en cualquir momento existe un árbol cuádruple válido, aunque todavía no se hayan procesado los demás pixeles..

En cuanto a la fusión de bloques, las hileras pares requieren más procesamiento puesto que ésta toma lugar en ellas; en particular, se debe hacer una verificación para una posible fusión en cada posición vertical par de la hilera. Una vez que ha ocurrido una fusión, se debe verificar si es posible otra fusión; específicamente hablando, para un pixel en la posición $(a2^i, b2^j)$, donde $a \bmod 2 = b \bmod 2 = 1$ e $i, j \geq 1$, es posible un máximo de $k = min(i, j)$ fusiones posibles; por ejemplo, en el pixel 28 de la **figura 4.10**, con posición (4, 4), se tiene un máximo de 2 fusiones, y es de esta manera como se ha obtenido el bloque *C* de la **figura 4.10c**.

Para efectuar la fusión se necesita almacenamiento extra para guardar la posición del árbol en la cual se añadirá el siguiente pixel, también se requiere guardar el apuntador a la siguiente hilera. Antes de intentar la fusión, el nodo correspondiente al siguiente pixel se añade al árbol, por ejemplo, el nodo 11 se añade al árbol de la **figura 4.14** antes de fusionar los nodos 1,2, 9 y 10 de la **figura 4.10b**. Se procede de manera similar con el procesamiento de cada hilera par añadiendo al árbol el nodo correspondiente al primer pixel en la siguiente hilera. por ejemplo, el nodo 17 en el árbol de la **figura 4.14** se añade antes de procesar la hilera 2 de la **figura 4.10b**.

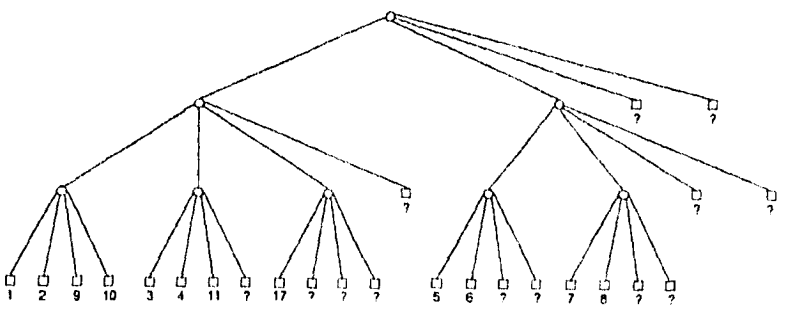


Figura 4.14 El árbol cuádruple antes de fusionar los nodos 1,2, 9 y 10 de la figura 4.10a

4.3.3 Construcción de una Imagen *raster* a Partir de un Árbol Cuádruple.

El método más obvio para construir una representación *raster* a partir de un árbol cuádruple es generar un arreglo que corresponda al árbol; sin embargo, este método requiere grandes cantidades de memoria, por lo cual no es práctico. Existen otros métodos para esta conversión, la mayoría de estos algoritmos recorren el árbol transversalmente por hileras de izquierda a derecha y visitan cada nodo del árbol cada vez que una hilera lo intersecta; por ejemplo, para los pixeles del 1 al 8 en la primera hilera de la **figura 4.15b**, el nodo *A* de la **figura 4.15d** se visita primero, seguido por los nodos *B*, *C*, *D* y *E*. Cada nodo, blanco o negro, en el nivel *j* del árbol se visita 2^j veces; cada visita da como salida una corrida de pixeles de longitud 2^j.

En los siguientes párrafos se discuten dos algoritmos: uno es un algoritmo de *arriba-abajo* y el otro es de *abajo-arriba*; el algoritmo *arriba-abajo* comienza en el nodo raíz cada vez que se visita a un nodo que se intersecta con la hilera; en contraste, el algoritmo *abajo-arriba* visita los bloques adyacentes en una hilera utilizando las técnicas de búsqueda de vecinos (descritas en la sección 4.2).

Ambos algoritmos localizan al bloque que contiene al segmento de la hilera procesada; los parámetros que se utilizan son las coordenadas de la esquina superior izquierda del pixel más a la izquierda del segmento que se desplegará, y las coordenadas de la esquina inferior derecha del bloque (puede ser un nodo no terminal) que contiene el segmento. Este bloque se particiona repetidamente hasta que se encuentra el bloque más pequeño correspondiente al nodo terminal; después, se identifica el cuadrante del nodo gris que contiene un bloque correspondiente al segmento que se desplegará. Por ejemplo, para localizar el bloque que contiene el segmento que comienza en la hilera 0 y columna 0 de la **figura 4.15b**, se particiona la imagen sucesivamente en bloques que tienen su esquina inferior derecha en (8, 8), (4, 4) y (2, 2), respectivamente; el resultado es el bloque *A*. Una vez que se ha localizado el inicio del bloque, se despliegan las corridas de pixeles correspondientes al color del bloque, para la hilera procesada; por ejemplo, para el bloque *A* se tiene una corrida de 2 pixeles en la primera hilera (pixeles 1 y 2).

Como un ejemplo de la aplicación del algoritmo *arriba-abajo*, considérese la imagen y el árbol cuádruple de la **figura 4.15**; los nodos R_i corresponden a nodos no terminales; los nodos terminales en la **figura 4.15** han sido etiquetados en el orden en el cual han sido visitados la primera vez. Para esta imagen, el resultado del algoritmo es la secuencia de cadenas *b332*, *b242*, *b242*, *b152*, *b224*, *b224*, *b8*, *b8*; existe una cadena para cada hilera de la imagen y las letras *b* y *n* se utilizan para indicar el color (blanco o negro, respectivamente) de las corridas, las corridas subsecuentes se obtienen alternando el color; por ejemplo, la cadena *b332* indica una corrida blanca de 3 pixeles, seguida de una negra de 3 pixeles y finalmente una blanca de 2 pixeles.

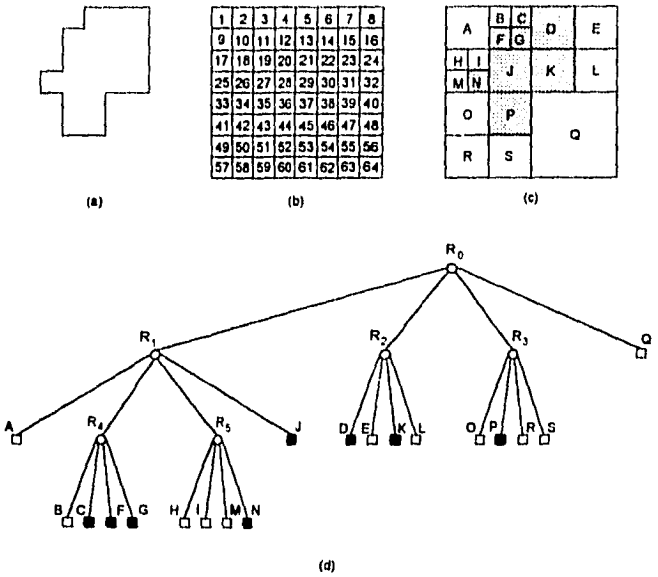


Figura 4.15 **Un ejemplo de (a) Una región, (b) Su representación en arreglo binario en la cual los pixeles están etiquetados de acuerdo al orden en que se visitan durante el proceso de construcción de la imagen *raster* a partir del árbol cuádruple, (c) Su máxima descomposición en bloques, y (d) El árbol cuádruple correspondiente.**

Cuando se está procesando la primera hilera, se comienza con el nodo R_0 y sucesivamente se visitan los nodos R_1 y A; R_4 y B; R_1, R_4 y C; R_2 y D; R_2 y E. Para la segunda hilera se visitan los nodos R_1 y A; R_1, R_4 y F; R_1, R_4 y G; R_2 y D; R_2 y E; y así, para los otras hileras.

El procedimiento *arriba-abajo* visita cada segmento de la hilera comenzado repetidamente la búsqueda desde la raíz del árbol; el algoritmo *abajo-arriba* previene esta búsqueda utilizando la estructura de árbol para localizar bloques adyacentes sucesivos. Por ejemplo, en la **figura 4.15**, una vez que la corrida correspondiente al bloque B en la primera hilera ha sido procesada, el siguiente nodo a visitar es C; este nodo puede localizarse utilizando el apuntador entre los nodos R_4 y C, esto contrasta con la ruta trazada por las ligas entre R_0, R_1 y C cuando se utiliza el algoritmo *arriba-abajo*. Los nodos adyacentes se localizan con las técnicas de búsqueda de vecinos, detallada en la sección 4.2; una vez que se ha localizado el vecino, el algoritmo *abajo-arriba* procede al igual que el algoritmo *arriba-abajo*.

La aplicación del algoritmo *abajo-arriba* a la imagen de la **figura 4.15** da como resultado la misma cadena de salida; sin embargo, el orden en el que se vistan los nodos es diferente; cuando se está procesando la primera hilera, se comienza con R_0 y sucesivamente se visitan R_1 y A; R_1, R_4 y B; R_4 y C; R_4, R_1, R_0, R_2 y D; R_2 y E; R_2; R_0. El último par de nodos indica que no existen vecinos, es decir, se ha llegado al final de la hilera. Para la segunda hilera se visitan R_1 y A; R_1, R_4 y F; R_4 y G; R_4, R_1, R_0, R_2 y D; R_2 y E; R_2; R_0; y así, para las otras hileras.

4.3.4 Los Árboles Cuádruples y la Compresión de Imágenes.

Los árboles cuádruples tienen la propiedad interesante de que conducen a cierta compresión de la imagen, en el sentido de que reducen la cantidad de datos requerida para codificar la imagen (y transmitirla). Como ya se ha visto en las secciones anteriores, la aproximación de la imagen con el árbol cuádruple consiste de una combinación de nodos blancos, negros y grises, cada uno de ellos representando bloques de la imagen; cuando la cantidad de nodos es tal que ocupan menos espacio que la imagen en sí, se dice que se ha logrado comprimir ésta. Sin embargo, la estructura de árbol cuádruple no fue diseñada expresamente para la compresión de imágenes, sino más bien como una herramienta auxiliar en el procesamiento digital de la misma, reconocimiento de patrones, etc.; si se desea comprimir una imagen existen mejores técnicas, las cuales fueron descritas en el capítulo 2.

Como se mencionó al inicio del capítulo, la principal característica de interés de los árboles cuádruples es su capacidad para dividir la imagen en regiones homogéneas; de esta manera, se pueden clasificar las regiones que tengan afinidad entre sí, siendo posible una mejor aplicación de los mapas IFS para la compresión de imágenes con técnicas fractales, la cual se describirá en el capítulo 6.

Parte II

Aplicaciones

Capítulo 5

Formatos Gráficos Comerciales

Capítulo 5. Formatos Gráficos Comerciales.

Los formatos gráficos comerciales son una estandarización para el manejo de imágenes en diversos paquetes de computadora. La mayoría de ellos manejan algún tipo de compresión con objeto de minimizar los requerimientos de almacenamiento de las imágenes, y de ahí el hecho de hablar de ellos en este capítulo. Otro motivo para describir los formatos gráficos comerciales más comunes en el mercado es que en la presente tesis se convertirán estos formatos a un nuevo tipo de compresión de imágenes: compresión con funciones iteradas, las cuales ya se presentaron en el capítulo 3 y se hablará de ellas más detalladamente en el capítulo 6.

En las siguientes secciones se presentarán los formatos típicos manejados por la mayoría de los paquetes: Mapas de bits BMP, Compuserve GIF, Archivos PCX, Targa TGA y Archivos TIFF. De cada uno de ellos se hablará de su estructura de archivo y el tipo de compresión, así también como de sus usos, ventajas y desventajas.

5.1 Mapas de Bits BMP.

Los datos de archivos BMP pueden contener imágenes de 1, 4, 8 ó 24 bits de color; si se desea almacenar una imagen con un número intermedio de bits por color en un archivo BMP es necesario redondear al número de bits más próximo y desperdiciar los bits intermedios. Los formatos BMP de 4 y 8 bits incluyen una paleta de colores, mientras que los archivos con 24 bits no lo hacen debido a que cada pixel puede especificar su propio color RGB.

Las imágenes en los archivos BMP se almacenan de abajo hacia arriba, es decir, la primera línea leída del archivo es la línea del fondo de la imagen. Las líneas en un archivo BMP se encuentran siempre alineadas a una palabra par al final y es importante tener esto en mente cuando se está descodificando una imagen con líneas de longitud non. La fórmula para calcular la longitud en bytes de una línea es *bits por color* * *div((ancho imagen + 7)/8)*, donde *div* es la función que regresa la parte entera de la división; por ejemplo, si el ancho de una imagen de 8 bits por pixel es 291 pixeles, la longitud de la línea almacenada en el archivo BMP es 296.

Las imágenes monocromáticas se almacenan con 8 pixeles por byte, en el cual cada uno de los bits indican los colores negro y blanco, según estén prendidos o apagados. Las imágenes de 4 bits por color se almacenan como una pila de *nibbles*[1] debido a que solamente se necesitan 4 bits como índice para la tabla de 16 colores, de esta manera, un byte define el color de 2 pixeles. Las imágenes con 8 bits por color se almacenan con un byte para cada pixel, cada uno de los bytes es el índice a la tabla de 256 colores. Finalmente, las imágenes de 24 bits por color se almacenan con 3 bytes por pixel para indicar el color RGB correspondiente (un byte para el rojo, uno para el verde y uno para el azul).

Si se trabaja únicamente con archivos BMP creados por aplicaciones Windows, siempre se encuentra que la estructura interna de estos archivos es predecible; sin embargo, los archivos BMP también pueden provenir de otras fuentes, la más común de ellas es el software *Presentation Manager* de OS/2. Estas diferencias se discuten en la siguiente sección.

5.1.1 Estructura de Archivo.

Un archivo BMP siempre comienza con un encabezado. Desafortunadamente, existen dos posibles encabezados, dependiendo de la fuente del archivo BMP; el más común es una estructura BITMAPINFOHEADER, la cual es utilizada en Windows. A continuación se presenta la estructura del encabezado en lenguaje C, modificada al español.

```
typedef struct tagENCABEZADOBMP {
      char Identificador[2];
      long TamanoArchivo;
      int  Reservado[2];
      long TamanoEncabezado;
      long TamanoInfo;
      long Ancho;
      long Altura;
      int  Planos;
      int  Bits;
      long Compresion;
      long TamanoImagen;
      long XPixelsPorMetro;
      long YPixelsPorMetro;
      long ColorUsado;
      long ColorImportante;
} ENCABEZADOBMP;
```

[1] El *nibble* es un conjunto de 4 bits, de esta manera, 2 *nibbles* forman un byte.

Alternativamente, un archivo BMP puede comenzar con una estructura BITMAPCOREHEADER, la cual es escrita por las aplicaciones OS/2. La versión en lenguaje C de esta estructura es:

```
typedef struct tagENCABEZADOBMPCORE {
      char Identificador[2];
      long TamanoArchivo;
      int  Reservado[2];
      long TamanoEncabezado;
      long TamanoInfo;
      int  Ancho;
      int  Altura;
      int  Planos;
      int  Bits;
} ENCABEZADOBMPCORE;
```

Ante la existencia de 2 encabezados diferentes podría surgir la pregunta de como diferenciar uno de otro cuando se abre un archivo BMP; la respuesta no está bien definida en ningún documento de Windows, sin embargo, una vez que se conozcan los campos de estas estructuras se puede idear un truco para diferenciarlas.

El campo *Identificador* siempre contiene la cadena "BM", identificando al archivo como un genuino archivo BMP; por supuesto, podría existir un archivo de texto que comenzara casualmente con este nombre y el lector de archivos BMP identificaría erróneamente al archivo, sin embargo, de cualquier manera se presentarían errores al tratar de leer la siguiente información.

El valor de *TamanoArchivo* es el tamaño total del archivo en bytes, algo que no es muy útil durante el proceso de descodificación porque el elemento *TamanoEncabezado* especifica que tan grande es el encabezado y además especifica el desplazamiento en el archivo a partir del cual comienzan los datos. La estructura en la cual se encuentra el elemento *TamanoArchivo* es un objeto separado en Windows llamado BITMAPFILEHEADER.

El elemento *TamanoInfo* del encabezado del archivo BMP es realmente la clave para definir que tipo de encabezado está siendo descodificado. Representa el número de bytes en el resto del encabezado, el cual es 40 si es un archivo BMP de Windows y 12 si es un archivo de OS/2. Una aplicación que pueda leer transparentemente los dos tipos de archivos debe leer primero el comienzo del archivo con una estructura tipo ENCABEZADOBMP y verificar el valor de *TamanoInfo*, si no es 40 se debe forzar al tipo ENCABEZADOBMPCORE y leer los valores en el encabezado de esta forma. En ambos casos, de cualquier manera se tiene que hacer una búsqueda para encontrar la paleta y la imagen.
Los elementos de *Ancho* y *Altura* definen la dimensión de la imagen en pixeles. El elemento *Planos* es siempre 1. El elemento *Bits* define el número de bits por color de la imagen, 1, 4, 8 ó 24.

El campo *Compresion* especifica el tipo de compresión utilizada en el archivo BMP; siempre es cero debido a que los archivos BMP no están comprimidos. La especificación del formato BMP define un tipo de compresión RLE cruda, pero es parte de otra especificación alterna (Formato Windows RLE) y no se discute en esta tesis. El tamaño de la propia imagen se define con *TamanoImagen*.

Los parámetros XPixelsPorMetro y YPixelsPorMetro especifican la resolución horizontal y vertical de un mapa de bits en pixeles por metro; esto se hace para permitir a una aplicación escoger uno entre varios mapas de bits de diferente resolución. El valor *ColorUsado* define el número de colores utilizados en la paleta para la imagen que está siendo descodificada; no es raro tener, por ejemplo, una paleta de 256 colores de los cuales solamente se utilizan unos pocos en la imagen. Si el valor es cero se están utilizando todas las entradas de la paleta. El *ColorImportante* le indica a Windows cuantos colores en la paleta de la imagen son importantes, lo cual puede ser muy útil si hay necesidad de hacer un remapeo de colores. Los colores importantes son aquellos que se utilizan más en una imagen y que pueden causar un cambio de color de la imagen al remapearlos.

Una vez que se ha descodificado el encabezado del archivo BMP, se debe leer la información de la paleta, si es que existe alguna, y después localizar los datos propios de la imagen. La estructura de la paleta varía de acuerdo al tipo de encabezado leído; contiene 4 bytes por entrada para mapas de bits de Windows y 3 bytes por entrada para mapas de bits de OS/2. Los datos de la paleta en el formato BMP también se encuentran definidos peculiarmente pues aparecen en el orden BGR (Blue, Green, Red) Azul, Verde y Rojo, es decir, al contrario del estándar RGB.

Se puede localizar la información de la imagen en un archivo BMP buscando el punto definido por el elemento *TamanoEncabezado*. Esto es mejor que solamente leer el encabezado y la paleta asumiendo que lo que sigue es la imagen, porque, al menos en teoría, es legal crear archivo BMP con datos adicionales entre el encabezado o la imagen, o con una paleta más pequeña.

5.1.2 Uso de los Archivos BMP.

El formato BMP para MS-Windows tiene diversas características singulares; específicamente, almacena imágenes de hasta 24 bits de color, y lo hace en un formato que es idéntico a los mapas de bits *independientes del dispositivo* que utiliza Windows; además, los archivos BMP se almacenan sin compresión, lo cual significa que pueden cargarse dentro de una aplicación Windows y desplegarse con un mínimo de tiempo y poca manipulación de datos.

La principal ventaja del formato BMP, que es la de no estar comprimido y ser fácil y rápido de leer, es también una de sus principales desventajas. Sin considerar los beneficios de la compresión de imágenes, los archivos BMP son bastante grandes y probablemente la mayoría de los usuarios procuran no tener demasiadas imágenes en este formato.

Aunque los archivos BMP se almacenan sin compresión y esto los hace ser más rápidos cuando se cargan en la mayoria de las aplicaciones, este podria no ser el caso cuando los archivos se guardan en medios relativamente lentos tales como discos flexibles y CD ROM's. Por ejemplo, si se leen los archivos de un CD ROM, el tiempo que toma efectuar la lectura es ligeramente menor que el tiempo de búsqueda de la unidad misma. De esta manera, leer relativamente pocos bytes de un dispositivo lento y después descomprimirlos lleva usualmente menos tiempo que leer grandes cantidades de datos sin comprimir, aunque el primer esquema necesite de una función para desempacar la información.

Por otro lado, mientras las imágenes sean más complejas, la habilidad de algunos algoritmos de compresión de imágenes disminuye. Por ejemplo, si se trabaja con imágenes digitalizadas o imágenes a color con *dither* es probable que los algoritmos de alta compresión, tales como el LZW que se aplica a los archivos GIF, no disminuyan en forma significativa el tamaño de los archivos, aunque el tiempo que se tardan procesando la imagen sea largo. Esto pasa casi siempre cuando se trata de comprimir imágenes en *color real* que han sido digitalizadas, pues los intentos de comprimir a éstas invariablemente hacen al archivo más grande; en este caso es probable que los archivos BMP no desperdicien tanto espacio como podría parecer en un principio.

5.2 Compuserve GIF.

De todos los formatos gráficos de imágenes, el formato GIF es sin duda el más difícil de comprender por razones que se discutirán más adelante. El acrónimo de GIF significa *Graphic Interchange Format* o formato para intercambio de gráficas; el formato GIF fue creado por CompuServe, la base de datos telefónica más grande del mundo, como un medio para intercambiar imágenes por módem. Desde la creación del formato GIF decenas de terabytes de imágenes digitalizadas han sido convertidas a este formato; el estándar GIF no está atado a ningún fabricante de hardware y por lo tanto puede ser utilizado por cualquiera; el software para ver imágenes GIF está disponible para casi todos los tipos de computadora, desde una PC hasta estaciones de trabajo Sun y Apolo.

Adicionalmente a esto, el formato GIF es el único formato gráfico que utiliza compresión LZW como medio principal para su almacenamiento, es decir, que las imágenes almacenadas en archivos GIF requiren menos espacio que cualquiera de los otros formatos.

5.2.1 Estructura de Archivo.

El formato GIF ha sufrido solamente una revisión desde su creación; las especificaciones originales fueron liberadas en 1987 y se conocen como GIF87a. La versión reciente es la liberada en 1989 conocida como GIF89a; esta versión soporta a la versión anterior pero posee nuevas definiciones del formato.

Un archivo GIF puede almacenar imágenes de hasta 256 colores y soporta imágenes de cualquier dimensión que se desee, aunque por lo general siempre se prefieren tamaños que concuerden con la resolución de los monitores VGA y SuperVGA. En muchos de los casos es común encontrar que las imágenes más pequeñas son rellenadas para alcanzar dichas dimensiones.

Los tamaños más comunes de imágenes son:

- 320 x 200 pixeles.
- 640 x 400 pixeles.
- 640 x 480 pixeles.
- 800 x 600 pixeles.
- 1024 x 768 pixeles.

Sin embargo, realmente estos valores no significan nada en una aplicación Windows puesto que una ventana puede ser del tamaño que se quiera.

Además de almacenar imágenes, los archivos GIF pueden guardar información acerca de las imágenes e instrucciones de como deben desplegarse, pues también pueden incluirse varias de ellas en un solo archivo.

El elemento más notable del estándar GIF es la manera con la cual se comprime la imagen utilizando un algoritmo de compresión LZW; esto hace a los archivos GIF más difíciles de tratar con ellos en muchos aspectos, sin embargo, los archivos GIF tienen una mejor razón de compresión que todos los otros formatos de archivos. El problema de utilizar compresión LZW es que se requiere bastante código, bastante memoria, y es tradicionalmente más lenta que la compresión sencilla con RLE.

Un archivo GIF simple, con una sola imagen y sin accesorios adicionales, consiste de un encabezado, un bloque de imagen y un bloque terminador. La noción de los bloques es bastante importante en los archivos GIF; a diferencia del formato BMP discutido anteriormente, los bloques en un archivo GIF no están restringidos a existir en un lugar fijo relativo al inicio del archivo, sino relativo a otro bloque.

La estructura de un encabezado GIF, encontrado siempre al inicio del archivo, es:

```
typedef struct tagENCABEZADOGIF {
      char          Identificador[6];
      unsigned int  AnchoPantalla;
      unsigned int  AlturaPantalla;
      unsigned char Banderas;
      unsigned char Fondo;
      unsigned char Aspecto;
} ENCABEZADOGIF;
```

El elemento *Identificador* del encabezado GIF contiene una de las dos cadenas siguientes "GIF87a" o "GIF89a", dependiendo del número de versión del archivo. Una rutina lectora de archivos GIF debe utilizar solamente los 3 primeros bytes para determinar si un archivo está en formato GIF; en teoría, una rutina bien escrita debe ser capaz de tratar con bloques básicos de imagen en un archivo GIF87a al igual que en el GIF89a, puesto que las nuevas características en la revisión de 1989 son para incrementar el número de bloques de extensión. En realidad, la mayoría de los archivos GIF todavía utilizan la especificación GIF87a debido a que no se ha encontrado un uso extensivo de estos bloques (la mayoría de los archivos GIF almacenan solamente una imagen).

Los elementos *AnchoPantalla* y *AlturaPantalla* definen las dimensiones de la pantalla que fueron utilizadas originalmente al crear el archivo; estas dimensiones no son necesariamente las mismas que el tamaño de la imagen. En un archivo que tiene múltiples imágenes, esta información podría utilizarse para escoger el modo gráfico más óptimo para ver todas las imágenes; en la práctica no tiene mucho significado bajo el ambiente Windows. Sin embargo, se debe notar que la mayoría de los lectores GIF verifican estos valores, aún cuando no se utilicen. Los valores más obvios para estos campos son las dimensiones de la primera imagen en el archivo GIF.

El elemento *Banderas* contiene varios valores útiles. Para comenzar, se debe probar *Banderas & 0x80*, si es verdadero, significa que después del encabezado viene un mapa de colores; el número de colores dentro de la paleta se calcula como *(Banderas & 0x0007) + 1* y existen 3 bytes para cada color. Si *Banderas & 0x08* es verdadero, la paleta se colores se encuentra ordenada con los colores más importantes al principio; esta bandera es raramente usada. Finalmente, *(1 << ((Banderas >> 4) + 1)* representa el número de bits por color en la imagen original de la cual se obtuvo el archivo GIF; este número podría no ser el mismo que el número de bits por color en la imagen almacenada; nuevamente se debe notar que este campo casi no se utiliza. Los últimos dos bits son específicos a la revisión GIF89a; estos bits son puestos a cero en los archivos GIF87a.

El elemento *Fondo* contiene el número del color para el fondo de las imágenes que vienen en el archivo GIF. En una tarjeta VGA corriendo bajo DOS este valor se utiliza para poner un fondo y borde correctos a las imágenes que están siendo desplegadas, en vez de asignarles un valor arbitrario; bajo ambiente Windows esto no es necesario aunque dicho valor se puede especificar como el color del fondo de la ventana. El campo *Aspecto* es único a los archivos GIF89a; siempre es puesto a cero en los archivos GIF87a. Este valor especifica la razón de aspecto de los pixeles en la imagen; si el campo no es cero, la razón de aspecto de los pixeles es *(Aspecto + 15)/64*. Nuevamente este elemento no tiene mucho significado bajo Windows porque no existe manera de cambiar la razón de aspecto de los pixeles de la pantalla. Se debe notar que debido a este campo está restringido a ser cero en archivos GIF87a, muchos lectores antiguos decidirán que tienen un archivo corrupto cuando encuentran un archivo GIF89a que utilice dicho campo.

Si el campo *Banderas* de un encabezado GIF indica que se encuentra disponible un mapa de colores, éste viene a continuación del encabezado. Este mapa de color es conocido como mapa global de color y debe ser considerado como un conjunto de colores por omisión que puede ser sobreescrito por las imágenes específicas del archivo GIF que está siendo leído. En la práctica, la mayoría de los archivos GIF (con una sola imagen y sin extensiones) tienen un mapa global de color como su única definición de colores. Debido a que el número máximo de colores en el formato GIF es 256 y el número de bytes de un color RGB es 3, la máxima memoria requerida para contener la información de la paleta es 768 bytes.

El siguiente byte que se lee en un archivo GIF, después de que se ha leído el encabezado y el mapa opcional de colores, especifica el tipo de bloque que sigue a continuación. Un bloque puede ser una de los tipos indicados por el byte leído; si es una coma el bloque contiene una imagen, si es un signo de exclamación el bloque es una extensión, y si es un punto y coma el bloque es un terminador. En un archivo GIF sencillo siempre se encuentra un bloque de imagen seguido de un terminador.

Un bloque de imagen consiste de un encabezado secundario para la imagen, seguido de los datos propios de la imagen. Esta es la definición del bloque:

```
typedef struct tagBLOQUEIMAGENGIF {
      unsigned int  Izquierda;
      unsigned int  Superior;
      unsigned int  Ancho;
      unsigned int  Altura;
      unsigned char Banderas;
} BLOQUEIMAGENGIF;
```

Si se encuentra una coma en el byte introductorio, vienen uno de estos bloques inmediatamente.

Los elementos *Izquierda* y *Superior* del bloque de imagen especifican las coordenadas a las cuales se debe desplegar la imagen en la pantalla del monitor. Los valores *Ancho* y *Altura* representan las dimensiones de la imagen que está siendo descodificada.

El elemento *Banderas* se comporta más o menos igual que la bandera global del encabezado, con una diferencia importante: si *Banderas & 0x40* es verdadero, las líneas del archivo se encuentran intercaladas. Un archivo GIF intercalado particiona la imagen en 4 subimágenes; la primera de ellas consiste de cada 8 líneas de la imagen, comenzando con la línea cero; la segunda consiste de cada 8 líneas, comenzando con la cuarta línea; la tercer subimagen consiste de cada 4 líneas, comenzando con la línea 2; la subimagen final consiste de cada 2 líneas, comenzando con la línea 1.

La utilidad de las imágenes intercaladas es que es posible hacerse de una idea razonable de toda la imagen cuando solamente un cuarto de ella ha sido descodificado; esto era más relevante cuando las computadoras trabajaban más lentas que ahora, pero aún existen aplicaciones que permiten visualizar los archivos GIF cuando se están descomprimiendo. Quizá no es sorpresa decir que este atributo de los archivos GIF complican en un grado considerable a aquellas aplicaciones que utilizan este formato.

El byte que sigue al bloque de imagen es el tamaño inicial del código para la imagen comprimida, algo que se discutirá en la siguiente sección; el siguiente byte es el primer byte del primer campo de la imagen comprimida. Los datos GIF comprimidos se almacenan en campos de hasta 255 bytes; cada campo consiste de un byte para indicar su longitud y después los datos del campo en sí; el último campo tiene una longitud cero.

5.2.2 Compresión en los archivos GIF.

La compresión RLE trabaja buscando esencialmente dos condiciones fijas en un archivo de imagen y remplazándolas con símbolos; en el caso de un campo de corridas de bytes, el símbolo puede representar más datos que los que remplaza y de esta manera los archivos comprimidos con algoritmos RLE pueden hacerse más pequeños. La compresión LZW trabaja dejando que los datos de la imagen definan los símbolos que se usarán; en un sentido figurado, es una compresión RLE en la cual el algoritmo de compresión se define a si mismo basándose en la imagen específica que se está codificando.

Se puede comprimir cualquier tipo de compresión con LZW. Los programas PKZIP y LHARC aplican la misma técnica para comprimir cualquier tipo de archivos; esto podría explicar porque los archivos GIF no se comprimen cuando se aplica el PKZIP a ellos, pues aplicar compresión LZW a archivos que ya han sido comprimidos no los hace más pequeños.

En los archivos GIF, los datos de la imagen siempre se comprimen basándose en un byte por pixel; esto podría parecer un desperdicio con imágenes de 16 colores, en las cuales se desperdiciarían 4 bits de un byte. En realidad, como se podrá ver más adelante, la compresión LZW realmente no comprime los bits sin usarse en un byte, y por lo tanto ignora esta información superflua. Es más fácil comprender la compresión LZW si se comienza por explicar el proceso de descompresión. El ejemplo que se presenta a continuación muestra la descompresión de un archivo GIF; se asume que el encabezado y el tamaño inicial del código ya han sido leídos, y que comienza el bloque de datos de la imagen.

Una función para descodificar datos comprimidos con LZW trabaja con tres objetos: un cadena de códigos, una tabla de códigos y una cadena de caracteres. La cadena de códigos son los datos que se leen del archivo comprimido, la cadena de caracteres son los datos descomprimidos, y la tabla de códigos consiste de una pila de entradas en la cual los códigos asociados del archivo se asocian con cadenas de datos. En un archivo GIF la tabla de códigos tiene 4096 entradas; es decir, los datos de la imagen en un archivo GIF pueden comprimirse hasta con 12 bits por código ($2^{12} = 4096$). Se debe notar que estos 12 bits se refieren al máximo número de bits que la función de compresión LZW puede utilizar en un código para la máxima eficiencia en la compresión; esto no es lo mismo que el número de bits de color que puede contener el archivo. La efectividad de la tabla de compresión mejora con el número máximo de bits permitidos en un código. En algunas casos se obtiene mejor compresión con un tamaño máximo de 14 bits por código, en vez de 12 bits; sin embargo, una función de compresión con 14 bits requiere una tabla con 16,384 entradas, aunque para propósitos prácticos se requieren 2 enteros y un byte, dando un total de 5 bytes por entrada, es decir, 81, 920 bytes para la tabla de códigos. Esto no solamente es una gran cantidad de memoria , sino que es muy larga para direeccionarla en lenguaje C con un simple arreglo. Tener 12 bits como un tamaño máximo de código es un compromiso razonable entre la compresión del archivo y la complejidad del código requerido para manejar los archivos GIF.

Otros códigos que se necesitan antes de comenzar la descompresión son:

- #define TAMANO_CODIGO (1 << bits_por_pixel)
- #define CODIGO_LIMPIAR TAMANO_CODIGO
- #define FIN_IMAGEN (CODIGO_LIMPIAR + 1)
- #define CODIGO_LIBRE (CODIGO_LIMPIAR + 2)

Antes de comenzar el proceso de descodificación, la tabla de códigos debe inicializarse parcialmente; específicamente, las primeras TAMANO_CODIGO entradas deben inicializarse con su posición en la tabla, es decir, la entrada cero se inicializa a cero, la entrada uno se inicializa a uno, etc. Para una imagen con 8 bits por color, es decir, 256 colores, se inicializan las entradas de 0 a 255; el código 256 es el código de limpiar, el código 257 es el fin de la imagen y el código 258 es el primer código libre. Si un descodificador encuentra el código de fin de imagen termina el proceso.

El uso del código de limpiar podría no ser muy claro por lo cual se explica a continuación. Cuando se comprime una imagen, la tabla de códigos podría llenarse; mientras existan menos de 4096 cadenas, la función codificadora es libre de utilizar las cadenas existentes y de añadir nuevas cadenas; sin embargo, si la tabla se llena se deben desechar estas cadenas y comenzar a llenar la tabla nuevamente. Por este motivo, es necesario señalar este evento de manera que cuando el archivo está siendo descodificado, la función de descodificación conoce qué debe desechar la tabla de cadenas y comenzar nuevamente; esta señal es el código de limpiar.

La figura 5.1 muestra el proceso de descodificación de un archivo GIF.

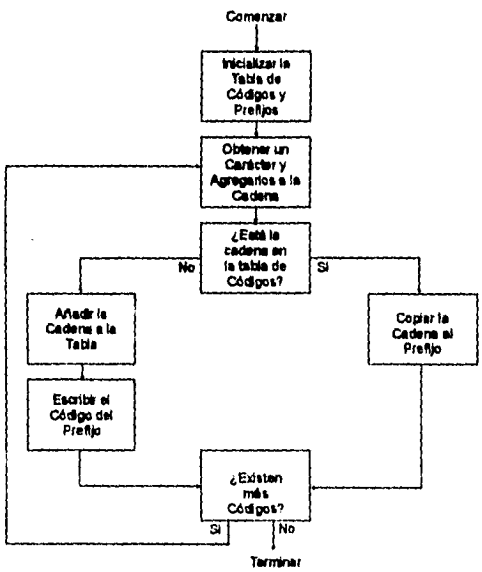


Figura 5.1 Diagrama de Flujo para el Proceso de Descompresión de un Archivo GIF.

A diferencia de los otros formatos, un codificador GIF no se preocupa del fin de las líneas; considera a la imagen entera como un bloque de datos y simplemente lo comprime. Un descodificador GIF reconstruye la imagen línea por línea porque generalmente así es como se utiliza.

Para comprimir un archivo GIF se efectúan los pasos contrarios al proceso de descompresión; simplemente se intercambian las cadenas de códigos y de caracteres. Es importante notar que la tabla de códigos de compresión está orientada a bits (esta es la razón por la cual es efectiva); no está restringida a tratar cadenas alineadas a bytes, tal como se hace en los algoritmos RLE. Los mapas de bits, a pesar de todo, son cadenas de bits aunque se almacenen como bytes; sin embargo, debido a que la información en una computadora se almacena en forma de bytes, es necesario que un descodificador GIF haga bastantes manipulaciones de bits y por lo tanto el proceso de descodificación sea un poco lento.

La especificación GIF para una imagen tiene establecido que ésta se guarde con un byte por pixel, no importan cuantos bits por color tenga. No existe desperdicio de espacio en efectuar esto porque la tabla de compresión comprime bits, no bytes; si 4 bits de un byte especifican información del color, solamente se empacan estos 4 bits.

5.2.3 Bloques de Extensión GIF.

La idea de los bloques de extensión fue mencionada en la sección 5.2.1, la cual habla sobre la estructura de los archivos GIF. Es posible ignorar totalmente estos bloques si se tiene interés únicamente por las imágenes primarias de un archivo GIF; una de las características de los bloques de extensión es que pueden saltarse sin conocer el contenido de ellos y esto no afecta a la aplicación. Las aplicaciones más sofisticadas de los archivos GIF probablemente hagan uso de los bloques de extensión; con estos bloques es posible crear animaciones de las imágenes un solo archivo GIF, archivos interactivos, archivos con elementos que aparecen y desaparecen, etc.

Un bloque de extensión comienza con un signo de admiración '!'; en la última revisión de los archivos GIF se tienen definidos 4 tipos de bloques de extensión.

- **Extensión para el Control Gráfico.** Permite definir la manera en la que se despliega la imagen, si se desea interrumpir el despliegue, cuanto tiempo dura éste, etc.
- **Extensión de Comentarios.** Permite añadir texto de comentarios en un archivo de imágenes.
- **Extensión de Texto Plano.** Contiene texto que se posiciona y despliega en la pantalla gráfica.
- **Extensión de Aplicación.** Contiene cualquier mejora a los archivos GIF, específica de la aplicación.

Un bloque de comentarios está definido por el valor 0FEH después del signo de admiración; consiste de hileras de texto ASCII, el cual se almacena como múltiples subbloques de datos. Un subbloque de datos se define como un byte para indicar la longitud del bloque, seguido de tantos bytes como se hayan definido en la longitud; el máximo tamaño de un comentario es de 255. El final de un bloque de comentarios se define con un byte que tiene una longitud cero.

Un bloque de texto plano se identifica con el byte 01H después del signo de exclamación; siguiendo a este byte se encuentra la siguiente estructura:

```
typedef struct tagBLOQUETEXTOGIF{
      char TamanoBloque;
      unsigned int Izquierda;
      unsigned int Superior;
      unsigned int AnchoReja;
      unsigned int AlturaReja;
      char AnchoCelda;
      char AlturaCelda;
      char ColorFrente;
      char ColorFondo;
} BLOQUETEXTOGIF;
```

Los elementos *Izquierda* y *Superior* indican el punto de inicio, en pixeles, del texto; los campos *AnchoReja* y *AlturaReja* especifican la distancia en pixeles de un carácter al siguiente; los elementos *AnchoCelda* y *AlturaCelda* especifican las dimensiones, en pixeles, de los caracteres que se escribirán. Hay que notar que el archivo GIF no define un tipo de letra para el texto, asume que se encuentra algún tipo disponible en el software interpretador (aunque es posible almacenar tipos de letras en un bloque de aplicación). Los elementos *ColorFrente* y *ColorFondo* son los índices al mapa de color para indicar los colores del texto y su fondo.

Después de la anterior estructura se encuentra el texto, definido en subbloques de datos como los descritos anteriormente.

Cualquier archivo GIF que tenga más de una imagen debe contener un bloque de control, de esta manera el descodificador GIF puede conocer la manera de desplegarlas. El byte que identifica a un bloque de control gráfico es 0F9H; inmediatamente después de este bloque se encuentra la siguiente estructura:

```
typedef struct tagBLOQUECONTROLGIF{
      char TamanoBloque;
      char Banderas;
      unsigned int Retardo;
      char ColorTransparente;
      char Terminador;
} BLOQUECONTROLGIF;
```

El campo *TamanoBloque* siempre tiene el valor 04H. Si *(Banderas & 0x01)* es verdadero, el elemento *ColorTransparente* contiene un índice a un color transparente válido; este color, cuando se despliega el contenido del siguiente bloque de imagen, no se debe desplegar, permitiendo de esta manera que la anterior imagen no sea cubierta totalmente y dando la impresión de transparencia. Si *(Banderas & 0x02)* es verdadero, el programa que despliegue la imagen debe esperar la entrada del usuario antes de desplegar la siguiente imagen del archivo.

Si el elemento *Retardo* es más grande que cero y se espera la entrada del usuario, el programa debe esperar a que pasen los segundos especificados o a que haya entrada del usuario, lo que suceda primero; si *Retardo* es mayor que cero y no se espera entrada del usuario, el programa debe esperar el número de segundos especificados.

El valor definido por *((Banderas >> 2) & 0x0007)* representa el método por medio del cual el programa borra la imagen anterior cuando se va a desplegar la siguiente imagen; se tienen las siguientes opciones:

- 0. Hacer nada.
- 1. Dejar la anterior imagen como está.
- 2. Restaurar el área con el color del fondo.
- 3. Restaurar el área con la imagen anterior a la que se está borrando.

Los bloques de aplicación son el último tipo de bloques de extensión y por definición, específicos a una aplicación. El byte que identifica a un bloque de aplicación es 0FFH, después del cual se tiene la siguiente estructura:

```
typedef struct tagBLOQUEAPLICACIONGIF {
      char TamanoBloque;
      char CadenaAplicacion[8];
      char Autentificacion[3];
} BLOQUEAPLICACIONGIF;
```

El campo *CadenaAplicacion* es una cadena de 8 bytes para identificar el software que sabe lo que debe hacer con el bloque en cuestión. El elemento *Autentificacion* contiene 3 bytes codificados con base en el contenido del campo *CadenaAplicación*; por ejemplo, se puede hacer un *XOR* de los tres primeros bytes de esta cadena y almacenarlos en *Autentificacion*; esto permite al software añadir información propietaria para asegurarse que en una posterior descodificación de este bloque se trate de un bloque propio y no de una bloque con la misma *CadenaAplicación*.

Siguiendo a esta estructura se encuentra la información de la aplicación, almacenada en subbloques de datos, como en los bloques de extensión descritos anteriormente.

5.2.4 Uso de los Archivos GIF.

La naturaleza ubicua de los archivos GIF los hace ser una buena elección entre los formatos gráficos si se desea crear imágenes de hasta 256 colores para portarse a una amplia variedad de plataformas; está fuera de duda que la mayoría de los sistemas de cómputo que cuentan con capacidades de despliegue gráfico incorporan un lector de archivos GIF. Aunque el estándar GIF no se diseñó para ser portable fácilmente, ha emigrado con más éxito a varios plataformas que otros formatos diseñado exprofeso para ello, tales como el TIFF. Se debe notar, sin embargo, que las aplicaciones de PC que usan gráficas de mapas de bits apenan están incorporando filtros para importar y exportar archivos GIF.

El formato GIF es ideal para almacenar y distribuir imágenes debido a que posee, tanto la posibilidad más pequeña de incompatibilidades, como la mejor y más efectiva técnica de compresión de imágenes de entre los otros formatos.

Otro uso de los archivos GIF es su utilización como medio para almacenar animaciones complejas, pues en este caso los bloques de control de imagen son una herramienta auxiliar valiosisima para indicar los movimientos y sobreposiciones de imágenes, y además, el tamaño del archivo resultante no es prohibitivamente grande.

5.3 Archivos PCX.

El estándar PCX es uno de los formatos de imágenes más antiguos y que aún se utiliza en sistemas de PC's, y probablemente es uno de los formatos gráficos más ampliamente soportado por procesadores de texto, paquetes de diseño y publicidad, capturadores de pantallas y otros.

Originalmente, el formato PCX nativo se desarrolló para el software *PC PaintBrush* de la compañía *ZSoft*, pero ha evolucionado conforme al desarrollo del hardware de despliegue y ha tenido una gran aceptación por otros desarrolladores de software. El formato PCX también es soportado en Windows con la aplicación *Windows PaintBrush* como una alternativa al formato BMP. Si se está creando software que intercambiará gráficas con otras aplicaciones es deseable que se incorporen rutinas para manejar los archivos PCX puesto que este formato es casi un estándar gráfico universal en los sistemas de PC's.

5.3.1 Estructura de Archivo.

Existen varios aspectos en la historia de los archivos PCX; el primero de ellos se refiere a sus antecedentes. El uso original de los archivos PCX fue para almacenar imágenes dibujadas a mano en vez de imágenes digitalizadas; esto no es de sorprenderse porque este formato apareció mucho antes de los digitalizadores estuvieran disponibles para el hardware de las PC's; la primera versión del *PC-Paintbrush* fue diseñada para correr en una PC XT 8088.

El formato PCX utiliza una forma particularmente simple de codificación RLE; los métodos de codificación y descodificación fueron creados para compensar la lentitud de los equipos PC iniciales y para favorecer las imágenes dibujadas con áreas bien definidas de color (tales como los histogramas, por ejemplo). Debido a que el procedimiento RLE utilizado para los archivos PCX fue diseñado con las dos condiciones anteriores, falla enormemente cuando se utiliza en imágenes digitalizadas; no es raro encontrar que las imágenes digitalizadas y almacenadas en formato PCX presenten compresión negativa, es decir, aumentan de tamaño al intentar comprimirlas.

La segunda consideración inherente en los archivos PCX es que éstos fueron diseñados antes de la tecnología actual de hardware de despliegue. El diseño original del formato PCX permitía un máximo de 16 colores; aunque se han hecho modificaciones posteriores que permiten manejar imágenes de 256 y hasta imágenes de 24 bits, dichas modificaciones no han sido del todo afortunadas, como se verá más adelante.

El último problema potencial con el formato PCX es el gran número de compañías que lo manejan. El formato PCX fue definido e introducido ampliamente al mundo comercial por *ZSoft*, después de lo cual surgieron otros desarrolladores de este formato; sin embargo, debido a que en un principio las especificaciones de *ZSoft* eran vagas, se interpretaron de diversas maneras y esto condujo a la aparición de numerosas aplicaciones que crean archivo PCX ilegales, es decir, archivos que no pueden ser leídos por software que se apega estrictamente al estándar. Existen maneras de tratar con estos archivos pero eso requiere bastante esfuerzo y dedicación, razón por la cual pocos programas lo hacen.

El archivo PCX es una entidad bastante simple; consiste de un encabezado de 128 bytes seguido por una gran cantidad de datos comprimidos con un algoritmo RLE. El estándar PCX para imágenes con 256 colores posee una paleta secundaria agregada al final de la imagen. Sin embargo, trabajar con archivos PCX no tiene la dificultad de trabajar con bloques como en el formato GIF, o con etiquetas y directorios como en el formato TIFF. El formato PCX define imágenes con 1 a 4 bits por color como planos interfoliados; las imágenes con 8 bits por color se almacenan con un byte por pixel. No existen imágenes PCX con 5 a 7 bits por color; una imagen con un número intermedio de bits por color se redondea a 256 colores si se convierte al formato PCX. Las imágenes con 24 bits por color se almacenan con 3 bytes por pixel, aunque esto es un caso inusual.

El encabezado de un archivo PCX es el siguiente:

```
typedef struct tagENCABEZADOPCX {
      char Manufactura;
      char Version;
      char Codificacion;
      char Bits;
      int  Xmin, Ymin;
      int  Xmax, Ymax;
      int  Hres;
      int  Vres;
      char Paleta[48];
      char Reservado;
      char PlanosColor;
      int  BytesPorLinea;
      int  TipoPaleta;
      char Relleno[58];
} ENCABEZADOPCX;
```

El campo de *Manufactura* siempre contiene el valor de 10; esta es la manera en que un lector de archivos GIF reconoce un archivo con este formato. El identificador *Version* indica la versión de *PC Paintbrush* que, presumiblemente, ha creado el archivo; este campo es bastante útil porque indica información acerca de la paleta. Específicamente, si el número de versión es cero, el archivo proviene de *PC Paintbrush* versión 2.5 y es antiguo; si *version* es 2, se utilizó *PC Paintbrush* versión 2.8 y contiene información válida para la paleta; si el valor es 3, proviene de *PC Paintbrush* versión 2.8 pero no contiene información de la paleta y se asume que se utilizó la paleta por omisión del paquete. Si el número de versión es 5 el archivo se creó con la versión 3.0 o superior de *PC Paintbrush*; esta es la versión correcta de los archivos PCX con imágenes de 256 o más colores.

El campo *Codificacion* especifica el tipo de compresión utilizada en este archivo; hasta el momento, existe solamente un tipo de compresión (RLE) y este campo siempre debe contener el valor de 1. Los elementos *Bits* y *PlanosColor* definen el número de colores en un archivo. Específicamente, el valor de *PlanosColor* es el número de planos de imágenes por línea y el valor de *Bits* es el número de bits por plano; en archivos planares, por ejemplo, un archivo PCX con 16 colores y 4 planos por línea, estos campos deben ser 4 y 1, respectivamente. Podría pensarse que es imposible tener más de un bit por plano, siendo un plano, por definición, una entidad con un solo bit por pixel; en realidad, este no es el caso, existen 2 tipos de archivos PCX que tienen planos de múltiples bits, los cuales se presentarán en la sección 5.3.2.

Los valores *Xmin* y *Ymin* definen la esquina superior izquierda de la imagen almacenada en el archivo PCX; los valores *Xmax* y *Ymax* definen la esquina inferior izquierda. El ancho de la imagen en pixeles se calcula como *Xmax - Xmin + 1* y la altura como *Ymax - Ymin + 1*; los valores *Xmin* y *Ymin* usualmente son cero pero no se debe asumir esto. El elemento *BytesPorLinea* indica al lector PCX que tan larga debe ser la línea descodificada

El elemento *TipoPaleta* contiene 1 para una paleta de color y cero para una paleta de grises; en realidad, ambos tipos de paletas están estructurados idénticamente, excepto que en la segunda se utilizan tonos de grises. La mayoría de los lectores PCX ignoran este campo. El campo *Paleta* contiene las definiciones de 16 colores RGB utilizados en el archivo; desafortunadamente, esto tiene poco uso en imágenes de 256 colores, por lo cual un archivo PCX con 8 bits por color usualmente no tiene algún valor almacenado en este campo.

Como se mencionó previamente, un archivo PCX de 256 colores almacena la información de la paleta al final de los datos de la imagen. Debido a que cada color en la paleta requiere 3 bytes, esta paleta tiene 768 bytes de longitud y siempre es precedida por un byte con el valor de 12. De esta manera, la paleta se puede encontrar localizando el fin del archivo, regresarse 769 bytes y leer el byte en esta posición; si es 12, se leen los siguientes 768 bytes como la paleta; si no es 12, posiblemente el archivo esté corrupto.

La compresión RLE de los archivos PCX es fácil de comprender. Después de que se ha leído el encabezado, el siguiente byte es el primer byte de la imagen comprimida; para descomprimir una línea de la imagen, se debe leer el primer byte y verificar si los dos bits más significativos está prendidos; si es así, se leen los siguientes *n* bytes del archivo, donde *n* es el valor almacenado en los 6 bits menos significativos. Si los dos bits más significativos no están prendidos, el byte debe escribirse tal cual al buffer en el cual se está descodificando la línea. Una vez que se ha leído un campo se debe verificar si se han escrito suficientes bytes al buffer para completar la línea, tal como está definido por *BytesPorLinea*; si este no es el caso, se descodifica el siguiente byte y se repite el proceso. Las líneas comprimidas en un archivo PCX está restringidas a terminar en valor par; cuando se descomprime una línea siempre se debe alcanzar la longitud definida por *BytesPorLinea*, como se había indicado anteriormente.

Existen varios cuestiones relevantes acerca de la compresión en los archivos PCX. Para comenzar, la longitud de un campo no puede ser mayor de 63 (el máximo valor de 6 bits); si se tienen líneas más largas se deben utilizar varios campos aunque la corrida de pixeles sea toda la línea. En segundo lugar, no existe un código de byte en sí, es decir, para un byte que tiene prendido sus dos bits más significativos, es necesario utilizar dos bytes para codificarlo, uno para denotar que es una corrida de longitud 1, y otro que es el byte en sí; de esta manera, existen cadenas que en vez de comprimirse duplican su longitud.

Finalmente, se debe notar que los bytes con valores arriba de 192 tienen prendidos sus dos bits más significativos; esto quiere decir que un cuarto de todos los posibles valores de pixeles, en un archivo de 256 colores sin muchas corridas de bytes, deben almacenarse con campos de dos bytes. Esto explica la baja razón de compresión de los archivos PCX con imágenes digitalizadas de 256 colores 0 24 bits por color.

5.3.2 Formatos de Líneas PCX.

Como regla, las líneas del formato PCX están estructuradas de manera que correspondan más o menos al hardware de despliegue en la PC, asumiendo que el archivo PCX se utiliza en software basado en DOS.

Los archivos PCX monocromáticos se almacenan con un bit por plano en cada línea; las imágenes que tienen de 2 a 4 bits por color se almacenan como planos interfoliados. Los archivos PCX con imágenes de 24 bits consisten de 3 planos, cada plano con 8 bits de profundidad; el primer plano contiene todos los pixeles rojos de la línea, el segundo todos los verdes y el tercero todos los azules;

La estructura de los archivos PCX de 24 bits es, efectivamente, orientada a pixeles y no a bytes; sin embargo, debido a que el tamaño de 3 bytes de un pixel es difícil de trabajar en la computadora, el formato PCX maneja la compresión de estos pixeles como bytes individuales, aplicando el proceso de compresión 3 veces para cada línea. En relación a otros formatos de imágenes de 24 bits, tales como el BMP o el Targa (discutido en la sección 5.4), el formato PCX es el único que intenta comprimir imágenes en color real. Los programas que manejan el formato targa, por ejemplo, dan al usuario la opción de comprimir una imagen de 24 bits, dependiendo de su contenido.

Los archivos PCX de 16 colores son, quizá, el tipo de imágenes más frecuentes que se pueden encontrar. Éstas representan el uso más común del formato PCX, siendo utilizadas para almacenar gráficas de negocios y en capturadores de pantalla, entre otras aplicaciones. Sin embargo, existen numerosas aplicaciones que no crean correctamente los archivos PCX; existen 3 principales variaciones de la forma en que las líneas de 16 colores se almacenan. Las especificaciones de *ZSoft* definen que un generador de archivos PCX no debe comprimir fuera del fin de línea; esto significa que para una imagen monocromática, dos líneas que pudieran ser comprimidas como una sola corrida deben dividirse en dos, de manera que en el proceso de descodificación se utilice correctamente el valor de *BytesPorLineas*. Los archivos PCX de 16 colores tienen 4 planos y las especificaciones de *ZSoft* dejan dudas acerca de como se debe almacenar una línea, ya sea como cuatro planos distintos, cada uno de ellos terminando en su frontera, o como un plano, terminando en la frontera de la línea; la segunda forma es preferida porque conduce a una compresión ligeramente mejor que la primer forma, sin embargo, esta última también es correcta.

A continuación se describe el problema que se presenta con estas particularidades. Un archivo PCX con una dimensión de 640 x 480 pixeles tiene un valor de 80 en el elemento *BytesPorLinea* y un valor de 4 para *PlanosColor*; hay 80 bytes en el plano de una línea, y 4 planos por línea. Si cada plano se comprime individualmente, se podría descomprimir 80 bytes del primer plano, seguido de 80 bytes para el segundo, y así. Pero si los 320 bytes de los 4 planos se comprimen juntos, al momento de desempacar podría ser que parte de un plano esté contenido al inicio del otro, causando de esta forma que no se lean correctamente los 80 bytes del plano individual. En este caso es mejor leer totalmente en un buffer los 320 bytes como si fueran un solo plano comprimido y posteriormente copiar los planos individuales, ya desempacados, a un segundo buffer. Con esto se soluciona el problema para la mayoría de los archivos PCX ilegales de 16 colores.

Otro problema no documentado por *ZSoft* es que existe una mutación común en los archivos PCX de 16 colores. En este tipo de archivos las imágenes no se guardan como planos, sino simplemente se almacenan con el formato de Windows de pilas de *nibbles*, al igual que los archivos BMP; cada byte en una línea almacena 2 pixeles. Este tipo de archivos es creado por el paquete de captura de pantalla *Tiffany*, junto con otros, y es aceptado por la aplicación *Windows PaintBrush*. Un lector de PCX archivos PCX puede definir si un archivo es de este tipo porque la imagen ha sido almacenada como un plano y 4 bits por plano, condición que no se puede dar en ninguna otra circunstancia.

5.3.3 Uso de los Archivos PCX.

Soportar el formato PCX hace que las aplicaciones entren a un estándar aceptado universalmente por la mayoría de las aplicaciones. Si bien es cierto que el formato GIF se está convirtiendo en un estándar, es innegable que el formato PCX lo ha sido por años. Sin embargo, está en proceso de perder esta distinción debido a que las imágenes se hacen cada vez más complejas y más grandes, y la poca eficiencia que presenta este formato al comprimir dichas imágenes deja mucho que desear, como ya se mencionó anteriormente; inclusive, las propias aplicaciones de *ZSoft* soportan otros formatos gráficos.

A pesar de todo esto, existe más soporte al formato PCX en todos los sistemas de PC's que a ningún otro formato de imágenes tipo mapa de bits. Otra de las razones al soporte del formato PCX es que, como se ha visto en esta sección, es fácil de trabajar con archivos PCX; el código para leerlos y escribirlos es simple y no requiere mucha memoria como, por ejemplo, el formato GIF.

Además, existe una circunstancia que ha ayudado a que se conserve este formato: parece que el hardware de despliegue de las PC's ha llegado a un estado estable de máxima profundidad en color (24 bits para imágenes en color real) y parece que no va a exceder este límite en un futuro próximo. Esto es una ventaja para el formato PCX porque posiblemente ya no soportaría otra revisión y adición más.

5.4 Targa TGA.

Hasta hace poco tiempo, el hardware de Targa y sus archivos eran algo exótico para los usuarios de las PC's; las tarjetas Targa fueron sistemas de despliegue de alto nivel, ofreciendo presentaciones de imágenes en color real, cuando las PC's apenas ofrecían gráficas monocromáticas decentes; desde luego, la desventaja de estas tarjetas era su costo. Además del hardware de despliegue, la compañía *Truevision* creó también varias aplicaciones para el manejo de imágenes en color real, tales como *Lumena*, una aplicación para retocar fotografías.

Por mucho tiempo fue necesario una tarjeta Targa para desplegar una representación aceptable de una imagen en un archivo con formato Targa (TGA) y por lo tanto, era raro encontrar estas imágenes y muy pocos tenían la necesidad de tratar con ellas. Actualmente, el uso del hardware de Targa ya no es tan raro y existen además otros dispositivos con características similares de alta resolución en color; adicionalmente a esto, en el mercado hay digitalizadores que generan imágenes con excelente resolución, propias para almacenarse en formato TGA. Finalmente, otro hecho que ha ayudado a la popularidad de las imágenes Targa es la introducción de Windows 3.1, en el cual está disponible un sistema para tratar con color de alta resolución, aunque quizás con un poco de imperfección.

Existen varias razones que hacen atractivos a los archivos Targa si una aplicación requiere tratar con color de alta resolución. Una de estas razones es su relativa obscuridad hasta hace poco y el poco acceso que se tenía al hardware de Targa; esto ha hecho que los programadores no corrompan las especificaciones de este formato, tal como pasó con los archivos PCX y TIFF. En realidad, las especificaciones de Targa no son ambiguas y definen un formato simple y atractivo de manejar. Otra razón para usar archivos Targa es el hecho de que el hardware de Targa fue uno de los primeros en soportar la salida de los digitalizadores de color y la mayoría del software de estos aparatos pueden generar imágenes en formato Targa. Esta es la razón por la cual actualmente paquetes como *Aldus PhotoStyler*, *PhotoPaint*, *ImageIn* y otros soportan este formato.

5.4.1 Color Targa.

Una de los aspecto más atractivos de los archivos Targa como medio para almacenar imágenes en color real es que ofrecen una gran variedad de opciones para hacerlo. Se puede hacer una comparación contra el formato PCX, en el cual el manejo de un imagen con 24 bits por color no tiene más que una opción: almacenarlo con compresión RLE.

A través de este capítulo se ha manejado que las imágenes en color real tienen 24 bits por color, es decir, tres bytes para la información de color del pixel; este no es cierto en el formato Targa. Las especificaciones de Targa definen 3 formatos de almacenamiento para imágenes en color real, soportando 16, 24 y 32 bits por pixel. Un archivo Targa de 24 bits es probablemente el más fácil de comprender puesto que es similar a los formatos de 24 bits PCX y BMP, discutidos anteriormente Cada pixel de la imagen consiste de 3 bytes, cada uno de ellos para los componentes RGB del color; la única diferencia entre el formato Targa de 24 bits y los otros formatos es que el color de los pixeles se almacena en forma inversa, es decir, primero el azul y posteriormente el verde y el rojo; esto es muy conveniente bajo Windows debido a que es como éste trabaja sus mapas de bits.

Los pixeles Targa con 32 bits son similares a los de 24, excepto que el byte extra sirve para indicar la transparencia, o lo que se conoce más propiamente como *canal alfa*; este valor indica a la aplicación la manera en que debe sobreponer una imagen Targa sobre otra. Cada pixel indica la cantidad de opacidad con la que se aplicará al pixel en la anterior imagen; si el bit más significativo de este byte está prendido, los otros 7 bits indican la cantidad de transparencia para ese pixel; si dicho bit está apagado, el pixel es totalmente opaco. Sin embargo, las facilidades que proporciona Windows para manipular mapas de bits no incluyen funciones para manejar imágenes transparentes y desarrollarlas es bastante complicado.

El formato Targa de 16 colores ofrece imágenes dosificadas en color real; define un pixel con 5 bits para componente RGB del color, en vez de 8; de esta manera, los 15 bits resultantes se pueden almacenar en 2 bytes, en vez de 3, y el bit final se utiliza para definir la transparencia; la **figura 5.2** ilustra la relación entre pixeles de 16 y 24 bits. El número reducido de bits de color en una imagen Targa de 16 bits no afecta la brillantez del color de dicha imagen; sin embargo, reduce la precisión de los colores que se pueden definir. Una imagen de 24 bits puede tener colores en un rango de 16,777,216 sombras posibles; un archivo de 16 bits tiene solamente 32,767 colores posibles con los que se puede trabajar; esto cubre el mismo rango de las imágenes de 24 bits pero con menos gradaciones (cambios de color); en la práctica, 16 bits por color es aún bastante respetable; a menos que se tenga un manejador de Windows capaz de desplegar color real, no será posible ver la diferencia. Se debe notar que los manejadores de color real para Windows que utilizan tarjetas Super VGA con alta resolución de color, reducen sus colores a 15 bits de información para el color; se necesita una tarjeta de despliegue Targa de 24 bits y un manejador adecuado de 24 bits en Windows para ver las diferencias entre las imágenes Targa de 16 y de 24 bits.

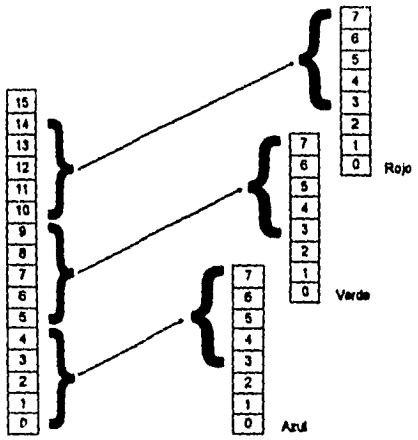


Figura 5.2 Proceso para Convertir una Imagen en Color Real de 24 bits a 16 bits.

Como se mencionó en las otras secciones anteriores, las imágenes de color real digitalizadas no se comprimen bien y en la mayoría de los casos es mejor no comprimirlas puesto que invariablemente se hacen más grandes. Una imagen de 640 x 480 pixeles con 24 bits por color almacenada con formato Targa de 24 bits requiere de aproximadamente 900K de espacio en disco; la misma imagen almacenada como un archivo de 16 bits solamente requiere un poco más de 600K, con una ligera pérdida en la resolución del color. Esto no es lo mismo que la compresión puesto que involucra desechar algo de información del color, pero no es un una mala opción.

Aunque los archivos Targa se utilizan para almacenar imágenes de color real, se debe notar que estos archivos pueden almacenar también imágenes monocromáticas y de 8 bits con paleta de colores y tonos de grises; en realidad, esto no es recomendable puesto que existen otros formatos que manejan mejor las imágenes de estos tipos.

El formato Targa puede almacenar imágenes ya sea sin compresión o con una forma simple de compresión RLE, la cual se describe más adelante. Aunque su algoritmo RLE es raramente efectivo cuando se aplica a imágenes digitalizadas, puede ser bastante útil para dibujar imágenes con paquetes de color de alta resolución tales como *Lumena* y *Aldus PhotoStyler* para Windows.

5.4.2 Estructura de Archivo.

Existen un número de extensiones complejas definidas por el formato Targa que proporcionan algunas de las funciones de los bloques de extensión de los archivos GIF y de las etiquetas TIFF; sin embargo, estas extensiones casi nunca se usan en la práctica y no se discuten en esta sección. Los otros elementos del formato Targa son bastante simples y ofrecen pocas complicaciones a las aplicaciones que desean leer y escribir imágenes en este formato.

Un archivo Targa comienza con un encabezado que define la imagen que se encuentra en el resto del archivo; este encabezado es:

```
typedef struct tagENCABEZADOTGA {
      char          TamanoIdentificador;
      char          TipoMapaColor;
      char          TipoImagen;
      unsigned int ComienzoMapaColor;
      unsigned int LongitudMapaColor;
      char          BitsMapaColor;
      unsigned int xComienzo;
      unsigned int yComienzo;
      unsigned int Ancho;
      unsigned int Altura;
      char          Bits;
      char          Descriptor;
} ENCABEZADOTGA;
```

Se debe notar que la auténtica especificación define una estructura diferente para el comienzo del archivo Targa; en la estructura definida por Targa se incluyen varios campos de longitud variable. Debido a que el lenguaje C no maneja estructuras de tamaño variable, la definición anterior presenta solamente la parte fija de la misma, más adelante se especifican los diferentes encabezados variables que pueden presentarse.

El campo *TamanoIdentificador* de un encabezado Targa define el número de bytes que se presentan después del encabezado, hasta un máximo de 255; este campo extra contiene una descripción o identificación para la imagen del archivo, si es que existe; la mayoría de los archivos Targa no utilizan esta característica y el campo *TamanoIdentificador* se pone a cero.

El campo *TipoMapaColor* define si el archivo posee un mapa de color, es decir, una paleta; las paletas se incluyen solamente para imágenes de 8 bits por color. Sin embargo, algunas aplicaciones Targa podrían incluir una paleta para una imagen en color real, de tal manera que la imagen pueda remapearse a esta paleta sin que se tenga que efectuar un proceso de cuantización de la paleta del sistema. Si este campo contiene cero, no existe información de la paleta, si es uno, la paleta continúa después del encabezado. De acuerdo a la especificación Targa, existen 256 tipos posibles de mapas de color, de los cuales los primeros 128 son reservados por *Truevision* para su propio uso y los otros 128 están disponibles para los desarrolladores de software; por esta razón, existe incertidumbre acerca de que hacer con estas paletas cuando se encuentra una.

El elemento *TipoImagen* tiene el valor que define el tipo de imagen del archivo; actualmente están especificados los siguientes tipos de imagen:

- 1. Imagen sin comprimir, con paleta.
- 2. Imagen RGB sin comprimir.
- 3. Imagen monocromática sin comprimir.
- 9. Imagen codificada con RLE, con paleta.
- 10. Imagen RGB codificada con RLE.
- 11. Imagen monocromática codificada con RLE.

La especificación Targa menciona que los valores de 0 al 127 deben ser considerados como códigos legales propietarios para almacenar imágenes que *Truevision* pueda desarrollar en el futuro; los valores mayores que 127 pueden utilizarse para tipos de almacenamiento propietarios del desarrollador de software. En realidad, los 6 valores descritos previamente son las más comunes de encontrar; verificar la presencia de uno de estos 6 número es suficiente para asegurar que se tiene un archivo Targa.

El campo *ComienzoMapaColor* define que color de la paleta es el primer color del mapa de colores del archivo; normalmente, este campo es cero para indicar que el primer color definido en el archivo corresponde al primer color de la imagen, aunque no siempre es el caso. El elemento *LongitudMapaColor* especifica el número de colores en el mapa de color; en archivos con más de 8 bits por color, es decir, con imágenes en color real, tanto *ComienzoMapaColor* como *LongitudMapaColor* tienen un valor de cero. El campo *BitsMapaColor* define como está estructurado el mapa de color en un archivo Targa, es decir, si tiene 16, 24 ó 32 bits por color; este valor indica el número de bits para la entrada a la paleta, no el número de bits en la imagen en sí; en una imagen en color real sin paleta este campo contiene cero. A pesar de la flexibilidad de las opciones disponibles para el mapa de color en las especificaciones Targa, los archivos con este formato raramente se utilizan para imágenes con paleta; su principal aplicación es para almacenar imágenes en color real puesto que existen mejores maneras de almacenar imágenes de 256 colores.

Los campos *xComienzo* y *yComienzo* especifican la distancia de la esquina superior izquierda de la pantalla a la esquina superior izquierda de la imagen; estos valores son cero en la mayoría de los casos. Los elementos *Ancho* y *Altura* definen las dimensiones en pixeles de la imagen. El campo *Bits* indica el número de bits por color del archivo Targa y la manera en que se debe interpretar la estructura de los pixeles; puede tener los valores 1, 8, 15, 16, 24 ó 32.

El campo *Descriptor* es un conjunto de banderas útiles para interpretar un archivo Targa; los 4 bits más significativos indican información acerca de la transparencia y no se discuten en esta sección, el bit 5 especifica la orientación de la imagen. Si *(Descriptor & 0x20)* es verdadero, la primera línea leída del archivo debe considerarse como la primer línea de la imagen, es decir, la imagen está de izquierda a derecha y de arriba hacia abajo; si *(Descriptor & 0x10)* es verdadero, la imagen está almacenada de derecha izquierda y de abajo hacia arriba y debe invertirse para ser desplegada correctamente.

Dos de los problemas más comunes encontrados cuando se escribe software para leer archivos Targa, es olvidar invertir la imagen cuando hay que hacerlo, y olvidar que los valores RGB para estos archivos están también invertidos como BGR.

Después de la parte fija del encabezado de los archivos Targa se encuentra el identificador de la imagen, si es que éste tienen un tamaño mayor que cero; después de este identificador se encuentra el mapa de color, si es que existe. El siguiente byte en el archivo debe ser el primer byte de los datos propios de la imagen.

Si el campo *TipoImagen* indica que la imagen está sin comprimir, las líneas del archivo pueden leerse con poca manipulación extra; sin embargo, el manejo de los 3 tipos de imágenes con compresión RLE requiere un poco más de trabajo. La codificación RLE del formato Targa es bastante sencilla; en este caso, el primer byte de una línea es el primer byte del campo comprimido y debe considerase como el byte clave. La longitud del campo es uno más el valor indicado en los 7 bits menos significativos del byte clave; si el bit más significativo está prendido, el campo es una corrida de pixeles, si no, es una cadena. Es importante mencionar que cuando se están descomprimiendo archivos Targa, excepto los monocromáticos, todos los campos se manejan en pixeles en vez de bytes; el número de bytes en un pixel varía de acuerdo al número de bits por color en la imagen, de esta manera, un campo comprimido de longitud 10 en un archivo de 24 bits por color realmente involucra 30 bytes de datos al descomprimirse, es decir, 10 pixeles de 3 bytes cada uno. Una vez que se ha descomprimido un campo, el siguiente byte debe considerarse como byte clave y se repite el mismo proceso hasta desempacar totalmente el archivo; al igual que otros formatos, los archivos Targa están restringidos a terminar en una palabra par. Una vez que se ha leído una línea, se debe invertir de izquierda a derecha si la bandera correspondiente en el campo *Descriptor* así lo indica. Se debe notar que el método de compresión RLE de los archivos Targa ocupa bastante tiempo de procesador, a comparación de la compresión RLE de los archivos PCX.

5.4.3 Uso de los Archivos TGA.

Hay diversas aplicaciones específicas para los archivos Targa, aún cuando nunca se tenga una Tarjeta Targa verdadera. Los archivos Targa son un buen medio para exportar archivos con imágenes digitalizadas y han llegado a convertirse en un formato por omisión para la mayoría de los paquetes de *Ray-Tracing*[2]. Además, el software que procesa archivos Targa está comenzando a aparecer en otras plataformas tales como los sistemas Macintosh.

La naturaleza simple del formato Targa lo hace atractivo para trabajar con color en alta resolución; aunque realmente no hace mucho, lo hace bien.

5.5 Archivos TIFF.

El formato TIFF fue diseñado por un comité formado por *Hewlett Packard*, *Microsoft* y *Aldus*; TIFF es un acrónimo de *Tag Image File Format* (Formato de Archivo de Imágenes Etiquetadas). Su propósito inicial fue proporcionar un formato de archivo que fuera independiente del tipo de imagen, de la aplicación y de la plataforma de hardware; es decir, un archivo TIFF podría almacenar cualquier tipo de imagen, ser utilizado en cualquier tipo de computadora y ser leíble por cualquier aplicación. Sin embargo, ninguna de estas especificaciones resultaron ciertas; es posible escribir archivos TIFF legales que incluyan todas estas características, o escribir archivos TIFF que no las incluyan pero que aún así sean legales también.

La característica más sobresaliente de las especificaciones TIFF es su flexibilidad, casi sin límite, puesto que puede adaptarse para cualquier propósito; la desventaja inherente en esto es que es casi imposible desarrollar un lector de archivo TIFF que pueda leer todos los tipos de éste; además, las especificaciones del formato TIFF son ambiguas en muchos aspectos. Como se mencionó en las secciones anteriores, una de las cosas que ocurren más frecuentemente cuando se leen imágenes de múltiples fuentes, es que las especificaciones ambiguas hacen que las interpretaciones de una imagen sean efectuadas en distinta forma por los distintos desarrolladores de software.

En la mayoría de los formatos, tal como con los archivos PCX, existen relativamente pocas especificaciones del diseño del formato; sin embargo, el formato TIFF tiene docenas de ellas. Lo más usual en estos casos es crear lectores de archivos TIFF para un rango finito y específico de aplicaciones, y escribir archivos TIFF para lectores específicos de éstos.

---

[2] El *Ray-Tracing* es un método de generación de imágenes realísticas.

5.5.1 Estructura de Archivo.

Una de las limitaciones más visibles de los formatos simples, tales como el PCX, es que ofrecen muy pocas probabilidades de expansión y por lo tanto tienden a ser un poco inflexibles. Dejar espacio para el "crecimiento" del formato asume que éste crecerá en un modo previsible; en el caso del formato PCX, cuando aparecieron las imágenes de 256 colores, los autores de este formato se dieron cuenta que no se había considerado espacio para este crecimiento.

Las especificaciones de TIFF definen que una imagen está construida por medio de bloques, de tal manera que cada bloque define ciertas características de la imagen que se está almacenando. Cada bloque se conoce como *etiqueta* (de aquí el nombre del formato) y cada imagen se define con directorios, o listas de etiquetas; un archivo TIFF puede contener varios directorios y por lo tanto múltiples imágenes en un solo archivo. El formato GIF define algo similar a esto con sus bloques de extensión, sin embargo, el formato TIFF es mucho más amplio en este aspecto.

Un archivo TIFF típico consiste de etiquetas que definen la dimensión de la imagen, etiquetas que definen como se almacena y comprime, etiquetas para definir la paleta, etc. Existen etiquetas TIFF opcionales para especificar el nombre del software que crea el archivo, etiquetas para mejorar el desempeño de la función de descompresión de la imagen, etiquetas para mejorar la impresión de un archivo en un dispositivo de salida específico, y otras más. En realidad, existen docenas de etiquetas disponibles para el desarrollador de software, y las aplicaciones son libres de añadir etiquetas propietarias a los archivos TIFF para su propio uso. Por definición, cualquier etiqueta que el lector TIFF no comprenda debe ser saltada; de esta manera, no se requiere que un software que lea un archivo TIFF con muchas etiquetas opcionales tenga que interpretarlas; sin embargo, esto no asegura que la información contenida en dichas etiquetas no sea útil o inclusive esencial para descodificar correctamente la imagen almacenada en el archivo. Cada etiqueta consiste de 12 bytes, de los cuales los primeros dos son un entero que define el tipo de etiqueta; un lector TIFF puede leer los dos primeros bytes para interpretar la etiqueta y si no la descifra, saltar los siguientes 10 bytes para leer la siguiente.

Un archivo TIFF comienza con un pequeño encabezado; por el momento se puede definir de la siguiente manera:

```
typedef struct tagENCABEZADOTIFF {
      int  TipoNumero;
      int  Version;
      long Desplazamiento;
} ENCABEZADOTIFF;
```

En realidad, por razones obvias que se verán a continuación, este encabezado no puede ser leído con una estructura en lenguaje C.

El elemento *TipoNumero* en un encabezado TIFF siempre contiene uno de los valores 4949H ó 4D4DH; es fácil ver que 49H es el código ASCII para la letra I y 4DH es el código ASCII para la letra M; éstas son las iniciales de Intel y Motorola, respectivamente. Si la primer palabra de un archivo TIFF contiene la constante 'II', todos los bytes del archivo están definidos al estilo Intel; si contiene 'MM' están definidos al estilo Motorola[3].

El elemento *Version* del encabezado TIFF es un entero que contiene el valor 42 o 2AH; sin embargo, hay que notar que este valor puede aparecer en cualquiera de los bytes del entero, dependiendo del tipo de número utilizado (Intel o Motorala). El campo *Desplazamiento* especifica el desplazamiento a partir del comienzo del archivo en el cual se encuentra el primer directorio de imagen; en un archivo con una imagen solamente existe un directorio; nuevamente, el orden de los bytes en este campo depende del tipo de número utilizado.

Un directorio de imagen consiste de un entero que especifica el número de etiquetas en el directorio, las etiquetas mismas y un entero largo; este último es cero si no existen más directorios de imagen, o si no es cero indica el desplazamiento al siguiente. Cada etiqueta TIFF se define con la siguiente estructura; nuevamente, no se debe leer de esta manera la etiqueta debido al tipo de número.

```
typedef struct tagETIQUETATIFF {
      unsigned int  NumeroEtiqueta;
      unsigned int  Tipo;
      unsigned long Longitud;
      unsigned long Desplazamiento;
} ETIQUETATIFF;
```

El elemento *NumeroEtiqueta* es una de las muchas constantes definidas para identificar el tipo de etiqueta; estas constantes se definen en la sección 5.5.4; por ejemplo, si el valor es 256, la etiqueta define el ancho de la imagen. Las etiquetas en un archivo TIFF están restringidas a escribirse en un orden numérico ascendente.

El campo *Tipo* especifica el tipo de valor que representa la etiqueta; se tienen los siguientes:

- 1. La etiqueta especifica un valor en un byte.
- 2. La etiqueta especifica un desplazamiento a una cadena ASCII.
- 3. La etiqueta especifica un entero corto.
- 4. La etiqueta especifica un entero largo.
- 5. La etiqueta especifica un desplazamiento a dos enteros largos, los cuales forman el numerador y denominador de un número racional.

[3] Específicamente, los número enteros en la arquitectura Motorola están con los bytes intercambiados en comparación con la arquitectura de Intel, es decir, los bytes más bajos se encuentran a la izquierda y los más altos a la derecha.

Si el valor que define la etiqueta requiere 4 o menos bytes, el valor puede encontrarse en el elemento *Desplazamiento*; si requiere más de 4 bytes (una cadena es un buen ejemplo), el elemento *Desplazamiento* especifica el desplazamiento donde se encuentra ésta dentro del archivo. La longitud de un objeto definido por una etiqueta se encuentra especificada en el elemento *Longitud*.

Como se puede ver, descodificar un archivo TIFF usualmente requiere de una buena cantidad de búsqueda en éste.

5.5.2 Aproximación de las Etiquetas TIFF.

La etiqueta más comúnmente encontrada es la de *Compresion*. Una imagen en un archivo TIFF puede comprimirse utilizando una gran variedad de algoritmos; a continuación se presenta una lista de los algoritmos más usados, aunque algunos de ellos han sido declarados obsoletos recientemente.

- 1. La imagen está sin comprimir.
- 2. La imagen está comprimida con códigos de Huffman.
- 3. La imagen está comprimida con la codificación del grupo 3 de la CCITT.
- 4. La imagen está comprimida con la codificación del grupo 4 de la CCITT.
- 5. La imagen está comprimida con el algoritmo LZW.
- 32773. La imagen está comprimida con empaquetamiento de bits, utilizado en las imágenes de Macintosh.

Uno de los aspectos positivos de los diferentes tipos de compresión que se pueden aplicar a los archivos TIFF es que es posible implementar la efectividad y desempeño que se requiera para éstos, desde las imágenes sin compresión hasta la compresión con razones que se acercan a las del formato GIF; la desventaja de esto es que se requiere bastante código de programación y casi nadie lo hace; como consecuencia de esto, para cualquier aplicación con facilidades modestas para importar archivos TIFF, siempre habrá un conjunto de archivos que no serán leíbles.

La compresión con códigos de Huffman y con los grupos 3 y 4 de la CCITT requieren una gran cantidad de memoria para descompactarse; esto es debido a las extensas tablas de códigos de Huffman necesarias para el proceso de descompresión. Además, la codificación Huffman fue diseñada originalmente para manejarse en hardware en vez de software y es la base para la transmisión de facsímiles. La circuitería de compresión de imágenes en una máquina de facsímil puede implementarse con lógica digital sencilla, la cual opera cientos de veces más rápido que la compresión y descompresión basada en software; la lógica que maneja la descompresión de Huffman en hardware pueden manejar genuinamente cadenas de bits en vez de bits empacados en bytes, como lo hace el software. Por estas razones, la codificación de Huffman hecha en software es inelegante y demasiado lenta.

El estándar TIFF permite almacenar imágenes de color ya sea utilizando una paleta, como en el caso de los archivos GIF, o como valores de color RGB, como los archivos Targa. Consecuentemente, existe una etiqueta para hacer esto; en archivos con paleta, la información de ésta se almacena en la etiqueta *MapaColor*, la cual define el desplazamiento en el archivo en donde se encuentra la tabla de colores; esta tabla contiene los valores RGB para los colores de la paleta de la imagen, pero como sucede frecuentemente con los archivos TIFF, la paleta está organizada en un modo muy diferente al de los otros formatos. Un mapa de color TIFF consiste de todos los valores rojos en la paleta, seguidos de todos los valores verdes, y finalmente todos los valores azules; cada color se almacena en un entero en vez de un byte; para convertir estos valores de 16 bits en valores de 8 bits se hace un corrimiento de 8 bits hacia la derecha. No es necesario remarcar que un mapa de colores del formato TIFF ocupa el doble de espacio que en otros formatos y además no se encuentra estructurado como valores RGB normales.

Cuando se lee un mapa de colores TIFF, la información actual del color se encuentra localizada típicamente a cierta distancia de la etiqueta; es necesario grabar la posición actual en el archivo, buscar en donde apunta el elemento *Desplazamiento* y regresar a la posición anterior para leer la siguiente etiqueta.

La manera en que se almacena la información en un archivo TIFF, es decir, si la imagen tiene una paleta o color RGB, es definida por la etiqueta *InterpretacionFotometrica*, la cual contiene uno de los siguientes valores:

- 0. La imagen es monocromática, con un solo plano en el cual todos los pixeles prendidos se deben considerar negros.
- 1. La imagen es monocromática, con un solo plano en el cual todos los pixeles prendidos se deben considerar blancos.
- 2. La imagen consiste de pixeles RGB, cada uno de los cuales requiere 3 bytes, 1 para cada componente RGB.
- 3. La imagen está almacenada con paleta de color.
- 4. La imagen es monocromática y se utiliza como máscara de transparencia, presumiblemente para otras imagen en el mismo archivo TIFF.

Hay dos cosas que se deben notar acerca de esta etiqueta: las imágenes de tipo 1 son las imágenes monocromáticas normales, y el tamaño actual y la profundidad de los pixeles RGB no tienen que ser necesariamente 24 bits en el formato TIFF; el número de bits por color está especificado por la etiqueta *BitsPorMuestra*.

La etiqueta *BitsPorMuestra* realmente define el número de bits requeridos para representar un pixel; éste es uno para archivos monocromáticos, 4 para los archivos con 16 colores y 8 para archivos con 256 colores; para imágenes en color real no es 24 bits, en realidad es una desplazamiento. Al respecto de esto, la especificación TIFF permite definir en los archivos RGB que cada uno de los componentes tengan diferente profundidad de color; por lo tanto, el desplazamiento definido en *BitsPorMuestra* apunta a una tabla de 3 enteros, cada uno de ellos especifica el número de bits en cada uno de los 3 componentes de color de un pixel; típicamente, los valores son (8, 8, 8); realmente es imposible imaginarse una aplicación que quiera definir, por ejemplo, un componente rojo con una precisión más grande que los otros componentes.

Finalmente, el otro problema potencial frecuentemente encontrado es la etiqueta *DesplazamientoBanda*, la cual requiere un poco más de explicación. Una de las primeras aplicaciones del formato TIFF fue tratar con la salida de digitalizadores; por su naturaleza, los digitalizadores de razonable calidad producen archivos enormes de salida y los digitalizadores de excelente calidad producen archivos que son tan grandes que no alcanza la memoria para contenerlos. Para remediar estos problemas, el formato TIFF especifica que las imágenes pueden dividirse en bandas; una banda es una porción horizontal de la imagen que la memoria puede contener, aún cuando no pueda contener la imagen total; la especificación TIFF define que una banda ideal ocupa menos de 8 Kb de memoria. En la mayoría de las aplicaciones de los archivos TIFF, se decide que la imagen sea una sola banda debido a que las imágenes con múltiples bandas tardan mucho más en desempacarse. En archivos con la imagen en una sola banda, la etiqueta *DesplazamientoBanda* indica el desplazamiento a partir del cual comienzan los datos de la imagen, los cuales deben interpretarse de acuerdo a la etiqueta *Compresion*; sin embargo, en archivos con múltiples bandas, *DesplazamientoBanda* apunta a una tabla de valores de subdesplazamientos, cada uno de ellos apuntando al inicio de la banda correspondiente de la imagen. Descodificar un archivo con múltiples bandas entraña bastante búsqueda a través del archivo.

5.5.3 Archivos TIFF del Mundo Real.

A diferencia de los archivos PCX, por ejemplo, todos los archivos TIFF son diferentes. Como se vio en la discusión anterior de las etiquetas, es posible estructurar un archivo TIFF legal de la manera que se quiera; se pueden añadir etiquetas para los requerimientos propios del desarrollador o simplemente porque nadie lo ha hecho antes, de cualquier manera las etiquetas extras de una aplicación pueden no tener sentido para otra y viceversa.

Una de las dificultades inherentes cuando se escriben lectores universales de archivos TIFF es encontrar ejemplos de grupos representativos de archivos TIFF para probarlos; por ejemplo, si se desea asegurar que un lector TIFF puede manejar correctamente los archivos RGB con diferentes profundidades para cada componente de color, es necesario encontrar alguna aplicación que los genere; por supuesto, ésta puede nunca existir. La mejor manera de comenzar es desarrollar un lector de archivos TIFF que pueda manejar los archivos que se han diseñado para no ser difíciles; una vez que se han dominado estos archivos, el lector podrá codificar y descodificar exitosamente un respetable número de archivos TIFF.

La primera etiqueta en un archivo TIFF es típicamente ignorada, es, ya sea *TipoSubarchivo* para los archivos antiguos, o *SubarchivoNuevo* para los recientes; la primera etiqueta es obsoleta e indica con un valor de cero que se tiene un archivo TIFF normal con una sola imagen, la segunda etiqueta tiene un valor de 1. Las especificaciones TIFF recomiendan no utilizar la etiqueta *TipoSubarchivo* pero aún existen un buen número de lectores TIFF que la buscan al leer el archivo, aunque también reconozcan la etiqueta *SubarchivoNuevo*.

Las etiquetas *Ancholmaen* y *LongitudImagen* son esenciales en cualquier archivo TIFF debido a que definen el tamaño de la imagen a desempacar. La etiqueta *BitsPorMuestra* se encuentra en casi todos los archivos TIFF y como se mencionó anteriormente, define la profundidad del color RGB de la imagen; el valor por omisión de esta etiqueta es 1, razón por la cual puede omitirse en un archivo monocromático. La etiqueta *Compresion* fue discutida en la sección 5.5.2; se encuentra en la mayoría de los archivos TIFF, aunque su valor por omisión es sin compresión, por lo tanto, se puede omitir en archivos sin comprimir.

La etiqueta *InterpretacionFotometrica* debe estar presente en todos los archivos TIFF puesto que define como se ha almacenado la imagen en el archivo y no tiene valor por omisión. La etiqueta *DesplazamientoBanda* también debe estar presente porque especifica la parte del archivo en la cual residen los datos de la imagen. La etiqueta *ConfiguracionPlanar* es opcional, aunque se encuentra en la mayoría de los archivos TIFF; contiene el valor uno para la mayoría de los tipos de archivos discutidos en esta sección, o dos para los archivos cuyas líneas se han almacenado como planos (al igual que en el formato PCX); los archivos tipo 2 no se discuten en esta sección debido a que casi nunca son utilizados.

Los datos de la imagen, apuntados por *DesplazamientoBanda*, se almacenan casi exactamente en la forma en que Windows lo requiere. Las imágenes con 2 ó 4 bits por color se almacenan en pilas de *nibbles*, es decir, igual que Windows almacena sus mapas de bits independientes del dispositivo; las imágenes en color real se almacenan con el orden de sus pixeles en rojo, verde y azul, siendo necesario intercambiar los bytes rojo y azul; las imágenes monocromáticas y de 8 bits por color se utilizan exactamente como aparecen en el archivo TIFF.

También se debe tener en mente que con la excepción del encabezado TIFF, ninguna de los otros objetos en un archivo TIFF necesitan estar en una posición fija; el directorio de imagen o los datos de la imagen pueden seguir al encabezado, dependiendo del gusto o necesidades del desarrollador de software.

Cuando se está creando software que trabaje con otras aplicaciones es muy útil revisar las etiquetas de los archivos TIFF que se supone la otra aplicación leerá, y después verificar qué es lo que la función escritora de archivos TIFF está haciendo diferente. En los archivos TIFF del mundo real, lo usual es identificar fuentes específicas y aplicaciones con las cuales se intercambiarán imágenes en este formato; es mucho más práctico escribir software que, por ejemplo, creará archivos para importar al paquete *PageMaker*, que crear archivos TIFF que se importarán a cualquier aplicación.

5.5.4 Etiquetas TIFF más comunes.

La siguiente lista muestra las etiquetas más comunes en los archivos TIFF; la lista no es exhaustiva.

- SubarchivoNuevo: 254 (FEH), tipo entero largo.
  Indica la naturaleza de la imagen en el archivo TIFF. Sus datos es un conjunto de 32 bits bandera; los bits no utilizados se ponen a cero; el valor por omisión es cero, lo cual indica que es un archivo con una sola imagen. Los bits se definen como sigue:

  bit 0. Prendido si la imagen es una versión reducida de otra imagen almacenada en el archivo.
  bit 1. Prendido si la imagen es una de las varias imágenes almacenadas en el archivo.
  bit 2. Prendido si la imagen es una máscara de transparencia.

- AnchoImagen: 256 (100H), tipo entero corto o largo.
  Define el ancho de la imagen en pixeles.

- LongitudImagen: 257 (101H), tipo entero corto o largo.
  Define la altura de la imagen en pixeles.

- BitsPorMuestra: 258 (102H), tipo entero corto.
  Define el número de bits por muestra; el valor por omisión es uno.

- Compresión: 259 (103H), tipo entero corto.
  Define el tipo de compresión utilizado en la imagen TIFF; el valor por omisión es uno. Los valores posibles son:

  1. Sin compresión.
  2. Codificación Huffman modificada.
  3. Codificación del Grupo 3 de la CCITT, compresión de facsímiles.
  4. Codificación del Grupo 4 de la CCITT, compresión de facsímiles.
  5. Compresión LZW.
  32773. Codificación con empaquetamiento de bits de Macintosh.

- IntepretacionFotometrica: 262 (106H), tipo entero corto.
  Indica como está almacenada la imagen. Sus valores son:

  0. Imagen monocromática almacenada al revés.
  1. Imagen monocromática almacenada normalmente.
  2. Imagen con color RGB.
  3. Imagen con paleta de color.
  4. Imagen monocromática para máscara de transparencia.

- DescripcionImagen: 270 (10EH), tipo ASCII.
  Contiene una descripción de la imagen.

- Manufactura: 271 (10FH), tipo ASCII.
  Define el nombre del fabricante de hardware que digitalizó la imagen.

- Modelo: 272 (110H), tipo ASCII.
  Especifica el número de modelo del hardware que digitalizó la imagen.

- DesplazamientoBanda: 273 (111H), tipo entero corto o largo.
  Indica el desplazamiento en el archivo de cada banda de la imagen.

- MuestrasPorPixel: 277 (115H), tipo entero corto.
  Especifica el número de muestras por pixel; el valor por omisión es uno.

- LineasPorBanda: 278 (116H), tipo entero corto o largo.
  Define el número de líneas de la imagen por banda.

- ContadorBytesBanda: 279 (117H), tipo entero corto o largo.
  Define el número de bytes en cada banda de la imagen.

- ResolucionX: 282 (11AH), tipo racional.
  Define el número de pixeles horizontales para la etiqueta *UnidadResolucion*.

- ResolucionY: 283 (11BH), tipo racional.
  Define el número de pixeles verticales para la etiqueta *UnidadResolucion*.

- ConfiguracionPlanar: 284 (11CH), tipo entero corto.
  Define si la imagen está almacenada continuamente o en planos discretos. Los valores reconocidos son:

  1. Los pixeles se almacenan continuamente en un solo plano.
  2. Los pixeles se almacenan en múltiples planos.

- UnidadRespuestaGris: 290 (122H), tipo entero corto.
  Esta etiqueta modifica la curva de respuesta del gris; el valor por omisión es uno. Los valores reconocidos son:

  1. Décimas de unidad.
  2. Centésimas de unidad.
  3. Milésimas de unidad.
  4. Diez milésimas de unidad.
  5. Cien milésimas de unidad.

- CurvaRespuestaGris: 291 (123H), tipo entero corto.
  Define la curva de respuesta al gris, la cual modifica los niveles de gris de una imagen para compensar al hardware de despliegue específico.

- UnidadResolucion: 296 (128H), tipo entero corto.
  Define las unidades de resolución para las etiquetas *ResolucionX* y *ResolucionY*. El valor por omisión es 2; los valores utilizados son:
  1. No se utiliza una unidad de medida absoluta.
  2. La resolución está definida en pulgadas.
  3. La resolución está definida en centímetros.

- CurvaRespuestaColor: 301 (12DH), tipo entero corto.
  Especifica las 3 curvas de respuesta al color; estas curvas permiten al lector TIFF redefinir los colores de la imagen para compensar al hardware de despliegue especifico.

- Software: 305 (131H), tipo ASCII.
  Especifica el nombre del software utilizado para crear la imagen.

- FechaHora: 306 (132H), tipo ASCII.
  Define la fecha y hora en que fue creada la imagen.

- Autor: 315 (13BH), tipo ASCII.
  Especifica quién creó la imagen.

- ComputadoraHost: 316 (13CH), tipo ASCII.
  Define el tipo de computadora utilizada para crear la imagen.

- Predictor: 317 (13DH), tipo entero corto.
  Especifica si existe un proceso de predicción cuando la imagen ha sido comprimida con LZW. El valor por omisión es uno, lo cual indica que no se ha utilizado un predictor.

- MapaColor: 320 (140H), tipo entero corto.
  Define el desplazamiento en el archivo donde se encuentra el mapa de color.

5.5.5 Uso de los archivos TIFF.

A pesar de sus excentricidades y disposiciones discordes, los archivos TIFF son genuinamente útiles; son una forma flexible de entrada sofisticada para una gran variedad de aplicaciones, y en algunos casos, ofrecen facilidades de importación que no poseen otros formatos.

Como se mencionó anteriormente, el software que lee y escribe archivos TIFF lo hace para una variedad finita de aplicaciones, por lo tanto, es común encontrar grupos de aplicaciones que comparten archivos TIFF; esto obliga al desarrollador a crear archivos que sean compatibles con cada uno de estos grupos.

Finalmente, es valioso notar que las especificaciones TIFF sufren de actualizaciones periódicas; actualmente están en vigor las especificaciones TIFF 5.0 y acaban de aparecer las especificaciones TIFF 6.0. Estas últimas son un superconjunto de las primeras por lo que los archivos TIFF 5.0 están también acordes con éstas.

La referencia más completa del formato TIFF son las mismas especificaciones TIFF, disponibles en el *TIFF Developer's Kit* de la compañía *Aldus*.

Capítulo 6

Compresión de Imágenes por Medio de Códigos IFS

Capítulo 6. Compresión de Imágenes por medio de Códigos IFS.

Este capítulo es esencial para el desarrollo de la presente tesis puesto que aquí se presentan los fundamentos teóricos y el trabajo práctico para la compresión de imágenes con técnicas fractales. Estas técnicas han sido investigadas y desarrolladas por la compañía Iterated Systemas Inc., fundada por Michael F. Barnsley; sin embargo, hasta fechas recientes (1993) han sido presentadas al público, pero hay que aclarar que solamente se ha publicado información muy genérica acerca de los esquemas de compresión.

En las siguientes secciones se explica el trabajo teórico y práctico que han desarrollado los investigadores Yuval Fisher, E. W. Jacobs y R. D. Boss, del *Naval Ocean Systems Center*, paralelamente al trabajo de M. F. Barnsley, pero utilizando su teoría básica. La sección 6.5 describe un esquema factible utilizado por Barnsley para la compresión de imágenes de 256 tonos de grises; estos algoritmos se ampliaron para en el desarrollo de la presente tesis para incluir la compresión de imágenes de 24 bits de color, tal como se detalle en el Capítulo 7.

6.1 Introducción a la Compresión con Técnicas Fractales.

El nombre de este esquema de compresión deriva su nombre del hecho de que el método utilizado para codificar imágenes comparte características comunes con los algoritmos de generación de fractales simples. De hecho las imágenes descodificadas por este esquema tienen características fractales; haciendo un acercamiento en algún detalle de la imagen descodificada es posible ver detalles cada vez más finos. Las bases para este método de codificación son completamente diferentes de los métodos tradiciones de compresión: una imagen se particiona en partes que pueden ser aproximadas por otras partes de la misma imagen después de algunas operaciones de escalamiento, translación y rotación. El resultado del proceso de codificación es un conjunto de transformaciones, las cuales, cuando se aplican iteradamente sobre una imagen inicial poseen un punto fijo que se aproxima a la imagen original. La **figura 6.1** muestra el proceso interactivo de descodificación de una imagen en 256 tonos de grises; se comienza con una cuadrícula y se aplican las transformaciones hasta que se alcanza un punto fijo, es decir, la imagen aproximada.

Figura 6.1 Una imagen arbitraria inicial y la primera, segunda y décima iteración.

Una de las primeras aplicaciones de los fractales a la generación de imágenes fue publicada por B. B. Mandelbrot sobre las escenas de montañas fractales generadas por R. F. Voss; éste utilizó algoritmos fractales para generar imágenes realísticas de paisajes terrestres. Poco después, M. F. Barnsley sugirió que dada una imagen, es posible definir algoritmos fractales para generar una aproximación de ella; debido a que que los algoritmos de fractales simples generan típicamente imágenes muy complicadas, Barnsley sugirió que solamente se necesitaba almacenar los parámetros más relevantes del algoritmo y de esta manera concluyó que los sistemas de funciones iteradas conducían a la compresión de imágenes.

La explicación de la teoría general de los códigos IFS fue presentada en la sección 3.2, capítulo 3; sin embargo, en las siguientes secciones se explican nuevamente algunas características de los códigos IFS pero ahora aplicados específicamente a la compresión de imágenes.

6.1.1 Sistemas de Funciones Iteradas IFS.

Barnsley creó el término *Iterated Function Systems* o Sistemas de Funciones Iteradas (abreviado con IFS por sus siglas en inglés) para describir una colección $(F, \delta, w_1, \ldots, w_n)$, donde F es un espacio métrico completo con una métrica δ, y una colección de mapas contractivos $w_1, \ldots, w_n$.

Un mapeo $W: F \rightarrow F$ de un espacio F con métrica δ a si mismo se denomina contractivo si existe un número real positivo $s < 1$, tal que:

$$\delta(w(x), w(y)) < s\,\delta(x, y); \quad \text{para toda } (x, y) \in F.$$

Típicamente, F se toma del espacio de subconjuntos compactos del plano R^2 y δ se toma como la métrica de Hausdorff. Una explicación detallada de estos conceptos cae fuera de esta tesis; por ahora es suficiente pensar en δ como la medida en que difieren dos subconjuntos de un plano.

6.1.1.1 Un Ejemplo Sencillo.

Para comprender un poco mejor el significado de los IFS y su aplicación a la compresión de imágenes, se presenta un ejemplo de como una imagen se define a través de transformaciones de si misma. El concepto principal es que una imagen puede construirse a partir de un conjunto de transformaciones, las cuales requieren menos memoria para almacenarse que la imagen original en sí.

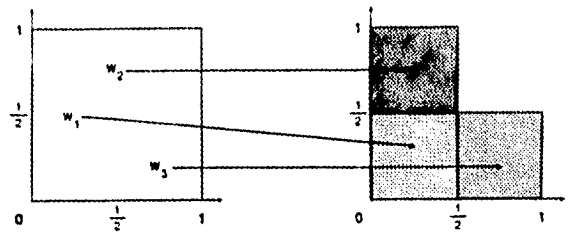


Figura 6.2 Tres transformaciones afines aplicadas a un cuadrado y la figura resultante.

Considérese las tres transformaciones presentadas en la **figura 6.2**; éstas son:

$$w_1 \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$w_2 \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 1/2 \end{bmatrix}$$

$$w_3 \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 1/2 \\ 0 \end{bmatrix}$$

y para cualquier conjunto S se tiene:

$$W(S) = \bigcup_{i=1}^{3} w_i(S)$$

Si se denota con W^{on} la *n-ésima* composición de W con si mismo, y considerando $I = [0, 1]$, se define $A_0 = \{ (x, y) : 0 \le x \le 1, 0 \le y \le 1 \} = I^2$ y $A_n = W(A_{n-1})$, de tal manera que si $n \to \infty$, entonces el conjunto A_n converge (en la métrica de Hausdorff) al conjunto límite A_∞; de hecho, para cualquier conjunto compacto $S \subset R^2$, $W^{on}(S) \to A_\infty$ conforme $n \to \infty$. La **figura 6.3** muestra A_1, A_2, A_3 y A_4; la **figura 6.4** presenta el conjunto límite A_∞, conocido como conjunto de *Sierpinski*.

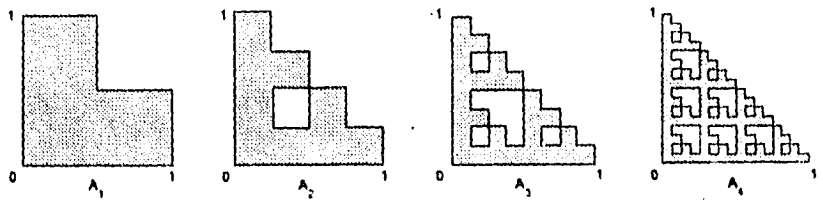


Figura 6.3 $A_1 = W(A_0)$ y las imágenes A_2, A_3 y A_4.

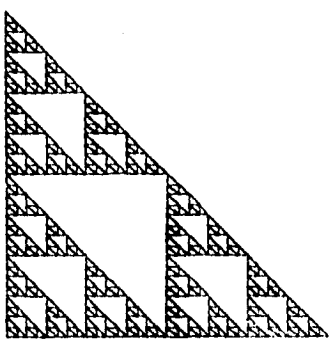


Figura 6.4 El conjunto límite $A_\infty = \lim_{n \to \infty} W^{on}(A_0)$.

El hecho de que todos los conjuntos compactos iniciales converjan a A_∞ cuando se iteran es importante y significa que el conjunto A_∞ se define solamente por los mapas w_i's; además, no es difícil notar que cualquier conjunto inicial que se itere con estos mapas converge a A_∞.

Cada w_i se especifica con 6 valores reales, de manera que, para el ejemplo anterior, se requieren solamente 18 números en punto flotante para definir la imagen; en precisión sencilla, esto es 72 bytes. La memoria requerida para almacenar la imagen del conjunto depende de la resolución; la figura 6.4 requiere 256 x 256 x 1 bit = 8192 bytes de memoria; por lo tanto, la razón de compresión para este ejemplo particular es 113.7:1.

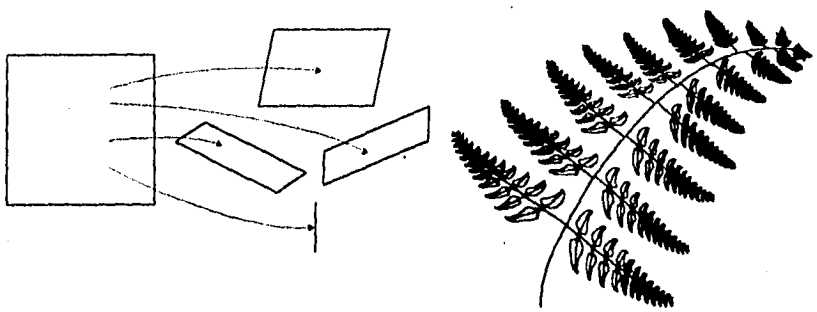


Figura 6.5 Transformaciones para el helecho y su conjunto límite.

Barnsley ha propuesto que los códigos IFS pueden utilizarse para almacenar imágenes arbitrarias en vez de conjuntos de Sierpinskii; por ejemplo, la figura 6.5 muestra 4 transformaciones que generan el conjunto límite, similar a una hoja de helecho.

Es difícil extender este esquema para codificar imágenes más generales; primero, encontrar las transformaciones que codifican a una imagen arbitraria es inherentemente difícil; segundo, ¿Cómo se puede codificar el color o los niveles de grises de una imagen?. Estos puntos se aclaran en las secciones siguientes.

6.1.1.2 Teoría IFS.

Los IFS son transformaciones afines de la forma $w(x) = Ax + b$, con $b \in R^2$ y A es una matriz de 2 x 2 con su normal menor a 1; es decir, $w(x)$ es contractiva. Esta transformación afín puede escalar, rotar y trasladar un conjunto; no es difícil demostrar que una colección de transformaciones afines $w_1, \ldots, w_n$ determinan un mapa

$$W = \bigcup_{i=1}^{n} w_i$$

que es contractivo utilizando la métrica de Hausdorff en el espacio de conjuntos compactos del plano R^2.

El siguiente teorema implica que el mapeo $W: F \rightarrow F$ tiene un punto fijo único, denotado por $|W|$.

Teorema del Punto Fijo del Mapeo Contractivo: Si F es un espacio métrico completo y $W: F \rightarrow F$ es una transformación contractiva, entonces existe un punto fijo único $|W| = lim_{n \rightarrow \infty} W^{on}(A_0)$, para cualquier $A_0 \in F$.

Como el conjunto límite es en realidad un punto fijo,

$$|W| = W(|W|) = w_1(|W|) \cup \ldots \cup w_n(|W|).$$

esto significa que el fractal $|W|$ se construye a partir de partes que son el resultado de escalar, rotar y trasladar al fractal mismo; de esta manera, es posible decir que las partes $w_1(|W|), \ldots, w_n(|W|)$ cubren al fractal $|W|$.

Dado un conjunto arbitrario f, en general no es posible cubrirlo exactamente con un número finito de transformaciones de él mismo; la pregunta obvia es ¿Qué pasa si la cobertura $W(f)$ se aproxima? Un corolario del teorema del punto fijo del mapeo contractivo, al cual Barnsley llama teorema del *collage*[1], define la relación entre el grado con el cual un conjunto puede cubrirse a sí mismo y el grado con el cual el punto fijo resultante se asemeja al conjunto original.

Teorema del *Collage*: Sea $W: F \rightarrow F$ una transformación contractiva con contractividad s, y sea $f \in F$, entonces $\delta(|W|, f) \leq (1 - s)^{-1} \delta(W(f), f)$.

Este teorema dice que entre más cercano esté $W(f)$ al conjunto original f, más cercano estará el punto fijo $|W|$ a f, y esto es especialmente verdadero cuando las transformaciones compuestas W son muy contractivas.

6.1.2 Sistemas de Funciones Iteradas Particionadas PIFS.

Una extensión a los sistemas de funciones iteradas IFS, debida a Barnsley y a Jacquin, son los *Sistemas de Funciones Iteradas Particionadas* o PIFS (siglas en inglés de Partitioned Iterated Function Systems). En los PIFS, los mapas $w_1, \ldots, w_n$ no se aplican a todo el plano sino a dominios restringidos; tales sistemas permiten una codificación simple de formas más generales. Por ejemplo, no es obvio como codificar el moño de la **figura 6.6** utilizando IFS, sin embargo, si la imagen se particiona y después se cubre con PIFS, el proceso se vuelve trivial; en este ejemplo, 4 de las transformaciones se restringen a aplicarse en el lado izquierdo del conjunto y las otras 4 en el lado derecho, de esta manera, se cubre completamente al moño.

[1] Este teorema también fue presentado en la sección 3.2.3.2, capítulo 3.

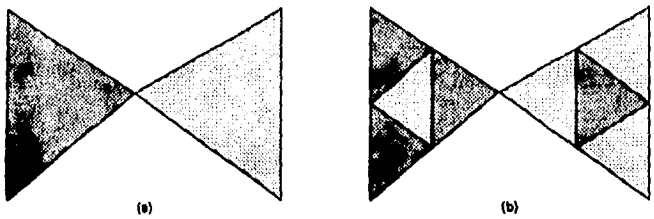


Figura 6.6 **(a) Un moño con una partición indicada por el sombreado, y (b) 8 transformaciones (indicadas por el sombreado) que forman un *collage* exacto.**

En todos los PIFS, una transformación w_i se especifica no solamente por un mapa afín, sino también por el dominio al cual se aplica w_i; el teorema del punto fijo del mapeo contractivo y el teorema del *collage* aún son válidos pero existe una diferencia importante, detallada a continuación. Para un IFS con una w_i expansiva, la iteración W^{on} no converge a un punto fijo único; la parte expansiva causa que el límite del conjunto crezca infinitamente en alguna dirección; esto no es necesariamente cierto para un PIFS, el cual puede contener transformaciones expansivas y aún tener un atractor cerrado; la discusión más detallada de este punto se presenta en la sección 6.2.2.

6.1.2.1 Compresión de Contornos utilizando PIFS.

Barnsley y Jacquin han realizado trabajos de compresión de contornos en los cuales han aplicado PIFS a imágenes de nubes. En este esquema, los contornos de la imagen se particionan en escalas mayores y escalas menores de contornos; se encuentran mapas afines $w_1, \ldots, w_n$, los cuales mapean las particiones en las escalas más grandes $P_1, \ldots, P_m$ a particiones en las escalas pequeñas $p_1, \ldots, p_m$, de tal manera que la distancia entre $w_j(P_j)$ y p_j se minimice. El mapa resultante $W = \cup_i w_i$ tiene un punto fijo que se aproxima a los contornos originales.

En el documento de Barnsley y Jacquin se describen los conceptos involucrados pero no se detalla el método de particionamiento, ni se demuestra un sistema totalmente automatizado para hacerlo. En un trabajo de Yuval Fisher, E. W. Jacobs y R. D. Boss, se aplican estos conceptos a un sistema automatizado de detección y compresión de contornos; en este sistema, los contornos se segmentan y clasifican de acuerdo a su curvatura en diferentes escalas y se efectúa una búsqueda para segmentos más largos (dominios) que mejor cubren a los segmentos más cortos; durante el proceso de codificación, los puntos finales de los dominios disponibles se ajustan de tal manera que el mapa resultante contenga los requerimientos necesarios para un PIFS contractivo. La aplicación de este sistema para contornos de mapas geográficos presenta una compresión moderada con buena fidelidad.

6.1.2.2 Compresión de Imágenes en Escalas de Grises con PIFS.

Los PIFS conducen de una manera natural a la compresión de imágenes en escalas de grises; en este caso, los mapas w_i ya no son transformaciones afines en el plano, sino que operan en R^3, con la nueva dimensión codificando el nivel de gris.

La compañía *Iterated Systems Inc.* utiliza algoritmos propietarios para codificar automáticamente imágenes en escalas de grises; una patente de su esquema utiliza una variación de la idea de los PIFS, es un algoritmo en el cual la imagen se particiona y se encuentra un IFS (en 3 dimensiones) para cada partición. Los detalles del esquema automático de partición se presentan en la sección 6.5

La primera publicación sobre el esquema automatizado PIFS fue presentada por Jacquin, al cual él llamó *Codificación de Imágenes con Transformaciones Iteradas*; en este trabajo se presentó la teoría basada en los operadores de Markov aplicados a espacios métricos, y se demostró la codificación de imágenes en escalas de grises. Las siguientes secciones discuten la teoría y métodos para codificar imágenes en escalas de grises utilizando transformaciones iteradas.

6.2 Teoría de la Codificación de Imágenes Fractales.

La meta de la codificación fractal es almacenar una imagen como un punto fijo de un mapeo $W: F \rightarrow F$, de un espacio métrico completo de imágenes F a si mismo. El espacio F se puede tomar de cualquier modelo razonable, tal como el espacio de todas las funciones medibles en un cuadrado unitario, etc. En este modelo, $f(x, y)$ representa el nivel de gris del punto (x, y) en la imagen, de esta manera, $f \in F$ es una imagen con una resolución infinita.

Para que exista un punto fijo de W, las transformaciones deben ser contractivas, con esto el teorema del punto fijo asegura la convergencia al punto fijo a partir de iteraciones de *cualquier imagen inicial*. Insistir que W^{0m} sea contractiva es menos restrictivo que requerir que W sea contractiva; la meta es construir el mapeo W con un punto fijo muy cercano a una imagen dada que se quiera codificar, de tal forma que W pueda almacenarse compactamente; la cercanía del punto fijo con la imagen se juzga con una métrica adecuada $\delta(f, g)$, para $f, g \in F$.

Una métrica sencilla que puede utilizarse es:

$$\delta_{sup}(f, g) = \sup_{(x, y) \in I^2} |f(x, y) - g(x, y)|$$

Con esta métrica, un mapa es contractivo si se contrae en la dirección z; otra métrica que se puede utilizar es la métrica *rms*:

$$\delta_{rms}(f, g) = \left[\int_{I^2} (f(x, y) - g(x, y))^2 \right]^{1/2}$$

Esta métrica tiene requerimientos de contractividad más complicados aunque es particularmente utilizada puesto que puede calcularse fácilmente utilizando algoritmos de regresión estándar; en el resto del capítulo la métrica *rms* se especificará cuando sea relevante.

6.2.1 Construcción del Mapa W.

Sea F un espacio de todas las funciones medibles $z = f(x, y)$, con $(x, y, f(x, y)) \in I^3$, el mapa W se construye a partir de los mapas locales w_i, definidos de la siguiente manera.

Sea $D_1, \ldots, D_n$ y $R_1, \ldots, R_n$ subconjuntos de I^2, llamados dominios y rangos, respectivamente (aunque no sean exactamente dominios y rangos); sea $v_1, \ldots, v_n : I^3 \rightarrow I^3$ una colección de mapas; se define a w_i con la restricción:

$$w_i = v_i |_{D_i \times I}$$

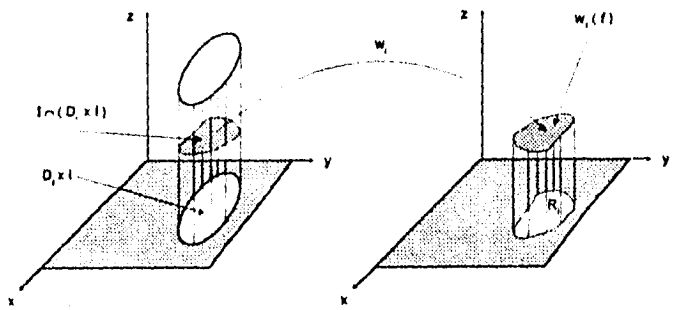


Figura 6.7 Cobertura de la imagen con partes de si misma.

Este conjunto de w_i es el que forma el PIFS; se dice que los mapas w_i forman *mosaicos* sobre I^2 para toda $f \in F$, $\cup_{i=1}^{n} w_i(f) \in F$. Esto significa lo siguiente: para cualquier imagen $f \in F$, cada D_i define una parte de la imagen $f \cap (D_i \times I)$ a la cual w_i está restringida; cuando el mapa w_i se aplica a esta parte, el resultado debe ser una gráfica de una función sobre R_i, e $I = \cup_{i=1}^{n} R_i$; este procedimiento se ilustra en la **figura 6.7**. Finalmente, el mapa W se define como:

$$W = \bigcup_{i=1}^{n} w_i$$

Dada una W, es fácil encontrar la imagen que está codificada; se comienza con cualquier imagen inicial f_0 y se calculan sucesivamente $W(f_0)$, $W(W(f_0))$, hasta que la imagen converja a $|W|$. Lo inverso es considerablemente más difícil: dada una imagen f ¿Cómo se puede encontrar un mapa W tal que $|W| = f$? Todavía no se ha encontrado una solución no trivial a este problema. En vez de eso, se busca una imagen $f' \in F$ tal que $\delta(f', f)$ sea mínima con $f = |W|$. Puesto que $|W| = W(|W|) = \cup_{i=1}^{n} w_i(|W|)$, es razonable buscar dominios $D_1, \ldots, D_n$ y sus transformaciones correspondientes $w_1, \ldots, w_n$, tal que:

$$f \approx W(f) = \bigcup_{i=1}^{n} w_i(f)$$

Esta ecuación dice: cubre a f con partes de si misma; las partes están definidas por D_i y la manera en que estas partes cubren a f está determinada por las w_i's. La igualdad en la ecuación anterior implicaría que $f = |W|$; cubrir a f exactamente con partes de si misma no es fácil, así que se busca la mejor cobertura posible con la esperanza que f y $|W|$ no se verán muy diferentes, es decir $\delta(|W|, f)$ sea pequeña. La base para esto proviene del teorema del *collage*.

De esta manera, el proceso de codificación es: particionar I^2 en un conjunto de rangos R_i; para cada R_i se busca una $D_i \in I^2$ y un mapa $w_i : D_i \times I \rightarrow I^2$, tal que $w_i(f)$ esté lo más cercana posible a $f \cap (R_i \times I)$, es decir:

$$\delta(f \cap (R_i \times I), w_i(f))$$

sea minimizada. El mapa W se especifica por medio de los mapas w_i; estos mapas deben escogerse de tal manera que W o W^{om} sea contractiva. Los métodos específicos de particionamiento se explican en secciones posteriores de este capítulo.

6.2.2 Mapas Eventualmente Contractivos.

Para la gente que conoce la teoría de los códigos IFS puede ser sorprendente que, cuando se construyen las transformaciones w_i, no es necesario imponer ninguna condición de contractividad a las transformaciones individuales; un requerimiento suficiente de contractividad es que W sea *eventualmente contractiva*. Un mapa $w : F \rightarrow F$ es eventualmente contractivo si existe un número entero positivo m, llamado *exponente de contractividad eventual*, tal que W^{om} sea contractiva, pero no viceversa.

A continuación se presenta una breve explicación de cómo una transformación $w : F \rightarrow F$ puede ser eventualmente contractiva pero no contractiva totalmente. El mapa W está compuesto de la unión de los mapas w_i operando en partes desjuntas de la imagen; si las w_i se escogen de tal manera que:

$$| w_i (x, y, z_1) - w_i (x, y, z_2) | < s_i | z_1 - z_2 |$$

y además $(x', y', z') = w_i(x, y, z)$ tiene una x', y' independientes de z, entonces W será δ_{sup} contractiva, si y solo si, cada w_i es δ_{sup} contractiva, es decir, cuando $s_i < 1$ para toda i; si alguna $s_i \geq 1$ entonces W no será δ_{sup} contractiva.

La transformación iterada W^{om} está compuesta de una posición de composiciones de la forma:

$$w_{i1} \circ w_{i2} \circ \ldots \circ w_{im}$$

Puesto que el producto de las contractividades define la contractividad de las composiciones, éstas pueden ser contractivas si cada una de ellas contiene suficientes w_i contractivas. De esta forma, W es eventualmente contractiva (en la métrica *sup*) si contiene suficiente *mezcla* tal que las w_i contractivas dominen a las expansivas; en la práctica, esta condición es relativamente simple de verificar, tal como se describe en la sección 6.3.3.

El teorema del *collage* puede generalizarse para el caso de mapas eventualmente contractivos. Sea s la contractividad de W, y sea σ la contractividad de W^{om}; se asume que W es contractiva ($\sigma < 1$) aunque s puede ser mayor que 1.

Teorema Generalizado del *Collage*: Para cada $f \in F$ y $w : F \to F$ siendo eventualmente contractivo, se tiene que:

$$\delta(|W|, f) \leq \frac{1}{1-\sigma} \frac{1-s^m}{1-s} \delta(W(f), f)$$

Con alguna notación extra es posible mejorar la ecuación anterior, sin embargo, esto no se hace por la siguiente razón: ambos, el teorema del *collage* y el teorema generalizado del *collage*, sirven solo como referencia puesto que ninguno de ellos proporciona fronteras útiles; los resultados empíricos han confirmado esto.

Es importante notar que un mapeo W que es contractivo en la métrica δ_{sup} puede ser eventualmente contractivo en la métrica δ_{rms}. A diferencia de la métrica *sup*, la condición de $s_i < 1$ no es suficiente para asegurar la contractividad para la métrica *rms*; sin embargo, para $s_i \geq 1$ los mapas aún pueden ser eventualmente contractivos.

6.3 Métodos de Implementación.

Esta sección contiene un ejemplo sencillo que ilustra el proceso de codificación descrito la sección 6.2, se presenta una discusión de las consideraciones relevantes en la codificación con buena compresión y fidelidad y se detallan algunas implementaciones específicas.

6.3.1 Compresión con Particiones Sencillas.

Considérese la imagen f que se muestra en la **figura 6.8a**, como una función entera valuada en la latice [0, 255] x [0, 255] con valores en [0, 255]; es una imagen de 256 x 256 pixeles con 256 niveles de grises. Sea $R = \{R_1, \ldots, R_{1024}\}$ el conjunto de los 1024 rectángulos de 8 x 8 pixeles los cuales no se intersectan; sea D la colección de todos los rectángulos de 16 x 16 pixeles de la imagen; para cada R_i se encuentra el mejor dominio en D, denotado D_i, y una w_i de la forma:

$$w_i \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_i & b_i & 0 \\ c_i & d_i & 0 \\ 0 & 0 & s_i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \\ o_i \end{bmatrix}$$

tal que $w_i(f)$ tome valores en R_i y que se minimice la ecuación $\delta(f \cap (R_i \times I), w_i(f))$.

La primera condición determina a_i, b_i, c_i, d_i, e_i y f_i utilizando la simetría del cuadrado, y la segunda condición determina s_i, o_i y la operación de simetría. En este ejemplo, D_i y w_i se escogieron buscando todas las opciones posibles; cada D_i se probó en 8 posibles orientaciones (correspondientes a los elementos del grupo simétrico del cuadrado). Se utilizó la métrica *rms* para minimizar el error, y s_i y o_i se calcularon con la regresión de los mínimos cuadrados. Para asegurar que exista $lim_{n\to\infty} W^{\circ n}$, W debe ser contractiva o eventualmente contractiva.

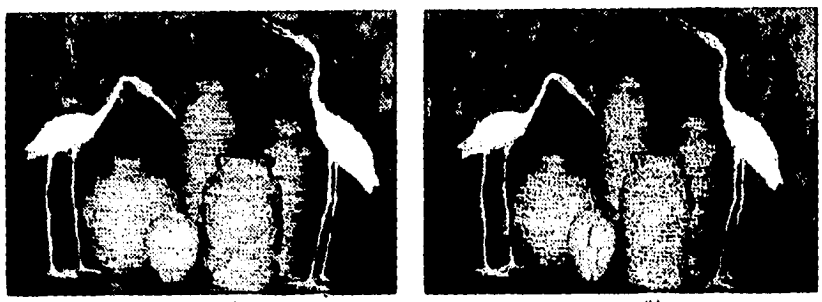
La s_i más grande permitida se denota como:

$$s_{max} = \max(s_i).$$

El resultado de aplicar el algoritmo con $s_{max} = 1.5$ da como resultado la $|W|$ de la figura 6.8b; la siguiente tabla resume algunos resultados de este ejemplo, todos con compresión 16:1.

s_{max}	$\delta_{rms}(w(f), f)$	$\delta_{rms}(\|W\|, f)$
0.7	20.04	21.48
0.9	19.77	21.16
1.2	19.42	21.16
1.5	19.42	20.97

El teorema del *collage* y el teorema generalizado del *collage* dicen que, a menos que $\delta(w(f), f)$ se mejore marcadamente, puede ser desventajoso permitir mapas menos contractivos; sin embargo, los resultados de la tabla anterior muestran que (para la métrica *rms*) aún cuando la cobertura sea más exacta se tiene solamente una ligera mejora en la fidelidad de la imagen. Se puede apreciar también que las fronteras dadas por ambos teorema son mucho mas grandes que el valor máximo posible de 255 (para 256 niveles de grises). De estos resultados es evidente que las fronteras del teorema del *collage* deben considerarse más bien como una motivación que valores útiles.



(a) (b)

Figura 6.8 (a) Imagen original, e (b) Imagen codificada con $s_{max} = 1.5$

6.3.2 Consideraciones en la Compresión de Imágenes.

Claramente, la selección de dominios, rangos y mapas w_i es una de las partes más importantes en un esquema de codificación fractal automatizado; la notación D y R se utiliza para denotar el conjunto de todos los dominios y rangos potenciales, respectivamente.

Puesto que la meta de la codificación es la compresión, un segundo aspecto importante es la especificación compacta del mapa w_i; para limitar la memoria, solamente se utiliza una w_i de la forma descrita anteriormente (8 números reales). Los valores s_i y o_i se calculan utilizando la regresión de los mínimos cuadrados para minimizar el error en la métrica *rms* y se almacenan utilizando un número fijo de bits; podría ser posible calcular los valores óptimos de s_i y o_i y discretizarlos para su almacenamiento, sin embargo, se obtiene una mejora significativa en la fidelidad si solamente se utilizan valores discretizados de s_i y o_i cuando se calcula el error durante la codificación. Puesto que especificar w_i requiere especificar un dominio D_i, éstos están restringidos a ser geométricamente simples; en la implementación descrita más adelante, D y R siempre son una colección de rectángulos.

Otro aspecto importante es el tiempo de codificación, el cual puede reducirse significativamente utilizando un esquema de clasificación de rangos y dominios; tanto los dominios como los rangos se clasifican utilizando algún criterio tal como la naturaleza de los bordes de la imagen, la brillantez, etc. Se tiene una reducción importante del tiempo de codificación si solamente se utilizan dominios de la misma clase que el rango al que debe cubrir.

En el ejemplo de la sección 6.3.1 el número de transformaciones es fijo; en contraste, en los algoritmos descritos a continuación se utiliza un tamaño de rango que varía dependiendo de la complejidad local de la imagen. Para una imagen dada, más transformaciones conducen a una mejor fidelidad pero con pobre compresión; este compromiso entre la compresión y la fidelidad conduce a dos diferentes algoritmos para codificar la imagen f: uno considerando la fidelidad y otro la compresión. Estos procedimientos se presentan en las tablas siguientes; en el pseudo-código, Tamaño(R_i) se refiere a la longitud del lado del rango, y en el caso de rectángulos, indica la longitud del lado más largo.

- Escoger un nivel de tolerancia e_c.
- Hacer $R_1 = I^2$ y marcarlo como no cubierto.
- Mientras haya rangos sin cubrir haz {
  - De entre todos los dominios posibles D, encontrar el dominio D_i y su w_i correspondiente que mejor cubra a R_i (es decir, que minimice el error).
  - Si $\delta(f \cap (R_i \times I), w_i(f)) < e_c$ o Tamaño(R_i) $\leq r_{min}$ entonces
    - Marcar R_i como cubierto y almacenar la transformación w_i.
  - Si no
    - Particionar R_i en rangos más pequeños los cuales se marcan como no cubiertos, y se borra R_i de la lista de rangos sin cubrir.

Fin de Mientras.

Pseudo-código para una fidelidad e_c.

- Escoger un número de rangos N_r.
- Inicializar una lista que contenga a $R_1 = I^2$ y marcarlo como cubierto.
- Mientras haya menos de N_r rangos en la lista haz {
  - De la lista de rangos encontrar el rango R_j de Tamaño(R_j) > r_{min} con el más largo $\delta(f \cap (R_i \times I), w_i(f))$, es decir, el peor cubierto.
  - Particionar R_j en rangos más pequeños, los cuales se agregan a la lista de rangos y se marcan como no cubiertos.
  - Borrar R_j, w_j y D_j de la lista.
  - Para cada rango no cubierto en la lista, encontrar y almacenar el dominio $D_j \in D$ y el mapa w_j que lo cubre mejor.

  Fin de Mientras.
- Escribir todas las w_j de la lista.

Pseudo-código para una compresión con N_r transformaciones.

Una generalización del ejemplo de la sección 6.3.1 (el cual puede implementarse siguiendo los pseudo-códigos) es seleccionar los rangos de un particionamiento en árbol cuádruple de la imagen. Una partición de árbol cuádruple es aquella en la cual cada cuadrado (comenzando con I^2) se subdivide en 4 pequeños cuadrados con un cuarto del área original.

Una generalización del esquema del árbol cuádruple, llamada *particionamiento H-V*, es particionar recursivamente la imagen a lo largo de líneas horizontales o verticales; los rangos y dominios se toman de la misma partición, la cual se escoge de manera que los rangos y dominios compartan propiedades que permiten a los dominios cubrir bien a los rangos. Los esquemas como este permiten que el proceso de codificación sea particularmente adaptativo a patrones y contenido de cada imagen individual; la **figura 9.a** muestra un particionamiento típico de una imagen en un árbol cuádruple y la **figura 9.b** presenta un particionamiento H-V.

Mientras que variar el número de rangos afecta directamente la compresión, afectar el número de posibles dominios tiene un efecto más complicado. Podría parecer que utilizando más dominios la compresión disminuiría puesto que se requiere más información para especificar un dominio particular utilizado, y también podría parecer que el tiempo de codificación aumentaría puesto que se deben buscar más dominios; sin embargo, con más dominios, un rango que hubiera sido particionado puede ahora cubrirse, de esta manera, se necesita cubrir menos rangos; similarmente, utilizando más dominios no garantiza una mejor fidelidad debido a que un rango que hubiera sido particionado se cubre ahora más pobremente.

Las secciones 6.3.4 y 6.3.5 contienen notas específicas de implementación para el esquema de codificación de imágenes basado en transformaciones iteradas.

Figura 6.9 (a) Una partición en árbol cuádruple, y (b) una partición H-V.

6.3.3 Selección de las w_i's.

La selección de las transformaciones w_i's afecta el hecho de que W sea contractiva o no; en realidad, la selección de la métrica con la cual se mide la contractividad es importante también. Para δ_{sup}, la cual requiere que w_i sea contractiva en la dirección z, es decir, $s_i < 1$ es suficiente para asegurar que W sea contractiva; si alguna w_i está expandiéndose en la dirección z entonces W está también expandiéndose pero aún puede ser eventualmente contractiva.

Un método efectivo para verificar la contractividad eventual en la métrica δ_{sup} es comenzar con una imagen f tal que $f(x, y) = 1$; se define w_i' como w_i con $o_i = 0$, y $W' = \cup_i w_i'$, se tiene entonces que:

$$\sup_{(x,y)\in I^2} \{ W^{\circ n}(f)(x, y) \}$$

es la contractiva σ de $W^{\circ n}$. Para verificar si un mapa es eventualmente contractivo se itera W' hasta que $\sigma < 1$; un procedimiento similar para determinar la contractividad eventual en la métrica *rms* todavía no está desarrollado. Estas pruebas solamente determinan la contractividad eventual después de que se ha codificado una imagen; en la práctica, es necesario tener un número suficiente de mapas contractivos de manera que durante la codificación exista un alta probabilidad de tener un mapeo eventualmente contractivo.

Las w_i's pueden ser contractivas en las direcciones x y y aunque esto no es necesario cuando $s_{max} < 1$; la contractividad en las direcciones x y y incrementa la *mezcla* de las transformaciones, promoviendo de esta manera los mapas eventualmente contractivos. La cuestión de la contractividad de x y y cuando s_{max} no está restringida aún no está resuelta totalmente.

Finalmente, es importante la manera en que se transforman las w_i's al dominio de los pixeles; un pixel puede ser, ya sea el promedio de todos los pixeles transformados que mapean a dicho pixel o bien, puede escogerse como uno de los pixeles del dominio transformado; la primera opción conduce a una fidelidad mejor con un costo computacional extra.

6.3.4 Particiones con Árboles Cuádruples.

La partición de una imagen en árbol cuádruple se presentó en el capítulo 4; en esta sección no se pretende describir nuevamente el proceso de particionamiento, sino describir los posibles esquemas para codificación de imágenes con transformaciones iteradas, basados en el particionamiento con árboles cuádruples. A continuación se presentan algunos de dichos esquemas de compresión.

En un esquema de compresión de imágenes, ésta se puede dividir en cuatro cuadrantes los cuales se codifican independientemente; si una imagen posee semejanza local con si misma en sentido general, entonces la calidad de la codificación no debe ser significativamente degradada limitando a que D se escoja de un cuadrante de la imagen; adicionalmente, la cardinalidad reducida de D decrementa el tiempo de codificación. Se puede utilizar una partición en dos niveles para tomar ventaja de las características locales de la imagen.

Con este esquema se han codificado imágenes de 256 x 256 pixeles, las cuales fueron divididas en subimágenes de 128 x 128 pixeles, cada una de ellas se codificaron independientemente con dominios de 16 x 16 y 8 x 8 para cubrir rangos de 8 x 8 y 4 x 4, respectivamente. Se calculó el error en los 4 cuadrantes de cada uno de los rangos de 8 x 8 para encontrar un criterio de error predeterminado basado en la métrica *sup* y *rms*; si el criterio determinado no se encontraba en un cuadrante, éste se codificaba como un rango de 4 x 4. Los dominios y rangos se clasificaron de tal manera que los dominios cubrían solamente a los rangos de la misma clase; esto permitió que los valores del escalamiento s y del desplazamiento o fueron dependientes de la clase, con al menos 8 valores para s y 128 valores para o. Este esquema de codificación utiliza un gran número de transformaciones, cada una de ellas con pequeños requerimientos de almacenamiento.

Otro esquema de compresión de imágenes con funciones iteradas se ha utilizado para comprimir imágenes de 256 tonos de grises; en este esquema se utilizan pocas transformaciones pero cada una de ellas requiere gran cantidad de espacio para almacenarse. El algoritmo está configurado con diferentes parámetros de compresión de tal manera que éstos se puedan variar para permitir un estudio general del desempeño de su implementación.

La codificación se efectúa con 4 diferentes opciones D^1, $D^{1/2}$, $D^{1/4}$ y $D^{1/8}$ para el conjunto D; estos dominios tienen un tamaño variable y están restringidos a tener sus esquinas en una latice con su espaciamiento horizontal y vertical fijo. Dos de los conjuntos de dominios tienen dominios rotados 45 grados; la combinación de estos dominios, espaciamientos y orientaciones para el conjunto de dominios se resume en la siguiente tabla.

Conjunto	Tamaño	Espaciamiento de la latice	¿Incluye rotación de 45 grados?
D^1	8, 16, 32, 64	Tamaño/2	si
$D^{1/2}$	8, 16, 32, 64	Tamaño/2	no
$D^{1/4}$	8, 16, 32, 64	Tamaño	si
$D^{1/8}$	8, 16, 32, 64	Tamaño	no

El proceso de codificación sigue los algoritmos presentados previamente; sin embargo, se utiliza un esquema para clasificar todos los dominios posibles en 72 clases; las clases se definen de acuerdo a las características del cuadrante tales como intensidad, brillo, etc. Escoger un esquema de clasificación que ordene los cuadrantes de un dominio de acuerdo a su brillo determina una operación simétrica la cual limita la búsqueda de una buena cobertura de un rango; esto se debe que solamente se verifica una de 8 posibles orientaciones.

Cuando se está cubriendo un rango, se toman los dominios (los cuales tienen sus lados con el doble de longitud que la de los rangos) de un número fijo de clases, n_c; estas clases se escogieron para que fueran lo más similar a la clasificación del rango. El número de bits utilizado para discretizar y almacenar s_i y o_i se ajusta de acuerdo al valor máximo permitido para s_i; estos parámetros se denotan con n_s, n_o y s_{max}, respectivamente. Los valores de s_i se restringieron a estar en el rango $s_{max} \geq |s_i| \geq (s_{max}/10)$ y $s_i = 0$. Los otros parámetros son D, e_c, r_{min} y el número de clases de dominios a buscar, n_c.

6.3.5 Particiones Horizontal-Vertical.

El particionamiento H-V deriva su nombre de las particiones horizontales y verticales que hace; a diferencia del particionamiento en árbol cuádruple, el cual divide un cuadrado en una manera fija, el particionamiento H-V permite un particionamiento variable en rectángulos. Cada paso del particionamiento consiste en subdividir un rectángulo (inicialmente la imagen completa) en dos rectángulos, ya sean horizontales o verticales. Cuando se particiona un rectángulo que contiene un borde horizontal o vertical, la partición ocurre a lo largo de dicho borde; si un rectángulo contiene otros bordes, la partición se hace de tal manera que el borde principal intersecte uno de los rectángulos generados en su esquina, con el otro rectángulo que no contenga a dicho borde; esto produce como resultado rectángulos que no contienen bordes o tienen bordes que corren diagonalmente.

El conjunto *D* es dinámico, pues cambia para cada paso del algoritmo, y consiste de todos los rectángulos previamente particionados. Puesto que el particionamiento intenta crear rectángulos con bordes que corren diagonalmente, es razonables esperar que la selección de *D* trabaje bien; en realidad, el número de dominios buscados cuando se utiliza la partición H-V es menor que el número de dominios usados en la partición con árbol cuádruple (por un factor de 60), pero los resultados de la codificación son comparables.

Para encontrar una partición horizontal, los valores de los pixeles del rectángulo se suman verticalmente; los bordes se detectan calculando las diferencias sucesivas de estas sumas. Las diferencias se miden con una función de tope de manera que las particiones se escogen preferentemente cercanas al centro del rectángulo; esto es necesario para prevenir muchas particiones estrechas. Las particiones verticales se encuentran de manera análoga.

Almacenar la partición en forma compacta es simple pero requiere más memoria que la partición en árbol cuádruple. La partición se almacena como una secuencia de niveles que denotan la orientación de la partición, y el desplazamiento que determina la posición de la partición en cada paso; conforme los rectángulos se van haciendo más pequeños, se requieren menos bits para almacenar el desplazamiento, de manera que el costo de la memoria decrece conforme se añaden más rectángulos; además, si se tienen más rectángulos también se incrementan los conjuntos de dominios. Por estas dos razones, el particionamiento H-V tiende a trabajar mejor a razones bajas de compresión.

Mapear los dominios en rangos es más difícil en este esquema que con el particionamiento en árbol cuádruple; en este último, se escoge que el tamaño de los dominios sea más grande en un factor entero que el tamaño de los rangos, de tal forma que la transformación promedio de los pixeles del dominio a los pixeles del rango es fácil. En el esquema de particionamiento H-V, el cálculo de la transformación promedio es computacionalmente intensivo; una alternativa es simplemente escoger un pixel representativo del dominio para el pixel del rango; otra alternativa que conduce a un mejor resultado con un costo computacional extra, es promediar los pixeles del dominio que mapean totalmente a un pixel del rango e ignorar a aquellos que contribuyen con una fracción de su valor.

6.4 Otros Aspectos Relevantes en la Compresión Fractal.

Finalmente, en esta sección se presentan algunos aspectos importantes en la compresión de imágenes por medio de transformaciones iteradas que se deben remarcar; primeramente, se describen algunos trabajos adicionales que se están efectuando en el campo de la compresión de imágenes a color con técnicas fractales, y a continuación se presentan algunas conclusiones generales sobre estas técnicas de compresión.

6.4.1 Trabajos Adicionales en este Campo.

Esta sección contiene un breve resumen de otros trabajos sobre la compresión fractal; la discusión se centra en la codificación de imágenes a color, post-procesamiento y eliminación de artifactos, métodos alternativos de codificación y descodificación, y diferentes esquemas de particionamiento.

Un método estándar para codificar imágenes de 24 bits por color es transformar la imagen de la representación RGB a la representación YIQ de luminiscencia y cromaticidad; la luminiscencia se comprime como se describió a lo largo de este capítulo, y la cromaticidad se transforma espacialmente y después se comprime. Utilizando este método, los resultados preliminares muestran que el desempeño es consistente (relativo a los métodos ADCT[2]) y similar a los resultados de la compresión de imágenes en niveles de grises. Otro método intenta codificar una de las señales R, G o B y después codificar los otros dos componentes restantes utilizando (cuando es posible) una transformación parecida a la utilizada en el primer componente; para algunas imágenes, este método trabaja bien, pero en general los resultados no son tan buenos como el método de transformación a YIQ.

Al igual que con otras técnicas de compresión, se puede efectuar post-procesamiento para remover artifactos en la imagen descodificada; algunos métodos de post-procesamiento conocidos son el *dither*, la detección de bordes, etc.

Un algoritmo alternativo para codificar imágenes con pocos artifactos, pero que aún no ha sido evaluado totalmente, cubre cada rango con una combinación lineal de los dominios; esto incrementa los conjuntos de dominios, el tiempo de búsqueda para encontrar una cobertura óptima y los requerimientos de almacenamiento para las transformaciones. Otros métodos alternativos se centran en disminuir el ya rápido tiempo de descodificación; una opción es descodificar la imagen a un tamaño pequeño hasta que el punto fijo esté bien aproximado, por lo tanto, solamente se requiere un par de iteraciones de W para alcanzar el punto fijo de la imagen total.

Finalmente, la investigación actual se concentra en encontrar esquemas alternativos de particionamiento. Por ejemplo, un esquema basado en descomposiciones triangulares es bastante prometedor debido a que sus requerimientos de memoria son los mismos que para el particionamiento H-V, pero los dominios y rangos son más versátiles, permitiendo a una w_i mapear con una posición arbitraria en vez de rotaciones de 90 grados.

[2] ADCT *Analogic/Digital Compression Techniques* o Técnicas de Compresión Analógica/Digital.

6.4.2 Conclusiones sobre la Compresión Fractal.

Ajustando los parámetros apropiados (n_s, n_o, s_{max}, D, R, e_c, r_{min} y n_c (descritos en la sección 6.3.4), se logran codificaciones con desempeñó, compresión y fidelidad razonables en un gran número de imágenes. Por otro lado, permitir transformaciones no contractivas trae consigo una mejora en la codificación; esto es contrario a lo que sugiere el teorema del *collage*, en el cual la frontera predicha es mucha más grande que la encontrada empíricamente durante el proceso de codificación de imágenes. Por este motivo, el teorema del *collage* y el teorema generalizado del *collage* debe utilizarse como un acercamiento a la frontera, no como un valor práctico.

En relación a otros métodos de compresión de imágenes, hablando estrictamente en términos de la relación señal a ruido de la imagen original y la imagen codificada, este método de compresión fractal no tiene el desempeño de un método ADCT, tal como el estándar JPEG[3], pero es similar.

A pesar de que la codificación con transformaciones iteradas es computacionalmente intensiva, el proceso de descodificación es computacionalmente sencillo comparado con los métodos ADCT. Debido a que el esquema de transformaciones iteradas se basa en una codificación referencial a si misma de la imagen, en oposición a las tablas de códigos de los métodos ADCT, es muy adecuada para codificar imágenes más generales que los algoritmos de cuantización vectorial; de esta manera, las transformaciones iteradas puede aplicarse mejor en sistemas que necesitan descodificar rápidamente un gran número de imágenes pre-codificadas (posiblemente proporcionadas por una base de datos central). Las futuras mejoras a este método pueden conducir a una aplicación más general.

Durante la implementación del método de transformaciones iteradas, la selección del método y los parámetros de escalamiento y desplazamiento son importantes, pero la selección de las particiones R y D es menos obvia. Los esquemas de particionamiento muy simples trabajan por la misma razón que los algoritmos de cuantización vectorial, es decir, la información en cada transformación es relativamente pequeña, de tal manera que se pueden utilizar un gran número de iteraciones manteniendo una compresión razonable. Las particiones más elaboradas, las cuales explotan mejor las propiedades de semejanza con si misma de la imagen, deben almacenarse como parte de la codificación, y por lo tanto, se incrementa su propia complejidad. Los datos indican que las mejoras en la compresión y fidelidad de la imagen son pequeñas cuando se generaliza de las particiones sencillas a un esquema más elaborado, tal como la partición H-V. Cuando se está diseñando un codificador con transformaciones iteradas, la meta podría ser obtener una complejidad óptima del algoritmo.

[3] JPEG *Joint Photographic Expert Group* o Unión de Grupos Fotográficos de Expertos. Es un comité de estandarización de la CCITT para la compresión de imágenes fotográficas.

6.6 Implementación Práctica del Sistema.

La implementación práctica del sistema para la compresión fractal de imágenes utiliza la teoría presentada anteriormente, pero no utiliza todos los parámetros descritos debido a la complejidad de los algoritmos. En las siguientes páginas se describen los conceptos y métodos seguidos para el desarrollo del sistema, objetivo de esta tesis.

6.5.1 Construcción de los Mapas IFS.

En una aplicación práctica, las transformaciones pueden definirse de la forma

$$w(\cdot) = 0.5A + t$$

donde A es una de las simetrías dadas en la tabla siguiente:

Simetría	Matriz	Descripción
0	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	Identidad
1	$\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$	Reflexión en el eje Y
2	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	Reflexión en el eje X
3	$\begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$	Rotación de 180°
4	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	Reflexión en la línea y=x
5	$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$	Rotación de 90°
6	$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$	Rotación de 270°
7	$\begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix}$	Reflexión en la línea y=-x

D_i es un pequeño bloque, del doble de tamaño de R_i, tal como se ilustra en la **Figura 6.10**. Si cada bloque R_i es del mismo tamaño para $i=1, 2, ..., N$, entonces el IFS particionado (PIFS) se especifica completamente utilizando las coordenadas x y y para cada i (R_x, R_y) de la esquina inferior izquierda de R_i, las coordenadas (D_x, D_y) de D_i y un número entero que indica la selección de una simetría de la tabla anterior. El resultado, al cual se le conoce como un IFS particionado, puede expresarse como se muestra en la **Figura 6.11**, en la cual también se muestra una imagen, como se codifica y cual es su atractor.

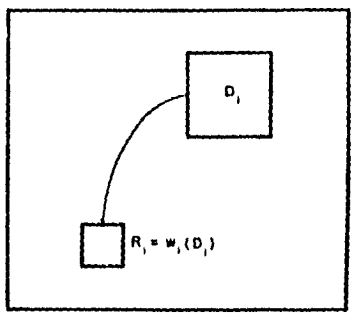
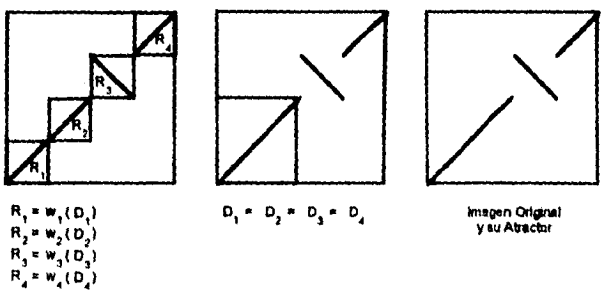


Figura 6.10 Un Dominio D_i y su Imagen $R_i = w_i(D_i)$, presentada como un PIFS.



Mapa	R_x	R_y	D_x	D_y	Simetría
1	0	0	0	0	0
2	4	4	0	0	0
3	8	8	0	0	2
4	12	12	0	0	0

Figura 6.11 Una Imagen de Prueba, su PIFS asociado que muestra los bloques D_i y R_i, y su atractor. También se muestra el código PIFS.

6.5.2 Mapas IFS para Imágenes en Tonos de Grises.

Para obtener un esquema de compresión para imágenes en tonos de grises se utilizan transformaciones de la forma

$$w_i \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_i & b_i & 0 \\ c_i & d_i & 0 \\ 0 & 0 & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} D_x \\ D_y \\ Q_i \end{bmatrix}$$

donde los coefecientes a_i, b_i, c_i y d_i son tales que la transformación actúa en el plano xy de acuerdo a la tabla de simetrías presentada en la sección anterior, con un factor de contractividad de 0.5. El coeficiente P es un entero positivo fijo tal que $0 < P \leq s$; y

$$v_i(z) = Pz + Q_i$$

Para presentar un ejemplo de como trabaja este sistema de compresión fractal, considérese el conjunto de dominios de la **Figura 6.12** y el particionamiento en rangos de la imagen de la **Figura 6.13**. Con estos elementos, un mapa IFS se especifica completamente definiendo los números D_x y D_y, un valor para Q y una simetría correspondiente a cada uno de los rangos. Esta idea se ilustra en la **Figura 6.14**.

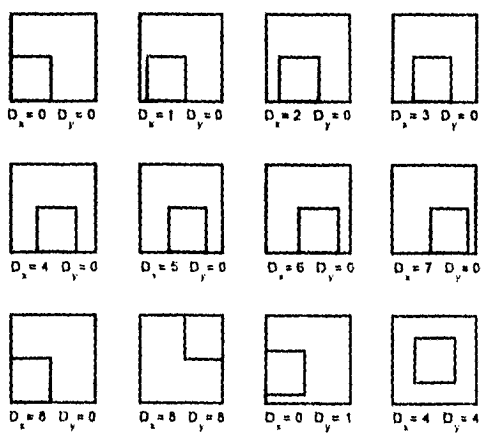


Figura 6.12 **Convención de Numeración para el Conjunto de Dominios a Analizar.**

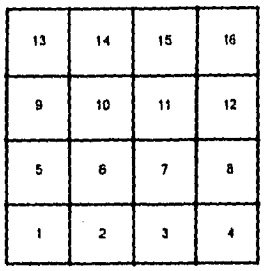


Figura 6.13 **Convención de Numeración para el Conjunto de Rangos a Cubrir por los Dominios.**

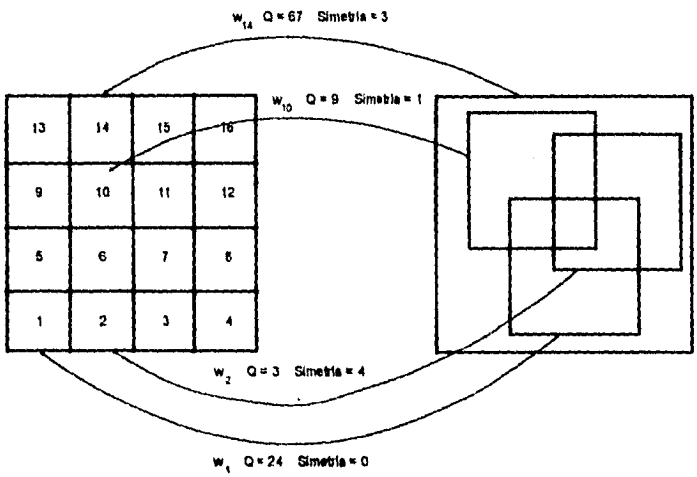


Figura 6.14 **Asociación de algunos Dominios con sus Rangos por medio de un Mapa IFS.**

Dos ejemplos de los mapas IFS definidos se presentan en la **Figura 6.15** y **Figura 6.16**; se muestran los atractores y las imágenes originales; para cada caso, la resolución es de 32x32 pixeles y 256 tonos de grises.

Mapa	D_x	D_y	Simetría	Q
1	0	0	0	0
2	0	0	0	0
3	1	0	0	1
4	7	0	0	35
5	0	0	0	0
6	1	0	0	1
7	7	0	0	35
8	1	0	3	1
9	0	0	0	0
10	7	0	0	35
11	1	0	3	1
12	0	0	0	0
13	7	0	0	35
14	1	0	3	1
15	0	0	0	0
16	0	0	0	0

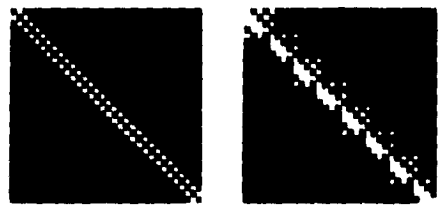


Figura 6.15 Imagen de Entrada y Salida para el Mapa IFS mostrado.

Mapa	D_x	D_y	Simetría	Q
1	0	0	0	0
2	0	0	0	0
3	1	0	0	1
4	7	0	0	35
5	0	0	0	0
6	1	0	0	1
7	7	0	0	35
8	1	0	3	1
9	0	0	0	0
10	7	0	0	35
11	1	0	3	1
12	0	0	0	0
13	7	0	0	35
14	1	0	3	1
15	0	0	0	0
16	0	0	0	0

Figura 6.16 Imagen de Entrada y Salida para el Mapa IFS mostrado.

6.5.3 Parámetros Controlados para la Compresión Fractal.

Debido a que buscar rangos y dominios en imágenes grandes es muy costoso en tiempo de cómputo, es preferible dividir primero la imagen en n subimágenes y tratar independientemente a cada una ellas como una subimagen; este es el primer parámetro a controlar.

El segundo parámetro es el tamaño del rango para particionar cada subimagen; dicho tamaño debe ser múltiplo del tamaño de la subimagen. Derivado de este parámetro se define el número de subimágenes en que se particiona la imagen; la fórmula es:

Número Subimágenes = (Ancho Imagen/Ancho Subimagen) *
(Alto Imagen/Alto Subimagen)

El tercer parámetro es el incremento en el dominio; esto se refiere a la manera en que se van a ir buscando los dominios; por ejemplo, si el incremento es 1 quiere decir que el primer dominio es $D_x = 0$, $D_y = 0$, el segundo es $D_x = 1$, $D_y = 0$, y así hasta D_x = Ancho Submagen - Ancho Dominio, $D_y = 0$; después se incrementa en 1 la coordenada y, $D_x = 0$, $D_y = 1$, el segundo es $D_x = 1$, $D_y = 1$, y así hasta D_x = Ancho Subimagen - Ancho Dominio y D_y = Alto Subimagen - Alto Dominio (es necesario aclarar que el tamaño del dominio es el mismo que el del rango). De esta manera, la fórmula para calcular el número de dominios a evaluar es:

Dominios en X = (Ancho Subimagen/Ancho Dominio - 1)*
(Ancho Dominio/Incremento) + 1
Dominios en Y = (Alto Subimagen/Alto Dominio - 1)*
(Alto Dominio/Incremento) + 1

No. de Dominios = Dominios en X * Dominios en Y

El cuarto parámetro es el número de simetrías a buscar en el dominio para una mejor cobertura del rango; el parámetro puede tomar los valores del 1 al 8.

Para entender mejor estos parámetros se tiene el siguiente ejemplo. Si se tiene una imagen de 128x128 pixeles, el tamaño de la subimagen es 32x32, el lado del domino es 4 y el incremento es 2, los cálculos son:

No. de Subimágenes = (128/32)*(128/32) = 16
No. de Dominios por Subimagen = ((32/4 - 1)*(4/2) + 1)*
((32/4 - 1)*(4/2) + 1) = 225
No. Total de Dominios = 16 * 225 = 3600

Para consultar estos parámetros del archivo fractal generado, se utiliza la opción de **Archivo->Detalles**, tal como se describe en la sección 7.3.4.4.

Parte III

Sistema Propuesto

Capítulo 7

Sistema para la Compresión de Imágenes con Técnicas Fractales

Capítulo 7. Sistema para la Compresión de Imágenes con Técnicas Fractales.

Una vez que se han descrito los elementos teóricos y se han dado ejemplos aplicativos de ellos podemos pasar a la descripción del sistema propuesto. En los siguientes apartados se describen los elementos que conforman al sistema desarrollado para la compresión de imágenes con técnicas fractales; se definen sus alcances, se da una descripción técnica y una guía de usuario, y se presentan los resultados obtenidos para su comparación con otras técnicas.

7.1 Alcances del Sistema.

El sistema para la compresión de imágenes desarrollado tiene como objetivo importar imágenes en los formatos comerciales BMP, PCX, GIF, TIFF y Targa y comprimirlas utilizando códigos IFS particionados, y viceversa, leer imágenes comprimidas fractalmente y exportarlas a los formatos ya mencionados.

El sistema es capaz de leer la mayoría de las variantes de estos formatos; por ejemplo, es posible leer imágenes monocromáticas, de 256 colores, de 256 tonos de grises y de 16 millones de colores del formato BMP. Esto se presentan más adelante en la descripción de las funciones del sistema.

La compresión fractal trabaja solamente con imágenes de 256 tonos de grises e imágenes en color real (24 bits de color); se excluyen las imágenes monocromáticas y de 256 colores, aunque, como se mencionó anteriormente, es posible leerlas y desplegarlas en la pantalla.

7.2 Descripción Técnica.

Para que este sistema sea susceptible de comprenderse a nivel técnico es necesario describir como está compuesto, es decir, cuales son sus módulos, cual es el flujo de información, que algoritmos se utilizan, etc.; todos estos aspectos se detallan a continuación.

7.2.1 Diagrama de Funciones.

En la figura 7.1 se muestra el diagrama de funciones del sistema. Éste tiene como objetivo presentar los módulos del sistema y el flujo de información entre ellos; la simbología es la siguiente.

Función del sistema.

Entidad externa al sistema.

Información fluyente.

Dirección del flujo de información.

Límites del sistema.

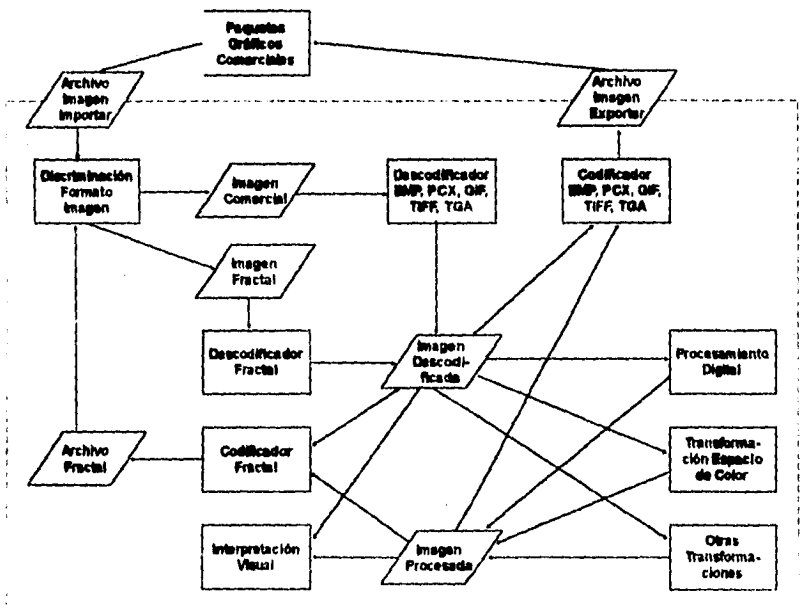


Figura 7.1 Diagrama de Funciones.

7.2.1.1 Discriminación del Formato de Imagen.

La información de entrada al sistema son los diversos archivos de imágenes en formatos gráficos manejados por paquetes comerciales tales como Corel Draw, Photo Finish, Paint Brush, etc., o bien, los archivos fractales generados por el mismo sistema. Si es un formato comercial la imagen se envía al descodificador correspondiente (BMP, PCX, GIF, TIFF y TGA); si es un archivo fractal se envía al descodificador fractal.

7.2.1.2 Descodificador BMP, PCX, GIF, TIFF y TGA.

Una vez que se encuentra que la imagen está almacenada en un archivo con formato comercial se procede a interpretarla con el descodificador correspondiente. En realidad está función se podría dividir en 5 funciones pero debido a que todas ellas son genéricas para este tipo de imagen se han englobado en una sola. Los descodificadores son capaces de interpretar los siguientes tipos de imágenes:

Descodificador BMP	Imagen monocromática. Imagen de 16 colores. Imagen de 256 colores. Imagen de 256 tonos de grises. Imagen de 24 bits de color.
Descodificador PCX	Imagen monocromática, compresión RLE. Imagen de 16 colores, compresión RLE. Imagen de 256 colores, compresión RLE. Imagen de 256 tonos de grises, compresión RLE. Imagen de 24 bits de color, compresión RLE.
Descodificador GIF	Imagen monocromática, compresión LZW. Imagen de 16 colores, compresión LZW. Imagen de 256 colores, compresión LZW. Imagen de 256 tonos de grises, compresión LZW.
Descodificador TIFF	Imagen monocromática, sin compresión. Imagen monocromática, bits empacados. Imagen de 16 colores, sin compresión. Imagen de 256 colores, sin compresión. Imagen de 256 tonos de grises, sin compresión. Imagen de 24 bits de color, sin compresión.

Descodificador TGA	Imagen monocromática, sin compresión. Imagen monocromática, compresión RLE. Imagen de 256 colores, sin compresión. Imagen de 256 colores, compresión RLE. Imagen de 256 tonos de grises, sin compresión. Imagen de 256 tonos de grises, compresión RLE. Imagen de 24 bits de color, sin compresión. Imagen de 24 bits de color, compresión RLE.

7.2.1.3 Descodificador Fractal.

Si se tiene una imagen fractal aquí es en donde se interpreta para su despliegue. El descodificador fractal reconoce los siguientes tipos de imágenes:

Descodificador Fractal	Imagen de 256 tonos de grises, compresión fractal. Imagen de 24 bits de color, compresión fractal.

Para descodificar los dos tipos de imágenes el descodificador cuenta con sus dos módulos respectivos; el módulo de 24 bits se basa en el módulo de 256 tonos de grises pues lo que hace es descomprimir la imagen en 3 pasos, cada uno de ellos corresponde a un componente del espacio YIQ; posteriormente transforma la imagen YIQ descodificada al espacio RGB para su despliegue en pantalla.

7.2.1.4 Interpretación Visual.

Cuando ya se tiene la imagen descodificada, ya sea que provenga de un formato comercial o de un archivo fractal, se procede a desplegarla en el monitor. Para esto se toma en cuenta el tipo de la imagen, es decir, cuantos colores tiene, si existe paleta o no, su dimensión, etc. Además, se efectúa también el manejo de ventanas para poder desplegar varias imágenes a la vez, capacidad de hacer el *scroll* en cada una de ellas y otras operaciones.

7.2.1.6 Codificador BMP, PCX, GIF, TIFF y TGA.

Para realizar la exportación de la imagen a cualquiera de los formatos comerciales manejados se cuenta con la función de codificación BMP, PCX, GIF, TIFF y TGA. Esta función toma la imagen descodificada que está desplegada en ese momento en el monitor y la exporta al formato seleccionado. Como se puede apreciar en el diagrama, no importa de que formato se haya importado la imagen, ésta simplemente se encuentra en una forma genérica en la cual puede codificarse de la manera que se desee; inclusive puede provenir de la función del descodificador fractal, en cuyo caso se está exportando un archivo fractal a un archivo gráfico comercial. Los codificadores son capaces de generar los siguientes tipos de imágenes:

Codificador BMP	**Imagen monocromática.** **Imagen de 16 colores.** **Imagen de 256 colores.** **Imagen de 256 tonos de grises.** **Imagen de 24 bits de color.**
Codificador PCX	**Imagen monocromática, compresión RLE.** **Imagen de 16 colores, compresión RLE.** **Imagen de 256 colores, compresión RLE.** **Imagen de 256 tonos de grises, compresión RLE.** **Imagen de 24 bits de color, compresión RLE.**
Codificador GIF	**Imagen monocromática, compresión LZW.** **Imagen de 16 colores, compresión LZW.** **Imagen de 256 colores, compresión LZW.** **Imagen de 256 tonos de grises, compresión LZW.**
Codificador TIFF	Imagen monocromática, sin compresión. Imagen monocromática, bits empacados. Imagen de 16 colores, sin compresión. Imagen de 256 colores, sin compresión. Imagen de 256 tonos de grises, sin compresión. Imagen de 24 bits de color, sin compresión.

Codificador TGA	Imagen monocromática, sin compresión. Imagen monocromática, compresión RLE. Imagen de 256 colores, sin compresión. Imagen de 256 colores, compresión RLE. Imagen de 256 tonos de grises, sin compresión. Imagen de 256 tonos de grises, compresión RLE. Imagen de 24 bits de color, sin compresión. Imagen de 24 bits de color, compresión RLE.

7.2.1.6 Codificador Fractal.

Al igual que los otros codificadores, el codificador fractal toma la imagen desplegada en pantalla y procede a comprimirla utilizando los códigos IFS. Es capaz de comprimir imágenes de 256 tonos de grises y de 24 bits de color, exclusivamente.
La compresión de imágenes de 24 bits de color está basada en el algoritmo para comprimir imágenes con 256 tonos de grises; lo que se hace es convertir la imagen del espacio de color RGB al espacio de color YIQ, separar estos componentes y comprimir independientemente cada uno de ellos. Una vez que el proceso de compresión ha terminado se tiene como resultado un archivo fractal, factible de leerse con el descodificador fractal para mostrarse en pantalla y exportarse a otros formatos.

7.2.1.7 Procesamiento Digital.

Esta es una función anexa para aplicar algunas técnicas del procesamiento digital a la imagen tales como complementación, definición de brillo y contraste, filtrado y operaciones entre imágenes; esto es con el objetivo de mejorar subjetivamente la calidad de la imagen.
Esta función recibe la imagen descodificada desplegada en pantalla, la transforma y genera otra imagen, la cual puede ser codificada fractalmente o exportarse a los formatos gráficos que se desee, tal como puede apreciarse en el diagrama de funciones.

7.2.1.8 Transformación del Espacio de Color.

En este módulo se efectúan las transformaciones del espacio de color RGB al espacio YIQ y viceversa; estas transformaciones son muy importantes ya que son las que permiten la compresión fractal de imágenes de 24 bits. Se pueden transformar tanto imágenes de 256 colores como de 24 bits. No se permiten transformar imágenes monocromáticas porque no tienen información de color; aunque es posible aplicar estas transformaciones a imágenes en tonos de grises, no tiene sentido hacerlo puesto que estas imágenes son el componente Y del espacio YIQ. A partir de esta función se genera una imagen procesada derivada de la imagen original; esta imagen procesada se puede exportar a otros formatos o comprimirse fractalmente.

7.2.1.9 Otros Transformaciones.

Existen otras transformaciones anexas al sistema, tales como: generación de un árbol cuádruple de la imagen, obtención del histograma, *dither* de 256 colores a 16 colores, remapeo de paleta y cambio de paleta. Al igual que en el procesamiento digital y en las transformaciones de espacio de color, se genera otra imagen que puede comprimirse fractalmente o exportarse a otros formatos.

7.2.2 Algoritmos.

Partiendo del diagrama de funciones se obtienen los algoritmos para la programación del sistema; estos algoritmos se presentan a continuación. Se hace especial énfasis en los algoritmos de codificación/decodificación fractal puesto que son la parte medular de este trabajo.

7.2.2.1 Discriminación del Formato de Imagen.

1. Leer el encabezado del archivo.
2. Buscar el formato.
3. En caso de ser:
    - 3.1 BMP, llamar al decodificador BMP.
    - 3.2 PCX, llamar al decodificador PCX.
    - 3.3 GIF, llamar al decodificador GIF.
    - 3.4 TIFF, llamar al decodificador TIFF.
    - 3.5 TGA, llamar al decodificador TGA.
    - 3.6 IFS, llamar al decodificador IFS.
    - 3.7 Otro, enviar mensaje de formato no reconocido.
4. Fin.

7.2.2.2 Descodificador BMP, PCX, GIF, TIFF y TGA.

Descodificador BMP

1. Abrir el archivo para leer.
2. Determinar si es un archivo BMP tipo OS2 o DOS.
3. Leer los parámetros de la imagen: ancho, altura, bits de color y número de planos.
4. Mientras haya líneas a leer del archivo.
   4.1 Leer línea del archivo.
   4.2 Copiar la línea a la imagen descodificada.
5. Cerrar el archivo.
6. Fin.

Descodificador PCX

1. Abrir el archivo para leer.
2. Leer los parámetros de la imagen: ancho, altura, bits de color y número de planos.
3. Mientras haya líneas a leer del archivo.
   3.1 Leer línea del archivo.
   3.2 Descodificar la línea con el algoritmo RLE.
   3.3 Copiar la línea a la imagen descodificada.
4. Cerrar el archivo.
5. Fin.

Descodificador GIF

1. Abrir el archivo para leer.
2. Mientras haya bloques de información en el archivo.
   2.1 Leer el bloque.
   2.2 Si es un bloque de imagen.
      2.2.1 Leer los parámetros de la imagen: ancho, altura, bits de color y número de planos.
      2.2.2 Descomprimir el bloque de imagen con el algoritmo LZW.
      2.2.3 Construir la imagen descodificada.
3. Cerrar el archivo.
4. Fin.

Descodificador TIFF

1. Abrir el archivo para leer.
2. Determinar si la imagen es para procesadores tipo Intel o Motorola.
3. Buscar el directorio de etiquetas.
4. Buscar en el directorio la etiqueta de los parámetros de la imagen.
5. Buscar en el directorio las etiquetas de la imagen y sus parámetros.
6. Leer los parámetros de la imagen: ancho, altura, bits de color y número de planos.
7. Si la imagen es una imagen sin comprimir
   7.1 Mientras haya líneas a leer del archivo.
      7.1.1 Leer línea del archivo.
      7.1.2 Copiar la línea a la imagen descodificada.
8. Si la imagen es de bits empacados.
   8.1 Mientras haya líneas a leer del archivo.
      8.1.1 Leer línea del archivo.
      8.1.2. Descodificar la línea con el algoritmo de bits empacados.
      8.1.3 Copiar la línea a la imagen descodificada.
9. Cerrar el archivo.
10. Fin.

Deacodificador TGA

1. Abrir el archivo para leer.
2. Leer los parámetros de la imagen: ancho, altura, bits de color y número de planos.
3. Si la imagen es una imagen sin comprimir
   3.1 Mientras haya líneas a leer del archivo.
      3.1.1 Leer línea del archivo.
      3.1.2 Copiar la línea a la imagen descodificada.
4. Si la imagen está comprimida con RLE.
   4.1 Mientras haya líneas a leer del archivo.
      4.1.1 Leer línea del archivo.
      4.1.2.Descodificar la línea con el algoritmo RLE.
      4.1.3 Copiar la línea a la imagen descodificada.
8. Cerrar el archivo.
9. Fin.

7.2.2.3 Descodificador Fractal.

Debido a que este algoritmo, junto con el de la compresión fractal son los más importantes, se detallará no solamente el algoritmo sino también un diagrama de flujo y explicaciones extras.
Los algoritmos de descompresión de imágenes fractales se dividen en dos: uno para imágenes de 256 tonos de grises y el otro para imágenes de 24 bits de color.

Algoritmo para descomprimir imágenes de tonos de grises.

1. Abrir el archivo para leer.
2. Leer los parámetros de la imagen: ancho, altura, bits de color y número de planos.
3. Leer el número de subimágenes en el archivo.
4. Pedir al usuario los parámetros para la descompresión fractal.
5. Inicializar el buffer de la imagen descodificada.
6. Mientras haya subimágenes en el archivo.
   6.1 Leer los mapas IFS de la subimagen.
   6.2 Mientras haya mapas.
      6.2.1 Construir el rango a partir de aplicar el mapa al dominio especificado.
      6.2.2 Copiar el rango construido a la subimagen descodificada.
   6.3 Copiar la subimagen descodificada a la imagen descodificada.
7. Cerrar el archivo.
8. Fin.

El diagrama de flujo se presenta en la **Figura 7.2**.

Algoritmo para descomprimir imágenes de 24 bits de color.

1. Abrir el archivo para leer.
2. Leer los parámetros de la imagen: ancho, altura, bits de color y número de planos.
3. Pedir al usuario los parámetros para la descompresión fractal.
4. Inicializar el buffer de la imagen descodificada.
5. Para cada componente de color en el espacio YIQ.
   5.1 Generar un archivo fractal en tonos de grises a partir del archivo de 24 bits.
   5.2 Descodificar el archivo con el algoritmo de 256 tonos de grises.
6. Unir las 3 imágenes descodificadas en una sola de 24 bits.
7. Transformar la imagen del plano YIQ al plano RGB.
8. Borrar los 3 archivos temporales.
9. Cerrar el archivo.
10. Fin.

El diagrama de flujo se presenta en la **Figura 7.3**.

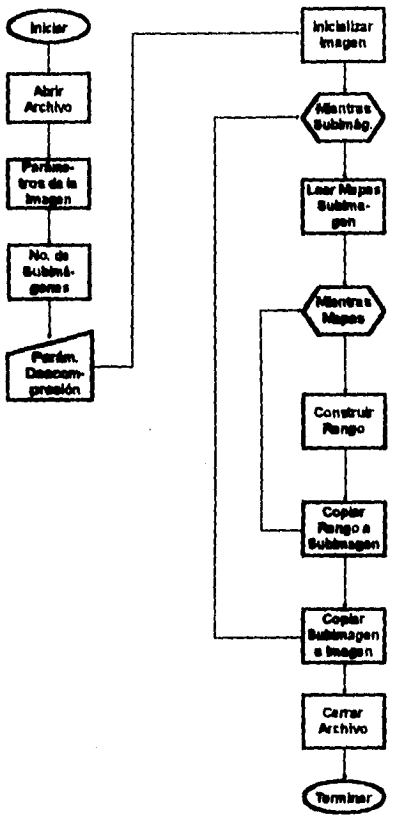


Figura 7.2 Diagrama de Flujo para la Descompresión de Imágenes Fractales de 256 tonos de grises.

Para ambos algoritmos, previamente se aplica al archivo el algoritmo LZW para descomprimirlo; ver la sección 7.2.2.6 de compresión fractal para más explicaciones al respecto.

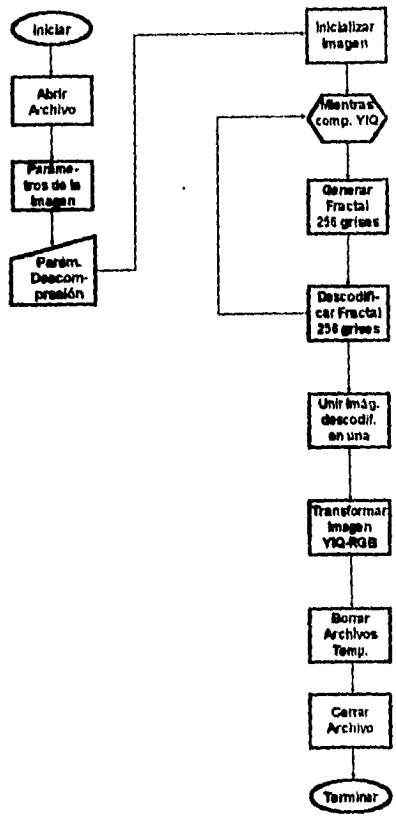


Figura 7.3 Diagrama de Flujo para la Descompresión de Imágenes Fractales de 24 bits de color.

7.2.2.4 Interpretación Visual.

1. Crear e inicializar una ventana de despliegue para la imagen.
2. Definir la paleta de la imagen.
3. Definir los parámetros de la imagen: ancho, altura, bits de color y número de planos.
4. Desplegar en la ventana la imagen descodificada en los otros algoritmos.

7.2.2.5 Codificador BMP, PCX, GIF, TIFF y TGA.

Codificador BMP

1. Abrir el archivo para escribir.
2. Escribir el encabezado del archivo con los parámetros de la imagen.
3. Mientras haya líneas a leer de la imagen desplegada en pantalla.
   3.1 Leer la línea de la imagen.
   3.2 Copiar la línea al archivo de salida.
4. Cerrar el archivo.
5. Fin.

Codificador PCX

1. Abrir el archivo para escribir.
2. Escribir el encabezado del archivo con los parámetros de la imagen.
3. Mientras haya líneas a leer de la imagen desplegada en pantalla.
   3.1 Leer la línea de la imagen.
   3.2 Codificar la línea con el algoritmo RLE.
   3.3 Copiar la línea al archivo de salida.
4. Cerrar el archivo.
5. Fin.

Codificador GIF

1. Abrir el archivo para escribir.
2. Escribir el bloque descriptor de la pantalla.
3. Escribir el bloque descriptor de la imagen.
4. Comprimir la imagen con el algoritmos LZW.
5. Escribir el bloque de la imagen.
6. Escribir bloque de comentario de la imagen.
7. Cerrar el archivo.
5. Fin.

Codificador TIFF

1. Abrir el archivo para escribir.
2. Escribir la etiqueta de la aplicación.
3. Escribir las etiquetas de la imagen y sus parámetros.

4. Si la imagen es una imagen sin comprimir
   4.1 Mientras haya líneas a leer de la imagen desplegada en pantalla.
      4.1.1 Leer la línea de la imagen.
      4.1.2 Copiar la línea al archivo de salida.
5. Si la imagen es de bits empacados.
   5.1 Mientras haya líneas a leer de la imagen desplegada en pantalla.
      5.1.1 Leer la línea de la imagen.
      5.1.2. Codificar la línea con el algoritmo de bits empacados.
      5.1.3 Copiar la línea al archivo de salida.
6. Cerrar el archivo.
7. Fin.

Codificador TGA

1. Abrir el archivo para escribir.
2. Escribir el encabezado del archivo con los parámetros de la imagen.
3. Si la imagen es una imagen sin comprimir
   3.1 Mientras haya líneas a leer de la imagen desplegada en pantalla.
      3.1.1 Leer la línea de la imagen.
      3.1.2 Copiar la línea al archivo de salida.
4. Si la imagen está comprimida con RLE.
   4.1 Mientras haya líneas a leer del archivo.
      4.1.1 Leer la línea de la imagen.
      4.1.2.Codificar la línea con el algoritmo RLE.
      4.1.3 Copiar la línea al archivo de salida.
8. Cerrar el archivo.
9. Fin.

7.2.2.6 Codificador Fractal.

Este algoritmo, junto con su contraparte para la descodificación, es el más importante del sistema desarrollado pues es dónde se plasma toda la teoría de compresión de imágenes con técnicas fractales, título de esta tesis. En los párrafos siguientes se detallan los algoritmos y diagramas de flujos para este nuevo esquema de compresión de imágenes.

Algoritmo para comprimir imágenes de tonos de grises.

1. Abrir el archivo para escribir.
2. Pedir al usuario los parámetros para la compresión fractal.

3. Escribir el encabezado del archivo con los parámetros de la imagen y de las subimágenes.
4. Particionar la imagen en subimágenes.
5. Mientras haya subimágenes en la imagen.
   5.1 Reducir la subimagen a la mitad.
   5.2 Definir dominios a partir de la subimagen reducida.
   5.3 Definir rangos a partir de la subimagen.
   5.4 Mientras haya rangos en la subimagen.
      5.4.1 Calcular el promedio del rango.
      5.4.2 Mientras haya dominios
         5.4.2.1. Encontrar el promedio mínimo cuadrado entre rango y dominio.
         5.4.2.2. Para las 8 simetrías del dominio.
            5.4.2.2.2 Aplicar la simetría al dominio.
            5.4.2.2.1. Encontrar la distancia mínima (error mínimo) entre el rango y el dominio.
            5.4.2.2.3. Si la distancia mínima es menor que la anterior guarda temporalmente el mapa óptimo construido.
      5.4.3 Escribir al archivo el mapa óptimo almacenado temporalmente.
6. Cerrar el archivo.
7. Fin.

El diagrama de flujo se presenta en la **Figura 7.4.**

Algoritmo para Comprimir imágenes de 24 bits de color.

1. Abrir el archivo para escribir.
2. Pedir al usuario los parámetros para la compresión fractal.
3. Transformar la imagen del espacio de color RGB al espacio YIQ.
4. Para cada componente de color en el espacio YIQ.
   4.1 Generar un archivo fractal en tonos de grises
   4.2 Codificar el archivo con el algoritmo de 256 tonos de grises.
5. Escribir el encabezado del archivo con los parámetros de la imagen y de las subimágenes.
6. Unir los 3 archivos fractales temporales en el archivo de 24 bits.
7. Borrar los archivos temporales.
8. Cerrar el archivo.
9. Fin.

El diagrama de flujo se presenta en la **Figura 7.5.**

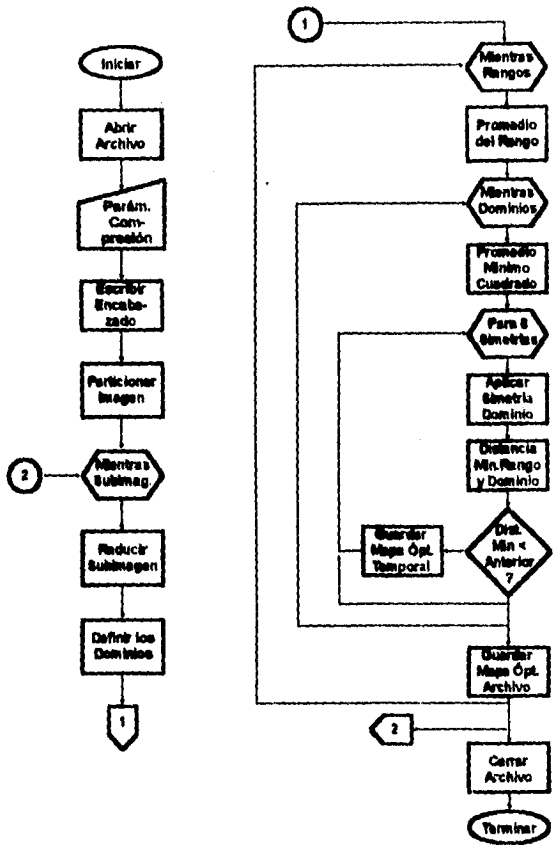


Figura 7.4 Diagrama de Flujo para la Compresión de Imágenes Fractales de 256 tonos de grises.

Para ambos tipos de imágenes, una vez que se ha generado el archivo fractal se comprime éste con el algoritmos LZW para lograr una razón de compresión más alta. La aplicación del LZW es una parte independiente de los algoritmos fractales pero podemos considerar ambos algoritmos como un todo para efecto de evaluación de la razón de compresión obtenida.

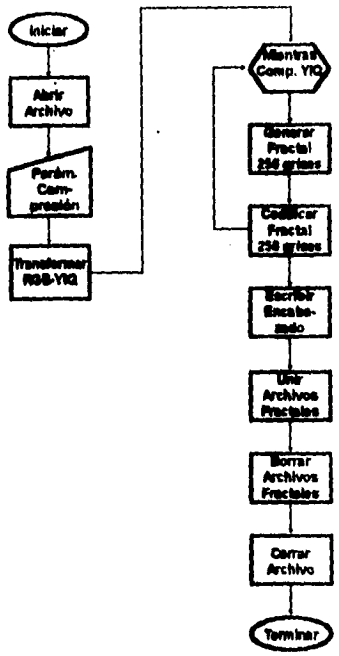


Figura 7.5 Diagrama de Flujo para la Compresión de Imágenes Fractales de 24 bits de color.

7.2.2.7 Procesamiento Digital.

Reescalar Imagen.

1. Tomar la imagen desplegada en pantalla y sus parámetros.
2. Inicializar la imagen descodificada reducida a la mitad.
3. Mientras haya pixeles en la imagen.
   3.1 Promediar el pixel con el pixel de la derecha y los 2 de abajo.
   3.2 Escribir el pixel nuevo a la imagen descodificada.
4. Fin.

Complementar Imagen.

1. Tomar la imagen desplegada en pantalla y sus parámetros.
2. Inicializar la imagen descodificada.
3. Mientras haya pixeles en la imagen.
   3.1 Aplicar la fórmula Pixel Nuevo = 255-Pixel.
   3.2 Escribir el pixel nuevo a la imagen descodificada.
4. Fin.

Brillo y Contraste.

1. Tomar la imagen desplegada en pantalla y sus parámetros.
2. Inicializar la imagen descodificada.
3. Mientras haya pixeles en la imagen.
   3.1 Aplicar la fórmula Pixel Nuevo = (Pixel + Brillo)*Contraste.
   3.2 Escribir el pixel nuevo a la imagen descodificada.
4. Fin.

Filtrar Imagen.

1. Tomar la imagen desplegada en pantalla y sus parámetros.
2. Inicializar la imagen descodificada.
3. Mientras haya líneas a leer de la imagen desplegada en pantalla.
   3.1 Leer 3 líneas simultáneamente.
   3.2 Mientras haya pixeles en las líneas.
      3.2.1. Aplicar la fórmula Pixel Nuevo = Pixeles Adyacentes * Filtro.
      3.2.2. Escribir el pixel nuevo a la imagen descodificada.
4. Fin.

Combinar Imagen.

1. Leer del archivo la imagen 1.
2. Leer del archivo la imagen 2.
3. Inicializar la imagen descodificada.
4. Si las dimensiones y número de colores de la imagen son iguales.
   4.1 Aplicar la fórmula Pixel Nuevo = (Pixel Imagen 1 *Operador* Pixel Imagen 2)
   (Donde *operador* es: resta, promedio o división).
   4.2 Escribir el pixel nuevo a la imagen descodificada.
5. Si no, Enviar mensaje que las imágenes no son compatibles.
6. Fin.

7.2.2.8 Transformación del Espacio de Color.

Transformación RGB-YIQ.

1. Tomar la imagen desplegada en pantalla y sus parámetros.
2. Inicializar la imagen descodificada.
3. Mientras haya pixeles en la imagen.
   3.1 Aplicar la fórmula Pixel Nuevo = Pixel * Matriz RGB-YIQ.
   3.2 Escribir el pixel nuevo a la imagen descodificada.
4. Remapear los pixeles para que queden en el rango de 0 a 255.
5. Fin.

Transformación YIQ-RGB.

1. Tomar la imagen desplegada en pantalla y sus parámetros.
2. Inicializar la imagen descodificada.
3. Remapear los pixeles del rango 0-255 al rango original YIQ.
4. Mientras haya pixeles en la imagen.
   4.1 Aplicar la fórmula Pixel Nuevo = Pixel * Matriz YIQ-RGB.
   4.2 Escribir el pixel nuevo a la imagen descodificada.
5. Fin.

Canal de Color.

1. Tomar la imagen desplegada en pantalla y sus parámetros.
2. Inicializar la imagen descodificada.
3. Mientras haya pixeles en la imagen.
   3.1 Si el canal es rojo poner a cero los componentes verde y azul.
   3.2 Si el canal es verde poner a cero los componentes rojo y azul.
   3.3 Si el canal es azul poner a cero los componentes rojo y verde.
4. Fin.

7.2.2.9 Otras Transformaciones.

Árbol Cuádruple.

1. Tomar la imagen desplegada en pantalla y sus parámetros.
2. Inicializar la imagen descodificada.
3. Pedir el criterio para la construcción del árbol.
4. Inicializar la lista de particiones a dividir con la imagen entera.

5. Mientras haya particiones en la imagen.
   5.1 Particionar la imagen en 4 rectángulos.
   5.2 Para las 4 particiones.
      5.2.1. Encontrar el histograma de colores.
      5.2.2. Si la moda es mayor que la aproximación requerida se guarda esta partición en el árbol cuádruple, junto con la moda.
      5.2.3. Si no, se guarda la partición en la lista de particiones a dividir.
6. Aplicar el árbol cuádruple a la imagen.
7. Fin.

Histograma.

1. Tomar la imagen desplegada en pantalla y sus parámetros.
2. Generar el histograma.
3. Normalizar los valores para su despliegue.
4. Desplegar el histograma.

Dither.

1. Tomar la imagen desplegada en pantalla y sus parámetros.
2. Si la imagen es de 256 colores o 256 tonos de grises.
   2.1. Inicializar la imagen descodificada para reducir de 256 a 16 colores.
   2.2. Mientras haya líneas a leer de la imagen desplegada en pantalla.
      2.2.1. Aplicar el dither a la línea.
      2.2.2. Copiar la línea a la imagen descodificada.
3. Fin.

Remapear Paleta.

1. Tomar la imagen desplegada en pantalla y sus parámetros.
2. Inicializar la imagen descodificada.
3. Ordenar ascendentemente la paleta de colores.
4. Copiar la imagen original a la imagen descodificada con la nueva paleta.
5. Fin.

Cambiar Paleta.

1. Tomar la imagen desplegada en pantalla y sus parámetros.
2. Inicializar la imagen descodificada.
3. Asignar la paleta seleccionada: grises o una estándar de colores.
4. Copiar la imagen original a la imagen descodificada con la nueva paleta.
5. Fin.

7.2.3 Hardware y Software utilizados.

Debido a la complejidad de los algoritmos fractales, los requerimientos de hardware y software para el desarrollo y ejecución del sistema son altos; éstos se detallan a continuación.

7.2.3.1 Hardware.

Para ejecutar el sistema:

Microcomputadora PC compatible con las siguientes características:

- Procesador 486 DX o SX, preferentemente a 33 Mhz.
- 4 Mbytes en RAM.
- Disco duro de 80 Mbytes.
- Coprocesador matemático (altamente recomendable).
- Tarjeta de video de 256 colores.

Para el desarrollo del sistema:

Microcomputadora PC descrita anteriormente.

Scan Jet II de Hewlette Packard para la adquisición de imágenes, con las siguientes características:

- Adquisición de imágenes de hasta 2048x2048 pixeles.
- Imágenes monocromáticas, 256 colores, 256 tonos de grises y 24 bits de color.

7.2.3.2 Software.

Para ejecutar el sistema:

- Sistema operativo MS-DOS 5.0.
- Windows 3.1 o superior.

Para el desarrollo del sistema:

- Compilador Microsoft Quick C 1.0 para la generación del archivo ejecutable.
- Paquete gráfico PhotoFinish 1.01, de ZSoft Corp para la manipulación de imágenes en formato BMP, PCX, GIF, TIFF, TGA, MSP y EPS.
- Software de demostración Hands-On para el procesamiento digital de imágenes.

Para la documentación del sistema:

- Procesador de textos Microsoft Word 6.0.
- Paquete Gráfico Harvard Graphics 1.0
- Paquete gráfico PhotoFinish 1.01, de ZSoft Corp.

7.2.3.3 Justificaciones.

El sistema se desarrollo bajo ambiente Windows y para ejecutarse en este mismo ambiente por las facilidades que brinda tanto al desarrollador como al usuario final: ejecutar múltiple aplicaciones, trabajar con varias ventanas a la vez, compartir información entre las aplicaciones, etc.

Se escogió el lenguaje C para su desarrollo por la rapidez de ejecución de su código, el excelente manejo de memoria y la portabilidad entre diferentes plataformas[1]. No se seleccionó C++ por la sobrecarga del manejo de clases que tienen estos compiladores y que para este sistema no iban a ser utilizadas.

El compilador que se escogió es Microsoft Quick C 1.0 para Windows debido a su facilidad de uso, a la rapidez de compilación, a la alta optimización del archivo final ejecutable y los requerimientos mínimos de hardware y software para su ejecución. Se evaluaron también Microsoft C++, Microsoft Visual C++ y Borland C++, los resultados se presentan en la tabla de la **Figura 7.6**.

[1] Sin embargo, esto no es totalmente cierto en ambiente Windows, ver la sección 7.2.3.4 para más información.

Compilador	Facilidad de Uso	Rapidez de Compilación	Optimización del Ejecutable	Requerimientos Mínimos de Hardware y software
Microsoft Quick C 1.0	V	V	V	V
Microsoft C++ 7.0 y el Software Development Kit	F	V	V	F
Microsoft Visual C++ 1.0	V	F	F	F
Borland C++ 3.0	V	F	F	V

V = Verdadero.
F = Falso.

Figura 7.6 Tabla Comparativa de Compiladores.

En cuanto a los otros paquetes utilizados para el desarrollo y documentación del sistema (Word, Harvard Graphics, Photofinish, etc.), también se escogieron por ser de los más completos en el mercado, aunque podrían haberse utilizado otros sin mayor problema.

7.2.3.4 Portabilidad.

Desgraciadamente, la restricción del ambiente Windows en el manejo de memoria hace que el sistema no sea totalmente portable a otras plataformas tales como Unix, o inclusive al mismo ambiente MS-DOS. Es necesario modificar la mayoría de las funciones de manejo de imágenes, codificación y descodificación para que utilicen llamadas estándares del manejo de memoria tales como malloc, calloc, free, etc. porque actualmente utilizan las funciones GlobalAlloc, LocalAlloc, GlobalFree, etc. de Windows.

La parte de despliegue gráfico obviamente está atada a Windows y necesitaría reescribirse completamente para migrarse a otra plataforma.

7.3 Guía del Usuario.

Para el usuario final del sistema, esta sección es la más importante porque le presenta las características del sistema y las opciones que posee. Se hablará más adelante de la instalación del sistema, los menúes y las pantallas.

7.3.1 Instalación del Sistema.

1. Crear un directorio en el disco duro en el cual se instalará el sistema.
2. Copiar el archivo fotofract.exe del disco A a este directorio.
3. Dar de alta un grupo en el Administrador de Programas de Windows.
4. Dar de alta en este grupo un nuevo elemento, seleccionando para ello el ejecutable que se encuentra en el directorio creado en el paso 1.

7.3.2 Ingreso al Sistema.

Para ejecutar la aplicación e ingresar al sistema simplemente haga doble click en el icono creado en el paso 4 de la instalación del sistema. La ventana que aparece se muestra en la **Figura 7.7**.

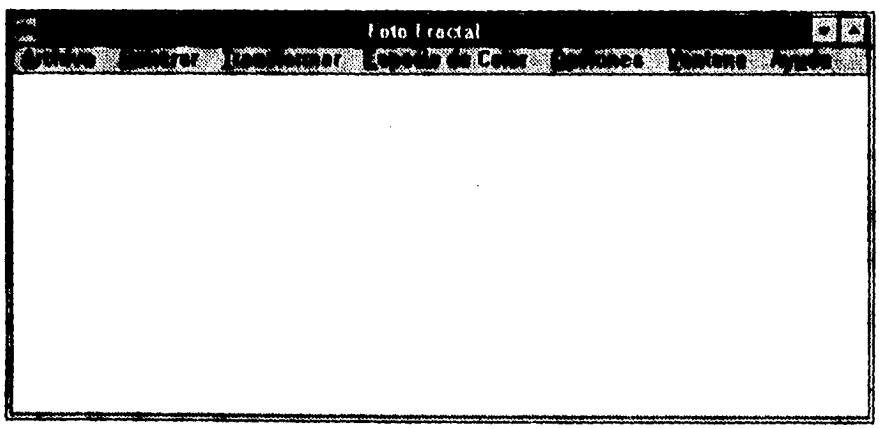


Figura 7.7 Ventana Principal del Sistema.

7.3.3 Descripción de Menúes.

Se presentan a continuación los submenúes de los que consta el sistema, así como una breve descripción de ellos.

7.3.3.1 Submenú Archivo.

La **Figura 7.8** presenta las opciones del submenú Archivo, descritas a continuación.

1. **Nuevo:** Crea una ventana nueva; actualmente esto no tiene aplicación.
2. **Abrir:** Abre un archivo para leer la imagen.
3. **Cerrar:** Cierra la ventana activa.
4. **Cerrar Todas:** Cierra todas las ventanas abiertas.
5. **Salvar:** Salva la imagen de la ventana activa.
6. **Información:** Presenta el diálogo de información general de una imagen.
7. **Detalles:** Presenta el diálogo de los detalles especificos de una imagen.
8. **Imprimir:** No funciona actualmente.
9. **Salir:** Termina la aplicación.

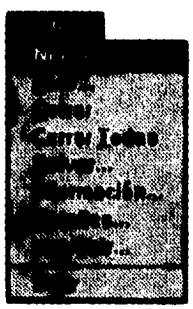


Figura 7.8 Submenú Archivo.

7.3.3.2 Submenú Generar.

La **Figura 7.9** presenta las opciones del submenú Generar, descritas a continuación.

1. **Imagen Fractal:** Comprime fractalmente la imagen de la ventana activa.
2. **Árbol Cuádruple:** Genera el árbol cuádruple de la imagen de la ventana activa.
3. **Histograma:** Presenta el diálogo con el histograma de la imagen.

Figura 7.9 Submenú Generar.

7.3.3.3 Submenú Transformar.

La **Figura 7.10** presenta las opciones del submenú Transformar, descritas a continuación.

1. **Reescalar Imagen:** Reescala a la mitad la imagen de la ventana activa.
2. **Complementar:** Complementa la imagen de la ventana activa.
3. **Brillo y Contraste:** Modifica el brillo y contraste de la imagen de la ventana activa.
4. **Filtrar:** Aplica un filtro de convolución a la imagen de la ventana activa.
5. **Combinar Imágenes:** Combina 2 imágenes por medio de un operador: **restar**, **promediar** y **dividir**.

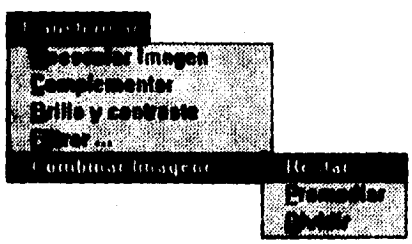


Figura 7.10 Submenú Transformar.

7.3.3.4 Submenú Espacio de Color.

La **Figura 7.11** presenta las opciones del submenú Espacio de Color, descritas a continuación.

1. **A Plano YIQ:** Se genera una imagen transformada al plano YIQ, en cualquiera de sus componentes o con todos.
2. **A Plano RGB:** Se genera una imagen transformada al plano RGB, en cualquiera de sus componentes o con todos.
3. **Canal de Color:** Se genera una imagen convertida a algún componente de color del plano RGB.

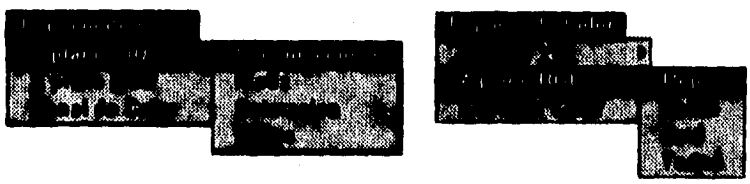


Figura 7.11 Submenú Espacio de Color.

7.3.3.5 Submenú Opciones.

La **Figura 7.12** presenta las opciones del submenú Opciones, descritas a continuación.

1. **Aplicar Dither:** Bandera para indicar si se aplica el dither a una imagen de 256 colores para convertirla a una de 16 colores.
2. **Remapear Paleta:** Ordena ascendentemente la paleta de la imagen.
3. **Cambiar Paleta:** Cambia la paleta de la imagen, ya sea a grises o a una paleta estándar de colores.

Figura 7.12 Submenú Opciones.

7.3.3.6 Submenú Ventana.

La **Figura 7.13** presenta las opciones del submenú Ventana, descritas a continuación.

1. **Mosaico:** Ordena las ventanas en modo mosaico.
2. **Cascada:** Ordena las ventanas en modo cascada.
3. **Arreglar Iconos:** Ordena los iconos de las ventanas minimizadas.

Figura 7.13 Submenú Ventana.

7.3.3.7 Submenú Ayuda.

La **Figura 7.14** presenta las opciones del submenú Ayuda, descritas a continuación.

1. **Acerca:** Presenta el diálogo de información de la aplicación.

Figura 7.14 Submenú Ayuda.

7.3.4 Descripción de Pantallas.

Para la entrada y salida de información el sistema cuenta con diversos diálogos, los cuales se describen a continuación.

7.3.4.1 Abrir Archivo.

Este diálogo permite al usuario seleccionar un archivo de imagen para abrir (**Figura 7.15**); es posible escribir directamente el nombre del archivo o bien navegar a través de las unidades de discos y directorios para seleccionar posteriormente un archivo de la lista, al presionar el botón de Abrir se verifica que el archivo seleccionado sea válido y termina el diálogo. El diálogo se llama desde las opciones siguientes: **Archivo->Abrir, Archivo->Información, Archivo->Detalles** y **Transformar->Combinar Imágenes**.

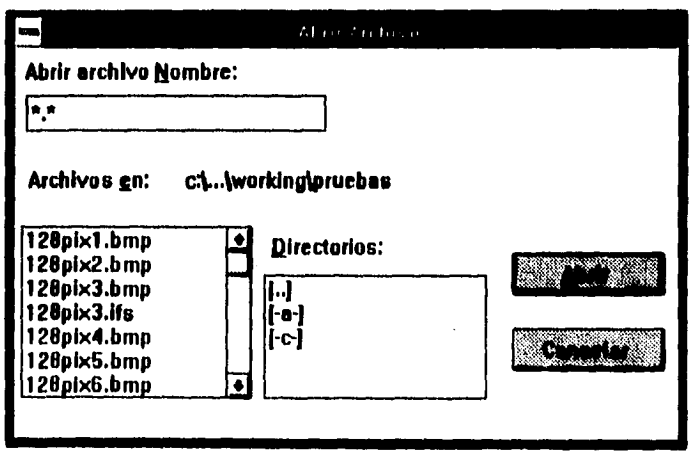


Figura 7.15 Diálogo de Abrir Archivo.

7.3.4.2 Salvar Archivo Como.

En este diálogo (**Figura 7.16**) se elige el archivo al cual se escribirá la imagen de la ventana activa. Al igual que en el diálogo de Abrir Archivo, es posible navegar a través de las unidades de disco y directorios. Una vez que se escribe el nombre del archivo a salvar se verifica si éste es válido o si no existe un archivo con este nombre, en cuyo caso se pregunta si se reemplaza o no. El diálogo se abre a través de la opción **Archivo->Salvar**.

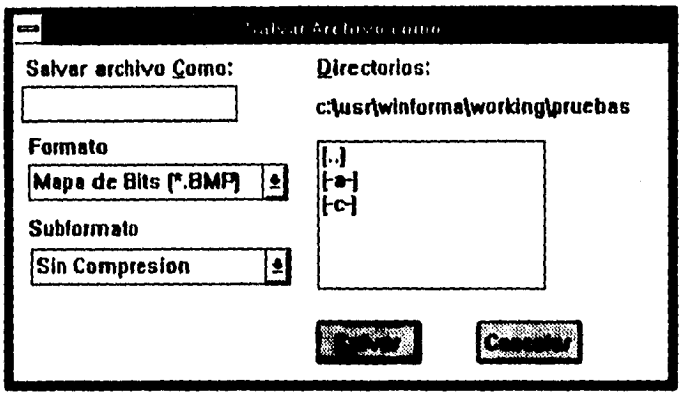


Figura 7.16 Diálogo de Salvar Archivo Como.

7.3.4.3 Información de Archivo.

Después de seleccionar un archivo se abre este diálogo para mostrar la información de un archivo (**Figura 7.17**).

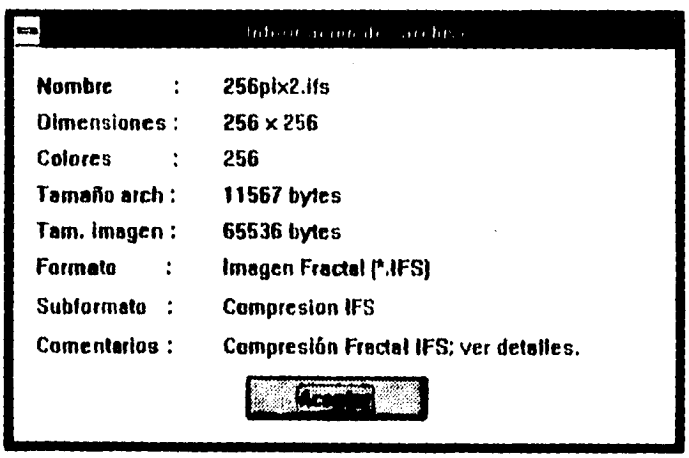


Figura 7.17 Diálogo de Información de Archivo.

7.3.4.4 Detalles de Archivo.

Nuevamente, al igual que en la Información del Archivo, después de seleccionar el archivo se presentan los detalles de éste (**Figura 7.18**).

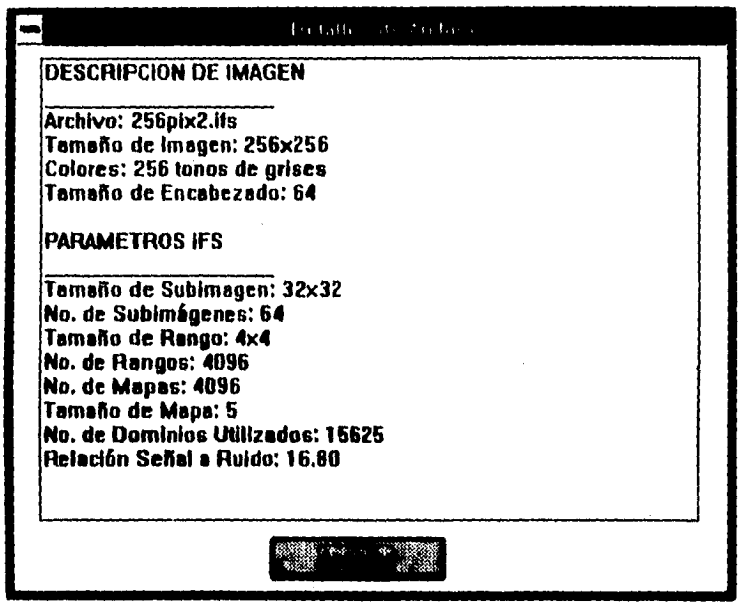


Figura 7.18 Diálogo de Detalles de Archivo.

7.3.4.5 Parámetros IFS.

Este diálogo permite al usuario definir los parámetros para la compresión y descompresión fractales (**Figura 7.19**).

Para la compresión se valida que:

- El ancho y alto de las subimágenes sean múltiplos del ancho y alto de la imagen, respectivamente.

- El lado del rango sea múltiplo de la subimagen.
- El incremento en el dominio sea menor que el lado del rango.
- El número de simetrías esté entre 1 y 8.

Para la descompresión se valida que:

- El número de iteraciones esté entre 0 y 32.
- El color inicial esté entre 0 y 255.
- El escalamiento esté entre 1 y 4.

El número de subimágenes y el número de dominios se calculan automáticamente de acuerdo a las fórmulas descritas en la sección 6.5.3.

El diálogo se presenta al seleccionar las opciones **Generar->Imagen Fractal, Archivo->Salvar** en formato IFS y **Archivo->Abrir** en formato IFS; en la primera y segunda opción se deben capturar ambos parámetros, en la tercera únicamente los parámetros de descompresión.

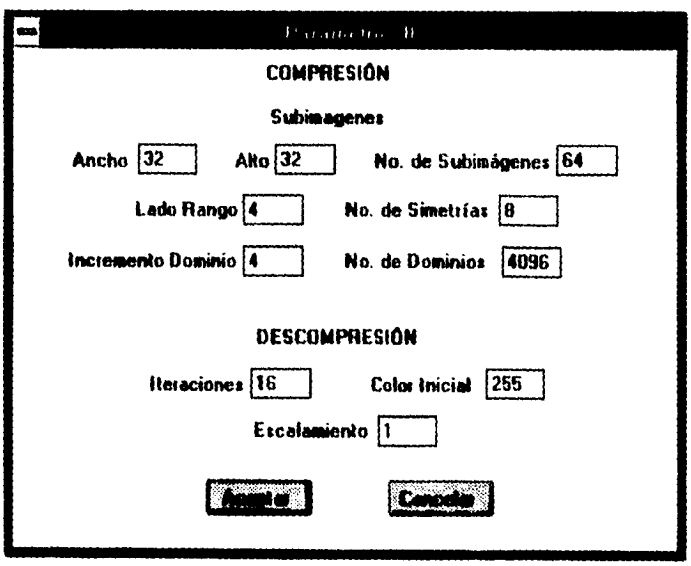


Figura 7.19 Parámetros IFS.

7.3.4.6 Parámetros del Árbol Cuádruple.

Aquí se solicitan los parámetros para construir el árbol cuádruple de la imagen (**Figura 7.20**). Se pide el ancho y alto mínimos de la subimagen o partición del árbol; la aproximación es el porcentaje que debe tener la moda del histograma para tomar como buena o no la partición; por ejemplo, 75 indica que 75% de los pixeles deben ser de un mismo color. Los valores para el ancho y alto de las subimágenes deben estar entre 2 y el ancho y alto de la imagen, respectivamente; el valor para la aproximación debe estar entre 1 y 100. El diálogo se abre a través de la opción **Generar->Árbol Cuádruple**.

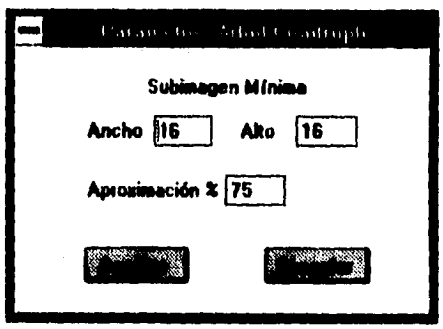


Figura 7.20 Parámetros Árbol Cuádruple.

7.3.4.7 Histograma.

El diálogo simplemente muestra el histograma de la imagen en la ventana activa, ya sea para 256 tonos de grises o 24 bits de color. En la **Figura 7.21** se presenta el histograma para una imagen en 256 tonos de grises.

7.3.4.8 Parámetros de Brillo y Contraste.

En el diálogo se piden los valores de brillo y contraste que se desea aplicar a la imagen (**Figura 7.22**); estos valores deben estar en el rango de -255 a 255 y 0 a 10, respectivamente. El diálogo se abre con la opción **Transformar->Brillo y Contraste**.

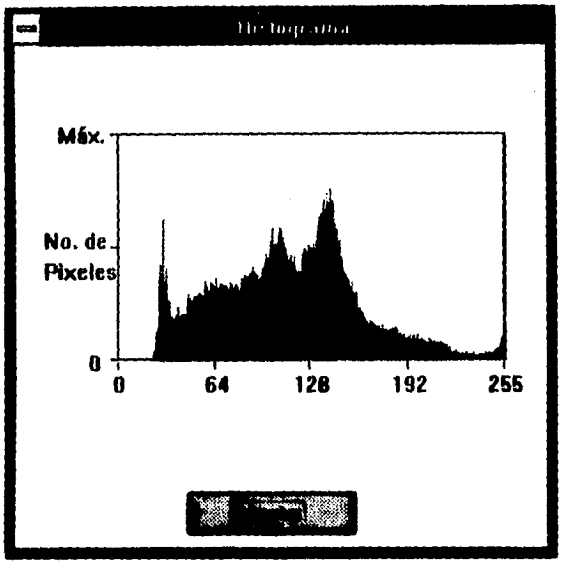


Figura 7.21 Histograma.

Figura 7.22 Parámetros de Brillo y Contraste.

7.3.4.9 Parámetros del Filtro.

Para el filtrado de la imagen es necesario definir una máscara de convolución; este diálogo permite capturar los 9 valores de la máscara (**Figura 7.23**). Se cuenta además con 6 opciones para aplicar filtros típicos paso bajas y paso altas predefinidos. La opción que activa este diálogo es **Transformar->Filtrar**.

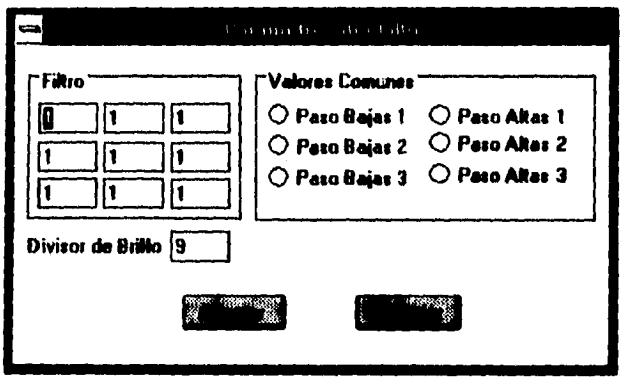


Figura 7.23 Parámetros del Filtro.

7.3.4.10 Acerca de la Aplicación.

Este diálogo (**Figura 7.24**) solamente tiene la función de desplegar la información general de la aplicación. Se activa con la opción **Ayuda->Acerca**.

Figura 7.24 Acerca de la Aplicación.

7.4 Resultados.

La presentación de resultados es una de las partes más importantes porque aquí es donde se sintetiza todo el trabajo efectuado; por esta razón, en la siguientes secciones se discuten los datos obtenidos, las gráficas, las comparaciones con otras técnicas de compresión, y otras cuestiones relevantes. No se presentan los datos de todas las imágenes utilizadas porque son similares, sino más bien se hace una división en 3 tipos de imágenes típicas.

7.4.1 Tablas y Gráficas.

Los resultados que se presentan son para imágenes de 256 tonos de grises, de tamaño 256x256 pixeles; conservando los mismos parámetros, pero utilizando imágenes de 512x512 pixeles, lo único que se cuadruplica es el tiempo de procesamiento porque una imagen de estas dimensiones contiene 4 imágenes de 256x256 pixeles.
Los datos para imágenes de 24 bits de color son exactamente los mismos solo que el tiempo de procesamiento aumenta 3 veces, una vez para cada componente RGB de la imagen.

Los parámetros que se utilizan para la compresión fractal son, como ya se mencionó previamente en la sección 6.5.3, la dimensión de la subimagen, la dimensión del rango y el número de dominios, siendo este último parámetro el resultado del incremento en el barrido de dominios y el número de simetrías; las variables dependientes a evaluar son la razón de compresión RC, la relación señal a ruido RSR y el tiempo de compresión. Los resultado son el promedio de varias imágenes típicas analizadas.

Como nota adicional hay que mencionar que al desplegar los detalles de un archivo fractal, se presentan todos los parámetros descritos en la sección 6.5.3.

7.4.1.1 Imagen de 16 Tonos de Grises.

La imagen de la **Figura 7.25** es una imagen típica de 16 tonos de grises convertida a 256 tonos de grises; como se puede apreciar, el histograma se encuentra disperso a través de los 256 tonos aunque sean solo unos pocos los tonos presentes

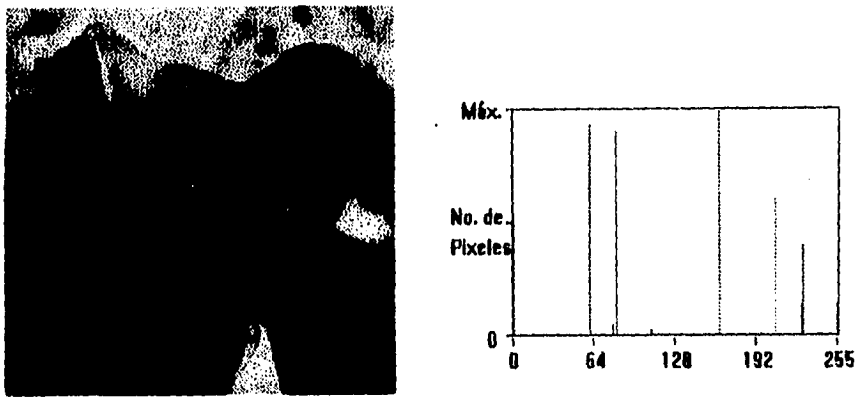


Figura 7.25 Imagen Típica de 16 Tonos de Grises y su Histograma.

Las tablas obtenidas para los diversos parámetros y variables dependientes se muestran a continuación.

- Valores de RC, RSR y Tiempo, variando el ancho de la subimagen.

Ancho Subimg.	Lado Rango	No. Dom.	RC	RSR (dB)	Tiempo (seg)
8x8	4	4	5:1	14.90	4
16x16	4	16	5:1	15.10	4
32x32	4	64	5:1	15.30	14
64x64	4	256	5:1	15.50	44
128x128	4	1024	5:1	15.50	160

- Valores de RC, RSR y Tiempo, variando el lado del rango.

Ancho Subimg.	Lado Rango	No. Dom.	RC	RSR (dB)	Tiempo (seg)
32x32	4	64	5:1	15.30	14
32x32	8	49	25:1	14.70	6
32x32	16	9	45:1	14.10	2

- Valores de RC, RSR y Tiempo, variando número de dominios.

Ancho Subimg.	Lado Rango	No. Dom.	RC	RSR (dB)	Tiempo (seg)
32x32	4	225	5:1	15.50	34
32x32	4	450	5:1	15.50	52
32x32	4	512	5:1	15.50	54
32x32	4	900	5:1	15.70	86
32x32	4	1,800	5:1	15.70	160
32x32	4	6,728	5:1	15.70	546

Las Figuras 7.26, 7.27 y 7.28 presentan estos resultados en forma gráfica; las gráficas no pretenden ser exactas, sino más bien muestran el comportamiento típico de las imágenes de 16 tonos de grises.

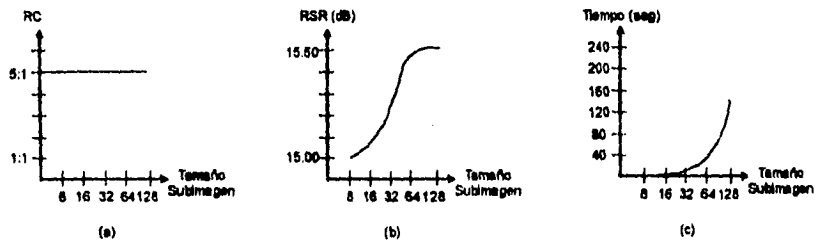


Figura 7.26 Gráficas de (a) Razón de Compresión, (b) Relación Señal a Ruido y (c) Tiempo, respecto al Tamaño de la Subimagen.

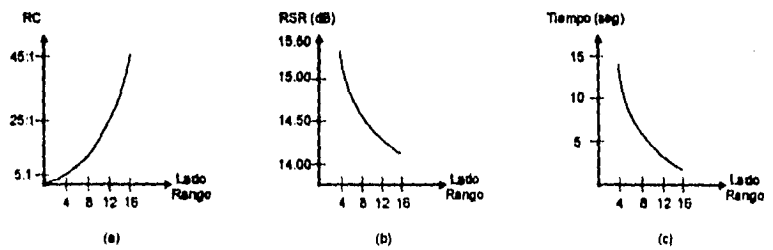


Figura 7.27 Gráficas de (a) Razón de Compresión, (b) Relación Señal a Ruido y (c) Tiempo, respecto al Lado del Rango.

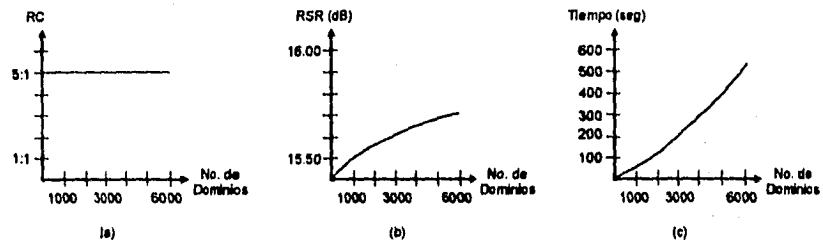


Figura 7.28 Gráficas de (a) Razón de Compresión, (b) Relación Señal a Ruido y (c) Tiempo, respecto al Número de Dominios.

7.4.1.2 Imagen de 256 Tonos de Grises, adquirida a 100 dpi.

La imagen de la **Figura 7.29** es una imagen típica de 256 tonos de grises adquirida con una resolución de 100 dpi (*Dot Per Inche* o puntos por pulgada). En el histograma se puede ver que la distribución de pixeles es un poco más denso que para una imagen de 16 tonos de grises.

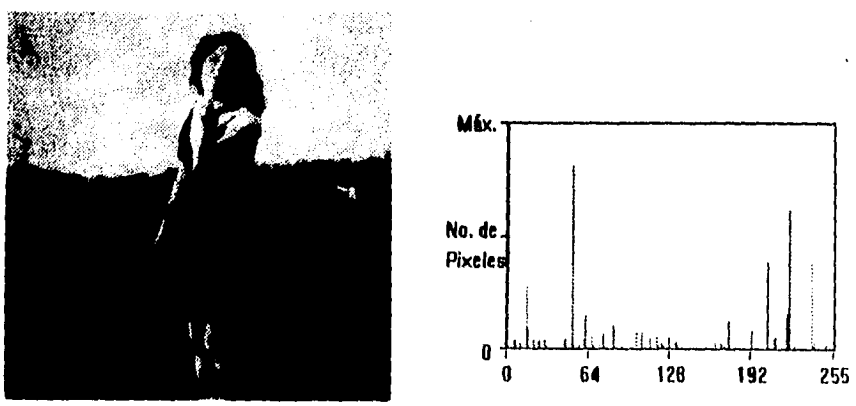


Figura 7.29 Imagen Típica de 256 Tonos de Grises Adquirida a 100 dpi y su Histograma.

Las tablas obtenidas para los diversos parámetros y variables dependientes se muestran a continuación.

- Valores de RC, RSR y Tiempo, variando el ancho de la subimagen.

Ancho Subimg.	Lado Rango	No. Dom.	RC	RSR (dB)	Tiempo (seg)
8x8	4	4	5:1	24.08	4
16x16	4	16	5:1	25.24	4
32x32	4	64	5:1	25.89	14
64x64	4	256	5:1	25.89	44
128x128	4	1024	5:1	25.89	160

- Valores de RC, RSR y Tiempo, variando el lado del rango.

Ancho Subimg.	Lado Rango	No. Dom.	RC	RSR (dB)	Tiempo (seg)
32x32	4	64	5:1	25.89	14
32x32	8	49	25:1	22.14	6
32x32	16	9	45:1	20.14	2

- Valores de RC, RSR y Tiempo, variando número de dominios.

Ancho Subimg.	Lado Rango	No. Dom.	RC	RSR (dB)	Tiempo (seg)
32x32	4	225	5:1	26.58	34
32x32	4	450	5:1	26.58	52
32x32	4	512	5:1	26.58	54
32x32	4	900	5:1	26.58	86
32x32	4	1,800	5:1	26.58	160
32x32	4	6,728	5:1	27.34	546

Las **Figuras 7.30, 7.31** y **7.32** presentan estos resultados en forma gráfica para las imágenes de 256 tonos de grises con 100 dpi.

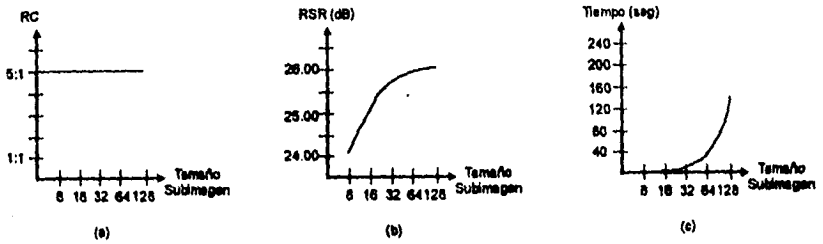


Figura 7.30 **Gráficas de (a) Razón de Compresión, (b) Relación Señal a Ruido y (c) Tiempo, respecto al Tamaño de la Subimagen.**

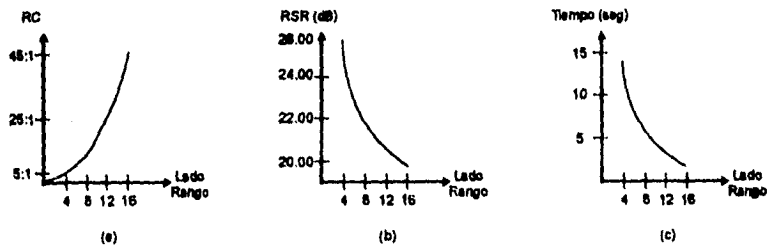


Figura 7.31 **Gráficas de (a) Razón de Compresión, (b) Relación Señal a Ruido y (c) Tiempo, respecto al Lado del Rango.**

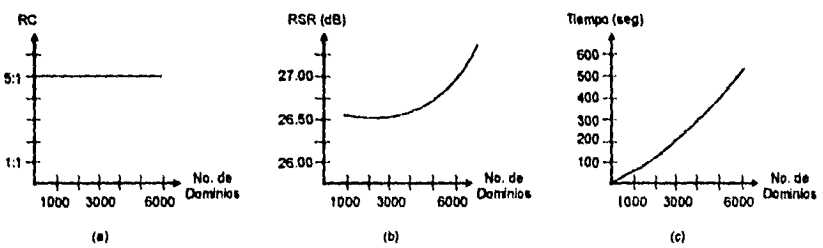


Figura 7.32 **Gráficas de (a) Razón de Compresión, (b) Relación Señal a Ruido y (c) Tiempo, respecto al Número de Dominios.**

7.4.1.3 Imagen de 256 Tonos de Grises, adquirida a 400 dpi.

La imagen de la **Figura 7.33** es una imagen típica de 256 tonos de grises adquirida con una resolución de 400 dpi. En el histograma se puede ver que la distribución de pixeles es bastante densa, comparada con los otros 2 tipos de imágenes.

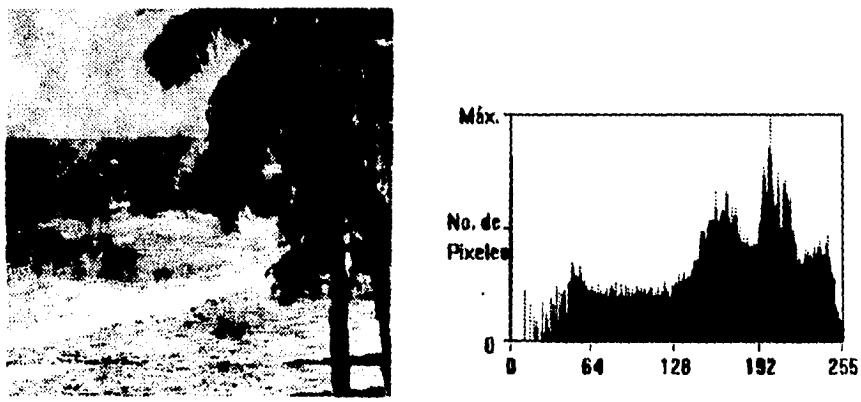


Figura 7.33 Imagen Típica de 256 Tonos de Grises Adquirida a 400 dpi y su Histograma.

Las tablas obtenidas para los diversos parámetros y variables dependientes se muestran a continuación.

- Valores de RC. RSR y Tiempo, variando el ancho de la subimagen.

Ancho Subimg.	Lado Rango	No. Dom.	RC	RSR (dB)	Tiempo (seg)
8x8	4	4	5:1	25.24	4
16x16	4	16	5:1	26.58	4
32x32	4	64	5:1	27.34	14
64x64	4	256	5:1	28.16	44
128x128	4	1024	5:1	29.08	160

- Valores de RC, RSR y Tiempo, variando el lado del rango.

Ancho Subimg.	Lado Rango	No. Dom.	RC	RSR (dB)	Tiempo (seg)
32x32	4	64	5:1	27.34	14
32x32	8	49	25:1	24.08	6
32x32	16	9	45:1	21.14	2

- Valores de RC, RSR y Tiempo, variando número de dominios.

Ancho Subimg.	Lado Rango	No. Dom.	RC	RSR (dB)	Tiempo (seg)
32x32	4	225	5:1	28.16	34
32x32	4	450	5:1	29.08	52
32x32	4	512	5:1	29.08	54
32x32	4	900	5:1	29.08	86
32x32	4	1,800	5:1	29.08	160
32x32	4	6,728	5:1	29.08	546

Las **Figuras 7.34, 7.35** y **7.36** presentan estos resultados en forma gráfica para las imágenes de 256 tonos de grises con 400 dpi.

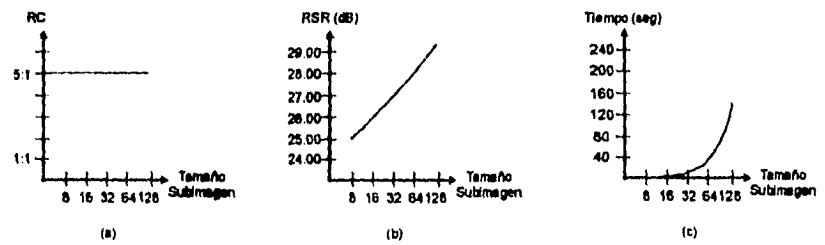


Figura 7.34 **Gráficas de (a) Razón de Compresión, (b) Relación Señal a Ruido y (c) Tiempo, respecto al Tamaño de la Subimagen.**

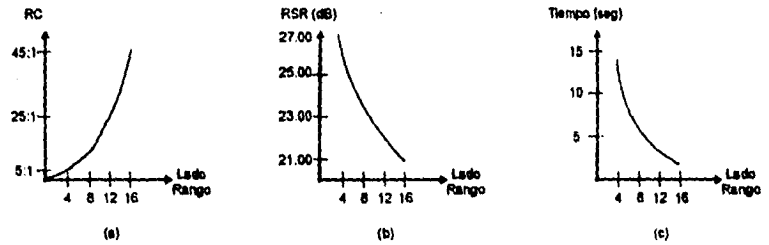


Figura 7.35 **Gráficas de (a) Razón de Compresión, (b) Relación Señal a Ruido y (c) Tiempo, respecto al Lado del Rango.**

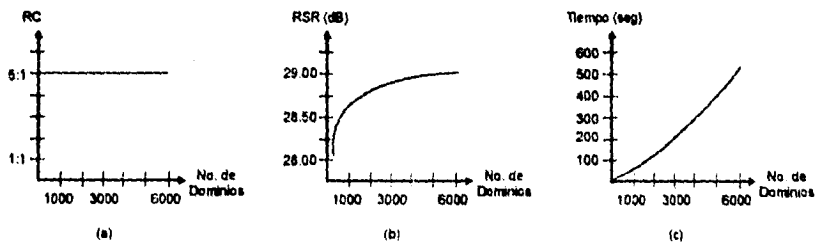


Figura 7.36 **Gráficas de (a) Razón de Compresión, (b) Relación Señal a Ruido y (c) Tiempo, respecto al Número de Dominios.**

7.4.2 Análisis.

Es necesario analizar el resultado de las prueba para llegar a las conclusiones respectivas. En las siguientes secciones se interpretan las tablas y gráficas, así como también se realiza la comparación de estas técnicas fractales con otras técnicas.

7.4.2.1 Interpretación de Tablas y Gráficas.

Comportamiento de la Razón de Compresión.

Como se puede ver en las gráficas, la razón de compresión es independiente del tamaño de las subimágenes en las cuales se divide la imagen, y del número de dominios utilizados; es obvio que la razón de compresión depende solamente del tamaño del rango puesto que este corresponde directamente a un mapa IFS de 5 bytes que se almacena en el archivo; entre menos rangos se tengan (es decir, un lado de rango más grande), más compresión se logra y viceversa

Comportamiento de la Relación Señal a Ruido.

Por otra parte, la relación señal a ruido es dependiente de todas las variables (tamaño de la subimagen, lado de rango y número de dominios utilizados); esto se debe a que todas estas variables afectan el cálculo que se efectúa para encontrar la distancia mínima entre el rango y los dominio a evaluar; esto es, aumentando el tamaño de las subimágenes se tienen más dominios a buscar y por lo tanto es factible que se encuentre uno con menor error en una subimagen grande que en una pequeña; aumentando el lado del rango aumenta la distancia entre el rango y sus dominios, y aumentando el número de dominios se tienen más dominios a buscar para cubrir al rango. Entre más grande sea la subimagen y se tengan más dominios, la calidad de la imagen comprimida se incrementa.

Comportamiento del Tiempo de Compresión.

Finalmente, el tiempo de compresión también es dependiente de todas las variables por las mismas razones descritas para la relación señal a ruido. Aquí cabe señalar que entre mayor sea la subimagen y se tengan mayor número de dominios, el tiempo de compresión aumenta exponencialmente; sin embargo, en la gráfica del comportamiento del tiempo con respecto al lado del rango, se tiene que entre mayor sea éste menor es el tiempo de codificación; esto no es totalmente cierto, este resultado se presenta debido a que se aumentó el tamaño del rango pero no se aumentó en la misma proporción el tamaño de la subimagen; si se hubiese aumentado este tamaño, el tiempo de compresión también aumenta porque son mayores los rangos a cubrir por los dominios.

7.4.2.2 Interpretación para la Imagen en 16 Tonos de Grises.

Analizando las tablas y gráficas para la imagen de 16 tonos de grises, es posible observar que generalmente este tipo de imagen no se comprime óptimamente por los cambios abruptos en los tonos de las imágenes; esto se debe a que los algoritmos de compresión fractal buscan la mínima diferencia posible entre un pixel y sus vecinos, de manera, que en el proceso de descompresión, los valores calculados de los pixeles caigan dentro de un mismo rango de tonos. Como se puede apreciar, la relación señal a ruido es totalmente pobre, sin embargo el resultado no es tan pobre como se podría esperar, tal como se muestra en la **Figura 7.37**. Esta imagen se obtuvo con los siguientes parámetros:

- Ancho Subimagen = 32
- Lado Rango = 4
- Incremento en Dominio = 4
- Simetrías = 8
- No. de Subimágenes = 16
- Dominios por Subimagen = 64
- Total de Dominios = 1024

Teniendo como resultado:

- Razón de Compresión = 5.75:1
- Relación Señal a Ruido = 16.80
- Tiempo de Compresión = 103 seg.

Como se puede apreciar en el histograma, éste es mucho más denso y diferente al de la imagen original, resultado del la interpolación de pixeles que efectúa intrínsecamente el algoritmo fractal.

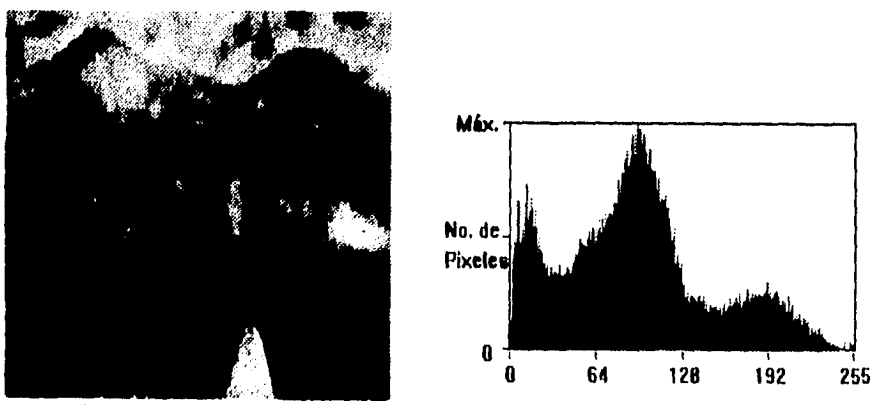


Figura 7.37 Imagen Comprimida, obtenida de la imagen en la Figura 7.25.

7.4.2.3 Interpretación para la Imagen en 256 Tonos de Grises a 100 dpi.

En el análisis de las tablas y gráficas para la imagen de 256 tonos de grises a 100 dpi se puede deducir que este tipo de imagen es medianamente aceptable para comprimir debido a que las diferencias entre los tonos de pixeles adyacentes no es muy grande, aunque la distribución de dichos pixeles no sea densa. La relación señal a ruido no es tan pobre como la imagen de 16 tonos de grises y los resultados son mucho mejor, tal como se muestra en la **Figura 7.38**

Esta imagen se obtuvo con los siguientes parámetros:

- Ancho Subimagen = 32
- Lado Rango = 4
- Incremento en Dominio = 4
- Simetrías = 8
- No. de Subimágenes = 16
- Dominios por Subimagen = 64
- Total de Dominios = 1024

Teniendo como resultado:

- Razón de Compresión = 5.90:1
- Relación Señal a Ruido = 25.24
- Tiempo de Compresión = 104 seg.

El histograma para esta imagen comprimida es más denso que para la imagen original; sin embargo, sí sigue la misma curva de distribución. También se puede notar un efecto de puntos blancos en la imagen; esto se debe a que el número de dominios escogido no fue suficiente para garantizar una buena cobertura para esos rangos.

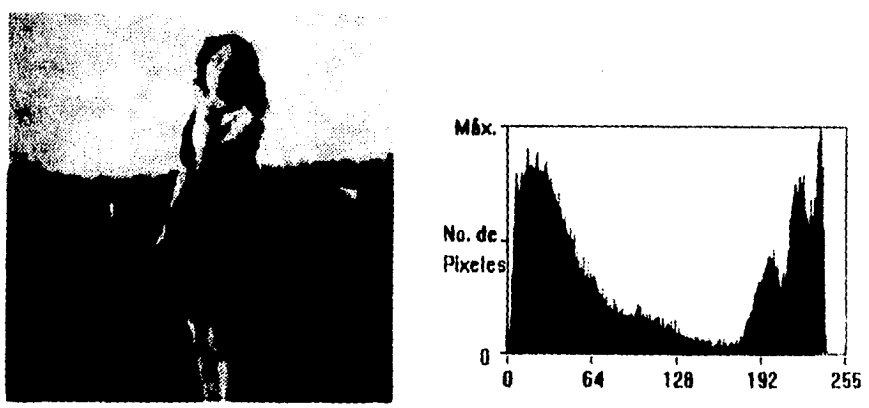


Figura 7.38 Imagen Comprimida, obtenida de la imagen en la Figura 7.29.

7.4.2.4 Interpretación para la Imagen en 256 Tonos de Grises a 400 dpi.

Como se observa en las tablas y gráficas para esta imagen, ésta es la mejor candidata para la compresión fractal porque tiene el histograma de distribución más denso, es decir, los cambios entre los pixeles de una región son suaves. La relación señal a ruido es la mejor que se obtuvo para las diversas imágenes analizadas y la imagen resultante es bastante aproximada a la original **(Figura 7.39)**.

Esta imagen se obtuvo con los siguientes parámetros:

- Ancho Subimagen = 32
- Lado Rango = 4
- Incremento en Dominio = 4
- Simetrías = 8
- No. de Subimágenes = 16
- Dominios por Subimagen = 64
- Total de Dominios = 1024

Teniendo como resultado:

- **Razón de Compresión** = 5.68:1
- **Relación Señal a Ruido** = 27.34
- Tiempo de Compresión = 110 seg.

El histograma para la imagen comprimida es muy similar al de la imagen original, lo cual denota una buena relación señal a ruido; para lograr una mejor relación se puede incrementar el número de dominios a buscar, sin embargo, esto aumenta exponencialmente el tiempo de compresión.

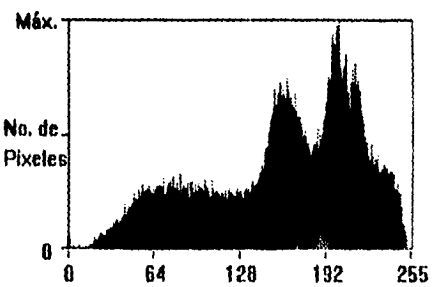


Figura 7.39 Imagen Comprimida, obtenida de la imagen en la Figura 7.33.

7.4.2.5 Observaciones Generales sobre la Compresión Fractal.

Las gráficas obtenidas durante las pruebas muestran que para comprimir fractalmente una imagen es mejor utilizar el mayor número de dominios posible; sin embargo, existen imágenes cuya relación señal a ruido no mejora aunque se aumenten el número de dominios; esto sucede principalmente con las imágenes de 16 tonos de grises. Para las imágenes de 256 tonos de grises se tiene que un 70% de las imágenes analizadas mejoran su relación señal a ruido al aumentar el número de dominios, mientras que el 30% restante no la mejora substancialmente.

Para la mayoría de las imágenes analizadas, los parámetros que dan una mejor combinación entre razón de compresión, relación señal a ruido y tiempo son:

- Ancho Subimagen = 32
- Lado Rango = 4
- Incremento en Dominio = 2
- Simetrías = 8
- No. de Subimágenes = 16
- Dominios por Subimagen = 255
- Total de Dominios = 16320

Cuyo resultados son:

- Razón de Compresión = 5.5:1
- Relación Señal a Ruido = 27.00
- Tiempo de Compresión = 340 seg.

También es importante señalar que existen imágenes en las cuales no trabaja bien el esquema de compresión; son imágenes que al descomprimirse presentan cuadros blancos debido a que no se encontró ningún dominio que cubriera de manera aceptable a un rango, aun cuando se haya escogido el mayor número de dominios posible.

Como se describió en el Capítulo 6, es posible utilizar esquemas de particionamiento que no sean tan sencillos como el implementado para este sistema; estos esquemas pueden ser el particionamiento con árbol cuádruple o el particionamiento horizontal-vertical. El particionamiento de árbol cuádruple se implementó y probó para este sistema, encontrándose que no había mejoría en la relación señal a ruido respecto al esquema de particiones sencillas, además de no aportar mejoras, el particionamiento incrementaba el tiempo de procesamiento y el tamaño del archivo resultante porque era necesario guardar la información respecto a la división de la imagen. Por todas estas razones, este esquema de partición fue descartado para la versión final del sistema.

El hecho de que utilizar un esquema de partición más complejo no conduce a una mejora significativa en la calidad de la imagen pero sí aumenta el tiempo de procesamiento, se encuentra también documentado en el trabajo de los investigadores Yuval Fisher, E. W. Jacobs y R. D. Boss; en los documentos de M. F. Barnsley no aparecen consignados estos datos.

Por otra parte, las técnicas de procesamiento digital implementadas no producen una mejora en la imagen comprimida, sino simplemente la transforman para darle otros aspecto; por ejemplo, aplicar un filtro paso altas remarca los bordes y contrastes de color pero eso no quiere decir que disminuya los errores causados por la compresión fractal; al contrario, en muchos de los casos estos errores se hacen más evidentes.

7.4.3 Comparación con otras Técnicas.

Con el análisis de los resultados, es posible ahora realizar una comparación de esta técnica de compresión con otras para evaluar su aplicación práctica.

7.4.3.1 Comparación de la Fidelidad de las Imágenes con otras Técnicas.

Debido a que este es un algoritmo para comprimir imágenes con pérdida de información, es definitivo que la calidad de las imágenes siempre sea menor que las de aquellas comprimidas sin pérdida de información con algoritmos tales como el LZW, los códigos de Huffman, RLE y otros.

En lo que respecta a su comparación con otras técnicas con pérdida de información, tales como el JPEG (basado en la transformada cosenoidal discreta), la cuantización vectorial y otras descritas en el capítulo 2, se tiene que este esquema de compresión presenta una calidad un poco menor con una relación señal a ruido de máximo 31 dB, mientras que el estándar JPEG alcanza hasta 37dB; es importante señalar que para que una imagen comprimida sea casi indistinguible de original se debe lograr una relación señal a ruido de mínimo 30 dB. Como ya se observó en las gráficas, la relación señal a ruido obtenida para la mayoría de las imágenes analizadas está por debajo de 30 dB, lo cual quiere decir que si hay diferencias perceptibles entre la imagen original y la comprimida.

7.4.3.2 Comparación de las Razones de Compresión con otras Técnicas.

La razón de compresión más común lograda es de 5:1, lo cual representa mucho más que la razón 2:1 lograda con algoritmos de compresión sin pérdida de información como LZW, códigos de Huffman, RLE y otros. Sin embargo, comparando esta técnica contra otras con pérdida de información tales como JPEG, se tiene una razón de compresión baja puesto que JPEG alcanza hasta 30:1 sin degradación notable; si bien es cierto que se puede incrementar la razón de compresión hasta 25:1 o 45:1 también es cierto que la degradación aumenta bastante para estas razones de compresión..

7.4.3.3 Comparación de la Velocidad de Compresión con otras Técnicas.

La velocidad de compresión de esta técnica es sumamente lenta debido a la complejidad de los algoritmos fractales; mientras que comprimir una imagen de 256x256 pixeles con LZW toma solo 3 segundos y con JPEG 15 segundos, comprimir esa misma imagen con la técnica fractal toma hasta 300 seg. (5 minutos) utilizando la mayor calidad posible. El tiempo de descodificación de la imagen también es grande pues consume hasta 20 seg., independientemente del tiempo de compresión.

El tiempo de compresión del sistema podría parecer exagerado; sin embargo, basándonos en las tablas presentadas por Yuval Fisher, E. W. Jacobs y R. D. Boss, los tiempos están dentro de un rango aceptable. Los tiempos que se presentan a continuación fueron obtenidos efectuando pruebas en una estación de trabajo Apollo 400 Hewlette Packard.

Tamaño Imagen	Rango	Compresión	RSR (dB)	Tiempo (seg)
512x512	4	15.95:1	33.13	1,170
512x512	4	17.04:1	33.19	3,627
512x512	4	16.74:1	33.30	3,627
512x512	4	10.49:1	35.92	6,201
512x512	4	24.62:1	30.85	2,340
512x512	8	36.78:1	30.71	8,775
256x256	4	9.09:1	30.63	164
256x256	4	9.97:1	31.53	5,265
256x256	4	11.85:1	30.58	4,329

En las notas publicadas por M. F. Barnsley no se proporcionan tablas comparativas de tiempo de compresión pero si se menciona el uso de tarjetas especiales para acelerar y optimar el software, lo cual indica que el tiempo de procesamiento del software también debe ser bastante alto.

Conclusiones

Conclusiones.

Son varias las conclusiones que pueden obtenerse del desarrollo del sistema; la primera y más importante es puntualizar que el objetivo de esta tesis fue plenamente alcanzado. *Foto Fractal*, el software desarrollado, es un sistema que incorpora las nuevas técnicas de compresión fractal, junto con la capacidad de importar y exportar imágenes de un formato comercial a otro. Se presentaron diversos problemas durante el desarrollo del sistema pero el principal fue la falta de información debido a lo reciente del tema; se comenzó con el capítulo 2 del libro *Image and Text Compression*[1], el cual presentaban los resultados del trabajo desarrollado por Yuval Fisher, E. W. Jacobs y R. D. Boss; sin embargo, la información presentada no era muy explícita en cuanto a una implementación práctica de un sistema de compresión fractal. Posteriormente, por medio de la red Internet se efectúo una búsqueda en las principales bibliotecas de Estados Unidos y Australia, encontrándose el libro *Fractal Image Compression*[2], el cual presentaba un algoritmo factible de implementarse en una microcomputadora; este algoritmo fue la base para el sistema desarrollado, del cual se ha hablado largamente en esta tesis.

Otra cuestiones que podemos concluir son que ventajas y desventajas tiene el sistema, tanto en software como en hardware, cuales son los posibles mejoras, etc.; esto se plasma a continuación.

Ventajas y Desventajas en Software.

Las ventajas que presenta el sistema en cuanto a software son las que proporciona Windows como plataforma gráfica: manejo multiventana, uso de ratón, transparencia en el uso de dispositivos periféricos como impresoras, etc.
Por otro lado, el único software comercial que existe para comprimir fractalmente una imagen es el desarrollado por M. F. Barnsley y su compañía Iterated Systems Inc., cuyo precio actual es de 700 dólares para la versión más sencilla de compresión de imágenes en tonos de grises; la venta de archivos *.obj y *.dll para utilizarlos en un sistema compresor fractal cuestan hasta 10,000 dólares, incluyendo tarjetas de hardware especial; además, Iterared Systems tiene la política de no permitir que la compañía que compró dichos archivos venda más de 15 copias de su sistema. Por este motivo, el sistema desarrollado en esta tesis tiene la gran ventaja de lograr compresiones aceptables, con calidad aceptable y en tiempos aceptables, a nivel académico.

[1] Ver bibliografía anexca.
[2] Idem.

En el aspecto de desventajas, la más obvia, aparte del largo tiempo de compresión ya discutido, es que el software no está desarrollado para utilizarse a nivel comercial por la gran cantidad de herramientas que le hace falta, tales como recortes, escalamientos, plumas, manejo de colores, etc.

Ventajas y Desventajas en Hardware.

La ventaja principal en hardware es que no se requiere ninguna tarjeta especial para comprimir imágenes, claro está que esto es a cambio de aumentar el tiempo de procesamiento.
Como ya se vió en los requerimientos de hardware, estos nos son mayores que los requerimientos de muchos paquetes actuales; sin embargo, aún existen una gran cantidad de usuarios con microcomputadoras menos potentes; esto podría ser una gran desventaja en la aceptación de este sistema.

Aplicaciones.

El sistema de compresión fractal es factible de utilizarse en sistema de procesamiento digital de imágenes, en los cuales se manejan una gran cantidad de imágenes que pueden comprimirse y que no requieren gran calidad; por ejemplo, en sistema médicos con imágenes radiológicas, de ultrasonido, etc. y en sistemas de procesamiento de imágenes vía satélite.

Otro campo fértil para el uso de este sistema son los paquetes de multimedia; en estos paquetes las imágenes se comprimen y almacenan una sola vez pero se leen muchas veces. Un ejemplo de paquete de multimedia en el cual se aplican actualmente las técnicas fractales para la compresión de imágenes, desarrolladas por Iterated Systems, es la enciclopedia Encarta de Microsoft; esta enciclopedia contiene una colección exhaustiva de artículos, animación, sonido, ilustraciones, gráficas y fotografías, toda la información está almacenada en un sólo CD-ROM de 600 Mbytes.

Crecimiento a Futuro.

El sistema desarrollado tiene un gran crecimiento a futuro por integrar estas nuevas técnicas de compresión fractal y su capacidad de exportar e importar imágenes a los formatos gráficas comerciales más populares.

Las mejoras que puede tener en cuanto a la compresión fractal en sí es la afinación de los algoritmos para que disminuya el tiempo de procesamiento sin afectar la calidad de la imagen comprimida, o bien, la creación de otros algoritmos nuevos; también se pueden probar otro tipo de particiones más elaboradas y que conduzcan a una mejor compresión con mayor calidad. Otra cuestión a mejorar es la especificación más compacta de los mapas; actualmente ocupan 5 bytes cada uno; posiblemente pudieran ocupar sólo 3.
Realmente todos estos tópicos son muy complejos y todavía están en plan de investigación en muchos institutos y centros; en noviembre de 1995 saldrá a la venta el libro *Fractal Image Compression: Theory and Applications to Digital Image Processing*, cuyo autor es el investigador Yuval Fisher, mencionado al inicio del Capitulo 6.

El sistema también puede crecer en la parte de aplicación de algoritmos del procesamiento digital de imágenes para mejorar la calidad de la imagen comprimida. Los algoritmos de filtrado que se pueden aplicar en las opciones del menú correspondiente son operaciones sencillas y por lo tanto no mejoran notablemente la imagen; se podrían implementar algoritmos más complejos de interpolación de pixeles de varios órdenes, detección de bordes, difusión de tonos y otros.

En lo referente a herramientas de trabajo para el procesamiento de la imagen, definitivamente el sistema es muy pobre; lo único que provee es una opción para reescalar una imagen a la mitad de su dimensión, permitir asociarle una paleta diferente y nada más. En este aspecto puede crecer muchísimo agregando todas las herramientas que poseen paquetes como Photo Finish, Corel Draw e inclusive el mismo Paint Brush de Windows; estas son: acercamientos, escalamiento de la imagen, cambio de paleta, manejo de plumas y aereosoles de diversos tamaños, recortes y creación de figuras geométicas, entre otras más.

Anexos

Índice Alfabético

Glosario de Términos

ADCT: *Analogic/Digital Compression Techniques*; Técnicas de Compresión analógica/Digital.

Algoritmo: Conjunto de pasos a seguir para efectuar un proceso.

Aliasing: Término en inglés para denotar las distorsiones tipo escalera que sufre una imagen cuando se disminuye o aumenta su resolución espacial y de color.

Árbol Cuádruple: Representación Espacial de Datos; se aplica principalmente para representar las regiones de una imagen por medio de un árbol compuesto de nodos, cada uno de ellos dividido a su vez en 4 nodos.

ASCII: *America Standar Code for Information Interchange*; Código Estándar Americano para el Intercambio de Información. Estándar para la representación de caracteres en modo texto.

Atractor: Objeto geométrico fractal resultado de aplicar repetidamente mapas IFS a una imagen inicial.

BMP: *Bit Map*; Mapa de Bits. Formato gráfico comercial desarrollado por Microsoft Windows.

Buffer: Localidades de memoria para el almacenamiento temporal de datos.

CAD: *Computer Assisted Design*; Diseño Asistido por Computadora. Se nombra así a las técnicas matemáticas y software desarrollado para resolver problemas de diseño de ingeniería, arquitectura y otras áreas, utilizando la computadora para ello.

CCITT: *Comité Consultantif International Telégraphique et Telephonique*; Comité Consultor Internacional de Telegrafía y Telefonía. Comité europeo para la estandarización de formatos de transmisión y comunicaciones.

Collage: Término en inglés para denotar el *pegado* de diversas partes de una imagen para recrear la imagen original.

Color: Luz compuesta de muchas longitudes de onda que producen una sensación visual.

Colorimetría: Ciencia que estudia la percepción y medición del color.

Color Real: Representación de imágenes de color utilizando 16 millones de tonos; se dice que es color real porque no hay diferencia perceptual entre la imagen del mundo real y la imagen discretizada.

DCT: *Discrete Cosine Transform*; Transformada Cosenoinal Discreta. Técnica matemática para transformar una señal discreta en el tiempo al dominio de la frecuencia discreta.

Diagrama de Cromaticidad: Diagrama para representar los colores que son perceptibles al ojo del ser humano.

Diálogo: Ventana para entrada/salida de información en Windows.

Dither: Método para disminuir los colores de una imagen cuando el hardware no es capaz de reproducirlos todos; esta disminución de colores busca que se reproduzcan todos los tonos combinando los colores disponibles. Un ejemplo de ello es disminuir una imagen de 256 colores a 16 colores.

DFT: *Discrete Fourier Transform*; Transformada Discreta de Fourier. Técnica matemática para transformar una señal discreta en el tiempo al dominio de la frecuencia discreta.

DM: *Delta Modulation*; Modulación Delta. Es una técnica para la conversión Analógica/Digital de señales e imágenes, similar al DPCM.

Dominio: Región de la imagen que sirve para cubrir otra parte (Rango) de la misma.

DPCM: *Differencial Pulse Code Modulation*; Codificación por Modulación de Pulsos Diferenciales. Es una técnica para la conversión Analógica/Digital de señales e imágenes, derivada del PCM.

EBCDIC: *Enhanced Binary Code Decimal Information Code*; Código de Intercambio Ampliado Binario Codificado a Decimal. Estándar de IBM para la representación de caracteres en modo texto.

Entropía: Es el inverso de la razón de compresión. Denota el número mínimo de bits requeridos para representar un símbolo de un alfabeto dado.

Espacio de Color: Representación conceptual gráfica del color; los espacios de color más comúnmente utilizados son RGB, HSV y HSL, definidos también en este glosario.

Espacio Métrico: Conjunto cuyos elementos son pares de puntos, con una función definida para encontrar la distancia entre ellos, de acuerdo a una métrica establecida.

Factor Gamma: Valor numérico que representa la desviación de la intensidad de un monitor con respecto a la imagen original debido a la no linealidad de los componentes electrónicos.

FFT: *Fast Fourier Transform*; Transformación Rápida de Fourier. Técnica matemática derivada de la Transformada Discreta de Fourier (DFT).

Formato Gráfico: Estándares para el almacenamiento de imágenes en archivos.

Frame: Imagen individual de una película; se puede decir que es equivalente a una fotografía. La sucesión rápida de *frames* genera una imagen animada.

Geometría Fractal: Geometría opuesta a la geometría euclidiana en la cual los objetos se caracterizan por no tener un escala o tamaño específico y definirse por medio de algoritmos recursivos en vez de fórmulas geométricas.

GIF: *Graphic Interchange Format*; Formato para Intercambio de Gráficas. Formato gráfico comercial desarrollado por CompuServe.

IFS: *Iterated Function Systems*; Sistemas de Funciones Iteradas. Técnicas matemáticas fractales para representar una imagen por medio de una serie de transformaciones geométricas de si misma.

Imagen: Conjunto de pixeles distribuidos en un plano; cada pixel representa una intensidad de color.

Imagen Binaria: Imagen cuyos pixeles toman los valores de 0 y 1 para denotar negro y blanco, respectivamente.

Imagen de Grises: Imagen cuyos pixeles toman valores en un rango definido de grises, por ejemplo, entre 0 y 255.

JPEG: *Join Photographic Expert Group*; Unión de Grupos Fotográficos de Expertos. Es un comité de estandarización de la CCITT para la compresión de imágenes fotográficas.

Luz: Forma de energía electromagnética definida por una longitud de onda específica.

LZ: Lempel-Ziv; técnica adaptativa para la compresión de datos sin pérdida de información; el nombre proviene de las iniciales de los apellidos de sus creadores.

LZW: Lempel-Ziv-Welch; técnica adaptativa para la compresión de datos sin pérdida de información; el nombre proviene de las iniciales de los apellidos de sus creadores, al igual que LZ.

HDTV: *High Definition Television*; Televisión de Alta Definición. Estándar para la transimisión de imágenes cuya resolución es más alta que la televisión normal; este estándar nació para proyectos científicos pero se está popularizando su uso en televisión comercial en Estados Unidos, Europa y Japón.

Histograma: Representación gráfica de la frecuencia de los pixeles de una imagen.

HSL: *Hue, Saturation and Lightness*; Matiz, Saturación y Obscuridad. Espacio de Color en forma de doble cono hexagonal; el color se representa igual que en el espacio HSV pero las intensidades del negro al blanco son más reducidas.

HSV: *Hue, Saturation and Value*; Matiz, Saturación e Intensidad. Espacio de color con forma de cono hexagonal en el cual el eje vertical representa las intensidades del negro al blanco y las caras del hexágono son los matices de los colores.

KLT: *Karhunen-Loeve Transform*; Transformada de Karhunen-Loeve. Técnica matemática para la transformación de bloques de pixeles de un espacio a otro de tal manera que se elimine la correlación que existe entre ellos.

Mapa: Conjunto de valores que conforman al IFS y que se aplica a un dominio para convertirlo en un rango.

Nibbles: Conjunto de 4 bits. Un byte contiene 2 nibbles.

NTSC: *National Television System Committee*; Comité del Sistema Nacional de Televisión. Estándar de video compuesto adoptado por la *National Television System Committee* de Estados Unidos. La señal de video NTSC codifica el rojo, verde, azul y la información del tiempo en una sola señal.

PAL: *Phase Alternation Line*; Línea de Alternación de Fase. Estándar de la televisión Europea, similar al estándar NTSC de Estados Unidos.

Paleta: Tabla de *lookup* para definir los valores RGB de un pixel; el valor de éste indica la entrada a la tabla.

PCM: *Pulse Code Modulation*; Codificación por Modulación de Pulsos. Es una técnica para la conversión Analógica/Digital de señales e imágenes.

PCX: Formato gráfico comercial desarrollado por Zsoft.

PIFS: *Partitioned Iterated Function Systems*; Sistemas de Funciones Iteradas Particionadas. Variante de los IFS en los cuales los mapas se aplican solamente a partes de la imagen llamadas dominios.

Procesamiento Digital de Imágenes: Conjunto de técnicas para el mejoramiento, análisis y síntesis de imágenes.

Rango: Región de la imagen la cual es cubierta por otra parte de la misma (Dominio), utilizando una transformación o mapa IFS.

Raster: Representación de una imagen en la cual ésta se forma por medio de líneas, cada una de ellas sigue físicamente a la otra.

Ray Tracing: Métodos de generación de imágenes realísticas en color real.

Razón de Compresión: Medida para la compresión de una imagen con respecto a su original.

Región: Parte específica de la imagen.

Relación Señal a Ruido: Medida de la fidelidad de una imagen procesada con respecto a su original.

Render: Reproducción de una imagen con todas sus características de brillos y matices.

RGB: *Red, Green and Blue*; Rojo, Verde y Azul. Espacio de color con forma de cubo; un punto dentro de este cubo representa el color resultado de combinar los tres colores primarios.

RLE: *Run Lenght Encoded*; Codificación de Longitud de Corridas. Estándar de compresión para imágenes monocromáticas y de color.

Tabla de Lookup: Tabla para el mapeo de valores; usualmente está compuesta por el índice de la tabla y el valor que éste representa. Un ejemplo de estas tablas son las paletas para imágenes de 256 colores.

Transformación Contractiva: Aplicación de un mapa IFS de tal manera que el rango resultante converja a un punto o región fija.

Telebrowsing: Término en inglés para denotar los sistemas de monitoreo remoto.

TGA: Formato gráfico comercial desarrollado por Targa para la representación de imágenes en color real.

TIFF: *Tag Image File Format*; Formato de Archivo de Imágenes Etiquetadas. Formato gráfico comercial desarrollado por Hewlette Packard, Microsoft y Aldus.

VQ: *Vectorial Quantizers*; Cuantizadores Vectoriales. Herramientas matemática para la compresión de imágenes por medio de bloques.

WHT: *Walsh-Hadamard Transform*; Transformada de Walsh-Hadamard. Técnica matemática para la transformación de bloques de pixeles de un espacio a otro.

YIQ: Representación intermedia del color que utiliza los componentes de luminiscencia Y, intensidad I y saturación Q (cuya combinación de estos dos últimos se conoce como cromaticidad). Es un estándar de video compuesto para una mejor transmisión de las imágenes debido a los anchos de banda menores que ocupan estas señales.

Bibliografía

1. Illumination and Color in Computer Generated Imagery.
Roy Hall.
Springer-Verlag.
U.S.A., 1987.

2. Image and Text Compression.
James A. Storer.
Kluwer Academic Publishers.
Norwell, Massachusetts U.S.A., 1992.

3. Digital Pictures. Representation and Compression.
Arun N. Netravali.
U.S.A., 1988.

4. The Science of Fractal Images.
Heinz-Otto Peitgen, Dietmar Saupe, et. al.
Springer-Verlag.
U.S.A., 1988.

5. Applications of Spatial Data Structures.
Hanan Samet.
Addison-Wesley Publishing Company.
U.S.A., 1991.

6. Windows Bittmapped Graphics.
Steve Rimmer.
Windcrest / McGraw-Hill.
U.S.A, 1993

7. Fractal Image Compression.
Michael F. Barnsley y Lyman P. Hurd.
AK Peters, LTD.
U.S.A. 1993.

8. Digital Image Processing.
Gregory A. Baxes.
John Wiley & Sons, Inc.
U.S.A. 1994.

9. Advanced Fractal Programming In C.
Roger T. Stevens.
M&T Books.
U.S.A., 1990.

10. Programming in Windows 3.1.
Tim Farrel, Runnoe Connally.
Que Corporation.
U.S.A., 1992.

11. Set Theory.
Felix Hausdorff.
Chelsea Publishing Company.
New York U.S.A., 1978.

Hemerografía

1. Construction of Fractal Objects With Iterated Function Systems.
Stephen Demko, Laurie Hodges, Bruce Naylor.
Georgia Institute of Technology
Atlanta, Georgia

Computer Graphics.
Volume 19.
Number 3.
July 22-26, 1985.

2. Harnessing Chaos for Image Synthesis.
Michael F. Barnsley, Arnauld Jacquin,
Francois Malassenet, Laurie Reuter, Alan D. Sloan.
Georgia Institute of Technology
Atlanta, Georgia

Computer Graphics.
Volume 22.
Number 4.
August 1988.