UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

"ARAGON"

DESARROLLO DE TECNICAS PARA EL REALCE DE IMAGENES

T E S I S

Que para obtener el Título de:

INGENIERO EN COMPUTACION

P r e s e n t a :

MIGUEL MIRANDA MIRANDA

Asesor: Ing. Donaciano Jiménez Vázquez

San Juan de Aragón Edo. de Méx. **1995**

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

UNIVERSIDAD NACIONAL AVENIDA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
ARAGÓN
DIRECCION

MIGUEL MIRANDA MIRANDA
P R E S E N T E .

En contestación a su solicitud de fecha 27 de septiembre del año en curso, relativa a la autorización que se le debe conceder para que el señor profesor, Ing. DONACIANO JIMENEZ VAZQUEZ pueda dirigirle el trabajo de Tesis denominado " DESARROLLO DE TECNICAS PARA EL REALCE DE IMAGENES ", con fundamento en el punto 6 y siguientes, del Reglamento para Exámenes Profesionales en esta Escuela, y toda vez que la documentación presentada por usted reúne los requisitos que establece el precitado Reglamento; me permito comunicarle que ha sido aprobada su solicitud.

Aprovecho la ocasión para reiterarle mi distinguida consideración.

ATENTAMENTE
"POR MI RAZA HABLARA EL ESPIRITU"
San Juan de Aragón, Mex., 29 de septiembre de 1994
EL DIRECTOR

M en I. CLAUDIO C. MERRIFIELD CASTRO

c c p Lic. Alberto Ibarra Rosas, Jefe de la Unidad Académica.
c c p Ing. Silvia Vega Muytoy, Jefe de la Carrera de Ingeniería en Computación.
c c p Ing. Donaciano Jiménez Vázquez, Asesor de Tesis.

CCMC'AIR'lla.

Agradecimientos.

Le estoy eternamente agradecido a la Universidad Nacional Autónoma de México, institución que me brindo la oportunidad de ser parte su comunidad y que hizo posible mi formación personal y profesional.

Igualmente agradezco el apoyo que he recibido, a lo largo de toda mi existencia, a mi familia. Mis padres, en especial, me han ayudado en los momentos más difíciles y han hecho un esfuerzo increible por fomentar el desarrollo de mis hermanos. Afortunadamente ese esfuerzo se ha visto recompensado, ya que la familia cuenta con tres profesionistas y un universitario que recien inició sus estudios.

Agradezco también a mis amigos y compañeros de la generación 90-94, con los cuales compartí cinco años de estudios, descubriendo día a día nuevas y excitantes experiencias. Juntos recorrimos el largo camino de la superación profesional y una etapa importante e inolvidable de nuestras vidas.

La gratitud se hace extensiva a los amigos que me han brindado su amistad a la largo de muchos años, haciendo a un lado las barreras que surgen, al tomar cada uno de nosotros caminos diferentes en nuestras vidas. En especial a Carlos Alberto quién leyó mi primer borrador, haciendome valiosos comentarios que contribuyeron a mejorar este trabajo. Siempre que he tenido confusiones, sus puntos de vista me han ayudado a aclarar mis pensamientos.

Brindo un agradecimiento especial a los profesores que comparten su conocimiento y con ello hacen posible el enriquecimiento cultural de sus alumnos. En especial hago un reconocimiento a mi asesor, que me brindó su completo apoyo y con ello hizo posible el desarrollo de este trabajo.

Deseo también dar las gracias a todo el personal de la gerencia de Planeación Estratégica de PEMEX Exploración y Producción, quienes mostraron un interés muy especial en la culminación de mi titulación. De no ser por su ayuda, proporcionandome los recursos necesarios, no hubiera sido posible la elaboración del presente trabajo. También quiero agradecerles la oportunidad y confianza que me brindaron al permitirme apoyarlos en el área de cómputo, iniciando con ello mi ejercicio profesional.

Contenido.

Prólogo.

El procesamiento digital de imágenes no es nuevo, comenzó en 1920 al transmitir fotografías por medio de un cable submarino que conectó a Londres con Nueva York. Desde entonces hasta nuestros días las técnicas y aplicaciones del procesamiento de imágenes han tenido un desarrollo enorme, significativamente mayor en la última década, gracias a los avances tecnológicos en ramas como la electrónica, la computación, las telecomunicaciones, la cibernética y la robótica. Sin embargo no se utiliza exclusivamente en estas especialidades, sino que a tenido una gran cantidad de aplicaciones, debido a que para los humanos nos es más fácil y más estimulante recibir información en forma visual.

Una aplicación de procesamiento digital de imágenes puede ser sencilla o compleja, dependiendo de el uso para el que este destinado. Se puede diseñar una aplicación orientada a la robótica, la cual requerirá de varios sistemas para la adquisición, procesamiento, segmentación, representación, descripción, reconocimiento e interpretación de las imágenes que perciba una cámara. Como otro ejemplo podemos crear un sistema que permita transmitir imágenes por medio de algún canal de comunicación, hacer esta tarea requeriría de sistemas de adquisición, segmentación, compresión, restauración y procesamiento de las imágenes.

El objetivo del presente trabajo es explorar diferentes técnicas para realzar ciertas características en las imágenes. Cuando una imagen sufre algún cambio debido a un proceso como rastreo, copiado o transmisión, la calidad de la imagen resultante puede ser menor que la original, quitándole características que podrían ser útiles para el usuario.

Los métodos de realce se emplean para mejorar la calidad de una imagen, de esta forma el uso de la imagen resultante será más efectivo para el usuario. También se pueden usar para suprimir o enfatizar características seleccionadas de una imagen a expensas de otras, con el objeto de obtener información difícilmente visible en la imagen original.

Las técnicas de realce tienen su fundamento en diferentes materias como lo son probabilidad, estadística, análisis de Fourier, por lo que es necesario contar con conocimientos básicos de estos tópicos.

El trabajo esta dividido en cinco capítulos.

Comienza el capítulo 1 con una introducción sobre la forma en que se representan las imágenes digitalizadas. Se exponen los pasos de un sistema de procesamiento digital de imágenes y los elementos que intervienen en su desarrollo.

El segundo capítulo proporciona los conocimientos necesarios para programar una aplicación de procesamiento de imágenes en lenguaje C, empleando una computadora personal.

El tercer capítulo expone las técnicas de realce de imágenes usando únicamente cambios en los niveles de gris en cada pixel, sin tomar en cuenta ningún otro tipo de relación.

El cuarto capítulo muestra las técnicas que analizan las relaciones espaciales de cada pixel dentro de la imagen, para realizar la transformación que realce las características deseadas.

El quinto capítulo trata sobre la transformación de la imagen al dominio de las frecuencias por medio de la transformada de Fourier, esto permite realizar operaciones en el dominio de las frecuencias, que impliquen cambios significativos en la calidad de la imagen al obtener la transformada inversa.

Finaliza la exposición con un apéndice que muestra un conjunto de programas correspondientes a las técnicas de realce desarrolladas a lo largo del trabajo.

Capítulo 1.

Introducción.

1.1 Representación digital de imágenes.

El término imagen monocromática o simplemente imagen, se refiere a una función de intensidad de luz bidimensional $f(x,y)$, donde x y y denotan coordenadas espaciales y el valor de f en cualquier punto (x,y) es proporcional al brillo (o nivel de gris) de la imagen en dicho punto. Es importante aclarar que las imágenes que se utilizarán en el presente trabajo serán monocromáticas. La fig. 1.1 ilustra la convención de ejes usado en el procesamiento digital de imágenes. A veces es útil ver la función de una imagen en perspectiva con el tercer eje que representa el brillo. Viendo en este sentido la fig. 1.1 podría aparecer como una serie de puntos activos localizados en regiones con numerosos cambios de niveles de brillo y regiones más uniformes o mesetas donde los niveles de brillo tuvieran pequeñas variaciones o fueran constantes.

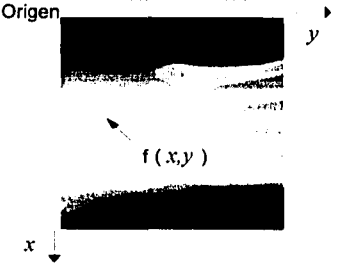


Fig. 1.1 Convención de ejes usado para la representación de imágenes digitalizadas.

Usando la convención de asignar proporcionalmente los valores más altos a áreas más brillantes, haríamos la altura de los componentes en el gráfico proporcional al brillo correspondiente en la imagen.

Para obtener una imagen digital $f(x,y)$ la imagen original se debe digitalizar en coordenadas espaciales y brillo. Una imagen digital puede ser considerada como una matriz donde los índices de renglones y columnas identifican un punto en la imagen y el valor correspondiente al elemento de la matriz identifica el nivel de gris en el punto. A los elementos de tales arreglos de imágenes son

llamados elementos de imagen, pixels o pels, los dos últimos son abreviaciones comunes de "picture elements".

Aunque el tamaño de una imagen digital varia con la aplicación, se tienen muchas ventajas si se seleccionan arreglos cuadrados con tamaño y número de niveles de gris que sean enteros potencias de 2. Por ejemplo, una imagen con un arreglo de 512 X 512 con 128 niveles de gris tiene la calidad comparable a una TV monocromática.

1.1.1 Muestreo y cuantización.

Las imágenes, de fotografías o radiografías impresas en papel o en algún otro material, son consideradas funciones continuas, en donde los niveles de gris son una función continua de la posición de la imagen. Antes de que una imagen sea procesada con ayuda de una computadora $f(x,y)$ debe ser digitalizada tanto espacialmente como en amplitud, la digitalización de las coordenadas espaciales (x,y) es llamada muestreo de la imagen, y la digitalización de amplitud es llamada cuantización de los niveles de gris.

La digitalización consiste de tomar muestras de los niveles de gris de la imagen en un arreglo de puntos de N X M. Dado que los niveles de gris en estos puntos pueden tomar cualquier valor en un rango continuo, para el procesamiento digital los niveles de gris necesitan estar cuantizados, esto significa que debemos dividir el rango de niveles de gris en K intervalos, donde el nivel de gris en cualquier punto toma uno de estos valores.

Al muestrear y cuantizar la imagen obtenemos una matriz como la mostrada en 1.1.

$$f(x,y)=\begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,M-1) \\ f(1,0) & f(1,1) & \dots & f(1,M-1) \\ \vdots & & & \\ f(N-1,0) & f(N-1,1) & \dots & f(N-1,M-1) \end{bmatrix} \qquad (1.1)$$

El lado derecho de la ecuación representa lo que comúnmente llamamos una imagen digital. Cada elemento del arreglo es referido como un elemento de la imagen, pixel o pel como fue indicado en la sección anterior.

El proceso para obtener el arreglo 1.1 es el que a continuación se explica: sea Z y R el conjunto de enteros y el conjunto de números reales respectivamente. El proceso de muestreo puede ser visto como una partición del plano xy en celdas, siendo las coordenadas de cada celda un par de elementos del producto cartesiano Z X Z (también escrito como Z^2), el cual es un conjunto de todos los pares ordenados de elementos (a,b), siendo a y b enteros de Z. Por lo tanto $f(x,y)$ es una imagen digital si (x,y) son enteros de Z X Z y f es una función que asigna un

valor de nivel de gris (que es un número real del conjunto R) a cada par de coordenadas (x,y) distintas. La cuantización consiste en la subdivisión de las amplitudes de las señales en un predeterminado número de niveles discretos de amplitud. Aplicando la cuantización a una imagen, Z reemplaza a R, y una imagen digital es entonces una imagen bidimensional (2-D) cuyos valores de coordenadas y amplitud son enteros.

Este proceso de digitalización requiere decisiones acerca de los valores para N, M y el número de niveles de gris discretos permitidos por cada pixel. Una práctica común en el procesamiento digital de imágenes es establecer estas cantidad como potencias de dos; esto es

$$N = 2^n, \quad M = 2^k \tag{1.2}$$

y

$$G = 2^m \tag{1.3}$$

donde G denota el número de niveles de gris. Usando las ecuaciones 1.2 y 1.3 podemos calcular el numero b, de bits requerido para guardar una imagen digitalizada.

$$b = N \times M \times m \tag{1.4}$$

si $M=N$,

$$b = N^2 m \tag{1.5}$$

1.2 Pasos fundamentales en el procesamiento digital de imágenes.

El procesamiento digital de imágenes abarca una amplio rango de fundamentos teóricos de hardware y software. En esta sección discutiremos los pasos fundamentales requeridos para llevar a cabo una tarea de procesamiento de imágenes.

Usaremos un ejemplo simple que seguiremos en toda la discusión, servirá para ilustrar el desarrollo del material en esta sección. Una aplicación que es muy fácil de conceptualizar sin tener ningún conocimiento previo en el uso de técnicas de procesamiento de imágenes es el de leer automáticamente las direcciones de los sobres de correo. La fig. 1.2 muestra que el objetivo es producir un resultado dentro del dominio del problema por medio del procesamiento de la imagen. El dominio del problema, en este ejemplo, son los sobres de correo y el objetivo es leer la dirección de cada pieza. Así la salida deseada en este caso es una cadena de caracteres alfanuméricos.

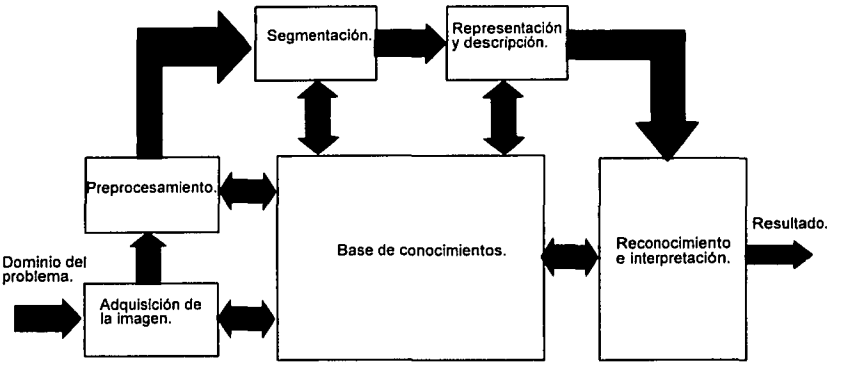


Fig. 1.2. Pasos fundamentales en el procesamiento de imágenes.

El primer paso en el proceso es la adquisición de la imagen, donde obtendremos una imagen digital. Hacer esto requiere de un sensor de imágenes y la capacidad de digitalizar la señal producida por el sensor. Como será discutido adelante, el sensor puede ser una cámara de TV monocromática o a color que produce una imagen de entrada del dominio del problema cada 1/30 seg. El sensor de la imagen puede ser una scanner de línea que produce una simple línea de la imagen en un tiempo determinado. En esta caso, el scanner de línea produce una imagen bidimensional al recorrer completamente el objeto. Si a la salida de la cámara u otro sensor de imágenes no está en forma digital, un convertidor análogo-digital lo digitalizará. La naturaleza del sensor y de la imagen producida son determinados por la aplicación. En términos de nuestro ejemplo, para la lectura del correo nos serviría un scanner de línea.

Después de que una imagen digital ha sido obtenida, el siguiente paso se encarga de preprocesar la imagen. La función principal del preprocesamiento es mejorar la imagen, con el objeto de incrementar los cambios que conduzcan al éxito de los procesos siguientes. En este ejemplo, el preprocesamiento se encarga de técnicas para realzar el contraste, remover ruido y aislar regiones cuya textura indique la posibilidad de que sea información alfanumérica.

La siguiente fase se encarga de la segmentación. La segmentación particiona una imagen de entrada en sus partes constituyentes u objetos. En general, la segmentación autónoma es una de las tareas más difíciles en el procesamiento de imágenes. Por un lado, un procedimiento riguroso de segmentación proporciona al proceso, soluciones exitosas a un conjunto de problemas en las imágenes. Por otro lado, un algoritmo de segmentación débil o errático casi siempre garantiza fallas eventuales. En términos de reconocimiento de caracteres, la función principal de la segmentación es extraer caracteres individuales y palabras del contexto.

La salida de la fase de segmentación usualmente es un dato puro: un pixel, que constituye o el límite de una región o todos los puntos de la región en si mismo. En cualquiera de los casos, es necesario convertir los datos a una forma conveniente para el procesamiento por computadora. La primera decisión que debe ser hecha es si el dato debe ser representado como un límite o como una región completa. La representación de límite es apropiado cuando nos enfocamos en características de formas externas, tales como esquinas e inflexiones. La representación regional es apropiada cuando nos enfocamos en propiedades internas, tales como textura o forma esquelética. En algunas aplicaciones, sin embargo, estas representaciones coexisten. Esta situación ocurre en aplicaciones de reconocimiento de caracteres, las cuales requieren algoritmos basados en formas de límites además de formas esqueléticas y otras propiedades internas.

Escoger una representación es solamente parte de la solución para transformar datos puros en una forma adecuada para el subsecuente procesamiento en computadora. Un método debe también ser específico al describir los datos para que sean resaltadas las características de interés. La descripción, también llamada selección de características, se encarga de extraer características para diferenciar una clase de objetos de otra. En términos de reconocimiento de caracteres, descriptores tales como lagos (agujeros) y bahías son características importantes que ayudan a diferenciar una parte del alfabeto de otra.

La última fase en la fig. 1.2 comprende reconocimiento e interpretación. Reconocimiento es el proceso que asigna una etiqueta para un objeto basado en la información proveída por sus descriptores. La interpretación involucra la asignación de un significado para un conjunto de objetos reconocidos. En términos de nuestro ejemplo, identificando un carácter como, digamos, una c requiere asociar el descriptor, para el carácter con la etiqueta c. La interpretación intenta asignar significado a un conjunto de entidades etiquetadas.

Hasta aquí no hemos dicho nada acerca de la necesidad de conocimientos previos o acerca de la interacción entre la base de conocimiento y los módulos del procesamiento en la fig. 1.2. El conocimiento acerca del dominio de un problema es codificado dentro de un sistema de procesamiento en la forma de una base de datos de conocimientos. Este conocimiento puede ser tan simple como detallar regiones de una imagen, donde la información de interés es conocida y esta ordenada para ser localizada, facilitando la interpretación que se tiene que hacer, por medio de una búsqueda en la información. La base de conocimiento también puede ser compleja, tal como una lista interrelacionada de los defectos posibles en un problema de inspección de materiales o una base de datos de imágenes conteniendo imágenes de satélites de alta resolución de una región, en relación con aplicaciones de detección de cambios. Además de dirigir la operación de cada módulo de procesamiento, la base de conocimiento también controla la interacción entre

módulos. Esta distinción es hecha en la fig. 1.2 por el uso de flechas de doble punta entre los módulos de procesamiento y la base de conocimientos, al contrario de las flechas de simple punta.

Es importante tener en mente que podemos visualizar los resultados del procesamiento de imágenes en la salida de cualquier paso de la fig. 1.2. También resaltaremos que no todas las aplicaciones de procesamiento de imágenes requieren la complejidad de las interacciones implicadas en la fig. 1.2. Numerosas aplicaciones prácticas son llevadas a cabo por las funciones provistas en la ruta exterior de la fig. 1.2. En realidad, no siempre todos estos módulos son necesitados. Por ejemplo, el realce de imágenes para interpretación visual humana va más allá de la fase de preprocesamiento. En general, funciones de procesamiento que incluyen reconocimiento e interpretación son asociadas con aplicaciones de análisis de imágenes en la cual el objetivo es automatizar, o parcialmente automatizar la extracción de información de una imagen. El reconocimiento de caracteres es un ejemplo.

1.3 Elementos de un sistema de procesamiento digital de imágenes.

A continuación se mencionarán los elementos de propósito general capaces de cumplir las operaciones de procesamiento de imágenes, que generalmente son:

1) adquisición,

2) almacenamiento,

3) procesamiento,

4) comunicación y

5) presentación de imágenes.

1.3.1 Adquisición de imágenes.

Dos elementos son requeridos para adquirir imágenes digitales. El primero es un dispositivo físico que es sensible a un ancho de banda dentro del espectro de energía electromagnética (tales como rayos x, rayos ultravioleta, luz visible o bandas infrarrojas) y que produce una señal eléctrica como salida, proporcional al nivel de energía sensado. El segundo, es llamado digitalizador, es un dispositivo para convertir la salida eléctrica del dispositivo físico sensor en una forma digital.

Como un ejemplo, consideremos un sistema básico de rayos x de imágenes. La salida de una fuente de rayos x es dirigido a un objeto y un medio sensitivo para rayos x es puesto en el otro lado del objeto. El medio entonces adquiere una imagen de los materiales (tales como huesos y tejidos), los cuales tienen diferentes grados de absorción de rayos x. El medio en sí mismo puede ser una

película, una cámara de televisión combinada con un convertidor de rayos x a fotones, o detectores discretos y cuyas salidas son combinados para reconstruir una imagen digital.

Otra categoría mayor de sensores es la que se emplea para detectar luz visible e infrarroja. Entre los dispositivos más frecuentemente usados para este propósito están los microdensitómetros, los disectores de imagen, las cámaras vidicom, y los arreglos de estado sólido fotosensitivos. El primer dispositivo requiere que la imagen a ser digitalizada sea una transparencia, el negativo de una película o una fotografía. Las cámaras vidicon y los arreglos de estado sólido pueden aceptar imágenes grabadas de esta manera, y también pueden digitalizar imágenes naturales que tengan suficiente intensidad de luz para excitar el detector.

En los microdensitómetros la transparencia o fotografía es montada en una superficie plana o es envuelta en un tambor. El rastreo se realiza recorriendo la superficie o rotando el tambor en relación a un rayo de luz (que puede ser un láser) el cual enfoca la imagen. En el caso de transparencias el rayo pasa a través de la película, en las fotografías es reflejado de la superficie de la imagen. En ambos casos el rayo es enfocado en un fotodetector y el nivel de gris de cualquier punto en la imagen es grabado por el detector basado en la intensidad del rayo. Una imagen digital es obtenida permitiendo solamente valores discretos de intensidad y posición en la salida. Aunque los microdensitómetros son dispositivos lentos, son capaces de altos grados de exactitud debido a la naturaleza esencialmente continua de la transducción mecánica usada en el proceso de digitalización.

La operación de las cámaras vidicon esta basada en el principio de fotoconductividad. Una imagen enfocada en la superficie de un tubo produce un patrón de conductividad variante, proporcional a la distribución del brillo de la imagen óptica. Un rayo de electrones independiente, enfocado con gran precisión, rastrea la parte posterior de la superficie del objetivo fotoconductivo, el cual crea una diferencia de potencial que produce en un dispositivo colector una señal proporcional a la entrada del patrón brillante, además de la posición correspondiente del rayo rastreador.

Los arreglos de estado sólido están compuestos de elementos de imágenes de silicón discretos, llamado fotositos, que tienen un voltaje de salida proporcional a la intensidad de la luz incidente. Los arreglos de estado sólido están organizados en uno de entre dos arreglos geométricos posibles: sensores de rastreo en línea y sensores de área. Un sensor de rastreo en línea consiste de una fila de fotositos y produce una imagen bidimensional por medio de movimiento relativo entre la escena y el detector. Los sensores de rastreo en línea son usados ampliamente en rastreo de imágenes en superficies planas. Un sensor de área esta compuesto de una matriz de fotositos y por lo tanto es capaz de capturar una imagen de la misma manera como, por decirlo, un tubo vidicon. Una ventaja significativa de los sensores de arreglo de estado sólido es que pueden

ser obturados a muy alta velocidad (por decirlo, 1/10,000 seg.). Esto lo hace ideal para aplicaciones donde se requiere el congelamiento de movimiento.

Son raros los sensores de rastreo en línea con rangos de resolución de 256 a 4096 elementos. Los rangos de resolución para sensores de área, son desde 32 X 32 de baja resolución, hasta los de 256 X 256 elementos de mediana resolución. Dispositivos de más alta resolución con 640 X 640 están disponibles, y sensores con resoluciones del orden de 1280 X 1280 elementos también están disponibles comercialmente a un relativamente alto, pero justificable, precio. La digitalización de imágenes es llevada a cabo por la alimentación de las señales de vídeo que generan estas cámaras a un digitalizador, como fue mencionado anteriormente.

1.3.2 Almacenamiento.

Una imagen de 8 bits de 1024 X 1024 pixeles requiere un millón de bytes de espacio para almacenamiento. Así que es siempre un reto, dar el almacenamiento adecuado a las imágenes. El almacenamiento digital para aplicaciones de procesamiento de imágenes está dividido en tres categorías principales:

(1) almacenamiento en términos de corto tiempo, para uso durante el procesamiento.

(2) almacenamiento en línea para recuperación relativamente rápida

(3) Almacenamiento para archivar, caracterizado por acceso no frecuente.

El almacenamiento es medido en bytes (ocho bits), Kbytes (mil bytes), y Tbytes (significa tera, o un trillón de bytes).

Un método de proveer almacenamiento en corto tiempo es la memoria de las computadoras. Otra es por tarjetas especializadas, llamadas frame buffers, que almacenan una o mas imágenes y pueden ser accesadas rápidamente, usualmente la velocidad de video es de 30 imágenes completas por segundo. El método más reciente permite virtualmente visualizar imágenes instantáneamente, además con scroll (cambios verticales) y pan (cambios horizontales). La cantidad de almacenamiento en una tarjeta frame buffer es limitado por su tamaño físico y por la densidad de almacenamiento de los chips de memoria usados. Es común tener 32 Mbytes de almacenamiento en una tarjeta frame buffer.

El almacenamiento en línea es generalmente realizado en discos magnéticos. Son comunes los discos Winchester con capacidades de varios Mbytes. Una tecnología más reciente, llamada almacenamiento óptico-magnético (MO), usa un láser y tecnología de materiales especializados para conseguir guardar un Gbyte de almacenamiento en una fuente óptica de 5 1/4. El factor clave que caracteriza el almacenamiento en línea es el acceso frecuente de datos; las cintas magnéticas y otros medios seriales son comúnmente usados para este fin. Los jukeboxes, que manejan entre 30 y 100 discos ópticos, proporcionan una solución efectiva a gran escala, en aplicaciones de

almacenamiento en línea que requieren capacidad de lectura y escritura. Los jukeboxes ópticos operan con el mismo principio que las rokolas musicales, en el sentido que el sistema mecánico es usado para insertar (recuperar) discos ópticos dentro de (y para) los drives ópticos.

Finalmente, el almacenamiento para archivar es caracterizado por requerimientos de almacenamiento masivo, pero infrecuente necesidad de acceso. Las cintas magnéticas y los discos ópticos son los medios usualmente empleados para esto. Las cintas magnéticas de alta densidad (6400 bytes/pulgada) pueden guardar una imagen de 1 Mbyte en cerca de 13 pies de cinta. El problema principal con las cintas magnéticas es su relativamente corto tiempo de vida - cerca de siete años- y la necesidad de controlar el medio ambiente de almacenamiento. La actual tecnología de discos ópticos de una escritura muchas lecturas (Write-Once-Read-Many WORM) puede almacenar en el orden de 1 Gbyte en un disco de 5 1/4. A diferencia de MO, están disponibles discos WORM con factores más grandes, con la capacidad de almacenar cerca de 6 Gbytes en discos de 12 pulgadas, y poco más de 10 Gbytes en discos de 14 pulgadas. Aunque no son borrables, los discos WORM tienen una período de vida que excede los 30 años sin requerimientos especiales en el medio ambiente. Cuando son guardados en un jukebox, los discos WORM pueden servir también como dispositivos de almacenamiento en línea en aplicaciones en las cuales son predominantes las operaciones de solo lectura. Un Tbyte de almacenamiento en WORM es ahora posible en un jukebox que ocupa un volumen menor que 150 pies cuadrados. Esta capacidad se traduce en un millón de imágenes de 8 bits con un tamaño de 1024 X 1024.

En aplicaciones para las cuales no es requerida la recuperación en forma digital, son muy comunes las imágenes almacenadas en forma analógica, usando principalmente película fotográfica o cintas de vídeo.

1.3.3 Procesamiento.

El procesamiento de imágenes digitales involucra procedimientos que son expresados en forma de algoritmos. De esta forma, con la excepción de adquisición de imágenes y presentación, la mayoría de las funciones de procesamiento de imágenes pueden ser implementadas con software. La única razón para adquirir hardware especializado para este fin, es la necesidad de mayor velocidad en aplicaciones que sobrepasan algunas limitaciones fundamentales de las computadoras. Por ejemplo, una aplicación importante de imágenes digitales es la microscopía con poca luz. Reducir el ruido de la imagen requiere el promedio de una imagen en base a numerosas imágenes, a razón de 30 imágenes por segundo, en la mayoría de los casos. La arquitectura de bus en todas las computadoras de alto desempeño, no pueden manejar la razón de datos requeridos para ejecutar esta operación. Por esta razón los sistemas de procesamiento de imágenes actuales son una

combinación de computadoras fuera del estándar y hardware especializado en procesamiento de imágenes, con el software corriendo en un servidor orquestando la operación total.

A la mitad de los años ochenta, numerosos modelos de sistemas de procesamiento de imágenes, fueron vendidos en todo el mundo, eran dispositivos periféricos bastante complejos que se instalaban en servidores igualmente complejos. Más tarde en los 80's y a comienzos de los 90's el mercado cambio al hardware de procesamiento de imágenes en la forma de tarjetas simples diseñadas con el objetivo de ser compatibles con la industria estándar de buses y adecuarse en gabinetes de estaciones de trabajo para ingeniería y computadoras personales. Además de la reducción de costos, este mercado cambio, sirviendo también como un catalizador para un número considerable de nuevas compañías, sobre todo en aquéllas dedicadas al desarrollo de software escrito especialmente para el procesamiento de imágenes.

Aunque los sistemas de procesamiento de imágenes a gran escala continúan siendo vendidos para aplicaciones de imágenes masivas, tales como procesamiento de imágenes de satélites, la tendencia continua hacia la miniaturización y fusión de computadoras pequeñas de propósito general equipadas con hardware para procesamiento de imágenes. En particular, el hardware principal para el procesamiento digital de imágenes que son adicionados en estas computadoras consiste de una combinación de un buffer digitalizador/frame para digitalización de imágenes y almacenamiento temporal, una también llamada unidad lógica aritmética (ALU) para la ejecución de operaciones aritméticas y lógicas en un marco amplio, y uno o dos frame buffer para acceso rápido a datos de imágenes durante el procesamiento. Una cantidad significante de software para procesamiento de imágenes básico puede ser obtenido comercialmente. Cuando es combinado con otro software para aplicaciones tales como hojas de presentación y gráficos, provee un excelente punto de inicio para la solución de problemas específicos del procesamiento digital de imágenes. Los dispositivos de visualización sofisticados, los procesadores de texto y los generadores de reportes facilitan la presentación de resultados.

El procesamiento de imágenes es caracterizado por soluciones especificas. Por consiguiente las técnicas que trabajan bien en un área, pueden ser totalmente inadecuadas en otra. La ventaja del poderoso hardware y software actual, es proveer un punto inicial mucho más amplio del que se tenía hace menos de una década (y por una fracción del costo). La solución actual de un problema específico generalmente continua requiriendo amplia investigación y desarrollo.

1.3.4 Comunicación.

La comunicación en el procesamiento digital de imágenes involucra principalmente la comunicación local entre sistemas de procesamiento de imágenes y la comunicación remota de un punto a otro, en relación con la transmisión de datos de imágenes. Hardware y software para comunicación local están actualmente disponibles para la mayoría de computadoras. Muchos libros de redes de computadoras explican claramente los protocolos de comunicación estándar.

La comunicación a través de grandes distancias presenta un reto más serio si el intento es comunicar datos de imágenes en lugar de resultados abstractos. Como puede ser evidente, las imágenes digitales contienen una cantidad significante de datos. Una línea telefónica de voz puede transmitir un rango máximo de 9600 bits/seg. De está forma para transmitir una imagen de ocho bits de 512 X 512 en este rango podría requerir de 5 minutos. Enlaces sin cables usando estaciones intermedias, tales como satélites, son más veloces, pero también su costo es considerablemente mayor. El punto es que la transmisión de imágenes completas sobre largas distancias es difícil. Las técnicas de compresión y descompresión de imágenes juegan un papel central para solucionar este problema.

1.3.5 Presentación.

Los monitores de TV a color y monocromáticos son el principal dispositivo de presentación usado en los modernos sistemas de procesamiento de imágenes. Los monitores son controlados por la salida(s) de un modulo hardware de visualización de imágenes en la computadora servidor o como parte del hardware asociado con el procesador de imágenes. La señal en la salida del módulo de presentación puede también ser alimentado en un dispositivo grabador que produce una copia dura (diapositivas, fotografías o transparencias) de la imagen vista en la pantalla. Otros medios de presentación incluye los tubos de rayos catódicos (CRT, Cathode Ray Tubes) de acceso aleatorio, y los dispositivos de impresión.

En sistemas de CRT de acceso aleatorio la posición horizontal y vertical del rayo de electrón del CRT es manipulado por la computadora la cual provee el controlador bidimensional necesario para producir una imagen como salida. En cada punto de deflexión, la intensidad del rayo es modulado usando un voltaje que es proporcional al valor del punto correspondiente en el arreglo numérico, variando desde la salida con intensidad cero para puntos que corresponden numéricamente al negro, al de intensidad máxima para puntos blancos. El patrón de intensidad de luz variable resultante es grabado por una cámara fotográfica enfocada en la superficie del tubo de rayos catódicos.

Los dispositivos de impresión de imágenes son útiles principalmente para trabajos de procesamiento de imágenes de baja resolución. Una forma simple para generar imágenes en tonos

de gris directamente en papel es usar la capacidad de impresión de una impresora de línea estándar. El nivel de gris de cualquier punto en las impresión puede ser controlado por el número y densidad de los caracteres impresos en ese punto. La selección apropiada del conjunto de caracteres cumple razonablemente bien con la distribución de niveles de gris. Otro medio común de grabar imágenes directamente en papel son las impresoras lásers, que producen imágenes con calidad aceptable.

Capítulo 2.

Programación en lenguaje C aplicada al procesamiento digital de imágenes.

2.1 El lenguaje C como plataforma para el desarrollo de aplicaciones de procesamiento digital de imágenes.

El lenguaje de programación C fue creado a principio de la década de los 70's por Dennis Ritchie. El objetivo al diseñar e implementar el compilador de C, era crear un lenguaje poderoso, elegante y flexible; que le proporcionara al programador un conjunto de operaciones elementales, con el fin de que construya sus propias herramientas, adecuadas a la resolución de una serie de problemas en particular. Por esta razón C tiene un grupo reducido de palabras claves, lo que no es una desventaja, sino al contrario, al tener pocos elementos para escribir programas, le da la capacidad de producir código ejecutable más veloz y altamente eficiente.

Las diferentes versiones de C proporcionan librerías para facilitar la tarea del programador. Las librerías son un conjunto de funciones, creadas por el desarrollador del compilador, que realiza cada una de ellas una tarea especifica. Dentro de las librerías, existe un grupo de funciones que permiten realizar operaciones de bajo nivel, equivalentes a las que realiza ensamblador; con la ventaja de que están incorporadas en un lenguaje de alto nivel. Estas funciones nos permiten tener el control directo sobre todos los componentes de la computadora, lo que lo hace ideal para el desarrollo de nuestras aplicaciones.

El lenguaje C ofrece una conversión de tipos de datos muy flexible. Permite todas las conversiones de tipos, es decir, que en la mayoría de las expresiones se pueden mezclar libremente los tipos de datos, por ejemplo enteros con reales, o enteros con caracteres. En general, los compiladores de C realizan una pequeña comprobación de error en tiempo de ejecución, como la comprobación de los límites de un arreglo. Sin embargo las comprobaciones son responsabilidad del programador. C se diseñó de esta forma, porque la comprobación en tiempo de ejecución ralentiza la ejecución del programa.

El lenguaje C permite también, la manipulación de bits, bytes y direcciones, que son los elementos básicos con los que trabaja una computadora. Esta posibilidad lo hace un lenguaje

adecuado para la programación a nivel de sistema, como las aplicaciones de procesamiento de imágenes.

La metodología de programación que emplea el lenguaje C es estructurada. En nuevas y diferentes versiones se incluyeron nuevos elementos y características diseñadas para programar con la filosofía de lenguajes orientados a objetos. Estas expansiones al C original lo convierte en un poderoso lenguaje de programación.

En este capitulo no se pretende mostrar todas las características del lenguaje C. Solamente explicaremos aquellas que son indispensables en el desarrollo de una aplicación de procesamiento digital de imágenes. Para la aplicación mostrada en el apéndice se utilizó el compilador Borland C versión 3.1 para DOS.

2.2 Modelos de memoria.

El modelo de memoria organiza la utilización del espacio en la memoria de la computadora, controla el tamaño del código y de los datos.

Borland C es capaz de compilar un programa usando seis diferentes tipos de modelos de memoria. El modelo que se use tendrá un profundo impacto en la forma en que el programa accederá los recursos del sistema.

Para entender como trabajan los diferentes modelos de memoria, daremos una breve explicación sobre la forma en que se direcciona la memoria de una computadora basada en la familia de procesadores Intel 80x86.

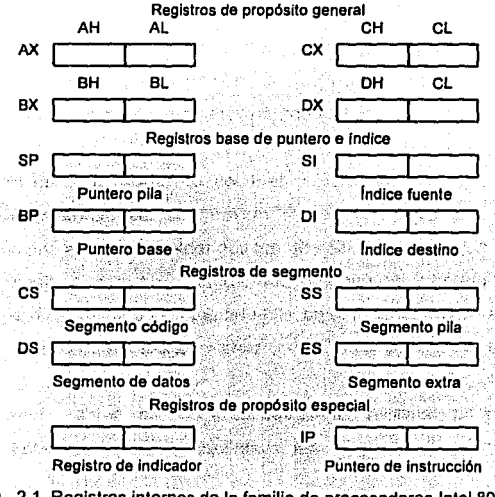


Fig. 2.1. Registros internos de la familia de procesadores Intel 80x86.

Los microprocesadores 80x86 contiene catorce registros, en donde es colocada la información a procesar o el programa de control. Los registros están organizados de la siguiente forma:

Registros de propósito general: Los 80x86 tiene cuatro registros de propósito general de 16 bits, AX, BX, CX, DX. Pueden ser utilizados en forma completa o dividiéndolos en dos registros de 8 bits cada uno, proporcionando acceso a un total de 8 registros de 8 bits. Las dos mitades de un registro se denominan baja L (Low) y alta H (High). Cuando uno de estos registros termina con la letra X, indica que se esta utilizando el registro completo de 16 bits.
En estos registros se colocan los valores que se van a procesar.
Registros de base de puntero e índice: Se encargan de las tareas de direccionamiento relativo, puntero de pila y movimiento de bloques. Estos registros son SP Puntero Pila (Stack Pointer), BP Puntero Base (Base Pointer), SI Indice Fuente (Source Index), DI Indice Destino (Destination Index). Los dos registros de 16 bits SP y BP se usan para manipular las pilas y para contener los desplazamientos en la pila actual. Los registros SI y DI son usados como valores índice para recorrer estructuras de datos complejas.
Registros de segmento: Se emplean para soportar el esquema de memoria segmentada de la familia 80x86. El microprocesador soporta cuatro módulos de código accesibles simultáneamente, llamados segmentos. Estos segmentos pueden ser direccionados por los registros de 16 bits CS Segmento Código (Code Segment), DS Segmento de Datos (Data Segment), SS Segmento de Pila (Stack Segment), ES Segmento Extra (Extra Segment). Estos segmentos se explicarán con más detalle posteriormente.
Registros de propósito especial: Consisten de registros de bandera (Flags), que almacenan el estado actual del microprocesador y el puntero de instrucción que apunta a la siguiente instrucción que se ejecutará. Estos varían en número de acuerdo al microprocesador específico que se este utilizando.

Como se menciono en los registros de segmento, la familia de procesadores 80x86 emplean una arquitectura segmentada de memoria capaz de direccionar 1 MB. El MegaByte de memoria se divide en segmentos de 6KB cada uno.

El código del programa que se encuentra en ejecución, reside en memoria y es direccionado por el registro CS. La base del segmento de datos activo es direccionado por el registro DS. Existe una pila que es utilizada para guardar resultados intermedios y llamadas a subrutinas, esta también tiene un segmento de memoria, la dirección base del segmento de pila activa se encuentra en el registro SS.

Para direccionar un elemento de la memoria, primero seleccionamos un segmento activo de uno de los cuatro contenidos en los registros de segmento CS, DS, SS, ES; posteriormente se

suministra una dirección de desplazamiento de 16 bits, para la búsqueda del elemento dentro del segmento. Es decir que primero se obtiene la dirección donde comienza el segmento; a partir de esa dirección base del segmento, nos desplazamos con una segunda dirección dentro de el segmento seleccionado. Las dos direcciones, la del segmento de 16 bits y la de desplazamiento de 16 bits se combinan para formar las dos mitades, inferior y superior, de un puntero de dirección virtual de 32 bits. Por lo tanto, la dirección de cualquier byte específico contenido en la memoria es una combinación del número de segmento y el desplazamiento de los 16 bits de orden inferior.

En los 80x86 se hace referencia a las direcciones con el formato segmento:desplazamiento.

Para tener acceso a la información almacenada en memoria dentro de un segmento cargado en uno de los registros de segmentos del microprocesador, requerimos de sólo una dirección de 16 bits. Si se quiere acceder a otro elemento de la memoria fuera de ese segmento, deben de ser cargados el registro de segmentos y el desplazamiento con los valores apropiados. Esta operación requiere de 20 bits. Sin embargo, como todos los registros tienen una longitud de 16 bits esta dirección requerirá 32 bits para guardarla. Por lo tanto, el resultado es que para acceder información fuera de un segmento actual, el procesador se tarda dos veces más en cargar dos registros de 16 bits que en cargar uno. Debido a ello, los programas se ejecutan más lentamente.

Como consecuencia, Turbo C define seis modelos diferentes de memoria, de los cuales se puede escoger el adecuado de acuerdo a la aplicación que se maneje.

Modelo Pequeñito (Tiny): Este modelo compila un programa en C de manera que todos los registros de segmento son inicializados con el mismo valor y todo el direccionamiento se realiza usando 16 bits. Este modelo de compilación produce códigos más pequeños y rápidos. Es usado para crear programas .COM

Modelo Pequeño (Small): En este modelo todos los direccionamientos se hacen usando sólo el desplazamiento de 16 bits. El segmento de código es separado de los datos, pila y segmentos extra que están en un segmento aparte. Esto nos indica que el tamaño total de un programa compilado empleando este modelo tiene 128 KB, divididos en código y datos. El tiempo de direccionamiento es el mismo que para el modelo pequeñito, pero el programa puede extenderse al doble de tamaño. Este es el modelo en que se compilan la mayoría de los programas en C.

Modelo Medio (Medium): Este modelo se emplea para grandes programas donde el código ocupa más de un segmento. Aquí el código puede usar múltiples segmentos y requiere para ello apuntadores de 32 bits. Sin embargo, la pila, datos y segmentos extra, están contenidos en un sólo segmento y usa direcciones de 16 bits. Este modelo es recomendable para programas grandes que utilizan pocos datos.

Modelo Compacto (Compact): En este modelo el código de programa está contenido en un segmento, pero los datos pueden ocupar múltiples segmentos, por tanto, todos los acceso a los

datos requieren direccionamiento de 32 bits, y el código direccionamiento de 16 bits. Se puede decir que este modelo representa el complemento del modelo medio. Es recomendable para programas que requieren gran cantidad de datos, pero poco código.

Modelo Grande (Large): Este modelo permite que el código y los datos usen segmentos múltiples. Sin embargo, el elemento único de datos más grande, como una matriz, no debe sobrepasar 64 KB. Este modelo se emplea cuando se tienen códigos y datos en gran cantidad.

Modelo Enorme (Huge): Este modelo es igual que el anterior, a excepción de que cada dato individual puede ser más grande de 64 KB. Este modelo funciona más lentamente que todos los anteriores.

Una imagen requiere de una gran cantidad de espacio en memoria para poder manipularse, por lo que es indispensable compilar todas las aplicaciones en modelo de memoria grande o enorme.

2.3 Memoria dinámica.

Los programas en C organizan la memoria de la forma descrita en la fig. 2.2.

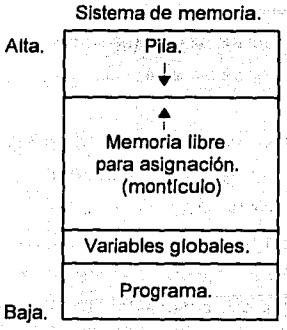


Fig. 2.2. Organización de la memoria por un programa en C.

El programa determina la cantidad de memoria que necesita la pila, esta crece hacia abajo conforme se utilice, de manera que la cantidad de memoria que necesita va determinada por el diseño del programa. Por ejemplo los programas que hacen uso de recursividad utilizará peticiones más grandes a la memoria que un programa que no la utilice, debido a que las variables locales están almacenadas en la pila, en cambio cada llamada recursiva a una función necesita espacio adicional en la pila. La memoria requerida por el programa y los datos globales es fija durante la

ejecución del programa. Cuando se realiza una petición de asignación se toma memoria de la zona de memoria libre, la cual comienza sobre la memoria de variables y crece hacia la pila. Esta región se llama montículo.

Podemos manipular la memoria contenida en el montículo con funciones contenidas en C, las cuales forman el sistema de asignación dinámica del lenguaje.

La función malloc(), pide memoria al sistema, si es capaz de concedérsela, esta porción permanece libre. La sintaxis de esta función es:

void *malloc(unsigned int tamaño);

en donde tamaño es el número de bytes que se van a asignar.

Si el sistema puede dar la cantidad de memoria solicitada, entonces devuelve un apuntador al primer byte de la zona memoria asignada. En el caso contrario, si el sistema detecta que no hay suficiente memoria disponible para satisfacer la petición de malloc(), la función devuelve un valor nulo.

Se debe de especificar un tipo explícito para que el apuntador void devuelto por malloc() sea compatible con tipo de apuntador al que se asigna. Además se debe confirmar que la función malloc(), devuelva un apuntador válido antes de usarlo.

Para hacer el cálculo del número exacto de bytes necesarios para el tipo de datos que se almacenarán hacemos uso de la función sizeof, que adicionalmente hace válido el programa para una gran cantidad de sistemas y hace más sencillo el mantenimiento cuando se cambia el objeto asignado.

La función opuesta de malloc es free(); devuelve al sistema la memoria asignada previamente:

free(void *apuntador);

Las aplicaciones de procesamiento digital de imágenes requieren espacios en memoria muy grandes. Por ejemplo una imagen de 512 X 512 pixeles, requiere para su almacenamiento en memoria de 262,144 bytes. Para asignar tamaños de memoria como estos, C nos ofrece funciones similares a las descritas con anterioridad. La función farmalloc() igual que malloc() a excepción de que permite asignar bloques de memoria mayores de 64 Kbytes. La función farfree() es la contraparte de farmalloc(). Realiza la misma tarea que free() pero para liberar bloque asignados por farmalloc().

2.4 Apuntadores.

Para aplicaciones de procesamiento digital de imágenes, se requiere de la manipulación directa de la RAM y la memoria de vídeo. Para hacer que las aplicaciones funcionen eficientemente es indispensable conocer algunas características especiales de los apuntadores, que se explican en esta sección

Un apuntador es una variable que contiene una dirección de memoria. Normalmente la dirección es la posición de otra variable en la memoria.

El formato general para la declaración de una variable apuntador es

tipo *nombre_de_variable;

Existen dos operadores especiales exclusivos de los apuntadores: * y &. El símbolo & devuelve la dirección de memoria de su operando. El símbolo * toma su operando como una dirección para obtener el contenido de dicha dirección.

En lenguaje C existe una relación estrecha entre apuntadores y arreglos. A tal grado, que podemos emplear indistintamente, indexación y aritmética de apuntadores. El compilador convierte todas las referencias a un arreglo en un apuntador a la base del apuntador. Por ejemplo, suponga que tenemos asignada cierta cantidad de memoria empleando la función farmalloc() y queremos obtener el quinto elemento de la dicha memoria, podemos hacerlo de dos formas:

p[5]

***(p+5)**

ambas devolverán el mismo valor. Note que la primera forma nos es más familiar y más sencilla de entender, por lo que es la que emplearemos a lo largo de todas las aplicaciones.

Podemos especificar el tipo de apuntador que vamos a utilizar en nuestra aplicación, de acuerdo con la cantidad de memoria que van a manipular. Para ello tenemos los modificadores de direccionamiento.

El apuntador far se emplea cuando es necesario acceder alguna región de la memoria que está fuera del segmento de datos. Una característica de los apuntadores far implementados en C es que la aritmética de apuntadores sólo afecta al desplazamiento. Es decir que cuando se incrementa un apuntador far con el valor 0000:FFFF, el nuevo valor será 0000:0000 y no 0001:0000. Por lo tanto aunque el apuntador pueda acceder a objetos fuera de su segmento, no puede acceder a más de 64 Kbytes.

El apuntador huge es igual que el far con ciertas extensiones. Su segmento está normalizado para comparar apuntadores huge. Un apuntador huge se puede incrementar cualquier número de veces. El apuntador huge no sufre el problema cíclico de los apuntadores far.

El acceso directo a la memoria de video debe ser forzosamente con un apuntador far. Además el apuntador que manipulará la imagen contenida en la memoria RAM deberá ser un apuntador huge, porque en la mayoría de los casos una imagen contiene más de 64 Kbytes de información.

2.5 Interrupciones.

Una interrupción es un tipo especial de instrucción, la cual provoca que la ejecución del programa se detenga, guardando el estado actual del sistema en la pila y salta a una rutina de manipulación de la interrupción que es determinada por el número de la interrupción. Cuando se finaliza la rutina, se realiza un retorno de interrupción que provoca que se reanude la ejecución del programa en el punto en que fue interrumpido.

Existen dos tipos básicos de interrupciones: las generadas por hardware y las provocadas por software. Las últimas son las aquí se explican, para su posterior manipulación.

Las computadoras personales están programadas con cierta cantidad de información antes de dejar la fabrica. Esta información está almacenada en la ROM (Memoria de Sólo Lectura). Contenida en la ROM existe un conjunto de rutinas que realizan operaciones de bajo nivel como el manejo de la pantalla y de los periféricos conectados a la computadora, las cuales forman el Sistema Básico de Entrada/Salida, conocido por sus siglas en inglés BIOS (Basic Input/Output System).

Los microprocesadores Intel 80x86 permiten a un programa ejecutar una interrupción del BIOS desde un programa en ensamblador o en C con la instrucción INT. Después de la instrucción se indica el número de interrupción, el cual es usado para encontrar el manejador adecuado de la interrupción. El microprocesador reserva los primeros 1024 bytes de memoria para usarlos como una tabla de vectores de interrupción. Esta tabla contiene las direcciones de los manejadores de la interrupción en la forma de segmento:desplazamiento. Por lo tanto cada dirección requiere 4 bytes. De esta forma, el microprocesador soporta 256 vectores de interrupción.

Cada interrupción es asociada con una categoría de funciones a las que accede, y estas funciones son determinadas por el contenido del registro AH. Si se requiere información adicional, se proporciona en los registros AL, BX, CX y DX.

La instrucción INT es utilizada en lenguaje ensamblador. C proporciona la función int86() que ejecuta una interrupción software. Su sintaxis es

int86(int numint, union REGS *entrada, union REGS *salida)

donde numint es el número de la interrupción, entrada es una unión que contiene los registros que se usarán para dar la información a los manejadores de la interrupción y salida es una unión que guardarán los valores devueltos por la rutina de interrupción.

La unión REGS esta definida como:

```
union REGS {
 struct WORDREGS x;
 struct BYTEREGS h;
};
struct BYTEREGS {
 unsigned char al, ah, bl, bh;
 unsigned char cl, ch, dl, dh;
};
struct WORDREGS {
 unsigned int ax, bx, cx, dx;
 unsigned int si, di, cflag, flags;
};
```

Como se puede observar, REGS consiste de una unión de dos estructuras. Con la estructura WORDREGS podemos accesar los registros completos del microprocesador, es decir los 16 bits de cada registro. Con BYTEREGS podemos accesar los dos elementos de cada registro, es decir 8 bits.

Las interrupciones son fundamentales en el desarrollo de aplicaciones de procesamiento de imágenes desarrolladas en Computadoras Personales. Son a veces el único camino de manipular directamente los componentes de la computadora.

2.6 Manejo de imágenes.

Para desarrollar una aplicación de procesamiento digital de imágenes, es necesario comenzar con el diseño de los procedimientos necesarios para manipular las imágenes que serán procesadas.

Explicamos en la introducción, que existen diferentes formas de obtener una imagen digitalizada, el resultado, después de usar alguno de los métodos de digitalización, serán archivos que contienen imágenes. La siguiente tarea es programar un medio ambiente de trabajo que nos permita manipular las imágenes contenidas en los archivos. El medio ambiente, inicialmente, debe cumplir con tres puntos:

Leer una imagen de un archivo, almacenándola en la RAM para su procesamiento.

Presentar en el monitor el contenido de la imagen almacenada en la RAM.

Guardar los cambios hechos en la imagen en un archivo.

2.6.1 Lectura de una imagen de un archivo para su almacenamiento en RAM.

Las imágenes que se manejan en el prototipo presentado en el apéndice, son imágenes que no tienen un formato especial. Están capturadas en forma binaria, es decir que cada byte de información contenido en el archivo representa un pixel. El objetivo es tomar cada uno de los bytes del archivo de una imagen y colocarlos en una región de la memoria de la computadora. Podemos cumplir este objetivo con el siguiente procedimiento:

1) Abrimos el archivo de la imagen. De esta forma sabremos si nuestro programa puede accesar la información contenida en el archivo.

2) Reservar en la memoria el espacio suficiente para la imagen. Si conocemos las dimensiones de la imagen, podemos hacer el cálculo del espacio requerido para su almacenamiento; si el sistema es capaz de proporcionarlo, nos devolverá la dirección inicial del bloque de memoria que guardamos en un apuntador; en cambio si no existe la cantidad de memoria necesaria terminará nuestro programa sin poder cumplir su propósito.

3) Colocar cada uno de los bytes del archivo en la memoria de la computadora, iniciando en la dirección que contiene el apuntador.

4) Cerrar el archivo.

2.6.2. Presentación de una imagen en el monitor.

Nuestro medio ambiente de trabajo, debe ser capaz de mostrarnos la imagen contenida en la memoria RAM en cualquier momento de la ejecución de la aplicación. Como mencionamos en la sección anterior, la imágenes que utilizamos no tienen un formato específico, por lo que no contienen información respecto a la paleta de colores que debe emplear. Sin embargo utilizaremos únicamente imágenes que fueron digitalizadas en escala de grises; entonces nuestro trabajo debe enfocarse en construir la paleta de grises, para que la imagen sea mostrada adecuadamente.

Esto lo podemos hacer con la siguiente metodología.

1) Ponemos el video de la computadora, en el modo adecuado para desplegar imágenes. Para este paso usamos la interrupción 10H (INT 10H), que manipula el modo de video. Como nuestro objetivo es poder trabajar con una resolución de 320 X 200 pixeles y una paleta de colores de 256 elementos, debemos darle al registro AX del microprocesador, los valores correspondientes para el modo de video seleccionado. El registro AH contendrá el valor 0. El registro AL será 13H (0x13 en notación hexadecimal de C).

2) Construimos la paleta de colores para que muestre imágenes en tonos de gris. La paleta de colores es un conjunto de registros localizados en la memoria de video, donde son almacenados los colores que son utilizados por la imagen. Los valores de los pixeles, contenidos en el archivo de la

imagen, son números que indican el registro, donde esta almacenada la combinación necesaria para crear el color que tendrá el pixel.

Para hacer más fácil la creación y combinación de colores en una computadora, se emplean modelos de colores. Fundamentalmente, un modelo de color consiste de un subespacio contenido en un sistema coordenado de 3 dimensiones, donde se representa cada color por un punto.

Los modelos usados comúnmente en la práctica son:

RGB (Red, Green, Blue): En este modelo los colores se forman por la combinación de tres colores básicos, Rojo, Verde y Azul.

CMY (Cyan, Magenta, Yellow): En este modelo los colores se forman por la combinación de tres colores básicos, Cían, Magenta y Amarillo.

YIQ: Este modelo es el estándar para los receptores de TV. La Y corresponde a la luminancia; I y Q son dos componentes cromáticas llamadas cromancia (Inphase) y cuadratura (Quadrature) respectivamente.

HSI (Hue, Saturation, Intensity): En este modelo cada color se define por tres características; tinte, saturación e intensidad.

Rebasa el objetivo de este trabajo el procesamiento de imágenes en color; por lo que solamente daremos una breve explicación del modelo RGB, que se emplea al formar la escala de grises en nuestra aplicación.

En el modelo RGB, cada color está formado por la combinación de tres componentes: Rojo, Verde y Azul; además se basa en el sistema de coordenadas cartesianas.

El subespacio de color que es de nuestro interés, es el cubo mostrado en la fig. 2.3 en el cual los valores Rojo, Verde y Azul están en tres esquinas, el Cían, Magenta y Amarillo están en otras tres esquinas. El negro es el origen y el blanco es la esquina más lejana al origen. En este modelo la escala de grises se extiende, del blanco al negro, a través de una línea que une estos dos puntos. Los colores son puntos contenidos dentro o sobre el cubo y son definidos por vectores que inician en el origen. Por conveniencia, se puede establecer que todos los valores de color han sido normalizados para que el cubo mostrado en la fig. 2.3 sea un cubo unitario. Esto es, que todos los valores de Rojo, Verde y Azul estén contenidos en el rango [0,1].

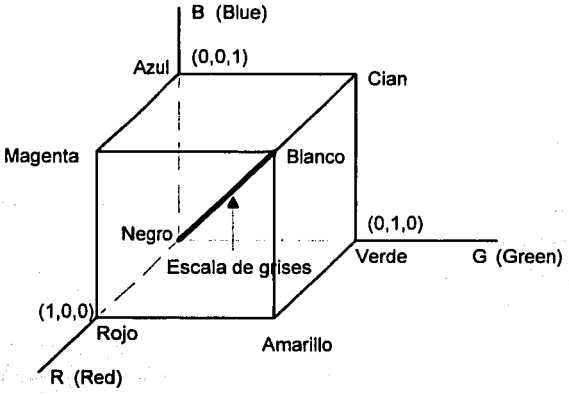


Fig. 2.3 Cubo de color RGB. Los puntos a lo largo de la diagonal principal corresponden a la escala de grises.

Podemos conceptualizar la paleta de colores como un registro de 256 elementos, cada uno compuesto de 3 elementos: uno para el Rojo, otro para el Verde y otro para el Azul, como se muestra en la fig. 2.4

Fig. 2.4 Conceptualización de la paleta de colores.

A partir del cubo unitario del modelo RGB mostrado en la figura 2.3 podemos definir la escala de grises como un conjunto de vectores con la característica de que sus 3 componentes, Rojo, Verde y Azul, tengan exactamente la misma longitud. Entonces la paleta de grises la podríamos formar con número consecutivos desde el cero hasta el 255. Sin embargo el valor del blanco de una PC en el modo 13H corresponde al valor 63, el valor 64 representa nuevamente el color negro. Si

proseguimos con la numeración después del 63, se repiten los niveles de grises definidos en el intervalo [0,63]. Esto nos obliga a cambiar la definición de nuestra paleta de colores, a la mostrada en la fig. 2.5.

	Rojo	Verde	Azul
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	1	1	1
5	1	1	1
6	1	1	1
7	1	1	1
.	.	.	.
.	.	.	.
.	.	.	.
251	62	62	62
252	63	63	63
253	63	63	63
254	63	63	63
255	63	63	63

Fig. 2.5 Llenado de la paleta de colores para la aplicación.

Es decir que cada conjunto de cuatro registros contendrá el mismo valor de gris, con el objetivo de ocupar completamente la paleta de colores, para permitir la mejor distribución de pixeles dentro de la escala de grises.

Para accesar la memoria de video, desde nuestra aplicación en C, utilizamos la interrupción del BIOS 10H (INT 10H) en donde le daremos los siguientes valores a los registros internos del microprocesador:

AH = 0x10

AL = 0x10

BX = Número de registro dentro de la paleta de colores.

CH = Valor del componente Verde.

CL = Valor del componente Azul.

DH = Valor del componente Rojo.

3) Inicializamos el valor de un apuntador a la posición inicial de la memoria de vídeo.

La dirección donde comienza la memoria de video en las PC con tarjeta VGA es la 0xA000. Para poder accesarla debemos crear un apuntador tipo far. El lenguaje C incluye una función para este propósito.

MK_FP(Dirección, Desplazamiento).

Con esta función podemos crear el apuntador a la memoria de vídeo de la siguiente forma:

char far *memvideo = MK_FP(0xA000, 0x0000)

4) Colccamos el valor de cada byte contenido en la RAM en la memoria de vídeo.

2.6.3. Almacenamiento de una imagen en un archivo.

Cuando cambiamos la apariencia de una imagen utilizando técnicas de realce, es deseable que podamos grabar la nueva imagen en un archivo. Utilizando la siguiente metodología cumplimos este propósito:

1) Abrir un nuevo archivo de tipo binario de lectura.

2) Colocar cada uno de los bytes contenidos en la memoria RAM en el archivo.

Capítulo 3.

Realce aplicando operaciones sobre puntos.

Iniciamos el estudio de las técnicas de realce de imágenes con los métodos de procesamiento que están basados solamente en la intensidad de los pixeles dentro de la escala de grises.

3.1 Modificación de escala de grises.

Una simple, pero sorprendentemente poderosa clase de operaciones de realce, involucra la modificación de la escala de grises en la imagen dada.

3.1.1 Negativo de una imagen.

Los negativos de una imagen digital son útiles en numerosas aplicaciones, por ejemplo para mostrar imágenes médicas, para fotografiar una pantalla con película positiva monocromática, con la idea de usar el negativo resultante como una imagen normal. El negativo de una imagen digital es obtenido usando la transformación $s=T(r)$ mostrada en la fig. 3.1(a) donde L es el número de niveles de gris. La idea es invertir el orden de negro a blanco para que la intensidad de la imagen resultante se decremente conforme la intensidad de la imagen de entrada se incremente. Por ejemplo para una imagen con 256 niveles de gris, la función de transformación es:

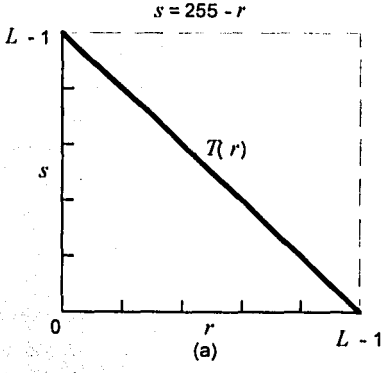


Fig. 3.1. Obteniendo el negativo de una imagen: (a) la función de transformación de niveles de gris.

(b) (c)

Fig. 3.1. (continuación) (b) una imagen (c) su negativo. En (a), r y s denotan la entrada y salida de los niveles de gris respectivamente.

3.1.2 Ajuste del contraste.

El bajo contraste en las imágenes puede ser resultado de una deficiente iluminación, la carencia de un rango dinámico adecuado en el sensor de imágenes, o una incorrecta apertura de las lentes durante la adquisición de la imagen. La idea detrás del ajuste del contraste es incrementar el rango dinámico de los niveles de gris en la imagen a ser procesada. La fig. 3.2(a) muestra una transformación típica usada para ajuste de contraste. Las localizaciones de los puntos (r_1,s_1) y (r_2,s_2) controla la forma de la función de transformación. Por ejemplo, si $r_1 = s_1$ y $r_2 = s_2$, la transformación es una función lineal que no produce cambios en los niveles de gris. Si $r_1 = r_2$, $s_1 = 0$ y $s_2 = L-1$, la transformación llegará a ser una función umbral que crea una imagen binaria. Valores intermedios de (r_1,s_1) y (r_2,s_2) producen varios grados de amplitud en los niveles de gris de la imagen de salida, afectando su contraste. En general se establece que $r_1 \leq r_2$ y $s_1 \leq s_2$ para que la función se incremente monótonamente. Esta condición conserva el orden de los niveles de gris, previniendo de esta forma, la creación de desajustes de intensidad en la imagen procesada. La fig. 3.2(b) muestra una imagen a 8 bits con bajo contraste, la fig. 3.2(c) muestra el resultado del ajuste del contraste, y la fig. 3.2(d) muestra el resultado del umbral de esta imagen, donde la salida toma el valor de 255 (blanco), para cualquier nivel de gris mayor que 128 en la imagen de entrada, y 0 (negro) para todo los demás valores.

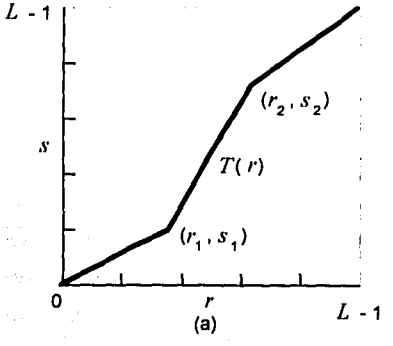
En lugar de especificar una función para cada una de las rectas que proporcione la escala de grises deseadas, como se muestra en la fig. 3.2 (a), podemos establecer una función de

transformación que permita expandir o comprimir el nivel de grises. De esta forma podremos obtener la imagen deseada con la aplicación sucesiva de esta transformación.

Supongamos que, en la imagen dada f, los niveles de gris están contenidos en $[a,b]$ y queremos que ocupe el rango $[r_1,r_k]$, podemos establecer la ecuación 3.1

$$s = \frac{r_k - r_1}{b-a}(r-a) + r_1 = \frac{r_k - r_1}{b-a}r + \frac{r_1 b - r_k a}{b-a} \qquad (3.1)$$

esta simple transformación lineal de escala de grises ajusta y cambia la escala para que los pixeles ocupen completamente el rango $[r_1,r_k]$. Como un ejemplo, de la aplicación sucesiva de 3.1, podemos obtener la imagen umbral, es decir una imagen binaria, utilizando la transformación de la ecuación 3.1 para que ocupe el rango [0,1]. Con esto obtenemos una imagen con dos niveles de gris, como consecuencia la imagen es demasiado obscura para ser visualizada. Si ahora a esta imagen resultante le aplicamos nuevamente la transformación de la ecuación 3.1 para que comprenda el rango [0,255] obtenemos la imagen umbral con dos valores de pixeles únicos: 0 y 255, la cual es justamente la imagen deseada.




(b)

Fig. 3.2 Ajuste de contraste: (a) Forma de la función de transformación; (b) Una imagen con bajo contraste; (c) resultado del ajuste del contraste; (d) resultado de la imagen umbral.

(c)

(d)

Fig. 3.2 (continuación) (c) resultado del ajuste del contraste; (d) resultado de la imagen umbral.

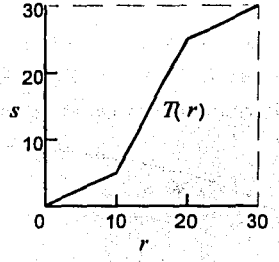
Una aproximación similar puede ser usado si el mayor número de niveles de gris de la imagen dada cae en el subrango $[a,b]$. En esta caso, podemos usar la transformación

$$s=\begin{cases}\frac{r_k-r_1}{b-a}(r-a)+r_1 & \text{para } a\le r\le b\\ r_1 & \text{para } r<a\\ r_k & \text{para } r>b\end{cases}$$

Esta transformación lineal, ajusta el intervalo $[a,b]$ de la escala original de gris, pero se comprimen los intervalos $[r_1,a]$ y $[b,r_k]$. En realidad los colapsa a simples puntos. Esto puede ser tolerable, si muy pocos puntos tienen niveles de gris en esos intervalos; de manera que solamente una pequeña cantidad de información sea perdida en la compresión.

Más generalmente, podemos extender regiones seleccionadas de la escala de grises, con el costo de compactar otras regiones; si nuestro objetivo es obtener detalles en las regiones extendidas, sin que represente un inconveniente la pérdida de información en las regiones comprimidas. Como un ejemplo simple, suponga que el rango de escala de grises es [0,30], entonces la transformación

$$s=\begin{cases} r/2 & \text{para } r \le 10 \\ 2r-15 & \text{para } 10 \le r \le 20 \\ (r/2)+15 & \text{para } 20 \le r \le 30 \end{cases}$$



comprime la escala de grises por un factor de r en los rangos [0,10] y [20,30] (en el original), mientras lo expande por un factor de 2 en el rango [10,20].

Podemos implementar cualquier transformación matemática deseada $s=t(r)$ (cuadrática, logarítmica o completamente arbitraria) sujeta solamente a la restricción de que el resultado caerá en los rangos permisibles $[r_1,r_k]$, es decir, que $r_1 \le t(r) \le r_k$ para todo $r_1 \le r \le r_k$. Esto siempre puede ser cumplido incorporando un cambio apropiado y un factor de escala en la transformación. Podemos hacer esto como se muestra a continuación: dada cualquier $T(r)$, ahora $T_1 = \min t(r)$ y $T_k = \max t(r)$ para $r_1 \le r \le r_k$. Entonces la transformación modificada T' definida por la ecuación 3.2

$$T'(r) = \frac{r_k - r_1}{T_k - T_1}\left[T(r) - T_1\right] + r_1 \qquad (3.2)$$

satisface $r_1 \le T'(r) \le r_k$ para todo $r_1 \le r \le r_k$.

3.1.3 Compresión de un rango dinámico.

A veces el rango dinámico de una imagen procesada excede la capacidad del dispositivo de salida (monitor) en cuyo caso solamente las partes brillantes de la imagen son visibles en la pantalla. Una forma efectiva de comprimir el rango dinámico de los valores de pixeles es ejecutar la transformación de intensidad de la ecuación 3.3

$$s = c\log\left(1+|r|\right) \qquad (3.3)$$

donde c es un parámetro y la función logaritmo ejecuta la compresión deseada. La figura 3.3(a) muestra la forma de esta función de transformación. La fig. 3.3(b) muestra un espectro de Fourier con valores en el rango $[0, R] = [0,512]$. Si esta función se transformara con la ecuación 3.1 para mostrarse en un sistema de 8 bits, los valores más brillantes dominarían la pantalla.

En este caso, los valores de $\log(1+|r|)$ están dentro del rango [0,2.7]. Queremos escalar este rango dentro de [0,L-1]=[0,255] para mostrarlo en el mismo sistema de 8 bits, esto se consigue seleccionando el factor de escala c=255/2.7. La figura 3.3(c) muestra el resultado después de la transformación y escala. Note el significante incremento en los detalles visibles.

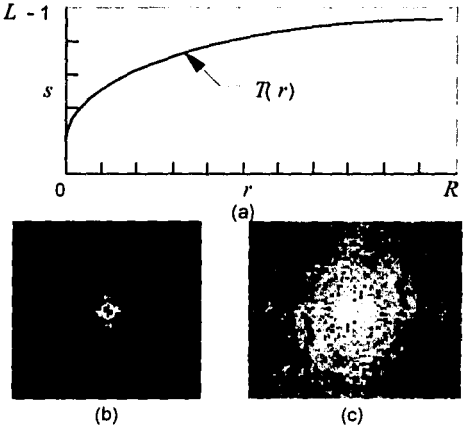


Fig. 3.3 Compresión de un rango dinámico: (a) función de transformación logarítmica; (b) imagen con un rango dinámico grande; (c) resultado después de la transformación.

3.1.4 División por planos de bits.

En lugar de elevar los rangos de intensidad de luz, puede desearse darle más brillo a la apariencia de la imagen por medio de bits específicos. Suponga que cada pixel en una imagen es representado por 8 bits. Imagine que la imagen está compuesta por 8 planos de 1 bit, representados por el plano 0 para el bit menos significativo al plano 7 para el bit más significativo. En términos de bytes, el plano 0 contiene todos los bits de orden más bajo en los bytes incluyendo los pixeles en la imagen y el plano 7 contiene todos los bits de mayor orden. La fig. 3.4 ilustra esta idea, ya que la fig. 3.4 muestra los varios planos de bits para la imagen mostrada en la fig. 3.3(c). Note que solamente los cinco bits más altos contiene datos visualmente significativos. Los otros

planos de bits contribuyen a dar detalles más refinados a la imagen. Note también que la imagen del plano 7 corresponde exactamente a la imagen umbral mostrada en la figura 3.2(d).

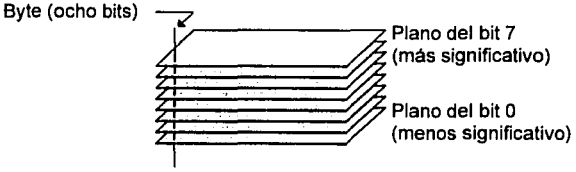


Fig. 3.4. Representación de planos de bits de una imagen digital de 8 bits.

7	6
5	4

Fig. 3.5 Planos de bit de la imagen de la figura 3.3(c). El número en los cuadros pequeños identifica los planos. El plano 7 contiene los bits más significativos.

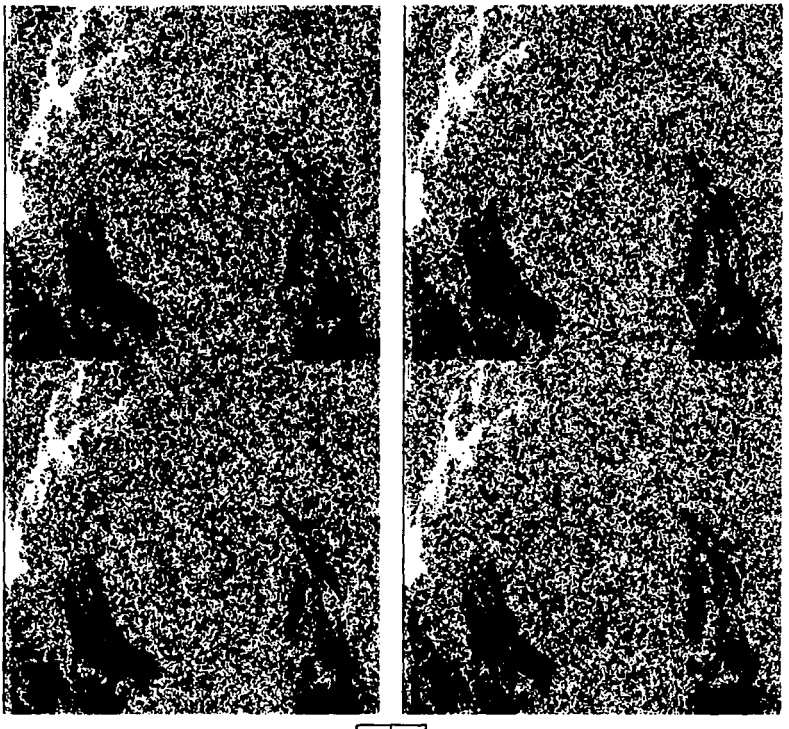


Fig. 3.5 (continuación) Planos de bit de la imagen de la figura 3.3(c). El número en los cuadros pequeños identifica los planos. El plano 0 contiene los bits menos significativos de los pixeles de la imagen original.

3.2 Modificación del histograma.

3.2.1 Histograma.

El histograma de una imagen digital con niveles de gris definidos en el rango [0, L-1] es una función discreta expresada por la ecuación 3.6

$$p(r_k) = \frac{n_k}{n} \qquad (3.6)$$

donde r_k es el k-ésimo nivel de gris, n_k es el número de pixeles en la imagen de cada uno de los niveles de gris, n es el número total de pixeles en la imagen y k = 0,1,2, ... , L-1. El histograma se puede conceptualizar como un vector que contiene el mismo número de elementos que los niveles de cuantización, en cada elemento, es guardado el número de pixeles que tienen el valor de gris correspondiente al elemento del vector.

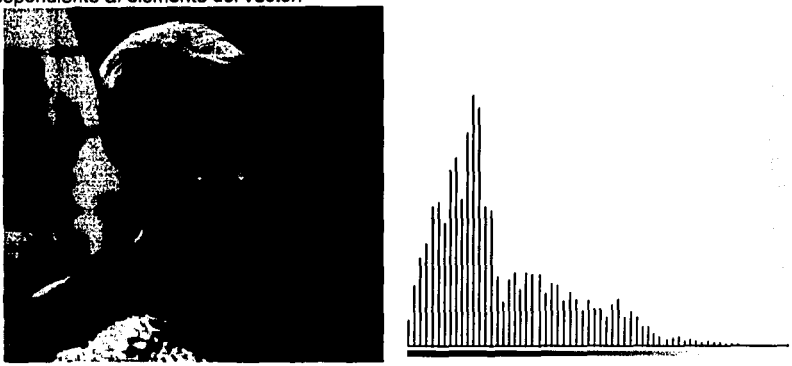


Fig. 3.6 Imagen digital y su histograma.

La función $p(r_k)$ nos da una estimación de la probabilidad de aparición del nivel de gris r_k. Una gráfica de esta función para todo los valores posibles de k nos da una descripción global de la apariencia de una imagen. Por ejemplo, la fig. 3.7 muestra los histogramas de cuatros tipos básicos de imágenes. El histograma mostrado en la fig. 3.7(a) muestra que los niveles de gris están concentrados en el extremo oscuro de la escala de grises; este histograma corresponde a una imagen oscura. Sucede lo contrario en la fig. 3.7(b). El histograma mostrado en la fig. 3.7(c) tiene una forma delgada, que indica un rango dinámico pequeño y por lo tanto corresponde a una imagen con bajo contraste. Como todos los niveles de gris están ubicados en la mitad de la escala de grises, la imagen aparecerá con grises oscuros. Finalmente, la fig. 3.7(d) muestra un histograma con extensión significante, corresponde a una imagen con alto contraste.

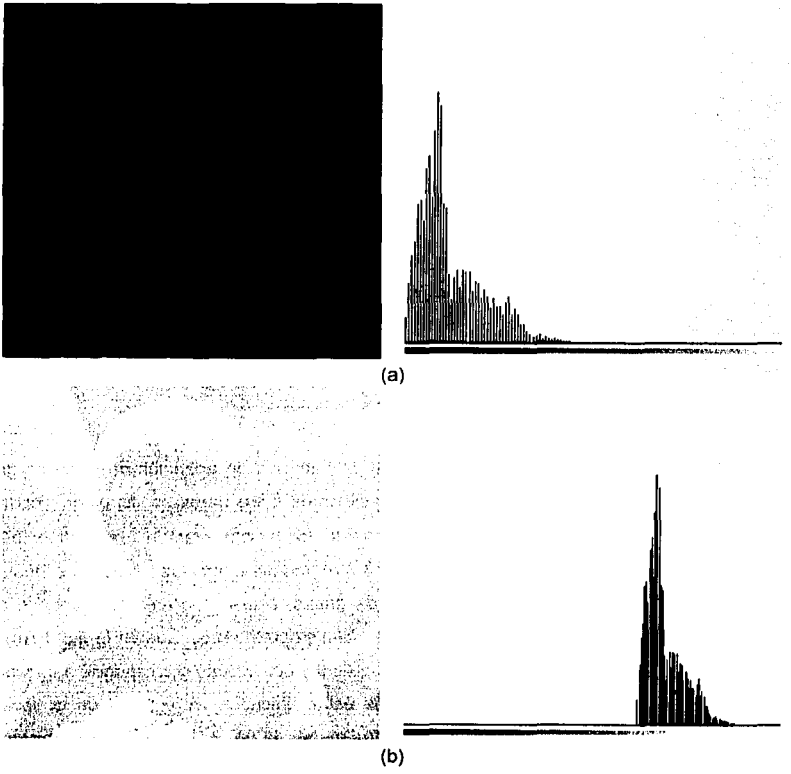


Fig. 3.7 Histogramas correspondientes a dos tipos básicos de imágenes; (a) imagen obscura; (b) imagen brillante.

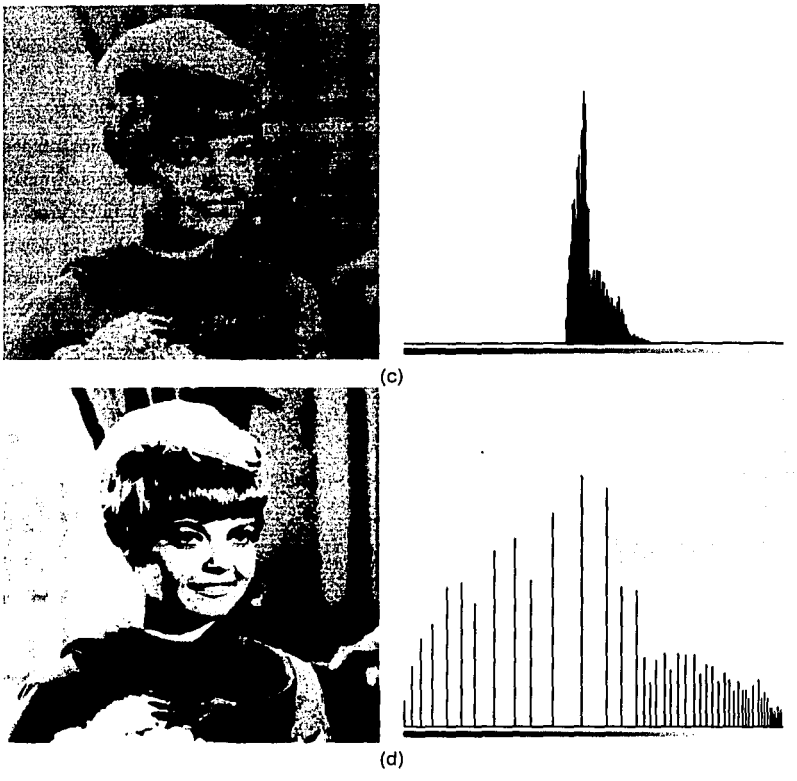


Fig. 3.7 (continuación) Histogramas correspondientes a dos tipos básicos de imágenes; (c) imagen con bajo contraste; (d) imagen con alto contraste.

3.2.2 Especificación del histograma.

El objetivo de la técnica de modificación del histograma es reescalar la imagen para que el histograma de la imagen realzada siga una forma deseada.

El proceso de especificación del histograma es una transformación de la forma $s=T(r)$ en donde el problema principal es obtener un mapeo que permita transformar el histograma para que la imagen tenga una distribución de probabilidad aproximada al histograma deseado.

Para obtener el mapeo que transforme el histograma de la imagen y como consecuencia la apariencia de la imagen podemos emplear el siguiente algoritmo.

1. Primero obtenemos el histograma de la imagen de entrada el cual denotamos como H_f

2. Definimos cual es la forma que deseamos darle al histograma, esto lo podemos obtener por medio de una función. Para el ejemplo que se proporciona en el presente trabajo se emplea una función de distribución normal, porque con ella podemos controlar el brillo y el contraste de la imagen, especificando la media y la varianza respectivamente, además la función de distribución normal es simétrica con respecto a la media lo que permite crear una imagen con tonalidades de gris balanceados. A este histograma lo denotamos como H_d

3. Calculamos los histogramas acumulados de la imagen de entrada y del histograma deseado. Definimos el histograma acumulado de H' como:

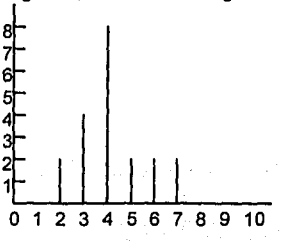
$H'(0) = H(0)$

$H'(n) = H'(n\text{-}1)+H(n)$

4. El mapeo que aproxime el histograma de la imagen con el histograma deseado se define como $m(r)=n$

donde n es tal que $H'_d(x)$ sea el valor más próximo a $H'_f(x)$.

Para visualizar el uso del algoritmo mostraremos un ejemplo sencillo. Supongamos que de una imagen obtenemos el siguiente histograma, en donde el rango de niveles de gris es [0,10].



El efecto que deseamos darle a la imagen, es hacerla mas brillante y con poco contraste. Para conseguir nuestro objetivo definimos un histograma con la forma de distribución normal con valores de $\mu=8$ y $\sigma^2=1$, con lo que obtenemos la siguiente tabla para valores enteros entre uno y diez.

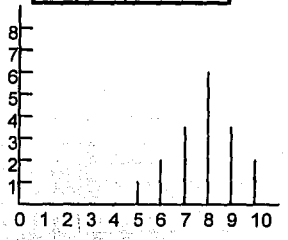
n	$f_d(n)$
0	0.0000006349117
1	0.0000134985669
2	0.0000348132629
3	0.0005445710576
4	0.005166746339
5	0.02973257231
6	0.1037768744
7	0.2196956447
8	0.2820947918
9	0.2196956447
10	0.1037768744

Dado que requerimos que los histogramas tengan el mismo número de elementos para una comparación adecuada, multiplicamos estos resultados con el número total de pixeles que deberá contemplar este histograma y redondeando los resultados obtenemos

n	$f_d(n)$	$f_d(n)$ acumulado
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
5	1	1
6	2	3
7	4	7
8	6	13
9	4	17
10	2	19

Al acumular el histograma observamos que el número total de elementos no es igual a veinte, el cual es el número de pixeles de la imagen de entrada. Para que nuestro algoritmo obtenga el mapeo correcto necesita que los histogramas acumulados contemplen el mismo número de pixeles. Para ajustar nuestra tabla obtenida podemos aplicar la expresión $H_d(n) = H_d(n) * 20 / H(19)$

n	$f_d(n)$	$f_d(n)$ ajustado
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
5	1	1
6	3	3
7	7	7
8	13	13
9	17	17
10	19	20



Encontramos la tabla de mapeo como se explica en el punto 4 del algoritmo.

n	Hi	Hd	m
0	0	0	0
1	0	0	0
2	2	0	6
3	6	0	7
4	14	0	8
5	16	1	9
6	18	3	9
7	20	7	10
8	20	13	10
9	20	17	10
10	20	20	10

Para visualizar la transformación graficamos el resultado del mapeo en la fig. 3.8

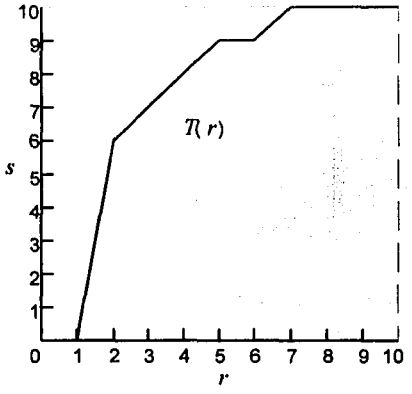
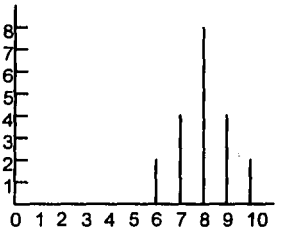


Fig. 3.8 Transformación necesaria para conseguir que el histograma tenga una forma aproximada a la distribución normal.

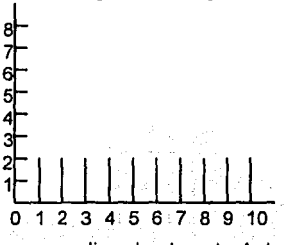
Aplicando el mapeo encontrado en la imagen original obtenemos

n	$f_{eq}(n)$
0	0
1	0
2	0
3	0
4	0
5	0
6	2
7	4
8	8
9	4
10	2

y su histograma correspondiente es



Cuando se aplica el algoritmo para obtener función de distribución uniforme se dice que se que la imagen pasa por el proceso de ecualización del histograma. Por ejemplo para ecualizar el histograma del ejemplo anterior, primero obtenemos el histograma que deseamos obtener. Sabemos que nuestro histograma deseado debe de contemplar el mismo número de pixeles que el histograma de la imagen original entonces dividiendo 20/10 nos da el valor de cada uno de los valores de grises, por lo que obtenemos el siguiente histograma.



Acumulando ambos histogramas y aplicando el punto 4 de nuestro algoritmo obtenemos el mapeo

n	Hi	Hd	m
0	0	0	0
1	0	2	0
2	2	4	1
3	6	6	3
4	14	8	7
5	16	10	8
6	18	12	9
7	20	14	10
8	20	16	10
9	20	18	10
10	20	20	10

La gráfica del mapeo encontrado se muestra en la fig. 3.9.

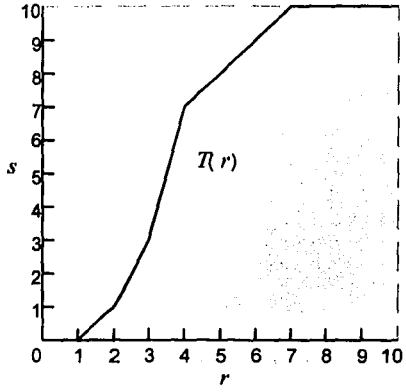
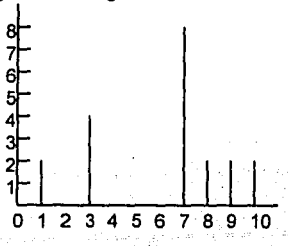


Fig. 3.9 Transformación necesaria para la ecualización del histograma.

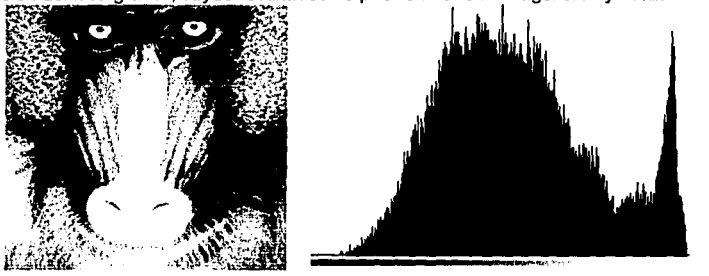
Aplicando el mapeo encontrado a la imagen original obtenemos

n	$f_e(n)$
0	0
1	2
2	0
3	4
4	0
5	0
6	0
7	8
8	2
9	2
10	2

Que da como resultado el siguiente histograma

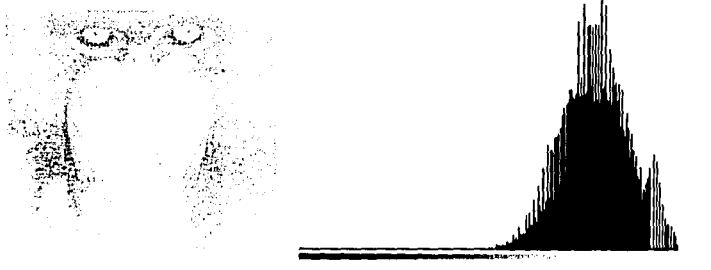


La fig. 3.10 nos muestra una imagen y su histograma, a la cual se le aplica el algoritmo de especificación del histograma, cuyos resultados se pueden ver en las figs. 3.11 y 3.12.

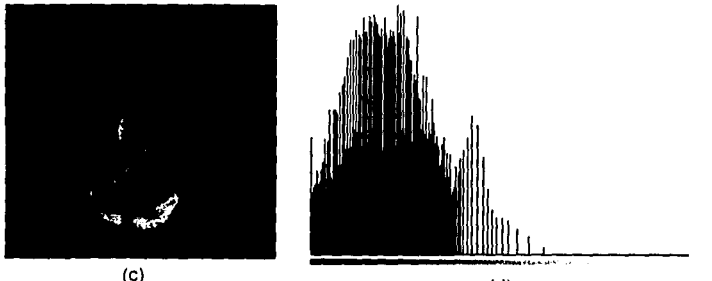


(a) (b)

Fig. 3.10 (a) Imagen original; (b) histograma de la imagen;

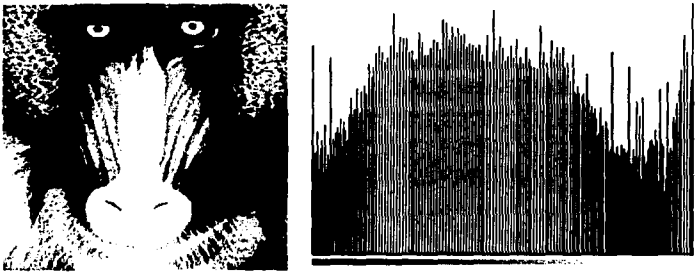


(a) (b)



(c) (d)

Fig. 3.11 (a) imagen después de la especificación del histograma con valores de μ=200 y σ^2=500 en la función de distribución normal, (b) histograma de la imagen; (c) imagen después de la especificación del histograma empleando también la función de distribución normal con valores de μ=50 y σ^2=1000 (d) su histograma.



(a) (b)

Fig. 3.12 (a) imagen después de la ecualización del histograma y (b) su histograma.

La figura 3.10(c) muestra el resultado de la ecualización del histograma. La mejora comparada a la imagen original es muy evidente. La figura 3.10(d) muestra el histograma de la imagen ecualizada. Note, que los niveles de gris de una imagen a la cual se le aplica la ecualización del histograma son extendidos y siempre tendiendo hacia el blanco. Este proceso incrementa el rango dinámico de niveles de gris y, consecuentemente, produce un incremento en el contraste de la imagen. En imágenes con histogramas estrechos y relativamente con pocos niveles de gris la ecualización del histograma mejora significativamente la apariencia visual de la imagen. Resultados similares de realce pueden haber sido conseguidos usando ajuste de contraste discutidos en las primeras secciones del presente capítulo. Claro que la ventaja de la ecualización del histograma sobre las técnicas manuales de manipulación de contraste es que el proceso es completamente automático.

Capítulo 4.

Realce aplicando operaciones espaciales.

Las operaciones espaciales se refieren a procedimientos que operan ya no únicamente sobre pixeles, como los explicados en el capítulo anterior, sino ahora sobre regiones de la imagen.

Las funciones de procesamiento de imágenes para operaciones espaciales pueden ser expresadas por la ecuación 4.1.

$$g(x,y) = T[f(x,y)] \qquad (4.1)$$

donde $f(x,y)$ es la imagen de entrada, $g(x,y)$ es la imagen procesada, y T es un operador de f, definido sobre alguna vecindad de (x, y).

4.1 Conceptos generales.

4.1.1 Vecindades.

La vecindad de un pixel (x, y) es un subimagen de área cuadrada que tiene como centro a (x, y). Por ejemplo los elementos de una vecindad de 3 X 3, son el pixel p colocado en las coordenadas (x, y) el cual tiene cuatro vecinos verticales y horizontales cuyas coordenadas son $(x+1, y)$, $(x-1, y)$, $(x, y+1)$, $(x, y-1)$ además de tener cuatro vecinos diagonales cuyas coordenadas son $(x+1, y+1)$, $(x+1, y-1)$, $(x-1, y+1)$, $(x-1, y-1)$. Este concepto es mostrado en la fig. 4.1.

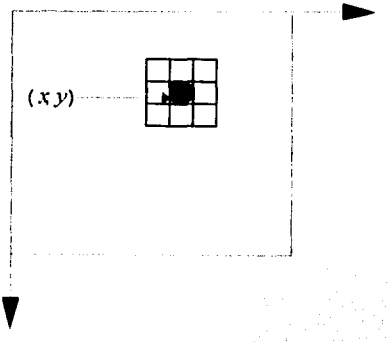


Fig. 4.1. La vecindad de 3 X 3 de un pixel (x, y) en una imagen

El contenido de una imagen puede ser revelado únicamente, cuando analizamos las relaciones espaciales de los niveles de gris. Si el valor de gris en una vecindad pequeña no cambia, nos encontramos en una área de valores de gris constantes; esto puede significar que la vecindad esta incluida en un objeto. Al contrario si el valor de gris cambia, podemos estar en el borde de un objeto. De esta forma, podemos reconocer áreas de valores de gris constantes y bordes. Las operaciones sobre puntos no proporciona este tipo de información. Para mejorar la calidad de las imágenes son necesarias nuevas clases de operaciones, que combinen los pixeles de una vecindad pequeña en forma adecuada, generando como resultado una nueva imagen.

Al realizar operaciones sobre vecindades, el centro de la subimagen es movida a través de todos los pixeles de la imagen, comenzando en la esquina superior izquierda, aplicando un operador en cada pixel (x,y) para obtener un nuevo valor g en ese pixel. Aunque existen otras formas de vecindades, como aproximaciones a círculos, los arreglos cuadrados son los más usados debido a su facilidad de implementación.

La forma más simple de T es cuando la vecindad es de 1 X 1. En este caso, g depende solamente de los valores de f y (x, y), convirtiendo a T en una transformación sobre puntos.

4.1.2 Máscaras o filtros espaciales.

El uso de vecindades permite una gran variedad de aplicaciones que van más alla del realce de imágenes. No obstante la aplicación específica, la propuesta general es permitir que los valores de f es una vecindad predefinida de (x, y) determinen los valores de g en (x, y). Uno de los enfoques principales dentro de esta formulación está basada en el uso de las llamadas máscaras, referidas también como templetes, ventanas o filtros. Básicamente, una máscara es un pequeño arreglo bidimensional, por ejemplo de 3 X 3, en el cual los valores de los coeficientes determinan la naturaleza del proceso, como por ejemplo el aclarado de imágenes.

El uso de máscaras espaciales en el procesamiento de imágenes se le conoce como filtrado espacial y las máscaras a su vez son llamadas filtros espaciales. Existen dos tipos principales de filtros espaciales, los lineales y los no lineales.

4.1.2.1 Filtros espaciales lineales.

Los filtros lineales están basados en el concepto que establece que la respuesta al impulso es igual a la transformada inversa de Fourier de la función de transferencia del sistema. Los filtros paso bajas atenúan o eliminan los componentes de alta frecuencia en el dominio de Fourier mientras las bajas frecuencias no son tocadas, esto es que el filtro deja pasar exclusivamente las frecuencias bajas. Los componentes de alta frecuencia caracterizan los bordes y otros detalles de la forma en una imagen, por lo tanto el efecto del filtrado paso bajas es crear imágenes borrosas. De manera similar, los filtros paso altas atenúan o eliminan los componentes de bajas frecuencias. Debido a que estos componentes son responsables de pequeñas variaciones características de una imagen, como contraste total e intensidad promedio, el resultado del filtrado paso altas es una reducción de estas características y una aparente aclarado de los bordes y otros detalles de la forma. Un tercer tipo de filtrado, llamado filtrado paso banda, remueve regiones de frecuencia seleccionadas entre bajas y altas frecuencias. Estos filtros son usados para restauración de imágenes y no son de interés en el realce de imágenes.

La fig. 4.2 muestra una sección transversal de los filtros paso bajas, paso altas y pasa banda en el dominio de la frecuencia y la forma que deben tener los elementos para construir los filtros espaciales correspondientes. El eje horizontal para las figuras del renglón superior corresponden a las frecuencias, y sus contrapartes en el renglón inferior son las coordenadas espaciales. La forma en el renglón inferior son usados como guía para especificar el filtro lineal espacial.

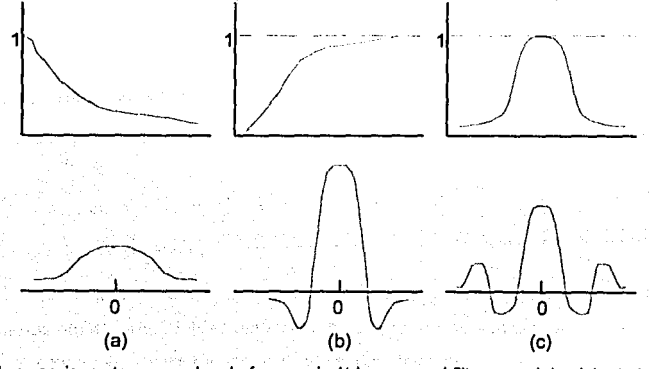


Fig. 4.2 arriba: secciones transversales de formas simétricas para el filtros en el dominio de la frecuencia circularmente simétricos. Abajo: secciones transversales de filtros correspondiente al dominio espacial.

No importando el tipo de filtro lineal usado, el enfoque básico es sumar los productos de los coeficientes de la máscara por la intensidad de los pixeles bajo la máscara en una localidad especifica de la imagen. En otras palabras multiplicamos cada pixel de la vecindad en el rango de la máscara con el factor correspondiente de la máscara, sumamos los productos y escribimos el resultado en la posición del pixel central. La fig. 4.3 muestra un máscara de 3 X 3. Denotando los pixeles en la vecindad bajo la máscara en cualquier locación por $z_1, z_2, z_3, \ldots, z_9$, la respuesta de la máscara lineal es calculada con la ecuación 4.2.

$$R = w_1 z_1 + w_2 z_2 + w_3 z_3 + \cdots + w_9 z_9 \qquad (4.2)$$

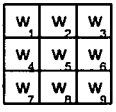


Fig. 4.3 Una máscara de 3 X 3 con coeficientes arbitrarios.

Con referencia a la figura 4.1, si el centro de la máscara esta en la locación (x,y) en la imagen, el nivel de gris del pixel localizado en (x,y) es reemplazado por R. La máscara entonces es movida al siguiente pixel de la imagen y se repite el proceso. Así continua hasta que todos los pixeles de la imagen hayan sido recorridos. En la práctica se crea una nueva imagen que guarde los valores de R, en lugar de cambiar el valor de los pixeles en su lugar. Esta práctica evita usar valores de gris, en la ecuación 4.2, que hayan sido alterados como resultado de una aplicación temprana de esta ecuación.

Generalizando la ecuación 4.2 para todos los puntos contenidos en la imagen obtenemos la ecuación 4.3.

$$g_{xy} = \sum_{k=-r}^{r} \sum_{l=-r}^{r} w_{kl} f_{x-k,y-l} = \sum_{k=-r}^{r} \sum_{l=-r}^{r} w_{-k,-l} f_{x+k,n+l} \qquad (4.3)$$

La ecuación 4.3 crea una máscara de tamaño impar con los coeficientes $(2r+1) \times (2r+1)$. Se puede observar que describe una operación de convolución. En comparación con la convolución continua, la integral es reemplazada por una suma discreta sobre los elementos. La fig. 4.4 nos muestra gráficamente un ejemplo de esta operación.

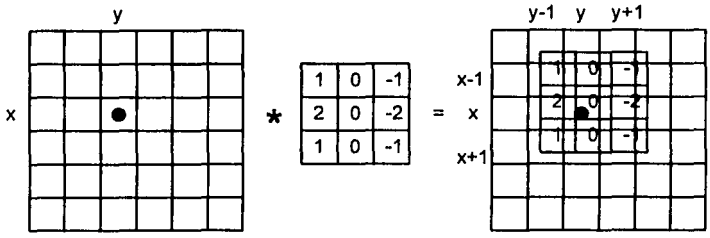


Fig. 4.4 Ilustración de la operación de convolución discreta con una máscara de 3 X 3.

4.1.2.2 Filtros espaciales no lineales.

Los filtros espaciales no lineales operan también en vecindades. Sin embargo, las operaciones están basadas directamente en los valores de los pixeles en la vecindad bajo consideración y no usan explícitamente coeficientes de la forma descrita en la ecuación 4.2. Por ejemplo, la reducción de ruido se puede conseguir efectivamente con un filtro no lineal cuya función básica es obtener el valor de gris medio en la vecindad donde el filtro esta localizado. Otro ejemplo incluye el filtro de máxima (con una respuesta $R = \max\{z_k | k = 1,2,3,...,9\}$), el cual es usado para buscar los puntos más brillantes en una imagen, y el filtro de mínima, el cual es usado para el propósito opuesto.

4.2 Filtros para suavizado de imágenes.

Los filtros para suavizado son usados para reducir el ruido en las imágenes. El suavizado es usado en fases de preprocesamiento, tales como remover pequeños detalles de la imagen antes de la extracción de objetos. Se puede conseguir la reducción de ruido por medio de suavizado de la imagen con un filtro lineal y también con un filtro no lineal.

4.2.1 Filtrado espacial paso bajas.

La forma de la respuesta al impulso necesitada para implementar un filtro espacial paso bajas indica que el filtro tiene que tener todos los coeficientes positivos (ver la fig. 4.2a). Para un filtro espacial de 3 X 3, el arreglo más simple consiste en una máscara en la cual todos los coeficientes tengan el valor de 1. Sin embargo, de la ecuación 4.2, la respuesta sería la suma de las multiplicaciones de los pixeles por nueve coeficientes, lo cual causaría que R sobrepase el rango de niveles de gris permitidos. La solución es escalar la suma dividiendo R entre 9. La fig. 4.5(a) muestra la máscara resultante. Máscaras más grandes siguen el mismo concepto, como lo muestran las figuras 4.5(b) y (c). Note que en todos los casos, la respuesta R es simplemente el

promedio de todos los pixeles en el área de la máscara. Por esta razón, el uso de máscara de la forma mostrada en la fig. 4.5 es referida como el promedio de la vecindad.

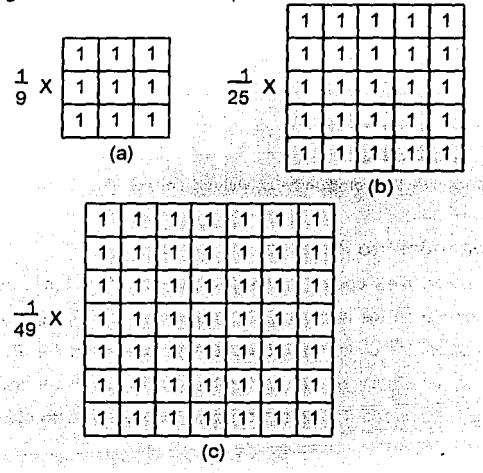


Fig. 4.5 Filtros espaciales paso bajas de varios tamaños.

La fig. 4.6 nos muestra un ejemplo de suavizado de una imagen por medio de la aplicación de máscaras sucesivamente más grandes. Note en particular la pérdida en la definición de bordes.

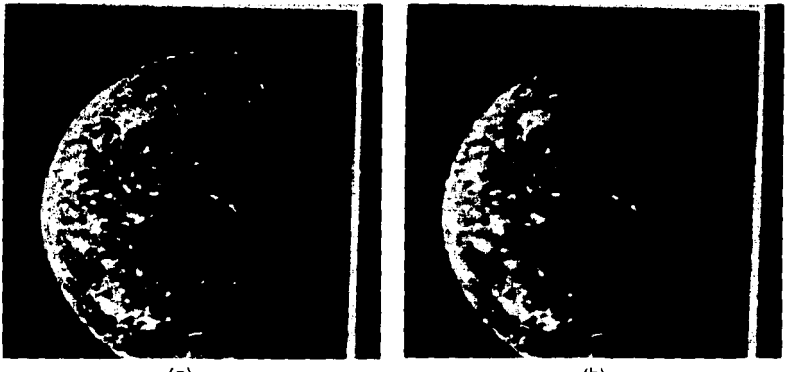


Fig. 4.6 (a) Imagen original; (b) resultado de filtrado espacial paso bajas con máscaras de tamaño nXn, donde n = 3.

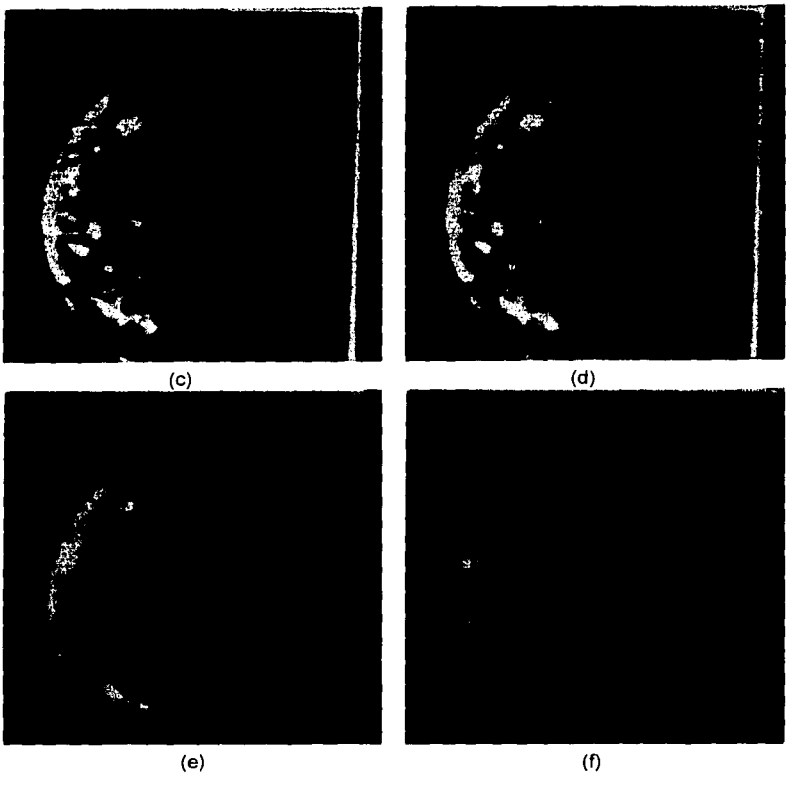


Fig. 4.6 (continuación)(c) - (f) resultado de filtrado espacial paso bajas con máscaras de tamaño n X n, donde n = 5, 7 ,15, 25.

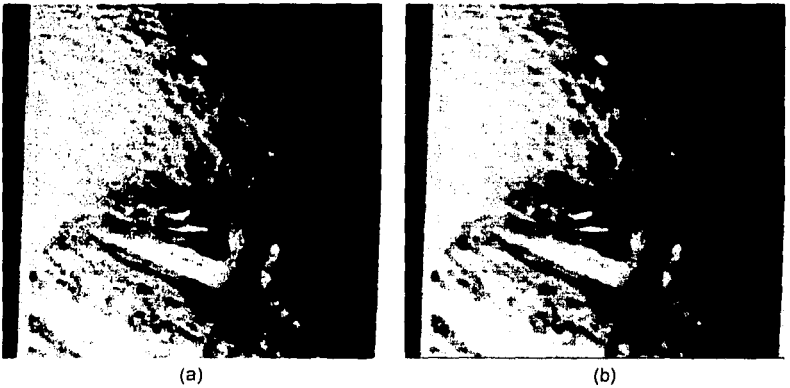
4.2.2 Filtrado de mediana.

Una de las principales dificultades del método de suavizado discutido en la sección precedente es que hace borrosos los bordes y otros detalles en la forma. Si nuestro objetivo es reducir ruido sin crear imágenes borrosas, una alternativa es el uso de filtros de mediana. Esto es, los grados de nivel de cada pixel es reemplazado por el valor medio de los niveles de gris en una vecindad del pixel, en lugar del promedio. Este método es particularmente efectivo cuando se quiere reducir el ruido, preservando la forma de los bordes. Como fue indicado anteriormente, los filtros de mediana son no lineales.

La mediana de m entre un conjunto de valores es aquel al cual la mitad del conjunto de valores es menor que m y la mitad son más grandes que m. Para llevar a cabo el filtrado de mediana en la vecindad de un pixel, primero ordenaremos los valores del pixel y sus vecinos, determinamos la mediana, y asignamos este valor al pixel. Por ejemplo, en una vecindades 3 X 3 la mediana es el 5to. valor más grande, en una vecindad de 5 X 5 el 13vo. valor más grande y así sucesivamente. Cuando varios valores en una vecindad son iguales, todos esos valores serán agrupados. Por ejemplo, suponga que una vecindad de 3 X 3 tiene los valores (10,20,20,20,15,20,20,25,100). Estos valores son ordenados como (10,15,20,20,20,20,20,25,100), de lo cual nos da como resultado una mediana de 20.

La función principal del filtrado de mediana es forzar los puntos con diferentes intensidades para que se parezcan a sus vecinos, realmente elimina los picos aislados de intensidad que aparecen en el área de la máscara.

El desempeño del filtro de mediana es deficiente cuando el número de pixeles que contienen ruido en la vecindad son es más grande que la mitad del número de pixeles en la vecindad.



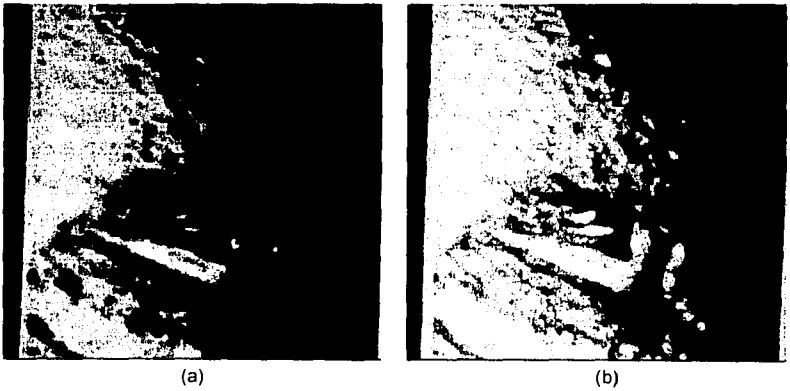
(a) (b)

Fig. 4.7 (a) Imagen original; (b) imagen suavizada con filtrado paso bajas

(c)

Fig. 4.7 (continuación) (c) imagen suavizada con filtrado de mediana

Podemos calcular otro tipo de filtrado, en base a la ordenación de los elementos en una vecindad, como tomar los valores mínimos o máximos en la vecindad.



(a) (b)

Fig. 4.8 (a) Aplicación del filtro de mínima. (b) Aplicación del filtro de máxima.

4.3 Filtros para agudizamiento de los bordes de imágenes.

4.3.1 Filtrado espacial paso altas básico.

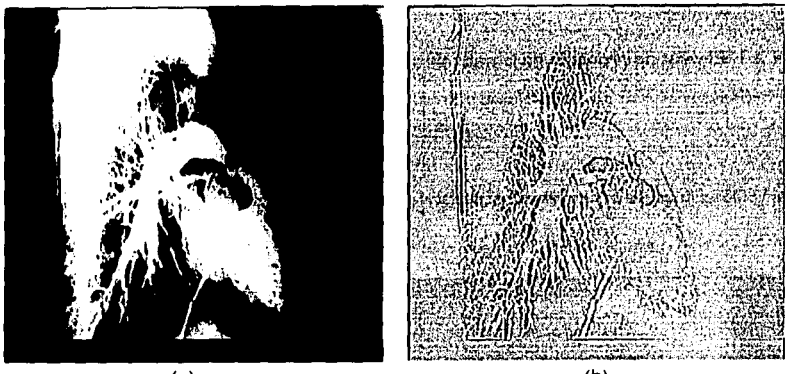
La forma de la respuesta al impulso requerida para implementar un filtro espacial paso altas, indica que el filtro debe tener coeficientes positivos cerca del centro y coeficientes negativos en la periferia (ver la fig. 4.2b). Para una máscara de 3 X 3, se puede cumplir esta condición seleccionando un valor positivo en la localidad central y coeficientes negativos en el resto de los coeficientes.

La fig. 4.9 muestra la implementación clásica de un filtro paso altas de 3 X 3. Note que la suma de los coeficientes es 0. De este modo cuando la máscara esta sobre una área de niveles de gris con valores constantes o poco variables, la salida de la máscara es cero o muy pequeña. Este resultado es consistente con lo esperado del filtro en el dominio de la frecuencia mostrado en la fig. 4.2(b). Note también que el filtro elimina los términos con frecuencia cero. Al eliminar estos términos se reduce el valor de nivel de gris promedio en la imagen a cero, reduciendo significativamente el contraste global de la imagen.

$$\frac{1}{9} \times \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Fig. 4.9 Un filtro espacial paso bajas básico.

Reducir el valor promedio de una imagen a cero implica que la imagen debe tener pocos niveles de gris. Como nosotros manejamos solamente valores positivos, los resultados del filtrado paso altas involucra alguna forma de ajuste de las escala de grises para que los niveles de gris del resultado final estén incluidos en el rango $[0, L\text{-}1]$. Tomar los valores absolutos de la imagen filtrada para hacer todos los valores positivos no es una buena idea porque los valores negativos grandes podrían aparecer muy brillantes en la imagen.



(a) (b)

Fig. 4.10 Imagen original (b) Imagen resultado del filtrado paso altas usando la máscara de la fig. 4.9.

4.3.2 Filtrado High-boost.

Una imagen resultado de un filtro paso altas puede ser calculado como la diferencia entre la imagen original y una versión filtrada con paso bajas; como se muestra en la ecuación 4.4.

$$\text{Filtrado paso altas} = \text{Original} - \text{Filtrado paso bajas} \qquad (4.4)$$

Como un ejercicio, es instructivo verificar la validez de la ecuación 4.4 usando la ecuación 4.2 en conjunción con las figs. 4.5 y 4.9. Multiplicando la imagen original por un factor amplificador, denotado por A, obtenemos la definición de un filtro high-boost o de énfasis a las altas frecuencias en el desarrollo de la ecuación 4.5.

$$\begin{aligned} \text{High-boost} &= (A)(\text{Original}) - \text{Paso bajas} \\ &= (A-1)(\text{Original}) + \text{Original} - \text{Paso bajas} \\ &= (A-1)(\text{Original}) + \text{Paso altas}. \end{aligned} \qquad (4.5)$$

Un valor de A=1 nos da el resultado del paso altas estándar. Cuando A>1, parte del original es adicionado de nuevo al resultado del paso altas, el cual parcialmente restaura los componentes de bajas frecuencias perdidos en la operación de filtrado. El resultado es que la imagen high-boost luce muy similar a la original, con un grado relativo de realce de bordes que depende del valor de A. El proceso general de substraer una imagen suavizada del original, esta dado en la primera línea de la ecuación 4.5 y es llamado enmascaramiento no agudo. Este método es una las herramientas básicas para aplicaciones de procesamiento de imágenes en la industria de la publicidad y la impresión.

En términos de implementación, el resultado precedente puede combinarse permitiendo que el coeficiente central de la máscara mostrada en la fig. 4.11 tome el valor de la ecuación 4.6.

$$w = 9A - 1 \qquad (4.6)$$

con $A \geq 1$. El valor de A determina la naturaleza del filtro.

$$\frac{1}{9} \times \begin{bmatrix} -1 & -1 & -1 \\ -1 & w & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Fig. 4.11 Máscara usada para el filtrado espacial High-boost. El valor del coeficiente central es $w=9A-1$ con $A \geq 1$

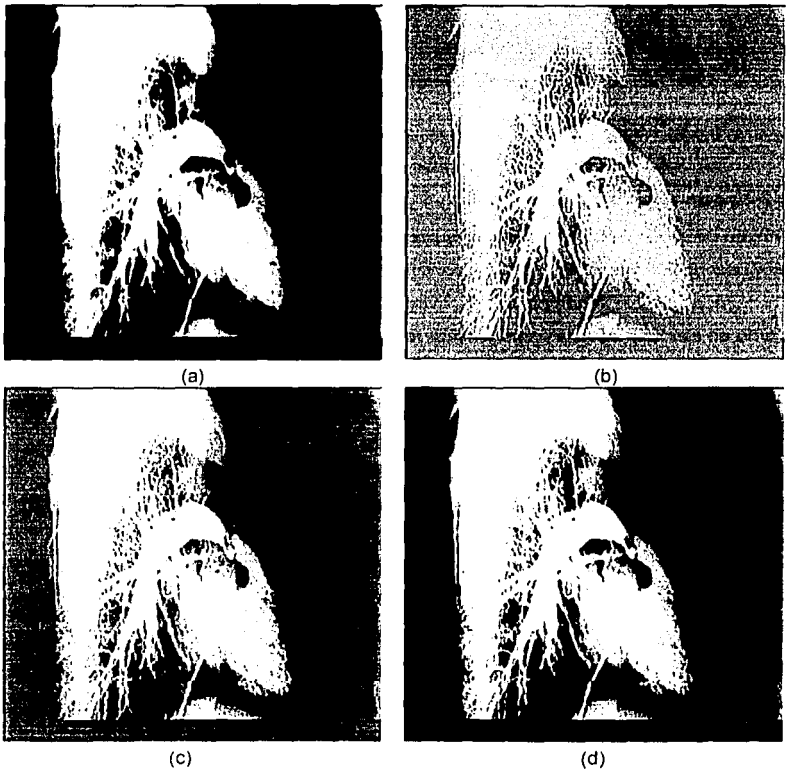


Fig. 4.12 (a) Imagen original; (b) resultado del filtrado High-boost, usando el valor de $w=1.2$.
(c) - (d) resultado del filtrado High-boost, usando valores de (c) $w=1.5$; y (d) $w=2.0$

Como en el caso de el filtrado paso bajas, es posible especificar filtros espaciales paso altas de tamaños más grandes que los anteriormente discutidos. Por instancia, un filtro paso altas básico de 7 X 7 tendría un coeficiente central de 48, el resto de los coeficientes serían -1, y el factor de normalización sería 1/49. Sin embargo, en la práctica los filtros paso altas más grandes que 3 X 3 casi nunca son necesitados.

4.4 Filtros para la detección de bordes.

4.4.1 Filtros de diferencia.

Promediar los pixeles sobre una región tiende a hacer borrosos los detalles en una imagen. Como promediar es análogo a la integración, se puede esperar que la diferenciación tenga el efecto opuesto y agudice una imagen.

El método más común de diferenciación en una aplicación de procesamiento de imágenes es el gradiente. Para una función $f(x,y)$, el gradiente de f en las coordenadas (x,y) esta definido como el vector de la expresión 4.7.

$$\nabla \mathrm{f} = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \dfrac{\partial f}{\partial y} \end{bmatrix} \qquad (4.7)$$

De análisis vectorial es conocido que el vector gradiente apunta en la dirección del nivel máximo de cambio de f en (x,y). En la detección de bordes una cantidad importante es la magnitud de este vector, que se calcula con la expresión 4.8.

$$\nabla f = \mathrm{mag}(\nabla \mathrm{f}) = \left[\left(\frac{\partial f}{\partial x} \right)^2 + \left(\frac{\partial f}{\partial y} \right)^2 \right]^{1/2} \qquad (4.8)$$

la ecuación 4.8 es la base para varias aproximaciones a la diferenciación de imágenes. Note de las ecuaciones 4.7 y 4.8 que el cálculo del gradiente de una imagen esta basado en la obtención de las derivadas parciales $\partial f/\partial x$ y $\partial f/\partial y$ en cada pixel de la imagen. Considere la región de una imagen mostrada en la fig. 4.13(a), donde las letras z denotan los valores de los niveles de gris. La ecuación 4.8 puede ser aproximada en el punto z_5 de varias formas. La más simple es usar la diferencia $(z_5 - z_8)$ en la dirección x y $(z_5 - z_6)$ en la dirección y, combinadas como se muestra en la expresión 4.9a.

$$\nabla f \approx \left[(z_5 - z_8)^2 + (z_5 - z_6)^2 \right]^{1/2} \qquad (4.9a)$$

En lugar de usar cuadrados y raíces cuadradas, podemos obtener resultados similares usando valores absolutos, como en la expresión 4.9b.

$$\nabla f \approx |z_5 - z_8| + |z_5 - z_6| \qquad (4.9b)$$

Esta aproximación es muy importante debido a que el cálculo de dos diferencias elevadas al cuadrado y una raíz cuadrada hace muy lenta la obtención de la imagen realzada. Por esa razón se elige la aproximación por medio de la suma de los valores absolutos de las diferencias.
Se puede verificar el error de nuestra aproximación con la expresión 4.9c.

$$\frac{(|z_5 - z_8| + |z_5 - z_6|) - [(z_5 - z_8)^2 + (z_5 - z_6)^2]^{1/2}}{[(z_5 - z_8)^2 + (z_5 - z_6)^2]^{1/2}} \qquad (4.9c)$$

Calculando los errores porcentuales (es decir el resultado de la expresión 4.9c multiplicado por 100) para valores de z en el intervalo [0,255], que es precisamente el intervalo que comprende la escala de grises, se puede comprobar que los valores de error son variables pero no rebasan el 41.2% de error.

Si se toman los pixeles indicados en la expresión 4.9a para calcular el gradiente en una vecindad de 2 X 2 se corre el riesgo de no detectar un borde, debido a que puede existir una significativa diferencia con respecto al pixel z_9, el cual forma parte de la vecindad. Para evitar esta situación podemos obtener mejores resultado al obtener los bordes usando las diferencias cruzadas, como se aprecia en la expresión 4.10a.

$$\nabla f \approx [(z_5 - z_9)^2 + (z_6 - z_8)^2]^{1/2} \qquad (4.10a)$$

o, usando valores absolutos, tal como en la expresión 4.10b.

$$\nabla f \approx |z_5 - z_9| + |z_6 - z_8| \qquad (4.10b)$$

La ecuación 4.10 puede ser implementada tomando el valor absoluto de la respuesta de las dos máscaras mostradas en la fig. 4.13(b) y sumar los resultados. Estas máscaras son llamados los operadores gradiente cruzados de Roberts.

Es difícil implementar máscaras de tamaño más grandes. Una aproximación a la ecuación 4.8, en el punto z_5, pero usando ahora una vecindad de 3 X 3, es calculada con la expresión 4.11.

$$\nabla f \approx |(z_7 + z_8 + z_9) - (z_1 + z_2 + z_3)| + |(z_3 + z_6 + z_9) - (z_1 + z_4 + z_7)| \qquad (4.11)$$

La diferencia entre el tercer y el primer renglón de la región de 3 X 3 aproxima la derivada en la dirección x, y la diferencia entre el tercer y la primera columna aproxima el derivativo en la dirección y. La máscaras mostradas en la fig. 4.13(c), llamada operadores Prewitt, pueden usarse para implementar la ecuación 4.11. Finalmente, la fig. 4.13(d) muestran otra pareja de máscaras, llamadas operadores de Sobel para aproximar la magnitud del gradiente.

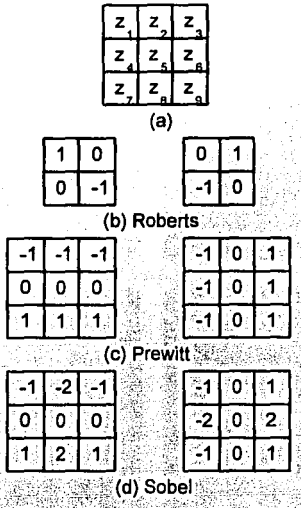


Fig. 4.13 Una región de 3 X 3 de una imagen (las letras z son los valores de nivel de gris) y varias máscaras usadas para calcular el derivativo en el punto con la etiqueta z_5. Note que todos los coeficientes de las máscaras suman 0, indicando una respuesta de 0 en áreas constantes, como es esperado de un operador derivativo.

El operador de Sobel tiene la ventaja de diferenciar y dar un efecto de suavizado, lo cual es atractivo porque los derivativos realzan el ruido. Para calcular el gradiente con los operadores mostrados en la Fig. 4.13 debemos convolucionar por separado ambas máscaras en cada vecindad de la imagen, para después obtener los valores absolutos de las dos convoluciones y finalmente sumarlos, lo que nos da el gradiente del pixel.

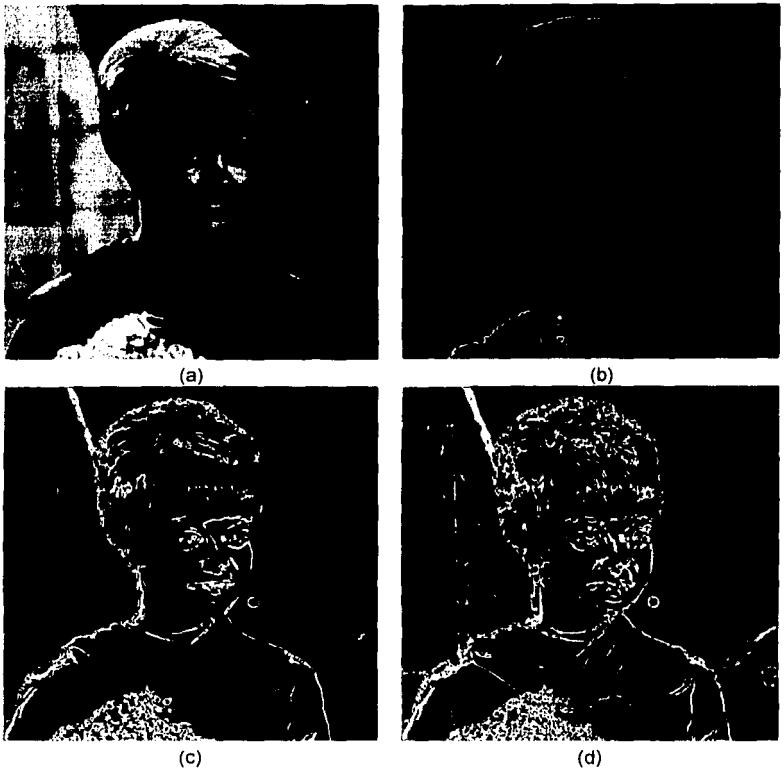


Fig. 4.14 Realce de bordes con técnicas del gradiente. (a) Imagen original; (b) imagen filtrada con el operador de Roberts; (c) imagen filtrada con el operador de Prewitt; (d) imagen filtrada con el operador de Sobel.

4.1.2 Diferenciación estadistica.

Otra forma de obtener el borde de una imagen es por medio de la diferenciación estadística, que consiste en la generación de una imagen por medio de la división de cada valor de pixel por su desviación estándar

$$g(x,y) = \frac{f(x,y)}{\sigma(x,y)} \qquad (4.12)$$

donde definida por la ecuación

$$\sigma(x,y)=\left\{\frac{1}{N_v}\sum_{(k,l)\in v}\sum\left[f(x-k,y-l)-\mu(x,y)\right]^2\right\}^{1/2} \qquad (4.13)$$

donde $\mu(x,y)$ es la media definida por la ecuación

$$\mu(x,y)=\frac{1}{N_v}\sum_{(k,l)\in v}\sum f(x-k,n-l) \qquad (4.14)$$

La imagen realzada $g(x,y)$ es incrementada en amplitud con respecto a los pixeles originales que se desvian significativamente de sus vecinos y es decrementada en amplitud con respecto a las demás partes. Como resultado se obtiene una imagen donde los bordes claros (que tienen bajo contraste) son realzados. Esta transformación es también llamada ajuste estadístico.

(a) (b)

Fig. 4.15 (a) Imagen original; (b) imagen resultado de la diferenciación estadística.

4.6 Ampliación e interpolación.

En ocasiones se desea ampliar una región dada de una imagen. Para ello se requiere tomar un segmento de la imagen y mostrarla como una imagen más grande.

4.6.1 Ampliación por réplica de pixeles.

Para ampliar una imagen haciendo una réplica de los pixeles de la imagen original, cada pixel a lo largo de una linea rastreada es repetido una vez y al finalizar cada linea esta es repetida. Esto es equivalente a tomar una imagen de M X N e intercalarla con renglones y columnas de ceros para obtener una matriz de $2M$ X $2N$ y convolucionar el resultado con la matriz H, definida en la expresión 4.16.

$$H=\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix} \qquad (4.16)$$

El procedimiento anteriormente descrito se representa gráficamente con un ejemplo en la fig. 4.16

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 5 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 3 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 5 & 6 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} * H = \begin{bmatrix} 1 & 1 & 3 & 3 & 2 & 2 \\ 1 & 1 & 3 & 3 & 2 & 2 \\ 4 & 4 & 5 & 5 & 6 & 6 \\ 4 & 4 & 5 & 5 & 6 & 6 \end{bmatrix}$$

Fig. 4.16 Procedimiento para ampliar una imagen por réplica de pixeles usando la matriz de la expresión 4.16.

(a) (b)

Fig. 4.17 Ampliación por réplica de pixeles. (a) Imagen original; (b) imagen ampliada.

4.6.2 Ampliación por interpolación lineal.

Para ampliar una imagen interpolando los pixeles tomamos una linea recta donde se ajustan los pixeles a lo largo de un renglón. De esta forma los pixeles a lo largo de cada columna son interpolados a lo largo de una línea recta. La representación matemática de este procedimiento es mostrado en la expresión 4.17.

$$\left.\begin{aligned} g_1(x,2y) &= f(x,y), \\ & 0 \le x \le M-1, 0 \le y \le N-1 \\ g_1(x,2y+1) &= \frac{1}{2}\left[f(x,y)+f(x,y+1)\right], \\ & 0 \le x \le M-1, 0 \le y \le N-1 \end{aligned}\right\} \quad (4.17)$$

La interpolación a través de las columnas nos da el resultado mostrado en la expresión 4.18.

$$\left.\begin{aligned} g(2x,y) &= g_1(x,y), \\ g(2x+1,y) &= \frac{1}{2}\left[g_1(x,y)+g_1(x+1,y)\right], \\ & 0 \le x \le M-1, 0 \le y \le 2N-1 \end{aligned}\right\} \quad (4.18)$$

Debemos asumir en la imagen original, que los pixeles que se encuentren fuera de [0,M-1] X [0,N-1] sean cero. El resultado anterior puede obtenerse convolucionando una imagen con dimensiones $2M$ X $2N$, intercalada con columnas y renglones de ceros, con la matriz mostrada en la expresión 4.19.

$$H = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix} \quad (4.19)$$

El procedimiento se muestra gráficamente en la fig. 4.18, donde interpolamos primero los renglones y después las columnas.

$$\begin{bmatrix} 1 & 7 \\ 3 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 4 & 7 & 3.5 \\ 0 & 0 & 0 & 0 \\ 3 & 2 & 1 & 0.5 \\ 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 4 & 7 & 3.5 \\ 2 & 3 & 4 & 2 \\ 3 & 2 & 1 & 0.5 \\ 1.5 & 1 & 0.5 & 0.25 \end{bmatrix}$$

Fig. 4.18 Procedimiento de ampliación por interpolación. Note que primero interpolamos por renglones y posteriormente por columnas.

(a) (b)

Fig. 4.19 Ampliación por interpolación lineal. (a) Imagen original; (b) Imagen ampliada e interpolada.

En la mayoría de los casos la interpolación lineal cumple muy satisfactoriamente su objetivo.

Capítulo 5.

Realce aplicando operaciones en el dominio de las frecuencias.

5.1 La transformada de Fourier.

5.1.1 Definición de la transformada de Fourier.

Sea $f(x)$ una función continua de una variable real x. La transformada de Fourier de $f(x)$, denotada por $\Im\{f(x)\}$, esta definida por la ecuación 5.1.

$$\Im\{f(x)\} = F(u) = \int_{-\infty}^{\infty} f(x) e^{-j2\pi ux} dx \qquad (5.1)$$

donde $j = \sqrt{-1}$.

Dada $F(u)$, podemos obtener $f(x)$ usando la transformada inversa de Fourier definida en la ecuación 5.2.

$$\Im^{-1}\{F(u)\} = f(x) = \int_{-\infty}^{\infty} F(u) e^{j2\pi ux} du \qquad (5.2)$$

Las ecuaciones 5.1 y 5.2, llamadas la pareja de transformadas de Fourier, existen si $f(x)$ es continua e integrable y $F(u)$ es integrable. Estas condiciones son casi siempre satisfechas en la práctica.

Nuestro interés esta concentrado en las funciones $f(x)$ que son reales. Sin embargo la transformada de Fourier de una función real, es generalmente compleja, lo cual es expresado en la ecuación 5.3.

$$F(u) = R(u) + jI(u) \qquad (5.3)$$

donde $R(u)$ e I(u) son los componentes real e imaginario de $F(u)$, respectivamente. Es conveniente expresar la ecuación 5.3 en forma exponencial que es la mostrada en la ecuación 5.4.

$$F(u) = |F(u)| e^{j\phi(u)} \qquad (5.4)$$

de donde definimos las ecuaciones 5.5 y 5.6

$$|F(u)| = [R^2(u) + I^2(u)]^{1/2} \qquad (5.5)$$

$$\phi(u) = \tan^{-1}\left[\frac{I(u)}{R(u)}\right] \tag{5.6}$$

La función de la magnitud $|F(u)|$ es llamado el espectro de Fourier de $f(x)$ y $\phi(u)$ su ángulo de fase. La raíz del espectro, esta definida en la ecuación 5.7.

$$P(u) = |F(u)|^2 = R^2(u) + I^2(u) \tag{5.7}$$

la cual es comúnmente referida como el espectro de potencias de $f(x)$. El término densidad espectral era también comúnmente usado para identificar el espectro de potencias.

La variable u que aparece en la transformada de Fourier es llamada la variable de frecuencia. Este nombre surge de la expresión del término exponencial $e^{-j2\pi ux}$, usando la formula de Euler obtenemos la ecuación 5.8.

$$e^{-j2\pi ux} = \cos 2\pi ux - j\sin 2\pi ux \tag{5.8}$$

Interpretando la integral de la ecuación 5.1 como un limite de términos discretos se hace evidente que $F(u)$ esta compuesto de una suma infinita de términos senoidales y cosenoidales y que cada valor de u determina la frecuencia de su pareja de senos y cosenos.

La transformada de Fourier puede ser fácilmente extendida a una función $f(x,y)$ de dos variables. Si $f(x,y)$ es continua e integrable y $F(u,v)$ es integrable, la pareja de transformadas de Fourier mostradas en las ecuaciones 5.9 y 5.10 existen.

$$\Im\{f(x,y)\} = F(u,v) = \int_{-\infty}^{\infty}\int f(x,y)e^{-j2\pi(ux+vy)}dxdy \tag{5.9}$$

$$\Im^{-1}\{F(x,y)\} = f(x,y) = \int_{-\infty}^{\infty}\int F(u,v)e^{j2\pi(ux+vy)}dudv \tag{5.10}$$

donde u y v son las variables de frecuencia.

Como en el caso unidimensional (1-D), el espectro de Fourier, la fase y el espectro de potencias están definidas respectivamente por las ecuaciones 5.11, 5.12 y 5.13.

$$|F(u,v)| = \left[R^2(u,v) + I^2(u,v)\right]^{1/2} \tag{5.11}$$

$$\phi(u,v) = \tan^{-1}\left[\frac{I(u,v)}{R(u,v)}\right] \tag{5.12}$$

$$P(u,v) = |F(u,v)|^2 = R^2(u,v) + I^2(u,v) \tag{5.13}$$

5.1.2 La transformada discreta de Fourier.

Suponga que una función continua $f(x)$ es discretizada en una secuencia como la mostrada en la expresión 5.14.

$$\{f(x_0), f(x_0+\Delta x), f(x_0+2\Delta x), \ldots, f(x_0+[N-1]\Delta x)\} \qquad (5.14)$$

tomando N muestras apartadas una unidad Δx, como es mostrado en la fig. 5.1. Para una variable discreta requerimos definir la ecuación 5.15.

$$f(x) = f(x_0 + x\Delta x) \qquad (5.15)$$

donde x asume ahora los valores discretos 0, 1, 2,...,N-1. En otras palabras, la secuencia $\{f(0), f(1), f(2), \ldots, f(N-1)\}$ denota cualquier N muestreado uniformemente espaciado de la función continua correspondiente.

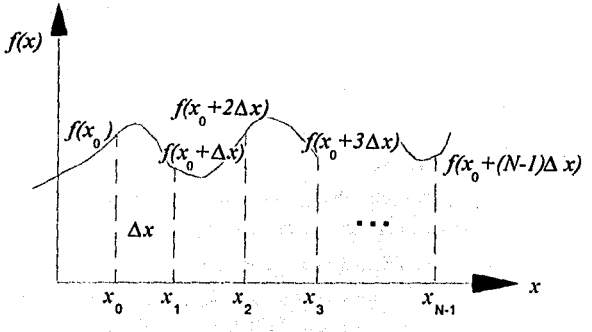


Fig. 5.1 Muestreo de una función continua.

Con la notación anterior en mente, la pareja de transformadas discretas de Fourier que son aplicadas a las funciones muestreadas están dadas por las ecuaciones 5.16 y 5.17.

$$F(u) = \frac{1}{N}\sum_{x=0}^{N-1} f(x) e^{-j2\pi ux/N} \qquad (5.16)$$

para u = 0, 1, 2,...,N-1,

$$f(x) = \sum_{u=0}^{N-1} F(u) e^{j2\pi ux/N} \qquad (5.17)$$

para x = 0, 1, 2,...,N-1.

Los valores u = 0, 1, 2,...,N-1 en la transformada discreta de Fourier (ecuación 5.16) corresponde al muestreo de la transformada continua en los valores 0, Δu, 2Δu,...,(N-1)Δu, por lo tanto $F(u)$ representa a $F(u\Delta u)$. Esta notación es similar a la usada para la función discreta $f(x)$, excepto que las muestras de $F(u)$ comienzan en el origen del eje de las frecuencias. Los términos Δu y Δx están relacionados por la expresión 5.18.

$$\Delta u = \frac{1}{N\Delta x} \tag{5.18}$$

En el caso de dos variables la pareja de transformadas de Fourier son las mostradas en las ecuaciones 5.19 y 5.20.

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(ux/M + vy/N)} \tag{5.19}$$

para u = 0, 1, 2,...,M-1, v = 0, 1, 2, ..., N-1,

$$f(x,y) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{j2\pi(ux/M + vy/N)} \tag{5.20}$$

para x = 0, 1, 2,...,M-1, v = 0, 1, 2,..., N-1.

El muestreo de una función continua esta ahora en un enrejado bidimensional (2-D), con divisiones de tamaño Δx y Δy en los ejes x y y, respectivamente. Como en el caso unidimensional, la función discreta $f(x,y)$ representa muestras de la función $f(x_0 + x\Delta x, y_0 + y\Delta y)$ para x = 0, 1, 2,...,M-1 y y = 0, 1, 2,...,N-1. Comentarios similares se aplican a $F(u,v)$. Los incrementos en el muestreo en los dominios espacial y de frecuencia están expresado por las ecuaciones 5.21 y 5.22.

$$\Delta u = \frac{1}{M\Delta x} \tag{5.21}$$

$$\Delta v = \frac{1}{N\Delta y} \tag{5.22}$$

Cuando las imágenes son muestreadas en un arreglo cuadrado, $M = N$ y podemos expresar las ecuaciones 5.23 y 5.24.

$$F(u,v) = \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(ux + vy)/N} \tag{5.23}$$

para u, v = 0, 1, 2,...,N-1,

$$f(x,y)=\frac{1}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}F(u,v)e^{j2\pi(ux+vy)/N} \qquad (5.24)$$

para $x, y = 0, 1, 2,...,N-1$. Note la inclusión de un término $1/N$ en las ecuaciones 5.23 y 5.24. Debido a que $F(u,v)$ y $f(x,y)$ son una pareja de transformadas de Fourier, la agrupación de estos términos constantes multiplicativos es arbitrara. En la práctica, las imágenes generalmente son digitalizadas en arreglos cuadrados, por lo tanto estaremos interesados con la pareja de transformaciones de Fourier mostradas en las ecuaciones 5.23 y 5.24. La formulación de las ecuaciones 5.19 y 5.20 es usada algunas veces cuando se tienen imágenes rectangulares y es importante conservar el tamaño de la imagen.

El espectro de Fourier, la fase, y el espectro de potencias de funciones discretas unidimensionales y bidimensionales también están dadas por las ecuaciones 5.5, 5.6, 5.7 y 5.11, 5.12, 5.13, respectivamente. La única diferencia es que las variables independientes son discretas.

5.1.3 Algunas propiedades de la transformada de Fourier.

5.1.3.1 Separabilidad.

La pareja de transformadas de Fourier de las ecuaciones 5.23 y 5.24 pueden ser expresadas en forma separada como se muestra en las ecuaciones 5.25 y 5.26.

$$F(u,v)=\frac{1}{N}\sum_{x=0}^{N-1}e^{-j2\pi ux/N}\sum_{y=0}^{N-1}f(x,y)e^{-j2\pi vy/N} \qquad (5.25)$$

para $u,v = 0, 1, 2,...,N - 1$.

$$f(x,y)=\frac{1}{N}\sum_{u=0}^{N-1}e^{j2\pi ux/N}\sum_{v=0}^{N-1}F(u,v)e^{j2\pi vy/N} \qquad (5.26)$$

para $x,y = 0, 1, 2,...,N - 1$.

La principal ventaja de la propiedad de separabilidad es que $F(u,v)$ o $f(x,y)$ pueden obtenerse en dos pasos, aplicando sucesivamente la transformada de Fourier en una dimensión o su inversa. Esta ventaja se hace evidente si la ecuación 5.25 se expresa de la forma mostrada en la ecuación 5.27.

$$F(u,v)=\frac{1}{N}\sum_{x=0}^{N-1}F(x,v)e^{-j2\pi ux/N} \qquad (5.27)$$

donde definimos a $F(x,v)$ en la ecuación 5.28,

$$F(x,v) = N\left[\frac{1}{N}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi vy/N}\right] \qquad (5.28)$$

para cada valor de x, la expresión dentro de los paréntesis en la ecuación 5.28 es una transformada en una dimensión, con valores de frecuencia v = 0, 1, 2,...,N - 1. De esta forma la función bidimensional $F(x,v)$ se obtiene tomando una transformada a lo largo de cada renglón de $f(x,y)$ y multiplicando el resultado por N. El resultado deseado, $F(u,v)$, es entonces obtenido tomando una transformada a lo largo de cada columna de $F(x,v)$, como se indica en la ecuación 5.27. El procedimiento se resume en la Fig. 5.2. El mismo resultado puede ser obtenido tomando primero las transformadas a lo largo de las columnas de $f(x,y)$ y entonces a lo largo de los renglones de los resultados.

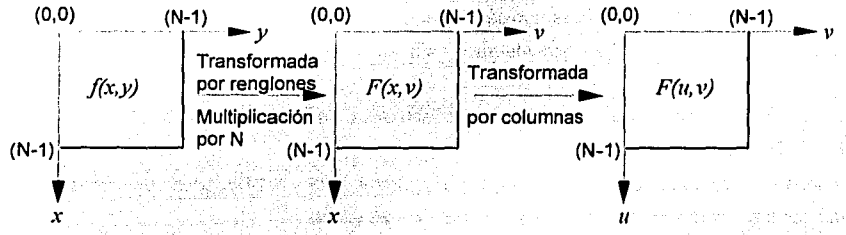


Fig. 5.2 Proceso para obtener la transformada de Fourier bidimensional como una serie de transformadas unidimensionales.

5.1.3.2 Translación.

Las propiedades de translación de la pareja de transformadas de Fourier son las mostradas en la ecuaciones 5.29 y 5.30

$$f(x,y)e^{j2\pi(u_0x + v_0y)/N} \Leftrightarrow F(u-u_0, v-v_0) \qquad (5.29)$$

$$f(x-x_0, y-y_0) \Leftrightarrow F(u,v)e^{-j2\pi(ux_0 + vy_0)/N} \qquad (5.30)$$

donde la doble flecha indica la correspondencia entre una función y su transformada de Fourier (y viceversa), como en las ecuaciones 5.23 y 5.24.

La ecuación 5.29 muestra que multiplicando $f(x,y)$ por el término exponencial indicado y tomando la transformada del producto obtenemos un cambio del origen en el plano de frecuencias al punto (u_0, v_0). En forma similar, multiplicando $F(u,v)$ por el término exponencial mostrando y tomando la transformada inversa movemos el origen del plano espacial a (x_0, y_0).

Para los algoritmos de filtrado haremos uso de la ecuación 5.29 con los valores de $u_0 = v_0 = N/2$,

$$e^{j2\pi(u_0x + v_0y)/N} = e^{j\pi(x+y)}$$

$$= (-1)^{x+y}$$

de lo anterior obtenemos la ecuación 5.31.

$$f(x,y)(-1)^{x+y} \Leftrightarrow F(u-N/2, v-N/2) \qquad (5.31)$$

De esta forma el origen de la transformada de Fourier de $f(x,y)$ puede ser movida al centro de su correspondiente frecuencia cuadrada N X N simplemente multiplicando $f(x,y)$ por $(-1)^{x+y}$. En el caso de una variable este cambio se reduce a la multiplicación de $f(x)$ por el término $(-1)^x$.

Note de la ecuación 5.30 que un cambio en $f(x,y)$ no afecta la magnitud de su transformada de Fourier, por lo que es válida la expresión 5.32.

$$\left|F(u,v)e^{-j2\pi(ux_0+vy_0)/N}\right| = \left|F(u,v)\right| \qquad (5.32)$$

5.1.3.3 Periodicidad y simetría conjugada.

La transformada discreta de Fourier y su inversa son periódicas teniendo como periodo a N; esto se expresa en la ecuación 5.33

$$F(u,v) = F(u+N,v) = F(u,v+N) = F(u+N,v+N) \qquad (5.33)$$

La validez de esta propiedad puede demostrarse substituyendo directamente las variables $(u+N)$ y $(v+N)$ en la ecuación 5.32. Aunque la ecuación 5.33 indica que $F(u,v)$ se repite para un número infinito de valores de u y v, solamente los valores de N de cada variable en cualquier variable son requeridos para obtener $f(x,y)$ de $F(u,v)$. En otras palabras, es necesario solamente un periodo de la transformada para especificar completamente a $F(u,v)$ en el dominio de las frecuencias. Comentarios similares se aplican a $f(x,y)$ en el dominio espacial.

Si $f(x,y)$ es real, la transformada de Fourier exhibe también simetría inversa, como esta expresado en la ecuación 5.34.

$$F(u,v) = F^*(-u,-v) \qquad (5.34)$$

o bien expresándolo en forma más interesante como se muestra en 5.35

$$\left|F(u,v)\right| = \left|F(-u,-v)\right| \qquad (5.35)$$

donde $F^*(u,v)$ es el conjugado complejo de $F(u,v)$. Para examinar las implicaciones de las ecuaciones 5.33 y 5.35 en el despliegue de la magnitud de la transformada, consideraremos el caso de una variable donde $F(u) = F(u+N)$ y $|F(u)| = |F(-u)|$. La propiedad de periodicidad indica que $F(u)$ tiene un periodo de tamaño N, y la propiedad de simetría muestra que la magnitud de la transformada esta centrada en el origen como lo muestra la figura 5.3(a). La figura 5.3(a) y los comentarios precedentes demuestran que las magnitudes de los valores de la transformada de $(N/2)+1$ a $N-1$ son reflejos de los valores de la mitad del periodo colocado a la izquierda del origen. Debido a que la transformada discreta de Fourier ha sido formulada para valores de u en el

intervalo [0,N-1], el resultado de esta formulación nos da dos mitades separadas de un mismo periodo en este intervalo. Para mostrar un periodo completo, es necesario mover el origen de la transformada al punto u=N/2, como nos muestra la fig. 5.3(b).

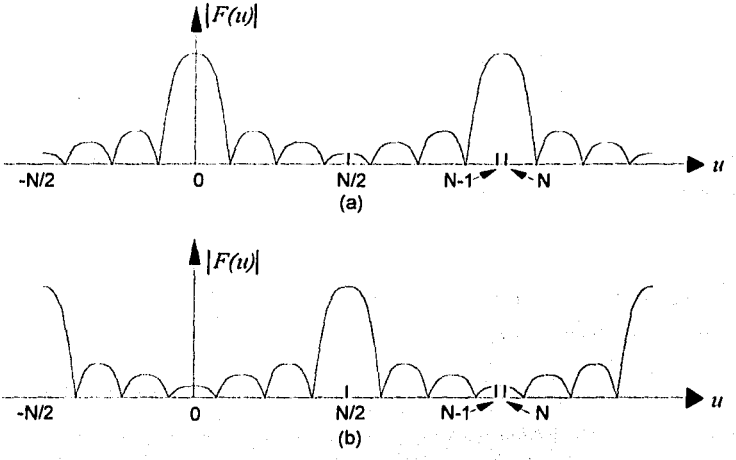
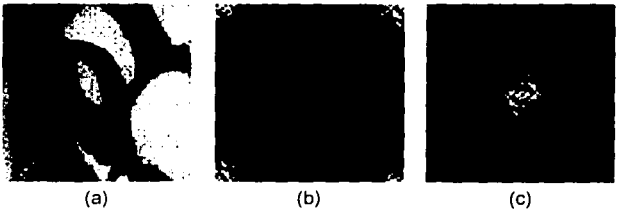


Fig. 5.3 Ilustración de la propiedad de periodicidad de la transformada de Fourier. (a) El espectro de Fourier muestra las mitades de un mismo periodo en el intervalo [0,N-1]; (b) El cambio en el espectro muestra un periodo completo en el mismo intervalo.

La misma observación es válida para la magnitud de la transformada de Fourier bidimensional, con la excepción que los resultados son considerablemente más difíciles de interpretar si el origen de la transformada no es cambiado al punto de frecuencia (N/2,N/2). Las figuras 5.4 muestran esta diferencia.



(a) (b) (c)

Fig. 5.4 (a) Una imagen; (b) espectro de Fourier sin cambios; (c) cambio del espectro de Fourier al centro del plano de frecuencias.

5.1.3.4 Convolución.

La convolución de dos funciones $f(x)$ y $g(x)$, denotada por $f(x) * g(x)$ esta definida por la expresión 5.36.

$$f(x) * g(x) = \int_{-\infty}^{\infty} f(\alpha) g(x-\alpha) d\alpha \qquad (5.36)$$

donde α es una variable comodín de integración. El mecanismo de la integral de convolución no es fácil de visualizar, comenzaremos la discusión con un ejemplo que ilustra gráficamente el uso de la ecuación 5.36.

El ejemplo demuestra la convolución de las funciones $f(x)$ y $g(x)$ mostradas en las figs. 5.5(a) y 5.5(b) respectivamente. Antes de que la integración se lleve a cabo, se debe formar la función $g(x-\alpha)$. Para hacer esto requerimos los dos pasos mostrados en las figs. 5.5(c) y 5.5(d). Esta operación simplemente refleja a $g(\alpha)$ en el origen para dar $g(-\alpha)$ y desplaza esta función por medio de x. Entonces, para cualquier valor de x, multiplicamos $f(\alpha)$ por el correspondiente $g(x-\alpha)$ e integramos el producto desde $-\infty$ hasta ∞ El producto de $f(\alpha)$ y $g(x-\alpha)$ es la porción sombreada de la fig. 5.5(e). Esta figura es valida para $0 \le x \le 1$. El producto es 0 para valores de α fuera del intervalo [0,x], así $f(x) * g(x) = x/2$, lo cual es el área de la región sombreada en la fig. 5.5(e). Para x en el intervalo [1,2], se aplica la fig. 5.5(f), y $f(x) * g(x) = (1 - x/2)$. Como $f(\alpha) g(x-\alpha)$ es cero para valores de x fuera del intervalo [0,2], finalmente obtenemos

$$F(x) * g(x) = \begin{cases} x/2 & 0 \le x \le 1 \\ 1 - x/2 & 1 \le x \le 2 \\ 0 & \text{para cualquier otro} \end{cases}$$

La fig. 5.5(g) muestra el resultado.

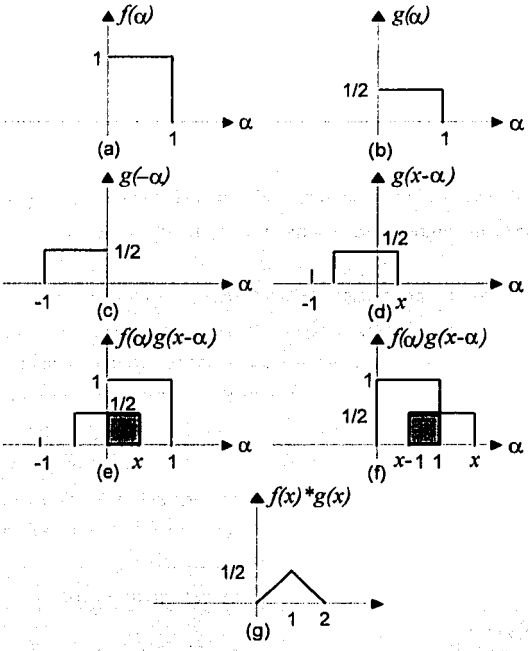


Fig. 5.5 Ilustración gráfica de la convolución. Las áreas sombreadas indican regiones donde el producto es diferente de cero.

La importancia de la convolución en el análisis en el dominio de las frecuencia es que $f(x)*g(x)$ y $F(u)G(u)$ constituyen una pareja de transformadas de Fourier. En otras palabras, si $f(x)$ tiene la transformada de Fourier $F(u)$ y $g(x)$ tiene la transformada de Fourier $G(u)$, entonces $f(x)*g(x)$ tiene su transformada de Fourier $F(u)G(u)$. Este resultado, se establece formalmente en la expresión 5.37

$$f(x)*g(x) \Leftrightarrow F(u)G(u) \qquad (5.37)$$

la cual indica que la convolución en dominio de x puede obtenerse tomando la transformada inversa de Fourier del producto $F(u)G(u)$. Un resultado análogo es que la convolución en el dominio de las frecuencias se reduce a una multiplicación en el dominio de x, esto se expresa en la expresión 5.38.

$$f(x)g(x) \Leftrightarrow F(u)*G(u) \qquad (5.38)$$

Estos dos resultados son comúnmente referidos como el teorema de convolución.

La convolución discreta de $f_e(x)$ y $g_e(x)$ esta definido por la expresión 5.40

$$f_c(x) * g_c(x) = \sum_{m=0}^{M-1} f_c(m) g_c(x-m) \tag{5.40}$$

para x = 0, 1, 2,..., M-1. La función de convolución es un arreglo discreto, periódico de largo M, con los valores x = 0, 1, 2,..., M-1 describiendo un periodo completo de $f_c(x) * g_c(x)$.

Los mecanismos de la convolución discreta son básicamente los mismos de la convolución continua. La únicas diferencias son que los desplazamientos tienen lugar en los incrementos discretos correspondientes a la separación entre muestras y que una sumatoria reemplaza la integración. Similarmente, las ecuaciones 5.37 y 5.38 también son aplicables al caso discreto.

La convolución bidimensional es análoga a la forma de la ecuación 5.36. Así que para dos funciones $f(x,y)$ y $g(x,y)$, podemos expresarla convolución con la ecuación 5.41.

$$f(x,y) * g(x,y) = \int_{-\infty}^{\infty}\int f(\alpha,\beta) g(x-\alpha, y-\beta) d\alpha d\beta \tag{5.41}$$

El teorema de convolución en dos dimensiones, es expresado por las relaciones 5.42 y 5.43

$$f(x,y) * g(x,y) \Leftrightarrow F(u,v)G(u,v) \tag{5.42}$$

$$f(x,y) g(x,y) \Leftrightarrow F(u,v) * G(u,v) \tag{5.43}$$

La convolución bidimensional de $f_c(x,y)$ y $g_c(x,y)$ esta definido por la relación 5.48.

$$f_c(x,y) * g_c(x,y) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f_c(m,n) g_c(x-m, y-n) \tag{5.48}$$

para x = 0, 1, 2,..., M-1 y y = 0, 1, 2,..., N-1.

5.1.4 Sistemas lineales.

En todo sistema hay una función de entrada (o función de excitación) y una función de salida (o función de respuesta). Un sistema está completamente caracterizado si se conoce la naturaleza de la dependencia de la salida sobre la entrada.

Si se supone que la respuesta de un sistema a la excitación $f_i(t)$ es la función $f_o(t)$, y si la respuesta de ese sistema a la excitación $f_i(t) = a_1 f_{i1}(t) + a_2 f_{i2}(t)$ es $f_o(t) = a_1 f_{o1}(t) + a_2 f_{o2}(t)$ se dice que es un sistema lineal.

Por tanto, un sistema lineal se puede definir como un sistema al cual se le puede aplicar el principio de superposición que establece: la respuesta de un sistema lineal producida por varias excitaciones actuando simultáneamente es igual a la suma de las respuestas que produce en el sistema cada excitación actuando por separado, este principio se muestra en la ecuación 5.48

$$\{a_1 f_{i1}(t) + a_2 f_{i2}(t)\} = a_1\{f_{i1}(t)\} + a_2\{f_{i2}(t)\} \tag{5.48}$$

Si la respuesta de un sistema a la excitación $f_i(t)$ es la función $f_o(t)$, y si la respuesta de ese sistema a la excitación $f_i(t-t_0)$ es la función $f_o(t-t_0)$, se dice que es un sistema invariante en el tiempo esto se expresa en 5.49.

$$\{f_i(t+t_0)\} = f_0(t+t_0) \tag{5.49}$$

Ahora definimos la función impulso unitario $\partial(t)$, conocida como función delta, la cual generalmente se expresa mediante las ecuaciones 5.50 y 5.51,

$$\partial(t) = \begin{cases} 0 & \text{si } t \neq 0 \\ \infty & \text{si } t = 0 \end{cases} \tag{5.50}$$

$$\int_{-\infty}^{\infty} \partial(t)dt = \int_{-\varepsilon}^{\varepsilon} \partial(t)dt = 1, \quad \varepsilon > 0 \tag{5.51}$$

La ecuación 5.50 indica que $\partial(t)$ es cero excepto en t=0, donde se hace infinita, de tal manera que cumple 5.51.

La transformada de Fourier de la función impulso se expresa en la ecuación 5.52

$$\Im[\partial(t)] = \int_{-\infty}^{\infty} \partial(t) e^{-jwt} dt = e^{-jwt}\Big|_{t=0} = 1 \tag{5.52}$$

La respuesta de un sistema lineal al impulso unitario $\partial(t)$, se denota por $h(t)$. Simbólicamente esto se expresa como en 5.53

$$\{\partial(t)\} = h(t) \tag{5.53}$$

Si el sistema es invariante, entonces, según la ecuación se observa que su respuesta a $\partial(t-\tau)$ está dada por $h(t-\tau)$, es decir como en la ecuación 5.54,

$$\{\partial(t-\tau)\} = h(t-\tau) \tag{5.54}$$

Entonces la respuesta $f_o(t)$ de un sistema lineal e invariante, a una entrada arbitraria $f_i(t)$, se puede expresar como la convolución de la entrada $f_i(t)$ y de la respuesta del sistema al impulso unitario h(t), es decir como en 5.55

$$f_0(t) = \int_{-\infty}^{\infty} f_i(\tau)h(t-\tau)d\tau = f_i(t) * h(t) \tag{5.55}$$

A la transformada de Fourier de la respuesta al impulso unitario de un sistema lineal, se denomina función del sistema y se expresa como en 5.56 y 5.57

$$H(w) = \Im[h(t)] = \int_{-\infty}^{\infty} h(t) e^{-jwt} dt \tag{5.56}$$

$$h(t) = \Im^{-1}[H(w)] = \frac{1}{2\pi}\int_{-\infty}^{\infty} H(w) e^{jwt} dw \tag{5.57}$$

Las ecuaciones 5.56 y 5.57 indican que la respuesta al impulso unitario, y la función del sistema constituyen un par de transformadas de Fourier.

Si $F_i(w)$ y $F_o(w)$ denotan las transformadas de Fourier, de la entrada $f_i(t)$ y de la salida $f_o(t)$ de un sistema lineal, respectivamente, entonces obtenemos las ecuaciones 5.58 y 5.59

$$F_o(w) = F_i(w)H(w) \tag{5.58}$$

$$f_o(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} Fi(w)H(w)e^{jwt}dw \tag{5.59}$$

La ecuación muestra que el espectro de frecuencia de la respuesta, $F_o(w)$, está relacionado con el espectro de frecuencia de la fuente, $F_i(w)$ por medio de la función del sistema, $H(w)$. Esto se ilustra en la fig. 5.6.

Se observa que $H(w)$ actúa como una función ponderadora de las componentes de diferente frecuencia en la entrada. En este sentido la relación indica la característica de filtro del sistema lineal. Si la característica de filtro es el interés principal, entonces generalmente se hace referencia al sistema como a un filtro.

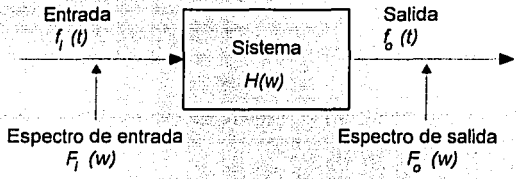


Fig. 5.6 Ilustración de la relación 5.58.

5.1.5 La transformada rápida de Fourier.

El número requerido de multiplicaciones y sumas complejas para implementar la ecuación 5.16 es proporcional a N^2. Esto es, que para cada uno de los N valores de u, la expansión de sumatorias requiere N multiplicaciones complejas de $f(x)$ por $e^{-j2\pi ux/N}$ y N-1 sumas de los resultados. Los términos de $e^{-j2\pi ux/N}$ pueden ser calculados una sola vez y guardarlos en una tabla para todas las aplicaciones subsecuentes. Por esta razón, la multiplicación de u por x en estos términos no es usualmente considerada una parte directa de la implementación.

La descomposición apropiada de la ecuación 5.16 puede hacer el número de operaciones de suma y multiplicación proporcionales a $N\log_2 N$. El procedimiento de descomposición es llamado el algoritmo de la transformada rápida de Fourier (FFT Fast Fourier Transform). La reducción en la proporcionalidad de N^2 a $N\log_2 N$ operaciones representa una reducción en el esfuerzo computacional. Obviamente, la aproximación de la transformada rápida ofrece una ventaja

computacional considerable sobre implementaciones directas de la transformada de Fourier, particularmente cuando N es relativamente muy grande.

En la siguiente discusión, desarrollaremos un algoritmo de transformada rápida de Fourier de una variable. Como fue indicado, una transformada de Fourier bidimensional puede ser obtenida por sucesivos pasos de la transformada unidimensional.

5.1.6 Algoritmo de la transformada rápida de Fourier.

El algoritmo de la transformada rápida de Fourier desarrollada es esta sección esta basado en el método de particiones sucesivas. Por conveniencia expresamos la ecuación 5.16 en la forma mostrada en la ecuación 5.82.

$$F(u) = \frac{1}{N}\sum_{x=0}^{N-1} f(x) W_N^{ux} \tag{5.82}$$

en donde definimos a W como la expresión mostrada en 5.83.

$$W_N = e^{-j2\pi / N} \tag{5.83}$$

y asumimos que N es de la forma expresada en 5.73.

$$N = 2^n \tag{5.73}$$

dos n es un entero positivo. Por consiguiente N puede ser expresado como lo muestra la ecuación 5.74.

$$N = 2M \tag{5.74}$$

donde M es también un entero positivo. la substitución de la ecuación 5.74 en la ecuación 5.82 nos da la expresión 5.75.

$$\begin{aligned} F(u) &= \frac{1}{2M}\sum_{x=0}^{2M-1} f(x) W_{2M}^{ux} \\ &= \frac{1}{2}\left[\frac{1}{M}\sum_{x=0}^{M-1} f(2x) W_{2M}^{u(2x)} + \frac{1}{M}\sum_{x=0}^{M-1} f(2x+1) W_{2M}^{u(2x+1)}\right] \end{aligned} \tag{5.75}$$

De la ecuación 5.83, $W_M^{ux} = W_M^{ux}$, de esta forma la ecuación 5.75 puede ser expresada como se muestra en la ecuación 5.76.

$$F(u) = \frac{1}{2}\left[\frac{1}{M}\sum_{x=0}^{M-1} f(2x) W_M^{ux} + \frac{1}{M}\sum_{x=0}^{M-1} f(2x+1) W_M^{ux} W_{2M}^{u}\right] \tag{5.76}$$

Si definimos la ecuación 5.77

$$F_{\text{par}}(u) = \frac{1}{M}\sum_{x=0}^{M-1} f(2x)W_M^{ux} \tag{5.77}$$

para u = 0, 1, 2,..., M-1 y la ecuación 5.78

$$F_{\text{impar}}(u) = \frac{1}{M}\sum_{x=0}^{M-1} f(2x+1)W_M^{ux} \tag{5.78}$$

para u = 0, 1, 2,..., M-1, se reduce la ecuación 5.76 a la ecuación 5.79.

$$F(u) = \frac{1}{2}\left[F_{\text{par}}(u) + F_{\text{impar}}(u)W_{2M}^{u}\right] \tag{5.79}$$

También, como $W_M^{u+M} = W_M^u$ y $W_{2M}^{u+M} = -W_{2M}^u$, las ecuaciones 5.77 y 5.78 nos dan la ecuación 5.80.

$$F(u+M) = \frac{1}{2}\left[F_{\text{par}}(u) - F_{\text{impar}}(u)W_{2M}^{u}\right] \tag{5.80}$$

El análisis cuidadoso de las ecuaciones 5.77 y 5.80 revela algunas propiedades interesantes de estas expresiones. Una transformada de N puntos puede ser calculada dividiendo la expresión original en dos partes, como es indicado en las ecuaciones 5.79 y 5.80. El cálculo de la primera mitad de $F(u)$ requiere la evaluación de dos transformadas de (N/2) puntos dadas en las ecuaciones 5.77 y 5.78. El valor resultante de $F_{\text{par}}(u)$ y $F_{\text{impar}}(u)$ son entonces substituidos en la ecuación 5.79 para obtener $F(u)$ para u = 0, 1, 2,..., (N/2-1). La otra mitad se obtiene directamente de la ecuación 5.80 sin evaluaciones adicionales en la transformada.

Para examinar las implicaciones computacionales de este procedimiento, sea $m(n)$ y $a(n)$ la representación del número de multiplicaciones y adiciones complejas, respectivamente, requeridas para implementarlo. Como antes, el número de muestras es 2^n, donde n es un positivo entero. Suponiendo primero que n=1. Una transformada de dos puntos requiere la evaluación de $F(0)$; entonces $F(1)$ se obtiene de la ecuación 5.80. Para obtener $F(0)$ calculamos primero $F_{\text{par}}(0)$ y $F_{\text{impar}}(0)$. En este caso M=1 y la ecuaciones 5.77 y 5.78 son transformadas de un punto. Ninguna multiplicación o adición es requerida para obtener $F_{\text{par}}(0)$ y $F_{\text{impar}}(0)$ porque la transformada de Fourier de un punto es la muestra. Una multiplicación de $F_{\text{impar}}(0)$ por W_2^0 y una adición nos da $F(0)$ utilizando la ecuación 5.79. De esta forma $F(1)$ se obtiene de la ecuación 5.80 con una adición más (la substracción es considerada de la misma forma que una suma). Como $F_{\text{impar}}(0)W_2^0$ ya había sido computada, el número total de operaciones requeridas para una transformada de dos puntos consiste de $m(1)$=1 multiplicaciones y $a(1)$=2 adiciones.

El siguiente valor permitido para n es 2. De acuerdo al desarrollo anterior, una transformada de cuatro puntos puede ser dividida en dos partes. La primera mitad de $F(u)$ requiere la evaluación de dos transformadas de dos puntos, como en las ecuaciones 5.77 y 5.78 para M=2. Una transformada de dos puntos requiere $m(1)$ multiplicaciones y $a(1)$ adiciones, así la evaluación de estas dos ecuaciones requiere un total de $2m(1)$ multiplicaciones y $2a(1)$ adiciones. Son necesarias dos multiplicaciones y adiciones para obtener $F(0)$ y $F(1)$ empleando la ecuación 5.79. Debido a que $F_{\text{impar}}(u)W_{2M}^{u}$ ya había sido calculado calculado para u={0,1}, dos adiciones más nos dan $F(2)$ y $F(3)$. El total entonces es $m(2)=2m(1)+2$ y $a(2)=2a(1)+4$.

Cuando n es igual a 3, se consideran dos transformadas de cuatro puntos en la evaluación de $F_{\text{par}}(u)$ y $F_{\text{impar}}(u)$. Ello requiere $2m(2)$ multiplicaciones y $2a(2)$ adiciones. Cuatro multiplicaciones y ocho adiciones más nos dan la transformada total. El total entonces es $m(3)=2m(2)+4$ y $a(3)=2a(2)+8$.

Continuando este argumento para cualquier valor entero positivo de n nos lleva a la expresión recursiva del número de multiplicaciones y adiciones requeridas para implementar la transformada rápida de Fourier, que se muestran en las ecuaciones 5.81 y 5.82.

$$m(n) = 2m(n-1) + 2^{n-1} \qquad n \geq 1 \qquad (5.81)$$

$$a(n) = 2a(n-1) + 2^{n} \qquad n \geq 1 \qquad (5.82)$$

donde $m(0)=0$ y $a(0)=0$ porque la transformada de un punto simple no requiere multiplicaciones ni divisiones.

La implementación de las ecuaciones 5.77, 5.78, 5.79, 5.80 constituye una sucesión doble del algoritmo de transformada rápida de Fourier. Este nombre viene del método de calcular una transformada de dos puntos de dos transformadas de un punto, una transformada de cuatro puntos de dos transformadas de dos puntos, y así sucesivamente, para cualquier N que sea igual a un entero potencia de 2.

5.1.7 La transformada rápida de Fourier inversa.

Hasta ahora no hemos discutido nada acerca de la transformada inversa de Fourier. La razón es que el algoritmo para implementar la transformada discreta de Fourier también pueda usarse (con pequeñas modificaciones en la entrada) para calcular la inversa. Para mostrar esto, retornaremos a las ecuaciones 5.16 y 5.17.

Tomando el conjugado complejo de la ecuación 5.17 y dividiendo ambos lados entre N nos da la ecuación 5.83.

$$\frac{1}{N} f^*(x) = \frac{1}{N} \sum_{u=0}^{N-1} F(u) e^{-j2\pi ux/N} \qquad (5.83)$$

Comparando este resultado con la ecuación 5.16 nos muestra que el lado derecho de la ecuación 5.83 tiene la forma de la transformada de Fourier. De esta forma poniendo en la entrada $F^*(u)$ a un algoritmo diseñado para calcular la transformada nos da la cantidad $f^*(x)/N$. Tomando el complejo conjugado y multiplicándolo por N nos da la inversa $f(x)$ deseada.

Para un arreglo bidimensional cuadrado tomamos el complejo conjugado de la ecuación 5.24, que es la mostrada en la ecuación 5.73.

$$f^*(x,y) = \frac{1}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F^*(u,v) e^{-j2\pi(ux+vy)/N} \qquad (5.73)$$

la cual es la forma de una transformada bidimensional. Por lo tanto, dando como entrada $F^*(u,v)$ en un algoritmo diseñado para calcular la transformada obtenemos $f^*(x,y)$. Al tomar el complejo conjugado de este resultado nos da $f(x,y)$. Cuando $f(x)$ o $f(x,y)$ es real, la operación del complejo conjugado es innecesaria, porque $f(x) = f^*(x)$ y $f(x,y) = f^*(x,y)$ para funciones reales.

5.2 Filtrado paso bajas.

Como fue indicado en la sección 4.1.2.1, los bordes y otros cambios en las formas (como el ruido) en los niveles de gris de una imagen contribuyen significativamente al contenido de altas frecuencias en la transformada de Fourier. Por lo tanto se consigue el suavizado, en el dominio de la frecuencia, atenuando un rango específico de componentes de altas frecuencias en la transformada de la imagen a ser procesada.

Por el propiedad de convolución obtenemos la ecuación 5.74

$$G(u,v) = H(u,v)F(u,v) \qquad (5.74)$$

donde $F(u,v)$ es la transformada de Fourier de la imagen a ser suavizada. El problema es seleccionar una función de transferencia de filtrado $H(u,v)$ que nos de $G(u,v)$ atenuando los componentes de alta frecuencia de $F(u,v)$. Entonces la transformada inversa nos dará la imagen suavizada $g(x,y)$. En las siguientes secciones consideraremos funciones de transferencia de filtrado que afectan las partes real e imaginaria de $F(u,v)$ exactamente de la misma manera. Tales filtros son referidos como filtros de cero cambios en fase porque no alteran la fase de la transformada.

5.2.1 Filtro paso bajas ideal.

Un filtro paso bajas ideal bidimensional (FPBI) es aquel cuya función de transferencia satisface la condición mostrada en 5.75

$$H(u,v) = \begin{cases} 1 & \text{si } D(u,v) \le D_0 \\ 0 & \text{si } D(u,v) > D_0 \end{cases} \qquad (5.75)$$

donde D_0 es una cantidad no negativa, y $D(u,v)$ es la distancia del punto (u,v) al origen del plano de frecuencias, calculada con la expresión 5.76.

$$D(u,v) = (u_2 + v_2)^{1/2} \qquad (5.76)$$

Los filtros paso bajas considerados en este capítulo son radialmente simétricos respecto al origen. Para este tipo de filtros, es suficiente especificar una sección transversal expresada como una función de la distancia al origen a lo largo de una línea radial. La función de transferencia completa puede ser generada rotando la sección transversal 360 grados alrededor del origen. La especificación de los filtros radialmente simétricos centrados en el cuadrado de frecuencia N X N esta basado en la suposición que el origen de la transformada de Fourier ha sido centrado en el cuadrado.

Para una sección transversal de un filtro paso bajas, el punto de transición entre $H(u,v)$=1 y $H(u,v)$=0 es llamado la frecuencia de corte. En el caso de la fig. 5.6, por ejemplo, la frecuencia de corte es D_0. Como la sección transversal es girada en torno al origen, el punto D_0 traza un círculo dando un lugar de frecuencias de corte, todas están a una distancia D_0 del origen. El concepto de frecuencia de corte es muy útil para especificar las características de los filtros. También sirve como una base común para comparar el comportamiento de los diferentes tipos de filtros.

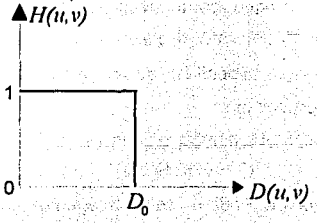


Fig. 5.6 Sección transversal de un filtro paso bajas.

El desempeño de los filtros paso bajas introducidos en esta sección son comparados usando las mismas frecuencias de corte para los lugares geométricos. Una manera de establecer un conjunto de lugares geométricos estándares es calcular círculos que encierren varias cantidades

del total de la potencia de la señal P_T. Esta cantidad es obtenida sumando la potencia de cada punto (u,v) para u,v=0,1,2,...,N-1; esto es como se expresa en la ecuación 5.77

$$P_T = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} P(u,v) \qquad (5.77)$$

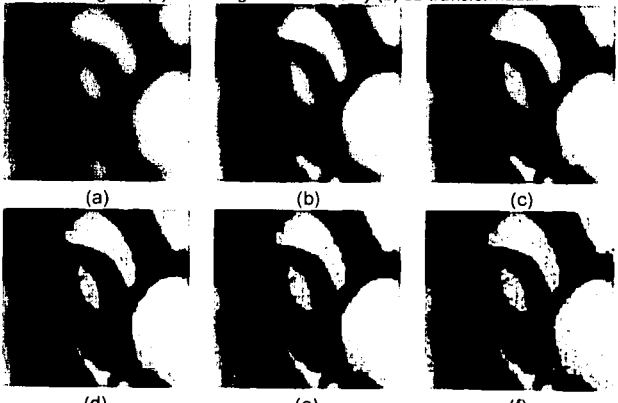
donde $P(u,v)$ esta dado por la ecuación 5.13. Si la transformada ha sido centrada, un círculo de radio r con origen en el centro del plano de frecuencias encierra β por ciento de la potencia, donde β es igual a la expresión 5.78

$$\beta = 100\left[\sum_{u}\sum_{v} P(u,v) / P_T\right] \qquad (5.78)$$

la sumatoria es tomada sobre los valores de (u,v) que están tanto en el interior del círculo como en la circunferencia.

(a) (b)

Fig. 5.7 (a) Una imagen de 64 X 64 y (b) su transformada.



(a) (b) (c)

(d) (e) (f)

Fig. 5.8 Resultado del filtrado paso bajas ideal (a) D_0=5, β =95.41; (b) D_0 =10, β=97.41; (c) D_0 =15, β=98.46; (d) D_0 =20, β=99.12; (e) D_0 =25, β=99.12; (f) D_0 =30; β=99.59

5.2.2 Filtro Butterworth paso bajas.

La función de transferencia de un filtro Butterworth paso bajas (FPBB) de orden n y con frecuencia de corte localizada a una distancia D_0 del origen esta definida por la ecuación 5.79

$$H(u,v) = \frac{1}{1+\left[D(u,v)/D_0\right]^{2n}} \qquad (5.79)$$

donde $D(u,v)$ esta dado por la ecuación 5.76.

A diferencia del filtro paso bajas ideal, la función de transferencia no tiene una forma discontinua que establezca un corte claro entre frecuencias pasadas y filtradas. Para filtros con funciones de transferencia suave, se acostumbra definir una frecuencia corte local en el punto para el cual $H(u,v)$ esta abajo de cierta fracción de su valor máximo. En este caso de la ecuación 5.79, $H(u,v)$=0.5 (50% abajo de su valor máximo que es 1) cuando $D(u,v) = D_0$. Otro valor comúnmente usado es $1/\sqrt{2}$ del valor máximo de $H(u,v)$. Para la ecuación 5.79, la modificación de la ecuación 5.80 nos da el valor deseado cuando $D(u,v) = D_0$.

$$H(u,v) = \frac{1}{1+\left[\sqrt{2}-1\right]\left[D(u,v)/D_0\right]^{2n}} \qquad (5.80)$$

$$H(u,v) = \frac{1}{1+0.414\left[D(u,v)/D_0\right]^{2n}}$$

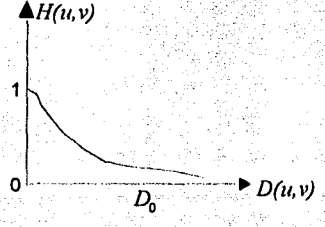


Fig. 5.9 Sección transversal de un filtro Butterworth paso bajas para n = 1

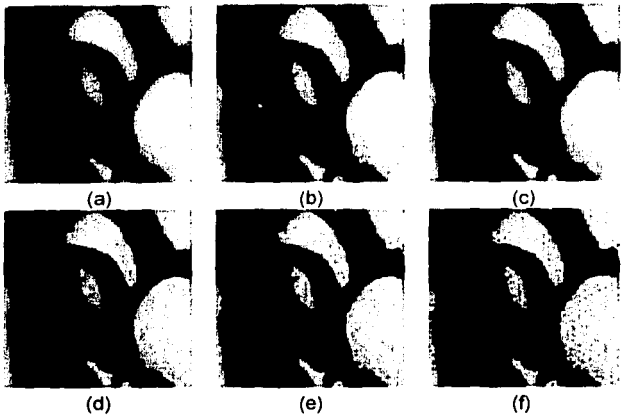


Fig. 5.10 Resultados del filtrado Butterworth paso bajas de primer orden (a) D_0=5, β =95.76; (b) D_0=10, β=98.25; (c) D_0 =15, β=99.00; (d) D_0 =20, β=99.37; (e) D_0 =25, β=99.53; (f) D_0 =30; β=99.76

5.3 Filtrado paso altas.

Una imagen puede hacerse borrosa atenuando los componentes de alta frecuencia de su transformada de Fourier. Debido a a que los bordes y otros cambio abruptos en los niveles de gris están asociados con los componentes de altas frecuencias, el agudizamiento de bordes en una imagen se puede conseguir en el dominio de las frecuencias con un proceso de filtrado paso altas, que atenúa los componentes de bajas frecuencias sin cambiar la información de altas frecuencias en la transformada de Fourier.

5.3.1 Filtro paso altas ideal.

Un filtro paso altas ideal bidimensional (FPAI) es aquel cuya función de transferencia satisface la relación mostrada en 5.81

$$H(u,v)=\begin{cases}0 & \text{si } D(u,v)\le D_0\\ 1 & \text{si } D(u,v)>D_0\end{cases} \qquad (5.81)$$

donde D_0 es la distancia de corte medida desde el origen del plano de frecuencias, y $D(u,v)$ esta dado por la ecuación 5.76. Este filtro es opuesto al filtro paso bajas ideal porque atenúa completamente todas las frecuencias dentro del círculo de radio D_0 mientras que pasan sin atenuación, todas las frecuencias afuera del círculo.

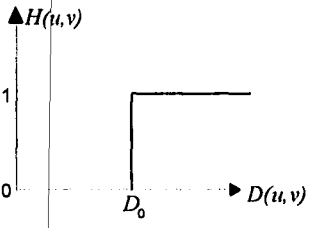
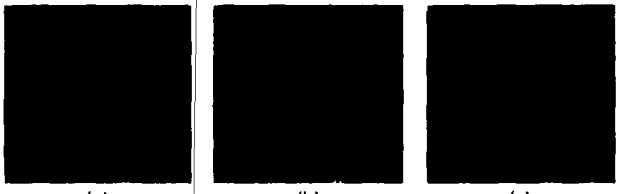


Fig. 5.11 Sección transversal de un filtro paso altas.



(a) (b) (c)

Fig. 5.12 Resultado del filtrado paso altas ideal (a) D_0=1, β =22.28; (b) D_0=3, β=8.64; (c) D_0 =5, β=4.59

5.3.2 Filtro Butterworth paso altas.

La función de transferencia de un filtro Butterworth paso altas (FBPA) de orden n y con una frecuencia de corte localizada a una distancia D_0 del origen esta definido por la relación 5.82

$$H(u,v) = \frac{1}{1+\left[D_0 / D(u,v)\right]^{2n}} \qquad (5.82)$$

donde $D(u,v)$ está definido por la ecuación 5.76.

Note que cuando $D(u,v) = D_0$, $H(u,v)$ está abajo de 1/2 de su valor máximo. Como en el caso del filtro Butterworth paso bajas, es común en la práctica seleccionar una frecuencia de corte localizada en los puntos para los cuales $H(u,v)$ esté abajo de $1/\sqrt{2}$ de su valor máximo. La ecuación 5.82 es fácilmente modificada para satisfacer esta restricción, usando la escala de la ecuación 5.83

$$H(u,v) = \frac{1}{1+\left[\sqrt{2}-1\right]\left[D_0 / D(u,v)\right]^{2n}} \qquad (5.83)$$

$$H(u,v) = \frac{1}{1+0.414\left[D_0 / D(u,v)\right]^{2n}}$$

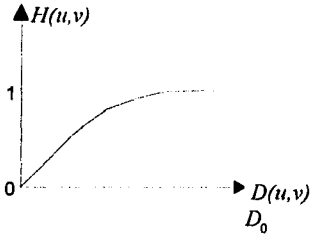


Fig. 5.13 Sección transversal de un filtro Butterworth paso altas para n = 1.

(a) (b) (c)

Fig. 5.14 Resultado del filtrado Butterworth paso altas (a) D_0=1, β =74.97; (b) D_0=3, β=24.51; (c) D_0=5, β=11.38

Apéndice

Programa de aplicación para el realce de imágenes.

En este apéndice se incluye el código en lenguaje C de los programas que emplean las técnicas que se explicaron a lo largo del presente trabajo. Todas las imágenes incluidas en el texto se obtuvieron con estos programas, los cuales fueron compilados con Borland C versión 3.1, usando el modelo de memoria Huge. Si se prefiere pueden ser compilados también con Turbo C versión 2.0. La aplicación está dividida en tres programas diferentes, cada uno de los cuales comprende las técnicas de un capítulo.
El siguiente es el primer programa y corresponde a las técnicas del capítulo 3 realce aplicando operaciones sobre puntos. También se pueden observar los conceptos introductorios expuestos en el capítulo 2, programación en lenguaje C aplicada al procesamiento digital de imágenes.

```
/*          UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
        ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES ARAGON
                    INGENIERIA EN COMPUTACION
        DESARROLLO DE TECNICAS PARA EL REALCE DE IMAGENES
              PROGRAMA DE APLICACION DE LAS TECNICAS
                  CORRESPONDIENTE AL CAPITULO 3,
              REALCE APLICANDO OPERACIONES SOBRE PUNTOS.
                PROGRAMADO POR MIGUEL MIRANDA MIRANDA.
                      NO. CUENTA 8718694-5.          */
#include <stdio.h>
#include <dos.h>
#include <alloc.h>
#include <math.h>
#include <stdlib.h>
#include <conio.h>
#include <float.h>
#include <ctype.h>
#include <mem.h>
#include <string.h>
#include <bios.h>

#define IMAGEN 0x13
#define TEXTO 0x03
#define PALETA 255.0

typedef unsigned char BYTE;
typedef BYTE huge *AP_IMG;
typedef unsigned long Histograma[256];
```

```
AP_IMG imagen;

typedef unsigned long DWORD;

DWORD r,c;
unsigned char *memvideo;
union REGS registros;
char O;
Histograma H;
float Max, Min;

/*  FUNCION QUE LEEN UNA IMAGEN DE UN ARCHIVO. LA IMAGEN NO DEBE
    TENER NINGUN FORMATO ESPECIFICO, SE NECESITA UNICAMENTE EL MAPA
    DE BITS Y CONOCER SUS DIMENSIONES EXACTAS EN CASO DE QUE NO SEA
    UNA IMAGEN CUADRADA.  */
void Leer_imagen(void)  {
 FILE *archivo;
 char nombre_archivo[40], answer;
 int i,j;
 unsigned char cnc, ccont, p;
 unsigned long int cont;

 do  {
    printf("\nESCRIBE EL NOMBRE DEL ARCHIVO: ");
    scanf("%s",nombre_archivo);
    archivo = fopen(nombre_archivo,"rb");
            if (archivo == NULL)  {
               printf("\nEL ARCHIVO %s NO EXISTE",nombre_archivo);
               printf("\nQUIERES INTENTARLO DE NUEVO? OPRIME LA TECLA N SI NO");
               printf("\nDESEAS INTENTARLO, O CUALQUIER OTRA TECLA EN CASO");
               printf("\nCONTRARIO");
               answer = getche(); answer = toupper(answer);   }
 }while((answer != 'N') && (archivo == NULL));
 if (answer == 'N') exit(0);
 do {
    printf("\nLA IMAGEN ES CUADRADA (N x N ELEMENTOS) (S/N)? ");
    cnc = getche();
 }while((cnc != 'n') && (cnc != 'N') && (cnc != 's') && (cnc != 'S'));
 if ((cnc == 's') || (cnc == 'S'))  {
    cont = 0;
    do  {
             ccont = fgetc(archivo);
             cont++;
    }while(!feof(archivo));
    c = (int)sqrt(cont);
    r = c;
    printf("\nLAS DIMENSIONES DE ANCHO Y LARGO SON %u",r);
    fclose(archivo);
    archivo = fopen(nombre_archivo,"rb");
            if (archivo == NULL)  {
               printf("\n EL ARCHIVO %s TIENE PROBLEMAS ",nombre_archivo);
               getch();              }   }
 else            {
```

```
    printf("\nCUANTAS COLUMNAS TIENE LA IMAGEN? ");
    scanf("%d",&c);
    printf("\nCUANTOS RENGLONES TIENE LA IMAGEN? ");
    scanf("%d",&r);   }
  imagen =(unsigned char *)farmalloc(r*c*sizeof(BYTE));
  if (imagen == NULL)          {
          printf("NO HAY ESPACIO EN MEMORIA \n");
          exit (1);   }
          for(i=0; i<r ;i++)
            for(j=0; j<c ;j++)   {
                    if (feof(archivo))  {
                      printf("SON ERRONEAS LAS DIMENSIONES DE LA IMAGEN ");
                      getch();
                      exit(0);    }
                    else
                      imagen[c*i+j]=fgetc(archivo);  }
  p = fgetc(archivo);
  if (feof(archivo))  {
    printf("\nLA IMAGEN HA SIDO LEIDA ");
    printf("\nOPRIMA CUALQUIER TECLA PARA CONTINUAR ");
    getch();      }
   else          {
          printf("SON ERRONEAS LAS DIMENSIONES DE LA IMAGEN ");
          exit(0);     }
  fclose(archivo);
  return;   }

/* FUNCION QUE CAMBIA EL MODO DE VIDEO UTILIZANDO UNA INTERRUPCION */
void Poner_modo(int modo) {
  registros.h.al = modo;
  registros.h.ah = 0x00;
  int86(0x10,&registros,&registros);
  return;              }

/* FUNCION QUE PONE LA PALETA DE COLORES PARA UNA ESCALA DE GRISES
   EN EL INTERVALO [0,255], CON 64 TONALIDADES DE GRISES DIFERENTES */
void Poner_paleta()  {
  unsigned int indice, i;
  union REGS ent, sal;

  ent.h.al = 0x10;
  ent.h.ah = 0x10;
  for (indice = 0; indice < 64; indice++)
  for (i = 0; i < 4; i++)      {
     ent.x.bx = indice*4+i;
     ent.h.dh = indice;
     ent.h.cl = indice;
     ent.h.ch = indice;
     int86(0x10, &ent, &sal);  }
 return;   }

/* FUNCION QUE PERMITE HACER EL AJUSTE DE UNA IMAGEN CON UNA CURVA
   LOGARITMICA */
```

```
void Img_log(AP_IMG apimg) {
 int i,j;
 float F;

 F = (PALETA)/log10(Max);
 for (i=0;i<r; i++)
   for (j=0; j<c; j++)
        apimg[c*i+j] = (BYTE)(F * log10(1+abs(apimg[c*i+j])));
 Histog(imagen, H);   }

/* FUNCION QUE PERMITE AJUSTAR LAS TONALIDADES DE GRIS DE LA IMAGEN
  PARA ELLO UTILIZA UNA FUNCION LINEAL */
void Ajusta(AP_IMG apimg)  {
 int i, j;
 float ini, fin, parcial;
 char resp;

 printf("\nAJUSTAR CON LA ESCALA DEFAULT? S/N ");
 resp = getch();
 if (resp == 's')   {
 for (i=0;i<r; i++)
  for (j=0; j<c; j++)
    apimg[c*i+j] = (BYTE)((PALETA / (Max-Min)) *
                    apimg[c*i+j]+(PALETA*Min)/(Max-Min));  }
 else
   if (resp == 'n')  {
        do {
          printf("\nDAME EL PRIMER ELEMENTO DEL RANGO [0 - 255] ");
          scanf("%f",&ini);
        } while ((ini < 0) || (ini > 255));
        do {
          printf("\nDAME EL ULTIMO ELEMENTO DEL RANGO [0 - 255] ");
          scanf("%f",&fin);
        } while ((fin < 0) || (fin > 255) || (fin <= ini));
        for (i=0;i<r; i++)
          for (j=0; j<c; j++) {
            parcial = ((fin-ini)/(float)(Max-Min) * (float)apimg[c*i+j])
                    +((ini*(float)Max-fin*(float)Min)/(float)(Max-Min));
            apimg[c*i+j] = (BYTE)(int)((parcial*10+5)/10);    }    }
 Histog(imagen, H);   }

/* FUNCION QUE OBTIENE EL NEGATIVO DE LA IMAGEN */
void Negativo(AP_IMG apimg)  {
 int i, j;

 for (i=0; i<r; i++)
   for (j=0; j<c; j++)
        apimg[c*i+j] = PALETA - apimg[c*i+j];
 Histog(imagen, H);    }

/* FUNCION QUE OBTIENE LOS PLANOS DE BITS QUE FORMAN UNA IMAGEN */
void Plano_bit(AP_IMG apimg)  {
 int i, j, k, m, numero, division, bit[8];
```

```
 do {
   printf("QUE PLANO ENTRE LOS OCHO POSIBLES (0 - 7) DESEAS VER ");
   scanf("%d", &numero);
 } while (numero < 0 || numero > 7);
 for (i=0; i<r; i++)
   for (j=0; j<c; j++)  {
            division = apimg[c*i+j];
            for (m=0; m<8; m++)     /* Limpio el arreglo bit */
              bit[m] = 0;
            k = 0;
            do {
              bit[k] = division%2;       /* Tengo el residuo */
              division = division/2;     /* Tengo la division */
              k++;    /* Tengo el numero de bits que ocupa el numero */
            } while (division != 0);
            if (bit[numero])   /* Obtenemos el elemento deseado */
              apimg[c*i+j] = 255;
            else
              apimg[c*i+j] = 0;  }
 Histog(imagen, H);
 return;  }

/* FUNCION QUE DESPLIEGA LA IMAGEN EN PANTALLA */
Poner_imagen(AP_IMG apimg)  {
 int i,j;

 Poner_modo(IMAGEN);
 Poner_paleta();
 memvideo = (unsigned char *)MK_FP(0xA000,0x0000);
  if (memvideo == NULL)
    { printf("NO HAY ESPACIO\n");
            exit(1);    }
  for (i=0;i<r; i++)
           for (j=0; j<c; j++) {
             if ((i < 200) && (j < 320))
               memvideo[320*i+j]=apimg[c*i+j];  }
  O=getch();  Poner_modo(TEXTO);
 return;  }

/* FUNCION QUE OBTIENE EL HISTOGRAMA DE LA IMAGEN */
void Histog(AP_IMG pimagen, Histograma histogra)  {
 unsigned long x, y;
 unsigned indice;

 Max = 0;
 Min = 0;
 for (x = 0; x <= 255; x++)
   histogra[x] = 0.;
 for (x = 0; x < r * c; x++)  {
   indice = ((unsigned unsigned char huge *)pimagen)[x];
   histogra[indice]++;
   if(indice>Max)
```

```
            Max = indice;
    if(indice<Min)
            Min = indice;       }
 return; }

/* FUNCION QUE DIBUJA UN PUNTO EN LA PANTALLA */
void Punto(unsigned x, unsigned y, unsigned char color) {
 char far *scr = (char far *)MK_FP(0xA000,0x0000);

 scr[y * 320 + x] = color;  }

/* FUNCION QUE GRAFICA EL HISTOGRAMA DE LA IMAGEN */
void Grafica(Histograma histogram)  {
 unsigned long x, y, maximo;

 maximo = Maximo(histogram);
 Poner_modo(IMAGEN);
 Poner_paleta();
 Lineas(32,190,288,190,255);
 for (x = 0; x <= 63; x++)
    for (y = 0; y <= 3; y++)  {
            Punto(32+x*4+y, 193, x*4+y);
            Punto(32+x*4+y, 194, x*4+y);
            Punto(32+x*4+y, 195, x*4+y);
            Punto(32+x*4+y, 196, x*4+y);   }
 for (x = 0; x <= 255; x++)
    Altura(32 + x, floor((histogram[x] * 180)/maximo), 200);
 for( ; !getch(); ) ;
 return;  }

/* FUNCION QUE DIBUJA LINEAS EN LA PANTALLA */
void Lineas(unsigned int A, unsigned int B, unsigned int C,
            unsigned int D, int T)  {
 unsigned long x;

 if (A == C)
    for (x = 0; x <= abs(D - B); x++)
            Punto(A, D - x, T);
 else
    for (x = 0; x <= abs(A - C); x++)
            Punto(A + x, B, T);   }

/* FUNCION QUE DETERMINA LA ALTURA DE LAS LINEAS DEL HISTOGRAMA */
void Altura(unsigned X, unsigned Y, int C)  {
 unsigned x;

 for (x = 0; x <= Y; x++)
    Punto(X, 190 - x, C);
 return;  }

/* FUNCION QUE DETERMINA EL VALOR MAXIMO DENTRO DEL HISTOGRAMA */
unsigned long Maximo(Histograma histogram)  {
 unsigned x;
```

```
 unsigned long aux;

 aux = histogram[0];
 for (x = 0; x <= 255; x++)
    if (histogram[x] >= aux)
          aux = histogram[x];
 return aux;  }

/* FUNCION QUE DETERMINA LA DISTANCIA ENTRE X Y Y */
unsigned long Dist(unsigned long x, unsigned long y)  {
 if ((x - y) <= 0)
    return (y - x);
 else
    return (x - y);  }

/* FUNCION QUE REALIZA EL MAPEO DE m DE LA IMAGEN CUANDO SE HAN OPERADO LOS
   HISTOGRAMAS (EL DE LA IMAGEN Y EL DEL HISTOGRAMA DESEADO). */
void Mapeo(imagen, m, X, Y)
unsigned unsigned char huge *imagen;
Histograma m;
unsigned long X;
unsigned long Y;  {
 unsigned long i, j;

 for (i = 0; i <= Y - 1; i++)
    for (j = 0; j <= X - 1; j++)
          imagen[i * X + j] = m[imagen[i * X + j]];
 return;   }

/* FUNCION QUE GENERA EL HISTOGRAMA ACUMULADO DE H. */
void Acumula(Histograma H)  {
 unsigned i;

 for (i = 0; i <= 254; i++)
    H[i + 1] += H[i];
 return; }

/* FUNCION QUE REALZA LA IMAGEN */
void Realce(Histograma H, Histograma G, Histograma m)  {
 unsigned long i, j = 0;

 for (i = 0; i <= 255; i++) {
    while (H[i] > G[j]) j++;
    if (Dist(H[i],G[j]) <= (Dist(H[i],G[j-1])))
          m[i] = j;
    else
          m[i] = j - 1;
    if ( j == 0 ) m[i] = 0;   }
 return;  }

/* FUNCION QUE SALVA LA IMAGEN EN UN ARCHIVO. LAS IMAGENES RESULTANTES NO
   TENDRAN FORMATO ALGUNO, CONSISTIRAN UNICAMENTE DEL MAPA DE BITS */
void Salvar_imagen(AP_IMG imagen)  {
```

```
FILE *genarchivo;
FILE *archi;
char nuevoarch[40];
char a;
int i,j;

printf("\nESCRIBE EL NOMBRE DEL NUEVO ARCHIVO: ");
do {
  scanf("%s",nuevoarch);
  genarchivo = fopen(nuevoarch,"rb");
  if (genarchivo == NULL) {
        printf("\nGUARDANDO EL ARCHIVO DE LA IMAGEN %s",nuevoarch);
        archi = fopen(nuevoarch,"wb");
        for(i=0; i<r ; i++)
          for(j=0; j<c ; j++)
            fprintf(archi,"%c",imagen[c*i+j]);
        fclose(archi);
        return;     }
  else {
        printf("\nEXISTE UN ARCHIVO CON EL MISMO NOMBRE ");
        printf("\nDESEAS SOBRE ESCRIBIRLO S/N ");
        a=getche();
        a=toupper(a);
        if(a =='N') {
          fclose(genarchivo);
          printf("ESCRIBE OTRO NOMBRE DIFERENTE \n");          }
        else      {
          archi = fopen(nuevoarch,"wb");
          for(i=0; i<r ; i++)
            for(j=0; j<c ; j++)
                  fprintf(archi,"%c",imagen[c*i+j]);
          fclose(archi);
          return; }              }
} while(a =='N');
return; }

/* FUNCION DEL MENU PRINCIPAL */
char *Menu(char *seleccion) {
 char opcion;

 clrscr();
 do {
  printf("\n\n  REALCE DE IMAGENES APLICANDO OPERACIONES SOBRE PUNTOS\n");
  printf("\n   OPCIONES \n");
  printf("\n   A : SALIR DE REALCE APLICANDO OPERACIONES SOBRE PUNTOS");
  printf("\n   B : LEER UNA IMAGEN DE UN ARCHIVO");
  printf("\n   C : MOSTRAR LA IMAGEN EN PANTALLA");
  printf("\n   D : GUARDAR LA IMAGEN EN UN ARCHIVO (SIN FORMATO)");
  printf("\n   E : MOSTRAR GRAFICAMENTE EL HISTOGRAMA");
  printf("\n   F : AJUSTE DE LA ESCALA DE GRISES CON FUNCION LOGARITMICA");
  printf("\n   G : AJUSTE DE LA ESCALA DE GRISES CON FUNCION LINEAL");
  printf("\n   H : OBTENER EL NEGATIVO DE LA IMAGEN");
  printf("\n   I : OBTENER LOS PLANOS DE BITS DE LA IMAGEN");
```

```
  printf("\n  J : ECUALIZACION DEL HISTOGRAMA");
  printf("\n  K : ESPECIFICACION DEL HISTOGRAMA");
  printf("\n\n ELIGE UNA OPCION: ");
  opcion=getche(); opcion = toupper(opcion);
 }while((opcion > 'K') || (opcion < 'A'));
 (*seleccion)=opcion;
 return(seleccion);  }

/* PRINCIPAL */
void main()  {
 int result;
 char *opcion;
 Histograma G, m;
 unsigned media, var, i;
 double gauss;

 clrscr();
 Leer_imagen();  Histog(imagen, H);
 do {
  opcion = NULL;    opcion=Menu(opcion);
  switch(*opcion)  {
   case 'B': farfree((void *) imagen);
                    Leer_imagen();          Histog(imagen, H);    break;
   case 'C': Poner_imagen(imagen);                                 break;
   case 'D': Salvar_imagen(imagen);                                break;
   case 'E': Grafica(H);  Poner_modo(TEXTO);                       break;
   case 'F': Img_log(imagen);  Poner_imagen(imagen); break;
   case 'G': Ajusta(imagen);        Poner_imagen(imagen); break;
   case 'H': Negativo(imagen); Poner_imagen(imagen); break;
   case 'I': Plano_bit(imagen);  Poner_imagen(imagen); break;
   case 'J':
            for (i = 0; i <= 255; i++)  /* Histograma uniforme */
              G[i] = (unsigned long)(c * r / 256.);
            Acumula(H); Acumula(G);  Realce(H, G, m);
            Mapeo(imagen, m, c, r);  Poner_imagen(imagen);
            Histog(imagen, H);
            break;
   case 'K':
            do {
              printf("\nCUAL ES LA MEDIA. ESTE VALOR DETERMINA EL BRILLO");
              printf("\nQUE TENDRA LA IMAGEN [DEBE DE SER ENTRE 0 Y 255] ");
              scanf("%u",&media);
            } while ((media < 0) && (media > 255));
            do {
              printf("\nCUAL ES LA VARIANZA. ESTE VALOR DETERMINA EL");
              printf("\nCONTRASTE DE LA IMAGEN [DEBE SER ENTRE 0 y 50000] ");
              scanf("%u",&var);
            } while ((var < 0) && (var > 50000));
            for (i = 0; i <= 255; i++) {
              gauss = expl(-pow(((float)i-(float)media),2)/(2.*(float)var))
                        / (sqrt((float)var)*sqrt(2.*M_PI));
              G[i] = (unsigned long)(c*r*gauss);   }
            Acumula(G);
```

```
            for (i = 0; i < 256; i++)
              G[i] = (unsigned long)(G[i] * c * r / G[255]);
            Acumula(H);          Realce(H, G, m);  Mapeo(imagen, m, c, r);
            Poner_imagen(imagen);          Histog(imagen, H);
            break;   }
 } while (*opcion != 'A');
 farfree((void *) imagen);  }
```

El siguiente es el programa que abarca las técnicas expuestas en el capítulo 4 realce aplicando operaciones espaciales. Las cabeceras, la declaración de variables globales y las funciones Leer_imagen, Poner_modo, Poner_paleta, Poner_imagen, Salvar_imagen fueron omitidas en el siguiente programa para evitar redundancias ya que son idénticas al listado anterior.

```
/* FUNCION QUE CREA LA MASCARA PARA EL FILTRADO PASO BAJAS */
float *Mask_Paso_bajas(int n, float *w)  {
 int i;

 for (i = 0; i < n*n; i++)
    w[i] = 1;
 return w;  }

/* FUNCION QUE CREA LA MASCARA PARA EL FILTRADO HIGH BOOST */
float *Mask_resta(int n, float *w)  {
 int i;
 float A;

 do {
  printf("\nDAME EL VALOR DE A (A >= 1 Y PUEDE TENER PARTE FRACCIONARIA) ");
  scanf("%f", &A);
 } while(A < 1);
 for (i = 0; i < n*n; i++)    {
   if (i != n*n/2)  /* Creo la mascara para el filtro */
            w[i] = -1;          /* pasobajas */
   else
            w[i] = 9*A-1;   }
 return w;  }

/* FUNCION QUE REALIZA LA CONVOLUCION DE z Y w */
long int Convolucion(int n, int *z, float *w)  {
 long int R;
 int i;
 float suma;

 suma = 0;
 for (i = 0; i < n * n; i++)
    suma += (float)z[i] * w[i];
 R = (long int)suma;
 return R;  }

/*FUNCION QUE DEVUELVE LA MEDIANA, EL VALOR MINIMO O MAXIMO SEGUN EL CASO*/
BYTE Mediana(unsigned M, int n, BYTE *z)  {
 int i,j;
 BYTE aux;
```

```
 for(i = 0; i < n*n; i++)
   for(j = 0; j < n*n-i; j++)   {
          if (z[j] > z[j+1])   {
            aux = z[j+1];
            z[j+1] = z[j];
            z[j] = aux;  }   }
 if (M==1)
   return z[0];
 else
   if (M==2)
          return z[n*n/2];
   else
          if (M==3)
            return z[n*n-1];
          else  {
            printf("\nTIPO DE FILTRO INVALIDO");
            exit(1);  }  }

/* FUNCION QUE GENERA LA MASCARA PARA EL FILTRADO PASO ALTAS */
float *Mask_Paso_altas(int n, float *w)  {
 int i;

 for (i = 0; i < n * n; i++) {  /* Creo la mascara para el filtro */
   if (i != (n * n / 2))       /* paso altas */
          w[i] = -1;
   else
          w[i] = n * n - 1;    }
 return w;  }

/* FUNCION QUE GENERA LA MASCARA NECESARIA PARA LA INTERPOLACION */
float *Mask_Interpola(float *w)  {
 w[0] = 0.25;  w[1] = 0.5;  w[2] = 0.25;
 w[3] = 0.5;   w[4] = 1;    w[5] = 0.5;
 w[6] = 0.25;  w[7] = 0.5;  w[8] = 0.25;
 return w;  }

/* FUNCIONES QUE GENERAN LOS OPERADORES DE ROBERTS */
int *Mask_Roberts1(int *w1)  {
 w1[0] = 1;  w1[1] = 0;
 w1[2] = 0;  w1[3] = -1;
 return w1;  }

int *Mask_Roberts2(int *w2)  {
 w2[0] = 0;   w2[1] = 1;
 w2[2] = -1;  w2[3] = 0;
 return w2;  }

/* FUNCIONES QUE GENERAN LOS OPERADORES DE PREWITT */
int *Mask_Prewitt1(int *w1)  {
 w1[0] = -1;  w1[1] = -1;  w1[2] = -1;
 w1[3] = 0;   w1[4] = 0;   w1[5] = 0;
 w1[6] = 1;   w1[7] = 1;   w1[8] = 1;
```

```
 return w1;  }

int *Mask_Prewitt2(int *w2)  {
 w2[0] = -1;  w2[1] = 0;  w2[2] = 1;
 w2[3] = -1;  w2[4] = 0;  w2[5] = 1;
 w2[6] = -1;  w2[7] = 0;  w2[8] = 1;
 return w2;  }

/* FUNCIONES QUE GENERAN LOS OPERADORES DE SOBEL */
int *Mask_Sobel1(int *w1)  {
 w1[0] = -1;  w1[1] = -2;  w1[2] = -1;
 w1[3] = 0;  w1[4] = 0;  w1[5] = 0;
 w1[6] = 1;  w1[7] = 2;  w1[8] = 1;
 return w1;  }

int *Mask_Sobel2(int *w2)  {
 w2[0] = -1;  w2[1] = 0;  w2[2] = 1;
 w2[3] = -2;  w2[4] = 0;  w2[5] = 2;
 w2[6] = -1;  w2[7] = 0;  w2[8] = 1;
 return w2;  }

/* FUNCION QUE REALIZA LA CONVOLUCION UNICAMENTE PARA LOS OPERADORES
  DE DETECCION DE BORDES */
int Convolucion_D(int n, int *z, int *w)  {
 int R;
 int i;
 long int suma;

 suma = 0;
 for (i = 0; i < n*n; i++)
    suma += z[i] * w[i];
 R = (int)abs(suma);
 return R;  }

/* FUNCION QUE OBTIENE LA MEDIA DE z */
int Media(int n, int *z)  {
 unsigned suma;
 int i, R;

 suma = 0;
 for (i = 0; i < n*n; i++)
    suma += z[i];
 R = suma / (n * n);
 return R;  }

/* FUNCION QUE OBTIENE LA DESVIACION ESTANDAR DE z */
float D_Standar(int n, int m, int *z)  {
 unsigned suma;
 int i;
 float R;

 suma = 0;
 for (i = 0; i < n*n; i++)
```

```
    suma += (z[i] - m) * (z[i] - m);
  R = sqrt(suma / (n*n));
  if (!R)
    R = 0.1;
  return R;   }

/* FUNCION QUE REALIZA EL FILTRADO PASO BAJAS */
void Paso_bajas(AP_IMG img)   {
  int i, j, k, m, n, cm, rm;
  int *z;
  float *w;
  FILE *fp;

  printf("\n\nQUE MAGNITUD TENDRA LA MASCARA DE FILTRADO \n3) 3 X 3");
  printf("\n5) 5 X 5 \n7) 7 X 7 \n15) 15 X 15 \n25) 25 X 25 \n? ");
  scanf("%d", &n);
  if (n == 3 || n == 5 || n == 7 || n == 15 || n == 25)   {
    w = malloc(n * n * sizeof(float)); /* Creo el espacio para la matriz */
    if (!w)   {              /* que contendra la mascara */
          printf("\nNO TENGO SUFICIENTE MEMORIA\n");
          exit(1); }
    w = Mask_Paso_bajas(n, w);
    z = malloc(n * n * sizeof(int)); /* Creo el espacio para la matriz */
    if (!z)  {        /* de los valores de la imagen a ser filtrados */
          printf("\nNO TENGO SUFICIENTE MEMORIA\n");
          exit(1);  }
    if (!(fp = fopen("pasobaja.bin","wb"))) /* Abro un archivo para */
          {           /* escribir la imagen generada por el filtro */
          printf("\NO PUDE ABRIR EL ARCHIVO PARA LA IMAGEN FILTRADA");
          getch(); return;   }
    for (i = 0; i < r; i++) /* Guarda el primer renglon de la imagen */
          fputc(img[i],fp);
    for (i = 1; i < r-1; i++)      /* Recorro los pixeles de la */
          for (j = 0; j < c; j++)  {  /* imagen original */
            if ((j != 0) && (j != (c-1)))   {
                  rm = i - (n / 2);  /* Calculo el 1 renglon de vecindad */
                  cm = j - (n / 2);  /* Calculo la 1 columna de vecindad */
                  for (k = 0; k < n; k++)      /* Creo la matriz de */
                    for (m = 0; m < n; m++)   /* la vecindad */
                          z[n*k+m] = img[c*(rm+k) + (cm+m)];
                  fputc((BYTE)(Convolucion(n, z, w)/(n*n)),fp);   }
            else
                  fputc(img[c*i+j],fp); } /* Escribo la primer y ultima c */
          for (i = 0; i < r; i++)
            fputc(img[c*(r-1)+j],fp); /* Escribo el ultimo renglon */
          fclose(fp);          free(z);     free(w);
          printf("\nLA IMAGEN FILTRADA SE GUARDO EN PASOBAJA.BIN ");
          printf("\nCON LAS MISMAS DIMENSIONES");
          getch();  }
  return;   }

/* FUNCION QUE REALIZA EL FILTRADO DE MEDIANA, DE MINIMA Y MAXIMA */
void Filtro_Mediana(AP_IMG img)   {
```

```
unsigned M,
int i, j, k, m, n, cm, rm;
BYTE *z;
FILE *fp;

printf("\n\nQUE MAGNITUD TENDRA LA MASCARA DE FILTRADO\n3) 3 X 3");
printf("\n5) 5 X 5 \n7) 7 X 7 \n15) 15 X 15 \n25) 25 X 25 \n? ");
scanf("%d", &n);
printf("\nQUE TIPO DE FILTRO APLICO");
do {
  printf("\n1) MINIMA \n2) MEDIANA \n3) MAXIMA \n? ");
  scanf("%d", &M);
}while((M < 1) && (M > 3));
if (n == 3 || n == 5 || n == 7 || n == 15 || n == 25)   {
  z = malloc(n*n * sizeof(BYTE)); /* Creo el espacio para la matriz */
  if(!z)  {         /* de los valores de la imagen a ser filtrados */
          printf("\nNO TENGO SUFICIENTE MEMORIA");
          exit(1);  }
  if (!(fp = fopen("MEDIANA.bin","wb")))  /* Abro un archivo para */
          {               /* escribir la imagen generada por el filtro */
          printf("\NO PUDE ABRIR EL ARCHIVO PARA LA IMAGEN FILTRADA");
          getch(); return;  }
  for (i = 0; i < r; i++) /* Guarda el primer renglon de la imagen */
          fputc(img[i],fp);
  for (i = 1; i < r-1; i++)       /* Recorro los pixeles de la */
          for (j = 0; j < c; j++)  {   /* imagen original */
            if ((j != 0) && (j != (c-1)))    {
                    rm = i - (n / 2);  /* Calculo el 1 renglon de vecindad */
                    cm = j - (n / 2);       /* Calculo la 1 columna de vecindad */
                    for (k = 0; k < n; k++)       /* Creo la matriz de */
                      for (m = 0; m < n; m++)   /* la vecindad */
                              z[n*k+m] = img[c*(rm+k) + (cm+m)];
                    fputc((BYTE)Mediana(M, n, z),fp);    }
            else
              fputc(img[c*i+j],fp);  }  /* Escribo la primer y ultima c */
  for (i = 0; i < r; i++)
          fputc(img[c*(r-1)+j],fp); /* Escribo el ultimo renglon */
  fclose(fp);    free(z);
  printf("\nLA IMAGEN FILTRADA SE GUARDO EN MEDIANA.BIN ");
  printf("\nCON LAS MISMAS DIMENSIONES");
  getch();    }
return;   }

/* FUNCION PARA EL AGUDIZAMIENTO DE LOS BORDES UTILIZANDO FILTROS
  PASO ALTAS Y HIGH BOOST */
void Sharpening(AP_IMG img)  {
  int i, j, k, m, n, cm, rm, R, mayor, menor, tipo;
  int *z;
  float *w;
  BYTE A;
  FILE *fp;

  do {
```

```
  printf("\nQUE TIPO DE FILTRO QUIERES APLICAR\n");
  printf("\n1) PASO ALTAS\n2) HIGH BOOST\n");
  scanf("%d",&tipo);
} while((tipo < 1) || (tipo > 2));
mayor = -255; /* Inicializo las variables que contendran los valores */
menor = 255; /* mayor y menor del resultado de la convolución */
n = 3;
w = malloc(n*n * sizeof(float)); /* Creo el espacio para la matriz */
if (!w)  {                       /* que contendra la mascara */
  printf("\nNO TENGO SUFICIENTE MEMORIA");
  exit(1); }
if (tipo == 1)
  w = Mask_Paso_altas(n, w);
else
  if (tipo == 2)
        w = Mask_resta(n, w);
  else return;
z = malloc(n*n * sizeof(int)); /* Creo el espacio para la matriz */
if (!z)  {          /* de los valores de la imagen a ser filtrados */
  printf("\nNO TENGO SUFICIENTE MEMORIA");
  exit(1); }
if (!(fp = fopen("PASOALTA.bin","wb"))) /* Abro un archivo para */
  {              /* escribir la imagen generada por el filtro */
  printf("\NO PUEDO ABRIR EL ARCHIVO PARA LA IMAGEN FILTRADA");
  getch(); return;  }
for (i = 0; i < r; i++) /* Guarda el primer renglon de la imagen */
  fputc(img[i],fp);
        /* Primero recorro la imagen con el filtro para obtener el valor
        mayor y menor para su futuro ajuste */
for (i = 1; i < r-1; i++)      /* Recorro los pixeles de la */
  for (j = 0; j < c; j++)  {  /* imagen original */
        rm = i - (n / 2);  /* Calculo el 1 renglon de vecindad */
        cm = j - (n / 2);      /* Calculo la 1 columna de vecindad */
        for (k = 0; k < n; k++)       /* Creo la matriz de */
          for (m = 0; m < n; m++)   /* la vecindad */
                z[n*k+m] = img[c*(rm+k) + (cm+m)];
        R = Convolucion(n, z, w)/(n*n); /* Convoluciono las mascaras */
        if (R > mayor)  /* Extraigo el valor mayor de la */
          mayor = R;  /* convolución */
        if (R < menor)  /* Extraigo el valor mayor de la */
          menor = R;   }  /* convolución */
/* Teniendo los valores indicados de mayor y menor ajustamos */
for (i = 1; i < r-1; i++)      /* Recorro los pixeles de la */
  for (j = 0; j < c; j++)  {  /* imagen original */
        if ((j != 0) && (j != (c-1))) { /* Elimino el primer y ultimo r */
          rm = i - (n / 2);  /* Calculo el 1 renglon de vecindad */
          cm = j - (n / 2);  /* Calculo la 1 columna de vecindad */
          for (k = 0; k < n; k++)       /* Creo la matriz de */
                for (m = 0; m < n; m++)   /* la vecindad */
                  z[n*k+m] = img[c*(rm+k) + (cm+m)];
          R = Convolucion(n, z, w)/(n*n);
          A = (BYTE)(255.0/(float)(mayor-menor)*(float)R
                + (-255.0*(float)menor/(mayor-menor))); /* Ajuste */
```

```
            fputc(A,fp);  }   /* Guardo resultado en el archivo */
          else
            fputc(img[c*i+j],fp);  } /* Escribo la primer y ultima c */
 for (i = 0; i < r; i++)
   fputc(img[c*(r-1)+j],fp); /* Escribo el ultimo renglon */
 fclose(fp);  free(z);  free(w);
  printf("\nLA IMAGEN FILTRADA SE GUARDO EN PASOALTA.BIN ");
  printf("\nCON LAS MISMAS DIMENSIONES");
 getch();  return;  }

/* FUNCION QUE AMPLIA UNA IMAGEN POR DUPLICACION DE PIXELES */
void Amplia(AP_IMG pimg)  {
 int i, j, ar, ac, dim;
 int xr, yc, xtr, ytc, razon;
 FILE *fp;

 printf("\nEN QUE RENGLON EMPIEZO A AMPLIAR LA IMAGEN ");
 scanf("%d",&xr);       /* renglon de inicio de la ampliacion */
 printf("\nEN QUE COLUMNA EMPIEZO DE AMPLIAR LA IMAGEN ");
 scanf("%d",&yc);       /* columna de inicio de la ampliacion */
 printf("\nQUE MAGNITUD VAMOS A AMPLIAR [MENOR A 128] ");
 scanf("%d",&dim);
 xtr = xr+dim;
 ytc = yc+dim;
 if ((xtr > r) || (ytc > c) || (xr < 0) || (yc < 0))  /* Verifico que no */
   {     /* exceda las dimensiones de la imagen que esta en memoria */
  printf("\n EXCEDES LAS DIMENSIONES DE LA IMAGEN ");  getch();
  return;  }
 razon = 2;
 if (!(fp = fopen("imgamp.bin","wb")))  /* Abro un archivo para */
   {              /* escribir la imagen ampliada */
  printf("\NO PUDE ABRIR EL ARCHIVO PARA LA IMAGEN AMPLIADA");
  getch();  return;   }
 for (i = xr; i < xtr; i++)         /* Realizo la ampliacion */
   for (ar = 0; ar < razon; ar++)
          for (j = yc; j < ytc; j++)
            for (ac = 0; ac < razon; ac++)
                    fputc(pimg[c*i+j],fp);
 fclose(fp);
 printf("\nLA IMAGEN AMPLIADA SE GUARDO EN IMGAMP.BIN");
 printf("\nLA CUAL ES DE DIMENSIONES CUADRADAS");
 getch();  return;  }

/* FUNCION QUE AMPLIA UNA IMAGEN POR INTERPOLACION LINEAL */
void Amplia_II(AP_IMG pimg)  {
 int i, j, ar, ac, dim;
 int xr, yc, xtr, ytc, razon;
 FILE *fp;

 printf("\nEN QUE RENGLON EMPIEZO A AMPLIAR LA IMAGEN ");
 scanf("%d",&xr);       /* renglon de inicio de la ampliacion */
 printf("\nEN QUE COLUMNA EMPIEZO DE AMPLIAR LA IMAGEN ");
 scanf("%d",&yc);       /* columna de inicio de la ampliacion */
```

```
printf("\nQUE MAGNITUD VAMOS A AMPLIAR [MENOR A 128] ");
scanf("%d",&dim);
xtr = xr+dim;
ytc = yc+dim;
if ((xtr > r) || (ytc > c) || (xr < 0) || (yc < 0)) /* Verifico que no */
  { /* exceda las dimensiones de la imagen que esta en memoria */
  printf("\n EXCEDES LAS DIMENSIONES DE LA IMAGEN");
  getch();  return; }
razon = 2;
if (!(fp = fopen("imgtmp.bin","wb"))) /* Abro un archivo para */
  {          /* escribir la imagen ampliada */
  printf("\nNO PUDE ABRIR EL ARCHIVO PARA LA IMAGEN AMPLIADA");
  getch(); return; }
for (i = xr; i < xtr; i++)        /* Realizo la ampliacion */
  for (ar = 0; ar < razon; ar++)
        for (j = yc; j < ytc; j++)
          for (ac = 0; ac < razon; ac++)  {
                if (ar==0)  {
                  if (ac==0)
                        fputc(pimg[c*i+j],fp);
                  else
                        fputc(0,fp); }
                else
                  fputc(0,fp);    }
fclose(fp);
printf("POR FAVOR CARGUE EL ARCHIVO IMGTMP.BIN ");
printf("LA CUAL ES UNA IMAGEN CUADRADA");
return;  }

/* FUNCION DE INTERPOLACION LINEAL */
void Interpola(AP_IMG pimg)  {
 int i, j, k, m, cm, rm;
 int *z;
 float *w;
 FILE *fp;

 w = malloc(9 * sizeof(float));  /* Creo el espacio para la matriz */
 if (!w)  {              /* que contendra la mascara */
  printf("\nNO TENGO SUFICIENTE MEMORIA");
  exit(1);  }
 w = Mask_Interpola(w);
 z = malloc(9 * sizeof(int));  /* Creo el espacio para la matriz */
 if (!z)  {       /* de los valores de la imagen a ser filtrados */
  printf("\nNO TENGO SUFICIENTE MEMORIA");
  exit(1);  }
 if (!(fp = fopen("imgaint.bin","wb")))  /* Abro un archivo para */
  {          /* escribir la imagen generada por el filtro */
  printf("\NO PUDE ABRIR EL ARCHIVO PARA LA IMAGEN INTERPOLADA");
  return;   }
 for (i = 0; i < r; i++) /* Guarda el primer renglon de la imagen */
   fputc(pimg[i],fp);
 for (i = 1; i < r-1; i++)      /* Recorro los pixeles de la */
   for (j = 0; j < c; j++)  {  /* imagen original */
```

```
            if ((j != 0) && (j != (c-1)))    {
              rm = i - (3 / 2);  /* Calculo el 1 renglon de vecindad */
              cm = j - (3 / 2);       /* Calculo la 1 columna de vecindad */
              for (k = 0; k < 3; k++)       /* Creo la matriz de */
                      for (m = 0; m < 3; m++)    /* la vecindad */
                        z[3*k+m] = pimg[c*(rm+k) + (cm+m)];
              fputc((BYTE)Convolucion(3, z, w),fp);  }
            else
              fputc(pimg[c*i+j],fp);  }  /* Escribo la primer y ultima c */
  for (i = 0; i < r; i++)
    fputc(pimg[c*(r-1)+j],fp); /* Escribo el ultimo renglon */
  fclose(fp);  free(z);  free(w);
  printf("\nLA IMAGEN AMPLIADA E INTERPOLADA ESTA GUARDADA EN IMGAINT.BIN");
  printf("\nLA CUAL ES UNA IMAGEN CUADRADA");
  getch();  return;  }

/* FILTROS DERIVATIVOS */
void Derivativos(AP_IMG img)  {
  int i, j, k, m, rm, cm, n, tipo;
  int *z, *w1, *w2;
  FILE *fp;
  BYTE S;

  do {
    printf("\nQUE TIPO DE OPERADORES APLICO PARA OBTENER LOS BORDES\n");
    printf("\n1) OPERADORES DE ROBERTS\n2) OPERADORES DE PREWITT");
    printf("\n3) OPERADORES DE SOBEL\nELIGUE UN TIPO ");
    scanf("%d",&tipo);
  } while ((tipo < 1) || (tipo > 3));
  if (tipo == 1)
    n = 2;
  else
    n = 3;
  w1 = malloc(n * n * sizeof(int));  /* Creo el espacio para la matriz */
  if (!w1)  {                  /* que contendra la mascara */
    printf("\nNO TENGO SUFICIENTE MEMORIA");
    exit(1);  }
  w2 = malloc(n * n * sizeof(int));  /* Creo el espacio para la matriz */
  if (!w2)  {                  /* que contendra la mascara */
    printf("\nNO TENGO SUFICIENTE MEMORIA");
    exit(1);  }
  switch(tipo)  {
            case 1: w1 = Mask_Roberts1(w1);          w2 = Mask_Roberts2(w2);       break;
            case 2: w1 = Mask_Prewitt1(w1);          w2 = Mask_Prewitt2(w2);       break;
            case 3: w1 = Mask_Sobel1(w1); w2 = Mask_Sobel2(w2);          break;  }
  z = malloc(9 * sizeof(int));  /* Creo el espacio para la matriz */
  if (!z)  {      /* de los valores de la imagen a ser filtrados */
    printf("\nNO TENGO SUFICIENTE MEMORIA");
    exit(1);  }
  if (!(fp = fopen("deriva.bin","wb")))  /* Abro un archivo para */
    {              /* escribir la imagen generada por el filtro */
    printf("\NO PUDE ABRIR EL ARCHIVO PARA LA IMAGEN FILTRADA");
    return;  }
```

```
 for (i = 0; i < r; i++) /* Guarda el primer renglon de la imagen */
   fputc(img[i],fp);
 for (i = 1; i < r-1; i++)       /* Recorro los pixeles de la */
   for (j = 0; j < c; j++)  {   /* imagen original */
          if ((j != 0) && (j != (c-1)))    {
            if (n == 2)  {
                  rm = i;
                  cm = j;         }
            else  {
                  rm = i - (n / 2);  /* Calculo el 1 renglon de vecindad */
                  cm = j - (n / 2);  } /* Calculo la 1 columna de vecindad */
            for (k = 0; k < n; k++)       /* Creo la matriz de */
                  for (m = 0; m < n; m++)    /* la vecindad */
                    z[n*k+m] = img[c*(rm+k) + (cm+m)];
            S = (BYTE)(Convolucion_D(n, z, w1) + Convolucion_D(n, z, w2));
            fputc(S,fp);  }
          else
            fputc(img[c*i+j],fp); } /* Escribo la primer y ultima columna */
 for (i = 0; i < r; i++)
   fputc(img[c*(r-1)+j],fp); /* Escribo el ultimo renglon */
 fclose(fp);  free(z);  free(w1);  free(w2);
 printf("\nLA IMAGEN FILTRADA ESTA GUARDADA EN DERIVA.BIN ");
 printf("\nTIENE LAS MISMAS DIMENSIONES QUE LA IMAGEN ORIGINAL");
 getch();  return;  }

void Estadistico(AP_IMG img)  {
 int i, j, k, m, n, cm, rm, M, tipo;
 int *z;
 FILE *fp;
 BYTE V;

 do {
   printf("\nQUE OPERACION ESTADISTICA DESEAS REALIZAR\n");
   printf("1) MAPEO POR RADIO CONTRASTE INVERSO\n");
   printf("2) AJUSTE ESTADISTICO\n");
   scanf("%d",&tipo);
 } while((tipo < 1) || (tipo > 2));
 printf("\n\nQUE MAGNITUD TENDRA LA MASCARA \n3) 3 X 3");
 printf("\n5) 5 X 5 \n7) 7 X 7 \n15) 15 X 15 \n25) 25 X 25 \n? ");
 scanf("%d", &n);
 if (n == 3 || n == 5 || n == 7 || n == 15 || n == 25)    {
   z = malloc(n*n * sizeof(int));  /* Creo el espacio para la matriz */
   if (!z)  {  /* de los valores de la imagen a ser filtrados */
          printf("\nNO TENGO SUFICIENTE MEMORIA");
          exit(1);  }
   if (!(fp = fopen("estadist.bin","wb")))  /* Abro un archivo para */
          {              /* escribir la imagen generada por el filtro */
          printf("\NO PUDE ABRIR EL ARCHIVO PARA LA IMAGEN PROCESADA");
          return;  }
   for (i = 0; i < r; i++) /* Guarda el primer renglon de la imagen */
          fputc(img[i],fp);
   for (i = 1; i < r-1; i++)       /* Recorro los pixeles de la */
          for (j = 0; j < c; j++)  {  /* imagen original */
```

```
            if ((j != 0) && (j != (c-1)))   {
                    rm = i - (n / 2);  /* Calculo el 1 renglon de vecindad */
                    cm = j - (n / 2);       /* Calculo la 1 columna de vecindad */
                    for (k = 0; k < n; k++)       /* Creo la matriz de */
                      for (m = 0; m < n; m++)    /* la vecindad */
                            z[n*k+m] = img[c*(rm+k) + (cm+m)];
                    M = Media(n,z);
                    if (tipo == 1)
                      V = (BYTE)(M / D_Standar(n, M, z)); /* Calculo el radio */
                    else
                      V = (BYTE)(img[c*i+j] / D_Standar(n, M, z));
                    fputc(V,fp);   }            /* Mapeo y lo guardo */
            else
                    fputc(img[c*i+j],fp); } /* Escribo la primer y ultima c */
    for (i = 0; i < r; i++)
            fputc(img[c*(r-1)+j],fp); /* Escribo el ultimo renglon */
    fclose(fp);    free(z);
    printf("\nLA IMAGEN FILTRADA FUE GUARDADA EN ESTADIST.BIN ");
    printf("\nTIENE LAS MISMAS DIMENSIONES QUE LA IMAGEN ORIGINAL");
    getch(); } return; }

char *Menu(char *seleccion) {
  char opcion;

  clrscr();
  do {
    printf("\n\n  REALCE DE IMAGENES APLICANDO OPERACIONES ESPACIALES\n");
    printf("\n   OPCIONES \n");
    printf("\n   A : SALIR DE REALCE APLICANDO OPERACIONES SOBRE PUNTOS");
    printf("\n   B : LEER UNA IMAGEN DE UN ARCHIVO");
    printf("\n   C : MOSTRAR LA IMAGEN EN PANTALLA");
    printf("\n   D : GUARDAR LA IMAGEN EN UN ARCHIVO (SIN FORMATO)");
    printf("\n   E : APLICA FILTRADO PASO BAJAS");
    printf("\n   F : APLICA FILTRADO DE MEDIANA, MIN O MAX");
    printf("\n   G : REALCE DE LOS BORDES");
    printf("\n   H : AMPLIA EL SEGMENTO DE UNA IMAGEN");
    printf("\n   I : AMPLIA E INTERPOLA EL SEGMENTO DE IMAGEN");
    printf("\n   J : OBTEN LOS BORDES DE LA IMAGEN");
    printf("\n   K : ANALISIS ESTADISTICO");
    printf("\n\n ELEGIR UNA REALIZACION: ");
    opcion=getche(); opcion = toupper(opcion);
  } while((opcion < 'A') || (opcion > 'K'));
  (*seleccion)=opcion;
  return(seleccion); }

void main() {
  int result;
  char *opcion;

  clrscr();
  Leer_imagen();
  do {
    opcion = NULL;
```

```
  opcion=Menu(opcion);
  switch(*opcion)  {
   case 'B': farfree((void *) imagen); Leer_imagen();  break;
   case 'C': Poner_imagen(imagen);          break;
   case 'D': Salvar_imagen(imagen);   break;
   case 'E': Paso_bajas(imagen);            break;
   case 'F': Filtro_Mediana(imagen);        break;
   case 'G': Sharpening(imagen);            break;
   case 'H': Amplia(imagen);     break;
   case 'I': Amplia_II(imagen);  farfree((void *) imagen);
                    Leer_imagen();          Interpola(imagen);  break;
   case 'J': Derivativos(imagen);           break;
   case 'K': Estadistico(imagen);           break;   }
 }while (*opcion != 'A');
 farfree((void *) imagen);  }
```

El listado que se muestra a continuación es una recopilación de cuatro programas, cada uno de ellos comprende un tipo de filtrado de los explicados en el capítulo 5 realce aplicando operaciones en el dominio de las frecuencias. La limitante que impidió que estos programas fueran una unidad fue el espacio requerido en memoria para su ejecución, por esta misma razón no se pueden procesar imágenes cuyo tamaño sea diferente de 64 X 64. Los cuatro programas son idénticos en cuanto a las funciones que obtienen la transformada y la transformada inversa de Fourier. El único cambio entre ellos es la función de filtrado correspondiente. Las cabeceras y las funciones Leer_imagen, Poner_imagen, Salvar_imagen son idénticas a las mostradas en el primer programa

```
typedef unsigned char BYTE;
typedef float Complejos[2];

Complejos Matriz[N][N];
unsigned char huge *Imagen;

/* FUNCION QUE REALIZA LA TRANSFORMADA RAPIDA RAPIDA DE FOURIER EN f */
void FFT(Complejos f[N+1], unsigned Ln)  {
  int n, k, nv2, nm1, i, j, l, le1, le, ip;
  float temp;
  Complejos T, U, W, Aux;

  n = (int)pow(2,Ln); nm1 = n-1; nv2 = n/2;
  j = 1;
  for (i = 1; i <= nm1; i++) {
    if (i < j) {
          Aux[0] = f[i][0]; Aux[1] = f[i][1];
          f[i][0] = f[j][0]; f[i][1] = f[j][1];
          f[j][0] = Aux[0]; f[j][1] = Aux[1];   }
    k = nv2;
    while (k < j)
          { j -= k; k /= 2; }
    j += k;   }  /* for */
  for (l = 1; l <= Ln; l++) {
    le = (int)pow(2,l);
    le1 = le/2;
    U[0] = 1.; U[1] = 0.;  /* Unidad */
    W[0] = (float)cos((double)PI / (double)le1);
```

```
    W[1] = (float)-sin((double)PI / (double)le1);
    for (j = 1; j <= le1; j++) {
            i = j;
            while ( i <= n ) {
                    ip = i + le1;
                    T[0] = f[ip][0] * U[0] - f[ip][1] * U[1];
                    T[1] = f[ip][0] * U[1] + f[ip][1] * U[0];
                    f[ip][0] = f[i][0] - T[0]; f[ip][1] = f[i][1] - T[1];
                    f[i][0] += T[0]; f[i][1] += T[1];
                    i +=le;   }
            temp = U[0];
            U[0] = W[0] * U[0] - W[1] * U[1];
            U[1] = W[1] * temp + W[0] * U[1]; }  /*  for  */ } /*  for  */
    for ( i = 1; i <= n; i++)
          {  f[i][0] /= (float)n; f[i][1] /= (float)n; }
  return;  }

/* FUNCION QUE REALIZA LA TRANSFORMADA RAPIDA DE FOURIER EN Matriz  */
void FFT2D(unsigned Ln)  {
  unsigned i, j;
  Complejos Aux[N + 1];

  for (i = 0; i <= N - 1; i++) {
    for (j = 0; j <= N - 1; j++){
            Aux[j+1][0] = Matriz[i][j][0];
            Aux[j+1][1] = Matriz[i][j][1];   }
    FFT(Aux,Ln);
    for (j = 0; j <= N - 1; j++) {
            Matriz[i][j][0] = N * Aux[j+1][0];
            Matriz[i][j][1] = N * Aux[j+1][1];  }  }
  for (i = 0; i <= N - 1; i++)  {
    for (j = 0; j <= N - 1; j++) {
            Aux[j+1][0] = Matriz[j][i][0];
            Aux[j+1][1] = Matriz[j][i][1];  }
    FFT(Aux,Ln);
    for (j = 0; j <= N - 1; j++) {
            Matriz[j][i][0] = Aux[j+1][0];
            Matriz[j][i][1] = Aux[j+1][1];   }   }
  return ;  }

/* FUNCION QUE REALIZA EL FILTRADO PASOBAJAS */
void Paso_bajas() {
  int i,j, fi, fj, D;
  int r;
  long float Pt, P;
  float H;

  do {
   printf("\nCUAL ES LA FRECUENCIA DE CORTE? [ENTRE 1 Y 63] ");
   scanf("%d",&r);
  } while((r < 1) || (r > 63));
  for (i = 0; i <= N/2; i++)
    for (j = 0; j < N/2-1; j++) {
```

```
            Pt += Matriz[i][j][0]*Matriz[i][j][0] +
                Matriz[i][j][1]*Matriz[i][j][1];
            fi = N/2 - i;
            fj = j - (N/2-1);
            D = (int)(sqrt(fi*fi + fj*fj));
            if (D > r) {
              Matriz[i][j][0] = 0.;
              Matriz[i][j][1] = 0.; }
            else
              P += Matriz[i][j][0]*Matriz[i][j][0] +
              Matriz[i][j][1]*Matriz[i][j][1];    }
for (i = N/2+1; i < N; i++)
  for (j = 0; j < N/2-1; j++) {
            Pt += Matriz[i][j][0]*Matriz[i][j][0] +
                Matriz[i][j][1]*Matriz[i][j][1];
            fi = N/2 - i;
            fj = j - (N/2-1);
            D = (int)(sqrt(fi*fi + fj*fj));
            if (D > r) {
              Matriz[i][j][0] = 0.;
              Matriz[i][j][1] = 0.; }
            else
            P += Matriz[i][j][0]*Matriz[i][j][0] +
                Matriz[i][j][1]*Matriz[i][j][1];    }
for (i = 0; i <= N/2; i++)
  for (j = N/2-1; j < N; j++) {
            Pt += Matriz[i][j][0]*Matriz[i][j][0] +
                Matriz[i][j][1]*Matriz[i][j][1];
            fi = N/2 - i;
            fj = j - (N/2-1);
            D = (int)(sqrt(fi*fi + fj*fj));
            if (D > r) {
              Matriz[i][j][0] = 0.;
              Matriz[i][j][1] = 0.;    }
            else
              P += Matriz[i][j][0]*Matriz[i][j][0] +
                    Matriz[i][j][1]*Matriz[i][j][1];    }
for (i = N/2+1; i < N; i++)
  for (j = N/2-1; j < N; j++) {
            Pt += Matriz[i][j][0]*Matriz[i][j][0] +
                Matriz[i][j][1]*Matriz[i][j][1];
            fi = N/2 - i;
            fj = j - (N/2-1);
            D = (int)(sqrt(fi*fi + fj*fj));
            if (D > r) {
              Matriz[i][j][0] = 0.;
              Matriz[i][j][1] = 0.;    }
            else
              P += Matriz[i][j][0]*Matriz[i][j][0] +
                    Matriz[i][j][1]*Matriz[i][j][1];    }
B = 100. * (P / Pt);
printf("EL PORCENTAJE DE LA POTENCIA FILTRADA ES %3.2f\n", B);
return;  }
```

```
/* PRINCIPAL */
void main() {
 Complejos M[N+1];
 Complejos Aux[N/2][N/2];
 unsigned i, j;
 unsigned char max;
 char resp;

 clrscr(); Leer_imagen(); Poner_modo(0x13); Poner_paleta();
 Poner_imagen(Imagen); Poner_modo(0x03);
 for (i = 0; i <= N-1; i++)  /*  Asignemos a la matriz de trabajo la */
   for (j = 0; j <= N-1; j++) {  /*  imagen.  */
          Matriz[i][j][0] = (float)*(Imagen + i * N + j)*(pow(-1,i+j));
          Matriz[i][j][1] = 0.;    }
 FFT2D(6);  /*  Realizamos la transformada de Matriz en 2D  */
 for (i = 0; i <= N-1; i++)  /*  Asignamos espectro de potencias  */
   for (j = 0; j <= N-1; j++)
         *(Imagen + i * N + j) = (unsigned char)sqrt(Matriz[i][j][0]*
                                 Matriz[i][j][0] + Matriz[i][j][1]*Matriz[i][j][1]);
 printf("\nVEMOS EL ESPECTRO DE POTENCIAS DE LA IMAGEN EN EL DOMINO");
 printf("\nDE LAS FRECUENCIAS ");
 getch();  Poner_modo(0x13);  Poner_paleta();
 Poner_imagen(Imagen);   Poner_modo(0x03);
 Paso_bajas();
 for (i = 0; i < N/2; i++)  /*  Esquina superior izquierda en Aux */
   for (j = 0; j < N/2; j++) {  /*  imagen.  */
          Aux[i][j][0] = Matriz[i][j][0];
          Aux[i][j][1] = - Matriz[i][j][1];  }
 for (i = N/2; i < N; i++)  /*  Esquina inferior derecha en superior i */
   for (j = N/2; j < N; j++) {  /*  imagen.  */
          Matriz[i-N/2][j-N/2][0] = Matriz[i][j][0];
          Matriz[i-N/2][j-N/2][1] = - Matriz[i][j][1];  }
 for (i = 0; i < N/2; i++)  /*  Aux en esquina inferior derecha*/
   for (j = 0; j < N/2; j++) {  /*  imagen.  */
          Matriz[i+N/2][j+N/2][0] = Aux[i][j][0];
          Matriz[i+N/2][j+N/2][1] = Aux[i][j][1];  }
 for (i = 0; i < N/2; i++)  /*  Esquina superior derecha en Aux */
   for (j = N/2; j < N; j++) {  /*  imagen.  */
          Aux[i][j-N/2][0] = Matriz[i][j][0];
          Aux[i][j-N/2][1] = - Matriz[i][j][1];  }
 for (i = N/2; i < N; i++)  /*  Esquina inferior izquierda en superior d */
   for (j = 0; j < N/2; j++) {  /*  imagen.  */
          Matriz[i-N/2][j+N/2][0] = Matriz[i][j][0];
          Matriz[i-N/2][j+N/2][1] = - Matriz[i][j][1];  }
 for (i = 0; i < N/2; i++)  /*  Aux en esquina inferior izquierda*/
   for (j = 0; j < N/2; j++) {  /*  imagen.  */
          Matriz[i+N/2][j][0] = Aux[i][j][0];
          Matriz[i+N/2][j][1] = Aux[i][j][1];  }

 FFT2D(6);  /*  Aplicamos la transfomada a Matriz conjugada  */
 for (i = 0; i <= N-1; i++)  /*  Recuperamos la original  */
   for (j = 0; j <= N-1; j++) {
```

```
            Matriz[i][j][1] = - Matriz[i][j][1];  /*  Asignamos a imagen  */
            *(Imagen + i * N + j) = (unsigned char)abs(Matriz[i][j][0]);   }
  printf("\n VEMOS LA IMAGEN RESULTADO DE LA TRANSFORMADA INVERSA ");
  getch();  Poner_modo(0x13);  Poner_paleta();
  Poner_imagen(Imagen);    Poner_modo(0x03);  Salvar_imagen(Imagen);
  farfree((void *)Imagen);   }

/* FUNCION QUE REALIZA EL FILTRADO PASO ALTAS */
void Paso_altas()  {
  int i,j, fi, fj, D;
  int r;
  long float Pt, P;
  float B;

  do {
   printf("\nCUAL ES LA FRECUENCIA DE CORTE? [ENTRE 1 Y 63] ");
   scanf("%d",&r);
  }while((r < 1) || (r > 63));
  for (i = 0; i <= N/2; i++)
    for (j = 0; j < N/2-1; j++) {
            Pt += Matriz[i][j][0]*Matriz[i][j][0] +
               Matriz[i][j][1]*Matriz[i][j][1];
            fi = N/2 - i;
            fj = j - (N/2-1);
            D = (int)(sqrt(fi*fi + fj*fj));
            if (D <= r) {
              Matriz[i][j][0] = 0.;
              Matriz[i][j][1] = 0.;   }
            else
              P += Matriz[i][j][0]*Matriz[i][j][0] +
                      Matriz[i][j][1]*Matriz[i][j][1];    }
  for (i = N/2+1; i < N; i++)
    for (j = 0; j < N/2-1; j++) {
            Pt += Matriz[i][j][0]*Matriz[i][j][0] +
               Matriz[i][j][1]*Matriz[i][j][1];
            fi = N/2 - i;
            fj = j - (N/2-1);
            D = (int)(sqrt(fi*fi + fj*fj));
            if (D <= r) {
              Matriz[i][j][0] = 0.;
              Matriz[i][j][1] = 0.;   }
            else
              P += Matriz[i][j][0]*Matriz[i][j][0] +
                      Matriz[i][j][1]*Matriz[i][j][1];    }
  for (i = 0; i <= N/2; i++)
    for (j = N/2-1; j < N; j++) {
            Pt += Matriz[i][j][0]*Matriz[i][j][0] +
               Matriz[i][j][1]*Matriz[i][j][1];
            fi = N/2 - i;
            fj = j - (N/2-1);
            D = (int)(sqrt(fi*fi + fj*fj));
            if (D <= r) {
              Matriz[i][j][0] = 0.;
```

```
            Matriz[i][j][1] = 0.;   }
          else
            P += Matriz[i][j][0]*Matriz[i][j][0] +
                 Matriz[i][j][1]*Matriz[i][j][1];    }
 for (i = N/2+1; i < N; i++)
   for (j = N/2-1; j < N; j++) {
          Pt += Matriz[i][j][0]*Matriz[i][j][0] +
             Matriz[i][j][1]*Matriz[i][j][1];
          fi = N/2 - i;
          fj = j - (N/2-1);
          D = (int)(sqrt(fi*fi + fj*fj));
          if (D <= r) {
            Matriz[i][j][0] = 0.;
            Matriz[i][j][1] = 0.; }
          else
            P += Matriz[i][j][0]*Matriz[i][j][0] +
                 Matriz[i][j][1]*Matriz[i][j][1];    }
 B = 100. * (P / Pt);
 printf("EL PORCENTAJE DE LA POTENCIA FILTRADA ES %3.2f\n", B);
 return;  }

/* FUNCION QUE REALIZA EL FILTRADO PASO ALTAS */
void Butterworth_bajas()  {
 int i,j, fi, fj, D;
 int r, n;
 long float Pt, P;
 float B;

 Pt = 0;
 P = 0;
 do {
  printf("\nCUAL ES LA FRECUENCIA DE CORTE? [ENTRE 1 Y 63] ");
  scanf("%d",&r);
 }while((r < 1) || (r > 63));
 do {
  printf("\nDE QUE ORDEN SERA EL FILTRO? [ENTRE 1 Y 10] ");
  scanf("%d",&n);
 }while((n < 1) || (n > 10));
 for (i = 0; i <= N/2; i++)
   for (j = 0; j < N/2-1; j++) {
          Pt += Matriz[i][j][0]*Matriz[i][j][0] +
             Matriz[i][j][1]*Matriz[i][j][1];
          fi = N/2 - i;
          fj = j - (N/2-1);
          D = (int)(sqrt(fi*fi + fj*fj));
          Matriz[i][j][0] = 1/(1+0.414*pow(D/r,2*n))*Matriz[i][j][0];
          Matriz[i][j][1] = 1/(1+0.414*pow(D/r,2*n))*Matriz[i][j][1];
          P += Matriz[i][j][0]*Matriz[i][j][0] +
             Matriz[i][j][1]*Matriz[i][j][1];    }
 for (i = N/2+1; i < N; i++)
   for (j = 0; j < N/2-1; j++) {
          Pt += Matriz[i][j][0]*Matriz[i][j][0] +
             Matriz[i][j][1]*Matriz[i][j][1];
```

```
            fi = N/2 - i;
            fj = j - (N/2-1);
            D = (int)(sqrt(fi*fi + fj*fj));
            Matriz[i][j][0] = 1/(1+0.414*pow(D/r,2*n))*Matriz[i][j][0];
            Matriz[i][j][1] = 1/(1+0.414*pow(D/r,2*n))*Matriz[i][j][1];
            P += Matriz[i][j][0]*Matriz[i][j][0] +
                Matriz[i][j][1]*Matriz[i][j][1];    }
  for (i = 0; i <= N/2; i++)
    for (j = N/2-1; j < N; j++) {
            Pt += Matriz[i][j][0]*Matriz[i][j][0] +
                Matriz[i][j][1]*Matriz[i][j][1];
            fi = N/2 - i;
            fj = j - (N/2-1);
            D = (int)(sqrt(fi*fi + fj*fj));
            Matriz[i][j][0] = 1/(1+0.414*pow(D/r,2*n))*Matriz[i][j][0];
            Matriz[i][j][1] = 1/(1+0.414*pow(D/r,2*n))*Matriz[i][j][1];
            P += Matriz[i][j][0]*Matriz[i][j][0] +
                Matriz[i][j][1]*Matriz[i][j][1];    }
  for (i = N/2+1; i < N; i++)
    for (j = N/2-1; j < N; j++) {
            Pt += Matriz[i][j][0]*Matriz[i][j][0] +
                Matriz[i][j][1]*Matriz[i][j][1];
            fi = N/2 - i;
            fj = j - (N/2-1);
            D = (int)(sqrt(fi*fi + fj*fj));
            Matriz[i][j][0] = 1/(1+0.414*pow(D/r,2*n))*Matriz[i][j][0];
            Matriz[i][j][1] = 1/(1+0.414*pow(D/r,2*n))*Matriz[i][j][1];
            P += Matriz[i][j][0]*Matriz[i][j][0] +
                Matriz[i][j][1]*Matriz[i][j][1];    }
  B = 100. * (P / Pt);
  printf("EL PORCENTAJE DE LA POTENCIA FILTRADA ES %3.2f\n", B);
  return;  }

/* FUNCION QUE REALIZA EL FILTRADO PASO ALTAS */
void Butterworth_altas()  {
  int i,j, fi, fj;
  float D;
  int r, n;
  long float Pt, P;
  float B;

  Pt = 0; P = 0;
  do {
    printf("\nCUAL ES LA FRECUENCIA DE CORTE? [ENTRE 2 Y 63] ");
    scanf("%d",&r);
  }while((r < 2) || (r > 63));
  do {
    printf("\nDE QUE ORDEN SERA EL FILTRO? [ENTRE 1 Y 10] ");
    scanf("%d",&n);
  }while((n < 1) || (n > 10));
  for (i = 0; i < N/2; i++)
    for (j = 0; j < N/2; j++) {
            Pt += Matriz[i][j][0]*Matriz[i][j][0] +
```

```
            Matriz[i][j][1]*Matriz[i][j][1];
        fi = N/2 - i;
        fj = j - N/2;
        D = sqrt(fi*fi + fj*fj);
        Matriz[i][j][0] = 1/(1+0.414*pow(r/D,2*n))*Matriz[i][j][0];
        Matriz[i][j][1] = 1/(1+0.414*pow(r/D,2*n))*Matriz[i][j][1];
        P += Matriz[i][j][0]*Matriz[i][j][0] +
            Matriz[i][j][1]*Matriz[i][j][1];    }
for (i = N/2; i < N; i++)
  for (j = 0; j < N/2; j++) {
        Pt += Matriz[i][j][0]*Matriz[i][j][0] +
            Matriz[i][j][1]*Matriz[i][j][1];
        fi = (N/2-1) - i;
        fj = j - N/2;
        D = sqrt(fi*fi + fj*fj);
        Matriz[i][j][0] = 1/(1+0.414*pow(r/D,2*n))*Matriz[i][j][0];
        Matriz[i][j][1] = 1/(1+0.414*pow(r/D,2*n))*Matriz[i][j][1];
        P += Matriz[i][j][0]*Matriz[i][j][0] +
            Matriz[i][j][1]*Matriz[i][j][1];    }
for (i = 0; i < N/2; i++)
  for (j = N/2; j < N; j++) {
        Pt += Matriz[i][j][0]*Matriz[i][j][0] +
            Matriz[i][j][1]*Matriz[i][j][1];
        fi = N/2 - i;
        fj = j - (N/2-1);
        D = sqrt(fi*fi + fj*fj);
        Matriz[i][j][0] = 1/(1+0.414*pow(r/D,2*n))*Matriz[i][j][0];
        Matriz[i][j][1] = 1/(1+0.414*pow(r/D,2*n))*Matriz[i][j][1];
        P += Matriz[i][j][0]*Matriz[i][j][0] +
            Matriz[i][j][1]*Matriz[i][j][1];    }
for (i = N/2; i < N; i++)
  for (j = N/2; j < N; j++) {
        Pt += Matriz[i][j][0]*Matriz[i][j][0] +
            Matriz[i][j][1]*Matriz[i][j][1];
        fi = (N/2-1) - i;
        fj = j - (N/2-1);
        D = sqrt(fi*fi + fj*fj);
        Matriz[i][j][0] = 1/(1+0.414*pow(r/D,2*n))*Matriz[i][j][0];
        Matriz[i][j][1] = 1/(1+0.414*pow(r/D,2*n))*Matriz[i][j][1];
        P += Matriz[i][j][0]*Matriz[i][j][0] +
            Matriz[i][j][1]*Matriz[i][j][1];    }
B = 100. * (P / Pt);
printf("EL PORCENTAJE DE LA POTENCIA FILTRADA ES %3.2f\n", B);
return;  }
```

Conclusiones.

El procesamiento digital de imágenes es una herramienta útil para el desarrollo de numerosas y diversas aplicaciones. Es empleada en medicina, biología, astronomía, robótica, publicidad, televisión, periodismo, etc. Su uso se ha extendido gracias al desarrollo de la tecnología en electrónica y la computación. Los avances en electrónica contribuyen construyendo nuevos equipos más potentes, con costos cada vez más bajos; la computación por su parte diseña algoritmos más eficientes, teniendo como fundamento las matemáticas discretas y aplicando a su vez, las nuevas herramientas de software que contribuyan a la creación de aplicaciones más complejas y sofisticadas.

Las técnicas de realce de imágenes, pueden formar parte de un sistema de procesamiento que cumpla una tarea compleja, o pueden formar una aplicación cuya única finalidad sea suprimir o enfatizar características que contribuyan al mejoramiento de la imagen. La selección de la técnica que cumpla el objetivo deseado, dependerá del objetivo que se pretende obtener y de las características de la imagen. Las técnicas que pueden ser útiles para el realce de una imagen; pueden ser inadecuadas para otra. El problema principal radica en seleccionar la técnica adecuada para conseguir el efecto deseado en la imagen.

Las técnicas expuestas en el presente trabajo contemplan:

-Obtención del negativo de una imagen: Muy empleado en la medicina, en el estudio de radiografías

-Ajuste del contraste: Manipula el contraste de la imagen en forma interactiva.

-Compresión de rango dinámico: Permite visualizar mayores detalles en la transformada de Fourier de una imagen.

-División por planos de bits: Secciona los bits de los pixeles de la imagen, para dar la posibilidad de cambiar la característica de alguno de ellos.

-Ecualización del histograma: Realiza un ajuste de contraste de manera automática, proporcionándole una distribución de los niveles de gris más uniforme a la imagen.

-Especificación del histograma: Realiza una modificación en el contraste de acuerdo a un histograma deseado. Su dificultad consiste en conseguir un histograma útil para el usuario.

-Filtrado paso bajas: Elimina ruido indeseado en una imagen. Tiene la desventaja de hacer borrosos los bordes.

-Filtrado de mediana: Elimina ruido, con la ventaja de que conserva en mayor medida el detalle de los bordes.

-Filtrado paso altas: Resalta los bordes de una imagen, con la desventaja de reducir el contraste de la imagen.

-Filtrado High boost: Se trata de un tipo de filtro paso altas interactivo, donde se pueden resaltar los bordes de una imagen, y controlar el contraste del resto de la imagen.

-Filtros de diferencia: Detecta los bordes de una imagen. Son empleados en robótica como la primera fase de procesamiento para el reconocimiento de patrones. Se pueden aplicar tres tipos de operadores Roberts, Prewitt, Sobel. Los operadores de Roberts obtiene los bordes con líneas delgadas. Los operadores Prewitt obtiene el mismo resultado remarcando las líneas con el doble de grosor. Los operadores Sobel, obtienen el resultado de los operadores Prewitt realizando adicionalmente un suavizado de la imagen.

-Diferenciación estadística: Obscurecen imágenes muy brillantes para obtener los bordes imperceptibles en la imagen original.

-Ampliación de imágenes por réplica de pixeles: Aumenta el tamaño de una imagen, con la desventaja de incrementar el ruido de cuantización.

-Ampliación de imágenes por interpolación: Aumenta la imagen interpolando los pixeles de la imagen original, se obtienen resultados muy satisfactorios.

-Filtros en el dominio de las frecuencias: Se fundamentan en la transformada de Fourier de una imagen. Al encontrarse la información de la imagen en el plano de frecuencias, los filtros paso bajas y paso altas se obtiene de una manera más natural e interactiva.

Como se puede observar existe una amplia gama de técnicas de realce que cumplen diferentes objetivos. La más adecuada será la que proporcione la imagen más útil para el usuario.

Bibliografía.

[1] Rafael C. Gonzalez, Richard E. Woods. Digital Image Processing. Addison-Wesley Publishing Company 1992

[2] Azriel Rosenfeld and Avinash C. Kak. Digital Picture Processing. Second edition. Volumen 1. Academic Press. 1982.

[3] Jahne Bernd. Digital Image Processing:Concepts, Algorithms and Scientific Aplications. Berlin:Springer. 1991

[4] William K. Pratt. Digital Image Processing. Second Edition. John Wiley & Sons, Inc. 1991.

[5] William B. Green. Digital Image Processing A systems approach. Second Edition. Van Nostrand Reinhold. 1989

[6] Kenneth R. Castleman. Digital Image Processing. Prentice-Hall. 1979.

[7] Jain K. Anil. Fundamentals of Digital Image Processing. Prentice Hall. 1989.

[8] Theo Pavlidis. Algorithms for Graphics and Image Processing. 1979.

[9] Rama Chellappa and Alexander A. Sawchuk. Digital Image Processing and Analysis. Volume 1. IEEE Computer Society.

[10] Loren Heiny. Advanced Graphics Programming using C/C++. Wiley & sons. 1993.

[11] Lee Adams. Programación avanzada de gráficos en C para Windows. McGraw Hill. 1993.

[12] Herbert Schild. Turbo C Programación avanzada. Segunda edición. McGraw Hill. 1992.

[13] Herbert Schildt. Programación en Turbo C. Segunda edición McGraw-Hill. 1992.

[14] Brian W. Kernighan, Dennis M. Ritchie. El lenguaje de programación C. Prentice Hall.

[15] Borland. Turbo C++ 3.0 User's guide. Borland international, Inc. 1992.

[16] Borland. Turbo C++ 3.0 Programmer's guide. Borlandc international, Inc. 1992.

[17] Ted Faison. Borland C++ 3.1. Prentice Hall. 1993.

[18] Willian H. Murray. III, Chris H. Pappas. 80386/80286 Programación en lenguaje ensamblador. McGraw Hill. 1990.