Biblioteca de programas para el procesamiento digital de imágenes

J.J. Báez-Rojas[1], F. Lobato-López[2,†], A. Padilla-Vivanco[3,†], F.S. Granados-Agustín[4,†] y A. Cornejo-Rodríguez
Instituto Nacional de Astrofísica Óptica y Electrónica
Apartado postal 51 y 216, Puebla, Pue., Mexico
e-mail: [1]jbaez@inaoep.mx; [2]flobato@inaoep.mx; [3]apadilla@inaoep.mx; [4]fgranados@inaoep.mx

Recibido el 27 de junio de 1997; aceptado el 18 de junio de 1998

El objetivo de este trabajo es conjuntar en una biblioteca los programas e imágenes que se usan comúnmente en los cursos de procesado digital de imágenes, pero que normalmente no se encuentran disponibles al público. La biblioteca contiene programas en lenguaje C para estaciones de trabajo, y archivos de imágenes sin formato. Se muestran los resultados de cada uno de los programas desarrollados.

Descriptores: Procesamiento de imágenes; resolución espacial; ruido de cuantización; modificación de histogramas; transformada de Fourier

The purpose of this work is to assemble in a library a set of programs and images that are commonly used on digital image processing courses, but they are not available easily. The library has programs written on C language and raw format images. The results of each program are shown.

Keywords: Digital image processing; spacial resolution; quantization noise; modification of hystograms; Fourier transform

PACS: 01.40.-d; 42.30.Yc

1. Introducción

En todos los cursos de procesamiento digital de imágenes, se requiere de un conjunto de imágenes y programas para demostrar de una manera sencilla y en la computadora una gran cantidad de fenómenos visuales, como por ejemplo los resultados obtenidos por el cambio de la resolución espacial en una imagen, ruido de cuantización, restauración de imágenes, detección de orillas, etc. La biblioteca que se presenta contiene programas para detectar orillas usando el laplaciano y el gradiente, para modificar la resolución espacial y la escala de grises, para obtener el histograma de una imagen y modificar su rango dinámico, adelgazamiento, desplegado, obtención de imágenes de superficies a partir de arreglos volumétricos de datos y obtención de la transformada de Fourier.

En la mayoria de los libros de texto para cursos de procesamiento digital de imágenes se presentan ejemplos y resultados de los diferentes operadores más usados en esta área. Generalmente en los libros de texto no aparecen los algoritmos y tampoco se explica en detalle la programación de éstos para poder obtener resultados semejantes a los mostrados en los mismos. La biblioteca de programas que se presenta aquí, ha sido construida a lo largo de tres cursos de procesamiento digital de imágenes dirigidos a estudiantes de doctorado y maestría en el programa de posgrado del INAOE. Los resultados parciales se han presentado en los congresos de la Sociedad Mexicana de Física (SMF) y han despertado interés en mucha gente, manifestando algunos su deseo de tener copias de varios de estos programas. Esto nos motivó a escribir el presente artículo y así darle una mayor difusión a este trabajo, que si bien es cierto se obtienen resultados clásicos que se muestran en los libros de texto no hay lugares en donde se puedan obtener los programas, o algoritmos. Existen programas comerciales, los cuales solo incluyen algunas de las operaciones mostradas. Aparte de que estos programas son caros, no permiten al usuario involucrarse en la manipulación de los programas y adecuarlos a sus necesidades particulares. La intención de esta biblioteca es familiarizar al usuario con las técnicas y algoritmos básicos del procesamiento digital de imágenes, así como permitirle cambiar parámetros y adecuar los programas a sus necesidades. Copias de estos programas e imágenes se pueden obtener por medio del internet en el servidor de Web de la Coordinación de óptica del INAOE (http://www-optica.inaoep.mx/~jbaez).

El procesamiento de imágenes analógico o digital, actualmente se usa en muchas de las áreas de investigación científica y desarrollo tecnológico. Básicamente, los métodos de procesamiento de imágenes se pueden dividir en dos: métodos en el dominio del espacio y métodos en el dominio de la frecuencia. Independientemente del dominio al que los métodos pertenecen por procesamiento debemos entender las operaciones que se aplican a la imagen y que la modifican [1, 2]. Dentro de las diferentes operaciones que se pueden aplicar a una imagen destacan las siguientes:

- Realce (modificación del intervalo dinámico, suavizado, realce de bordes).
- Restauración (eliminación de ruido, eliminación de borrosidad y deformaciones, filtrado recursivo).
- Reconstrucción (filtrado inverso, mínimos cuadrados).
- Segmentación (detección de bordes, detección de ciertas características y texturas).

Este trabajo presenta programas que utilizan métodos pertenecientes al dominio espacial. Las imágenes que se usan con estos programas son imágenes monocromáticas, también llamadas imágenes en blanco y negro.

Para estar en condiciones adecuadas para el procesamiento por computadora, una imagen, representada por una función bidimensional $f(x, y)$ (Fig. 1), debe ser digitalizada tanto espacialmente como en amplitud (intensidad).

La digitalización de las coordenadas espaciales (x, y) se denomina muestreo de la imagen, mientras que la digitalización de amplitud se llama cuantización de intensidad o nivel de gris. Este último término es aplicable a las imágenes monocromáticas y refleja el hecho de que éstas varían del negro al blanco en tonos de gris.

Supóngase que una imagen continua se muestrea uniformemente obteniéndose una matriz de M filas y N columnas, donde cada muestra está cuantizada en intensidad. Esta matriz, llamada imagen digital, puede ser representada como

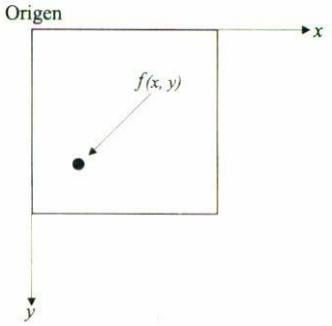


FIGURA 1. La imagen de un objeto puede ser representada como una función bidimensional, de variables continuas, la cual debe ser discretizada, función de variables discretas, para poder procesarse por medio de la computadora.

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

donde x e y son ahora variables discretas: $x = 0, 1, 2, \dots, M-1$; $y = 0, 1, 2, \dots, N-1$. Cada elemento de la matriz se llama elemento de la imagen o pixel (del inglés *Picture Element*). Tomando como referencia la Fig. 2, se observa que $f(0,0)$ representa el pixel del origen de la imagen, $f(0,1)$ el pixel que se encuentra a su derecha, y así sucesivamente. Es una práctica habitual hacer que M, N y el número de niveles discretos de intensidad de cada pixel cuantizado sean potencias enteras de 2; esto facilita la manipulación de la imagen dentro de los algoritmos.

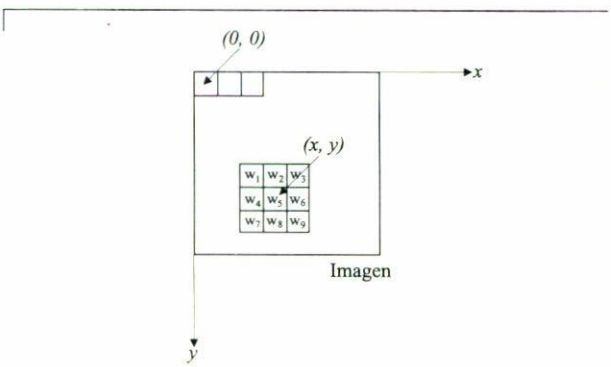


FIGURA 2. Los métodos en el dominio del espacio utilizan para su procesamiento la vecindad, de 3×3, de un elemento de la imagen. Esta vecindad puede ser mayor y no necesariamente cuadrada.

1.1. Métodos en el dominio del espacio

Los métodos en el dominio del espacio son procedimientos que operan de forma directa sobre los pixeles que componen la imagen. Las funciones de procesamiento en el dominio espacial se pueden expresar como

$$g(x,y) = h[f(x,y)], \tag{2}$$

donde $f(x,y)$ es la imagen de entrada, $g(x,y)$ es la imagen que se obtiene y $h[.]$ es un operador definido sobre la vecindad del pixel localizado en las coordenadas (x,y).

La forma más común para definir un entorno de vecindad sobre (x,y) es tomar una zona rectangular o cuadrada de la imagen centrada en (x,y), como se muestra en la Fig. 2. El centro de la "subimagen" definida, pixel w_5, se irá desplazando de pixel a pixel, comenzando por ejemplo, en la esquina superior izquierda y el operador se irá aplicando a cada pixel localizado en (x,y) para obtener $g(x,y)$. Aunque otras configuraciones de vecindad, tal como el círculo, se usan algunas veces, la cuadrada es con mucho la configuración predominante debido a la facilidad con que se trabaja con ella.

Las técnicas de procesamiento en el dominio espacial frecuentemente usadas están basadas en el uso de las llamadas máscaras de convolución (plantillas, ventanas o filtros). Una máscara es, básicamente, una matriz (por ejemplo de 3×3 elementos), cuyos coeficientes se eligen de forma que podamos detectar una propiedad específica de la imagen.

Como se observa en la Fig. 2, si llamamos $w_1, w_2, \dots, w_9$ a los coeficientes de la máscara y consideramos los 8 vecinos de (x,y) podemos obtener el procedimiento general para realizar una determinada operación:

$$\begin{aligned} h[f(x,y)] = {} & w_1 f(x-1,y-1) + w_2 f(x-1,y) + w_3 f(x-1,y+1) + w_4 f(x,y-1) + w_5 f(x,y) + w_6 f(x,y+1) \\ & + w_7 f(x+1,y-1) + w_8 f(x+1,y) + w_9 f(x+1,y+1), \end{aligned} \tag{3}$$

esto en un entorno de vecindad de (x,y) de 3×3.

Cabe señalar que la técnica de procesamiento por entorno de vecindad no está limitada a áreas de 3×3 ni a operadores lineales.

2. Imagen digital, resolución espacial y ruido de cuantización

Como hemos mencionado, las imágenes monocromáticas pueden ser consideradas como un arreglo rectangular (matriz) de elementos (pixeles); cada uno de estos elementos tiene asociado un valor numérico entero que representa una propiedad física de la imagen en ese punto en particular. A este valor numérico nos vamos a referir como el valor del pixel. Normalmente tanto los valores de cada pixel como las coordenadas de su posición son números enteros. En los métodos en el dominio de la frecuencia se trabaja con pixeles complejos obtenidos al aplicar la transformada de Fourier a una imagen.

Al número de pixeles que forman la imagen se le llama resolución espacial y depende del sistema de adquisición de datos. El tamaño típico de los arreglos son de: 64×64, 128×128 256×256 y 512×512 (casi siempre el tamaño de una imagen es una potencia de 2^n pixeles). Sin embargo, las ideas aquí expuestas pueden ser generalizadas fácilmente para cualquier tamaño espacial de la imagen, incluso imágenes rectangulares. La Fig. 3 muestra los resultados de cambiar la resolución espacial. Debe notarse que conforme decrece la resolución espacial (número de pixeles por unidad de superficie), se van perdiendo los detalles de la imagen. En los resultados mostrados se ha amplificado el tamaño del pixel para poder compararlos con la original. A continuación se muestra el algoritmo que desarrolla el cambio de la resolución espacial:

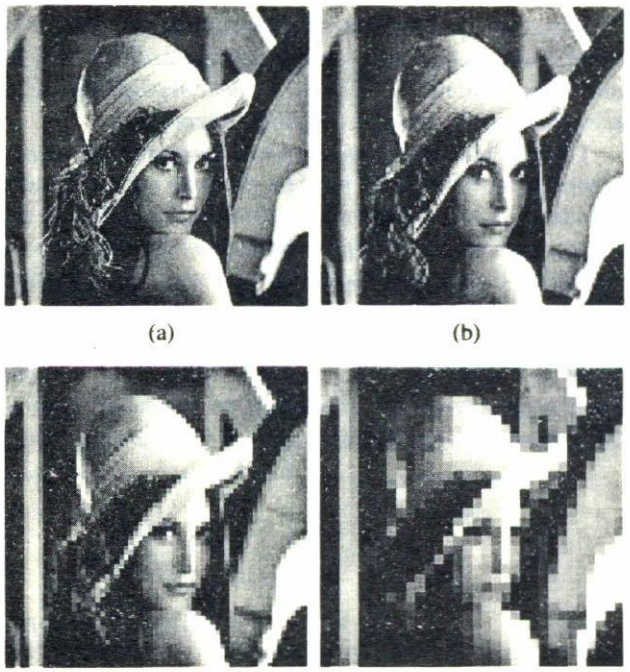


FIGURA 3. En esta figura se muestran los efectos de reducir la resolución espacial en una imagen. (a) Original, consta de 256×256 pixeles. (b) 128×128 pixeles, la resolución se disminuyó a la mitad. (c) 64×64 pixeles. (d) 32×32 pixeles. En la medida en que disminuye el número de pixeles se pierden detalles de la imagen.

Considerando que la imagen es cuadrada y que las dimensiones son potencias de 2; sean:
Imagen_Ini = Imagen original con dimensiones: *Renglon_Ini* = *Columna_Ini* = *Dimen_Ini*
Imagen_Fin = Imagen resultante con dimensiones: *Renglon_Fin* = *Columna_Fin* = *Dimen_Fin*-
Dimen_Ini = *Dimen_Fin* debido a que en la imagen resultante se ha amplificado el tamaño del pixel
% Factor de Escala.
Escala = *Dimen_Ini/Dimen_Fin*
Escala_2 = *Escala* * *Escala*
% Se inicia el cambio de la resolución espacial.

```
for (i = 1; i<= Dimen_Ini; i =i+Escala)do      % Primero se toman todos los pixeles de la
  for (j=1; j<= Dimen_Ini; j = j+Escala) do     %imagen en pasos de Escala.
    Suma = 0
    for (k = i; k<= i+Escala-1; i++) do        % Ahora sumamos el valor de los pixeles
      for (l = j; j<= j+Escala-1; j++) do      %que hay entre cada paso de Escala.
        Suma = Suma + Imagen_Ini(k,l)
      end
    end
    Valor_Pixel = Suma/Escala_2        % Promediamos la suma de los pixeles.
    for (k = i; k<= i+Escala; i++) do        % Ahora colocamos el valor en cada uno de los pixeles.
      for (l=j; j<= j+Escala; j++) do        %que hay entre cada paso de Escala pero en Imagen_Fin.
        Imagen_Fin(k, l) = Valor_Pixel
          end
        end
      end
    end
```

Los intervalos de valores para los pixeles de una imagen son determinados por el sistema de detección. El valor máximo del intervalo está determinado por el número de *bits* asociado a cada pixel y se obtienen por la relación $Valor\_del\_Pixel = 2^n$ donde n es el número de *bits*. Al número de *bits* por pixel se le llama definición de la imágen. Por otro lado, el sistema de desplegado tiene inherente un intervalo dinámico de valores. Este intervalo dinámico de valores determina el número de tonos en gris (niveles de gris) que puede desplegar dicho sistema. La mayoría de los sistemas monocromáticos (blanco y negro) de desplegado tienen una capacidad de 256 niveles de gris. La manera más común de desplegar una imagen en un monitor blanco y negro es estableciendo un mapeo entre estos dos intervalos. Es decir, que se mapean los valores de los pixeles a niveles de gris. Este mapeo en general se realiza de manera lineal. Comúnmente se asocia el tono de gris mínimo (negro) al pixel de valor más pequeño del intervalo de valores y el tono de gris máximo (blanco) al pixel de mayor valor del intervalo. Ésta es una convención, pero de la misma manera se puede asociar el tono blanco al pixel con el menor valor y el negro al pixel con mayor valor.

Nótese que *Valor_del_Pixel* y nivel de gris son dos conceptos diferentes, pero algunos autores se refieren a éstos de una manera indistinta. Sobre todo cuando las imágenes con las cuales se trabaja tienen valores del pixel en el intervalo [0, 255].

Disminución en la definición de la imagen significa disminuir el número de *bits* que representa a un pixel. Esto también significa una reducción en los niveles de gris al momento de desplegar dicha imagen. Al efecto de disminuir la definición de la imagen, también se le conoce como ruido de cuantización, estos efectos se pueden apreciar en la Fig. 4. Debe notarse que conforme el número de *bits* por pixel disminuye, aparece dicho ruido de cuantización. A continuación se muestra el algoritmo que desarrolla la reducción en los niveles de gris o ruido de cuantización:

FIGURA 4. Conforme disminuye la definición de la imagen, aparece el ruido de cuantización. El ruido de cuantización se manifiesta como regiones con el mismo tono de gris. Todas las imágenes consisten de 256×256 pixeles. (a) 6 *bits* por pixel, (b) 5 *bits* por pixel, (c) 4 *bits* por pixel, (d) 3 *bits* por pixel, (e) 2 *bits* por pixel, (f) 1 *bit* por pixel.

Considerando que la imagen es cuadrada y que las dimensiones son potencias de 2; sean:

Imagen_Ini = Imagen original con dimensiones: *Renglon_Ini* = *Columna_Ini* = *Dimen_Ini*

Tonos_Ini = Número de tonos de la imagen original.

Imagen_Fin = Imagen resultante con dimensiones: *Renglon_Fin* = *Columna_Fin* = *Dimen_Fin*

Tonos_Fin = Número de tonos deseados de la imagen resultante.

Dimen_Ini = *Dimen_Fin* debido a que a la imagen resultante se le ha modificado su escala de grises no su resolucion espacial.

% Relación de tonos.

Relacion_Tonos = *Tonos_Ini/Tonos_Fin*

%Cambiamos la escala de grises.

for (*i* = 1; *i*<= *Dimen_Ini*; *i*++) do
  for (*j* = 1; *j*<= *Dimen_Ini*; *j*++) do
    Imagen_Fin(i,j) = *Imagen_Ini(i,j)/Relacion_Tonos*
    end
  end

% La imagen resultante ahora tiene el número de tonos que se deseaba y solo falta distribuirlos en todo el rango; es decir el número de tonos distribuidos en el rango de [0,255].

% Hallamos el máximo y mínimo valor de la imagen.

Max_Img = Máximo valor de *Imagen_Fin*

Min_Img = Mínimo valor de *Imagen_Fin*
%Escalamos o mapeamos *Imagen_Fin* al rango de [0,255]
for *(i = 1; i<= Dimen_Ini; i++)* do
for *(j = 1; j<= Dimen_Ini; j++)* do
Imagen_Fin(i, j) = (255/(Max_Img−Min_Img))(Imagen_Fin(i, j)−Min_Img)*
end
end

3. Histograma de una imagen digital y modificación del intervalo dinámico

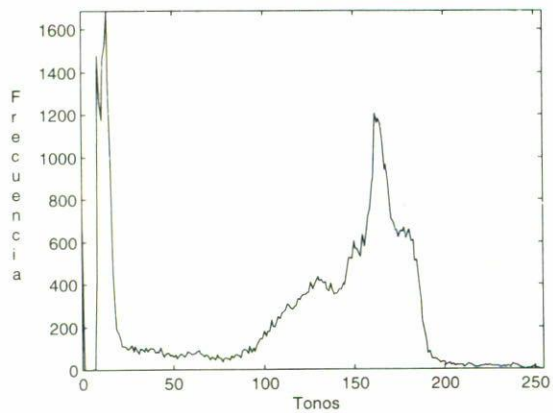
La primera aproximación al análisis estadístico de una imagen digital es construir su histograma [3–5]. El histograma nos permite saber cuántos pixeles de un cierto valor contiene la imagen, y en cuantas regiones se pueden clasificar los valores de los pixeles. Con el histograma es muy fácil visualizar cómo es el cambio de intensidades de una región a otra. A partir del histograma se pueden determinar valores de umbral que se pueden usar en la segmentación de la imagen. De manera que la obtención del histograma de una imagen es de crucial importancia. La modificación del histograma resulta de una técnica de resaltado de las características de la imagen; también conocida como igualación del histograma [3].

La modificación del intervalo dinámico en una imagen, también conocida como modificación de la escala de grises, es una operación puntual que resulta en una transformación del valor de gris asociado a cada pixel [6]. Esta última es una técnica sencilla, muy útil y se usa frecuentemente como parte de las técnicas de resaltado en las imágenes. Las funciones usadas en esta transformación dependen de las características de la imagen original y de las características deseadas en la imagen resultante. La Fig. 5 muestra la imágen original, la Gráfica 1 muestra su histograma, la Fig. 6 muestra el resultado después de haber transformado la escala de gris con una función logarítmica $[y = \log(1 + x)]$. Esta modificación redistribuye los valores de pixel, de manera que la región de valores bajos los resalta. En la Fig. 6 se puede apreciar que el hombre de la cámara tiene guantes, también se aprecian los botones del abrigo, así como las detalles en el pantalón. Debe notarse también que los detalles en la zona brillante se pierden. Esto se debe a la forma de la función logarítmica. La Gráfica 2 muestra el histograma de la imagen modificada. A continuación se muestra el algoritmo para obtener el histograma de una imagen:

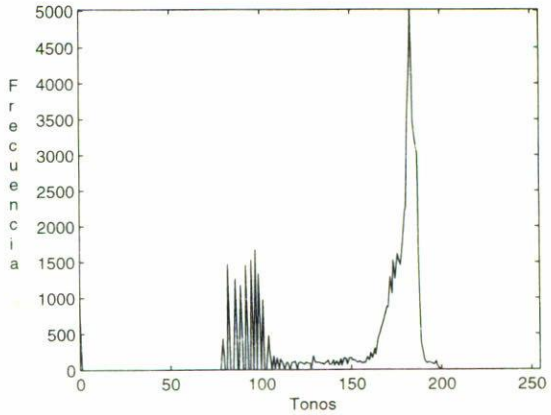
FIGURA 5. Imagen original, consiste de 256 × 256 pixeles, con 8 *bits* por pixel.

FIGURA 6. Imagen resultante después de haber realizado una trasformación logarítmica en su escala de gris a la imagen de la Fig. 5. Se pueden apreciar detalles en la zona oscura, que no se notaban en la original.

Considerando que las dimensiones de la imagen son conocidas así como el número de tonos que contiene; aunque podemos considerar que la imagen tiene 256 tonos de grises; sean:
Imagen_Ini = Imagen original con dimensiones: *Renglon_Ini* y *Columna_Ini*
Tonos_Total = Número de tonos de *Imagen_Ini*. (Puede considerarse que tiene el número máximo y todo el histograma resultara respecto al rango de [0, 255].)
%Inicializamos el arreglo que almacenara los valores del histograma.
for *(i = 1; i<= Tonos _Total; i++)* do
Histograma(i) =0
end
% Obtenemos el histograma de *Imagen_Ini*.
for *(i = 1; i<= Renglon_Ini; i++)* do



GRAFICA 1. Histograma de la imagen original, nótese los detalles en la zona brillante, y la homogeneidad de la zona oscura.



GRAFICA 2. Histograma de la imagen modificada, a la imagen original se le transformó su intervalo dinámico usando una función logarítmica, esto se refleja en un ensanchamiento de la zona de valores oscuros y una contracción en la zona de valores brillantes.

```
    for (j = 1; j<= Columna_Ini; j++) do
      Histograma(Imagen_Ini(i,j)) = Histograma(Imagen_Ini(i,j)) + 1
    end
  end
```

y ahora se muestra el algoritmo para modificar el histograma de una imagen por medio de una función logarítmica:

Considerando que las dimensiones de la imagen son conocidas; sea:

Imagen_Ini = Imagen original con dimensiones: *Renglon_Ini* y *Columna_Ini*

Imagen_Fin = Imagen resultante.

```
% Modificamos Imagen _Ini por medio de una función logarítmica
  for (i = 1; i<= Renglon_Ini; i++) do
    for (j = 1; j<= Columna_Ini; j++) do
      Imagen_Fin(i,j) = log(1 + Imagen_Ini(i,j))
    end
  end
```

Se puede utilizar cualquier función para modificar el histograma, por ejemplo, cuadrática, cúbica, raiz cuadrada, raiz cúbica, gausiana, etc.

4. Detección de orillas

La detección de orillas [7–9] es muy útil en un gran número de aplicaciones en el área del procesamiento digital de imágenes. Dentro de esas aplicaciones está el reconocimiento de patrones y el análisis de imágenes, entre otros. En el reconocimiento de patrones el primer paso es segmentar una imagen en diferentes regiones correspondiendo a diferentes objetos componentes de la imagen original; la detección de orillas es el primer paso en la segmentación de imágenes. En el análisis de imágenes un problema fundamental es la detección de orillas debido a que éstas caracterizan las fronteras de los objetos y por esta razón son importantes en la segmentación, registro e identificación de los objetos en las imágenes. La detección de orillas es un caso particular de la clasificación de pixeles. En la detección de orillas se toma en cuenta la vecindad y el valor del pixel, esto con la comparación de un umbral. En casi todos los libros de texto sobre procesamiento digital de imágenes se encuentra la explicación de los operadores laplaciano y gradiente. Éstos dos son los más comunes en la detección de orillas y sirven como base para otros operadores.

Básicamente una orilla en una imagen es un "frontera" o contorno en la cual un cambio significativo en algún aspecto físico de la imagen ocurre, tal como reflectancia de la superficie, iluminación, etc. Estos cambios en los aspectos físicos se manifiestan en formas diferentes, incluyendo cambios en la intensidad, color y textura de la imagen. En el presente trabajo sólo se están considerando cambios en la intensidad de la imagen. Esta biblioteca contiene dos de las técnicas más significativas para la detección de orillas: el gradiente y el laplaciano. Tanto el gradiente como el laplaciano son dos operadores que se pueden interpretar como la convolución de dos matrices. En la Fig. 7 se muestran las orillas detectadas con el laplaciano y el gradiente, tomando como imagen original la Fig. 3a. En la Fig. 8 se muestran los filtros usados en cada caso. A continuación se muestran los algoritmos empleados para la detección de orillas tanto con el gradiente como con el laplaciano:

(a) (b) (c)

FIGURA 7. (a) Orillas detectadas con el laplaciano, (b) orillas horizontales detectadas con el gradiente, (c) orillas verticales detectadas con el gradiente. Estos operadores son sensibles al ruido.

	1	
1	-4	1
	1	

(a)

-1	-1	-1
1	1	1

(b)

-1		1
-1		1
-1		1

(c)

FIGURA 8. (a) Máscaras usadas para obtener las orillas de la Fig. 7: (a) laplaciano, (b) gradiente horizontal, (c) gradiente vertical.

Considerando que la imagen es cuadrada y se conocen sus dimensiones; sean:

Imagen_Ini = Imagen original con dimensiones: *Renglon_Ini* = *Columna_Ini* = *Dimen_Ini*

Imagen_Fin = Imagen resultante con dimensiones: *Renglon_Fin* = *Columna_Fin* = *Dimen_Fin*

Umbral = Valor que determina si el pixel resultante pertenece a una orilla o no.

Dimen_Fil = Tamaño del filtro; en este caso en particular la dimensión del filtro es 3 × 3, por tanto tomamos el valor de 3, en caso de ser mayor tomamos ese valor.

Dimen_Ini = *Dimen_Fin* debido a que el procesamiento no afecta las dimensiones.

% Inicializamos el filtro considerando que *Laplaciano_Fil* es una matriz cuya dimensión es *Dimen_Fil*

Laplaciano_Fil[Dimen_Fil][Dimen_Fil] = [0, 1, 0; 1, -4, 1; 0, 1, 0]

% Se inicia la detección de orillas.

```
for (i = 1; i<= Dimen_Ini; i = i+Dimen_Fil) do      % Primero se toman todos los pixeles de la
  for (j = 1; j<= Dimen_Ini; j = j+Dimen_Fil) do      % imagen en pasos de Dimen_Fil.
    Suma_Conv = 0
    for (k = i; k<= i+Dimen_Fil-1; k++) do      % Ahora convolucionamos el valor de los pixeles
      for (l = j; l<= j+Dimen_Fil-1; l++) do      % que hay entre cada paso de Dimen_Fil.
        Suma_Conv = Suma_Conv + Laplaciano_Fil(k-i+1, l-j+1)*Imagen_Ini(k, l)
      end
    end
    if (Suma_Conv>= Umbral) do
      Imagen_Fin(i,j) = 1
    else
      Imagen_Fin(i, j) = 0
  end
end
```

% Hallamos el máximo y mínimo valor de la imagen.

Max_Img = Máximo valor de *Imagen_Fin* =1

Min_Img = Mínimo valor de *Imagen_Fin* =0

%Escalamos o mapeamos *Imagen_Fin* al rango de [0,255]

```
for (i = 1; i<= Dimen_Ini; i++) do
  for (j = 1; j<= Dimen_Ini; j++) do
    Imagen_Fin(i,j) = (255/(Max_Img − Min_Img))*(Imagen_Fin(i,j) − Min_Img)
end                    end
```

Para el caso del gradiente es exactamente lo mismo con la pequeña diferencia que los filtros que se utilizan son los siguientes:

Gradiente_Hor[3][3]= [-1,0,1; -1,0,1; -1,0,1]

Gradiente_Ver[3][3]= [-1,-1,-1; 0,0,0; 1,1,1]

5. Adelgazamiento

En varias aplicaciones de visión por computadora, es suficiente con caracterizar los objetos en una escena solo por sus estructuras, éstas normalmente están compuestas de líneas y arcos. Como ejemplos se tienen: manuscritos, caracteres impresos, cromosomas, estructuras celulares, circuitos eléctricos, diagramas en ingenieria, etc. En tales casos el grosor de las líneas en los patrones no contribuye a su reconocimiento. Una de las características que tienen las escenas mencionadas anteriormente es que son escenas binarias, es decir, escenas en donde sólo intervienen dos valores de pixel, por comodidad diremos pixeles blancos o pixeles negros. El programa que incluimos en esta biblioteca produce el adelgazamiento de líneas, o dicho de otra manera, obtiene la estructura de "esqueleto" de una escena; este programa sólo se dedica a trabajar sobre imágenes binarias. Sin embargo existen algoritmos que pueden obtener estructuras del esqueleto en imágenes multivaluadas. Multivaluado significa que los pixeles tienen más de dos valores. La Fig. 9 muestra una imagen original binaria y su estructura de "esqueleto" o adelgazada. Antes de mostrar el algoritmo que se utilizó es necesaria una breve explicación del mismo ya que el adelgazamiento de una región necesita del cumplimiento de un conjunto de condiciones.

Obtener el esqueleto de una región puede definirse mediante una transformación del eje central, la cual no se emplea directamente ya que requiere de un costo computacional alto [19]; pero en cambio se ha propuesto una serie de algoritmos, computacionalmente eficientes, que producen una representación del eje central de la región; estos son conocidos como algoritmos de adelgazamiento. Este tipo de algoritmos elimina las orillas de una región, pero bajo las siguientes condiciones: (1) no elimina puntos finales, (2) no rompe la continuidad y (3) no provoca una erosión excesiva de la región.

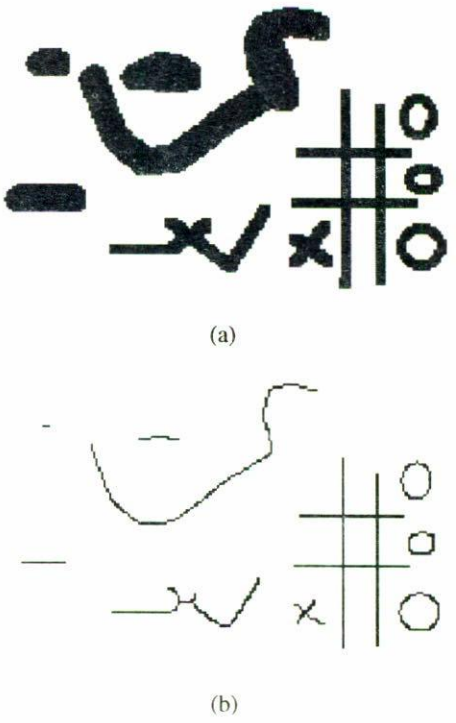
El algoritmo que se empleo [19] en este caso considera que los pixeles de la región tienen un valor de 1 y los pixeles del fondo un valor de 0; y consiste básicamente en la repetición sucesiva de dos pasos aplicados a los "puntos de contorno" de la región; un punto de contorno es cualquier pixel con valor 1 y que además tiene al menos un valor 0 en su vecindad. Con base a la vecindad mostrada en la Fig. 2, el primer paso que se aplica consiste en marcar un punto de contorno como candidato para eliminarse si las siguientes 4 condiciones se cumplen:

$$1)\ 2 \leq N(w_5) \leq 6, \qquad 2)\ S(w_5) = 1,$$

$$3)\ w_2 * w_6 * w_8 = 0, \qquad 4)\ w_6 * w_8 * w_4 = 0,$$

donde $N(w_5)$ es el número de vecinos de w_5 que no son fondo (valor 0), o sea,

$$N(w_5) = w_2 + w_3 + w_6 + w_9 + w_8 + w_7 + w_4 + w_1$$



(a)

(b)

FIGURA 9. Obtención del esqueleto de una imagen. (a) Original de 123×190 pixeles con 1 *bit* por pixel. (b) Esqueleto de la imagen (a).

y $S(w_5)$ es el número de transiciones $0 \rightarrow 1$ en la secuencia ordenada $w_2 \rightarrow w_3 \rightarrow w_6 \rightarrow w_9 \rightarrow w_8 \rightarrow w_7 \rightarrow w_4 \rightarrow w_1 \rightarrow w_2$. En el paso 2 las condiciones 1) y 2) se mantienen igual pero las condiciones 3) y 4) se cambian a:

$$3')\ w_2 * w_6 * w_4 = 0\,,$$

$$4')\ w_2 * w_8 * w_4 = 0\,.$$

Si en alguno de los pasos no se cumple una o más de las condiciones el pixel en cuestión no se marca y se continua con el siguiente; sin embargo, los pixeles sólo se marcan y no se borran sino hasta que todos los pixeles han sido procesados, esto previene el cambio en la estructura de la región durante la evaluación de alguno de los pasos del algoritmo.

Con estos dos pasos una iteración del algoritmo consiste en:

1) Se aplica el paso 1 a todos los pixeles y se marcan los que cumplen con las 4 condiciones.

2) Se borran todos los pixeles marcados, es decir, se ponen en 0.

3) Se aplica el paso 2 a todos los pixeles marcando aquellos que cumplen con las 4 condiciones.

4) Se borran los pixeles marcados.

Este proceso continúa iterativamente hasta que no hay más pixeles que se eliminen y por tanto se obtiene la imagen adelgazada.

A continuación se muestra el algoritmo empleado en el adelgazamiento de una región; pero debido a lo largo y repetitivo sólo se mostrará explícitamente la parte correspondiente al paso 1 ya que la parte correspondiente al paso 2 es similar con la excepción de las condiciones 3) y 4):

```
Considerando que la imagen es binaria y se conocen sus dimensiones, sean:
        Imagen_In = Imagen original con dimensiones: Renglon_Ini y Columna_Ini.
        Imagen_fin = Imagen resultante con dimensiones: Renglon_Ini y Columna_Ini.
        Img_proc_s1 = Imagen resultante del paso 1.
        Img_proc_s2 = Imagen resultante del paso 2.
        Img_proc = Imagen empleada como temporal para los pixeles marcados pero no borrados.
        Vecin = Matriz con dimensiones: Renglon_Ven y Columna_Ven; utilizada para almacenar una
                region alrededor del pixel, en este caso, Renglon_Ven = 3 y Columna_Ven = 3.
        Tran = Vector de dimensión 1 × 8 donde guardamos el valor de las transiciones 0 → 1.
Img_proc_s1 = Imagen_In;        % Se asigna la imagen original a la matriz del paso 1.
Img_proc = Imagen_In;           % Se asigna la imagen original a la matriz temporal.
num_pc_del = 1;         % Número de puntos de contacto borrados
% Ahora adelgazamos la imagen.
while num_pc_del != 0 do
num_pc_del = 0;
   for (i = 1; i <= Renglon_Ini; i = i+1) do        % INICIO DEL PASO 1.
    for (j = 1; j <= Columna_Ini; j = j+1) do
     for (k = i-1; k <= i+Renglon_Ven-2; k++) do        % Ahora almacenamos una región de 3 × 3 alrededor del
      for (l = j-1; j <= j+Columna_Ven-2; l++) do       % pixel que se procesa.
       Vecin(k-i+2, l-j+2) = Img_proc_s1(k,l);
      end
     end
     if (Img_proc_s1(i,j) == 1) do          % Checamos si es un punto de contorno.
     Np = 0;
     for (k = 1; k <= Renglon_Ven; k++) do          % Ahora vemos cuantos vecinos hay con cero
     for (l = 1; l <= Columna_Ven; l++) do          % o pertenecen al fondo.
       Np = Np + Vecin(k,l);
       end
     end
     if (Np < 8) do         % Si se cumple entonces tenemos un verdadero punto de contorno.
      if ((Np >= 2) & (Np <= 6)) do         % Verificamos la primera condición.
       Tran(1,1) = Vecin(1,2) – Vecin(1,3);         % Se calculan las transiciones 0 → 1.
       Tran(1,2) = Vecin(1,3) – Vecin(2,3);
       Tran(1,3) = Vecin(2,3) – Vecin(3,3);
       Tran(1,4) = Vecin(3,3) – Vecin(3,2);
       Tran(1,5) = Vecin(3,2) – Vecin(3,1);
       Tran(1,6) = Vecin(3,1) – Vecin(2,1);
       Tran(1,7) = Vecin(2,1) – Vecin(1,1);
       Tran(1,8) = Vecin(1,1) – Vecin(1,2);
       Sp = 0;
       for (k = 1; k <= 8; k++) do
        if Tran(1,k) == –1 do
       Sp = Sp+1;
       end
      end
       if Sp == 1       % Verificamos la segunda condición.
       East = Vecin(1,2)*Vecin(2,3)*Vecin(3,2);
       if East == 0 do          % Verificamos la tercera condición del paso 1.
       South = Vecin(2,3)*Vecin(3,2)*Vecin(2,1);
       if South == 0 do         % Verificamos la cuarta condición del paso 1.
       Img_proc(i,j) = 0.5;         % Y si se han cumplido las 4 condiciones se marca el pixel
                        % en la matriz temporal para posteriormente borrarlo.
        end
       end
      end
     end
```

```
   end
  end
 end
end
num_pc_del_t1 = 0;
 for (i = 1; i <= Renglon_Ini; i = i+1) do        % Se borran todos los pixeles señalados por el paso 1.
  for (j = 1; j <= Columna_Ini; j = j+1) do
   if img_proc(i,j) == 0.5 do
     img_proc_s1(i,j) = 0;
     img_proc(i,j) = 0;
    num_pc_del_t1 = num_pc_del_t1+1;        % Se calcula el número de pixeles borrados en el paso 1.
   end
  end
 end       % FIN DEL PASO 1.
img_proc_s2 = img_proc_s1;        % Se pasa la imagen resultante del paso 1 a la matriz del paso 2.
% Y aquí inicia el algoritmo del paso 2 que es exactamente igual al algoritmo entre INICIO y FIN DEL
% PASO 1 con las siguientes excepciones; img_proc_s1 cambia a img_proc_s2, num_pc_del_t1 cambia a
% num_pc_del_t2 y las condiciones 3) y 4) se evalúan como:
for (i =1; i <= Renglon_Ini; i = i+1) do        % INICIO DEL PASO 2.
•
•
•
    if Sp == 1 do        % Verificamos la segunda condición.
    North = Vecin(1,2)*Vecin(2,3)*Vecin(2,1);
    if North == 0 do        % Verificamos la tercera condición del paso 2.
      West = Vecin(1,2)*Vecin(3,2)*Vecin(2,1);
      if West == 0        % Verificamos la cuarta condición del paso 2.
       img_proc(i, j)=0.5;        % Y si se han cumplido las 4 condiciones se marca el pixel
                    % en la matriz para posteriormente borrarlo.
      end
     end
    end
•
•
•
 end % FIN DEL PASO 2.
if num_pc_del_t1 != 0 do        % Checamos si en los pasos 1 o 2 se han borrado pixeles.
  num_pc_del = num_pc_del_t1;
else if num_pc_del_t2 != 0 do
  num_pc_del = num_pc_del_t2;
end
img_proc_s1 = img_proc_s2;        % Se pasa la imagen resultante del paso 2 a la matriz del paso 1.
end       % Fin del ciclo while.
```

6. Imágenes de superficies

Existe un conjunto de técnicas que permiten obtener arreglos volumétricos de datos (voxeles). Como ejemplo de estas técnicas podemos mencionar la tomografía computarizada (CT), resonancia magnética (MRI), tomografía por emisión de fotones (PET), ultrasonido, etc. Aun cuando los ejemplos mencionados pertenecen a técnicas usadas frecuentemente en el área de la medicina, existen también arreglos volumétricos en otras áreas de la ciencia y la tecnología.

Los arreglos volumétricos de datos se obtienen al apilar imágenes bidimensionales. La Fig. 10 muestra un conjunto de 4 rebanadas de un volumen de datos obtenidos con tomografía computarizada.

Para la visualización de los volúmenes de datos existe un conjunto de técnicas muy variadas, por ejemplo, trazo de rayos [10, 11], obtención de volúmenes (*Volume Rendering*) [12, 13], cubos alineados (*Marching cubes*) [14], modelos ópticos [15], imágenes de superficies [16], etc. La técnica más simple en su formulación matemática, y de fácil programación es la llamada imágenes de superficies. Tiene la gran ventaja de que el tiempo de cómputo es muy corto, pero es sensible al ruido y puede generar artefactos, hoyos y superficies no existentes en el volumen. Sin embargo dada su simplicidad y brevedad en el tiempo de visualización, se usa muy frecuentemente como una primera aproximación para la visualización.

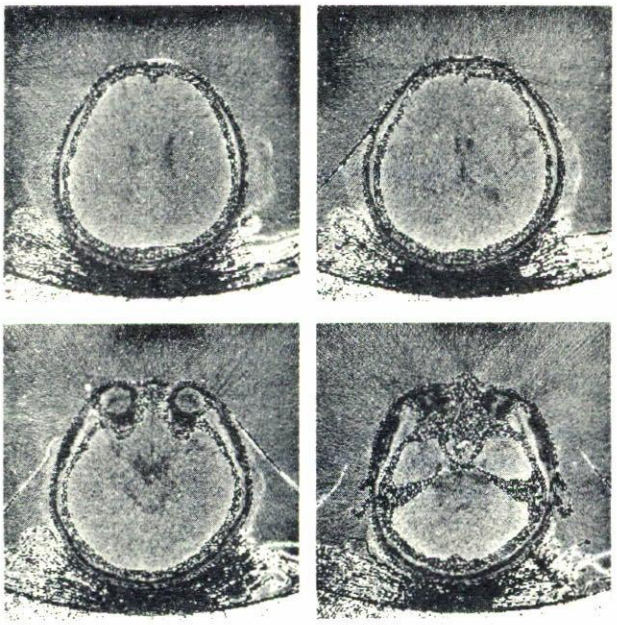


FIGURA 10. Se muestran 4 rebanadas que pertenecen a un volumen de 116. Cada rebanada consiste de 256 × 256 pixeles con una definición de 2 *bytes* por pixel. El volumen fue obtenido con tomografía computarizada tradicional.

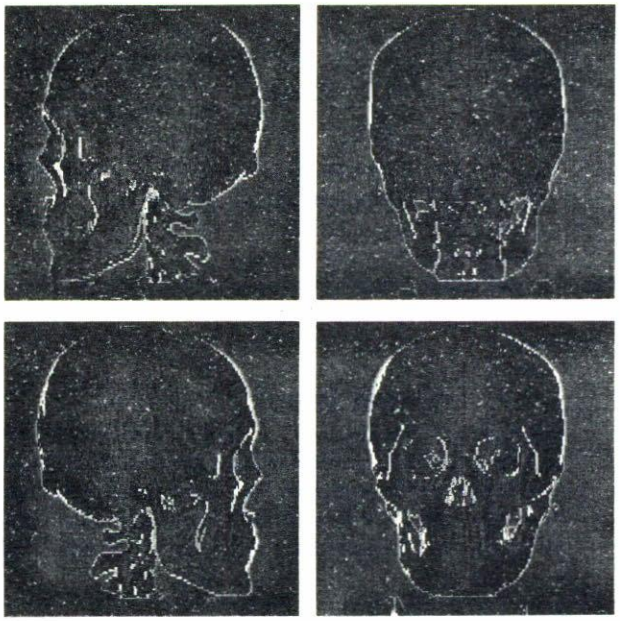


FIGURA 11. Imágenes de superficie obtenidas de un volumen de tomografía computarizada. Se presentan 4 vistas del volumen.

Las imágenes de superficies, están basadas en técnicas de umbral para la clasificación de los datos. El primer paso es hacer una clasificación binaria sobre el volumen completo, de esta manera se detecta la superficie. Posteriormente se determinan los factores de profundidad e inclinación. Finalmente se sombrea usando una combinación del factor de profundidad y el factor de inclinación. A continuación se explican brevemente cada uno de los pasos involucrados en la visualización de volúmenes de datos por medio de imágenes de superficie; con lo cual y con las diferentes técnicas que se han presentado se puede realizar, relativamente fácil, el algoritmo.

La detección de la superficie dentro del volumen es equivalente a una clasificación binaria de los datos; es decir, todo el volumen se compara contra un umbral, clasificando de esta manera si los datos pertenecen o no a la superficie. Se debe tener cuidado en la elección del umbral, ya que éste puede llegar a erosionar demasiado al volumen total de datos produciendo hoyos o superficies que no son parte del objeto a visualizar. La técnica de umbral para la detección de superficies simplifica el tiempo de cómputo y para una primera aproximación es eficiente.

El factor de profundidad es la cantidad que nos permite ver de manera diferente a dos objetos con las mismas características y cuya única diferencia es la distancia a la que se encuentran. Denotemos por F_p al factor de profundidad y escojamos un plano de referencia, el cual será el mismo para todo el volumen. Si asignamos valores de gris al factor de profundidad, observaremos que para *voxeles* que pertenecen a una superficie y se encuentran más cerca del plano de referencia, tendrán un F_p menor, por tal razón se verán más oscuros; de la misma manera los *voxeles* que se encuentran a una mayor distancia, tendrán un F_p mayor y se verán más brillantes. Esto significa, que debemos encontrar una relación inversa, pues de acuerdo a la experiencia cotidiana, entre más cercanos se encuentran los objetos, estos se ven más brillantes.

El factor de inclinación, θ, surge al tener dos elementos con el mismo F_p, pero que pertenecen a diferentes superficies; por consiguiente estos dos elementos deberán de visualizarse de manera diferente. Para poder asignar el sombreado a la superficie son necesarios y suficientes F_p y θ. El factor de inclinación se obtiene a partir del factor de profundidad usando la siguiente relación:

$$\theta = \arctan\left[\left|\frac{F_p(i+1,j) - F_p(i-1,j)}{2}\right|\right]. \qquad (4)$$

Una vez que se han determinado F_p y θ, el sombreado se asigna considerando ambos; F_p y θ, ponderados con dos constantes que son determinadas por ensayo y error:

$$Gris(i,j) = A \cdot f\left(\frac{1}{F_p}\right) + B \cdot f(\theta)\,, \qquad (5)$$

donde $f(\theta) = \sqrt{\cos\theta}$.

Y la imagen resultante será la imagen de la superficie detectada y sombreada.

En la Fig. 11 se muestran imágenes de superficies obtenidas a partir de un arreglo volumétrico. Dicho arreglo volumétrico se obtuvo con tomografía computarizada tradicional.

7. Transformada de Fourier

La transformada de Fourier es una herramienta poderosa que se usa cotidianamente en el procesamiento digital de señales, y en particular en el procesamiento digital de imágenes. Existen varios algoritmos para obtener la transformada de Fourier discreta usando instrumentos digitales, a estos algoritmos se les conoce como la transformada rápida de Fourier [17]. La mayoría de estos algoritmos obtienen la transformada de Fourier en un tiempo corto, pero tienen como requisito que el tamaño de la imagen sea una potencia de 2^n, con n entero. El programa que aquí se presenta obtiene la transformada de Fourier sin hacer optimizaciones extras, pero permite obtener el espectro de frecuencias de cualquier imagen, independientemente de su tamaño. Esta elasticidad se refleja en el tiempo de cómputo, el cual es más largo que para otro tipo de algoritmos; pero para fines educativos e ilustrativos, este algoritmo es suficiente. El algoritmo de la transformada de Fourier discreta que se empleó, se basa en la descomposición que por columnas y renglones se puede hacer; para resaltar este hecho tomemos la definición de la transformada de Fourier discreta [3]:

$$X(k_1,k_2)=\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}x(n_1,n_2)\times\exp\left[-j\frac{2\pi}{N_1N_2}(k_1n_1+k_2n_2)\right]; \quad (6)$$

reacomodando términos tenemos

$$X(k_1,k_2)=\sum_{n_2=0}^{N_2-1}\overbrace{\sum_{n_1=0}^{N_1-1}x(n_1,n_2)\exp\left[-j\frac{2\pi}{N_1}(k_1n_1)\right]}^{f(k_1,n_2)}\times\exp\left[-j\frac{2\pi}{N_2}(k_2n_2)\right]. \quad (7)$$

De la Ec. (7) podemos notar que la tranformada de Fourier discreta puede obtenerse evaluando primero $f(k_1,n_2)$ a partir de $x(n_1,n_2)$ y de aquí $X(k_1,k_2)$ a partir de $f(k_1,n_2)$. Si consideramos un valor fijo de n_2, digamos $n_2=0$; entonces $x(n_1,n_2=0)$ representa una columna de $x(n_1,n_2)$, y $f(k_1,n_2=0)$ representa la transformada de Fourier discreta en una dimensión de N_1 puntos de $x(n_1,n_2)$ con respecto a la variable n_1. De esta forma, $f(k_1,0)$ se obtiene de $x(n_1,n_2)$ calculando una transformada de Fourier discreta en una dimensión de N_1 puntos y como hay N_2 valores diferentes de n_2 en $f(k_1,n_2)$ que nos interesan entonces $f(k_1,n_2)$ se obtiene de $x(n_1,n_2)$ calculando N_2 transformadas de Fourier discretas en una dimensión de N_1 puntos.

Una vez que se obtiene $f(k_1,n_2)$, de la Ec. (7) podemos obtener $X(k_1,k_2)$ a partir de $f(k_1,n_2)$ mediante

$$X(k_1,k_2)=\sum_{n_2=0}^{N_2-1}f(k_1,n_2)\exp\left[-j\frac{2\pi}{N_2}(k_2n_2)\right]. \quad (8)$$

Para obtener $X(k_1,k_2)$ a partir de $f(k_1,n_2)$ se considera un valor fijo de k_1, digamos $k_1=0$; entonces $f(k_1=0,n_2)$ representa un renglón de $f(k_1,n_2)$, y $X(k_1=0,n_2)$ representa la transformada de Fourier discreta en una dimensión de N_2 puntos de $f(k_1=0,n_2)$ con respecto a la variable n_2. De esta forma, $X(0,k_2)$ se obtiene de $f(k_1,n_2)$ calculando una transformada de Fourier discreta en una dimensión de N_2 puntos y como hay N_1 valores diferentes de k_2 en $X(k_1,k_2)$ que nos interesan entonces $X(k_1,k_2)$ se obtiene de $f(k_1,n_2)$ calculando N_1 transformadas de Fourier discretas en una dimensión de N_2 puntos. La transformada de Fourier discreta puede obtenerse ya sea calculando primero por columnas y posteriormente por renglones o viceversa.

A continuación se presenta el algoritmo empleado en el cálculo de la transformada discreta de Fourier, cabe señalar que para manejar la función exponencial se ha empleado la fórmula de Euler y que a diferencia de la explicación anterior aquí se calcula primero la transformada de Fourier discreta por renglones y posteriormente por columnas:

```
Considerando que las dimensiones de la imagen son conocidas; sean:
    Imagen_Ini = Imagen original con dimensiones: Renglon_Ini y Columna_Ini.
    Imagen_Fin = Imagen resultante con dimensiones: Renglon_Fin y Columna_Fin.
    Matriz_Real = Imagen de la parte real de la transformada.
    Matriz_Imag = Imagen de la parte imaginaria de la transformada.
    Temp_Real = Matriz temporal para la parte real de la transformada.
    Temp_Imag = Matriz temporal para la parte imaginaria de la transformada.
% Se inicia la transformada de Fourier discreta.
    Argumento = (2*π)/Columna_Ini;
    for (i = 1; i <= Renglon_Ini; i++) do      % Primero se calcula la transformada de Fourier
    for (j = 1; j <= Columna_Ini; j++) do      % por renglones.
      REAL = 0;
      IMAG = 0;
      for (k = 1; k <= Columna_Ini; i++) do
        REAL = REAL + Imagen_Ini(i,k)*cos(Argumento*j*k);
        IMAG = IMAG + Imagen_Ini(i,k)*sin(Argumento*j*k);
        end
```

```
end
Matriz_Real(i,j) = REAL;
Matriz_Imag(i,j) = IMAG;
end
end
Argumento = (2*π)/Renglon_Ini;
for (i = 1; i <= Columna_Ini; i++) do        % Ahora se calcula la transformada de Fourier
 for (j = 1; j <= Renglon_Ini; j++) do       % por columnas.
  REAL = 0;
  IMAG = 0;
  for (k = 1; k <= Renglon_Ini; k++) do
  REAL = REAL + Matriz_Real(k,i)*cos(Argumento*j*k) + Matriz_Imag(k,i)*sin(Argumento*j*k);
  IMAG = IMAG + Matriz_Imag(k,i)*cos(Argumento*j*k) - Matriz_Real(k,i)*sin(Argumento*j*k);
  end
  Temp_Real(i,j) = REAL;
  Temp_Imag(i,j) = IMAG;
  end
end
for (i = 1; i <= Renglon_Ini; i++) do        % Finalmente obtenemos la transformada de Fourier.
 for (j = 1; j <= Columna_Ini; j++) do
  Matriz_Real(i,j) = Temp_Real(i,j);
  Matriz_Imag(i,j) = Temp_Imag(i,j);
  end
 end
```

Obteniendo la parte real y la parte imaginaria de la transformada de Fourier discreta, podemos obtener la magnitud y fase; es necesario mencionar que los valores que se obtienen de la transformada o son muy pequeños o son muy grandes, teniéndose la necesidad de desplegar los resultados no sin antes aplicarles alguna transformación; la transformacion más conocida [1, 19] consiste en un mapeo logarítmico de la siguiente forma, $\log_{10}(1 + |Imagen|)$, esto nos permite expander los valores pequeños y comprimir los valores grandes.

En la Fig. 12 se presentan una serie de cuadros y su transformada de Fourier. Estos resultados ilustran la propiedad de escalamiento que tiene la transformada de Fourier y fueron desplegados utilizando el mapeo logarítmico.

8. Desplegado de imágenes

Existen varios programas que permiten el desplegado de imágenes en las estaciones de trabajo, por ejemplo "xview" o "imagetool". Obtener alguno de estos dos programas es muy sencillo, y permiten desplegar imágenes con una gran variedad de formatos. Sin embargo, las imágenes que se usan en el procesamiento digital son imágenes sin formato, es decir son solo las matrices con los valores de los pixeles. De manera que todas las ventajas que ofrecen los programas de desplegado disponibles en el sistema UNIX no se pueden aprovechar. Para poder desplegar las imágenes que usamos en el procesamiento digital, tenemos dos opciones, la primera es que una vez terminado el proceso agregarle a la imagen resultante un formato para poder desplegarla. La segunda opción es crear un sistema que nos permita desplegar imágenes sin formato. Dada la ventaja de la segunda opción, en esta biblioteca se incluye un programa de desplegado de imágenes.

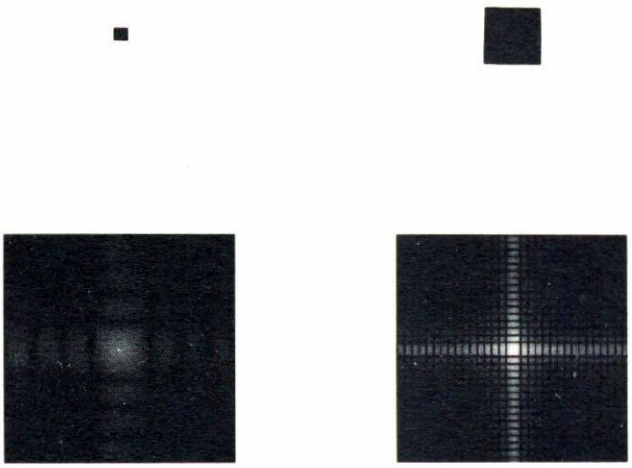


FIGURA 12. El renglón superior muestra las imágenes originales, el renglón inferior su respectiva transformada de Fourier.

Los resultados que se presentan en este trabajo son impresiones (impresora LaserJet 6MP a 600 dpi) de las imágenes que se procesaron y desplegaron en el monitor de la computadora. Las imágenes fueron desplegadas con el programa de desplegado que se incluye en esta biblioteca. Este programa puede desplegar imágenes en el monitor de una estación de trabajo [20, 21]. Las imágenes pueden ser rectangulares o cuadradas y el número de *bytes* asignados a cada pixel puede ser uno o dos. Cada nivel de gris es denotado por un entero con una asignación arbitraria de 0 para el nivel más obscuro y 255 para el más brillante. Este programa no usa formato, tiene varios parámetros que se le dan como argumentos de entrada y puede cambiar el rango dinámico en el momento

de desplegar con solo modificar los argumentos en la entrada. Dada la extensión y complejidad de los algoritmos que forman este programa no se presentan aquí.

9. Conclusiones

Esta biblioteca contiene los programas y las imágenes necesarias para poder hacer una demostración visual en la computadora sobre los conceptos básicos del procesamiento digital de imagenes. Este tipo de demostraciones visuales, causan fuerte impacto en los estudiantes dando como resultado una mejor comprensión de los conceptos.

Es necesario mencionar que los programas de ésta biblioteca están destinados para ser usados solo con imágenes monocromáticas, y están escritos en lenguaje C [18].

Debido a que los listados de los programas son muy largos no se incluyen es este texto, pero los archivos fuente así como las imágenes originales y procesadas pueden ser adquiridos directamente en el INAOE (http://www-optica.inaoep.mx/~jbaez).

---

† Becarios del CONACyT.

1. A. Rosenfeld and A.C. Kak, *Digital Picture Processing*, Vol. 1, (Academic Press, 1982).
2. K.S. Fu, R.C. Gonzalez y C.S.G. Lee, *ROBOTICA. Control, detección, visión e inteligencia*, (McGraw-Hill).
3. J.S. Lim, *Two-dimensional Signal and Image Processing*, (Prentice Hall, 1990).
4. P. Stucki, *Advances in Digital Image Processing Theory, Application, Implementation*, (Plenum Press).
5. J.C. Russ, *The Image Processing Handbook*, (CRC Press, 1992).
6. B. Jähne, *Digital Image Processing. Concept, Algorithms, and Scientific Applications*, (Springer-Verlag, 1995).
7. A.K. Jain, *Fundamentals of Digital Image Processing*, (Prentice Hall, 1989).
8. A. Rosenfield, A.C. Kak, *Digital Picture Processing*, (Academic Press, 1982).
9. C. Watkins, A. Sadun, and S. Marenka, *Modern Image Processing: Warping, Morphing and Classical Techniques*, (Academic Press Professional, 1993).
10. M. Levoy, "A Hybrid Ray Tracer for Rendering Polygon and Volume Data", *IEEE Computer Graphics & Applications*, March 1990.
11. K. Heinz Höne and R. Bernstein, "Shadding 3D images from CT Using Gray level gradientes", *IEEE Transactions on Medical Imaging*, **MI-5**, No. 1 March (1986).
12. R.A. Drebin, L. Carpenter, and P. Hanrahan, "Volume Rendering", *Computer Graphics* **22**, No. 4 August (1988).
13. D.R. Ney, E.K. Fishman, and D. Magid, "Volumetric Rendering of Computed Tomography Data: Principles and Techniques", *IEEE Computer Graphics & Applications*, March (1990).
14. W.E. Lorensen and H.E. Cline, "Marching Cubes: a High Resolution 3D Surface Construction Algorithm, *Computer Graphics* **21**, No. 24 July (1987).
15. J.J. Baez-Rojas *et al.*, *Optics Communications* **91** (1992) .
16. H. Rusinck, N. Karp, and C. Cutting, "Three-dimensional rendering of medical images: Surfaces and volume approach", *SPIE* **1091**, Medical Imaging III: Images Capture and Display (1989).
17. E. Oran Brigham, *The Fast Fourier Transform*, (Prentice Hall, 1974).
18. C.B.W. Kernighan, and D.M. Ritchie, *El lenguaje de Programación C*, (Prentice Hall, 1988).
19. R.C. Gonzalez and R.E. Woods, *Digital Image Processing*, (Addison-Wesley).
20. J. Gettis, R. Newman and R.W. Sheifler, *Xlib–C Language X Interface*.
21. O. Jones, *Introduction to the X Window System*, (Prentice Hall, 1989).