DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico III

Marche un telebeam Don Niembraaaaaa...

Métodos Numéricos
Primer Cuatrimestre - 2015

Integrante	LU	Correo electrónico

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

Resumen

En este trabajo se utilizaran distintas técnicas para obtener un re-escalamiento de imágenes. Se utilizara vecino más cercano, interpolación de polinomios bilineal, splines cúbicos, y distintas variantes de los métodos anteriormente mencionados. Se implementaran algoritmos para los mismos, dando la posibilidad de re-escalar las imágenes en distintos tamaños(siempre mayor al original). Se llevara a cabo una experimentación con su respectivo análisis. Como las imágenes obtenidas, no contienen integramente información original, se utilizaran las métricas de Error Cuadrático Medio (ECM) y Peak to Signal Noise Ratio (PSNR) para estudiar en forma cuantitativa la calidad de las mismas. También se considerara la calidad subjetiva, y el tiempo de computo.

Palabras Clave: re-escalamiento imágenes, interpolación, ECM, PSNR

1. Intoducción

Las imágenes digitales se obtienen a través de dispositivos de conversión análogico-digital como un escáner, una cámara fotográfica digital o directamente desde el ordenador utilizando cualquier programa de tratamiento de imágenes. La información digital que genera cualquiera de los medios citados es almacenada en la computadora mediante bits (unos y ceros).

Hay distintos tipos de formatos, pero a grandes rasgos se clasifican en imágenes mapa de bits e imágenes vectoriales. En general contienen una cabecera que contiene atributos (dimensiones de la imagen, tipo de codificación, etc.), seguida de los datos de la imagen en sí misma. La estructura de los atributos y de los datos de la imagen es distinto en cada formato. También puede contener metadatos, con información extra, como por ejemplo la fecha y lugar de creación en el caso de una imagen tomada con una camara digital.

Los formatos de mapa de bits representan a la imagen como una matriz de pixeles. Estos son la menor unidad homogénea en color que forma parte de una imagen. Hay que tener presente que los mismos no tienen un tamaño especifico, ya que depende del tamaño del monitor con que se visualiza la imagen, o del zoom que se aplique sobre la misma. A su vez cada formato tiene una forma especifica de representar el color de un pixel, las principales son el RGB (rojo, verde y azul), el HLS (tono, luminosidad, saturación) y el CMYK (cian, magenta, amarillo y negro), escala de grises(cada color es una tonalidad distinta del gris).

De esta manera las podemos caracterizar a las imágenes por su altura y anchura (en píxeles) y por su profundidad de color (en bits por píxel), que determina el número de colores distintos que se pueden almacenar en cada punto individual. Los principales formatos de mapas de bits son los siguientes: BMP, TIFF, XCF, PICT, JPG, GIF, PNG, PSD.

Por otra parte los formatos vectoriales representan a la imagen con objetos geométricos independientes (segmentos, polígonos, arcos, etc.), cada uno de ellos definido por distintos atributos matemáticos de forma, de posición, de color, etc. Las líneas que componen la imagen están definidas por vectores (de ahí su nombre). El interés principal de los gráficos vectoriales es poder ampliar el tamaño de una imagen a voluntad sin sufrir la pérdida de calidad que sufren los mapas de bits. De la misma forma, permiten mover, estirar y retorcer imágenes de manera relativamente sencilla. Actualmente los procesadores traducen los gráficos vectoriales a mapas de bits para poder representarlos en pantalla. El formato mas utilizado es SVG.

En este trabajo para re-escalar imágenes utilizando los distintos métodos, primero se las convertirá a escala de grises, se la expandira y luego se aplicara alguna técnica para rellenarla. Se utilizaran los métodos de vecino más cercano, interpolación de polinomios bilineal y splines cúbicos. También distintas variantes de los anteriores.

La interpolación de polinomios consiste en dada una terna de puntos (x_0, y_0), (x_1, y_1), $\dots$ (x_n, y_n), obtener un polinomio que los interpole, es decir que verifique $p(x_0) = y_0$, $p(x_1) = y_1$, $\dots$ $p(x_n) = y_n$. En general, la interpolación de una serie de puntos es usada para aproximar una función continua en un cierto intervalo. Los polinomios son funciones continuas, de clase C^∞, además es muy fácil trabajar con ellos, y suelen aproximar bien a las funciones continuas que típicamente se usan. En general se utiliza

el polinomio de menor grado que verifique la anterior condición, ya que los polinomios de grado mas grande pueden tener mayor cantidad de oscilaciones.

Se puede garantizar la existencia del polinomio interpolador, ya que hay un teorema que establece que dado un conjunto de $n+1$ puntos $\exists!$ polinomio de grado a lo sumo n que interpole. Hay distintos métodos para obtener a este ultimo. Uno de ellos es el método de interpolación de lagrange, el cual se basa en construir primero los polinomios $L_{n,k}$ definidos como se indica en la ecuación 1. [1]

$$L_{n,k} = \prod_{\substack{i=0 \\ i \neq k}} \frac{(x - x_i)}{(x_k - x_i)}^n \tag{1}$$

Estos últimos tienen grado n y se verifica que $L_{n,k}(x_i) = 0$ y que $L_{n,k}(x_k) = 1$. El polinomio de grado a lo sumo n que interpola los $n+1$ puntos se construye según la ecuación 2.

$$p(x) = \sum_{k=0}^{n} y_k L_{n,k}(x) \tag{2}$$

Por ejemplo para realizar una interpolación bilineal entre dos puntos (x_0, y_0) y (x_1, y_1), el polinomio interpolador de lagrange sera de grado a lo sumo uno, por lo será una recta que pasa por dos puntos. En la ecuación 3

$$p(x) = L_{1,0}(x)y_0 + L_{1,1}(x)y_1 \tag{3}$$

Despejando la ecuación 3 obtenemos una formula mas clara para el mismo, donde además podemos distinguir la pendiente y la ordenada al origen.

$$p(x) = \frac{y_1 - y_0}{x_1 - x_0}x - x_0\frac{y_1 - y_0}{x_1 - x_0} + y_0 \tag{4}$$

Otro de los métodos utilizados en este trabajo es el de splines cúbicos. Dada una función f definida en $[a, b]$ y un conjunto de nodos $a = x_0 < x_1 < \ldots < x_n = b$ un trazador cúbico S para f es una función tal que en cada subintervalo $[x_j, x_{j+1}]$ con $j = 0, 1, \ldots, n-1$ es un polinomio cubico $S_j(x)$, y que verifica las siguientes condiciones:[1]

- $S(x_j) = f(x_j) \; \forall \; j = 0, 1, \ldots, n$
- $S_{j+1}(x_{j+1}) = S_j(x_{j+1}) \; \forall \; j = 0, 1, \ldots, n-2$
- $S'_{j+1}(x_{j+1}) = S'_j(x_{j+1}) \; \forall \; j = 0, 1, \ldots, n-2$
- $S''_{j+1}(x_{j+1}) = S''_j(x_{j+1}) \; \forall \; j = 0, 1, \ldots, n-2$
- Alguna de las siguientes condiciones
  - $S''(x_0) = S''(x_n) = 0 \; \forall \; j = 0, 1, \ldots, n-2$ (condicion natural)
  - $S'(x_0) = f'(x_0) y S'(x_n) = f'(x_n)$ (condicion sujeta)

Escribiendo a los S_j en la forma $S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$, y planteando las condiciones anteriormente mencionadas se puede obtener un sistema lineal de $n+1$ ecuaciones y $n+1$ incógnitas donde estas son los c_j. Además una vez determinadas estas últimas, se puede obtener en forma univoca los a_j,b_j y d_j. En ambos casos queda un sistema lineal cuadrado $Ac = b$ donde A es una matriz estrictamente diagonal dominante. Esto ultimo implica que la matriz es inversible y por lo tanto el sistema tiene solución única. En el caso de la condición natural, que fue el utilizado en este trabajo práctico, mas especificamente se obtiene:

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ h_0 & 2(h_0+h_1) & h_1 & \ddots & & & \vdots \\ 0 & h_1 & 2(h_1+h_2) & h_2 & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & \cdots & \cdots & \cdots & 0 & 0 & 1 \end{bmatrix}$$

$$b = \begin{bmatrix} 0 \\ \frac{3}{h_1}(a_2-a_1) - \frac{3}{h_0}(a_1-a_0) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n-a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1}-a_{n-2}) \\ 0 \end{bmatrix}$$

$$c = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

donde $h_j = x_{j+1} - x_j$.

2. Desarollo

En este trabajo practico se aplicaran distintos métodos para re-escalar una imagen, es decir obtener una imagen ïgual", pero con una cantidad de pixeles mayor. Para esto en todos los casos, se ejecutara desde el programa en C++ un script de matlab que dada una imagen en cualquier formato, obtenga un archivo".csv" con la matriz que representa esa imagen convertida a escala de grises. Luego se utilizara el mismo para aplicarle los métodos para re-escalarla, obteniendo un archivo ".csv" con la imagen final, y nuevamente se llamara a un script de matlab para obtener una imagen en formato BMP en blanco y negro.

Lo primero que se aplicara a la matriz de la imagen de entrada en escala de grises es aumentar su tamaño según un parámetro $k \in \mathbb{N}_{>0}$ que indica la cantidad de filas y columnas que seran insertadas entre cada par de puntos consecutivos, tal como se puede ver en la figura 5. Estas nuevas filas y columnas seran rellenadas provisoriamente con -1.

1	2	3
4	5	6
7	8	9

(a) Imagen original

1	-1	-1	2	-1	-1	3
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
4	-1	-1	5	-1	-1	6
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
7	-1	-1	8	-1	-1	9

(b) Imagen expandida

Figura 1: Expansión de una imagen para un k de 3.

Luego se aplicaran distintos métodos para rellenar la imagen.

2.1. Vecino mas cercano

Se llevaron a cabo tres versiones de este método. La original consiste en recorrer la matriz expandida sustituyendo en cada posición los -1 por el valor de la matriz original mas cercano. Se utiliza una función auxiliar que para cada posición nos devuelve el vecino mas cercano. Hay que notar que se puede dar el caso de que halla dos vecinos mas cercanos, en este caso el algoritmo implementado tomara alguno de ellos.

Una segunda versión considera no solo los valores originales como los mas cercanos, sino que en cada paso considera también los valores que ya fueron completados. Para esto es importante el orden en que es completada la matriz, para este trabajo es completada por filas de izquierda a derecha, de arriba hacia abajo. En este caso el vecino mas cercano resulta ser el elemento de la izquierda o el de arriba, es por esto que el algoritmo se simplifica bastante. En el siguiente fragmento podemos encontrar el código del algoritmo efectivamente implementado.

```
1 void porFil(vector<vector<int> > &expandida, int k)
2 {
```

```
3     for(int i = 0; i < expandida.size(); i++)
4     {
5       for(int j = 0; j < expandida[i].size() ; j++)
6       {
7          if(expandida[i][j] == -1)
8           {
9             if(j%(k+1)== 0) {expandida[i][j] = expandida[i-1][j];}
10            else {expandida[i][j] = expandida[i][j-1];}
11          }
12
13      }
14    }
15 }
```

Una tercera versión calcula los promedios...

2.2. Splines

Implementamos dos variantes distintas del método de interpolación por medio de splines. Antes de entrar en detalle sobre las diferencias que tiene cada una, vamos a explicar algunos aspectos fundamentales que comparten ambas implementaciones. En los dos casos vamos a trabajar con bloques de píxeles de la imagen expandida (inicialmente con valores -1 en los píxeles nuevos) y a cada uno de estos bloques los vamos a tratar como si fueran imágenes independientes, calculando los splines correspondientes y pudiendo así hallar los valores de los píxeles nuevos.

Una vez definido el bloque de la imagen con el que se va a trabajar, la idea va a ser recorrer las *filas* del bloque que poseen píxeles de la imagen original (es decir aquellas filas que no todos sus píxeles tienen el valor -1). Supongamos que trabajamos con el siguiente bloque:

P_{00}	-1	-1	P_{03}
-1	-1	-1	-1
-1	-1	-1	-1
P_{30}	-1	-1	P_{33}

(a) Bloque

Figura 2: Bloque de imagen expandida con los valores de los píxeles originales. Indexamos desde 0.

Donde P_{00}, P_{03}, P_{30} y P_{33} son píxeles de la imagen original, cuyos valores ya conocemos. Lo que vamos a hacer es empezar tomando la primera fila (que no todos sus píxeles son -1) y calcularemos el spline correspondiente a los puntos $(0, P_{00})$ y $(3, P_{03})$ ya que 0 y 3 son las coordenadas de los píxeles originales en esta fila y P_{00} y P_{03} sus respectivos valores. Una vez que tenemos dicho spline, podemos utilizarlo para hallar los valores del segundo y tercer píxel de esta fila, luego de hacer esto la primera fila ya tendrá valores válidos en sus píxeles. Repetimos el mismo proceso pero ahora para la última fila y ahora con los puntos $(0, P_{30})$ y $(3, P_{33})$ (observemos que si tuvieramos un bloque de tamaño más grande, tendríamos que repetir este proceso tantas veces como filas no agregadas existan). Para hallar los splines definimos los intervalos según las coordenadas de los píxeles conocidos (en este caso como hay solo dos entonces va a haber un solo intervalo) y utilizamos el algoritmo descripto en [1] para hallar el polinomio cúbico por cada intervalo. Nuestra imagen expandida quedaría de la siguiente manera:

P_{00}	P_{01}	P_{02}	P_{03}
-1	-1	-1	-1
-1	-1	-1	-1
P_{30}	P_{31}	P_{32}	P_{33}

(a) Bloque

Figura 3: Bloque de imagen expandida luego de hacer interpolación en la primera y última fila.

Ahora solo quedaría calcular los valores de los píxeles de las filas agregadas. Si observar con atención la Figura 3 vemos que podemos usar la misma idea pero ahora trabajando con las columnas. Es decir, al principio vimos que la primera y última filas tenían píxeles originales en los extremos que pudimos usar para interpolar y hallar los valores de los píxeles nuevos. Ahora podemos hacer exactamente lo mismo, pero generando un spline por columna y así pudiendo calcular los valores de los píxeles intermedios.
A continuación mostramos graficamente la secuencia de pasos que realiza el algoritmo para calcular los píxeles restantes.

P_{00}	P_{01}	P_{02}	P_{03}
P_{10}	-1	-1	-1
P_{20}	-1	-1	-1
P_{30}	P_{31}	P_{32}	P_{33}

(a) Spline para columna 1.

P_{00}	P_{01}	P_{02}	P_{03}
P_{10}	P_{11}	-1	-1
P_{20}	P_{21}	-1	-1
P_{30}	P_{31}	P_{32}	P_{33}

(b) Spline para columna 2.

P_{00}	P_{01}	P_{02}	P_{03}
P_{10}	P_{11}	P_{12}	-1
P_{20}	P_{21}	P_{22}	-1
P_{30}	P_{31}	P_{32}	P_{33}

(c) Spline para columna 3.

P_{00}	P_{01}	P_{02}	P_{03}
P_{10}	P_{11}	P_{12}	P_{13}
P_{20}	P_{21}	P_{22}	P_{23}
P_{30}	P_{31}	P_{32}	P_{33}

(d) Spline para columna 4.

Figura 4

En resumen lo que hacemos entonces es lo siguiente:
Sea B el bloque de la imagen expandida, T es el tamaño de dicho bloque y k la cantidad de píxeles a agregar entre cada par de píxeles originales, entonces

1. Para las filas $F(B)_0$, $F(B)_{k+1}$, $F(B)_{2(k+1)}$, ..., $F(B)_{T-1}$:
   a) Calcular S_0, S_1, ..., S_{T-1} utilizando los valores de los píxeles originales de cada una de las filas
   b) Utilizar S_0 para reemplazar los píxeles con -1 de la primera fila, S_1 para los de la fila $k+1$, ..., S_{T-1} para los de la fila $T-1$.
2. Para las columnas $C(B)_0$, $C(B)_1$, $C(B)_2$, ..., $C(B)_{T-1}$:
   a) Calcular S'_0, S'_1, ..., S'_{T-1} utilizando los valores de los píxeles originales de cada una de las columnas (algunos van a ser los originales de la imagen, otros los que calculamos en el paso anterior)
   b) Utilizar S'_0 para reemplazar los píxeles con -1 de la primera columna, S'_1 para los de la segunda, ..., S'_{T-1} para los de la $T-1$.

Un algoritmo equivalente podría ser primero trabajar con las columnas 0, $k+1$, $2(k+1)$, ..., $T-1$. Calcular los respectivos splines, utilizarlos para hallar los valores de los píxeles intermedios de cada una

de estas columnas y luego trabajar con las filas 0, 1, 2, ..., $T-1$. Esto lo podemos hacer debido a que entre cada par de píxeles originales contiguos se agrega siempre la misma cantidad de píxeles nuevos, independientemente si los originales estan en la misma fila o en la misma columna.

Ahora que ya sabemos como trabaja el algoritmo con un bloque particular de la imagen expandida, vamos a ver que hacemos cuando tenemos la imagen entera. Para esto como dijimos al principio existen dos variantes: la primera es procesar la imagen de a bloques de un tamaño fijo, la segunda es ir variando el tamaño del bloque de acuerdo a ciertos criterios. Vamos a comenzar viendo la primera que es la más sencilla.

Si tuvieramos una imagen de 512x512 por ejemplo y quisiéramos utilizar interpolación por splines para agrandarla dado un cierto k, como ya tenemos el algoritmo que procesa bloques de la imagen, una opción podría ser tomar un bloque de 512x512 que comience en el primer píxel de la primera fila y aplicar dicho algoritmo. De esta manera, al finalizar vamos a tener la imagen correctamente (sin píxeles cuyo valor sea -1) expandida. Sin embargo, quizás tenemos una imagen donde existe algún patrón definido en el cambio de tonalidad. Es decir, quizás podemos dividir la imagen en distintas partes del mismo tamaño y, al observar estas partes nos damos cuenta que podríamos expandir una independientemente de la otra, para hacer esto la idea es simplemente ir moviendonos sobre los bloques de la imagen expandida e ir aplicando el algoritmo sobre cada uno de ellos. Veamoslo con el ejemplo de la Figura 1 b) y tomemos un tamaño de bloque igual a 4 con respecto a la imagen expandida (los bloques siempre son cuadrados):

1	-1	-1	2	-1	-1	3
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
4	-1	-1	5	-1	-1	6
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
7	-1	-1	8	-1	-1	9

(a) Imagen expandida

1	-1	-1	2
-1	-1	-1	-1
-1	-1	-1	-1
4	-1	-1	5

(b) Primer bloque

2	-1	-1	3
-1	-1	-1	-1
-1	-1	-1	-1
5	-1	-1	6

(c) Segundo bloque

4	-1	-1	5
-1	-1	-1	-1
-1	-1	-1	-1
7	-1	-1	8

(d) Tercer bloque

5	-1	-1	6
-1	-1	-1	-1
-1	-1	-1	-1
8	-1	-1	9

(e) Cuarto bloque

Figura 5: Imagen expandida partida en bloques de tamaño 4.

Al observar las imágenes, nos puede llamar la atención el hecho de que cuando dos bloques están uno seguido del otro, la primera columna del segundo es la última del primero, al igual que cuando dos bloques están uno debajo del otro, la primera fila del que está abajo es la última del que está arriba.

Uno esperaría que los bloques sean disjuntos, sin embargo, si así fuera entonces no podríamos definir correctamente los polinomios cúbicos al construir los splines porque por ejemplo el segundo bloque de la Figura 5 comenzaría con una columna de píxeles donde todos valen -1, con lo cual el extremo izquierdo correspondiente a este intervalo no estaría definido. Esto nos sugiere que además, es necesario que la última columna de cada bloque tenga valores de los píxeles originales, es decir sea un múltiplo de $k + 1$.

Una dificultad con la que nos podemos encontrar es que, al ir moviendonos en bloques del tamaño fijado sobre la imagen expandida, suceda que nos pasemos de las dimensiones de dicha imagen. Cuando eso suceda lo que va a hacer el algoritmo es simplemente "retroceder" cierta cantidad de píxeles para formar un bloque del tamaño esperado. Más especificamente cuando hablamos de retroceder píxeles nos referimos a que si estamos queriendo procesar un bloque que supera la cantidad de columnas de la imagen expandida, entonces nos movemos horizontalmente en la imagen tantos píxeles como la diferencia entre la última columna del bloque y el ancho de la imagen. Si nos estamos pasando en la cantidad de filas de la imagen entonces nos movemos verticalmente tantos píxeles como la diferencia entre la última fila del bloque y el alto de la imagen.

2.3. Interpolación Bilineal

3. Experimentación

3.1. Calidad cuantificable

Explicar las metricas, y que nosotros para experimentar reducimos la imagen, y como lo hicimos con matlab.

3.2. Vecino mas cerano

Los graficos y discusion de los resultados para este metodo particular

3.3. Splines

Los graficos y discusion de los resultados para este metodo particular

3.4. Bilineal

Los graficos y discusion de los resultados para este metodo particular

3.5. Comparación métodos

Comparacion entre los distintos metodos, cual tarda mas tiempo, cual tiene mayor calidad

4. Conclusiones

Referencias

[1] Burden R. L., Faires J.D.- Numerical Analysis