



DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico III

Marche un telebeam Don Niembraaaaa...

Métodos Numéricos
Primer Cuatrimestre - 2015

Integrante	LU	Correo electrónico
Germán Pinzón	475/13	pinzon.german.94@gmail.com
Angel More	931/12	angel_21_fer@hotmail.com
Julián Armagno	377/12	julian.armagno@gmail.com
Jorge Porto	376/11	cuanto.p.p@gmail.com



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>

Resumen

En este trabajo se utilizarán distintas técnicas para obtener re-escalamiento de imágenes. Implementaremos vecino más cercano, interpolación bilineal, splines cúbicos, y distintas variantes de los métodos anteriormente mencionados. Se llevará a cabo una experimentación con su respectivo análisis tanto cuantitativo como cualitativo. Para el análisis cuantitativo utilizaremos las métricas de Error Cuadrático Medio (ECM) y Peak to Signal Noise Ratio (PSNR) para estudiar el impacto de las diferencias entre las imágenes originales y las resultantes. El análisis cualitativo será realizado observando las diferencias más notables entre las imágenes resultantes y originales y también entre las imágenes que devuelven distintas variantes de un mismo método.

Palabras Clave: zoom, imagen, interpolación, Lagrange, ECM, PSNR

Índice

1. Introducción	1
2. Desarrollo	3
2.1. Vecino mas cercano	3
2.2. Interpolación Bilineal	4
2.3. Splines	6
2.3.1. Bloques fijos	6
2.3.2. Bloques variables	9
3. Experimentación	10
3.1. Vecino Mas Cercano	11
3.2. Bilineal	16
3.3. Splines	23
3.3.1. Bloques fijos	23
3.3.2. Bloques variables	34
3.3.3. Tiempo de cómputo	46
3.3.4. Comparación de variantes	47
3.4. Comparando los 3 metodos	49
4. Conclusiones	51
5. Apéndices	52

1. Introducción

En el presente trabajo se nos plantea como objetivo el re-escalamiento de imágenes, específicamente ampliarlas.

Se trabajará con imágenes en escala de grises por lo que dada una imagen de $m \times n$, contendrá $m \times n$ pixels cada uno con un valor entre 0-255.

Para ampliar las imágenes, a partir de un valor $k \in \mathbb{N}_{>0}$ insertaremos entre la fila i y la fila $i + 1$, k filas con $i = 1, \dots, m - 1$, y de manera análoga para las columnas. De esta forma obtendremos una nueva imagen con $(m - 1) * k + m$ filas y $(n - 1) * k + n$ columnas.

El problema que ahora se genera es ¿que valores asignar a los pixels de las columnas y filas agregadas?. Para esto, emplearemos distintos criterios de asignación de valor a partir de ciertos métodos.

En primera instancia consideraremos el método de vecino más cercano, para esto el valor de un píxel nuevo será igual a aquel cuya distancia a otro píxel, en una vecindad definida, sea la mínima. En particular la vecindad tomada será de aquellos cuatro valores más cercano con respecto a los pixels originales (pixels de la imagen sin ampliar, *imagen original*).

Otro método empleado fue mediante la interpolación de polinomios, esta consiste en que dada un conjunto de puntos $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$ se busca obtener un polinomio P 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. Dado que siempre existe un polinomio interpolador para $n + 1$ puntos, de grado a lo sumo n que los interpole¹, una de la forma de obtenerlo es mediante 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.

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

El polinomio de grado a lo sumo n que interpola los $n + 1$ puntos se construye según la ecuación 2.

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

Por lo que el segundo método empleado consiste en usar interpolación bilineal entre dos puntos (x_0, y_0) y (x_1, y_1) , en nuestro caso entre dos pixels, por lo que el polinomio interpolador de Lagrange será de grado a lo sumo uno es decir, será una recta que pasa por dos puntos. En la ecuación 3

$$P(x) = L_{1,0}(x)y_0 + L_{1,1}(x)y_1 = \frac{x - x_1}{x_0 - x_1}y_0 + \frac{x - x_0}{x_1 - x_0}y_1 \quad (3)$$

o equivalentemente obtenemos una fórmula mas clara para el mismo, donde además podemos distinguir la pendiente y la ordenada al origen.

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

Ses $f(x_i) = y_i$, en nuestro caso el valor de f depende de dos puntos, por lo que el valor del píxel p_{ij} se obtendrá extendiendo la ecuación 4 a:

$$p(i, j) = \frac{f(i, j_1) - f(i, j_0)}{j_1 - j_0} + f(i, j_0) \quad (5)$$

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

- $S(x_j) = f(x_j) \forall j = 0, 1, \dots, n$
- $S_{j+1}(x_{j+1}) = S_j(x_{j+1}) \forall j = 0, 1, \dots, n - 2$

- $S'_{j+1}(x_{j+1}) = S'_j(x_{j+1}) \forall j = 0, 1, \dots, n-2$
- $S''_{j+1}(x_{j+1}) = S''_j(x_{j+1}) \forall j = 0, 1, \dots, n-2$
- Y una de las siguientes condiciones
 - $S''(x_0) = S''(x_n) = 0 \forall j = 0, 1, \dots, n-2$ (condición natural)
 - $S'(x_0) = f'(x_0)yS'(x_n) = f'(x_n)$ (condición 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 . En el caso de la condición natural, que fue el utilizado en este trabajo práctico, mas específicamente se obtiene:

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

$$b = \begin{bmatrix} 0 \\ \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \\ 0 \end{bmatrix}; \quad c = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

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

donde A es una matriz estrictamente diagonal dominante. Esto último implica que la matriz es invertible y por lo tanto el sistema tiene solución única. Una vez determinado c , se pueden obtener los a_j, b_j y d_j .

A partir de los resultados obtenidos en cada método buscaremos introducir alguna modificación en los mismos con el fin de obtener alguna mejora temporal y/o cualitativa. La forma en que se medirá la calidad de la imagen obtenida será a través de el *Error Cuadrático Medio* (ECM) y *Peak to Signal Noise Ratio* (PSNR) los mismos se definen como:

$$\text{ECM}(I, \bar{I}) = \frac{1}{mn} \sum_{i=1}^m \sum_{j=1}^n |I_{ij} - \bar{I}_{ij}|^2 \quad (6)$$

y

$$\text{PSNR}(I, \bar{I}) = 10 \log_{10} \left(\frac{255^2}{\text{ECM}(I, \bar{I})} \right). \quad (7)$$

Con I e \bar{I} la imagen original y la ampliada respectivamente, de dimensiones mxn. Como para utilizar esta métrica es necesario que las imágenes tengan igual dimensiones, aquellas con las que trabajaremos serán reducidas y luego ampliadas, con los métodos con los que trabajemos, a su tamaño original.

2. Desarrollo

En este trabajo práctico se aplicarán distintos métodos para re-escalar una imagen, es decir obtener una imagen igual, pero con una cantidad de pixeles mayor. Para esto en todos los casos, se ejecutará 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 utilizará el mismo para aplicarle los métodos para re-escalarla, obteniendo un archivo ".csv" con la imagen final, y nuevamente se llamará a un script de matlab para obtener una imagen en formato TIFF en blanco y negro.

Lo primero que se aplicará 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 serán insertadas entre cada par de puntos consecutivos, tal como se puede ver en la figura 9. Estas nuevas filas y columnas serán llenadas provisoriamente con -1 .

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

1	2	3
4	5	6
7	8	9

(a) Imagen original

(b) Imagen expandida

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

Luego se aplicarán distintos métodos para llenar 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 más cercano. Se utiliza una función auxiliar que para cada posición nos devuelve el vecino más cercano. Hay que notar que se puede dar el caso de que halla dos vecinos más cercanos, en este caso el algoritmo implementado tomará alguno de ellos.

Una segunda versión considera no solo los valores originales como los más cercanos, sino que en cada paso considera también los valores que ya fueron completados, es decir es una versión dinámica del anterior método. 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 más cercano resulta ser el elemento de la izquierda o el de arriba, es por esto que el algoritmo se simplifica bastante. Para evitar el aglomeramiento de un número particular, es decir que un mismo número se repita en toda una fila, y en las siguientes de abajo, se decidió que en las columnas múltiplos de $k + 1$ se tome como más cercano al elemento de arriba, y en los demás casos al de la izquierda.

En el siguiente fragmento podemos encontrar el seudo-código del algoritmo efectivamente implementado.

Algoritmo 1 vecinoMasCercano(expandida, k)

```

1: for  $i \leftarrow [0 : cantidad\_filas]$  do
2:   for  $j \leftarrow [0 : cantidad\_columnas]$  do
3:     if  $expandida[i][j] == -1$  then
4:       if  $j \bmod (k + 1) == 0$  then
5:          $expandida[i][j] \leftarrow expandida[i - 1][j]$ 
6:       else
7:          $expandida[i][j] \leftarrow expandida[i][j - 1]$ 
8:       end if
9:     end if
10:   end for
11: end for

```

También se implementó una tercera versión como una variación de la anterior, es decir completando la matriz de izquierda a derecha de arriba hacia abajo, y considerar como posibles más cercanos no solo a los originales, sino también a los elementos completados hasta el momento, pero que en lugar de decidir con algún criterio cual elemento tomar como más cercano, tomamos un promedio de todos ellos.

2.2. Interpolación Bilineal

La interpolación bilineal intuye que el valor correcto de cada pixel es un promedio de sus valores más cercanos, a priori, este método tendría que ser un poco más completo que vecino más cercano. En primera instancia, dentro del contexto del presente trabajo, a la interpolación bilineal la consideramos como una técnica que consiste en llenar los píxeles utilizando interpolaciones lineales entre píxeles consecutivos de la imagen original, primero completando aquellas posiciones correspondientes a filas, es decir, completando de a k filas y luego sobre la matriz resultante, completando aquellas posiciones correspondientes a columnas.

Por ejemplo tomando un fracción de 2x2 de una imagen, obtenemos lo siguiente:

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

(a) Bloque

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

Luego interpolamos linealmente por filas, de a k filas, obteniendo:

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: Fracción de imagen expandida luego de interpolar por filas, de a k filas.

Finalmente, a la matriz resultante la interpolamos linealmente por columnas, todas las columnas, obteniendo:

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

(a) Bloque

Figura 4: Fracción de imagen expandida luego de interpolar por columnas, todas las columnas.

El polinomio de grado 1 para interpolar linealmente, lo definimos de la siguiente manera, suponiendo X_1 y X_2 pixeles originales, y X un pixel a llenar:

$$f(X) = f(X_1) + \frac{f(X_2) - f(X_1)}{(X_2 - X_1)}(X - X_1) \quad (8)$$

Luego, ideamos distintas variantes de este método.

En primer lugar, pensamos que pasaría si en vez de interpolar con píxeles originales consecutivos, interpolemos ignorando un pixel, es decir, sin utilizar toda la información original de la imagen. Este razonamiento se produjo al pensar que pasaría en las imágenes donde no haya tanta variación de colores(en este caso, variaciones de tonos de grises).

Como la interpretación del método de interpolación bilineal es la denominación de cada pixel como un promedio de sus valores más cercanos, también pensamos que sería interesante interpolar por diagonales, en los sectores de la imagen que lo permitiera. Estos sectores quedan limitados en el centro de la imagen ya que en las esquinas no hay la cantidad mínima de píxeles para interpolar(2 o mas píxeles). Ante esto, procedemos a usar el metodo bilineal original, para completar la imagen.

Por último también verificaremos que sucedería al interpolar bilinealmente de a bloques, es decir tomando fracciones de la imagen, en donde en las puntas están los píxeles originales, y los demás serán calculados en base a estos 4 píxeles, siguiendo la siguiente ecuación:

Sean $Q_{11} = (X_1, Y_1)$, $Q_{12} = (X_1, Y_2)$, $Q_{21} = (X_2, Y_1)$, $Q_{22} = (X_2, Y_2)$, $R_1 = (X, Y_1)$, $R_2 = (X, Y_2)$ y $P = (X, Y)$, donde Q_{11} , Q_{12} , Q_{21} y Q_{22} son los 4 píxeles originales, P el pixel a llenar y R_1 , R_2 las proyecciones ortogonales de P a las rectas Y_1 y Y_2 respectivamente:

$$f(R_1) = \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \quad (9)$$

$$f(R_2) = \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \quad (10)$$

$$f(P) = \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2) \quad (11)$$

Q_{11}	R_1	-1	Q_{21}
-1	P	-1	-1
-1	-1	-1	-1
Q_{12}	R_2	-1	Q_{22}

(a) Bloque

Figura 5: Fracción de imagen expandida donde se muestra el ejemplo de interpolacion bilineal por bloques.

Ante las distintas variantes del método en estudio, promovemos que en cuanto a la variante de ignorar un pixel en el método de interpolación bilineal, si lo usamos para imágenes donde no haya grandes cambios de colores seguidos, el mismo debe devolvernos similares resultados (Objetivos y subjetivos) que las otras 3 variantes del método. Y para imágenes donde haya gran cantidad de cambios de color la variante debería dar peor que las otras 3.

En cuanto a las variantes de interpolar por diagonales y por bloques, deben dar similares resultados (Objetivos y subjetivos) que el método original, sin importa que imagen usamos, debido a que en la implementación se utilizan cuentas similares. Aunque, ya que en la variante original se realizan menos casteos de las variables, esperamos que el resultado sea levemente mejor en calidad objetiva que el de la variante por bloques.

2.3. Splines

2.3.1. Bloques fijos

Implementamos dos variantes distintas del método de interpolación por medio de splines: 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. En los dos casos vamos a trabajar con bloques de píxeles de la imagen expandida (initialmente con valores -1 en los píxeles nuevos), calculando los splines correspondientes y pudiendo así hallar los valores de los píxeles nuevos. Empezaremos explicando la primera implementación.

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 6: 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 7: 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_{10}	P_{11}	-1	-1	P_{10}	P_{11}	P_{12}	-1	P_{10}	P_{11}	P_{12}	P_{13}
P_{20}	-1	-1	-1	P_{20}	P_{21}	-1	-1	P_{20}	P_{21}	P_{22}	-1	P_{20}	P_{21}	P_{22}	P_{23}
P_{30}	P_{31}	P_{32}	P_{33}												

(a) Spline para columna 1.

(b) Spline para columna 2.

(c) Spline para columna 3.

(d) Spline para columna 4.

Figura 8

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 $F(B)_0, F(B)_{k+1}, F(B)_{2(k+1)}, \dots, F(B)_{T-1}$:
 - a) Calcular S_0, S_1, \dots, 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 $k + 1, \dots, S_{T-1}$ para los de la fila $T - 1$.
2. Para las columnas $C(B)_0, C(B)_1, C(B)_2, \dots, C(B)_{T-1}$:
 - a) Calcular $S'_0, S'_1, \dots, S'_{T-1}$ 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 $0, k + 1, 2(k + 1), \dots, T - 1$. 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 $0, 1, 2, \dots, T - 1$. 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 están 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. Si tuvieramos una imagen de 512x512 por ejemplo y quisieramos 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 moviéndonos sobre los bloques de la imagen expandida e ir aplicando el algoritmo sobre cada uno de ellos. Veámoslo 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 9: 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 moviéndonos en bloques del tamaño fijo sobre la imagen expandida, suceda que nos pasemos de las dimensiones de dicha imagen. Cuando eso sucede lo que va a hacer el algoritmo es simplemente “retroceder” cierta cantidad de píxeles para formar un bloque del tamaño esperado. Más específicamente 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.

2.3.2. Bloques variables

Para el caso anterior el tamaño de los bloques era fijo, si teníamos dos puntos conocidos $P_{i,j}$ y $P_{i,j+1}$, dentro del bloque con el que trabajamos, los valores intermedios serían calculados creando un spline

por fila para estos puntos. Pero, ¿Que sucede si el valor de ambos difieren de manera considerable? Esto podría implicar que uno es muy claro y el otro no luego, al momento de ampliar la imagen aquellos nuevos pixels que se encuentren entre los $P_{i,j}$ y $P_{i,j+1}$ se verán afectados por estos dos, pudiendo tener un valor intermedio a estos.

Supongamos que ambos puntos representaban dos objetos distintos o se trataba de sus de bordes, entonces en la imagen ampliada nos gustaría que aun siga existiendo esta distinción, por ejemplo si se pasa de un color blanco a otro negro no nos gustaría que los pixels intermedio queden en alguna otra tonalidad. Con los splines cúbicos no siempre se puede evitar esto, aun cuando el tamaño del bloque no sea el de toda la imagen deberíamos tomar bloques muy pequeños para poder evitarlo. Por lo que para solucionar esto propusimos que el tamaño de un bloque, sea al comenzar, desde el primer punto conocido de la fila $P_{i,0}$ con la que se trabaja, hasta el punto anterior a un $P_{i,t}$ tal que $|f(P_{i,0}) - f(P_{i,t})| \geq \alpha$ (siendo f la función que retorna el valor de un pixel p) con a un valor determinado. Luego, con la misma idea el bloque se tomara desde $P_{i,t}$ hasta un $P_{i,t+r}$ o hasta el final de la fila. Notemos que al igual que en el caso anterior una vez que se completan las filas podemos aplicar la misma idea a las columnas y así obtener completar los valores.

Pero entonces al utilizar spline cúbicos para interpolar por ejemplo desde $P_{i,0}$ a $P_{i,t-(k+1)}$ y otro para $P_{i,t-(k+1)}$ a $P_{i,t+r}$ los valores entre $P_{i,t-(k+1)}$ y $P_{i,t}$ aun no tienen ningún valor asignado, como queremos que en la imagen ampliada estos sean exactamente igual a $P_{i,t-(k+1)}$ o a $P_{i,t}$ y no un valor intermedio entre estos. Optaremos por completar a los mismos siguiendo la idea de vecino mas cercano definiendo como vecindad a los valores originales mas cercano.

Para este caso el tamaño de los bloques a tomar esta ligado a que valor de α a utilizar, notemos que si usamos valores muy chicos, es decir permitimos una diferencia mínima, posiblemente terminemos aplicando el método de vecino mas cercano en casi toda la imagen y perdamos la aplicación de spline. Mientras que por el contrario, si permitimos diferencias muy grandes entonces caeríamos en la implementación anterior (splines con bloques fijos). Para comprobar si esto sucede experimentaremos con varios valores y veremos si existe algún valor o rango de valores óptimos el cual hace reducir el ECM y si estas variaciones influyen o no en el tiempo de computo.

3. Experimentación

Para poder tener una idea de como se comportan las distintas implementaciones, hemos seleccionado un conjunto de imágenes (cada una con su particularidad que será explicada brevemente en la sección correspondiente) por cada método. De esta manera podemos ver, para ese conjunto, cual es la mejor variante en términos tanto cuantitativos (PSNR) como cualitativos (analizando visualmente como quedan las imágenes, artifacts y demás). Si bien esto nos da información sobre las variantes de cada método, se trata de una experimentación bastante aislada en el sentido de que permite comparar variantes de un mismo método y nunca relaciona un método con otro. Es por eso que decidimos, una vez terminada esta etapa, realizar una nueva experimentación seleccionando un nuevo conjunto de imágenes y probando con ellas la mejor variante de cada método, de esta manera podemos tener un candidato a la mejor implementación y vemos como se comportan distintos métodos.

El tiempo de cómputo está presentado en segundos y fue tomado sin tener en cuenta las operaciones realizadas en MATLAB y las operaciones de escritura en archivos, es solo el procesamiento realizado por los algoritmos.

Las imágenes con las que se realizaron los experimentos fueron reducidas con el script de MATLAB `reducirImagen.m`, para luego comparar lo que devuelve el método con la imagen original.

La mayoría de las veces vamos a estar usando, sin perdida de generalidad, imágenes cuadradas y valores de k , tales que $(anchoDeImagen + k)/(k + 1)$ de como resultado un número entero. Decidimos esto para no trabajar con casos bordes de acuerdo a la esquematización del zoom dada por la catedra.

3.1. Vecino Mas Cercano

Siguiendo los pasos explicados en el desarrollo se aplicaron los métodos de vecino mas cercano en distintas experiencias, utilizando las imágenes que se describen a continuación:

Confites: Es una imagen de 257x257, tiene la particularidad de que contiene pequeños confites, para los cuales si se aplicara un método de baja calidad podrían no distinguirse visualmente.

MonaLisa: Es una imagen de 257x257, el clásico cuadro pintado por Leonardo Da Vinci. Con la misma podemos ver el efecto de los métodos sobre una pintura de una figura humana.

Palabras: Es una imagen de 641x361 que contiene palabras de distintos tamaños, color, y escritas en diferentes direcciones.

Globo: Es la imagen de un globo aerostático de 1425x1425, el mismo esta pintado con pequeños rectángulos de distintos colores. Podemos estudiar si los mismos se ven bien luego de aplicar los métodos.

Leopardo: Es una imagen de 2821x1861 de un leopardo. El mismo contiene manchas, y podemos estudiar la calidad de las mismas al aplicarle las distintas versiones del método.

Venus: Es una imagen de 4097x4097 del planeta venus. Se considera que es de alta calidad.

Las siguientes imágenes fueron utilizadas solo para un k de 3:

Hombre Colores: Es una imagen de 961x541 de la sombra negra de un hombre que mira una pantalla de colores que representan el ADN.

Lena: Es una imagen de 513x513 de la clásica Lena.

Pez: Es una imagen de 2049x2049 de un pez en un océano.

Fantasma: Es una imagen de 201x201 del dibujo de un fantasma.

Para las diez imágenes anteriormente descriptas y un k fijo de 3, y se obtuvieron los siguientes resultados para la métrica PSNR:

Imagen	Original	Dinámico	Promedio
Fantasma	23.04	20.71	19.72
Confites	24.90	22.02	21.32
MonaLisa	27.24	22.02	21.32
Lena	22.96	22.96	22.14
Palabras	16.83	15.16	16.43
Hombre Colores	15.80	16.01	15.80
Globo	31.92	29.61	28.55
Leopardo	26.20	24.22	24.18
Pez	29.92	31.69	29.92
Venus	29.98	28.15	28.42

Tabla 1: Resultados de PSNR obtenidos para las 10 imágenes ejecutadas en las variantes del método de vecino más cercano

Y los siguiente resultados para el tiempo(en segundos):

Imagen	Original	Dinámico	Promedio
Fantasma	0.010947	0.010397	0.01063
Confites	0.014678	0.018079	0.018615
MonaLisa	0.019127	0.010217	0.015209
Lena	0.033704	0.028173	0.042221
Palabras	0.026632	0.025306	0.030714
Hombre Colores	0.057624	0.051142	0.067947
Globo	0.21042	0.182193	0.247481
Leopardo	0.542769	0.486599	0.643381
Pez	0.451839	0.378842	0.530081
Venus	1.69007	1.44239	2.03498

Tabla 2: Tiempos de ejecución para las 10 imágenes ejecutadas en las variantes del método de vecino mas cercano

En la tabla 2 las imágenes se encuentran ordenadas por cantidad de pixeles en forma ascendente. Se puede ver como el tiempo de computo aumenta con la misma. También se puede apreciar que para una misma imagen que el método que toma mas tiempo es el de promedios, seguido por el original y por último el dinámico.

Para estudiar la diferencia entre aplicar los métodos a dos imágenes distintas, pero de igual cantidad de pixeles, se hicieron pruebas para distintos valores de k sobre dos ejemplares.

Con la imagen de mona Lisa y confites, ambas de 257x257 pixeles, se obtuvieron los siguientes resultados:

k	Original	Dinámico	Promedio
1	28.73	28.73	30.70
3	24.90	22.02	21.32
7	20.86	18.22	17.19

Tabla 3: Resultados de PSNR de la imagen confites ejecutadas en las variantes del método de vecino mas cercano.

k	Original	Dinámico	Promedio
1	30.64	30.65	32.75
3	27.24	25.18	24.05
7	23.94	21.37	18.64

Tabla 4: Resultados de PSNR de la imagen mona Lisa ejecutadas en las variantes del método de vecino mas cercano.

k	Original	Dinámico	Promedio
1	0.021428	0.021667	0.029415
3	0.020178	0.009146	0.016885
7	0.007989	0.00505	0.015385

Tabla 5: Resultados de tiempo de computo de la imagen confites ejecutadas en las variantes del método de vecino mas cercano.

k	Original	Dinámico	Promedio
1	0.021445	0.027944	0.030327
3	0.007708	0.014268	0.023579
7	0.015515	0.01302	0.018491

Tabla 6: Resultados de tiempos de computo de la imagen mona Lisa ejecutadas en las variantes del método de vecino mas cercano.

En ambos casos para todas las versiones del método, para un $k=1$ la perdida de calidad de la imagen es apreciable a simple vista. Al aumentar el k , en todos los casos se obtiene un PSNR menor, con lo que el error cuadrático medio es mayor. A simple vista se distingue claramente la perdida de calidad al aumentar k .

Para ambas imágenes el PSNR da mayor para el método original, seguido del dinámico y el promedio. No obstante para $k=1$ el método de promedio da un PSNR mayor a los otros dos.

Este ultimo método genera un efecto apreciable de desenfoque. Sin embargo para ambas imágenes, para $k=3$ y 7 , visualmente la imagen obtenida presenta claras diferencias con la original. Esto no se refleja en la métrica de PSNR, que a pesar de ser menor, debería haber una diferencia mas significativa.

Podemos ver que en todos los casos el PSNR da mayor para la imagen de mona lisa, con lo que podemos concluir que para el método de reconstrucción de la imagen, es importante el contenido específico de la misma.

También para imágenes de distintas dimensiones en pixeles, se hicieron estudios de la métrica de PSNR.

Para la imagen de palabras se obtuvieron los siguientes resultados para los distintos valores de k :

k	Original	Dinámico	Promedio
1	19.97	19.96	22.30
3	16.83	15.16	16.43
4	16.04	14.27	15.62
7	14.47	12.96	14.54
9	13.83	12.65	14.10

Tabla 7: Resultados de PSNR de la imagen palabras ejecutadas en las variantes del método de vecino mas cercano.

k	Original	Dinámico	Promedio
1	0.025911	0.02098	0.036611
3	0.062486	0.064187	0.07201
4	0.027722	0.024487	0.034728
7	0.02872	0.022452	0.034051
9	0.019196	0.017627	0.022647

Tabla 8: Resultados de tiempo de la imagen palabras ejecutadas en las variantes del método de vecino mas cercano.

Nuevamente en todos los casos a partir de $k=1$ se puede observar la perdida de calidad en la imagen, y siempre disminuye la calidad al aumentar k , tanto cualitativamente observando la imagen como cuantitativamente utilizando la métrica.

En este caso particular podríamos decir que al estar formada la imagen por palabras separadas, el vacío blanco que hay entre las mismas hace que el método sea ineficiente, a tal punto que el método de promedio que en la anterior experiencia, y en posteriores da de una calidad menor, en este caso no. Esto

puede apreciarse a simple vista en las imágenes. También se puede apreciar el efecto de desenfoque del método de promedio.

En las siguientes figuras podemos ver como se pasa de una imagen legible a una ilegible variando el k.

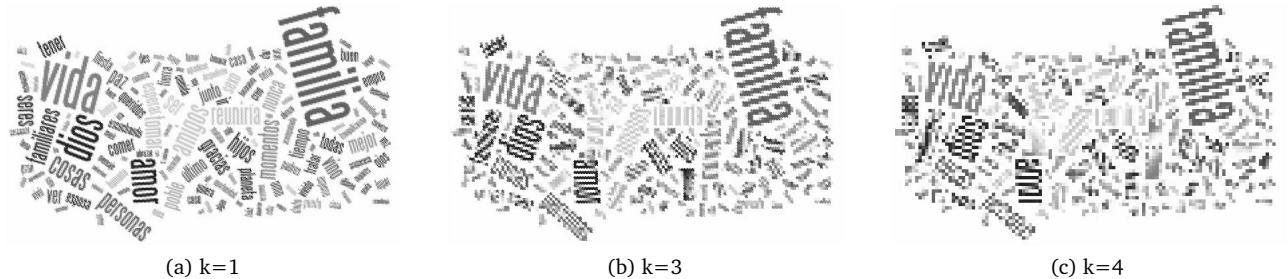


Figura 10: Perdida de calidad en la imagen en el método del vecino mas cercano version Original, al aumentar k

En la figura 10 podemos apreciar como disminuye la calidad del método al ir aumentando k. Para k=1 las palabras son legibles, para k=3 solo las palabras grandes resultan legibles. Para k=4 se pierde más calidad aun, y por ejemplo la palabra "amigos", que resulta legible para k=3, deja de serlo.

Para la imagen de globo, se obtuvieron los siguientes resultados:

k	Original	Dinámico	Promedio
1	35.64	35.64	37.62
3	31.92	29.61	28.55
7	28.32	25.87	22.22
15	25.42	22.90	15.74

Tabla 9: Resultados de PSNR de la imagen globo ejecutadas en las variantes del método de vecino mas cercano.

k	Original	Dinámico	Promedio
1	0.433569	0.419082	0.475161
3	0.202544	0.189253	0.251413
7	0.15614	0.126274	0.192875
15	0.144155	0.110308	0.180333

Tabla 10: Resultados de tiempo de la imagen globo ejecutadas en las variantes del método de vecino mas cercano.

Cualitativamente hasta para k 3 la es indistinguible al ojo la pérdida de calidad de las imágenes, y a partir de k 7, ya comienza a ser mas apreciable.

Nuevamente la métrica indica que el método mas eficiente es el Original, seguido del Dinámico y del Promedio. Para k=1 se obtiene que el método de promedio es mas eficiente. Para este ultimo puede observarse el desenfoque del mismo, que para k chico es aceptable, pero para k grande el efecto es tal que la imagen pierde su forma.

En base a los resultados obtenidos, podemos decir que el método de promedio solo es útil para k chico. En la figura 11 podemos ver el efecto que causa el promedio sobre la imagen para k=15.

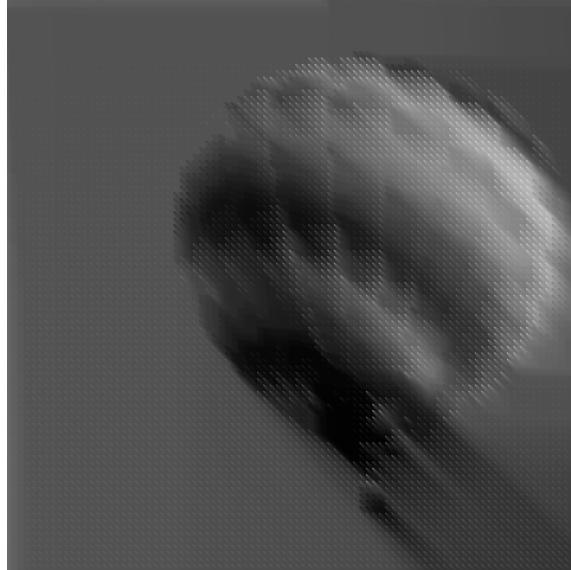


Figura 11: Imagen obtenida para por el método de promedios para k=15

Para la imagen de leopardo se obtuvieron los siguientes resultados para PSNR:

k	Original	Dinámico	Promedio
1	29.49	29.48	31.72
2	28.44	25.92	26.65
3	26.20	24.22	24.18
4	25.35	23.14	22.55
9	22.47	20.46	18.06
11	21.82	19.87	16.75
14	21.04	19.08	14.82
59	16.92	15.62	10.84

Tabla 11: Resultados de PSNR de la imagen leopardo ejecutadas en las variantes del método de vecino mas cercano.

k	Original	Dinámico	Promedio
1	1.0866	1.0504	1.22644
2	0.657721	0.61781	0.78355
3	0.524817	0.454536	0.638131
4	0.484191	0.379946	0.559047
9	0.37489	0.295424	0.46353
11	0.368663	0.280538	0.45557
14	0.360687	0.276276	0.441304
59	0.3529086	0.260347	0.432781

Tabla 12: Resultados de tiempo de la imagen leopardo ejecutadas en las variantes del método de vecino mas cercano.

Nuevamente obtenemos el mismo orden para la calidad indicada por la métrica, que tiene una excepción para k=1 y 2 , donde el promedio obtiene mayor calidad. Se puede ver también la disminución de la calidad al aumentar k. Este efecto se puede apreciar cualitativamente. Las manchas del leopardo son visibles hasta para k=14, sin embargo es notoria la diferencia entre este último valor de k y los primeros cuatro. Estos últimos, tienen una calidad que puede considerarse muy buena.

Hasta para $k=1$ puede observarse el desenfoque del promedio. No obstante para $k=14$ el efecto distorsiona a la imagen demasiado.

Se hizo una prueba para un k grande de 59, y se obtuvo como resultado que ni siquiera era observable el leopardo, para ninguna de las versiones.

Para la imagen de venus se obtuvieron los siguientes resultados para PSNR:

k	Original	Dinámico	Promedio
1	33.39	33.39	35.42
3	29.98	28.15	28.42
7	27.16	25.39	23.50
15	24.98	23.33	16.26
31	23.18	21.40	8.93
63	21.42	19.53	8.58
127	19.66	17.71	8.51
255	17.27	15.78	8.50

Tabla 13: Resultados de PSNR de la imagen venus ejecutadas en las variantes del método de vecino mas cercano.

k	Original	Dinámico	Promedio
1	3.45692	3.34598	3.8553
3	1.68882	1.48089	2.03063
7	1.26326	1.02672	1.55511
15	1.17502	0.885899	1.4469
31	1.1376	0.873169	1.42114
63	1.13693	0.838262	1.40754
127	1.12636	0.828674	1.39681
255	1.13641	0.831078	1.43211

Tabla 14: Resultados de tiempo de la imagen venus ejecutadas en las variantes del método de vecino mas cercano.

Nuevamente obtenemos que según la métrica el método de mayor calidad es el Original, seguido del Dinamico, y el Promedio. También para los primeros k probados el método de promedio tiene mayor calidad. A partir de $k=31$ la imagen obtenida por este último método es demasiado oscura y casi no se ve, cuando para los otros dos métodos la imagen tiene una calidad aceptable. Este último hecho se lo podríamos atribuir a que la imagen tiene bordes negros alrededor del planeta venus, que se promedian con la imagen. En los demás promedios puede verse el efecto de desenfoque del promedio.

Al ser una imagen de alta calidad, para los primeros k considerados y los métodos Original y Dinámico , el resultado obtenido es indistinguible a los ojos de la imagen original.

3.2. Bilineal

La experimentación empezará aplicando zoom con $k = 2$ a todas las imágenes, y luego empezaremos a variar el k para las últimas 3 imágenes, ya que al ser grande, hay mayor cantidad de k válidos. Se dispuso de una serie de experimentaciones sobre el siguiente conjunto de 10 imágenes:

- Las imágenes `imagen1.tiff` de 511x511, `imagen2.tiff` de 1024x1024 y `imagen3.tiff` de 511x511 fueron elegidas por ser medianas y no poseer tanto cambio de colores muy marcados.

- Las imágenes `imagen4.tiff` de 1024x1024 y `imagen7.tiff` de 511x511 fueron elegidas por ser medianas y por tener personas en las mismas.
- Las imágenes `imagen5.tiff` de 1024x1024 y `imagen6.tiff` de 511x511 fueron elegidas por ser medianas y por tener muchas variaciones de colores.
- La imagen `imagen8.tiff` de 3289x3289 fue elegida por ser grande y por tener personas en la misma.
- La imagen `imagen9.tiff` de 3997x3997 fue elegida por ser grande y por tener muchas variaciones de colores.
- La imagen `imagen9.tiff` de 4202x4303 fue elegida por ser grande y no poseer tanto cambio de colores muy marcados.

Con $k=2$, obtuvimos los siguiente resultados de PSNR:

Imagenes	Original	Ignorando	Diagonal	Bloques
1	32.6180	30.0720	31.7492	32.6217
2	32.0807	31.3422	31.8774	32.0938
3	34.4520	32.9934	34.4501	34.4509
4	27.7731	23.6071	26.8081	27.7777
5	21.9590	19.7860	21.7686	21.9598
6	20.9380	19.7715	20.7258	20.9392
7	29.9870	26.6496	29.3290	29.9945
8	37.0023	31.9828	36.2431	37.0397
9	24.0090	21.4330	23.6539	24.0111
10	34.9894	31.8309	34.9878	34.9894

Tabla 15: Tabla de PSNR de las 10 imágenes ejecutadas en las 4 variantes del método de interpolación bilineal

Con $k=2$, obtuvimos los siguiente resultados de tiempo(en segundos):

Imagenes	Original	Ignorando	Diagonal	Bloques
1	5.60	4.96	4.92	5.08
2	5.90	5.93	5.91	5.96
3	4.96	4.92	4.93	4.94
4	6.44	6.00	5.95	6.00
5	5.91	5.92	5.90	5.92
6	4.93	4.98	4.93	4.95
7	4.92	4.93	4.91	4.94
8	18.21	18.52	17.75	18.90
9	24.35	24.44	23.82	24.55
10	27.07	27.11	26.52	27.43

Tabla 16: Tabla de tiempos de las 10 imágenes ejecutadas en las 4 variantes del método de interpolación bilineal

Variando el k , con las imágenes `imagen8.tiff`, `imagen9.tiff` y `imagen10.tiff` se obtuvieron los siguientes resultados:

Con la imagen `imagen8.tiff`:

k	Original	Ignorando	Diagonal	Bloques
3	34.1484	29.3108	33.3469	34.1729
5	30.6667	26.8999	30.1220	30.6836
7	28.3968	25.3233	27.9253	28.4112

Tabla 17: Tabla de PSNR de la imagen8 ejecutadas en las 4 variantes del método de interpolación bilineal, variando el k en 3, 5 y 7.

k	Original	Ignorando	Diagonal	Bloques
3	17.73	17.27	16.60	17.47
5	16.37	16.25	15.67	16.78
7	15.92	16.06	15.70	16.65

Tabla 18: Tabla de tiempos de la imagen8 ejecutadas en las 4 variantes del método de interpolación bilineal, variando el k en 3, 5 y 7.

Con la imagen `imagen9.tiff`:

k	Original	Ignorando	Diagonal	Bloques
3	22.4157	19.8676	22.1082	22.4174
5	20.4671	18.0751	20.1985	20.4688
8	18.7939	16.5599	18.5421	18.7957

Tabla 19: Tabla de PSNR de la imagen9 ejecutadas en las 4 variantes del método de interpolación bilineal, variando el k en 3, 5 y 8.

k	Original	Ignorando	Diagonal	Bloques
3	22.93	23.26	22.55	23.81
5	22.04	21.95	21.58	22.14
8	21.83	21.15	19.93	21.55

Tabla 20: Tabla de tiempos de la imagen9 ejecutadas en las 4 variantes del método de interpolación bilineal, variando el k en 3, 5 y 8.

Con la imagen `imagen10.tiff`:

k	Original	Ignorando	Diagonal	Bloques
1	38.2838	34.9337	38.2821	38.2838
5	30.9075	27.6721	30.9051	30.9075
8	27.4966	25.3160	27.4906	27.4967

Tabla 21: Tabla de PSNR de la imagen10 ejecutadas en las 4 variantes del método de interpolación bilineal, variando el k en 1, 5 y 8.

k	Original	Ignorando	Diagonal	Bloques
1	33.77	33.33	32.35	33.02
5	23.48	23.47	22.77	24.43
8	23.42	23.55	21.65	23.75

Tabla 22: Tabla de tiempos de la imagen10 ejecutadas en las 4 variantes del método de interpolación bilineal, variando el k en 1, 5 y 8.

Es importante destacar que el tiempo de cómputo no se vio afectado por las variantes del método, sino solamente, como era de esperar, por el tamaño de cada imagen.

Luego de analizar los resultados objetivos, procederemos a analizar subjetivamente las imágenes. Con k=2, en las variantes del método original, por diagonales y por bloques se pudo observar un cierto efecto de desenfoque de la imagen en donde los contornos de los objetos no estaban bien definidos. Esto se ve en todas las imágenes excepto en la imagen5, que se nota muy despixelada, seguramente debido a la cantidad de colores y letras que presenta la misma. La siguiente comparación muestra lo recientemente enunciado.



Figura 12: Fracción Imagen 4 - Bilineal por bloques



Figura 13: Fracción Imagen 4 - Orginal



Figura 14: Fracción Imagen 5 - Bilineal por bloques



Figura 15: Fracción Imagen 5 - Orginal

Tambien es notable que el método con la variante de ignorar un pixel, en las imagenes que varia el color seguido, deja más detalles especiales de lo que esperabamos, como una imagen despixelada, o con franjas blancas y negras, que si le aplicáramos zoom con el visor de imagenes tradicional, queda al descubierto una trama negra más notoria. En cambio en las que no varía el color seguido, deja imágenes aceptables. En la siguiente comparación se observa lo recientemente detallado.

Esto concuerda tambien con los valores resultantes del PSNR calculado. En donde con la variante de ignorar un pixel, la imagen 4 obtuvo casi 4 puntos menos de PSNR comparado con las otras variantes, mientras que la imagen 2 obtuvo similar valor para todas las variantes.



Figura 16: Fracción Imagen 4 - Bilineal ignorando un pixel



Figura 17: Fracción Imagen 4 - Orginal

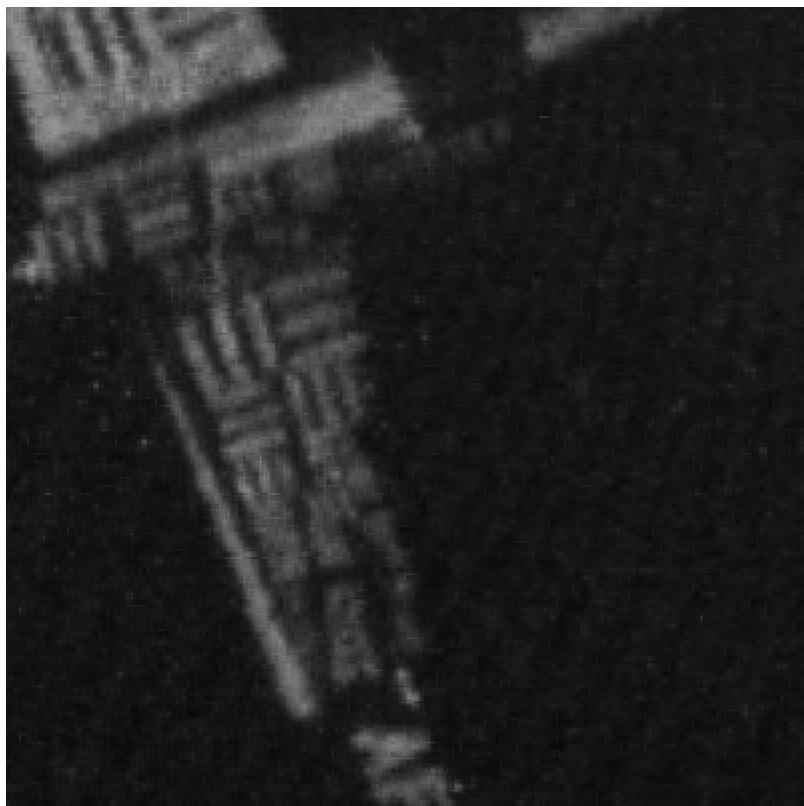


Figura 18: Fracción Imagen 2 - Bilineal por bloques

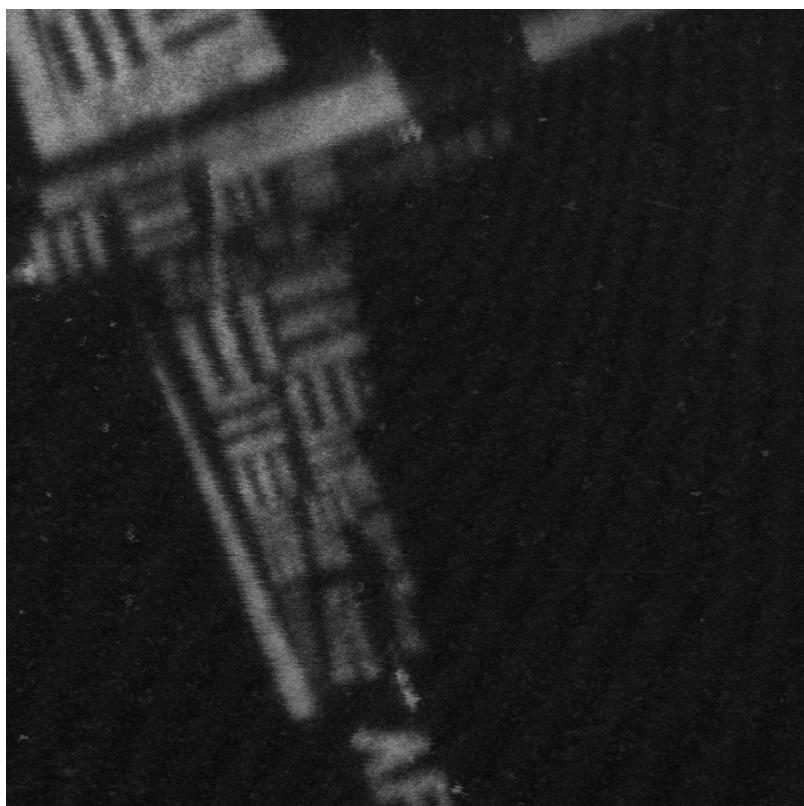


Figura 19: Fracción Imagen 2 - Orginal

Comparando las variantes original, por diagonal y por bloques sobre cada una de las otras imágenes, no se noto diferencias visibles al sentido humano comparando cada una de ellas, es decir comparando la misma foto pero con diferentes variantes. De pudo apreciar, que para las imágenes grandes a medida que aumentamos el k ayudándonos con la opción del zoom del visor de imágenes del visor de imágenes tradicional, las mismas contenian un desenfoque cada vez más notorio. Esto se evidenció notablemente en la `imagen9.tiff`, ademas dio un valor bajo de PSNR en comparación a las otras 2 imágenes grandes, posiblemente a causa de la gran cantidad de variaciones de colores muy seguidas.

Con las imágenes donde no había cambios tan seguidos de colores, se pudo verificar que la variante de ignorar un pixel, retorno un valor similar de PSNR, con diferencias de 1 o 2 puntos en relación a las otras variantes. Nunca se obtuvo mejor valor de PSNR. En cuanto con las imágenes que si tenian cambios seguidos de color, el valor de PSNR dio muy por debajo de lo que dieron las otras variantes, llegando a diferencias de entre 4 y 6 puntos. Además se encontraron artifacts muy notorios en las imágenes como los expuestos en la sección de resultados.

Tambien mostramos que la variante original y la de por bloques, casi devolvieron similares valores de PSNR, con minimas diferencias, siempre obteniendo mayor valor la variante por bloques. Suponiamos que iban a dar similares pero que la variante original iba a dar siempre mejor que por bloques debido a que en la implementación se realizan menos casteos de variables y eso podria haber arrastrado errores de precisión.

En cuanto a los valores de PSNR de la variante de diagonales, siempre obtuvimos valores con diferencias de a lo sumo 1 punto, lo que marcaría que las imágenes elegidas posiblemente no tengan una continuidad diagonal de color en todos los sectores.

Para destacar es que los resultados arrojaron que a medida que el k aumenta, si la imagen posee bastantes cambios abruptos de colores, se obtendrán imágenes con menos calidad objetiva y subjetiva.

Tambien observamos que el tiempo de cómputo no se vio alterado al cambiar de variante, por lo que podemos asegurar que solo el tamaño de cada imagen influye en el mismo.

Para finalizar con el estudio de este método, concluimos que la mejor variante en cuanto calidad objetiva y calidad subjetiva es la variante por bloques.

3.3. Splines

3.3.1. Bloques fijos

Como mencionamos anteriormente en el desarrollo, implementamos dos variantes distintas de este método y por lo tanto experimentamos con ambas variantes. Las imágenes seleccionadas se encuentran en la carpeta Experimentación/Splines y a continuación explicaremos qué particularidad tiene cada imagen junto con los k que vamos a trabajar.

Tanque: esta imagen fue elegida para ver como afecta en los resultados el hecho de tener todo un terreno relativamente uniforme junto con un objeto más o menos mediano (en este caso el tanque), de tonalidad similar. Al igual que Cuadrícula y Grises, las dimensiones de esta imagen son 511x511 y trabajaremos con $k = 1, 2$ y 4 .

Cuadrícula: nos pareció interesante incluir esta imagen por el hecho de contener, en su mayor parte, cuadrados y así poder ver el comportamiento del algoritmo de splines por bloques de tamaño fijo. Además como está compuesta básicamente por líneas, podemos ver que tan tolerante es al k , es decir, cuanta información se pierde.

Grises: al igual que Cuadrícula, esta imagen se puede dividir pero en rectángulos y es interesante ver si el algoritmo por bloques fijos puede funcionar relativamente bien, ya que los bloques que usa deben ser necesariamente cuadrados. Además, dado que existe un cambio de tonalidad por cada rectángulo adyacente, esto puede significar una experimentación interesante en la variante de bloques de tamaño variable porque sirve para evaluar que tan bien puede identificar ese cambio de tonalidad.

Lago:

Auto y Rosa: elegimos estas imágenes por el nivel de detalle que contienen. El auto sobre todo, tiene una cantidad importante de iluminación y reflejos, cosa que ninguna otra imagen hasta ahora tenía. Las dimensiones de Auto (4096x4096) nos permiten trabajar con varios valores de k que serán 2, 4, 6, 8 y 12 mientras que con Rosa (2047x2047) vamos a usar $k = 1, 2, 5$.

Piano: esta imagen fue elegida para ver como se comportan los algoritmos tanto con el reflejo de las teclas (que está bien delimitado) como con los bordes de las mismas. Sus dimensiones son 1549x1549 y trabajaremos con $k = 1, 2, 3$.

Si bien presentamos algunos recortes de las imágenes utilizadas en esta sección que ilustran las diferencias más relevantes entre la imagen original y las imágenes procesadas por el algoritmo, también mencionamos otras diferencias importantes que, por cuestiones de espacio y claridad, preferimos decir donde se encuentran y que se vean directo de las imágenes adjuntas al trabajo. Ahora que ya explicamos los aspectos que van a compartir las experiencias de ambas variantes, pasamos a hablar específicamente de la variante del método que utiliza bloques de tamaño fijo. La idea va a ser tratar de ver las diferencias en las imágenes al tomar bloques de un tamaño u otro, eligiendo estos tamaños en base a la imagen con la cual vamos a estar trabajando. El enfoque que adoptaremos en general va a ser el siguiente: dada una imagen y un k , aplicar el método una vez con un bloque del tamaño de la imagen y otra vez con bloques de cierto tamaño, que será decidido según ciertos aspectos de la imagen. Por ejemplo, si la imagen es relativamente parecida en todas partes (porque se trata de un terreno por ejemplo) salvo por un área delimitada, el tamaño del bloque va a ser el tamaño de este área. De esta manera, estamos separando de alguna manera a la parte que difiere en la imagen de las partes que son parecidas. Decimos en general porque en algunos casos no se va a cumplir que la imagen posea esta uniformidad y ahí va a ser cuando cambiemos un poco este enfoque. En casos donde la imagen posee muchos detalles y es muy cambiante, vamos a tomar bloques de tamaño chico bajo la hipótesis de que como es probable que de un bloque a otro la imagen cambie significativamente, al tomar bloques reducidos vamos a estar descartando información sobre otra parte de la imagen que no aporta al área que queremos procesar, porque justamente en esa otra parte la imagen pudo haber cambiado mucho.

Comencemos analizando los resultados obtenidos para las imágenes de 511x511. Para la imagen Tanque.tiff, utilizamos bloques de un tamaño que aproxima a la mitad del tanque (esto es debido a que el tanque es rectangular y los bloques deben ser cuadrados por como está diseñado el algoritmo). Haciendo un análisis cualitativo de nuestras imágenes, comenzando por las que se procesaron con $k = 1$, podemos observar que entre la que trabaja con un bloque del tamaño entero de la imagen (256x256) y la que trabaja con bloques de 50x50 prácticamente no hay diferencias que se puedan percibir fácilmente de manera visual. Si tratamos de ver muy detenidamente las diferencias entre ambas, podemos encontrar que, en la parte más oscura del tanque (más o menos por la mitad izquierda) la imagen procesada con bloques de 256 remarcaba un poco más el blanco del borde. Sin embargo, como ya dijimos, es prácticamente imperceptible. Lo que sí se puede percibir notoriamente es la diferencia entre cualquiera de estas imágenes y la imagen original. Lo que más llama la atención son los detalles del tanque que se pierden totalmente, por ejemplo las líneas negras que tiene en la parte de arriba que se borronean. En la parte del cañón se produce un despixelado bastante fuerte, probablemente mayor al que se produce en las otras partes del tanque como por ejemplo en la de abajo donde están las ruedas. Una de las razones posibles que podrían explicar esto es que, justamente, los píxeles del cañón están aislados del resto del tanque y sus vecinos son los píxeles del terreno, entonces por más que tomemos un bloque grande de 256x256 o bloques del tamaño de la mitad del tanque, seguramente en el/los bloque/s correspondiente/s al cañón del tanque van a estar tanto los píxeles del cañón como los del terreno y, seguramente van a predominar estos últimos. Observemos que en las líneas negras que mencionamos al principio sucede algo similar, se pierde mucha de esa información debido a que las líneas son muy finas y por lo tanto, en el bloque que tomemos que contengan estas líneas, van a predominar los píxeles grises. Todo esto sumado a que cuando realizamos la reducción del tamaño de la imagen ya se pierde una cantidad de información significativa y las partes donde están los detalles se ven más afectadas.

Para $k = 2$ y $k = 4$ lo que sucede es que las diferencias que notamos para $k = 1$ tanto entre las imágenes con zoom como entre alguna de ellas y la original aumentan bastante. En el caso particular de $k = 2$ ahora quizás es más fácil identificar diferencias entre la imagen aumentada con un bloque de 171 y la procesada con bloques de 30 sobre todo en la parte de las ruedas del tanque. Sin embargo, lo más relevante es que ahora la parte del cañón ya se comienza a confundir con el terreno. Y si pasamos a $k = 4$, para ambas imágenes, el cañón ya se perdió por completo.

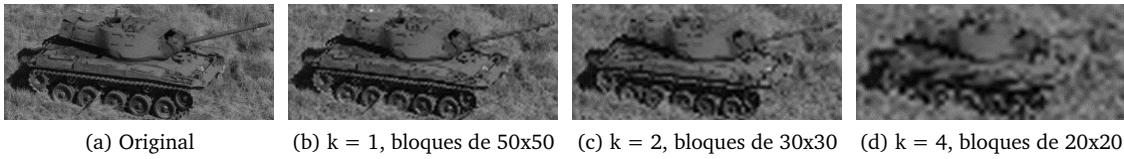


Figura 20: Comparación entre la imagen original y la procesada con distintos k y tamaños de bloque chicos

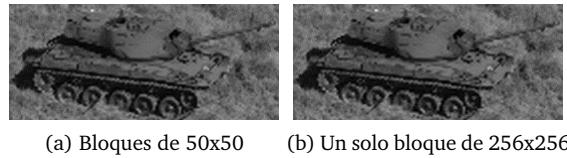


Figura 21: Comparación en base al tamaño de los bloques para $k = 1$

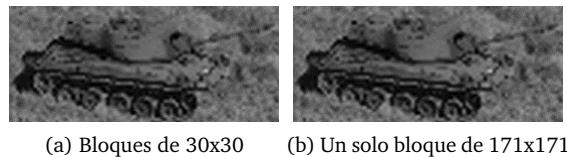


Figura 22: Comparación en base al tamaño de los bloques para $k = 2$

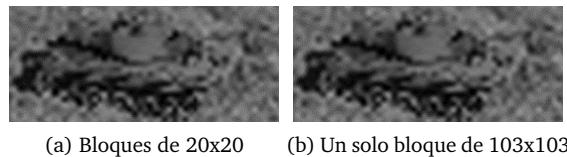


Figura 23: Comparación en base al tamaño de los bloques para $k = 4$

En resumen, pareciera que las diferencias entre las imágenes procesadas con bloques relativamente chicos en relación al tamaño de las imágenes y las que se procesan con un bloque del tamaño de la imagen, van aumentando de a poco a medida que aumentamos el k . Si bien procesarlas de una u otra manera no evita (al menos en este caso) los problemas más relevantes (como el despixelado o que se confunda totalmente una parte de la imagen con el resto de los píxeles la rodean), existen algunas pequeñas diferencias como por ejemplo en las ruedas del tanque. Dado que estas diferencias no son visualmente muy relevantes, no podemos estar seguros de si contribuyen a una reconstrucción mejor o peor de la imagen ya que son diferencias muy pequeñas. Para resolver esto vamos a cuantificar un poco las cosas, a continuación mostramos el PSNR de cada imagen junto con el tiempo de cómputo que llevó procesarla:

Tanque			
K	Bloque	PSNR	T. de cómputo
1	256	29.9208	0.216126
1	50	29.9234	0.138024
2	171	27.5585	0.148907
2	30	27.5685	0.086371
4	103	25.4209	0.093326
4	20	25.4398	0.079059

Podemos ver como el análisis cualitativo que habíamos hecho sobre las imágenes, concuerda bastante con los números presentados en la tabla. Para empezar, vemos como efectivamente a medida que aumentamos el k el PSNR decrece y eso significa que el error entre la imagen esperada y la resultante es mayor. También es cierto que las diferencias entre las imágenes procesadas con un bloque y las que procesamos con varios bloques aumentan a medida que aumenta el k ya que para $k = 1$ la diferencia es de 0.0026, para $k = 2$ de 0.01 y para $k = 4$ de 0.0189. Lo interesante de esto es que dado que estas diferencias son muy chicas, no eramos capaces de decidir si estaban ayudando a reconstruir mejor o peor la imagen porque visualmente era muy difícil. Ahora podemos ver que, en los tres casos el PSNR es mayor cuando trabajamos la imagen con bloques del tamaño de la mitad del tanque.

En cuanto al tiempo de cómputo, se observa que si bien todos los tiempos medidos son bastante chicos, al trabajar con las imágenes reducidas de mayor tamaño (k chico), el tiempo es mayor. Si bien trabajar con cualquier valor de k va a resultar en una imagen expandida del mismo tamaño, lo que sucede al trabajar con k pequeños es que los splines que se van a ir generando van a tener más información, con lo cual van a demandar un costo computacional más grande. Esto se puede ver también por la diferencia de tiempos que existe, para un mismo k , cuando se trabaja con bloques de tamaño chico y cuando se trabaja con un bloque grande. La tabla muestra que, en el primer caso, los tiempos son menores, lo que significa que concuerda con lo que dijimos recién, ya que al trabajar con bloques pequeños, si bien vamos a generar más splines, van a tener menos información. Por lo tanto una posible hipótesis en cuanto al tiempo de cómputo es que generar cierta cantidad de splines con muchos puntos es más costoso que generar quizás, una mayor cantidad pero con menos puntos cada uno.

Pasemos ahora a analizar la imagen Grises y veamos si sucede algo parecido a lo que pasó con la anterior. Para esta imagen, al igual que la anterior, vamos a trabajar con bloques de un tamaño cercano al tamaño de los rectángulos (además del caso de trabajar con un solo bloque que siempre lo usamos). En este caso vuelve a pasar que las diferencias entre las imágenes que trabajan con muchos bloques y las que trabajan con un solo bloque aumentan a medida que aumenta el k . Comenzando por $k = 1$, podemos ver que el tercer y cuarto rectángulos de la primera columna (en los otros es difícil de ver esto) difieren levemente entre la imagen 256 y 38: en el primer caso pareciera estar cubierto por líneas horizontales y en el extremo derecho también por líneas verticales, mientras que en el segundo caso pasa algo parecido pero en la parte de arriba de esos rectángulos no existen esas líneas. En principio es difícil tratar de pensar un posible motivo por el cual sucede esto ya que solo es apreciable en algunas partes de la imagen y no en todas. En cuanto a las diferencias con la imagen original, podemos observar que existe una leve variación entre el tamaño de los rectángulos: en la original los rectángulos de la columna izquierda y derecha parecieran ser un tanto más chicos que los de las imágenes resultantes y el del medio un poco más grande. Para $k = 2$ la diferencia en los tamaños de los rectángulos entre las imágenes resultantes y la imagen original es significativamente más notoria al igual que el cuadriculado que observamos antes en algunos rectángulos, que ahora se puede observar en otros como los que se encuentran más cercanos al centro de la imagen. Finalmente para $k = 4$ podemos ver que los dos defectos que nombramos antes se potenciaron bastante al igual que las diferencias entre las imágenes procesadas. A grandes rasgos se puede notar que la mayor parte de los rectángulos de la imagen 103 posee una especie de cuadriculado mientras que los de la imagen 38 solo líneas, en los extremos de ambos casos varía levemente este patrón. Vemos que al igual que nos pasó con la imagen Tanque, si bien existen diferencias entre la imagen procesada con un

bloque solo y la que procesada con varios bloques, ambas comparten un defecto mayor que es el cambio en el tamaño de los rectángulos que es lo más apreciable a la vista. A continuación presentamos la tabla con los resultados sobre el tiempo de cómputo y el PSNR:

Grises			
K	Bloque	PSNR	T. de cómputo
1	256	37.3803	0.213046
1	50	37.4554	0.12934
2	171	34.0313	0.142121
2	25	34.0659	0.104159
4	103	31.7862	0.093746
4	15	31.8087	0.072207

Si bien el procesamiento por bloques sigue siendo superior al procesamiento con un solo bloque en términos de PSNR, al igual que con la imagen Tanque, aca la diferencia entre el PSNR de una y de otra va disminuyendo a medida que aumenta el k , a diferencia del caso anterior. Visualmente esto tiene algún sentido ya que, tanto en un caso o en otro a medida que el k aumentaba el rayado y el cuadrículado se hacían cada vez más visibles.

Con respecto al tiempo de cómputo, no tiene mucho sentido analizarlo mucho ya que era bastante esperable que nos de muy parecido al que tomamos con Tanque, porque estamos trabajando con una imagen de las mismas dimensiones y varios de los tamaños de los bloques son los mismos y otros son muy parecidos.

Ahora vamos a presentar el análisis correspondiente a la última imagen de 511, Cuadrícula, que representa un caso bastante patológico. Decimos esto porque al aumentar el k de 1 a 2 la cantidad de información que se pierde es muy grande, lo cual es bastante lógico porque la imagen está compuesta básicamente por líneas y estas líneas contienen una cantidad de píxeles bastante reducida. Sin embargo, nos pareció una instancia interesante debido a que como el algoritmo trabaja con bloques necesariamente cuadrados y salvo por algunos pocos sectores, esta imagen es prácticamente un cuadriculado, ahora ya no es necesario tomar el tamaño del bloque como la mitad de algún sector (como sucedía en las imágenes anteriores). Para $k = 1$ lo que podemos observar si miramos la imagen que fue procesada tomando bloques del tamaño del cuadrado y la que fue procesada con un solo bloque del tamaño total de la imagen, es que esta última remarca las líneas. Si miramos en detalle, además de las líneas también remarca algunas partes de los números (se puede ver fácilmente en la parte inferior de los números 2). Si bien ambas variantes reconstruyen la parte de las líneas correctamente (con sus diferencias, pero podemos decir que es visualmente aceptable), el mayor problema viene con los números. Los números del eje vertical se entienden bastante pero los del eje horizontal parecen haber perdido una cantidad de píxeles importante. Probando con otros tamaños de bloque más grandes (por ejemplo cercanos al tamaño de uno de estos números del eje horizontal o tomar bloques más chicos que uno de los cuadrados de la imagen) se puede ver que la calidad de estos números no mejora, con lo cual empezamos a pensar que no es un problema de la elección del tamaño del bloque, si no de la cantidad de información perdida a la hora de reducir la imagen.

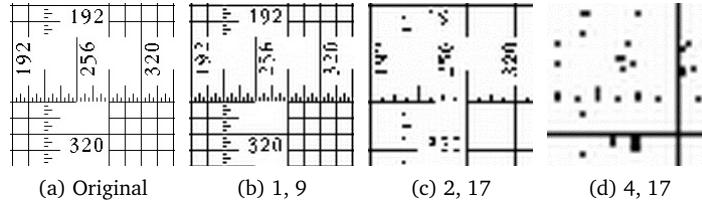


Figura 24: Comparación entre la imagen original y la procesada con distintos k . Debajo de cada imagen se encuentra su respectivo k y tamaño de bloque utilizado.

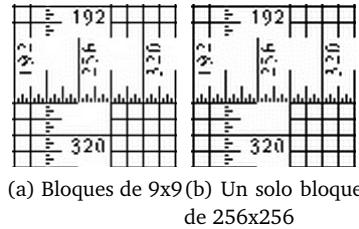


Figura 25: Comparación en base al tamaño de los bloques para $k = 1$

Para los casos con $k = 2$ y $k = 4$, las imágenes resultantes difieren mucho de la original porque la cantidad de información que se pierde al reducirlas es muy significativa, por el tipo de imagen que representan, esto se puede observar fácilmente al reducir la imagen original con esos valores de k y ver que queda algo realmente muy distinto a lo que uno esperaría, lo cual no pasa con $k = 1$ y es por eso que el método funciona mejor. Entonces podemos concluir que la diferencia que existe entre la imagen original y las procesadas con $k = 2$ y $k = 4$ no es una consecuencia del método utilizado si no del tipo de imagen, muy sensible a reducciones.

A continuación, presentamos la tabla correspondiente a esta imagen:

Cuadrícula			
K	Bloque	PSNR	T. de cómputo
1	256	11.3605	0.220708
1	9	14.1944	0.269276
2	171	9.2938	0.142723
2	17	10.0905	0.112803
4	103	8.4355	0.093314
4	17	9.0206	0.071696

Vemos como impacta el hecho de usar bloques del tamaño de los cuadrados para $k = 1$ (que, por lo que explicamos anteriormente sobre las reducciones, es el caso que más vale la pena analizar). Si bien visualmente la única diferencia significativa que existe entre estas imágenes es que una remarca más las líneas que la otra, éste fue el caso en el cual se pudo apreciar la mayor diferencia entre procesar por bloques y hacerlo solo con un bloque. Concluimos que esto sucedió justamente porque la imagen está dividida en cuadrados y los bloques con los que trabaja nuestro algoritmo deben ser cuadrados también.

En este caso, los tiempos medidos nos pueden estar diciendo algo más ya que vemos que el tiempo para $k = 1$ y un tamaño de bloque de 9 píxeles, es levemente mayor que el tiempo para un bloque de 256 y el mismo k . Esto contradice lo que veníamos pensando por los datos tomados de las imágenes anteriores, sin embargo, como se tratan de tiempos muy cortos y la diferencia no es demasiado grande,

puede que sea un caso particular que no debe ser tenido en cuenta. Seguiremos analizando este aspecto en las próximas imágenes y probablemente nos digan algo más ya que al ser de dimensiones más grandes, los tiempos van a ser mayores.

Ahora pasaremos a analizar las imágenes más grandes, empezando por Lago. Dado que en esta imagen podemos identificar tres sectores que son el lago, el cielo y el terreno y son sectores relativamente grandes los tres (junto con la imagen en general, que también es grande), vamos a trabajar con tamaños de bloques bastante variables y no tan chicos como los de las imágenes anteriores. Además, para cambiar un poco, vamos a dejarlos fijos cuando varíe el k . Estos bloques van a ser de 50, 100 y 200.

Visualmente lo primero que podemos decir es que esta vez sí son realmente imperceptibles las diferencias entre las imágenes procesadas con un tramaño de bloque u otro, incluso para los k más grandes. Más aún, lo que sucede es que si bien las diferencias entre las imágenes resultantes y la imagen original existen y son apreciables, de alguna manera no afectan la calidad de la imagen para los primeros valores de k . Esto último se debe a lo que dijimos inicialmente: estamos trabajando con una imagen donde existen tres secciones bastante grandes y que además, los bordes no están bien definidos (es decir, no hay una linea bien definida que divida una sección de la otra) entonces si existe alguna diferencia en los bordes, visualmente no va a ser muy perceptible a menos que sea muy grande. porque en la imagen inicial el borde ya es irregular. A medida que se aumenta el k lo que notamos es que estas diferencias en los arboles se hacen más significativas y ya en $k = 5$ y $k = 7$ el cambio es bastante grande, incluso en el terreno se nota que la calidad es mala.

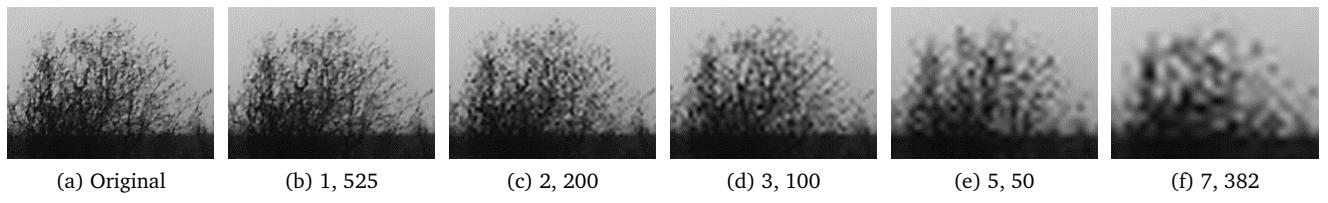


Figura 26: Comparación entre la imagen original y la procesada con distintos k . Debajo de cada imagen se encuentra su respectivo k y tamaño de bloque utilizado.

Dado que probamos el método con varios valores de k y varios tamaños de bloque, para presentar mejor la información usaremos una tabla por cada valor de k :

Lago $k = 1$		
Tamaño de bloque	PSNR	Tiempo de cómputo
50	43.8854	4.08434
100	43.9019	3.55443
200	43.9061	3.27132
1525	43.9126	7.16835

Lago k = 2		
Tamaño de bloque	PSNR	Tiempo de cómputo
50	38.9604	2.34028
100	38.9562	2.16114
200	38.9574	2.20069
1017	38.9569	4.48417

Lago k = 3		
Tamaño de bloque	PSNR	Tiempo de cómputo
50	36.8325	1.70029
100	36.8275	1.49301
200	36.8242	1.38925
763	36.8231	3.36569

Lago k = 5		
Tamaño de bloque	PSNR	Tiempo de cómputo
50	35.2486	1.21937
100	35.2415	1.23198
200	35.2409	1.12401
509	35.2398	2.38993

Lago k = 7		
Tamaño de bloque	PSNR	Tiempo de cómputo
50	34.5746	0.908093
100	34.5554	0.822251
200	34.5525	0.781545
382	34.5534	1.97297

En todas las tablas se cumple que el tiempo de cómputo correspondiente a procesar la imagen con un solo bloque grande, es mayor que utilizando bloques de tamaños más pequeños. Esta diferencia se acentúa cuando la imagen de la cual partimos es más grande y, a medida que vamos tomando k mayores la diferencia se va haciendo más chica. En conclusión, la diferencia es notoria, lo que sucede es que cuando trabajamos con imágenes chicas el tiempo de cómputo ya de por sí es chico y es por eso que tanto trabajar con un bloque grande como trabajar con bloques chicos va a demandar un costo computacional bajo. También podemos ver que esta diferencia es relevante cuando la diferencia del tamaño de bloque es necesariamente grande ya que para los tamaños 50, 100 y 200 que utilizamos, en $k = 1, 3$ y 7 nos dió un tiempo de cómputo mayor para 50 que para 100 y mayor para 100 que para 200.

En cuanto a los PSNR observados, no se puede concluir nada sobre si es mejor utilizar un tamaño de bloque u otro ya que son todos muy similares y, por ejemplo si miramos para $k = 1$, se registra un aumento a medida que aumenta el tamaño de bloque pero para $k = 3, 5, 7$ pasa exactamente lo contrario. $k = 2$ es el caso que nos incita a pensar que, para este tipo de imágenes, poco influye en el PSNR un tamaño de bloque u otro, ya que a diferencia de los demás k , no hay un crecimiento o decrecimiento estricto del PSNR según el tamaño del bloque.

Pasemos ahora a analizar la imagen Piano. Este es otro ejemplo para el cuál visualmente el método funciona muy bien. Las diferencias entre imágenes procesadas con distintos tamaños de bloque son, al igual que en el caso anterior, imperceptibles. Además de que, la única diferencia realmente notoria entre la imagen original y las resultantes, es el borde de las teclas negras (sobre todo de la tercera), el cual se despixelea un poco.

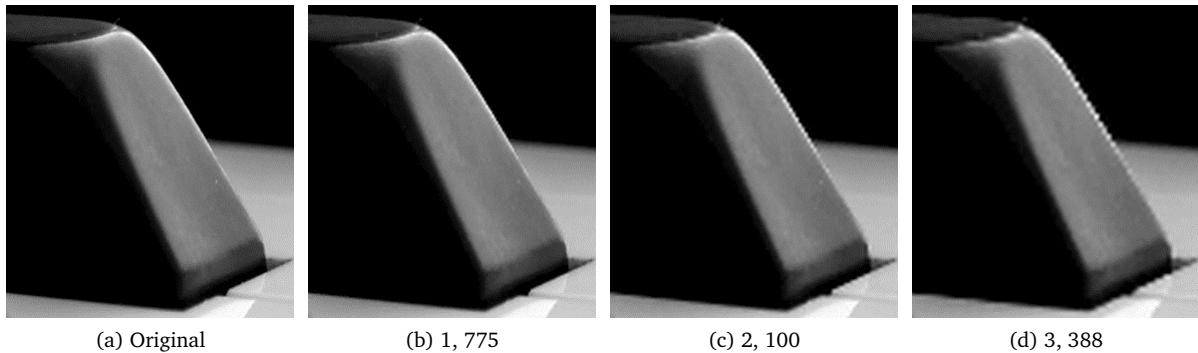


Figura 27: Comparación entre la imagen original y la procesada con distintos k . Debajo de cada imagen se encuentra su respectivo k y tamaño de bloque utilizado.

Esta diferencia aumenta con el k . Una posible razón por la cual sucede esto es por la iluminación que tiene la imagen, el borde de la tercera tecla negra está bien definido en la imagen original pero al ser blanco por el reflejo de la luz y tener de un lado negro y del otro gris, como el spline va a tener en cuenta toda esta información se va a terminar mezclando de alguna manera y al estar bien definido va a resultar notorio. Es por eso que no pasa tanto con las otras teclas donde el borde es más difuso. La tabla para esta imagen es la siguiente:

Piano			
K	Tamaño de bloque	PSNR	Tiempo de cálculo
1	200	47.8004	0.849895
1	775	47.8024	1.87312
2	100	44.8576	0.60743
2	517	44.8571	1.17809
3	50	42.8922	0.428638
3	388	42.8862	0.884801

Como podemos ver, no queda claro aca tampoco cuál es el tamaño de bloque a utilizar ya que, dependiendo el k a veces tomar un tamaño de bloque relativamente mediano (en comparación al tamaño de la imagen) da mejor que procesar la imagen como un solo bloque mientras otras veces pasa lo contrario. En relación al tiempo de cálculo, se mantiene nuestra teoría de que procesando la imagen con un solo bloque grande es más costoso que con muchos de tamaño menor por la cantidad de información necesaria

para construir cada spline.

Analicemos ahora la imagen Rosa. Vamos a estar trabajando con un tamaño de bloque bastante chico en relación al tamaño de la imagen, debido a que ésta contiene numerosos detalles y por lo tanto de un bloque a otro puede cambiar significativamente. Los k van a ser 1, 2 y 5. En este caso nuevamente las diferencias entre imágenes procesadas con un tamaño de bloque u otro no son relevantes. En cuanto a las diferencia con la imagen original, éstas son realmente apreciables recién para $k = 5$, entre ellas podemos encontrar la forma de algunas gotas y, en mayor medida, los bordes. El despixelado que se produce en los bordes es notable por el mismo motivo que en la imagen Piano, al tener gotas que tienen una tonalidad más clara que los borde, sombras que son mucho más oscuras o mismo el reflejo de la luz, se mezclan los tonos y, cuando el borde está bien definido se genera el despixelado.

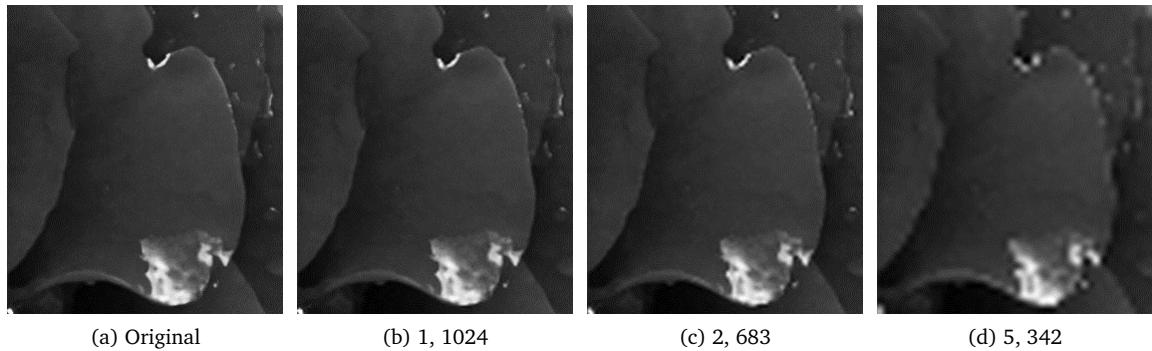


Figura 28: Comparación entre la imagen original y la procesada con distintos k . Debajo de cada imagen se encuentra su respectivo k y tamaño de bloque utilizado.

Si observamos la imagen completa, podemos ver que en los bordes de la parte inferior izquierda no se produce despixelado o por lo menos no se puede ver, porque ya vienen difusos. La tabla para esta imagen es la siguiente:

Rosa			
K	Tamaño de bloque	PSNR	Tiempo de cómputo
1	40	43.871	2.01534
1	1024	43.9082	3.19154
2	20	39.0773	1.50334
2	683	39.1616	2.03111
5	10	32.1426	0.967969
5	342	32.2075	1.07513

Al parecer nuestra hipótesis de que el método podía funcionar mejor tomando la imagen de a pequeños bloques por tener muchos detalles, no era cierta. Esto se ve en los números dado que para los tres valores de k utilizados, obtenemos un PSNR mayor cuando procesamos la imagen con un solo bloque del tamaño de la misma. También se probó con valores de tamaño de bloque mucho más reducidos como 5x5 y aún así seguía dando mayor PSNR procesando la imagen con un solo bloque, aunque al igual que sucede con estos resultados, las diferencias de PSNR no fueron demasiado grandes.

Finalmente analizamos nuestra última imagen para Splines: Auto. Dado que el tamaño de esta imagen es el más grande, vamos a trabajar con más valores de k , que serán los siguientes: 2, 4, 6, 8 y 12. Dado que esta imagen tiene muchísimos detalles y, considerando los resultados obtenidos con la imagen Rosa,

esta vez decidimos tomar un tamaño de bloque más. De esta manera apuntamos a tener un tamaño chico, otro mediano-grande y el que siempre usamos del tamaño de la imagen, para poder ver si vuelve a suceder que gana el tamaño del bloque mayor o fue un resultado particular de la imagen anterior. Los tamaños de los bloques los vamos a mostrar explicitamente cuando veamos la tabla ya que varían según el k .

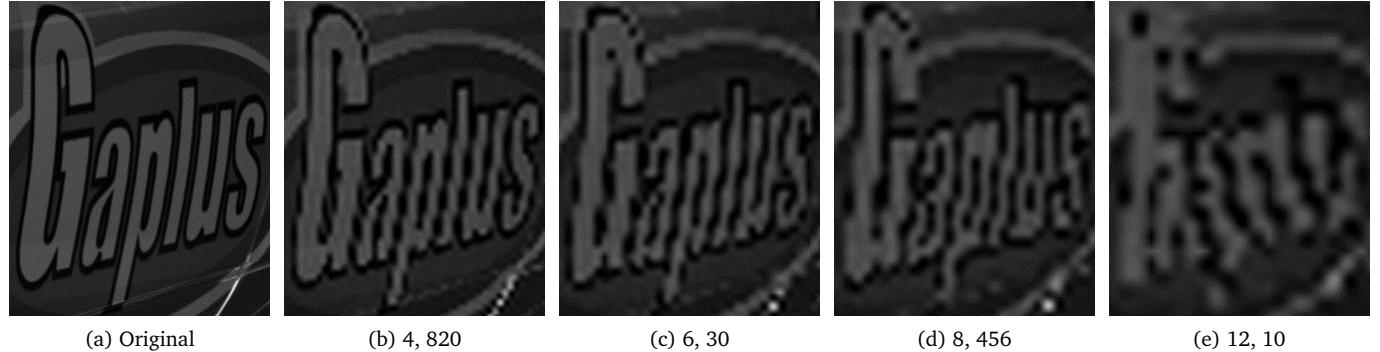


Figura 29: Comparación entre la imagen original y la procesada con distintos k . Debajo de cada imagen se encuentra su respectivo k y tamaño de bloque utilizado.

Una vez más, sucede que las diferencias entre las imágenes procesadas tanto con bloques chicos como medianos y un único bloque se llegan a apreciar bien recién para $k = 12$. Las diferencias con la imagen original hasta $k = 6$ se notan más que nada en las líneas claras que representan los reflejos, las cuales se empiezan a segmentar de alguna manera, alternando con el color oscuro del auto. Para $k = 8$ y $k = 12$ ya podemos ver que los bordes se empiezan a despixelar bastante, es fácilmente apreciable en los del parabrisas (sobre todo en el que está sobre el reflejo de la luz y hace que se mezcle la parte clara con la oscura) y en las letras del logo.

Para poder ver si efectivamente sucede lo mismo que con la imagen Rosa, con respecto a los bloques y el PSNR, a continuación mostramos las tablas correspondientes a esta imagen:

Auto $k = 2$		
Tamaño de bloque	PSNR	Tiempo de ejecución
50	38.3178	4.11688
300	38.3146	3.50407
1366	38.317	8.08269

Auto $k = 4$		
Tamaño de bloque	PSNR	Tiempo de ejecución
40	34.0937	2.63596
250	34.0984	2.44113
820	34.0993	4.97891

Auto k = 6		
Tamaño de bloque	PSNR	Tiempo de ejecución
30	31.806	2.07926
200	31.8033	1.51972
586	31.8026	3.8546

Auto k = 8		
Tamaño de bloque	PSNR	Tiempo de ejecución
20	30.2226	1.93031
150	30.2372	1.83401
456	30.2389	3.22826

Auto k = 12		
Tamaño de bloque	PSNR	Tiempo de ejecución
10	28.2243	2.01117
100	28.2178	1.44431
316	28.2196	2.69884

Al igual que sucedió con la imagen Rosa, fueron pocos los casos en donde tomar bloques de tamaño pequeño significó una mejor en el PSNR, exactamente sucedió en $k = 2, 6$. Las diferencias entre el tiempo de cómputo procesando la imagen con un bloque grande y procesándola con muchos de tamaño menor sigue siendo significativa y creciente cuando disminuye el k .

Como conclusión de toda esta extensa experimentación, podemos decir que cuando tenemos una imagen que podemos dividir en bloques cuadrados de igual tamaño (como lo fue Cuadrícula), conviene utilizar bloques precisamente de ese tamaño. Ahora bien si nuestra imagen es grande y sabemos poco sobre ella, a nivel PSNR da lo mismo si utilizamos muchos bloques o un bloque grande para procesarla, sin embargo es preferible utilizar bloques relativamente chicos porque, según lo observado en la experimentación, el tiempo de cómputo va a ser menor. En resumen, utilizar bloques de tamaño chico parece ser la mejor alternativa.

3.3.2. Bloques variables

Para comparar la performance cualitativa así como el tiempo de computo, de aplicar esta variante al método anterior, se trabajarán con las mismas imágenes que utilice el mismo. Como la cantidad de puntos que se van a utilizar para conformar un spline, tamaño del bloque, ahora está ligado a la tolerancia de diferencia con que se trabaje, α . Experimentaremos con distintos valores para cada imagen, empezaremos con una diferencia mínima de 40, aproximadamente una tolerancia del 16 %, e iremos incrementando este valor en 40 hasta una diferencia máxima de 200, 78 %, no optaremos por valores más bajo ya que se asemejaría a aplicar vecino más cercano, ni valores mayores ya que sería similar a utilizar el método

anterior tomando como bloque toda la imagen. Además trabajaremos con distintos valores de k por imagen para ver cómo repercute (si es que lo hace).

Una hipótesis que podemos elaborar es que a mayor k la imagen resultante difiere más de la original, ya que se pierden más datos.

Tanque k = 1		
Alpha	PSNR	Tiempo de cómputo
40	29.0891	0.13004
80	29.7871	0.10551
120	29.9202	0.10270
160	29.9207	0.09884
200	29.9207	0.10022

Figura 30: Imagen *Tanque* de 256x256.

Tanque k = 2		
Alpha	PSNR	Tiempo de cómputo
40	27.1585	0.09565
80	27.4977	0.06943
120	27.5592	0.06588
160	27.5584	0.06567
200	27.5584	0.06640

Figura 31: Imagen *Tanque* de 171x171.

Tanque k = 4		
Alpha	PSNR	Tiempo de cómputo
40	25.0083	0.06687
80	25.3644	0.04870
120	25.4196	0.04186
160	25.4209	0.04552
200	25.4209	0.04341

Figura 32: Imagen *Tanque* de 103x103.

La elección de la imagen *Tanque* estaba ligada a que presenta una uniformidad de color, se puede observar un paisaje con solo un objeto, el tanque, y a la vista la tonalidad del mismo no difiere mucho del fondo. Por lo que la particularidad de este método solo se aplicaría a valores de α bajos. Esto es lo que se puede observar en la figura 32, solo el valor $\alpha = 40$ es el que presenta mayor diferencia entre los distintos alphas. Sin embargo, al utilizar este valor de alpha se obtiene un mayor error, esto se puede deber a que como se trabaja con una diferencia muy baja no es muy restrictiva, por lo que estaría tomando como objetos y/o bordes distintos a partes de las imagen que no lo son y por lo tanto esta replicando valores erróneos. Como consecuencia se presentan una gran cantidad de irregularidades en la misma, algunas de ellas conocidas como artifacts, no solo se perciben en la mayor parte de la imagen si no que se acentúan sobre todo en el tanque, en su contorno, observándose un mayor pixelado en esta sección como se puede observar en la figura 33a. Otro análisis que se puede hacer a partir de estos valores es que, para k mayores el error aumenta. Esto se debe a que cuando se usan k de mayor valor el tamaño de la imagen a la que

se quiere hacer zoom es más pequeña y por lo tanto se tiene menos información, se puede observar al comparar las siguientes figuras donde para $k=4$ la forma y/o la claridad del tanque difieren de mucho que para $k = 1$. Como consecuencia de estos dos últimos análisis, a simple vista, podemos concluir que para cualquiera de los α se distingue que mientras agrandamos el k la imagen se pixela aun más. También se puede ver que las imágenes con $k = 1$, entre sí, no difieren de mucho lo mismo para $k = 2$, sin embargo para $k = 4$ recién con los últimos tres valores de α se logra una mejor resolución respecto de ese valor de k .

A continuación se presentan algunos de los casos descriptos anteriormente:

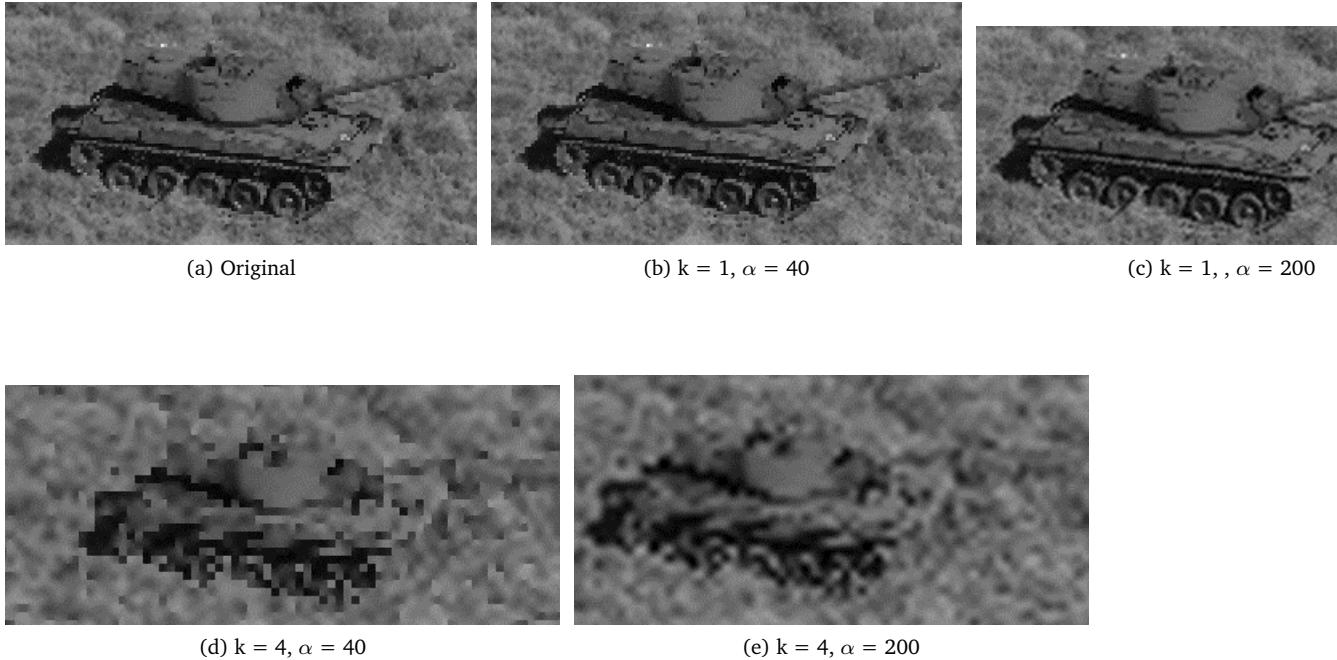


Figura 33: Comparación entre la imagen original y la procesada con algunos valores de k y α

Grises k = 1		
Alpha	PSNR	Tiempo de cálculo
40	34.8417	0.10531
80	34.9531	0.10530
120	35.5708	0.10583
160	36.2357	0.10046
200	37.3802	0.09850

Figura 34: Imagen Grises de 256x256.

Grises k = 2		
Alpha	PSNR	Tiempo de cómputo
40	31.9387	0.07164
80	32.0324	0.06495
120	32.5460	0.06931
160	33.0999	0.06441
200	34.0313	0.06476

Figura 35: Imagen *Grises* de 171x171.

Grises k = 4		
Alpha	PSNR	Tiempo de cómputo
40	30.0913	0.04642
80	30.1640	0.04716
120	30.5872	0.04666
160	31.0427	0.04474
200	31.7862	0.04337

Figura 36: Imagen *Grises* de 103x103.

Al igual que para la imagen anterior corroboramos que el ECM aumenta a medida que lo hace el k , sin embargo en esta caso no aumenta tanto comparados como por ejemplo para las figuras anteriores. Esto podría deberse al tipo de imagen con que ahora trabajamos, la cual son bloques de distintas tonalidades de gris donde no hay muchos detalles que se puedan perder. Algo que a partir de la figura se vuelve a repetir es que para valores de α mayores el ECM disminuye y por lo tanto el PSNR aumenta, originalmente la idea es que la particularidad de este método (hasta donde tomar el spline y que hacer entre el final de un spline y el comienzo del siguiente) debe ser aplicado cuando hay grandes cambios ahora podemos empezar a deducir el valor de esta diferencia. Aún más, empezamos a notar que esta variación tiene graves consecuencias cuando se impone una condición menos restrictiva obteniendo un mayor error cuadrático. En cuanto a lo que se puede analizar al observar las imágenes resultantes estas no poseen una gran diferencia cuando se las observa, para $k=4$ se pueden observar imágenes más ruidosas respecto de los otros k , pero en cuanto a comparar con los α las diferencias entre estos son casi imperceptibles.

Cuadricula k = 1		
Alpha	PSNR	Tiempo de cómputo
40	9.9147	0.21802
80	9.9147	0.22067
120	9.9147	0.22008
160	9.9147	0.21392
200	9.9147	0.21454

Figura 37: Imagen *Cuadricula* de 256x256.

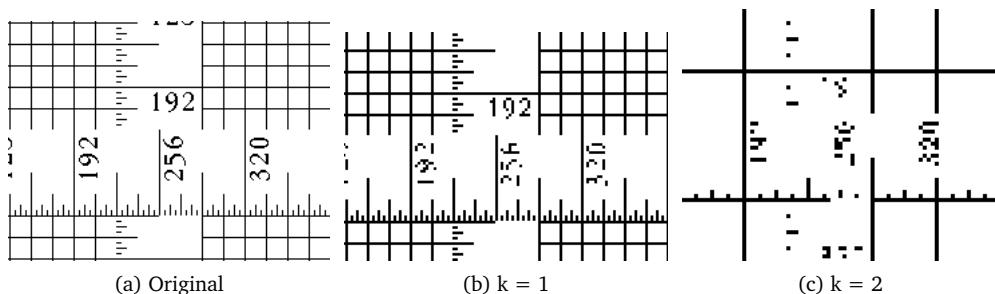
Cuadricula k = 2		
Alpha	PSNR	Tiempo de cómputo
40	8.6801	0.11451
80	8.6801	0.11160
120	8.6801	0.11131
160	8.6801	0.10601
200	8.6801	0.10855

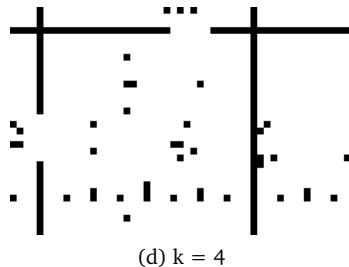
Figura 38: Imagen Cuadricula de 171x171.

Cuadricula k = 4		
Alpha	PSNR	Tiempo de cómputo
40	7.8399	0.06869
80	7.8399	0.07140
120	7.8399	0.06757
160	7.8399	0.06740
200	7.8399	0.06472

Figura 39: Imagen Cuadricula de 103x103.

La imagen *Cuadricula* posee un nivel de detalle mayor que las imágenes anteriores, en esta se observan una gran cantidad de líneas de color negro en un fondo blanco, y en algunas partes estas están muy próximas. Si reducimos mucho la imagen los detalles se podrían perder más fácilmente, aumentando el ECM. Cuando observamos los valores arrojados por las figuras 37, 38, 39 vemos que esto sucede, es más el error se incrementa en un rango no despreciable mientras que para los distintos valores de α obtenemos los mismos resultados. Al analizar esto, junto con lo que observamos en las imágenes obtenidas, podemos establecer que como solo interviene dos colores, el blanco y el negro, con valores 0 y 255 respectivamente, siempre vamos a caer en el caso de que un mismo spline no se utilice para toda una fila sino hasta cierto punto, sin importar el valor de α . En esta imagen es en las que más se puede apreciar como actúa la variación respecto al método original. En las imágenes obtenidas con $k = 1$ (figura 40b), se notan que las líneas tienen más grosor, esto es acción de la replicación de los valores que establece la idea de vecino más cercano. Para las imágenes obtenidas con los restantes k también se observa esto pero la imagen resultante difiere, y por mucho, de la original (la mayoría de las trazas de color negro se perdieron y fueron irrecuperables por el método) figuras 40c y 40d. A su vez trabajar con esta figura nos plantea un nuevo posible experimento a analizar, dado que es evidente la parte en la que actúa vecino más cercano e incluso llega a alterar la imagen, se podrían optar por aplicar otros métodos entre un spline o el otro.



Figura 40: Comparación entre la imagen original y la procesada con algunos valores de k

Piano k = 1		
Alpha	PSNR	Tiempo de cómputo
40	44.5710	0.84183
80	46.0033	0.83165
120	46.5206	0.85242
160	47.1249	0.82678
200	47.4640	0.83176

Figura 41: Imagen *Piano1* de 775x775.

Piano k = 2		
Alpha	PSNR	Tiempo de cómputo
40	41.9140	0.54356
80	43.1382	0.52312
120	43.6982	0.53191
160	44.0354	0.52499
200	44.4402	0.52178

Figura 42: Imagen *Piano2* de 517x517.

Piano k = 3		
Alpha	PSNR	Tiempo de cómputo
40	39.1148	0.43361
80	40.6090	0.40930
120	41.5049	0.41024
160	42.1322	0.40237
200	42.3513	0.39569

Figura 43: Imagen *Piano3* de 388x388.

Una de las particularidades de *Piano* era observar cómo se comporta nuestro método frente a imágenes que tenían algún tipo de efecto, en este caso reflejos y sobre todo uno tan sutil. Como el reflejo de esta imagen es muy claro, al aumentar el k creeríamos que los puntos a interpolar quedarían más claros eliminando dicho efecto. Además deseábamos que se siga manteniendo una clara distinción entre las secciones oscuras que conforman cada tecla respecto de las más claras. Si analizamos en cuanto a los valores de las figuras 41, 42 y 43 observamos que al variar el k , el PSNR disminuye, pero levemente,

en este caso como estamos trabajando con una imagen de mayor tamaño pudimos trabajar con varios k sin que se altere demasiado la imagen. Al Fijar un k e ir variando los α las imágenes son muy similares visualmente entre si y respecto de la imagen original. El efecto de reflejo sigue existiendo y visualmente casi sin modificaciones, lo que nos lleva a pensar que el método de spline es útil aun para imágenes con estos detalles. En lo que respecta a las diferencias entre las secciones de las teclas, estas siguen manteniendo sus bordes bien delimitados. En este tipo de imágenes donde la cantidad de valores distintos no se limita a una sola franja como en *Cuadricula*, sino que se encuentran en mayor cantidad, el método parece funcionar sin alteraciones visibles. En cambio sí fijamos un valor de alpha y variamos el k entonces se pueden observar algunos artifacts en los bordes de cada tecla, aunque en menor cantidad respecto de las anteriores imágenes.

A continuación se presenta una de las zonas en las que se presentaron las mayores diferencias con respecto a la imagen original. Como puede observarse las irregularidades no alteran demasiado a la imagen en este caso.

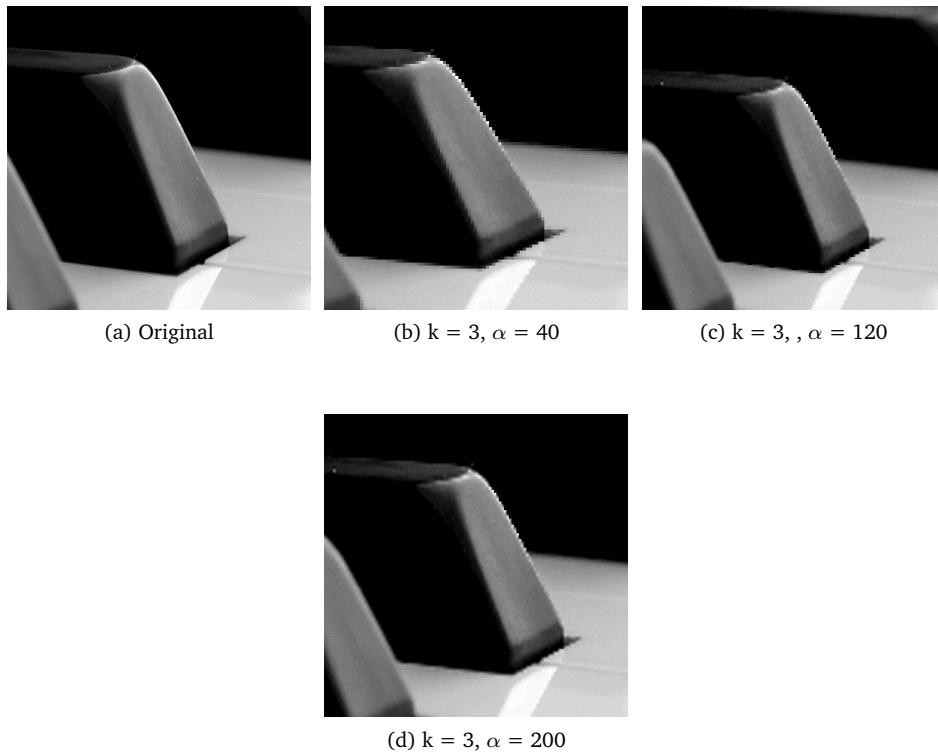


Figura 44: Comparación entre la imagen original y la procesada con $k = 3$ y α variable

Rosa $k = 1$		
Alpha	PSNR	Tiempo de cómputo
40	37.5912	1.56572
80	39.4595	1.46849
120	41.6560	1.46509
160	43.1872	1.42817
200	43.8159	1.43899

Figura 45: Imagen Rosa de 1024x1024.

Rosa k = 2		
Alpha	PSNR	Tiempo de cómputo
40	35.6237	1.04024
80	36.8667	0.95534
120	38.0043	0.94318
160	38.8049	0.90853
200	39.1056	0.90503

Figura 46: Imagen *Rosa* de 683x683.

Rosa k = 5		
Alpha	PSNR	Tiempo de cómputo
40	30.0870	0.64700
80	30.7894	0.57826
120	31.4542	0.55010
160	31.9094	0.53741
200	32.1540	0.53208

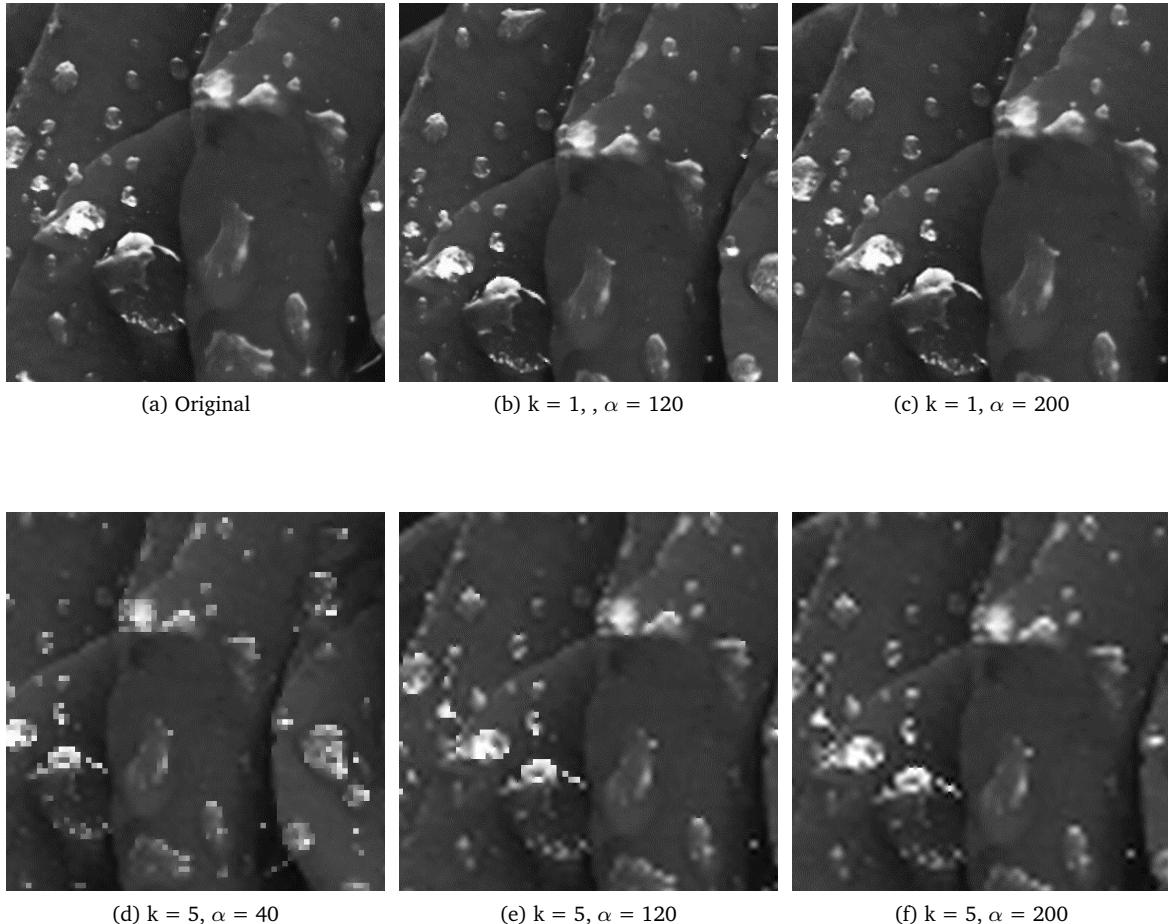
Figura 47: Imagen *Rosa* de 342x342.

La imagen *Rosa* presenta tonalidades de grises similares, por lo que creíamos que no se iba a ver muy alterada al variar el k o el valor de alpha. Sin embargo como se observa en las figuras 45, 45 y 45. Los valores difieren bastante entre cada k usada y alphas empleados. Las mayores diferencias visuales que encontramos, y que pueden ser las causantes de obtener estos valores, son entre las gotas de agua en la rosa. Estos últimos detalles son de una tonalidad opuesta al fondo en donde se encuentran y al ser de muy pequeño tamaño es mas dificultoso trabajar con los mismos. Al utilizar valores de alphas muy bajos, observamos que las gotas en algunos casos aumentan su tamaño, aunque levemente, esto se puede deber a que en esas zonas la variación del método actúa mas frecuentemente y como ya analizamos anteriormente (por ejemplo para la imagen *Cuadricula*) cuando los detalles son de menor tamaño el método no es muy eficaz. Sin embargo, pese a este cambio de tamaño, notamos que las tonalidades de las gotas son muy parecidas a la original, hecho que no ocurre al aumentar alpha. Cuando se aumenta el valor de alpha un mismo spline podría aplicarse tanto en las gotas como en el resto de la rosa, como tiene mas puntos en los que los valores son mayores (por ser tonalidades de grises oscuros) al parecer los bordes de las gotas se ven afectados fuertemente por estos, recién en el centro de las mismas resultan ser mas claros.

Para esta imagen pudimos ver que tomas valores de alpha muy grandes muy pequeños tienen consecuencias notables a la vista.

Como ocurrió en los casos anteriores al aumentar el valor de k la imagen fue perdiendo calidad y casi siempre es evidente entre los bordes de los distintos objetos que componen a una imagen, en este caso entre los pétalos de la rosa donde se evidencia un notable pixelado.

A continuación alguno de las diferencias mencionadas anteriormente:

Figura 48: Comparación entre la imagen original y la procesada con distintos k y α

Las siguientes dos imágenes fueron utilizadas primordialmente para comparar el tiempo de computo respecto de las anteriores, ya que estas son de mayor dimensión. Además estas nos van a permitir trabajar con una mayor cantidad de k .

Lago $k = 1$		
Alpha	PSNR	Tiempo de cálculo
40	40.2223	3.34901
80	42.3168	3.25231
120	43.6315	3.21029
160	43.8883	3.20526
200	43.9086	3.22257

Figura 49: Imagen *Lago* de 1525x1525.

Lago k = 2		
Alpha	PSNR	Tiempo de cálculo
40	38.3059	2.13227
80	38.7459	2.02789
120	38.9188	2.01046
160	38.9522	2.00238
200	38.9567	2.01118

Figura 50: Imagen *Lago* de 1017x1017.

Lago k = 3		
Alpha	PSNR	Tiempo de cálculo
40	36.1527	1.63916
80	36.5939	1.56956
120	36.7805	1.54919
160	36.8150	1.54711
200	36.8230	1.55647

Figura 51: Imagen *Lago* de 763x763.

Lago k = 5		
Alpha	PSNR	Tiempo de cálculo
40	34.4929	1.26164
80	34.9108	1.20746
120	35.1538	1.18800
160	35.2248	1.18297
200	35.2395	1.18572

Figura 52: Imagen *Lago* de 509x509.

Lago k = 7		
Alpha	PSNR	Tiempo de cálculo
40	33.6532	1.11109
80	34.1031	1.06358
120	34.4193	1.04979
160	34.5379	1.03440
200	34.5532	1.03910

Figura 53: Imagen *Lago* de 382x382.

Trabajando con mas valores de k, pudimos corroborar que se sigue cumpliendo que al aumentar el valor del mismo, el ECM aumenta y por lo tanto el PSNR disminuye. También observamos que no siempre se incrementa en la misma magnitud, las imágenes en las que existen secciones de tonalidades similares, se siguen manteniendo. En esta imagen aun aumentando el valor de k, las zonas del cielo y del agua permanecían, visiblemente, similares entre si y también con respecto a la imagen original; no sucedía lo

mismo para el sector del terreno donde si empezaron a aparecer artifacts, al igual que en la mayoría de los casos cuando se producían grandes cambios de tonalidad, en este caso entre las distintas superficies del terreno. Los artifacts presentes en esta imagen fueron en los bordes, apareciendo un notable pixelado, ejemplo 54d, mientras que en las secciones más oscura la imagen fue mucho mas ruidosa, 54e. Aún cuando se aumentaba el valor de alpha siendo que en otras imagenes esto a veces permitía algun tipo de corrección.

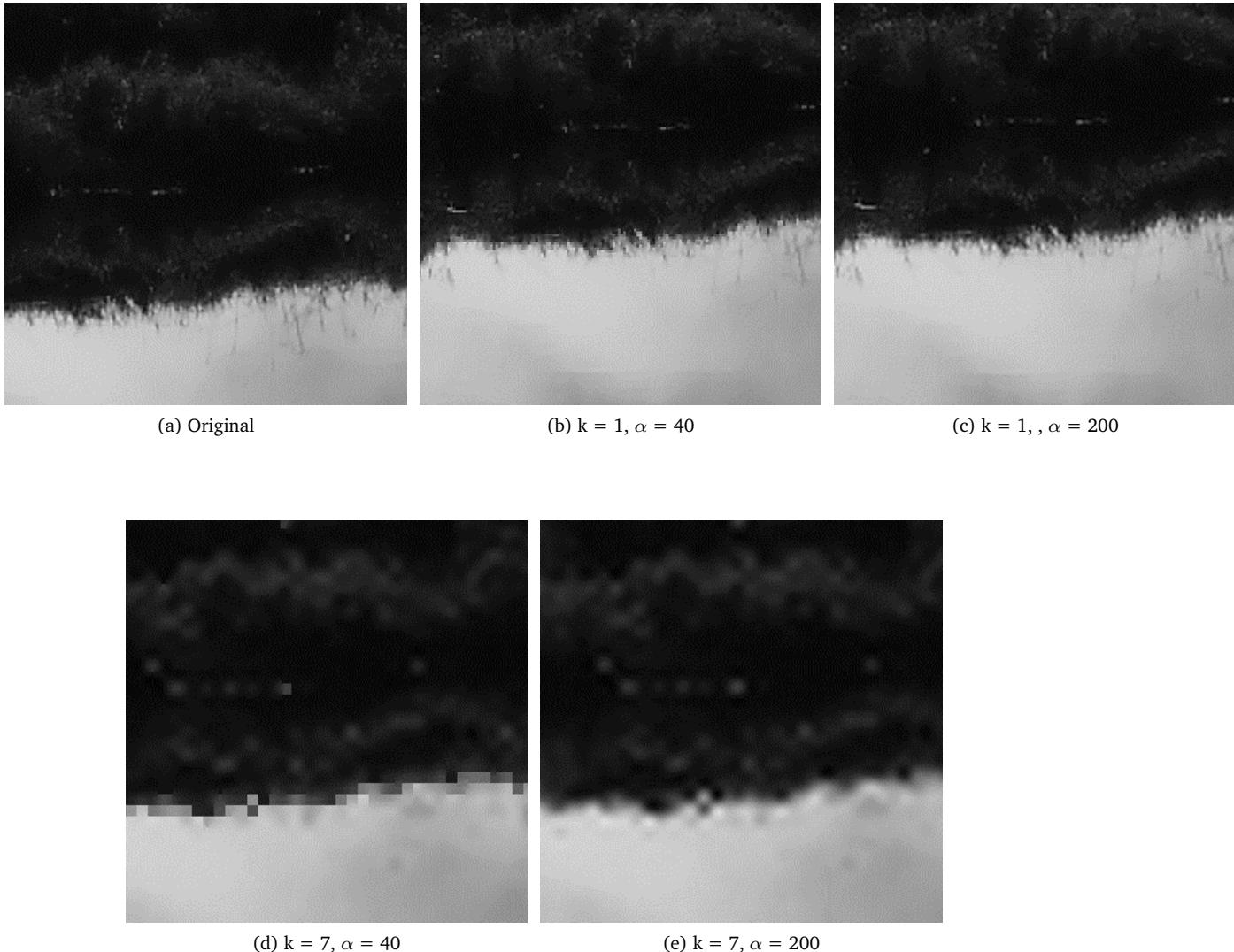


Figura 54: Comparación entre la imagen original y la procesada con variando k y α

Auto k = 2		
Alpha	PSNR	Tiempo de cómputo
40	35.6394	3.93213
80	36.2444	3.76378
120	36.9386	3.69821
160	37.5303	3.61576
200	37.9097	3.59944

Figura 55: Imagen *Auto* de 1366x1366.

Auto k = 4		
Alpha	PSNR	Tiempo de cómputo
40	31.9600	2.64685
80	32.3515	2.50842
120	32.8263	2.42699
160	33.2800	2.37932
200	33.6171	2.37221

Figura 56: Imagen *Auto* de 820x820.

Auto k = 6		
Alpha	PSNR	Tiempo de cómputo
40	29.9170	2.23046
80	30.2599	2.1101
120	30.6349	2.03897
160	30.9838	2.04038
200	31.2740	2.00317

Figura 57: Imagen *Auto* de 586x586.

Auto k = 8		
Alpha	PSNR	Tiempo de cómputo
40	28.6351	2.00131
80	28.9342	1.88967
120	29.2648	1.82881
160	29.5556	1.80021
200	29.7952	1.79442

Figura 58: Imagen *Auto* de 456x456.

Auto k = 12		
Alpha	PSNR	Tiempo de cómputo
40	26.7966	1.77177
80	26.9982	1.73024
120	27.2693	1.65314
160	27.5041	1.62439
200	27.7421	1.61716

Figura 59: Imagen *Auto* de 316x316.

Luego de todas las imágenes empleadas podemos deducir que la variación que establecimos, en general, resulta mas eficaz cuando la restricción es menor es decir a valores mayores de alphas. Si alpha es muy chico empieza a distinguir sectores que no son deseados, aunque el tipo de imagen con la que se trabaja influye notablemente. En *auto* los reflejos y las calcomanías sobre el automóvil comienzan a exagerarse a tal punto de obtener un gran pixelado. Esto último podría deberse a que como en las mayorías de las

imágenes al aumentar el valor de k , se pierde mucha información, ya que trabajamos con imágenes mas pequeñas. Sin embargo, en este caso aún para los valores de k mas bajos las imágenes obtenidas varían visualmente mucho respecto de la original y lo hace para los distintos valores alpha similarmente con lo ocurrido para la imagen *Rosa*. Las causas de estas alteración podrían deberse nuevamente a los detalles presentes en la imagen, como ya mencionamos para valores de α muy pequeños comienza comienza a distinguir mas fuertemente distintos sectores de la imagen pero para alpha mayores, en este caso donde los detalles se contraponen fuertemente con el fondo, comienzan a verse influidos por los puntos más oscuros.

A continuación se observa alguno de los casos mencionados anteriormente, por ejemplo como aún para valores bajos de k , alguno de los bordes de color blanco comienzan a tener una forma cuadriculada en vez de su forma original y como este pixelado se va haciendo más notable.

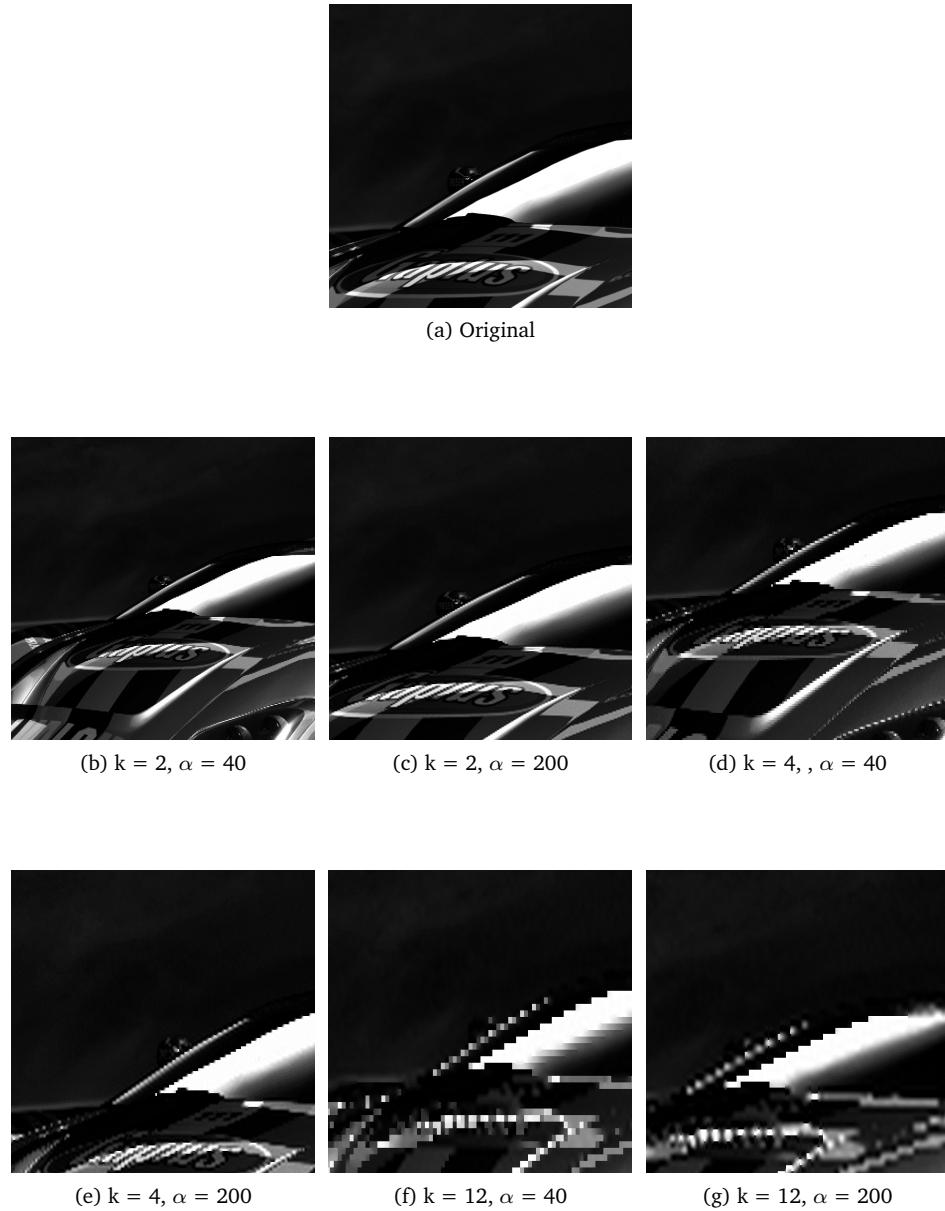


Figura 60: Comparación entre la imagen original y la procesada con distintos k y α

3.3.3. Tiempo de cómputo

A partir de los resultados obtenidos podemos deducir que el tiempo de computo para este método no depende del tamaño de la imagen con la que se este trabajando ya que a veces la reconstrucción de

una imagen cuyo tamaño original es menor tarda más que una de mayor dimensión. Pero, si se observa que en general el tiempo aumenta para valores de α menor, lo que posiblemente se deba a que en estos casos se realiza una mayor variación en la forma de conseguir el valor de un pixel, y dichas fluctuaciones repercuten notablemente en el tiempo de cómputo.

3.3.4. Comparación de variantes

A partir de las experimentaciones realizadas para el método de splines tomando tanto un tamaño fijo del bloque o siendo este variable. Dado que siempre observamos que a mayor PSNR obtenemos visualmente una imagen con mejor calidad y/o similar a la original, procedimos a realizar una tabla comparativa entre los distintos valores de PSNR obtenidos.

Tanque		
K	PSNR Bloques variables	PSNR Bloques fijos
1	29.9207	29.9234
2	27.5592	27.5685
4	25.4209	25.4398

Grises		
K	PSNR Bloques variables	PSNR Bloques fijos
1	37.3802	37.4554
2	34.0313	34.0659
4	31.7862	31.8087

Cuadrícula		
K	PSNR Bloques variables	PSNR Bloques fijos
1	9.9147	14.1944
2	8.6801	10.0905
4	7.8399	9.0206

Piano		
K	PSNR Bloques variables	PSNR Bloques fijos
1	47.4640	47.8024
2	44.4402	44.8576
3	42.3513	42.8922

Rosa		
K	PSNR Bloques variables	PSNR Bloques fijos
1	43.8159	43.9082
2	39.1056	39.1616
5	32.1540	32.2075

Lago		
K	PSNR Bloques variables	PSNR Bloques fijos
1	43.9086	43.9126
2	38.9567	38.9604
3	36.8230	36.8325
5	35.2395	35.2486
7	34.5532	34.5746

Auto		
K	PSNR Bloques variables	PSNR Bloques fijos
2	37.9097	38.3178
4	33.6171	34.0993
6	31.2740	31.8060
8	29.7952	30.2389
12	27.7421	28.2243

Lo que podemos observar de las comparaciones anteriores es que el método de Splines con bloques fijos obtuvo los mayores valores de PSNR, y por lo tanto las imágenes obtenidas menos diferencias respecto de la original. Este resultado comenzaba a deducirse a lo largo de la experimentación de Spline 2, cuando observábamos que cuando el valor de α aumentaba obteníamos un PSNR mayor (y como bien mencionamos a medida que el valor de alpha aumenta la variación del método se asemeja al método de spline con bloques de tamaño igual a la imagen). Las causas de este resultado podríamos atribuirlas a la decisión de la variante más importante introducida, que fue el aplicar la idea de vecino mas cercano entre último punto de un spline y el comienzo del otro. La idea era tratar de mantener los bordes bastante delimitados, sin embargo esto trajo consecuencias en el resto de la imagen y solo era mas eficaz cuando las diferencias fueron mayor a 200. Una forma de probar obtener mejores resultados (si es que es posible) sería aplicar otro método en vez del de vecino, ya que el mismo es uno de los métodos mas simple (aunque creímos que esta combinacion podría generar mejores resultados) y podría estar más susceptible a errores. En conclusión con los resultados presentes podemos atribuir que para obtener mejores resultados cualitativos el método de spline con bloques de tamaño fijo es superior al de tamaño variable. Sin embargo una dificultad que se nos presenta en el primero método es ¿que tamaño de bloque utilizar? ya que como se puede ver en 3.3.1 los mejores resultados no siempre se obtuvieron para bloques de tamaños similares, en algunos casos el tamaño elegido fue muy bajo y en otro casi el tamaño de la imagen total. Como en algunas imágenes la diferencia es muy notable (por ejemplo para *Cuadricula*), aun tener que buscar entre distintos tamaño del bloque seria mucho mas eficaz (en cuanto a calidad) que aplicando el segundo método evaluando en tan solo $\alpha = 200$ o algún valor superior (que a partir de los resultados obtenidos creemos que es el mejor valor a tomar). Sin embargo si queremos ponderar el tiempo de computo quizás sea preferible optar por el segundo método donde los tiempos de cómputos en general fueron menores, bastando con una sola ejecución y en general, salvo excepciones, los PSNR

no variaron en un amplio rango. Para la experimentación final elegiremos trabajar con el método Spline con bloques fijos ya que es el que mejor resultados cualitativos obtuvo y además es lo que ponderaremos en dicha experimentación.

3.4. Comparando los 3 métodos

Dado que hasta ahora estuvimos comparando las distintas variantes de cada método particular, la idea ahora es armar un nuevo conjunto de imágenes y probar la mejor variante de cada método (en base a la experimentación anterior) con este conjunto. El conjunto se va a definir utilizando dos imágenes de las que usamos con Splines (Lago y Piano), una de las que usamos en bilineal (Imagen6, de ahora en más Mono) y una última de las que usamos en Vecino más cercano (Venus). El objetivo de esta última experimentación es elegir de alguna manera la mejor variante entre las mejores. Elegimos el nuevo conjunto de imágenes en base a imágenes ya usadas debido a que esto nos permite reutilizar ciertos datos y además, dado la cantidad de imágenes usadas, creemos que no aportaría demasiado agregar nuevas.

En el caso de la variante de bloques fijos, para la experiencia con las imágenes Mono y Venus probamos distintos tamaños de bloque y nos quedamos con el que mejor PSNR obtuvimos. A continuación, mostramos las dos tablas correspondientes a estos datos:

k	Tamaño del bloque	PSNR
1	20	22.5945
1	50	22.6002
1	100	22.5923
2	30	20.2488
2	70	20.2438
2	100	20.2425
4	10	18.8017
4	40	18.7371
4	70	18.7327

Tabla 23: Tabla para Mono

k	Tamaño del bloque	PSNR
1	300	38.2384
1	400	38.2386
1	500	38.2394
3	100	31.6407
3	150	31.64
3	200	31.6412
7	10	28.4955
7	50	28.4538
7	100	28.4476

Tabla 24: Tabla para Venus

Ahora presentaremos las tablas correspondientes a la comparación entre las mejores variantes de cada método.

k	Vecino mas cercano		Bilineal		Splines	
	PSNR	Tiempo	PSNR	Tiempo	PSNR	Tiempo
1	22.7730	0.06488	23.1450	0.06606	22.6002	0.13694
2	20.3643	0.04333	20.9392	0.05184	20.2488	0.08457
4	18.7550	0.02919	19.4993	0.03871	18.8017	0.09307

Tabla 25: Tabla de PSNR y tiempo de computo de `mono.tiff` ejecutadas en las mejores variantes de cada método de interpolación variando el k en 1, 2 y 4.

k	Vecino mas cercano		Bilineal		Splines	
	PSNR	Tiempo	PSNR	Tiempo	PSNR	Tiempo
1	43.3944	0.56738	48.2636	0.55920	47.8024	1.87312
2	35.7378	0.31294	44.9598	0.37904	44.8576	0.60743
3	37.6288	0.25222	42.7730	0.30192	42.8922	0.42864

Tabla 26: Tabla de PSNR y tiempo de computo de `Piano.tiff` ejecutadas en las mejores variantes de cada método de interpolación variando el k en 1, 2 y 3.

k	Vecino mas cercano		Bilineal		Splines	
	PSNR	Tiempo	PSNR	Tiempo	PSNR	Tiempo
1	41.0961	2.25588	43.8323	2.13407	43.9126	7.16835
3	35.8696	0.96663	37.6747	1.13976	38.9604	2.34028
5	34.2581	0.75247	36.0974	1.10317	36.8325	1.70029

Tabla 27: Tabla de PSNR y tiempo de computo de `Lago.tiff` ejecutadas en las mejores variantes de cada método de interpolación variando el k en 1, 3 y 5.

k	Vecino mas cercano		Bilineal		Splines	
	PSNR	Tiempo	PSNR	Tiempo	PSNR	Tiempo
1	35.42	3.85530	37.7469	3.91639	38.2394	6.64268
3	29.98	1.68882	32.1074	2.10631	31.6412	2.91500
7	27.16	1.26326	29.0710	1.64526	28.4955	3.12556

Tabla 28: Tabla de PSNR y tiempo de computo de `venus.tiff` ejecutadas en las mejores variantes de cada método de interpolación variando el k en 1, 3 y 7.

A partir de los resultados obtenidos al comparar distintos métodos, nos encontramos con el hecho de que, para las imágenes con las que trabajamos, el método bilineal fue el que obtuvo en la mayoría de los casos mayor PSNR, y como consecuencia un menor ECM, cuando esto no sucedió entonces el método de Splines se sobrepuso.

Pese a que creímos que el método mas eficaz iba a ser el de Spline, esto no ocurrió. Primero analizemos lo sucedido con el método de vecino mas cercano, en este caso se cumplió lo que intuimos, que no iba a ser mejor que los otros dos métodos para una imagen. Esto se puede deber a que es el método mas simple, el mismo no tiene un análisis fuerte a la hora de asignar el valor de un pixel desconocido. Funciona de manera golosa, donde el valor de un pixel tan solo va a ser igual al valor del pixel mas cercano, establecida por una vecindad v de 4 pixel. Lo cual evidencia un notable error, que es creer que para cada píxel p del conjunto de píxeles a determinar utilizando v , va a existir en v otro píxel p' (el más cercano a p) con un valor igual o muy parecido al que realmente debería tener p . Sobre todo por el tamaño del a vecindad ya que en realidad cada píxel puede tomar 256 valores distintos. Por esto es que los métodos de Spline o Bilineal van a dar mejores resultados, ya que construyen funciones continuas en

base a los píxeles conocidos y esto genera un rango mayor de valores posibles. Sin embargo una particularidad que se puede observar en la tabla Tabla 25 es que para $k=1$ el método de vecino más cercano obtiene un PSNR mayor que el de Spline. Esto se puede principalmente a dos factores primero, como se esta trabajando con $k = 1$ no hay mucha pérdida de información ya que al reducir la imagen si estamos trabajando con la fila_i cuyos pixels son P_{ij} , con $1 \leq j \leq n$ (n cantidad de columnas) entonces solo se retiraron los P_{ij} tal que j es numero par, quedando disponible pixels que no están muy alejados entre si. El segundo hecho influyente se debe a que como para este caso se emplea la variación de vecino más cercano que toma el promedio de los valores de la vecindad definida, si existe una cierta distribución de las tonalidades (como ocurre en este caso en la imagen *Mono*) entonces tiene sentido que aquellos P_{ij} retirados tengan una tonalidad similar que a la de sus vecinos, pudiendo ser muy similar al promedio de estos.

Con respecto a la comparación entre el Método bilineal y el de Splines, fue aca cuando obtuvimos resultados contrarios a lo esperado. A la hora de analizar las causas de este hecho un posible factor influyente es que el método de Spline, es el que mayor cálculos realiza para asignar el valor a cada pixel, dado que se trabaja con aritmética finita cada cálculo es susceptible a tener un determinado margen de error y, si además cada resultado obtenido se utiliza en otros cálculos entonces el error se propaga y aumenta. Por lo que, con respecto a los métodos la implementación de Spline posee esta gran desventaja. En algunos casos este error puede ser despreciable comparado con los resultados obtenidos con otro método que utiliza menos cálculos pero, donde el valor que se elige para cada píxel está más alejado del que realmente debería usar; lo que sucede por ejemplo con Spline y Vecino más cercano, salvo la excepción de la tabla ??, ahora que profundizamos sobre una de las desventajas que posee el metodo de Spline podemos complementar esa excepción ocurrida al decir que en este caso el error cometido no fue despreciable. Otro factor influyente es el tipo de imagen con la que se trabaje, los cambios de tonalidad en la misma también pueden influir notablemente. La idea básica de ambos métodos, se basa en que si se tienen dos puntos conocidos P_i y P_k y se quieren calcular los valores P_t con $i < t < k$, el método bilineal trazará una recta entre P_i y P_k donde los valores de los P_t serán definidos por los puntos generados por la pendiente de la misma. Mientras que en Splines los puntos serán definidos por un polinomio cúbico, el cual permite que los valores se tomen de manera menos estricta, considerando un mayor distribución de los mismos.

En imágenes que posean una distribución de tonos similar, los resultados obtenidos por los métodos de Splines y Bilineal no deberían variar en demasía pero, dado el problema aritmético que se presenta en Splines, esto hace que interfiera de manera no despreciable causando el error observable. En cambio, si en la imágenes se presentan facciones de tonos muy variado entonces, según esta idea el método de Splines debería ser mejor y vimos que esto sucedió en la imagen *Lago* donde para todos los k utilizados el método de Spline fue superior. El mayor problema pudo estar en los bordes entre las secciones del cielo, la tierra y el lago (siendo la segunda de una tonalidad totalmente opuesta a las restantes). En el método Bilineal si se quiere calcular el valor de un pixel muy cercano a los bordes, se verá fuertemente influenciado por un valor muy bajo y otro muy alto, alterando el valor del pixel respecto al que debería tener. En cambio en Splines este valor depende de más puntos y como tiene una traza mas suave nos va a permitir tener una mejor aproximación. Las imágenes con las que trabajamos nos permitieron plantear estas teorías luego de observar los resultados obtenidos, una posible experimentación adicional sería trabajar con mas imágenes que presenten similares características a la de imagen *Lago* y ver si esto sigue sucediendo.

4. Conclusiones

En el presente trabajo pudimos aplicar los conceptos de interpolación numérica para hacer zoom en imágenes. Aprendimos las diferencias fundamentales de los distintos métodos de interpolación a nivel teórico y observamos su comportamiento realizando una experimentación exhaustiva sobre conjuntos de distintos tipos de imágenes. El hecho de trabajar específicamente con imágenes nos permitió implementar ciertas variaciones de los métodos conocidos y mediante la experimentación realizada efectuamos comparaciones entre estas variantes para elegir la que mejores resultados dió en cada conjunto. La ex-

perimentación también hizo que nos diéramos cuenta que, en ocasiones, un método tan simple como vecino más cercano o interpolación bilineal puede dar mejores resultados a nivel calidad que un método mucho más sofisticado como splines. Esto es un claro ejemplo del valor que tiene la experimentación ya que de alguna manera nos salva de cometer la equivocación de elegir un método porque posee una base teórica más compleja cuando en realidad existen otros más simples que se adecúan mejor a nuestras necesidades. Además de que los métodos que son conceptualmente más sofisticados muchas veces están acompañados por un costo computacional mayor, como es el caso de splines.

El análisis cualitativo de las imágenes procesadas hizo que podamos saber cómo se comportan las distintas variantes implementadas de cada método en la práctica. Luego, el análisis cuantitativo del tiempo de cómputo y el PSNR nos permitió tanto complementar como relacionar las observaciones visuales realizadas en una primera instancia. Esto último fue muy importante cuando las diferencias entre distintos resultados eran imperceptibles a la vista. Además fue importante a la hora de elegir configuraciones, en las implementaciones que requerían ciertos parámetros adicionales, de acuerdo al PSNR y tiempo de cómputo. En un futuro nos gustaría implementar los métodos de este trabajo práctico en videos también y observar sus resultados.

Referencias

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

5. Apéndices

Apéndice A: Enunciado

Métodos Numéricos
Primer Cuatrimestre 2015
Trabajo Práctico 3



Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Marche un telebeam Don Niembraaaaa...



Introducción

Según se ha publicado en algunos medios deportivos de cuestionable credibilidad, el Comité Ejecutivo de la Asociación de Fútbol Argentina (AFA) quiere, con el fin de reconquistar los espacios de poder otra vez ostentados, marcar tendencia en la incorporación de tecnología de última generación para la resolución de situaciones conflictivas durante los partidos. Para ello se busca dar el primer paso mediante el desarrollo de un prototipo que permita decidir en tiempo real, mediante una imagen de la televisión, si la pelota traspasó o no la línea de gol. El objetivo ulterior de semejante empresa es destrozar al sistema utilizado durante la Copa del Mundo 2014, principalmente por sus elevados costos de implementación y mantenimiento.

El Equipo de Desarrollos de Métodos Numéricos (EDMN) fue contactado para hacerse cargo del desafío, teniendo en nuestras manos el futuro de un negocio millonario y, por qué no, eventualmente la posibilidad de llegar nuevamente a la final de un mundial. Como propuesta, el prototipo en *Fase 0* se basará en la utilización de técnicas de interpolación aplicadas al procesamiento de señales, en particular para el re-escalamiento de imágenes de alta definición.

Definición del problema y metodología

Para resolver el problema planteado en la sección anterior, se considera el siguiente contexto. Dada una imagen de $m \times n$, con $i = 1, \dots, m$ y $j = 1, \dots, n$, en escala de grises, se busca obtener un re-escalamiento de la misma de un factor f predeterminado de antemano. En particular, consideraremos solamente agrandar la imagen, tomando $f > 1$, y para simplificar algunas cuestiones técnicas menores asumiremos que recibimos como parámetro un número $k \in \mathbb{N}_{>0}$ que denota la cantidad de filas (columnas) que serán agregadas entre dos filas i e $i+1$, $i = 1, \dots, m-1$ (columnas j y $j+1$) de la imagen original. Luego, el ratio f quedará automáticamente determinado por el cociente $\frac{(k+1)*(m-1)+1}{m}$. Las figuras 61 y 62 muestran la transformación sobre un ejemplo correspondiente a una imagen de 3×3 y tomando $k = 2$.

1	2	3
4	5	6
7	8	9

Figura 61: Imagen original

1			2			3
4			5			6
7			8			9

Figura 62: Imagen re-escalada

El problema a resolver consiste en determinar cómo llenar los casilleros grises, es decir, aquellos que no contienen la información original de la imagen. Para ellos, se propone considerar al menos los siguientes tres métodos:

1. *Vecino más cercano*: Consiste en llenar aquellas posiciones correspondientes a nuevos píxeles replicando los valores de alguno de los píxeles que se encuentran en un vecindario de la posición en consideración.
2. *Interpolación bilineal*: Consiste en llenar los píxeles utilizando interpolaciones lineales entre píxeles consecutivos de la imagen original, primero completando aquellas posiciones correspondientes a filas y columnas con información original de la imagen y, luego, llenando el resto. También es

posible interpretar el método como uno de dos fases, donde primero se aplica sobre las filas y luego, sobre la matriz resultante, se aplica por columnas (o viceversa).

3. *Interpolación por Splines:* Similar al anterior, pero considerando interpolar utilizando splines y tomando una cantidad de píxeles mayor. Una alternativa a considerar es tomar la información de bloques de un tamaño fijo (por ejemplo, $4 \times 4, 8 \times 8$, etc.), con el tamaño de bloque a ser determinado experimentalmente.

Cada método tienen sus propias características, ventajas y desventajas particulares. Para realizar un análisis cuantitativo, llamamos I a la imagen real (ideal) que deberíamos obtener con nuestro algoritmo, y sea \bar{I} la imagen efectivamente reconstruida. Consideramos entonces dos medidas, directamente relacionadas entre ellas, como el *Error Cuadrático Medio* (ECM) y *Peak to Signal Noise Ratio* (PSNR), denotados por $\text{ECM}(\bar{I})$ y $\text{PSNR}(I, \bar{I})$, respectivamente, y definidos como:

$$\text{ECM}(I, \bar{I}) = \frac{1}{mn} \sum_{i=1}^m \sum_{j=1}^n |I_{ij} - \bar{I}_{ij}|^2 \quad (12)$$

y

$$\text{PSNR}(I, \bar{I}) = 10 \log_{10} \left(\frac{255^2}{\text{ECM}(I, \bar{I})} \right). \quad (13)$$

En conjunto con los valores obtenidos para estas métricas, es importante realizar un análisis conjunto el tiempo de ejecución de cada método y los denominados *artifacts* que produce cada uno de ellos. Se denomina *artifact* a aquellos errores visuales resultantes de la aplicación de un método o técnica. La búsqueda de este tipo de errores complementa el estudio cuantitativo mencionado anteriormente incorporando un análisis cualitativo (y eventualmente subjetivo) sobre las imágenes generadas.

Enunciado

Se pide implementar un programa en C o C++ que implemente como mínimo los tres métodos mencionados anteriormente, y que dada una imagen y un valor k aplique estas técnicas para re-escalar la misma. A su vez, es necesario explicar en detalle cómo se utilizan y aplican los métodos descriptos en 1, 2 y 3 en el contexto propuesto. Los grupos deben a su vez plantear, describir y realizar de forma adecuada los experimentos que consideren pertinentes para la evaluación de los métodos, justificando debidamente las decisiones tomadas y analizando en detalle los resultados obtenidos. Sumado a los experimentos planteados para la evaluación de los métodos, se pide que cada grupo utilice al menos dos casos de prueba con imágenes de una resolución considerable (digamos, mayor a 1024×1024 o 2048×2048 como mínimo).

El formato de archivos, la modalidad de ejecución y el esquema de experimentación queda abierto a elección de cada grupo, siendo extremadamente importante elegir un formato adecuado y, además, proveer en la entrega una descripción detallada con las instrucciones para la ejecución del programa y los pasos a seguir para replicar los experimentos realizados. Además, se debe incluir un apéndice donde se detalle cómo evaluaron la correctitud de toda la implementación.

Apéndice B: Validación de la implementación y código fuente relevante

Para verificar que los algoritmos implementados operen como esperamos, además de la misma observación de la imagen, lo cual muchas veces resultó útil, se utilizó una carpeta aparte llamada "validacion_tp3" con una versión del trabajo práctico que nos permitía correr matrices pequeñas. Podemos encontrar en la misma una instancia de prueba llamada "matrizEntrada.txt" con sus respectivas salidas para cada método. Se utilizaron otras instancias, pero no están documentadas, ya que se fueron eliminando en el proceso. Para verificar de manera más efectiva la correctitud de los métodos, esta versión toma como entrada matrices de tipo unsigned int, y como salida matrices tipo double. El proceso de constatar que la salida cumple con nuestras expectativas se hace a mano, y en ocasiones solo sobre una parte del resultado.

Para el caso de splines también se verificó la correctitud de la función dame_Spline imprimiendo en pantalla los coeficientes obtenidos para casos conocidos de antemano.

Método Vecino Mas cercano: Tomando como vecindad para un P_{ij} los 4 vecinos mas cercano respecto de la imagen original

```
void origin(vector<vector<int> > &expandida, int k)
{
    int dato_f, dato_c;
    for(int i = 0; i < expandida.size(); i++)
    {
        dato_c = 0;
```

```
        for(int j = 0; j < expandida[i].size() ; j++)
        {
            if(expandida[i][j] != -1) {dato_f = i; dato_c = j;}
            else
            {
                if(j == dato_c + k + 1) {dato_c = j;}
                expandida[i][j] = mas_cercano(k, dato_c, dato_f, i, j, expandida);
            }
        }
    }

int mas_cercano(int k, int dato_c, int dato_f, int i, int j, vector<vector <int> > &expandida)
{
    int datoSig_c = dato_c + k + 1;
    int datoSig_f = dato_f + k + 1;
    if(dato_f == i)
    {
        if(abs(j - dato_c) < abs(j - datoSig_c)) {return expandida[dato_f][dato_c];}
        else {return expandida[dato_f][datoSig_c];}
    }
    if(dato_c == j)
    {
        if(abs(i - dato_f) < abs(i - datoSig_f)) {return expandida[dato_f][dato_c];}
        else {return expandida[datoSig_f][dato_c];}
    }
    else
    {
        int a = (dato_c - j)*(dato_c - j) + (dato_f - i)*(dato_f - i);
        int b = (datoSig_c - j)*(datoSig_c - j) + (dato_f - i)*(dato_f - i);
        int c = (datoSig_f - i)*(datoSig_f - i) + (dato_c - j)*(dato_c - j);
        int d = (datoSig_f - i)*(datoSig_f - i) + (datoSig_c - j)*(datoSig_c - j);
        int cercano = min(min(a, b), min(c, d));

        if(cercano == a) {return expandida[dato_f][dato_c];}
        else
        {
            if (cercano == b) {return expandida[dato_f][datoSig_c];}
            else
            {
                if(cercano == c) {return expandida[datoSig_f][dato_c];}
                else
                {
                    return expandida[datoSig_f][datoSig_c];
                }
            }
        }
    }
}
```

Método Bilineal original:

```
void originBilineal(vector<vector<int> > &expandida, int k)
{
    //Primero interpolo por filas, de a k filas
    int dato_c;
```

```
for(int i = 0; i < expandida.size(); i = i+k+1)
{
    dato_c = 0;
    for(int j = 0; j < expandida[i].size() ; j++)
    {
        if(expandida[i][j] != -1 && i != expandida.size() - 1) {dato_c = j;}
        else
        {
            if(j == dato_c + k + 1 && j != expandida[i].size() -1) {dato_c = j;}
            expandida[i][j] = calculo_bilineal_por_filas(k, dato_c, i, j, expandida);
        }
    }
}

//Ahora interpo por columnas, de a una fila
int dato_f;
for(int j = 0; j < expandida[0].size(); j++)
{
    dato_f = 0;
    for(int i = 0; i < expandida.size() ; i++)
    {
        if(expandida[i][j] != -1 && j != expandida[0].size() - 1) {dato_f = i;}
        else
        {
            if(i == dato_f + k + 1 && i != expandida.size() -1) {dato_f = i;}
            expandida[i][j] = calculo_bilineal_por_columnas(k, dato_f, i, j, expandida);
        }
    }
}
}

int calculo_bilineal_por_filas(int k, int dato_c, int i, int j, vector<vector <int> > &expandida)
{
    int limite_c = expandida[0].size() - 1;
    int datoSig_c;

    if(dato_c + k + 1 <= limite_c) {datoSig_c = dato_c + k +1;}
    else {datoSig_c = limite_c;}

    int res = (int)((double)expandida[i][dato_c] + ((double)expandida[i][datoSig_c] -
    (double)expandida[i][dato_c])/((double)(k+1))*((double)j-(double)dato_c));

    if(res < 255) {return res;}
    else {return 255;} //no estoy seguro de esto
}
```

Método Spline con tamaños fijos.

```
void MetodoSpline(Bloque b, vector<vector<int> >& imagen, int k)
{
vector<Spline> splinesPorFila, splinesPorColumna;

//me armo un spline por cada fila que contiene pixeles originales
for(int i = b.desde; i < b.desde + b.tam; i += k + 1)
{
    vector<int> x;
```

```

vector<double> a;

for(int j = b.desdej; j < b.desdej + b.tam; j += k + 1)
{
    x.push_back(j);
    a.push_back(imagen[i][j]);
}

splinesPorFila.push_back(dame_Spline(x, a, k));
}

for(int i = b.desdei; i < b.desdei + b.tam; i += k + 1) //salto de a k + 1
//filas para no tocar las que tienen todos -1
{
int p = 0; //p va a ser el punto "anterior" al punto que quiero calcular usando interpolacion

for(int j = b.desdej + 1; j < b.desdej + b.tam; j += k + 1, p++) //salto de a k + 1 columnas
//para no tocar los pixeles originales, con el proximo for voy rellenando los pixeles nuevos
//entre cada par de pixeles originales
{
//como voy saltando d a k + 1 filas y para calcular los splines por fila hicimos lo mismo,
// divido por k + 1 para obtener el spline correspondiente a la fila con la que estoy trabajando
    double a_j = splinesPorFila[(i - b.desdei) / (k + 1)].a[p];
    double b_j = splinesPorFila[(i - b.desdei) / (k + 1)].b[p];
    double c_j = splinesPorFila[(i - b.desdei) / (k + 1)].c[p];
    double d_j = splinesPorFila[(i - b.desdei) / (k + 1)].d[p];

    for(int t = 0; t < k; t++)
    {
        int interpolacion = a_j + b_j * (t + 1) + c_j * (t + 1) * (t + 1)
        + d_j * (t + 1) * (t + 1) * (t + 1);

        if(interpolacion > 255)
            interpolacion = 255;
        if(interpolacion < 0)
            interpolacion = 0;

        imagen[i][j + t] = interpolacion;
    }
}
}

//Ahora es simetrico, como por cada fila donde habia pixeles originales ahora estan calculados
//todos los pixeles nuevos entre cada par de pixeles originales, faltan calcular
//los pixeles de las filas que estan entre cada par de filas de pixeles
//originales. O sea las filas que tienen todos -1.
//Para eso me armo un spline por columna y despues el recorrido y demas es todo simetrico
//al paso anterior.
}

```

```

Spline dame_Spline(vector<int>& x, vector<double>& a, unsigned int k)
{
    unsigned int n = x.size() - 1;
    double h_i = k + 1;
    vector<double> alpha(n + 1, 0);

```

```
for(int i = 1; i < n; i++)
    alpha[i] = ((double)3 / h_i)*(a[i + 1] - 2 * a[i] + a[i - 1]);

vector<double> l(n + 1, 0), u(n + 1, 0), z(n + 1, 0);
l[0] = 1;

for(int i = 1; i < n; i++)
{
    l[i] = h_i * ((double)4 - u[i - 1]);
    u[i] = h_i / l[i];
    z[i] = (alpha[i] - h_i*z[i - 1]) / l[i];//antes era a[i]
}

vector<double> c(n + 1, 0), b(n, 0), d(n, 0);

l[n] = 1;
z[n] = 0;
c[n] = 0;

for(int j = n - 1; j >= 0; j--)
{
    c[j] = z[j] - u[j]*c[j + 1];
    b[j] = (a[j + 1] - a[j]) / h_i - h_i * (c[j + 1] + (double)2 * c[j]) / (double)3;
    d[j] = (c[j + 1] - c[j]) / ((double)3 * h_i);
}

a.pop_back();

Spline s(a, b, c, d);

return s;
}
```