



**DEPARTAMENTO  
DE COMPUTACION**

Facultad de Ciencias Exactas y Naturales - UBA

## Trabajo Práctico 3

Marche un telebeam Don Niembraaaaaa...”

---

Métodos Numéricos  
Primer Cuatrimestre de 2015

Integrante	LU	Correo electrónico
Gastón Zanitti	058/10	gzanitti@gmail.com
Ricardo Colombo	156/08	ricardogcolombo@gmail.com
Dan Zajdband	144/10	Dan.zajdband@gmail.com
Franco Negri	893/13	franconegri200@gmail.com
Alejandro Albertini	924/12	ale.dc@hotmail.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>

# Índice

<b>1. Introducción</b>	<b>3</b>
<b>2. Desarrollo</b>	<b>4</b>
2.1. Vecinos mas cercanos . . . . .	4
2.2. Interpolación Bilineal . . . . .	4
2.3. Interpolación por Splines . . . . .	7
<b>3. Análisis</b>	<b>10</b>
3.1. Metodología de testeo . . . . .	10
3.1.1. Artifacts . . . . .	10
3.2. Correctitud de la implementación . . . . .	11
3.3. Ventana óptima para método de Splines . . . . .	13
3.4. Análisis de los métodos . . . . .	14
3.4.1. Análisis de los métodos para imágenes con símbolos alfanuméricos . . . . .	14
3.4.2. Análisis de los métodos para paisajes . . . . .	17
3.5. Análisis de los métodos para rostros . . . . .	20
3.5.1. Conclusiones del análisis de los métodos . . . . .	22
3.6. Análisis de tiempos . . . . .	23
<b>4. Conclusiones</b>	<b>25</b>
<b>5. Bibliografía</b>	<b>26</b>
5.1. Bibliografía . . . . .	26
<b>6. Apéndice</b>	<b>27</b>
6.1. Compilación y formato de ejecución del programa . . . . .	27
6.1.1. Compilación . . . . .	27
6.1.2. Formato de ejecución . . . . .	27

# 1. Introducción

En el presente trabajo práctico nos encargaremos de analizar el problema de la interpolación de polinomios mediante distintos métodos. Para ello se nos ofrece como marco la necesidad de realizar zoom a distintas imágenes, con el fin de crear un prototipo que permita decidir en tiempo real si una pelota entró o no dentro de un arco de fútbol.

Nuestro objetivo es entonces, dada una imagen de  $n \times m$  píxeles de tamaño original y un número natural  $k$  que denota la cantidad de filas y columnas que se quieren agregar entre cada píxel, encontrar la forma más óptima de rellenar estos valores.

Presentaremos entonces tres técnicas a detallar con sus respectivas ventajas y desventajas:

1. Vecinos
2. Interpolación Bilineal
3. Interpolación por splines

Además, con el fin de poder realizar un análisis cuantitativo sobre cada método, consideraremos dos medidas que comparan las imágenes originales contra sus transformadas, ofreciendo una noción de error o ruido:

1. Error cuadrático medio (ECM)
2. Peak to signal noise ratio (PSNR)

El error cuadrático medio nos permitirá medir de alguna manera cuan alejada esta la imagen transformada de la original. Así, un mayor error cuadrático medio indicará que la imagen transformada se corresponde menos con la imagen original.

Por otra parte el PSNR es una métrica muy utilizada para comparar la calidad de una imagen reconstruida contra la original. Un alto valor de PSNR indica una mejor calidad de la imagen, mientras que un valor bajo indica una peor calidad.

Por último, nuestra visión preliminar del problema nos indica que si bien es posible que los tres algoritmos retornen buenos resultados para valores de  $k$  relativamente bajos ( $k = 1$  y quizás hasta 2) y la versión de vecinos sea mucho más eficiente temporalmente, es probable que esta deje de ser útil muy rápidamente a medida que crece  $k$ . Por el contrario, tanto la interpolación Bilineal como la interpolación por splines deberían funcionar mejor para valores de  $k$  grandes, en el caso de esta última, siendo la que menos ruido introduzca (por la necesidad de que el polinomio sea derivable en cada intersección, suavizando el cambio de uno a otro).

## 2. Desarrollo

A continuación presentaremos el desarrollo de los experimentos que exploran las técnicas mencionadas en la introducción con su respectivo análisis:

### 2.1. Vecinos mas cercanos

El primer método que analizaremos será el de vecinos. Este consiste en rellenar los valores de cada una de las columnas nuevas de la imagen replicando su valor más próximo. La principal ventaja de este método es la simpleza de su implementación, que consta únicamente de dos bucles para iterar la matriz original. Dicha característica también le provee de una eficiencia del orden de  $O(nxm)$  siendo  $n$  el alto y  $m$  ancho de la imagen destino.

---

**TP3 1** void vecinos(Matriz \*image, Matriz \*imageRes , int k)

---

```
1: for 0 to imageRes→rows - 1 do
2:   for 0 to imageRes→cols - 1 do
3:     imageRes→at(i, j) = image→at(round(i/(k+1)), round(j/(k+1)))
4:   end for
5: end for
```

---

Como se mencionó anteriormente, a pesar su alta eficiencia temporal en comparación a los demás métodos, nuestra intuición nos dice que este va ser el que presente una mayor cantidad de ruido, debido a que simplemente se están replicando los píxeles de las imágenes. Como resultado, teniendo en cuenta que lo único que se logra es .añadir grosor.<sup>a</sup> los píxeles, deberían conseguirse imágenes de mayor tamaño pero con una ganancia igual de rápida en los valores de ruido a medida que los valores de  $k$  aumentan.

### 2.2. Interpolación Bilineal

El segundo método que analizaremos será el de interpolación Bilineal. En este caso, la idea consiste en generar un polinomio entre dos puntos consecutivos de la imagen para, por medio de este, calcular los valores necesarios para la extensión.

Primero realizaremos el cálculo por filas y una vez calculados estos valores, repetiremos el mismo procedimiento por columnas. Sean entonces  $Q_{11}$ ,  $Q_{12}$ ,  $Q_{21}$ ,  $Q_{22}$  los cuatro puntos de la imagen original sobre los que queremos interpolar, el objetivo es conseguir un polinomio  $P$  que valga lo mismo en cada uno de estos puntos y aproxime los nuevos valores intermedios. Usaremos entonces para esto el polinomio interpolador de Lagrange.

Interpolando entonces en el eje X obtenemos la siguiente fórmula:

$$f(x, y_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21})$$
$$f(x, y_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22})$$

Ahora, realizando el mismo procedimiento pero en el eje Y, obtenemos lo siguiente:

$$f(x, y) \approx \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - y_1}{y_2 - y_1} f(x, y_2)$$

Si notamos, los puntos que acompañan a las bases polinómicas de Lagrange son los mismos que calculamos sobre el eje X, por lo que podemos realizar el remplazo para llegar a una fórmula cerrada:

$$f(x, y) \approx \frac{y_2 - y}{y_2 - y_1} \left( \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \right) + \frac{y - y_1}{y_2 - y_1} \left( \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \right)$$

Distribuyendo los valores dentro de los paréntesis, obtenemos la ecuación final

$$f(x, y) = \frac{1}{(x_2 - x_1)(y_2 - y_1)} (f(Q_{11})(x_2 - x)(y_2 - y) + f(Q_{21})(x - x_1)(y_2 - y) + f(Q_{12})(x_2 - x)(y - y_1) + f(Q_{22})(x - x_1)(y - y_1))$$

Notar que tanto los valores del X e Y de los puntos con los que generamos la formula, como el valor en el punto son constantes que no varían mientras mantengamos los 4 píxeles escogidos con lo cual podríamos reutilizar esto para los demás cálculos y además la distancia de los puntos  $x_1$  y  $x_2$  es 1 como el de los valores de  $y_1$  e  $y_2$  por lo que el denominador en la formula se puede quitar, quedándonos una formula de una recta.

$$f(x) = Q_{11}.valor * (Q_{22}.x - x) * (Q_{22}.y - y) + Q_{21}.val * (x - Q_{11}.x) * (Q_{22}.y - y) + Q_{12}.valor * (Q_{22}.x - x) * (y - Q_{11}.y) + Q_{22}.val * (x - Q_{11}.x) * (y - Q_{11}.y)$$

Por tanto si se hace primero sobre el eje X la ecuación de la recta y luego sobre el Y, como a la inversa. Ahora, gracias a esta formula, podemos conseguir los valores de las posiciones  $(x, y)$  que agregamos a nuestra imagen para realizar el zoom.

Para facilitar la lectura y escritura del ejercicio vamos a definir una estructura que se llama punto. Dentro de la misma vamos a tener 3 valores, el primero es el valor en x, el segundo en y y el tercero un valor que sera de la imagen original. En el siguiente código las variables  $q11$ ,  $q12$ ,  $q21$ ,  $q22$  son del tipo descripto anteriormente y se utilizan para definir los 4 puntos en los cuales se va a realizar la formula de la recta evaluada en el punto, la matriz A representa la imagen original y la matriz Res representa la imagen extendida. Veamos el siguiente ejemplo para aclarar quienes son los píxeles que utilizamos, supongamos los primeros 4 píxeles de la siguiente forma de una imagen.

Luego utilizando  $k = 2$  agrandamos la imagen dejando en el medio 2 píxeles entre cada píxel de la imagen original.

Cuadro 1: píxeles de imagen original

q11	q12
q21	q22

Cuadro 2: Píxeles de imagen aumentada

q11	a	b	q12
c	d	e	f
g	h	i	j
q21	k	l	q22

---

**TP3 2** void bilinear(matriz A, vector Res,int k)

---

```
Para i= 0...CantFilas - 1
  Para j= 0...CantColumnas - 1
    q11 = < 0,0, Ai,j >
    q12 = < 0,k + 1, Ai,j+1 >
    q21 = < k + 1,0, Ai+1,j >
    q22 = < k + 1,k + 1, Ai+1,j+1 >
  Para x=0...k + 1
    Para y=0...k + 1
      valorRes = polinomioInterpolador(q11,q12,q21,q22,x,y)
      Resi*(k+1)+x,j*(k+1)+y = valorRes
```

---

Como los valores de q11,q12,q21 y q22 son los valores de la imagen original y podemos realizar la formula con los mismos.

Donde *polinomiointerpolador* es la función que se encarga de generar el polinomio interpolador(que en este caso es una recta) en el punto, de la siguiente manera: Para el caso del polinomio interpolador

---

**TP3 3** void polinomioInterpolador(punto q11,punto q12, punto q21, punto q22, int x, int y)

---

```
denominador = 1/ ((q22.x-q11.x)* (q22.y-q11.y))
numerador1= q11.valor* (q22.x-res.x)*(q22.y-res.y) + q21.val * (res.x-q11.x)*(q22.y-res.y)
numerador2= q12.valor* (q22.x-res.x)*(res.y-q11.y) + q22.val * (res.x-q11.x)*(res.y-q11.y)
retorno ((numerador1+numerador2)*denominador)
```

---

se agregaron 2 lineas al final de la rutina, las cuales aplican saturación en caso de ser necesario cuando luego de realizar todas las cuentas el valor que nos queda es mayor a 255, fijándolo en 255, y cuando el valor es menor a 0, fijándolo en 0.

En este método, comparado con el anterior que solo replicaba píxeles vecinos, se está calculando un polinomio para tratar de introducir cierto nivel de suavidad entre los puntos de la imagen original a medida que se recorren los píxeles. Un dato importante a tener en cuenta es que dado que el grado del polinomio aumenta a medida que la cantidad de puntos a interpolar es mayor, decidimos que esta se realice entre solo dos puntos de la imagen original, para ofrecer un mejor desempeño entre puntos, haciendo que el polinomio calculado sea mas operativo y evitando que este oscile demasiado (situación conocida como Fenómeno de Runge).

La complejidad de este algoritmo no es alta. Se recorren n filas y para cada una de ellas m columnas para recorrer toda la matriz, luego se fijan 4 puntos a procesar (que eso tiene complejidad O(1) para acceder a la posición de la matriz) y para cada conjunto de puntos, se realizan 2 ciclos de complejidad O(k), cada uno para recorrer los píxeles cercanos. Luego se genera el polinomio interpolador, el denominador se consigue a través de 2 restas y 1 multiplicación, y el numerador tiene un costo de 12 multiplicaciones y 8 restas. Por ultimo, se multiplican el numerador con el denominador, por lo tanto armar el polinomio interpolador cuesta 10 restas y 14 multiplicaciones. Si consideramos que las restas y multiplicaciones no son muy costosas y que crear el polinomio cuesta O(1) (a muy grandes rasgos), la complejidad del algoritmo Bilineal seria O(n\*m\*k<sup>2</sup>) con n cantidad de filas, m cantidad de columnas y k cantidad de filas/columnas agregadas entre 2 filas/columnas, que por lo general en la práctica, el k es muchísimo más chico que n y m, el tamaño de la imagen.

### 2.3. Interpolación por Splines

Por último nos centraremos en la interpolación por splines. Este método, similar al anterior, requiere el cálculo de Splines (al igual que antes, por filas y luego por columnas) para obtener los valores de los casilleros a extender.

Decidimos para este caso utilizar splines naturales. Recordemos que la condición de suavidad de los splines naturales es que  $S''(a) = 0$  y  $S''(b) = 0$ . Esto determina que la interpolación en los bordes va a ser suave.

Además, en este caso, dado que la interpolación por medio de splines trata de generar polinomios para un segmento específico de la imagen, realizaremos un análisis sobre el algoritmo de interpolación por splines para tratar de obtener el tamaño de ventana más óptimo. Dado que este algoritmo solo incluye los valores de un recuadro de tamaño específico, creemos que agregar más valores puntos a considerar por el spline, no presentará beneficio alguno porque se estarían dejando de lado los valores de la imagen externos al punto a calcular (que estarían influyendo sobre un polinomio que intenta interpolar valores posiblemente lejos de los suyos).

Sea  $S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$  nuestro polinomio interpolador para cada  $j$  intervalo entre 0 y  $n - 1$ , necesitamos entonces resolver los coeficientes. Utilizamos la construcción de Splines despejando estos últimos en función de  $c_j$  para formar el siguiente sistema de ecuaciones  $Ax = b$ :

$$A = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ h_0 & 2(h_0 + h_1) & h_1 & \dots & 0 \\ 0 & h_1 & 2(h_1 + h_2) & \dots & h_{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

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

donde  $h = x_{j+1} - x_j$ . Para nuestra aplicación  $h = k$  y se mantiene fijo, ya que la distancia entre los puntos de la nueva imagen es igual a  $k$ . Una vez resuelto este sistema y con los valores de  $c_0, \dots, c_n$  ya calculados, podemos despejar los coeficientes que necesitábamos en base a las siguientes ecuaciones (que se derivan de las condiciones del mismo spline):

$a_j$  es el valor del píxel en la posición  $j$  de la imagen original en la fila o columna iterada por el spline

$$b_j = \frac{1}{k}(a_{j+1} - a_j) - \frac{k}{3}(2c_j + c_{j+1})$$

$$d_j = \frac{c_{j+1} - c_j}{3k}$$

La implementación del algoritmo Bicubico consta de dos partes muy similares, ya que primero se recorre la matriz por columnas para realizar el método de splines y luego se realiza el método por filas para completar la matriz. Definimos para él una clase llamada spline donde almacenaremos los arreglos *as*, *bs*, *cs* y *ds*, estos contienen los coeficientes del polinomio para la posición *i* de la fila o columna en donde estemos calculando los splines.

---

**TP3 4** void bicubico(matriz A, vector Res,int k)

---

```

1: Para i= 0..CantFilas - 1
2:   Para j= 0..CantColumnas - 1
3:     Splinespline = calcularSpline(CantColumnas, columnaj, k)
4:   Para j= 0..CantColumnas - 1
5:     Para l= 0..k + 1
6:       valor= spline.a[i] + spline.b[i] * l + spline.c[i] * j2 + spline.d[i] * j3
7:       saturar(valor)
8:       Resi*(k+1),j*(k+1)+l = valor
9: Para i= 0..CantColumnas - 1
10:  Para j= 0..CantFilas - 1
11:    spline= calcularSpline(CantFilas, fila(j), k)
12:  Para j= 0..CantFilas - 1
13:    Para l= 0..k + 1
14:      valor= spline.a[i] + spline.b[i] * l + spline.c[i] * j2 + spline.d[i] * j3
15:      saturar(valor)
16:      Resj*(k+1)+l,i = valor

```

---

La función *saturar* hace que si el valor es mayor a 255 o menor a 0 los fija en esos dos valores respectivamente.

Como mencionamos anteriormente se puede ver que de las lineas 1-8 se realiza el splines por columnas, luego en las lineas 9-16 se realiza el splines por filas, por lo tanto solo analizaremos el primer bloque (lineas 1- 8) para el siguiente es análogo.

En la linea 3 se llama al algoritmo de splines pasándole los *n* puntos de la imagen con el cual vamos a calcular coeficientes para cada polinomio, una vez obtenido esto se recorren los *k* puntos entre cada par de píxeles de la imagen resultante y se evalúa el polinomio en ese punto, logrando así el valor de cada punto en la imagen resultante.



---

**TP3 5** spline calcularSpline(int cant,arreglo(int) pixelesOriginales,int k)

---

```
1: arreglo alfa[cantColumnas]
2: Para j= 0..Cant - 1
3:    $alfa_j = (3/k) * (pixelesOriginales_{j+1} - pixelesOriginales_j) - (3/k) * (pixelesOriginales_j -$ 
    $pixelesOriginales_{j-1})$ 
4: arreglo(float) ln ,cn , zn
5: l[0]=1,c[0]=0,z[0]=0
6: Para i= 0..Cant - 1
7:    $l_i = 2 * (2 * k) - k * c_{i-1}$ 
8:    $c_i = k/l_i$ 
9:    $z_i = alfa_i - (k * z_{i-1})/l_i$ 
10: arreglo(int) as,bs,cs,ds
11: Para i= 0..Cant - 1
12:    $cs_i = z_i - c_i * cs_{i+1}$ 
13:    $bs_i = (as_{i+1} - asi)/k - k * (cs_{i+1} + 2 * cs_i)/3$ 
14:    $(cs_i + 1] - cs[i])/(3 * k)$ 
15: devolver spline(as,bs,cs,ds)
```

---

En este algoritmo [1] estamos calculando los coeficientes del polinomio dado un arreglo de elementos que van a ser nuestros puntos.

Este método, que podría considerarse un refinamiento del anterior, introduce la particularidad de que se le pide al polinomio interpolador que las intersecciones de las funciones que interpolan al punto  $n - 1$  y  $n$  y al  $n$  y al  $n + 1$ , sean derivables. Como resultado, se agrega mucha más suavidad entre puntos que con la técnica anterior que solo respetaba que las funciones empiecen y terminen en el mismo punto (dando lugar a posibles picos, como en el caso de que dos puntos se interpolen con una recta ascendente y los siguientes con una descendente). Además, esta técnica evita de forma natural la oscilación del polinomio mencionada en el método anterior dado que siempre se toman polinomios por partes y, como mencionamos, al usar splines naturales podemos garantizar que la interpolación es suave en sus bordes, además de serlo en los bordes generados por los límites de cada spline. Gracias a esto, cuando se intenta reducir el error de interpolación se puede incrementar el número de partes del polinomio que se usa para construir el spline, en lugar de incrementar su grado.

### 3. Análisis

#### 3.1. Metodología de testeo

A continuación se presenta un análisis comparativo de los tres métodos implementados. Cabe mencionar que, dado que la experimentación requería que ambas imágenes (original y modificada) tengan el mismo tamaño para poder realizar un análisis cuantitativo de los algoritmos mediante las medidas de comparación que se mencionaron en la introducción, decidimos achicar la imagen original mediante un script que creamos para luego agrandarla mediante nuestros métodos y poder comparar los resultados obtenidos con la imagen original.

El recortador de imágenes, que se encuentra en la carpeta **Recortador de imágenes**, toma una imagen como input la cual se quiere achicar para poder comparar con si misma en tamaño original una vez aplicado el algoritmo de zoom y un entero  $k$  que es el factor de achicamiento, que debe ser el mismo que usemos a la hora de aplicar zoom sobre esta imagen achicada. Este script lo hicimos utilizando *OpenCV*, de modo similar a las funciones utilizadas para hacer zoom en la experimentación base.

El comportamiento de este programa es sencillo, partiendo de una imagen original toma un pixel cada  $k$  pixeles hasta llegar al final de la imagen. De esta manera, al aplicar los algoritmos de zoom, estas  $k$  posiciones serán completados nuevamente y podremos comparar como completa cada método la imagen. Puede que algunas columnas del final queden truncadas ya que no siempre es posible partir una imagen perfectamente de esta manera, pero consideramos que esas pocas columnas son despreciables para una imagen lo suficientemente grande.

Para asegurarnos que todos los cálculos sean lo mas precisos posibles utilizamos la extensión *.png* para guardar todas las imágenes con las que trabajamos, ya que este formato utiliza un algoritmo de compresión sin pérdida que nos asegura que ninguna información se pierde al momento de guardar las imágenes.

##### 3.1.1. Artifacts

Además de las métricas ya mencionadas brevemente en la introducción realizaremos un análisis visual de las imágenes y de los errores que se puedan percibir.

Para ello prestaremos atención a los artifacts que se puedan observar en las imágenes reconstruidas. Un artifact digital es un error no deseado o no intencionado en los datos debido a la manipulación de la información. Existen muchas causas por las cuales se pueden generar artifacts, desde algún desperfecto en el hardware, debido a la compresión de los datos, etc.

Además existen variados tipos de artifacts, a continuación describiremos algunos:

- **Ruido:**  
El ruido en imágenes digitales es mas claramente visible sobre superficies uniformes donde se pueden observar cierto tipo de granularidades.
- **Aliasing:**  
El aliasing de una imagen puede describirse de manera informal como los 'serruchos' que pueden observarse cerca de los bordes de los objetos.
- **Sharpening:**  
El Sharpening ocurre cuando al agrandar una imagen se sobre dimensionan los bordes de tal manera que la imagen parezca tener mas contraste.

A continuación se muestra un ejemplo de cada uno de estas aberraciones:

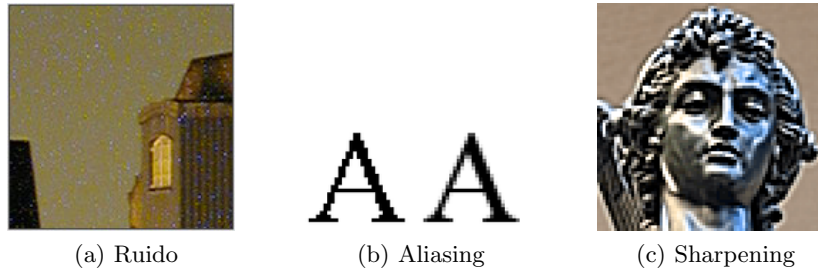


Figura 1: Artifacts

### 3.2. Correctitud de la implementación

El primer método para comprobar rápidamente la correctitud de la implementación de nuestros algoritmos fue sencillamente comparar "a ojo" las imágenes resultantes con la original, observando el nivel de detalle obtenido. Al comienzo, teniendo errores de implementación, notamos de este modo que la implementación necesitaba mejoras.

Luego, para obtener mayor rigor, procedimos a comparar nuestros algoritmos con aquellos que vienen por defecto en opencv. Consideramos que estos algoritmos son lo suficientemente fiables como para tomarlos como punto de referencia. Tomamos una imagen:



(a) .

Figura 2: Imagen Original

Y la reescalamos en la mismas dimensiones tanto con nuestros algoritmos como con los implementados en Opencv.

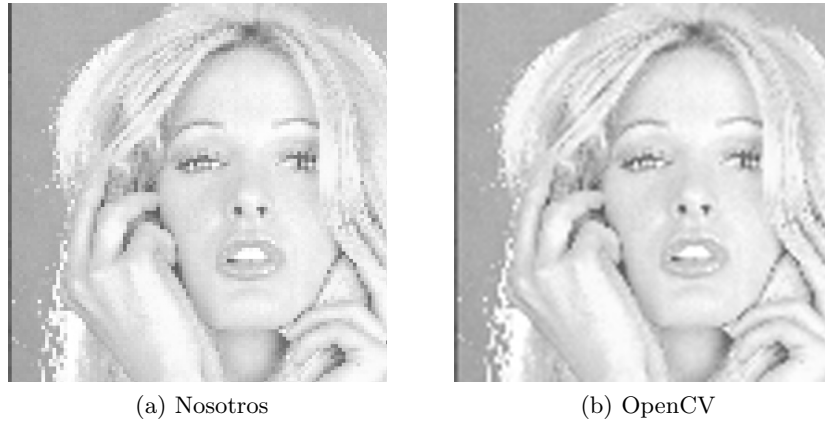


Figura 3: Comparación de correctitud contra Opencv: Vecinos Mas Cercanos

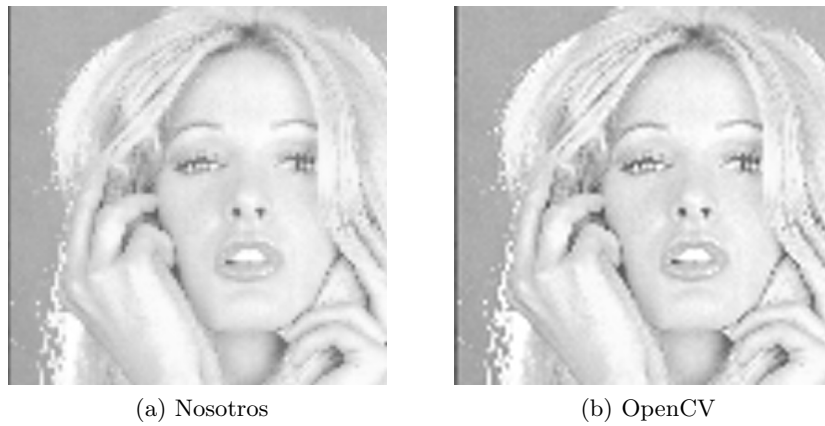


Figura 4: Comparación de correctitud contra Opencv: Bilineal

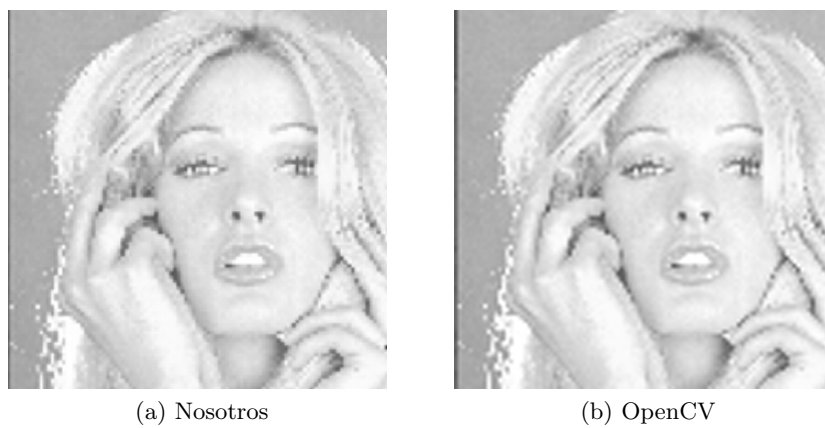


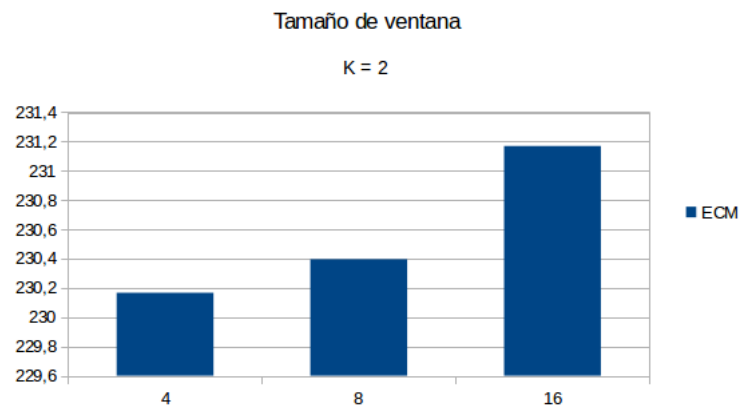
Figura 5: Comparación de correctitud contra Opencv: Bicubico con ventanas de  $4 \times 4$  píxeles

Como puede verse, nuestros algoritmos arrojan resultados muy similares a Opencv. Por lo que consideramos que su implementación es correcta.

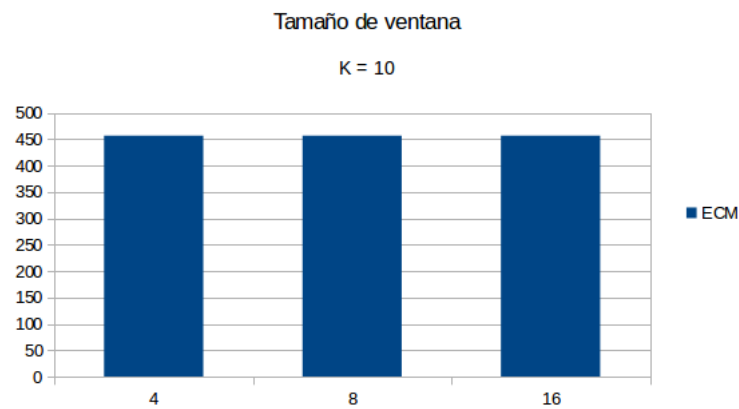
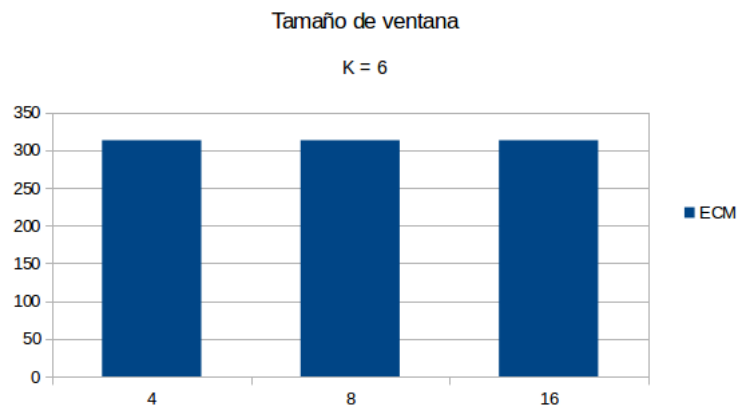
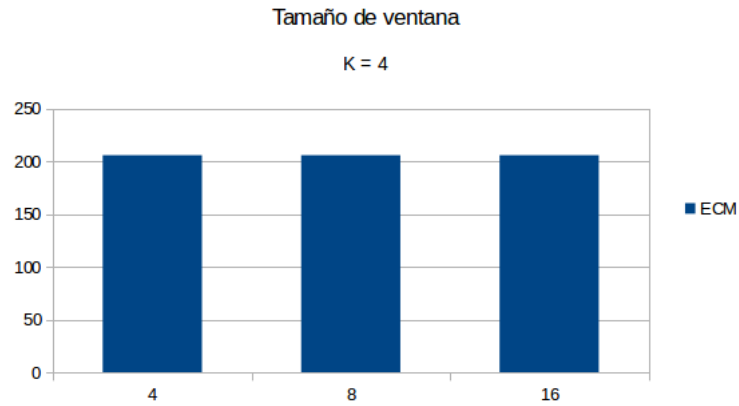
### 3.3. Ventana óptima para método de Splines

Nuestro primer análisis se encargará de encontrar un valor óptimo para la cantidad de las ventanas utilizadas en el método de splines. Para dicho fin, elegimos correr varias instancias del método con valores de  $K$  crecientes para distintos valores de ventana (4, 8, y 16 para poder comparar los resultados). Se presentan entonces los resultados obtenidos. Vale la pena destacar que no se muestra información respecto al PSNR debido a que presentaba exactamente el mismo comportamiento y no ofrecía información extra alguna.

Como habíamos presupuesto en la introducción, agrandar el tamaño de la ventana solo hace que se tengan en cuenta valores para el punto que se quiere calcular que no depende directamente de este.



Como puede verse a continuación, agrandar el tamaño de la ventana deja de presentar beneficio alguno para valores mas altos de  $K$  porque los resultados se vuelven constantes:



A partir de este punto, el algoritmo de splines utiliza una ventana de tamaño cuatro, dado que es la que presenta mejores resultados. De todos formas, no es cierto que siempre sea preferible una ventana más chica a una que incluya mas puntos, porque en problemas donde se quieren calcular por ejemplo trayectorias, es deseable considerar mas puntos para tener mas información.

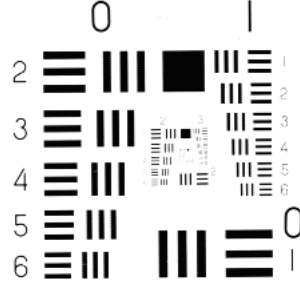
### 3.4. Análisis de los métodos

#### 3.4.1. Análisis de los métodos para imágenes con símbolos alfanuméricos

En esta sección analizaremos los tres algoritmos sobre imágenes con símbolos alfanuméricos. Para ellos usamos la imagen mostrada mas abajo para la cual aplicaremos los tres algoritmos implementados con diferentes ks.

Para ello, tomamos la imagen original de  $256 \times 256$  y reducimos su tamaño de la manera ya mencionada al principio de esta sección. De esta manera al aplicarle el  $k$  se obtiene un tamaño similar al original. Para este experimento, las imágenes reconstruidas a través de este procedimiento tendrán un tamaño entre  $255 \times 255$  y  $226 \times 256$ .

La característica principal que queremos testear será la capacidad de discernimiento de estos símbolos después de aplicados los métodos de zoom.



Primero realizamos las pruebas con el valor mínimo de  $k$  ( $k = 1$ ), obteniendo los siguientes resultados:

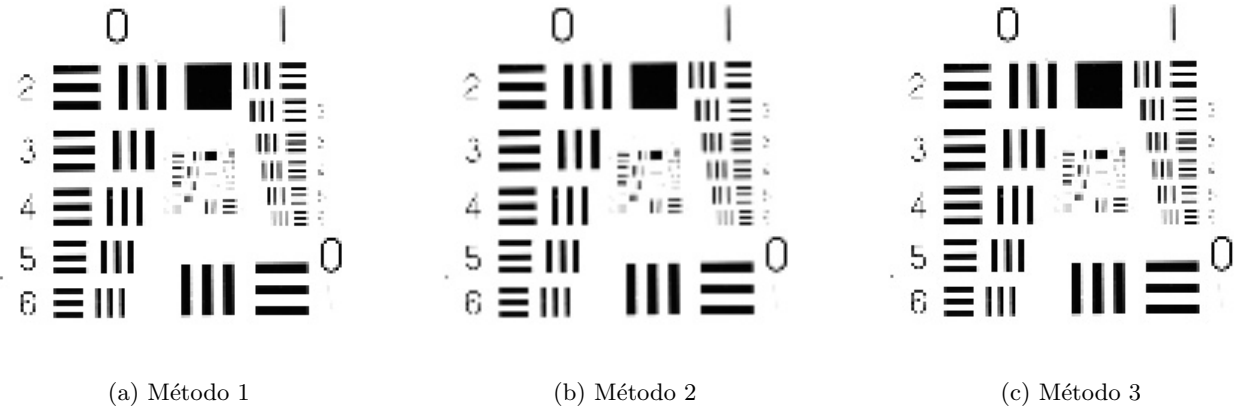


Figura 6: Comparación de métodos para  $k = 1$

Como podemos ver, las tres imágenes introducen artifacts que todavía no desmejoran la imagen a un nivel en el que sea imposible su comprensión, por lo menos en los dígitos mas externos (distinto para los números internos de la imagen que, debido a su tamaño inicial, ya son casi imperceptibles con este  $k$  mínimo). Notese como el método 2 (el algoritmo bilineal), como consecuencia de la nivelación entre los valores de alto contraste del dibujo y su fondo blanco, empieza a introducir una leve cantidad de ruido alrededor de las zonas negras. La misma situación se plantea en el método tres (el algoritmo bicubico), pero con la diferencia de que el difuminado introducido es mucho menos visible.

Observemos los resultados para un  $k$  un poco mas elevado ( $k = 2$ ):

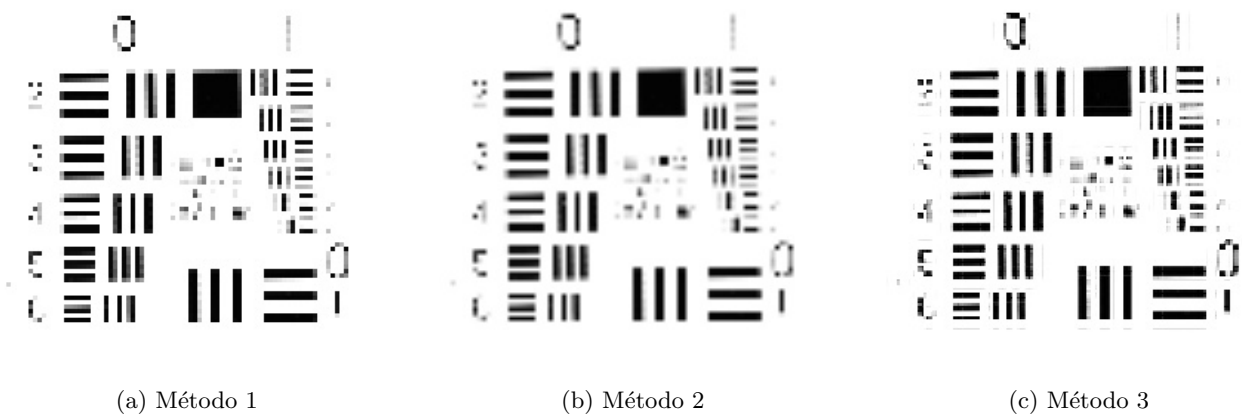


Figura 7: Comparación de métodos para  $k = 2$

En esta ocasión, el comportamiento sigue los lineamientos generales del caso anterior, con la salvedad de que ninguna de las tres imágenes ya es comprensible. Vemos como el ruido comentado en el caso anterior avanza rápido en el algoritmo bilineal, para casi difuminar la imagen por completo. En el caso del algoritmo que utiliza splines, se puede empezar a ver un pequeño sombreado alrededor de los bordes de los elementos en la imagen, pero a diferencia del método dos, esta solo se extiende a las cercanías y no avanza por toda la imagen.

Por ultimo, presentamos los resultados para  $k = 4$ :



Figura 8: Comparación de métodos para  $k = 4$

Como era de esperarse las tres imágenes resultantes ya perdieron comprensión en su totalidad. Además, el segundo y tercer método, presentan una alta cantidad de ruido por el difuminado producido respecto de la imagen original. Queda entonces a la vista una característica que no estábamos considerando hasta entonces en nuestro análisis. El método de los vecinos puede llegar a ofrecer resultados favorables si se cumplen algunas características deseables (nuestra intuición preveía que esta implementación seria superado por los anteriores en cualquier situación) como en este caso. El alto contraste entre las imágenes, hace que en los métodos que introducen cierta correlación o suavizado entre píxeles generen un sombreado que hace mas borrosas las imágenes y que las vuelve menos claras. En contra de nuestros pronósticos, el método de los vecinos podría ser un excelente candidato en estos casos.

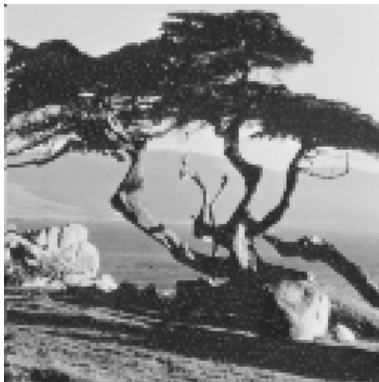


### 3.4.2. Análisis de los métodos para paisajes

En esta sección analizamos como se comportan los algoritmos desarrollados para fotos de paisajes. Consideramos una imagen que no presente grandes contrastes como en el análisis anterior y en la que se puedan analizar tanto detalles puntuales (la definición de las ramas del árbol) así como aquellos mucho mas definidos (piedras y ramas que cubren un gran porcentaje de la imagen). Tomamos la siguiente imagen de  $256 \times 256$ :



Y reconstruimos la imagen con la metodología ya mencionada en apartados anteriores para  $k = 1$ . Los resultados son los siguientes:



(a) Método 1



(b) Método 2



(c) Método 3

Figura 9: Comparación de métodos para  $k = 1$

En este primer ejemplo, con un  $k$  mínimo, ninguna de las tres imágenes presenta una calidad demasiado desmejorada.

El *ECM* y *PSNR* obtenido para cada uno de los métodos fue el siguiente:

- Método 1: ECM de 315,723 y PSNR de 23,1377.
- Método 2: ECM de 115,114 y PSNR de 27,5195.
- Método 3: ECM de 332,629 y PSNR de 22,9112

En particular nos sorprendió ver el ECM de la implementación que utiliza interpolación por splines, que visualmente está más cerca al algoritmo de los vecinos más cercanos (el cual se perfilaba como el de peor rendimiento de los tres y terminó en segundo lugar) que al método dos (de el cual, de hecho, suponíamos era una mejora tanto con el error cuadrático medio como con el PSNR).

En el método 1 puede notarse ya cierto aliasing en los bordes mas prominentes de la fotografía, como el tronco de la izquierda. Además si se observa la sombra de los arboles, que en la imagen original resultaba uniforme puede verse ahora como el método 1 y 3 introdujo una gran cantidad de ruido en esos lugares.

Ahora lo hacemos para  $k = 2$ , se obtiene esto:

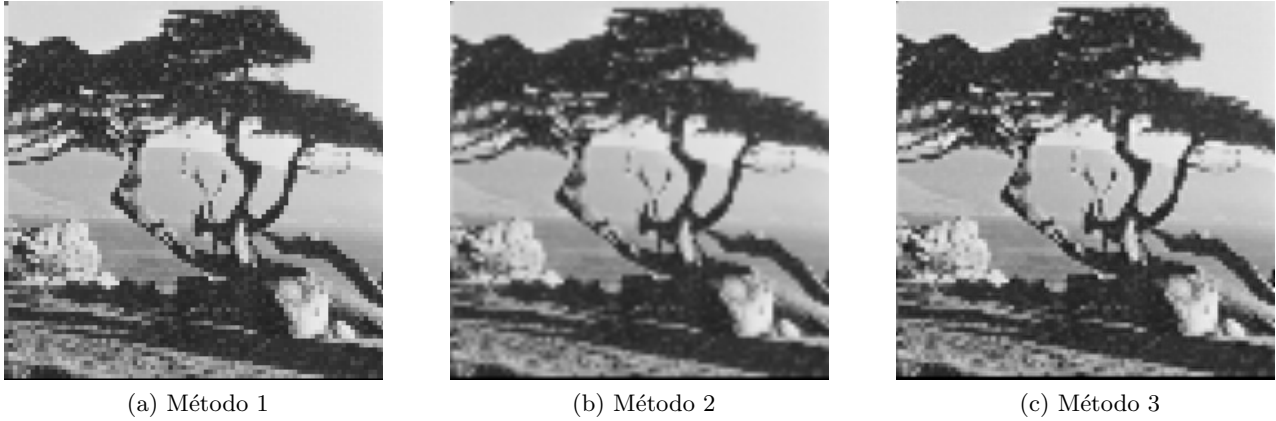


Figura 10: Comparación de métodos para  $k = 2$

En este segundo caso, aun con un pequeño cambio en el  $k$ , podemos apreciar como el método de los vecinos y el de splines (método 1 y 3 respectivamente) empiezan a introducir una gran cantidad de ruido. En el caso del método dos, a diferencia de lo ocurrido durante el análisis de caracteres alfanuméricos, el suavizado que se produce en la imagen si ayuda a que esta se mantenga entendible y el difuminado termina favoreciendo a la comprensión de la misma (esto no ocurría en los caracteres alfanuméricos, porque este mismo suavizado termina oscureciéndola y quitándole claridad). Una vez mas, los valores de error cuadrático medio y la relación señal/ruido apoyan el análisis realizado e incluso demuestran que la diferencia introducida en el ECM se duplica entre cada método.

- Método 1: ECM de 724,517 y PSNR de 19,5303.
- Método 2: ECM de 293,755 y PSNR de 23,4509.
- Método 3: ECM de 444,18 y PSNR de 21,6552

Además la implementación de vecinos mas cercanos 1 presenta un aliasing muy notable en casi toda la imagen.

Ahora lo hacemos para  $k = 3$ , se obtiene esto:

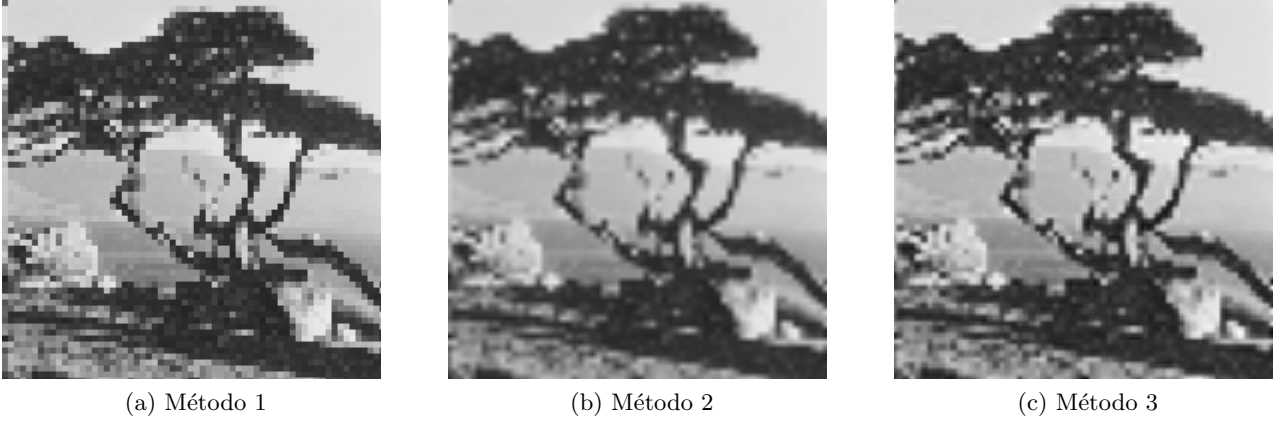


Figura 11: Comparación de métodos para  $k = 3$

En este caso, podemos ver nuevamente como el aumento mínimo del valor de  $k$  produce niveles altísimos de pérdida de definición en las tres imágenes. Contraria a nuestra intuición, el método dos sigue ofreciendo un mejor desempeño en este caso, incluso contra el método de Splines. El método de los vecinos, que no presenta ningún suavizado, queda rápidamente relegada al ultimo lugar en cuanto a la cantidad de ruido (Notese como ya es difícil diferenciar las zonas de mayor definición como las ramas del árbol e incluso empieza a estar comprometida nuestra capacidad de diferenciar donde empieza la piedra que se encuentra en la zona media izquierda y donde lo hace la superficie del piso). En el método dos podemos confirmar que, como en el caso anterior, a cambio de introducir cierto sombreado en la imagen se consigue la mejor definición colocándolo nuevamente como el de mejor desempeño. Una vez mas, los valores de  $ECM$  y  $PSNR$  son, como era de esperarse, los siguientes:

- Método 1:  $ECM$  de 1099,83 y  $PSNR$  de 17,7175.
- Método 2:  $ECM$  de 400,854 y  $PSNR$  de 22,1009.
- Método 3:  $ECM$  de 493,992 y  $PSNR$  de 21,1936

Notese como los métodos dos y tres presentan una pequeña diferencia comparándola con la cantidad de error introducida por el primer método.

Testeamos con un valor extremo de  $k$  para ver cuales son los resultados, para  $k = 10$  se obtiene:

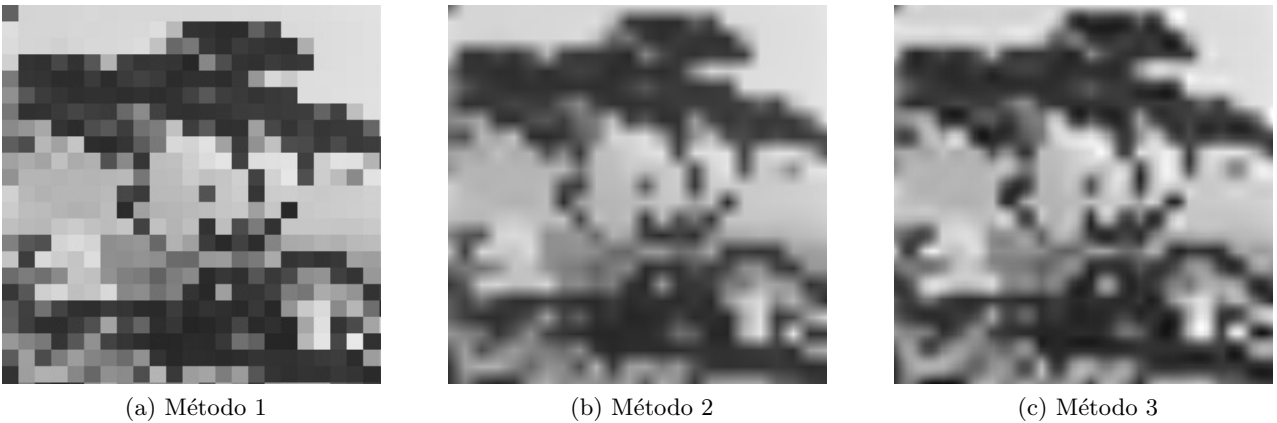


Figura 12: Comparación de métodos para  $k = 10$

Como esperábamos, ninguna de las tres imágenes es ya reconocible. El método uno, vecinos, es claramente el mas perjudicado con una imagen final que recuerda a las viejas imágenes de 8bits debido a su alto grado de aliasing. En el caso del método de splines, la imagen es algo mas clara, sin embargo y contrario a nuestra intuición, a este nivel se observan cierto cuadriculado en la imagen, producto de las ventanas que tomamos para su aproximación. Curiosamente, esperábamos que sea el método Bilineal quien cuente con este desventaja, producto de la falta de suavizado entre los píxeles vecinos.

- Método 1: ECM de 2976,63 y PSNR de 13,3936.
- Método 2: ECM de 1266,65 y PSNR de 17,1042.
- Método 3: ECM de 1454,78 y PSNR de 16,5028

Como puede verse, el método 2 continua siendo el que produce menos ECM y el que obtiene mejor PSNR incluso para estos niveles altísimos de  $k$ .

### 3.5. Análisis de los métodos para rostros

En esta ultima sección analizamos como se comportan los métodos para fotos de rostros. Centraremos nuestro análisis en el comportamiento de los tres algoritmos frente a rasgos particulares del rostro para poder valorar la calidad de los mismos. Tomamos la siguiente fotografía de  $256 \times 256$ :



Presentamos el mismo análisis que en las imágenes anteriores ( $k = 1$ ,  $k = 2$  y  $k = 3$ )



(a) Método 1



(b) Método 2



(c) Método 3

Figura 13: Comparación de métodos para  $k = 1$

ECM y PSNR para  $k = 1$ :

- Método 1: ECM de 95,6087 y PSNR de 28,3258.
- Método 2: ECM de 32,6121 y PSNR de 32,997.
- Método 3: ECM de 105,23 y PSNR de 27,9094



(a) Método 1



(b) Método 2



(c) Método 3

Figura 14: Comparación de métodos para  $k = 2$

ECM y PSNR para  $k = 2$ :

- Método 1: ECM de 235,245 y PSNR de 24,4156.
- Método 2: ECM de 87,1968 y PSNR de 28,7258.
- Método 3: ECM de 150,451 y PSNR de 26,3569



(a) Método 1



(b) Método 2



(c) Método 3

Figura 15: Comparación de métodos para  $k = 3$

ECM y PSNR para  $k = 3$ :

- Método 1: ECM de 389,559 y PSNR de 22,2251.

- Método 2: ECM de 113,193 y PSNR de 27,5926.
- Método 3: ECM de 159,421 y PSNR de 26,1054.

Este análisis vuelve a repetir el mismo comportamiento que vimos en el caso de las imágenes de paisajes en cuanto a la relación entre el error introducido por cada uno de los métodos (aunque con distintos valores, la relación en el fondo es la misma). Sin embargo, y como una apreciación totalmente subjetiva, consideramos que el método tres (aproximación por splines), conserva mucho mejor las características del rostro de la imagen que los otros dos; el primer caso debido a la introducción excesiva de aliasing de la imagen, y el caso tres debido a que, debido al suavizado general de toda la imagen que ya mencionamos en ocasiones anteriores, difumina demasiado los detalles menores del rostro de la imagen.

Testeamos con un valor extremo de  $k$  para ver cuales son los resultados, para  $k = 10$  se obtiene:

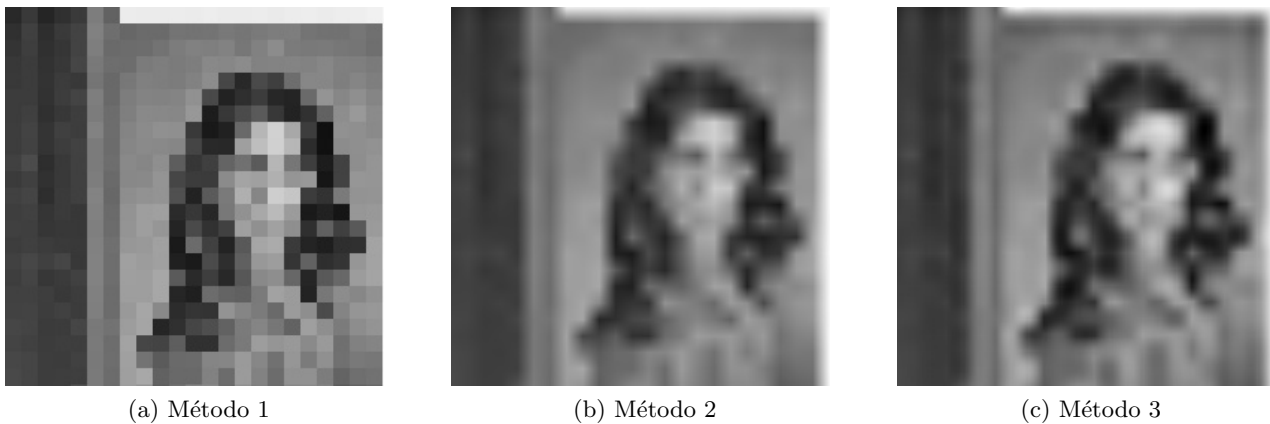


Figura 16: Comparación de métodos para  $k = 3$

ECM y PSNR para  $k = 10$ :

- Método 1: ECM de 1563,05 y PSNR de 16,1911.
- Método 2: ECM de 684,334 y PSNR de 19,7781.
- Método 3: ECM de 639,892 y PSNR de 20,0697

Una vez mas, vuelve a repetirse resultados similares que en el caso anterior para  $k = 10$ . De nuevo el método de los vecinos pierde frente a los demás por una gran diferencia (tanto 'visual' como con respecto al ECM y el PSNR). Las diferencias entre el método Bilineal y el método de splines es ,muy pequeña aunque en este caso resulta levemente mejor el método 3.

### 3.5.1. Conclusiones del análisis de los métodos

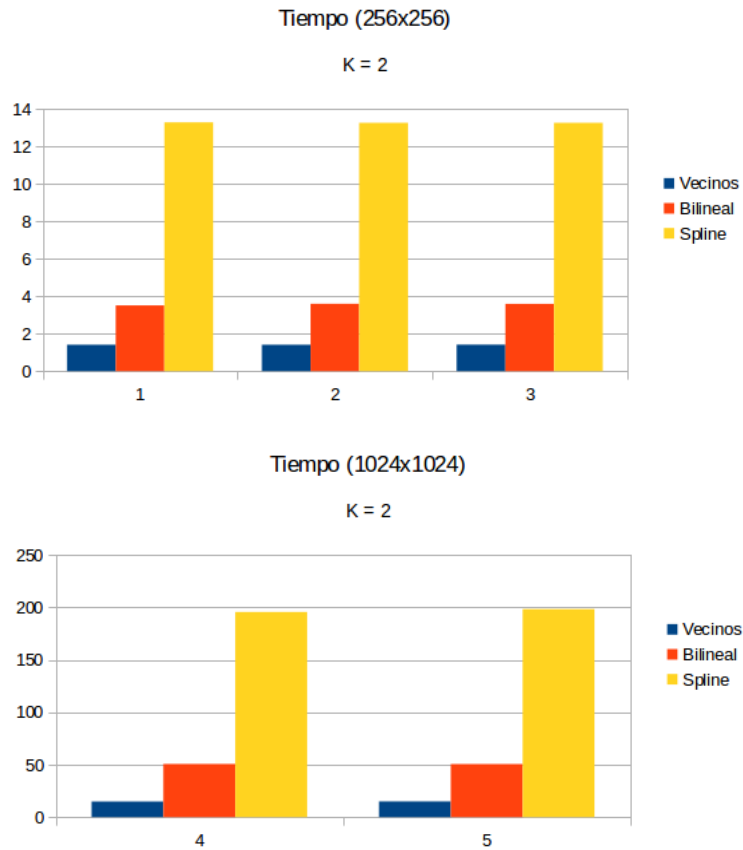
Como pudimos apreciar a lo largo de los distintos análisis realizados, no existe un método que sea claramente un ganador. Si bien es cierto que el método dos es el que introduce la menor cantidad de errores, una apreciación mas 'subjetiva' y alejada de los números nos deja las siguientes sensaciones:

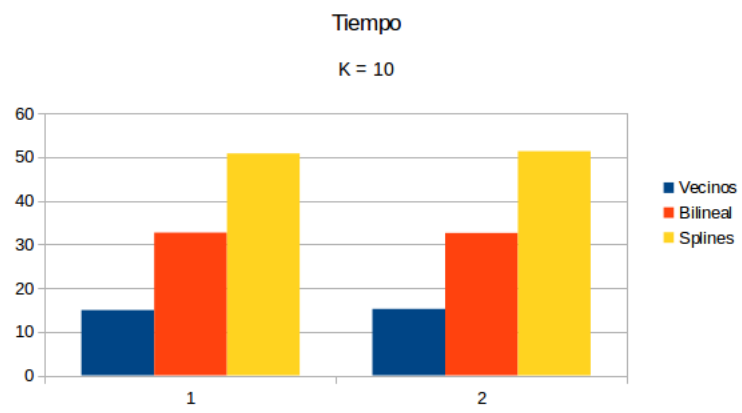
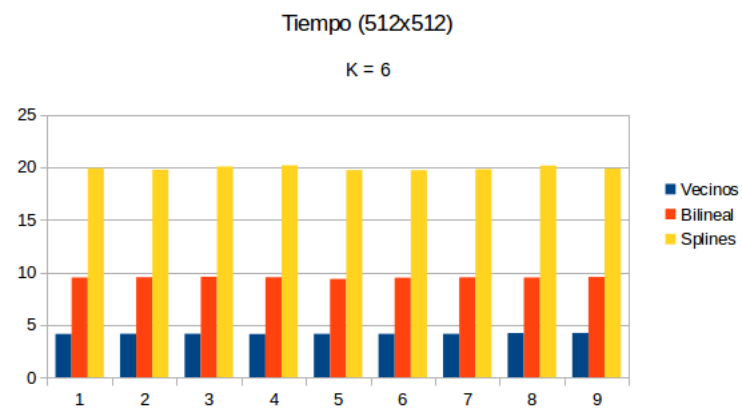
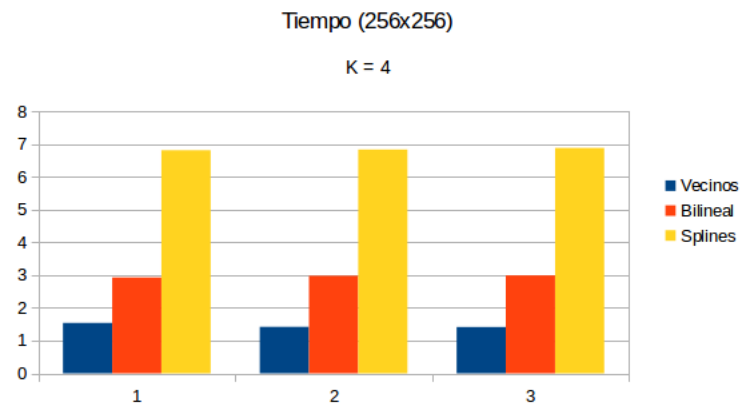
1. El método de los vecinos puede llegar a mostrar mejores resultados en situaciones de imágenes que consisten en elementos frente a un fondo uniforme con un alto contraste debido a que al no realizar ninguna interpolación entre los píxeles evita introducir sombreados innecesarios.

2. El método Bilineal, a pesar de ser el mejor en cuanto a errores y la cantidad de ruido introducida en las imágenes, no siempre es considerado el algoritmo óptimo según nuestro análisis. En situaciones donde la imagen presenta regiones de alta cantidad de detalles (como puede ser el rostro de una persona en el caso analizado), genera un sombreado que hace perder definición en estas zonas. Situación que no ocurre para el método de Splines. Sin embargo, en imágenes donde no hay regiones de grandes detalles (como puede ser un paisaje) si lo consideramos el óptimo.
3. El método de interpolación de Splines, que presenta valores cercanos al mejor candidato (la implementación Bilineal) en la mayoría de los casos, tiene la ventaja de que, al introducir menor cantidad de 'sombreado' que este ultimo, conserva mejor aquellos detalles mencionados en donde el método Bilineal perdía claridad.

### 3.6. Análisis de tiempos

El análisis de tiempo, a diferencia del de los métodos, no ofreció ninguna respuesta que no hayamos podido intuir durante la codificación de los algoritmos. Es claro que a medida que el método se perfecciona en la búsqueda de resultados más suaves, también aumenta el tiempo necesario de cálculo.







## 4. Conclusiones

Como se mencionó con anterioridad, nuestra intuición relacionaba fuertemente a los métodos en cuanto a calidad/desempeño. Como se pudo ver a lo largo del análisis realizado, el método de splines nunca logró sacar una diferencia significativa respecto al método de interpolación Bilineal. También se puede apreciar como este último tiene un mejor desempeño temporal en todos los casos. Esto coloca a la interpolación Bilineal como la mejor opción en cuanto a tiempo y calidad, dado que la ganancia por splines es mínima respecto al tiempo extra. En un momento ( $K = 2$ ), nos llamó poderosamente la atención que el método Bilineal haya obtenido un mejor resultado que el método de splines, pero teniendo en cuenta como se terminaron comportando ambos métodos a lo largo de todo el análisis, ahora ya no parece un resultado tan anómalo. Sin embargo, no logramos llegar a una conclusión que justifique el porque de una diferencia tan significativa.

En cuanto al método de los vecinos, como bien dijimos al principio, se comporta relativamente bien para valores de  $k$  mínimos, pero enseguida que este crece, el método pierde fiabilidad.

## 5. Bibliografía

### 5.1. Bibliografía

#### Referencias

- [1] Richard L. Burden *Numerical Analysis*, 9th edition, 2011.

## 6. Apéndice

### 6.1. Compilación y formato de ejecución del programa

#### 6.1.1. Compilación

Como se encuentra mencionado en el archivo *README.txt*, en la carpeta *src* de la carpeta raíz del trabajo se encuentra un *Makefile*. Así es que ejecutando el comando

*make*

se genera el ejecutable *tp*.

#### 6.1.2. Formato de ejecución

El comando para correr el experimento para una instancia es:

*./tparchivoEntradametodo k*

Donde:

- *archivoEntrada* es la imagen original
- *metodo* es un entero entre 0 y 2 donde
  - 0 es el metodo de vecinos
  - 1 es el metodo Bilineal
  - 2 es el metodo de Splines
- *k* es un entero que indica cuanto zoom se hará en la foto