

# **Tratamiento Digital de Imágenes**

**Informe Práctica**

## **Redimensionamiento de Imágenes**

**Antonio López Hernández  
3º Grado Ingeniería Informática  
Curso 2015-2016**



**UNIVERSIDAD DE ALMERÍA**

# **INDICE**

- . Introducción**
- . Método: Vecino Más Cercano**
- . Método: Interpolación Lineal**
- . Comparaciones**
- . Código**
- . Bibliografía**

# Introducción

Mi práctica se ha basado en el método de tratamiento de imágenes de redimensionamiento. Redimensionar es el proceso de establecer unas nuevas dimensiones para una imagen dada, es decir vamos a modificar el tamaño de una imagen.

A simple vista puede parecer un proceso sencillo, pero no es así. Cuando hablamos de hacer una imagen más grande o más pequeña hablamos en porcentajes, aumentar una imagen será un porcentaje mayor a 100% y reducirla por el contrario será hablar de un porcentaje mayor al 100%. Podemos observar rápidamente que hacer un 200% de una imagen es duplicar su tamaño y un 50% sería reducirla a la mitad.

Cuando hablamos de porcentajes redondos se ve bastante claro, si tenemos una imagen de 100px cuadrados pensábamos rápidamente que aplicar un 50% de porcentaje nos dejará una imagen de 50px, lo importante es pensar: ¿Cuáles son los 50px que voy a elegir para generar la nueva imagen? Aquí nos encontramos los problemas, obviamente no vamos a elegir los 50px primeros dado que pediríamos mucha información en la imagen, en este caso podríamos pensar en decir: uno si uno no, así conseguiríamos 50px y con mucha información, es sencillo, pero que ocurre cuando elegimos un redimensionamiento del 73% ¿Cuáles son los píxeles que vamos a elegir y cuáles no?

Una vez que tenemos esto en mente investigamos acerca de cómo seleccionamos esos píxeles, de ahí encontramos los métodos más famosos para el reescalado de imágenes.

Hay que tener en cuenta que este programa solo funciona con imágenes en escala de grises.

## Método: Vecino Más Cercano

Este método es el más sencillo, pero por otro lado es el que peor resultado proporciona.

El método del vecino más cercano consiste en crear una nueva matriz con las nuevas medidas calculadas dependiendo del porcentaje de redimensionamiento que se nos ha pedido, en ese momento vamos a rellenar esta nueva matriz aplicando una fórmula matemática muy sencilla:

$$c = (c' - 1) \frac{T_{am}-1}{T_{am'}-1} + 1$$

De esta fórmula indicar que

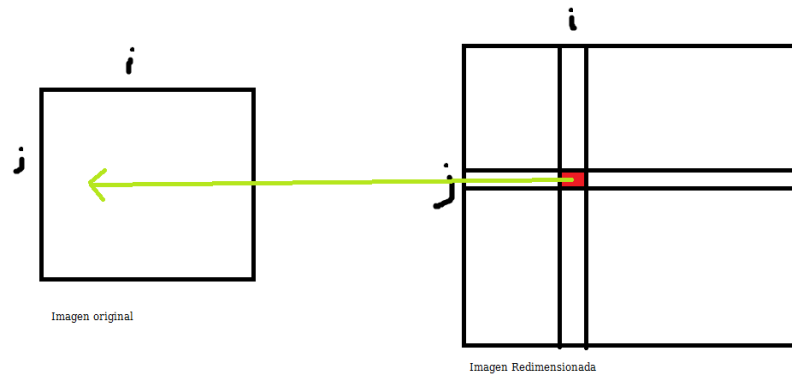
- C es la posición de la matriz original
- C' es la posición de la nueva matriz
- Tam es el tamaño máximo de alto o ancho de la imagen principal
- Tam' es el tamaño máximo de alto o ancho de la imagen creada

El objetivo es rellenar la nueva matriz, debemos de elegir que pixel vamos a poner en la matriz, el método del vecino más cercano junto con esta fórmula consiste en elegir de la matriz original que pixel es el que vamos a poner en la nueva matriz.

Necesitamos darle la posición de la i y de la j, por tanto aplicamos la fórmula dos veces:

```
imagen2(i, j) = imagen(((i - 1)*z1) + 1 , ((j - 1)*z2) + 1);
```

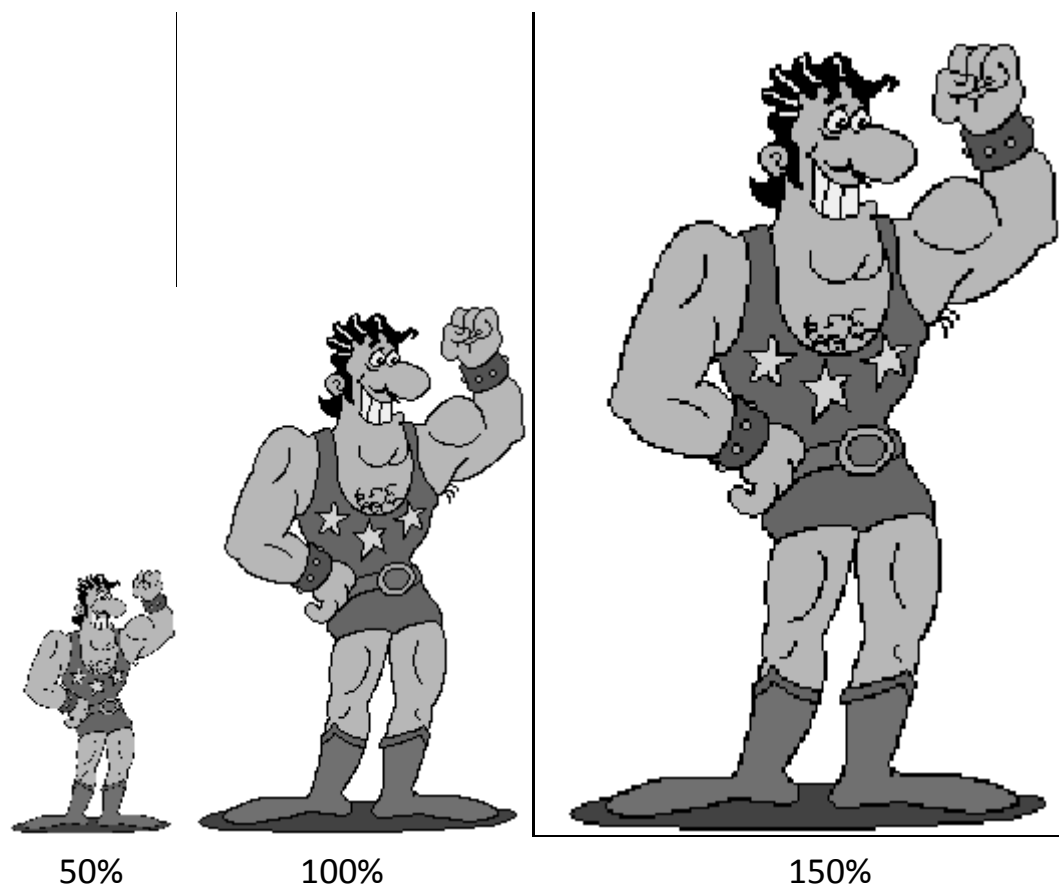
imagen2, es la imagen creada, e imagen es la imagen de la que partimos. En la imagen2(i,j) tenemos que insertar un pixel, para ello aplicamos la fórmula para elegir cual es la mejor i cual es la mejor j de la imagen original



En esta imagen podemos ver en que consiste, tenemos que rellenar la Imagen Redimensionada eligiendo puntos de la Imagen original.

Por esto el vecino más cercano no es un buen método, selecciona un solo punto de la original.

Aquí tenemos un ejemplo de una figura redimensionada al 50% y al 150%



Como vemos la imagen sufre cambios y podemos ver bastante claros los pixeles, se ven las imágenes pixeladas.

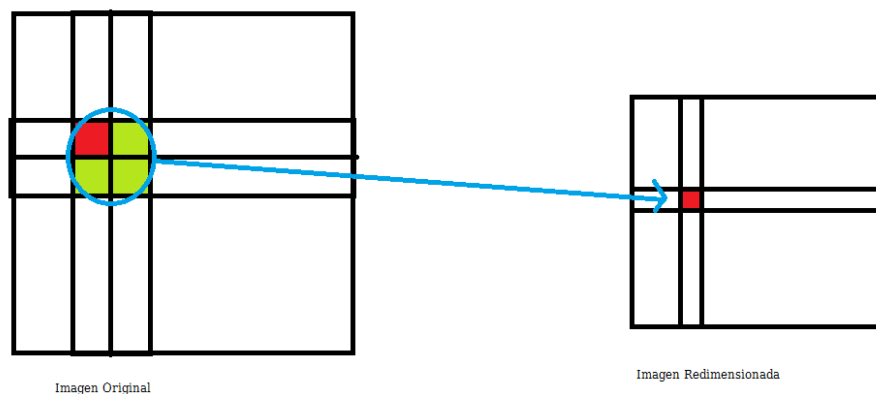
NOTA: La tercera imagen no genera esa línea lateral, lo ha generado Word al poner 3 imágenes consecutivas

# Método: Interpolación Lineal

El método de interpolación lineal es algo más complejo, aunque una vez comprendido el método del vecino más cercano y cómo funciona el tratamiento de imágenes no resulta más sencillo desarrollarlo.

Este método consiste en elegir varios puntos de uno dado y realizar una media de valores, es decir, vamos a elegir un punto, como hacíamos en el vecino más cercano, y además, vamos a coger sus 4 puntos colindantes y vamos a realizar una media de valores, así conseguiremos un valor medio y aproximado.

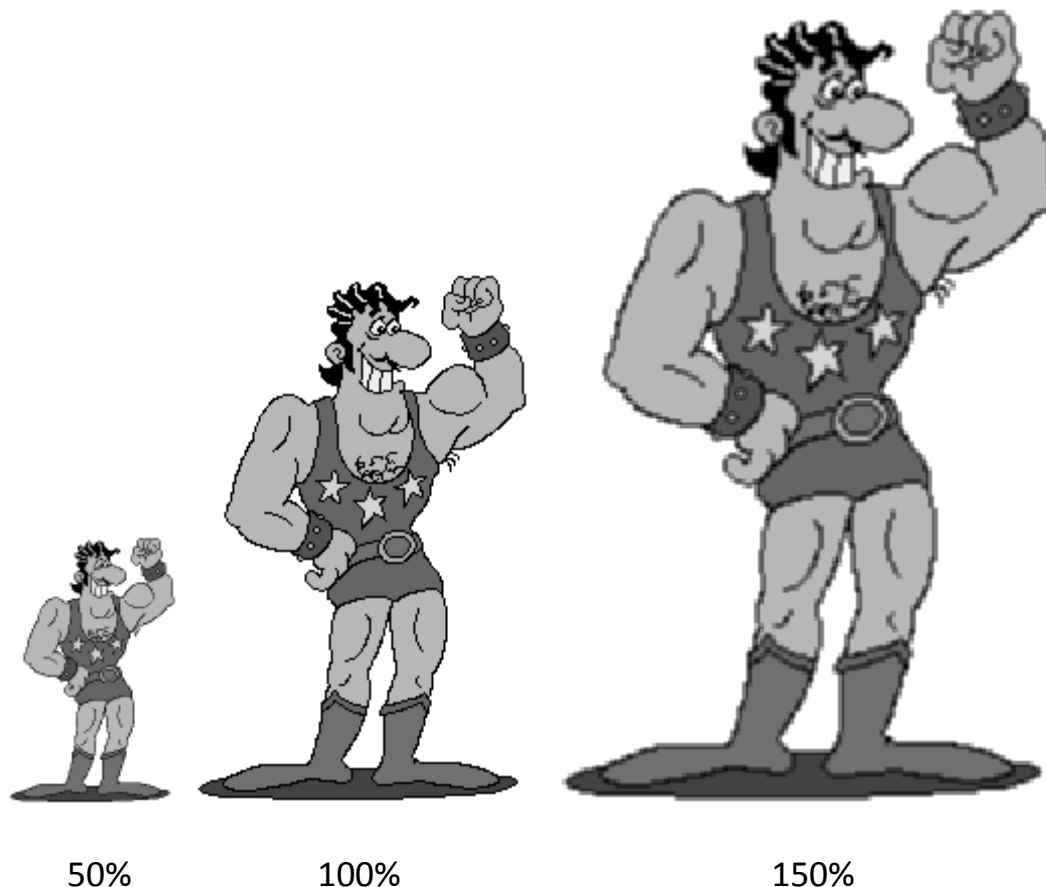
La fórmula a aplicar es la misma pero algo modificado, para verlo gráficamente:



Elegimos un punto de la imagen original, y seleccionamos esos puntos cercanos, con esos 4 puntos realizamos una media de valores para elegir un valor medio para la imagen redimensionada.

Hacer esto es un proceso un poco más lento pero da un resultado mucho mejor.

Aquí se puede ver con un ejemplo como en el anterior con tres imágenes cada una a un redimensionamiento de 50%, 100% y 150%.



Se aprecia mucho el cambio en la imagen del 50%, ahora se ve mucho mejor.






```
imagen2(i, j) = (
    imagen(((i - 1)*z1) + 1           ,      ((j - 1)*z2) + 1)      +
    imagen(((i - 1)*z1) + 1 + a      ,      ((j - 1)*z2) + 1)      +
    imagen(((i - 1)*z1) + 1           ,      ((j - 1)*z2) + 1 + b)  +
    imagen(((i - 1)*z1) + 1 + a      ,      ((j - 1)*z2) + 1 + b)
)/4;
```

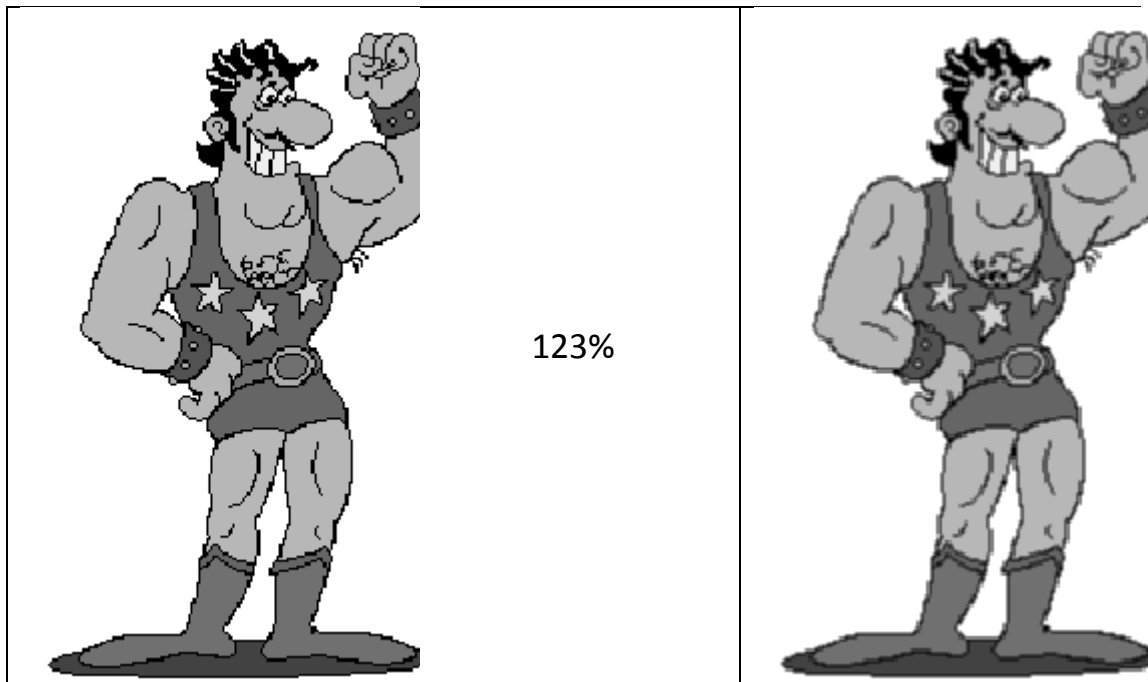
Como podemos ver en el código usamos la misma fórmula que usábamos para el vecino más cercano para seleccionar un punto, y con las letras a y b nos ayudamos a seleccionar los puntos cercanos a ese punto seleccionado.



# Comparaciones

Dado que el código funciona con cualquier imagen y con cualquier porcentaje, para esta comparación vamos a usar "Hercules\_Grey.bmp" dado que es una imagen donde se aprecia bastante el cambio.

<p>Método Vecino más Cercano</p>	<p>Imagen Original</p> 	<p>Método Interpolación Lineal</p>
	<p>23%</p>	
	<p>78%</p>	



En términos de redimensión inferior vemos como el método de interpolación lineal es considerablemente mejor dado que se aprecia mejor la imagen.

Y como observamos cuando redimensionamos para aumentar el tamaño el método de interpolación lineal sigue dando mejores resultados.

Podemos comprobar rápidamente con otra imagen la mayor diferencia.



Con imágenes en blanco y negro, la interpolación lineal nos consigue grises, que suavizan el contorno, pero perdemos la característica de tenerla en dos colores, pasamos a tener grises.

# Código

```
#include <C_General.hpp>
#include <C_Trace.hpp>
#include <C_File.hpp>
#include <C_Arguments.hpp>
#include <C_Matrix.hpp>
#include <C_Image.hpp>
#include <C_Arguments.hpp>
#include <iostream>

using namespace std;

//Definicion de los métodos que vamos a utilizar
void RedVecCer(C_Image imagen, int nanchoImg, int naltoImg);
void RedLineal(C_Image imagen, int nanchoImg, int naltoImg);
void Pause();

int main()
{
    C_Image imagen;
    double porcentaje;
    string nombre;
    int metodo;
    int anchoImg;
    int altoImg;
    int nanchoImg;
    int naltoImg;
    string nombreImg;

    //Nos debe dar el nombre de la imagen añadiendo también la extensión
    cout << "Introduce el nombre de la imagen que desea redimensionar: " <<
endl;
    cin >> nombreImg;
    //Transformamos el string en una char para poder usar la funcion ReadBMP
    const char * cadena = nombreImg.c_str();
    imagen.ReadBMP(cadena);

    cout<<"Leyendo imagen"<<endl;

    anchoImg = imagen.LastCol();
    altoImg = imagen.LastRow();

    //Con este bucle vamos a permitir usar el programa varias veces sin tener
    que parar y volver a ejecutar
    do{

        //Dame el valor al que quieres redimensionar
        // 100% la deja igual -> 50% ponerla a la mitad 200% implica
        duplicar el tamaño
        cout << "Introduce el porcentaje de redimensionamiento (No es
necesario insertar %): " <<endl;
        cin >> porcentaje;

        //Pasamos el porcentaje a numero decimal y creamos las nuevas
        medidas
        porcentaje = porcentaje / 100;
        nanchoImg = anchoImg * porcentaje;
        naltoImg = altoImg * porcentaje;
```

```
        //Con este bucle prevenimos de que no se inserten otros valores que  
no sean los del menú
```

```
        do{  
  
            cout << "****Introduce el metodo mediante el que desea  
redimensionar**** " << endl;  
            cout << "1 --> Interpolacion Lineal" << endl;  
            cout << "2 --> Metodo Vecino mas cercano" << endl;  
            cout << "3 --> Salir" << endl;  
            cin >> metodo;  
        } while (metodo < 0 && metodo >= 4);
```

```
        switch (metodo){  
        case 1:  
            RedLineal(imagen, nanchoImg, naltoImg);  
            break;  
        case 2:  
            RedVecCer(imagen, nanchoImg, naltoImg);  
            break;  
        case 3:  
            cout << "ADIOS" << endl;  
            Pause();  
            return 0;  
            break;  
        };//Fin del switch  
  
    } while (true); //Fin del bucle de programa  
    return 0;  
} //Fin del MAIN
```

```
//Redimensionamiento por interpolación Lineal
```

```
//En un punto juntaremos 4 puntos de la imagen anterior
```

```
void RedLineal(C_Image imagen, int nanchoImg, int naltoImg){  
    C_Image imagen2(1, naltoImg, 1, nanchoImg);
```

```
    double x1 = imagen.LastRow() - 1;  
    double y1 = imagen.LastCol() - 1;  
    double x2 = imagen2.LastRow() - 1;  
    double y2 = imagen2.LastCol() - 1;  
    double z1 = x1 / x2;  
    double z2 = y1 / y2;  
    int a = 1;  
    int b = 1;
```

```
    for (int i = 1; i <= naltoImg; i++){  
        for (int j = 1; j <= nanchoImg; j++){  
            a = 1;  
            b = 1;
```

```
        //Con estos if prevenimos salirnos de la matriz a la hora de  
elegir los puntos
```

```
        if (i == naltoImg){  
            a = 0;  
        }  
        if (j == nanchoImg){  
            b = 0;  
        }  
    }
```

```

        imagen2(i, j) = (
            // Ahora tenemos que hacer esto 4 veces y hacer la
            media
            //Vamos a elegir un punto y ademas el de su derecha,
            el de abajo y el diagonal
            //Con estos 4 valores vamos a realizar una media
            imagen(((i - 1)*z1) + 1, ((j -
1)*z2) + 1) +
            imagen(((i - 1)*z1) + 1 + a, ((j - 1)*z2)
+ 1) +
            imagen(((i - 1)*z1) + 1, ((j -
1)*z2) + 1 + b) +
            imagen(((i - 1)*z1) + 1 + a, ((j - 1)*z2)
+ 1 + b)
        )/4;
    }
}

cout << "LISTO" << endl;
imagen2.WriteBMP("RedLineal.bmp");
Pause();
}

```

```

//Redimensionamiento por vecino más cercano
void RedVecCer(C_Image imagen, int nanchoImg, int naltoImg){
    C_Image imagen2(1, naltoImg, 1, nanchoImg);

    double x1 = imagen.LastRow() - 1;
    double y1 = imagen.LastCol() - 1;
    double x2 = imagen2.LastRow() - 1;
    double y2 = imagen2.LastCol() - 1;
    double z1 = x1 / x2;
    double z2 = y1 / y2;

    for (int i = 1; i <= naltoImg; i++){
        for (int j = 1; j <= nanchoImg; j++){
            imagen2(i, j) = imagen(((i - 1)*z1) + 1
((j - 1)*z2)
+ 1);
        }
    }

    cout << "LISTO" << endl;
    imagen2.WriteBMP("RedVecino.bmp");
    Pause();
}

```

# Bibliografía

Hay muchos más métodos y muchas maneras de realizarlos, yo he decido hacerlos sobre estos dos. Aquí dejo una serie de enlaces a fuentes donde podemos encontrar mucha más información y que me han sido de gran utilidad.

Desde este enlace a la página de Microsoft podemos encontrar todos los métodos que ya están desarrollados por Microsoft y podemos encontrar entre sus librerías.

- <https://msdn.microsoft.com/es-es/library/system.drawing.drawing2d.interpolationmode.aspx>
- <http://pendientedemigracion.ucm.es/info/sevipres/P4/01/ANEXOS01.php>
- [http://www.uv.es/diazj/cn\\_tema5](http://www.uv.es/diazj/cn_tema5)