

# Arquitecturas Paralelas Imágenes, C y OpenCV

Versión 1.0, 27 de enero de 2019

## 1. Objetivos

- Familiarizarse con el entorno Netbeans, la programación en C, los punteros y la biblioteca de procesamiento de imágenes OpenCV.
- Saber como se almacena una imagen RGB TrueColor.
- Saber acceder a los valores RGB (3 canales) de cualquier pixel de una imagen TrueColor.
- Saber realizar operaciones básicas sobre una imagen RGB TrueColor.
- Saber utilizar punteros y castings para acceder al valor de un pixel de una imagen.

## 2. Ejercicios

1. Compilar y ejecutar el código ejemplo con una imagen TrueColor, es decir, para una profundidad de color de 24 bits por pixel. En Netbeans será necesario indicar los directorios donde se encuentran los archivos de cabecera (ficheros include, \*.h) y las librerías (librerías de enlace dinámico, \*.dll). Mostrar primero la imagen letrasRGB.png, y posteriormente fruits.jpg, tal como indica la figura 1.



(a) Imagen letras.png.



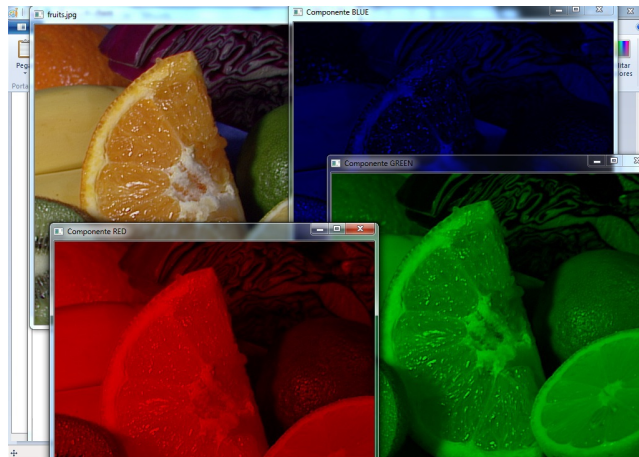
(b) Imagen fruits.jpg.

Figura 1: Imágenes utilizadas

2. Mostrar 3 imágenes tal como muestra la figura 2, cada una con una componente RGB. Para el fichero letrasRGB.png, la imagen de la componente azul tiene el fondo azul, las letras en azul no aparecen y las letras rojas y verdes aparecen negras. ¿Por qué?
3. Mostrar una nueva imagen cuya parte superior sea la parte inferior de fruits, y cuya parte inferior sea la parte superior. El resultado debe ser el que indica la figura 3.
4. Crear una imagen de 1 único canal de tamaño 256x256. El nivel de gris de cada fila será igual al índice de la propia fila.



(a) Imagen letrasRGB.



(b) Imagen fruits.

Figura 2: Solución del ejercicio.



(a) Imagen inicial.



(b) Imagen resultado.

Figura 3: Solución del ejercicio.

### 3. Ejercicios Propuestos

1. Se propone crear una nueva imagen compuesta a partir de otra ya existente tal como muestra la figura 4. En esta nueva imagen se podrá ver en una columna, la imagen original y sus tres componentes, roja, verde y azul.



Figura 4: Solución del ejercicio propuesto.

2. Mostrar una nueva imagen más grande que fruits, en la que se encuentre la imagen fruits entrada a lo ancho y a lo alto. puntos donde no haya imagen serán negros. El resultado debe ser el que indica la figura 5.
3. Mostrar una nueva imagen cuya parte derecha sea la parte izquierda de fruits, y cuya parte izquierda sea la parte derecha. El resultado debe ser el que indica la figura 6.
4. Crear una imagen de 1 canal que sea un tablero de ajedrez. El tamaño de la imagen será de 512x512 pixels.
5. Crear una imagen de 3 canales que sea un tablero de ajedrez. El tamaño de la imagen será de 512x512 pixels.
6. Crear una imagen de 4 canales que sea un tablero de ajedrez. El tamaño de la imagen será de 512x512 pixels.



(a) Imagen inicial.



(b) Imagen resultado.

Figura 5: Solución del ejercicio propuesto.



(a) Imagen inicial.



(b) Imagen resultado.

Figura 6: Solución del ejercicio propuesto.

## 4. Imágenes

### 4.1. Little endian

Si la profundidad de color de un pixel es de 24 bits, cada una de las tres componentes de color, rojo, verde y azul, se representan por 8 bits. Por lo tanto cada componente de color puede valer de 0 a 255, de 00h a FFh. La distribución de las tres componentes es la siguiente:

```
+---+---+---+
| -R- | -G- | -B- |
+---+---+---+
```

El primer byte representa el nivel de rojo, el segundo el de verde y el tercero el de azul. Como el valor de color de un pixel es de más de un byte, y debe ser almacenado en una memoria direccionada a nivel de byte, los tres bytes pueden ser almacenados en dos órdenes distintos. Como la arquitectura Intel es little endian, el byte menos significativo se almacena en la dirección más baja y el byte más significativo en la dirección más alta.

```
+---+
| -B- |
+---+
| -G- |
+---+
| -R- |
+---+
```

Por lo tanto, a pesar de que hablamos siempre del modelo de color RGB, la primera componente de color que encontramos almacenada en memoria es la componente azul.

### 4.2. Almacenamiento de una imagen en memoria

Una imagen de F filas y C columnas se almacena en memoria a partir de la dirección `img->imageData` tal como indica la figura 7. En la figura 8 se puede observar como acceder a la información de cualquier pixel de la imagen. El punto de partida siempre es `img->imageData`, es decir, la dirección en memoria a partir de la cual se almacena la información de los píxeles de una imagen. Para acceder directamente al pixel fila f y columna c, hay que sumarle a esta dirección lo que ocupan en memoria las f filas previas, y las c columnas previas en la fila donde está.

### 4.3. Tipo de datos de una componente de color

Cada componente de color es un byte y su rango de valores es desde 00h a FFh. Por lo tanto, es un valor entero sin signo de un byte. En el lenguaje C el tipo de datos correspondiente es un `unsigned char`.

### 4.4. Luminancia

Se puede considerar que el equivalente psicológico de la luminancia es el brillo. Considerando el caso de la emisión o reflexión de luz por parte de superficies planas y difusas, la luminancia indicaría la cantidad de flujo luminoso que el ojo percibiría para un punto de vista particular.

En una señal de vídeo, la luminancia es la componente que codifica la información de luminosidad de la imagen. En términos generales, es algo muy similar a la versión en blanco y negro de la imagen original, aunque realmente se debería hablar de escala de grises. Luminancia y crominancia combinadas proporcionan la señal denominada señal de vídeo compuesto. Forman parte de la codificación de vídeo en los estándares de TV NTSC y PAL, entre otros.

Es un término muy utilizado en el tratamiento digital de imágenes para caracterizar a cada pixel. En el sistema de color RGB, la luminancia Y de un pixel se calcula a partir de la siguiente expresión matemática:

$$Y = 0.299R + 0.587G + 0.114B$$

Una imagen en color puede aparecer como una imagen en blanco y negro o escala de grises, cuando el valor de todos los pixels es el mismo. Si el valor de un pixel RGB es 00h 00h 00h se verá como negro, FFh FFh FFh como blanco, y cualquier otro valor, XXh XXh XXh, como determinado gris.

Si alguien escribe esta línea de código tendrá problemas:

```
unsigned char luminancia = 0.114 * (*pImg++) + 0.587 * (*pImg++) + 0.299 * (*pImg++);
```



Imagen de F filas y C columnas					
	0	1	2	C	
0	pixel 0,0	pixel 0,1	pixel 0,2	...	pixel 0,C
1	pixel 1,0	pixel 1,1	pixel 1,2	...	pixel 1,C
...	...	...	...	...	...
F	pixel F,0	pixel F,1	pixel F,2	...	pixel F,C

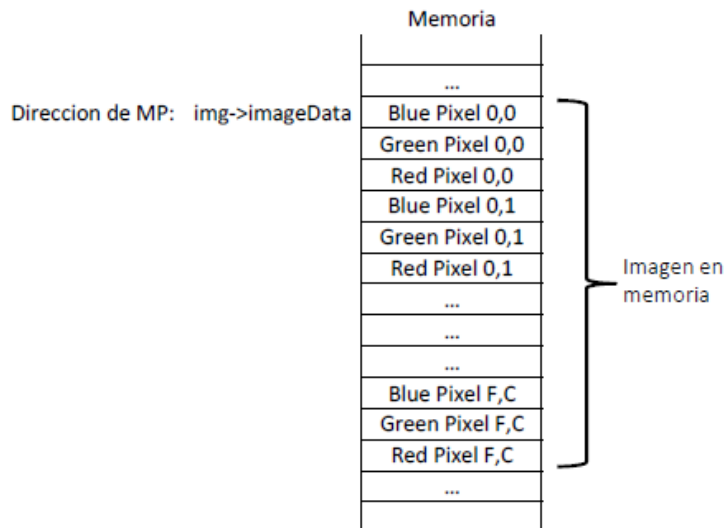


Figura 7: Almacenamiento de una imagen en memoria

¿Por qué? Al escribir la anterior línea estamos suponiendo que el compilador va a hacer los calculos en el orden que está escrita la expresión. Pero no es así, el compilador puede crear código ejecutable que haga la misma expresión pero en otro orden, por ejemplo,  $0.299 * (*pImg++) + 0.114 * (*pImg++) + 0.587 * (*pImg++)$ . Si no usamos punteros la propiedad conmutativa de la suma garantiza que el resultado es siempre igual. Pero en nuestro caso al usar punteros, primero multiplicamos 0.299 coeficiente del rojo por  $(*pImg++)$ , que es la componente azul.

## 5. OpenCV

### 5.1. `cvLoadImage`

IPL image header

```
typedef struct _IplImage {
    int nSize;
    int ID;
    int nChannels;
    int alphaChannel;
    int depth;
    char colorModel[4];
    char channelSeq[4];
    int origin;
    int align;
    int width;
    int height;
    struct _IplROI *roi;
    struct _IplImage *maskROI;
    void *imageId;
    struct _IplTileInfo *tileInfo;
    int imageSize;
    char *imageData;
}
```

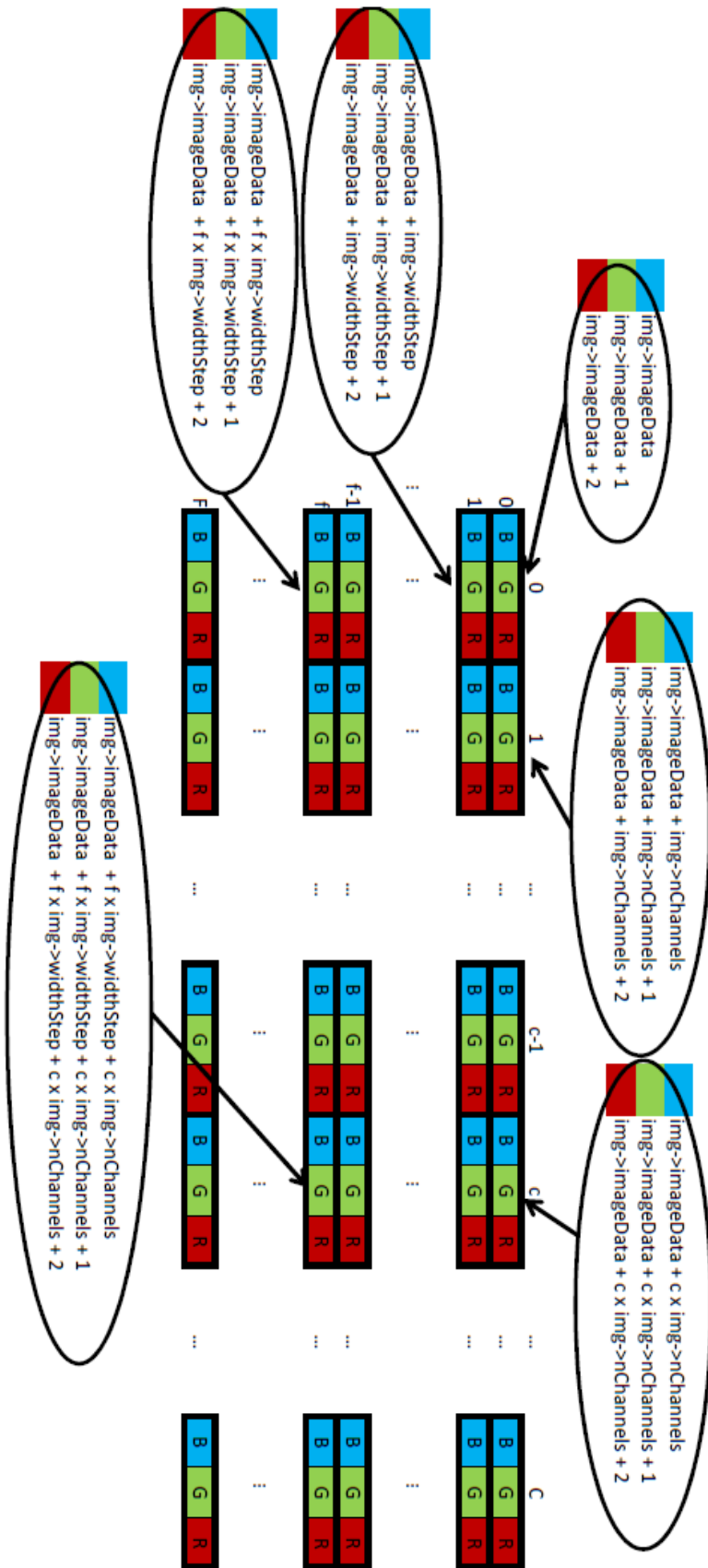


Imagen de  $F \times C$  pixels, 3 canales por pixel  
 $\text{img->widthStep} = C \times \text{img->nChannels} + [0, 1, 2, 3]$ . El tamaño de una fila debe ser múltiplo de 4

Figura 8: Detalle las direcciones de memoria de los píxeles y sus componentes de color.

```

    int widthStep;
    int BorderMode[4];
    int BorderConst[4];
    char *imageDataOrigin;
}
IplImage;

```

nSize sizeof(IplImage)  
 ID Version, always equals 0  
 nChannels Number of channels. Most OpenCV functions support 1-4 channels.  
 alphaChannel Ignored by OpenCV  
 depth Channel depth in bits + the optional sign bit (IPL\_DEPTH\_SIGN). The supported depths are: IPL\_DEPTH\_8U Unsigned 8-bit integer IPL\_DEPTH\_8S Signed 8-bit integer IPL\_DEPTH\_16U Unsigned 16-bit integer IPL\_DEPTH\_16S Signed 16-bit integer IPL\_DEPTH\_32S Signed 32-bit integer IPL\_DEPTH\_32F Single-precision floating point IPL\_DEPTH\_64F Double-precision floating point  
 colorModel Ignored by OpenCV. The OpenCV function CvtColor requires the source and destination color spaces as parameters. channelSeq Ignored by OpenCV  
 dataOrder 0 = IPL\_DATA\_ORDER\_PIXEL - interleaved color channels, 1 - separate color channels. CreateImage only creates images with interleaved channels. For example, the usual layout of a color image is: b00g00r00b10g10r10:::  
 origin 0 - top-left origin, 1 - bottom-left origin (Windows bitmap style) align Alignment of image rows (4 or 8). OpenCV ignores this and uses widthStep instead.  
 width Image width in pixels  
 height Image height in pixels  
 roi Region Of Interest (ROI). If not NULL, only this image region will be processed.  
 maskROI Must be NULL in OpenCV  
 imageId Must be NULL in OpenCV  
 tileInfo Must be NULL in OpenCV  
 imageSize Image data size in bytes. For interleaved data, this equals image->height image->widthStep  
 imageData A pointer to the aligned image data  
 widthStep The size of an aligned image row, in bytes  
 BorderMode Border completion mode, ignored by OpenCV  
 BorderConst Border completion mode, ignored by OpenCV  
 imageDataOrigin A pointer to the origin of the image data (not necessarily aligned). This is used for image deallocation.

The IplImage structure was inherited from the Intel Image Processing Library, in which the format is native. OpenCV only supports a subset of possible IplImage formats, as outlined in the parameter list above. In addition to the above restrictions, OpenCV handles ROIs differently. OpenCV functions require that the image size or ROI size of all source and destination images match exactly. On the other hand, the Intel Image Processing Library processes the area of intersection between the source and destination images (or ROIs), allowing them to vary independently.

## 5.2. cvLoadImage

Carga una imagen desde un fichero a memoria. En memoria toda la información de la imagen se encuentra almacenada como una estructura de datos IplImage.

```

IplImage* cvLoadImage(
    const char* filename,
    int iscolor = CV_LOAD_IMAGE_COLOR);

```

filename Name of file to be loaded. iscolor Specific color type of the loaded image: CV\_LOAD\_IMAGE\_COLOR the loaded image is forced to be a 3-channel color image CV\_LOAD\_IMAGE\_GRAYSCALE the loaded image is forced to be grayscale CV\_LOAD\_IMAGE\_UNCHANGED the loaded image will be loaded as is. The function cvLoadImage loads an image from the specified file and returns the pointer to the loaded image. Currently the following file formats are supported:

- Windows bitmaps - BMP, DIB
- JPEG files - JPEG, JPG, JPE
- Portable Network Graphics - PNG



- Portable image format - PBM, PGM, PPM
- Sun rasters - SR, RAS
- TIFF files - TIFF, TIF

Note that in the current implementation the alpha channel, if any, is stripped from the output image, e.g. 4-channel RGBA image will be loaded as RGB.

### 5.3. cvNamedWindow

Crea una ventana. Toda ventana necesita un identificador para posteriormente hacer referencia a ella. El identificador de la ventana es el texto que se pasa como parámetro en esta función, y aparece posteriormente como el título de la ventana.

```
int cvNamedWindow( const char* name, int flags );
```

name Name of the window in the window caption that may be used as a window identifier. flags Flags of the window. Currently the only supported flag is CV\_WINDOW\_AUTOSIZE. If this is set, window size is automatically adjusted to fit the displayed image (see ShowImage ), and the user can not change the window size manually.

The function cvNamedWindow creates a window which can be used as a placeholder for images and trackbars. Created windows are referred to by their names. If a window with the same name already exists, the function does nothing.

### 5.4. cvShowImage

Muestra la imagen almacenada en la estructura de datos IplImagen en la ventana indicada.

```
void cvShowImage( const char* name, const CvArr* image );
```

name Name of the window. image Image to be shown.

The function cvShowImage shows the image in the specified window. If the window was created with CV\_WINDOW\_AUTOSIZE flag then the image is shown with its original size, otherwise the image is scaled to fit the window.

### 5.5. cvDestroyWindow

Destroys a window.

```
void cvDestroyWindow( const char* name );
```

name Name of the window to be destroyed.

The function cvDestroyWindow destroys the window with the given name.

### 5.6. cvCreateImage

Creates an image header and allocates the image data.

```
IplImage* cvCreateImage(CvSize size, int depth, int channels);
```

size Image width and height depth Bit depth of image elements. See IplImage for valid depths. channels Number of channels per pixel. See IplImage for details. This function only creates images with interleaved channels.

cvCreateImage se usa conjuntamente con la estructura CvSize y la función cvSize

```
typedef struct CvSize
{
    int width; /* anchura en pixels */
    int height; /* altura en pixels */
}
CvSize;
```

```
inline CvSize cvSize( int width, int height );
```

Vemos un ejemplo de aplicación:

```
CvSize tamano = cvSize(100,200);
```

o

```
// crea una imagen de 800 pixels de ancho, 600 pixels de alto, cada canal/color se  
representa por un entero de 8 bits sin signo, y 3 canales/colores  
img = cvCreateImage(cvSize(800, 600), IPL_DEPTH_8U, 3);
```

## 5.7. cvCloneImage

Makes a full copy of an image, including the header, data, and ROI.

```
IplImage* cvCloneImage(const IplImage* image);
```

image The original image The returned IplImage\* points to the image copy.  
Ejemplo:

```
IplImage* image = NULL;  
IplImage* imageDraw = NULL;  
image = cvLoadImage(argv[1], CV_LOAD_IMAGE_COLOR);  
imageDraw = cvCloneImage(image);
```

## 5.8. cvReleaseImage

Deallocates the image header and the image data.

```
void cvReleaseImage(IplImage** image);
```

image Double pointer to the image header

## 5.9. cvWaitKey

Waits for a pressed key.

```
delay Delay in milliseconds.
```

```
int cvWaitKey( int delay=0 );
```

The function cvWaitKey waits for key event infinitely (delay <= 0) or for delay milliseconds. Returns the code of the pressed key or -1 if no key was pressed before the specified time had elapsed. Note: This function is the only method in HighGUI that can fetch and handle events, so it needs to be called periodically for normal event processing, unless HighGUI is used within some environment that takes care of event processing.

## 5.10. Casting en C: paso de \*char a \*unsigned char

Un casting en C es indicar al compilador que una variable anteriormente definida de un determinado tipo, sea tratada como si fuese de un tipo distinto. Para realizar un casting de una variable, se pone antes de ella entre paréntesis el nuevo tipo con el cual la queremos considerar. Tenemos los siguientes ejemplos (de <http://www.aui.ma/personal/~O.Iraqi/csc1401/casting.htm>):

```

1 pthread_t threads[NTHREADS];
2
3 int i;
4 int indice[NTHREADS];
5 for (i = 0; i < NTHREADS; i++) {
6     indice[i] = i;
7     pthread_create(&threads[i], NULL, (void *) &mosaico_thread, (void *) &indice[
8         i]);
9 }
10
11 for (i = 0; i < NTHREADS; i++) {
12     pthread_join(threads[i], NULL);
13 }

```

En OpenCV el campo `imageData` del tipo de dato `IplImage` se define como `char *`. Realmente el campo `imageData` apunta a una zona de memoria donde se encuentra la información de los pixels, tal como vimos en 4.1, la información de los pixels puede ser del tipo Unsigned 8-bit integer, Signed 8-bit integer, Unsigned 16-bit integer, Signed 16-bit integer, Signed 32-bit integer, Single-precision floating point, Double-precision floating point. Por lo tanto, la solución que adoptó OpenCV es dar al campo `imageData` el tipo genérico `char *`, y después el programador debe hacer un casting al tipo de dato que esté usando.

En `TrueColor` cada pixel se define por tres bytes, con valores de 00h a FFh, es decir, como un entero e 8 bits sin signo. Por lo tanto será necesario hacer el siguiente casting:

```

1 IplImage *in, *out;
2 unsigned char *ptrIn, *ptrOut;
3 ...
4 ptrIn = (unsigned char*) (in->imageData);
5 ptrOut = (unsigned char*) (output->imageData);

```

## 5.11. Alineación en memoria del campo `imageData` en `IplImage`

El campo `imageData` de un `IplImage` está alineado en 4 o 8 bytes para mejorar la velocidad de procesado. Esto quiere decir que cada fila de pixels empieza en una dirección que es múltiplo entero de 4 o 8. Como consecuencia podemos considerar que una fila de pixels ocupa un múltiplo entero de 4 o 8 bytes.

Consideremos el siguiente ejemplo donde campo `imageData` de `IplImage` está alineado en 4 bytes. La imagen es en color `TrueColor` y de tamaño 176 filas por 74 columnas. El tamaño de una fila no es  $74 \times 3 = 222$  bytes, el tamaño de cada fila es 224 bytes. El motivo ya ha sido explicado, cada fila debe empezar en una dirección múltiplo de 4 o ocupar en memoria un múltiplo de 4 bytes. El múltiplo de 4 mayor que 222 y más cercano a 222 es 224. Esto no ocurre cuando por ejemplo una imagen en color `TrueColor` y de tamaño 100 filas por 100 columnas.

Para facilitar el trabajo y olvidarse de la alineación en memoria, uno de los parámetros de `IplImage` es `widthStep`, que indica el número de bytes utilizados para representar cada fila de una imagen. Por ejemplo, la imagen de anchura 74 píxeles y color `TrueColor` tiene un `widthStep` igual a 224. Cada fila ocupa 224 bytes, de los cuales los 222 primeros se corresponderán con los píxeles de cada fila, y los 2 restantes en este ejemplo OpenCv los rellena con `'0'`.

Si la imagen es en blanco y negro cada pixel ocupará un byte. Si una imagen de este tipo con una anchura 101 píxeles, el valor de `widthStep` es igual a 104. Cada fila es una ristra de 104 bytes, de los cuales los 101 primeros se corresponden con los píxeles de cada fila, los 3 restantes valen `'0'`.

El siguiente ejemplo de función toma una imagen en color y devuelve una imagen en blanco y negro. Para la imagen devuelta, un pixel es blanco si el correspondiente pixel de la imagen en color supera cierto umbral, y negro en caso contrario.

```

1 void SimpleThreshold(const IplImage* in, IplImage * output) {
2     int i, j;
3     unsigned char *ptrIn, *ptrOut;
4     for (i = 0; i < in->height; i++) {
5         ptrIn = (unsigned char*) (in->imageData + i * in->widthStep);
6         ptrOut = (unsigned char*) (output->imageData + i * output->widthStep);
7         for (j = 0; j < in->width; j++) {

```

```

8         If(ptrIn[3 * j + 0] > 120 && ptrIn[3 * j + 1] > 120 &&
9             ptrIn[3 * j + 2] > 120)
10            ptrOut[j] = 255;
11        else
12            ptrOut[j] = 0;
13    }
14 }
15 }

```

<http://www.bit-bangers.com/archives/236>

<http://chi3x10.wordpress.com/2008/05/07/be-aware-of-memory-alignment-of-iplimage-in-opencv/>

<http://blogs.ua.es/pablosuau/2008/06/11/manejo-del-widthstep-en-las-imagenes-de-opencv/>

## 5.12. Recorrer una imagen

Una imagen se puede recorrer componente a componente de color o bien pixel a pixel. Dependiendo de lo que queramos hacer con la imagen, debemos elegir una forma u otra.

Si queremos recorrer la imagen y hacer una única operación en cada una de las 3 componentes de color RGB, podemos recorrer la imagen componente a componente de color. No es el caso más habitual.

```

1 // convierte una imagen en negra
2 int cc;
3 char *pImg = imgOriginal->imageData;
4 for (cc = 0; cc < imgOriginal->imageSize; cc++) {
5     *pImg++ = 0;
6 }

```

Muchas veces no es posible utilizar el acceso componente a componente, pues debido a la alineación de las filas en memoria, al final de una fila puede haber de 1 a 3 bytes de relleno. El siguiente código convierte una imagen en azul, pero sólo funciona correctamente si una fila de pixels en memoria ocupa exactamente un número múltiplo de 4 bytes, es decir, si `imgOriginal->stepSize = imgOriginal->height * imgOriginal->nChannels`.

```

1 // convierte una imagen en azul
2 int cc;
3 char *pImg = imgOriginal->imageData;
4 for (cc = 0; cc < imgOriginal->imageSize; cc = +3) {
5     *pImg++ = 255;
6     *pImg++ = 0;
7     *pImg++ = 0;
8 }
9

```

Los más frecuente es que necesitemos hacer operaciones a nivel de pixel, con lo cual necesitaremos recorrer la imagen por filas y columnas, es decir, si seleccionamos una fila y una columna entonces seleccionamos un pixel. Este código convierte una imagen en azul y funciona siempre independientemente del tamaño de una fila.

```

1 int fila, columna;
2 char *pImg = imgOriginal->imageData;
3 for (fila = 0; fila < imgOriginal->height; fila++) {
4     for (columna = 0; columna < imgOriginal->width; columna++) {
5         *pImg++ = 255;
6         *pImg++ = 0;
7         *pImg++ = 0;
8     }
9 }

```

## 5.13. Crear una imagen en blanco y negro o escala de grises

El campo `nChannels` indica el número de canales de color de la imagen. Las imágenes en escala de grises tienen un sólo canal, mientras que las de color tienen 3 o 4 canales.

Las imágenes en escala de grises se almacenan utilizando un único canal, es éste se almacena 1 byte por píxel que permite codificar 255 niveles de gris distintos: siendo 0 el negro y 255 el blanco.

En la imágenes de color cada píxel se codifica con tres canales, donde en cada canal se almacena la información de los tres colores rojo, verde y azul. Cada uno de estos colores se representa por 1 byte. Por tanto, cada píxel se codificará con 3 bytes, siendo el número total de colores posibles del orden de 16,7 millones.

Aunque también existen imágenes RGBA de cuatro canales, el cuarto canal denominado alphaChannel el cual indica el nivel de transparencia de un píxel, es ignorado por OpenCV. Tiene utilidad en con aplicaciones en visión nocturna o imágenes por satélite.

## 5.14. Includes y librerías de OpenCV 3.0.0

Los ficheros de cabecera o .h para OpenCV se encuentran en el directorio c:/OpenCV3.0.0/include. En ellos se encuentran las definiciones de los tipos de datos y los prototipos de funciones que aporta OpenCV. En las opciones del compilador deben ser añadidos los necesarios para nuestra aplicación ejemplo:

```
#include <opencv/cv.h>
#include <opencv/highgui.h>
```

Las funciones de OpenCV se encuentran en forma de biblioteca de enlace dinámico o DLL, en el directorio c:/OpenCV3.0.0/x86 o c:/OpenCV3.0.0/x64 según trabajemos en 32 o 64 bits respectivamente. La biblioteca necesaria es opencv\_world300.dll, dentro de este fichero se encuentra el código de todas las funciones del apartado 5 que vamos a utilizar. En opciones del linkador añadiremos el directorio correspondiente y el fichero opencv\_world300.dll.

En la figura 9 se puede ver el proceso de compilación y linkado o montaje de un programa.

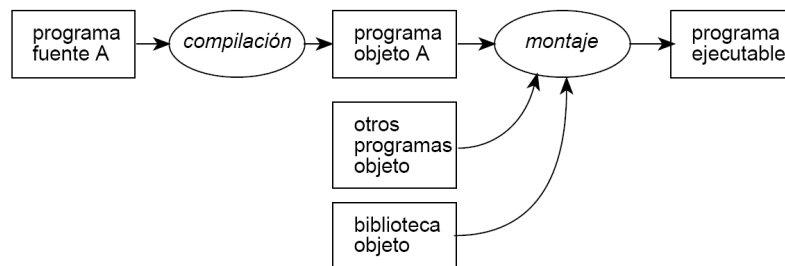


Figura 9: Compilado y linkado de un programa

Un prototipo de función le da información importante al compilador. En el prototipo se indican el tipo de dato que retorna la función, el número, tipo y orden de parámetros que recibe la misma. El compilador utiliza los prototipos para verificar las llamadas a funciones. El prototipo de la función siguiente:

```
int maximo(int, int);
```

indica que la función de nombre máximo retorna como resultado un valor de tipo entero. Además informa que la función debe ser llamada con dos parámetros del tipo entero también.

Por ejemplo, el prototipo de la función cvLoadImage es:

```
CVAPI(IplImage*) cvLoadImage( const char* filename, int iscolor CV_DEFAULT(
    CV_LOAD_IMAGE_COLOR));
```

## 6. Programación C

### 6.1. Punteros

Un puntero no es más que una variable que almacena una dirección de memoria. En una máquina de 32 bits, un puntero es un variable de 4 bytes que almacena un entero entre 00000000h y FFFFFFFFh. En una máquina de 64 bits, un puntero es una variable de 8 bytes que almacena un entero entre 0000000000000000h y FFFFFFFFFFFFFFFFh.

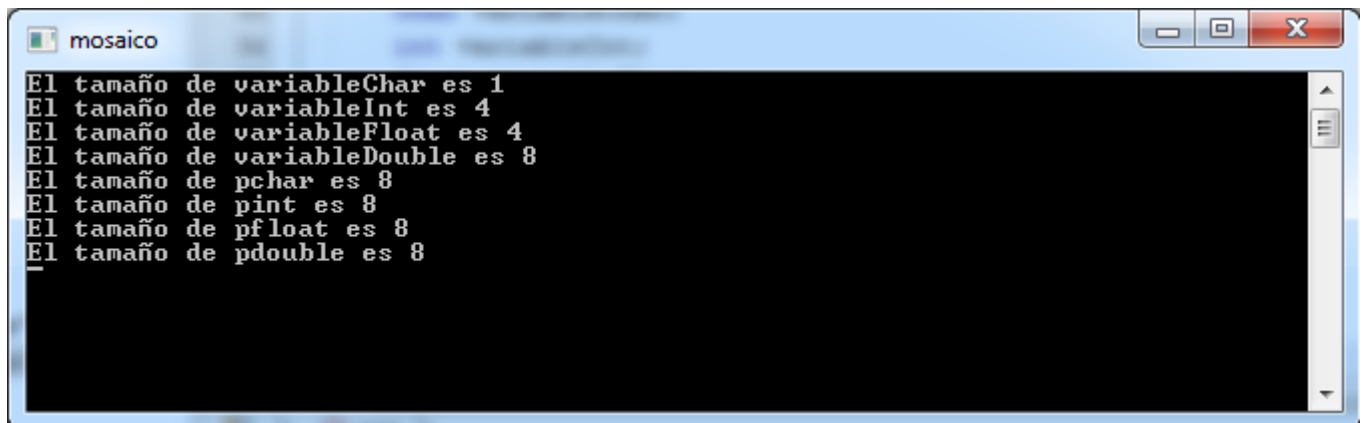
Un char ocupa en memoria 1 byte, un entero 4 bytes, a float 4 bytes, y un double 8 bytes. Un puntero a un char, a un entero, a un float, a un double, a cualquier tipo de dato, siempre tiene el mismo tamaño, 4 bytes en una máquina de 32 bits y 8 bytes en una máquina de 64 bits. ¿Por qué? Porque un puntero siempre almacena una dirección de memoria. A partir de ahora consideraremos que trabajamos siempre con una máquina de 64 bits. El siguiente código muestra las afirmaciones anteriores.

```
char variableChar;
int variableInt;
float variableFloat;
double variableDouble;

char * pchar;
int * pint;
float * pfloat;
double * pdouble;

printf("El tamaño de variableChar es %d\n", sizeof (variableChar));
printf("El tamaño de variableInt es %d\n", sizeof (variableInt));
printf("El tamaño de variableFloat es %d\n", sizeof (variableFloat));
printf("El tamaño de variableDouble es %d\n", sizeof (variableDouble));

printf("El tamaño de pchar es %d\n", sizeof (pchar));
printf("El tamaño de pint es %d\n", sizeof (pint));
printf("El tamaño de pfloat es %d\n", sizeof (pfloat));
printf("El tamaño de pdouble es %d\n", sizeof (pdouble));
```



```
mosaico
El tamaño de variableChar es 1
El tamaño de variableInt es 4
El tamaño de variableFloat es 4
El tamaño de variableDouble es 8
El tamaño de pchar es 8
El tamaño de pint es 8
El tamaño de pfloat es 8
El tamaño de pdouble es 8
```

Figura 10: Salida del código ejemplo tamaño variables y punteros.

Como su nombre indica, un puntero apunta a algún sitio, dado que un puntero contiene una dirección de memoria, entonces un puntero apunta a una dirección de memoria. Hay punteros a char, a int, a float y a double. Por lo tanto, en esa dirección de memoria apuntada deberá encontrarse almacenado un dato del mismo tipo. El dato apuntado por el puntero puede accederse con el operando indirección o \*. Las variables que se crean en un programa se almacenan en memoria, podemos saber la posición que ocupan con el operando &. El código siguiente lo explica, el resultado puede verse en la figura 11.

```
double * pdouble;
double variableDouble = 2.0;

printf("\npdouble es un puntero que contiene la direccion de memoria %d", pdouble
);
printf("\nvariableDouble es una variable que contiene el real %f", variableDouble
);
printf("\nvariableDouble esta en memoria en la direccion %d", variableDouble);

pdouble = &variableDouble;
```



```
printf("\n\npdouble apunta ahora a la direccion donde esta variableDouble: %d",
    pdouble);
printf("\n\nNo es lo mismo %f que %d", variableDouble, pdouble);
printf("\n\nEs lo mismo %d que %d", &variableDouble, pdouble);
printf("\n\nEs lo mismo %f que %f", variableDouble, *pdouble);
```

```
pdouble es un puntero que contiene la direccion de memoria 0
variableDouble es una variable que contiene el real 2.000000
variableDouble esta en memoria en la direccion 0

pdouble apunta ahora a la direccion donde esta variableDouble: 2280088
No es lo mismo 2.000000 que 2280088
Es lo mismo 2280088 que 2280088
Es lo mismo 2.000000 que 2.000000
```

Figura 11: Operadores \* y &.

Cuando se define un puntero, éste debe tener un valor inicial. Ese valor inicial normalmente es NULL o cero. Uno de los principales errores que se cometen cuando se trabaja con punteros, es intentar acceder a una dirección de memoria no asignada a nuestro programa. En las computadoras modernas, un programa únicamente puede acceder a direcciones de memoria asignadas a ese programa. Esto tiene sentido, un programa no puede acceder a la memoria con la que trabaja otro programa, pues modificaría su código o sus variables. El siguiente código da como resultado la figura 12. La variable pdouble es un puntero que se crea con valor inicial 0. Cuando se intenta mostrar en pantalla el dato double que está a partir de la dirección 0, el programa finaliza con el mensaje Segmentation fault. El programa ha intentado acceder a la dirección 0 de memoria, dirección que no le pertenece, y por lo tanto, el programa es abortado.

```
double * pdouble;

printf("La direccion de pdouble es %d\n", pdouble);
printf("El dato double apuntado es %d\n", *pdouble);
```

```
La direccion de pdouble es 0
/cygdrive/C/Program Files/NetBeans 8.0.2/ide/bin/nativeexecution/dorun.sh: line
33: 12508 Segmentation fault (core dumped) sh "$<SHFILE>"
Press [Enter] to close the terminal ...
```

Figura 12: Ejemplo de mal uso de punteros.

Si cambiamos hardware o compilador nos podemos encontrar que un tipo de datos tiene un tamaño distinto. El tipo int, dependiendo de máquina o compilador puede tener 16 o 32 bits. Por este motivo, es mejor utilizar los tipos de datos proporcionados por el estándar C99. inttypes.h normaliza un tamaño para todos los tipos de datos. Para los enteros y los enteros sin signo tenemos:

- int8\_t: entero de 8 bits
- int16\_t: entero de 16 bits
- int32\_t: entero de 32 bits
- int64\_t: entero de 64 bits

- uint8\_t: entero sin signo de 8 bits
- uint16\_t: entero sin signo de 16 bits
- uint32\_t: entero sin signo de 32 bits
- uint64\_t: entero sin signo de 64 bits

## 6.2. Acceso a los campos de un puntero. El operador ->

Un puntero puede apuntar a una estructura y acceder a sus campos:

```
struct Dato
{
    int campo1, campo2;
    char campo3 [30];
};
struct Dato x;
struct Dato *ptr;
...
ptr = &x;
(*ptr).campo1 = 33;
strcpy ( (*ptr).campo3, "hola" );
```

Para hacer más cómodo el trabajo con punteros a estructuras, C dispone del operador ->, que se utiliza de esta forma:

```
ptr->campo
```

que es equivalente a escribir

```
(*ptr).campo
```

Así, el ejemplo anterior quedaría de esta forma:

```
ptr = &x;
ptr->campo1 = 33;
strcpy ( ptr->campo3, "hola" );
```

## 6.3. ¿Dónde está mi ejecutable?

Después de compilar un programa en C y si no hay errores, se genera un fichero binario ejecutable. En Windows ese fichero va a tener la extensión exe. Normalmente Windows oculta las extensiones de los ficheros, pero podemos verlas viendo sus propiedades tal como muestra la figura 13 . Muchas veces no nos damos cuenta de su existencia, porque el entorno de programación o IDE nos da la posibilidad de ejecutar el programa generado. Realmente lo que hace el IDE por nosotros es llamar a ese fichero exe generado.

Resulta evidente que si un cliente nos pide un programa, debemos darle la posibilidad de ejecutarlo sin necesidad de un IDE, entregándole dicho fichero exe. En Netbeans para Windows el fichero exe lo podemos encontrar en directorio dist de nuestro proyecto.

## 6.4. Parámetros por línea de comandos

Para poder pasar parámetros a un programa a través de la línea de comandos nos valemos de la siguiente declaración de la función main():

```
int main(int argc, char *argv[])
```

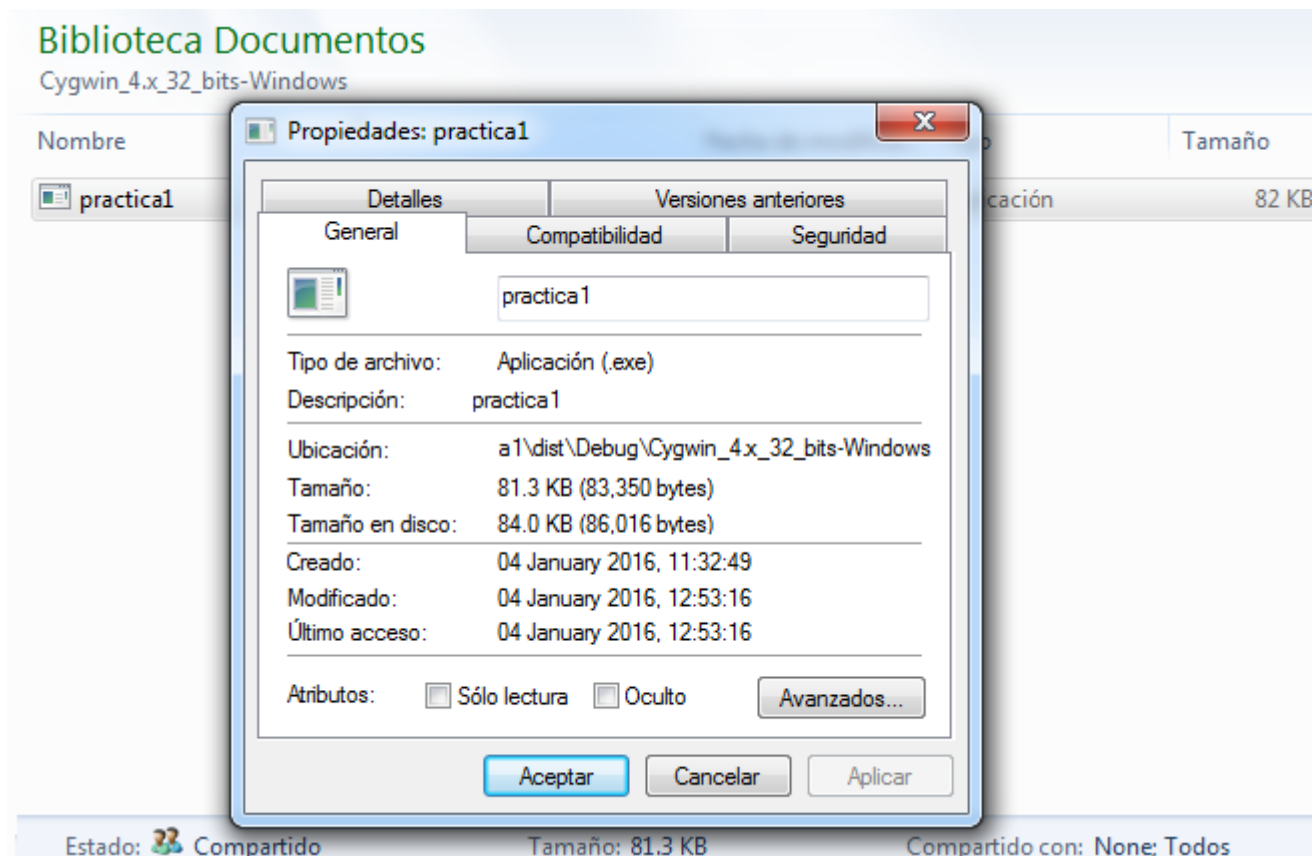


Figura 13: Propiedades del fichero ejecutable generado.

El primer argumento entero `argc`, contiene el número de argumentos recibidos por el programa, debemos considerar que siempre será el número de argumentos pasados más uno, ya que el primer argumento se reserva para contener el nombre del programa. El segundo es un apuntador a un array de chars que contiene los parámetros pasados en el mismo orden en que fueron escritos. Supongamos que llamemos a un programa de la siguiente manera:

```
./mcd entero1 entero2
```

`argc` contendrá el valor 3, debido al nombre del programa, y los dos argumentos pasados. `argv[0]` contendrá el valor `mcd`, `arg[1]` será `entero1`, `arg[2]` será `entero2`.

## 6.5. EXIT\_SUCCESS y EXIT\_FAILURE

La función `main()` devuelve un entero. Si la ejecución del programa se realizó con éxito `main()` devuelve el valor `EXIT_SUCCESS`. Si la ejecución no se realizó con éxito el `main()` devuelve el valor `EXIT_FAILURE`. Estos valores pueden ser usados por un script que llame al programa considerado para saber si el programa se ejecutó correctamente o no, y ser empleado en una sentencia condicional. En la figura 14 se puede ver como finaliza el programa ejemplo desde NetBeans, con éxito y además sin éxito cuando por ejemplo no es capaz de localizar el fichero especificado en los parámetros de línea de comandos.



Figura 14: Ejecución con y sin éxito de un programa

Será frecuente que al trabajar con punteros se produzcan errores es su manejo y por lo tanto, el programa intente

acceder a alguna dirección de memoria no asignada y el sistema operativo finalice anticipadamente su ejecución. En este caso, el programa devolverá el valor `EXIT_FAILURE`.

## 7. Ejemplos

### 7.1. Leer y mostrar una imagen desde un fichero

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #include <opencv/cv.h>
5  #include <opencv/highgui.h>
6
7  int main(int argc, char** argv) {
8
9      if (argc != 2) {
10         printf("Usage: %s image_file_name\n", argv[0]);
11         return EXIT_FAILURE;
12     }
13
14     //CV_LOAD_IMAGE_COLOR = 1 forces the resultant IplImage to be colour.
15     //CV_LOAD_IMAGE_GRAYSCALE = 0 forces a greyscale IplImage.
16     //CV_LOAD_IMAGE_UNCHANGED = -1
17     IplImage* Img1 = cvLoadImage(argv[1], CV_LOAD_IMAGE_UNCHANGED);
18
19     // Always check if the program can find a file
20     if (!Img1) {
21         printf("Error: fichero %s no leido\n", argv[1]);
22         return EXIT_FAILURE;
23     }
24
25     // a visualization window is created with title 'image'
26     cvNamedWindow(argv[1], CV_WINDOW_NORMAL);
27     // img is shown in 'image' window
28     cvShowImage(argv[1], Img1);
29     cvWaitKey(0);
30
31
32     // memory release for img before exiting the application
33     cvReleaseImage(&Img1);
34
35     // Self-explanatory
36     cvDestroyWindow(argv[1]);
37
38     return EXIT_SUCCESS;
39 }
40
```

### 7.2. Crear una imagen sólo con componente azul

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #include <opencv/cv.h>
5  #include <opencv/highgui.h>
6
7  int main(int argc, char **argv) {
8
9      if (argc != 2) {
```

```

10     printf("Error: Usage %s image_file_name\n", argv[0]);
11     return EXIT_FAILURE;
12 }
13
14 //CV_LOAD_IMAGE_COLOR = 1 forces the resultant IplImage to be colour.
15 //CV_LOAD_IMAGE_GRAYSCALE = 0 forces a greyscale IplImage.
16 //CV_LOAD_IMAGE_UNCHANGED = -1
17 IplImage* Img = cvLoadImage(argv[1], CV_LOAD_IMAGE_UNCHANGED);
18
19 // Always check if the program can find the image file
20 if (!Img) {
21     printf("Error: file %s not found\n", argv[1]);
22     return EXIT_FAILURE;
23 }
24
25 // a visualization window is created with title: image file name
26 cvNamedWindow(argv[1], CV_WINDOW_NORMAL);
27 // Img is shown in 'image' window
28 cvShowImage(argv[1], Img);
29 cvWaitKey(0);
30
31
32 // Esto en la version 3.0.0 ha dejado de funcionar
33 // cambiar cvGetSize por cvSize
34 //IplImage* ImgRED = cvCreateImage(cvGetSize(Img), IPL_DEPTH_8U, 3);
35 //IplImage* ImgGREEN = cvCreateImage(cvGetSize(Img), IPL_DEPTH_8U, 3);
36 //IplImage* ImgBLUE = cvCreateImage(cvGetSize(Img), IPL_DEPTH_8U, 3);
37
38 // Crea las imagenes para las tres componentes
39 IplImage* ImgRED = cvCreateImage(cvSize(Img->width, Img->height), IPL_DEPTH_8U,
40     3);
41 IplImage* ImgGREEN = cvCreateImage(cvSize(Img->width, Img->height), IPL_DEPTH_8U,
42     3);
43 IplImage* ImgBLUE = cvCreateImage(cvSize(Img->width, Img->height), IPL_DEPTH_8U,
44     3);
45
46 unsigned char *pImg;
47 unsigned char *pImgRED;
48 unsigned char *pImgGREEN;
49 unsigned char *pImgBLUE;
50
51 int fila, columna;
52
53 for (fila = 0; fila < Img->height; fila++) {
54
55     pImg = (unsigned char *) Img->imageData + fila * Img->widthStep;
56     pImgRED = (unsigned char *) ImgRED->imageData + fila * ImgRED->widthStep;
57     pImgGREEN = (unsigned char *) ImgGREEN->imageData + fila * ImgGREEN->
58         widthStep;
59     pImgBLUE = (unsigned char *) ImgBLUE->imageData + fila * ImgBLUE->widthStep;
60
61     for (columna = 0; columna < Img->width; columna++) {
62         //Imagen BLUE
63         *pImgBLUE++ = *pImg++;
64         *pImgBLUE++ = 0;
65         *pImgBLUE++ = 0;
66
67         //Imagen GREEN
68         *pImgGREEN++ = 0;
69         *pImgGREEN++ = *pImg++;
70         *pImgGREEN++ = 0;
71
72         //Imagen RED

```

```

69         *pImgRED++ = 0;
70         *pImgRED++ = 0;
71         *pImgRED++ = *pImg++;
72
73     }
74 }
75
76 // crea y muestras las ventanas con las imagenes
77 cvNamedWindow("Componente RED", CV_WINDOW_AUTOSIZE);
78 cvShowImage("Componente RED", ImgRED);
79 cvNamedWindow("Componente GREEN", CV_WINDOW_AUTOSIZE);
80 cvShowImage("Componente GREEN", ImgGREEN);
81 cvNamedWindow("Componente BLUE", CV_WINDOW_AUTOSIZE);
82 cvShowImage("Componente BLUE", ImgBLUE);
83
84 cvWaitKey(0);
85
86 // memory release for images before exiting the application
87 cvReleaseImage(&Img);
88 cvReleaseImage(&ImgRED);
89 cvReleaseImage(&ImgGREEN);
90 cvReleaseImage(&ImgBLUE);
91
92
93 // Self-explanatory
94 cvDestroyWindow(argv[1]);
95 cvDestroyWindow("Componente RED");
96 cvDestroyWindow("Componente GREEN");
97 cvDestroyWindow("Componente BLUE");
98
99
100 return EXIT_SUCCESS;
101 }

```

### 7.3. Crear una imagen por componente

A partir de una imagen RGB se pueden crear tres imágenes con cada una de las componentes roja, verde y azul. La figura 15 indica como asignar las componentes de la imagen RGB a cada una de las tres imágenes.

Imagen RGB	Imagen B	Image G	Imagen R
Blue Pixel 1	Blue Pixel 1	0	0
Green Pixel 1	0	Green Pixel 1	0
Red Pixel 1	0	0	Red Pixel 1
Blue Pixel 2	Blue Pixel 2	0	0
Green Pixel 2	0	Green Pixel 2	0
Red Pixel 2	0	0	Red Pixel 2
...	...	...	...
...	...	...	...
...	...	...	...
Blue Pixel FxC	Blue Pixel FxC	0	0
Green Pixel FxC	0	Green Pixel FxC	0
Red Pixel FxC	0	0	Red Pixel FxC

Figura 15: Descomposición de una imagen en sus componentes RGB



## 8. Software

Si tienes un equipo de 64 bits, se recomienda instalar:

- NetBeans IDE 8.x con soporte para C/C++, <https://netbeans.org/downloads/start.html?platform=windows&lang=en&option=cpp&bits=x64>.
- El compilador Cygwin C/C++ para 64 bits, [http://cygwin.com/setup-x86\\_64.exe](http://cygwin.com/setup-x86_64.exe).
- Instalar la librería de tratamiento de imágenes OpenCV, [http://sourceforge.net/projects/opencvlibrary/files/latest/download?source=top3\\_dlp\\_t5](http://sourceforge.net/projects/opencvlibrary/files/latest/download?source=top3_dlp_t5).

Será necesario configurar el proyecto para que Netbeans use el compilador de 64 bits tal como indica la figura 16.

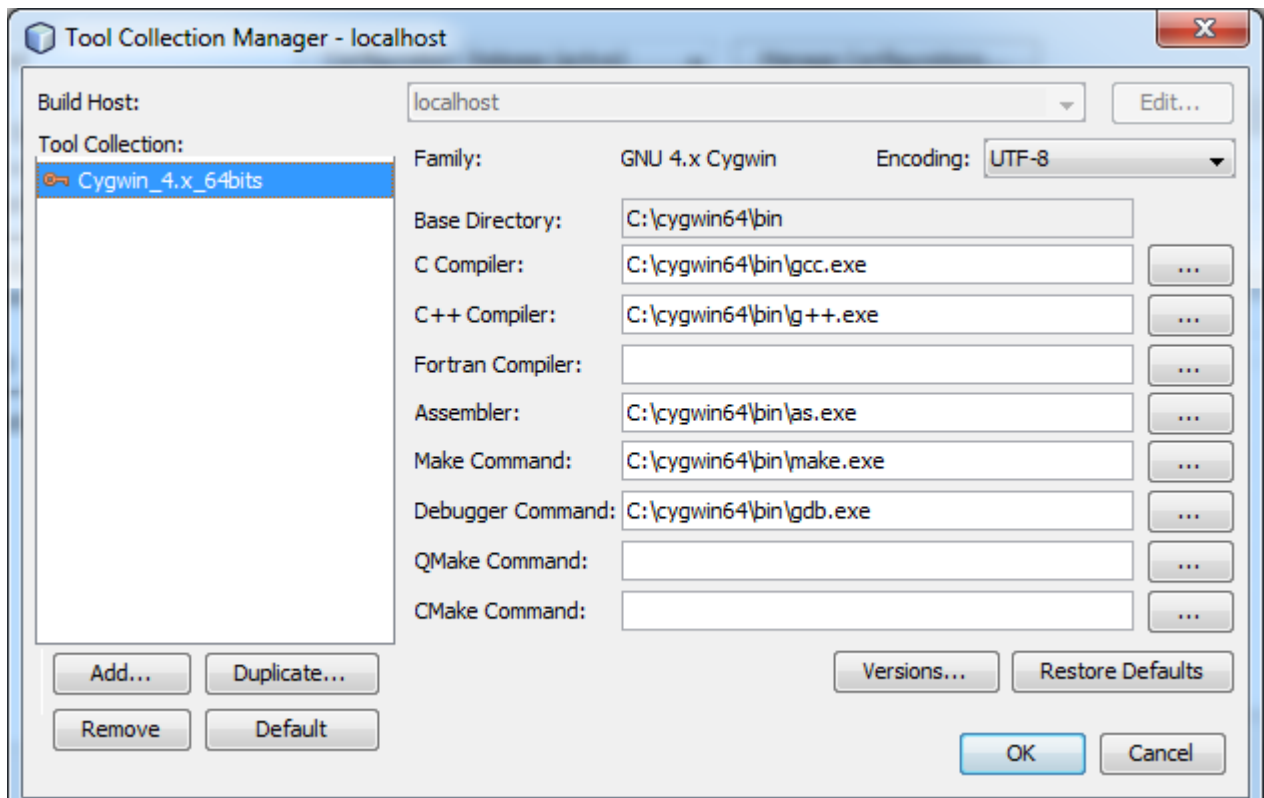


Figura 16: Proyecto para 64 bits.

Si tienes un equipo de 32 bits, se recomienda instalar:

- NetBeans IDE 8.x con soporte para C/C++, <https://netbeans.org/downloads/start.html?platform=windows&lang=en&option=cpp&bits=x86>.
- El compilador Cygwin C/C++ para 32 bits, <http://cygwin.com/setup-x86.exe>.
- Instalar la librería de tratamiento de imágenes OpenCV, [http://sourceforge.net/projects/opencvlibrary/files/latest/download?source=top3\\_dlp\\_t5](http://sourceforge.net/projects/opencvlibrary/files/latest/download?source=top3_dlp_t5).

Será necesario configurar el proyecto para que Netbeans use el compilador de 32 bits tal como indica la figura 17.

## 9. Requisitos

- Flujograma de cómo resolver el ejercicio planteado.
- Programas en C, no en C++.
- Uso de punteros, no arrays.

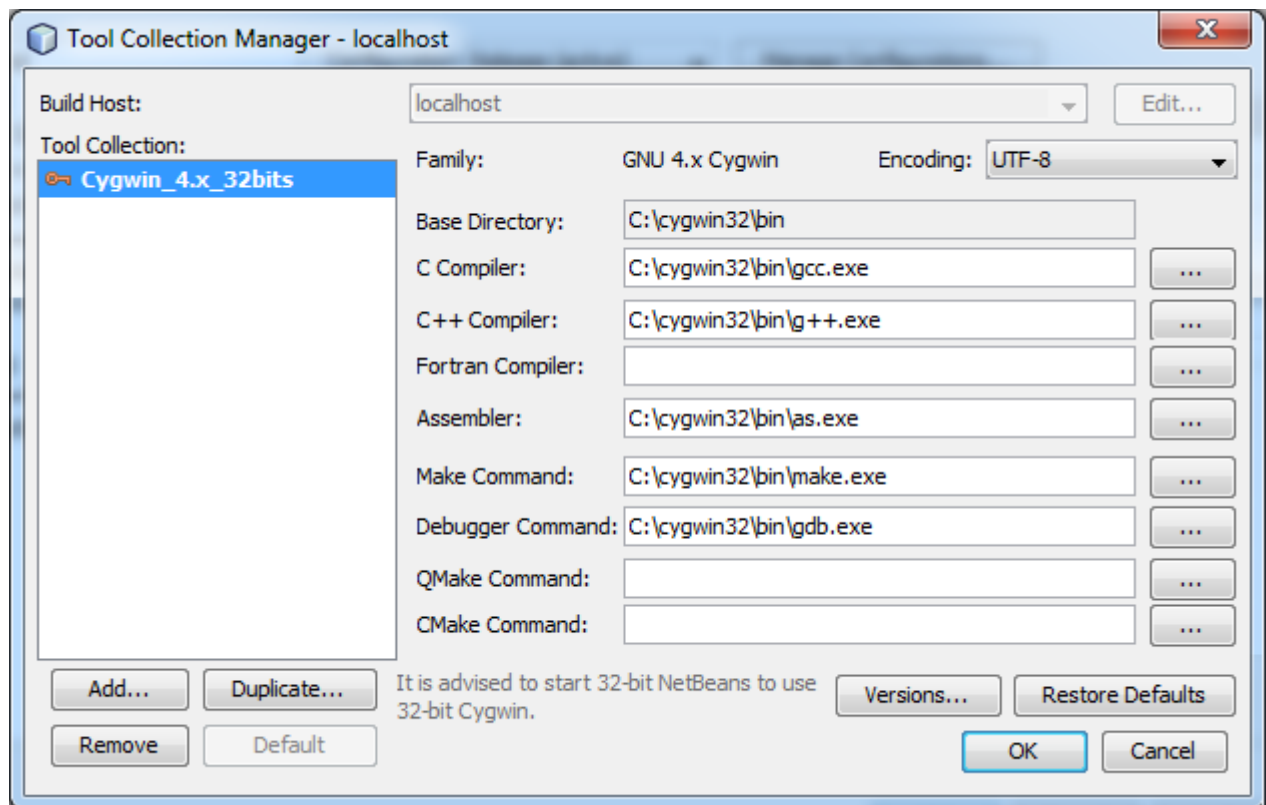


Figura 17: Proyecto para 32 bits.

- Únicamente uso de las funciones openCV del apartado 5, ninguna otra.
- Cuando el programa tiene unas cuantas líneas y empieza a no estar claramente indentado, podéis usar la siguiente opción de Netbeans: Menú-Source-Format. El código quedará mucho más legible.

## 10. Fallos frecuentes

- Normalmente, la extensión de un fichero con una imagen comprimida con JPEG es .jpg. Por ejemplo, fruits.jpg.
- Dependiendo de las opciones de visualización, Windows puede ocultar las extensiones de los fichero. A veces suele suceder que al no ver las extensiones, por equivocación éstas se ponen dos veces (fruits.jpg.jpg, evidentemente, esto está mal).