ASD: ARQUITECTURA DE SISTEMAS DISTRIBUIDOS. 3º GTI.
PRÁCTICA 1: PRESTACIONES EN FILTROS DE IMÁGENES.

1. OBJETIVOS Y PREPARACIÓN.

En esta práctica vamos a demostrar cómo se puede mejorar el rendimiento de un programa, o bien ayudando al compilador, o bien escribiendo el código de alto nivel de la forma más adecuada para que el computador aproveche mejor el paralelismo.

Para ello, se usará una forma sencilla de medir tiempos de ejecución y se calcularán aceleraciones para varias versiones de un mismo programa, usando el IDE VS2010 (Visual Studio 2010 de Microsoft).

En concreto, se va a estudiar la optimización de filtros de tratamiento de imágenes, atacando a la parte del código donde está la mayor parte del tiempo de ejecución (bucle central de bucles anidados). El filtro que se probará realiza un difuminado de la imagen, que sirve para eliminar "ruido" (es decir, suprimir o reducir los píxeles con valores muy diferentes de sus píxeles vecinos) y para otras aplicaciones.

Se compararán los tiempos de ejecución del filtro de la librería OpenCV con 2 versiones escritas en lenguaje C++.

Antes de acudir al laboratorio el alumno deberá:

- Repasar los conceptos básicos del tema 1. Los que se van a pedir expresamente en esta práctica son: aceleración, fórmula del tiempo de ejecución, CPI.
- Entender **cómo están organizados los esquemas de código que se proporcionan** con este enunciado. Ver
  - 4. APÉNDICE: ESQUELETO DEL CÓDIGO DE PRUEBA
  - 5. APÉNDICE: FUNCIÓN SIMPLE PARA FILTRO DE IMÁGENES

  Notar que `img->height` es la altura en píxeles de la imagen e `img->width` es la anchura en píxeles de la imagen
- Dibujar en un esquema de los píxeles de una imagen RGB (un byte por color; 3 bytes por píxel: Red, Green, Blue) y del algoritmo que hace la función `asd_blurring_simplest()`. Para ello se debe saber que los datos de una imagen RGB se organizan como un vector, cuyos elementos contienen 3 bytes, de esta forma (siendo un byte cada celda de este diagrama, y Nf: número de filas, Nc = número de columnas):

R	G	B	R	G	B	…	R	G	B
`Pixel (0,0)`			`Pixel (0,1)`			…	`Pixel (Nf-1,Nc-1)`		

2. REALIZACIÓN DE LA PRÁCTICA.

El profesor explicará:

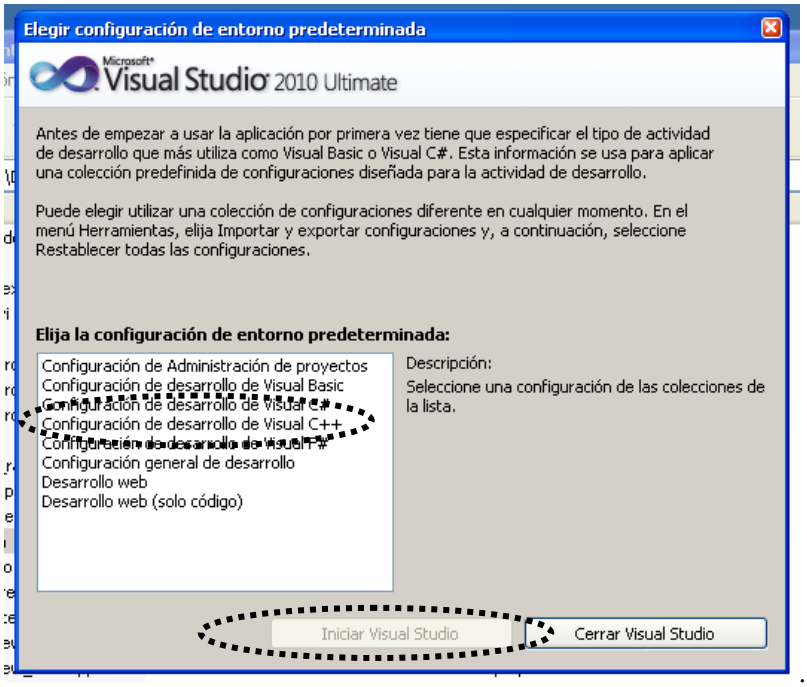
- *Como medir tiempo usando la clase QPTimer que se da (Ver* APÉNDICE 1: LA CLASE QPTIMER *).*
- *Este comentario del filtro // Remark: this loop has been chosen to avoid boundary conditions be checked. // in fact two rows and two columns are not processed (this is a little difference: less than 0.3%)*
- *Como cambiar las opciones de optimización de la compilación.*

En el aula se dará un proyecto de VS2010 con todas las configuraciones ya preparadas (entre ellas, una serie de rutas configuradas).

NOTA: Si se quiere trabajar en otro PC (p ej en casa) o se quiere empezar un proyecto nuevo de VS2010, hay que realizar una serie de pasos de configuración (ver carpeta "CONFIGURATION OF WINDOWS AND VS2010 FOR OPENCV" en ev.us.es dentro de esta práctica).

Descargar fichero comprimido de ev.us.es. Descomprimirlo sobre una carpeta de C: (NUNCA trabajar sobre el comprimido).

A continuación abrir la "solución" A*SD_P1.sln* haciendo doble click (una solución es un conjunto de proyectos de VS2010). Si VS2010 abriera una ventana sobre elección de configuración del lenguaje, seleccionar "configuración de desarrollo de Visual C++" y luego "Iniciar Visual Studio" (tardará más de un minuto en configurar herramientas de C++), según se ve en la siguiente figura:



Vamos a comparar la duración de dos rutinas que hacen lo mismo con el vídeo, pero escritas de forma diferente:

- cvSmooth(), que pertenece a OpenCV
- asd_blurring_simplest().

2.1. Código más "simple"

Empecemos comparando la rutina de la librería OpenCV *cvSmooth()*, con la "normal" o simple que cualquier programador de alto nivel escribiría, es decir, *asd_blurring_simplest()*.

Ver el código C++ para asegurarse de que los dos cronómetros (*crono_my_own_blurring y crono_opencv_lib*) están midiendo estas rutinas.

Ejecutar y anotar tiempos de ambas para estas dos opciones de optimización de la compilación: /Od, /O2

Recordar: para cambiar estas opciones de compilación, lo más fácil es hacer click con el botón derecho del ratón sobre el nombre del Proyecto (en el explorador de las soluciones), y en el árbol de navegación de la siguiente figura, elegir primero: /Od, es decir, toda optimización está deshabilitada. Generar solución y ejecutar sin iniciar la depuración (Crtl – F5)

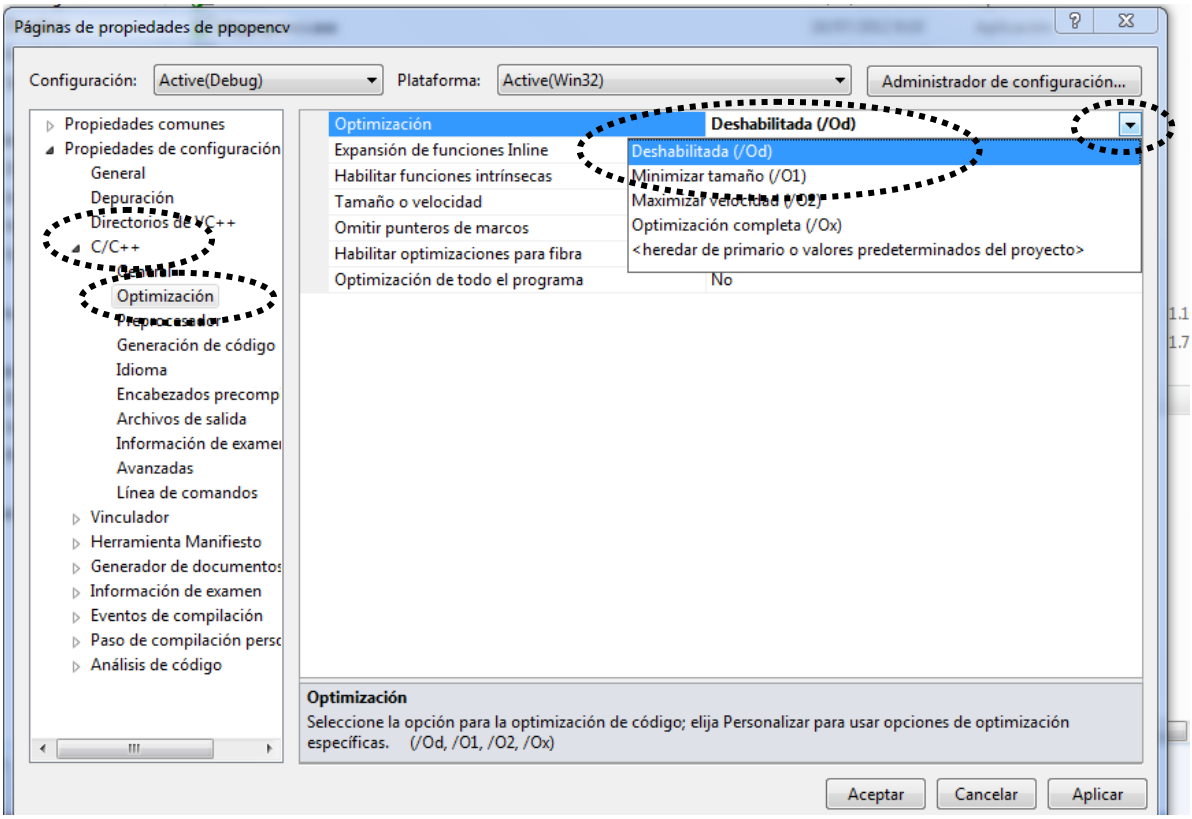
Después, para la otra medición, elegir /O2 , para mayor velocidad de ejecución. Volver a Generar la solución y ejecutar sin iniciar la depuración (Crtl – F5)

A continuación anotar la aceleración y los frames por segundo (fps) que se podrían procesar entre ambas rutinas para ambas opciones.

NOTA: La ejecución será más rápida, si no se define esta constante (se ha preparado el código con compilación condicional, para que así no se muestren los vídeos). Para ello, comentar esta línea:

```
#define __VIDEO_ON
```

Contestar: ¿por qué sale igual tiempo de cvSmooth() tanto para /Od como para /O2?
¿Cuál de los tiempos es más correcto para estas mediciones: el mínimo o la media (Minimum time or Mean time)?
¿Por qué?



2.2. Cálculo del CPI

A continuación el profesor explicará:

- *Como despejar CPI de fórmula del tiempo de ejecución.*
- *Usando el cañón: como arrancar el debugger, pararse en bucle central y ver desensamblado (ver figura siguiente).*

Aquí vamos a trabajar con *asd_blurring_simplest ()*, y calcular su CPI.

Primero, para hallar el número de instrucciones de la rutina, se arranca el depurador, se cuentan las instrucciones del bucle central del ensamblador. NOTA: se puede copiar texto de la ventana desensamblado y pegarlo en un editor para contar líneas (cada línea es una instrucción).

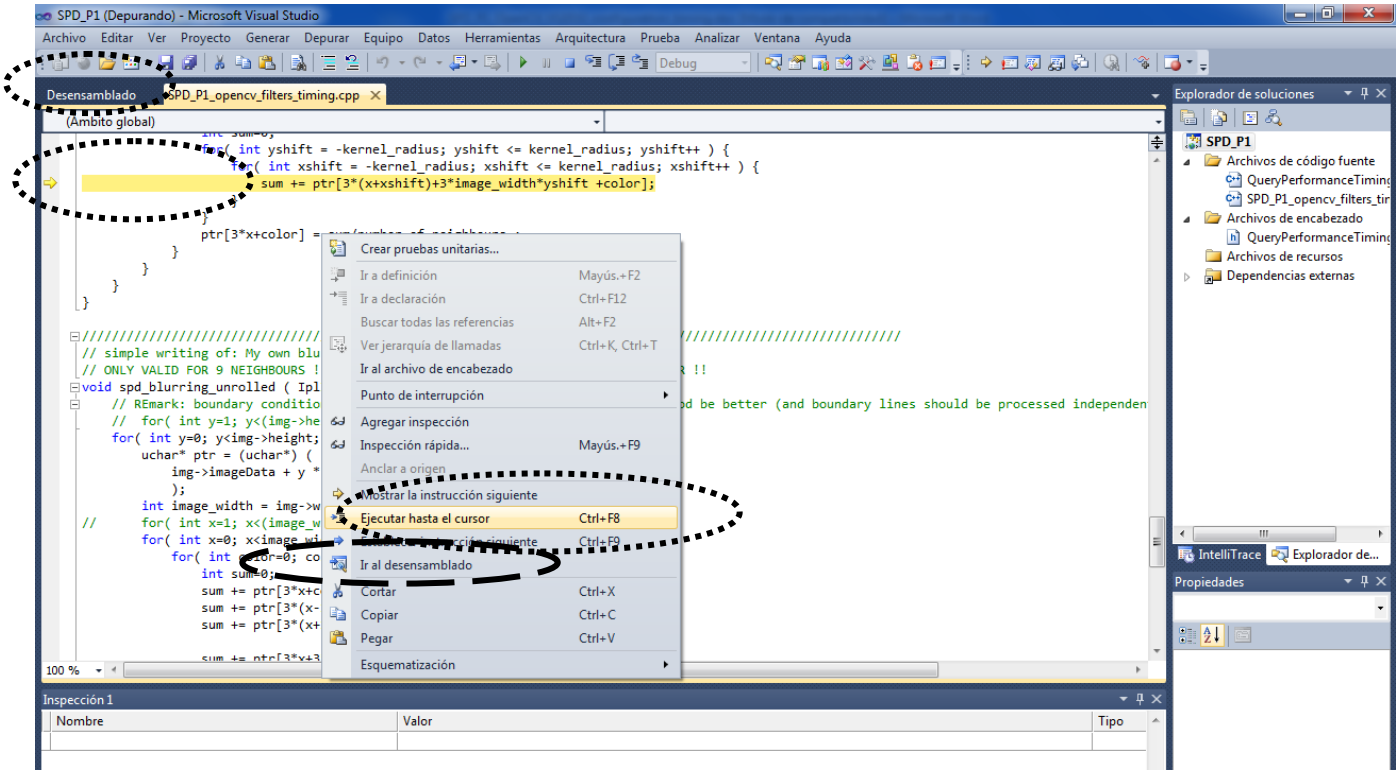
Finalmente se multiplican por el número de iteraciones de los otros bucles externos.

Hallar también el CPI

Hacer lo anterior para la opción /Od y la /O2.

Contestar a:

- ¿Qué error se comete aproximadamente al medir sólo las instrucciones del bucle central del ensamblador, cuando se usa /O2?
- Atendiendo a los tres parámetros T, Ninstr, CPI, ¿de dónde viene la principal diferencia del tiempo de ejecución entre la ejecución con la opción /Od y con la /O2.


SPD_P1 (Depurando) - Microsoft Visual Studio
Archivo Editar Ver Proyecto Generar Depurar Equipo Datos Herramientas Arquitectura Prueba Analizar Ventana Ayuda
Debug
Desensamblado
SPD_P1_opencv_filters_timing.cpp
(Ámbito global)
sum += ptr[3*(x+xshift)+3*image_width*yshift +color];
Crear pruebas unitarias...
Ir a definición Mayús.+F2
Ir a declaración Ctrl+F12
Buscar todas las referencias Alt+F2
Ver jerarquía de llamadas Ctrl+K, Ctrl+T
Ir al archivo de encabezado
Punto de interrupción
Agregar inspección
Inspección rápida... Mayús.+F9
Anclar a origen
Mostrar la instrucción siguiente
Ejecutar hasta el cursor Ctrl+F8
Ir al desensamblado
Cortar Ctrl+X
Copiar Ctrl+C
Pegar Ctrl+V
Esquematización
Explorador de soluciones
SPD_P1
Archivos de código fuente
Archivos de encabezado
Archivos de recursos
Dependencias externas
IntelliTrace
Explorador de...
Propiedades
Inspección 1
Nombre
Valor
Tipo

3. APÉNDICE: ESQUELETO DEL CÓDIGO DE PRUEBA

```
// project def and includes
…
//-----------------------------------------------------
// My own blurring function
void asd_blurring_simplest ( IplImage* img , int kernel_radius);
…
int main( int argc, char** argv ) {
	char *file_name="megamind.avi";
	// video to be processed

// OpenCV function to create windows:
#ifdef __VIDEO_ON
	cvNamedWindow( "ASD_P1-in", CV_WINDOW_AUTOSIZE );
	cvNamedWindow( "ASD_P1-out-OPENCV", CV_WINDOW_AUTOSIZE );
	cvNamedWindow( "ASD_P1-out-MY-OWN-BLURRING", CV_WINDOW_AUTOSIZE );
#endif
…
// structures to point to a frame of a video, and capturing the first frame
IplImage* newframe;
IplImage* oldframe;
newframe = cvQueryFrame( g_capture );
…
// main loop where frames are processed
while(newframe … ) {

	oldframe=newframe;

	// code to start the timer is introduced here to measure filter times

	// Do the smoothing using OpenCV
	cvSmooth( oldframe, out, CV_BLUR , 3, 3 );
	// code to stop the timer is introduced here to measure filter times

	// code to start the timer is introduced here to measure filter times
…
	//UNCOMMENT THIS FUNCTION (and coment the other) WHEN NECESSARY :
	asd_blurring_simplest ( oldframe , 1 ); //this function modify image (parameter)
	// code to stop the timer is introduced here to measure filter times
…
	// code to Show the original and the smoothed image in output windows is by here

	// a new frame is captured to iterate the loop
	newframe = cvQueryFrame( g_capture );
	}  // end of:   while(newframe  … ) {

// print the times of the different filters

////////////////////////////////////////////////////////////////////////////////////////
////////////////////
// My own blurring function (the "simplest")
void asd_blurring_simplest ( IplImage* img , int kernel_radius) {

…
// see Appendix 4. APÉNDICE: FUNCIÓN SIMPLE PARA FILTRO DE IMÁGENES
}
```

4. APÉNDICE: FUNCIÓN SIMPLE PARA FILTRO DE IMÁGENES

```
//////////////////////////////////////////////////////////////////
// My own blurring function (the "simplest")
void asd_blurring_simplest ( IplImage* img , int kernel_radius) {
  int number_of_neighbours = (1+2*kernel_radius)*(1+2*kernel_radius);

  // Remark: this loop has been chosen to avoid boundary conditions be checked.
  // in fact 2 rows and 2 columns are not processed (a little difference: less than 0.3%)
  for( int y=1; y<(img->height)-1; y++ ) {

    //THIS POINTER ptr (to unsigned char) will point to the
    // first byte of a row in a RGB image
    // Do not worry about the fields of this structure; they are given by OpenCV
    uchar* ptr = (uchar*) (
      img->imageData + y * img->widthStep
      );

    int image_width = img->width;
    // Remark: this loop has been chosen to avoid boundary conditions be checked.
    // in fact 2 rows and 2 columns are not processed (a little difference: less than 0.3%)
    for( int x=1; x<(image_width -1); x++ ) {
      for( int color=0; color<3; color++ ) {
        int sum=0;
        for( int yshift = -kernel_radius; yshift <= kernel_radius; yshift++ ) {
          for( int xshift = -kernel_radius; xshift <= kernel_radius; xshift++ ) {
            sum += ptr[3*(x+xshift)+3*image_width*yshift +color];
          }
        }
        ptr[3*x+color] = sum/number_of_neighbours ;
      }
    }
  }
}
```

5. APÉNDICE: LA CLASE QPTIMER

```
QPTimer crono;  //creates an object QPTimer

crono.Calibrate(); //calibrates timer overhead and set cronometer to zero

// repeat several times the test for more timing precision.
for ( int times=0; times < 10; times++)
{
       crono.Start();  // start timer

       // Do Something
       function_to_be_measured ();

       crono.Stop();  // stop timer and increment the number of measures

       crono.Reset();   // set timer to zero
}

// print timing results (for the previous the number
       //   of measures; crono.ResetAll() would have reset this number)
crono.PrintMinimumTime   (" Minimum time in seconds is:   ");
crono.PrintMeanTime      (" Mean time in seconds is   :   ");
crono.PrintMinimumCycles (" Minimum time in cycles is :   ");
crono.PrintMeanCycles    (" Mean time in cycles is    :   ");
```

TABLA DE RESULTADOS
ASD: ARQUITECTURA DE SISTEMAS DISTRIBUIDOS. 3º GTI.
PRÁCTICA 1. PRESTACIONES DE FILTROS DE IMÁGENES.
NUM GRUPO:
ALUMNOS:

OBJETIVOS Y PREPARACIÓN.

- Dibujar un diagrama de cómo se organizan los datos de una imagen RGB.

Dibujar en un esquema de los píxeles de una imagen RGB (3 bytes por píxel: Red, Green, Blue) y del algoritmo que hace la función simplest ()

2.1 Código más "simple"

	CvSmooth	asd_blurring_simplest
Tiempo con /Od		
Aceleración de asd_blurring_simplest respecto de CvSmooth con /Od		
Tiempo con /O2		
Aceleración de asd_blurring_simplest respecto de CvSmooth con /O2		

Contestar ¿por qué sale igual tiempo de cvSmooth() tanto para /Od como para /O2?

¿Cuál de los tiempos es más correcto para estas mediciones: el mínimo o la media (Minimum time or Mean time)?¿Por qué?

2.2 Cálculo del CPI

	asd_blurring_simplest con /Od	asd_blurring_simplest con /O2
Número instrucciones por iteración de .asm		
Número instrucciones por píxel		
Número instrucciones por iteración aprox. del programa		
CPI aprox.		

Contestar: ¿Qué error se comete aproximadamente al medir sólo las instrucciones del bucle central del ensamblador cuando se usa /O2?

Atendiendo a los tres parámetros T, Ninstr, CPI, ¿de dónde viene la principal diferencia del tiempo de ejecución entre la ejecución con la opción /Od y con la /O2.