UNIVERSIDAD POLITECNICA DE VALENCIA

OpenCV + OpenGL: creando una superficie 3D a partir de una imagen 2D

Apellidos, nombre	Agustí Melchor, Manuel (magusti@disca.upv.es)
Departamento	Dpto. De Ing. De Sistemas y Computadores
Centro	Universidad Politécnica de Valencia

UNIVERSIDAD POLITECNICA DE VALENCIA

1 Resumen de las ideas clave

En este artículo vamos a presentar como el computador puede representar una imagen (bidimensional) leída de fichero en una ventana en pantalla y, a partir de esa información, recrear una versión tridimensional de la misma. La fig. 1 muestra un ejemplo de la salida que se pretende obtener.

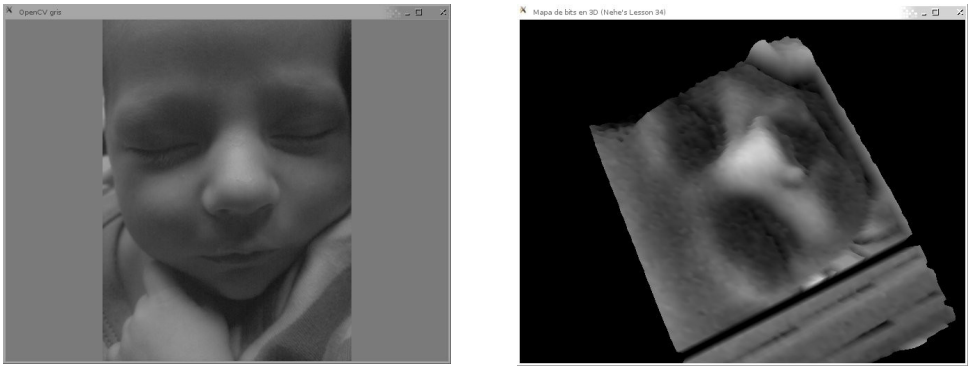

Figura 1: Ejemplo de imagen representada en 2D (izquierda) y 3D (derecha).

La utilidad de esta conversión es básicamente ofrecer una visualización diferente, puesto que la imagen plana (en general, sin otra información de apoyo como conocer los parámetros de la cámara que la ha tomado) no tiene posibilidad de recuperar la información real de la distancia a los puntos detectar o permitir una calibración a posteriori. Así pues, teniendo en cuenta que algunas consideraciones son arbitrarias vamos a jugar un poco y adentrarnos en el mundo del 3D.

Utilizaremos imágenes desde fichero mediante el uso de OpenCV [1, 2] y OpenGL [3] para generar la visualización en 3D. Para ello tomaremos como partida OpenCV versión 1.1 y OpenGL 3.2[1] y GLX 1.4. Para su desarrollo nos centraremos en la plataforma GNU/Linux, aunque todo lo expuesto es transportable a otras en que se hayan instalado estas librerías.

2 Introducción

Este artículo mostrará cómo realizar una versión tridimensional a partir de una imagen plana, tomando como asunción que la luminosidad de la imagen representa lo “alto” que está un punto, pensando que los puntos más oscuros están más abajo (les llega menos luz) y que los más altos presentan valores más claros (brillantes). En el campo de los gráficos por computador este proceso se suele denominar *terrain* o *height map*.

Este mapa de superficie lo hemos tomado del tutorial de cómo levantar una superficie del sitio web de *i* [4] a partir de una textura. Este ejemplo nos ofrece una rejilla tridimensional que permite asignar a cada vértice un valor de altura a partir del contenido de un fichero. Este fichero tiene formato RAW (en crudo y binario), esto es, sin formato o, mejor dicho, sin cabecera que lo explique. Este fichero es una imagen de un paisaje y se utiliza para facilitar la creación de

[1]Información obtenida con la orden *glxinfo* | *grep version*.

superficies sin necesidad de especificar el modelo por vértices y la textura para cada polígono, con la consiguiente flexibilidad y ahorro de recursos.

La aplicación ofrece un sencillo interfaz: la posibilidad de acercarse o alejarse con las flechas del cursor (arriba y abajo) y de intercambiar el modo de visualización entre sombreado y modelo de alambres (con la tecla barra espaciadora). Un detalle de la imagen del ejemplo de partida se puede ver ampliada en la fig, 2 en los dos modos.

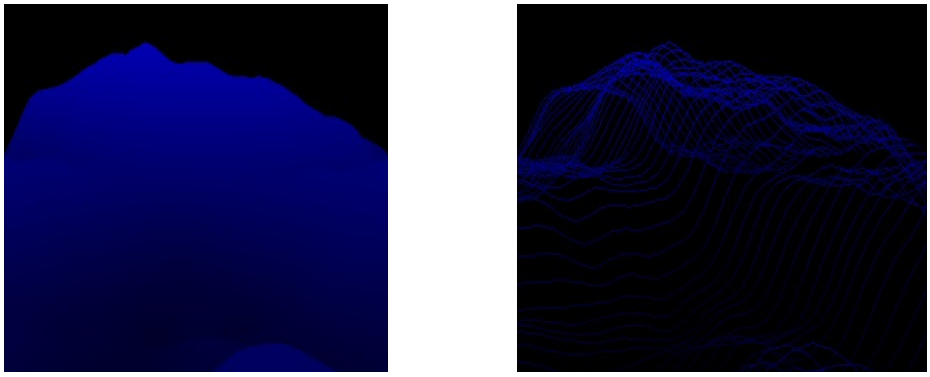

Figura 2: Detalle del modelo de sombreado (izquierda) y del modelo de alambres (derecha) del fichero del ejemplo original.

Como dice el propio tutorial cualquier formato de datos podría ser la fuente de partida para crear una superficie,incluso un fichero de audio … De momento, vamos a tomar una de las imágenes que podemos tener en nuestro equipo y utilizar a OpenCV para que decodifique el formato. De ahí, haremos una correspondencia entre los puntos de la imagen y los de la rejilla, para obtener así la representación tridimensional buscada.

3 Objetivos

Una vez que el lector haya leído este documento y explorado el código que se referencia y proporciona, será capaz de:

- Seguir la ejecución de los ejemplos.
- Explicar los diferentes elementos que componen la solución final.
- Elaborar nuevos métodos de asignación de valores para la representación tridimensional.

4 Requisitos

Al descargar el ejemplo de Nehe [4], que si no lo has hecho es el momento par a poder seguir los comentarios, hemos optado por la versión para GNU/Linux[2]. Esta incorpora un fichero *Makefile* para generar el ejecutable.

Para nuestro caso es necesario modificar las órdenes con que se compile para que también se pueda utilizar la librería de OpenCV. En concreto, los cambios afectan a dos reglas:.

[2]Portado a GNU/Linux y GLX por Patrick Schubert 2003

```
%.o: %.c
	$(CC) -c -g `pkg-config opencv --cflags`  -o $@ $<
lesson34: $(OBJECTS)
	$(CC) -g -o lesson34 $(OBJECTS) -L/usr/X11R6/lib -lm -lGL -lXxf86vm
	`pkg-config opencv --libs`
```

En las líneas mostradas se puede ver que se hace uso de la utilidad *pkg-config* para que, se escojan los parámetros pertinentes para a distribución y versión instalada.

Por otra parte la impaciencia me llevó a probar sin más a compilar el ejemplo y aparecieron una serie de errores debidos a la falta de una de las librerías:

```
$ make
gcc -Os -Wall -ansi -pedantic -ffloat-store -c -g -o main.o main.c
main.c:9:38: error: X11/extensions/xf86vmode.h: No existe el fichero o el directorio
main.c:19: error: expected ‘=’, ‘,’, ‘;’, ‘asm’ or ‘__attribute__’ before ‘bRender’
main.c: In function ‘LoadRawFile’:
main.c:84: warning: ignoring return value of ‘fread’, declared with attribute warn_unused_result
main.c: In function ‘RenderHeightMap’:
main.c:128: error: ‘bRender’
undeclared (first use in this function)
main.c:128: error: (Each undeclared identifier is reported only once
main.c:128: error: for each function it appears in.)
main.c: In function ‘update’:
main.c:212: error: ‘bRender’ undeclared (first use in this function)
lmake: *** [main.o] Error 1
```

En mi caso resulto fácil averiguar cual era por que las otras ya las había utilizado en anteriores desarrollos, pero puedes utilizar el mismo sistema par averiguar en tu caso si has de instalar alguna más. La orden *dpkg* no ayudará a saber si tenemos instalado algo relativo a *xxf86vm*:

```
$ dpkg -l *libxxf86vm*
||/ Nombre             Versión  Descripción
un  libxxf86vm-dev     <ninguna>      (no hay ninguna descripción disponible)
ii  libxxf86vm1        1:1.1.0-2      X11 XFree86 video mode extension library
un  libxxf86vm1-dbg    <ninguna>      (no hay ninguna descripción disponible)
```

A partir de aquí, sólo falta instalar lo necesario con *apt-get* desde un terminal en las distribuciones derivadas de Debian:

```
$ sudo apt-get install libxxf86vm-dev
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes extras:
  x11proto-xf86vidmode-dev
```

```
Se instalarán los siguientes paquetes NUEVOS:
  libxxf86vm-dev x11proto-xf86vidmode-dev
...
Configurando x11proto-xf86vidmode-dev (2.3-2) ...
Configurando libxxf86vm-dev (1:1.1.0-2) ...
```

5 Implementación

Para dar lugar a la funcionalidad de OpenCV hay que incluir los ficheros de cabecera y sustituir en el código la carga del fichero indicado en el código por una secuencia que permita cargar cualquier fichero de tipo imagen (conocido por OpenCV, claro). Los cambios en el programa principal se muestran en el listado 1.

```
#include <cv.h>
#include <cxcore.h>
#include <highgui.h>


...


int main(int argc, char **argv)
{
 int codic;
...
  if (argc == 1)  {
   printf("Necesite un nom de fitxer d'imatge per a treballar: %s rutaFitxer\n", argv[0]);  exit( 1 );
  }
 ...


  codic = OpenCV_RawData(argv[1], &files, &columnes, &g_HeightMap);
  // Load  data des de una image OpenCV
...
 return 0;
}
```

Listado 1: Modificaciones al código original para el uso de la funconalidad de OpenCV.

Si todo ha ido bien, al respecto de la carga del fichero de imagen por parte de OpenCV, sólo queda extraer la información de los puntos de la imagen, convertirlos y asignarlos a la variable *g_HeightMap* que hemos pasado por referencia a la función *OpenCV_RawData* que se muestra en el listado 2.

```
int OpenCV_RawData(char *rutaFitxer, int *files, int *columnes, unsigned char **pHeightMap)
{
 IplImage *imgOrg, *imgDst;
 unsigned char *aux;
 int x, y;
 CvScalar colorDst;

// Cargar la imagen de partida de fichero
 imgOrg = cvLoadImage( rutaFitxer, CV_LOAD_IMAGE_UNCHANGED);
 if (!imgOrg) {
   fprintf(stderr, "Problemas al crear la imagen en grises\n");
   return( -1 );
 }
 fprintf(stderr, "Se ha leido de %s  una imagen de %dx%d, de %d plano/s.\n",
        rutaFitxer, imgOrg->width, imgOrg->height, imgOrg->nChannels );
 // Crear la imagen donde se guardar<E1> el resultado
 imgDst = cvCreateImage(cvSize(imgOrg->width, imgOrg->height),IPL_DEPTH_8U, 1);
 if (!imgDst)  {
   fprintf(stderr, "Problemas al crear la imagen en grises\n");
   return( -2 );
 }
 cvCvtColor(imgOrg,  imgDst, CV_BGR2GRAY);
 fprintf(stderr, "Se ha creado una imagen de %dx%d, de %d plano/s.\n",
        imgDst->width, imgDst->height, imgDst->nChannels );
 (*files) = imgOrg->width;
 (*columnes) = imgOrg->height;
 *pHeightMap = (unsigned char *)malloc(imgOrg->width * imgOrg->height);
  if (pHeightMap == NULL)   return( -3 );
  else  { // Convertir y asignar
    aux = *pHeightMap;
    for ( x = 0; x < imgOrg->width; x++ )
    for ( y = 0; y < imgOrg->height; y++ )   {
     colorDst = cvGet2D(imgDst,y,(imgOrg->width-1)-x);
    *aux = (unsigned char)round(colorDst.val[0]);     aux++;
   } // Fin de "   for ( y = 0; y < imgOrg->height; y++ )"
  }// Fin de if-else ( imgOrg->nChannels != 1)
 cvReleaseImage( &imgOrg );  cvReleaseImage( &imgDst );
 return( 0 );
}// Fi de OpenCV_RawData
```

Listado 2: Modificaciones al código original para el uso de la funcionalidad de OpenCV.

Bueno, a parte yo he incluido un par más de modificaciones para que diga qué teclas puedo utilizar y dar alguna posibilidad más de cambio del punto de vista del observador. En cualquier caso, si no hemos cometido ningún fallo de transcripción ya se puede compilar y ejecutar desde la línea de órdenes con un fichero que tenía a mano (*meua.jpg*):

```
$ make && lesson34 meua.jpg
```

Prueba con una imagen tuya, puedes poner la ruta absoluta o relativa ;-) y, quizá, obtener algo al estilo de la fig. 3 o mejor.

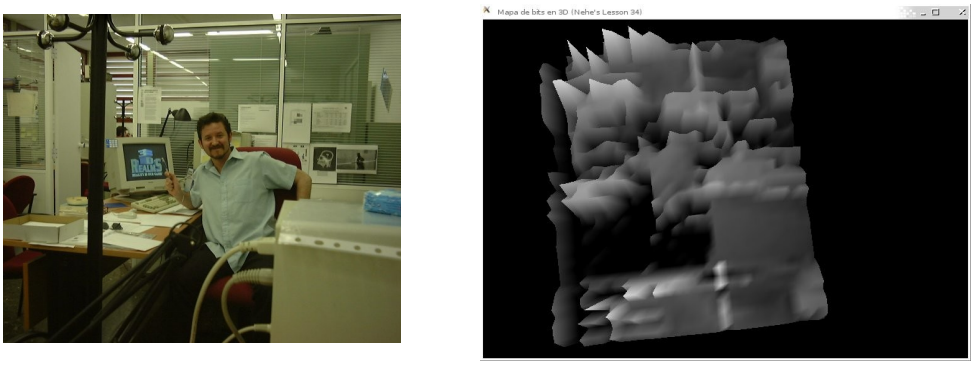

Figura 3: Otro ejemplo de salida de la aplicación realizada 2D (izquierda) y 3D (derecha).

6 Conclusión

Este artículo ha explorado un camino de proyección de información visual (gráfica) pensada para un plano (2D) a un espacio tridimensional. Trabajando con librerías multiplataforma (OpenCV y OpenGL) y sin ninguna atadura al sistema operativo permite portar esta aplicación a cualquier sistema y entorno de desarrollo para la creación del ejecutable.

Ahora, apreciado lector, es tu turno. Te animo a rebuscar en el código para ver que la asignación de filas y columnas entre la imagen y la rejilla no es la más acertada ... Otra cosa siempre interesante en una visualización 3D es ser capaz de variar el punto de vista del usuario (de la cámara en la terminología de síntesis de imagen), ampliar el soporte para variar las condiciones de partida del código ofrecido ayudará a "ver mejor" o desde nuevos puntos de vista el contenido de una imagen*. Suerte y ya nos contarás.*

7 Bibliografía y referencias

[1] "Open Computer Vision Library". Disponible en <http://sourceforge.net/projects/opencvlibrary/>. Última consulta mayo de 2011.

[2] "OpenCVWiki" , Disponible en http://opencv.w.illowgarage.com/wiki/Welcome.

[3] OpenGL, <http://connect.creativelabs.com/openal/default.aspx>. Última consulta enero de 2011.

[4] Jeff Molofee (NeHe)). Lesson: 34, <http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=34>. Última consulta mayo de 2011.