UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Integración de imagen real y sintética: OpenCV y OpenGL

Apellidos, nombre	Agustí i Melchor, Manuel (magusti@disca.upv.es)
Departamento	Departamento de Informática de Sistemas y Computadores (DISCA)
Centro	Escola Tècnica Superior d'Enginyeria Informàtica Universitat Politècnica de València

1 Resumen de las ideas clave

Este artículo se vertebra alrededor de cómo el computador puede generar una escena tridimensional, que integre tanto imagen sintética como la imagen real que capta una cámara digital conectada al computador. Se trata de mostrar un ejemplo de uso de las capacidades multimedia de los computadores actuales.

No es objetivo de este documento revisar el código hasta sus mínimos detalles, sino identificar las operaciones que hacen posibles los diferentes pasos y que permitirán al lector reutilizar este conocimiento en aplicaciones que demanden estas funcionalidades. El código fuente se puede obtener a vuelta de un correo electrónico dirigido al autor de este documento.

2 Introducción

Los esquemas tradicionales de aplicaciones de síntesis de imagen en 3D se sirven de fotografías para incorporar en sus escenas trozos del mundo real y dotar así de mayor realidad a su resultado. El cine, la televisión, los videojuegos o las aplicaciones de comunicaciones, por citar algunas actuales, demandan no solo mezclar imagen sintética y real, sino también usar fuentes de imagen real en movimiento, como una secuencia de vídeo pregrabado o una que está siendo tomada en ese momento.

En este artículo se muestra un posible diseño de una aplicación que utilizará OpenCV [1] para obtener esas imágenes tomadas en vivo y OpenGL [2] para la generación de la imagen 3D. Vamos a poner en una misma aplicación a OpenCV y OpenGL a colaborar para obtener información visual de carácter tridimensional mezclando imagen sintética y real. **Cómo integrar imágenes tomadas por una cámara real en una escena generada por computador es nuestra tarea.**

Para que el lector interesado, en función de su familiaridad con la temática, pueda experimentar, se exploran primero pequeños ejemplos de uso de OpenGL. Después se expondrá el tutorial de Millán [5] que muestra cómo tratar las imágenes que gestiona OpenCV como texturas en OpenGL. A partir de ahí expondremos las ampliaciones necesarias para realizar la tarea propuesta.

Figura 1: La prisión de cristal (también llamada “La zona fantasma”), fotograma de la película Superman (1978).

Veremos cómo se puede crear una aproximación a la “prisión de cristal”, véase la fig. 1[1], de la primera película de *Superman*: es un cristal con forma de polígono 2D que da vueltas sobre sí mismo simulando estar en alguna zona del

[1]La imagen se ha tomado de la página web de <https://myintuitivelife.files.wordpress.com/2012/03/superman-ii-dim.jpg>.

espacio. Dentro de ese monolito transparente se puede ver “apretados” a los que allí están encarcelados.

3 Objetivos

Una vez que el lector haya explorado los contenidos relacionados con la aplicación que aquí se diseña, será capaz de:

- Crear un plano en 3D sobre cuyas caras proyectar imágenes reales, como fotografías o vídeo.
- Identificar cómo OpenGL muestra imágenes en formato de mapa de bits con la ayuda de texturas.
- Identificar cómo OpenCV puede manipular directamente un conjunto de formatos de imágenes.
- Identificar cómo se puede acceder desde OpenGL a las estructuras de datos de OpenCV o, análogamente, cómo se pueden pasar las imágenes del formato soportado por OpenCV a las texturas de OpenGL.
- Resolver las dependencias para compilar una aplicación que use ambas bibliotecas de funciones.

4 Desarrollo

Veamos ahora cómo se pueden incorporar imágenes, desde fichero, a OpenGL y después plantearemos cómo hacer que sea OpenCV quien se ocupe de obtener las imágenes y nos ocuparemos de “comunicar” ambas librerías.

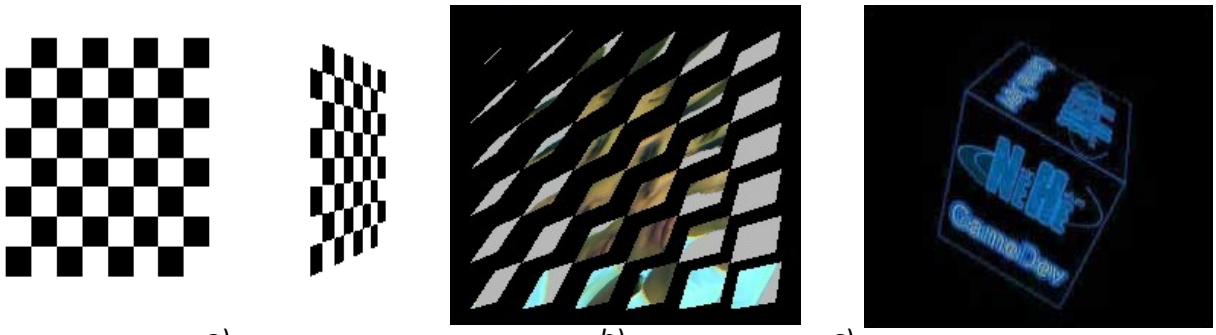
4.1 Ejemplos de base para el uso de OpenGL

Una imagen se puede añadir a una escena renderizada por OpenGL con *glDrawPixels* (véase ejemplo 8-3 de [2]), pero es necesario que el motor de renderizado aplique las transformaciones oportunas a la imagen cuando se quiere que sea aplicado a la imagen un efecto de perspectiva, iluminación o cualquier otro parámetro de la escena. En ese caso, la imagen se denomina una “textura mapeada”, esto es, tiene definida una correspondencia con, al menos, las coordenadas de un polígono en la escena.

Ejemplos del uso de texturas se muestran en la fig. 2. En la bibliografía de OpenGL se pueden encontrar descripciones exhaustivas de lo que es una textura y cómo se mapea una imagen a un objeto. Lo que sigue es una recopilación de las principales cuestiones a tener en cuenta en las versiones de OpenGL previas a la versión 4.0 que es donde se han probado los ejemplos.

El primer ejemplo que se muestra en la fig. 2a) es del ejemplo 9-1 de [2]. El listado 1 muestra la parte de código, de este ejemplo, que se encarga de la asignación del mapa de bits al objeto de tipo textura mediante la función *glTextImage2D*. La imagen se crea de manera algorítmica en tiempo de ejecución. Cabe destacar que el tamaño de la textura ha de ser potencia de dos (véase Specifiying the Texture, cap. 9 de [2]), pudiendo reescalar la imagen con *gluScaleImage* para obtener una que lo sea. Y también que, para optimizar, en aplicaciones como la nuestra en que se han de mostrar las imágenes capturadas como texturas en tiempo real, interesa crear una textura y utilizar la función *glTexSubImage2D* para reemplazar el contenido de la textura. Además,

esta función no está restringida al uso de imágenes que tengan un tamaño potencia de dos como es habitual en las secuencias de vídeo.



a) *b)* *c)*

Figura 2: Ejemplos de uso de texturas. Imágenes tomadas de [2] (a) [3] (b) y [4] (c).

```
…
#define checkImageWidth 64
#define checkImageHeight 64
static GLubyte checkImage[checkImageHeight][checkImageWidth][4];
static GLuint texName;
...
  glGenTextures(1, &texName);
  glBindTexture(GL_TEXTURE_2D, texName);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,  GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,  GL_NEAREST);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, checkImageWidth, checkImageHeight,
                  0, GL_RGBA, GL_UNSIGNED_BYTE, checkImage);
..
```

Listado 1: Selección del código del ejemplo 9-1 de [2] relacionado con el uso de texturas.

Se utiliza *glGenTextures* para [2] obtener un identificador del objeto y *glBindTexture* para asignarle el tipo textura. La función glGenTextures genera los nombres para el número de texturas indicadas, esto es, enteros que servirán de identificadores. De manera que se ha comprobado que no están en uso antes de la llamada a esta función y se garantiza que no se devuelven si ya estuvieran generadas y no han sido liberadas (con *glDeleteTextures*) previamente.

Las texturas así creadas, no tienen dimensión, se les asigna en la llamada glBinTexture. Esta función especifica el destino como bidimensional (GL_TEXTURE_2D); rompiendo, si existe, otras asignaciones previas. Puesto que los nombres y los contenidos de las texturas son [2] locales al espacio del objeto en un contexto de renderizado, hay que repetir este proceso en cada nueva secuencia de generación de la escena a renderizar por OpenGL.

La función glTexImage2D completa el proceso asignando, entre otros, valores concretos de anchura y altura, número y orden de las componentes de color, así como los valores de los píxeles de la imagen. El primer elemento leído se asigna a la esquina inferior izquierda de la textura, se continúa rellenando esa fila hacia las columnas de la derecha y, así, sucesivamente con el resto de filas. El último de los elementos que se lean se asigna a la esquina superior derecha. El resto de líneas del listado 5 definen otros parámetros relativos al ajuste entre la imagen y el polígono.

```
…
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glutBuild2DMipmaps(GL_TEXTURE_2D, depth, width, height, GL_RGB, GL_UNSIGNED_BYTE, bits);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
glEnable(GL_TEXTURE_2D);.
..
```

Listado 2: Selección del código de [3] relacionado con el uso de texturas.

En [3] se asume que la imagen ya está convertida a una estructura de datos de la aplicación y almacenada en un vector de bytes. Conocidos los parámetros que definen la imagen en cuanto a profundidad de color y resolución espacial se puede definir con instrucciones como las que muestra el listado 5. Este ejemplo además muestra el uso de la función *gluBuild2DMipmaps* para preparar versiones de la imagen a diferentes escalas de resolución que son utilizadas para mostrar la misma imagen a diferentes distancias del observador, con diferente escala o nivel de detalle cada una.

También podemos ver en [4] y otros tutoriales de este mismo autor, una gran variedad de cuestiones relativas a su uso en OpenGL. En ellos se asume que se dispone de una librería que puede leer y “entender” los formatos de ficheros gráficos que se piensen utilizar, incluida la posible compresión que se haya utilizado en el mismo. Así, que se puede generar la versión plana (sin la información que se guarda en la cabecera del fichero y descomprimida) que se almacena en una estructura de datos a partir de la que se puede hacer uso de las funciones indicadas.

4.2 OpenCV y OpenGL

Necesitamos todavía resolver cómo se puede pasar información entre las estructuras de datos que soportan OpenCV y OpenGL. Para ello revisamos el trabajo de Millán [5] cuya salida se muestra en la fig. 3. y cuyo funcionamiento se esboza en el diagrama de la fig. 4.

Este ejemplo permite llevar, en tiempo de ejecución, la imagen leída desde un fichero gráfico en disco, que leeremos utilizando OpenCV, a un elemento de OpenGL. En concreto, el código, a partir de un fichero JPEG, que debe existir junto al ejecutable y que tiene el nombre fijado en el código. La función *loadTexture* hace los ajustes oportunos para asignar la imagen a la textura.

Hay que hacer notar la importancia del valor *GL_BGR* para obtener correctamente los componentes de color. También hay que hacer notar que la

imagen aparece en pantalla al revés (puesto que como se ha comentado en el apartado anterior), es la manera en que se hacen corresponder en el código los píxeles de la imagen con los téxeles de la textura.

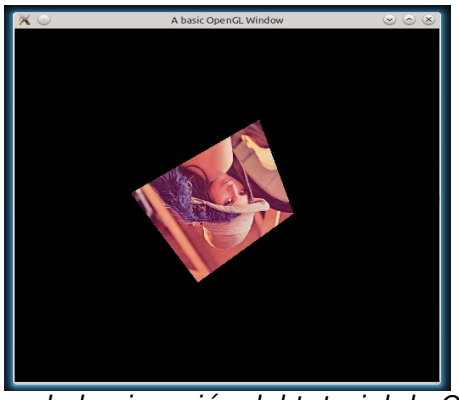


Figura 3: Captura de la ejecución del tutorial de OpenCV & OpenGL [5].

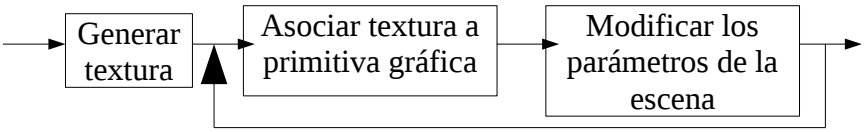


Figura 4: Diagrama básico del tutorial OpenCV & OpenGL de [5].

La aplicación sigue el diagrama esbozado en la fig. 4 y el código (se puede obtener completo de [5]) se resume en los listados 3 y 4. La imagen se lee una sola vez y se asigna a una textura; el resto del programa corresponde con un bucle de actualizaciones.

Estas corresponden a la actualización de la posición del único objeto en la escena (un “plano”) al que se aplica al textura. Para ello se modificará el valor de la variable *angle* que se utiliza para indicar la rotación del plano.

```
#include <stdio.h>
#include <GL/glut.h>
#include <GL/gl.h>
#include <ctype.h>
#include <cv.h>
#include <highgui.h>

GLuint texture; // Our texture
IplImage *imagenText1;
GLfloat angle = 0.0;
```

Listado 3: Código del tutorial OpenCV & OpenGL [5] (primera parte).

```
//The IplImage to OpenGL texture function
int loadTexture_Ipl(IplImage *image, GLuint *text) {
   if (image==NULL) return -1;
   glGenTextures(1, text);
   glBindTexture( GL_TEXTURE_2D, *text ); //bind the texture to it's array
   glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
   glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
   glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
   glTexIm age2D(GL_TEXTURE_2D, 0, GL_RGB, image->width, image->height,
                        0, GL_BGR, GL_UNSIGNED_BYTE, image->imageData);
   return 0;
}

void plane (void) {
   glBindTexture( GL_TEXTURE_2D, texture ); //bind the texture
   glRotatef( angle, 1.0f, 1.0f, 1.0f );
   glBegin (GL_QUADS);
   glTexCoord2d(0.0,0.0); glVertex2d(-1.0,-1.0); //with our vertices we have to assign a texcoord
   glTexCoord2d(1.0,0.0); glVertex2d(+1.0,-1.0); //so that our texture has some points to draw to
   glTexCoord2d(1.0,1.0); glVertex2d(+1.0,+1.0);
   glTexCoord2d(0.0,1.0); glVertex2d(-1.0,+1.0);
   glEnd();
}
...
int main (int argc, char **argv) {
   glutInit (&argc, argv);
...

   imagenText1=cvLoadImage("lena.jpg");
   loadTexture_Ipl( imagenText1, &texture );     //The load iplimage to opengl texture
   glutMainLoop ();

   FreeTexture( texture );  //Free our texture
   return 0;
}
```

Listado 4: Código del tutorial OpenCV & OpenGL [5] (segunda parte).

4.3 Construcción y distribución del ejecutable

La solución aquí presentada se ha llevado a cabo sobre una distribución de GNU/Linux Ubuntu 12.04, una versión 2.0 de OpenCV y una versión 2.1 de OpenGL (en la implementación de Mesa 8.0.4). Para construir el ejecutable será necesario compilar el código fuente con las cabeceras de las bibliotecas de funciones que implementan tanto el API de OpenCV como el de OpenGL y enlazar los ficheros objetos con las librerías correspondientes.

Suele ser un poco difuso el proceso de determinación de qué librerías son necesarias y cuál es el orden de compilación a utilizar. Voy a exponer una posible secuencia de órdenes para determinar cómo resolver las **dependencias** que existen en este caso.

Nuestra aplicación utiliza unas librerías que ha indicado con la inclusión de los ficheros de cabecera, véase el listado 3. Podemos agruparlas en tres grupos: las propias del lenguaje C (*stdio* y *cvtype*), las de OpenGL (*glut* y *gl*) y las de OpenCV (*cv* y *highgui*).

Para el caso de la librería estándar de C, los parámetros necesarios para la compilación y el enlazado los obtiene el compilador de las rutas por defecto del sistema. Para averiguar en nuestro sistema qué paquetes es necesario tener instalados para resolver las dependencias de OpenGL, buscaremos los que hacen referencia a las cabeceras del código con la ayuda de la orden *dpkg*. Con esta aplicación no se depende de un instalador gráfico en concreto, de hecho, casi todos los existentes (*dselect*, *synaptic*, *aptitude*, *muon*, etc.) son interfaces, más o menos elaboradas, a esta aplicación.

El listado 5 es un extracto de la salida de la orden *dpkg*, al ser preguntada por cosas relacionadas con cada uno de los ficheros de cabecera y en la que se solo se han dejado los resultados sobre los que se quiere poner la atención del lector. Como respuesta a la consulta sobre *glut* se devuelve en mi caso que están instalados *freeglut3* y *freeglut3-dev*. Se puede ver la versión del paquete y la descripción breve en cada línea. El primero es[2] el binario de la biblioteca dinámica de funciones, es el paquete habitual a instalar cuando solo se necesita como soporte de ejecución para alguna aplicación. El segundo incluye los ficheros de cabeceras, la biblioteca estática y la documentación asociada. Este es necesario para poder desarrollar, esto es, compilar, documentarse y generar una versión estática, si se necesita. Hay otra cabecera relativa a OpenGL, para la que procederemos del mismo modo, véase también el listado 5, con la orden *dpkg* pero con el parámetro "*gl*" para confirmar que se .dispone de ese paquete. En este caso corresponde a los paquetes relativos a la implementación libre de Mesa. Y, al respecto de OpenCV, se puede obtener con una orden similar a las del listado 5 las librerías necesarias.

```
$ dpkg -l "*glut*"
...
ii  freeglut3                 2.6.0-1ubuntu3      OpenGL Utility Toolkit
ii  freeglut3-dev             2.6.0-1ubuntu3      OpenGL Utility Toolkit development files
...


$ dpkg -l "*gl*"
...
ii  libglu1-mesa              8.0.4-0ubuntu0.7     Mesa OpenGL utility library (GLU)
ii  libglu1-mesa-dev          8.0.4-0ubuntu0.7    Mesa OpenGL utility library -- development files
ii  libglapi-mesa             8.0.4-0ubuntu0.7    free implementation of the GL API -- shared library
...
```

Listado 5: Paquetes relacionados con las bibliotecas de funciones GLUT y GL.

[2]El lector lo puede comprobar con la orden *dpkg -L freeglut3*.

Para averiguar los parámetros de compilación recurriremos, cuando sea posible, a la orden *pkg-config* que los actualizará, si es necesario, en función de la distribución utilizada. Así que finalmente, nuestra línea de órdenes para compilar es del estilo de la que aparece en el listado 6.

```
$ gcc OpenCV_OpenGL.cpp -o OpenCV_OpenGL `pkg-config --cflags opencv glu gl` -lglut `pkg-config --libs opencv glu gl`
```

Listado 6: Línea de órdenes para compilar el ejemplo OpenCV_OpenGL.cpp.

4.4 ¿Y ahora un cubo en 3D?

Como generalización, se podría ampliar el número de "planos" para, p. ej., alternar en ellos la visualización de imágenes obtenidas desde la cámara con las que se pueden obtener de los ficheros locales a modo de tiovivo o carrusel. Véase como ejemplo el modo "Carousel" de *XscreenSaver*[3] que muestra la fig. 5.

Figura 5: XScreenSaver funcionando en modo "Carousel".

Es cuestión de tomar el código anterior y añadir unos cuantos paneles más .... pero voy a proponer al lector explorar el ejemplo de S. Morf[4], que está basado en la variante de J. Oliver[5] de cómo conectar OpenCV y OpenGL. Utilice lo expuesto en el apartado "Construcción y distribución del ejecutable" para escribir la línea de órdenes que permite compilarlo. Por cierto, yo lo he guardado en un fichero *.c*, Vea el ejemplo indicado si no sabe a qué me refiero. ¡Ah, se necesitan pequeñísimos retoques para ser compilado sin errores! Revíselo, lo dejo a la experimentación del lector ...

El ejemplo, véase la fig. 6, muestra cómo llevar las imágenes a las caras (externas e internas) de un cubo "sin tapas". Aunque no es un cubo cerrado podría servir de límite visual de una escena y hacer "ver" o percibir al usuario, la escena como una caja en la que puede estar inmerso o ser vista desde fuera, fig. 6. En lugar de dibujar un solo rectángulo esta aplicación dibuja cuatro rectángulos para proyectar en ellos la imagen de la cámara. Dejo de manos del

---

[3] El sitio web de este protector de pantalla es <http://www.jwz.org/xscreensaver/screenshots/>. Se puede instalar desde los repositorios habituales.

[4] Se puede encontrar este ejemplo del 2008/2009 en la URL <http://www.gamedev.net/topic/539970-demo-headtracking-with-opencv—opengl/> y en el foro "OpenGL coding: advanced" <Forum: https://www.opengl.org/discussion_boards/showthread.php/167541-demo-opengl-opencv-tetures-with-webcamImages?s=011a9042c6816b58fb0687be6e9355c3>.

[5] OpenCV_GL.c (2007). <http://ljudmila.org/~julian/share/code/OpenCV_GL.c>.

lector, explorar el código para incluir la posibilidad de mostrar una imagen obtenida desde fichero e imágenes diferentes para cada rectángulo.

Figura 6: Capturas de la ejecución de la aplicación cubo3D.

5 Conclusión

Antes de cerrar este artículo quisiera agradecer a D. Millán su contribución por la prueba de concepto de cómo se comparte información entre objetos de OpenCV y OpenGL y que ha servido para desarrollar este artículo.

Tras la lectura y experimentación con los contenidos relativos a este documento, el lector será capaz de:

- Identificar el uso de texturas en OpenGL para mostrar información en formato de mapa de bits.
- Identificar cómo se puede acceder desde OpenGL a las estructuras de datos de OpenCV o, análogamente, cómo se pueden pasar las imágenes del formato soportado por OpenCV a las texturas de OpenGL.
- Resolver las dependencias para compilar una aplicación que use ambas bibliotecas de funciones y compilar el código resultante.

6 Bibliografía

[1] Open Computer Vision Library <http://sourceforge.net/projects/opencvlibrary/>. Consultada el 17 de marzo de 2015.
[2] Introduction to OpenGL. OpenGL Programming, Guide. Capítulo 1. <http://www.glprogramming.com/red/>. Consultada el 17 de marzo de 2015.
[3] M. J. Kilgard. (1994). Texture Warping. <https://www.opengl.org/archives/resources/code/samples/glut_examples/examples/examples.html>. Consultada el 17 de marzo de 2015.
[4] NeHe. Texture Mapping. <http://nehe.gamedev.net/tutorial/lessons_06__10/17010/>. Consultada el 17 de marzo de 2015.
[5] D. Millán. (2008). OpenCV & OpenGL. <http://blog.damiles.com/2008/10/opencv-opengl/>. Consultada el 17 de marzo de 2015.
[6] OpenGL Software Development Kit. URL <https://www.opengl.org/sdk/>. Consultada el 17 de marzo de 2015.