02/09/2016

Texturas

Seminario ISGI (S8)

R. Vivó

R. Vivó

Texturas

Seminario ISGI (S8)

El uso de texturas es, probablemente, el avance más importante en gráficos por computador para dotar a los objetos de una apariencia real. Las texturas permiten diferenciar de una manera muy sencilla el color de cada vértice del objeto dejando atrás la apariencia lisa de un objeto pintado con una única pintura. En este seminario se aprende cómo definir una textura y cómo aplicarla a un objeto.

¿Qué son y cómo se cargan?

Podríamos definir una textura como una función que, para cada fragmento, nos indica cómo variar el color calculado por el modelo de sombreado. La definición es tan amplia que cabe casi cualquier cosa. De hecho es así, una textura puede servir para definir el color del fragmento como si el objeto estuviera empapelado con papel pintado, para darle apariencia de superficie pulida que refleja lo que hay alrededor, para hacer como que el objeto es de mármol, para simular irregularidades en la superficie, etc.

Probablemente la mejor forma de entender una textura, y seguramente la más utilizada, es pensar en una imagen que se pega a la superficie de un objeto como si fuera de papel adhesivo. A este tipo de texturas se las conoce como texturas de superposición porque la imagen se superpone a la superficie del objeto definiendo su color en cada punto.

Nuestro punto de partida es entonces una imagen. La imagen está compuesta por pixeles y en la mayoría de los casos la tendremos en un fichero en un cierto formato (bmp, gif, png, jpg, etc.). Si la imagen la vemos en pantalla con cualquier visor de imágenes lo que vemos es una matriz de *NxM* pixeles RGB. A cada uno de estos pixeles se le suele llamar **texel**. En definitiva, nuestra textura será un conjunto de *NxM* texeles y un espacio de referencia. El espacio de la textura es un cuadrado en R^2 de extremos (0,0) y (1,1) de tal manera que dado un punto *(s,t)* del espacio de la textura localizamos un texel (que puede ser el mismo que otra coordenada próxima).

Para poder trabajar con la textura primero tenemos que leerla del fichero y cargarla en memoria. Desafortunadamente OpenGL no suministra funciones de utilidad para la carga y descarga de imágenes de fichero por lo que tendremos que recurrir a alguna librería especializada en este tipo de operaciones.

Hemos elegido FreeImage por su uso sencillo, posibilidad de cargar más de 20 formatos y ser de código libre y abierto. Podemos encontrarla en http://freeimage.sourceforge.net desde donde nos bajaremos el distribuible para Win32 que lleva:

1. FreeImage.h Este fichero es el de cabecera que hay que incluir en nuestra aplicación, por lo que lo depositaremos en un camino accesible (por ejemplo en el mismo directorio que glut.h).
2. FreeImage.lib Este fichero es la librería con la que debemos enlazar nuestro proyecto y debemos depositarla en lugar accesible (por ejemplo en el mismo directorio que glut32.lib).
3. FreeImage.dll Este fichero es la librería dinámica que usaremos en ejecución por lo que debe estar en un directorio alcanzable por la variable PATH (por ejemplo en el mismo directorio que glut32.dll).

La librería necesita de una inicialización y una liberación que haremos en nuestro programa principal así:

```
void main(int argc, char** argv)
{
	FreeImage_Initialise();
	glutInit(&argc,argv);
	…
	glutMainLoop();
	FreeImage_DeInitialise();
}
```

Para cargar una imagen de fichero en una textura en memoria incluiremos en nuestro código la siguiente función:

```
loadImageFile(char* nombre)
// Uso de FreeImage para cargar la imagen en cualquier formato
// nombre: nombre del fichero con extensión en el mismo directorio que el proyecto
//         o con su path completo
{
	// Detección del formato, lectura y conversion a BGRA
	FREE_IMAGE_FORMAT formato = FreeImage_GetFileType(nombre,0);
	FIBITMAP* imagen = FreeImage_Load(formato, nombre);
	FIBITMAP* imagen32b = FreeImage_ConvertTo32Bits(imagen);

	// Lectura de dimensiones y colores
	int w = FreeImage_GetWidth(imagen32b);
	int h = FreeImage_GetHeight(imagen32b);
	GLubyte* texeles = FreeImage_GetBits(imagen32b);

	// Carga como textura actual
	glTexImage2D( GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_BGRA,
		      GL_UNSIGNED_BYTE, texeles);

	// Liberar recursos
	FreeImage_Unload(imagen);
	FreeImage_Unload(imagen32b);
}
```

Aprovechando la misma librería podemos construir también una función de utilidad para que grabe la imagen que hemos dibujado como un fichero de imagen al disco. La función es la siguiente:

```
void saveScreenshot(char* nombre, int ancho, int alto)
// Utiliza FreeImage para grabar un png
// nombre: Nombre del fichero con extensión p.e. salida.png
// ancho: Ancho del viewport en pixeles
// alto: Alto del viewport en pixeles
{
	int pix = ancho * alto;
	BYTE *pixels = new BYTE[3*pix];
	glReadBuffer(GL_FRONT);
	glReadPixels(0,0,ancho,alto,GL_BGR,GL_UNSIGNED_BYTE, pixels);
	FIBITMAP *img = FreeImage_ConvertFromRawBits(pixels, ancho, alto,
				ancho*3, 24, 0xFF0000, 0x00FF00, 0x0000FF, false);
	FreeImage_Save(FIF_PNG, img, nombre, 0);
	delete pixels;
}
```

No se ha incluido en el código anterior la gestión de errores por limpieza y brevedad, aunque hay que hacerla en cualquier caso. Es conveniente leer la documentación de FreeImage (que se encuentra en la distribución) para despejar dudas y hacer esa gestión.

Pasos para texturar un objeto

Para definir qué textura se aplicará a un objeto y cómo se hará descomponemos el proceso en dos fases:

1. Fase de carga. En la fase de carga definimos cuál es la textura y la activamos. Para esto necesitamos:

    a. Generar un objeto textura. Un objeto textura es una textura cargada en un contenedor de OpenGL al que podremos hacer referencia después. Podemos definir múltiples objetos textura para tener diferentes texturas. Para generar un nuevo objeto textura solicitamos un identificador a OpenGL así:

    ```
    GLuint mitex;
    glGenTextures(1,&mitex);
    ```

    b. Seleccionar el objeto textura. Hay un objeto textura actual al que se le aplican los cambios siguiendo la técnica de la máquina de estados. Para hacer uno de los objetos textura el actual se usa la orden:

    ```
    glBindTexture(GL_TEXTURE_2D, mitex);
    ```

    c. Cargar la imagen. Usaremos la función `loadImageFile()` que hemos visto en el apartado anterior. La imagen se carga en el objeto de textura actual.

    d. Habilitar texturas. Para indicar a OpenGL que haga caso a las texturas hay que emplear la orden:

    ```
    glEnable(GL_TEXTURE_2D);
    ```

2. Fase de aplicación. Si la fase de carga se hace en la inicialización de la aplicación, la de aplicación se debe hacer en la de dibujo, sobre todo si se pretende cambiar la forma en la que se aplica la textura o si tenemos varias texturas.

    a. Seleccionar el objeto textura corriente con `glBindTexture`.

    b. Definir los filtros. Los filtros son las maneras en que se hace corresponder texeles y pixeles. Hay dos tipos de filtros el de *magnificación* cuando un téxel es más pequeño que un píxel y el de *minificación* que es cuando el téxel es más grande que el píxel. Se pueden usar diferentes tipos de filtros. El más sencillo es el que usa el valor más próximo disponible `GL_NEAREST`, aunque podemos usar otro que produce mejores resultados `GL_LINEAR` a fuerza de hacer más operaciones. Para indicar el tipo de filtrado usamos la orden:

    ```
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    ```

    donde se han usado como ejemplo los dos tipos de filtrado.

    c. Definir la forma de combinar. La textura se puede combinar con el color obtenido en la iluminación `GL_MODULATE`, puede reemplazar totalmente al color calculado `GL_REPLACE` o se puede usar el canal alfa con `GL_BLEND`. Para indicar cómo combinar la textura con el color subyacente usamos la orden:

    ```
    glTexEnvi(GL_TEX_ENV, GL_TEX_ENV_MODE, GL_MODULATE)
    ```

donde los dos primeros parámetros son fijos y el tercero indica la combinación elegida.

d. Dar coordenadas de textura. Para pegar la textura al objeto hay que indicar, para cada vértice, cuál es su coordenada en el espacio de la textura. El espacio de la textura nos lo podemos imaginar como un plano donde la textura se repite en cuadrados de 1x1 alineada con el origen (ver figura 1).

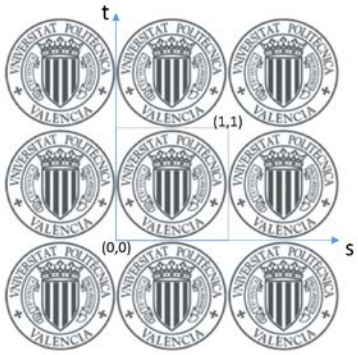


Figura 1. Espacio de la textura

Antes de hacer la llamada a `glVertex` hay que indicar la coordenada que tiene en el espacio de la textura con la orden:

```
glTexCoord2f(s,t)
```

donde `s` y `t` son las coordenadas del vértice en el espacio de la textura.

Si los filtros y la forma de combinar son los mismos para todas las texturas, las acciones b y c de la fase de aplicación pueden llevarse a la fase de definición.

Ejercicio S8E01: Dibujar un quad con una textura repetida 6 veces.

Ejercicio S8E02: Dibujar dos teteras con diferente textura y modo de combinación una al lado de la otra. Observar que glutSolidTeapot ya define las coordenadas de textura.

Una última cuestión relativa a la forma de combinar la textura con el color del objeto iluminado con `GL_MODULATE` es la forma en la que quedan afectados los brillos. Si no decimos nada, primero se calcular la iluminación -incluidos los brillos- y después se combina con la textura, con lo que los brillos quedarán "detrás" de la textura. Hay otra opción para resaltar los brillos que es calcular la iluminación sin ellos, combinarla con la textura y después aplicar los brillos. Para habilitar esta forma de proceder usaremos la orden:

```
glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL, GL_SEPARATE_SPECULAR_COLOR)
```

que podemos desactivar usando `GL_SINGLE_COLOR` como segundo parámetro.

Ejercicio S8E03: Dibujar dos teteras iluminadas con los brillos separados e integrados en la textura respectivamente.

Generación automática de coordenadas de textura

A veces es complicado generar las coordenadas de textura y nos es suficiente con "proyectar" una textura sobre el objeto para que parezca de un determinado material. Afortunadamente podemos usar una

opción de OpenGL que genera por nosotros las coordenadas de textura, basándose en las coordenadas que tienen los vértices, con alguna indicación por nuestra parte para orientar la textura.

Cuando se habilita la generación automática de textura OpenGL no hace caso de las coordenadas que nosotros hayamos consignado con `glTexCoord`. Para habilitar la generación automática de coordenadas de textura usaremos la orden:

```
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
```

que corresponden a las coordenadas de una textura bidimensional como es nuestra imagen.

Para alinear la textura con el objeto usaremos las siguientes funciones:

```
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
glTexGenfv(GL_S, GL_OBJECT_PLANE, planoS);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
glTexGenfv(GL_T, GL_OBJECT_PLANE, planoT);
```

donde `planoS` y `planoT` son vectores de 4 `GLfloat` que representan los coeficientes de las ecuaciones implícitas de dos planos. Las coordenadas de textura *s* y *t* son las distancias del vértice a los planos `planoS` y `planoT` respectivamente. Todo este cálculo lo hace OpenGL por lo sólo nos debemos de preocupar de dar los planos correctos según la orientación que queramos (ver figura).

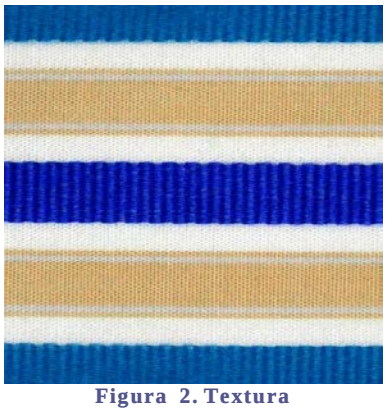


Figura 2. Textura

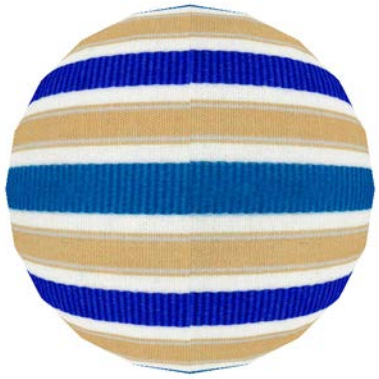


Figura 3. Esfera texturada. planoS={1,0,0,0}; planoT={0,1,0,0}

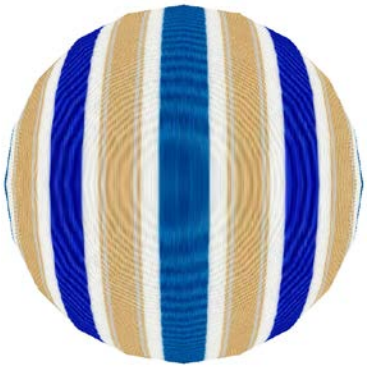


Figura 4. Esfera texturada. planoS={0,0,1,0}; planoT={1,0,0,0}

El modo `GL_OBJECT_LINEAR` las coordenadas de textura se calculan en coordenadas del modelo, o sea a partir de los valores de `glVertex`, por lo que cuando el objeto se mueve la textura también lo hace como si fuera pegada.

Nota: Cuidado con la tetera de GLUT que originalmente se dibuja con el eje z como su vertical, lo que puede desorientar al aplicar este tipo de texturas.

Ejercicio S8E04: Buscar una textura de mármol y aplicarla a una tetera mediante generación automática de coordenadas de textura. Mostrar la tetera iluminada. Observar la diferencia generando o no automáticamente las coordenadas de textura.

Reflejo del entorno

Una de las opciones más interesantes de la generación automática de coordenadas de textura es cuando se hace en función de la posición del observador y la normal en cada punto. Si pensamos en una visual

como una pelota que parte del observador y rebota en la superficie del objeto, el objeto que alcance después de rebotar es el que vemos reflejado donde botó. Es lo que pasa cuando vemos los objetos reflejados en un espejo.

OpenGL puede calcular la dirección del rebote automáticamente y asignar unas coordenadas de una textura. El efecto es que vemos en el objeto cómo se refleja la textura como si fuera lo que tenemos alrededor. El efecto es impresionante cuando movemos el objeto y vemos como se desplazan los reflejos.

Para hacer esto simplemente tenemos que indicarlo así:

```
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
```

acordándose de habilitar la generación automática con `glEnable` como antes.

Ejercicio S8E05: Buscar una imagen de cielo con nubes, ponerla como fondo y dibujar una tetera que lo refleje.

Ejercicio S8E01: Dibujar un quad con una textura repetida 6 veces.

```
/************************************************
ISGI::Quad Texturado
Roberto Vivo', 2013 (v1.0)

Dibuja un quad con una textura repetida 6 veces

Dependencias:
+GLUT +FreeGlut
************************************************/
#define PROYECTO "ISGI::S8E01::Quad con Textura"
//#define GL_BGRA 0x80E1

#include <iostream>                             // Biblioteca de entrada salida
#include <gl\freeglut.h>                        // Biblioteca grafica
#include <freeimage/FreeImage.h>                // Biblioteca de gestion de imagenes
using namespace std;

//Variables globales

static int xantes,yantes;                       // Valor del pixel anterior
static float girox=0,giroy=0;                   // Valor del giro a acumular
static float escalado=1;                        // Valor del escalado acumulado
static enum Interaccion {GIRO,ESCALADO} accion;  // Tipo de acción de inspección

void saveScreenshot(char* nombre, int ancho, int alto)
// Utiliza FreeImage para grabar un png
// nombre: Nombre del fichero con extensión p.e. salida.png
// ancho: Ancho del viewport en pixeles
// alto: Alto del viewport en pixeles
{
        int pix = ancho * alto;
        BYTE *pixels = new BYTE[3*pix];
        glReadBuffer(GL_FRONT);
        glReadPixels(0,0,ancho,alto,GL_BGR_EXT,GL_UNSIGNED_BYTE, pixels);
        FIBITMAP *img = FreeImage_ConvertFromRawBits(pixels, ancho, alto, ancho*3, 24,
                                                     0xFF0000, 0x00FF00, 0x0000FF, false);
        FreeImage_Save(FIF_PNG, img, nombre, 0);
        cout << "Captura de ventana realizada en " << nombre << endl;
        delete pixels;
}

void loadTexture()
// Funcion de carga de texturas e inicializacion
{
        //1a. GENERAR UN OBJETO TEXTURA
        GLuint tex0;
        glGenTextures(1,&tex0);

        //1b. ACTIVAR EL OBJETO TEXTURA
        glBindTexture(GL_TEXTURE_2D,tex0);

        //1c. CARGAR LA IMAGEN QUE SERVIRA DE TEXTURA
        FREE_IMAGE_FORMAT formato = FreeImage_GetFileType("upv.jpg",0);
        // Automatically detects the format
        FIBITMAP* imagen = FreeImage_Load(formato, "upv.jpg");
        if(imagen==NULL) cerr << endl << "NO SE ENCONTRO LA IMAGEN" << endl;
        FIBITMAP* imagen32b = FreeImage_ConvertTo32Bits(imagen);
        int w = FreeImage_GetWidth(imagen32b);
        int h = FreeImage_GetHeight(imagen32b);
        GLubyte* pixeles = FreeImage_GetBits(imagen32b);
        // FreeImage loads in BGR format, so you need GL_BGRA.
        glTexImage2D( GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_BGRA_EXT, GL_UNSIGNED_BYTE, pixeles);
        FreeImage_Unload(imagen);
        FreeImage_Unload(imagen32b);

        //1d. HABILITAR LAS TEXTURAS
        glEnable(GL_TEXTURE_2D);

        //2a. SELECCIONAR EL OBJETO TEXTURA
        //glBindTexture(GL_TEXTURE_2D,tex0);
```

```
        //2b. DEFINIR COMO SE APLICARÁ LA TEXTURA EN ESE OBJETO
        // Texel menor que pixel
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        // Texel mayor que pixel
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        // La textura se repite en abcisas
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        // La textura se repite en ordenadas
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

        //2c. DEFINIR LA FORMA DE COMBINAR
        // Asigna solo el color de la textura al fragmento
        glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
}

void init()
// Funcion propia de inicializacion
{
        // Mensajes por consola
        cout << PROYECTO << " running" << endl;
        cout << "Version: OpenGL " <<  glGetString(GL_VERSION) << endl;

        cout << "Arrastre con boton izquierdo: Gira la pieza" << endl;
        cout << "Arrastre con boton derecho: Aumenta o disminuye" << endl;
        cout << "S: Captura la ventana en captura.png" << endl;

        glClearColor(1.0,1.0,1.0,1.0);                  // Color de fondo a blanco
        glEnable(GL_DEPTH_TEST);                        // Habilita visibilidad

        loadTexture();

}

void display()
// Funcion de atencion al dibujo
{
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);     // Borra la pantalla
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();

  gluLookAt(0,0,5,0,0,0,0,1,0);                         // Situa la camara

  // La inspeccion se realiza en el sistema de coordenadas relativo
  // al objeto. Por eso los giros son respecto a los ejes X,Y del
  // modelo. VIEW*RX*RY*S
  glPushMatrix();
  glRotatef(girox,1,0,0);                               // Giro en x
  glRotatef(giroy,0,1,0);                               // Giro en y
  glScalef(escalado,escalado,escalado);                 // Escalado

  //2d. DAR LAS COORDENADAS DE TEXTURA EN CADA VERTICE
  glBegin(GL_QUADS);
  glTexCoord2f(-1,-1);
  glVertex3f(-1,-1,0);
  glTexCoord2f(2,-1);
  glVertex3f(1,-1,0);
  glTexCoord2f(2,1);
  glVertex3f(1,1,0);
  glTexCoord2f(-1,1);
  glVertex3f(-1,1,0);
  glEnd();
  glPopMatrix();

  glutSwapBuffers();                                    // Intercambia los buffers
}

void reshape(GLint w, GLint h)
// Funcion de atencion al redimensionamiento
{
        // Usamos toda el area de dibujo
        glViewport(0,0,w,h);
```

```
	// Definimos la camara (matriz de proyeccion)
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();

	float razon = (float) w / h;

	/* CAMARA PERSPECTIVA */
	gluPerspective(60,razon,1,10);
}

void onClick(int button,int state, int x, int y)
// Funcion de atencion al boton del raton
// button: GLUT_LEFT|MIDDLE|RIGHT_BUTTON
// state: GLUT_UP|DOWN
// x,y: pixel respecto a vertice superior izquierdo
{
	//La inspección puede ser girando la pieza o variando el tamaño
	switch (button)
	{
		case GLUT_LEFT_BUTTON:		// Girar la pieza
			accion= GIRO;
			xantes=x;		// Guarda el valor del pixel picado
			yantes=y;
			break;
		case GLUT_RIGHT_BUTTON:		// Escalar la pieza
			accion= ESCALADO;
			yantes=y;		// La escala se maneja con mvto. vertical del ratón
			break;
	};
}

void onMotion(int x, int y)
// Funcion de atencion al raton con el boton pulsado
// x,y: coordenadas del cursor referidas al pixel superior izquierdo(0,0)
{
	static const float pix2deg = 1.0;	// Factor de conversión pixel a grados
	static const float pix2fac = 0.01;	// Factor de conversión pixel a escalado

	switch(accion){				// La accion la determina el boton pulsado
	case GIRO:				// La acumulación del giro se produce aqui
		girox+= (y - yantes) * pix2deg;	// y crece hacia abajo. giro antihorario en x
		giroy+= (x - xantes) * pix2deg;	// x crece hacia derecha. giro antihorario en y
		yantes=y;
		xantes=x;
	break;
	case ESCALADO:				// La acumulación del escalado se lleva en "escalado"
		escalado+= (yantes - y) * pix2fac;// y crece hacia abajo. escalado crece hacia arriba
		yantes=y;
	break;
	};

	glutPostRedisplay();
}

void onKey(unsigned char tecla, int x, int y)
// Funcion de atencion al teclado
{
	switch(tecla){
	case 'S':						// Screenshot
		GLint vport[4];
		glGetIntegerv(GL_VIEWPORT, vport);		// Recupera el viewport corriente
		saveScreenshot("captura.png", vport[2], vport[3]);
		break;
	case 27:						// Pulso escape
		exit(0);
	}
	glutPostRedisplay();
}


void main(int argc, char** argv)
// Programa principal
{
```

```
  FreeImage_Initialise();                              // Inicializacion de FreeImage
  glutInit(&argc, argv);                               // Inicializacion de GLUT
  glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);  // Alta de buffers a usar
  glutInitWindowSize(400,400);                         // Tamanyo inicial de la ventana
  glutCreateWindow(PROYECTO);                          // Creacion de la ventana con su titulo
  glutDisplayFunc(display);              // Alta de la funcion de atencion a display
  glutReshapeFunc(reshape);              // Alta de la funcion de atencion a reshape
  glutMouseFunc(onClick);                // Alta de la funcion de atencion al click del ratón
  glutMotionFunc(onMotion);              // Alta de la funcion de atencion al movimiento del ratón
  glutKeyboardFunc(onKey);               // Alta de la funcion de atencion al teclado
  init();                                // Inicializacion propia
  glutMainLoop();                        // Puesta en marcha del programa
  FreeImage_DeInitialise();              // Cierre de FreeImage
}
```

Ejercicio S8E02: Dibujar dos teteras con diferente textura y modo de combinación una al lado de la otra. Observar que glutSolidTeapot ya define las coordenadas de textura.

```
/************************************************
ISGI::Dos Texturas
Roberto Vivo', 2013 (v1.0)

Dibuja dos teteras con diferente textura y modo de
combinacion

Dependencias:
+GLUT +FreeGlut
************************************************/
#define PROYECTO "ISGI::S8E02::Dos texturas"
//#define GL_BGRA 0x80E1

#include <iostream>                             // Biblioteca de entrada salida
#include <gl\freeglut.h>                        // Biblioteca grafica
#include <freeimage/FreeImage.h>                // Biblioteca de gestion de imagenes
using namespace std;

//Variables globales
static int xantes,yantes;                       // Valor del pixel anterior
static float girox=0,giroy=0;                   // Valor del giro a acumular
static float escalado=1;                        // Valor del escalado acumulado
static enum Interaccion {GIRO,ESCALADO} accion; // Tipo de acción de inspección
static  GLuint tex[2];                          // Ids de texturas

{…}

void loadImageFile(char* nombre)
// Uso de FreeImage para cargar la imagen en cualquier formato
// nombre: nombre del fichero con extensión en el mismo directorio que el proyecto
//         o con su path completo
{
        // Detección del formato, lectura y conversion a BGRA
        FREE_IMAGE_FORMAT formato = FreeImage_GetFileType(nombre,0);
        FIBITMAP* imagen = FreeImage_Load(formato, nombre);
        FIBITMAP* imagen32b = FreeImage_ConvertTo32Bits(imagen);

        // Lectura de dimensiones y colores
        int w = FreeImage_GetWidth(imagen32b);
        int h = FreeImage_GetHeight(imagen32b);
        GLubyte* texeles = FreeImage_GetBits(imagen32b);

        // Carga como textura actual
        glTexImage2D( GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_BGRA_EXT, GL_UNSIGNED_BYTE, texeles);

        // Liberar recursos
        FreeImage_Unload(imagen);
        FreeImage_Unload(imagen32b);
}

void loadTexture()
// Funcion de carga de texturas e inicializacion
{
        //1a. Generar un objeto textura
        glGenTextures(2,tex);
        //1b. Activar el objeto textura
        glBindTexture(GL_TEXTURE_2D,tex[0]);
        //1c. Cargar la imagen que servira de textura
        loadImageFile("bambu.jpg");

        //1b. Activar el objeto textura
        glBindTexture(GL_TEXTURE_2D,tex[1]);
        //1c. Cargar la imagen que servira de textura
        loadImageFile("rojos.jpg");
        //1d. Habilitar las texturas
        glEnable(GL_TEXTURE_2D);
}

void loadLight()
```

```
{
        //Luces
        glEnable(GL_LIGHT0);

        //Materiales
        GLfloat mat_diffuse[] = {0.5,0.5,1,1.0};                //Kd
        GLfloat mat_specular[] = {1,1,1,1.0};                   //Ks
        GLfloat mat_shininess[] = {100.0};                      //n
        glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
        glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
        glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

        //Habilita la iluminación
        glEnable(GL_LIGHTING);
}


void display()
// Funcion de atencion al dibujo
{
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);// Borra la pantalla
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();

        gluLookAt(0,0,5,0,0,0,0,1,0);                           // Situa la camara

        // La inspeccion se realiza en el sistema de coordenadas relativo
        // al objeto. Por eso los giros son respecto a los ejes X,Y del
        // modelo. VIEW*RX*RY*S

        glRotatef(girox,1,0,0);                                 // Giro en x
        glRotatef(giroy,0,1,0);                                 // Giro en y
        glScalef(escalado,escalado,escalado);                   // Escalado

        //2a. Seleccionar el objeto textura
        glBindTexture(GL_TEXTURE_2D,tex[0]);
        //2b. Definir como se aplicará la textura en ese objeto
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        //2c. Definir la forma de combinar
        glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
        // Dibujar el primer objeto
        glPushMatrix();
        glTranslatef(-1,0,0);
        glRotatef(-90,0,1,0);
        glutSolidTeapot(1.0);
        glPopMatrix();

        //2a. Seleccionar el objeto textura
        glBindTexture(GL_TEXTURE_2D,tex[1]);
        //2b. Definir como se aplicará la textura en ese objeto
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        //2c. Definir la forma de combinar
        glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);//Combina con la iluminacion

        // Dibujar el segundo objeto
        glPushMatrix();
        glTranslatef(1,0,0);
        glRotatef(-90,0,1,0);
        glutSolidTeapot(1.0);
        glPopMatrix();

        glutSwapBuffers();                                  // Intercambia los buffers
}
```

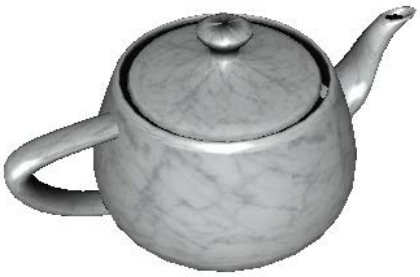
{…}

Ejercicio S8E03: Dibujar dos teteras iluminadas con los brillos separados e integrados en la textura respectivamente.

```
/*************************************************
ISGI::Brillos en Texturas
Roberto Vivo', 2013 (v1.0)

Dibuja dos teteras con la misma textura, una con el
brillo separado de la textura y otra integrado

Dependencias:
+GLUT +glext +FreeGlut
*************************************************/
#define PROYECTO "ISGI::S8E03::Brillos y Textura"

#include <iostream>                             // Biblioteca de entrada salida
#include <gl\freeglut.h>                        // Biblioteca grafica
#include <GL/glext.h>                           // Biblioteca de extensiones de GL
#include <freeimage/FreeImage.h>                // Biblioteca de gestion de imagenes
using namespace std;

//Variables globales

static int xantes,yantes;                       // Valor del pixel anterior
static float girox=0,giroy=0;                   // Valor del giro a acumular
static float escalado=1;                        // Valor del escalado acumulado
static enum Interaccion {GIRO,ESCALADO} accion; // Tipo de acción de inspección
static  GLuint tex[1];                          // Ids de texturas

…

void loadTexture()
// Funcion de carga de texturas e inicializacion
{
        //1b. Activar el objeto textura
        glBindTexture(GL_TEXTURE_2D,tex[0]);

        //1c. Cargar la imagen que servira de textura
        loadImageFile("rojos.jpg");

        //1d. Habilitar las texturas
        glEnable(GL_TEXTURE_2D);

        //2a. Seleccionar el objeto textura
        glBindTexture(GL_TEXTURE_2D,tex[0]);

        //2b. Definir como se aplicará la textura en ese objeto
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        //2c. Definir la forma de combinar
        glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

}

void display()
// Funcion de atencion al dibujo
{
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);// Borra la pantalla
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();

        gluLookAt(0,0,5,0,0,0,0,1,0);                           // Situa la camara

        glRotatef(girox,1,0,0);                                 // Giro en x
        glRotatef(giroy,0,1,0);                                 // Giro en y
        glScalef(escalado,escalado,escalado);                   // Escalado

        // Brillos separados de la textura
        glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL, GL_SEPARATE_SPECULAR_COLOR);
```

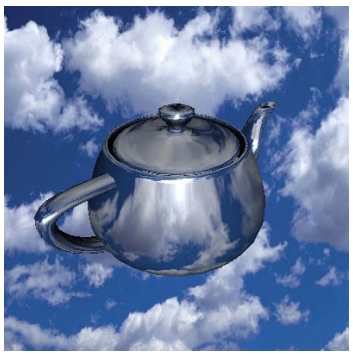
```
        // Dibujar el primer objeto
        glPushMatrix();
        glTranslatef(-1,0,0);
        glRotatef(-90,0,1,0);
        glutSolidTeapot(1.0);
        glPopMatrix();

        // Brillos integrados en la textura
        glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL, GL_SINGLE_COLOR);

        // Dibujar el segundo objeto
        glPushMatrix();
        glTranslatef(1,0,0);
        glRotatef(-90,0,1,0);
        glutSolidTeapot(1.0);
        glPopMatrix();
        glutSwapBuffers();                                      // Intercambia los buffers
}
```

Ejercicio S8E04: Buscar una textura de mármol y aplicarla a una tetera mediante generación automática de coordenadas de textura. Mostrar la tetera iluminada. Observar la diferencia generando o no automáticamente las coordenadas de textura.



```
/*************************************************
ISGI::Texturas automáticas
Roberto Vivo', 2013 (v1.0)

Dibuja dos teteras con la misma textura, una con el
brillo separado de la textura y otra integrado

Dependencias:
+GLUT +glext +FreeGlut
*************************************************/
#define PROYECTO "ISGI::S8E04::Texturas automáticas"

…

void loadTexture()
// Funcion de carga de texturas e inicializacion
{
        //1b. Activar el objeto textura
        glBindTexture(GL_TEXTURE_2D,tex[0]);
        //1c. Cargar la imagen que servira de textura
        loadImageFile("marmol.jpg");
        //1d. Habilitar las texturas
        glEnable(GL_TEXTURE_2D);
        //2a. Seleccionar el objeto textura
        glBindTexture(GL_TEXTURE_2D,tex[0]);
        //2b. Definir como se aplicará la textura en ese objeto
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        //2c. Definir la forma de combinar
        glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

        // Generacion automática de texturas
        glEnable(GL_TEXTURE_GEN_S);
        glEnable(GL_TEXTURE_GEN_T);

        // Alineacion de la textura
        GLfloat planoS[] = {1,0,0,0};
        GLfloat planoT[] = {0,1,0,0};
        glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
        glTexGenfv(GL_S, GL_OBJECT_PLANE, planoS);
        glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
        glTexGenfv(GL_T, GL_OBJECT_PLANE, planoT);
}

void display()
// Funcion de atencion al dibujo
{
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);// Borra la pantalla
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        gluLookAt(0,0,5,0,0,0,0,1,0);                   // Situa la camara
        glRotatef(girox,1,0,0);                         // Giro en x
        glRotatef(giroy,0,1,0);                         // Giro en y
        glScalef(escalado,escalado,escalado);           // Escalado

        // Brillos separados de la textura
        glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL, GL_SEPARATE_SPECULAR_COLOR);

        // Dibujar el objeto
        glPushMatrix();
        glutSolidTeapot(1.5);
        glPopMatrix();

        glutSwapBuffers()                               // Intercambia los buffers
}
```

Ejercicio S8E05: Buscar una imagen de cielo con nubes, ponerla como fondo y dibujar una tetera que lo refleje.



```
/*************************************************
ISGI::Mapa de Entorno Esferico
Roberto Vivo', 2013 (v1.0)

Dibuja un fondo que se refleja en una tetera

Dependencias:
+GLUT +glext +FreeGlut
*************************************************/
#define PROYECTO "ISGI::S8E05::Mapa de Entorno"

…

void loadTexture()
// Funcion de carga de texturas e inicializacion
{
        //1b. Activar el objeto textura
        glBindTexture(GL_TEXTURE_2D,tex[0]);

        //1c. Cargar la imagen que servira de textura
        loadImageFile("nubes.jpg");

        //1d. Habilitar las texturas
        glEnable(GL_TEXTURE_2D);

        //2a. Seleccionar el objeto textura
        glBindTexture(GL_TEXTURE_2D,tex[0]);

        //2b. Definir como se aplicará la textura en ese objeto
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        //2c. Definir la forma de combinar
        glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

        // Generacion automática de texturas
        glEnable(GL_TEXTURE_GEN_S);
        glEnable(GL_TEXTURE_GEN_T);

        // La texturas se calculan como mapa de entorno
        glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
        glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
}


void loadBackground()
// Funcion de carga de la textura actual como fondo de la ventana
{
        // Configura la proyeccion con camara ortgrafica estandar
        glMatrixMode(GL_PROJECTION);
        glPushMatrix();
        glLoadIdentity();
        glOrtho(-1,1,-1,1,-10,10);

        // Dibuja un ppligono textura que ocupa toda la ventana
        glMatrixMode(GL_MODELVIEW);
        glPushMatrix();                 // Salva el estado
        glLoadIdentity();
        glPushAttrib(GL_ENABLE_BIT);    // Salva el estado de habilitados
        glDisable(GL_DEPTH_TEST);       // Deshabilita el z-buffer
        glDisable(GL_LIGHTING);         // Deshabilita la iluminacion
        glEnable(GL_TEXTURE_2D);        // Habilita las texturas por si acaso
        glDisable(GL_TEXTURE_GEN_S);    // Deshabilita la generacion automatica por si acaso
        glDisable(GL_TEXTURE_GEN_T);
        glBegin(GL_POLYGON);            // Quad texturado
          glTexCoord2f(0,0);
          glVertex3f(-1,-1,0);
          glTexCoord2f(1,0);
```

```
          glVertex3f(1,-1,0);
          glTexCoord2f(1,1);
          glVertex3f(1,1,0);
          glTexCoord2f(0,1);
          glVertex3f(-1,1,0);
        glEnd();
        glPopMatrix();                  // Restablece la modelview
        glPopAttrib();                  // Restablece lo que hubiera habilitado

        glMatrixMode(GL_PROJECTION);
        glPopMatrix();                  // Restablece la projetion
        glMatrixMode(GL_MODELVIEW);     // Pone la modelview como corriente
}

void display()
// Funcion de atencion al dibujo
{
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);       // Borra la pantalla

        loadBackground();                                       // Carga el fondo

        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();

        gluLookAt(0,0,5,0,0,0,0,1,0);                           // Situa la camara

        glRotatef(girox,1,0,0);                                 // Giro en x
        glRotatef(giroy,0,1,0);                                 // Giro en y
        glScalef(escalado,escalado,escalado);                   // Escalado

        // Brillos separados de la textura
        glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL, GL_SEPARATE_SPECULAR_COLOR);

        // Dibujar el objeto
        glPushMatrix();
        glutSolidTeapot(1.5);
        glPopMatrix();

        glutSwapBuffers();                                      // Intercambia los buffers
}
```