Interacción

Seminario ISGI (S6)

R. Vivó

R. Vivó

Interacción

Seminario ISGI (S6)

Una de las cosas más fascinantes de lo gráficos por computador es la participación del usuario en qué se dibuja y cómo. Es el fundamento del videojuego. En este seminario se explica cómo podemos capturar la intención del usuario utilizando el teclado y el ratón y qué consecuencias tiene en la fase de actualización de la escena. También se explica cómo construir menús sencillos para ampliar la capacidad de elección del usuario sobre la forma en que se dibuja la escena y cómo seleccionar objetos de la escena.

Captura de teclas

A diferencia de como sucede en otras API's, OpenGL se ha construido como una librería gráfica independiente de la interfaz. Por ello necesitamos de otras librerías construidas por encima que traten los eventos de usuario. La librería GLUT sirve para, entre otras cosas y de manera muy sencilla y efectiva, capturar los eventos producidos por el usuario.

La forma de operar es mediante *callbacks*, una para cada tipo de evento. Para capturar la tecla que pulsa el usuario cuando el foco está en nuestra ventana de dibujo, registraremos primero la *callback* `onKey()`:

```
glutKeyboardFunc(onKey)
```

que debemos definir así:

```
void onKey(unsigned char tecla, int x, int y){…}
```

donde se reciben el carácter que corresponde a la tecla pulsada y la posición del cursor en ese momento. Es importante tener en cuenta que la posición del cursor viene referida al extremo **superior izquierdo** de la ventana de dibujo, que es diferente del origen del *viewport* de OpenGL -inferior izquierdo-.

La tecla representada por su carácter ASCII. Si queremos capturar teclas especiales como las de función o las flechas de avance podemos registrar la *callback*:

```
glutSpecialFunc(onSpecialKey)
```

donde la *callback* se define así:

```
void onSpecialKey(int specialKey, int x, int y){…}
```

Ahora la tecla viene representada por un entero que es una constante de GLUT. Entre ellas, por ejemplo, podemos usar `GLUT_KEY_LEFT`, `GLUT_KEY_RIGHT`, `GLUT_KEY_UP` y `GLUT_KEY_DOWN` para identificar las flechas. Para la lista completa mirar la documentación de GLUT.

Captura de la posición del cursor

La parte más interesante es la de capturar la posición del cursor que manejamos con el ratón o cualquier otro dispositivo a cuyo movimiento este asociado. Hay varias formas de capturar este evento. La primera de ellas va asociada al pulsado de los botones del ratón.

Usando botones del ratón

Para capturar este evento registraremos la *callback*:

```
glutMouseFunc(onMouse)
```

donde la *callback* se define así:

```
void onMouse(int button, int state, int x, int y){…}
```

Como parámetros de entrada al atender el evento encontramos `button` que es una de las constantes `GLUT_LEFT_BUTTON`, `GLUT_MIDDLE_BUTTON` o `GLUT_RIGHT_BUTTON` según el evento lo genere el botón izquierdo, central o derecho del ratón respectivamente; `state` que puede tomar los valores `GLUT_UP` o `GLUT_DOWN` según el botón haya sido apretado o liberado; `x` e `y` que son las coordenadas referidas al extremo superior izquierdo del área de dibujo en píxeles en el momento que se hizo la interacción con el botón.

Captura del movimiento del ratón

La siguiente posibilidad es capturar el movimiento del ratón continuamente. Para ello se usan las funciones de registro:

```
glutMotionFunc(onMotion)
```

```
glutPassiveMotionFunc(onPassiveMotion)
```

con las definiciones de las *callback* siguientes:

```
void onMotion(int x, int y){…}
```

```
void onPassiveMotion(int x, int y){…}
```

En ambos casos `x` e `y` son las coordenadas del cursor referidas al extremo superior izquierdo de la ventana en cada momento. La diferencia consiste en si se ha de tener pulsado un botón -`onMotion`- o no es necesario -`onPassiveMotion`-.

Ejercicio S6E01: Dibujar un objeto en modelo de alambres o modelo sólido según se pulsen las teclas "a" o "s". Indicar en todo momento en el título de la ventana la posición del cursor en coordenadas del viewport. Al pulsar la tecla "esc" forzar el cierre de la aplicación.

Manipuladores

Entendemos como manipulador una interfaz para manejar algún aspecto concreto del gráfico tridimensional según las órdenes del usuario de la aplicación. Así, por ejemplo, una **interfaz de inspección** de un objeto es un manipulador que captura los eventos del usuario y los aplica al movimiento de uno o varios objetos de la escena de la misma manera que haríamos si lo tuviéramos en la mano y lo girásemos para ver todos sus lados.

Una **interfaz de edición de objetos** permite, con la ayuda de elementos adicionales sensibles a la selección, cambiar el tamaño, posición u orientación del objeto en la escena. Es lo que hacemos cuando dibujamos un cuadrado en cualquier programa de dibujo, y variamos su forma estirando de algún punto sensible -como un pequeño cuadrado en cada vértice-. Este tipo de interfaz es más complicada pues necesita conocer qué elemento sensible es el que se ha seleccionado con el cursor.

Una de las interfaces más interesantes en 3D es la de **manipulación de la cámara**. Podemos encontrarnos con varios paradigmas de interacción que habitualmente usamos sin darnos cuenta en cualquier videojuego.

El modo de interacción de **bola de cristal o *trackball*** consiste en situar la cámara en una superficie esférica imaginaria mirando hacia el centro de la escena y mover la cámara sobre la superficie para observar la escena desde cualquier punto de vista. Este modo se puede complementar, en proyección perspectiva, con la variación del tamaño de la esfera de manera que los objetos de alejan o se acercan según se aumente o reduzca el radio de la esfera.

Otro modo muy utilizado es el de **movimiento de cabeza**. En este paradigma el observador está en un punto de la escena y mira a su alrededor variando la dirección de observación para inspeccionar el entorno. El punto de vista es fijo, al contrario que el anterior, y lo que cambia es el punto de interés.

Por último, en el **modo de navegación** se permite tanto el movimiento del punto de vista como el de interés. Usamos este modo cuando desplazamos nuestro avatar en los juegos en primera persona. El modo de navegación puede adaptarse al movimiento propio del avatar corriendo, a la conducción de un coche o al vuelo con un avión.

Ejercicio S6E02: Implementar una interfaz de inspección sencilla de una tetera asociando el movimiento del ratón a giros sobre los ejes X e Y cuando se mantenga pulsado el botón izquierdo del ratón. Dibujar unos ejes que no queden afectados por los giros.

Ejercicio S6E03: Implementar la interfaz de bola de cristal sobre la escena anterior.

Menús de contexto

Una manera tradicional de conocer la intención del usuario es la posibilidad de selección de una acción entre un conjunto de ellas. Este modo de interacción es conocido como interfaz por menús.

La librería GLUT integra la posibilidad de crear menús, abrirlos allá donde esté el cursor (menús de pop-up) y devolver el elemento seleccionado por el usuario. Para ello seguiremos la siguiente secuencia:

1. Crear el menú. Para crear el menú hay que registrar una *callback* que lo atienda. Usaremos la función:

   ```
   glutCreateMenu(onMenu)
   ```

   donde `onMenu` es el nombre de la *callback*. Vemos luego cómo definirla. `gluCreateMenu()` devuelve un entero que es el identificador único de nuestro menú.

2. Asociar entradas al menú. Cada entrada es una tira de caracteres a la que se le debe asociar un número entero -diferente del resto para poder distinguirla-. Este número se le pasará a la *callback* cuando el usuario seleccione esta opción del menú. La forma de asociar una entrada es así:

   ```
   glutAddMenuEntry(tira_caracteres,valor_entero)
   ```

   Las entradas del menú se van insertando de arriba hacia abajo.

3. Enganchar el menú a un botón del ratón. Cuando el usuario pulse el botón del ratón aparecerá nuestro menú. Para asociar el menú al botón se usa la función:

   ```
   glutAttachMenu(boton)
   ```

   donde `boton` es una de las constantes `GLUT_LEFT_BUTTON`, `GLUT_RIGHT_BUTTON` o `GLUT_MIDDLE_BUTTON`.

4. Construir la *callback* de atención. Cuando se genere el evento pulsar_botón-selección_entrada-soltar_boton se llamará a la *callback* con el valor entero asociado a la entrada del menú seleccionada. La *callback* se define así:

   ```
   void onMenu(valor_entero){…}
   ```

Desde luego es posible crear menús en cascada y diferentes menús activando el que nos interese. Es fácil seguir la documentación de la GLUT con las indicaciones dadas. No obstante, hay que tener en cuenta que la interfaz que ofrece GLUT para la creación de menús es muy rudimentaria y, aunque útil, debemos pensar en otras librerías específicas de creación de interfaces planas, como QT por ejemplo, para aplicaciones serias o más complicadas.

Ejercicio S6E04: Construir una aplicación que dibuje un cubo, una esfera o una tetera según la opción de menú de pop-up seleccionada. Cada opción de menú abrirá un submenú en cascada que permita seleccionar uno de los tres colores rojo, verde o azul como color de dibujo.

Selección de objetos mediante el ratón (picking)

Una de las acciones más habituales en la interacción con un gráfico es señalar un objeto con el cursor. Con ello seleccionamos uno entre varios sobre el que queremos ejercer alguna acción, por ejemplo hacerlo girar para examinarlo.

La información que nos suministra la interfaz es, además del botón pulsado, las coordenadas del pixel según el sistema de referencia de la ventana de dibujo. Para saber que objeto se proyecta sobre ese pixel en concreto habría que aplicar la trasformación inversa de la cadena de transformaciones modelo-vista-proyección-marco a la visual que pasa por ese pixel y calcular cuál es el objeto que interseca con esa visual más próximo al ojo.

Un sistema más sencillo -y en general más rápido- es hacer dos pasadas de render usando el backbuffer sin presentar el dibujo en la primera y dibujando normalmente en la segunda. En la primera etapa, si cada objeto se dibuja sólido y de un color único, podemos identificar el elegido sin más que mirar el color del pixel donde se encuentra el cursor cuando se pulsó el botón de selección.

El algoritmo es el siguiente:

1. Capturar el evento de selección. Por ejemplo si queremos seleccionar objetos al pulsar el botón izquierdo del ratón registraremos la callback onClick(boton,estado,x,y) con glutMouseFunc(onClick).
2. Pasar x,y a coordenadas del viewport (pick)
3. Dibujar los objetos seleccionables de un color único sin hacer el swap de buffers
4. Leer el color en el buffer para las coordenadas del pick. Para ello usaremos la orden de OpenGL:

   glReadPixels(pickx,picky,1,1,GL_RED,GL_UNSIGNED_BYTE,pixel)

   donde pickx y picky son las cordenadas del pixel en el sistema del marco, 1,1 se refieren a las dimensiones del rectangulo a leer, GL_RED es la constante que indica el canal que se va a leer -en este ejemplo el rojo-, GL_UNSIGNED_BYTE es la constante que indica el tipo de lo que se lee y pixel es el array -en este caso de 1x1- de elementos del tipo anterior que se leen.

5. Identificar el objeto según el color leído
6. Dibujar de nuevo normalmente haciendo el swap de buffers

Con el ejemplo anterior podemos direccionar hasta 256 objetos. Si se necesitan más pueden usarse otros primarios -el verde y el azul-.

Ejercicio S6E05: Dibujar tres objetos separados. Al seleccionar uno de ellos se pone a girar sobre su eje Y hasta que se seleccione otro o se pique sobre el fondo.

Ejercicio S6E01: Dibujar un objeto en modelo de alambres o modelo sólido según se pulsen las teclas "a" o "s". Indicar en todo momento en el título de la ventana la posición del cursor en coordenadas del viewport. Al pulsar la tecla "esc" forzar el cierre de la aplicación.

```
/*************************************************
ISGI::Captura de teclado
Roberto Vivo', 2013 (v1.0)

Captura el pulsado de teclas. Cona "a" dibuja en alambrico
y con "s" en solido. Muestra la posicion del cursor en el
titulo de la ventana.

Dependencias:
+GLUT
*************************************************/
#define PROYECTO "ISGI::S6E01::Catura del teclado"

#include <iostream>                                     // Biblioteca de entrada salida
#include <sstream>                                      // Biblioteca de manejo de strings
#include <cmath>                                        // Biblioteca matematica de C
#include <gl\freeglut.h>                                // Biblioteca grafica
using namespace std;

static enum ModoDibujo {ALAMBRICO,SOLIDO} modo = ALAMBRICO;

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

        glClearColor(1.0,1.0,1.0,1.0);                  // Color de fondo a blanco

        glEnable(GL_DEPTH_TEST);                        // Habilita visibilidad
}

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

  glColor3f(0,0,1);                                     // Color de dibujo a azul
  // Dibuja la tetera segun el modo de dibujo
  if(modo == ALAMBRICO)
          glutWireTeapot(1.0);
  else if(modo == SOLIDO)
          glutSolidTeapot(1.0);                         // Dibuja una tetera en el origen

  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 onKey(unsigned char tecla, int x, int y)
// Funcion de atencion al teclado
{
        stringstream titulo;
        titulo<<x<<","<<y;
        glutSetWindowTitle(titulo.str().c_str());        // Pone el pixel en el titulo

        switch(tecla){
        case 'a':
                modo = ALAMBRICO;
                break;
        case 's':
                modo = SOLIDO;
                break;
        case 27:                                         // Pulso escape
                exit(0);
        }
        glutPostRedisplay();
}

void main(int argc, char** argv)
// Programa principal
{
  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
  std::cout << PROYECTO << " running" << std::endl;      // Mensaje por consola
  glutDisplayFunc(display);                              // Alta de la funcion de atencion a display
  glutReshapeFunc(reshape);                              // Alta de la funcion de atencion a reshape
  glutKeyboardFunc(onKey);                               // Alta de la funcion de atencion al teclado
  init();                                                // Inicializacion propia
  glutMainLoop();                                        // Puesta en marcha del programa
}
```

Ejercicio S6E02: Implementar una interfaz de inspección sencilla de una tetera asociando el movimiento del ratón a giros sobre los ejes X e Y cuando se mantenga pulsado el botón izquierdo del ratón. Dibujar unos ejes que no queden afectados por los giros.

```
/*************************************************
ISGI::Inspeccion de objetos
Roberto Vivo', 2013 (v1.0)

Captura el movimiento del raton para inspeccionar un
objeto de la escena. La inspección se realiza en el
sistema de coordenadas del modelo (GIROS RELATIVOS)

Dependencias:
+GLUT
*************************************************/
#define PROYECTO "ISGI::S6E02::Inspeccion SCM"

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

//Variables globales

static GLuint ejes;                             // Identificador de los ejes
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 init()
// Funcion propia de inicializacion
{
        cout << "Version: OpenGL " <<  glGetString(GL_VERSION) << endl;

        cout << "Inspeccion::Arrastre con boton izquierdo: Gira la pieza" << endl;
        cout << "Inspeccion::Arrastre con boton derecho: Aumenta o disminuye" << endl;

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

        // Crea una lista para dibujar los ejes
        ejes = glGenLists(1);
        glNewList(1,GL_COMPILE);
        glPushAttrib(GL_CURRENT_BIT|GL_LINE_BIT);
        glLineWidth(4.0);
        glBegin(GL_LINES);
        glColor3f(1.0,0.0,0.0);                 // Eje X en rojo
        glVertex3f(0,0,0);
        glVertex3f(1,0,0);
        glColor3f(0,1,0);                       // Eje Y en verde
        glVertex3f(0,0,0);
        glVertex3f(0,1,0);
        glColor3f(0,0,1);                       // Eje Z en azul
        glVertex3f(0,0,0);
        glVertex3f(0,0,1);
        glEnd();
        glColor3f(0.5,0.5,0.5);                 // Origen en gris
        glutWireCube(0.1);
        glPopAttrib();
        glEndList();
}

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
  glCallList(ejes);                                      // Dibuja los ejes fijos
```

```
  // 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
  glColor3f(1,0,1);                                             // Color de dibujo a magenta
  glutWireTeapot(1.0);
  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 aquí
                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 main(int argc, char** argv)
// Programa principal
{
  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
  std::cout << PROYECTO << " running" << std::endl;         // Mensaje por consola
  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
  init();                                                   // Inicializacion propia
  glutMainLoop();                                           // Puesta en marcha del programa
}
```

/ Otra posibilidad es que los giros se acumulen por la izquierda a giros anteriores para que el objeto siempre gire sobre los ejes fijos. Los giros se producen cada vez sobre el objeto ya girado. La matriz MODELVIEW que queremos es MV= VIEW * Ractual * Racumulado * Escalado donde Racumulado(i+1)= Ractual(i)*Racumulado(i). */*

```
…
static float coef[16];                          // Matriz MODELVIEW
…
void init()
// Funcion propia de inicializacion
{
…
        glGetFloatv(GL_MODELVIEW_MATRIX, coef);  // Inicializa coef a la matriz identidad
…
}
void display()
// Funcion de atencion al dibujo
{
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);      // Borra la pantalla
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();

  // Los giros se acumulan por la izquierda sobre el objeto girado, así siempre
  // se gira sobre el sistema fijo. VIEW*R*Racumulado*S

  // Primero calculamos los giros R*Racumulado
  glRotatef(girox,1,0,0);                        // 3.Aplicamos el giro actual en X fijo
  glRotatef(giroy,0,1,0);                        // 2.Aplicamos el giro actual en Y fijo
  glMultMatrixf(coef);                           // 1.Aplicamos el giro anterior
  glGetFloatv(GL_MODELVIEW_MATRIX, coef);        // Nos guardamos la orientacion para la proxima

  // Restablecemos la MODELVIEW y componemos en el orden correcto
  glLoadIdentity();
  gluLookAt(1,2,5,0,0,0,0,1,0);                  // Situa la camara MV=VIEW
  glCallList(ejes);                              // Dibuja los ejes fijos
  glMultMatrixf(coef);                           // Gira el acumulado MV=VIEW*R*Racum
  glScalef(escalado,escalado,escalado);          // Escala el modelo MV=VIEW*R*Racum*S

  glColor3f(1,0,1);                              // Color de dibujo a magenta
  glutWireTeapot(1.0);

  glutSwapBuffers();                             // Intercambia los buffers
}
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
                        girox=0;                // Cuando se escala no se acumula ningún giro
                        giroy=0;
                        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 en la MODELVIEW
                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();
}
```

Ejercicio S6E03: Implementar la interfaz de bola de cristal sobre la escena anterior.

```
/*************************************************
ISGI::Camara Bola de Cristal Dual
Roberto Vivo', 2013 (v1.0)

Captura el movimiento del raton para mover la camara
en la escena sobre una Bola de Cristal. El movimiento
se aplica a los objetos por dualidad, es decir con
signo contrario manteniendo la camara fija.

Dependencias:
+GLUT
*************************************************/
#define PROYECTO "ISGI::S6E03::Bola de Cristal Dual"

#include <iostream>                             // Biblioteca de entrada salida
#include <cmath>                                // Biblioteca matematica de C
#include <gl\freeglut.h>                        // Biblioteca grafica
using namespace std;

//Variables globales

static GLuint ejes;                             // Identificador de los ejes
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 float coef[16];                          // Matriz MODELVIEW
static enum Interaccion {GIRO,ESCALADO} accion; // Tipo de acción de inspección

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: Mueve la camara en la Bola" << endl;
        cout << "Arrastre con boton derecho: Acerca o aleja la camara" << endl;
        cout << "q: Posicion inicial de la camara" << endl;
        cout << "esc: Salir" << endl;

        glClearColor(1.0,1.0,1.0,1.0);          // Color de fondo a blanco
        glEnable(GL_DEPTH_TEST);                // Habilita visibilidad
        glGetFloatv(GL_MODELVIEW_MATRIX, coef); // Inicializa coef a la matriz identidad

        // Crea una lista para dibujar los ejes
        ejes = glGenLists(1);
        glNewList(1,GL_COMPILE);
        glPushAttrib(GL_CURRENT_BIT|GL_LINE_BIT);
        glLineWidth(4.0);
        glBegin(GL_LINES);
        glColor3f(1.0,0.0,0.0);                 // Eje X en rojo
        glVertex3f(0,0,0);
        glVertex3f(1,0,0);
        glColor3f(0,1,0);                       // Eje Y en verde
        glVertex3f(0,0,0);
        glVertex3f(0,1,0);
        glColor3f(0,0,1);                       // Eje Z en azul
        glVertex3f(0,0,0);
        glVertex3f(0,0,1);
        glEnd;
        glColor3f(0.5,0.5,0.5);                 // Origen en gris
        glutWireCube(0.1);
        glPopAttrib();
        glEndList();
}

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

```
  glLoadIdentity();

  // Los giros se acumulan con signo contrario por la izquierda sobre el objeto girado,
  // así siempre se gira sobre el sistema fijo. VIEW*R*Racumulado*zoom

  // Primero calculamos los giros R*Racumulado (signos contrarios)
  glRotatef(-girox,1,0,0);                      // 3.Aplicamos el giro actual en X fijo
  glRotatef(-giroy,0,1,0);                      // 2.Aplicamos el giro actual en Y fijo
  glMultMatrixf(coef);                          // 1.Aplicamos el giro anterior
  glGetFloatv(GL_MODELVIEW_MATRIX, coef);       // Nos guardamos la orientacion para la proxima

  // Restablecemos la MODELVIEW y componemos en el orden correcto
  glLoadIdentity();
  gluLookAt(0,0,5,0,0,0,0,1,0);                 // Situa la camara fija MV=VIEW
  glMultMatrixf(coef);                          // Gira el acumulado MV=VIEW*R*Racum
  glScalef(escalado,escalado,escalado);         // Escala el modelo MV=VIEW*R*Racum*zoom

  // Dibuja la escena
  glCallList(ejes);                             // Dibuja los ejes
  glColor3f(1,0,1);                             // Color de dibujo a magenta
  glutWireTeapot(1.0);

  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 escena
                        accion= GIRO;
                        xantes=x;               // Guarda el valor del pixel picado
                        yantes=y;
                        break;
                case GLUT_RIGHT_BUTTON:         // Escalar la escena
                        accion= ESCALADO;
                        yantes=y;               // La escala se maneja con mvto. vertical del ratón
                        girox=0;                // Cuando se escala no se acumula ningún giro
                        giroy=0;
                        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 en la MODELVIEW
```

```
                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 'q':                                // Vuelve a la posicion original
                girox = 0;
                giroy = 0;
                escalado = 1;
                glPushMatrix();
                glLoadIdentity();
                glGetFloatv(GL_MODELVIEW_MATRIX,coef);   // Inicializa coef a la identidad
                glPopMatrix();
                break;
        case 27:                                 // Salir de la aplicacion
                exit(0);
        }
        glutPostRedisplay();
}

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

Ejercicio S6E04: Construir una aplicación que dibuje un cubo, una esfera o una tetera según la opción de menú de pop-up seleccionada. Cada opción de menú abrirá un submenú en cascada que permita seleccionar uno de los tres colores rojo, verde o azul como color de dibujo.

```
/*************************************************
ISGI::Menus de popup de GLUT
Roberto Vivo', 2013 (v1.0)

Muestra un menu de piezas y colores para seleccionar
al pulsar el boton derecho del raton

Dependencias:
+GLUT
*************************************************/
#define PROYECTO "ISGI::S6E04::Menus GLUT"

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

//Variables globales

static GLuint ejes;                             // Identificador de los ejes
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,NADA} accion;            // Tipo de acción de inspección
static enum Piezas {ESFERA,CUBO,TETERA} pieza = ESFERA;         // Pieza a dibujar
static enum Colores {ROJO,VERDE,AZUL} color;                    // Opcion de color

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 central: Escala la pieza" << endl;
        cout << "Boton derecho: Menu de piezas y colores" << endl;

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

        // Crea una lista para dibujar los ejes
        ejes = glGenLists(1);
        glNewList(1,GL_COMPILE);
        glPushAttrib(GL_CURRENT_BIT|GL_LINE_BIT);
        glLineWidth(4.0);
        glBegin(GL_LINES);
        glColor3f(1.0,0.0,0.0);                         // Eje X en rojo
        glVertex3f(0,0,0);
        glVertex3f(1,0,0);
        glColor3f(0,1,0);                               // Eje Y en verde
        glVertex3f(0,0,0);
        glVertex3f(0,1,0);
        glColor3f(0,0,1);                               // Eje Z en azul
        glVertex3f(0,0,0);
        glVertex3f(0,0,1);
        glEnd();
        glColor3f(0.5,0.5,0.5);                         // Origen en gris
        glutWireCube(0.1);
        glPopAttrib();
        glEndList();
}

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
  glCallList(ejes);                                      // Dibuja los ejes fijos

  // 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

  // Seleccion del color
  if(color == ROJO)
          glColor3f(1,0,0);
  else if(color == VERDE)
          glColor3f(0,1,0);
  else if(color == AZUL)
          glColor3f(0,0,1);

  // Seleccion de la pieza a dibujar
  if(pieza == ESFERA)
        glutWireSphere(1.0,20,20);
  else if(pieza == CUBO)
          glutWireCube(1.0);
  else if(pieza == TETERA)
          glutWireTeapot(1.0);
  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
{
        // Si ha levantado el boton pasamos a hacer NADA
        if(state == GLUT_UP){
                accion = NADA;
                return;
        }

        //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_MIDDLE_BUTTON:        // Escalar la pieza
                        accion= ESCALADO;
                        yantes=y;               // La escala se maneja con mvto. vertical del ratón
                        break;
                case GLUT_RIGHT_BUTTON:         // Por aqui no pasa pues se asocio al menu
```

```
                        accion = NADA;
        }
}

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 aquí
                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:                                  // Acumulación del escalado en "escalado"
                escalado+= (yantes - y) * pix2fac;      // y crece hacia abajo. escalado hacia arriba
                yantes=y;
        case NADA:
                ;
        }
        glutPostRedisplay();
}

void onMenu(int opcion)
// Funcion de atencion al menu de popup
{
        switch(opcion){
        case 0:
                pieza = ESFERA;
                break;
        case 1:
                pieza = CUBO;
                break;
        case 2:
                pieza = TETERA;
                break;
        case 3:
                color = ROJO;
                break;
        case 4:
                color = VERDE;
                break;
        case 5:
                color = AZUL;
        }
        glutPostRedisplay();
}

void initMenu()
// Construye el menu de popup
{
        int menucolores = glutCreateMenu(onMenu);
        glutAddMenuEntry("ROJO",3);
        glutAddMenuEntry("VERDE",4);
        glutAddMenuEntry("AZUL",5);

        glutCreateMenu(onMenu);
        glutAddMenuEntry("ESFERA", 0);
        glutAddMenuEntry("CUBO", 1);
        glutAddMenuEntry("TETERA",2);
        glutAddSubMenu("Color",menucolores);
        glutAttachMenu(GLUT_RIGHT_BUTTON);                      // Ya no pasa por onClick
}

void main(int argc, char** argv)
// Programa principal
{
  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
  initMenu();                            // Construye menus
  init();                                // Inicializacion propia
  glutMainLoop();                        // Puesta en marcha del programa
}
```

Ejercicio S6E05: Dibujar tres objetos separados. Al seleccionar uno de ellos se pone a girar sobre su eje Y hasta que se seleccione otro o se pique sobre el fondo.

```
/*************************************************
ISGI::Seleccion de objetos
Roberto Vivo', 2013 (v1.0)

Permite picar sobre un objeto para ponerlo a girar.

Dependencias:
+GLUT
*************************************************/
#define PROYECTO "ISGI::S6E02::Inspeccion SCM"

#include <iostream>
#include <GL/freeglut.h>
using namespace std;

static GLubyte selected[1];                     // Pixel leído donde se picó
static bool onSelection;                        // Activar/Desactivar la selección
static GLint pickx,picky;                       // Localización del pixel en coordenadas del marco
static float giroT(0),giroS(0),giroC(0);        // Giro particular de cada objeto

void loadPickName(char name)
//Carga un identificativo para la selección
{
        //Simplemente se pinta de rojo 'name' todo lo que se dibuje hasta otra carga de nombre
        //cuando se está en modo selección
        if(onSelection)glColor3ub(name,0x00,0x00);
};

void init()
//Inicializaciones
{
        // Mensajes por consola
        cout << PROYECTO << " running" << endl;
        cout << "Version: OpenGL " <<  glGetString(GL_VERSION) << endl;
        cout << "Selecciona el objeto que gira con el boton izquierdo" << endl;

        glClearColor(1.0,1.0,1.0,1.0);                  // Color de fondo
};
void onDisplay()
{
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();

        gluLookAt(1,2,3,0,0,0,0,1,0);                   // Posiciona la camara


        if(onSelection){                                // Render en backbuffer sin swap
                // Primer objeto
                glPushMatrix();
                glTranslatef(-2,0,0);
                glRotatef(giroT,0,1,0);
                loadPickName('T');
                glutSolidTeapot(0.5);                   // Tetera 'T'
                glPopMatrix();
                // Segundo objeto
                glPushMatrix();
                glRotatef(giroS,0,1,0);
                loadPickName('S');                      // Esfera 'S'
                glutSolidSphere(0.5,20,20);
                glPopMatrix();
                // Tercer objeto
                glPushMatrix();
                glTranslatef(1.5,0,0);
                glRotatef(giroC,0,1,0);
                loadPickName('C');                      // Cubo 'C'
                glutSolidCube(0.5);
                glPopMatrix();
```

```
                //Se lee el canal rojo de un rectangulo de 1x1 pixel en pickx,picky (DC)
                glReadPixels(pickx,picky,1,1,GL_RED,GL_UNSIGNED_BYTE, selected);

                onSelection= false;
        }
        else{                                                   // Render normal
                // Primer objeto
                glPushMatrix();
                glTranslatef(-2,0,0);
                glRotatef(giroT,0,1,0);
                glColor3f(1,0,0);
                glutWireTeapot(0.5);
                glPopMatrix();
                // Segundo objeto
                glPushMatrix();
                glRotatef(giroS,0,1,0);
                glColor3f(0,1,0);
                glutWireSphere(0.5,20,20);
                glPopMatrix();
                // Tercer objeto
                glPushMatrix();
                glTranslatef(1.5,0,0);
                glRotatef(giroC,0,1,0);
                glColor3f(0,0,1);
                glutWireCube(0.5);
                glPopMatrix();

                glutSwapBuffers();                              // Intercambia los buffers
        }
}

void onReshape(int w, int 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)
//Callback de respuesta a presionar el botón del ratón
{

        if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN){     //Botón izquierdo abajo
                GLint vport[4];
                onSelection= true;                                //Se activa el modo selección
                glGetIntegerv(GL_VIEWPORT, vport);                //Recupera el viewport corriente
                pickx=x; picky=vport[3]-y;                        //Coordenadas en el sistema del marco
                glutPostRedisplay();                              //Evento para procesar la selección
        }
}

void onIdle()
// Funcion de atencion al evento idle
{
        switch((char) selected[0]){
        case 'T':
                giroT += 0.1;
                glutPostRedisplay();
                break;
        case 'S':
                giroS += 0.1;
                glutPostRedisplay();
                break;
```

```
        case 'C':
                giroC += 0.1;
                glutPostRedisplay();
        }
}

void main(int argc, char** argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_RGB|GLUT_DOUBLE|GLUT_DEPTH);
        glutInitWindowSize(400,300);
        glutInitWindowPosition(0,0);
        glutCreateWindow(PROYECTO);
        glutReshapeFunc(onReshape);
        glutDisplayFunc(onDisplay);
        glutMouseFunc(onClick);
        glutIdleFunc(onIdle);
        init();

        glutMainLoop();
};
```