Dpto. de Lenguajes y Sistemas Informáticos
Escuela Técnica Superior de Ingenierías Informática y Telecomunicación

Práctica 5
de Informática Gráfica
Grupo A

Autores: Francisco J. Melero

Curso 2018/19

Capítulo 1

Interacción

1.1. Objetivos

El objetivo de esta práctica es:

- Aprender a desarrollar aplicaciones gráficas interactivas, gestionando los eventos de entrada de ratón para mover la cámara.
- Aprender a realizar operaciones de selección de objetos en la escena.
- Crear una cámara con proyección paralela y realizar zoom con la misma.

1.2. Desarrollo

Partiendo de las prácticas anteriores, se añadirá la siguiente funcionalidad:

1. Colocar dos cámaras en la escena, una ortogonal y otra perspectiva.
2. Mover la cámara usando el ratón en modo *examinar*, haciendo zoom.
3. Seleccionar objetos de la escena usando el ratón.

1.2.1. Colocar varias cámaras en la escena

Se añadirá al código un vector de cámaras (objetos de una clase `Camara`), y se incluirá en la escena un mecanismo de control para saber qué cámara de las existentes está activa. Al menos se habrán de colocar dos cámaras, una de ellas deberá tener proyección ortogonal y otra proyección en perspectiva. Se activarán con las teclas F1, y F2.

1.2.2. Mover la cámara usando el ratón y el teclado en modo *examinar*

Con las teclas de flecha se mueve la cámara activa en torno al origen de coordenadas. Se añadirá el código para que los dos grados de libertad en el movimiento del ratón consigan el mismo efecto que pulsar las flechas.

Con la rueda del ratón el sistema realizará zoom, independientemente de que se encuentre en una cámara perspectiva u ortogonal. Para controlar la cámara con el ratón es necesario hacer que los cambios de posición del ratón afecten a la posición de la cámara, y en *glut* eso se hace indicando las funciones que queremos que procesen los eventos de ratón (en el programa principal antes de la llamada a *glutMainLoop*();):

```
glutMouseFunc( clickRaton );
glutMotionFunc( ratonMovido );
```

y declarar estas funciones en el código.

La función *clickRaton* será llamada cuando se actúe sobre algún botón del ratón. La función *ratonMovido* cuando se mueva el ratón manteniendo pulsado algún botón.

El cambio de orientación de la cámara activa se gestionará en cada llamada a *ratonMovido*, que solo recibe la posición del cursor, por tanto debemos comprobar el estado de los botones del ratón cada vez que se llama a *clickRaton*, determinando que se puede comenzar a mover la cámara sólo cuando se ha pulsado el botón derecho. La información de los botones se recibe de *glut* cuando se llama al callback:

```
void clickRaton( int boton, int estado, int x, int y );
```

por tanto bastará con analizar los valores de *boton* y *estado*, y almacenar información que nos permita saber si el botón derecho está pulsado y la posición en la que se encontraba el cursor cuando se pulsó

```
if ( boton == GLUT_RIGHT_BUTTON )
{  if ( estado == GLUT_DOWN )
   {
          // Se pulsa el bot\'on, por lo que se entra en el
              // estado \"moviendo c\'amara\"
   }
   else {
      // Se levanta el bot\'on, por lo que se sale
                    //del estado \"moviendo c\'amara\"
       }
  glutPostRedisplay();
}
```

En la función *ratonMovido* comprobaremos si el botón derecho está pulsado, en cuyo caso actualizaremos la posición de la cámara a partir del desplazamiento del cursor

```
void Escena::ratonMovido( int x, int y )
{   ......
    ......
    if ( estadoRaton==MOVIENDO_CAMARA_FIRSTPERSON)
    {
        camaras[camaraActiva].girar(x-xant,y-yant);
                            xant=x;yant;=;
   }
}
```

En el método `Camara::girar`, cada cámara recalcula el valor de sus parámetros de giro en X y en Y (`observer_angle_x` y `observer_angle_y`).

Si hasta ahora en la práctica la transformación de visualización se hacía, por ejemplo, en

```
void change_observer()
{
    // posicion del observador
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0,0,-Observer_distance);
    glRotatef(Observer_angle_x,1,0,0);
    glRotatef(Observer_angle_y,0,1,0);
}
```

ahora habrá que hacer

```
void Escena::change_observer()
{
    // posicion del observador
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    camaras[camaraActiva].setObservador();
}
```

de forma que en cada posicionamiento de la cámara se invoque la cámara con los parámetros adecuados.

1.2.3. Seleccionar objetos de la escena usando el ratón.

Se incluirán en la escena los objetos generados en las prácticas 2, 3 y 4, y cualquier otro objeto que se desee. El usuario, al hacer clic con el botón izquierdo del ratón sobre uno de los objetos de la escena, transformará el objeto a color (o material, si la luz está encendida) amarillo. Volviendo a hacer clic sobre el objeto, recuperará su apariencia original.

Selección por codificación de colores

Para determinar qué objeto ha sido seleccionado habremos de usar un código de color para cada objeto seleccionable. Se trata de crear una función de dibujado distinta para cuando queremos seleccionar, y cuando el usuario hace clic con el botón secundario, se pinta la escena "para seleccionar" en el buffer trasero y se lee el color del pixel donde el usuario ha hecho clic. Si no se hace un intercambio de buffers, el usuario jamás verá esa escena "rara", y el programa seguirá su proceso natural.

En resumen, los pasos a seguir son:

- Llamar a la función o método dibujaSeleccion()
- Leer el pixel (x,y) dado por la función gestora del evento de ratón
- Averiguar a qué objeto hemos asignado el color de dicho pixel
- **No intercambiar buffers**

Un código de ejemplo, que dibuja cuatro patos cada uno de un color sería:

```
void Escena::dibujaSeleccion() {

glDisable(GL_DITHER); // deshabilita el degradado
for(int i = 0; i < 2; i++)
   for(int j = 0; j < 2; j++) {
        glPushMatrix();
        switch (i*2+j) { // Un color para cada pato
                case 0: glColor3ub(255,0,0);break;
                case 1: glColor3ub(0,255,0);break;
                case 2: glColor3ub(0,0,255);break;
                case 3: glColor3ub(250,0,250);break;
        }
        glTranslatef(i*3.0,0,-j * 3.0);
        pato.dibuja();
        glPopMatrix();
 }
glEnable(GL_DITHER);
}
```

Para comprobar el color del pixel, usaremos la función glReadPixels:

```
void glReadPixels(GLint x, GLint y, GLsizei width, GLsizei height,
                  GLenum format, GLenum type, GLvoid *pixels);
```

donde

- x,y : la esquina inferior izquierda del cuadrado a leer (en nuestro caso el x,y, del pick)
- width,height: ancho y alto del área a leer (1,1 en nuestro caso)
- format: Tipo de dato a leer (coincide con el format del buffer, GL_RGB o GL_RGBA).
- type: tipo de dato almacenado en cada pixel, según hayamos definido el glColor (p.ej. GL_UNSIGNED_BYTE de 0 a 255, o GL_FLOAT de 0.0 a 1.0)
- pixels: El array donde guardaremos los pixels que leamos. Es el resultado de la función.

En el caso del procesamiento del pick, una vez dibujado el buffer, llamaríamos a un método o función que, en función del color del pixel nos miraría en la tabla de asignación de colores a objetos qué objeto estaríamos seleccionando.

Para que esto funcione, hay varias cosas a tener en cuenta:

- Los colores se han de definir con glColor3ub, es decir, como enteros de 0 a 255
- Es posible que el monitor no esté en modo trueColor y no devuelva exactamente el valor que pusimos (hay que tenerlo en cuenta si no funciona bien).
- Hay que desactivar el GL_DITHER, GL_LIGHTING, GL_TEXTURE

1.3. Evaluación

La evaluación de la práctica, sobre 10 puntos, se hará del modo siguiente:

- Posicionar dos cámaras (2 punto)
- Gestión con ratón el movimiento de la cámara (3 puntos)
- Zoom de las cámaras (2 puntos)
- Selección de objetos (3 puntos)

1.4. Extensiones

Se podrá realizar la gestión de la cámara, en lugar de con los dos grados de libertad, con una clase cámara más completa:

```
class Camara {
 _vertex3f  eye;
 _vertex3f at;
 _vertex3f up;

 int tipo; // ORTOGONAL o Perspectiva
 float left,right,near,far; // o bien aspect, fov, near, far;

 Camara( ... ) ; // con los parametros necesarios
 rotarXExaminar(float angle);
 rotarYExaminar(float angle);
 rotarZExaminar(float angle);
 rotarXFirstPerson(float angle);
 rotarYFirstPerson(float angle);
 rotarYFirstPerson(float angle);
 mover(float x, float y, float z);
 zoom(float factor);

 setObserver()  { gluLookAt( ..... ) }; // completar
 setProyeccion();

}
```

1.5. Duración

La práctica se realizará durante **dos** sesiones.

1.6. Bibliografía

- Mark Segal y Kurt Akeley;*The OpenGL Graphics System: A Specification (version 4.1)*; http://www.opengl.org/
- Edward Angel;*Interactive Computer Graphics. A top-down approach with OpenGl*; Addison-Wesley, 2000
- J. Foley, A. van Dam, S. Feiner y J. F. Hughes; *Computer Graphics: Principles And Practice, 2 Edition*; Addison-Wesley,1992
- M. E. Mortenson; *Geometric Modeling*; John Wiley & Sons,1985

- http://www.lighthouse3d.com/opengl/picking/index.php