Práctica 5

Interacción

5.1. Objetivos

El objetivo de esta práctica es:

- Aprender a desarrollar aplicaciones gráficas interactivas, gestionando los eventos de entrada de ratón
- Aprender a realizar operaciones de selección de objetos en la escena.
- Afianzar los conocimientos de los parámetros de la cámara para su correcta ubicación y orientación en la escena.

5.2. Desarrollo

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

1. Colocar varias cámaras en la escena
2. Mover la cámara usando el ratón y el teclado en modo *primera persona*.
3. Seleccionar objetos de la escena usando el ratón de forma que la cámara pase a funcionar en modo *examinar*.

Adicionalmente, como funcionalidad extra, se podrá implementar:

1. Una cámara en *tercera persona* que siga un objeto en movimiento (aleatorio o guiado por teclado) en la escena.
2. la función de zoom para una cámara con una proyección ortogonal

5.2.1. Colocar varias cámaras en la escena

Se añadirá al código un vector de cámaras (objetos de una clase `Camara` si se está programando en C++), 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 tres cámaras, ofreciendo las vistas clásicas de frente, alzado y perfil de la escena completa. Al menos una de ellas deberá tener proyección ortogonal y al menos otra proyección en perspectiva. Se activarán con las teclas F1, F2 y F3.

5.2.2. Mover la cámara usando el ratón y el teclado en modo *primera persona*

Con las teclas A,S,D y W se moverá la cámara activa por la escena (W: avanzar, S: retroceder, A: desplazamiento a la izquierda, D: desplazamiento a la derecha, R: reiniciar posición), conservando la dirección en la que se está mirando. Para ello será necesario modificar únicamente el punto VRP o *eye*.

Por otro lado, girar la cámara a derecha o izquierda, arriba o abajo (modificar el VPN o el *lookAt* se realizará siguiendo los movimientos del ratón con el botón derecho pulsado.

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 botón, por lo que se entra en el estado "moviendo cámara"
   }
   else{
// Se levanta el botón, por lo que se sale del estado "moviendo cámara"
   }
}
```

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

En el método `Camara::girar`, cada cámara recalcula el valor de sus parámetros (VPN o `lookAt`) en función del incremento de x e y recibido.

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

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

5.2.3. Seleccionar objetos de la escena usando el ratón de forma que la cámara pase a funcionar en modo *examinar*

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, centrará el foco de atención sobre el centro de dicho objeto (calculado como el centro de su caja envolvente). A partir de ese momento, la cámara entrará en modo EXAMINAR, y el movimiento del ratón con el botón derecho pulsado permitirá observar el objeto seleccionado desde cualquier ángulo.

Movimiento de la cámara en modo EXAMINAR

En el método `Camara::girar`, cuando se está en modo examinar, la cámara recalcula el valor de sus parámetros (VRP o `eye`) en función del incremento de x e y recibido, de forma que orbite en torno al punto `lookAt`, que es el centro del objeto seleccionado.

En el modo EXAMINAR, las teclas A,S,D y W no tienen efecto sobre la cámara, salvo que se implemente la funcionalidad extra de zoom, que se realizará con las teclas W y S.

Selección

Para seleccionar se debe crear una función de selección (*pick*). Hay dos formas de hacerlo: usando el modo GL_SELECT de OpenGL, o usando una codificación de colores en el buffer trasero.

Selección modo GL_SELECT

Cuando se pulse el botón izquierdo desde la función *clickRaton* se debe llamar a una función que gestione el `pick`:

```
int pick( int x, int y)
```

siendo, x, y la posición del cursor que se va a usar para realizar la selección, y devuelve el ID del objeto que se ha seleccionado (-1 si no se ha seleccionado nada).

Para poder seleccionar los distintos componentes de la escena se deben añadir identificadores (*names*) al dibujarlos.

El procedimiento de selección (*pick*), debe realizar los siguientes pasos:

```
// 1. Declarar buffer de selección
      glSelectBuffer(...)
// 2. Obtener los parámetros del viewport
      glGetIntegerv (GL_VIEWPORT, viewport)
// 3. Pasar OpenGL a modo selección
      glRenderMode (GL_SELECT)
// 4. Fijar la transformación de proyección para la seleccion
      glMatrixMode (GL_PROJECTION);
      glLoadIdentity ();
      gluPickMatrix (x,(viewport[3] - y),5.0, 5.0, viewport);
      MatrizProyeccion(); // SIN REALIZAR LoadIdentity !
// 5. Dibujar la escena con Nombres
      dibujarConNombres();
// 6. Pasar OpenGL a modo render
      hits = glRenderMode (GL_RENDER);
// 7. Restablecer la transformación de proyección (sin gluPickMatrix)
// 8. Analizar el contenido del buffer de selección
// 9. Devolver el resultado
```

Interpretación del buffer de selección

En el buffer devuelve, para cada primitiva intersecada (la variable `hits` devuelta al hacer `glRenderMode(GL_RENDER)` nos dice el número de primitivas intersecadas con el clic del ratón):

- el número de identificadores o nombres asociados a la primitiva
- el intervalo de profundidades de la primitiva: zmin y zmax.
- los nombres asociados a la primitiva.

Se deberá crear un método en la escena o una función que gestione el buffer de nombres, y devuelva el identificador del objeto más cercano al observador.

Nombres

Para distinguir un objeto de otro, OpenGL utiliza una pila de enteros como identificadores. Dos primitivas se considera que corresponden a objetos distintos cuando el contenido de la pila de nombres es distinto. Para ello proporciona las siguientes funciones:

```
glLoadName(i) // Sustituye el nombre activo por i
glPushName(i) // Apila el nombre i
glPopName() // Desapila un nombre
glInitNames() // Vacia la pila de nombres
```

El método o función `dibujarConNombres` difiere del dibujar normal en que hace uso de la pila de nombres, por lo que es el que se invoca en el modo GL_SELECT. Antes de ponerte a codificar, debes decidir cómo colocar los identificadores y añadirlos a los objetos, teniendo en cuenta lo que necesitas seleccionar.

Procura gestionar bien la pila de nombres, asegurándote de que está vacia al comienzo del ciclo de dibujo y que los `glPushName` y `glPopName` están balanceados

Selección por codificación de colores

Hay un mecanismo más simple aún para determinar qué primitiva ha sido seleccionada. Se trata 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, 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

5.3. Evaluación

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

- Posicionar varias cámaras (2 punto)
- Gestión con ratón y teclado de cámara en primera persona (3 puntos)
- Selección de objeto (2 puntos)
- Gestión con ratón de cámara en modo examinar (3 puntos)

5.4. Extensiones

Se podrá realizar una animación de un objeto (p.ej. un animal que se mueva por el suelo) y se le pondrá una cámara que le realice un seguimiento desde atrás, lo que viene a ser una cámara en *tercera persona*. El movimiento del objeto podrá ser automático o controlado por teclado. En este caso, tanto la posición `eye` como `lookAt` vienen determinadas, de forma directa o indirecta, por el objeto que se persigue.

Como debe haber al menos una cámara en modo ortogonal, se podrá implementar la función zoom para dicha cámara, pues sabemos que la imagen resultante es independiente de la distancia del observador al plano de proyección. Para ello habrá que modificar los parámetros del frustum.

5.5. Duración

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

5.6. 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