Informática Gráfica

Juan Carlos Torres

Curso 2024/25

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Disclaimer
You can edit this page to suit your needs. For instance, here we have a no copyright statement, a colophon and some other information. This page is based on the corresponding page of Ken Arroyo Ohori's thesis, with minimal changes.

Colophon
This document was typeset with the help of KOMA-Script and LaTeX using the kaobook class.

Sistemas interactivos: Selección 7

7.1 Subsistema de entrada . . 58
7.2 Interacción con GLUT . . 59
7.3 Interacción con Unity3D 61
7.4 Selección 62
7.4.1 Selección en OpenGL . . 64
7.4.2 Selección en Unity 67
7.5 Ejercicios 67

Un sistema gráfico interactivo es un conjunto de hardware y software que permite a los usuarios crear, manipular y visualizar modelos, respondiendo de manera fluida a las entradas del usuario. Estos sistemas permiten la interacción entre el usuario y la escena, adaptándose de manera inmediata a cambios como el movimiento de la cámara, la modificación de objetos o la selección de opciones a través de dispositivos de entrada.

En la primera lección se describieron algunos ejemplos típicos de estos sistemas.

Para que un sistema sea interactivo, su respuesta debe ser lo suficientemente rápida para que el usuario perciba la relación causa-efecto de sus acciones sobre los cambios en el modelo. Esto no implica un límite de tiempo de respuesta estricto, como en los sistemas de tiempo real. La **latencia** es el tiempo que transcurre entre la acción del usuario y la actualización de la imagen o la respuesta en el sistema de realidad virtual. Por tanto, en un sistema interactivo debe haber una latencia baja[1].

1: En un sistema interactivo puede haber operaciones específicas que no sean interactivas

Los sistemas de realidad virtual son los que requieren la latencia más baja posible. Una latencia baja es crucial para que la experiencia de realidad virtual sea inmersiva y cómoda. Latencias altas pueden generar efectos negativos como mareos o incomodidad, ya que el cerebro percibe el desajuste entre los movimientos y la actualización visual. Idealmente, la latencia en un sistema de realidad virtual debería ser inferior a 20 milisegundos para garantizar una experiencia fluida.

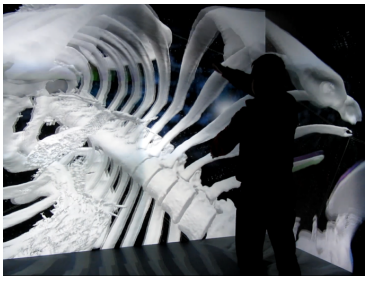

Figura 7.1: Interacción gestual con Kinect en un sistema de realidad virtual.

Un aspecto esencial en el diseño de sistemas interactivos es ofrecer retroalimentación como respuesta a las acciones realizadas. Normalmente, esta información es la visualización de la escena modificada (como consecuencia del movimiento de un personaje o la creación de un objeto). En algunos casos se puede utilizar retroalimentación no gráfica (como vibración o sonido) o retroalimentación visual adicional a la visualización de la escena.

La realimentación ($feedback$) es el mecanismo que permite al sistema brindar información relevante al usuario para decidir la siguiente acción. La información de realimentación generada por el sistema depende del estado actual y de la información previamente ingresada por el usuario. Se puede utilizar con diferentes fines: mostrar el estado del sistema, como parte de una función de entrada o para reducir la incertidumbre del usuario. Por ejemplo, al dibujar líneas, una vez introducido el primer punto, se puede mostrar la línea que se crearía en la posición actual del cursor mientras éste se mueve.

Con frecuencia, el usuario debe ingresar una posición que cumpla con una determinada condición, como tener la misma coordenada X que la posición previa. En estas situaciones, la aplicación puede ayudar al usuario proporcionando {técnicas de interacción} que aseguren el cumplimiento de las restricciones de entrada. En el caso anterior, se puede hacer que el cursor solo se mueva en horizontal. Por ejemplo, al

crear elementos en un escenario, el sistema puede ajustar su altitud para que se apoyen en el elemento que esté en esa posición.

Nos ocuparemos en esta lección del subsistema de interacción, centrándonos en las operaciones de selección.

7.1. Subsistema de entrada

Los componentes básicos de un sistema gráfico interactivo son (figura 7.2):

Hardware

- Dispositivos de visualización: Se utilizan para mostrar la escena. Suelen ser monitores, pero también pueden ser otros equipos, como sistemas de proyección o gafas de realidad virtual.
- Dispositivos de entrada: Permiten la interacción del usuario y pueden incluir dispositivos más allá del ratón y teclado, como controladores táctiles, sensores de posición, dispositivos hápticos u otros dispositivos (figura 7.1).

Software

- Modelo geométrico: Contiene la representación de la escena, que podrá ser modificada según las operaciones de entrada recibidas.
- Motor de renderizado: Genera las imágenes a partir del modelo geométrico.
- Subsistema de interacción: Se encarga de interpretar y procesar las acciones realizadas por el usuario.

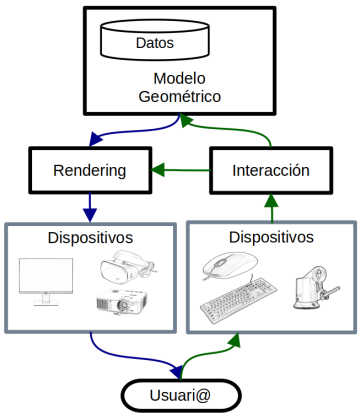

Figura 7.2: Arquitectura de un sistema gráfico interactivo.

Cuando se actúa sobre un dispositivo de entrada el subsistema de interacción interpreta la acción, envía al modelo geométrico las ordenes correspondientes y si es necesario envía al motor de rendering la petición para incluir información de realimentación.

En este proceso es importante conseguir que el código de la aplicación sea independiente del modelo y tipo de dispositivo concreto utilizado. Para conseguirlo se independiza la capa de software que accede directamente a los dispositivos de forma que la aplicación a modelos abstractos de los dispositivos (**dispositivos lógios**). Esto permite por ejemplo que una aplicación pueda leer posiciones de pantalla con diferentes dispositivos, incluso con dispositivos que no generan una posición como información de salida.

Idealmente las APIS gráficas pueden definir los siguientes dispositivos lógicos:

Locator: Devuelve una posición.
Pick: Devuelve el identificador de un objeto.
Stroke: Devuelve una lista de posiciones.
String: Devuelve una cadena de caracteres.
Choice: Devuelve uno de entre n items.
Valuator: Devuelve un número.

La sincronización entre los dispositivos y la aplicación puede realizarse de varios modos. La aplicación puede realizar una **petición** al dispositivo y quedar suspendida hasta que se introduzca la información. Este es el modo de funcionamiento de las operaciones de lectura de un programa. Por ejemplo la sentencia

```
cin >> x;
```

suspenderá el programa hasta que se introduzca un valor. Este modo de funcionamiento es adecuado cuando el programa no puede continuar su ejecución hasta que se realice la lectura. En un sistema interactivo se usa para acciones concretas que necesitan que deben realizarse para que el programa pueda continuar, por ejemplo dar el nombre del archivo que contiene el modelo que se va a cargar.

En algunos casos el dispositivo puede estar realizando una medida de una variable (por ejemplo la altura de agua de un depósito) y este valor puede ser consultado por el programa cuando lo necesite. En este modo de funcionamiento (denominado **sample**) el programa no se suspende ya que el dato está siempre disponible. Este modo es útil cuando la aplicación debe muestrear una variable continua registrada por un dispositivo.

7.2. Interacción con GLUT

La mayor parte de las operaciones en los sistemas interactivos se realizan en modo **evento**. En este modo la acción sobre el dispositivo genera un evento que es capturado por el subsistema de entrada.

La aplicación registra las funciones (**callback**) que se deben ejecutar como respuesta a los eventos que necesita procesar.

El sistema de gestión de eventos (activado al llamar a glutMainLoop) que se encarga de encolar los eventos que se producen para los que hay un callback registrado y de llamar a dichas funciones para dar respuesta a los eventos (figura 7.3).

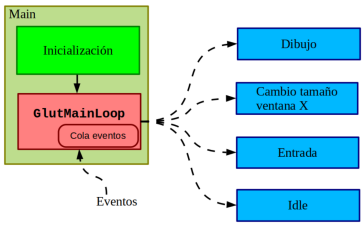

Figura 7.3: Estructura típica de un programa OpenGL con glut. Los rectangulos azules represetan callbacks.

Una vez arrancada la aplicación el flujo de ejecución de la aplicación queda gestionado por GLUT que irá detectando los eventos producidos (tanto externos como internos) y llamando a los diferentes callbacks cada vez que suceda un evento [Kil96]. El sistema de gestión de eventos de GLUT se activa con la llamada a $glutMainLoop()$.

Los principales eventos registrados por GLUT responden a entradas de dispositivos (teclado, ratón, spaceball), modificación de la ventana gráfica o eventos internos (dibujo, temporizador, función de fondo). Para algunos dispositivos hay mas de un evento asociado.

Para cada evento hay una función de GLUT encargada de registrar el callback que lo va a procesar. Las funciones de registro de callback de ratón son las siguientes:

glutMouseFunc: eventos de cambio de estado de los botones del ratón.
glutMotionFunc: eventos de desplazamiento con algún botón pulsado.
glutPassiveMotionFunc: eventos de desplazamiento sin botones pulsados.

La interfaz de los callback debe ser la siguiente[2]:

2: La interfaz está fijada ya que las llama GLUT.

```
MouseFunc(int button, int state, int x, int y);
MotionFunc(int x, int y);
PassiveMotionFunc(int x, int y);
```

El parámetro *button* recibe el código del botón sobre el que se ha actuado (GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, o GLUT_-RIGHT_BUTTON); *state* el estado actual del botón (*GLUT_UP* o *GLUT_DOWN*), x e y reciben la posición del cursor en coordenadas de la ventana.

Con este modo de funcionamiento no se puede predecir el orden en el que se van a ejecutar las funciones de los distintos callback, por lo que no se puede presuponer cual va a ser el estado de la aplicación cuando se está procesando un evento. Así, por ejemplo, no podemos dibujar una linea de realimentación desde un callback de ratón, porque no se sabe en que estado se encuentra el proceso de dibujo. En este caso será necesario pasar información indicando la línea a dibujar a la función que dibuja, este paso se puede realizar con variables compartidas o con llamadas a funciones del módulo de dibujo.

El siguiente fragmento de código muestra como utilizar los callback de ratón para girar la cámara:

```
1  int MOUSE_LEFT_DOWN=0;
2  int MOUSE_X, MOUSE_Y;
3
4  void clickRaton( int boton, int estado, int x, int y )
5  {
6    if(boton==GLUT_LEFT_BUTTON && estado==GLUT_DOWN) {
7      MOUSE_LEFT_DOWN=1;
8      MOUSE_X=x;
9      MOUSE_Y=y;
10   }
11   else if(boton==GLUT_LEFT_BUTTON && estado==GLUT_UP) {
12     MOUSE_LEFT_DOWN=0;
13     ISINTERACTING=0;
14   }
15   glutPostRedisplay();
16 }
17
18 void RatonMovido( int x, int y )
19 {
20   float ax,ay,az,d;
21   getCamara (ax, ay, az, d);
22   if(MOUSE_LEFT_DOWN) {
23     if(x!=MOUSE_X) ay+=x-MOUSE_X;
24     if(y!=MOUSE_Y) ax+=y-MOUSE_Y;
25     setCamara (ax, ay, az,  d);
26     MOUSE_X=x;
27     MOUSE_Y=y;
28     glutPostRedisplay();
29   }
30 }
```

La variable *MOUSE_LEFT_DOWN* se utiliza para saber si es el botón derecho el que está pulsado cuando se produce un movimiento de ratón. Las variables *MOUSE_X* y *MOUSE*$_Y$ para recordar la última posición del ratón y calcular con el desplazamiento el desplazamiento a aplicar a la

Estas funciones se deberán registrar como callback con las siguientes llamadas:

```
glutMouseFunc (clickRaton);
glutMotionFunc (RatonMovido);
```

cámara. Las funciones $getCamara$ y $setCamara$ acceden a los ángulos de giro de la cámara y su distancia.

7.3. Interacción con Unity3D

Unity escucha continuamente los dispositivos de entrada conectados al sistema (teclados, ratones, gamepads, pantallas táctiles, etc.). Cada vez que ocurre un evento de entrada (por ejemplo, presionar una tecla, mover el ratón, tocar la pantalla), Unity lo detecta y lo almacena en un sistema de entrada interno. Unity no gestiona los eventos de entrada en tiempo real, sino que acumula el estado de los dispositivos de entrada entre cada frame.

La interacción se programa en scripts asociados a los componentes de la escena, que pueden tener un método $update$ que se ejecuta en cada frame. Para detectar las entradas del usuario, los scripts utilizan la clase $Input$ que ofrece métodos como $Input.GetKeyDown$ (para ver si una tecla esta pulsada), Input.GetMouseButtonUp, y Input.GetAxis, que ayudan a consultar el estado de diferentes tipos de entrada en cualquier momento durante la ejecución del juego.

En cada ciclo de frame, Unity procesa los cambios en los estados de entrada (por ejemplo, si una tecla fue presionada, liberada o si un botón del ratón fue pulsado) y actualiza los estados que se pueden consultar mediante la clase $Input$.

Para procesar las entradas de ratón se pueden usar las siguientes funciones de $Input$:

Input.GetMouseButtonDown(int button): Devuelve true en el frame en el que se presiona un botón del ratón.
Input.GetMouseButton(int button): Devuelve true mientras el botón del ratón está pulsado.
Input.GetMouseButtonUp(int button): Devuelve true en el frame en el que se libera el botón del ratón.
Input.mousePosition: Devuelve la posición del cursor del ratón en la pantalla, en coordenadas de ventana.

La clase $Input$ implementa un dispositivo lógico $GetAxis$ que representa un desplazamiento horizontal o vertical del cursor sobre el que se puede actuar desde diferentes dispositivos.

Además, los script asociados a los objetos pueden también consultar la situación del ratón respecto al objeto:

OnMouseDown: se ha presionado el botón del ratón mientras el puntero está sobre el objeto.
OnMouseUp: se ha liberado el botón del ratón cuando el puntero está sobre el objeto.
OnMouseEnter: El puntero del ratón ha entrado en el área del objeto.
OnMouseExit: Se llama cuando el puntero del ratón sale del área del objeto.
OnMouseOver: Se llama cada frame mientras el puntero del ratón está sobre el objeto.

El siguiente script gira una cámara orbital:

```
using UnityEngine;

public class CameraController : MonoBehaviour
{
  public float mouseSensitivity = 100.0f;  // Sensibilidad del
    raton
  public float distanceFromOrigin = 10.0f; // Distancia de la
    camara al origen
  private float xRotation = 0.0f;
  private float yRotation = 0.0f;

  void Start()
  {
  }

  void Update()
  {
    // Obtener la entrada del raton
    float mouseX = Input.GetAxis("Mouse X") * mouseSensitivity *
    Time.deltaTime;
    float mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity *
    Time.deltaTime;
    // Acumular las rotaciones alrededor de los ejes X e Y
    xRotation -= mouseY;  // Movimiento vertical
    yRotation += mouseX;  // Movimiento horizontal
    // Limitar la rotacion vertical a 90 grados
    xRotation = Mathf.Clamp(xRotation, -90.0f, 90.0f);
    // Rotar la camara alrededor del origen en el eje horizontal
    transform.RotateAround(Vector3.zero, Vector3.up, mouseX *
    mouseSensitivity * Time.deltaTime);
    // Rotar en el eje vertical (xRotation)
    transform.RotateAround(Vector3.zero, transform.right, -mouseY
    * mouseSensitivity * Time.deltaTime);
    // Asegurarnos de que la camara siempre mire al origen
    transform.LookAt(Vector3.zero);
    }
}
```

7.4. Selección

La operación de selección permite indicar interactivamente un componente de la escena. Esta operación es esencial en cualquier sistema gráfico, ya que es necesaria para realizar cambios en la escena de forma interactiva, para por ejemplo: eliminar componentes, cambiar sus atributos, copiarlos o aplicarles transformaciones geométricas.

Para poder realizar la selección (*pick*) los componentes de la escena deben tener asociados identificadores. Por ejemplo, para poder seleccionar triángulos, cada uno debe tener asociado un identificador, en este caso puede ser un entero. En otras casos el identificador puede estar compuestos por varios items,por ejemplo en un escena en la que aparecen diferentes tipos de objetos articulados el identificador puede ser un par de enteros: (objeto, segmento del objeto).

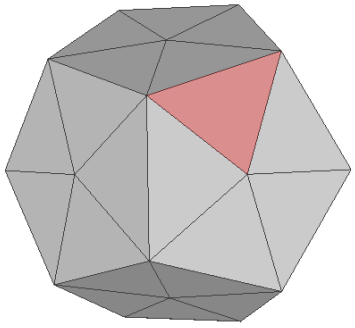

Figura 7.4: Selección de un triángulo.

Los identificadores se pueden asociar a distintos niveles, permitiendo controlar el **ámbito** de la selección. En una misma aplicación podemos

necesitar seleccionar poliedros, triángulos, aristas o vértices para realizar diferentes operaciones.

El proceso que se realiza para seleccionar un componente de la escena implica:

1. El usuario hace click en una posición de pantalla.
2. Se busca el componente (en el ámbito seleccionado) que se corresponde con esa posición de pantalla.
3. El sistema muestra (como información de feedback) el objeto seleccionado (figura 7.4).

En algunos casos no es necesario dar información de realimentación.

Dependiendo del método de selección empleado, la correspondencia puede implicar que el componente sea el que se dibuje en ese pixel, que se proyecte en el pixel, o que se dibuje o proyecte próximo al pixel.

La búsqueda se puede realizar de varios modos. Los mas usados son: Intersección rayo escena y Codificando el id de objeto como color.

Intersección rayo escena

Para determinar que componente se debe seleccionar se puede construir la semirecta que se proyecta en el pixel partiendo de la cámara (a estas semirectas se les suele llamar **rayos**). Para construirla se calcula la posición 3D del pixel en el plano de recortado delantero[3].

Se utiliza este rayo para calcular su intersección con todos los componentes de la escena[4]. El cálculo de la intersección dependerá de la geometría del componente, es diferente calcular la intersección con una esfera, con un triángulo, un paralelepípedo o un triángulo, por citar algunos ejemplos.

A modo de ejemplo, para calcular la intersección de un rayo con una esfera (figura 7.5) se debe encontrar si hay algún que cumpla simultaneamente la ecuación del rayo, que en forma paramétrica es:

$$\mathbf{r}(t) = \mathbf{O} + t \cdot \mathbf{d} \tag{7.1}$$

donde $\mathbf{O}$ es la posición de la cámara, $\mathbf{d}$ el vector director de la recta y $t \in (0, \infty)$ el parámetro que permite recorrer la recta.

La ecuación de la esfera es:

$$(\mathbf{p} - \mathbf{C})^2 - R^2 = 0 \tag{7.2}$$

donde $\mathbf{C}$ es el centro de la esfera y R es su radio.

Para resolver la intersección debemos encontrar puntos que cumplan ambas ecuaciones:

$$(\mathbf{O} + t \cdot \mathbf{d} - \mathbf{C})^2 - R^2 = 0 \tag{7.3}$$

Que es una ecuación de segundo grado, cuya solución es:

3: Y se utiliza la ecuación de la recta que pasa por dos puntos.

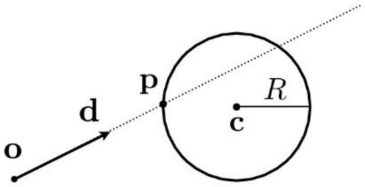

Figura 7.5: Intersección rayo-esfera.

4: Aquí lo que se considere un componente dependerá del ámbito de la selección.

$$
\begin{aligned}
& t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \\
& con \\
& a = d \cdot d \\
& b = 2(\mathbf{O} - \mathbf{C}) \cdot d \\
& c = (\mathbf{O} - \mathbf{C}) \cdot (\mathbf{O} - \mathbf{C}) - R^2
\end{aligned}
\tag{7.4}
$$

Por tanto habrá intersección si $(b^2 - 4ac) > 0$. Ene este caso podremos calcular el punto de intersección.

Si hay varios objetos intersectados podremos decidir quedarnos con el visible, que es el que está mas cerca de la cámara (menor valor de t).

Codificando el id de objeto como color

Para realizar la selección se dibuja la escena haciendo que como colores se usen los identificadores de los componentes. De esta forma, una vez dibujada la escena, podemos leer el valor del pixel en que se encuentra el cursor. La figura 7.7 corresponde a la visualización de identificadores de la escena de la figura 7.6.

En este proceso es necesario que no se modifique el color asignado a cada objeto, para ello se deben desactivar el cálculo de iluminación y el difuminado. También es esencial asignar los colores como valores enteros para evitar que se produzcan errores por redondeo.

Otro aspecto esencial es hacer que esta imagen de selección no se muestre al usuario. Esto se puede conseguir dibujando la imagen en un frame oculto o no intercambiando los buffers de imagen tras la selección.

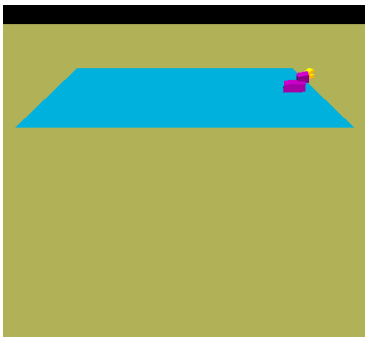

Figura 7.6: Escena de respresentada con ilunibnación.

Figura 7.7: Imagen generada con identificadores como colores correspondiente a la escena de la figura 7.6.

7.4.1. Selección en OpenGL

OpenGL disponía de un modo específico de selección (basado en recortar la imagen en tornos al cursor) que ha quedado obsoleto a partir de la versión 3.0. La forma mas simple de realizar una selección en las versiones actuales es Codificando el id de objeto como color.

Para realizar la selección por color se debe dibujar la escena cuando se quiera realizar la selección llamando a la función de dibujo desactivando la iluminación y el dithering. Para garantizar que se dibuja la misma escena es conveniente tener una sola versión del método de dibujo. Para ello se puede crear una función que dibuje la escena, y hacer que el callback de dibujo la llame después de activar la iluminación y el dithering:

```
1 void dibujar()
2 {
3   glEnable(GL_LIGHTING);
4   glEnable(GL_DITHER);
5   glClearColor(0,0,0,1);   // Fija el color de fondo
6   dibujoEscena();
7   glutSwapBuffers();
8 }
```

En este caso la función *dibujar* será la que se active como callback:

```
glutDisplayFunc( dibujar );
```

El método *dibujaescena* será el encargado de renderizar la escena, tanto para visualizarla como para seleccionar. Por tanto deberá asignar tanto los materiales usados al renderizar como los colores que codifican los identificadores para la selección [5].

5: Con iluminación activada se usarán los materiales y con iluminación desactivada los colores.

El siguiente fragmento de código muestra un fragmento de coódigo de ejemplo:

```
void dibujoEscena()
{
  glClear( GL_COLOR_BUFFER_BIT  | GL_DEPTH_BUFFER_BIT );
  ...
  glLightfv( GL_LIGHT0, GL_POSITION, pos );
  glTranslatef(-5, 3,-5);
  // Dibuja elemento no seleccionable
  glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, gris );
  dibujarFondo();
  // Dibuja elementos seleccionables compuestos
  for(i=0;i<n;++i)
    glMaterialfv( GL_FRONT, GL_AMBIENT_AND_DIFFUSE,color);
    glPushMatrix();
      glTranslatef(pato[i].x, 0,pato[i].z);
      ColorSeleccion( i, 0);
      dibujaCuerpo();
      ...
      glPushMatrix();
        glMaterialfv( GL_FRONT, GL_AMBIENT_AND_DIFFUSE,naranja);
        ColorSeleccion( i, 1);
        dibujaParte1();
        ...
     glPopMatrix();
   glPopMatrix();
}
```

En este ejemplo asumimos que tenemos n objetos con la misma estructura. Cada objeto tiene un cuerpo y varias partes que queremos que se selecciones independientemente. A cada elemento seleccionable se le asignan dos identificadores, el primero es el número de objeto (i), el segundo es el número de la parte del objeto.

La codificación de los colores (que en el ejemplo anterior se realiza en *ColorSeleccio*) se debe realizar dando los colores como enteros, para evitar errores de redondeo. Si no se tienen mas de 255 objetos y estos no tiene mas de 255 partes, la función *ColorSeleccio* del ejemplo anterior podría ser:

```
void ColorSeleccion( int i, int Parte)
{
  unsigned char r = (i & 0xFF);
  unsigned char g = (Parte & 0xFF);
  glColor4ub(r,g,0,0);
}
```

En este caso, al no necesitar mas de 255 valores podemos dejar sin usar los canales verde y alfa.

La asignación de color se realiza con la función glColor4ub que recibe cuatro $ubnsignedbytes$ para evitar que se produzcan errores por redondeo.

La selección se iniciará con un evento de entrada, que deberá realizar la selección e interpretar la información obtenida. Por ejemplo, si la selección se realiza al pulsar el botón izquierdo del ratón, el código del callback podría ser:

```
1  void clickRaton( int boton, int estado, int x, int y )
2  {
3    int parte=-1, i=-1,k=0;
4    if(boton== GLUT_LEFT_BUTTON && estado == GLUT_DOWN) {
5    // Se ha actuado sobre el izquierdo que esta ahora pulsado
6    k=pick(x,y,&i,&parte);
7    if(k>-1) { // se ha seleccionado algo
8      switch (parte) {
9        case 0:
10       ...
11   }
12 }
```

En el que se ha creado el código de selección en la función $pick$ que usa como parámetros la posición del cursor (x, y) y el identificador encontrado, devolviendo un valor negativo si no se han leído identificadores. La función pick es el núcleo del proceso de selección:

```
1  int pick(int x, int y,int * i, int * parte)
2  {
3    GLint viewport[4];
4    unsigned char data[4];
5    int f=0;
6    glGetIntegerv (GL_VIEWPORT, viewport);
7    glDisable(GL_DITHER);
8    glDisable(GL_LIGHTING);
9    glClearColor(0,0,1,1);   // Fija el color de fondo a negro
10   dibujoEscena();
11   glFlush();
12   glReadPixels(x, viewport[3] - y,1,1, GL_RGBA, GL_UNSIGNED_BYTE,
       data);
13   *i=data[0];
14   *parte=data[1];
15   if(data[2]>0) f=-1;
16   glEnable(GL_LIGHTING);
17   glEnable(GL_DITHER);
18   return f
```

Las tres primeras líneas (3,4 y 5 declaran variables para almacenar el tamaño de la zona de dibujo ($vierport$) el color que se va a leer de la imagen de selección y el valor a devolver indicando si se ha seleccionado algún elemento. La línea 6 obtiene las dimensiones del área de dibujo, que se necesita para poder invertir la coordenada Y.

A continuación se desactiva el dithering y la iluminación y se fija el color de fondo del buffer de imagen (que puede ser distinto del que se va a usar para renderizar la escena). Este color debe ser diferente de los usados para codificar los identificadores de los objetos. En este ejemplo se está asignado color al canal verde que no se usa en los identificadores.

A continuación se dibuja la escena (línea 10). Se realiza una llamada a $glFlush$, que fuerza a que se realicen todas las funciones de OpenGL que se han enviado hasta ese momento y se lee el color del pixel que se encuentra en la posición del cursor. Los argumentos de esta función son: la posición x,y del cursor, el ancho y alto del área de pixels que se leen, el formato y el buffer en el que se van a leer.

Las líneas siguientes (13, 14 y 15) decodifican el identificador y determinan si se ha pulsado algún objeto seleccionable.

Por últinmo se activa la iluminación y el dithering (esto es necesario solo si no se han incluido en el callback de dibujo).

7.4.2. Selección en Unity

La selección en Unity se puede realizar con el método $OnMouseDown$ o calculando la intersección rayo-objeto con el método $Raycast$[6]. El siguiente código realiza la selección usando el $Raycast$:

6: Es necesario que el objeto tenga asociado un colisionador.

```
1  void Update()
2  {
3    if (Input.GetMouseButtonDown(0)){
4      // Devuelve verdadero si se ha pulsado el boton en este frame
5      RaycastHit hitInfo = new RaycastHit();
6      bool hit = Physics.Raycast(Camera.main.ScreenPointToRay(Input.
       mousePosition), out hitInfo); //Devuelve verdadero si el rayo
        intersecta el collider de un objeto
7      if (hit) {
8        Selected = hitInfo.transform.gameObject; // Objeto
       seleccionado
9        Debug.Log("Hit " + Selected.name); // Muestra el nombre del
       objeto en consola
10     }
11   }
12 }
```

7.5. Ejercicios

1. Dar un ejemplo de situaciones en las que sea preferible utilizar cada uno de los modos de entrada.
2. ¿Es posible implementarla lectura de cadenas de caracteres, utilizando como dispositivo físico un ratón? ¿Porqué / cómo?
3. ¿Es posible crear funciones de entrada en modo evento en un sistema gráfico que tenga solo entrada en modo muestreo?
4. Supongamos un sistema gráfico que dispone tan sólo de entrada en modo evento. ¿Es posible crear una librería de funciones que, utilizando las funciones de entrada de dicho sistema, implemente entrada en modo sample?
5. Realizar una implementación en pseudocodigo de una función de lectura de posición del cursos en modo muestreo para glut.
6. Redactar un algoritmo que permita editar un rectángulo, utilizando dos puntos para seleccionar la operación a realizar. El punto central del borde inferior debe permitir cambiar la posición y la esquina superior derecha el tamaño. Al pulsar el botón izquierdo estando

sobre alguno de los puntos de selección, se entra en el proceso de edición, del que se sale al soltarlo.

7. Implementar en C el procedimiento que permitiría realizar un resaltado, cambiando el color de las aristas del objeto poligonal sobre el que se hace click.
8. Suponiendo el modelo articulado del ejercicio 12 implementado en OpenGL explica detalladamente que tendrías que programar para permitir que se seleccionasen los diferentes tramos de la escalera.
9. Como se debe modificar el callback de ratón del ejercicio anterior para que se puedan modificar los grados de libertad interactivamente cuando se selecciona un tramo, desplazando el ratón mientras el botón izquierdo siga pulsado.
10. Como se debe modificar la codificación del identificador en el ejercicio anterior si queremos tener varias escaleras.
11. Se tiene una escena compuesta por M objetos representados por mallas cada una con N triángulos. Las mallas están almacenadas usando la siguiente clase Malla:

```
typedef struct{
  float x,y,z;
} punto;

typedef struct{
  int v0,v1,v2;
} cara;

class Malla : public Objeto3D
{
  public:
  std::vector<punto> v;  // vertices
  std::vector<cara>  c;  // caras
  std::vector<punto> nv; // normales de vertice
  std::vector<punto> nc; // normales de cara
    ...
  void draw(){
    float color[4] = { 0.5, 1.0, 0.5, 1 };
    glShadeModel(GL_FLAT);
    glMaterialfv (GL_FRONT, GL_AMBIENT_AND_DIFFUSE, color);
    glBegin(GL_TRIANGLES);
    for(int i =0; i<caras.size();i++){
      glNormal3f(nc[i].x,nc[i].y,nc[i].z);
      glVertex3f(v[c[i].v0].x,v[c[i].v0].y,v[c[i].v0].z);
      glVertex3f(v[c[i].v1].x,v[c[i].v1].y,v[c[i].v1].z);
      glVertex3f(v[c[i].v2].x,v[c[i].v2].y,v[c[i].v2].z);
    }
    glEnd();
  }
};
```

Explica como se pueden asignar y codificar los identificadores para que usando el método de selección por color se obtenga el número de objeto y el número de triángulo en la malla.

12. Con la misma representación del ejercicio anterior explica que se debe cambiar para poder seleccionar aristas y vértices.
13. En el ejercicio anterior ¿Como se puede conseguir que se seleccionen los vértices cuando se haga click en una zona próxima al vértice?.
14. ¿Hay alguna limitación al tamaño del modelo impuesta por la

forma de asignar los identificadores? ¿Se puede conseguir que no haya límites al tamaño del modelo?