

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.

CC BY-NC-SA

 This book is released into the public domain using the CC BY-NC-SA. This license enables reusers to distribute, remix, adapt, and build upon the material in any medium or format for noncommercial purposes only, and only so long as attribution is given to the creator. If you remix, adapt, or build upon the material, you must license the modified material under identical terms.

To view a copy of the CC BY-NC-SA code, visit:

<https://creativecommons.org/licenses/by-nc-sa/4.0/>

Colophon

This document was typeset with the help of KOMA-Script and L^AT_EX using the kaobook class.

8

Lectura de posiciones

Una posición es una coordenada. Una forma simple de leer coordenadas 3D es utilizar las operaciones de lectura del lenguaje para leer tres números reales. Por ejemplo en C++:

```
std::cin >> x >> y >> z;
```

Esto no nos sirve en un sistema gráfico interactivo ya que se realiza en modo petición (deteniendo la ejecución del programa) y porque obliga a que la persona que usa el sistema tenga que saber las coordenadas que quiere introducir.

Imaginemos que creamos un juego de la tablero (figura 8.1). Cuando se quiera mover una ficha el usuario tendrá que indicar las coordenadas de la posición en la que quiere dejarla (aunque en el caso de un juego de casillas se podría indicar el número de la casilla).

Introducir posiciones no solo es esencial para poder crear y editar objetos, también hace falta para introducir transformaciones geométricas y parámetros de las cámaras y luces.

8.1. Dispositivos de posicionamiento

Existe una gran variedad de dispositivos que permiten introducir posiciones. Podemos agruparlos en los siguientes grupos en función de la información que generan:

Desplazamientos 2D: Dispositivos que devuelven el desplazamiento realizado sobre una superficie, como el ratón. El sistema de entrada puede generar una posición 2D en pantalla integrando estos desplazamiento y usando como información de realimentación un cursor.

Desplazamientos 3D: Dispositivos que devuelven desplazamientos en 3 dimensiones como el spaceMouse (figura 8.3).

Posiciones 2D: Devuelve una posición en un plano o en pantalla. Ejemplos de este grupo son las pantallas táctiles, los lápiz ópticos y las tabletas digitalizadoras (figura 8.2).

Posiciones 3D: Devuelve una posición en el espacio. Este es el caso los brazos de medición, los dispositivos de seguimiento (trackers) y los dispositivos hapticos.

Gestos: Devuelven los gestos realizados. Estos sistemas reconocen todo o parte del cuerpo y permiten entregar información de los gestos realizados. Por ejemplo la Kinect reconoce todo el cuerpo y los dispositivos Leap Motion identifican gestos realizados con la mano (figura 8.4).

El software del sistema usado para gestionar la interacción muestra abstracciones de estos dispositivos que permiten que los programas puedan funcionar con diferentes tipos de dispositivos. En algunos casos

8.1 Dispositivos de posicionamiento	70
8.2 Métodos de posicionamiento	71
8.3 Lectura en OpenGL	74
8.4 Lectura en Unity	75
8.5 Entrada de transformaciones geométricas	76
8.6 Ejercicios	77



Figura 8.1: En un juego de tablero la forma natural de interactuar es apuntar a la posición en la que queremos dejar la ficha.



Figura 8.2: Tableta digitalizadora.



Figura 8.3: SpaceMouse. Devuelve desplazamientos 3D y rotaciones 3D.

incluso haciendo que la información obtenida sea diferente de la generada por el programa. Por ejemplo, los ratones generan desplazamientos, pero el callback de ratón de glut devuelve una posición 2D.

Una característica importante de los dispositivos de entrada es el número de variables independientes (grados de libertad) que permiten controlar. Se suele identificar con las siglas *DF* abreviatura de la denominación en inglés (Degree of Freedom). El dispositivo de la figura 8.3 tiene *6DF*.

8.2. Métodos de posicionamiento

Los métodos que se pueden usar para leer posiciones dependerán de si se está trabajando con modelos 2D o 3D y de los dispositivos de entrada disponibles. Nos vamos a centrar en lectura de posiciones utilizando dispositivos 2D, que generan directa o indirectamente una posición en pantalla.

En el proceso de lectura el usuario indicará que va a introducir una posición (por ejemplo usando una opción de un menú para introducir un objeto en el escenario, o pulsando un botón del ratón) e irá desplazando el cursor hasta colocarlo en la posición deseada. Por último confirmará la posición pulsando un botón. Como respuesta a estas acciones el programa debe inicializar la lectura de posiciones (esto puede implicar por ejemplo mostrar una imagen diferente para cursor del ratón), e irá actualizando la posición del cursor con los desplazamientos que realiza el usuario, hasta que este indique el fin de la operación. Al finalizar se deberá calcular la posición introducida en coordenadas de la escena (si no se ha estado haciendo en el ciclo de seguimiento) y actualizar el modelo. La figura 8.5 muestra la correspondencia de las acciones del usuario y de la aplicación.

En el proceso que realiza la aplicación intervienen tres eventos: El inicio de la lectura se realiza como respuesta a un evento *glutMouseFunc*, cada iteración en el ciclo de seguimiento responde a un evento *glutMotionFunc* y el fin de la entrada a otro evento *glutMouseFunc* como se muestra en la figura 8.6. El siguiente fragmento es un ejemplo de lectura de posición en una aplicación 2D que introduce líneas, el dibujo comienza con la pulsación del botón izquierdo y termina cuando se libera el botón izquierdo. En el ciclo de seguimiento se dibuja una linea elástica como información de realimentación.

```

1 int dibujaLinea=0;
2 int linea[4];
3
4 void clickRaton( int boton, int estado, int x, int y )
{
6   if(boton==GLUT_LEFT_BUTTON && estado==GLUT_DOWN) {
7     linea[0]=x;
8     linea[1]=y;
9     linea[2]=x;
10    linea[3]=y;
11    dibujaLinea=1;
12  }
13  else if(boton==GLUT_LEFT_BUTTON && estado==GLUT_UP) {
14    dibujaLinea=0;
}

```



Figura 8.4: Interacción con gestos de la mano usando Leap Motion.

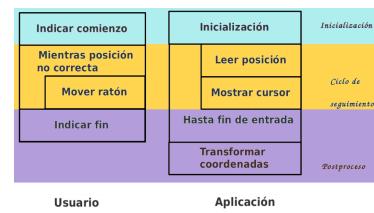


Figura 8.5: Correspondencia entre acciones del usuario y de la aplicación en una lectura de posición.

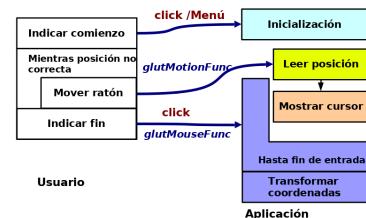


Figura 8.6: Estructura del proceso de lectura en callback.

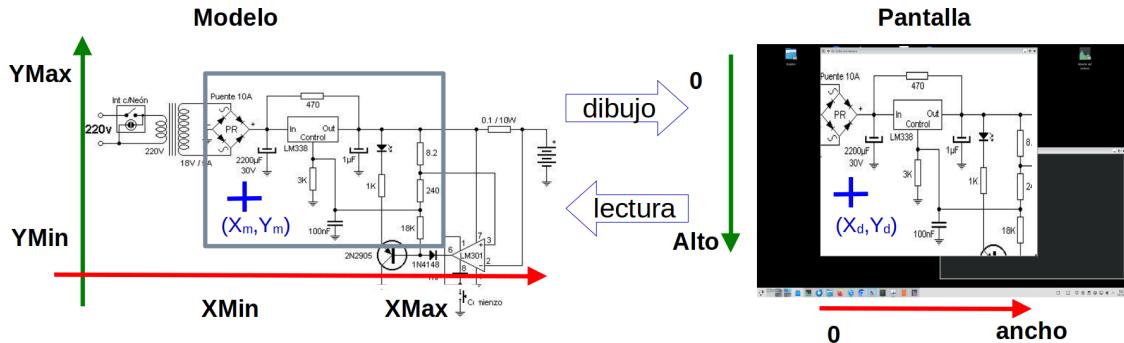


Figura 8.7: Conversión entre coordenadas de la escena y de pantalla en una aplicación 2D.

```

15     addLine(); // Crea la linea en el modelo.
16 }
17 glutPostRedisplay();
18 }

20 void RatonMovido( int x, int y )
21 {
22     if(dibujaLinea) {
23         linea[2]=x;
24         linea[3]=y;
25         glutPostRedisplay();
26     }
27 }

29 Dibuja()
30 {
31     ...
32     if(dibujaLinea) {
33         glBegin (GL_LINES);
34         glColor3f (1, 1, 0);
35         glVertex2f (linea[0],linea[1]);
36         glVertex2f (linea[2],linea[3]);
37         glEnd();
38     }
39     ....
40 }
```

Entrada de posiciones 2D

En una aplicación 2D (por ejemplo para el dibujo de planos de circuitos), el modelo geométrico está definido en el plano. Lo que se muestra en pantalla es una zona rectangular del modelo (figura 8.7). En este caso, la conversión entre coordenadas de la escena y coordenadas de pantalla se realiza con un escalado y dos traslaciones. Estas transformaciones se pueden invertir, por lo que conociendo el área del modelo que se está visualizando (*ventana*) y el área de la pantalla en la que se dibuja (*vireport*) se puede calcular la transformación de coordenadas de pantalla (x_d, y_d) a coordenadas de mundo (x_m, y_m) :

$$\begin{aligned} X_m &= X_{min} + X_d * (X_{max} - X_{min}) / \text{ancho} \\ Y_m &= Y_{max} - Y_d * (Y_{max} - Y_{min}) / \text{alto} \end{aligned} \quad (8.1)$$

Entrada de posiciones 3D

En una aplicación 3D en cada pixel de pantalla se proyecta una semirecta, por lo que no es posible invertir la transformación. Así, en la figura 8.8 en el punto marcado en la imagen se proyecta toda la semirecta naranja. Si se está utilizando un dispositivo de entrada 2D es necesario dar información adicional para determinar la profundidad a la que se está indicando el punto.

La determinación del punto 3D puede realizarse con diferentes técnicas. Revisaremos algunas de ellas.

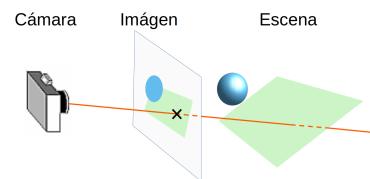


Figura 8.8: Conversión entre coordenadas de la escena y de pantalla en una aplicación 3D.

Utilización de tres vistas

Una de las formas más simples de introducir una posición 3D es descomponerla en varios posicionamientos en 2D. Para ello se puede visualizar la escena como tres vistas ortogonales (sobre los planos XY, YZ y ZX) como se muestra en la figura 8.9. La posición del cursor se muestra en las tres vistas de forma que el usuario modificar dos de las coordenadas en cada una de estas vistas.

Cursor 3D

Otra alternativa es dar como información de realimentación una visualización del cursor que permita percibir su posición en el espacio. Esto puede hacerse visualizando la sombra arrojada por el cursor sobre los objetos de la escena, activando la eliminación de partes ocultas para el cursor o mostrando su proyección sobre los planos del sistema de coordenadas (figura 8.10).

Si se utiliza un dispositivo 2D (como un ratón) para interaccionar será necesario utilizar un botón para seleccionar los ejes sobre los que actúa el dispositivo, por ejemplo moviendo la X y la Y con el botón izquierdo y la Z y la Y con el derecho.

Restricción a una superficie: intersección rayo escena

Vimos en la lección anterior como seleccionar objetos calculando la intersección de un rayo que pasa por el centro de proyección de la cámara y el punto correspondiente a un pixel en el plano de recortado delantero. Se puede utilizar el mismo mecanismo para seleccionar una posición sobre la superficie de un objeto. En este caso en lugar de devolver el identificador del objeto mas cercano con el que se produce la intersección se devolverán las coordenadas del punto de intersección.

Este procedimiento se puede usar para obtener posiciones restringidas a una superficie aunque la superficie no se está dibujando. Por ejemplo

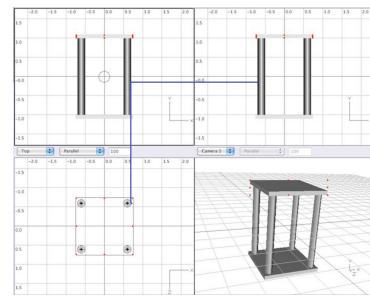


Figura 8.9: Entrada de una posición en una aplicación 3D usando tres vistas ortogonales.

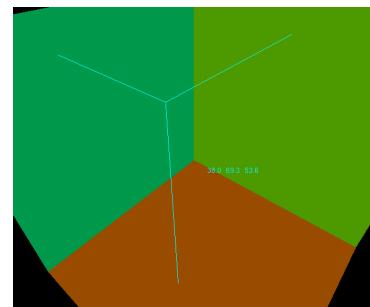


Figura 8.10: Visualización de un cursor 3D mostrando la proyección sobre los planos del sistema de coordenadas.

podemos obtener posiciones en el plano $Y = 0$ calculando la intersección con este plano, que no tiene porque existir en la escena.

Restricción a una superficie: selección de punto

También vimos que se pueden seleccionar componentes codificando su identificador como colores. Podemos realizar un proceso similar para obtener una posición, utilizando como color la posición.

Hay algunas diferencias entre el uso para selección y para posicionamiento. En posicionamiento necesitamos introducir los colores con valores reales, usando una componente del color para cada coordenada. Además necesitamos que se interpolen los colores dentro de los triángulos, por lo que deberemos utilizar sombreado de vértices.

La representación normalizada del color con valores de componentes entre 0 y 1, obliga a normalizar las coordenadas en la caja envolvente de la escena:

$$\begin{aligned}x' &= (x - X_{min}) / (X_{max} - X_{min}) \\y' &= (y - Y_{min}) / (Y_{max} - Y_{min}) \\z' &= (z - Z_{min}) / (Z_{max} - Z_{min})\end{aligned}\quad (8.2)$$

Indirectamente: Selección de objeto

Podemos usar una operación de selección para identificar un objeto y tomar como posición de entrada un punto preestablecido del mismo (por ejemplo el centro de su caja envolvente).

8.3. Lectura en OpenGL

Para leer una posición con OpenGL podemos usar cualquiera de los métodos anteriores.

Para utilizar varias zonas de dibujo se puede hacer uso de las funciones de creación de ventanas de glut:

```
int ventana, subventana;
ventana = glutCreateWindow ("titulo");
subventana = glutCreateSubWindow (ventana,x1,y1,ancho,alto);
```

La primera crea una ventana en el monitor devolviendo el identificador asignado. La segunda crea una subventana en la ventana que se indica comenzando en el pixel $(x1, y1)$. La subventana tendrá dimension ancho x alto, mientras que ventana será una zona de dibujo que ocupará el fragmento restante de la ventana original.

Cada zona de dibujo tendrá su propio contexto para OpenGL: sus propios callback, sus menús y su estado. Cuando necesitemos cambiar la zona de dibujo utilizaremos la función:

```
glutSetWindow( identificadorDeVentana );
```

Es necesario tener en cuenta que los eventos de redibujado se generan para la ventana activa, por lo que si queremos redibujarlas todas tendremos que activarlas una a una y generar el evento de redibujado en cada una de ellas.

Para crear una visualización como la de la figura 8.9 deberemos crear cuatro subventanas en la misma ventana y utilizar proyecciones diferentes para cada una de ellas¹.

Para utilizar un cursor 3D tendremos que hacer que el desplazamiento del ratón afecte a diferentes coordenadas en función del botón que se haya pulsado.

Para construir un rayo se puede utilizar la función *gluUnProject* que calcula un punto en el espacio de la escena en el rayo que pasa por un pixel. Con este punto y el centro de proyección (la posición (0, 0, 0)) podemos calcular la ecuación de la recta que contiene el rayo.

La función *gluUnProject* toma la posición en la pantalla de un punto, la profundidad, las matrices de transformación de modelo y de proyección, el viewport, y devuelve las coordenadas desproyectadas en el espacio del mundo (X_m, Y_m, Z_m). La interfaz de la función es:

```
int gluUnProject(double X, double Y, double Z,
    double *modelMatrix, double *projMatrix, int *viewport,
    double *Xm, double *Ym, double *Zm);
```

Donde (X, Y) es la coordenada de pantalla en pixeles (con la Y invertida), Z es la profundidad en espacio del ZBuffer, que es cero en el plano de recortado delantero y 1 en el trasero.

La profundidad de un pixel se puede leer con la función *glReadPixels*:

```
glReadPixels(X, Y, 1, 1, GL_DEPTH_COMPONENT, GL_FLOAT, &Z);
```

en la que (X, Y) es la posición del pixel, con la Y invertida.

1: Veremos como definir las proyecciones en la lección siguiente

8.4. Lectura en Unity

La forma mas simple de obtener una posición 3D en la escena de manera interactiva utilizando el ratón en Unity 3D es usando la clase *Ray* que permite generar un rayo desde la cámara y la posición del cursor en la pantalla hacia el espacio 3D de la escena. Podemos calcular la intersección del rayo con los objetos de la escena usando el método *Raycast*. El siguiente código muestra como obtener posiciones sobre un plano (no visible) colocado a una altura *planeHeight*. La posición obtenida se encuentra en la variable *hitPosition*:

```
using UnityEngine;
public class RaycastToPlane : MonoBehaviour
{
    public float planeHeight = 0f; // Altura del suelo
    void Update()
    {
        // Detectar si se ha hecho clic con el botón izquierdo
        if (Input.GetMouseButtonDown(0))
        {
            // Obtener la posición del cursor en la pantalla
```

```
Vector3 mousePos = Input.mousePosition;
// Convertir la posicion del cursor en un rayo
Ray ray = Camera.main.ScreenPointToRay(mousePos);
// Crear un plano virtual (plano XZ) a altura planeHeight
Plane groundPlane = new Plane(Vector3.up, new Vector3(0,
planeHeight, 0));
float distance; // Distancia desde el origen del rayo a
interseccion
// Realizar la interseccion del rayo con el plano
if (groundPlane.Raycast(ray, out distance))
{
    // Calcular el punto de interseccion en el espacio 3D
    Vector3 hitPosition = ray.GetPoint(distance);
    Debug.Log("Posicion 3D seleccionada: " + hitPosition);
    ...
}
}
```

8.5. Entrada de transformaciones geométricas

Las transformaciones geométricas se pueden definir a partir de pares de puntos, un punto inicial y un punto final. De esta forma podemos usar los métodos de entrada de posiciones para indicar transformaciones geométricas de forma interactiva.

Una traslación se puede definir mediante el vector que va de la posición original a la posición nueva (figura 8.11). El vector de traslación será:

$$\mathbf{P}_t = \mathbf{P}_2 - \mathbf{P}_1 \quad (8.3)$$

Si el cálculo del vector de traslación se realiza en el ciclo de seguimiento se puede mostrar el objeto trasladado como información de realimentación.

Para especificar una rotación mediante dos puntos en dos dimensiones calculamos el ángulo giro como el ángulo formado por los vectores que va del origen de coordenadas a los dos puntos (figura 8.12). En una rotación 3D se debe especificar en primer lugar el eje de giro. Una vez dado este, el ángulo se puede indicar mediante dos puntos en el plano perpendicular al eje.

De forma análoga se puede indicar el factor de escala realizando el cociente de las coordenadas de los dos puntos (figura 8.13):

$$\mathbf{S}_t = (x_2/x_1, y_2/y_1) \quad (8.4)$$

En cualquier caso, la transformación se puede calcular en el ciclo de seguimiento, dando como información de realimentación el resultado de la transformación.

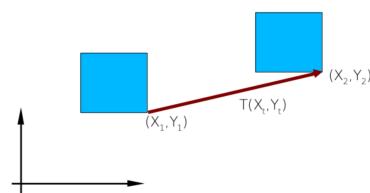


Figura 8.11: Entrada de un vector de translación usando dos puntos.

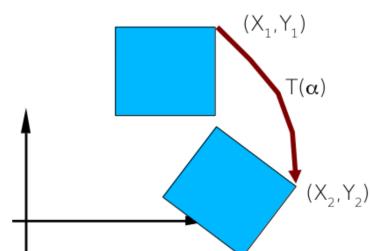


Figura 8.12: Entrada de un vector de rotación usando dos puntos.

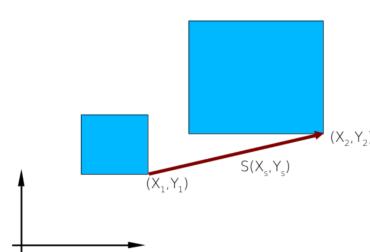


Figura 8.13: Entrada de un vector de escalado usando dos puntos

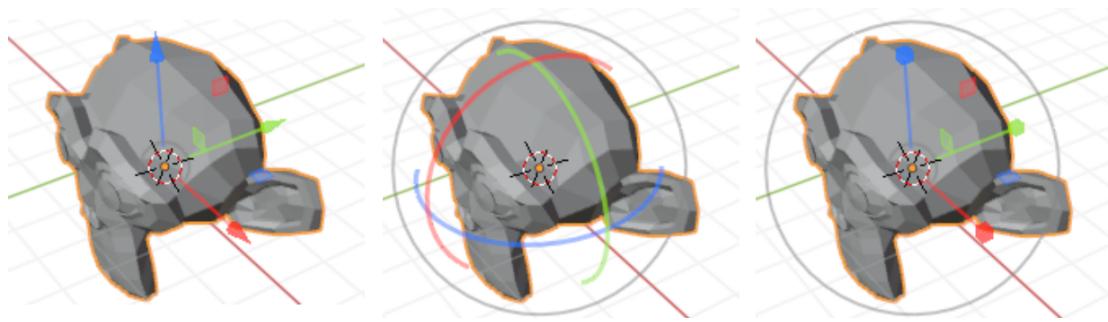


Figura 8.14: Gizmos para traslación, rotación y escalado.

Utilización de gizmos

Los programas interactivos suelen usar gizmos (manipuladores) para realizar las transformaciones geométricas de forma interactiva. Los gizmos son representaciones visuales que permiten manipular objetos en un entorno 3D. Por ejemplo, las flechas o manijas que permiten mover, rotar o escalar un objeto se podrían describir como herramientas de manipulación.

Cada transformación geométrica tiene un gizmo específico que se visualiza sobre los objetos cuando se les va a aplicar una transformación. El usuario puede seleccionar cada uno de los ejes del gizmo y modificar el valor de la componente correspondiente de la transformación geométrica.

8.6. Ejercicios

1. Escriba un procedimiento para la introducción de líneas utilizando la técnica de la línea elástica.
2. Una restricción gravitacional es una transformación de los puntos introducidos que hace que no se puedan introducir posiciones demasiado cerca de las ya introducidas. Supón una aplicación en la que se introducen puntos. Cada nuevo punto introducido se añade a un vector de puntos. Programa una restricción gravitacional que haga que los nuevos puntos se comprueben respecto a los ya existentes y si hay alguna a una distancia menor un valor prefijado u se devuelva el identificador del punto ya existen sin añadir un nuevo punto.
3. Describir cómo se podría realizar la interacción con un ratón, de tal forma que el cursor se moviese sobre un rectángulo, haciendo que cuando salga por un lado, aparezca por el lado opuesto.
4. Implementar en OpenGL un método para introducir una poligonal definida sobre un plano en el espacio.
5. Describe usando pseudocódigo el proceso de interacción para transformar objetos en OpenGL con Glut.
6. ¿Cómo se pueden implementar gizmos para transformar las mallas de un sistema que visualiza mallas indexadas con OpenGL con Glut?
7. ¿Se podría utilizar una lectura de posición para implementar una operación de selección?

8. Escribe la función de callback de ratón de glut para que se seleccione un objeto 2D pulsando el botón izquierdo y se traslade en el plano hasta que se libere el botón.
9. ¿Es posible implementar una lectura de posiciones en modo muestreo usando Glut?
10. Programa una función para obtener las coordenadas de un punto sobre una malla de triángulos?