Práctica 1:

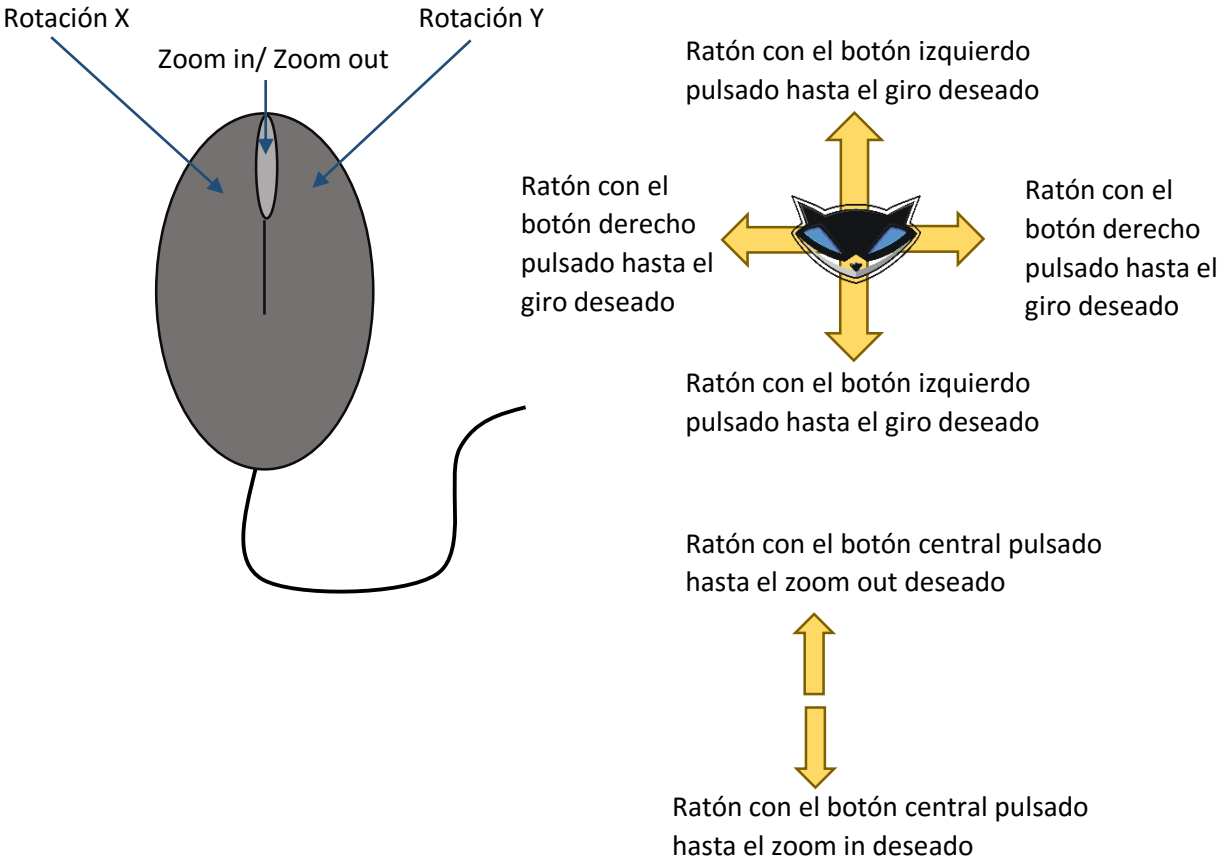
Introducción a GLSL

Parte opcional

Adrián David Morillas Marco y Guillermo Meléndez Morales
Diseño y Desarrollo de Videojuegos + Ingeniería de Computadoras

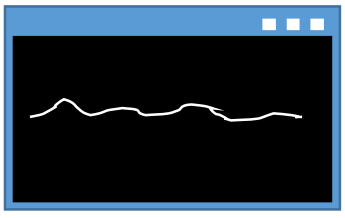
Control de la rotación de la cámara con el ratón

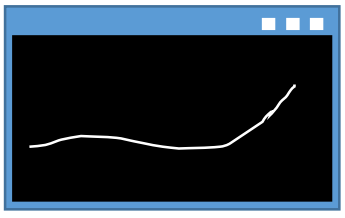
Para poder controlar la rotación, emplearemos el mismo método que empleamos en el apartado de la parte opcional: control de la cámara con el teclado. Es decir modificaremos la matriz view, mediante una matriz auxiliar:



Para controlar la libertad del ratón a la hora de indicar que rotación se desea hacer, emplearemos lo siguiente:

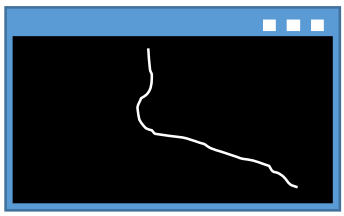
- Rotación en el eje x si el ratón lleva un trayecto más o menos horizontal, es decir, que no se exceda el trayecto hacia arriba o abajo:





La imagen de la izquierda sería el trayecto que se reconocería a la hora de hacer la rotación, mientras que la imagen de la derecha no se reconocería y no efectuaría rotación alguna.

- Rotación en el eje y si el ratón lleva un trayecto más o menos vertical, es decir, que no se exceda el trayecto hacia la derecha o izquierda:



La imagen de la izquierda sería el trayecto que se reconocería a la hora de hacer la rotación, mientras que la imagen de la derecha no se reconocería y no efectuaría rotación alguna.

Esto en lenguaje de programación se traduciría como:

```
// variables del método mouseFunc
// diferenciax sera la variable que calcule la diferencia de la coordenada x original menos la final
float diferenciax;
// diferenciax sera la variable que calcule la diferencia de la coordenada y original menos la final
float diferenciay;
// actualx será la variable encargada de guardar la coordenada x incial
float actualx = 0.0f;
// actualx será la variable encargada de guardar la coordenada y incial
float actualy = 0.0f;
// angulo será la variable encargada de incrementar/ decrementar el angulo de rotacion de x e y
float angulo = 0.1f;

// Controla el giro de la cámara utilizando el ratón.
void mouseFunc(int button, int state, int x, int y)
{

// se crea una matrix auxiliar que se empleara para trasladar y rotar
glm::mat4 aux (1.0f);

// si se pulsa el raton...
if (state == 0){
        // se muestra por consola que se ha pulsado un botón del raton
        std::cout << "Se ha pulsado el botón ";
        // se recoge la coordenada x inicial
        actualx = x;
        // se recoge la coordenada y inicial
        actualy = y;

// si se suelta el botón una vez pulsada...
}else
        // se muestra por consola que se ha soltado el botón del ratón
        std::cout << "Se ha soltado el botón ";
        // se resta la coordenada x inicial por la actual
        diferenciax = actualx - x;
        // se resta la coordenada y inicial por la actual
        diferenciay = actualy - y;
```

```
// si se pulsa el boton izquierdo del ratón...
if (button == 0){
	// se muestra por consola que el botón se ha pulsado
	std::cout << "de la izquierda del ratón " << std::endl;
	// se marca un margen para expresar el movimiento
	if (abs(diferenciax) <= 50){
		// se configura la matriz para rotar en el eje X
		aux = glm::rotate(aux, angulo*(diferenciay / 100.0f), glm::vec3(1.0f, 0.0f, 0.0f));
		// se multiplica la matriz aux por la matriz view
		view = aux*view;
	}
}

// si se pulsa el boton central del ratón...
if (button == 1) {
	// se muestra por consola que el botón se ha pulsado
	std::cout << "central del ratón " << std::endl;
	// se configura la matriz para hacer zoom in u out
	aux = glm::translate(aux, glm::vec3(0.0f, 0.0f, diferenciay/100.0f));
	// se multiplica la matriz aux por la matriz view
	view = aux*view;
}

// si se pulsa el boton derecho del ratón....
if (button == 2){
	// se muestra por consola que el botón se ha pulsado
	std::cout << "de la derecha del ratón " << std::endl;
	// se marca un margen para expresar el movimiento
	if (abs(diferenciay) <= 50){
		// se configura la matriz para rotar en el eje Y
		aux = glm::rotate(aux,angulo*(diferenciax/100.0f), glm::vec3(0.0f, 1.0f, 0.0f));
		// se multiplica la matriz aux por la matriz view
		view = aux*view;
	}
}

// se muestra por pantalla que acción se ha realizado
std::cout << "en la posición " << x << " " << y << std::endl << std::endl;
// se le informa a la clase main que se actualiza la matriz
IGlib::setViewMat(view);
}
```

Rotación de un tercer cubo con Bézier

La curva de Bézier es una curva cuyas características son:

- Ha de interpolar el primer y último punto de control
- La tangente de la curva en el primer punto vendrá dada por el segmento P0-P1
- La tangente de la curva en el último punto vendrá dada por el segmento P(n-1)-Pn
- La curva debe ser simétrica respecto a t y (t-1)

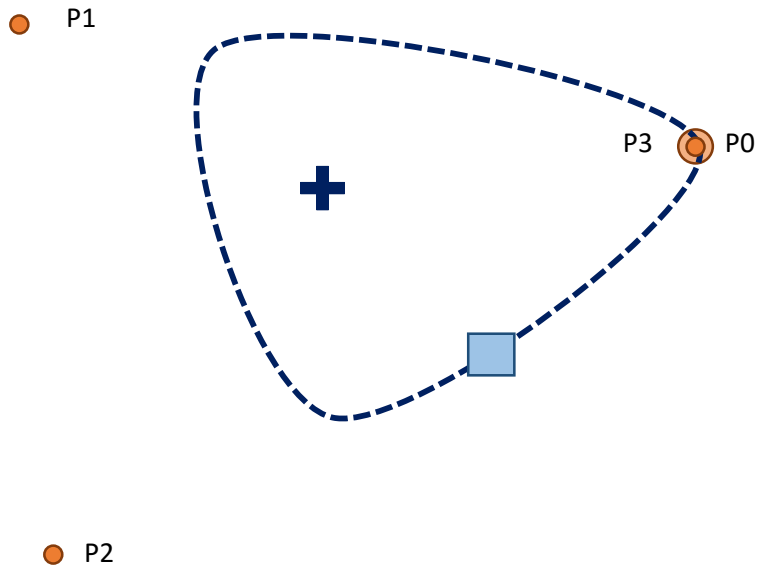
Nosotros aplicaremos una curva con estas características a la traslación de un nuevo cubo, para ello emplearemos el siguiente código en el main():

```
//Se crea el objeto de la parte opcional

objId3 = IGlib::createObj(cubeNTriangleIndex, cubeNVertex, CubeTriangleIndex,cubeVertexPos, cubeVertexColor, cubeVertexNormal, cubeVertexTexCoord, cubeVertexTangent);
```

A continuación nos dirigimos al método Idle () donde crearemos la curva de Bézier y la aplicaremos a la traslación del nuevo cubo. Nuestro objetivo será crear una curva cerrada para hacer un movimiento rotatorio cuyo centro será el primer cubo creado.

Crearemos 4 puntos de control, para cerrar la curva haremos que el primer y último punto de control sean iguales, es decir que coincidan en el mismo punto. La curva se realizara en 2D, es decir en el plano xz, por lo que la coordenada y tendrá un valor nulo:



El grado de esta curva es n-1 es decir, es de grado 3, por lo consiguiente la ecuación es:

$$\big(x(t), y(t), z(t)\big) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + (1-t)t^2 P_2 + t^2 P_3$$

Tras sustituir los puntos en dicha ecuación, sustituiremos en la matriz de traslación dx por x(t), dy por 0 y dz por z(t). En cuanto a t, lo iremos aumentando hasta que alcance el valor 1, reseteando la variable de nuevo a 0.

Lo mencionado anteriormente, en código de programación se traduce a:

```
//cubo de bezier
        // xt y zt serán las variables que almacenaran los valores de Bezier
        float xt, zt;

        // t será la variable encargada de mover el cubo por la curva
        static float t = 0.0f;
        // t siempre mantendra un valor entre 0 y 1
        t = (t <= 1.0f) ? t + 0.0001f : 0.0f;
        // P0 (8, 0, -6)
        glm::vec3 P0 = glm::vec3(8, 0, -6);
        // P1 (-20, 0, -20)
        glm::vec3 P1 = glm::vec3(-20, 0, -20);
        // P2 (20, 0, 100)
        glm::vec3 P2 = glm::vec3(20, 0, 100);
        // P3 (8, 0, -6)
        glm::vec3 P3 = glm::vec3(8, 0, -6);;

        // x(t) = .....
        xt = pow((1 - t), 3)*P0.x + 3*pow((1-t),2)*t*P1.x + (1-t)*(pow(t, 2))*P2.x + pow(t, 3)*P3.x ;
        // z(t) = ....
        zt = pow((1 - t), 3)*P0.z + 3*pow((1-t),2)*t*P1.z + (1-t)*(pow(t, 2))*P2.z + pow(t, 3)*P3.z ;

        // se crea una matriz model
        glm::mat4 model3(1.0f);
        // se aplica una traslación siendo la curva de Bezier
        model3 = glm::translate(model3, glm::vec3(xt, 0.0f, zt));
        // se crea una matriz de escalado
        glm::mat4 escalado3(1.0f);
        // se escala el cubo
        escalado3 = glm::scale(escalado3, glm::vec3(0.5f, 0.5f, 0.5f));
        // se multiplica el escalado por el cubo
        model3 = escalado3*model3;

        // se actualiza la matriz
        IGlib::setModelMat(objId3, model3);
```

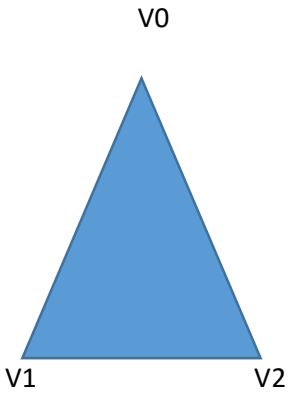
Creación de una pirámide y su adición a la escena

Para crear una pirámide, ha sido necesario crear un nuevo archivo del tipo “.h”.

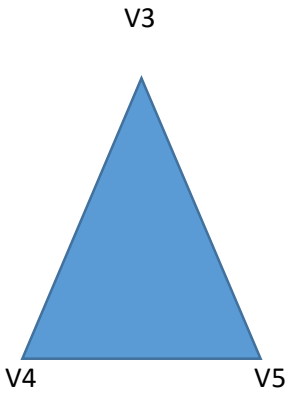
En este archivo, se han tenido que definir los vértices y las normales para así poder mostrar por pantalla la figura.

Para los vértices se ha tenido en cuenta los siguientes triángulos y vértices para definir cada cara:

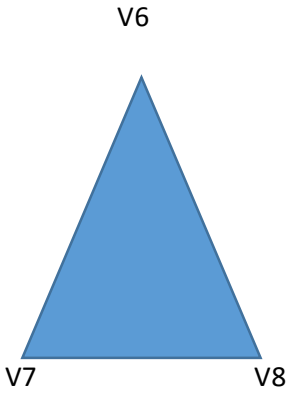
Cara frontal (z=1):



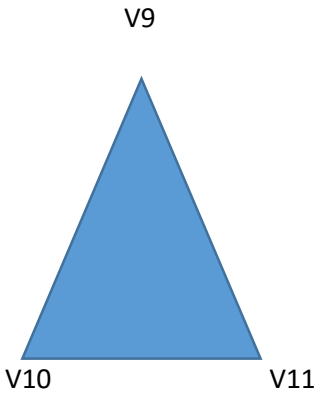
Cara frontal (z=-1):



Cara lateral derecha (x=1):



Cara lateral izquierda (x=-1):



Base de la pirámide (y=-1) Vista desde arriba y con el eje Z como arriba y abajo y el eje X como izquierda y derecha (V12=(-1.0, -1.0, -1.0) v13(-1.0, -1.0, 1.0) v14(1.0, -1.0, -1.0) V15(1.0, -1.0, 1.0)

Para calcular las normales se ha tenido en cuenta el valor del pico de la pirámide V0=V3=V6=V9=(0.0, 2.0, 0.0) y de que eje depende cada cara, es decir, si es la cara z=1, z=-1, etc. De tal manera, el vector normal sería, para cada vértice, (x,1,z), tomando valores “x” o “z” según la cara a la que pertenezca el vértice(x=1 si es la cara derecha, con z a cero; x= -1 si es la cara izquierda, con z a cero; z=1 con la cara frontal, con x=0; z=-1 con la cara

trasera, con x=0. Con la base solo es necesario el vector (0,1,0) ya que no está influenciado por los otros dos ejes, al contrario que los otros).