Tarea3

Cámara

Instrucciones:

- Para esta tarea estudia:

https://docs.gameloft.org/3d-training/#Math_Prerequisites

https://docs.gameloft.org/3d-training/#3_How_to_implement_the_camera_in_your_application

- Programa un InputManager que sea singleton o solo una variable global para controlar las entradas del teclado. Independientemente del método a utilizar, se utilizara una variable para guardar en ella en forma de bits las teclas presionadas. Utilizar la función Key() para el registro de las teclas.

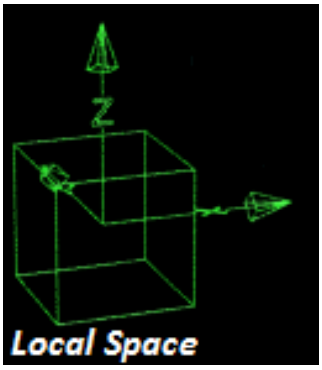
  Ejemplo Singleton: https://docs.gameloft.org/wp-content/uploads/2012/04/SingletonSample2.txt

- **Implementar cámara "simple approach"**

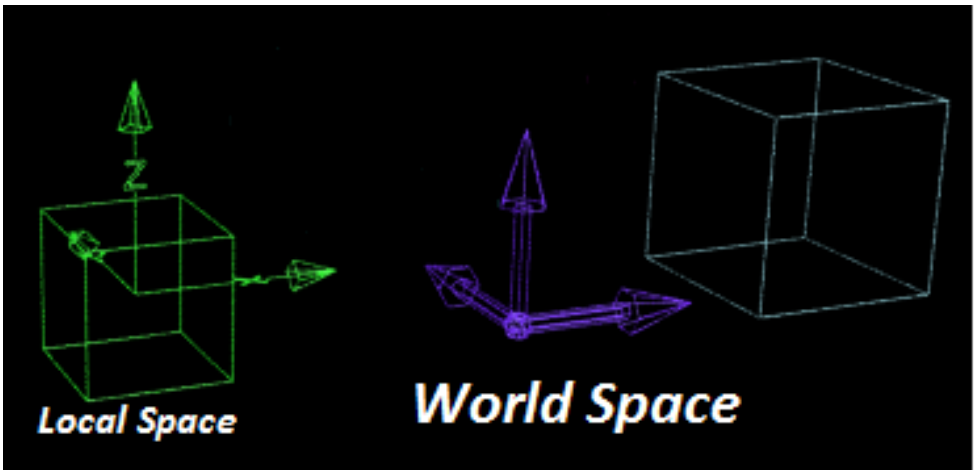
- La cámara deberá usar para **movimiento las teclas [W, A, S, D] y para rotación UP/DOWN, LEFT/RIGHT**

Explicación:

1. Primero, un artista construye un modelo en algún software como 3DStudioMax. Todos los vértices están expresados en el "**local space**" o "**object space**", significa que están expresados tomando como referencia el pivote (pivote =(0,0,0)).



2. Después ese modelo es instanciado como 1 o varios objetos que son puestos en escena (**world space**). Ahora, el origen 0, 0, 0 es el centro de la escena y tus objetos tomaran nuevas coordenadas y orientación (ejem. rotación) tomando como nueva referencia el nuevo 0,0,0 (centro de la escena).



3. Por lo tanto, ahora tendrás (del modelo inicial) 1 o varios objetos, posiblemente diferentes unos de otros por la posición, rotación y escala en el **world space.**

4. Un espectador ingresa (cámara) y ahora el/ella se vuelve el centro del universo. Por lo tanto, el origen 0,0,0 será tomado desde el espectador (cámara) y todos tus objetos tomaran nuevas coordenadas y rotación expresadas por el nuevo orinen (la posición de la cámara). La escala se mantiene como es.

   "I understand how the engines work now. It came to me in a dream. The engines don't move the ship at all. The ship stays where it is and the engines move the universe around it."

   —Cubert Farnsworth

5. Una característica de la cámara es tener además de una World Matrix como cualquier otro objeto en escena, es tener una View Matrix, y ¿Cómo se construye la View Matrix? (Matriz de la Cámara)

   a. La cámara es un objeto como cualquier otro objeto en la escena.

   b. La camara es traída desde su **local space** al **world space** por medio de una **World Matrix.**

   c. Ahora, tú quieres mover el world entero desde la cámara. Por lo tanto tienes que traer tu cámara desde el **World Space** a su **Local Space.** Para lograr esto necesitaremos la inversa de la anterior transformación, lo cual nos deja:

      View Matrix = el inverso de la world matrix de la cámara

Teniendo en cuenta que **worldMatrix = ~~scale *~~ rotationZ * rotationX * rotationY * traslation** (descartamos la escala, porque no tiene sentido tenerla en la cámara, seria "útil" para hacer un zoom in y zoom out, pero hay otros métodos, y en este training no usaremos zoom)

Por lo tanto:

viewMatrix = inverse(rotationZ*rotationX*rotationY*traslation)

igual a:

inverse(traslation) * inverse(rotationY)*inverse(rotationX)*inverse(rotationZ)

dónde:

inverse(traslation) = SetTraslation(-posCam.x, -posCam.y, -posCam.z)

inverse(rotationX) = SetRotationX(-rotAngleCam.x)

****en tu practica omitiremos la rotación en el eje Z***

6. Leer:

   https://docs.gameloft.org/3d-training/#3_How_to_implement_the_camera_in_your_application

   "**I. Simple approach" + "I + II For both approaches:"**

7. Recuerda usar el Simple Approach.

8. Para ver que la cámara haga bien las rotaciones, tienes que agregar la perspectiva de rotación, que hará el mapa de tu frustum (lo que ve la cámara) sea llevado a un plano 2D tomando en cuenta la perspectiva (donde los objetos lejanos se verán pequeños - de hecho son escalados pos un factor de minificación):

   a. En la función **Init()** de tu clase cámara, agrega el cálculo de la proyección (**esta matriz no cambiara y será la misma durante todo el training**):

      Projection.SetPerspective(fov, (float)Globals::screenWidht / Globals::screenHeight, near, far);

      Ejemplo**: fov = 1.0f, near = 0.1f, far = 500.0f**

   b. Tendrás que mandar a la GPU la WorldViewProjection matrix, calculada por wolrdMatrix de tu triangulo * view Matrix *projectionMatrix, en tu código de c++.

Guía:

1. Para activar y desactivar los bits en una variable para almacenar las teclas presionadas, utiliza operaciones a nivel de bits. Ejemplo:

```
void Key ( ESContext *esContext, unsigned char key, bool bIsPressed)
{
if(bIsPressed)

{
      switch(key)
           {
            case Globals::KEY_W:
            keyPressed |= 1<<Globals::_KEY_W; //Al precionar la tecla
            break;
            }
else

{
      switch(key)
           {
            case Globals::KEY_W:
            keyPressed &=~ (1<<Globals::_KEY_W); // Al soltar la tecla
            break;
            }
}
```

Donde _KEY_W y KEY_W en Globals.h tiene un valor:

```
static enum Key
{
  KEY_A               = 65,
  KEY_B,
  KEY_C,
  KEY_D,
  KEY_E,
  KEY_F,
  KEY_G,
  KEY_H,
  KEY_I,
  KEY_J,
  KEY_K,
  KEY_L,
  KEY_M,
  KEY_N,
  KEY_O,
  KEY_P,
  KEY_Q,
  KEY_R,
  KEY_S,
  KEY_T,
```

```
  KEY_U,
  KEY_V,
  KEY_W,
  _KEY_W          = 1,
  _KEY_S,
  _KEY_A,
  _KEY_D,
  _KEY_UP,
  _KEY_DOWN,
  _KEY_LEFT,
  _KEY_RIGHT,
  _KEY_NONE,
}
```

2. En la función update de NewTrainingFramework verificaras si se presionó una tecla para indicar algún movimiento, ejemplo:

```
void ProcessUserInput(float delta)
{

      if((keyPressed & (1<<Globals::_KEY_W)) != 0)
            cam.move_Forward(delta);
      if((keyPressed & (1<<Globals::_KEY_A)) != 0)
            cam.move_Left(delta);
      if((keyPressed & (1<<Globals::_KEY_S)) != 0)
            cam.move_Backward(delta);
      if((keyPressed & (1<<Globals::_KEY_D)) != 0)
            cam.move_Right(delta);
      if((keyPressed & (1<<Globals::_KEY_UP)) != 0)
            cam.rotate_Up(delta);
      if((keyPressed & (1<<Globals::_KEY_DOWN)) != 0)
            cam.rotate_Down(delta);
      if((keyPressed & (1<<Globals::_KEY_LEFT)) != 0)
            cam.rotate_Left(delta);
      if((keyPressed & (1<<Globals::_KEY_RIGHT)) != 0)
            cam.rotate_Right(delta);

}


void Update ( ESContext *esContext, float deltaTime )
{
  // angle += (float)( deltaTime * 40.0f * M_PI/180);

   ProcessUserInput(deltaTime);

}
```

3. Crea la clase Camera, las siguientes funciones serán necesarias:

```
void move_Forward(float deltaTime);
void move_Backward(float deltaTime);
void move_Left(float deltaTime);
void move_Right(float deltaTime);
void rotate_Up(float deltaTime);
void rotate_Down(float deltaTime);
void rotate_Left(float deltaTime);
void rotate_Right(float deltaTime);
```

donde para movimiento:

```
Vector3 positionInfo;// vector declarado en Camera.h
Vector3 m_rotateInfo;// No hay que olvidar poner en cero en el
constructor de Camera

void Camera::move_Forward(float deltaTime)
{
	Vector4 moveL;
	Matrix	worldMatrix;
	Vector4 moveW ;

	worldMatrix = getWorldMatrix(); //camculamos la WorldMatrix
	//alteramos solo el eje Z
	moveL = Vector4(0, 0, -deltaTime * SPEED_CAM,1);
	//Necesitamos la posicion en el world space
	moveW = moveL * worldMatrix;

	m_positionInfo.x += moveW.x;
	m_positionInfo.y += moveW.y;
	m_positionInfo.z += moveW.z;

}
```

Para la rotación solo afectaremos el eje correspondiente:

```
void Camera::rotate_Up(float deltaTime)
{
	float angle =(float) (m_rotateInfo.x * 180/M_PI);
	if(angle>=-90) //delimitamos rotacion arriba y abajo a 90°
	m_rotateInfo.x += ( -deltaTime * SPEED_CAM_ROT * M_PI/180);}
```

4. La World Matrix que tendrá la cámara (todos los objetos tendrán una world matrix) solo actualizara sus valores para las matrices de rotación en X y Y, las demás matrices se mantendrán como matriz identidad:

```
Matrix worldMatrix;
```

```
Matrix cam_rotationXMatrix;
Matrix cam_rotationYMatrix;
Matrix cam_translationMatrix;

cam_rotationXMatrix.SetIdentity();
cam_rotationYMatrix.SetIdentity();
cam_translationMatrix.SetIdentity();

cam_rotationXMatrix.SetRotationX(m_rotateInfo.x);
cam_rotationYMatrix.SetRotationY(m_rotateInfo.y);

worldMatrix = cam_rotationXMatrix * cam_rotationYMatrix *
cam_translationMatrix;
```

5. La viewMatrix es la inversa de la WordMatrix, y la matriz de Escalamiento se descarta por que esta se utilizaría si tuviéramos zoom, y no es un requerimiento.

   ViewMatrix = TraslationM X RotationYM X RotationXM;

   Al ser el inversion de la Wold Matrix necesitaremos poner en negativo los valores:

```
Matrix cam_rotationXMatrix;
Matrix cam_rotationYMatrix;
Matrix cam_translationMatrix;
Matrix cam_ViewMatrix;

cam_rotationXMatrix.SetIdentity();
cam_rotationYMatrix.SetIdentity();
cam_translationMatrix.SetIdentity();

cam_translationMatrix.SetTranslation(-m_positionInfo.x, -
m_positionInfo.y, -m_positionInfo.z);
cam_rotationXMatrix.SetRotationX(-m_rotateInfo.x);
cam_rotationYMatrix.SetRotationY(-m_rotateInfo.y);
```

6. El movimiento de la cámara estará regulado por una velocidad constante por lo tanto:

   movimiento = vec4(**deltaTime * SPEED_CAM** , 0,0,1);

7. Limitar el movimiento de rotación de la cámara en el eje X (rotación hacia arriba y abajo)a 90º, como lo vimos en el ejemplo del paso 2:

```
float angle =(float) (m_rotateInfo.x * 180/M_PI);
if(angle>=-90) //delimitamos rotacion arriba y abajo a 90°
m_rotateInfo.x += RotateX;
```

8. En NewTrainingFramework, en la función **`void Draw()`** necesitaremos calcular la **W**orld**V**iew**P**rojectionMatrix

```
Matrix u_wvp =  worldMatrix * cam.get_viewMatrix()
*cam.get_projectionMatrix();
```

Dónde:

worldMatrix → es la world matrix que estamos calculando para el triángulo, es la world matrix del objeto.

9. Mandar como uniform la matriz "View Matrix Projection" al Vertex Shader desde **`void Draw ( ESContext *esContext )`**, para multiplicarla por la posición local del objeto (triangulo).

```
if(myShaders.uWVPLoc!=-1)
{
Matrix wMatrix = WorldMatrix_Calc();
glUniformMatrix4fv(myShaders. uWVPLoc, 1, GL_FALSE, & u_wvp.m[0][0]);
}
```

*No olvides declarar **uWVPLoc** en Shaders.h y tomar la localización en Shaders.cpp

10. En el vertex shader TriangleShaderVS.vs agregaras la uniform `u_WVP`:

```
attribute vec3 a_posL;
attribute vec4 a_color;
varying vec4 v_color;
uniform mat4 u_WMatrix;
uniform mat4 u_WVP;
void main()
{
```

11. Ahora para ver las cosas desde la perspectiva de la cámara vamos a multiplicar la WorldViewProjection matrix por la posición de los vértices (a_PosL):

```
void main()
{
vec4 posL = vec4(a_posL, 1.0);
gl_Position =  u_WVP * posL;
v_color = a_color;
}
```