Tarea2

Rotar Triangulo

Instrucciones:

- Para esta tarea estudia **Math_Prerequisites** de la siguiente página:

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

- Estudia las operaciones de "Utilities/Math.cpp"
- Rotar el triángulo en el eje X.

Guía:

1. Declara una variable flotante para calcular el ángulo en NewTrainingFramework.cpp

```
GLuint vboId;
Shaders myShaders;
float angle;

int Init ( ESContext *esContext )
{

.
.
```

2. En "`void Update(…)`" usar la siguiente instrucción para calcular el ángulo en radianes:

```
angle += (float)( deltaTime * 40.0f * M_PI/180);
```

```
y en la parte de las cabeceras pon las siguientes lineas para poder
utilizar M_PI:
```

```
#define _USE_MATH_DEFINES
#include <math.h>
```

3. Se necesitara calcular la "World Matrix" del triángulo, donde se especificara el ángulo de rotación; el cálculo deberá hacerse en la función "`void Draw(…)`" , recuerda que la World Matrix está compuesta de:

   `worldMatrix` = scaleMatrix*rotZMatrix* rotXMatrix* rotYMatrix*traslationMatrix;

   (Escala X Rotación X Traslación)

Por lo tanto declara una matriz para cada miembro que compone la World Matrix, ejemplo:

```
Matrix scaleMatrix;
Matrix rotationXMatrix;
Matrix rotationYMatrix;
Matrix rotationZMatrix;
Matrix translationMatrix;

Matrix worldMatrix;

scaleMatrix.SetIdentity();
rotationXMatrix.SetIdentity();
rotationXMatrix.SetRotationX(angle);
.
.
.
```

4. La Word Matrix la mandaremos al shader, pero necesitamos saber en qué parte del program (shader compilado) está la localización del atributo o uniform que queremos mandar, ya que el program hace un índice de atributos y de uniforms, para saber utilizamos el siguiente codigo:

```
worldMatrixLoc = glGetUniformLocation(program, "u_WMatrix");
```

Si `u_WMatrix` no es utilizado en nuetro shader (vertex o fragment shader) regresara como resultado -1

5. `u_WMatrix` debe de estar declarado en nuestro vertex shader

```
attribute vec3 a_posL;
attribute vec4 a_color;
varying vec4 v_color;
uniform mat4 u_WMatrix;

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

6. Indicamos que shader vamos a utilizar, es este caso: "`glUseProgram(myShaders.program);`" y mandamos como uniform la "World Matrix" del triángulo al shader con la instrucción "`glUniformMatrix4fv`".

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

7. En el Vertex Shader multiplicaremos la “World Matrix” por la “Posición Local” y el resultado lo asignaremos a “`gl_Position`”

Resultado: