

## Ejercicio 2: Triángulo rotando

Partiendo del código generado en la práctica 1, se pide crear un programa que muestre una ventana de OpenGL con un triángulo rotando. Para ello, se podrá usar el código visto en clase, e implementar las nuevas clases pedidas a continuación.

### Estructura Vertex :

Esta estructura representa la información que puede contener un vértice. Por el momento sólo tendrá un registro de tipo "Vector4f" llamado "posición". Se aconseja guardarlo en un archivo "vertex.h"

### Clase Object3D:

Esta clase representa un objeto 3D que podrá ser dibujado por la clase Render. Deberá tener las siguientes propiedades (todas públicas, para facilitar el acceso a ellas):

- Vector4f position;
- Vector4f rotation;
- Vector4f scale;
- Matrix4x4f modelMatrix;
- std::vector<Vertex> vertexList
  - o Lista de vértices del objeto
- std::vector<int> idList :
  - o Lista con los índices que se usarán para dibujar la lista de vértices
- void createTriangle() :
  - o Método que inicializa la lista de vértices con 3 vértices formando un triángulo cuyo centro se encontrará en el origen de coordenadas (0,0,0)
- virtual void move(double timeStep):
  - o Método virtual (podrá ser reescrito por clases herederas) que actualiza la posición/rotación/escalado del objeto. Deberá consultar el estado del teclado, y si se pulsa la tecla "D" aumentará el ángulo de giro en el eje Y.
  - o Si se pulsa la tecla "A" disminuirá el ángulo de giro en el eje Y
- void updateModelMatrix():
  - o Método que actualiza la matriz modelo en base a los datos de posición, rotación y escalado de este objeto.

## Clase InputManager:

Esta clase deberá implementar un gestor de eventos de teclas/ratón que podrá consultarse durante la ejecución del programa para poder actualizar los estados de los objetos. Se puede realizar de la siguiente manera:

- static std::map<int,bool> keysState:
  - o Mapa que guarda el estado del teclado. Se puede consultar si una tecla está pulsada o no
- static void keyboardManager( .... )
  - o Método de tipo “GLFWKeyFunc”, que recibirá los eventos de teclado y actualizará el mapa de teclas
- static void init(GLFWWindow\* window)
  - o Método que recibe por parámetros la ventana de glfw sobre la que se gestionarán los eventos. Se inicializará el evento de teclado usando la función keyboardManager como parámetro.

## Clase Render:

Esta clase se usará para inicializar GLFW, crear una ventana y dibujar objetos en ella. Se pueden quitar los métodos para dibujar por terminal usados en la primera práctica (no se usarán más). A continuación se describen los nuevos métodos que se añadirán:

- GLFWWindow\* window
  - o Atributo que almacena una referencia a la ventana de opengl que usaremos para dibujar
- std::vector<Object3D\*> objectList
  - o Lista de objetos que se dibujarán en la ventana de openGL
- std::map<int,bufferObject> bufferObjects
  - o Lista de identificadores de buffer
- void initGL()
  - o La clase render recibía en su constructor el tamaño del framebuffer (coincide con la resolución de la pantalla o tamaño de la ventana). Este método se usará para crear una ventana de openGL con esos datos, e inicializar el sistema de renderizado.

- También llamará al método “initInputManager” de la clase InputManager para poder inicializar los eventos de teclado con la ventana creada anteriormente.
- void putObject(Object3D\* obj)
  - Método que añade un objeto a la lista de objetos. Debe crear su bufferObject y almacenar en el mapa bufferObjets los identificadores de buffers creados/cargados en GPU.
- void removeObject(Object3D\* obj)
  - Método que elimina un objeto a la lista de objetos. Debe destruir su bufferObject y buffers creados/cargados en GPU.
- void drawGL()
  - Método que dibuja los objetos en la ventana de openGL. Este método recorre la lista de objetos, y por cada uno de ellos los carga en GPU y dibuja sus datos de vértices
- void mainLoop()
  - Método que se mantiene en un bucle mientras esté abierta la ventana de OpenGL.
  - El bucle seguirá el patrón visto en clase:
    - Limpiar framebuffer
    - Actualizar los eventos de ventana
    - Actualizar el estado de los objetos llamando a su método “move”
    - Dibujar esos objetos en pantalla
    - Intercambiar buffers de imagen

## Programa principal:

Usando las clases creadas anteriormente, se pide implementar un programa que interactúe con el usuario por teclado, y que realice las siguientes operaciones:

- Crear una clase "Render" de tamaño 640\*480 píxeles , y llamar a su método “initGL”
- Crear una variable "triangulo" de tipo Object3D, y llamar a su método “createTriangle” para que almacene los vértices e identificadores de vértices para un triángulo
- Dibujarlo en el buffer (putObject), y mostrar el buffer por ventana (mainLoop)
- Cuando se pulsen las teclas “A” o “D” el triángulo girará acorde con lo indicado en su método “update”.

## Evaluación:

La práctica se considerará aprobada si al menos muestra un triángulo en la ventana de OpenGL. La práctica tendrá una penalización del 50% si se entrega tarde.

## Entrega:

Se debe entregar el código desarrollado a través del blackboard antes de la fecha de entrega dada. Si se entrega más tarde, la práctica tendrá una penalización del 50% del total de la puntuación. En caso de trabajar en grupo ambos alumnos deben entregar la práctica a través del blackboard.

Nombrado del fichero:

Entregar un fichero en formato zip siguiendo la siguiente nomenclatura:

- PRGR2\_NOMBREAPELLIDO1\_NOMBREAPELLIDO2.zip

La práctica debe defenderse ante el profesor. Deben estar ambos alumnos presentes en caso de haber sido realizada en parejas. Las fechas de defensa se comunicarán una vez realizadas las entregas.