Programación 3D

Práctica 4: Carga de mallas

En esta práctica, vamos a añadir soporte de carga de mallas en formato OBJ.

Clase Mesh

Vamos a añadir un método estático a la clase para que, además de poder construir una malla proceduralmente mediante código, podamos cargarla de un archivo de disco. El método será el siguiente:

```
static std::shared_ptr<Mesh> load(
      const char* filename,
      const std::shared_ptr<Shader>& shader = nullptr);
```

El método recibe el nombre del archivo del cual cargar la malla, y un puntero al shader que utilizarán todos los materiales para pintarla (es opcional, si no se especifica, los materiales utilizarán el shader por defecto).

Para realizar la carga de la malla, usaremos la librería TinyObjLoader. Un fichero obj tiene está pinta:

```
# List of geometric vertices, with (x, y, z [,w]) coordinates, w is
optional and defaults to 1.0.
  v 0.123 0.234 0.345 1.0
  v ...
  ...
  # List of texture coordinates, in (u, [v ,w]) coordinates, these will
vary between 0 and 1, v and w are optional and default to 0.
  vt 0.500 1 [0]
  vt ...
```

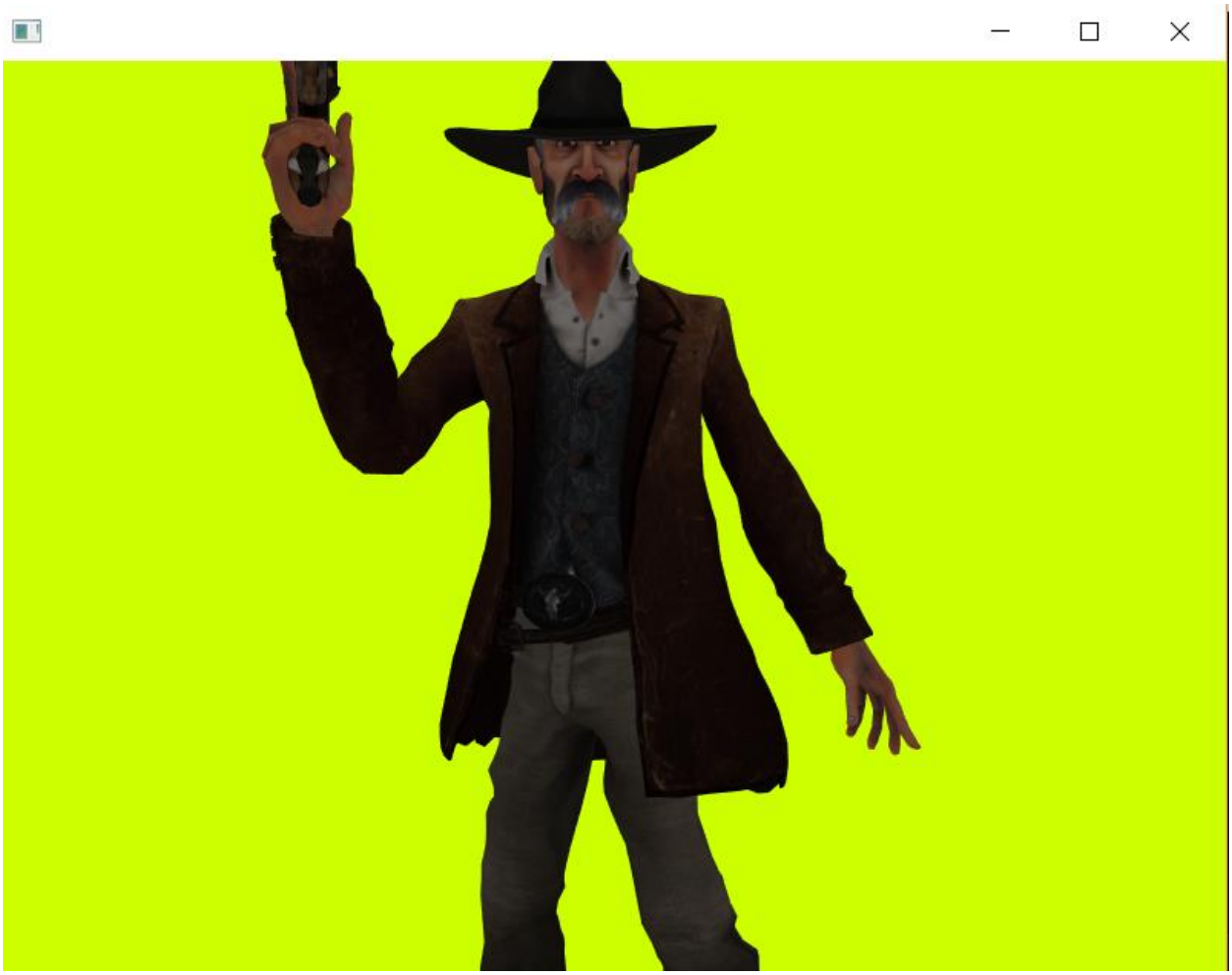
```
  ...
  # List of vertex normals in (x,y,z) form; normals might not be unit
vectors.
  vn 0.707 0.000 0.707
  vn ...
  ...
 …
  # Polygonal face element (see below)
  f 1 2 3
  f 3/1 4/2 5/3
  f 6/4/1 3/5/3 7/6/5
  f 7//1 8//2 9//3
  f ...
  ...
  # Line element (see below)
  l 5 8 1 2 4 9
```

(fuente: wikipedia)

Al cargar un fichero de malla, podemos asumir que las texturas están en el mismo directorio que la malla. Las propiedades de los materiales vienen en el fichero mtl (cada malla del modelo tiene *linkado* un nombre de un marerial).

Programa principal

Realizaremos un nuevo programa en el que cargaremos la malla **"data/box_stack.obj" y "data/gunslinger.obj"** . Crearemos una cámara y un modelo con la malla cargada, y añadiremos ambos al mundo. Al pulsar las teclas de cursor arriba y cursor abajo, la cámara se moverá hacia delante y hacia atrás en la dirección en la que mira. Al pulsar los cursores izquierda y derecha, se moverá en la dirección correspondiente.



El uso de la librería tinyobj es muy sencillo:

1- Importamos:

```
#include "../../lib/tinyobjloader-master/tiny_obj_loader.h"
```

2- Antes debemos definir:

```
#define TINYOBJLOADER_IMPLEMENTATION
```

3- Cargamos el modelo:

```
tinyobj::attrib_t attrib;
std::vector<tinyobj::shape_t> shapes;
std::vector<tinyobj::material_t> materials;
std::string warn, err;

if (!tinyobj::LoadObj(&attrib, &shapes, &materials, &warn, &err,
MODEL_PATH.c_str())) {
        throw std::runtime_error(warn + err);
}
```

4 -Recorremos el fichero cagado haciendo algo similar a:

```
	for (const auto& shape : shapes) {
		for (const auto& index : shape.mesh.indices) {
			Vertex* vertexAux = new Vertex();
			vertexAux->pos.x = attrib.vertices[3 * index.vertex_index + 0];
			vertexAux->pos.y = attrib.vertices[3 * index.vertex_index + 1];
			vertexAux->pos.z = attrib.vertices[3 * index.vertex_index + 2];
			vertexAux->color.r = 1;
			vertexAux->color.g = 1;
			vertexAux->color.b = 1;
			vertexAux->m_texturePos.x = attrib.texcoords[2 *
index.texcoord_index + 0];
			vertexAux->m_texturePos.y = attrib.texcoords[2 *
index.texcoord_index + 1];
			meshAux->m_vertexArray->push_back(*vertexAux);
			meshAux->m_indexArray.push_back(meshAux->m_indexArray.size());
		}
		}
```