** Este archivo lo he generado convirtiendo un archivo .md a pdf. En caso de que se quiera ver de manera correcta, verlo desde el link del repositorio.**
https://github.com/JorgeBarcena3/OpenGL-Renderer/

OpenGL 3D Model Rendering - Jorge Bárcena Lumbreras

Este proyecto consistía en renderizar, a traves de OpenGL, una serie de modelos 3D. Para ello he utilizado la libreria de **TINYOBJLOADER** que se encarga de procesar los archivos .obj; tambien he utilizado **GLM** para realizar las transformaciones de los objetos; tambien me he ayudado de la libreria de **SFML** para gestionar la ventana de la aplicacaion; he utilizado **GLAD** para crear el contexto de OpenGL; por ultimo he utilizado las librerias **TARGA** Y **SOIL** para la lectura de archivos de imagen en las texturas. Todo este proyecto se han llevado a cabo con el lenguaje de programacion **C++**.

Se especificaran algunos detalles del proyecto, sobre como funcionan ciertos objetos y notas sobre su arquitectura.

Todo el código esta documentado para que doxygen genere los documentos pertinentes (Se debe tener activada la opción de JAVA_DOC_BRIEF).

Índice

Librerias externas

Para la realización de este proyecto he utilizado las siguientes versiones de librerías externas:

- TINYOBJLOADER: Lectura de archivos .obj.
- GLM: Funciones matemáticas de OpenGL.
- SFML 2.5.1: Creación de la aplicación de ventana.
- GLAD: Crear el contexto de OpenGL.
- TARGA: Carga de archivos TGA.
- SOIL: Carga de archivos de imagen.
- RapidXML: Lectura y escritura de archivos XML.

Utilizacion de la Demo

Para ver la demo en funcionamiento, podemos ejecutar y compilar el proyecto en Visual Studio 2019 o ejecutar el archivo llamado "OpenGLRendering.exe" situado en la carpeta de "binaries".

Controles

- **A**: Mover la camara hacia su izquierda.
- **D**: Mover la camara hacia su derecha.
- **W**: Mover la camara hacia donde está mirando.
- **S**: Mover la camara hacia el vector inverso de donde está mirando.
- **P**: Activar/Desactivar el postprocesing.
- **SPACE** : Subir la camara en el eje Y del mundo.
- **LEFT CONTROL** : Bajar la camara en el eje Y del mundo.
- **LEFT CLICK** : Mientras presionemos podremos rotar la camara en sus ejes X y Y

Parametros de configuracion

En esta aplicación hay una serie de parametros a configurar para que existan algunas rutas de texturas por defecto, en caso de que no se pueda cargar la original, o bien para gestionar los nombres de los paramentros dentro de los shaders.

A continuacion se presentan una lista de los paramentros por defecto que se aplicaran en caso de que el usuario no establezca algun otro paramentro.

```
ConfigOptions::ConfigPaths::configSettingsMap["texture_default_path"] = "default/texture.tga";
ConfigOptions::ConfigPaths::configSettingsMap["shader_myMaterialKa"] = "myMaterial.Ka";
ConfigOptions::ConfigPaths::configSettingsMap["shader_myMaterialKd"] = "myMaterial.Kd";
ConfigOptions::ConfigPaths::configSettingsMap["shader_myMaterialKs"] = "myMaterial.Ks";
ConfigOptions::ConfigPaths::configSettingsMap["shader_pointLight_array"] = "pointLights";
ConfigOptions::ConfigPaths::configSettingsMap["shader_directionalLight_array"] = "directionalLight";
ConfigOptions::ConfigPaths::configSettingsMap["camera_shader_path"] = "camera/";
ConfigOptions::ConfigPaths::configSettingsMap["skybox_shader_path"] = "skybox/";
ConfigOptions::ConfigPaths::configSettingsMap["postprocesing_shader_path"] = "postprocessing/";
ConfigOptions::ConfigPaths::configSettingsMap["vertexShader_name"] = "vertexShader.vglsl";
ConfigOptions::ConfigPaths::configSettingsMap["fragmentShader_name"] = "fragmentShader.fglsl";
ConfigOptions::ConfigPaths::configSettingsMap["shader_camera_matrix"] = "camera_matrix";
ConfigOptions::ConfigPaths::configSettingsMap["shader_model_matrix"] = "model_matrix";
ConfigOptions::ConfigPaths::configSettingsMap["shader_camera_position"] = "camera_pos";
ConfigOptions::ConfigPaths::configSettingsMap["skybox_path"] = "skybox/SD/sky-cube-map-";
```

En caso de que el usuario desee establecer alguno de estos parametros de forma personalizada, debe incluirlos en el archivo de carga XML, siguiendo la siguiente extructura:

```
<Configuration name="nombre" value="valor" />
```

Se puede visualizar un ejemplo de uso en el siguiente link.

Carga de escenas mediante XML

La herramienta permite cargar escenas mediante archivos XML. La herramienta por defecto cogera un archivo llamado *"scene.xml"* situado en el mismo lugar que el ejecutable. En caso que queramos escoger otro archivo podemos ejecutar la herramienta desde la consola de comandos y pasar un parametro llamado *"–scene"* seguido de la ruta del archivo de escena a cargar. El comando quedaría de la siguiente forma.

```
> OpenGLRendering.exe --scene "Nuevo path"
```

Respecto a la estructura del XML, se puede consultar desde el siguiente archivo de ejemplo.

Arquitectura

He de mencionar dos clases importantes. La primera de todas es la clase **Scene**, que se encarga de manejar toda una escena. Esta escena contiene las demas clases y es la que gestiona los inputs del usuario.

Tambien he de mencionar la clase de **SFMLApplication**, esta clase abstrae la utilizacion de la libreria SMFL a nuestra herramienta. En caso de que no se quiera utilizar esta libreria, solamente se deberá emular esta clase con la nueva libreria.

Iluminación

Para aplicar un modelo de iluminacion en nuestra herramienta se ha hecho mediante el fragment Shader. El modelo utilizado para la representación de la luz es el **Modelo de Phong**. El fragment shader queda de la siguiente forma:

```
void main()
{

    vec3 N = normalize(normal_vec);
    vec3 L = normalize(camera_position - vertex_pos);

    vec4 finalColor = vec4(0);
    int lights = 0;

    for(int i = 0; i < DIRECTIONAL_N; i++)
    {
        if(directionalLight[i].eneabled == 1)
        {
            finalColor += CalculateDirectionalLight(directionalLight[i], N, L);
            lights += 1;
        }
    }

    for(int i = 0; i < POINT_N; i++)
    {
        if(pointLights[i].eneabled == 1)
        {
            finalColor += CalculatePointLight(pointLights[i], N, vertex_pos, L);
            lights += 1;
        }
    }

    finalColor /= lights;

    vec4 alphaValue = texture(diffuse_sampler, tx_coord);
    fragment_color = vec4(vec3(finalColor), alphaValue.a);
}
```

Como hemos podido ver, la herramienta admite dos tipos de luces, luces de tipo direccional y luces posicionales; hasta un maximo de 5 de cada tipo.

Texturas

En la herramienta hay dos posibles tipos de textura:

1. **Texture2D**: Es una textura 2D obtenida a partir de una imagen.
2. **CubeMap**: Es la textura que se aplicará al skybox de la escena, esta textura esta compuesta por 6 texturas.

Para cargar cualquier tipo de textura se utiliza la función estática de la clase Textura, *load_texture(const std::string& texture_path)*. Esta función intenta cargar la textura con el path *Texture_path*, y si no carga la textura por defecto.

Efecto de postproceso

La escena tiene un modulo de postproceso que se puede activar o descativar en cualquier momento de la ejecución del programa. Actualmente hay 4 efectos de postproceso activados que se pueden elegir cambiando el path *"postprocesing_shader_path"* por la carpeta donde se encuentran los shaders.