Documentación de Práctica Final: Motor Gráfico 3D

1. Descripción General

Esta práctica consiste en la implementación de un software de visualización 3D en C++ utilizando la API gráfica **OpenGL 3.3 Core Profile**. El software es capaz de cargar una escena descrita en un archivo de texto externo (`scene.txt`), renderizar geometría compleja como terrenos y mallas, y aplicar efectos visuales avanzados mediante post-procesado.

El proyecto ha sido estructurado para ser modular y extensible, permitiendo la navegación interactiva por el entorno mediante una cámara controlada por el usuario.

2. Arquitectura del Código

El código se ha organizado siguiendo una arquitectura orientada a objetos, separando la lógica de inicialización, actualización y renderizado:

- **Scene (Clase Principal):** Gestiona el ciclo de vida de la aplicación, la carga de recursos, la configuración de shaders y el bucle de renderizado principal.
- **`Node` (Grafo de Escena):** Implementación de una estructura jerárquica de árbol. Cada nodo contiene su propia matriz de transformación local y una lista de hijos. Esto permite que las transformaciones (rotación, traslación) se propaguen de padres a hijos automáticamente.
- **`Camera`:** Encapsula la lógica de la matriz de Vista y Proyección, permitiendo movimiento tipo "Free Look" (FPS).
- **`Terrain`:** Clase encargada de generar una malla poligonal a partir de una imagen de mapa de alturas (Heightmap).
- **`Skybox`:** Renderiza un cubo envolvente con texturas para simular el entorno lejano.
- **Gestión de Recursos:** Se utiliza la librería **SOIL2** para la carga de texturas y **GLM** para las operaciones matemáticas.

3. Características Implementadas

A continuación, se enumeran las características técnicas desarrolladas para cumplir con los requisitos de la práctica:

3.1.

Carga de Escena desde Archivo

El software lee un archivo `scene.txt` ubicado junto al ejecutable. Este archivo define qué elementos cargar (skybox, terreno, cubos) y sus texturas correspondientes, instanciando los objetos dinámicamente en tiempo de ejecución.

3.2.

Grafo de Escena (Scene Graph)

Se ha implementado un sistema de nodos (`Node.hpp`). En lugar de renderizar objetos de forma plana, se añaden a un nodo raíz (`root`). El motor recorre recursivamente el grafo llamando a `update_and_render`, calculando las matrices de mundo multiplicando la transformación del padre por la local.

3.3.

Mallas de Elevación (Terreno)

El terreno se genera procesando una imagen en escala de grises (`height-map.png`).

- Se lee el brillo de cada píxel para determinar la altura (`y`) de cada vértice.
- Se calculan las **normales** de forma dinámica basándose en la altura de los vértices vecinos para asegurar una iluminación correcta.

3.4.

Modelo de Iluminación

Se utiliza un shader personalizado (`vertex_shader` y `fragment_shader`) que implementa iluminación difusa y ambiental (Modelo Phong/Blinn-Phong simplificado).

- Las normales se transforman al espacio de la vista (View Space) para cálculos precisos.
- La luz interactúa con la textura del objeto.

3.5.

Uso de Texturas y Transparencia

- **Texturizado:** Todos los objetos (terreno, cubo, skybox) tienen coordenadas UV y texturas aplicadas.
- **Transparencia (Blending):** Se ha habilitado `GL_BLEND` para objetos específicos (como el cubo flotante), permitiendo ver a través de ellos mediante el canal Alpha de la textura o un valor uniforme (`u_alpha`).

3.6.

Skybox

Implementación de un mapa de cubos (`samplerCube`) que se renderiza deshabilitando la escritura en el Z-Buffer, garantizando que siempre aparezca como fondo infinito.

3.7.

Post-Proceso (Framebuffer)

Se ha implementado una técnica de renderizado en dos pasos:

1. **Off-screen Rendering:** La escena 3D se dibuja en un Framebuffer Object (FBO) personalizado con una textura de color y un Renderbuffer de profundidad.
2. **Screen Quad:** Se dibuja un rectángulo que cubre toda la pantalla usando la textura del paso anterior. En este paso se aplica un **Shader de Post-proceso** que combina:
    - **Filtro Sepia:** Alteración de colores mediante matrices RGB.
    - **Viñeteado:** Oscurecimiento de los bordes mediante coordenadas UV.

3.8.

Animación y Control

- **Animación Automática:** El cubo presente en la escena rota sobre su eje Y automáticamente en cada frame, gestionado a través de la actualización del nodo en el grafo.
- **Cámara:** El usuario puede navegar libremente usando el teclado y el ratón.

3.9.

Optimizaciones OpenGL

- **Face Culling:** Se ha activado `glEnable(GL_CULL_FACE)` para evitar que la GPU renderice las caras traseras de los objetos, ahorrando procesamiento.
- **Minimización de cambios de estado:** Se agrupan las llamadas de renderizado y configuración de uniforms.

4. Manual de Usuario

Para ejecutar el software, abra el archivo ejecutable situado en la carpeta `binaries`. Asegúrese de que la carpeta `assets` y el archivo `scene.txt` se encuentran en el mismo directorio.

Controles:

- **W / S:** Moverse hacia adelante / atrás.
- **A / D:** Moverse hacia la izquierda / derecha.
- **Q / E:** Subir / Bajar verticalmente.
- **Ratón (Clic Izquierdo + Arrastrar):** Rotar la vista (mirar alrededor).
- **ESC:** Cerrar la aplicación.

5. Bibliotecas de Terceros

Para el desarrollo se han utilizado las siguientes bibliotecas (incluidas en la carpeta `libraries`):

- **SDL2:** Creación de ventana y gestión de eventos de entrada.
- **GLAD:** Carga de punteros de funciones OpenGL.
- **GLM:** Biblioteca matemática para vectores y matrices.
- **SOIL2:** Carga de imágenes para texturas.