

# Motor Gráfico 3D en OpenGL

## 1. Descripción General

Este proyecto consiste en la implementación de un software de gráficos 3D basado en **OpenGL 3.3 (Core Profile)** utilizando C++. El motor carga una escena tridimensional compuesta por un terreno generado por mapas de elevación, un cielo (Skybox) y múltiples modelos 3D con iluminación y texturas.

El software integra técnicas avanzadas como **iluminación Phong, transparencias con mezcla alpha, coloración procedimental de terreno, niebla y post-procesado** en tiempo real mediante framebuffers.

## 2. Instrucciones de Uso (Controles)

El usuario puede navegar e interactuar con la escena utilizando los siguientes controles:

- **W / A / S / D:** Mover la cámara (Adelante, Izquierda, Atrás, Derecha).
- **Mouse:** Orientar la vista (Mira alrededor).
- **Tecla F:** Alternar efectos visuales de cámara (Ciclo: Normal -> Sepia -> Visión Nocturna).
- **ESC:** Cerrar la aplicación.

## 3. Arquitectura del Software

El proyecto sigue una estructura de **Grafo de Escena** (Scene Graph) para gestionar las transformaciones jerárquicas de forma eficiente:

- **Node (Clase Base):** Gestiona la posición, rotación y escala. Se encarga de calcular las matrices locales y globales.
- **Mesh (Hereda de Node):** Representa objetos 3D cargados desde archivos externos (ej. `.obj`). Implementa su propio renderizado e iluminación.
- **Terrain (Hereda de Node):** Genera la malla del suelo a partir de una imagen de altura y gestiona su coloreado.
- **Scene:** Clase principal que contiene la cámara, las luces, la lista de nodos y gestiona el ciclo de vida (Update/Render).
- **Camera:** Gestiona las matrices de Vista y Proyección.
- **Light (Hereda de Node):** Representa una fuente de luz puntual en la escena. Al heredar de Node, posee posición 3D y se integra en el grafo de escena, permitiendo que la iluminación sea dinámica y posicional.

## 4. Características Técnicas Implementadas

### 4.1. Terreno Avanzado (Heightmaps y Color Procedimental)

El terreno se genera dinámicamente leyendo un mapa de alturas ([height-map.png](#)).

- **Suavizado de Normales:** Se calculan las normales en el Vertex Shader promediando los píxeles vecinos para evitar el aspecto "facetado" y lograr una iluminación suave.
- **Coloración Procedimental:** En lugar de usar texturas estáticas, el terreno calcula su color en tiempo real mediante un algoritmo en el Fragment Shader. Se interpolan dos colores base (Gris Roca y Blanco Nieve) dependiendo de la altura de cada vértice ([Height](#)), logrando una transición suave y natural en las cimas de las montañas sin necesidad de cargar archivos de imagen adicionales.
- **Niebla Exponencial:** Se aplica una niebla grisácea que aumenta con la densidad en función de la distancia a la cámara.

### 4.2. Modelo de Iluminación (Phong)

Los modelos 3D ([Mesh](#)) implementan el modelo de iluminación **Phong** completo en GLSL:

- **Componente Ambiental:** Luz base constante para evitar sombras negras absolutas.
- **Componente Difusa:** Luz direccional basada en la normal de la superficie (simulando el sol).
- **Componente Especular:** Brillos intensos en la superficie del objeto que dependen de la posición de la cámara ([viewPos](#)) y el material.

### 4.3. Post-Procesado (Framebuffers)

La escena no se dibuja directamente en pantalla, sino en un **Framebuffer** oculto (Textura). Posteriormente, se dibuja un rectángulo que ocupa toda la pantalla aplicando shaders de efectos visuales:

- **Modo Normal:** Color real de la escena.



- **Modo Sepia:** Conversión matemática de colores para simular cine antiguo.



- **Modo Visión Nocturna:** Filtro verde con alto contraste y luminosidad ajustada.



#### 4.4. Transparencias y Blending

Se ha implementado soporte para objetos translúcidos (ej. "Gato Fantasma"):

- Se utiliza `glEnable(GL_BLEND)` con la función `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)`.
- El motor gestiona el orden de renderizado: primero se dibujan los objetos opacos y después los transparentes, desactivando la escritura en el Z-Buffer (`glDepthMask(GL_FALSE)`) para garantizar una visualización correcta.

#### 4.5. Skybox

Se utiliza un **Cube Map** para renderizar un fondo de cielo envolvente que da sensación de profundidad infinita a la escena.

#### 4.6. Animación Procedimental

El motor soporta animaciones automáticas en los elementos de la escena sin necesidad de archivos de animación externos (esqueletos). Estas transformaciones se calculan en cada fotograma (`Update`) basándose en el tiempo (`DeltaTime`):

- **Rotación Continua:** Los modelos giran sobre su eje Y a una velocidad constante.
- **Levitación (Floating):** Se utiliza una función matemática sinusoidal (`sin(time)`) para modificar la altura del objeto transparente, creando un efecto de flotación suave y cíclico.

#### 4.7. Carga de Escena Dinámica (Archivo Externo)

Para cumplir con los requisitos de extensibilidad y evitar el uso de configuraciones fijas en el código (*hardcoding*), se ha implementado un sistema de carga de escenas basado en archivos de texto plano (`.txt`).

- **Lógica de Carga:** Al iniciar la aplicación, la clase `Scene` lee el archivo `assets/scene.txt` línea por línea utilizando `std::ifstream` y `std::stringstream`.
- **Análisis de Datos (Parsing):** El sistema identifica palabras clave para instanciar objetos dinámicamente en el grafo de escena:
  - `TERRAIN`: Define las dimensiones y el mapa de alturas del suelo.
  - `LIGHT`: Configura la posición y el color de la iluminación ambiental y puntual.
  - `MESH`: Especifica la ruta del modelo 3D (OBJ/FBX), su posición inicial y su nivel de opacidad.
- **Ventajas:** Esta implementación permite modificar la composición completa de la escena (añadir múltiples modelos, cambiar luces o el terreno) sin necesidad de recompilar el código fuente, facilitando la iteración y el diseño del nivel.

## 5. Bibliotecas de Terceros Utilizadas

- **SDL3:** Gestión de ventana, contexto OpenGL y entrada (teclado/ratón).
- **GLAD:** Carga de punteros de funciones OpenGL modernas.
- **GLM:** Biblioteca de matemáticas para gráficos (vectores, matrices, transformaciones).
- **ASSIMP:** Carga de modelos 3D en diversos formatos (OBJ, FBX, etc.).
- **SOIL2:** Carga de imágenes y texturas.

## 6. Enlace al Código Fuente

El código completo del proyecto está disponible en el siguiente repositorio:

<https://github.com/penterrin/Grafica>

## 7. Problemas Conocidos y Soluciones

Nota sobre Warnings de Compilación: Durante el proceso de enlazado pueden aparecer advertencias relacionadas con la librería Assimp ("no se encontró PDB 'assimp-vc143-mtd.pdb'..."). Estas advertencias se deben a que la biblioteca de terceros estática no incluye los símbolos de depuración originales (archivos .pdb) del equipo donde

fue compilada. Esto no afecta a la estabilidad ni al código fuente del proyecto, y es un comportamiento esperado al usar esta versión de la librería externa.