Renderizado Dinámico en Tiempo Real de Escenas Marinas: Iluminación Avanzada y Skybox con OpenGL

Marco Yu
marco.yu@upr.edu
University of Puerto Rico at Mayagüez
Mayagüez, Puerto Rico

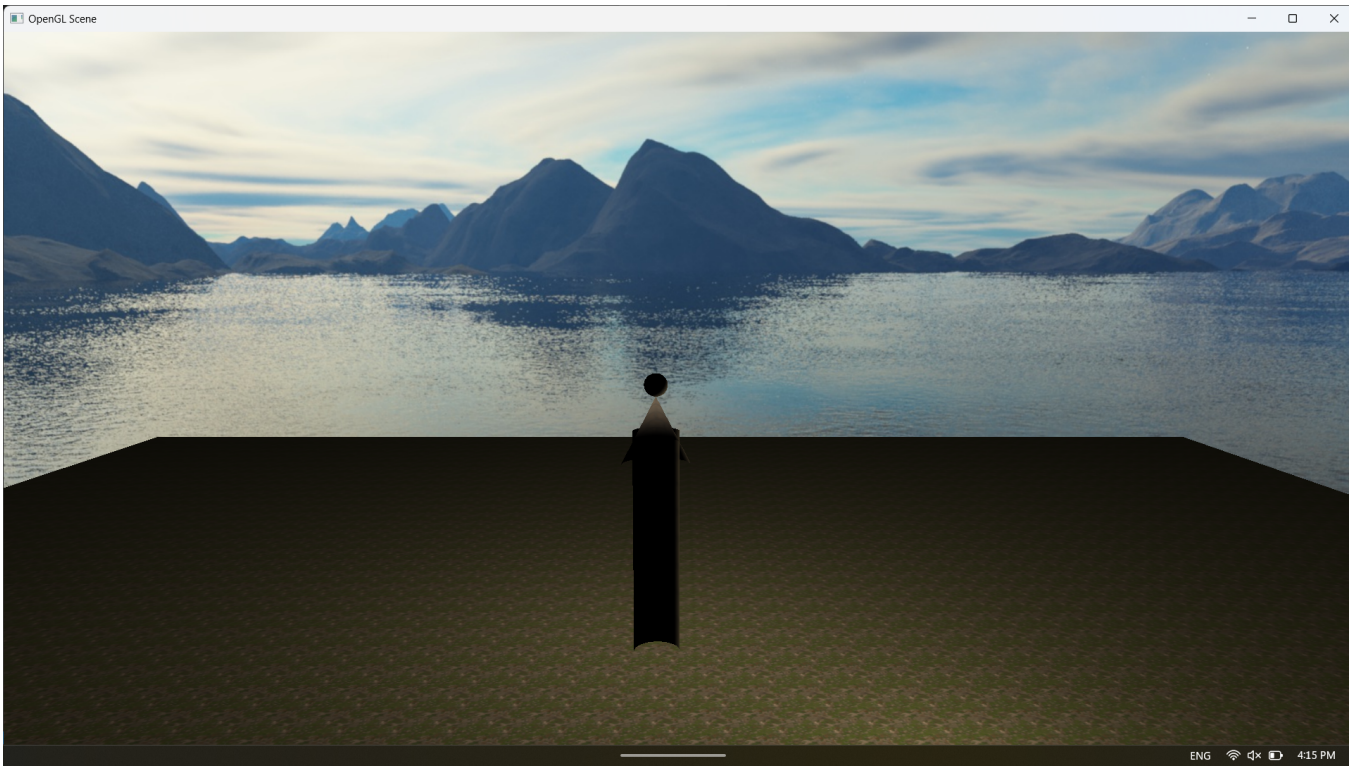

Figure 1: Vista de la escena renderizada mostrando un faro interactivo con skybox panorámico marino e iluminación avanzada.

Abstract

Este trabajo presenta la implementación de un sistema de renderizado en tiempo real que combina iluminación avanzada, un skybox dinámico y una escena interactiva, utilizando OpenGL. La simulación incluye un modelo de iluminación basado en Phong extendido, mapas normales y rugosidad, y optimizaciones como el frustum culling. Los resultados muestran una escena marina realista y dinámica con un equilibrio entre calidad visual y rendimiento.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Keywords

OpenGL, Iluminación Avanzada, Skybox, Renderizado en Tiempo Real, Frustum Culling, Normal Mapping, Interactividad, Shaders

ACM Reference Format:
Marco Yu. 2024. Renderizado Dinámico en Tiempo Real de Escenas Marinas: Iluminación Avanzada y Skybox con OpenGL. In . ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introducción

Durante las últimas dos décadas, el desarrollo de API gráficas como OpenGL y DirectX ha transformado profundamente la capacidad de las computadoras para renderizar gráficos complejos en tiempo real. Estas herramientas han permitido a los desarrolladores crear experiencias visuales inmersivas que anteriormente solo eran posibles en el ámbito cinematográfico. Este avance ha sido impulsado tanto por mejoras en hardware como por el desarrollo de técnicas gráficas avanzadas que optimizan el uso de los recursos computacionales. Tecnologías como el mapeo de normales, la iluminación dinámica y

las texturas de alta resolución han evolucionado para ser accesibles incluso en dispositivos de consumo, democratizando el acceso a gráficos de alta calidad.

El renderizado en tiempo real juega un papel crucial en numerosas aplicaciones, desde videojuegos y simulaciones educativas hasta sistemas de entrenamiento para la industria y la defensa. Estas aplicaciones requieren un equilibrio delicado entre calidad visual y rendimiento computacional, especialmente en escenarios interactivos donde cada milisegundo cuenta para mantener la fluidez de la experiencia del usuario.

En este contexto, mi proyecto busca explorar y combinar técnicas modernas de gráficos computacionales para crear una escena marina inmersiva y dinámica. Inspirado en la belleza de los paisajes naturales, como océanos y montañas, este trabajo representa una manifestación visual de conceptos teóricos aprendidos durante mi curso de gráficos por computadora. Más allá de ser un simple ejercicio técnico, este proyecto se alinea con la motivación personal de integrar tecnología y naturaleza en un entorno visualmente impactante.

Este trabajo no solo representa un avance técnico en la implementación de gráficos en tiempo real, sino que también sirve como modelo para aplicaciones prácticas en videojuegos y simulaciones educativas. La capacidad de ofrecer entornos interactivos de alta calidad visual tiene implicaciones directas en la formación profesional y la educación, demostrando cómo un pipeline gráfico bien optimizado puede abordar los desafíos de tiempo real sin comprometer la calidad visual.

Además, el desarrollo del proyecto me permitió consolidar habilidades prácticas en OpenGL, desde la construcción de shaders personalizados hasta la gestión de texturas y optimizaciones avanzadas como el frustum culling. Este proceso no solo fortaleció mi comprensión de los fundamentos de los gráficos computacionales, sino que también subrayó la importancia de la atención al detalle y el rigor técnico en la creación de sistemas gráficos interactivos. En última instancia, este proyecto destaca cómo la teoría y la práctica se combinan para resolver problemas gráficos complejos de manera creativa y eficiente.

1.1 Contribuciones

- Un sistema interactivo que permite la navegación por teclado y mouse en una escena 3D.
- Integración de un skybox panorámico marino para entornos inmersivos.
- Implementación de iluminación avanzada utilizando el modelo Phong extendido.
- Optimizaciones como el frustum culling para maximizar el rendimiento.
- **Modelado Procedural del Faro:** La construcción del faro se realizó mediante la combinación de primitivas geométricas básicas, ensambladas estratégicamente para minimizar la cantidad de polígonos. Cada parte del modelo fue diseñada pensando en la eficiencia del renderizado:
  - **Base del Faro:** Cilindro con una textura difusa para simular una superficie rocosa o de concreto.
  - **Cuerpo Principal:** Un cilindro alargado texturizado con detalles que simulan desgaste y características arquitectónicas.
  - **Techo Cónico:** Construido mediante una primitiva cónica, texturizada para capturar la estética típica de los faros.
  - **Lámpara Superior:** Esfera que simula el sistema de iluminación, con texturas translúcidas o brillantes.
- **Plano del Terreno:** El terreno en el que se sitúa el faro fue modelado como un plano extendido que actúa como base para la escena. Este plano incluye:
  - *Texturas de Alta Resolución:* Aplicación de texturas de terreno que simulan pasto, arena o rocas, integradas con mapas normales para aportar mayor realismo visual.
  - *Mosaicos Texturizados:* Uso de coordenadas UV adecuadas para replicar patrones sin distorsión o bordes visibles, manteniendo una calidad uniforme incluso en vistas de cerca.
- **Integración de Texturas Avanzadas:**
  - Mapas normales y de rugosidad aplicados al faro y al terreno, que simulan detalles geométricos adicionales sin incrementar la cantidad de polígonos.
  - Texturas difusas combinadas con iluminación avanzada, logrando una apariencia realista bajo diferentes condiciones de luz.

1.2 Contexto y Estado del Arte

El renderizado en tiempo real ha evolucionado significativamente en las últimas décadas, con avances en modelos de iluminación como Phong y técnicas de texturizado como el normal mapping. Aunque las soluciones existentes han mejorado la calidad visual, persisten desafíos para equilibrar calidad y rendimiento, especialmente en hardware de consumo. Este proyecto busca cerrar esta brecha mediante la integración de técnicas modernas y optimizadas.

2 Metodología

2.1 Arquitectura del Sistema

El sistema está desarrollado en C++ utilizando OpenGL y GLSL. Las principales componentes incluyen:

2.2 Clases Principales

El código del proyecto está organizado en una serie de clases que representan los diferentes elementos de la escena y el flujo de renderizado. A continuación, se describen las principales clases y su funcionalidad:

- **Camera:** Representa una cámara en espacio 3D, incluyendo manejo de posición, orientación y transformación para vistas dinámicas.
- **DirectionalLightData:** Contiene los datos necesarios para describir una luz direccional, incluyendo dirección e intensidad.
- **Light:** Representa una fuente de luz en la escena, con propiedades ajustables como color y atenuación.
- **Lighthouse:** Modela un faro texturizado, incluyendo componentes básicos de iluminación.
- **Logger:** Clase sencilla y segura para registrar eventos e información relevante en un entorno multihilo.

- **Mesh:** Representa una malla 3D con vértices, índices y texturas asociadas.
- **Plane:** Modela un plano texturizado que sirve como terreno en la escena.
- **PointLightData:** Contiene datos para describir una luz puntual, incluyendo posición y atenuación.
- **Scene:** Representa la escena completa que será renderizada, gestionando todos los objetos y su interacción.
- **Shader:** Maneja la carga, compilación y uso de shaders en OpenGL, facilitando la configuración del pipeline gráfico.
- **ShaderManager:** Gestiona múltiples shaders, permitiendo alternar entre ellos según las necesidades de la escena.
- **Texture:** Representa una textura 2D o un cubemap en OpenGL, con soporte para carga y configuración.
- **TextureManager:** Clase singleton para gestionar las texturas utilizadas en la aplicación, optimizando el uso de memoria.
- **Vertex:** Define un vértice con posición, normal, color y coordenadas de textura.

2.3 Transformaciones Geométricas

Las transformaciones geométricas son un pilar fundamental en cualquier pipeline gráfico, ya que permiten posicionar, orientar y escalar los objetos dentro de una escena tridimensional. Este proyecto utiliza matrices de transformación, que son herramientas matemáticas eficientes para realizar estas operaciones de manera compacta y uniforme.

Las transformaciones se implementan mediante álgebra lineal en los shaders de vértices, aprovechando la capacidad de las GPU para realizar cálculos matriciales de manera simultánea para miles de vértices. En este contexto, las transformaciones geométricas no solo afectan la ubicación de los objetos, sino que también garantizan una correcta proyección y visualización desde la perspectiva de la cámara.

Matrices de Transformación:

- **Matriz Modelo (M):** Esta matriz aplica transformaciones locales como traslación, rotación y escalado a los objetos. Cada transformación tiene una representación matricial específica:
  - *Traslación:* Mueve el objeto en el espacio 3D.

$$T = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - *Rotación:* Gira el objeto alrededor de un eje.
  - *Escalado:* Ajusta el tamaño del objeto.
- **Matriz Vista (V):** Convierte las coordenadas al espacio de la cámara para garantizar una perspectiva coherente.
- **Matriz de Proyección (P):** Realiza la proyección perspectiva o ortográfica, asegurando que los objetos se muestren correctamente en el espacio 2D de la pantalla.

subsectionIluminación y Sombras La iluminación en gráficos por computadora es esencial para proporcionar profundidad y realismo a una escena. En este proyecto, se implementa el modelo de iluminación Phong extendido, que combina componentes ambientales, difusas y especulares para simular cómo la luz interactúa con las superficies.

Modelo de Iluminación Phong:

- **Componente Ambiental:** Representa la luz indirecta en la escena.
- **Componente Difusa:** Simula cómo la luz se dispersa al impactar una superficie.

$$I_{\text{diff}} = k_{\text{diff}} \cdot I_{\text{light}} \cdot \max(0, \vec{N} \cdot \vec{L})$$

- **Componente Especular:** Modela los reflejos brillantes observados en superficies pulidas.

$$I_{\text{spec}} = k_{\text{spec}} \cdot I_{\text{light}} \cdot (\max(0, \vec{R} \cdot \vec{V}))^n$$

Además, el sistema soporta múltiples fuentes de luz, como luces puntuales y direccionales, cada una de ellas con características únicas de atenuación física.

2.4 Normal Mapping

El mapeo de normales es una técnica avanzada que mejora significativamente la apariencia de las superficies al simular detalles geométricos sin aumentar la cantidad de polígonos. Este método utiliza texturas especiales que contienen información sobre la orientación de las normales en cada punto de la superficie.

- **Normal Perturbada:** Cada píxel del mapa de normales redefine la dirección de la normal en ese punto específico, creando la ilusión de una superficie compleja.
- **Espacio Tangente:** Para integrar el mapeo de normales en el pipeline gráfico, se calcula una base ortonormal (tangente, bitangente, normal) que permite transformar las normales de la textura al espacio de mundo.

La implementación de normal mapping en este proyecto permite añadir detalles como rugosidad o relieves en el faro y el terreno, mejorando significativamente la calidad visual sin impactar el rendimiento.

2.5 Frustum Culling

Frustum culling es una técnica de optimización que elimina del pipeline gráfico los objetos que no están dentro del campo de visión de la cámara. Esto reduce significativamente la carga de trabajo de la GPU al evitar el procesamiento de geometría que no será visible en la pantalla.

- **Volumen de Visión:** Está definido por seis planos (izquierdo, derecho, superior, inferior, cercano y lejano). Cada plano se representa con una ecuación general de plano:

$$ax + by + cz + d = 0$$

- **Pruebas de Intersección:** Para cada objeto, se realiza una prueba de intersección con estos planos utilizando bounding spheres o bounding boxes, descartando aquellos que están completamente fuera.

En este proyecto, frustum culling ha permitido mantener tasas de cuadros estables incluso en escenas complejas con múltiples objetos.

2.6 Movimiento de la Cámara

La cámara en este proyecto es completamente interactiva, lo que permite a los usuarios navegar por la escena utilizando teclado y mouse. El sistema emplea transformaciones basadas en ángulos de Euler para manejar la orientación de la cámara.

- **Yaw y Pitch:** Los movimientos horizontales y verticales de la cámara se controlan mediante los ángulos de yaw y pitch, que ajustan la dirección del vector frontal.

$$\vec{Front}.x = \cos(\text{Yaw}) \cdot \cos(\text{Pitch})$$

- **Proyección Perspectiva:** La cámara utiliza una proyección perspectiva para garantizar que los objetos se representen correctamente en 3D, con proporciones realistas y sensación de profundidad.

Este sistema interactivo permite a los usuarios explorar la escena de manera natural, mejorando la inmersión.

3 Resultados

3.1 Detalles del Hardware

El sistema fue desarrollado y probado en una computadora con las siguientes especificaciones:

- **Nombre del dispositivo:** YUr-PC
- **Procesador:** AMD Ryzen 5 4500U con Radeon Graphics (2.38 GHz)
- **Memoria RAM instalada:** 32.0 GB (31.4 GB utilizable)
- **ID del dispositivo:** 3D7D8F14-9772-4446-9143-6966F78936C3
- **Tipo de sistema:** Sistema operativo de 64 bits, procesador basado en x64
- **Soporte de entrada táctil:** Soporte para lápiz y 10 puntos de toque
- **Edición de Windows:** Windows 11 Home, versión 23H2
- **Fecha de instalación:** 10 de abril de 2023
- **Compilación del sistema operativo:** 22631.4602
- **Paquete de Experiencia de Windows:** Windows Feature Experience Pack 1000.22700.1055.0

3.2 Rendimiento del Renderizado

El proyecto demuestra un rendimiento fluido con tasas de cuadros superiores a 60 FPS, garantizando una experiencia visual suave y sin interrupciones incluso en configuraciones gráficas avanzadas. Esto subraya la optimización eficiente y la capacidad del sistema para manejar escenas complejas en tiempo real.

4 Discusión

El sistema desarrollado presenta un equilibrio notable entre calidad visual y rendimiento, aprovechando optimizaciones como el frustum culling y técnicas avanzadas de iluminación. Sin embargo, el uso de mapas normales introduce artefactos menores en ciertas superficies debido a errores en el cálculo del espacio tangente. En el futuro, se buscará integrar un modelo PBR completo y sombras más suaves para mejorar la calidad visual.

5 Conclusión y Trabajos Futuros

Este proyecto demuestra cómo combinar iluminación avanzada, skyboxes dinámicos y optimizaciones de rendimiento para renderizar una escena interactiva y visualmente impactante en tiempo real. En el futuro, se planea integrar modelos PBR (Physically-Based Rendering) y técnicas híbridas de rasterización y trazado de rayos.

Acknowledgments

Agradecimientos a los profesores y compañeros de la clase COMP 4046 por su apoyo y comentarios valiosos durante el desarrollo de este proyecto.

Referencias

- Marco Yu. 2024. Examen Final: Proyecto paper ACM SIGGRAPH. *GitHub Repository.* Disponible en: https://github.com/marcoyuuu/Examen-Final-Proyecto-paper-ACM-SIGGRAPH