



Departamento de Ciencia de la Computación

Computación Gráfica

Informe del Proyecto Final

Perez Choque Justo Alfredo
justo.perez@ucsp.edu.pe

Ruiz Trelles Abimael Eduardo
abimael.ruiz@ucsp.edu.pe

Profesor: Manuel Eduardo Loaiza Fernández
Grupo: CCOMP8-1

Semestre VIII
2024-1

“Los alumnos declaran haber realizado el presente trabajo de acuerdo a las normas de la Universidad Católica San Pablo”

1. Introducción

En el ámbito de la computación gráfica, la capacidad de crear entornos tridimensionales envolventes y visualmente atractivos es fundamental. Este proyecto final del curso de Computación Gráfica se centra en la aplicación práctica de OpenGL y GLFW con C++ para desarrollar una escena compleja y dinámica. El objetivo principal es integrar diversos elementos como iluminación dinámica, animación de objetos y reacciones ambientales en un entorno cohesivo. La escena incluye desde pastos que se mueve con el viento hasta la interacción de un árbol con las corrientes atmosféricas, además de la presencia de nubes estáticas y una disposición de iluminación mediante un sol y una luna giratoria. Con el uso de dos cámaras para diferentes perspectivas, y la inclusión de varios modelos detallados. Este proyecto demuestra nuestras habilidades técnicas en la programación de gráficos y nuestra capacidad para diseñar y ejecutar entornos virtuales.

2. Motivación

La motivación radica en explorar el poder de la computación gráfica para crear mundos virtuales inmersivos. Además, se busca **aprender** sobre las capacidades de OpenGL y GLFW para desarrollar entornos visualmente atractivos.

El desafío de simular la interacción entre elementos naturales, como el viento y el pasto, junto con la implementación de modelos de animales, ofrece una oportunidad única para aplicar conceptos avanzados de gráficos. La inclusión de un sol y una luna en rotación alrededor de la escena añade un nivel adicional de complejidad al crear variaciones de iluminación que resaltan las texturas y detalles de cada componente.

Esta exploración no solo busca enriquecer nuestro conocimiento técnico, sino también profundizar nuestra comprensión de cómo la computación gráfica puede transformar ideas abstractas en experiencias visuales vívidas y convincentes.

3. Descripción del Proyecto

El proyecto incorpora elementos dinámicos como pasto que reacciona al viento y un árbol que se mueve con las corrientes de aire, así como nubes estáticas en el cielo. La iluminación dinámica, influenciada por un sol y una luna en rotación, añade variaciones atmosféricas que realzan la textura y profundidad de cada componente. Además, la escena incluye una casa y modelos detallados de un panda, un tigre, un zorro y una llama, todos iluminados para destacar su presencia.

3.1. Herramientas

Para el desarrollo de este proyecto se utilizaron varias bibliotecas y herramientas avanzadas de C++ y OpenGL, con el propósito de crear una escena gráfica rica en detalles y animaciones. A continuación se describen las herramientas y bibliotecas utilizadas:

El proyecto se basa en la utilización de tres repositorios principales, cada uno encargado de diferentes aspectos de la escena:

- **Repositorio del Árbol[1]:** Este repositorio proporcionó el algoritmo de generación de árboles detallados junto a sus funcionalidades con el viento.
- **Repositorio del Pasto[5]:** Se utilizó para integrar un sistema de pasto dinámico que reacciona a las fuerzas del viento.
- **Repositorio del Sol y la Luna[4]:** Este repositorio facilitó la implementación de únicamente el sol y la luna.

Las bibliotecas principales usadas para este proyecto fueron:

- **C++ y OpenGL:** El núcleo del proyecto se basa en C++ con OpenGL para la representación gráfica. La biblioteca GLFW se utilizó para manejar la ventana y los eventos de entrada, mientras que GLAD se encargó de la carga de las funciones de OpenGL.
- **GLFW:** Biblioteca utilizada para el manejo de ventanas, contextos OpenGL y eventos de entrada, lo que facilita la creación de aplicaciones gráficas multiplataforma.
- **GLAD:** Es un cargador de funciones de OpenGL. Se utilizó para gestionar la inicialización de las funciones de OpenGL.
- **GLM (OpenGL Mathematics):** Biblioteca matemática utilizada para manejar vectores y matrices. En este proyecto, se utilizó principalmente para transformaciones, como la perspectiva y la manipulación de las coordenadas de los modelos.
- **DevIL (Developer's Image Library):** Biblioteca para el manejo de imágenes, utilizada para cargar y manipular texturas.

Además, se utilizaron modelos .obj con .dds y junto con sus respectivas texturas para representar los animales y la casa en la escena. Los modelos son: panda, tigre, casa, nubes, llama y zorro. Y el muy bien documentado tutorial de OpenGL[2].

3.2. Matemática/Animación

3.2.1. Pasto

- **Optimización de polígonos:** Al crear una escena 3D con vegetación, es crucial optimizar el número de polígonos para mejorar el rendimiento. Para modelar pasto, utilizamos billboards y Niveles de Detalle (LOD).
- **Posición tridimensional de la base:** La posición tridimensional de la base de los vértices del pasto se define como:

$$\text{base_position} = (x, y, z)$$

- **Posición final proyectada:** Calculamos la posición final en el espacio proyectado para cada vértice del quad basado en la posición de la base y las posiciones relativas de los vértices:

$$gl_Position = u_projection \cdot u_view \cdot (\text{base_position} + \text{vertexPosition}[i])$$

- **Aplicación de texturas:** Aplicamos texturas a los quads del pasto. Las coordenadas de textura para cada vértice del quad se especifican como:

$$\text{textCoords}[i] = (s_i, t_i) \quad \text{para } i = 0, 1, 2, 3$$

- **Simulación del volumen:** Para simular el volumen de los mechones de pasto, creamos configuraciones cruzadas de quads y los rotamos aleatoriamente en el eje Y:

$$\text{modelRandY} = \text{rotationY}(\text{random}(\text{base_position}.zx) \cdot \pi)$$

- **Nivel de Detalle (LOD):** Implementamos LOD para ajustar dinámicamente la cantidad de quads según la distancia a la cámara:

$$\text{dist_length} = \|\text{base_position} - \text{u_cameraPosition}\|$$

- **Animación de viento:** Introducimos animación de viento mediante una textura de flujo que indica la dirección y fuerza del viento. Ajustamos la posición de los quads en el geometry shader para simular el movimiento del pasto:

$$uv = \frac{\text{base_position}.xz}{10} + \text{windDirection} \cdot \text{windStrength} \cdot \text{u_time}$$

3.2.2. Arbol

- **Movimiento de las Hojas**

El movimiento de las hojas se actualiza de acuerdo con los fotogramas (*frames*). La velocidad de rotación en los ejes *x* y *y*, es decir, qué tan rápido giran, puede ser representada como:

$$\text{wind_offset}[0] = \text{wind_offset}[0] + \frac{\text{time_delta}}{8}$$

$$\text{wind_offset}[1] = \text{wind_offset}[1] + \frac{\text{time_delta}}{12}$$

Donde *time_delta* representa el intervalo de tiempo entre fotogramas.

- **Sistema L-System para Árboles**

El sistema L-System realiza transformaciones aleatorias de rotación, traslación y escala, con parámetros que incluyen el nivel (*level*) y la cantidad de ramas (*max_level*) utilizando recursión.

```
add_sub_trunks_lsystem_random(t,level,max_level,is_end_point)
    t0 = transformación base
    t1 = transformación para la rama superior
    b0 = transformación para las ramas laterales / ramitas
    b1 = transformación para las ramas laterales / ramitas
    b2 = transformación para las ramas laterales / ramitas
```

Donde *t0*, *t1*, *b0*, *b1* y *b2* son transformaciones que incluyen traslación, rotación y escala, afectadas por valores aleatorios (*rand_vals*) para generar variabilidad en la forma de las ramas y el tronco.

3.2.3. Sol y Luna

El sol y la luna giran alrededor del escenario siguiendo una trayectoria elíptica. La posición del sol se calcula utilizando las siguientes fórmulas:

$$\begin{aligned} posXSol &= 0; \\ posYSol &= \text{ellipseRadiusY} \cdot \cos(\text{rotationAngle}); \\ posZSol &= -\text{ellipseRadiusZ} \cdot \sin(\text{rotationAngle}); \end{aligned}$$

Donde:

- *ellipseRadiusY*: Radio de la elipse en el eje *y*.
- *ellipseRadiusZ*: Radio de la elipse en el eje *z*.
- *rotationAngle*: Ángulo de rotación del sol y la luna alrededor del escenario.

Estas ecuaciones describen cómo se calcula la posición del sol y la luna en relación con su movimiento elíptico alrededor del escenario.

3.3. Shaders

3.3.1. Pasto

El pasto se modela utilizando shaders en OpenGL para optimizar el rendimiento y la apariencia visual. En el vertex shader, los puntos base del pasto, generados en una cuadrícula, son procesados para determinar la posición final de cada vértice del quad. Luego, el geometry shader transforma estos puntos en quads que representan los mechones de pasto. Finalmente, en el fragment shader se calcula el color de cada píxel de los quads, permitiendo aplicar texturas y efectos visuales adicionales como la reacción del pasto al viento.

3.3.2. Arbol

El vertex shader se encarga de transformar las posiciones locales al espacio global, calcular normales transformadas y preparar las coordenadas de textura y sombra. Por otro lado, el fragment shader realiza interpolación bilineal para obtener el color de la textura del tronco, utilizando una función de mezcla para ajustar entre valores de textura alta y baja. Además, aplica iluminación ambiental, difusa y especular junto con sombreado para determinar el color final del tronco. Los shaders de las hojas, enfocados en rotaciones para simular movimiento, no aplican interpolación bilineal adicional en el fragment shader, priorizando la aplicación estándar de iluminación. Este diseño optimiza la calidad visual y el rendimiento en el contexto de OpenGL.

3.3.3. Sol y Luna

Los shaders del sol y la luna son básicos y funcionan como fuentes de luz. El vertex shader transforma las coordenadas de vértice al espacio de vista y calcula las normales, mientras que el fragment shader aplica iluminación ambiental, difusa y especular. Utiliza una textura opcional y modula el color del objeto esférico según las propiedades de la luz y el material, incluyendo la reflexión especular basada en la posición de la cámara y la luz incidente.

4. Funcionalidades de su programa (Keyboard, movimiento de cámara, iluminación)

- **Movimiento de cámara:**

- **ASDW:** Permite mover la cámara en las direcciones izquierda, derecha, adelante y atrás respectivamente, facilitando la navegación por el entorno tridimensional.
- **Shift Derecho:** Eleva la cámara, proporcionando una vista superior del escenario.
- **SPACEBAR:** Baja la cámara, permitiendo una vista inferior o cercana al suelo.

- **Generación de elementos:**

- **R:** Genera nuevos árboles de manera procedural, añadiendo variabilidad y dinamismo al entorno gráfico.

5. Problemas durante la implementación

- **No hay un estándar para .OBJ:** Inconsistencia en el formato y estructura de archivos .OBJ descargados, dificultando la integración y procesamiento uniforme.
- **Unir los repositorios:** Necesidad de integrar y gestionar eficientemente tres repositorios separados que contienen modelos de árbol, césped, sol y luna.
- **Curva de aprendizaje para modificar y juntar cada uno:** Esfuerzo significativo requerido para comprender y modificar los modelos descargados, así como para integrarlos correctamente en la escena.

- **Iluminación inicial con GOURAD cambiada a Phong:** Ajustes y problemas encontrados al migrar de un modelo de iluminación GOURAD a uno más complejo como Phong en los shaders.
- **Conflicto con glad y glew, conversión a glad:** Dificultades surgidas durante la transición de GLEW a glad para la gestión de extensiones OpenGL, implicando ajustes y correcciones extensivas.
- **Sincronización de luces:** Necesidad de optimizar y coordinar adecuadamente múltiples fuentes de luz en la escena para asegurar una representación visual coherente y efectiva.

6. Experimentos

En esta sección, describimos los experimentos realizados para optimizar y ajustar los elementos visuales del proyecto utilizando OpenGL y GLFW.

6.0.1. Pasto

Se configuró el Nivel de Detalle (LOD) del pasto para controlar su generación en función de la distancia a la cámara. A distancias lejanas, el LOD se ajustó para reducir o eliminar la generación de pasto, minimizando así la carga computacional y optimizando el rendimiento del sistema.



Figura 1: Ejemplo de configuración de LOD para el pasto.

6.0.2. Hojas del arbol

Para el árbol, se ajustó la cantidad de hojas generadas mediante un L-system. Se encontró que un alto número de hojas afectaba negativamente la iluminación y el rendimiento, por lo que se estableció un valor de 0 como nivel inicial y 6 como el nivel máximo, esto para equilibrar el detalle visual con la eficiencia computacional.

Además, se modificó la generación del tronco para evitar ramas gruesas que comprometieran el realismo visual. El ajuste se realizó con un valor de 1,1, permitiendo la creación de troncos más naturales y estéticamente agradables.

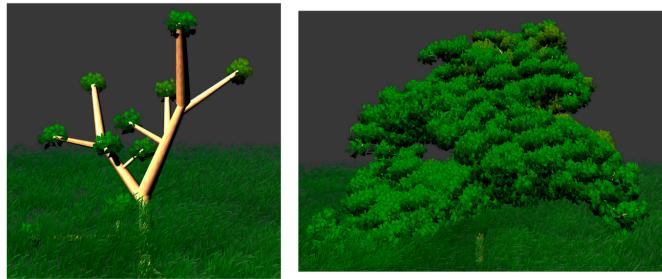


Figura 2: Configuración del tronco del árbol para mejorar el realismo visual.

6.0.3. Shader del Tronco

El fragment shader del tronco fue ajustado para proporcionar una iluminación sutil y realista. Se asignó un valor de $(0,4,0,25,0,2)$ para *bark_low* y $(0,6,0,4,0,2)$ para *barkhigh* para mantener la cohesión visual con el entorno general del proyecto, resaltando texturas y detalles sin saturar la escena con una iluminación excesiva.

Estos experimentos demostraron la capacidad de configurar elementos gráficos complejos con precisión, equilibrando eficiencia y calidad visual para crear un entorno virtual convincente y funcional.

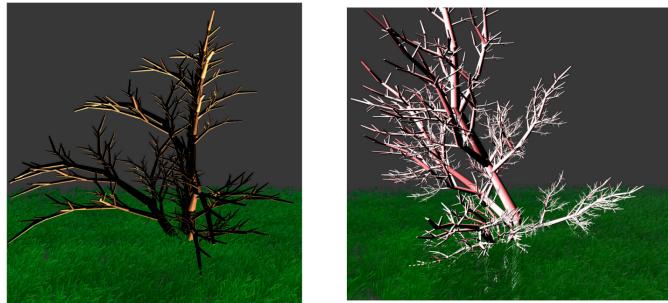


Figura 3: Configuración del tronco aplicado al shader.

7. Resultados

El código fuente se puede encontrar en [3]. Con la configuración presentada anteriormente, se evaluaron los resultados obtenidos en términos de iluminación y renderizado para los diferentes elementos del proyecto.

El pasto se ilumina únicamente con iluminación ambiental, lo cual proporciona una apariencia natural y coherente en todo tipo de condiciones lumínicas. El uso del Nivel de Detalle (LOD) permitió optimizar la generación del pasto, asegurando que se representara adecuadamente sin comprometer el rendimiento del sistema.

Para el árbol, se implementó una combinación de iluminación ambiental y difusa. Esto permitió resaltar los detalles de las hojas y el tronco de manera realista, adaptándose a las variaciones de iluminación ambiental en la escena.

Todos los demás objetos, como la casa, el panda, el tigre, el zorro y la llama, fueron renderizados utilizando el modelo de iluminación Phong. Esta técnica proporciona un equilibrio entre los efectos de iluminación especular, difusa y ambiental, garantizando una representación visual detallada y estéticamente atractiva.

Estos resultados demuestran la eficacia de las técnicas de iluminación implementadas y la configuración óptima de parámetros para cada elemento del entorno virtual, contribuyendo a la creación de un ambiente gráfico cohesivo y visualmente impresionante.

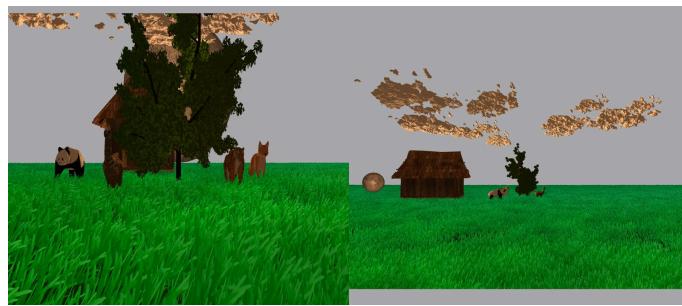


Figura 4: Resultado de día.



Figura 5: Resultado de noche.

8. Próximos Pasos

- **Implementación de sombras:** Integrar un sistema de sombreado dinámico para mejorar la percepción de profundidad y realismo en la escena.
- **Animación de los animales:** Desarrollar y aplicar animaciones detalladas para los modelos de animales, mejorando la interactividad y el dinamismo del entorno.
- **Mejora en la iluminación solar para un efecto más realista:** Optimizar la representación visual del sol para lograr efectos de luz más naturales y dramáticos en la escena.
- **Realismo en el entorno, particularmente en el terreno:** Refinar la representación del terreno (como tierra), especialmente el pasto, para aumentar el nivel de detalle y realismo visual.

9. Conclusiones

El proyecto de Computación Gráfica utilizando OpenGL y GLFW ha demostrado la efectividad en la creación de un entorno virtual realista. La optimización mediante técnicas como el Nivel de Detalle (LOD) y la configuración precisa de shaders permitieron equilibrar eficiencia y calidad visual.

La integración de iluminación dinámica y shaders adaptativos proporcionó una experiencia visual envolvente, destacando la capacidad para gestionar recursos gráficos eficazmente. La experimentación con parámetros como la densidad de hojas en árboles y la iluminación del tronco subrayó la importancia de ajustar detalles visuales sin comprometer el rendimiento.

En resumen, este proyecto no solo fortaleció el dominio de OpenGL y GLFW, sino que también ilustró cómo aplicar conceptos avanzados de computación gráfica para crear entornos virtuales convincentes y educativos.

Referencias

- [1] Damien Doy. *Tutorials OpenGL with code*. en. <https://github.com/damdoy/opengl-examples/tree/master/tree>.
- [2] *Learn OpenGL, extensive tutorial resource for learning Modern OpenGL*. en. <https://learnopengl.com>. Accessed: 2024-7-1.
- [3] *Nuestro Proyecto Final*. en. <https://github.com/flechita-veloz/Rotacion-de-sol-y-luna-en-un-paisaje-con-OpenGL>. Accessed: 2024-7-1.
- [4] *OpenGL Sphere, Sun and Moon*. en. https://www.songho.ca/opengl/gl_sphere.html. Accessed: 2024-7-1.
- [5] Svetlana. *grass-tutorial_finalcode: Final code of the grass modeling tutorial. This program displays an expanse of windswept grass*. en. https://github.com/Vulpinii/grass-tutorial_finalcode.