320018277
GPO: 05

PROYECTO FINAL 2026-1

Montiel Juárez Oscar Iván

24 de noviembre de 2025

1. Alcance del Proyecto

El objetivo principal de este proyecto es la creación de una escena 3D interactiva en tiempo real utilizando la API gráfica OpenGL 4.0 y el lenguaje de programación C++. La escena es una recreación temática de la casa y el cuarto de los protagonistas de la serie animada "Phineas y Ferb".

El alcance del proyecto abarca las siguientes áreas clave:

- **Modelado y Escenografía:** Creación de la icónica casa (exterior) y el cuarto de los protagonistas, complementado con 7 objetos característicos de la serie modelados individualmente.
- **Gestión de Activos (Assets):** Integración de una variedad de activos, incluyendo:
  - Modelos 3D de terceros (Pez, Cabina Telefónica) obtenidos de *TurboSquid* bajo licencia *Free Non Profit*.
  - Texturas para dichos modelos creadas manualmente por el autor.
  - Texturas para el entorno (paredes, piso) generadas mediante herramientas de Inteligencia Artificial, con licencia *Free Non Profit*.
  - Un fondo de cielo *Skydome* con textura HDR (*High Dynamic Range*) obtenida de *PolyHeaven* (Çitrus Orchard Road"por Jarod Guest y Dimitrios Sawa), bajo licencia *Free Non Profit*.
- **Animación Kinemática:** Implementación de 5 sistemas de animación distintos, activados por el usuario, que se dividen en:
  - **Animaciones Complejas (3):**
    1. **Ciclo Día/Noche:** Un sistema integral donde un modelo de Sol rota, su luz puntual se intensifica y atenúa, la luz ambiental de la escena cambia, y una lámpara interna se enciende de forma automática durante la "noche".
    2. **Nubes de Techo:** Una animación basada en shaders que desplaza las coordenadas UV de la textura del techo, simulando nubes en movimiento.
    3. **Péndulo de Newton:** Una animación kinemática por fases que simula el movimiento del péndulo, sincronizada con un efecto de sonido personalizado.
  - **Animaciones Simples (2):**
    1. **Nado del Pez:** Una animación compuesta que incluye traslación del pez, un giro de 180° (flip) para cambiar de dirección, y una oscilación de la aleta caudal.
    2. **Puertas y Cortinas:** Un sistema de animación basado en estados (abierto/cerrado) que aplica transformaciones de traslación y rotación a múltiples objetos simultáneamente.
- **Iluminación y Renderizado:**
  - Implementación de un *Skydome* con una textura HDR y un shader personalizado.
  - Un sistema de iluminación dinámico (modelo Blinn-Phong) que gestiona una luz direccional y cuatro luces puntuales, con estado dinámico.
  - Renderizado de objetos opacos y translúcidos (vidrios, pecera, sol) mediante el manejo de *blending* y el búfer de profundidad.
- **Interactividad y Sonido:**

- Control de una cámara libre en primera persona (*fly-cam*) mediante el teclado (WASD) y el ratón.
- Controles de teclado para activar y desactivar las 5 animaciones.
- Inclusión de un efecto de sonido `.wav` (creado por el autor) para el péndulo, utilizando la biblioteca `windows.h`.

2. Estado del Arte en Gráficos por Computadora

El campo de los gráficos por computadora en tiempo real ha experimentado avances significativos en los últimos años. Este proyecto se sitúa en el contexto de las tecnologías modernas mientras mantiene un enfoque educativo accesible.

2.1. Tecnologías de Vanguardia en la Industria

En la industria profesional actual, los estándares de calidad incluyen:

- **Ray Tracing en Tiempo Real:** Tecnología implementada en tarjetas gráficas RTX que simula el comportamiento físico de la luz, permitiendo reflexiones, refracciones y sombras más realistas.
- **DLSS/FSR:** Técnicas de upscaling inteligente que usan inteligencia artificial para mejorar el rendimiento manteniendo la calidad visual.
- **Physically Based Rendering (PBR):** Modelos de materiales que simulan con precisión cómo interactúa la luz con diferentes superficies basándose en principios físicos reales.
- **Global Illumination:** Técnicas que calculan no solo la luz directa sino también la luz indirecta que rebota entre las superficies.
- **Simulaciones Físicas en Tiempo Real:** Motores de física avanzados que calculan colisiones, fluidos, telas y partículas de manera realista.

2.2. Posicionamiento del Proyecto

Si bien este proyecto no implementa todas las tecnologías de vanguardia mencionadas, representa un paso significativo en el aprendizaje y aplicación de conceptos fundamentales:

- **Shader Programming:** El uso de shaders personalizados para animaciones (nubes) y efectos especiales (skydome HDR) demuestra comprensión de conceptos fundamentales en gráficos modernos.
- **Iluminación Dinámica:** El sistema de ciclo día/noche implementado, aunque básico comparado con soluciones profesionales, incorpora principios de iluminación dinámica que son esenciales en motores gráficos modernos.
- **Gestión de Recursos:** La integración de texturas HDR y el manejo eficiente de múltiples modelos 3D refleja prácticas estándar en el desarrollo de aplicaciones gráficas.
- **Animaciones Programáticas:** Las animaciones kinemáticas implementadas, aunque no son simulaciones físicas completas, muestran el entendimiento de interpolaciones y transformaciones que son la base de sistemas de animación más complejos.

2.3. Brecha Tecnológica y Oportunidades de Mejora

La principal brecha entre este proyecto y las soluciones profesionales radica en:

- La ausencia de un motor de física completo para simulaciones dinámicas
- El uso de iluminación basada en modelos simplificados en lugar de PBR
- La falta de técnicas de optimización avanzadas como Level of Detail (LOD) y frustum culling
- La implementación manual de sistemas que en entornos profesionales son manejados por motores gráficos establecidos

Sin embargo, este proyecto sirve como base fundamental para entender los principios que subyacen a las tecnologías más avanzadas, proporcionando una plataforma sólida para futuras especializaciones en motores gráficos como Unreal Engine 5 o Unity, los cuales construyen sobre estos mismos conceptos fundamentales pero con implementaciones optimizadas y características adicionales.

3. Limitantes

A pesar de los resultados obtenidos, el proyecto se encuentra con las siguientes limitantes técnicas y de alcance:

- **Optimización y Rendimiento:** El proyecto está optimizado para su ejecución en una sola máquina (el equipo de desarrollo). No se garantiza un *framerate* estable en hardware con especificaciones inferiores.
- **Simulación Física:** Todas las animaciones son representaciones **kinemáticas** (siguen un guion predefinido) y no **dinámicas**. No se implementó un motor de física para calcular colisiones, gravedad o transferencia de momento real (ej. en el péndulo).
- **Portabilidad:** El código fuente utiliza bibliotecas específicas de Windows (`windows.h` y `winmm.lib`) para la reproducción de sonido. El proyecto no es portable a macOS o Linux sin una refactorización de esta funcionalidad.
- **Licenciamiento de Activos:** El uso de modelos y texturas bajo licencias *Free Non Profit* restringe el uso de este proyecto estrictamente a fines académicos y no comerciales.
- **Renderizado Avanzado:** El *tone mapping* (mapeo de tonos) solo se aplica de forma básica en el *Skydome*. La escena principal no utiliza un *pipeline* HDR completo, por lo que la alta intensidad del cielo no ïlumina"globalmente la escena, y los brillos intensos (como el sol o la lámpara) se saturan a blanco (1.0).

4. Metodología de Software Aplicada

Para un proyecto de esta naturaleza, con requisitos visuales y funcionales que evolucionan constantemente ("hacer la luz más brillante", çambiar la rotación del sol"), se aplicó una **Metodología de Desarrollo Iterativo e Incremental**.

Este enfoque permite construir el software por módulos funcionales, probando cada componente antes de integrar el siguiente. El flujo de trabajo del proyecto fue el siguiente:

1. **Fase 1: Configuración y Prototipo Base**
    - Configuración del entorno (GLEW, GLFW, GLM, `stb_image`).
    - Creación de la ventana y las clases auxiliares `Shader`, `Camera` y `Model`.
    - Implementación de la cámara libre y carga de un modelo de prueba.
2. **Fase 2: Carga de Escena Estática**
    - Carga de todos los modelos estáticos (casa, cuarto, muebles) en sus posiciones.
    - Implementación de un sistema de iluminación estático (una luz direccional) y renderizado de transparencias (ventanas).
3. **Fase 3: Desarrollo Incremental de Animaciones (Iteraciones)**
    - **Iteración 1 (Cortinas y Puertas):** Se implementa la lógica de animación más simple. Se define un *flag* booleano (`cortinaCerrada`) y un `factorCierre` (0.0 o 1.0). En el bucle de dibujado, se aplica una transformación `glm::translate` o `glm::rotate` multiplicada por este factor.
    - **Iteración 2 (Pez):** Se implementa una animación compuesta. Se añaden variables de estado (`fishSwimming`, `fishXOffset`, `fishFlipped`, `tailAngle`). Se desarrolla la lógica de traslación, la máquina de estados para el "flip"de 180° y la oscilación de la cola basada en `sin()`.
    - **Iteración 3 (Péndulo):** Se implementa una animación kinemática por fases (`currentPhase`, `phaseTime`) usando funciones trigonométricas. Se añade el sonido `PlaySound` en el cambio de fase.
    - **Iteración 4 (Nubes):** Se crea un nuevo par de shaders (`clouds.vs/frag`). La animación se delega a la GPU, pasando `time` y `cloudsMoving` como *uniforms*.
    - **Iteración 5 (Ciclo Día/Noche y Skydome):** La iteración más compleja.
        - Se crea el `skydomeShader` y se implementa la lógica de renderizado HDR (`stb_loadf`, `GL_RGB16F`).
        - Se añade el modelo del Sol y su rotación basada en `sunAngle`.
        - Se refactoriza el bloque de iluminación de `main.cpp` para hacerlo dinámico, creando el `dayFactor` y la lógica de interruptor (Sol vs. Lámpara).
4. **Fase 4: Refinamiento y Documentación**
    - Ajuste fino de todos los parámetros (velocidad, color, intensidad de luz).
    - Depuración de orden de renderizado (translúcidos) y gestión de texturas (problema de "sangrado"de texturas).
    - Redacción de la documentación técnica.

5. Planificación del Proyecto (Diagrama de Gantt)

Para gestionar el desarrollo del proyecto, se utilizó un Diagrama de Gantt que desglosa las tareas a lo largo de 8 semanas, desde el 17 de septiembre hasta el 11 de noviembre, más una semana adicional para la entrega. Este enfoque permitió visualizar la carga de trabajo y asegurar que tanto el modelado como la programación de animaciones se completaran a tiempo.

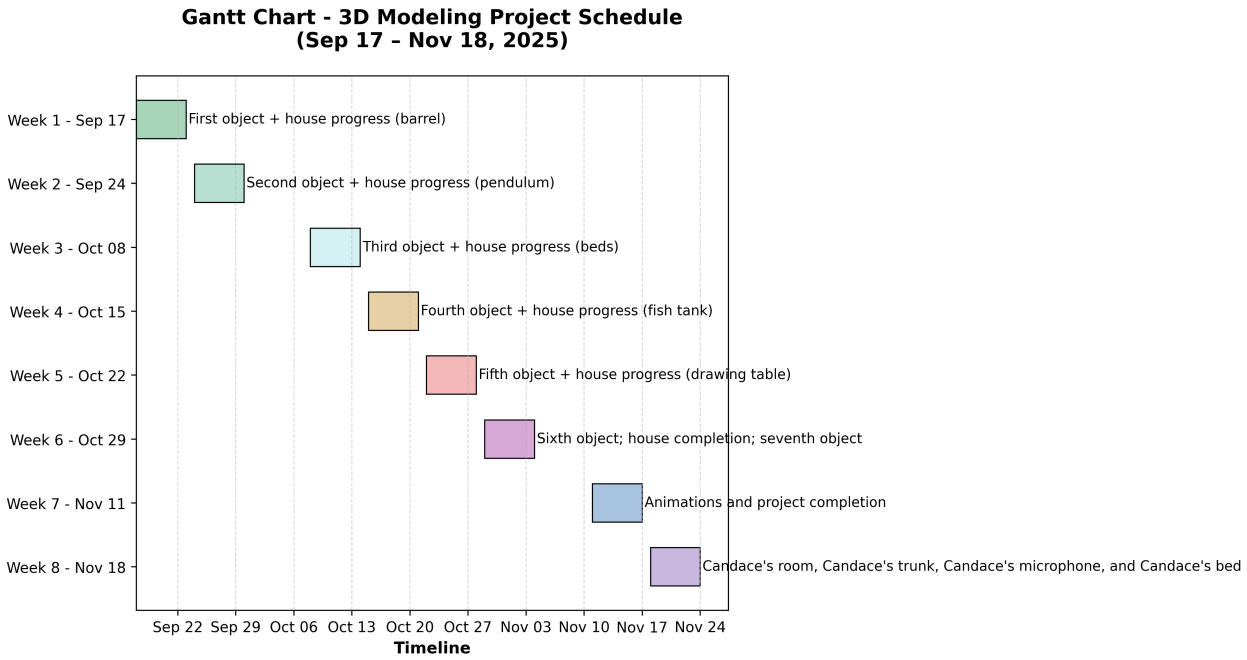


Figura 1: Diagrama de Gantt del Proyecto, mostrando la distribución de tareas en 8 semanas.

Como se observa en el diagrama (Figura **??**), la metodología de trabajo fue incremental:

- **Semanas 1 a 5:** Se dedicaron a la creación progresiva de los 7 objetos requeridos, asignando aproximadamente una semana a cada objeto principal (Barril, Péndulo, Camas, Pecera, Mesa de dibujo). En paralelo, se trabajaron los avances generales de la casa (estructura, paredes, interiores, cortinas).
- **Semana 6:** Fue una semana crucial de integración, donde se finalizaron los últimos objetos (Escritorio y Lámpara) y se completó el modelado de la casa al 100 %.
- **Semana 7:** Se reservó por completo para la implementación de las 5 animaciones (Péndulo, Pez, Sol, Nubes/Cortinas), las pruebas finales y la preparación de la entrega.
- **Semana 8:** Se realizó el modelado del cuarto de Candace, incluyendo el baúl y el micrófono. Además, se pasaron dos objetos (la cabina y el escritorio de Phineas) al cuarto de Candace para enriquecer la escena.

Esta planificación se alinea con la metodología iterativa, permitiendo construir la escena por partes funcionales (modelos) antes de integrar la capa final de complejidad (animaciones).

6. Análisis de Costos

En este apartado se presenta un desglose de los costos asociados al desarrollo del proyecto, así como una propuesta de precio de venta. Es importante destacar que el proyecto fue realizado por un único desarrollador (el autor) y que se utilizaron herramientas y activos bajo licencias gratuitas para fines educativos.

6.1. Costos de Desarrollo

- **Mano de obra:** El proyecto tomó 8 semanas de desarrollo activo, más aproximadamente 4 meses de aprendizaje y teoría (duración de un semestre). Considerando que este es mi primer proyecto profesional y busco establecer mi portafolio en el mercado, se ha valorado mi tiempo a una tarifa de entrada de $25 MXN/hora. Esta tarifa refleja mi nivel actual como estudiante que busca ganar experiencia y construir un portafolio sólido. El cálculo de horas es:
  - Desarrollo: 8 semanas $\times$ 20 horas/semana = 160 horas.
  - Aprendizaje: 16 semanas $\times$ 10 horas/semana = 160 horas.
  - Total de horas: $160 + 160 = 320$ horas.

  Costo de mano de obra: 320 horas $\times$ 25 MXN/hora = $8,000$ MXN.
- **Equipo:** Se utilizó una computadora con tarjeta gráfica 3050ti, Ryzen 7 de nueva generación y enfriamiento líquido. Considerando un costo de la computadora de $24,000 MXN y una vida útil de 3 años (156 semanas), el costo semanal de depreciación es $24,000/156 \approx 153{,}85$ MXN/semana. El proyecto tomó 8 semanas de desarrollo, por lo que el costo de depreciación es $8 \times 153{,}85 \approx 1,230{,}80$ MXN.
- **Electricidad:** Considerando que la computadora consume aproximadamente 500W bajo carga y se usó 20 horas a la semana durante 8 semanas, el consumo total de energía es $500W \times$ 20 horas/semana $\times$ 8 semanas = 80 kWh. Asumiendo un costo de $3.00 MXN/kWh, el costo de electricidad es $80 \times 3{,}00 = 240$ MXN.
- **Software:**
  - **Autodesk Maya:** Se utilizó la versión educativa gratuita. Costo: $0 MXN.
  - **Blender:** Aunque no se utilizó, es una alternativa gratuita. Costo: $0 MXN.
  - **Visual Studio:** Entorno de desarrollo gratuito. Costo: $0 MXN.
  - **OpenGL:** API gráfica gratuita. Costo: $0 MXN.
- **Activos:**
  - **Modelos 3D:** Los modelos de pez y cabina telefónica se obtuvieron de TurboSquid bajo licencia Free Non Profit. Costo: $0 MXN.
  - **Texturas:** Las texturas para los modelos y el entorno se crearon manualmente o se generaron con IA bajo licencia Free Non Profit. Costo: $0 MXN.
  - **Skydome HDR:** Obtenido de PolyHeaven bajo licencia Free Non Profit. Costo: $0 MXN.

6.2. Total de Costos

- Mano de obra: $8,000 MXN
- Equipo (depreciación): $1,230.80 MXN
- Electricidad: $240 MXN
- Software: $0 MXN
- Activos: $0 MXN
- **Total:** $9,470.80 MXN

6.3. Precio de Venta

Considerando que este es mi primer proyecto profesional y busco establecer mi presencia en el mercado, se ha optado por una estrategia de precios de entrada. Esta estrategia reconoce que, aunque el valor real del trabajo es mayor, es importante ofrecer un precio competitivo para ganar experiencia y construir un portafolio sólido.

- Costo total: $9,470.80 MXN
- Margen de ganancia (20 %): $1,894.16 MXN
- **Precio de venta sugerido:** $11,364.96 MXN (aproximadamente $635 USD)

Sin embargo, entendiendo la naturaleza de proyecto inicial y mi objetivo de ingresar al mercado, se podría considerar un precio promocional de $9,500 MXN (aproximadamente $530 USD) para clientes interesados en este tipo de desarrollos.

6.4. Justificación del Precio de Entrada

El precio reducido se justifica por las siguientes razones:

- **Primer proyecto profesional:** Como desarrollador que ingresa al mercado, reconozco que mi valor principal en este momento es la demostración de habilidades técnicas más que la experiencia extensiva.
- **Inversión en portafolio:** Este proyecto representa una inversión en mi portafolio personal, por lo que estoy dispuesto a aceptar un retorno financiero menor a cambio de ganar exposición y experiencia.
- **Competitividad:** En el mercado de desarrolladores junior, es esencial ofrecer precios competitivos que reflejen tanto la calidad del trabajo como la etapa profesional actual.
- **Relación calidad-precio:** Aunque el precio es de entrada, el proyecto demuestra habilidades avanzadas en gráficos por computadora, incluyendo animaciones complejas, shaders personalizados y gestión de recursos, ofreciendo excelente valor por el precio.
- **Accesibilidad:** Al mantener un precio bajo, busco hacer el proyecto accesible para clientes o estudios que puedan estar interesados en contratar a un desarrollador junior con habilidades demostradas.

6.5. Justificación del Uso de Autodesk Maya

Se eligió Autodesk Maya para el modelado 3D por las siguientes razones:

- **Licencia educativa gratuita:** Como estudiante, tengo acceso a una licencia gratuita de Maya, lo que permite aprovechar una herramienta industrial sin costo.
- **Interfaz y flujo de trabajo:** Maya es ampliamente utilizado en la industria del cine y los videojuegos, y aprender su flujo de trabajo es valioso para mi formación profesional.
- **Herramientas de modelado:** Maya ofrece un conjunto robusto de herramientas de modelado, texturizado y animación que son estándar en la industria.

- **Integración con pipelines:** Aunque en este proyecto no se utilizó un pipeline complejo, Maya permite una integración con motores de juego y software de renderizado avanzado.
- **Preparación para el mercado laboral:** El dominio de Maya es una habilidad valorada en estudios profesionales, por lo que su uso en este proyecto contribuye a mi preparación para oportunidades laborales futuras.

Aunque Blender es una alternativa gratuita y de código abierto muy capaz, la decisión de usar Maya se basó en la disponibilidad de la licencia educativa y el deseo de adquirir experiencia en una herramienta utilizada ampliamente en estudios profesionales.

7. Diagrama de Flujo del Software

El flujo de ejecución del programa se centra en un bucle principal (Game Loop) que se ejecuta en cada fotograma. Este bucle es responsable de actualizar el estado, procesar la entrada del usuario y renderizar la escena.

Nota: El diagrama de flujo visual completo, generado a partir del código fuente, se anexa al final de este documento.

El flujo lógico del programa, como se detalla en el diagrama, sigue los siguientes pasos principales:

1. **Inicio y Configuración:**
   - El programa inicia e inicializa las bibliotecas fundamentales: `srand` para aleatoriedad, `glfwInit` para la ventana y `glewInit` para las funciones de OpenGL.
   - Se crean la ventana y se configuran los *callbacks* para el teclado y el ratón.
   - Se cargan todos los recursos en memoria: Shaders (lighting, skydome, clouds), Modelos 3D (.obj), y Texturas (incluyendo la HDR para el cielo y las texturas de utilería blanco/-negro).
2. **Bucle Principal (`while` loop):**
   - El programa entra en el bucle principal mientras la ventana no deba cerrarse.
   - **Actualización de Lógica:** Se calcula `deltaTime`. Se comprueba el estado de todas las animaciones activas (Péndulo, Nubes, Pez, Sol) y se actualizan sus variables de estado (tiempos, ángulos, posiciones).
   - **Procesamiento de Entrada:** Se procesan los eventos del teclado (WASD, teclas 1-5) y se actualiza la cámara.
3. **Pipeline de Renderizado (Por Fotograma):**
   - **Paso 1: Limpiar Pantalla.** Se limpia el búfer de color y profundidad (`glClear`).
   - **Paso 2: Dibujar Skydome.** Se activa el `skydomeShader` y se dibuja la esfera del cielo primero, usando una prueba de profundidad de `GL_LEQUAL` para asegurar que siempre quede en el fondo.
   - **Paso 3: Configurar Luces.** Se activa el `lightingShader`. Se ejecuta la lógica de `if (sunAnimationActive)` para determinar la intensidad de la luz direccional y de las luces puntuales (Sol y Lámpara).
   - **Paso 4: Dibujar Opacos.** Con `glDepthMask(TRUE)` y `glDisable(BLEND)`, se dibujan todos los objetos sólidos (casa, muebles, pasto, etc.).

- **Paso 5: Dibujar Translúcidos.** Se activa `glEnable(BLEND)` y se deshabilita la escritura de profundidad (`glDepthMask(FALSE)`). Se dibujan en orden los vidrios, la pecera y el modelo del sol.
- **Paso 6: Intercambiar Búferes.** Se muestra la imagen renderizada en la pantalla (`glfwSwapBuffers`).

4. **Fin del Programa:**
   - Al salir del bucle (usuario cierra la ventana), se liberan todos los recursos (texturas) y se termina GLFW.

8. Documentación del Código

La documentación técnica no consiste en explicar cada línea de código, sino en describir la **arquitectura** de los sistemas clave y la **lógica** detrás de su implementación.

8.1. Gestión de Activos y Licenciamiento

Un componente fundamental del proyecto es la correcta gestión de los activos 3D.

- **Modelos de Internet:** Se utilizaron dos modelos de la plataforma *TurboSquid* bajo licencia *Free Non Profit* sin atribución de autor directa: el pez y la cabina telefónica. **Las texturas de estos modelos fueron creadas manualmente por el autor**.
- **Texturas de IA:** Las texturas de la escena (paredes, pisos) se generaron usando herramientas de inteligencia artificial y se utilizaron bajo una licencia *Free Non Profit*.
- **Skydome HDR:** El fondo del cielo es un activo de alta calidad de *PolyHeaven*. El modelo utilizado es **Çitrus Orchard Road (pure sky)"** de los artistas **Jarod Guest y Dimitrios Sawa**, y se usa bajo la licencia *Free Non Profit* de la plataforma.
- **Sonido:** El efecto de sonido del péndulo es una grabación original del autor, creada al golpear dos imanes, y procesada en el archivo `golpe_esferas.wav`.

8.2. Estructura del Software

El proyecto se centra en un archivo `main.cpp` que actúa como orquestador, apoyado por clases auxiliares estándar de OpenGL:

- `main.cpp`: Contiene el *Game Loop*, toda la lógica de estado de las animaciones, la gestión de *callbacks* de GLFW (teclado/ratón) y el orden de renderizado.
- `Shader.h`: Clase auxiliar para cargar, compilar y activar programas de shaders GLSL.
- `Camera.h`: Clase que gestiona la cámara libre (posición, orientación) y calcula la matriz *View*.
- `Model.h`: Clase (basada en Assimp) que carga archivos `.obj`, gestionando sus mallas y texturas.
- `stb_image.h`: Biblioteca de un solo encabezado (incluida en `main.cpp`) vital para cargar texturas, especialmente el archivo `skybox/cielo.hdr` gracias a su función `stbi_loadf`.

8.3. El Bucle Principal (Game Loop)

El orden de operaciones dentro del `while` loop es crítico para el correcto funcionamiento:

1. **Actualización de Lógica:** Se calcula `deltaTime` y se actualizan los valores de estado de todas las animaciones activas (ej. `sunAngle`, `phaseTime`, `fishXOffset`).
2. **Entrada y Matrices:** Se procesan los eventos (`glfwPollEvents`) y se calculan las matrices `view` y `projection` maestras.
3. **Dibujado del Skydome:** Se dibuja **primero** que todo lo demás, usando su propio shader (`skydomeShader`) y trucos de profundidad (`glDepthFunc(GL_LEQUAL)`) y de matriz de vista (`glm::mat4(glm::mat3(view))`) para que parezca infinito.
4. **Activación de Shaders de Escena:** Se activa el `lightingShader` y el `cloudsShader`, y se les pasan las matrices `view` y `projection`.
5. **Configuración de Iluminación:** Se ejecuta la lógica de `if (sunAnimationActive)` que determina la intensidad de todas las luces de la escena basándose en el `dayFactor`.
6. **Dibujado de Objetos Opacos:** Se dibujan todos los modelos sólidos con `glDepthMask(GL_TRUE)` (casa, muebles).
7. **Dibujado de Objetos Translúcidos:** Se activa `glEnable(GL_BLEND)` y `glDepthMask(GL_FALSE)`. Se dibujan los vidrios, la pecera y el modelo del sol.
8. **Swap Buffers:** Se intercambian los búferes.

8.4. Arquitectura de Animación (por Sistema)

8.4.1. Sistema 1: Puertas y Cortinas (Simple)

- **Lógica:** Basada en un estado booleano simple, `puertaAbierta`.
- **Activación:** La `KeyCallback` (tecla '5') invierte el estado de `puertaAbierta` y actualiza `factorCierre` a `1.0f` (cerrado) o `0.0f` (abierto).
- **Ejecución:** En el bucle de dibujado, la matriz `model` de cada cortina se multiplica por una traslación. Esta animación es binaria y no interpolada.

Listing 1: Lógica de animación de puertas (rotación)

```
glm::vec3 pivotPuerta(-3.975f, 5.921f, 0.4826f);
float rotPuerta = puertaAbierta ? 90.0f : 0.0f; // 90 grados o 0

glm::mat4 modelPuerta = glm::mat4(1.0f);
modelPuerta = glm::translate(modelPuerta, pivotPuerta);
modelPuerta = glm::rotate(modelPuerta, glm::radians(rotPuerta),
    glm::vec3(0.0f, 1.0f, 0.0f));
modelPuerta = glm::translate(modelPuerta, -pivotPuerta);
// ...
puerta_cuarto_phineas.Draw(lightingShader);
```

8.4.2. Sistema 2: Pez (Simple)

- **Lógica:** Es una máquina de estados simple con dos componentes de animación independientes.
- **Componente 1 (Cola):** Animación oscilatoria constante. La variable `tailAngle` se actualiza en cada frame usando `sin(tailTime)`.

Listing 2: Lógica de oscilación de la cola del pez.

```
// tailTime se incrementa con deltaTime
tailAngle = TAIL_AMPLITUDE * sin(tailTime * TAIL_FREQ);
// ...
// Se aplica la rotacion en el pivote de la cola
tailModel = glm::translate(tailModel, tailPivotLocal);
tailModel = glm::rotate(tailModel, glm::radians(currentTailAngle),
    glm::vec3(0.0f, 1.0f, 0.0f));
tailModel = glm::translate(tailModel, -tailPivotLocal);
```

- **Componente 2 (Cuerpo):** Máquina de estados para traslación y giro.
  - **Estado de Traslación:** La variable `fishXOffset` se incrementa o decrementa con `FISH_FORWARD_SPEED`. La dirección depende del booleano `fishFlipped`.
  - **Estado de Giro (Flip):** Un temporizador `fishSwimTime` se incrementa. Al llegar a `FLIP_TIME`, se activa el booleano `isFlipping`.
  - Durante el flip, `flipRotation` se interpola de 0 a 180 (o de 180 a 360) basado en `flipProgress`.
  - Al terminar el flip, `isFlipping` se pone en falso y `fishFlipped` se invierte, cambiando la dirección de traslación.

8.4.3. Sistema 3: Péndulo de Newton (Compleja)

- **Lógica:** Animación kinemática por fases, no es una simulación física.
- **Estado:** Controlado por `currentPhase` (int) y `phaseTime` (float).
- **Ejecución:** Un `if/else if` comprueba la fase actual:
  - **Fase 0 (Inicio):** La esfera 1 (`angle1`) se mueve de -30° a 0° usando `cos(omega * phaseTime)`.
  - **Fases Impares (1, 3, ...):** La esfera 4 (`angle4`) oscila de 0° a 30° y vuelve a 0° usando `sin(omega * phaseTime)`.
  - **Fases Pares (2, 4, ...):** La esfera 1 (`angle1`) oscila de 0° a -30° y vuelve a 0° usando `sin(omega * phaseTime)`.
- **Detalle Sutil:** Las esferas centrales (2 y 3) se mueven ligeramente (`angleMiddle`) para dar una sensación de impacto.
- **Sonido:** Cuando una fase termina (`phaseTime >= halfPeriodTime`), se activa `advancePhase`, se incrementa `currentPhase`, y se llama a `PlaySound` para reproducir el `.wav` personalizado (choque de imanes).

8.4.4. Sistema 4: Techo de Nubes (Compleja - Shader)

- **Lógica:** Esta animación se delega casi por completo a la GPU, lo cual es muy eficiente.
- **CPU (`main.cpp`):** El código C++ simplemente activa el `cloudsShader` y le pasa dos *uniforms*: `cloudsMoving` (booleano) y `cloudsTime` (float).
- **GPU (Shader `clouds.vs`):** El Vertex Shader es el que realiza la animación.

Listing 3: Shader de Vértice (clouds.vs) - Lógica de Animación

```
// ...
uniform float time;
uniform bool cloudsMoving;
uniform float speed;
// ...
void main()
{
    // ...
    if (cloudsMoving) {
        // mod() crea un valor que se repite de 0.0 a 1.0
        float offset = mod(time * speed, 1.0);
        // Desplaza la coordenada UV en el eje X
        TexCoords = vec2(uv.x - offset, uv.y);
    } else {
        TexCoords = uv;
    }
    gl_Position = projection * view * model * vec4(aPos, 1.0);
}
```

- **GPU (Shader `clouds.frag`):** El Fragment Shader implementa .ªlpha testing"manual. La textura del techo tiene partes transparentes (alpha bajo). La línea `discard` le dice a la GPU que no dibuje esos píxeles, creando el efecto de çorte"de las nubes.

Listing 4: Shader de Fragmento (clouds.frag) - Alpha Discard

```
// ...
void main()
{
    vec4 texColor = texture(texture1, TexCoords);
    // Si el pixel es muy transparente, no lo dibujes
    if (texColor.a < 0.1)
        discard;
    FragColor = texColor;
}
```

8.4.5. Sistema 5: Ciclo Día/Noche y Skydome (Compleja)

- **Lógica:** Es el sistema más complejo, ya que combina tres componentes: el renderizado del cielo, la animación del sol, y la lógica de iluminación global.
- **Componente 1 (Renderizado del Skydome):**

- **Carga (C++):** Se usa `stbi_loadf` para cargar `cielo.hdr` en un buffer de `float`. Se sube a la GPU con el formato interno `GL_RGB16F`.
- **Renderizado (C++):** Se dibuja **primero**. Se activa `glDepthFunc(GL_LEQUAL)` para que el domo solo se dibuje en los píxeles del fondo (donde Z=1.0).
- **Shader (`skydome.vs`):** Implementa dos trucos clave. Primero, elimina la traslación de la cámara (`view = glm::mat4(glm::mat3(view))`) para que el usuario nunca se acerque al cielo. Segundo, fuerza la coordenada Z de la esfera a 1.0 (`gl_Position = pos.xyww;`) para que siempre esté detrás de todos los demás objetos.

Listing 5: Shader de Vértice (skydome.vs) - Truco de Profundidad

```
// ...
void main()
{
    TexCoords = aTexCoords;
    vec4 pos = projection * view * vec4(aPos, 1.0);
    // pos.xyww hace que z = w. Despues de la division de
    // perspectiva, z/w se vuelve w/w = 1.0 (el fondo)
    gl_Position = pos.xyww;
}
```

- **Componente 2 (Animación del Sol):**
  - **Posición (C++):** `sunAngle` se actualiza con `deltaTime`. Se calcula una nueva `sunPos` usando `glm::rotate` sobre el eje X.
  - **Dibujado (C++):** El modelo `sol` se dibuja en el pase de translúcidos, aplicando la *misma* matriz de rotación usada para calcular `sunPos`.
- **Componente 3 (Iluminación Dinámica):**
  - **El "Factor de Día"(C++):** Esta es la variable clave. Se calcula usando `dayFactor = sin(glm::radians(sunAngle))` para el rango de 0° a 180°. Esto da una interpolación suave de 0 (noche) a 1 (mediodía) y de vuelta a 0 (noche). De 181° a 359°, el `dayFactor` permanece en 0.
  - **Interpolación (C++):** La luz ambiental (`dirLight`) y la intensidad del sol (`pointLights[0]`) se interpolan linealmente (usando `glm::mix`) entre sus valores de noche (casi 0) y sus valores de día (máximos).
  - **Interruptor (C++):** El sistema actúa como un interruptor. Si `dayFactor >0.0` (es de día), la lámpara (`pointLights[1]`) se apaga. Si `dayFactor == 0.0` (es de noche), la lámpara se enciende. (Ver `lstlisting` de la sección anterior).

9. Galería de Proyecto (Resultados)

A continuación, se presenta una serie de capturas de pantalla que ilustran los resultados finales del proyecto, mostrando la escena, los modelos y las animaciones implementadas.

Figura 2: Vistas generales de la escena: (Izquierda) Exterior de la casa. (Derecha) Interior del cuarto de Phineas y Ferb.

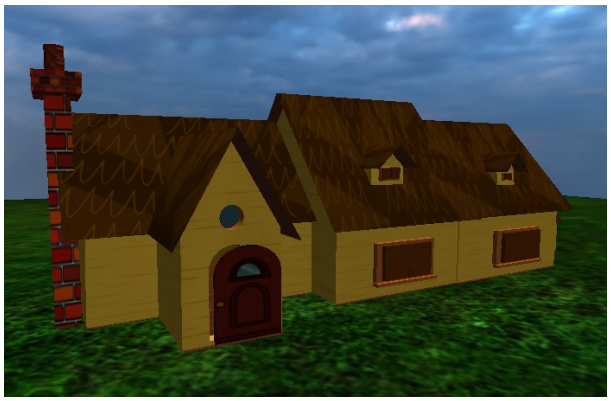


Figura 3: Animaciones simples: (Izquierda) Lógica de apertura/cierre de puertas y cortinas (estado cerrado). (Derecha) Animación del pez nadando en la pecera.

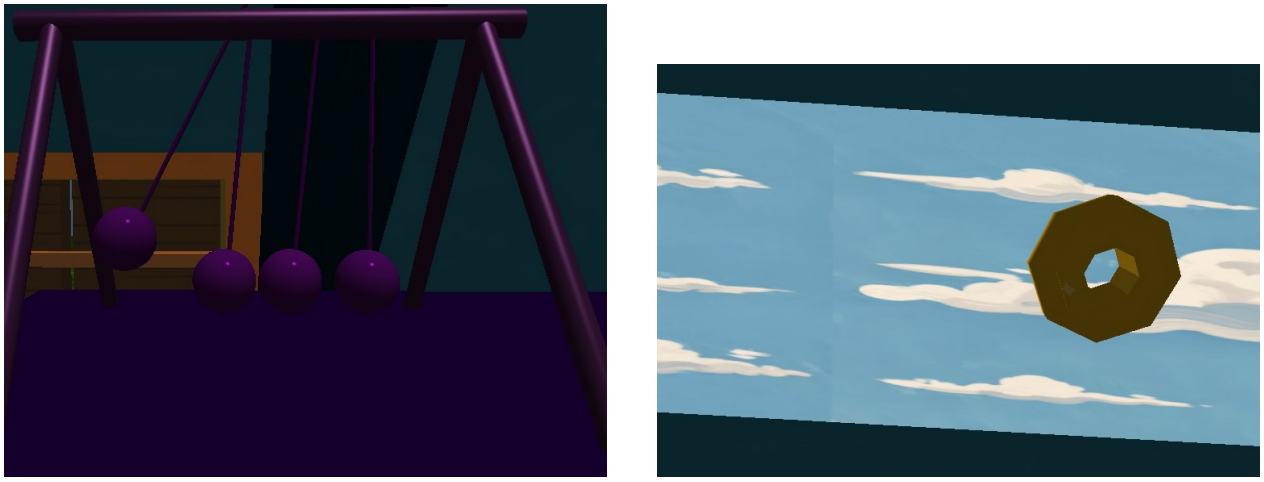
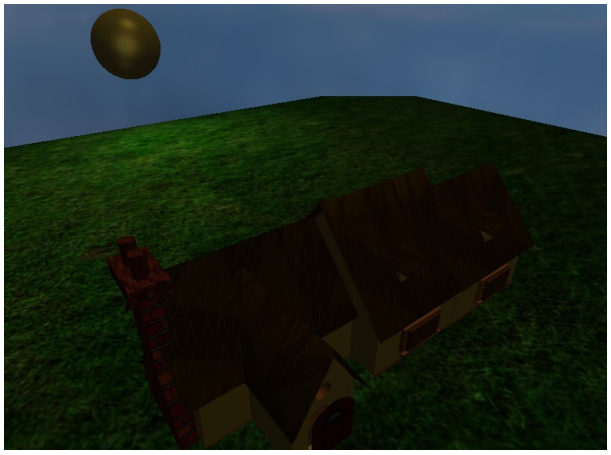


Figura 4: Animaciones complejas: (Izquierda) Animación kinemática por fases del péndulo de Newton. (Derecha) Animación del techo de nubes controlada por shader.



a.48a.48

Figura 5: Sistema de ciclo Día/Noche: (Izquierda) El sol visible durante su rotación de día. (Derecha) Escena de noche, donde el sol se ha ocultado y la lámpara del escritorio se enciende automáticamente.

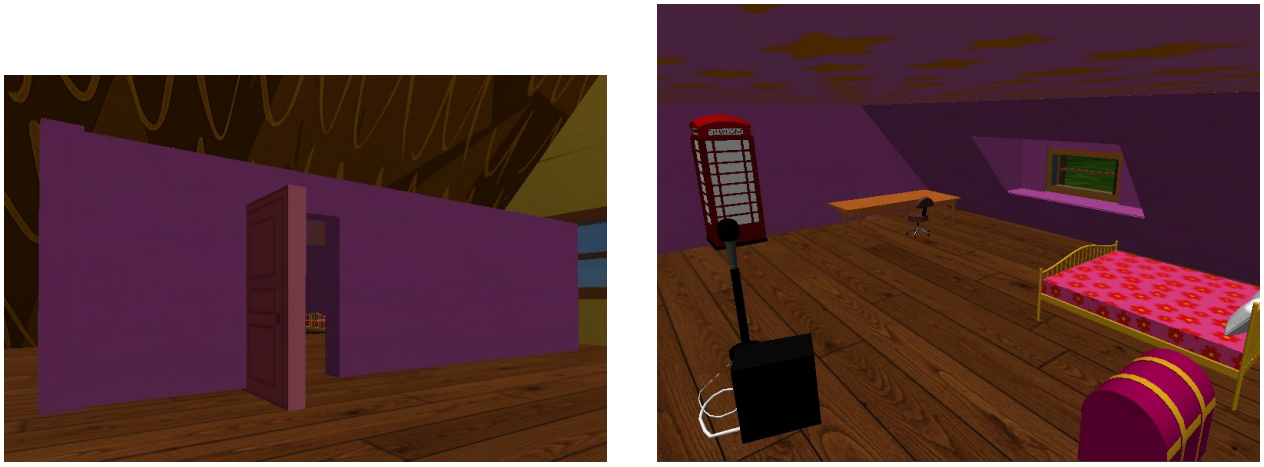


Figura 6: Expansión del proyecto - Cuarto de Candace: (Izquierda) Vista exterior del cuarto de Candace. (Derecha) Vista interior que muestra el espacio y disposición del mobiliario.

Figura 7: Detalles del cuarto de Candace: Se muestran los objetos modelados específicamente para este espacio - la cama, el baúl característico y el micrófono, elementos icónicos del personaje.

10. Conclusiones

Se logró cumplir con el objetivo principal del proyecto, desarrollando una escena 3D funcional, interactiva y estéticamente coherente con la temática de "Phineas y Ferb". Se implementaron con éxito los 7 modelos de objetos y los 5 sistemas de animación requeridos (3 complejos y 2 simples).

El desarrollo del proyecto permitió la aplicación y el aprendizaje profundo de conceptos fundamentales de los gráficos por computadora en tiempo real, destacando:

- **Arquitectura de Renderizado:** El manejo del orden de dibujado, la gestión de los búferes de color y profundidad (`glDepthMask`), y el uso de `glDepthFunc` para el *Skydome*.
- **Iluminación Dinámica:** La implementación de un sistema de iluminación que evoluciona en el tiempo, combinando luces direccionales y puntuales, y alterando el estado de la escena (día/noche).
- **Programación de Shaders (GLSL):** El uso de múltiples programas de shaders para diferentes propósitos (`lighting`, `skydome`, `clouds`) y el paso de *uniforms* para controlar animaciones (`time`) y lógica (`discard`).
- **Gestión de Activos:** La integración exitosa de modelos de terceros (Pez, Cabina) con texturas personalizadas, y el uso de texturas generadas por IA, respetando las licencias no comerciales.
- **Renderizado HDR:** La carga y renderizado de texturas de alto rango dinámico (`.hdr`) con `stb_image` y la configuración de formatos de textura flotante (`GL_RGB16F`).
- **Audio:** La integración de un efecto de sonido original (choque de imanes) en una animación kinemática, sincronizado con los eventos del programa.

Como trabajo a futuro, el proyecto podría expandirse para incluir técnicas más avanzadas, como un sistema de *tone mapping* global para toda la escena (no solo el cielo), la implementación de sombras (*shadow mapping*) proyectadas por el sol dinámico, y la optimización del renderizado mediante *instancing*.

Diagrama de Flujo - main.cpp (Proyecto Final OpenGL)