Universidad Nacional Autónoma de México

Facultad de Ingeniería

Computación gráfica e interacción humano-computadora

Proyecto Final

317142165

Profesor: Ing. Carlos Aldair Roman Balbuena

grupo: 05

Fecha de entrega: 25 de noviembre 2025

Semestre: 2026 - 1

Manual de Usuario

La finalidad de este manual es brindar una guía a los usuarios sobre la forma correcta de interactuar dentro del proyecto “Monster House”. Aquí se explica, de manera sencilla y detallada, el uso de los controles, las características disponibles y las acciones esenciales que permiten que la experiencia sea funcional e inmersiva.

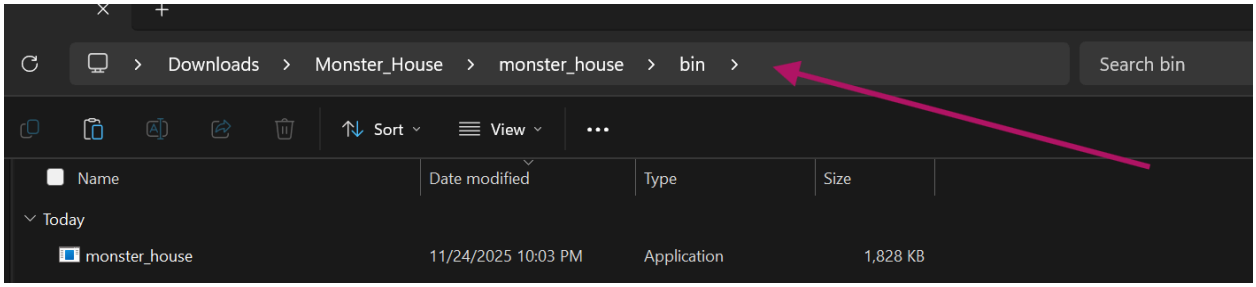
Requisitos mínimos del sistema

- Sistema operativo: Windows 10 o superior (64 bits).
- Procesador: Intel Core i3 o equivalente.
- RAM: 4 GB mínimo.
- GPU: Soporte para OpenGL 3.3 o superior.
- Resolución recomendada: 1366x768 o superior.

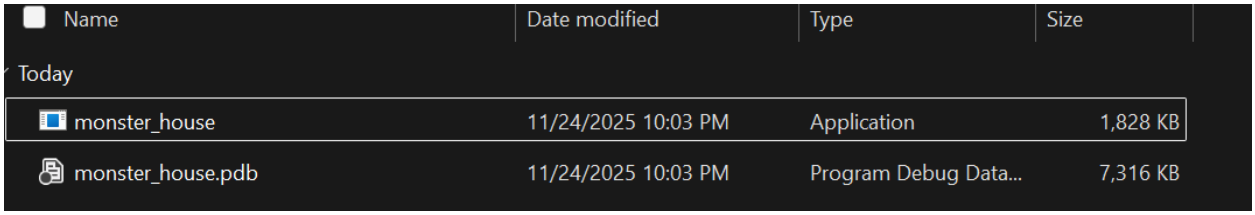
Nota: No muevas ni cambies el nombre de estas carpetas, ya que contienen shaders y modelos 3D indispensables para el funcionamiento.

Instrucciones para ejecutar el entorno

1. En el directorio donde se hizo el git clone, navegar hasta el directorio que se muestra en la imágen.



2. Una vez ubicado el archivo monster_house con extensión de Application; ejecutarlo con doble click.



Controles de usuario:

El desplazamiento se gestiona mediante el siguiente mapeo de teclas:

- W / Flecha Arriba: Avanzar. Mueve al personaje hacia adelante.
- S / Flecha Abajo: Retroceder. Mueve al personaje hacia atrás.
- A: Mover hacia la izquierda (desplazamiento lateral).
- D: Mover hacia la derecha (desplazamiento lateral).
- Esc: Salir del programa.

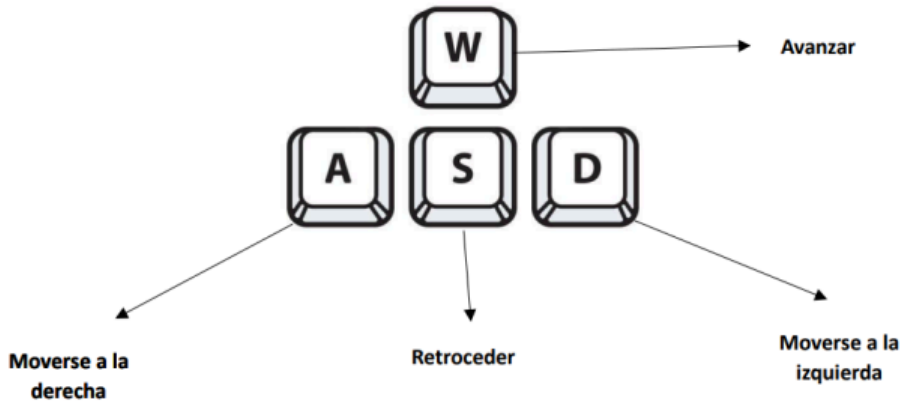


Figura o. Controles de usuario

Realismo gráfico

- El espacio virtual toma de referencia la casa de la película “Monster House”.

Fachada:

Figura 1 y 2. Imágen de la película Monster House y fachada en OpenGL.

Figura 3. Fachada modelada en Blender.

Cuarto 1:

Figura 4. Cama modelada en Blender para el cuarto 1.

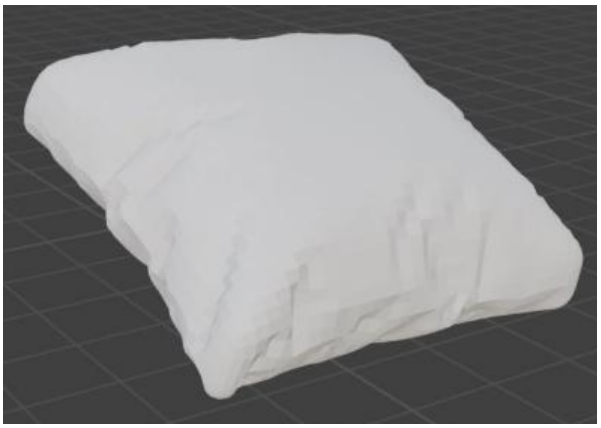


Figura 5. Almohada modelada en Blender para el cuarto 1.

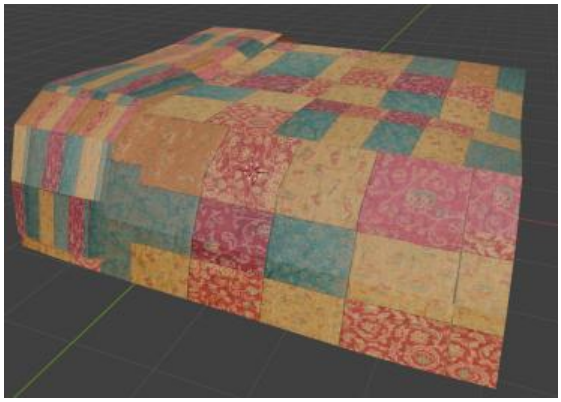


Figura 6. Colcha modelada en Blender para el cuarto 1.

Figura 7. Balón de BasketBall modelado en blender para el cuarto 1

Figura 8. Mueble modelado en blender para el cuarto 1.

Figura 9 y 10. Imagen de referencia para el cuarto 1 y cuarto en OpenGL.

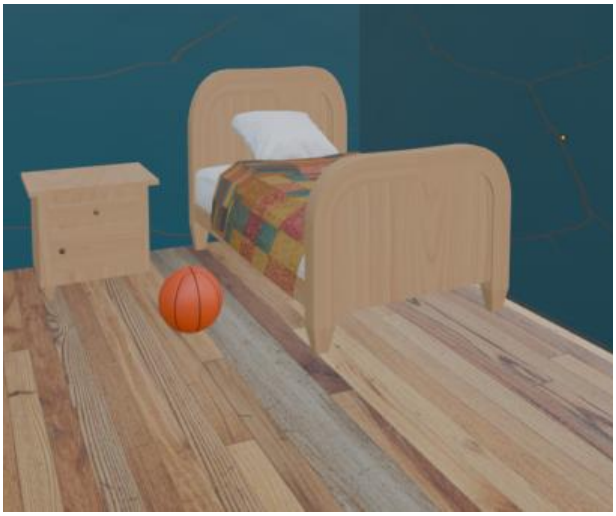


Figura 11. Cuarto completo modelado en Blender.

Comedor:

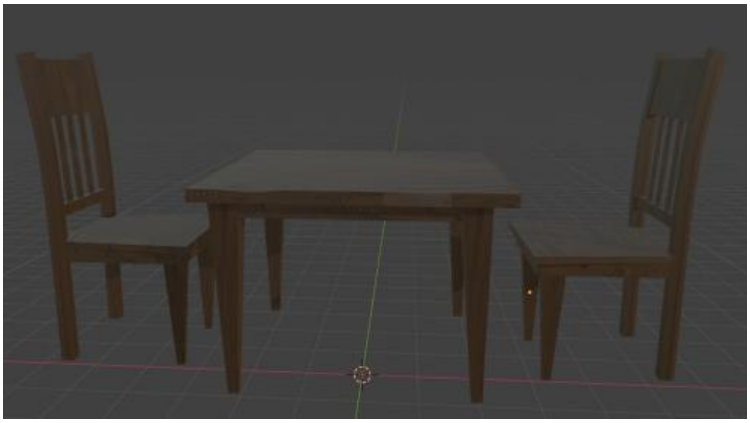


Figura 12. Sillas y mesa modeladas en blender.

Figura 13. Mueblecito de comedor modelado en Blender.

Figura 14. Cuadro modelado en Blender.

Figura 15 y 16. Imágen de referencia para el comedor y el comedor desplegado en OpenGL.

Los modelos fueron creados y exportados desde Blender con texturas integradas. El proceso de mapeo de texturas depende de que el cargador de modelos (Model class) pueda encontrar el archivo de imagen en el disco. El modelo 3D almacena la ruta de la textura, y si esta ruta es incorrecta (el problema más común), el objeto aparecerá sin textura.

Para que funcione, las imágenes deben estar en una ruta accesible. Una vez cargada, la textura se activa en una unidad de textura. El *fragment shader* luego utiliza una variable sampler para leer el color de esa unidad, completando el renderizado del detalle visual.

Manual Técnico

Objetivos del Proyecto

El alumno deberá aplicar y demostrar los conocimientos adquiridos durante el curso mediante la creación de una recreación 3D en OpenGL.

Diagrama de flujo del software

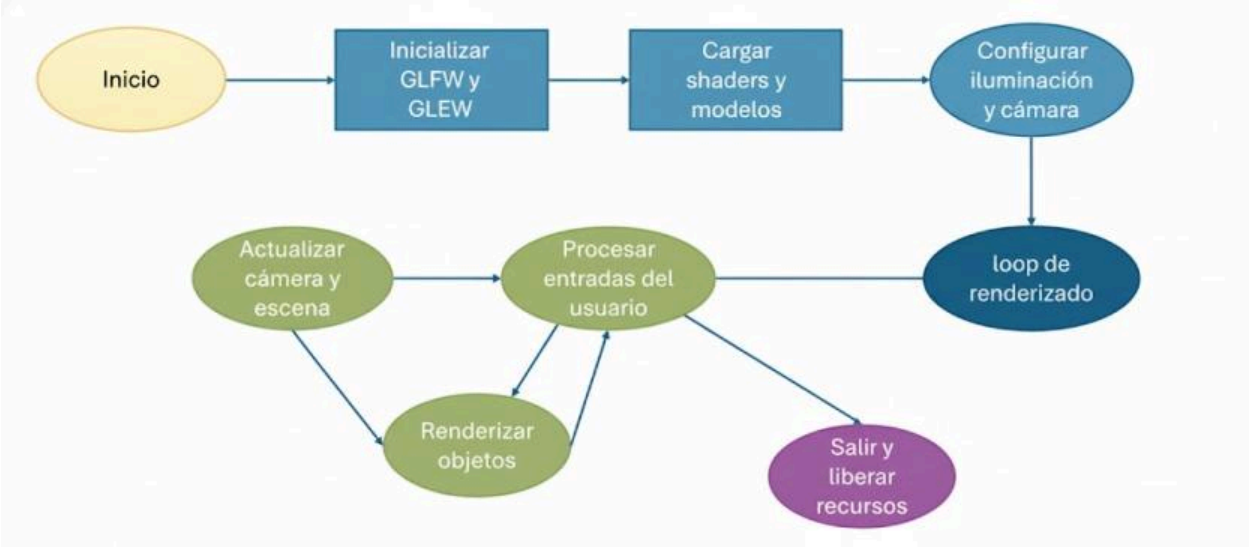
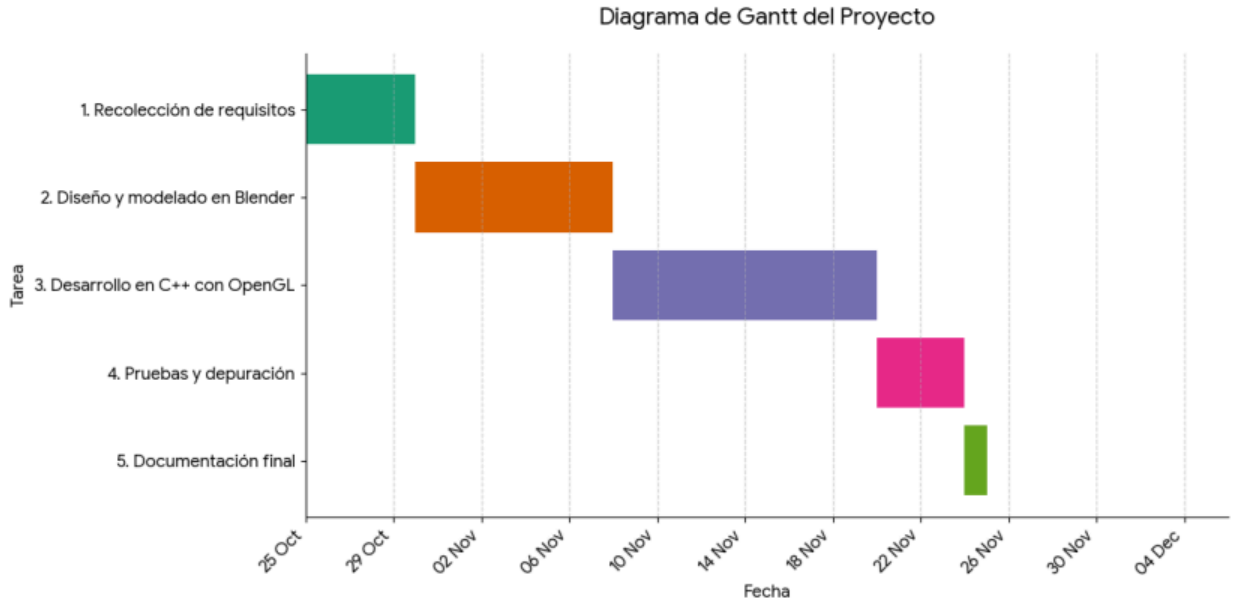


Diagrama de Gantt



Alcance del proyecto

Interactividad y cámara

- Implementación de una cámara libre en primera persona que permite al usuario recorrer la escena utilizando teclado y mouse.
- Control dinámico de movimiento (W, A, S, D o flechas) y rotación mediante desplazamiento del mouse.

Escena 3D

- Recreación detallada de una fachada arquitectónica y un interior compuesto por dos habitaciones (comedor, cuarto).
- Integración de 10 modelos 3D exportados desde Blender en formato .fbx.

Iluminación

- Sistema de iluminación avanzado basado en el modelo de Phong.
- Configuración de colores, intensidades y atenuaciones en cada tipo de luz.

Materiales y texturizado

- Uso de materiales con texturas diffuse y specular para cada modelo, aplicadas con mapeo UV desde Blender.

Renderizado en tiempo real

- Renderizado eficiente de la escena con control de profundidad (Z-buffer).
- Configuración de VAO/VBO para mallas.
- Uso de estructuras y funciones organizadas en clases: Model, Mesh, Camera, Shader.

Limitantes

1. Compatibilidad de formato y texturas

- El formato FBX no permite incrustar (*embed*) las texturas bitmap directamente. Esto obliga a que los archivos de textura (.png, .jpg) se mantengan en una ruta de acceso válida (a menudo la misma carpeta o una ruta relativa) para que puedan ser localizados y cargados correctamente por la aplicación de destino (OpenGL/C++). Si la ruta falla, el modelo aparece sin texturas.

- Las texturas especulares (map_Ks) no se exportan automáticamente desde Blender si no se configuran explícitamente en el nodo del material, lo que complica la simulación de reflectividad si no se hace manualmente.

2. Renderizado limitado a Phong básico

- El sistema de iluminación está basado en el modelo de iluminación Phong, lo cual limita los efectos visuales avanzados como reflejos en tiempo real, oclusión ambiental o materiales PBR más realistas.
- No se utilizaron técnicas como shadow mapping, screen-space reflections ni normal mapping, lo cual restringe la fidelidad visual.

3. Rendimiento dependiente del hardware

- Al no implementar técnicas de optimización como frustum culling, LOD (Level of Detail) o batches de instanciado, el rendimiento puede verse afectado en equipos con GPU integrada o baja potencia gráfica.

4. Interacción básica

- La navegación está limitada al movimiento libre mediante teclado y ratón. No se implementaron colisiones, zonas de interacción avanzadas ni detección de eventos en objetos.
- No hay menú ni interfaz gráfica integrada que facilite la experiencia de usuario.

5. Precisión visual sujeta a exportación

- Las escalas, rotaciones y orientaciones de los modelos exportados desde Blender pueden no coincidir exactamente con la escena original si no se aplican correctamente las transformaciones (Ctrl + A ↠ Apply All Transforms).

6. Ausencia de efectos atmosféricos

- No se incluyeron efectos de postprocesado como niebla, desenfoque, bloom, ni tono ambiental dinámico, lo cual reduce la sensación de realismo.

Metodología de software aplicada

Se utilizó el modelo en cascada, el cual sigue una secuencia lineal de etapas. Este modelo fue adecuado para el desarrollo del entorno gráfico 3D, permitiendo una planificación estructurada del trabajo. Las etapas aplicadas fueron:

1. **Recolección de requisitos**
   Se definieron los elementos esenciales del escenario, incluyendo la temática, número de objetos mínimos requeridos, tipo de interacción y sistemas de iluminación.
2. **Diseño del entorno y modelado de objetos**
   Se planificó la disposición del espacio 3D y se modelaron los objetos en Blender, considerando proporciones, texturas, materiales y organización del espacio virtual.
3. **Desarrollo en C++ con OpenGL**
   Se implementó el código fuente utilizando C++ y la librería OpenGL. Se integraron los modelos .fbx exportados desde Blender, se aplicaron transformaciones, texturas, iluminación y cámaras interactivas.
4. **Pruebas visuales e interacción**
   Se realizaron pruebas constantes para verificar el correcto renderizado de los modelos, la respuesta a eventos de teclado y mouse, y el comportamiento de las luces dinámicas y objetos animados.
5. **Documentación**
   Se elaboró un manual técnico y de usuario, registrando tanto el diseño, estructura del código y funcionamiento general del entorno, como las instrucciones necesarias para la navegación e interacción.

Documentación del código

1. Variables Globales Relevantes

Variable	Tipo	Descripción
window	GLFWwindow*	Puntero a la ventana de GLFW, el contexto principal de renderizado.

camera	Camera	Instancia de la clase Camera para la vista.
sceneManager	std::unique_ptr<SceneManager>	Administrador central de la escena, maneja la jerarquía de objetos y el renderizado.

2. Funciones de Inicialización (Start)

bool initializeWindow()

- Descripción: Configura e inicializa la ventana de GLFW y GLAD, el *loader* de OpenGL. Establece el contexto gráfico y los *callbacks* de entrada.
- Parámetros: Ninguno.
- Retorno: true si la inicialización fue exitosa, false en caso contrario.

void loadModels(...)

- Descripción: Se encarga de cargar los modelos 3D esenciales para la escena. En la versión actual del código, solo carga el piso y la casa principal.
- Parámetros:
  - loadingScreen: Referencia a la pantalla de carga para mostrar el progreso.
  - house: Puntero a Model* donde se cargará el modelo de la casa ("monster_house/models/MonsterHouseFinal.fbx").
  - piso: Puntero a Model* donde se cargará el modelo del piso lunar ("monster_house/models/piso.fbx").
- Acción Específica (Casa y Piso): Utiliza la ruta de los archivos FBX para cargar los datos del modelo en las estructuras Model.

void defineMaterials(...)

- Descripción: Define las propiedades del material (ambiente, difuso, especular, transparencia) para los objetos principales de la escena.
- Parámetros (Relevantes):
  - floorMaterial: Referencia a Material para configurar las propiedades del piso.
  - houseMaterial: Referencia a Material para configurar las propiedades de la casa principal.

    - (Otros Materiales): Se establecen a Material() (vacío/neutro) para los objetos eliminados.
- Configuración Específica:
    - Piso: Se configura con una alta contribución ambiental y difusa grisácea/azulada, y un bajo especular para simular la textura rocosa opaca de la superficie lunar.
    - Casa: Se configura con propiedades más balanceadas, típicas de una estructura de color gris o metálico, con un especular moderado.

void setupLighting(...)

- Descripción: Configura y agrega las luces a la escena a través del LightManager del SceneManager.
- Parámetros (Relevantes):
    - sunLightIdx: Referencia a size_t para almacenar el índice de la luz global del Sol.
    - interiorCasaLightIdx: Referencia a size_t para almacenar el índice de la luz local interior de la casa.
- Acción: Crea instancias de Light, define su Position, Color y Power, y las añade al gestor de luces.

void createSceneObjects(...)

- Descripción: Instancia los objetos renderizables (RenderableObject) de la escena a partir de los modelos y shaders cargados, les asigna su material, posición, rotación y escala, y los añade al SceneManager.
- Parámetros (Relevantes):
    - piso: Puntero al modelo Model del piso.
    - house: Puntero al modelo Model de la casa.
    - mLightsShader: Puntero al Shader principal para objetos con múltiples luces.
    - floorMaterial: Referencia al Material del piso.
    - houseMaterial: Referencia al Material de la casa.
    - interiorCasaLightIdx: Índice de la luz interior para que la casa se vea afectada por ella.
- Instanciación Específica:
    - Piso:
        - Crea un RenderableObject con pisoLunar y mLightsShader.
        - Posición: (0.0f, 0.0f, 0.0f) (Centro de la escena).
        - Rotación: (-90.0f, -90.0f, 0.0f) (Ajuste para que el modelo quede plano).
        - Escala: (5.0f) (Muy grande).

        - Asigna floorMaterial.
    - Casa (MonsterHouse):
        - Crea un RenderableObject con house y mLightsShader.
        - Posición: (0.0f, 16.5f, 0.0f) (Posición ajustada sobre el piso lunar).
        - Rotación: (-90.0f, 0.0f, 0.0f) (Orientación base).
        - Escala: (1.0f) (Escala original).
        - Asigna houseMaterial.
        - Añade interiorCasaLightIdx a sus luces afectadas.

3. Función de Loop Principal (`Update`)

bool Update()

- Descripción: La función principal del *render loop* (bucle de juego). Calcula el tiempo delta, procesa la entrada, borra el *buffer* de pantalla, actualiza las posiciones de las cámaras (incluyendo el desplazamiento vertical del salto) y llama a los métodos de actualización y renderizado del SceneManager.
- Parámetros: Ninguno.
- Retorno: true para continuar el bucle.
- Acción Específica (Cámaras):
    - Calcula el desplazamiento vertical por física (jumpDisplacement).
    - Actualiza la posición de la cámara activa.
    - Llama a sceneManager»update(deltaTime) para actualizar la lógica de todos los objetos, incluyendo rotaciones u órbitas.
    - Llama a sceneManager»render() para dibujar la escena, renderizando el cielo (Cubemap), el Piso y la Casa.

4. Clases del Proyecto (Modularizadas)

RenderableObject

- Uso (Piso y Casa): Es la clase base para los objetos estáticos de la escena. Las instancias del piso y la casa son de este tipo.
- Funcionalidades Clave:
    - Almacena un puntero a un modelo (Model*) y un shader (Shader*).
    - Define la matriz de transformación (model matrix) mediante glm::vec3 de posición, rotación y escala.
    - Almacena las propiedades del material (Material).
    - Método render(): Dibuja el modelo usando su shader y enviando las propiedades de luz y material.
    - Método addAffectedLight(): Permite especificar qué luces locales afectan al objeto (usado para la luz interior de la casa).

Model

- Uso (Piso y Casa): Contiene la estructura de datos para un modelo 3D estático (mallas, vértices, texturas) cargado a partir de archivos como FBX.
- Funcionalidades Clave: Abstrae la carga de modelos, normalmente utilizando librerías como Assimp, y almacena las mallas (Mesh) que componen el objeto.

Análisis de Costos y Precio de Venta del Proyecto

Recurso/Actividad	Cantidad	Costo Unitario (MXN)	Costo Total (MXN)	Comentarios
Horas de desarrollo (programación OpenGL)	60 h	$100/h	$6,000	Codificación de lógica, animaciones, shaders, renderizado, etc.
Horas de modelado 3D (Blender)	60 h	$100/h	$6,000	Modelado y texturizado de objetos en Blender
Documentación (manuales, reporte)	10 h	$80/h	$800	Redacción técnica y de usuario
Software utilizado (licencias gratuitas)	-	$0	$0	Se usó software de código abierto (Blender, GLEW, GLFW, Assimp)
Hardware (PC personal)	1 mes de uso	$500 amortización	$500	Uso de computadora propia para render y programación
Electricidad (estimado)	1 mes	$150	$150	Costo aproximado de energía para sesiones de trabajo

Capacitación previa (cursos o clases)	1 curso	$1,000	$1,000	Curso universitario de Computación Gráfica
Total estimado de costos			$14,450 MXN	

Estimación del precio de venta

Para determinar el precio de venta se considera:

- **Margen de utilidad del 50%, por tratarse de un desarrollo técnico personalizado.**
- **Costos indirectos, como posibles mantenimientos o soporte básico.**
- **Valor agregado por la interactividad, uso educativo o presentación visual atractiva.**

Cálculo:

$$Precio\ de\ venta\ = Costo\ Total\ x\ 1.5\ = 14450\ x\ 1.5\ = \$21675$$

CONCLUSION

El desarrollo de este proyecto de computación gráfica en C++ y OpenGL no solo sirvió como una asignación académica, sino como una prueba de fuego para consolidar y aplicar la totalidad de los conocimientos adquiridos durante el semestre. El objetivo principal de

construir un entorno virtual funcional, fue plenamente alcanzado al integrar con éxito *shaders*, iluminación avanzada (Modelo de Phong), gestión de recursos 3D (modelos FBX) y cámara.

Desde mi perspectiva, el proyecto resultó ser considerablemente más retador de lo anticipado. La complejidad no residió únicamente en el código, sino en la interdependencia de los sistemas: desde la configuración inicial de GLFW y GLAD hasta la correcta interpretación de los datos de los modelos y las matrices de transformación. Fue un ejercicio intenso de debugging y de adquisición de nuevas habilidades, especialmente en la correcta implementación de shaders personalizados. A pesar de los obstáculos, logramos un alcance funcional muy sólido en el renderizado y la interacción del usuario.

Si bien el proyecto cumplió con los requisitos básicos, es crucial identificar las áreas donde se presentaron limitaciones o donde la implementación pudo ser más robusta, lo cual constituye una lección invaluable para mis futuros desarrollos:

1. Integración de Assets (Texturas FBX): La mayor frustración inicial provino del manejo de texturas. La limitación del formato FBX de no incrustar las texturas, obligándonos a manejar rutas de archivo externas y relativas, causó retrasos y fallos de loading. Esto nos enseñó la importancia crítica de estandarizar los pipelines de exportación desde Blender.
2. Gestión de la Iluminación Especular: Nos apoyamos fuertemente en los valores fijos de Material.specular en C++ para simular el brillo. Esto fue, en parte, una consecuencia de que los mapas especulares no se exportaran correctamente del modelo FBX, lo que resultó en una simulación de reflectividad menos precisa y más uniforme de lo que se hubiera logrado con un mapeo de textura adecuado.

En definitiva, el proyecto fue un éxito en la síntesis de conocimientos, pero nos dejó claro que la robustez del pipeline de assets y la utilización completa de todas las capacidades de los shaders son los siguientes pasos críticos en mi aprendizaje de gráficos por computadora.