UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ingeniería

Proyecto Final

Asignatura:
Computación Gráfica e Interacción Humano-Computadora (1590)

Grupo: 05
Semestre: 2025-2

Alumno:
Quintero Montero Francisco Joshua

Profesor:
ING. CARLOS ALDAIR ROMAN BALBUENA

Entrega: 20 de mayo de 2025

Tabla de contenidos

Manual de Usuario

Objetivos

Objetivo General

Brindar una guía clara y accesible para que usuarios puedan descargar, instalar, iniciar y navegar de manera intuitiva en el entorno virtual desarrollado en OpenGL, permitiendo una experiencia inmersiva y sin complicaciones técnicas.

Objetivos Específicos

- Explicar paso a paso el proceso de instalación y ejecución del proyecto para facilitar su puesta en marcha.
- Describir los controles básicos para el desplazamiento y la interacción dentro del entorno virtual.
- Informar sobre las funcionalidades de control de iluminación para que el usuario pueda personalizar la experiencia visual dentro de la simulación.
- Fomentar una experiencia de usuario agradable y fluida mediante una presentación clara y amigable de las instrucciones.

Requisitos Mínimos del Sistema

Para garantizar el correcto funcionamiento del proyecto y una experiencia fluida durante la navegación del entorno 3D, se recomienda que el sistema cumpla al menos con los siguientes requisitos mínimos:

Hardware

- Procesador (CPU): Intel Core i5 de 4 núcleos o equivalente AMD
- Memoria RAM: 8 GB
- Tarjeta Gráfica (GPU): NVIDIA GeForce GTX 960 o superior con soporte para OpenGL 4.0+
- Almacenamiento: 1 GB de espacio libre en disco (para ejecutable, modelos, texturas y recursos)

- Pantalla: Resolución mínima de 1280x720

Software

- Sistema Operativo: Windows 10 o superior (64 bits)
- Librerías requeridas(incluidas en la instalación):
    - OpenGL 4.0 o superior (soportado por la mayoría de GPUs modernas).
    - GLFW (librería para gestión de ventanas y entrada).
    - GLEW (extensiones de OpenGL).
    - SOIL2 (carga de texturas).
    - OpenAL (audio).

Requisitos adicionales

- Permisos de ejecución: El programa debe ejecutarse con permisos suficientes para abrir ventanas emergentes y acceder a archivos externos (modelos y texturas).
- Soporte para ejecución de archivos .exe en modo Release.
- Se debe mantener la estructura de carpetas y librerías adjuntas junto al ejecutable para asegurar la correcta carga de recursos y dependencias.

¡Bienvenido!

Este proyecto de simulación 3D creado con OpenGL te sumergirá en una experiencia virtual basada en la casa del programa de televisión *Un Show Más*, donde tendrás la libertad de explorar tanto el exterior como el interior, incluyendo una detallada recreación de la cocina y la habitación de *Mordecai* y *Rigby*.

Pasos para iniciar la experiencia

1. Descargar la carpeta **Release.zip** de nuestro repositorio en Github ubicado en la rama main en la siguiente liga: https://github.com/joshuaqm/319098147_proyectoFinal_gpo05
2. Descomprimir y ubicar el archivo ejecutable llamado **ProyectoFinal.exe.**

3. Hacer doble clic sobre él.
4. Esperar unos segundos mientras se carga el entorno 3D. Una vez iniciado, se abrirá una ventana con la escena principal.

Escenario Inicial

Al comenzar la simulación, aparecerás frente a la casa, observando su fachada. La ambientación incluye elementos como:

- Césped
- Cielo
- Árboles y arbustos
- Un faro decorativo

Estos elementos han sido diseñados para crear un entorno visualmente agradable y coherente con la temática del proyecto.

Controles de Movimiento

Para desplazarse dentro del entorno virtual, se deben utilizar las siguientes teclas:

Tecla	Acción
W	Avanzar
S	Retroceder
A	Moverse a la izquierda
D	Moverse a la derecha
Espacio	Elevarse (moverse hacia arriba)

Shift	Descender (moverse hacia abajo)
Ctrl + (W/A/S/D)	Aumentar la velocidad de movimiento
Mouse	Controlar la orientación de la cámara para mirar alrededor en la escena

Controles de Iluminación General

Puedes cambiar las condiciones de iluminación de la escena utilizando las siguientes teclas:

Tecla	Función
N	Alternar entre iluminación de día y de noche
L	Encender o apagar las luces puntuales de la escena
F	Encender o apagar la luz spotlight (linterna)

NOTA: La primera luz puntual está ubicada fuera de la casa, en el faro. La segunda se encuentra dentro, en la recreación de la cocina, y la tercera está en la lámpara sobre la mesa, dentro de la habitación. Asegúrate de acercarte a estos objetos y activar las luces para observar el efecto que producen sobre los demás elementos.

Interacción Dentro de la Casa

Al ingresar a la casa y recorrer su interior, hay una recreación de la cocina. Dentro de ella, puedes controlar manualmente la posición de una fuente de luz para observar cómo afecta la iluminación a los distintos objetos.

Tecla	Movimiento de la luz
U	Mover la luz hacia adelante
J	Mover la luz hacia atrás
H	Mover la luz hacia la izquierda
K	Mover la luz hacia la derecha
Flecha ↑ (arriba)	Elevar la luz
Flecha ↓ (abajo)	Bajar la luz

Controles de Interacción de Sonido

Subiendo todas las escaleras de la casa, hay una habitación en la que encontrarás un radio. Dentro de ella, puedes controlar la reproducción de audio que produce dicho objeto.

Tecla	Función
M	Alternar entre reproducción o pausa del audio en la escena

Manual Técnico

Objetivos

Objetivo general

Aplicar y demostrar los conocimientos adquiridos en el curso de Computación Gráfica mediante el desarrollo de una recreación tridimensional (3D) utilizando la biblioteca OpenGL y algún software de modelado.

Objetivos específicos

- Modelar una fachada y un espacio (real o ficticio) que incluyan al menos cinco objetos en cada cuarto, con un alto grado de fidelidad visual respecto a imágenes de referencia.
- Implementar ambientación coherente que complemente el entorno virtual, respetando criterios estéticos y técnicos.
- Documentar el proceso de desarrollo mediante un manual técnico que incluya diagramas, análisis de costos del proyecto, y la documentación del código fuente.
- Proporcionar un manual de usuario que explique de forma clara la interacción dentro del entorno tridimensional generado.
- Utilizar herramientas de modelado como *Maya Autodesk 2025* para generar modelos 3D, que posteriormente serán integrados al entorno OpenGL, asegurando una transición eficiente entre software de diseño y entorno de programación gráfica.

Alcances

Este proyecto tiene como objetivo principal desarrollar un recorrido virtual tridimensional e interactivo de un entorno arquitectónico, que incluye una fachada y dos espacios interiores, utilizando la biblioteca gráfica OpenGL versión 3.

Los alcances específicos del proyecto incluyen:

- Aplicar texturas, iluminación y ambientación que mejoren el realismo visual.
- Implementar cuatro animaciones.

- Desarrollar una interfaz básica para la interacción del usuario dentro del entorno virtual, conforme al manual de usuario.
- Documentar detalladamente el proyecto, incluyendo diagramas, metodología de desarrollo, análisis de costos y explicación del código implementado.
- La entrega del proyecto en un repositorio público de GitHub, manteniendo el historial de cambios y sin archivos comprimidos.
- Exportar e integrar correctamente los modelos 3D creados en *Maya Autodesk 2025* al entorno OpenGL, garantizando compatibilidad de formatos, optimización de geometría y preservación de texturas y materiales dentro de las limitaciones técnicas establecidas por el proyecto.

Limitaciones

Se presentan diversas restricciones tanto técnicas como académicas que definieron su alcance. Estas limitaciones se detallan a continuación:

- No se permite reproducir espacios con restricciones legales, como instalaciones de la UNAM, edificios gubernamentales o lugares prohibidos.
- La conversión y uso de modelos 3D están limitados por las restricciones de tamaño y rendimiento de OpenGL, limitando la complejidad geométrica y el peso máximo (100 MB por modelo).
- Las capacidades del equipo de hardware disponible presentan restricciones que pueden afectar la fluidez y rendimiento óptimo durante el desarrollo y ejecución del proyecto.
- El proyecto es individual; aunque la primera parte se trabajó en equipo (fachada y un cuarto), la entrega final debe incluir elementos recreados y amueblados de forma individual.

Análisis de costos del proyecto

A continuación se presenta el análisis y desglose detallado de los costos incurridos durante el desarrollo y despliegue del producto final, considerando tanto los recursos humanos como técnicos empleados. Este análisis incluye la inversión de tiempo, licenciamiento de software,

adquisición de hardware y otros insumos necesarios para la correcta ejecución y entrega del proyecto, proporcionando una base sólida para la estimación del costo total y el precio de venta sugerido.

Costos de desarrollo

Concepto	Descripción	Cantidad tiempo	Costo Unitario (MXN)	Costo Total (MXN)
Costos de capital humano				
Desarrollo	Trabajo en modelado, programación y aprendizaje	60 horas	$120	$7,200
Costos de software				
Licencia Autodesk Maya	Licencia profesional mensual	1 mes	$5,000	$5,000
Suscripción ChatGPT Plus	Generación de texturas y adaptación de código	1 mes	$400	$400
Costos de hardware				
Computadora de escritorio para desarrollo	Equipo capaz de ejecutar Maya y OpenGL (procesador, RAM, etc.)	1 unidad	$18,000 (estimado)	$18,000
Otros costos				
Otros recursos	Texturas, modelos de pago y herramientas adicionales	-	-	$3,000 (estimado)
			Subtotal	$33,600

Precio de venta estimado

Se propone un precio de venta de **$45,000 MXN**, considerando:

- Costos directos de desarrollo, licencias y hardware.
- Valor agregado por calidad, animaciones y funcionalidad interactiva.
- Tiempo invertido y especialización técnica requerida.
- Margen para cubrir riesgos, mantenimiento y mejoras futuras.

El precio de venta propuesto de **$45,000 MXN** se fundamenta en factores que van más allá de los costos directos de desarrollo y materiales. Primero, incluye la recuperación de los costos invertidos en licencias de software, hardware y recursos necesarios para la producción, los cuales suman un subtotal de **$33,600 MXN**.

Además, este precio considera el valor agregado del proyecto, reflejado en la calidad visual obtenida, las animaciones programadas y la funcionalidad interactiva implementada para ofrecer una experiencia inmersiva y profesional al usuario final.

El tiempo invertido, estimado en 60 horas, representa una inversión significativa en aprendizaje y desarrollo, que abarca tanto la adquisición de conocimientos técnicos en Autodesk Maya y OpenGL como la aplicación práctica de estos en el proyecto.

Finalmente, el precio incluye un margen adecuado para cubrir riesgos inherentes al desarrollo de software, posibles costos de mantenimiento y futuras mejoras, así como para remunerar la especialización y dedicación del desarrollador, garantizando la viabilidad económica y profesional del proyecto.

Diagrama de Gantt

El siguiente diagrama de Gantt muestra el flujo de trabajo realizado a lo largo del tiempo para lograr los objetivos del proyecto, incluyendo descripciones sobre las actividades realizadas.

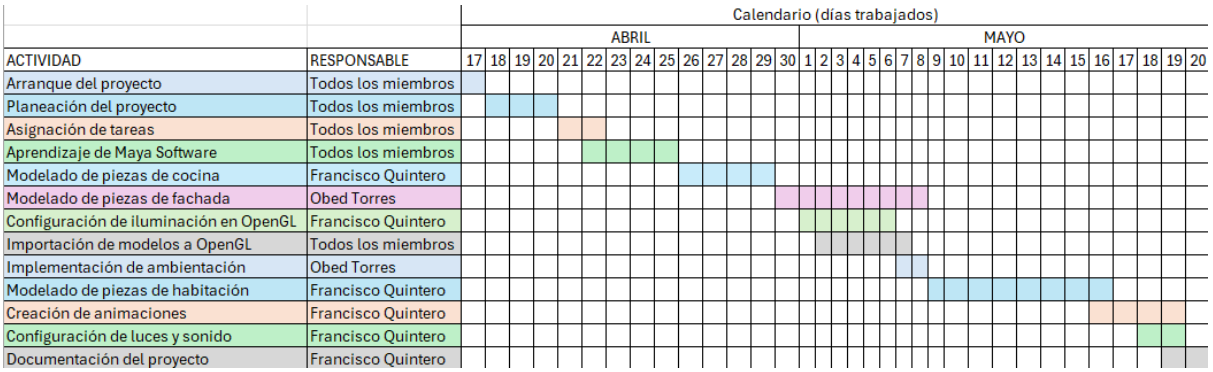

Figura 0. Diagrama de Gantt del proyecto.

Diagrama de flujo

El flujo general del software se organiza en los siguientes pasos:

1. **Inicio de la aplicación**
    - Ejecución del archivo ProyectoFinal.exe (modo Release) o desde entorno de desarrollo (Debug).
    - Inicialización de GLFW y creación de ventana OpenGL.
    - Configuración de GLEW para acceso a funciones modernas de OpenGL.
    - Obtención del tamaño del framebuffer.
2. **Carga y configuración de recursos**
    - Carga de librerías y archivos encabezado para OpenGL, GLFW, GLEW, SOIL2, stb_image, glm y OpenAL para audio.
    - Definición y carga de shaders (lightingShader, lampShader).
    - Carga de modelos 3D (.obj, .fbx), agrupados por ambientes (exterior, cocina, habitación, etc.).
    - Configuración de VBO y VAO para geometría básica (e.g., cubos para luces).
    - Inicialización y configuración del audio con OpenAL, carga de archivo WAV y configuración 3D de la fuente de audio.
3. **Inicialización del entorno gráfico y variables globales**
    - Configuración de viewport con dimensiones actuales.
    - Definición de variables globales: cámara, posición de luces, estados de animaciones, flags para interacción (linterna, luces, día/noche).
    - Asignación de callbacks:
        - Teclado (KeyCallback)
        - Mouse (MouseCallback)
        - Función de movimiento (DoMovement)
    - Configuración de entrada del mouse para captura y movimiento relativo.
4. **Bucle principal de renderizado (Main Loop)**
    - Cálculo del deltaTime para animaciones y movimientos suaves.
    - Actualización de animaciones:
        - Animación de drift circular del carrito.
        - Transición día/noche con interpolación suave.
        - Animación de salto del personaje Rigby en trampolín.

    - Movimiento de árboles simulado por efecto de viento.
    - Caída de hojas de los árboles.
- Actualización de audio:
    - Actualización de posición y orientación del "listener" (cámara).
    - Control de reproducción de audio.
- Procesamiento de eventos de entrada (teclado y mouse).
- Aplicación de movimientos según input del usuario.
- Limpieza del buffer de color y profundidad.
- Configuración y envío de variables uniformes al shader para iluminación (luz direccional día/noche, luces puntuales, spotlight linterna).
- Renderizado de modelos 3D con sus transformaciones (traslación, rotación, escala).
- Renderizado de luces puntuales como objetos visuales (cubos).
- Swap buffers para mostrar frame renderizado.

5. Interacción del usuario

- Navegación 3D libre con cámara (WASD, espacio, shift).
- Control en tiempo real de:
    - Posición de la luz puntual principal (teclas H, K, U, J, Up, Down).
    - Estado de iluminación global (tecla L para encender/apagar luces).
    - Cambio día/noche (tecla N con transición suave).
    - Activación/desactivación de linterna (tecla 1).
    - Control de reproducción de audio (tecla M).
- Movimiento de cámara influye en dirección y posición del spotlight (linterna).
- Animaciones activas y sincronizadas en tiempo real (carrito, salto, viento).

6. Cierre de la aplicación

- Al cerrar ventana se termina bucle de renderizado.
- Limpieza y liberación de recursos gráficos (buffers, shaders, modelos).
- Detención y liberación de recursos de audio (fuente, buffer, contexto OpenAL).
- Terminación de GLFW.

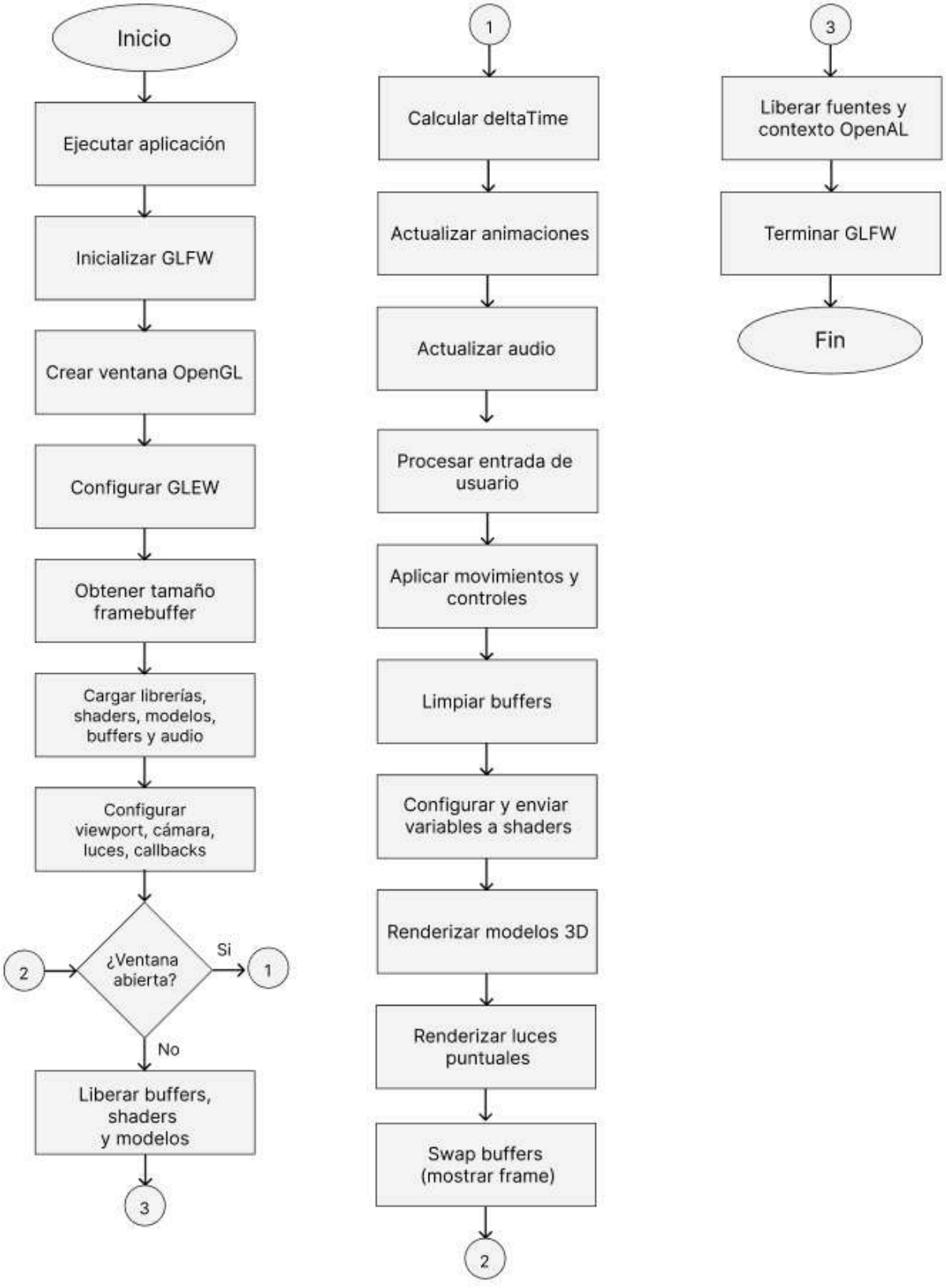

Figura 1. Diagrama de flujo del programa

Documentación del código

El código incluye la implementación de modelos 3D de una casa y otros objetos para ambientar el exterior además de interactuar con la escena mediante la cámara y fuentes de luz. A continuación, se describe cada sección del código:

1. Inclusión de Bibliotecas

- **GLEW (OpenGL Extension Wrangler)**: Carga extensiones de OpenGL para acceder a funciones modernas de renderizado.
- **GLFW**: Maneja la creación de ventanas, entradas de teclado/ratón y eventos del sistema.
- **STB Image**: Permite cargar texturas desde archivos de imagen (como PNG o JPG).
- **GLM (OpenGL Mathematics)**: Proporciona estructuras para operaciones matemáticas (vectores, matrices) esenciales para gráficos 3D.
- **SOIL2**: Facilita la carga de texturas para modelos 3D.
- **dr_wav**: Biblioteca minimalista para decodificar archivos de audio WAV.
- **OpenAL**: API de audio para reproducción 3D.
- **Random (C++ Standard Library)**: Generación de números aleatorios para efectos como la caída de hojas.
- **Otras bibliotecas de modelado (Shader, Camera, Model)**: Permiten cargar y renderizar los modelos 3D, aplicar shaders y manipular la cámara.

```
#include <iostream>
#include <cmath>
// GLEW
#include <GL/glew.h>
// GLFW
#include <GLFW/glfw3.h>
// Other Libs
#include "stb_image.h"
#include <random>
// GLM Mathematics
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
//Load Models
#include "SOIL2/SOIL2.h"
// Audio includes
#include <AL/al.h>
#include <AL/alc.h>
// Other includes
#include "Shader.h"
#include "Camera.h"
#include "Model.h"
#define DR_WAV_IMPLEMENTATION
#include "dr_wav.h"
```

Figura 2. Inclusión de bibliotecas y archivos de encabezado.

2. Configuración de la Cámara

La cámara se define utilizando una clase Camera, que permite el movimiento de la vista 3D dentro de la escena. El código usa funciones como ProcessKeyboard y ProcessMouseMovement para mover y rotar la cámara según las entradas del teclado y del ratón.

```
// Function prototypes
// Funciones de interaccion
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode);
void MouseCallback(GLFWwindow* window, double xPos, double yPos);
void DoMovement();
// Camera
Camera  camera(glm::vec3(0.0f, 0.0f, 135.0f));
```

Figura 3. Funciones de movimiento y declaración de cámara sintética.

3. Variables Globales

El programa utiliza variables globales para controlar animaciones y efectos dinámicos. Algunas de las más relevantes incluyen:

- **Iluminación:**
    - **pointLightPositions**: Almacena las posiciones de las luces puntuales en la escena.
    - **deltaTime y lastFrame**: Controlan el tiempo entre cuadros, permitiendo un movimiento de cámara suave y consistente.
- **Animaciones**:
    - **driftAngle y rotationSpeed**: Controlan el movimiento circular del carrito de golf.
    - **jumpProgress y squashFactor**: Gestionan el salto y deformación del personaje (Rigby).
    - **hojasYOffset**: Desplazamiento vertical para simular la caída de hojas de los árboles.
- **Transiciones**:
    - **timeOfDay** y **transitionActive**: Interpolan entre estados de día y noche.
    - **isNight**: Bandera para alternar el modo nocturno.
- **Efectos de Luz**:
    - **flashlightOn**: Habilita/deshabilita la linterna (spotlight).

    - **lightsOff**: Apaga/enciende las luces puntuales de la escena.
- **Audio**:
    - **audioPlaying**: Estado de reproducción del sonido ambiental.
    - **audioSourcePos**: Posición fija de la fuente de audio en el mundo 3D.

Estas variables se modifican en tiempo real mediante interacciones del usuario (teclas N, L, F, M) o durante las actualizaciones del bucle principal (deltaTime).

```
// Luces puntuales
glm::vec3 pointLightPositions[] = {
    glm::vec3(-14.0f, 12.5f, -11.0f),
    glm::vec3(-17.0f, 17.0f, 80.0f),
    glm::vec3(12.45f, 30.3f, -16.4f),
};

// Deltatime
GLfloat deltaTime = 0.0f;   // Time be
GLfloat lastFrame = 0.0f;   // Time of
```

Figura 4. Declaración de luces puntuales.

4. Inicialización de GLFW y GLEW

- **GLFW** se inicializa y se crea una ventana para renderizar la escena 3D. Si la ventana no se crea correctamente, el programa finaliza.
- **GLEW** se inicializa para asegurarse de que las funciones de OpenGL estén disponibles.

```
// Create a GLFWwindow object that we can use for GLFW's functions
GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "Proyecto Final

if (nullptr == window)
{
    std::cout << "Failed to create GLFW window" << std::endl;
    glfwTerminate();

    return EXIT_FAILURE;
}
```

Figura 5. Inicialización de ventana.

5. Shaders

Se cargan dos shaders:

- **lightingShader**: Se usa para los objetos principales de la escena, como la casa, árboles, y arbustos.
- **lampShader**: Se usa para representar las fuentes de luz (luces puntuales) en la escena.

```
Shader lightingShader("Shader/lighting.vs", "Shader/lighting.frag");
Shader lampShader("Shader/lamp.vs", "Shader/lamp.frag");
```

Figura 6. Inicialización de Shaders de iluminación.

6. Carga de Modelos

Se cargan modelos 3D de objetos de la escena utilizando la clase Model. Los modelos representan la casa, los árboles, las puertas, las ventanas, la estufa, la cama y otros objetos de la cocina y la casa.

```
//Modelos exterior
Model Casa((char*)"Models/casa.obj");
Model Arbusto((char*)"Models/arbusto.fbx");
Model Arbol((char*)"Models/arbol.obj");
Model Farol((char*)"Models/faro.obj");
Model Cesped((char*)"Models/cesped.obj");
Model Cielo((char*)"Models/cieloo.obj");

// Modelos interior
Model Puerta_Cocina((char*)"Models/puerta_cocina.obj");
Model Ventana_Cocina((char*)"Models/ventana_cocina.obj");
Model Pared_Cocina((char*)"Models/pared_cocina.obj");
```

Figura 7. Importación de modelos en 3D.

7. Definición de VBO y VAO

- **VBO (Vertex Buffer Object)**: Almacena los vértices del cubo que se utilizará como base de la escena.
- **VAO (Vertex Array Object)**: Organiza las configuraciones de los atributos de vértices, como las posiciones y normales.

```
// First, set the container's VAO (and VBO)
GLuint VBO, VAO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
glBindVertexArray(VAO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
```

Figura 8. Inicialización de VBO y VAO.

8. Configuración de las Luces

El programa incluye tres tipos de luces:

- **Luz direccional:** Simula el sol/luna, interpola entre valores diurnos/nocturnos (color, dirección e intensidad), se controla con la tecla N.
- **Luces puntuales:** Tres fuentes de luz fijas en la escena, intensidad controlada por la tecla L (on/off), cada una con atenuación y color personalizados. La primera luz puntual está ubicada libremente en la cocina, la segunda simula la luz generada por el faro y la tercera simula la luz generada por la lámpara de la habitación
- **Spotlight (linterna):** Sigue la dirección de la cámara, cono de luz ajustable (núcleo y bordes), se activa/desactiva con la tecla F

Todas las luces se calculan en el shader y responden dinámicamente a interacciones.

```
// Luz puntual 1
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"), 0.2f * light
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), 1.0f * light
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"), 0.5f * ligh
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].linear"), 0.09f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"), 0.032f);
```

Figura 9. Configuración de la luz puntual 1.

9. Movimiento de la Cámara

El movimiento de la cámara se controla mediante teclas (W, A, S, D, Space, Shift) y también se ajusta cuando se presionan teclas como Control. El ratón se utiliza para girar la cámara y mirar alrededor de la escena.

```
void DoMovement()
{
    float multiplier = 1.0f;
    if (keys[GLFW_KEY_LEFT_CONTROL])
    {
        multiplier = 6.0f;
    }

    if (keys[GLFW_KEY_W]) {
        camera.ProcessKeyboard(FORWARD, deltaTime * multiplier);
    }
```

Figura 10. Función que controla el movimiento de la cámara por medio de entradas del teclado.

10. Dibujar Modelos

Se configura la matriz de transformación para cada modelo (traslación, rotación y escala) antes de dibujar en la pantalla. Esto incluye tanto los modelos exteriores (como la casa, arbustos, árboles) como los objetos interiores (como la estufa, mesa, sillas, cama, buró).

```
//Silla 1
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-15.3, 6.4f, -5.0f));
model = glm::scale(model, glm::vec3(1.2f, 1.2f, 1.2f));
model = glm::rotate(model, glm::radians(315.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Silla.Draw(lightingShader);
```

Figura 11. Transformaciones y dibujo del modelo.

11. Configuración de las luces puntuales

Las luces puntuales se dibujan usando un pequeño cubo para representar la luz en las posiciones definidas. La luz cambia su intensidad y color dependiendo de la entrada del usuario (tecla L).

```
// Dibujar ejes de coordenadas
lampShader.Use();
glUniformMatrix4fv(glGetUniformLocation(lampShader.Program, "view"), 1, GL_
glUniformMatrix4fv(glGetUniformLocation(lampShader.Program, "projection"),
for (int i = 0; i < 3; i++) {
    glm::mat4 model(1);
    model = glm::translate(model, pointLightPositions[i]);
    model = glm::scale(model, glm::vec3(0.2f));
    glUniformMatrix4fv(glGetUniformLocation(lampShader.Program, "model"), 1
    glUniform3f(glGetUniformLocation(lampShader.Program, "lampColor"), 1.0

    glBindVertexArray(VAO);
    glDrawArrays(GL_TRIANGLES, 0, 36);
    glBindVertexArray(0);
}
```

Figura 12. Dibujo de luces puntuales en la escena.

12. Funciones de Audio

El programa incluye un sistema de audio 3D utilizando OpenAL para reproducir sonidos posicionales en la escena. Las funciones clave son:

- **loadWavFile**: Carga un archivo de audio en formato WAV y lo prepara para su reproducción.

- **Parámetros**:
  - filename: Ruta del archivo WAV.
  - buffer: Buffer de OpenAL donde se almacenará el audio.
- **Retorno**: bool (éxito o fallo).

```
bool loadWavFile(const char* filename, ALuint buffer) {
    drwav wav;
    if (!drwav_init_file(&wav, filename, NULL)) {
        std::cerr << "Error al cargar el archivo WAV: " << filename << std::endl;
        return false;
    }

    float* pSampleData = (float*)malloc(wav.totalPCMFrameCount * wav.channels * sizeof(float));
    drwav_read_pcm_frames_f32(&wav, wav.totalPCMFrameCount, pSampleData);

    short* pcmData = (short*)malloc(wav.totalPCMFrameCount * wav.channels * sizeof(short));
```

Figura 13. Fragmento de la función que carga el archivo de audio.

- **initAudio**: Inicializa el dispositivo y contexto de OpenAL, carga el archivo de audio y configura la fuente de sonido.
  - **Parámetros**: wavFile (ruta del archivo de audio).
  - **Retorno**: bool (éxito o fallo).

```
bool initAudio(const char* wavFile) {
    audioDevice = alcOpenDevice(nullptr);
    if (!audioDevice) return false;

    audioContext = alcCreateContext(audioDevice, nullptr);
    if (!audioContext || !alcMakeContextCurrent(audioContext)) {
        if (audioContext) alcDestroyContext(audioContext);
        alcCloseDevice(audioDevice);
        return false;
    }

    alGenBuffers(1, &audioBuffer);
    alGenSources(1, &audioSource);

    if (!loadWavFile(wavFile, audioBuffer)) return false;
```

Figura 14. Función que inicializa el audio.

- **updateListener**: Actualiza la posición y orientación del oyente (cámara) y controla la reproducción basada en la distancia a la fuente de audio.
  - **Parámetros**:
    - listenerPos: Posición de la cámara.
    - listenerDir: Dirección hacia donde mira la cámara.

```
void updateListener(const glm::vec3& listenerPos, const glm::vec3& listenerDir) {
    // Calcula la distancia entre cámara y fuente de audio
    float distance = glm::distance(listenerPos, audioSourcePos);

    ALint state;
    alGetSourcei(audioSource, AL_SOURCE_STATE, &state);

    if (distance > 40.0f) {
        // Pausa la fuente solo si está sonando y el usuario NO la ha pausado manu
        if (state == AL_PLAYING && audioPlaying) {
            alSourcePause(audioSource);
            //std::cout << "Audio pausado automáticamente por distancia\n";
        }
    }
    else {
        // Reproduce la fuente solo si está pausada y el usuario quiere que suene
        if (state != AL_PLAYING && audioPlaying) {
```

Figura 15. Fragmento de la función que controla la atenuación y el sonido espacial.

- **toggleAudioPlayback**: Alterna entre reproducir y pausar el audio. Se activa con la tecla M.

```
void toggleAudioPlayback() {
    ALint state;
    alGetSourcei(audioSource, AL_SOURCE_STATE, &state);

    if (state == AL_PLAYING) {
        alSourcePause(audioSource);
        audioPlaying = false;
    }
    else {
        alSourcePlay(audioSource);
        audioPlaying = true;
    }
}
```

Figura 16. Función que alterna entre reproducir y pausar.

- **cleanupAudio**: Libera los recursos de OpenAL al finalizar el programa.

```
void cleanupAudio() {
    alSourceStop(audioSource);
    alDeleteSources(1, &audioSource);
    alDeleteBuffers(1, &audioBuffer);
    alcMakeContextCurrent(nullptr);
    alcDestroyContext(audioContext);
    alcCloseDevice(audioDevice);
}
```

Figura 17. Función que libera los recursos de OpenAL

13. Funciones de Animación

El programa incluye varias animaciones dinámicas creadas a partir de funciones que aplican transformaciones en tiempo real a los modelos:

- **animateCircularDrift**: Anima un carrito de golf en movimiento circular alrededor de un punto pivote.
    - **Mecánica**:
        - driftAngle: Ángulo que se incrementa con el tiempo.
        - circleRadius: Radio de la trayectoria circular.

```
void animateCircularDrift(float deltaTime) {
    driftAngle += rotationSpeed * deltaTime; // Aumenta el angulo continuamente

    // Mantener el angulo entre 0 y 2Ï€ para evitar overflow
    if (driftAngle > 2 * glm::pi<float>()) {
        driftAngle -= 2 * glm::pi<float>();
    }
}
```

Figura 18. Función que crea la animación de drift del carrito de golf.

- **UpdateDayNightTransition**: Interpola suavemente entre los estados de día y noche.
    - **Variables clave**:
        - timeOfDay: Valor entre 0 (día) y 1 (noche).
        - transitionSpeed: Velocidad de la transición.

```
void UpdateDayNightTransition(float deltaTime) {
    if (!transitionActive) return;

    float target = isNight ? 1.0f : 0.0f;
    float direction = (target > timeOfDay) ? 1.0f : -1.0f;

    timeOfDay += direction * deltaTime * transitionSpeed;

    // Clamp y finalización
    if ((direction > 0.0f && timeOfDay >= target) || (direction < 0.0f && timeOfDay <= target)) {
        timeOfDay = target;
        transitionActive = false;  // transición terminada
    }
}
```

Figura 19. Función que realiza una transición suave entre iluminación de día y noche.

- **updateTrampolineJump**: Simula el salto de un personaje (Rigby) en un trampolín con efectos de "squash and stretch".
    - **Mecánica**:
        - jumpProgress: Controla la altura del salto.
        - squashFactor: Compresión del modelo durante el salto.

```
void updateTrampolineJump(float deltaTime) {
    if (isAscending) {
        jumpProgress += jumpSpeed * deltaTime;
        if (jumpProgress >= 1.0f) {
            jumpProgress = 1.0f;
            isAscending = false;
        }
        // Squash disminuye durante el ascenso
        squashFactor = 1.0f - jumpProgress;
    }
    else {
        jumpProgress -= jumpSpeed * deltaTime;
        if (jumpProgress <= 0.0f) {
            jumpProgress = 0.0f;
            isAscending = true;
        }
        // Squash aumenta durante el descenso
        squashFactor = jumpProgress;
    }
}
```

Figura 20. Función que anima los saltos del personaje en el trampolin.

- **actualizarCaidaHojas**: Anima la caída aleatoria de hojas de los árboles.
    - **Variables**:
        - hojasYOffset: Desplazamiento vertical de cada hoja.
        - hojasVelocidadCaida: Velocidad de caída.

```
void actualizarCaidaHojas(float deltaTime) {
    for (int i = 0; i < 3; ++i) {
        hojasYOffset[i] -= hojasVelocidadCaida * deltaTime;

        // Multiplica el límite base por un factor aleatorio en cada reinicio
        float limiteInferiorAleatorio = hojasLimiteInferiorBase * disFactor(gen);

        if (hojasYOffset[i] < limiteInferiorAleatorio) {
            hojasYOffset[i] = hojasLimiteSuperior;
            // No necesitas almacenar el límite, lo recalculas cada vez
        }
    }
}
```

Figura 21. Función que anima la trayectoria de caída de las hojas.

- **applyTreeWind**: Aplica efectos de viento a árboles y arbustos (inclinación y escalado).
    - **Parámetros**:
        - model: Matriz de transformación del modelo.
        - offset: Desfase para variar el efecto entre objetos.

```
glm::mat4 applyTreeWind(glm::mat4 model, float offset) {
    // Rotación en Z (inclinación del árbol)
    float tilt = sin(windTime * windSpeed + offset) * windIntensity;
    model = glm::rotate(model, tilt, glm::vec3(0.0f, 0.0f, 1.0f));

    // Escalado en X/Y para simular hojas moviéndose
    float scaleX = 1.0f + sin(windTime * windSpeed * 2.0f + offset) * windIntensity * 0.1f;
    float scaleY = 1.0f - sin(windTime * windSpeed * 2.0f + offset) * windIntensity * 0.05f;
    model = glm::scale(model, glm::vec3(scaleX, scaleY, 1.0f));

    return model;
}
```

Figura 22. Función que simula movimiento del viento en árboles, arbustos y hojas

14. Configuración del Spotlight (Linterna)

La linterna es un foco direccional que sigue la cámara y se controla con la tecla F.

- **Estructura Spotlight**:
  Define propiedades como posición, dirección, cortes de luz (inner/outer cone) y atenuación.

```
// Parámetros del spotlight
struct Spotlight {
    glm::vec3 position;
    glm::vec3 direction;
    glm::vec3 ambient;
    glm::vec3 diffuse;
    glm::vec3 specular;
    float cutOff;
    float outerCutOff;
    float constant;
    float linear;
    float quadratic;
};
```

Figura 23. Estructura spotlight.

UpdateFlashlight: Actualiza la posición y dirección del spotlight para que coincida con la cámara.

- **Parámetros**:
  - camera: Referencia a la cámara.
  - flashlight: Referencia al spotlight.

```
void UpdateFlashlight(Camera& camera, Spotlight& flashlight) {
    flashlight.position = camera.GetPosition();
    flashlight.direction = camera.GetFront();
}
```

Figura 24. Función que mueve el spotlight a la dirección en la que apunte la cámara sintética.

Interacción:

- La tecla F alterna el estado (flashlightOn).
- En el shader, se envían las propiedades del spotlight solo si está activado.

```
if (keys[GLFW_KEY_F] && !keyPressed3) {
    flashlightOn = !flashlightOn;
    keyPressed3 = true;
}
else if (!keys[GLFW_KEY_F]) {
    keyPressed3 = false;
}
```

Figura 25. Controles que actualizan el estado de spotlight.

15. Bucle Principal de Renderizado

Dentro del bucle principal del programa:

- Se actualiza el deltaTime.
- Se ejecutan las funciones de animación.
- Se procesa la entrada del teclado y del ratón.
- Se limpia el búfer de color y profundidad.
- Se usan los shaders correspondientes para dibujar la escena.
- Se ajustan las luces según las entradas del usuario.

16. Funciones de Entrada

- **KeyCallback**: Detecta y responde a las teclas presionadas, como la tecla N para alternar entre día y noche y la tecla L para encender o apagar las luces.
- **MouseCallback**: Controla el movimiento del ratón para cambiar la dirección de la cámara.

17. Finalización

Cuando el bucle de la ventana termina (cuando se cierra la ventana o se presiona la tecla Escape), se terminan todos los recursos de GLFW y el programa finaliza.

Resultados

A continuación se muestran algunos de los modelos que fueron recreados en Maya Autodesk y algunos otros que fueron descargados de internet y sus implementaciones en el espacio virtual; además se muestran las imágenes de referencia utilizadas en comparativa con capturas de pantalla de la ejecución de la ejecución y el listado de elementos recreados.

Exterior

Para recrear la fachada de la casa fue necesario modelar los siguientes objetos:

- Chimenea
- Escaleras
- Ventanas (cuadradas, rectangulares, ventanales)
- Pieza en L
- Balcón
- Columnas
- Pasamanos
- Puerta (principal y del garage)

Aunque al final todo fue unido en una sola pieza, requirió el modelado de elementos individuales para su manejo más preciso de cada una de las piezas.

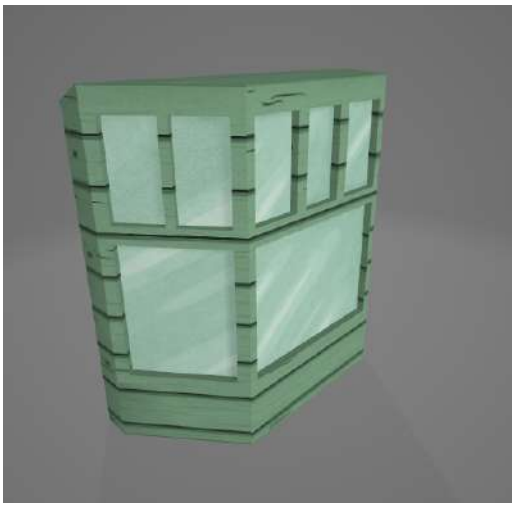

Figura 26. Balcón recreado en Maya

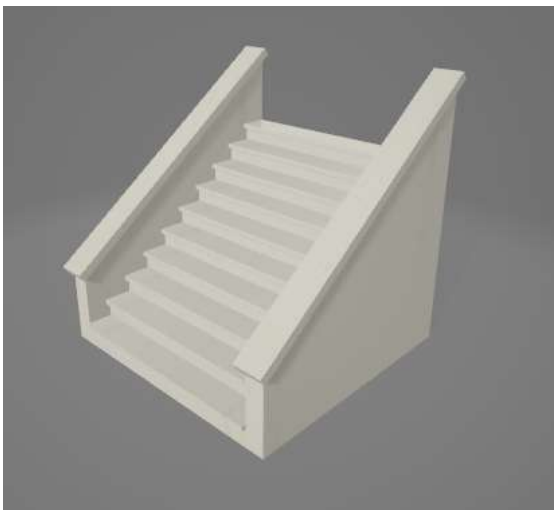

Figura 27. Escaleras recreadas en Maya

Figura 28. Imagen de referencia de la fachada

Figura 29. Captura de pantalla de la fachada

Para agregar ambientación como se ve en la imagen de referencia, decidimos importar los siguientes modelos:

- Faro
- Árbol
- Arbusto
- Carrito de golf

Además, agregamos un modelo para hacer una simulación de skybox para agregar más detalles a la casa y complementar la escena.

Figura 30. Faro importado de Sketchfab.

Figura 31. Carrito importado de Sketchfab.

Figura 32. Captura de pantalla de la ambientación

Interior

Para recrear el interior (cocina) fue necesario modelar los siguientes objetos:

- Pieza de piso y techo
- Pared
- Ventana
- Pieza de madera pared
- Refrigerador
- Mesa
- Silla
- Alacena inferior
- Alacena superior
- Alacena grande
- Tostadora
- Estufa
- Campana

Figura 33. Estufa recreada en Maya

Figura 34. Silla recreada en Maya

Figura 35. Imagen de referencia de la cocina

Figura 36. Captura de pantalla de la cocina

Para recrear la habitación fue necesario modelar los siguientes objetos:

- Archivero
- Buró
- Lámpara
- Radio
- Cajones
- Cama
- Pared de la habitación
- Pared de la habitación con puerta
- Rigby
- Televisión de la habitación
- Trampolin
- Ventana de la habitación y cortinas

Figura 37. Ventana y cortinas recreadas en Maya

Figura 38. Trampolin recreado en Maya

Figura 39. Imagen de referencia de la habitación.

Figura 40. Captura de pantalla de la habitación.

Animaciones

A continuación, se muestran capturas que ilustran el correcto funcionamiento de las animaciones, acompañadas su ubicación exacta dentro del entorno.

Figura 41. Animación de carrito de golf haciendo drifting fuera del garage de la casa.

Figura 41. Animación de transición entre iluminación de día y noche (escena de día)

Figura 42. Animación de transición entre iluminación de día y noche (escena de noche)

Figura 43. Animación de movimiento de vegetación por acción del viento. Animación de caída de hojas de los árboles.

Figura 42. Animación de personaje saltando en el trampolín de la habitación.

Figura 43. Reproducción de audio desde la radio de la habitación.

Iluminación

A continuación, se presentan los efectos de las luces puntuales y los focos (spotlights) en la escena, ambientada con una luz direccional tenue.

Figura 44. Luz del faro exterior (Fuente de luz puntual).

Figura 44. Luz de la fuente de la cocina (Fuente de luz puntual).

Figura 45. Luz de la lámpara de la habitación (Fuente de luz puntual).

Figura 46. Luz de la cámara sintética (Fuente de luz spotlight).

Conclusiones

Con el desarrollo de este proyecto se logró recrear un espacio virtual desde cero, aplicando los conocimientos adquiridos en la asignatura de computación gráfica. Fue un gran reto, especialmente en el modelado, la implementación de la iluminación dinámica y la reproducción de audio dentro del entorno virtual.

Configurar correctamente las animaciones por medio de funciones que aplican transformaciones fue una experiencia que me pareció interesante, ya que se pueden lograr efectos visuales que enriquecen la dinámica del entorno y mejoran la experiencia virtual para el usuario. La experiencia previa con la cámara sintética y funciones de OpenGL fue fundamental para lograr una experiencia satisfactoria al interactuar con el entorno; ajustar parámetros como posición, color e intensidad de las luces fue posible gracias a lo aprendido en el laboratorio. Aunque se enfrentaron dificultades técnicas, como problemas con las normales, materiales y texturas al exportar desde Maya, se logró resolverlos realizando múltiples pruebas unitarias, ajustando distintos parámetros hasta que funcionara correctamente.

Los conocimientos adquiridos gracias al proyecto podrían aplicarse a proyectos más grandes de desarrollo de entornos 3D, modelado o inclusive para seguir mejorando este mismo programa para añadir más interacciones y convertirlo en un videojuego sencillo.

Referencias

[1] Buy Autodesk Maya 2026 Software | 3D Animation Software. (s. f.). https://www.autodesk.com/products/maya/overview

[2] Comment, D. S. J. 1. 2. 0. (s. f.). *Las 10 mejores plantillas de estimación de costos con muestras y ejemplos*. https://www.slideteam.net/blog/las-10-mejores-plantillas-de-estimacion-de-costos-con-muestras-y-ejemplos

[3] Montilla, C. (2020, 18 marzo). *Cuánto cobrar por un Render / Modelado 3D - [El método 100% efectivo]*. Yo Soy Arquitecto. https://yosoyarquitecto.com/cuanto-cobrar-por-un-render-modelado/

[4] Pricing ChatGPT. (s. f.). OpenAI. https://openai.com/es-419/chatgpt/pricing/

[5] "Rigby || Regular Show" (https://skfb.ly/pqs6T) by metekrts is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

[6] "Mordecai" (https://skfb.ly/ozw7D) by D.p is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

[7] "Stylized Bush" (https://skfb.ly/oTDpD) by Daniel is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

[8] "Pillow" (https://skfb.ly/6ntWQ) by 3DMish is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

[9] "Lemon Leaf" (https://skfb.ly/6pJnz) by YouniqueĪdeaStudio is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

[10] "Cart || Regular Show" (https://skfb.ly/pqssG) by metekrts is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).