Manual Técnico del Proyecto

ESPAÑOL

Introducción

Este manual técnico documenta el proyecto final desarrollado para el laboratorio de la asignatura "Computación Gráfica e Interacción Humano-Computadora" en la Facultad de Ingeniería de la Universidad Nacional Autónoma de México, durante el semestre 2025-2. El proyecto fue realizado en equipo por Francisco Joshua Quintero Montero y Obed Torres Pimentel, consiste en la creación de un entorno 3D interactivo inspirado en la casa de la serie *Un Show Más*, utilizando OpenGL y GLFW en el lenguaje de programación C++.

El entorno recrea una casa con dos espacios principales: una **cocina**, desarrollada en equipo en el laboratorio de la materia, que incluye modelos tridimensionales, iluminación dinámica, sombras, una figura tipo skybox y una cámara sintética para navegación; y una **sala de estar**, creada individualmente por mí, Obed Torres Pimentel, con cinco objetos específicos y animaciones interactivas para un carrito, un televisor, la puerta del garaje y la puerta principal. El proyecto integra conceptos de modelado 3D, texturizado, iluminación, animaciones y optimización gráfica aprendidos en el curso.

Objetivos

Desarrollar una aplicación gráfica interactiva en tres dimensiones utilizando OpenGL y GLFW en C++, con las siguientes características:

- **Modelos tridimensionales**: Diseñar e implementar una fachada de casa, una cocina con múltiples objetos y una sala de estar con cinco objetos específicos, todos modelados con precisión utilizando Maya Autodesk 2025 y optimizados para OpenGL.
- **Iluminación dinámica**: Implementar luces direccionales, luces puntuales (*point lights*), sombras y transiciones día/noche para mejorar el realismo del entorno.
- **Animaciones**: Incorporar al menos cuatro animaciones, incluyendo 2 simples (puerta del garaje y puerta principal) y 2 complejas (animaciones del televisor con ruido estático y movimiento del carrito con rotación y traslación).
- **Interacción del usuario**: Permitir la navegación mediante una cámara sintética controlada por teclado y ratón, y la manipulación de luces y objetos mediante teclas específicas.

- **Ambientación**: Incluir una figura que simula una skybox para que se vea mejor la ambientación, además de agregar objetos para ambientar el entorno
- **Optimización**: Garantizar un rendimiento adecuado en hardware limitado, respetando restricciones de tamaño de modelos (menos de 100 MB por objeto).

El proyecto busca demostrar la aplicación práctica de conocimientos en gráficos por computadora, programación orientada a objetos y desarrollo de software, creando una experiencia inmersiva que refleje la estética de la referencia seleccionada.

Imágenes de Referencia y Objetos a Realizar en Cada Cuarto

Fachada seleccionada:
La fachada está inspirada en la casa de *Un Show Más*, con un diseño que incluye una chimenea, escaleras, ventanas variadas (cuadradas, rectangulares, ventanales), una pieza en L, un balcón, columnas, pasamanos, una puerta principal y una puerta de garaje. La ambientación exterior incluye césped, árboles, arbustos, un faro decorativo y una tipo skybox que simula un cielo, diseñados para integrarse coherentemente con la escena.

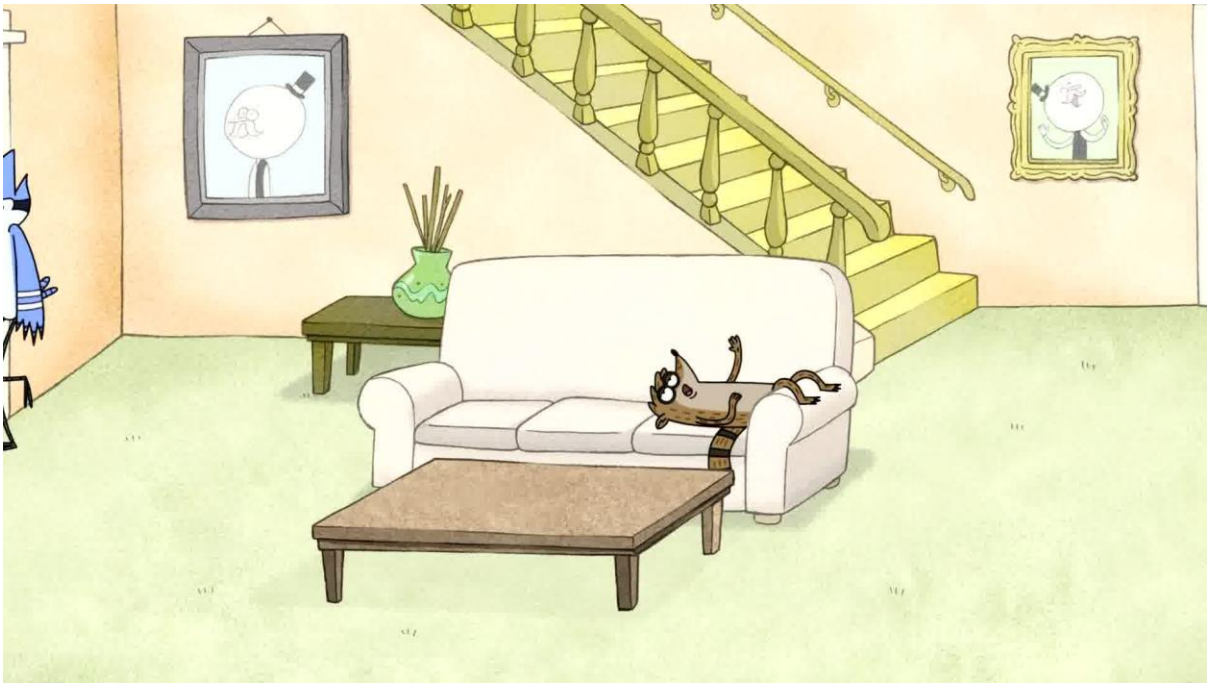
Primer cuarto seleccionado (cocina):
Desarrollada en equipo, la cocina recrea un espacio interior detallado con los siguientes objetos:

1. Refrigerador
2. Mesa
3. Silla
4. Estufa

5. Alacena

Segundo cuarto seleccionado (sala de estar):
Creada individualmente, la sala de estar incluye los siguientes objetos con animaciones específicas:

1. Alfombra
2. Televisor
3. Sofá
4. Mesas cuadradas
5. Cuadros



Alcance del Proyecto

El proyecto tiene como alcance la creación de una representación tridimensional interactiva de una casa inspirada en la serie *Un Show Más*, utilizando OpenGL y GLFW en C++. La implementación se centra en modelar con precisión visual una fachada exterior, una cocina desarrollada en equipo, y una sala de estar creada individualmente, junto con una ambientación adecuada que complemente la escena. El alcance específico incluye:

- **Modelado tridimensional**: Diseñar una fachada con elementos como chimenea, escaleras, ventanas variadas, balcón, columnas, pasamanos, puerta principal y puerta de garaje. Modelar una cocina con cinco objetos (refrigerador, mesa, silla, estufa, alacena) y una sala de estar con cinco objetos (carrito, televisor, sofá, mesa de centro, lámpara), todos creados en Maya Autodesk 2025 y optimizados para cumplir con el límite de tamaño de 100 MB por objeto.
- **Iluminación y sombras**: Implementar luces direccionales para transiciones día/noche, luces puntuales manipulables por el usuario, y sombras para mejorar el realismo de la escena.
- **Animaciones**: Incorporar cuatro animaciones: tres simples (movimiento del carrito, apertura de la puerta del garaje, apertura de la puerta principal) y una compleja (animación del televisor con efecto de ruido estático, sincronizada con transformaciones adicionales).
- **Interacción del usuario**: Desarrollar una interfaz básica que permita navegar el entorno mediante una cámara sintética controlada por teclado (W, A, S, D, Espacio, Shift) y ratón, manipular luces (teclas U, J, H, K, flechas) y alternar entre día/noche (tecla N) o encender/apagar luces (tecla L).
- **Ambientación**: Incluir una figura tipo skybox que simule un cielo dinámico, junto con objetos exteriores como césped, árboles, arbustos y un faro decorativo para enriquecer la escena.
- **Documentación**: Elaborar un manual técnico que detalle la metodología, diagramas (flujo, Gantt), y

explicación del código, y un manual de usuario que describa la interacción con el entorno.

- **Entrega**: Publicar el proyecto en un repositorio de GitHub (https://github.com/obeedt/319093166_ProyectoFinal_Grupo05), asegurando un historial de cambios claro y la integración correcta de modelos exportados desde Maya Autodesk 2025 a OpenGL, preservando texturas y materiales.

Limitantes

El desarrollo del proyecto enfrentó varias restricciones técnicas y académicas que definieron su alcance. Estas limitaciones incluyen:

- **Restricciones de espacios**: No se permitió la reproducción de espacios con restricciones legales, como instalaciones de la UNAM o edificios gubernamentales, lo que llevó a elegir una casa ficticia inspirada en *Un Show Más*.
- **Calidad gráfica**: Se evitó el uso de estilos gráficos retro de baja calidad (e.g., PS1, PS2), priorizando un estándar visual moderno y coherente con las capacidades de OpenGL.
- **Código base obligatorio**: El proyecto se desarrolló sobre una plantilla proporcionada por el profesor, limitando modificaciones estructurales para cumplir con los lineamientos del curso.
- **Tamaño de modelos**: Cada modelo debía ser menor a 100 MB para garantizar un rendimiento adecuado, lo que requirió optimización de geometrías y texturas en Maya Autodesk 2025.
- **Compatibilidad de modelos**: La exportación de modelos desde Maya a OpenGL presentó desafíos, como distorsiones en texturas, problemas con normales o errores de carga, que obligaron a realizar múltiples ajustes y reexportaciones.
- **Conocimiento inicial limitado**: El equipo comenzó sin experiencia previa en Maya Autodesk 2025, lo que implicó una curva de aprendizaje pronunciada para modelado, texturizado y exportación.
- **Restricciones académicas**: Las animaciones se limitaron a las técnicas cubiertas en el curso, enfocándose en transformaciones básicas y una animación ya más compleja para el televisor, debido a la falta de formación formal en animaciones avanzadas.

Diagrama de Gant

Documentación del Código

El código del proyecto, implementado en C++ utilizando OpenGL y GLFW, se encuentra principalmente en el archivo `main.cpp`, además que se usan otras librerías para que funcione correctamente. A continuación, se detalla su estructura y funcionamiento, explicando las secciones clave que permiten la creación del entorno 3D interactivo.

1. Bibliotecas

El código utiliza las siguientes bibliotecas esenciales para gráficos 3D, entrada y carga de recursos:

```
#include <iostream>
#include <cmath>

// GLEW
#include <GL/glew.h>

// GLFW
#include <GLFW/glfw3.h>

// Other Libs
#include "stb_image.h"

// GLM Mathematics
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>

//Load Models
#include "SOIL2/SOIL2.h"


// Other includes
#include "Shader.h"
#include "Camera.h"
#include "Model.h"
#include "Texture.h"
```

- GLEW (<GL/glew.h>): Carga extensiones de OpenGL para funcionalidades avanzadas.
- GLFW (<GLFW/glfw3.h>): Gestiona la ventana, teclado y ratón.
- GLM (<glm/glm.hpp>, <glm/gtc/matrix_transform.hpp>, <glm/gtc/type_ptr.hpp>): Maneja operaciones matemáticas para transformaciones 3D.
- STB Image (stb_image.h): Carga imágenes para texturas.

- SOIL2 (SOIL2/SOIL2.h): Facilita la carga de texturas.

- Clases personalizadas (Shader.h, Camera.h, Model.h, Texture.h): Gestionan shaders, cámara sintética, modelos 3D y texturas.
  - iostream y cmath: Soporte para entrada/salida y cálculos matemáticos.

2. Definición de Funciones

Se definen funciones clave para manejar entrada, movimiento y animaciones:

```
// Function prototypes
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode);
void MouseCallback(GLFWwindow* window, double xPos, double yPos);
void DoMovement();
void Animation(float deltaTime);
void Animation_garage(float deltaTime);
void AnimationCarrito(float deltaTime);
void updateNoiseTexture(); // Función que definiremos después
```

- void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode): Procesa eventos de teclado, como movimiento, alternancia día/noche (tecla N), encendido/apagado de luces (tecla L), y activación de animaciones (P, G, C, T).
- void MouseCallback(GLFWwindow* window, double xPos, double yPos): Ajusta la orientación de la cámara según el movimiento del ratón.
- void DoMovement(): Mueve la cámara y luces puntuales según teclas presionadas (W, A, S, D, Espacio, Shift, U, J, H, K, flechas).
  - void Animation(float deltaTime): Controla la animación de la puerta principal (apertura/cierre).
  - void Animation_garage(float deltaTime): Gestiona la animación de la puerta del garaje (arriba/abajo).
  - void AnimationCarrito(float deltaTime): Maneja la animación del carrito (movimiento, giro, regreso).
  - void updateNoiseTexture(): Genera texturas de ruido procedural para el efecto estático del televisor.

3. Definición de la Cámara

La cámara sintética se define con la clase Camera:

```
// Camera
Camera  camera(glm::vec3(0.0f, 0.0f, 135.0f));
```

  - Posición inicial: (0, 0, 135) para una vista exterior de la casa.
- Control: Movimiento con teclas (W, A, S, D, Espacio, Shift) y orientación con ratón, ajustada por DoMovement y MouseCallback. La variable multiplier (aumentada con Ctrl) acelera el movimiento.
- Variables auxiliares: lastX, lastY, y firstMouse rastrean la posición del ratón para una navegación suave.

4. Atributos de Luz y Animaciones

Atributos de luz:

```
// Light attributes
glm::vec3 lightPos(0.0f, 0.0f, 0.0f);
bool active;
bool isNight = false; // Para determinar si es de noche o d♦a
float transitionSpeed = 1.0f; // Velocidad de transici♦n de d♦a a noche
bool keyPressed = false; // Para almacenar el estado previo de la tecla N
bool lightsOff = false;
bool keyPressed2 = false; // Para almacenar el estado previo de la tecla L
```

- glm::vec3 lightPos(0.0f, 0.0f, 0.0f): Posición inicial de una luz (no usada directamente, se priorizan luces puntuales).
  - glm::vec3 lightColor(1.0f, 1.0f, 1.0f): Color blanco por defecto.
  - glm::vec3 pointLightPositions[]: Dos luces puntuales en (-14.0f, 12.5f, -11.0f) y (-17.0f, 17.0f, 80.0f).
  - bool isNight: Alterna entre día y noche (tecla N).
  - bool lightsOff: Enciende/apaga luces puntuales (tecla L).
  - float transitionSpeed = 1.0f: Controla la velocidad de transición día/noche.

Atributos de animaciones:

Puerta principal

```
//Animacion puerta
bool animPuerta = false;
bool abriendo = true;
float rotPuerta = 0.0f;
float pivotOffset = 0.5f;
float doorSpeed = 90.0f;
```

Puerta del garaje

```
//Animacion garage
bool animGarage = false;
float rotPuerta_Garage = 0.0f;
bool abriendoGarage = true;
float pivotOffsetGarage = 0.5f; // Ajusta según tu modelo
```

Carrito

```
//Animacion carrito
bool animCarrito = false;       // Activa/desactiva la animación
float posCarrito = 0.0f;        // Posición actual (0 a distanciaTotal)
const float distanciaTotal = 45.0f; // Distancia máxima de recorrido
const float velocidadCarrito = 11.0f; // Unidades por segundo
bool enPosicionFinal = false;   // Estado del carrito
bool girando = false;              // Controla si está en modo "giro"
float anguloGiro = 0.0f;           // Ángulo acumulado de rotación
const float velocidadGiro = 200.0f; // Grados/segundo de rotación (ajusta según necesidad)
const int vueltasRequeridas = 2;    // Número de vueltas completas antes de regresar
```

Televisión

```
//animacion tele
// Global variables
GLuint tvScreenTexture;      // Texture for static effect
GLuint tvOffTexture;         // Texture for black screen (off state)
GLuint tvNoiseTexture;       // Procedural noise texture
bool tvOn = false;
float tvNoiseIntensity = 0.5f;
float animationTime = 0.0f;  // Renamed to avoid ambiguity with std::time
```

5. Definición de la Pantalla

La ventana se crea con GLFW, configurada con dimensiones 800x600 y el título "Proyecto Final - Computación Gráfica"

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

6. Definición de los Shaders Utilizados

Se utilizan tres shaders para diferentes propósitos:

lightingShader : Aplica iluminación direccional y puntuales a los modelos, con soporte para texturas difusas y especulares.

lampShader : Renderiza luces puntuales como cubos pequeños con color uniforme.

tvShader : Genera el efecto de ruido estático en la pantalla del televisor, combinando texturas y ruido procedural.

```
Shader lightingShader("Shader/lighting.vs", "Shader/lighting.frag");
Shader lampShader("Shader/lamp.vs", "Shader/lamp.frag");
// Cargar shader especial para TV
Shader tvShader("shaders/tv.vs", "shaders/tv.frag");
```

7. Texturas Utilizadas para la Televisión

El televisor utiliza tres texturas para simular encendido/apagado y estática:

```
// Load textures
tvScreenTexture = TextureLoading::LoadTexture("Models/ruido.jpg");
tvOffTexture = TextureLoading::LoadTexture("Models/negro.jpg");
```

- tvScreenTexture: Cargada desde Models/ruido.jpg, representa el ruido estático.
- tvOffTexture: Cargada desde Models/negro.jpg, muestra una pantalla negra cuando el televisor está apagado.

8. Definición de los Modelos Utilizados

Los modelos 3D, creados en Maya Autodesk 2025, se cargan en el programa. Los modelos utilizados son:

```
//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");
Model Garage((char*)"Models/garage.obj");
```

```
// Modelos interior cocina
Model Puerta_Cocina((char*)"Models/puerta_cocina.obj");
Model Ventana_Cocina((char*)"Models/ventana_cocina.obj");
Model Pared_Cocina((char*)"Models/pared_cocina.obj");
Model Pared_Cocina_Anterior((char*)"Models/pared_cocina_anterior.obj");
Model Pared_Madera((char*)"Models/pared_madera.obj");
Model Piso_Cocina((char*)"Models/piso_cocina.obj");
Model Alacena((char*)"Models/alacena.obj");
Model Alacena_Superior((char*)"Models/alacena_superior.obj");
Model Alacena_Grande((char*)"Models/alacena_grande.obj");
Model Campana((char*)"Models/campana.obj");
Model Estufa((char*)"Models/estufa.obj");
Model Mesa((char*)"Models/mesa.obj");
Model Refrigerador((char*)"Models/refrigerador.obj");
Model Silla((char*)"Models/silla.obj");
Model Tostadora((char*)"Models/tostadora.obj");
```

```
Model Puerta((char*)"Models/puertaAnim.obj");
Model Puerta_Garage((char*)"Models/puerta_garage.obj");
Model Carrito((char*)"Models/carrito.obj");

// Modelo de la sala
Model Sala((char*)"Models/sala.obj");
Model Tele((char*)"Models/tele.obj");
Model PantallaTele((char*)"Models/pantalla.obj"); // Modelo plano para la pantalla
```

9. Implementación

El bucle principal while maneja el renderizado y la interacción. A continuación, se detallan los aspectos solicitados.

Implementación de las Luces: El sistema de iluminación combina una luz direccional (día/noche) y dos luces puntuales, configuradas en el shader lightingShader:

- Luz direccional (transición día/noche):

```
// Interpolaci◆n entre d◆a y noche
float timeOfDay = isNight ? 1.0f : 0.0f; // 1.0f = noche, 0.0f = d◆a
float lerpFactor = timeOfDay * transitionSpeed; // Factores de interpolaci◆n

// Lerp para la direcci◆n de la luz
glm::vec3 dayLightDir(-0.2f, -1.0f, -0.3f); // Direcci◆n de luz de d◆a
glm::vec3 nightLightDir(-0.2f, -1.0f, -0.3f); // Direcci◆n de luz de noche (podr◆as cambiarla si quieres)

// Lerp para la luz ambiental
glm::vec3 dayAmbient(0.5f, 0.5f, 0.5f); // Luz ambiental de d◆a
glm::vec3 nightAmbient(0.1f, 0.1f, 0.3f); // Luz ambiental de noche (m◆s azul)

// Lerp para la luz difusa
glm::vec3 dayDiffuse(0.8f, 0.8f, 0.8f); // Luz difusa de d◆a
glm::vec3 nightDiffuse(0.2f, 0.2f, 0.5f); // Luz difusa de noche

// Lerp para la luz especular
glm::vec3 daySpecular(0.5f, 0.5f, 0.5f); // Luz especular de d◆a
glm::vec3 nightSpecular(0.2f, 0.2f, 0.5f); // Luz especular de noche

// Lerp para la brillos del material
float dayShininess = 32.0f;
float nightShininess = 16.0f;
```

```
// Luz direccional
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"),
    glm::mix(dayLightDir, nightLightDir, lerpFactor).x,
    glm::mix(dayLightDir, nightLightDir, lerpFactor).y,
    glm::mix(dayLightDir, nightLightDir, lerpFactor).z);

glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"),
    glm::mix(dayAmbient, nightAmbient, lerpFactor).x,
    glm::mix(dayAmbient, nightAmbient, lerpFactor).y,
    glm::mix(dayAmbient, nightAmbient, lerpFactor).z);

glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"),
    glm::mix(dayDiffuse, nightDiffuse, lerpFactor).x,
    glm::mix(dayDiffuse, nightDiffuse, lerpFactor).y,
    glm::mix(dayDiffuse, nightDiffuse, lerpFactor).z);

glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"),
    glm::mix(daySpecular, nightSpecular, lerpFactor).x,
    glm::mix(daySpecular, nightSpecular, lerpFactor).y,
    glm::mix(daySpecular, nightSpecular, lerpFactor).z);

glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"),
    glm::mix(dayShininess, nightShininess, lerpFactor));
```

La tecla N alterna isNight, cambiando timeOfDay entre 0 (día) y 1 (noche). La función glm::mix interpola linealmente entre valores diurnos y nocturnos para dirección, luz ambiental, difusa, especular y brillo. La luz ambiental nocturna (nightAmbient) tiene un tono azulado, mientras que la difusa (nightDiffuse) reduce la intensidad para simular oscuridad.

- Luces puntuales:

```
// Configuraci◆n de la luz puntual 1
float lightsOffFactor = lightsOff ? 0.0f : 1.0f; // Factor de luz apagada
float lerpFactor2 = lightsOff * 1.0f; // Factor de interpolaci◆n

// Luz puntual 1
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"),
    pointLightPositions[0].x, pointLightPositions[0].y, pointLightPositions[0].z);

glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"),
    0.2f * lightsOffFactor, 0.2f * lightsOffFactor, 0.2f * lightsOffFactor);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"),
    1.0f * lightsOffFactor, 1.0f * lightsOffFactor, 1.0f * lightsOffFactor);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"),
    1.0f * lightsOffFactor, 1.0f * lightsOffFactor, 1.0f * lightsOffFactor);

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);
```

```
// Configuraci♦n de la luz puntual 2
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].position"),
    pointLightPositions[1].x, pointLightPositions[1].y, pointLightPositions[1].z);

// Color amarillento (m♦s intenso en el componente rojo y verde, menos azul)
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].ambient"),
    0.5f * lightsOffFactor, 0.5f * lightsOffFactor, 0.1f * lightsOffFactor);  // Amarillo suave ambiental

glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].diffuse"),
    3.0f * lightsOffFactor, 2.5f * lightsOffFactor, 1.0f * lightsOffFactor);  // Amarillo brillante difuso

glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].specular"),
    1.5f * lightsOffFactor, 1.2f * lightsOffFactor, 0.3f * lightsOffFactor);  // Amarillo especular

// Par♦metros de atenuaci♦n para mayor alcance (valores m♦s bajos = mayor alcance)
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].linear"), 0.05f);    // Reducido para mayor alcance
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].quadratic"), 0.01f); // Reducido para mayor alcance
```

La tecla L alterna lightsOff, aplicando lightsOffFactor (0 para apagado, 1 para encendido). La primera luz tiene un color blanco, mientras que la segunda tiene un tono amarillento (más rojo/verde). Los parámetros de atenuación (constant, linear, quadratic) controlan el alcance, siendo más amplio en la segunda luz.

Carga de Modelos: Los modelos se cargan en el bucle principal mediante la clase Model, aplicando transformaciones (traslación, rotación, escala) vía matrices (glm::mat4). Cada modelo se renderiza con Draw(lightingShader) o Draw(tvShader) para la pantalla del televisor, usando la matriz de modelo (modelLoc) para posicionarlos correctamente.

```
//Carga de modelos
view = camera.GetViewMatrix();

//////////////////////-Ambientacion-////////////////////////////
//Arbusto central
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(58.0f, -10.0f, 20.0f));
model = glm::scale(model, glm::vec3(18.0f, 20.0f, 18.0f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Arbusto.Draw(lightingShader);
//Arbusto izquierdo
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(45.0f, -10.0f, 22.0f));
model = glm::scale(model, glm::vec3(10.0f, 10.0f, 10.0f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Arbusto.Draw(lightingShader);
//Arbusto derecho
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(75.0f, -10.0f, 21.0f));
model = glm::scale(model, glm::vec3(10.0f, 10.0f, 10.0f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Arbusto.Draw(lightingShader);
//Arbusto derecho 2
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(85.0f, -10.0f, 21.0f));
model = glm::scale(model, glm::vec3(8.0f, 8.0f, 8.0f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Arbusto.Draw(lightingShader);

//Arbol derecha
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(90.0f, -10.0f, -30.0f));
model = glm::scale(model, glm::vec3(12.0f, 12.0f, 12.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Arbol.Draw(lightingShader);
//Arbol izquierda
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-90.0f, -10.0f, 30.0f));
model = glm::scale(model, glm::vec3(12.0f, 12.0f, 12.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Arbol.Draw(lightingShader);
```

```
//Arbol 1
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-80.0f, -10.0f, -30.0f));
model = glm::scale(model, glm::vec3(10.0f, 10.0f, 10.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Arbol.Draw(lightingShader);
//Arbol 2
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-150.0f, -10.0f, -70.0f));
model = glm::scale(model, glm::vec3(8.0f, 8.0f, 8.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Arbol.Draw(lightingShader);
//Arbol 2
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-230.0f, -10.0f, -70.0f));
model = glm::scale(model, glm::vec3(8.0f, 8.0f, 8.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Arbol.Draw(lightingShader);

//Faro
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-17.0f, -10.0f, 80.0f));
model = glm::scale(model, glm::vec3(0.4f, 0.4f, 0.4f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Farol.Draw(lightingShader);

//Cesped
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-17.0f, -13.0f, 80.0f));
model = glm::scale(model, glm::vec3(10.0f, 10.0f, 10.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Cesped.Draw(lightingShader);

//Cielo
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-17.0f, -14.0f, 80.0f));
model = glm::scale(model, glm::vec3(10.0f, 10.0f, 10.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Cielo.Draw(lightingShader);
```

```
////////////////////////-Fachada-////////////////////////////////
//Casa
model = glm::mat4(1);
model = glm::scale(model, glm::vec3(6.0f, 6.0f, 6.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Casa.Draw(lightingShader);
//Piso general (provisional)
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-3.0f, 5.55f, 3.0f));
model = glm::scale(model, glm::vec3(2.7f, 1.0f, 1.9f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Piso_Cocina.Draw(lightingShader);
//Techo
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(3.75f, 13.7f, 2.65f));
model = glm::scale(model, glm::vec3(2.55f, 1.0f, 1.75));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Piso_Cocina.Draw(lightingShader);

//Garage
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(9.4f, -4.35f, 23.0f)); // 3. Mover de vuelta
model = glm::scale(model, glm::vec3(7.0f, 7.0f, 7.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Garage.Draw(lightingShader);

////////////////////////-Sala-////////////////////////////////
//Cuarto completo de la sala
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-9.91f, 4.1f, 10.0f));
model = glm::scale(model, glm::vec3(1.0, 1.0f, 1.0f));
//model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Sala.Draw(lightingShader);
```

```
//////////////////////-Cocina-////////////////////////////////
//////////////////////-Estructura-////////////////////////////////
//Piso
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-10.2f, 4.1f, -12.6f));
model = glm::scale(model, glm::vec3(1.28, 1.0f, 0.84f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Piso_Cocina.Draw(lightingShader);
//Techo
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-10.2f, 13.0f, -12.7f));
model = glm::scale(model, glm::vec3(1.28, 1.0f, 0.84f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Piso_Cocina.Draw(lightingShader);
//Pared izquierda
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-15.57f, 3.3f, -9.6f));
model = glm::scale(model, glm::vec3(1.0f, 2.06f, 0.85f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Pared_Cocina.Draw(lightingShader);
//Pared derecha
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(12.0f, 3.3f, -9.6f));
model = glm::scale(model, glm::vec3(1.0f, 2.06f, 0.85f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Pared_Cocina.Draw(lightingShader);
//Pared posterior
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-10.2f, 3.3f, -32.3f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(1.0f, 2.06f, 1.28f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Pared_Cocina.Draw(lightingShader);
//Pared anterior
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-10.2f, 3.3f, -14.2f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(1.0f, 2.06f, 1.28f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Pared_Cocina_Anterior.Draw(lightingShader);
////Ventana
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-21.2f, 10.2f, -20.37f));
model = glm::scale(model, glm::vec3(3.35f, 3.35f, 1.0f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Ventana_Cocina.Draw(lightingShader);
```

```
//Puerta interior
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-4.8f, 7.2f, -20.15f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Puerta_Cocina.Draw(lightingShader);
//Puerta exterior
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-4.8f, 7.2f, -21.35f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Puerta_Cocina.Draw(lightingShader);
//Pared Madera 1
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-7.9f, 6.2f, -20.3f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Pared_Madera.Draw(lightingShader);
//Pared Madera 2
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.6f, 6.2f, -20.3f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Pared_Madera.Draw(lightingShader);
```

```
//////////////////////////-Objetos-////////////////////////////////
////Alacena 1
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-19.0f, 4.74f, -17.57f));
model = glm::scale(model, glm::vec3(1.2f, 1.0f, 1.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Alacena.Draw(lightingShader);
////Alacena 2
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-21.1f, 4.74f, -14.0f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Alacena.Draw(lightingShader);
////Alacena 3
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-19.0f, 4.74f, -10.5f));
model = glm::scale(model, glm::vec3(1.2f, 1.0f, 1.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Alacena.Draw(lightingShader);
////Alacena 4
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-17.1f, 4.74f, -14.0f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Alacena.Draw(lightingShader);
////Alacena 5
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-13.1f, 4.74f, -14.0f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Alacena.Draw(lightingShader);
//Alacena 6
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-15.0f, 11.5f, -19.1f));
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Alacena_Superior.Draw(lightingShader);
//Alacena 7
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-25.5f, 9.8f, -17.0f));
model = glm::scale(model, glm::vec3(1.2f, 1.2f, 1.2f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Alacena_Grande.Draw(lightingShader);
```

```
    ////Estufa
    model = glm::mat4(1);
    model = glm::translate(model, glm::vec3(-21.15f, 5.91f, -23.85f));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
    Estufa.Draw(lightingShader);
    ////Campana
    model = glm::mat4(1);
    model = glm::translate(model, glm::vec3(-19.2f, 6.1f, -23.4f));
    glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 200.0f);
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
    Campana.Draw(lightingShader);
    //Refrigerador
    model = glm::mat4(1);
    model = glm::translate(model, glm::vec3(-16.8f, 4.7f, -19.1f));
    model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0f));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
    Refrigerador.Draw(lightingShader);
    //Mesa
    model = glm::mat4(1);
    model = glm::translate(model, glm::vec3(-11.3, 6.82f, -11.75));
    model = glm::scale(model, glm::vec3(0.85f, 0.9f, 0.85f));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
    Mesa.Draw(lightingShader);
    //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);
    //Silla 2
    model = glm::mat4(1);
    model = glm::translate(model, glm::vec3(-24.3, 6.4f, -9.6f));
    model = glm::scale(model, glm::vec3(1.2f, 1.2f, 1.2f));
    model = glm::rotate(model, glm::radians(225.0f), glm::vec3(0.0f, 1.0f, 0.0f));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
    Silla.Draw(lightingShader);
    //Tostadora
    model = glm::mat4(1);
    model = glm::translate(model, glm::vec3(-11.5f, 4.74f, -16.5f));
    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));
    Tostadora.Draw(lightingShader);
```

```
//////////////////////-Animaciones-/////////////////////////////
// Puerta abriendo/cerrandose
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(pivotOffset + 2.0f, 10.0f, 17.0f)); // 3. Mover de vuelta
model = glm::scale(model, glm::vec3(3.3f, 3.3f, 3.3f));
model = glm::rotate(model, glm::radians(-rotPuerta), glm::vec3(0, 1, 0)); // 2. Rotar
model = glm::translate(model, glm::vec3(-pivotOffset, 0.0f, 0.0f)); // 1. Mover al origen
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Puerta.Draw(lightingShader);

//Puerta garage abriendo arriba/abajo
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(pivotOffsetGarage + 23.0f, 0.35f, 24.0f)); // 3. Mover de vuelta
model = glm::scale(model, glm::vec3(7.0f, 7.0f, 7.0f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::rotate(model, glm::radians(rotPuerta_Garage), glm::vec3(0, 1, 0)); // 2. Rotar
model = glm::translate(model, glm::vec3(-pivotOffsetGarage, 0.0f, 0.0f)); // 1. Mover al origen
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Puerta_Garage.Draw(lightingShader);

//Carrito golf moviendose adelante/atras
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(23.6f, -6.65f, 13.0f)); // Posición base
model = glm::translate(model, glm::vec3(0.0f, 0.0f, posCarrito));
if (girando) {
    model = glm::rotate(model, glm::radians(anguloGiro), glm::vec3(0.0f, 1.0f, 0.0f));
}
model = glm::scale(model, glm::vec3(1.9f, 1.9f, 1.9f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Carrito.Draw(lightingShader);
```

Shaders Utilizados para la Televisión: El shader tvShader renderiza la pantalla del televisor:

```
// Draw TV screen (with appropriate texture)
tvShader.Use();
modelLoc = glGetUniformLocation(tvShader.Program, "model");
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
tvShader.setMat4("view", view);
tvShader.setMat4("projection", projection);
tvShader.setFloat("time", animationTime);
tvShader.setFloat("noiseIntensity", tvNoiseIntensity);
tvShader.setBool("tvOn", tvOn);

// Bind textures for TV
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, tvOn ? tvScreenTexture : tvOffTexture);
tvShader.setInt("screenTexture", 0);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, tvNoiseTexture);
tvShader.setInt("noiseTexture", 1);
glActiveTexture(GL_TEXTURE2);
glBindTexture(GL_TEXTURE_2D, tvOffTexture);
tvShader.setInt("offTexture", 2);

// Draw TV screen
PantallaTele.Draw(tvShader);

// Unbind textures to prevent leakage
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, 0);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, 0);
glActiveTexture(GL_TEXTURE2);
glBindTexture(GL_TEXTURE_2D, 0);
```

El shader combina tvScreenTexture (ruido estático) cuando tvOn es true, o muestra tvOffTexture (pantalla negra) cuando está apagado. La variable animationTime anima el ruido, y noiseIntensity ajusta su visibilidad. Las texturas se unen a unidades específicas (GL_TEXTURE0, GL_TEXTURE1, GL_TEXTURE2) para el renderizado.

Animaciones: Se implementan cuatro animaciones, activadas por teclas:

- Puerta principal (tecla P): Rota la puerta hasta 90° para abrirla o 0° para cerrarla, usando un pivote (pivotOffset) para un movimiento realista. Se detiene al alcanzar los límites.

```
void Animation(float deltaTime) {
    if (!animPuerta) return;

    float rotation = doorSpeed * deltaTime;
    rotPuerta += abriendo ? rotation : -rotation;

    // Limitar rotación y alternar estado
    if (abriendo && rotPuerta >= 90.0f) {
        rotPuerta = 90.0f;
        abriendo = false;
        animPuerta = false; // Opcional: detener animación al llegar
    }
    else if (!abriendo && rotPuerta <= 0.0f) {
        rotPuerta = 0.0f;
        abriendo = true;
        animPuerta = false; // Opcional: detener animación al llegar
    }
}
```

Puerta del garaje (tecla G): Rota la puerta hacia arriba (90°) o abajo (0°), con un pivote ajustado para simular una apertura de garaje. Se detiene al completar el movimiento.

```
void Animation_garage(float deltaTime) {
    if (!animGarage) return;

    float rotation = doorSpeed * deltaTime;
    rotPuerta_Garage += abriendoGarage ? rotation : -rotation;

    // Limitar rotación y alternar estado
    if (abriendoGarage && rotPuerta_Garage >= 90.0f) {
        rotPuerta_Garage = 90.0f;
        abriendoGarage = false;
        animGarage = false;
    }
    else if (!abriendoGarage && rotPuerta_Garage <= 0.0f) {
        rotPuerta_Garage = 0.0f;
        abriendoGarage = true;
        animGarage = false;
    }
}
```

Carrito (tecla C): El carrito se mueve 45 unidades (distanciaTotal), gira dos vueltas completas (vueltasRequeridas) a 200°/segundo, y regresa al origen. La animación termina tras completar el ciclo.

```
void AnimationCarrito(float deltaTime) {
    if (!animCarrito) return;

    // Fase 1: Movimiento hacia el destino
    if (!enPosicionFinal && !girando) {
        posCarrito += velocidadCarrito * deltaTime;
        if (posCarrito >= distanciaTotal) {
            posCarrito = distanciaTotal;
            girando = true; // Activa la fase de giro
        }
    }
    // Fase 2: Giro en el destino
    else if (girando) {
        anguloGiro += velocidadGiro * deltaTime;

        // Comprueba si completó las vueltas
        if (anguloGiro >= 360.0f * vueltasRequeridas) {
            anguloGiro = 0.0f;
            girando = false;
            enPosicionFinal = true; // Prepara el regreso
        }
    }
    // Fase 3: Regreso al origen
    else if (enPosicionFinal) {
        posCarrito -= velocidadCarrito * deltaTime;
        if (posCarrito <= 0.0f) {
            posCarrito = 0.0f;
            enPosicionFinal = false;
            animCarrito = false; // Fin del ciclo
        }
    }
}
```

Televisor (tecla T): Alterna entre encendido (tvOn = true, muestra tvScreenTexture y apagado (tvOffTexture, pantalla negra). La variable animationTime anima el ruido, actualizado por updateNoiseTexture cada fotograma.

```
// Draw TV screen (with appropriate texture)
tvShader.Use();
modelLoc = glGetUniformLocation(tvShader.Program, "model");
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
tvShader.setMat4("view", view);
tvShader.setMat4("projection", projection);
tvShader.setFloat("time", animationTime);
tvShader.setFloat("noiseIntensity", tvNoiseIntensity);
tvShader.setBool("tvOn", tvOn);
```

Para esta última animaci{on fue necesario crearle su propio shader con sus archivos .vs y .frag para realizar la animación de prendido y apagado de la pantalla

tv.vs: Procesa las posiciones, normales y coordenadas de textura de los vértices, calcula la posición del fragmento en el espacio del mundo y pasa las coordenadas de textura al fragment shader. Se utiliza para transformar los vértices del modelo de la pantalla (pantalla.obj) y preparar datos (coordenadas UV, normales) para el fragment shader. Se configura con matrices de modelo, vista y proyección.

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoords;

out vec2 TexCoords;
out vec3 FragPos;
out vec3 Normal;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main() {
    TexCoords = aTexCoords;
    FragPos = vec3(model * vec4(aPos, 1.0));
    Normal = mat3(transpose(inverse(model))) * aNormal;
    gl_Position = projection * view * vec4(FragPos, 1.0);
}
```

tv.frag: Fragment shader para la pantalla del televisor. Combina texturas (estática, ruido procedural, pantalla apagada) para generar el efecto de encendido (ruido dinámico) o apagado (pantalla negra). Se aplica a la pantalla del televisor, usando tvScreenTexture, tvNoiseTexture y tvOffTexture. Variables como time, tvOn y noiseIntensity controlan la animación del ruido y el estado de encendido/apagado.

```
#version 330 core
out vec4 FragColor;

in vec2 TexCoords;
in vec3 FragPos;
in vec3 Normal;

uniform sampler2D screenTexture;
uniform sampler2D noiseTexture;
uniform sampler2D offTexture;
uniform float time;
uniform bool tvOn;
uniform float noiseIntensity;

void main() {
    // Flip Y coordinate to match OpenGL texture orientation
    vec2 adjustedUV = vec2(TexCoords.x, 1.0 - TexCoords.y);

    if (tvOn) {
        vec3 baseColor = texture(screenTexture, adjustedUV).rgb;
        vec2 noiseUV = mod(adjustedUV + vec2(time * 0.1), 1.0); // Clamp UVs to [0,1]
        vec3 noise = texture(noiseTexture, noiseUV).rgb;
        float scanLine = sin(adjustedUV.y * 100.0 + time * 5.0) * 0.05;
        vec3 finalColor = mix(baseColor, noise, noiseIntensity) + vec3(scanLine);
        FragColor = vec4(finalColor, 1.0);
    } else {
        FragColor = texture(offTexture, adjustedUV);
    }
}
```

Análisis de Costos

Este apartado presenta un desglose detallado de los costos asociados al desarrollo del proyecto, los costos consideran el hardware, software, mano de obra, consumo energético, depreciación del equipo y servicios de internet, con un total estimado basado en 160 horas de trabajo. El análisis incluye la recuperación de costos directos, valor agregado por calidad y funcionalidad, tiempo invertido, y un margen para riesgos y mejoras futuras.

Desglose de Costos

Concepto	Descripción	Cantidad	Costo Unitario (MXN)	Costo Total (MXN)
Costo de hardware				
Laptop Dell G3 15-3590	Equipo usado de 2020 para desarrollo	1 unidad	8,000	8,000
Monitor 24”	Pantalla para visualización	1 unidad	3,500	3,500
Mouse y Teclado	Periféricos básicos	1 conjunto	500	500
Subtotal Hardware				12,000
Costo de software				
Autodesk Maya 2025	Licencia educativa mensual	1 mes	1,200	1,200
Visual Studio Community	Gratuito	-	0	0
Bibliotecas (GLFW, GLM)	Gratuitas	-	0	0
Subtotal Software				1,200
Costo de mano de obra				

Concepto	Descripción	Cantidad	Costo Unitario (MXN)	Costo Total (MXN)
Desarrollo	Diseño, modelado, programación y pruebas	160 horas	120	19,200
Subtotal Mano de Obra				19,200
Otros costos				
Energía Eléctrica	Consumo estimado del equipo	20.8 kWh	3.5 por kWh	73
Internet	Plan mensual para desarrollo	1 mes	600	600
Depreciación Laptop	Valor anualizado (4,000 MXN/año ÷ 12)	1 mes	333	333
Subtotal Otros Costos				1,006
Total				33,406

Detalles de Cálculos

Hardware:

- **Laptop Dell G3 15-3590 (2020)**: Precio original de ~20,000 MXN en 2020. Valor estimado en 2025 tras depreciación (40% del valor original): 8,000 MXN. Depreciación anual: 20,000 MXN / 5 años = 4,000 MXN/año. Para 1 mes: 4,000 MXN / 12 = 333 MXN.
- **Monitor 24"**: Ajustado a 3,500 MXN (anteriormente ~3,000 MXN). ○

Mouse y Teclado: Ajustado a 500 MXN (anteriormente ~400 MXN). ○ **Total**

Hardware: 8,000 + 3,500 + 500 = 12,000 MXN.

Software:

- **Autodesk Maya 2025**: Licencia educativa mensual ajustada a 1,200 MXN (anteriormente ~1,000 MXN).
- **Visual Studio y Bibliotecas**: Gratuitas, sin costo adicional.

Mano de Obra:

- **Horas de Trabajo**: 160 horas estimadas para diseño, modelado en Maya, programación en

OpenGL, y pruebas.

- **Tarifa por Hora**: 120 MXN/hora (ajustada por inflación desde ~120 MXN/hora).
- **Total Mano de Obra**: 160 * 120 = 29,200 MXN.

Energía Eléctrica:

- **Consumo**: Laptop (100W) + Monitor (30W) = 130W. Total: 130W * 160 horas = 20,800 Wh = 20.8 kWh.
- **Costo por kWh**: 3.5 MXN (ajustado desde ~3 MXN).
- **Total Energía**: 20.8 * 3.5 = 72.8 MXN (~73 MXN).

Internet:

- **Costo Mensual**: 600 MXN (ajustado desde ~500 MXN) para un plan adecuado al desarrollo.

Depreciación:

- Calculada como 333 MXN para 1 mes de uso intensivo del laptop.

Costo Esperado del Proyecto

El costo total estimado del proyecto es de **33,406 MXN**, que cubre los gastos directos de hardware, software, mano de obra, energía, internet y depreciación. Este monto se fundamenta en:

Recuperación de Costos Directos: Incluye el hardware (12,000 MXN), software (1,200 MXN), energía (73 MXN), internet (600 MXN), y depreciación (333 MXN), sumando 14,206 MXN.

Mano de Obra: 19,200 MXN por 160 horas de trabajo especializado en modelado 3D, programación y animaciones.

Valor Agregado: Reflejado en la calidad visual (modelos detallados, texturas, iluminación), animaciones programadas (puerta, garaje, carrito, televisor), y funcionalidad interactiva (cámara sintética, controles de luz).

Tiempo Invertido: 160 horas representan una inversión significativa en aprendizaje de Autodesk Maya y OpenGL, y su aplicación práctica.

Margen para Riesgos y Mejoras: Incluye un margen implícito para cubrir riesgos inherentes al desarrollo, posibles costos de mantenimiento y futuras mejoras, así como para remunerar la especialización y dedicación del desarrollador.

Se propone un precio de venta de **40,000 MXN**, considerando el valor agregado por la experiencia inmersiva y profesional ofrecida al usuario final, superando los costos directos para garantizar viabilidad económica y profesionalidad del proyecto.

Conclusiones

El desarrollo de un entorno 3D interactivo inspirado en Un Show Más utilizando OpenGL y GLFW en C++ ha sido un proceso integral que abarcó desde la planificación hasta la implementación iterativa. El proyecto permitió la creación de una casa virtual detallada y funcional, con una cocina desarrollada en equipo, pero el cuarto de forma individual, donde se aplicaron los conocimientos adquiridos a lo largo del curso.

Este proyecto incluyó modelos 3D complejos, sistemas de iluminación dinámica (día/noche y luces puntuales), animaciones (puerta principal, puerta del garaje, carrito y televisor con ruido estático), y una cámara sintética para una interacción inmersiva. A pesar de enfrentar desafíos como el modelado detallado en Autodesk Maya, la integración de texturas, y la resolución de problemas en la programación de shaders y animaciones, se logró un resultado satisfactorio. Además, se encontraron ciertas dificultades al optimizar el rendimiento y alinear las texturas con los modelos.

Las futuras mejoras, como la incorporación de físicas realistas, optimización del rendimiento, o la adición de más interacciones, muestran el potencial del proyecto para seguir evolucionando. En resumen, el proyecto fue desafiante y enriquecedor de realizar. Esto se debe a que, en términos de aprendizaje, desarrollo técnico y creatividad, fue necesario demostrar el dominio de herramientas avanzadas y la capacidad de resolver problemas complejos de manera innovadora.

Technical Project Manual

ENGLISH

Introduction

This technical manual documents the final project developed for the laboratory of the subject "Computer Graphics and Human-Computer Interaction" at the Faculty of Engineering of the Universidad Nacional Autónoma de México, during the 2025-2 semester. The project, carried out as a team by Francisco Joshua Quintero Montero and Obed Torres Pimentel, consists of creating an interactive 3D environment inspired by the house from the series *Regular Show*, using OpenGL and GLFW in the C++ programming language.

El entorno recrea una casa con dos espacios principales: una **cocina**, desarrollada en equipo en el laboratorio de la materia, que incluye modelos tridimensionales, iluminación dinámica, sombras, una figura tipo skybox y una cámara sintética para navegación; y una **sala de estar**, creada individualmente por mí, Obed Torres Pimentel, con cinco objetos específicos y animaciones interactivas para un carrito, un televisor, la puerta del garaje y la puerta principal. El proyecto integra conceptos de modelado 3D, texturizado, iluminación, animaciones y optimización gráfica aprendidos en el curso.

Objectives

Develop an interactive tridimensional graphical application using OpenGL and GLFW in C++, with the following features:

- **Tridimensional models: Design and implement a house facade, a kitchen with multiple objects, and a living room with five specific objects, all modeled with precision using Maya Autodesk 2025 and optimized for OpenGL.**
- **Dynamic Lighting: Implement directional lights, point lights, shadows, and day/night transitions to improve the realism of the environment**.
- **Animations: Incorporate at least four animations, including 2 simple ones (garage door and main door) and 2 complex ones (television static noise animation and cart movement with rotation and translation).**
- **User Interaction: Enable navigation through a synthetic camera controlled by keyboard and mouse, and manipulation of lights and objects using specific keys.**

- **Ambiance: Include a figure that simulates a skybox to improve the look of the decoration, apart from adding objects to decorate the setting.**
- **Optimization: Ensure adequate performance on limited hardware, adhering to model size restrictions (less than 100 MB per object).**

The project aims to demonstrate the practical application of knowledge in computer graphics, object-oriented programming, and software development, creating an immersive experience that reflects the aesthetic of the chosen reference.

Reference Images and Objects for Each Room

Selected Facade:
The facade is inspired by the house from *Regular Show*, featuring a design with a chimney, stairs, varied windows (square, rectangular, large windows), an L-shaped section, a balcony, columns, handrails, a main door, and a garage door. The exterior ambiance includes grass, trees, bushes, a decorative streetlamp, and a skybox simulating a sky, designed to integrate coherently with the scene.

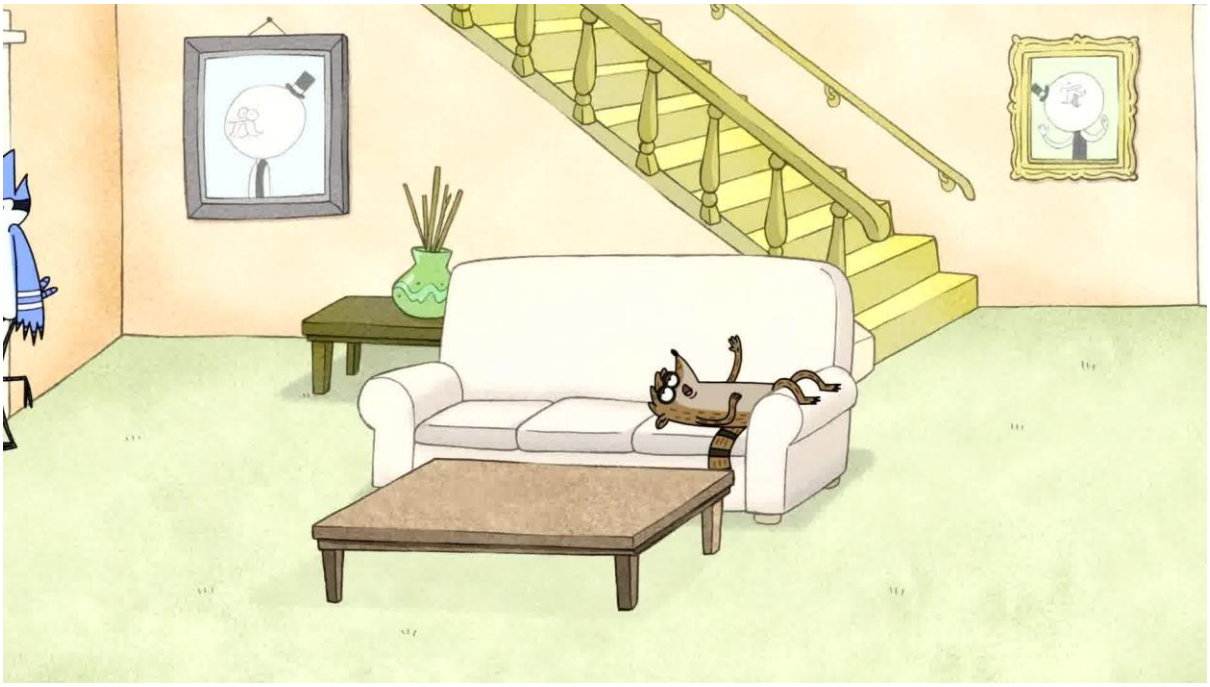
First Selected Room (Kitchen):
Developted collaboratively, the kitchen recreates a detailed interior space with the following objects:

6. Refrigerator
7. Table
8. Chair
9. Stove

10. Pantry

Second Selected Room (Living Room):
Created individually, the living room includes the following objects with specific animations:

6. Carpet
7. Television
8. Sofa
9. Square tables
10. Paintings



Project Scope

The project aims to create an interactive three-dimensional representation of a house inspired by the series *Regular Show*, using OpenGL and GLFW in C++. The implementation focuses on visually accurate modeling of an exterior facade, a collaboratively developed kitchen, and an individually created living room, along with appropriate ambiance to complement the scene. The specific scope includes:

- **Tridimensional model:** Design a facade with elements such as a chimney, stairs, varied windows, a balcony, columns, handrails, a main door, and a garage door. Model a kitchen with five objects (refrigerator, table, chair, stove, pantry) and a living room with five objects (cart, television, sofa, coffee table, lamp), all created in Maya Autodesk 2025 and optimized to meet the 100 MB size limit per object.
- **Lightning and shadows**: Implement directional lights for day/night transitions, user-manipulable point lights, and shadows to enhance the realism of the scene.
- **Animations**: Incorporate four animations: three simple ones (cart movement, garage door opening, main door opening) and a complex one (television with static noise effect, synchronized with additional transformations).
- **User Interaction:** Develop a basic interface allowing navigation through a synthetic camera controlled by keyboard (W, A, S, D, Space, Shift) and mouse, light manipulation (keys U, J, H, K, arrows), and alternate between day/night (N key) or turning lights on/off (L key).
- **Ambiance**: Include a skybox figure simulating a dynamic sky, along with exterior objects such as grass, trees, buches, and a decorative streetlamp to enrich the scene.
- **Documentation**: Create a technical manual detailing the methodology, diagrams (flowchart, Gantt), and code explanation, as well as a user manual describing interaction with the environment.

- **Delivery**: Publish the project on a GitHub repository. (https://github.com/obeedt/319093166_ProyectoFinal_Grupo05),), ensuring a clear change history and correct integration of models exported from Maya Autodesk 2025 to OpenGL, preserving textures and materials.

Limitations

The project development faced several technical and academic restrictions that shaped its scope. These limitations include

- **Space Restrictions:** Reproduction of spaces with legal restrictions were not allowed, such as UNAM facilities or government buildings, leading to the choice of a fictional house inspired by *Regular Show*.
- **Graphical Quality:** Use of low-quality retro graphical styles (e.g., PS1, PS2) was avoided, prioritizing a modern visual standard consistent with OpenGL capabilities.
- **Mandatory Code Base:** The project was developed using a template provided by the professor, limiting structural modifications to comply with course guidelines.
- **Model Size:** Each model had to be under 100 MB to ensure adequate performance, requiring optimization of geometries and textures in Maya Autodesk 2025.
- **Model Compatibility:** Exporting models from Maya to OpenGL presented challenges, such as texture distortions, normal issues or loading errors, needing multiple adjustments and re-exports.
- **Limited Initial Knowledge:** The team started with no prior experience in Maya Autodesk 2025, which meant a steep learning curve for modeling, texturing and exporting.
- **Academic Restrictions:** Animations were limited to techniques covered in the course, focusing on basic transformations and a more complex animation for the television, due to the lack of formal training in advanced animations.

Gant´s Diagram

Code Documentation

The projects code, implemented in C++ using OpenGL and GLFW, is primarily located in the main.cpp file, along with other libraries required for proper functionality. Below is a detailed explanation of its structure and operation, highlighting the key sections that enable the creation of the interactive 3D environment.

10. Libraries

The code utilizes the following essential libraries for 3D graphics, input handling, and resource loading:

```cpp
#include <iostream>
#include <cmath>

// GLEW
#include <GL/glew.h>

// GLFW
#include <GLFW/glfw3.h>

// Other Libs
#include "stb_image.h"

// GLM Mathematics
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>

//Load Models
#include "SOIL2/SOIL2.h"


// Other includes
#include "Shader.h"
#include "Camera.h"
#include "Model.h"
#include "Texture.h"
```

- GLEW (<GL/glew.h>): Load OpenGL extensions for advanced funcionalities.
- GLFW (<GLFW/glfw3.h>): Manages the window, keyboard, and mouse input.
- GLM (<glm/glm.hpp>, <glm/gtc/matrix_transform.hpp>, <glm/gtc/type_ptr.hpp>): Handles mathematical operations for 3D transformations.
- STB Image (stb_image.h): Loads images for textures.

- SOIL2 (SOIL2/SOIL2.h): Facilitates the loading of textures.

- Clases personalizadas (Shader.h, Camera.h, Model.h, Texture.h): Manages shaders, the synthetic camera, 3D models and textures
  - iostream y cmath: Provides support for input/output operations and mathematical calculations.

11. Function Definitions

Key functions are defined to handle input, movement, and animations:

```
// Function prototypes
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode);
void MouseCallback(GLFWwindow* window, double xPos, double yPos);
void DoMovement();
void Animation(float deltaTime);
void Animation_garage(float deltaTime);
void AnimationCarrito(float deltaTime);
void updateNoiseTexture(); // Función que definiremos después
```

- void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode): Processes keyboard events, such as movement, day/night toggling (N key), light on/off (L key), and animation activation (P, G, C, T keys).
- void MouseCallback(GLFWwindow* window, double xPos, double yPos): Adjusts the camera orientation based on mouse movement.
- void DoMovement(): Moves the camera and point lights based on pressed keys (W, A, S, D, Space, Shift, U, J, H, K, arrows).
  - void Animation(float deltaTime): Controls the animation of the main door (opening/closing)
  - void Animation_garage(float deltaTime): Manages the animation of the garage door (up/down).
  - void AnimationCarrito(float deltaTime): Handles the animation of the cart (movement, rotation, return).
  - void updateNoiseTexture(): Generates procedural noise textures for the television's static effect.

12. Camera Definitions

The synthetic camera is defined with the Camera class:

```
// Camera
Camera  camera(glm::vec3(0.0f, 0.0f, 135.0f));
```

  - Initial Position: (0, 0, 135) for an exterior view of the house.
- Control: Movement with keys (W, A, S, D, Space, Shift) and orientation with the mouse, adjusted by DoMovement and MouseCallback. The multiplier variable (increased with Ctrl) accelerates movement.

- Auxiliary Variables: lastX, lastY, and firstMouse track the mouse position for smooth navigation.

13. Light and Animation Attributes

Light Attributes:

```
// Light attributes
glm::vec3 lightPos(0.0f, 0.0f, 0.0f);
bool active;
bool isNight = false; // Para determinar si es de noche o d�a
float transitionSpeed = 1.0f; // Velocidad de transici�n de d�a a noche
bool keyPressed = false; // Para almacenar el estado previo de la tecla N
bool lightsOff = false;
bool keyPressed2 = false; // Para almacenar el estado previo de la tecla L
```

- glm::vec3 lightPos(0.0f, 0.0f, 0.0f): Initial position of a light (not used directly, prioritizing point lights).
  - glm::vec3 lightColor(1.0f, 1.0f, 1.0f): Default white color.
  - glm::vec3 pointLightPositions[]: Two point lights at (-14.0f, 12.5f, -11.0f) and (-17.0f, 17.0f, 80.0f).
  - bool isNight: Alternates between day and night (N key).
  - bool lightsOff: Turns ppint light son/off (L key).
  - float transitionSpeed = 1.0f: Controls the day/night transition speed.

Animation Attributes:

Main Door

```
//Animacion puerta
bool animPuerta = false;
bool abriendo = true;
float rotPuerta = 0.0f;
float pivotOffset = 0.5f;
float doorSpeed = 90.0f;
```

Garage Door

```
//Animacion garage
bool animGarage = false;
float rotPuerta_Garage = 0.0f;
bool abriendoGarage = true;
float pivotOffsetGarage = 0.5f; // Ajusta según tu modelo
```

Cart

```
//Animacion carrito
bool animCarrito = false;       // Activa/desactiva la animación
float posCarrito = 0.0f;        // Posición actual (0 a distanciaTotal)
const float distanciaTotal = 45.0f; // Distancia máxima de recorrido
const float velocidadCarrito = 11.0f; // Unidades por segundo
bool enPosicionFinal = false;   // Estado del carrito
bool girando = false;              // Controla si está en modo "giro"
float anguloGiro = 0.0f;           // Ángulo acumulado de rotación
const float velocidadGiro = 200.0f; // Grados/segundo de rotación (ajusta según necesidad)
const int vueltasRequeridas = 2;    // Número de vueltas completas antes de regresar
```

Television

```
//animacion tele
// Global variables
GLuint tvScreenTexture;      // Texture for static effect
GLuint tvOffTexture;         // Texture for black screen (off state)
GLuint tvNoiseTexture;       // Procedural noise texture
bool tvOn = false;
float tvNoiseIntensity = 0.5f;
float animationTime = 0.0f;  // Renamed to avoid ambiguity with std::time
```

14. Screen Definition

The window is created with GLFW, configured with 800x600 dimensions and the title "Final Project - Computer Graphics."

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

15. Shaders Used Definitions

Three shaders are used for different purposes:

lightingShader: Applies directional and point lighting to models, supporting diffuse and specular textures

lampShader: Renders point lights as small cubes with uniform color.

tvShader: Generates the static noise effect on the television screen, combining textures and procedural noise.

```
Shader lightingShader("Shader/lighting.vs", "Shader/lighting.frag");
Shader lampShader("Shader/lamp.vs", "Shader/lamp.frag");
// Cargar shader especial para TV
Shader tvShader("shaders/tv.vs", "shaders/tv.frag");
```

16. Textures Used for the Television

The television uses three textures to simulate on/off and static:

```
// Load textures
tvScreenTexture = TextureLoading::LoadTexture("Models/ruido.jpg");
tvOffTexture = TextureLoading::LoadTexture("Models/negro.jpg");
```

- tvScreenTexture: Loaded form Models/ruido.jpg, represents static noise.
- tvOffTexture: Loaded from Models/negro.jpg, displays a black screen when the televisión is off.

17. Models Used Definition

3D models, created in Maya Autodesk 2025, are loaded into the program. The models used are:

```
//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");
Model Garage((char*)"Models/garage.obj");
```

```
// Modelos interior cocina
Model Puerta_Cocina((char*)"Models/puerta_cocina.obj");
Model Ventana_Cocina((char*)"Models/ventana_cocina.obj");
Model Pared_Cocina((char*)"Models/pared_cocina.obj");
Model Pared_Cocina_Anterior((char*)"Models/pared_cocina_anterior.obj");
Model Pared_Madera((char*)"Models/pared_madera.obj");
Model Piso_Cocina((char*)"Models/piso_cocina.obj");
Model Alacena((char*)"Models/alacena.obj");
Model Alacena_Superior((char*)"Models/alacena_superior.obj");
Model Alacena_Grande((char*)"Models/alacena_grande.obj");
Model Campana((char*)"Models/campana.obj");
Model Estufa((char*)"Models/estufa.obj");
Model Mesa((char*)"Models/mesa.obj");
Model Refrigerador((char*)"Models/refrigerador.obj");
Model Silla((char*)"Models/silla.obj");
Model Tostadora((char*)"Models/tostadora.obj");
```

```
Model Puerta((char*)"Models/puertaAnim.obj");
Model Puerta_Garage((char*)"Models/puerta_garage.obj");
Model Carrito((char*)"Models/carrito.obj");

// Modelo de la sala
Model Sala((char*)"Models/sala.obj");
Model Tele((char*)"Models/tele.obj");
Model PantallaTele((char*)"Models/pantalla.obj"); // Modelo plano para la pantalla
```

18. Implementation

The main while loop handles rendering and interaction. Below are the requested aspects in detail.

Lighting Implementation: The lighting system combines a directional light (day/night) and two point lights, configured in the shader lightingShader:

- Directional Light (Day/Night transition):

```
// Interpolaci♦n entre d♦a y noche
float timeOfDay = isNight ? 1.0f : 0.0f; // 1.0f = noche, 0.0f = d♦a
float lerpFactor = timeOfDay * transitionSpeed; // Factores de interpolaci♦n

// Lerp para la direcci♦n de la luz
glm::vec3 dayLightDir(-0.2f, -1.0f, -0.3f); // Direcci♦n de luz de d♦a
glm::vec3 nightLightDir(-0.2f, -1.0f, -0.3f); // Direcci♦n de luz de noche (podr♦as cambiarla si quieres)

// Lerp para la luz ambiental
glm::vec3 dayAmbient(0.5f, 0.5f, 0.5f); // Luz ambiental de d♦a
glm::vec3 nightAmbient(0.1f, 0.1f, 0.3f); // Luz ambiental de noche (m♦s azul)

// Lerp para la luz difusa
glm::vec3 dayDiffuse(0.8f, 0.8f, 0.8f); // Luz difusa de d♦a
glm::vec3 nightDiffuse(0.2f, 0.2f, 0.5f); // Luz difusa de noche

// Lerp para la luz especular
glm::vec3 daySpecular(0.5f, 0.5f, 0.5f); // Luz especular de d♦a
glm::vec3 nightSpecular(0.2f, 0.2f, 0.5f); // Luz especular de noche

// Lerp para la brillos del material
float dayShininess = 32.0f;
float nightShininess = 16.0f;
```

```
// Luz direccional
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"),
    glm::mix(dayLightDir, nightLightDir, lerpFactor).x,
    glm::mix(dayLightDir, nightLightDir, lerpFactor).y,
    glm::mix(dayLightDir, nightLightDir, lerpFactor).z);

glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"),
    glm::mix(dayAmbient, nightAmbient, lerpFactor).x,
    glm::mix(dayAmbient, nightAmbient, lerpFactor).y,
    glm::mix(dayAmbient, nightAmbient, lerpFactor).z);

glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"),
    glm::mix(dayDiffuse, nightDiffuse, lerpFactor).x,
    glm::mix(dayDiffuse, nightDiffuse, lerpFactor).y,
    glm::mix(dayDiffuse, nightDiffuse, lerpFactor).z);

glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"),
    glm::mix(daySpecular, nightSpecular, lerpFactor).x,
    glm::mix(daySpecular, nightSpecular, lerpFactor).y,
    glm::mix(daySpecular, nightSpecular, lerpFactor).z);

glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"),
    glm::mix(dayShininess, nightShininess, lerpFactor));
```

The N key toggles isNight, switching timeOfDay between 0 (day) and 1 (night). The glm::mix function linearly interpolates between day and night values for direction, ambient light, diffuse light, specular light, and brightness. The nighttime ambient light (nightAmbient) has a bluish tone, while the diffuse light (nightDiffuse) reduces intensity to simulate darkness.

- Point Lights:

```
// Configuraci◆n de la luz puntual 1
float lightsOffFactor = lightsOff ? 0.0f : 1.0f; // Factor de luz apagada
float lerpFactor2 = lightsOff * 1.0f; // Factor de interpolaci◆n

// Luz puntual 1
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"),
    pointLightPositions[0].x, pointLightPositions[0].y, pointLightPositions[0].z);

glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"),
    0.2f * lightsOffFactor, 0.2f * lightsOffFactor, 0.2f * lightsOffFactor);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"),
    1.0f * lightsOffFactor, 1.0f * lightsOffFactor, 1.0f * lightsOffFactor);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"),
    1.0f * lightsOffFactor, 1.0f * lightsOffFactor, 1.0f * lightsOffFactor);

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);
```

```
// Configuraci?n de la luz puntual 2
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].position"),
    pointLightPositions[1].x, pointLightPositions[1].y, pointLightPositions[1].z);

// Color amarillento (m?s intenso en el componente rojo y verde, menos azul)
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].ambient"),
    0.5f * lightsOffFactor, 0.5f * lightsOffFactor, 0.1f * lightsOffFactor);  // Amarillo suave ambiental

glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].diffuse"),
    3.0f * lightsOffFactor, 2.5f * lightsOffFactor, 1.0f * lightsOffFactor);  // Amarillo brillante difuso

glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].specular"),
    1.5f * lightsOffFactor, 1.2f * lightsOffFactor, 0.3f * lightsOffFactor);  // Amarillo especular

// Par?metros de atenuaci?n para mayor alcance (valores m?s bajos = mayor alcance)
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].linear"), 0.05f);    // Reducido para mayor alcance
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].quadratic"), 0.01f); // Reducido para mayor alcance
```

The L key toggles lightsOff, applying lightsOffFactor (0 for off, 1 for on). The first light has a white color, while the second has a yellowish tone (more red/green). Attenuation parameters (constant, linear, quadratic) control the range, with the second light having a wider reach.

Models Loading: Models are loaded in the main loop using the Model class, applying transformations (translation, rotation, scaling) via matrices (glm::mat4). Each model is rendered with Draw(lightingShader) or Draw(tvShader) for the television screen, using the model matrix (modelLoc) for correct positioning.

```
//Carga de modelos
view = camera.GetViewMatrix();

//////////////////////-Ambientacion-//////////////////////////////
//Arbusto central
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(58.0f, -10.0f, 20.0f));
model = glm::scale(model, glm::vec3(18.0f, 20.0f, 18.0f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Arbusto.Draw(lightingShader);
//Arbusto izquierdo
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(45.0f, -10.0f, 22.0f));
model = glm::scale(model, glm::vec3(10.0f, 10.0f, 10.0f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Arbusto.Draw(lightingShader);
//Arbusto derecho
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(75.0f, -10.0f, 21.0f));
model = glm::scale(model, glm::vec3(10.0f, 10.0f, 10.0f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Arbusto.Draw(lightingShader);
//Arbusto derecho 2
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(85.0f, -10.0f, 21.0f));
model = glm::scale(model, glm::vec3(8.0f, 8.0f, 8.0f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Arbusto.Draw(lightingShader);

//Arbol derecha
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(90.0f, -10.0f, -30.0f));
model = glm::scale(model, glm::vec3(12.0f, 12.0f, 12.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Arbol.Draw(lightingShader);
//Arbol izquierda
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-90.0f, -10.0f, 30.0f));
model = glm::scale(model, glm::vec3(12.0f, 12.0f, 12.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Arbol.Draw(lightingShader);
```

```
//Arbol 1
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-80.0f, -10.0f, -30.0f));
model = glm::scale(model, glm::vec3(10.0f, 10.0f, 10.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Arbol.Draw(lightingShader);
//Arbol 2
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-150.0f, -10.0f, -70.0f));
model = glm::scale(model, glm::vec3(8.0f, 8.0f, 8.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Arbol.Draw(lightingShader);
//Arbol 2
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-230.0f, -10.0f, -70.0f));
model = glm::scale(model, glm::vec3(8.0f, 8.0f, 8.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Arbol.Draw(lightingShader);

//Faro
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-17.0f, -10.0f, 80.0f));
model = glm::scale(model, glm::vec3(0.4f, 0.4f, 0.4f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Farol.Draw(lightingShader);

//Cesped
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-17.0f, -13.0f, 80.0f));
model = glm::scale(model, glm::vec3(10.0f, 10.0f, 10.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Cesped.Draw(lightingShader);

//Cielo
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-17.0f, -14.0f, 80.0f));
model = glm::scale(model, glm::vec3(10.0f, 10.0f, 10.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Cielo.Draw(lightingShader);
```

```
////////////////////////-Fachada-////////////////////////////////
//Casa
model = glm::mat4(1);
model = glm::scale(model, glm::vec3(6.0f, 6.0f, 6.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Casa.Draw(lightingShader);
//Piso general (provisional)
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-3.0f, 5.55f, 3.0f));
model = glm::scale(model, glm::vec3(2.7f, 1.0f, 1.9f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Piso_Cocina.Draw(lightingShader);
//Techo
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(3.75f, 13.7f, 2.65f));
model = glm::scale(model, glm::vec3(2.55f, 1.0f, 1.75));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Piso_Cocina.Draw(lightingShader);

//Garage
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(9.4f, -4.35f, 23.0f)); // 3. Mover de vuelta
model = glm::scale(model, glm::vec3(7.0f, 7.0f, 7.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Garage.Draw(lightingShader);

////////////////////////-Sala-////////////////////////////////
//Cuarto completo de la sala
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-9.91f, 4.1f, 10.0f));
model = glm::scale(model, glm::vec3(1.0, 1.0f, 1.0f));
//model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Sala.Draw(lightingShader);
```

```
//////////////////////-Cocina-////////////////////////////////
//////////////////////-Estructura-////////////////////////////////
//Piso
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-10.2f, 4.1f, -12.6f));
model = glm::scale(model, glm::vec3(1.28, 1.0f, 0.84f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Piso_Cocina.Draw(lightingShader);
//Techo
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-10.2f, 13.0f, -12.7f));
model = glm::scale(model, glm::vec3(1.28, 1.0f, 0.84f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Piso_Cocina.Draw(lightingShader);
//Pared izquierda
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-15.57f, 3.3f, -9.6f));
model = glm::scale(model, glm::vec3(1.0f, 2.06f, 0.85f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Pared_Cocina.Draw(lightingShader);
//Pared derecha
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(12.0f, 3.3f, -9.6f));
model = glm::scale(model, glm::vec3(1.0f, 2.06f, 0.85f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Pared_Cocina.Draw(lightingShader);
//Pared posterior
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-10.2f, 3.3f, -32.3f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(1.0f, 2.06f, 1.28f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Pared_Cocina.Draw(lightingShader);
//Pared anterior
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-10.2f, 3.3f, -14.2f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(1.0f, 2.06f, 1.28f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Pared_Cocina_Anterior.Draw(lightingShader);
////Ventana
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-21.2f, 10.2f, -20.37f));
model = glm::scale(model, glm::vec3(3.35f, 3.35f, 1.0f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Ventana_Cocina.Draw(lightingShader);
```

```
//Puerta interior
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-4.8f, 7.2f, -20.15f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Puerta_Cocina.Draw(lightingShader);
//Puerta exterior
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-4.8f, 7.2f, -21.35f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Puerta_Cocina.Draw(lightingShader);
//Pared Madera 1
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-7.9f, 6.2f, -20.3f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Pared_Madera.Draw(lightingShader);
//Pared Madera 2
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.6f, 6.2f, -20.3f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Pared_Madera.Draw(lightingShader);
```

```
//////////////////////////-Objetos-////////////////////////////////
////Alacena 1
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-19.0f, 4.74f, -17.57f));
model = glm::scale(model, glm::vec3(1.2f, 1.0f, 1.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Alacena.Draw(lightingShader);
////Alacena 2
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-21.1f, 4.74f, -14.0f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Alacena.Draw(lightingShader);
////Alacena 3
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-19.0f, 4.74f, -10.5f));
model = glm::scale(model, glm::vec3(1.2f, 1.0f, 1.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Alacena.Draw(lightingShader);
////Alacena 4
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-17.1f, 4.74f, -14.0f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Alacena.Draw(lightingShader);
////Alacena 5
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-13.1f, 4.74f, -14.0f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Alacena.Draw(lightingShader);
//Alacena 6
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-15.0f, 11.5f, -19.1f));
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Alacena_Superior.Draw(lightingShader);
//Alacena 7
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-25.5f, 9.8f, -17.0f));
model = glm::scale(model, glm::vec3(1.2f, 1.2f, 1.2f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Alacena_Grande.Draw(lightingShader);
```

```
    ////Estufa
    model = glm::mat4(1);
    model = glm::translate(model, glm::vec3(-21.15f, 5.91f, -23.85f));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
    Estufa.Draw(lightingShader);
    ////Campana
    model = glm::mat4(1);
    model = glm::translate(model, glm::vec3(-19.2f, 6.1f, -23.4f));
    glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 200.0f);
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
    Campana.Draw(lightingShader);
    //Refrigerador
    model = glm::mat4(1);
    model = glm::translate(model, glm::vec3(-16.8f, 4.7f, -19.1f));
    model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0f));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
    Refrigerador.Draw(lightingShader);
    //Mesa
    model = glm::mat4(1);
    model = glm::translate(model, glm::vec3(-11.3, 6.82f, -11.75));
    model = glm::scale(model, glm::vec3(0.85f, 0.9f, 0.85f));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
    Mesa.Draw(lightingShader);
    //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);
    //Silla 2
    model = glm::mat4(1);
    model = glm::translate(model, glm::vec3(-24.3, 6.4f, -9.6f));
    model = glm::scale(model, glm::vec3(1.2f, 1.2f, 1.2f));
    model = glm::rotate(model, glm::radians(225.0f), glm::vec3(0.0f, 1.0f, 0.0f));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
    Silla.Draw(lightingShader);
    //Tostadora
    model = glm::mat4(1);
    model = glm::translate(model, glm::vec3(-11.5f, 4.74f, -16.5f));
    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));
    Tostadora.Draw(lightingShader);
```

```
//////////////////////-Animaciones-/////////////////////////////
// Puerta abriendo/cerrandose
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(pivotOffset + 2.0f, 10.0f, 17.0f)); // 3. Mover de vuelta
model = glm::scale(model, glm::vec3(3.3f, 3.3f, 3.3f));
model = glm::rotate(model, glm::radians(-rotPuerta), glm::vec3(0, 1, 0)); // 2. Rotar
model = glm::translate(model, glm::vec3(-pivotOffset, 0.0f, 0.0f)); // 1. Mover al origen
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Puerta.Draw(lightingShader);

//Puerta garage abriendo arriba/abajo
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(pivotOffsetGarage + 23.0f, 0.35f, 24.0f)); // 3. Mover de vuelta
model = glm::scale(model, glm::vec3(7.0f, 7.0f, 7.0f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::rotate(model, glm::radians(rotPuerta_Garage), glm::vec3(0, 1, 0)); // 2. Rotar
model = glm::translate(model, glm::vec3(-pivotOffsetGarage, 0.0f, 0.0f)); // 1. Mover al origen
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Puerta_Garage.Draw(lightingShader);

//Carrito golf moviendose adelante/atras
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(23.6f, -6.65f, 13.0f)); // Posición base
model = glm::translate(model, glm::vec3(0.0f, 0.0f, posCarrito));
if (girando) {
    model = glm::rotate(model, glm::radians(anguloGiro), glm::vec3(0.0f, 1.0f, 0.0f));
}
model = glm::scale(model, glm::vec3(1.9f, 1.9f, 1.9f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Carrito.Draw(lightingShader);
```

Shaders Used for the Television: The tvShader renders the television screen:

```
// Draw TV screen (with appropriate texture)
tvShader.Use();
modelLoc = glGetUniformLocation(tvShader.Program, "model");
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
tvShader.setMat4("view", view);
tvShader.setMat4("projection", projection);
tvShader.setFloat("time", animationTime);
tvShader.setFloat("noiseIntensity", tvNoiseIntensity);
tvShader.setBool("tvOn", tvOn);

// Bind textures for TV
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, tvOn ? tvScreenTexture : tvOffTexture);
tvShader.setInt("screenTexture", 0);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, tvNoiseTexture);
tvShader.setInt("noiseTexture", 1);
glActiveTexture(GL_TEXTURE2);
glBindTexture(GL_TEXTURE_2D, tvOffTexture);
tvShader.setInt("offTexture", 2);

// Draw TV screen
PantallaTele.Draw(tvShader);

// Unbind textures to prevent leakage
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, 0);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, 0);
glActiveTexture(GL_TEXTURE2);
glBindTexture(GL_TEXTURE_2D, 0);
```

The shader combines tvScreenTexture (static noise) when tvOn is true, or displays tvOffTexture (black screen) when its off. The animationTime variable animates the noise, and noiseIntensity adjusts its visibility. Textures are bound to specific units (GL_TEXTURE0, GL_TEXTURE1, GL_TEXTURE2) for rendering.

Animations: Four animations are implemented, activated by keys:

- Main door (P key): Rotates the door to 90° to open or 0° to close, using a pivot (pivotOffset) for realistic movement. It stops at the limits.

```
void Animation(float deltaTime) {
    if (!animPuerta) return;

    float rotation = doorSpeed * deltaTime;
    rotPuerta += abriendo ? rotation : -rotation;

    // Limitar rotación y alternar estado
    if (abriendo && rotPuerta >= 90.0f) {
        rotPuerta = 90.0f;
        abriendo = false;
        animPuerta = false; // Opcional: detener animación al llegar
    }
    else if (!abriendo && rotPuerta <= 0.0f) {
        rotPuerta = 0.0f;
        abriendo = true;
        animPuerta = false; // Opcional: detener animación al llegar
    }
}
```

Garage Door (G key): Rotates the door upward (90°) or downward (0°), with an adjusted pivot to simulate garage opening. It stops upon completion.

```
void Animation_garage(float deltaTime) {
    if (!animGarage) return;

    float rotation = doorSpeed * deltaTime;
    rotPuerta_Garage += abriendoGarage ? rotation : -rotation;

    // Limitar rotación y alternar estado
    if (abriendoGarage && rotPuerta_Garage >= 90.0f) {
        rotPuerta_Garage = 90.0f;
        abriendoGarage = false;
        animGarage = false;
    }
    else if (!abriendoGarage && rotPuerta_Garage <= 0.0f) {
        rotPuerta_Garage = 0.0f;
        abriendoGarage = true;
        animGarage = false;
    }
}
```

Cart (C key): The cart moves 45 units (distanciaTotal), performs two full rotations (vueltasRequeridas) at 200°/second, and returns to the origin. The animation ends after completing the cycle.

```
void AnimationCarrito(float deltaTime) {
    if (!animCarrito) return;

    // Fase 1: Movimiento hacia el destino
    if (!enPosicionFinal && !girando) {
        posCarrito += velocidadCarrito * deltaTime;
        if (posCarrito >= distanciaTotal) {
            posCarrito = distanciaTotal;
            girando = true; // Activa la fase de giro
        }
    }
    // Fase 2: Giro en el destino
    else if (girando) {
        anguloGiro += velocidadGiro * deltaTime;

        // Comprueba si completó las vueltas
        if (anguloGiro >= 360.0f * vueltasRequeridas) {
            anguloGiro = 0.0f;
            girando = false;
            enPosicionFinal = true; // Prepara el regreso
        }
    }
    // Fase 3: Regreso al origen
    else if (enPosicionFinal) {
        posCarrito -= velocidadCarrito * deltaTime;
        if (posCarrito <= 0.0f) {
            posCarrito = 0.0f;
            enPosicionFinal = false;
            animCarrito = false; // Fin del ciclo
        }
    }
}
```

Television (T key): Toggles between on (tvOn = true, displays tvScreenTexture) and off (tvOffTexture, black screen). The animationTime variable animates the noise, updated by updateNoiseTexture each frame.

```
// Draw TV screen (with appropriate texture)
tvShader.Use();
modelLoc = glGetUniformLocation(tvShader.Program, "model");
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
tvShader.setMat4("view", view);
tvShader.setMat4("projection", projection);
tvShader.setFloat("time", animationTime);
tvShader.setFloat("noiseIntensity", tvNoiseIntensity);
tvShader.setBool("tvOn", tvOn);
```

For this last animation, it was neccesary a dedicated shader created with its own .vs and .frag files to handle the on/off screen animation:

tv.vs: Processes vertex positions, normal and texture coordinated, calculates the fragment position in world space, and passes texture coordinates to the fragment shader. Used to transform the vertices of the screen model (pantalla.obj) and prepare data (UV coordinates, normals) for the fragment shader. Configured with model, view, and projection matrices.

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoords;

out vec2 TexCoords;
out vec3 FragPos;
out vec3 Normal;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main() {
    TexCoords = aTexCoords;
    FragPos = vec3(model * vec4(aPos, 1.0));
    Normal = mat3(transpose(inverse(model))) * aNormal;
    gl_Position = projection * view * vec4(FragPos, 1.0);
}
```

tv.frag: Fragment shader for the television screen. Combines textures (static, procedural noise, off screen) to generate the on (dynamic noise) or off (black screen) effect. Applied to the television screen, using tvScreenTexture, tvNoiseTexture, and tvOffTexture. Variables like time, tvOn, and noiseIntensity control the noise animation and on/off state.

```
#version 330 core
out vec4 FragColor;

in vec2 TexCoords;
in vec3 FragPos;
in vec3 Normal;

uniform sampler2D screenTexture;
uniform sampler2D noiseTexture;
uniform sampler2D offTexture;
uniform float time;
uniform bool tvOn;
uniform float noiseIntensity;

void main() {
    // Flip Y coordinate to match OpenGL texture orientation
    vec2 adjustedUV = vec2(TexCoords.x, 1.0 - TexCoords.y);

    if (tvOn) {
        vec3 baseColor = texture(screenTexture, adjustedUV).rgb;
        vec2 noiseUV = mod(adjustedUV + vec2(time * 0.1), 1.0); // Clamp UVs to [0,1]
        vec3 noise = texture(noiseTexture, noiseUV).rgb;
        float scanLine = sin(adjustedUV.y * 100.0 + time * 5.0) * 0.05;
        vec3 finalColor = mix(baseColor, noise, noiseIntensity) + vec3(scanLine);
        FragColor = vec4(finalColor, 1.0);
    } else {
        FragColor = texture(offTexture, adjustedUV);
    }
}
```

Cost Analysis

This section provides a detailed breakdown of the costs associated with the development of the project. The costs include hardware, software, labor, energy consumption, equipment depreciation, and internet services, with an estimated total based on 160 hours of work. The analysis considers the recovery of direct costs, added value from quality and functionality, time invested, and a margin for risks and future improvements.

Cost Breakdown

Concept	Description	Quantity	Unit Cost (MXN)	Total Cost (MXN)
Hardware Cost				
Dell G3 15-3590 Laptop	Used 2020 equipment for development	1 unit	8,000	8,000
24" Monitor	Display for visualization	1 unit	3,500	3,500
Mouse and Keyboard	Basic peripherals	1 set	500	500
Hardware Subtotal				12,000
Software Cost				
Autodesk Maya 2025	Monthly educational license	1 month	1,200	1,200
Visual Studio Community	Free	-	0	0
Libraries (GLFW, GLM)	Free	-	0	0
Software Subtotal				1,200
Labor Cost				
Development	Design, modeling, programming, and testing	160 hours	120	19,200
Labor Subtotal				19,200
Other Costs				

Concept	Description	Quantity	Unit Cost (MXN)	Total Cost (MXN)
Electricity	Estimated equipment consumption	20.8 kWh	3.5 per kWh	73
Internet	Monthly plan for development	1 month	600	600
Laptop Depreciation	Annualized value (4,000 MXN/year ÷ 12)	1 month	333	333
Other Costs Subtotal				1,006
Total				33,406

Calculation Details

- **Hardware**:
  - **Dell G3 15-3590 Laptop (2020)**: Original price of ~20,000 MXN in 2020. Estimated value in 2025 after depreciation (40% of original value): 8,000 MXN. Annual depreciation: 20,000 MXN / 5 years = 4,000 MXN/year. For 1 month: 4,000 MXN / 12 = 333 MXN.
  - **24" Monitor**: Adjusted to 3,500 MXN (previously ~3,000 MXN).
  - **Mouse and Keyboard**: Adjusted to 500 MXN (previously ~400 MXN).
  - **Total Hardware**: 8,000 + 3,500 + 500 = 12,000 MXN.
- **Software**:
  - **Autodesk Maya 2025**: Monthly educational license adjusted to 1,200 MXN (previously ~1,000 MXN).
  - **Visual Studio and Libraries**: Free, no additional cost.
- **Labor**:
  - **Work Hours**: 160 hours estimated for design, modeling in Maya, OpenGL programming, and testing.
  - **Hourly Rate**: 120 MXN/hour (adjusted for inflation from ~120 MXN/hour).
  - **Total Labor**: 160 * 120 = 19,200 MXN.
- **Electricity**:
  - **Consumption**: Laptop (100W) + Monitor (30W) = 130W. Total: 130W * 160 hours = 20,800 Wh = 20.8 kWh.

- **Cost per kWh**: 3.5 MXN (adjusted from ~3 MXN).
- **Total Electricity**: 20.8 * 3.5 = 72.8 MXN (~73 MXN).
- **Internet**:
- **Monthly Cost**: 600 MXN (adjusted from ~500 MXN) for a development-suitable plan.
- **Depreciation**:
- Calculated as 333 MXN for 1 month of intensive laptop use.

Expected Project Cost

The total estimated project cost is **33,406 MXN**, covering direct expenses for hardware, software, labor, electricity, internet, and depreciation. This amount is based on:

- **Direct Cost Recovery**: Includes hardware (12,000 MXN), software (1,200 MXN), electricity (73 MXN), internet (600 MXN), and depreciation (333 MXN), totaling 14,206 MXN.
- **Labor**: 19,200 MXN for 160 hours of specialized work in 3D modeling, programming, and animations.
- **Added Value**: Reflected in visual quality (detailed models, textures, lighting), programmed animations (door, garage, cart, TV), and interactive functionality (synthetic camera, light controls).
- **Time Invested**: 160 hours represent a significant investment in learning Autodesk Maya and OpenGL and their practical application.
- **Margin for Risks and Improvements**: Includes an implicit margin to cover inherent development risks, potential maintenance costs, future enhancements, and to compensate for the developer's expertise and dedication.

A sale price of **40,000 MXN** is proposed, considering the added value of the immersive and professional experience offered to the end user, exceeding direct costs to ensure economic viability and project professionalism.

Conclusions

The development of an interactive 3D environment inspired by *Regular Show* using OpenGL and GLFW in C++ has been a comprehensive process that spanned from planning to iterative implementation. The project enabled the creation of a detailed and functional virtual house, with a collaboratively developed kitchen and an individually crafted living room, where the knowledge acquired throughout the course was applied.

This project included complex 3D models, dynamic lighting systems (day/night and point lights), animations (main door, garage door, cart, and TV with static noise), and a synthetic camera for immersive interaction. Despite facing challenges such as detailed modeling in Autodesk Maya, texture integration, and resolving issues in shader and animation programming, a satisfactory result was achieved. Additionally, certain difficulties were encountered in optimizing performance and aligning textures with models.

Future improvements, such as incorporating realistic physics, optimizing performance, or adding more interactions, demonstrate the project's potential for further evolution. In summary, the project was challenging and enriching to undertake. This is because, in terms of learning, technical development, and creativity, it required demonstrating mastery of advanced tools and the ability to solve complex problems innovatively.