319146853

Manual Técnico

Introducción

Este manual técnico presenta la documentación detallada del proyecto final del laboratorio de la asignatura “Computación Gráfica e Interacción Humano Computadora”.

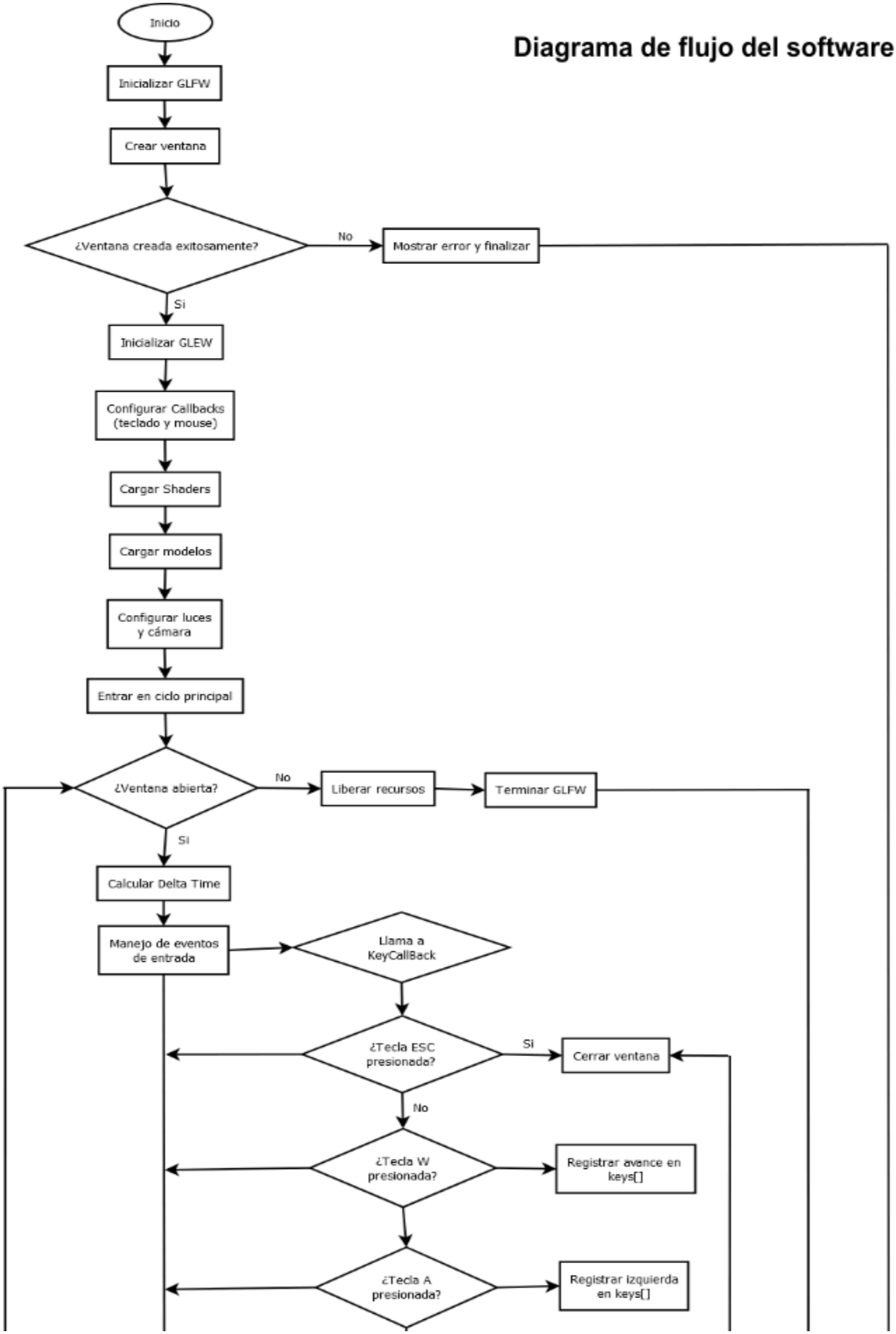
El proyecto consiste en la recreación de un entorno 3D interactivo, el cual debe de incluir modelos, animaciones, y demás conceptos abordados en las distintas sesiones de laboratorio de dicha materia.

Este documento aborda algunos elementos del proyecto como lo son los objetivos, el alcance, las limitaciones, la metodología aplicada, el flujo de software, un cronograma de actividades y una versión concisa del código realizado.

Objetivo

Elaborar una aplicación gráfica interactiva en tres dimensiones utilizando las tecnologías OpenGL y GLFW en el lenguaje de programación C++. La aplicación debe permitir la visualización de una escena compleja y detallada, que incluya modelos tridimensionales, iluminación avanzada y animaciones controladas por el usuario. Además, el alumno debe aplicar y demostrar de manera práctica todos los conocimientos teóricos y técnicos adquiridos a lo largo del curso, integrando conceptos de gráficos por computadora, programación orientada a objetos y desarrollo de software.

Diagrama de flujo del software



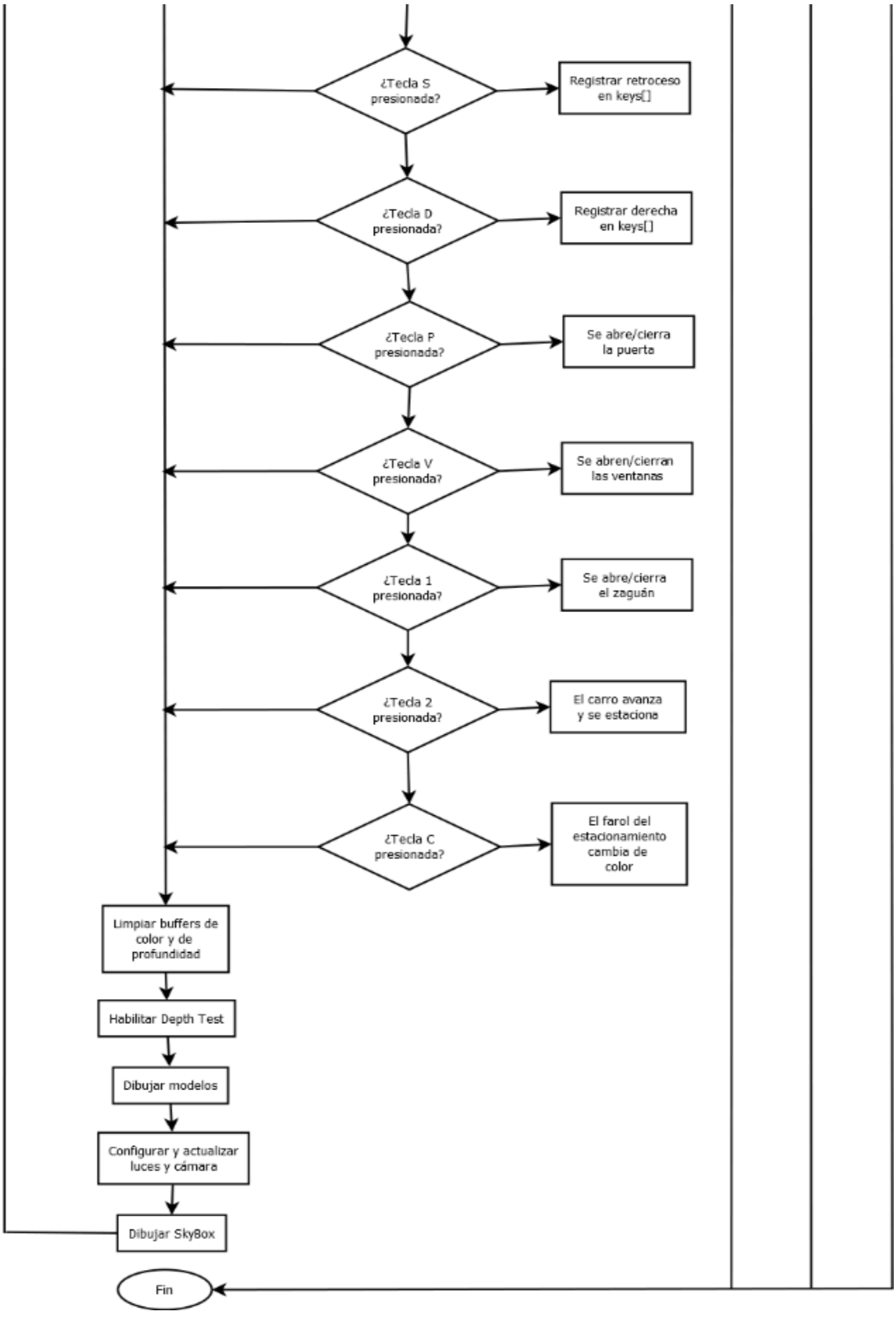
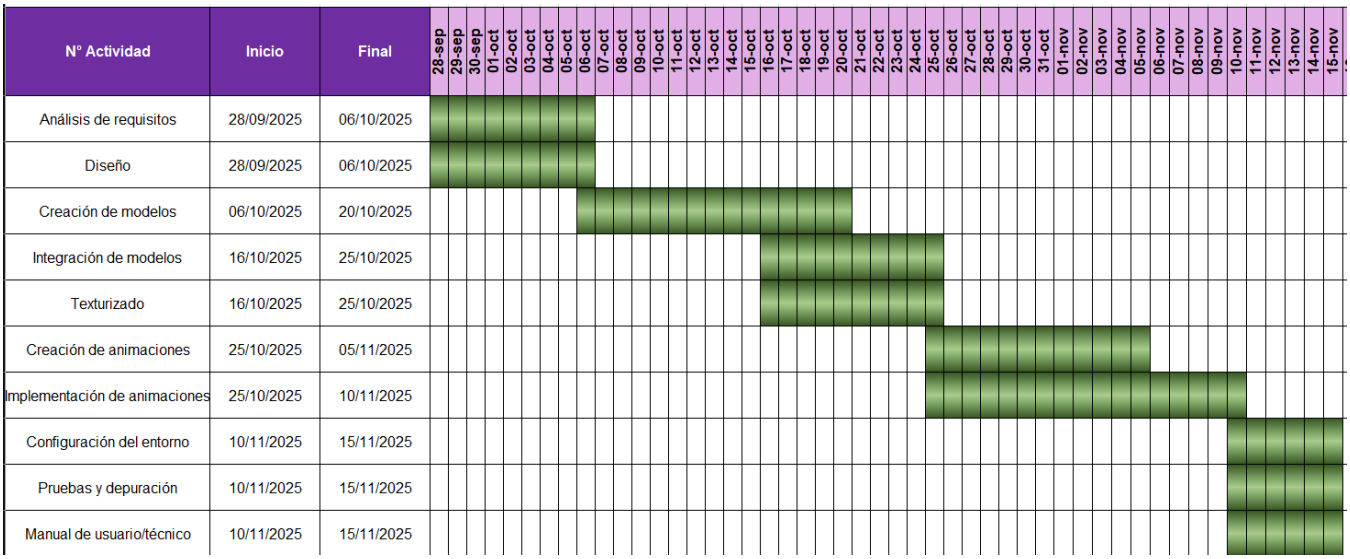

¿Tecla S presionada?
Registrar retroceso en keys[]
¿Tecla D presionada?
Registrar derecha en keys[]
¿Tecla P presionada?
Se abre/cierra la puerta
¿Tecla V presionada?
Se abren/cierran las ventanas
¿Tecla 1 presionada?
Se abre/cierra el zaguán
¿Tecla 2 presionada?
El carro avanza y se estaciona
¿Tecla C presionada?
El farol del estacionamiento cambia de color
Limpiar buffers de color y de profundidad
Habilitar Depth Test
Dibujar modelos
Configurar y actualizar luces y cámara
Dibujar SkyBox
Fin

Diagrama de Gantt



Nº Actividad	Inicio	Final
Análisis de requisitos	28/09/2025	06/10/2025
Diseño	28/09/2025	06/10/2025
Creación de modelos	06/10/2025	20/10/2025
Integración de modelos	16/10/2025	25/10/2025
Texturizado	16/10/2025	25/10/2025
Creación de animaciones	25/10/2025	05/11/2025
Implementación de animaciones	25/10/2025	10/11/2025
Configuración del entorno	10/11/2025	15/11/2025
Pruebas y depuración	10/11/2025	15/11/2025
Manual de usuario/técnico	10/11/2025	15/11/2025

Alcance del proyecto

El proyecto desarrollado se centra en la creación de un entorno 3D interactivo que incluye los siguientes elementos:

- Modelos tridimensionales: Esta sección abarca tanto modelos 3D creados por el propio estudiante, como aquellos generados con la asistencia de herramientas de inteligencia artificial presentadas por el profesor.
- Iluminación: Implementación de puntos de luz (pointlights) para asegurar una iluminación adecuada dentro del entorno virtual.
- Animaciones: Incorporación de animaciones, como la apertura de puertas y ventanas.
- Interacción del usuario: El usuario tendrá la capacidad de interactuar de manera inmersiva con el entorno virtual.
- Integración de Herramientas de Inteligencia Artificial: El proyecto incluirá la utilización de herramientas de IA para asistir en la creación y optimización de modelos 3D. Esto permitirá a los estudiantes explorar cómo la IA puede mejorar la eficiencia y calidad de los gráficos en aplicaciones tridimensionales.

Limitantes

- ➔ Rendimiento del Hardware: La sofisticación de los modelos tridimensionales y los efectos de iluminación implementados exigen un equipo con capacidad gráfica y de procesamiento elevada. En sistemas con hardware limitado, se pueden experimentar tasas de cuadros por segundo (FPS) bajas, lo cual deteriora la experiencia interactiva del usuario.
- ➔ Optimización de Contenidos: Debido a las restricciones de tiempo y recursos, ciertos modelos y animaciones no fueron completamente optimizados. Esto puede resultar en un uso excesivo de recursos, impactando negativamente el rendimiento en dispositivos menos potentes.
- ➔ Interacción del Usuario: La efectividad de la interacción depende de la precisión en el manejo de la cámara y los controles de teclado. Usuarios sin experiencia en aplicaciones 3D pueden encontrar estos controles complicados.
- ➔ Compatibilidad Multiplataforma: El desarrollo y las pruebas del proyecto se realizaron principalmente en un sistema operativo específico. Aunque OpenGL y GLFW son tecnologías multiplataforma, es posible que el proyecto presente problemas de rendimiento o errores en otros sistemas operativos sin ajustes adicionales.
- ➔ Gestión de Texturas y Materiales: Manejar las texturas y materiales de los modelos 3D puede ser complejo. La calidad y resolución de las texturas afectan directamente al rendimiento y la apariencia visual del entorno. Texturas de alta resolución pueden consumir muchos recursos, mientras que texturas de baja resolución pueden no ofrecer el nivel de detalle deseado.
- ➔ Compatibilidad de Drivers y Software: La compatibilidad de los controladores gráficos y las versiones de software puede presentar problemas. Las diferentes versiones de OpenGL y GLFW, así como los controladores de la tarjeta gráfica, pueden tener comportamientos distintos o presentar bugs que afecten la ejecución del proyecto. Asegurar que el entorno funcione de manera consistente en varias configuraciones de hardware y software puede ser un desafío significativo.

Metodología de software

Para la realización de este proyecto, se empleó una metodología de desarrollo de software iterativa e incremental, abarcando las siguientes fases:

1. **Análisis de Requerimientos:** En esta fase inicial, se llevó a cabo la identificación y recopilación detallada de las funcionalidades clave y los requisitos específicos solicitados por el usuario. Esta etapa es crucial para establecer una base sólida y comprender plenamente las expectativas y necesidades del proyecto.
2. **Diseño:** Una vez definidos los requerimientos, se procedió a la estructuración detallada del código y a la creación de los modelos necesarios. Esta fase incluye la elaboración de diagramas de diseño y la planificación de la arquitectura del software, asegurando que todos los componentes del sistema estén bien integrados y funcionen de manera coherente.
3. **Implementación:** Durante esta etapa, se desarrolló el código fuente del proyecto, integrando los modelos y bibliotecas necesarias. Se llevaron a cabo las tareas de codificación según el diseño preestablecido, asegurando que cada funcionalidad se implemente correctamente y de acuerdo a los estándares de calidad definidos.
4. **Pruebas:** Una vez implementadas las funcionalidades, se procedió a la fase de pruebas. Aquí se verificó minuciosamente el funcionamiento del sistema, identificando y corrigiendo errores puntuales. Las pruebas abarcaron tanto aspectos funcionales como no funcionales, garantizando que el proyecto cumpla con los requisitos especificados y funcione de manera eficiente.
5. **Documentación:** Simultáneamente, se elaboró la documentación necesaria, incluyendo tanto el manual técnico como el manual de usuario. El manual técnico proporciona detalles sobre la arquitectura, diseño y código del sistema, mientras que el manual de usuario ofrece instrucciones claras y concisas sobre cómo interactuar con la aplicación.
6. **Iteración:** Finalmente, se llevaron a cabo revisiones periódicas y se repitieron oportunamente las fases anteriores para añadir mejoras y refinamientos. Esta etapa asegura que el proyecto evolucione de manera continua, incorporando feedback y ajustándose a cualquier cambio en los requerimientos o nuevas necesidades que surjan.

Documentación del código

En este apartado encontramos las cosas nuevas que se implementaron en la parte del código. Desde variables nuevas que se crearon para poder llevar un control de las animaciones que van a tener los objetos, como también si es que se crearon nuevos shaders o encabezados de tipo .h.

```
// Window dimensions
const GLuint WIDTH = 1200, HEIGHT = 1000;
int SCREEN_WIDTH, SCREEN_HEIGHT;

// Camera
Camera  camera(glm::vec3(0.0f, 0.0f, 3.0f));
GLfloat lastX = WIDTH / 2.0;
GLfloat lastY = HEIGHT / 2.0;
bool keys[1024];
bool firstMouse = true;
```

Parámetros de ventana y creación de cámara en tercera persona con control de posición, vista y proyección.

Variables de inicio - Se declararon nuevas variables para poder controlar el movimiento de las animaciones simples, se pueden identificar por ser de tipo flotante o de tipo booleana en el caso de las que necesitan regresar a una posición inicial.

```
// Variables para controlar la animación
bool lightsOn = false; // Indica si las luces están encendidas
float animationTime = 0.0f; // Temporizador para controlar la animación
int currentStage = 0; // Estado de la animación
float tiempoColor = glfwGetTime();
float r = 0.8f;
float g = abs(sin(tiempoColor * 5.0f)) * 0.6f + 0.4f;
float b = 0.6f;
bool ventanaDesplazada = false;  // Indica si las ventanas están desplazadas
float ventanaPosZ = 0.0f;  // Posición inicial de las ventanas en el eje z

// KeyFrame de la Puerta de la casa
// Variables para Animacion de puerta con keyframes
float puertaInicio = 0.0f;                  // Angulo cerrado
float puertaFin = glm::radians(90.0f);      // Angulo abierto
float puertaActual = 0.0f;                  // Angulo interpolado
float puertaT = 0.0f;                       // Progreso 0 → 1
bool puertaAnimando = false;
bool puertaEstaAbierta = false;

// Animacion de la Puerta de la entrada (Estacionamiento)
bool animPuerta = false;        // Para activar/desactivar la animación
float rotPuerta = 0.0f;         // Ángulo de apertura de las puertas (en grados)
bool puertaMuseoAbierta = false; //estado actual (abierta/cerrada)

// Keyframes
// Posición inicial global de la escena (offset general)
glm::vec3 PosIni(0.0f, 0.0f, 0.0f);

// Variables ligadas a ese offset
float posX = PosIni.x;
float posY = PosIni.y;
float posZ = PosIni.z;

// --- Coche ---
bool animCoche = false;      // para prender/apagar la animación
bool sentidoCoche = false;   // false = va recto, true = ya dio vuelta
bool faseFinalCoche = false; // nueva bandera para la 3ra fase
float movCocheZ = 0.0f;      // avance hacia adelante/atrás
float movCocheX = 0.0f;      // avance lateral después de la curva
float rotCoche = 0.0f;      // rotación en Y (giro del coche)

// Variables para el kaliz y fuego
float giro = 0;
float tiempoFuego;
float velocidad = 5.0f;
```

```
// ---------- MESA + SILLAS TERRAZA ----------
glm::vec3 terrazaPos = glm::vec3(20.0f, 0.0f, -9.0f); // Posición base del set
glm::vec3 terrazaScale = glm::vec3(1.0f);             // Escala global (1 = normal)
float terrazaRotY = 0.0f;                          // Rotación global en Y (grados)

// ---------- CÁMARA DE SEGURIDAD ----------
glm::vec3 camSegPos = glm::vec3(20.0f, 3.5f, 18.3f); // posición base en el mundo
glm::vec3 camSegScale = glm::vec3(1.0f);             // escala global (1 = tamaño normal)
float camSegBaseRotY = 0.0f;                        // rotación fija del soporte (en Y)

// Paneo automático de la cabeza de la cámara
float camSegPanAngle = 0.0f;   // ángulo actual de paneo
float camSegPanSpeed = 25.0f;  // velocidad en grados por segundo
float camSegPanLimit = 45.0f;  // límite +/- del paneo
bool  camSegPanForward = true;   // true = girando hacia +, false = hacia -

// ---------- ARBUSTO LOW POLY ----------
glm::vec3 arbustoPos = glm::vec3(23.0f, 0.0f, 3.0f);  // posición en el mundo
glm::vec3 arbustoScale = glm::vec3(1.0f);               // escala global
float     arbustoRotY = 0.0f;                          // rotación en Y (grados)


// ---------- VENTILADOR DE TECHO ----------
glm::vec3 fanPos = glm::vec3(-5.0f, 3.2f, -8.5f);
glm::vec3 fanScale = glm::vec3(0.5f);             // escala global
float     fanRotY = 0.0f;                        // rotación fija en Y (orientación del ventilador)
float     fanBladesAngle = 0.0f;                  // ángulo actual de las aspas
float     fanBladesSpeed = 180.0f;                // velocidad de giro (grados por segundo)


// ---------- RELOJ DE PARED ----------
glm::vec3 clockPos = glm::vec3(-11.5f, 1.7f, -3.5f); // posición en la pared
glm::vec3 clockScale = glm::vec3(0.8f);               // escala global
float     clockRotY = 90.0f;

// ángulos de manecillas
float clockMinuteAngle = 19.0f;  // manecilla larga (segundero/minutero rápido)
float clockHourAngle = 8.0f;  // manecilla corta

// velocidades (grados por segundo)
float clockMinuteSpeed = 60.0f; // más rápido
float clockHourSpeed = 10.0f; // más lento
```

```
// Deltatime
GLfloat deltaTime = 0.0f;   // Time between current frame and last frame
GLfloat lastFrame = 0.0f;   // Time of last frame


//KEYFRAME PUERTA
struct KeyFrame {
    glm::vec3 position;
    glm::vec3 rotation;

    //Variables para GUARDAR Key Frames
    float posX;     //Variable para PosicionX
    float posY;     //Variable para PosicionY
    float posZ;     //Variable para PosicionZ
    float incX;     //Variable para IncrementoX
    float incY;     //Variable para IncrementoY
    float incZ;     //Variable para IncrementoZ
};

std::vector<KeyFrame> sillaKeyframes = {
    { glm::vec3(1.16f, -0.15f, -0.73f), glm::vec3(0.0f, 0.0f, 0.0f) },      // posicion inicial
    { glm::vec3(1.16f, -0.15f, -0.73f), glm::vec3(0.0f, 90.0f, 0.0f) },     // giro 90
    { glm::vec3(1.20f, -0.15f, -0.50f), glm::vec3(0.0f, 180.0f, 0.0f) }     // se mueve y se gira más
};

int currentKeyframe = 0;
float interp = 0.0f;
bool animarSilla = false;
```

Shaders y Modelos 3D

```
Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting.frag");
Shader lampShader("Shaders/lamp.vs", "Shaders/lamp.frag");
Shader tvNoiseShader("Shaders/tv.vs", "Shaders/tv.frag");
Shader SkyBoxshader("Shaders/SkyBox.vs", "Shaders/SkyBox.frag");
Shader colorShader("shaders/colorShader.vs", "shaders/colorShader.frag");
Shader AnimFuego("Shaders/AnimFuego.vs", "Shaders/AnimFuego.frag");

// MODELOS 3D
Model Mono((char*)"Models/oscar/mono.obj");
Model Kaliz((char*)"Models/oscar/kaliz.obj");
Model Fuego((char*)"Models/oscar/Fuego.obj");
Model Agua((char*)"Models/oscar/Agua.obj");
Model BastonSom((char*)"Models/oscar/bastonSom.obj");
Model CarroPared((char*)"Models/oscar/carroPared.obj");
Model LLantaDer((char*)"Models/oscar/LlantaDer.obj");
Model LlantaIzq((char*)"Models/oscar/LlantaIzq.obj");
Model Estructura((char*)"Models/Estructura/Estructura.obj");
Model Cuadros1((char*)"Models/Casa/cuadros1.obj");
Model Cuadros2((char*)"Models/Casa/cuadros2.obj");
Model casa((char*)"Models/Casa/casa.obj");

Model Cuadro1((char*)"Models/cuadros/cuadro1.obj");
Model Cuadros3((char*)"Models/cuadros/cuadros3.obj");
Model Barra((char*)"Models/barra/barra.obj");
Model Statua((char*)"Models/Statua/statua.obj");
Model Statua2((char*)"Models/Statua/statua2.obj");
Model Statua3((char*)"Models/Statua/statua3.obj");
Model Statua4((char*)"Models/Statua/statua4.obj");
Model Statua5((char*)"Models/Statua/statua5.obj");
Model Statua6((char*)"Models/Statua/statua6.obj");

Model tv((char*)"Models/tv_2/tv.obj");
Model barra((char*)"Models/barra/barra.obj");
Model puerta((char*)"Models/puerta/puerta.obj");
Model ventanas((char*)"Models/ventanas/ventanas.obj");
Model entrada((char*)"Models/entrada/entrada.obj");

Model PuertaDer((char*)"Models/puerta/PuertaDer.obj");
Model PuertaIzq((char*)"Models/puerta/PuertaIzq.obj");
Model Coches((char*)"Models/Coches/Coches.obj");
Model Coche((char*)"Models/Coches/Coche.obj");
Model Cuadros31((char*)"Models/cuadros/cuadros31.obj");
Model Cuadros32((char*)"Models/cuadros/cuadros32.obj");
Model Cuadros33((char*)"Models/cuadros/cuadros33.obj");
```

Carga de modelos 3D - Se decidió primero cargar los objetos que iban a permanecer de forma estática o que no requirieron del uso de algún shader diferente a lighting shader.

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

// Dobby
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Mono.Draw(lightingShader);

// Kaliz
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Kaliz.Draw(lightingShader);

// Baston Sombrero
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
BastonSom.Draw(lightingShader);

// Carro Pared
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
CarroPared.Draw(lightingShader);

// ====== LLANTA DERECHA ======
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
LLantaDer.Draw(lightingShader);

// ====== LLANTA IZQUIERDA ======
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
LlantaIzq.Draw(lightingShader);

// Cuadros Sala 1
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Cuadros1.Draw(lightingShader);

model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Cuadros2.Draw(lightingShader);
```

```
// Cuadros Sala 2
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Cuadro1.Draw(lightingShader);

model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Cuadros3.Draw(lightingShader);

// Barra
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Barra.Draw(lightingShader);

// Statua
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Statua.Draw(lightingShader);

// Statua 2
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Statua2.Draw(lightingShader);

// Statua 3
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Statua3.Draw(lightingShader);

// Statua 4
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Statua4.Draw(lightingShader);

// Statua 5
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Statua5.Draw(lightingShader);

// Statua 6
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Statua6.Draw(lightingShader);
```

```
// Cuadros Sala 3
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Cuadros31.Draw(lightingShader);

model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Cuadros32.Draw(lightingShader);

model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Cuadros33.Draw(lightingShader);

// ESTRUCTURA EXTERNA
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Estructura.Draw(lightingShader);

// EDIFICIO
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
casa.Draw(lightingShader);

// COCHES
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Coches.Draw(lightingShader);

//PUERTA DE LA ENTRADA DEL MUSEO (izquierda)
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(25.6f, 2.4f, 18.25f));
model = glm::scale(model, glm::vec3(1.0f, 1.2f, 0.8f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0, 1, 0));
model = glm::rotate(model, glm::radians(rotPuerta), glm::vec3(0.0f, 1.0f, 0.0f));

glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"),
    1.0f, 1.0f, 1.0f, 1.0f);

PuertaIzq.Draw(lightingShader);
```

```
//PUERTA DE LA ENTRADA DEL MUSEO (derecha)
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::translate(model, glm::vec3(25.6f, 2.4f, 13.55f));
model = glm::scale(model, glm::vec3(1.0f, 1.2f, 0.8f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0, 1, 0));
model = glm::rotate(model, glm::radians(-rotPuerta), glm::vec3(0.0f, 1.0f, 0.0f));

glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"),
    1.0f, 1.0f, 1.0f, 1.0f);
PuertaDer.Draw(lightingShader);


// PUERTA ENTRADA DEL MUSEO
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(25.6f, 0.0f, 2.3f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0, 1, 0));
model = glm::scale(model, glm::vec3(1.2f, 1.45f, 2.5f));
model = glm::rotate(model, puertaActual, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
puerta.Draw(lightingShader);


// COCHE
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(posX + movCocheX, posY, posZ + movCocheZ));
model = glm::translate(model, glm::vec3(28.0f, 0.85f, 15.5f));
model = glm::scale(model, glm::vec3(2.0f, 1.5f, 2.0f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0, 1, 0));
model = glm::rotate(model, glm::radians(rotCoche), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Coche.Draw(lightingShader);

// Ventanas
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, ventanaPosZ));  // Aplicar el desplazamiento en z
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
ventanas.Draw(lightingShader);

//Entrada
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
entrada.Draw(lightingShader);
```

```
//Puerta con KEYFRAME
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-7.1f, 0.35f, 3.6f));
model = glm::rotate(model, puertaActual, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
puerta.Draw(lightingShader);

model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-7.1f, 0.35f, 3.6f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0, 1, 0));
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -1.87f));
model = glm::rotate(model, puertaActual, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
puerta.Draw(lightingShader);

//barra
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
barra.Draw(lightingShader);
```

Posteriormente se decidió que vinieran al final los objetos que requerían de un shader diferente para crearse. En este caso solo contamos con 2 que requerían de esto. De igual forma aquí se decidió colocar el SkyBox.

```
//television ----usa Shader---
tvNoiseShader.Use();
glUniform1f(glGetUniformLocation(tvNoiseShader.Program, "time"), glfwGetTime());
// Configuración de matrices para la TV
model = glm::mat4(1);
glUniformMatrix4fv(glGetUniformLocation(tvNoiseShader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(glGetUniformLocation(tvNoiseShader.Program, "view"), 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(glGetUniformLocation(tvNoiseShader.Program, "projection"), 1, GL_FALSE, glm::value_ptr(projection));
// Dibuja la TV utilizando el shader de ruido
tv.Draw(tvNoiseShader);

lightingShader.Use();


// ===== AGUA en Kaliz =====
AnimFuego.Use();
tiempoFuego = glfwGetTime() * velocidad;

// uniforms de matrices
modelLoc = glGetUniformLocation(AnimFuego.Program, "model");
viewLoc = glGetUniformLocation(AnimFuego.Program, "view");
projLoc = glGetUniformLocation(AnimFuego.Program, "projection");
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));

// modelo base (sin transformaciones)
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));

glUniform1f(glGetUniformLocation(AnimFuego.Program, "time"), tiempoFuego);

// dibuja el modelo Agua con el shader AnimFuego
Agua.Draw(AnimFuego);
glBindVertexArray(0);
```

```
// ===== FUEGO EN EL CÁLIZ =====
AnimFuego.Use();
tiempoFuego = glfwGetTime() * velocidad;

// uniforms de matrices
modelLoc = glGetUniformLocation(AnimFuego.Program, "model");
viewLoc = glGetUniformLocation(AnimFuego.Program, "view");
projLoc = glGetUniformLocation(AnimFuego.Program, "projection");
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));

// MATRIZ DEL FUEGO
model = glm::mat4(1.0f);

// SOLO movimiento vertical (sube / baja alrededor de su posición actual)
model = glm::translate(model, glm::vec3(0.0f, 0.2f * sin(tiempoFuego), 0.0f));

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(AnimFuego.Program, "time"), tiempoFuego);

Fuego.Draw(AnimFuego);
glBindVertexArray(0);

// Dibujar el skybox al final de la escena
glDepthFunc(GL_LEQUAL);  // Cambiar función de profundidad para el skybox
SkyBoxshader.Use();
// Configurar vista y proyección para el skybox
view = glm::mat4(glm::mat3(camera.GetViewMatrix())); // Quitar traslación
glm::mat4 projection = glm::perspective(camera.GetZoom(), (GLfloat)SCREEN_WIDTH / (GLfloat)SCREEN_HEIGHT, 0.1f, 1000.0f);
glUniformMatrix4fv(glGetUniformLocation(SkyBoxshader.Program, "view"), 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(glGetUniformLocation(SkyBoxshader.Program, "projection"), 1, GL_FALSE, glm::value_ptr(projection));
// Vincular y dibujar el cubemap
glBindVertexArray(skyboxVAO);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_CUBE_MAP, cubemapTexture);
glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);
glDepthFunc(GL_LESS);  // Restablecer la función de profundidad
```

Objetos generados desde el código con primitivas:

```
//------------------Objetos generados con primitivas-----------------------------
    // ================== MESA + 4 SILLAS TERRAZA (CUBOS) ==================
    lightingShader.Use();

    // Aseguramos que los uniforms estén correctos
    modelLoc = glGetUniformLocation(lightingShader.Program, "model");
    viewLoc = glGetUniformLocation(lightingShader.Program, "view");
    projLoc = glGetUniformLocation(lightingShader.Program, "projection");

    glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
    glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));

    glBindVertexArray(VAO);

    // Matriz base del set (traslación + rotación + escala global)
    glm::mat4 baseTerraza = glm::mat4(1.0f);
    baseTerraza = glm::translate(baseTerraza, terrazaPos);
    baseTerraza = glm::rotate(
        baseTerraza,
        glm::radians(terrazaRotY),
        glm::vec3(0.0f, 1.0f, 0.0f)
    );
    baseTerraza = glm::scale(baseTerraza, terrazaScale);
```

```
// ---------- MESA ----------

// Tablero de la mesa
glm::mat4 m = baseTerraza;
m = glm::translate(m, glm::vec3(0.0f, 0.8f, 0.0f));       // altura del tablero
m = glm::scale(m, glm::vec3(2.0f, 0.08f, 1.2f));          // largo x ancho x grosor
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(m));
glDrawArrays(GL_TRIANGLES, 0, 36);

// Patas de la mesa (4 esquinas)
float offsetX = 0.9f;
float offsetZ = 0.5f;
float alturaPataMesa = 0.8f;

glm::vec3 patasMesa[4] = {
    glm::vec3(offsetX, alturaPataMesa / 2.0f,  offsetZ),
    glm::vec3(-offsetX, alturaPataMesa / 2.0f,  offsetZ),
    glm::vec3(offsetX, alturaPataMesa / 2.0f, -offsetZ),
    glm::vec3(-offsetX, alturaPataMesa / 2.0f, -offsetZ)
};

for (int i = 0; i < 4; ++i) {
    m = baseTerraza;
    m = glm::translate(m, patasMesa[i]);
    m = glm::scale(m, glm::vec3(0.12f, alturaPataMesa, 0.12f));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(m));
    glDrawArrays(GL_TRIANGLES, 0, 36);
}

// ---------- SILLAS (4) ----------
// Cada silla estará en un lado de la mesa
// Usamos una función "lambda" mental: baseTerraza -> base de cada silla

auto dibujarSilla = [&](glm::vec3 posLocalSilla, float rotLocalY)
    {
        glm::mat4 baseSilla = baseTerraza;
        baseSilla = glm::translate(baseSilla, posLocalSilla);
        baseSilla = glm::rotate(
            baseSilla,
            glm::radians(rotLocalY),
            glm::vec3(0.0f, 1.0f, 0.0f)
        );

        // Asiento
        glm::mat4 s = baseSilla;
        s = glm::translate(s, glm::vec3(0.0f, 0.45f, 0.0f));
        s = glm::scale(s, glm::vec3(0.6f, 0.08f, 0.6f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(s));
        glDrawArrays(GL_TRIANGLES, 0, 36);

        // Respaldo
        s = baseSilla;
        s = glm::translate(s, glm::vec3(0.0f, 0.75f, -0.25f));
        s = glm::scale(s, glm::vec3(0.6f, 0.7f, 0.08f));
```

```
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(s));
        glDrawArrays(GL_TRIANGLES, 0, 36);

        // Patas de la silla
        float hPata = 0.45f;
        glm::vec3 patasSilla[4] = {
            glm::vec3(0.22f, hPata / 2.0f,  0.22f),
            glm::vec3(-0.22f, hPata / 2.0f,  0.22f),
            glm::vec3(0.22f, hPata / 2.0f, -0.22f),
            glm::vec3(-0.22f, hPata / 2.0f, -0.22f)
        };

        for (int i = 0; i < 4; ++i) {
            s = baseSilla;
            s = glm::translate(s, patasSilla[i]);
            s = glm::scale(s, glm::vec3(0.08f, hPata, 0.08f));
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(s));
            glDrawArrays(GL_TRIANGLES, 0, 36);
        }
    };

// Posiciones locales de las 4 sillas alrededor de la mesa
dibujarSilla(glm::vec3(0.0f, 0.0f, 1.4f), 180.0f); // frente
dibujarSilla(glm::vec3(0.0f, 0.0f, -1.4f), 0.0f); // atrás
dibujarSilla(glm::vec3(1.4f, 0.0f, 0.0f), -90.0f); // derecha
dibujarSilla(glm::vec3(-1.4f, 0.0f, 0.0f), 90.0f); // izquierda

glBindVertexArray(0);

// ================= FAROLA (CUBOS) =================
lightingShader.Use();

// VOLVER A OBTENER LOS UNIFORMS PARA ESTE SHADER
modelLoc = glGetUniformLocation(lightingShader.Program, "model");
viewLoc = glGetUniformLocation(lightingShader.Program, "view");
projLoc = glGetUniformLocation(lightingShader.Program, "projection");

// Aseguramos view y projection correctos
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));

glBindVertexArray(VAO);

// Escala global en X,Z (grosor) y factor de altura en Y
float S = 1.0f;   // misma base/grosor
float H = 2.5f;   // farola más alta

// BASE de concreto
glm::mat4 farolaModel = glm::mat4(1.0f);
farolaModel = glm::translate(
    farolaModel,
    lamparaPos + glm::vec3(0.0f, 0.15f, 0.0f)
);
farolaModel = glm::scale(
```

```
// BASE de concreto
glm::mat4 farolaModel = glm::mat4(1.0f);
farolaModel = glm::translate(
    farolaModel,
    lamparaPos + glm::vec3(0.0f, 0.15f, 0.0f)
);
farolaModel = glm::scale(
    farolaModel,
    glm::vec3(0.7f * S, 0.3f * S, 0.7f * S)
);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(farolaModel));
glDrawArrays(GL_TRIANGLES, 0, 36);

// Poste principal (vertical) - MÁS ALTO
farolaModel = glm::mat4(1.0f);
farolaModel = glm::translate(
    farolaModel,
    lamparaPos + glm::vec3(0.0f, 1.3f * H, 0.0f)
);
farolaModel = glm::scale(
    farolaModel,
    glm::vec3(0.15f * S, 2.5f * H, 0.15f * S)
);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(farolaModel));
glDrawArrays(GL_TRIANGLES, 0, 36);

// Brazo horizontal - sube junto con la altura
farolaModel = glm::mat4(1.0f);
farolaModel = glm::translate(
    farolaModel,
    lamparaPos + glm::vec3(0.4f * S, 2.5f * H, 0.0f)
);
farolaModel = glm::scale(
    farolaModel,
    glm::vec3(0.8f * S, 0.12f * S, 0.12f * S)
);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(farolaModel));
glDrawArrays(GL_TRIANGLES, 0, 36);

// Poste pequeño que baja del brazo
farolaModel = glm::mat4(1.0f);
farolaModel = glm::translate(
    farolaModel,
    lamparaPos + glm::vec3(0.8f * S, 2.3f * H, 0.0f)
);
farolaModel = glm::scale(
    farolaModel,
    glm::vec3(0.10f * S, 0.6f * H, 0.10f * S)
);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(farolaModel));
glDrawArrays(GL_TRIANGLES, 0, 36);
```

```
  // Cabeza de la lámpara
  farolaModel = glm::mat4(1.0f);
  farolaModel = glm::translate(
      farolaModel,
      lamparaPos + glm::vec3(0.8f * S, 2.0f * H, 0.0f)
  );
  farolaModel = glm::scale(
      farolaModel,
      glm::vec3(0.5f * S, 0.35f * S, 0.5f * S)
  );
  glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(farolaModel));
  glDrawArrays(GL_TRIANGLES, 0, 36);

  // Techo de la lámpara
  farolaModel = glm::mat4(1.0f);
  farolaModel = glm::translate(
      farolaModel,
      lamparaPos + glm::vec3(0.8f * S, 2.25f * H, 0.0f)
  );
  farolaModel = glm::scale(
      farolaModel,
      glm::vec3(0.7f * S, 0.1f * S, 0.7f * S)
  );
  glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(farolaModel));
  glDrawArrays(GL_TRIANGLES, 0, 36);

  glBindVertexArray(0);

================ CÁMARA DE SEGURIDAD (CUBOS) ==================
  colorShader.Use();

  modelLoc = glGetUniformLocation(colorShader.Program, "model");
  viewLoc = glGetUniformLocation(colorShader.Program, "view");
  projLoc = glGetUniformLocation(colorShader.Program, "projection");
  GLint colorLoc = glGetUniformLocation(colorShader.Program, "colorAlpha");

  glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
  glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));

  GLint useTexLoc = glGetUniformLocation(colorShader.Program, "useTexture");
  if (useTexLoc != -1)
      glUniform1i(useTexLoc, 0);

  glBindVertexArray(VAO);

  // ================== MATRIZ BASE ==================
  glm::mat4 baseCam = glm::mat4(1.0f);
  baseCam = glm::translate(baseCam, camSegPos);
  baseCam = glm::rotate(baseCam, glm::radians(camSegBaseRotY), glm::vec3(0.0f, 1.0f, 0.0f));
  baseCam = glm::scale(baseCam, camSegScale);

  // ---------- COLOR PRINCIPAL (PLATEADO FRÍO) ----------
  glUniform4f(colorLoc, 0.65f, 0.65f, 0.7f, 1.0f); // para el brazo y placa
```

```
// ---------- COLOR PRINCIPAL (PLATEADO FRÍO) ----------
glUniform4f(colorLoc, 0.65f, 0.65f, 0.7f, 1.0f); // para el brazo y placa

// ---------- PLACA DE PARED ----------
m = baseCam;
m = glm::translate(m, glm::vec3(0.0f, 0.0f, 0.0f));
m = glm::scale(m, glm::vec3(0.8f, 1.0f, 0.1f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(m));
glDrawArrays(GL_TRIANGLES, 0, 36);

// ---------- BRAZO DEL SOPORTE ----------
m = baseCam;
m = glm::translate(m, glm::vec3(0.0f, -0.1f, -0.35f));
m = glm::scale(m, glm::vec3(0.15f, 0.15f, 0.7f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(m));
glDrawArrays(GL_TRIANGLES, 0, 36);

// ---------- ARTICULACIÓN ----------
m = baseCam;
m = glm::translate(m, glm::vec3(0.0f, -0.1f, -0.7f));
m = glm::scale(m, glm::vec3(0.25f, 0.25f, 0.25f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(m));
glDrawArrays(GL_TRIANGLES, 0, 36);

// ================= CABEZA DE LA CÁMARA =================
glm::mat4 baseCamHead = baseCam;
baseCamHead = glm::translate(baseCamHead, glm::vec3(0.0f, -0.1f, -0.9f));
baseCamHead = glm::rotate(baseCamHead, glm::radians(camSegPanAngle), glm::vec3(0.0f, 1.0f, 0.0f));

// ---------- CUERPO (PLATEADO) ----------
glUniform4f(colorLoc, 0.75f, 0.75f, 0.8f, 1.0f);
m = baseCamHead;
m = glm::translate(m, glm::vec3(0.0f, 0.0f, -0.2f));
m = glm::scale(m, glm::vec3(0.9f, 0.6f, 0.7f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(m));
glDrawArrays(GL_TRIANGLES, 0, 36);

// ---------- FRONTAL (PLATEADO) ----------
m = baseCamHead;
m = glm::translate(m, glm::vec3(0.0f, 0.0f, -0.65f));
m = glm::scale(m, glm::vec3(0.7f, 0.5f, 0.3f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(m));
glDrawArrays(GL_TRIANGLES, 0, 36);

// ---------- LENTE (NEGRO BRILLANTE) ----------
glUniform4f(colorLoc, 0.1f, 0.1f, 0.1f, 1.0f);
m = baseCamHead;
m = glm::translate(m, glm::vec3(0.0f, 0.0f, -0.9f));
m = glm::scale(m, glm::vec3(0.25f, 0.25f, 0.4f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(m));
glDrawArrays(GL_TRIANGLES, 0, 36);
```

```
// ================= RESET =================
glUniform4f(colorLoc, 1.0f, 1.0f, 1.0f, 1.0f);
glBindVertexArray(0);

// ================ ARBUSTO LOW POLY (CUBOS) ================
colorShader.Use();

// Uniforms de matrices
modelLoc = glGetUniformLocation(colorShader.Program, "model");
viewLoc = glGetUniformLocation(colorShader.Program, "view");
projLoc = glGetUniformLocation(colorShader.Program, "projection");

glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
glUniform1i(glGetUniformLocation(colorShader.Program, "useTexture"), 0);

glBindVertexArray(VAO);

// Matriz base del arbusto
glm::mat4 arbBase = glm::mat4(1.0f);
arbBase = glm::translate(arbBase, arbustoPos);
arbBase = glm::rotate(arbBase, glm::radians(arbustoRotY), glm::vec3(0.0f, 1.0f, 0.0f));
arbBase = glm::scale(arbBase, arbustoScale * 0.8f);

glm::mat4 arbPart;

// ---------- TRONCO ----------
glUniform4f(glGetUniformLocation(colorShader.Program, "colorAlpha"),
    0.35f, 0.22f, 0.07f, 1.0f);

arbPart = arbBase;
arbPart = glm::translate(arbPart, glm::vec3(0.0f, 0.20f, 0.0f));
arbPart = glm::scale(arbPart, glm::vec3(0.20f, 0.4f, 0.20f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(arbPart));
glDrawArrays(GL_TRIANGLES, 0, 36);

// ---------- HOJAS ----------
glUniform4f(glGetUniformLocation(colorShader.Program, "colorAlpha"),
    0.10f, 0.38f, 0.15f, 1.0f);

// Bloque central
arbPart = arbBase;
arbPart = glm::translate(arbPart, glm::vec3(0.0f, 0.8f, 0.0f));
arbPart = glm::scale(arbPart, glm::vec3(1.3f, 1.0f, 1.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(arbPart));
glDrawArrays(GL_TRIANGLES, 0, 36);

// Lateral izquierdo
arbPart = arbBase;
arbPart = glm::translate(arbPart, glm::vec3(-0.5f, 0.7f, 0.0f));
arbPart = glm::scale(arbPart, glm::vec3(0.9f, 0.9f, 0.9f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(arbPart));
glDrawArrays(GL_TRIANGLES, 0, 36);
```

```
// Lateral derecho
arbPart = arbBase;
arbPart = glm::translate(arbPart, glm::vec3(0.5f, 0.7f, 0.0f));
arbPart = glm::scale(arbPart, glm::vec3(0.9f, 0.9f, 0.9f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(arbPart));
glDrawArrays(GL_TRIANGLES, 0, 36);

// Superior
arbPart = arbBase;
arbPart = glm::translate(arbPart, glm::vec3(0.0f, 1.25f, 0.0f));
arbPart = glm::scale(arbPart, glm::vec3(1.0f, 0.8f, 1.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(arbPart));
glDrawArrays(GL_TRIANGLES, 0, 36);

// Reset color
glUniform4f(glGetUniformLocation(colorShader.Program, "colorAlpha"),
    1.0f, 1.0f, 1.0f, 1.0f);
glBindVertexArray(0);

// ================= VENTILADOR DE TECHO (CUBOS) =================
colorShader.Use();

// Uniforms de matrices
modelLoc = glGetUniformLocation(colorShader.Program, "model");
viewLoc = glGetUniformLocation(colorShader.Program, "view");
projLoc = glGetUniformLocation(colorShader.Program, "projection");

glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
glUniform1i(glGetUniformLocation(colorShader.Program, "useTexture"), 0);

glBindVertexArray(VAO);

// Matriz base del ventilador (traslación + rotación + escala global)
glm::mat4 fanBase = glm::mat4(1.0f);
fanBase = glm::translate(fanBase, fanPos);
fanBase = glm::rotate(fanBase, glm::radians(fanRotY), glm::vec3(0.0f, 1.0f, 0.0f));
fanBase = glm::scale(fanBase, fanScale);

glm::mat4 fm;

// ---------- PLAFÓN (base pegada al techo) ----------
glUniform4f(glGetUniformLocation(colorShader.Program, "colorAlpha"),
    0.7f, 0.7f, 0.7f, 1.0f);  // gris claro

fm = fanBase;
fm = glm::translate(fm, glm::vec3(0.0f, 0.05f, 0.0f));
fm = glm::scale(fm, glm::vec3(0.8f, 0.1f, 0.8f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(fm));
glDrawArrays(GL_TRIANGLES, 0, 36);
```

```
// ---------- TUBO VERTICAL ----------
glUniform4f(glGetUniformLocation(colorShader.Program, "colorAlpha"),
    0.55f, 0.55f, 0.55f, 1.0f);  // gris medio

fm = fanBase;
fm = glm::translate(fm, glm::vec3(0.0f, -0.35f, 0.0f));
fm = glm::scale(fm, glm::vec3(0.12f, 0.7f, 0.12f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(fm));
glDrawArrays(GL_TRIANGLES, 0, 36);

// ---------- CUERPO / MOTOR ----------
glUniform4f(glGetUniformLocation(colorShader.Program, "colorAlpha"),
    0.4f, 0.4f, 0.4f, 1.0f);

fm = fanBase;
fm = glm::translate(fm, glm::vec3(0.0f, -0.75f, 0.0f));
fm = glm::scale(fm, glm::vec3(0.6f, 0.25f, 0.6f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(fm));
glDrawArrays(GL_TRIANGLES, 0, 36);

// ========= BASE DE LAS ASPAS =========
glm::mat4 fanHead = fanBase;
fanHead = glm::translate(fanHead, glm::vec3(0.0f, -0.75f, 0.0f));
fanHead = glm::rotate(fanHead, glm::radians(fanBladesAngle), glm::vec3(0.0f, 1.0f, 0.0f));

// ---------- ASPAS ----------
glUniform4f(glGetUniformLocation(colorShader.Program, "colorAlpha"),
    0.65f, 0.65f, 0.65f, 1.0f);

for (int i = 0; i < 4; ++i) {
    float offsetAngle = i * 90.0f;

    glm::mat4 blade = fanHead;
    blade = glm::rotate(blade, glm::radians(offsetAngle), glm::vec3(0.0f, 1.0f, 0.0f));
    blade = glm::translate(blade, glm::vec3(0.9f, 0.0f, 0.0f));
    blade = glm::scale(blade, glm::vec3(1.6f, 0.07f, 0.25f));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(blade));
    glDrawArrays(GL_TRIANGLES, 0, 36);
}

// Reset color
glUniform4f(glGetUniformLocation(colorShader.Program, "colorAlpha"),
    1.0f, 1.0f, 1.0f, 1.0f);
glBindVertexArray(0);
```

```
// ================= RELOJ DE PARED (CUBOS) =================
    colorShader.Use();

    // Uniforms de matrices
    modelLoc = glGetUniformLocation(colorShader.Program, "model");
    viewLoc = glGetUniformLocation(colorShader.Program, "view");
    projLoc = glGetUniformLocation(colorShader.Program, "projection");

    glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
    glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));

    glBindVertexArray(VAO);

    // Matriz base del reloj (posición + rotación + escala)
    glm::mat4 clockBase = glm::mat4(1.0f);
    clockBase = glm::translate(clockBase, clockPos);
    clockBase = glm::rotate(clockBase, glm::radians(clockRotY), glm::vec3(0.0f, 1.0f, 0.0f));
    clockBase = glm::scale(clockBase, clockScale);

    glm::mat4 cm;

    // ========== COLOR BASE GRIS ==========
    glUniform1i(glGetUniformLocation(colorShader.Program, "useTexture"), 0);

    // ---------- CUERPO DEL RELOJ (FONDO) ----------
    glUniform4f(glGetUniformLocation(colorShader.Program, "colorAlpha"),
        0.6f, 0.6f, 0.6f, 1.0f); // gris medio

    cm = clockBase;
    cm = glm::translate(cm, glm::vec3(0.0f, 0.0f, 0.0f));
    cm = glm::scale(cm, glm::vec3(1.8f, 1.8f, 0.15f));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(cm));
    glDrawArrays(GL_TRIANGLES, 0, 36);

    // ---------- ARO EXTERIOR ----------
    glUniform4f(glGetUniformLocation(colorShader.Program, "colorAlpha"),
        0.3f, 0.3f, 0.3f, 1.0f); // gris oscuro

    cm = clockBase;
    cm = glm::scale(cm, glm::vec3(1.95f, 1.95f, 0.2f));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(cm));
    glDrawArrays(GL_TRIANGLES, 0, 36);

    // ---------- MARCAS HORARIAS ----------
    glUniform4f(glGetUniformLocation(colorShader.Program, "colorAlpha"),
        0.1f, 0.1f, 0.1f, 1.0f); // casi negro

    for (int i = 0; i < 12; ++i) {
        float ang = glm::radians(i * 30.0f); // 360 / 12
        glm::mat4 mark = clockBase;
        mark = glm::rotate(mark, ang, glm::vec3(0.0f, 0.0f, 1.0f));
        mark = glm::translate(mark, glm::vec3(0.0f, 0.9f, 0.06f));
        mark = glm::scale(mark, glm::vec3(0.15f, 0.25f, 0.12f));
```

```
for (int i = 0; i < 12; ++i) {
    float ang = glm::radians(i * 30.0f); // 360 / 12
    glm::mat4 mark = clockBase;
    mark = glm::rotate(mark, ang, glm::vec3(0.0f, 0.0f, 1.0f));
    mark = glm::translate(mark, glm::vec3(0.0f, 0.9f, 0.06f));
    mark = glm::scale(mark, glm::vec3(0.15f, 0.25f, 0.12f));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(mark));
    glDrawArrays(GL_TRIANGLES, 0, 36);
}

// ---------- MANECILLA LARGA (MINUTERO) ----------
glUniform4f(glGetUniformLocation(colorShader.Program, "colorAlpha"),
    0.15f, 0.15f, 0.15f, 1.0f); // gris oscuro

glm::mat4 minH = clockBase;
minH = glm::rotate(minH, glm::radians(clockMinuteAngle), glm::vec3(0.0f, 0.0f, 1.0f));
minH = glm::translate(minH, glm::vec3(0.0f, 0.9f, 0.08f));
minH = glm::scale(minH, glm::vec3(0.12f, 1.8f, 0.14f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(minH));
glDrawArrays(GL_TRIANGLES, 0, 36);

// ---------- MANECILLA CORTA (HORA) ----------
glUniform4f(glGetUniformLocation(colorShader.Program, "colorAlpha"),
    0.15f, 0.15f, 0.15f, 1.0f); // gris oscuro

glm::mat4 hourH = clockBase;
hourH = glm::rotate(hourH, glm::radians(clockHourAngle), glm::vec3(0.0f, 0.0f, 1.0f));
hourH = glm::translate(hourH, glm::vec3(0.0f, 0.6f, 0.09f));
hourH = glm::scale(hourH, glm::vec3(0.16f, 1.2f, 0.16f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(hourH));
glDrawArrays(GL_TRIANGLES, 0, 36);

// ---------- TAPITA CENTRAL ----------
glUniform4f(glGetUniformLocation(colorShader.Program, "colorAlpha"),
    0.05f, 0.05f, 0.05f, 1.0f); // gris casi negro

cm = clockBase;
cm = glm::translate(cm, glm::vec3(0.0f, 0.0f, 0.11f));
cm = glm::scale(cm, glm::vec3(0.3f, 0.3f, 0.18f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(cm));
glDrawArrays(GL_TRIANGLES, 0, 36);

// Reset
glUniform4f(glGetUniformLocation(colorShader.Program, "colorAlpha"),
    1.0f, 1.0f, 1.0f, 1.0f);
glBindVertexArray(0);
```

Animaciones que dependen de la interacción del usuario

En este caso, las animaciones sencillas son controladas por las variables booleanas, y declaradas en su funcionamiento en el apartado de KeyCallBack, en este queda descrito que tecla va a activar qué función y cuál va a ser el movimiento por realizar.

```
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode)
{
    if (GLFW_KEY_ESCAPE == key && GLFW_PRESS == action)
    {
        glfwSetWindowShouldClose(window, GL_TRUE);
    }

    if (key >= 0 && key < 1024)
    {
        if (action == GLFW_PRESS)
        {
            keys[key] = true;
        }
        else if (action == GLFW_RELEASE)
        {
            keys[key] = false;
        }
    }
    if (key == GLFW_KEY_SPACE && action == GLFW_PRESS)
    {
        active = !active;
        lightsOn = active;

        if (!active) { // apagar
            Light1 = Light2 = Light3 = Light4 = glm::vec3(0.0f);
        }
        else {       // encender
            Light1 = glm::vec3(1.0f);
            Light2 = glm::vec3(2.0f);
            Light3 = glm::vec3(2.5f);
            Light4 = glm::vec3(3.0f);
        }
    }
    //Apertura/Cerrado de ventanas
    if (key == GLFW_KEY_V && action == GLFW_PRESS) {
        if (ventanaDesplazada) {
            ventanaPosZ = 0.0f;  // Regresar a la posición inicial
        }
        else {
            ventanaPosZ = -0.5f;  // Desplazar 1 unidad en el eje z
        }
        ventanaDesplazada = !ventanaDesplazada;  // Cambiar el estado de desplazamiento
    }
```

```
    }
    // PUERTA DEL EDIFICIO
    if (key == GLFW_KEY_P && action == GLFW_PRESS)
    {
        if (!puertaEstaAbierta) {
            puertaT = 0.0f;
            puertaAnimando = true;
            puertaEstaAbierta = true;
        }
        else {
            puertaT = 0.0f;
            puertaAnimando = true;
            puertaEstaAbierta = false;
        }
    }
    // PUERTA DE LA ENTRADA DEL MUSEO
    if (key == GLFW_KEY_1 && action == GLFW_PRESS)
    {
        if (!animPuerta) {
            animPuerta = true;
        }
    }

    // COCHE
    if (keys[GLFW_KEY_2])
    {
        animCoche = !animCoche;
    }

    //Animación del faro
    if (key == GLFW_KEY_C && action == GLFW_PRESS)
    {
        indiceColorLampara = (indiceColorLampara + 1) % 3;
    }
}
```

Animaciones que se inicializan automáticamente al ejecutar el programa:

Estas animaciones no dependen de la interacción del usuario

```
// ----- ANIMACIÓN DE PÁNEO DE LA CÁMARA DE SEGURIDAD -----
if (camSegPanForward) {
    camSegPanAngle += camSegPanSpeed * deltaTime;
    if (camSegPanAngle > camSegPanLimit) {
        camSegPanAngle = camSegPanLimit;
        camSegPanForward = false;   // cambia de sentido
    }
}
else {
    camSegPanAngle -= camSegPanSpeed * deltaTime;
    if (camSegPanAngle < -camSegPanLimit) {
        camSegPanAngle = -camSegPanLimit;
        camSegPanForward = true;    // cambia de sentido
    }
}

// ----- ANIMACIÓN DEL VENTILADOR DE TECHO -----
fanBladesAngle += fanBladesSpeed * deltaTime;
if (fanBladesAngle > 360.0f)
    fanBladesAngle -= 360.0f;


// ----- ANIMACIÓN DEL RELOJ DE PARED -----
clockMinuteAngle += clockMinuteSpeed * deltaTime;
clockHourAngle += clockHourSpeed * deltaTime;

if (clockMinuteAngle > 360.0f) clockMinuteAngle -= 360.0f;
if (clockHourAngle > 360.0f) clockHourAngle -= 360.0f;
```

Animaciones complejas - Para la creación de las animaciones complejas se requirió de la creación de nuevos shaders que las controlaran.

Tenemos Tv.vs y Tv.frag estos Shaders juegan con la parte especular de nuestro televisor, mostrando que esta se quedo encendida y no da imagen, lo único que muestra es ruido.

Tv.vs

```
#version 330 core
layout (location = 0) in vec3 position;
layout (location = 1) in vec3 normal;
layout (location = 2) in vec2 texCoords;

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

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

void main()
{
    FragPos = vec3(model * vec4(position, 1.0));
    Normal = mat3(transpose(inverse(model))) * normal;
    TexCoords = texCoords;

    gl_Position = projection * view * model * vec4(position, 1.0);
}
```

Tv.frag

```
#version 330 core

struct Material
{
    sampler2D diffuse;
    float shininess;
};

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

out vec4 FragColor;

uniform Material material;
uniform float time;

// Función para aplicar ruido usando la textura difusa
vec3 applyNoiseEffect(vec2 coords, float time)
{
    vec3 noiseColor = texture(material.diffuse, coords + vec2(sin(time), cos(time))).rgb;
    return noiseColor;
}

void main()
{
    vec4 baseColor = texture(material.diffuse, TexCoords);
    vec3 noiseEffect = applyNoiseEffect(TexCoords, time);
    vec3 finalColor = mix(baseColor.rgb, noiseEffect, 0.5); // Ajusta la mezcla según se requiera
    FragColor = vec4(finalColor, baseColor.a);
}
```

La segunda animación compleja es la animación del movimiento del cáliz de fuego, para esto se necesitan los siguientes Shaders:

AnimFuego.vs

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

const float amplitude = 0.01;
const float frequency = 4.0;
const float PI = 3.14159;
out vec2 TexCoords;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;
uniform float time;

void main()
{
  float distance = length(aPos);
  float effect = amplitude*sin(-PI*distance*frequency+time);
  gl_Position = projection*view*model*vec4(aPos.x+effect,aPos.y+effect, aPos.z+effect,1);
  TexCoords=vec2(aTexCoords.x+effect,aTexCoords.y+effect);

}
```

AnimFuego.frag

```
#version 330 core
out vec4 FragColor;
in vec2 TexCoords;
uniform sampler2D texture1;

void main()
{

  vec4 texColor= texture(texture1,TexCoords);
    if(texColor.a < 0.1)
        discard;
    FragColor = texColor;

}
```

En resumen, hay 5 animaciones automáticas que se ejecutan al iniciar el programa.

Paneo de la cámara	Oscila entre ángulos
Ventilador de techo	Las aspas giran continuamente
Reloj de pared	Las manecillas giran con velocidad distinta.
Cáliz de fuego	Animación visual generada desde shaders (AnimFuego.vs, AnimFuego.frag)
TV sin señal	Animación visual generada desde shaders (tv.vs, tv.frag)

Y 6 animaciones controladas por el usuario

Animación	Tecla	Tipo
Puerta principal	1	Incremental (rotación progresiva)
Puerta secundaria / interior	P	Keyframe interpolado
Movimiento del coche	2	Fases de traslación + rotación
Desplazamiento de ventanas	V	Traslación lineal
Cambio de color de la farola	C	Ciclo entre colores
Cambio de color de las luces de la casa	ESPACIO	Posición / dirección variable

Conclusión

El desarrollo de una aplicación gráfica interactiva en 3D con OpenGL y GLFW en C++ ha sido un proceso integral que abarcó desde el análisis de requerimientos hasta la iteración continua. El proyecto permitió la creación de un entorno virtual detallado y funcional, donde se pudieron aplicar los distintos conocimientos adquiridos a lo largo del curso.

Este proyecto incluyó modelos 3D, iluminación y animaciones, permitiendo una interacción inmersiva del usuario. A pesar de enfrentar desafíos como la creación de objetos y el modelado de estos en un software dedicado como lo es Blender. Además de contar con ciertas incógnitas al momento de texturizar cada cosa que es parte del proyecto.

Las futuras mejoras, como la implementación de físicas realistas, muestran el potencial del proyecto para seguir creciendo. En resumen, el proyecto fue desafiante y entretenido de realizar. Esto debido a que, en términos de aprendizaje, desarrollo técnico y creatividad, se tuvo que demostrar el conocimiento que se adquirió durante el semestre.