UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

LABORATORIO DE COMPUTACION GRAFICA E INTERACCION HUMANO-COMPUTADORA

Ing. Edén Espinoza Urzua

Semestre 2026-1

Proyecto Final

Documentación Tecnica

Grupo: 4

Alumno:

Zavala Mendoza Luis Enrique

OBJETIVOS:

Representación visual de la cocina y la sala de la casa de Marceline

El proyecto busca recrear de manera interactiva la cocina y la sala de la casa de Marceline, utilizando herramientas gráficas como OpenGL. La intención es ofrecer una versión visual clara y estilizada de estos dos espacios, tomando como referencia su diseño dentro de *Adventure Time* y manteniendo la atmósfera oscura y peculiar que caracteriza al personaje.

Simulación de los espacios principales

La simulación se enfocará únicamente en estos dos ambientes. En la sala se incluirán elementos distintivos como el sillón desgastado, el bajo hacha de Marceline y algunos objetos decorativos propios de su estilo. En la cocina se añadirán muebles, utensilios y detalles simples que reflejen el aspecto rústico y algo desordenado que suele tener en la serie. Ambos espacios se conectarán de manera coherente para que el usuario pueda entender su distribución general.

Interacción y exploración del usuario

El usuario podrá recorrer libremente la sala y la cocina, observando los detalles y la organización de cada área. La interacción será básica: desplazamiento mediante teclado y posibilidad de explorar el entorno desde distintos ángulos, con el objetivo de ofrecer una experiencia sencilla pero inmersiva.

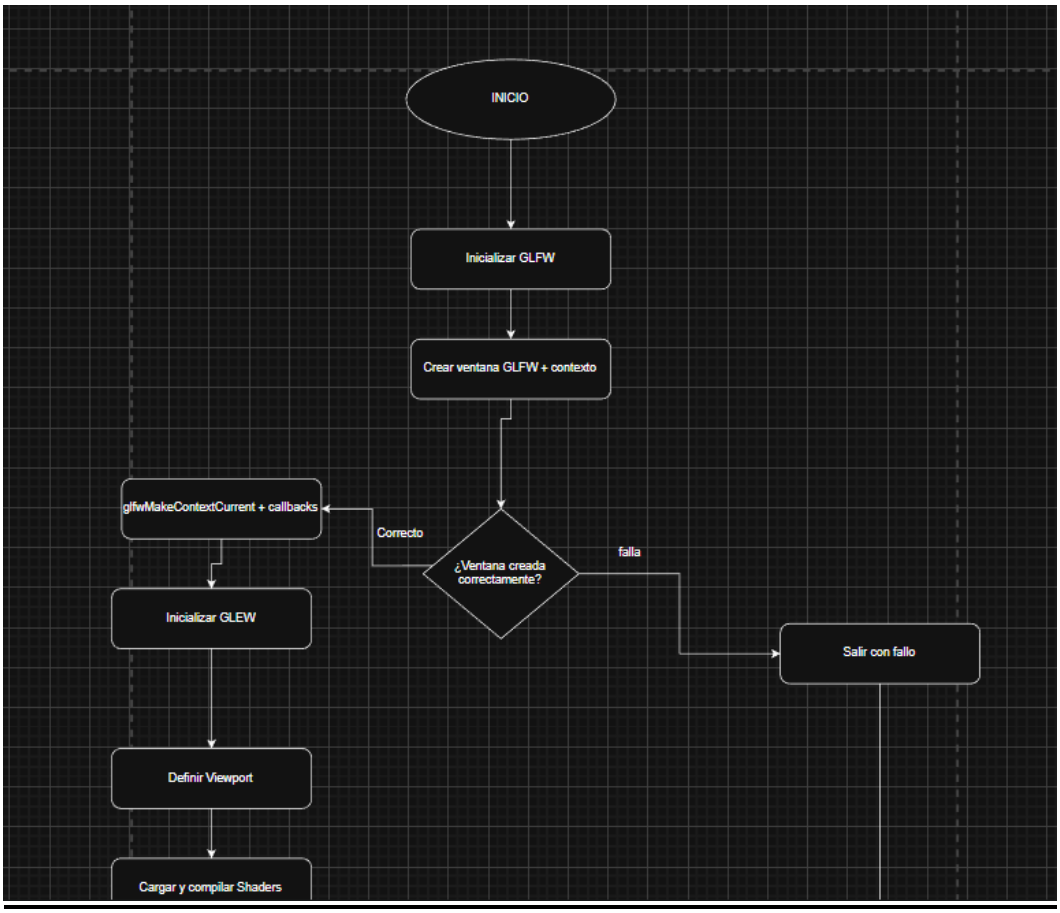
Modelado 3D simplificado

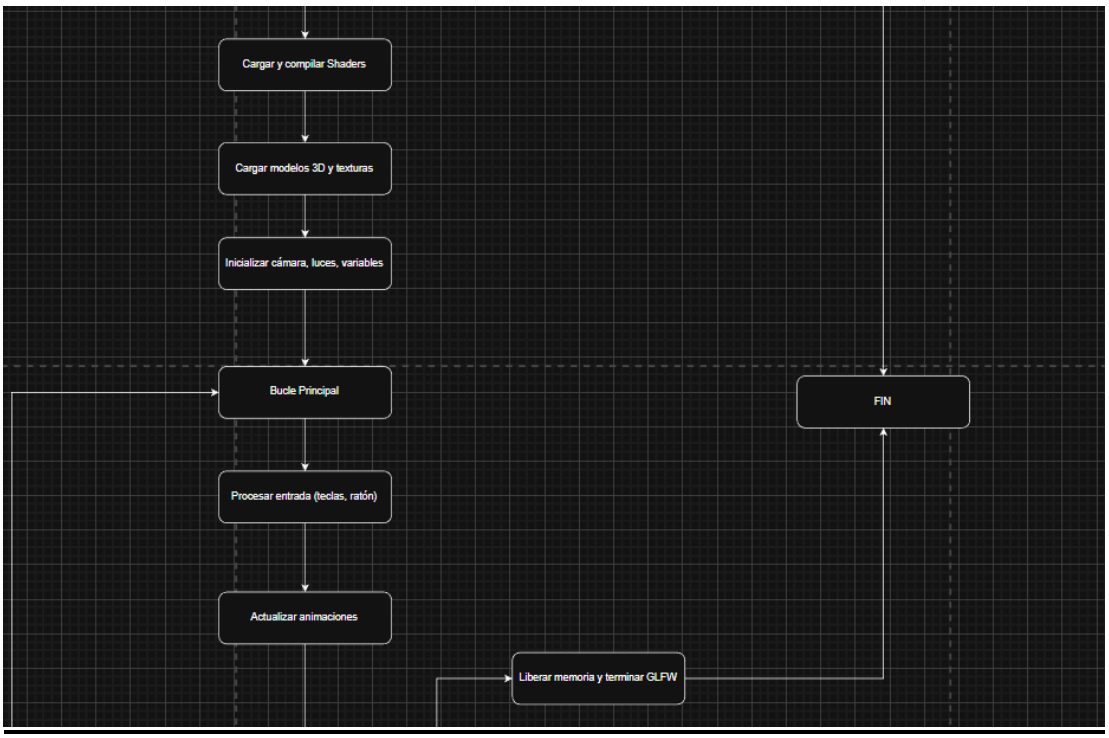
Los modelos de los muebles y objetos de ambas habitaciones se crearán en Blender. Aunque no se buscará un nivel extremo de realismo, sí se intentará mantener un estilo visual cercano a la serie, con texturas simples, colores planos y detalles característicos de los ambientes de Marceline.

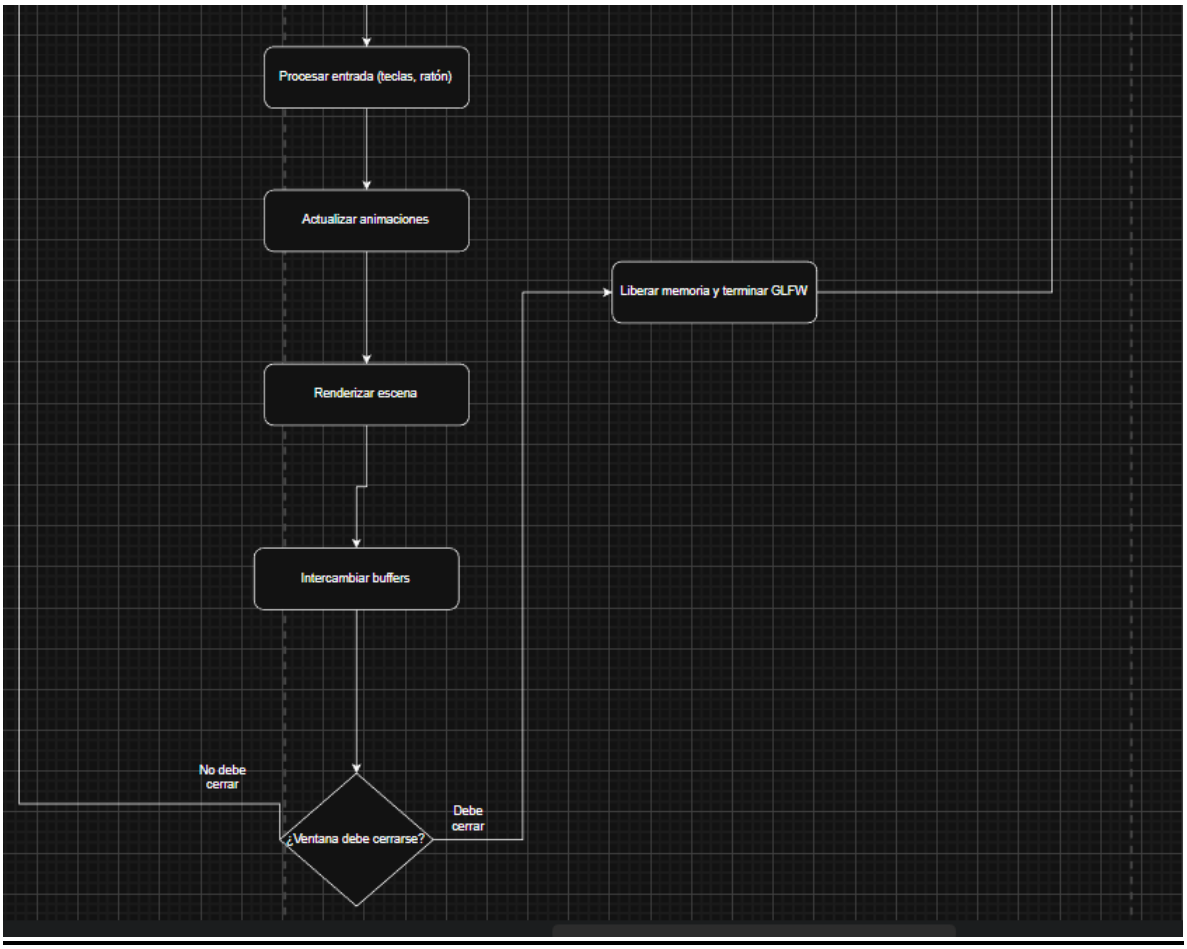
Implementación en OpenGL

La representación final se desarrollará en OpenGL, integrando iluminación básica para resaltar el ambiente sombrío y cálido de la casa. Se emplearán shaders sencillos, animaciones mínimas si fuese necesario y movimiento de cámara controlado con teclado para explorar ambas habitaciones sin complicaciones.

DIAGRAMA DE FLUJO:





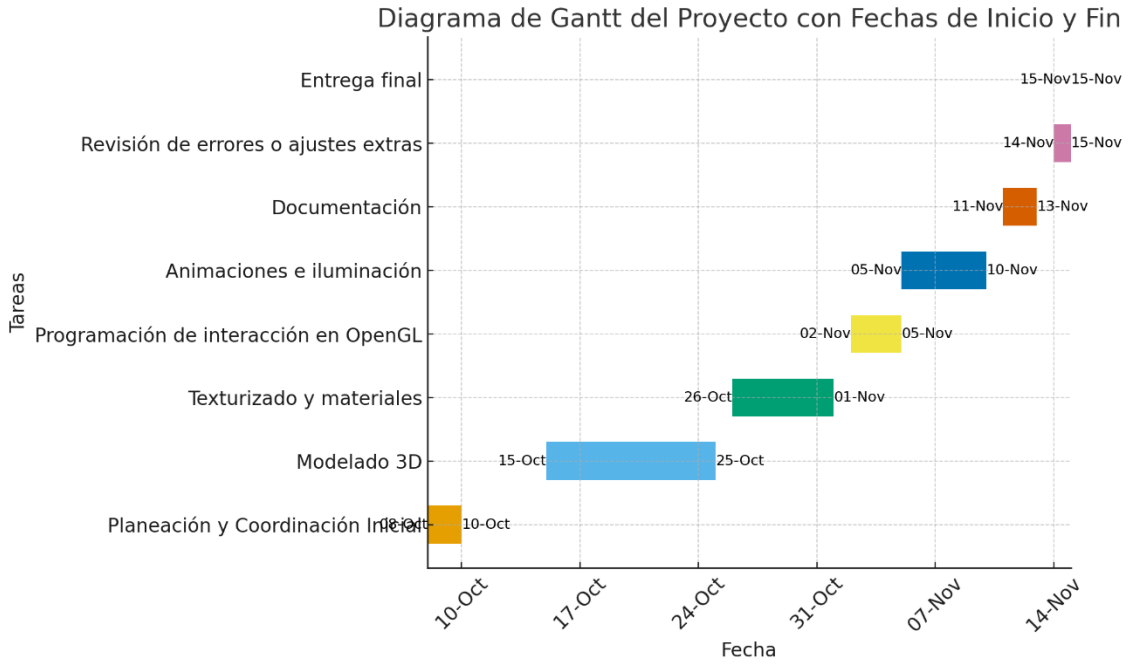


ALCANCE DEL PROYECTO

El proyecto consiste en crear una representación visual interactiva de la sala, la cocina y la fachada de la casa de Marceline, tomando como referencia su estilo dentro de *Adventure Time*. Aunque se mantiene la estética característica de la serie, se añadieron algunos detalles propios para darle más profundidad al entorno. Todo el modelo fue desarrollado en OpenGL, buscando que el usuario pueda recorrer estas áreas de manera sencilla y disfrutar de un espacio visualmente atractivo. La intención es que quien lo explore pueda apreciar la ambientación oscura y peculiar de la casa, tanto por dentro como por fuera, sin perder la esencia del diseño original.

Vista de la casa de la serie animada.

Diagrama de Gantt



Componentes Principales del Alcance

Modelado de la casa

Fachada

Representamos unos de los lugares más emblemáticos de la casa como lo es la fachada con un diseño exterior gótico y algo sombrío, la puerta principal oscura y las paredes con texturas de madera envejecida.

Modelado 3D de elementos clave dentro la casa:

Cocina

Un mueble sartén, olla y un volteador también cuenta con una silla que no siempre se ve dentro de la cocina.

Sala

el cuadro, la mesita donde tiene una lampara, los sillones y los bancos que tiene junto a la barra de la cocina

Implementación en OpenGL

- Desarrollo de un entorno virtual interactivo que permita al usuario navegar por la casa
- Uso de shaders para efectos visuales animados
- Animaciones básicas y complejas

Interacción del usuario:

Se permite la navegación en el entorno visual mediante controles de cámara (movimiento con teclas W, A, S, D, Espacio y Shift también cuenta con el movimiento de raton para rotación de la cámara.

Posibilidad de interactuar con los modelos y ver animaciones con las teclas K, P, O y C.

Entorno Tecnico

- Uso de bibliotecas como GLFW para la gestión de ventanas y entrada, GLAD para la carga de funciones OpenGL, y GLM para operaciones matemáticas.
- Carga de modelos en formato FBX y texturas en formato PNG y JPG.
- Implementación del sistema de cámara para navegación libre en el entorno.

Limitantes:

En el desarrollo de este proyecto nos enfrentamos a algunas limitantes que afectan su funcionalidad y desempeño que se explican a continuación:

1. Técnicas:

- **Compatibilidad de hardware:**
  Para ejecutar el entorno 3D que incluye la sala, la cocina y la fachada de la casa de Marceline, es necesario contar con un equipo que soporte OpenGL 3.3 o superior. En dispositivos antiguos o con capacidades gráficas muy limitadas, la experiencia puede verse afectada, especialmente en la carga de texturas y la renderización de sombras.

- **Rendimiento:**
  Aunque el proyecto utiliza modelos relativamente simples, la inclusión de varias texturas estilizadas, iluminación puntual y efectos básicos de sombreado puede causar pequeñas caídas de rendimiento en computadoras con GPUs de gama baja.
  El tiempo de carga del entorno suele rondar entre 20 y 30 segundos dependiendo del equipo, y el consumo de memoria, si bien no es elevado, puede ser un inconveniente en dispositivos con muy poca RAM.

2. Conocimiento y experiencia:

- La curva de aprendizaje al trabajar con OpenGL, junto con el modelado básico en Blender, presentó varios desafíos. Uno de los más notables fue la correcta aplicación de shaders simples (como Phong) para lograr una iluminación coherente con la estética oscura y suave de la casa de Marceline.
  También hubo dificultades para ajustar las texturas y evitar problemas de estiramiento o mal mapeo, lo cual requirió varias pruebas hasta obtener un resultado visual adecuado.

Por otro lado, uno de los retos más grandes fue familiarizarme con el uso de Blender para crear los modelos de la sala, la cocina y la fachada. Al inicio, el proceso resultó bastante tardado, ya que era necesario comprender distintas herramientas básicas del modelado 3D. Entre ellas, aprender a usar modificadores, realizar un mapeo UV adecuado para que las texturas se acomodaran correctamente, y corregir errores comunes como caras o vértices mal colocados.
Además, fue necesario entender cómo exportar los modelos sin perder materiales ni proporciones, de forma que al llevarlos a OpenGL pudieran visualizarse tal como se habían diseñado. Todo este proceso requirió práctica constante, pero permitió mejorar significativamente la calidad del entorno final.

Gran parte del tiempo de desarrollo se dedicó a aprender desde cero cómo modelar y aplicar texturas en Blender. Esta etapa fue la más extensa del proyecto, ya que

implicó familiarizarse con las herramientas básicas del programa y comprender el proceso completo para crear modelos utilizables dentro del entorno 3D.

Aun con las limitaciones mencionadas, el proyecto logra cumplir su propósito principal: ofrecer una representación visual clara y atractiva de la sala, la cocina y la fachada de la casa de Marceline, permitiendo explorar estos espacios de manera sencilla y con una ambientación acorde a la estética de la serie.

METODOLOGÍA DE SOFTWARE APLICADA

Para el desarrollo del proyecto se aplicó la metodología cascada, la cual se caracteriza por avanzar de forma secuencial a través de etapas bien definidas, donde cada fase depende de la finalización completa de la anterior. A continuación, se detalla el proceso seguido.

- **Planeación y Coordinación Inicial (8-10 octubre)**

En esta fase se delimitaron los objetivos del proyecto se eligió el espacio a recrear (la casa de Marceline), y se organizaron os tiempos para las entregas. Se estableció una línea base de trabajo y se consideraron los requerimientos del laboratorio.

- **Modelado 3D (15 - 25 de octubre)**

En esta fase se dio inicio al desarrollo técnico, comenzando con el modelado de los componentes clave del entorno. Se incluyó el patio, que alberga la cancha de baloncesto, así como las lucarnas sobre el techo de la casa y las celosías ubicadas debajo de la estructura. Además, se modelaron detalles importantes como las puertas, las alas y las escaleras de la entrada principal. En cuanto al interior, se diseñaron elementos decorativos que aportarían un ambiente más acogedor, como los sillones, la banquita del patio, y las sillas, bancos y mesas, creando así un espacio más amigable y hogareño. Finalmente, se desarrolló el modelo principal de la casa, asegurando que todos estos elementos se integraran de manera coherente y funcional.

- **Texturizado y materiales (26 octubre – 1 de noviembre)**

Una vez completado el modelado, se procedió a aplicar las texturas necesarias para lograr un ambiente que fuera a la vez animado y gótico, fiel a la estética presentada en

la serie. Para ello, se utilizaron materiales específicos y mapas de colores que ayudaron a resaltar el carácter único de la casa, manteniendo la atmósfera característica de la obra.

- **Programación de interacción en OpenGL (2 – 5 de noviembre)**

Estos últimos puntos fueron de los más complicados de lograr. En esta etapa se presentaron varios inconvenientes, como problemas para que las texturas se leyeran correctamente al momento de exportar desde Blender, ya sea por rutas incorrectas o por errores generados por algunas texturas. Además, fue en este punto donde se implementaron los controles de cámara sintética, la carga de shaders y los mecanismos de navegación del usuario, permitiendo que éste pudiera explorar el entorno de manera interactiva.

- **Animaciones e iluminación (5-10 de noviembre)**

En esta parte se desarrollo la parte de iluminaciones tanto en la cámara como en el ambiente, idea que en mucha parte se elimino para respetar el estilo de animación de la serie, donde casi todo es parejo en cuestiones de iluminación dentro de la casa. En esta parte se desarrollaron las animaciones de los objetos, tanto las simples como las complejas, dentro de esta apartado se considera la investigación de animación por huesos y entender como poder poner ciertas físicas a algunos componentes dentro del entorno para simular interacción y colisión entre objetos, a su vez se implementa y se comprende como hacer retardos dentro de las animaciones, poner componentes de velocidad y hacer tanto rotación como traslación a la vez.

- **Documentación (11-13 de noviembre)**

Aquí se redacta el manual de usuario y la presente documento, se prepara el video explicativo y se implemento el empaquetado el ejecutable en su versión reléase, cosa que fallo por un par de errores extraños dentro del archivo y no deja que funcione.

- **Revisión de errores o ajustes extras (14 – 15 noviembre)**

En esta etapa vi los errores posibles que tuviera el código o corrección de los modelos, así como errores y complementación de la documentación.

- **Entrega final (15 de noviembre)**

El proyecto se entrega en un repositorio de GitHub que incluyo todo el código fuente, modelos, ejecutable, y documentación, conforme a los lineamientos establecidos por el profesor

Esta metodología permitió una aplicación funcional que cumple con los objetivos educativos del proyecto, a pesar de las limitantes de tiempo y de los recursos. La combinación de principios agiles con un enfoque iterativo resulto adecuada para un proyecto académico de esta escala.

Documentación de código:

A continuación, se presenta la documentación por bloques del código fuente, explicando la funcionalidad de cada sección, el uso de librerías externas, la lógica de renderizado, las técnicas de sombreado aplicadas, así como el manejo de eventos e interacción del usuario**.**

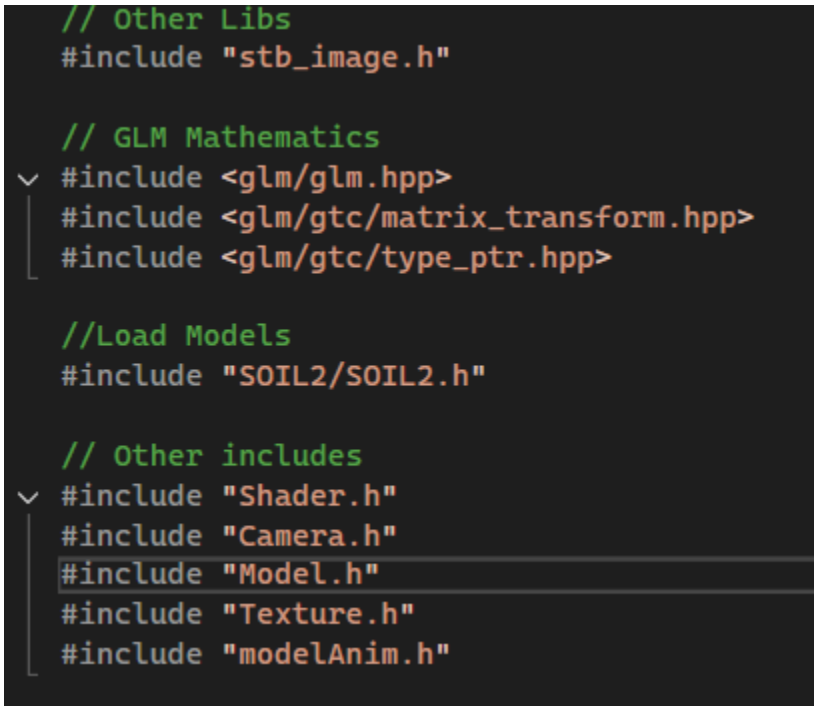
1. Inclusión de Bibliotecas y Cabeceras

Se incluyen las bibliotecas estándar de C++ y las librerías esenciales para el desarrollo gráfico moderno en OpenGL.

Se incluyen:

- Librerías Básicas: iostream y cmath para operaciones de entrada/salida y funciones matemáticas.
- OpenGL: GLEW (GL/glew.h) para cargar las extensiones y funciones de OpenGL, y GLFW (GLFW/glfw3.h) para la gestión de ventanas y la interacción del usuario.
- Utilidades de Carga: stb_image.h y SOIL2 (SOIL2/SOIL2.h) se utilizan para cargar datos de imágenes para texturas y modelos.
- GLM: La librería GLM (glm/glm.hpp, glm/gtc/matrix_transform.hpp, etc.) proporciona las herramientas de matemáticas de gráficos esenciales (vectores, matrices y transformaciones).
- Clases Personalizadas: Archivos de cabecera como Shader.h, Camera.h, Model.h y modelAnim.h encapsulan la lógica para manejar *shaders*, la

cámara, y la carga y representación de modelos estáticos y animados por huesos.



2. Prototipos de Funciones y Variables Globales

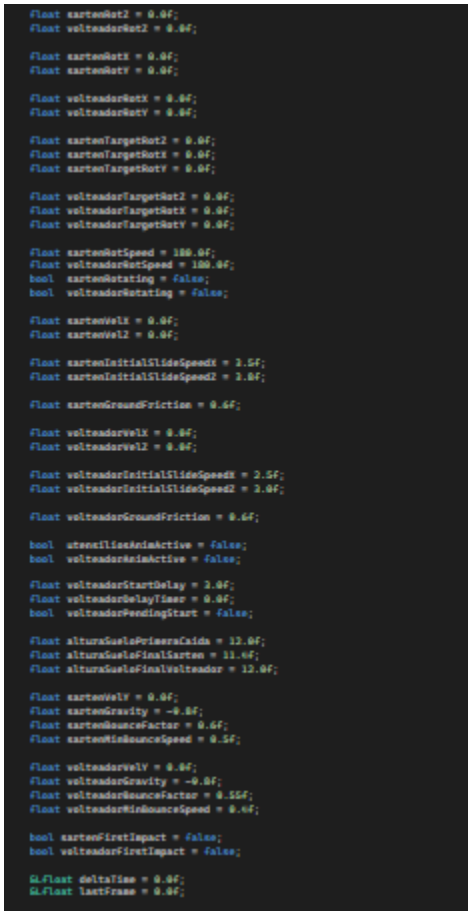
Se definen las funciones *callback* para el manejo de eventos, las funciones principales de actualización, y todas las variables globales que mantienen el estado del programa (cámara, luces, entradas y parámetros de animación).

- **Cámara y Control:** Se inicializa el objeto Camera con una posición específica. Las variables keys[], lastX, lastY y firstMouse gestionan la entrada para la navegación en primera persona.
- **Iluminación:** Se definen los vectores glm::vec3 para las posiciones de las luces puntuales (pointLightPositions[]) y una variable Light1 que controla el color de la luz puntual con el tiempo.
- **Animación:** Se declaran variables *float* y *bool* para controlar el movimiento de los modelos (cuadro, puerta, cajón, sartén y volteador), incluyendo velocidades, distancias de apertura, factores de física (gravedad, fricción, rebote) y ángulos de rotación.

```
// 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();

// Window dimensions
const GLuint WIDTH = 1800, HEIGHT = 1600;
int SCREEN_WIDTH, SCREEN_HEIGHT;

// Camera
Camera camera(glm::vec3(80.0f, 20.0f, -50.0f),
    glm::vec3(0.0f, 1.0f, 0.0f),
    180.0f,
    0.0f);

GLfloat lastX = WIDTH / 2.0;
GLfloat lastY = HEIGHT / 2.0;
bool keys[1024];
bool firstMouse = true;
```

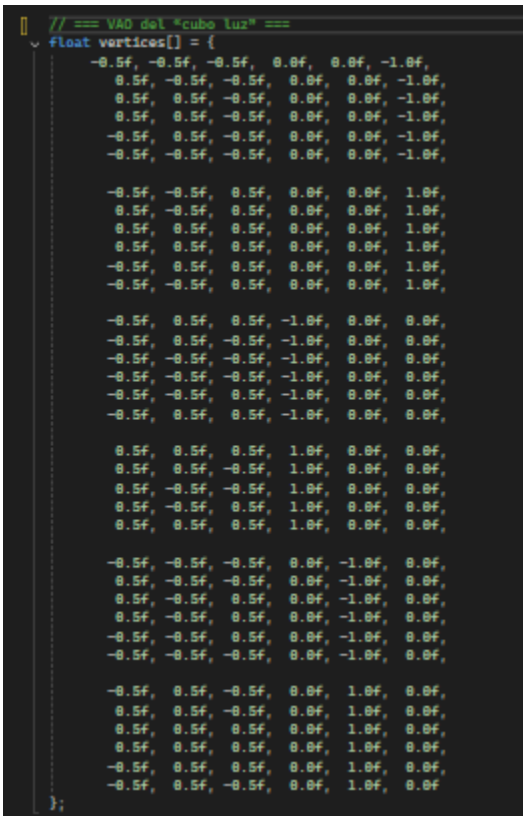
```
// Light attributes
glm::vec3 lightPos(0.0f, 0.0f, 0.0f);
bool active;

// Positions of the point lights
glm::vec3 pointLightPositions[] = {
    glm::vec3(0.0f,2.0f, 0.0f),
    glm::vec3(0.0f,0.0f, 0.0f),
    glm::vec3(0.0f,0.0f,  0.0f),
    glm::vec3(0.0f,0.0f, 0.0f)
};
```

```
// --- Cuadro ladeado + animación de acomodo ---
float cuadroAngle = 8.0f;
bool  cuadroReturn = false;
float cuadroDegPerSec = 60.0f;
glm::vec3 cuadroTopLocal = glm::vec3(2.0f, 40.0f, 10.0f);

// --- Puerta (apertura/cierre con P) ---
float puertaAngle = 0.0f;
float puertaTargetAngle = 0.0f;
float puertaOpenDeg = 90.0f;
float puertaDegPerSec = 90.0f;
bool  puertaAnimating = false;
glm::vec3 puertaHingeLocal = glm::vec3(32.5f, -2.0f, -48.7f);

// --- Cajón (traslación con O) ---
float cajonOffset = 0.0f;
float cajonTarget = 0.0f;
float cajonOpenDist = 3.0f;
float cajonSpeed = 0.8f;
bool  cajonAnimating = false;
glm::vec3 cajonDirLocal = glm::vec3(-1.0f, 0.0f, 0.0f);

// --- SARTÉN + VOLTEADOR ---
glm::vec3 sartenPosInitLocal = glm::vec3(-33.4f, 30.6f, -68.65f);
glm::vec3 volteadorPosInitLocal = glm::vec3(-29.5f, 32.6f, -68.0f);

glm::vec3 sartenPosLocal = sartenPosInitLocal;
glm::vec3 volteadorPosLocal = volteadorPosInitLocal;

glm::vec3 sartenPivotLocalAire = glm::vec3(0.0f, 0.0f, 0.0f);
glm::vec3 sartenPivotLocalSuelo = glm::vec3(0.0f, -0.8f, 0.0f);
glm::vec3 sartenPivotLocalActual = sartenPivotLocalAire;

glm::vec3 volteadorPivotLocalAire = glm::vec3(0.0f, 0.0f, 0.0f);
glm::vec3 volteadorPivotLocalSuelo = glm::vec3(0.0f, -0.5f, 0.0f);
glm::vec3 volteadorPivotLocalActual = volteadorPivotLocalAire;
```

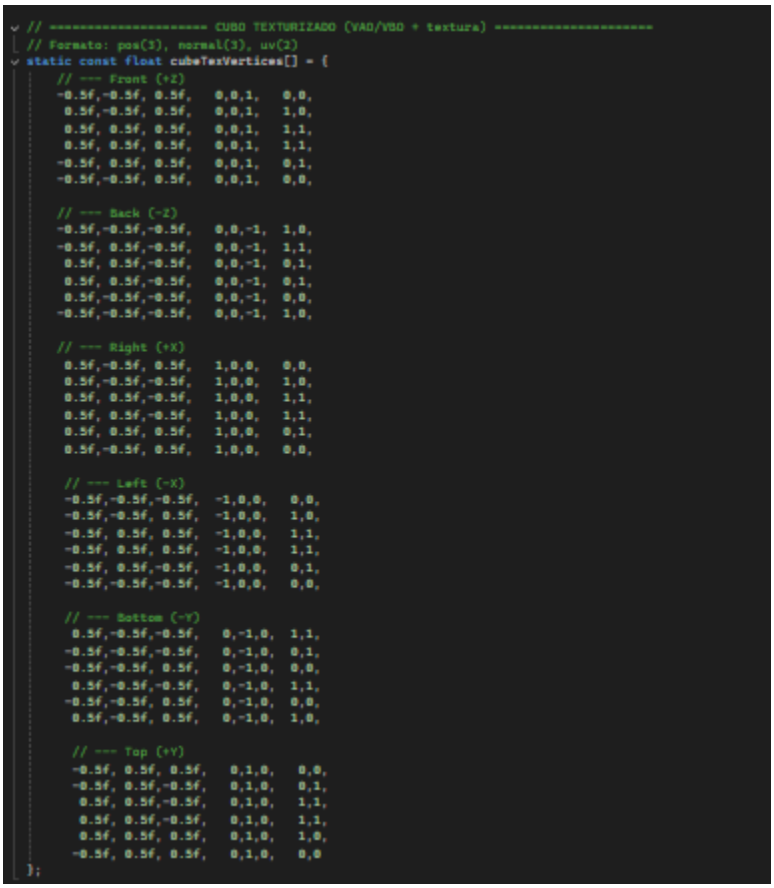


3. VAO/VBO y Geometría de Cubos

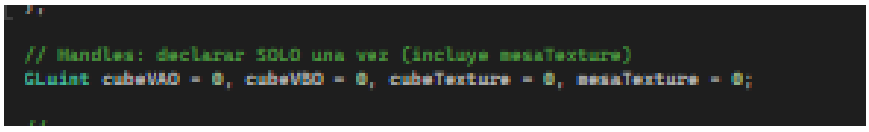
Se definen los arreglos de vértices globales para los cubos utilizados como indicadores de luz y como primitivas texturizadas.



```
// === VAO del "cubo luz" ===
float vertices[] = {
    -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
     0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
     0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
     0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
    -0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
    -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,

    -0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
     0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
     0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
     0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
    -0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
    -0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f,

    -0.5f,  0.5f,  0.5f, -1.0f,  0.0f,  0.0f,
    -0.5f,  0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
    -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
    -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
    -0.5f, -0.5f,  0.5f, -1.0f,  0.0f,  0.0f,
    -0.5f,  0.5f,  0.5f, -1.0f,  0.0f,  0.0f,

     0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f,
     0.5f,  0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
     0.5f, -0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
     0.5f, -0.5f,  0.5f,  1.0f,  0.0f,  0.0f,
     0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f,

    -0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,
     0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,
     0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
     0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
    -0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
    -0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,

    -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,
     0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,
     0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
     0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
    -0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
    -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f
};
```



```
// ==================== CUBO TEXTURIZADO (VAO/VBO + textura) ====================
// Formato: pos(3), normal(3), uv(2)
static const float cubeTexVertices[] = {
    // --- Front (+Z)
    -0.5f,-0.5f, 0.5f,   0,0,1,   0,0,
     0.5f,-0.5f, 0.5f,   0,0,1,   1,0,
     0.5f, 0.5f, 0.5f,   0,0,1,   1,1,
     0.5f, 0.5f, 0.5f,   0,0,1,   1,1,
    -0.5f, 0.5f, 0.5f,   0,0,1,   0,1,
    -0.5f,-0.5f, 0.5f,   0,0,1,   0,0,

    // --- Back (-Z)
    -0.5f,-0.5f,-0.5f,   0,0,-1,  1,0,
    -0.5f, 0.5f,-0.5f,   0,0,-1,  1,1,
     0.5f, 0.5f,-0.5f,   0,0,-1,  0,1,
     0.5f, 0.5f,-0.5f,   0,0,-1,  0,1,
     0.5f,-0.5f,-0.5f,   0,0,-1,  0,0,
    -0.5f,-0.5f,-0.5f,   0,0,-1,  1,0,

    // --- Right (+X)
     0.5f,-0.5f, 0.5f,   1,0,0,   0,0,
     0.5f,-0.5f,-0.5f,   1,0,0,   1,0,
     0.5f, 0.5f,-0.5f,   1,0,0,   1,1,
     0.5f, 0.5f,-0.5f,   1,0,0,   1,1,
     0.5f, 0.5f, 0.5f,   1,0,0,   0,1,
     0.5f,-0.5f, 0.5f,   1,0,0,   0,0,

    // --- Left (-X)
    -0.5f,-0.5f,-0.5f,  -1,0,0,   0,0,
    -0.5f,-0.5f, 0.5f,  -1,0,0,   1,0,
    -0.5f, 0.5f, 0.5f,  -1,0,0,   1,1,
    -0.5f, 0.5f, 0.5f,  -1,0,0,   1,1,
    -0.5f, 0.5f,-0.5f,  -1,0,0,   0,1,
    -0.5f,-0.5f,-0.5f,  -1,0,0,   0,0,

    // --- Bottom (-Y)
     0.5f,-0.5f,-0.5f,   0,-1,0,  1,1,
    -0.5f,-0.5f,-0.5f,   0,-1,0,  0,1,
    -0.5f,-0.5f, 0.5f,   0,-1,0,  0,0,
     0.5f,-0.5f,-0.5f,   0,-1,0,  1,1,
    -0.5f,-0.5f, 0.5f,   0,-1,0,  0,0,
     0.5f,-0.5f, 0.5f,   0,-1,0,  1,0,

    // --- Top (+Y)
    -0.5f, 0.5f, 0.5f,   0,1,0,   0,0,
    -0.5f, 0.5f,-0.5f,   0,1,0,   0,1,
     0.5f, 0.5f,-0.5f,   0,1,0,   1,1,
     0.5f, 0.5f,-0.5f,   0,1,0,   1,1,
     0.5f, 0.5f, 0.5f,   0,1,0,   1,0,
    -0.5f, 0.5f, 0.5f,   0,1,0,   0,0
};
```



```
// Handles: declarar SOLO una vez (incluye mesaTexture)
GLuint cubeVAO = 0, cubeVBO = 0, cubeTexture = 0, mesaTexture = 0;
```

4. Función main() - Inicialización y Ciclo de Renderizado

La función main() es el punto de entrada que inicializa el sistema, carga los recursos, y contiene el bucle principal de la aplicación.

4.1. Configuración del Contexto Gráfico y Carga de Shaders

Se inicializa GLFW, se crea la ventana y se configura GLEW. Se cargan los tres *shaders* clave: lightingShader (para la mayoría de los objetos), lampShader (para la fuente de luz) y animShader (para el modelo esquelético animado). Luego se cargan todos los modelos estáticos y animados, y se configuran los VAOs/VBOs para los cubos.

4.2. Ciclo de Juego (Game Loop)

El bucle while (!glfwWindowShouldClose(window)) se ejecuta en cada *frame*.

Actualización de Tiempo: Se calcula deltaTime para asegurar la consistencia del movimiento.

Entrada y Animación: Se llaman a glfwPollEvents(), DoMovement() y Animation().

Configuración de Luces: Se activa lightingShader y se envían las propiedades de iluminación.

- La luz puntual (pointLights[0]) se configura con un color que varía con la función seno del tiempo (std::abs(std::sin(glfwGetTime() * Light1.x))) para crear un efecto pulsante.
- El foco de luz (spotLight) se vincula a la posición y dirección de la cámara, simulando una linterna.

Dibujo de Modelos: Se dibuja la escena completa: cubos texturizados, modelos estáticos (casa, muebles, etc.), objetos animados (puerta, cajón, cuadro, sartén, volteador), y el modelo Marceline (animado por huesos).

Indicador de Luz: Se activa lampShader para dibujar el cubo simple que representa la fuente de luz puntual.

Dentro del ciclo while, se calcula el deltaTime y se llaman las funciones de entrada y animación. Posteriormente, se configura la iluminación, donde la luz puntual (pointLights[0]) tiene un color que varía en el tiempo (std::abs(std::sin(glfwGetTime()))) y el *spotlight* sigue la posición y dirección de la cámara. Finalmente, se dibujan todos los modelos y primitiva

```cpp
int main()
{
    // Init GLFW
    glfwInit();

    GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "Animacion maquina de estados", nullptr, nullptr);

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

    glfwMakeContextCurrent(window);
    glfwGetFramebufferSize(window, &SCREEN_WIDTH, &SCREEN_HEIGHT);

    // Callbacks
    glfwSetKeyCallback(window, KeyCallback);
    glfwSetCursorPosCallback(window, MouseCallback);

    glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);

    // GLEW
    glewExperimental = GL_TRUE;
    if (GLEW_OK != glewInit())
    {
        std::cout << "Failed to initialize GLEW" << std::endl;
        return EXIT_FAILURE;
    }

    // Viewport
    glViewport(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT);

    // Shaders
    Shader lightingShader("Shader/lighting.vs", "Shader/lighting.frag");
    Shader lampShader("Shader/lamp.vs", "Shader/lamp.frag");
    Shader animShader("Shader/anim.vs", "Shader/anim.frag");

    // Models estáticos
    Model casa((char*)"Models/casacompleta.obj");
    Model LamparaTecho((char*)"Models/lamparatecho.obj");
    Model Cuadro((char*)"Models/cuadro.obj");
    Model Puerta((char*)"Models/puerta.obj");
    Model Mueble((char*)"Models/mueble.obj");
    Model Cajon((char*)"Models/cajon.obj");

    // Nuevos modelos
    Model Sarten((char*)"Models/sarten.obj");
    Model Volteador((char*)"Models/volteador.obj");

    // Modelo animado por huesos
    ModelAnim Marceline((char*)"Models/marceline3.dae");
    Marceline.initShaders(animShader.Program);
```

```
// === VAO/VBO para cubo luz (ya existente) ===
GLuint VBO, VAO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);

glBindVertexArray(VAO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

// Posición
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(0);
// Normal
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 * sizeof(float)));
glEnableVertexAttribArray(1);

glBindVertexArray(0);

// Material para lightingShader (como lo tenías)
lightingShader.Use();
glUniform1i(glGetUniformLocation(lightingShader.Program, "Material.difuse"), 0);
glUniform1i(glGetUniformLocation(lightingShader.Program, "Material.specular"), 1);

glm::mat4 projection = glm::perspective(
    camera.GetZoom(),
    (GLfloat)SCREEN_WIDTH / (GLfloat)SCREEN_HEIGHT,
    0.1f,
    100.0f
);

// === VAO/VBO del cubo texturizado ===
glGenVertexArrays(1, &cubeVAO);
glGenBuffers(1, &cubeVBO);

glBindVertexArray(cubeVAO);
glBindBuffer(GL_ARRAY_BUFFER, cubeVBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(cubeTexVertices), cubeTexVertices, GL_STATIC_DRAW);

// Atributos: pos(0), normal(1), texcoord(2)
GLsizei stride = 8 * sizeof(float);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, stride, (void*)0);
glEnableVertexAttribArray(0);

glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, stride, (void*)(3 * sizeof(float)));
glEnableVertexAttribArray(1);

glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, stride, (void*)(6 * sizeof(float)));
glEnableVertexAttribArray(2);

glBindVertexArray(0);
```

```cpp
// === textura Models/lampara2.png ===
{
    int w, h, channels;
    unsigned char* data = SOIL_load_image("Models/lampara2.png", &w, &h, &channels, SOIL_LOAD_AUTO);
    if (!data) {
        std::cout << "Error cargando textura Models/lampara2.png" << std::endl;
    }
    else {
        glGenTextures(1, &cubeTexture);
        glBindTexture(GL_TEXTURE_2D, cubeTexture);

        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

        GLint internalFormat = (channels == 4) ? GL_RGBA : GL_RGB;
        GLenum  format = (channels == 4) ? GL_RGBA : GL_RGB;

        glTexImage2D(GL_TEXTURE_2D, 0, internalFormat, w, h, 0, format, GL_UNSIGNED_BYTE, data);
        glGenerateMipmap(GL_TEXTURE_2D);

        glBindTexture(GL_TEXTURE_2D, 0);
        SOIL_free_image_data(data);
    }
}

// === textura Models/mesa.png ===
{
    int w, h, channels;
    unsigned char* data = SOIL_load_image("Models/mesa.png", &w, &h, &channels, SOIL_LOAD_AUTO);
    if (!data) {
        std::cout << "Error cargando textura Models/mesa.png" << std::endl;
    }
    else {
        glGenTextures(1, &mesaTexture);
        glBindTexture(GL_TEXTURE_2D, mesaTexture);

        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LI                AR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

        GLint internalFormat = (channels == 4) ? GL_RGBA : GL_RGB;
        GLenum  format = (channels == 4) ? GL_RGBA : GL_RGB;

        glTexImage2D(GL_TEXTURE_2D, 0, internalFormat, w, h, 0, format, GL_UNSIGNED_BYTE, data);
        glGenerateMipmap(GL_TEXTURE_2D);

        glBindTexture(GL_TEXTURE_2D, 0);
        SOIL_free_image_data(data);
    }
}
```

```
// Game loop
while (!glfwWindowShouldClose(window))
{
    // DeltaTime
    GLfloat currentFrame = glfwGetTime();
    deltaTime = currentFrame - lastFrame;
    lastFrame = currentFrame;

    // Input
    glfwPollEvents();
    DoMovement();
    Animation();

    // Clear
    glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // OpenGL options
    glEnable(GL_DEPTH_TEST);

    // === SHADER DE ILUMINACIÓN ===
    lightingShader.Use();

    glUniform1i(glGetUniformLocation(lightingShader.Program, "diffuse"), 0);

    GLint viewPosLoc = glGetUniformLocation(lightingShader.Program, "viewPos");
    glUniform3f(viewPosLoc, camera.GetPosition().x, camera.GetPosition().y, camera.GetPosition().z);

    // Dir light
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.6f, 0.6f, 0.6f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.6f, 0.6f, 0.6f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.3f, 0.3f, 0.3f);

    // Point light 1
    glm::vec3 lightColor;
    lightColor.x = std::abs(std::sin(glfwGetTime() * Light1.x));
    lightColor.y = std::abs(std::sin(glfwGetTime() * Light1.y));
    lightColor.z = std::abs(std::sin(glfwGetTime() * Light1.z)); // + evita negativos

    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"),
        pointLightPositions[0].x, pointLightPositions[0].y, pointLightPositions[0].z);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"),
        lightColor.x, lightColor.y, lightColor.z);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"),
        lightColor.x, lightColor.y, lightColor.z);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"),
        1.0f, 0.2f, 0.2f);
    glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].constant"), 1.0f);
    glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].linear"), 0.045f);
    glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"), 0.075f);

    // SpotLight
    glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.position"),
        camera.GetPosition().x, camera.GetPosition().y, camera.GetPosition().z);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.direction"),
        camera.GetFront().x, camera.GetFront().y, camera.GetFront().z);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.ambient"), 0.2f, 0.2f, 0.8f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.diffuse"), 0.2f, 0.2f, 0.8f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.specular"), 0.0f, 0.0f, 0.0f);
    glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.constant"), 1.0f);
    glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.linear"), 0.3f);
    glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.quadratic"), 0.7f);
    glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.cutOff"), glm::cos(glm::radians(12.0f)));
    glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.outerCutOff"), glm::cos(glm::radians(18.0f)));

    glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 5.0f);

    glm::mat4 view = camera.GetViewMatrix();
```

```
// ==================== DIBUJAR CUBO(S) TEXTURIZADO(S) ====================
{
    // Sampler en la unidad 0 (compatibilidad con tu shader)
    glActiveTexture(GL_TEXTURE0);

    // Por si tu shader usa "diffuse" o "material.diffuse"
    glUniform1i(glGetUniformLocation(lightingShader.Program, "diffuse"), 0);
    GLint matDiff = glGetUniformLocation(lightingShader.Program, "material.diffuse");
    if (matDiff >= 0) glUniform1i(matDiff, 0);

    glUniform1i(glGetUniformLocation(lightingShader.Program, "transparency"), 0);

    glBindVertexArray(cubeVAO);

    // --- CUBO 1: lampara2 ---
    glBindTexture(GL_TEXTURE_2D, cubeTexture);
    glm::mat4 m1(1.0f);
    m1 = glm::translate(m1, glm::vec3(20.0f, 17.0f, -71.0f));
    m1 = glm::scale(m1, glm::vec3(15.0f, 15.0f, 3.0f));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(m1));
    glDrawArrays(GL_TRIANGLES, 0, 36);

    // --- CUBO 2: lampara2 ---
    glBindTexture(GL_TEXTURE_2D, cubeTexture);
    glm::mat4 m2(1.0f);
    m2 = glm::translate(m2, glm::vec3(20.0f, 12.5f, -67.0f));
    m2 = glm::scale(m2, glm::vec3(15.0f, 6.0f, 6.0f));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(m2));
    glDrawArrays(GL_TRIANGLES, 0, 36);

    // --- CUBO 3: lampara2 (pilar izq) ---
    glBindTexture(GL_TEXTURE_2D, cubeTexture);
    glm::mat4 m3(1.0f);
    m3 = glm::translate(m3, glm::vec3(12.0f, 14.0f, -67.0f));
    m3 = glm::scale(m3, glm::vec3(3.0f, 9.5f, 6.0f));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(m3));
    glDrawArrays(GL_TRIANGLES, 0, 36);
```

```
// --- CASA ---
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(2.0f, 0.0f, -3.0f));
model = glm::rotate(model, glm::radians(0.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(1.0f));

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

// --- LÁMPARA DE TECHO ---
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(casaModel));
glUniform1i(glGetUniformLocation(lightingShader.Program, "transparency"), 0);
LamparaTecho.Draw(lightingShader);

// --- MUEBLE ---
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(casaModel));
glUniform1i(glGetUniformLocation(lightingShader.Program, "transparency"), 0);
Mueble.Draw(lightingShader);

// --- CAJÓN ---
{
    glm::mat4 cajonLocal = glm::translate(glm::mat4(1.0f), cajonDirLocal * cajonOffset);
    glm::mat4 modelCajon = casaModel * cajonLocal;
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(modelCajon));
    glUniform1i(glGetUniformLocation(lightingShader.Program, "transparency"), 0);
    Cajon.Draw(lightingShader);
}

// --- PUERTA ---
{
    glm::vec4 hingeWorld = casaModel * glm::vec4(puertaHingeLocal, 1.0f);

    glm::mat4 puertaLocal = glm::mat4(1.0f);
    puertaLocal = glm::translate(puertaLocal, puertaHingeLocal);
    puertaLocal = glm::rotate(puertaLocal, glm::radians(puertaAngle), glm::vec3(0.0f, 1.0f, 0.0f));
    puertaLocal = glm::translate(puertaLocal, -puertaHingeLocal);

    glm::mat4 modelPuerta = casaModel * puertaLocal;
    glm::vec4 newHingeWorld = modelPuerta * glm::vec4(puertaHingeLocal, 1.0f);

    glm::vec3 deltaH = glm::vec3(hingeWorld - newHingeWorld);
    modelPuerta = glm::translate(glm::mat4(1.0f), deltaH) * modelPuerta;

    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(modelPuerta));
    glUniform1i(glGetUniformLocation(lightingShader.Program, "transparency"), 0);
    Puerta.Draw(lightingShader);
}

// --- CUADRO ---
{
    glm::vec4 anchorWorld = casaModel * glm::vec4(cuadroTopLocal, 1.0f);
    glm::mat4 cuadroLocal = glm::mat4(1.0f);
    cuadroLocal = glm::translate(cuadroLocal, cuadroTopLocal);
    cuadroLocal = glm::rotate(cuadroLocal, glm::radians(cuadroAngle), glm::vec3(0.0f, 0.0f, 1.0f));
    cuadroLocal = glm::translate(cuadroLocal, -cuadroTopLocal);

    glm::mat4 modelCuadro = casaModel * cuadroLocal;
    glm::vec4 newAnchorWorld = modelCuadro * glm::vec4(cuadroTopLocal, 1.0f);
    glm::vec3 delta = glm::vec3(anchorWorld - newAnchorWorld);
    modelCuadro = glm::translate(glm::mat4(1.0f), delta) * modelCuadro;

    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(modelCuadro));
    glUniform1i(glGetUniformLocation(lightingShader.Program, "transparency"), 0);
    Cuadro.Draw(lightingShader);
}
```

```
// --- SARTÉN & VOLTEADOR ---
{
    // Sartén
    glm::mat4 modelSarten = casaModel;
    modelSarten = glm::translate(modelSarten, sartenPosLocal);
    modelSarten = glm::translate(modelSarten, sartenPivotLocalActual);
    modelSarten = glm::rotate(modelSarten, glm::radians(sartenRotX), glm::vec3(1.0f, 0.0f, 0.0f));
    modelSarten = glm::rotate(modelSarten, glm::radians(sartenRotY), glm::vec3(0.0f, 1.0f, 0.0f));
    modelSarten = glm::rotate(modelSarten, glm::radians(sartenRotZ), glm::vec3(0.0f, 0.0f, 1.0f));
    modelSarten = glm::translate(modelSarten, -sartenPivotLocalActual);
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(modelSarten));
    glUniform1i(glGetUniformLocation(lightingShader.Program, "transparency"), 0);
    Sarten.Draw(lightingShader);

    // Volteador
    glm::mat4 modelVolteador = casaModel;
    modelVolteador = glm::translate(modelVolteador, volteadorPosLocal);
    modelVolteador = glm::translate(modelVolteador, volteadorPivotLocalActual);
    modelVolteador = glm::rotate(modelVolteador, glm::radians(volteadorRotX), glm::vec3(1.0f, 0.0f, 0.0f)
    modelVolteador = glm::rotate(modelVolteador, glm::radians(volteadorRotY), glm::vec3(0.0f, 1.0f, 0.0f)
    modelVolteador = glm::rotate(modelVolteador, glm::radians(volteadorRotZ), glm::vec3(0.0f, 0.0f, 1.0f)
    modelVolteador = glm::translate(modelVolteador, -volteadorPivotLocalActual);
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(modelVolteador));
    glUniform1i(glGetUniformLocation(lightingShader.Program, "transparency"), 0);
    Volteador.Draw(lightingShader);
}

// --- MARCELINE (huesos) ---
{
    animShader.Use();

    GLint aModelLoc = glGetUniformLocation(animShader.Program, "model");
    GLint aViewLoc = glGetUniformLocation(animShader.Program, "view");
    GLint aProjLoc = glGetUniformLocation(animShader.Program, "projection");

    glUniformMatrix4fv(aViewLoc, 1, GL_FALSE, glm::value_ptr(view));
    glUniformMatrix4fv(aProjLoc, 1, GL_FALSE, glm::value_ptr(projection));

    glUniform3f(glGetUniformLocation(animShader.Program, "light.direction"), -0.2f, -1.0f, -0.3f);
    glUniform3f(glGetUniformLocation(animShader.Program, "light.ambient"), 0.6f, 0.6f, 0.6f);
    glUniform3f(glGetUniformLocation(animShader.Program, "light.diffuse"), 0.6f, 0.6f, 0.6f);
    glUniform3f(glGetUniformLocation(animShader.Program, "light.specular"), 0.3f, 0.3f, 0.3f);

    glUniform3f(glGetUniformLocation(animShader.Program, "viewPos"),
        camera.GetPosition().x,
        camera.GetPosition().y,
        camera.GetPosition().z);

    glUniform3f(glGetUniformLocation(animShader.Program, "material.specular"), 0.5f, 0.5f, 0.5f);
    glUniform1f(glGetUniformLocation(animShader.Program, "material.shininess"), 16.0f);

    glm::mat4 modelMarceline = glm::mat4(1.0f);
    modelMarceline = glm::translate(modelMarceline, glm::vec3(6.0f, 10.0f, -5.0f));
    modelMarceline = glm::scale(modelMarceline, glm::vec3(14.0f));

    glUniformMatrix4fv(aModelLoc, 1, GL_FALSE, glm::value_ptr(modelMarceline));
    Marceline.Draw(animShader);
}
```

5. Funciones de Interacción del Usuario y Animación

5.1 Movement(): Control de Movimiento

Esta función se llama en cada *frame* para procesar el movimiento de la cámara y la luz. La cámara utiliza deltaTime para un movimiento consistente (velocidad en unidades/segundo), mientras que la luz se mueve mediante incrementos fijos por *frame*.

```cpp
// Moves/alters the camera positions based on user input
void DoMovement()
{
    if (keys[GLFW_KEY_W] || keys[GLFW_KEY_UP]) { camera.ProcessKeyboard(FORWARD, deltaTime); }
    if (keys[GLFW_KEY_S] || keys[GLFW_KEY_DOWN]) { camera.ProcessKeyboard(BACKWARD, deltaTime); }
    if (keys[GLFW_KEY_A] || keys[GLFW_KEY_LEFT]) { camera.ProcessKeyboard(LEFT, deltaTime); }
    if (keys[GLFW_KEY_D] || keys[GLFW_KEY_RIGHT]) { camera.ProcessKeyboard(RIGHT, deltaTime); }

    // Luz
    if (keys[GLFW_KEY_T]) { pointLightPositions[0].x += 0.01f; }
    if (keys[GLFW_KEY_G]) { pointLightPositions[0].x -= 0.01f; }
    if (keys[GLFW_KEY_Y]) { pointLightPositions[0].y += 0.01f; }
    if (keys[GLFW_KEY_H]) { pointLightPositions[0].y -= 0.01f; }
    if (keys[GLFW_KEY_U]) { pointLightPositions[0].z -= 0.01f; }
    if (keys[GLFW_KEY_J]) { pointLightPositions[0].z += 0.01f; }

    // Movimiento vertical de la cámara
    if (keys[GLFW_KEY_SPACE]) { camera.ProcessKeyboard(UP, deltaTime); }
    if (keys[GLFW_KEY_LEFT_SHIFT] || keys[GLFW_KEY_RIGHT_SHIFT]) { camera.ProcessKeyboard(DOWN, deltaTime); }
}
```

5.2 keyCallback(): Eventos Únicos

Procesa la lógica que se activa al presionar o soltar una tecla. Es el punto de control para iniciar las animaciones de estado.

- **'C', 'P', 'O':** Establecen el objetivo de movimiento (targetAngle, cajonTarget) y activan la bandera Animating para iniciar la interpolación en Animation().
- **'K':** Resetea las posiciones y velocidades, e inicia la animación de física del sartén y el volteador.

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

if (keys[GLFW_KEY_SPACE])
{
    active = !active;
    if (active)
        Light1 = glm::vec3(0.2f, 0.8f, 1.0f);
    else
        Light1 = glm::vec3(0.0f);
}
```

5.3 Animation(): Lógica de Máquina de Estados y Física

Esta función implementa la lógica de movimiento paso a paso, asegurando que todas las transformaciones se realicen en función de deltaTime para una velocidad constante.

Animaciones por Interpolación Lineal/Angular

Las animaciones de la puerta, cajón y el enderezamiento del cuadro (cuadroReturn) se basan en una interpolación suave. El valor actual se mueve hacia el valor objetivo a una tasa definida por sus respectivas variables de velocidad.

Física de Rebote (Sartén y Volteador)

Esta es una simulación de física básica que utiliza el tiempo (deltaTime) para la integración de la velocidad y la posición.

- Caída: La gravedad (sartenGravity) acelera la velocidad vertical (sartenVelY).
- Impacto: Al golpear el suelo, la velocidad vertical se invierte y se reduce por el BounceFactor (controla la altura del rebote).
- Rotación de Impacto: En el primer impacto, se activan las rotaciones objetivo (sartenTargetRotZ/X/Y), y la rotación se interpola suavemente usando sartenRotSpeed.
- Fricción: La velocidad horizontal se reduce por el GroundFriction después de cada rebote.
- Fin: La animación de física se detiene cuando la velocidad vertical es menor que sartenMinBounceSpeed.

```
void Animation()
{
    // Cuadro
    if (cuadroReturn) {
        if (cuadroAngle > 0.0f) {
            cuadroAngle -= cuadroDegPerSec * deltaTime;
            if (cuadroAngle < 0.0f) cuadroAngle = 0.0f;
        }
        else {
            cuadroReturn = false;
        }
    }

    // Puerta
    if (puertaAnimating) {
        float diff = puertaTargetAngle - puertaAngle;
        float step = puertaDegPerSec * deltaTime;
        if (std::abs(diff) <= step) {
            puertaAngle = puertaTargetAngle;
            puertaAnimating = false;
        }
        else {
            puertaAngle += (diff > 0.0f ? step : -step);
        }
    }

    // Cajón
    if (cajonAnimating) {
        float diff = cajonTarget - cajonOffset;
        float step = cajonSpeed * deltaTime;
        if (std::abs(diff) <= step) {
            cajonOffset = cajonTarget;
            cajonAnimating = false;
        }
        else {
            cajonOffset += (diff > 0.0f ? step : -step);
        }
    }
```

```
// Sartén: movimiento con rebote
if (utensiliosAnimActive)
{
    sartenVelY += sartenGravity * deltaTime;
    sartenPosLocal.y += sartenVelY * deltaTime;

    sartenPosLocal.x += sartenVelX * deltaTime;
    sartenPosLocal.z += sartenVelZ * deltaTime;

    float sueloActualY = (!sartenFirstImpact ? alturaSueloPrimeraCaida : alturaSueloFinalSarten);

    if (sartenPosLocal.y <= sueloActualY)
    {
        sartenPosLocal.y = sueloActualY;

        if (!sartenFirstImpact)
        {
            sartenTargetRotZ = sartenRotZ + -90.0f;
            sartenTargetRotX = sartenRotX + 90.0f;
            sartenTargetRotY = sartenRotY + -5.0f;

            sartenRotating = true;
            sartenPivotLocalActual = sartenPivotLocalSuelo;

            sartenVelX = sartenInitialSlideSpeedX;
            sartenVelZ = sartenInitialSlideSpeedZ;

            sartenFirstImpact = true;
        }

        sartenVelX *= sartenGroundFriction;
        sartenVelZ *= sartenGroundFriction;

        if (std::abs(sartenVelX) < 0.05f) sartenVelX = 0.0f;
        if (std::abs(sartenVelZ) < 0.05f) sartenVelZ = 0.0f;

        sartenVelY = -sartenVelY * sartenBounceFactor;

        if (std::abs(sartenVelY) < sartenMinBounceSpeed)
        {
            sartenVelY = 0.0f;
            utensiliosAnimActive = false;
            sartenPosLocal.y = alturaSueloFinalSarten;
        }
    }
}
```

```
// Rotación suave del sartén hacia los ángulos objetivo
if (sartenRotating)
{
    float step = sartenRotSpeed * deltaTime;

    float diffZ = sartenTargetRotZ - sartenRotZ;
    if (std::abs(diffZ) <= step) sartenRotZ = sartenTargetRotZ;
    else                         sartenRotZ += (diffZ > 0.0f ? step : -step);

    float diffX = sartenTargetRotX - sartenRotX;
    if (std::abs(diffX) <= step) sartenRotX = sartenTargetRotX;
    else                         sartenRotX += (diffX > 0.0f ? step : -step);

    float diffY = sartenTargetRotY - sartenRotY;
    if (std::abs(diffY) <= step) sartenRotY = sartenTargetRotY;
    else                         sartenRotY += (diffY > 0.0f ? step : -step);

    if (sartenRotZ == sartenTargetRotZ &&
        sartenRotX == sartenTargetRotX &&
        sartenRotY == sartenTargetRotY)
    {
        sartenRotating = false;
    }
}
```

5.4. MouseCallback(): Control de Orientación de Cámara

Esta función procesa el desplazamiento del ratón para actualizar la orientación de la cámara, permitiendo el control de la vista en primera persona.

```
void MouseCallback(GLFWwindow* window, double xPos, double yPos)
{
    if (firstMouse)
    {
        lastX = xPos;
        lastY = yPos;
        firstMouse = false;
    }

    GLfloat xOffset = xPos - lastX;
    GLfloat yOffset = lastY - yPos;

    lastX = xPos;
    lastY = yPos;

    camera.ProcessMouseMovement(xOffset, yOffset);
}
```

Conclusiones:

Durante el desarrollo de este proyecto pude adquirir y aplicar conocimientos tanto de modelado, iluminación y animación, en la etapa de modelado aprendí a utilizar de manera eficiente las herramientas que ofrece blender, desde las operaciones básicas como rotar, trasladar y escalar, hasta la creación de formas mas complejas, esto partiendo de las primitivas como de los cubos, cilindros y curvas, Bezier para estructuras como la parte trasera de la silla y columnas o redondeos para formar curvas y no solo planos.

Para lograr todo esto use modificadores como los booleanos para realizar cortes precisos en ciertas caras para hacer las ventanas y puertas y algunos otros como simetrizar y repetir para la parte de las celosías de la casa. A su vez comprendí la

importancia que tiene el mapeo UV al aplicar texturas, adaptando el método adecuado según el modelo. Una vez finalizado el modelado, se separaron los objetos por tipo de material para integrarlos adecuadamente a OpenGL dentro de esto ajuste la rugosidad y la especularidad.

Ya en OpenGL comprendí aun mejor la importancia de los shaders y su utilidad para cada cosa, tanto para colores, iluminación texturas y animaciones, entre otras cosas.

Ya en la etapa de animación se pudo integrar de manera adecuada y sencilla en algunos de los modelos hechos, como el cuadro, el cajón y la puerta, ya en lo que respecta a el sartén y el volteador subió un poco la complejidad porque había que poner mas estados para que la animación se viera lo mejor posible y diera la impresión de que tiene físicas y colisiones entre objetos y para la parte de la animación de Marceline haciendo la parte de huesos tuve que usar un programa externo para la creación de la animación y que se viera lo mejor posible, pero esto para blender, ya que al meterlo a OpenGL había que crear nuevos shaders para que se pudieran cargar las animaciones y se vieran las texturas adecuadamente.

Este proyecto represento una experiencia de aprendizaje completa donde logre consolidar conocimientos técnicos y creativos, además de que pude superar varias dificultades o retos presentados a lo largo del proyecto en varias de las etapas, especialmente de las mas complejas a mi parecer la integración del modelado hecho a ponerlo en OpenGL y la parte de animación por huesos. Esto cumple con mis expectativas del curso, el cual me ayuda con herramientas para mi desarrollo profesional.