Universidad Nacional Autónoma de México

Facultad de Ingeniería

Computación Gráfica e

Interacción Humano-Computadora

Grupo: 05

Profesor: Carlos Aldair Roman Balbuena

Proyecto 2: Manual Técnico

Integrantes:

- 319181386

Fecha de entrega: 25 de Noviembre de 2025

Semestre 2025-2

Tabla de contenido

Objetivos

Objetivo General

Crear un recorrido virtual interactivo en OpenGL 3, que permita al usuario explorar de forma realista dos habitaciones y una fachada tridimensional, integrando cámara sintética, animaciones, texturas y efectos de iluminación, con el fin de aplicar los principios de la computación gráfica y la interacción humano-computadora en un entorno 3D funcional y visualmente coherente.

Objetivos Específicos

- Modelar dos habitaciones y una fachada con estilo definido.
- Implementar una cámara sintética interactiva.
- Integrar cuatro animaciones en el entorno 3D.
- Aplicar iluminación y texturas realistas en OpenGL.
- Optimizar recursos y rutas del proyecto.
- Elaborar documentación técnica y de usuario bilingüe.
- Realizar un análisis de costos del proyecto.
- Publicar el proyecto funcional en GitHub.

Metas

- Crear un recorrido virtual funcional en OpenGL 3 con cámara sintética y animaciones.
- Lograr una ambientación visual realista en dos habitaciones y una fachada.
- Garantizar que el usuario pueda interactuar libremente con el entorno.
- Entregar un ejecutable estable y optimizado del proyecto.
- Documentar completamente el desarrollo, en español e inglés.
- Cumplir con todos los requerimientos técnicos y de presentación solicitados.

Alcances

- El proyecto permitirá recorrer dos habitaciones y una fachada en 3D.
- Incluirá una cámara sintética con controles de movimiento y rotación.
- Contará con al menos cinco animaciones interactivas.
- Aplicará materiales, texturas e iluminación tipo Phong para mayor realismo.
- Se documentará el desarrollo técnico, usuario y costos del proyecto.
- El código y ejecutable estarán disponibles en GitHub con rutas relativas.

Limitantes

Durante el desarrollo del proyecto se presentaron algunas limitaciones que influyeron en el proceso de creación y optimización del recorrido 3D. Entre las principales se encuentran:

1. **Capacidad del hardware:** El rendimiento del equipo afectó la velocidad de renderizado y la fluidez del recorrido, especialmente al manejar modelos con alta cantidad de polígonos y texturas detalladas.

2. **Tiempo de desarrollo:** El modelado, texturizado y programación requirieron más tiempo del previsto, lo que limitó la posibilidad de agregar más detalles o animaciones avanzadas.

3. **Conocimientos técnicos:** La curva de aprendizaje de OpenGL y el manejo de shaders representaron un desafío, especialmente al implementar iluminación y animaciones coordinadas.

4. **Compatibilidad de formatos:** En algunos casos, al exportar modelos desde Blender hacia OpenGL, surgieron problemas de escala, orientación o carga de texturas que requirieron ajustes manuales.

5. **Optimización de animaciones:** La integración de animaciones dentro del entorno 3D fue limitada por la complejidad del código y la necesidad de mantener un rendimiento fluido.

Hitos del proyecto

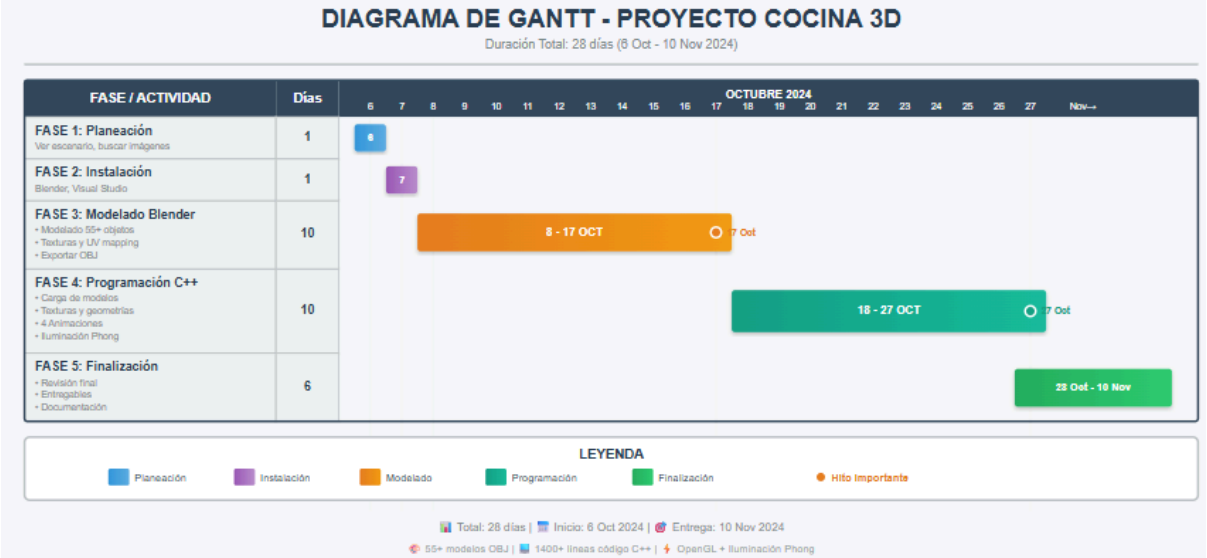
- Selección del escenario y referencias visuales
- Instalación y configuración de herramientas de trabajo
- Modelado 3D de objetos y espacios en Blender
- Exportación e integración de modelos en OpenGL
- Implementación de iluminación, cámara y animaciones
- Revisión y optimización del proyecto
- Documentación técnica y manual de usuario bilingüe completados
- Entrega del proyecto ejecutable y publicación en GitHub

Cronograma del proyecto

Fase	Actividades	Duración	Fecha límite
Fase 1: Planeación del proyecto	● Ver el escenario que sería. ● Buscar las imágenes del escenario.	1 día	6 de Octubre.
Fase 2: Instalación de blender y visual studio.	● Instalación de aplicaciones. ● Configuraciones iniciales de blender y Visual Studio.	1 día	7 Octubre.
Fase 3: Modelado en blender de los objetos.	● Modelado de blender de objetos a utilizar. ● Elegir texturas y colocar. Exportar modelos de blender en formato Obj.	10 días.	17 Octubre.
Fase 4: Creación del código en visual studio.	● Código fundamental para cargar modelos. ● Creación de modelos mediante coordenadas. ● Carga de texturas. ● Creación de animaciones. ● Poner iluminación.	10 días.	27 de Octubre.
Fase 5: Finalización.	● Revisión final. ● Preparación de entregables. ● Documentación final.	6 días.	10 de Noviembre

Diagrama del flujo del software

Diagrama de Gantt



Cotización del proyecto

La cotización que hice fue de un equipo de proyectos pequeño no en gran escala:

Costos por personal:

Rol	Días	Tarifa/Día	Subtotal
Director de Proyecto	20	$2,800	$56,000
Analista	3	$1,800	$5,400
Modelador 3D	10	$2,200	$22,000
Artista de Texturas	8	$1,900	$15,200
Programador Senior C++	12	$3,000	$36,000
Programador de Gráficos	8	$2,800	$22,400
Desarrollador de Animaciones	6	$2,400	$14,400
Ingeniero de Integración	5	$2,100	$10,500
QA / Tester	6	$1,400	$8,400
Documentador Técnico	4	$1,600	$6,400
TOTAL PERSONAL			**$196,700 MXN**

Software y licencias

Concepto	Costo
Licencia Blender (Open Source)	$0
Visual Studio Community (Gratis)	$0
Licencias OpenGL/GLEW/GLFW (Open Source)	$0
Software de gestión de proyectos (Trello/Notion)	$800
Herramientas de edición de imágenes (GIMP/Photopea)	$1,200
SUBTOTAL SOFTWARE	**$2,000 MXN**

Resumen total de la cotización:

Categoría	Monto
👥 Personal	$196,700
💿 Software y Licencias	$2,000
SUBTOTAL	**$198,700 MXN**
🔧 Contingencia (15%)	$29,805
SUBTOTAL + CONTINGENCIA	**$228,505 MXN**
💰 Utilidad/Ganancia (25%)	$57,126
TOTAL DEL PROYECTO	**$285,631 MXN**

Introducción

Para la creación de nuestro recorrido en OpenGL 3, fue necesario el uso de diferentes aplicaciones, una de ellas fue blender mediante este software pude modelar los diferentes modelos y escenarios que necesitábamos, en mi caso elegí modelar un escenario ficticio de naruto, el restaurante de ichiraku, esta caricatura se me hace muy especial debido a que cuando sufri depresion solía ver naruto y me alegraba todos los días por eso decidí hacer el recorrido acerca de este tema.

Para poder hacer el proyecto aprendimos a programar en OpenGL mediante el uso de diversas librerías como glew, glf, soil, entre otras. Las cuales nos ayudan a facilitar la programación de nuestro recorrido, además hicimos uso de diferentes shaders, los cuales nos ayudaron con la iluminación y la forma en la que se vería el programa. Por último cabe mencionar que todo lo que aprendimos en laboratorio fue de gran ayuda para la realización de este proyecto lo que fue la aplicación de texturas, el uso de librerías, los distintos tipos de iluminación, y la animación fueron de gran ayuda para la finalización de este proyecto.

A continuación muestrare las imágenes de referencia que utilizaremos para este proyecto:

Fachada:

Cuarto 1:

Cuarto2:

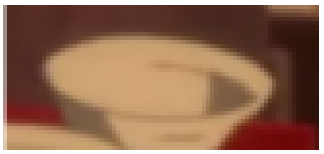
Cuarto 1ObjetosTeoria:

1)Olla con cuchara

2)Mueble de estufa

3)Asientos

4) Tazones



5)Campana

Cuarto 2ObjetosTeoria:

1)Estante sin cosas

2)Cajas

3)Refrigerador

4)Costal harina

5)Tarros

Estado del Arte de los Recorridos Virtuales

En la actualidad, los recorridos virtuales se han convertido en una herramienta ampliamente utilizada para visualizar espacios digitales de manera interactiva. Gracias a los avances en modelado 3D, motores gráficos y tecnologías de realidad virtual y aumentada, estos recorridos permiten a los usuarios explorar ambientes simulados con un alto nivel de detalle. Hoy en día, son empleados en sectores como bienes raíces, museos, educación, arquitectura, videojuegos y turismo, con el objetivo de ofrecer experiencias inmersivas sin la necesidad de estar físicamente en el lugar. Las tendencias actuales incluyen el uso de animaciones, iluminación dinámica, texturas realistas, integración con dispositivos VR y experiencias narrativas que mejoran la interacción del usuario. En este contexto, nuestro proyecto se fundamenta en estas prácticas modernas, buscando replicar un recorrido virtual funcional, visualmente atractivo e interactivo, acorde a los estándares contemporáneos en diseño digital y simulación de espacios.

Marco Teórico

Blender.

Blender es un software de modelado, animación y renderizado en 3D de código abierto que permite crear desde objetos simples hasta escenas complejas con iluminación, materiales, texturas y movimiento. Esta herramienta es ampliamente utilizada en la industria del cine,

los videojuegos y la animación por su capacidad de generar entornos realistas y controlar cada detalle del modelado.

En la creación de nuestro proyecto, Blender fue esencial para diseñar y construir los modelos y escenarios. Gracias a sus herramientas de esculpido, texturizado y renderizado, se pudieron crear los elementos del recorrido, como el escenario ficticio del restaurante Ichiraku, inspirado en la serie Naruto.

OpenGL.

OpenGL (Open Graphics Library) es una API de gráficos multiplataforma que permite renderizar imágenes 2D y 3D mediante el uso de hardware acelerado. Es una de las bibliotecas más utilizadas para la creación de entornos gráficos interactivos, simulaciones y videojuegos.

El conocimiento adquirido en el laboratorio fue fundamental para integrar todos estos elementos: la aplicación de texturas, la configuración de luces y materiales, así como la animación básica de los modelos. Todo esto permitió dar vida al entorno tridimensional y lograr una experiencia visual más inmersiva.

Metodología

1. Planeación del proyecto

En esta primera fase se definió el tema y el concepto general del recorrido. Se decidió recrear el restaurante Ichiraku de la serie Naruto, un espacio simbólico dentro del universo del anime que tiene un valor emocional personal.
Durante esta etapa se recopilaron imágenes de referencia del escenario, se analizaron los elementos principales que lo componen y se estableció el estilo visual que se mantendría en todo el proyecto.

2. Modelado en Blender

En esta fase se realizó el modelado 3D de los objetos y escenarios utilizando Blender.
Se crearon los diferentes componentes del entorno: paredes, mesas, lámparas, carteles y demás elementos decorativos. Se aplicaron texturas y materiales para lograr una apariencia

más realista, cuidando detalles como la escala, proporción y colores para mantener coherencia visual con la ambientación del restaurante Ichiraku.
Finalmente, los modelos se exportaron en formato .obj para poder ser integrados posteriormente en OpenGL.

3. Programación en OpenGL

Una vez finalizado el modelado, se procedió a la parte técnica del proyecto, donde se programó el recorrido dentro del entorno 3D y los 7 objetos que se nos pidió hacer en OpenGL y animaciones que se nos solicitaron.
Se emplearon librerías como **GLEW**, **GLFW** y **SOIL**, las cuales permitieron gestionar texturas, ventanas y renderizados.
También se implementaron **shaders** para controlar la iluminación y reflejar los diferentes materiales del entorno. En esta etapa se ajustaron parámetros como intensidad, color y posición de la luz para lograr un ambiente visual adecuado. De igual manera se aplicaron las animaciones necesarias.

4. Aplicación de texturas e iluminación

Se aplicaron las texturas diseñadas en Blender a los modelos dentro de OpenGL, verificando su correcta proyección y escala.
Posteriormente, se configuraron las fuentes de luz (direccional, ambiental y especular) para simular la iluminación del restaurante. Esta parte fue fundamental para mejorar la calidad visual del recorrido y darle profundidad a la escena.

5. Pruebas y optimización

Durante esta fase se realizaron pruebas de funcionamiento del recorrido 3D.
Se ajustaron valores de cámara, posiciones de modelos, efectos de luz y rendimiento general del programa. Además, se corrigieron errores de carga de texturas y se optimizó el código para asegurar una ejecución fluida.

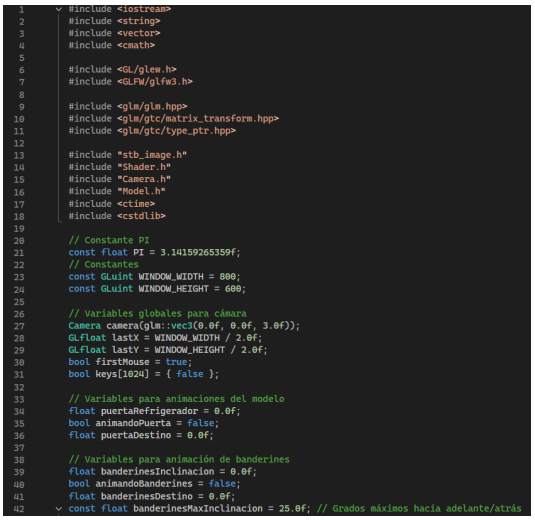
6. Integración y presentación final

Finalmente, se integraron todos los elementos: modelos, iluminación, animaciones y recorrido de cámara.
El resultado fue un entorno tridimensional interactivo que permite explorar el restaurante

Ichiraku en tiempo real, combinando el arte del modelado 3D con la programación en OpenGL.

CÓDIGO

Importa todas las librerías necesarias (OpenGL, GLFW, GLM para matemáticas 3D, stb_image para texturas). Define constantes como el tamaño de ventana 800x600 y PI. Declara variables globales: la cámara inicial, control del mouse, un arreglo para detectar teclas presionadas, y variables para animar la puerta del refrigerador y los banderines (ángulos, destinos, estados).



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

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

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

#include "stb_image.h"
#include "Shader.h"
#include "Camera.h"
#include "Model.h"
#include <ctime>
#include <cstdlib>

// Constante PI
const float PI = 3.14159265359f;
// Constantes
const GLuint WINDOW_WIDTH = 800;
const GLuint WINDOW_HEIGHT = 600;

// Variables globales para cámara
Camera camera(glm::vec3(0.0f, 0.0f, 3.0f));
GLfloat lastX = WINDOW_WIDTH / 2.0f;
GLfloat lastY = WINDOW_HEIGHT / 2.0f;
bool firstMouse = true;
bool keys[1024] = { false };

// Variables para animaciones del modelo
float puertaRefrigerador = 0.0f;
bool animandoPuerta = false;
float puertaDestino = 0.0f;

// Variables para animación de banderines
float banderinesInclinacion = 0.0f;
bool animandoBanderines = false;
float banderinesDestino = 0.0f;
const float banderinesMaxInclinacion = 25.0f; // Grados máximos hacia adelante/atrás
```

Configura el sistema de 4 luces puntuales (3 lámparas + Luna) con sus posiciones en el mundo 3D y estados encendido/apagado. Define variables de tiempo (deltaTime) para animaciones suaves. Crea el sistema de agua del grifo: una estructura Gota con posición y velocidad, un vector de hasta 50 gotas, la posición del grifo, y control de tiempo para generar gotas cada 0.02 segundos. Define el sistema de fuegos artificiales: estructura Partícula con posición, velocidad, color, vida y tamaño, vector de 1000 partículas máximo, y control de tiempo para explosiones cada 1.5 segundos.

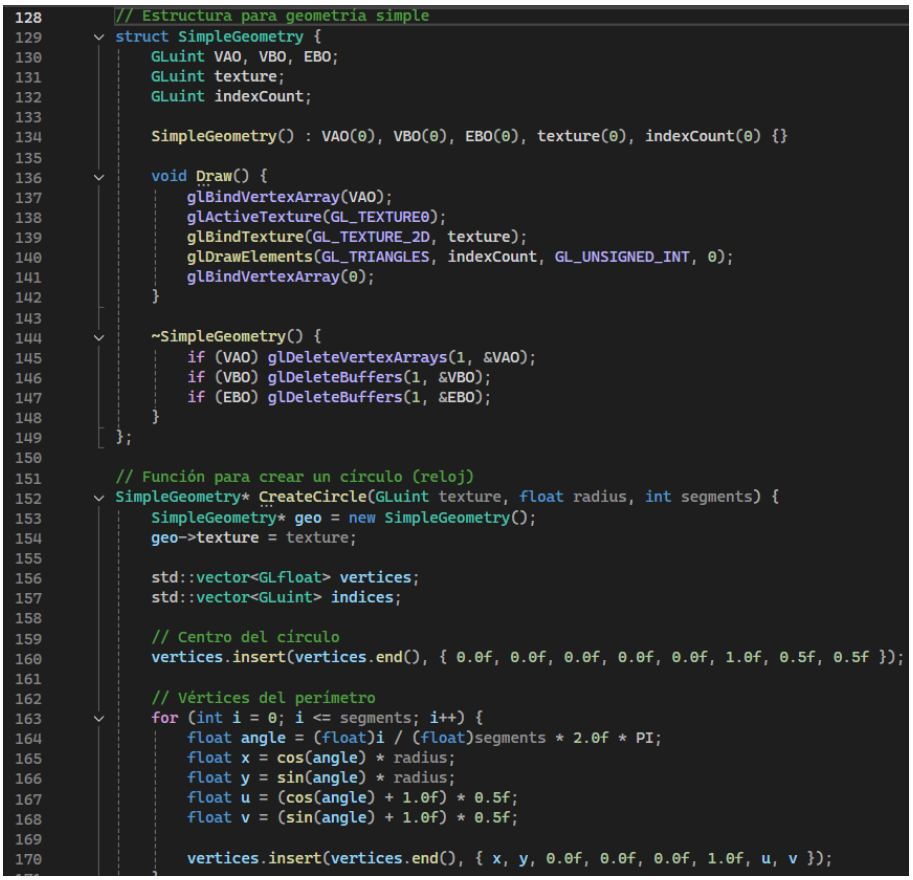
```
    // Sistema de iluminación
    bool lightStates[4] = { false, false, false, true }; // Estados de las 3 lámparas + Luna

    // Posiciones de las luces puntuales
    glm::vec3 pointLightPositions[] = {
        glm::vec3(0.0f, 3.0f, -2.0f),    // LamparasCuarto2
        glm::vec3(0.0f, 3.0f, 2.0f),     // LamparasCuarto1
        glm::vec3(-2.9f, 2.0f, 5.0f),     // LinternaLocal
        glm::vec3(4.0f, 8.0f, 5.0f)      // Luna
    };

    // Deltatime
    GLfloat deltaTime = 0.0f;
    GLfloat lastFrame = 0.0f;
    GLfloat currentFrame = 0.0f;
    // Variables para animación de agua del grifo (AGREGAR)
    bool aguaCayendo = false;
    struct Gota {
        glm::vec3 posicion;
        float velocidad;
        bool activa;
    };
    std::vector<Gota> gotas;
    const int MAX_GOTAS = 50;
    glm::vec3 posicionGrifo = glm::vec3(1.97f, 1.3f, 1.77f); // Ajusta según tu fregadero
    float tiempoUltimaGota = 0.0f;
    const float intervaloGotas = 0.02f;
    // Variables para fuegos artificiales
    bool fuegosActivos = false;
    struct Particula {
        glm::vec3 posicion;
        glm::vec3 velocidad;
        glm::vec3 color;
        float vida;
        float vidaInicial;
        float tamano;
        bool activa;
    };
    std::vector<Particula> particulas;
    const int MAX_PARTICULAS = 1000;
    float tiempoUltimoFuego = 0.0f;
    const float intervaloFuegos = 1.5f;
```

Declara las funciones principales: KeyCallback (detecta teclas), MouseCallback (captura mouse para rotar cámara), ProcessMovement (mueve cámara y actualiza animaciones cada frame), y LoadTexture (carga imágenes JPG/PNG como texturas OpenGL). La función LoadTexture recibe la ruta de una imagen y la convierte en textura OpenGL. Primero genera un ID único de textura, usa stbi_load para leer el archivo y obtiene ancho, alto y canales de color. Si la imagen cargó correctamente, determina el formato (RGB si tiene 3 canales, RGBA si tiene 4, RED si tiene 1). Luego vincula la textura, la sube a la GPU con glTexImage2D, genera mipmaps para diferentes distancias, y configura parámetros de repetición (WRAP) y filtrado (MIN/MAG_FILTER) para que se vea suave. Imprime mensaje de éxito o error, libera la memoria temporal y regresa el ID de textura.

```
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode);
void MouseCallback(GLFWwindow* window, double xPos, double yPos);
void ProcessMovement();

GLuint LoadTexture(const char* path)
{
    GLuint textureID = 0;
    glGenTextures(1, &textureID);

    int width = 0, height = 0, nrChannels = 0;
    stbi_set_flip_vertically_on_load(true);
    unsigned char* data = stbi_load(path, &width, &height, &nrChannels, 0);

    if (data)
    {
        GLenum format = GL_RGB;
        if (nrChannels == 4)
            format = GL_RGBA;
        else if (nrChannels == 1)
            format = GL_RED;

        glBindTexture(GL_TEXTURE_2D, textureID);
        glTexImage2D(GL_TEXTURE_2D, 0, format, width, height, 0, format, GL_UNSIGNED_BYTE, data);
        glGenerateMipmap(GL_TEXTURE_2D);

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

        std::cout << "Textura cargada: " << path << std::endl;
    }
    else
    {
        std::cout << "Error al cargar: " << path << std::endl;
    }

    stbi_image_free(data);
    glBindTexture(GL_TEXTURE_2D, 0);

    return textureID;
}
// Estructura para geometría simple
```

Después define la estructura SimpleGeometry que almacena geometría básica creada manualmente: tiene VAO (arreglo de vértices), VBO (buffer de datos), EBO (buffer de índices), la textura asignada, y cantidad de índices. Incluye un método Draw() que activa el VAO, vincula la textura al slot 0, dibuja triángulos con glDrawElements, y un destructor que limpia la memoria GPU al terminar.

A Partir de este punto se crean todas las estructuras necesarias para los elementos que se nos pidió en laboratorio, en mi caso hice un reloj,una mesa,un cuchillo,un queso, una tabla de picar, una caja y un mantel.



```
128     // Estructura para geometría simple
129     struct SimpleGeometry {
130         GLuint VAO, VBO, EBO;
131         GLuint texture;
132         GLuint indexCount;
133
134         SimpleGeometry() : VAO(0), VBO(0), EBO(0), texture(0), indexCount(0) {}
135
136         void Draw() {
137             glBindVertexArray(VAO);
138             glActiveTexture(GL_TEXTURE0);
139             glBindTexture(GL_TEXTURE_2D, texture);
140             glDrawElements(GL_TRIANGLES, indexCount, GL_UNSIGNED_INT, 0);
141             glBindVertexArray(0);
142         }
143
144         ~SimpleGeometry() {
145             if (VAO) glDeleteVertexArrays(1, &VAO);
146             if (VBO) glDeleteBuffers(1, &VBO);
147             if (EBO) glDeleteBuffers(1, &EBO);
148         }
149     };
150
151     // Función para crear un círculo (reloj)
152     SimpleGeometry* CreateCircle(GLuint texture, float radius, int segments) {
153         SimpleGeometry* geo = new SimpleGeometry();
154         geo->texture = texture;
155
156         std::vector<GLfloat> vertices;
157         std::vector<GLuint> indices;
158
159         // Centro del circulo
160         vertices.insert(vertices.end(), { 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.5f, 0.5f });
161
162         // Vértices del perímetro
163         for (int i = 0; i <= segments; i++) {
164             float angle = (float)i / (float)segments * 2.0f * PI;
165             float x = cos(angle) * radius;
166             float y = sin(angle) * radius;
167             float u = (cos(angle) + 1.0f) * 0.5f;
168             float v = (sin(angle) + 1.0f) * 0.5f;
169
170             vertices.insert(vertices.end(), { x, y, 0.0f, 0.0f, 0.0f, 1.0f, u, v });
```

```
    }

    // Índices
    for (int i = 1; i <= segments; i++) {
        indices.push_back(0);
        indices.push_back(i);
        indices.push_back(i + 1);
    }

    geo->indexCount = (GLuint)indices.size();

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);

    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    // Posición
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
    // Normal
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    // TexCoords
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

    glBindVertexArray(0);

    return geo;
}

// Función para crear una manecilla del reloj
SimpleGeometry* CreateClockHand(GLuint texture, float width, float length) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;
```

```
    std::vector<GLfloat> vertices = {
        // Posición              Normal             TexCoords
        -width / 2, 0.0f, 0.01f,   0.0f, 0.0f, 1.0f,   0.0f, 0.0f,
         width / 2, 0.0f, 0.01f,   0.0f, 0.0f, 1.0f,   1.0f, 0.0f,
         width / 2, length, 0.01f, 0.0f, 0.0f, 1.0f,   1.0f, 1.0f,
        -width / 2, length, 0.01f, 0.0f, 0.0f, 1.0f,   0.0f, 1.0f
    };

    std::vector<GLuint> indices = { 0, 1, 2, 2, 3, 0 };
    geo->indexCount = 6;

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);

    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

    glBindVertexArray(0);

    return geo;
}

// Función para crear un cubo
SimpleGeometry* CreateCube(GLuint texture) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    std::vector<GLfloat> vertices = {
        // Cara frontal (z+)
```

```cpp
// Función para crear un cubo
SimpleGeometry* CreateCube(GLuint texture) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    std::vector<GLfloat> vertices = {
        // Cara frontal (z+)
        -0.5f, -0.5f,  0.5f,  0.0f, 0.0f, 1.0f,  0.0f, 0.0f,
         0.5f, -0.5f,  0.5f,  0.0f, 0.0f, 1.0f,  1.0f, 0.0f,
         0.5f,  0.5f,  0.5f,  0.0f, 0.0f, 1.0f,  1.0f, 1.0f,
        -0.5f,  0.5f,  0.5f,  0.0f, 0.0f, 1.0f,  0.0f, 1.0f,

        // Cara trasera (z-)
        -0.5f, -0.5f, -0.5f,  0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
        -0.5f,  0.5f, -0.5f,  0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
         0.5f,  0.5f, -0.5f,  0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
         0.5f, -0.5f, -0.5f,  0.0f, 0.0f, -1.0f, 0.0f, 0.0f,

         // Cara superior (y+)
        -0.5f,  0.5f, -0.5f,  0.0f, 1.0f, 0.0f,  0.0f, 1.0f,
        -0.5f,  0.5f,  0.5f,  0.0f, 1.0f, 0.0f,  0.0f, 0.0f,
         0.5f,  0.5f,  0.5f,  0.0f, 1.0f, 0.0f,  1.0f, 0.0f,
         0.5f,  0.5f, -0.5f,  0.0f, 1.0f, 0.0f,  1.0f, 1.0f,

         // Cara inferior (y-)
        -0.5f, -0.5f, -0.5f,  0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
         0.5f, -0.5f, -0.5f,  0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
         0.5f, -0.5f,  0.5f,  0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
        -0.5f, -0.5f,  0.5f,  0.0f, -1.0f, 0.0f, 0.0f, 0.0f,

         // Cara derecha (x+)
         0.5f, -0.5f, -0.5f,  1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
         0.5f,  0.5f, -0.5f,  1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
         0.5f,  0.5f,  0.5f,  1.0f, 0.0f, 0.0f,  0.0f, 1.0f,
         0.5f, -0.5f,  0.5f,  1.0f, 0.0f, 0.0f,  0.0f, 0.0f,

         // Cara izquierda (x-)
        -0.5f, -0.5f, -0.5f,  -1.0f, 0.0f, 0.0f, 0.0f, 0.0f
        -0.5f, -0.5f,  0.5f,  -1.0f, 0.0f, 0.0f, 1.0f, 0.0f
        -0.5f,  0.5f,  0.5f,  -1.0f, 0.0f, 0.0f, 1.0f, 1.0f
        -0.5f,  0.5f, -0.5f,  -1.0f, 0.0f, 0.0f, 0.0f, 1.0f
    };

```

```
    std::vector<GLuint> indices = {
        0, 1, 2,  2, 3, 0,    // Frontal
        4, 5, 6,  6, 7, 4,    // Trasera
        8, 9, 10, 10, 11, 8,  // Superior
        12, 13, 14, 14, 15, 12, // Inferior
        16, 17, 18, 18, 19, 16, // Derecha
        20, 21, 22, 22, 23, 20  // Izquierda
    };

    geo->indexCount = 36;

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);

    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

    glBindVertexArray(0);

    return geo;
}

// Función para crear superficie de mesa
SimpleGeometry* CreateMesaTop(GLuint texture) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    std::vector<GLfloat> vertices = {
        // Superficie superior
        -0.5f, 0.05f, -0.5f,  0.0f, 1.0f, 0.0f,  0.0f, 0.0f,
         0.5f, 0.05f, -0.5f,  0.0f, 1.0f, 0.0f,  1.0f, 0.0f,
         0.5f, 0.05f,  0.5f,  0.0f, 1.0f, 0.0f,  1.0f, 1.0f,
        -0.5f, 0.05f,  0.5f,  0.0f, 1.0f, 0.0f,  0.0f, 1.0f,

        // Superficie inferior
        -0.5f, 0.0f, -0.5f,  0.0f, -1.0f, 0.0f,  0.0f, 0.0f,
         0.5f, 0.0f, -0.5f,  0.0f, -1.0f, 0.0f,  1.0f, 0.0f,
         0.5f, 0.0f,  0.5f,  0.0f, -1.0f, 0.0f,  1.0f, 1.0f,
        -0.5f, 0.0f,  0.5f,  0.0f, -1.0f, 0.0f,  0.0f, 1.0f,

        // Borde frontal (Z-)
        -0.5f, 0.0f, -0.5f,  0.0f, 0.0f, -1.0f,  0.0f, 0.0f,
         0.5f, 0.0f, -0.5f,  0.0f, 0.0f, -1.0f,  1.0f, 0.0f,
         0.5f, 0.05f, -0.5f,  0.0f, 0.0f, -1.0f,  1.0f, 1.0f,
        -0.5f, 0.05f, -0.5f,  0.0f, 0.0f, -1.0f,  0.0f, 1.0f,

        // Borde trasero (Z+)
        -0.5f, 0.0f,  0.5f,  0.0f, 0.0f, 1.0f,  0.0f, 0.0f,
         0.5f, 0.0f,  0.5f,  0.0f, 0.0f, 1.0f,  1.0f, 0.0f,
         0.5f, 0.05f,  0.5f,  0.0f, 0.0f, 1.0f,  1.0f, 1.0f,
        -0.5f, 0.05f,  0.5f,  0.0f, 0.0f, 1.0f,  0.0f, 1.0f,

        // Borde derecho (X+)
         0.5f, 0.0f, -0.5f,  1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
         0.5f, 0.0f,  0.5f,  1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
         0.5f, 0.05f,  0.5f,  1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
         0.5f, 0.05f, -0.5f,  1.0f, 0.0f, 0.0f,  0.0f, 1.0f,

         // Borde izquierdo (X-)
        -0.5f, 0.0f, -0.5f,  -1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
        -0.5f, 0.0f,  0.5f,  -1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
        -0.5f, 0.05f,  0.5f,  -1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
        -0.5f, 0.05f, -0.5f,  -1.0f, 0.0f, 0.0f,  0.0f, 1.0f
    };

    std::vector<GLuint> indices = {
        0, 1, 2,  2, 3, 0,      // Superior
        4, 6, 5,  4, 7, 6,      // Inferior
        8, 9, 10, 10, 11, 8,    // Borde frontal
        12, 14, 13, 12, 15, 14, // Borde trasero
        16, 17, 18, 18, 19, 16, // Borde derecho
        20, 22, 21, 20, 23, 22  // Borde izquierdo
    };
```

```
    geo->indexCount = 36;

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);

    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

    glBindVertexArray(0);

    return geo;
}
// Función para crear pata de mesa
SimpleGeometry* CreatePataMesa(GLuint texture) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    std::vector<GLfloat> vertices = {
        // Cilindro simplificado como cubo delgado
        -0.05f, -0.7f, -0.05f,  0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
         0.05f, -0.7f, -0.05f,  0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
         0.05f, -0.7f,  0.05f,  0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
        -0.05f, -0.7f,  0.05f,  0.0f, -1.0f, 0.0f, 0.0f, 1.0f,

        -0.05f, 0.0f, -0.05f,  0.0f, 1.0f, 0.0f,  0.0f, 0.0f,
         0.05f, 0.0f, -0.05f,  0.0f, 1.0f, 0.0f,  1.0f, 0.0f,
         0.05f, 0.0f,  0.05f,  0.0f, 1.0f, 0.0f,  1.0f, 1.0f,
        -0.05f, 0.0f,  0.05f,  0.0f, 1.0f, 0.0f,  0.0f, 1.0f
    };
```

```
    std::vector<GLuint> indices = {
        0, 1, 2,  2, 3, 0,  // Abajo
        4, 6, 5,  4, 7, 6,  // Arriba
        0, 1, 5,  5, 4, 0,  // Laterales
        1, 2, 6,  6, 5, 1,
        2, 3, 7,  7, 6, 2,
        3, 0, 4,  4, 7, 3
    };

    geo->indexCount = 36;

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);

    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

    glBindVertexArray(0);

    return geo;
}
// Función para crear Narutomaki (cilindro pequeño)
SimpleGeometry* CreateNarutomaki(GLuint texture) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    std::vector<GLfloat> vertices = {
        // Base inferior (y = 0.0f)
        -0.05f, 0.0f, -0.05f,  0.0f, -1.0f, 0.0f,  0.0f, 0.0f,
         0.05f, 0.0f, -0.05f,  0.0f, -1.0f, 0.0f,  1.0f, 0.0f,
```

```
         0.05f, 0.0f,  0.05f,  0.0f, -1.0f, 0.0f,  1.0f, 1.0f,
        -0.05f, 0.0f,  0.05f,  0.0f, -1.0f, 0.0f,  0.0f, 1.0f,

        // Base superior (y = 0.05f)
        -0.05f, 0.05f, -0.05f,  0.0f, 1.0f, 0.0f,  0.0f, 0.0f,
         0.05f, 0.05f, -0.05f,  0.0f, 1.0f, 0.0f,  1.0f, 0.0f,
         0.05f, 0.05f,  0.05f,  0.0f, 1.0f, 0.0f,  1.0f, 1.0f,
        -0.05f, 0.05f,  0.05f,  0.0f, 1.0f, 0.0f,  0.0f, 1.0f,

        // Lado frontal (Z+)
        -0.05f, 0.0f,  0.05f,  0.0f, 0.0f, 1.0f,  0.0f, 0.0f,
         0.05f, 0.0f,  0.05f,  0.0f, 0.0f, 1.0f,  1.0f, 0.0f,
         0.05f, 0.05f,  0.05f,  0.0f, 0.0f, 1.0f,  1.0f, 1.0f,
        -0.05f, 0.05f,  0.05f,  0.0f, 0.0f, 1.0f,  0.0f, 1.0f,

        // Lado trasero (Z-)
        -0.05f, 0.0f, -0.05f,  0.0f, 0.0f, -1.0f,  1.0f, 0.0f,
         0.05f, 0.0f, -0.05f,  0.0f, 0.0f, -1.0f,  0.0f, 0.0f,
         0.05f, 0.05f, -0.05f,  0.0f, 0.0f, -1.0f,  0.0f, 1.0f,
        -0.05f, 0.05f, -0.05f,  0.0f, 0.0f, -1.0f,  1.0f, 1.0f,

        // Lado derecho (X+)
         0.05f, 0.0f, -0.05f,  1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
         0.05f, 0.0f,  0.05f,  1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
         0.05f, 0.05f,  0.05f,  1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
         0.05f, 0.05f, -0.05f,  1.0f, 0.0f, 0.0f,  0.0f, 1.0f,

         // Lado izquierdo (X-)
        -0.05f, 0.0f, -0.05f,  -1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
        -0.05f, 0.0f,  0.05f,  -1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
        -0.05f, 0.05f,  0.05f,  -1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
        -0.05f, 0.05f, -0.05f,  -1.0f, 0.0f, 0.0f,  0.0f, 1.0f
    };

    std::vector<GLuint> indices = {
        0, 1, 2,  2, 3, 0,      // Base inferior
        4, 6, 5,  4, 7, 6,      // Base superior
        8, 9, 10, 10, 11, 8,    // Frontal
        12, 14, 13, 12, 15, 14, // Trasero
        16, 17, 18, 18, 19, 16, // Derecho
        20, 22, 21, 20, 23, 22  // Izquierdo
    };

    geo->indexCount = 36;

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);

    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

    glBindVertexArray(0);

    return geo;
}

// Función para crear Sartén
SimpleGeometry* CreateSarten(GLuint texture) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    std::vector<GLfloat> vertices = {
        // Base inferior
        -0.2f, 0.0f, -0.2f,  0.0f, -1.0f, 0.0f,  0.0f, 0.0f,
         0.2f, 0.0f, -0.2f,  0.0f, -1.0f, 0.0f,  1.0f, 0.0f,
         0.2f, 0.0f,  0.2f,  0.0f, -1.0f, 0.0f,  1.0f, 1.0f,
        -0.2f, 0.0f,  0.2f,  0.0f, -1.0f, 0.0f,  0.0f, 1.0f,

        // Base superior
        -0.2f, 0.02f, -0.2f,  0.0f, 1.0f, 0.0f,  0.0f, 0.0f,
         0.2f, 0.02f, -0.2f,  0.0f, 1.0f, 0.0f,  1.0f, 0.0f,
         0.2f, 0.02f,  0.2f,  0.0f, 1.0f, 0.0f,  1.0f, 1.0f,
        -0.2f, 0.02f,  0.2f,  0.0f, 1.0f, 0.0f,  0.0f, 1.0f,
```

```
        // Lado frontal
        -0.2f, 0.0f,  0.2f,  0.0f, 0.0f, 1.0f,  0.0f, 0.0f,
         0.2f, 0.0f,  0.2f,  0.0f, 0.0f, 1.0f,  1.0f, 0.0f,
         0.2f, 0.02f,  0.2f,  0.0f, 0.0f, 1.0f,  1.0f, 1.0f,
        -0.2f, 0.02f,  0.2f,  0.0f, 0.0f, 1.0f,  0.0f, 1.0f,

        // Lado trasero
        -0.2f, 0.0f, -0.2f,  0.0f, 0.0f, -1.0f,  1.0f, 0.0f,
         0.2f, 0.0f, -0.2f,  0.0f, 0.0f, -1.0f,  0.0f, 0.0f,
         0.2f, 0.02f, -0.2f,  0.0f, 0.0f, -1.0f,  0.0f, 1.0f,
        -0.2f, 0.02f, -0.2f,  0.0f, 0.0f, -1.0f,  1.0f, 1.0f,

        // Lado derecho
         0.2f, 0.0f, -0.2f,  1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
         0.2f, 0.0f,  0.2f,  1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
         0.2f, 0.02f,  0.2f,  1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
         0.2f, 0.02f, -0.2f,  1.0f, 0.0f, 0.0f,  0.0f, 1.0f,

         // Lado izquierdo
        -0.2f, 0.0f, -0.2f,  -1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
        -0.2f, 0.0f,  0.2f,  -1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
        -0.2f, 0.02f,  0.2f,  -1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
        -0.2f, 0.02f, -0.2f,  -1.0f, 0.0f, 0.0f,  0.0f, 1.0f
    };

    std::vector<GLuint> indices = {
        0, 1, 2,  2, 3, 0,
        4, 6, 5,  4, 7, 6,
        8, 9, 10, 10, 11, 8,
        12, 14, 13, 12, 15, 14,
        16, 17, 18, 18, 19, 16,
        20, 22, 21, 20, 23, 22
    };

    geo->indexCount = 36;

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);

    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
```

```
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

    glBindVertexArray(0);

    return geo;
}

// Función para crear Tabla de Picar
SimpleGeometry* CreateTablaPicar(GLuint texture) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    std::vector<GLfloat> vertices = {
        // Base inferior
        -0.2f, 0.0f, -0.15f,  0.0f, -1.0f, 0.0f,  0.0f, 0.0f,
         0.2f, 0.0f, -0.15f,  0.0f, -1.0f, 0.0f,  1.0f, 0.0f,
         0.2f, 0.0f,  0.15f,  0.0f, -1.0f, 0.0f,  1.0f, 1.0f,
        -0.2f, 0.0f,  0.15f,  0.0f, -1.0f, 0.0f,  0.0f, 1.0f,

        // Base superior
        -0.2f, 0.02f, -0.15f,  0.0f, 1.0f, 0.0f,  0.0f, 0.0f,
         0.2f, 0.02f, -0.15f,  0.0f, 1.0f, 0.0f,  1.0f, 0.0f,
         0.2f, 0.02f,  0.15f,  0.0f, 1.0f, 0.0f,  1.0f, 1.0f,
        -0.2f, 0.02f,  0.15f,  0.0f, 1.0f, 0.0f,  0.0f, 1.0f,

        // Lado frontal
        -0.2f, 0.0f,  0.15f,  0.0f, 0.0f, 1.0f,  0.0f, 0.0f,
         0.2f, 0.0f,  0.15f,  0.0f, 0.0f, 1.0f,  1.0f, 0.0f,
         0.2f, 0.02f,  0.15f,  0.0f, 0.0f, 1.0f,  1.0f, 1.0f,
        -0.2f, 0.02f,  0.15f,  0.0f, 0.0f, 1.0f,  0.0f, 1.0f,

        // Lado trasero
        -0.2f, 0.0f, -0.15f,  0.0f, 0.0f, -1.0f,  1.0f, 0.0f,
```

```
         0.2f, 0.0f, -0.15f,  0.0f, 0.0f, -1.0f,  0.0f, 0.0f,
         0.2f, 0.02f, -0.15f,  0.0f, 0.0f, -1.0f,  0.0f, 1.0f,
        -0.2f, 0.02f, -0.15f,  0.0f, 0.0f, -1.0f,  1.0f, 1.0f,

        // Lado derecho
         0.2f, 0.0f, -0.15f,  1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
         0.2f, 0.0f,  0.15f,  1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
         0.2f, 0.02f,  0.15f,  1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
         0.2f, 0.02f, -0.15f,  1.0f, 0.0f, 0.0f,  0.0f, 1.0f,

         // Lado izquierdo
        -0.2f, 0.0f, -0.15f,  -1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
        -0.2f, 0.0f,  0.15f,  -1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
        -0.2f, 0.02f,  0.15f,  -1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
        -0.2f, 0.02f, -0.15f,  -1.0f, 0.0f, 0.0f,  0.0f, 1.0f
    };

    std::vector<GLuint> indices = {
        0, 1, 2,  2, 3, 0,
        4, 6, 5,  4, 7, 6,
        8, 9, 10, 10, 11, 8,
        12, 14, 13, 12, 15, 14,
        16, 17, 18, 18, 19, 16,
        20, 22, 21, 20, 23, 22
    };

    geo->indexCount = 36;

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);

    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
    glEnableVertexAttribArray(1);
```

```
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

    glBindVertexArray(0);

    return geo;
}
// Función para crear hoja de cuchillo (rectángulo delgado alargado)
SimpleGeometry* CreateHojaCuchillo(GLuint texture) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    std::vector<GLfloat> vertices = {
        // Base inferior
        -0.025f, 0.0f, -0.2f,  0.0f, -1.0f, 0.0f,  0.0f, 0.0f,
         0.025f, 0.0f, -0.2f,  0.0f, -1.0f, 0.0f,  1.0f, 0.0f,
         0.025f, 0.0f,  0.2f,  0.0f, -1.0f, 0.0f,  1.0f, 1.0f,
        -0.025f, 0.0f,  0.2f,  0.0f, -1.0f, 0.0f,  0.0f, 1.0f,

        // Base superior
        -0.025f, 0.02f, -0.2f,  0.0f, 1.0f, 0.0f,  0.0f, 0.0f,
         0.025f, 0.02f, -0.2f,  0.0f, 1.0f, 0.0f,  1.0f, 0.0f,
         0.025f, 0.02f,  0.2f,  0.0f, 1.0f, 0.0f,  1.0f, 1.0f,
        -0.025f, 0.02f,  0.2f,  0.0f, 1.0f, 0.0f,  0.0f, 1.0f,

        // Lado frontal (Z+)
        -0.025f, 0.0f,  0.2f,  0.0f, 0.0f, 1.0f,  0.0f, 0.0f,
         0.025f, 0.0f,  0.2f,  0.0f, 0.0f, 1.0f,  1.0f, 0.0f,
         0.025f, 0.02f,  0.2f,  0.0f, 0.0f, 1.0f,  1.0f, 1.0f,
        -0.025f, 0.02f,  0.2f,  0.0f, 0.0f, 1.0f,  0.0f, 1.0f,

        // Lado trasero (Z-)
        -0.025f, 0.0f, -0.2f,  0.0f, 0.0f, -1.0f,  1.0f, 0.0f,
         0.025f, 0.0f, -0.2f,  0.0f, 0.0f, -1.0f,  0.0f, 0.0f,
         0.025f, 0.02f, -0.2f,  0.0f, 0.0f, -1.0f,  0.0f, 1.0f,
        -0.025f, 0.02f, -0.2f,  0.0f, 0.0f, -1.0f,  1.0f, 1.0f,

        // Lado derecho (X+)
         0.025f, 0.0f, -0.2f,  1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
         0.025f, 0.0f,  0.2f,  1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
         0.025f, 0.02f,  0.2f,  1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
         0.025f, 0.02f, -0.2f,  1.0f, 0.0f, 0.0f,  0.0f, 1.0f,
```

```
        // Lado izquierdo (X-)
        -0.025f, 0.0f, -0.2f,  -1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
        -0.025f, 0.0f,  0.2f,  -1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
        -0.025f, 0.02f,  0.2f,  -1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
        -0.025f, 0.02f, -0.2f,  -1.0f, 0.0f, 0.0f,  0.0f, 1.0f
    };

    std::vector<GLuint> indices = {
        0, 1, 2,  2, 3, 0,      // Base inferior
        4, 6, 5,  4, 7, 6,      // Base superior
        8, 9, 10, 10, 11, 8,    // Frontal
        12, 14, 13, 12, 15, 14, // Trasero
        16, 17, 18, 18, 19, 16, // Derecho
        20, 22, 21, 20, 23, 22  // Izquierdo
    };

    geo->indexCount = 36;

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);

    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

    glBindVertexArray(0);

    return geo;
}

// Función para crear mango de cuchillo (cubo más pequeño)
```

```
SimpleGeometry* CreateMangoCuchillo(GLuint texture) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    std::vector<GLfloat> vertices = {
        // Base inferior
        -0.025f, 0.0f, -0.05f,  0.0f, -1.0f, 0.0f,  0.0f, 0.0f,
         0.025f, 0.0f, -0.05f,  0.0f, -1.0f, 0.0f,  1.0f, 0.0f,
         0.025f, 0.0f,  0.05f,  0.0f, -1.0f, 0.0f,  1.0f, 1.0f,
        -0.025f, 0.0f,  0.05f,  0.0f, -1.0f, 0.0f,  0.0f, 1.0f,

        // Base superior
        -0.025f, 0.02f, -0.05f,  0.0f, 1.0f, 0.0f,  0.0f, 0.0f,
         0.025f, 0.02f, -0.05f,  0.0f, 1.0f, 0.0f,  1.0f, 0.0f,
         0.025f, 0.02f,  0.05f,  0.0f, 1.0f, 0.0f,  1.0f, 1.0f,
        -0.025f, 0.02f,  0.05f,  0.0f, 1.0f, 0.0f,  0.0f, 1.0f,

        // Lado frontal (Z+)
        -0.025f, 0.0f,  0.05f,  0.0f, 0.0f, 1.0f,  0.0f, 0.0f,
         0.025f, 0.0f,  0.05f,  0.0f, 0.0f, 1.0f,  1.0f, 0.0f,
         0.025f, 0.02f,  0.05f,  0.0f, 0.0f, 1.0f,  1.0f, 1.0f,
        -0.025f, 0.02f,  0.05f,  0.0f, 0.0f, 1.0f,  0.0f, 1.0f,

        // Lado trasero (Z-)
        -0.025f, 0.0f, -0.05f,  0.0f, 0.0f, -1.0f,  1.0f, 0.0f,
         0.025f, 0.0f, -0.05f,  0.0f, 0.0f, -1.0f,  0.0f, 0.0f,
         0.025f, 0.02f, -0.05f,  0.0f, 0.0f, -1.0f,  0.0f, 1.0f,
        -0.025f, 0.02f, -0.05f,  0.0f, 0.0f, -1.0f,  1.0f, 1.0f,

        // Lado derecho (X+)
         0.025f, 0.0f, -0.05f,  1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
         0.025f, 0.0f,  0.05f,  1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
         0.025f, 0.02f,  0.05f,  1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
         0.025f, 0.02f, -0.05f,  1.0f, 0.0f, 0.0f,  0.0f, 1.0f,

         // Lado izquierdo (X-)
        -0.025f, 0.0f, -0.05f,  -1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
        -0.025f, 0.0f,  0.05f,  -1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
        -0.025f, 0.02f,  0.05f,  -1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
        -0.025f, 0.02f, -0.05f,  -1.0f, 0.0f, 0.0f,  0.0f, 1.0f
    };

    std::vector<GLuint> indices = {
```

```
        0, 1, 2,  2, 3, 0,
        4, 6, 5,  4, 7, 6,
        8, 9, 10, 10, 11, 8,
        12, 14, 13, 12, 15, 14,
        16, 17, 18, 18, 19, 16,
        20, 22, 21, 20, 23, 22
    };

    geo->indexCount = 36;

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);

    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

    glBindVertexArray(0);

    return geo;
}
// Función para crear una gota de agua (cubo alargado con grosor medio)
SimpleGeometry* CreateGotaAgua(GLuint texture) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    // Gotas con grosor medio (0.008) y largas (0.04)
    std::vector<GLfloat> vertices = {
        -0.008f, -0.04f,  0.008f,  0.0f, 0.0f, 1.0f,  0.0f, 0.0f,
         0.008f, -0.04f,  0.008f,  0.0f, 0.0f, 1.0f,  1.0f, 0.0f,
         0.008f,  0.04f,  0.008f,  0.0f, 0.0f, 1.0f,  1.0f, 1.0f,
```

```
        -0.008f,  0.04f,  0.008f,  0.0f, 0.0f, 1.0f,  0.0f, 1.0f,

        -0.008f, -0.04f, -0.008f,  0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
        -0.008f,  0.04f, -0.008f,  0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
         0.008f,  0.04f, -0.008f,  0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
         0.008f, -0.04f, -0.008f,  0.0f, 0.0f, -1.0f, 0.0f, 0.0f,

        -0.008f,  0.04f, -0.008f,  0.0f, 1.0f, 0.0f,  0.0f, 1.0f,
        -0.008f,  0.04f,  0.008f,  0.0f, 1.0f, 0.0f,  0.0f, 0.0f,
         0.008f,  0.04f,  0.008f,  0.0f, 1.0f, 0.0f,  1.0f, 0.0f,
         0.008f,  0.04f, -0.008f,  0.0f, 1.0f, 0.0f,  1.0f, 1.0f,

        -0.008f, -0.04f, -0.008f,  0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
         0.008f, -0.04f, -0.008f,  0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
         0.008f, -0.04f,  0.008f,  0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
        -0.008f, -0.04f,  0.008f,  0.0f, -1.0f, 0.0f, 0.0f, 0.0f,

         0.008f, -0.04f, -0.008f,  1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
         0.008f,  0.04f, -0.008f,  1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
         0.008f,  0.04f,  0.008f,  1.0f, 0.0f, 0.0f,  0.0f, 1.0f,
         0.008f, -0.04f,  0.008f,  1.0f, 0.0f, 0.0f,  0.0f, 0.0f,

        -0.008f, -0.04f, -0.008f,  -1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
        -0.008f, -0.04f,  0.008f,  -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
        -0.008f,  0.04f,  0.008f,  -1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
        -0.008f,  0.04f, -0.008f,  -1.0f, 0.0f, 0.0f, 0.0f, 1.0f
    };

    std::vector<GLuint> indices = {
        0, 1, 2,  2, 3, 0,
        4, 5, 6,  6, 7, 4,
        8, 9, 10, 10, 11, 8,
        12, 13, 14, 14, 15, 12,
        16, 17, 18, 18, 19, 16,
        20, 21, 22, 22, 23, 20
    };

    geo->indexCount = 36;

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);
```

```
    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

    glBindVertexArray(0);

    return geo;
}
// Crear esfera pequeña para partículas de fuegos artificiales
SimpleGeometry* CreateParticulaFuego(GLuint texture) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    std::vector<GLfloat> vertices = {
        // Cara frontal
        -0.5f, -0.5f,  0.5f,  0.0f, 0.0f, 1.0f,  0.0f, 0.0f,
         0.5f, -0.5f,  0.5f,  0.0f, 0.0f, 1.0f,  1.0f, 0.0f,
         0.5f,  0.5f,  0.5f,  0.0f, 0.0f, 1.0f,  1.0f, 1.0f,
        -0.5f,  0.5f,  0.5f,  0.0f, 0.0f, 1.0f,  0.0f, 1.0f,

        // Cara trasera
        -0.5f, -0.5f, -0.5f,  0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
        -0.5f,  0.5f, -0.5f,  0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
         0.5f,  0.5f, -0.5f,  0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
         0.5f, -0.5f, -0.5f,  0.0f, 0.0f, -1.0f, 0.0f, 0.0f,

         // Cara superior
        -0.5f,  0.5f, -0.5f,  0.0f, 1.0f, 0.0f,  0.0f, 1.0f,
        -0.5f,  0.5f,  0.5f,  0.0f, 1.0f, 0.0f,  0.0f, 0.0f,
         0.5f,  0.5f,  0.5f,  0.0f, 1.0f, 0.0f,  1.0f, 0.0f,
         0.5f,  0.5f, -0.5f,  0.0f, 1.0f, 0.0f,  1.0f, 1.0f,
```

```
        // Cara inferior
        -0.5f, -0.5f, -0.5f,  0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
         0.5f, -0.5f, -0.5f,  0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
         0.5f, -0.5f,  0.5f,  0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
        -0.5f, -0.5f,  0.5f,  0.0f, -1.0f, 0.0f, 0.0f, 0.0f,

        // Cara derecha
         0.5f, -0.5f, -0.5f,  1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
         0.5f,  0.5f, -0.5f,  1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
         0.5f,  0.5f,  0.5f,  1.0f, 0.0f, 0.0f,  0.0f, 1.0f,
         0.5f, -0.5f,  0.5f,  1.0f, 0.0f, 0.0f,  0.0f, 0.0f,

         // Cara izquierda
        -0.5f, -0.5f, -0.5f,  -1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
        -0.5f, -0.5f,  0.5f,  -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
        -0.5f,  0.5f,  0.5f,  -1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
        -0.5f,  0.5f, -0.5f,  -1.0f, 0.0f, 0.0f, 0.0f, 1.0f
    };

    std::vector<GLuint> indices = {
        0, 1, 2,  2, 3, 0,
        4, 5, 6,  6, 7, 4,
        8, 9, 10, 10, 11, 8,
        12, 13, 14, 14, 15, 12,
        16, 17, 18, 18, 19, 16,
        20, 21, 22, 22, 23, 20
    };

    geo->indexCount = 36;

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);

    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
```

```
        glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
        glEnableVertexAttribArray(1);
        glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
        glEnableVertexAttribArray(2);
        glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

        glBindVertexArray(0);

        return geo;
    }
    void CrearExplosion(glm::vec3 posicion, glm::vec3 color) {
        int particulasPorExplosion = 250;  // MUCHAS MÁS PARTÍCULAS
        int creadas = 0;

        for (int i = 0; i < MAX_PARTICULAS && creadas < particulasPorExplosion; i++) {
            if (!particulas[i].activa) {
                particulas[i].activa = true;
                particulas[i].posicion = posicion;
                particulas[i].vida = 3.0f + (rand() % 150) / 50.0f;  // Viven MÁS tiempo
                particulas[i].vidaInicial = particulas[i].vida;
                particulas[i].color = color;
                particulas[i].tamano = 0.35f + (rand() % 150) / 500.0f;  // MUY GRANDES (0.35-0.65)

                // Velocidades MUY RÁPIDAS para explosiones grandes
                float theta = (rand() % 360) * PI / 180.0f;
                float phi = (rand() % 180) * PI / 180.0f;
                float velocidad = 5.0f + (rand() % 600) / 100.0f;  // 5.0 a 11.0 (muy rápido)

                particulas[i].velocidad = glm::vec3(
                    velocidad * sin(phi) * cos(theta),
                    velocidad * cos(phi),
                    velocidad * sin(phi) * sin(theta)
                );

                creadas++;
            }
        }
    }
    int main()
    {
        if (!glfwInit())
        {
            std::cout << "Error al inicializar GLFW" << std::endl;
```

Esta sección del código crea la proyección en perspectiva de la cámara y luego carga texturas adicionales e inicializa geometrías personalizadas mediante funciones especializadas. Define texturas para elementos específicos como un reloj, manecillas, cubos, mesa de cocina, narutomaki, sartén, tabla de picar, cuchillo y agua. Posteriormente crea objetos geométricos simples usando estas texturas: un círculo para la cara del reloj, manecillas de hora y minuto, un cubo, el tablero y patas de una mesa,queso, una mantel, tabla de picar con cuchillo (hoja y mango separados), gotas de agua y partículas de fuego para efectos visuales, todos almacenados en estructuras SimpleGeometry para ser renderizados posteriormente en la escena 3D. Además de cargar los modelos 3d que modelamos en blender.

```
        return EXIT_FAILURE;
    }

    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
    glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);

    GLFWwindow* window = glfwCreateWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Modelado Jerarquico - Cocina 3D", nullptr, nullptr);

    if (!window)
    {
        std::cout << "Error al crear ventana GLFW" << std::endl;
        glfwTerminate();
        return EXIT_FAILURE;
    }

    glfwMakeContextCurrent(window);
    glfwSetKeyCallback(window, KeyCallback);
    glfwSetCursorPosCallback(window, MouseCallback);
    glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);

    glewExperimental = GL_TRUE;
    if (glewInit() != GLEW_OK)
    {
        std::cout << "Error al inicializar GLEW" << std::endl;
        glfwTerminate();
        return EXIT_FAILURE;
    }

    int screenWidth = 0, screenHeight = 0;
    glfwGetFramebufferSize(window, &screenWidth, &screenHeight);
    glViewport(0, 0, screenWidth, screenHeight);
    glEnable(GL_DEPTH_TEST);
    glClearColor(0.05f, 0.05f, 0.15f, 1.0f);

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

    std::cout << "\n=== CONTROLES ===" << std::endl;
    std::cout << "W/A/S/D: Mover cámara" << std::endl;
    std::cout << "Mouse: Mirar alrededor" << std::endl;
    std::cout << "O/C: Abrir/Cerrar puerta del refrigerador" << std::endl;
```

```
    std::cout << "N: Mover banderines hacia atrás" << std::endl;
    std::cout << "M: Regresar banderines a posición normal" << std::endl;
    std::cout << "G: Abrir/Cerrar grifo (agua)" << std::endl;
    std::cout << "F: Activar/Desactivar fuegos artificiales" << std::endl;
    std::cout << "1/2/3: Encender/Apagar lámparas de cocina" << std::endl;
    std::cout << "4: Encender/Apagar luz de la Luna" << std::endl;
    std::cout << "ESC: Salir\n" << std::endl;
    // Carga de texturas
    GLuint textureArroz = LoadTexture("images/5138.jpg");
    GLuint textureBarandal = LoadTexture("images/fondo-abstracto-con-textura (1).jpg");
    GLuint textureBase = LoadTexture("images/O5X8JM0.jpg");
    GLuint textureBotes = LoadTexture("images/OE5TN00.jpg");
    GLuint textureBolsaHarinaAbierta1 = LoadTexture("images/Illustration03.jpg");
    GLuint textureBolsaHarinaAbierta = LoadTexture("images/Illustration03.jpg");
    GLuint textureBolsasHarina = LoadTexture("images/Illustration03.jpg");
    GLuint textureBoteBasura = LoadTexture("images/fondo-abstracto-con-textura.jpg");
    GLuint textureCadenaLinterna = LoadTexture("images/5178.jpg");
    GLuint textureCaja = LoadTexture("images/16.jpg");
    GLuint textureCajas2 = LoadTexture("images/16.jpg");
    GLuint textureCajasMadera = LoadTexture("images/370.jpg");
    GLuint textureCampana = LoadTexture("images/5178.jpg");
    GLuint textureCortina2 = LoadTexture("images/preview.jpg");
    GLuint textureCortina = LoadTexture("images/169180-OVUV5U-320.jpg");
    GLuint textureCortinas = LoadTexture("images/preview.jpg");
    GLuint textureCubeta = LoadTexture("images/OR6ILJ0.jpg");
    GLuint textureElectricidad = LoadTexture("images/5178.jpg");
    GLuint textureEscaleras = LoadTexture("images/3690.jpg");
    GLuint textureEstufa = LoadTexture("images/textura-de-placa-de-metal.jpg");
    GLuint textureFregadero = LoadTexture("images/fondo-abstracto-con-textura.jpg");
    GLuint textureGas = LoadTexture("images/OQV54F0.jpg");
    GLuint textureJalador = LoadTexture("images/OR6ILJ0.jpg");
    GLuint textureLamparasCocina = LoadTexture("images/fondo-abstracto-con-textura.jpg");
    GLuint textureLamparasCocina1 = LoadTexture("images/2259.jpg");
    GLuint textureLamparas1 = LoadTexture("images/2259.jpg");
    GLuint textureLamparas = LoadTexture("images/textura-de-placa-de-metal.jpg");
    GLuint textureLibrero = LoadTexture("images/3690.jpg");
    GLuint textureLuna = LoadTexture("images/2259.jpg");
    GLuint textureLinternaLocal = LoadTexture("images/2259.jpg");
    GLuint textureMesa = LoadTexture("images/OQV54F0.jpg");
    GLuint textureOlla = LoadTexture("images/5178.jpg");
    GLuint textureOlla1 = LoadTexture("images/5178.jpg");
    GLuint textureOlla2 = LoadTexture("images/acuarela-de-oro-liquido-espacio-de-copia.jpg");
    GLuint textureOlla3 = LoadTexture("images/acuarela-de-oro-liquido-espacio-de-copia.jpg");
```

```
    GLuint texturePasto = LoadTexture("images/preview1.jpg");
    GLuint texturePared3 = LoadTexture("images/3690.jpg");
    GLuint texturePared4 = LoadTexture("images/3690.jpg");
    GLuint texturePared = LoadTexture("images/18366.jpg");
    GLuint texturePlatos = LoadTexture("images/OIVY360.jpg");
    GLuint texturePiso2 = LoadTexture("images/8469.jpg");
    GLuint texturePrimerPiso = LoadTexture("images/liquid-marbling-paint-texture-background-fluid
    GLuint texturePuertas = LoadTexture("images/3690.jpg");
    GLuint textureRefrigerador = LoadTexture("images/OIVY360.jpg");
    GLuint textureRefrigerador1 = LoadTexture("images/OIVY360.jpg");
    GLuint textureSilla2 = LoadTexture("images/fondo-abstracto-con-textura.jpg");
    GLuint textureSilla = LoadTexture("images/textura-grunge-roja.jpg");
    GLuint textureTapa = LoadTexture("images/OR6ILJ0.jpg");
    GLuint textureTapa1 = LoadTexture("images/OR6ILJ0.jpg");
    GLuint textureTapa2 = LoadTexture("images/OR6ILJ0.jpg");
    GLuint textureTanqueAgua = LoadTexture("images/fondo-abstracto-con-textura.jpg");
    GLuint textureTapeteAlmacen = LoadTexture("images/47074.jpg");
    GLuint textureTapeteCocina = LoadTexture("images/abstract-blur-empty-green-gradient-studio-we
    GLuint textureTazon = LoadTexture("images/OIVY360.jpg");
    GLuint textureTazones2 = LoadTexture("images/OIVY360.jpg");
    GLuint textureTejado = LoadTexture("images/6593778.jpg");
    GLuint textureTejadoMini = LoadTexture("images/3690.jpg");
    GLuint textureTejadoMiniPiso2 = LoadTexture("images/3690.jpg");
    GLuint textureTuboFuera = LoadTexture("images/5178.jpg");
    GLuint textureTuboGas = LoadTexture("images/5178.jpg");
    GLuint textureVentanaSegundoPiso = LoadTexture("images/3690.jpg");
    GLuint textureBanderines = LoadTexture("images/preview.jpg");

    // Carga de modelos
    Model Arroz((char*)"Models/Arroz.obj");
    Model Barandal((char*)"Models/Barandal.obj");
    Model Base((char*)"Models/Base.obj");
    Model Botes((char*)"Models/Botes.obj");
    Model BolsaHarinaAbierta1((char*)"Models/BolsaHarinaAbierta1.obj");
    Model BolsaHarinaAbierta((char*)"Models/BolsaHarinaAbierta.obj");
    Model BolsasHarina((char*)"Models/BolsasHarina.obj");
    Model BoteBasura((char*)"Models/BoteBasura.obj");
    Model CadenaLinterna((char*)"Models/CadenaLinterna.obj");
    Model Caja((char*)"Models/Caja.obj");
    Model Cajas2((char*)"Models/Cajas2.obj");
    Model CajasMadera((char*)"Models/CajasMadera.obj");
    Model Campana((char*)"Models/Campana.obj");
    Model Cortina2((char*)"Models/Cortina2.obj");
```

```
Model Cortina((char*)"Models/Cortina.obj");
Model Cortinas((char*)"Models/Cortinas.obj");
Model Cubeta((char*)"Models/Cubeta.obj");
Model Electricidad((char*)"Models/Electricidad.obj");
Model Escaleras((char*)"Models/Escaleras.obj");
Model Estufa((char*)"Models/Estufa.obj");
Model Fregadero((char*)"Models/Fregadero.obj");
Model Gas((char*)"Models/Gas.obj");
Model Jalador((char*)"Models/Jalador.obj");
Model LamparasCocina((char*)"Models/LamparasCocina.obj");
Model LamparasCocina1((char*)"Models/LamparasCocina1.obj");
Model Librero((char*)"Models/Librero.obj");
Model Luna((char*)"Models/Luna.obj");
Model LinternaLocal((char*)"Models/LinternaLocal.obj");
Model Mesa((char*)"Models/Mesa.obj");
Model Olla((char*)"Models/Olla.obj");
Model Olla1((char*)"Models/Olla1.obj");
Model Olla2((char*)"Models/Olla2.obj");
Model Olla3((char*)"Models/Olla3.obj");
Model Pasto((char*)"Models/Pasto.obj");
Model Pared3((char*)"Models/Pared3.obj");
Model Pared4((char*)"Models/Pared4.obj");
Model Pared((char*)"Models/Pared.obj");
Model Platos((char*)"Models/Platos.obj");
Model Piso2((char*)"Models/Piso2.obj");
Model PrimerPiso((char*)"Models/PrimerPiso.obj");
Model Puertas((char*)"Models/Puertas.obj");
Model Refrigerador((char*)"Models/Refrigerador.obj");
Model Refrigerador1((char*)"Models/Refrigerador1.obj");
Model Silla2((char*)"Models/Silla2.obj");
Model Silla((char*)"Models/Silla.obj");
Model Tapa((char*)"Models/Tapa.obj");
Model Tapa1((char*)"Models/Tapa1.obj");
Model Tapa2((char*)"Models/Tapa2.obj");
Model TanqueAgua((char*)"Models/TanqueAgua.obj");
Model TapeteAlmacen((char*)"Models/TapeteAlmacen.obj");
Model TapeteCocina((char*)"Models/TapeteCocina.obj");
Model Tazon((char*)"Models/Tazon.obj");
Model Tazones2((char*)"Models/Tazones2.obj");
Model Tejado((char*)"Models/Tejado.obj");
Model TejadoMini((char*)"Models/TejadoMini.obj");
Model TejadoMiniPiso2((char*)"Models/TejadoMiniPiso2.obj");
Model TuboFuera((char*)"Models/TuboFuera.obj");
```

```
Model TuboGas((char*)"Models/TuboGas.obj");
Model VentanaSegundoPiso((char*)"Models/VentanaSegundoPiso.obj");
Model Banderines((char*)"Models/Banderines.obj");

glm::mat4 projection = glm::perspective(glm::radians(45.0f), (GLfloat)screenWidth / (GLfloat)screenHeight, 0.1f, 100.0f);

// Crear geometrías personalizadas
GLuint textureReloj = LoadTexture("images/OIVV360.jpg");
GLuint textureManecilla = LoadTexture("images/fondo-abstracto-con-textura.jpg");
GLuint textureCubo = LoadTexture("images/16.jpg");
GLuint textureMesaCocina = LoadTexture("images/3690.jpg");
GLuint textureNarutomaki = LoadTexture("images/OIVV360.jpg");
GLuint textureSarten = LoadTexture("images/fondo-abstracto-con-textura.jpg");
GLuint textureTabla = LoadTexture("images/fondo-de-diseno-de-textura-de-madera-de-roble.jpg");
GLuint textureCuchilloHoja = LoadTexture("images/fondo-abstracto-con-textura.jpg");
GLuint textureCuchilloMango = LoadTexture("images/3690.jpg");
GLuint textureAgua = LoadTexture("images/papel-pintado-azul-monocromatico-minimalista.jpg");
GLuint textureParticula = LoadTexture("images/acuarela-de-oro-liquido-espacio-de-copia.jpg"); // Textura brillante

SimpleGeometry* ParticulaFuego = CreateParticulaFuego(textureParticula);
SimpleGeometry* RelojCirculo = CreateCircle(textureReloj, 0.5f, 60);
SimpleGeometry* ManecillaHora = CreateClockHand(textureManecilla, 0.05f, 0.3f);
SimpleGeometry* ManecillaMinuto = CreateClockHand(textureManecilla, 0.03f, 0.45f);
SimpleGeometry* Cubo = CreateCube(textureCubo);
SimpleGeometry* MesaTop = CreateMesaTop(textureMesaCocina);
SimpleGeometry* Pata = CreatePataMesa(textureMesaCocina);
SimpleGeometry* Narutomaki = CreateNarutomaki(textureNarutomaki);
SimpleGeometry* Sarten = CreateSarten(textureSarten);
SimpleGeometry* TablaPicar = CreateTablaPicar(textureTabla);
SimpleGeometry* HojaCuchillo = CreateHojaCuchillo(textureCuchilloHoja);
SimpleGeometry* MangoCuchillo = CreateMangoCuchillo(textureCuchilloMango);
SimpleGeometry* GotaAgua = CreateGotaAgua(textureAgua);

// Inicializar el vector de gotas
gotas.resize(MAX_GOTAS);
for (int i = 0; i < MAX_GOTAS; i++) {
    gotas[i].activa = false;
    gotas[i].velocidad = 2.0f;
}
// Inicializar partículas de fuegos artificiales
particulas.resize(MAX_PARTICULAS);
for (int i = 0; i < MAX_PARTICULAS; i++) {
    particulas[i].activa = false;
```

Este código representa el bucle principal de renderizado que se ejecuta en cada frame. Calcula el tiempo delta entre frames, limpia la pantalla y configura el sistema de iluminación (luz direccional, 4 luces puntuales con estados encendido/apagado, y una luz spot). Luego renderiza todos los objetos de la escena en orden: primero dibuja las partículas de fuegos artificiales si están activas usando blending para transparencia, después renderiza todos los modelos 3D estáticos de la cocina (arroz, muebles, electrodomésticos, decoraciones) aplicando sus respectivas texturas y valores de brillo, y finalmente dibuja las geometrías personalizadas animadas como el reloj con manecillas giratorias, un cubo, una mesa con cuatro patas, banderines controlables manualmente, tabla de picar, mantel, queso, un cuchillo con hoja y mango, y gotas de agua animadas que caen del grifo cuando está activado, todas con sus transformaciones matriciales correspondientes.

```
    }
    srand(time(NULL)); // Para números aleatorios
    glm::vec3 posReloj = glm::vec3(-2.0f, 2.0f, 0.15f);
    glm::vec3 posCaja = glm::vec3(1.50f, 4.80f, -0.30f);
    glm::vec3 posMesa = glm::vec3(0.80f, 5.0f, -2.40f);
    glm::vec3 posTabla = glm::vec3(-1.50f, 1.17f, 1.5f);
    glm::vec3 posSarten = glm::vec3(0.80f, 5.05f, -2.40f);

    while (!glfwWindowShouldClose(window))
    {
        currentFrame = (GLfloat)glfwGetTime();
        deltaTime = currentFrame - lastFrame;
        lastFrame = currentFrame;

        glfwPollEvents();
        ProcessMovement();

        glClearColor(0.05f, 0.05f, 0.15f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        lightingShader.Use();

        GLint viewPosLoc = glGetUniformLocation(lightingShader.Program, "viewPos");
        glm::vec3 cameraPos = camera.GetPosition();
        glUniform3f(viewPosLoc, cameraPos.x, cameraPos.y, cameraPos.z);

        glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.05f, 0.05f, 0.1f);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.1f, 0.1f, 0.15f);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.2f, 0.2f, 0.3f);

        for (int i = 0; i < 4; ++i) {
            std::string prefix = "pointLights[" + std::to_string(i) + "]";

            glm::vec3 ambient, diffuse, specular;
            float constant, linear, quadratic;

            if (i == 3) {
                ambient = lightStates[i] ? glm::vec3(0.2f, 0.2f, 0.3f) : glm::vec3(0.0f);
                diffuse = lightStates[i] ? glm::vec3(0.8f, 0.8f, 1.2f) : glm::vec3(0.0f);
                specular = lightStates[i] ? glm::vec3(1.0f, 1.0f, 1.3f) : glm::vec3(0.0f);
                constant = 1.0f;
                linear = 0.014f;
                quadratic = 0.0007f;
            }
            else {
                ambient = lightStates[i] ? glm::vec3(0.15f, 0.05f, 0.0f) : glm::vec3(0.0f);       // tenue
                diffuse = lightStates[i] ? glm::vec3(1.0f, 0.4f, 0.1f) : glm::vec3(0.0f);         // naranja-rojizo "chakra"
                specular = lightStates[i] ? glm::vec3(1.0f, 0.6f, 0.2f) : glm::vec3(0.0f);        // brillo cálido

                // Atenuación suave para que no se expanda demasiado
                constant = 1.0f;
                linear = 0.07f;
                quadratic = 0.017f;
            }

            glUniform3f(glGetUniformLocation(lightingShader.Program, (prefix + ".position").c_str()),
                pointLightPositions[i].x, pointLightPositions[i].y, pointLightPositions[i].z);
            glUniform3f(glGetUniformLocation(lightingShader.Program, (prefix + ".ambient").c_str()),
                ambient.x, ambient.y, ambient.z);
            glUniform3f(glGetUniformLocation(lightingShader.Program, (prefix + ".diffuse").c_str()),
                diffuse.x, diffuse.y, diffuse.z);
            glUniform3f(glGetUniformLocation(lightingShader.Program, (prefix + ".specular").c_str()),
                specular.x, specular.y, specular.z);
            glUniform1f(glGetUniformLocation(lightingShader.Program, (prefix + ".constant").c_str()), constant);
            glUniform1f(glGetUniformLocation(lightingShader.Program, (prefix + ".linear").c_str()), linear);
            glUniform1f(glGetUniformLocation(lightingShader.Program, (prefix + ".quadratic").c_str()), quadratic);
        }

        // Inicializar spotLight (apagado) - IMPORTANTE para que el shader funcione
        glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.position"), 0.0f, 0.0f, 0.0f);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.direction"), 0.0f, -1.0f, 0.0f);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.cutOff"), glm::cos(glm::radians(12.5f)));
        glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.outerCutOff"), glm::cos(glm::radians(15.0f)));
        glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.ambient"), 0.0f, 0.0f, 0.0f);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.diffuse"), 0.0f, 0.0f, 0.0f);
        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"), 0.09f);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.quadratic"), 0.032f);

        glm::mat4 view = camera.GetViewMatrix();
        glm::mat4 model = glm::mat4(1);

        GLint modelLoc = glGetUniformLocation(lightingShader.Program, "model");

        glUniformMatrix4fv(glGetUniformLocation(lightingShader.Program, "projection"), 1, GL_FALSE, glm::value_ptr(projection));
        glUniformMatrix4fv(glGetUniformLocation(lightingShader.Program, "view"), 1, GL_FALSE, glm::value_ptr(view));

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

        // IMPORTANTE: Configurar los samplers de textura
        glUniform1i(glGetUniformLocation(lightingShader.Program, "material.diffuse"), 0);
        glUniform1i(glGetUniformLocation(lightingShader.Program, "material.specular"), 0);

        // ===== PRIMERO: FUEGOS ARTIFICIALES (para que estén detrás) =====
        if (fuegosActivos) {
            glEnable(GL_BLEND);
            glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
            // NO desactives el depth test aquí para que respeten la profundidad
            // glDisable(GL_DEPTH_TEST); <-- QUITA ESTA LÍNEA

            for (int i = 0; i < MAX_PARTICULAS; i++) {
                if (particulas[i].activa) {
                    model = glm::mat4(1.0f);
                    model = glm::translate(model, particulas[i].posicion);

                    float factorVida = particulas[i].vida / particulas[i].vidaInicial;
                    float escala = particulas[i].tamano * factorVida;
                    model = glm::scale(model, glm::vec3(escala));

                    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
                    glBindTexture(GL_TEXTURE_2D, texturaParticula);
                    glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 128.0f);

                    ParticulaFuego->Draw();
                }
            }

            // glEnable(GL_DEPTH_TEST); <-- QUITA ESTA LÍNEA TAMBIÉN
            glDisable(GL_BLEND);
        }
        // Renderizado de objetos opacos
        model = glm::mat4(1.0f);
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));

        // Bind texture y dibujar Arroz
        glActiveTexture(GL_TEXTURE0);
        glBindTexture(GL_TEXTURE_2D, textureArroz);
```

```
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 48.0f);
Arroz.Draw(lightingShader);

// Base
glBindTexture(GL_TEXTURE_2D, textureBase);
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 32.0f);
Base.Draw(lightingShader);

// Barandal
glBindTexture(GL_TEXTURE_2D, textureBarandal);
Barandal.Draw(lightingShader);

// Botes
glBindTexture(GL_TEXTURE_2D, textureBotes);
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 24.0f);
Botes.Draw(lightingShader);

// BolsaHarinaAbierta
glBindTexture(GL_TEXTURE_2D, textureBolsaHarinaAbierta);
BolsaHarinaAbierta.Draw(lightingShader);

// BolsaHarinaAbierta1
glBindTexture(GL_TEXTURE_2D, textureBolsaHarinaAbierta1);
BolsaHarinaAbierta1.Draw(lightingShader);

// BolsasHarina
glBindTexture(GL_TEXTURE_2D, textureBolsasHarina);
BolsasHarina.Draw(lightingShader);

// BoteBasura
glBindTexture(GL_TEXTURE_2D, textureBoteBasura);
BoteBasura.Draw(lightingShader);

// CadenaLinterna
glBindTexture(GL_TEXTURE_2D, textureCadenaLinterna);
CadenaLinterna.Draw(lightingShader);

// Caja
glBindTexture(GL_TEXTURE_2D, textureCaja);
Caja.Draw(lightingShader);

// Cajas2
glBindTexture(GL_TEXTURE_2D, textureCajas2);
Cajas2.Draw(lightingShader);

// CajasMadera
glBindTexture(GL_TEXTURE_2D, textureCajasMadera);
CajasMadera.Draw(lightingShader);

// Campana
glBindTexture(GL_TEXTURE_2D, textureCampana);
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 56.0f);
Campana.Draw(lightingShader);

// Cortina
glBindTexture(GL_TEXTURE_2D, textureCortina);
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
Cortina.Draw(lightingShader);

// Cortinas
glBindTexture(GL_TEXTURE_2D, textureCortinas);
Cortinas.Draw(lightingShader);

// Cortina2
glBindTexture(GL_TEXTURE_2D, textureCortina2);
Cortina2.Draw(lightingShader);

// Cubeta
glBindTexture(GL_TEXTURE_2D, textureCubeta);
Cubeta.Draw(lightingShader);

// Electricidad
glBindTexture(GL_TEXTURE_2D, textureElectricidad);
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 40.0f);
Electricidad.Draw(lightingShader);

// Escaleras
glBindTexture(GL_TEXTURE_2D, textureEscaleras);
Escaleras.Draw(lightingShader);

// Estufa
glBindTexture(GL_TEXTURE_2D, textureEstufa);
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 64.0f);
Estufa.Draw(lightingShader);

// Fregadero
glBindTexture(GL_TEXTURE_2D, textureFregadero);
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 72.0f);
Fregadero.Draw(lightingShader);

// Gas
glBindTexture(GL_TEXTURE_2D, textureGas);
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 48.0f);
Gas.Draw(lightingShader);

// Jalador
glBindTexture(GL_TEXTURE_2D, textureJalador);
Jalador.Draw(lightingShader);


// LamparasCocina
glBindTexture(GL_TEXTURE_2D, textureLamparasCocina);
LamparasCocina.Draw(lightingShader);

// Librero
glBindTexture(GL_TEXTURE_2D, textureLibrero);
Librero.Draw(lightingShader);

// Mesa
glBindTexture(GL_TEXTURE_2D, textureMesa);
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 40.0f);
Mesa.Draw(lightingShader);

// Olla
glBindTexture(GL_TEXTURE_2D, textureOlla);
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 96.0f);
Olla.Draw(lightingShader);

// Olla1
glBindTexture(GL_TEXTURE_2D, textureOlla1);
Olla1.Draw(lightingShader);

// Olla2
glBindTexture(GL_TEXTURE_2D, textureOlla2);
Olla2.Draw(lightingShader);

// Olla3
glBindTexture(GL_TEXTURE_2D, textureOlla3);
```

```
        Olla3.Draw(lightingShader);

        // Pasto
        glBindTexture(GL_TEXTURE_2D, texturePasto);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 8.0f);
        Pasto.Draw(lightingShader);

        // Pared3
        glBindTexture(GL_TEXTURE_2D, texturePared3);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 24.0f);
        Pared3.Draw(lightingShader);

        // Pared4
        glBindTexture(GL_TEXTURE_2D, texturePared4);
        Pared4.Draw(lightingShader);

        // Pared
        glBindTexture(GL_TEXTURE_2D, texturePared);
        Pared.Draw(lightingShader);

        // Platos
        glBindTexture(GL_TEXTURE_2D, texturePlatos);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 56.0f);
        Platos.Draw(lightingShader);

        // PrimerPiso
        glBindTexture(GL_TEXTURE_2D, texturePrimerPiso);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 32.0f);
        PrimerPiso.Draw(lightingShader);

        // Piso2
        glBindTexture(GL_TEXTURE_2D, texturePiso2);
        Piso2.Draw(lightingShader);

        // Puertas
        glBindTexture(GL_TEXTURE_2D, texturePuertas);
        Puertas.Draw(lightingShader);

        // Refrigerador
        glBindTexture(GL_TEXTURE_2D, textureRefrigerador);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 80.0f);
        Refrigerador.Draw(lightingShader);
```

```
        // Silla2
        glBindTexture(GL_TEXTURE_2D, textureSilla2);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 20.0f);
        Silla2.Draw(lightingShader);

        // Silla
        glBindTexture(GL_TEXTURE_2D, textureSilla);
        Silla.Draw(lightingShader);

        // Tapa
        glBindTexture(GL_TEXTURE_2D, textureTapa);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 48.0f);
        Tapa.Draw(lightingShader);

        // Tapa1
        glBindTexture(GL_TEXTURE_2D, textureTapa1);
        Tapa1.Draw(lightingShader);

        // Tapa2
        glBindTexture(GL_TEXTURE_2D, textureTapa2);
        Tapa2.Draw(lightingShader);

        // TapeteAlmacen
        glBindTexture(GL_TEXTURE_2D, textureTapeteAlmacen);
        TapeteAlmacen.Draw(lightingShader);

        // TanqueAgua
        glBindTexture(GL_TEXTURE_2D, textureTanqueAgua);
        TanqueAgua.Draw(lightingShader);

        // TapeteCocina
        glBindTexture(GL_TEXTURE_2D, textureTapeteCocina);
        TapeteCocina.Draw(lightingShader);

        // Tazon
        glBindTexture(GL_TEXTURE_2D, textureTazon);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 64.0f);
        Tazon.Draw(lightingShader);

        // Tazones2
        glBindTexture(GL_TEXTURE_2D, textureTazones2);
        Tazones2.Draw(lightingShader);
```

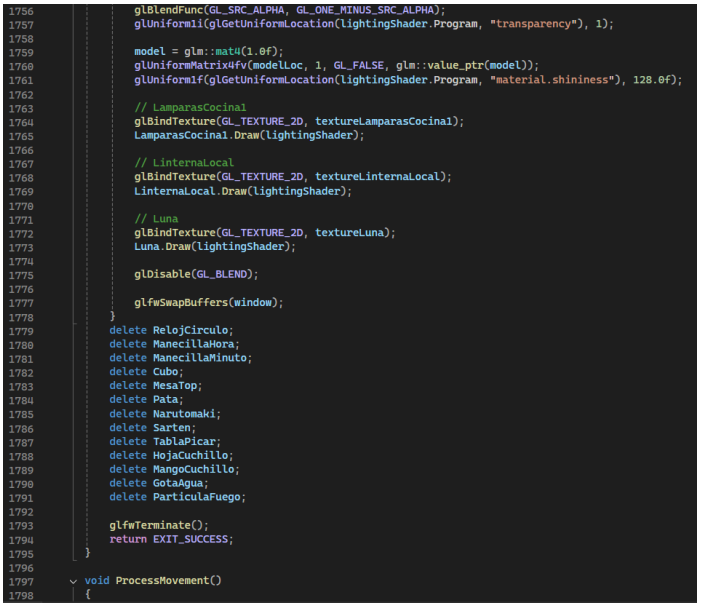
```
        // Tejado
        glBindTexture(GL_TEXTURE_2D, textureTejado);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
        Tejado.Draw(lightingShader);

        // TejadoMini
        glBindTexture(GL_TEXTURE_2D, textureTejadoMini);
        TejadoMini.Draw(lightingShader);

        // TejadoMiniPiso2
        glBindTexture(GL_TEXTURE_2D, textureTejadoMiniPiso2);
        TejadoMiniPiso2.Draw(lightingShader);

        // TuboFuera
        glBindTexture(GL_TEXTURE_2D, textureTuboFuera);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 80.0f);
        TuboFuera.Draw(lightingShader);

        // TuboGas
        glBindTexture(GL_TEXTURE_2D, textureTuboGas);
        TuboGas.Draw(lightingShader);

        // VentanaSegundoPiso
        glBindTexture(GL_TEXTURE_2D, textureVentanaSegundoPiso);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 24.0f);
        VentanaSegundoPiso.Draw(lightingShader);

        // ===== BANDERINES CON CONTROL MANUAL (ADELANTE/ATRÁS) =====
        model = glm::mat4(1.0f);

        // La rotación se aplica en el eje X para mover la punta hacia adelante/atrás
        model = glm::rotate(model, glm::radians(banderinesInclinacion), glm::vec3(1.0f, 0.0f, 0.0f));

        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        glBindTexture(GL_TEXTURE_2D, textureBanderines);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
        Banderines.Draw(lightingShader);

        // ===== RELOJ =====
        model = glm::mat4(1.0f);
        model = glm::translate(model, posReloj);
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 32.0f);
```

```
1627            RelojCirculo->Draw();
1628
1629            // Manecilla de hora (gira lento)
1630            float horaAngulo = (float)glfwGetTime() * 0.1f;
1631            model = glm::translate(glm::mat4(1.0f), posReloj);
1632            model = glm::rotate(model, horaAngulo, glm::vec3(0.0f, 0.0f, -1.0f));
1633            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1634            ManecillaHora->Draw();
1635
1636            // Manecilla de minuto (gira rápido)
1637            float minutoAngulo = (float)glfwGetTime() * 0.5f;
1638            model = glm::translate(glm::mat4(1.0f), posReloj);
1639            model = glm::rotate(model, minutoAngulo, glm::vec3(0.0f, 0.0f, -1.0f));
1640            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1641            ManecillaMinuto->Draw();
1642
1643            // ===== CUBO (CAJA) =====
1644            model = glm::mat4(1.0f);
1645            model = glm::translate(model, posCaja);
1646            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1647            glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
1648            Cubo->Draw();
1649
1650            // ===== MESA CON PATAS =====
1651            // Superficie de la mesa
1652            model = glm::mat4(1.0f);
1653            model = glm::translate(model, posMesa);
1654            model = glm::scale(model, glm::vec3(1.5f, 1.0f, 1.5f));
1655            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1656            glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 48.0f);
1657            MesaTop->Draw();
1658
1659            // Pata 1 (esquina -X, -Z)
1660            model = glm::mat4(1.0f);
1661            model = glm::translate(model, posMesa + glm::vec3(-0.65f, 0.0f, -0.65f));
1662            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1663            Pata->Draw();
1664
1665            // Pata 2 (esquina +X, -Z)
1666            model = glm::mat4(1.0f);
1667            model = glm::translate(model, posMesa + glm::vec3(0.65f, 0.0f, -0.65f));
1668            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1669            Pata->Draw();
```

```
1671            // Pata 3 (esquina -X, +Z)
1672            model = glm::mat4(1.0f);
1673            model = glm::translate(model, posMesa + glm::vec3(-0.65f, 0.0f, 0.65f));
1674            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1675            Pata->Draw();
1676
1677            // Pata 4 (esquina +X, +Z)
1678            model = glm::mat4(1.0f);
1679            model = glm::translate(model, posMesa + glm::vec3(0.65f, 0.0f, 0.65f));
1680            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1681            Pata->Draw();
1682
1683            // ===== TABLA DE PICAR =====
1684            model = glm::mat4(1.0f);
1685            model = glm::translate(model, posTabla);
1686            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1687            glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 24.0f);
1688            TablaPicar->Draw();
1689
1690            // ===== SARTÉN =====
1691            model = glm::mat4(1.0f);
1692            model = glm::translate(model, posSarten);
1693            model = glm::scale(model, glm::vec3(2.8f, 1.0f, 2.8f));
1694            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1695            glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 64.0f);
1696            Sarten->Draw();
1697
1698            // ===== NARUTOMAKI (3 piezas en fila) =====
1699            for (int i = 0; i < 3; i++) {
1700                model = glm::mat4(1.0f);
1701                model = glm::translate(model, glm::vec3(-1.60f + i * 0.1f, 1.20f, 1.5f));
1702                glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1703                glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 20.0f);
1704                Narutomaki->Draw();
1705            }
1706            // ===== CUCHILLO (HOJA + MANGO) =====
1707            glm::vec3 posCuchillo = glm::vec3(-1.80f, 1.20f, 1.5f);
1708
1709            // Hoja del cuchillo (rotada 45 grados)
1710            model = glm::mat4(1.0f);
1711            model = glm::translate(model, posCuchillo);
1712            model = glm::rotate(model, glm::radians(45.0f), glm::vec3(0.0f, 1.0f, 0.0f));
1713            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
```

```
1714            glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 96.0f);
1715            HojaCuchillo->Draw();
1716
1717            // Mango del cuchillo (atrás de la hoja)
1718            model = glm::translate(model, glm::vec3(0.0f, 0.0f, -0.25f));
1719            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1720            glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
1721            MangoCuchillo->Draw();
1722
1723            // ===== GOTAS DE AGUA =====
1724            if (aguaCayendo) {
1725                glEnable(GL_BLEND);
1726                glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
1727
1728                for (int i = 0; i < MAX_GOTAS; i++) {
1729                    if (gotas[i].activa) {
1730                        model = glm::mat4(1.0f);
1731                        model = glm::translate(model, gotas[i].posicion);
1732                        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1733                        glBindTexture(GL_TEXTURE_2D, textureAgua);
1734                        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 128.0f);
1735                        GotaAgua->Draw();
1736                    }
1737                }
1738
1739                glDisable(GL_BLEND);
1740            }
1741
1742            // Animación del refrigerador
1743            glm::vec3 posBisagraRefrigerador = glm::vec3(2.5f, 1.0f, -3.60f);
1744            model = glm::mat4(1.0f);
1745            model = glm::translate(model, posBisagraRefrigerador);
1746            model = glm::rotate(model, glm::radians(-puertaRefrigerador), glm::vec3(0.0f, 1.0f, 0.0f));
1747            model = glm::translate(model, -posBisagraRefrigerador);
1748            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1749            glBindTexture(GL_TEXTURE_2D, textureRefrigerador1);
1750            glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 80.0f);
1751            Refrigerador1.Draw(lightingShader);
1752
1753
1754            // Objetos transparentes (lámparas)
1755            glEnable(GL_BLEND);
1756            glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

Este código finaliza el bucle de renderizado con tres secciones importantes: primero anima la puerta del refrigerador usando transformaciones de rotación alrededor de una bisagra (permitiendo abrirla/cerrarla según el valor de la variable puertaRefrigerador); luego activa el modo de transparencia con blending para renderizar objetos semitransparentes como las lámparas de cocina, linterna y la luna; finalmente intercambia los buffers de pantalla para mostrar el frame renderizado. Al salir del bucle principal, libera la memoria de todas las geometrías personalizadas creadas dinámicamente, termina GLFW y cierra el programa exitosamente.



```
            glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
            glUniform1i(glGetUniformLocation(lightingShader.Program, "transparency"), 1);

            model = glm::mat4(1.0f);
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
            glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 128.0f);

            // LamparasCocina1
            glBindTexture(GL_TEXTURE_2D, textureLamparasCocina1);
            LamparasCocina1.Draw(lightingShader);

            // LinternaLocal
            glBindTexture(GL_TEXTURE_2D, textureLinternaLocal);
            LinternaLocal.Draw(lightingShader);

            // Luna
            glBindTexture(GL_TEXTURE_2D, textureLuna);
            Luna.Draw(lightingShader);

            glDisable(GL_BLEND);

            glfwSwapBuffers(window);
        }
        delete RelojCirculo;
        delete ManecillaHora;
        delete ManecillaMinuto;
        delete Cubo;
        delete MesaTop;
        delete Pata;
        delete Narutomaki;
        delete Sarten;
        delete TablaPicar;
        delete HojaCuchillo;
        delete MangoCuchillo;
        delete GotaAgua;
        delete ParticulaFuego;

        glfwTerminate();
        return EXIT_SUCCESS;
    }

    void ProcessMovement()
    {
```

Esta sección de código maneja toda la interacción del usuario y las animaciones de la escena a través de tres funciones principales. ProcessMovement() actualiza continuamente el movimiento de la cámara con las teclas WASD, anima suavemente la apertura/cierre de la puerta del refrigerador, controla la inclinación de los banderines, simula gotas de agua cayendo del grifo con física de gravedad, y genera explosiones de fuegos artificiales con partículas de colores aleatorios que caen por gravedad. KeyCallback() captura las pulsaciones de teclas para: alternar el encendido/apagado de cuatro luces (teclas 1-4), abrir/cerrar la puerta del refrigerador (O/C), mover banderines hacia atrás o a su posición normal (N/M), activar/desactivar el agua del grifo (G) y los fuegos artificiales (F), además de cerrar la ventana con ESC. MouseCallback() rastrea el movimiento del mouse para controlar la dirección de visión de la cámara en primera persona.

```
        // Movimiento de cámara
        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);

        // Animación automática de la puerta
        if (animandoPuerta) {
            if (abs(puertaRefrigerador - puertaDestino) > 0.1f) {
                if (puertaRefrigerador < puertaDestino)
                    puertaRefrigerador += 80.0f * deltaTime;
                else
                    puertaRefrigerador -= 80.0f * deltaTime;
            }
            else {
                puertaRefrigerador = puertaDestino;
                animandoPuerta = false;
            }
        }

        // **ANIMACIÓN: Banderines moviéndose hacia adelante/atrás**
        if (animandoBanderines) {
            if (abs(banderinesInclinacion - banderinesDestino) > 0.1f) {
                if (banderinesInclinacion < banderinesDestino)
                    banderinesInclinacion += 60.0f * deltaTime;
                else
                    banderinesInclinacion -= 60.0f * deltaTime;
            }
            else {
                banderinesInclinacion = banderinesDestino;
                animandoBanderines = false;
            }
        }

        // **ANIMACIÓN: Agua cayendo del grifo**
        if (aguaCayendo) {
            // Crear nuevas gotas
            if (currentFrame - tiempoUltimaGota >= intervaloGotas) {
                for (int i = 0; i < MAX_GOTAS; i++) {
```

```
                    if (!gotas[i].activa) {
                        gotas[i].activa = true;
                        gotas[i].posicion = posicionGrifo;
                        gotas[i].velocidad = 2.0f + (rand() % 100) / 100.0f;
                        tiempoUltimaGota = currentFrame;
                        break;
                    }
                }
            }

            // Actualizar posición de gotas activas
            for (int i = 0; i < MAX_GOTAS; i++) {
                if (gotas[i].activa) {
                    gotas[i].posicion.y -= gotas[i].velocidad * deltaTime;
                    gotas[i].velocidad += 3.0f * deltaTime; // Gravedad

                    // Desactivar gota si llega al fregadero
                    if (gotas[i].posicion.y < 1.0f) {
                        gotas[i].activa = false;
                    }
                }
            }
        }
        // **ANIMACIÓN: Fuegos artificiales**
        if (fuegosActivos) {
            if (currentFrame - tiempoUltimoFuego >= intervaloFuegos) {
                // MÁS SEPARADAS HORIZONTALMENTE (de -8 a +8)
                glm::vec3 posExplosion = glm::vec3(
                    -8.0f + (rand() % 1600) / 100.0f,    // -8 a 8 en X (MUY ANCHO)
                    7.0f + (rand() % 200) / 100.0f,      // 7 a 9 en Y (altura similar)
                    -10.0f                               // Z FIJO (sin profundidad)
                );

                // Colores vivos y variados
                int tipoColor = rand() % 5;
                glm::vec3 colorExplosion;

                switch (tipoColor) {
                case 0: colorExplosion = glm::vec3(1.0f, 0.2f, 0.2f); break; // Rojo
                case 1: colorExplosion = glm::vec3(0.2f, 1.0f, 0.2f); break; // Verde
                case 2: colorExplosion = glm::vec3(0.2f, 0.2f, 1.0f); break; // Azul
                case 3: colorExplosion = glm::vec3(1.0f, 1.0f, 0.2f); break; // Amarillo
                case 4: colorExplosion = glm::vec3(1.0f, 0.2f, 1.0f); break; // Magenta
```

```
                }

                CrearExplosion(posExplosion, colorExplosion);
                tiempoUltimoFuego = currentFrame;

                std::cout << "¡BOOM! Explosión en el fondo Z=" << posExplosion.z << std::endl;
            }

            // Actualizar partículas activas
            for (int i = 0; i < MAX_PARTICULAS; i++) {
                if (particulas[i].activa) {
                    // Actualizar posición
                    particulas[i].posicion += particulas[i].velocidad * deltaTime;

                    // Aplicar gravedad suave
                    particulas[i].velocidad.y -= 2.5f * deltaTime;

                    // Fricción del aire
                    particulas[i].velocidad *= 0.98f;

                    // Reducir vida
                    particulas[i].vida -= deltaTime;

                    // Desactivar si ya no tiene vida
                    if (particulas[i].vida <= 0.0f) {
                        particulas[i].activa = false;
                    }
                }
            }
        }
    }
    void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode)
    {
        (void)scancode;
        (void)mode;

        if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
            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 (action == GLFW_PRESS)
    {
        switch (key)
        {
        case GLFW_KEY_1:
            lightStates[0] = !lightStates[0];
            std::cout << "Luz 1 (LamparasCocina1): " << (lightStates[0] ? "ENCENDIDA" : "APAGADA") << std::endl;
            break;
        case GLFW_KEY_2:
            lightStates[1] = !lightStates[1];
            std::cout << "Luz 2 (Lamparas1): " << (lightStates[1] ? "ENCENDIDA" : "APAGADA") << std::endl;
            break;
        case GLFW_KEY_3:
            lightStates[2] = !lightStates[2];
            std::cout << "Luz 3 (LinternaLocal): " << (lightStates[2] ? "ENCENDIDA" : "APAGADA") << std::endl;
            break;
        case GLFW_KEY_4:
            lightStates[3] = !lightStates[3];
            std::cout << "Luz Luna: " << (lightStates[3] ? "ENCENDIDA" : "APAGADA") << std::endl;
            break;
        case GLFW_KEY_O:
            puertaDestino = 90.0f;  // Abrir
            animandoPuerta = true;
            std::cout << "Abriendo puerta del refrigerador..." << std::endl;
            break;
        case GLFW_KEY_C:
            puertaDestino = 0.0f;   // Cerrar
            animandoPuerta = true;
            std::cout << "Cerrando puerta del refrigerador..." << std::endl;
            break;
        case GLFW_KEY_N: // Mover banderines hacia ATRÁS
            banderinesDestino = -banderinesMaxInclinacion;
            animandoBanderines = true;
            std::cout << "Banderines moviéndose hacia atrás..." << std::endl;
            break;

        case GLFW_KEY_M: // Regresar banderines a posición NORMAL
            banderinesDestino = 0.0f;
            animandoBanderines = true;
```

```
            std::cout << "Banderines regresando a posición normal..." << std::endl;
            break;
        case GLFW_KEY_G: // Toggle agua del grifo
            aguaCayendo = !aguaCayendo;
            std::cout << "Agua del grifo: " << (aguaCayendo ? "ABIERTA" : "CERRADA") << std::endl;
            if (!aguaCayendo) {
                for (int i = 0; i < MAX_GOTAS; i++) {
                    gotas[i].activa = false;
                }
            }
            break;

        case GLFW_KEY_F: // Toggle fuegos artificiales
            fuegosActivos = !fuegosActivos;
            std::cout << "Fuegos artificiales: " << (fuegosActivos ? "ACTIVADOS" : "DESACTIVADOS") << std::endl;
            if (!fuegosActivos) {
                for (int i = 0; i < MAX_PARTICULAS; i++) {
                    particulas[i].activa = false;
                }
            }
            break;
        }
    }
}

void MouseCallback(GLFWwindow* window, double xpos, double ypos)
{
    (void)window;

    if (firstMouse)
    {
        lastX = (GLfloat)xpos;
        lastY = (GLfloat)ypos;
        firstMouse = false;
    }

    GLfloat xOffset = (GLfloat)(xpos - lastX);
    GLfloat yOffset = (GLfloat)(lastY - ypos);

    lastX = (GLfloat)xpos;
    lastY = (GLfloat)ypos;

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

Resultados

El proyecto logró cumplir los objetivos planteados, desarrollando una experiencia de recorrido 3D interactivo basada en el restaurante Ichiraku de la serie *Naruto*. Gracias al uso de Blender para el modelado y OpenGL para la programación, se logró integrar de manera efectiva la parte artística con la técnica, obteniendo un entorno visualmente atractivo, funcional y dinámico.

Entre los principales resultados obtenidos se encuentran:

- **Modelado 3D detallado:** Se crearon los diferentes elementos del restaurante, incluyendo mobiliario, decoración y objetos característicos, logrando una representación fiel al entorno original del anime, este fue realizado en blender.

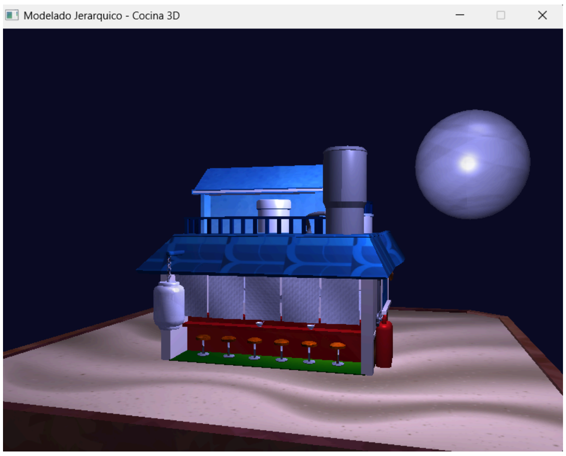
- **Aplicación de texturas y materiales:** Cada modelo cuenta con texturas adecuadas y materiales que simulan correctamente la apariencia de madera, cerámica y otros elementos presentes en el escenario.

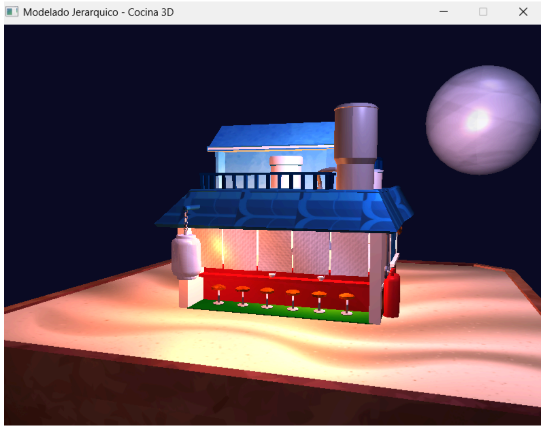
- **Iluminación realista:** Se implementaron diversos tipos de luces mediante shaders, logrando un ambiente cálido y coherente con la atmósfera del restaurante Ichiraku.

- **Animaciones integradas:** Se incorporaron animaciones básicas en algunos objetos del recorrido, fueron 5, la simulación de caída de agua del fregadero, el abrir y cerrar de la puerta del refrigerador, un reloj, subir la cortina y fuegos artificiales . Estas animaciones permitieron dar vida al escenario y mejorar la sensación de interactividad.

- **Interactividad y recorrido:** El usuario puede explorar el escenario de manera libre, recorriendo los diferentes espacios y observando los detalles de los modelos desde múltiples ángulos.

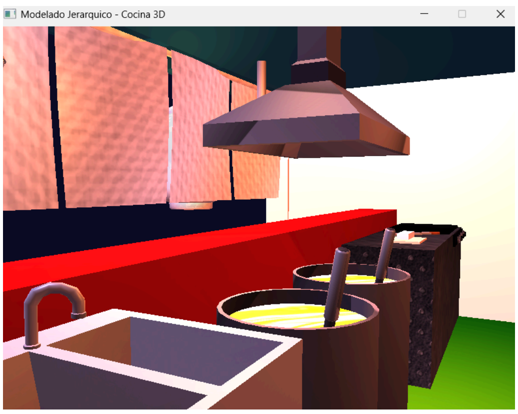
- **Integración de librerías y optimización:** Se utilizaron librerías como GLEW, GLFW y SOIL para facilitar la programación y la gestión de texturas, logrando un rendimiento estable y fluido del recorrido.


Modelado Jerarquico - Cocina 3D


Modelado Jerarquico - Cocina 3D

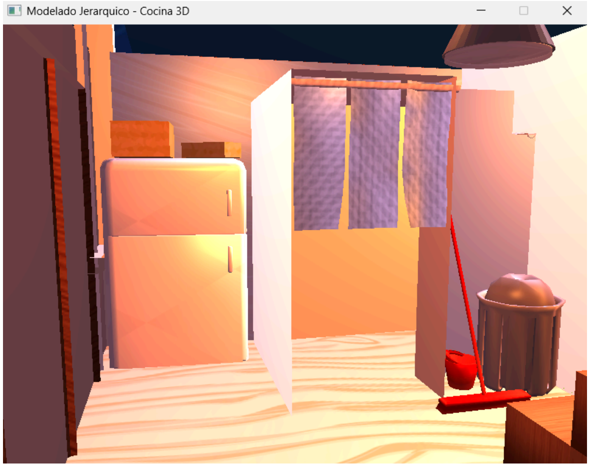

Modelado Jerarquico - Cocina 3D

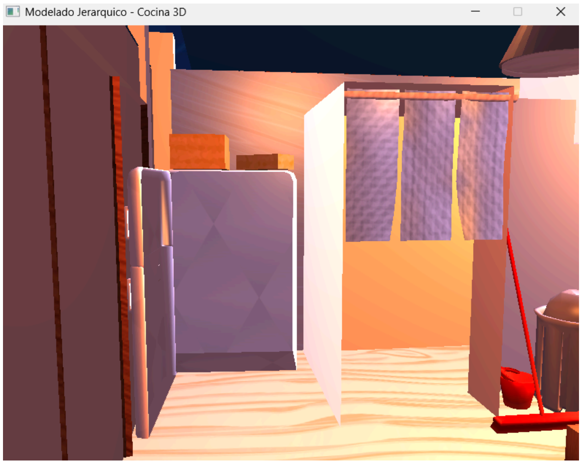

Modelado Jerarquico - Cocina 3D

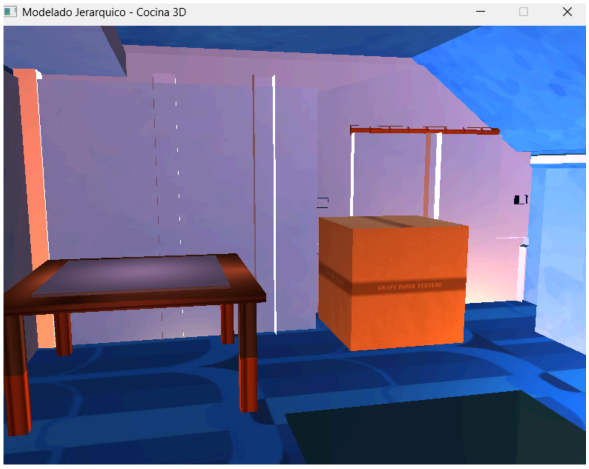

Modelado Jerarquico - Cocina 3D

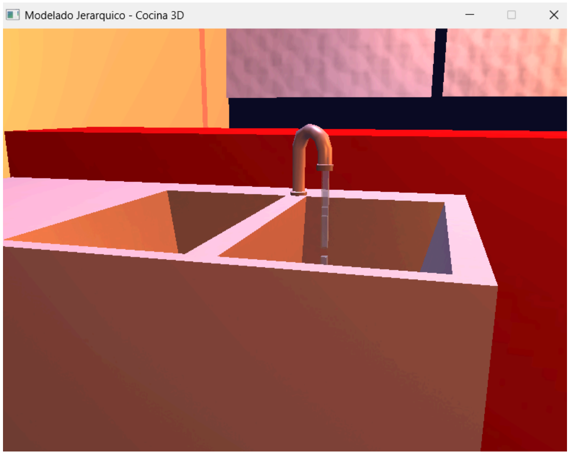

Modelado Jerarquico - Cocina 3D

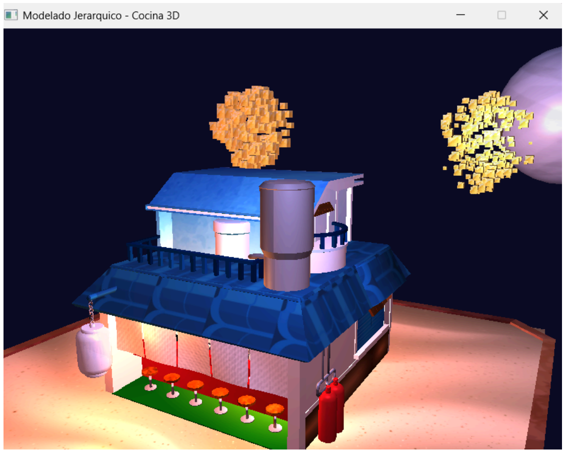

Modelado Jerarquico - Cocina 3D


Modelado Jerarquico - Cocina 3D


Modelado Jerarquico - Cocina 3D


Modelado Jerarquico - Cocina 3D


Modelado Jerarquico - Cocina 3D


Modelado Jerarquico - Cocina 3D

- **LINK DE ARCHIVOS: https://github.com/samgarciagallegos-cloud/319181386_PROYECTOFINAL2026-1_GPO-5**

Conclusiones

Mediante la realización de este proyecto aprendí a combinar modelado 3D en Blender con programación gráfica en OpenGL, aplicando texturas, iluminación y animaciones para crear recorridos interactivos, al mismo tiempo que desarrollé habilidades técnicas y creativas en diseño y desarrollo de entornos tridimensionales. Con esto creo que se concluye y se reafirma todo lo aprendido en clase de Gráfica.

Bibliografía

- Blender Foundation. (2023). *Blender 3D: Noob to Pro*. https://docs.blender.org/manual/es/latest/
- Eberly, D. H. (2010). *3D game engine design: A practical approach to real-time computer graphics* (2.ª ed.). Morgan Kaufmann.
- Foley, J. D., van Dam, A., Feiner, S. K., & Hughes, J. F. (2014). *Computer graphics: Principles and practice* (3.ª ed.). Addison-Wesley.
- GLFW. (2023). *GLFW – A library for OpenGL, OpenGL ES and Vulkan development on the desktop*. https://www.glfw.org
- Kilgard, M. J. (2009). *OpenGL programming for the X Window System*. Addison-Wesley.
- Shreiner, D., Sellers, G., Kessenich, J., & Licea-Kane, B. (2013). *OpenGL programming guide: The official guide to learning OpenGL, Version 4.3* (8.ª ed.). Addison-Wesley.
- SOIL – Simple OpenGL Image Library. (2023). http://www.lonesock.net/soil.html

Objectives

General Objective

To create an interactive virtual tour in OpenGL 3, allowing the user to realistically explore two rooms and a three-dimensional façade, integrating a synthetic camera, animations, textures, and lighting effects, in order to apply the principles of computer graphics and human-computer interaction in a functional and visually coherent 3D environment.

Specific Objectives

- Model two rooms and a façade with a defined style.
- Implement an interactive synthetic camera.
- Integrate four animations into the 3D environment.
- Apply realistic lighting and textures in OpenGL.
- Optimize project resources and routes.
- Prepare bilingual technical and user documentation.
- Conduct a project cost analysis.
- Publish the functional project on GitHub.

Goals

- Create a functional virtual tour in OpenGL 3 with a synthetic camera and animations.
- Achieve a realistic visual environment in two rooms and a façade.
- Ensure the user can freely interact with the environment.
- Deliver a stable and optimized project executable.
- Fully document the development in Spanish and English.
- Meet all requested technical and presentation requirements.

Scope

- The project will allow you to explore two rooms and a facade in 3D.
- It will include a synthetic camera with movement and rotation controls.
- It will feature at least five interactive animations.
- It will use Phong-style materials, textures, and lighting for greater realism.
- The technical development, user experience, and costs of the project will be documented.
- The code and executable will be available on GitHub with relative paths.

Limitations

During project development, several limitations arose that influenced the process of creating and optimizing the 3D walkthrough. The main ones included:

- Hardware capacity: The hardware's performance impacted rendering speed and the smoothness of the walkthrough, especially when handling models with high polygon counts and detailed textures.
- Development time: Modeling, texturing, and programming took longer than expected, limiting the ability to add more details or advanced animations.
- Technical knowledge: The learning curve for OpenGL and shader management was challenging, especially when implementing coordinated lighting and animations.
- Format compatibility: In some cases, when exporting models from Blender to OpenGL, issues arose with scaling, orientation, or texture loading that required manual adjustments.
- Animation optimization: The integration of animations within the 3D environment was limited by code complexity and the need to maintain smooth performance.

Project milestones

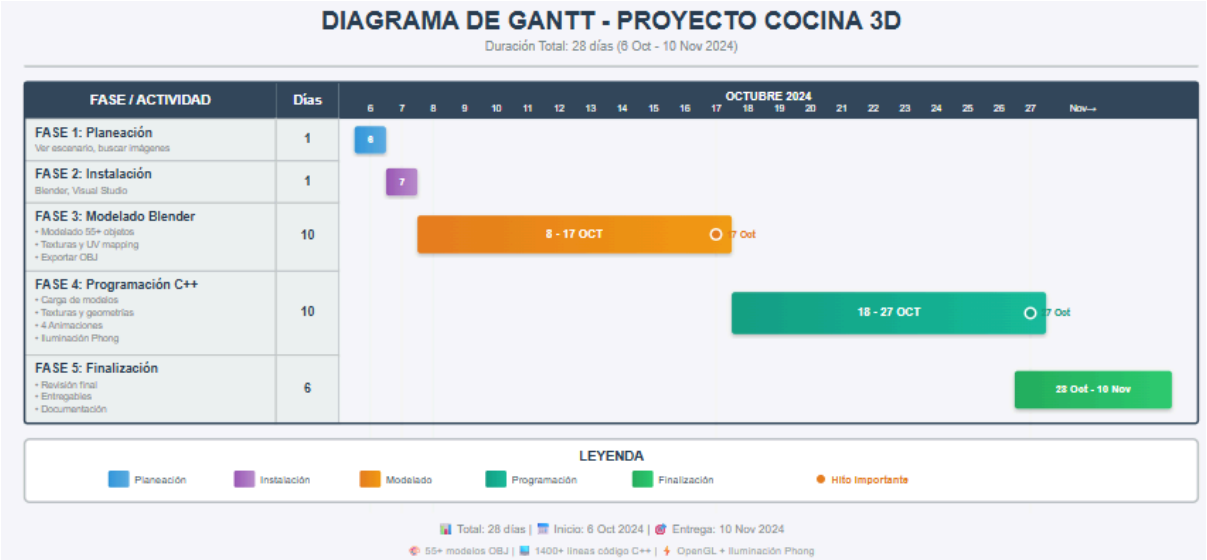
- Stage selection and visual references
- Installation and configuration of work tools
- 3D modeling of objects and spaces in Blender
- Exportation and integration of models in OpenGL
- Implementation of lighting, camera, and animations
- Project review and optimization
- Technical documentation and bilingual user manual completed
- Delivery of the executable project and publication on GitHub

Project schedule

Phase	Activities	Duration	Deadline
Phase 1: Project Planning	● See the scenario it would be. ● Find images of the scenario.	1 día	6 de Octubre.
Phase 2: Installation of Blender and Visual Studio.	● Application installation. ● Initial configurations of Blender and Visual Studio.	1 día	7 Octubre.
Phase 3: Modeling the objects in Blender.	● Modeling the objects to be used in Blender. ● Choosing and placing textures. ● Exporting Blender models in OBJ format.	10 días.	17 Octubre.
Phase 4: Creating the code in Visual Studio.	● Essential code for loading models. ● Creating models using coordinates. ● Loading textures. ● Creating animations. ● Adding lighting.	10 días.	27 de Octubre.
Phase 5: Completion.	● Final review. ● Preparation of deliverables. ● Final documentation.	6 días.	10 de Noviembre

Software flow diagram

Gantt chart



Project quote

The quote I provided was for a small, non-large-scale project team:
Staff costs:

Rol	Días	Tarifa/Día	Subtotal
Director de Proyecto	20	$2,800	$56,000
Analista	3	$1,800	$5,400
Modelador 3D	10	$2,200	$22,000
Artista de Texturas	8	$1,900	$15,200
Programador Senior C++	12	$3,000	$36,000
Programador de Gráficos	8	$2,800	$22,400
Desarrollador de Animaciones	6	$2,400	$14,400
Ingeniero de Integración	5	$2,100	$10,500
QA / Tester	6	$1,400	$8,400
Documentador Técnico	4	$1,600	$6,400
TOTAL PERSONAL			**$196,700 MXN**

Software and licenses

Concepto	Costo
Licencia Blender (Open Source)	$0
Visual Studio Community (Gratis)	$0
Licencias OpenGL/GLEW/GLFW (Open Source)	$0
Software de gestión de proyectos (Trello/Notion)	$800
Herramientas de edición de imágenes (GIMP/Photopea)	$1,200
SUBTOTAL SOFTWARE	**$2,000 MXN**

Total quote summary:

Categoría	Monto
👥 Personal	$196,700
💿 Software y Licencias	$2,000
SUBTOTAL	**$198,700 MXN**
🔧 Contingencia (15%)	$29,805
SUBTOTAL + CONTINGENCIA	**$228,505 MXN**
💰 Utilidad/Ganancia (25%)	$57,126
TOTAL DEL PROYECTO	**$285,631 MXN**

Introduction

To create our tour in OpenGL 3, it was necessary to use different applications, one of them was Blender, through this software I was able to model the different models and scenarios that we needed, in my case I chose to model a fictional Naruto scenario, the Ichiraku restaurant, this cartoon is very special to me because when I suffered from depression I used to watch Naruto and it made me happy every day, so I decided to make the tour about this topic.

To complete the project, we learned to program in OpenGL using various libraries such as glew, glf, soil, and others. These libraries helped us program our route more easily. We also used different shaders, which helped us with lighting and the look of the program. Finally, it's worth mentioning that everything we learned in the lab was extremely helpful in completing this project. Texture application, library use, different types of lighting, and animation were all very helpful in completing this project.

Below are the reference images we'll use for this project:

Facade:

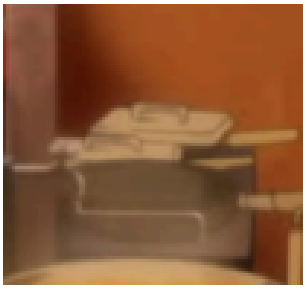
Room 1:

Room 2:

Room 1ObjectsTheory:

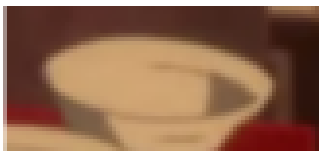
1)Pot with spoon

2) Stove cabinet



3) Seats

4) Bowls



5)Kitchen hood

Room 2 Objects Theory:

1) Empty shelf

2)Boxes

3) Refrigerator

4) Flour sack

5) Jars

State of the Art of Virtual Tours

Currently, virtual tours have become a widely used tool for interactively visualizing digital spaces. Thanks to advances in 3D modeling, graphics engines, and virtual and augmented reality technologies, these tours allow users to explore simulated environments with a high level of detail. Today, they are used in sectors such as real estate, museums, education, architecture, video games, and tourism, with the goal of offering immersive experiences without the need to be physically present. Current trends include the use of animations, dynamic lighting, realistic textures, integration with VR devices, and narrative experiences that enhance user interaction. In this context, our project is based on these modern practices, seeking to replicate a functional, visually appealing, and interactive virtual tour, in accordance with contemporary standards in digital design and space simulation.

Theoretical Framework

Blender.

Blender is an open-source 3D modeling, animation, and rendering software that allows you to create everything from simple objects to complex scenes with lighting, materials, textures, and motion. This tool is widely used in the film, video game, and animation industries for its ability to generate realistic environments and control every detail of the modeling.

In creating our project, Blender was essential for designing and building the models and sets. Thanks to its sculpting, texturing, and rendering tools, we were able to create the elements of the course, such as the fictional setting of the Ichiraku restaurant, inspired by the Naruto series.

OpenGL.

OpenGL (Open Graphics Library) is a cross-platform graphics API that allows 2D and 3D rendering using accelerated hardware. It is one of the most widely used libraries for creating interactive graphic environments, simulations, and video games.

The knowledge acquired in the lab was essential for integrating all these elements: applying textures, configuring lighting and materials, and basic model animation. All of this made it possible to bring the 3D environment to life and achieve a more immersive visual experience.

Methodology

1. Project planning

In this first phase, the theme and general concept of the tour were defined. It was decided to recreate the Ichiraku restaurant from the Naruto series, a symbolic space within the anime universe that holds personal emotional value.

During this stage, reference images of the set were collected, the main elements were analyzed, and the visual style that would be maintained throughout the project was established.

2. Modeling in Blender

In this phase, the 3D modeling of the objects and settings was carried out using Blender.

The different components of the environment were created: walls, tables, lamps, signs, and other decorative elements. Textures and materials were applied to achieve a more realistic appearance, paying attention to details such as scale, proportion, and colors to maintain visual consistency with the Ichiraku restaurant's setting.

Finally, the models were exported in .obj format so they could be later integrated into OpenGL.

3. OpenGL Programming

Once the modeling was complete, the technical part of the project began. We programmed the route within the 3D environment and the seven objects we were asked to create in OpenGL, along with the requested animations.

Libraries such as GLEW, GLFW, and SOIL were used to manage textures, windows, and rendering.

Shaders were also implemented to control the lighting and reflect the different materials in the environment. During this stage, parameters such as light intensity, color, and position were adjusted to achieve a suitable visual environment. The necessary animations were also applied.

4. Application of textures and lighting

The textures designed in Blender were applied to the models using OpenGL, verifying their correct projection and scale.

Subsequently, the light sources (directional, ambient, and specular) were configured to simulate the restaurant's lighting. This was essential to improve the visual quality of the tour and give the scene depth.

5. Testing and optimization

During this phase, functional tests were conducted on the 3D walkthrough.

Camera settings, model positions, lighting effects, and overall program performance were adjusted. Additionally, texture loading errors were fixed and the code was optimized to ensure smooth execution.

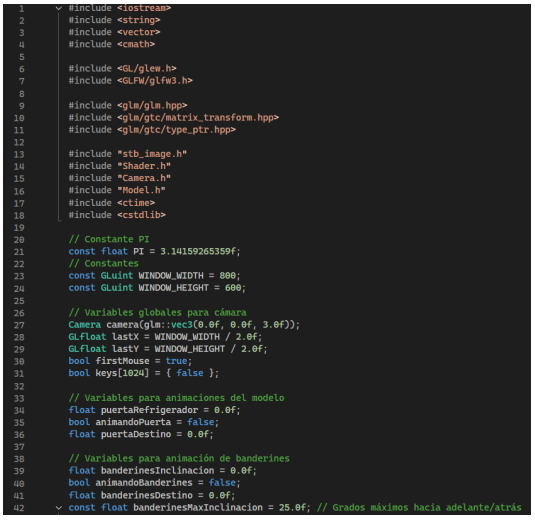
6. Integration and final presentation

Finally, all the elements were integrated: models, lighting, animations, and camera tracking.

The result was an interactive 3D environment that allows you to explore Ichiraku restaurant in real time, combining the art of 3D modeling with OpenGL programming.

CODE

Import all the necessary libraries (OpenGL, GLFW, GLM for 3D math, stb_image for textures). Define constants such as the 800x600 window size and PI. Declare global variables: the initial camera, mouse control, an array to detect keystrokes, and variables to animate the refrigerator door and flags (angles, destinations, states).



```
#include <iostream>
#include <string>
#include <vector>
#include <cmath>

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

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

#include "stb_image.h"
#include "Shader.h"
#include "Camera.h"
#include "Model.h"
#include <ctime>
#include <cstdlib>

// Constante PI
const float PI = 3.14159265359f;
// Constantes
const GLuint WINDOW_WIDTH = 800;
const GLuint WINDOW_HEIGHT = 600;

// Variables globales para cámara
Camera camera(glm::vec3(0.0f, 0.0f, 3.0f));
GLfloat lastX = WINDOW_WIDTH / 2.0f;
GLfloat lastY = WINDOW_HEIGHT / 2.0f;
bool firstMouse = true;
bool keys[1024] = { false };

// Variables para animaciones del modelo
float puertaRefrigerador = 0.0f;
bool animandoPuerta = false;
float puertaDestino = 0.0f;

// Variables para animación de banderines
float banderinesInclinacion = 0.0f;
bool animandoBanderines = false;
float banderinesDestino = 0.0f;
const float banderinesMaxInclinacion = 25.0f; // Grados máximos hacia adelante/atrás
```

Configure the 4-light system (3 lamps + Moon) with their positions in the 3D world and on/off states. Define time variables (deltaTime) for smooth animations. Create the tap water system: a Drop structure with position and velocity, a vector of up to 50 drops, the tap position, and a timer to generate drops every 0.02 seconds. Define the fireworks system: a Particle structure with position, velocity, color, lifetime, and size, a vector of up to 1000 particles, and a timer for explosions every 1.5 seconds.

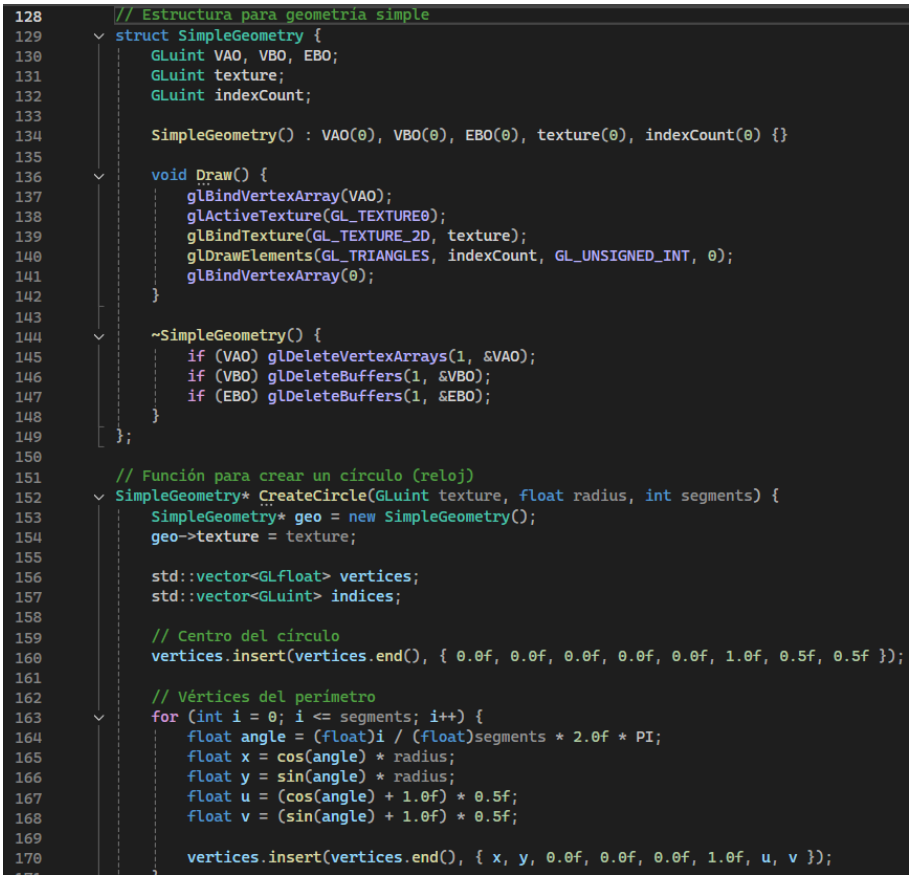
```
    // Sistema de iluminación
    bool lightStates[4] = { false, false, false, true }; // Estados de las 3 lámparas + Luna

    // Posiciones de las luces puntuales
    glm::vec3 pointLightPositions[] = {
        glm::vec3(0.0f, 3.0f, -2.0f),    // LamparasCuarto2
        glm::vec3(0.0f, 3.0f, 2.0f),     // LamparasCuarto1
        glm::vec3(-2.9f, 2.0f, 5.0f),     // LinternaLocal
        glm::vec3(4.0f, 8.0f, 5.0f)      // Luna
    };

    // Deltatime
    GLfloat deltaTime = 0.0f;
    GLfloat lastFrame = 0.0f;
    GLfloat currentFrame = 0.0f;
    // Variables para animación de agua del grifo (AGREGAR)
    bool aguaCayendo = false;
    struct Gota {
        glm::vec3 posicion;
        float velocidad;
        bool activa;
    };
    std::vector<Gota> gotas;
    const int MAX_GOTAS = 50;
    glm::vec3 posicionGrifo = glm::vec3(1.97f, 1.3f, 1.77f); // Ajusta según tu fregadero
    float tiempoUltimaGota = 0.0f;
    const float intervaloGotas = 0.02f;
    // Variables para fuegos artificiales
    bool fuegosActivos = false;
    struct Particula {
        glm::vec3 posicion;
        glm::vec3 velocidad;
        glm::vec3 color;
        float vida;
        float vidaInicial;
        float tamano;
        bool activa;
    };
    std::vector<Particula> particulas;
    const int MAX_PARTICULAS = 1000;
    float tiempoUltimoFuego = 0.0f;
    const float intervaloFuegos = 1.5f;
```

It declares the main functions: KeyCallback (detects keys), MouseCallback (captures mouse to rotate camera), ProcessMovement (moves camera and updates animations every frame), and LoadTexture (loads JPG/PNG images as OpenGL textures). The LoadTexture function takes an image path and converts it to an OpenGL texture. It first generates a unique texture ID, uses stbi_load to read the file, and gets the width, height, and color channels. If the image loaded correctly, it determines the format (RGB if it has 3 channels, RGBA if it has 4, RED if it has 1). It then binds the texture, uploads it to the GPU with glTexImage2D, generates mipmaps for different distances, and sets up looping (WRAP) and filtering (MIN/MAG_FILTER) parameters to make it look smooth. It prints success or error messages, frees the temporary memory, and returns the texture ID.

```
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode);
void MouseCallback(GLFWwindow* window, double xPos, double yPos);
void ProcessMovement();

GLuint LoadTexture(const char* path)
{
    GLuint textureID = 0;
    glGenTextures(1, &textureID);

    int width = 0, height = 0, nrChannels = 0;
    stbi_set_flip_vertically_on_load(true);
    unsigned char* data = stbi_load(path, &width, &height, &nrChannels, 0);

    if (data)
    {
        GLenum format = GL_RGB;
        if (nrChannels == 4)
            format = GL_RGBA;
        else if (nrChannels == 1)
            format = GL_RED;

        glBindTexture(GL_TEXTURE_2D, textureID);
        glTexImage2D(GL_TEXTURE_2D, 0, format, width, height, 0, format, GL_UNSIGNED_BYTE, data);
        glGenerateMipmap(GL_TEXTURE_2D);

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

        std::cout << "Textura cargada: " << path << std::endl;
    }
    else
    {
        std::cout << "Error al cargar: " << path << std::endl;
    }

    stbi_image_free(data);
    glBindTexture(GL_TEXTURE_2D, 0);

    return textureID;
}
// Estructura para geometría simple
```

Next, it defines the SimpleGeometry structure, which stores basic geometry created manually: it has a VAO (vertex array), a VBO (data buffer), an EBO (index buffer), the assigned texture, and the number of indices. It includes a Draw() method that activates the VAO, binds the texture to slot 0, draws triangles with glDrawElements, and a destructor that clears the GPU memory upon completion.

From this point, all the necessary structures are created for the elements requested in the lab. In my case, I made a clock, a table, a knife, a cheese, a chopping board, a box, and a tablecloth.



```
// Estructura para geometría simple
struct SimpleGeometry {
    GLuint VAO, VBO, EBO;
    GLuint texture;
    GLuint indexCount;

    SimpleGeometry() : VAO(0), VBO(0), EBO(0), texture(0), indexCount(0) {}

    void Draw() {
        glBindVertexArray(VAO);
        glActiveTexture(GL_TEXTURE0);
        glBindTexture(GL_TEXTURE_2D, texture);
        glDrawElements(GL_TRIANGLES, indexCount, GL_UNSIGNED_INT, 0);
        glBindVertexArray(0);
    }

    ~SimpleGeometry() {
        if (VAO) glDeleteVertexArrays(1, &VAO);
        if (VBO) glDeleteBuffers(1, &VBO);
        if (EBO) glDeleteBuffers(1, &EBO);
    }
};

// Función para crear un círculo (reloj)
SimpleGeometry* CreateCircle(GLuint texture, float radius, int segments) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    std::vector<GLfloat> vertices;
    std::vector<GLuint> indices;

    // Centro del círculo
    vertices.insert(vertices.end(), { 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.5f, 0.5f });

    // Vértices del perímetro
    for (int i = 0; i <= segments; i++) {
        float angle = (float)i / (float)segments * 2.0f * PI;
        float x = cos(angle) * radius;
        float y = sin(angle) * radius;
        float u = (cos(angle) + 1.0f) * 0.5f;
        float v = (sin(angle) + 1.0f) * 0.5f;

        vertices.insert(vertices.end(), { x, y, 0.0f, 0.0f, 0.0f, 1.0f, u, v });
```

```
    }

    // Índices
    for (int i = 1; i <= segments; i++) {
        indices.push_back(0);
        indices.push_back(i);
        indices.push_back(i + 1);
    }

    geo->indexCount = (GLuint)indices.size();

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);

    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    // Posición
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
    // Normal
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    // TexCoords
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

    glBindVertexArray(0);

    return geo;
}

// Función para crear una manecilla del reloj
SimpleGeometry* CreateClockHand(GLuint texture, float width, float length) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;
```

```
    std::vector<GLfloat> vertices = {
        // Posición              Normal             TexCoords
        -width / 2, 0.0f, 0.01f,   0.0f, 0.0f, 1.0f,   0.0f, 0.0f,
         width / 2, 0.0f, 0.01f,   0.0f, 0.0f, 1.0f,   1.0f, 0.0f,
         width / 2, length, 0.01f, 0.0f, 0.0f, 1.0f,   1.0f, 1.0f,
        -width / 2, length, 0.01f, 0.0f, 0.0f, 1.0f,   0.0f, 1.0f
    };

    std::vector<GLuint> indices = { 0, 1, 2, 2, 3, 0 };
    geo->indexCount = 6;

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);

    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

    glBindVertexArray(0);

    return geo;
}

// Función para crear un cubo
SimpleGeometry* CreateCube(GLuint texture) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    std::vector<GLfloat> vertices = {
        // Cara frontal (z+)
```

```cpp
// Función para crear un cubo
SimpleGeometry* CreateCube(GLuint texture) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    std::vector<GLfloat> vertices = {
        // Cara frontal (z+)
        -0.5f, -0.5f,  0.5f,  0.0f, 0.0f, 1.0f,  0.0f, 0.0f,
         0.5f, -0.5f,  0.5f,  0.0f, 0.0f, 1.0f,  1.0f, 0.0f,
         0.5f,  0.5f,  0.5f,  0.0f, 0.0f, 1.0f,  1.0f, 1.0f,
        -0.5f,  0.5f,  0.5f,  0.0f, 0.0f, 1.0f,  0.0f, 1.0f,

        // Cara trasera (z-)
        -0.5f, -0.5f, -0.5f,  0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
        -0.5f,  0.5f, -0.5f,  0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
         0.5f,  0.5f, -0.5f,  0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
         0.5f, -0.5f, -0.5f,  0.0f, 0.0f, -1.0f, 0.0f, 0.0f,

         // Cara superior (y+)
        -0.5f,  0.5f, -0.5f,  0.0f, 1.0f, 0.0f,  0.0f, 1.0f,
        -0.5f,  0.5f,  0.5f,  0.0f, 1.0f, 0.0f,  0.0f, 0.0f,
         0.5f,  0.5f,  0.5f,  0.0f, 1.0f, 0.0f,  1.0f, 0.0f,
         0.5f,  0.5f, -0.5f,  0.0f, 1.0f, 0.0f,  1.0f, 1.0f,

         // Cara inferior (y-)
        -0.5f, -0.5f, -0.5f,  0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
         0.5f, -0.5f, -0.5f,  0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
         0.5f, -0.5f,  0.5f,  0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
        -0.5f, -0.5f,  0.5f,  0.0f, -1.0f, 0.0f, 0.0f, 0.0f,

         // Cara derecha (x+)
         0.5f, -0.5f, -0.5f,  1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
         0.5f,  0.5f, -0.5f,  1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
         0.5f,  0.5f,  0.5f,  1.0f, 0.0f, 0.0f,  0.0f, 1.0f,
         0.5f, -0.5f,  0.5f,  1.0f, 0.0f, 0.0f,  0.0f, 0.0f,

         // Cara izquierda (x-)
        -0.5f, -0.5f, -0.5f,  -1.0f, 0.0f, 0.0f, 0.0f, 0.0f
        -0.5f, -0.5f,  0.5f,  -1.0f, 0.0f, 0.0f, 1.0f, 0.0f
        -0.5f,  0.5f,  0.5f,  -1.0f, 0.0f, 0.0f, 1.0f, 1.0f
        -0.5f,  0.5f, -0.5f,  -1.0f, 0.0f, 0.0f, 0.0f, 1.0f
    };

```

```
    std::vector<GLuint> indices = {
        0, 1, 2,  2, 3, 0,    // Frontal
        4, 5, 6,  6, 7, 4,    // Trasera
        8, 9, 10, 10, 11, 8,  // Superior
        12, 13, 14, 14, 15, 12, // Inferior
        16, 17, 18, 18, 19, 16, // Derecha
        20, 21, 22, 22, 23, 20  // Izquierda
    };

    geo->indexCount = 36;

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);

    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

    glBindVertexArray(0);

    return geo;
}

// Función para crear superficie de mesa
SimpleGeometry* CreateMesaTop(GLuint texture) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    std::vector<GLfloat> vertices = {
        // Superficie superior
        -0.5f, 0.05f, -0.5f,  0.0f, 1.0f, 0.0f,  0.0f, 0.0f,
         0.5f, 0.05f, -0.5f,  0.0f, 1.0f, 0.0f,  1.0f, 0.0f,
         0.5f, 0.05f,  0.5f,  0.0f, 1.0f, 0.0f,  1.0f, 1.0f,
        -0.5f, 0.05f,  0.5f,  0.0f, 1.0f, 0.0f,  0.0f, 1.0f,

        // Superficie inferior
        -0.5f, 0.0f, -0.5f,  0.0f, -1.0f, 0.0f,  0.0f, 0.0f,
         0.5f, 0.0f, -0.5f,  0.0f, -1.0f, 0.0f,  1.0f, 0.0f,
         0.5f, 0.0f,  0.5f,  0.0f, -1.0f, 0.0f,  1.0f, 1.0f,
        -0.5f, 0.0f,  0.5f,  0.0f, -1.0f, 0.0f,  0.0f, 1.0f,

        // Borde frontal (Z-)
        -0.5f, 0.0f, -0.5f,  0.0f, 0.0f, -1.0f,  0.0f, 0.0f,
         0.5f, 0.0f, -0.5f,  0.0f, 0.0f, -1.0f,  1.0f, 0.0f,
         0.5f, 0.05f, -0.5f,  0.0f, 0.0f, -1.0f,  1.0f, 1.0f,
        -0.5f, 0.05f, -0.5f,  0.0f, 0.0f, -1.0f,  0.0f, 1.0f,

        // Borde trasero (Z+)
        -0.5f, 0.0f,  0.5f,  0.0f, 0.0f, 1.0f,  0.0f, 0.0f,
         0.5f, 0.0f,  0.5f,  0.0f, 0.0f, 1.0f,  1.0f, 0.0f,
         0.5f, 0.05f,  0.5f,  0.0f, 0.0f, 1.0f,  1.0f, 1.0f,
        -0.5f, 0.05f,  0.5f,  0.0f, 0.0f, 1.0f,  0.0f, 1.0f,

        // Borde derecho (X+)
         0.5f, 0.0f, -0.5f,  1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
         0.5f, 0.0f,  0.5f,  1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
         0.5f, 0.05f,  0.5f,  1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
         0.5f, 0.05f, -0.5f,  1.0f, 0.0f, 0.0f,  0.0f, 1.0f,

         // Borde izquierdo (X-)
        -0.5f, 0.0f, -0.5f,  -1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
        -0.5f, 0.0f,  0.5f,  -1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
        -0.5f, 0.05f,  0.5f,  -1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
        -0.5f, 0.05f, -0.5f,  -1.0f, 0.0f, 0.0f,  0.0f, 1.0f
    };

    std::vector<GLuint> indices = {
        0, 1, 2,  2, 3, 0,      // Superior
        4, 6, 5,  4, 7, 6,      // Inferior
        8, 9, 10, 10, 11, 8,    // Borde frontal
        12, 14, 13, 12, 15, 14, // Borde trasero
        16, 17, 18, 18, 19, 16, // Borde derecho
        20, 22, 21, 20, 23, 22  // Borde izquierdo
    };
```

```
    geo->indexCount = 36;

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);

    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

    glBindVertexArray(0);

    return geo;
}
// Función para crear pata de mesa
SimpleGeometry* CreatePataMesa(GLuint texture) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    std::vector<GLfloat> vertices = {
        // Cilindro simplificado como cubo delgado
        -0.05f, -0.7f, -0.05f,  0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
         0.05f, -0.7f, -0.05f,  0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
         0.05f, -0.7f,  0.05f,  0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
        -0.05f, -0.7f,  0.05f,  0.0f, -1.0f, 0.0f, 0.0f, 1.0f,

        -0.05f, 0.0f, -0.05f,  0.0f, 1.0f, 0.0f,  0.0f, 0.0f,
         0.05f, 0.0f, -0.05f,  0.0f, 1.0f, 0.0f,  1.0f, 0.0f,
         0.05f, 0.0f,  0.05f,  0.0f, 1.0f, 0.0f,  1.0f, 1.0f,
        -0.05f, 0.0f,  0.05f,  0.0f, 1.0f, 0.0f,  0.0f, 1.0f
    };

    std::vector<GLuint> indices = {
        0, 1, 2,  2, 3, 0,  // Abajo
        4, 6, 5,  4, 7, 6,  // Arriba
        0, 1, 5,  5, 4, 0,  // Laterales
        1, 2, 6,  6, 5, 1,
        2, 3, 7,  7, 6, 2,
        3, 0, 4,  4, 7, 3
    };

    geo->indexCount = 36;

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);

    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

    glBindVertexArray(0);

    return geo;
}
// Función para crear Narutomaki (cilindro pequeño)
SimpleGeometry* CreateNarutomaki(GLuint texture) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    std::vector<GLfloat> vertices = {
        // Base inferior (y = 0.0f)
        -0.05f, 0.0f, -0.05f,  0.0f, -1.0f, 0.0f,  0.0f, 0.0f,
         0.05f, 0.0f, -0.05f,  0.0f, -1.0f, 0.0f,  1.0f, 0.0f,
```

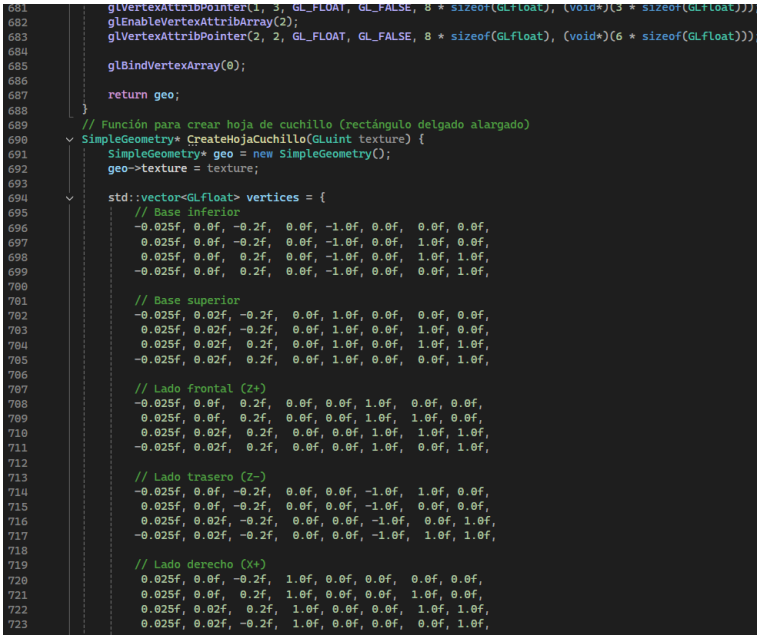
```
         0.05f, 0.0f,  0.05f,  0.0f, -1.0f, 0.0f,  1.0f, 1.0f,
        -0.05f, 0.0f,  0.05f,  0.0f, -1.0f, 0.0f,  0.0f, 1.0f,

        // Base superior (y = 0.05f)
        -0.05f, 0.05f, -0.05f,  0.0f, 1.0f, 0.0f,  0.0f, 0.0f,
         0.05f, 0.05f, -0.05f,  0.0f, 1.0f, 0.0f,  1.0f, 0.0f,
         0.05f, 0.05f,  0.05f,  0.0f, 1.0f, 0.0f,  1.0f, 1.0f,
        -0.05f, 0.05f,  0.05f,  0.0f, 1.0f, 0.0f,  0.0f, 1.0f,

        // Lado frontal (Z+)
        -0.05f, 0.0f,  0.05f,  0.0f, 0.0f, 1.0f,  0.0f, 0.0f,
         0.05f, 0.0f,  0.05f,  0.0f, 0.0f, 1.0f,  1.0f, 0.0f,
         0.05f, 0.05f,  0.05f,  0.0f, 0.0f, 1.0f,  1.0f, 1.0f,
        -0.05f, 0.05f,  0.05f,  0.0f, 0.0f, 1.0f,  0.0f, 1.0f,

        // Lado trasero (Z-)
        -0.05f, 0.0f, -0.05f,  0.0f, 0.0f, -1.0f,  1.0f, 0.0f,
         0.05f, 0.0f, -0.05f,  0.0f, 0.0f, -1.0f,  0.0f, 0.0f,
         0.05f, 0.05f, -0.05f,  0.0f, 0.0f, -1.0f,  0.0f, 1.0f,
        -0.05f, 0.05f, -0.05f,  0.0f, 0.0f, -1.0f,  1.0f, 1.0f,

        // Lado derecho (X+)
         0.05f, 0.0f, -0.05f,  1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
         0.05f, 0.0f,  0.05f,  1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
         0.05f, 0.05f,  0.05f,  1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
         0.05f, 0.05f, -0.05f,  1.0f, 0.0f, 0.0f,  0.0f, 1.0f,

         // Lado izquierdo (X-)
        -0.05f, 0.0f, -0.05f,  -1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
        -0.05f, 0.0f,  0.05f,  -1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
        -0.05f, 0.05f,  0.05f,  -1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
        -0.05f, 0.05f, -0.05f,  -1.0f, 0.0f, 0.0f,  0.0f, 1.0f
    };

    std::vector<GLuint> indices = {
        0, 1, 2,  2, 3, 0,      // Base inferior
        4, 6, 5,  4, 7, 6,      // Base superior
        8, 9, 10, 10, 11, 8,    // Frontal
        12, 14, 13, 12, 15, 14, // Trasero
        16, 17, 18, 18, 19, 16, // Derecho
        20, 22, 21, 20, 23, 22  // Izquierdo
    };
```

```
    geo->indexCount = 36;

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);

    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

    glBindVertexArray(0);

    return geo;
}

// Función para crear Sartén
SimpleGeometry* CreateSarten(GLuint texture) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    std::vector<GLfloat> vertices = {
        // Base inferior
        -0.2f, 0.0f, -0.2f,  0.0f, -1.0f, 0.0f,  0.0f, 0.0f,
         0.2f, 0.0f, -0.2f,  0.0f, -1.0f, 0.0f,  1.0f, 0.0f,
         0.2f, 0.0f,  0.2f,  0.0f, -1.0f, 0.0f,  1.0f, 1.0f,
        -0.2f, 0.0f,  0.2f,  0.0f, -1.0f, 0.0f,  0.0f, 1.0f,

        // Base superior
        -0.2f, 0.02f, -0.2f,  0.0f, 1.0f, 0.0f,  0.0f, 0.0f,
         0.2f, 0.02f, -0.2f,  0.0f, 1.0f, 0.0f,  1.0f, 0.0f,
         0.2f, 0.02f,  0.2f,  0.0f, 1.0f, 0.0f,  1.0f, 1.0f,
        -0.2f, 0.02f,  0.2f,  0.0f, 1.0f, 0.0f,  0.0f, 1.0f,
```

```
        // Lado frontal
        -0.2f, 0.0f,  0.2f,  0.0f, 0.0f, 1.0f,  0.0f, 0.0f,
         0.2f, 0.0f,  0.2f,  0.0f, 0.0f, 1.0f,  1.0f, 0.0f,
         0.2f, 0.02f,  0.2f,  0.0f, 0.0f, 1.0f,  1.0f, 1.0f,
        -0.2f, 0.02f,  0.2f,  0.0f, 0.0f, 1.0f,  0.0f, 1.0f,

        // Lado trasero
        -0.2f, 0.0f, -0.2f,  0.0f, 0.0f, -1.0f,  1.0f, 0.0f,
         0.2f, 0.0f, -0.2f,  0.0f, 0.0f, -1.0f,  0.0f, 0.0f,
         0.2f, 0.02f, -0.2f,  0.0f, 0.0f, -1.0f,  0.0f, 1.0f,
        -0.2f, 0.02f, -0.2f,  0.0f, 0.0f, -1.0f,  1.0f, 1.0f,

        // Lado derecho
         0.2f, 0.0f, -0.2f,  1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
         0.2f, 0.0f,  0.2f,  1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
         0.2f, 0.02f,  0.2f,  1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
         0.2f, 0.02f, -0.2f,  1.0f, 0.0f, 0.0f,  0.0f, 1.0f,

         // Lado izquierdo
        -0.2f, 0.0f, -0.2f,  -1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
        -0.2f, 0.0f,  0.2f,  -1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
        -0.2f, 0.02f,  0.2f,  -1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
        -0.2f, 0.02f, -0.2f,  -1.0f, 0.0f, 0.0f,  0.0f, 1.0f
    };

    std::vector<GLuint> indices = {
        0, 1, 2,  2, 3, 0,
        4, 6, 5,  4, 7, 6,
        8, 9, 10, 10, 11, 8,
        12, 14, 13, 12, 15, 14,
        16, 17, 18, 18, 19, 16,
        20, 22, 21, 20, 23, 22
    };

    geo->indexCount = 36;

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);

    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
```

```
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

    glBindVertexArray(0);

    return geo;
}

// Función para crear Tabla de Picar
SimpleGeometry* CreateTablaPicar(GLuint texture) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    std::vector<GLfloat> vertices = {
        // Base inferior
        -0.2f, 0.0f, -0.15f,  0.0f, -1.0f, 0.0f,  0.0f, 0.0f,
         0.2f, 0.0f, -0.15f,  0.0f, -1.0f, 0.0f,  1.0f, 0.0f,
         0.2f, 0.0f,  0.15f,  0.0f, -1.0f, 0.0f,  1.0f, 1.0f,
        -0.2f, 0.0f,  0.15f,  0.0f, -1.0f, 0.0f,  0.0f, 1.0f,

        // Base superior
        -0.2f, 0.02f, -0.15f,  0.0f, 1.0f, 0.0f,  0.0f, 0.0f,
         0.2f, 0.02f, -0.15f,  0.0f, 1.0f, 0.0f,  1.0f, 0.0f,
         0.2f, 0.02f,  0.15f,  0.0f, 1.0f, 0.0f,  1.0f, 1.0f,
        -0.2f, 0.02f,  0.15f,  0.0f, 1.0f, 0.0f,  0.0f, 1.0f,

        // Lado frontal
        -0.2f, 0.0f,  0.15f,  0.0f, 0.0f, 1.0f,  0.0f, 0.0f,
         0.2f, 0.0f,  0.15f,  0.0f, 0.0f, 1.0f,  1.0f, 0.0f,
         0.2f, 0.02f,  0.15f,  0.0f, 0.0f, 1.0f,  1.0f, 1.0f,
        -0.2f, 0.02f,  0.15f,  0.0f, 0.0f, 1.0f,  0.0f, 1.0f,

        // Lado trasero
        -0.2f, 0.0f, -0.15f,  0.0f, 0.0f, -1.0f,  1.0f, 0.0f,
```

```
         0.2f, 0.0f, -0.15f,  0.0f, 0.0f, -1.0f,  0.0f, 0.0f,
         0.2f, 0.02f, -0.15f,  0.0f, 0.0f, -1.0f,  0.0f, 1.0f,
        -0.2f, 0.02f, -0.15f,  0.0f, 0.0f, -1.0f,  1.0f, 1.0f,

        // Lado derecho
         0.2f, 0.0f, -0.15f,  1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
         0.2f, 0.0f,  0.15f,  1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
         0.2f, 0.02f,  0.15f,  1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
         0.2f, 0.02f, -0.15f,  1.0f, 0.0f, 0.0f,  0.0f, 1.0f,

         // Lado izquierdo
        -0.2f, 0.0f, -0.15f,  -1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
        -0.2f, 0.0f,  0.15f,  -1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
        -0.2f, 0.02f,  0.15f,  -1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
        -0.2f, 0.02f, -0.15f,  -1.0f, 0.0f, 0.0f,  0.0f, 1.0f
    };

    std::vector<GLuint> indices = {
        0, 1, 2,  2, 3, 0,
        4, 6, 5,  4, 7, 6,
        8, 9, 10, 10, 11, 8,
        12, 14, 13, 12, 15, 14,
        16, 17, 18, 18, 19, 16,
        20, 22, 21, 20, 23, 22
    };

    geo->indexCount = 36;

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);

    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
    glEnableVertexAttribArray(1);
```



```
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

    glBindVertexArray(0);

    return geo;
}
// Función para crear hoja de cuchillo (rectángulo delgado alargado)
SimpleGeometry* CreateHojaCuchillo(GLuint texture) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    std::vector<GLfloat> vertices = {
        // Base inferior
        -0.025f, 0.0f, -0.2f,  0.0f, -1.0f, 0.0f,  0.0f, 0.0f,
         0.025f, 0.0f, -0.2f,  0.0f, -1.0f, 0.0f,  1.0f, 0.0f,
         0.025f, 0.0f,  0.2f,  0.0f, -1.0f, 0.0f,  1.0f, 1.0f,
        -0.025f, 0.0f,  0.2f,  0.0f, -1.0f, 0.0f,  0.0f, 1.0f,

        // Base superior
        -0.025f, 0.02f, -0.2f,  0.0f, 1.0f, 0.0f,  0.0f, 0.0f,
         0.025f, 0.02f, -0.2f,  0.0f, 1.0f, 0.0f,  1.0f, 0.0f,
         0.025f, 0.02f,  0.2f,  0.0f, 1.0f, 0.0f,  1.0f, 1.0f,
        -0.025f, 0.02f,  0.2f,  0.0f, 1.0f, 0.0f,  0.0f, 1.0f,

        // Lado frontal (Z+)
        -0.025f, 0.0f,  0.2f,  0.0f, 0.0f, 1.0f,  0.0f, 0.0f,
         0.025f, 0.0f,  0.2f,  0.0f, 0.0f, 1.0f,  1.0f, 0.0f,
         0.025f, 0.02f,  0.2f,  0.0f, 0.0f, 1.0f,  1.0f, 1.0f,
        -0.025f, 0.02f,  0.2f,  0.0f, 0.0f, 1.0f,  0.0f, 1.0f,

        // Lado trasero (Z-)
        -0.025f, 0.0f, -0.2f,  0.0f, 0.0f, -1.0f,  1.0f, 0.0f,
         0.025f, 0.0f, -0.2f,  0.0f, 0.0f, -1.0f,  0.0f, 0.0f,
         0.025f, 0.02f, -0.2f,  0.0f, 0.0f, -1.0f,  0.0f, 1.0f,
        -0.025f, 0.02f, -0.2f,  0.0f, 0.0f, -1.0f,  1.0f, 1.0f,

        // Lado derecho (X+)
         0.025f, 0.0f, -0.2f,  1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
         0.025f, 0.0f,  0.2f,  1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
         0.025f, 0.02f,  0.2f,  1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
         0.025f, 0.02f, -0.2f,  1.0f, 0.0f, 0.0f,  0.0f, 1.0f,
```

```
        // Lado izquierdo (X-)
        -0.025f, 0.0f, -0.2f,  -1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
        -0.025f, 0.0f,  0.2f,  -1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
        -0.025f, 0.02f,  0.2f,  -1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
        -0.025f, 0.02f, -0.2f,  -1.0f, 0.0f, 0.0f,  0.0f, 1.0f
    };

    std::vector<GLuint> indices = {
        0, 1, 2,  2, 3, 0,      // Base inferior
        4, 6, 5,  4, 7, 6,      // Base superior
        8, 9, 10, 10, 11, 8,    // Frontal
        12, 14, 13, 12, 15, 14, // Trasero
        16, 17, 18, 18, 19, 16, // Derecho
        20, 22, 21, 20, 23, 22  // Izquierdo
    };

    geo->indexCount = 36;

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);

    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

    glBindVertexArray(0);

    return geo;
}

// Función para crear mango de cuchillo (cubo más pequeño)
```

```cpp
SimpleGeometry* CreateMangoCuchillo(GLuint texture) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    std::vector<GLfloat> vertices = {
        // Base inferior
        -0.025f, 0.0f, -0.05f,  0.0f, -1.0f, 0.0f,  0.0f, 0.0f,
         0.025f, 0.0f, -0.05f,  0.0f, -1.0f, 0.0f,  1.0f, 0.0f,
         0.025f, 0.0f,  0.05f,  0.0f, -1.0f, 0.0f,  1.0f, 1.0f,
        -0.025f, 0.0f,  0.05f,  0.0f, -1.0f, 0.0f,  0.0f, 1.0f,

        // Base superior
        -0.025f, 0.02f, -0.05f,  0.0f, 1.0f, 0.0f,  0.0f, 0.0f,
         0.025f, 0.02f, -0.05f,  0.0f, 1.0f, 0.0f,  1.0f, 0.0f,
         0.025f, 0.02f,  0.05f,  0.0f, 1.0f, 0.0f,  1.0f, 1.0f,
        -0.025f, 0.02f,  0.05f,  0.0f, 1.0f, 0.0f,  0.0f, 1.0f,

        // Lado frontal (Z+)
        -0.025f, 0.0f,  0.05f,  0.0f, 0.0f, 1.0f,  0.0f, 0.0f,
         0.025f, 0.0f,  0.05f,  0.0f, 0.0f, 1.0f,  1.0f, 0.0f,
         0.025f, 0.02f,  0.05f,  0.0f, 0.0f, 1.0f,  1.0f, 1.0f,
        -0.025f, 0.02f,  0.05f,  0.0f, 0.0f, 1.0f,  0.0f, 1.0f,

        // Lado trasero (Z-)
        -0.025f, 0.0f, -0.05f,  0.0f, 0.0f, -1.0f,  1.0f, 0.0f,
         0.025f, 0.0f, -0.05f,  0.0f, 0.0f, -1.0f,  0.0f, 0.0f,
         0.025f, 0.02f, -0.05f,  0.0f, 0.0f, -1.0f,  0.0f, 1.0f,
        -0.025f, 0.02f, -0.05f,  0.0f, 0.0f, -1.0f,  1.0f, 1.0f,

        // Lado derecho (X+)
         0.025f, 0.0f, -0.05f,  1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
         0.025f, 0.0f,  0.05f,  1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
         0.025f, 0.02f,  0.05f,  1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
         0.025f, 0.02f, -0.05f,  1.0f, 0.0f, 0.0f,  0.0f, 1.0f,

         // Lado izquierdo (X-)
        -0.025f, 0.0f, -0.05f,  -1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
        -0.025f, 0.0f,  0.05f,  -1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
        -0.025f, 0.02f,  0.05f,  -1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
        -0.025f, 0.02f, -0.05f,  -1.0f, 0.0f, 0.0f,  0.0f, 1.0f
    };

    std::vector<GLuint> indices = {
        0, 1, 2,  2, 3, 0,
        4, 6, 5,  4, 7, 6,
        8, 9, 10, 10, 11, 8,
        12, 14, 13, 12, 15, 14,
        16, 17, 18, 18, 19, 16,
        20, 22, 21, 20, 23, 22
    };

    geo->indexCount = 36;

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);

    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

    glBindVertexArray(0);

    return geo;
}
// Función para crear una gota de agua (cubo alargado con grosor medio)
SimpleGeometry* CreateGotaAgua(GLuint texture) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    // Gotas con grosor medio (0.008) y largas (0.04)
    std::vector<GLfloat> vertices = {
        -0.008f, -0.04f,  0.008f,  0.0f, 0.0f, 1.0f,  0.0f, 0.0f,
         0.008f, -0.04f,  0.008f,  0.0f, 0.0f, 1.0f,  1.0f, 0.0f,
         0.008f,  0.04f,  0.008f,  0.0f, 0.0f, 1.0f,  1.0f, 1.0f,
```

```
        -0.008f,  0.04f,  0.008f,  0.0f, 0.0f, 1.0f,  0.0f, 1.0f,

        -0.008f, -0.04f, -0.008f,  0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
        -0.008f,  0.04f, -0.008f,  0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
         0.008f,  0.04f, -0.008f,  0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
         0.008f, -0.04f, -0.008f,  0.0f, 0.0f, -1.0f, 0.0f, 0.0f,

        -0.008f,  0.04f, -0.008f,  0.0f, 1.0f, 0.0f,  0.0f, 1.0f,
        -0.008f,  0.04f,  0.008f,  0.0f, 1.0f, 0.0f,  0.0f, 0.0f,
         0.008f,  0.04f,  0.008f,  0.0f, 1.0f, 0.0f,  1.0f, 0.0f,
         0.008f,  0.04f, -0.008f,  0.0f, 1.0f, 0.0f,  1.0f, 1.0f,

        -0.008f, -0.04f, -0.008f,  0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
         0.008f, -0.04f, -0.008f,  0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
         0.008f, -0.04f,  0.008f,  0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
        -0.008f, -0.04f,  0.008f,  0.0f, -1.0f, 0.0f, 0.0f, 0.0f,

         0.008f, -0.04f, -0.008f,  1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
         0.008f,  0.04f, -0.008f,  1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
         0.008f,  0.04f,  0.008f,  1.0f, 0.0f, 0.0f,  0.0f, 1.0f,
         0.008f, -0.04f,  0.008f,  1.0f, 0.0f, 0.0f,  0.0f, 0.0f,

        -0.008f, -0.04f, -0.008f,  -1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
        -0.008f, -0.04f,  0.008f,  -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
        -0.008f,  0.04f,  0.008f,  -1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
        -0.008f,  0.04f, -0.008f,  -1.0f, 0.0f, 0.0f, 0.0f, 1.0f
    };

    std::vector<GLuint> indices = {
        0, 1, 2,  2, 3, 0,
        4, 5, 6,  6, 7, 4,
        8, 9, 10, 10, 11, 8,
        12, 13, 14, 14, 15, 12,
        16, 17, 18, 18, 19, 16,
        20, 21, 22, 22, 23, 20
    };

    geo->indexCount = 36;

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);
```

```
    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

    glBindVertexArray(0);

    return geo;
}
// Crear esfera pequeña para partículas de fuegos artificiales
SimpleGeometry* CreateParticulaFuego(GLuint texture) {
    SimpleGeometry* geo = new SimpleGeometry();
    geo->texture = texture;

    std::vector<GLfloat> vertices = {
        // Cara frontal
        -0.5f, -0.5f,  0.5f,  0.0f, 0.0f, 1.0f,  0.0f, 0.0f,
         0.5f, -0.5f,  0.5f,  0.0f, 0.0f, 1.0f,  1.0f, 0.0f,
         0.5f,  0.5f,  0.5f,  0.0f, 0.0f, 1.0f,  1.0f, 1.0f,
        -0.5f,  0.5f,  0.5f,  0.0f, 0.0f, 1.0f,  0.0f, 1.0f,

        // Cara trasera
        -0.5f, -0.5f, -0.5f,  0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
        -0.5f,  0.5f, -0.5f,  0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
         0.5f,  0.5f, -0.5f,  0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
         0.5f, -0.5f, -0.5f,  0.0f, 0.0f, -1.0f, 0.0f, 0.0f,

         // Cara superior
        -0.5f,  0.5f, -0.5f,  0.0f, 1.0f, 0.0f,  0.0f, 1.0f,
        -0.5f,  0.5f,  0.5f,  0.0f, 1.0f, 0.0f,  0.0f, 0.0f,
         0.5f,  0.5f,  0.5f,  0.0f, 1.0f, 0.0f,  1.0f, 0.0f,
         0.5f,  0.5f, -0.5f,  0.0f, 1.0f, 0.0f,  1.0f, 1.0f,
```

```
        // Cara inferior
        -0.5f, -0.5f, -0.5f,  0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
         0.5f, -0.5f, -0.5f,  0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
         0.5f, -0.5f,  0.5f,  0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
        -0.5f, -0.5f,  0.5f,  0.0f, -1.0f, 0.0f, 0.0f, 0.0f,

        // Cara derecha
         0.5f, -0.5f, -0.5f,  1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
         0.5f,  0.5f, -0.5f,  1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
         0.5f,  0.5f,  0.5f,  1.0f, 0.0f, 0.0f,  0.0f, 1.0f,
         0.5f, -0.5f,  0.5f,  1.0f, 0.0f, 0.0f,  0.0f, 0.0f,

         // Cara izquierda
        -0.5f, -0.5f, -0.5f,  -1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
        -0.5f, -0.5f,  0.5f,  -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
        -0.5f,  0.5f,  0.5f,  -1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
        -0.5f,  0.5f, -0.5f,  -1.0f, 0.0f, 0.0f, 0.0f, 1.0f
    };

    std::vector<GLuint> indices = {
        0, 1, 2,  2, 3, 0,
        4, 5, 6,  6, 7, 4,
        8, 9, 10, 10, 11, 8,
        12, 13, 14, 14, 15, 12,
        16, 17, 18, 18, 19, 16,
        20, 21, 22, 22, 23, 20
    };

    geo->indexCount = 36;

    glGenVertexArrays(1, &geo->VAO);
    glGenBuffers(1, &geo->VBO);
    glGenBuffers(1, &geo->EBO);

    glBindVertexArray(geo->VAO);

    glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
```

```
        glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
        glEnableVertexAttribArray(1);
        glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
        glEnableVertexAttribArray(2);
        glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

        glBindVertexArray(0);

        return geo;
    }
    void CrearExplosion(glm::vec3 posicion, glm::vec3 color) {
        int particulasPorExplosion = 250;  // MUCHAS MÁS PARTÍCULAS
        int creadas = 0;

        for (int i = 0; i < MAX_PARTICULAS && creadas < particulasPorExplosion; i++) {
            if (!particulas[i].activa) {
                particulas[i].activa = true;
                particulas[i].posicion = posicion;
                particulas[i].vida = 3.0f + (rand() % 150) / 50.0f;  // Viven MÁS tiempo
                particulas[i].vidaInicial = particulas[i].vida;
                particulas[i].color = color;
                particulas[i].tamano = 0.35f + (rand() % 150) / 500.0f;  // MUY GRANDES (0.35-0.65)

                // Velocidades MUY RÁPIDAS para explosiones grandes
                float theta = (rand() % 360) * PI / 180.0f;
                float phi = (rand() % 180) * PI / 180.0f;
                float velocidad = 5.0f + (rand() % 600) / 100.0f;  // 5.0 a 11.0 (muy rápido)

                particulas[i].velocidad = glm::vec3(
                    velocidad * sin(phi) * cos(theta),
                    velocidad * cos(phi),
                    velocidad * sin(phi) * sin(theta)
                );

                creadas++;
            }
        }
    }
    int main()
    {
        if (!glfwInit())
        {
            std::cout << "Error al inicializar GLFW" << std::endl;
```

This section of code creates the camera's perspective projection and then loads additional textures and initializes custom geometries using specialized functions. It defines textures for specific elements such as a clock, hands, cubes, kitchen table, Narutomaki, frying pan, chopping board, knife, and water. It then creates simple geometric objects using these textures: a circle for the clock face, hour and minute hands, a cube, a table top and legs, cheese, a tablecloth, a chopping board with a knife (separate blade and handle), water droplets, and fire particles for visual effects, all stored in SimpleGeometry structures to be later rendered in the 3D scene. In addition to loading the 3D models that we modeled in Blender.

```
        return EXIT_FAILURE;
    }

    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
    glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);

    GLFWwindow* window = glfwCreateWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Modelado Jerarquico - Cocina 3D", nullptr, nullptr);

    if (!window)
    {
        std::cout << "Error al crear ventana GLFW" << std::endl;
        glfwTerminate();
        return EXIT_FAILURE;
    }

    glfwMakeContextCurrent(window);
    glfwSetKeyCallback(window, KeyCallback);
    glfwSetCursorPosCallback(window, MouseCallback);
    glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);

    glewExperimental = GL_TRUE;
    if (glewInit() != GLEW_OK)
    {
        std::cout << "Error al inicializar GLEW" << std::endl;
        glfwTerminate();
        return EXIT_FAILURE;
    }

    int screenWidth = 0, screenHeight = 0;
    glfwGetFramebufferSize(window, &screenWidth, &screenHeight);
    glViewport(0, 0, screenWidth, screenHeight);
    glEnable(GL_DEPTH_TEST);
    glClearColor(0.05f, 0.05f, 0.15f, 1.0f);

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

    std::cout << "\n=== CONTROLES ===" << std::endl;
    std::cout << "W/A/S/D: Mover cámara" << std::endl;
    std::cout << "Mouse: Mirar alrededor" << std::endl;
    std::cout << "O/C: Abrir/Cerrar puerta del refrigerador" << std::endl;
```

```
    std::cout << "N: Mover banderines hacia atrás" << std::endl;
    std::cout << "M: Regresar banderines a posición normal" << std::endl;
    std::cout << "G: Abrir/Cerrar grifo (agua)" << std::endl;
    std::cout << "F: Activar/Desactivar fuegos artificiales" << std::endl;
    std::cout << "1/2/3: Encender/Apagar lámparas de cocina" << std::endl;
    std::cout << "4: Encender/Apagar luz de la Luna" << std::endl;
    std::cout << "ESC: Salir\n" << std::endl;
    // Carga de texturas
    GLuint textureArroz = LoadTexture("images/5138.jpg");
    GLuint textureBarandal = LoadTexture("images/fondo-abstracto-con-textura (1).jpg");
    GLuint textureBase = LoadTexture("images/O5X8JM0.jpg");
    GLuint textureBotes = LoadTexture("images/OE5TN00.jpg");
    GLuint textureBolsaHarinaAbierta1 = LoadTexture("images/Illustration03.jpg");
    GLuint textureBolsaHarinaAbierta = LoadTexture("images/Illustration03.jpg");
    GLuint textureBolsasHarina = LoadTexture("images/Illustration03.jpg");
    GLuint textureBoteBasura = LoadTexture("images/fondo-abstracto-con-textura.jpg");
    GLuint textureCadenaLinterna = LoadTexture("images/5178.jpg");
    GLuint textureCaja = LoadTexture("images/16.jpg");
    GLuint textureCajas2 = LoadTexture("images/16.jpg");
    GLuint textureCajasMadera = LoadTexture("images/370.jpg");
    GLuint textureCampana = LoadTexture("images/5178.jpg");
    GLuint textureCortina2 = LoadTexture("images/preview.jpg");
    GLuint textureCortina = LoadTexture("images/169180-OVUV5U-320.jpg");
    GLuint textureCortinas = LoadTexture("images/preview.jpg");
    GLuint textureCubeta = LoadTexture("images/OR6ILJ0.jpg");
    GLuint textureElectricidad = LoadTexture("images/5178.jpg");
    GLuint textureEscaleras = LoadTexture("images/3690.jpg");
    GLuint textureEstufa = LoadTexture("images/textura-de-placa-de-metal.jpg");
    GLuint textureFregadero = LoadTexture("images/fondo-abstracto-con-textura.jpg");
    GLuint textureGas = LoadTexture("images/OQV54F0.jpg");
    GLuint textureJalador = LoadTexture("images/OR6ILJ0.jpg");
    GLuint textureLamparasCocina = LoadTexture("images/fondo-abstracto-con-textura.jpg");
    GLuint textureLamparasCocina1 = LoadTexture("images/2259.jpg");
    GLuint textureLamparas1 = LoadTexture("images/2259.jpg");
    GLuint textureLamparas = LoadTexture("images/textura-de-placa-de-metal.jpg");
    GLuint textureLibrero = LoadTexture("images/3690.jpg");
    GLuint textureLuna = LoadTexture("images/2259.jpg");
    GLuint textureLinternaLocal = LoadTexture("images/2259.jpg");
    GLuint textureMesa = LoadTexture("images/OQV54F0.jpg");
    GLuint textureOlla = LoadTexture("images/5178.jpg");
    GLuint textureOlla1 = LoadTexture("images/5178.jpg");
    GLuint textureOlla2 = LoadTexture("images/acuarela-de-oro-liquido-espacio-de-copia.jpg");
    GLuint textureOlla3 = LoadTexture("images/acuarela-de-oro-liquido-espacio-de-copia.jpg");
```

```
    GLuint texturePasto = LoadTexture("images/preview1.jpg");
    GLuint texturePared3 = LoadTexture("images/3690.jpg");
    GLuint texturePared4 = LoadTexture("images/3690.jpg");
    GLuint texturePared = LoadTexture("images/18366.jpg");
    GLuint texturePlatos = LoadTexture("images/OIVY360.jpg");
    GLuint texturePiso2 = LoadTexture("images/8469.jpg");
    GLuint texturePrimerPiso = LoadTexture("images/liquid-marbling-paint-texture-background-fluid
    GLuint texturePuertas = LoadTexture("images/3690.jpg");
    GLuint textureRefrigerador = LoadTexture("images/OIVY360.jpg");
    GLuint textureRefrigerador1 = LoadTexture("images/OIVY360.jpg");
    GLuint textureSilla2 = LoadTexture("images/fondo-abstracto-con-textura.jpg");
    GLuint textureSilla = LoadTexture("images/textura-grunge-roja.jpg");
    GLuint textureTapa = LoadTexture("images/OR6ILJ0.jpg");
    GLuint textureTapa1 = LoadTexture("images/OR6ILJ0.jpg");
    GLuint textureTapa2 = LoadTexture("images/OR6ILJ0.jpg");
    GLuint textureTanqueAgua = LoadTexture("images/fondo-abstracto-con-textura.jpg");
    GLuint textureTapeteAlmacen = LoadTexture("images/47074.jpg");
    GLuint textureTapeteCocina = LoadTexture("images/abstract-blur-empty-green-gradient-studio-we
    GLuint textureTazon = LoadTexture("images/OIVY360.jpg");
    GLuint textureTazones2 = LoadTexture("images/OIVY360.jpg");
    GLuint textureTejado = LoadTexture("images/6593778.jpg");
    GLuint textureTejadoMini = LoadTexture("images/3690.jpg");
    GLuint textureTejadoMiniPiso2 = LoadTexture("images/3690.jpg");
    GLuint textureTuboFuera = LoadTexture("images/5178.jpg");
    GLuint textureTuboGas = LoadTexture("images/5178.jpg");
    GLuint textureVentanaSegundoPiso = LoadTexture("images/3690.jpg");
    GLuint textureBanderines = LoadTexture("images/preview.jpg");

    // Carga de modelos
    Model Arroz((char*)"Models/Arroz.obj");
    Model Barandal((char*)"Models/Barandal.obj");
    Model Base((char*)"Models/Base.obj");
    Model Botes((char*)"Models/Botes.obj");
    Model BolsaHarinaAbierta1((char*)"Models/BolsaHarinaAbierta1.obj");
    Model BolsaHarinaAbierta((char*)"Models/BolsaHarinaAbierta.obj");
    Model BolsasHarina((char*)"Models/BolsasHarina.obj");
    Model BoteBasura((char*)"Models/BoteBasura.obj");
    Model CadenaLinterna((char*)"Models/CadenaLinterna.obj");
    Model Caja((char*)"Models/Caja.obj");
    Model Cajas2((char*)"Models/Cajas2.obj");
    Model CajasMadera((char*)"Models/CajasMadera.obj");
    Model Campana((char*)"Models/Campana.obj");
    Model Cortina2((char*)"Models/Cortina2.obj");
```

```
        Model Cortina((char*)"Models/Cortina.obj");
        Model Cortinas((char*)"Models/Cortinas.obj");
        Model Cubeta((char*)"Models/Cubeta.obj");
        Model Electricidad((char*)"Models/Electricidad.obj");
        Model Escaleras((char*)"Models/Escaleras.obj");
        Model Estufa((char*)"Models/Estufa.obj");
        Model Fregadero((char*)"Models/Fregadero.obj");
        Model Gas((char*)"Models/Gas.obj");
        Model Jalador((char*)"Models/Jalador.obj");
        Model LamparasCocina((char*)"Models/LamparasCocina.obj");
        Model LamparasCocina1((char*)"Models/LamparasCocina1.obj");
        Model Librero((char*)"Models/Librero.obj");
        Model Luna((char*)"Models/Luna.obj");
        Model LinternaLocal((char*)"Models/LinternaLocal.obj");
        Model Mesa((char*)"Models/Mesa.obj");
        Model Olla((char*)"Models/Olla.obj");
        Model Olla1((char*)"Models/Olla1.obj");
        Model Olla2((char*)"Models/Olla2.obj");
        Model Olla3((char*)"Models/Olla3.obj");
        Model Pasto((char*)"Models/Pasto.obj");
        Model Pared3((char*)"Models/Pared3.obj");
        Model Pared4((char*)"Models/Pared4.obj");
        Model Pared((char*)"Models/Pared.obj");
        Model Platos((char*)"Models/Platos.obj");
        Model Piso2((char*)"Models/Piso2.obj");
        Model PrimerPiso((char*)"Models/PrimerPiso.obj");
        Model Puertas((char*)"Models/Puertas.obj");
        Model Refrigerador((char*)"Models/Refrigerador.obj");
        Model Refrigerador1((char*)"Models/Refrigerador1.obj");
        Model Silla2((char*)"Models/Silla2.obj");
        Model Silla((char*)"Models/Silla.obj");
        Model Tapa((char*)"Models/Tapa.obj");
        Model Tapa1((char*)"Models/Tapa1.obj");
        Model Tapa2((char*)"Models/Tapa2.obj");
        Model TanqueAgua((char*)"Models/TanqueAgua.obj");
        Model TapeteAlmacen((char*)"Models/TapeteAlmacen.obj");
        Model TapeteCocina((char*)"Models/TapeteCocina.obj");
        Model Tazon((char*)"Models/Tazon.obj");
        Model Tazones2((char*)"Models/Tazones2.obj");
        Model Tejado((char*)"Models/Tejado.obj");
        Model TejadoMini((char*)"Models/TejadoMini.obj");
        Model TejadoMiniPiso2((char*)"Models/TejadoMiniPiso2.obj");
        Model TuboFuera((char*)"Models/TuboFuera.obj");
```

```
    Model TuboGas((char*)"Models/TuboGas.obj");
    Model VentanaSegundoPiso((char*)"Models/VentanaSegundoPiso.obj");
    Model Banderines((char*)"Models/Banderines.obj");

    glm::mat4 projection = glm::perspective(glm::radians(45.0f), (GLfloat)screenWidth / (GLfloat)screenHeight, 0.1f, 100.0f);

    // Crear geometrías personalizadas
    GLuint textureReloj = LoadTexture("images/OIVV360.jpg");
    GLuint textureManecilla = LoadTexture("images/fondo-abstracto-con-textura.jpg");
    GLuint textureCubo = LoadTexture("images/16.jpg");
    GLuint textureMesaCocina = LoadTexture("images/3690.jpg");
    GLuint textureNarutomaki = LoadTexture("images/OIVV360.jpg");
    GLuint textureSarten = LoadTexture("images/fondo-abstracto-con-textura.jpg");
    GLuint textureTabla = LoadTexture("images/fondo-de-diseno-de-textura-de-madera-de-roble.jpg");
    GLuint textureCuchilloHoja = LoadTexture("images/fondo-abstracto-con-textura.jpg");
    GLuint textureCuchilloMango = LoadTexture("images/3690.jpg");
    GLuint textureAgua = LoadTexture("images/papel-pintado-azul-monocromatico-minimalista.jpg");
    GLuint textureParticula = LoadTexture("images/acuarela-de-oro-liquido-espacio-de-copia.jpg"); // Textura brillante

    SimpleGeometry* ParticulaFuego = CreateParticulaFuego(textureParticula);
    SimpleGeometry* RelojCirculo = CreateCircle(textureReloj, 0.5f, 60);
    SimpleGeometry* ManecillaHora = CreateClockHand(textureManecilla, 0.05f, 0.3f);
    SimpleGeometry* ManecillaMinuto = CreateClockHand(textureManecilla, 0.03f, 0.45f);
    SimpleGeometry* Cubo = CreateCube(textureCubo);
    SimpleGeometry* MesaTop = CreateMesaTop(textureMesaCocina);
    SimpleGeometry* Pata = CreatePataMesa(textureMesaCocina);
    SimpleGeometry* Narutomaki = CreateNarutomaki(textureNarutomaki);
    SimpleGeometry* Sarten = CreateSarten(textureSarten);
    SimpleGeometry* TablaPicar = CreateTablaPicar(textureTabla);
    SimpleGeometry* HojaCuchillo = CreateHojaCuchillo(textureCuchilloHoja);
    SimpleGeometry* MangoCuchillo = CreateMangoCuchillo(textureCuchilloMango);
    SimpleGeometry* GotaAgua = CreateGotaAgua(textureAgua);

    // Inicializar el vector de gotas
    gotas.resize(MAX_GOTAS);
    for (int i = 0; i < MAX_GOTAS; i++) {
        gotas[i].activa = false;
        gotas[i].velocidad = 2.0f;
    }
    // Inicializar partículas de fuegos artificiales
    particulas.resize(MAX_PARTICULAS);
    for (int i = 0; i < MAX_PARTICULAS; i++) {
        particulas[i].activa = false;
```

This code represents the main rendering loop that runs every frame. It calculates the delta time between frames, clears the screen, and sets up the lighting system (directional light, 4 point lights with on/off states, and a spot light). Then it renders all the objects in the scene in order: first, it draws the firework particles if they are active using blending for transparency, then it renders all the static 3D models of the kitchen (rice, furniture, appliances, decorations) applying their respective textures and brightness values, and finally it draws the animated custom geometries like the clock with rotating hands, a cube, a table with four legs, manually controllable pennants, a cutting board, a tablecloth, cheese, a knife with a blade and handle, and animated water drops falling from the tap when it is activated, all with their corresponding matrix transformations.

```
    }
    srand(time(NULL)); // Para números aleatorios
    glm::vec3 posReloj = glm::vec3(-2.0f, 2.0f, 0.15f);
    glm::vec3 posCaja = glm::vec3(1.50f, 4.80f, -0.30f);
    glm::vec3 posMesa = glm::vec3(0.80f, 5.0f, -2.40f);
    glm::vec3 posTabla = glm::vec3(-1.50f, 1.17f, 1.5f);
    glm::vec3 posSarten = glm::vec3(0.80f, 5.05f, -2.40f);

    while (!glfwWindowShouldClose(window))
    {
        currentFrame = (GLfloat)glfwGetTime();
        deltaTime = currentFrame - lastFrame;
        lastFrame = currentFrame;

        glfwPollEvents();
        ProcessMovement();

        glClearColor(0.05f, 0.05f, 0.15f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        lightingShader.Use();

        GLint viewPosLoc = glGetUniformLocation(lightingShader.Program, "viewPos");
        glm::vec3 cameraPos = camera.GetPosition();
        glUniform3f(viewPosLoc, cameraPos.x, cameraPos.y, cameraPos.z);

        glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.05f, 0.05f, 0.1f);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.1f, 0.1f, 0.15f);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.2f, 0.2f, 0.3f);

        for (int i = 0; i < 4; ++i) {
            std::string prefix = "pointLights[" + std::to_string(i) + "]";

            glm::vec3 ambient, diffuse, specular;
            float constant, linear, quadratic;

            if (i == 3) {
                ambient = lightStates[i] ? glm::vec3(0.2f, 0.2f, 0.3f) : glm::vec3(0.0f);
                diffuse = lightStates[i] ? glm::vec3(0.8f, 0.8f, 1.2f) : glm::vec3(0.0f);
                specular = lightStates[i] ? glm::vec3(1.0f, 1.0f, 1.3f) : glm::vec3(0.0f);
                constant = 1.0f;
                linear = 0.014f;
```

```
                quadratic = 0.0007f;
            }
            else {
                ambient = lightStates[i] ? glm::vec3(0.15f, 0.05f, 0.0f) : glm::vec3(0.0f);       // tenue
                diffuse = lightStates[i] ? glm::vec3(1.0f, 0.4f, 0.1f) : glm::vec3(0.0f);         // naranja-rojizo "chakra"
                specular = lightStates[i] ? glm::vec3(1.0f, 0.6f, 0.2f) : glm::vec3(0.0f);        // brillo cálido

                // Atenuación suave para que no se expanda demasiado
                constant = 1.0f;
                linear = 0.07f;
                quadratic = 0.017f;
            }

            glUniform3f(glGetUniformLocation(lightingShader.Program, (prefix + ".position").c_str()),
                pointLightPositions[i].x, pointLightPositions[i].y, pointLightPositions[i].z);
            glUniform3f(glGetUniformLocation(lightingShader.Program, (prefix + ".ambient").c_str()),
                ambient.x, ambient.y, ambient.z);
            glUniform3f(glGetUniformLocation(lightingShader.Program, (prefix + ".diffuse").c_str()),
                diffuse.x, diffuse.y, diffuse.z);
            glUniform3f(glGetUniformLocation(lightingShader.Program, (prefix + ".specular").c_str()),
                specular.x, specular.y, specular.z);
            glUniform1f(glGetUniformLocation(lightingShader.Program, (prefix + ".constant").c_str()), constant);
            glUniform1f(glGetUniformLocation(lightingShader.Program, (prefix + ".linear").c_str()), linear);
            glUniform1f(glGetUniformLocation(lightingShader.Program, (prefix + ".quadratic").c_str()), quadratic);
        }

        // Inicializar spotLight (apagado) - IMPORTANTE para que el shader funcione
        glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.position"), 0.0f, 0.0f, 0.0f);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.direction"), 0.0f, -1.0f, 0.0f);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.cutOff"), glm::cos(glm::radians(12.5f)));
        glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.outerCutOff"), glm::cos(glm::radians(15.0f)));
        glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.ambient"), 0.0f, 0.0f, 0.0f);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.diffuse"), 0.0f, 0.0f, 0.0f);
        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"), 0.09f);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.quadratic"), 0.032f);

        glm::mat4 view = camera.GetViewMatrix();
        glm::mat4 model = glm::mat4(1);

        GLint modelLoc = glGetUniformLocation(lightingShader.Program, "model");
```

```
        glUniformMatrix4fv(glGetUniformLocation(lightingShader.Program, "projection"), 1, GL_FALSE, glm::value_ptr(projection));
        glUniformMatrix4fv(glGetUniformLocation(lightingShader.Program, "view"), 1, GL_FALSE, glm::value_ptr(view));

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

        // IMPORTANTE: Configurar los samplers de textura
        glUniform1i(glGetUniformLocation(lightingShader.Program, "material.diffuse"), 0);
        glUniform1i(glGetUniformLocation(lightingShader.Program, "material.specular"), 0);

        // ===== PRIMERO: FUEGOS ARTIFICIALES (para que estén detrás) =====
        if (fuegosActivos) {
            glEnable(GL_BLEND);
            glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
            // NO desactives el depth test aquí para que respeten la profundidad
            // glDisable(GL_DEPTH_TEST); <-- QUITA ESTA LÍNEA

            for (int i = 0; i < MAX_PARTICULAS; i++) {
                if (particulas[i].activa) {
                    model = glm::mat4(1.0f);
                    model = glm::translate(model, particulas[i].posicion);

                    float factorVida = particulas[i].vida / particulas[i].vidaInicial;
                    float escala = particulas[i].tamano * factorVida;
                    model = glm::scale(model, glm::vec3(escala));

                    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
                    glBindTexture(GL_TEXTURE_2D, texturaParticula);
                    glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 128.0f);

                    ParticulaFuego->Draw();
                }
            }

            // glEnable(GL_DEPTH_TEST); <-- QUITA ESTA LÍNEA TAMBIÉN
            glDisable(GL_BLEND);
        }
        // Renderizado de objetos opacos
        model = glm::mat4(1.0f);
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));

        // Bind texture y dibujar Arroz
        glActiveTexture(GL_TEXTURE0);
        glBindTexture(GL_TEXTURE_2D, textureArroz);
```

```
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 48.0f);
Arroz.Draw(lightingShader);

// Base
glBindTexture(GL_TEXTURE_2D, textureBase);
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 32.0f);
Base.Draw(lightingShader);

// Barandal
glBindTexture(GL_TEXTURE_2D, textureBarandal);
Barandal.Draw(lightingShader);

// Botes
glBindTexture(GL_TEXTURE_2D, textureBotes);
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 24.0f);
Botes.Draw(lightingShader);

// BolsaHarinaAbierta
glBindTexture(GL_TEXTURE_2D, textureBolsaHarinaAbierta);
BolsaHarinaAbierta.Draw(lightingShader);

// BolsaHarinaAbierta1
glBindTexture(GL_TEXTURE_2D, textureBolsaHarinaAbierta1);
BolsaHarinaAbierta1.Draw(lightingShader);

// BolsasHarina
glBindTexture(GL_TEXTURE_2D, textureBolsasHarina);
BolsasHarina.Draw(lightingShader);

// BoteBasura
glBindTexture(GL_TEXTURE_2D, textureBoteBasura);
BoteBasura.Draw(lightingShader);

// CadenaLinterna
glBindTexture(GL_TEXTURE_2D, textureCadenaLinterna);
CadenaLinterna.Draw(lightingShader);

// Caja
glBindTexture(GL_TEXTURE_2D, textureCaja);
Caja.Draw(lightingShader);

// Cajas2
glBindTexture(GL_TEXTURE_2D, textureCajas2);
Cajas2.Draw(lightingShader);

// CajasMadera
glBindTexture(GL_TEXTURE_2D, textureCajasMadera);
CajasMadera.Draw(lightingShader);

// Campana
glBindTexture(GL_TEXTURE_2D, textureCampana);
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 56.0f);
Campana.Draw(lightingShader);

// Cortina
glBindTexture(GL_TEXTURE_2D, textureCortina);
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
Cortina.Draw(lightingShader);

// Cortinas
glBindTexture(GL_TEXTURE_2D, textureCortinas);
Cortinas.Draw(lightingShader);

// Cortina2
glBindTexture(GL_TEXTURE_2D, textureCortina2);
Cortina2.Draw(lightingShader);

// Cubeta
glBindTexture(GL_TEXTURE_2D, textureCubeta);
Cubeta.Draw(lightingShader);

// Electricidad
glBindTexture(GL_TEXTURE_2D, textureElectricidad);
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 40.0f);
Electricidad.Draw(lightingShader);

// Escaleras
glBindTexture(GL_TEXTURE_2D, textureEscaleras);
Escaleras.Draw(lightingShader);

// Estufa
glBindTexture(GL_TEXTURE_2D, textureEstufa);
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 64.0f);
Estufa.Draw(lightingShader);

// Fregadero
glBindTexture(GL_TEXTURE_2D, textureFregadero);
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 72.0f);
Fregadero.Draw(lightingShader);

// Gas
glBindTexture(GL_TEXTURE_2D, textureGas);
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 48.0f);
Gas.Draw(lightingShader);

// Jalador
glBindTexture(GL_TEXTURE_2D, textureJalador);
Jalador.Draw(lightingShader);


// LamparasCocina
glBindTexture(GL_TEXTURE_2D, textureLamparasCocina);
LamparasCocina.Draw(lightingShader);

// Librero
glBindTexture(GL_TEXTURE_2D, textureLibrero);
Librero.Draw(lightingShader);

// Mesa
glBindTexture(GL_TEXTURE_2D, textureMesa);
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 40.0f);
Mesa.Draw(lightingShader);

// Olla
glBindTexture(GL_TEXTURE_2D, textureOlla);
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 96.0f);
Olla.Draw(lightingShader);

// Olla1
glBindTexture(GL_TEXTURE_2D, textureOlla1);
Olla1.Draw(lightingShader);

// Olla2
glBindTexture(GL_TEXTURE_2D, textureOlla2);
Olla2.Draw(lightingShader);

// Olla3
glBindTexture(GL_TEXTURE_2D, textureOlla3);
```

```
        Olla3.Draw(lightingShader);

        // Pasto
        glBindTexture(GL_TEXTURE_2D, texturePasto);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 8.0f);
        Pasto.Draw(lightingShader);

        // Pared3
        glBindTexture(GL_TEXTURE_2D, texturePared3);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 24.0f);
        Pared3.Draw(lightingShader);

        // Pared4
        glBindTexture(GL_TEXTURE_2D, texturePared4);
        Pared4.Draw(lightingShader);

        // Pared
        glBindTexture(GL_TEXTURE_2D, texturePared);
        Pared.Draw(lightingShader);

        // Platos
        glBindTexture(GL_TEXTURE_2D, texturePlatos);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 56.0f);
        Platos.Draw(lightingShader);

        // PrimerPiso
        glBindTexture(GL_TEXTURE_2D, texturePrimerPiso);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 32.0f);
        PrimerPiso.Draw(lightingShader);

        // Piso2
        glBindTexture(GL_TEXTURE_2D, texturePiso2);
        Piso2.Draw(lightingShader);

        // Puertas
        glBindTexture(GL_TEXTURE_2D, texturePuertas);
        Puertas.Draw(lightingShader);

        // Refrigerador
        glBindTexture(GL_TEXTURE_2D, textureRefrigerador);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 80.0f);
        Refrigerador.Draw(lightingShader);
```

```
        // Silla2
        glBindTexture(GL_TEXTURE_2D, textureSilla2);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 20.0f);
        Silla2.Draw(lightingShader);

        // Silla
        glBindTexture(GL_TEXTURE_2D, textureSilla);
        Silla.Draw(lightingShader);

        // Tapa
        glBindTexture(GL_TEXTURE_2D, textureTapa);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 48.0f);
        Tapa.Draw(lightingShader);

        // Tapa1
        glBindTexture(GL_TEXTURE_2D, textureTapa1);
        Tapa1.Draw(lightingShader);

        // Tapa2
        glBindTexture(GL_TEXTURE_2D, textureTapa2);
        Tapa2.Draw(lightingShader);

        // TapeteAlmacen
        glBindTexture(GL_TEXTURE_2D, textureTapeteAlmacen);
        TapeteAlmacen.Draw(lightingShader);

        // TanqueAgua
        glBindTexture(GL_TEXTURE_2D, textureTanqueAgua);
        TanqueAgua.Draw(lightingShader);

        // TapeteCocina
        glBindTexture(GL_TEXTURE_2D, textureTapeteCocina);
        TapeteCocina.Draw(lightingShader);

        // Tazon
        glBindTexture(GL_TEXTURE_2D, textureTazon);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 64.0f);
        Tazon.Draw(lightingShader);

        // Tazones2
        glBindTexture(GL_TEXTURE_2D, textureTazones2);
        Tazones2.Draw(lightingShader);

```

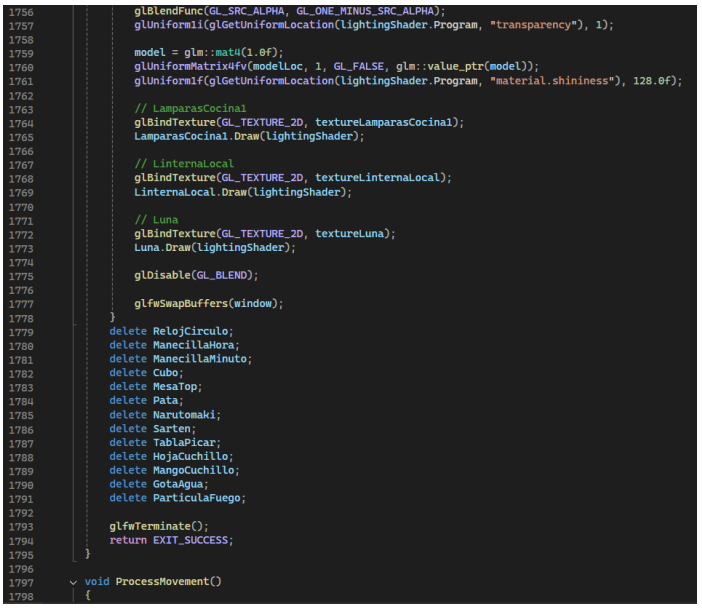
```
        // Tejado
        glBindTexture(GL_TEXTURE_2D, textureTejado);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
        Tejado.Draw(lightingShader);

        // TejadoMini
        glBindTexture(GL_TEXTURE_2D, textureTejadoMini);
        TejadoMini.Draw(lightingShader);

        // TejadoMiniPiso2
        glBindTexture(GL_TEXTURE_2D, textureTejadoMiniPiso2);
        TejadoMiniPiso2.Draw(lightingShader);

        // TuboFuera
        glBindTexture(GL_TEXTURE_2D, textureTuboFuera);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 80.0f);
        TuboFuera.Draw(lightingShader);

        // TuboGas
        glBindTexture(GL_TEXTURE_2D, textureTuboGas);
        TuboGas.Draw(lightingShader);

        // VentanaSegundoPiso
        glBindTexture(GL_TEXTURE_2D, textureVentanaSegundoPiso);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 24.0f);
        VentanaSegundoPiso.Draw(lightingShader);

        // ===== BANDERINES CON CONTROL MANUAL (ADELANTE/ATRÁS) =====
        model = glm::mat4(1.0f);

        // La rotación se aplica en el eje X para mover la punta hacia adelante/atrás
        model = glm::rotate(model, glm::radians(banderinesInclinacion), glm::vec3(1.0f, 0.0f, 0.0f));

        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        glBindTexture(GL_TEXTURE_2D, textureBanderines);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
        Banderines.Draw(lightingShader);

        // ===== RELOJ =====
        model = glm::mat4(1.0f);
        model = glm::translate(model, posReloj);
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 32.0f);
```

```
1627            RelojCirculo->Draw();
1628
1629            // Manecilla de hora (gira lento)
1630            float horaAngulo = (float)glfwGetTime() * 0.1f;
1631            model = glm::translate(glm::mat4(1.0f), posReloj);
1632            model = glm::rotate(model, horaAngulo, glm::vec3(0.0f, 0.0f, -1.0f));
1633            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1634            ManecillaHora->Draw();
1635
1636            // Manecilla de minuto (gira rápido)
1637            float minutoAngulo = (float)glfwGetTime() * 0.5f;
1638            model = glm::translate(glm::mat4(1.0f), posReloj);
1639            model = glm::rotate(model, minutoAngulo, glm::vec3(0.0f, 0.0f, -1.0f));
1640            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1641            ManecillaMinuto->Draw();
1642
1643            // ===== CUBO (CAJA) =====
1644            model = glm::mat4(1.0f);
1645            model = glm::translate(model, posCaja);
1646            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1647            glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
1648            Cubo->Draw();
1649
1650            // ===== MESA CON PATAS =====
1651            // Superficie de la mesa
1652            model = glm::mat4(1.0f);
1653            model = glm::translate(model, posMesa);
1654            model = glm::scale(model, glm::vec3(1.5f, 1.0f, 1.5f));
1655            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1656            glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 40.0f);
1657            MesaTop->Draw();
1658
1659            // Pata 1 (esquina -X, -Z)
1660            model = glm::mat4(1.0f);
1661            model = glm::translate(model, posMesa + glm::vec3(-0.65f, 0.0f, -0.65f));
1662            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1663            Pata->Draw();
1664
1665            // Pata 2 (esquina +X, -Z)
1666            model = glm::mat4(1.0f);
1667            model = glm::translate(model, posMesa + glm::vec3(0.65f, 0.0f, -0.65f));
1668            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1669            Pata->Draw();
```

```
1671            // Pata 3 (esquina -X, +Z)
1672            model = glm::mat4(1.0f);
1673            model = glm::translate(model, posMesa + glm::vec3(-0.65f, 0.0f, 0.65f));
1674            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1675            Pata->Draw();
1676
1677            // Pata 4 (esquina +X, +Z)
1678            model = glm::mat4(1.0f);
1679            model = glm::translate(model, posMesa + glm::vec3(0.65f, 0.0f, 0.65f));
1680            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1681            Pata->Draw();
1682
1683            // ===== TABLA DE PICAR =====
1684            model = glm::mat4(1.0f);
1685            model = glm::translate(model, posTabla);
1686            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1687            glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 24.0f);
1688            TablaPicar->Draw();
1689
1690            // ===== SARTÉN =====
1691            model = glm::mat4(1.0f);
1692            model = glm::translate(model, posSarten);
1693            model = glm::scale(model, glm::vec3(2.8f, 1.0f, 2.8f));
1694            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1695            glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 64.0f);
1696            Sarten->Draw();
1697
1698            // ===== NARUTOMAKI (3 piezas en fila) =====
1699            for (int i = 0; i < 3; i++) {
1700                model = glm::mat4(1.0f);
1701                model = glm::translate(model, glm::vec3(-1.60f + i * 0.1f, 1.20f, 1.5f));
1702                glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1703                glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 20.0f);
1704                Narutomaki->Draw();
1705            }
1706            // ===== CUCHILLO (HOJA + MANGO) =====
1707            glm::vec3 posCuchillo = glm::vec3(-1.80f, 1.20f, 1.5f);
1708
1709            // Hoja del cuchillo (rotada 45 grados)
1710            model = glm::mat4(1.0f);
1711            model = glm::translate(model, posCuchillo);
1712            model = glm::rotate(model, glm::radians(45.0f), glm::vec3(0.0f, 1.0f, 0.0f));
1713            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
```

```
1714            glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 96.0f);
1715            HojaCuchillo->Draw();
1716
1717            // Mango del cuchillo (atrás de la hoja)
1718            model = glm::translate(model, glm::vec3(0.0f, 0.0f, -0.25f));
1719            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1720            glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
1721            MangoCuchillo->Draw();
1722
1723            // ===== GOTAS DE AGUA =====
1724            if (aguaCayendo) {
1725                glEnable(GL_BLEND);
1726                glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
1727
1728                for (int i = 0; i < MAX_GOTAS; i++) {
1729                    if (gotas[i].activa) {
1730                        model = glm::mat4(1.0f);
1731                        model = glm::translate(model, gotas[i].posicion);
1732                        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1733                        glBindTexture(GL_TEXTURE_2D, textureAgua);
1734                        glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 128.0f);
1735                        GotaAgua->Draw();
1736                    }
1737                }
1738
1739                glDisable(GL_BLEND);
1740            }
1741
1742            // Animación del refrigerador
1743            glm::vec3 posBisagraRefrigerador = glm::vec3(2.5f, 1.0f, -3.60f);
1744            model = glm::mat4(1.0f);
1745            model = glm::translate(model, posBisagraRefrigerador);
1746            model = glm::rotate(model, glm::radians(-puertaRefrigerador), glm::vec3(0.0f, 1.0f, 0.0f));
1747            model = glm::translate(model, -posBisagraRefrigerador);
1748            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1749            glBindTexture(GL_TEXTURE_2D, textureRefrigerador1);
1750            glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 80.0f);
1751            Refrigerador1.Draw(lightingShader);
1752
1753
1754            // Objetos transparentes (lámparas)
1755            glEnable(GL_BLEND);
1756            glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

This code completes the rendering loop with three important sections: first, it animates the refrigerator door using rotation transformations around a hinge (allowing it to open/close depending on the value of the refrigeratorDoor variable); then, it enables transparency mode with blending to render semi-transparent objects like the kitchen lamps, lantern, and the moon; finally, it swaps the display buffers to show the rendered frame. Upon exiting the main loop, it frees the memory of all dynamically created custom geometries, terminates GLFW, and successfully closes the program.



```
1756            glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
1757            glUniform1i(glGetUniformLocation(lightingShader.Program, "transparency"), 1);
1758
1759            model = glm::mat4(1.0f);
1760            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1761            glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 128.0f);
1762
1763            // LamparasCocina1
1764            glBindTexture(GL_TEXTURE_2D, textureLamparasCocina1);
1765            LamparasCocina1.Draw(lightingShader);
1766
1767            // LinternaLocal
1768            glBindTexture(GL_TEXTURE_2D, textureLinternaLocal);
1769            LinternaLocal.Draw(lightingShader);
1770
1771            // Luna
1772            glBindTexture(GL_TEXTURE_2D, textureLuna);
1773            Luna.Draw(lightingShader);
1774
1775            glDisable(GL_BLEND);
1776
1777            glfwSwapBuffers(window);
1778        }
1779        delete RelojCirculo;
1780        delete ManecillaHora;
1781        delete ManecillaMinuto;
1782        delete Cubo;
1783        delete MesaTop;
1784        delete Pata;
1785        delete Narutomaki;
1786        delete Sarten;
1787        delete TablaPicar;
1788        delete HojaCuchillo;
1789        delete MangoCuchillo;
1790        delete GotaAgua;
1791        delete ParticulaFuego;
1792
1793        glfwTerminate();
1794        return EXIT_SUCCESS;
1795    }
1796
1797    void ProcessMovement()
1798    {
```

This section of code handles all user interaction and scene animations through three main functions. ProcessMovement() continuously updates camera movement with the WASD keys, smoothly animates the opening/closing of the refrigerator door, controls the tilt of the flags, simulates water droplets falling from the faucet with gravity physics, and generates firework explosions with randomly colored particles falling due to gravity. KeyCallback() captures keystrokes to: toggle four lights on/off (keys 1-4), open/close the refrigerator door (O/C), move flags back or to their normal position (N/M), turn the faucet water and fireworks on/off (G) and close the window with ESC. MouseCallback() tracks mouse movement to control the first-person camera's viewing direction.

```
        // Movimiento de cámara
        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);

        // Animación automática de la puerta
        if (animandoPuerta) {
            if (abs(puertaRefrigerador - puertaDestino) > 0.1f) {
                if (puertaRefrigerador < puertaDestino)
                    puertaRefrigerador += 80.0f * deltaTime;
                else
                    puertaRefrigerador -= 80.0f * deltaTime;
            }
            else {
                puertaRefrigerador = puertaDestino;
                animandoPuerta = false;
            }
        }

        // **ANIMACIÓN: Banderines moviéndose hacia adelante/atrás**
        if (animandoBanderines) {
            if (abs(banderinesInclinacion - banderinesDestino) > 0.1f) {
                if (banderinesInclinacion < banderinesDestino)
                    banderinesInclinacion += 60.0f * deltaTime;
                else
                    banderinesInclinacion -= 60.0f * deltaTime;
            }
            else {
                banderinesInclinacion = banderinesDestino;
                animandoBanderines = false;
            }
        }

        // **ANIMACIÓN: Agua cayendo del grifo**
        if (aguaCayendo) {
            // Crear nuevas gotas
            if (currentFrame - tiempoUltimaGota >= intervaloGotas) {
                for (int i = 0; i < MAX_GOTAS; i++) {
```

```
                    if (!gotas[i].activa) {
                        gotas[i].activa = true;
                        gotas[i].posicion = posicionGrifo;
                        gotas[i].velocidad = 2.0f + (rand() % 100) / 100.0f;
                        tiempoUltimaGota = currentFrame;
                        break;
                    }
                }
            }

            // Actualizar posición de gotas activas
            for (int i = 0; i < MAX_GOTAS; i++) {
                if (gotas[i].activa) {
                    gotas[i].posicion.y -= gotas[i].velocidad * deltaTime;
                    gotas[i].velocidad += 3.0f * deltaTime; // Gravedad

                    // Desactivar gota si llega al fregadero
                    if (gotas[i].posicion.y < 1.0f) {
                        gotas[i].activa = false;
                    }
                }
            }
        }
        // **ANIMACIÓN: Fuegos artificiales**
        if (fuegosActivos) {
            if (currentFrame - tiempoUltimoFuego >= intervaloFuegos) {
                // MÁS SEPARADAS HORIZONTALMENTE (de -8 a +8)
                glm::vec3 posExplosion = glm::vec3(
                    -8.0f + (rand() % 1600) / 100.0f,    // -8 a 8 en X (MUY ANCHO)
                    7.0f + (rand() % 200) / 100.0f,      // 7 a 9 en Y (altura similar)
                    -10.0f                               // Z FIJO (sin profundidad)
                );

                // Colores vivos y variados
                int tipoColor = rand() % 5;
                glm::vec3 colorExplosion;

                switch (tipoColor) {
                case 0: colorExplosion = glm::vec3(1.0f, 0.2f, 0.2f); break; // Rojo
                case 1: colorExplosion = glm::vec3(0.2f, 1.0f, 0.2f); break; // Verde
                case 2: colorExplosion = glm::vec3(0.2f, 0.2f, 1.0f); break; // Azul
                case 3: colorExplosion = glm::vec3(1.0f, 1.0f, 0.2f); break; // Amarillo
                case 4: colorExplosion = glm::vec3(1.0f, 0.2f, 1.0f); break; // Magenta
```

```
                }

                CrearExplosion(posExplosion, colorExplosion);
                tiempoUltimoFuego = currentFrame;

                std::cout << "¡BOOM! Explosión en el fondo Z=" << posExplosion.z << std::endl;
            }

            // Actualizar partículas activas
            for (int i = 0; i < MAX_PARTICULAS; i++) {
                if (particulas[i].activa) {
                    // Actualizar posición
                    particulas[i].posicion += particulas[i].velocidad * deltaTime;

                    // Aplicar gravedad suave
                    particulas[i].velocidad.y -= 2.5f * deltaTime;

                    // Fricción del aire
                    particulas[i].velocidad *= 0.98f;

                    // Reducir vida
                    particulas[i].vida -= deltaTime;

                    // Desactivar si ya no tiene vida
                    if (particulas[i].vida <= 0.0f) {
                        particulas[i].activa = false;
                    }
                }
            }
        }
    }
    void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode)
    {
        (void)scancode;
        (void)mode;

        if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
            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 (action == GLFW_PRESS)
    {
        switch (key)
        {
        case GLFW_KEY_1:
            lightStates[0] = !lightStates[0];
            std::cout << "Luz 1 (LamparasCocina1): " << (lightStates[0] ? "ENCENDIDA" : "APAGADA") << std::endl;
            break;
        case GLFW_KEY_2:
            lightStates[1] = !lightStates[1];
            std::cout << "Luz 2 (Lamparas1): " << (lightStates[1] ? "ENCENDIDA" : "APAGADA") << std::endl;
            break;
        case GLFW_KEY_3:
            lightStates[2] = !lightStates[2];
            std::cout << "Luz 3 (LinternaLocal): " << (lightStates[2] ? "ENCENDIDA" : "APAGADA") << std::endl;
            break;
        case GLFW_KEY_4:
            lightStates[3] = !lightStates[3];
            std::cout << "Luz Luna: " << (lightStates[3] ? "ENCENDIDA" : "APAGADA") << std::endl;
            break;
        case GLFW_KEY_O:
            puertaDestino = 90.0f;  // Abrir
            animandoPuerta = true;
            std::cout << "Abriendo puerta del refrigerador..." << std::endl;
            break;
        case GLFW_KEY_C:
            puertaDestino = 0.0f;   // Cerrar
            animandoPuerta = true;
            std::cout << "Cerrando puerta del refrigerador..." << std::endl;
            break;
        case GLFW_KEY_N: // Mover banderines hacia ATRÁS
            banderinesDestino = -banderinesMaxInclinacion;
            animandoBanderines = true;
            std::cout << "Banderines moviéndose hacia atrás..." << std::endl;
            break;

        case GLFW_KEY_M: // Regresar banderines a posición NORMAL
            banderinesDestino = 0.0f;
            animandoBanderines = true;
            std::cout << "Banderines regresando a posición normal..." << std::endl;
            break;
        case GLFW_KEY_G: // Toggle agua del grifo
            aguaCayendo = !aguaCayendo;
            std::cout << "Agua del grifo: " << (aguaCayendo ? "ABIERTA" : "CERRADA") << std::endl;
            if (!aguaCayendo) {
                for (int i = 0; i < MAX_GOTAS; i++) {
                    gotas[i].activa = false;
                }
            }
            break;

        case GLFW_KEY_F: // Toggle fuegos artificiales
            fuegosActivos = !fuegosActivos;
            std::cout << "Fuegos artificiales: " << (fuegosActivos ? "ACTIVADOS" : "DESACTIVADOS") << std::endl;
            if (!fuegosActivos) {
                for (int i = 0; i < MAX_PARTICULAS; i++) {
                    particulas[i].activa = false;
                }
            }
            break;
        }
    }
}

void MouseCallback(GLFWwindow* window, double xpos, double ypos)
{
    (void)window;

    if (firstMouse)
    {
        lastX = (GLfloat)xpos;
        lastY = (GLfloat)ypos;
        firstMouse = false;
    }

    GLfloat xOffset = (GLfloat)(xpos - lastX);
    GLfloat yOffset = (GLfloat)(lastY - ypos);

    lastX = (GLfloat)xpos;
    lastY = (GLfloat)ypos;

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

Results

The project achieved its objectives by developing an interactive 3D walkthrough experience based on the Ichiraku restaurant from the Naruto series. Using Blender for modeling and OpenGL for programming, the project effectively integrated the artistic and technical aspects, creating a visually appealing, functional, and dynamic environment.

Key results include:

- **Detailed 3D Modeling:** The restaurant's various elements were created, including furniture, decor, and distinctive objects, achieving a faithful representation of the original anime setting. This model was created in Blender.

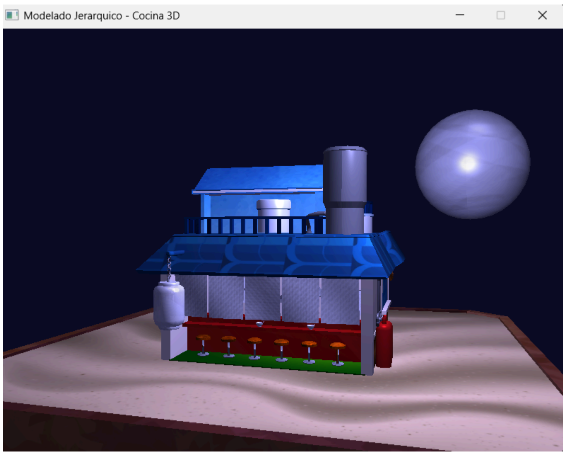
- **Application of Textures and Materials:** Each model features appropriate textures and materials that accurately simulate the appearance of wood, ceramics, and other elements present in the setting.

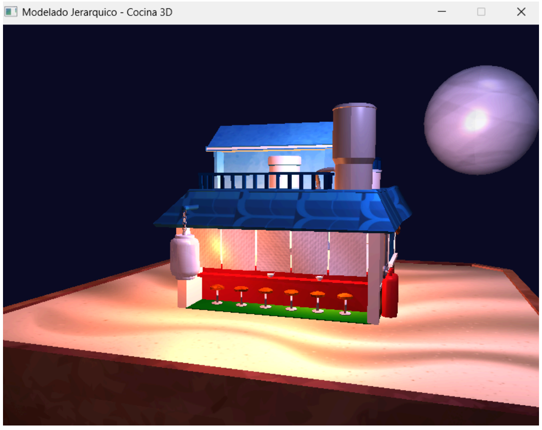
- **Realistic Lighting:** Various types of lighting were implemented using shaders, achieving a warm ambiance consistent with the atmosphere of Ichiraku Restaurant.

- **Integrated Animations:** Basic animations were incorporated into some of the objects in the set. There were five: a simulation of falling water from the sink, the opening and closing of the refrigerator door, a clock, the raising of the curtain, and fireworks. These animations brought the set to life and enhanced the sense of interactivity.

- **Interactivity and Traversal:** The user can freely explore the set, walking through the different spaces and observing the details of the models from multiple angles.

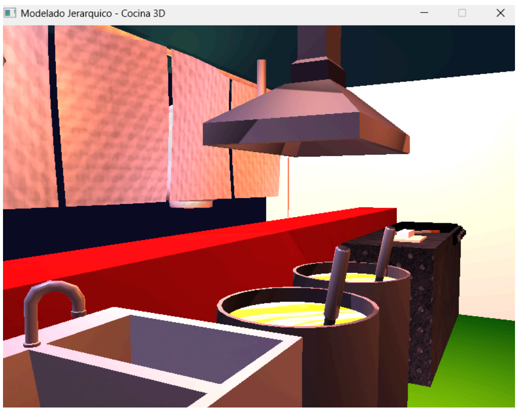
- **Library Integration and Optimization:** Libraries such as GLEW, GLFW, and SOIL were used to facilitate programming and texture management, achieving stable and fluid performance of the path.


Modelado Jerarquico - Cocina 3D


Modelado Jerarquico - Cocina 3D

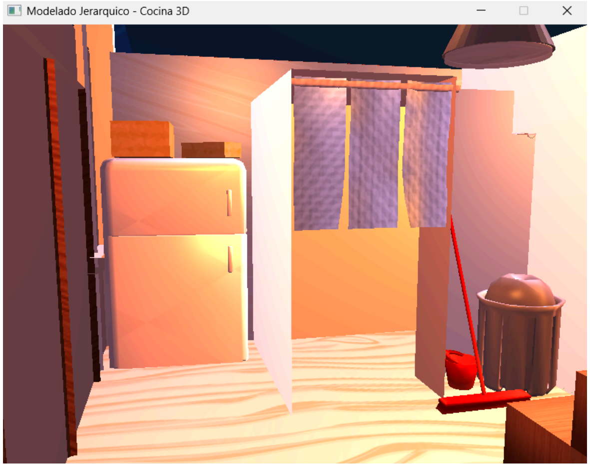

Modelado Jerarquico - Cocina 3D

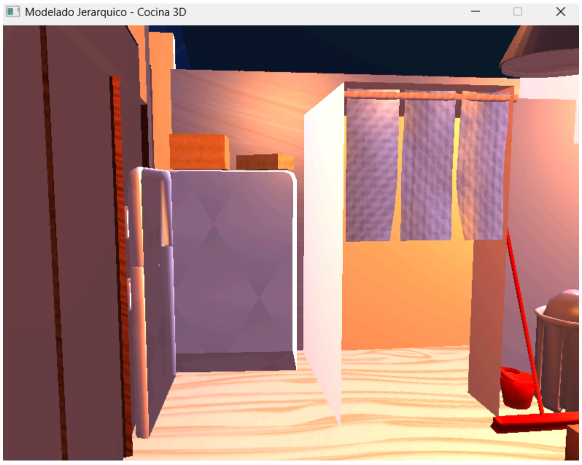

Modelado Jerarquico - Cocina 3D

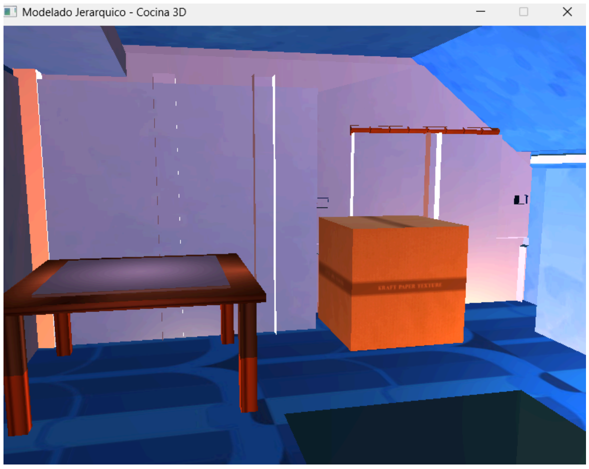

Modelado Jerarquico - Cocina 3D

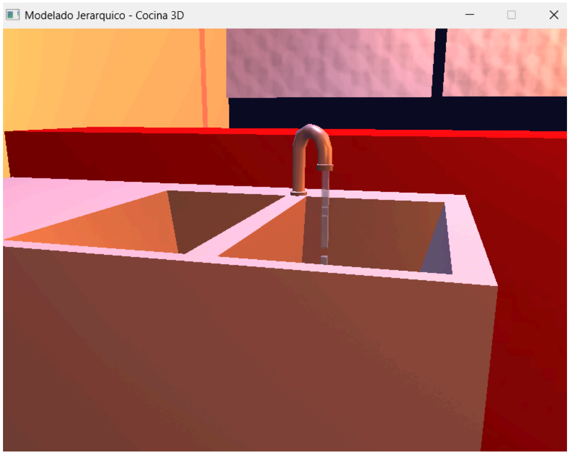

Modelado Jerarquico - Cocina 3D

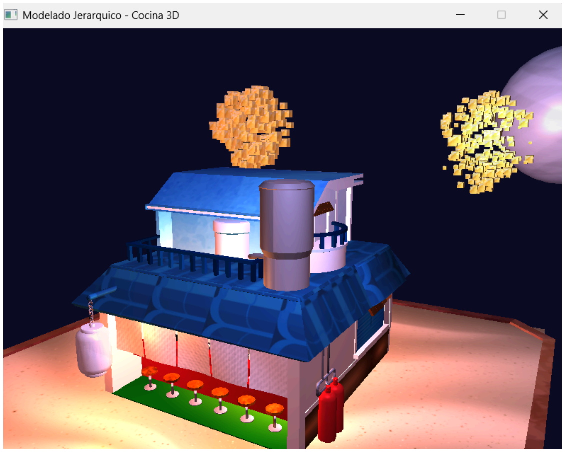

Modelado Jerarquico - Cocina 3D


Modelado Jerarquico - Cocina 3D


Modelado Jerarquico - Cocina 3D


Modelado Jerarquico - Cocina 3D


Modelado Jerarquico - Cocina 3D


Modelado Jerarquico - Cocina 3D

- **FILE https://github.com/samgarciagallegos-cloud/319181386_PROYECTOFINAL2026-1_GPO-5**

Conclusions

Through this project, I learned to combine 3D modeling in Blender with OpenGL graphics programming, applying textures, lighting, and animations to create interactive paths. At the same time, I developed technical and creative skills in the design and development of 3D environments. I believe this concludes and reinforces everything I learned in Graphics class.

Bibliography

- Blender Foundation. (2023). *Blender 3D: Noob to Pro*. https://docs.blender.org/manual/es/latest/
- Eberly, D. H. (2010). *3D game engine design: A practical approach to real-time computer graphics* (2.ª ed.). Morgan Kaufmann.
- Foley, J. D., van Dam, A., Feiner, S. K., & Hughes, J. F. (2014). *Computer graphics: Principles and practice* (3.ª ed.). Addison-Wesley.
- GLFW. (2023). *GLFW – A library for OpenGL, OpenGL ES and Vulkan development on the desktop*. https://www.glfw.org
- Kilgard, M. J. (2009). *OpenGL programming for the X Window System*. Addison-Wesley.
- Shreiner, D., Sellers, G., Kessenich, J., & Licea-Kane, B. (2013). *OpenGL programming guide: The official guide to learning OpenGL, Version 4.3* (8.ª ed.). Addison-Wesley.
- SOIL – Simple OpenGL Image Library. (2023). http://www.lonesock.net/soil.html