



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.....	3
Objetivo General.....	3
Objetivos Específicos.....	3
Metas.....	3
Alcances.....	3
Limitantes.....	4
Hitos del proyecto.....	4
Cronograma del proyecto.....	5
Diagrama del flujo de software.....	6
Diagrama de gantt.....	7
Cotización del proyecto.....	7
Introducción.....	8
Estados del arte recorridos virtuales.....	12
Marco Teórico.....	12
Blender.....	12
OpenGL.....	13
Metodología.....	13
Código.....	15
Resultados.....	37
Conclusiones.....	40
Github.....	40
Bibliografía.....	40
Objectives.....	41
General Objective.....	41
Specific Objectives.....	41
Goals.....	41
Scope.....	41
Constraints.....	42
Project Milestones.....	42
Project Schedule.....	43
Software Flow Diagram.....	44
Gantt Chart.....	45
Project Estimate.....	45
Introduction.....	46
Theoretical Framework.....	50
Blender.....	50
OpenGL.....	51
Methodology.....	51
Code.....	52
Results.....	75
Conclusions.....	78
Github.....	78
Bibliography.....	78

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

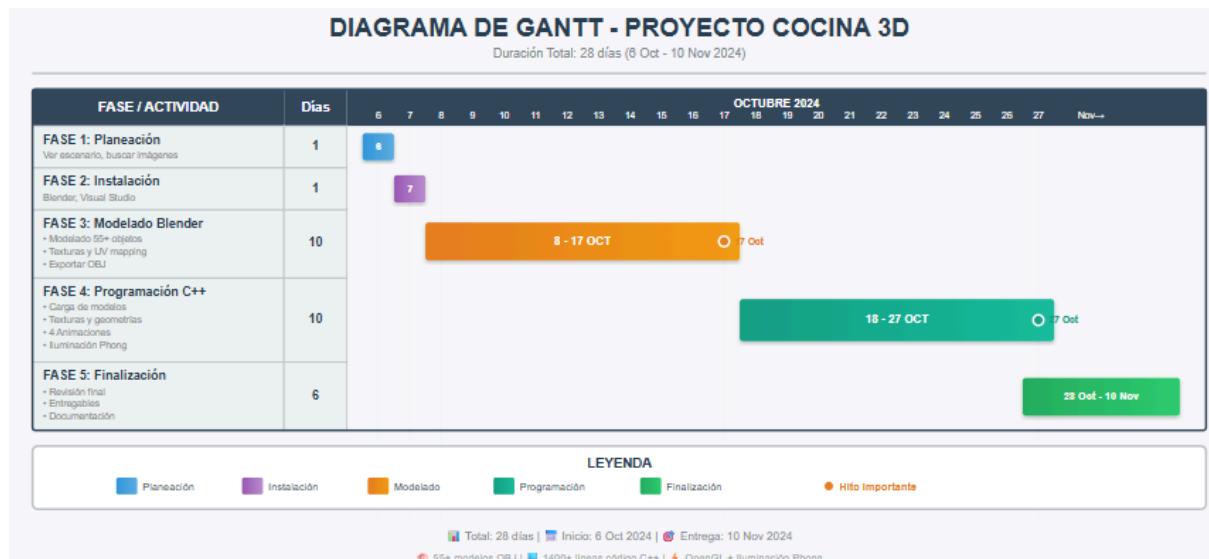
Cronograma del proyecto

Fase	Actividades	Duración	Fecha límite
Fase 1: Planeación del proyecto	<ul style="list-style-type: none"> • Ver el escenario que sería. • Buscar las imágenes del escenario. 	1 día	6 de Octubre.
Fase 2: Instalación de blender y visual studio.	<ul style="list-style-type: none"> • Instalación de aplicaciones. • Configuraciones iniciales de blender y Visual Studio. 	1 día	7 Octubre.
Fase 3: Modelado en blender de los objetos.	<ul style="list-style-type: none"> • Modelado de blender de objetos a utilizar. • Elegir texturas y colocar. <p>Exportar modelos de blender en formato Obj.</p>	10 días.	17 Octubre.
Fase 4: Creación del código en visual studio.	<ul style="list-style-type: none"> • 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.	<ul style="list-style-type: none"> • 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, glfw, 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

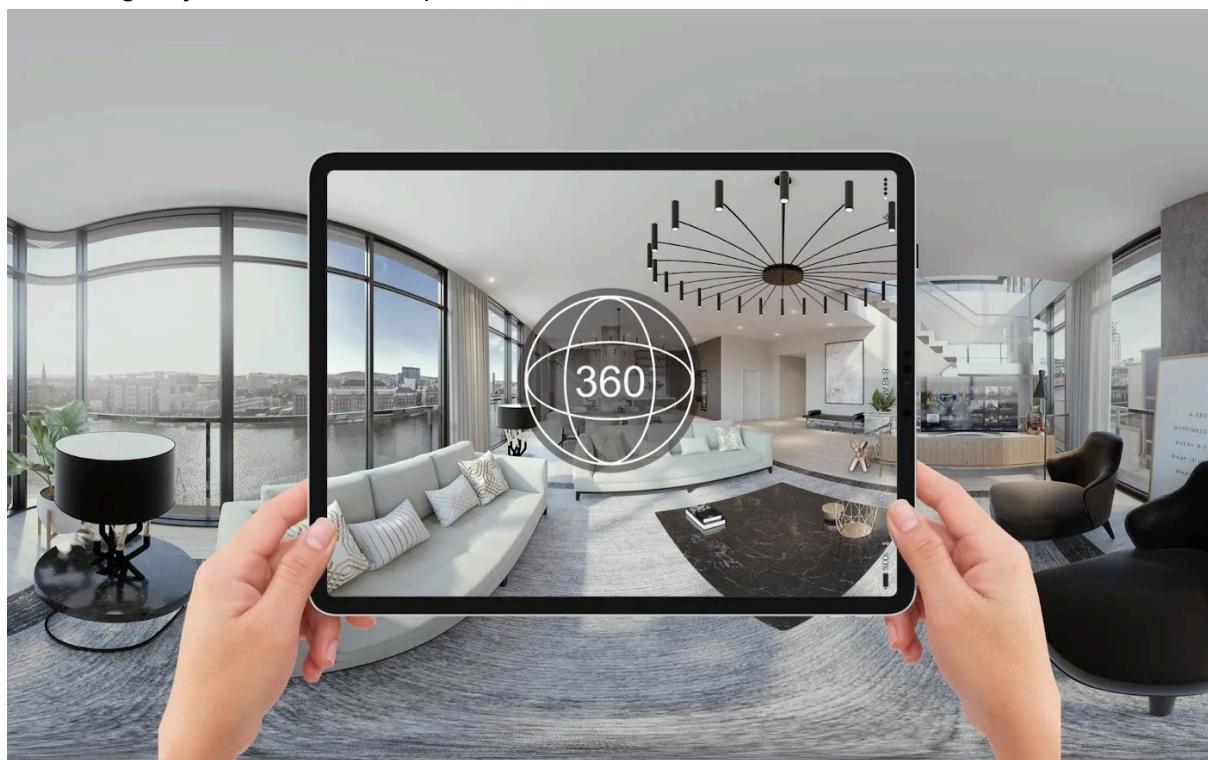


5)Tarros



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).

```

1  //<include <iostream>
2  #include <string>
3  #include <vector>
4  #include <cmath>
5
6  #include <GL/glew.h>
7  #include <GLFW/glfw3.h>
8
9  #include <glm/glm.hpp>
10 #include <glm/gtc/matrix_transform.hpp>
11 #include <glm/gtc/type_ptr.hpp>
12
13 #include <stb_image.h>
14 #include <shader.h>
15 #include <camera.h>
16 #include <Model.h>
17 #include <ctime>
18 #include <stdlib.h>
19
20 // Constante PI
21 const float PI = 3.14159265359f;
22
23 // Variables
24 const GLuint WINDOW_WIDTH = 800;
25 const GLuint WINDOW_HEIGHT = 600;
26
27 // Variables globales para cámara
28 Camera camera(glm::vec3(0.0f, 0.0f, 3.0f));
29 GLfloat lastX = WINDOW_WIDTH / 2.0f;
30 GLfloat lastY = WINDOW_HEIGHT / 2.0f;
31 bool firstMouse = true;
32 bool keys[1024] = { false };
33
34 // Variables para animaciones del modelo
35 float puertaRefrigerador = 0.0f;
36 bool animandoPuerta = false;
37 float puertaDestino = 0.0f;
38
39 // Variables para animación de banderines
40 float banderinesInclinación = 0.0f;
41 bool animandoBanderines = false;
42 float banderinesDestino = 0.0f;
43 const float banderinesMaxInclinación = 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.

```

43  [ ] SISTEMA DE ILUMINACION
44  bool lightStates[4] = { false, false, false, true }; // Estados de las 3 lámparas + Luna
45
46  // Posiciones de las luces puntuales
47  glm::vec3 pointLightPositions[] = {
48      glm::vec3(0.0f, 3.0f, -2.0f), // LamparasCuarto2
49      glm::vec3(0.0f, 3.0f, 2.0f), // LamparasCuarto1
50      glm::vec3(-2.0f, 2.0f, 5.0f), // LinternalLocal
51      glm::vec3(4.0f, 8.0f, 5.0f) // Luna
52  };
53
54  // Deltatime
55  GLfloat deltaTime = 0.0f;
56  GLfloat lastFrame = 0.0f;
57  GLfloat currentFrame = 0.0f;
58  // Variables para animación de agua del grifo (AGREGAR)
59  bool aguaCayendo = false;
60
61  struct Gota {
62      glm::vec3 posicion;
63      float velocidad;
64      bool activa;
65  };
66  std::vector<Gota> gotas;
67  const int MAX_GOTAS = 50;
68  glm::vec3 posicionGrifo = glm::vec3(1.97f, 1.3f, 1.77f); // Ajusta según tu fregadero
69  float tiempolatimaGota = 0.0f;
70  const float intervaloGotas = 0.02f;
71  // Variables para fuegos artificiales
72  bool fuegosActivos = false;
73  struct Particula {
74      glm::vec3 posicion;
75      glm::vec3 velocidad;
76      glm::vec3 color;
77      float vida;
78      float vidaInicial;
79      float tamano;
80      bool activa;
81  };
82  std::vector<Particula> particulas;
83  const int MAX_PARTICULAS = 1000;
84  float tiempolatimofuego = 0.0f;
85  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.

```
85
86     void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode);
87     void MouseCallback(GLFWwindow* window, double xPos, double yPos);
88     void ProcessMovement();
89
90     GLuint LoadTexture(const char* path)
91     {
92         GLuint textureID = 0;
93         glGenTextures(1, &textureID);
94
95         int width = 0, height = 0, nrChannels = 0;
96         stbi_set_flip_vertically_on_load(true);
97         unsigned char* data = stbi_load(path, &width, &height, &nrChannels, 0);
98
99         if (data)
100     {
101         GLenum format = GL_RGB;
102         if (nrChannels == 4)
103             format = GL_RGBA;
104         else if (nrChannels == 1)
105             format = GL_RED;
106
107         glBindTexture(GL_TEXTURE_2D, textureID);
108         glTexImage2D(GL_TEXTURE_2D, 0, format, width, height, 0, format, GL_UNSIGNED_BYTE, data);
109         glGenerateMipmap(GL_TEXTURE_2D);
110
111         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
112         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
113         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
114         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
115
116         std::cout << "Textura cargada: " << path << std::endl;
117     }
118     else
119     {
120         std::cout << "Error al cargar: " << path << std::endl;
121     }
122
123     stbi_image_free(data);
124     glBindTexture(GL_TEXTURE_2D, 0);
125
126     return textureID;
127 }
```

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 círculo
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 });
171     }
172 }
```

```

171     }
172
173     // indices
174     for (int i = 1; i <= segments; i++) {
175         indices.push_back(0);
176         indices.push_back(i);
177         indices.push_back(i + 1);
178     }
179
180     geo->indexCount = (GLuint)indices.size();
181
182     glGenVertexArrays(1, &geo->VAO);
183     glGenBuffers(1, &geo->VBO);
184     glGenBuffers(1, &geo->EBO);
185
186     glBindVertexArray(geo->VAO);
187
188     glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
189     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
190
191     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
192     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
193
194     // Posición
195     glEnableVertexAttribArray(0);
196     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
197     // Normal
198     glEnableVertexAttribArray(1);
199     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
200     // TexCoords
201     glEnableVertexAttribArray(2);
202     glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
203
204     glBindVertexArray(0);
205
206     return geo;
207 }
208
209 // Función para crear una manecilla del reloj
210 SimpleGeometry* CreateClockHand(GLuint texture, float width, float length) {
211     SimpleGeometry* geo = new SimpleGeometry();
212     geo->texture = texture;
213
214     std::vector<GLfloat> vertices = {
215         // Posición           Normal           TexCoords
216         -width / 2, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
217         width / 2, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
218         width / 2, length, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f,
219         -width / 2, length, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f
220     };
221
222     std::vector<GLuint> indices = { 0, 1, 2, 2, 3, 0 };
223     geo->indexCount = 6;
224
225     glGenVertexArrays(1, &geo->VAO);
226     glGenBuffers(1, &geo->VBO);
227     glGenBuffers(1, &geo->EBO);
228
229     glBindVertexArray(geo->VAO);
230
231     glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
232     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
233
234     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
235     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
236
237     glEnableVertexAttribArray(0);
238     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
239     glEnableVertexAttribArray(1);
240     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
241     glEnableVertexAttribArray(2);
242     glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
243
244     glBindVertexArray(0);
245
246     return geo;
247 }
248
249 // Función para crear un cubo
250 SimpleGeometry* CreateCube(GLuint texture) {
251     SimpleGeometry* geo = new SimpleGeometry();
252     geo->texture = texture;
253
254     std::vector<GLfloat> vertices = {
255         // Cara frontal (z+)

```

```

249     // Función para crear un cubo
250     SimpleGeometry* CreateCube(GLuint texture) {
251         SimpleGeometry* geo = new SimpleGeometry();
252         geo->texture = texture;
253
254         std::vector<GLfloat> vertices = {
255             // Cara frontal (z+)
256             -0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
257             0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f,
258             0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
259             -0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,
260
261             // Cara trasera (z-)
262             -0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
263             -0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
264             0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
265             0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f,
266
267             // Cara superior (y+)
268             -0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
269             -0.5f, 0.5f, 0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
270             0.5f, 0.5f, 0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
271             0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
272
273             // Cara inferior (y-)
274             -0.5f, -0.5f, -0.5f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
275             0.5f, -0.5f, -0.5f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
276             0.5f, -0.5f, 0.5f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
277             -0.5f, -0.5f, 0.5f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
278
279             // Cara derecha (x+)
280             0.5f, -0.5f, -0.5f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
281             0.5f, 0.5f, -0.5f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
282             0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
283             0.5f, -0.5f, 0.5f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
284
285             // Cara izquierda (x-)
286             -0.5f, -0.5f, -0.5f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
287             -0.5f, -0.5f, 0.5f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
288             -0.5f, 0.5f, 0.5f, -1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
289             -0.5f, 0.5f, -0.5f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f
290         };
291     }

```

```

292     std::vector<GLuint> indices = {
293         0, 1, 2, 3, 0,      // Frontal
294         4, 5, 6, 7, 4,    // Trasera
295         8, 9, 10, 10, 8,  // Superior
296         12, 13, 14, 14, 15, 12, // Inferior
297         16, 17, 18, 18, 19, 16, // Derecha
298         20, 21, 22, 22, 23, 20 // Izquierda
299     };
300
301     geo->indexCount = 36;
302
303     glGenVertexArrays(1, &geo->VAO);
304     glGenBuffers(1, &geo->VBO);
305     glGenBuffers(1, &geo->EBO);
306
307     glBindVertexArray(geo->VAO);
308
309     glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
310     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
311
312     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
313     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
314
315     glEnableVertexAttribArray(0);
316     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
317     glEnableVertexAttribArray(1);
318     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
319     glEnableVertexAttribArray(2);
320     glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
321
322     glBindVertexArray(0);
323
324     return geo;
325 }
326
327 // Función para crear superficie de mesa
328 SimpleGeometry* CreateMesaTop(GLuint texture) {
329     SimpleGeometry* geo = new SimpleGeometry();
330     geo->texture = texture;
331
332     std::vector<GLfloat> vertices = {
333         // Superficie superior
334         -0.5f, 0.05f, -0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
335         0.5f, 0.05f, -0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
336         0.5f, 0.05f, 0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
337         -0.5f, 0.05f, 0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
338
339         // Superficie inferior
340         -0.5f, 0.0f, -0.5f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
341         0.5f, 0.0f, -0.5f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
342         0.5f, 0.0f, 0.5f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
343         -0.5f, 0.0f, 0.5f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
344
345         // Borde frontal (Z-)
346         -0.5f, 0.0f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f,
347         0.5f, 0.0f, -0.5f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
348         0.5f, 0.05f, -0.5f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
349         -0.5f, 0.05f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
350
351         // Borde trasero (Z+)
352         -0.5f, 0.0f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
353         0.5f, 0.0f, 0.5f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f,
354         0.5f, 0.05f, 0.5f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f,
355         -0.5f, 0.05f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,
356
357         // Borde derecho (X+)
358         0.5f, 0.0f, -0.5f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
359         0.5f, 0.0f, 0.5f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
360         0.5f, 0.05f, 0.5f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
361         0.5f, 0.05f, -0.5f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
362
363         // Borde izquierdo (X-)
364         -0.5f, 0.0f, -0.5f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
365         -0.5f, 0.0f, 0.5f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
366         -0.5f, 0.05f, 0.5f, -1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
367         -0.5f, 0.05f, -0.5f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f
368     };
369
370     std::vector<GLuint> indices = {
371         0, 1, 2, 2, 3, 0,      // Superior
372         4, 6, 5, 4, 7, 6,    // Inferior
373         8, 9, 10, 10, 11, 8, // Borde frontal
374         12, 14, 13, 12, 15, 14, // Borde trasero
375         16, 17, 18, 18, 19, 16, // Borde derecho
376         20, 22, 21, 20, 23, 22 // Borde izquierdo
377     };
378 }

```

```

379     geo->indexCount = 36;
380
381     glGenVertexArrays(1, &geo->VAO);
382     glGenBuffers(1, &geo->VBO);
383     glGenBuffers(1, &geo->EBO);
384
385     glBindVertexArray(geo->VAO);
386
387     glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
388     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
389
390     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
391     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
392
393     glEnableVertexAttribArray(0);
394     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
395     glEnableVertexAttribArray(1);
396     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
397     glEnableVertexAttribArray(2);
398     glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
399
400     glBindVertexArray(0);
401
402     return geo;
403 }
404 // Función para crear pata de mesa
405 SimpleGeometry* CreatePataMesa(GLuint texture) {
406     SimpleGeometry* geo = new SimpleGeometry();
407     geo->texture = texture;
408
409     std::vector<GLfloat> vertices = {
410         // Cilindro simplificado como cubo delgado
411         -0.05f, -0.7f, -0.05f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
412         0.05f, -0.7f, -0.05f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
413         0.05f, -0.7f, 0.05f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
414         -0.05f, -0.7f, 0.05f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
415
416         -0.05f, 0.0f, -0.05f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
417         0.05f, 0.0f, -0.05f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
418         0.05f, 0.0f, 0.05f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
419         -0.05f, 0.0f, 0.05f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f
420     };
421
422     std::vector<GLuint> indices = {
423         0, 1, 2, 2, 3, 0, // Abajo
424         4, 6, 5, 4, 7, 6, // Arriba
425         0, 1, 5, 5, 4, 0, // Laterales
426         1, 2, 6, 6, 5, 1,
427         2, 3, 7, 7, 6, 2,
428         3, 0, 4, 4, 7, 3
429     };
430
431     geo->indexCount = 36;
432
433     glGenVertexArrays(1, &geo->VAO);
434     glGenBuffers(1, &geo->VBO);
435     glGenBuffers(1, &geo->EBO);
436
437     glBindVertexArray(geo->VAO);
438
439     glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
440     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
441
442     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
443     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
444
445     glEnableVertexAttribArray(0);
446     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
447     glEnableVertexAttribArray(1);
448     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
449     glEnableVertexAttribArray(2);
450     glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
451
452     glBindVertexArray(0);
453
454     return geo;
455 }
456 // Función para crear Narutomaki (cilindro pequeño)
457 SimpleGeometry* CreateNarutomaki(GLuint texture) {
458     SimpleGeometry* geo = new SimpleGeometry();
459     geo->texture = texture;
460
461     std::vector<GLfloat> vertices = {
462         // Base inferior (y = 0.0f)
463         -0.05f, 0.0f, -0.05f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
464         0.05f, 0.0f, -0.05f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,

```

```

465     0.05f,  0.0f,  0.05f,  0.0f, -1.0f,  0.0f,  1.0f, 1.0f,
466     -0.05f, 0.0f,  0.05f,  0.0f, -1.0f,  0.0f,  0.0f, 1.0f,
467
468     // Base superior (y = 0.05f)
469     -0.05f, 0.05f, -0.05f,  0.0f, 1.0f, 0.0f,  0.0f, 0.0f,
470     0.05f, 0.05f, -0.05f,  0.0f, 1.0f, 0.0f,  1.0f, 0.0f,
471     0.05f, 0.05f,  0.05f,  0.0f, 1.0f, 0.0f,  1.0f, 1.0f,
472     -0.05f, 0.05f,  0.05f,  0.0f, 1.0f, 0.0f,  0.0f, 1.0f,
473
474     // Lado frontal (Z+)
475     -0.05f, 0.0f,  0.05f,  0.0f, 0.0f, 1.0f,  0.0f, 0.0f,
476     0.05f, 0.0f,  0.05f,  0.0f, 0.0f, 1.0f,  1.0f, 0.0f,
477     0.05f, 0.05f,  0.05f,  0.0f, 0.0f, 1.0f,  1.0f, 0.0f,
478     -0.05f, 0.05f,  0.05f,  0.0f, 0.0f, 1.0f,  0.0f, 1.0f,
479
480     // Lado trasero (Z-)
481     -0.05f, 0.0f, -0.05f,  0.0f, 0.0f, -1.0f,  1.0f, 0.0f,
482     0.05f, 0.0f, -0.05f,  0.0f, 0.0f, -1.0f,  0.0f, 0.0f,
483     0.05f, 0.05f, -0.05f,  0.0f, 0.0f, -1.0f,  0.0f, 1.0f,
484     -0.05f, 0.05f, -0.05f,  0.0f, 0.0f, -1.0f,  1.0f, 0.0f,
485
486     // Lado derecho (X+)
487     0.05f, 0.0f, -0.05f,  1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
488     0.05f, 0.0f,  0.05f,  1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
489     0.05f, 0.05f,  0.05f,  1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
490     0.05f, 0.05f, -0.05f,  1.0f, 0.0f, 0.0f,  0.0f, 1.0f,
491
492     // Lado izquierdo (X-)
493     -0.05f, 0.0f, -0.05f, -1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
494     -0.05f, 0.0f,  0.05f, -1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
495     -0.05f, 0.05f,  0.05f, -1.0f, 0.0f, 0.0f,  1.0f, 1.0f,
496     -0.05f, 0.05f, -0.05f, -1.0f, 0.0f, 0.0f,  0.0f, 1.0f,
497 };
498
499     std::vector<GLuint> indices = {
500     0, 1, 2, 2, 3, 0,           // Base inferior
501     4, 6, 5, 4, 7, 6,           // Base superior
502     8, 9, 10, 10, 11, 8,        // Frontal
503     12, 14, 13, 12, 15, 14,    // Trasero
504     16, 17, 18, 18, 19, 16,    // Derecho
505     20, 22, 21, 20, 23, 22    // Izquierdo
506 };
507
508
509     geo->indexCount = 36;
510
511     glGenVertexArrays(1, &geo->VAO);
512     glGenBuffers(1, &geo->VBO);
513     glGenBuffers(1, &geo->EBO);
514
515     glBindVertexArray(geo->VAO);
516
517     glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
518     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
519
520     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
521     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
522
523     glEnableVertexAttribArray(0);
524     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
525     glEnableVertexAttribArray(1);
526     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
527     glEnableVertexAttribArray(2);
528     glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
529
530     glBindVertexArray(0);
531
532     return geo;
533 }
534
535     // Función para crear Sartén
536     SimpleGeometry* CreateSarten(GLuint texture) {
537     SimpleGeometry* geo = new SimpleGeometry();
538     geo->texture = texture;
539
540     std::vector<GLfloat> vertices = {
541     // Base inferior
542     -0.2f, 0.0f, -0.2f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
543     0.2f, 0.0f, -0.2f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
544     0.2f, 0.0f, 0.2f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
545     -0.2f, 0.0f, 0.2f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
546
547     // Base superior
548     -0.2f, 0.02f, -0.2f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
549     0.2f, 0.02f, -0.2f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
550     0.2f, 0.02f, 0.2f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
551     -0.2f, 0.02f, 0.2f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
552     -0.2f, 0.02f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
553     0.2f, 0.02f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f
554 };
555 }
```

```

552     // Lado frontal
553     -0.2f, 0.0f, 0.2f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
554     0.2f, 0.0f, 0.2f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f,
555     0.2f, 0.02f, 0.2f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
556     -0.2f, 0.02f, 0.2f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,
557
558     // Lado trasero
559     -0.2f, 0.0f, -0.2f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
560     0.2f, 0.0f, -0.2f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f,
561     0.2f, 0.02f, -0.2f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
562     -0.2f, 0.02f, -0.2f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
563
564     // Lado derecho
565     0.2f, 0.0f, -0.2f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
566     0.2f, 0.0f, 0.2f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
567     0.2f, 0.02f, 0.2f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
568     0.2f, 0.02f, -0.2f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
569
570     // Lado izquierdo
571     -0.2f, 0.0f, -0.2f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
572     -0.2f, 0.0f, 0.2f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
573     -0.2f, 0.02f, 0.2f, -1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
574     -0.2f, 0.02f, -0.2f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
575
576     };
577     std::vector<GLuint> indices = {
578         0, 1, 2, 2, 3, 0,
579         4, 6, 5, 4, 7, 6,
580         8, 9, 10, 10, 11, 8,
581         12, 14, 13, 12, 15, 14,
582         16, 17, 18, 18, 19, 16,
583         20, 22, 21, 20, 23, 22
584     };
585
586     geo->indexCount = 36;
587
588     glGenVertexArrays(1, &geo->VAO);
589     glGenBuffers(1, &geo->VBO);
590     glGenBuffers(1, &geo->EBO);
591
592     glBindVertexArray(geo->VAO);
593
594     glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
595     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
596
597     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
598     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
599
600     glEnableVertexAttribArray(0);
601     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
602     glEnableVertexAttribArray(1);
603     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
604     glEnableVertexAttribArray(2);
605     glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
606
607     glBindVertexArray(0);
608
609     return geo;
610 }
611
612 // Función para crear Tabla de Pícar
613 SimpleGeometry* CreateTablaPicar(GLuint texture) {
614     SimpleGeometry* geo = new SimpleGeometry();
615     geo->texture = texture;
616
617     std::vector<GLfloat> vertices = {
618         // Base inferior
619         -0.2f, 0.0f, -0.15f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
620         0.2f, 0.0f, -0.15f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
621         0.2f, 0.0f, 0.15f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
622         -0.2f, 0.0f, 0.15f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
623
624         // Base superior
625         -0.2f, 0.02f, -0.15f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
626         0.2f, 0.02f, -0.15f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
627         0.2f, 0.02f, 0.15f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
628         -0.2f, 0.02f, 0.15f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
629
630         // Lado frontal
631         -0.2f, 0.0f, 0.15f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
632         0.2f, 0.0f, 0.15f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f,
633         0.2f, 0.02f, 0.15f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
634         -0.2f, 0.02f, 0.15f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,
635
636         // Lado trasero
637         -0.2f, 0.0f, -0.15f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
638         0.2f, 0.0f, -0.15f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
639         0.2f, 0.02f, -0.15f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
640
641         // Lado derecho
642         0.2f, 0.0f, -0.15f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
643         0.2f, 0.0f, 0.15f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
644         0.2f, 0.02f, 0.15f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
645         0.2f, 0.02f, -0.15f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
646
647         // Lado izquierdo
648         -0.2f, 0.0f, -0.15f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
649         -0.2f, 0.0f, 0.15f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
650         -0.2f, 0.02f, 0.15f, -1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
651         -0.2f, 0.02f, -0.15f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
652
653     };
654
655     std::vector<GLuint> indices = {
656         0, 1, 2, 2, 3, 0,
657         4, 6, 5, 4, 7, 6,
658         8, 9, 10, 10, 11, 8,
659         12, 14, 13, 12, 15, 14,
660         16, 17, 18, 18, 19, 16,
661         20, 22, 21, 20, 23, 22
662     };
663
664     geo->indexCount = 36;
665
666     glGenVertexArrays(1, &geo->VAO);
667     glGenBuffers(1, &geo->VBO);
668     glGenBuffers(1, &geo->EBO);
669
670     glBindVertexArray(geo->VAO);
671
672     glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
673     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
674
675     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
676     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
677
678     glEnableVertexAttribArray(0);
679     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
680     glEnableVertexAttribArray(1);

```

```

681     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
682     glEnableVertexAttribArray(2);
683     glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
684
685     glBindVertexArray(0);
686
687     return geo;
688 }
689
690 // Función para crear hoja de cuchillo (rectángulo delgado alargado)
691 SimpleGeometry* CreateHojaCuchillo(GLuint texture) {
692     SimpleGeometry* geo = new SimpleGeometry();
693     geo->texture = texture;
694
695     std::vector<GLfloat> vertices = {
696         // Base inferior
697         -0.025f, 0.0f, -0.2f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
698         0.025f, 0.0f, -0.2f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
699         0.025f, 0.0f, 0.2f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
700         -0.025f, 0.0f, 0.2f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
701
702         // Base superior
703         -0.025f, 0.02f, -0.2f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
704         0.025f, 0.02f, -0.2f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
705         0.025f, 0.02f, 0.2f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
706         -0.025f, 0.02f, 0.2f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
707
708         // Lado frontal (Z+)
709         -0.025f, 0.0f, 0.2f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
710         0.025f, 0.0f, 0.2f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
711         0.025f, 0.02f, 0.2f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f,
712         -0.025f, 0.02f, 0.2f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,
713
714         // Lado trasero (Z-)
715         -0.025f, 0.0f, -0.2f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
716         0.025f, 0.0f, -0.2f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f,
717         0.025f, 0.02f, -0.2f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
718         -0.025f, 0.02f, -0.2f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
719
720         // Lado derecho (Y+)
721         0.025f, 0.0f, -0.2f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
722         0.025f, 0.0f, 0.2f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
723         0.025f, 0.02f, 0.2f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
724
725         // Lado izquierdo (X-)
726         -0.025f, 0.0f, -0.2f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
727         0.025f, 0.0f, -0.2f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
728         0.025f, 0.02f, -0.2f, -1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
729         -0.025f, 0.02f, -0.2f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
730     };
731
732     std::vector<GLuint> indices = {
733         0, 1, 2, 2, 3, 0,           // Base inferior
734         4, 6, 5, 4, 7, 6,           // Base superior
735         8, 9, 10, 10, 11, 8,           // Frontal
736         12, 14, 13, 12, 15, 14,           // Trasero
737         16, 17, 18, 18, 19, 16,           // Derecho
738         20, 22, 21, 20, 23, 22           // Izquierdo
739     };
740
741     geo->indexCount = 36;
742
743     glGenVertexArrays(1, &geo->VAO);
744     glGenBuffers(1, &geo->VBO);
745     glGenBuffers(1, &geo->EBO);
746
747     glBindVertexArray(geo->VAO);
748
749     glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
750     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
751
752     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
753     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
754
755     glEnableVertexAttribArray(0);
756     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
757     glEnableVertexAttribArray(1);
758     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
759     glEnableVertexAttribArray(2);
760     glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
761
762     glBindVertexArray(0);
763
764     return geo;
765 }
766
767 // Función para crear mango de cuchillo (cubo más pequeño)

```

```

768     SimpleGeometry* CreateMangoCuchillo(GLuint texture) {
769         SimpleGeometry* geo = new SimpleGeometry();
770         geo->texture = texture;
771
772         std::vector<GLfloat> vertices = {
773             // Base inferior
774             -0.025f, 0.0f, -0.05f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
775             0.025f, 0.0f, -0.05f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
776             0.025f, 0.0f, 0.05f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
777             -0.025f, 0.0f, 0.05f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
778
779             // Base superior
780             -0.025f, 0.02f, -0.05f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
781             0.025f, 0.02f, -0.05f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
782             0.025f, 0.02f, 0.05f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
783             -0.025f, 0.02f, 0.05f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
784
785             // Lado frontal (Z+)
786             -0.025f, 0.0f, 0.05f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
787             0.025f, 0.0f, 0.05f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f,
788             0.025f, 0.02f, 0.05f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
789             -0.025f, 0.02f, 0.05f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,
790
791             // Lado trasero (Z-)
792             -0.025f, 0.0f, -0.05f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
793             0.025f, 0.0f, -0.05f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f,
794             0.025f, 0.02f, -0.05f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
795             -0.025f, 0.02f, -0.05f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
796
797             // Lado derecho (X+)
798             0.025f, 0.0f, -0.05f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
799             0.025f, 0.0f, 0.05f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
800             0.025f, 0.02f, 0.05f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
801             0.025f, 0.02f, -0.05f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
802
803             // Lado izquierdo (X-)
804             -0.025f, 0.0f, -0.05f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
805             -0.025f, 0.0f, 0.05f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
806             -0.025f, 0.02f, 0.05f, -1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
807             -0.025f, 0.02f, -0.05f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
808         };
809
810         std::vector<GLuint> indices = {
811             0, 1, 2, 2, 3, 0,
812             4, 6, 5, 4, 7, 6,
813             8, 9, 10, 10, 11, 8,
814             12, 14, 13, 12, 15, 14,
815             16, 17, 18, 18, 19, 16,
816             20, 22, 21, 20, 23, 22
817         };
818
819         geo->indexCount = 36;
820
821         glGenVertexArrays(1, &geo->VAO);
822         glGenBuffers(1, &geo->VBO);
823         glGenBuffers(1, &geo->EBO);
824
825         glBindVertexArray(geo->VAO);
826
827         glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
828         glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
829
830         glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
831         glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
832
833         glEnableVertexAttribArray(0);
834         glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
835         glEnableVertexAttribArray(1);
836         glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
837         glEnableVertexAttribArray(2);
838         glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
839
840         glBindVertexArray(0);
841
842         return geo;
843     }
844
845     // Función para crear una gota de agua (cubo alargado con grosor medio)
846     SimpleGeometry* CreateGotaAgua(GLuint texture) {
847         SimpleGeometry* geo = new SimpleGeometry();
848         geo->texture = texture;
849
850         // Gotas con grosor medio (0.008) y largas (0.04)
851         std::vector<GLfloat> vertices = {
852             -0.008f, -0.04f, 0.008f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
853             0.008f, -0.04f, 0.008f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
854             0.008f, 0.04f, 0.008f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f,
855             -0.008f, 0.04f, 0.008f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,

```

```

854 |     -0.008f, 0.04f, 0.008f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,
855 |     -0.008f, -0.04f, -0.008f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
856 |     -0.008f, 0.04f, -0.008f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
857 |     0.008f, 0.04f, -0.008f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
858 |     0.008f, -0.04f, -0.008f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f,
859 |     0.008f, -0.04f, -0.008f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f,
860 |     0.008f, -0.04f, 0.008f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
861 |     -0.008f, 0.04f, -0.008f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
862 |     -0.008f, 0.04f, 0.008f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
863 |     0.008f, 0.04f, 0.008f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
864 |     0.008f, 0.04f, -0.008f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
865 |     -0.008f, -0.04f, -0.008f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
866 |     0.008f, -0.04f, -0.008f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
867 |     0.008f, -0.04f, 0.008f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
868 |     0.008f, -0.04f, 0.008f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
869 |     -0.008f, -0.04f, 0.008f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
870 |     0.008f, -0.04f, -0.008f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
871 |     0.008f, 0.04f, -0.008f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
872 |     0.008f, 0.04f, 0.008f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
873 |     0.008f, -0.04f, 0.008f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
874 |     -0.008f, -0.04f, -0.008f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
875 |     0.008f, -0.04f, -0.008f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
876 |     -0.008f, -0.04f, 0.008f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
877 |     -0.008f, 0.04f, 0.008f, -1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
878 |     -0.008f, 0.04f, -0.008f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
879 |     0.008f, 0.04f, -0.008f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
880 |     };
881 |
882 |     std::vector<GLuint> indices = {
883 |     0, 1, 2, 2, 3, 0,
884 |     4, 5, 6, 6, 7, 4,
885 |     8, 9, 10, 10, 11, 8,
886 |     12, 13, 14, 14, 15, 12,
887 |     16, 17, 18, 18, 19, 16,
888 |     20, 21, 22, 22, 23, 20
889 |     };
890 |
891 |     geo->indexCount = 36;
892 |
893 |     glGenVertexArrays(1, &geo->VAO);
894 |     glGenBuffers(1, &geo->VBO);
895 |     glGenBuffers(1, &geo->EBO);
896 |
897 |     glBindVertexArray(geo->VAO);
898 |     glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
899 |     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
900 |
901 |     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
902 |     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
903 |
904 |     glEnableVertexAttribArray(0);
905 |     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
906 |     glEnableVertexAttribArray(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
907 |     glEnableVertexAttribArray(2, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
908 |
909 |     glBindVertexArray(0);
910 |
911 |     return geo;
912 |
913 | }
914 |
915 | // Crear esfera pequeña para partículas de fuegos artificiales
916 | SimpleGeometry* CreateParticulaFuego(GLuint texture) {
917 |     SimpleGeometry* geo = new SimpleGeometry();
918 |     geo->texture = texture;
919 |
920 |     std::vector<GLfloat> vertices = {
921 |         // Cara frontal
922 |         -0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
923 |         0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
924 |         0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
925 |         -0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
926 |         -0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
927 |
928 |         // Cara trasera
929 |         -0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f, 0.0f,
930 |         0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f, 0.0f,
931 |         0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
932 |         0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
933 |
934 |         // Cara superior
935 |         -0.5f, 0.5f, -0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
936 |         -0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
937 |         0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
938 |         0.5f, 0.5f, -0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
939 |         };
940 |
941 |         // Cara inferior
942 |         -0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
943 |         0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
944 |         0.5f, -0.5f, 0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
945 |         -0.5f, -0.5f, 0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
946 |
947 |         // Cara derecha
948 |         0.5f, -0.5f, -0.5f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
949 |         0.5f, 0.5f, -0.5f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
950 |         0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f,
951 |         0.5f, -0.5f, 0.5f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f,
952 |
953 |         // Cara izquierda
954 |         -0.5f, -0.5f, -0.5f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f,
955 |         -0.5f, -0.5f, 0.5f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
956 |         -0.5f, 0.5f, 0.5f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
957 |         -0.5f, 0.5f, -0.5f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f
958 |     };
959 |
960 |     std::vector<GLuint> indices = {
961 |     0, 1, 2, 2, 3, 0,
962 |     4, 5, 6, 6, 7, 4,
963 |     8, 9, 10, 10, 11, 8,
964 |     12, 13, 14, 14, 15, 12,
965 |     16, 17, 18, 18, 19, 16,
966 |     20, 21, 22, 22, 23, 20
967 |     };
968 |
969 |     geo->indexCount = 36;
970 |
971 |     glGenVertexArrays(1, &geo->VAO);
972 |     glGenBuffers(1, &geo->VBO);
973 |     glGenBuffers(1, &geo->EBO);
974 |
975 |     glBindVertexArray(geo->VAO);
976 |     glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
977 |     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
978 |
979 |     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
980 |     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
981 |
982 |     glEnableVertexAttribArray(0);

```

```

983     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
984     glEnableVertexAttribArray(1);
985     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
986     glEnableVertexAttribArray(2);
987     glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
988
989     glBindVertexArray(0);
990
991     return geo;
992 }
993
994     void CreateExplosion(glm::vec3 posicion, glm::vec3 color) {
995         int partículasPorExplosión = 250; // MUCHAS MÁS PARTÍCULAS
996         int creadas = 0;
997
998         for (int i = 0; i < MAX_PARTICULAS && creadas < partículasPorExplosión; i++) {
999             if (!partículas[i].activa) {
1000                 partículas[i].activa = true;
1001                 partículas[i].posición = posicion;
1002                 partículas[i].vida = 3.0f + (rand() % 150) / 50.0f; // Viven MÁS tiempo
1003                 partículas[i].vidaMinima = partículas[i].vida;
1004                 partículas[i].color = color;
1005                 partículas[i].tamaño = 0.35f + (rand() % 150) / 500.0f; // MUY GRANDES (0.35-0.65)
1006
1007                 // Velocidades MUY RÁPIDAS para explosiones grandes
1008                 float theta = (rand() % 360) * PI / 180.0f;
1009                 float phi = (rand() % 180) * PI / 180.0f;
1010                 float velocidad = 5.0f + (rand() % 600) / 100.0f; // 5.0 a 11.0 (muy rápido)
1011
1012                 partículas[i].velocidad = glm::vec3(
1013                     velocidad * sin(phi) * cos(theta),
1014                     velocidad * sin(phi),
1015                     velocidad * sin(phi) * sin(theta)
1016                 );
1017
1018                 creadas++;
1019             }
1020         }
1021     }
1022
1023     int main()
1024     {
1025         if (!glfwInit())
1026         {
1027             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.

```

1026         return EXIT_FAILURE;
1027     }
1028
1029     glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
1030     glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
1031     glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
1032     glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
1033     glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
1034
1035     GLFWwindow* window = glfwCreateWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Modelado Jerarquico - Cocina 3D", nullptr, nullptr);
1036
1037     if (!window)
1038     {
1039         std::cout << "Error al crear ventana GLFW" << std::endl;
1040         glfwTerminate();
1041         return EXIT_FAILURE;
1042     }
1043
1044     glfwMakeContextCurrent(window);
1045     glfwSetKeyCallback(window, KeyCallback);
1046     glfwSetCursorPosCallback(window, MouseCallback);
1047     glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);
1048
1049     glewExperimental = GL_TRUE;
1050     if (glewInit() != GLEW_OK)
1051     {
1052         std::cout << "Error al inicializar GLEW" << std::endl;
1053         glfwTerminate();
1054         return EXIT_FAILURE;
1055     }
1056
1057     int screenWidth = 0, screenHeight = 0;
1058     glfwGetFramebufferSize(window, &screenWidth, &screenHeight);
1059     glViewport(0, 0, screenWidth, screenHeight);
1060     glEnable(GL_DEPTH_TEST);
1061     glClearColor(0.05f, 0.05f, 0.15f, 1.0f);
1062
1063     Shader lightingShader("Shader/Lighting.vs", "Shader/Lighting.frag");
1064
1065     std::cout << "\n== CONTROLES ==\n";
1066     std::cout << "#/A/S/D: Mover cámara" << std::endl;
1067     std::cout << "Mouse: Mirar alrededor" << std::endl;
1068     std::cout << "O/C: Abrir/cerrar puerta del refrigerador" << std::endl;

```

```

1069     std::cout << "N: Mover banderines hacia atrás" << std::endl;
1070     std::cout << "M: Regresar banderines a posición normal" << std::endl;
1071     std::cout << "G: Abrir/Cerrar grifo (agua)" << std::endl;
1072     std::cout << "F: Activar/Desactivar fuegos artificiales" << std::endl;
1073     std::cout << "1/2/3: Encender/Apagar lámparas de cocina" << std::endl;
1074     std::cout << "4: Encender/Apagar luz de la Luna" << std::endl;
1075     std::cout << "ESC: Salir\n" << std::endl;
1076
1077 // Carga de texturas
1078     GLuint textureArroz = LoadTexture("images/5138.jpg");
1079     GLuint textureBarandal = LoadTexture("images/fondo-abstracto-con-textura (1).jpg");
1080     GLuint textureBote = LoadTexture("images/0E5TN0.jpg");
1081     GLuint textureBolsaharinaAbierta1 = LoadTexture("images/Illustration03.jpg");
1082     GLuint textureBolsaharinaAbierta = LoadTexture("images/Illustration03.jpg");
1083     GLuint textureBolsasHarina = LoadTexture("images/Illustration03.jpg");
1084     GLuint textureBoteBasura = LoadTexture("images/fondo-abstracto-con-textura.jpg");
1085     GLuint textureCadenalIntern = LoadTexture("images/5178.jpg");
1086     GLuint textureCaja = LoadTexture("images/16.jpg");
1087     GLuint textureCaja2 = LoadTexture("images/370.jpg");
1088     GLuint textureCajasMadera = LoadTexture("images/5178.jpg");
1089     GLuint textureCampana = LoadTexture("images/5178.jpg");
1090     GLuint textureCortina2 = LoadTexture("images/preview.jpg");
1091     GLuint textureCortina = LoadTexture("images/169180-OVUVSU-320.jpg");
1092     GLuint textureCortinas = LoadTexture("images/preview.jpg");
1093     GLuint textureCubetas = LoadTexture("images/0R6ILJ0.jpg");
1094     GLuint textureElectricidad = LoadTexture("images/5178.jpg");
1095     GLuint textureEscaleras = LoadTexture("images/3690.jpg");
1096     GLuint textureEstufa = LoadTexture("images/textura-de-placa-de-metal.jpg");
1097     GLuint textureFregadero = LoadTexture("images/fondo-abstracto-con-textura.jpg");
1098     GLuint textureGas = LoadTexture("images/Q0V5UF0.jpg");
1099     GLuint textureJalader = LoadTexture("images/0R6ILJ0.jpg");
1100     GLuint textureLamparasCocina = LoadTexture("images/fondo-abstracto-con-textura.jpg");
1101     GLuint textureLamparasCocinal = LoadTexture("images/2259.jpg");
1102     GLuint textureLamparas1 = LoadTexture("images/2259.jpg");
1103     GLuint textureLamparas = LoadTexture("images/textura-de-placa-de-metal.jpg");
1104     GLuint textureLibrene = LoadTexture("images/3690.jpg");
1105     GLuint textureLuna = LoadTexture("images/2259.jpg");
1106     GLuint textureInternalLocal = LoadTexture("images/2259.jpg");
1107     GLuint textureMesa = LoadTexture("images/QV54F0.jpg");
1108     GLuint textureOlla = LoadTexture("images/5178.jpg");
1109     GLuint textureOlla1 = LoadTexture("images/5178.jpg");
1110     GLuint textureOlla2 = LoadTexture("images/acuarela-de-oro-liquido-espacio-de-copia.jpg");
1111     GLuint textureOlla3 = LoadTexture("images/acuarela-de-oro-liquido-espacio-de-copia.jpg");
1112
1113     GLuint texturePasto = LoadTexture("images/preview1.jpg");
1114     GLuint texturePared3 = LoadTexture("images/3690.jpg");
1115     GLuint texturePared4 = LoadTexture("images/3690.jpg");
1116     GLuint texturePlatos = LoadTexture("images/18366.jpg");
1117     GLuint texturePiso2 = LoadTexture("images/8469.jpg");
1118     GLuint texturePrimerPiso = LoadTexture("images/liquid-marbling-paint-texture-background-fluid");
1119     GLuint texturePuertas = LoadTexture("images/3690.jpg");
1120     GLuint textureRefrigerador = LoadTexture("images/0IV360.jpg");
1121     GLuint textureRefrigidor1 = LoadTexture("images/0IV360.jpg");
1122     GLuint textureSilla2 = LoadTexture("images/fondo-abstracto-con-textura.jpg");
1123     GLuint textureSilla = LoadTexture("images/textura-prung-roja.jpg");
1124     GLuint textureTapa = LoadTexture("images/0R6ILJ0.jpg");
1125     GLuint textureTapa1 = LoadTexture("images/0R6ILJ0.jpg");
1126     GLuint textureTapa2 = LoadTexture("images/0R6ILJ0.jpg");
1127     GLuint textureTangueAqua = LoadTexture("images/fondo-abstracto-con-textura.jpg");
1128     GLuint textureTangueAlmacen = LoadTexture("images/U7074.jpg");
1129     GLuint textureTapeteCocina = LoadTexture("images/abstract-blur-empty-green-gradient-studio-w");
1130     GLuint textureTazon = LoadTexture("images/0IV360.jpg");
1131     GLuint textureTazones2 = LoadTexture("images/0IV360.jpg");
1132     GLuint textureTejado = LoadTexture("images/6592778.jpg");
1133     GLuint textureTejadoMini = LoadTexture("images/3690.jpg");
1134     GLuint textureTejadoMinPiso2 = LoadTexture("images/3690.jpg");
1135     GLuint textureTuboFuera = LoadTexture("images/5178.jpg");
1136     GLuint textureTuboGas = LoadTexture("images/5178.jpg");
1137     GLuint textureVentanaSegundoPiso = LoadTexture("images/3690.jpg");
1138     GLuint textureBanderines = LoadTexture("images/preview.jpg");
1139
1140 // Carga de modelos
1141     Model Arroz((char*)"Models/Arroz.obj");
1142     Model Barandal((char*)"Models/Baranda.obj");
1143     Model Base((char*)"Models/Base.obj");
1144     Model Botes((char*)"Models/Botes.obj");
1145     Model BolsaharinaAbierta1((char*)"Models/BolsaharinaAbierta1.obj");
1146     Model BolsaharinaAbierta((char*)"Models/BolsaharinaAbierta.obj");
1147     Model Bolsaharina((char*)"Models/BolsasHarina.obj");
1148     Model BoteBasura((char*)"Models/BoteBasura.obj");
1149     Model CadenalIntern((char*)"Models/Cadenal.interna.obj");
1150     Model Caja((char*)"Models/Caja.obj");
1151     Model Caja2((char*)"Models/Caja2.obj");
1152     Model CajasMadera((char*)"Models/CajasMadera.obj");
1153     Model Campana((char*)"Models/Campana.obj");
1154     Model Cortina2((char*)"Models/Cortina2.obj");

```

```

1155 Model Cortina((char*)"Models/Cortina.obj");
1156 Model Cortina((char*)"Models/Cortinas.obj");
1157 Model Cubeta((char*)"Models/Cubeta.obj");
1158 Model Electricidad((char*)"Models/Electricidad.obj");
1159 Model Escaleras((char*)"Models/Escaleras.obj");
1160 Model Estufa((char*)"Models/Estufa.obj");
1161 Model Fregadero((char*)"Models/Fregadero.obj");
1162 Model Gas((char*)"Models/Gas.obj");
1163 Model Jalador((char*)"Models/Jalador.obj");
1164 Model LamparasCocina((char*)"Models/LamparasCocina.obj");
1165 Model LamparasCocina1((char*)"Models/LamparasCocina1.obj");
1166 Model Libro((char*)"Models/libro.obj");
1167 Model Luna((char*)"Models/Luna.obj");
1168 Model Linternalocal((char*)"Models/Linternalocal.obj");
1169 Model Mesa((char*)"Models/Mesa.obj");
1170 Model Olla((char*)"Models/Olla.obj");
1171 Model Olla1((char*)"Models/Olla1.obj");
1172 Model Olla2((char*)"Models/Olla2.obj");
1173 Model Olla3((char*)"Models/Olla3.obj");
1174 Model Pasto((char*)"Models/Pasto.obj");
1175 Model Pared3((char*)"Models/Pared3.obj");
1176 Model Pared4((char*)"Models/Pared4.obj");
1177 Model Pared((char*)"Models/Pared.obj");
1178 Model Platos((char*)"Models/Platos.obj");
1179 Model Piso2((char*)"Models/Piso2.obj");
1180 Model PrimerPiso((char*)"Models/PrimerPiso.obj");
1181 Model Puertas((char*)"Models/Puertas.obj");
1182 Model Refrigerador((char*)"Models/Refrigerador.obj");
1183 Model Refrigerador1((char*)"Models/Refrigerador1.obj");
1184 Model Silla2((char*)"Models/Silla2.obj");
1185 Model Silla((char*)"Models/Silla.obj");
1186 Model Tapia((char*)"Models/Tapia.obj");
1187 Model Tapia1((char*)"Models/Tapia1.obj");
1188 Model Tapia2((char*)"Models/Tapia2.obj");
1189 Model TanqueAgua((char*)"Models/TanqueAgua.obj");
1190 Model TapeteAlmacen((char*)"Models/TapeteAlmacen.obj");
1191 Model TapeteCocina((char*)"Models/TapeteCocina.obj");
1192 Model Tazon((char*)"Models/Tazon.obj");
1193 Model Tazones2((char*)"Models/Tazones2.obj");
1194 Model Tejado((char*)"Models/Tejado.obj");
1195 Model TejadoMin1((char*)"Models/TejadoMin1.obj");
1196 Model TejadoMin1Piso2((char*)"Models/TejadoMin1Piso2.obj");
1197 Model TuboFuera((char*)"Models/TuboFuera.obj");
1198 Model TuboGas((char*)"Models/TuboGas.obj");
1199 Model VentanaSegundoPiso((char*)"Models/VentanaSegundoPiso.obj");
1200 Model Bandineres((char*)"Models/Banderines.obj");
1201
1202 glm::mat4 projection = glm::perspective(glm::radians(45.0f), (GLfloat)screenWidth / (GLfloat)screenHeight, 0.1f, 100.0f);
1203
1204 // Crear geometrias personalizadas
1205 GLuint textureReloj = LoadTexture("images/OIV360.jpg");
1206 GLuint textureMesa = LoadTexture("images/fondo-abstrato-con-textura.jpg");
1207 GLuint texturePiso = LoadTexture("images/fondo-de-piso.jpg");
1208 GLuint textureMesacocina = LoadTexture("images/3696.jpg");
1209 GLuint textureNaranjito = LoadTexture("images/OIV369.jpg");
1210 GLuint textureSarten = LoadTexture("images/fondo-abstrato-con-textura.jpg");
1211 GLuint textureTabla = LoadTexture("images/fondo-de-diseño-de-textura-de-madera-de-roble.jpg");
1212 GLuint textureChillito = LoadTexture("images/fondo-abstrato-con-textura.jpg");
1213 GLuint textureCuchillo = LoadTexture("images/3698.jpg");
1214 GLuint textureAgua = LoadTexture("images/agua/piel-agua-azul-monocromático-minimalista.jpg");
1215 GLuint textureParticula = LoadTexture("images/azucarera-de-oro-líquido-espacio-de-copia.jpg"); // Textura brillante
1216
1217 SimpleGeometry* ParticulaFuego = CreateParticulaFuego(textureFuego);
1218 SimpleGeometry* RelojCirculo = CreateCircle(textureReloj, 0.5f, 60);
1219 SimpleGeometry* Naranjito = CreateCubo(textureNaranjito, 0.05f, 0.3f);
1220 SimpleGeometry* NaranjitoMunto = CreateClockwise(textureNaranjito, 0.05f, 0.45f);
1221 SimpleGeometry* Cubo = CreateCubo(textureCubo);
1222 SimpleGeometry* MesaTop = CreateMesTop(textureMesacocina);
1223 SimpleGeometry* Pata = CreatePataMes(textureMesacocina);
1224 SimpleGeometry* Narval = CreateCone(textureNarval);
1225 SimpleGeometry* GotaAgua = CreateSphere(textureAgua);
1226 SimpleGeometry* TablaPiso = CreateTablePiso(texturePiso);
1227 SimpleGeometry* HojaCuchillo = CreateHojaCuchillo(textureCuchillo);
1228 SimpleGeometry* MangoCuchillo = CreateMangoCuchillo(textureCuchilloMango);
1229 SimpleGeometry* GotaAgua = CreateGotaAgua(textureAgua);
1230
1231 // Inicializar el vector de gotas
1232 gotas.resize(MAX_GOTAS);
1233 for (int i = 0; i < MAX_GOTAS; i++) {
1234     gotas[i].activa = false;
1235     gotas[i].velocidad = 2.0f;
1236 }
1237
1238 // Inicializar las partículas de fuegos artificiales
1239 particulas.resize(MAX_PARTICULAS);
1240 for (int i = 0; i < MAX_PARTICULAS; i++) {
1241     particulas[i].activa = false;
1242 }

```

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.

```

1241     }
1242     srand(time(NULL)); // Para números aleatorios
1243     glm::vec3 posReloj = glm::vec3(-2.0f, 2.0f, 0.15f);
1244     glm::vec3 posCaja = glm::vec3(1.5f, 4.0f, -0.3f);
1245     glm::vec3 posMesa = glm::vec3(0.8f, 5.0f, -2.4f);
1246     glm::vec3 posTabla = glm::vec3(-1.5f, 1.1f, 1.5f);
1247     glm::vec3 posSarten = glm::vec3(0.8f, 5.0f, -2.4f);
1248
1249     while (!glfwWindowShouldClose(window))
1250     {
1251         currentTime = (GLfloat)glfwGetTime();
1252         deltaTime = currentTime - lastFrame;
1253         lastFrame = currentTime;
1254
1255         glfwPollEvents();
1256         ProcessMovement();
1257
1258         glClearColor(0.05f, 0.05f, 0.15f, 1.0f);
1259         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
1260
1261         lightingShader.Use();
1262
1263         GLint viewPosLoc = glGetUniformLocation(lightingShader.Program, "viewPos");
1264         glm::vec3 cameraPos = camera.GetPosition();
1265         glUniform3f(viewPosLoc, cameraPos.x, cameraPos.y, cameraPos.z);
1266
1267         glUniform3f(glGetUniformLocation(lightingShader.Program, "dirlight.direction"), -0.2f, -1.0f, -0.3f);
1268         glUniform3f(glGetUniformLocation(lightingShader.Program, "dirlight.ambient"), 0.05f, 0.05f, 0.1f);
1269         glUniform3f(glGetUniformLocation(lightingShader.Program, "dirlight.diffuse"), 0.1f, 0.1f, 0.15f);
1270         glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.2f, 0.2f, 0.3f);
1271
1272         for (int i = 0; i < 4; ++i)
1273         {
1274             std::string prefix = "pointLights[" + std::to_string(i) + "]";
1275
1276             glm::vec3 ambient, diffuse, specular;
1277             float constant, linear, quadratic;
1278
1279             if (i == 3)
1280                 ambient = lightStates[i] ? glm::vec3(0.2f, 0.2f, 0.3f) : glm::vec3(0.0f);
1281                 diffuse = lightStates[i] ? glm::vec3(0.8f, 0.8f, 1.2f) : glm::vec3(0.0f);
1282                 specular = lightStates[i] ? glm::vec3(1.0f, 1.0f, 1.3f) : glm::vec3(0.0f);
1283                 constant = 1.0f;
1284                 linear = 0.014f;
1285
1286                 quadratic = 0.0007f;
1287
1288             else
1289                 ambient = lightStates[i] ? glm::vec3(0.15f, 0.05f, 0.0f) : glm::vec3(0.0f); // tenue
1290                 diffuse = lightStates[i] ? glm::vec3(1.0f, 0.4f, 0.1f) : glm::vec3(0.0f); // naranja-rojizo "chakra"
1291                 specular = lightStates[i] ? glm::vec3(1.0f, 0.6f, 0.2f) : glm::vec3(0.0f); // brillo cálido
1292
1293                 // Atenuación suave para que no se expanda demasiado
1294                 constant = 1.0f;
1295                 linear = 0.07f;
1296                 quadratic = 0.017f;
1297
1298             glUniform3f(glGetUniformLocation(lightingShader.Program, (prefix + ".position").c_str()),
1299                         pointLightPositions[i].x, pointLightPositions[i].y, pointLightPositions[i].z);
1300             glUniform3f(glGetUniformLocation(lightingShader.Program, (prefix + ".ambient").c_str()),
1301                         ambient.x, ambient.y, ambient.z);
1302             glUniform3f(glGetUniformLocation(lightingShader.Program, (prefix + ".diffuse").c_str()),
1303                         diffuse.x, diffuse.y, diffuse.z);
1304             glUniform3f(glGetUniformLocation(lightingShader.Program, (prefix + ".specular").c_str()),
1305                         specular.x, specular.y, specular.z);
1306             glUniform1f(glGetUniformLocation(lightingShader.Program, (prefix + ".constant").c_str()), constant);
1307             glUniform1f(glGetUniformLocation(lightingShader.Program, (prefix + ".linear").c_str()), linear);
1308             glUniform1f(glGetUniformLocation(lightingShader.Program, (prefix + ".quadratic").c_str()), quadratic);
1309
1310             // Inicializar spotlight (apagado) - IMPORTANTE para que el shader funcione
1311             glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.position"), 0.0f, 0.0f, 0.0f);
1312             glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.direction"), 0.0f, -1.0f, 0.0f);
1313             glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.cutOff"), glm::cos(glm::radians(12.5f)));
1314             glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.outerCutOff"), glm::cos(glm::radians(15.0f)));
1315             glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.ambient"), 0.0f, 0.0f, 0.0f);
1316             glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.diffuse"), 0.0f, 0.0f, 0.0f);
1317             glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.specular"), 0.0f, 0.0f, 0.0f);
1318             glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.constant"), 1.0f);
1319             glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.linear"), 0.09f);
1320             glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.quadratic"), 0.032f);
1321
1322             glm::mat4 view = camera.GetViewMatrix();
1323             glm::mat4 model = glm::mat4(1);
1324
1325             GLint modelLoc = glGetUniformLocation(lightingShader.Program, "model");
1326
1327             glUniformMatrix4fv(glGetUniformLocation(lightingShader.Program, "projection"), 1, GL_FALSE, glm::value_ptr(projection));
1328             glUniformMatrix4fv(glGetUniformLocation(lightingShader.Program, "view"), 1, GL_FALSE, glm::value_ptr(view));
1329
1330             glUniform1i(glGetUniformLocation(lightingShader.Program, "transparency"), 0);
1331
1332             // IMPORTANTE: Configurar los samplers de textura
1333             glUniform1i(glGetUniformLocation(lightingShader.Program, "Material.diffuse"), 0);
1334             glUniform1i(glGetUniformLocation(lightingShader.Program, "Material.specular"), 0);
1335
1336             // ===== PRIMERO: FUEGOS ARTIFICIALES (para que estén detrás) =====
1337             if (fuegosActivos)
1338             {
1339                 glEnable(GL_BLEND);
1340                 glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
1341                 // NO desactive el depth test aquí para que respeten la profundidad
1342                 // glDisable(GL_DEPTH_TEST); <-- QUITA ESTA LÍNEA
1343
1344                 for (int i = 0; i < MAX_PARTICULAS; i++)
1345                 {
1346                     if (particulas[i].activa)
1347                     {
1348                         model = glm::mat4(1.0f);
1349                         model = glm::translate(model, particulas[i].posicion);
1350
1351                         float factorVida = particulas[i].vida / particulas[i].vidaInicial;
1352                         float escala = particulas[i].tamano * factorVida;
1353                         model = glm::scale(model, glm::vec3(escala));
1354
1355                         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1356                         glBindTexture(GL_TEXTURE_2D, textureParticula);
1357                         glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 128.0f);
1358
1359                         ParticulaFuego->Draw();
1360                     }
1361
1362                     // glEnable(GL_DEPTH_TEST); <-- QUITA ESTA LÍNEA TAMBÍEN
1363                     // glDisable(GL_BLEND);
1364
1365                     // Renderizado de objetos opacos
1366                     model = glm::mat4(1.0f);
1367                     glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1368
1369                     // Bind texture y dibujar Arroz
1370                     glActiveTexture(GL_TEXTURE0);
1371                     glBindTexture(GL_TEXTURE_2D, textureArroz);
1372
1373                 }
1374             }
1375
1376             // glDisable(GL_DEPTH_TEST); <-- QUITA ESTA LÍNEA TAMBÍEN
1377             // glEnable(GL_BLEND);
1378
1379             // Bind texture y dibujar Arroz
1380             glActiveTexture(GL_TEXTURE0);
1381             glBindTexture(GL_TEXTURE_2D, textureArroz);
1382
1383         }
1384
1385         // Bind texture y dibujar Arroz
1386         glActiveTexture(GL_TEXTURE0);
1387         glBindTexture(GL_TEXTURE_2D, textureArroz);
1388
1389     }

```

```

1370     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 48.0f);
1371     Arroz.Draw(lightingShader);
1372
1373     // Base
1374     glBindTexture(GL_TEXTURE_2D, textureBase);
1375     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 32.0f);
1376     Base.Draw(lightingShader);
1377
1378     // Barandal
1379     glBindTexture(GL_TEXTURE_2D, textureBarandal);
1380     Barandal.Draw(lightingShader);
1381
1382     // Botes
1383     glBindTexture(GL_TEXTURE_2D, textureBotes);
1384     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 24.0f);
1385     Botes.Draw(lightingShader);
1386
1387     // BolsaHarinaAbierta
1388     glBindTexture(GL_TEXTURE_2D, textureBolsaHarinaAbierta);
1389     BolsaHarinaAbierta.Draw(lightingShader);
1390
1391     // BolsaHarinaAbierta1
1392     glBindTexture(GL_TEXTURE_2D, textureBolsaHarinaAbierta1);
1393     BolsaHarinaAbierta1.Draw(lightingShader);
1394
1395     // BolsasHarina
1396     glBindTexture(GL_TEXTURE_2D, textureBolsasHarina);
1397     BolsasHarina.Draw(lightingShader);
1398
1399     // BoteBasura
1400     glBindTexture(GL_TEXTURE_2D, textureBoteBasura);
1401     BoteBasura.Draw(lightingShader);
1402
1403     // CadenaIntern
1404     glBindTexture(GL_TEXTURE_2D, textureCadenaIntern);
1405     CadenaIntern.Draw(lightingShader);
1406
1407     // Caja
1408     glBindTexture(GL_TEXTURE_2D, textureCaja);
1409     Caja.Draw(lightingShader);
1410
1411     // Cajas2
1412     glBindTexture(GL_TEXTURE_2D, textureCajas2);
1413
1414
1415     // CajasMadera
1416     glBindTexture(GL_TEXTURE_2D, textureCajasMadera);
1417     CajasMadera.Draw(lightingShader);
1418
1419     // Campana
1420     glBindTexture(GL_TEXTURE_2D, textureCampana);
1421     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 56.0f);
1422     Campana.Draw(lightingShader);
1423
1424     // Cortina
1425     glBindTexture(GL_TEXTURE_2D, textureCortina);
1426     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
1427     Cortina.Draw(lightingShader);
1428
1429     // Cortinas
1430     glBindTexture(GL_TEXTURE_2D, textureCortinas);
1431     Cortinas.Draw(lightingShader);
1432
1433     // Cortina2
1434     glBindTexture(GL_TEXTURE_2D, textureCortina2);
1435     Cortina2.Draw(lightingShader);
1436
1437     // Cubeta
1438     glBindTexture(GL_TEXTURE_2D, textureCubeta);
1439     Cubeta.Draw(lightingShader);
1440
1441     // Electricidad
1442     glBindTexture(GL_TEXTURE_2D, textureElectricidad);
1443     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 40.0f);
1444     Electricidad.Draw(lightingShader);
1445
1446     // Escaleras
1447     glBindTexture(GL_TEXTURE_2D, textureEscaleras);
1448     Escaleras.Draw(lightingShader);
1449
1450     // Estufa
1451     glBindTexture(GL_TEXTURE_2D, textureEstufa);
1452     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 64.0f);
1453     Estufa.Draw(lightingShader);
1454
1455
1456     // Fregadero
1457     glBindTexture(GL_TEXTURE_2D, textureFregadero);
1458     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 72.0f);
1459     Fregadero.Draw(lightingShader);
1460
1461     // Gas
1462     glBindTexture(GL_TEXTURE_2D, textureGas);
1463     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 48.0f);
1464     Gas.Draw(lightingShader);
1465
1466     // Jalador
1467     glBindTexture(GL_TEXTURE_2D, textureJalador);
1468     Jalador.Draw(lightingShader);
1469
1470     // LamparasCocina
1471     glBindTexture(GL_TEXTURE_2D, textureLamparasCocina);
1472     LamparasCocina.Draw(lightingShader);
1473
1474     // Librero
1475     glBindTexture(GL_TEXTURE_2D, textureLibrero);
1476     Librero.Draw(lightingShader);
1477
1478     // Mesa
1479     glBindTexture(GL_TEXTURE_2D, textureMesa);
1480     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 40.0f);
1481     Mesa.Draw(lightingShader);
1482
1483     // Olla
1484     glBindTexture(GL_TEXTURE_2D, textureOlla);
1485     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 96.0f);
1486     Olla.Draw(lightingShader);
1487
1488     // Olla1
1489     glBindTexture(GL_TEXTURE_2D, textureOlla1);
1490     Olla1.Draw(lightingShader);
1491
1492     // Olla2
1493     glBindTexture(GL_TEXTURE_2D, textureOlla2);
1494     Olla2.Draw(lightingShader);
1495
1496     // Olla3
1497     glBindTexture(GL_TEXTURE_2D, textureOlla3);

```

```

1498     Olla3.Draw(lightingShader);
1499
1500     // Pasto
1501     glBindTexture(GL_TEXTURE_2D, texturePasto);
1502     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 8.0f);
1503     Pasto.Draw(lightingShader);
1504
1505     // Pared3
1506     glBindTexture(GL_TEXTURE_2D, texturePared3);
1507     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 24.0f);
1508     Pared3.Draw(lightingShader);
1509
1510     // Pared4
1511     glBindTexture(GL_TEXTURE_2D, texturePared4);
1512     Pared4.Draw(lightingShader);
1513
1514     // Pared
1515     glBindTexture(GL_TEXTURE_2D, texturePared);
1516     Pared.Draw(lightingShader);
1517
1518     // Platos
1519     glBindTexture(GL_TEXTURE_2D, texturePlatos);
1520     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 56.0f);
1521     Platos.Draw(lightingShader);
1522
1523     // PrimerPiso
1524     glBindTexture(GL_TEXTURE_2D, texturePrimerPiso);
1525     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 32.0f);
1526     PrimerPiso.Draw(lightingShader);
1527
1528     // Piso2
1529     glBindTexture(GL_TEXTURE_2D, texturePiso2);
1530     Piso2.Draw(lightingShader);
1531
1532     // Puertas
1533     glBindTexture(GL_TEXTURE_2D, texturePuertas);
1534     Puertas.Draw(lightingShader);
1535
1536     // Refrigerador
1537     glBindTexture(GL_TEXTURE_2D, textureRefrigerador);
1538     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 80.0f);
1539     Refrigerador.Draw(lightingShader);
1540
1541
1542     // Silla2
1543     glBindTexture(GL_TEXTURE_2D, textureSilla2);
1544     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 28.0f);
1545     Silla2.Draw(lightingShader);
1546
1547     // Silla
1548     glBindTexture(GL_TEXTURE_2D, textureSilla);
1549     Silla.Draw(lightingShader);
1550
1551     // Tapa
1552     glBindTexture(GL_TEXTURE_2D, textureTapa);
1553     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 48.0f);
1554     Tapa.Draw(lightingShader);
1555
1556     // Tapal
1557     glBindTexture(GL_TEXTURE_2D, textureTapal);
1558     Tapal.Draw(lightingShader);
1559
1560     // Tapa2
1561     glBindTexture(GL_TEXTURE_2D, textureTapa2);
1562     Tapa2.Draw(lightingShader);
1563
1564     // TapeteAlmacen
1565     glBindTexture(GL_TEXTURE_2D, textureTapeteAlmacen);
1566     TapeteAlmacen.Draw(lightingShader);
1567
1568     // TanqueAgua
1569     glBindTexture(GL_TEXTURE_2D, textureTanqueAgua);
1570     TanqueAgua.Draw(lightingShader);
1571
1572     // TapeteCocina
1573     glBindTexture(GL_TEXTURE_2D, textureTapeteCocina);
1574     TapeteCocina.Draw(lightingShader);
1575
1576     // Tazon
1577     glBindTexture(GL_TEXTURE_2D, textureTazon);
1578     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 64.0f);
1579     Tazon.Draw(lightingShader);
1580
1581     // Tazones2
1582     glBindTexture(GL_TEXTURE_2D, textureTazones2);
1583     Tazones2.Draw(lightingShader);
1584
1585     // Tejado
1586     glBindTexture(GL_TEXTURE_2D, textureTejado);
1587     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
1588     Tejado.Draw(lightingShader);
1589
1590     // TejadoMini
1591     glBindTexture(GL_TEXTURE_2D, textureTejadoMini);
1592     TejadoMini.Draw(lightingShader);
1593
1594     // TejadoMiniPiso
1595     glBindTexture(GL_TEXTURE_2D, textureTejadoMiniPiso2);
1596     TejadoMiniPiso2.Draw(lightingShader);
1597
1598     // TuboFuera
1599     glBindTexture(GL_TEXTURE_2D, textureTuboFuera);
1600     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 80.0f);
1601     TuboFuera.Draw(lightingShader);
1602
1603     // TuboGas
1604     glBindTexture(GL_TEXTURE_2D, textureTuboGas);
1605     TuboGas.Draw(lightingShader);
1606
1607     // VentanaSegundoPiso
1608     glBindTexture(GL_TEXTURE_2D, textureVentanaSegundoPiso);
1609     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 24.0f);
1610     VentanaSegundoPiso.Draw(lightingShader);
1611
1612     // ===== BANDERINES CON CONTROL MANUAL (ADELANTE/ATRÁS) =====
1613     model = glm::mat4(1.0f);
1614
1615     // La rotación se aplica en el eje X para mover la punta hacia adelante/atrás
1616     model = glm::rotate(model, glm::radians(banderinesInclinacion), glm::vec3(1.0f, 0.0f, 0.0f));
1617
1618     glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1619     glBindTexture(GL_TEXTURE_2D, textureBanderines);
1620     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
1621     Banderines.Draw(lightingShader);
1622
1623     // ===== RELOJ =====
1624     model = glm::mat4(1.0f);
1625     model = glm::translate(model, posReloj);
1626     glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1627     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 32.0f);

```

```

1627 |     RelojCirculo->Draw();
1628 |
1629 |     // Manecilla de hora (gira lento)
1630 |     float horaAngulo = (float)glfGetTime() * 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)glfGetTime() * 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(glfGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
1648 |     Caja->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(glfGetUniformLocation(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();
1670 |
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(glfGetUniformLocation(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(2.0f, 1.0f, 2.0f));
1694 |     glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1695 |     glUniform1f(glfGetUniformLocation(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(glfGetUniformLocation(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::rotate(model, glm::radians(45.0f), glm::vec3(0.0f, 1.0f, 0.0f));
1712 |     glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1713 |
1714 |     glUniform1f(glfGetUniformLocation(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(glfGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
1721 |     MangoCuchillo->Draw();
1722 |
1723 |     // ===== GOTAS DE AGUA =====
1724 |     if (agujaCayendo) {
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(glfGetUniformLocation(lightingShader.Program, "material.shininess"), 128.0f);
1735 |                 GotaAgua->Draw();
1736 |             }
1737 |         }
1738 |         glDisable(GL_BLEND);
1739 |     }
1740 |
1741 |     // Animación del refrigerador
1742 |     glm::vec3 posBisagraRefrigerador = glm::vec3(2.5f, 1.0f, -3.00f);
1743 |     model = glm::mat4(1.0f);
1744 |     model = glm::translate(model, posBisagraRefrigerador);
1745 |     model = glm::rotate(model, glm::radians(-puertaRefrigerador), glm::vec3(0.0f, 1.0f, 0.0f));
1746 |     model = glm::translate(model, -posBisagraRefrigerador);
1747 |     glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1748 |     glBindTexture(GL_TEXTURE_2D, textureRefrigerador);
1749 |     glUniform1f(glfGetUniformLocation(lightingShader.Program, "material.shininess"), 80.0f);
1750 |     Refrigerador.Draw(lightingShader);
1751 |
1752 |
1753 |     // Objetos transparentes (lámparas)
1754 |     glEnable(GL_BLEND);
1755 |     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.

```

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     // LamparasCocina
1764     glBindTexture(GL_TEXTURE_2D, textureLamparasCocina);
1765     LamparasCocina.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 {

```

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.

```

1799 // Movimiento de cámara
1800 if (keys[GLFW_KEY_W] || keys[GLFW_KEY_UP])
1801     camera_ProcessKeyboard(FORWARD, deltaTime);
1802 if (keys[GLFW_KEY_S] || keys[GLFW_KEY_DOWN])
1803     camera_ProcessKeyboard(BACKWARD, deltaTime);
1804 if (keys[GLFW_KEY_A] || keys[GLFW_KEY_LEFT])
1805     camera_ProcessKeyboard(LEFT, deltaTime);
1806 if (keys[GLFW_KEY_D] || keys[GLFW_KEY_RIGHT])
1807     camera_ProcessKeyboard(RIGHT, deltaTime);
1808
1809 // Animación automática de la puerta
1810 if (animandoPuerta) {
1811     if (abs(puertaRefrigerador - puertaDestino) > 0.1f) {
1812         if (puertaRefrigerador < puertaDestino)
1813             puertaRefrigerador += 80.0f * deltaTime;
1814         else
1815             puertaRefrigerador -= 80.0f * deltaTime;
1816     }
1817     else {
1818         puertaRefrigerador = puertaDestino;
1819         animandoPuerta = false;
1820     }
1821 }
1822
1823 // **ANIMACIÓN: Banderines moviéndose hacia adelante/atrás**
1824 if (animandoBanderines) {
1825     if (abs(banderinesInclinacion - banderinesDestino) > 0.1f) {
1826         if (banderinesInclinacion < banderinesDestino)
1827             banderinesInclinacion += 60.0f * deltaTime;
1828         else
1829             banderinesInclinacion -= 60.0f * deltaTime;
1830     }
1831     else {
1832         banderinesInclinacion = banderinesDestino;
1833         animandoBanderines = false;
1834     }
1835 }
1836
1837 // **ANIMACIÓN: Agua cayendo del grifo**
1838 if (aguaCayendo) {
1839     // Crear nuevas gotas
1840     if (currentFrame - tiempoUltimagota > intervaloGotas) {
1841         for (int i = 0; i < MAX_GOTAS; i++) {
1842             if (gotas[i].activa) {
1843                 gotas[i].activa = true;
1844                 gotas[i].posicion = posicionGrifo;
1845                 gotas[i].velocidad = 2.0f + (rand() % 100) / 100.0f;
1846                 tiempoUltimagota = currentFrame;
1847                 break;
1848             }
1849         }
1850     }
1851
1852     // Actualizar posición de gotas activas
1853     for (int i = 0; i < MAX_GOTAS; i++) {
1854         if (gotas[i].activa) {
1855             gotas[i].posicion.y -= gotas[i].velocidad * deltaTime;
1856             gotas[i].velocidad += 3.0f * deltaTime; // Gravedad
1857
1858             // Desactivar gota si llega al fregadero
1859             if (gotas[i].posicion.y < 1.0f) {
1860                 gotas[i].activa = false;
1861             }
1862         }
1863     }
1864 }
1865 // **ANIMACIÓN: Fuegos artificiales**
1866 if (fuegosActivos) {
1867     if (currentFrame - tiempoUltimoFuego >= intervaloFuegos) {
1868         // MÁS SEPARADAS HORIZONTALMENTE (de -8 a +8)
1869         glm::vec3 posExplosion = glm::vec3(
1870             -8.0f + (rand() % 160) / 100.0f, // -8 a 8 en X (MUY ANCHO)
1871             7.0f + (rand() % 200) / 100.0f, // 7 a 9 en Y (altura similar)
1872             -10.0f // Z FIJO (sin profundidad)
1873         );
1874
1875         // Colores vivos y variados
1876         int tipoColor = rand() % 5;
1877         glm::vec3 colorExplosion;
1878
1879         switch (tipoColor) {
1880             case 0: colorExplosion = glm::vec3(1.0f, 0.2f, 0.2f); break; // Rojo
1881             case 1: colorExplosion = glm::vec3(0.2f, 1.0f, 0.2f); break; // Verde
1882             case 2: colorExplosion = glm::vec3(0.2f, 0.2f, 1.0f); break; // Azul
1883             case 3: colorExplosion = glm::vec3(1.0f, 1.0f, 0.2f); break; // Amarillo
1884             case 4: colorExplosion = glm::vec3(1.0f, 0.2f, 1.0f); break; // Magenta
1885         }
1886
1887         CrearExplosion(posExplosion, colorExplosion);
1888         tiempoUltimoFuego = currentFrame;
1889
1890         std::cout << "BOOM! Explosión en el fondo Z=" << posExplosion.z << std::endl;
1891     }
1892
1893     // Actualizar partículas activas
1894     for (int i = 0; i < MAX_PARTICULAS; i++) {
1895         if (particulas[i].activa) {
1896             // Actualizar posición
1897             particulas[i].posicion += particulas[i].velocidad * deltaTime;
1898
1899             // Aplicar gravedad suave
1900             particulas[i].velocidad.y -= 2.5f * deltaTime;
1901
1902             // Fricción del aire
1903             particulas[i].velocidad *= 0.98f;
1904
1905             // Reducir vida
1906             particulas[i].vida -= deltaTime;
1907
1908             // Desactivar si ya no tiene vida
1909             if (particulas[i].vida <= 0.0f) {
1910                 particulas[i].activa = false;
1911             }
1912         }
1913     }
1914 }
1915
1916 void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode)
1917 {
1918     (void)scancode;
1919     (void)mode;
1920
1921     if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
1922         glfwSetWindowShouldClose(window, GL_TRUE);
1923
1924     if (key >= 0 && key < 1024)
1925     {
1926         if (action == GLFW_PRESS)
1927             keys[key] = true;
1928     }
1929 }

```

```

1928     else if (action == GLFW_RELEASE)
1929     {
1930         keys[key] = false;
1931     }
1932     if (action == GLFW_PRESS)
1933     {
1934         switch (key)
1935         {
1936             case GLFW_KEY_1:
1937                 lightStates[0] = !lightStates[0];
1938                 std::cout << "luz 1 (LamparasCocinal): " << (lightStates[0] ? "ENCENDIDA" : "APAGADA") << std::endl;
1939                 break;
1940             case GLFW_KEY_2:
1941                 lightStates[1] = !lightStates[1];
1942                 std::cout << "luz 2 (Lamparas): " << (lightStates[1] ? "ENCENDIDA" : "APAGADA") << std::endl;
1943                 break;
1944             case GLFW_KEY_3:
1945                 lightStates[2] = !lightStates[2];
1946                 std::cout << "luz 3 (LinternalLocal): " << (lightStates[2] ? "ENCENDIDA" : "APAGADA") << std::endl;
1947                 break;
1948             case GLFW_KEY_4:
1949                 lightStates[3] = !lightStates[3];
1950                 std::cout << "luz Luna: " << (lightStates[3] ? "ENCENDIDA" : "APAGADA") << std::endl;
1951                 break;
1952             case GLFW_KEY_O:
1953                 puertaDestino = 90.0f; // Abrir
1954                 animandoPuerta = true;
1955                 std::cout << "Abriendo puerta del refrigerador..." << std::endl;
1956                 break;
1957             case GLFW_KEY_C:
1958                 puertaDestino = 0.0f; // Cerrar
1959                 animandoPuerta = true;
1960                 std::cout << "Cerrando puerta del refrigerador..." << std::endl;
1961                 break;
1962             case GLFW_KEY_N: // Mover banderines hacia ATRÁS
1963                 banderinesDestino = -banderinesMaxInclinacion;
1964                 animandoBanderines = true;
1965                 std::cout << "Banderines moviéndose hacia atrás..." << std::endl;
1966                 break;
1967             case GLFW_KEY_M: // Regresar banderines a posición NORMAL
1968                 banderinesDestino = 0.0f;
1969                 animandoBanderines = true;
1970
1971             std::cout << "Banderines regresando a posición normal..." << std::endl;
1972             break;
1973         case GLFW_KEY_G: // Tapon agua del grifo
1974             aguaCayendo = !aguaCayendo;
1975             std::cout << "Agua del grifo: " << (aguaCayendo ? "ABIERTA" : "CERRADA") << std::endl;
1976             if (!aguaCayendo)
1977             {
1978                 for (int i = 0; i < MAX_GOTAS; i++)
1979                 {
1980                     gotas[i].activa = false;
1981                 }
1982             }
1983             break;
1984         case GLFW_KEY_F: // Toggle Fuegos artificiales
1985             fuegosActivos = !fuegosActivos;
1986             std::cout << "Fuegos artificiales: " << (fuegosActivos ? "ACTIVADOS" : "DESACTIVADOS") << std::endl;
1987             if (!fuegosActivos)
1988             {
1989                 for (int i = 0; i < MAX_PARTICULAS; i++)
1990                 {
1991                     particulas[i].activa = false;
1992                 }
1993             }
1994         }
1995     }
1996     void MouseCallback(GLFWwindow* window, double xpos, double ypos)
1997     {
1998         (void)window;
1999
2000         if (firstMouse)
2001         {
2002             lastX = (GLfloat)xpos;
2003             lastY = (GLfloat)ypos;
2004             firstMouse = false;
2005         }
2006
2007         GLfloat xOffset = (GLfloat)(xpos - lastX);
2008         GLfloat yOffset = (GLfloat)(lastY - ypos);
2009
2010         lastX = (GLfloat)xpos;
2011         lastY = (GLfloat)ypos;
2012
2013         camera.ProcessMouseMovement(xOffset, yOffset);

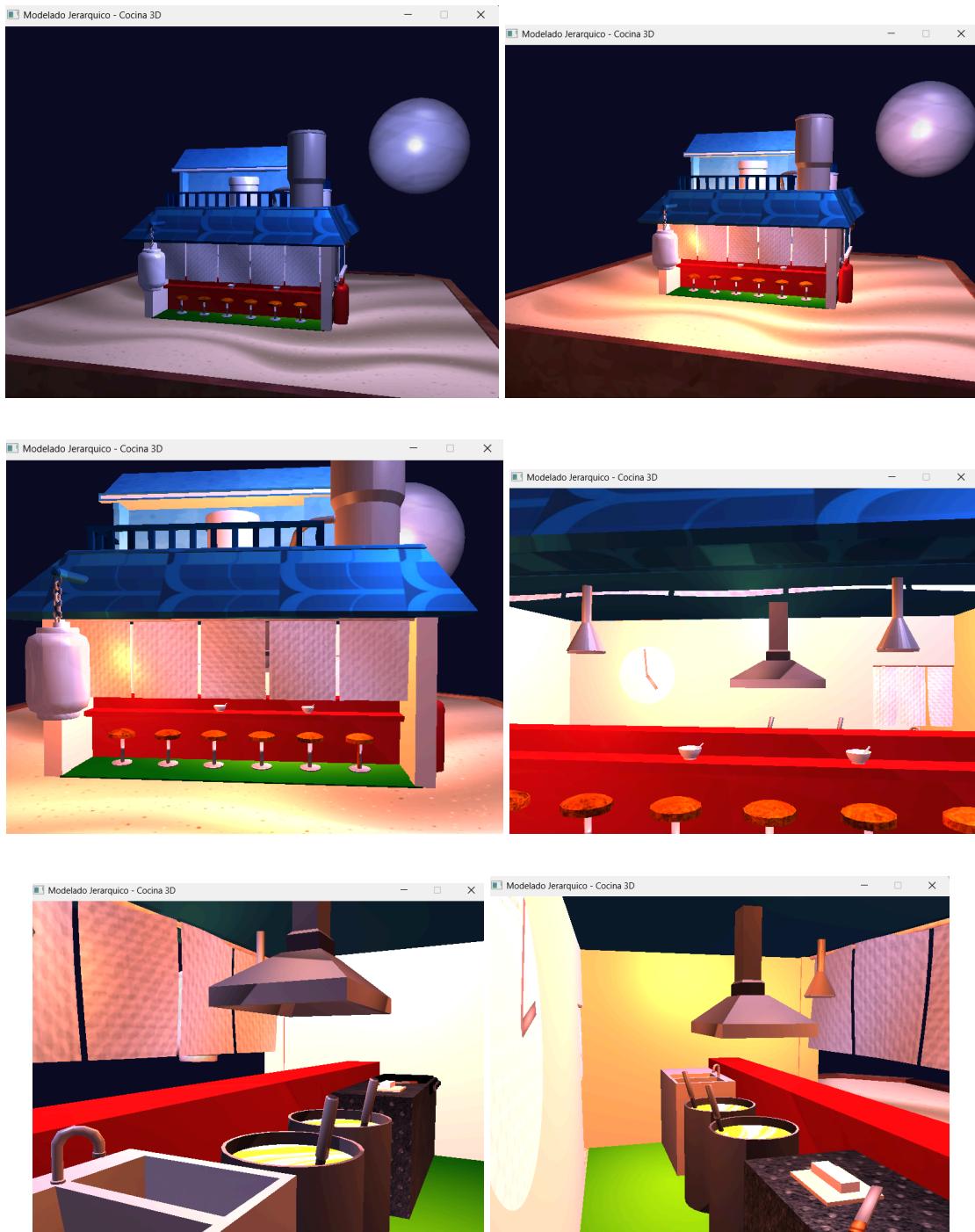
```

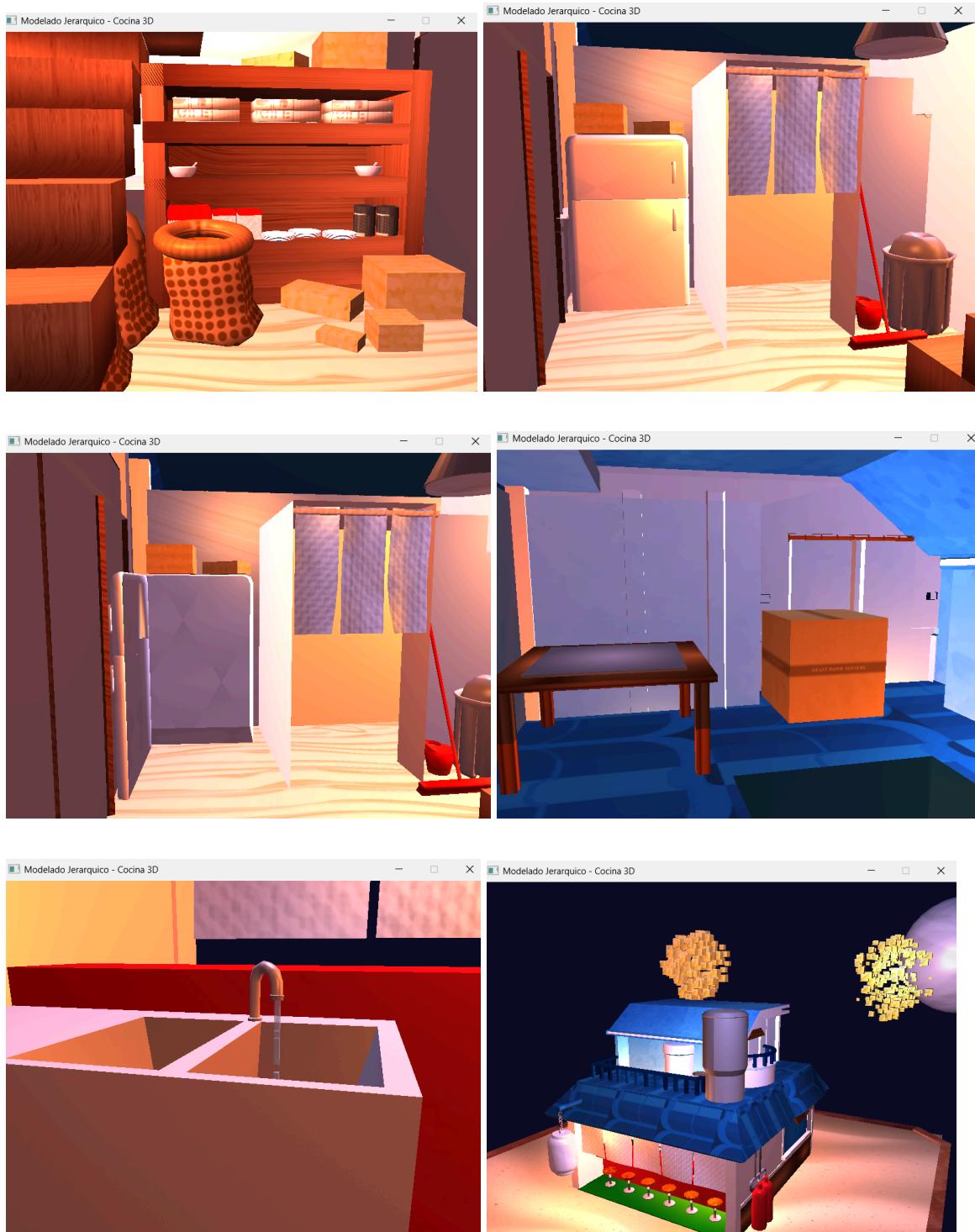
Resultados

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.





- **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

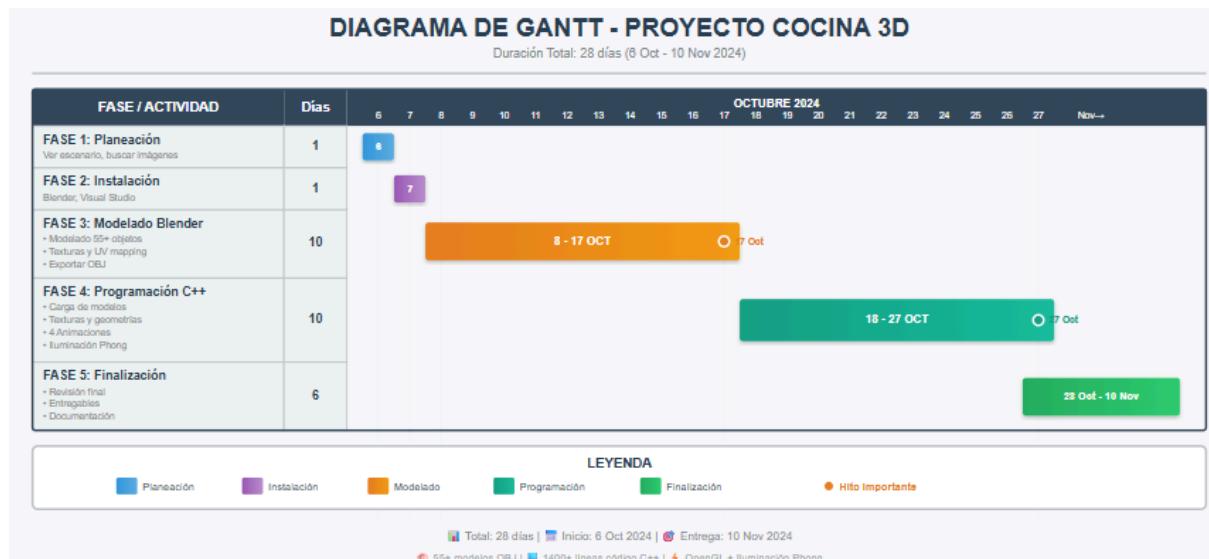
Project schedule

Phase	Activities	Duration	Deadline
Phase 1: Project Planning	<ul style="list-style-type: none"> • See the scenario it would be. • Find images of the scenario. 	1 día	6 de Octubre.
Phase 2: Installation of Blender and Visual Studio.	<ul style="list-style-type: none"> • Application installation. • Initial configurations of Blender and Visual Studio. 	1 día	7 Octubre.
Phase 3: Modeling the objects in Blender.	<ul style="list-style-type: none"> • 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.	<ul style="list-style-type: none"> • Essential code for loading models. • Creating models using coordinates. • Loading textures. • Creating animations. • Adding lighting. 	10 días.	27 de Octubre.
Phase 5: Completion.	<ul style="list-style-type: none"> • 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, glfw, 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 1 Objects Theory:

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

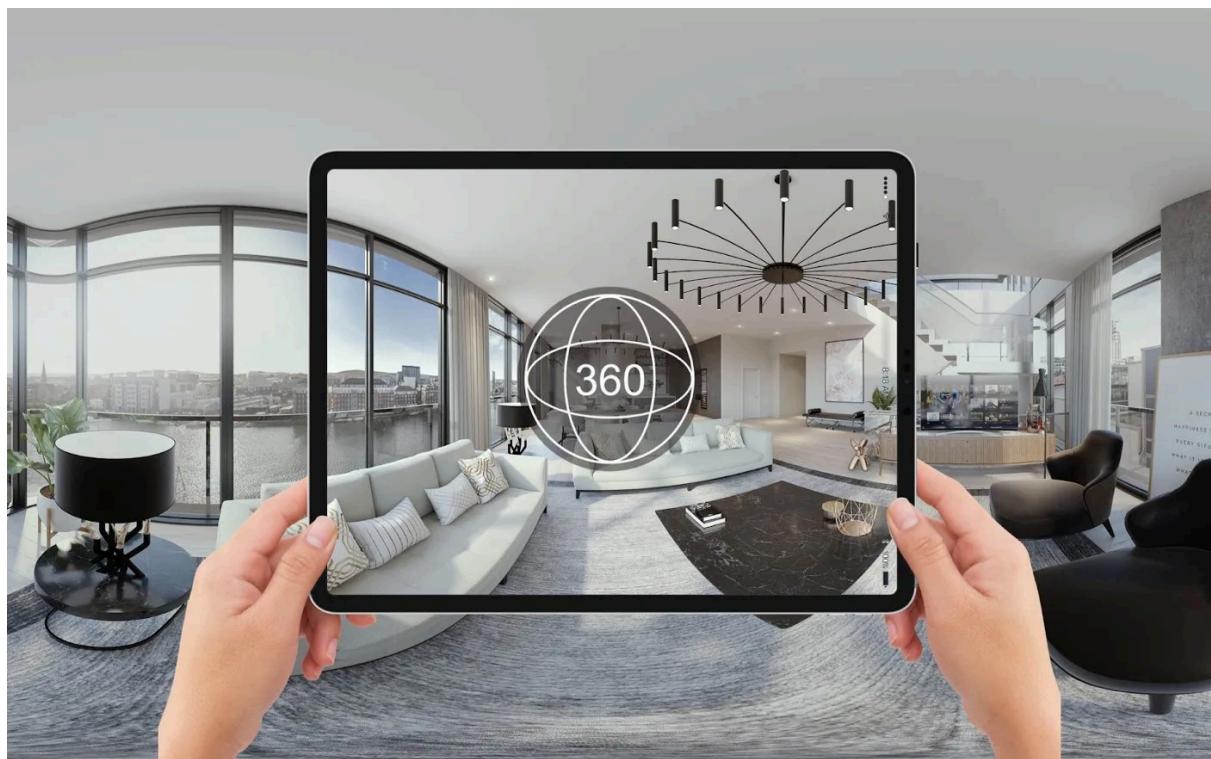


5) Jars



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).

```

1  #include <iostream>
2  #include <string>
3  #include <vector>
4  #include <cmath>
5
6  #include <GL/glew.h>
7  #include <GLFW/glfw3.h>
8
9  #include <glm/glm.hpp>
10 #include <glm/gtc/matrix_transform.hpp>
11 #include <glm/gtc/type_ptr.hpp>
12
13 #include "stb_image.h"
14 #include "Shader.h"
15 #include "Camera.h"
16 #include "Model.h"
17 #include <ctime>
18 #include <stdlib.h>
19
20 // Constante PI
21 const float PI = 3.14159265359f;
22 // Constantes
23 const int WINDOW_WIDTH = 800;
24 const GLuint WINDOW_HEIGHT = 600;
25
26 // Variables globales para cámara
27 Camera camera(glm::vec3(0.0f, 0.0f, 3.0f));
28 GLfloat lastX = WINDOW_WIDTH / 2.0f;
29 GLfloat lastY = WINDOW_HEIGHT / 2.0f;
30 bool firstMouse = true;
31 bool keys[1024] = { false };
32
33 // Variables para animaciones del modelo
34 float puertaRefrigerador = 0.0f;
35 bool animandoPuerta = false;
36 float puertaDestino = 0.0f;
37
38 // Variables para animación de banderines
39 float banderinesInclinacion = 0.0f;
40 bool animandoBanderines = false;
41 float banderinesDestino = 0.0f;
42 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.

```

43     // Sistema de iluminación
44     bool lightStates[4] = { false, false, false, true }; // Estados de las 3 lámparas + Luna
45
46     // Posiciones de las luces puntuales
47     glm::vec3 pointLightPositions[] = {
48         glm::vec3(0.6f, 3.0f, -2.0f), // LamparasCuarto2
49         glm::vec3(0.6f, 3.0f, 2.0f), // LamparasCuarto1
50         glm::vec3(-2.9f, 2.0f, 5.0f), // LinternaLocal
51         glm::vec3(4.0f, 8.0f, 5.0f) // Luna
52     };
53
54     // Deltatime
55     GLfloat deltaTime = 0.0f;
56     GLfloat lastFrame = 0.0f;
57     GLfloat currentFrame = 0.0f;
58     // Variables para animación de agua del grifo (AGREGAR)
59     bool aguaCayendo = false;
60     struct Gota {
61         glm::vec3 posicion;
62         float velocidad;
63         bool activa;
64     };
65     std::vector<Gota> gotas;
66     const int MAX_GOTAS = 50;
67     glm::vec3 posicionGrifo = glm::vec3(1.97f, 1.3f, 1.77f); // Ajusta según tu fregadero
68     float tiempolatigotasa = 0.0f;
69     const float intervaloGotas = 0.02f;
70     // Variables para fuegos artificiales
71     bool fuegosActivos = false;
72     struct Particula {
73         glm::vec3 posicion;
74         glm::vec3 velocidad;
75         glm::vec3 color;
76         float vida;
77         float vidaInicial;
78         float tamano;
79         bool activa;
80     };
81     std::vector<Particula> particulas;
82     const int MAX_PARTICULAS = 1000;
83     float tiempolatigofuego = 0.0f;
84     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.

```

85     void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode);
86     void MouseCallback(GLFWwindow* window, double xPos, double yPos);
87     void ProcessMovement();
88
89     GLuint LoadTexture(const char* path)
90     {
91         GLuint textureID = 0;
92         glGenTextures(1, &textureID);
93
94         int width = 0, height = 0, nrChannels = 0;
95         stbi_set_flip_vertically_on_load(true);
96         unsigned char* data = stbi_load(path, &width, &height, &nrChannels, 0);
97
98         if (data)
99         {
100             GLenum format = GL_RGB;
101             if (nrChannels == 4)
102                 format = GL_RGBA;
103             else if (nrChannels == 1)
104                 format = GL_RED;
105
106             glBindTexture(GL_TEXTURE_2D, textureID);
107             glTexImage2D(GL_TEXTURE_2D, 0, format, width, height, 0, format, GL_UNSIGNED_BYTE, data);
108             glGenerateMipmap(GL_TEXTURE_2D);
109
110             glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
111             glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
112             glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
113             glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
114
115             std::cout << "Textura cargada: " << path << std::endl;
116         }
117         else
118         {
119             std::cout << "Error al cargar: " << path << std::endl;
120         }
121
122         stbi_image_free(data);
123         glBindTexture(GL_TEXTURE_2D, 0);
124
125         return textureID;
126     }
127 // Fuentes: www.gamedev.tips

```

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.

```

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 círculo
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 });
171      }
172  }

```

```

171     }
172
173     // indices
174     for (int i = 1; i <= segments; i++) {
175         indices.push_back(0);
176         indices.push_back(i);
177         indices.push_back(i + 1);
178     }
179
180     geo->indexCount = (GLuint)indices.size();
181
182     glGenVertexArrays(1, &geo->VAO);
183     glGenBuffers(1, &geo->VBO);
184     glGenBuffers(1, &geo->EBO);
185
186     glBindVertexArray(geo->VAO);
187
188     glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
189     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
190
191     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
192     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
193
194     // Posición
195     glEnableVertexAttribArray(0);
196     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
197     // Normal
198     glEnableVertexAttribArray(1);
199     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
200     // TexCoords
201     glEnableVertexAttribArray(2);
202     glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
203
204     glBindVertexArray(0);
205
206     return geo;
207 }
208
209 // Función para crear una manecilla del reloj
210 SimpleGeometry* CreateClockHand(GLuint texture, float width, float length) {
211     SimpleGeometry* geo = new SimpleGeometry();
212     geo->texture = texture;
213
214     std::vector<GLfloat> vertices = {
215         // Posición           Normal           TexCoords
216         -width / 2, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
217         width / 2, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
218         width / 2, length, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f,
219         -width / 2, length, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f
220     };
221
222     std::vector<GLuint> indices = { 0, 1, 2, 2, 3, 0 };
223     geo->indexCount = 6;
224
225     glGenVertexArrays(1, &geo->VAO);
226     glGenBuffers(1, &geo->VBO);
227     glGenBuffers(1, &geo->EBO);
228
229     glBindVertexArray(geo->VAO);
230
231     glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
232     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
233
234     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
235     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
236
237     glEnableVertexAttribArray(0);
238     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
239     glEnableVertexAttribArray(1);
240     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
241     glEnableVertexAttribArray(2);
242     glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
243
244     glBindVertexArray(0);
245
246     return geo;
247 }
248
249 // Función para crear un cubo
250 SimpleGeometry* CreateCube(GLuint texture) {
251     SimpleGeometry* geo = new SimpleGeometry();
252     geo->texture = texture;
253
254     std::vector<GLfloat> vertices = {
255         // Cara frontal (z+)

```

```

249     // Función para crear un cubo
250     SimpleGeometry* CreateCube(GLuint texture) {
251         SimpleGeometry* geo = new SimpleGeometry();
252         geo->texture = texture;
253
254         std::vector<GLfloat> vertices = {
255             // Cara frontal (z+)
256             -0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
257             0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f,
258             0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
259             -0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,
260
261             // Cara trasera (z-)
262             -0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
263             -0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
264             0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
265             0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f,
266
267             // Cara superior (y+)
268             -0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
269             -0.5f, 0.5f, 0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
270             0.5f, 0.5f, 0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
271             0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
272
273             // Cara inferior (y-)
274             -0.5f, -0.5f, -0.5f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
275             0.5f, -0.5f, -0.5f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
276             0.5f, -0.5f, 0.5f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
277             -0.5f, -0.5f, 0.5f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
278
279             // Cara derecha (x+)
280             0.5f, -0.5f, -0.5f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
281             0.5f, 0.5f, -0.5f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
282             0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
283             0.5f, -0.5f, 0.5f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
284
285             // Cara izquierda (x-)
286             -0.5f, -0.5f, -0.5f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
287             -0.5f, -0.5f, 0.5f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
288             -0.5f, 0.5f, 0.5f, -1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
289             -0.5f, 0.5f, -0.5f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f
290         };
291     }

```

```

292     std::vector<GLuint> indices = {
293         0, 1, 2, 3, 0,      // Frontal
294         4, 5, 6, 7, 4,      // Trasera
295         8, 9, 10, 11, 8,    // Superior
296         12, 13, 14, 14, 15, // Inferior
297         16, 17, 18, 18, 19, // Derecha
298         20, 21, 22, 23, 20 // Izquierda
299     };
300
301     geo->indexCount = 36;
302
303     glGenVertexArrays(1, &geo->VAO);
304     glGenBuffers(1, &geo->VBO);
305     glGenBuffers(1, &geo->EBO);
306
307     glBindVertexArray(geo->VAO);
308
309     glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
310     glBindBuffer(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
311
312     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
313     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
314
315     glEnableVertexAttribArray(0);
316     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
317     glEnableVertexAttribArray(1);
318     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
319     glEnableVertexAttribArray(2);
320     glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
321
322     glBindVertexArray(0);
323
324     return geo;
325 }
326
327 // Función para crear superficie de mesa
328 SimpleGeometry* CreateMesaTop(GLuint texture) {
329     SimpleGeometry* geo = new SimpleGeometry();
330     geo->texture = texture;
331
332     std::vector<GLfloat> vertices = {
333         // Superficie superior
334         -0.5f, 0.05f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
335         0.5f, 0.05f, -0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
336         0.5f, 0.05f, 0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
337         -0.5f, 0.05f, 0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
338
339         // Superficie inferior
340         -0.5f, 0.0f, -0.5f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
341         0.5f, 0.0f, -0.5f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
342         0.5f, 0.0f, 0.5f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
343         -0.5f, 0.0f, 0.5f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
344
345         // Borde frontal (Z-)
346         -0.5f, 0.0f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f,
347         0.5f, 0.0f, -0.5f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
348         0.5f, 0.05f, -0.5f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
349         -0.5f, 0.05f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
350
351         // Borde trasero (Z+)
352         -0.5f, 0.0f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
353         0.5f, 0.0f, 0.5f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f,
354         0.5f, 0.05f, 0.5f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f,
355         -0.5f, 0.05f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,
356
357         // Borde derecho (X+)
358         0.5f, 0.0f, -0.5f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
359         0.5f, 0.0f, 0.5f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
360         0.5f, 0.05f, 0.5f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
361         0.5f, 0.05f, -0.5f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
362
363         // Borde izquierdo (X-)
364         -0.5f, 0.0f, -0.5f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
365         -0.5f, 0.0f, 0.5f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
366         -0.5f, 0.05f, 0.5f, -1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
367         -0.5f, 0.05f, -0.5f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f
368     };
369
370     std::vector<GLuint> indices = {
371         0, 1, 2, 2, 3, 0,      // Superior
372         4, 6, 5, 4, 7, 6,      // Inferior
373         8, 9, 10, 10, 11, 8,    // Borde frontal
374         12, 14, 13, 12, 15, 14, // Borde trasero
375         16, 17, 18, 18, 19, 16, // Borde derecho
376         20, 22, 21, 20, 23, 22 // Borde izquierdo
377     };
378 }

```

```

379     geo->indexCount = 36;
380
381     glGenVertexArrays(1, &geo->VAO);
382     glGenBuffers(1, &geo->VBO);
383     glGenBuffers(1, &geo->EBO);
384
385     glBindVertexArray(geo->VAO);
386
387     glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
388     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
389
390     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
391     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
392
393     glEnableVertexAttribArray(0);
394     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
395     glEnableVertexAttribArray(1);
396     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
397     glEnableVertexAttribArray(2);
398     glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
399
400     glBindVertexArray(0);
401
402     return geo;
403 }
404 // Función para crear pata de mesa
405 SimpleGeometry* CreatePataMesa(GLuint texture) {
406     SimpleGeometry* geo = new SimpleGeometry();
407     geo->texture = texture;
408
409     std::vector<GLfloat> vertices = {
410         // Cilindro simplificado como cubo delgado
411         -0.05f, -0.7f, -0.05f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
412         0.05f, -0.7f, -0.05f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
413         0.05f, -0.7f, 0.05f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
414         -0.05f, -0.7f, 0.05f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
415
416         -0.05f, 0.0f, -0.05f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
417         0.05f, 0.0f, -0.05f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
418         0.05f, 0.0f, 0.05f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
419         -0.05f, 0.0f, 0.05f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f
420     };
421
422     std::vector<GLuint> indices = {
423         0, 1, 2, 2, 3, 0, // Abajo
424         4, 6, 5, 4, 7, 6, // Arriba
425         0, 1, 5, 5, 4, 0, // Laterales
426         1, 2, 6, 6, 5, 1,
427         2, 3, 7, 7, 6, 2,
428         3, 0, 4, 4, 7, 3
429     };
430
431     geo->indexCount = 36;
432
433     glGenVertexArrays(1, &geo->VAO);
434     glGenBuffers(1, &geo->VBO);
435     glGenBuffers(1, &geo->EBO);
436
437     glBindVertexArray(geo->VAO);
438
439     glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
440     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
441
442     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
443     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
444
445     glEnableVertexAttribArray(0);
446     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
447     glEnableVertexAttribArray(1);
448     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
449     glEnableVertexAttribArray(2);
450     glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
451
452     glBindVertexArray(0);
453
454     return geo;
455 }
456 // Función para crear Narutomaki (cilindro pequeño)
457 SimpleGeometry* CreateNarutomaki(GLuint texture) {
458     SimpleGeometry* geo = new SimpleGeometry();
459     geo->texture = texture;
460
461     std::vector<GLfloat> vertices = {
462         // Base inferior (y = 0.0f)
463         -0.05f, 0.0f, -0.05f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
464         0.05f, 0.0f, -0.05f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,

```

```

465     0.05f, 0.0f, 0.05f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
466     -0.05f, 0.0f, 0.05f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
467
468     // Base superior (y = 0.05f)
469     -0.05f, 0.05f, -0.05f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
470     0.05f, 0.05f, -0.05f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
471     0.05f, 0.05f, 0.05f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
472     -0.05f, 0.05f, 0.05f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
473
474     // Lado frontal (Z+)
475     -0.05f, 0.0f, 0.05f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
476     0.05f, 0.0f, 0.05f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f,
477     0.05f, 0.05f, 0.05f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
478     -0.05f, 0.05f, 0.05f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,
479
480     // Lado trasero (Z-)
481     -0.05f, 0.0f, -0.05f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
482     0.05f, 0.0f, -0.05f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f,
483     0.05f, 0.05f, -0.05f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
484     -0.05f, 0.05f, -0.05f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
485
486     // Lado derecho (X+)
487     0.05f, 0.0f, -0.05f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
488     0.05f, 0.0f, 0.05f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
489     0.05f, 0.05f, 0.05f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
490     0.05f, 0.05f, -0.05f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
491
492     // Lado izquierdo (X-)
493     -0.05f, 0.0f, -0.05f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
494     -0.05f, 0.0f, 0.05f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
495     -0.05f, 0.05f, 0.05f, -1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
496     -0.05f, 0.05f, -0.05f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
497 };
498
499     std::vector<GLuint> indices = {
500     0, 1, 2, 2, 3, 0,      // Base inferior
501     4, 6, 5, 4, 7, 6,      // Base superior
502     8, 9, 10, 10, 11, 8,    // Frontal
503     12, 14, 13, 12, 15, 14, // Trasero
504     16, 17, 18, 18, 19, 16, // Derecho
505     20, 22, 21, 20, 23, 22 // Izquierdo
506 };
507
508     geo->indexCount = 36;
509
510     glGenVertexArrays(1, &geo->VAO);
511     glGenBuffers(1, &geo->VBO);
512     glGenBuffers(1, &geo->EBO);
513
514     glBindVertexArray(geo->VAO);
515
516     glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
517     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
518
519     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
520     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
521
522     glEnableVertexAttribArray(0);
523     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
524     glEnableVertexAttribArray(1);
525     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
526     glEnableVertexAttribArray(2);
527     glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
528
529     glBindVertexArray(0);
530
531     return geo;
532 }
533
534     // Función para crear Sartén
535     SimpleGeometry* CreateSarten(GLuint texture) {
536     SimpleGeometry* geo = new SimpleGeometry();
537     geo->texture = texture;
538
539     std::vector<GLfloat> vertices = {
540     // Base inferior
541     -0.2f, 0.0f, -0.2f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
542     0.2f, 0.0f, -0.2f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
543     0.2f, 0.0f, 0.2f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
544     -0.2f, 0.0f, 0.2f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
545
546     // Base superior
547     -0.2f, 0.02f, -0.2f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
548     0.2f, 0.02f, -0.2f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
549     0.2f, 0.02f, 0.2f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
550     -0.2f, 0.02f, 0.2f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
551
552     // Lado frontal
553     -0.2f, 0.05f, -0.05f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
554     0.2f, 0.05f, -0.05f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
555     0.2f, 0.05f, 0.05f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
556     -0.2f, 0.05f, 0.05f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
557
558     // Lado trasero
559     -0.2f, 0.05f, 0.05f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
560     0.2f, 0.05f, 0.05f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
561     0.2f, 0.05f, -0.05f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
562     -0.2f, 0.05f, -0.05f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
563
564     // Lado derecho
565     0.05f, 0.05f, -0.05f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
566     0.05f, 0.05f, 0.05f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
567     0.05f, 0.05f, 0.05f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
568     0.05f, 0.05f, -0.05f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
569
570     // Lado izquierdo
571     -0.05f, 0.05f, 0.05f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
572     -0.05f, 0.05f, -0.05f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
573     -0.05f, 0.05f, -0.05f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
574     -0.05f, 0.05f, 0.05f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
575
576     // Sartén
577     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
578     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
579     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
580     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
581     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
582     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
583     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
584     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
585     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
586     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
587     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
588     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
589     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
590     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
591     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
592     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
593     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
594     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
595     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
596     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
597     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
598     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
599     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
600     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
601     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
602     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
603     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
604     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
605     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
606     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
607     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
608     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
609     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
610     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
611     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
612     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
613     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
614     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
615     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
616     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
617     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
618     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
619     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
620     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
621     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
622     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
623     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
624     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
625     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
626     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
627     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
628     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
629     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
630     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
631     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
632     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
633     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
634     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
635     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
636     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
637     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
638     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
639     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
640     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
641     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
642     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
643     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
644     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
645     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
646     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
647     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
648     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
649     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
650     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
651     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
652     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
653     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
654     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
655     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
656     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
657     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
658     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
659     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
660     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
661     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
662     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
663     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
664     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
665     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
666     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
667     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
668     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
669     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
670     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
671     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
672     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
673     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
674     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
675     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
676     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
677     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
678     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
679     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
680     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
681     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
682     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
683     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
684     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
685     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
686     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
687     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
688     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
689     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
690     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
691     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
692     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
693     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
694     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
695     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
696     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
697     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
698     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
699     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
700     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
701     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
702     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
703     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
704     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
705     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
706     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
707     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
708     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
709     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
710     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
711     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
712     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
713     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
714     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
715     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
716     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
717     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
718     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
719     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
720     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
721     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
722     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
723     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
724     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
725     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
726     0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
727     0.
```

```

552     // Lado frontal
553     -0.2f, 0.0f, 0.2f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
554     0.2f, 0.0f, 0.2f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f,
555     0.2f, 0.02f, 0.2f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
556     -0.2f, 0.02f, 0.2f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,
557
558     // Lado trasero
559     -0.2f, 0.0f, -0.2f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
560     0.2f, 0.0f, -0.2f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f,
561     0.2f, 0.02f, -0.2f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
562     -0.2f, 0.02f, -0.2f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
563
564     // Lado derecho
565     0.2f, 0.0f, -0.2f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
566     0.2f, 0.0f, 0.2f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
567     0.2f, 0.02f, 0.2f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
568     0.2f, 0.02f, -0.2f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
569
570     // Lado izquierdo
571     -0.2f, 0.0f, -0.2f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
572     -0.2f, 0.0f, 0.2f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
573     -0.2f, 0.02f, 0.2f, -1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
574     -0.2f, 0.02f, -0.2f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
575
576     };
577     std::vector<GLuint> indices = {
578         0, 1, 2, 2, 3, 0,
579         4, 6, 5, 4, 7, 6,
580         8, 9, 10, 10, 11, 8,
581         12, 14, 13, 12, 15, 14,
582         16, 17, 18, 18, 19, 16,
583         20, 22, 21, 20, 23, 22
584     };
585
586     geo->indexCount = 36;
587
588     glGenVertexArrays(1, &geo->VAO);
589     glGenBuffers(1, &geo->VBO);
590     glGenBuffers(1, &geo->EBO);
591
592     glBindVertexArray(geo->VAO);
593
594     glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
595     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
596
597     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
598     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
599
600     glEnableVertexAttribArray(0);
601     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
602     glEnableVertexAttribArray(1);
603     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
604     glEnableVertexAttribArray(2);
605     glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
606
607     glBindVertexArray(0);
608
609     return geo;
610 }
611
612 // Función para crear Tabla de Pícar
613 SimpleGeometry* CreateTablaPicar(GLuint texture) {
614     SimpleGeometry* geo = new SimpleGeometry();
615     geo->texture = texture;
616
617     std::vector<GLfloat> vertices = {
618         // Base inferior
619         -0.2f, 0.0f, -0.15f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
620         0.2f, 0.0f, -0.15f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
621         0.2f, 0.0f, 0.15f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
622         -0.2f, 0.0f, 0.15f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
623
624         // Base superior
625         -0.2f, 0.02f, -0.15f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
626         0.2f, 0.02f, -0.15f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
627         0.2f, 0.02f, 0.15f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
628         -0.2f, 0.02f, 0.15f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
629
630         // Lado frontal
631         -0.2f, 0.0f, 0.15f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
632         0.2f, 0.0f, 0.15f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f,
633         0.2f, 0.02f, 0.15f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
634         -0.2f, 0.02f, 0.15f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,
635
636         // Lado trasero
637         -0.2f, 0.0f, -0.15f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
638         0.2f, 0.0f, -0.15f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
639         -0.2f, 0.02f, -0.15f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
640
641         // Lado derecho
642         0.2f, 0.0f, -0.15f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
643         0.2f, 0.0f, 0.15f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
644         0.2f, 0.02f, 0.15f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
645         0.2f, 0.02f, -0.15f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
646
647         // Lado izquierdo
648         -0.2f, 0.0f, -0.15f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
649         -0.2f, 0.0f, 0.15f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
650         -0.2f, 0.02f, 0.15f, -1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
651         -0.2f, 0.02f, -0.15f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
652
653     };
654
655     std::vector<GLuint> indices = {
656         0, 1, 2, 2, 3, 0,
657         4, 6, 5, 4, 7, 6,
658         8, 9, 10, 10, 11, 8,
659         12, 14, 13, 12, 15, 14,
660         16, 17, 18, 18, 19, 16,
661         20, 22, 21, 20, 23, 22
662     };
663
664     geo->indexCount = 36;
665
666     glGenVertexArrays(1, &geo->VAO);
667     glGenBuffers(1, &geo->VBO);
668     glGenBuffers(1, &geo->EBO);
669
670     glBindVertexArray(geo->VAO);
671
672     glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
673     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
674
675     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
676     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
677
678     glEnableVertexAttribArray(0);
679     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
680     glEnableVertexAttribArray(1);

```

```

681     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
682     glEnableVertexAttribArray(2);
683     glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
684
685     glBindVertexArray(0);
686
687     return geo;
688 }
689
690 // Función para crear hoja de cuchillo (rectángulo delgado alargado)
691 SimpleGeometry* CreateHojaCuchillo(GLuint texture) {
692     SimpleGeometry* geo = new SimpleGeometry();
693     geo->texture = texture;
694
695     std::vector<GLfloat> vertices = {
696         // Base inferior
697         -0.025f, 0.0f, -0.2f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
698         0.025f, 0.0f, -0.2f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
699         0.025f, 0.0f, 0.2f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
700         -0.025f, 0.0f, 0.2f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
701
702         // Base superior
703         -0.025f, 0.02f, -0.2f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
704         0.025f, 0.02f, -0.2f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
705         0.025f, 0.02f, 0.2f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
706         -0.025f, 0.02f, 0.2f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
707
708         // Lado frontal (Z+)
709         -0.025f, 0.0f, 0.2f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
710         0.025f, 0.0f, 0.2f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
711         0.025f, 0.02f, 0.2f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f,
712         -0.025f, 0.02f, 0.2f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,
713
714         // Lado trasero (Z-)
715         -0.025f, 0.0f, -0.2f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
716         0.025f, 0.0f, -0.2f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f,
717         0.025f, 0.02f, -0.2f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
718         -0.025f, 0.02f, -0.2f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
719
720         // Lado derecho (Y+)
721         0.025f, 0.0f, -0.2f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
722         0.025f, 0.0f, -0.2f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
723         0.025f, 0.02f, -0.2f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
724
725         // Lado izquierdo (X-)
726         -0.025f, 0.0f, -0.2f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
727         0.025f, 0.0f, -0.2f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
728         0.025f, 0.02f, -0.2f, -1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
729         -0.025f, 0.02f, -0.2f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
730     };
731
732     std::vector<GLuint> indices = {
733         0, 1, 2, 2, 3, 0,           // Base inferior
734         4, 6, 5, 4, 7, 6,           // Base superior
735         8, 9, 10, 10, 11, 8,           // Frontal
736         12, 14, 13, 12, 15, 14,           // Trasero
737         16, 17, 18, 18, 19, 16,           // Derecho
738         20, 22, 21, 20, 23, 22           // Izquierdo
739     };
740
741     geo->indexCount = 36;
742
743     glGenVertexArrays(1, &geo->VAO);
744     glGenBuffers(1, &geo->VBO);
745     glGenBuffers(1, &geo->EBO);
746
747     glBindVertexArray(geo->VAO);
748
749     glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
750     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
751
752     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
753     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
754
755     glEnableVertexAttribArray(0);
756     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
757     glEnableVertexAttribArray(1);
758     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
759     glEnableVertexAttribArray(2);
760     glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
761
762     glBindVertexArray(0);
763
764     return geo;
765 }
766
767 // Función para crear mango de cuchillo (cubo más pequeño)

```

```

768     SimpleGeometry* CreateMangoCuchillo(GLuint texture) {
769         SimpleGeometry* geo = new SimpleGeometry();
770         geo->texture = texture;
771
772         std::vector<GLfloat> vertices = {
773             // Base inferior
774             -0.025f, 0.0f, -0.05f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
775             0.025f, 0.0f, -0.05f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
776             0.025f, 0.05f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
777             -0.025f, 0.05f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
778
779             // Base superior
780             -0.025f, 0.02f, -0.05f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
781             0.025f, 0.02f, -0.05f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
782             0.025f, 0.02f, 0.05f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
783             -0.025f, 0.02f, 0.05f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
784
785             // Lado frontal (Z+)
786             -0.025f, 0.0f, 0.05f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
787             0.025f, 0.0f, 0.05f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
788             0.025f, 0.02f, 0.05f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,
789             -0.025f, 0.02f, 0.05f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,
790
791             // Lado trasero (Z-)
792             -0.025f, 0.0f, -0.05f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
793             0.025f, 0.0f, -0.05f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f,
794             0.025f, 0.02f, -0.05f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
795             -0.025f, 0.02f, -0.05f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
796
797             // Lado derecho (X+)
798             0.025f, 0.0f, -0.05f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
799             0.025f, 0.0f, 0.05f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
800             0.025f, 0.02f, 0.05f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
801             0.025f, 0.02f, -0.05f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
802
803             // Lado izquierdo (X-)
804             -0.025f, 0.0f, -0.05f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
805             -0.025f, 0.0f, 0.05f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
806             -0.025f, 0.02f, 0.05f, -1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
807             -0.025f, 0.02f, -0.05f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
808         };
809
810         std::vector<GLuint> indices = {
811             0, 1, 2, 2, 3, 0,
812             4, 6, 5, 4, 7, 6,
813             8, 9, 10, 10, 11, 8,
814             12, 14, 13, 12, 15, 14,
815             16, 17, 18, 18, 19, 16,
816             20, 22, 21, 20, 23, 22
817         };
818
819         geo->indexCount = 36;
820
821         glGenVertexArrays(1, &geo->VAO);
822         glGenBuffers(1, &geo->VBO);
823         glGenBuffers(1, &geo->EBO);
824
825         glBindVertexArray(geo->VAO);
826
827         glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
828         glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
829
830         glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
831         glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
832
833         glEnableVertexAttribArray(0);
834         glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
835         glEnableVertexAttribArray(1);
836         glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
837         glEnableVertexAttribArray(2);
838         glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
839
840         glBindVertexArray(0);
841
842         return geo;
843     }
844
845     // Función para crear una gota de agua (cubo alargado con grosor medio)
846     SimpleGeometry* CreateGotaAgua(GLuint texture) {
847         SimpleGeometry* geo = new SimpleGeometry();
848         geo->texture = texture;
849
850         // Gotas con grosor medio (0.008) y largas (0.04)
851         std::vector<GLfloat> vertices = {
852             -0.008f, -0.04f, 0.008f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
853             0.008f, -0.04f, 0.008f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
854             0.008f, 0.04f, 0.008f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,
855             -0.008f, 0.04f, 0.008f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,

```

```

854 |     -0.008f, 0.04f, 0.008f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,
855 |     -0.008f, -0.04f, -0.008f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
856 |     -0.008f, 0.04f, -0.008f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
857 |     0.008f, 0.04f, -0.008f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
858 |     0.008f, -0.04f, -0.008f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f,
859 |     0.008f, -0.04f, -0.008f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f,
860 |     -0.008f, 0.04f, -0.008f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
861 |     -0.008f, 0.04f, 0.008f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
862 |     0.008f, 0.04f, 0.008f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
863 |     0.008f, -0.04f, 0.008f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
864 |     0.008f, -0.04f, -0.008f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
865 |     -0.008f, -0.04f, -0.008f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
866 |     0.008f, -0.04f, -0.008f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
867 |     0.008f, -0.04f, 0.008f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
868 |     0.008f, -0.04f, 0.008f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
869 |     -0.008f, -0.04f, 0.008f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
870 |     0.008f, -0.04f, -0.008f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
871 |     0.008f, 0.04f, -0.008f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
872 |     0.008f, 0.04f, 0.008f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
873 |     0.008f, -0.04f, 0.008f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
874 |     0.008f, -0.04f, -0.008f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
875 |     -0.008f, -0.04f, -0.008f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
876 |     -0.008f, -0.04f, 0.008f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
877 |     -0.008f, 0.04f, 0.008f, -1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
878 |     -0.008f, 0.04f, -0.008f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
879 |     };
880 | };
881 | std::vector<GLuint> indices = {
882 |     0, 1, 2, 2, 3, 0,
883 |     4, 5, 6, 6, 7, 4,
884 |     8, 9, 10, 10, 11, 8,
885 |     12, 13, 14, 14, 15, 12,
886 |     16, 17, 18, 18, 19, 16,
887 |     20, 21, 22, 22, 23, 20
888 | };
889 | geo->indexCount = 36;
890 | glGenVertexArrays(1, &geo->VAO);
891 | glGenBuffers(1, &geo->VBO);
892 | glGenBuffers(1, &geo->EBO);
893 | glGenBuffers(1, &geo->EBO);
894 | glBindVertexArray(geo->VAO);
895 | glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
896 | glBindBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
897 | glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
898 | glBindBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
899 | glEnableVertexAttribArray(0);
900 | glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
901 | glEnableVertexAttribArray(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
902 | glEnableVertexAttribArray(2, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
903 | glBindVertexArray(0);
904 | return geo;
905 | }
906 | // Crear esfera pequeña para partículas de fuegos artificiales
907 | SimpleGeometry* CreateParticulaFuego(GLuint texture) {
908 |     SimpleGeometry* geo = new SimpleGeometry();
909 |     geo->texture = texture;
910 |     std::vector<GLfloat> vertices = {
911 |         // Cara frontal
912 |         -0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
913 |         0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
914 |         0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
915 |         -0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
916 |         // Cara trasera
917 |         -0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f, 0.0f,
918 |         -0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f, 0.0f,
919 |         0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
920 |         0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
921 |         // Cara superior
922 |         -0.5f, 0.5f, -0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
923 |         -0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
924 |         0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
925 |         0.5f, 0.5f, -0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
926 |         // Cara inferior
927 |         -0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
928 |         -0.5f, -0.5f, 0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
929 |         0.5f, -0.5f, 0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
930 |         0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
931 |         -0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
932 |         -0.5f, 0.5f, 0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
933 |         // Cara izquierda
934 |         -0.5f, -0.5f, -0.5f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f,
935 |         -0.5f, -0.5f, 0.5f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f,
936 |         0.5f, -0.5f, 0.5f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f,
937 |         0.5f, -0.5f, -0.5f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f,
938 |         -0.5f, 0.5f, -0.5f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f
939 |     };
940 |     std::vector<GLuint> indices = {
941 |         0, 1, 2, 2, 3, 0,
942 |         4, 5, 6, 6, 7, 4,
943 |         8, 9, 10, 10, 11, 8,
944 |         12, 13, 14, 14, 15, 12,
945 |         16, 17, 18, 18, 19, 16,
946 |         20, 21, 22, 22, 23, 20
947 |     };
948 |     geo->indexCount = 36;
949 |     glGenVertexArrays(1, &geo->VAO);
950 |     glGenBuffers(1, &geo->VBO);
951 |     glGenBuffers(1, &geo->EBO);
952 |     glBindVertexArray(geo->VAO);
953 |     glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
954 |     glBindBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
955 |     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
956 |     glBindBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
957 | };
958 | std::vector<GLuint> indices = {
959 |     0, 1, 2, 2, 3, 0,
960 |     4, 5, 6, 6, 7, 4,
961 |     8, 9, 10, 10, 11, 8,
962 |     12, 13, 14, 14, 15, 12,
963 |     16, 17, 18, 18, 19, 16,
964 |     20, 21, 22, 22, 23, 20
965 | };
966 | geo->indexCount = 36;
967 | glBindVertexArray(geo->VAO);
968 | glBindBuffer(GL_ARRAY_BUFFER, geo->VBO);
969 | glBindBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), &vertices[0], GL_STATIC_DRAW);
970 | glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, geo->EBO);
971 | glBindBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint), &indices[0], GL_STATIC_DRAW);
972 | glEnableVertexAttribArray(0);
973 | 
```

```

983     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)0);
984     glEnableVertexAttribArray(1);
985     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
986     glEnableVertexAttribArray(2);
987     glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
988
989     glBindVertexArray(0);
990
991     return geo;
992 }
993
994     void CreateExplosion(glm::vec3 posicion, glm::vec3 color) {
995         int partículasPorExplosión = 250; // MUCHAS MÁS PARTÍCULAS
996         int creadas = 0;
997
998         for (int i = 0; i < MAX_PARTÍCULAS && creadas < partículasPorExplosión; i++) {
999             if (!partículas[i].activa) {
1000                 partículas[i].activa = true;
1001                 partículas[i].posición = posicion;
1002                 partículas[i].vida = 3.0f + (rand() % 150) / 50.0f; // Viven MÁS tiempo
1003                 partículas[i].vidaInicial = partículas[i].vida;
1004                 partículas[i].color = color;
1005                 partículas[i].tamaño = 0.35f + (rand() % 150) / 500.0f; // MUY GRANDES (0.35-0.65)
1006
1007                 // Velocidades MUY RÁPIDAS para explosiones grandes
1008                 float theta = (rand() % 360) * PI / 180.0f;
1009                 float phi = (rand() % 180) * PI / 180.0f;
1010                 float velocidad = 5.0f + (rand() % 600) / 100.0f; // 5.0 a 11.0 (muy rápido)
1011
1012                 partículas[i].velocidad = glm::vec3(
1013                     velocidad * sin(phi) * cos(theta),
1014                     velocidad * sin(phi),
1015                     velocidad * sin(phi) * sin(theta)
1016                 );
1017
1018                 creadas++;
1019             }
1020         }
1021     }
1022
1023     int main()
1024     {
1025         if (!glfwInit())
1026         {
1027             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.

```

1026         return EXIT_FAILURE;
1027     }
1028
1029     glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
1030     glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
1031     glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
1032     glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
1033     glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
1034
1035     GLFWwindow* window = glfwCreateWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Modelado Jerarquico - Cocina 3D", nullptr, nullptr);
1036
1037     if (!window)
1038     {
1039         std::cout << "Error al crear ventana GLFW" << std::endl;
1040         glfwTerminate();
1041         return EXIT_FAILURE;
1042     }
1043
1044     glfwMakeContextCurrent(window);
1045     glfwSetKeyCallback(window, KeyCallback);
1046     glfwSetCursorPosCallback(window, MouseCallback);
1047     glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);
1048
1049     glewExperimental = GL_TRUE;
1050     if (glewInit() != GLEW_OK)
1051     {
1052         std::cout << "Error al inicializar GLEW" << std::endl;
1053         glfwTerminate();
1054         return EXIT_FAILURE;
1055     }
1056
1057     int screenWidth = 0, screenHeight = 0;
1058     glfwGetFramebufferSize(window, &screenWidth, &screenHeight);
1059     glViewport(0, 0, screenWidth, screenHeight);
1060     glEnable(GL_DEPTH_TEST);
1061     glClearColor(0.05f, 0.05f, 0.15f, 1.0f);
1062
1063     Shader lightingShader("Shader/Lighting.vs", "Shader/Lighting.frag");
1064
1065     std::cout << "\n==== CONTROLES ===" << std::endl;
1066     std::cout << "#/A/S/D: Mover cámara" << std::endl;
1067     std::cout << "Mouse: Mirar alrededor" << std::endl;
1068     std::cout << "O/C: Abrir/cerrar puerta del refrigerador" << std::endl;

```

```

1069    std::cout << "N: Mover banderines hacia atrás" << std::endl;
1070    std::cout << "M: Regresar banderines a posición normal" << std::endl;
1071    std::cout << "G: Abrir/Cerrar grifo (agua)" << std::endl;
1072    std::cout << "F: Activar/Desactivar fuegos artificiales" << std::endl;
1073    std::cout << "1/2/3: Encender/Apagar lámparas de cocina" << std::endl;
1074    std::cout << "4: Encender/Apagar luz de la Luna" << std::endl;
1075    std::cout << "ESC: Salir\n" << std::endl;
1076
1077 // Carga de texturas
1078    GLuint textureArroz = LoadTexture("images/5138.jpg");
1079    GLuint textureBarandal = LoadTexture("images/fondo-abstracto-con-textura (1).jpg");
1080    GLuint textureBase = LoadTexture("images/05X3M0.jpg");
1081    GLuint textureBote = LoadTexture("images/0E5TN0.jpg");
1082    GLuint textureBolsaHarinaAbierta1 = LoadTexture("images/Illustration03.jpg");
1083    GLuint textureBolsaHarinaAbierta = LoadTexture("images/Illustration03.jpg");
1084    GLuint textureBolsasHarina = LoadTexture("images/Illustration03.jpg");
1085    GLuint textureBoteBasura = LoadTexture("images/fondo-abstracto-con-textura.jpg");
1086    GLuint textureCadenalInterna = LoadTexture("images/5178.jpg");
1087    GLuint textureCaja = LoadTexture("images/16.jpg");
1088    GLuint textureCaja2 = LoadTexture("images/370.jpg");
1089    GLuint textureCajasMadera = LoadTexture("images/5178.jpg");
1090    GLuint textureCortina2 = LoadTexture("images/preview.jpg");
1091    GLuint textureCortina = LoadTexture("images/169180-OVUVSU-320.jpg");
1092    GLuint textureCortinas = LoadTexture("images/preview.jpg");
1093    GLuint textureCubetas = LoadTexture("images/0R6ILJ0.jpg");
1094    GLuint textureElectricidad = LoadTexture("images/5178.jpg");
1095    GLuint textureEscaleras = LoadTexture("images/3690.jpg");
1096    GLuint textureEstufa = LoadTexture("images/textura-de-placa-de-metal.jpg");
1097    GLuint textureFregadero = LoadTexture("images/fondo-abstracto-con-textura.jpg");
1098    GLuint textureGas = LoadTexture("images/Q0V5UF0.jpg");
1099    GLuint textureJaladeros = LoadTexture("images/0R6ILJ0.jpg");
1100    GLuint textureLamparasCocina = LoadTexture("images/fondo-abstracto-con-textura.jpg");
1101    GLuint textureLamparasCocina1 = LoadTexture("images/2259.jpg");
1102    GLuint textureLamparas1 = LoadTexture("images/2259.jpg");
1103    GLuint textureLamparas = LoadTexture("images/textura-de-placa-de-metal.jpg");
1104    GLuint textureLibrene = LoadTexture("images/3690.jpg");
1105    GLuint textureLuna = LoadTexture("images/2259.jpg");
1106    GLuint textureInternalLocal = LoadTexture("images/2259.jpg");
1107    GLuint textureMesa = LoadTexture("images/QV54F0.jpg");
1108    GLuint textureOlla = LoadTexture("images/5178.jpg");
1109    GLuint textureOlla1 = LoadTexture("images/5178.jpg");
1110    GLuint textureOlla2 = LoadTexture("images/acuarela-de-oro-liquido-espacio-de-copia.jpg");
1111    GLuint textureOlla3 = LoadTexture("images/acuarela-de-oro-liquido-espacio-de-copia.jpg");
1112
1113    GLuint texturePasto = LoadTexture("images/preview1.jpg");
1114    GLuint texturePared3 = LoadTexture("images/3690.jpg");
1115    GLuint texturePared4 = LoadTexture("images/3690.jpg");
1116    GLuint texturePlatos = LoadTexture("images/0TV360.jpg");
1117    GLuint texturePiso2 = LoadTexture("images/8469.jpg");
1118    GLuint texturePrimerPiso = LoadTexture("images/liquid-marbling-paint-texture-background-fluid");
1119    GLuint texturePuertas = LoadTexture("images/3690.jpg");
1120    GLuint textureRefrigerador = LoadTexture("images/0TV360.jpg");
1121    GLuint textureRefrigador1 = LoadTexture("images/0TV360.jpg");
1122    GLuint textureSilla2 = LoadTexture("images/fondo-abstracto-con-textura.jpg");
1123    GLuint textureSilla = LoadTexture("images/textura-prung-roja.jpg");
1124    GLuint textureTapa = LoadTexture("images/0R6ILJ0.jpg");
1125    GLuint textureTapa1 = LoadTexture("images/0R6ILJ0.jpg");
1126    GLuint textureTapa2 = LoadTexture("images/0R6ILJ0.jpg");
1127    GLuint textureTangueAqua = LoadTexture("images/fondo-abstracto-con-textura.jpg");
1128    GLuint textureTangueAlmacen = LoadTexture("images/07074.jpg");
1129    GLuint textureTapeeteCocina = LoadTexture("images/abstract-blur-empty-green-gradient-studio-w");
1130    GLuint textureTazon = LoadTexture("images/0TV360.jpg");
1131    GLuint textureTazones2 = LoadTexture("images/0TV360.jpg");
1132    GLuint textureTejado = LoadTexture("images/6592778.jpg");
1133    GLuint textureTejadoMini = LoadTexture("images/3690.jpg");
1134    GLuint textureTejadoMinipiso2 = LoadTexture("images/3690.jpg");
1135    GLuint textureTuboFuera = LoadTexture("images/5178.jpg");
1136    GLuint textureTuboGas = LoadTexture("images/5178.jpg");
1137    GLuint textureVentanaSegundoPiso = LoadTexture("images/3690.jpg");
1138    GLuint textureBanderines = LoadTexture("images/preview.jpg");
1139
1140 // Carga de modelos
1141    Model Arroz((char*)"Models/Arroz.obj");
1142    Model Barandal((char*)"Models/Barandal.obj");
1143    Model Base((char*)"Models/Base.obj");
1144    Model Botes((char*)"Models/Botes.obj");
1145    Model BolsaHarinaAbierta1((char*)"Models/BolsaHarinaAbierta1.obj");
1146    Model BolsaHarinaAbierta((char*)"Models/BolsaHarinaAbierta.obj");
1147    Model BolsaHarina((char*)"Models/BolsaHarina.obj");
1148    Model BoteBasura((char*)"Models/BoteBasura.obj");
1149    Model CadenalInterna((char*)"Models/Cadenal.interna.obj");
1150    Model Caja((char*)"Models/Caja.obj");
1151    Model Caja2((char*)"Models/Caja2.obj");
1152    Model CajasMadera((char*)"Models/CajasMadera.obj");
1153    Model Campana((char*)"Models/Campana.obj");
1154    Model Cortina2((char*)"Models/Cortina2.obj");

```

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.

```

1241     }
1242     srand(time(NULL)); // Para números aleatorios
1243     glm::vec3 posReloj = glm::vec3(-2.0f, 2.0f, 0.15f);
1244     glm::vec3 posCaja = glm::vec3(1.5f, 4.0f, -0.3f);
1245     glm::vec3 posMesa = glm::vec3(0.8f, 5.0f, -2.4f);
1246     glm::vec3 posTabla = glm::vec3(-1.5f, 1.1f, 1.5f);
1247     glm::vec3 posSarten = glm::vec3(0.8f, 5.0f, -2.4f);
1248
1249     while (!glfwWindowShouldClose(window))
1250     {
1251         currentTime = (GLfloat)glfwGetTime();
1252         deltaTime = currentTime - lastFrame;
1253         lastFrame = currentTime;
1254
1255         glfwPollEvents();
1256         ProcessMovement();
1257
1258         glClearColor(0.05f, 0.05f, 0.15f, 1.0f);
1259         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
1260
1261         lightingShader.Use();
1262
1263         GLint viewPosLoc = glGetUniformLocation(lightingShader.Program, "viewPos");
1264         glm::vec3 cameraPos = camera.GetPosition();
1265         glUniform3f(viewPosLoc, cameraPos.x, cameraPos.y, cameraPos.z);
1266
1267         glUniform3f(glGetUniformLocation(lightingShader.Program, "dirlight.direction"), -0.2f, -1.0f, -0.3f);
1268         glUniform3f(glGetUniformLocation(lightingShader.Program, "dirlight.ambient"), 0.05f, 0.05f, 0.1f);
1269         glUniform3f(glGetUniformLocation(lightingShader.Program, "dirlight.diffuse"), 0.1f, 0.1f, 0.15f);
1270         glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.2f, 0.2f, 0.3f);
1271
1272         for (int i = 0; i < 4; ++i)
1273         {
1274             std::string prefix = "pointLights[" + std::to_string(i) + "]";
1275
1276             glm::vec3 ambient, diffuse, specular;
1277             float constant, linear, quadratic;
1278
1279             if (i == 3)
1280                 ambient = lightStates[i] ? glm::vec3(0.2f, 0.2f, 0.3f) : glm::vec3(0.0f);
1281                 diffuse = lightStates[i] ? glm::vec3(0.8f, 0.8f, 1.2f) : glm::vec3(0.0f);
1282                 specular = lightStates[i] ? glm::vec3(1.0f, 1.0f, 1.3f) : glm::vec3(0.0f);
1283                 constant = 1.0f;
1284                 linear = 0.014f;
1285
1286                 quadratic = 0.0007f;
1287
1288             else
1289                 ambient = lightStates[i] ? glm::vec3(0.15f, 0.05f, 0.0f) : glm::vec3(0.0f); // tenue
1290                 diffuse = lightStates[i] ? glm::vec3(1.0f, 0.4f, 0.1f) : glm::vec3(0.0f); // naranja-rojizo "chakra"
1291                 specular = lightStates[i] ? glm::vec3(1.0f, 0.6f, 0.2f) : glm::vec3(0.0f); // brillo cálido
1292
1293                 // Atenuación suave para que no se expanda demasiado
1294                 constant = 1.0f;
1295                 linear = 0.07f;
1296                 quadratic = 0.017f;
1297
1298             glUniform3f(glGetUniformLocation(lightingShader.Program, (prefix + ".position").c_str()),
1299                         pointLightPositions[i].x, pointLightPositions[i].y, pointLightPositions[i].z);
1300             glUniform3f(glGetUniformLocation(lightingShader.Program, (prefix + ".ambient").c_str()),
1301                         ambient.x, ambient.y, ambient.z);
1302             glUniform3f(glGetUniformLocation(lightingShader.Program, (prefix + ".diffuse").c_str()),
1303                         diffuse.x, diffuse.y, diffuse.z);
1304             glUniform3f(glGetUniformLocation(lightingShader.Program, (prefix + ".specular").c_str()),
1305                         specular.x, specular.y, specular.z);
1306             glUniform1f(glGetUniformLocation(lightingShader.Program, (prefix + ".constant").c_str()), constant);
1307             glUniform1f(glGetUniformLocation(lightingShader.Program, (prefix + ".linear").c_str()), linear);
1308             glUniform1f(glGetUniformLocation(lightingShader.Program, (prefix + ".quadratic").c_str()), quadratic);
1309
1310             // Inicializar spotlight (apagado) - IMPORTANTE para que el shader funcione
1311             glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.position"), 0.0f, 0.0f, 0.0f);
1312             glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.direction"), 0.0f, -1.0f, 0.0f);
1313             glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.cutOff"), glm::cos(glm::radians(12.5f)));
1314             glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.outerCutOff"), glm::cos(glm::radians(15.0f)));
1315             glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.ambient"), 0.0f, 0.0f, 0.0f);
1316             glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.diffuse"), 0.0f, 0.0f, 0.0f);
1317             glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.specular"), 0.0f, 0.0f, 0.0f);
1318             glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.constant"), 1.0f);
1319             glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.linear"), 0.09f);
1320             glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.quadratic"), 0.032f);
1321
1322             glm::mat4 view = camera.GetViewMatrix();
1323             glm::mat4 model = glm::mat4(1);
1324
1325             GLint modelLoc = glGetUniformLocation(lightingShader.Program, "model");
1326
1327             glUniformMatrix4fv(glGetUniformLocation(lightingShader.Program, "projection"), 1, GL_FALSE, glm::value_ptr(projection));
1328             glUniformMatrix4fv(glGetUniformLocation(lightingShader.Program, "view"), 1, GL_FALSE, glm::value_ptr(view));
1329
1330             glUniform1i(glGetUniformLocation(lightingShader.Program, "transparency"), 0);
1331
1332             // IMPORTANTE: Configurar los samplers de textura
1333             glUniform1i(glGetUniformLocation(lightingShader.Program, "Material.diffuse"), 0);
1334             glUniform1i(glGetUniformLocation(lightingShader.Program, "Material.specular"), 0);
1335
1336             // ===== PRIMERO: FUEGOS ARTIFICIALES (para que estén detrás) =====
1337             if (fuegosActivos)
1338             {
1339                 glEnable(GL_BLEND);
1340                 glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
1341                 // NO desactive el depth test aquí para que respeten la profundidad
1342                 // glDisable(GL_DEPTH_TEST); <-- QUITA ESTA LÍNEA
1343
1344                 for (int i = 0; i < MAX_PARTICULAS; i++)
1345                 {
1346                     if (particulas[i].activa)
1347                     {
1348                         model = glm::mat4(1.0f);
1349                         model = glm::translate(model, particulas[i].posicion);
1350
1351                         float factorVida = particulas[i].vida / particulas[i].vidaInicial;
1352                         float escala = particulas[i].tamano * factorVida;
1353                         model = glm::scale(model, glm::vec3(escala));
1354
1355                         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1356                         glBindTexture(GL_TEXTURE_2D, textureParticula);
1357                         glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 128.0f);
1358
1359                         ParticulaFuego->Draw();
1360                     }
1361
1362                     // glEnable(GL_DEPTH_TEST); <-- QUITA ESTA LÍNEA TAMBÍEN
1363                     // glDisable(GL_BLEND);
1364
1365                     // Renderizado de objetos opacos
1366                     model = glm::mat4(1.0f);
1367                     glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1368
1369                     // Bind texture y dibujar Arroz
1370                     glActiveTexture(GL_TEXTURE0);
1371                     glBindTexture(GL_TEXTURE_2D, textureArroz);
1372
1373                 }
1374             }
1375
1376             // glDisable(GL_DEPTH_TEST); <-- QUITA ESTA LÍNEA TAMBÍEN
1377             // glEnable(GL_BLEND);
1378
1379             // Bind texture y dibujar Arroz
1380             glActiveTexture(GL_TEXTURE0);
1381             glBindTexture(GL_TEXTURE_2D, textureArroz);
1382
1383         }
1384
1385         // Bind texture y dibujar Arroz
1386         glActiveTexture(GL_TEXTURE0);
1387         glBindTexture(GL_TEXTURE_2D, textureArroz);
1388
1389     }

```

```

1370     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 48.0f);
1371     Arroz.Draw(lightingShader);
1372
1373     // Base
1374     glBindTexture(GL_TEXTURE_2D, textureBase);
1375     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 32.0f);
1376     Base.Draw(lightingShader);
1377
1378     // Barandal
1379     glBindTexture(GL_TEXTURE_2D, textureBarandal);
1380     Barandal.Draw(lightingShader);
1381
1382     // Botes
1383     glBindTexture(GL_TEXTURE_2D, textureBotes);
1384     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 24.0f);
1385     Botes.Draw(lightingShader);
1386
1387     // BolsaHarinaAbierta
1388     glBindTexture(GL_TEXTURE_2D, textureBolsaHarinaAbierta);
1389     BolsaHarinaAbierta.Draw(lightingShader);
1390
1391     // BolsaHarinaAbierta1
1392     glBindTexture(GL_TEXTURE_2D, textureBolsaHarinaAbierta1);
1393     BolsaHarinaAbierta1.Draw(lightingShader);
1394
1395     // BolsasHarina
1396     glBindTexture(GL_TEXTURE_2D, textureBolsasHarina);
1397     BolsasHarina.Draw(lightingShader);
1398
1399     // BoteBasura
1400     glBindTexture(GL_TEXTURE_2D, textureBoteBasura);
1401     BoteBasura.Draw(lightingShader);
1402
1403     // CadenaIntern
1404     glBindTexture(GL_TEXTURE_2D, textureCadenaIntern);
1405     CadenaIntern.Draw(lightingShader);
1406
1407     // Caja
1408     glBindTexture(GL_TEXTURE_2D, textureCaja);
1409     Caja.Draw(lightingShader);
1410
1411     // Cajas2
1412     glBindTexture(GL_TEXTURE_2D, textureCajas2);
1413
1414
1415     // CajasMadera
1416     glBindTexture(GL_TEXTURE_2D, textureCajasMadera);
1417     CajasMadera.Draw(lightingShader);
1418
1419     // Campana
1420     glBindTexture(GL_TEXTURE_2D, textureCampana);
1421     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 56.0f);
1422     Campana.Draw(lightingShader);
1423
1424     // Cortina
1425     glBindTexture(GL_TEXTURE_2D, textureCortina);
1426     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
1427     Cortina.Draw(lightingShader);
1428
1429     // Cortinas
1430     glBindTexture(GL_TEXTURE_2D, textureCortinas);
1431     Cortinas.Draw(lightingShader);
1432
1433     // Cortina2
1434     glBindTexture(GL_TEXTURE_2D, textureCortina2);
1435     Cortina2.Draw(lightingShader);
1436
1437     // Cubeta
1438     glBindTexture(GL_TEXTURE_2D, textureCubeta);
1439     Cubeta.Draw(lightingShader);
1440
1441     // Electricidad
1442     glBindTexture(GL_TEXTURE_2D, textureElectricidad);
1443     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 40.0f);
1444     Electricidad.Draw(lightingShader);
1445
1446     // Escaleras
1447     glBindTexture(GL_TEXTURE_2D, textureEscaleras);
1448     Escaleras.Draw(lightingShader);
1449
1450     // Estufa
1451     glBindTexture(GL_TEXTURE_2D, textureEstufa);
1452     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 64.0f);
1453     Estufa.Draw(lightingShader);
1454
1455
1456     // Fregadero
1457     glBindTexture(GL_TEXTURE_2D, textureFregadero);
1458     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 72.0f);
1459     Fregadero.Draw(lightingShader);
1460
1461     // Gas
1462     glBindTexture(GL_TEXTURE_2D, textureGas);
1463     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 48.0f);
1464     Gas.Draw(lightingShader);
1465
1466     // Jalador
1467     glBindTexture(GL_TEXTURE_2D, textureJalador);
1468     Jalador.Draw(lightingShader);
1469
1470     // LamparasCocina
1471     glBindTexture(GL_TEXTURE_2D, textureLamparasCocina);
1472     LamparasCocina.Draw(lightingShader);
1473
1474     // Librero
1475     glBindTexture(GL_TEXTURE_2D, textureLibrero);
1476     Librero.Draw(lightingShader);
1477
1478     // Mesa
1479     glBindTexture(GL_TEXTURE_2D, textureMesa);
1480     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 40.0f);
1481     Mesa.Draw(lightingShader);
1482
1483     // Olla
1484     glBindTexture(GL_TEXTURE_2D, textureOlla);
1485     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 96.0f);
1486     Olla.Draw(lightingShader);
1487
1488     // Olla1
1489     glBindTexture(GL_TEXTURE_2D, textureOlla1);
1490     Olla1.Draw(lightingShader);
1491
1492     // Olla2
1493     glBindTexture(GL_TEXTURE_2D, textureOlla2);
1494     Olla2.Draw(lightingShader);
1495
1496     // Olla3
1497     glBindTexture(GL_TEXTURE_2D, textureOlla3);

```

```

1498     Olla3.Draw(lightingShader);
1499
1500     // Pasto
1501     glBindTexture(GL_TEXTURE_2D, texturePasto);
1502     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 8.0f);
1503     Pasto.Draw(lightingShader);
1504
1505     // Pared3
1506     glBindTexture(GL_TEXTURE_2D, texturePared3);
1507     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 24.0f);
1508     Pared3.Draw(lightingShader);
1509
1510     // Pared4
1511     glBindTexture(GL_TEXTURE_2D, texturePared4);
1512     Pared4.Draw(lightingShader);
1513
1514     // Pared
1515     glBindTexture(GL_TEXTURE_2D, texturePared);
1516     Pared.Draw(lightingShader);
1517
1518     // Platos
1519     glBindTexture(GL_TEXTURE_2D, texturePlatos);
1520     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 56.0f);
1521     Platos.Draw(lightingShader);
1522
1523     // PrimerPiso
1524     glBindTexture(GL_TEXTURE_2D, texturePrimerPiso);
1525     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 32.0f);
1526     PrimerPiso.Draw(lightingShader);
1527
1528     // Piso2
1529     glBindTexture(GL_TEXTURE_2D, texturePiso2);
1530     Piso2.Draw(lightingShader);
1531
1532     // Puertas
1533     glBindTexture(GL_TEXTURE_2D, texturePuertas);
1534     Puertas.Draw(lightingShader);
1535
1536     // Refrigerador
1537     glBindTexture(GL_TEXTURE_2D, textureRefrigerador);
1538     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 80.0f);
1539     Refrigerador.Draw(lightingShader);
1540
1541
1542     // Silla2
1543     glBindTexture(GL_TEXTURE_2D, textureSilla2);
1544     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 28.0f);
1545     Silla2.Draw(lightingShader);
1546
1547     // Silla
1548     glBindTexture(GL_TEXTURE_2D, textureSilla);
1549     Silla.Draw(lightingShader);
1550
1551     // Tapa
1552     glBindTexture(GL_TEXTURE_2D, textureTapa);
1553     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 48.0f);
1554     Tapa.Draw(lightingShader);
1555
1556     // Tapal
1557     glBindTexture(GL_TEXTURE_2D, textureTapal);
1558     Tapal.Draw(lightingShader);
1559
1560     // Tapa2
1561     glBindTexture(GL_TEXTURE_2D, textureTapa2);
1562     Tapa2.Draw(lightingShader);
1563
1564     // TapeteAlmacen
1565     glBindTexture(GL_TEXTURE_2D, textureTapeteAlmacen);
1566     TapeteAlmacen.Draw(lightingShader);
1567
1568     // TanqueAgua
1569     glBindTexture(GL_TEXTURE_2D, textureTanqueAgua);
1570     TanqueAgua.Draw(lightingShader);
1571
1572     // TapeteCocina
1573     glBindTexture(GL_TEXTURE_2D, textureTapeteCocina);
1574     TapeteCocina.Draw(lightingShader);
1575
1576     // Tazon
1577     glBindTexture(GL_TEXTURE_2D, textureTazon);
1578     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 64.0f);
1579     Tazon.Draw(lightingShader);
1580
1581     // Tazones2
1582     glBindTexture(GL_TEXTURE_2D, textureTazones2);
1583     Tazones2.Draw(lightingShader);
1584
1585     // Tejado
1586     glBindTexture(GL_TEXTURE_2D, textureTejado);
1587     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
1588     Tejado.Draw(lightingShader);
1589
1590     // TejadoMini
1591     glBindTexture(GL_TEXTURE_2D, textureTejadoMini);
1592     TejadoMini.Draw(lightingShader);
1593
1594     // TejadoMiniPiso
1595     glBindTexture(GL_TEXTURE_2D, textureTejadoMiniPiso2);
1596     TejadoMiniPiso2.Draw(lightingShader);
1597
1598     // TuboFueras
1599     glBindTexture(GL_TEXTURE_2D, textureTuboFueras);
1600     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 80.0f);
1601     TuboFueras.Draw(lightingShader);
1602
1603     // TuboGas
1604     glBindTexture(GL_TEXTURE_2D, textureTuboGas);
1605     TuboGas.Draw(lightingShader);
1606
1607     // VentanaSegundoPiso
1608     glBindTexture(GL_TEXTURE_2D, textureVentanaSegundoPiso);
1609     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 24.0f);
1610     VentanaSegundoPiso.Draw(lightingShader);
1611
1612     // ===== BANDERINES CON CONTROL MANUAL (ADELANTE/ATRÁS) =====
1613     model = glm::mat4(1.0f);
1614
1615     // La rotación se aplica en el eje X para mover la punta hacia adelante/atrás
1616     model = glm::rotate(model, glm::radians(banderinesInclinacion), glm::vec3(1.0f, 0.0f, 0.0f));
1617
1618     glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1619     glBindTexture(GL_TEXTURE_2D, textureBanderines);
1620     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
1621     Banderines.Draw(lightingShader);
1622
1623     // ===== RELOJ =====
1624     model = glm::mat4(1.0f);
1625     model = glm::translate(model, posReloj);
1626     glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1627     glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 32.0f);

```

```

1627 |     RelojCirculo->Draw();
1628 | 
1629 |     // Manecilla de hora (gira lento)
1630 |     float horaAngulo = (float)glfGetTime() * 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)glfGetTime() * 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 (CASA) =====
1644 |     model = glm::mat4(1.0f);
1645 |     model = glm::translate(model, posCasa);
1646 |     glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1647 |     glUniform1f(glfGetUniformLocation(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(glfGetUniformLocation(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();
1670 | 
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(glfGetUniformLocation(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(2.0f, 1.0f, 2.0f));
1694 |     glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1695 |     glUniform1f(glfGetUniformLocation(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(glfGetUniformLocation(lightingShader.Program, "material.shininess"), 20.0f);
1704 |         Narutomaki->Draw();
1705 |     }
1706 | 
1707 |     // ===== CUCHILLO (HOJA + MANGO) =====
1708 |     glm::vec3 posCuchillo = glm::vec3(-1.80f, 1.20f, 1.5f);
1709 | 
1710 |     // Hoja del cuchillo (rotada 45 grados)
1711 |     model = glm::mat4(1.0f);
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 | 
1715 |     // ===== HOJA DEL CUCHILLO =====
1716 |     HojaCuchillo->Draw();
1717 | 
1718 |     // Mango del cuchillo (atrás de la hoja)
1719 |     model = glm::translate(model, glm::vec3(0.0f, 0.0f, -0.25f));
1720 |     glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1721 |     glUniform1f(glfGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
1722 |     MangoCuchillo->Draw();
1723 | 
1724 |     // ===== GOTAS DE AGUA =====
1725 |     if (agujaCayendo) {
1726 |         glEnable(GL_BLEND);
1727 |         glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
1728 | 
1729 |         for (int i = 0; i < MAX_GOTAS; i++) {
1730 |             if (gotas[i].activa) {
1731 |                 model = glm::mat4(1.0f);
1732 |                 model = glm::translate(model, gotas[i].posicion);
1733 |                 glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
1734 |                 glBindTexture(GL_TEXTURE_2D, textureAgua);
1735 |                 glUniform1f(glfGetUniformLocation(lightingShader.Program, "material.shininess"), 128.0f);
1736 |                 GotaAgua->Draw();
1737 |             }
1738 |         }
1739 |         glDisable(GL_BLEND);
1740 |     }
1741 | 
1742 |     // Animación del refrigerador
1743 |     glm::vec3 posBisagraRefrigerador = glm::vec3(2.5f, 1.0f, -3.00f);
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, textureRefrigerador);
1750 |     glUniform1f(glfGetUniformLocation(lightingShader.Program, "material.shininess"), 80.0f);
1751 |     Refrigerador.Draw(lightingShader);
1752 | 
1753 |     // Objeto transparente (lámparas)
1754 |     glEnable(GL_BLEND);
1755 |     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     // LámparasCocina1
1764     glBindTexture(GL_TEXTURE_2D, textureLámparasCocina1);
1765     LámparasCocina1.Draw(lightingShader);
1766
1767     // LinternasLocal
1768     glBindTexture(GL_TEXTURE_2D, textureLinternaLocal);
1769     LinternasLocal.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 RelojCírculo;
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.

```

1799 // Movimiento de cámara
1800 if (keys[GLFW_KEY_W] || keys[GLFW_KEY_UP])
1801     camera_ProcessKeyboard(FORWARD, deltaTime);
1802 if (keys[GLFW_KEY_S] || keys[GLFW_KEY_DOWN])
1803     camera_ProcessKeyboard(BACKWARD, deltaTime);
1804 if (keys[GLFW_KEY_A] || keys[GLFW_KEY_LEFT])
1805     camera_ProcessKeyboard(LEFT, deltaTime);
1806 if (keys[GLFW_KEY_D] || keys[GLFW_KEY_RIGHT])
1807     camera_ProcessKeyboard(RIGHT, deltaTime);
1808
1809 // Animación automática de la puerta
1810 if (animandoPuerta) {
1811     if (abs(puertaRefrigerador - puertaDestino) > 0.1f) {
1812         if (puertaRefrigerador < puertaDestino)
1813             puertaRefrigerador += 80.0f * deltaTime;
1814         else
1815             puertaRefrigerador -= 80.0f * deltaTime;
1816     } else {
1817         puertaRefrigerador = puertaDestino;
1818         animandoPuerta = false;
1819     }
1820 }
1821
1822
1823 // **ANIMACIÓN: Banderines moviéndose hacia adelante/atrás**
1824 if (animandoBanderines) {
1825     if (abs(banderinesInclinacion - banderinesDestino) > 0.1f) {
1826         if (banderinesInclinacion < banderinesDestino)
1827             banderinesInclinacion += 60.0f * deltaTime;
1828         else
1829             banderinesInclinacion -= 60.0f * deltaTime;
1830     } else {
1831         banderinesInclinacion = banderinesDestino;
1832         animandoBanderines = false;
1833     }
1834 }
1835
1836
1837 // **ANIMACIÓN: Agua cayendo del grifo**
1838 if (aguaCayendo) {
1839     // Crear nuevas gotas
1840     if (currentFrame - tiempoUltimagota > intervaloGotas) {
1841         for (int i = 0; i < MAX_GOTAS; i++) {
1842             if (gotas[i].activa) {
1843                 gotas[i].activa = true;
1844                 gotas[i].posicion = posicionGrifo;
1845                 gotas[i].velocidad = 2.0f + (rand() % 100) / 100.0f;
1846                 tiempoUltimagota = currentFrame;
1847                 break;
1848             }
1849         }
1850     }
1851
1852     // Actualizar posición de gotas activas
1853     for (int i = 0; i < MAX_GOTAS; i++) {
1854         if (gotas[i].activa) {
1855             gotas[i].posicion.y -= gotas[i].velocidad * deltaTime;
1856             gotas[i].velocidad += 3.0f * deltaTime; // Gravedad
1857
1858             // Desactivar gota si llega al fregadero
1859             if (gotas[i].posicion.y < 1.0f) {
1860                 gotas[i].activa = false;
1861             }
1862         }
1863     }
1864 }
1865 // **ANIMACIÓN: Fuegos artificiales**
1866 if (fuegosActivos) {
1867     if (currentFrame - tiempoUltimoFuego >= intervaloFuegos) {
1868         // MÁS SEPARADAS HORIZONTALMENTE (de -8 a +8)
1869         glm::vec3 posExplosion = glm::vec3(
1870             -8.0f + (rand() % 160) / 100.0f, // -8 a 8 en X (MUY ANCHO)
1871             7.0f + (rand() % 200) / 100.0f, // 7 a 9 en Y (altura similar)
1872             -10.0f // Z FIJO (sin profundidad)
1873         );
1874
1875         // Colores vivos y variados
1876         int tipoColor = rand() % 5;
1877         glm::vec3 colorExplosion;
1878
1879         switch (tipoColor) {
1880             case 0: colorExplosion = glm::vec3(1.0f, 0.2f, 0.2f); break; // Rojo
1881             case 1: colorExplosion = glm::vec3(0.2f, 1.0f, 0.2f); break; // Verde
1882             case 2: colorExplosion = glm::vec3(0.2f, 0.2f, 1.0f); break; // Azul
1883             case 3: colorExplosion = glm::vec3(1.0f, 1.0f, 0.2f); break; // Amarillo
1884             case 4: colorExplosion = glm::vec3(1.0f, 0.2f, 1.0f); break; // Magenta
1885         }
1886
1887         CrearExplosion(posExplosion, colorExplosion);
1888         tiempoUltimoFuego = currentFrame;
1889
1890         std::cout << "BOOM! Explosión en el fondo Z=" << posExplosion.z << std::endl;
1891     }
1892
1893     // Actualizar partículas activas
1894     for (int i = 0; i < MAX_PARTICULAS; i++) {
1895         if (particulas[i].activa) {
1896             // Actualizar posición
1897             particulas[i].posicion += particulas[i].velocidad * deltaTime;
1898
1899             // Aplicar gravedad suave
1900             particulas[i].velocidad.y -= 2.5f * deltaTime;
1901
1902             // Fricción del aire
1903             particulas[i].velocidad *= 0.98f;
1904
1905             // Reducir vida
1906             particulas[i].vida -= deltaTime;
1907
1908             // Desactivar si ya no tiene vida
1909             if (particulas[i].vida <= 0.0f) {
1910                 particulas[i].activa = false;
1911             }
1912         }
1913     }
1914 }
1915
1916 void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode)
1917 {
1918     (void)scancode;
1919     (void)mode;
1920
1921     if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
1922         glfwSetWindowShouldClose(window, GL_TRUE);
1923
1924     if (key >= 0 && key < 1024)
1925     {
1926         if (action == GLFW_PRESS)
1927             keys[key] = true;
1928     }
1929 }

```

```

1928     else if (action == GLFW_RELEASE)
1929     {
1930         keys[key] = false;
1931     }
1932     if (action == GLFW_PRESS)
1933     {
1934         switch (key)
1935         {
1936             case GLFW_KEY_1:
1937                 lightStates[0] = !lightStates[0];
1938                 std::cout << "luz 1 (LamparasCocinal): " << (lightStates[0] ? "ENCENDIDA" : "APAGADA") << std::endl;
1939                 break;
1940             case GLFW_KEY_2:
1941                 lightStates[1] = !lightStates[1];
1942                 std::cout << "luz 2 (Lamparas): " << (lightStates[1] ? "ENCENDIDA" : "APAGADA") << std::endl;
1943                 break;
1944             case GLFW_KEY_3:
1945                 lightStates[2] = !lightStates[2];
1946                 std::cout << "luz 3 (LinternalLocal): " << (lightStates[2] ? "ENCENDIDA" : "APAGADA") << std::endl;
1947                 break;
1948             case GLFW_KEY_4:
1949                 lightStates[3] = !lightStates[3];
1950                 std::cout << "luz Luna: " << (lightStates[3] ? "ENCENDIDA" : "APAGADA") << std::endl;
1951                 break;
1952             case GLFW_KEY_O:
1953                 puertaDestino = 90.0f; // Abrir
1954                 animandoPuerta = true;
1955                 std::cout << "Abriendo puerta del refrigerador..." << std::endl;
1956                 break;
1957             case GLFW_KEY_C:
1958                 puertaDestino = 0.0f; // Cerrar
1959                 animandoPuerta = true;
1960                 std::cout << "Cerrando puerta del refrigerador..." << std::endl;
1961                 break;
1962             case GLFW_KEY_N: // Mover banderines hacia ATRÁS
1963                 banderinesDestino = -banderinesMaxInclinacion;
1964                 animandoBanderines = true;
1965                 std::cout << "Banderines moviéndose hacia atrás..." << std::endl;
1966                 break;
1967             case GLFW_KEY_M: // Regresar banderines a posición NORMAL
1968                 banderinesDestino = 0.0f;
1969                 animandoBanderines = true;
1970
1971             std::cout << "Banderines regresando a posición normal..." << std::endl;
1972             break;
1973         case GLFW_KEY_G: // Tapon agua del grifo
1974             aguaCayendo = !aguaCayendo;
1975             std::cout << "Agua del grifo: " << (aguaCayendo ? "ABIERTA" : "CERRADA") << std::endl;
1976             if (!aguaCayendo)
1977             {
1978                 for (int i = 0; i < MAX_GOTAS; i++)
1979                 {
1980                     gotas[i].activa = false;
1981                 }
1982             }
1983             break;
1984         case GLFW_KEY_F: // Toggle Fuegos artificiales
1985             fuegosActivos = !fuegosActivos;
1986             std::cout << "Fuegos artificiales: " << (fuegosActivos ? "ACTIVADOS" : "DESACTIVADOS") << std::endl;
1987             if (!fuegosActivos)
1988             {
1989                 for (int i = 0; i < MAX_PARTICULAS; i++)
1990                 {
1991                     particulas[i].activa = false;
1992                 }
1993             }
1994         }
1995     }
1996     void MouseCallback(GLFWwindow* window, double xpos, double ypos)
1997     {
1998         (void)window;
1999
2000         if (firstMouse)
2001         {
2002             lastX = (GLfloat)xpos;
2003             lastY = (GLfloat)ypos;
2004             firstMouse = false;
2005         }
2006
2007         GLfloat xOffset = (GLfloat)(xpos - lastX);
2008         GLfloat yOffset = (GLfloat)(lastY - ypos);
2009
2010         lastX = (GLfloat)xpos;
2011         lastY = (GLfloat)ypos;
2012
2013         camera.ProcessMouseMovement(xOffset, yOffset);

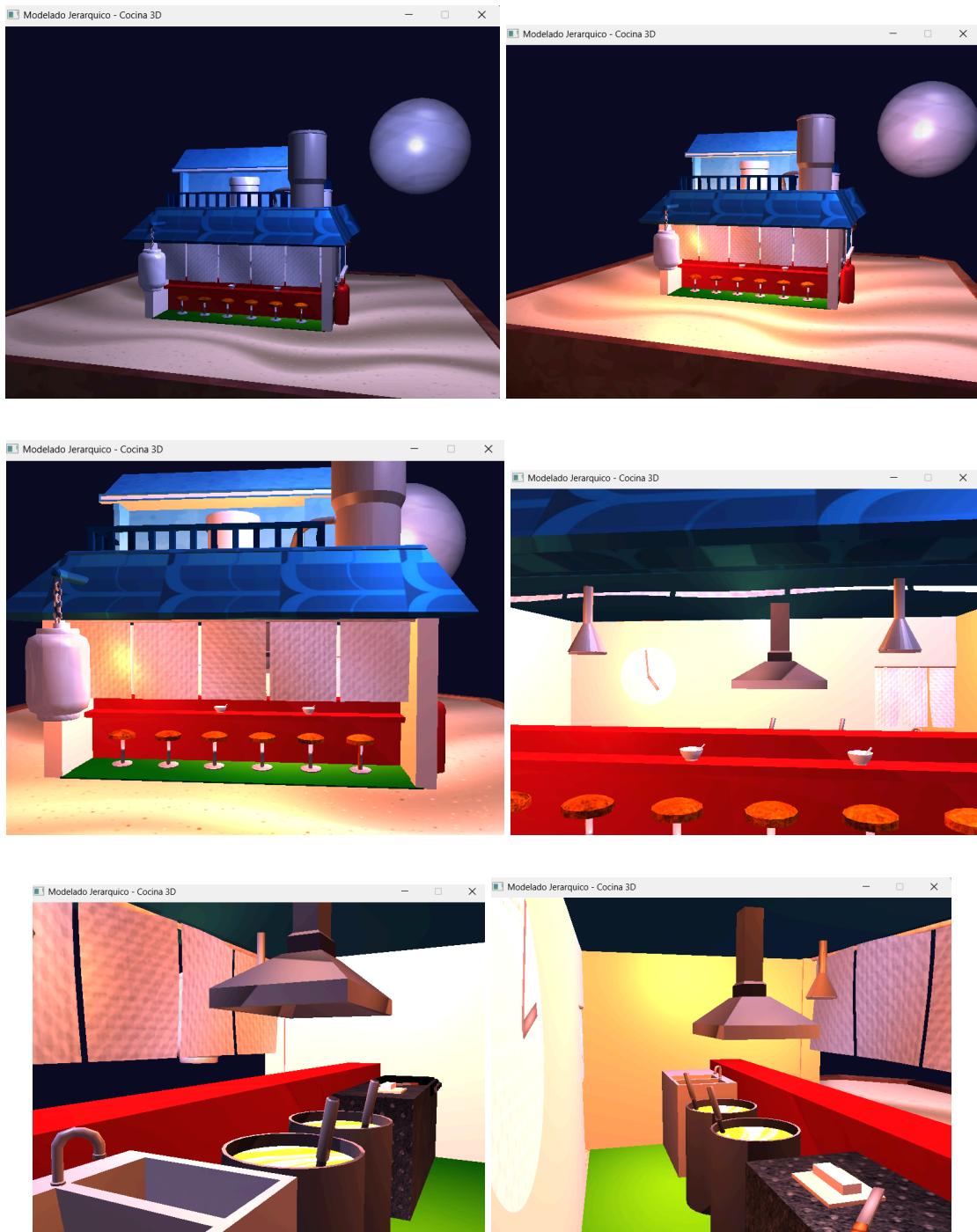
```

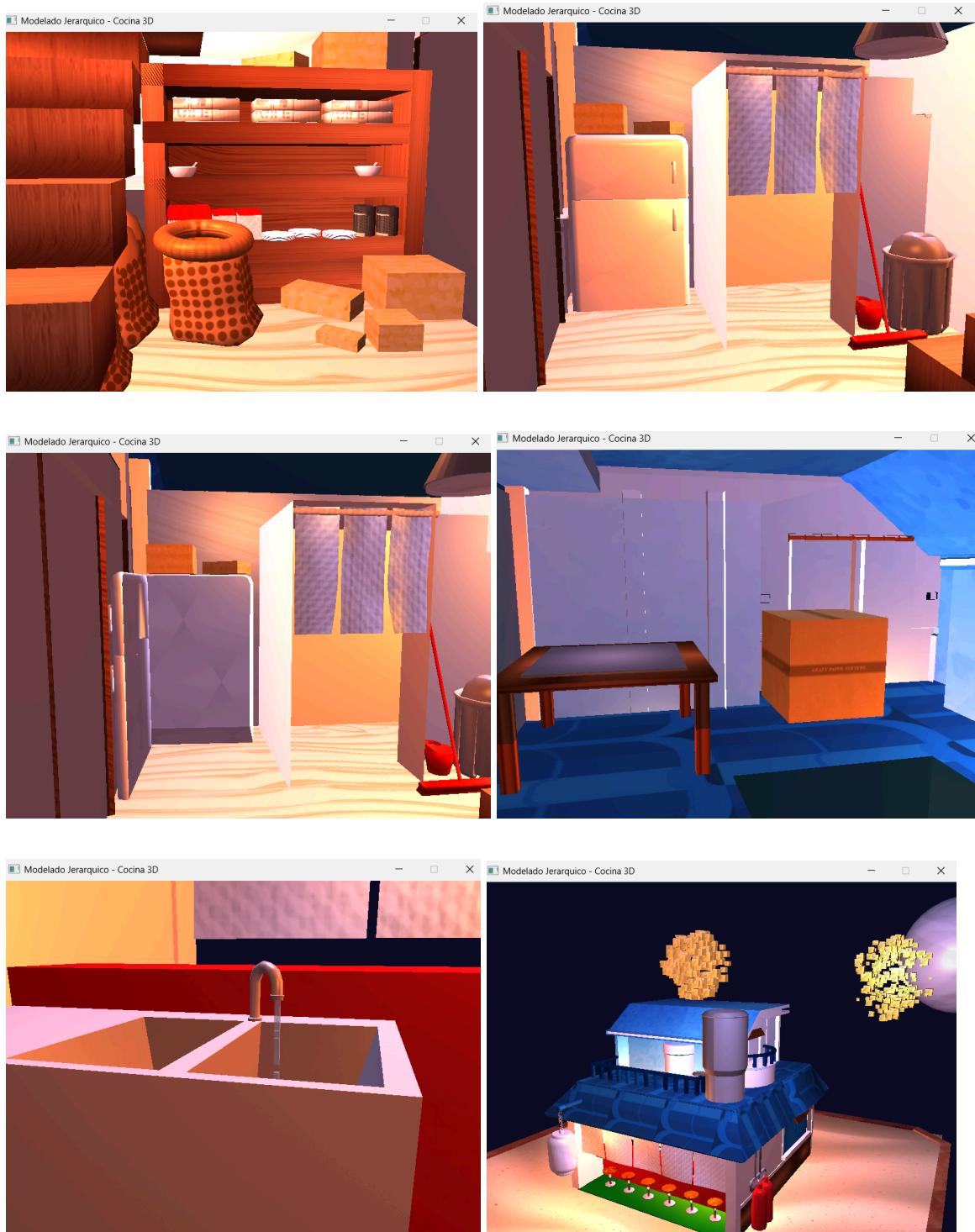
Results

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.





- **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>