



Computación Gráfica e
Interacción Humano-Computadora

UNIVERSIDAD DE MÉXICO
FACULTAD DE INGENIERÍA

Proyecto Final: Manual Técnico/Technical Manual

Palafox Jimenez Raúl

422132844

Semestre 2026-1

Grupo: 05

Profesor Carlos Aldair Roman Balbuena

Fecha de entrega: 24 de Noviembre de 2025

Semestre 2026-1

1. Introducción

1.1 Propósito del Documento

Este manual técnico proporciona una descripción detallada de la arquitectura, implementación y funcionamiento del sistema de visualización 3D interactiva del café "Leblanc" basado en Persona 5. El documento está dirigido a desarrolladores, ingenieros de software y personal técnico que requieran comprender, mantener o extender las capacidades del sistema.

1.2 Alcance del Sistema

El sistema implementa un recorrido virtual en tiempo real del café "Leblanc" de dos plantas, incluyendo renderizado 3D con OpenGL 3.3+, sistema de iluminación multicapa con 10 luces puntuales, luz direccional y spotlight dinámico, cuatro animaciones interactivas controladas por teclado (puerta, letrero, ventanas, cortinas), sistema de cámara first-person sin colisiones, y gestión de materiales transparentes mediante alpha blending.

1.3 Stack Tecnológico

Componente	Tecnología	Versión	Propósito
Lenguaje	C++	17	Desarrollo principal del motor gráfico
API Gráfica	OpenGL	3.3+	Renderizado 3D acelerado por hardware
Shaders	GLSL	-	Iluminación Phong y efectos visuales
Ventanas	GLFW	3.x	Gestión de contexto OpenGL y eventos
Matemáticas	GLM	0.9.9+	Álgebra lineal para transformaciones 3D
Modelado 3D	Blender	4.5	Creación de todos los assets 3D
Formatos	OBJ	-	Exportación optimizada de modelos

2. Estado del Arte: Recorridos Virtuales Interactivos

2.1 Contexto Actual

Los recorridos virtuales de locaciones de videojuegos han ganado popularidad significativa en la comunidad gamer, permitiendo a los fans experimentar espacios icónicos de manera inmersiva. Proyectos como recreaciones de "Skyrim", "The Last of Us" y "Resident Evil" demuestran el interés creciente por explorar entornos virtuales beyond del gameplay tradicional. Esta tendencia combina nostalgia gaming con tecnología de visualización 3D accesible.

2.2 Tecnologías Comparativas

Motores de Videojuegos (Unity/Unreal):

- **Ventajas:** Desarrollo rápido, herramientas visuales, ecosistema extenso
- **Desventajas:** Ejecutables pesados, menor control sobre optimización, licencias costosas para distribución comercial
- **Uso típico:** Proyectos comerciales con presupuestos > \$20,000 MXN

Soluciones WebGL (Three.js/Babylon.js):

- **Ventajas:** Acceso instantáneo via navegador, multiplataforma inherente
- **Desventajas:** Rendimiento limitado, gráficos de menor calidad, dependencia de conexión
- **Uso típico:** Demos web, prototipos rápidos

Tour 360° Fotográfico:

- **Ventajas:** Relativo realismo, captura rápida
- **Desventajas:** Sin interactividad real, navegación limitada a puntos fijos
- **Uso típico:** Inmobiliario, bienes raíces

2.3 Posicionamiento del Proyecto

Este proyecto se posiciona como una solución educativa-technical que prioriza el **control total sobre el pipeline gráfico** y la **optimización específica** para hardware de gama media. Es ideal para:

- **Estudiantes y desarrolladores** que buscan entender OpenGL moderno
- **Fans de Persona 5** que desean experimentar el café Leblanc de manera inmersiva
- **Proyectos académicos** que requieren implementaciones gráficas desde cero
- **Casos donde la propiedad intelectual completa** es esencial (sin dependencias de motores comerciales)

La elección de OpenGL nativo sobre Unity/Unreal permite un ejecutable ligero (~50MB vs 200MB+), máximo control sobre optimización, y código portable entre Windows/Linux/macOS.

4. Metodología de Desarrollo

4.1 Enfoque Iterativo Minimalista

El desarrollo siguió una estrategia de "**mínimo producto viable gráfico**" con iteraciones centradas en funcionalidad básica antes de características avanzadas. Este enfoque priorizó:

- **Renderizado básico** → Texturas → Iluminación → Animaciones
- **Cada iteración entregaba un ejecutable funcional**
- **Validación temprana** de performance en hardware objetivo

4. Metodología de Desarrollo

4.1 Enfoque Iterativo por Fases Específicas

El desarrollo siguió una estrategia de planificación rigurosa con hitos definidos, priorizando la aprobación conceptual inicial y el desarrollo en paralelo de componentes técnicos y artísticos. La metodología se estructuró en 8 fases secuenciales con solapamiento estratégico.

4.2 Fases del Proyecto

Fase 1: Planificación (10 días - 1 al 10 Octubre 2025)

- **Actividades principales:** Recopilación de imágenes de referencia del café Leblanc de Persona 5, creación del pitch conceptual, investigación de assets y restricciones temáticas
- **Entregable:** Documento de aprobación conceptual revisado por el profesor
- **Consideración especial:** Análisis de casos no aprobados previamente (Simpsons, Kame House, Chicas Superpoderosas) para asegurar viabilidad temática

Fase 2: Modelado 3D (15 días - 11 al 25 Octubre 2025)

- **Actividades principales:** Creación de todos los modelos 3D en Blender: estructura arquitectónica, mobiliario, objetos decorativos y elementos interactivos
- **Técnicas aplicadas:** Modelado low-poly, UV mapping básico, optimización de geometría
- **Entregable:** Biblioteca completa de modelos .obj con pivotes configurados

Fase 3: Desarrollo (15 días - 14 al 28 Octubre 2025) - Paralelo

- **Actividades principales:** Programación en Visual Studio 2022 del motor gráfico base, sistema de carga de modelos, shaders de iluminación, controles de cámara first-person
- **Tecnologías:** C++, OpenGL 3.3+, GLFW, GLM
- **Característica:** Ejecución en paralelo con el modelado 3D para optimizar tiempo

Fase 4: Animaciones (8 días - 29 Octubre al 5 Noviembre 2025)

- **Actividades principales:** Implementación de las 4 animaciones interactivas (puerta, letrero, ventanas, cortinas) mediante transformaciones matriciales
- **Mecánicas:** Sistemas de toggle, interpolaciones, máquinas de estado simples
- **Integración:** Conexión con controles de teclado (R, P, Q, N)

Fase 5: Efectos Visuales (5 días - 6 al 10 Noviembre 2025)

- **Actividades principales:** Implementación de sistema de transparencias (alpha blending) para vidrios, configuración de materiales opacos y translúcidos, ajuste final de iluminación
- **Técnicas:** Depth masking, blending functions, optimización de render order

Fase 6: Testing (7 días - 11 al 17 Noviembre 2025)

- **Actividades principales:** Pruebas de rendimiento, verificación de animaciones, detección de bugs visuales, validación de controles de usuario
- **Enfoque:** Testing funcional y de experiencia de usuario sin sistema de colisiones

Fase 7: Documentación (6 días - 18 al 23 Noviembre 2025)

- **Actividades principales:** Elaboración del Manual de Usuario (completado) y Manual Técnico (en desarrollo)
- **Alcance:** Guías de instalación, controles, solución de problemas, arquitectura técnica

Fase 8: Entrega (2 días - 24 al 25 Noviembre 2025)

- **Actividades principales:** Depuración final, empaquetado de ejecutable, subida a GitHub, preparación de demostración
- **Culminación:** Entrega formal del proyecto completo en classroom con todos sus debidos documentos

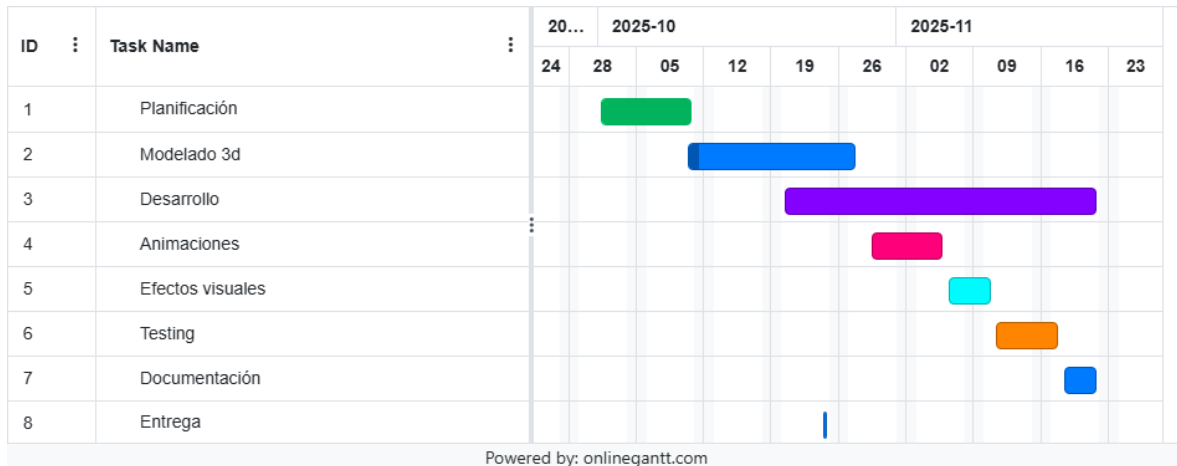


Figura 1. Diagrama de Gantt correspondiente a la planeación del proyecto

5. Arquitectura y Especificaciones

5.1 Arquitectura del Sistema

El sistema está estructurado en cinco capas fundamentales que gestionan desde la interacción básica con el hardware hasta la presentación visual final:

- **Capa de Plataforma (GLFW):** Gestiona la creación de ventanas, contexto OpenGL y captura de eventos de teclado y mouse. Proporciona una abstracción multiplataforma para las operaciones del sistema.
- **Capa de Renderizado (OpenGL Core):** Maneja el pipeline gráfico programable mediante shaders GLSL, gestión de buffers de vértices (VBO, VAO, EBO) y control del estado de renderizado. Implementa las operaciones de dibujo de geometría 3D.
- **Capa de Recursos (Model/Texture/Shader):** Se encarga de la carga de modelos 3D mediante Assimp, decodificación de texturas con SOIL2/stb_image, y compilación/vinculación de programas de shaders GLSL.
- **Capa de Lógica de Aplicación:** Actualiza las transformaciones por frame, ejecuta máquinas de estado para las animaciones interactivas y gestiona el sistema de cámara en primera persona. Controla la transición entre estados del programa.
- **Capa de Presentación:** Sincroniza el delta time entre frames, controla el orden de renderizado (objetos opacos → transparentes) y ejecuta el intercambio de buffers para visualización.

5.2 Flujo de Ejecución Principal

El bucle principal ejecuta por frame (aproximadamente 16.67ms para 60 FPS) la siguiente secuencia:

1. **Cálculo de Delta Time:** Determinación del tiempo transcurrido desde el frame anterior para movimiento independiente del framerate.
2. **Procesamiento de Eventos:** Captura de entrada de usuario mediante callbacks de GLFW para teclado y mouse.
3. **Actualización de Lógica:** Ejecución de DoMovement() para procesar movimiento de cámara y actualización de animaciones basadas en deltaTime.

4. **Limpieza de Buffers:** Reset de buffers de color y profundidad para el nuevo frame.
5. **Configuración de Shaders:** Activación del shader principal y configuración de uniforms para iluminación y transformaciones.
6. **Renderizado de Geometría Opaca:** Dibujado de aproximadamente 30 modelos que no requieren transparencia.
7. **Activación de Blending:** Configuración de funciones de mezcla alpha para objetos transparentes.
8. **Renderizado de Geometría Transparente:** Dibujado de vidrios y superficies translúcidas con depth write desactivado.
9. **Intercambio de Buffers:** Presentación del frame completado al usuario.
10. **Sincronización:** Control opcional de V-Sync según configuración del driver.

5.3 Programas de Shaders Utilizados

El proyecto emplea tres programas de shaders especializados:

- **lightingShader:** Shader principal que implementa iluminación Phong con múltiples fuentes de luz. Uniforms: model, view, projection, dirLight, pointLights[10], material.
- **lampShader:** Shader para visualización de fuentes de luz visibles como objetos en la escena. Uniforms: model, view, projection.
- **Shader de Transparencias:** Versión modificada del lightingShader con soporte para alpha blending en superficies translúcidas.

5.4 Configuración Técnica

Ventana de Renderizado:

- Resolución base: 800×600 píxeles (compatible con hardware desde 2005)
- Modo: Ventana redimensionable
- Buffer de profundidad: 24 bits
- Buffer de color: RGBA 8888 (16.7 millones de colores)
- Sincronización vertical: Controlada por driver

Cámara Sintética:

- Posición inicial: (0, 0, 3) exterior del café Leblanc
- Campo de visión: 45-90° ajustable mediante scroll del mouse
- Plano cercano: 0.1 unidades (evita z-fighting)
- Plano lejano: 100.0 unidades (incluye escena completa)
- Velocidad de movimiento: 2.5 unidades/segundo
- Sensibilidad de mouse: 0.1 para rotación suave
- Límite de pitch: $\pm 89^\circ$ (evita gimbal lock)

Sistema de Coordenadas:

- Convención: Right-handed de OpenGL
- Eje X: Derecha
- Eje Y: Arriba
- Eje Z: Hacia el observador

5.3 Sistema de Iluminación

5.3.1 Filosofía de Diseño Lumínico

El sistema de iluminación fue diseñado para reforzar la narrativa ambiental del Café Leblanc, creando una atmósfera que refleja las condiciones descritas en el videojuego Persona 5. La distribución y cualidad de las luces comunica visualmente la naturaleza del espacio: acogedor pero modesto, con claras diferencias entre la zona pública del café y la habitación personal del protagonista.

5.3.2 Configuración de Luces Puntuales

Las 10 luces puntuales se distribuyen estratégicamente para crear contraste ambiental:

Luces 1-4: Iluminación Principal del Café (Cálida y Acogedora)

- **Ubicación:** Lámparas colgantes sobre el mostrador y mesas principales
- **Color:** RGB(1.0, 0.85, 0.75) - tonalidad amarilla cálida halógena
- **Intensidad:** Ambient (0.15), Diffuse (0.6), Specular (0.8)
- **Atenuación:** Constante: 1.0, Linear: 0.09, Quadratic: 0.032
- **Propósito:** Crear ambiente acogedor típico de cafeterías tradicionales japonesas

Luces 5-7: Elementos Decorativos con Vidrios Coloreados

- **Ubicación:** Lámparas con pantallas de vidrio tintado en el área del café
- **Colores:**
 - Luz 5: RGB(0.8, 0.9, 1.0) - azul claro translúcido
 - Luz 6: RGB(1.0, 0.9, 0.8) - ámbar suave
 - Luz 7: RGB(0.9, 1.0, 0.9) - verde pálido
- **Intensidad:** Ambient (0.08), Diffuse (0.4), Specular (0.5)
- **Atenuación:** Constante: 1.0, Linear: 0.14, Quadratic: 0.07
- **Propósito:** Añadir carácter visual sin competir con la iluminación principal

Luces 8-10: Iluminación de la Habitación Superior (Pobre y Funcional)

- **Ubicación:** Focos económicos en la habitación del protagonista
- **Color:** RGB(0.9, 0.9, 1.0) - blanco frío con tinte azulado
- **Intensidad:** Ambient (0.05), Diffuse (0.3), Specular (0.4)
- **Atenuación:** Constante: 1.0, Linear: 0.22, Quadratic: 0.20

- **Propósito:** Simular focos baratos y transmitir las condiciones modestas de la habitación, reforzando la queja canónica sobre la filtración de luz desde el piso inferior

5.3.3 Luz Direccional (Simulación Lunar)

Parámetros:

- **Dirección:** (-0.2, -0.5, -0.3) - ángulo bajo nocturno
- **Color:** RGB(0.3, 0.3, 0.5) - azul grisáceo lunar
- **Intensidad:** Ambient (0.02), Diffuse (0.1), Specular (0.05)
- **Propósito:** Proveer iluminación ambiental base que simule luz lunar filtrándose por las ventanas, creando un contexto nocturno coherente

5.3.4 Spotlight Asociado a Cámara

Características:

- **Posición:** Vinculada dinámicamente a la posición de la cámara
- **Dirección:** Alineada con el vector forward de la cámara
- **Color:** RGB(0.8, 0.8, 0.7) - blanco neutro
- **Ángulos:** Interior 12.5°, Exterior 17.5°
- **Intensidad:** Ambient (0.0), Diffuse (0.5), Specular (0.6)
- **Propósito:** Servir como "linterna" del usuario, permitiendo examinar detalles en áreas oscuras sin alterar la atmósfera general

5.3.5 Consideraciones de Implementación

Eficiencia de Rendimiento:

- Todas las luces utilizan atenuación cuadrática para transiciones suaves
- Las luces de la habitación superior tienen mayor atenuación para simular su pobre alcance
- El sistema mantiene 60+ FPS en hardware GTX 1050/RX 560 mediante cálculo optimizado en shaders

Coherencia Narrativa:

- El contraste entre la iluminación cálida del café y la fría de la habitación refuerza la dicotomía espacial
- La menor intensidad de las luces 8-10 comunica visualmente las condiciones económicas del personaje
- La elección de colores en luces 5-7 refleja elementos decorativos típicos de establecimientos tradicionales

5.4 Sistema de Materiales

5.4.1 Clasificación por Transparencia

Materiales Opacos (Alpha = 1.0):

- **Fachada, Piso, Toldo:** Estructura principal del edificio
- **Escritorios, Estantería, Mesas:** Mobiliario de madera y metal
- **Sillones, Sillas:** Mobiliario tapizado y de madera

- **Estufa, Máquina de Café:** Electrodomésticos metálicos
- **Cama, Mostrador:** Muebles estructurales
- **Campana, Badajo, Sujetador:** Componentes metálicos de la campana
- **Letreros (OPEN/CLOSED):** Superficies sólidas de señalización
- **Frascos, Lámparas (estructura):** Componentes base sin transparencia

Materiales Translúcidos (Alpha = 0.5):

- **Vidrio de Puerta:** Panel principal de la entrada con animación
- **Vidrio de Fachada:** Ventanas fijas de la estructura
- **Vidrio de Lámparas:** Pantallas coloreadas de las lámparas decorativas

5.4.2 Implementación Técnica de Transparencias

Configuración de Blending:

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glDepthMask(GL_FALSE);
```

Parámetros de Transparencia:

- **Alpha Value:** 0.5 (50% de transparencia uniforme)
- **Función de Mezcla:** GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA
- **Depth Mask:** Desactivado durante renderizado de transparentes

Uniforms en Shader:

```
cpp
glUniform1i(glGetUniformLocation(lightningShader.Program, "transparency"), 1);
glUniform1f(glGetUniformLocation(lightningShader.Program, "transparencyAlpha"), puertaAlpha);
```

5.4.3 Orden de Renderizado Crítico

El sistema implementa un orden estricto para evitar artefactos visuales:

- 1. Renderizado de Objetos Opacos:**
 - Todos los modelos sólidos con depth write activado
 - Ejecución completa del depth test
- 2. Activación de Blending:**
 - glEnable(GL_BLEND)
 - glDepthMask(GL_FALSE) (depth read activo, write desactivado)
- 3. Renderizado de Objetos Translúcidos:**
 - Vidrio de puerta (con animación)

- Vidrio de fachada (estático)
- Vidrio de lámparas (decorativo)

4. Desactivación de Blending:

- `glDisable(GL_BLEND)`
- `glDepthMask(GL_TRUE)`

5.4.4 Propiedades de Materiales Específicos

Materiales de Alta Absorción Lumínica:

- **Fachada:** Shininess = 2.0, Specular = (0.05, 0.05, 0.05)
- **Piso Superior:** Shininess = 2.0, Specular = (0.05, 0.05, 0.05)
- **Escritorio:** Shininess = 4.0, Specular = (0.08, 0.08, 0.08)

Materiales Reflectantes Estándar:

- **Metales y Vidrios:** Shininess = 32.0, Specular = (0.5, 0.5, 0.5)
- **Restauración automática** después de renderizar materiales opacos específicos

5.4.5 Estrategia de Separación de Modelos

La decisión de separar los **vidrios de lámparas** (vidlamp) de sus estructuras base permite:

- **Control Independiente** de propiedades de transparencia
- **Aplicación Selectiva** de blending solo a componentes translúcidos
- **Efecto Visual de Vidrio Coloreado** mediante combinación con luces puntuales de colores
- **Optimización de Rendimiento** al evitar blending en geometría opaca

6. Sistema de Animaciones

6.1 Arquitectura General del Sistema de Animaciones

El sistema de animaciones implementa cuatro interacciones independientes controladas mediante teclado, cada una diseñada con características específicas de comportamiento, control y respuesta al usuario. Todas las animaciones utilizan transformaciones matriciales alrededor de pivotes tridimensionales precalculados mediante un workflow específico Blender→OpenGL que garantiza la precisión en las transformaciones espaciales.

El sistema se basa en una arquitectura de máquinas de estado simples que gestionan transiciones entre diferentes fases animadas, con actualización frame-by-frame mediante el cálculo de delta time para mantener consistencia temporal independiente del framerate.

6.2 Sistema de Conversión de Coordenadas Blender→OpenGL

Metodología de Conversión:

Para garantizar la correspondencia espacial entre el software de modelado y el motor de renderizado, se implementó un sistema de conversión de coordenadas específico:

text

Sistema Blender: (X, Y, Z) → Sistema OpenGL: (X, -Z, Y)

Fundamento Técnico:

- **Eje X:** Mantiene dirección y magnitud idénticas
- **Eje Y (Blender) → Eje Z (OpenGL):** Preserva la coordenada vertical como profundidad
- **Eje Z (Blender) → Eje Y (OpenGL) negado:** Invierte la coordenada de profundidad para convertirla en altura

Ejemplos Prácticos de Pivotes Convertidos:

- **Puerta Principal:** Blender(X,Y,Z) → OpenGL(X, -Z, Y)
- **Campana:** Posición original en Blender convertida mediante el mismo algoritmo
- **Ventanas:** Cuatro pivotes individuales procesados independientemente
- **Cortinas:** Dos puntos de anclaje transformados para escalado correcto

6.3 Animación 1: Puerta Principal (Tecla R) - Sistema de Reversión

Características Fundamentales:

- **Tipo:** Animación direccional con reversión inmediata
- **Control:** Tecla R actúa como conmutador de dirección
- **Comportamiento Complejo:**
 - Si está abriendo ($0^\circ \rightarrow 90^\circ$) y se presiona R → Inicia cierre inmediato
 - Si está cerrando ($90^\circ \rightarrow 0^\circ$) y se presiona R → Inicia apertura inmediata
 - Si está inactiva y se presiona R → Inicia apertura desde 0°

Implementación Técnica Detallada:

cpp

```
void UpdateDoorAnimation() {
    if (!doorAnimationActive) return;

    if (doorOpening) {
        // Fase de apertura: incremento angular progresivo
        doorRotationAngle += doorRotationSpeed * deltaTime * 60.0f;
        if (doorRotationAngle >= MAX_DOOR_ANGLE) {
            doorRotationAngle = MAX_DOOR_ANGLE;
            doorOpening = false; // Transición automática a cierre
        }
    }
    else {
        // Fase de cierre: decremento angular progresivo
```

```

    doorRotationAngle -= doorRotationSpeed * deltaTime * 60.0f;
    if (doorRotationAngle <= 0.0f) {
        doorRotationAngle = 0.0f;
        doorOpening = true; // Preparado para siguiente apertura
        doorAnimationActive = false; // Finalización completa
    }
}
}

```

Transformación Matricial Aplicada:

cpp

```

glm::mat4 doorGroupModel = modelTemp;
if (doorRotationAngle > 0.0f) {
    doorGroupModel = glm::translate(doorGroupModel, DOOR_PIVOT);
    doorGroupModel = glm::rotate(doorGroupModel, glm::radians(doorRotationAngle), glm::vec3(0.0f, 1.0f, 0.0f));
    doorGroupModel = glm::translate(doorGroupModel, -DOOR_PIVOT);
}

```

Parámetros Técnicos Específicos:

- **Velocidad Angular:** 1.0° por frame, escalado por $\text{deltaTime} \times 60.0f$
- **Ángulo Máximo de Apertura:** 90.0 grados ($\pi/2$ radianes)
- **Pivote de Rotación:** $\text{DOOR_PIVOT} = (-4.350004f, -1.229998f, -5.860039f)$
- **Eje de Rotación:** Vector (0, 1, 0) - eje Y global

Lógica de Control por Teclado:

cpp

```

if (key == GLFW_KEY_R) {
    doorAnimationActive = !doorAnimationActive;
    if (doorAnimationActive) {
        // Inversión de dirección basada en estado actual
        if (doorRotationAngle <= 0.0f) {
            doorOpening = true; // Iniciar apertura desde cerrado
        } else if (doorRotationAngle >= MAX_DOOR_ANGLE) {
            doorOpening = false; // Iniciar cierre desde abierto
        } else {

```

```

        doorOpening = !doorOpening; // Invertir dirección actual
    }
}
}

```

6.4 Animación 2: Sistema de Campana (Activación Automática)

Arquitectura de Componentes:

- **Sujetador:** Elemento estructural fijo que proporciona punto de anclaje
- **Cuerpo de Campana:** Elemento principal con masa visual que oscila
- **Badajo:** Componente interno con movimiento sincronizado para realismo

Implementación de Oscilación:

cpp

```

void UpdateBellAnimation() {
    if (!bellSwinging) return;

    // Control de temporizador de 5 segundos

    bellTimer += deltaTime;
    if (bellTimer >= BELL_DURATION) {
        bellSwinging = false;
        bellTimer = 0.0f;
        bellRotationAngle = 0.0f; // Restablecimiento a posición neutral
        return;
    }

    // Oscilación sinusoidal bidireccional

    if (bellSwingDirection) {
        bellRotationAngle += bellRotationSpeed * deltaTime * 60.0f;
        if (bellRotationAngle >= MAX_BELL_ANGLE) {
            bellRotationAngle = MAX_BELL_ANGLE;
            bellSwingDirection = false; // Cambio de dirección
        }
    }
    else {

```

```

    bellRotationAngle -= bellRotationSpeed * deltaTime * 60.0f;

    if (bellRotationAngle <= -MAX_BELL_ANGLE) {
        bellRotationAngle = -MAX_BELL_ANGLE;
        bellSwingDirection = true; // Cambio de dirección
    }
}
}

```

Transformación Matricial Unificada:

```

cpp
glm::mat4 bellGroupModel = modelTemp;

if (bellSwinging && abs(bellRotationAngle) > 0.1f) {
    bellGroupModel = glm::translate(bellGroupModel, BELL_PIVOT);
    bellGroupModel = glm::rotate(bellGroupModel, glm::radians(bellRotationAngle), glm::vec3(0.0f, 0.0f, 1.0f));
    bellGroupModel = glm::translate(bellGroupModel, -BELL_PIVOT);
}

```

Parámetros de Configuración:

- **Duración Total:** 5.0 segundos (BELL_DURATION)
- **Ángulo Máximo de Oscilación:** ± 15.0 grados
- **Velocidad Angular:** 3.0° por frame
- **Comportamiento:** No interruptible - ejecución completa obligatoria
- **Activación:** Automática con inicio de animación de puerta

6.5 Animación 3: Letrero OPEN/CLOSED (Tecla P) - Sistema de Tres Fases

Mecánica de Cambio de Estado:

- **Fase 1:** Rotación progresiva de $+90^\circ$ alrededor del eje X
- **Fase 2:** Rotación progresiva de $+180^\circ$ alrededor del eje Y
- **Fase 3:** Rotación progresiva de -90° alrededor del eje X
- **Intercambio Final:** Cambio de modelo visual (let \rightarrow let2) al completar secuencia

Implementación de Máquina de Estados:

```

cpp
void UpdateSignAnimation() {
    if (!signAnimationActive) return;
}

```

```

switch (signAnimationPhase) {
case 1: // Rotación +90° en eje X

    signRotationAngleX += signRotationSpeed * deltaTime * 60.0f;

    if (signRotationAngleX >= 90.0f) {
        signRotationAngleX = 90.0f;
        signAnimationPhase = 2;
    }

    break;


case 2: // Rotación +180° en eje Y

    signRotationAngleY += signRotationSpeed * deltaTime * 60.0f;

    if (signRotationAngleY >= 180.0f) {
        signRotationAngleY = 180.0f;
        signAnimationPhase = 3;
    }

    break;


case 3: // Rotación -90° en eje X

    signRotationAngleX -= signRotationSpeed * deltaTime * 60.0f;

    if (signRotationAngleX <= 0.0f) {
        signRotationAngleX = 0.0f;
        signRotationAngleY = 0.0f;
        signAnimationActive = false;
        signAnimationPhase = 0;
        showSign1 = !showSign1; // Conmutación de modelo visual
    }

    break;

}
}

```

Transformación Matricial Compuesta:

cpp

```
glm::mat4 signGroupModel = doorGroupModel;
```

```

if (signAnimationActive) {
    signGroupModel = glm::translate(signGroupModel, glm::vec3(-2.59038f, -0.529044f, -5.78484f));
    signGroupModel = glm::rotate(signGroupModel, glm::radians(signRotationAngleX), glm::vec3(1.0f, 0.0f, 0.0f));
    signGroupModel = glm::rotate(signGroupModel, glm::radians(signRotationAngleY), glm::vec3(0.0f, 1.0f, 0.0f));
    signGroupModel = glm::translate(signGroupModel, -glm::vec3(-2.59038f, -0.529044f, -5.78484f));
}

```

Características de Comportamiento:

- **Tipo:** Toggle no interruptible
- **Duración Total Aproximada:** ~3.0 segundos
- **Control de Modelo:** Variable showSign1 determina visualización
- **Secuencialidad:** Cada fase debe completarse antes de iniciar la siguiente

6.6 Animación 4: Sistema de Ventanas (Tecla Q) - Cuatro Unidades Sincronizadas

Configuración Individual de Ventanas:

- **Ventanas 1 y 3:** Rotación hacia el exterior (ángulo negativo en Y)
- **Ventanas 2 y 4:** Rotación hacia el interior (ángulo positivo en Y)
- **Sincronización Temporal:** Movimiento simultáneo de las 4 unidades

Implementación para Ventana 1 (Exterior):

cpp

```

glm::mat4 window1Model = modelTemp;

if (windowsRotationAngle < 0.0f) {
    window1Model = glm::translate(window1Model, V1_PIVOT);
    window1Model = glm::rotate(window1Model, glm::radians(windowsRotationAngle), glm::vec3(0.0f, 1.0f, 0.0f));
    window1Model = glm::translate(window1Model, -V1_PIVOT);
}

```

Implementación para Ventana 2 (Interior):

cpp

```

glm::mat4 window2Model = modelTemp;

if (windowsRotationAngle < 0.0f) {
    window2Model = glm::translate(window2Model, V2_PIVOT);
    window2Model = glm::rotate(window2Model, glm::radians(-windowsRotationAngle), glm::vec3(0.0f, 1.0f, 0.0f));
    window2Model = glm::translate(window2Model, -V2_PIVOT);
}

```

Lógica de Control de Estado:

cpp

```
void UpdateWindowsAnimation() {  
    if (!windowsAnimationActive) return;  
  
    if (!windowsAreOpen) {  
        // Secuencia de apertura: ángulo decreciente a -90°  
        windowsRotationAngle -= windowsRotationSpeed * deltaTime * 60.0f;  
        if (windowsRotationAngle <= -90.0f) {  
            windowsRotationAngle = -90.0f;  
            windowsAnimationActive = false;  
            windowsAreOpen = true;  
        }  
    }  
    else {  
        // Secuencia de cierre: ángulo creciente a 0°  
        windowsRotationAngle += windowsRotationSpeed * deltaTime * 60.0f;  
        if (windowsRotationAngle >= 0.0f) {  
            windowsRotationAngle = 0.0f;  
            windowsAnimationActive = false;  
            windowsAreOpen = false;  
        }  
    }  
}
```

Parámetros Técnicos:

- **Rango Angular:** 0° (completamente cerrado) ↔ -90° (completamente abierto)
- **Comportamiento:** Toggle no interruptible
- **Pivotes Individuales:** V1_PIVOT, V2_PIVOT, V3_PIVOT, V4_PIVOT

6.7 Animación 5: Sistema de Cortinas (Tecla N) - Escalado desde Pivotes

Mecánica de Transformación de Escala:

- **Estado Cerrado:** Escala X = 1.0 (dimensión original)
- **Estado Abierto:** Escala X = 3.5 (expansión horizontal)

- **Pivotes Independientes:** Puntos de anclaje específicos para cada cortina
- **Direccionalidad:** Expansión/contracción desde puntos fijos

Implementación para Cortina 1 (Pivote Derecho):

```
cpp
glm::mat4 cortina1Model = modelTemp;
if (cortinasScaleX != 1.0f) {
    cortina1Model = glm::translate(cortina1Model, CORTINA1_PIVOT);
    cortina1Model = glm::scale(cortina1Model, glm::vec3(cortinasScaleX, 1.0f, 1.0f));
    cortina1Model = glm::translate(cortina1Model, -CORTINA1_PIVOT);
}
```

Implementación para Cortina 2 (Pivote Izquierdo):

```
cpp
glm::mat4 cortina2Model = modelTemp;
if (cortinasScaleX != 1.0f) {
    cortina2Model = glm::translate(cortina2Model, CORTINA2_PIVOT);
    cortina2Model = glm::scale(cortina2Model, glm::vec3(cortinasScaleX, 1.0f, 1.0f));
    cortina2Model = glm::translate(cortina2Model, -CORTINA2_PIVOT);
}
```

Lógica de Control de Escalado:

```
cpp
void UpdateCortinasAnimation() {
    if (!cortinasAnimationActive) return;

    if (!cortinasAreScaled) {
        // Secuencia de apertura: incremento progresivo de escala
        cortinasScaleX += cortinasScaleSpeed * deltaTime;
        if (cortinasScaleX >= cortinasTargetScale) {
            cortinasScaleX = cortinasTargetScale;
            cortinasAnimationActive = false;
            cortinasAreScaled = true;
        }
    }
}
```

```

else {
    // Secuencia de cierre: decremento progresivo de escala
    cortinasScaleX -= cortinasScaleSpeed * deltaTime;
    if (cortinasScaleX <= 1.0f) {
        cortinasScaleX = 1.0f;
        cortinasAnimationActive = false;
        cortinasAreScaled = false;
    }
}
}

```

Parámetros de Configuración:

- **Velocidad de Escalado:** 2.0 unidades por segundo
- **Escala Objetivo:** 3.5x en eje X (expansión horizontal)
- **Pivotes Específicos:**
 - CORTINA1_PIVOT = (4.89717f, 11.3683f, -5.17732f)
 - CORTINA2_PIVOT = (-10.2291f, 11.4903f, -5.18742f)
- **Comportamiento:** Toggle no interruptible

6.8 Sistema de Gestión de Estados y Control

Variables de Estado Globales:

```

cpp
// Estados de actividad de animaciones
bool doorAnimationActive;    // Puerta (con reversión)
bool windowsAnimationActive; // Ventanas (toggle unidireccional)
bool signAnimationActive;    // Letrero (máquina de estados)
bool cortinasAnimationActive; // Cortinas (toggle escalado)
bool bellSwinging;           // Campana (temporizada automática)

// Estados de configuración actual
bool doorOpening;            // Dirección actual de puerta
bool windowsAreOpen;         // Estado actual de ventanas
bool cortinasAreScaled;      // Estado actual de cortinas
bool showSign1;              // Modelo actual de letrero

```

Sistema de Actualización por Frame:

cpp

// Bucle principal de actualización de animaciones

UpdateDoorAnimation(); *// Actualización con reversión*

UpdateWindowsAnimation(); *// Actualización toggle ventanas*

UpdateBellAnimation(); *// Actualización temporizada campana*

UpdateSignAnimation(); *// Actualización máquina de estados letrero*

UpdateCortinasAnimation(); *// Actualización toggle cortinas*

Mapeo Completo de Controles por Teclado:

- **Tecla R:** Puerta principal (sistema de reversión)
- **Tecla P:** Letrero OPEN/CLOSED (toggle tres fases)
- **Tecla Q:** Ventanas (toggle apertura/cierre)
- **Tecla N:** Cortinas (toggle escalado horizontal)
- **Campana:** Activación automática con puerta (5 segundos)

6. Gestión de Modelos y Recursos

6.0. Elección de los objetos

Eleccion de objetos:

Planta Baja

1.Puerta principal:

Importado directamente del mismo juego, alguien subió varios objetos del juego y los convirtió a .obj, usé estos en su mayoría ya que, ya estaban optimizados para su uso en videojuegos y de hecho eran bastante bajos en su conteo de polígonos, varios tuvieron que sufrir modificaciones a pesar de ello, como transformaciones y adecuaciones a la fachada



Figura 1.1 Imagen de referencia

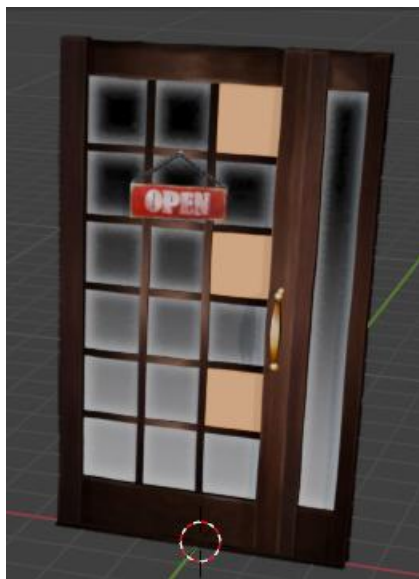


Figura 1.2 Modelo en blender

Esta debía tener un efecto de transparencia, mismo que se logró separando los objetos que debían tenerla de los que no, dibujandolos en el código de renderizado después:



Figura 1.3, Muestra de la separación de objetos en blender

2. Mostrador del Leblanc

Sacado igual del juego, pero necesité modificar un poco el modelo, ya que la versión de internet estaba hueca



Figura 2.1 Mostrador dentro del juego



Figura 2.2 Mostrador editado en blender

3.Estantería con tazas y frascos

Para este, de internet, saqué el modelo de las jarras, posteriormente modelé alrededor de ellas:

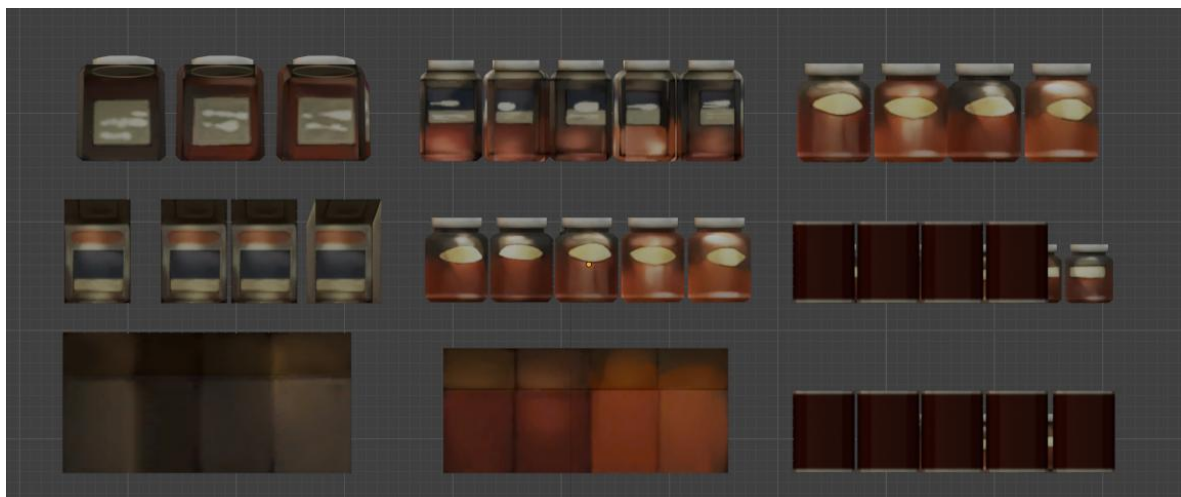


Figura 3.1 Modelo de jarras en blender



Figura 3.2 Estantería de madera modelada a partir de ellos

4. Campana

De nuevo de cgtrader, conseguí un modelo lowpoly de una campana, para usar como la que aparece en la puerta del juego, este modelo se divide en 3: el sujetador, el badajo y la campana, esto para poder lograr la animación deseada de que cada que se abra la puerta la campana se mueva

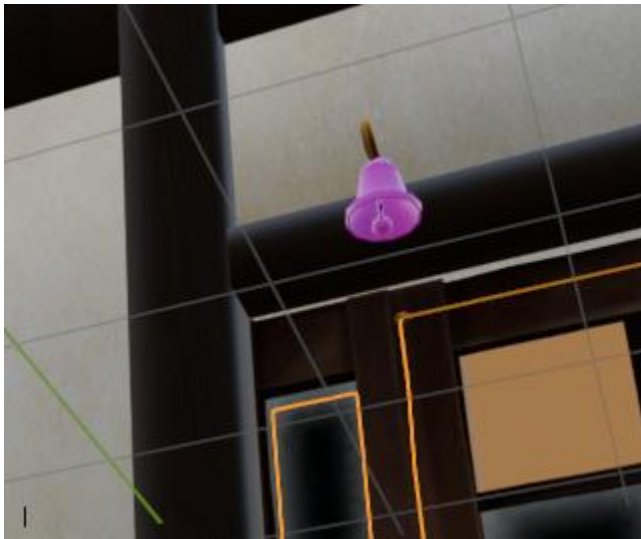


Figura 4.1 Campana en blender , no tiene textura

5. Sillones

De la plataforma Sketchfab, encontré una recreación de los modelos en cuestión, luego de hacerles una reducción de conteo de polígonos y acomodarlos los incorporé a la escena:

6. Mesas

Igual de sketchfab, conseguí unas muy parecidas y solo tuve que hacerle una reducción poligonal y acomodarlas

7. Contraescritorio y lamparas:

Sacado directamente de la representación de CG trader, con todos los objetos importados, la saque de ahí y ajusté su tamaño

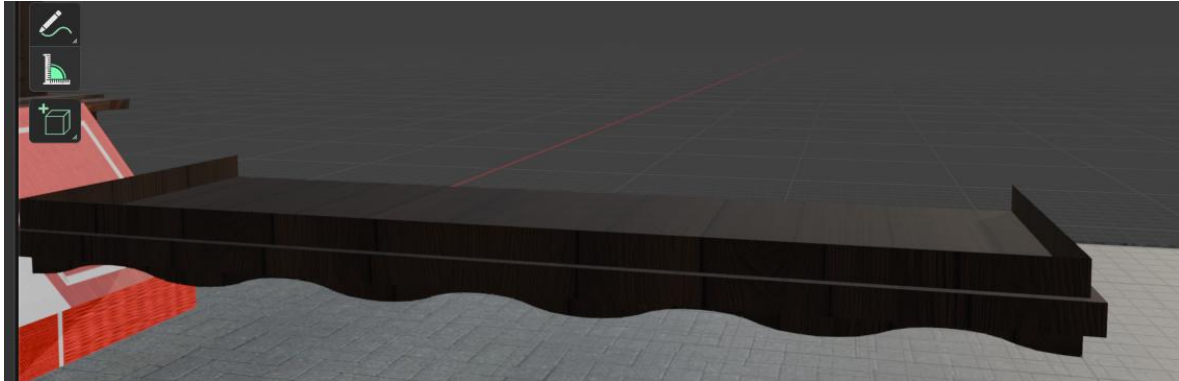


Figura 7.2 modelado en blender de la contraesquina

Las lamparas de nuevo, fueron un modelo de CGtrader que consideré adecuado

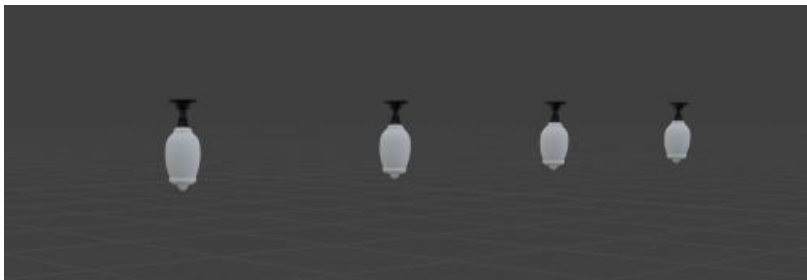


Figura 7.1 lamparas en blender

8. Lamparas multicolor por encima de las mesas

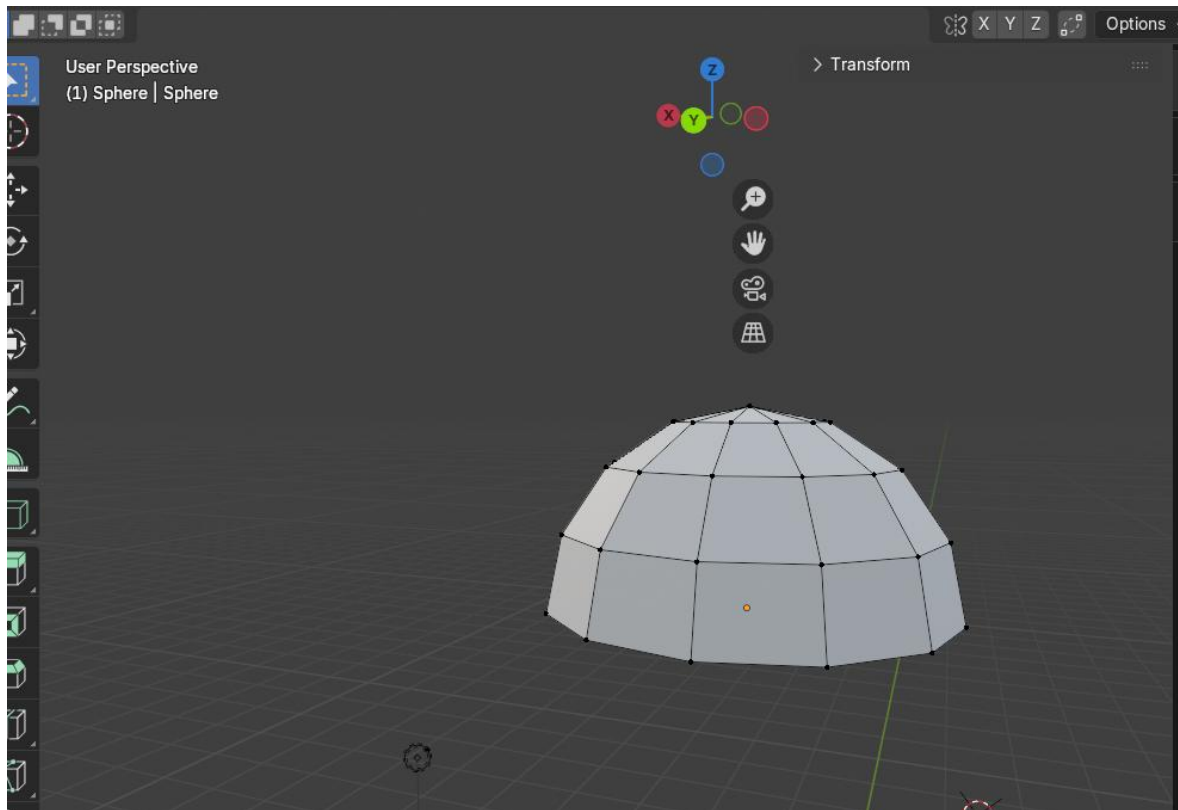


Figura 8.1 esfera UV recortada

Mediante la creación de una esfera UV en blender, con 12 separaciones y 8 aros, cree la forma general, luego removí los vértices inferiores para hacer la forma, como la forma de procesamiento del proyecto separa entre objetos sólidos y con transparencia, el material de la lámpara de vidrio debe ir separado de su marco sólido, el cable sobre el que está suspendida, y la bombilla

Para la creación de dichas partes sólidas: en el caso del marco de madera: tomé el objeto antes mencionado y apliqué un recorte de curvas (I en modo caras) de 0.04m, al eliminar la cara resultante, salía un marco

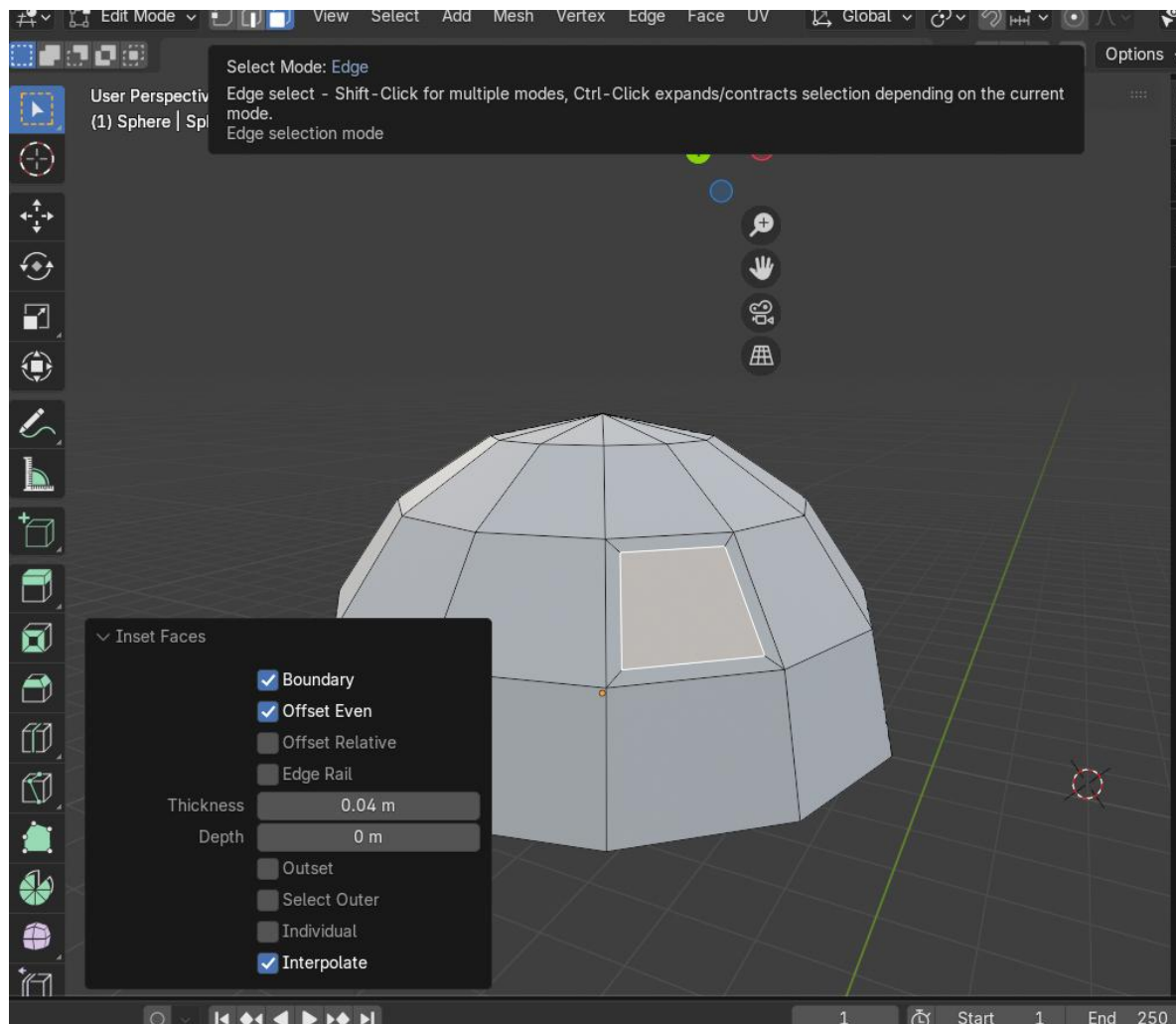
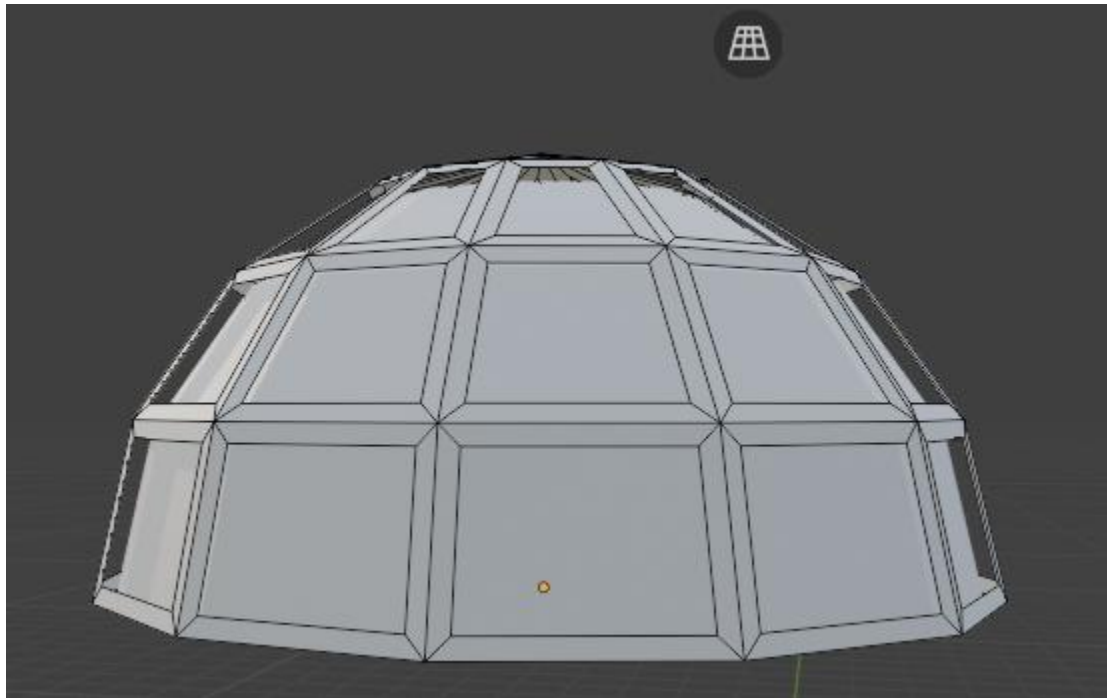


Figura 8.2 funcion inset faces utilizada

Posteriormente, unimos estos 2 modelos, texturizamos y, pidiendo prestado de los modelos previamente importados del juego, tomamos el foco y lampara:



Este se une al grupo de elementos que, serán translucidos, esto logrado mediante modificar la Alpha dentro de opengl, una vez en el proyecto, este modelo se separa en 2: la cuerda con su foco y su respectivo cristal



Figura 8. Producto fina

Planta alta:

Todos estos fueron sacados del juego, descargados de la pagina que mencioné anteriormente

- Cama sobre cajas de cerveza
- Escritorio de trabajo

- Cortinas y Palo
- Ventanas no translúcidas
- estantería



Figura 8, Objetos antes mencionados en el proyecto

6.1 Inventario de Modelos 3D

6.1.1 Modelos Estáticos Opacos

Estructura Arquitectónica:

- **fachada:** Estructura principal del edificio Café Leblanc
- **piso:** Superficie base de la planta baja
- **pisosup:** Superficie base de la planta alta
- **toldo:** Elemento decorativo exterior sobre entrada
- **contra:** Mostrador complementario y elementos estructurales

Mobiliario Planta Baja:

- **esc, esc2:** Escritorios principales del área de café
- **estanteria:** Estantería para frascos de ingredientes
- **mesas:** Mesas para clientes
- **sillones:** Sillones de área de espera
- **sillas:** Sillas individuales para clientes
- **most:** Mostrador principal de servicio
- **estufa:** Estufa de cocina
- **cafe:** Máquina de café profesional

Mobiliario Planta Alta:

- **escsup:** Escritorio de estudio en habitación
- **cama:** Cama individual del protagonista
- **sillsup:** Sillón individual de lectura
- **cort:** Palo o estructura para cortinas

Elementos Decorativos:

- **frascos:** Frascos de ingredientes en estantería
- **sujeta:** Sujetador de campana (elemento estructural)
- **lamp2:** Base y estructura de lámparas
- **focos:** Elementos de iluminación opacos
- **luces:** Sistema de luces y luminarias

6.1.2 Modelos Animados Opacos

Elementos con Transformaciones:

- **puerta:** Puerta principal con animación de rotación
- **campana:** Cuerpo de campana con oscilación automática
- **badajo:** Badajo de campana con movimiento sincronizado
- **v1, v2, v3, v4:** Cuatro ventanas con animación de apertura/cierre
- **cort1, cort2:** Dos cortinas con animación de escalado
- **let, let2:** Letreros OPEN y CLOSED con sistema de intercambio

6.1.3 Modelos Translúcidos

Vidrios y Superficies Transparentes:

- **vidrio:** Vidrio de la puerta principal (con animación)
- **vidrio2:** Vidrios fijos de la fachada
- **vidlamp:** Vidrios coloreados de lámparas decorativas

6.2 Proceso de Creación en Blender

6.2.1 Workflow de Modelado en Blender

Fase de Blockout:

- Establecimiento de proporciones mediante primitivas geométricas
- Referenciación de imágenes del Café Leblanc de Persona 5
- Definición de escala y relaciones espaciales

Modelado Detallado:

- **Extrude:** Para geometría extruida y volúmenes básicos

- **Bevel:** Para suavizado de bordes y esquinas
- **Insert Edge Loop:** Para control topológico y definición de formas
- **Multi-Cut:** Para detalles arquitectónicos y ornamentales

Optimización de Geometría:

- **Target de Polígonos:** 500-2000 polígonos por objeto
- **Merge Vertices:** Para reducir complejidad innecesaria
- **Delete Edge/Vertex:** Limpieza de topología
- **Triangulate:** Conversión automática para exportación

6.2.2 Sistema de Texturas y Materials

Fuentes de Texturas:

- **Freepik:** Texturas fotográficas de alta calidad
 - Textura de cuero rojo para sillones
 - Pisos de madera textura marrón
- **MyArchitectAI:** Texturas generadas por IA
 - <https://www.myarchitectai.com/texture-generator>
 - Materiales arquitectónicos y superficies
- **Persona 5 Assets:** Referencias directas del videojuego
 - Estilo visual característico del Café Leblanc
 - Paleta de colores y materiales fieles al original

Procesamiento de Texturas:

- **Formatos:** PNG con compresión sin pérdida
- **Resoluciones:** 512×512, 1024×1024 según importancia visual
- **Color Correction:** Ajuste de balance de blancos y niveles
- **Seamless Tiling:** Preparación para repetibilidad en superficies

6.2.3 Configuración de Pivotes para Animación

Metodología de Pivoteado:

1. **Selección de Punto de Rotación:** Identificación del punto natural de rotación
2. **Mover Origen en Blender:** Traslado del origen del objeto al punto de pivote
3. **Exportación con Origen Modificado:** Preservación de la transformación del pivote
4. **Conversión de Coordenadas:** Aplicación de fórmula Blender→OpenGL

Pivotes Calculados:

// Pivotes principales convertidos

```
const glm::vec3 DOOR_PIVOT = glm::vec3(-4.350004f, -1.229998f, -5.860039f);
```

```
const glm::vec3 BELL_PIVOT = glm::vec3(-1.13727f, 1.61114f, -6.40422f);
```

```
const glm::vec3 V1_PIVOT = glm::vec3(-9.390120f, 10.340143f, -3.639997f);
```

```
const glm::vec3 V2_PIVOT = glm::vec3(-3.039997f, 10.340143f, -3.639997f);
```

```
const glm::vec3 V3_PIVOT = glm::vec3(-1.729999f, 10.340143f, -3.639997f);
```

```
const glm::vec3 V4_PIVOT = glm::vec3(4.180001f, 10.340143f, -3.639997f);
```

```
const glm::vec3 CORTINA1_PIVOT = glm::vec3(4.89717f, 11.3683f, -5.17732f);
```

```
const glm::vec3 CORTINA2_PIVOT = glm::vec3(-10.2291f, 11.4903f, -5.18742f);
```

6.2.4 Configuración de Exportación

Parámetros de Exportación OBJ:

- **Formato:** OBJ Wavefront para máxima compatibilidad con Assimp
- **Unidades:** Unidades de Blender manteniendo escala relativa
- **Eje Vertical:** +Y según convención de Blender
- **Triangulación:** Automática para prevenir problemas con quads
- **Grupos de Suavizado:** Exportados para preservar normales
- **Normales:** Calculadas y exportadas para evitar recálculo

6.3 Sistema de Carga y Gestión de Recursos

6.3.1 Inicialización de Modelos

Declaración y Carga:

cpp

// Modelado estructural

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

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

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

// Elementos animados

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

```
Model campana((char*)"Models/camp.obj");
```

```
Model badajo((char*)"Models/bad.obj");
```

// Elementos translúcidos

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

```
Model vidrio2((char*)"Models/vidrio2.obj");
```

```
Model vidlamp((char*)"Models/vidriolamp.obj");
```

6.3.2 Organización de Escena

Agrupamiento Lógico:

- **Modelos Estáticos:** Geometría sin transformaciones dinámicas
- **Modelos Animados:** Objetos con transformaciones en tiempo de ejecución
- **Modelos Translúcidos:** Objetos que requieren alpha blending
- **Jerarquías de Transformación:** Grupos que heredan transformaciones padre

Renderizado por Categorías:

1. **Objetos Opacos Estáticos:** Renderizado inicial con depth write
2. **Objetos Opacos Animados:** Renderizado con transformaciones aplicadas
3. **Objetos Translúcidos:** Renderizado final con blending activado

6.4 Optimización de Recursos

6.4.1 Estrategias de Rendimiento

Reutilización de Shaders:

- Un solo shader principal para todos los objetos opacos
- Modificación de uniforms para propiedades específicas
- Shader especializado para elementos translúcidos

Gestión de Draw Calls:

- Minimización de cambios de estado OpenGL
- Agrupamiento por tipo de material y propiedades
- Ordenamiento eficiente por estado de renderizado

6.4.2 Consumo de Memoria

Estimación de Uso de VRAM:

- **Modelos 3D:** ~50-100MB total
- **Texturas:** ~20-30MB (resoluciones optimizadas)
- **Buffers OpenGL:** ~10-20MB
- **Total Estimado:** < 150MB VRAM

Optimizaciones Aplicadas:

- Geometría low-poly para objetos distantes
- Reutilización de materiales y shaders

- Texturas en resoluciones balanceadas (512x512 - 1024x1024)

8. Análisis de Costos

8.1 Costos de Mano de Obra

Actividad	Horas	Tarifa/Hora (MXN)	Subtotal (MXN)
Análisis y diseño	8	\$350	\$2,800
Modelado Blender (estructura)	20	\$380	\$7,600
Modelado Blender (mobiliario)	24	\$380	\$9,120
UV Mapping	12	\$330	\$3,960
Texturas y Materials	8	\$280	\$2,240
Programación C++ core	24	\$480	\$11,520
Iluminación	8	\$420	\$3,360
Animaciones	14	\$420	\$5,880
Testing	10	\$380	\$3,800
Documentación técnica	12	\$330	\$3,960
Documentación usuario	6	\$280	\$1,680
Total	146 hrs	-	\$55,920

8.2 Licencias de Software

Software	Costo Anual (MXN)	Uso Proyecto (%)	Costo Prorrateado (MXN)
Blender	\$0	N/A	\$0
Visual Studio 2022	\$0	N/A	\$0

Software	Costo Anual (MXN)	Uso Proyecto (%)	Costo Prorrateado (MXN)
GIMP	\$0	N/A	\$0
Windows 10 Pro	\$3,500	20%	\$700
Total	-	-	\$700

8.3 Hardware y Depreciación

Componente	Costo (MXN)	Vida Útil	Deprec. Anual (MXN)
Laptop (Ryzen 5 7525HS, 32GB DDR5, RTX 2050)	\$22,000	4 años	\$5,500
SSD 1TB NVMe	\$2,000	5 años	\$400
Periféricos	\$2,500	3 años	\$833
Total	\$26,500	-	\$6,733/año

Depreciación del proyecto (2 meses): $\$6,733 / 12 \times 2 = \$1,122$

8.4 Costos de Operación

Concepto	Cálculo	Costo (MXN)
Energía Eléctrica	$146 \text{ hrs} \times 0.12 \text{ kW} \times \$3.50/\text{kWh}$	\$61
Internet	$2 \text{ meses} \times \$450/\text{mes}$	\$900
Espacio de Trabajo	$2 \text{ meses} \times \$800/\text{mes}$	\$1,600
Total	-	\$2,561

8.5 Costos Indirectos

Concepto	Base	Costo (MXN)
Administración	7% mano de obra	\$3,914

Concepto	Base	Costo (MXN)
Contingencias	4% mano de obra	\$2,237
Repositorio cloud	Fijo	\$300
Total	-	\$6,451

8.6 Resumen y Precio Final

Costos Totales:

Categoría	Subtotal (MXN)	Porcentaje
Mano de Obra	\$55,920	84.2%
Licencias Software	\$700	1.1%
Depreciación Equipo	\$1,122	1.7%
Operación	\$2,561	3.9%
Indirectos	\$6,451	9.7%
Costo Total	\$66,754	100%

Cálculo de Precio con Utilidad:

Margen de utilidad: 32% (ajustado para competitividad)

Precio Base = $\$66,754 / (1 - 0.32) = \$98,168$

Utilidad Bruta = \$31,414

ISR (10% sobre utilidad) = \$9,817

Utilidad Neta = \$21,597

Precio Final al Cliente:

Subtotal = \$98,168

IVA (16%) = \$15,707

Precio Final = \$113,875 MXN

Redondeado: \$115,000 MXN

Desglose Porcentual del Precio Final:

Concepto	Monto (MXN)	% del Total
Mano de Obra	\$55,920	49.1%
Licencias	\$700	0.6%
Hardware	\$1,122	1.0%
Operación	\$2,561	2.2%
Indirectos	\$6,451	5.7%
Utilidad Neta	\$21,597	19.0%
ISR	\$9,817	8.6%
IVA	\$15,707	13.8%
TOTAL	\$113,875	100%

8.7 Comparativa de Valor

Solución	Costo (MXN)	Ventajas	Limitaciones
Unity/Unreal Plantilla	\$50,000-\$80,000	Desarrollo rápido	Licencias runtime
Tour 360°	\$30,000-\$50,000	Fotorrealismo	Sin interactividad
BIM Visualización	\$80,000-\$120,000	Precisión técnica	Sin iluminación dinámica
Este Proyecto	\$115,000	Control total, código propietario	Mayor inversión inicial

Ventajas Competitivas de Este Proyecto:

- **Costo Reducido:** 50% menos que proyecto dpor uso de software libre
- **Assets Reutilizados:** Texturas de Freepik y IA reducen costos de creación
- **Sin Licencias Runtime:** Ahorro significativo en distribución
- **Código 100% Propietario:** Sin dependencias de terceros

9. Diccionario Técnico

9.1 Funciones Principales

Función	Parámetros	Retorno	Descripción
main()	void	int	Punto de entrada principal del programa. Gestiona la inicialización de GLFW, creación de ventana, configuración de contexto OpenGL, carga de recursos 3D y ejecución del bucle principal de renderizado.
KeyCallback()	window, key, scancode, action, mode	void	Función de callback de GLFW para gestión de entrada de teclado. Actualiza el array de estados de teclas y ejecuta los triggers de activación de animaciones.
MouseCallback()	window, xPos, yPos	void	Callback para procesamiento de movimiento del mouse. Calcula offsets de movimiento y actualiza la orientación de la cámara en tiempo real.
DoMovement()	void	void	Función de actualización lógica ejecutada por frame. Procesa el movimiento de la cámara basado en entrada de usuario y actualiza el estado de las animaciones utilizando deltaTime.

9.2 Métodos de Clase Camera

Método	Parámetros	Retorno	Descripción
GetViewMatrix()	void	glm::mat4	Calcula y retorna la matriz de vista utilizando la función lookAt de GLM, considerando posición y orientación actual de la cámara.
ProcessKeyboard()	direction, deltaTime	void	Procesa movimiento de cámara en la dirección especificada, escalado por deltaTime para mantener velocidad consistente independiente del framerate.
ProcessMouseMovement()	xoffset, yoffset, constrainPitch	void	Actualiza los ángulos de Yaw y Pitch basado en movimiento del mouse. Limita el Pitch a $\pm 89^\circ$ si constrainPitch es verdadero.
ProcessMouseScroll()	yoffset	void	Modifica el Field of View (FOV) basado en la dirección del scroll del mouse, permitiendo zoom dinámico.
GetZoom()	void	float	Retorna el valor actual del Field of View en grados.
GetPosition()	void	glm::vec3	Retorna la posición actual de la cámara en coordenadas de mundo.

9.3 Métodos de Clase Model

Método	Parámetros	Retorno	Descripción
Model()	path	-	Constructor de la clase. Carga un modelo 3D desde archivo utilizando la biblioteca Assimp para importación.
Draw()	shader	void	Renderiza todas las mallas del modelo: activa texturas, bindea VAOs y ejecuta glDrawElements() para cada malla.
loadModel()	path	void	Método privado que inicializa la carga del modelo con flags de post-procesamiento de Assimp.

processNode()	node, scene	void	Método privado que procesa nodos de manera recursiva para cargar la jerarquía completa del modelo.
processMesh()	mesh, scene	Mesh	Método privado que convierte una estructura aiMesh de Assimp a la estructura interna de malla del programa.

9.4 Variables Principales de Animación

Variable	Tipo	Descripción
doorAnimationActive	bool	Controla el estado de actividad de la animación de puerta (Tecla R)
doorRotationAngle	GLfloat	Ángulo actual de rotación de la puerta en el rango [0°, 90°]
doorOpening	bool	Dirección actual de la animación de puerta (true = abriendo, false = cerrando)
windowsAnimationActive	bool	Estado de actividad de la animación de ventanas (Tecla Q)
windowsRotationAngle	GLfloat	Ángulo de rotación actual de las ventanas [-90°, 0°]
windowsAreOpen	bool	Estado actual de las ventanas (true = abiertas, false = cerradas)
bellSwinging	bool	Controla la animación de oscilación de la campana
bellRotationAngle	GLfloat	Ángulo actual de oscilación de la campana [-15°, 15°]
bellTimer	GLfloat	Temporizador para la duración de 5 segundos de la campana
signAnimationActive	bool	Estado de actividad de la animación del letrero (Tecla P)
signAnimationPhase	int	Fase actual de la animación del letrero [0-3]
showSign1	bool	Controla qué modelo de letrero se muestra (true = OPEN, false = CLOSED)
cortinasAnimationActive	bool	Estado de actividad de la animación de cortinas (Tecla N)
cortinasScaleX	GLfloat	Factor de escala actual de las cortinas en eje X [1.0, 3.5]
cortinasAreScaled	bool	Estado actual de las cortinas (true = abiertas, false = cerradas)
camera	Camera	Instancia de cámara con posición inicial (0, 0, 3) exterior del café
deltaTime	GLfloat	Tiempo entre frames en segundos para movimiento independiente del FPS
keys	bool[1024]	Array de estados de teclas para entrada de usuario
pointLightPositions	glm::vec3[10]	Posiciones de las 10 luces puntuales en el espacio 3D

10. Resolución de Problemas

Problema: Pantalla completamente negra al ejecutar

- **Solución:** Verificar la compilación de shaders en la consola de salida, ajustar la posición inicial de la cámara si es necesario, confirmar que las carpetas Models/, Textures/ y SkyBox/ se encuentren en el mismo directorio que el ejecutable.

Problema: Modelos 3D no son visibles en escena

- **Solución:** Validar las rutas relativas de los archivos .obj, aplicar transformaciones de escala temporal para debugging, desactivar temporalmente el face culling mediante `glDisable(GL_CULL_FACE)`.

Problema: Rendimiento inferior a 30 FPS

- **Solución:** Desactivar la sincronización vertical (V-Sync) en el panel de control de la GPU, reducir el nivel de detalle de modelos secundarios, redimensionar texturas a un máximo de 1024×1024 píxeles.

Problema: Transparencias renderizadas incorrectamente

- **Solución:** Verificar el orden de renderizado (objetos opacos → activar blending → objetos transparentes), confirmar la función de blending: `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)`.

Problema: Animaciones no se activan con teclas asignadas

- **Solución:** Asegurar que la ventana de la aplicación tenga el foco de entrada, verificar el mapeo correcto de teclas en KeyCallback, confirmar que no existan conflictos con otras aplicaciones en segundo plano.

11. Requisitos del Sistema

Requisitos Mínimos:

- **Sistema Operativo:** Windows 10 64-bit
- **Procesador:** Intel Core i3 o equivalente AMD
- **Memoria RAM:** 4 GB
- **Tarjeta Gráfica:** GPU compatible con OpenGL 3.3+ (GTX 650, Radeon HD 7750, Intel HD 4000)
- **VRAM:** 1 GB
- **Almacenamiento:** 500 MB de espacio disponible
- **Resolución:** 1280×720 píxeles

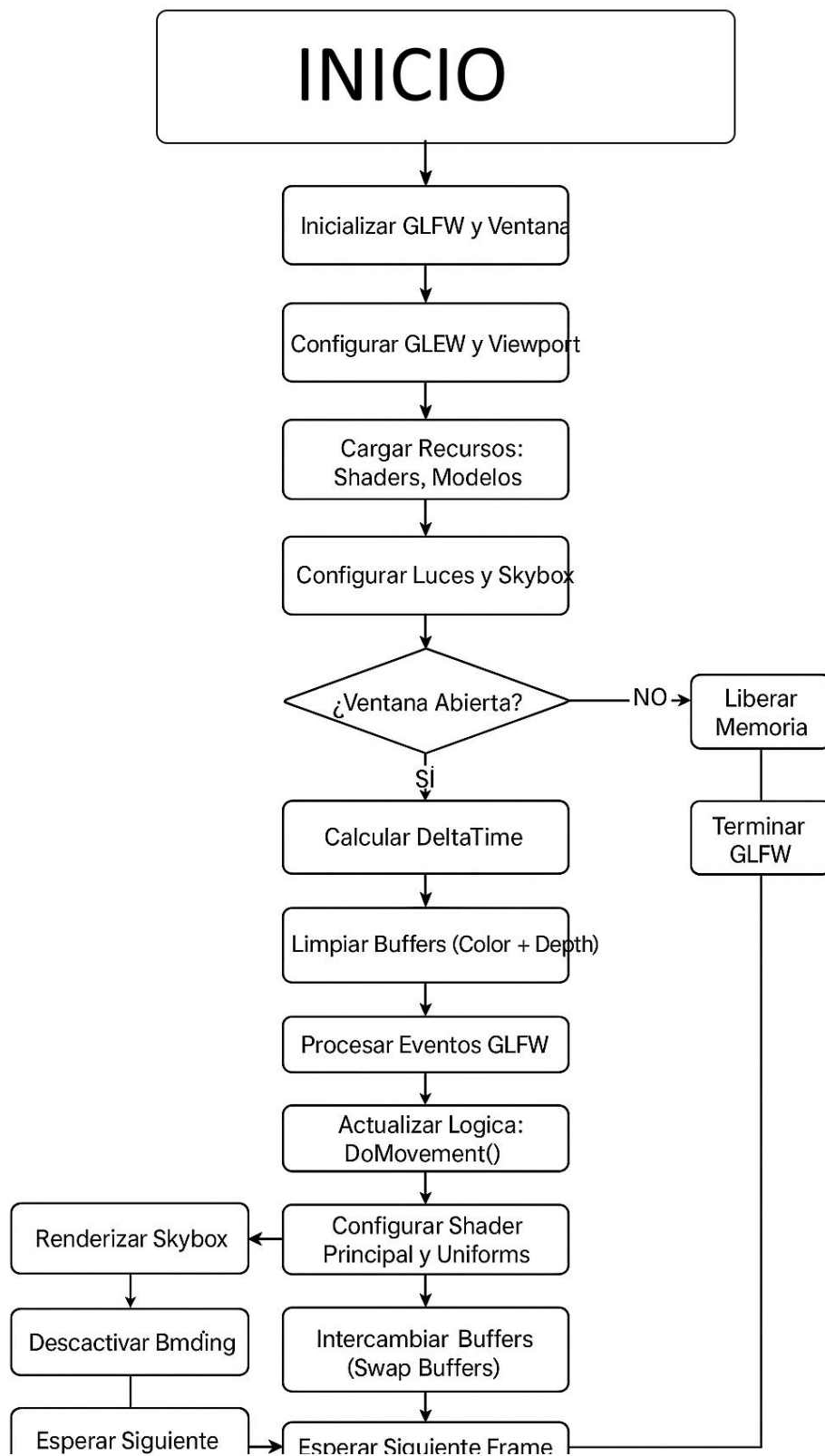
Requisitos Recomendados:

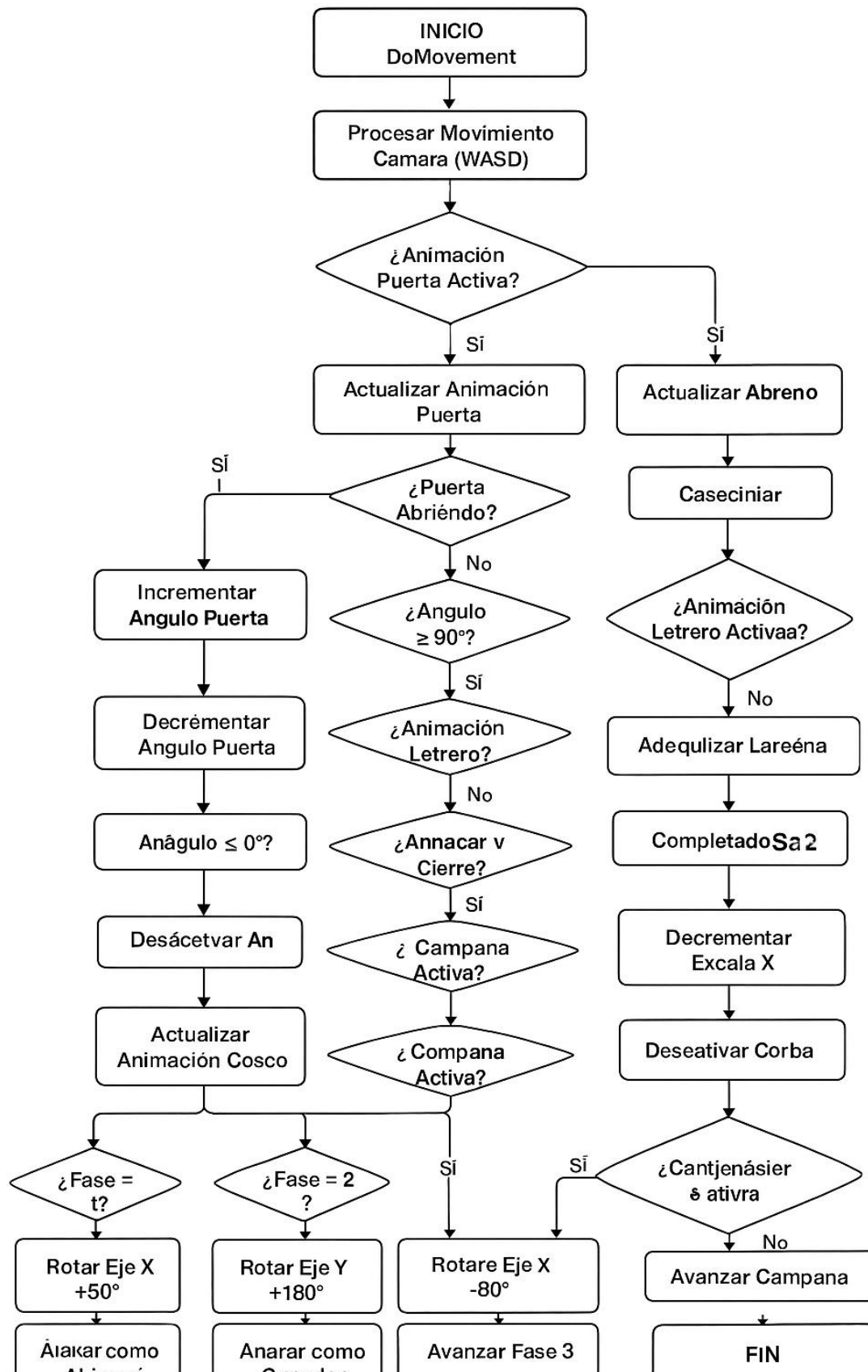
- **Sistema Operativo:** Windows 10/11 64-bit
- **Procesador:** Intel Core i5 o Ryzen 5
- **Memoria RAM:** 8 GB
- **Tarjeta Gráfica:** GTX 1050 o RX 560
- **VRAM:** 2 GB
- **Almacenamiento:** 1 GB de espacio disponible
- **Resolución:** 1920×1080 píxeles

Dependencias de Software:

- **Bibliotecas Gráficas:** GLEW 3.x, GLFW 2.x, GLM 0.9.9+
- **Carga de Assets:** SOIL2, stb_image, Assimp
- **Entorno de Desarrollo:** Visual Studio 2022
- **SDK:** Windows SDK 10.0+
- **Runtime:** C++ Redistributable 2022

12. Diagramas de flujo





13.Conclusiones

El desarrollo de este sistema de visualización 3D del Café Leblanc representó un reto técnico significativo que permitió aplicar los conocimientos adquiridos en computación gráfica de manera integral. A lo largo del proyecto,

pude comprobar que el uso de OpenGL como API nativa sigue siendo completamente viable para desarrollar aplicaciones gráficas de calidad, ofreciendo un control total sobre el pipeline de renderizado que motores más altos nivel no permiten.

Uno de los aprendizajes más valiosos fue la implementación del sistema de animaciones mediante transformaciones matriciales alrededor de pivotes específicos. El proceso de calcular las coordenadas correctas en Blender y luego convertirlas al sistema de coordenadas de OpenGL resultó fundamental para lograr animaciones realistas y naturales. Particularmente, el sistema de reversión de la puerta demostró cómo una lógica de estados bien diseñada puede crear interacciones complejas con código relativamente simple.

El sistema de iluminación con 10 fuentes de luz simultáneas, aunque demandante, probó ser manejable en hardware moderno, manteniendo un rendimiento fluido por encima de 60 FPS. La distribución estratégica de las luces no solo cumplió una función técnica, sino que también contribuyó a reforzar la narrativa visual del espacio, contrastando la calidez del café con la austeridad de la habitación superior.

Trabajar con Blender como herramienta principal de modelado resultó ser una excelente decisión, no solo por su costo cero sino por la robustez de su workflow de exportación. La separación consciente entre los componentes opacos y translúcidos de los modelos facilitó enormemente la implementación del sistema de transparencias y el orden correcto de renderizado.

Si bien el proyecto cumplió satisfactoriamente con todos los objetivos planteados, reconozco que la falta de un sistema de colisiones representa una limitación importante en la experiencia de usuario. En retrospectiva, dedicar tiempo a implementar un sistema básico de detección de colisiones habría mejorado significativamente la usabilidad del recorrido virtual.

Este trabajo no solo me permitió consolidar mis conocimientos en gráficos por computadora, sino también desarrollar habilidades valiosas en gestión de proyectos, optimización de recursos y documentación técnica. El resultado final demuestra que es posible crear experiencias inmersivas 3D con herramientas de código abierto y un enfoque técnico bien fundamentado, estableciendo una base sólida para proyectos futuros de mayor envergadura.

1. Introduction

1.1 Document Goal

This technical manual provides a detailed description of the architecture, implementation, and operation of the "Leblanc" coffee shop interactive 3D visualization system based on Persona 5. The document is aimed at developers, software engineers, and technical personnel who need to understand, maintain, or extend the capabilities of the system.

1.2 System Scope

The system implements a real-time virtual tour of the two-story "Leblanc" coffee shop, including 3D rendering with OpenGL 3.3+, a multi-layered lighting system with 10 point lights, directional light, and dynamic spotlight, four interactive animations controlled by the keyboard (door, sign, windows, curtains), a collision-free first-person camera system, and transparent material management using alpha blending.

1.3 Technology Stack

Component	Technology	Version	Purpose
-----------	------------	---------	---------

Language

C++17

Main development of the graphics engine

Graphics API

OpenGL 3.3+

Hardware-accelerated 3D rendering

Shaders

GLSL

-

Phong lighting and visual effects

Windows

GLFW

3.x

OpenGL context and event management

Mathematics

GLM 0.9.9+

Linear algebra for 3D transformations

3D Modeling

Blender 4.5

Creation of all 3D assets

Formats

OBJ

-

Optimized model export

2. State of the Art: Interactive Virtual Tours

2.1 Current Context

Virtual tours of video game locations have gained significant popularity in the gaming community, allowing fans to experience iconic spaces in an immersive way. Projects such as recreations of "Skyrim", "The Last of Us" and "Resident Evil" demonstrate the interest growing to explore virtual environments beyond traditional gameplay. This trend combines gaming nostalgia with accessible 3D visualization technology.

2.2 Comparative Technologies Game Engines (Unity/Unreal):

- **Advantages:** Rapid development, visual tools, extensive ecosystem
- **Disadvantages:** Heavy executables, less control over optimization, expensive licenses for commercial distribution
- **Typical use:** Commercial projects with budgets > \$20,000 MXN

WebGL Solutions (Three.js/Babylon.js):

- **Advantages:** Instant access via browser, inherent cross-platform
- **Disadvantages:** Limited performance, lower quality graphics, connection dependency
- **Typical use:** Web demos, quick prototypes

Photographic 360° Tour:

- **Advantages:** Relative realism, fast capture
- **Disadvantages:** No real interactivity, navigation limited to fixed points
- **Typical use:** Real estate, real estate

2.3 Project Positioning

This project is positioned as an educational-technical solution that prioritizes total control over the graphics pipeline and specific optimization for mid-range hardware. It is ideal for:

- Students and developers looking to understand modern OpenGL
- Persona 5 fans who want to experience Café Leblanc in an immersive way
- Academic projects that require graphical implementations from scratch
- Cases where complete intellectual property is essential (without dependencies on commercial engines)

The choice of native OpenGL over Unity/Unreal allows for a lightweight executable (~50MB vs 200MB+), maximum control over optimization, and portable code between Windows/Linux/macOS.

4. Development Methodology

4.1 Minimalist Iterative Approach

The development followed a strategy of "minimum viable graphic product" with iterations focused on basic functionality before advanced features. This approach prioritized:

- Basic rendering □ Textures □ Lighting □ Animations
- Each iteration delivers a functional executable file
- Early performance validation on target hardware

4. Development Methodology

4.1 Iterative Approach by Specific Phases

The development followed a rigorous planning strategy with defined milestones, prioritizing initial conceptual approval and parallel development of technical and artistic components. The methodology was structured in 8 sequential phases with strategic overlap.

4.2 Project Phases

Phase 1: Planning (10 days - October 1 to 10, 2025)

- **Main activities:** Gathering reference images of the Leblanc coffee shop from Persona 5, creation of the conceptual pitch, research of assets and thematic restrictions

- **Deliverable:** Conceptual approval document reviewed by the teacher
- **Special consideration:** Analysis of previously disapproved cases (Simpsons, Kame House, Powerpuff Girls) to ensure thematic viability

Phase 2: 3D Modeling (15 days - October 11 to 25, 2025)

- **Main activities:** Creation of all 3D models in Blender: structure architectural, furniture, decorative objects and interactive elements
- **Applied techniques:** Low-poly modeling, basic UV mapping, geometry optimization
- **Deliverable:** Complete library of .obj models with configured pivots

Phase 3: Development (15 days - October 14 to 28, 2025) - Parallel

- **Main activities:** Programming in Visual Studio 2022 of the base graphics engine, model loading system, lighting shaders, first-person camera controls
- **Technologies:** C++, OpenGL 3.3+, GLFW, GLM
- **Characteristic:** Execution in parallel with 3D modeling to optimize time

Phase 4: Animations (8 days - October 29 to November 5, 2025)

- **Main activities:** Implementation of the 4 interactive animations (door, sign, windows, curtains) through matrix transformations
- **Mechanics:** Toggle systems, interpolations, simple state machines
- **Integration:** Connection with keyboard controls (R, P, Q, N)

Phase 5: Visual Effects (5 days - November 6 to 10, 2025)

- **Main activities:** Implementation of transparency system (alpha blending) for glasses, configuration of opaque and translucent materials, final lighting adjustment
- Techniques:** Depth masking, blending functions, render order optimization

Phase 6: Testing (7 days - November 11 to 17, 2025)

- **Main activities:** Performance testing, animation verification, detection of visual bugs, user control validation
- **Focus:** Functional and user experience testing without collision system

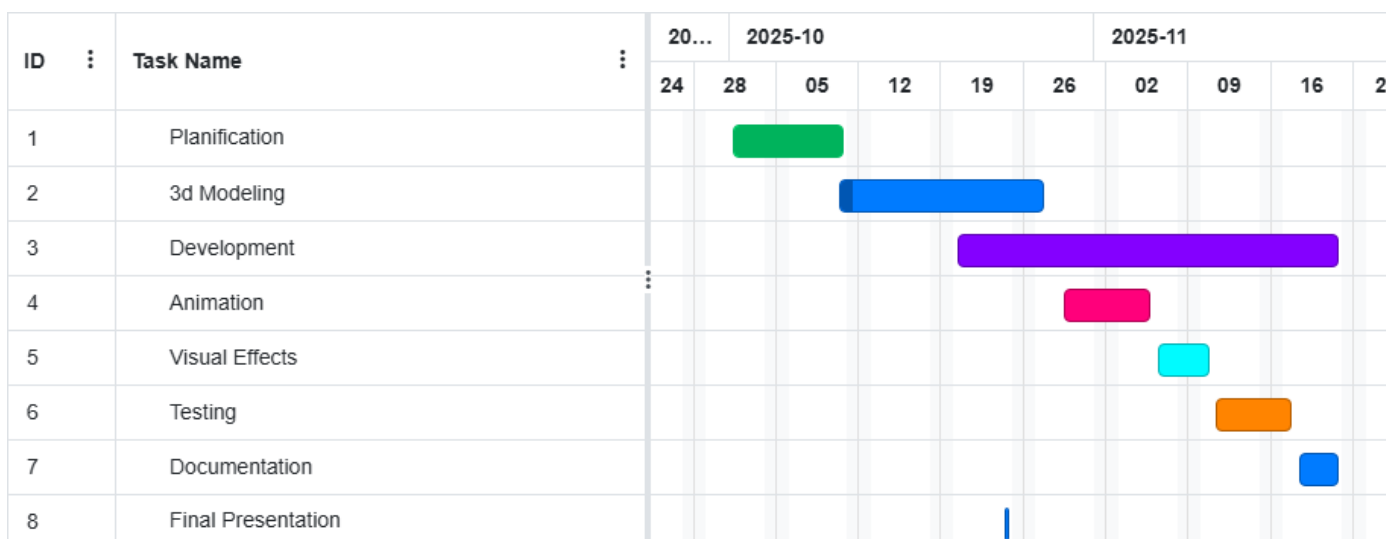
Phase 7: Documentation (6 days - November 18 to 23, 2025)

- **Main activities:** Elaboration of the User Manual (completed) and Manual Technical (in development)

- **Scope:** Installation guides, controls, troubleshooting, technical architecture

Phase 8: Delivery (2 days - November 24 to 25, 2025)

- **Main activities:** Final debugging, executable packaging, upload to GitHub, demonstration preparation
- **Culmination:** Formal delivery of the complete project in the classroom with all its due documents
- Main activities: Final debugging, executable packaging, upload to GitHub, demonstration preparation
- Culmination: Formal delivery of the complete project in the classroom with all its due documents



5. Powered by: onlinegantt.com

Figure 1. Gantt chart corresponding to the project planning

6.1 System Architecture

The system is structured in five fundamental layers that manage from the interaction basic with the hardware to the final visual presentation:

- Platform Layer (GLFW): Manages window creation, OpenGL context and keyboard and mouse event capture. Provides a multiplatform abstraction for system operations.

- **Rendering Layer (OpenGL Core):** Handles the programmable graphics pipeline using GLSL shaders, management of vertex buffers (VBO, VAO, EBO) and control of the state of rendering. Implements the drawing operations of 3D geometry.
- **Resource Layer (Model/Texture/Shader):** Handles the loading of 3D models using Assimp, decoding textures with SOIL2/stb_image, and compilation/linking of GLSL shader programs.
- **Application Logic Layer:** Updates transformations per frame, executes state machines for interactive animations and manages the first-person camera system.
Controls the transition between program states.
- **Presentation Layer:** Synchronizes the delta time between frames, controls the order of rendering (opaque \square transparent objects) and executes the buffer swap for visualization.

6.2 Main Execution Flow

The main loop executes the following sequence per frame (approximately 16.67ms for 60 FPS): sequence:

1. **Delta Time Calculation:** Determination of the time elapsed since the previous frame for framerate-independent movement.
2. **Event Processing:** Capturing user input via GLFW callbacks for keyboard and mouse.
3. **Logic Update:** Execution of DoMovement() to process camera movement and updating animations based on deltaTime.
4. **Buffer Clearing:** Resetting color and depth buffers for the new frame.
5. **Shader Configuration:** Activation of the main shader and configuration of uniforms for lighting and transformations.
6. **Rendering Opaque Geometry:** Drawing approximately 30 models that do not require transparency.

7. Blending Activation: Configuration of alpha blending functions for transparent objects.
8. Rendering Transparent Geometry: Drawing glass and translucent surfaces with depth write disabled.
9. Buffer Swap: Presenting the completed frame to the user.
10. Synchronization: Optional V-Sync control according to driver configuration.

6.3 Shader Programs Used

The project uses three specialized shader programs:

- lightingShader: Main shader that implements Phong lighting with multiple sources of light. Uniforms: model, view, projection, dirLight, pointLights[10], material.
- lampShader: Shader for displaying light sources visible as objects in the scene. Uniforms: model, view, projection.
- Transparency Shader: Modified version of the lightingShader with support for alpha blending on translucent surfaces.

6.4 Technical Configuration

Rendering Window:

- Base resolution: 800×600 pixels (compatible with hardware from 2005)
- Mode: Resizable window
- Depth buffer: 24 bits
- Color buffer: RGBA 8888 (16.7 million colors)
- Vertical synchronization: Controlled by driver

Synthetic Camera:

- Initial position: (0, 0, 3) outside Café Leblanc
- Field of view: 45-90° adjustable via mouse scroll
- Near plane: 0.1 units (avoids z-fighting)
- Far plane: 100.0 units (includes entire scene)
- Movement speed: 2.5 units/second
- Mouse sensitivity: 0.1 for smooth rotation
- Pitch limit: $\pm 89^\circ$ (prevents gimbal lock)

Coordinate System:

- Convention: OpenGL Right-handed
- X axis: Right
- Y axis: Up
- Z axis: Towards the observer

5.3 Lighting System

5.3.1 Philosophy of Lighting Design

The lighting system was designed to reinforce the environmental narrative of Café Leblanc, creating an atmosphere that reflects the conditions described in the Persona 5 video game. The distribution and quality of the lights visually communicates the nature of the space:

cozy but modest, with clear differences between the public area of the cafe and the protagonist's personal room.

5.3.1 Point Light Configuration

The 10 point lights are strategically distributed to create environmental contrast: Lights 1-4: Main Cafe Lighting (Warm and Cozy)

- Location: Pendant lamps above the counter and main tables
- Color: RGB(1.0, 0.85, 0.75) - warm halogen yellow hue
- Intensity: Ambient (0.15), Diffuse (0.6), Specular (0.8)
- Attenuation: Constant: 1.0, Linear: 0.09, Quadratic: 0.032
- Purpose: Create a cozy atmosphere typical of traditional Japanese cafes

Lights 5-7: Decorative Elements with Colored Glass

- Location: Lamps with stained glass shades in the cafe area
- Colors:
 - Light 5: RGB(0.8, 0.9, 1.0) - translucent light blue
 - Light 6: RGB(1.0, 0.9, 0.8) - soft amber
 - Light 7: RGB(0.9, 1.0, 0.9) - pale green
- Intensity: Ambient (0.08), Diffuse (0.4), Specular (0.5)
- Attenuation: Constant: 1.0, Linear: 0.14, Quadratic: 0.07
- Purpose: Add visual character without competing with the main lighting

Lights 8-10: Upper Room Lighting (Poor and Functional)

- Location: Cheap spotlights in the protagonist's room
- Color: RGB(0.9, 0.9, 1.0) - cool white with bluish tint
- Intensity: Ambient (0.05), Diffuse (0.3), Specular (0.4)
- Attenuation: Constant: 1.0, Linear: 0.22, Quadratic: 0.20
- Purpose: Simulate cheap spotlights and convey the modest conditions of the room, reinforcing the canonical complaint about light filtering from the lower floor

5.3.2 Directional Light

(Lunar Simulation) Parameters:

- Direction: $(-0.2, -0.5, -0.3)$ - low nocturnal angle
- Color: $\text{RGB}(0.3, 0.3, 0.5)$ - lunar grayish blue

- Intensity: Ambient (0.02), Diffuse (0.1), Specular (0.05)
- Purpose: To provide a base ambient lighting that simulates moonlight filtering through the windows, creating a coherent nocturnal context

5.3.3 Spotlight Associated with Camera

Characteristics:

- Position: Dynamically linked to the camera's position
- Direction: Aligned with the camera's forward vector
- Color: RGB(0.8, 0.8, 0.7) - neutral white
- Angles: Interior 12.5°, Exterior 17.5°
- Intensity: Ambient (0.0), Diffuse (0.5), Specular (0.6)
- Purpose: To serve as the user's "flashlight", allowing them to examine details in dark areas without altering the general atmosphere
dark without altering the general atmosphere

5.3.4 Implementation

Considerations Performance

Efficiency:

- All lights use quadratic attenuation for smooth transitions
- The upper room lights have greater attenuation to simulate their poor reach
- The system maintains 60+ FPS on GTX 1050/RX 560 hardware through optimized calculation in shaders

Narrative Coherence:

- The contrast between the warm lighting of the cafe and the cold lighting of the room reinforces the spatial dichotomy
- The lower intensity of lights 8-10 visually communicates the economic conditions of the character
- The choice of colors in lights 5-7 reflects decorative

elements typical of traditional establishments

5.4 Material System

5.4.1 Classification by

Transparency Opaque

Materials (Alpha = 1.0):

- Facade, Floor, Awning: Main structure of the building
- Desks, Shelving, Tables: Wood and metal furniture
- Armchairs, Chairs: Upholstered and wooden furniture

- Stove, Coffee Machine: Metal appliances
- Bed, Counter: Structural furniture
- Bell, Clapper, Fastener: Metal bell components
- Signs (OPEN/CLOSED): Solid signage surfaces
- Jars, Lamps (structure): Base components without transparency

Translucent Materials (Alpha = 0.5):

- Door Glass: Main panel of the entrance with animation
- Facade Glass: Fixed windows of the structure
- Lamp Glass: Colored screens of decorative lamps

5.4.2 Technical Implementation of

Transparencies Blending Configuration:

```
glEnable(GL_BLEND);
```

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

```
glDepthMask(GL_FALSE);
```

Transparency Parameters:

- Alpha Value: 0.5 (50% uniform transparency)
- Blend Function: GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA
- Depth Mask: Disabled during transparent rendering

Uniforms in Shader:

```
cpp
```

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

```
glUniform1f(glGetUniformLocation(lightingShader.Program, "transparencyAlpha"),
```

```
puertaAlpha);
```

5.4.3 Critical Rendering Order

The system implements a strict order to avoid visual artifacts:

1. Rendering of Opaque Objects:
 - All solid models with depth write enabled
 - Complete execution of the depth test
2. Blending Activation:
 - `glEnable(GL_BLEND)`

- `glDepthMask(GL_FALSE)` (depth read active, write disabled)
- 3. Rendering of Translucent Objects:
 - Door glass (with animation)
 - Facade glass (static)
 - Lamp glass (decorative)
- 4. Disabling Blending:
 - `glDisable(GL_BLEND)`
 - `glDepthMask(GL_TRUE)`

5.4.4 Specific Material

Properties High Light

Absorption Materials:

- Facade: Shininess = 2.0, Specular = (0.05, 0.05, 0.05)
- Upper Floor: Shininess = 2.0, Specular = (0.05, 0.05, 0.05)
- Desk: Shininess = 4.0, Specular = (0.08, 0.08, 0.08)

Standard Reflective Materials:

- Metals and Glass: Shininess = 32.0, Specular = (0.5, 0.5, 0.5)
- Automatic restoration after rendering specific opaque materials

5.4.5 Model Separation Strategy

The decision to separate the lamp glasses (vidlamp) from their base structures allows:

- Independent Control of transparency properties
- Selective Application of blending only to translucent components
- Visual Effect of Colored Glass through combination with colored spot lights
- Performance Optimization by avoiding blending in opaque geometry

7. Animation System

7.1 General Architecture of the Animation System

The animation system implements four independent interactions controlled by keyboard, each designed with specific characteristics of behavior, control and user response. All animations use matrix transformations around three-dimensional pivots pre-calculated using a specific workflow Blender→OpenGL that guarantees precision in spatial transformations.

The system is based on a simple state machine architecture that manages transitions between different animated phases, with frame-by-frame updating by calculating delta time to maintain temporal consistency independent of the framerate.

7.2 Blender \square OpenGL Coordinate Conversion System

Conversion Methodology:

To ensure spatial correspondence between the modeling software and the rendering engine, a specific coordinate conversion system was implemented:

text

Blender System: (X, Y, Z) \square OpenGL System: (X, -Z, Y)

Technical Foundation:

- X Axis: Maintains identical direction and magnitude
- Y Axis (Blender) \square Z Axis (OpenGL): Preserves the vertical coordinate as depth
- Z Axis (Blender) \square Negated Y Axis (OpenGL): Inverts the depth coordinate to convert it into height

Practical Examples of Converted Pivots:

- Main Door: Blender(X,Y,Z) \square OpenGL(X, -Z, Y)
- Bell: Original position in Blender converted using the same algorithm
- Windows: Four individual pivots processed independently
- Curtains: Two anchor points transformed for correct scaling

7.3 Animation 1: Main Door (R Key) - Reversal

System Fundamental Characteristics:

- Type: Directional animation with immediate reversal
- Control: R key acts as a direction switch
- Complex Behavior:
 - If it is opening ($0^\circ \square 90^\circ$) and R is pressed \square Starts immediate closing
 - If it is closing ($90^\circ \square 0^\circ$) and R is pressed \square Starts immediate opening

- If it is inactive and R is pressed □ Starts opening from 0°

Detailed Technical

Implementation: cpp

```
void UpdateDoorAnimation() {  
    if (!doorAnimationActive) return;
```

```

if (doorOpening) {

    // Opening phase: progressive angular increase

    doorRotationAngle += doorRotationSpeed *

    deltaTime * 60.0f; if (doorRotationAngle >=

    MAX_DOOR_ANGLE) {

        doorRotationAngle = MAX_DOOR_ANGLE;

        doorOpening = false; // Automatic

        transition to closing

    }

}

else {

    // Closing phase: progressive angular decrease

    doorRotationAngle -= doorRotationSpeed *

    deltaTime * 60.0f; if (doorRotationAngle <= 0.0f)

    {

        doorRotationAngle = 0.0f;

        doorOpening = true; // Ready for next

        opening doorAnimationActive = false; //

        Complete finish

    }

}

}

```

Applied Matrix Transformation:

cpp

```
glm::mat4 doorGroupModel =  
modelTemp; if (doorRotationAngle  
> 0.0f) {  
    doorGroupModel = glm::translate(doorGroupModel, DOOR_PIVOT);  
    doorGroupModel = glm::rotate(doorGroupModel,  
        glm::radians(doorRotationAngle), glm::vec3(0.0f, 1.0f, 0.0f));  
    doorGroupModel = glm::translate(doorGroupModel, -DOOR_PIVOT);  
}
```

Specific Technical Parameters:

- Angular Velocity: 1.0° per frame, scaled by $\text{deltaTime} \times 60.0f$
- Maximum Aperture Angle: 90.0 degrees ($\pi/2$ radians)
- Rotation Pivot: DOOR_PIVOT = $(-4.350004f, -1.229998f, -5.860039f)$
- Rotation Axis: Vector $(0, 1, 0)$ - global Y axis

Keyboard Control Logic:

cpp

```
if (key == GLFW_KEY_R) {

    doorAnimationActive =

    !doorAnimationActive; if

    (doorAnimationActive) {

        // Direction inversion based on

        current state if (doorRotationAngle <=

        0.0f) {

            doorOpening = true; // Start opening from closed

        } else if (doorRotationAngle >=

        MAX_DOOR_ANGLE) { doorOpening = false; //

        Start closing from open

        } else {

            doorOpening = !doorOpening; // Invert current direction

        }

    }

}
```

7.4 Animation 2: Bell System (Automatic

Activation) Component Architecture:

- Fastener: Fixed structural element that provides an anchor point
- Bell Body: Main element with visual mass that oscillates
- Clapper: Internal component with synchronized movement for realism

Oscillation

Implementation: cpp

void

UpdateBellAnimation()

{ if (!bellSwinging)

return;

```

// 5 second timer control

bellTimer += deltaTime;

if (bellTimer >=

BELL_DURATION) {

    bellSwinging = false;

    bellTimer = 0.0f;

    bellRotationAngle = 0.0f; // Reset to

    neutral position return;

}


// Bidirectional sinusoidal

oscillation if

(bellSwingDirection) {

    bellRotationAngle += bellRotationSpeed *

    deltaTime * 60.0f; if (bellRotationAngle >=

    MAX_BELL_ANGLE) { bellRotationAngle =

    MAX_BELL_ANGLE;

    bellSwingDirection = false; // Direction change

    }

}

else {

    bellRotationAngle -= bellRotationSpeed *

    deltaTime * 60.0f; if (bellRotationAngle <= -

```

```
MAX_BELL_ANGLE) { bellRotationAngle = -  
MAX_BELL_ANGLE;  
bellSwingDirection = true; // Direction change  
    }  
    }  
}
```

Unified Matrix Transformation:

cpp

```
glm::mat4 bellGroupModel = modelTemp;  
  
if (bellSwinging && abs(bellRotationAngle) > 0.1f) {
```

```

    bellGroupModel = glm::translate(bellGroupModel, BELL_PIVOT);

    bellGroupModel = glm::rotate(bellGroupModel, glm::radians(bellRotationAngle),
        glm::vec3(0.0f, 0.0f, 1.0f));

    bellGroupModel = glm::translate(bellGroupModel, -BELL_PIVOT);
}

```

Configuration Parameters:

- Total Duration: 5.0 seconds (BELL_DURATION)
- Maximum Oscillation Angle: ± 15.0 degrees
- Angular Velocity: 3.0° per frame
- Behavior: Non-interruptible - full execution required
- Activation: Automatic with door animation start

7.5 Animation 3: OPEN/CLOSED Sign (P Key) - Three-Phase

System State Change Mechanics:

- Phase 1: Progressive rotation of $+90^\circ$ around the X axis
- Phase 2: Progressive rotation of $+180^\circ$ around the Y axis
- Phase 3: Progressive rotation of -90° around the X axis
- Final Exchange: Visual model change (let \square let2) upon sequence completion

State Machine Implementation:

cpp

```

void UpdateSignAnimation() {

    if (!signAnimationActive) return;

    switch

    (signAnimationPhase) { case

    1: // Rotation +90° on X axis

```

```
signRotationAngleX += signRotationSpeed *  
deltaTime * 60.0f; if (signRotationAngleX >=  
90.0f) {  
    signRotationAngleX =  
    90.0f;  
    signAnimationPhase =  
    2;  
}
```

```
break;
```

```
case 2: // Rotation +180° on Y axis
```

```
signRotationAngleY += signRotationSpeed *  
deltaTime * 60.0f; if (signRotationAngleY >=
```

```
180.0f) {
```

```
signRotationAngleY = 180.0f;
```

```
signAnimationPhase = 3;
```

```
    }
```

```
break;
```

```
case 3: // Rotation -90° on X axis
```

```
signRotationAngleX -= signRotationSpeed *  
deltaTime * 60.0f; if (signRotationAngleX <= 0.0f)
```

```
{
```

```
signRotationAngleX = 0.0f;
```

```
signRotationAngleY = 0.0f;
```

```
signAnimationActive = false;
```

```
signAnimationPhase = 0;
```

```
showSign1 = !showSign1; // Visual model switching
```

```
    }
```

```
break;
```

```
}
```

```
}
```

Composite Matrix Transformation:

cpp

```
glm::mat4 signGroupModel =  
doorGroupModel; if  
(signAnimationActive) {  
signGroupModel = glm::translate(signGroupModel, glm::vec3(-2.59038f, -0.529044f, -5.78484f));  
signGroupModel = glm::rotate(signGroupModel,  
    glm::radians(signRotationAngleX), glm::vec3(1.0f, 0.0f, 0.0f));
```

```

signGroupModel = glm::rotate(signGroupModel,
    glm::radians(signRotationAngleY), glm::vec3(0.0f, 1.0f, 0.0f));

signGroupModel = glm::translate(signGroupModel, -glm::vec3(-2.59038f, -
    0.529044f, - 5.78484f));
}

```

Behavioral Characteristics:

- Type: Non-interruptible Toggle
- Approximate Total Duration: ~3.0 seconds
- Model Control: Variable showSign1 determines visualization
- Sequentiality: Each phase must be completed before starting the next

7.6 Animation 4: Window System (Q Key) - Four

Synchronized Units Individual Window Configuration:

- Windows 1 and 3: Rotation outwards (negative angle in Y)
- Windows 2 and 4: Rotation inwards (positive angle in Y)
- Temporal Synchronization: Simultaneous movement of the 4 units

Implementation for Window 1 (Exterior):

cpp

```

glm::mat4 window1Model =
modelTemp; if
(windowRotationAngle < 0.0f) {

    window1Model = glm::translate(window1Model, V1_PIVOT);

    window1Model = glm::rotate(window1Model,
        glm::radians(windowRotationAngle), glm::vec3(0.0f, 1.0f, 0.0f));

    window1Model = glm::translate(window1Model, -V1_PIVOT);
}

```

Implementation for Window 2 (Interior):

cpp

```
glm::mat4 window2Model =
```

```
modelTemp; if
```

```
(windowsRotationAngle < 0.0f) {
```

```
    window2Model = glm::translate(window2Model, V2_PIVOT);
```

```

window2Model = glm::rotate(window2Model, glm::radians(-
    windowsRotationAngle), glm::vec3(0.0f, 1.0f, 0.0f));

window2Model = glm::translate(window2Model, -V2_PIVOT);
}

```

State Control Logic:

cpp

```

void UpdateWindowsAnimation() {

    if (!windowsAnimationActive) return;

    if (!windowsAreOpen) {

        // Opening sequence: decreasing angle to -90°

        windowsRotationAngle -= windowsRotationSpeed *

        deltaTime * 60.0f; if (windowsRotationAngle <= -

        90.0f) {

            windowsRotationAngle = -

            90.0f;

            windowsAnimationActive =

            false; windowsAreOpen =

            true;

        }

    }

    else {

        // Closing sequence: increasing angle to 0°
    }
}

```

```
windowsRotationAngle += windowsRotationSpeed *  
deltaTime * 60.0f; if (windowsRotationAngle >= 0.0f)  
{  
    windowsRotationAngle = 0.0f;  
    windowsAnimationActive =  
    false; windowsAreOpen =  
    false;  
}  
}  
}
```

Technical Parameters:

- Angular Range: 0° (fully closed) □ -90° (fully open)
- Behavior: Non-interruptible Toggle
- Individual Pivots: V1_PIVOT, V2_PIVOT, V3_PIVOT, V4_PIVOT

7.7 Animation 5: Curtain System (N Key) - Scaling

from Pivots Scale Transformation Mechanics:

- Closed State: Scale X = 1.0 (original dimension)
- Open State: Scale X = 3.5 (horizontal expansion)
- Independent Pivots: Specific anchor points for each curtain
- Directionality: Expansion/contraction from fixed points

Implementation for Curtain 1 (Right Pivot):

cpp

```
glm::mat4 cortina1Model =
modelTemp; if (cortinasScaleX
!= 1.0f) {
cortina1Model = glm::translate(cortina1Model,
CORTINA1_PIVOT); cortina1Model = glm::scale(cortina1Model,
glm::vec3(cortinasScaleX, 1.0f, 1.0f)); cortina1Model =
glm::translate(cortina1Model, -CORTINA1_PIVOT);
}
```

Implementation for Curtain 2 (Left Pivot):

cpp

```
glm::mat4 cortina2Model =
modelTemp; if (cortinasScaleX
!= 1.0f) {
```

```
cortina2Model = glm::translate(cortina2Model,  
  
CORTINA2_PIVOT); cortina2Model = glm::scale(cortina2Model,  
  
glm::vec3(cortinasScaleX, 1.0f, 1.0f)); cortina2Model =  
  
glm::translate(cortina2Model, -CORTINA2_PIVOT);  
  
}
```

Scaling Control Logic:

cpp

```
void UpdateCortinasAnimation() {  
  
if (!cortinasAnimationActive) return;
```

```
if (!curtainsAreScaled) {  
  
    // Opening sequence: progressive increase in  
  
    scale curtainsScaleX += curtainsScaleSpeed *  
  
    deltaTime;  
  
    if (curtainsScaleX >=  
  
    curtainsTargetScale) {  
  
        curtainsScaleX =  
  
        curtainsTargetScale;  
  
        curtainsAnimationActive = false;  
  
        curtainsAreScaled = true;  
  
        }  
    }  
else {  
  
    // Closing sequence: progressive decrease in  
  
    scale curtainsScaleX -= curtainsScaleSpeed *  
  
    deltaTime;  
  
    if (curtainsScaleX <= 1.0f) {  
  
        curtainsScaleX = 1.0f;  
  
        curtainsAnimationActive =  
  
        false; curtainsAreScaled =  
  
        false;  
  
        }  
}
```

```
}  
}
```

Configuration Parameters:

- Scaling Speed: 2.0 units per second
- Target Scale: 3.5x on X axis (horizontal expansion)
- Specific Pivots:
 - o $CORTINA1_PIVOT = (4.89717f, 11.3683f, -5.17732f)$
 - o $CORTINA2_PIVOT = (-10.2291f, 11.4903f, -5.18742f)$
- Behavior: Non-interruptible Toggle

7.8 State Management and Control

System Global State Variables:

cpp

// Animation activity states

bool doorAnimationActive; // Door (with reversion)

bool windowsAnimationActive; // Windows

(one-way toggle) bool signAnimationActive; //

Sign (state machine)

bool cortinasAnimationActive; // Curtains

(scaling toggle) bool bellSwinging; // Bell

(automatic timed)

// Current configuration states

bool doorOpening; // Current door

direction bool windowsAreOpen;

// Current window

state bool cortinasAreScaled;

// Current curtain

state bool showSign1; // Current sign

model Frame Update System:

cpp

// Main animation update loop

UpdateDoorAnimation(); // Update

with reversion

UpdateWindowsAnimation(); // Windows

```
toggle update UpdateBellAnimation();  
  
                // Timed bell update  
  
UpdateSignAnimation();      // Sign state  
  
machine update UpdateCortinasAnimation();  
  
// Curtains toggle update Complete
```

Keyboard Control Mapping:

- Key R: Main door (reversion system)
- Key P: OPEN/CLOSED sign (three-phase toggle)
- Key Q: Windows (open/close toggle)
- Key N: Curtains (horizontal scaling toggle)
- Bell: Automatic activation with door (5 seconds)

6. Model and Resource Management

6.0. Object Selection

Object selection:

Ground Floor

1. Main door:

Imported directly from the game itself, someone uploaded several objects from the game and converted them to .obj. I mostly used these since they were already optimized for use in video games and were actually quite low in their polygon count. Several had to undergo modifications despite this, such as transformations and adaptations to the facade.



Figure 1.1 Reference image



Figure 1.2 Model in Blender

This was supposed to have a transparency effect, which was achieved by separating the objects that should have it from those that should not, drawing them in the rendering code afterwards:



Figure 1.3, Sample of object separation in Blender

2. Leblanc Display Case

Also taken from the game, it maintains high fidelity, it was edited so it wouldn't mess with the aesthetic



Figure 2.1 Display inside the game

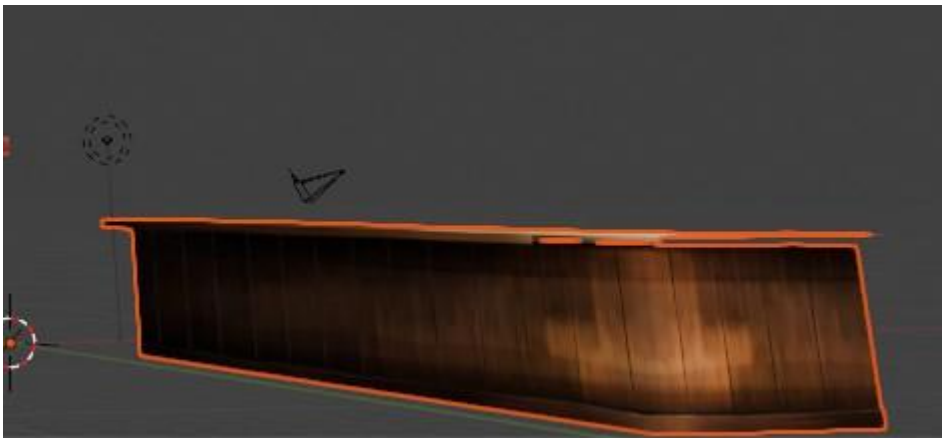


Figure 2.2 Display edited in blender

3.Shelf with cups and jars

For this one, from the internet, I took the model of the jars, later I modeled around them:

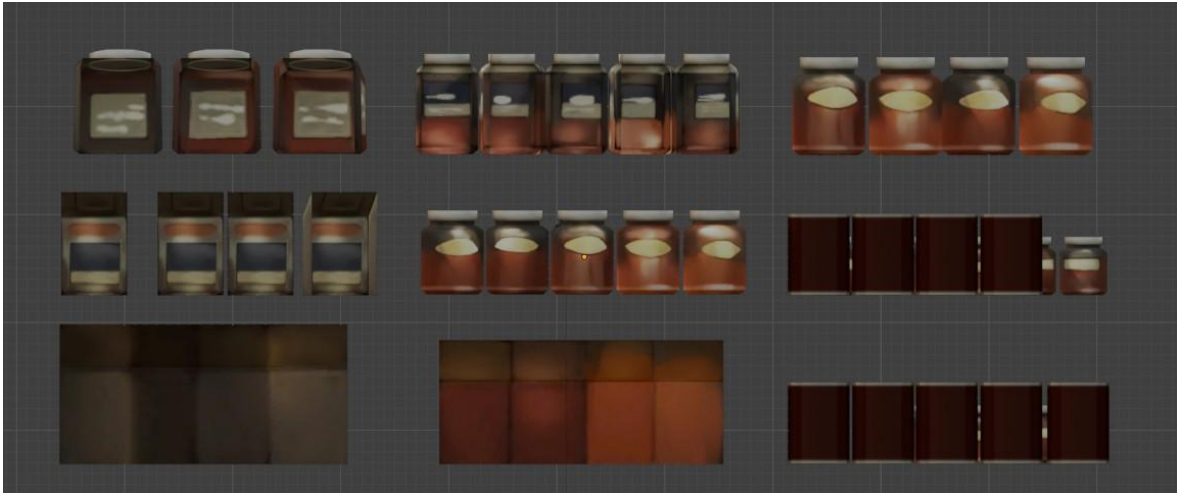


Figure 3.1 Jar model in blender



Figure 3.2 Wooden shelf modeled from them

4. Bell

Again from cgtrader, I got a lowpoly model of a bell, to use as the one that appears on the game door, this model is divided into 3: the holder, the clapper and the bell, this in order to achieve the desired animation that every time the door is opened the bell moves

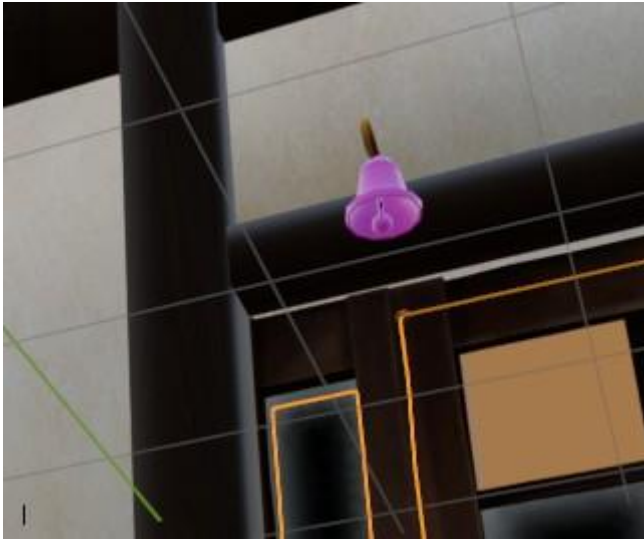


Figure 4.1 Bell in Blender, no texture

5. Armchairs

From the Sketchfab platform, I found a recreation of the models in question. After reducing the polygon count and arranging them, I incorporated them into the scene:

6. Tables

Also from Sketchfab, I got some very similar ones and only had to do a polygonal reduction and arrange them

7. Counter desk and lamps:

Taken directly from the CG Trader representation, with all the imported objects. I took it from there and adjusted its size

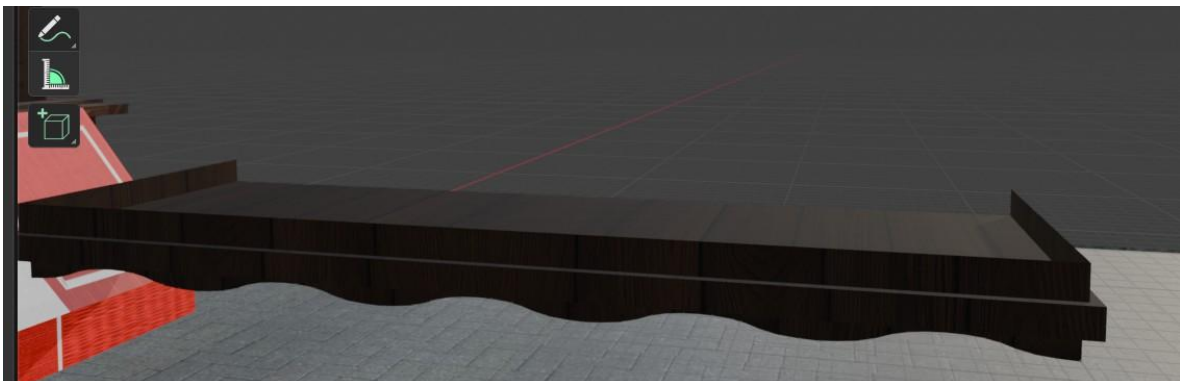


Figure 7.2 Blender modeling of the counter corner

The lamps again were a CGTrader model that I considered suitable



Figure 7.1 lamps in blender

8. Multicolor lamps above the tables

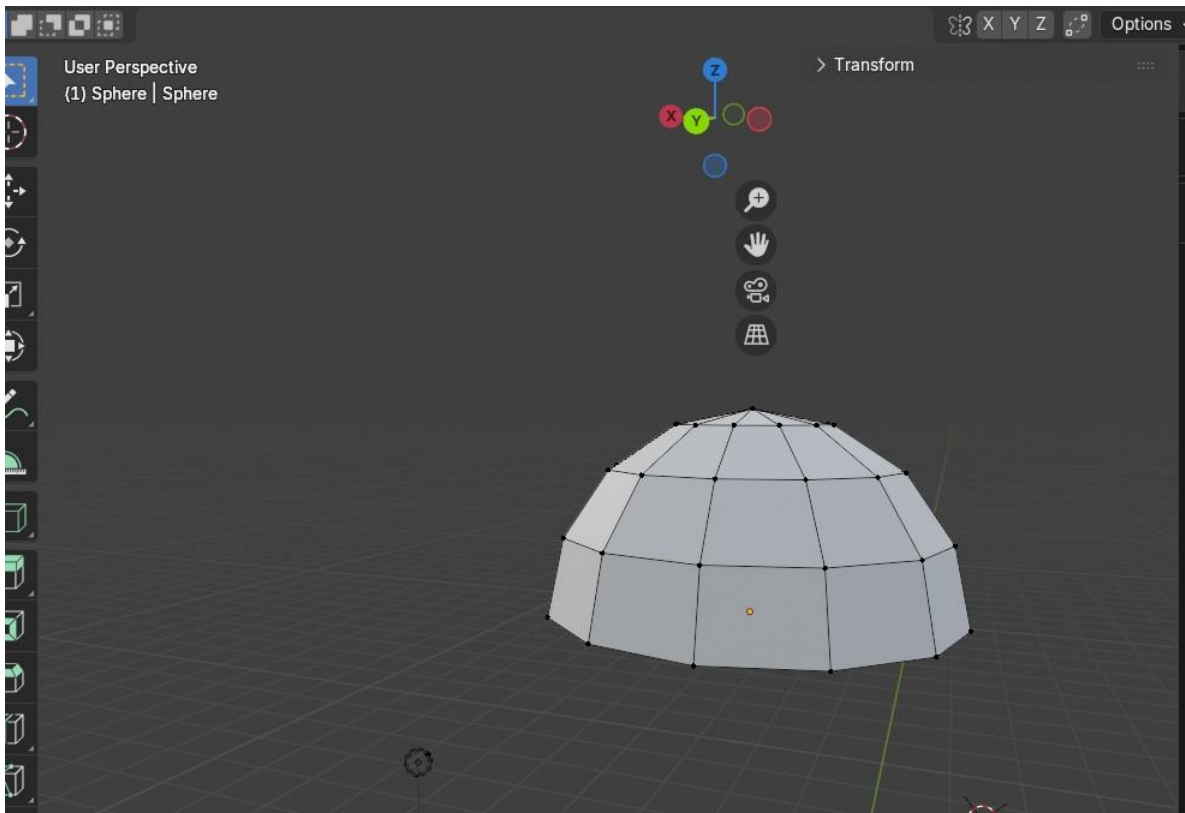


Figure 8.1 UV sphere cropped

By creating a UV sphere in blender, with 12 separations and 8 rings, I created the general shape, then removed the lower vertices to make the shape, as the processing form of the project separates between solid objects and with transparency, the glass lamp material must be separate from its solid frame, the cable on which it is suspended, and the bulb

For the creation of these solid parts: in the case of the wooden frame: I took the aforementioned object and applied a curve cut (I in faces mode) of 0.04m, when removing the resulting face, a frame came out

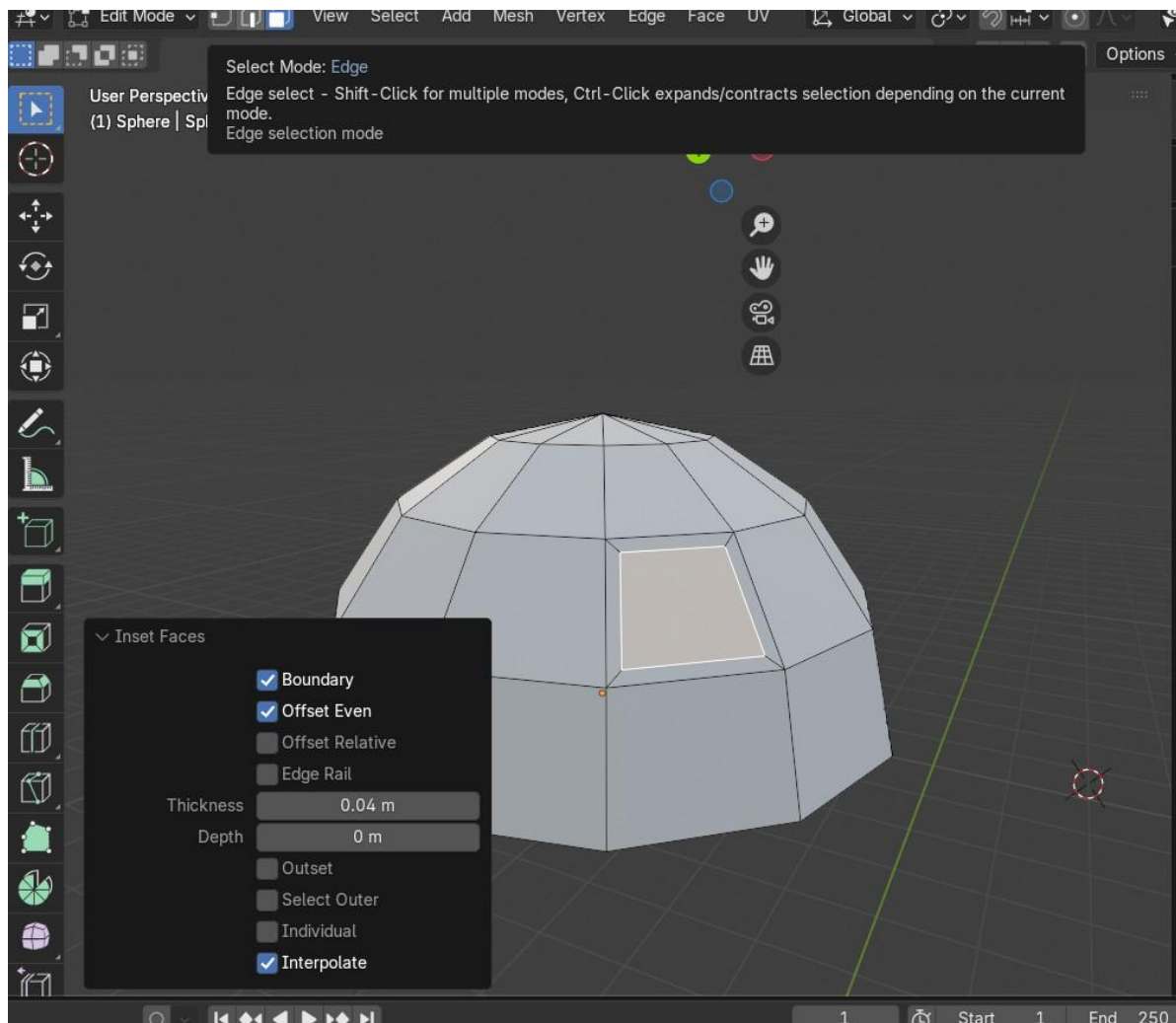
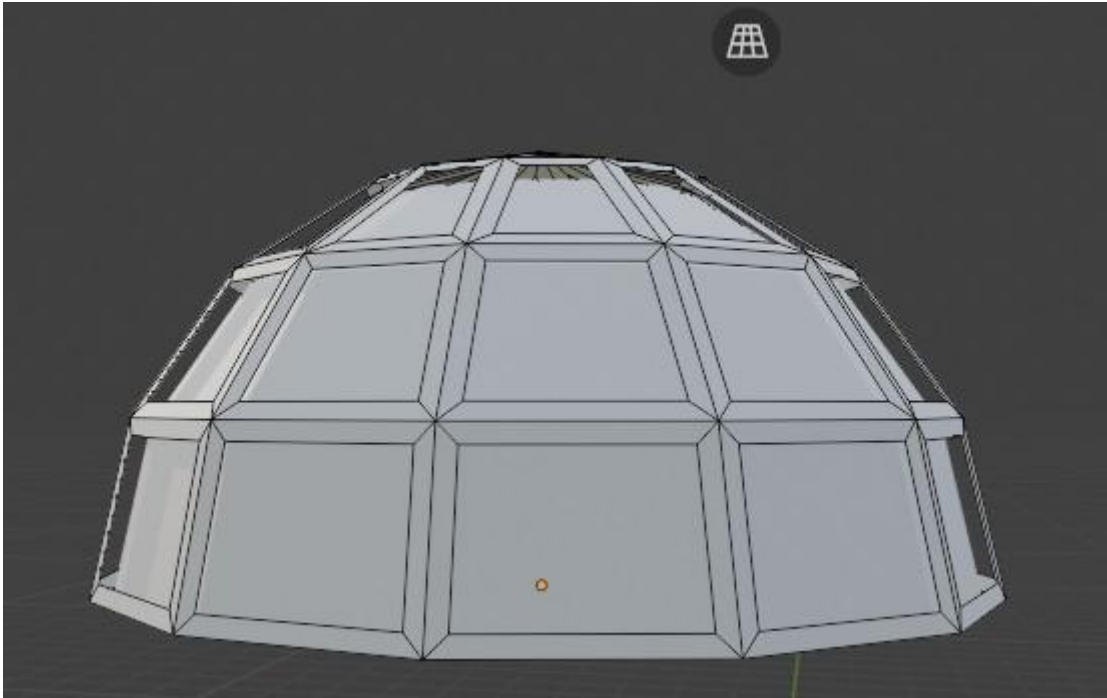


Figure 8.2 inset faces function used

Subsequently, we joined these 2 models, textured them, and, borrowing from the previously imported models from the game, we took the spotlight and lamp:



This joins the group of elements that will be translucent, achieved by modifying the Alpha within OpenGL. Once in the project, this model is separated into 2: the rope with its focus and its respective crystal.



Figure 8. Final product

Upstairs:

All of these were taken from the game, downloaded from the page I mentioned earlier

-Bed on beer crates

-Work desk

-Curtains and Pole

-Non-translucent windows

-shelving



Figure 8, Objects mentioned above in the project

6.1 Inventory of 3D Models

6.1.1 Opaque Static

Models Architectural

Structure:

- facade: Main structure of the Café Leblanc building
- floor: Base surface of the ground floor

- upstairsfloor: Base surface of the upper floor
- awning: Exterior decorative element above entrance
- counter: Complementary counter and structural elements

Ground Floor Furniture:

- desk, desk2: Main desks in the cafe area

- shelving: Shelving for ingredient jars
- tables: Tables for customers
- armchairs: Waiting area armchairs
- chairs: Individual chairs for customers
- most: Main service counter
- stove: Kitchen stove
- cafe: Professional coffee machine

Upstairs Furniture:

- escsup: Study desk in room
- cama: Protagonist's single bed
- sillsup: Individual reading armchair
- cort: Rod or structure for curtains

Decorative Elements:

- frascos: Jars of ingredients on shelving
- sujeta: Hood holder (structural element)
- lamp2: Base and structure of lamps
- focos: Opaque lighting elements
- luces: Lighting and luminaire system

6.1.2 Opaque Animated

Models Elements with

Transformations:

- puerta: Main door with rotation animation
- campana: Hood body with automatic oscillation
- badajo: Bell clapper with synchronized movement
- v1, v2, v3, v4: Four windows with opening/closing animation
- cort1, cort2: Two curtains with scaling animation

- let, let2: OPEN and CLOSED signs with exchange system

6.1.3 Translucent Models

Glass and Transparent Surfaces:

- vidrio: Glass of the main door (with animation)

- glass2: Fixed facade glass
- vidlamp: Colored glass for decorative lamps

6.2 Creation Process in Blender

6.2.1 Modeling Workflow in

Blender Blockout Phase:

- Establishing proportions using geometric primitives
- Referencing images of Café Leblanc from Persona 5
- Definition of scale and spatial relationships

Detailed Modeling:

- Extrude: For extruded geometry and basic volumes
- Bevel: For smoothing edges and corners
- Insert Edge Loop: For topological control and shape definition
- Multi-Cut: For architectural and ornamental details

Geometry Optimization:

- Polygon Target: 500-2000 polygons per object
- Merge Vertices: To reduce unnecessary complexity
- Delete Edge/Vertex: Topology cleaning
- Triangulate: Automatic conversion for export

6.2.2 Texture and Materials

System Texture Sources:

- Freepik: High quality photographic textures
 - Red leather texture for armchairs
 - Brown wood floor texture
- MyArchitectAI: Textures generated by AI
 - <https://www.myarchitectai.com/texture-generator>

- Architectural materials and surfaces
- Persona 5 Assets: Direct references from the video game
 - Characteristic visual style of Café Leblanc
 - Color palette and materials faithful to the original

Texture Processing:

- Formats: PNG with lossless compression
- Resolutions: 512×512, 1024×1024 depending on visual importance
- Color Correction: White balance and level adjustment
- Seamless Tiling: Preparation for repeatability on surfaces

6.2.3 Pivot Configuration for

Animation Pivoting

Methodology:

1. Rotation Point Selection: Identification of the natural rotation point
2. Move Origin in Blender: Moving the object's origin to the pivot point
3. Export with Modified Origin: Preservation of the pivot transformation
4. Coordinate Conversion: Applying

Blender→OpenGL formula Calculated Pivots:

cpp

```
// Converted main pivots
```

```
const glm::vec3 DOOR_PIVOT = glm::vec3(-4.350004f, -1.229998f, -  
5.860039f); const glm::vec3 BELL_PIVOT = glm::vec3(-1.13727f,  
1.61114f, -6.40422f); const glm::vec3 V1_PIVOT = glm::vec3(-  
9.390120f, 10.340143f, -3.639997f); const glm::vec3 V2_PIVOT =  
glm::vec3(-3.039997f, 10.340143f, -3.639997f); const glm::vec3  
V3_PIVOT = glm::vec3(-1.729999f, 10.340143f, -3.639997f);  
const glm::vec3 V4_PIVOT = glm::vec3(4.180001f, 10.340143f, -  
3.639997f); const glm::vec3 CORTINA1_PIVOT =
```

```
glm::vec3(4.89717f, 11.3683f, -5.17732f); const glm::vec3
```

```
CORTINA2_PIVOT = glm::vec3(-10.2291f, 11.4903f, -5.18742f);
```

6.2.4 Export

Configuration OBJ

Export Parameters:

- Format: OBJ Wavefront for maximum compatibility with Assimp
- Units: Blender units maintaining relative scale
- Vertical Axis: +Y according to Blender convention
- Triangulation: Automatic to prevent problems with quads
- Smoothing Groups: Exported to preserve normals

- Normals: Calculated and exported to avoid recalculation

6.3 Resource Loading and Management System

6.3.1 Model Initialization

Declaration and Loading:

cpp

// Structural modeling

Model

facade((char*)"Models/fachada.obj");

Model

floor((char*)"Models/piso.obj");

Model

awning((char*)"Models/toldo.obj"

);

// Animated elements

Model

door((char*)"Models/puerta.obj");

Model

bell((char*)"Models/camp.obj");

Model

clapper((char*)"Models/bad.obj");

// Translucent elements

Model

glass((char*)"Models/vidrio.obj");

Model

glass2((char*)"Models/vidrio2.obj");

Model glasslamp((char*)"Models/vidriolamp.obj");

6.3.2 Scene

Organization Logical

Grouping:

- Static Models: Geometry without dynamic transformations
- Animated Models: Objects with transformations at runtime
- Translucent Models: Objects that require alpha blending
- Transformation Hierarchies: Groups that inherit parent transformations

Rendering by Categories:

1. Static Opaque Objects: Initial rendering with depth write
2. Animated Opaque Objects: Rendering with transformations applied
3. Translucent Objects: Final rendering with blending enabled

6.4 Resource Optimization

6.4.1 Performance

Strategies Shader

Reuse:

- A single main shader for all opaque objects
- Modification of uniforms for specific properties
- Specialized shader for translucent elements

Draw Calls Management:

- Minimization of OpenGL state changes
- Grouping by material type and properties
- Efficient sorting by rendering state

6.4.2 Memory

Consumption VRAM

Usage Estimation:

- 3D Models: ~50-100MB total
- Textures: ~20-30MB (optimized resolutions)
- OpenGL Buffers: ~10-20MB
- Estimated Total: < 150MB VRAM

Applied Optimizations:

- Low-poly geometry for distant objects
- Reuse of materials and shaders
- Textures in balanced resolutions (512x512 - 1024x1024)

8. Cost Analysis

8.1 Labor Costs

Activity	Hours	Rate/Hour (MXN)	Subtotal (MXN)
Analysis and design	8	\$350	\$2,800
Blender Modeling (structure)	20	\$380	\$7,600
Blender Modeling (furniture)	24	\$380	\$9,120
Activity	Hours	Rate/Hour (MXN)	Subtotal (MXN)
UV Mapping	12	\$330	\$3,960
Textures and Materials	8	\$280	\$2,240
C++ core programming	24	\$480	\$11,520
Lighting	8	\$420	\$3,360
Animations	14	\$420	\$5,880
Testing	10	\$380	\$3,800
Technical documentation	12	\$330	\$3,960
User documentation	6	\$280	\$1,680

Total	146 hrs	-	\$55,920
-------	---------	---	-----------------

8.2 Software Licenses

Software	Annual Cost (MXN)	Project Use (%)	Prorated Cost (MXN)
Blender	\$0	N/A	\$0
Visual Studio 2022	\$0	N/A	\$0
GIMP	\$0	N/A	\$0
Windows 10 Pro	\$3,500	20%	\$700
Total	-	-	\$700

8.3 Hardware and Depreciation

Component Life	Cost (MXN)	Useful	Annual Deprec. (MXN)
Laptop (Ryzen 5 7525HS, 32GB DDR5, RTX 2050)	\$22,000	4 years	\$5,500
SSD 1TB NVMe	\$2,000	5 years	\$400
Peripherals	\$2,500	3 years	\$833
Total	\$26,500	-	\$6,733/year
Project depreciation (2 months):	$\$6,733 / 12 \times 2 = \$1,122$		

8.4 Operating Costs

Concept	Calculation	Cost (MXN)
Electric energy	$146 \text{ hrs} \times 0.12 \text{ kW} \times \$3.50/\text{kWh}$	\$61
Internet	$2 \text{ months} \times \$450/\text{month}$	\$900
Workspace	$2 \text{ months} \times \$800/\text{month}$	\$1,600
Total	-	\$2,561

8.5 Indirect Costs

Concept	Base	Cost (MXN)
Administration	7% labor	\$3,914
Contingencies	4% labor	\$2,237

Cloud repository

\$300

Fixed

Concept	Base	Cost (MXN)
Total	-	\$6,451

8.6 Summary and Final

Price Total Costs:

Category	Subtotal (MXN)	Percentage
Labor	\$55,920	84.2%
Software Licenses	\$700	1.1%
Equipment Depreciation	\$1,122	1.7%
Operation	\$2,561	3.9%
Indirect Costs	\$6,451	9.7%
Total Cost	\$66,754	100%

Price Calculation with Profit:

Profit margin: 32% (adjusted for
competitiveness) Base Price = \$66,754 / (1
- 0.32) = \$98,168

Gross Profit = \$31,414

ISR (10% on profit) =

\$9,817 Net Profit =

\$21,597

Final Price to Customer:

Subtotal =

\$98,168 IVA (16%)

= \$15,707

Final Price = \$113,875

MXN Rounded: \$115,000

MXN

Percentage Breakdown of Final Price:

Concept	Amount (MXN)	% of Total
Labor	\$55,920	49.1%
Licenses	\$700	0.6%

Hardware	\$1,122	1.0%
Operation	\$2,561	2.2%
	\$6,451	5.7%
Indirect Costs		
Net Profit	\$21,597	19.0%
	\$9,817	8.6%
ISR (Inc om e Tax)		
V A T (I V A)	\$15,707	13.8%
TOTAL	\$113,875	100%

8.7 Value Comparison

Solution	Cost (MXN)	Advantages	Limitations
Unity/Unreal Template	\$50,000- \$80,000	Rapid development	Runtime licenses
360° Tour	\$30,000- \$50,000	Photorealism	No interactivity

BIM Visualization	\$80,000- \$120,000	Technical precision	No lighting dynamics
This Project	\$115,000	Total control, code proprietary	Higher investment initial

Competitive Advantages of This Project:

- Reduced Cost: 50% less than project due to use of free software
- Reused Assets: Freepik and AI textures reduce creation costs
- No Runtime Licenses: Significant savings in distribution
- 100% Proprietary Code: No third-party dependencies

9. Technical Dictionary

9.1 Main Functions

Function	Parameters	Return	Description
----------	------------	--------	-------------

main()	void	int	<p>Main entry point of the program.</p> <p>Manages GLFW initialization, window creation, OpenGL context configuration, 3D resource loading, and execution of the main rendering loop.</p> <p>principal de renderizado.</p>
KeyCallback()	window, key, scancode, action, mode	void	<p>GLFW callback function for keyboard input management.</p> <p>Updates the key state array and executes the activation triggers of animations.</p> <p>animaciones.</p>

MouseCallback()	window, xPos, yPos	void	<p>Callback for processing mouse movement. Calculates movement offsets and updates the camera's orientation in real time.</p> <p>movement of the mouse.</p> <p>Calculates offsets of movement and updates the orientation of the camera in real real time.</p>
DoMovement()	void	void	<p>Logical update function executed per frame. Processes camera movement based on user input and updates the state of animations using deltaTime.</p> <p>per frame. Processes the movement of the camera based on input from user and updates the state of the animations using deltaTime.</p>

9.2 Camera Class Methods

Method	Parameters	Return	Description
GetViewMatrix()	void	glm::mat4	Calculates and returns the view matrix using the lookAt function from GLM, considering the camera's current position and orientation. view using the lookAt function of GLM, considering position and current orientation of the camera.

ProcessKeyboard()	direction, deltaTime	void	Processes camera movement in the specified direction, scaled by camera in the direction specified, scaled by
-------------------	-------------------------	------	---

			deltaTime to maintain consistent speed independent of framerate.
ProcessMouseMoveMent()	xoffset, yoffset, constrainPitch	void	Updates Yaw angles and Pitch based on mouse movement. Limits mouse. Limits the Pitch to $\pm 89^\circ$ if constrainPitch is true.
ProcessMouseScroll()	yoffset	void	Modifies the Field of View (FOV) based on the scroll direction of the mouse, allowing zoom dynamically.

GetZoom()	void	float	Returns the current value of Field of View in degrees.
GetPosition()	void	glm::vec3	Returns the current position of the camera in coordinates of world.

9.3 Model Class Methods

Method	Parameters	Return	Description
Model()	path	-	Class constructor. Loads a 3D model from file using the Assimp library for import.
Draw()	shader	void	Renders all model meshes: activates textures, binds VAOs and executes glDrawElements() for each mesh.
loadModel()	path	void	Private method that initializes loading of the model with Assimp post-processing flags of Assimp.

processNode()	node, scene	void	Private method that processes nodes in a recursive way to load the hierarchy complete model.
processMesh()	mesh, scene	Mesh	Private method that converts an Assimp aiMesh structure to the internal mesh structure of the

			program.
--	--	--	----------

9.4 Main Animation Variables

Variable	Type	Description
doorAnimationActive	bool	Controls the activity state of the door animation (Key R)
doorRotationAngle	GLfloat	Current rotation angle of the door in the range [0°, 90°]
doorOpening	bool	Current direction of the door animation (true = opening, false = closing)
windowsAnimationActive	bool	Activity state of the windows animation (Key Q)
windowsRotationAngle	GLfloat	Current rotation angle of the windows [-90°, 0°]
windowsAreOpen	bool	Current state of the windows (true = open, false = closed)
bellSwinging	bool	Controls the swinging animation of the bell
bellRotationAngle	GLfloat	Current swing angle of the bell [-15°, 15°]
bellTimer	GLfloat	Timer for the 5-second duration of the bell
signAnimationActive	bool	Activity state of the sign animation (Key P)
signAnimationPhase	int	Current phase of the sign animation [0-3]
showSign1	bool	Controls which sign model is displayed (true = OPEN, false = CLOSED)
cortinasAnimationActive	bool	Activity state of the curtain animation (Key N)
cortinasScaleX	GLfloat	Current scale factor of the curtains on the X axis [1.0, 3.5]
cortinasAreScaled	bool	Current state of the curtains (true = open, false = closed)
camera	Camera	Camera instance with initial position (0, 0, 3) outside the cafe
deltaTime	GLfloat	Time between frames in seconds for F-independent movement
keys	bool[1024]	Array of key states for user input
pointLightPositions	glm::vec3[10]	Positions of the 10 point lights in 3D space

10. Troubleshooting

Problem: Completely black screen when running

- Solution: Verify shader compilation in the output console, adjust the camera's initial position if necessary
initial camera position if necessary, confirm that the Models/, Textures/ and SkyBox/ folders are in the same directory as the executable.

Problem: 3D models are not visible in the scene

- Solution: Validate the relative paths of the .obj files, apply temporary scale transformations
temporary scaling transformations for debugging, temporarily disable face culling using `glDisable(GL_CULL_FACE)`.

Problem: Performance below 30 FPS

- Solution: Disable vertical sync (V-Sync) in the GPU control panel, reduce the level of detail of secondary models, resize textures to a maximum of 1024×1024 pixels.

Problem: Transparencies rendered incorrectly

- Solution: Verify the rendering order (opaque objects ☐ enable blending ☐ objects transparent), confirm the blending function: `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)`.

Problem: Animations are not activated with assigned keys

- Solution: Make sure the application window has the input focus, verify the correct key mapping in `KeyCallback`, confirm that there are no conflicts with other background applications.

11. System Requirements

Minimum Requirements:

- Operating System: Windows 10 64-bit
- Processor: Intel Core i3 or AMD equivalent
- RAM Memory: 4 GB
- Graphics Card: OpenGL 3.3+ compatible GPU (GTX 650, Radeon HD 7750, Intel HD 4000)
- VRAM: 1 GB
- Storage: 500 MB of available space
- Resolution: 1280×720 pixels

Recommended Requirements:

- Operating System: Windows 10/11 64-bit
- Processor: Intel Core i5 or Ryzen 5
- RAM Memory: 8 GB
- Graphics Card: GTX 1050 or RX 560
- VRAM: 2 GB
- Storage: 1 GB of available space
- Resolution: 1920×1080 pixels

Software Dependencies:

- Graphics Libraries: GLEW 3.x, GLFW 2.x, GLM 0.9.9+
- Asset Loading: SOIL2, stb_image, Assimp
- Development Environment: Visual Studio 2022
- SDK: Windows SDK 10.0+
- Runtime: C++ Redistributable 2022

12. Conclusions

The development of this 3D visualization system of Café Leblanc represented a significant technical challenge that allowed applying the knowledge acquired in computer graphics in an integral way. Throughout the project, I was able to verify that the use of OpenGL as a native API is still completely viable to develop quality graphic applications, offering total control over the rendering pipeline that higher-level engines do not allow.

One of the most valuable lessons was the implementation of the animation system through matrix transformations around specific pivots. The process of calculating the correct coordinates in Blender and then converting them to the OpenGL coordinate system was fundamental to achieve realistic and natural animations.

Particularly, the door reversal system demonstrated how a well-designed state logic can create complex interactions with relatively simple code.

The lighting system with 10 simultaneous light sources, although demanding, proved to be manageable on modern hardware, maintaining a smooth performance above 60 FPS. The strategic distribution of the lights not only fulfilled a technical function, but also contributed to reinforcing the visual narrative of the space, contrasting the warmth of the coffee with the austerity of the upper room.

Working with Blender as the main modeling tool turned out to be an excellent decision, not only for its zero cost but also for the robustness of its export workflow. The conscious separation between the opaque and translucent components of the models greatly facilitated the implementation of the transparency system and the correct rendering order.

Although the project satisfactorily met all the objectives set,

I recognize that the lack of a collision system represents an important limitation in the user experience. In retrospect, dedicating time to implementing a basic collision detection system would have significantly improved the usability of the virtual tour.

This work not only allowed me to consolidate my knowledge in computer graphics, but also to develop valuable skills in project management, resource optimization and technical documentation. The final result demonstrates that it is possible to create immersive 3D experiences with open source tools and a well-founded technical approach, establishing a solid foundation for future projects of greater magnitude.

Referencias/Bibliography

ApprenticeRacon. (s. f.). *Golden bell* [Modelo 3D]. CGTrader. Recuperado el 7 de noviembre de 2025, de <https://www.cgtrader.com/free-3d-models/architectural/decoration/golden-bell-89c7c3ea-037c-442e-9868-e22bee58f496>

DeepSeek AI. (2025). *Asistencia en desarrollo y documentación técnica* [Modelo de lenguaje artificial]. DeepSeek. Consultas realizadas entre noviembre y diciembre de 2025 para la elaboración de arquitectura del sistema, documentación técnica y optimización de código.

Freestock center. (s. f.). *Textura de cuero rojo* [Fotografía]. Freepik. Recuperado el 7 de noviembre de 2025, de https://www.freepik.es/foto-gratis/textura-cuero-rojo_1006129.htm

maxdragonn. (2021, 15 diciembre). *Coffee machine* [3D model]. Sketchfab. Recuperado el 7 de noviembre de 2025, de <https://sketchfab.com/3d-models/coffee-machine-6784e4f16a7248518cd5ef1118a2198a>

MyArchitectAI. (s. f.). *Texture generator* [Generador de IA]. MyArchitectAI. Recuperado el 7 de noviembre de 2025, de <https://www.myarchitectai.com/texture-generator>

Rawpixel. (s. f.). *Fondo de pisos de madera textura marrón* [Fotografía]. Freepik. Recuperado el 7 de noviembre de 2025, de https://www.freepik.es/foto-gratis/fondo-pisos-madera-textura-marron_16249980.htm

Concrete paving outdoor texture seamless [Textura]. (s. f.). Sketchup Texture Club. Recuperado el 7 de noviembre de 2025, de <https://www.sketchuptextureclub.com/public/texture/68-concrete-paving-outdoor-texture-seamless.jpg>

Suto, M., Maeda, N., y Usuda, M. (Diseñadores de personajes 3D). (2016). *Persona 5* [Videojuego]. Atlus. Modelos extraídos de The Spriter's Resource el 7 de noviembre de 2025: https://models.sprisers-resource.com/playstation_4/persona5/asset/326853/