



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

COMPUTACIÓN GRÁFICA E INTERACCIÓN

HUMANO-COMPUTADORA



PROFESOR: ING. CARLOS ALDAIR ROMÁN BALBUENA

**PROYECTO 2 FINAL  
(MANUAL TÉCNICO)**

ALUMNO: 318328278

GRUPO: 05

FECHA DE ENTREGA:

25 DE NOVIEMBRE DE 2025

SEMESTRE

2026-1

## ÍNDICE

<b>1. MANUAL TÉCNICO EN ESPAÑOL.....</b>	<b>4</b>
1.1. INTRODUCCIÓN.....	4
1.2. OBJETIVOS DEL PROYECTO.....	4
1.3. PLANEACIÓN Y DESARROLLO.....	5
1.3.1. METODOLOGÍA DE SOFTWARE APLICADA.....	5
1.3.2. DIAGRAMA DE GANTT.....	7
1.3.3. ALCANCE DEL PROYECTO.....	8
1.3.4. LIMITANTES.....	8
1.4. DESARROLLO TÉCNICO.....	10
1.4.1. OBJETIVOS Y ALCANCES TÉCNICOS.....	10
1.4.2. MODELADO.....	10
1.4.3. TEXTURIZACIÓN Y MAPEADO UV.....	12
1.4.4. ESTRUCTURA DEL CÓDIGO.....	14
1.4.5. CARGA DE MODELOS.....	19
1.4.5.1. SKYBOX.....	20
1.4.6. CÁMARA SINTÉTICA.....	21
1.4.7. ILUMINACIÓN.....	22
1.4.8. ANIMACIONES.....	22
1.4.8.1. SIMPLES.....	23
1.4.8.2. COMPLEJAS.....	24
1.4.9. DIAGRAMA DE FLUJO DEL SISTEMA.....	27
<b>2. REFERENCIAS VISUALES.....</b>	<b>28</b>
2.1. COMPARATIVA IMÁGENES DE REFERENCIA.....	28
2.2. OBJETOS RECREADOS.....	32
2.2.1. FACHADA (CASA DE BETTY).....	32
2.2.2. COMEDOR DE LA CASA DE BETTY.....	32
2.2.3. RECÁMARA DE BETTY.....	34
<b>3. ANÁLISIS DE COSTOS.....</b>	<b>36</b>
3.1. MATERIALES.....	36
3.2. SOFTWARE.....	36

3.3. MANO DE OBRA.....	36
3.4. INFRAESTRUCTURA.....	36
3.5. COSTO TOTAL.....	37
<b>4. REPOSITORIO EN GITHUB.....</b>	<b>37</b>
<b>5. CONCLUSIONES.....</b>	<b>38</b>
<b>6. REFERENCIAS.....</b>	<b>39</b>
<b>7. ENGLISH TECHNICAL MANUAL.....</b>	<b>40</b>
7.1. INTRODUCTION.....	40
7.2. PROJECT OBJECTIVES.....	40
7.3. PLANNING AND DEVELOPMENT.....	41
7.3.1. APPLIED SOFTWARE METHODOLOGY.....	41
7.3.2. GANTT CHART.....	43
7.3.3. PROJECT SCOPE.....	44
7.3.4. LIMITATIONS.....	44
7.4. TECHNICAL DEVELOPMENT.....	46
7.4.1. TECHNICAL OBJECTIVES AND SCOPES.....	46
7.4.2. MODELING.....	46
7.4.3. TEXTURING AND UV MAPPING.....	48
7.4.4. CODE STRUCTURE.....	50
7.4.5. MODEL LOADING.....	55
7.4.5.1. SKYBOX.....	56
7.4.6. SYNTHETIC CAMERA.....	57
7.4.7. LIGHTING.....	58
7.4.8. ANIMATIONS.....	58
7.4.8.1. SIMPLE ONES.....	59
7.4.8.2. COMPLEX ONES.....	60
7.4.9. SYSTEM FLUX DIAGRAM.....	63
<b>8. VISUAL REFERENCES.....</b>	<b>64</b>
8.1. COMPARATIVE REFERENCE IMAGES.....	64
8.2. RECREATED OBJECTS.....	68
8.2.1. FACADE (BETTY'S HOUSE).....	68

8.2.2. BETTY'S HOUSE DINING ROOM.....	68
8.2.3. BETTY'S BEDROOM.....	70
<b>9. COST ANALYSIS.....</b>	<b>72</b>
9.1. MATERIALS.....	72
9.2. SOFTWARE.....	72
9.3. LABOR.....	72
9.4. INFRASTRUCTURE.....	72
9.5. TOTAL COSTS.....	72
<b>10. GITHUB REPOSITORY.....</b>	<b>73</b>
<b>11. CONCLUSIONS.....</b>	<b>74</b>
<b>12. REFERENCES.....</b>	<b>75</b>

## 1. MANUAL TÉCNICO EN ESPAÑOL

### 1.1. INTRODUCCIÓN

Este Manual Técnico contempla las técnicas básicas de computación gráficas vistas a lo largo de la materia, para identificar la diversidad de áreas de aplicación. Se hizo uso de varias herramientas, entre ellas, de modelado 3D y edición de imágenes, para finalmente integrarlas al entorno de programación de OpenGL. Como producto final, se recreó virtualmente una casa de dos pisos junto con un cuarto dentro de ésta.

La ejecución de este proyecto requiere la aplicación de una metodología de software, así como el modelado de cinco objetos en dos cuartos, la implementación de mínimo cuatro animaciones contextualizadas, el manejo de texturas e iluminación para alcanzar un resultado lo más acercado a un escenario realista (contra las imágenes de referencia de esta documentación) para implementar una aceptable interacción humano-computadora.

Adicionalmente, se incluirá un análisis de costos para estimar los parámetros financieros del desarrollo de este proyecto en el mercado laboral.

El Manual Técnico se encuentra disponible en su versión en español y en inglés en este mismo documento.

### 1.2. OBJETIVOS DEL PROYECTO

**Objetivo General:** Aplicar y demostrar las técnicas de proyecciones, transformaciones, modelado geométrico y jerárquico, carga de modelos, etc., adquiridos durante todo el curso.

#### Objetivos específicos:

- Recrear en 3D en el lenguaje OpenGL la casa de Betty de la novela colombiana *Yo soy Betty, la fea* (Gaitán, F. 1999), la cual contempla la fachada de la casa con vista a la calle y la banqueta que circunscribe ésta. Así como incluir la estructura interna de la casa y los cuartos existentes.

- Recrear cinco objetos contextualizados en dos cuartos de la casa (el comedor y la recámara de Betty), y ambientarlos conforme a las Imágenes de Referencias encontradas en este Manual.
- Implementar cuatro animaciones con lógica y contexto dentro del ambiente virtual.
- Presentar un análisis de costos del proyecto.

## 1.3. PLANEACIÓN Y DESARROLLO

### 1.3.1. METODOLOGÍA DE SOFTWARE APLICADA

El desarrollo de este proyecto se llevó a cabo mediante la aplicación de un Modelo de Cascada Adaptado (Figura 1.3.1.1).

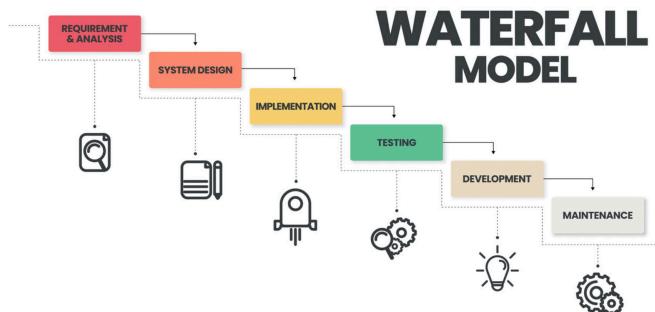


Figura 1.3.1.1. Metodología Waterfall.

Imagen obtenida de: <https://coworkingfy.com/wp-content/uploads/2023/03/metodologia-waterfall-5.jpg>

Esta metodología fue seleccionada debido al desarrollo secuencial del proyecto, donde el diseño y modelado de la geometría fue indispensable y no negociable antes de implementar las funcionalidades interactivas, como las animaciones, iluminación y manejo de cámara.

El modelo se implementó en las siguientes fases secuenciales:

#### 1. Análisis de requisitos y conceptualización:

- Selección y obtención de la fachada y el espacio interior/exterior de referencia.

- Definición de los cinco objetos a modelar en cada cuarto.
- Definición de las cuatro animaciones requeridas y sus contextos.

**2. Diseño de la arquitectura y modelado geométrico:**

- Modelado de la geometría de la casa (fachada y espacio), así como incluir puertas, ventanas, cuartos y escaleras.
- Modelado detallado de los cinco objetos de cada cuarto definidos en la fase anterior.
- Integración de los modelos con el código base de OpenGL.

**3. Implementación de funcionalidades y texturizado:**

- Búsqueda y aplicación de texturizado y materiales a los cinco objetos y posteriormente a la fachada.
- Codificación de las transformaciones para las cuatro animaciones.
- Ajuste de la ambientación mediante el uso correcto de la iluminación.
- Desarrollo del manejo de cámara.

**4. Análisis de costos del proyecto:**

- Recopilación de recursos, personal y tiempo invertido para el desarrollo del proyecto.
- Comparación contra cotizaciones promedio del mercado laboral.

**5. Pruebas y documentación:**

- Pruebas de funcionalidad para asegurar que todas las animaciones, interacciones de la cámara y el ejecutable funcionaran.
- Revisión del cumplimiento de los límites de peso de los objetos (menos de 100 MB).
- Limpieza de geometría y economización de tamaño de texturas y modelos.
- Redacción del Manual Técnico y el Manual de Usuario en español e inglés.

### 1.3.2. DIAGRAMA DE GANTT

Fecha de asignación: 06 de octubre de 2025.

Fecha de entrega: 25 de noviembre de 2025.

En la Figura 1.3.2.1 se muestra un diagrama de Gantt que muestra el avance del desarrollo del segundo proyecto final a lo largo de aproximadamente 7 semanas, desde la fecha de asignación hasta la fecha de entrega.

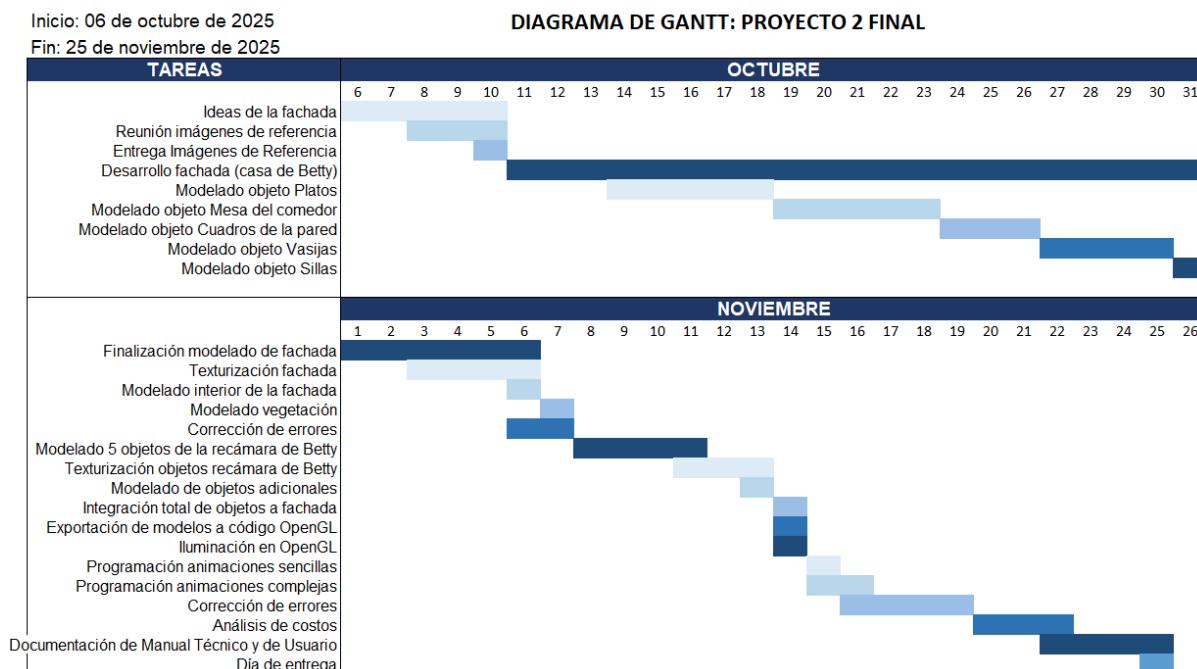


Figura 1.3.2.1. Diagrama de Gantt.

#### Octubre: Modelado y Estructura

El enfoque principal de octubre es la fase de modelado geométrico. Las tareas se centran en la creación de la estructura base y los objetos:

- Se inicia el Modelado de los 5 Objetos requeridos para cada escena.
- Se realiza la Carga e Integración de Modelos con el código base del curso.
- Paralelamente, se va desarrollando la fachada de la casa.

#### Noviembre: Cierre, Pulido y Documentación

Noviembre es el mes de la Implementación de Funcionalidades, transformando los modelos estáticos en un ambiente interactivo, y finalmente de cierre y documentación, asegurando la funcionalidad completa y la entrega formal.

- Se finaliza la codificación de las 4 animaciones.
- Se implementa el sistema de manejo de cámara.
- Se realizan pruebas funcionales y control de calidad.
- Se realiza el análisis de costos.
- Se lleva a cabo la redacción del Manual Técnico y Manual de Usuario.
- Finalmente, se genera el ejecutable y se sube el proyecto a GitHub, cumpliendo con la fecha de entrega final el 25 de noviembre.

### 1.3.3. ALCANCE DEL PROYECTO

El proyecto establece como su alcance primordial la recreación tridimensional (3D) de una fachada y dos espacios interior/exterior específicos, utilizando un código base en común y la librería OpenGL. El proyecto debe aplicar y demostrar los conocimientos adquiridos a lo largo del semestre, los cuales incluye:

- Proyecciones, puertos de vista y transformaciones geométricas.
- Modelado geométrico.
- Modelado jerárquico.
- Carga de modelos y cámara sintética.
- Texturizado.
- Iluminación.
- Animación.

### 1.3.4. LIMITANTES

Dentro de las limitantes en el desarrollo del proyecto se encuentran las siguientes:

- a) Temática Prohibida: Se exceptúa cualquier espacio de ciudad universitaria, Rick and Morty, los Simpsons, la casa de Kamehouse de Dragon Ball, la casa de las chicas superpoderosas, la cabaña de Gravity Falls, la casa de Garfield y la casa del Coraje el perro cobarde.

- b) Peso de Archivos: Todo objeto recreado o descargado tiene que pesar menos de 100 MB.
- c) Objetos Repetidos: Los objetos recreados repetidamente contarán como un solo objeto para la evaluación de los 5 elementos en cada cuarto.
- d) Animaciones Repetidas: Animaciones repetidas (como puertas y ventanas que hagan lo mismo) cuentan como una sola animación.

## 1.4. DESARROLLO TÉCNICO

### 1.4.1. OBJETIVOS Y ALCANCES TÉCNICOS

- Recrear con precisión un mínimo de cinco objetos distintivos de la imagen de referencia en los dos cuartos, con geometría y texturizado correcto para alcanzar un alto nivel de realismo del espacio virtual.
- Codificar y ejecutar un total de cuatro animaciones funcionales y contextuales, las cuales serán sencillas y complejas (no lineales).
- Uso correcto de la iluminación para lograr una ambientación adecuada y realista dentro del espacio virtual.
- Implementar un sistema funcional y documentado de manejo de cámara que permita la interacción del usuario con el ambiente recreado.
- Asegurar que todo objeto modelado o descargado pese menos de 100 MB.

### 1.4.2. MODELADO

El modelado de la totalidad de la escena, incluyendo la fachada, los dos espacios de interior/exterior y los cinco (5) objetos principales requeridos para cada cuarto, se realizó utilizando el software Autodesk Maya 2023. Maya fue seleccionado como la herramienta principal para la creación de la geometría debido a sus capacidades técnicas, esenciales para un proyecto basado en OpenGL.

La elección de Maya 2023 sobre otras alternativas se debe a su precisión geométrica y su control sobre la topología de la malla, de gran apoyo para crear modelos con un bajo número de polígonos para economizar recursos computacionales.

También por la configuración y poder de mapas UV que posee, bastante completa para modificar correctamente las texturas sobre los objetos modelados y conseguir mayor realismo y economización de espacio, según sea el caso.

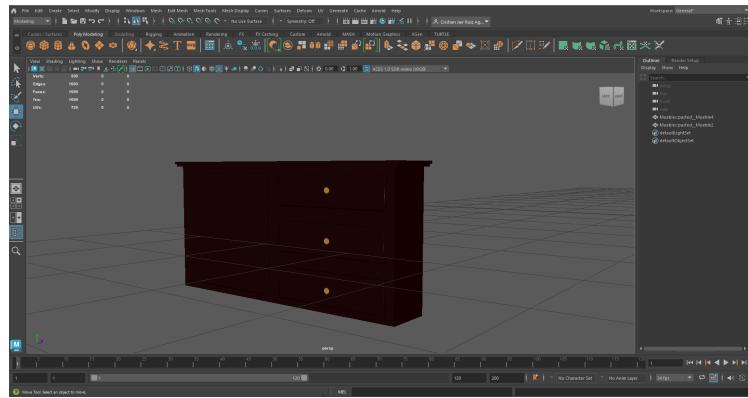


Figura 1.4.2.1. Interfaz de Maya 2023 modelando uno de los 5 objetos del comedor.

Los objetos modelados a partir de los polígonos que brinda Maya fueron:

- Fachada (casa de Bety): jardín, hojas de los árboles, banqueta, rejas, planta baja y primer piso.
- Puertas.
- Ventanas.
- Alfombra.

Los objetos pertenecientes al comedor son:

- Cuadros con fotos.
- Sillas.
- Mantel de mesa.
- Vasijas.
- Platos.

Los objetos pertenecientes a la recámara de Betty son:

- Cama.
- Burós a los costados de la cama.
- Escritorio.
- Computadora de escritorio.
- Armario.

Los objetos extraídos de Internet<sup>1</sup> y fueron adaptados al proyecto fueron:

- Tronco de los árboles.
- Vasijas.
- Mariposa.
- Sillas.
- Monitor.
- Teclado.
- Almohadas.

#### 1.4.3. TEXTURIZACIÓN Y MAPEADO UV

El realismo del ambiente virtual se basa en la aplicación de texturas sobre la geometría modelada. Este proceso es configurado y mejorado por el Mapeado UV, una técnica que permite proyectar imágenes bidimensionales sobre superficies tridimensionales.

##### 1. Texturización

La texturización es la fase donde se asignan mapas de imagen (texturas) a los modelos 3D para simular las propiedades físicas y visuales de los materiales (color, rugosidad, especularidad, etc.). Este proceso se realizó en varias ocasiones con apoyo del software Maya para asignar a cada material del proyecto su respectiva textura, como se muestra en la Figura 1.4.2.2.

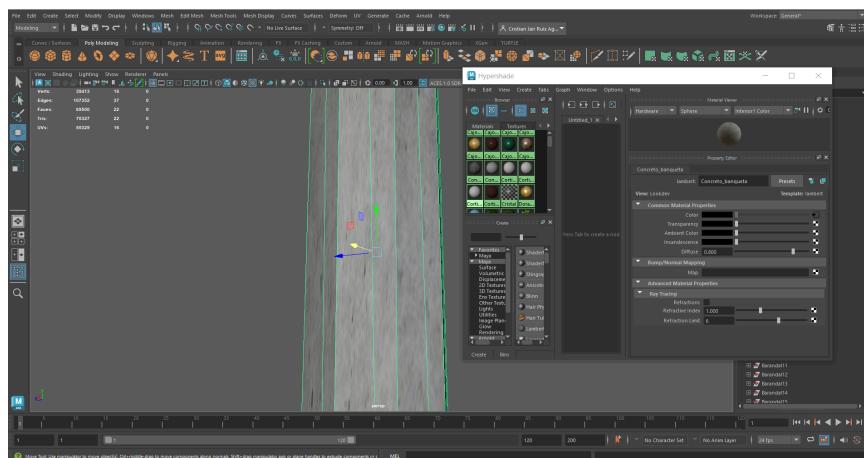


Figura 1.4.2.2. Asignación de material en software Maya.

<sup>1</sup> Los créditos de los objetos descargados de Internet se encuentran en las Referencias de este documento.

Cabe mencionar que para la implementación de las texturas se tuvo la precaución de que todas las imágenes utilizadas, independientemente de su tamaño absoluto, tuvieran dimensiones que fueran una potencia de dos. Esto significa que tanto el ancho (W) como la altura (H) de las imágenes deben ser valores como 2, 4, 8, 16, 32, ..., 512, 1024, 2048, etc. (Tabla 1.4.2.2).

Textura	Tamaño en pixeles
 Pared de la sala de la casa	256 x 256
 Tronco	128 x 128
 Madera color chocolate	8 x 8

Tabla 1.4.2.2. Ejemplo de algunas texturas y sus tamaños.

## 2. Mapeado UV

El Mapeado UV es la herramienta para trabajar la interfaz técnica entre la textura 2D y el modelo 3D. Consiste en la creación de un sistema de coordenadas (U, V) que mapea cada punto de la superficie de la malla 3D (X, Y, Z) a un punto específico en el plano 2D de la textura.

El software Maya posee diversas herramientas para trabajar con las coordenadas de texturas de los objetos modelados, sin embargo, la técnica más empleada fue transformar las superficies texturizadas a planares, después orientadas al eje al que estuvieran mirando. De esa forma se logró que las texturas no dieran la apariencia

de estar estiradas y verse con más realismo. En la Figura 1.4.2.3 se observa la edición de un muro de textura de ladrillos, al cual se le requirió aplicar una superficie tipo planar hacia el eje x para dar mayor realismo.

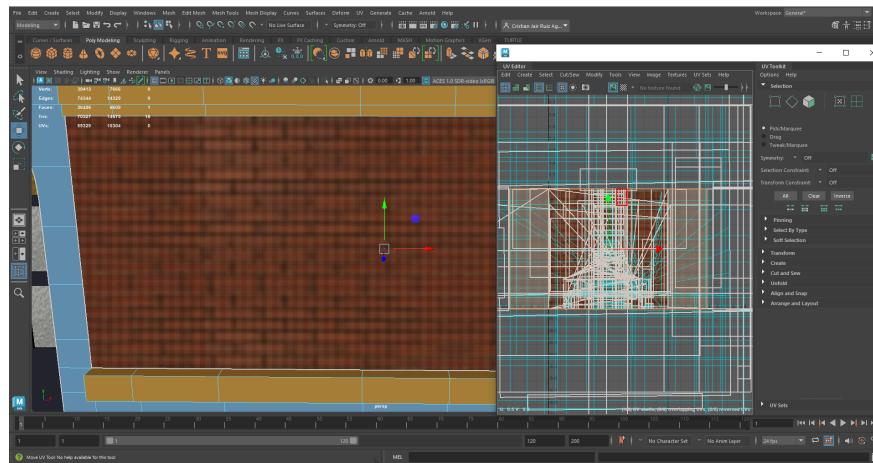


Figura 1.4.2.3. Interfaz editor UV de Maya.

#### 1.4.4. ESTRUCTURA DEL CÓDIGO

La estructura del proyecto sigue un diseño monolítico que se basa principalmente en un único archivo fuente C++ que integra tanto la lógica de inicialización y control como las funciones de renderizado. Se recurrió al uso del software Microsoft Visual Studio para realizar todas las funciones de programación del proyecto.

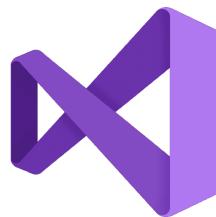


Figura 1.4.4.1. Logotipo de Microsoft Visual Studio.

El código se organiza en las siguientes secciones lógicas y funcionales:

**1. Librerías y Dependencias:** Se incluyen las librerías estándar de C++ y la librería gráfica principal, junto con las dependencias externas requeridas para la funcionalidad avanzada, como se muestra en la Figura 1.4.4.2:

```
#include <iostream>
#include <cmath>

// GLEW
#include <GL/glew.h>

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

// Other Libs
#include "stb_image.h"

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

// Load Models
#include "SOIL2/SOIL2.h"

// Other includes
#include "Shader.h"
#include "Camera.h"
#include "Model.h"
#include "Texture.h"
```

Figura 1.4.4.2. Librerías estándar y gráficas del código cpp.

**#include <windows.h>**: Para funciones específicas del sistema operativo Windows.

**#include <GL/glew.h>**: Utilizada para gestionar las extensiones de OpenGL para la compatibilidad con las características más modernas.

**#include <GL/glut.h>**: La librería base para la gestión de ventanas y eventos de input.

**#include <iostream>, <vector>, <stdlib.h>, <math.h>**: Librerías estándar para operaciones matemáticas, gestión de memoria y entrada/salida.

**#include "Camera.h", #include "Model.h"**: Cabeceras propias que contienen la definición de las clases para la gestión de la cámara y la carga de modelos 3D externos.

**Variables Globales y de Estado**: Se declaran variables globales para almacenar el estado de la escena, como la posición, rotación y escala de los objetos.

```

// Window dimensions
const GLuint WIDTH = 800, HEIGHT = 600;
int SCREEN_WIDTH, SCREEN_HEIGHT;

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

// ===== SIMPLE ANIMATIONS =====

// CORTINA1 ANIMATION
bool Cortina_abierta1 = true; // Indica si la cortina esta abierta
bool anima_cortina1 = false; // Indica si la animacion esta en curso
float cortinaAbrel = 1.0f; // cantidad de escalamiento (apertura) de la cortina

```

Figura 1.4.4.3. Declaración de variables de ventana, cámara, de luz y de una animación simple.

**Funciones de Inicialización init():** Esta función se ejecuta una única vez al inicio del programa y se encarga de:

- Configurar el contexto de OpenGL y los modos de renderizado (ej. Depth Test).
- Inicializar la librería GLEW.
- Cargar las texturas y los modelos 3D (Model.h) desde los archivos externos.
- Empezar la configuración inicial de la Ambientación (iluminación y materiales).

#### Bucle Principal de Renderizado (Game Loop):

- La lógica de dibujo se encuentra dentro del bucle:

```

// Game loop
while (!glfwWindowShouldClose(window))
{

```

Este bucle es responsable de limpiar los buffers, aplicar la matriz de vista (cámara) y la matriz de proyección, y realizar las llamadas a las funciones que contienen el código de dibujado de la fachada, los dos espacios, los 5 objetos y la ambientación general. Esto se observa en la Figura 1.4.4.4.

```

// Game loop
while (!glfwWindowShouldClose(window))
{
    // Calculate deltatime of current frame
    GLfloat currentTime = glfwGetTime();
    deltaTime = currentTime - lastFrame;
    lastFrame = currentTime;

    // Check if any events have been activated (key pressed, mouse moved etc.) and call corresponding response functions
    glfwPollEvents();
    DoMovement();

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

    // OpenGL options
    glEnable(GL_DEPTH_TEST);

    //Load Model

    // Use cooresponding shader when setting uniforms/drawing objects
    lightingShader.Use();

    glUniform1i(glGetUniformLocation(lightingShader.Program, "material.diffuse"), 0);
    glUniform1i(glGetUniformLocation(lightingShader.Program, "specular"), 1);

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

```

Figura 1.4.4.4. Código inicial del “Game Loop”.

**Funciones de Interacción y Eventos keyboard(), mouse():** Contienen la lógica para la gestión de las entradas del usuario (teclado y ratón), permitiendo la:

- Activación y control de las 4 animaciones.
- Control del manejo de cámara para la navegación del usuario en el ambiente virtual.

```

// Is called whenever a key is pressed/released via GLFW
void KeyCallback(GLFWwindow* window, int key, int scanCode, int action, int mode)
{
    if (GLFW_KEY_ESCAPE == key && GLFW_PRESS == action)
    {
        glfwSetWindowShouldClose(window, GL_TRUE);
    }

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

    if (keys[GLFW_KEY_1] && !anima_cortina1) { // ANIMATION CORTINA1
        anima_cortina1 = true;
    }

    if (keys[GLFW_KEY_2] && !anima_cortina2) { // ANIMATION CORTINA2
        anima_cortina2 = true;
    }

    if (keys[GLFW_KEY_3] && !anima_reja) { // ANIMATION REJA
        anima_reja = true;
    }

    if (keys[GLFW_KEY_4] && !anima_puertaEntrada) { // ANIMATION PUERTA ENTRADA
        anima_puertaEntrada = true;
    }
}

```

Figura 1.4.4.5. Bloque de código para gestión de entradas del teclado.

```

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

    GLfloat xOffset = xPos - lastX;
    GLfloat yOffset = lastY - yPos; // Reversed since y-coordinates go from bottom to left

    lastX = xPos;
    lastY = yPos;

    camera.ProcessMouseMovement(xOffset, yOffset);
}

```

Figura 1.4.4.6. Bloque de código para gestión de entradas del mouse.

**Shaders:** se ejecutan directamente en la GPU para el renderizado moderno de OpenGL.

- Vertex Shader (.vs): Se ejecuta por cada vértice de la geometría. Su función principal es aplicar las transformaciones geométricas (Modelo, Vista, Proyección) para posicionar los modelos en la pantalla.
- Fragment Shader (.frag): Se ejecuta por cada píxel potencial y determina su color final.

Se pueden tener varios Shaders porque el pipeline de renderizado es un proceso de múltiples etapas, y cada Shader está especializado para ejecutar tareas específicas en diferentes tipos de datos, como dibujo o animaciones complejas. Los shaders incluídos en el proyecto (Figura 1.4.4.7) abarcan tareas como manejar colores, iluminación, animaciones complejas, manejo de SkyBox y carga de modelos y sus texturas.

□	anim.frag
□	anim.vs
□	Anim2.frag
□	Anim2.vs
□	lamp.frag
□	lamp.vs
□	lighting.frag
□	lighting.vs
□	modelLoading.frag
□	modelLoading.vs
□	SkyBox.frag
□	SkyBox.vs

Figura 1.4.4.7. Shaders contenidos en el proyecto.

#### 1.4.5. CARGA DE MODELOS

Los modelos se exportan a un formato estándar de intercambio 3D, como .OBJ o .FBX. En este proyecto se usó el formato .OBJ. Estos archivos contienen los datos de la malla:

- Posición de Vértice: Coordenadas (X, Y, Z).
- Vector Normal: para el cálculo de la iluminación.
- Coordenadas de Textura: Coordenadas (U, V).

Estos modelos fueron almacenados en su respectiva carpeta dentro del directorio Models. Cada modelo con su respectivo archivo .mtl y texturas.

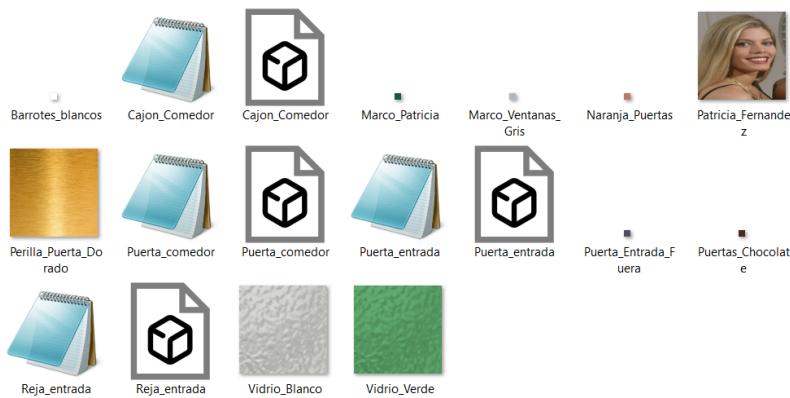


Figura 1.4.5.1. Archivos de objetos exportados desde Maya.

El código C++ utiliza la librería auxiliar (Model.h) para leer e interpretar el archivo exportado, y se especifica su ruta para cargarlos como se muestra a continuación (Figura 1.4.5.2).

```

// OBJECTS

Model CasaBety((char*)"Models/CasaBety/Casa_de_Bety.obj");
Model Platos((char*)"Models/Platos/Platos.obj");
Model BaseMesa((char*)"Models/MesaComedor/Mesa.obj");
Model Sillas((char*)"Models/Sillas/Sillas.obj");
Model Vasijas((char*)"Models/Vasijas/Vasijas.obj");
Model Cuadros((char*)"Models/Cuadros/Cuadros.obj");
Model Muebles((char*)"Models/Muebles/Muebles.obj");

// OBJECTS WITH SOME TRANSPARENCY

Model MantelMesa((char*)"Models/MesaComedor/Mantel_mesa.obj");
Model Vasos((char*)"Models/Vasos/Vasos.obj");
Model Vidrios((char*)"Models/Ventanas/Vidrios.obj");

// SIMPLE ANIMATION OBJECTS

Model Reja_entrada((char*)"Models/Animaciones_sencillas/Reja_entrada.obj");
Model Cortina1((char*)"Models/Cortinas/Cortina1.obj");
Model Cortina2((char*)"Models/Cortinas/Cortina2.obj");
Model PuertaEntrada((char*)"Models/Animaciones_sencillas/Puerta_entrada.obj");
Model PuertaComedor((char*)"Models/Animaciones_sencillas/Puerta_comedor.obj");
Model Cajon((char*)"Models/Animaciones_sencillas/Cajon_Comedor.obj");

```

Figura 1.4.5.2. Carga de modelos a OpenGL.

#### 1.4.5.1. SKYBOX

La implementación del Skybox sigue el estándar de OpenGL utilizando la técnica de Cube Maps (Mapas de Cubo) para que el entorno se perciba como tridimensional y envolvente. Estas se cargan y se unen en una sola textura de tipo:

*GL\_TEXTURE\_CUBE\_MAP*

El Skybox se renderiza utilizando la función de profundidad configurada a:

*GL\_EQUAL*

Esto asegura que el cubo de fondo se dibuje en las posiciones de profundidad más lejanas para que todos los objetos de la escena se dibujen sobre él.

```
//SkyBox
GLuint skyboxVBO, skyboxVAO;
glGenVertexArrays(1, &skyboxVAO);
glGenBuffers(1, &skyboxVBO);
 glBindVertexArray(skyboxVAO);
 glBindBuffer(GL_ARRAY_BUFFER, skyboxVBO);
 glBufferData(GL_ARRAY_BUFFER, sizeof(skyboxVertices), &skyboxVertices, GL_STATIC_DRAW);
 glEnableVertexAttribArray(0);
 glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);

// Load textures
vector<const GLchar*> faces;
faces.push_back("SkyBox/right.tga");
faces.push_back("SkyBox/left.tga");
faces.push_back("SkyBox/top.tga");
faces.push_back("SkyBox/bottom.tga");
faces.push_back("SkyBox/back.tga");
faces.push_back("SkyBox/front.tga");

GLuint cubemapTexture = TextureLoading::LoadCubemap(faces);
```

Figura 1.4.5.1.1. Manejo de buffers y carga de modelos del Skybox.

#### 1.4.6. CÁMARA SINTÉTICA

El proyecto utiliza un modelo de cámara tipo Free Look implementado en la clase auxiliar *Camera.h*. Esto permite al usuario moverse libremente y navegar alrededor de la fachada y a través del interior/exterior. La cámara se define por tres vectores clave:

- Posición: La ubicación actual de la cámara en el espacio (X, Y, Z).
- Dirección Frontal (Front): El vector que apunta hacia donde mira la cámara.
- Dirección Superior (Up): El vector que define la orientación vertical de la cámara.

```
// Camera
Camera camera(glm::vec3(0.0f, 0.0f, 3.0f));
```

Figura 1.4.6.1. Posición inicial de la cámara.

#### Manejo de Cámara (Interacción):

- El movimiento frontal, lateral y vertical se controla mediante callbacks del teclado (KeyCallback), modificando el vector Posición de la cámara. La velocidad del movimiento es constante.
- La rotación de la cámara se gestiona mediante callbacks del mouse (MouseCallback). Los movimientos del mouse modifican el vector Dirección Frontal para dar el efecto de mirar alrededor y brindar una mejor experiencia de usuario.

#### 1.4.7. ILUMINACIÓN

La configuración de la iluminación se basa en la combinación de tres componentes principales:

- **Luz Ambiental:** Representa la luz indirecta o dispersa en la escena para que ninguna parte del modelo quede completamente oscura, incluso si no está directamente iluminada por una fuente de luz.
- **Luz Difusa:** Es el componente más significativo para el color y el brillo del objeto. Refleja la luz que se dispersa uniformemente en todas las direcciones después de golpear la superficie. Depende directamente de la dirección de la luz y la normal del vértice.
- **Luz Especular:** Simula los "puntos brillantes" o reflejos que aparecen en las superficies pulidas o brillosas. Depende de la posición de la cámara (la dirección de la vista) y la dirección de la luz. Sirve para dar una sensación de materialidad.

```
// Directional light
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.668f, 0.649f, 0.582f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.668f, 0.649f, 0.582f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.5f, 0.5f, 0.5f);
```

Figura 1.4.7.1. Bloque de código para configurar la luz direccional.

#### 1.4.8. ANIMACIONES

Las animaciones se manejan tanto en la función *KeyCallBack* como *DoMovement* para que se activen cuando el usuario presione una tecla, y después se realice la animación, y no se detenga hasta que el movimiento programado concluya.

Desde el software Maya se configuró a los objetos animables para ser independientes de la fachada y colocar su pivote en el centro del mundo, ya que OpenGL toma como referencia ese centro del mundo. Posteriormente, con transformaciones de OpenGL, se colocaron en sus respectivos lugares.

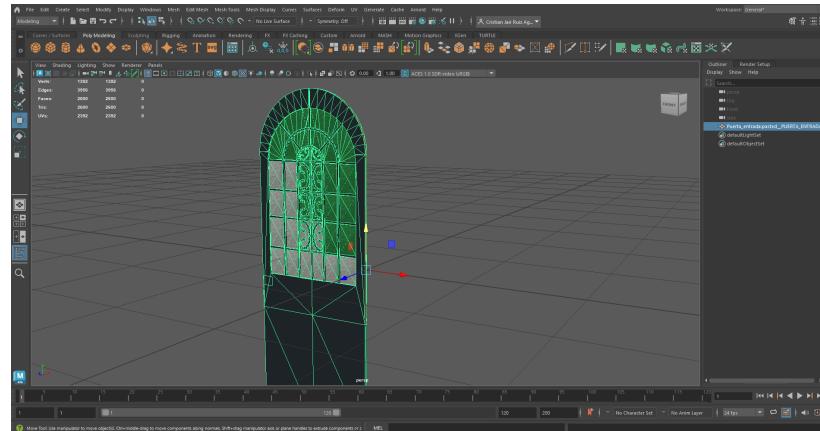


Figura 1.4.8.1. Configuración de objetos animables en el centro del mundo.

```
// PUERTA ENTRADA
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-11.972f, 1.027f, -6.617f));
model = glm::rotate(model, glm::radians(puertaEntradaAbre), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
PuertaEntrada.Draw(lightingShader);
```

Figura 1.4.8.2. Objeto animable con transformaciones para posicionarlo conforme a la fachada.

### 1.4.8.1. SIMPLES

**1. Cortinas del comedor:** Las cortinas poseen una variable que reduce su escala en un eje que les permita dar la impresión de que son abiertas.

```
//ANIMATION CORTINA1
if (anima_cortina1) {

    if (!Cortina_abierta1) { // Si esta cerrada

        if (cortinaAbre1 < 1.0f) { // Abre
            cortinaAbre1 += 0.05f;
        }
        else {
            Cortina_abierta1 = true; // ya esta abierta
            anima_cortina1 = false;
        }
    }
    else { // Si esta abierta

        if (cortinaAbre1 > 0.225f) {
            cortinaAbre1 -= 0.05f;
        }
        else {
            Cortina_abierta1 = false; // ya esta cerrada
            anima_cortina1 = false;
        }
    }
}
```

Figura 1.4.8.1.1. Bloque de código con la lógica para manejar la animación de las cortinas.

**2. Reja de entrada, puerta de entrada, puerta de comedor, cajón, puerta de la recámara de Betty:** Se le asignó una variable de rotación para simular apertura limitada a los tres objetos con la misma lógica mostrada en la siguiente imagen.

```
//ANIMATION REJA
if (anima_reja) {

    if (!rejaAbierta) { // Si esta cerrada

        if (rejaAbre > -29.236f) { // Abre
            rejaAbre -= 2.5f;
        }
        else {
            rejaAbierta = true; // ya esta abierta
            anima_reja = false;
        }
    }
    else { // Si esta abierta

        if (rejaAbre < 46.297f) {
            rejaAbre += 2.5f;
        }
        else {
            rejaAbierta = false; // ya esta cerrada
            anima_reja = false;
        }
    }
}
```

Figura 1.4.8.1.2. Bloque de código con la lógica para manejar la animación de las puertas.

### 1.4.8.2. COMPLEJAS

Las animaciones complejas se diferencian principalmente por el hecho de que implementan ecuaciones y funciones matemáticas para ser independientes de la interacción del usuario y poseer autonomía de movimiento.

#### 1. Hojas de árbol:

Se utiliza una variable de tiempo que se actualiza continuamente en el Bucle Principal de Renderizado y se pasa al Vertex Shader como un uniform. El movimiento sinusoidal se calcula en función de este tiempo para una oscilación continua.

El grado de desplazamiento de cada vértice se modula utilizando su coordenada 'Y' original (altura).

La razón por la que las hojas se mueven mucho más arriba que de abajo es puramente algorítmica y se implementa con la línea

```
windFactor *= position.y;
```

o su equivalente.

Cuando la posición 'Y' es cercana a cero (base del árbol), el windFactor resultante es muy pequeño, resultando en un movimiento casi nulo, simulando la rigidez del tronco.

Conforme la posición 'Y' aumenta (subiendo a las ramas y hojas), el windFactor se amplifica, generando un desplazamiento lateral notable.

```
uniform mat4 view;
uniform mat4 projection;
uniform float time;

// Función hash para obtener un valor pseudoaleatorio
float hash(vec2 p) {
    return fract(sin(dot(p, vec2(12.9898, 78.233))) * 43758.5453);
}

void main()
{
    float normalizedHeight = aPos.y * 0.3;
    float windWave = sin(aPos.x * frequency + aPos.z * 1.5 + time * 10.0);

    float displacement = amplitude * windWave * normalizedHeight; //desplazamiento.

    vec3 newPos = aPos;
    newPos.x += displacement;
    newPos.z += displacement * 0.5; // Un desplazamiento un poco menor en Z

    gl_Position = projection * view * model * vec4(newPos, 1.0);
    TexCoords = aTexCoords;
}
```

Figura 1.4.8.2.1. Bloque de código del Shader que ejecuta la animación de las hojas de los árboles (anim.vs).

## 2. Mariposa

Se utiliza una variable de tiempo que se actualiza continuamente y se pasa al Vertex Shader como un uniform.

Se calcula un factor de desplazamiento sinusoidal (waveFactor) que determina cuánto se moverá el vértice en el frame actual. Para que el ondeado sea más intenso en ciertas partes del ala (como en el borde exterior) y menos intenso en la base, el waveFactor se modula o se atenúa basándose en una de las coordenadas del vértice.

```

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

out vec2 TexCoords;

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

void main()
{
    float horizontalOffset = sin(time * 0.05) * 4.0; // Aleteo rapido
    float verticalOffset = -cos(time * 6.0) * 0.04; // Asciende y descende rapido
    /
    float direccion = horizontalOffset >= 0.0 ? 1.0 : -1.0;

    vec4 position = vec4(aPos.x + horizontalOffset, aPos.y +verticalOffset, aPos.z, 1.0);

    gl_Position = projection * view * model * position;
    TexCoords = aTexCoords;
}

```

Figura 1.4.8.2.2. Bloque de código del Shader que ejecuta la animación de la elevación y descenso de la mariposa (Anim2.vs).

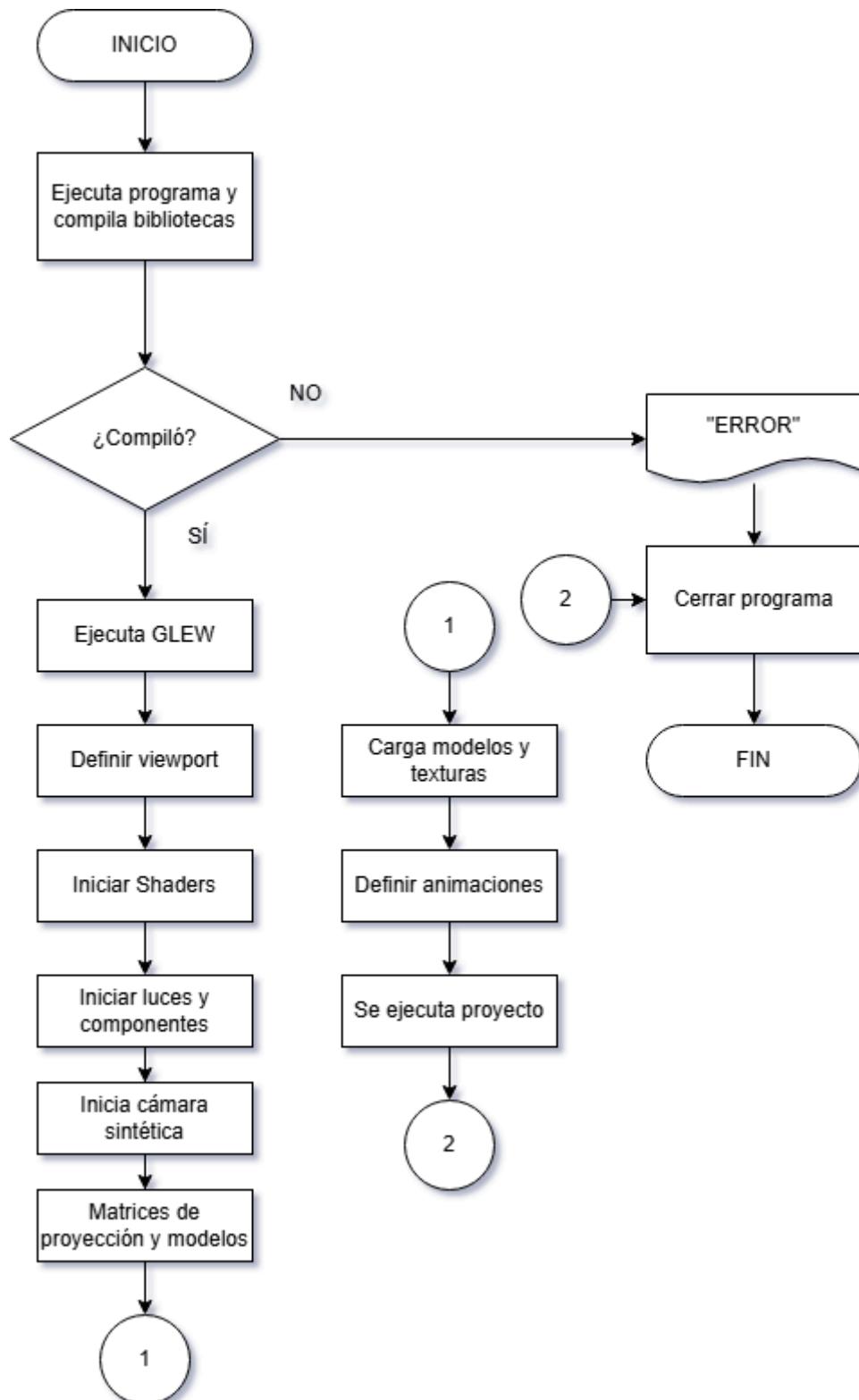
Se hace uso del bloque DoMovement para simular el aleteo de las alas:

```

//ANIMATION MARIPOSA - ALETEO CONSTANTE
if (alaSubiendo) {
    aleteoAbre += 50.0f;
    if (aleteoAbre >= limiteRotacion)
        alaSubiendo = false;
}
else {
    aleteoAbre -= 50.0f;
    if (aleteoAbre <= -limiteRotacion)
        alaSubiendo = true;
}

```

Figura 1.4.8.2.3. Bloque de código con la lógica del aleteo de las alas de la mariposa.

**1.4.9. DIAGRAMA DE FLUJO DEL SISTEMA**

## 2. REFERENCIAS VISUALES

A continuación se muestran varias capturas y fotografías de la fachada (la casa de Betty) y los dos espacios (comedor y recámara de Betty) seleccionadas para tomar como base y trabajar a partir de éstas.

### 2.1. COMPARATIVA IMÁGENES DE REFERENCIA

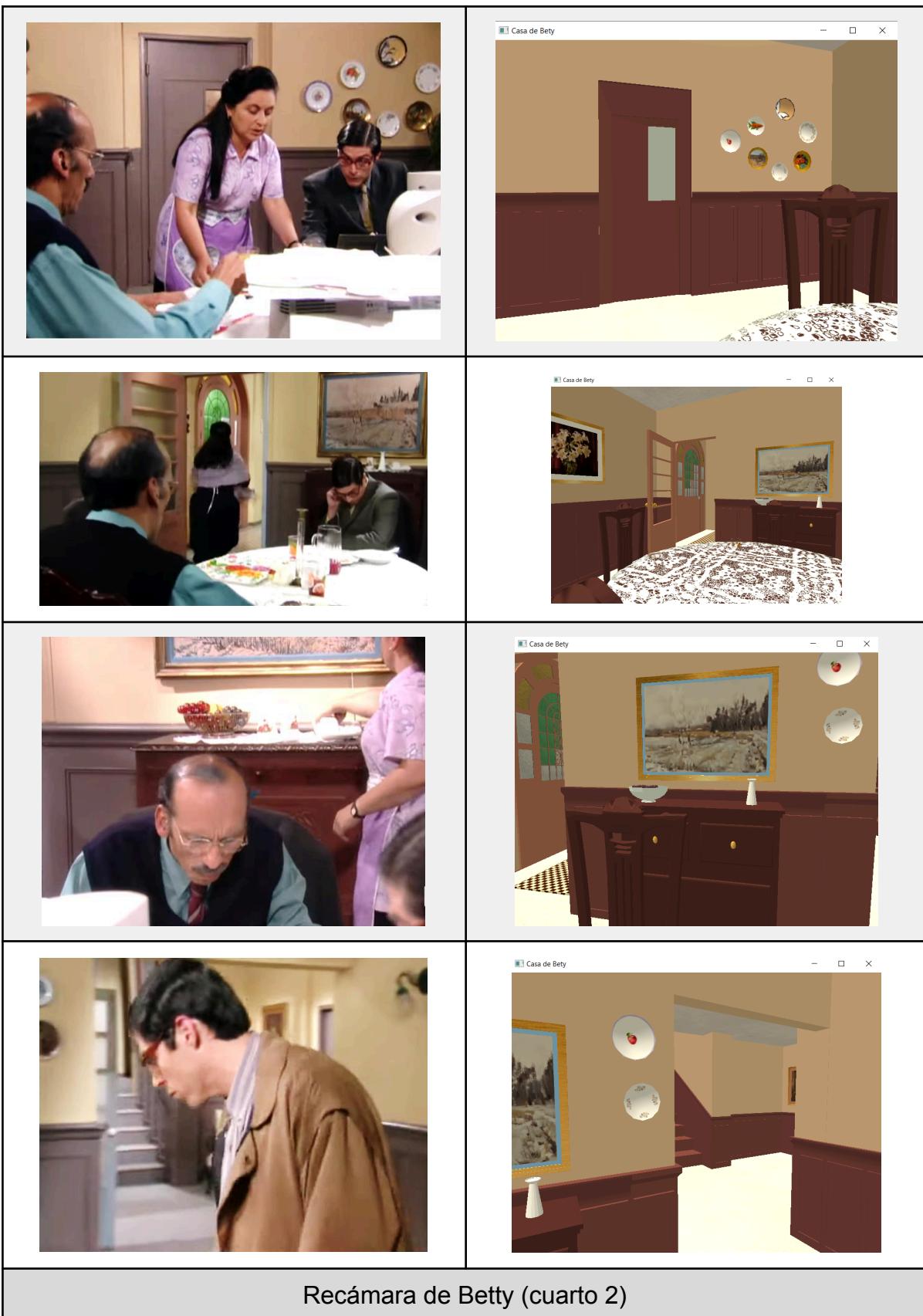
En la Tabla 2.1.1 se muestra una comparación entre las imágenes de referencia propuestas para desarrollar este proyecto y las que resultaron después de desarrollar la totalidad del ambiente virtual.

Para las imágenes de referencia del comedor de la casa de Betty se descartaron a los personajes que aparecen en las escenas.

Las imágenes mostradas en el ambiente virtual de la recámara de Betty tienen iluminación de día, como se señaló en las imágenes de referencias cuando se propusieron.

Imagen de referencia	Ambiente virtual
Casa de Betty (fachada)	
	
	





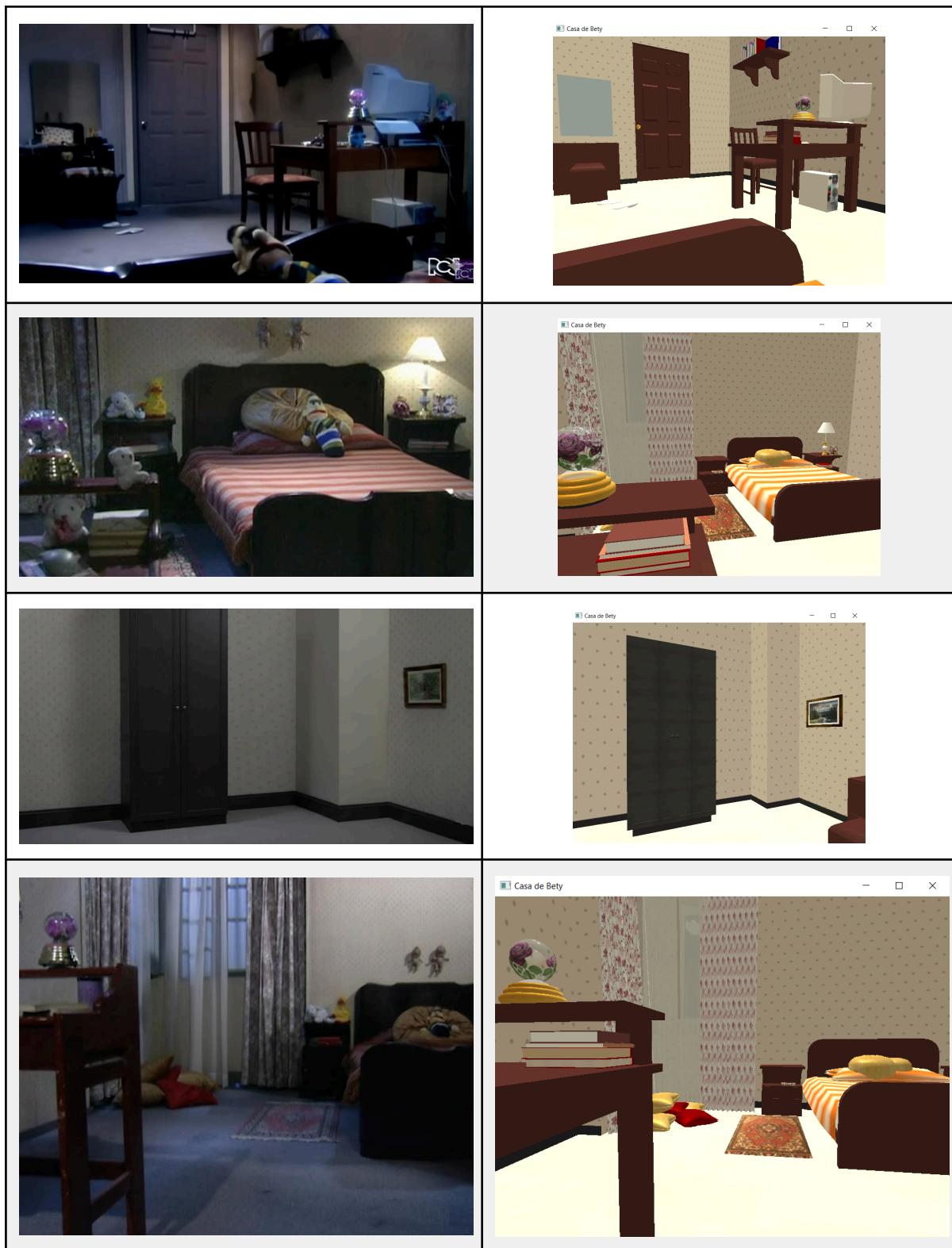
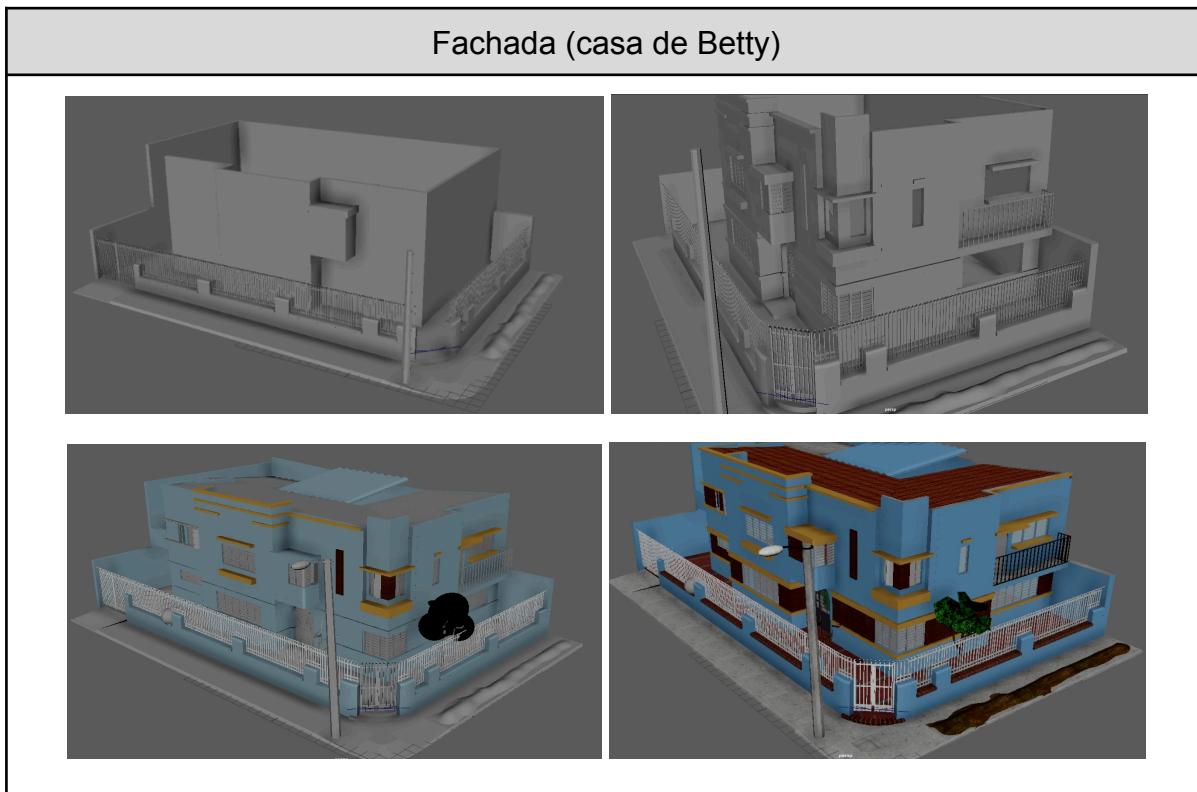


Tabla 2.1.1. Comparación imágenes de referencia contra capturas del proyecto.

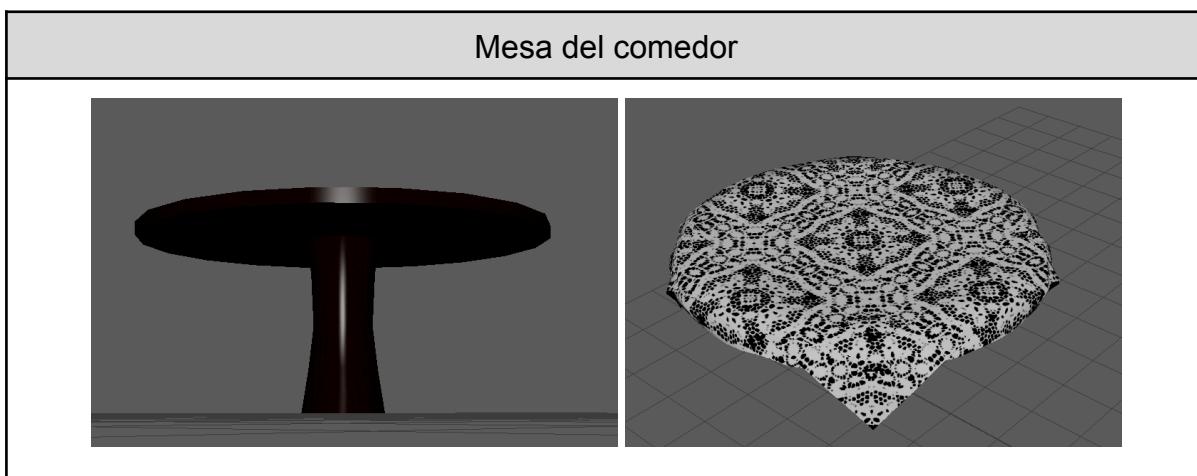
## 2.2. OBJETOS RECREADOS

Cabe mencionar que en esta sección solo se incluirán los objetos enlistados en el documento de las Imágenes de Referencia. No se incluirán objetos extra, como muebles, jardines, el poste de la calle, balcones, techo, libros, etc.

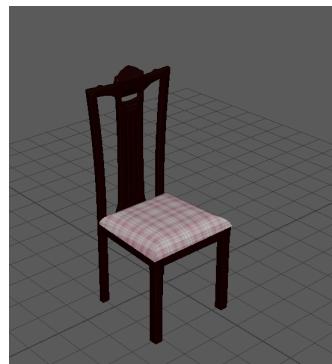
### 2.2.1. FACHADA (CASA DE BETTY)



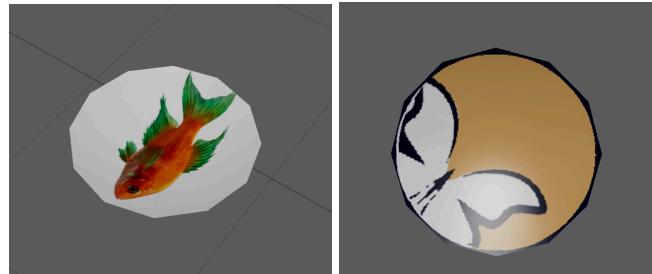
### 2.2.2. COMEDOR DE LA CASA DE BETTY



Sillas de la mesa del comedor



Platos de la pared trasera



Vasijas por la pared lateral

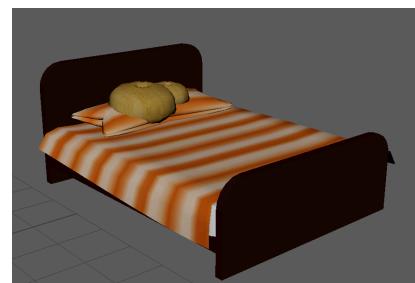


Cuadros de las tres paredes

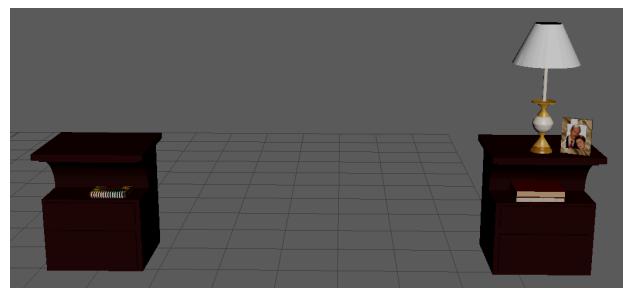


### 2.2.3. RECÁMARA DE BETTY

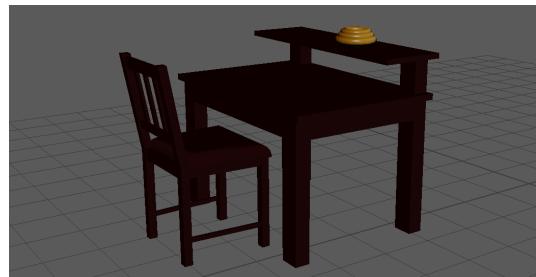
Cama



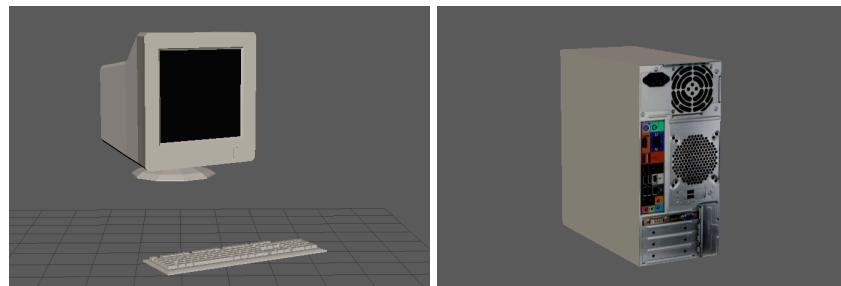
Burós a los costados de la cama



**Escritorio**



**Computadora de escritorio**



**Armario**



### 3. ANÁLISIS DE COSTOS

Este análisis estima el costo real de producir el recorrido virtual en OpenGL 3, considerando materiales digitales, software, hardware, mano de obra, infraestructura y un margen comercial para determinar el precio de venta profesional del proyecto.

#### 3.1. MATERIALES

Tipo de material	Descripción	Costo estimado (MXN)
Modelado propio	Aproximadamente 90% de los modelos están hechos a mano.	0
Modelos externos	6 modelos gratuitos de TurboSquid.	600

#### 3.2. SOFTWARE

Software	Licencia	Costo estimado (MXN)
Maya 2023	Educativa (gratuita)	0
Paint Tool Sai	Pago previo	1000
SkyBox y otros	Gratuitos	0

Se aplica un costo de desgaste proporcional del equipo usado: 150 MXN.

#### 3.3. MANO DE OBRA

Horas trabajadas en total	Tarifa por hora	Total
150 (3hrs al día)	300 MXN	45,000 MXN

#### 3.4. INFRAESTRUCTURA

Concepto	Costo estimado (MXN)
Electricidad	25
Internet	80
Total:	105

### 3.5. COSTO TOTAL

Rubro	Costo (MXN)
Materiales	600
Software	400
Hardware (desgaste)	150
Mano de obra	45,000
Infraestructura	105
Documentación	3,000
Subtotal	49,255
Contingencias (10%)	4,925
Total de producción:	54,180

Se aplica un margen comercial del 30% para trabajos de recreación 3D.

**Costo total de producción:** 54,180 MXN.

**Precio sugerido de venta:** 70,434 MXN.

### 4. REPOSITORIO EN GITHUB

Enlace a la documentación y contenido completo del proyecto en repositorio GitHub:

[https://github.com/ElanRuiz/318328278\\_PROYECTOFINAL2026-1\\_GPO05](https://github.com/ElanRuiz/318328278_PROYECTOFINAL2026-1_GPO05)

## 5. CONCLUSIONES

Este proyecto final de Computación Gráfica ha demostrado la aplicación integral y práctica de los conocimientos adquiridos a lo largo del curso, los cuales fueron bastante arduos de aplicar. El objetivo de recrear un ambiente virtual 3D se logró mediante la combinación de modelado geométrico y técnicas avanzadas de programación en OpenGL, validando la metodología de Cascada.

El aprendizaje más significativo que me llevo es la consolidación del pipeline completo de gráficos 3D. Pude realizar el modelado en Maya con la precisión necesaria para recrear la fachada y los espacios, asegurándome de que los cinco objetos distintivos y los elementos obligatorios como puertas y ventanas, fueran lo más parecidos posible a mis imágenes de referencia. Entendí que el realismo no depende solo de la geometría, sino de un texturizado correcto, lo cual me obligó a dominar el Mapeado UV para que mis texturas se vieran bien en OpenGL.

A nivel de código, siento que realmente superé el reto de generar las cuatro animaciones totalmente por código. Logré que todas fueran contextuales, pero el verdadero logro fue codificar las dos animaciones complejas. Me costó entender que para ser complejas, tenían que ser no lineales, lo cual me llevó a implementar efectos en el Vertex Shader, como el ondeado de las hojas y la mariposa. Esta técnica demostró ser fundamental, al igual que la implementación de los Shaders para controlar la ambientación.

El tener que generar el análisis de costos y el Manual Técnico y el Manual de Usuario en español e inglés me obligó a documentar mi trabajo de manera profesional y rigurosa, sin depender completamente de traductores. Además, el subir todo a un repositorio de GitHub funcional que demostrará mis cambios y avances durante el semestre me enseñó la importancia del control de versiones en un proyecto de software. Ver el proyecto completo —desde la conceptualización hasta el ejecutable final— me da una enorme satisfacción y me reafirma que, con esfuerzo y la aplicación de la metodología correcta, fui capaz de demostrar todos los conocimientos que adquirí en la materia.

## 6. REFERENCIAS

- Crick, L. J. (2018, junio 22). Old pot 3D model [Modelo 3D]. TurboSquid.  
<https://www.turbosquid.com/es/3d-models/old-pot-3d-model-1299027>
- vonherin. (2019, febrero 23). Low poly dead tree (simple) [Modelo 3D]. TurboSquid.  
<https://www.turbosquid.com/3d-models/tree-simple-dead-model-1380914>
- Mughal, U. (2025, junio 20). Dining table with chairs 3D [Modelo 3D]. TurboSquid.  
<https://www.turbosquid.com/es/3d-models/dining-table-with-chairs-3d-2423456>
- Lohsu (septiembre 2022). Old Wooden Chair 3D Model. [Modelo 3D]. TurboSquid.  
<https://www.turbosquid.com/3d-models/old-wooden-chair-3d-3d-model-1962521>
- anastasiaartist (abril 2024). Pillow 3D [Modelo 3D]. TurboSquid.  
<https://www.turbosquid.com/3d-models/pillow-3d-2222308>
- Berna91 (julio 2010). Free CRT Screen 3D Model [Modelo 3D]. TurboSquid.  
<https://www.turbosquid.com/3d-models/free-crt-screen-3d-model/546831>
- Benzzero (diciembre 2010). Free Keyboard 3D Model [Modelo 3D]. TurboSquid.  
<https://www.turbosquid.com/3d-models/free-keyboard-3d-model/577140>
- Gaitán, F. (1999–2001). *Yo soy Betty, la fea* [Serie de televisión]. RCN Televisión.

## 7. ENGLISH TECHNICAL MANUAL

### 7.1. INTRODUCTION

This Technical Manual covers the basic computer graphics techniques seen throughout the course, to identify the diversity of application areas. Several tools were used, including 3D modeling and image editing, to finally integrate them into the OpenGL programming environment. As a final product, a two-story house with a room inside was virtually recreated.

The execution of this project requires the application of a software methodology, as well as the modeling of five objects in two rooms, the implementation of a minimum of four contextualized animations, the management of textures and lighting to achieve a result as close as possible to a realistic scenario (compared to the reference images in this documentation) to implement acceptable human-computer interaction.

Additionally, a cost analysis will be included to estimate the financial parameters of the development of this project in the labor market.

The Technical Manual is available in its Spanish and English versions in this same document.

### 7.2. PROJECT OBJECTIVES

**General Objective:** To apply and demonstrate the techniques of projections, transformations, geometric and hierarchical modeling, model loading, etc., acquired throughout the course.

#### Specific objectives:

- Recreate in 3D using the OpenGL language Betty's house from the Colombian novel *Yo soy Betty, la fea* (Gaitán, F. 1999), including the house's façade with a view of the street and the sidewalk surrounding it. As well as including the internal structure of the house and the existing rooms.

- Recreate five contextualized objects in two rooms of the house (the dining room and Betty's bedroom), and set them up according to the Reference Images found in this Manual.
- Implement four animations with logic and context within the virtual environment.
- Present a cost analysis of the project.

## 7.3. PLANNING AND DEVELOPMENT

### 7.3.1. APPLIED SOFTWARE METHODOLOGY

The development of this project was carried out through the application of an Adapted Waterfall Model (Figure 7.3.1.1).



Figure 7.3.1.1. Waterfall methodology.

This methodology was selected due to the sequential development of the project, where the design and modeling of the geometry was indispensable and non-negotiable before implementing interactive functionalities, such as animations, lighting, and camera handling.

The model was implemented in the following sequential phases:

#### 1. Requirements analysis and conceptualization:

- Selection and obtaining of the façade and the interior/exterior reference space.
- Definition of the five objects to model in each room.
- Definition of the four required animations and their contexts.

## 2. Architectural design and geometric modeling:

- Modeling of the house's geometry (façade and space), as well as including doors, windows, rooms, and stairs.
- Detailed modeling of the five objects in each room defined in the previous phase.
- Integration of the models with the OpenGL base code.

## 3. Implementation of functionalities and texturing:

- Search and application of texturing and materials to the five objects and subsequently to the façade.
- Coding of the transformations for the four animations.
- Ambient adjustment through the correct use of lighting.
- Development of camera handling.

## 4. Project cost analysis:

- Collection of resources, personnel, and time invested in the development of the project.
- Comparison against average quotes from the labor market.

## 5. Testing and documentation:

- Functionality tests to ensure that all animations, camera interactions, and the executable work.
- Review of compliance with the object weight limits (less than 100 MB).
- Geometry cleanup and economization of texture and model size.
- Drafting of the Technical Manual and the User Manual in Spanish and English.

### 7.3.2. GANTT CHART

Assignment date: October 06, 2025.

Delivery date: November 25, 2025.

Figure 7.3.2.1 shows a Gantt chart illustrating the progress of the development of the second final project over approximately 7 weeks, from the assignment date to the delivery date.

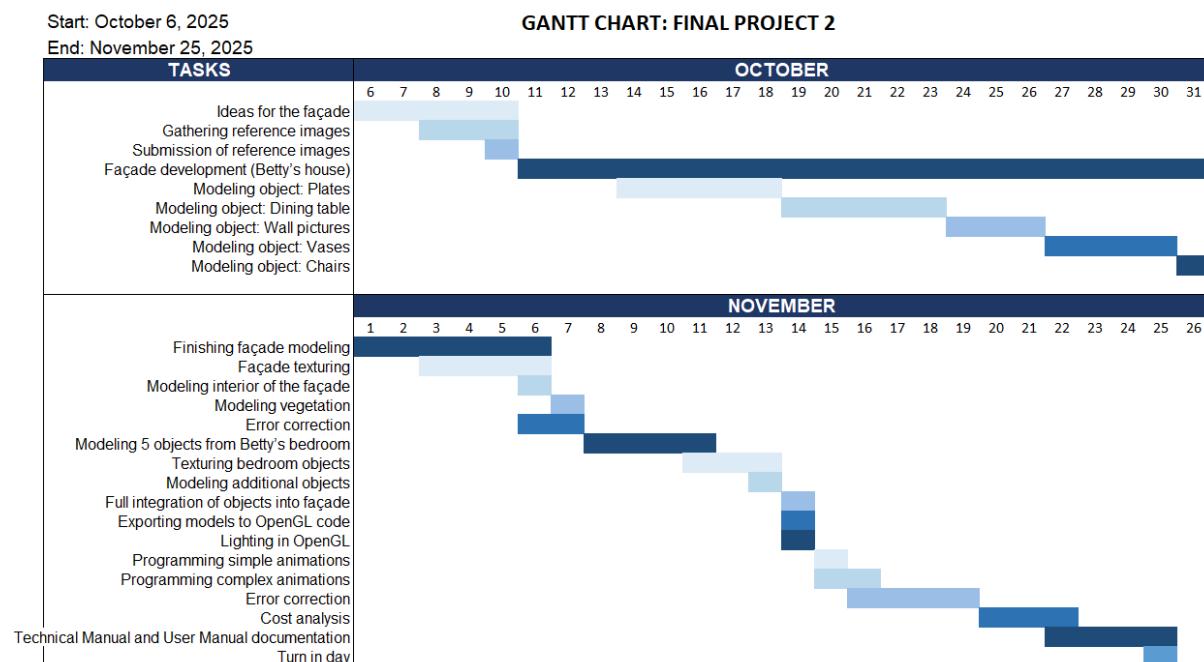


Figure 7.3.2.1. Gantt Chart.

### October: Modeling and Structure

The main focus of October is the geometric modeling phase. The tasks center on creating the base structure and objects:

- The Modeling of the 5 required Objects for each scene is initiated.
- The Loading and Integration of Models with the course's base code is carried out.
- Concurrently, the house's façade is developed.

### November: Closing, Polishing, and Documentation

November is the month for the Implementation of Functionalities, transforming the static models into an interactive environment, and finally for closing and documentation, ensuring complete functionality and formal delivery.

- The coding of the 4 animations is finalized.
- The camera handling system is implemented.
- Functional tests and quality control are performed.
- The cost analysis is carried out.
- The drafting of the Technical Manual and User Manual is completed.
- Finally, the executable is generated and the project is uploaded to GitHub, meeting the final delivery date of November 25.

### 7.3.3. PROJECT SCOPE

The project establishes as its primary scope the three-dimensional (3D) recreation of a façade and two specific interior/exterior spaces, using a common base code and the OpenGL library. The project must apply and demonstrate the knowledge acquired throughout the semester, which includes:

- Projections, viewports, and geometric transformations.
- Geometric modeling.
- Hierarchical modeling.
- Model loading and synthetic camera.
- Texturing.
- Lighting.
- Animation.

### 7.3.4. LIMITATIONS

Within the limitations in the development of the project are the following:

- a) Forbidden Subject Matter: Any space in university city, *Rick and Morty*, *The Simpsons*, the Kame House from *Dragon Ball*, *The Powerpuff Girls* house, the

*Gravity Falls* cabin, *Garfield's* house, and the *Courage the Cowardly Dog* house are excepted.

- b) File Weight: Every recreated or downloaded object must weigh less than 100MB.
- c) Repeated Objects: Repeatedly recreated objects will count as a single object for the evaluation of the 5 elements in each room.
- d) Repeated Animations: Repeated animations (such as doors and windows that do the same thing) count as a single animation.

## 7.4. TECHNICAL DEVELOPMENT

### 7.4.1. TECHNICAL OBJECTIVES AND SCOPES

- Recreate with precision a minimum of five distinctive objects from the reference image in the two rooms, with correct geometry and texturing to achieve a high level of realism in the virtual space.
- Code and execute a total of four functional and contextual animations, which will be simple and complex (non-linear).
- Correct use of lighting to achieve an adequate and realistic ambiance within the virtual space.
- Implement a functional and documented camera handling system that allows user interaction with the recreated environment.
- Ensure that every modeled or downloaded object weighs less than 100 MB.

### 7.4.2. MODELING

The modeling of the entire scene, including the façade, the two interior/exterior spaces, and the five (5) main objects required for each room, was carried out using the Autodesk Maya 2023 software. Maya was selected as the main tool for geometry creation due to its technical capabilities, essential for a project based on OpenGL.

The choice of Maya 2023 over other alternatives is due to its geometric precision and its control over mesh topology, which greatly supports the creation of models with a low polygon count to economize computational resources.

Also, for the configuration and power of the UV maps it possesses, which is quite complete for correctly modifying the textures on the modeled objects and achieving greater realism and space economization, as the case may be.

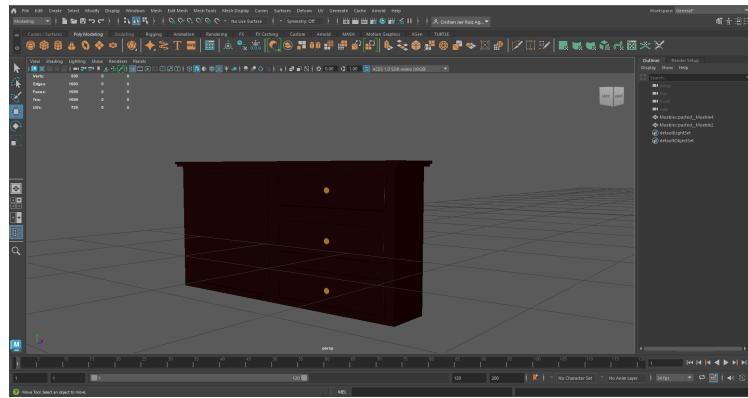


Figure 7.4.2.1. Maya 2023 Interface modeling one of the 5 dining room objects.

The objects modeled from the polygons provided by Maya were:

- Façade (Betty's house): garden, tree leaves, sidewalk, railings, ground floor, and first floor.
- Doors.
- Windows.
- Rug.

The objects belonging to the dining room are:

- Picture frames with photos.
- Chairs.
- Tablecloth.
- Vases.
- Plates.

The objects belonging to Betty's bedroom are:

- Bed.
- Nightstands next to the bed.
- Desk.
- Desktop computer.
- Wardrobe.

The objects extracted from the Internet<sup>2</sup> and adapted to the project were:

- Tree trunk.
- Vases.
- Butterfly.
- Chairs.
- Monitor.
- Keyboard.
- Pillows.

#### 7.4.3. TEXTURING AND UV MAPPING

The realism of the virtual environment is based on the correct application of textures on the modeled geometry. This process is mediated by UV Mapping, an essential technique in the 3D graphics pipeline that allows projecting two-dimensional images onto three-dimensional surfaces.

##### 1. Texturing

Texturing is the phase where image maps (textures) are assigned to 3D models to simulate the physical and visual properties of materials (color, roughness, specularity, etc.). This process was performed multiple times with the support of Maya software to assign the respective texture to each material in the project, as shown in Figure 7.4.2.2.

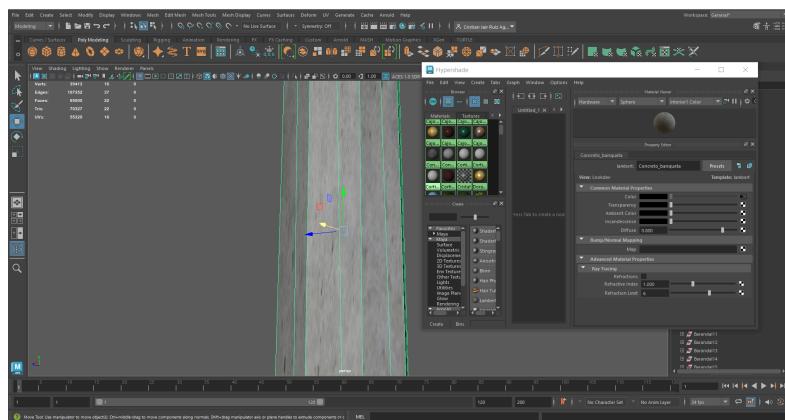


Figure 7.4.2.2. Material Assignment in Maya software.

<sup>2</sup> The credits for the objects downloaded from the Internet are found in the References of this document.

It is worth mentioning that for the implementation of the textures, care was taken that all the images used, regardless of their absolute size, had dimensions that were a power of two. This means that both the width (W) and the height (H) of the images must be values like 2, 4, 8, 16, 32, ..., 512, 1024, 2048, etc. (Table 7.4.2.2).

Texture	Pixels size
 Living room	256 x 256
 Tree	128 x 128
 Chocolate wood	8 x 8

Table 7.4.2.2. Example of some textures and their sizes.

## 7. UV Mapping

UV Mapping is the tool to work the technical interface between the 2D texture and the 3D model. It consists of creating a coordinate system (U, V) that maps each point on the surface of the 3D mesh (X, Y, Z) to a specific point on the 2D plane of the texture.

Maya software possesses various tools for working with the texture coordinates of the modeled objects; however, the most frequently used technique was to transform the textured surfaces into planar surfaces, then oriented to the axis they were facing. This way, the textures did not look stretched and appeared more realistic. Figure

7.4.2.3 shows the editing of a brick texture wall, to which a planar type surface was required to be applied towards the x-axis for greater realism.

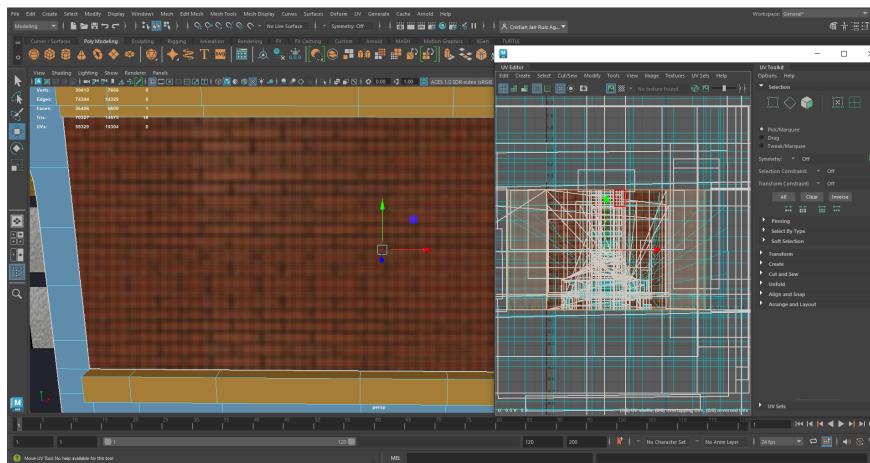


Figure 7.4.2.3. Maya UV Editor Interface.

#### 7.4.4. CODE STRUCTURE

The project structure follows a monolithic design that is mainly based on a single C++ source file that integrates both the initialization and control logic and the rendering functions. Microsoft Visual Studio software was used to perform all the project's programming functions.

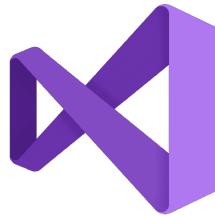


Figure 7.4.4.1. Microsoft Visual Studio Logo.

The code is organized into the following logical and functional sections:

**1. Libraries and Dependencies:** Standard C++ libraries and the main graphics library are included, along with the external dependencies required for advanced functionality, as shown in Figure 7.4.4.2:

```
#include <iostream>
#include <cmath>

// GLEW
#include <GL/glew.h>

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

// Other Libs
#include "stb_image.h"

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

// Load Models
#include "SOIL2/SOIL2.h"

// Other includes
#include "Shader.h"
#include "Camera.h"
#include "Model.h"
#include "Texture.h"
```

Figure 7.4.4.2. Standard and graphical libraries of the cpp code.

- **#include <windows.h>**: For specific Windows operating system functions.
- **#include <GL/glew.h>**: Used to manage OpenGL extensions for compatibility with the most modern features.
- **#include <GL/glut.h>**: The base library for window management and input events.
- **#include <iostream>, <vector>, <stdlib.h>, <math.h>**: Standard libraries for mathematical operations, memory management, and input/output.
- **#include "Camera.h", #include "Model.h**: Own headers that contain the class definition for camera management and loading external 3D models.

**Global and State Variables:** Global variables are declared to store the state of the scene, such as the position, rotation, and scale of the objects.

```

// Window dimensions
const GLuint WIDTH = 800, HEIGHT = 600;
int SCREEN_WIDTH, SCREEN_HEIGHT;

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

// ===== SIMPLE ANIMATIONS =====

// CORTINA1 ANIMATION
bool Cortina_abierta1 = true; // Indica si la cortina esta abierta
bool anima_cortina1 = false; // Indica si la animacion esta en curso
float cortinaAbre1 = 1.0f; // cantidad de escalamiento (apertura) de la cortina

```

Figure 7.4.4.3. Declaration of window, camera, light, and a simple animation variables.

**Initialization Functions `init()`:** This function is executed only once at the beginning of the program and is responsible for:

- Configuring the OpenGL context and rendering modes (e.g., Depth Test).
- Initializing the GLEW library.
- Loading textures and 3D models (`Model.h`) from external files.
- Establishing the initial configuration of the Ambientation (lighting and materials).

### Main Rendering Loop (Game Loop):

- The drawing logic is found within the `while`

```

// Game loop
while (!glfwWindowShouldClose(window))
{

```

- This loop is responsible for clearing the buffers, applying the view matrix (camera) and the projection matrix, and making calls to the functions that contain the drawing code for the façade, space, the 5 objects, and the general ambientation, ensuring visual update in each frame. This is observed in Figure 7.4.4.4.

```

// Game loop
while (!glfwWindowShouldClose(window))
{
    // Calculate deltatime of current frame
    GLfloat currentTime = glfwGetTime();
    deltaTime = currentTime - lastFrame;
    lastFrame = currentTime;

    // Check if any events have been activated (key pressed, mouse moved etc.) and call corresponding response functions
    glfwPollEvents();
    DoMovement();

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

    // OpenGL options
    glEnable(GL_DEPTH_TEST);

    //Load Model

    // Use cooresponding shader when setting uniforms/drawing objects
    lightingShader.Use();

    glUniform1i(glGetUniformLocation(lightingShader.Program, "material.diffuse"), 0);
    glUniform1i(glGetUniformLocation(lightingShader.Program, "specular"), 1);

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

```

Figure 7.4.4.4. Initial code of the "Game Loop".

**Interaction and Event Functions keyboard(), mouse():** They contain the logic for managing user inputs (keyboard and mouse), allowing for:

- Activation and control of the 4 animations.
- Control of camera handling for user navigation in the virtual environment.

```

// Is called whenever a key is pressed/released via GLFW
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode)
{
    if (GLFW_KEY_ESCAPE == key && GLFW_PRESS == action)
    {
        glfwSetWindowShouldClose(window, GL_TRUE);
    }

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

    if (keys[GLFW_KEY_1] && !anima_cortina1) { // ANIMATION CORTINA1
        anima_cortina1 = true;
    }

    if (keys[GLFW_KEY_2] && !anima_cortina2) { // ANIMATION CORTINA2
        anima_cortina2 = true;
    }

    if (keys[GLFW_KEY_3] && !anima_reja) { // ANIMATION REJA
        anima_reja = true;
    }

    if (keys[GLFW_KEY_4] && !anima_puertaEntrada) { // ANIMATION PUERTA ENTRADA
        anima_puertaEntrada = true;
    }
}

```

Figure 7.4.4.5. Code block for keyboard input management.

```

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

    GLfloat xOffset = xPos - lastX;
    GLfloat yOffset = lastY - yPos; // Reversed since y-coordinates go from bottom to left

    lastX = xPos;
    lastY = yPos;

    camera.ProcessMouseMovement(xOffset, yOffset);
}

```

Figure 7.4.4.6. Code block for mouse input management.

**Shaders:** are executed directly on the GPU for modern OpenGL rendering.

- Vertex Shader (.vs): Is executed for each vertex of the geometry. Its main function is to apply geometric transformations (Model, View, Projection) to position the models on the screen.
- Fragment Shader (.frag): Is executed for each potential pixel and determines its final color.

You can have several Shaders because the rendering pipeline is a multi-stage process, and each Shader is specialized to execute specific tasks on different types of data, such as drawing or complex animations. The shaders included in the project (Figure 7.4.4.7) cover tasks such as managing colors, lighting, complex animations, SkyBox handling, and loading models and their textures.

```

anim.frag
anim.vs
Anim2.frag
Anim2.vs
lamp.frag
lamp.vs
lighting.frag
lighting.vs
modelLoading.frag
modelLoading.vs
SkyBox.frag
SkyBox.vs

```

Figure 7.4.4.7. Shaders contained in the project.

#### 7.4.5. MODEL LOADING

Models are exported to a standard 3D exchange format, such as .OBJ or .FBX. In this project, the .OBJ format was used. These files contain the mesh data:

- Vertex Position: Coordinates (X, Y, Z).
- Normal Vector:
- Essential for lighting calculation.
- Texture Coordinates: Coordinates (U, V) that link the geometry with the texture.

These models were stored in their respective folder within the Models directory. Each model with its respective .mtl file and textures.

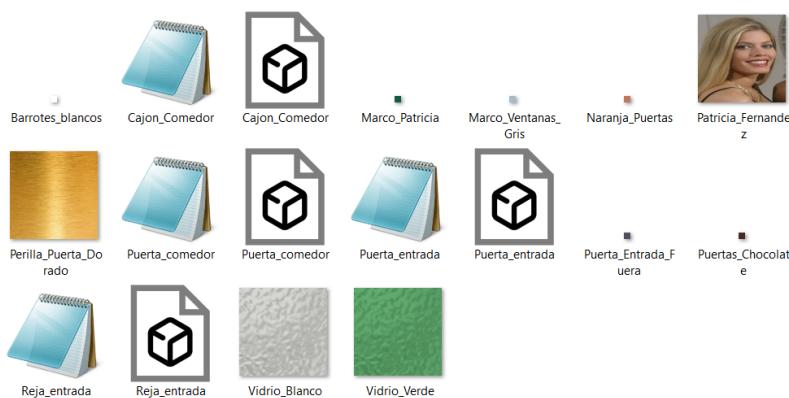


Figure 7.4.5.1. Object files exported from Maya.

The C++ code uses the auxiliary library (Model.h) to read and interpret the exported file, and its path is specified to load them as shown below (Figure 7.4.5.2).

```

// OBJECTS

Model CasaBety((char*)"Models/CasaBety/Casa_de_Bety.obj");
Model Platos((char*)"Models/Platos/Platos.obj");
Model BaseMesa((char*)"Models/MesaComedor/Mesa.obj");
Model Sillas((char*)"Models/Sillas/Sillas.obj");
Model Vasijas((char*)"Models/Vasijas/Vasijas.obj");
Model Cuadros((char*)"Models/Cuadros/Cuadros.obj");
Model Muebles((char*)"Models/Muebles/Muebles.obj");

// OBJECTS WITH SOME TRANSPARENCY

Model MantelMesa((char*)"Models/MesaComedor/Mantel_mesa.obj");
Model Vasos((char*)"Models/Vasos/Vasos.obj");
Model Vidrios((char*)"Models/Ventanas/Vidrios.obj");

// SIMPLE ANIMATION OBJECTS

Model Reja_entrada((char*)"Models/Animaciones_sencillas/Reja_entrada.obj");
Model Cortina1((char*)"Models/Cortinas/Cortina1.obj");
Model Cortina2((char*)"Models/Cortinas/Cortina2.obj");
Model PuertaEntrada((char*)"Models/Animaciones_sencillas/Puerta_entrada.obj");
Model PuertaComedor((char*)"Models/Animaciones_sencillas/Puerta_comedor.obj");
Model Cajon((char*)"Models/Animaciones_sencillas/Cajon_Comedor.obj");

```

Figure 7.4.5.2. Loading Models to OpenGL.

#### 7.4.5.1. SKYBOX

The Skybox implementation follows the OpenGL standard using the Cube Maps technique so that the environment is perceived as three-dimensional and immersive. These are loaded and bound into a single texture of type:

*GL\_TEXTURE\_CUBE\_MAP*

The Skybox is rendered using the depth function configured to:

*GL\_LESS*

This ensures that the background cube is drawn at the farthest depth positions so that all objects in the scene are drawn over it.

```
//SkyBox
GLuint skyboxVBO, skyboxVAO;
glGenVertexArrays(1, &skyboxVAO);
glGenBuffers(1, &skyboxVBO);
 glBindVertexArray(skyboxVAO);
 glBindBuffer(GL_ARRAY_BUFFER, skyboxVBO);
 glBufferData(GL_ARRAY_BUFFER, sizeof(skyboxVertices), &skyboxVertices, GL_STATIC_DRAW);
 glEnableVertexAttribArray(0);
 glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);

// Load textures
vector<const GLchar*> faces;
faces.push_back("SkyBox/right.tga");
faces.push_back("SkyBox/left.tga");
faces.push_back("SkyBox/top.tga");
faces.push_back("SkyBox/bottom.tga");
faces.push_back("SkyBox/back.tga");
faces.push_back("SkyBox/front.tga");

GLuint cubemapTexture = TextureLoading::LoadCubemap(faces);
```

Figure 7.4.5.1.1. Buffer Management and Skybox Model Loading.

#### 7.4.6. SYNTHETIC CAMERA

Loading Library (Model.h): The project uses a Free Look camera model implemented in the auxiliary class Camera.h. This allows the user to move freely through the virtual space, navigating around the façade and through the interior/exterior. The camera is defined by three key vectors:

- Position: The current location of the camera in space (X, Y, Z).
- Front Direction (Front): The vector that points to where the camera is looking.
- Up Direction (Up): The vector that defines the vertical orientation of the camera.

```
// Camera
Camera camera(glm::vec3(0.0f, 0.0f, 3.0f));
```

Figure 7.4.6.1. Initial Camera Position.

#### Camera Handling (Interaction):

- Translational Movement: Frontal, lateral, and vertical movement is controlled by keyboard callbacks (KeyCallback), modifying the camera's Position vector. Movement speed is constant.
- Rotational Movement (Look Around): Camera rotation is managed by mouse callbacks (MouseCallback). Mouse movements (pitch and yaw) modify the Front Direction vector, allowing the user to look around. This ensures

compliance with the camera handling requirement and provides a complete interactive experience.

#### 7.4.7. LIGHTING

The lighting configuration is based on the combination of three main components:

- **Ambient Light:** Represents indirect or scattered light in the scene. It ensures that no part of the model is completely dark, even if it is not directly illuminated by a light source.
- **Diffuse Light:** It is the most significant component for the object's color and brightness. It reflects light that is scattered uniformly in all directions after hitting the surface. It directly depends on the direction of light and the vertex normal.
- **Specular Light:** Simulates the "bright spots" or reflections that appear on polished or shiny surfaces. It depends on the camera position (the view direction) and the direction of light. It is essential for giving a sense of materiality.

```
// Directional light
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.668f, 0.649f, 0.582f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.668f, 0.649f, 0.582f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.5f, 0.5f, 0.5f);
```

Figure 7.4.7.1. Code Block for Configuring Directional Light.

#### 7.4.8. ANIMATIONS

Animations are handled in both the KeyCallBack and DoMovement functions to ensure they are activated when the user presses a key, and then the animation is performed, and does not stop until the programmed movement concludes.

From the Maya software, the animated objects were configured to be independent of the facade and to place their pivot at the center of the world, since OpenGL takes that world center as a reference. Subsequently, with OpenGL transformations, they were placed in their respective locations.

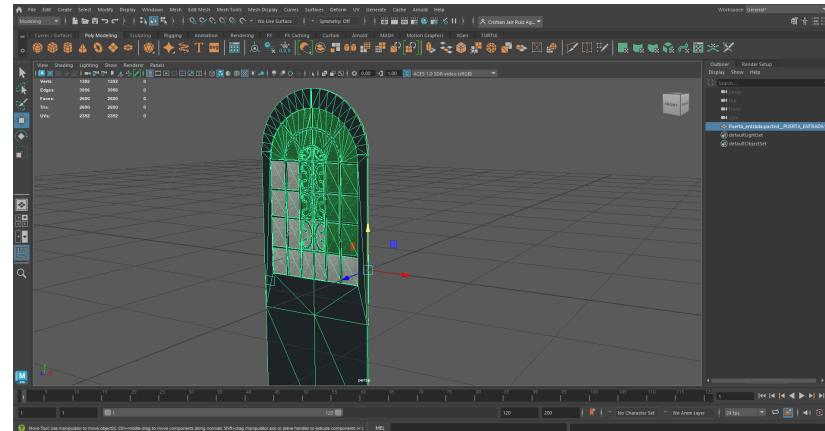


Figure 7.4.8.1. Configuration of animatable objects at the center of the world.

```
// PUERTA ENTRADA
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-11.972f, 1.027f, -6.617f));
model = glm::rotate(model, glm::radians(puertaEntradaAbre), glm::vec3(0.0f, 1.0f, 0.0f));

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
PuertaEntrada.Draw(lightingShader);
```

Figure 7.4.8.2. Animatable object with transformations to position it according to the façade.

### 7.4.8.1. SIMPLE ONES

**1. Dining room curtains:** The curtains have a variable that reduces their scale on one axis to give the impression that they are opening.

```
//ANIMATION CORTINA1
if (anima_cortina1) {

    if (!Cortina_abierta1) { // Si esta cerrada

        if (cortinaAbrel < 1.0f) { // Abre
            cortinaAbrel += 0.05f;
        }
        else {
            Cortina_abierta1 = true; // ya esta abierta
            anima_cortina1 = false;
        }
    }
    else { // Si esta abierta

        if (cortinaAbrel > 0.225f) {
            cortinaAbrel -= 0.05f;
        }
        else {
            Cortina_abierta1 = false; // ya esta cerrada
            anima_cortina1 = false;
        }
    }
}
```

Figure 7.4.8.1.1. Code block with the logic to handle the curtain animation.

**2. Entrance gate, entrance door, dining room door, drawer:** A rotation variable was assigned to simulate limited opening to the three objects with the same logic shown in the following image.

```
//ANIMATION REJA
if (anima_reja) {

    if (!rejaAbierta) { // Si esta cerrada

        if (rejaAbre > -29.236f) { // Abre
            rejaAbre -= 2.5f;
        }
        else {
            rejaAbierta = true; // ya esta abierta
            anima_reja = false;
        }
    }
    else { // Si esta abierta

        if (rejaAbre < 46.297f) {
            rejaAbre += 2.5f;
        }
        else {
            rejaAbierta = false; // ya esta cerrada
            anima_reja = false;
        }
    }
}
```

Figure 7.4.8.1.2. Code block with the logic to handle the door animation.

#### 7.4.8.2. COMPLEX ONES

Complex animations are mainly differentiated by the fact that they implement mathematical equations to, in the same way, be independent of user interaction.

##### 1. Tree Leaves:

A time variable is used, which is continuously updated in the Main Rendering Loop and passed to the Vertex Shader as a uniform. The sinusoidal movement is calculated based on this time for continuous oscillation. The degree of displacement of each vertex is modulated using its original 'Y' coordinate (height).

The reason the leaves move much more at the top than at the bottom is purely algorithmic and is implemented with the line

```
windFactor *= position.y;
```

or its equivalent.

When the 'Y' position is close to zero (base of the tree), the resulting windFactor is very small, resulting in almost no movement, simulating the stiffness of the trunk. As the 'Y' position increases (moving up to the branches and leaves), the windFactor is amplified, generating a noticeable lateral displacement.

```
uniform mat4 view;
uniform mat4 projection;
uniform float time;

// Función hash para obtener un valor pseudoaleatorio
float hash(vec2 p) {
    return fract(sin(dot(p, vec2(12.9898, 78.233))) * 43758.5453);
}

void main()
{
    float normalizedHeight = aPos.y * 0.3;
    float windWave = sin(aPos.x * frequency + aPos.z * 1.5 + time * 10.0);

    float displacement = amplitude * windWave * normalizedHeight; //desplazamiento.

    vec3 newPos = aPos;
    newPos.x += displacement;
    newPos.z += displacement * 0.5; // Un desplazamiento un poco menor en Z

    gl_Position = projection * view * model * vec4(newPos, 1.0);
    TexCoords = aTexCoords;
}
```

Figure 7.4.8.2.1. Code block of the Shader that executes the tree leaves animation ([anim.vs](#)).

## 2. Butterfly

A time variable is used, which is continuously updated and passed to the Vertex Shader as a uniform.

A sinusoidal displacement factor (waveFactor) is calculated, which determines how much the vertex will move in the current frame. To make the waving more intense in certain parts of the wing (like the outer edge) and less intense at the base, the waveFactor is modulated or attenuated based on one of the vertex coordinates.

```

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

out vec2 TexCoords;

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

void main()
{
    float horizontalOffset = sin(time * 0.05) * 4.0; // Aleteo rapido
    float verticalOffset = -cos(time * 6.0) * 0.04; // Asciende y desciende rapido
    /
    float direccion = horizontalOffset >= 0.0 ? 1.0 : -1.0;

    vec4 position = vec4(aPos.x + horizontalOffset, aPos.y +verticalOffset, aPos.z, 1.0);

    gl_Position = projection * view * model * position;
    TexCoords = aTexCoords;
}

```

Figure 7.4.8.2.2. Code block of the Shader that executes the butterfly's elevation and descent animation (Anim2.vs).

The DoMovement block is used to simulate the flapping of the wings:

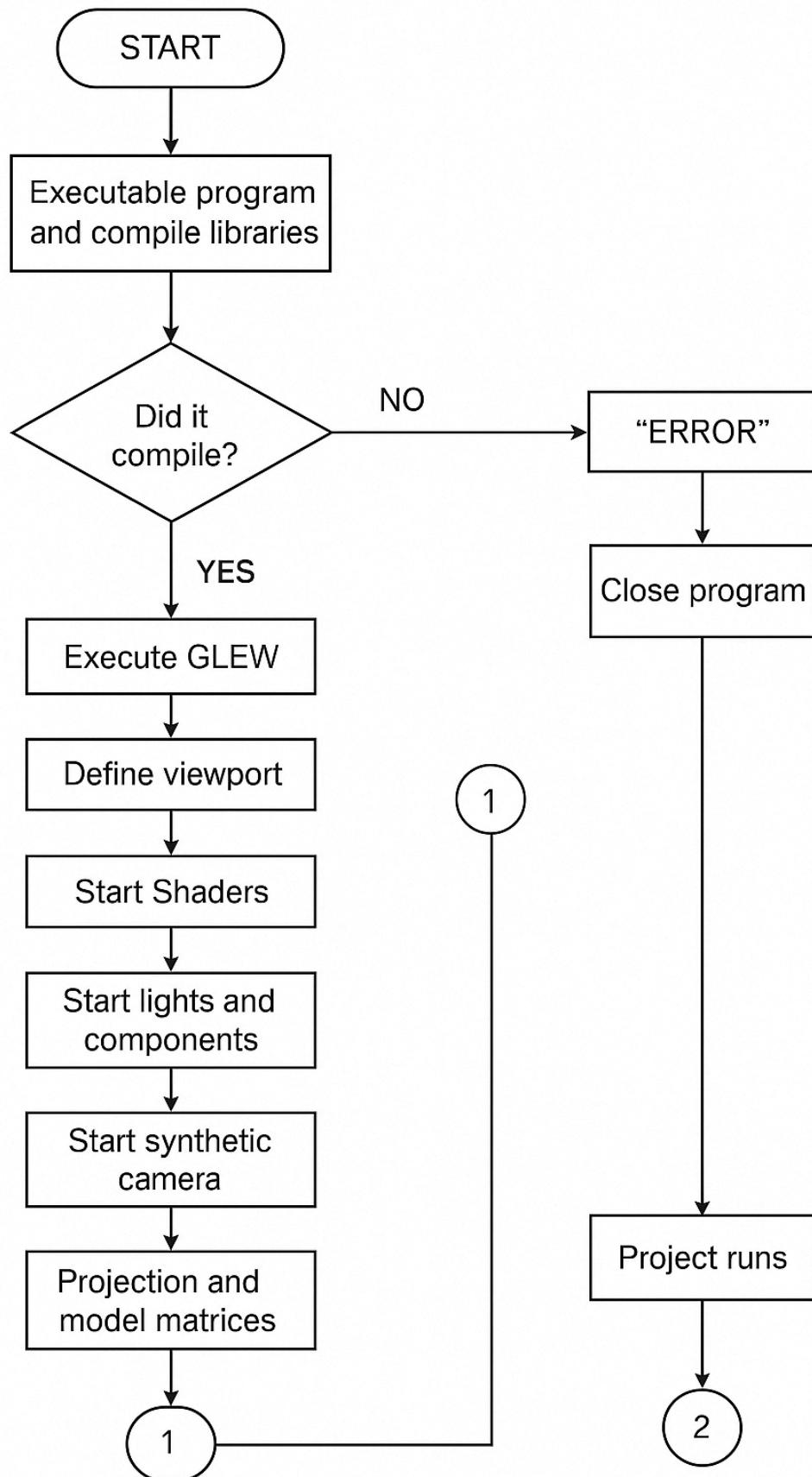
```

//ANIMATION MARIPOSA - ALETEO CONSTANTE
if (alaSubiendo) {
    aleteoAbre += 50.0f;
    if (aleteoAbre >= limiteRotacion)
        alaSubiendo = false;
}
else {
    aleteoAbre -= 50.0f;
    if (aleteoAbre <= -limiteRotacion)
        alaSubiendo = true;
}

```

Figure 7.4.8.2.3. Code block with the logic for the butterfly wing flapping.

## 7.4.9. SYSTEM FLUX DIAGRAM



## 8. VISUAL REFERENCES

Here are several screenshots and photographs of the façade (Betty's house) and the two spaces (dining room and Betty's bedroom) selected to be used as a basis and to work from.

### 8.1. COMPARATIVE REFERENCE IMAGES

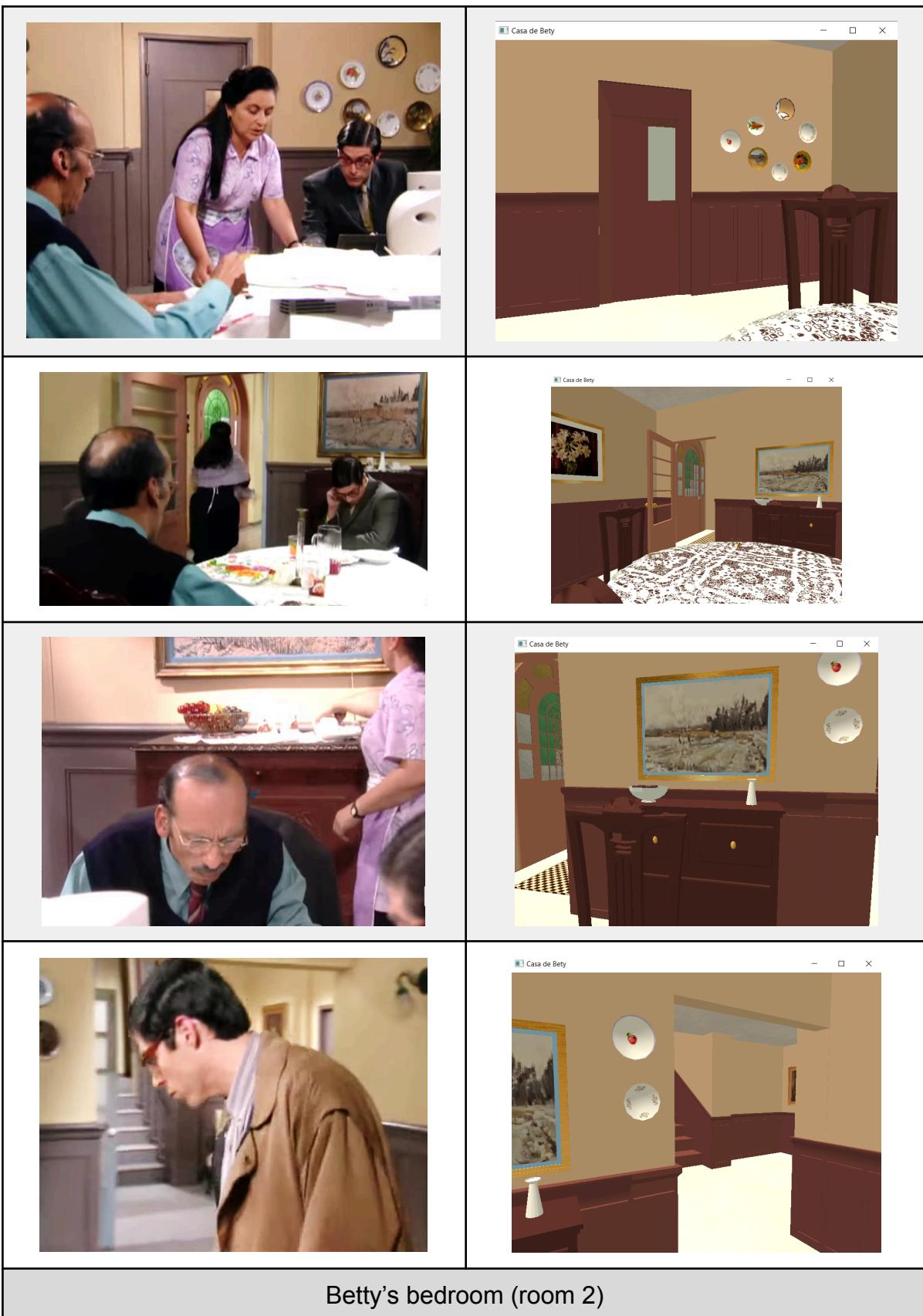
Table 8.1.1 shows a comparison between the reference images proposed for developing this project and the images that resulted after developing the entire virtual environment.

For the reference images of Betty's house dining room, the characters appearing in the scenes were discarded.

The images shown in the virtual environment of Betty's bedroom have daylight illumination, as noted in the reference images when they were proposed.

Reference Images	Virtual environment
Betty's house (façade)	
	
	





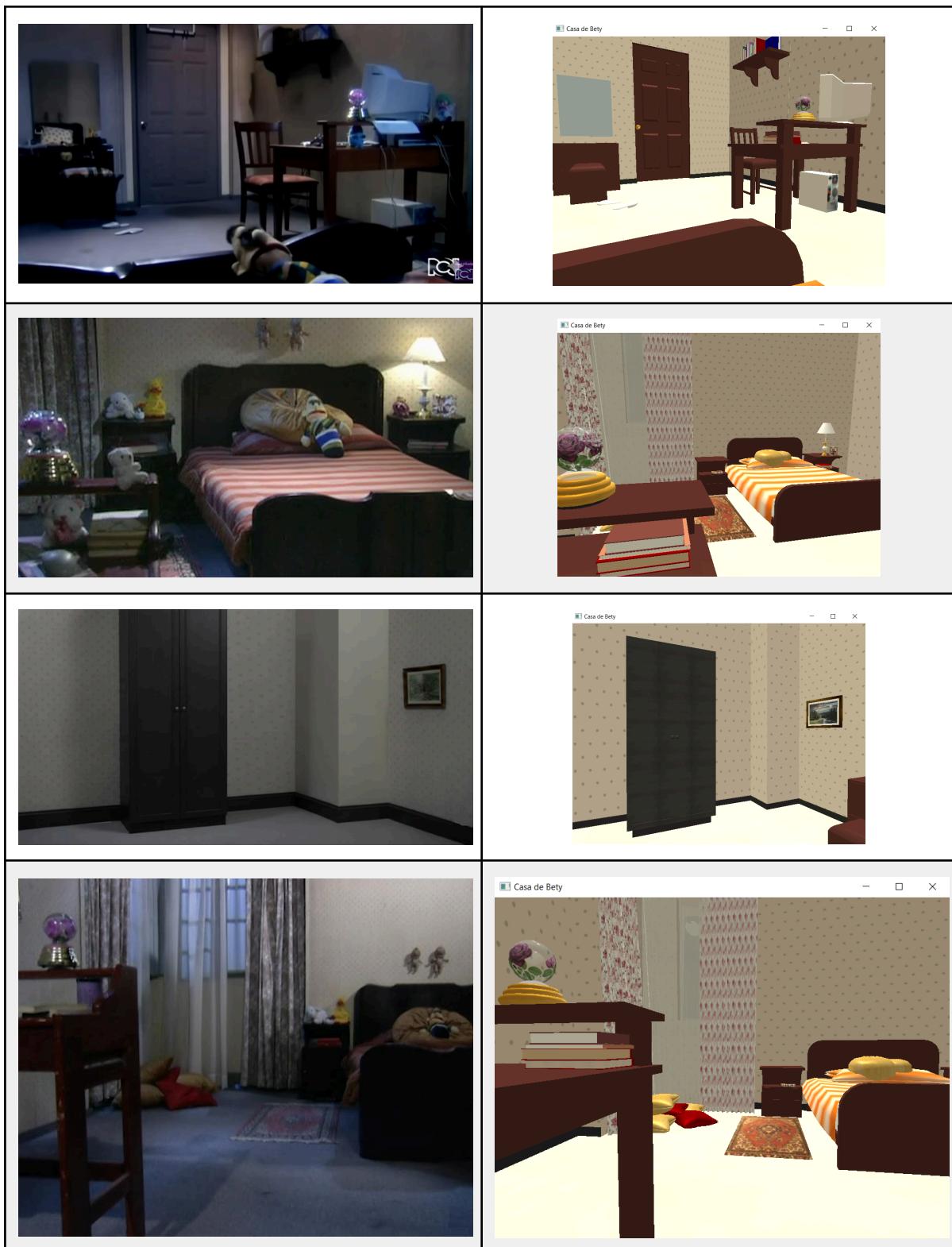
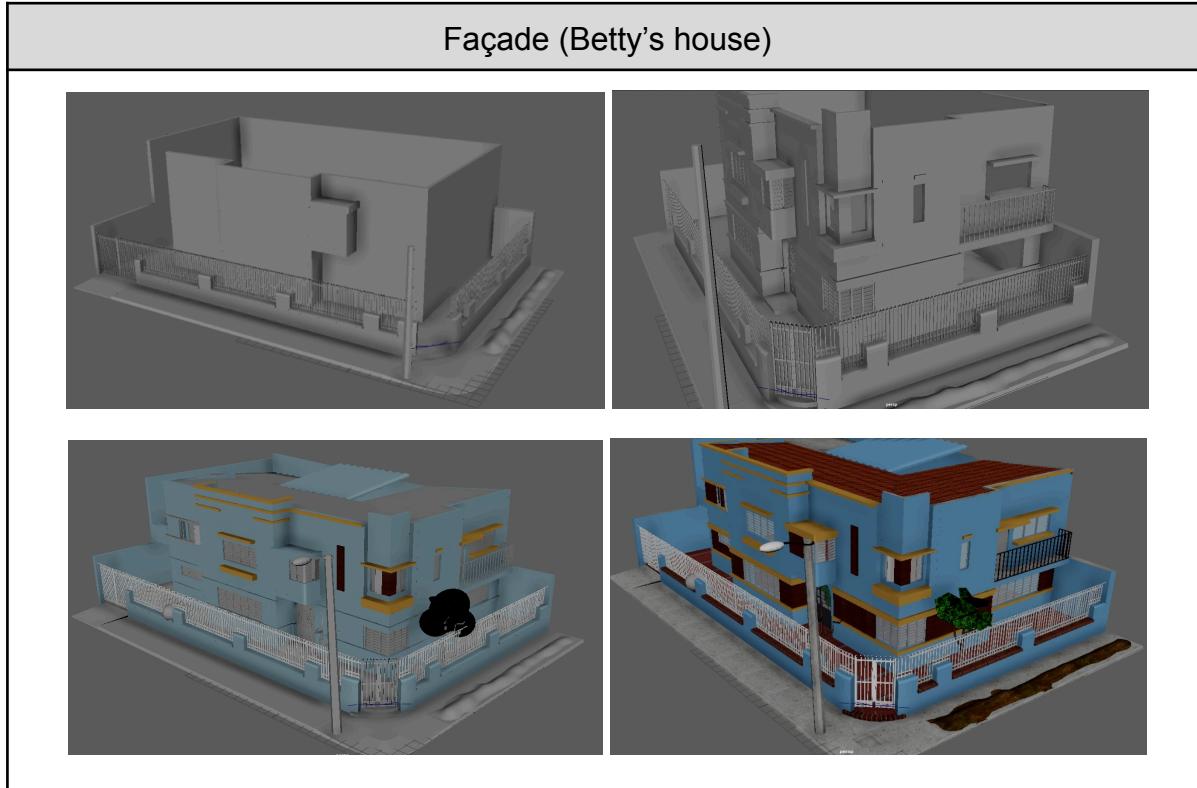


Table 8.1.1. Comparison of reference images versus project captures.

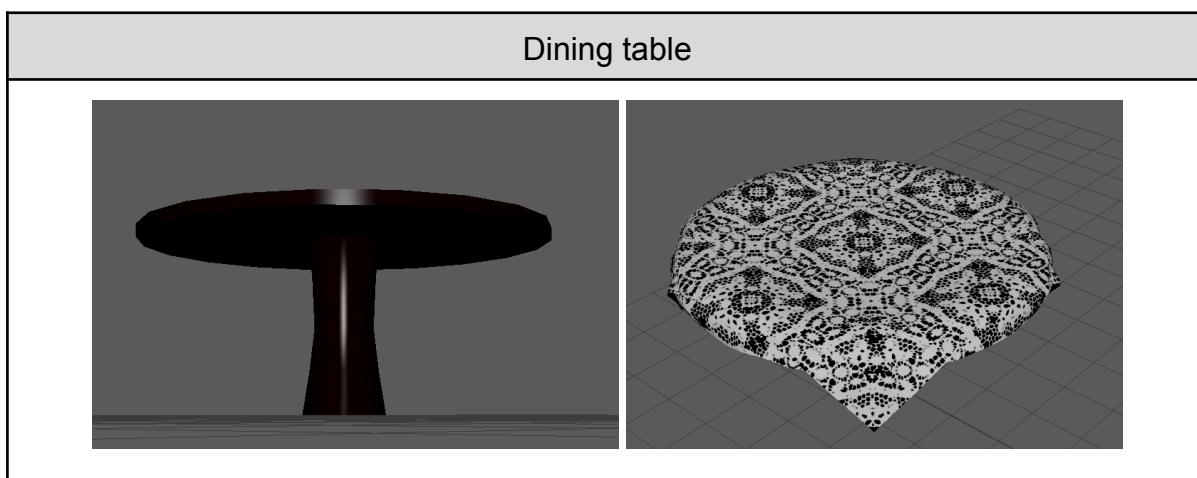
## 8.2. RECREATED OBJECTS

It is worth mentioning that this section will only include the objects listed in the Reference Images document. Extra objects, such as furniture, gardens, the street lamp post, balconies, roof, books, etc., will not be included.

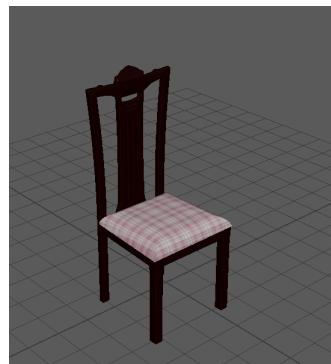
### 8.2.1. FAÇADE (BETTY'S HOUSE)



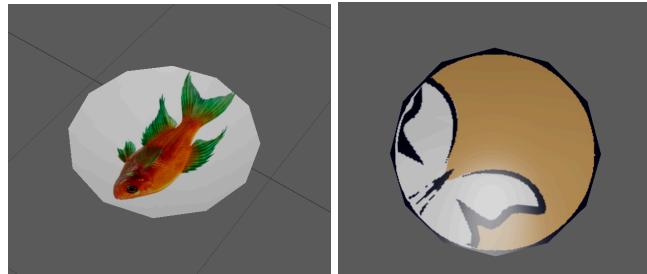
### 8.2.2. BETTY'S HOUSE DINING ROOM



Dining table's chairs



Plates at the back



Vases by the side wall

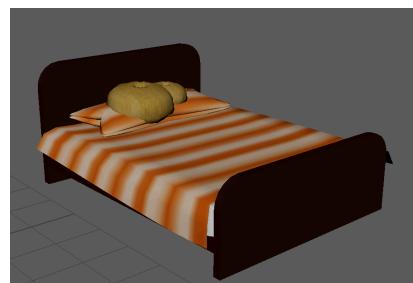


Picture frames on the three walls

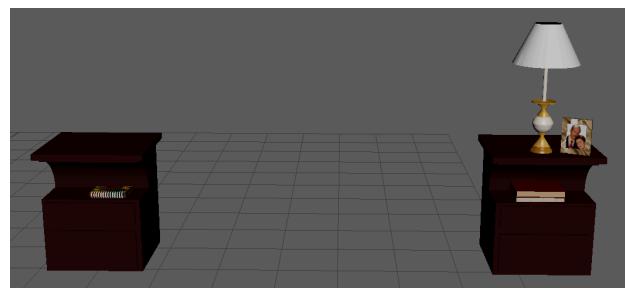


### 8.2.3. BETTY'S BEDROOM

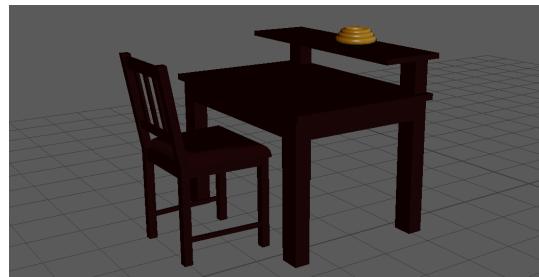
Bed



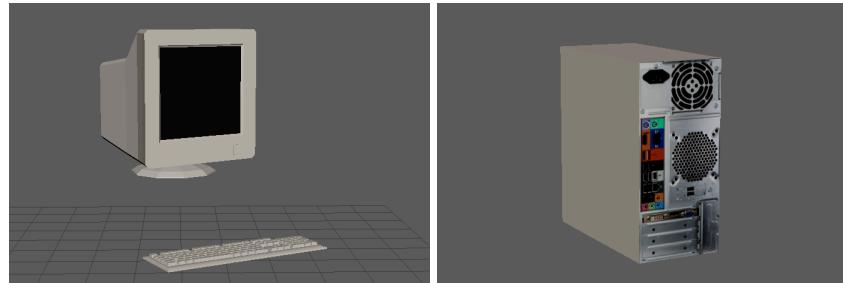
Nightstands



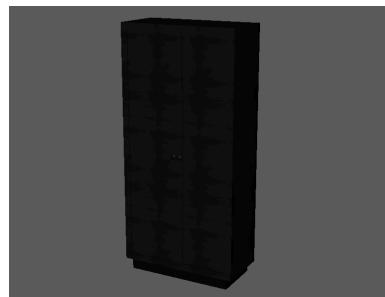
**Desk**



**Desktop computer**



**Closet**



## 9. COST ANALYSIS

This analysis estimates the real cost of producing the virtual tour in OpenGL 3, considering digital materials, software, hardware, labor, infrastructure, and a commercial margin to determine the professional selling price of the project.

### 9.1. MATERIALS

Material	Description	Estimated cost (MXN)
Own modeling	Approximately 90% of the models are handcrafted.	0
External models	6 free models from TurboSquid	600

### 9.2. SOFTWARE

Software	License	Estimated cost (MXN)
Maya 2023	Educational (free)	0
Paint Tool Sai	Paid	1000
SkyBox and others	Free	0

### 9.3. LABOR

Total of worked hours	Hourly rate	Total
150 (3hrs per day)	300 MXN	45,000 MXN

### 9.4. INFRASTRUCTURE

Concept	Estimated cost (MXN)
Electricity	25
Internet	80
Total:	105

### 9.5. TOTAL COSTS

Concept	Cost (MXN)

Materials	600
Software	400
Hardware (wear out)	150
Labor	45,000
Infrastructure	105
Documentation	3,000
Subtotal	49,255
Contingencies (10%)	4,925
Total production:	54,180

A commercial margin of 30% is applied for 3D recreation work.

**Total production cost:** 54,180 MXN.

**Suggested selling price:** 70,434 MXN.

## 10. GITHUB REPOSITORY

Link to the full documentation and content of the project in the GitHub repository:

[https://github.com/ElanRuiz/318328278\\_PROYECTOFINAL2026-1\\_GPO05](https://github.com/ElanRuiz/318328278_PROYECTOFINAL2026-1_GPO05)

## 11. CONCLUSIONS

This final project in Computer Graphics has demonstrated the integral and practical application of the knowledge acquired throughout the course, which was quite arduous to apply. The objective of recreating a 3D virtual environment was achieved through the combination of geometric modeling and advanced programming techniques in OpenGL, validating the Waterfall methodology.

The most significant learning I take away is the consolidation of the complete 3D graphics pipeline. I was able to perform the modeling in Maya with the necessary precision to recreate the façade and the spaces, ensuring that the five distinctive objects and the mandatory elements like doors and windows were as similar as possible to my reference images. I understood that realism does not depend only on geometry, but on correct texturing, which forced me to master UV Mapping so that my textures looked good in OpenGL.

At the code level, I feel I truly overcame the challenge of generating the four animations entirely through code. I managed to make them all contextual, but the real achievement was coding the two complex animations. I struggled to understand that to be complex, they had to be non-linear, which led me to implement effects in the Vertex Shader, such as the waving of the leaves and the butterfly. This technique proved to be fundamental, as was the implementation of Shaders to control the ambiance.

Having to generate the cost analysis and the Technical Manual and User Manual in Spanish and English forced me to document my work professionally and rigorously, without relying completely on translators. Furthermore, uploading everything to a functional GitHub repository that demonstrated my changes and progress during the semester taught me the importance of version control in a software project. Seeing the complete project—from conceptualization to the final executable—gives me enormous satisfaction and reaffirms that, with effort and the application of the correct methodology, I was capable of demonstrating all the knowledge I acquired in the subject.

## 12. REFERENCES

- Crick, L. J. (2018, June 22). Old pot 3D model [3D Model]. TurboSquid.  
<https://www.turbosquid.com/es/3d-models/old-pot-3d-model-1299027>
- vonherin. (2019, February 23). Low poly dead tree (simple) [3D Model]. TurboSquid.  
<https://www.turbosquid.com/3d-models/tree-simple-dead-model-1380914>
- Mughal, U. (2025, June 20). Dining table with chairs 3D [3D Model]. TurboSquid.  
<https://www.turbosquid.com/es/3d-models/dining-table-with-chairs-3d-2423456>
- Lohsu (September 2022). Old Wooden Chair 3D Model. [3D Model]. TurboSquid.  
<https://www.turbosquid.com/3d-models/old-wooden-chair-3d-3d-model-1962521>
- anastasiaartist (April 2024). Pillow 3D [3D Model]. TurboSquid.  
<https://www.turbosquid.com/3d-models/pillow-3d-2222308>
- Berna91 (July 2010). Free CRT Screen 3D Model [3D Model]. TurboSquid.  
<https://www.turbosquid.com/3d-models/free-crt-screen-3d-model/546831>
- Benzzero (December 2010). Free Keyboard 3D Model [3D Model]. TurboSquid.  
<https://www.turbosquid.com/3d-models/free-keyboard-3d-model/577140>
- Gaitán, F. (1999–2001). Yo soy Betty, la fea [Television series]. RCN Televisión.