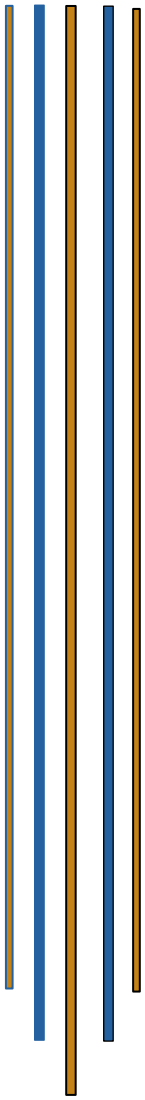
Universidad Nacional Autónoma de México



Facultad de Ingeniería

Ingeniería en Computación

Computación Grafica e Interacción Humano Computadora

Proyecto Final – Manual Técnico

Alumno:

Número de cuenta: 319095706

Grupo 05

Fecha: 25/11/2025

Índice

Objetivo.

El objetivo de este proyecto es identificar e integrar cada etapa del pipeline gráfico, configurando y utilizando correctamente las matrices de transformación para manipular la escena, aplicando un modelo de iluminación que considere componentes ambiente, difusa y especular, e implementando controles de cámara interactiva que permitan explorar el entorno 3D; además, se busca modelar objetos tridimensionales en Maya, aplicarles texturas y materiales utilizando adecuadamente mapas de textura y coordenadas UV, ajustar propiedades visuales como brillo, componente difusa, especular y opacidad, implementar diferentes tipos de luz para mejorar la estética y el realismo de la escena, simulando condiciones de iluminación interior y exterior, y finalmente ensamblar todos los objetos modelados en una composición final coherente y visualmente atractiva.

Introducción.

El presente manual describe la recreación de una fachada y dos espacios en 3D. Su objetivo es documentar de forma estructurada la arquitectura del proyecto, las dependencias empleadas, la organización del código y las modificaciones realizadas para lograr la casa, objetos y cuartos en 3D. La fachada recreada es la casa de la familia Griffin de la caricatura animada “Family Guy” y los espacios/cuartos recreados son la de la sala principal de la casa y la cocina.

Este documento no está dirigido al usuario final, sino a quien necesite entender o dar soporte al proyecto: incluye la descripción del entorno de desarrollo, estructura de directorios, archivos fuente relevantes y proceso de compilación/ejecución. De esta forma se garantiza la reproducibilidad del proyecto y la trazabilidad entre los requisitos del laboratorio y la implementación realizada.

Alcance del Proyecto.

El proyecto comprende la recreación en OpenGL de la sala y cocina de la casa de Family Guy a partir de referencias visuales, respetando la distribución general del espacio, la ambientación característica y la integración de los elementos que aparecen en la escena original. Esto implica que el escenario no es genérico, sino que se construye específicamente para parecerse al entorno de la serie, incluyendo muros, piso, puertas y los objetos que estructuran la habitación.

Dentro de este entorno se modelan y colocan al menos cinco objetos por cada cuarto claramente identificables en las referencias (capturas de la serie), cada uno con sus transformaciones correspondientes y con materiales o texturas acordes al estilo cartoon de la serie. Además, se contempla el uso del código base del laboratorio y la integración de cámara e iluminación para que la escena pueda recorrerse o visualizarse de forma correcta dentro del motor que se estuvo utilizando durante el curso.

El proyecto también abarca la implementación de las animaciones que solicita: animaciones generadas desde el código, con sentido dentro de la escena y no simples traslaciones sin contexto.

Limitantes.

Durante el desarrollo del proyecto se presentaron varias limitantes, la primera fue la dependencia de las referencias visuales de la serie, al tratarse de escenarios donde todo esta en 2D, muchas proporciones, profundidades y vistas laterales no existen tal cual, para apreciarlas, por lo que en esos detalles use la imaginación para completar los objetos. Otra limitante fue el uso del software Maya, se empezó sin tener ninguna base para su uso, pero conforme se iba avanzando con las actividades de laboratorio y a su vez con los primeros objetos para el proyecto, se fue familiarizando mas con el entorno y todas las herramientas que ofrece.

Metodología de Software aplicada.

Para desarrollar el proyecto se usaron dos metodologías de forma complementaria. Para la fachada se aplicó ***Desarrollo Iterativo***, ya que conforme avanzaba el semestre se aprendían nuevas herramientas de modelado y conceptos de iluminación. Por eso la fachada se fue rehaciendo y mejorando en ciclos: primero un boceto sencillo, luego una versión mejor estructurada, después la integración de más detalles con curvas y, al final, la aplicación de texturas y correcciones visuales.

Por otro lado, para los modelos individuales se adaptó la metodología ***SCRUM***, usando sobre todo sprints semanales con metas claras: modelar y texturizar un objeto en cada uno. Al final de cada sprint se revisaba la calidad del modelo; si no cumplía lo esperado, se programaban mejoras para la semana siguiente junto con el nuevo objeto. Cuando hubo retrasos, se reorganizaron los sprints para terminar todos los modelos, reservar uno para correcciones generales y otro final para colocar todos los objetos en la fachada y hacer el render en OpenGL, evaluando siempre cómo se veían en conjunto con la iluminación aplicada.

Diagrama de Gantt.

Actividad	Descripcion	Duracion	Inicio	Fin
act - 1	Empezar a modelar los objetos	11	19/09/2025	30/09/2025
act - 2	Empezar a modelar la fachada de la casa	4	20/09/2025	24/10/2025
act - 3	Texturizado a los objetos	22	30/09/2025	22/10/2025
act - 4	Texturizado a las paredes de la casa	10	22/10/2025	01/11/2025
act - 5	Termino de modelado	22	30/09/2025	22/10/2025
act - 6	Termino de texturizado	4	01/11/2025	05/11/2025
act - 7	Últimos detalles a la casa y objetos	3	02/11/2025	05/11/2025
act - 8	Implementación en código	2	05/11/2025	07/11/2025

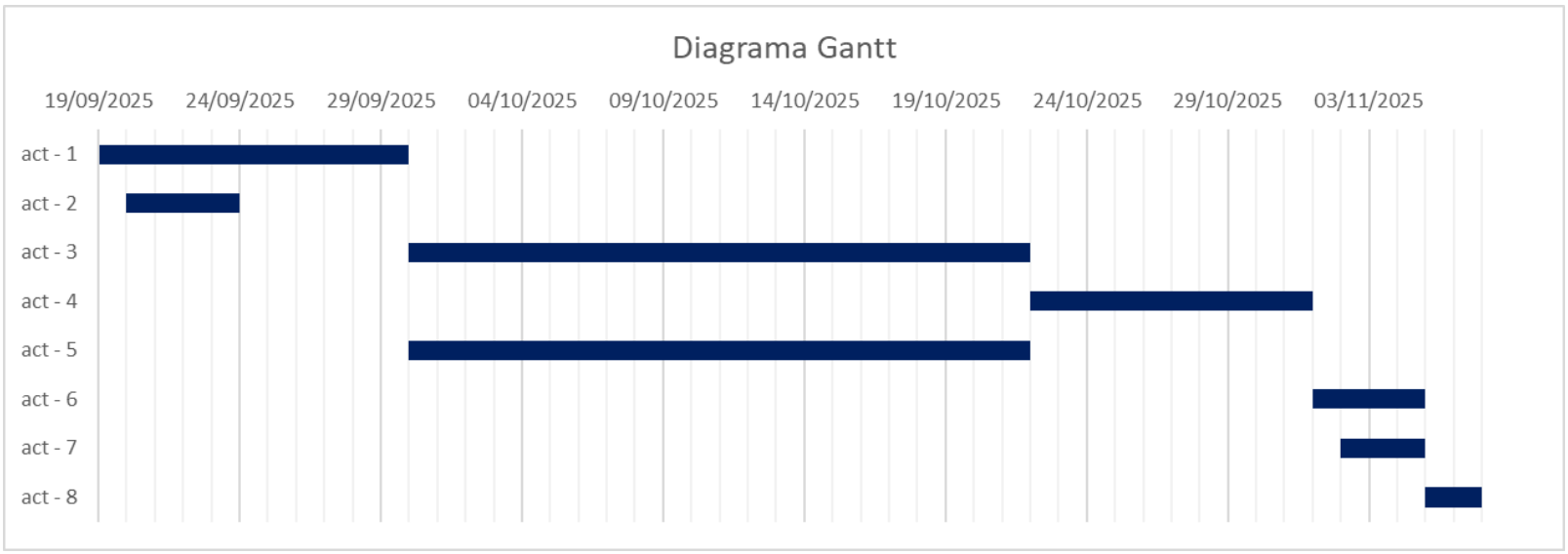
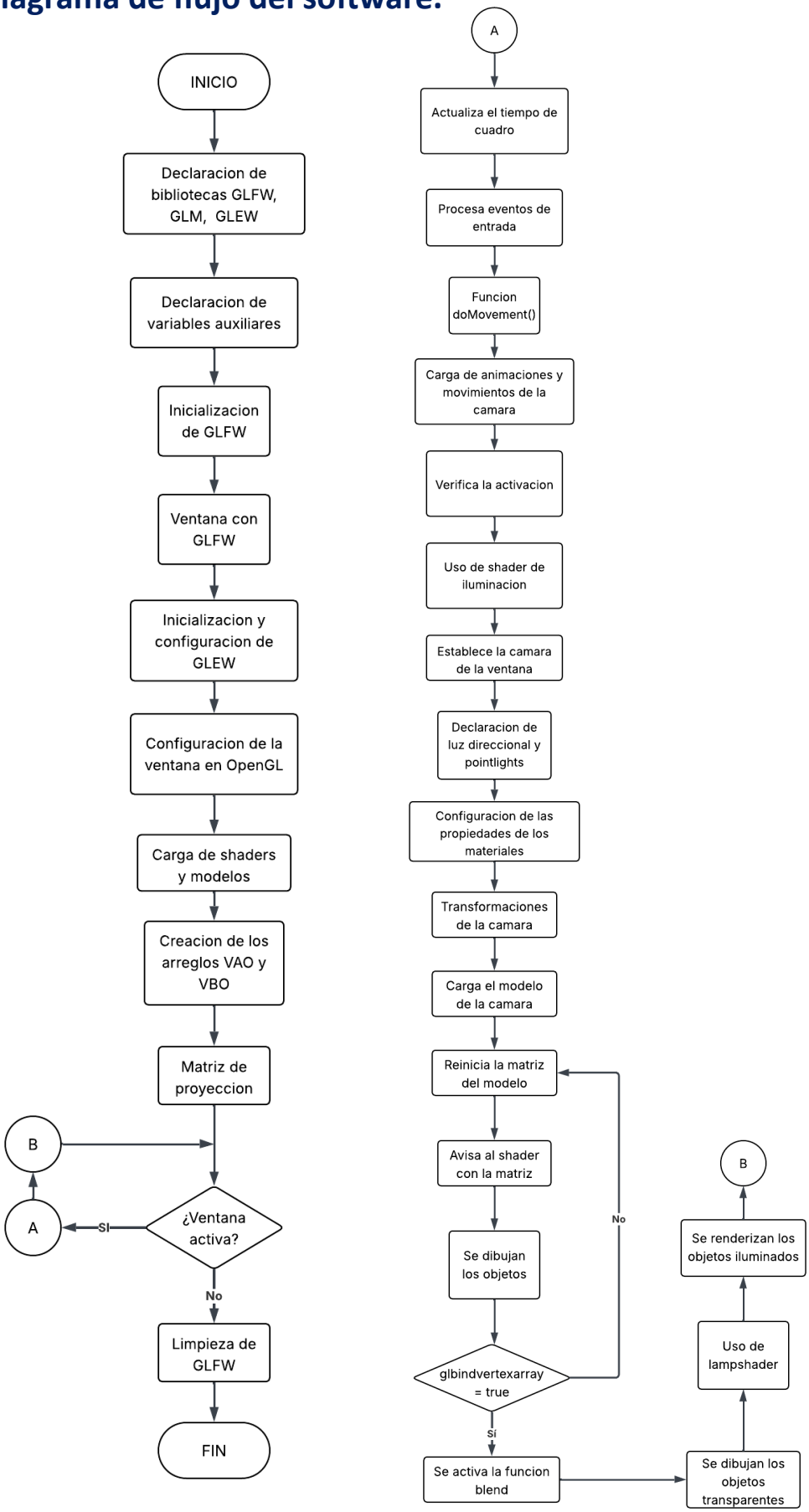


Diagrama de flujo del software.



Herramientas empleadas.

Autodesk Maya 2023.

Software de modelado, animación y render 3D profesional. Usado para crear las mallas (objetos), ajustar sus UVs, ponerles materiales básicos y hasta animarlas.

Visual Studio 2022.

IDE (entorno de desarrollo integrado) para programar en C++ (y otros lenguajes) en Windows. Ofrece: editor, resaltado de sintaxis, autocompletado, gestor de proyectos, compilador y depurador.

OpenGL.

Es una API gráfica multiplataforma para dibujar gráficos 2D y 3D usando la GPU. Conjunto de funciones que el código llama para crear ventanas de render, mandar vértices, aplicar texturas, luces, etc.

Lenguaje C++.

Es el lenguaje de programación con el que se escribe toda la lógica del proyecto: inicialización de la ventana, carga de modelos, movimiento de cámara, animaciones, llamadas a OpenGL, etc.

Librerías gráficas y de utilidades.

- GLFW (manejo de ventanas, entradas y eventos)
- GLEW (extensiones de OpenGL)
- GLM (matemáticas para gráficos basado en OpenGL)
- ASSIMP (importación de modelos 3D)
- SOIL2 (carga de texturas)

Diccionario de funciones.

KeyCallback: Sirve para leer las entradas del Teclado.

MouseCallback: Lee entrada del movimiento de un mouse.

DoMovement: Reproduce el movimiento de las animaciones.

Diccionario de datos.

Dato	Descripción
WIDTH	Valor de ancho de la ventana
HEIGHT	Valor de alto de la ventana
SCREEN_WIDTH, SCREEN_HEIGHT	Valores que almacenan el alto y ancho de la ventana
camera	Representa la cámara en el espacio 3D
lastX, lastY	Almacenan las coordenadas del último posicionamiento del cursor del mouse
keys	Almacena el estado de las teclas
firstMouse	Indica si es la primera vez que se detecta el movimiento del mouse
doorOpen, doorCurrentAngle, doorTargetAngle, prevP	Variables para la animación de la puerta (rotación)
garageOpen, prevO, garageCurrentRotX, garageCurrentY, garageCurrentZ, garageTargetRotX, garageTargetY, garageTargetZ	Variables para la animación de la puerta de la cochera (rotación y traslación)
microOpen, microCurrentAngle, microTargetAngle, prevU	Variables para la animación de la puerta del mircoondas (abrir/cerrar)
mugAngle	Variable para la animación de la taza (girar)

toastJumping, toastTime, toastYOffset, toastRotX, prevT, toastJumpDuration, toastJumpHeight, PI, TWO_PI	Variables para la animación del pan tostado
deltatime	El tiempo que ha pasado entre el fotograma actual y el fotograma anterior.
lastFrame	Almacena el tiempo del último fotograma procesado
window	Puntero a la ventana creada por GLFW
model	Matriz de modelo utilizada para transformar los vértices en el espacio del objeto al espacio del mundo
lightingShader	Instancia de un shader utilizado para renderizar la iluminación de los objetos
lampShader	Instancia de un shader utilizado para renderizar la lámpara o fuente de luz
PisoSala, PisoCocina, Pisoizq, Pisoder, paredSalaIzq, paredSalaDer, paredSalaEsc, pareddetrasesc, paredfrontal, paredfrentelineas, paredtraseracasa, paredfachadaizquierda, paredEntradaCocina, marcopuertaAcocina, paredizqcocina, paredatrascocina, paredfrentecocina, ventana1, ventana2, ventana3, ventana4, ventana5, ventana6, ventana7, ventana8, ventana9, puertafueraizq, marcopuerta, puertaroja, salatecho, techo1, techo2, triangulo1, triangulo2, ventana10, ventana11, ventana12, ventana13, ventana14,	Modelos obj para representarlos en el espacio tridimensional.

ventana15, ventana16, ventana17, casitaarribader, casitaarribaizq, casitaarribagrande, casitaizq, techocasitaizq, puertacasitaizq, cochera, casitader, techocasitader, puertaAcocina, escalera, barandal, mesalamp, base, cabeza, sillon1, sillon2, mesarectangular, tapete, muebletv, tv, refri, cocinaintegralarriba1, cocinaintegralarriba2, cocinaintegralabajo1, cocinaintegralabajo2, cocinaintegralabajo3, basearriba1, basearriba2, basearriba3, stove, ventanainteriorcocina, microwave, platomicrowave, puertamicro, mug, tostadora, toast.	

Documentación del código.

El programa implementa una escena 3D renderizada con OpenGL usando GLFW para la creación de la ventana, GLEW para la carga de funciones modernas de OpenGL y GLM para las matemáticas 3D. Sobre esa base se cargan múltiples modelos (Model.h) que representan los elementos de la casa (pisos, paredes, ventanas, muebles) y se controlan con una cámara en primera persona (Camera.h).

Librerías y archivos

```
// 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"
```

- GLFW (<GLFW/glfw3.h>): Utilizada para la creación de la ventana, el manejo del contexto de OpenGL y la captura de eventos de entrada (teclado y ratón).
- GLEW (<GL/glew.h>): Empleada para cargar las funciones modernas de OpenGL y facilitar el uso de extensiones.
- GLM (<glm/...>): Biblioteca matemática para gráficos que proporciona tipos y operaciones para vectores, matrices y transformaciones 3D (traslación, rotación, proyección).
- stb_image.h y SOIL2: Utilizadas para la carga de texturas desde archivos de imagen.
- Shader.h, Camera.h y Model.h: Clases auxiliares que encapsulan, respectivamente, el manejo de shaders, el control de la cámara y la carga/dibujado de modelos 3D.

Declaración de variables globales

- **Dimensiones de ventana**

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

WIDTH y HEIGHT son las dimensiones lógicas de la ventana.

SCREEN_WIDTH y SCREEN_HEIGHT almacenan las dimensiones reales del framebuffer, necesarias para configurar el viewport.

- **Cámara**

```
// Camera
Camera  camera(glm::vec3(0.0f, 2.0f, 3.0f));
```

Se instancia una cámara en una posición ligeramente elevada respecto al piso, lo que permite visualizar la escena desde el inicio. La cámara se emplea para generar la matriz de vista en cada cuadro.

- **Gestión de entrada**

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

keys[] conserva el estado (presionada/no presionada) de cada tecla.

Las variables relacionadas con el ratón (firstMouse, lastX, lastY) permiten calcular los desplazamientos del cursor y, con ello, modificar la orientación de la cámara.

- **Iluminación**

```
// Positions of the point lights
glm::vec3 pointLightPositions[] = {
    glm::vec3(2.79f,  1.52f,  -0.45f),
    glm::vec3(2.3f, -3.3f, -4.0f),
    glm::vec3(-4.0f,  2.0f, -12.0f),
    glm::vec3(0.0f,  0.0f, -3.0f)
};

glm::vec3 LightP1;

glm::vec3 Light1 = glm::vec3(0);
```

pointLightPositions contiene las posiciones fijas de cuatro luces puntuales que se representarán en la escena.

Light1 y active permiten habilitar o deshabilitar dinámicamente el aporte de color de una de las luces mediante el teclado.

- **Control de animaciones**
  - doorOpen indica el estado lógico (abierta/cerrada).
  - doorTargetAngle es el ángulo objetivo (0° o 90°).
  - doorCurrentAngle es el ángulo que realmente se utiliza para dibujar en ese cuadro.
  - prevP evita que la animación se active varias veces por una misma pulsación.
  - Para la cochera se emplea el mismo esquema, pero con tres componentes (rotación y dos traslaciones), ya que la animación es más compleja.
- **Control temporal**

deltaTime y lastFrame permiten calcular el tiempo transcurrido entre un cuadro y el siguiente. De este modo, el movimiento de la cámara y las animaciones se vuelven independientes de la velocidad de fotogramas.

- **Funcion main()**

La función main() concentra la fase de inicialización, la carga de recursos y el bucle principal de renderizado.

1. Inicialización de GLFW y creación de ventana:
    - Se inicializa la biblioteca GLFW.
    - Se especifica que se trabajará con OpenGL 3.3 en perfil core.
    - Se crea la ventana principal. En caso de falla, el programa se detiene y libera los recursos.
    - Se asocia la ventana al contexto actual mediante glfwMakeContextCurrent(window) y se registran los callbacks de teclado y ratón. También se deshabilita el cursor para permitir un control tipo “primera persona”.
2. Inicialización de GLEW y configuración de OpenGL:
    - GLEW se inicializa una vez que existe un contexto válido.
    - Se configura el viewport con las dimensiones del framebuffer.
    - Se habilita la prueba de profundidad para que el renderizado respete la distancia de los objetos.
3. Creación de shaders y carga de modelos:
    - Se instancian dos shaders:

        lightingShader: responsable del renderizado de los objetos de la escena con iluminación.

        lampShader: empleado para dibujar los indicadores de las luces puntuales.
    - Posteriormente, se cargan todos los modelos 3D necesarios. Esta carga se hace fuera del bucle de renderizado para evitar sobrecostes en tiempo de ejecución. Cada instancia de Model corresponde a un elemento de la casa o de su mobiliario.

```
Model Pisoizq((char*)"Models/Objetos/pisolateralizq.obj");
Model Pisoder((char*)"Models/Objetos/pisolateralder.obj");
Model paredSalaIzq((char*)"Models/Objetos/paredSalaIzq.obj");
Model paredSalaDer((char*)"Models/Objetos/paredSalaDer.obj");
Model paredSalaEsc((char*)"Models/Objetos/paredSalaEsc.obj");
Model pareddetrasesc((char*)"Models/Objetos/pareddetrasesc.obj");
Model paredfrontal((char*)"Models/Objetos/paredfrontal.obj");
Model paredfrentelineas((char*)"Models/Objetos/paredfrentelineas.obj");
Model paredtraseracasa((char*)"Models/Objetos/paredtraseracasa.obj");
Model paredfachadaizquierda((char*)"Models/Objetos/paredfachadaizquierda.obj");
Model ventana1((char*)"Models/Objetos/ventana1.obj");
Model ventana2((char*)"Models/Objetos/ventana2.obj");
Model ventana3((char*)"Models/Objetos/ventana3.obj");
Model ventana4((char*)"Models/Objetos/ventana4.obj");
Model ventana5((char*)"Models/Objetos/ventana5.obj");
Model ventana6((char*)"Models/Objetos/ventana6.obj");
Model ventana7((char*)"Models/Objetos/ventana7.obj");
Model ventana8((char*)"Models/Objetos/ventana8.obj");
Model ventana9((char*)"Models/Objetos/ventana9.obj");
Model puertafueraizq((char*)"Models/Objetos/puertafueraizq.obj");
Model marcopuerta((char*)"Models/Objetos/marcopuerta.obj");
Model puertaroja((char*)"Models/Objetos/puertaroja.obj");
Model salatecho((char*)"Models/Objetos/salatecho.obj");
Model techo1((char*)"Models/Objetos/techo1.obj");
Model techo2((char*)"Models/Objetos/techo2.obj");
Model triangulo1((char*)"Models/Objetos/triangulo1.obj");
Model triangulo2((char*)"Models/Objetos/triangulo2.obj");
Model ventana10((char*)"Models/Objetos/ventana10.obj");
```

4. Configuración de buffers (VAO, VBO, EBO)

   Se crea un VAO y los buffers asociados para almacenar los datos de un cubo. Este cubo se reutiliza para dibujar las lámparas (pequeños cubos que marcan las posiciones de las point lights). Se definen los atributos de vértice: posición, normal y coordenadas de textura.

5. Carga de texturas

   Se crean y configuran dos texturas (difusa y especular) mediante stbi_load. Se establecen los parámetros de envoltura y filtrado y se asignan las texturas a las unidades que espera el shader.

6. Creación de la matriz de proyección

   Esta matriz permanece constante durante la ejecución y se envía al shader en cada cuadro.

```
glm::mat4 projection = glm::perspective(camera.GetZoom(), (GLfloat)SCREEN_WIDTH / (GLfloat)SCREEN_HEIGHT,
```

- **Bucle principal de renderizado**

El bucle se ejecuta mientras la ventana no reciba la orden de cierre:

1. Cálculo de tiempo por cuadro

Se determina deltaTime a partir del tiempo actual y del tiempo del cuadro anterior.

2. Procesamiento de eventos e interacción
   - glfwPollEvents() procesa los eventos de entrada capturados por GLFW.
   - DoMovement() actualiza la posición de la cámara y el estado de las animaciones con base en las teclas almacenadas en keys[].
3. Limpieza del buffer de dibujo

   Se limpia el color y el z-buffer para comenzar el nuevo cuadro.
4. Activación del shader de iluminación y envío de parámetros

   Se activa lightingShader y se envían:
   - la posición de la cámara,
   - los parámetros de la luz direccional,
   - los parámetros de las cuatro luces puntuales,
   - el reflector ligado a la cámara,
   - las propiedades del material (brillo, texturas).

   - Luz direccional:

     Una luz direccional simula una fuente de luz “infinitamente lejana”, como el sol. Debido a esa distancia “infinita”, los rayos de luz llegan paralelos a toda la escena. Por eso no se especifica posición, sino una dirección

     Esa luz afecta a todos los objetos por igual, sin perder intensidad con la distancia.

```
// Directional light
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.35f, 0.35f, 0.35f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.7f, 0.7f, 0.7f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.7f, 0.7f, 0.7f);
```

Los tres componentes que se envía (ambient, diffuse y specular) son el modelo de iluminación clásico:

- Ambient: Luz ambiental o de fondo. Asegura que incluso las partes que no reciben luz directa no queden totalmente negras.

- Diffuse: Luz difusa. Depende del ángulo entre la luz y la normal de la superficie.
- Specular: Brillo especular. Da los reflejos más intensos.

- Pointlights

Igual que con la luz direccional, cada luz puntual puede tener:

- ambient: nivel mínimo de luz que aporta.
- diffuse: iluminación principal.
- specular: brillo.

5. Construcción de las matrices de vista y proyección

   La matriz de vista se obtiene a partir del objeto camera. Ambas matrices se envían al shader.

6. Dibujado de los modelos

   Para cada modelo se realiza:

   - Inicialización de la matriz de modelo (glm::mat4(1)),
   - Aplicación de las transformaciones necesarias (en el caso de la puerta y la cochera, se aplican las transformaciones animadas),
   - Envío de la matriz al shader.
   - Llamado a Draw(...).

   En determinados objetos se activa el blending para permitir transparencias y luego se desactiva nuevamente.

- **Modelos con animación**

Objetos con transformaciones dinámicas:

- Puerta: Rotacion en eje Y para simular el movimiento de la puerta
- Cochera: Rotacion en el eje X y traslacion en eje Y y Z para simular el movimiento de la puerta de una cochera
- Lampara: Transparencia para simular el uso de una lampara con el pointlight.

- Microondas y taza: Rotación de la puerta para simular abrir y cerrar el microondas, y rotación de la taza dentro del microondas.
- Pan de la tostadora: Brinco y giro del pan saliendo de la tostadora.

```
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-0.52f, 1.062f, 3.9f));
model = glm::rotate(model, doorCurrentAngle, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
puertaroja.Draw(lightingShader);
```

```
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-6.284f, 0.923f, 0.253f));
// movimiento extra de la animación (subir y meter hacia -Z)
model = glm::translate(model, glm::vec3(0.0f, garageCurrentY, garageCurrentZ));
// rotación de la puerta de la cochera en X
model = glm::rotate(model, garageCurrentRotX, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
cochera.Draw(lightingShader);
```

```
glEnable(GL_BLEND);//Avtiva la funcionalidad para trabajar el canal alfa
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0,1.0,0.0,0.75);
cabeza.Draw(lightingShader);
glDisable(GL_BLEND);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0, 1.0, 1.0, 1.0);
glBindVertexArray(0);
```

```
// Puerta del microondas (abre/cierra con U)
model = glm::mat4(1);
// Trasladas la puerta a su posición en el mundo
model = glm::translate(model, glm::vec3(-1.665f, 0.917f, 3.273f));
// La rotas 90° alrededor del eje Y cuando está abierta
model = glm::rotate(model, microCurrentAngle, glm::vec3(0.0f, 1.0f, 0.0f));

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
puertamicro.Draw(lightingShader);

// Taza: gira usando mugAngle
model = glm::mat4(1);

// Si quieres posicionarla en algún lugar, puedes hacer algo así:
model = glm::translate(model, glm::vec3(-1.88f, 0.845f, 3.436f));

model = glm::rotate(model, mugAngle, glm::vec3(0.0f, 1.0f, 0.0f));

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
mug.Draw(lightingShader);
```

```
model = glm::mat4(1);
// posición base + desplazamiento del salto en Y
model = glm::translate(model, glm::vec3(-4.551f, 0.875f + toastYOffset, 2.317f));
// giro de 0 a 360° sobre el eje X mientras está en el aire
model = glm::rotate(model, toastRotX, glm::vec3(1.0f, 0.0f, 0.0f));

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
toast.Draw(lightingShader);
```

- **Dibujado de las luces visibles**

Se cambia al shader de lámpara (lampShader), se vuelven a enviar las matrices de vista y proyección y se dibuja un cubo pequeño en cada posición almacenada en pointLightPositions.

- **Intercambio de buffers**

Finalmente, se invoca glfwSwapBuffers(window) para mostrar en pantalla el contenido del cuadro recién dibujado.

Al terminar el bucle se eliminan los VAO/VBO/EBO creados y se llama a glfwTerminate() para liberar los recursos de GLFW.

- **Función DoMovement()**

Esta función se ejecuta en cada cuadro y su responsabilidad es traducir el estado del teclado (previamente capturado en el callback) en acciones sobre la escena:

1. Movimiento de cámara: Si las teclas W, A, S, D o las flechas están activas, se invoca a camera.ProcessKeyboard(...), pasando también deltaTime para que la velocidad de desplazamiento sea uniforme.

```
void DoMovement()
{

    // Camera controls
    if (keys[GLFW_KEY_W] || keys[GLFW_KEY_UP])
    {
        camera.ProcessKeyboard(FORWARD, deltaTime);
    }

    if (keys[GLFW_KEY_S] || keys[GLFW_KEY_DOWN])
    {
        camera.ProcessKeyboard(BACKWARD, deltaTime);
    }

    if (keys[GLFW_KEY_A] || keys[GLFW_KEY_LEFT])
    {
        camera.ProcessKeyboard(LEFT, deltaTime);
    }

    if (keys[GLFW_KEY_D] || keys[GLFW_KEY_RIGHT])
    {
        camera.ProcessKeyboard(RIGHT, deltaTime);
    }
```

2. Apertura/cierre de puerta (tecla P): se detecta la pulsación única (usando prevP) y se alterna el estado lógico de la puerta. Según el estado, se fija el ángulo objetivo en 0° o 90°. Posteriormente, se interpola del ángulo actual al ángulo objetivo mediante un factor dependiente del tiempo, lo que genera un movimiento suave.

```
// Toggle puerta con P
if (keys[GLFW_KEY_P] && !prevP) {
    doorOpen = !doorOpen;
    doorTargetAngle = doorOpen ? glm::radians(90.0f) : 0.0f;
}
prevP = keys[GLFW_KEY_P];

// Suavizado hacia el ángulo objetivo
float speed = 4.0f;
doorCurrentAngle += (doorTargetAngle - doorCurrentAngle) * speed * deltaTime;
```

3. Apertura/cierre de cochera (tecla O): se aplica el mismo patrón que en la puerta, pero actualizando tres valores (rotación en X y desplazamientos en Y y Z) para simular un movimiento más realista. También aquí se realiza una interpolación progresiva hacia los valores objetivo.

```
// ====== PUERTA DE COCHERA ======
if (keys[GLFW_KEY_O] && !prevO) {
    garageOpen = !garageOpen;

    if (garageOpen) {
        // abrir: rotar en X a -90°, subir un poco en Y y meter un poco en -Z
        garageTargetRotX = glm::radians(-80.0f);
        garageTargetY = 0.85;
        garageTargetZ = -0.206f;
    }
    else {
        // cerrar: todo de regreso
        garageTargetRotX = 0.0f;
        garageTargetY = 0.0f;
        garageTargetZ = 0.0f;
    }
}
prevO = keys[GLFW_KEY_O];

// suavizado de la cochera
{
    float gSpeed = 2.0f;
    garageCurrentRotX += (garageTargetRotX - garageCurrentRotX) * gSpeed * deltaTime;
    garageCurrentY += (garageTargetY - garageCurrentY) * gSpeed * deltaTime;
    garageCurrentZ += (garageTargetZ - garageCurrentZ) * gSpeed * deltaTime;
}
```

4. Apertura/cierre de puerta de microondas (tecla U): Se sigue el mismo patrón de detección de pulsación única utilizando la variable prevU. Cuando se detecta el flanco de la tecla U, se alterna el estado lógico microOpen y, en función de este estado, se establece microTargetAngle en 0° (cerrado) o −90° (abierto). De forma análoga a la puerta principal, no se aplica el cambio de ángulo de manera instantánea, sino que se interpola desde microCurrentAngle hacia microTargetAngle con un factor de suavizado dependiente de deltaTime, lo que produce una animación fluida de apertura y cierre alrededor del eje Y.

```
// Toggle puerta microondas con U
if (keys[GLFW_KEY_U] && !prevU) {
    microOpen = !microOpen;
    microTargetAngle = microOpen ? glm::radians(-90.0f) : 0.0f;
}
prevU = keys[GLFW_KEY_U];

// Suavizado de la puerta del microondas
{
    float microSpeed = 4.0f;  // qué tan rápido llega al objetivo
    microCurrentAngle += (microTargetAngle - microCurrentAngle) * microSpeed * deltaTime;
}
```

5. Rotación continua de la taza mientras el microondas está abierto: La función también controla la animación de la taza (mug) mediante la variable mugAngle. Siempre que microOpen es verdadero, en cada cuadro se incrementa mugAngle a partir de una velocidad angular fija (90°/s expresados en radianes) multiplicada por deltaTime. Además, se normaliza el valor del ángulo restándole una vuelta completa (2π) cuando se excede dicho valor, evitando acumulaciones numéricas innecesarias. Este ángulo actualizado se utiliza más adelante en la transformación del modelo de la taza, generando un giro continuo mientras la puerta del microondas permanece abierta.

```
// Rotación continua de la taza mientras el micro está "abierto"
if (microOpen) {
    float mugSpeed = glm::radians(90.0f); // 90° por segundo
    mugAngle += mugSpeed * deltaTime;

    // opcional: para que el valor no crezca infinito
    if (mugAngle > glm::two_pi<float>())
        mugAngle -= glm::two_pi<float>();
}
```

6. Animación del pan tostado (tecla T): salto y giro: Para el pan tostado se implementa una animación discreta que se dispara al presionar la tecla T. Mediante la variable prevT se detecta el flanco de la tecla y, si no hay una animación en curso (!toastJumping), se inicia el salto reiniciando el tiempo acumulado toastTime y activando la bandera toastJumping. Mientras esta bandera está activa, en cada cuadro se incrementa toastTime, se calcula un parámetro normalizado s en el intervalo [0,1] y, a partir de él, se actualizan toastYOffset (desplazamiento vertical)

usando una función seno para simular una trayectoria de subida y bajada suave, y toastRotX para aplicar un giro completo de 360° sobre el eje X durante todo el salto. Al finalizar (cuando s llega a 1), la animación se detiene, se desactiva toastJumping y los valores se restauran a su pose original, garantizando que el pan regrese a su posición de reposo.

```
// ====== PAN TOSTADO: salto + giro con T ======
if (keys[GLFW_KEY_T] && !prevT && !toastJumping) {
    // inicia la animación solo en el flanco de la tecla
    toastJumping = true;
    toastTime = 0.0f;
}
prevT = keys[GLFW_KEY_T];
if (toastJumping) {
    toastTime += deltaTime;
    float s = toastTime / toastJumpDuration; // [0,1]
    if (s >= 1.0f) {
        s = 1.0f;
        toastJumping = false;        // termina la animación
    }
    // movimiento vertical tipo "parábola": sube y baja
    toastYOffset = toastJumpHeight * sinf(PI * s);
    // giro completo de 360° sobre X durante el salto
    toastRotX = TWO_PI * s;
}
else {
    // cuando no hay animación, vuelve a su pose original
    toastYOffset = 0.0f;
    toastRotX = 0.0f;
}
```

- **Callbacks de entrada**

1. KeyCallback(...)
    - Si se presiona la tecla Escape, se solicita el cierre de la ventana.
    - Para cualquier otra tecla, se actualiza el arreglo keys[] con el estado PRESIONADO o LIBERADO, lo que permite a DoMovement() conocer qué teclas están activas sin depender del evento puntual.

- La tecla Espacio conmuta el estado de una de las luces (active y Light1), lo que permite activar o desactivar el aporte de color de la primera luz puntual.

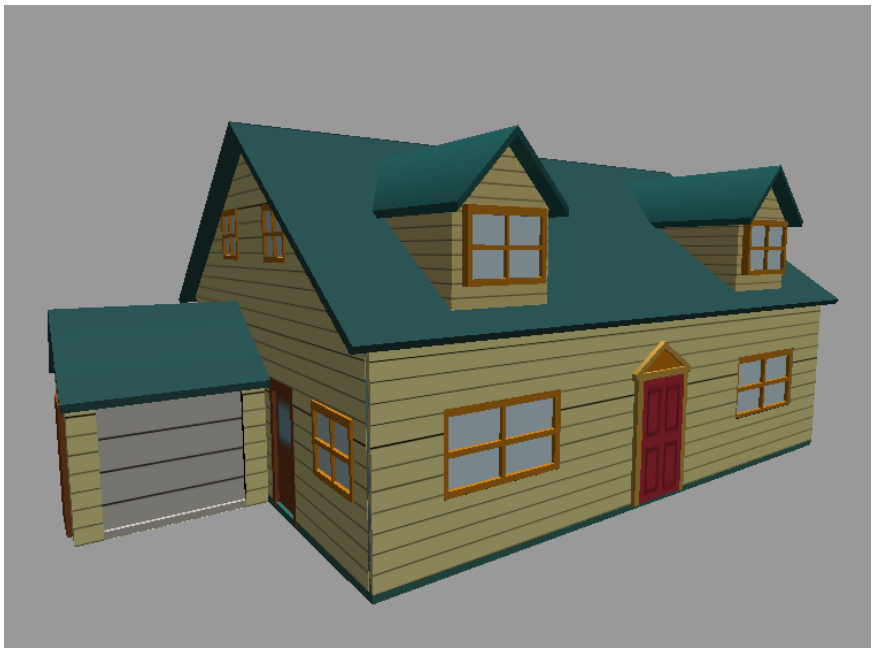
```
void KeyCallback(GLFWwindow *window, int key, int scancode, int action, int mode)
{
    if (GLFW_KEY_ESCAPE == key && GLFW_PRESS == action)
    {
        glfwSetWindowShouldClose(window, GL_TRUE);
    }
    if (key >= 0 && key < 1024)
    {
        if (action == GLFW_PRESS)
        {
            keys[key] = true;
        }
        else if (action == GLFW_RELEASE)
        {
            keys[key] = false;
        }
    }
    if (keys[GLFW_KEY_SPACE])
    {
        active = !active;
        if (active)
            Light1 = glm::vec3(1.0f, 1.0f, 0.0f);
        else
            Light1 = glm::vec3(0.0f, 0.0f, 0.0f);
    }
}
```

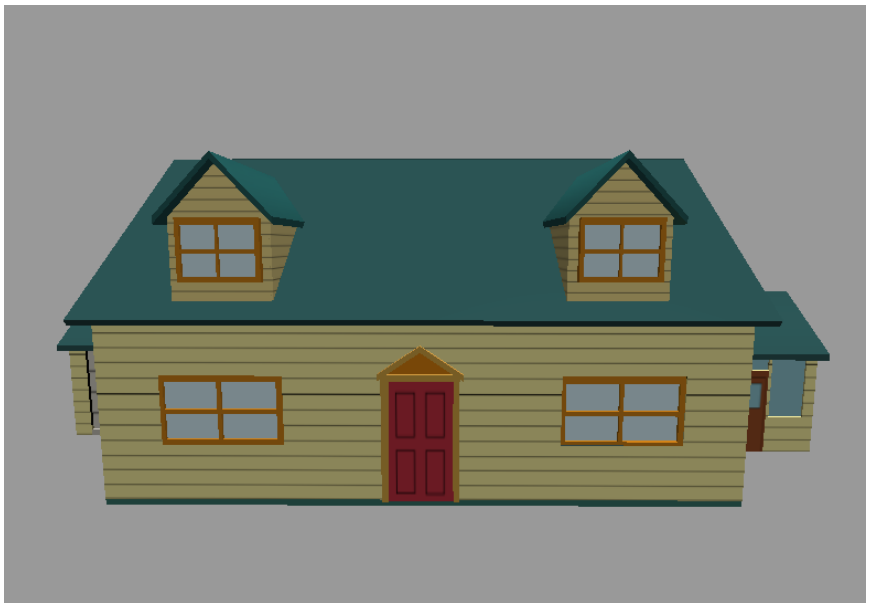
2. MouseCallback(...)
   - En la primera llamada se almacenan las coordenadas iniciales del cursor.
   - En las siguientes llamadas se calcula el desplazamiento horizontal y vertical del ratón y se pasa esa información a camera.ProcessMouseMovement(...). De este modo se actualiza la orientación de la cámara.

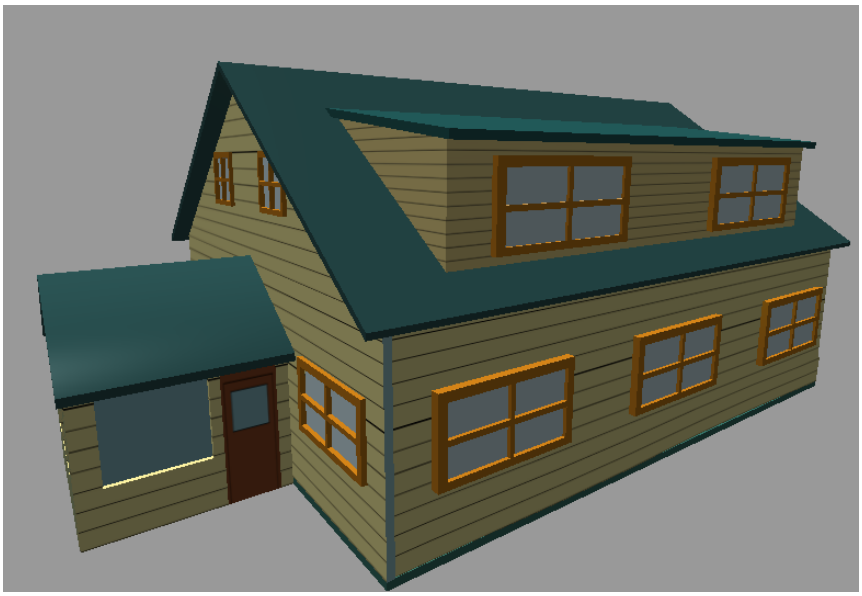
```
void MouseCallback(GLFWwindow *window, double xPos, double yPos)
{

    if (firstMouse)
    {
        lastX = xPos;
        lastY = yPos;
        firstMouse = false;
    }

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

    lastX = xPos;
    lastY = yPos;

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

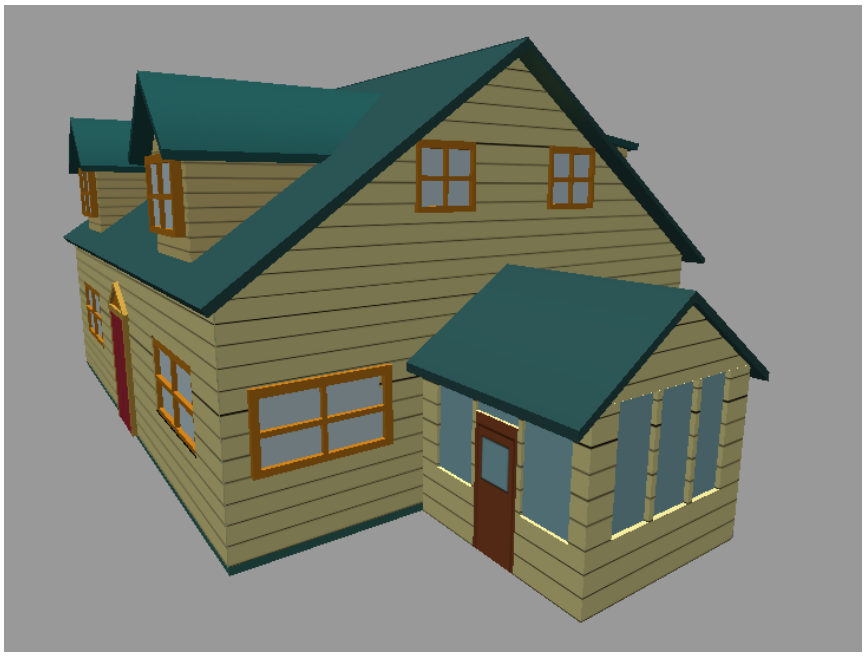
Resultados

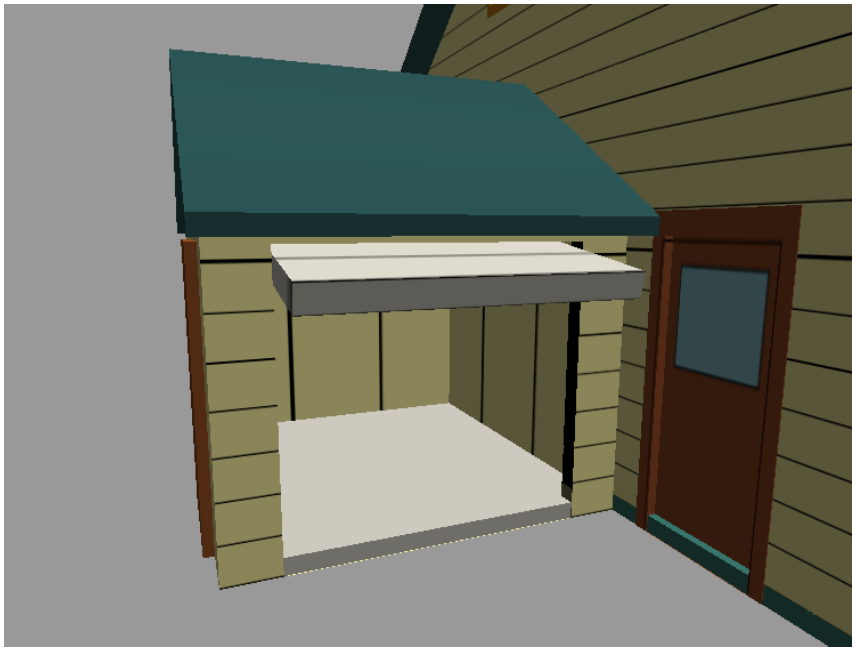
A continuación, se presentarán algunas imágenes de los modelos cargados en OpenGL.

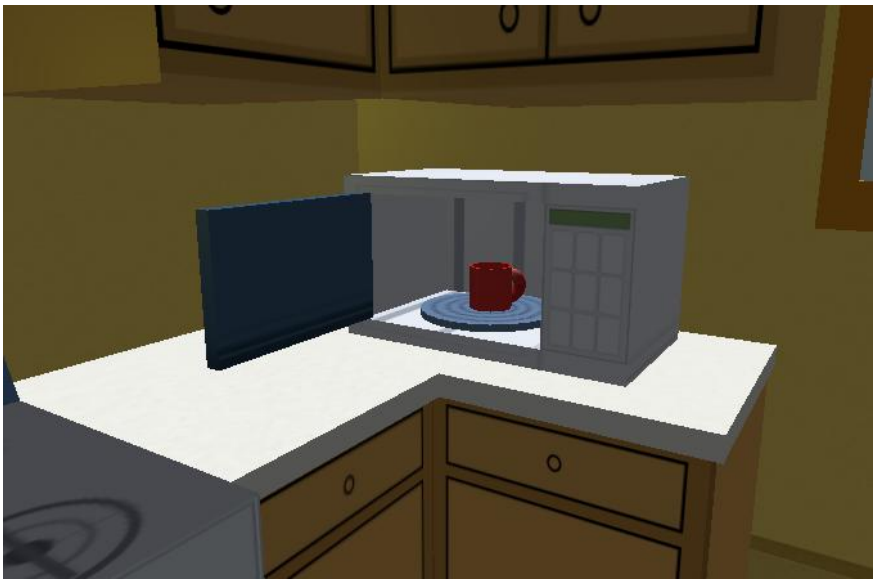


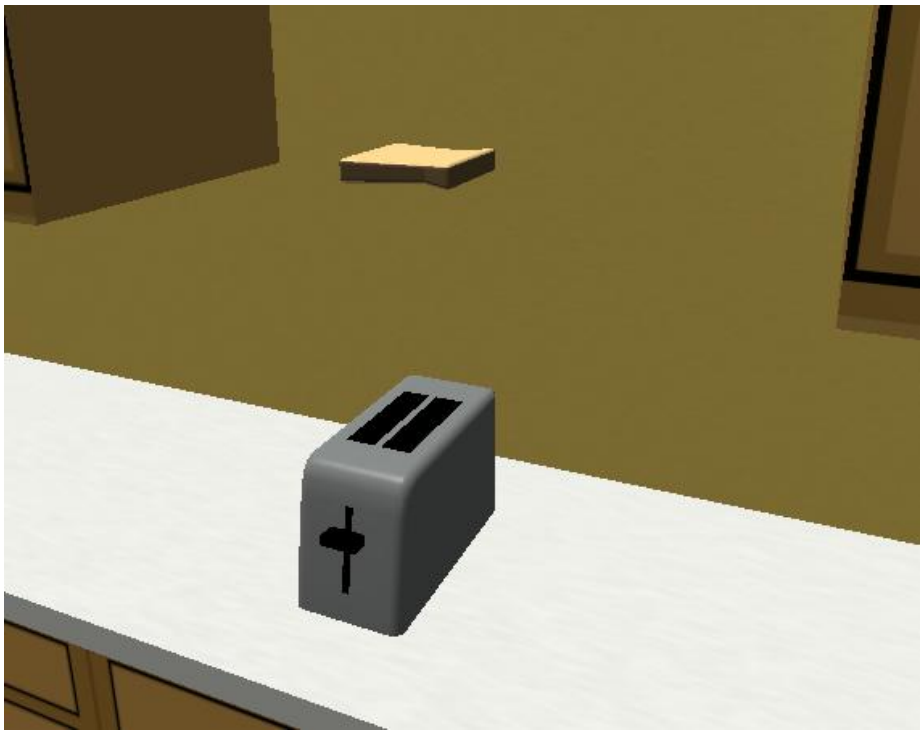


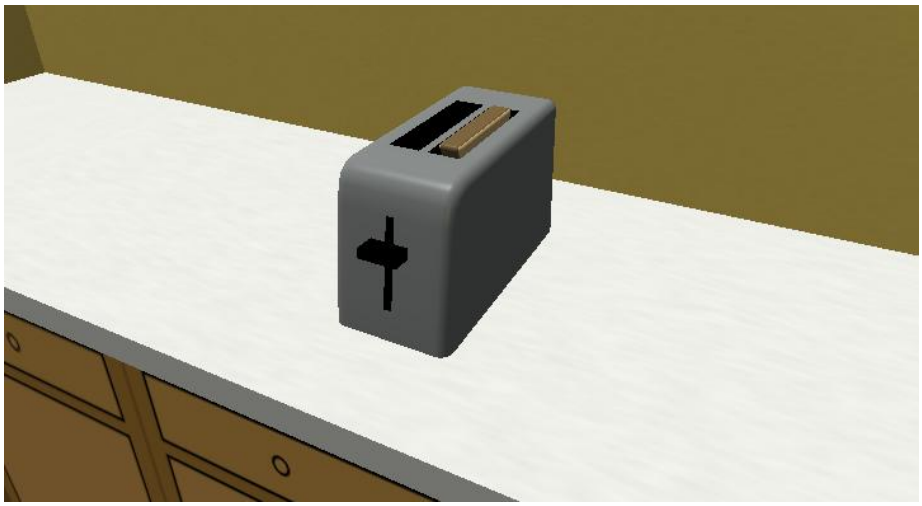












Análisis de costos del proyecto.

- **Costos directos:**

Recursos	Costos	Detalles
Software de efectos visuales y animación 3D: Autodesk Maya	$3,185.00 MXN (mensual)	El software se puede adquirir de forma gratuita con un correo institucional de estudiante.
Laptop	$18,000.00 MXN	Una maquina con los requerimientos suficientes de buena calidad para ejecutar los programas y tener un buen desempeño grafico tiene en promedio este precio.
Conocimiento	$3,250.00 MXN	Se considera el salario mínimo por hora, se multiplica por las horas dedicadas a las clases y horas de laboratorio para adquirir el conocimiento necesario para realizar este proyecto acabo.
Tiempo trabajado	50 horas ($34.00 MXN p/hora)	Considerando el día justo en que se empezó a trabajar con los modelos y la casa, las semanas que se requirieron para desarrollar el proyecto, se contabilizaron aproximadamente 50 horas trabajas.

Considerando que la laptop no fue usada exclusivamente para el desarrollo del proyecto, se determinara el costo solo por los 4 meses usados directamente para la materia, laboratorio y desarrollo del proyecto.

En promedio, la vida útil de una laptop suele ser de 4 años:

$$\frac{4\ meses}{48\ meses}\ x\ \$18{,}000.00\ \text{MXN} = \$1{,}500.00\ \text{MXN}$$

$$50\ x\ \$34.00\ MXN = \$1{,}700.00\ MXN$$

$$\$1{,}700.00\ MXN + \$1{,}500.00\ \text{MXN} + \$3{,}250.00\ \text{MXN} = \$6{,}450.00\ \text{MXN}$$

Total: $6,450.00 MXN

- **Costos indirectos:**

Recursos	Costos	Detalles
Internet/electricidad	$700.00 MXN	Se estima un gasto mensual aproximado de $700.00 MXN correspondiente al servicio de internet y al consumo de energía eléctrica necesarios para el uso continuo de la laptop durante el modelado, renderizado y documentación del proyecto. El monto considera la parte proporcional de estos servicios destinada al desarrollo del proyecto en el periodo trabajado (2 meses).

$700.00 MXN x 2 = $1,400.00 MXN

Total: $1,400.00 MXN

Costo total de desarrollo: $7,850.00 MXN

- **Precio por el proyecto:**

Para determinar el precio de venta del proyecto, primero se calculó el costo total de desarrollo, considerando los recursos directos (software, equipo de cómputo, tiempo de trabajo y conocimiento especializado) e indirectos (servicios como internet y electricidad), obteniendo un monto de $7,850.00 MXN. Sobre esta base se definió un margen de ganancia del 30%, que representa $2,355.00 MXN, con el objetivo de compensar las horas invertidas en el modelado, animación, pruebas y documentación, así como recuperar la inversión inicial en infraestructura y servicios necesarios para la ejecución del proyecto. En consecuencia, el precio de venta propuesto para el proyecto

asciende a $10,205.00 MXN (se redondea a $10,200.00 MXN), cantidad que refleja tanto los costos reales de producción como una ganancia razonable por el trabajo realizado.

Precio del proyecto: $10,200.00 MXN

- **Justificación.**

La determinación del costo del proyecto se justifica a partir de una estimación sistemática de los recursos necesarios para su desarrollo. En primer lugar, se identificaron los costos directos, que incluyen el equipo de cómputo, el conocimiento técnico adquirido y el tiempo efectivamente invertido en el modelado, animación y documentación. Aunque se menciona el software Autodesk Maya como herramienta principal y se indica su costo comercial mensual, este monto no se incorporó al cálculo económico, ya que para el proyecto se utilizó una licencia educativa gratuita asociada al correo institucional del estudiante. Por ello, el programa se registra como recurso indispensable, pero sin representar un desembolso monetario real dentro del análisis. En cambio, sí se considera el costo de la laptop de forma amortizada, suponiendo una vida útil promedio de cuatro años y contabilizando únicamente los cuatro meses en que se empleó de manera directa en la asignatura y en el proyecto, lo que equivale a $1,500.00 MXN. A esto se suma el valor del conocimiento (calculado con base en el salario mínimo por hora y las horas de clases y laboratorio necesarias para adquirir las competencias requeridas) y las 50 horas de trabajo específico sobre el proyecto, valuadas a $34.00 MXN por hora. La suma de estos elementos da como resultado un costo directo de $6,450.00 MXN, que refleja de manera realista los recursos esenciales empleados.

En segundo término, se integran los costos indirectos, representados por los servicios de internet y electricidad. Se estima un gasto mensual promedio de $700.00 MXN, proyectado para dos meses de trabajo intensivo, lo que arroja un total de $1,400.00 MXN. Este monto corresponde a la parte proporcional del consumo necesario para mantener en operación el equipo de cómputo durante el modelado, renderizado y elaboración de la documentación técnica. Al sumar costos directos e indirectos se

obtiene un costo total de desarrollo de $7,850.00 MXN, que sirve como base para la fijación del precio final.

Finalmente, el precio de venta de $10,200.00 MXN se justifica mediante la aplicación de un margen de ganancia del 30 % sobre el costo de desarrollo. Este margen busca compensar el esfuerzo invertido en las distintas etapas del proyecto (planeación, modelado, animación, pruebas y documentación), así como recuperar la inversión en equipo y servicios utilizados, aun cuando el software principal haya sido accesible sin costo gracias a la licencia de estudiante. De esta manera, el precio propuesto no es arbitrario, sino el resultado de un análisis de costos estructurado que equilibra la viabilidad económica del proyecto con una retribución justa por el trabajo profesional realizado.

Estado del arte del proyecto.

En la actualidad, los recorridos virtuales se han consolidado como una herramienta clave en distintos sectores que requieren visualizar y explorar espacios tridimensionales sin necesidad de presencia física. En arquitectura y bienes raíces, por ejemplo, se emplean para mostrar viviendas, oficinas o proyectos aún en fase de diseño, permitiendo al usuario desplazarse libremente por el entorno, cambiar el punto de vista y observar detalles de distribución, iluminación y mobiliario. En museos y patrimonio cultural se utilizan tours inmersivos para hacer accesibles exposiciones o sitios históricos a distancia, mientras que en la industria del entretenimiento y los videojuegos se integran entornos interactivos que combinan narrativa, exploración y elementos lúdicos. El avance de tecnologías como motores gráficos en tiempo real, WebGL y dispositivos de realidad virtual ha impulsado recorridos cada vez más realistas, con mejor calidad de texturas, iluminación dinámica y una navegación más intuitiva.

Dentro del ámbito académico, los recorridos virtuales se utilizan como laboratorio didáctico para comprender conceptos de gráficos por computadora, interacción humano–

computadora y diseño de interfaces. Al construir un entorno recorrible, el estudiante debe aplicar de manera integrada modelado 3D, mapeo UV, texturizado, configuración de materiales, iluminación y programación de cámara, lo que convierte al proyecto en un caso práctico del pipeline gráfico completo. El proyecto desarrollado en este manual se inserta en esta tendencia: la recreación de la fachada, sala y cocina de la casa de Family Guy en Maya y su posterior exportación a OpenGL se plantea como un recorrido virtual controlado por el usuario, donde es posible desplazarse por la escena, observar los objetos desde distintos ángulos y apreciar el comportamiento de las luces y animaciones. De esta forma, se replica a pequeña escala la lógica de los recorridos virtuales profesionales, pero con un enfoque formativo que prioriza la comprensión técnica del proceso y la experimentación con los elementos principales de un entorno 3D interactivo.

Conclusiones

Comenzando el curso la idea de modelar y texturizar me parecía algo sumamente complicado y complejo, pero al realizar este proyecto, comprender e integrar todas las etapas del pipeline grafico me permitió entender y vivir cada parte de este proceso. Modelar los objetos tuvieron su complejidad, dependiendo de que tan perfecto uno lo quería, el detalle a la cosa más mínima hacía que el tiempo modelando fuera mucho, también el crear desde cero las imágenes para texturizar los objetos fueron un reto, ya que debía ver la forma de que una imagen tenga todos los colores del objeto, para este último me apoye de la inteligencia artificial para crear las imágenes.

La iluminación también llevo su tiempo, determinar los valores de las componentes para que al reflejarse la luz en la casa y en los muebles, estos se vean lo más parecido que en el software Maya, fue un proceso tardado.

En la parte del código me lleve el mayor reto al exportar los objetos, ya que en algunas cosas OpenGL no me mostraba las cosas como se veían en Maya, por lo que debía analizar si el

texturizado se llevó de buena manera o me falto un paso aplicar. Sin duda, algo clave en la realización de este proyecto fue la paciencia y constancia.

En conclusión, este proyecto me permitió reforzar y aprender nuevos conocimientos sobre gráficos 3D, la organización, planeación y metodologías de software para poder elaborar proyectos de una forma más eficiente.