Universidad Nacional Autónoma de México

Facultad de Ingeniería

Materia:

Laboratorio de computación grafica e interacción humano-computadora

Alumno:

Josué Vázquez Cárdenas

Profesor:

Ing. Edén Espinoza Urzúa

Semestre:

2026-1

Grupo:

04

Entregable:

Proyecto Final

Índice

Objetivo

El alumno deberá aplicar y demostrar los conocimientos adquiridos durante el curso mediante la creación de una recreación 3D en OpenGL.

Objetivos específicos:

Recreación de un escenario: Modelar y renderizar con las herramientas proporcionadas la casa de la caricatura Billy and Mandy, incluyendo 7 objetos, con sus respectivas texturas y mapas de normales.

Implementación de iluminación: Desarrollar un sistema de iluminación Phong que cumpla con los diferentes tipos de fuente de luz para dar realismo a la escena.

Desarrollo de animaciones: Programar comportamientos dinámicos en los objetos divididos en: Animaciones simples y animaciones complejas.

Independencia del software: Generar un ejecutable final que sea funcional y portable, gestionando las dependencias de las librerías dinámicas para la ejecución en otro entorno.

Diagrama de Gantt

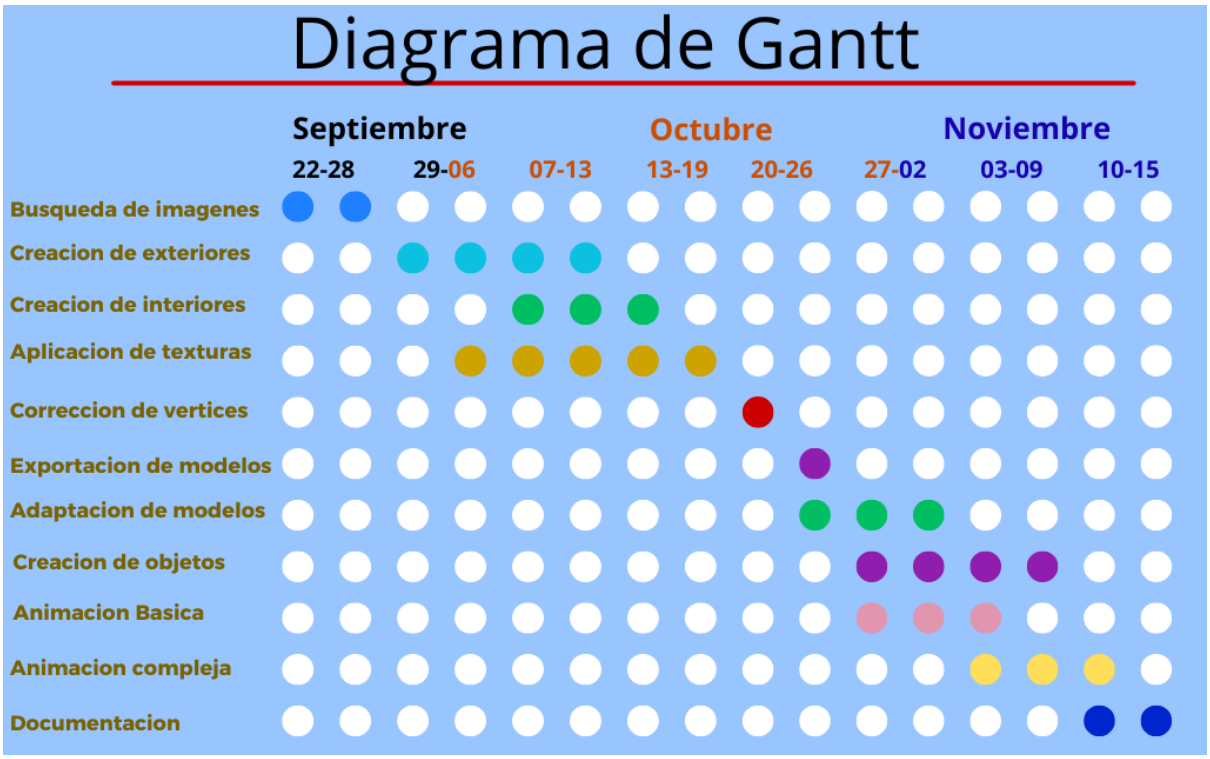


Diagrama de Gantt

Este proyecto tuvo un plazo del 22 de septiembre al 15 de noviembre de 2025 en el cual se desarrollaron diferentes tareas las cuales fueron:

Búsqueda de imágenes (22-28 de septiembre): La fase inicial de este proyecto fue la búsqueda de modelos a realizar se eligió la caricatura de Billy y Mandy como inspiración para la ejecución de este proyecto. En esta búsqueda se obtuvieron las imágenes necesarias para modelar la fachada, sala, baño y objetos.

Creación de exteriores (29 de septiembre-13 de octubre): En este lapso se optó por utilizar blender como motor de modelado del proyecto, se realizaron diferentes pruebas para entender el funcionamiento de blender y se comenzó con el modelado exterior de la casa donde incluía, árbol, ventanas, puertas, chimenea y paredes.

Creación de interiores (06-13 de octubre): Continuando con el modelado en blender se modelaron los interiores de la casa, en donde se realizaron los marcos de las puertas, la separación por cuartos, los pisos y techos de cada uno de ellos.

Aplicación de texturas (06-19 de octubre): El proceso de texturizado se inició desde la finalización del modelado de los exteriores para facilitar la visualización del modelo, las texturas fueron diseñadas en Paint, Gemini y extraídas de internet.

Corrección en modelado (20-23 de octubre): Una vez finalizado el modelado de la casa, se revisaron fallos en los cuales existía duplicidad de vértices, error de ubicación o deformaciones en los objetos. Además, triangular cada una de las caras de los objetos correcciones de escalas y rotaciones.

Exportación de modelos (23-26 de octubre): Los modelos se exportaron desde blender utilizando un formato .obj, con el objetivo de visualizarlo en OpenGL.

Adaptación de modelos (23 de octubre-02 de noviembre): Los modelos realizados, se adaptaron para que se pudieran visualizar, de forma correcta en textura, ubicación y posición.

Creación de objetos (27 de octubre- 09 de noviembre): Los objetos realizados en OpenGL (arbustos, televisión, lampara, sofá, taza de baño, bañera y maceta con planta) tuvieron un tiempo de realización más longevo ya que durante el proceso se realizaron las animaciones simples con los objetos que se solicitaban.

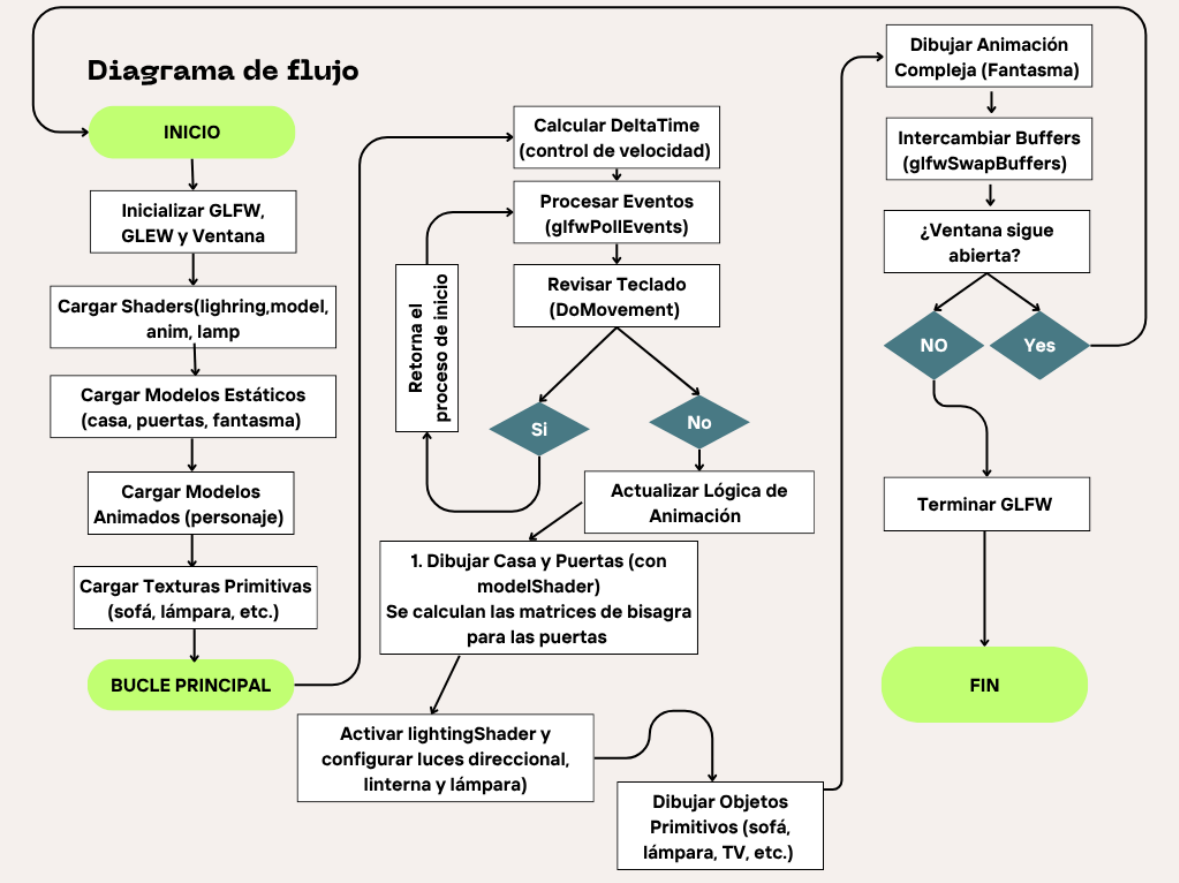
Animación básica (27 de octubre-03 de noviembre): Las animaciones básicas (movimiento de lampara, encendido de lampara y apertura de puertas) se realizaron con la creación de cada uno de los objetos implicados.

Animación compleja (03-10 de noviembre): Las animaciones complejas (Billy caminando, espectro saliendo de televisión) fueron más densas debido a que los

modelos de Billy y el espectro se extrajeron de internet y se cargaron al código con sus debidas texturas y correcciones de posición.

Documentación (10-15 de noviembre): Esta tarea se llevó a cabo una vez finalizado el proyecto en su parte práctica, para tener una visión de todo lo realizado.

Diagrama de flujo



Alcance

El proyecto se desarrolla en una aplicación visual grafica en tiempo real, enfocada en la recreación de un escenario especifico la casa de Billy and Mandy y la implementación de técnicas en la computación grafica.

Este sistema permite que el usuario final pueda hacer distintas acciones dentro del entorno virtual:

Navegación libre: Exploración del escenario en primera persona mediante el movimiento de teclado y mouse con libertad de movimiento en los diferentes ejes.

Interacción del entorno: Capacidad de modificar el estado de objetos específicos mediante eventos del teclado.

Visualización de animaciones: Observación de comportamientos dinámicos, tanto cíclicos (personajes caminando, lampara oscilando, etc.

El software implementa un motor gráfico basado en OpenGL con las siguientes capacidades técnicas:

Carga de recursos: Importación de modelos 3D, imágenes y objetos animados, pipeline de iluminación Phong, soportando una luz direccional, múltiples luces puntuales con atenuación cuadrática, 1 luz tipo Spotlight acoplada a la cámara, Sistema de animación: Calculo de matrices de modelo en CPU para animaciones procedimentales (traslación, rotación, escala).

Animación a partir de un esqueleto, Implementación de Hardware Skinning en el Vertex Shader para la deformación de mallas basadas en huesos.

Limitantes

El sistema presenta algunas restricciones técnicas y funcionales como lo son:

Restricciones de plataforma y software: La aplicación esta compilada para entornos Windows sin embargo no es compatible con otros entornos como lo son macOS o Linux.

Ausencia de sombras dinámicas: El modelo de iluminación implementado no aplica técnicas de mapeo de sombras por lo que los objetos no proyectan sobre otros ni sobre el suelo.

Física de los objetos: Los objetos carecen de un relleno que impida el paso a los usuarios o que permita verlos por dentro.

El entorno de OpenGL limita geométricamente a realizar figuras de forma mas clara y precisa como en blender, además de que aplicar las texturas es algo más complejo.

ASSIMP es sensible a otros formatos que no sean jpg, lo que puede generar errores de visión o falta de texturas.

Metodología

La metodología principal del proyecto fue modelo en cascada, este enfoque lineal permitió avanzar ordenadamente a través de fases bien definidas, asegurando que cada etapa se completara para continuar con la siguiente:

Fase 1 Análisis de requisitos:

Requisitos funcionales: Se identificaron las necesidades de un entorno 3D navegable, interactivo y con animaciones simples y complejas.

Selección de temática: Se visualizo el alcance visualizado en la serie Billy and Mandy, determinando la lista de 7 objetos a modelar para cumplir con la rúbrica.

Fase 2: Diseño del sistema

Diseño de la escena: En blender se diseño una fachada e interiores lo mas realista posible, construyendo cada uno de los elementos que lo componen como las ventanas y puertas.

Diseño de clases: Se estructuro el programa separando la lógica en clases especializadas para camara, shader, model y modelanim.

Diseño de la interacción: Se definieron los mapas de las teclas para la navegación por medio de (WASD) y la activación de eventos con diferentes teclas.

Fase 3: Implementación

Configuración de un entorno virtual, inicializando ventanas con GLFW y carga de extensiones con GLEW, un motor de renderizado, con la programación de los shaders GLSL para soportar el texturizado de la iluminación Phong (Luz direccional, focal y spotlight).

Lógica de animación: Implementación de algoritmos matemáticos para las transformaciones geométricas (matrices de modelo).

Fase 4: Pruebas

Validación visual: Se verifico que la iluminación fuera la correcta y que los objetos se visualizaran de la mejor forma posible.

Se realizaron pruebas de animación, donde se comprobó la fluidez de las transiciones y la correcta ejecución de la secuencia compleja, además de valorar la fluides de cada animación de acuerdo con el uso de deltaTime.

Fase 5: Despliegue y mantenimiento

La fase final se centró en la generación de un producto entregable, se generó un manual técnico y uno de usuario, donde se plasmará todo lo elaborado durante el proyecto.

Documentación del código

Este proyecto fue elaborado en el entorno OpenGL, este proyecto este compuesto por un archivo main donde inicializaremos cada una de los componentes, en este apartado tenemos la lógica principal de las animaciones y de la interacción con el usuario.

Este proyecto se apoya de otros archivos proporcionados por el profesor entre los más relevantes se encuentra:

Shader.h: Este ayuda a extraer la lectura, compilación y conexión de los archivos. frag y .vs

Camera.h: Esta clase es fundamental ya que ayuda a crear una cámara sintética, con la cual podemos navegar en primera persona a través de los modelos realizados. Además, procesa la entrada del teclado y el mouse para generar una matriz de vista.

Model.h/mesh.h: Son herramientas que facilitan la carga de los modelos 3D estáticos como lo son los creados por autoría propia (casa, árbol, puertas) y los extraídos de diversas fuentes (Billy y espectro). Estos utilizan bibliotecas Assimp para leer los archivos y asignar sus texturas.

ModelAnim.h/ meshAnim.h: Estas clases son un aporte del profesor Carlos Aldair Roman Balbuena, que gestionan la carga de modelos 3D animados como lo es la animación de Billy (movimiento.dae), esta clase se basa en la jerarquía de huesos y calculan su transformación en cada fotograma.

Shaders

Este proyecto este compuesto por distintos shaders los cuales cumplían con una función específica en los proyectos.

Lighting: Es un shader principal que cumple con la función de iluminar los objetos siguiendo el modelo de iluminación Phong y texturas en todos los tipos de fuentes de luz (direccional, focal y linterna).

modelLoading: Fue utilizado para la textura básica de la casa y las puertas.

anim: Aplico el funcionamiento de la piel del esqueleto del personaje animado multiplicando la posición del vértice por la posición de la matriz, donde una suma ponderada de las transformaciones de los huesos que afectan el vértice.

lamp: Este shader dibuja un color sólido y genera una luz usada para el debug.

Archivo principal (ProyectoF.cpp)

Librerías (Líneas 7-27):

Para este proyecto se utilizaron las librerías proporcionadas, las cuales están compuestas por librerías de operaciones matemáticas, carga de GLEW y GLFW, así como el shader, cámara, modelo y modelo animado.

Prototipos y configuración de ventana (Líneas 30-44): En esta sección de código se enfoca en los prototipos de funcionamiento por teclado y mouse, además de las animaciones y el tamaño de pantalla donde se ejecutará.

Variables de cámara e iluminación (Líneas 49-62): Se registra el objeto cámara en su posición inicial dentro del mundo se ajusta en eje Y y X, se inicializa un arreglo que almacena la ubicación de las luces.

Vértices de primitivas (Líneas 65-102): Declaración de la matriz de geometría para todos los objetos, obtiene los datos de posición, coordenadas de textura, para cada cubo.

Variables de textura (Líneas 105-113): Variables que se utilizaran como identificadores para cada textura que se cargue.

Variables animaciones simples (Líneas 117-128): Se declaran cada una de las variables que se utilizaran en las animaciones simples, movimiento de lámpara, encendido y apagado y animación de apertura y cierre de puertas

Variables de animaciones complejas (Líneas 131-136): Se declaran las animaciones que se utilizaran para la animación compleja (Aparición del espectro), se creo una variable de tiempo de ejecución y una variable para suspender la animación, hasta que se active.

Main

Inicialización (Líneas 141-166): Se activan las librerías GLFW y GLEW, así como las dimensiones para la ventana principal, la conexión entre los movimientos del teclado y mouse, la configuración para que analice lo que tiene que dibujar en la ventana.

Carga de shaders y modelos (Líneas 169-183): En esta sección se cargan los shaders que estaremos utilizando:lighting shader, lampshader, modelshader y animshader.

Los modelos que ocuparemos para la creación de la casa, puertas, espectro y Billy.

Configuración de geometría primitiva (Líneas 186-202): En este bloque se prepara la geometría para el uso de la GPU. Se crea un VAO para almacenar la configuración y VBO para almacenar los datos, finalmente se definen los atributos para indicar al shader como leer la posición, normal y coordenadas de textura.

Configuración de texturas (Líneas 205-211): Se prepara el motor de renderizado para la carga de texturas y se declaran las variables que utilizara SOIL2 para la carga de las texturas.

Carga de texturas (Líneas 214-364): En este bloque se separó por secciones cada una de las texturas, para cada una se le crea una variable, se inicializa su configuración, los parámetros de texturizado, carga de la textura que se utilizara, calculo de los niveles del mipmap y finalmente se desvincula la textura actual.

Gracias a esto cada objeto pudo tener su textura de cada uno de los objetos (arbustos, sillón, televisión dividida en 3 partes para la textura inicial de la pantalla, textura final de la pantalla y carcasa de la televisión, lampara, planta, baño e inodoro).

Main → Game loop

Bucle principal del renderizado (Líneas 371-391): Es este bloque se inicializa el ciclo while, en el cual se ejecutará cada fotograma en el inicio del bloque encontramos, Tiempo de calculo para movimiento de animaciones fluidas, entrada para procesar los movimientos, lógica de la animación, asignación de un fondo de color, profundidad y preparación de la matriz vista.

Renderizado de la casa y puertas (Líneas 394-432): En este bloque se dibuja la geometría de la casa y puertas, al shader se le envían matrices de vista y proyección, se dibuja el modelo de la casa y se comienza con la lógica para la animación de las puertas donde se le asigna una bisagra, para que funcione en la apertura y cierre de la puerta.

Configuración del shader de iluminación (Líneas 435-450): Dibuja el escenario estático al activar el lightingShader para todos los objetos dinámicos y de decoración, se obtiene la ubicación en la memoria en las variables como model, view y projection). Además, se envían los datos que son compuestos para toda la escena como cámara, perspectiva y posición del espectador.

Configuración de fuentes de luz (Líneas 454-473): Este bloque envía los datos de las fuentes de luz a los uniforms del ligjting shader y se configuran los tipos de luz para cada fuente, luz direccional y luz focal.

Dibujo de arbustos (Líneas 480-507): Se dibuja 5 arbustos fuera de la casa, cada uno este compuesto por 3 figuras geométricas de diferentes tamaños, apiladas una encima de otra, para dar un efecto de arbusto, se vinculan las texturas asignadas en cada uno de los casos.

Dibujo del sofá (Líneas 510-534): Para el dibujo del sofá estuvo compuesto por 8 partes (base del sofá, respaldo, descansa brazos izquierdos y derecho, cojín izquierdo y derecho) cada uno con sus medidas correspondientes, de la misma forma cada uno tenia dimensiones distintas, por lo que se le agrego una variable para ajustar la textura de cada objeto.

Dibujo del inodoro y bañera (Líneas 537-571): El inodoro estuvo compuesto por 3 partes en la cual tenia una base, la taza y un tanque de agua, se utilizo la misma textura que la bañera, para darle una forma redonda a la taza se crearon cubos y se fueron escalando para darle una vista uniforme.

Para la bañera se utilizó una figura geométrica para el borde de la bañera, un conjunto de cubos unidos para que se forme el cuerpo de la bañera y finalmente se asignaron las patas para la bañera.

Dibujo de la planta (Líneas 574-610): Para el dibujo de la planta se utilizaron 6 figuras geométricas, la primera forma la repisa de la planta, la maceta, el tallo y 3 hojas es diferentes posiciones, para las hojas se utilizaron rotaciones para darle un efecto de hoja de una planta.

Animaciones simples de la lampara (Líneas 615-648): En este bloque se implementan cada una de las animaciones que implican a la lampara, para la primera se calcula la matriz y con ayuda de una variable calcula el tiempo para simular el tambaleo.

Para la segunda animación se configura la luz (pointLights), la posición de la luz no es estática por lo que para calcularla se utiliza multiplicando la posición de la bombilla por una matriz animada de la lampara, para realizar esta animación se creó una luz jerárquica que se mueve y tambalea junto con el objeto, finalmente se agrego un interruptor para encender o apagar la lampara, presionando la tecla “N” se activa o desactiva la luz de la lampara.

Dibujo de la lampara (Líneas 651-671): En este bloque se dibuja la lampara, el cual es un objeto compuesto, la primera parte de la lampara esta compuesta por una base con su cuerpo de la lampara, sin embargo, al tener una textura diferente se dividió en dos partes, para la segunda parte, se construyo el farol por lo que se apilaron diferentes figuras geométricas de diferente tamaño para darle esta ilusión.

Dibujo de Tv carcasa (Líneas 675-700): Para este objeto se divide en dos partes la pantalla y la carcasa de la TV, la carcasa esta compuesta por 4 patas, un botón y dos bloques cuadrados que representan el cuerpo de la pantalla, se declararon la misma textura para todo el objeto y se aplicaron las texturas de acuerdo con cada una de sus dimensiones.

Implementación de animación compleja Tv (Líneas 703-741): En esta sección se implementa la primera animación compleja, que se activa a través de la letra P, la animación principal es que un espectro salga de la tv, para esto se cargo un objeto de fantasma que sale de la pantalla, se calculo el movimiento lineal en el eje z con un movimiento sinusoidal en el eje x, el fantasma se dibujó con lightingShader para que las texturas reaccionen a la luz de la escena.

Además, se agregó una pequeña animación que cambiaba la textura de la pantalla de Tv simulando la apertura de un portal.

Modelo de espectro obtenido de:

Cruz Ponce, A. A. (2021, 17 de abril). *Fantasma* [Modelo 3D]. Sketchfab. https://sketchfab.com/3d-models/fantasma-8928e3d13773497291605e5ef1fc0aed

Animación compleja caminata de Billy (Líneas 745-773): En este bloque se utilizo el modelo de Billy, se utilizó el shaderanim que procesa la deformación del esqueleto. Calcula la matriz para posicionar y escalar a Billy en el jardín. Finalmente se realiza una llamada al modelo para dibujarlo con la lógica interna de ModelAnim, que ayuda a calculas las matrices de los huesos en tiempo real y dibuja el modelo animado.

Modelo de Billy obtenido de:

calvinwil5782. (2023, 13 de noviembre). *Billy* [Modelo 3D]. Sketchfab. https://sketchfab.com/3d-models/billy-04a755991fee4d99ba6027a80e6aeb65

Movimientos de cámara (Líneas 776-796): En esta sección se genera los movimientos del usuario a través de las teclas ASDW con la cual se puede desplazar a lo largo del mundo.

Función KeyCallback (Líneas 798-866): Esta función es el manejador de eventos de teclado, donde se ejecuta automáticamente por GLFW solo cuando la tecla es presionada o soltada. El manejo es doble la primera parte es para las acciones en el teclado, como abrir y cerrar puertas, tambaleo de la lampara, cerrar programa, etc.

La otra parte actualiza el arreglo global, marcando la tecla como la presionada o soltada, este arreglo es leído por cada uno de los fotogramas.

Función de la animación (Líneas 868-913): Es una función lógica que llama a cada fotograma, por lo que su función es activar las variables para las animaciones.

La animación de las puertas: Implementan una interpolación lineal, donde verifica cada una de las banderas hasta alcanzar un ángulo de 0 o 90°.

La lampara calcula el ángulo usando una función sinusoidal para crear una oscilación suave.

Temporizador de animación compleja: Si la variable asignada es verdadera, incrementa el deltatime para ser independiente, esta variable es la que controla la trayectoria no lineal del fantasma, la animación tiene un lapso de 5 segundos.

Función MouseCallback (Líneas 916-929): Esta función se apoya de la librería GLFW para controlar el movimiento del mouse. Su función principal es la rotación de la cámara, evita saltos bruscos de la ventana evitando movimientos rápidos o cambio de ángulo.

Función ObjetoDraw (Líneas 932-940): Esta función ayuda a renderizar la primitiva del cubo cargando una sola vez en el VAO, utiliza diferentes transformaciones. Aplica las transformaciones necesarias permitiendo ensamblar objetos complejos a partir de múltiples cubos.

Conclusión

Este proyecto es un recopilatorio de todas las practicas realizadas a lo largo del semestre creando un entorno virtual interactivo basado en las aventuras de Billy and Mandy. Atreves de una herramienta de renderizado se cumplió con el objetivo de integrar modelado, texturizado, iluminación y animación en tiempo real.

En este proyecto se utilizaron distintas herramientas que son fundamentales en OpenGL, el desarrollo de un modelo desde cero en Blender fue interesante ya que es un mundo el cual te puede generar infinitas posibilidades de modelado de objetos, el aprender a usar esta herramienta es algo muy positivo, debido a que me ayuda a visualizar una nueva área de desarrollo.

Dominar los pipelines gráficos es algo complejo, pero permiten un control sobre toda la visualización, logrando efectos de iluminación complejos utilizando múltiples fuentes de luz que otorgan profundidad y realismo.

Las animaciones complejas, ayudan a observar movimiento con funciones matemáticas, para crear trayectorias no lineales y comportamientos gráficos.

Este proyecto me ayudo a ver de una forma diferente las animaciones 3D, sobre todo las animaciones antiguas donde se puede observar errores muy absurdos de geometría iluminación o texturizado, sin embargo, con el paso del tiempo las nuevas herramientas nos han ayudado a mejorar gráficamente para tener proyectos

visuales donde la línea del realismo y lo generado desde un motor grafico es muy delgada.

Referencias

Cruz Ponce, A. A. (2021, 17 de abril). Fantasma [Modelo 3D]. Sketchfab. https://sketchfab.com/3d-models/fantasma-8928e3d13773497291605e5ef1fc0aed

calvinwil5782. (2023, 13 de noviembre). Billy [Modelo 3D]. Sketchfab. https://sketchfab.com/3d-models/billy-04a755991fee4d99ba6027a80e6aeb65

Integración de GitHub Desktop con el Proyecto de Computación … [Video]. (n.d.). YouTube. https://www.youtube.com/watch?v=XZYgHmnB1vU

Animación Básica en OpenGL [Video]. (n.d.). YouTube. https://www.youtube.com/watch?v=luybf6bTzhc

Carga de Texturas en OpenGL Introducción a las UV s [Video]. (n.d.). YouTube. https://www.youtube.com/watch?v=21jOb3w_T8k

Animación por Keyframes en OpenGL: Movimiento y Rotación [Video]. (n.d.). YouTube. https://www.youtube.com/watch?v=5l1A6Lxa3GQ

IEP. (2022, 7 de julio). Metodología Waterfall: Modelo de gestión de proyectos en cascada. IEP. Recuperado de https://iep.edu.es/metodologia-waterfall/

Universidad de Valladolid. (s. f.). Metodología Waterfall (modelo en cascada). Universidad de Valladolid. Recuperado de https://uvadoc.uva.es/bitstream/10324/71366/1/TFG-G6918.pdf

Rominaescalonab. (s. f.). 59 ideas de Billy y Mandy [Tablero de imágenes]. Pinterest. Recuperado de https://cl.pinterest.com/rominaescalonab/billy-y-mandy/

Código main completo

```
// ProyectoF.cpp
// Proyecto Final - Casa de Billy y Mandy
// Laboratorio de Computación Gráfica e interaccion humano-computadora
//Alumno:Josué Vázquez Cárdenas

//Librerias utilizadas
#include <iostream>
#include <cmath>

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

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

// SOIL2 for texture loading
#include "SOIL2/SOIL2.h"

// Custom Headers
#include "Shader.h"
#include "Camera.h"
#include "Model.h"
#include "modelAnim.h"


// Function prototypes
//Registra movimientos en el teclado
void KeyCallback(GLFWwindow* window, int key, int scancode, int action,
int mode);
//Registra movimiento en el mouse
void MouseCallback(GLFWwindow* window, double xPos, double yPos);
//Detecta el movimiento en teclas presionadas
void DoMovement();
//Funcion para analisis de las animaciones
void Animation();
//Dibuja las primitivas del cubo
void ObjetoDraw(glm::mat4 base, glm::vec3 escala, glm::vec3 traslado,
GLint uniformModel);

// Window dimensions
const GLuint WIDTH = 1600, HEIGHT = 900;
int SCREEN_WIDTH, SCREEN_HEIGHT;


// Camera
//Posicion inicial de la camara
```

```
Camera camera(glm::vec3(-5.0f, 5.0f, 37.0f));
GLfloat lastX = WIDTH / 2.0f;//Posicion del mouse en X
GLfloat lastY = HEIGHT / 2.0f;//Posicion del mouse en Y
bool keys[1024];//Arreglo para registro de teclas
bool firstMouse = true; //Evitar salto de camara al entrar a la venta

// Light attributes
bool active; // For debug light
glm::vec3 Light1 = glm::vec3(0); // For debug light color animation
glm::vec3 PosIni(-0.0f, 0.2f, -0.0f);

// Positions of the point lights
glm::vec3 pointLightPositions[] = {
        glm::vec3(0.0f, 5.0f, 0.0f),glm::vec3(0.0f, 0.0f,
0.0f),glm::vec3(0.0f, 0.0f, 0.0f),glm::vec3(0.0f, 0.0f, 0.0f)};

// Cube vertices (Pos, Normal, TexCoords)
float vertices[] = {
                -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,  0.0f, 0.0f,
                 0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,  1.0f, 0.0f,
                 0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,  1.0f, 1.0f,
                 0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,  1.0f, 1.0f,
                -0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,  0.0f, 1.0f,
                -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,  0.0f, 0.0f,
                -0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f,  0.0f, 0.0f,
                 0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f,  1.0f, 0.0f,
                 0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,  1.0f, 1.0f,
                 0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,  1.0f, 1.0f,
                -0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,  0.0f, 1.0f,
                -0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f,  0.0f, 0.0f,
                -0.5f,  0.5f,  0.5f, -1.0f,  0.0f,  0.0f,  1.0f, 0.0f,
                -0.5f,  0.5f, -0.5f, -1.0f,  0.0f,  0.0f,  1.0f, 1.0f,
                -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,  0.0f, 1.0f,
                -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,  0.0f, 1.0f,
                -0.5f, -0.5f,  0.5f, -1.0f,  0.0f,  0.0f,  0.0f, 0.0f,
                -0.5f,  0.5f,  0.5f, -1.0f,  0.0f,  0.0f,  1.0f, 0.0f,
                 0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f,  1.0f, 0.0f,
                 0.5f,  0.5f, -0.5f,  1.0f,  0.0f,  0.0f,  1.0f, 1.0f,
                 0.5f, -0.5f, -0.5f,  1.0f,  0.0f,  0.0f,  0.0f, 1.0f,
                 0.5f, -0.5f, -0.5f,  1.0f,  0.0f,  0.0f,  0.0f, 1.0f,
                 0.5f, -0.5f,  0.5f,  1.0f,  0.0f,  0.0f,  0.0f, 0.0f,
                 0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f,  1.0f, 0.0f,
                -0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,  0.0f, 1.0f,
                 0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,  1.0f, 1.0f,
                 0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,  1.0f, 0.0f,
                 0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,  1.0f, 0.0f,
                -0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,  0.0f, 0.0f,
                -0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,  0.0f, 1.0f,
                -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,  0.0f, 1.0f,
                 0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,  1.0f, 1.0f,
                 0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,  1.0f, 0.0f,
                 0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,  1.0f, 0.0f,
                -0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,  0.0f, 0.0f,
```

```
                -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,  0.0f, 1.0f
        };

// --- Variables de textura ---
GLuint sofaTexture;//Textura del sofa
GLuint lampBaseTexture;
GLuint lampShadeTexture;//Textura de lampara
GLuint tvCasingTexture;//Textura de la tv
GLuint tvScreenTexture;//Textura de la tv apagada
GLuint inodoroTexture;//Textura del inodoro
GLuint plantaTexture;//Textura de la planta interior
GLuint arbustoTexture;//Textura del arbusto
GLuint tvStaticTexture;//Textura del portal

//Variables para Animaciones Simples
//Variables de puertas
float lampWobble = 0.0f;
bool lampAnimationActive = true;
bool lampLightActive = true; // Interruptor para la luz

// Variables de animacion de puertas
float doorE_angle = 0.0f;
float doorS_angle = 0.0f;
float doorB_angle = 0.0f;

bool doorE_open = false;
bool doorS_open = false;
bool doorB_open = false;

//Variables Animacion compleja Tv
bool tvAnimationActive = false; // Inicia apagada
float tvAnimTime = 0.0f;      // Temporizador para la animacion

// Deltatime
GLfloat deltaTime = 0.0f;
GLfloat lastFrame = 0.0f;

int main()
{
        // Init GLFW
        glfwInit();
        //Creacion de la ventana de visualizacion
        GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "Proyecto
Final - Casa Billy y Mandy", nullptr, nullptr);

        //Prueba de error, se genera en casa de que la pantalla no se
muestre
        if (nullptr == window)
        {
                std::cout << "Failed to create GLFW window" << std::endl;
                glfwTerminate();
                return EXIT_FAILURE;
        }
```

```
        //Especificaciones de pantalla
        glfwMakeContextCurrent(window);
        glfwGetFramebufferSize(window, &SCREEN_WIDTH, &SCREEN_HEIGHT);
        //Registros de CALLBACKS, para teclado y mouse
        glfwSetKeyCallback(window, KeyCallback);
        glfwSetCursorPosCallback(window, MouseCallback);

        //Inicializacion de GLEW
        glewExperimental = GL_TRUE;
        if (GLEW_OK != glewInit())
        {
                std::cout << "Failed to initialize GLEW" << std::endl;
                return EXIT_FAILURE;
        }
        glViewport(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT);

        // Shaders
        Shader lightingShader("Shader/lighting.vs",
"Shader/lighting.frag");//Shader para la iluminacion Phong
        Shader lampShader("Shader/lamp.vs", "Shader/lamp.frag");//Shader
para objetos simples
        Shader modelShader("Shader/modelLoading.vs",
"Shader/modelLoading.frag");//Shader para la carga de modelos
        Shader animShader("Shader/anim.vs", "Shader/anim.frag");//Shader
para la animacion de Billy

        // Carga de Modelos
        Model casa((char*)"Models/Casa/Casa.obj");//Modelo de la casa
        //Modelo de las puertas
        Model PuertaE((char*)"Models/Puertas/PuertaE.obj");
        Model PuertaS((char*)"Models/Puertas/PuertaS.obj");
        Model PuertaB((char*)"Models/Puertas/PuertaB.obj");

        Model Fantasma((char*)"Models/Espectro/ghost2.obj");//Modelo del
espectro
        ModelAnim
animacionPersonaje("Animaciones/Billy/movimiento.dae");//Modelo de la
animacion del personaje
        animacionPersonaje.initShaders(animShader.Program);//Animacion del
personaje

        // VAO y VBO para los cubos (primitivas)
        GLuint VBO, VAO;
        glGenVertexArrays(1, &VAO); //Creacion del (VAO) para la
configuracion
        glGenBuffers(1, &VBO);//Creacion del (VBO) para almacenar los
datos de los vertices
        glBindVertexArray(VAO);//Activacion del VAO
        glBindBuffer(GL_ARRAY_BUFFER, VBO);//Activacion del VBO
        glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices,
GL_STATIC_DRAW);

        // Vertex Attribs
```

```
        //Atributos de posiciones
        glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 *
sizeof(GLfloat), (GLvoid*)0);
        glEnableVertexAttribArray(0);
        //Atributos de Normales
        glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 *
sizeof(GLfloat), (GLvoid*)(3 * sizeof(GLfloat)));
        glEnableVertexAttribArray(1);
        //Atributos de texturas
        glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 *
sizeof(GLfloat), (GLvoid*)(6 * sizeof(GLfloat)));
        glEnableVertexAttribArray(2);

        // Set texture units
        lightingShader.Use();
        glUniform1i(glGetUniformLocation(lightingShader.Program,
"material.diffuse"), 0);
        glUniform1i(glGetUniformLocation(lightingShader.Program,
"material.specular"), 1);

        // --- Carga de Texturas ---
        int width, height;//Dimensiones de la textura
        unsigned char* image;//Puntero a los datos de pixeles

        // Textura Arbusto
        glGenTextures(1, &arbustoTexture);//variable para la textura
        glBindTexture(GL_TEXTURE_2D, arbustoTexture);//Inicializacion de
la textura para su configuracion
        //Configuracion de los parametros de texturizado
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

        //Carga de la imagen arbusto.jpg
        image = SOIL_load_image("images/arbusto.jpg", &width, &height, 0,
SOIL_LOAD_RGB);
        if (image) {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,
GL_RGB, GL_UNSIGNED_BYTE, image);
                glGenerateMipmap(GL_TEXTURE_2D);//Calculo de niveles del
mipmap
        }
        glBindTexture(GL_TEXTURE_2D, 0);//Desvincula la textura actual

        // Textura Sofa
        glGenTextures(1, &sofaTexture);//variable para la textura
        glBindTexture(GL_TEXTURE_2D, sofaTexture);//Inicializacion de la
textura para su configuracion
        //Configuracion de los parametros de texturizado
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

```
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

        //Carga de la imagen sofa.jpg
        image = SOIL_load_image("images/sofa.jpg", &width, &height, 0,
SOIL_LOAD_RGB);
        if (image) {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,
GL_RGB, GL_UNSIGNED_BYTE, image);
                glGenerateMipmap(GL_TEXTURE_2D);//Calculo de niveles del
mipmap
        }
        glBindTexture(GL_TEXTURE_2D, 0);//Desvincula la textura actual

        // Texturas Lampara (Base)
        glGenTextures(1, &lampBaseTexture);//variable para la textura
        glBindTexture(GL_TEXTURE_2D, lampBaseTexture);//Inicializacion de
la textura para su configuracion
        //Configuracion de los parametros de texturizado
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

        //Carga de la imagen lampara.jpg
        image = SOIL_load_image("images/lampara.JPG", &width, &height, 0,
SOIL_LOAD_RGB);
        if (image) {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,
GL_RGB, GL_UNSIGNED_BYTE, image);
                glGenerateMipmap(GL_TEXTURE_2D);//Calculo de niveles del
mipmap
        }
        glBindTexture(GL_TEXTURE_2D, 0);//Desvincula la textura actual

        // Texturas Lampara (Pantalla)
        glGenTextures(1, &lampShadeTexture);//variable para la textura
        glBindTexture(GL_TEXTURE_2D, lampShadeTexture);//Inicializacion de
la textura para su configuracion
        //Configuracion de los parametros de texturizado
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

        //Carga de la imagen Tela.jpg
        image = SOIL_load_image("images/Tela.JPG", &width, &height, 0,
SOIL_LOAD_RGB);
        if (image) {
```

```
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,
GL_RGB, GL_UNSIGNED_BYTE, image);
                glGenerateMipmap(GL_TEXTURE_2D);//Calculo de niveles del
mipmap
        }
        glBindTexture(GL_TEXTURE_2D, 0);//Desvincula la textura actual

        // Texturas TV (Carcasa)
        glGenTextures(1, &tvCasingTexture);//variable para la textura
        glBindTexture(GL_TEXTURE_2D, tvCasingTexture);//Inicializacion de
la textura para su configuracion
        //Configuracion de los parametros de texturizado
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

        //Carga de la imagen cuerpo.png
        image = SOIL_load_image("images/cuerpo.PNG", &width, &height, 0,
SOIL_LOAD_RGB);
        if (image) {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,
GL_RGB, GL_UNSIGNED_BYTE, image);
                glGenerateMipmap(GL_TEXTURE_2D);//Calculo de niveles del
mipmap
        }
        glBindTexture(GL_TEXTURE_2D, 0);//Desvincula la textura actual

        // Texturas TV (Pantalla)
        glGenTextures(1, &tvScreenTexture);//variable para la textura
        glBindTexture(GL_TEXTURE_2D, tvScreenTexture);//Inicializacion de
la textura para su configuracion
        //Configuracion de los parametros de texturizado
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

        //Carga de la imagen pantalla.png
        image = SOIL_load_image("images/pantalla.PNG", &width, &height, 0,
SOIL_LOAD_RGB);
        if (image) {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,
GL_RGB, GL_UNSIGNED_BYTE, image);
                glGenerateMipmap(GL_TEXTURE_2D);//Calculo de niveles del
mipmap
        }
        glBindTexture(GL_TEXTURE_2D, 0);//Desvincula la textura actual

        // Textura TV portal
        glGenTextures(1, &tvStaticTexture);//variable para la textura
```

```
        glBindTexture(GL_TEXTURE_2D, tvStaticTexture);//Inicializacion de
la textura para su configuracion
        //Configuracion de los parametros de texturizado
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

        //Carga de la imagen portal.jpg
        image = SOIL_load_image("images/portal.jpg", &width, &height, 0,
SOIL_LOAD_RGB);
        if (image) {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,
GL_RGB, GL_UNSIGNED_BYTE, image);
                glGenerateMipmap(GL_TEXTURE_2D);//Calculo de niveles del
mipmap
        }
        glBindTexture(GL_TEXTURE_2D, 0);//Desvincula la textura actual

        // Textura Inodoro Bañera
        glGenTextures(1, &inodoroTexture);//variable para la textura
        glBindTexture(GL_TEXTURE_2D, inodoroTexture);//Inicializacion de
la textura para su configuracion
        //Configuracion de los parametros de texturizado
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

        //Carga de la imagen baño.jpg
        image = SOIL_load_image("images/baño.jpg", &width, &height, 0,
SOIL_LOAD_RGB);
        if (image) {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,
GL_RGB, GL_UNSIGNED_BYTE, image);
                glGenerateMipmap(GL_TEXTURE_2D);//Calculo de niveles del
mipmap
        }
        glBindTexture(GL_TEXTURE_2D, 0);//Desvincula la textura actual

        // Textura Planta
        glGenTextures(1, &plantaTexture);//variable para la textura
        glBindTexture(GL_TEXTURE_2D, plantaTexture);//Inicializacion de la
textura para su configuracion
        //Configuracion de los parametros de texturizado
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

```
        //Carga de la imagen planta.jpg
        image = SOIL_load_image("images/planta.jpg", &width, &height, 0,
SOIL_LOAD_RGB);
        if (image) {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,
GL_RGB, GL_UNSIGNED_BYTE, image);
                glGenerateMipmap(GL_TEXTURE_2D);//Calculo de niveles del
mipmap
        }
        glBindTexture(GL_TEXTURE_2D, 0);//Desvincula la textura actual


        // Matriz de Proyección
        glm::mat4 projection = glm::perspective(camera.GetZoom(),
(GLfloat)SCREEN_WIDTH / (GLfloat)SCREEN_HEIGHT, 0.1f, 100.0f);

        // Game loop
        while (!glfwWindowShouldClose(window))
        {
                //Gestion de tiempo y entrada
                // Deltatime, PollEvents, DoMovement
                GLfloat currentFrame = (float)glfwGetTime();
                deltaTime = currentFrame - lastFrame;
                lastFrame = currentFrame;

                glfwPollEvents();//Procesa eventos de ventana para cerrar
y pulsaciones en el teclado
                DoMovement();//Procesa entradas de movimiento constante

                Animation(); // Actualiza las animaciones

                // Clear buffers, Enable Depth Test
                glClearColor(0.5f, 0.8f, 1.0f, 1.0f);//Color del fondo
                glClear(GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT);//Limpieza y profundidad del buffer
                glEnable(GL_DEPTH_TEST);//Profundidad en objetos 3D

                // --- MATRICES COMUNES ---
                glm::mat4 view = camera.GetViewMatrix();//Matriz de vista
                glm::mat4 model = glm::mat4(1.0f);//Matriz de modelo

                // Dibujo de la casa
                modelShader.Use();
                //Matrices para la camara y proyeccion al shader

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

        glUniformMatrix4fv(glGetUniformLocation(modelShader.Program,
"projection"), 1, GL_FALSE, glm::value_ptr(projection));
                //Renderizado de la casa
                model = glm::mat4(1.0f);
```

```
        glUniformMatrix4fv(glGetUniformLocation(modelShader.Program,
"model"), 1, GL_FALSE, glm::value_ptr(model));
                casa.Draw(modelShader);//Dibuja el modelo  de la casa

                //// Dibujo de cada una de las puertas animadas
                // === Puerta Entrada
                glm::mat4 modelDoorE = glm::mat4(1.0f);
                glm::vec3 hingePosE = glm::vec3(6.0f, 0.0f, -
4.8f);//Posicion de la bisagra
                //Transformaciones en la puerta entrada
                modelDoorE = glm::translate(modelDoorE, -hingePosE);
                modelDoorE = glm::rotate(modelDoorE,
glm::radians(doorE_angle), glm::vec3(0.0f, 0.5f, 0.0f));
                modelDoorE = glm::translate(modelDoorE, hingePosE);

        glUniformMatrix4fv(glGetUniformLocation(modelShader.Program,
"model"), 1, GL_FALSE, glm::value_ptr(modelDoorE));
                PuertaE.Draw(modelShader);//Dibuja el modelo  de la puerta
entrada

                // === Puerta Sala
                glm::mat4 modelDoorS = glm::mat4(1.0f);
                glm::vec3 hingePosS = glm::vec3(5.3f, 0.0f,
2.35f);//Posicion de la bisagra
                //Transformaciones en la puerta Sala
                modelDoorS = glm::translate(modelDoorS, -hingePosS);
                modelDoorS = glm::rotate(modelDoorS,
glm::radians(doorS_angle), glm::vec3(0.0f, 0.5f, 0.0f));
                modelDoorS = glm::translate(modelDoorS, hingePosS);

        glUniformMatrix4fv(glGetUniformLocation(modelShader.Program,
"model"), 1, GL_FALSE, glm::value_ptr(modelDoorS));
                PuertaS.Draw(modelShader);//Dibuja el modelo  de la puerta
Sala

                // === Puerta Baño
                glm::mat4 modelDoorB = glm::mat4(1.0f);
                glm::vec3 hingePosB = glm::vec3(3.3f, 0.0f,
5.25f);//Posicion de la bisagra
                //Transformaciones en la puerta Baño
                modelDoorB = glm::translate(modelDoorB, -hingePosB);
                modelDoorB = glm::rotate(modelDoorB,
glm::radians(doorB_angle), glm::vec3(0.0f, 0.5f, 0.0f));
                modelDoorB = glm::translate(modelDoorB, hingePosB);

        glUniformMatrix4fv(glGetUniformLocation(modelShader.Program,
"model"), 1, GL_FALSE, glm::value_ptr(modelDoorB));
                PuertaB.Draw(modelShader);//Dibuja el modelo  de la puerta
Baño

                // DIBUJAR OBJETOS CON LUZ (lightingShader)
                lightingShader.Use();
```

```
                //Optimizacion para la ubicacion de cada frame
                GLint modelLoc =
glGetUniformLocation(lightingShader.Program, "model");
                GLint viewLoc =
glGetUniformLocation(lightingShader.Program, "view");
                GLint projLoc =
glGetUniformLocation(lightingShader.Program, "projection");
                GLint viewPosLoc =
glGetUniformLocation(lightingShader.Program, "viewPos");

                //Envio de Uniforms "Globales"
                glUniformMatrix4fv(viewLoc, 1, GL_FALSE,
glm::value_ptr(view));
                glUniformMatrix4fv(projLoc, 1, GL_FALSE,
glm::value_ptr(projection));
                //Envio de posicion de la camara al shader
                glUniform3f(viewPosLoc, camera.GetPosition().x,
camera.GetPosition().y, camera.GetPosition().z);
                //Declaracion del brillo a utilizar
                glUniform1f(glGetUniformLocation(lightingShader.Program,
"material.shininess"), 5.0f);
                //Ubicacion para la repeticion de texturas
                GLint tilingLoc =
glGetUniformLocation(lightingShader.Program, "texTiling");

                // --- Configuracion de Luces para lightingShader
                // Luz Direccional (Sol)
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"dirLight.direction"), -0.2f, -1.0f, -0.3f);
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"dirLight.ambient"), 0.6f, 0.6f, 0.6f);//Luz ambiental
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"dirLight.diffuse"), 0.6f, 0.6f, 0.6f);//Color de la luz
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"dirLight.specular"), 0.3f, 0.3f, 0.3f);//Color del brillo

                // Luz de Punto 0 (Luz de debug)
                //Calcula un color dinamico para un efecto de parpadeo
                glm::vec3 lightColor;
                lightColor.x = abs(sin((float)glfwGetTime() * Light1.x));
                lightColor.y = abs(sin((float)glfwGetTime() * Light1.y));
                lightColor.z = sin((float)glfwGetTime() * Light1.z);
                //Envia propiedades de la luz al punto del shader
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].position"), pointLightPositions[0].x,
pointLightPositions[0].y, pointLightPositions[0].z);
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].ambient"), 0.0f, 0.0f, 0.0f);
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].diffuse"), lightColor.x * 0.5f, lightColor.y * 0.5f,
lightColor.z * 0.5f);
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].specular"), 1.0f, 0.2f, 0.2f);
```

```
                //Parametros de atenuacion constante, lineal y cuadratica
                glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].constant"), 1.0f);
                glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].linear"), 0.045f);
                glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].quadratic"), 0.075f);


                //-------------------Dibujo de objetos--------------------
-

                //DIBUJAR LOS ARBUSTOS-- -

                glBindVertexArray(VAO);//Activa el VAO del cubo
                glActiveTexture(GL_TEXTURE0);//Activa su textura en 0
                glBindTexture(GL_TEXTURE_2D, arbustoTexture);//Vincula a
su textura
                glUniform2f(tilingLoc, 1.0f, 1.0f); // Asegura que la
textura no se repita

                glm::mat4 modelArbustoBase = glm::mat4(1.0f);
                // Arbusto 1
                ObjetoDraw(modelArbustoBase, glm::vec3(1.5f, 0.8f, 1.5f),
glm::vec3(0.0f, 0.6f, 6.2f), modelLoc);//Parte central
                ObjetoDraw(modelArbustoBase, glm::vec3(1.0f, 0.2f, 1.0f),
glm::vec3(0.0f, 1.1f, 6.2f), modelLoc);//Parte superior
                ObjetoDraw(modelArbustoBase, glm::vec3(1.0f, 0.2f, 1.0f),
glm::vec3(0.0f, 0.1f, 6.2f), modelLoc);//Parte inferior
                // Arbusto 2
                ObjetoDraw(modelArbustoBase, glm::vec3(0.7f, 0.5f, 0.7f),
glm::vec3(3.0f, 0.25f, 6.2f), modelLoc);//Parte central
                ObjetoDraw(modelArbustoBase, glm::vec3(0.8f, 0.6f, 0.8f),
glm::vec3(3.0f, 0.8f, 6.2f), modelLoc);//Parte superior
                ObjetoDraw(modelArbustoBase, glm::vec3(1.2f, 0.8f, 1.2f),
glm::vec3(3.0f, 0.5f, 6.2f), modelLoc);//Parte inferior
                // Arbusto 3
                ObjetoDraw(modelArbustoBase, glm::vec3(1.7f, 0.8f, 1.7f),
glm::vec3(-3.0f, 0.6f, 6.2f), modelLoc);//Parte central
                ObjetoDraw(modelArbustoBase, glm::vec3(1.2f, 0.2f, 1.2f),
glm::vec3(-3.0f, 1.1f, 6.2f), modelLoc);//Parte superior
                ObjetoDraw(modelArbustoBase, glm::vec3(1.2f, 0.2f, 1.2f),
glm::vec3(-3.0f, 0.1f, 6.2f), modelLoc);//Parte inferior
                // Arbusto 4
                ObjetoDraw(modelArbustoBase, glm::vec3(1.7f, 0.8f, 1.7f),
glm::vec3(-8.0f, 0.6f, 6.2f), modelLoc);//Parte central
                ObjetoDraw(modelArbustoBase, glm::vec3(1.2f, 0.2f, 1.2f),
glm::vec3(-8.0f, 1.1f, 6.2f), modelLoc);//Parte superior
                ObjetoDraw(modelArbustoBase, glm::vec3(1.2f, 0.2f, 1.2f),
glm::vec3(-8.0f, 0.1f, 6.2f), modelLoc);//Parte inferior
                // Arbusto 5
                ObjetoDraw(modelArbustoBase, glm::vec3(1.7f, 0.8f, 1.7f),
glm::vec3(-12.0f, 0.6f, 6.2f), modelLoc);//Parte central
```

```
                ObjetoDraw(modelArbustoBase, glm::vec3(1.2f, 0.2f, 1.2f),
glm::vec3(-12.0f, 1.1f, 6.2f), modelLoc);//Parte superior
                ObjetoDraw(modelArbustoBase, glm::vec3(1.2f, 0.2f, 1.2f),
glm::vec3(-12.0f, 0.1f, 6.2f), modelLoc);//Parte inferior

                glBindTexture(GL_TEXTURE_2D, 0);// Desvincula la textura

                // Dibujo del sofá ---
                glBindVertexArray(VAO);//Activa el VAO del cubo
                glActiveTexture(GL_TEXTURE0);//Activa su textura en 0
                glBindTexture(GL_TEXTURE_2D, sofaTexture);//Vincula a su
textura

                //Ubicacion del sofa
                glm::mat4 modelSofaBase = glm::translate(glm::mat4(1.0f),
glm::vec3(-1.5f, 0.0f, -0.9f));
                //Ajuste de la textura
                glUniform2f(tilingLoc, 6.0f, 2.0f);
                //Base del sofa
                ObjetoDraw(modelSofaBase, glm::vec3(4.0f, 0.8f, 1.5f),
glm::vec3(0.0f, 0.4f, 0.0f), modelLoc);
                //Ajuste de la textura
                glUniform2f(tilingLoc, 6.0f, 2.0f);
                //Respaldo de sofa
                ObjetoDraw(modelSofaBase, glm::vec3(4.0f, 1.8f, 0.5f),
glm::vec3(0.0f, 0.9f, -1.0f), modelLoc);
                //Ajuste de la textura
                glUniform2f(tilingLoc, 1.35f, 1.1f);
                //Reposa brazos derecho
                ObjetoDraw(modelSofaBase, glm::vec3(1.35f, 1.1f, 0.3f),
glm::vec3(0.83f, 1.4f, -0.59f), modelLoc);
                //Reposa brazos izquierdo
                ObjetoDraw(modelSofaBase, glm::vec3(1.35f, 1.1f, 0.3f),
glm::vec3(-0.83f, 1.4f, -0.59f), modelLoc);
                //Cojin izquierdo
                ObjetoDraw(modelSofaBase, glm::vec3(0.5f, 0.6f, 1.5f),
glm::vec3(-1.75f, 1.1f, 0.0f), modelLoc);
                //Cojin derecho
                ObjetoDraw(modelSofaBase, glm::vec3(0.5f, 0.6f, 1.5f),
glm::vec3(1.75f, 1.1f, 0.0f), modelLoc);
                glBindTexture(GL_TEXTURE_2D, 0);// Desvincula la textura

                // Dibujo del inodoro
                glBindVertexArray(VAO);//Activa el VAO del cubo
                glActiveTexture(GL_TEXTURE0);//Activa su textura en 0
                glBindTexture(GL_TEXTURE_2D, inodoroTexture); // Usamos la
textura del baño
                glUniform2f(tilingLoc, 1.0f, 1.0f); // Tiling 1x1

                // --- Matriz Base del Inodoro ---
                glm::mat4 modelInodoroBase =
glm::translate(glm::mat4(1.0f), glm::vec3(2.8f, 0.0f, -7.25f));
                // Tanque
```

```
                ObjetoDraw(modelInodoroBase, glm::vec3(0.5f, 0.9f, 1.2f),
glm::vec3(0.0f, 1.2f, 0.4f), modelLoc);
                // Taza
                ObjetoDraw(modelInodoroBase, glm::vec3(1.5f, 0.2f, 1.0f),
glm::vec3(-0.4f, 0.10f, 0.4f), modelLoc);
                ObjetoDraw(modelInodoroBase, glm::vec3(0.5f, 0.1f, 0.4f),
glm::vec3(-0.7f, 0.30f, 0.4f), modelLoc);
                ObjetoDraw(modelInodoroBase, glm::vec3(0.6f, 0.1f, 0.5f),
glm::vec3(-0.7f, 0.40f, 0.4f), modelLoc);
                ObjetoDraw(modelInodoroBase, glm::vec3(0.7f, 0.1f, 0.6f),
glm::vec3(-0.7f, 0.50f, 0.4f), modelLoc);
                ObjetoDraw(modelInodoroBase, glm::vec3(0.8f, 0.1f, 0.7f),
glm::vec3(-0.7f, 0.60f, 0.4f), modelLoc);
                ObjetoDraw(modelInodoroBase, glm::vec3(0.9f, 0.1f, 0.8f),
glm::vec3(-0.7f, 0.70f, 0.4f), modelLoc);
                ObjetoDraw(modelInodoroBase, glm::vec3(1.0f, 0.1f, 0.9f),
glm::vec3(-0.7f, 0.80f, 0.4f), modelLoc);

                // --- DIBUJAR LA BAÑERA ---
                //Ubicacion de la bañera
                glm::mat4 modelBaneraBase =
glm::translate(glm::mat4(1.0f), glm::vec3(2.0f, 0.0f, -4.0f));
                // Parte superior de la bañera
                ObjetoDraw(modelBaneraBase, glm::vec3(1.6f, 0.2f, 2.6f),
glm::vec3(0.0f, 1.2f, 0.0f), modelLoc);
                //Forro de la bañera
                ObjetoDraw(modelBaneraBase, glm::vec3(1.4f, 0.2f, 2.3f),
glm::vec3(0.0f, 1.0f, 0.0f), modelLoc);
                ObjetoDraw(modelBaneraBase, glm::vec3(1.4f, 0.2f, 2.2f),
glm::vec3(0.0f, 0.8f, 0.0f), modelLoc);
                ObjetoDraw(modelBaneraBase, glm::vec3(1.4f, 0.2f, 2.1f),
glm::vec3(0.0f, 0.6f, 0.0f), modelLoc);
                ObjetoDraw(modelBaneraBase, glm::vec3(1.4f, 0.2f, 2.0f),
glm::vec3(0.0f, 0.4f, 0.0f), modelLoc);
                //Patas
                ObjetoDraw(modelBaneraBase, glm::vec3(0.1f, 0.4f, 0.1f),
glm::vec3(0.50f, 0.2f, 0.9f), modelLoc);
                ObjetoDraw(modelBaneraBase, glm::vec3(0.1f, 0.4f, 0.1f),
glm::vec3(0.5f, 0.2f, -0.9f), modelLoc);
                ObjetoDraw(modelBaneraBase, glm::vec3(0.1f, 0.4f, 0.1f),
glm::vec3(-0.5f, 0.2f, -0.9f), modelLoc);
                ObjetoDraw(modelBaneraBase, glm::vec3(0.1f, 0.4f, 0.1f),
glm::vec3(-0.5f, 0.2f, 0.9f), modelLoc);

                glBindTexture(GL_TEXTURE_2D, 0);// Desvincula la textura

                // Dibujar la planta
                glBindVertexArray(VAO);//Activa el VAO del cubo
                glActiveTexture(GL_TEXTURE0);//Activa su textura en 0
                glBindTexture(GL_TEXTURE_2D, plantaTexture);// Usamos la
textura del baño
                glUniform2f(tilingLoc, 1.0f, 1.0f); // Tiling 1x1
```

```
                //Ubicacion de la planta
                glm::mat4 modelPlantaBase =
glm::translate(glm::mat4(1.0f), glm::vec3(-2.5f, 2.0f, -2.75f));

                // Repisa
                ObjetoDraw(modelPlantaBase, glm::vec3(1.2f, 0.1f, 0.4f),
glm::vec3(0.0f, 0.05f, 0.0f), modelLoc);
                //Maceta
                ObjetoDraw(modelPlantaBase, glm::vec3(0.4f, 0.4f, 0.4f),
glm::vec3(0.0f, 0.3f, 0.0f), modelLoc);
                // Tallo
                ObjetoDraw(modelPlantaBase, glm::vec3(0.1f, 0.5f, 0.1f),
glm::vec3(0.0f, 0.7f, 0.0f), modelLoc);
                // Hoja 1
                glm::mat4 modelHoja1 = modelPlantaBase;
                //Ubicacion de la hoja
                modelHoja1 = glm::translate(modelHoja1, glm::vec3(0.150f,
0.9f, 0.0f));
                //Rotacion del cubo
                modelHoja1 = glm::rotate(modelHoja1, glm::radians(30.0f),
glm::vec3(0.0f, 0.0f, 1.0f));
                ObjetoDraw(modelHoja1, glm::vec3(0.5f, 0.05f, 0.3f),
glm::vec3(0.0f, 0.0f, 0.0f), modelLoc);
                // Hoja 2
                glm::mat4 modelHoja2 = modelPlantaBase;
                //Ubicacion de la hoja
                modelHoja2 = glm::translate(modelHoja2, glm::vec3(-0.150f,
0.9f, 0.0f));
                //Rotacion del cubo
                modelHoja2 = glm::rotate(modelHoja2, glm::radians(-30.0f),
glm::vec3(0.0f, 0.0f, 1.0f));
                ObjetoDraw(modelHoja2, glm::vec3(0.5f, 0.05f, 0.3f),
glm::vec3(0.0f, 0.0f, 0.0f), modelLoc);
                // Hoja 3
                glm::mat4 modelHoja3 = modelPlantaBase;
                //Ubicacion de la hoja
                modelHoja3 = glm::translate(modelHoja3, glm::vec3(0.0f,
0.8f, 0.0f));
                //Rotacion del cubo
                modelHoja3 = glm::rotate(modelHoja3, glm::radians(90.0f),
glm::vec3(0.0f, 1.0f, 0.0f));
                ObjetoDraw(modelHoja3, glm::vec3(0.5f, 0.05f, 0.3f),
glm::vec3(0.0f, 0.0f, 0.0f), modelLoc);

                glBindTexture(GL_TEXTURE_2D, 0);// Desvincula la textura


                // Calcula la Matriz de Lámpara Tambaleante
                //Posicion de la lampara
                glm::vec3 lampBaseWorldPos = glm::vec3(1.5f, 0.0f, -0.9f);
                //Creacion de la matriz de la lampara
                glm::mat4 modelLampBase = glm::translate(glm::mat4(1.0f),
lampBaseWorldPos);
```

```
                //Aplica la rotacion de la lampara generando una especie
de tambaleo
                modelLampBase = glm::rotate(modelLampBase,
glm::radians(lampWobble), glm::vec3(0.0f, 0.0f, 1.0f));

                // Configuracion de la Luz de la Lámpara (pointLights[1])
                // Se coloca la posicion de la bombilla a la base de la
lampara
                glm::vec3 lampBulbLocalPos = glm::vec3(0.0f, 2.8f, 0.0f);
                //Posicion final de la luz aplicada en el mundo
                glm::vec3 finalLightPos = modelLampBase *
glm::vec4(lampBulbLocalPos, 1.0f);

                //Envio de posicion al shader para realizar la animacion
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].position"), finalLightPos.x, finalLightPos.y,
finalLightPos.z);

                //interruptor para determinar si la lampara se encuentra
apagada o encendida
                if (lampLightActive)
                {
                        // Luz ENCENDIDA (Amarilla)

        glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].ambient"), 0.2f, 0.2f, 0.0f);

        glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].diffuse"), 0.8f, 0.8f, 0.0f);

        glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].specular"), 1.0f, 1.0f, 1.0f);
                }
                else
                {
                        // Luz APAGADA (Negro)

        glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].ambient"), 0.0f, 0.0f, 0.0f);

        glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].diffuse"), 0.0f, 0.0f, 0.0f);

        glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].specular"), 0.0f, 0.0f, 0.0f);
                }
                // La atenuación se envía siempre
                glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].constant"), 1.0f);
                glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].linear"), 0.09f);
                glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].quadratic"), 0.032f);
```

```
                // Dibujo de la Lampara
                lightingShader.Use();
                glBindVertexArray(VAO);//Activa el VAO del cubo
                glActiveTexture(GL_TEXTURE0);//Activa su textura en 0
                glBindTexture(GL_TEXTURE_2D, lampBaseTexture);// Vincula
la textura de la lampara
                glUniform2f(tilingLoc, 1.0f, 1.0f);

                ObjetoDraw(modelLampBase, glm::vec3(0.8f, 0.1f, 0.8f),
glm::vec3(0.0f, 0.05f, 0.0f), modelLoc);//Base de la lampara
                ObjetoDraw(modelLampBase, glm::vec3(0.1f, 1.5f, 0.1f),
glm::vec3(0.0f, 0.8f, 0.0f), modelLoc);//Primer palo de la lampara
                ObjetoDraw(modelLampBase, glm::vec3(1.0f, 0.1f, 1.0f),
glm::vec3(0.0f, 1.6f, 0.0f), modelLoc);//plato de la lampara
                ObjetoDraw(modelLampBase, glm::vec3(0.5f, 0.05f, 0.5f),
glm::vec3(0.0f, 1.65f, 0.0f), modelLoc);//Segundo plato de la lampara
                ObjetoDraw(modelLampBase, glm::vec3(0.1f, 1.0f, 0.1f),
glm::vec3(0.0f, 2.1f, 0.0f), modelLoc);//Segundo palo de la lampara

                glBindTexture(GL_TEXTURE_2D, lampShadeTexture);// Vincula
la textura de la lampara para la parte de la tela
                glUniform2f(tilingLoc, 1.0f, 1.0f);
                //Dibujo de la lampara
                ObjetoDraw(modelLampBase, glm::vec3(1.0f, 0.2f, 1.0f),
glm::vec3(0.0f, 2.6f, 0.0f), modelLoc);
                ObjetoDraw(modelLampBase, glm::vec3(0.9f, 0.2f, 0.9f),
glm::vec3(0.0f, 2.8f, 0.0f), modelLoc);
                ObjetoDraw(modelLampBase, glm::vec3(0.8f, 0.2f, 0.8f),
glm::vec3(0.0f, 3.0f, 0.0f), modelLoc);
                ObjetoDraw(modelLampBase, glm::vec3(0.7f, 0.2f, 0.7f),
glm::vec3(0.0f, 3.2f, 0.0f), modelLoc);
                ObjetoDraw(modelLampBase, glm::vec3(0.6f, 0.2f, 0.6f),
glm::vec3(0.0f, 3.4f, 0.0f), modelLoc);
                glBindTexture(GL_TEXTURE_2D, 0);// Desvincula la textura


                // DIBUJAR LA TV Y ANIMACIÓN COMPLEJA
                glBindVertexArray(VAO);//Activa el VAO del cubo
                glActiveTexture(GL_TEXTURE0);//Activa su textura en 0

                // Dibujar la carcasa de la TV
                //Ubicacion
                glm::mat4 modelTVBase = glm::translate(glm::mat4(1.0f),
glm::vec3(-1.5f, 0.5f, 3.0f));
                //Rotacion
                modelTVBase = glm::rotate(modelTVBase,
glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
                glBindTexture(GL_TEXTURE_2D, tvCasingTexture);// Usamos la
textura del baño
                glUniform2f(tilingLoc, 2.0f, 1.5f);//Ajuste de dimensiones
por textura
                //Cuerpo Tv
```

```
                ObjetoDraw(modelTVBase, glm::vec3(2.0f, 1.5f, 1.0f),
glm::vec3(0.0f, 0.75f, 0.0f), modelLoc);
                ObjetoDraw(modelTVBase, glm::vec3(1.5f, 1.0f, 0.5f),
glm::vec3(0.0f, 0.75f, -0.7f), modelLoc);
                glUniform2f(tilingLoc, 0.1f, 0.5f);//Ajuste de dimensiones
por textura

                //Patas
                ObjetoDraw(modelTVBase, glm::vec3(0.1f, 0.5f, 0.1f),
glm::vec3(-0.9f, -0.25f, 0.4f), modelLoc);
                glUniform2f(tilingLoc, 0.1f, 0.5f);//Ajuste de dimensiones
por textura
                ObjetoDraw(modelTVBase, glm::vec3(0.1f, 0.5f, 0.1f),
glm::vec3(0.9f, -0.25f, 0.4f), modelLoc);
                glUniform2f(tilingLoc, 0.1f, 0.5f);//Ajuste de dimensiones
por textura
                ObjetoDraw(modelTVBase, glm::vec3(0.1f, 0.5f, 0.1f),
glm::vec3(-0.9f, -0.25f, -0.4f), modelLoc);
                glUniform2f(tilingLoc, 0.1f, 0.5f);//Ajuste de dimensiones
por textura
                ObjetoDraw(modelTVBase, glm::vec3(0.1f, 0.5f, 0.1f),
glm::vec3(0.9f, -0.25f, -0.4f), modelLoc);
                glUniform2f(tilingLoc, 0.15f, 0.15f);//Ajuste de
dimensiones por textura
                //Boton
                ObjetoDraw(modelTVBase, glm::vec3(0.15f, 0.15f, 0.1f),
glm::vec3(0.9f, 0.15f, 0.55f), modelLoc);

                // Dibujar la pantalla de TV dinamica
                if (tvAnimationActive)
                {
                        // Usa la textura del portal
                        glBindTexture(GL_TEXTURE_2D, tvStaticTexture);
                }
                else
                {
                        // Usa la textura normal de pantalla apagada
                        glBindTexture(GL_TEXTURE_2D, tvScreenTexture);
                }
                glUniform2f(tilingLoc, 1.0f, 1.0f);//Ajusta la textura
                ObjetoDraw(modelTVBase, glm::vec3(1.6f, 1.1f, 0.1f),
glm::vec3(0.0f, 0.75f, 0.51f), modelLoc);
                glBindTexture(GL_TEXTURE_2D, 0);//Dedvincula la textura

                // Animacion compleja salida del espectro
                if (tvAnimationActive)
                {
                        // Calcula la animacion no lineal
                        float emergeZ = 0.4f + (tvAnimTime *
0.5f);//Movimiento lineal el espectro avanza en Z
                        float wiggleX = sin(tvAnimTime * 8.0f) *
0.3f;//Movimiento sinusoidal, el espectro se tambalea en el eje de X
```

```
                        // Crear la matriz base del espectro
                        glm::mat4 modelFantasma = modelTVBase; // Parte de
la TV
                        modelFantasma = glm::translate(modelFantasma,
glm::vec3(wiggleX, 0.85f, emergeZ)); // Aplica animacion
                        modelFantasma = glm::rotate(modelFantasma,
glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Rotacion, para
ajustar la posicion del fantasma
                        modelFantasma = glm::scale(modelFantasma,
glm::vec3(0.28f));//Tamaño del espectro

                        // Dibujar el modelo .obj
                        GLint modelLoc =
glGetUniformLocation(lightingShader.Program, "model");
                        //Se le aplica la iluminacion de la escena al
fantasma
                        glUniformMatrix4fv(modelLoc, 1, GL_FALSE,
glm::value_ptr(modelFantasma));

                        // Reseteo del Tiling a 1x1 para el fantasma
                        GLint tilingLoc =
glGetUniformLocation(lightingShader.Program, "texTiling");
                        glUniform2f(tilingLoc, 1.0f, 1.0f);

                        // Dibujo del fantasma con el lightingShader
                        Fantasma.Draw(lightingShader);
                }

                //------Animacion compleja Billy caminando
                //Activacion del shader para la animacion
                animShader.Use();

                //Se obtienen las ubicaiones uniform para el shader
                modelLoc = glGetUniformLocation(animShader.Program,
"model");
                viewLoc = glGetUniformLocation(animShader.Program,
"view");
                projLoc = glGetUniformLocation(animShader.Program,
"projection");

                //Se envían las matrices de camara y proyeccion
                glUniformMatrix4fv(viewLoc, 1, GL_FALSE,
glm::value_ptr(view));
                glUniformMatrix4fv(projLoc, 1, GL_FALSE,
glm::value_ptr(projection));

                //Configuracion de la matriz para el modelo de Billy
                model = glm::mat4(1.0f);
                model = glm::translate(model, glm::vec3(-5.0f, 0.0f,
15.0f));//Ajuste de ubicacion
                model = glm::scale(model, glm::vec3(1.8f)); // Escala del
personaje
```

```
                glUniformMatrix4fv(modelLoc, 1, GL_FALSE,
glm::value_ptr(model));

                //Dibuja el personaje animado
                animacionPersonaje.Draw(animShader);
                glBindVertexArray(0);//Desvincula el VAO

                //// Swap the screen buffers
                glfwSwapBuffers(window);
        }

        // Terminate GLFW
        glfwTerminate();
        return 0;//Finaliza el programa con exito
}

// DoMovement
void DoMovement()
{
        // Controles de Cámara
        if (keys[GLFW_KEY_W] || keys[GLFW_KEY_UP])//Movimiento hacia
adelante
        {
                camera.ProcessKeyboard(FORWARD, deltaTime);
        }
        if (keys[GLFW_KEY_S] || keys[GLFW_KEY_DOWN])//Movimiento hacia
atras
        {
                camera.ProcessKeyboard(BACKWARD, deltaTime);
        }
        if (keys[GLFW_KEY_A] || keys[GLFW_KEY_LEFT])//Movimiento hacia la
izquierda
        {
                camera.ProcessKeyboard(LEFT, deltaTime);
        }
        if (keys[GLFW_KEY_D] || keys[GLFW_KEY_RIGHT])//Movimiento hacia la
derecha
        {
                camera.ProcessKeyboard(RIGHT, deltaTime);
        }

}
//Funcion para control de las animaciones
void KeyCallback(GLFWwindow* window, int key, int scancode, int action,
int mode)
{
        // Controles de Puertas
        if (action == GLFW_PRESS)
        {
                if (key == GLFW_KEY_1)
                        doorE_open = !doorE_open; // apertura de la puerta
de la entrada
```

```
		if (key == GLFW_KEY_2)
			doorS_open = !doorS_open; // apertura de la puerta
de la sala

		if (key == GLFW_KEY_3)
			doorB_open = !doorB_open; // apertura de la puerta
del baño
	}

	// Control de Animación de Lámpara
	if (key == GLFW_KEY_M && action == GLFW_PRESS)
	{
		lampAnimationActive = !lampAnimationActive; // Movimiento
de la lampara
	}

	// Control de Luz de Lámpara
	if (key == GLFW_KEY_N && action == GLFW_PRESS)
	{
		lampLightActive = !lampLightActive; // Interruptor de la
LUZ de la lampara
	}

	// Control Animacion Compleja TV ---
	if (key == GLFW_KEY_P && action == GLFW_PRESS)
	{
		if (!tvAnimationActive)//Solo inicia mientras la tv no
este activa
		{
			tvAnimationActive = true;//Activa la bandera
			tvAnimTime = 0.0f; // Reinicia el tiempo de la
animación
		}
	}

	// Cerrar ventana
	if (GLFW_KEY_ESCAPE == key && GLFW_PRESS == action)
	{
		glfwSetWindowShouldClose(window, GL_TRUE);//Señal de
cierre de ventana
	}

	// Actualizar array de teclas
	if (key >= 0 && key < 1024)
	{
		if (action == GLFW_PRESS)
		{
			keys[key] = true;//Marca la tecla como presionada
		}
		else if (action == GLFW_RELEASE)
		{
			keys[key] = false;//Marca la tecla como soltada
		}
```

```
        }
        // Control de Luz de Debug
        if (key == GLFW_KEY_SPACE && action == GLFW_PRESS)
        {
                active = !active;
                if (active)
                {
                        Light1 = glm::vec3(0.2f, 0.8f, 1.0f);
                }
                else
                {
                        Light1 = glm::vec3(0);
                }
        }
}

void Animation() {

        // Animación de Puertas
        float doorSpeed = 1.5f; // velocidad en grados por frame

        // Puerta Entrada
        if (doorE_open && doorE_angle < 90.0f)
                doorE_angle += doorSpeed;//Abrir
        else if (!doorE_open && doorE_angle > 0.0f)
                doorE_angle -= doorSpeed;//Cerrar

        // Puerta Salida
        if (doorS_open && doorS_angle < 90.0f)
                doorS_angle += doorSpeed;//Abrir
        else if (!doorS_open && doorS_angle > 0.0f)
                doorS_angle -= doorSpeed;//Cerrar

        // Puerta Baño
        if (doorB_open && doorB_angle < 90.0f)
                doorB_angle += doorSpeed;//Abrir
        else if (!doorB_open && doorB_angle > 0.0f)
                doorB_angle -= doorSpeed;//Cerrar

        // Animación Simple de Lámpara
        if (lampAnimationActive)
        {
                lampWobble = sin((float)glfwGetTime() * 2.0f) *
5.0f;//Angulo de tambaleo de la lampara
        }
        else
        {
                lampWobble = 0.0f;//En caso de estar activa resetea
        }

        // --- NUEVO: Actualización de Animación Compleja TV ---
        if (tvAnimationActive)
        {
```

```
                tvAnimTime += deltaTime; // Incrementa el temporizador

                // Detener la animación después de 5 segundos
                if (tvAnimTime > 5.0f)
                {
                        tvAnimationActive = false;//Desactiva la bandera
                        tvAnimTime = 0.0f; //Resetea el temporizador
                }
        }
}

// MouseCallback
void MouseCallback(GLFWwindow* window, double xPos, double yPos)
{
        if (firstMouse)
        {
                lastX = (float)xPos;//Establece la posicion inicial en X
                lastY = (float)yPos;//Establece la posicion inicial en Y
                firstMouse = false;//Desactiva la bandera
        }
        GLfloat xOffset = (float)xPos - lastX;//Calculo del movimiento del
mouse
        GLfloat yOffset = lastY - (float)yPos;//Se invierte el eje Y
        lastX = (float)xPos;//Almacena la posicion para el proximo
fotograma
        lastY = (float)yPos;
        camera.ProcessMouseMovement(xOffset, yOffset);//Envia la posicion
de la camara para calcular los nuevos angulos
}

// Función para dibujar primitivas de cubo
void ObjetoDraw(glm::mat4 base, glm::vec3 escala, glm::vec3 traslado,
GLint uniformModel)
{
        glm::mat4 modelo = glm::mat4(1.0f);//Inicializa la matriz
        modelo = glm::translate(modelo, traslado);//Aplica la posicion de
cada pieza
        modelo = glm::scale(modelo, escala);//Ajusta el tamaño de cada
pieza
        modelo = base * modelo;
        glUniformMatrix4fv(uniformModel, 1, GL_FALSE,
glm::value_ptr(modelo));//Envia la matriz al shader
        glDrawArrays(GL_TRIANGLES, 0, 36);//Dibuja el cubo con 36 vertices
definidos en el VAO
}
```