

**UNIVERSIDAD VERACRUZANA**  
**FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA**

**ALUMNO**

Raziel de Jesús Rodríguez  
Pérez

**MATRICULA**

zS21020229

**EXPERIENCIA EDUCATIVA**

Graficación por Computadora

**FACILITADOR**

Yulian Berumen Diaz

**ACTIVIDAD**

Documentación Técnica

**Boca del Rio, Ver. A 9 de Diciembre del 2025**

# INDICE

<b>INTRODUCCION .....</b>	<b>3</b>
<b>REQUISITOS .....</b>	<b>4</b>
<b>1. Requisitos Funcionales .....</b>	<b>4</b>
<b>2. Requisitos No Funcionales .....</b>	<b>6</b>
<b>DISEÑO .....</b>	<b>8</b>
<b>Arquitectura utilizada .....</b>	<b>8</b>
<b>Algoritmos y estructuras de datos utilizados .....</b>	<b>9</b>
<b>DIAGRAMAS .....</b>	<b>11</b>
<b>IMPLEMENTACIÓN .....</b>	<b>15</b>
<b>RESULTADOS .....</b>	<b>24</b>
<b>MANUAL DE USUARIO .....</b>	<b>26</b>
<b>CONCLUSION .....</b>	<b>29</b>
<b>REFERENCIAS .....</b>	<b>30</b>

# INTRODUCCION

El presente proyecto tiene como propósito aplicar de manera práctica los conocimientos adquiridos en la materia de Graficación por Computadora, desarrollando una animación completa en C con OpenGL. A lo largo del semestre se estudiaron conceptos fundamentales como modelado 3D, transformaciones geométricas, iluminación, texturizado y animación mediante estructuras de datos; este proyecto reúne todos esos elementos en una mini-película original protagonizada por un personaje propio.

La animación, titulada **FIEE: El Mensajero**, cuenta una breve historia en cinco escenas, donde un pequeño robot atraviesa distintos entornos y obstáculos para cumplir con una misión. Cada parte del proyecto, desde el modelado de escenarios hasta la lógica de animación por frames, fue construida con técnicas vistas en clase, integrando también estructuras de datos como listas, colas y árboles para gestionar la secuencia de escenas, la interpolación de movimiento y la jerarquía del personaje.

## Objetivos y Alcance

El desarrollo de este proyecto tiene como objetivos principales:

- Implementar de forma práctica los conceptos de graficación trabajados durante el curso, empleando OpenGL para construir una animación 3D funcional y visualmente coherente.
- Diseñar y modelar un personaje original, aplicando texturas, iluminación, articulaciones y un sistema de animación basado en keyframes.
- Crear una mini-película de entre 2 y 3 minutos, compuesta por múltiples escenas que se reproducen mediante una cola de escenas y se controlan en tiempo real.
- Aplicar estructuras de datos listas para los frames, colas para las escenas, pilas para acciones y árboles para la estructura jerárquica del robot con el fin de organizar y gestionar la animación.
- Incorporar interacción del usuario mediante menú y controles de teclado/mouse para pausar, reanudar o reiniciar la reproducción.

El alcance del proyecto abarca la construcción del personaje, el modelado de escenarios, el sistema de animación completo, la cámara dinámica, la implementación de iluminación y texturas, así como la integración de todos estos elementos dentro de una narrativa visual continua.

## REQUISITOS

### 1. Requisitos Funcionales

Los siguientes requisitos reflejan directamente lo que implementa este proyecto de animación 3D en OpenGL:

#### **RF1. Reproducción completa de la mini-película**

El sistema debe reproducir en tiempo real la animación de FIEE a lo largo de sus cinco escenas: *El Despertar*, *La Misión*, *El Viaje*, *La Entrega* y *El Propósito*. La transición entre escenas debe realizarse automáticamente utilizando la cola de escenas definida en el programa.

#### **RF2. Control de la animación por parte del usuario**

El usuario debe poder pausar, reanudar y reiniciar la animación ya sea desde el menú principal o mediante los controles de teclado/mouse definidos en *main.c*. Esto permite detener escenas, continuar la narrativa o iniciar desde cero en cualquier momento.

#### **RF3. Gestión de escenas con una estructura de cola**

El proyecto implementa una cola que almacena el orden de reproducción de las escenas. Cada vez que una escena termina, el sistema desencola la siguiente y actualiza *escenaActual*, garantizando un flujo lineal y ordenado de la historia.

#### **RF4. Animación del robot FIEE mediante una lista de frames**

Cada movimiento del robot incluyendo caminar, girar, saltar, agacharse, levantar brazos, etc. se genera a partir de listas enlazadas de FrameCompleto. El programa interpola entre frames para suavizar la animación y actualiza las articulaciones del árbol jerárquico del personaje.

#### **RF5. Representación jerárquica del personaje con un árbol de nodos**

FIEE está construido mediante un árbol jerárquico donde cada parte del cuerpo (cabeza, brazos, piernas, torso, mochila) es un nodo conectado. Esto permite que transformaciones como rotaciones o inclinaciones afecten correctamente a las partes dependientes.

#### **RF6. Renderizado de escenarios y objetos 3D**

Cada escena debe dibujar correctamente sus elementos: paredes, ventanas, escritorio, armario, estantería, colina, río, piedras de salto, casa, nubes, sol, camino y cielo. Estos objetos están implementados en *objetos.c* y deben mostrarse con sus transformaciones apropiadas.

#### **RF7. Carga y aplicación de texturas**

El sistema debe cargar texturas (madera, metal, hojas, colina, mochila, ropa del robot, etc.) y aplicarlas tanto al personaje como a los objetos del entorno utilizando STB\_IMAGE.

#### **RF8. Iluminación dinámica**

Cada escena debe configurarse con iluminación distinta para reforzar su atmósfera: luz cálida en interiores, luz del sol en exteriores, iluminación nocturna, halos y rayos de luz, entre otros efectos implementados en tus funciones.

### **RF9. Cámara manipulable por el usuario**

El programa incluye funciones para mover la cámara, hacer zoom o rotarla con el mouse, permitiendo inspeccionar libremente las escenas además del movimiento automático predefinido.

## **2. Requisitos No Funcionales**

Estos requisitos describen las cualidades que debe mantener el proyecto para funcionar correctamente y verse bien.

### **RNF1. Fluidez en la animación**

Las transiciones entre frames deben sentirse naturales. La interpolación implementada en `aplicarFrameCompletoInterpolado()` debe garantizar que el movimiento de FIEE no presente saltos bruscos entre poses.

### **RNF2. Organización modular del código**

El proyecto debe mantener una estructura clara separando responsabilidades:

- *main.c*: control general de animación y cámara
- *escenas.c*: dibujo de cada escena
- *robot.c*: lógica del personaje y su animación
- *objetos.c*: modelado de elementos del entorno

Esto facilita depuración y mantenimiento.

### **RNF3. Coherencia visual en toda la narrativa**

El estilo de modelado, las texturas, los colores y la iluminación deben ser consistentes entre escenas para que la mini-película se perciba como un trabajo unificado y profesional.

### **RNF4. Eficiencia en el manejo de memoria**

Las listas de frames deben limpiarse al cargar cada escena usando `limpiarFramesCompletos()`, evitando fugas de memoria durante la reproducción.

### **RNF5. Estabilidad durante la ejecución**

El sistema debe evitar fallos incluso durante cambios de escena, carga de texturas, manipulación de cámara o reproducción de frames. El manejo correcto de punteros y estructuras dinámicas garantiza confiabilidad.

### **RNF6. Facilidad de uso**

El usuario debe poder interactuar de manera intuitiva con la animación:

- Teclas claras para pausa, reanudar, reiniciar
- Mouse para zoom o rotación
- Menú simple y accesible

### **RNF7. Portabilidad del proyecto**

El programa debe poder compilarse en cualquier entorno compatible con OpenGL, GLUT y STB\_IMAGE sin requerir configuraciones complejas adicionales.

### **RNF8. Calidad visual adecuada**

Aunque el proyecto utiliza primitivas básicas de OpenGL, debe ofrecer un acabado visual agradable mediante texturas correctas, sombras simuladas, efectos de luz y composición cinematográfica.

# DISEÑO

## Arquitectura utilizada

Para este proyecto decidí organizar todo el código en una arquitectura modular que me permitiera trabajar cada parte de la animación de manera independiente, pero manteniendo una comunicación clara entre los módulos. Esto me ayudó muchísimo a evitar que todo se mezclara en un solo archivo gigante y difícil de mantener. En esencia, dividí el programa en cuatro partes principales: el control general, el dibujo de las escenas, el modelado y animación del personaje, y la biblioteca de objetos que forman el entorno.

El archivo **main.c** funciona como el centro operativo de toda la animación. Ahí manejo la ventana principal, el menú, los controles del usuario y, sobre todo, el ciclo de tiempo que hace posible que la animación avance. Cada vez que se actualiza la pantalla, *main* calcula la posición de la cámara, revisa qué escena corresponde reproducir y llama a los módulos necesarios para dibujarla. También administro desde ahí la secuencia de escenas: cuando se termina una, paso automáticamente a la siguiente con ayuda de una estructura de cola que preparé desde el inicio.

Por otro lado, en **escenas.c** me enfoqué en el aspecto cinematográfico. Aquí dibujo el entorno de cada escena: la oficina postal, el armario, el río, las piedras de salto, la colina, la casa del final, el cielo, los árboles, las nubes, la iluminación, etc. Cada escena tiene su propia atmósfera y composición visual, y este módulo se encarga de que todo se muestre correctamente sin preocuparme por la lógica del personaje o del tiempo. Solo decido qué objetos aparecen, cómo se colocan y cuál es la iluminación apropiada para transmitir lo que FIEE vive en ese momento.

El archivo **objetos.c** funciona como una biblioteca completa de modelos reutilizables. Aquí definí todos los elementos del mundo: el escritorio, el armario, la estantería, las paredes, el río, la colina, las rocas, la casa y muchos más. De esta forma, cuando quiero usar un objeto en cualquier escena, simplemente llamo a su

función de dibujo. Esto me permitió mantener un estilo visual coherente y evitar duplicar código por todas partes.

Finalmente, **robot.c** es probablemente el módulo más complejo del proyecto, porque aquí vive el personaje principal: FIEE. Implementé una estructura jerárquica que representa su cuerpo como un árbol de articulaciones: el torso es el nodo principal, y de ahí se conectan la cabeza, los brazos, las piernas, la mochila, etc. Todo se mueve de manera dependiente, de modo que si giro el torso, automáticamente se mueven las extremidades que están unidas a él. También aquí almacené todo el sistema de animación basado en frames: cada pose importante del robot se guarda como un frame completo, y luego hago interpolación entre estos para que la animación sea fluida y natural. En este módulo también cargué las texturas del robot, como la gorra, el torso, los zapatos y los ojos, que ayudan a darle personalidad.

En conjunto, esta arquitectura hace que el proyecto sea más fácil de entender, de mantener y de extender. Cada módulo sabe exactamente cuál es su trabajo, pero al mismo tiempo todos cooperan para que la animación completa funcione como una mini película.

## Algoritmos y estructuras de datos utilizados

Uno de los aspectos más importantes de este proyecto fue el uso de estructuras de datos para poder organizar la animación como lo pide el curso. No las usé solo por cumplir, sino porque realmente me ayudaron a resolver problemas concretos.

La primera estructura clave es la **lista enlazada de frames**, que utilicé para manejar la animación completa de FIEE. Cada frame guarda una pose del robot: los ángulos de la cabeza, de los brazos, de las piernas, la posición global, la rotación del cuerpo, e incluso si lleva la carta o no. En lugar de depender de animaciones automáticas, preferí definir cada postura importante y luego interpolar entre ellas.

La interpolación hace que los movimientos sean progresivos y no bruscos. Esta lista se va recorriendo frame por frame mientras avanza el tiempo, y cuando cada escena termina, limpio la lista y cargo los frames adecuados para la siguiente.

La segunda estructura fundamental es la **cola de escenas**. Como mi mini película tiene un orden narrativo definido, usé una cola para almacenar el orden exacto de las escenas: Despertar → Misión → Viaje → Entrega → Propósito. Esto me facilita desencolar la siguiente escena en cuanto la actual termina, sin tener que hacer condicionales o estructuras complicadas. Cuando reinicio la animación, simplemente vuelvo a encolar todas las escenas en su orden original.

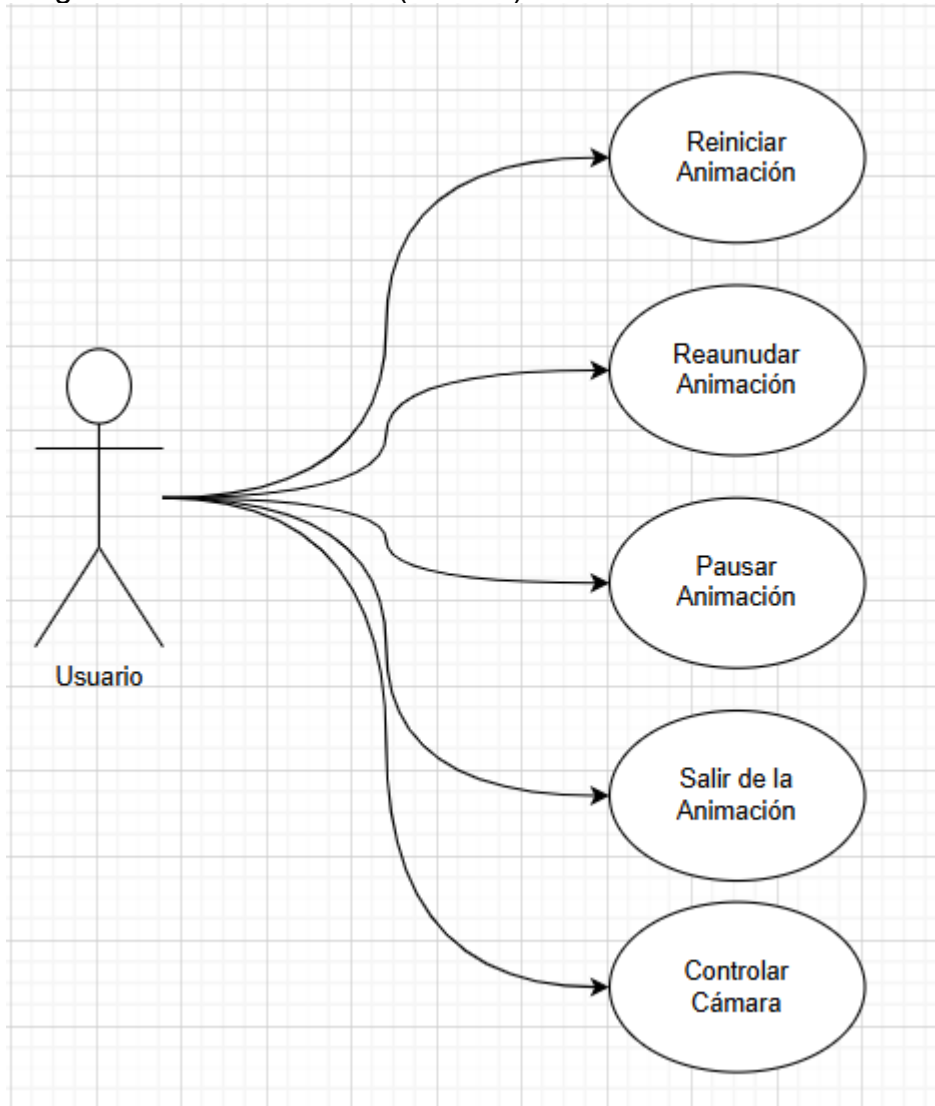
La tercera estructura es el **árbol jerárquico del robot**, que representa la estructura física de FIEE. Aquí cada parte del cuerpo es un nodo, y las transformaciones se propagan desde el torso a las extremidades. Esto es crucial para animar un personaje articulado, porque evita que cada parte se mueva de manera independiente y sin coherencia. Si el torso se inclina, la cabeza, los brazos y las piernas acompañan ese movimiento. Esta estructura también me permitió organizar las funciones de dibujo para que se renderice primero el torso, luego sus hijos, y así sucesivamente.

Además de estas estructuras, la animación utiliza un algoritmo de **interpolación lineal**, que toma dos frames y genera una transición suave dependiendo del tiempo transcurrido. Esta técnica es esencial para que FIEE no se mueva “cuadro por cuadro”, sino de manera fluida como si realmente estuviera caminando, saltando o girando.

En general, todas estas estructuras trabajan juntas: la lista de frames controla el movimiento del robot, la cola controla el orden de la historia, y el árbol jerárquico controla cómo se mueve físicamente cada parte del personaje.

# DIAGRAMAS

Diagrama de casos de uso (usuario)



## Descripción de los Casos de Uso

- ✓ **Reproducir animación**
  - El usuario selecciona "Reproducir".
  - El sistema carga escenas y frames desde cero.
  - La animación inicia.

✓ **Pausar animación**

- El usuario presiona Espacio o clic en el menú.
- El sistema detiene el avance del tiempo.

✓ **Reanudar animación**

- El usuario vuelve a presionar Espacio.
- La animación continúa desde donde se pausó.

✓ **Reiniciar animación**

- El usuario presiona "R".
- Vuelve a cargar la escena 1 y reinicia tiempos y frames.

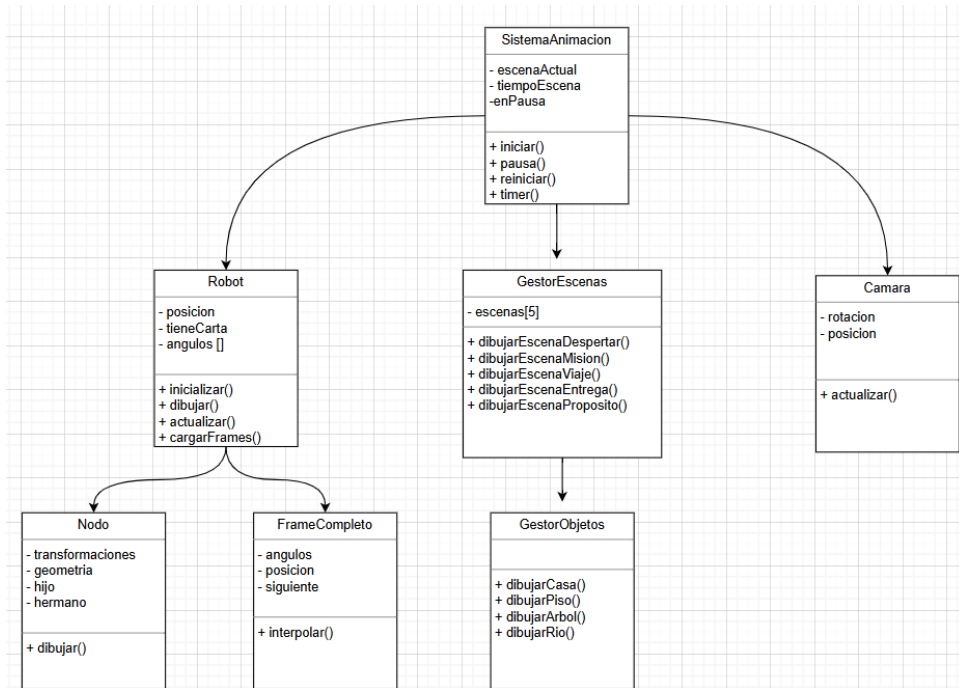
✓ **Controlar cámara**

- El usuario usa el mouse para rotar la vista o hacer zoom.
- Puede activar el modo debug para tener más control.

✓ **Salir del programa**

- El usuario oprime "ESC".
- Se liberan recursos y el programa finaliza.

## Diagrama de clases UML



### SistemaAnimacion (main.c)

Clase principal que coordina toda la aplicación. Gestiona el ciclo de vida de la animación, controla el estado de reproducción (play/pausa), maneja eventos de teclado y mouse, y coordina las transiciones entre escenas. Implementa el bucle principal con `glutMainLoop()`.

### GestorEscenas (escenas.c)

Responsable de renderizar cada una de las 5 escenas del viaje de FIEE:

- Despertar: Robot despierta en oficina postal abandonada
- Misión: Encuentra la carta en el armario
- Viaje: Cruza el río y escala la colina
- Entrega: Llega a la casa y entrega la carta
- Propósito: Se aleja satisfecho hacia el horizonte

Cada escena incluye iluminación específica, objetos del entorno y configuración de cámara.

## **Robot (robot.c)**

Controla el personaje principal (FIEE). Implementa un sistema de animación basado en keyframes con interpolación suave. Gestiona las texturas del robot, la carta que lleva en la mano, y coordina la estructura jerárquica de nodos para brazos, piernas, cabeza y torso. Carga frames específicos para cada escena.

## **Nodo**

Implementa la estructura jerárquica del robot mediante un árbol de transformaciones. Cada nodo representa una parte del cuerpo (cabeza, torso, brazos, piernas) con sus propias transformaciones locales (traslación, rotación, escala). Permite animación articulada mediante relaciones padre-hijo.

## **FrameCompleto**

Estructura de datos que almacena un keyframe completo de animación. Contiene todos los ángulos de articulaciones (hombros, codos, caderas, rodillas), posición global del robot y estado de la carta. Forma una lista enlazada para secuencias de animación con interpolación lineal entre frames.

## **GestorObjetos (objetos.c)**

Biblioteca de funciones para dibujar elementos del entorno 3D. Incluye objetos arquitectónicos (pisos, paredes, escritorio, casa), elementos naturales (árboles, arbustos, rocas, río, colinas, nubes, estrellas) y efectos especiales (rayos de sol, telarañas). Maneja texturas para realismo visual.

## **Cámara**

Sistema de visualización con dos modos:

- Modo Producción: Cámaras fijas predefinidas para cada escena
- Modo Debug: Control libre con mouse para desarrollo (rotación, zoom, pan)

Permite alternar entre modos con la tecla 'D' y mostrar información de posición para configurar nuevas cámaras.

# IMPLEMENTACIÓN

Para implementar mi mini-película en OpenGL, organicé el proyecto en varios módulos en C: main.c, escenas.c, objetos.c y robot.c. Cada archivo tiene una responsabilidad clara: control general del programa, construcción de escenas, biblioteca de modelos y sistema de animación del personaje. Todo está pensado para cumplir con lo que pide el proyecto final: uso de OpenGL, texturas, iluminación, cámara, estructuras de datos y un sistema de escenas animadas en tiempo real.

## 1. Librerías que utilizo y por qué

Desde el código, las librerías base que uso son:

- **OpenGL + GLUT + GLU**

En robot.c incluyo <GL/glut.h>, y también uso GLU para las primitivas avanzadas como cilindros y esferas con normales y texturas.

Con esto manejo:

- Dibujado de primitivas 3D (glutSolidCube, glutSolidSphere, cilindros con gluCylinder y esferas con gluSphere).
- Transformaciones (glTranslatef, glRotatef, glScalef) en todos los objetos y partes del robot.
- Iluminación básica (glLightfv, glEnable(GL\_LIGHTING)) que voy configurando distinto en cada escena para darle atmósfera (oficina oscura, viaje al atardecer, noche estrellada, etc.).
- El loop principal y callbacks de ventana, menú, teclado y mouse a través de GLUT, en main.c.

- **STB\_IMAGE**

Implemento #define STB\_IMAGE\_IMPLEMENTATION y uso stb\_image.h dentro de robot.c para cargar imágenes y convertirlas en texturas OpenGL. Específicamente, hago una función cargar\_textura(const char\* archivo) donde:

- Leo el archivo de imagen con stbi\_load.
- Genero un GLuint con glGenTextures.

- Cargo los píxeles con `glTexImage2D` usando formato RGBA.
- Configuro filtros de min/mag y modo de repetición (`GL_LINEAR`, `GL_REPEAT`), y libero la memoria de la imagen. Esta función la reutilizo para texturas de gorra, torso, mochila, zapatos, metal del cuerpo y ojos del robot.
- **Librerías estándar de C**  
Uso `<math.h>`, `<stdio.h>` y funciones de memoria dinámicas (`malloc`, `free`) para:
  - Manejar cálculos de interpolación y ángulos.
  - Mensajes de depuración en consola (por ejemplo cuando cargo una textura o cuando cambio de frame).
  - Reservar memoria para los nodos del árbol del robot y para la lista enlazada de frames.

## 2. Herramientas y entorno

A nivel herramientas, el proyecto se compila con un toolchain de C compatible con OpenGL (por ejemplo MinGW/MSYS2 en Windows) y el ejecutable abre dos ventanas OpenGL: una para el menú y otra para la animación principal. Esa configuración la hago en `main`, al crear `ventanaPrincipal` y la ventana del menú, activando también `GL_DEPTH_TEST`, `GL_TEXTURE_2D` y la iluminación global tipo “cartoon”.

## 3. Explicación del código por módulos

### 3.1. `main.c` – Control general, ventanas, cámara y menú

En `main.c` concentro la lógica de alto nivel: inicio GLUT, configuro los modos de display, creo las ventanas, habilito profundidad, texturas y la iluminación principal, y registro todos los callbacks: `display`, `displayMenu`, `keyboard`, `mouseClick`, `mouseMotion`, etc.

## Menú principal

Tengo una función `displayMenu()` donde cambio a una proyección ortográfica (`gluOrtho2D`) y dibujo el menú en 2D: título, estado (PAUSADO / REPRODUCIENDO) y cuatro botones: REPRODUCIR, PAUSAR, REINICIAR y SALIR.

Para eso uso:

- `dibujarTextoCentrado(x, y, texto)` para escribir texto en pantalla, calculando el ancho aproximado de la cadena para centrarlo.
- `dibujarBoton(...)` que dibuja un rectángulo con `GL_QUADS`, un borde en `GL_LINE_LOOP` y el texto encima. Cuando el botón está seleccionado, cambio el color para resaltarlo.

Con el mouse y el teclado, cambio `botonSeleccionado` y, al hacer clic, ejecuto las acciones:

- Reproducir/Reanudar: activo `animacionIniciada` y quito la pausa.
- Pausar: activo `enPausa`.
- Reiniciar: llamo a `reiniciarAnimacion()`.
- Salir: destruyo las ventanas y llamo a `exit(0)`.

## Ventana principal y cámara

En la ventana principal, el `display()` limpia color y profundidad, prepara la matriz de vista y coloca la cámara. Cuando tengo modo debug de cámara activo, la posición y ángulos de la cámara se ajustan con el mouse (rotación con `mouseMotion` y zoom con el scroll del mouse dentro de `mouseClick`). Cada movimiento actualiza variables como `cameraAngleX`, `cameraAngleY` y `zoom`, y tengo una función para imprimir la posición de la cámara en consola cuando suelto el mouse.

Esto me permite ajustar las tomas cinematográficas como pide la rúbrica del proyecto: diferentes ángulos para cada escena, variando la perspectiva sin mover los objetos.

## Control del tiempo y escenas

En main.c también llevo el control del tiempo global de la animación mediante un timer() (no lo pegamos aquí pero está en tu archivo) que:

- Suma el delta de tiempo.
- Llama a la función de actualización del robot (actualizar\_robot) pasándole el dt y la escena actual.
- Decide cuándo cambiar de escena, apoyándose en una cola de escenas que va desencolando el siguiente valor de TipoEscena.

Además, controlo la pausa con la función togglePausa(), que solo cambia un booleano y muestra en consola si está en pausa o reanudado.

## 3.2. escenas.c – Construcción de cada escena y atmósfera visual

En escenas.c me dedico a “vestir el escenario” de cada una de las cinco escenas: Despertar, Misión, Viaje, Entrega y Propósito. Cada función dibujarEscenaX() hace tres cosas principales:

1. Configurar la iluminación específica de la escena.
2. Posicionar los objetos del entorno usando funciones de objetos.c.
3. Dibujar el robot con dibujar\_robot() en la posición correspondiente.

Ejemplo: Escena de la Misión (oficina/armario)

En dibujarEscenaMision():

- Habilito GL\_LIGHTING, GL\_LIGHT0 y GL\_LIGHT1. La primera luz es ambiental y difusa tenue (oficina oscura), y la segunda es una luz cálida que simula el brillo mágico de la carta en el armario, con atenuación.
- Dibujo piso, tablas del piso, techo y vigas, y las cuatro paredes usando funciones como dibujar\_piso, dibujar\_pared\_trasera, dibujar\_pared\_derecha, etc.

- Coloco muebles: armario, estantería, escritorio, lámpara, tablón de anuncios, reloj, grietas en el piso y telarañas en las esquinas usando varias transformaciones `glTranslatef` y `glRotatef`.
- Finalmente, dibujo a FIEE dentro de la escena con `dibujar_robot()` para que el personaje esté físicamente integrado.

### Paisajes exteriores: Viaje, Entrega y Propósito

Para las escenas exteriores (`dibujarEscenaViaje`, `dibujarEscenaEntrega`, `dibujarEscenaProposito`):

- Creo un terreno amplio con pasto usando `dibujar_cesped`, escalado para cubrir toda el área visible.
- Construyo un camino largo que atraviesa la escena con `dibujar_camino`, escalando un cubo para simular la franja de tierra que lleva a la casa.
- Distribuyo árboles, rocas y arbustos usando arreglos de coordenadas y bucles `for` que recorren las posiciones, aplican `glTranslatef` y llaman a `dibujar_arbol`, `dibujarRocaGrande` y `dibujarArbusto`.
- En la escena final de la noche (`dibujarEscenaProposito`), genero estrellas aleatorias con `generarEstrellas(...)` y las dibujo en las tres paredes de cielo (fondo, derecha e izquierda), además de usar iluminación muy tenue y una luz cálida de la casa.
- La casa la coloco con `dibujar_casa_simple` o `dibujar_casa_simple_puerta_abierta`, según la escena, ajustando su escala y posición para que combine con la narrativa (antes y después de entregar la carta).

En resumen, `escenas.c` es donde convertí el guion del robot mensajero en un espacio 3D coherente, reutilizando intensivamente los objetos que definí en `objetos.c` y combinándolos con iluminación y cámara para que cada escena tenga su personalidad visual.

### 3.3. objetos.c – Biblioteca de modelos reutilizables

objetos.c funciona como mi “catálogo” de modelos. Aquí defino las funciones que dibujan cada elemento del mundo usando primitivas de OpenGL, a veces con texturas y a veces solo con colores y transparencias.

Algunos ejemplos importantes:

- Terreno y camino
  - dibujar\_césped() escala un cubo para simular una gran plancha de pasto que sirve de base al escenario.
  - dibujar\_camino() dibuja un cubo que luego escalo en las escenas para que parezca un camino largo.
- Colina, río y piedras de salto (Escena 3)
  - dibujar\_colina() y dibujar\_colina\_empinada() usan esferas escaladas con texturas (tex\_colina, tex\_colina\_empinada) para construir volúmenes tipo montaña.
  - dibujar\_rio() crea una superficie de agua semitransparente con glEnable(GL\_BLEND) y dos cubos escalados, uno más claro arriba simulando brillo.
  - dibujar\_piedra\_salto() usa una esfera achatada con textura para las piedras donde FIEE salta.
- Vegetación y ambiente
  - dibujar\_arbol() mezcla un cilindro texturizado para el tronco y tres esferas con textura de hojas para la copa, logrando árboles sencillos pero creíbles.
  - dibujar\_nube() desactiva la iluminación y usa varias esferas blancas para formar una nube volumétrica.
  - dibujar\_sol() dibuja un sol con halo usando esferas y blending, también con iluminación desactivada para que brille por sí mismo.

- Casa de la escena final
  - `dibujar_casa_simple()` arma la casa a partir de cubos para las paredes, y triángulos para los hastiales, más techo y puerta. Otra variante (`dibujar_casa_simple_puerta_abierta`) se usa en la escena donde ya se abrió la puerta.

Este archivo me permitió no repetir lógica: una vez que tengo un `dibujar_arbol()` o `dibujar_cesped()`, solo juego con `glTranslatef`, `glRotatef` y `glScalef` desde `escenas.c` para poblar el mundo.

### 3.4. robot.c Árbol jerárquico, animación con frames y texturas del personaje

`robot.c` es el corazón del personaje FIEE. Aquí combiné estructuras de datos (árbol y lista enlazada), texturas, interpolación y lógica de escena.

-Árbol jerárquico del robot

Defino una estructura `Nodo` que representa cada parte del cuerpo: tiene traslación (`tx`, `ty`, `tz`), rotación (`rx`, `ry`, `rz`), escala (`sx`, `sy`, `sz`), un puntero a una función de dibujo y punteros a hijo y hermano para formar un árbol general.

Con `crearNodo(...)` reservo memoria y configuro las transformaciones iniciales. Con `agregarHijo(padre, hijo)` conecto los nodos formando la jerarquía: torso como nodo padre, cabeza como hijo, brazos y piernas como hijos del torso, etc.

La función `dibujarNodo(Nodo* n)`:

- Hace `glPushMatrix()`.
- Aplica traducciones, rotaciones y escalas del nodo.
- Llama a su función dibujar (por ejemplo, `dibujar_cabeza`, `dibujar_torso`, `dibujar_brazo`).
- Llama recursivamente a `dibujarNodo(n->hijo)` y al final a `dibujarNodo(n->hermano)`.

Gracias a esta estructura, si yo giro el torso, automáticamente se arrastran cabeza, brazos, piernas y mochila, como pide el requisito de usar árboles para el esqueleto del personaje.

-Lista enlazada de frames completos

Para la animación, defino FrameCompleto como una estructura que guarda:

- Ángulos de todas las articulaciones: cabeza, torso, hombros, codos, caderas y rodillas.
- Posición global del robot (pos\_x, pos\_y, pos\_z) y rotación global rot\_y.
- Un flag tiene\_carta que indica si FIEE ya está cargando la carta.
- Un puntero sig al siguiente frame, convirtiendo todo en una lista enlazada.

Uso:

- agregarFrameCompleto(FrameCompleto f) para construir la lista al cargar cada escena; recorre hasta el último y encadena el nuevo nodo.
- limpiarFramesCompletos() para liberar memoria, reiniciar la lista y dejar todo preparado para cambiar a la siguiente escena.

Esto cumple el requerimiento de manejar listas como estructura de datos para la secuencia de animación del personaje.

-Actualización e interpolación de frames

La función actualizar\_robot(float dt, int escena, float t) se encarga de avanzar la animación:

1. Si la lista está vacía, retorna (no hay nada que animar).
2. La primera vez, inicializa frameCompletoActual con el primer nodo y frameCompletoSiguiente con el segundo; también resetea tiempoFrame.
3. Incrementa tiempoFrame con dt y calcula un factor  $\alpha = \text{tiempoFrame} / \text{duracionFrame}$ . Cuando  $\alpha \geq 1.0$ , pasa al siguiente par de frames y reinicia el tiempo.

4. Según la escena, actualiza `fiee_tiene_carta`:

- A partir de cierto frame en la escena de la misión, la carta ya está en su poder.
- En escenas posteriores (viaje, entrega, propósito) mantiene la carta.

5. Llama a `aplicarFrameCompletoInterpolado(frameCompletoActual, frameCompletoSiguiente, alpha)` para obtener los ángulos finales que se aplicarán a los nodos del robot.

De esta forma, en lugar de “saltar” de pose en pose, los movimientos del robot son continuos y suaves.

-Texturas del personaje

También en `robot.c` tengo variables globales para las texturas específicas del personaje:

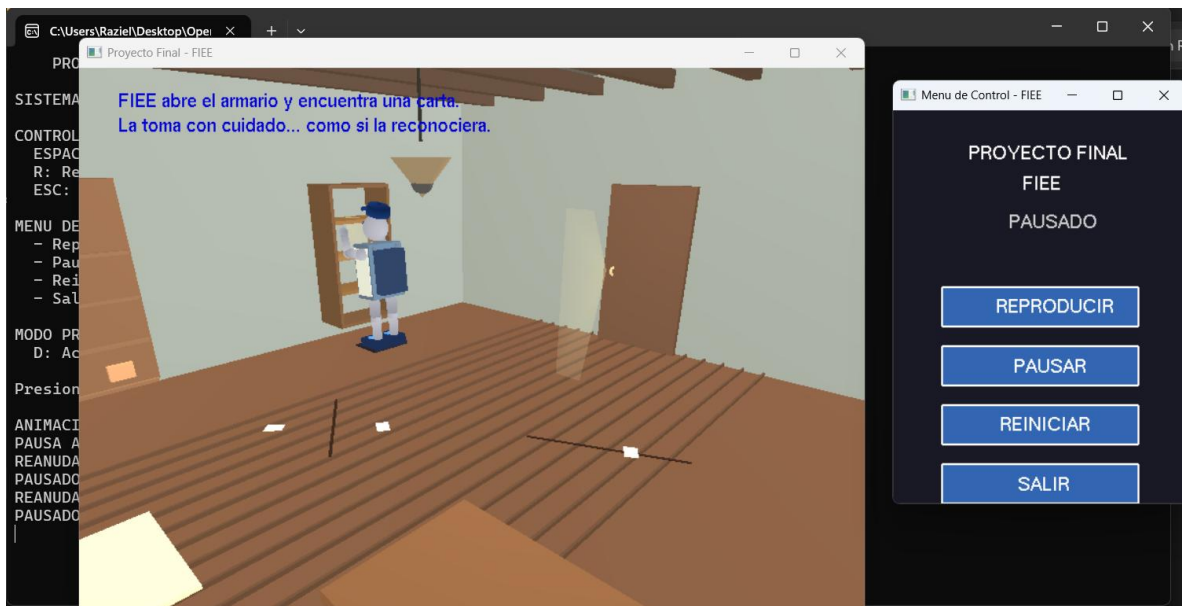
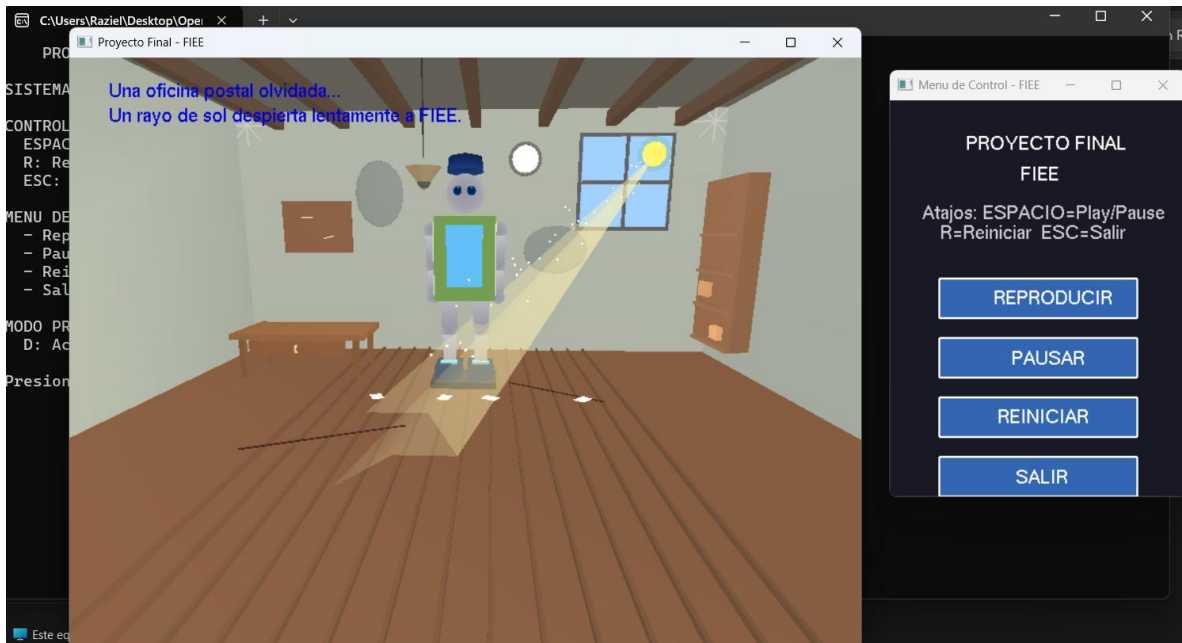
`textura_gorra`, `textura_torso`, `textura_mochila`, `textura_zapatos`, `textura_metal`, `textura_ojos`.

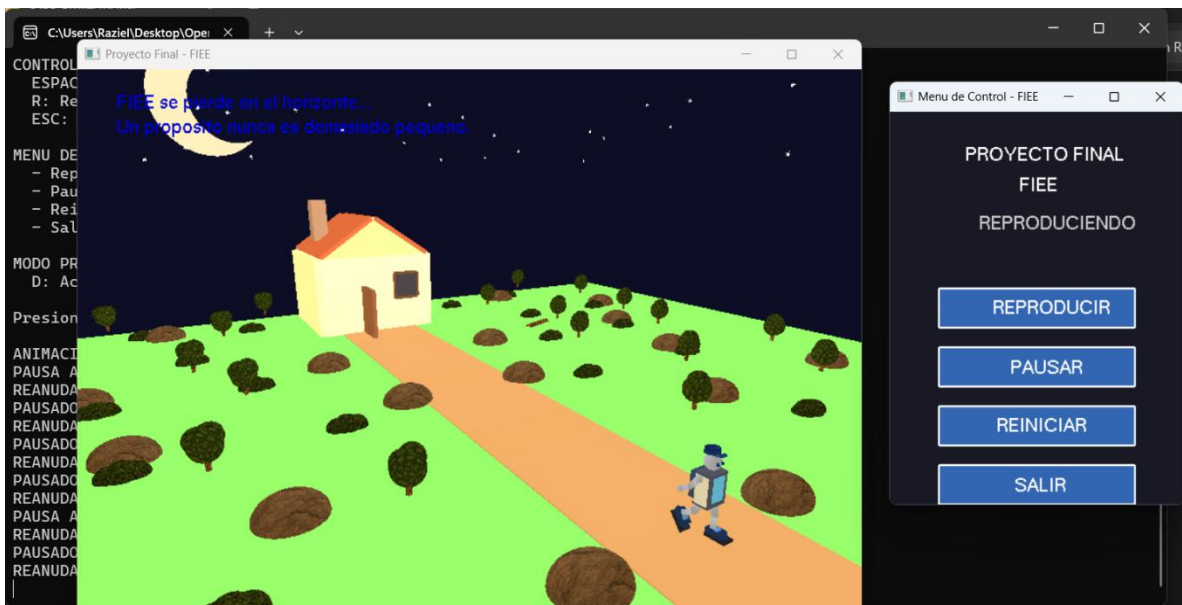
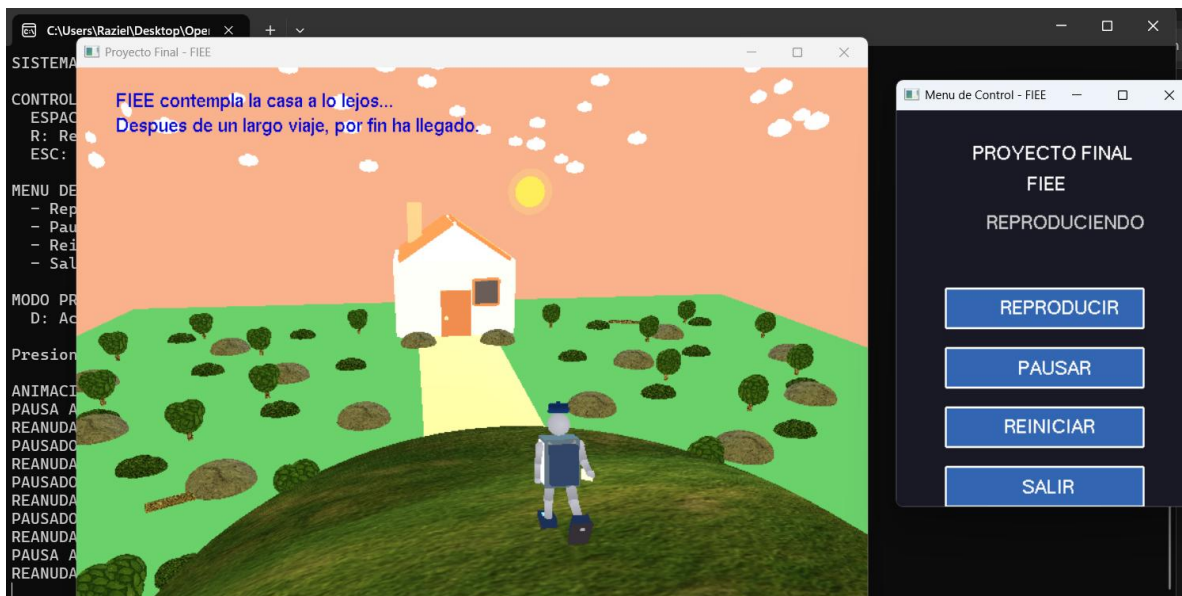
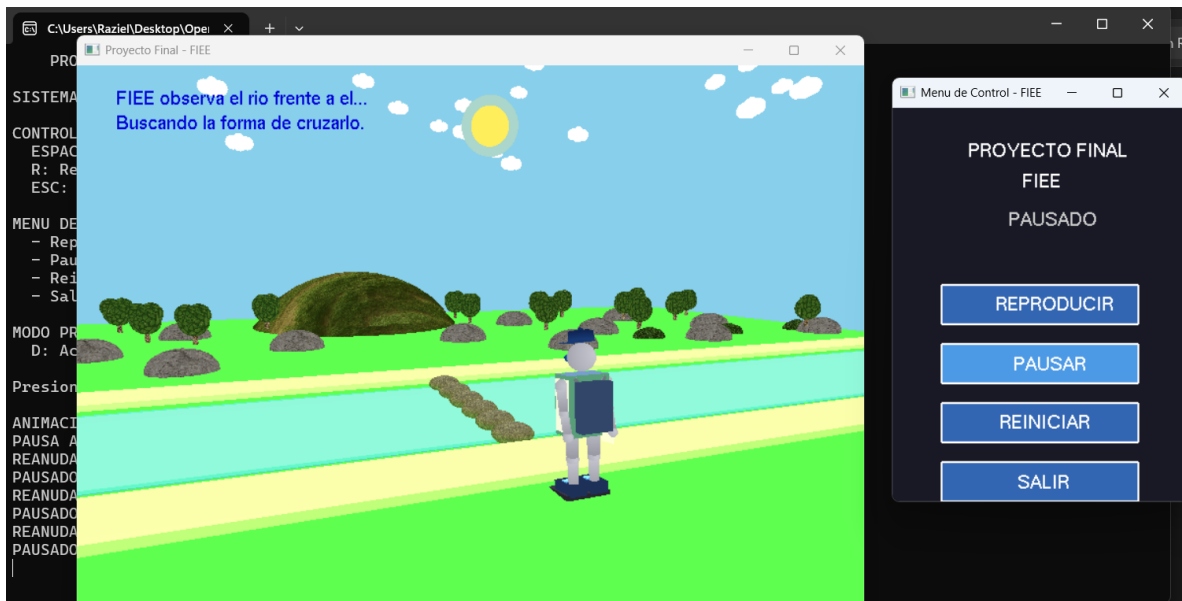
Hay funciones auxiliares como `esfera_textura` y `cilindro_textura` que:

- Activan `GL_TEXTURE_2D`, enlazan la textura correspondiente, crean un `GLUquadric` y habilitan el modo textura (`gluQuadricTexture(q, GL_TRUE)`).
- Dibujan la geometría (esfera o cilindro) con la textura aplicada.
- Desactivan texturas y limpian el `quadric`.

Con esto texturizo partes clave: ojos, torso, mochila, zapatos y metal del cuerpo, lo cual cubre el requerimiento de texturizado del personaje.

# RESULTADOS





# MANUAL DE USUARIO

Este manual explica el uso y funcionamiento de la mini-película interactiva desarrollada en OpenGL, protagonizada por el robot mensajero FIEE.

El usuario podrá reproducir, pausar, reiniciar y visualizar la animación desde distintas perspectivas mediante un menú interactivo y controles de teclado y mouse.

## 1- Requisitos del sistema

Para ejecutar el programa correctamente, el usuario necesita:

- Sistema operativo Windows (recomendado).
- Librerías OpenGL y GLUT funcionando correctamente.
- Ejecutable del proyecto generado con MinGW/MSYS2 o similar.
- Mouse y teclado funcionales.

```
PS C:\Users\Raziel\Desktop\OpenGL\RJRP_ProyectoFinal> g++ main.c robot.c escenas.c objetos.c -lfreeglut -lglu32 -lopengl32 -o proyecto.exe
```

## 2- Iniciar la aplicación

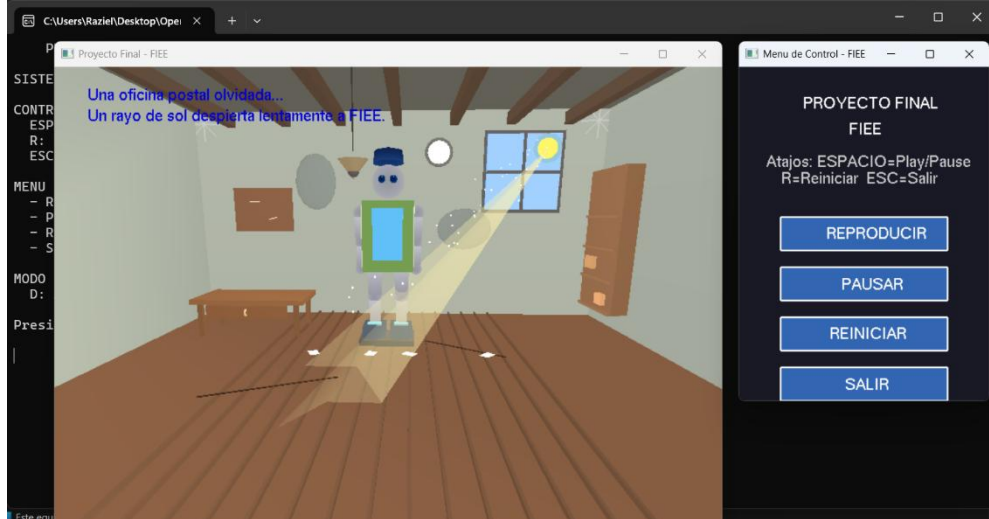
1. Abrir la carpeta donde se encuentra el archivo ejecutable.
2. Verificar que la carpeta /texturas se encuentre junto al EXE.
3. Ejecutar el archivo .exe dando doble clic o desde consola.

Al iniciar, aparecerán dos ventanas:

- La ventana principal de la animación.
- La sub-ventana del menú interactivo.

## 3- Interfaz principal

La interfaz está compuesta por:



## Ventana de Animación

Aquí se muestra la mini-película con FIEE actuando dentro de cada escena.

Lo que el usuario verá:

- El mundo 3D (escenarios, ambiente, objetos).
- Las animaciones del robot.
- Las cámaras cinematográficas fijas por escena.

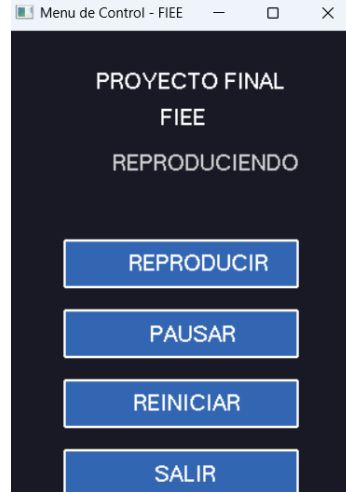
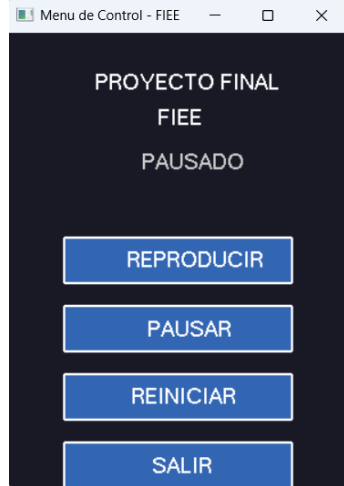


## Menú de Control

El menú contiene cuatro botones:

- Reproducir
- Pausar
- Reiniciar
- Salir

Cada botón cambia de color al seleccionarlo.



#### 4. Controles del usuario

El usuario puede interactuar con la animación usando el teclado y el mouse:

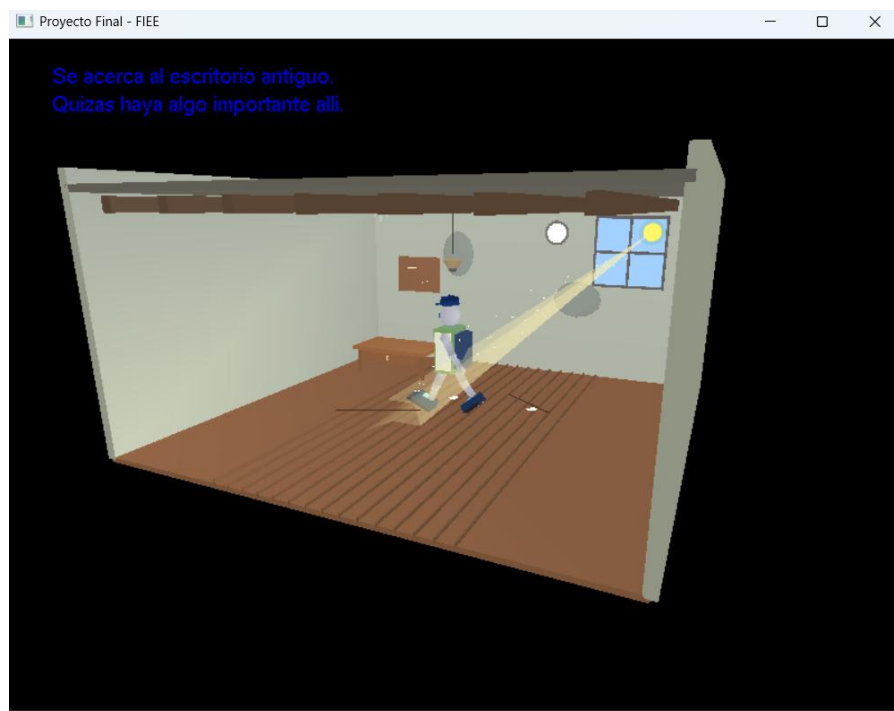
##### Controles de teclado

Tecla	Acción
<b>ESPACIO</b>	Reproducir / Pausar la animación
<b>R</b>	Reiniciar desde la escena 1
<b>D</b>	Activar/Desactivar modo de cámara libre (debug)
<b>C</b>	Mostrar en consola la posición actual de la cámara
<b>ESC</b>	Salir del programa

##### Controles del mouse (modo debug)

Cuando el usuario presiona D, puede usar el mouse para controlar libremente la cámara:

- Mover el mouse presionado → Rotar la cámara alrededor de la escena
- Rueda del mouse (scroll) → Zoom in / Zoom out
- Click izquierdo → Activar arrastre
- Click izquierdo suelto → Detener rotación



# CONCLUSION

El desarrollo de este proyecto me permitió integrar de manera práctica todo lo aprendido durante el curso de Graficación por Computadora. Logré construir una mini-película completamente funcional en OpenGL, con un personaje principal animado mediante un sistema propio de keyframes, un esqueleto jerárquico y un conjunto de escenarios coherentes que siguen una narrativa clara. Implementar a FIEE su cuerpo articulado, su movimiento fluido y sus reacciones fue uno de los mayores logros, ya que requirió la combinación de estructuras de datos, texturas, interpolación, iluminación y modelado 3D. También destaco como logro importante la modularidad del proyecto: cada escena, objeto y parte del robot está separada en su propio módulo, lo que permite entender el código y mantenerlo sin que se vuelva caótico.

Otro logro significativo fue el uso de estructuras de datos no solo “por cumplir”, sino de forma natural en el funcionamiento del programa: la lista enlazada de frames para la animación, el árbol para la estructura corporal del robot y la cola para manejar la secuencia narrativa. Esto me ayudó a comprender que el diseño de datos puede ser una herramienta creativa cuando se combina con graficación. Además, incluí un sistema de cámara manual (modo debug) que me permitió ajustar la composición cinematográfica, así como integración de texturas en partes del robot y en varios objetos del entorno.

A pesar de los avances, también encontré algunas limitaciones. Una de ellas es que muchas formas geométricas son relativamente simples debido al uso de primitivas básicas de OpenGL; objetos más complejos hubieran requerido modelado externo o técnicas adicionales. También hubo limitaciones en la iluminación: aunque logré simular ambientes distintos, OpenGL fijo no permite efectos avanzados como sombras dinámicas o iluminación física. La animación, aunque fluida, dependió totalmente de keyframes manuales, lo que hace que movimientos complejos tomen mucho tiempo de ajustar.

Finalmente, por cuestiones de tiempo, no integré detección de colisiones, físicas, animaciones secundarias o lógica avanzada de interacción.

En general, este proyecto representa un reto cumplido: pude construir una narrativa visual completa, un personaje propio y un mundo 3D funcional partiendo desde cero. Aunque hay muchas mejoras posibles, el resultado final demuestra un dominio sólido de OpenGL clásico, estructuras de datos, animación manual y diseño modular, cumpliendo ampliamente con los objetivos planteados para el proyecto final.

## REFERENCIAS

1. Zavala, F. (2025, October 12). Introducción al Texture Mapping en OpenGL. *Código En Llamas*. Retrieved from <https://codigoenllamas.com>
2. OpenGL FAQ / 21 Texture Mapping. (n.d.). Retrieved from <https://www.opengl.org/archives/resources/faq/technical/texture.htm>
3. bullet3/examples/ThirdPartyLibs/stb\_image/stb\_image.h at master · bulletphysics/bullet3. (n.d.). Retrieved from [https://github.com/bulletphysics/bullet3/blob/master/examples/ThirdPartyLibs/stb\\_image/stb\\_image.h](https://github.com/bulletphysics/bullet3/blob/master/examples/ThirdPartyLibs/stb_image/stb_image.h)