UNIVERSIDAD AUTÓNOMA DE MÉXICO

FACULTAD DE INGENÍERIA

LAB. COMPUTACIÓN GRÁFICA E INTERACCIÓN HUMANO-COMPUTADORA

GRUPO: 1

PROYECTO FINAL: MANUAL TÉCNICO

NO. CUENTA: 319162538

PROFESOR: EDEN ESPINOZA URZUA

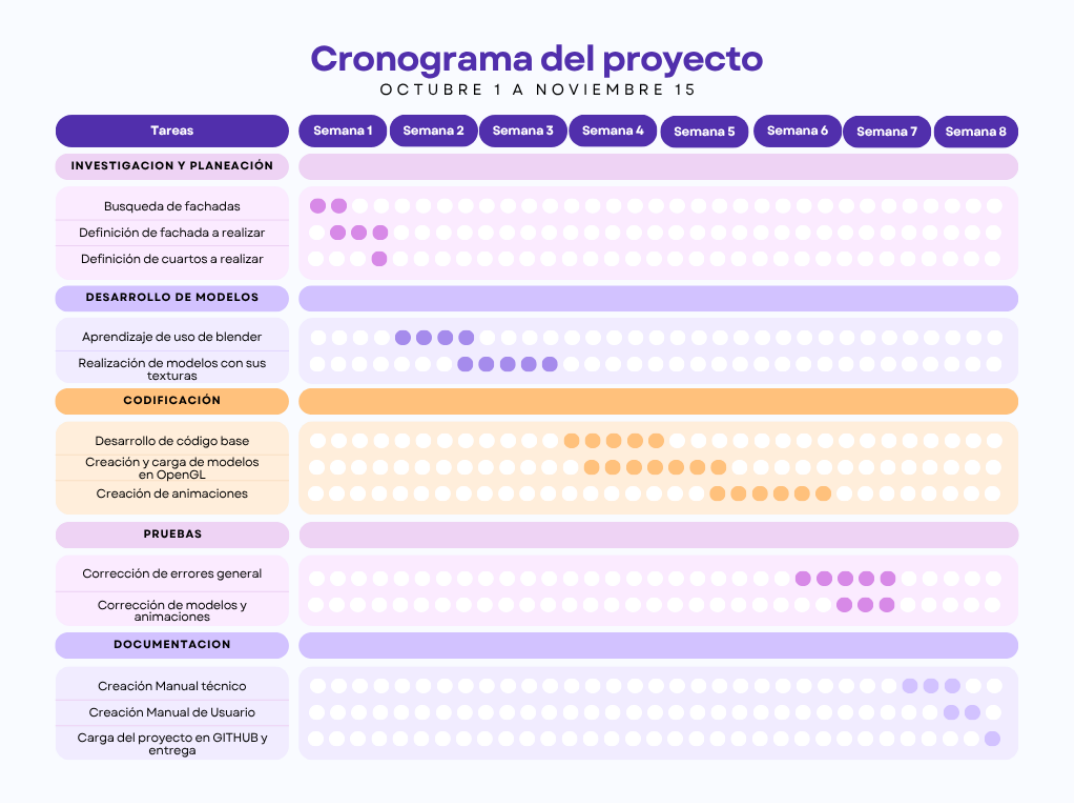
SEMESTRE: 2026 - 1

Índice

Objetivos

- Aplicar los conocimientos del curso para desarrollar una recreación 3D de la fachada y dos cuartos interiores de la Torre de la serie *Teen Titans Go*, utilizando OpenGL para construir las escenas. Se debe asegurar que la representación virtual sea lo más fiel posible a las imágenes de referencia, incluyendo la ambientación correspondiente a cada espacio.
- Integrar un flujo de trabajo de modelado externo (Blender) con la renderización en OpenGL, modelando un mínimo de siete objetos distintos en OpenGL y extras en Blender, aplicando texturas y geometría adecuadas, dentro del código base proporcionado.
- Implementar un sistema de animaciones coherentes y generadas por código para dar vida a la escena , desarrollando cinco animaciones requeridas (tres sencillas y dos complejas), demostrando el manejo de transformaciones y lógica de programación dentro del entorno de OpenGL.

Diagrama de Gantt



Alcance del proyecto

El presente proyecto tiene como alcance el desarrollo de una aplicación gráfica funcional en OpenGL que recrea una ambientación 3D basada en la serie Teen Titans Go, delimitándose a lo siguiente:

1. **Desarrollo del Entorno:** La creación de una escena 3D que representa fielmente la fachada exterior y dos cuartos interiores de la Torre de los Teen Titans, basándose en imágenes de referencia.

2. **Modelado y Texturizado**: La creación e integración de al menos siete objetos 3D únicos y evaluables modelados en OpenGL, junto con Blender y texturizados adecuadamente.

3. **Implementación de Animaciones:** La programación de cinco animaciones generadas por código que sean coherentes con el entorno.

4. **Software Funcional:** La entrega de un archivo ejecutable funcional construido sobre el código base proporcionado por el profesor.

5. **Cámara y vista:** El movimiento de cámara fue implementado de forma fluida y en tiempo real, ofreciendo libertad para explorar el espacio desde diferentes ángulos y perspectivas, mejorando la navegación y la interacción del usuario con el recorrido.

Limitantes

A pesar de los logros alcanzados, la implementación de este se enfrentaron ciertas limitaciones que afectaron en cierta medida el desarrollo hasta el final:

1. **Recursos técnicos:** La capacidad de procesamiento y memoria disponible en las estaciones de trabajo condicionó la complejidad de las texturas y el número de luces dinámicas que se pudieron implementar sin afectar el rendimiento en tiempo real.

2. **Herramientas y entorno:** Aunque Blender y OpenGL ofrecieron amplias posibilidades, algunas funcionalidades avanzadas como shaders personalizados o efectos visuales complejos no se exploraron a fondo debido a la curva de aprendizaje y el tiempo disponible.

3. **Limitaciones de Animación:** Todas las animaciones implementadas deben ser generadas exclusivamente por código. No se permite el uso de animaciones pre-renderizadas o importadas desde Blender.

4. **Escalabilidad:** El proyecto se limitó y ajusto a los requerimientos establecidos en la evaluación, por lo que añadir o implementar mas conceptos provocaban fallo o errores en la programación y mucha exigencia grafica para le computador.

Metodología de software aplicada

Para la gestión de este proyecto se opto por el uso de la metodología de desarrollo incremental, involucrando planificación inicial de un modelo en cascada y flexibilidad de construcción por etapas. Entre las etapas tenemos las siguientes:

- **Fase 1: Investigación y Planeación (Semanas 1-2)** Esta fase inicial definió el alcance y los cimientos conceptuales del proyecto. Las tareas incluyeron la búsqueda de referencias visuales, la definición de la fachada de la Torre de los Teen Titans y la selección de los dos cuartos a recrear.

- **Fase 2: Desarrollo de Modelos (Semanas 2-4)** Corresponde al primer incremento tangible del proyecto: la creación de los *assets* (activos) 3D. Esta etapa se centró en el aprendizaje de Blender y la realización y texturizado de todos los modelos requeridos para la escena.

- **Fase 3: Codificación (Semanas 4-6)** En esta fase se tomó el incremento de "Modelos" y se le dio vida en el entorno gráfico. Comprendió el desarrollo sobre el código base, la implementación de la lógica para la carga de los modelos en OpenGL y la creación de las animaciones programadas.

- **Fase 4: Pruebas (Semanas 6-7)** Esta fase se dedicó al refinamiento y depuración del incremento de "Codificación". Se enfocó en la corrección de errores generales de la aplicación, así como en los ajustes visuales y funcionales de los modelos y las animaciones.

- **Fase 5: Documentación y Entrega (Semanas 7-8)** El incremento final consistió en empaquetar el producto para su entrega. Incluyó la creación de los manuales técnico y de usuario, y la preparación final del repositorio de GitHub para la entrega del proyecto.

Este modelo incremental aseguró un flujo de trabajo ordenado, donde cada etapa (Modelado, Código, Pruebas) se construyó sobre el éxito de la anterior, facilitando la detección de problemas y garantizando que todos los componentes estuvieran listos para la fecha de entrega.

Documentación

1. **Funciones principales de control**

Int main()

Su propósito es ejecutar el código y ser la entrada principal del programa, en esta función se lleva a cabo lo siguiente

- **Inicialización:** Configura GLWF, GLEW, crea la ventana de OpenGL y establece las funciones de callback (teclado-mouse).

- **Carga de Recursos:** Carga y compila los shaders (lightingShader, lampShader), carga todos los modelos 3D (.obj) y las texturas (stbi_load).

- **Configuración de VAO/VBO:** Define el búfer de vértices para los objetos primitivos (cubos) que se usan en funciones como DrawMueble y DrawReloj, o para el dibujo en general dentro del ambiente de OpenGL.

- **Game Loop (while):** Es el corazón del programa. En cada fotograma, calcula el deltaTime, procesa las animaciones, maneja la entrada (DoMovement), configura la iluminación (luces direccionales, puntuales y spotlight), y finalmente dibuja todos los objetos en la escena.

- **Limpieza:** Termina glfwTerminate() al salir.

Void keyCallBack()

Manejar toda la entrada de teclado que no es movimiento, si no eventos de presionar una vez como las animaciones que tenemos, en esta función se lleva a cabo lo siguiente:

- **Entrada de teclado:** Cuando se presionan las teclas SPACE, 1, 2, 3, 4, 5 y R se interactúa con el entorno y permite reproducir animaciones que fueron calculadas con el delta time.

    - **ESPACIO:** Enciende y apaga la luz de la lámpara (active).
    - **1:** Inicia/detiene la animación del reloj (gClockAnimating).
    - **2:** Abre/cierra la puerta del comedor (gDoorComedor).
    - **3:** Tira/levanta la escoba (gEscoba).
    - **4:** Activa/desactiva la bola disco (gDisco).
    - **5:** Inicia la animación de caminar del perro (dogAnim).

- **R:** Reinicia el estado de las animaciones del reloj, el perro y la bola disco.

Void MouseCallback()

Esta función se encarga de manejar el movimiento del mouse, otros aspectos importantes de los que se encarga son:

- Calcular el desplazamiento del mouse xOffset y yOffset desde el último fotograma.

- Enviar toda la información obtenida a la clase camera para así poder rotar la vista.

Void DoMovment()

Esta función se encargará de manejar la entrada del teclado para el movimiento de cámara, entre sus principales propósitos tenemos los siguientes:

- Revisa si las teclas W, A, S, D o flechas están siendo presionadas en el fotograma actual.

- Llama al método camera.ProcessKeyboard para mover la posición de la cámara.

Estas son el corazón de nuestro programa ya que controlan los elementos más básicos, pero a la vez los más importantes en OpenGL.

2. **Funciones de lógicas de animación**

Void AnimationDog()

Esta función se encargará de calcular los ángulos de las articulaciones del perro para así poder simular su caminata, en ella podemos encontrar diversos aspectos más:

- Revisa si la animación está activa (dogAnim == 1).

- Utiliza una máquina de estados simple (bool step) para alternar el movimiento de las patas (ej. RLegs += 0.3f y luego RLegs -= 0.3f).

- Actualiza las variables globales (RLegs, FLegs, head, tail, earL, earR) que se usarán al dibujar el perro.

- Mueve la posición global del perro (dogPos.x) para que avance.

auto animations = [&](Animation& d, float dt)

Esta función se encargará de calcular la interpolación de los ángulos, en ella podemos observar también los siguiente:

- Se usa para la puerta, la escoba y la bola disco.
- Compara el ángulo actual (d.angle) con el ángulo objetivo (d.target).
- Incrementa o disminuye suavemente el ángulo actual basado en d.speed y deltaTime, asegurando que no se pase del objetivo.

Estas funciones son la clave para poder llevar a cabo nuestras animaciones de manera exitosa en cuestión de una de las complejas que se implementó.

3. **Funciones de Dibujado (Primitivas / Modelado Jerárquico).**

Void DrawMueble()

Esta función se encargara de dinijar el mueble de la televisión usando 5 cubos escalados implementando lo siguiente:

- Utiliza la matriz base para definir la posición y rotación del mueble.
- Dibuja 5 componentes distintos, todos relativos a base:
  - Dos cubos escalados verticalmente para los laterales ((1) Lateral izquierdo, (2) Lateral derecho).
  - Dos cubos escalados horizontalmente para la base y la tapa ((3) Base inferior, (4) Tapa superior).
  - Un cubo final para la repisa interna ((5) Repisa intermedia).
- Aplica la textura MuebleTV a todas las partes.void DrawTele()

Void DrawTele()

Esta función se encargará de dibujar la mesa de la parte trasera de la sala de estar, implementado lo siguiente:

- Define una matriz base para la posición general.
- Dibuja el (1) Tablero principal, un cubo escalado horizontalmente con la textura Mesa.

- Define una función lambda interna (auto drawPata) para simplificar el código, ya que las 4 patas son idénticas.

- Llama a drawPata cuatro veces con diferentes desplazamientos (xOff, zOff) para colocar las patas en las esquinas.

Void DrawReloj()

Se encargará de dibujar un reloj funcional en la pared y también de procesar su animación, es una función primitiva algo compleja que involucra lo siguiente:

- Recibe el gClockTime (tiempo acumulado) para calcular los ángulos.

- Dibuja el cuerpo y la carátula del reloj usando texturas MarcoTV y BaseReloj.

- Calcula los ángulos para las manecillas (minuteAngle, hourAngle) usando el tiempo y la operación módulo (fmodf) para crear un ciclo.

- Dibuja las manecillas (hora, minutos) como cubos delgados y largos, aplicando la rotación correspondiente (glm::rotate) sobre el eje Z (profundidad) para que giren correctamente.

Void DrawEscoba()

Se encargará de dibujar la escoba y de igual forma manejar su aminación, para ello se toma en cuenta lo siguiente:

- Recibe el anguloCaida (calculado por la lambda animations).

- Aplica esta rotación a la matriz base, haciendo que toda la escoba gire desde su punto de origen (la base).Dibuja el (1) Palo como un cubo largo y delgado.

- Dibuja la (2) Cabeza (el bloque de plástico).

- Utiliza una lambda (drawCerda) para dibujar 5 cubos pequeños y anchos que simulan las cerdas ((3) Cerdas).

Void DrawSillaSimple()

Se encargará de dibujar una silla que acompañara la mesa dibujada anteriormente, para ella se toma en cuenta lo siguiente:

- Utiliza la misma técnica de ensamblaje que la mesa.

- Dibuja un Asiento (cubo plano).

- Dibuja un Respaldo (cubo alto y delgado).

- Utiliza una lambda (pata) para dibujar las 4 patas en sus posiciones.

Void DrawLampara()

Esta se encargara de dibujar la lampara de la sala de estar con la capacidad de mostrar un foco brillante que se puede prender y apagar, para ello se hizo lo siguiente:

- Usa el lightingShader (normal) para dibujar la Base y el Tubo metálico, aplicando texturas para que reciban sombras.

- Cambia activamente al lampShader (el shader de brillo).

- Dibuja los Focos (la parte de arriba) usando el lampShader. Esto hace que esos cubos ignoren la iluminación y se vean blancos y brillantes, simulando una luz encendida.

- Al final, restaura el lightingShader para no afectar al resto de la escena.

Con estas funciones logramos implementar los objetos requeridos en nuestro plano 3D y también amueblarlo para que se vea mas fiel a la fachada original.

4. Funciones de Dibujado (Modelos .obj Animados)

Void DrawDoorHinged()

Esta función se encargara de dibujar un modelo .obj de la puerta de la cocina y le apliara una rotación sobre un pitovte asignado en la bisagra en lugar de rotar en su centro, para ello se hizo lo siguiente:

- Esta función recibe el modelo a dibujar (Comedor) y su struct de animación (gDoorComedor).

- Utiliza la técnica de transformación de pivote para simular una bisagra:
    - **glm::translate(M, d.hingeLocal):** Se traslada la matriz al punto exacto de la bisagra (coordenadas obtenidas de Blender).
    - **glm::rotate(M, d.angle):** Se aplica la rotación de apertura (el ángulo d.angle es calculado por la lambda animations).
    - **glm::translate(M, -d.hingeLocal):** Se regresa la matriz a su origen.

- Este orden (Mover-Rotar-Regresar) asegura que el modelo gire alrededor del punto hingeLocal en lugar de su propio centro.

Void DrawDiscoBall()

Esta función será encargada de manejar la animación compleja compuesta de la bola disco, que involucra dos modelos separados que es la esfera disco y el tubo para formar un solo objeto, esto llevado acabo de la siguiente forma:

- Utiliza un pivot (pivote) fijo que define el punto de anclaje en el techo (coordenadas de Blender).

- La función realiza dos operaciones de dibujado separadas:
- Para el tubo:
    - Usa la fórmula de pivote (Trasladar -> Escalar ->Trasladar-Inverso).
    - glm::scale estira el tubo en el eje Y basándose en anim.currentLength, haciendo que parezca que baja del techo.

- Para la esfera:
    - Usa la misma fórmula de pivote (Trasladar $\to$ Rotar/Trasladar $\to$ Trasladar-Inverso).
    - Aplica una glm::translate vertical (en Y) basada en anim.currentLength * 1.3f. Este 1.3f es un factor de ajuste para sincronizar la bajada de la esfera con el estiramiento del tubo.
    - Aplica una glm::rotate sobre el eje Y basada en anim.rot.y (calculada en el Game Loop) para hacerla girar continuamente.

- El resultado es una animación compleja donde el tubo se estira y la esfera baja y gira, todo sincronizado por las variables del struct Animation.

5. Struct Animation

Esta estructura unifica el control de todas las animaciones y cada objeto animado (como la puerta, la escoba o la bola disco) tiene su propia instancia de esta struct para almacenar su estado. Contiene variables para:

- Control General: Un interruptor (animate) para activar/desactivar la animación y su posición (pos).

- Rotaciones: Variables como angle (ángulo actual), target (ángulo objetivo) y hingeLocal (la bisagra) para controlar la puerta y la escoba.

- Estiramiento: Variables como currentLength (largo actual) y maxLength (largo máximo) para controlar la animación de bajada de la bola disco.

6. Sistema de iluminación

El entorno de la fachada se ilumina usando una combinación de tres tipos de fuentes de luz, que trabajan juntas para crear una escena dinámica y realista, una luz direccional, luces puntuales y un foco de luz (spotlight).

Luz direccional

- Esta luz afecta a todos los objetos de la escena por igual, independientemente de su posición. Proporciona la iluminación base y general para todo el entorno.

- Se define con una dirección fija (apuntando hacia abajo) y tiene componentes ambientales, difusos y especulares de intensidad media (0.3f a 0.8f). Su función principal es asegurar que la escena nunca esté en oscuridad total y que las sombras tengan un relleno suave.

Luces puntuales

La escena utiliza un arreglo de luces puntuales, pero la más importante es pointLights[0], que está vinculada a la lámpara de pie, llevando consigo lo siguiente:

- **Posición:** La luz está colocada manualmente en las coordenadas (-4.0, 52.0, -10.8), coincidiendo con la ubicación física de la lámpara en la escena.

- **Control:** Se enciende y apaga usando la tecla ESPACIO, que alterna la variable active.

- **Lógica:** Cuando active es false, todas las propiedades de la luz (ambiente, difusa, especular) se establecen en 0.0, apagándola por completo. Cuando active es true, la luz se enciende con un color amarillo (1.0, 1.0, 0.0) de alta intensidad, iluminando el área circundante.

Foco de luz/Linterna

Esta luz está directamente vinculada a la cámara estilo primera persona y se controla de la siguiente manera:

- La posición del foco es siempre la posición de la cámara (camera.GetPosition()).

- La dirección del foco es siempre hacia donde mira la cámara (camera.GetFront()).
- Se crea un cono de luz enfocado que ilumina cualquier objeto al que el usuario apunte directamente y las propiedades cutOff y outerCutOff definen el tamaño y la suavidad de los bordes de este cono de luz.

7. Variables del programa

Variables globales

Estas variables declaradas al inicio del programa almacenan el estado y los recursos esenciales que deben ser accesibles desde cualquier función (como main, KeyCallback y las funciones de dibujado).

Variables de control de escena y tiempo

- **camera:** Es una instancia de la clase Camera y almacena la posición y orientación del observador en el mundo 3D, es controlada por las funciones DoMovement (para moverse) y MouseCallback (para girar la vista).

- **deltaTime / lastFrame:** Estas dos variables son cruciales para que las animaciones sean fluidas. deltaTime almacena el tiempo exacto (en segundos) que tardó en renderizarse el último fotograma. Todas las animaciones se multiplican por deltaTime para asegurar que se muevan a la misma velocidad, sin importar si la computadora es rápida o lenta.

- **keys[]:** Un arreglo de booleanos que registra qué teclas están siendo presionadas. Es usado por DoMovement para permitir el movimiento continuo de la cámara.

Recursos de renderizado

- **VAO / VBO:** Almacenan el ID de OpenGL para el cubo primitivo. Este cubo es la base geométrica para todos los objetos construidos manualmente en la escena, como la televisión, la mesa, el reloj y la lámpara.

- **lightingShader / lampShader:** Dos instancias de la clase Shader que contienen los programas de GPU. lightingShader se usa para el 99% de la escena (objetos con sombras y texturas), mientras lampShader se usa solo para los objetos que deben brillar (como los focos de la lámpara).

- **PantallaTV, MarcoTV, MuebleTV, etc:** Variables de tipo unsigned int que guardan el ID único de cada textura después de ser cargada en la memoria de la GPU.

Variables de estado de animación

- **gDoorComedor, gEscoba, gDisco:** Son las instancias del struct Animation y cada una almacena el estado completo de un objeto animado.

- **Variables del Perro (dogPos, RLegs, head, etc.):** A diferencia de las animaciones simples, la animación compleja del perro requiere un conjunto dedicado de variables float e int para almacenar el ángulo de cada articulación (patas, cabeza, cola) y su posición global.

- **active / Light1:** Variables que controlan el estado de la luz de la lámpara. active es el interruptor (true/false) que se activa con la tecla Espacio, y Light1 almacena el color de la luz.

Estas variables se usan de manera continua y dinámica en todo nuestro código para lograr todos los aspectos implementados en esta fachada.

Conclusiones

El desarrollo de este proyecto permitió aplicar de forma práctica los fundamentos del curso, culminando en una recreación 3D funcional de la Torre de los Teen Titans Go en OpenGL, se logró integrar exitosamente un flujo de trabajo completo, desde el modelado y texturizado de los siete objetos requeridos en Blender hasta su renderización y ambientación en la escena. La implementación de las cinco animaciones generadas por código, incluyendo dos complejas, fue crucial para añadir dinamismo al entorno y demostrar el manejo de transformaciones.

El principal desafío técnico fue la correcta importación de los modelos de Blender al código base de OpenGL, asegurando la fidelidad visual de las texturas y escalas. Asimismo, la programación de animaciones complejas que fueran coherentes con el contexto requirió una sólida comprensión de la lógica matemática. Este proyecto no solo cumple con los requisitos técnicos, sino que también consolida las habilidades en la creación de entornos gráficos interactivos, desde la conceptualización hasta la entrega documentada.