Universidad Nacional Autónoma de México

Facultad de ingeniería

Proyecto Final
Manual Técnico

Computación Gráfica e interacción Humano/Computadora

Grupo 5 Teoría Semestre: 2026-1

Por:

320010941

Objetivo:

Aplicar y demostrar los conocimientos adquiridos durante el curso mediante la recreación 3D de una fachada y sus espacios interiores en OpenGL

- Recrear una fachada y dos cuartos interiores. Cada cuarto deberá estar amueblado con varios objetos debido a la imagen de referencia.
- **Interactividad y Navegación:** Implementar una cámara sintética que permita al usuario navegar libremente por el escenario, simulando un recorrido virtual en primera persona.
- **Dinamismo y Animación:** Integrar 5 animaciones generadas por código: 3 sencillas y 2 complejas. Estas animaciones deben ser coherentes con el contexto del escenario y no limitarse a transformaciones básicas.

Tecnologías utilizadas

Visual Studio Community

Un Entorno de Desarrollo Integrado (IDE) gratuito y completo de Microsoft que permite a desarrolladores individuales, estudiantes y equipos pequeños crear aplicaciones para Android, iOS, Windows, web y la nube.

Ofrece un conjunto de herramientas robustas, un potente depurador, y es compatible con múltiples lenguajes de programación, como C#, C++ y Python, entre otros.

OpenGL

Una API (Interfaz de Programación de Aplicaciones) multiplataforma y de código abierto utilizada para renderizar gráficos 2D y 3D. Junto con un conjunto estándar de funciones que los desarrolladores usan para interactuar con el hardware gráfico (GPU) y crear visualizaciones complejas, como videojuegos, simulaciones y aplicaciones de CAD. Directamente incluida en la parte de los controladores de la tarjeta gráfica, no se puede instalar por separado

Paint

Aplicación de edición gráfica básica y fácil de usar que viene incluida con Microsoft Windows para poder crear y editar imágenes y dibujos simples, con herramientas para dibujar, colorear, recortar, redimensionar imágenes y agregar texto.

Gimp

Editor de imágenes gratuito y de código abierto para retocar fotografías, componer imágenes y crear gráficos, y es compatible con sistemas operativos como Windows, macOS y Linux. Ideal para manejar las imágenes que serán las texturas

Autodesk Maya

Es un software 3D profesional para crear personajes realistas y efectos dignos de películas taquilleras. Es una herramienta estándar en la industria del cine y los videojuegos para la animación, el modelado, la simulación y la creación de efectos visuales.

Diagrama de flujo del software

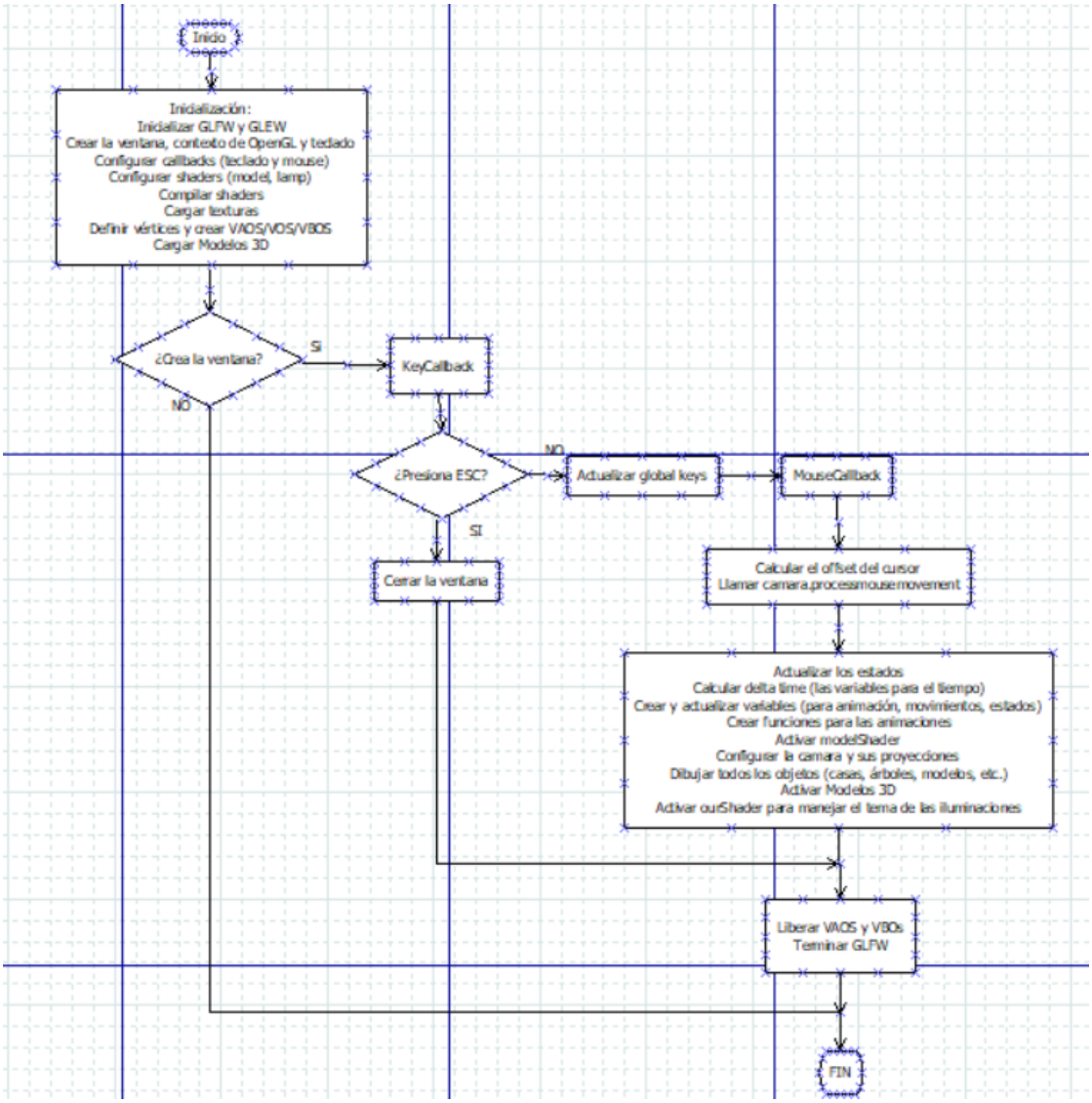


Diagrama de Gantt

Actividad	Inicio	Final	10/10/2025	11/10/2025	12/10/2025	13/10/2025	14/10/2025	15/10/2025	16/10/2025	17/10/2025	18/10/2025	19/10/2025	20/10/2025	21/10/2025	22/10/2025	23/10/2025	24/10/2025
Investigación	10/10/2025	15/10/2025															
Planificación	10/10/2025	16/10/2025															
Codificación	10/10/2025	14/11/2025															
Documentación	30/10/2025	06/11/2025															

Alcance del proyecto

Para empezar, veo muy importante mencionar el hecho de que no se usará algún software especializado en modelado 3D (3ds Max, Blender, Maya, etc). Esto porque los espacios del proyecto fueron pedidos en el entorno de OpenGL utilizando el código en C++ que usamos a lo largo de las prácticas del semestre. Los objetos modelados en 3D importados externamente de alguna de estas aplicaciones ya mencionadas serán contadas de forma extra.

Definiendo los límites del proyecto de lo que vamos y no vamos a realizar, tomando en cuenta también los recursos que necesitamos para este proyecto:

- Computadora para realizar todo el proyecto (en este caso mi laptop)
- Software dedicado: Visual Studio Community, OpenGL
- Excel para realizar diagramas
- Equipo de trabajo de 1 integrante (nada más yo)

El proyecto recreará el escenario de pueblo paleta, específicamente de los juegos remakes: Pokémon rojo fuego y verde hoja, junto con la casa de rojo, y el laboratorio del profesor Oak

Imágenes de referencia:

Pueblo paleta:

Sala y cuarto de la casa de Rojo:

Escenario: Pueblo paleta con las 2 casas, el interior -tanto la sala de estar y el cuarto- de la casa de rojo (el protagonista del juego) y el laboratorio del profesor Oak.

Modelado: Modelado y texturizado de 7 objetos diferentes ubicados dentro de las casas de este mismo escenario.

Navegación: La implementación de una cámara sintética que permite al usuario moverse libremente por el escenario.

Animaciones: La integración de 5 animaciones en total, generadas por código, divididas en 3 sencillas y 2 complejas. (El documento de Teoría solo pide 4 animaciones, pero el de Laboratorio es más específico con 5, así que es mejor usar ese número).

Entregables:

- Código fuente del proyecto en C++.
- El proyecto funcional en un repositorio de GitHub.
- Un archivo ejecutable del mismo escenario.
- Documentación (manual técnico y de usuario.)

Elementos fuera del alcance:

Como ya se mencionó, no usaremos ninguna aplicación dedicada al modelado 3D como blender o maya, y si se usan elementos diseñados de estas aplicaciones, serán contadas como un extra:

- **Físicas:** El proyecto no incluirá un motor de físicas. El usuario puede atravesar objetos (es decir, no hay detección de colisiones).
- **Interactividad avanzada:** "La interacción se limita a la navegación. El usuario no puede tomar, mover o interactuar con los objetos del escenario.

- **Sonido:** El proyecto no incluye efectos de sonido ni música de fondo; se intentó usar la librería irrklang e intentar añadirla al proyecto, pero ahora resulta que ya la cobran cuando antes era de uso “no comercial”.

Costos del proyecto.

Tomando en cuanto los requisitos que tenemos desde los alcances de este, el tiempo y la dedicación invertidas, tanto como mi trabajo y conocimientos que emplee para la realización del proyecto.

1. Costo de Producción:

Para no malbaratar mi mano de obra, estoy tomando en cuenta el costo de producción, las horas de trabajo y el software de realización

Fórmula: Costo de Producción = (Horas de Trabajo × Tarifa de Costo/Hora) + Costos Indirectos

- **Tiempo dedicado al proyecto:**
    - Estimación total de horas dedicadas al proyecto.
        - Investigación y Planificación: Una semana (10/10/2025 al 16/10/2025)
        - Modelado 3D y Texturizado: 2 meses (16/10/2025 al 25/10/2025)
        - Codificación (Cámara, animaciones, escena): 2 meses (16/10/2025 al 25/10/2025)
        - Documentación: 5 días
        - **Total de horas:** 150 horas
    - Tarifa de costo: $150 MXN / hora
- **Costos Indirectos:**
        - Servicios necesarios (Electricidad, Internet): $700 MXN

Cálculo de Costo:

- Costo de Mano de Obra: 150 horas * $150/hora = $22,500 MXN
- Costos Indirectos: $700 MXN
- **Costo Total de Producción: $23,200 MXN**

Estoy valuando mi costo de producción en $ 23,2000 XN. Este monto refleja las 150 horas dedicadas a la realización del proyecto en todos sus ámbitos, desde el inicio de la investigación del tema escenario a recrear hasta el desarrollo junto con la codificación, investigación y utilización de los recursos, bibliotecas, código etc.

Estableciendo una tarifa de costo de $150 MXN/hora, cubriendo el tiempo de un desarrollador junior. Además, incluye $ 700 MXN en costos indirectos de servicios.

2. Precio de Venta

Para vender mi proyecto, estoy contemplando 2 métodos debido al análisis anterior:

Costo-Plus

Tomar el costo establecido más un margen de ganancia

- Costo de Producción: $ 23,200 MXN
- Margen de Ganancia del 40%: $23,200 * 0.40 = $9280 MXN
- **Precio de Venta: $32,480 MXN**

Tarifa de Mercado

Considerando los conocimientos y experiencia que tengo en el tema, así como también la gestión completa del proyecto en toda su realización, la tarifa por hora es más alta.

- Total de Horas: 150 horas
- Tarifa de Servicio Profesional (Ej. $350 MXN / hora):
- **Precio de Venta: 150 * $350 = $52,250 MXN**

Este precio es competitivo para el desarrollo de una aplicación de visualización 3D a medida con las características solicitadas.

Limitantes

La única limitante que tengo para este proyecto fue no usar Blender, Maya 3D o algún software dedicado al modelado digital, esto hace que la calidad gráfica no sea tan realista, pero un poco más apegada a lo que fue en sus inicios el escenario recreado en Pokémon

Metodología de software aplicada

Para este proyecto, veo más viable usar la metodología en cascada, dado que los requisitos son fijos y definidos desde un inicio (las especificaciones en classroom dadas desde un inicio). No van a cambiar. Además, es un proyecto individual, por lo que los roles y ceremonias de Scrum son innecesarios (y así tengo más libertad para definir que parte de mi mente va a hacer cada cosa)

Metodología en cascada

Es un modelo lineal y secuencial donde cada fase debe completarse antes de iniciar la siguiente. Este modelo se ajusta perfectamente a un proyecto como este, porque los requisitos son fijos. Permitiendo describir el proceso de forma ordenada.

Fases del desarrollo dada la metodología elegida

1. **Fase de Análisis y Requerimientos:**
    - Un entorno virtual en OpenGL, un mínimo de 7 objetos modelados, 5 animaciones (3 sencillas, 2 complejas), y la documentación completa.
    - Se seleccionó la temática de la fachada y los cuartos (temática de videojuegos), presentando las imágenes de referencia para su aprobación.
2. **Fase de Diseño:**
    - Se planificó la estructura del software, decidiendo el uso del código base proporcionado. A partir de aquí se definió cómo se organizarían los modelos, la iluminación y la cámara sintética. Se crearon los diagramas de flujo del software y el diagrama de Gantt para planificar los tiempos de desarrollo.
3. **Fase de Implementación (Codificación):**
    - Se llevó a cabo el modelado 3D de los 7 objetos y el texturizado.
    - Se escribió el código en C++ y OpenGL para:
        - Renderizar la escena y los objetos.
        - Implementar la cámara sintética para el recorrido virtual.
        - Implementar las animaciones
4. **Fase de Pruebas (Verificación):**
    - Se realizaron pruebas funcionales para asegurar que el ejecutable funcionara correctamente.

    - Se comprobó que todos los requisitos (como el texturizado y la cantidad de objetos) se cumplieran.
5. **Fase de Mantenimiento (Entrega):**
    - Se compiló la versión final del ejecutable.
    - Se redactaron los manuales de usuario y técnico.
    - Se subió el proyecto funcional al repositorio de GitHub, asegurando el uso de rutas dinámicas.

Documentación del código

Hay algunos cambios que hice en mi código con relación a las prácticas trabajadas en el semestre:

Para Clase Camera:

- Implementa una cámara tipo FPS (First Person Shooter) usando Ángulos de Euler.
- Yaw: Rotación en el eje Y (mirar izquierda/derecha).
- Pitch (Cabeceo): Rotación en el eje X (mirar arriba/abajo).
- Se recalculan los vectores Front, Right y Up usando trigonometría básica para asegurar que el movimiento siempre sea relativo a donde mira el jugador. (esto para que sea más amigable de mover con AWSD y el mouse)

Para el cambio del día:

- Se utiliza interpolación lineal (Lerp) para transicionar suavemente entre colores. 'transitionFactor' va de 0.0 a 1.0.
- float transitionSpeed = 0.5f; // Controla qué tan rápido anochece/amanece.

Para las texturas:

- GL_NEAREST: Filtrado "Pixel Art". Al acercarse, se ven los píxeles duros en lugar de borrosos.
- Se cargan dos texturas diferentes (agua.png y agua2.png) en distintas IDs de OpenGL.
- No se mezclan en el shader, sino que se alternarán en el bucle principal para crear un efecto de "flipbook" o animación de cuadros.

Para la iluminación:

- Las Luces Puntuales (que son los Postes de luz a lado de las casas): Tienen una posición específica y atenuación.

- Linear y Quadratic: Coeficientes que hacen que la luz pierda fuerza con la distancia (física real).
- Ecuación: Intensidad = 1.0 / (Constant + Linear*Dist + Quadratic*Dist^2)

Para la animación:

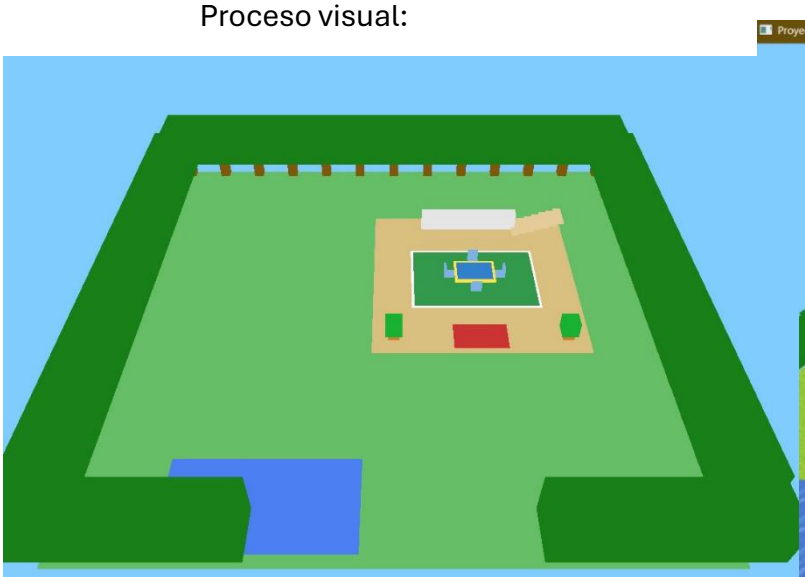
- Detección de Objetivo: glm::distance verifica si el Pokémon está cerca de su destino. Si llega (< 2.0f), se elige una nueva coordenada aleatoria (rand) dentro de un rango definido.
- Cálculo de Trayectoria Curva (Steering Behavior simple): glm::cross(direccion, UP): Obtiene un vector perpendicular a la dirección de vuelo (para un movimiento más “natural” de ho-oh y mew, ya que mew se la pasa romeando en el lore de Pokémon y en los juegos también se esconde en donde sea).
- sin(time): genera un valor oscilante (-1 a 1).
- Posicion += Direccion * Velocidad * DeltaTime: Asegura que vuele a la misma velocidad en una PC rápida o lenta.

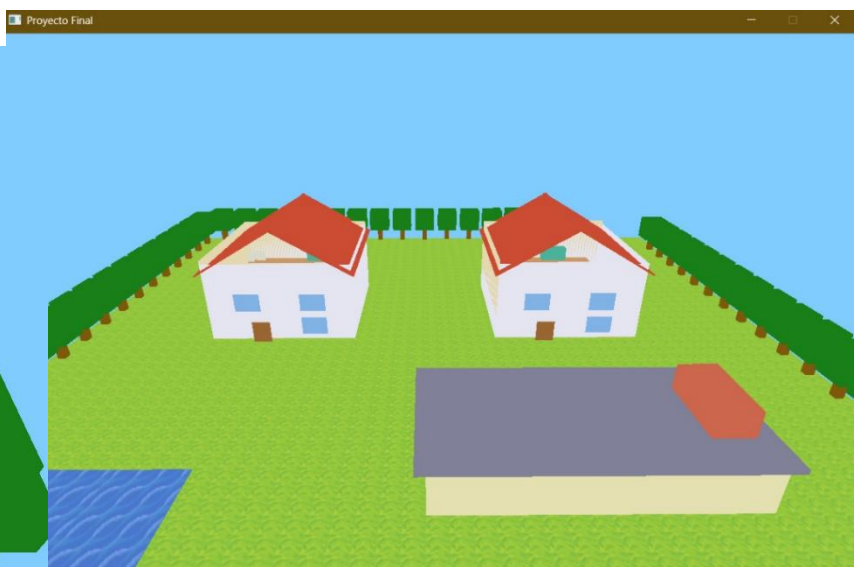
Primera versión del código, no se pondrán todas las versiones por optimización del documento

hola1 (Ámbito global)

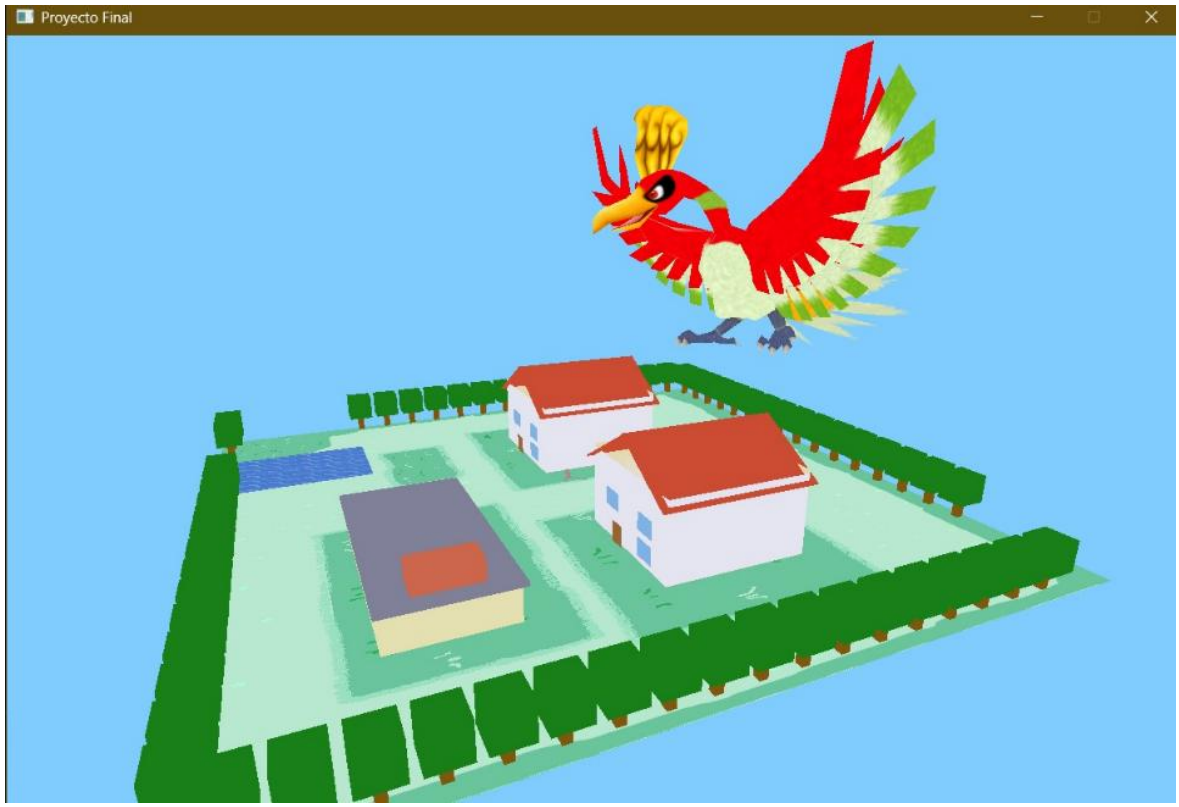
```
273             glfwSetWindowShouldClose(window, true);
274         if (key >= 0 && key < 1024) {
275             if (action == GLFW_PRESS)
276                 keys[key] = true;
277             else if (action == GLFW_RELEASE)
278                 keys[key] = false;
279         }
280     }
281
282     // Callback para el mouse
283     void MouseCallback(GLFWwindow* window, double xPos, double yPos) {
284         if (firstMouse) {
285             lastX = xPos;
286             lastY = yPos;
287             firstMouse = false;
288         }
289
290         GLfloat xOffset = xPos - lastX;
291         GLfloat yOffset = lastY - yPos;
292
293         lastX = xPos;
294         lastY = yPos;
295
296         camera.ProcessMouseMovement(xOffset, yOffset);
297     }
```

Proceso visual:





```
808             model = houseBaseModel;
809             model = glm::translate(model, glm::vec3(blue_X, blue_Y, blue_Z));
810             model = glm::scale(model, glm::vec3(blue_W - (frameThick * 2), blue_H, blue_D - (frameThick * 2)));
811             glUniform3fv(uniformColor, 1, glm::value_ptr(tableTopColor));
812             glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
813             glDrawArrays(GL_TRIANGLES, 0, 36);
814
815             // 2b. Marco Negro (4 piezas, plano)
816             // Arriba (Z+)
817             model = houseBaseModel;
818             model = glm::translate(model, glm::vec3(blue_X, frameY_Blue, blue_Z + (blue_D / 2.0f) - (frameThick / 2.0f)));
819             model = glm::scale(model, glm::vec3(blue_W, blue_H, frameThick));
820             glUniform3fv(uniformColor, 1, glm::value_ptr(blackColor));
821             glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
822             glDrawArrays(GL_TRIANGLES, 0, 36);
823             // Abajo (Z-)
824             model = houseBaseModel;
825             model = glm::translate(model, glm::vec3(blue_X, frameY_Blue, blue_Z - (blue_D / 2.0f) + (frameThick / 2.0f)));
826             model = glm::scale(model, glm::vec3(blue_W, blue_H, frameThick));
827             glUniform3fv(uniformColor, 1, glm::value_ptr(blackColor));
828             glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
829             glDrawArrays(GL_TRIANGLES, 0, 36);
830             // Izquierda (X-)
831             model = houseBaseModel;
832             model = glm::translate(model, glm::vec3(blue_X - (blue_W / 2.0f) + (frameThick / 2.0f), frameY_Blue, blue_Z));
833             model = glm::scale(model, glm::vec3(frameThick, blue_H, blue_D - (frameThick * 2)));
834             glUniform3fv(uniformColor, 1, glm::value_ptr(blackColor));
```

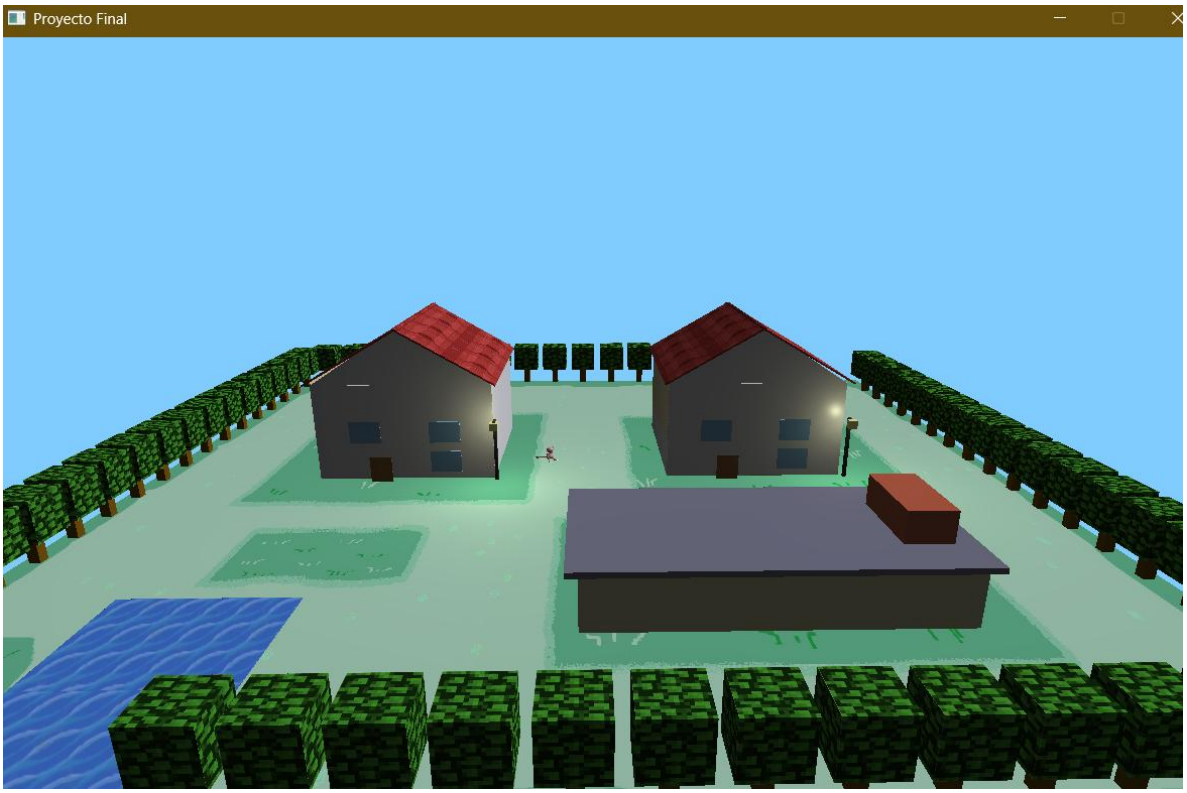


Versión final (2351 líneas de código):

Model.h ProyectoFinal.cpp

hola1 (Ámbito global) main()

```
290     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
291     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
292     // Configurar filtrado (pixelado)
293     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
294     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
295
296     // Cargar la imagen
297     unsigned char* agua2 = stbi_load("images/agua2.png", &textureWidth, &textureHeight, &nrChannels, 0); // <- CAMBIO AQUÍ
298
299     if (agua2) // <- CAMBIO AQUÍ
300     {
301         // Detectar formato
302         GLenum format;
303         if (nrChannels == 3)
304             format = GL_RGB;
305         else if (nrChannels == 4)
306             format = GL_RGBA;
307         else
308             format = GL_RGB;
309
310         glTexImage2D(GL_TEXTURE_2D, 0, format, textureWidth, textureHeight, 0, format, GL_UNSIGNED_BYTE, agua2); // <- CAMBIO AQUÍ
311         glGenerateMipmap(GL_TEXTURE_2D);
312     }
313     else
314     {
315         std::cout << "Failed to load texture: images/agua2.png" << std::endl; // <- CAMBIO AQUÍ
```



Conclusiones

La realización de este proyecto fue un recorrido integral que trascendió el objetivo inicial de "recrear un escenario". El verdadero aprendizaje no provino de la simple construcción de formas, sino de la superación de los desafíos de arquitectura de software necesarios para pasar de un renderizado estático y simple a una escena 3D dinámica, unificada e iluminada. Fue una simulación completa del ciclo de desarrollo en computación gráfica: desde la planificación y definición de alcances, pasando por una re-arquitectura de software (al momento de trabajar mis archivos, los códigos y los shaders), la depuración a bajo nivel (el pipeline de normales) y, finalmente, la implementación de sistemas dinámicos que le dieron vida a la escena. El resultado es la prueba tangible de un profundo aprendizaje, desde cómo dibujar un cubo hasta cómo orquestar un mundo 3D coherente e iluminado.

También quiero decir a manera de “agradecimiento final” que esta materia me gustó mucho, ya que es la única donde tú tienes el control creativo casi por completo, es muy interactiva y visual y eso fomenta un mejor aprendizaje para personas con estos métodos de aprendizaje, además de que disfruté mucho todo el proceso desde las prácticas hasta la culminación del proyecto.

Referencias

- Kessenich, J., Sellers, G., & Shreiner, D. (2016). *OpenGL programming guide: The official guide to learning OpenGL, version 4.5 with SPIR-V* (9ª ed.). Addison-Wesley Professional.
- Foley, J. D., van Dam, A., Feiner, S. K., & Hughes, J. F. (1995). *Computer graphics: Principles and practice* (2ª ed. en C). Addison-Wesley.
- Lelis, B. (2020). *Learn OpenGL: Learn modern OpenGL graphics programming in a step-by-step, example-oriented way*. Packt Publishing.
- Lelis, B. (s.f.). *Learn OpenGL*. Recuperado el 15 de noviembre de 2025, de https://learnopengl.com
- Khronos Group. (s.f.). *OpenGL registry*. Recuperado el 15 de noviembre de 2025, de https://registry.khronos.org/OpenGL/
- Royce, W. W. (1970). Managing the development of large software systems. *Proceedings of IEEE WESCON 26*, 328-388.
- Tutorials Point. (s.f.). *SDLC - Waterfall model*. Recuperado el 15 de noviembre de 2025, de https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm