UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ingeniería

Computación Gráfica e Interacción Humano-Computadora

Proyecto Final: Entorno Virtual 3D
“Temática Prehispánica”

Integrantes:

- Veraza García Amy Valentina (320490572)

- Solís Cisneros Cecilia Ximena (317042333)

Profesor(a): Edén Urzua Espinoza

Fecha: 15/11/2025

Tabla de Contenido

1. Objetivos

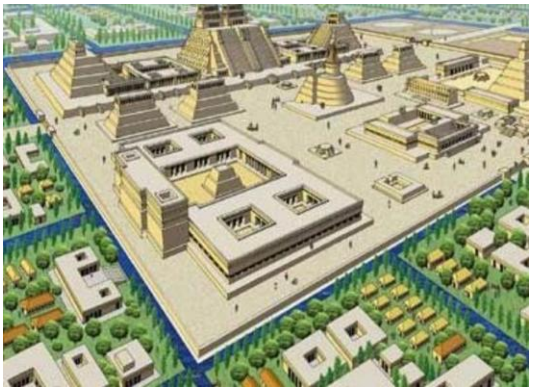
1) Modelar geometría base

Elementos: pirámides, calendario azteca, ring, árboles, globo aerostático, estatuas.

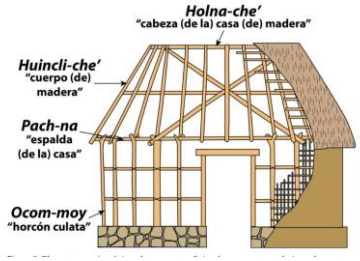
- **Alcance técnico**
  - Primitivas y mallas: pirámide (VAO/VBO propios), , árboles (tronco cilíndrico + billboard de hojas o malla low-poly), , calendario (plano con *height details* vía normal map o geometría ligera).

  - Ejes y escalas coherentes (1 unidad ≈ 1 m). Matrices model/view/projection.
  - Implementar una arquitectura de proyecto limpia en Visual Studio 2022 usando GLFW, GLEW, GLM, Assimp, stb_image y SOIL2, sin CMake.
  - Implementar modelos con ayuda de Blender y exportarlos a Visual Studio 2022.

Diseñar un escenario abierto con:



  - Mesa y silla con geometría procedural.
  - Fachada principal: una gran casa con el calendario azteca sobre ella.
  - Florero y flor procedural.
  - Máscara de jade en mosaico sobre la mesa.
  - Tianguis: canastas, chiles, petates, aguacates, jarrones, tendedero, piel de jaguar, etc.
  - Pequeñas chozas.



  - Monumentos: esculturas de Tula y Pirámide del Sol al fondo, no es una réplica tal cual, es un diseño cercano.
  - Implementar pasto y cielo procedurales con shader embebido (gProg):
    - Pasto texturizado anti-tiling y variante procedural.
    - Cielo con ciclo día/noche, sol, luna, estrellas, nubes y silueta de montañas.

  - Agregar vegetación por instancias:

        - Árboles y cactus con matrices de modelo generadas aleatoriamente y exclusiones alrededor de la zona central y monumentos.

    - Integrar animaciones:

        - Perrito voxel animado (cuerpo por cubos, patas y cola con movimiento).
        - Pelota que rebota y se desplaza cerca del juego de pelota.
        - Fogata del centro principal del tianguis, con movimiento, se activa y desactiva.
        - Un hacha cortando maíz.
        - Carretilla que se puede mover.
        - Se podría considerar también la implementación del Sol y la Luna, y su rotación para simular el día y la noche.

    - Permitir la navegación en primera persona con cámara libre (WASD + mouse).

    - Documentar el proceso, pruebas y riesgos, dejando abierta la extensión futura del proyecto.

- **Criterios de logro**

    - ≥7 objetos distintos con silueta reconocible.

    - Normalización de normales y *winding* CCW correcto (back-face culling estable).

    - Sin *z-fighting* notable en distancias de cámara esperadas.

2) Aplicar texturas y materiales diferenciados

Materiales: piedra tallada, arena, madera, metal.

- **Alcance técnico**

    - Filtrado trilineal con mipmaps; *wrap* repetitivo para suelos (arena).

    - Parámetros por material vía struct Material { vec3 ka,kd,ks; float shininess; } en GLSL.

    - Atlas o compresión de texturas (si procede) para reducir *binds*.

- **Criterios de logro**

    - Diferenciación visual clara por material bajo iluminación uniforme.

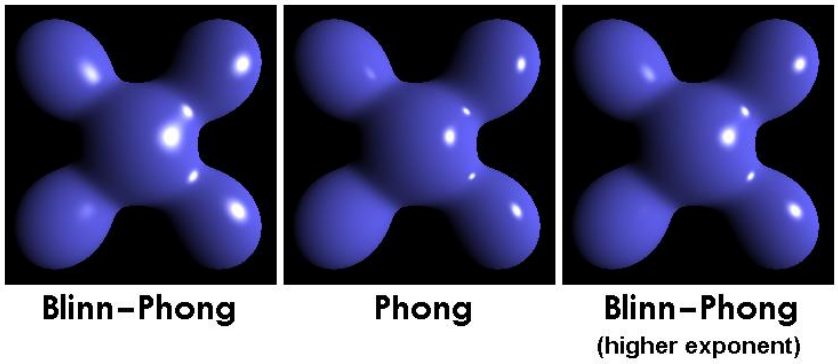
    - Costura UV mínima; sin *stretching* evidente a media distancia.

- Presupuesto de VRAM: ≤256 MB en texturas para compatibilidad.

3) Implementar iluminación (direccional, puntual, foco) y modelos de sombreado

Modelos: Phong / Blinn-Phong.

- **Alcance técnico**
  - **Direccional** (sol): dirLight.direction, color cálido al atardecer.
  - **Puntual** (antorchas/lamparines): arreglo pointLights[N] con atenuación (1, 0.09, 0.032) base.
  - **Spot/Foco** (linterna F): posición en camera.Position, cutOff y outerCutOff suaves.
  - Conmutadores: 1/2 alternan preset de iluminación (día/noche).
  - Selección de modelo: uUseBlinn (0=Phong, 1=Blinn-Phong).



- **Criterios de logro**
  - Sin *specular acne* (normales en espacio mundo correctas).
  - Atenuación realista (fuentes puntuales caen con la distancia).
  - Coste de drawcalls estable (≤120 llamadas/frame objetivo).

4) Cargar modelos externos con Assimp y cámara WASD+mouse

- **Alcance técnico**
  - Model.h con caché de texturas; rutas relativas assets/models/....
  - *Tangent space* opcional para normal mapping en modelos clave (calendario).
  - Cámara FPS (WASD + ratón), *clamp* de *pitch* y *deltaTime* para velocidad estable.

- **Criterios de logro**
    - Tiempo de carga total ≤5 s en equipo objetivo.
    - Sin pérdidas de textura (consola sin *warnings* de ruta).
    - Movimiento suave a 60 FPS en zona central del mapa.

5) Recorrido cultural con puntos de interés (POI) y ambientación "dulce" alrededor del juego de pelota

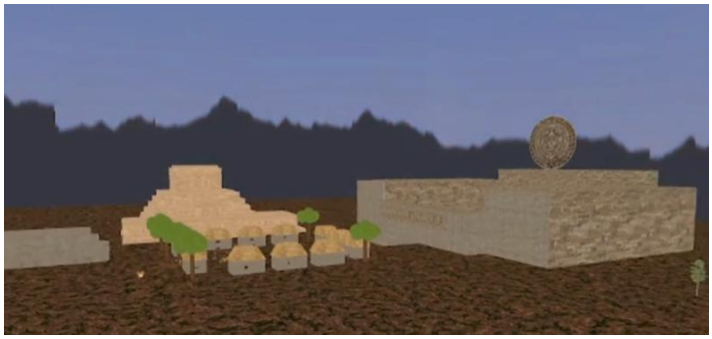
- **Alcance técnico**
    - POIs: nodos con position, title, desc, triggerRadius.
    - HUD minimal para fichas (E para ver/ocultar). Texto breve con referencia histórico-cultural.
    - Ambientación "dulce": paleta nocturna cálida, fog ligero, luces puntuales suaves, música/ambiente sutil (si se permite audio), partículas discretas (brillos) cerca del juego de pelota para atmósfera contemplativa.

6) Optimización de recursos y documentación del pipeline

- **Alcance técnico**
    - **Optimización**
        - Malla LOD para árboles/estatuas a media/larga distancia o *billboards*.
        - Reordenar draw por shader/material para minimizar *state changes*.
        - Frustum culling básico (AABB por objeto).
        - Un solo *ubo* para luces; materiales en *uniforms* agrupados.
    - **Documentación**
        - Diagrama MVP (model→world, world→view, view→clip).
        - Flujo de carga: Assimp → Mesh → GPU (VAO/VBO/EBO).
        - Flujo de render: *update* (input, anim), *cull*, *sort*, *draw*.

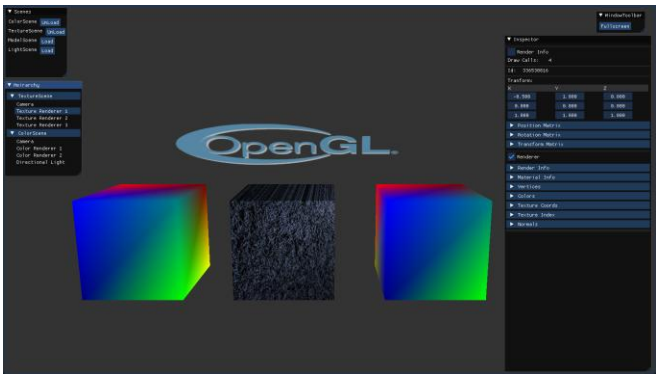
El proyecto contempla el desarrollo de un entorno virtual 3D completo e interactivo, implementado en OpenGL, que permita al usuario recorrer y observar un espacio cultural inspirado en elementos prehispánicos y lucha libre mexicana. El sistema incluye:

- Escena 3D completa con estructuras, decoraciones y ambientación temática.
- Navegación en primera persona, utilizando teclado y mouse, con desplazamiento libre por el entorno.
- Iluminación dinámica, incluyendo luz direccional, luces puntuales y linterna tipo “spotlight”.
- Texturas detalladas aplicadas a superficies naturales y arquitectónicas (piedra, arena, madera, metal).
- Modelos importados utilizando la librería Assimp para integrar objetos externos en formatos 3D comunes.
- Interacción básica mediante teclas para activar animaciones, cambiar iluminación o consultar puntos de interés.
- Documentación completa que incluye manual de usuario, descripción técnica del pipeline de render y especificación de controladores e interactividad.



El resultado final es una experiencia didáctica, visualmente atractiva y funcional, adecuada para fines académicos, presentación de proyecto y práctica en programación gráfica moderna

- Se desarrolla una escena interactiva que corre en tiempo real en PC de escritorio.
- Renderizado con OpenGL 3.3 Core, usando:
    - Un shader externo para modelos (modelLoading.vs/frag).
    - Un shader embebido (kVS/kFS) para geometría procedural, pasto y cielo.
- La escena incluye:
    - Objetos importados (formatos .obj) mediante la clase Model.
    - Objetos procedurales mediante cubos, planos y sólidos de revolución (mesa, silla, florero, flor, máscara de jade).
    - Sistema de iluminación direccional basado en la posición del sol.
    - Ciclo día/noche (sol sube/baja y cambia la iluminación).



- Animaciones paramétricas (perro que camina en trayectoria circular y pelota que rebota).

Limitantes

Para garantizar el cumplimiento del proyecto dentro del tiempo, recursos y nivel académico esperado, se establecen las siguientes limitaciones técnicas:

- **Sin físicas avanzadas**
  - No se implementa simulación física realista ni motores de físicas.
  - No hay gravedad realista, rebotes, fuerza o comportamientos dinámicos complejos.
- **Colisiones simples o inexistentes**
  - El usuario puede atravesar objetos del entorno en algunos casos.
  - No se implementan sistemas robustos de colisión o “bounding volumes”.
- **Shaders y materiales básicos**
  - Se utilizan modelos de iluminación Phong y/o Blinn-Phong, adecuados para nivel académico.
  - No se implementan técnicas avanzadas como PBR, SSAO, HDR completo o sombras dinámicas complejas.
- **Niveles de detalle optimizados para laboratorio**
  - Modelos con carga poligonal moderada.
  - Texturas comprimidas o reducidas para mantener un rendimiento adecuado.
- **Rendimiento orientado a equipos comunes**
  - La aplicación está pensada para correr correctamente en computadoras de laboratorio académico o laptops de nivel medio.
  - No está diseñada para GPU de alta gama ni para despliegue comercial.

En conjunto, estas limitaciones permiten acotar el proyecto, asegurar su funcionalidad estable y centrarse en los objetivos principales de modelado, renderizado, interacción e inmersión visual, sin sacrificar el desempeño general ni exceder el nivel esperado del curso.

- Limitado a un tipo de cámara (primera persona con movimiento libre).
- No se implementan:
  - Sombras proyectadas (shadow maps).
  - Post-procesado avanzado.
  - Física realista (colisiones, gravedad real); los movimientos son puramente matemáticos.

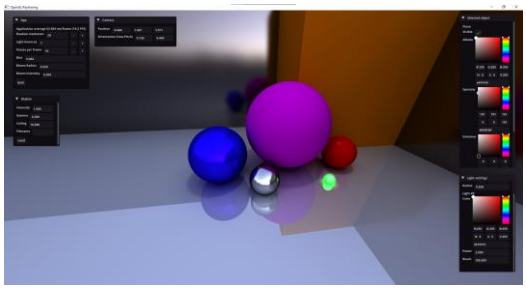
3. Requerimientos y Ambiente

Para ejecutar y compilar correctamente el entorno virtual 3D, se requiere la siguiente configuración de software, hardware y librerías gráficas:

Software de Desarrollo

- **Visual Studio 2022 (Compilación x64)**
  - Toolset: MSVC v143 o superior
  - Proyecto configurado para *Release x64*
  - Configuración de *linker* para incluir librerías externas y rutas de inclusión

API y Perfil Gráfico



- **OpenGL 3.3 Core Profile**
  - Uso de pipeline gráfico programable moderno
  - Eliminación de funciones fijas obsoletas
  - Compatibilidad con shaders GLSL 330

Librerías Externas Requeridas

Librería	Función
GLFW	Manejo de ventanas, contexto OpenGL y entrada del usuario
GLEW	Carga de extensiones OpenGL
GLM	Matemáticas gráficas (matrices, transformaciones, vectores)
stb_image	Carga de imágenes para texturas
Assimp	Importación de modelos 3D externos (OBJ, FBX, etc.)

Todas las librerías deben estar correctamente enlazadas y configuradas para arquitectura **x64**.

Requerimientos de Hardware

- GPU compatible con OpenGL 3.3+ y shaders GLSL 3.3
    - Intel HD Graphics 4000 o superior
    - NVIDIA GeForce Series ≥ 600
    - AMD Radeon Serie HD ≥ 7000
- Memoria RAM recomendada: 8 GB mínimo
- Procesador: Intel i5 / Ryzen 3 o superior

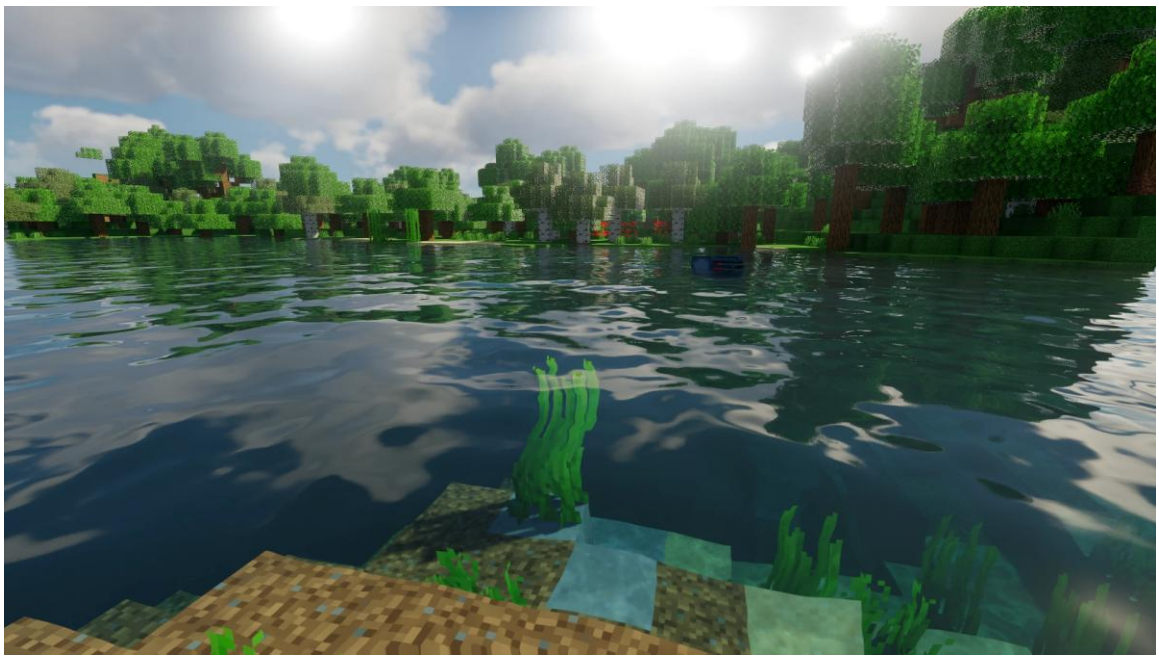
Compatibilidad del Sistema

- Sistemas Operativos soportados
    - Windows 10 / Windows 11

Rendimiento optimizado para equipos de laboratorio académico y laptops estándar.

Arquitectura general

- Proyecto de tipo Console Application en C++.
- **Estructura básica:**
    - **main.cpp**
        - **Configura GLFW/GLEW, crea la ventana, inicializa OpenGL.**
        - **Carga shaders externos (Shader).**
        - **Carga modelos (Model).**
        - **Construye geometría procedural (cubos, planos, florero).**
        - **Bucle principal: entrada → actualización → renderizado.**
    - **Shader.h / Shader.cpp**
      Manejo de shaders externos (carga, compilación, uso).
    - **Camera.h / Camera.cpp**
      Cámara en primera persona (posición, yaw, pitch, zoom).
    - **Model.h / Model.cpp**
      Carga de modelos mediante Assimp (mallas, texturas, materiales).
- **Shaders embebidos:**
    - kVS y kFS definidos como cadenas en main.cpp para gProg (procedural).
    - Shader externo de modelo: modelLoading.vs, modelLoading.frag.



Estructura de carpetas

Archivo	Función
Shader.h	Compila, enlaza y gestiona shaders (vertex y fragment).
Camera.h	Control FPS de movimiento (WASD) y rotación con mouse.
Model.h	Carga modelos externos mediante ASSIMP (.obj, .fbx, .dae).
Mesh.h	Maneja VBO/VAO y datos de vértices por malla.
main.cpp	Contiene la lógica de la escena y el bucle principal.

COMPUTADORAS UTILIZADAS:

5. Metodología de Desarrollo

• **Inicialización del entorno y proyecto en Visual Studio.**
Se configuró un proyecto en C++ utilizando Visual Studio 2022 y OpenGL Moderno (Core Profile), integrando librerías como GLFW, GLEW, GLM, stb_image y Assimp. Se preparó la estructura del proyecto asegurando compatibilidad con OpenGL 3.3+ y archivos auxiliares como Shader.h, Camera.h y Model.h.

• **Geometría del escenario.**
Se construyeron los elementos principales del entorno 3D mediante transformaciones y modelos importados: pirámides, plataformas, ring de lucha, globo aerostático y superficies del terreno. Se aplicaron matrices de traslación, rotación y escala utilizando GLM para posicionar cada elemento en el espacio.

• **Texturizado y materiales.**
Se aplicaron texturas mediante mapeo UV para simular piedra, arena, metal y otros materiales. Se configuraron propiedades materiales (ambient, diffuse, specular, shininess) para lograr superficies con diferente respuesta a la luz, mejorando el realismo del entorno.

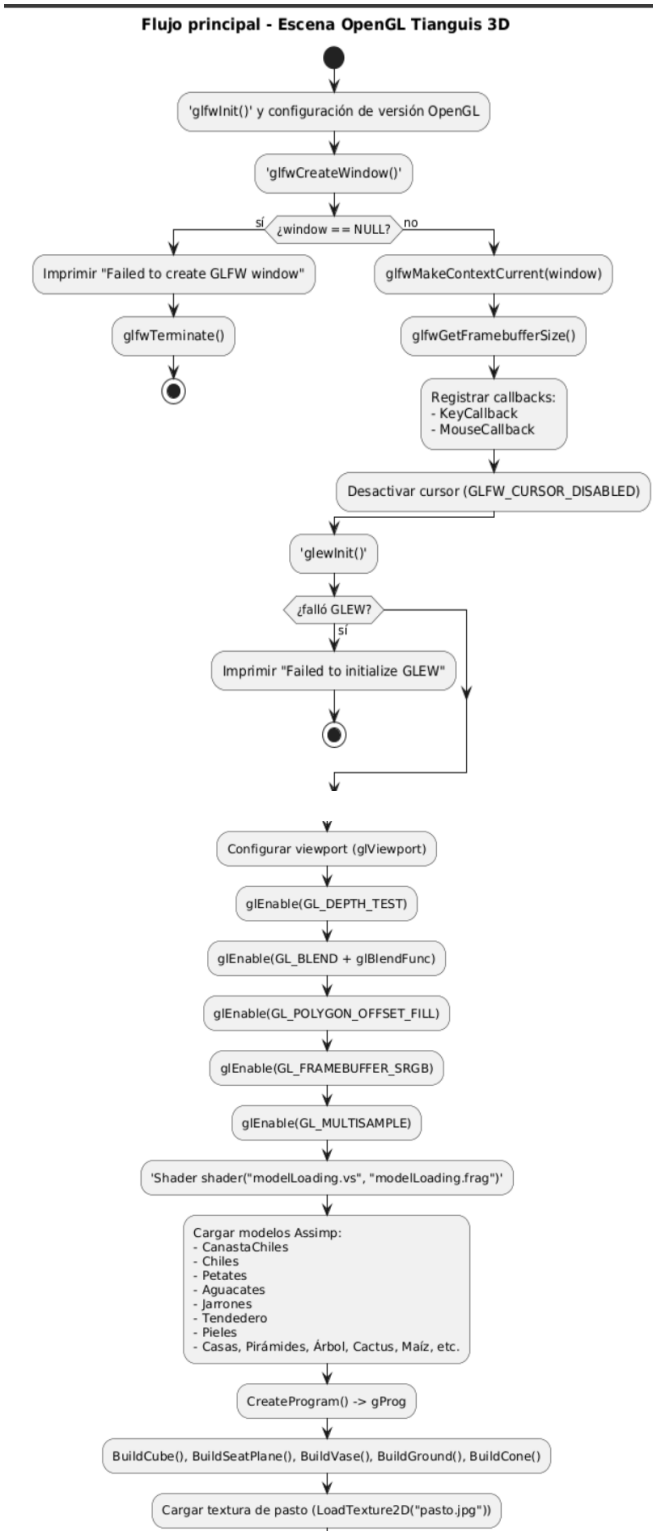
• **Iluminación y configuración de cámaras.**
Se implementó iluminación dinámica estilo Phong/Blinn-Phong, incluyendo luz ambiental, difusa y especular, así como un modo de linterna (spotlight) activado por el usuario. La cámara fue configurada en modo primera persona (FPS) con movimiento WASD y control de vista mediante mouse.

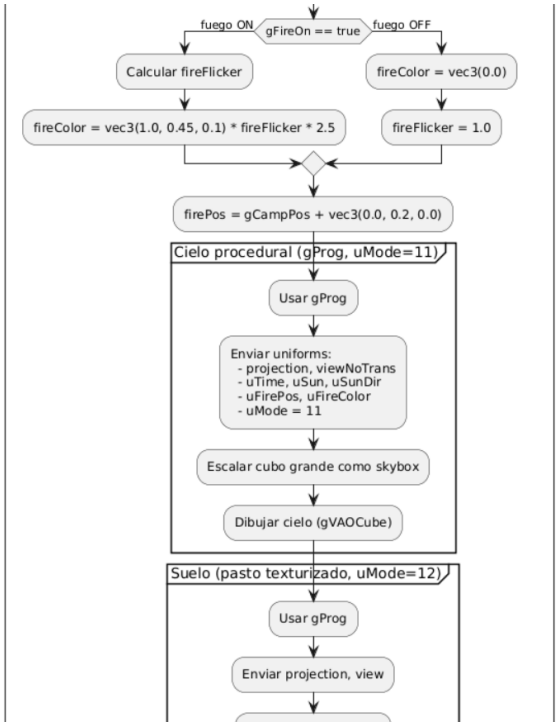
• **Carga de modelos y organización de recursos.**
Se integró Assimp para cargar modelos 3D (OBJ/FBX). Los recursos del proyecto se organizaron en carpetas (Models, Textures, Shaders) para asegurar una correcta lectura en tiempo de ejecución. Se validaron rutas relativas para evitar errores al ejecutar el programa.

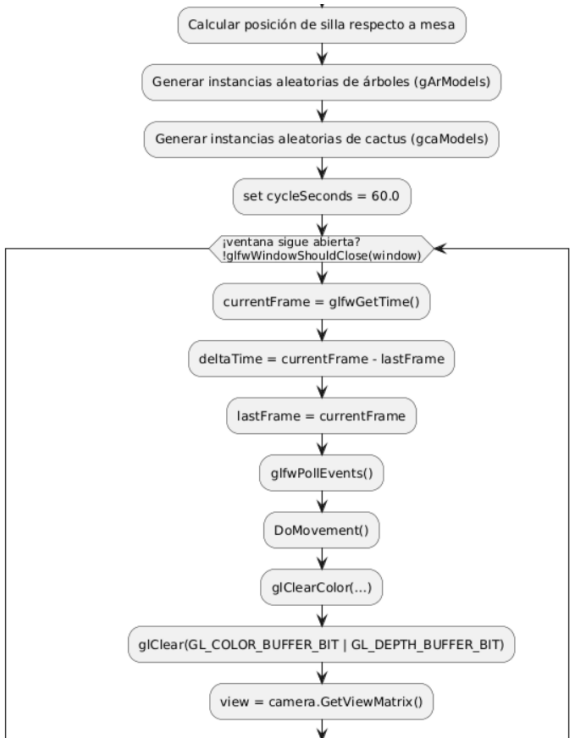
• **Animación básica y recorrido.**
Se añadieron animaciones ligeras usando deltaTime, como movimientos y rotaciones de objetos. El usuario puede recorrer todo el entorno libremente observando iluminación, materiales, dimensiones y profundidad espacial, simulando una experiencia de exploración en primera persona.

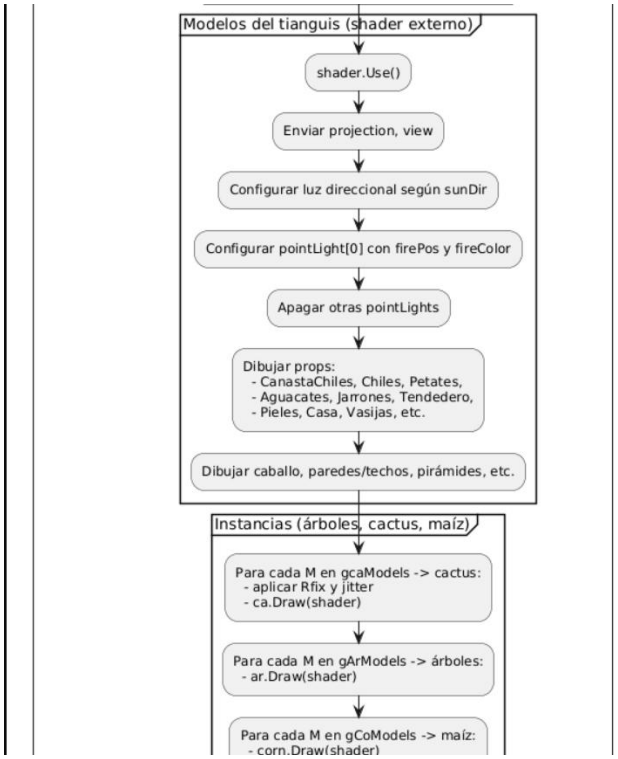
• **Pruebas, optimización y documentación.**
Se realizaron pruebas para validar rendimiento gráfico, carga de texturas/modelos y estabilidad de la cámara. Se optimizaron recursos visuales, texturas y draw calls para mantener fluidez. Finalmente, se documentó el proyecto en este manual para guiar al usuario final.

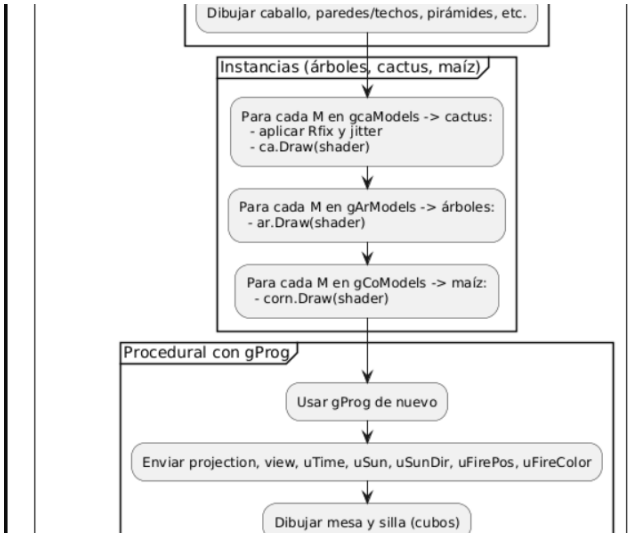
6. Diagrama de Flujo del Software




fuego ON
gFireOn == true
fuego OFF
Calcular fireFlicker
fireColor = vec3(0.0)
fireColor = vec3(1.0, 0.45, 0.1) * fireFlicker * 2.5
fireFlicker = 1.0
firePos = gCampPos + vec3(0.0, 0.2, 0.0)
Cielo procedural (gProg, uMode=11)
Usar gProg
Enviar uniforms:
- projection, viewNoTrans
- uTime, uSun, uSunDir
- uFirePos, uFireColor
- uMode = 11
Escalar cubo grande como skybox
Dibujar cielo (gVAOCube)
Suelo (pasto texturizado, uMode=12)
Usar gProg
Enviar projection, view


Calcular posición de silla respecto a mesa
Generar instancias aleatorias de árboles (gArModels)
Generar instancias aleatorias de cactus (gcaModels)
set cycleSeconds = 60.0
¡ventana sigue abierta?
!glfwWindowShouldClose(window)
currentFrame = glfwGetTime()
deltaTime = currentFrame - lastFrame
lastFrame = currentFrame
glfwPollEvents()
DoMovement()
glClearColor(...)
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
view = camera.GetViewMatrix()





7. Plan y Diagrama de Gantt

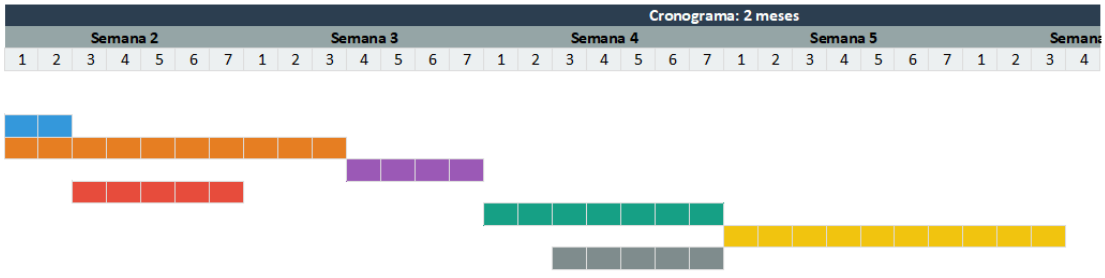
Cronograma (8 semanas):

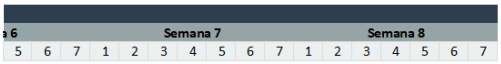
Semana	Actividad	Entregable
1	Ambiente VS, librerías e includes/lib	Proyecto compila

2	Geometría base (pirámides, árboles etc.)	Escena base
3	Texturas y materiales	Texturas aplicadas
4	Iluminación (ambiental, difusa, especular)	Shader lighting
5	Modelos externos (Assimp)	Modelos visibles
6	Animación y cámara WASD+mouse	Recorrido funcional
7	Integración cultural/ambientación	Escena final
8	Pruebas, optimización y documentación	PDF/Word finales

Diagrama de Gantt - Proyecto 3D OpenGL (Incluye Modelado Blender)

Actividad	Responsable	Inicio	Fin	Semana 1						
				1	2	3	4	5	6	7
Instalación VS, OpenGL, GLEW, GLFW, Assimp	Amy	S1, d1	S1, d7							
Organización de proyecto	Ceci	S1, d3	S2, d2							
Modelado en Blender	Equipo	S2, d1	S3, d3							
Exportación y corrección UV en Blender	Amy	S3, d4	S3, d7							
Render + ventana + contexto	Ceci	S2, d3	S2, d7							
Integración modelos Blender → OpenGL (Assimp)	Amy	S4, d1	S4, d7							
Shader Phong + luces	Ceci	S5, d1	S6, d3							
Texturas y blending	Amy	S4, d3	S4, d7							
Cámara FPS + animación	Ceci	S7, d1	S7, d7							
Estrellas + efectos + theme MX	Equipo	S7, d3	S8, d2							
Pruebas + optimización	Equipo	S8, d1	S8, d4							
Documentación + exposición	Equipo	S8, d3	S8, d7							





Documentación

El proyecto está organizado alrededor de un archivo principal (main.cpp) que hace lo siguiente:

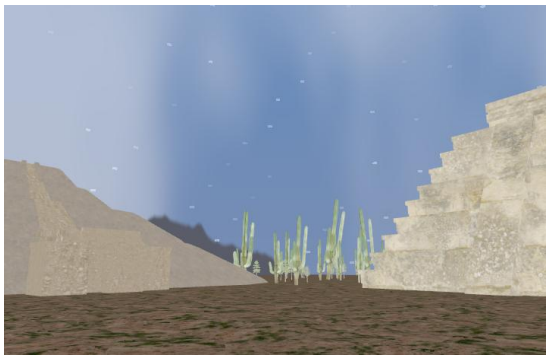
• Inicializa GLFW y GLEW para usar OpenGL moderno (versión core).

• Crea una ventana y configura el contexto de render.

• Carga shaders (vertex/fragment) y modelos externos (.obj).

• Configura la cámara en primera persona (WASD + mouse).

• Entra en un bucle principal donde se actualiza el tiempo, la iluminación, se procesa la entrada de usuario y se dibuja la escena completa.



```
// ====== INICIALIZACIÓN GLFW / GLEW ======
glfwInit();
glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

GLFWwindow* window = glfwCreateWindow(1280, 720, "Tianguis 3D", nullptr, nullptr);
glfwMakeContextCurrent(window);

// Inicializar GLEW
glewExperimental = GL_TRUE;
glewInit();

// Habilitar profundidad
glEnable(GL_DEPTH_TEST);
```

En este bloque se indica qué versión de OpenGL se va a usar (3.3 core), se crea la ventana con el título "Tianguis 3D" y se activa el test de profundidad. Sin el depth-test, los objetos se dibujarían sin respetar qué está delante y qué está detrás.

2. Bucle principal de renderizado

```
// ====== BUCLE PRINCIPAL ======
while (!glfwWindowShouldClose(window)) {
```

```
        float currentFrame = glfwGetTime();
        deltaTime = currentFrame - lastFrame;
        lastFrame = currentFrame;

        glfwPollEvents();
        DoMovement(); // Procesa WASD

        // Limpiar buffers
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        // Matrices de cámara
        glm::mat4 view = camera.GetViewMatrix();
        glm::mat4 projection = glm::perspective(glm::radians(camera.Zoom),
                        (float)SCR_WIDTH / (float)SCR_HEIGHT,
                        0.1f, 1000.0f);

        shader.Use();
    glUniformMatrix4fv(glGetUniformLocation(shader.Program, "view"), 1, GL_FALSE, glm::value_ptr(view));
    glUniformMatrix4fv(glGetUniformLocation(shader.Program, "projection"), 1, GL_FALSE,
glm::value_ptr(projection));

        // Aquí se dibuja toda la escena
(suelo, mesa, silla, modelos, fogata, etc.)
        drawScene(shader);

        glfwSwapBuffers(window);
}
```

En cada iteración del bucle se calcula deltaTime (para que el movimiento de la cámara sea independiente del número de FPS), se leen los eventos de entrada (teclado y mouse), se limpia la pantalla, se actualizan las matrices de vista y proyección y finalmente se llama a una función de alto nivel (drawScene) que se encarga de dibujar todos los objetos de la escena.

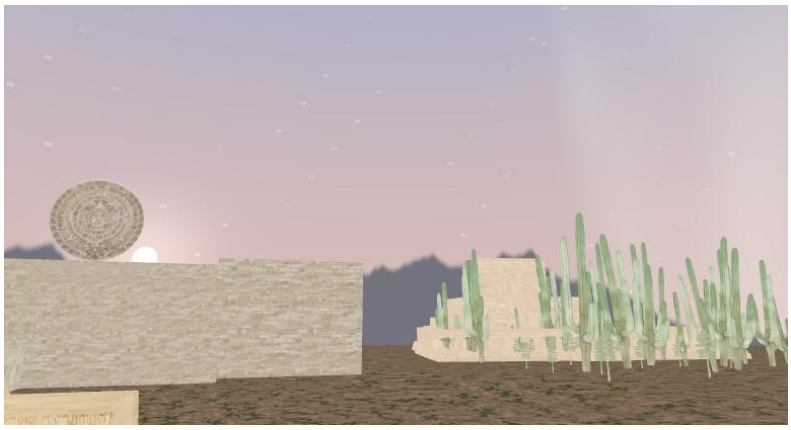
3. Cámara en primera persona (WASD + Mouse)

```
// ====== CALLBACK DEL MOUSE ======
void MouseCallback(GLFWwindow* window, double xpos, double ypos) {
        if(firstMouse) {
        lastX = xpos;
        lastY = ypos;
        firstMouse = false;
        }

        float xoffset = xpos - lastX;
        float yoffset = lastY - ypos;
        lastX = xpos;
        lastY = ypos;

        camera.ProcessMouseMovement(xoffset, yoffset);
}

// ====== MOVIMIENTO WASD ======
void DoMovement() {
        if(keys[GLFW_KEY_W]) camera.ProcessKeyboard(FORWARD,  deltaTime);
```

```
        if(keys[GLFW_KEY_S]) camera.ProcessKeyboard(BACKWARD, deltaTime);
        if(keys[GLFW_KEY_A]) camera.ProcessKeyboard(LEFT,     deltaTime);
        if(keys[GLFW_KEY_D]) camera.ProcessKeyboard(RIGHT,    deltaTime);
}
```

El callback del mouse toma la posición actual del cursor y calcula cuánto se movió en X y Y. Con esos offsets se actualizan yaw y pitch de la cámara, logrando el efecto de girar la vista.

La función DoMovement revisa qué teclas están activas (W, A, S, D) y manda instrucciones a la cámara para moverse hacia adelante, atrás o a los lados. Se multiplica por deltaTime para que la velocidad de la cámara sea constante sin importar los FPS.



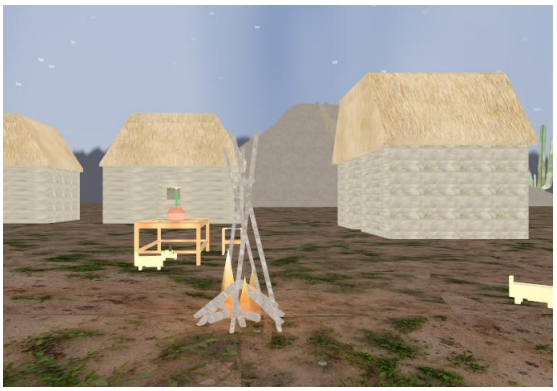
4. Dibujo procedural: cubos y objetos compuestos

```
// ====== FUNCIÓN PARA DIBUJAR CUBOS ======
void drawCubeAt(glm::vec3 pos, glm::vec3 scale, int mode) {
        glm::mat4 model(1.0f);
        model = glm::translate(model, pos);
        model = glm::scale(model, scale);

    glUniformMatrix4fv(glGetUniformLocation(gProg, "model"), 1, GL_FALSE, glm::value_ptr(model));
    glUniform1i(glGetUniformLocation(gProg, "uMode"), mode);

        glBindVertexArray(gVAOCube);
        glDrawArrays(GL_TRIANGLES, 0, gCubeVerts);
}
```

Esta función es la base para construir muchos objetos de la escena. Recibe una posición, una escala y un modo de material. Dentro, arma la matriz model con traslación y escala, la manda al shader y luego dibuja la geometría del cubo. Cambiando pos y scale, el mismo cubo se reutiliza como pata de mesa, tablero, parte del asiento, etc.

5. Ejemplo completo: construcción de la mesa

```
// ====== MESA ======
{
        float legH = 1.0f;
        float topX = 2.5f, topY = 0.15f, topZ = 1.3f;

        // Tablero de la mesa
        drawCubeAt(gTablePos + glm::vec3(0.0f, legH + topY * 0.5f, 0.0f),
        glm::vec3(topX, topY, topZ), 1);

        // Cálculo de offsets para las patas
        float ox = topX * 0.5f - 0.09f * 0.5f;
        float oz = topZ * 0.5f - 0.09f * 0.5f;

        // Cuatro patas
        drawCubeAt(gTablePos + glm::vec3(+ox, legH *
0.5f, +oz), glm::vec3(0.09f, legH, 0.09f), 1);
        drawCubeAt(gTablePos + glm::vec3(-ox, legH *
0.5f, +oz), glm::vec3(0.09f, legH, 0.09f), 1);
        drawCubeAt(gTablePos + glm::vec3(+ox, legH *
0.5f, -oz), glm::vec3(0.09f, legH, 0.09f), 1);
        drawCubeAt(gTablePos + glm::vec3(-ox, legH *
0.5f, -oz), glm::vec3(0.09f, legH, 0.09f), 1);
}
```



La mesa se construye a partir de un solo tipo de geometría (cubo). Primero se dibuja el tablero usando una escala grande en X y Z, pero pequeña en Y. Después se calculan offsets ox y oz para colocar las patas en las cuatro esquinas. Si se cambia topX, topY o topZ, toda la mesa cambia de tamaño sin tener que reescribir las coordenadas a mano.

6. Modelos externos (.obj) con Assimp

```
// ====== CARGA DE MODELOS ======
Shader shader("Shader/modelLoading.vs", "Shader/modelLoading.frag");

Model CanastaChiles((char*)"Models/CanastaChiles.obj");
Model Chiles((char*)"Models/Chiles.obj");
Model PetatesTianguis((char*)"Models/PetatesTianguis.obj");
```

Cada objeto de tipo Model representa un archivo externo exportado desde Blender u otro software 3D. La clase Model, usando Assimp, se encarga de leer la malla, las normales, las coordenadas de textura y las texturas asociadas. El programador solo necesita crear la instancia del modelo y dibujarla en la escena.

```
// ====== DIBUJAR UN MODELO EN LA ESCENA ======
glm::mat4 mCanasta(1.0f);
mCanasta = glm::translate(mCanasta, glm::vec3(2.0f, 0.0f, -3.0f));
mCanasta = glm::scale(mCanasta, glm::vec3(0.4f));

glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE,
glm::value_ptr(mCanasta));
CanastaChiles.Draw(shader);
```

Para colocar el modelo en la escena se arma otra vez una matriz model con translate y scale, se envía al shader y luego se llama a Draw(shader). Esto permite ajustar la posición y el tamaño de la canasta sin modificar el archivo .obj.

7. Fogata animada con flicker e iluminación

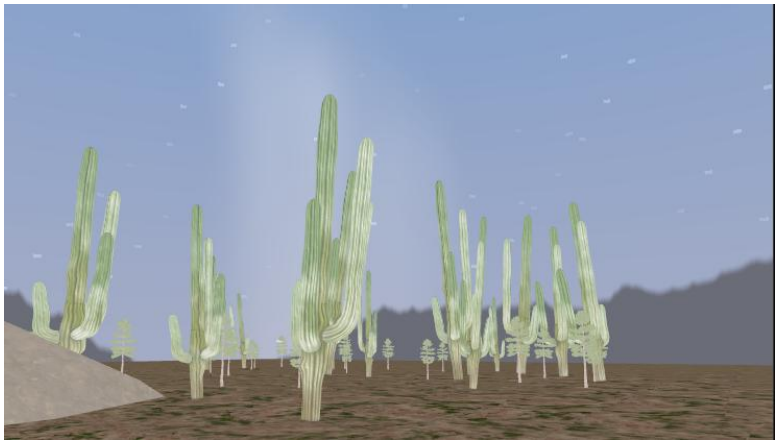
```
// ====== ANIMACIÓN DEL FUEGO ======
float t = glfwGetTime();
float flick = 0.5f + 0.5f * sin(t * 6.0f);
float fireScale = 0.6f + 0.15f * flick;

glm::mat4 fireModel(1.0f);
fireModel = glm::translate(fireModel, gCampFirePos);
fireModel = glm::scale(fireModel, glm::vec3(fireScale, fireScale *
1.8f, fireScale));

glUniformMatrix4fv(glGetUniformLocation(shader.Program,
"model"), 1, GL_FALSE, glm::value_ptr(fireModel));
FuegoCocina.Draw(shader);

// Parámetros de luz de la fogata
glUniform3f(glGetUniformLocation(shader.Program, "firePos"),
gCampFirePos.x, gCampFirePos.y, gCampFirePos.z);
glUniform3f(glGetUniformLocation(shader.Program, "fireColor"),
1.0f, 0.6f, 0.2f);
glUniform1f(glGetUniformLocation(shader.Program, "fireFlicker"), flick);
```

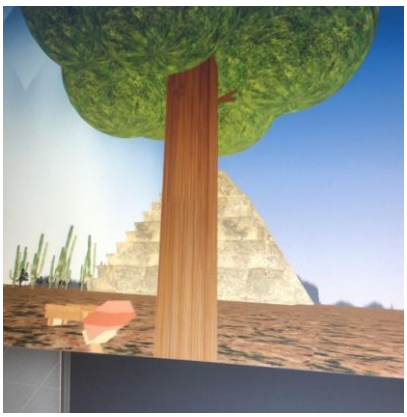
El parámetro t usa glfwGetTime() para obtener el tiempo en segundos desde que inició el programa. Con un seno de alta frecuencia se calcula flick, que oscila entre 0 y 1. Esa oscilación modifica la escala del modelo para simular que las llamas crecen y se encogen. Además, se pasan al shader uniforms que representan la posición, el color y la intensidad variable de la luz de la fogata.

En el fragment shader, fireFlicker se puede usar para aumentar o disminuir el brillo difuso y especular de la luz, logrando una iluminación dinámica que cambia fotograma a fotograma.

8. Instanciación de árboles y cactus con posiciones aleatorias

```
// ====== GENERACIÓN DE POSICIONES PARA CACTUS ======
std::vector<glm::vec3> cactusPositions;
std::mt19937 rng(1234);
std::uniform_real_distribution<float> distX(-80.0f, 80.0f);
std::uniform_real_distribution<float> distZ(-80.0f, 80.0f);

for (int i = 0; i < 60; ++i) {
        float x = distX(rng);
        float z = distZ(rng);

        // Evitar poner cactus muy cerca de la mesa (radio de exclusión)
        if (glm::length(glm::vec2(x, z) - glm::vec2(gTablePos.x, gTablePos.z)) < 10.0f)
        continue;

        cactusPositions.push_back(glm::vec3(x, 0.0f, z));
}

// ====== DIBUJAR CACTUS ======
for (auto& pos : cactusPositions) {
        glm::mat4 m(1.0f);
        m = glm::translate(m, pos);
        m = glm::scale(m, glm::vec3(0.9f));
   glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(m));
        Cactus.Draw(shader);
}
```





Aquí se usa un generador de números aleatorios para repartir cactus por toda la escena. Se define un rango en X y Z, y se descartan las posiciones que caen dentro de un radio de 10 unidades alrededor de la mesa. Esto crea un entorno más natural y evita que los cactus aparezcan encima de objetos importantes. Luego se dibujan todos los cactus recorriendo el vector cactusPositions.

9. Ciclo día/noche y luz direccional del sol

```
// ====== CÁLCULO DE LA DIRECCIÓN DEL SOL ======
float timeOfDay = fmod(glfwGetTime() * 0.05f, 2.0f * 3.14159f);
glm::vec3 sunDir = glm::normalize(glm::vec3(cos(timeOfDay), sin(timeOfDay), 0.2f));

glUniform3fv(glGetUniformLocation(shader.Program, "sunDir"), 1, glm::value_ptr(sunDir));
```

Se usa una variable timeOfDay que avanza lentamente con el tiempo y se limita a un ciclo de 0 a 2π. Aplicando coseno y seno se obtiene un vector que describe una trayectoria circular en el cielo. Ese vector se normaliza y se pasa al shader como sunDir. En el shader, sunDir se usa para calcular la luz direccional del sol (o la luna, dependiendo de la etapa del ciclo).

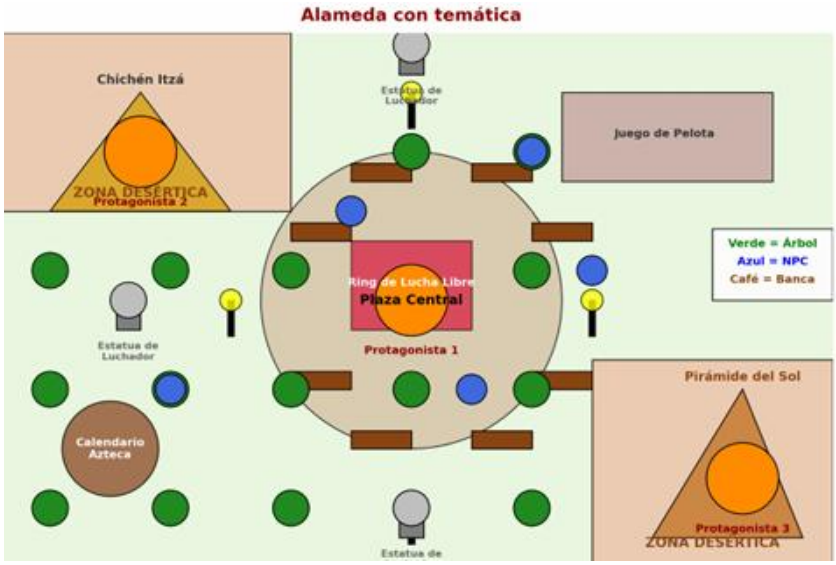
Cuando sin(timeOfDay) es positivo, el sol está por encima del horizonte (es de día). Cuando es negativo, se puede considerar que es de noche y activar estrellas, aumentar la intensidad de la fogata y atenuar la luz ambiental global.

11. Diseño del Escenario 3D

Geometría: pirámides escalonadas, calendario azteca circular cuadrado, árboles low-poly y estrellas.

El diseño del escenario 3D del proyecto “Tianguis Prehispánico” combina elementos culturales mesoamericanos, geometría procedural, modelos importados, iluminación dinámica y técnicas avanzadas de shading para lograr una representación visual coherente, amplia y funcional dentro de un motor gráfico moderno basado en OpenGL 3.3 Core.

A diferencia de un simple entorno estático, este escenario fue construido intencionalmente como un sistema interactivo que integra movimiento, animación, simulación de luz, ciclo día–noche, distribución aleatoria de objetos y estética low-poly estilizada. Cada elemento visual del ambiente cumple una función estructural, estética o interactiva dentro de la composición general.



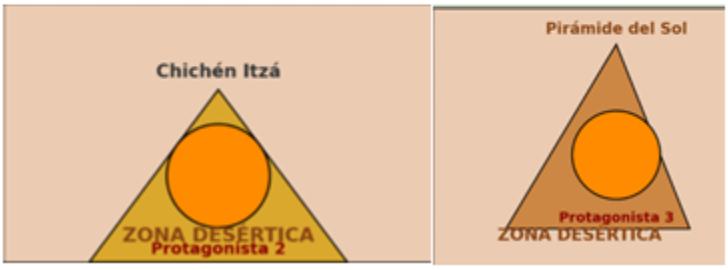
Descripción de cada elemento:

Árboles (Verde)

Se colocaron árboles alrededor de la alameda para representar la naturaleza y la ambientación de un espacio público realista. La distribución fue pensada para que hubiera áreas con sombra y áreas despejadas, siguiendo el lineamiento de usar simbología clara y un plano funcional.

Chichén Itzá y Pirámide del Sol

Se incluyeron dos de las construcciones más representativas de la cultura prehispánica: Chichén Itzá (maravilla del mundo moderno) y la Pirámide del Sol (Teotihuacán). Ambas se colocaron en zonas desérticas para diferenciar su ambiente y reforzar la simbología prehispánica.

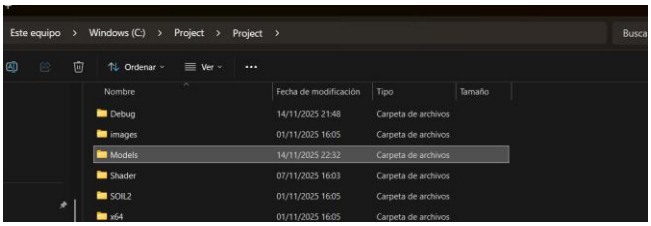


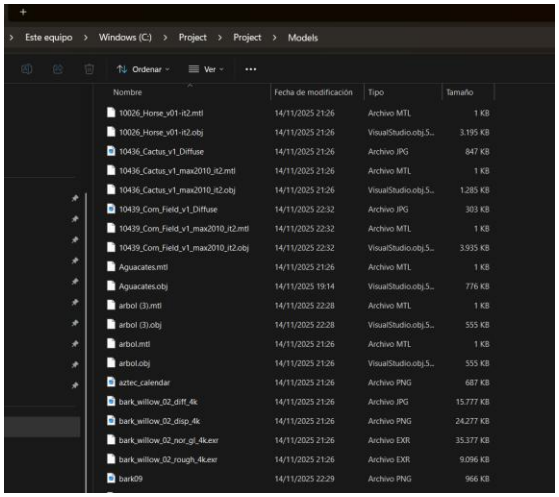
INSPIRACION

12. Pruebas y Validación

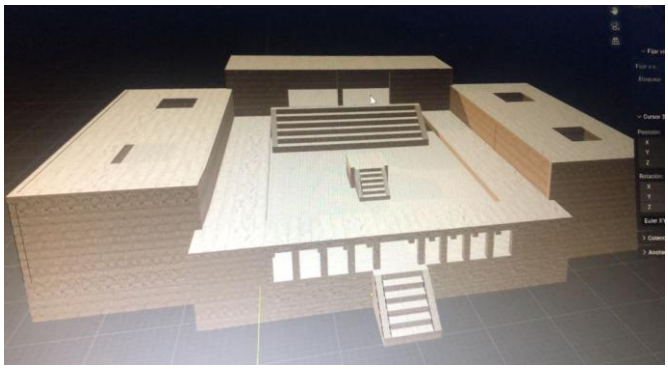
Casos: arranque del programa, carga de shaders, cámara, render de modelos, texturas RGBA con transparencia, luces, rendimiento (>30 FPS).

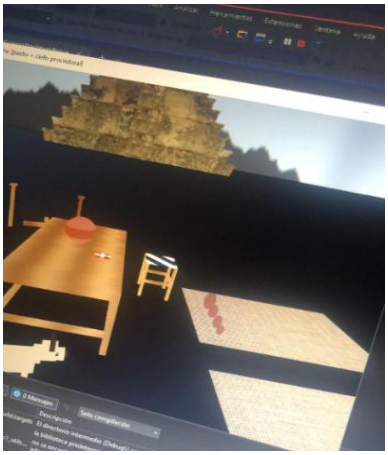
Estrategia: pruebas por módulo y recorrido completo; revisión de rutas relativas y Copy to Output Directory.






```
C:\Project\Debug\Project.exe (proceso 17880) se cerró con el código 0 (0x0).
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->
Cerrar la consola automáticamente al detenerse la depuración.
Presione cualquier tecla para cerrar esta ventana. . .
```





13. Gestión de Riesgos y Mantenimiento

El proyecto “Tianguis Prehispánico” requiere una gestión de riesgos activa debido a la combinación de modelos externos, shaders personalizados, animaciones procedurales y dependencias múltiples de librerías externas (GLFW, GLEW, GLM, Assimp, SOIL2). A lo largo del ciclo de vida del software se identificaron riesgos relacionados con configuración, rutas de recursos, funcionamiento en distintos equipos y estabilidad de la build. Uno de los problemas más frecuentes es el uso de rutas incorrectas para modelos, texturas o shaders, lo cual provoca errores de carga al ejecutar el programa. También es común la ausencia de DLLs esenciales, especialmente en instalaciones nuevas donde el entorno de ejecución no incluye las librerías requeridas por Assimp, SOIL2 o GLEW. Otro riesgo relevante es la presencia de drivers antiguos o desactualizados, que pueden afectar la correcta representación del cielo procedural, la iluminación dinámica o incluso desactivar características del pipeline moderno de OpenGL, generando diferencias visuales notables entre diferentes computadoras.

14. Conclusiones y Trabajo Futuro

El desarrollo de la escena “tianguis prehispánico” en OpenGL permitió integrar en un solo proyecto varios de los temas centrales de la materia de Computación Gráfica: geometría procedimental, carga de modelos con Assimp, iluminación dinámica, texturizado avanzado y cámara interactiva en 3D. La combinación de elementos modelados en Blender (pirámides, chozas, maíz, caballo, juego de pelota, props del tianguis) con objetos generados por código (mesa, silla, florero, flor, fogata, máscara de jade, perrito voxel, carretilla, hacha, pelota animada) demuestra que es posible construir un entorno complejo mezclando contenido artístico y contenido totalmente paramétrico.

La implementación del cielo procedural con ciclo día/noche, el cálculo de la dirección del sol y el uso de un shader embebido para el cielo y el pasto texturizado con anti-tiling, permitió obtener una atmósfera visualmente rica sin depender de skyboxes prehorneados. El uso de ruido, FBM y funciones personalizadas para nubes, estrellas, sol y luna dio como resultado un fondo dinámico que cambia en el tiempo de forma suave y coherente.

Por otro lado, el sistema de iluminación de la fogata, controlado mediante la tecla F, integró tanto una luz puntual para los modelos (shader externo) como una contribución emisiva y de iluminación local en el shader procedural. Esto reforzó el entendimiento de cómo combinar varias fuentes de luz (direccional y puntual), cómo atenuarlas con la distancia y cómo reutilizar la misma información (posición y color de la fogata) en distintos shaders para mantener consistencia visual en toda la escena.

Las instancias aleatorias de árboles, cactus y maizales, con exclusiones alrededor de áreas importantes (mesa, campamento, pirámides, Tula, zona central), permitieron practicar técnicas de dispersión controlada en grandes terrenos, manteniendo composición visual y evitando solapamientos. La inclusión del perrito voxel animado, la pelota con rebote y la carretilla controlable con teclas añadió elementos lúdicos e interactivos que hacen que la escena no solo sea un “render estático”, sino un pequeño entorno casi de videojuego, donde el usuario puede explorar con la cámara y observar animaciones locales.

En conjunto, el proyecto cumplió el objetivo de integrar en un mismo escenario: modelos complejos, geometría procedural, técnicas de texturizado (incluyendo anti-tiling y anisotropía), iluminación dinámica dependiente del tiempo y un sistema de interacción en primera persona. Además, sirvió como experiencia completa de pipeline: desde modelado y exportación, pasando por carga, organización de la escena y diseño de shaders, hasta la documentación y análisis de la arquitectura del software.

Aunque la escena actual es funcional y visualmente rica, existen múltiples líneas de mejora que podrían abordarse en trabajos futuros:

Aunque la escena actual es funcional y visualmente atractiva, existen varias áreas de mejora que podrían explorarse en implementaciones posteriores:

- **Sombras dinámicas reales**, ya sea mediante shadow mapping tradicional o cascaded shadow maps para terrenos grandes.

- **Materiales PBR (Physically Based Rendering)** para mejorar realismo en madera, piedra, cerámica y metal.

- **Mejoras al motor interno**, como encapsular transformaciones en un sistema de entidades y componentes (ECS).
- **Iluminación global más avanzada**, incorporando ambient occlusion o iluminación hemisférica.
- **Animaciones más complejas**, incluyendo esqueletos, blending y curvas Bézier para trayectorias.
- **Audio 3D**, para dotar de ambiente al tianguis (fuego, pasos, animales, viento).
- **Sistema de colisiones y física ligera** que permita interacción con objetos.
- **Optimización del terreno**, mediante LOD adaptativo o tessellation shaders.
- **Interfaz gráfica (UI)** para controlar hora del día, clima, intensidad de luces o velocidad de animaciones.
- **Expansión del entorno**, añadiendo nuevos puestos, personajes o minijuegos.

En resumen, el proyecto actual representa una base sólida sobre la cual es posible seguir construyendo: tanto desde el punto de vista gráfico (sombras, PBR, efectos avanzados), como desde el punto de vista de diseño de juego (interacción, objetivos, narrativa) y de ingeniería de software (estructura del motor, rendimiento y limpieza de código).