National Autonomous University of Mexico

Engineering Faculty

Graphical Computing and Human Computer Interaction

Final Project

317142165

Teacher: Ing. Carlos Aldair Roman Balbuena

group: 05

Delivery Date: November 25, 2025

Semester: 2026 - 1

User Manual

The purpose of this manual is to provide a guide to users on the correct way to interact within the "Monster House" project. It explains, in a simple and detailed way, the use of the controls, the features available and the essential actions that allow the experience to be functional and immersive.

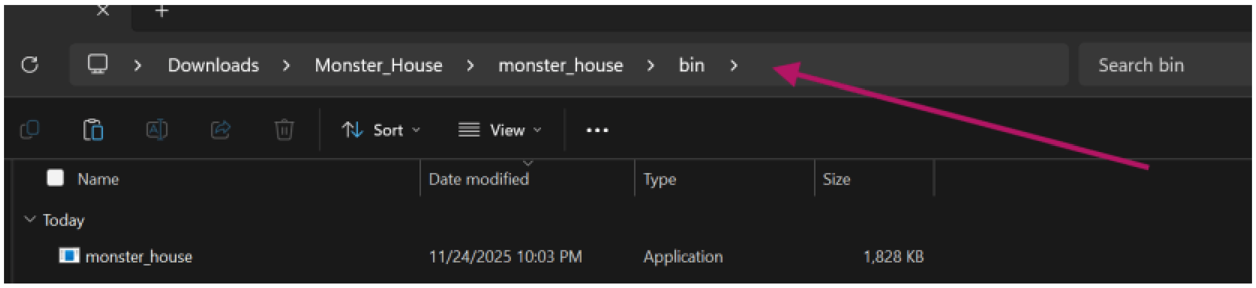
Minimum System Requirements

- Operating system: Windows 10 or higher (64-bit).
- Processor: Intel Core i3 or equivalent.
- RAM: 4 GB minimum.
- GPUs: Support for OpenGL 3.3 or higher.
- Recommended Resolution: 1366x768 or higher.

Note: Do not move or change the name of these folders, as they contain shaders and 3D models that are indispensable for operation.

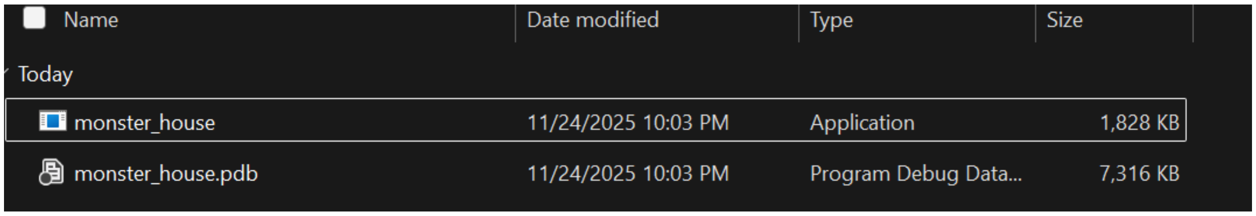
Instructions for running the environment

1. In the directory where the git clone was made, navigate to the directory shown in the image.



Name	Date modified	Type	Size
Today			
monster_house	11/24/2025 10:03 PM	Application	1,828 KB

2. Once the monster_house file with Application extension is located; run it with double click.



Name	Date modified	Type	Size
Today			
monster_house	11/24/2025 10:03 PM	Application	1,828 KB
monster_house.pdb	11/24/2025 10:03 PM	Program Debug Data...	7,316 KB

User controls:

Scrolling is managed using the following key mapping:

- W/Up Arrow: Advance. Move the character forward.
- S/Down Arrow: Go back. Move the character back.
- A: Move left (side scroll).
- D: Move to the right (side).
- Esc – Exit the program.

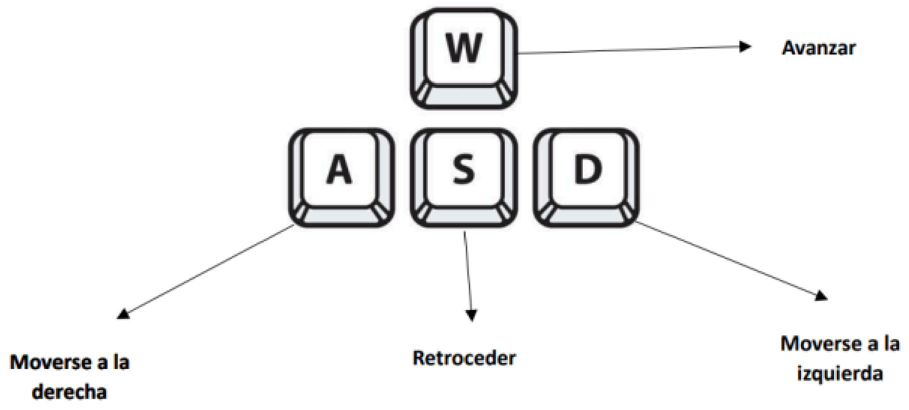


Figure 0. User Controls

Graphic Realism

- The virtual space takes reference to the house of the film "Monster House".

Facade:

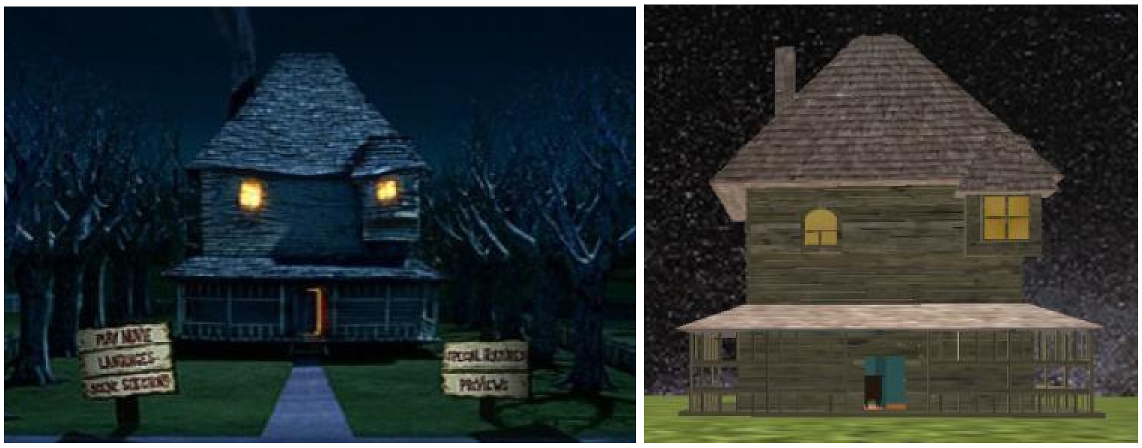


Figure 1 and 2. Picture the movie Monster House and facade at OpenGL.

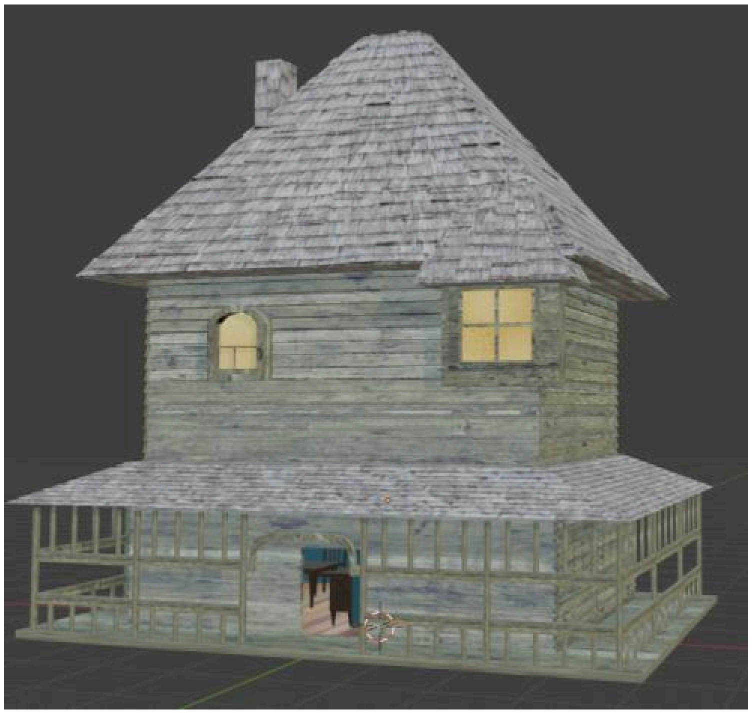


Figure 3. Facade modeled on Blender.

Fourth 1:

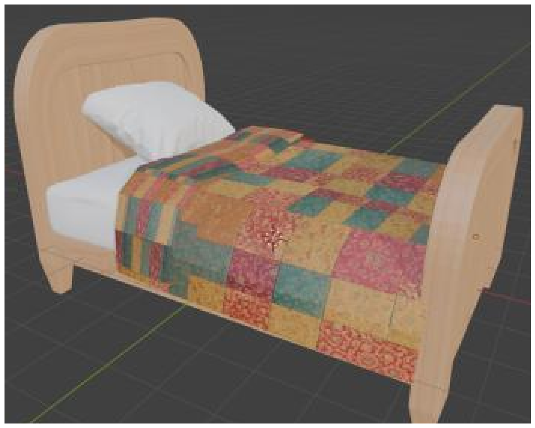


Figure 4. Bed patterned in Blender for the fourth 1.

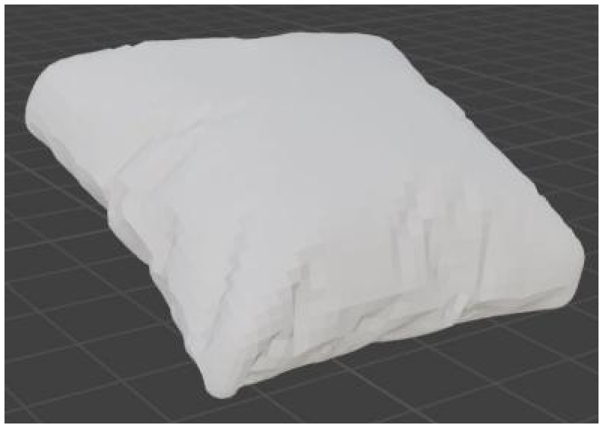


Figure 5. Pillow patterned in Blender for the fourth 1.

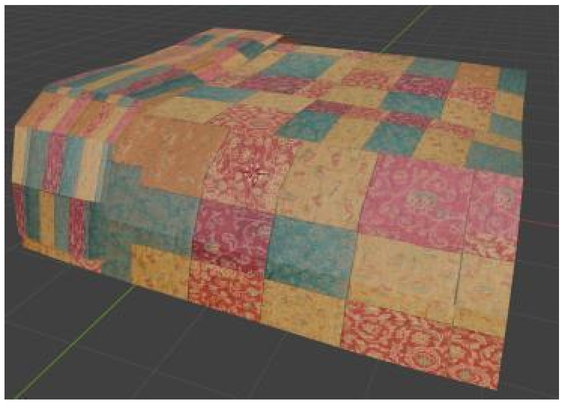


Figure 6. Patterned quilt in Blender for fourth 1.

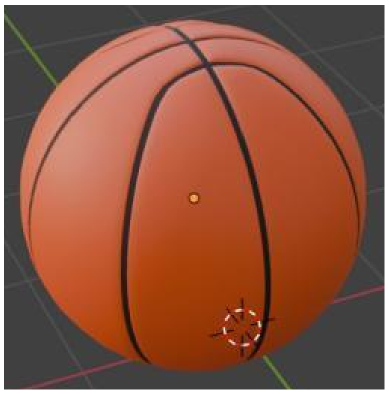


Figure 7. BasketBall ball modeled in blender for fourth 1

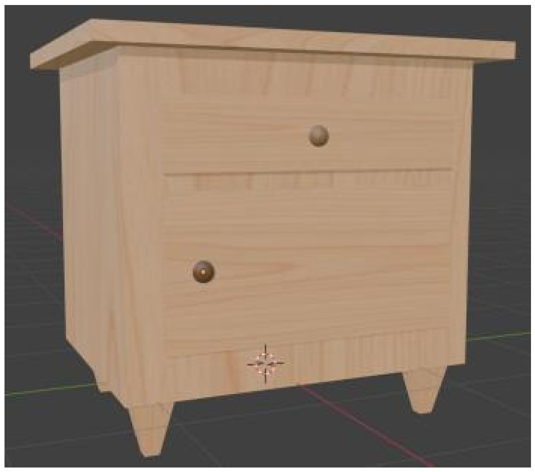


Figure 8. Furniture modeled in blender for the fourth 1.

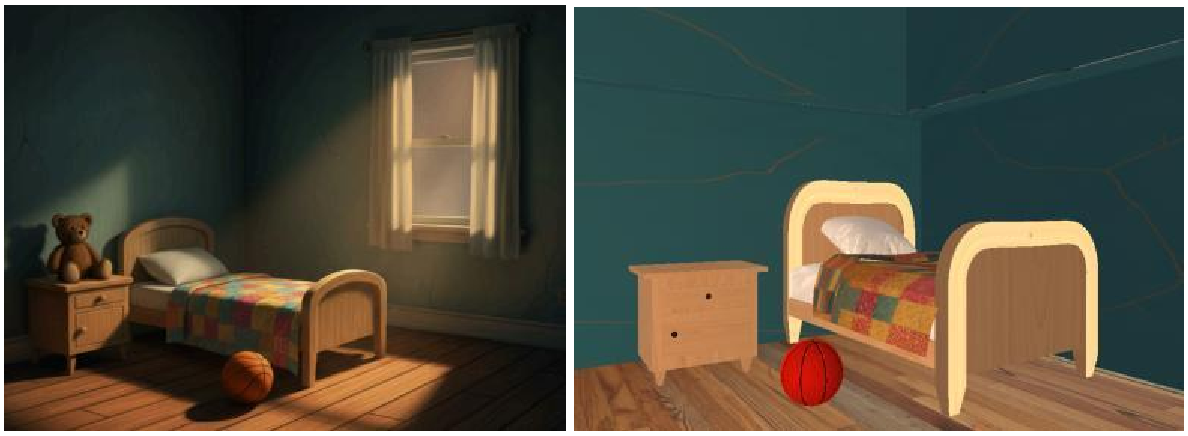


Figure 9 and 10. Reference image for the fourth 1 and fourth in OpenGL.

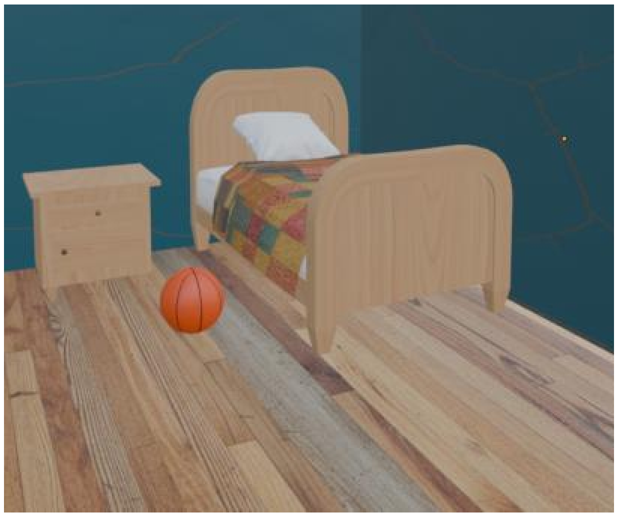


Figure 11. Full room modeled on Blender.

Dining Room:

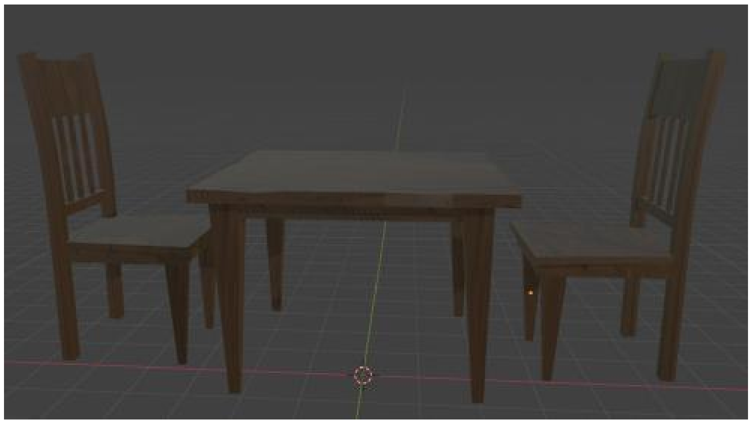


Figure 12. Chairs and table patterned in blender.

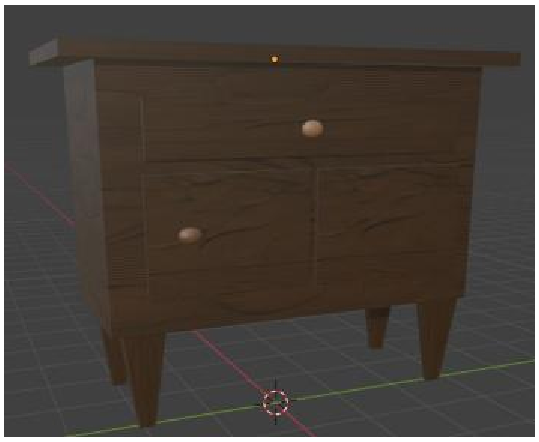


Figure 13. Mueble dining room patterned in Blender.

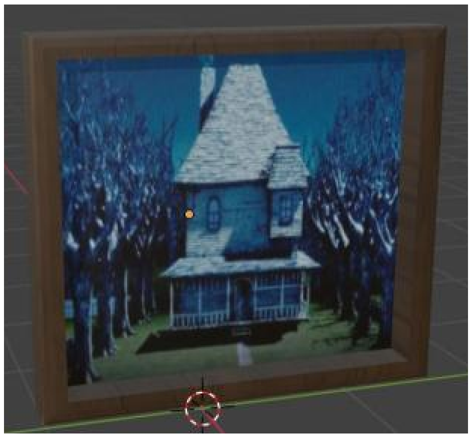


Figure 14. Picture modeled on Blender.

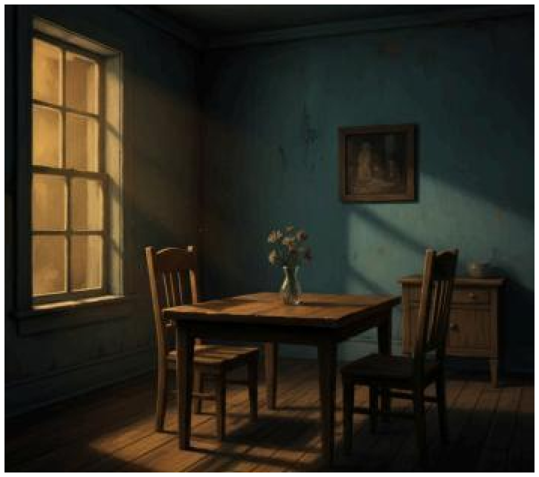
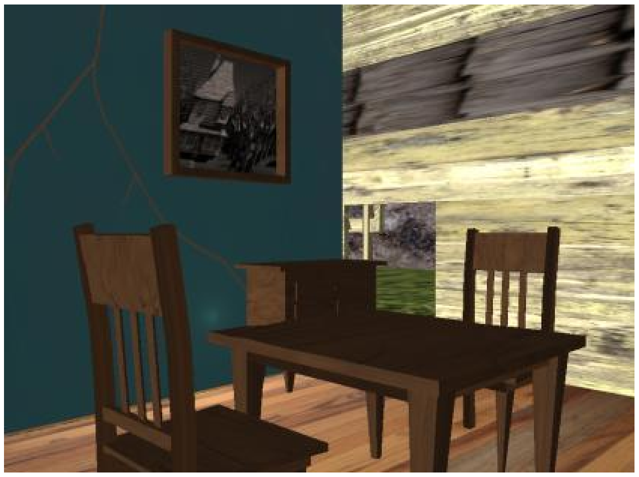



Figure 15 and 16. Make reference images for the dining room and dining room displayed at OpenGL.

The models were created and exported from Blender with integrated textures. The texture mapping process depends on the model loader (Model class) being able to find the image file on disk. The 3D model stores the texture path, and if this path is incorrect (the most common problem), the object will appear without texture.

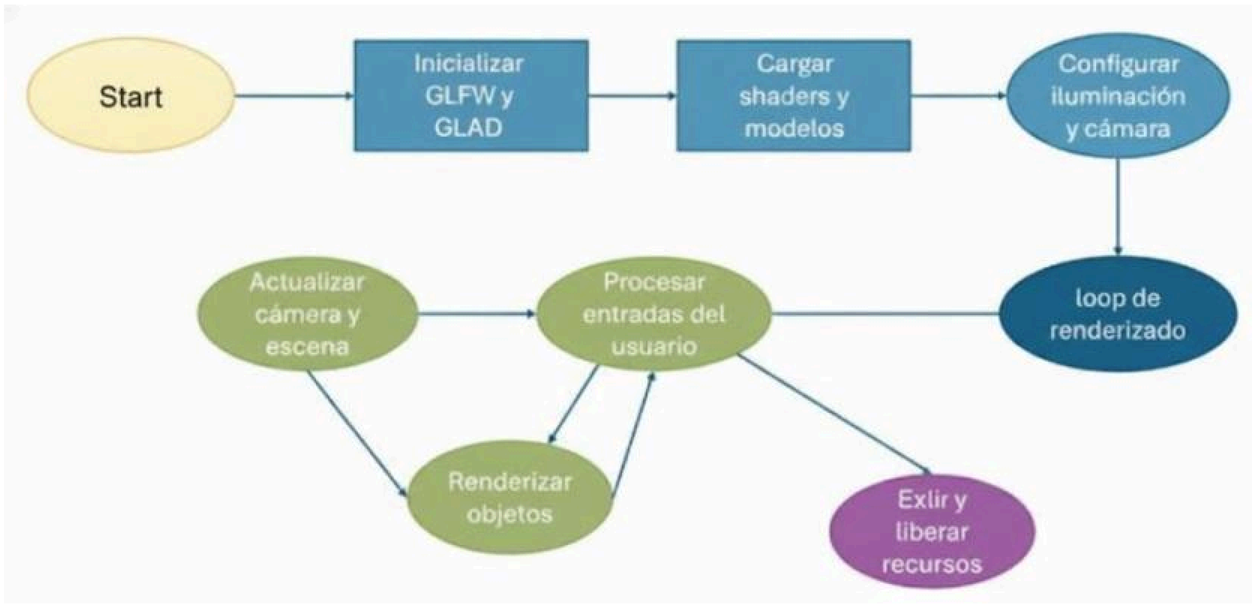
For it to work, images must be in an accessible path. Once loaded, the texture is activated in a texture unit. The *fragment shader* then uses a sampler variable to read the color of that drive, completing the rendering of visual detail.

Technical Manual

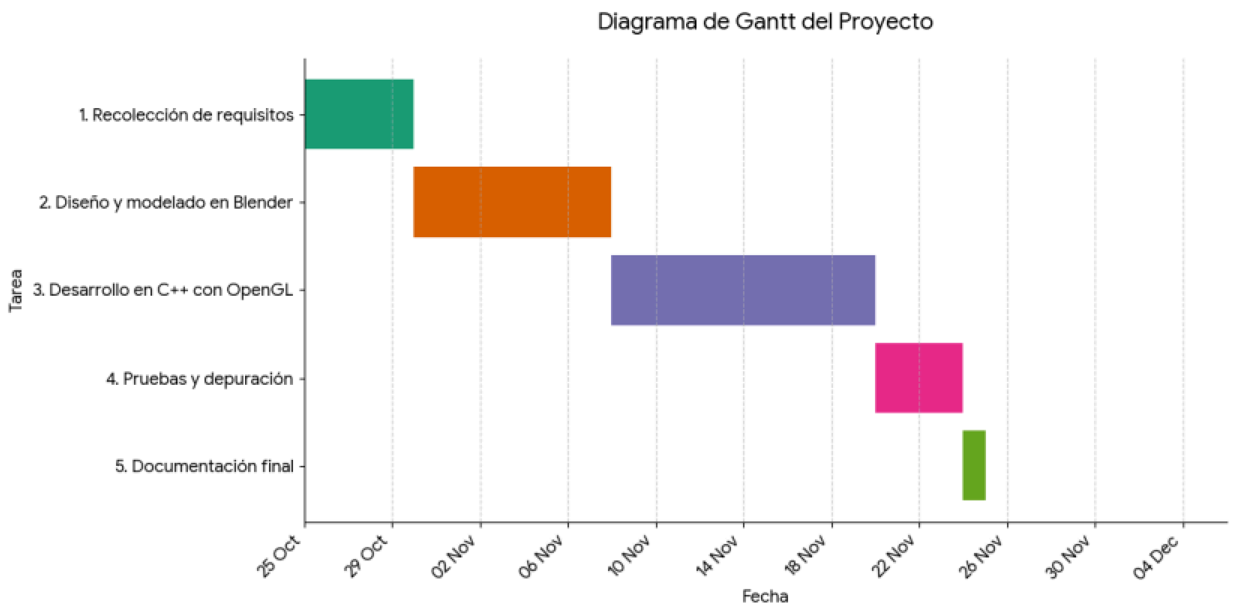
Project Goals

The student will need to apply and demonstrate the knowledge gained during the course by creating a 3D recreation in OpenGL.

Software Flow Diagram



Gantt Diagram



Project Scope

Interactivity and camera

- Implementation of a free first-person camera that allows the user to walk the scene using a keyboard and mouse.
- Dynamic motion control (W, A, S, D, or arrows) and rotation by mouse scroll.

Scene 3D

- Detailed recreation of an architectural facade and an interior composed of two rooms (dining room, room).
- Integration of 10 3D models exported from Blender in .fbx format.

Lighting

- Advanced lighting system based on the Phong model.
- Configuration of colors, intensities and attenuations in each type of light.

Materials and texturing

- Use of materials with diffuse and specular textures for each model, applied with UV mapping from Blender.

Real-Time Rendering

- Efficient rendering of the scene with depth control (Z-buffer).
- VAO/VBO configuration for meshes.
- Use of structures and functions organized into classes: Model, Mesh, Camera, Shader.

Limitations

1. Formatting and Texture Support

- The FBX format does not allow you to embed (*embed*) bitmap textures directly. This forces texture files (.png, .jpg) to be kept in a valid path (often the same folder or relative path) so that they can be successfully located and loaded by the target application (OpenGL/C++). If the path fails, the model appears without textures.
- Specular textures (map_Ks) are not automatically exported from Blender if they are not explicitly configured on the material node, complicating reflectivity simulation if not done manually.

2. Rendering limited to Basic Phong

- The lighting system is based on the Phong lighting model, which limits advanced visual effects such as real-time reflections, ambient occlusion or more realistic PBR materials.
- No techniques were used such as shadow mapping, screen-space reflections or normal mapping, which restricts visual fidelity.

3. Hardware-dependent performance

- By failing to implement optimization techniques such as frustum culling, LOD (Level of Detail), or instantiated batches, performance can be impacted on equipment with integrated GPU or low graphical power.

4. Basic Interaction

- Navigation is limited to free movement by keyboard and mouse. No collisions, advanced interaction zones, or event detection were implemented on objects.
- There is no integrated menu or graphical interface that facilitates the user experience.

5. Visual accuracy subject to export

- The scales, rotations, and orientations of models exported from Blender may not exactly match the original scene if the transformations are not correctly applied (Ctrl + A ↣ Apply All Transforms).

6. Absence of atmospheric effects

- Post-processing effects such as fog, blur, bloom, and dynamic ambient tone were not included, reducing the sense of realism.

Applied Software Methodology

We used the cascading model, which follows a linear sequence of stages. This model was suitable for the development of the 3D graphic environment, allowing a structured planning of the work. The stages applied were:

1. **Requirements Collection**
   Essentials of the scenario were defined, including the theme, number of minimum objects required, type of interaction, and lighting systems.
2. **Environment Design and Object Modeling**
   Space layout 3D was planned and objects were modeled in Blender, considering proportions, textures, materials, and virtual space organization.
3. **Developing in C++ with OpenGL**
   The source code was implemented using C++ and the OpenGL library. The .fbx models exported from Blender were integrated, transformations, textures, lighting and interactive cameras were applied.
4. **Visual Tests and Engagement**
   Constant tests were performed to verify the correct rendering of models, the response to keyboard and mouse events, and the behavior of dynamic lights and animated objects.
5. **Documentation**
   A user and technical manual was produced, recording both the design, structure of the code, and overall operation of the environment, as well as the necessary instructions for navigation and interaction.

Code Documentation

1. Relevant Global Variables

Variable	Type	Description
window	GLFWwindow*	Pointer to the GLFW window, the main rendering context.
camera	Camera	An instance of the Camera class for the view.
sceneManager	std::unique_ptr<SceneManager>	Central manager of the scene, handles the object hierarchy and rendering.

2. Initialization Features (Start)

initializeWindow() bool

- Description: Configures and initializes the GLFW and GLAD window, the OpenGL *loader*. Sets the graphical context and input *callbacks*.
- Parameters: None.
- Return: true if initialization was successful, false otherwise.

void loadModels(...)

- Description: It is responsible for loading the essential 3D models for the scene. In the current version of the code, it only loads the floor and main house.
- Parameters:
    - loadingScreen: Reference to the upload screen to show progress.
    - house: Pointer to Model* where the house model will be loaded ("monster_house/models/MonsterHouseFinal.fbx").
    - floor: Pointer to Model* where the lunar floor model will be loaded ("monster_house/models/piso.fbx").

- Specific Action (Home and Floor): Uses the FBX file path to load the model data into the Model structures.

void defineMaterials(...)

- Description: Defines the properties of the material (environment, diffuse, specular, transparency) for the main objects in the scene.
- Parameters (Relevant):
    - floorMaterial: Reference to Material to configure floor properties.
    - houseMaterial: Reference to Material to configure the properties of the main house.
    - (Other Materials): Set to Material() (empty/neutral) for deleted objects.
- Specific Configuration:
    - Floor: It is configured with a high environmental contribution and diffuse gray/blue, and a specular low to simulate the opaque rocky texture of the lunar surface.
    - Home: It is configured with more balanced properties, typical of a grey or metallic structure, with a moderate specular.

void setupLighting(...)

- Description: Sets up and adds the lights to the scene through the LightManager in SceneManager.
- Parameters (Relevant):
    - sunLightIdx: Reference to size_t to store the Sun's global light index.
    - interiorCasaLightIdx: Reference size_t to store the index of the local light inside the house.
- Action: Creates Light instances, sets their Position, Color, and Power, and adds them to the Light Manager.

void createSceneObjects(...)

- Description: Instantiate the renderable objects (RenderableObject) in the scene from the uploaded models and shaders, assign them their material, position, rotation, and scale, and add them to the SceneManager.
- Parameters (Relevant):
    - floor: Pointer to the floor Model model.
    - house: Pointer to the Model model of the house.
    - mLightsShader: Pointer to the main Shader for objects with multiple lights.
    - floorMaterial: Reference to the floor Material.

    - houseMaterial: Reference to the Material of the house.
    - interiorCasaLightIdx: Index of indoor light so that the house is affected by it.
- Specific Instantiation:
    - Floor:
        - Creates an RenderableObject with pisoLunar and mLightsShader.
        - Position: (0.0f, 0.0f, 0.0f)
        - Rotation: (-90.0f, -90.0f, 0.0f) (Tuning to make the model flat).
        - Scale: (5.0f)
        - Assigns floorMaterial.
    - House (MonsterHouse):
        - Creates an RenderableObject with house and mLightsShader.
        - Position: (0.0f, 16.5f, 0.0f)
        - Rotation: (-90.0f, 0.0f, 0.0f) (Base Orientation).
        - Scale: (1.0f)
        - Assigns houseMaterial.
        - Adds interiorCasaLightIdx to your affected lights.

3. Parent Loop Function (`Update`)

update() bool

- Description: The main function of the *render loop*. Calculates delta time, processes input, clears the screen *buffer*, updates camera positions (including vertical jump scrolling), and calls the SceneManager update and rendering methods.
- Parameters: None.
- Return: true to continue the loop.
- Specific Action (Cameras):
    - Calculates the vertical offset by physical (jumpDisplacement).
    - Updates the position of the active camera.
    - Calls sceneManager->update(deltaTime) to update the logic of all objects, including rotations or orbits.
    - Call sceneManager->render() to draw the scene, rendering the sky (Cubemap), the floor, and the house.

4. Project Classes (Overridden)

RenderableObject

- Usage (Floor and Home): This is the base class for static objects in the scene. The instances of the floor and house are of this type.
- Key Features:
    - Stores a pointer to a model (Model*) and a shader (Shader*).

  - Defines the transformation matrix (model matrix) by using position, rotation, and scale glm::vec3.
  - Stores the material properties (Material).
  - render() method: Draw the model using its shader and sending the light and material properties.
  - addAffectedLight() method: Specify which local lights affect the object (used for the interior light of the house).

Model

- Usage (Floor and Home): Contains the data structure for a static 3D model (meshes, vertices, textures) uploaded from files such as FBX.
- Key Features: Abstracts model load, typically using libraries such as Assimp, and stores the meshes (Mesh) that make up the object.

Project Cost and Sale Price Analysis

Resource/Activity	Quantity	Unit Cost (MXN)	Total Cost (MXN)	Comments
Development hours (scheduling OpenGL)	60 h	$100/h	$6,000	Coding logic, animations, shaders, rendering, etc.
Modeling hours 3D (Blender)	60 h	$100/h	$6,000	Modeling and texturing objects in Blender
Documentation (manuals, report)	10 h	$80/h	$800	Technical and user writing
Software used (free licenses)	-	$0	$0	Open source software (Blender, GLEW, GLFW, Assimp) was used

Hardware (Personal PC)	1 month of use	$500 Amort.	$500	Own computer use for rendering and programming
Electricity (estimated)	1 month	$150	$150	Approximate energy cost for work sessions
Pre-training (courses or classes)	1 Course	$1,000	$1,000	University Course in Graphic Computing
Estimated Total Costs			$14,450 MXN	

Estimate Sales Price

In order to determine the selling price, it is considered:

- **Utility margin of 50%, because it is a personalized technical development.**
- **Indirect costs, such as possible maintenance or basic support.**
- **Value added by interactivity, educational use or attractive visual presentation.**

Calculation:

Selling Price = Total Cost x 1.5 $21,675 = $14,450 x 1.5

CONCLUSION

The development of this graphic computing project in C++ and OpenGL not only served as an academic assignment, but as a test of fire to consolidate and apply the totality of the knowledge acquired during the semester. The main goal of building a functional virtual environment was fully achieved by successfully integrating *shaders*, advanced lighting (Phong Model), 3D resource management (FBX models), and camera.

From my perspective, the project proved to be considerably more challenging than anticipated. The complexity resided not only in the code, but in the interdependence of the systems: from the initial configuration of GLFW and GLAD to the correct interpretation of the data of the transformation models and matrices. It was an intense exercise in debugging and acquiring new skills, especially in the correct implementation of custom shaders. Despite the obstacles, we achieved a very solid functional reach in rendering and user interaction.

While the project met the basic requirements, it is crucial to identify areas where limitations were presented or where implementation could be more robust, which is an invaluable lesson for my future developments:

1. Integrating Assets (FBX Textures): The biggest initial frustration came from handling textures. The limitation of the FBX format of not embedding textures, forcing us to handle external and relative file paths, caused delays and loading failures. This taught us the critical importance of standardizing export pipelines from Blender.
2. Specular Lighting Management: We rely heavily on the fixed values of Material.specular in C++ to simulate brightness. This was, in part, a consequence of specular maps not being properly exported from the FBX model, resulting in a less accurate and more uniform reflectivity simulation than would have been achieved with proper texture mapping.

Ultimately, the project was a success in synthesizing knowledge, but it made it clear to us that the robustness of the assets pipeline and the full utilization of all shader capabilities are the next critical steps in my computer graphics learning.