NATIONAL AUTONOMOUS UNIVERSITY OF MEXICO

School of Engineering

Final Project

Subject:

Computer Graphics and Human-Computer Interaction (1590)

Group: 05

Semester: 2025-2

Student:

Quintero Montero Francisco Joshua

Professor:

ENG. CARLOS ALDAIR ROMAN BALBUENA

Due date: May 20, 2025

Table of contents

User Manual

Objectives

General Objective

Provide a clear and accessible guide so that users can download, install, launch, and navigate intuitively within the virtual environment developed in OpenGL, enabling an immersive experience without technical complications.

Specific Objectives

- Explain step-by-step the installation and execution process of the project to facilitate its setup.
- Describe the basic controls for movement and interaction within the virtual environment.
- Inform about the lighting control functionalities so the user can customize the visual experience within the simulation.
- Promote a pleasant and smooth user experience through a clear and user-friendly presentation of the instructions.

Minimum System Requirements

To ensure the proper functioning of the project and a smooth experience while navigating the 3D environment, it is recommended that the system meets at least the following minimum requirements:

Hardware

- Processor (CPU): Intel Core i5 quad-core or equivalent AMD
- RAM: 8 GB
- Graphics Card (GPU): NVIDIA GeForce GTX 960 or higher with support for OpenGL 4.0+
- Storage: 1 GB of free disk space (for executable, models, textures, and resources)
- Display: Minimum resolution of 1280x720

Software

- Operating System: Windows 10 or higher (64-bit)
- Required Libraries (included in the installation):
  - OpenGL 4.0 or higher (supported by most modern GPUs)
  - GLFW (library for window management and input)
  - GLEW (OpenGL extensions)
  - SOIL2 (texture loading)
  - OpenAL (audio)

Additional Requirements

- Execution permissions: The program must run with sufficient permissions to open popup windows and access external files (models and textures).
- Support for running .exe files in Release mode.
- The folder structure and attached libraries must be maintained alongside the executable to ensure proper loading of resources and dependencies.

¡Welcome!

This 3D simulation project created with OpenGL will immerse you in a virtual experience based on the house from the *Regular Show* television show, where you will have the freedom to explore both the exterior and interior, including a detailed recreation of *Mordecai* and *Rigby's* kitchen and bedroom.

Steps to Start the Experience

1. Download the **Release.zip** folder from our GitHub repository, located in the main branch, at the following link: https://github.com/joshuaqm/319098147_proyectoFinal_gpo05
2. Unzip and locate the executable file named **ProyectoFinal.exe**.
3. Double-click it.

4. Wait a few seconds while the 3D environment loads. Once launched, a window will open with the main scene.

Initial Scenario

When you start the simulation, you will appear in front of the house, observing its facade. The setting includes elements such as:

- Grass
- Sky
- Trees and bushes
- A decorative lighthouse

These elements have been designed to create a visually pleasing environment consistent with the project's theme.

Movement Controls

To move around the virtual environment, use the following keys:

Key	**Action**
W	Go forward
S	Go backwards
A	Move to the left
D	Move to the right
Space	To rise (to move upwards)

Shift	Descend (move down)
Ctrl + (W/A/S/D)	Increase movement speed
Mouse	Control the camera orientation to look around the scene

General Lighting Controls

You can change the lighting conditions of the scene using the following keys:

Key	**Action**
N	Toggle between day and night lighting
L	Turn the scene's point lights on or off
F	Turn on or off the spotlight (flashlight)

NOTE: The first point light is located outside the house, in the lighthouse. The second is inside, in the kitchen area, and the third is in the lamp above the table inside the room. Make sure to approach these objects and activate the lights to observe the effect they have on the other elements.

Interaction Inside The House

When entering the house and exploring its interior, there is a recreation of the kitchen. Inside it, you can manually control the position of a light source to see how it affects the lighting on the different objects.

Key	Action
U	Move the light forward
J	Move the light backwards
H	Move the light to the left
K	Move the light to the right
Flecha ↑ (arriba)	Raise the light
Flecha ↓ (abajo)	Lower the light

Sound Interaction Controls

Up all the stairs in the house, there's a room where you'll find a radio. Inside, you can control the audio playback produced by the object.

Key	Action
M	Toggle audio playback or pause in the scene

Technical Manual

Objectives

General Objective

Apply and demonstrate the knowledge acquired in the Computer Graphics course through the development of a three-dimensional (3D) recreation using the OpenGL library and modeling software.

Specific Objectives

- Model a facade and a space (real or fictional) that includes at least five objects in each room, with a high degree of visual fidelity compared to reference images.
- Implement coherent ambiance that complements the virtual environment, respecting aesthetic and technical criteria.
- Document the development process through a technical manual that includes diagrams, project cost analysis, and source code documentation.
- Provide a user manual that clearly explains the interaction within the generated three-dimensional environment.
- Use modeling tools such as Maya Autodesk 2025 to generate 3D models, which will later be integrated into the OpenGL environment, ensuring an efficient transition between design software and graphical programming environment.

Project Scope

The main objective of this project is to develop an interactive three-dimensional virtual tour of an architectural environment, including a facade and two interior spaces, using the OpenGL graphics library version 3.

The specific scopes of the project include:

- Apply textures, lighting, and ambiance to enhance visual realism.
- Implement four animations.

- Develop a basic interface for user interaction within the virtual environment, according to the user manual.
- Thoroughly document the project, including diagrams, development methodology, cost analysis, and explanation of the implemented code.
- Deliver the project in a public GitHub repository, maintaining the change history and without compressed files.
- Properly export and integrate the 3D models created in Maya Autodesk 2025 into the OpenGL environment, ensuring format compatibility, geometry optimization, and preservation of textures and materials within the technical limitations established by the project.

Limitations

There are various technical and academic restrictions that defined its scope. These limitations are detailed below:

- It is not allowed to recreate spaces with legal restrictions, such as UNAM facilities, government buildings, or prohibited places.
- The conversion and use of 3D models are limited by OpenGL size and performance restrictions, limiting geometric complexity and maximum weight (100 MB per model).
- The available hardware capabilities impose restrictions that may affect smoothness and optimal performance during development and execution of the project.
- The project is individual; although the first part was worked on as a team (facade and one room), the final delivery must include elements recreated and furnished individually.

Project Cost Analysis

Below is the detailed analysis and breakdown of costs incurred during the development and deployment of the final product, considering both human and technical resources employed. This analysis includes time investment, software licensing, hardware acquisition, and other supplies necessary for the correct execution and delivery of the project, providing a solid basis for estimating the total cost and suggested sale price.

Development Costs

Concept	**Description**	**Time quantity**	**Unit Cost (MXN)**	**Total Cost (MXN)**
Human capital costs				
Development	Work on modeling, programming, and learning	60 hours	$120	$7,200
Software costs				
Autodesk Maya License	Professional monthly license	1 month	$5,000	$5,000
ChatGPT Plus Subscription	Texture generation and code adaptation	1 month	$400	$400
Hardware costs				
Desktop computer for development	Equipment capable of running Maya and OpenGL (processor, RAM, etc.)	1 unit	$18,000 (estimated)	$18,000
Other costs				
Other resources	Paid textures, models, and additional tools	-	-	$3,000 (estimated)
			Subtotal	$33,600

Estimated Sale Price

A sale price of $45,000 MXN is proposed, considering:

- Direct costs of development, licenses, and hardware.
- Added value due to quality, animations, and interactive functionality.
- Time invested and required technical specialization.

- Margin to cover risks, maintenance, and future improvements.

The proposed sale price of $45,000 MXN is based on factors beyond direct development and material costs. First, it includes the recovery of costs invested in software licenses, hardware, and resources necessary for production, totaling a subtotal of $33,600 MXN.

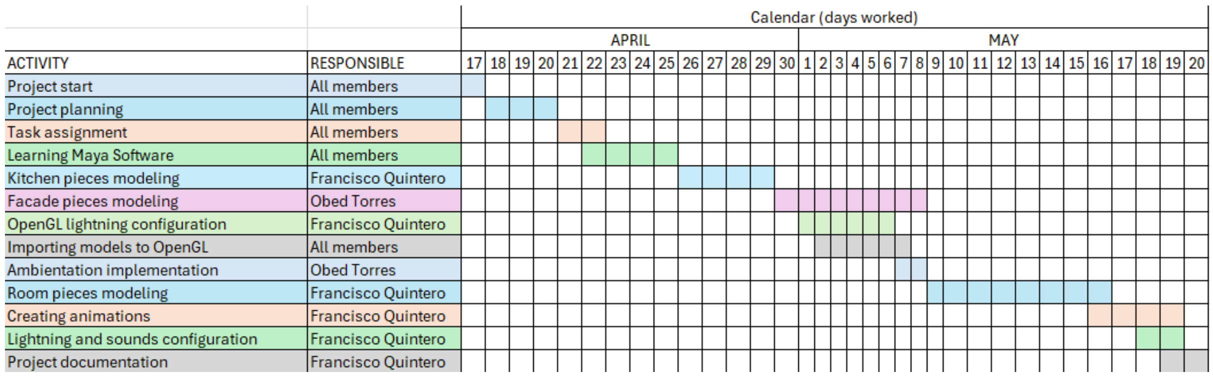
Additionally, this price considers the added value of the project, reflected in the visual quality achieved, programmed animations, and interactive functionality implemented to offer an immersive and professional experience to the end user.

The time invested, estimated at 60 hours, represents a significant investment in learning and development, covering both the acquisition of technical knowledge in Autodesk Maya and OpenGL and its practical application in the project.

Finally, the price includes an adequate margin to cover inherent risks of software development, possible maintenance costs and future improvements, as well as to compensate the specialization and dedication of the developer, ensuring the economic and professional viability of the project.

Gantt Chart

The following Gantt chart shows the workflow carried out over time to achieve the project objectives, including descriptions of the activities performed.



		Calendar (days worked)																																	
		APRIL														MAY																			
ACTIVITY	RESPONSIBLE	17	18	19	20	21	22	23	24	25	26	27	28	29	30	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Project start	All members	■																																	
Project planning	All members		■	■	■																														
Task assignment	All members					■	■																												
Learning Maya Software	All members						■	■	■	■																									
Kitchen pieces modeling	Francisco Quintero										■	■	■	■																					
Facade pieces modeling	Obed Torres														■	■	■	■	■	■	■	■	■												
OpenGL lightning configuration	Francisco Quintero															■	■	■	■	■	■														
Importing models to OpenGL	All members																■	■	■	■	■	■													
Ambientation implementation	Obed Torres																					■	■												
Room pieces modeling	Francisco Quintero																							■	■	■	■	■	■	■	■				
Creating animations	Francisco Quintero																														■	■	■	■	
Lightning and sounds configuration	Francisco Quintero																																■	■	
Project documentation	Francisco Quintero																																	■	■

Figure 0. Project Gantt Chart.

Flowchart

The overall software flow is organized into the following steps:

1. **Application Start**
    - Execution of the ProyectoFinal.exe file (Release mode) or from the development environment (Debug).
    - Initialization of GLFW and creation of OpenGL window.
    - Configuration of GLEW for access to modern OpenGL functions.
    - Retrieval of framebuffer size.
2. **Resource Loading and Configuration**
    - Loading libraries and header files for OpenGL, GLFW, GLEW, SOIL2, stb_image, glm, and OpenAL for audio.
    - Definition and loading of shaders (lightingShader, lampShader).
    - Loading 3D models (.obj, .fbx), grouped by environments (exterior, kitchen, room, etc.).
    - Configuration of VBO and VAO for basic geometry (e.g., cubes for lights).
    - Initialization and configuration of audio with OpenAL, loading WAV file, and 3D source setup.
3. **Graphics Environment Initialization and Global Variables**
    - Viewport configuration with current dimensions.
    - Definition of global variables: camera, light positions, animation states, interaction flags (flashlight, lights, day/night).
    - Assignment of callbacks:
        - Keyboard (KeyCallback)
        - Mouse (MouseCallback)
        - Movement function (DoMovement)
    - Mouse input configuration for capture and relative movement.
4. **Main Rendering Loop**
    - DeltaTime calculation for smooth animations and movements.
    - Animation updates:
        - Circular drift animation of the cart.
        - Day/night transition with smooth interpolation.
        - Jump animation of the character Rigby on the trampoline.

    - Tree movement simulated by wind effect.
    - Falling leaves from the trees.
  - Audio updates:
    - Update listener position and orientation (camera).
    - Audio playback control.
  - Input event processing (keyboard and mouse).
  - Application of movements based on user input.
  - Clearing color and depth buffers.
  - Setting and sending uniform variables to shaders for lighting (directional day/night light, point lights, flashlight spotlight).
  - Rendering 3D models with transformations (translation, rotation, scale).
  - Rendering point lights as visual objects (cubes).
  - Buffer swapping to display rendered frame.

5. **User Interaction**
   - Free 3D navigation with camera (WASD, space, shift).
   - Real-time control of:
     - Main point light position (keys H, K, U, J, Up, Down).
     - Global lighting state (key L to toggle lights on/off).
     - Day/night change (key N with smooth transition).
     - Flashlight activation/deactivation (key 1).
     - Audio playback control (key M).
   - Camera movement influences spotlight direction and position (flashlight).
   - Active and synchronized real-time animations (cart, jump, wind).
6. **Application Closure**
   - Closing the window terminates the rendering loop.
   - Cleanup and release of graphic resources (buffers, shaders, models).
   - Stopping and releasing audio resources (source, buffer, OpenAL context).
   - GLFW termination.

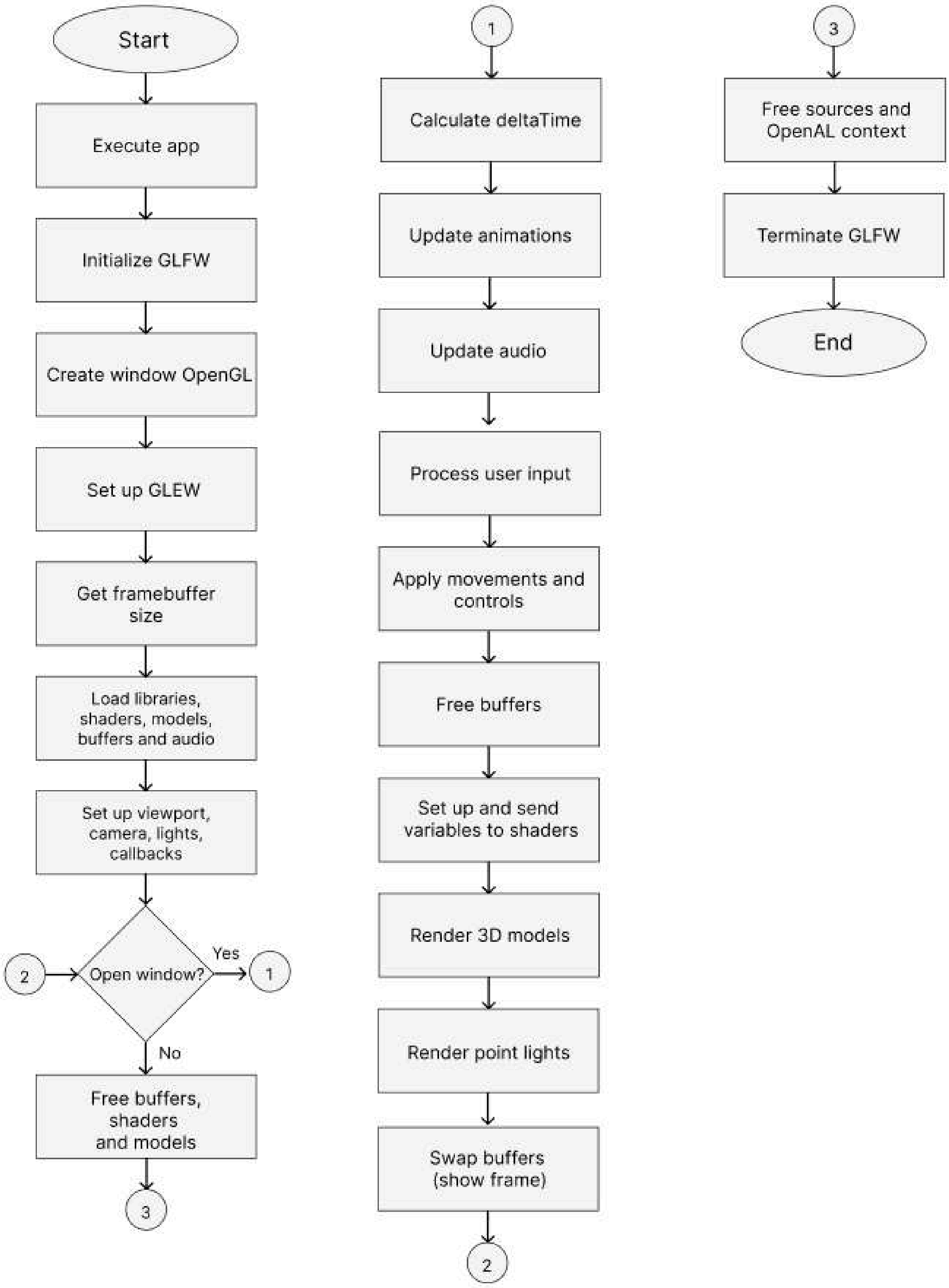

Start
Execute app
Initialize GLFW
Create window OpenGL
Set up GLEW
Get framebuffer size
Load libraries, shaders, models, buffers and audio
Set up viewport, camera, lights, callbacks
2
Open window?
Yes
1
No
Free buffers, shaders and models
3
1
Calculate deltaTime
Update animations
Update audio
Process user input
Apply movements and controls
Free buffers
Set up and send variables to shaders
Render 3D models
Render point lights
Swap buffers (show frame)
2
3
Free sources and OpenAL context
Terminate GLFW
End

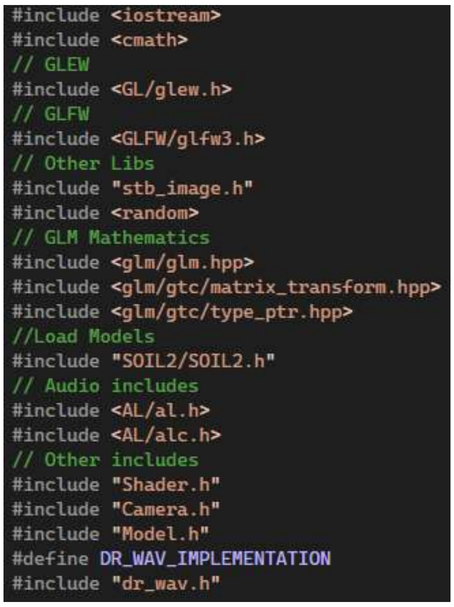
Figure 1. Program flowchart

Code Documentation

The code includes the implementation of 3D models of a house and other objects to set the exterior environment, as well as interaction with the scene through the camera and light sources. Below is a description of each code section:

1. Library Inclusion

- **GLEW (OpenGL Extension Wrangler):** Loads OpenGL extensions to access modern rendering functions.
- **GLFW:** Manages window creation, keyboard/mouse input, and system events.
- **STB Image:** Allows loading textures from image files (such as PNG or JPG).
- **GLM (OpenGL Mathematics):** Provides structures for mathematical operations (vectors, matrices) essential for 3D graphics.
- **SOIL2:** Facilitates texture loading for 3D models.
- **dr_wav:** Minimalist library for decoding WAV audio files.
- **OpenAL:** Audio API for 3D playback.
- **Random (C++ Standard Library):** Generates random numbers for effects like falling leaves.
- **Other modeling libraries (Shader, Camera, Model):** Allow loading and rendering 3D models, applying shaders, and manipulating the camera.

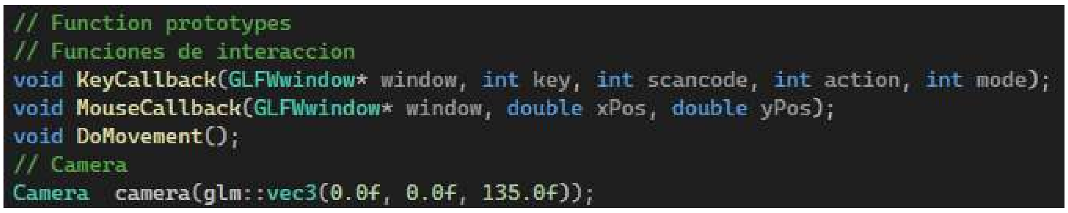


```
#include <iostream>
#include <cmath>
// GLEW
#include <GL/glew.h>
// GLFW
#include <GLFW/glfw3.h>
// Other Libs
#include "stb_image.h"
#include <random>
// GLM Mathematics
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
//Load Models
#include "SOIL2/SOIL2.h"
// Audio includes
#include <AL/al.h>
#include <AL/alc.h>
// Other includes
#include "Shader.h"
#include "Camera.h"
#include "Model.h"
#define DR_WAV_IMPLEMENTATION
#include "dr_wav.h"
```

Figure 2. Inclusion of libraries and header files.

2. Camera Configuration

The camera is defined using a Camera class, which allows movement of the 3D view within the scene. The code uses functions such as ProcessKeyboard and ProcessMouseMovement to move and rotate the camera based on keyboard and mouse input.



```
// Function prototypes
// Funciones de interaccion
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode);
void MouseCallback(GLFWwindow* window, double xPos, double yPos);
void DoMovement();
// Camera
Camera  camera(glm::vec3(0.0f, 0.0f, 135.0f));
```

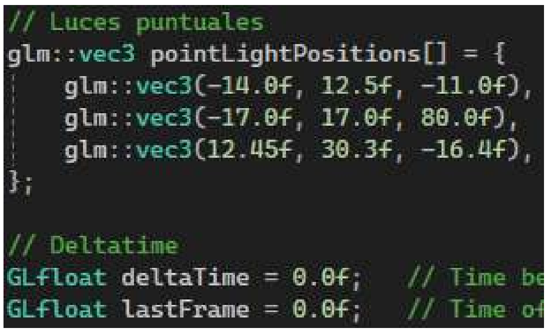
Figure 3. Movement functions and synthetic camera declaration.

3. Global Variables

The program uses global variables to control animations and dynamic effects. Some of the most relevant include:

- **Lighting:**
  - pointLightPositions: Stores the positions of the point lights in the scene.
  - deltaTime and lastFrame: Control the time between frames, allowing smooth and consistent camera movement.
- **Animations:**
  - driftAngle and rotationSpeed: Control the circular movement of the golf cart.
  - jumpProgress and squashFactor: Manage the jump and deformation of the character (Rigby).
  - hojasYOffset: Vertical offset to simulate falling leaves from the trees.
- **Transitions:**
  - timeOfDay and transitionActive: Interpolate between day and night states.
  - isNight: Flag to toggle night mode.
- **Light Effects:**
  - flashlightOn: Enables/disables the flashlight (spotlight).
  - lightsOff: Turns point lights on/off in the scene.
- **Audio:**
  - audioPlaying: State of ambient sound playback.
  - audioSourcePos: Fixed position of the audio source in the 3D world.

These variables are modified in real time through user interactions (keys N, L, F, M) or during main loop updates (deltaTime).

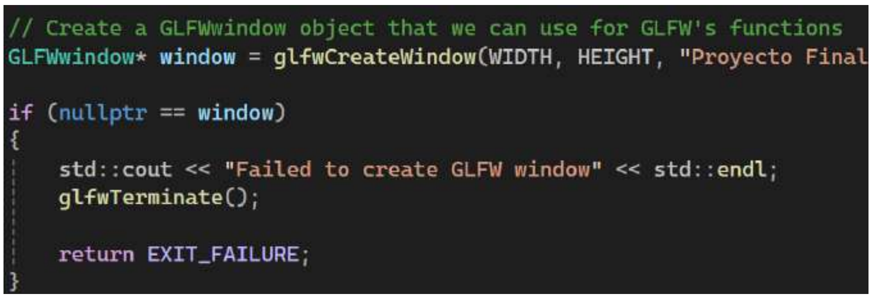


```
// Luces puntuales
glm::vec3 pointLightPositions[] = {
    glm::vec3(-14.0f, 12.5f, -11.0f),
    glm::vec3(-17.0f, 17.0f, 80.0f),
    glm::vec3(12.45f, 30.3f, -16.4f),
};

// Deltatime
GLfloat deltaTime = 0.0f;   // Time be
GLfloat lastFrame = 0.0f;   // Time of
```

Figure 4. Point lights declaration.

4. GLFW and GLEW Initialization

- GLFW is initialized and a window is created to render the 3D scene. If the window is not created properly, the program terminates.
- GLEW is initialized to ensure OpenGL functions are available.



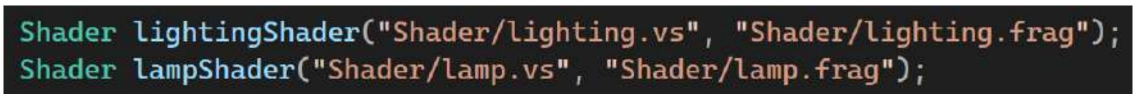
```
// Create a GLFWwindow object that we can use for GLFW's functions
GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "Proyecto Final

if (nullptr == window)
{
    std::cout << "Failed to create GLFW window" << std::endl;
    glfwTerminate();

    return EXIT_FAILURE;
}
```

Figure 5. Window initialization.

5. Shaders

Two shaders are loaded:

- lightingShader: Used for main objects in the scene, such as the house, trees, and bushes.
- lampShader: Used to represent light sources (point lights) in the scene.

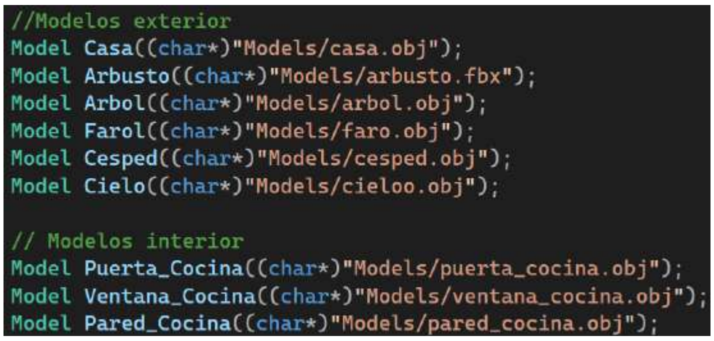


```
Shader lightingShader("Shader/lighting.vs", "Shader/lighting.frag");
Shader lampShader("Shader/lamp.vs", "Shader/lamp.frag");
```

Figure 6. Initialization of lighting shaders.

6. Model Loading

3D models of scene objects are loaded using the Model class. Models represent the house, trees, doors, windows, stove, bed, and other objects in the kitchen and house.

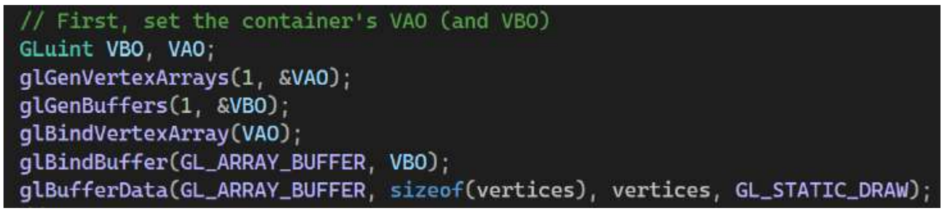


```
//Modelos exterior
Model Casa((char*)"Models/casa.obj");
Model Arbusto((char*)"Models/arbusto.fbx");
Model Arbol((char*)"Models/arbol.obj");
Model Farol((char*)"Models/faro.obj");
Model Cesped((char*)"Models/cesped.obj");
Model Cielo((char*)"Models/cieloo.obj");

// Modelos interior
Model Puerta_Cocina((char*)"Models/puerta_cocina.obj");
Model Ventana_Cocina((char*)"Models/ventana_cocina.obj");
Model Pared_Cocina((char*)"Models/pared_cocina.obj");
```

Figure 7. 3D model import.

7. VBO and VAO Definition

- VBO (Vertex Buffer Object): Stores vertices of the cube used as the base of the scene.
- VAO (Vertex Array Object): Organizes vertex attribute configurations, such as positions and normals.



```
// First, set the container's VAO (and VBO)
GLuint VBO, VAO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
glBindVertexArray(VAO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
```

Figure 8. Initialization of VBO and VAO.

8. Configuración de las Luces

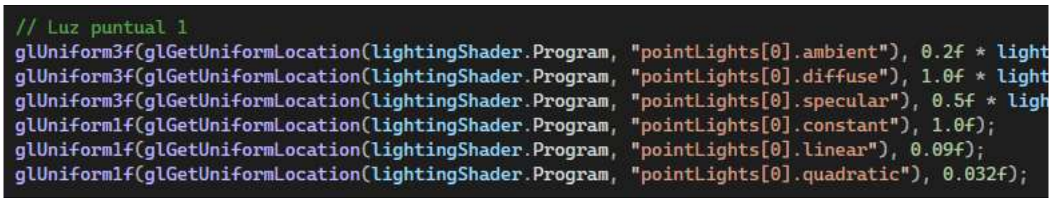
8. Light Configuration

The program includes three types of lights:

- **Directional light**: Simulates sun/moon, interpolates between day/night values (color, direction, intensity), controlled with key N.
- **Point lights**: Three fixed light sources in the scene, intensity controlled by key L (on/off), each with custom attenuation and color. The first point light is freely located in the kitchen, the second simulates light generated by the lighthouse, and the third simulates light from the lamp in the room.

- **Spotlight (flashlight)**: Follows the camera direction, adjustable light cone (inner/outer), toggled with key F.

All lights are calculated in the shader and respond dynamically to interactions.

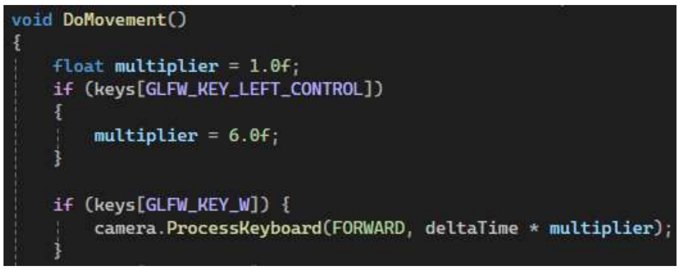


```
// Luz puntual 1
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"), 0.2f * light
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), 1.0f * light
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"), 0.5f * ligh
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].linear"), 0.09f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"), 0.032f);
```

Figure 9. Configuration of point light 1.

9. Camera Movement

Camera movement is controlled using keys (W, A, S, D, Space, Shift) and also adjusts when keys like Control are pressed. The mouse is used to rotate the camera and look around the scene.

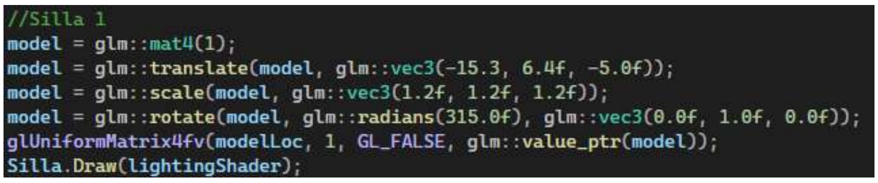


```
void DoMovement()
{
    float multiplier = 1.0f;
    if (keys[GLFW_KEY_LEFT_CONTROL])
    {
        multiplier = 6.0f;
    }

    if (keys[GLFW_KEY_W]) {
        camera.ProcessKeyboard(FORWARD, deltaTime * multiplier);
    }
```

Figure 10. Function controlling camera movement via keyboard input.

10. Drawing Models

The transformation matrix for each model (translation, rotation, scale) is configured before drawing on the screen. This includes both exterior models (house, bushes, trees) and interior objects (stove, table, chairs, bed, nightstand).

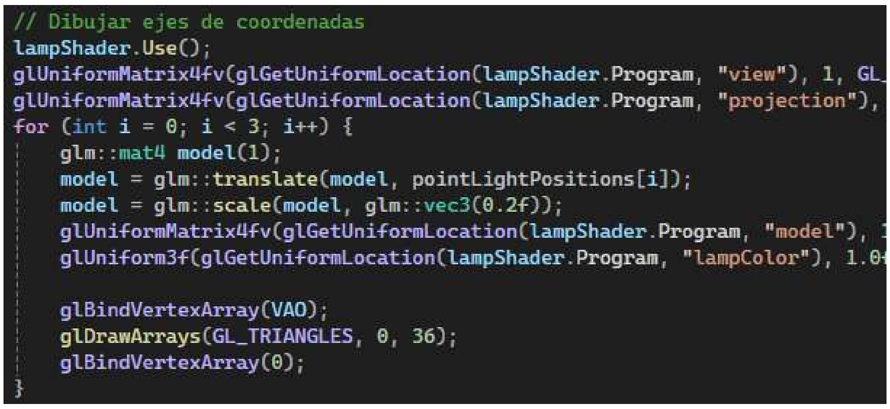


```
//Silla 1
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-15.3, 6.4f, -5.0f));
model = glm::scale(model, glm::vec3(1.2f, 1.2f, 1.2f));
model = glm::rotate(model, glm::radians(315.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Silla.Draw(lightingShader);
```

Figure 11. Model transformations and drawing.

11. Point Light Configuration

Point lights are drawn using a small cube to represent the light at defined positions. The light changes intensity and color depending on user input (key L).



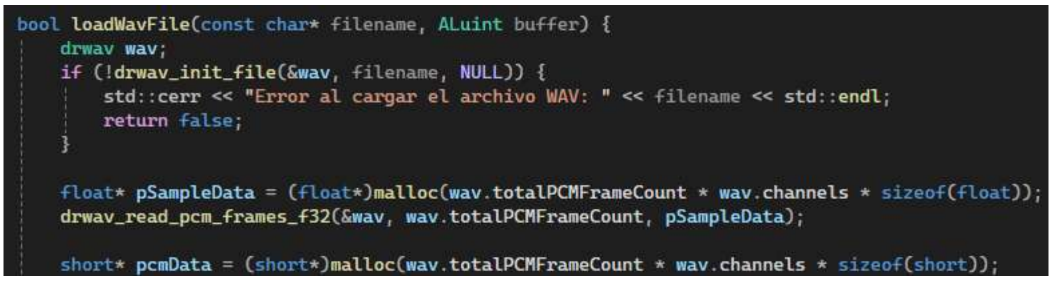
```
// Dibujar ejes de coordenadas
lampShader.Use();
glUniformMatrix4fv(glGetUniformLocation(lampShader.Program, "view"), 1, GL_
glUniformMatrix4fv(glGetUniformLocation(lampShader.Program, "projection"),
for (int i = 0; i < 3; i++) {
    glm::mat4 model(1);
    model = glm::translate(model, pointLightPositions[i]);
    model = glm::scale(model, glm::vec3(0.2f));
    glUniformMatrix4fv(glGetUniformLocation(lampShader.Program, "model"), 1
    glUniform3f(glGetUniformLocation(lampShader.Program, "lampColor"), 1.0f

    glBindVertexArray(VAO);
    glDrawArrays(GL_TRIANGLES, 0, 36);
    glBindVertexArray(0);
}
```

Figure 12. Drawing point lights in the scene.

12. Audio Functions

The program includes a 3D audio system using OpenAL to play positional sounds in the scene. Key functions are:

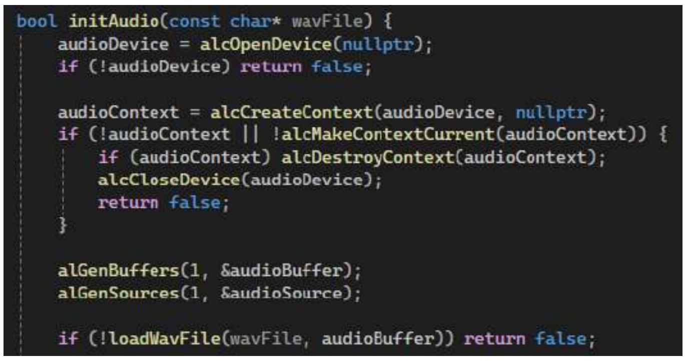
- **loadWavFile**: Loads a WAV audio file and prepares it for playback.
    - **Parameters**:
        - filename: Path to WAV file.
        - buffer: OpenAL buffer where audio will be stored.
    - **Return**: bool (success or failure).



```
bool loadWavFile(const char* filename, ALuint buffer) {
    drwav wav;
    if (!drwav_init_file(&wav, filename, NULL)) {
        std::cerr << "Error al cargar el archivo WAV: " << filename << std::endl;
        return false;
    }

    float* pSampleData = (float*)malloc(wav.totalPCMFrameCount * wav.channels * sizeof(float));
    drwav_read_pcm_frames_f32(&wav, wav.totalPCMFrameCount, pSampleData);

    short* pcmData = (short*)malloc(wav.totalPCMFrameCount * wav.channels * sizeof(short));
```

Figure 13. Fragment of the function that loads the audio file.

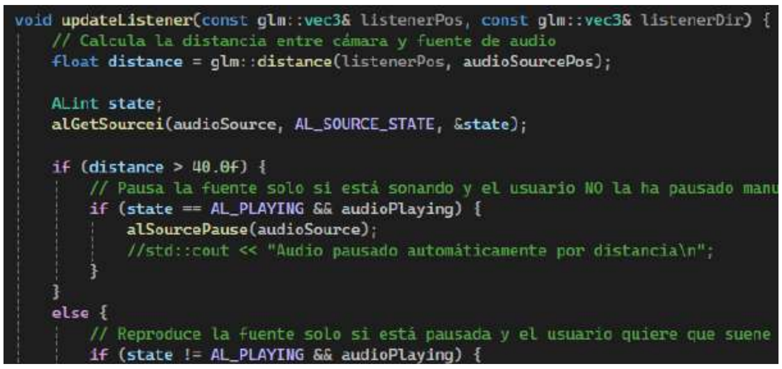
- **initAudio**: Initializes OpenAL device and context, loads the audio file, and sets up the sound source.
    - Parameters: wavFile (audio file path).
    - **Return**: bool (success or failure).



```cpp
bool initAudio(const char* wavFile) {
    audioDevice = alcOpenDevice(nullptr);
    if (!audioDevice) return false;

    audioContext = alcCreateContext(audioDevice, nullptr);
    if (!audioContext || !alcMakeContextCurrent(audioContext)) {
        if (audioContext) alcDestroyContext(audioContext);
        alcCloseDevice(audioDevice);
        return false;
    }

    alGenBuffers(1, &audioBuffer);
    alGenSources(1, &audioSource);

    if (!loadWavFile(wavFile, audioBuffer)) return false;
```

Figure 14. Function that initializes audio.

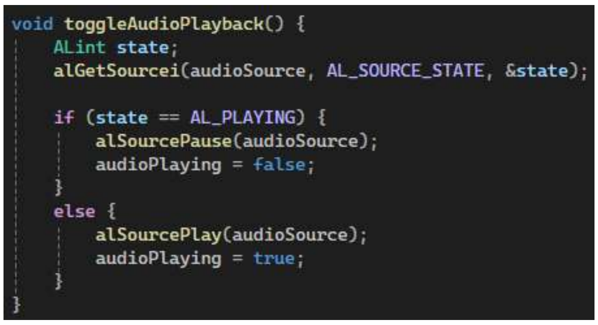
- **updateListener**: Updates listener position and orientation (camera) and controls playback based on distance to the audio source.
    - **Parameters**:
        - listenerPos: Camera position.
        - listenerDir: Direction the camera is facing.



```cpp
void updateListener(const glm::vec3& listenerPos, const glm::vec3& listenerDir) {
    // Calcula la distancia entre cámara y fuente de audio
    float distance = glm::distance(listenerPos, audioSourcePos);

    ALint state;
    alGetSourcei(audioSource, AL_SOURCE_STATE, &state);

    if (distance > 40.0f) {
        // Pausa la fuente solo si está sonando y el usuario NO la ha pausado manu
        if (state == AL_PLAYING && audioPlaying) {
            alSourcePause(audioSource);
            //std::cout << "Audio pausado automáticamente por distancia\n";
        }
    }
    else {
        // Reproduce la fuente solo si está pausada y el usuario quiere que suene
        if (state != AL_PLAYING && audioPlaying) {
```

Figure 15. Fragment of the function controlling attenuation and spatial sound.

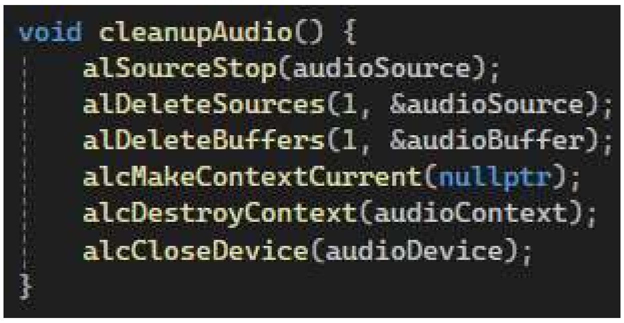
- **toggleAudioPlayback**: Toggles between playing and pausing audio. Activated with key M.



```cpp
void toggleAudioPlayback() {
    ALint state;
    alGetSourcei(audioSource, AL_SOURCE_STATE, &state);

    if (state == AL_PLAYING) {
        alSourcePause(audioSource);
        audioPlaying = false;
    }
    else {
        alSourcePlay(audioSource);
        audioPlaying = true;
    }
}
```

Figure 16. Function that toggles audio playback.

- **cleanupAudio**: Releases OpenAL resources when the program ends.



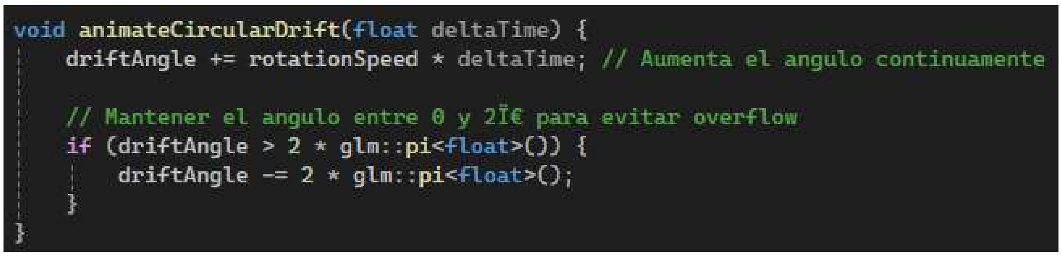
```
void cleanupAudio() {
    alSourceStop(audioSource);
    alDeleteSources(1, &audioSource);
    alDeleteBuffers(1, &audioBuffer);
    alcMakeContextCurrent(nullptr);
    alcDestroyContext(audioContext);
    alcCloseDevice(audioDevice);
}
```

Figure 17. Function that frees OpenAL resources.

13. Animation Functions

The program includes several dynamic animations created with functions that apply real-time transformations to models:

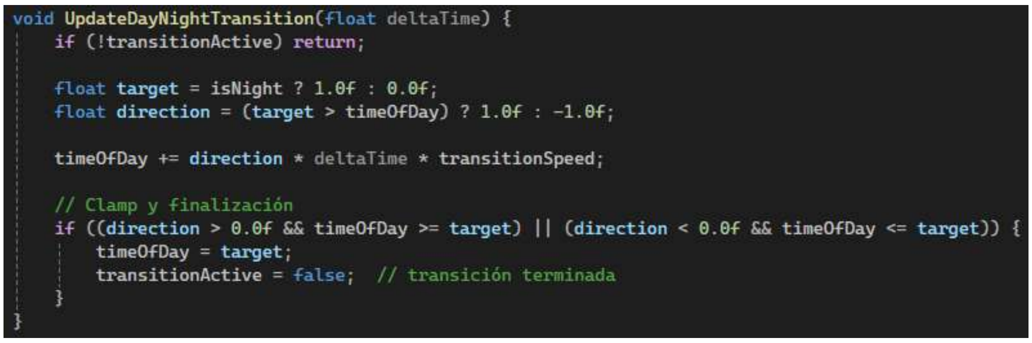
- **animateCircularDrift**: Animates a golf cart moving in a circular path around a pivot point.
    - **Mechanics**:
        - driftAngle: Angle increasing over time
        - circleRadius: Radius of circular path



```
void animateCircularDrift(float deltaTime) {
    driftAngle += rotationSpeed * deltaTime; // Aumenta el angulo continuamente

    // Mantener el angulo entre 0 y 2Ï€ para evitar overflow
    if (driftAngle > 2 * glm::pi<float>()) {
        driftAngle -= 2 * glm::pi<float>();
    }
}
```

Figure 18. Function creating golf cart drift animation.

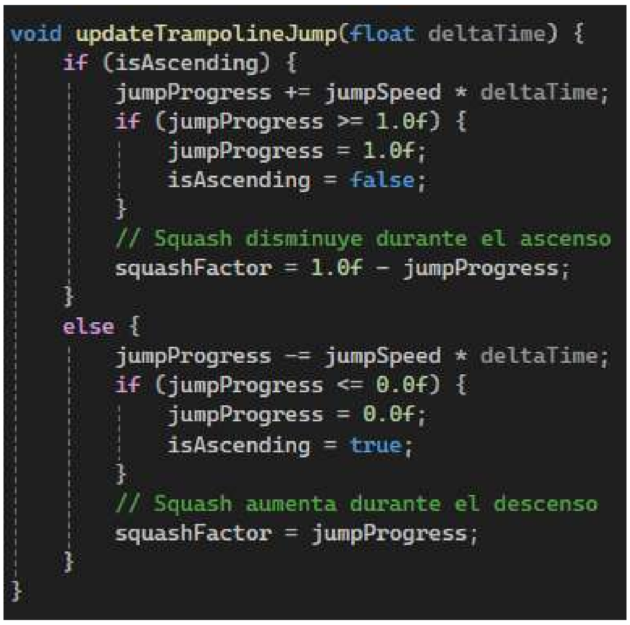
- **UpdateDayNightTransition**: Smoothly interpolates between day and night states.
    - **Key variables**:
        - timeOfDay: Value between 0 (day) and 1 (night).
        - transitionSpeed: Transition speed.



```
void UpdateDayNightTransition(float deltaTime) {
    if (!transitionActive) return;

    float target = isNight ? 1.0f : 0.0f;
    float direction = (target > timeOfDay) ? 1.0f : -1.0f;

    timeOfDay += direction * deltaTime * transitionSpeed;

    // Clamp y finalización
    if ((direction > 0.0f && timeOfDay >= target) || (direction < 0.0f && timeOfDay <= target)) {
        timeOfDay = target;
        transitionActive = false;  // transición terminada
    }
}
```

Figure 19. Function performing smooth day/night lighting transition.

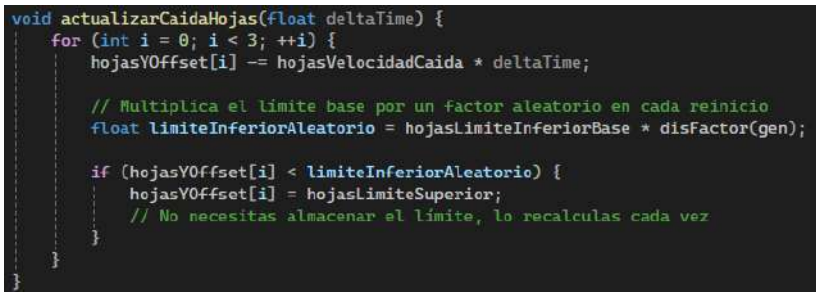
- **updateTrampolineJump**: Simulates a character's (Rigby) jump on a trampoline with "squash and stretch" effects.
    - **Mechanics**:
        - jumpProgress: Controls jump height.
        - squashFactor: Model compression during jump.



```
void updateTrampolineJump(float deltaTime) {
    if (isAscending) {
        jumpProgress += jumpSpeed * deltaTime;
        if (jumpProgress >= 1.0f) {
            jumpProgress = 1.0f;
            isAscending = false;
        }
        // Squash disminuye durante el ascenso
        squashFactor = 1.0f - jumpProgress;
    }
    else {
        jumpProgress -= jumpSpeed * deltaTime;
        if (jumpProgress <= 0.0f) {
            jumpProgress = 0.0f;
            isAscending = true;
        }
        // Squash aumenta durante el descenso
        squashFactor = jumpProgress;
    }
}
```

Figure 20. Function animating character jumps on the trampoline.

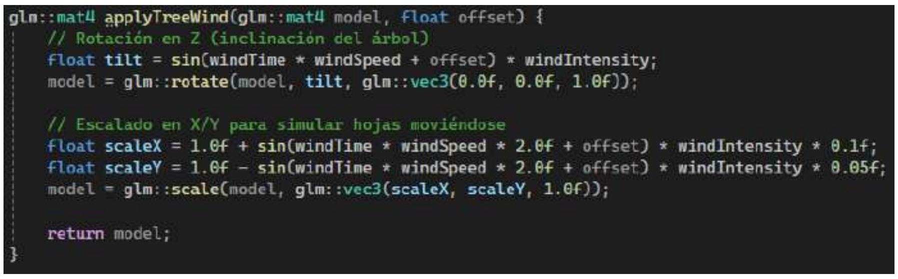
- **actualizarCaidaHojas**: Animates random falling leaves from trees.
    - **Variables**:
        - hojasYOffset: Vertical displacement of each leaf.
        - hojasVelocidadCaida: Fall speed.



```
void actualizarCaidaHojas(float deltaTime) {
    for (int i = 0; i < 3; ++i) {
        hojasYOffset[i] -= hojasVelocidadCaida * deltaTime;

        // Multiplica el límite base por un factor aleatorio en cada reinicio
        float limiteInferiorAleatorio = hojasLimiteInferiorBase * disFactor(gen);

        if (hojasYOffset[i] < limiteInferiorAleatorio) {
            hojasYOffset[i] = hojasLimiteSuperior;
            // No necesitas almacenar el límite, lo recalculas cada vez
        }
    }
}
```

Figure 21. Function animating leaf fall trajectory.

- **applyTreeWind**: Applies wind effects to trees and bushes (tilt and scaling).
    - **Parameters**:
        - model: Model transformation matrix.
        - offset: Phase shift to vary effect between objects.



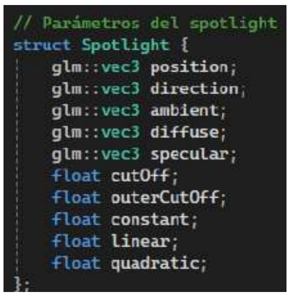
```
glm::mat4 applyTreeWind(glm::mat4 model, float offset) {
    // Rotación en Z (inclinación del árbol)
    float tilt = sin(windTime * windSpeed + offset) * windIntensity;
    model = glm::rotate(model, tilt, glm::vec3(0.0f, 0.0f, 1.0f));

    // Escalado en X/Y para simular hojas moviéndose
    float scaleX = 1.0f + sin(windTime * windSpeed * 2.0f + offset) * windIntensity * 0.1f;
    float scaleY = 1.0f - sin(windTime * windSpeed * 2.0f + offset) * windIntensity * 0.05f;
    model = glm::scale(model, glm::vec3(scaleX, scaleY, 1.0f));

    return model;
}
```

Figure 22. Function simulating wind movement in trees, bushes, and leaves.

14. Spotlight (Flashlight) Configuration

The flashlight is a directional spotlight that follows the camera and is controlled with key F.

- **Spotlight Structure:**
  Defines properties such as position, direction, light cutoffs (inner/outer cone), and attenuation.

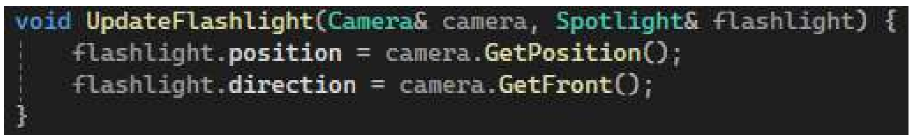


```
// Parámetros del spotlight
struct Spotlight {
    glm::vec3 position;
    glm::vec3 direction;
    glm::vec3 ambient;
    glm::vec3 diffuse;
    glm::vec3 specular;
    float cutOff;
    float outerCutOff;
    float constant;
    float linear;
    float quadratic;
};
```

Figure 23. Estructura spotlight.

UpdateFlashlight: : Updates spotlight position and direction to match the camera.

- **Parámetros**:
  - camera: Reference to the camera.
  - flashlight: Reference to the spotlight.

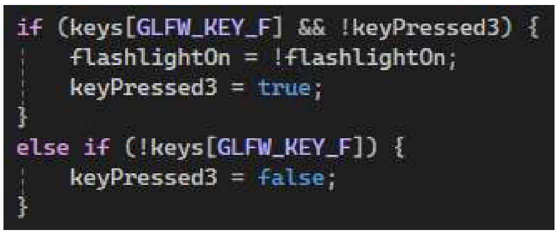


```
void UpdateFlashlight(Camera& camera, Spotlight& flashlight) {
    flashlight.position = camera.GetPosition();
    flashlight.direction = camera.GetFront();
}
```

Figure 24. Function moving the spotlight to the direction the synthetic camera points.

Interaction:

- Key F toggles the flashlight state (flashlightOn).
- In the shader, spotlight properties are sent only if it is activated.



```
if (keys[GLFW_KEY_F] && !keyPressed3) {
    flashlightOn = !flashlightOn;
    keyPressed3 = true;
}
else if (!keys[GLFW_KEY_F]) {
    keyPressed3 = false;
}
```

Figure 25. Controls updating spotlight state.

15. Main Rendering Loop

Inside the program’s main loop:

- deltaTime is updated.
- Animation functions are executed.
- Keyboard and mouse input is processed.
- Color and depth buffers are cleared.
- Appropriate shaders are used to draw the scene.
- Lights are adjusted based on user input.

16. Input Functions

- **KeyCallback**: Detects and responds to pressed keys, such as N to toggle day/night and L to turn lights on/off.
- **MouseCallback**: Controls mouse movement to change camera direction.

17. Termination

When the window loop ends (window is closed or Escape key is pressed), all GLFW resources are terminated and the program ends.

Results

Below are some of the models that were recreated in Maya Autodesk and others that were downloaded from the internet, along with their implementations in the virtual space; also shown are the reference images used for comparison with screenshots of the execution and the list of recreated elements.

Exterior

To recreate the house facade, it was necessary to model the following objects:

- Chimney
- Stairs
- Windows (square, rectangular, large windows)
- L-shaped piece
- Balcony
- Columns
- Handrails
- Doors (main and garage)

Although everything was ultimately combined into a single piece, individual elements were modeled for more precise handling of each part.

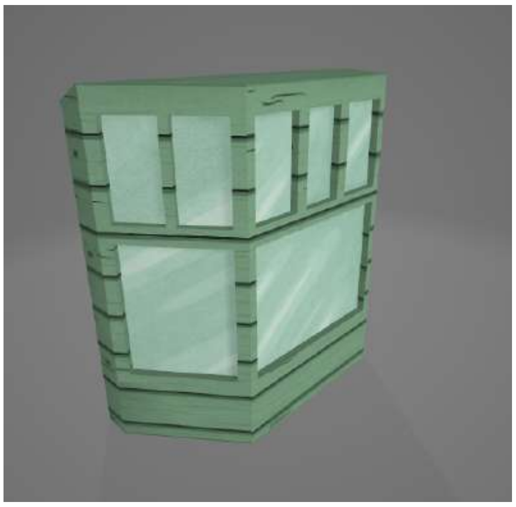


Figure 26. Balcony recreated in Maya

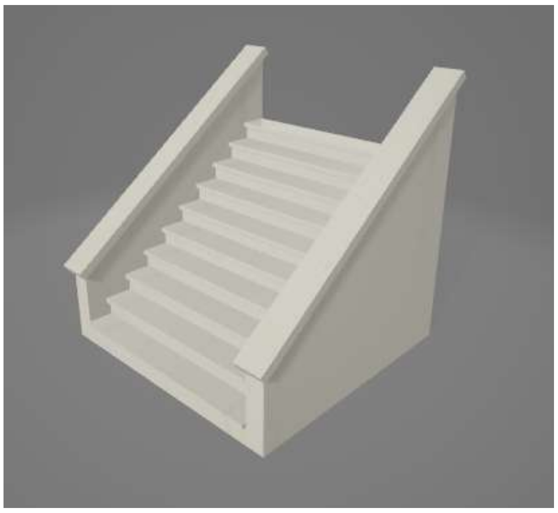


Figure 27. Stairs recreated in Maya

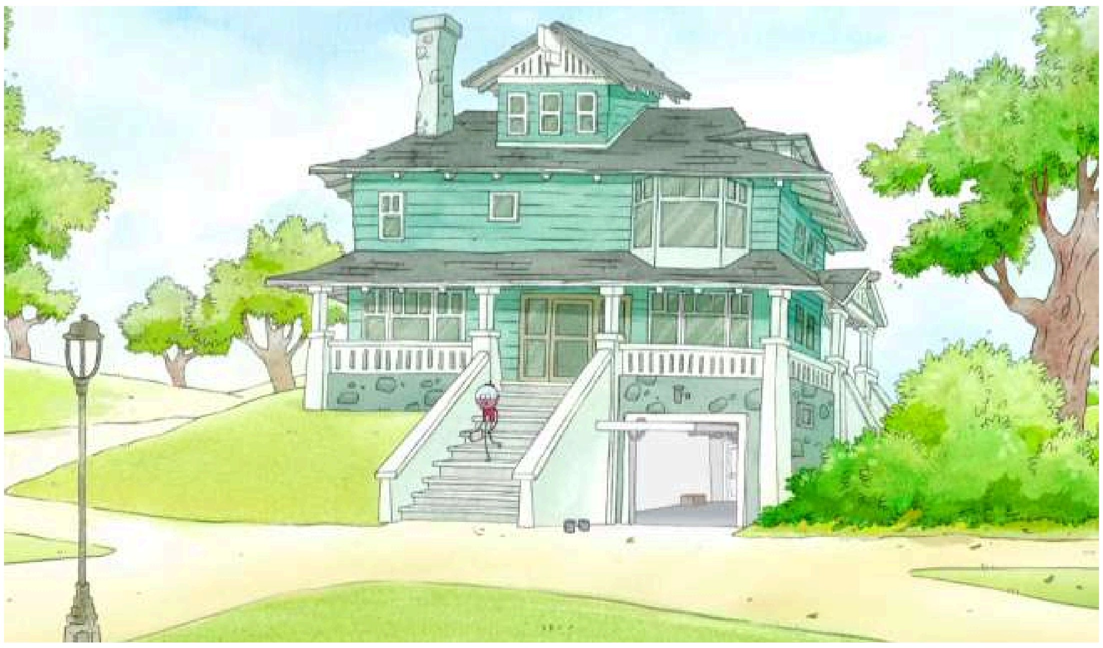


Figure 28. Reference image of the facade

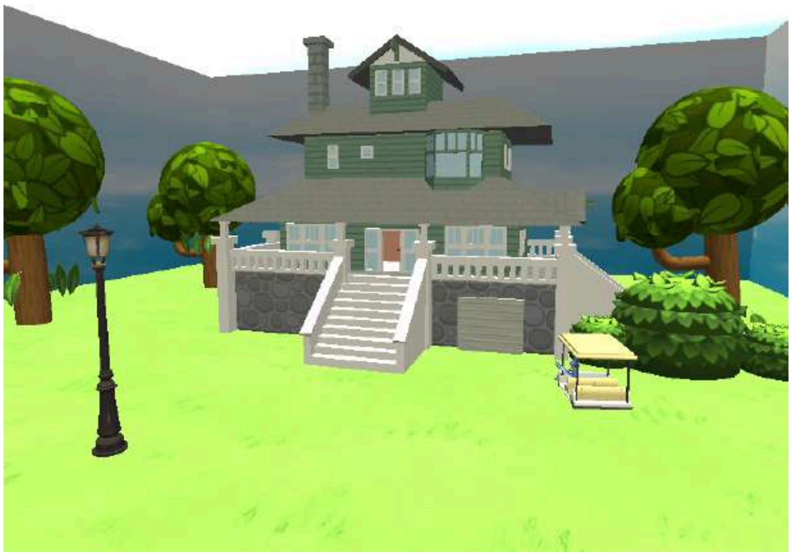


Figure 29. Screenshot of the facade

To add ambiance as seen in the reference image, we decided to import the following models:

- Lighthouse
- Tree
- Bush
- Golf cart

Additionally, we added a model to simulate a skybox to add more details to the house and complement the scene.

Figure 30. Lighthouse imported from Sketchfab

Figure 31. Golf cart imported from Sketchfab

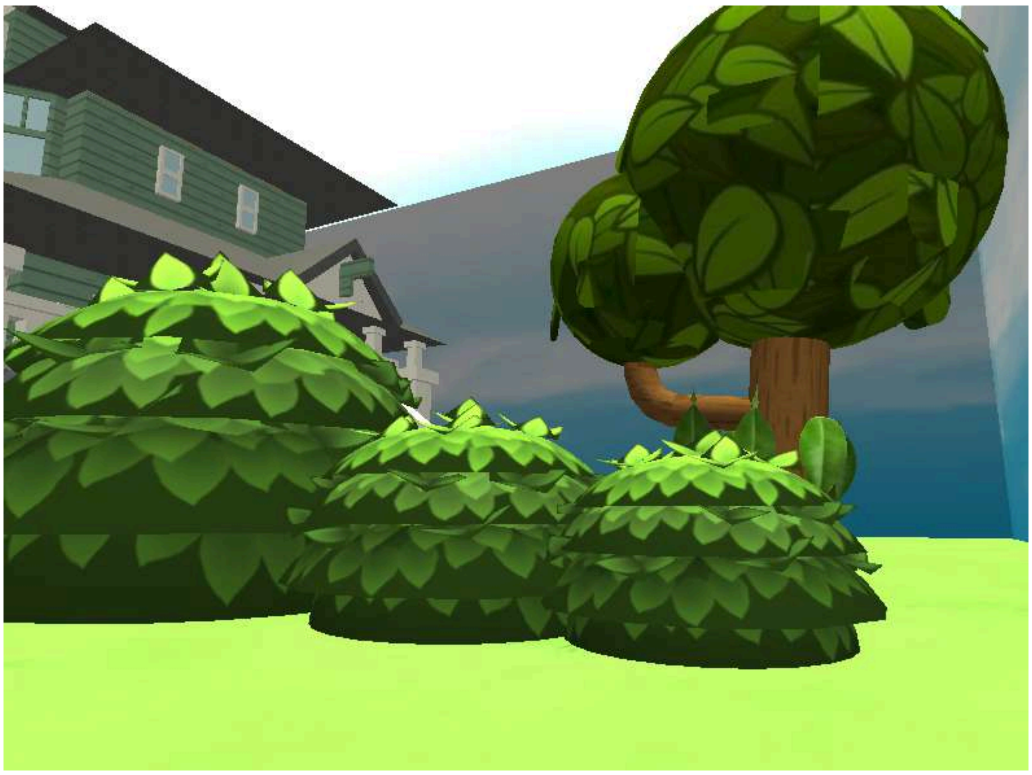


Figure 32. Screenshot of the ambiance

Interior

To recreate the interior (kitchen), it was necessary to model the following objects:

- Floor and ceiling piece
- Wall
- Window
- Wooden wall piece
- Refrigerator
- Table
- Chair
- Lower cabinet
- Upper cabinet
- Large cabinet
- Toaster
- Stove
- Hood

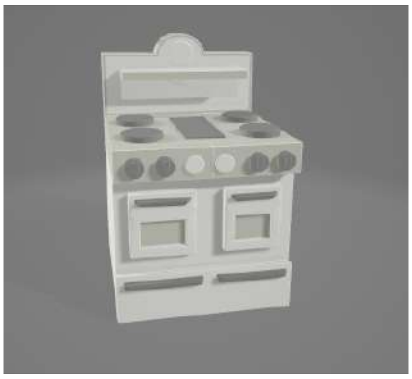


Figure 33. Stove recreated in Maya

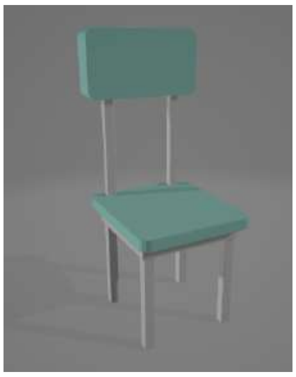


Figure 34. Chair recreated in Maya

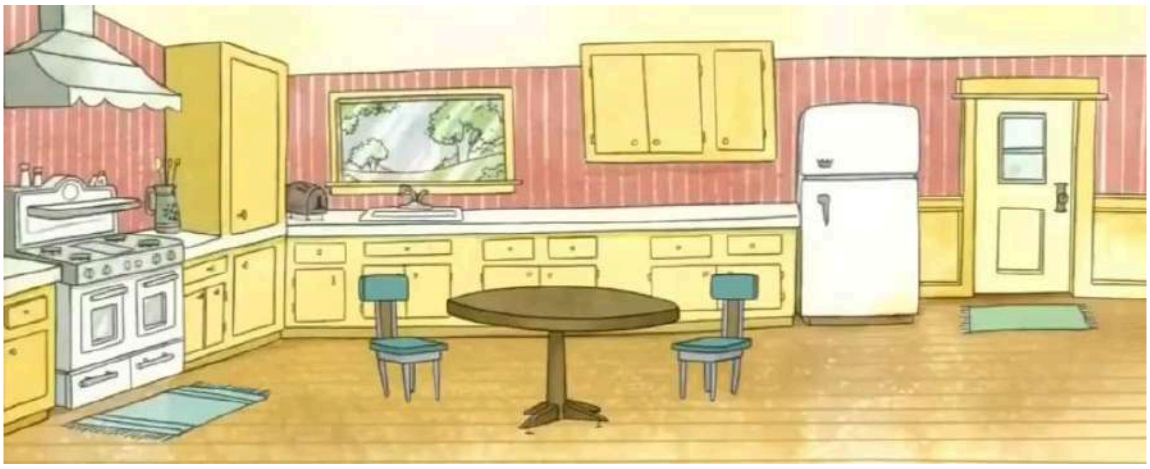


Figure 35. Reference image of the kitchen

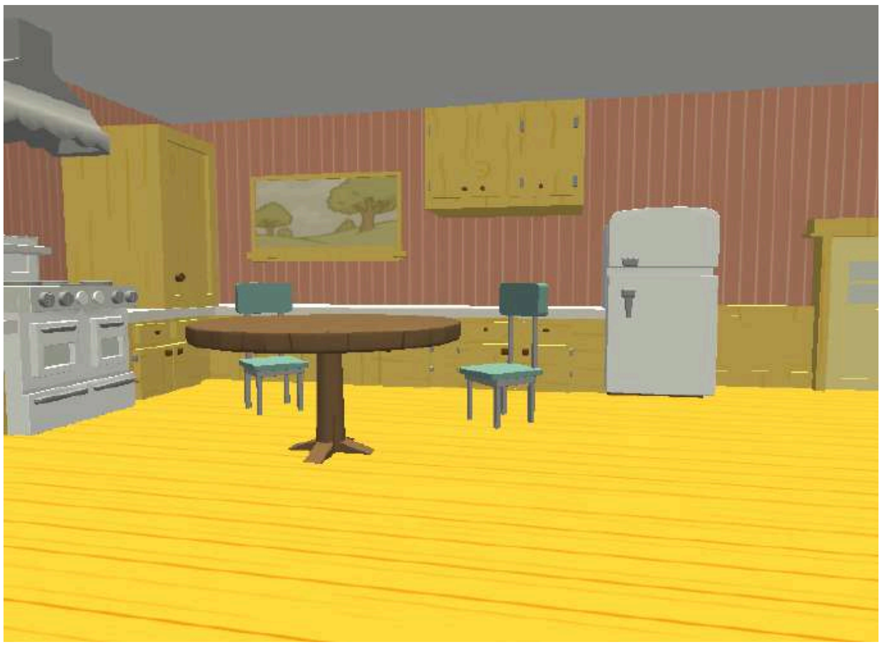


Figure 36. Screenshot of the kitchen

To recreate the bedroom, it was necessary to model the following objects:

- Filing cabinet

- Nightstand
- Lamp
- Radio
- Drawers
- Bed
- Bedroom wall
- Bedroom wall with door
- Rigby
- Bedroom television
- Trampoline
- Bedroom window and curtains

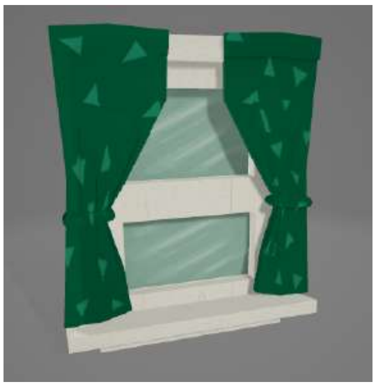


Figure 37. Window and curtains recreated in Maya

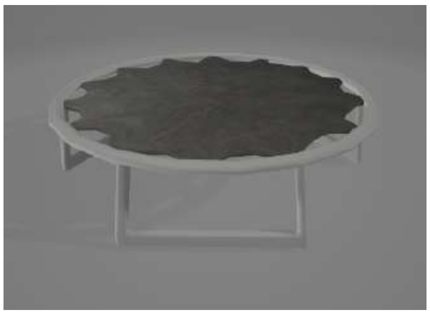


Figure 38. Trampoline recreated in Maya

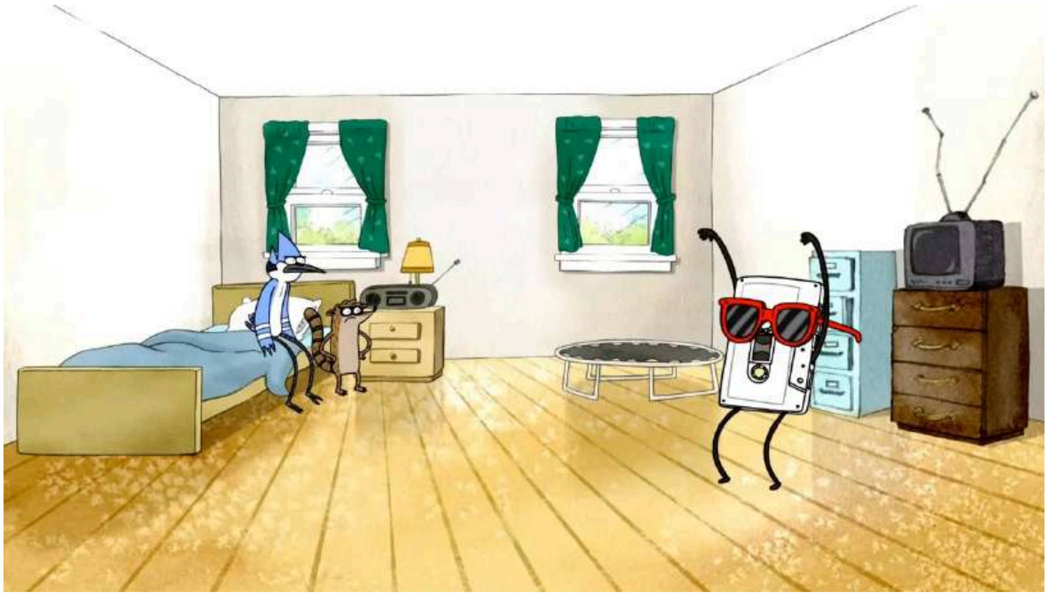


Figure 39. Reference image of the bedroom

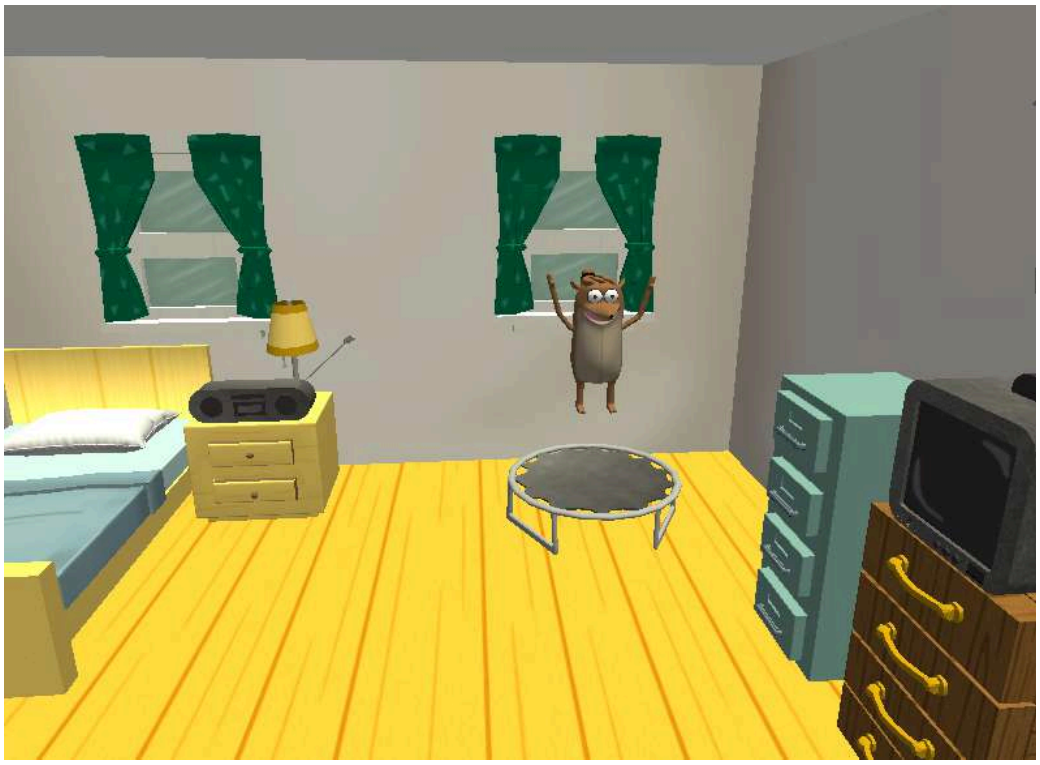


Figure 40. Screenshot of the bedroom

Animations

Below are screenshots illustrating the correct functioning of the animations, along with their exact location within the environment.

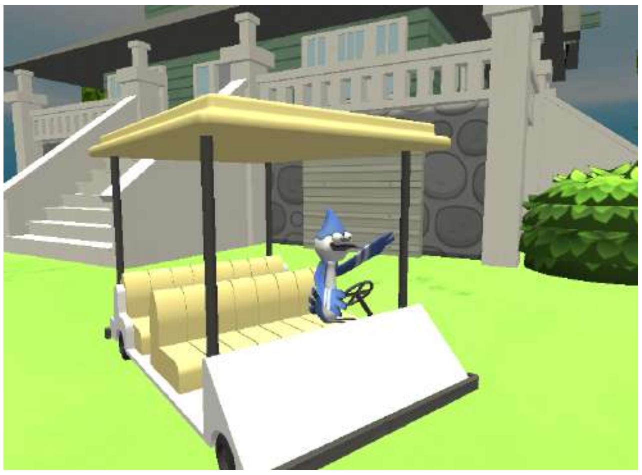


Figure 41. Golf cart drifting animation outside the garage.

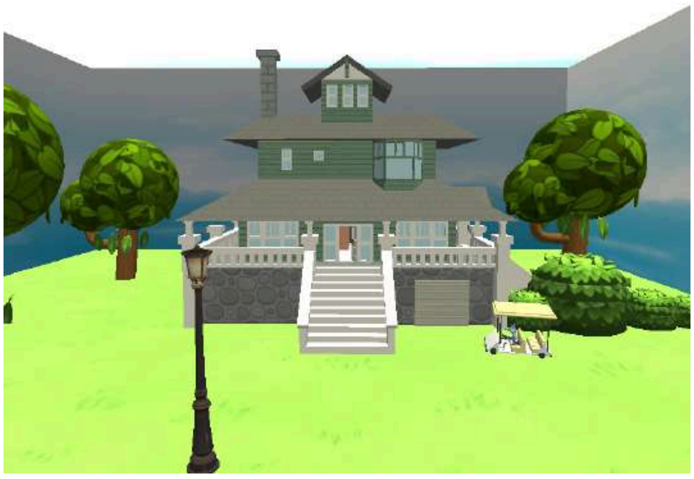


Figure 41. Daylight to night lighting transition animation (day scene).

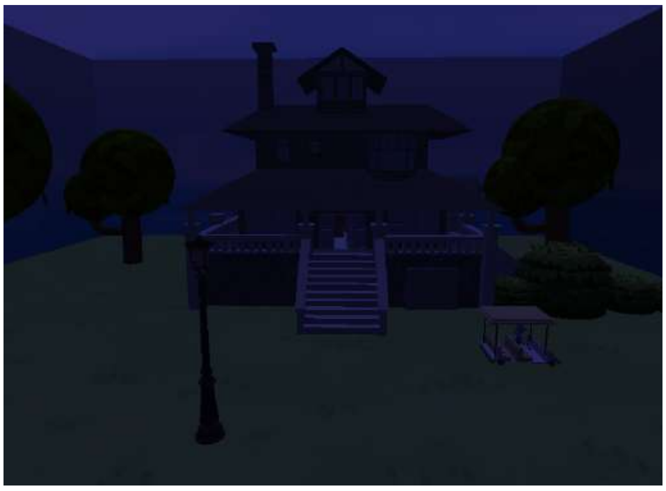


Figure 42. Daylight to night lighting transition animation (night scene).

Figure 43. Vegetation movement animation by wind effect. Falling leaves animation.

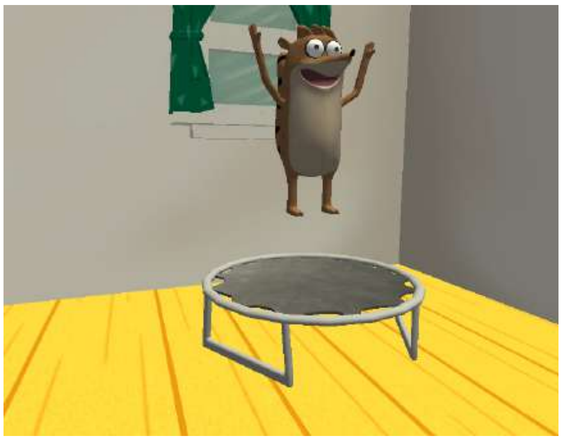


Figure 42. Character jumping animation on the bedroom trampoline.

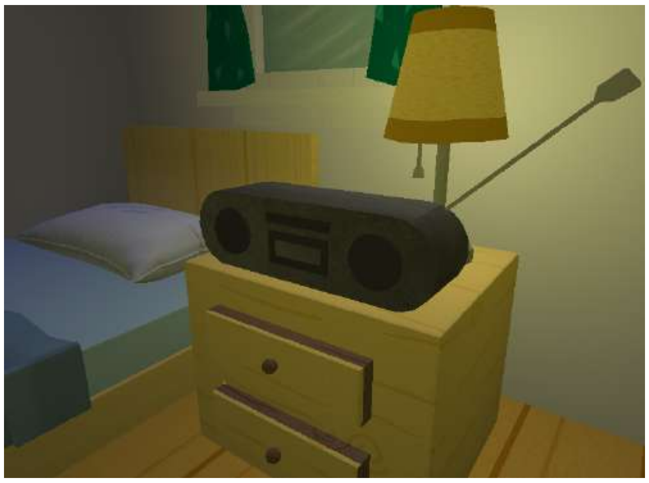


Figure 43. Audio playback from the bedroom radio.

Lighting

Below are the effects of the point lights and spotlights in the scene, set with soft directional lighting.

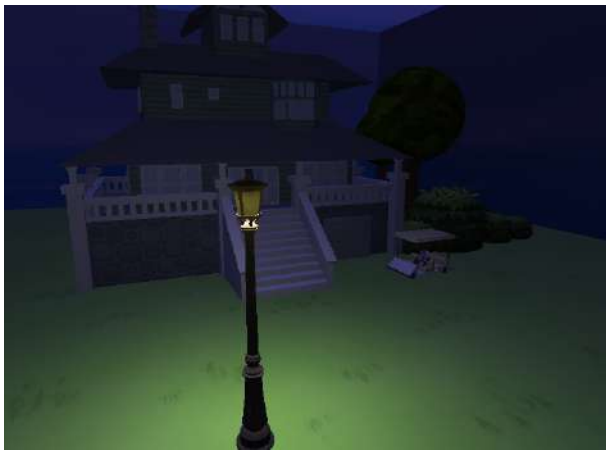


Figure 44. Lighthouse exterior light (Point light source).

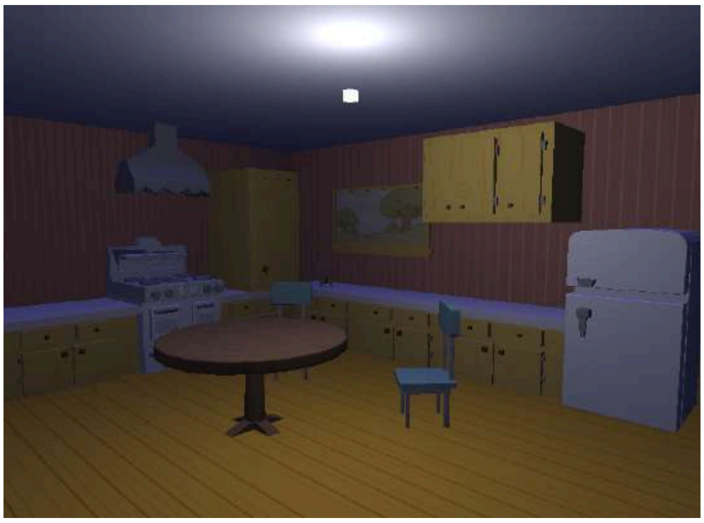


Figure 44. Kitchen light source (Point light source).

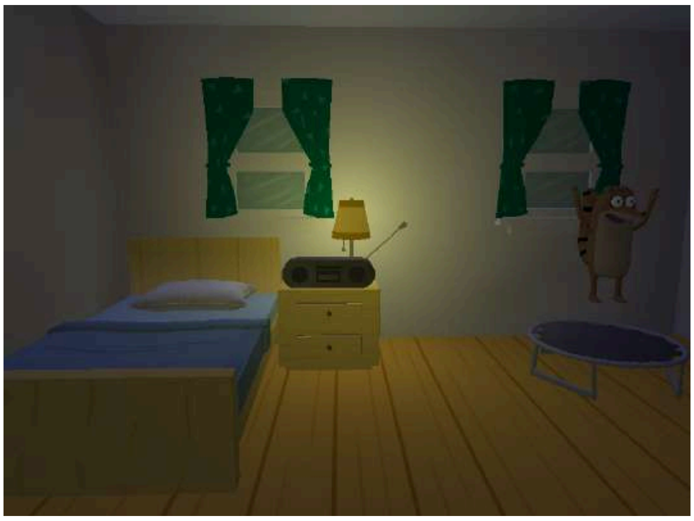


Figure 45. Bedroom lamp light (Point light source).

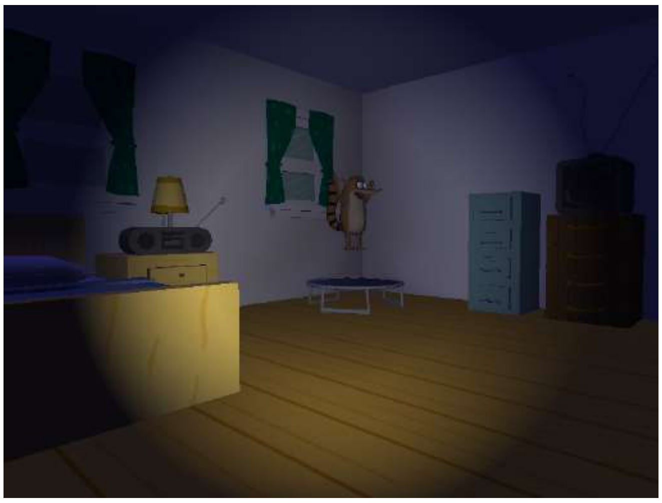


Figure 46. Synthetic camera light (Spotlight source).

Conclusions

With the development of this project, a virtual space was successfully recreated from scratch, applying the knowledge acquired in the computer graphics course. It was a significant challenge, especially in modeling, implementing dynamic lighting, and audio playback within the virtual environment.

Properly configuring animations through functions that apply transformations was an interesting experience, as it allows achieving visual effects that enrich the environment's dynamics and enhance the virtual experience for the user. Previous experience with the synthetic camera and OpenGL functions was fundamental to achieving a satisfactory experience when interacting with the environment; adjusting parameters such as light position, color, and intensity was possible thanks to what was learned in the lab. Although technical difficulties were encountered, such as issues with normals, materials, and textures when exporting from Maya, these were resolved through multiple unit tests, adjusting various parameters until everything worked correctly.

The knowledge gained from this project could be applied to larger projects involving 3D environment development, modeling, or even to continue improving this same program by adding more interactions and turning it into a simple video game.

References

[1] Buy Autodesk Maya 2026 Software | 3D Animation Software. (s. f.). https://www.autodesk.com/products/maya/overview

[2] Comment, D. S. J. 1. 2. 0. (s. f.). *Las 10 mejores plantillas de estimación de costos con muestras y ejemplos*. https://www.slideteam.net/blog/las-10-mejores-plantillas-de-estimacion-de-costos-con-muestras-y-ejemplos

[3] Montilla, C. (2020, 18 marzo). *Cuánto cobrar por un Render / Modelado 3D - [El método 100% efectivo]*. Yo Soy Arquitecto. https://yosoyarquitecto.com/cuanto-cobrar-por-un-render-modelado/

[4] Pricing ChatGPT. (s. f.). OpenAI. https://openai.com/es-419/chatgpt/pricing/

[5] "Rigby || Regular Show" (https://skfb.ly/pqs6T) by metekrts is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

[6] "Mordecai" (https://skfb.ly/ozw7D) by D.p is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

[7] "Stylized Bush" (https://skfb.ly/oTDpD) by Daniel is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

[8] "Pillow" (https://skfb.ly/6ntWQ) by 3DMish is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

[9] "Lemon Leaf" (https://skfb.ly/6pJnz) by YouniqueĪdeaStudio is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

[10] "Cart || Regular Show" (https://skfb.ly/pqssG) by metekrts is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).