Universidad Nacional Autónoma de México

Facultad de Ingeniería

Ingeniería en Computación

Computación Grafica e Interacción Humano Computadora

Final Project – Technical Manual

Student:

ID number: 319095706

Group 05

Date: 25/11/2025

Index

Objective.

The objective of this project is to identify and integrate each stage of the graphics pipeline, correctly configuring and using transformation matrices to manipulate the scene, applying a lighting model that considers ambient, diffuse, and specular components, and implementing interactive camera controls that allow exploration of the 3D environment; in addition, it aims to model three-dimensional objects in Maya, apply textures and materials to them by appropriately using texture maps and UV coordinates, adjust visual properties such as brightness, diffuse component, specular component, and opacity, implement different types of light to enhance the aesthetics and realism of the scene by simulating interior and exterior lighting conditions, and finally assemble all modeled objects into a coherent and visually appealing final composition.

Introduction.

This manual describes the recreation of a house and two 3D spaces. Its purpose is to document in a structured manner the architecture of the project, the dependencies used, the organization of the code, and the modifications made to obtain the 3D house, objects, and rooms. The recreated façade is the house of the Griffin family from the animated cartoon “Family Guy,” and the recreated spaces/rooms are the main living room of the house and the kitchen.

This document is not intended for the end user, but rather for those who need to understand or provide support for the project: it includes a description of the development environment, directory structure, relevant source files, and the compilation/execution process. In this way, the reproducibility of the project and the traceability between the laboratory requirements and the implementation carried out are ensured.

Project Scope.

The project involves the recreation in OpenGL of the living room and kitchen of the Family Guy house based on visual references, preserving the general layout of the space, its characteristic ambiance, and the integration of the elements that appear in the original scene. This means that the environment is not generic, but is built specifically to resemble the setting from the series, including walls, floor, doors, and the objects that structure the room.

Within this environment, at least five objects per room are modeled and placed, each one clearly identifiable in the references (screenshots from the series), with their corresponding transformations and with materials or textures consistent with the cartoon style of the show. In addition, the use of the laboratory's base code and the integration of camera and lighting are considered, so that the scene can be correctly navigated or visualized within the engine used throughout the course.

The project also covers the implementation of the required animations: animations generated from the code, meaningful within the scene and not merely simple, contextless translations.

Limitations.

During the development of the project, several limitations arose. The first was the dependence on the show's visual references: since the settings are entirely in 2D, many proportions, depths, and side views are not explicitly depicted, so for those details I had to rely on imagination to complete the objects. Another limitation was the use of the Maya software, which was initially approached with no prior experience; however, as the laboratory activities progressed and the first objects for the project were created, I gradually became more familiar with the environment and all the tools it offers.

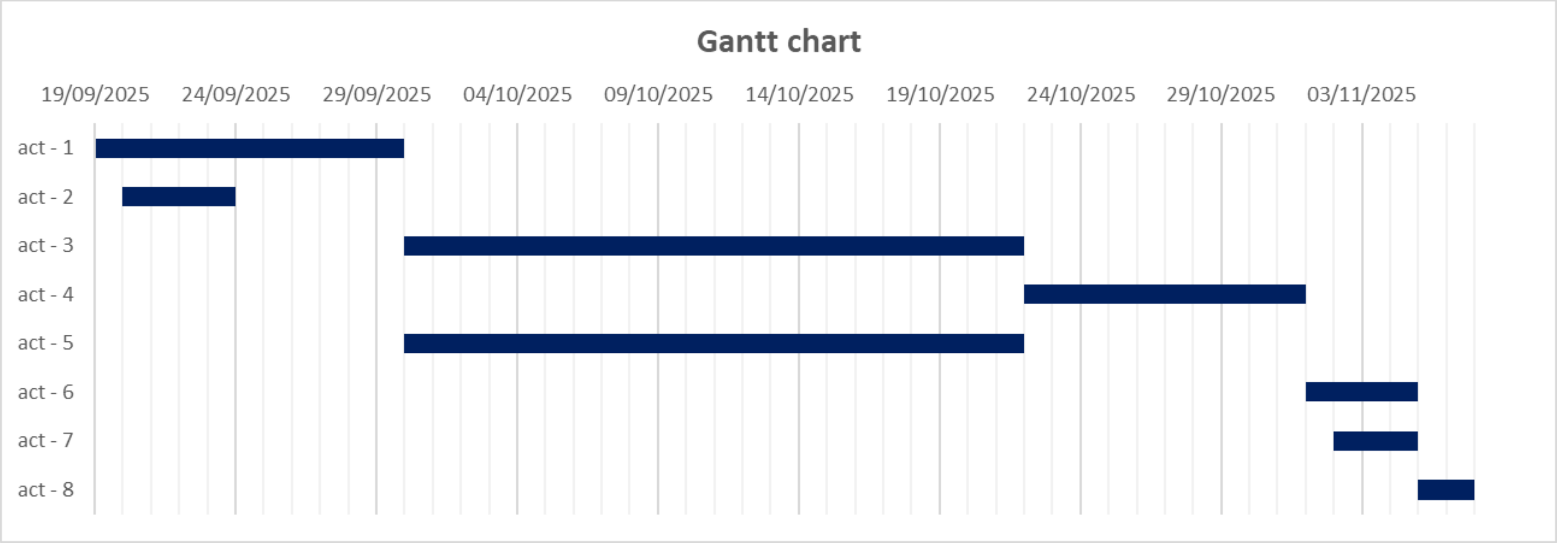
Applied Software Development Methodology.

Two methodologies were used in a complementary way to develop the project. For the house, Iterative Development was applied, since new modeling tools and lighting concepts were learned as the semester progressed. For this reason, the house was repeatedly rebuilt and improved in cycles: first a simple sketch, then a more structured version, later the integration of additional details with curves, and finally the application of textures and visual adjustments.

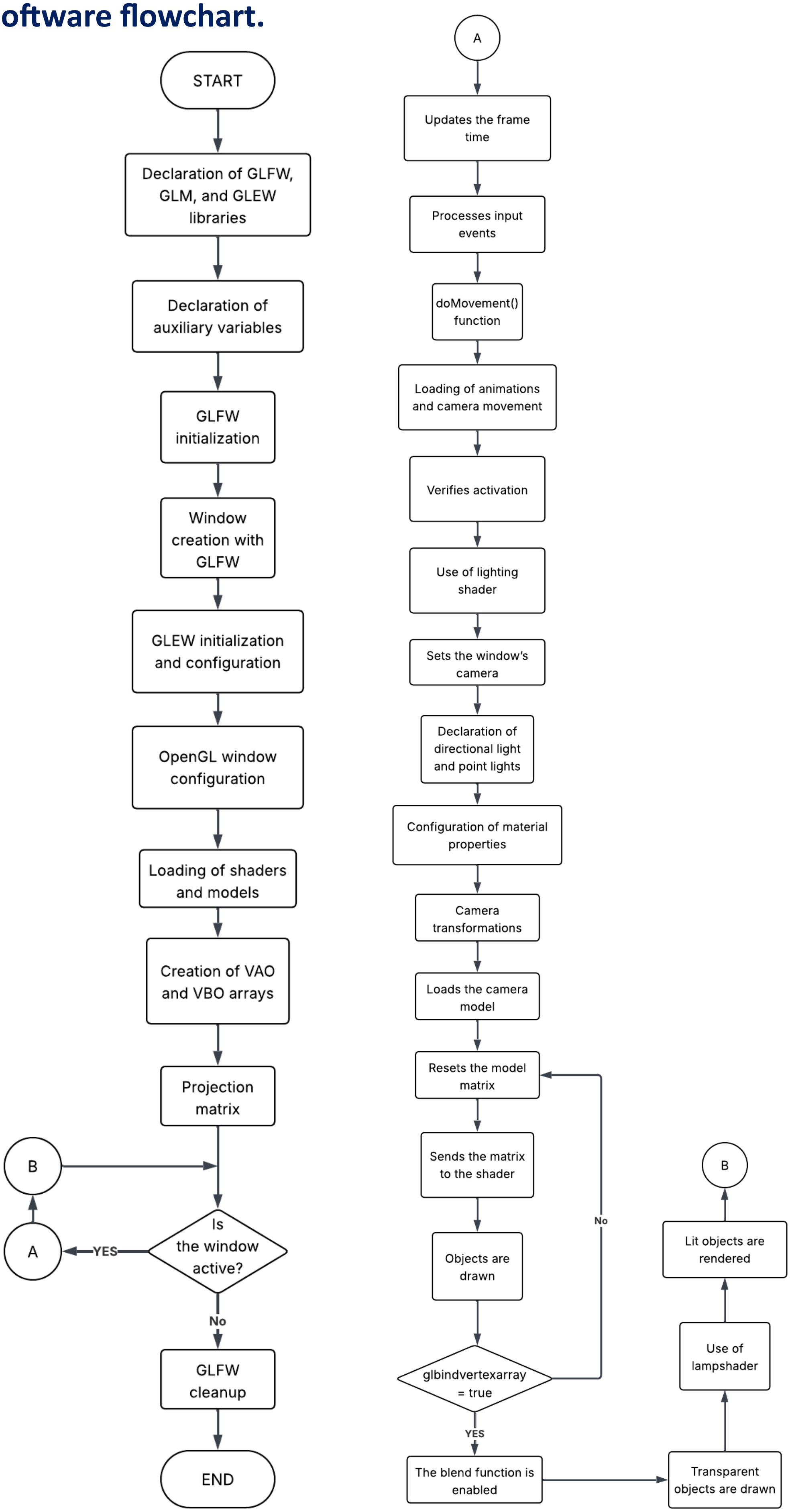
On the other hand, for the individual models, the SCRUM methodology was adapted, mainly using weekly sprints with clear goals: to model and texture one object in each sprint. At the end of every sprint, the quality of the model was reviewed; if it did not meet expectations, improvements were scheduled for the following week along with the new object. When delays occurred, the sprints were reorganized to complete all the models, reserving one sprint for general corrections and a final one to place all the objects on the house and perform the render in OpenGL, constantly assessing how they looked together with the applied lighting.

Gantt Chart.

Activities	Description	Days	Start	End
act - 1	Start modeling the objects.	11	19/09/2025	30/09/2025
act - 2	Start modeling the house.	4	20/09/2025	24/10/2025
act - 3	Texturing the objects.	22	30/09/2025	22/10/2025
act - 4	Texturing the house walls.	10	22/10/2025	01/11/2025
act - 5	Completion of modeling.	22	30/09/2025	22/10/2025
act - 6	Completion of texturing.	4	01/11/2025	05/11/2025
act - 7	Final details on the house and objects.	3	02/11/2025	05/11/2025
act - 8	Code implementation.	2	05/11/2025	07/11/2025



Software flowchart.


START
Declaration of GLFW, GLM, and GLEW libraries
Declaration of auxiliary variables
GLFW initialization
Window creation with GLFW
GLEW initialization and configuration
OpenGL window configuration
Loading of shaders and models
Creation of VAO and VBO arrays
Projection matrix
B
Is the window active?
YES
A
No
GLFW cleanup
END
A
Updates the frame time
Processes input events
doMovement() function
Loading of animations and camera movement
Verifies activation
Use of lighting shader
Sets the window's camera
Declaration of directional light and point lights
Configuration of material properties
Camera transformations
Loads the camera model
Resets the model matrix
Sends the matrix to the shader
Objects are drawn
glbindvertexarray = true
No
YES
The blend function is enabled
Transparent objects are drawn
Use of lampshader
Lit objects are rendered
B

Tools Used.

Autodesk Maya 2023.

Professional 3D modeling, animation, and rendering software. Used to create meshes (objects), adjust their UVs, assign basic materials, and even animate them.

Visual Studio 2022.

IDE (integrated development environment) for programming in C++ (and other languages) on Windows. It provides an editor, syntax highlighting, autocompletion, project manager, compiler, and debugger.

OpenGL.

A cross-platform graphics API for drawing 2D and 3D graphics using the GPU. It is a set of functions that the code calls to create render windows, send vertices, apply textures, lights, etc.

C++ language.

The programming language used to write all the project logic: window initialization, model loading, camera movement, animations, OpenGL calls, etc.

Graphics and utility libraries.

- GLFW (window, input, and event handling)
- GLEW (OpenGL extensions)
- GLM (OpenGL-based graphics mathematics)
- ASSIMP (3D model import)
- SOIL2 (texture loading)

Dictionary of Functions.

KeyCallback: Used to read keyboard input.

MouseCallback: Reads input from mouse movement.

DoMovement: Executes the movement of the animations.

Data Dictionary.

Dato	Descripción
WIDTH	Window width value.
HEIGHT	Window height value.
SCREEN_WIDTH, SCREEN_HEIGHT	Values that store the window height and width.
camera	Represents the camera in 3D space.
lastX, lastY	Store the coordinates of the last mouse cursor position.
keys	Stores the state of the keys.
firstMouse	Indicates whether it is the first time mouse movement is detected.
doorOpen, doorCurrentAngle, doorTargetAngle, prevP	Variables for the door animation (rotation).
garageOpen, prevO, garageCurrentRotX, garageCurrentY, garageCurrentZ, garageTargetRotX, garageTargetY, garageTargetZ	Variables for the garage door animation (rotation and translation).
microOpen, microCurrentAngle, microTargetAngle, prevU	Variables for the microwave door animation (open/close).
mugAngle	Variable for the mug animation (rotation).

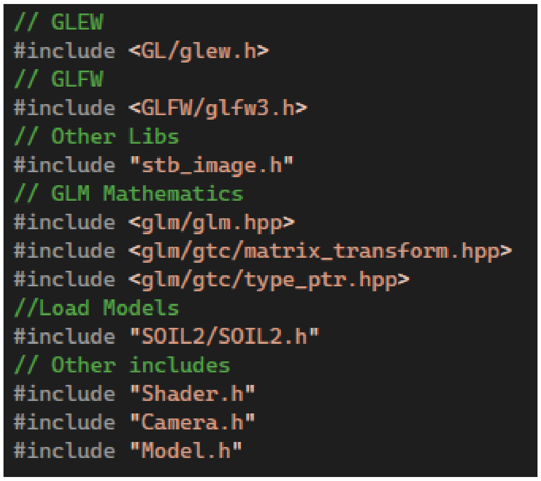
toastJumping, toastTime, toastYOffset, toastRotX, prevT, toastJumpDuration, toastJumpHeight, PI, TWO_PI	Variables for the toast animation.
deltatime	The time that has elapsed between the current frame and the previous frame.
lastFrame	Stores the time of the last processed frame
window	Pointer to the window created by GLFW.
model	Model matrix used to transform vertices from object space to world space.
lightingShader	Instance of a shader used to render the lighting of the objects
lampShader	Instance of a shader used to render the lamp or light source.
PisoSala, PisoCocina, Pisoizq, Pisoder, paredSalaIzq, paredSalaDer, paredSalaEsc, pareddetrasesc, paredfrontal, paredfrentelineas, paredtraseracasa, paredfachadaizquierda, paredEntradaCocina, marcopuertaAcocina, paredizqcocina, paredatrascocina, paredfrentecocina, ventana1, ventana2, ventana3, ventana4, ventana5, ventana6, ventana7, ventana8, ventana9, puertafueraizq, marcopuerta, puertaroja, salatecho, techo1, techo2, triangulo1, triangulo2, ventana10, ventana11, ventana12, ventana13, ventana14, ventana15, ventana16, ventana17, casitaarribader, casitaarribaizq,	OBJ models used to represent them in three-dimensional space.

casitaarribagrande, casitaizq, techocasitaizq, puertacasitaizq, cochera, casitader, techocasitader, puertaAcocina, escalera, barandal, mesalamp, base, cabeza, sillon1, sillon2, mesarectangular, tapete, muebletv, tv, refri, cocinaintegralarriba1, cocinaintegralarriba2, cocinaintegralabajo1, cocinaintegralabajo2, cocinaintegralabajo3, basearriba1, basearriba2, basearriba3, stove, ventanainteriorcocina, microwave, platomicrowave, puertamicro, mug, tostadora, toast.	

Code Documentation.

The program implements a 3D scene rendered with OpenGL, using GLFW for window creation, GLEW for loading modern OpenGL functions, and GLM for 3D mathematics. On this foundation, multiple models (Model.h) are loaded to represent the elements of the house (floors, walls, windows, furniture) and are controlled with a first-person camera (Camera.h).

Libraries and Files

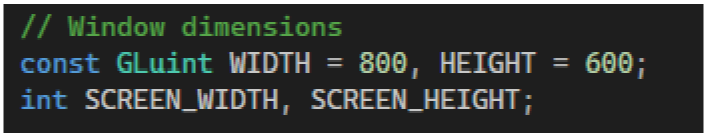


```
// GLEW
#include <GL/glew.h>
// GLFW
#include <GLFW/glfw3.h>
// Other Libs
#include "stb_image.h"
// GLM Mathematics
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
//Load Models
#include "SOIL2/SOIL2.h"
// Other includes
#include "Shader.h"
#include "Camera.h"
#include "Model.h"
```

- GLFW (<GLFW/glfw3.h>): Used for creating the window, managing the OpenGL context, and capturing input events (keyboard and mouse).
- GLEW (<GL/glew.h>): Used to load modern OpenGL functions and facilitate the use of extensions.
- GLM (<glm/...>): Mathematics library for graphics that provides types and operations for vectors, matrices, and 3D transformations (translation, rotation, projection).
- stb_image.h and SOIL2: Used for loading textures from image files.
- Shader.h, Camera.h, and Model.h: Helper classes that encapsulate, respectively, shader management, camera control, and the loading/drawing of 3D models.

Declaration of global variables.

- **Window dimensions.**

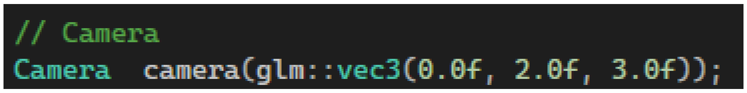


```
// Window dimensions
const GLuint WIDTH = 800, HEIGHT = 600;
int SCREEN_WIDTH, SCREEN_HEIGHT;
```

`WIDTH` and `HEIGHT` are the logical dimensions of the window.

`SCREEN_WIDTH` and `SCREEN_HEIGHT` store the actual framebuffer dimensions, which are needed to configure the viewport.

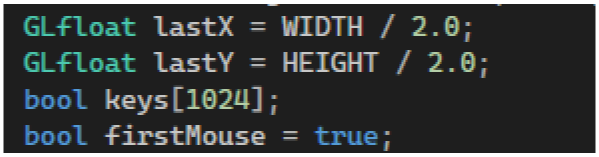
- **Camera**



```
// Camera
Camera  camera(glm::vec3(0.0f, 2.0f, 3.0f));
```

A camera is instantiated at a position slightly elevated above the floor, which allows the scene to be viewed from the start. The camera is used to generate the view matrix in each frame.

- **Input handling**

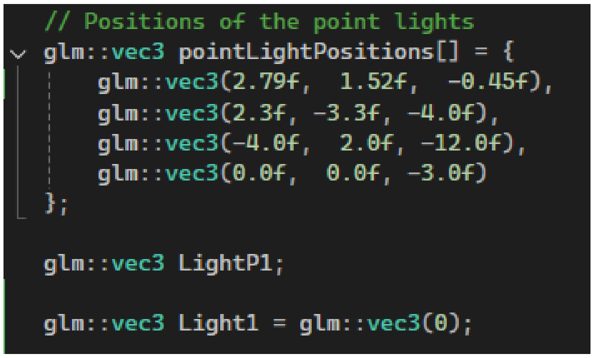


```
GLfloat lastX = WIDTH / 2.0;
GLfloat lastY = HEIGHT / 2.0;
bool keys[1024];
bool firstMouse = true;
```

The `keys[]` array stores the state (pressed/not pressed) of each key.

The mouse-related variables (`firstMouse`, `lastX`, `lastY`) make it possible to calculate cursor offsets and thus modify the camera's orientation.

- **Lighting**



```
// Positions of the point lights
glm::vec3 pointLightPositions[] = {
    glm::vec3(2.79f,  1.52f,  -0.45f),
    glm::vec3(2.3f, -3.3f, -4.0f),
    glm::vec3(-4.0f,  2.0f, -12.0f),
    glm::vec3(0.0f,  0.0f, -3.0f)
};

glm::vec3 LightP1;

glm::vec3 Light1 = glm::vec3(0);
```

\`pointLightPositions\` contains the fixed positions of four point lights that will be rendered in the scene.

\`Light1\` and \`active\` make it possible to enable or disable the color contribution of one of the lights dynamically via the keyboard.

- **Animation control**
  - doorOpen indicates the logical state (open/closed).
  - doorTargetAngle is the target angle (0° or 90°).
  - doorCurrentAngle is the angle actually used to draw in that frame.
  - prevP prevents the animation from being triggered multiple times by a single key press.
  - The same scheme is used for the garage, but with three components (one rotation and two translations), since the animation is more complex.

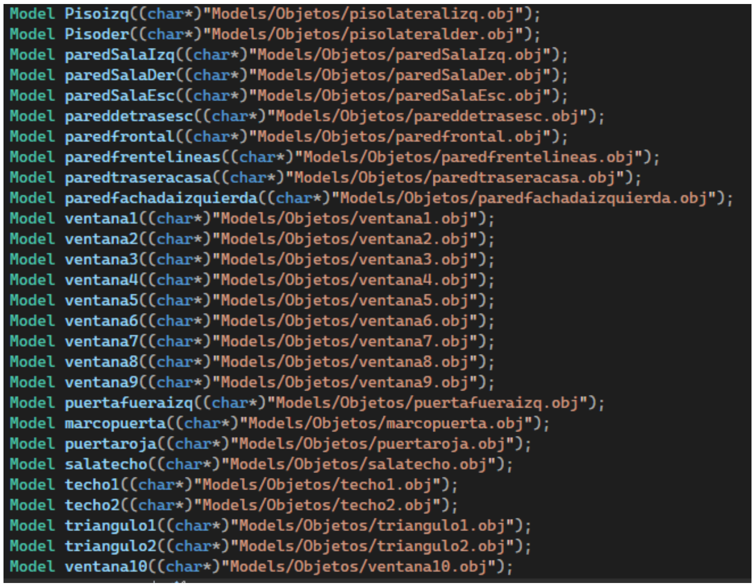
- **Time control**

deltaTime and lastFrame make it possible to calculate the time elapsed between one frame and the next. In this way, camera movement and animations become independent of the frame rate.

- **main() function**

The main() function brings together the initialization phase, resource loading, and the main rendering loop.

1. Initialization of GLFW and window creation:
    - The GLFW library is initialized.
    - It is specified that OpenGL 3.3 will be used in core profile.
    - The main window is created. If this fails, the program stops and frees the resources.
    - The window is bound to the current context through glfwMakeContextCurrent(window), and the keyboard and mouse callbacks are registered. The cursor is also disabled to allow a “first-person” style of control.
2. Initialization of GLEW and OpenGL configuration:
    - GLEW is initialized once a valid context exists.
    - The viewport is configured with the framebuffer dimensions.
    - Depth testing is enabled so that rendering respects the distance of the objects.
3. Creation of shaders and loading of models:
    - Two shaders are instantiated:
        - lightingShader: responsible for rendering the scene objects with lighting.
        - lampShader: used to draw the indicators of the point lights.
    - All the required 3D models are then loaded. This loading is done outside the rendering loop to avoid runtime overhead. Each Model instance corresponds to an element of the house or its furniture.



```
Model Pisoizq((char*)"Models/Objetos/pisolateralizq.obj");
Model Pisoder((char*)"Models/Objetos/pisolateralder.obj");
Model paredSalaIzq((char*)"Models/Objetos/paredSalaIzq.obj");
Model paredSalaDer((char*)"Models/Objetos/paredSalaDer.obj");
Model paredSalaEsc((char*)"Models/Objetos/paredSalaEsc.obj");
Model pareddetrasesc((char*)"Models/Objetos/pareddetrasesc.obj");
Model paredfrontal((char*)"Models/Objetos/paredfrontal.obj");
Model paredfrentelineas((char*)"Models/Objetos/paredfrentelineas.obj");
Model paredtraseracasa((char*)"Models/Objetos/paredtraseracasa.obj");
Model paredfachadaizquierda((char*)"Models/Objetos/paredfachadaizquierda.obj");
Model ventana1((char*)"Models/Objetos/ventana1.obj");
Model ventana2((char*)"Models/Objetos/ventana2.obj");
Model ventana3((char*)"Models/Objetos/ventana3.obj");
Model ventana4((char*)"Models/Objetos/ventana4.obj");
Model ventana5((char*)"Models/Objetos/ventana5.obj");
Model ventana6((char*)"Models/Objetos/ventana6.obj");
Model ventana7((char*)"Models/Objetos/ventana7.obj");
Model ventana8((char*)"Models/Objetos/ventana8.obj");
Model ventana9((char*)"Models/Objetos/ventana9.obj");
Model puertafueraizq((char*)"Models/Objetos/puertafueraizq.obj");
Model marcopuerta((char*)"Models/Objetos/marcopuerta.obj");
Model puertaroja((char*)"Models/Objetos/puertaroja.obj");
Model salatecho((char*)"Models/Objetos/salatecho.obj");
Model techo1((char*)"Models/Objetos/techo1.obj");
Model techo2((char*)"Models/Objetos/techo2.obj");
Model triangulo1((char*)"Models/Objetos/triangulo1.obj");
Model triangulo2((char*)"Models/Objetos/triangulo2.obj");
Model ventana10((char*)"Models/Objetos/ventana10.obj");
```

4. Configuration of buffers (VAO, VBO, EBO)

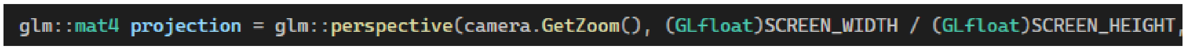
   A VAO and the associated buffers are created to store the data of a cube. This cube is reused to draw the lamps (small cubes that mark the positions of the point lights). The vertex attributes are defined: position, normal, and texture coordinates.

5. Loading of textures

   Two textures (diffuse and specular) are created and configured using stbi_load. Wrapping and filtering parameters are set, and the textures are assigned to the units expected by the shader.

6. Creation of the projection matrix

   This matrix remains constant during execution and is sent to the shader in every frame.



```
glm::mat4 projection = glm::perspective(camera.GetZoom(), (GLfloat)SCREEN_WIDTH / (GLfloat)SCREEN_HEIGHT,
```

- **Main rendering loop.**

The loop runs while the window has not received the close command:

1. Per-frame time calculation

   `deltaTime` is determined from the current time and the time of the previous frame.
2. Event processing and interaction
   - glfwPollEvents() processes the input events captured by GLFW.
   - DoMovement() updates the camera position and the state of the animations based on the keys stored in keys[].
3. Drawing buffer clearing

   The color buffer and the depth buffer are cleared to begin the new frame.
4. Activation of the lighting shader and parameter submission

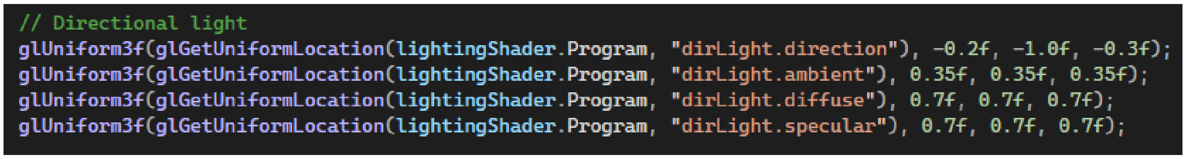
   lightingShader is activated and the following are sent:

   - the camera position,
   - the directional light parameters,
   - the parameters of the four point lights,
   - the spotlight linked to the camera,
   - the material properties (shininess, textures).

   - Directional light:

     A directional light simulates a light source that is “infinitely far away,” such as the sun. Because of that “infinite” distance, the light rays reach the entire scene in parallel. That is why a direction is specified instead of a position.

     This light affects all objects equally, without losing intensity over distance.



```
// Directional light
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.35f, 0.35f, 0.35f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.7f, 0.7f, 0.7f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.7f, 0.7f, 0.7f);
```

The three components that are sent (ambient, diffuse, and specular) correspond to the classic lighting model:

- Ambient: Background or ambient light. Ensures that even parts that do not receive direct light are not completely black.
- Diffuse: Diffuse light. Depends on the angle between the light and the surface normal.
- Specular: Specular highlight. Produces the brightest reflections.

- Pointlights

As with the directional light, each point light can have:

- ambient: the minimum light level it contributes.
- diffuse: the main illumination.
- specular: the highlight.

5. Construction of the view and projection matrices

   The view matrix is obtained from the camera object. Both matrices are sent to the shader.

6. Drawing of the models

   For each model, the following steps are performed:

   - Initialization of the model matrix (`glm::mat4(1)`),
   - Application of the necessary transformations (for the door and the garage, the animated transformations are applied),
   - Sending of the matrix to the shader,
   - Call to `Draw(...)`.

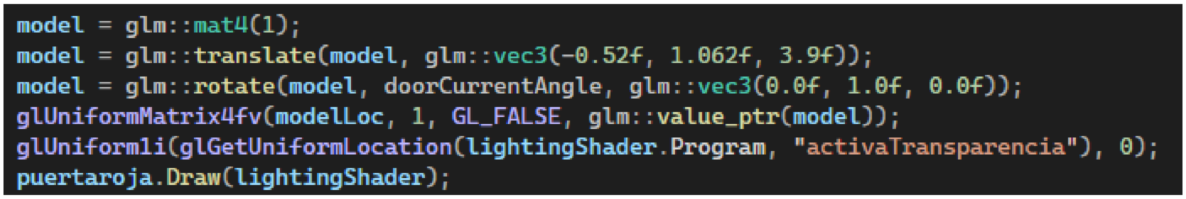
   For certain objects, blending is enabled to allow transparency and then disabled again.

- **Animated models**

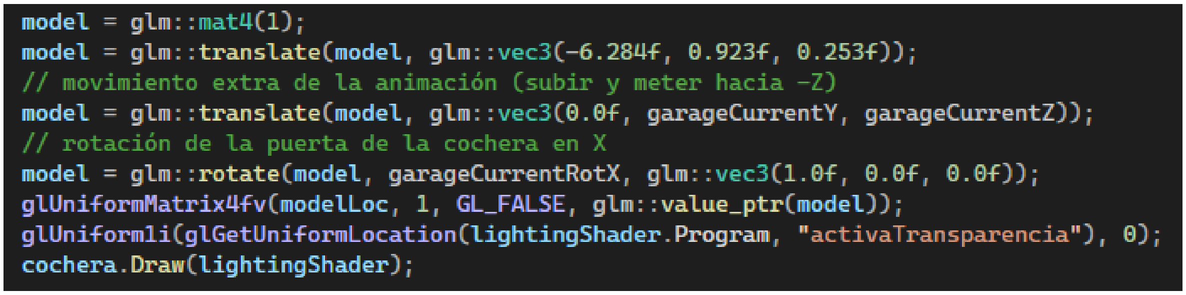
Objects with dynamic transformations:

- Door: Rotation around the Y axis to simulate the door’s movement.
- Garage door: Rotation around the X axis and translation along the Y and Z axes to simulate the movement of a garage door.

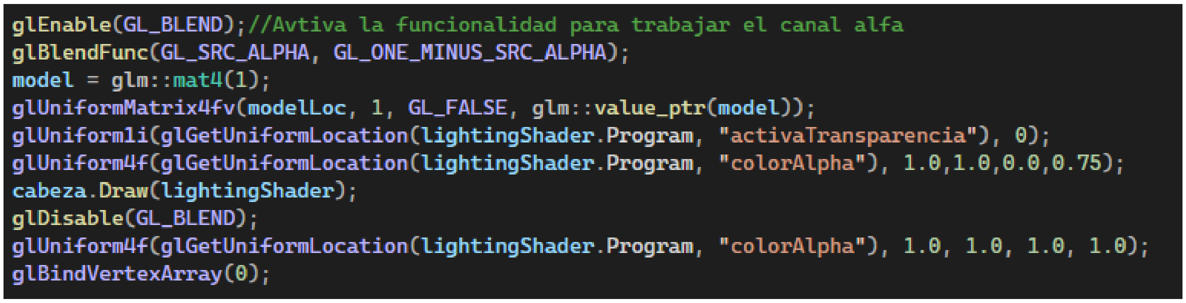
- Lamp: Transparency to simulate a lamp using the point light.
- Microwave and mug: Door rotation to simulate opening and closing the microwave, and rotation of the mug inside the microwave.
- Toast in the toaster: Jump and spin of the toast as it pops out of the toaster.



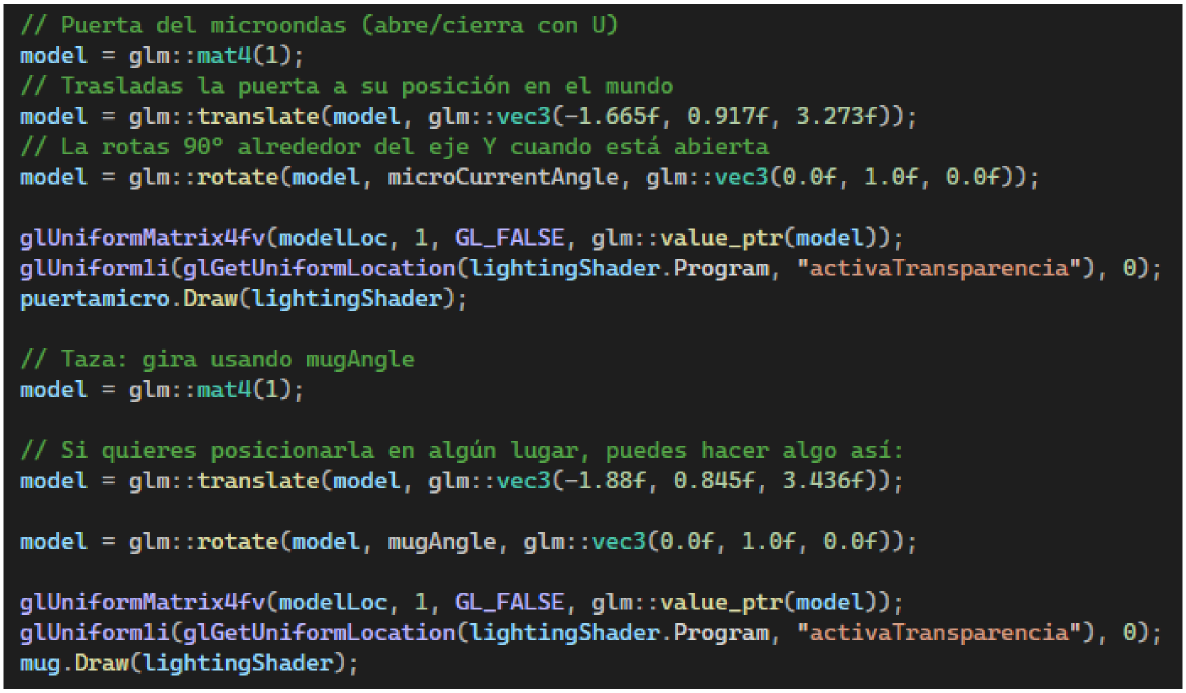
```
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-0.52f, 1.062f, 3.9f));
model = glm::rotate(model, doorCurrentAngle, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
puertaroja.Draw(lightingShader);
```



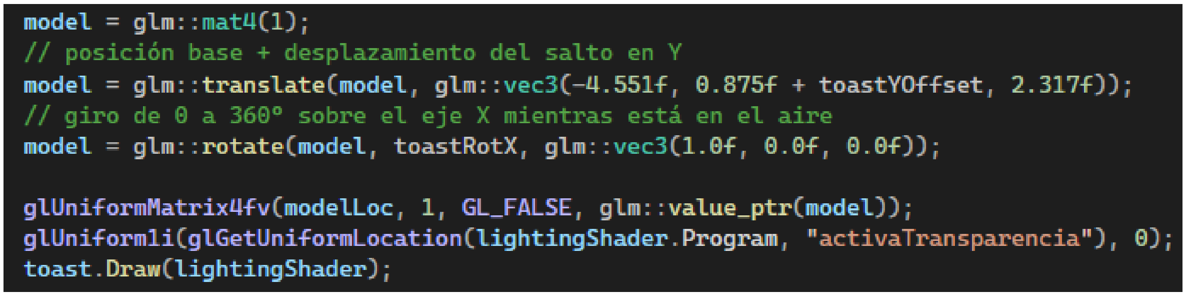
```
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-6.284f, 0.923f, 0.253f));
// movimiento extra de la animación (subir y meter hacia -Z)
model = glm::translate(model, glm::vec3(0.0f, garageCurrentY, garageCurrentZ));
// rotación de la puerta de la cochera en X
model = glm::rotate(model, garageCurrentRotX, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
cochera.Draw(lightingShader);
```



```
glEnable(GL_BLEND);//Avtiva la funcionalidad para trabajar el canal alfa
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0,1.0,0.0,0.75);
cabeza.Draw(lightingShader);
glDisable(GL_BLEND);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0, 1.0, 1.0, 1.0);
glBindVertexArray(0);
```



```
// Puerta del microondas (abre/cierra con U)
model = glm::mat4(1);
// Trasladas la puerta a su posición en el mundo
model = glm::translate(model, glm::vec3(-1.665f, 0.917f, 3.273f));
// La rotas 90° alrededor del eje Y cuando está abierta
model = glm::rotate(model, microCurrentAngle, glm::vec3(0.0f, 1.0f, 0.0f));

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
puertamicro.Draw(lightingShader);

// Taza: gira usando mugAngle
model = glm::mat4(1);

// Si quieres posicionarla en algún lugar, puedes hacer algo así:
model = glm::translate(model, glm::vec3(-1.88f, 0.845f, 3.436f));

model = glm::rotate(model, mugAngle, glm::vec3(0.0f, 1.0f, 0.0f));

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
mug.Draw(lightingShader);
```



```
model = glm::mat4(1);
// posición base + desplazamiento del salto en Y
model = glm::translate(model, glm::vec3(-4.551f, 0.875f + toastYOffset, 2.317f));
// giro de 0 a 360° sobre el eje X mientras está en el aire
model = glm::rotate(model, toastRotX, glm::vec3(1.0f, 0.0f, 0.0f));

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
toast.Draw(lightingShader);
```

- **Rendering of visible lights**

The lamp shader (lampShader) is activated, the view and projection matrices are resent, and a small cube is drawn at each position stored in pointLightPositions.

- **Buffer swapping**

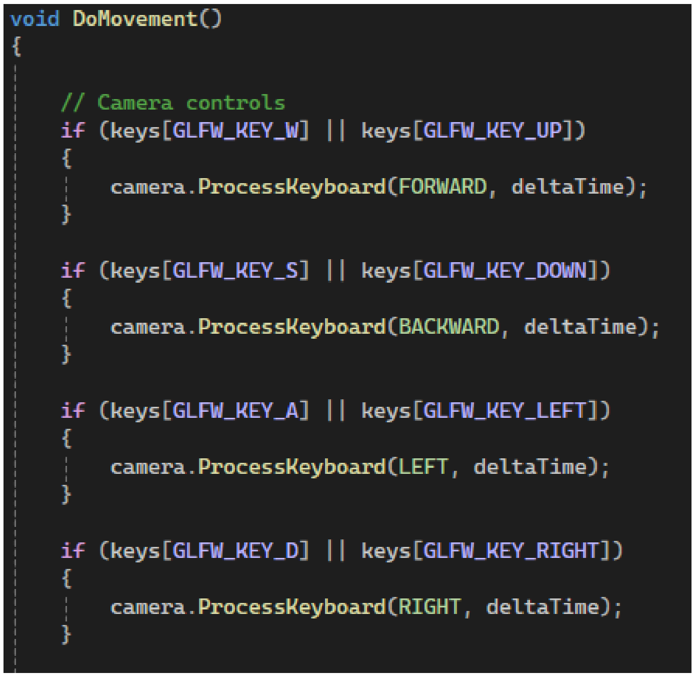
Finally, glfwSwapBuffers(window) is called to display on screen the contents of the newly rendered frame.

Once the loop finishes, the created VAOs/VBOs/EBOs are deleted, and glfwTerminate() is called to release GLFW resources.

- **DoMovement() function**

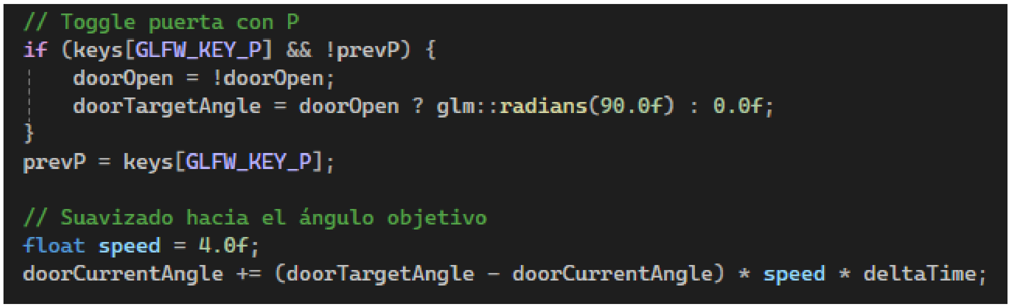
This function is executed in every frame and its responsibility is to translate the keyboard state (previously captured in the callback) into actions on the scene:

1. Camera movement: If the W, A, S, D keys or the arrow keys are active, camera.ProcessKeyboard(...) is called, also passing deltaTime so that the movement speed remains uniform.



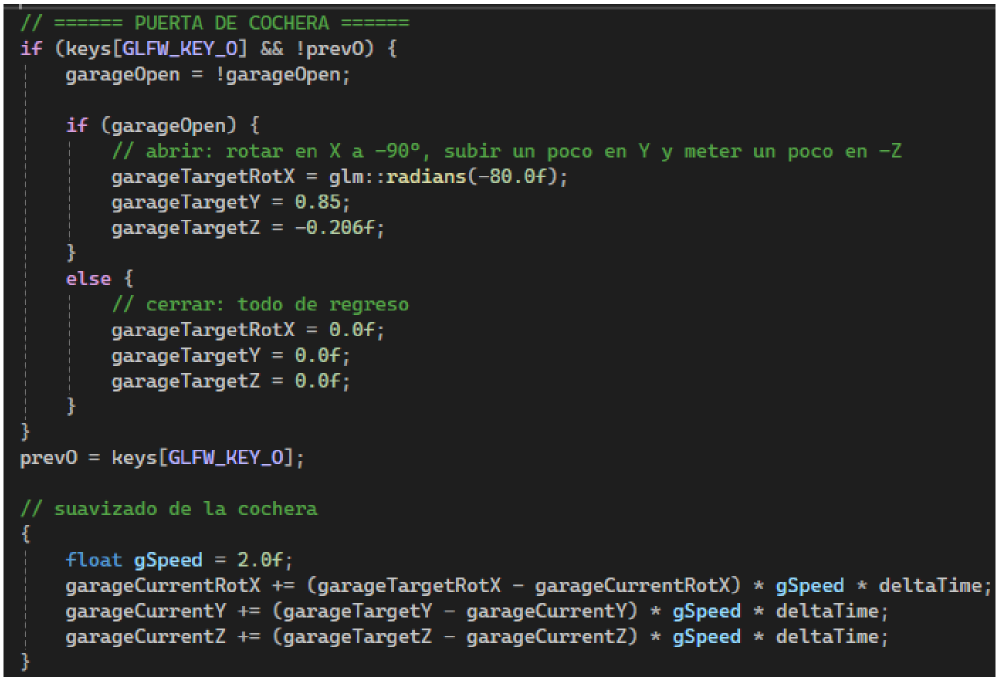
```
void DoMovement()
{

    // Camera controls
    if (keys[GLFW_KEY_W] || keys[GLFW_KEY_UP])
    {
        camera.ProcessKeyboard(FORWARD, deltaTime);
    }

    if (keys[GLFW_KEY_S] || keys[GLFW_KEY_DOWN])
    {
        camera.ProcessKeyboard(BACKWARD, deltaTime);
    }

    if (keys[GLFW_KEY_A] || keys[GLFW_KEY_LEFT])
    {
        camera.ProcessKeyboard(LEFT, deltaTime);
    }

    if (keys[GLFW_KEY_D] || keys[GLFW_KEY_RIGHT])
    {
        camera.ProcessKeyboard(RIGHT, deltaTime);
    }
```

2. Opening/closing the door (key P): the single press is detected (using prevP) and the logical state of the door is toggled. Depending on the state, the target angle is set to 0° or 90°. The current angle is then interpolated toward the target angle using a time-dependent factor, which produces smooth motion.



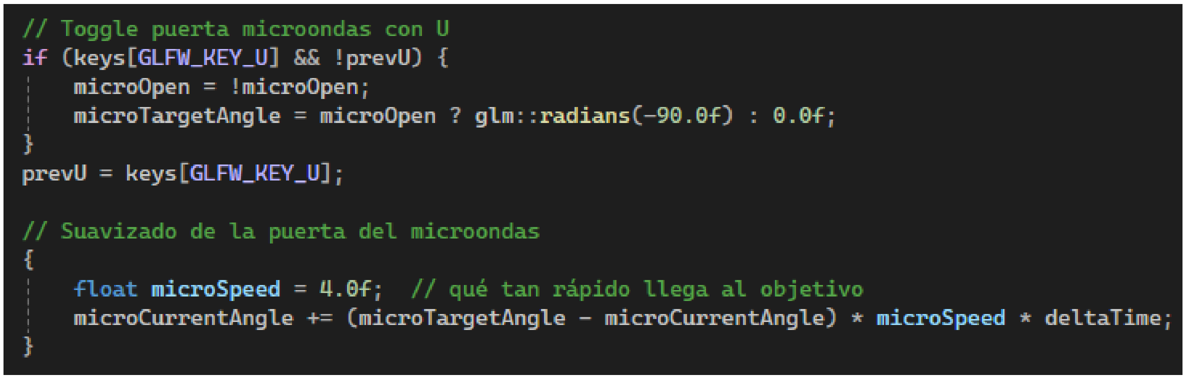
```
// Toggle puerta con P
if (keys[GLFW_KEY_P] && !prevP) {
    doorOpen = !doorOpen;
    doorTargetAngle = doorOpen ? glm::radians(90.0f) : 0.0f;
}
prevP = keys[GLFW_KEY_P];

// Suavizado hacia el ángulo objetivo
float speed = 4.0f;
doorCurrentAngle += (doorTargetAngle - doorCurrentAngle) * speed * deltaTime;
```

3. Opening/closing the garage (key O): the same pattern as for the door is applied, but three values are updated (rotation on X and translations on Y and Z) to simulate a more realistic movement. Here as well, a progressive interpolation toward the target values is performed.



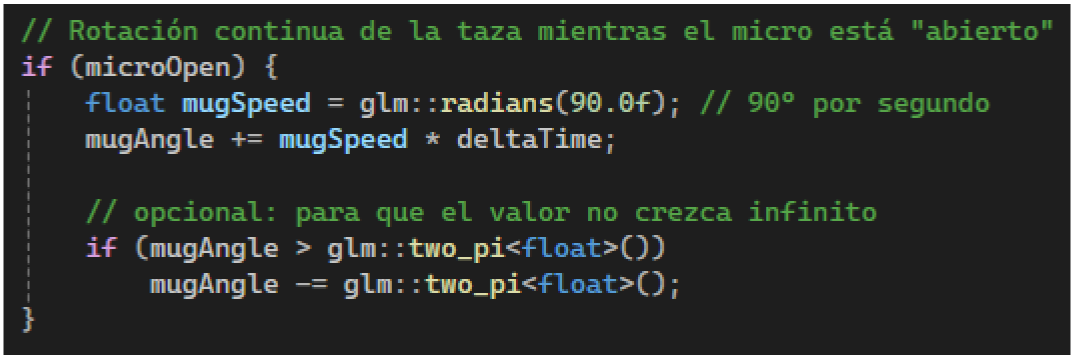
```
// ====== PUERTA DE COCHERA ======
if (keys[GLFW_KEY_O] && !prevO) {
    garageOpen = !garageOpen;

    if (garageOpen) {
        // abrir: rotar en X a -90°, subir un poco en Y y meter un poco en -Z
        garageTargetRotX = glm::radians(-80.0f);
        garageTargetY = 0.85;
        garageTargetZ = -0.206f;
    }
    else {
        // cerrar: todo de regreso
        garageTargetRotX = 0.0f;
        garageTargetY = 0.0f;
        garageTargetZ = 0.0f;
    }
}
prevO = keys[GLFW_KEY_O];

// suavizado de la cochera
{
    float gSpeed = 2.0f;
    garageCurrentRotX += (garageTargetRotX - garageCurrentRotX) * gSpeed * deltaTime;
    garageCurrentY += (garageTargetY - garageCurrentY) * gSpeed * deltaTime;
    garageCurrentZ += (garageTargetZ - garageCurrentZ) * gSpeed * deltaTime;
}
```

4. Opening/closing of the microwave door (key U): The same single-press detection pattern is followed using the variable prevU. When the edge of the U key press is detected, the logical state microOpen is toggled and, based on this state, microTargetAngle is set to 0° (closed) or −90° (open). Analogous to the main door, the angle change is not applied instantaneously; instead, it is interpolated from microCurrentAngle toward microTargetAngle with a smoothing factor dependent on deltaTime, which produces a smooth opening and closing animation around the Y axis.



```
// Toggle puerta microondas con U
if (keys[GLFW_KEY_U] && !prevU) {
    microOpen = !microOpen;
    microTargetAngle = microOpen ? glm::radians(-90.0f) : 0.0f;
}
prevU = keys[GLFW_KEY_U];

// Suavizado de la puerta del microondas
{
    float microSpeed = 4.0f;  // qué tan rápido llega al objetivo
    microCurrentAngle += (microTargetAngle - microCurrentAngle) * microSpeed * deltaTime;
}
```

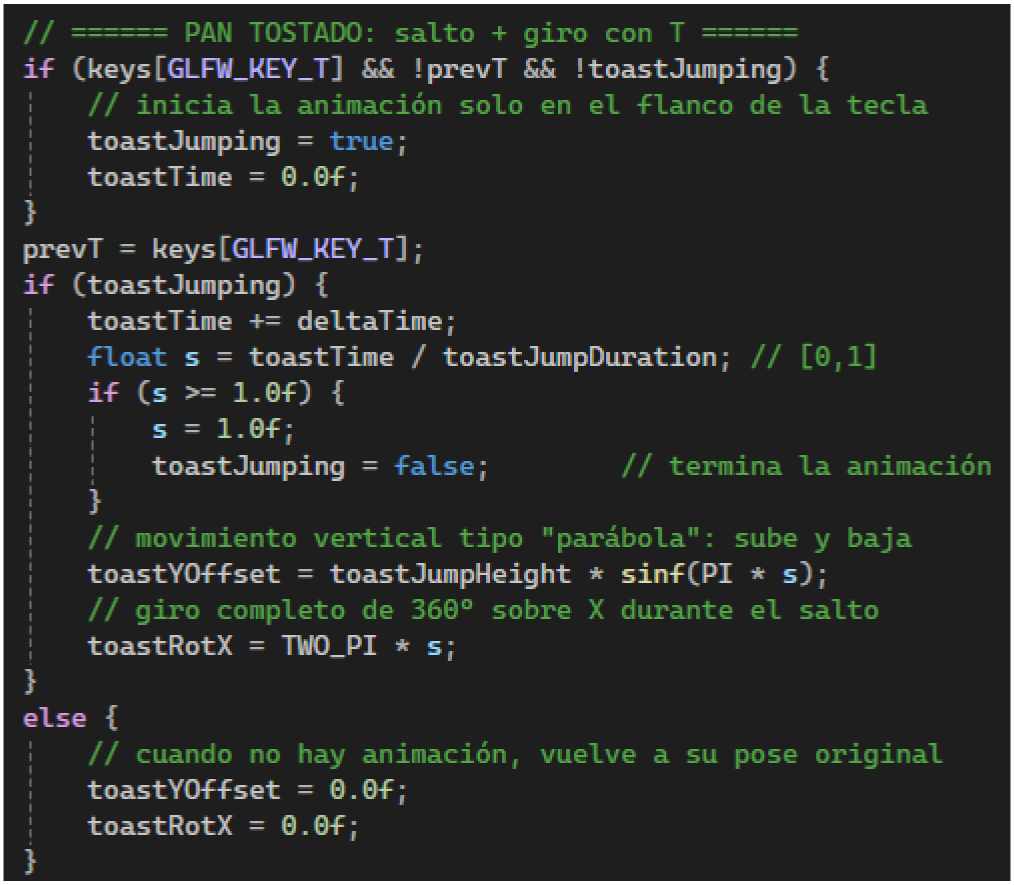
5. Continuous rotation of the mug while the microwave is open: The function also controls the mug animation through the variable mugAngle. Whenever microOpen is true, mugAngle is incremented in each frame using a fixed angular velocity (90°/s expressed in radians) multiplied by deltaTime. In addition, the angle value is normalized by subtracting a full turn (2π) when it exceeds that value, thus avoiding unnecessary numerical accumulation. This updated angle is later used in the transformation of the mug model, generating a continuous rotation while the microwave door remains open.



```
// Rotación continua de la taza mientras el micro está "abierto"
if (microOpen) {
    float mugSpeed = glm::radians(90.0f); // 90º por segundo
    mugAngle += mugSpeed * deltaTime;

    // opcional: para que el valor no crezca infinito
    if (mugAngle > glm::two_pi<float>())
        mugAngle -= glm::two_pi<float>();
}
```

6. Toast animation (key T): jump and spin: For the toast, a discrete animation is implemented that is triggered by pressing the T key. Using the variable prevT, the key edge is detected and, if there is no animation in progress (!toastJumping), the jump is started by resetting the accumulated time toastTime and activating the flag toastJumping. While this flag is active, toastTime is incremented in each frame, a normalized parameter s is computed in the interval [0, 1], and, based on it, toastYOffset (vertical offset) is updated using a sine function to simulate a smooth upward and downward trajectory, and toastRotX is updated to apply a full 360°

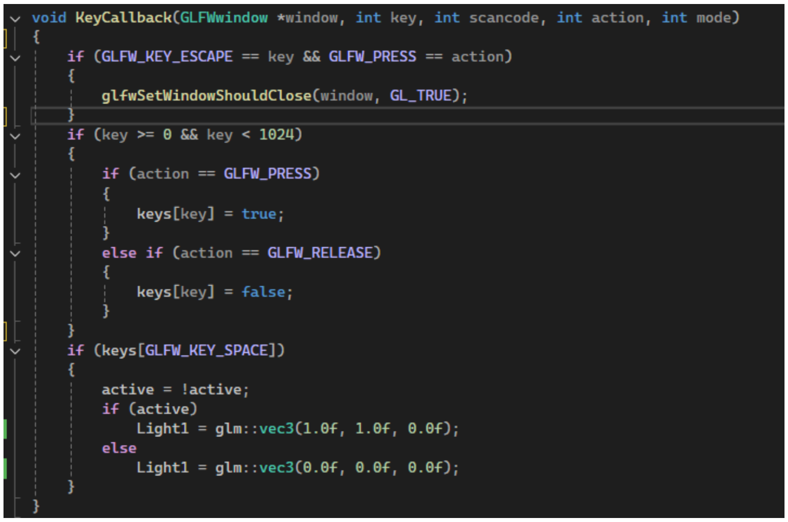
rotation around the X axis during the entire jump. At the end (when s reaches 1), the animation stops, toastJumping is deactivated, and the values are restored to their original pose, ensuring that the toast returns to its resting position.



```
// ====== PAN TOSTADO: salto + giro con T ======
if (keys[GLFW_KEY_T] && !prevT && !toastJumping) {
    // inicia la animación solo en el flanco de la tecla
    toastJumping = true;
    toastTime = 0.0f;
}
prevT = keys[GLFW_KEY_T];
if (toastJumping) {
    toastTime += deltaTime;
    float s = toastTime / toastJumpDuration; // [0,1]
    if (s >= 1.0f) {
        s = 1.0f;
        toastJumping = false;        // termina la animación
    }
    // movimiento vertical tipo "parábola": sube y baja
    toastYOffset = toastJumpHeight * sinf(PI * s);
    // giro completo de 360° sobre X durante el salto
    toastRotX = TWO_PI * s;
}
else {
    // cuando no hay animación, vuelve a su pose original
    toastYOffset = 0.0f;
    toastRotX = 0.0f;
}
```

- **Input callbacks**

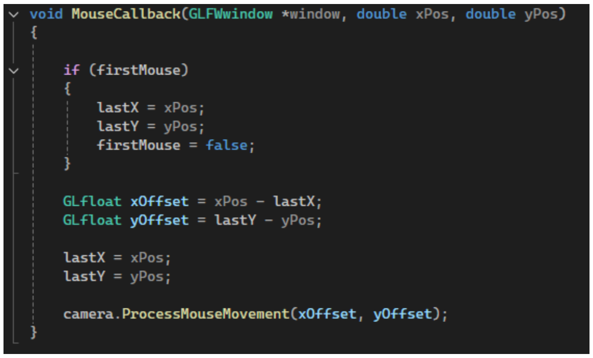
1. KeyCallback(...)
    - If the Escape key is pressed, the window is requested to close.
    - For any other key, the keys[] array is updated with the PRESSED or RELEASED state, which allows DoMovement() to know which keys are active without relying on the individual event.
    - The Space key toggles the state of one of the lights (active and Light1), which makes it possible to enable or disable the color contribution of the first point light.



```
void KeyCallback(GLFWwindow *window, int key, int scancode, int action, int mode)
{
    if (GLFW_KEY_ESCAPE == key && GLFW_PRESS == action)
    {
        glfwSetWindowShouldClose(window, GL_TRUE);
    }
    if (key >= 0 && key < 1024)
    {
        if (action == GLFW_PRESS)
        {
            keys[key] = true;
        }
        else if (action == GLFW_RELEASE)
        {
            keys[key] = false;
        }
    }
    if (keys[GLFW_KEY_SPACE])
    {
        active = !active;
        if (active)
            Light1 = glm::vec3(1.0f, 1.0f, 0.0f);
        else
            Light1 = glm::vec3(0.0f, 0.0f, 0.0f);
    }
}
```

2. MouseCallback(...)

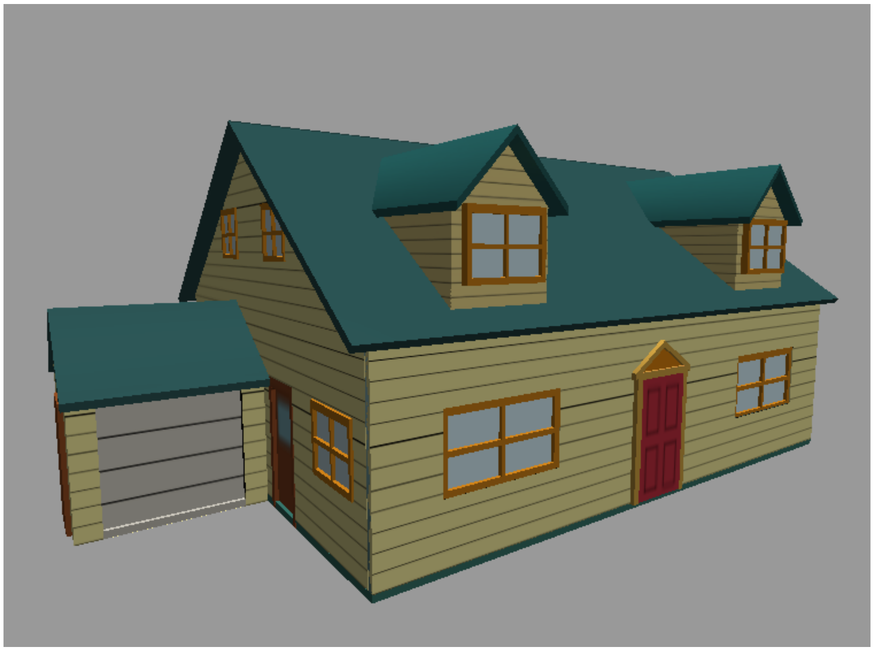
    - On the first call, the initial cursor coordinates are stored.
    - On subsequent calls, the horizontal and vertical mouse offsets are calculated and passed to `camera.ProcessMouseMovement(...)`. In this way, the camera’s orientation is updated.

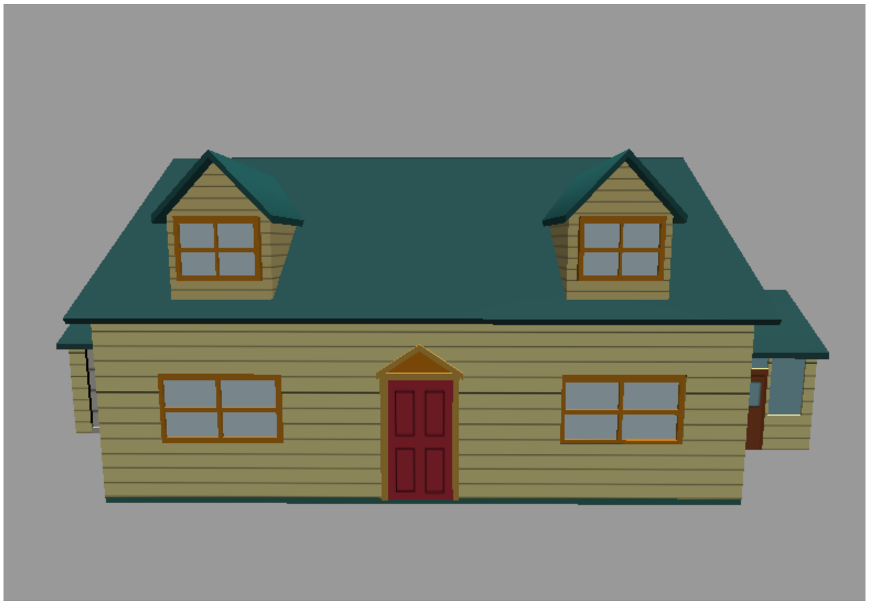


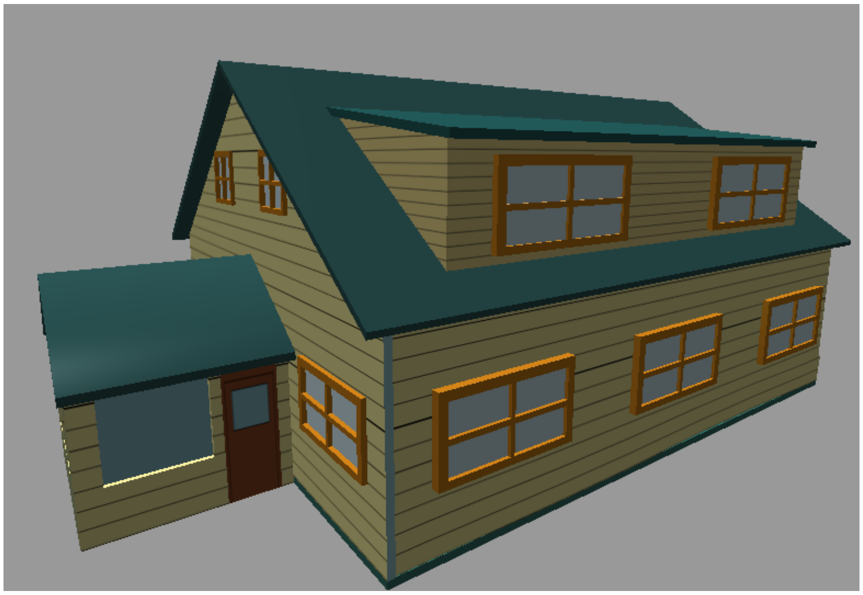
```
void MouseCallback(GLFWwindow *window, double xPos, double yPos)
{

    if (firstMouse)
    {
        lastX = xPos;
        lastY = yPos;
        firstMouse = false;
    }

    GLfloat xOffset = xPos - lastX;
    GLfloat yOffset = lastY - yPos;

    lastX = xPos;
    lastY = yPos;

    camera.ProcessMouseMovement(xOffset, yOffset);
}
```

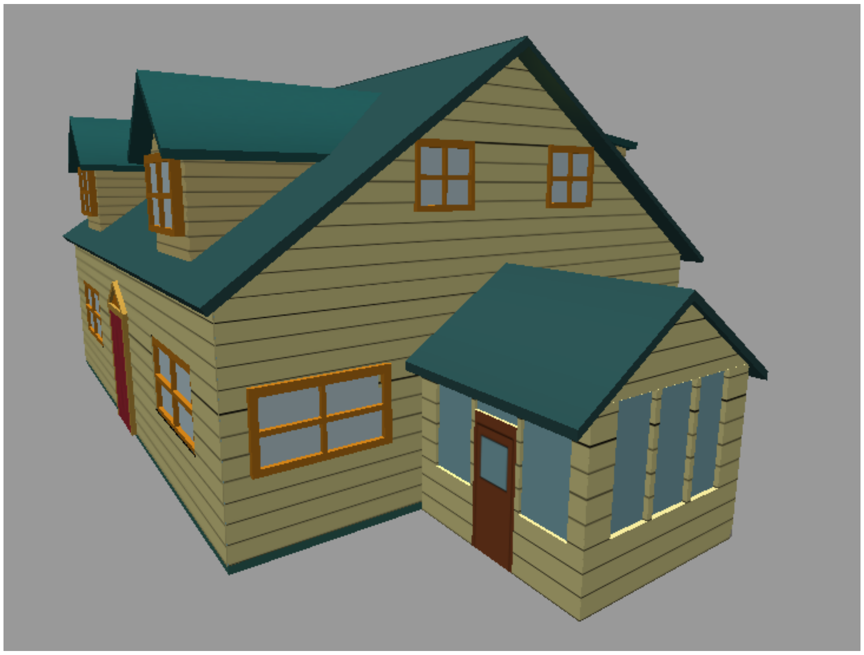
Results

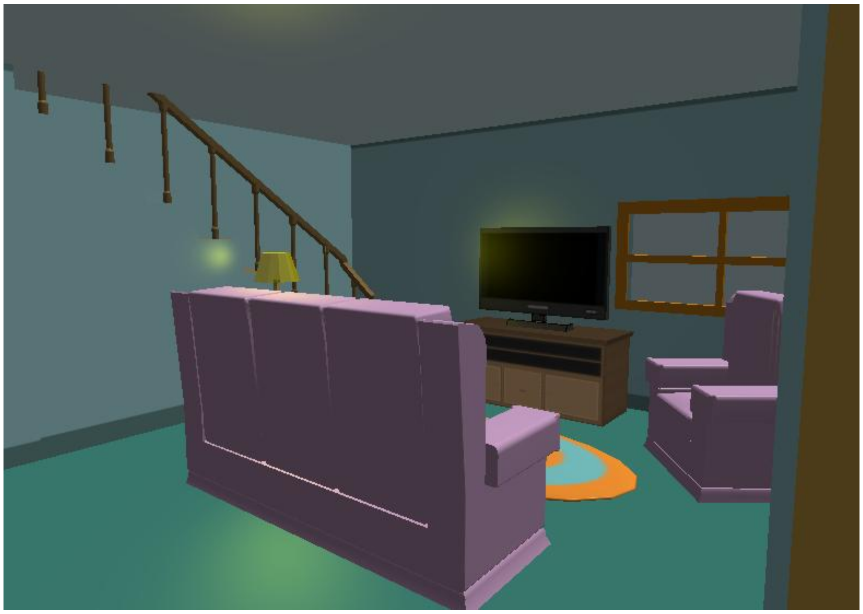
Below, some images of the models loaded in OpenGL will be presented.

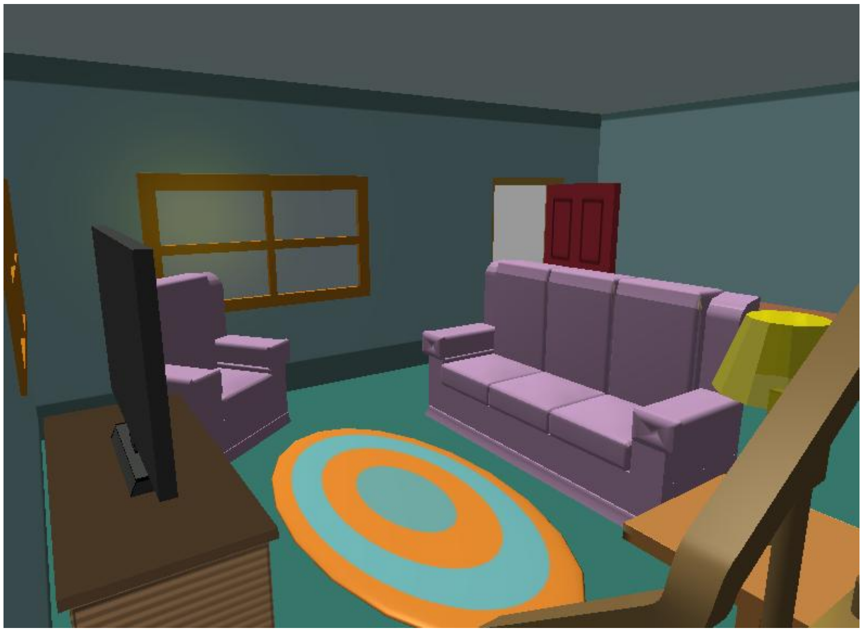


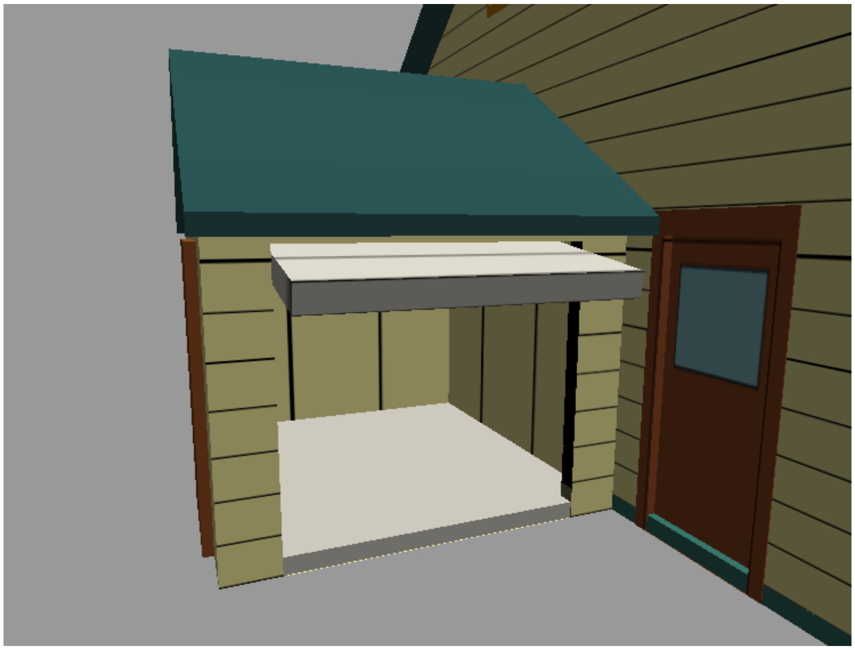


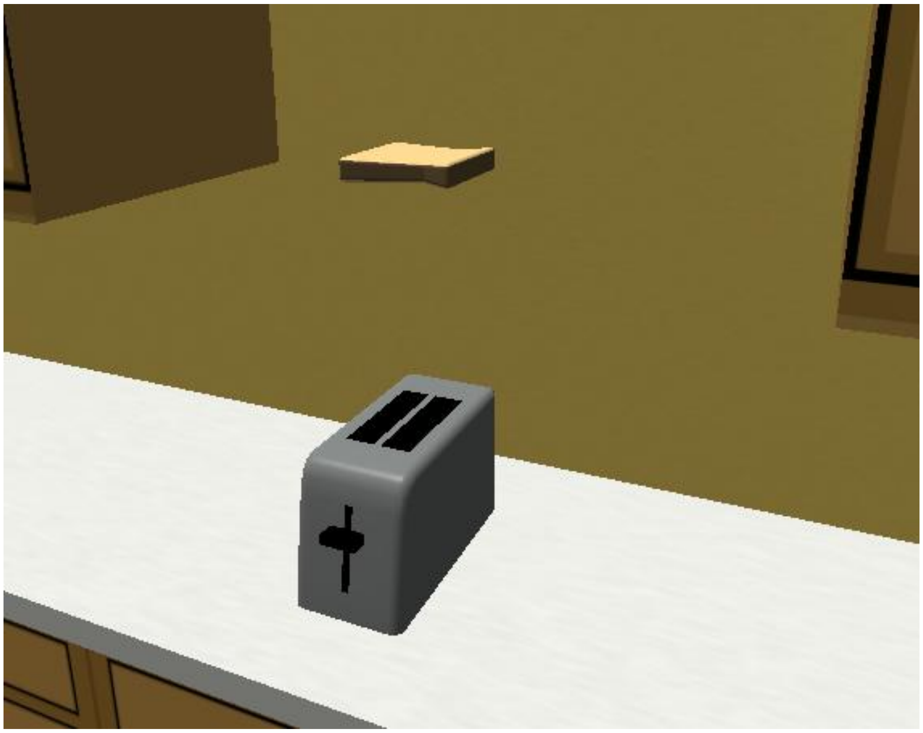


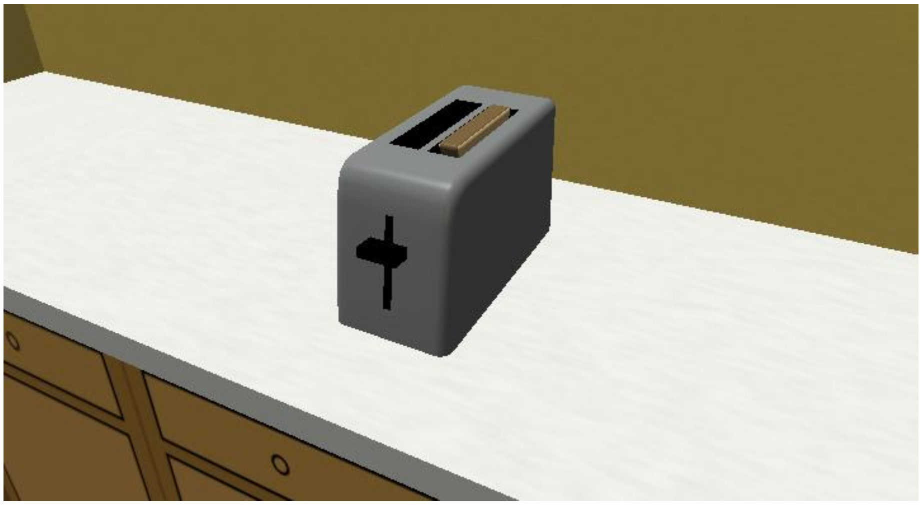












Project Cost Analysis.

- **Direct Costs:**

Resources	Costs	Details
Visual effects and 3D animation software: Autodesk Maya	$3,185.00 MXN (monthly)	The software can be obtained free of charge with a student institutional email account.
Laptop	$18,000.00 MXN	A machine with adequate, good-quality specifications to run the programs and achieve good graphics performance has, on average, this price.
Knowledge	$3,250.00 MXN	The minimum hourly wage is considered and multiplied by the hours devoted to classes and laboratory sessions to acquire the knowledge required to carry out this project.
Time worked	50 hours ($34.00 MXN per hour)	Considering the exact day work on the models and the house began, and the weeks required to develop the project, approximately 50 hours of work were recorded.

Considering that the laptop was not used exclusively for the development of the project, the cost will be calculated only for the 4 months used directly for the course, laboratory, and project development.

On average, the useful life of a laptop is about 4 years:

$$\frac{4\ months}{48\ months}\ x\ \$18{,}000.00\ \text{MXN} = \$1{,}500.00\ \text{MXN}$$

$$50\ x\ \$34.00\ MXN = \$1{,}700.00\ MXN$$

$$\$1{,}700.00\ MXN + \$1{,}500.00\ \mathrm{MXN} +\ \$3{,}250.00\ \mathrm{MXN} = \$6{,}450.00\ \mathrm{MXN}$$

Total: $6,450.00 MXN

- **Indirect Costs:**

Resources	Costs	Details
Internet/electricity	$700.00 MXN	An approximate monthly expense of $700.00 MXN is estimated for the internet service and the electricity consumption required for the continuous use of the laptop during the modeling, rendering, and documentation of the project. The amount considers the proportional share of these services allocated to the development of the project during the period worked (2 months).

$700.00 MXN x 2 = $1,400.00 MXN

Total: $1,400.00 MXN

<u>Total development cost: $7,850.00 MXN</u>

- **Price for the project:**

To determine the selling price of the project, the total development cost was first calculated, considering direct resources (software, computer equipment, working time, and specialized knowledge) and indirect resources (services such as internet and electricity), yielding an amount of $7,850.00 MXN. On this basis, a profit margin of 30% was defined, which represents $2,355.00 MXN, with the aim of compensating the hours invested in modeling, animation, testing, and documentation, as well as recouping the initial investment in infrastructure and services required for the execution of the project. Consequently, the proposed selling price for the project amounts to $10,205.00 MXN (rounded to $10,200.00 MXN), an amount that reflects both the actual production costs and a reasonable profit for the work performed.

Project price: $10,200.00 MXN

- **Justification.**

The determination of the project cost is justified on the basis of a systematic estimate of the resources required for its development. First, the direct costs were identified, which include the computer equipment, the technical knowledge acquired, and the time effectively invested in modeling, animation, and documentation. Although Autodesk Maya is mentioned as the main tool and its monthly commercial cost is indicated, this amount was not incorporated into the financial calculation, since an educational license linked to the student's institutional e-mail account was used for the project. For this reason, the program is recorded as an essential resource but does not represent an actual monetary outlay within the analysis. By contrast, the cost of the laptop is taken into account on an amortized basis, assuming an average useful life of four years and counting only the four months in which it was used directly in the course and in the project, which is equivalent to $1,500.00 MXN. This is added to the value of knowledge (calculated on the basis of the minimum hourly wage and the hours of classes and laboratory sessions required to acquire the necessary competencies) and to the 50 hours of specific work on the project, valued at $34.00 MXN per hour. The sum of these elements results in a direct cost of $6,450.00 MXN, which realistically reflects the essential resources used.

Second, the indirect costs represented by internet and electricity services are included. An average monthly expense of $700.00 MXN is estimated and projected over two months of intensive work, yielding a total of $1,400.00 MXN. This amount corresponds to the proportional share of consumption required to keep the computer equipment in operation during the modeling, rendering, and preparation of the technical documentation. By adding direct and indirect costs, a total development cost of $7,850.00 MXN is obtained, which serves as the basis for setting the final price.

Finally, the selling price of $10,200.00 MXN is justified by applying a profit margin of 30% to the development cost. This margin seeks to compensate for the effort invested in the different stages of the project (planning, modeling, animation, testing, and documentation), as well as to recover the investment in equipment and services used, even though the main software was available at no cost thanks to the student license. Thus, the proposed price is not arbitrary but the result of a structured cost analysis that balances the economic feasibility of the project with fair remuneration for the professional work carried out.

State of the Art of the Project.

Nowadays, virtual tours have become a key tool in various sectors that require the visualization and exploration of three-dimensional spaces without the need for physical presence. In architecture and real estate, for example, they are used to showcase homes, offices, or projects that are still in the design phase, allowing the user to move freely through the environment, change the viewpoint, and observe details of layout, lighting, and furnishings. In museums and cultural heritage, immersive tours are used to make exhibitions or historical sites accessible remotely, while in the entertainment and video game industry, interactive environments are integrated that combine narrative, exploration, and playful elements. The advancement of technologies such as real-time graphics engines, WebGL, and virtual reality devices has driven increasingly realistic tours, with better texture quality, dynamic lighting, and more intuitive navigation.

Within the academic field, virtual tours are used as a didactic laboratory to understand concepts of computer graphics, human–computer interaction, and interface design. When building a navigable environment, the student must apply in an integrated manner 3D modeling, UV mapping, texturing, material configuration, lighting, and camera programming, which turns the project into a practical case of the complete graphics pipeline. The project developed in this manual is part of this trend: the recreation of the façade, living room, and kitchen of the Family Guy house in Maya and their subsequent export to OpenGL is conceived as a user-controlled virtual tour, where it is possible to move through the scene, observe the objects from different angles, and appreciate the behavior of lights and animations. In this way, the logic of professional virtual tours is replicated on a small scale, but with a formative focus that prioritizes the technical understanding of the process and experimentation with the main elements of an interactive 3D environment.

Conclusions

At the beginning of the course, the idea of modeling and texturing seemed extremely difficult and complex to me, but by carrying out this project and understanding and integrating all the stages of the graphics pipeline, I was able to understand and experience each part of the process. Modeling the objects had its own complexity, depending on how perfect one wanted them to be; attending to even the smallest detail made the modeling time quite long. Creating the images from scratch to texture the objects was also a challenge, since I had to find a way for a single image to contain all the colors of the object; for this last part I relied on artificial intelligence to generate the images.

Lighting also took time: determining the component values so that, when the light was reflected on the house and the furniture, they would look as similar as possible to how they looked in Maya was a lengthy process.

In the code part I faced the biggest challenge when exporting the objects, since in some cases OpenGL did not display things the same way they appeared in Maya, so I had to analyze whether the texturing had been done correctly or if I had skipped a step. Without a doubt, patience and perseverance were key in carrying out this project.

In conclusion, this project allowed me to reinforce and learn new knowledge about 3D graphics, as well as organization, planning, and software methodologies to be able to develop projects in a more efficient way.