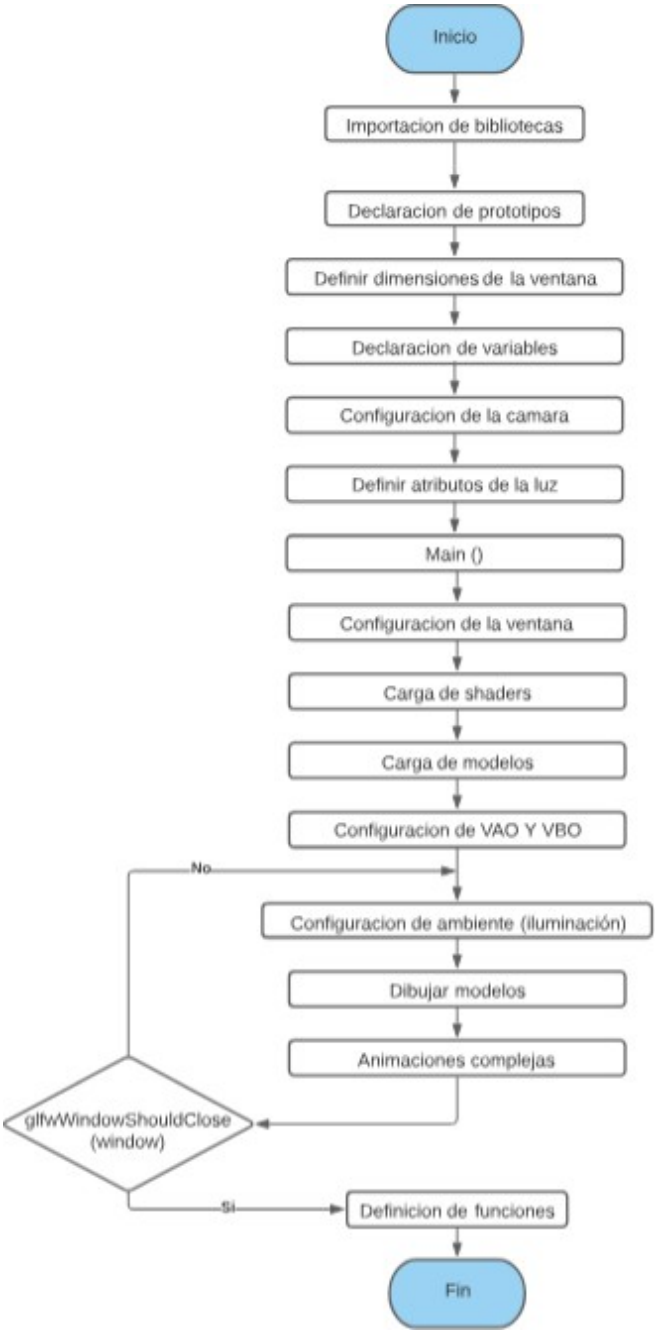
115007310
Group: 5

Technical Manual

Target

The student must apply and demonstrate the knowledge acquired throughout the course.

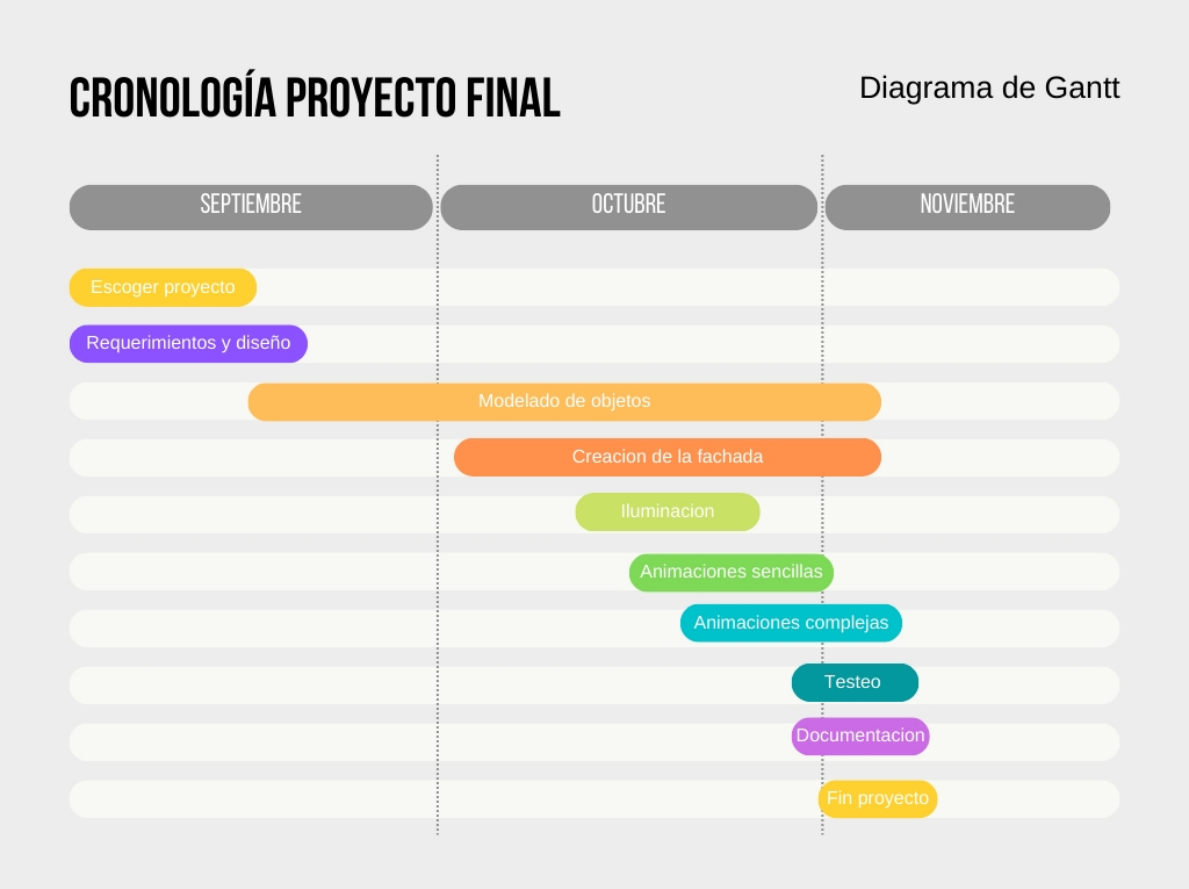
Software flowchart



Software methodology (Waterfall)

- **Requirements**
  - Select the space and the elements to be recreated
  - Technical and user manual
  - Project in the repository with visualization of changes
  - Realism of virtual space vs. reference photo
  - Executable File
  - Model 7 elements already specified in the pdf
  - 3 simple animations and 2 complex animations
  - Correct lighting (ambience)
  - Camera operation (synthetic)

- **Design**
  - Gantt Chart



  - Limitations:
    - I anticipate that hardware equipment performance could be affected when handling multiple models in the program, with possible

compilation delays in OpenGL. To address this issue, I plan to optimize the code.
    - I am aware of the need to comply with the schedule, however, when looking at it I consider that in the end there are several tasks in parallel, which could generate accumulation of work. In this sense, I consider the possibility of omitting the incorporation of the skybox, despite my initial desire to include it.
    - The complexity of the project does not seem to be that much, however, I foresee that there could be challenges in problem solving, which I will surely be able to solve. However, I sense that rather than difficult, the project will be laborious and time-consuming.
    - Adopting a software methodology will be new territory for me because, although I have studied these methodologies in previous subjects, this will be the first time I will be applying them in practice.

- Scope
    - The main objective of this project is the creation of a three-dimensional virtual space that realistically reproduces the elements selected in the reference pdf, plus 7 elements with animations.

- **Implementation**

In this implementation stage, the Gantt chart was followed.

- To model the objects and the facade in Maya, the following steps were followed:
    - I searched for models on sites like turboSquid that looked like the ones I wanted.
    - The models were imported into Maya.
    - Changes were made to the models
    - Textures were added to the objects.
    - When necessary, they adjusted their UV maps

- To transfer the objects and the facade to OpenGL, the following steps were followed:
    - Objects were added to the folder "models".
    - An object of the model class is instantiated and assigned a name together with the path of the object.
    - The object was drawn
    - If necessary, the object was rotated or moved.

- The simple animations were as follows:
  - Opening and closing a cabinet door (kitchen unit)
  - Moving the arm of a dummy
  - Rotating ceiling fan propellers

- The complex animations were as follows:
  - A blender turned on making it shake a little sideways
  - Bread coming out of a toaster

- Adequate lighting was added:
  - 4 pointlights located in a lamp
  - Spotlight
  - Dirlight

- Camera operation
  - It is a synthetic camera, as it was by default

I was uploading everything mentioned above little by little to my GitHub repository. The most significant development time was this part of the implementation, especially for the realization of the facade and the models.

- **Tests**

No unit testing, integration testing, system testing, user acceptance testing or any of those, what I did was to do what is called **"observational testing"** or **"inspection testing"**, which is only a testing approach that is based on the direct visual review of a system to identify potential bugs, defects or areas for improvement. It's not the most effective type of testing, but it was what suited my needs to see what the space looked like and thus improve the project.

- **Delivery**

Once all the above is done, the last commit to GitHub is done and the project is delivered taking into account all the requirements.

Code documentation

- **Import of libraries**
  Includes libraries for window management (GLFW), OpenGL extensions (GLEW), mathematics (GLM), image loading (stb_image), and textures.

(SOIL2). Classes are also imported to manage shaders, cameras and models in the application.

```
#include <iostream>
#include <cmath>

// 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"
```

- **Declaration of prototypes**
  They are for functions related to user interaction in the OpenGL window, "KeyCallback" handles keyboard events, "MouseCallback" handles mouse events, and "DoMovement" performs actions associated with user movement in the 3D scene.

```
// Function prototypes
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode);
void MouseCallback(GLFWwindow* window, double xPos, double yPos);
void DoMovement();
```

- **Window dimensions**
  They define constants for the width and height of the window (WIDTH and HEIGHT) with values 800 and 600, respectively. In addition, variables (SCREEN_WIDTH and SCREEN_HEIGHT) are declared to store the actual dimensions of the screen.

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

- **Declaration of variables for simple and complex animations** Here we declare the variables that will be used to create simple and complex animations.

the animation, the ones starting with *"rot"* control the rotation of the object. Everything is initialized to 0 or false.

```
36  //Variables para animacion (sencilla)
37  //Gabinete
38  float rotPuertaGab = 0.0f;
39  bool animPuertaGab = false;
40  bool animPuertaGab2 = true;
41  //Maniqui
42  float rotBrazoManiqui = 0.0f;
43  bool animBrazoManiqui = false;
44  bool animBrazoManiqui2 = true;
45  //Ventilador
46  float rotVentilador = 0.0f;
47  bool animVentilador = false;
48
49  // Variables para animacion (compleja)
50  // Licuadora
51  float tiempo;
52  float speed;
53
54  //Tostadora
55  float tiempo2;
56  float speed2;
```

- **Camera setup and operation**
  A starting position is assigned to the camera, "lastX" and "lastY" store the last cursor position on their respective axis.

```
59      // Camera
60      Camera  camera(glm::vec3(0.0f, 0.0f, 3.0f));
61      GLfloat lastX = WIDTH / 2.0;
62      GLfloat lastY = HEIGHT / 2.0;
63      bool keys[1024];
64      bool firstMouse = true;
```

- **Light attributes and positions**
  The 4 pointlights used are positioned and placed in a lamp.

```
66      // Light attributes
67      glm::vec3 lightPos(0.0f, 0.0f, 0.0f);
68      bool active;
69
70      // Positions of the point lights
71      glm::vec3 pointLightPositions[] = {
72          glm::vec3(0.27f, 4.0f, 3.16f),
73          glm::vec3(0.27f, 4.0f, 3.95f),
74          glm::vec3(-0.1f, 4.0f, 3.55f),
75          glm::vec3(0.7f, 4.0f, 3.55f)
76      };
```

- **Vertices**

A set of vertices is defined to define the geometry of a 3D cube.

```
78    float vertices[] = {
79        -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
80           0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
81           0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
82           0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
83          -0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
84          -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
```

- **Processor timing variables**

"deltaTime" represents the time between the current frame and the last frame, while *"lastFrame"* stores the time of the last frame.

```
127    // Deltatime
128    GLfloat deltaTime = 0.0f;   // Time between current frame and last frame
129    GLfloat lastFrame = 0.0f;   // Time of last frame
```

- **Main function**
  - **OpenGL configuration**
    The window is created, GLEW and GLFW are configured. Additionally define the dimensions of the viewport according to the size of the created window.

```
131  int main()
132  {
133      // Init GLFW
134      glfwInit();
135      // Set all the required options for GLFW
136      /*glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
137      glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
138      glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
139      glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
140      glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);*/
141
142      // Create a GLFWwindow object that we can use for GLFW's functions
143      GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "Proyecto final - 115007310", nullptr, nullptr);
144
145      if (nullptr == window)
146      {
147          std::cout << "Failed to create GLFW window" << std::endl;
148          glfwTerminate();
149
150          return EXIT_FAILURE;
151      }
152
153      glfwMakeContextCurrent(window);
154
155      glfwGetFramebufferSize(window, &SCREEN_WIDTH, &SCREEN_HEIGHT);
```

  - **Shader loading**
    The shaders for illumination and lamp are loaded.
    Also the shaders used for the toaster and blender animations.

```
178     Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting.frag");
179     Shader lampShader("Shaders/lamp.vs", "Shaders/lamp.frag");
180
181     Shader AnimLicuadora("Shaders/animLicuadora.vs", "Shaders/animLicuadora.frag");
182     Shader AnimTostadora("Shaders/animTostadora.vs", "Shaders/animTostadora.frag");
```

- **Loading of models**

  The models to be subsequently drawn are loaded, the path of the folder where they are located must be well specified.

```
Model Fachada((char*)"Models/Fachada/Fachada.obj");
Model Ventanas((char*)"Models/Fachada/Ventanas.obj");
Model table((char*)"Models/Mesa/mesa.obj");
Model cama((char*)"Models/Cama/cama.obj");
Model Gabinete((char*)"Models/Gabinete/Gabinete.obj");
Model puertaGabinete((char*)"Models/Gabinete/puertaGabinete.obj");
Model Silla((char*)"Models/Silla/Silla.obj");
Model Maniqui((char*)"Models/Maniqui/Maniqui.obj");
Model BrazoManiqui((char*)"Models/Maniqui/BrazoManiqui.obj");
Model Planta((char*)"Models/Planta/planta.obj");
Model Lampara((char*)"Models/Lampara/Lampara.obj");
Model VentiladorSoporte((char*)"Models/Ventilador/VentiladorSoporte.obj");
Model VentiladorHelices((char*)"Models/Ventilador/VentiladorHelices.obj");
Model LicuadoraArriba((char*)"Models/Licuadora/licuadoraArriba.obj");
Model LicuadoraAbajo((char*)"Models/Licuadora/licuadoraAbajo.obj");
Model Tostadora((char*)"Models/Tostadora/Tostadora.obj");
Model Pan((char*)"Models/Tostadora/Pan.obj");
Model Sillon((char*)"Models/Sillon/sillon.obj");
Model Cuadro((char*)"Models/Cuadro/Cuadro.obj");
Model Refrigerador((char*)"Models/Refrigerador/Refrigerador.obj");
Model Microondas((char*)"Models/Microondas/Microondas.obj");
Model Television((char*)"Models/Television/Television.obj");
```

- **VAO and VBO configuration**

  Sets the VAO (Vertex Array Object) and the VBO (Vertex Buffer Object). Defines position and normal attributes for the vertices and enables them in the VAO.

```
206     // First, set the container's VAO (and VBO)
207     GLuint VBO, VAO;
208     glGenVertexArrays(1, &VAO);
209     glGenBuffers(1, &VBO);
210     glBindVertexArray(VAO);
211     glBindBuffer(GL_ARRAY_BUFFER, VBO);
212     glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
213     // Position attribute
214     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), (GLvoid*)0);
215     glEnableVertexAttribArray(0);
216     // normal attribute
217     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 * sizeof(float)));
218     glEnableVertexAttribArray(1);
219
220     // Set texture units
221     lightingShader.Use();
```

- **Main loop**
  - **Environment configuration (lighting)**

Using the lightning shader a directional light, 4 pointlights and a spotlight are created.

```
// Directional light (Solo debe haber una)
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.4f, 0.4f, 0.4f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.4f, 0.4f, 0.4f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.5f, 0.5f, 0.5f); //Intensidad


// Point light 1
glm::vec3 lightColor;
lightColor.x = abs(sin(glfwGetTime() * Light1.x));
lightColor.y = abs(sin(glfwGetTime() * Light1.y));
lightColor.z = sin(glfwGetTime() * Light1.z);


glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"), pointLightPositions[0].x, poi
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"), lightColor.x, lightColor.y, li
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), lightColor.x, lightColor.y, li
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"), 0.2f, 0.2f, 0.2f);
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);
```

...

```
// Point light 4
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].position"), pointLightPositions[3].x, point
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].ambient"), 0.0f, 0.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].diffuse"), 0.0f, 0.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].specular"), 0.2f, 0.2f, 0.2f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].linear"), 0.09f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].quadratic"), 0.032f);

// SpotLight
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.position"), camera.GetPosition().x, camera.GetPo
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.direction"), camera.GetFront().x, camera.GetFron
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.ambient"), 1.0f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.diffuse"), 1.0f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.specular"), 1.0f, 1.0f, 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.linear"), 0.32f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.quadratic"), 0.45f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.cutOff"), glm::cos(glm::radians(12.0f)));
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.outerCutOff"), glm::cos(glm::radians(15.0f)));

// Set material properties
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
```

- **Drawing from models**

  In this section all the models to be used are loaded, which are the facade, bed, cabinet, table, chair, mannequin, plant, lamp, fan, blender base and toaster.

  All of these objects mentioned above were loaded first because they are solid objects (no transparency).

```
347         //Cargas de modelos
348
349         //---------------------------Fachada---------------------------//
350         model = glm::mat4(1);
351         glUniformMatrix4fv(glGetUniformLocation(lightingShader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
352         Fachada.Draw(lightingShader);
353
354         //---------------------------Cama---------------------------//
355         view = camera.GetViewMatrix();
356         model = glm::mat4(1);
357         model = glm::translate(model, glm::vec3(2.7f, 4.5f, 3.8f));
358         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
359         //glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
360         cama.Draw(lightingShader);
361
362         //---------------------------Gabinete---------------------------//
363         model = glm::mat4(1);
364         glUniformMatrix4fv(glGetUniformLocation(lightingShader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
365         Gabinete.Draw(lightingShader);
366
367         model = glm::mat4(1);
368         model = glm::translate(model, glm::vec3(6.99f, 3.23f, -8.30f));
369         model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
370         model = glm::rotate(model, glm::radians(rotPuertaGab), glm::vec3(0.0f, 1.0f, 0.0f)); //Animacion
371         glUniformMatrix4fv(glGetUniformLocation(lightingShader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
372         puertaGabinete.Draw(lightingShader);
```

Then the alpha channel is activated to load transparent or semi-transparent models, which in this case were the windows of the house. Finally the alpha channel is deactivated so as not to affect what follows in the code.

```
//solido -> transparente -> semitransparente
model = glm::mat4(1);
glEnable(GL_BLEND);//Activa la funcionalidad para trabajar el canal alfa
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0f, 1.0f, 1.0f, 0.50f);
Ventanas.Draw(lightingShader);
glDisable(GL_BLEND);  //Desactiva el canal alfa
glBindVertexArray(0);
```

- **Complex animation**

The complex animations of the blender and the toaster are drawn here because they use shaders.

```
473     ////----------------------------Animacion compleja Licudora----------------------------//
474     speed = 10;
475     speed += 0.1f;
476     tiempo = glfwGetTime() * speed;
477
478     AnimLicuadora.Use();
479     modelLoc = glGetUniformLocation(AnimLicuadora.Program, "model");
480     viewLoc = glGetUniformLocation(AnimLicuadora.Program, "view");
481     projLoc = glGetUniformLocation(AnimLicuadora.Program, "projection");
482
483     glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
484     glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
485     glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
486     model = glm::mat4(1);
487     glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
488     glUniform1f(glGetUniformLocation(AnimLicuadora.Program, "time"), tiempo);
489     LicuadoraArriba.Draw(lampShader);
490
491     glBindVertexArray(0);
492
493     ////----------------------------Animacion compleja Tostadora----------------------------//
494
495     tiempo2 = glfwGetTime() * speed;
496
497     AnimTostadora.Use();
```

animLicuadora.vs

To perform it, an amplitude, distance, frequency and time were defined, which multiply the sine function for the blender to oscillate.
In addition, I defined a side effect which multiplies the aforementioned effect by the attenuation, this side effect was applied to X and Z.

```
1   #version 330 core
2   layout (location = 0) in vec3 aPos;
3   layout (location = 1) in vec3 aNormal;
4   layout (location = 2) in vec2 aTexCoords;
5
6   const float amplitude = 0.01;
7   const float frequency = 1.0;
8   const float lateralAttenuation = 0.2;
9   const float PI = 3.14159;
10
11  out vec2 TexCoords;
12
13  uniform mat4 model;
14  uniform mat4 view;
15  uniform mat4 projection;
16  uniform float time;
17
18  void main()
19  {
20    float distance = length(aPos);
21    float effect = amplitude * sin(PI * distance * frequency + time);
22    float lateralEffect = effect * lateralAttenuation;
23    gl_Position = projection * view * model * vec4(aPos.x + lateralEffect, aPos.y, aPos.z + lateralEffect, 1);
24    TexCoords = vec2(aTexCoords.x + lateralEffect, aTexCoords.y + lateralEffect);
25  }
```

animTostadora.vs

To make the bread come out of the blender it takes a cosine, this is multiplied by amplitude and frequency which causes the following: Higher Frequency: The bread moves faster.
Lower Frequency: The bread moves slower.

Greater Amplitude: Bread rises and falls more, it is as if the bread flies higher.
Lower Amplitude: More subtle movement.

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoords;

const float amplitude = 0.03;
const float frequency = 0.15;
const float PI = 3.14159;
out vec2 TexCoords;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;
uniform float time;

void main()
{
  float effect = amplitude * cos(PI * frequency * time);

  gl_Position = projection * view * model * vec4(aPos.x, aPos.y + effect, aPos.z, 1);
  TexCoords = vec2(aTexCoords.x, aTexCoords.y);
}
```

- **Keyboard functions**
  "KeyCallback" handles keyboard events, *"MouseCallback"* handles mouse events, and *"DoMovement"* performs actions associated with user movement in the 3D scene. All of them are of type void.

```
527     // Moves/alters the camera positions based on user input
528     void DoMovement()
```

```
636     // Is called whenever a key is pressed/released via GLFW
637     void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode)
```

```
676     void MouseCallback(GLFWwindow* window, double xPos, double yPos)
677     {
```

Dictionary of functions

Function	Variables	Description
void DoMovement	camera pointLightflositions animfluertaGab animfluertaGab2 rotfluertaGab animBrazoManiqui animBrazoManiqui2 rotBrazzoManiq ui animFan rotVentilator rotVentilator	This function is constantly activated when a key is pressed because it is inside the *while* and will not be canceled until the program is closed. • camera is used to move the camera by pressing the W,A,S,D keys or the arrow keys. • pointLightflositions was used to move the pointlights in a simpler way because you just print the position and then you can arrange them in the pointlight array, you move them with the keys T,G,Y,H ,U,J • If the animation of the cabinet door is activated (animfluertaGab en

		true), the rotfluertaGab variable is set to perform a rotating animation on the cabinet door. • If the animation of the mannequin arm is enabled (animArmMannequin is true), the rotBrazoMannequin variable is set to perform a rotation animation on the mannequin arm. • If fan animation is enabled (animVentilador is true), the rotVentilador variable is set to perform a rotation animation on the fan.
void KeyCallback	animfluertaGab animBrazzoManiqu i animVentilator	This function is activated when a key is pressed or released. • Check if the escape key is pressed, if it is pressed, exit the program. • Pressing the fl key activates the cabinet door animation. • Pressing the 0 key activates the animation of the dummy arm • Pressing the L key activates the fan propeller animation.
void MouseCallback	firstMouse lastX lastY firstMouse camera	This function handles the mouse movement events in the window • firstMouse: Controls whether it is the first mouse movement to initialize the positions. • lastX and lastY: Store the last X and Y coordinates of the mouse to calculate the difference. • xflos and yflos: Represent the current mouse coordinates in the GLFW window. • Camera: Represents the camera; used to adjust the orientation in response to camera movement. mouse.

int main	keys[1024], animfluertaGab, animBrazoManiqui, animVentilador, rotfluertaGab, rotBrazoManiqui, rotVentilador, pointLightflositions[4], deltaTime, lastFrame, lastX, lastY, firstMouse, WIDT , EIGHHT, SCREE _WIDTNH, SCREE _ EIGNHHT, Shader lightingShader, Shader lampShader, Shader AnimLicuadora, Shader AnimTostadora, Model Fachada, Model Ventanas, Model Sillon, GLuint VBO, VAO, glm::mat4 projection, speed, time, time2	It is the main of the code, so this is the most **important** part **of the code**, here the window is created, the 3D models are loaded, the shaders for lighting and animations are defined, it sets the loop(while) where all the events are managed like updating the scene and rendering the objects imported from Maya and part of the animations are implemented. In general everything is important in the program, but this is the most important part because it is in the main. • keys[1024]: Key status. • animfluertaGab, animBrazoManiqui, animVentilador: Animation triggers. • rotfluertaGab, rotBrazoManiqui, rotVentilador: Rotation angles. • pointLightflositions[4]: pointLightflositions[4]: light flositions. • deltaTime: Time difference. • lastFrame: Time of the last frame. • lastX, lastY: Coordinates of the last mouse movement. • firstMouse: first mouse movement. • WIDT , EIGHHT: Window dimensions. • SCREE _WIDTNH, SCREE _ EIGNHHT: Width and height of the display window. • Shader lightingShader, Shader lampShader, Shader AnimLicuadora, Shader AnimTostadora: Shaders. • Model Facade, Model Windows, …, Model Armchair: 3D Models. • GLuint VBO, VAO: Buffer identifiers and vertex array. • glm::mat4 projection: Projection matrix. • speed: Animation speed.

		• time, time2 : Time multiplied by speed in animations.

Cost analysis

Variable cost	Fixed cost
Maya License	Internet
	Water
	Transportation
	Income
	% computer
	Models
	Programmer

Maya License
Monthly cost: $2,797
Total cost for 3 months: $2,797 x 3 = $8,391

Internet
Monthly cost: $500
Total cost for 3 months: $500 x 3 = $1,500

Water
Monthly cost: $587.13
Total cost for 3 months: $587.13 x 3 = $1,761.39

Transportation
No travel was done, everything was done from home.
Cost= $0

Income
No rent is paid, but among the other expenses of the house an approximate of $4,000 was considered.

Computer
Computer cost: $18,000 Estimated
useful life: 5 years

Annual Depreciation:
Annual depreciation = Cost of the computer / Useful life
Annual depreciation = $18,000 / 5 = $3,600 per year

Depreciation for the Duration of the Project (3 months):
Depreciation for the project = Annual depreciation * (Duration of the project in months / 12)
Depreciation for the project = $3,600 * (3 / 12) = $900

Models
Cost for each model: $500
Cost of 15 models: $500 x 15 = $7500

OpenGL programming
Cost per hour as programmer: $600 Hours
invested in programming: 20 Cost: $600 x 25:
$12000

Total project cost:

$8,391+$1,500+$1,761.39+$4,000+$900+$7500+$12000= $36,052.39

Conclusions

Making this project was not an easy task, even though I did not model the objects from scratch it was a little difficult to modify them to fit my project. It was not easy to texture either since my facade was not a single solid color and I had to apply several textures on the same face, especially the wood models were the difficult ones to texture, but I learned to use the different types of projections to texture correctly.

For this project we put into practice everything we learned in class throughout the semester (modeling, animations, lighting, etc.) and I think that even though it was not as perfect as I wanted, I managed to show that I did learn during the semester. I also liked that we had to be self-taught in the use of the modeling software (Maya) and the photo editor (gimp) which is a very important skill in our environment as computer engineers. The only thing I didn't like as much as I liked was the complex animations.

Something that I also liked was the fact that we also had to use a software methodology, which are widely used in the workplace to maintain order and documentation of a project. I had only seen them in class, but I had never used one to do a project of my own until now and I realize their importance.

In spite of all the long process and dedication I had for the whole project, I enjoyed the process because it is a very different project than the ones that are usually left in the other subjects. When you see the result it is great because you realize how much you learned to do in 11 weeks during the semester.

In addition to all the above I found it very interesting to do a cost analysis since it is the first time I do one and I think it is very useful because if at some point I have a FreeLancer job I already have an idea of how much I could charge for a project.