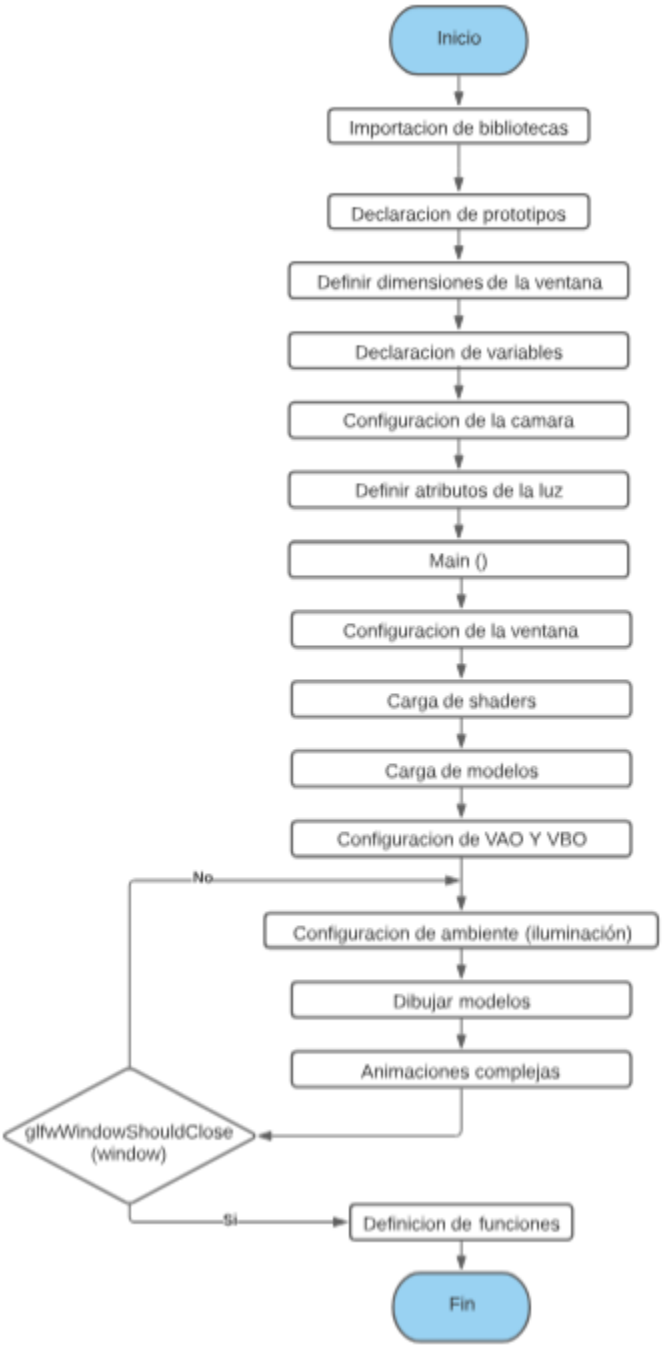
115007310
Grupo: 5

Manual técnico

Objetivo

El alumno deberá aplicar y demostrar los conocimientos adquiridos durante todo el curso.

Diagrama de flujo de software

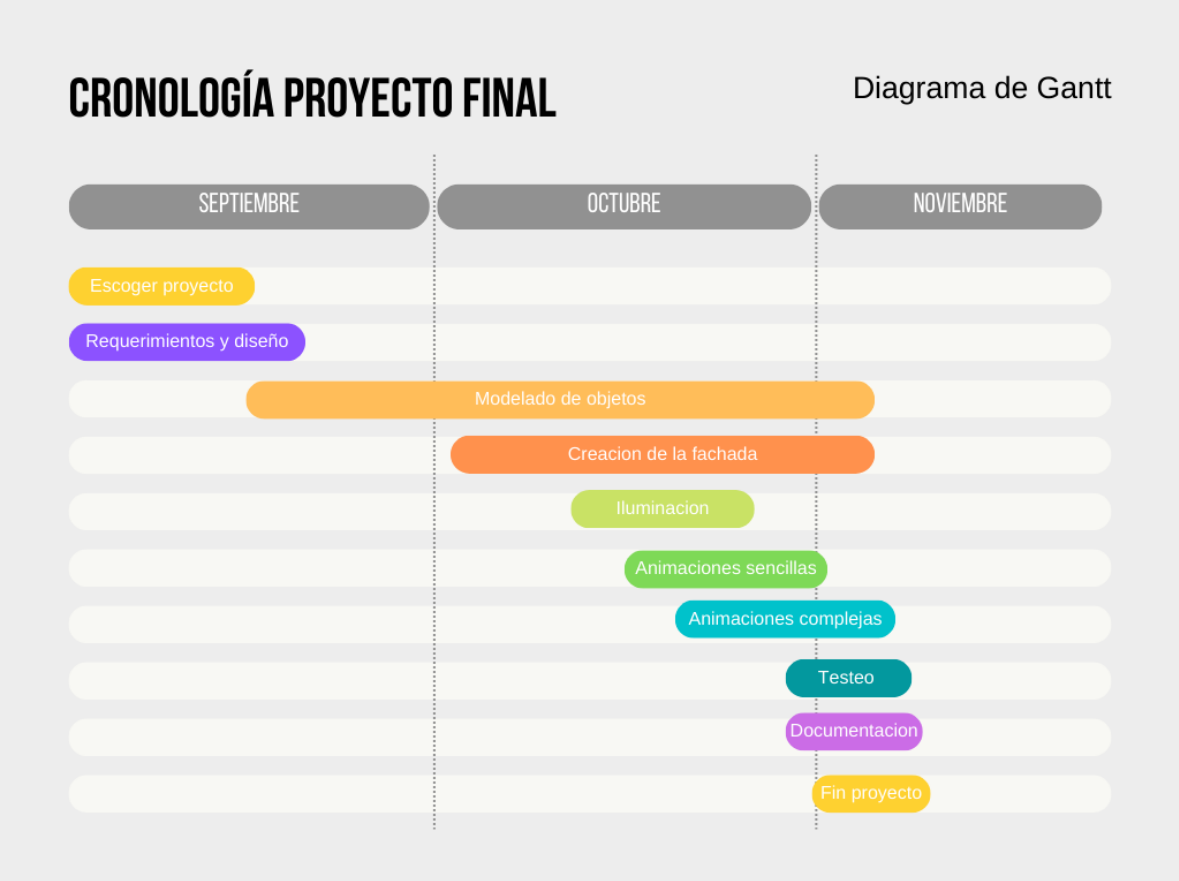


115007310
Grupo: 5

Metodología de software (Cascada)

- **Requisitos**
  - Seleccionar el espacio y los elementos a recrear
  - Manual técnico y de usuario
  - Proyecto en el repositorio con visualización de cambios
  - Realismo del espacio virtual contra la foto de referencia
  - Archivo Ejecutable
  - Modelar 7 elementos ya especificados en el pdf
  - 3 animaciones sencillas y 2 animaciones complejas
  - Iluminación correcta (ambientación)
  - Manejo de cámara (sintética)

- **Diseño**
  - Diagrama de Gantt



  - Limitantes:
    - Anticipo que el rendimiento del equipo de hardware podría verse afectado al manejar varios modelos en el programa, con posibles

demoras en la compilación en OpenGL. Para abordar esta cuestión, planeo optimizar el código.
    - Estoy consciente de la necesidad de cumplir con el cronograma, sin embargo, al verlo considero que al final hay varias tareas en paralelo, lo que podría generar acumulación de trabajo. En este sentido, considero la posibilidad de omitir la incorporación del skybox, a pesar de mi deseo inicial de querer incluirlo.
    - La complejidad del proyecto parece no ser tanta, sin embargo, preveo que podrían surgir desafíos en la resolución de problemas, los cuales seguramente podré resolver. Sin embargo, presiento que más que difícil, el proyecto será laborioso y tomará tiempo
    - La adopción de una metodología de software será un nuevo territorio para mí, ya que, aunque he estudiado estas metodologías en materias anteriores, será la primera vez que las aplicaré en la práctica.

- Alcance
    - Este proyecto tiene como objetivo principal la creación de un espacio virtual tridimensional que reproduzca de manera realista los elementos seleccionados en el pdf de referencia, además de 7 elementos con animaciones.

- **Implementación**

En esta etapa de implementación, se siguió el diagrama de Gantt realizado.

- Para modelar los objetos y la fachada en Maya, se siguieron los siguientes pasos:
    - Se buscaron modelos en páginas como turboSquid que se parecieran a los que yo quería.
    - Se importaron los modelos a Maya.
    - Se realizaron los cambios a los modelos
    - Se agregaron texturas a los objetos.
    - Cuando era necesario de ajustaban sus mapas UV

- Para pasar los objetos y la fachada a OpenGL, se siguieron los siguientes pasos:
    - Se agregaban los objetos a la carpeta “models”
    - Se instancia un objeto de la clase model y se le asigna un nombre junto con la ruta del obj
    - Se dibujaba el objeto
    - En caso de ser necesario se rotaba o trasladaba el objeto

- Las animaciones sencillas fueron las siguientes:
  - Abrir y cerrar la puerta de un gabinete (mueble de cocina)
  - Mover el brazo de un maniquí
  - Rotar las hélices de un ventilador de techo

- Las animaciones complejas fueron las siguientes:
  - Una licuadora encendida haciendo que se agite un poco hacia los lados
  - Que salga un pan de una tostadora

- Se agregó la iluminación adecuada:
  - 4 pointlights situados en una lampara
  - Spotlight
  - Dirlight

- Manejo de cámara
  - Es cámara sintética, asi estaba por defecto

Todo lo mencionado anteriormente lo iba subiendo poco a poco a mi repositorio de GitHub.
El tiempo de desarrollo mas significativo fue esta parte de la implementación, sobre todo por la realización de la fachada y los modelos.

- **Pruebas**

No se realizaron pruebas unitarias, de integracion, de sistema, de aceptación de usuario ni alguna de esas, lo que hice fue hacer lo que se llama **“prueba de observación”** o **“prueba de inspección”**, el cual solo es un enfoque de prueba que se basa en la revisión visual directa de un sistema para identificar posibles errores, defectos o áreas de mejora. No es el tipo de prueba más eficaz, pero era lo que se adaptaba a mis necesidades de ver que se viera relista el espacio y de esa manera mejorar el proyecto.

- **Entrega**

Una vez teniendo todo lo anterior se hace el ultimo commit a GitHub y se entrega el proyecto teniendo en cuenta todos los requerimientos.

Documentación del codigo

- **Importación de bibliotecas**
  Incluye bibliotecas para manejar ventanas (GLFW), extensiones de OpenGL (GLEW), matemáticas (GLM), cargar imágenes (stb_image), y texturas

(SOIL2). También se importan clases para gestionar shaders, cámaras y modelos en la aplicación.

```
#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"
```

- **Declaración de prototipos**
  Son para funciones relacionadas con la interacción del usuario en la ventana de OpenGL, "KeyCallback" maneja eventos de teclado, "MouseCallback" maneja eventos de ratón, y "DoMovement" realiza acciones asociadas con el movimiento del usuario en la escena tridimensional.

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

- **Dimensiones de la ventana**
  Definen constantes para el ancho y alto de la ventana (WIDTH y HEIGHT) con valores 800 y 600, respectivamente. Además, se declaran variables (SCREEN_WIDTH y SCREEN_HEIGHT) para almacenar las dimensiones reales de la pantalla.

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

- **Declaración de las variables para animaciones sencillas y complejas**
  Aquí se declaran las variables que se van a utilizar para la realización de las animaciones sencillas y complejas, las booleanas son utilizadas para activar

la animación, las que empiezan con *"rot"* controlan la rotación del objeto. Todo esta inicializado en 0 o en falso.

```
//Variables para animacion (sencilla)
 //Gabinete
 float rotPuertaGab = 0.0f;
 bool animPuertaGab = false;
 bool animPuertaGab2 = true;
 //Maniqui
 float rotBrazoManiqui = 0.0f;
 bool animBrazoManiqui = false;
 bool animBrazoManiqui2 = true;
 //Ventilador
 float rotVentilador = 0.0f;
 bool animVentilador = false;

// Variables para animacion (compleja)
 // Licuadora
 float tiempo;
 float speed;

 //Tostadora
 float tiempo2;
 float speed2;
```

- **Configuración y manejo de la cámara**
  Se le asigna una posicion inicial a la cámara, "lastX" y "lastY" almacenan la ultima posicion del cursor en su respectivo eje.

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

- **Atributos y posiciones de la luz**
  Se posicionan los 4 pointlight utilizados, los cuales van en una lampara

```
// Light attributes
glm::vec3 lightPos(0.0f, 0.0f, 0.0f);
bool active;

// Positions of the point lights
glm::vec3 pointLightPositions[] = {
    glm::vec3(0.27f, 4.0f, 3.16f),
    glm::vec3(0.27f, 4.0f, 3.95f),
    glm::vec3(-0.1f, 4.0f, 3.55f),
    glm::vec3(0.7f, 4.0f, 3.55f)
};
```

- **Vértices**

Se define un conjunto de vértices para para definir la geometría de un cubo en 3D.

```
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,
```

- **Variables del tiempo del procesador**

"deltaTime" representa el tiempo entre el cuadro actual y el último cuadro, mientras que *"lastFrame"* almacena el tiempo del último cuadro.

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

- **Función principal**
  - **Configuración de OpenGL**

    Se crea le ventana, configura GLEW y GLFW. Adicionalmente define las dimensiones del viewport en función del tamaño de la ventana creada.

```
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);
```

  - **Carga de shaders**

    Se cargan los shaders para iluminación y lámpara.
    Tambien los shaders utilizados para las animaciones de la tostadora y la licuadora.

```
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");
```

- **Carga de modelos**

  Se cargan los modelos que posteriormente se van a dibujar, en estos tiene que estar bien especificada la ruta de la carpeta donde se ubican

```
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");
```

- **Configuración de VAO y VBO**

  Establece el VAO (Vertex Array Object) y el VBO (Vertex Buffer Object). Define atributos de posición y normales para los vértices y los habilita en el 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();
```

- **Bucle principal**
  - **Configuración del ambiente (iluminación)**

Usando el lightning shader se crea una directional light, 4 pointlights y un spotlight

```
261         // Directional light (Solo debe haber una)
262         glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
263         glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.4f, 0.4f, 0.4f);
264         glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.4f, 0.4f, 0.4f);
265         glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.5f, 0.5f, 0.5f); //Intensidad
266
267
268         // Point light 1
269         glm::vec3 lightColor;
270         lightColor.x = abs(sin(glfwGetTime() * Light1.x));
271         lightColor.y = abs(sin(glfwGetTime() * Light1.y));
272         lightColor.z = sin(glfwGetTime() * Light1.z);
273
274
275         glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"), pointLightPositions[0].x, poin
276         glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"), lightColor.x, lightColor.y, lig
277         glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), lightColor.x, lightColor.y, lig
278         glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"), 0.2f, 0.2f, 0.2f);
279         glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].constant"), 1.0f);
280         glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].linear"), 0.09f);
281         glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"), 0.032f);
```

…

```
303         // Point light 4
304         glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].position"), pointLightPositions[3].x, point
305         glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].ambient"), 0.0f, 0.0f, 0.0f);
306         glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].diffuse"), 0.0f, 0.0f, 0.0f);
307         glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].specular"), 0.2f, 0.2f, 0.2f);
308         glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].constant"), 1.0f);
309         glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].linear"), 0.09f);
310         glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].quadratic"), 0.032f);
311
312         // SpotLight
313         glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.position"), camera.GetPosition().x, camera.GetPo
314         glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.direction"), camera.GetFront().x, camera.GetFron
315         glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.ambient"), 1.0f, 1.0f, 1.0f);
316         glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.diffuse"), 1.0f, 1.0f, 1.0f);
317         glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.specular"), 1.0f, 1.0f, 1.0f);
318         glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.constant"), 1.0f);
319         glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.linear"), 0.32f);
320         glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.quadratic"), 0.45f);
321         glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.cutOff"), glm::cos(glm::radians(12.0f)));
322         glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.outerCutOff"), glm::cos(glm::radians(15.0f)));
323
324         // Set material properties
325         glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
```

- **Dibujar de modelos**

  En esta sección se cargan todos los modelos que se van a utilizar los cuales son la fachada, cama, gabinete, mesa, silla, maniquí, planta, lampara, ventilador, la base de la licuadora y tostadora.

  Todos estos objetos mencionados anteriormente se cargaron primero debido a que son objetos solidos (no llevan transparencia)

```
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);
```

Posteriormente se activa el canal alfa para la carga de modelos transparentes o semitransparentes, que en este caso fueron las ventanas de la casa. Al final se desactiva el canal alfa para no afecta lo que prosigue en el codigo

```
//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);
```

- **Animación compleja**

Aquí se hacen las animaciones complejas de la licuadora y la tostadora, se dibujan aquí debido a que utilizan 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

Para realizarla se definió una amplitud, distancia, frecuencia y tiempo las cuales multiplican a la función seno para que la licuadora oscile.
Además, definí un efecto lateral el cual multiplica el efecto mencionado anteriormente por la atenuación, este efecto lateral se le aplico a X y 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

Para hacer que el pan salga de la licuadora ocupe un coseno, este se multiplica por amplitud y frecuencia el cual ocasiona lo siguiente:
Mayor Frecuencia: El pan se mueve más rápido.
Menor Frecuencia: El pan se mueve más lento.

Mayor Amplitud: Pan sube y baja más, es como si el pan saliera volando más alto.
Menor Amplitud: Movimiento más sutil.

```
#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);
}
```

- **Funciones del teclado**
  "KeyCallback" maneja eventos de teclado, *"MouseCallback"* maneja eventos de ratón, y *"DoMovement"* realiza acciones asociadas con el movimiento del usuario en la escena tridimensional. Todas son de tipo 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     {
```

Diccionario de funciones

Función	Variables	Descripción
void DoMovement	camera pointLightPositions animPuertaGab animPuertaGab2 rotPuertaGab animBrazoManiqui animBrazoManiqui2 rotBrazoManiqui animVentilador rotVentilador	Esta función lo que hace es activarse constantemente cuando se presiona una tecla debido a que está dentro del *while* y no se cancelara hasta que se cierre el programa. • camera se utiliza para mover la cámara al presionar las teclas W,A,S,D o las flechas • pointLightPositions se utilizo para mover los pointlight de manera mas sencilla ya que solo imprimias la posicion y despues podias acomodarlos en el arreglo de pointlights, se mueve con las teclas T,G,Y,H,U,J • Si la animación de la puerta del gabinete está activada (animPuertaGab es

		true), la variable rotPuertaGab se ajusta para realizar una animación de rotación en la puerta del gabinete. • Si la animación del brazo del maniquí está activada (animBrazoManiqui es true), la variable rotBrazoManiqui se ajusta para realizar una animación de rotación en el brazo del maniquí. • Si la animación del ventilador está activada (animVentilador es true), la variable rotVentilador se ajusta para realizar una animación de rotación en el ventilador.
void KeyCallback	animPuertaGab animBrazoManiqui animVentilador	Esta función lo que hace es activarse cuando se presiona o se suelte una tecla • Verifica si se presiona la tecla escape, si lo hace sale del programa • Si se presiona la tecla P se activa la animación de la puerta del gabinete • Si se presiona la tecla O se activa la animación del brazo del maniquí • Si se presiona la tecla L se activa la animación de las hélices del ventilador
void MouseCallback	firstMouse lastX lastY firstMouse camera	Esta función lo que hace es manejar los eventos de movimiento del ratón en la ventana • firstMouse: Controla si es el primer movimiento del ratón para inicializar las posiciones. • lastX y lastY: Almacenan las últimas coordenadas X e Y del ratón para calcular la diferencia. • xPos y yPos: Representan las coordenadas actuales del ratón en la ventana GLFW. • Camera: Representa la cámara; se utiliza para ajustar la orientación en respuesta al movimiento del ratón.

int main	keys[1024], animPuertaGab, animBrazoManiqui, animVentilador, rotPuertaGab, rotBrazoManiqui, rotVentilador, pointLightPositions[4], deltaTime, lastFrame, lastX, lastY, firstMouse, WIDTH, HEIGHT, SCREEN_WIDTH, SCREEN_HEIGHT, Shader lightingShader, Shader lampShader, Shader AnimLicuadora, Shader AnimTostadora, Model Fachada, Model Ventanas, Model Sillon, GLuint VBO, VAO, glm::mat4 projection, speed, tiempo, tiempo2	Es el main del codigo, por lo que esta es la parte mas importante del codigo, aquí se crea la ventana, se cargan los modelos 3D, se definen los shaders para la iluminación y las animaciones, establece el bucle(while) donde se gestionan todos los eventos como actualizar la escena y renderizar los objetos importados desde Maya y se implementan parte de las animaciones. En general todo es importante en el programa, pero esta es la parte mas importante debido a esto esta en el main. • keys[1024]: Estado de teclas. • animPuertaGab, animBrazoManiqui, animVentilador: Activadores de animaciones. • rotPuertaGab, rotBrazoManiqui, rotVentilador: Ángulos de rotación. • pointLightPositions[4]: Posiciones de luces. • deltaTime: Diferencia de tiempo. • lastFrame: Tiempo del último cuadro. • lastX, lastY: Coordenadas del último movimiento del ratón. • firstMouse: Primer movimiento del ratón. • WIDTH, HEIGHT: Dimensiones de la ventana. • SCREEN_WIDTH, SCREEN_HEIGHT: Ancho y alto de la ventana de visualización. • Shader lightingShader, Shader lampShader, Shader AnimLicuadora, Shader AnimTostadora: Shaders. • Model Fachada, Model Ventanas, ..., Model Sillon: Modelos 3D. • GLuint VBO, VAO: Identificadores de búfer y arreglo de vértices. • glm::mat4 projection: Matriz de proyección. • speed: Velocidad de animaciones.

		• tiempo, tiempo2: Tiempo multiplicado por velocidad en animaciones.

Análisis de costos

Costo variable	Costo fijo
Licencia Maya	Internet
	Agua
	Transporte
	Renta
	% computadora
	Modelos
	Programador

Licencia Maya
Costo mensual: $2,797
Costo total para 3 meses: $2,797 x 3 = $8,391

Internet
Costo mensual: $500
Costo total para 3 meses: $500 x 3 = $1,500

Agua
Costo mensual: $587.13
Costo total para 3 meses: $587.13 x 3 = $1,761.39

115007310
Grupo: 5

Transporte
No se realizó ningún viaje, todo se hizo desde casa.
Costo= $0

Renta
No se paga renta, pero entre los demas gastos de la casa se consideró un aproximado de $4,000

Computadora
Costo de la computadora: $18,000
Vida útil estimada: 5 años

Depreciación Anual:
Depreciación anual = Costo de la computadora / Vida útil
Depreciación anual = $18,000 / 5 = $3,600 por año

Depreciación para la Duración del Proyecto (3 meses):
Depreciación para el proyecto = Depreciación anual * (Duración del proyecto en meses / 12)
Depreciación para el proyecto = $3,600 * (3 / 12) = $900

Modelos
Costo por cada modelo: $500
Costo de 15 modelos: $500 x 15 = $7500

Programación en OpenGL
Costo por hora como programador: $600
Horas invertidas en la programación: 20
Costo: $600 x 25: $12000

Costo total del proyecto:

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

115007310
Grupo: 5

Conclusiones

Realizar este proyecto no fue una tarea sencilla, a pesar de que no modelaba los objetos desde cero me resulto un poco difícil modificarlos para adecuarlos a mi proyecto. Tampoco fue fácil texturizar ya que mi fachada no era de un solo color solido y tenia que aplicar varias texturas en una misma cara, sobre todo los modelos de madera eran los difíciles de texturizar, pero aprendí a usar los distintos tipos de proyecciones para texturizar correctamente.

Para este proyecto pusimos en practica todo lo aprendido en clase a lo largo del semestre (modelado, animaciones, iluminación, etc.) y creo yo que a pesar de que no me quedo tan perfecto como yo quería logre plasmar que si aprendí durante el semestre, además me gusto que tuviéramos que ser autodidactas para el manejo del software de modelado (Maya) y el editor de fotos (gimp) lo cual es una habilidad muy importante en nuestro ambiente de ingenieros en computación. Lo único que no me gustó tanto como me quedo fueron las animaciones complejas.

Algo que también me gustó fue el hecho de que tambien tuvimos que utilizar una metodología de software, las cuales son muy utilizadas en el ámbito laboral para mantener un orden y la documentación de un proyecto. Solo las había visto en clase, pero nunca había utilizado una para realizar un proyecto mío hasta ahora y me doy cuenta de su importancia.

A pesar de todo el largo proceso y dedicación que tuve para todo el proyecto disfruté el proceso ya que es un proyecto muy diferente al que suelen dejar en las demas materias, al ver el resultado es padre porque te das cuenta de todo lo que aprendiste a hacer en 11 semanas durante el semestre.

Además de todo lo anterior me pareció muy interesante hacer un análisis de costos ya que es la primera vez que hago uno y pienso que es muy útil ya que si en algun momento tengo un trabajo de FreeLancer ya tengo una idea de cuanto podría cobrar por algun proyecto.