UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

INGENIERÍA EN COMPUTACIÓN

COMPUTACIÓN GRÁFICA e INTERACCIÓN HUMANO COMPUTADORA

PROYECTO FINAL – LABORATORIO

MANUAL DE USUARIO Y MANUAL TÉCNICO

NOMBRE COMPLETO: Colón Palacios Emmanuel

N.º de Cuenta: 317254523

GRUPO DE TEORÍA: 04

GRUPO DE LABORATORIO: 06

SEMESTRE 2023-1

FECHA DE ENTREGA LÍMITE: 06/11/2022

Contenido

Objetivo

Mediante el uso de las herramientas aprendidas durante el curso, como OpenGL, Maya y Gimp, así como de la teoría aplicada en práctica en las sesiones que hemos tenido en el laboratorio, el alumno aplicará dichos conocimientos en un ambiente virtual (fachada y 7 objetos) recreado dentro de OpenGL.

Alcance

El ambiente para recrear será la fachada e interior de la *Estación Pingüi-fónica* del popular juego *Club Penguin*, cabe mencionar que tendrá la fachada del evento "*Fiesta Medieval*" y no la original, además de que el interior será el último recibido en actualizaciones, siendo recreado con 7 objetos seleccionados previamente y que serán mencionados una vez más en este documento.

Para obtener mayor grado de inmersión se colocó un *SkyBox* que simula un entorno nevado y un piso que simula estar en la nieve.

Será explorable gracias al manejo de la cámara introducida en el código, más adelante se explicará cómo manipularla.

Introducción

Como se mencionó en la pestaña de **Alcance**, el espacio a recrear será el edificio "***Estación Pingüi-fónica***", junto con su primer interior, a continuación, se muestran los antes mencionados.

- Fachada

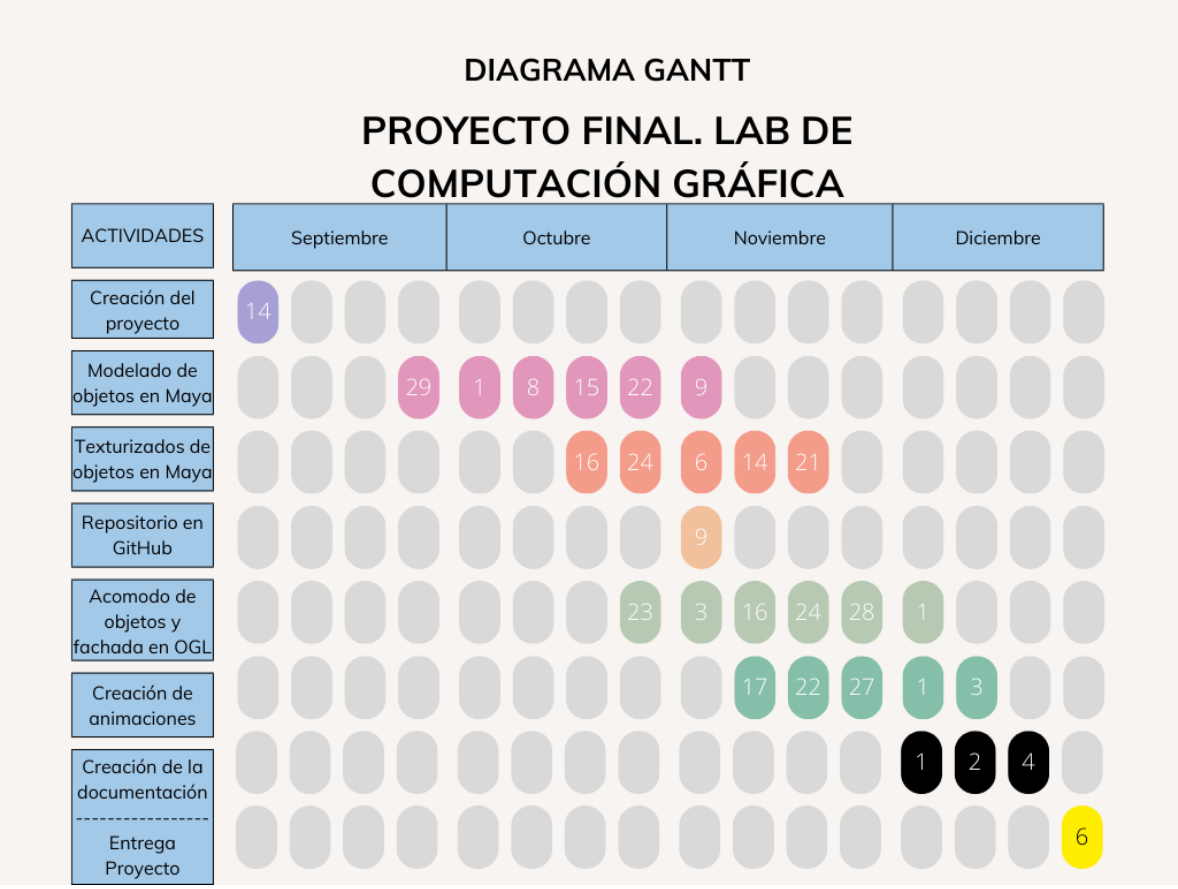
- Interior

Del interior mostrado se seleccionaron 7 objetos, los cuales son:

- Cabina telefónica.
- Teléfono fijo.
- Teléfono gigante.
- Monitor.
- Silla.
- Escritorio
- Máquina de café.

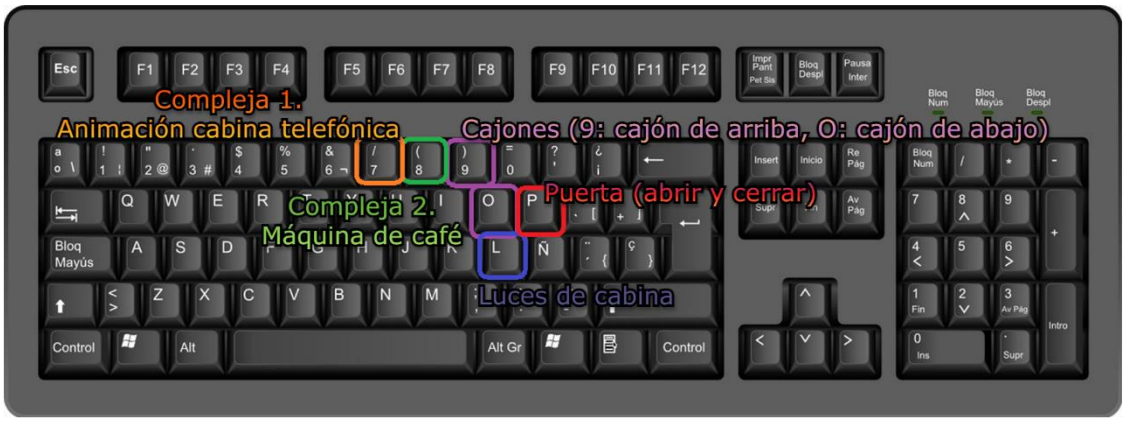
Estos serán los siete objetos para recrear en el software MAYA 2023 para luego ser utilizados en OpenGL para recrear nuestro espacio virtual.

Diagrama de Gantt



Asignación de teclas

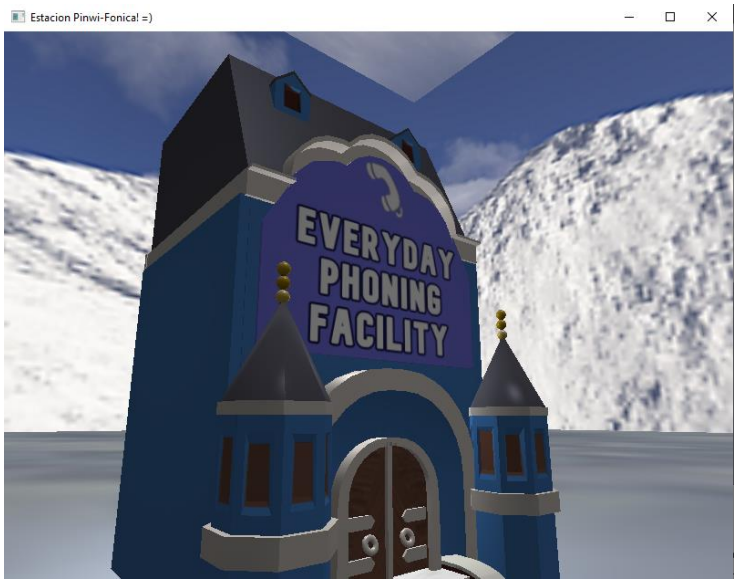
Animaciones

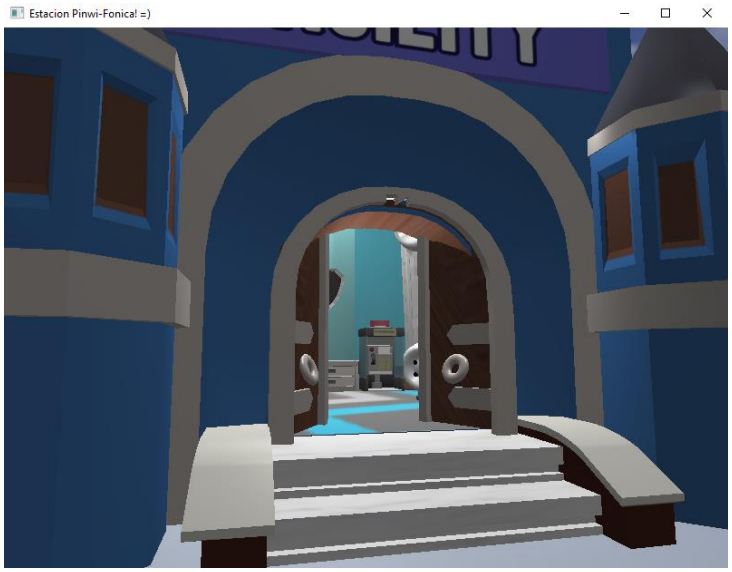


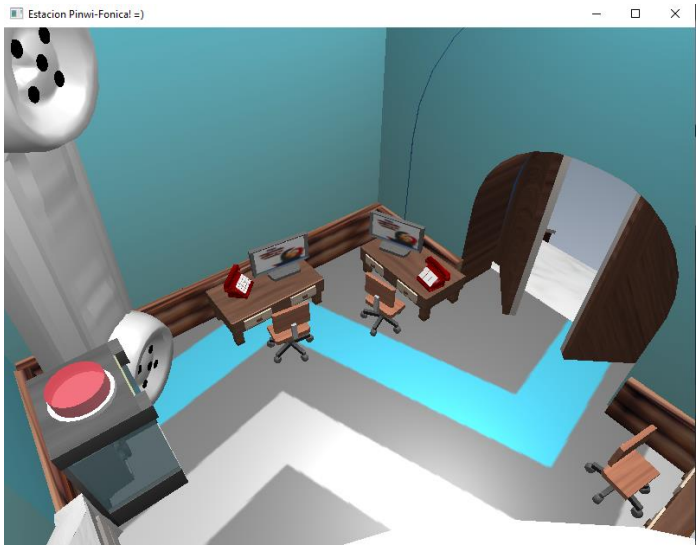
Movimiento de la cámara

También hace uso del ratón para la dirección a la que está mirando la cámara.

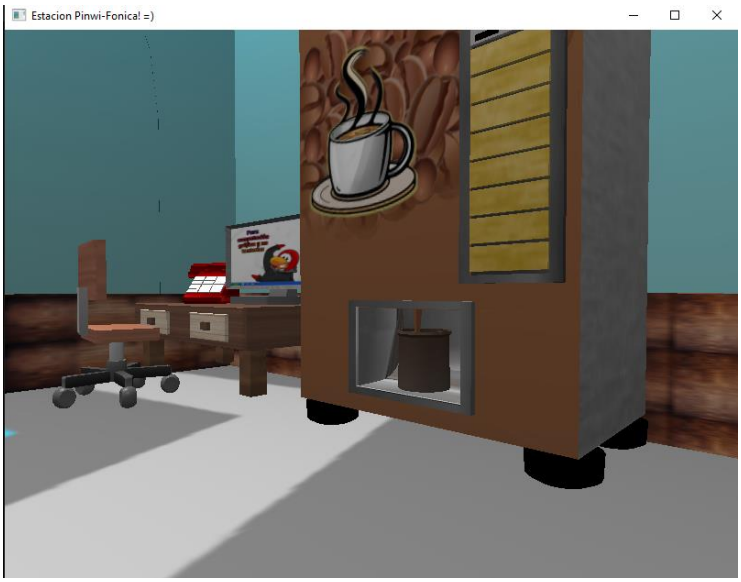
Recorrido del ambiente en OpenGL




Estacion Pinwi-Fonica! =)

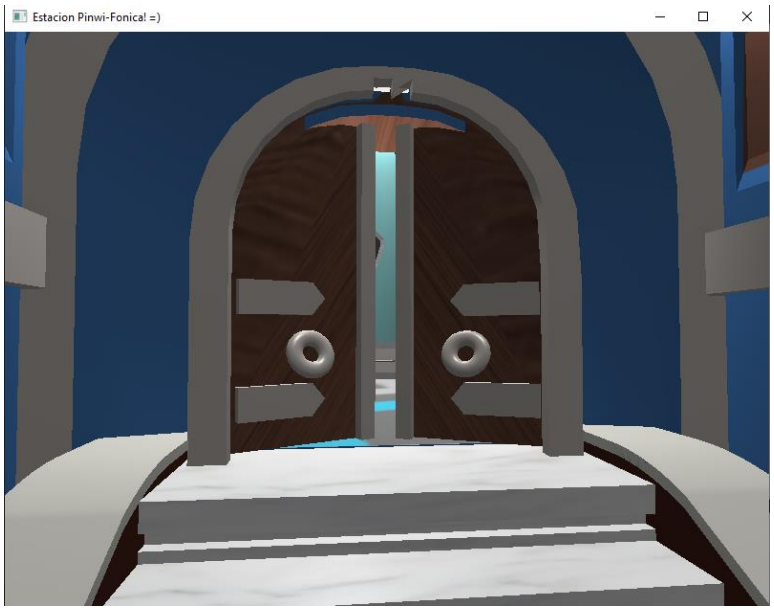

Estacion Pinwi-Fonica! =)

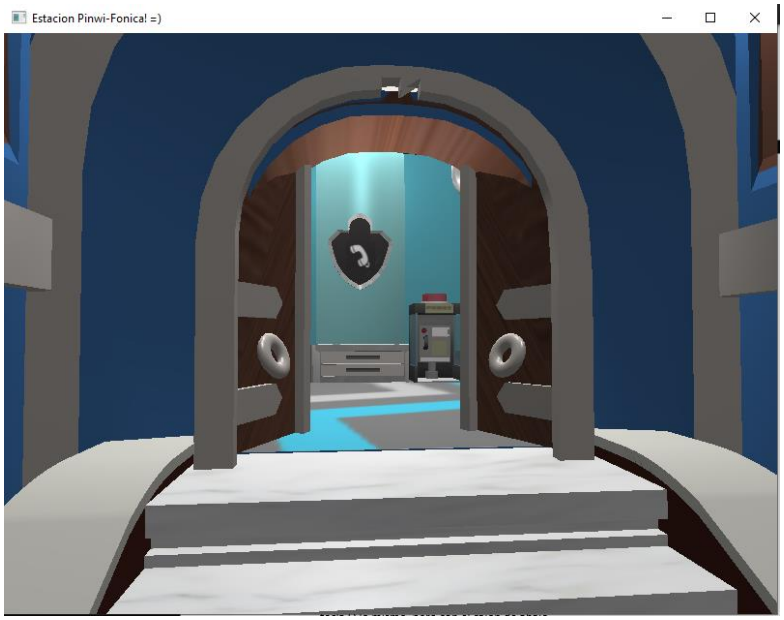

Estacion Pinwi-Fonica! =)
PHONE


Estacion Pinwi-Fonica! =)

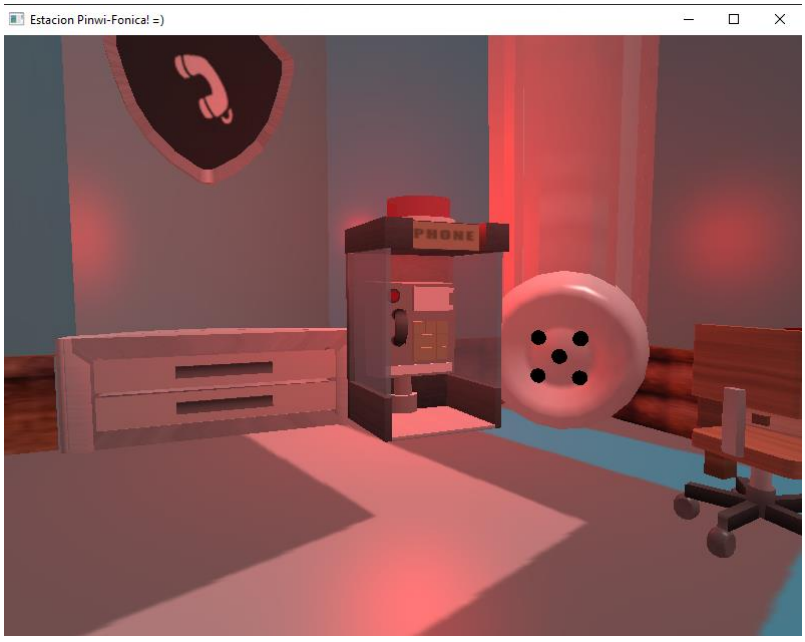
Animaciones

1. **PRIMERA ANIMACIÓN SIMPLE:** Con la tecla P podemos abrir y cerrar las puertas de la fachada.

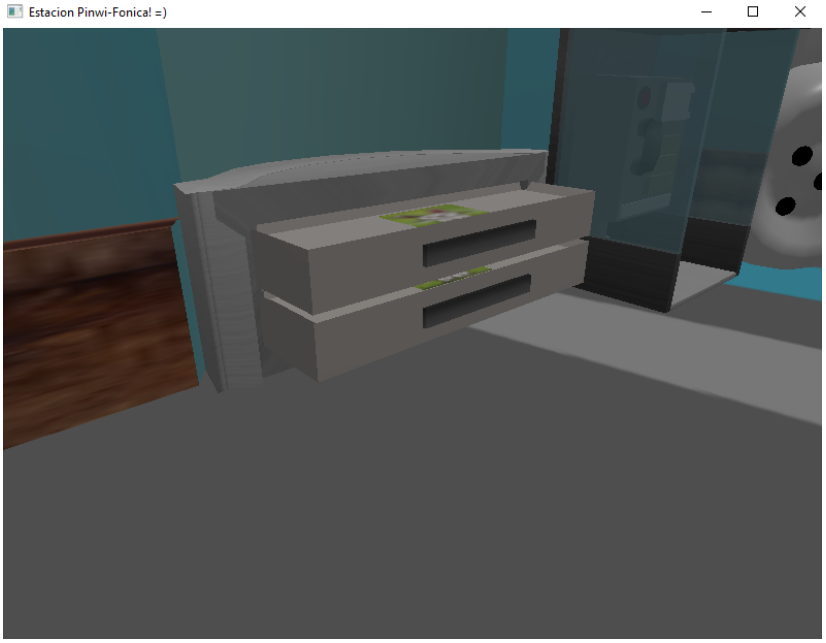


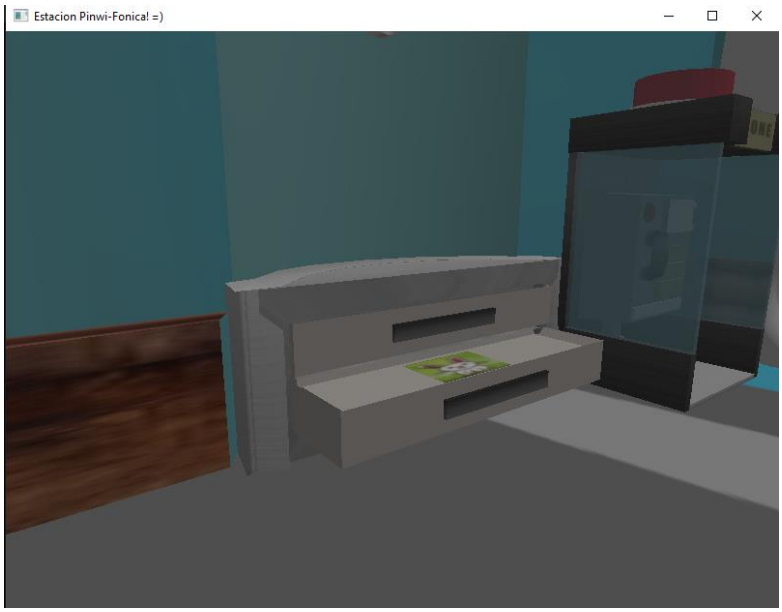


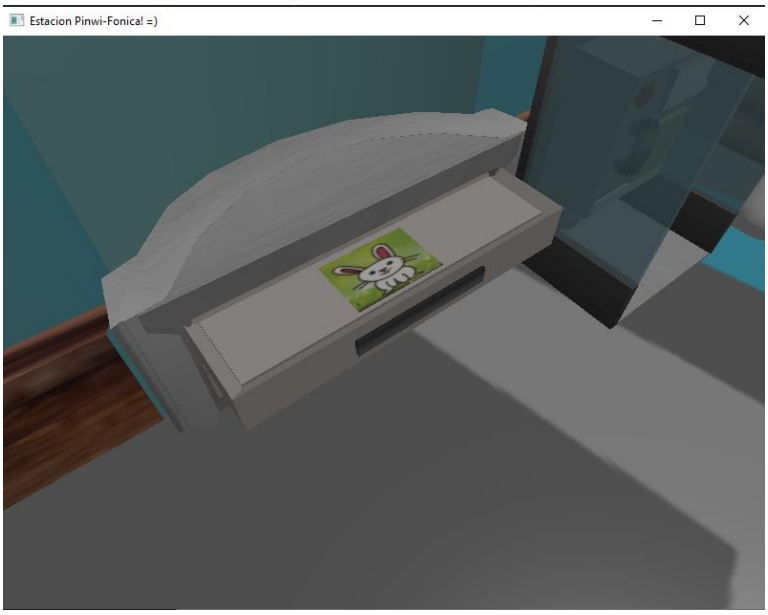
2. **SEGUNDA ANIMACIÓN SIMPLE:** Con la tecla L podemos prender las luces de la cabina telefónica, simulando una llamada entrante



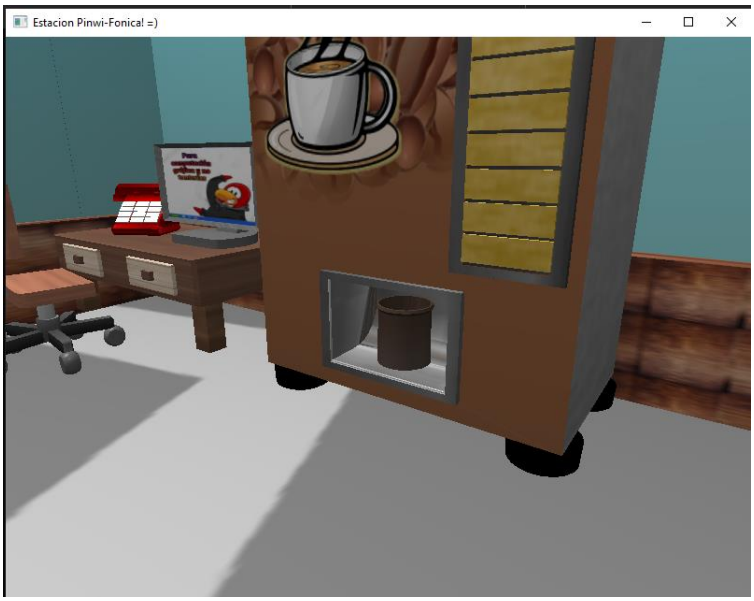
3. **TERCERA ANIMACIÓN SIMPLE:** Con la tecla 9 puedes abrir y cerrar el cajón de arriba, con la tecla O lo mismo, pero con el cajón de abajo.







4. **PRIMERA ANIMACIÓN COMPLEJA**: Con la tecla 8 la máquina de café comienza a preparar un café, baja líquido y una tapa. Animación con KEYFRAMES.




Estacion Pinwi-Fonica! =)

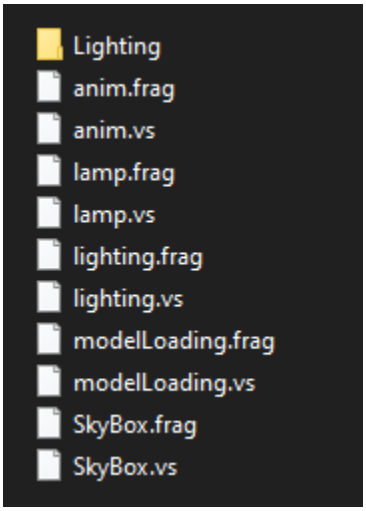

Estacion Pinwi-Fonica! =)

5. **SEGUNDA ANIMACIÓN COMPLEJA**: Con la tecla 7 presionas una secuencia de botones en la cabina telefónica que te permiten acceder a un equipamiento secreto. Animación con KEYFRAMES.

Estructura del proyecto

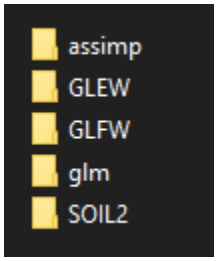
Shaders

Hicimos uso de varios shaders que fueron inicializados en el código final, tanto para iluminación como para animaciones y el skybox.



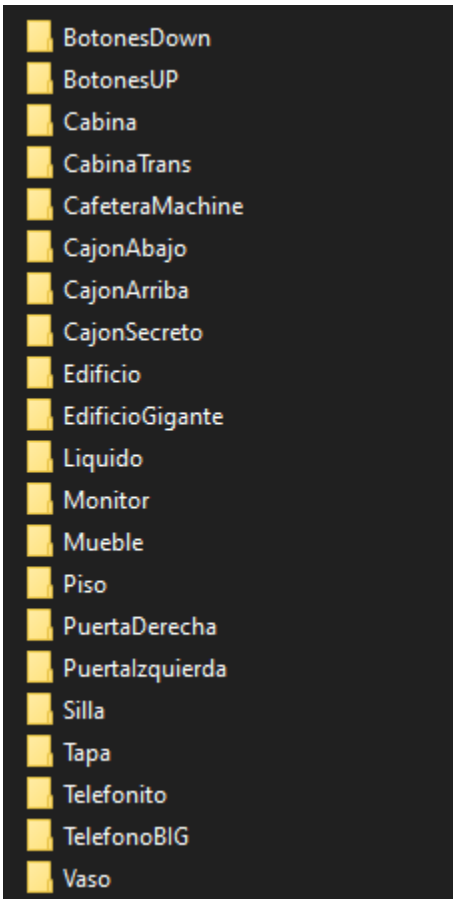
Bibliotecas

También hicimos uso de bibliotecas que nos dotaban de métodos necesarios para introducir modelos a OpenGL.



Modelos y texturas

Todos los modelos hechos en MAYA 2023 exportados a nuestro proyecto para ser implementados correctamente con OpenGL.



Código

Librerías en código

```
#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"
#include "Texture.h"
#include "modelAnim.h"
```

Incluyendo las declaraciones de todas las bibliotecas y shaders que se usarán en el código.

Prototipos de las funciones

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

Declarando los prototipos de las funciones necesarias para realizar animaciones en respuesta a una tecla de entrada del teclado.

Ventana y dimensiones

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

Cámara

```
// Camera
Camera  camera(glm::vec3(-95.0f, 11.0f, -25.0f));
GLfloat lastX = WIDTH / 2.0;
GLfloat lastY = HEIGHT / 2.0;
bool keys[1024];
bool firstMouse = true;
float range = 0.0f;
float rot = 0.0f;
float movCamera = 0.0f;
```

Posición en la que estará la cámara al momento de ejecutar el código en ventana.

Variables y estructura creadas para animaciones complejas con keyframes

```
// Keyframes

float posX = PosIni.x, posY = PosIni.y, posZ = PosIni.z, rotRodIzq = 0;

//Primera animación compleja
float cafeKF = 4.0f, tapaKF = 3.85f, tapaVKF = 4.0f, vasoVKF = 5.0f, vasoKF = 3.85f;

//Segunda animación compleja
float botonUKF = 5.15f, botonDKF = 5.15f, cajonSecretoKF = 5.15f;

#define MAX_FRAMES 9
// Contador Animación compleja 1.
int i_max_steps = 190;
int i_curr_steps = 0;

// Contador Animación compleja 2.
int i_max_steps2 = 190;
int i_curr_steps2 = 0;

typedef struct _frame
{
    //Variables para GUARDAR Key Frames
    float posX;     //Variable para PosicionX
    float posY;     //Variable para PosicionY
    float posZ;     //Variable para PosicionZ
    float incX;     //Variable para IncrementoX
    float incY;     //Variable para IncrementoY
    float incZ;     //Variable para IncrementoZ
    float rotRodIzq;
    float rotInc;

    // Primera animación //
    float cafeKF = 0.0f, cafeInc = 0.0f;
    float tapaKF = 0.0f, tapaInc = 0.0f;
    float tapaVKF = 0.0f, tapaVInc = 0.0f;
    float vasoVKF, vasoVInc;
    float vasoKF = 0.0f, vasoInc = 0.0f;

    // Segunda animación //
    float botonUKF = 0.0f, botonDKF = 0.0f, cajonSecretoKF = 0.0f;
    float botonUInc = 0.0f, botonDInc = 0.0f, cajonSecretoInc = 0.0f;


}FRAME;
```

```
FRAME KeyFrame[MAX_FRAMES];
int FrameIndex = 4;         //introducir datos
bool play = false;
int playIndex = 0;

FRAME KeyFrame2[MAX_FRAMES];
int FrameIndex2 = 5;            //introducir datos
bool play2 = false;
int playIndex2 = 0;

// Variables para ANIMAR

#define CAFE_MOV 0.01f
#define TAPA_MOV 0.1f
#define TAPAV_MAX 4.0f

float rotPuertaD = 0.0f, rotPuertaI= 0.0f, movCajon = 0.0f, movCajon2 = 0.0f;
bool abierto = false;
bool abrir = true;
bool cerrar = false;
bool abiertoCajon = false, abrirCajon = true, cerrarCajon = false;
bool abiertoCajon2 = false, abrirCajon2 = true, cerrarCajon2 = false;
```

Variables y estructura necesaria para las animaciones con KeyFrames y para la posición correcta de la cámara.

Luces de ambiente y de la cabina telefónica

```
// Positions of the point lights
glm::vec3 pointLightPositions[] = {
    glm::vec3(posX,posY+11.4,posZ-5.3),
    glm::vec3(posX+ 2.0 ,posY + 6.1, posZ - 4.5),
    glm::vec3(posX,posY + 11.4,posZ + 2.0),
    glm::vec3(0,0,0)
};

glm::vec3 LightP1;
glm::vec3 LightP2;
glm::vec3 LightP3;
```

Son las luces que se encienden con la tecla ESPACIO y con L como se especificó en la sección ***Asignación de teclas*.**

Funciones necesarias para las animaciones con keyframes

```
void resetElements(void)
{

    cafeKF = KeyFrame[0].cafeKF;
    tapaKF = KeyFrame[0].tapaKF;
    tapaVKF = KeyFrame[0].tapaVKF;
    vasoVKF = KeyFrame[0].vasoVKF;
    vasoKF = KeyFrame[0].vasoKF;


}

void resetElements2(void)
{
    botonUKF = KeyFrame2[0].botonUKF;
    botonDKF = KeyFrame2[0].botonDKF;
    cajonSecretoKF = KeyFrame2[0].cajonSecretoKF;
}

void interpolation(void)
{


    KeyFrame[playIndex].cafeInc = (KeyFrame[playIndex + 1].cafeKF - KeyFrame[playIndex].cafeKF) / i_max_steps;
    KeyFrame[playIndex].tapaInc = (KeyFrame[playIndex + 1].tapaKF - KeyFrame[playIndex].tapaKF) / i_max_steps;
    KeyFrame[playIndex].tapaVInc = (KeyFrame[playIndex + 1].tapaVKF - KeyFrame[playIndex].tapaVKF) / i_max_steps;
    KeyFrame[playIndex].vasoVInc = (KeyFrame[playIndex + 1].vasoVKF - KeyFrame[playIndex].vasoVKF) / i_max_steps;
    KeyFrame[playIndex].vasoInc = (KeyFrame[playIndex + 1].vasoKF - KeyFrame[playIndex].vasoKF) / i_max_steps;
}

void interpolation2(void)
{


    KeyFrame2[playIndex2].botonUInc = (KeyFrame2[playIndex2 + 1].botonUKF - KeyFrame2[playIndex2].botonUKF) / i_max_steps;
    KeyFrame2[playIndex2].botonDInc = (KeyFrame2[playIndex2 + 1].botonDKF - KeyFrame2[playIndex2].botonDKF) / i_max_steps;
    KeyFrame2[playIndex2].cajonSecretoInc = (KeyFrame2[playIndex2 + 1].cajonSecretoKF - KeyFrame2[playIndex2].cajonSecretoKF) / i_max_steps;


}
```

La resetElements() hace que comience la animación desde el primer estado en las animaciones con KeyFrames, mientras que Interpolation() hace que circulen correctamente las animaciones guardadas en el arreglo de tipo FRAME.

Creación de Shaders necesarios para el ambiente

```
Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting.frag");
Shader lampShader("Shaders/lamp.vs", "Shaders/lamp.frag");
Shader SkyBoxshader("Shaders/SkyBox.vs", "Shaders/SkyBox.frag");
Shader animShader("Shaders/anim.vs", "Shaders/anim.frag");
```

Creación de modelos importados desde Maya con .obj

```
// Fachada //
Model Edificio((char*)"Models/EdificioGigante/EdificioGigante.obj");
Model PuertaIzq((char*)"Models/PuertaIzquierda/PuertaIzquierda.obj");
Model PuertaDer((char*)"Models/PuertaDerecha/PuertaDerecha.obj");
Model CajonAbajo((char*)"Models/CajonAbajo/CajonAbajo.obj");
Model CajonArriba((char*)"Models/CajonArriba/CajonArriba.obj");

// 7 Objetos //
Model Cabina((char*)"Models/Cabina/Cabina.obj");
Model CabinaTrans((char*)"Models/CabinaTrans/CabinaTrans.obj"); //Transparente
Model CabinaBotonUP((char*)"Models/BotonesUP/BotonesUP.obj");
Model CabinaBotonDOWN((char*)"Models/BotonesDown/BotonesDown.obj");
Model CajonSecreto((char*)"Models/CajonSecreto/CajonSecreto.obj");
Model CafeteraMachine((char*)"Models/CafeteraMachine/CafeteraMachine.obj");
Model Monitor((char*)"Models/Monitor/Monitor.obj");
Model Mueble((char*)"Models/Mueble/Mueble.obj");
Model Silla((char*)"Models/Silla/Silla.obj");
Model TelefonoFijo((char*)"Models/Telefonito/Telefonito.obj");
Model TelefonoGigante((char*)"Models/TelefonoBIG/TelefonoBIG.obj");


// Modelos extra //
Model Piso((char*)"Models/Piso/Piso.obj");
Model Vaso((char*)"Models/Vaso/Vaso.obj");
Model Tapa((char*)"Models/Tapa/Tapa.obj");
Model Liquido((char*)"Models/Liquido/Liquido.obj");
```

Inicialización de todos los modelos a utilizar, importando los archivos .obj desde la carpeta Models de nuestro proyecto.

Inicialización de keyframes

```
for(int i=0; i<MAX_FRAMES; i++)
{
    KeyFrame[i].cafeKF = 0;
    KeyFrame[i].cafeInc = 0;
    KeyFrame[i].tapaKF = 0;
    KeyFrame[i].tapaInc = 0;
    KeyFrame[i].tapaVKF = 0;
    KeyFrame[i].tapaVInc = 0;
    KeyFrame[i].vasoVKF = 0;
    KeyFrame[i].vasoVInc = 0;
    KeyFrame[i].vasoKF = 0;
    KeyFrame[i].vasoInc = 0;
}


KeyFrame[0].vasoVKF = vasoVKF;
KeyFrame[0].vasoKF = vasoKF;
KeyFrame[0].cafeKF = cafeKF;
KeyFrame[0].tapaKF = tapaKF;
KeyFrame[0].tapaVKF = tapaVKF;

KeyFrame[1].vasoKF = 3.83;
KeyFrame[1].vasoVKF = 4.0f;
KeyFrame[1].cafeKF = cafeKF;
KeyFrame[1].tapaKF = tapaKF;
KeyFrame[1].tapaVKF = tapaVKF;

KeyFrame[2].vasoKF = 3.8f;
KeyFrame[2].vasoVKF = 4.0f;
KeyFrame[2].cafeKF = 3.5f;
KeyFrame[2].tapaKF = 3.85f;
KeyFrame[2].tapaVKF = 4.0f;

KeyFrame[3].vasoKF = 3.75f;
KeyFrame[3].vasoVKF = 4.0f;
KeyFrame[3].cafeKF = 4.0f;
KeyFrame[3].tapaKF = 3.75f;
KeyFrame[3].tapaVKF = 3.75f;
```

En esta parte se especifican los cambios en los estados especificados que gracias a Interpolation se irán mostrando en directo.

```
for (int w = 0; w < MAX_FRAMES; w++)
{
    KeyFrame2[w].botonUKF = 0;
    KeyFrame2[w].botonDKF = 0;
    KeyFrame2[w].cajonSecretoKF = 0;
}

KeyFrame2[0].botonUKF = botonUKF;
KeyFrame2[0].botonDKF = botonDKF;
KeyFrame2[0].cajonSecretoKF = cajonSecretoKF;


KeyFrame2[1].botonUKF = 5.185;
KeyFrame2[1].botonDKF = botonDKF;
KeyFrame2[1].cajonSecretoKF = cajonSecretoKF;


KeyFrame2[2].botonUKF = 5.185;
KeyFrame2[2].botonDKF = 5.185;
KeyFrame2[2].cajonSecretoKF = cajonSecretoKF;

KeyFrame2[3].botonUKF = 5.185;
KeyFrame2[3].botonDKF = 5.185;
KeyFrame2[3].cajonSecretoKF = 4.88;

KeyFrame2[4].botonUKF = 5.15;
KeyFrame2[4].botonDKF = 5.15;
KeyFrame2[4].cajonSecretoKF = 4.88;
```

Skybox

```
glEnable(GL_BLEND);

GLuint skyboxVBO, skyboxVAO;
glGenVertexArrays(1, &skyboxVAO);
glGenBuffers(1,&skyboxVBO);
glBindVertexArray(skyboxVAO);
glBindBuffer(GL_ARRAY_BUFFER, skyboxVBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(skyboxVertices),&skyboxVertices,GL_STATIC_DRAW);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT,GL_FALSE, 3 * sizeof(GLfloat), (GLvoid *)0);

// Load textures
vector<const GLchar*> faces;
faces.push_back("SkyBox/right.tga");
faces.push_back("SkyBox/left.tga");
faces.push_back("SkyBox/top.tga");
faces.push_back("SkyBox/bottom.tga");
faces.push_back("SkyBox/back.tga");
faces.push_back("SkyBox/front.tga");

GLuint cubemapTexture = TextureLoading::LoadCubemap(faces);

glm::mat4 projection = glm::perspective(camera.GetZoom(), (GLfloat)SCREEN_WIDTH / (GLfloat)SCREEN_HEIGHT, 0.1f, 1000.0f);

glDisable(GL_BLEND);
```

Carga de los archivos .tga que nos servirán para tener un skybox nevado para una mayor inmersividad.

```
// Draw skybox as last

glDepthFunc(GL_LEQUAL);  // Change depth function so depth test passes when values are equal to depth buffer's content
SkyBoxshader.Use();
view = glm::mat4(glm::mat3(camera.GetViewMatrix()));    // Remove any translation component of the view matrix
glUniformMatrix4fv(glGetUniformLocation(SkyBoxshader.Program, "view"), 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(glGetUniformLocation(SkyBoxshader.Program, "projection"), 1, GL_FALSE, glm::value_ptr(projection));


// skybox cube
glEnable(GL_BLEND);
glBindVertexArray(skyboxVAO);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_CUBE_MAP, cubemapTexture);
glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);
glDepthFunc(GL_LESS); // Set depth function back to default

glDisable(GL_BLEND);


// Swap the screen buffers
glfwSwapBuffers(window);
```

Inicialización de iluminación direccional y pointlights

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


// Point light 1
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"), pointLightPositions[0].x, pointLightPositions[0].y, poi
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"), 0.05f, 0.05f, 0.05f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), LightP1.x, LightP1.y, LightP1.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"), LightP1.x, LightP1.y, LightP1.z);
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 2
glm::vec3 lightColor2;


lightColor2.x = abs(0.5 * sin(glfwGetTime() * LightP2.x * 2));
lightColor2.y = abs(0.5 * sin(glfwGetTime() * LightP2.y * 2));
lightColor2.z = 0.5 * sin(glfwGetTime() * LightP2.z * 0.2);

glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].position"), pointLightPositions[1].x, pointLightPositions[1].y, poi
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].ambient"), lightColor2.x, lightColor2.y, lightColor2.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].diffuse"), lightColor2.x, lightColor2.y, lightColor2.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].specular"), LightP2.x, LightP2.y, LightP2.z);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].linear"), 0.045f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].quadratic"), 0.075f);

// Point light 3


glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].position"), pointLightPositions[2].x, pointLightPositions[2].y, poi
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].ambient"), 0.05f, 0.05f, 0.05f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].diffuse"), LightP1.x, LightP1.y, LightP1.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].specular"), LightP1.x, LightP1.y, LightP1.z);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].linear"), 0.09f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].quadratic"), 0.032f);
```

Las luces del interior de la fachada se inician con la tecla ESPACIO.

Inicialización de los modelos, fachada y los 7 seleccionados

```
// Fachada //
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::translate(model, glm::vec3(0.0f, 2.6f, 0));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0.0);
Edificio.Draw(lightingShader);

view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::translate(model, glm::vec3(1.7f, 5.5f, 4.3f));
model = glm::rotate(model, glm::radians(rotPuertaD), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0.0);
PuertaDer.Draw(lightingShader);

view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::translate(model, glm::vec3(-1.7f, 5.5f, 4.3f));
model = glm::rotate(model, glm::radians(rotPuertaI), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0.0);
PuertaIzq.Draw(lightingShader);

view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::translate(model, glm::vec3(0.0f, 2.6f, movCajon));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0.0);
CajonArriba.Draw(lightingShader);

view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::translate(model, glm::vec3(0.0f, 2.6f, movCajon2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0.0);
CajonAbajo.Draw(lightingShader);
```

En esta parte se dibujan todos los modelos ya importados previamente con el método .Draw.

```
// .Draw de modelos elegidos en el proyecto //

// Cabina //
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::translate(model, glm::vec3(5.15f, 3.9f, 2.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0.0f);
Cabina.Draw(lightingShader);

view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::translate(model, glm::vec3(botonUKF, 3.9f, 2.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0.0f);
CabinaBotonUP.Draw(lightingShader);

view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::translate(model, glm::vec3(botonDKF, 3.9f, 2.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0.0f);
CabinaBotonDOWN.Draw(lightingShader);

view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::translate(model, glm::vec3(cajonSecretoKF, 3.9f, 2.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0.0f);
CajonSecreto.Draw(lightingShader);
```

Objeto translúcido, canal Alpha.

```
// objetos transparentes //

glEnable(GL_BLEND); //inicia transparencia

view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::translate(model, glm::vec3(5.15f, 3.9f, 2.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 4.0f);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0f, 1.0f, 1.0f, 1.0f);
CabinaTrans.Draw(lightingShader);

glDisable(GL_BLEND); //termina transparencia
```

Con la función glEnable(GL_BLEND) y el lighting shader tendremos objetos translúcidos, en este caso tanto las ventanas como las sirenas de la cabina telefónica pasarán por esta transparencia.

Animaciones

```
void animacion()
{
    // Animación sencilla 1. Puerta Abrir/Cerrar, activado con tecla P //
    if (abierto) {

        if (abrir)
        {
            printf("\nQue se abran las puertas!");
            rotPuertaD -= 0.2;
            rotPuertaI += 0.2;
            if (rotPuertaD < -66.0f)
            {
                abrir = false;
            }

        }

        if (cerrar)
        {
            printf("\nQue se cierren las puertas!");
            rotPuertaD += 0.2;
            rotPuertaI -= 0.2;
            if (rotPuertaD >= 0.0f)
            {
                cerrar = false;

            }
        }

    }
```

```
// Animación sencilla 2. Cajón abierto y cerrado, activado con tecla 9 el cajón de arriba y el de abajo con O //

if (abiertoCajon) {

    if (abrirCajon)
    {
        movCajon += 0.01;
        if (movCajon >= 0.5f)
        {
            abrirCajon = false;
        }

    }

    if (cerrarCajon)
    {
        movCajon -= 0.01;
        if (movCajon <= 0.02f)
        {
            cerrarCajon = false;

        }
    }

}

if (abiertoCajon2) {

    if (abrirCajon2)
    {
        movCajon2 += 0.01;
        if (movCajon2 >= 0.5f)
        {
            abrirCajon2 = false;
        }

    }

    if (cerrarCajon2)
    {
        movCajon2 -= 0.01;
        if (movCajon2 <= 0.02f)
        {
            cerrarCajon2 = false;

        }
    }

}
```

```
    //Animación compleja 1. Café
if (play)
{
    if (i_curr_steps >= i_max_steps) //end of animation between frames?
    {
        playIndex++;
        if (playIndex > FrameIndex - 2) //end of total animation?
        {
            printf("\nCafe servido!");
            playIndex = 0;
            play = false;
        }
        else //Next frame interpolations
        {
            i_curr_steps = 0; //Reset counter
            //Interpolation
            interpolation();
        }
    }
    else
    {
        //Draw animation

        cafeKF += KeyFrame[playIndex].cafeInc;
        tapaKF += KeyFrame[playIndex].tapaInc;
        tapaVKF += KeyFrame[playIndex].tapaVInc;
        vasoVKF += KeyFrame[playIndex].vasoVInc;
        vasoKF += KeyFrame[playIndex].vasoInc;

            i_curr_steps++;
        }

}
```

```
//Animación compleja 2. Cajón secreto
if (play2)
{
    if (i_curr_steps2 >= i_max_steps2) //end of animation between frames?
    {
        playIndex2++;
        if (playIndex2 > FrameIndex2 - 2)   //end of total animation?
        {
            printf("\nCajon desbloqueado, toma tu equipamiento!");
            playIndex2 = 0;
            play2 = false;
        }
        else //Next frame interpolations
        {
            i_curr_steps2 = 0; //Reset counter
            //Interpolation
            interpolation2();
        }
    }
    else
    {
        //Draw animation

        botonUKF += KeyFrame2[playIndex2].botonUInc;
        botonDKF += KeyFrame2[playIndex2].botonDInc;
        cajonSecretoKF += KeyFrame2[playIndex2].cajonSecretoInc;

        i_curr_steps2++;
    }
```

```
// Animación sencilla 3. Luces de la cabina telefónica encienden y apagan, se activa con L //

if (keys[GLFW_KEY_L])
{

    active = !active;
    if (active) {
        LightP2 = glm::vec3(1.0f, 0.0f, 0.0f);
        printf("Llamada entrante, luces de la cabina encendidas!");
    }
    else {

        LightP2 = glm::vec3(0.0f, 0.0f, 0.0f);
    }
}
```

En esta sección se programó el comportamiento de las animaciones, posteriormente se determinó la tecla que utilizaría para activarse las animaciones.

Función KeyCallback

```
if (keys[GLFW_KEY_8])
{
    if (play == false && (FrameIndex > 1))
    {
        printf("\nSirviendo café!");
        resetElements();
        //First Interpolation
        interpolation();

        play = true;
        playIndex = 0;
        i_curr_steps = 0;
    }
    else
    {
        printf("\n Cafe servido!");
        play = false;
    }

}
```

```
if (keys[GLFW_KEY_7])
{
    if (play2 == false && (FrameIndex2 > 1))
    {
        printf("\nSecuencia secreta activada, desbloqueando cajón secreto!");
        resetElements2();
        //First Interpolation
        interpolation2();

        play2 = true;
        playIndex2 = 0;
        i_curr_steps2 = 0;
    }
    else
    {
        printf("\nCajón mostrado!");
        play2 = false;
    }

}
```

```
if (keys[GLFW_KEY_SPACE])
{
    active = !active;
    if (active) {

        LightP1 = glm::vec3(1.0f, 1.0f, 1.0f);
        LightP3 = glm::vec3(1.0f, 1.0f, 1.0f);

    }
    else{
        LightP1 = glm::vec3(0.0f, 0.0f, 0.0f);
        LightP3 = glm::vec3(1.0f, 1.0f, 1.0f);

    }
}
```

```
// Animación sencilla 3. Luces de la cabina telefónica encienden y apagan, se activa con L //

if (keys[GLFW_KEY_L])
{

    active = !active;
    if (active) {
        LightP2 = glm::vec3(1.0f, 0.0f, 0.0f);
        printf("Llamada entrante, luces de la cabina encendidas!");
    }
    else {

        LightP2 = glm::vec3(0.0f, 0.0f, 0.0f);
    }
}
```

```
if (keys[GLFW_KEY_P])
{
    abierto = true;
    if (rotPuertaD < -65.0f) {

        cerrar = true;
        abrir = false;
    }
    else if (rotPuertaD > 0.0f) {
        printf("\nQue se abran las puertas!");
        cerrar = false;
        abrir = true;

    }

}
```

```
if (keys[GLFW_KEY_9])
{
    abiertoCajon = true;
    if (movCajon >= 0.5f) {
        printf("\nCajon Cerrado!");
        cerrarCajon = true;
        abrirCajon = false;
    }
    else if (movCajon <= 0.02f) {
        printf("\nCajon Abierto!");
        cerrarCajon = false;
        abrirCajon = true;

    }

}

if (keys[GLFW_KEY_O])
{
    abiertoCajon2 = true;
    if (movCajon2 >= 0.5f) {
        printf("\nCajon Cerrado!");
        cerrarCajon2 = true;
        abrirCajon2 = false;
    }
    else if (movCajon2 <= 0.02f) {
        printf("Cajon Abierto!");
        cerrarCajon2 = false;
        abrirCajon2 = true;

    }

}
```

Asignación de teclas a utilizar para las cinco animaciones, movimiento de la cámara y también el encendido de las luces del interior.