UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

LAB. COMPUTACION GRAFICA E INTERACCION HUMANO-COMPUTADORA

SEMESTRE 2022-2

FINAL PROJECT

TECHNICAL MANUAL

419048901 - GPO04

PROFESOR ING. CARLOS ALDAIR ROMAN BALBUENA

27 / MAYO / 2022

CONTENTS

Introduction

Computer graphics is considered to be the field of visual computing, where computers are used both to generate visual images synthetically and to integrate or change visual and spatial information from the real world or from a fictional space.

It is divided into 2D and 3D graphics areas, where the 2D area considers vectors and raster graphics. While the 3D area considers vectors and matrices.

3D modeling is the technique used to create forms in third dimension through programs installed on a computer. Models or parts of a model can be created and then assembled to observe how they function as a single system.

Current software for graphics generation goes beyond just storing polygons in computer memory. Graphics result from techniques in the use of shading, texturing and rasterization (in reference to bitmaps).

An Application Programming Interface (API) is a set of codes that can be used to allow various applications to communicate with each other. It is something that performs a similar task to the user interface when it comes to fostering interaction between person and program, only, applied solely and exclusively within the software environment.

OpenGL is an API that provides us with a large set of functions that we can use to manipulate 3D and 2D graphics and images. Among the most outstanding features of OpenGL are the following:

- Reduce the complexity of interfacing with different graphics cards by presenting the programmer with a single, uniform API.

- Hide the different capabilities of the various hardware platforms by requiring all implementations to support the full OpenGL feature set (using software emulation if necessary).

- The basic operation of OpenGL is to accept primitive actions such as points, lines and polygons, and convert them to pixels. This process is performed by a graphics pipeline known as the OpenGL State Machine.

Computer Graphics Utilities

- ❖ Computer Graphics has multiplied its importance in robot training and machine vision.

- ❖ In scientific simulation, not only medical, there are also many job opportunities. It is fundamental in biology simulation, in astronomy, in basic physics, etc.

- ❖ This is also true in health, not only in visualization but also in the propagation of viruses, synthesis of medicines, etc.

- ❖ In population analysis, in animal populations, in ecosystem simulations, in thermodynamic simulations or in climatology.

Informative data

- The first major breakthrough in computer graphics was the development of Sketchpad in 1962 by Ivan Sutherland.

- 3D modeling is used in advertising agencies, the television industry and by artists and designers specializing in digital imaging.
-

Objectives

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

- To implement the cursor concepts related to modeling, texturing, ambience (lighting and shading) and animation.

- Performing a 3D recreation in OpenGL on a real or fictitious facade and space through reference images or sketches.

- That the student can retake his programming skills (C++) by applying it to a fundamental concept of the course.

Timeline

Gantt Diagram							
Activities	**April**		**May**				
	Wk1	Wk2	Wk1	Wk2	Wk3	Wk4	
Create models							
Texturing models							
Export objects by selection							
Import in VS							
Create setting							
Build animations							
User tests							
Documentation							
Delivery							

WK1 - April	19/04/2022	-	22/04/2022
WK2 - April	25/04/2022	-	29/04/2022
WK1 - May	02/05/2022	-	06/05/2022
WK2 - May	09/05/2022	-	13/05/2022
WK3 - May	16/05/2022	-	20/05/2022
WK4 - May	23/05/2022	-	27/05/2022

Software and tools to use

Maya/Autodesk: 3D computer graphics development, special effects, animation and drawing.

Adobe Photoshop: image editor, photo retouching and graphics.

GIMP: editing of digital images in bitmap form, both drawings and photographs. It is a free and open source program.

GitHub: project storage using the Git version control system.

GitHub Desktop: application to interact with GitHub using a GUI instead of the command line or a web browser.

TurboSquid: digital media website offering stock 3D models used in 3D graphics (OBJ).

Mixamo: website offering 3D computer graphics through predefined characters and animations in collada (DAE).

Color Calculator: normalized RGB vector color calculator.

Visual Studio: IDE for Windows compatible with C++

Scope

Description

Select a facade and two spaces that can be real or fictitious and present reference images of these spaces for their 3D recreation in OpenGL.

In the reference images, 5 elements must be visualized in each space that the student is going to **recreate virtually and where these objects must be as close as possible to their reference image, as well as their setting.**

Fachada de Gimnasio

Two interior spaces

Elements to recreate in OpenGL are the following: Room 1 (Gymnasium)

- Dumbbell rack
- Shelf or object cabinet (accessories)
- Reclining benches
- Treadmill
- Swiss balls
- Multistation for pull-ups
- Weights / Bars

Room 2 (Box Club)

- Standing punching bags
- Round punching bags
- Training dolls
- Boxing ring
- Exhibition gloves

Note: The details to be considered are the **black/yellow** colors that will predominate in the interior and exterior design as well as in the elements and ambiance.

Resolution

C++ programming language and **Visual Studio Community 2022** source code editor.

A template with the most important headers will be used:

- ➔ GLEW.h (load and operate OpenGL extensions)
- ➔ GLFW.h (window/device utilities with OpenGL)
- ➔ stbImage.h (file formats PNG, GIF, etc.)
- ➔ GLM.h (mathematical operations)
- ➔ SOIL2.h (stb format extension for textures)
- ➔ Shader.h (load and operate Shaders)
- ➔ Camera.h (camera management)
- ➔ Model.h (load and operate models)
- ➔ Texture.h (load and operate textures)
- ➔ modelAnim.h (load and operate animations)

Modeling

In Maya we will work the tool through the creation of objects which we will work in detail with Modeling tools, having an organization in the Outliner that allows us to group objects and parts of objects in Groups that will serve us to export by selection (.OBJ) so that we can load models in OpenGL. Specifically exporting to the folders /Models/Ambiente and /Models/Gym.

Models such as the collage animation (.DAE) is created on the Mixamo site that allows you to choose a sports character and an abdominal exercise.

Texturing

In Maya in complement with GIMP (activate alpha channel) and Photoshop (texture creation) a snapshot of UV maps (.PNG) is generated for the models so that they can be edited and manipulated for the correct texturing with Lambert and Phong type materials with respect to lighting. The textures are stored with their respective models.

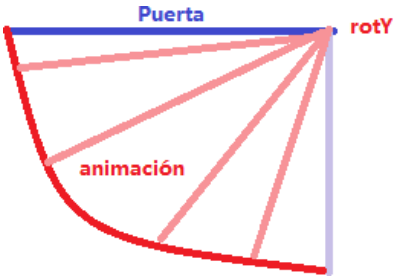
Lighting and shading

Shaders set up by the teacher have been modified to meet the design requirements. The lightShader for lighting and assigning transparency properties on objects as well as incident light; the lampShader for loading models as is without attribute extension; the SkyBoxShader for the main gymnasium scenario in the middle of a busy city; and finally, the animShader that allows loading and operating preset animations such as the example of the collado model (.DAE).

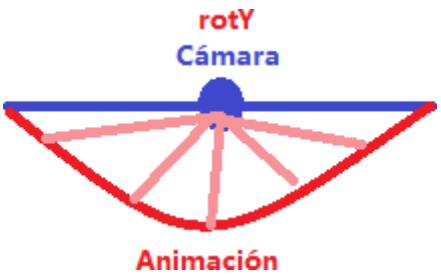
Animation

Animations centralizing pivots to the objects and thus achieving to move, rotate and scale their values, increasing and/or subtracting according to the desired action.

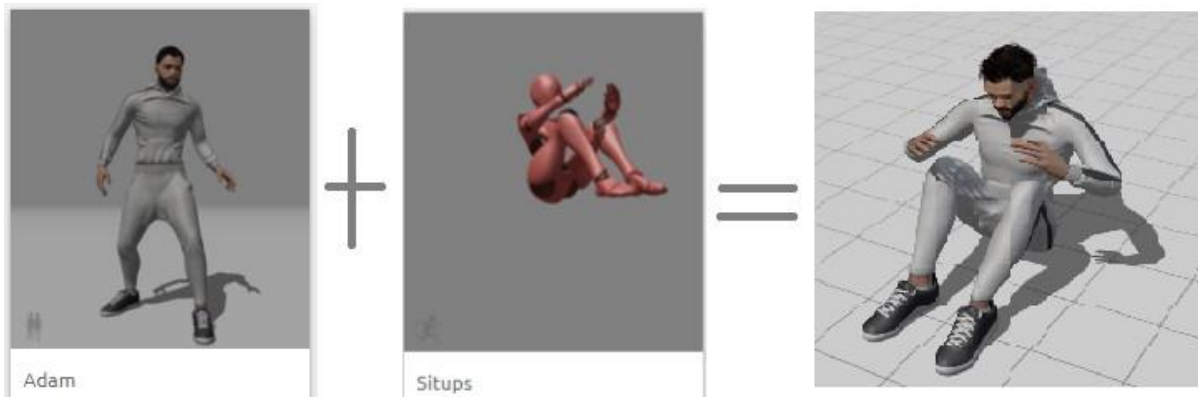
Animation 1: A door that opens outwards is to be configured with the "F" key that by means of a Boolean variable allows to identify the action and through a general function to manipulate the rotation (rotDoor) of the door.



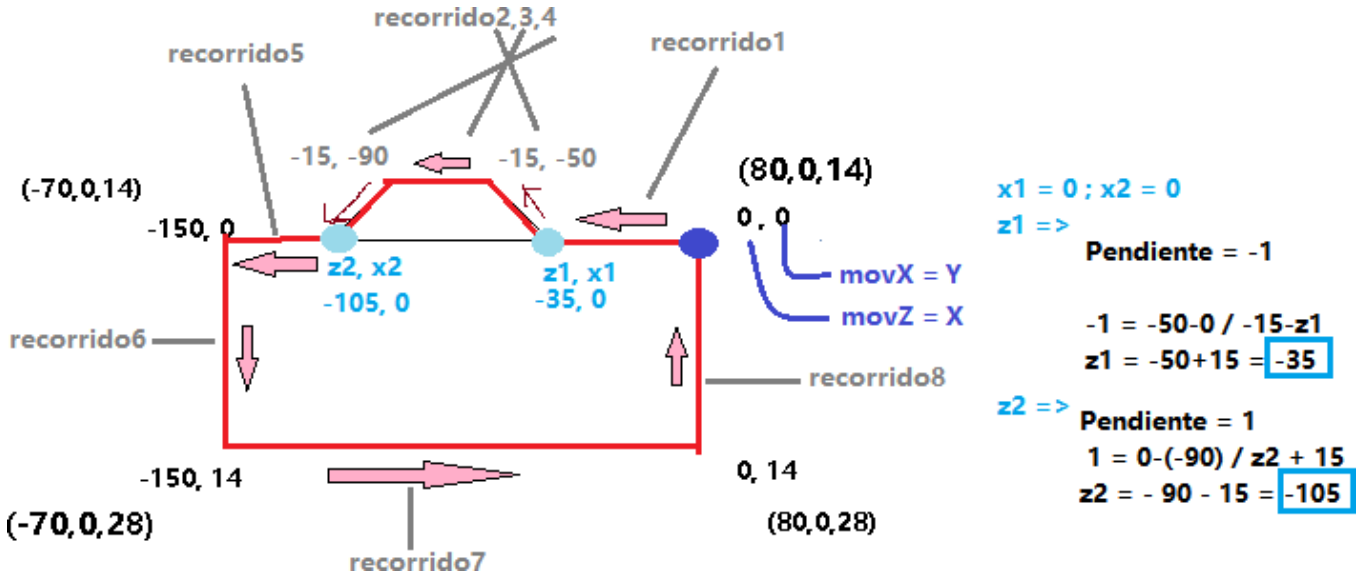
Animation 2: Two security cameras monitoring the set area, a permanent activation is intended within the general animation function that allows to manipulate the rotation (rotCam) of the camera from -90° to 90°.



Animation 3: Collado model (.DAE) exported in MIXAMO of the sports character performing sit-ups with frames set and loaded in AnimShader.



Animation 4: Sports car ride providing arrival and departure ambience through the streets outside the gymnasium approaching the entrance and rejoining its normal traffic, this in a cycle.



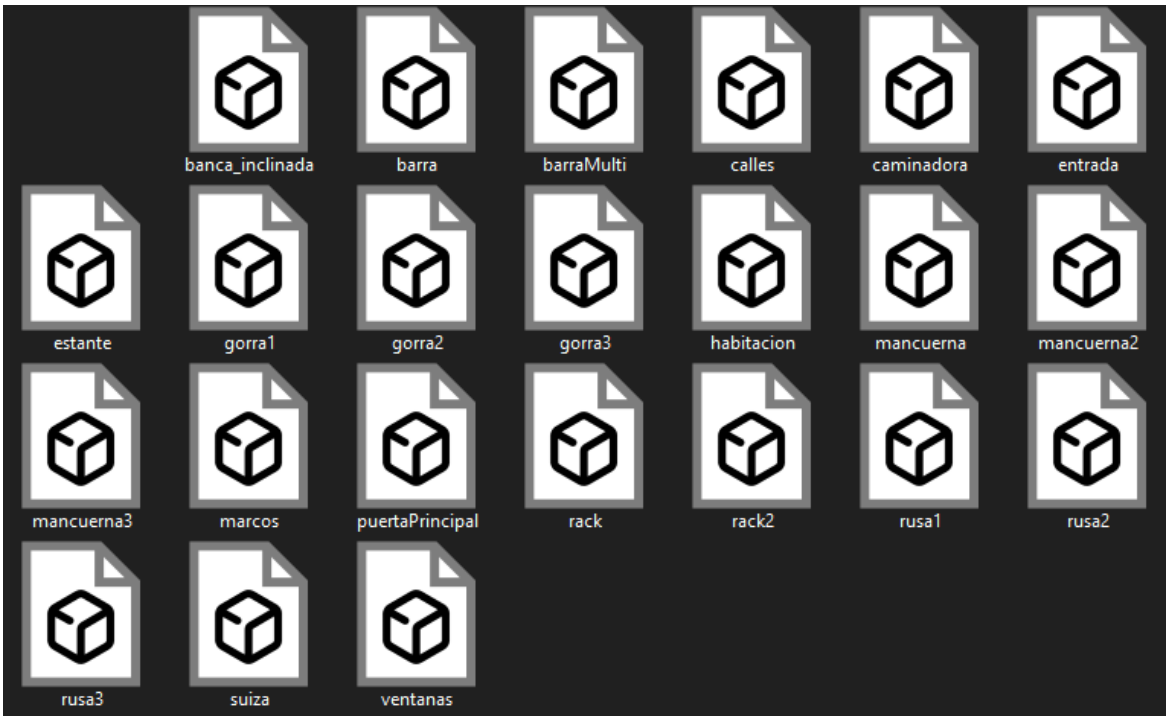
Archives

Modeling

Divided into folders to distinguish between the different components of the recreation of virtual spaces, we have the gym, the boxing club and the outdoor environment.

Files stored in /Models/Ambiente

Files stored in /Models/Gym

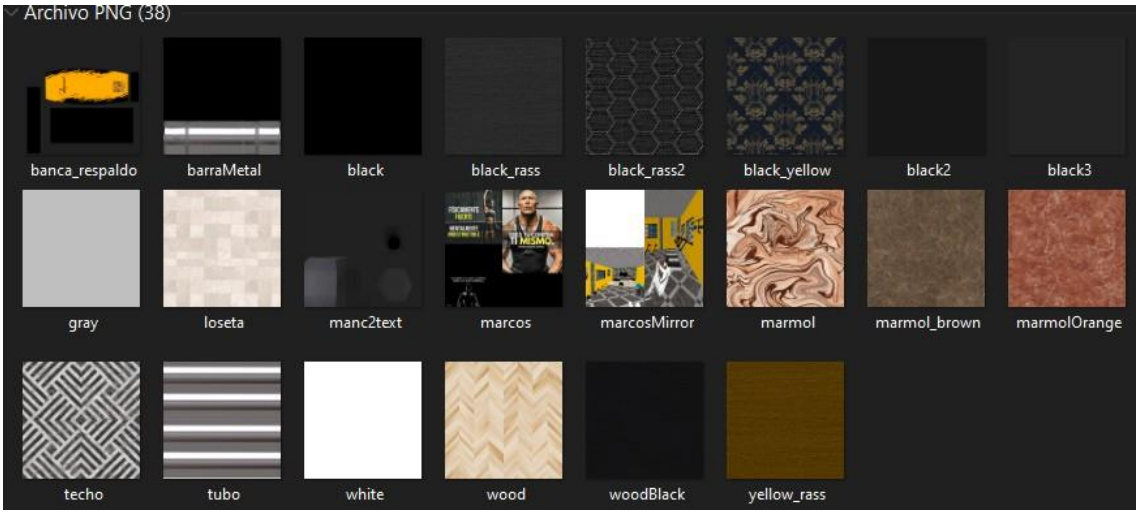


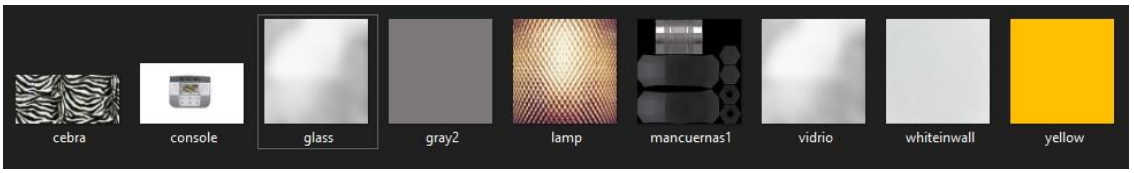
Files stored in /Models/Box

Texturing

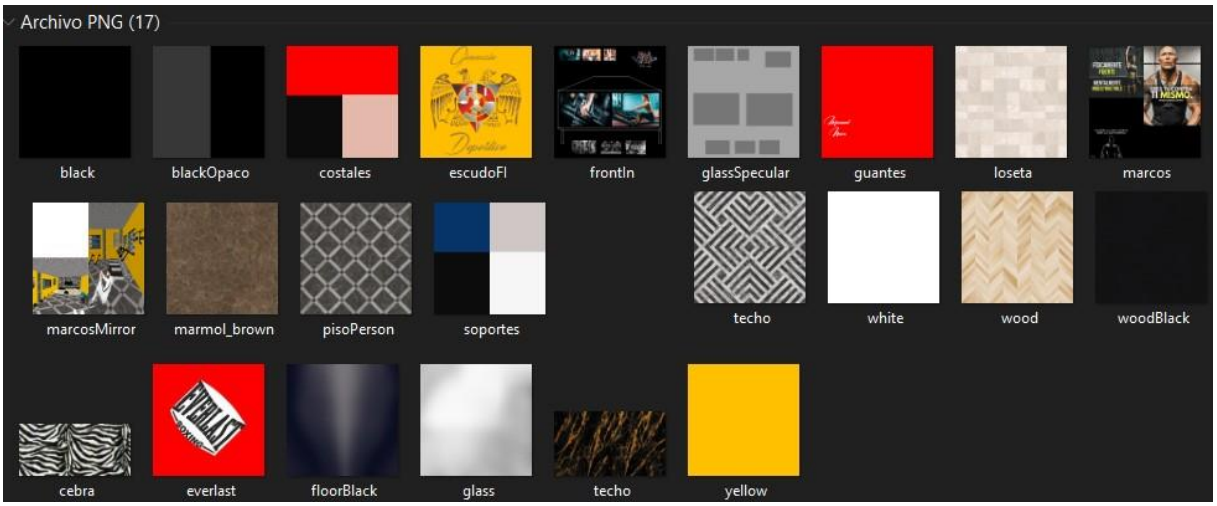
Files stored in /Models/Ambiente

Files stored in /Models/Gym



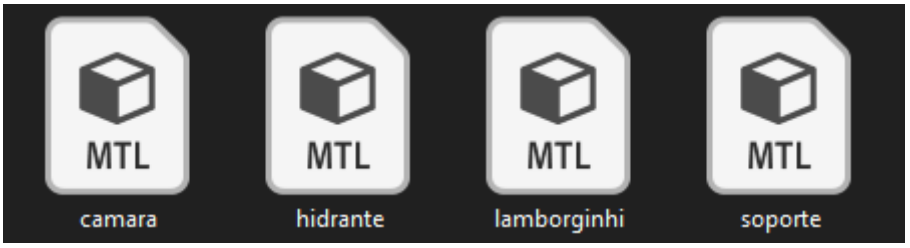


Files stored in /Models/Box

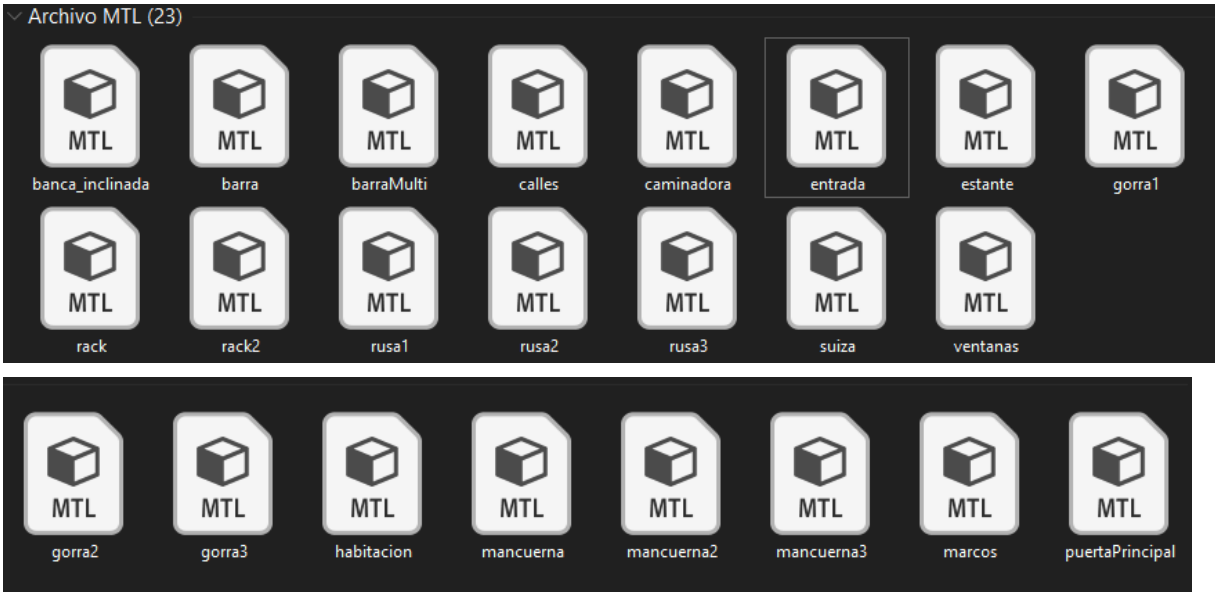


Lighting and shading

Files stored in /Models/Ambiente



Files stored in /Models/Gym



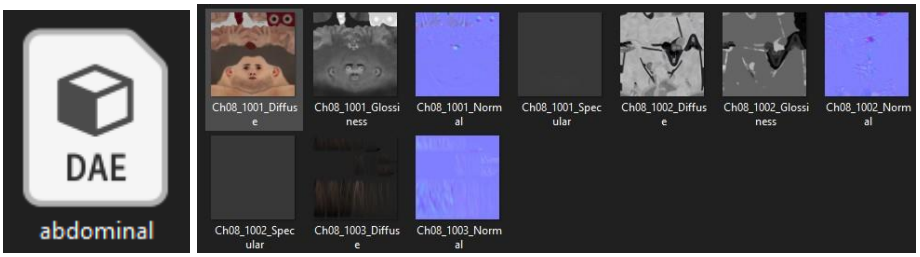
Files stored in /Models/Box

Animation

For the door animation we used the MainDoor OBJ, for the sports car animation we used the lamborginhi OBJ.

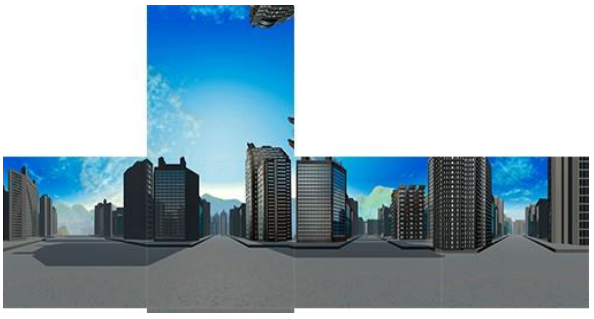
Security camera animation was supplemented in two OBJ support and camera.

Collado animation (.DAE) located in /Animations with their respective /textures.



Environment

For ambiance, the hydrant was added to the streets and the SkyBox located in /SkyBox complements the gym in the exterior design.
Images found and assembled into a .TGA file in Photoshop.



Limitations

In carrying out the practice, certain details were omitted that are difficult to comply with according to the established time and even due to the limitation in the study of the modeling topics. Below I will list those that stand out the most:

- Shading using directional light (I used flat light).

- Mirror effect (covered with images).
- Reflex effects (complementary to the previous one).
- Windows superimposed on the SkyBox (covered with low transparency).
- Animations with preloaded KeyFrames to improve performance.
- Poor outdoor ambience due to overload of animation per hill.
- Limited texturing on gym walls due to poor implementation between tinkercad software (temporary) and Maya.
- Own interaction animations with the main elements recreated due to their complexity.

Documentation

Project Highlights Variables (419048901_Project_GPO04)

Synthetic Camera Handling

camera: vector with X, Y and Z values of initial camera view position
lastX: initial horizontal camera screen position
lastY: initial vertical camera display position

keys: define the use of 1024 keys for use in the program
firstMouse: indicates the occupation of the mouse for use in the program.

Door animation

rotDoor: actual value of rotation angle of main door of the gymnasium
actionDoor: indicates whether the action of opening the main door of the gym
openDoor: indicates the open/closed status of the main door of the gym

Security camera animation

rotCam: actual value of security camera rotation angle
camDerecha: indicates right/left status of the security camera

Virtual car animation

posIniciCar: vector with X, Y and Z values of initial position of the virtual
movKitXY: actual value of the virtual carriage movement in X and Y
rotKit: actual value of virtual carriage rotation angle
circuito: indicates whether the virtual carriage travel action is performed.
recorridos1-8: indicates the route statusa of the virtual route of the virtual trolley

Character animation (personal trainer

posIniPerson: vector with X, Y and Z values of the trainer's initial position.

Documentation

Code + notes

...yectoFinal\ProyectoFinal\419048901_Proyecto_Gpo04.cpp 1

```cpp
/**
 * @file 419048901_Proyecto_GPO04.cpp
 * @brief Archivo principal CPP (main program) del proyecto
 * @author NumCuenta: 419048901
 * @date 11/05/2022
 */

// Operaciones E/S
#include <iostream>

// Oeraciones Matematicas
#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"

// 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();

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

// Camera
```

We load libraries including those of management Shaders, Camera, Texture, Patterns, Animations

```
49 Camera  camera(glm::vec3(0.0f, 10.0f, 25.0f));
50 GLfloat lastX = WIDTH / 2.0;
51 GLfloat lastY = HEIGHT / 2.0;
52 bool keys[1024];
53 bool firstMouse = true;
54
55
56 // Light attributes
57 glm::vec3 lightPos(0.0f, 0.0f, 0.0f);
58 glm::vec3 PosIni(-16.0f, 1.0f, -70.0f);
59 glm::vec3 lightDirection(0.0f, -1.0f, -1.0f);
60 bool active;
61
62 bool encendido = false;
63 // Positions of the point lights
64 glm::vec3 pointLightPositions[] = {
65     glm::vec3(0.0f, 19.0f, 0.0f)
66 };
67
68 // Position of the SpotLight
69 glm::vec3 spotLightPosition = glm::vec3(0.0f, 19.0f, 0.0f);
70
71 int dir = 0;
72 // Directions of the SpotLight
73 glm::vec3 spotLightDir[] = {
74     glm::vec3(0.0f,-1.0f, 0.0f),  // Abajo
75     glm::vec3(1.0f,0.0f, 0.0f),  // Derecha
76     glm::vec3(0.0f,0.0f, -1.0f), // Atras
77     glm::vec3(-1.0f,0.0f, 0.0f), // Izquierda
78     glm::vec3(0.0f,0.0f, 1.0f),  // Frente
79     glm::vec3(0.0f,1.0f, 0.0f),  // Arriba
80     glm::vec3(0.0f,-1.0f, 0.0f)  // Abajo
81 };
82
83 float vertices[] = {
84      -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
85         0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
86         0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
87         0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
88        -0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
89        -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
90
91        -0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
92         0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
93         0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
94         0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
95        -0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
96        -0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
97
```

```
 98         -0.5f,  0.5f,  0.5f, -1.0f,  0.0f,  0.0f,
 99         -0.5f,  0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
100         -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
101         -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
102         -0.5f, -0.5f,  0.5f, -1.0f,  0.0f,  0.0f,
103         -0.5f,  0.5f,  0.5f, -1.0f,  0.0f,  0.0f,
104
105          0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f,
106          0.5f,  0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
107          0.5f, -0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
108          0.5f, -0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
109          0.5f, -0.5f,  0.5f,  1.0f,  0.0f,  0.0f,
110          0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f,
111
112         -0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,
113          0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,
114          0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
115          0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
116         -0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
117         -0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,
118
119         -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,
120          0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,
121          0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
122          0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
123         -0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
124         -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f
125 };
126
127 GLfloat skyboxVertices[] = {
128     // Positions
129     -1.0f,  1.0f, -1.0f,
130     -1.0f, -1.0f, -1.0f,
131     1.0f, -1.0f, -1.0f,
132     1.0f, -1.0f, -1.0f,
133     1.0f,  1.0f, -1.0f,
134     -1.0f,  1.0f, -1.0f,
135
136     -1.0f, -1.0f,  1.0f,
137     -1.0f, -1.0f, -1.0f,
138     -1.0f,  1.0f, -1.0f,
139     -1.0f,  1.0f, -1.0f,
140     -1.0f,  1.0f,  1.0f,
141     -1.0f, -1.0f,  1.0f,
142
143     1.0f, -1.0f, -1.0f,
144     1.0f, -1.0f,  1.0f,
145     1.0f,  1.0f,  1.0f,
146     1.0f,  1.0f,  1.0f,
```

```
147      1.0f,  1.0f, -1.0f,
148      1.0f, -1.0f, -1.0f,
149
150      -1.0f, -1.0f,  1.0f,
151      -1.0f,  1.0f,  1.0f,
152      1.0f,  1.0f,  1.0f,
153      1.0f,  1.0f,  1.0f,
154      1.0f, -1.0f,  1.0f,
155      -1.0f, -1.0f,  1.0f,
156
157      -1.0f,  1.0f, -1.0f,
158      1.0f,  1.0f, -1.0f,
159      1.0f,  1.0f,  1.0f,
160      1.0f,  1.0f,  1.0f,
161      -1.0f,  1.0f,  1.0f,
162      -1.0f,  1.0f, -1.0f,
163
164      -1.0f, -1.0f, -1.0f,
165      -1.0f, -1.0f,  1.0f,
166      1.0f, -1.0f, -1.0f,
167      1.0f, -1.0f, -1.0f,
168      -1.0f, -1.0f,  1.0f,
169      1.0f, -1.0f,  1.0f
170  };
171
172  glm::vec3 Light1 = glm::vec3(0);
173  glm::vec3 Light2 = glm::vec3(0);
174  glm::vec3 Light3 = glm::vec3(0);
175  glm::vec3 Light4 = glm::vec3(0);
176
177  /**
178   * \var rotDor, actionDoor, openDoor
179   * \brief Variables Animación de Puerta
180   */
181  float rotDoor = 0.0f;
182  bool actionDoor = false, openDoor = false;
183
184  /**
185   * \var rotCam, CamDerecha
186   * \brief Variables Animación de Camara Seguridad
187   */
188  float rotCam = 0.0;
189  bool CamDerecha = false;
190
191  /**
192   * \var posIniCar, movKitXY, rotKit, circuito, recorridos1-8
193   * \brief Variables Animación del coche
194   */
195  glm::vec3 PosIniCar(80.0f, 0.0f, 14.0f);
```

```
196 float movKitX = 0.0;
197 float movKitZ = 0.0;
198 float rotKit  = 0.0;
199
200 bool circuito = false;
201 bool recorrido1 = true; bool recorrido2 = false; bool recorrido3 = false;
      bool recorrido4 = false;
202 bool recorrido5 = false;bool recorrido6 = false; bool recorrido7 = false;
      bool recorrido8 = false;
203
204 /**
205  * \var posIniPerson
206  * \brief Variable Animación del Personaje
207  */
208 glm::vec3 PosIniPerson(-16.0f, 0.0f, -70.0f);
209
210 // Deltatime
211 GLfloat deltaTime = 0.0f;   // Time between current frame and last frame
212 GLfloat lastFrame = 0.0f;   // Time of last frame
213
214
215 /**
216  * \fn int main()
217  * \brief Funcion del programa principal
218  * \return Devuelve 0 de programa exitoso
219  */
220 int main()
221 {
222     // Init GLFW
223     glfwInit();
224
225     // Set all the required options for GLFW
226     /*glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
227     glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
228     glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
229     glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
230     glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);*/
231
232     // Create a GLFWwindow object that we can use for GLFW's functions
233     GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "Proyecto
        Gimnasio \"FORM\" :  419048901 - GPO04", nullptr, nullptr);
234
235     if (nullptr == window)
236     {
237         std::cout << "Failed to create GLFW window" << std::endl;
238         glfwTerminate();
239
240         return EXIT_FAILURE;
241     }
```

```
242
243     glfwMakeContextCurrent(window);
244     glfwGetFramebufferSize(window, &SCREEN_WIDTH, &SCREEN_HEIGHT);
245
246     // Set the required callback functions
247     glfwSetKeyCallback(window, KeyCallback);
248     glfwSetCursorPosCallback(window, MouseCallback);
249
250     // GLFW Options
251     //glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);
252
253     // Set this to true so GLEW knows to use a modern approach to
        retrieving function pointers and extensions
254     glewExperimental = GL_TRUE;
255
256     // Initialize GLEW to setup the OpenGL Function pointers
257     if (GLEW_OK != glewInit())
258     {
259         std::cout << "Failed to initialize GLEW" << std::endl;
260         return EXIT_FAILURE;
261     }
262
263     // Define the viewport dimensions
264     glViewport(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT);
265
266     // Carga de Shaders
267     Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting.frag");
268     Shader lampShader("Shaders/lamp.vs", "Shaders/lamp.frag");
269     Shader SkyBoxshader("Shaders/SkyBox.vs", "Shaders/SkyBox.frag");
270     Shader animShader("Shaders/anim.vs", "Shaders/anim.frag");
271
272     // Carga de modelos de gimnasio
273     Model Piso((char*)"Models/Gym/calles.obj");
274     Model Habitacion((char*)"Models/Gym/habitacion.obj");
275     Model Entrada((char*)"Models/Gym/entrada.obj");
276     Model Puerta((char*)"Models/Gym/puertaPrincipal.obj");
277     Model Estante((char*)"Models/Gym/estante.obj");
278     Model Ventanas((char*)"Models/Gym/ventanas.obj");
279     Model Marcos((char*)"Models/Gym/marcos.obj");
280     Model Hidrante((char*)"Models/Ambiente/hidrante.obj");
281
282     Model Banca_inclinada((char*)"Models/Gym/banca_inclinada.obj");
283     Model Caminadora((char*)"Models/Gym/caminadora.obj");
284     Model Barra((char*)"Models/Gym/barra.obj");
285     Model Rack((char*)"Models/Gym/rack.obj");
286     Model Rack2((char*)"Models/Gym/rack2.obj");
287     Model Multi((char*)"Models/Gym/barraMulti.obj");
288
289     Model Mancuerna((char*)"Models/Gym/mancuerna.obj");
```

We load the models to use through his res- relative path perspective

```
290     Model Mancuerna2((char*)"Models/Gym/mancuerna2.obj");
291     Model Mancuerna3((char*)"Models/Gym/mancuerna3.obj");
292     Model Rusa1((char*)"Models/Gym/rusa1.obj");
293     Model Rusa2((char*)"Models/Gym/rusa2.obj");
294     Model Rusa3((char*)"Models/Gym/rusa3.obj");
295     Model Suiza((char*)"Models/Gym/suiza.obj");
296     Model Gorra1((char*)"Models/Gym/gorra1.obj");
297     Model Gorra2((char*)"Models/Gym/gorra2.obj");
298     Model Gorra3((char*)"Models/Gym/gorra3.obj");
299
300     // Carga de modelos de animación
301     ModelAnim animacionPersonaje("Animaciones/abdominal.dae");
302     Model Carro((char*)"Models/Ambiente/lamborginhi.obj");
303     Model Soporte((char*)"Models/Ambiente/soporte.obj");
304     Model Camara((char*)"Models/Ambiente/camara.obj");
305
306
307     // First, set the container's VAO (and VBO)
308     GLuint VBO, VAO;
309     glGenVertexArrays(1, &VAO);
310     glGenBuffers(1, &VBO);
311     glBindVertexArray(VAO);
312     glBindBuffer(GL_ARRAY_BUFFER, VBO);
313     glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices,
          GL_STATIC_DRAW);
314     // Position attribute
315     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat),
          (GLvoid*)0);
316     glEnableVertexAttribArray(0);
317     // normal attribute
318     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float),
          (void*)(3 * sizeof(float)));
319     glEnableVertexAttribArray(1);
320
321     // Set texture units
322     lightingShader.Use();
323     glUniform1i(glGetUniformLocation(lightingShader.Program,
          "material.diffuse"), 0);
324     glUniform1i(glGetUniformLocation(lightingShader.Program,
          "material.specular"),1);
325
326     // SkyBox attributes
327     GLuint skyboxVBO, skyboxVAO;
328     glGenVertexArrays(1, &skyboxVAO);
329     glGenBuffers(1, &skyboxVBO);
330     glBindVertexArray(skyboxVAO);
331     glBindBuffer(GL_ARRAY_BUFFER, skyboxVBO);
332     glBufferData(GL_ARRAY_BUFFER, sizeof(skyboxVertices), &skyboxVertices,
          GL_STATIC_DRAW);
```

We create the arrangements of VBO/VAO and their initial attributes

```
333     glEnableVertexAttribArray(0);
334     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat),
          (GLvoid*)0);
335
336     // Load textures
337     vector<const GLchar*> faces;
338     faces.push_back("SkyBox/right.tga");
339     faces.push_back("SkyBox/left.tga");
340     faces.push_back("SkyBox/top.tga");
341     faces.push_back("SkyBox/bottom.tga");
342     faces.push_back("SkyBox/back.tga");
343     faces.push_back("SkyBox/front.tga");
344
345     GLuint cubemapTexture = TextureLoading::LoadCubemap(faces);
346
347     // Load matrix Projection
348     glm::mat4 projection = glm::perspective(camera.GetZoom(), (GLfloat)
          SCREEN_WIDTH / (GLfloat)SCREEN_HEIGHT, 0.1f, 100.0f);
349
350
351     // Game loop
352     while (!glfwWindowShouldClose(window))
353     {
354
355         // Calculate deltatime of current frame
356         GLfloat currentFrame = glfwGetTime();
357         deltaTime = currentFrame - lastFrame;
358         lastFrame = currentFrame;
359
360         // Check if any events have been activiated (key pressed, mouse
              moved etc.) and call corresponding response functions
361         glfwPollEvents();
362         DoMovement();
363         animacion();
364
365         // Clear the colorbuffer
366         glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
367         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
368
369         // OpenGL options
370         glEnable(GL_DEPTH_TEST);
371
372
373         // Use cooresponding shader when setting uniforms/drawing objects
374         /* -------- Lighting Shader --------*/
375         lightingShader.Use();
376         GLint viewPosLoc = glGetUniformLocation(lightingShader.Program,
              "viewPos");
377         glUniform3f(viewPosLoc, camera.GetPosition().x, camera.GetPosition
```

We load the faces of the SkyBox, that is, its 6 textures

```
            ().y, camera.GetPosition().z);
378
379         // Directional light
380         glUniform3f(glGetUniformLocation(lightingShader.Program,
              "dirLight.direction"), 0.2f, -1.0f, -0.3f);
381         glUniform3f(glGetUniformLocation(lightingShader.Program,
              "dirLight.ambient"), 0.45f,0.45f,0.45f);  // Luz ambiente +
              DiffuseModify
382         glUniform3f(glGetUniformLocation(lightingShader.Program,
              "dirLight.diffuse"), 0.1f, 0.1f, 0.1f);
383         glUniform3f(glGetUniformLocation(lightingShader.Program,
              "dirLight.specular"), 0.35f, 0.35f, 0.35f);
384
385         // Point light
386         glUniform3f(glGetUniformLocation(lightingShader.Program,
              "pointLights[0].position"), pointLightPositions[0].x,
              pointLightPositions[0].y, pointLightPositions[0].z);
387         glUniform3f(glGetUniformLocation(lightingShader.Program,
              "pointLights[0].ambient"), 1.0f, 1.0f, 1.0f);
388         glUniform3f(glGetUniformLocation(lightingShader.Program,
              "pointLights[0].diffuse"), 1.0f, 1.0f, 1.0f);
389         glUniform3f(glGetUniformLocation(lightingShader.Program,
              "pointLights[0].specular"), 1.0f, 1.0f, 1.0f);
390         glUniform1f(glGetUniformLocation(lightingShader.Program,
              "pointLights[0].constant"), 1.0f);
391         glUniform1f(glGetUniformLocation(lightingShader.Program,
              "pointLights[0].linear"), 0.045f);
392         glUniform1f(glGetUniformLocation(lightingShader.Program,
              "pointLights[0].quadratic"), 0.0075f);
393
394         // SpotLight GIANT
395         glUniform3f(glGetUniformLocation(lightingShader.Program,
              "spotLight.position"), spotLightPosition.x, spotLightPosition.y,
               spotLightPosition.z);
396         glUniform3f(glGetUniformLocation(lightingShader.Program,
              "spotLight.direction"), spotLightDir[dir].x, spotLightDir
              [dir].y, spotLightDir[dir].z);
397         glUniform3f(glGetUniformLocation(lightingShader.Program,
              "spotLight.ambient"), 0.05f, 0.05f, 0.05f);
398         glUniform3f(glGetUniformLocation(lightingShader.Program,
              "spotLight.diffuse"), 0.2f, 0.2f, 0.2f);
399         glUniform3f(glGetUniformLocation(lightingShader.Program,
              "spotLight.specular"),0.05f, 0.05f, 0.05f);
400         glUniform1f(glGetUniformLocation(lightingShader.Program,
              "spotLight.constant"), 1.0f);
401         glUniform1f(glGetUniformLocation(lightingShader.Program,
              "spotLight.linear"), 0.045f);
402         glUniform1f(glGetUniformLocation(lightingShader.Program,
              "spotLight.quadratic"), 0.0075f);
```

```
403         glUniform1f(glGetUniformLocation(lightingShader.Program,
              "spotLight.cutOff"), glm::cos(glm::radians(12.5f)));
404         glUniform1f(glGetUniformLocation(lightingShader.Program,
              "spotLight.outerCutOff"), glm::cos(glm::radians(15.0f)));
405
406         // Set material properties
407         glUniform1f(glGetUniformLocation(lightingShader.Program,
              "material.shininess"), 32.0f);
408
409         // Create camera transformations
410         glm::mat4 view = camera.GetViewMatrix();
411
412         // Get the uniform locations
413         GLint modelLoc = glGetUniformLocation(lightingShader.Program,
              "model");
414         GLint viewLoc = glGetUniformLocation(lightingShader.Program,
              "view");
415         GLint projLoc = glGetUniformLocation(lightingShader.Program,
              "projection");
416
417         // Pass the matrices to the shader
418         glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
419         glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr
              (projection));
420
421         // Obtener matriz de Vista
422         view = camera.GetViewMatrix();
423
424         // Operar y dibujar modelo de PISO
425         glm::mat4 model(1);
426         model = glm::mat4(1);
427         model = glm::translate(model, glm::vec3(0.0f, 0.8f, 0.0f));
428         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
429         glUniform1i(glGetUniformLocation(lightingShader.Program,
              "activaTransparencia"), 0);
430         Piso.Draw(lightingShader);
431
432         // Operar y dibujar modelo de HIDRANTE
433         model = glm::mat4(1);
434         model = glm::translate(model, glm::vec3(0.0f, 0.0f, 5.0f));
435         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
436         Hidrante.Draw(lightingShader);
437
438         // Operar y dibujar modelo de HABITACION GIMNASIO
439         model = glm::mat4(1);
440         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
441         Habitacion.Draw(lightingShader);
442
443         // Operar y dibujar modelos (ELEMENTO) Bancas inclinadas
```

We put arrays of models at 1's
To start operations basics about them

```
444         model = glm::mat4(1);
445         model = glm::translate(model, glm::vec3(13.0f, 0.0f, -41.0f));
446         model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f,
              -1.0f, 0.0f));
447         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
448         Banca_inclinada.Draw(lampShader);
449
450         model = glm::mat4(1);
451         model = glm::translate(model, glm::vec3(13.0f, 0.0f, -33.0f));
452         model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f,
              -1.0f, 0.0f));
453         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
454         Banca_inclinada.Draw(lampShader);
455
456         // Operar y dibujar modelos (ELEMENTO) Caminadoras
457         model = glm::mat4(1);
458         model = glm::translate(model, glm::vec3(22.0f, 0.0f, -68.0f));
459         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
460         Caminadora.Draw(lampShader);
461
462         model = glm::mat4(1);
463         model = glm::translate(model, glm::vec3(13.0f, 0.0f, -68.0f));
464         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
465         Caminadora.Draw(lampShader);
466
467         model = glm::mat4(1);
468         model = glm::translate(model, glm::vec3(4.0f, 0.0f, -68.0f));
469         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
470         Caminadora.Draw(lampShader);
471
472         model = glm::mat4(1);
473         model = glm::translate(model, glm::vec3(-5.0f, 0.0f, -68.0f));
474         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
475         Caminadora.Draw(lampShader);
476
477         // Operar y dibujar modelos (ELEMENTO) Barras
478         model = glm::mat4(1);
479         model = glm::translate(model, glm::vec3(-20.0f, 0.0f, -64.0f));
480         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
481         Barra.Draw(lampShader);
482
483         model = glm::mat4(1);
484         model = glm::translate(model, glm::vec3(-28.0f, 0.0f, -53.0f));
485         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
486         Barra.Draw(lampShader);
487
488         // Operar y dibujar modelos (ELEMENTO) Racks mancuernas
489         model = glm::mat4(1);
490         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
```

```
491         Rack.Draw(lampShader);
492
493         model = glm::mat4(1);
494         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
495         Rack2.Draw(lampShader);
496
497         // Operar y dibujar modelos (ELEMENTO) Mancuernas
498         model = glm::mat4(1);
499         model = glm::translate(model, glm::vec3(3.0f, 0.3f, 0.0f));
500         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
501         Mancuerna.Draw(lampShader);
502         model = glm::mat4(1);
503         model = glm::translate(model, glm::vec3(-4.0f, 0.3f, 0.0f));
504         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
505         Mancuerna.Draw(lampShader);
506         model = glm::mat4(1);
507         model = glm::translate(model, glm::vec3(-3.0f, 0.3f, 0.5f));
508         model = glm::rotate(model, glm::radians(15.0f), glm::vec3(0.0f,
              -1.0f, 0.0f));
509         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
510         Mancuerna.Draw(lampShader);
511
512         // Operar y dibujar modelo Estante
513         model = glm::mat4(1);
514         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
515         Estante.Draw(lightingShader);
516
517         // Operar y dibujar modelo (ELEMENTO) Multiejercicios - Fondos
518         model = glm::mat4(1);
519         model = glm::translate(model, glm::vec3(-12.0f, 9.0f, -88.5f));
520         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
521         Multi.Draw(lightingShader);
522
523         // Operar y dibujar modelos (ELEMENTO) Pesas Rusas
524         model = glm::mat4(1);
525         model = glm::translate(model, glm::vec3(-21.0f, 0.0f, -42.0f));
526         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
527         Rusa1.Draw(lightingShader);
528         model = glm::mat4(1);
529         model = glm::translate(model, glm::vec3(-23.0f, 0.0f, -51.0f));
530         model = glm::rotate(model, glm::radians(30.0f), glm::vec3(0.0f,
              1.0f, 0.0f));
531         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
532         Rusa1.Draw(lightingShader);
533
534         model = glm::mat4(1);
535         model = glm::translate(model, glm::vec3(-21.0f, 0.0f, -52.0f));
536         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
537         Rusa2.Draw(lightingShader);
```

```
538         model = glm::mat4(1);
539         model = glm::translate(model, glm::vec3(-20.2f, 0.0f, -55.0f));
540         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
541         Rusa2.Draw(lightingShader);
542
543         model = glm::mat4(1);
544         model = glm::translate(model, glm::vec3(-21.0f, 0.0f, -62.0f));
545         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
546         Rusa3.Draw(lightingShader);
547
548         // Operar y dibujar modelos (ELEMENTO) Pelotas Suizas
549         model = glm::mat4(1);
550         model = glm::translate(model, glm::vec3(-15.0f, 0.0f, -26.5f));
551         model = glm::scale(model, glm::vec3(1.5f, 1.5f, 1.5f));
552         model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f,
              1.0f, 0.0f));
553         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
554         Suiza.Draw(lightingShader);
555
556         model = glm::mat4(1);
557         model = glm::translate(model, glm::vec3(-19.0f, 0.0f, -25.2f));
558         model = glm::scale(model, glm::vec3(1.5f, 1.5f, 1.5f));
559         model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f,
              1.0f, 0.0f));
560         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
561         Suiza.Draw(lightingShader);
562
563         // Operar y dibujar modelos de Marcos Interiores
564         model = glm::mat4(1);
565         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
566         Marcos.Draw(lightingShader);
567
568         // Operar y dibujar modelo de Entrada
569         model = glm::mat4(1);
570         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
571         Entrada.Draw(lightingShader);
572
573         // Operar y dibujar modelo de Entrada - Puerta
574         model = glm::mat4(1);
575         model = glm::translate(model, glm::vec3(12.3f, 1.4f, -31.7f));
576         model = glm::rotate(model, glm::radians( rotDoor ), glm::vec3
              (0.0f, 1.0f, 0.0f));
577         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
578         Puerta.Draw(lightingShader);
579
580         // Operar y dibujar modelos (ELEMENTO) Accesorios - Gorras
581         model = glm::mat4(1);
582         model = glm::translate(model, glm::vec3(0.0f, 4.12f, 3.4f));
583         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
```

```
584         Gorra1.Draw(lightingShader);
585
586         model = glm::mat4(1);
587         model = glm::translate(model, glm::vec3(0.0f, 4.12f, 3.5f));
588         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
589         Gorra2.Draw(lightingShader);
590
591         model = glm::mat4(1);
592         model = glm::translate(model, glm::vec3(0.0f, 4.12f, 3.7f));
593         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
594         Gorra3.Draw(lightingShader);
595
596         model = glm::mat4(1);
597         model = glm::translate(model, glm::vec3(9.0f, 6.0f, 13.5f));
598         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
599         Gorra3.Draw(lightingShader);
600
601         // Operar y dibujar modelo de Carro
602         model = glm::mat4(1);
603         model = glm::translate(model, PosIniCar + glm::vec3(movKitX, 0,
              movKitZ));
604         model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f,
              1.0f, 0.0));
605         model = glm::scale(model, glm::vec3(1.5f, 1.5f, 1.5f));
606         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
607         Carro.Draw(lightingShader);
608
609         // Operar y dibujar modelo de Soportes de Camaras
610         model = glm::mat4(1);
611         model = glm::translate(model, glm::vec3(-12.0f, 15.0f, -82.6f));
612         model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
613         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
614         Soporte.Draw(lightingShader);
615
616         model = glm::mat4(1);
617         model = glm::translate(model, glm::vec3(-8.0f, 16.0f, -13.0f));
618         model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
619         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
620         Soporte.Draw(lightingShader);
621
622         // Operar y dibujar modelo de Camaras
623         model = glm::mat4(1);
624         model = glm::translate(model, glm::vec3(-12.0f, 15.0f, -82.6f));
625         model = glm::rotate(model, glm::radians(rotCam), glm::vec3(0.0f,
              1.0f, 0.0));
626         model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
627         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
628         Camara.Draw(lightingShader);
629
```

In basic operations variables are placed allow you to change your position initial

```
630         model = glm::mat4(1);
631         model = glm::translate(model, glm::vec3(-8.0f, 16.0f, -13.0f));
632         model = glm::rotate(model, glm::radians(rotCam), glm::vec3(0.0f,
              1.0f, 0.0));
633         model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
634         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
635         Camara.Draw(lightingShader);
636
637         /* -------- Modelos con transparencia --------*/
638         glEnable(GL_BLEND);// Activa la funcionalidad para trabajar el
              canal alfa
639         glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
640
641         // Operar y dibujar modelo de Ventanas
642         model = glm::mat4(1);
643         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
644         glUniform1i(glGetUniformLocation(lightingShader.Program,
              "activaTransparencia"), 1);
645         glUniform4f(glGetUniformLocation(lightingShader.Program,
              "colorAlpha"), 0.0f, 0.0f, 0.0f, 0.05f);
646         Ventanas.Draw(lightingShader);
647
648         glDisable(GL_BLEND);  //Desactiva el canal alfa
649         glUniform4f(glGetUniformLocation(lightingShader.Program,
              "colorAlpha"), 1.0f, 1.0f, 1.0f, 1.0f);
650         glBindVertexArray(0);
651
652
653         // Also draw the lamp object, again binding the appropriate shader
654         //lampShader.Use();
655         //// Get location objects for the matrices on the lamp shader
              (these could be different on a different shader)
656         //modelLoc = glGetUniformLocation(lampShader.Program, "model");
657         //viewLoc = glGetUniformLocation(lampShader.Program, "view");
658         //projLoc = glGetUniformLocation(lampShader.Program,
              "projection");
659
660         //// Set matrices
661         //glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
662         //glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr
              (projection));
663         //model = glm::mat4(1);
664         //model = glm::translate(model, lightPos);
665         //model = glm::scale(model, glm::vec3(0.2f)); // Make it a smaller
              cube
666         //glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr
              (model));
667         //// Draw the light object (using light's vertex attributes)
668         ////for (GLuint i = 0; i < 1; i++)
```

```
669         ////{
670         ////    model = glm::mat4(1);
671         ////    model = glm::translate(model, pointLightPositions[i]);
672         ////    model = glm::scale(model, glm::vec3(0.2f)); // Make it a
              smaller cube
673         ////    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr
              (model));
674         ////    glBindVertexArray(VAO);
675         ////    glDrawArrays(GL_TRIANGLES, 0, 36);
676         ////}
677
678         //glBindVertexArray(0);
679
680
681         /* -------- Animacion Shader --------*/
682         /*_________ Personaje Animado (Abdominales)  __________*/
683         animacionPersonaje.initShaders(animShader.Program);
684         animShader.Use();
685         modelLoc = glGetUniformLocation(animShader.Program, "model");
686         viewLoc = glGetUniformLocation(animShader.Program, "view");
687         projLoc = glGetUniformLocation(animShader.Program, "projection");
688
689         glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
690         glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr
              (projection));
691
692         glUniform3f(glGetUniformLocation(animShader.Program,
              "material.specular"), 0.5f, 0.5f, 0.5f);
693         glUniform1f(glGetUniformLocation(animShader.Program,
              "material.shininess"), 12.0f);
694         glUniform3f(glGetUniformLocation(animShader.Program,
              "light.ambient"), 0.75f, 0.75f, 0.75f);
695         glUniform3f(glGetUniformLocation(animShader.Program,
              "light.diffuse"), 0.75f, 0.75f, 0.75f);
696         glUniform3f(glGetUniformLocation(animShader.Program,
              "light.specular"), 0.5f, 0.5f, 0.5f);
697         glUniform3f(glGetUniformLocation(animShader.Program,
              "light.direction"), 0.0f, -1.0f, -1.0f);
698         view = camera.GetViewMatrix();
699
700         model = glm::mat4(1);
701         model = glm::translate(model, glm::vec3(PosIniPerson.x,
              PosIniPerson.y, PosIniPerson.z));
702         model = glm::scale(model, glm::vec3(0.06f));// ESCALAR ANIMACION
              al 6%
703         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
704         animacionPersonaje.Draw(animShader);
705         glBindVertexArray(0);
706
```

```
707
708
709         /* -------- SKYBOX Shader --------*/
710         // Atributos SKYBOX
711         glDepthFunc(GL_LEQUAL);  // Change depth function so depth test
              passes when values are equal to depth buffer's content
712         SkyBoxshader.Use();
713         view = glm::mat4(glm::mat3(camera.GetViewMatrix()));    // Remove
              any translation component of the view matrix
714         glUniformMatrix4fv(glGetUniformLocation(SkyBoxshader.Program,
              "view"), 1, GL_FALSE, glm::value_ptr(view));
715         glUniformMatrix4fv(glGetUniformLocation(SkyBoxshader.Program,
              "projection"), 1, GL_FALSE, glm::value_ptr(projection));
716
717         // Dibujar SKYBOX
718         glBindVertexArray(skyboxVAO);
719         glActiveTexture(GL_TEXTURE1);
720         glBindTexture(GL_TEXTURE_CUBE_MAP, cubemapTexture);
721         glDrawArrays(GL_TRIANGLES, 0, 36);
722         glBindVertexArray(0);
723         glDepthFunc(GL_LESS); // Set depth function back to default
724
725
726         // Swap the screen buffers
727         glfwSwapBuffers(window);
728     }
729
730
731     // Terminate GLFW, clearing any resources allocated by GLFW.
732     glfwTerminate();
733     return 0;
734 }
735
736 /**
737  * \fn void DoMovement()
738  * \brief Modifica posiciones de Camara respecto a Entradas de Usuario
739  */
740 void DoMovement()
741 {
742
743     // Controles de Camara
744     if (keys[GLFW_KEY_W] || keys[GLFW_KEY_UP])
745     {
746         camera.ProcessKeyboard(FORWARD, deltaTime);
747     }
748
749     if (keys[GLFW_KEY_S] || keys[GLFW_KEY_DOWN])
750     {
751         camera.ProcessKeyboard(BACKWARD, deltaTime);
```

```
752     }
753
754     if (keys[GLFW_KEY_A] || keys[GLFW_KEY_LEFT])
755     {
756         camera.ProcessKeyboard(LEFT, deltaTime);
757     }
758
759     if (keys[GLFW_KEY_D] || keys[GLFW_KEY_RIGHT])
760     {
761         camera.ProcessKeyboard(RIGHT, deltaTime);
762     }
763
764
765     // Control de Animacion Puerta
766     if (keys[GLFW_KEY_F])
767     {
768         actionDoor = true;
769     }
770
771     // Control de Animacion Coche
772     if (keys[GLFW_KEY_Z])
773     {
774         circuito = true;
775     }
776
777     if (keys[GLFW_KEY_X])
778     {
779         circuito = false;
780     }
781
782
783 }
784
785
786 /**
787  * \fn void animacion()
788  * \brief Realiza animaciones de objetos, modificando las variables para
      operaciones basicas
789  */
790 void animacion()
791 {
792
793     //Movimiento de Camara Seguridad
794     rotCam += (CamDerecha) ? 0.3f : -0.3f ;
795     CamDerecha = (rotCam >= 90.0f) ? false : CamDerecha;
796     CamDerecha = (rotCam <= -90.0f) ? true : CamDerecha;
797
798
799     //Movimiento de Puerta
```

'F' Open/Close the door

'Z' move the car

'X' stop the car

Camera rotation will move in a way constant through the rotation of its angle about the y-axis

```
800     if (actionDoor) {
801         rotDoor += (openDoor) ? -0.8f : 0.8f ;
802         if (rotDoor <= 0.0f) {
803             openDoor = false;
804             actionDoor = false;
805         }
806         if (rotDoor >= 90.0f) {
807             openDoor = true;
808             actionDoor = false;
809         }
810     }
811
812
813     //Movimiento del coche
814     if (circuito)
815     {
816         if (recorrido1)
817         {
818             rotKit = 0.0f;
819             movKitX -= 0.2f;
820             if ( movKitX < -35.0f )
821             {
822                 recorrido1 = false;
823                 recorrido2 = true;
824             }
825         }
826
827         if (recorrido2)
828         {
829             rotKit = -45.0f;
830             movKitX -= 0.1f;
831             movKitZ -= 0.1f;
832             if ( movKitX < -50.0f && movKitZ < -15.0f )
833             {
834                 recorrido2 = false;
835                 recorrido3 = true;
836             }
837         }
838
839         if (recorrido3)
840         {
841             rotKit = 0.0f;
842             movKitX -= 0.05f;
843             if ( movKitX < -90.0f )
844             {
845                 recorrido3 = false;
846                 recorrido4 = true;
847             }
848         }
```

The car journey will be done according to animation resolution complex through movement ments in X, Z and their angle rotation about Y

```
849
850         if (recorrido4)
851         {
852             rotKit = 45.0f;
853             movKitX -= 0.1f;
854             movKitZ += 0.1f;
855             if ( movKitX < -105.0f && movKitZ > 0.0f )
856             {
857                 recorrido4 = false;
858                 recorrido5 = true;
859             }
860         }
861
862
863         if (recorrido5)
864         {
865             rotKit = 0.0f;
866             movKitX -= 0.2f;
867             if (movKitX < -150.0f)
868             {
869                 recorrido5 = false;
870                 recorrido6 = true;
871             }
872         }
873
874         if (recorrido6)
875         {
876             rotKit = 90.0f;
877             movKitZ += 0.2f;
878             if ( movKitZ > 14.0f )
879             {
880                 recorrido6 = false;
881                 recorrido7 = true;
882             }
883         }
884
885         if (recorrido7)
886         {
887             rotKit = 180.0f;
888             movKitX += 0.2f;
889             if ( movKitX > 0.0f )
890             {
891                 recorrido7 = false;
892                 recorrido8 = true;
893             }
894         }
895
896         if (recorrido8)
897         {
```

```
898             rotKit = -90.0f;
899             movKitZ -= 0.2f;
900             if (movKitZ < 0.0f)
901             {
902                 recorrido8 = false;
903                 movKitX = 0.0f;
904                 movKitZ = 0.0f;
905                 recorrido1 = true;
906             }
907         }
908 
909     }
910 
911 
912 }
913 
914 
915 /**
916  * \fn void KeyCallback()
917  * \brief Opera cada que se presiona/libera una tecla a través de GLFW
918  */
919 void KeyCallback(GLFWwindow *window, int key, int scancode, int action,
      int mode)
920 {
921     if (GLFW_KEY_ESCAPE == key && GLFW_PRESS == action)
922     {
923         glfwSetWindowShouldClose(window, GL_TRUE);
924     }
925 
926     if (key >= 0 && key < 1024)
927     {
928         if (action == GLFW_PRESS)
929         {
930             keys[key] = true;
931         }
932         else if (action == GLFW_RELEASE)
933         {
934             keys[key] = false;
935         }
936     }
937 
938 }
939 
940 
941 /**
942  * \fn void MouseCallback()
943  * \brief Procesa los movimientos del Mouse sobre la Camara en Ventana
      Principal
944  */
```

```
945 void MouseCallback(GLFWwindow *window, double xPos, double yPos)
946 {
947     if (firstMouse)
948     {
949         lastX = xPos;
950         lastY = yPos;
951         firstMouse = false;
952     }
953 
954     GLfloat xOffset = xPos - lastX;
955     GLfloat yOffset = lastY - yPos;  // Reversed since y-coordinates go
          from bottom to left
956 
957     lastX = xPos;
958     lastY = yPos;
959 
960     camera.ProcessMouseMovement(xOffset, yOffset);
961 }
```

Conclusion

The development of this project and its implementation served me as practice for modeling, texturing, shading, lighting and animation of 3D objects that allow me to recreate virtual spaces so that I can solve or represent real or fictitious situations. In this virtual space I use basic and complex operations to load models through 5 Shaders for the design and elaboration of lighting, shading, texturing and animation. In a deeper way the use of modules for the handling of synthetic camera and a visual representation that allows users or customers to use it properly.

The requirements and objectives were met with the help of C++ programming through the use of header files, shaders, models, classes, among many other structures and modules that allowed the complete development of this project. In addition, the use of specialized software for modeling and manipulation of 3D files in different formats (OBJ, DAE, MB, etc).

References

colaboradores de Wikipedia. (2020, 29 noviembre). Computación gráfica. Wikipedia, la enciclopedia libre. Recuperado 6 de mayo de 2022, de https://es.wikipedia.org/wiki/Computaci%C3%B3n_gr%C3%A1fica

Quispe, I. (2020, 1 septiembre). ¿Qué es el modelado 3D? Arcux. Recuperado 6 de mayo de 2022, de https://arcux.net/blog/que-es-el-modelado-3d/

Ríos, Y. (2019, 15 noviembre). OpenGL: qué es y para qué sirve. Profesional Review. Recuperado 22 de mayo de 2022, de https://www.profesionalreview.com/2019/11/15/opengl/

U. (2022, 29 abril). ¿Por qué estudiar Computación Gráfica? Centro Universitario de Tecnología y Arte Digital. Recuperado 22 de mayo de 2022, de https://u-tad.com/por-que-estudiar-computacion-grafica

Apuntes Ing. Carlos Aldair Roman Balbuena. Laboratorio de Computación Gráfica E Interacción Humano-Computadora. UNAM. Facultad de Ingeniería. [Consulta 22 de mayo del 2022]