

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO



SEMESTRE: 2021 - II



LAB. COMPUTACIÓN GRÁFICA E
INTERACCIÓN HUMANO-COMPUTADORA
ING. CARLOS ALDAIR ROMAN BALBUENA

GRUPO: 08

ALUMNO: MARCELO ROMERO ADOLFO

PROYECTO FINAL
MANUAL TÉCNICO

FECHA DE ENTREGA:
29/07/2021

Índice

Objetivos del proyecto:	2
Descripción:	2
Diagrama de Gant:	3
Alcance del proyecto:	3
Limitantes:	3
Documentación del código:	4
.....	4

Manual técnico

Objetivos del proyecto:

1. El alumno deberá aplicar y demostrar los conocimientos adquiridos durante todo el curso.
2. Representar en un ambiente virtual la casa icónica de la serie Rick y Morty desarrollando en Visual Studio el código que permita visualizar cada uno de los elementos propuestos para recrear dicho escenario, Además de presentar modelos con animación sujeta a las características de la serie.
3. El proyecto debe de funcionar de manera correcta según lo implementado en el respectivo repositorio git de descarga

Descripción:

El alumno deberá seleccionar una fachada y un espacio que pueden ser reales o ficticios y presentar imágenes de referencia de dichos espacios para su recreación 3D en OpenGL.

En la imagen de referencia se debe visualizar 7 objetos que el alumno va a recrear virtualmente y donde dichos objetos deben ser lo más parecido a su imagen de referencia, así como su ambientación.

Mi selección fue recrear la casa de Rick y Morty de la serie animada con el mismo con el mismo nombre, la cual fue realizada con la mayoría de los detalles planteados en la propuesta:



Diagrama de Gant:

Actividades	JUNIO		JULIO			
	Semana 3	Semana 4	Semana 1	Semana 2	Semana 3	Semana 4
Planeación de desarrollo de Proyecto						
Recreación del repositorio en GIT						
Búsqueda de modelos con licencia.						
Búsqueda e implementación de texturas						
Implementación de modelos al proyecto.						
Implementación de animaciones						
Pruebas						
Documentación del proyecto.						
Integración de proyecto al repositorio						

Alcance del proyecto:

El proyecto cumple con todas las características solicitadas, se lograron encontrar todos los modelos que se requerían para implementar, al igual que las animaciones realizadas fueron desarrolladas de manera correcta según lo solicitado, específicamente I que realiza la nave descendiendo de manera parabólica se tomaron en cuenta los conceptos de este efecto. Finalmente, el proyecto está dentro del repositorio listo para poderse clonar y probar sin errores.

Limitantes:

Para este proyecto los mayores obstáculos que se presentaron fueron a la hora de implementar modelos ya que estos podían ser de tamaños no soportables para el programa Visual Studio, sin embargo la mayoría de lo que se agrego era necesario para que ambiente virtual tuviera todos los elementos importantes que se deben

contar, un ejemplo de lo que digo es el modelo de mees3.obj dentro de la carpeta de Models el cual estaba planeado de anexar pero los límites que soporta el programa no lo permitieron, otra gran problemática fue al momento de anexar al repositorio pues este tiene ciertas restricciones que no permitían meter todo el proyecto por lo que fue necesario comprimir los archivos que tenían esa dificultad al subirse, para todo lo demás era cuestión de practica para recrear los modelos que tuviera que agregar y el código viene referenciado de las practicas vistas en clase por lo que era realizar los cambios requeridos para lograr el efecto deseado.

Documentación del código:

```
1 //Marcelo Romero Adolfo
2 //Proyecto Final Computacion Grafica Gpo 08
3
4 #include <iostream>
5 #include <cmath>
6
7 // GLEW
8 #include <GL/glew.h>
9
10 // GLFW
11 #include <GLFW/glfw3.h>
12
13 // Other Libs
14 #include "stb_image.h"
15
16 // GLM Mathematics
17 #include <glm/glm.hpp>
18 #include <glm/gtc/matrix_transform.hpp>
19 #include <glm/gtc/type_ptr.hpp>
20
21 //Load Models
22 #include "SOIL2/SOIL2.h"
23
24
25 // Other includes
26 #include "Shader.h"
27 #include "Camera.h"
28 #include "Model.h"
29 #include "Texture.h"
30
31
32 // Function prototypes
33 void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int
mode);
34 void MouseCallback(GLFWwindow* window, double xPos, double yPos);
35 void DoMovement();
36 void animation();
37
38 // Window dimensions
39 const GLuint WIDTH = 800, HEIGHT = 600;
40 int SCREEN_WIDTH, SCREEN_HEIGHT, i=0, j=0, k=0;
41
42 // Camera
43 Camera camera(glm::vec3(0.0f, 50.0f, 550.0f));
44 GLfloat lastX = WIDTH / 2.0;
45 GLfloat lastY = HEIGHT / 2.0;
46 bool keys[1024];
47 bool firstMouse = true;
48 float range = 0.0f;
49 float rot = 0.0f;
50
51
```

Declaracion
de librerias

Declaracion
de variables

```
52 // Light attributes
53 glm::vec3 lightPos(0.0f, 0.0f, 0.0f);
54 glm::vec3 PosIni(0.0f, 0.0f, 0.0f);
55 bool active;
56
57 //Animación para nave y robot
58 float movKitX = 0.0;
59 float movKitZ = 0.0;
60 float movKitY = 0.0;
61 float movKitX2 = 0.0;
62 float movKitY2 = 0.0;
63 float movKitZ2 = 0.0;
64 float rotKit = 90.0;
65 float rotKitX2 = 0.0;
66 float rotKitY2 = 0.0;
67 float rotKitZ2 = 0.0;
68 bool circuito = false;
69 bool circuito2 = false;
70 bool recorrido1 = true;
71 bool recorrido2 = false;
72 bool recorrido3 = false;
73 bool recorrido4 = false;
74 bool recorrido5 = false;
75 bool recorrido6 = true;
76 bool recorrido7 = false;
77 bool recorrido8 = false;
78 bool recorrido9 = false;
79 bool recorrido10 = false;
80 bool recorrido11 = false;
81 bool recorrido12 = false;
82 bool recorrido13 = false;
83
84 // Deltatime
85 GLfloat deltaTime = 0.0f; // Time between current frame and last frame
86 GLfloat lastFrame = 0.0f; // Time of last frame
87
88 // Keyframes
89 float posX = PosIni.x, posY = PosIni.y, posZ = PosIni.z, rotRodIzq = 0,
rotRodDer, rotBraIzq, rotBraDer, BrazM, BrazRob;
90
91 #define MAX_FRAMES 9
92 int i_max_steps = 190;
93 int i_curr_steps = 0;
94 typedef struct _frame
95 {
96     //Variables para GUARDAR Key Frames
97     float posX; //Variable para PosicionX
98     float posY; //Variable para PosicionY
99     float posZ; //Variable para PosicionZ
100     float incX; //Variable para IncrementoX
101     float incY; //Variable para IncrementoY
102     float incZ; //Variable para IncrementoZ

```

Declaracion de
variables para el
recorrido de
modelos

Declaración de
variables de
control

```

183 float rotRodIzq;
184 float rotRodDer;
185 float rotBraIzq;
186 float rotBraDer;
187 float BrazM;
188 float BrazRob;
189 float rotInc;
190 float rotInc2;
191 float rotInc3;
192 float rotInc4;
193 float rotInc5;
194 float rotInc6;
195
196 }FRAME;
197
198 FRAME KeyFrame[MAX_FRAMES];
199 int FrameIndex = 0; //Introducir datos
200 bool play = false;
201 int playIndex = 0;
202
203 // Positions of the point lights
204 glm::vec3 pointLightPositions[] = {
205     glm::vec3(posX,posY,posZ),
206     glm::vec3(0,0,0),
207     glm::vec3(0,0,0),
208     glm::vec3(0,0,0)
209 };
210
211 glm::vec3 LightP1;
212
213 void saveFrame(void)
214 {
215     printf("FrameIndex %d\n", FrameIndex);
216
217     KeyFrame[FrameIndex].posX = posX;
218     KeyFrame[FrameIndex].posY = posY;
219     KeyFrame[FrameIndex].posZ = posZ;
220
221     KeyFrame[FrameIndex].rotRodIzq = rotRodIzq;
222     KeyFrame[FrameIndex].rotRodDer = rotRodDer;
223     KeyFrame[FrameIndex].rotBraIzq = rotBraIzq;
224     KeyFrame[FrameIndex].rotBraDer = rotBraDer;
225     KeyFrame[FrameIndex].BrazM = BrazM;
226     KeyFrame[FrameIndex].BrazRob = BrazRob;
227
228     FrameIndex++;
229 }
230
231 void resetElements(void)
232 {
233     posX = KeyFrame[0].posX;
234     posY = KeyFrame[0].posY;
235     posZ = KeyFrame[0].posZ;
236
237     rotRodIzq = KeyFrame[0].rotRodIzq;
238     rotRodDer = KeyFrame[0].rotRodDer;
239     rotBraIzq = KeyFrame[0].rotBraIzq;
240     rotBraDer = KeyFrame[0].rotBraDer;
241     BrazM = KeyFrame[0].BrazM;
242     BrazRob = KeyFrame[0].BrazRob;
243 }
244
245 void interpolation(void)
246 {
247     KeyFrame[playIndex].incX = (KeyFrame[playIndex + 1].posX - KeyFrame
248     [playIndex].posX) / 1_max_steps;
249     KeyFrame[playIndex].incY = (KeyFrame[playIndex + 1].posY - KeyFrame
250     [playIndex].posY) / 1_max_steps;
251     KeyFrame[playIndex].incZ = (KeyFrame[playIndex + 1].posZ - KeyFrame
252     [playIndex].posZ) / 1_max_steps;
253
254     KeyFrame[playIndex].rotInc = (KeyFrame[playIndex + 1].rotRodIzq - KeyFrame
255     [playIndex].rotRodIzq) / 1_max_steps;
256     KeyFrame[playIndex].rotInc2 = (KeyFrame[playIndex + 1].rotRodDer - KeyFrame
257     [playIndex].rotRodDer) / 1_max_steps;
258     KeyFrame[playIndex].rotInc3 = (KeyFrame[playIndex + 1].rotBraIzq - KeyFrame
259     [playIndex].rotBraIzq) / 1_max_steps;
260     KeyFrame[playIndex].rotInc4 = (KeyFrame[playIndex + 1].rotBraDer - KeyFrame
261     [playIndex].rotBraDer) / 1_max_steps;
262     KeyFrame[playIndex].rotInc5 = (KeyFrame[playIndex + 1].BrazM - KeyFrame
263     [playIndex].BrazM) / 1_max_steps;
264     KeyFrame[playIndex].rotInc6 = (KeyFrame[playIndex + 1].BrazRob - KeyFrame
265     [playIndex].BrazRob) / 1_max_steps;
266 }
267
268 int main()
269 {
270     // Init GLFW
271     glfwInit();
272     // Set all the required options for GLFW
273     glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
274     glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
275     glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
276     glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
277
278     glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
279
280     // Create a GLFWwindow object that we can use for GLFW's functions
281     GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "ProyectoFinalLab",
282     nullptr, nullptr);
283
284     if (nullptr == window)
285     {
286         std::cout << "Failed to create GLFW window" << std::endl;
287         glfwTerminate();
288         return EXIT_FAILURE;
289     }
290
291     glfwMakeContextCurrent(window);
292
293     glfwSetFramebufferSizeCallback(window, &SCREEN_WIDTH, &SCREEN_HEIGHT);
294
295     // Set the required callback functions
296     glfwSetKeyCallback(window, KeyCallback);
297     glfwSetCursorPosCallback(window, MouseCallback);
298     printf("Xf", glfwGetTime());
299
300     // GLFW Options
301     glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);
302
303     // Set this to true so GLEW knows to use a modern approach to retrieving
304     // function pointers and extensions
305     glewExperimental = GL_TRUE;
306     // Initialize GLEW to setup the OpenGL Function pointers
307     if (GLEW_OK != glewInit())
308     {
309         std::cout << "Failed to initialize GLEW" << std::endl;
310         return EXIT_FAILURE;
311     }
312
313     // Define the viewport dimensions
314     glViewport(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT);
315
316     // OpenGL options
317     glEnable(GL_DEPTH_TEST);
318     glEnable(GL_BLEND);
319     glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
320
321     Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting.frag");
322     Shader lampShader("Shaders/lamp.vs", "Shaders/lamp.frag");
323     Shader SkyBoxShader("Shaders/SkyBox.vs", "Shaders/SkyBox.frag");
324     Shader shader("Shaders/modelLoading.vs", "Shaders/modelLoading.frag");
325
326     Model CasaRym((char*)"Models/CasaRym/CasaRym.obj");
327     Model Rick((char*)"Models/Rick/CuerpoRick.obj");
328     Model BrazoDer((char*)"Models/Rick/BrazoDer.obj");
329
330     Model BrazoIzR((char*)"Models/Rick/BrazoIzR.obj");
331     Model Nada((char*)"Models/Nada/Nada.obj");
332     Model Portal((char*)"Models/Portal/Portal.obj");
333     Model Alien((char*)"Models/Alien/Alien.obj");
334     Model models[3] = { Nada, Alien, Portal };
335     Model Robot((char*)"Models/Robot/CuerpoRob.obj");
336     Model BRobot((char*)"Models/Robot/BrazRob.obj");
337     Model Nave((char*)"Models/Nave/Nave.obj");
338     Model Boxm((char*)"Models/boxm/boxm.obj");
339     Model CuerpoM((char*)"Models/mees/CuerpoM2.obj");
340     Model modelsM[2] = { Nada, CuerpoM };
341     Model HombreDM((char*)"Models/mees/HombreDM.obj");
342     Model BrazDM((char*)"Models/mees/BrazDM.obj");
343     Model HombreIM((char*)"Models/mees/HombreIM.obj");
344     Model BrazIM((char*)"Models/mees/BrazIM.obj");
345     Model M1((char*)"Models/mees/mees.obj");
346     Model M2((char*)"Models/mees/mees2.obj");
347     Model M3((char*)"Models/mees/mees3.obj"); //Deje el modelo "mees.obj"
348     //aporposito ya que agregar "mees3.obj" satura el programa
349     Model models2[4] = { Nada,M3,M2,M1 };
350
351     //Inicialización de KeyFrames
352     for (int i = 0; i < MAX_FRAMES; i++)
353     {
354         KeyFrame[i].posX = 0;
355         KeyFrame[i].incX = 0;
356         KeyFrame[i].incY = 0;
357         KeyFrame[i].incZ = 0;
358         KeyFrame[i].rotRodIzq = 0;
359         KeyFrame[i].rotRodDer = 0;
360         KeyFrame[i].rotBraIzq = 0;
361         KeyFrame[i].rotBraDer = 0;
362         KeyFrame[i].BrazM = 0;
363         KeyFrame[i].BrazRob = 0;
364         KeyFrame[i].rotInc = 0;
365         KeyFrame[i].rotInc2 = 0;
366         KeyFrame[i].rotInc3 = 0;
367         KeyFrame[i].rotInc4 = 0;
368         KeyFrame[i].rotInc5 = 0;
369         KeyFrame[i].rotInc6 = 0;
370     }
371
372     // Set up vertex data (and buffer(s)) and attribute pointers
373     GLfloat vertices[] =
374     {
375         // Positions           // Normals           // Texture Coords
376         -0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f,
377
378

```

Declaración de variables de control de modelos

Definiendo KeyFrames a sus respectivos modelos

Anexando shaders

Definiendo KeyFrames a sus respectivos modelos

Anexando Modelos desde su carpeta correspondiente

Inicializando KeyFrames

Integrando funcion de KeyFrames a sus respectivos modelos


```

299     0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
300     0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
301     0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
302     -0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
303     -0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f,
304
305     -0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
306     0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f,
307     0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
308     0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
309     -0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,
310     -0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
311
312     -0.5f, 0.5f, 0.5f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
313     -0.5f, 0.5f, -0.5f, -1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
314     -0.5f, -0.5f, -0.5f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
315     -0.5f, -0.5f, -0.5f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
316     -0.5f, -0.5f, 0.5f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
317     -0.5f, 0.5f, 0.5f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
318
319     0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
320     0.5f, 0.5f, -0.5f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
321     0.5f, -0.5f, -0.5f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
322     0.5f, -0.5f, -0.5f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
323     0.5f, -0.5f, 0.5f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
324     0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
325
326     -0.5f, -0.5f, -0.5f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
327     0.5f, -0.5f, -0.5f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f,
328     0.5f, -0.5f, 0.5f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
329     0.5f, -0.5f, 0.5f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,
330     -0.5f, -0.5f, 0.5f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,
331     -0.5f, -0.5f, -0.5f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,
332
333     -0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
334     0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
335     0.5f, 0.5f, 0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
336     0.5f, 0.5f, 0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
337     -0.5f, 0.5f, 0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
338     -0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
339
340 );
341
342 GLfloat skyboxVertices[] = {
343     // Positions
344     -1.0f, 1.0f, -1.0f,
345     -1.0f, -1.0f, -1.0f,
346     1.0f, -1.0f, -1.0f,
347     1.0f, -1.0f, 1.0f,
348     1.0f, 1.0f, 1.0f,
349     -1.0f, 1.0f, 1.0f,
350
351     -1.0f, -1.0f, 1.0f,
352     -1.0f, -1.0f, -1.0f,
353     -1.0f, 1.0f, -1.0f,
354     -1.0f, 1.0f, 1.0f,
355     -1.0f, -1.0f, 1.0f,
356     -1.0f, -1.0f, -1.0f,
357
358     1.0f, -1.0f, -1.0f,
359     1.0f, -1.0f, 1.0f,
360     1.0f, 1.0f, 1.0f,
361     1.0f, 1.0f, -1.0f,
362     1.0f, -1.0f, -1.0f,
363     1.0f, -1.0f, 1.0f,
364
365     -1.0f, -1.0f, 1.0f,
366     -1.0f, 1.0f, 1.0f,
367     1.0f, 1.0f, 1.0f,
368     1.0f, 1.0f, -1.0f,
369     1.0f, -1.0f, -1.0f,
370     -1.0f, -1.0f, 1.0f,
371
372     -1.0f, 1.0f, -1.0f,
373     1.0f, 1.0f, -1.0f,
374     1.0f, 1.0f, 1.0f,
375     1.0f, 1.0f, -1.0f,
376     -1.0f, 1.0f, 1.0f,
377     -1.0f, 1.0f, -1.0f,
378
379     -1.0f, -1.0f, -1.0f,
380     -1.0f, -1.0f, 1.0f,
381     1.0f, -1.0f, -1.0f,
382     1.0f, -1.0f, 1.0f,
383     -1.0f, -1.0f, 1.0f,
384     1.0f, -1.0f, -1.0f,
385 };
386
387 GLuint indices[] = {
388     // Note that we start from 0!
389     0,1,2,3,
390     4,5,6,7,
391     8,9,10,11,
392     12,13,14,15,
393     16,17,18,19,
394     20,21,22,23,
395     24,25,26,27,
396     28,29,30,31,
397     32,33,34,35
398 };
399
400 // Positions all containers
401 glm::vec3 cubePositions[] = {

```

Se mantuvo el modelado del cubo de las practicas por si se llegaba a ocupar

```

403     glm::vec3(0.0f, 0.0f, 0.0f),
404     glm::vec3(2.0f, 5.0f, -15.0f),
405     glm::vec3(-1.5f, -2.2f, -2.5f),
406     glm::vec3(-3.8f, -2.0f, -12.3f),
407     glm::vec3(2.4f, -0.4f, -3.5f),
408     glm::vec3(-1.7f, 3.0f, -7.5f),
409     glm::vec3(1.3f, -2.0f, -2.5f),
410     glm::vec3(1.5f, 2.0f, -2.5f),
411     glm::vec3(1.5f, 0.2f, -1.5f),
412     glm::vec3(-1.3f, 1.0f, -1.5f)
413 };
414
415 // First, set the container's VAO (and VBO)
416 GLuint VBO, VAO, EBO;
417 glGenVertexArrays(1, &VAO);
418 glGenBuffers(1, &VBO);
419 glGenBuffers(1, &EBO);
420
421 glBindVertexArray(VAO);
422 glBindBuffer(GL_ARRAY_BUFFER, VBO);
423 glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
424
425 glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
426 glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);
427
428 // Position attribute
429 glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)0);
430 glEnableVertexAttribArray(0);
431 // Normals attribute
432 glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)(3 * sizeof(GLfloat)));
433 glEnableVertexAttribArray(1);
434 // Texture Coordinate attribute
435 glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)(6 * sizeof(GLfloat)));
436 glEnableVertexAttribArray(2);
437 glBindVertexArray(0);
438
439 // Then, we set the light's VAO (VBO stays the same. After all, the vertices are the same for the light object (also a 3D cube))
440 GLuint lightVAO;
441 glGenVertexArrays(1, &lightVAO);
442 glBindVertexArray(lightVAO);
443 // We only need to bind to the VBO (to link it with glVertexAttribPointer), no need to fill it; the VBO's data already contains all we need.
444 glBindBuffer(GL_ARRAY_BUFFER, VBO);
445 // Set the vertex attributes (only position data for the lamp)
446 glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)0);
447 glEnableVertexAttribArray(0);
448 glBindVertexArray(0);
449
450 //SkyBox
451 GLuint skyboxVBO, skyboxVAO;
452 glGenVertexArrays(1, &skyboxVAO);
453 glGenBuffers(1, &skyboxVBO);
454 glBindVertexArray(skyboxVAO);
455 glBindBuffer(GL_ARRAY_BUFFER, skyboxVBO);
456 glBufferData(GL_ARRAY_BUFFER, sizeof(skyboxVertices), &skyboxVertices, GL_STATIC_DRAW);
457 glEnableVertexAttribArray(0);
458 glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);
459
460 // Load textures
461 vector<const GLchar*> faces;
462 faces.push_back("SkyBox/gatosS.tga");
463 faces.push_back("SkyBox/gatosS.tga");
464 faces.push_back("SkyBox/gatosS.tga");
465 faces.push_back("SkyBox/gatosS.tga");
466 faces.push_back("SkyBox/gatosS.tga");
467 faces.push_back("SkyBox/gatosS.tga");
468
469 GLuint cubemapTexture = TextureLoading::LoadCubemap(faces);
470
471 glm::mat4 projection = glm::perspective(camera.GetZoom(), (GLfloat) SCREEN_WIDTH / (GLfloat) SCREEN_HEIGHT, 0.1f, 1000.0f);
472
473 // Game loop
474 while (!glfwWindowShouldClose(window))
475 {
476     // Calculate deltatime of current frame
477     GLfloat currentFrame = glfwGetTime();
478     deltaTime = currentFrame - lastFrame;
479     lastFrame = currentFrame;
480
481     // Check if any events have been activated (key pressed, mouse moved etc.) and call corresponding response functions
482     glfwPollEvents();
483     DoMovement();
484     animacion();
485
486     // Clear the colorbuffer
487     glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
488     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
489
490
491
492
493
494

```

Anexando textura para el Skybox desde su carpeta

```

495 // Use corresponding shader when setting uniforms/drawing objects
496 lightingShader.Use();
497 GLint viewPosLoc = glGetUniformLocation(lightingShader.Program,
498     "viewPos");
499 // Set material properties
500 glUniform3f(glGetUniformLocation(lightingShader.Program,
501     "material.shininess"), 32.0f);
502 // Here we set all the uniforms for the 5/6 types of lights we have. We
503 // have to set them manually and index
504 // the proper PointLight struct in the array to set each uniform
505 // variable. This can be done more code-friendly
506 // by defining light types as classes and set their values in there, or
507 // by using a more efficient uniform approach
508 // by using 'Uniform buffer objects', but that is something we discuss
509 // in the 'Advanced GLSL' tutorial.
510 // Directional light
511 glUniform3f(glGetUniformLocation(lightingShader.Program,
512     "dirLight.direction"), -0.2f, -1.0f, -0.3f);
513 glUniform3f(glGetUniformLocation(lightingShader.Program,
514     "dirLight.ambient"), 1.0f, 1.0f, 1.0f);
515 glUniform3f(glGetUniformLocation(lightingShader.Program,
516     "dirLight.diffuse"), 0.4f, 0.4f, 0.4f);
517 glUniform3f(glGetUniformLocation(lightingShader.Program,
518     "dirLight.specular"), 0.5f, 0.5f, 0.5f);
519 // Point light 1
520 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights
521 [0].position"), pointLightPositions[0].x, pointLightPositions[0].y,
522 pointLightPositions[0].z);
523 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights
524 [0].ambient"), 0.05f, 0.05f, 0.05f);
525 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights
526 [0].diffuse"), lightP1.x, lightP1.y, lightP1.z);
527 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights
528 [0].specular"), lightP1.x, lightP1.y, lightP1.z);
529 glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights
530 [0].constant"), 1.0f);
531 glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights
532 [0].linear"), 0.09f);
533 glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights
534 [0].quadratic"), 0.032f);
535 // Point light 2
536 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights
537 [1].position"), pointLightPositions[1].x, pointLightPositions[1].y,
538 pointLightPositions[1].z);
539 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights
540 [1].ambient"), 0.05f, 0.05f, 0.05f);
541 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights
542 [1].diffuse"), 1.0f, 1.0f, 0.0f);
543 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights
544 [1].specular"), 1.0f, 1.0f, 0.0f);
545 glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights
546 [1].constant"), 1.0f);
547 glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights
548 [1].linear"), 0.09f);
549 glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights
550 [1].quadratic"), 0.032f);
551 // Point light 3
552 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights
553 [2].position"), pointLightPositions[2].x, pointLightPositions[2].y,
554 pointLightPositions[2].z);
555 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights
556 [2].ambient"), 0.05f, 0.05f, 0.05f);
557 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights
558 [2].diffuse"), 0.0f, 1.0f, 1.0f);
559 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights
560 [2].specular"), 0.0f, 1.0f, 1.0f);
561 glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights
562 [2].constant"), 1.0f);
563 glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights
564 [2].linear"), 0.09f);
565 glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights
566 [2].quadratic"), 0.032f);
567 // Point light 4
568 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights
569 [3].position"), pointLightPositions[3].x, pointLightPositions[3].y,
570 pointLightPositions[3].z);
571 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights
572 [3].ambient"), 0.05f, 0.05f, 0.05f);
573 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights
574 [3].diffuse"), 1.0f, 0.0f, 1.0f);
575 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights
576 [3].specular"), 1.0f, 0.0f, 1.0f);
577 glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights
578 [3].constant"), 1.0f);
579 glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights
580 [3].linear"), 0.09f);
581 glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights
582 [3].quadratic"), 0.032f);
583 // SpotLight
584 glUniform3f(glGetUniformLocation(lightingShader.Program,
585     "spotLight.position"), camera.GetPosition().x, camera.GetPosition().y,
586     camera.GetPosition().z);
587 glUniform3f(glGetUniformLocation(lightingShader.Program,
588     "spotLight.direction"), camera.GetFront().x, camera.GetFront().y,
589     camera.GetFront().z);
590 glUniform3f(glGetUniformLocation(lightingShader.Program,
591     "spotLight.ambient"), 0.0f, 0.0f, 0.0f);
592 glUniform3f(glGetUniformLocation(lightingShader.Program,
593     "spotLight.diffuse"), 0.0f, 0.0f, 0.0f);
594 glUniform3f(glGetUniformLocation(lightingShader.Program,
595     "spotLight.specular"), 0.0f, 0.0f, 0.0f);
596 glUniform1f(glGetUniformLocation(lightingShader.Program,
597     "spotLight.constant"), 1.0f);
598 glUniform1f(glGetUniformLocation(lightingShader.Program,
599     "spotLight.linear"), 0.09f);
600 glUniform1f(glGetUniformLocation(lightingShader.Program,
601     "spotLight.quadratic"), 0.032f);
602 glUniform1f(glGetUniformLocation(lightingShader.Program,
603     "spotLight.cutOff"), glm::cos(glm::radians(12.5f)));
604 glUniform1f(glGetUniformLocation(lightingShader.Program,
605     "spotLight.outerCutOff"), glm::cos(glm::radians(15.0f)));
606 // Set material properties
607 glUniform1f(glGetUniformLocation(lightingShader.Program,
608     "material.shininess"), 32.0f);
609 // Create camera transformations
610 glm::mat4 view;
611 view = camera.GetViewMatrix();
612 // Get the uniform locations
613 GLint modelLoc = glGetUniformLocation(lightingShader.Program, "model");
614 GLint viewLoc = glGetUniformLocation(lightingShader.Program, "view");
615 GLint projLoc = glGetUniformLocation(lightingShader.Program,
616     "projection");
617 // Pass the matrices to the shader
618 glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
619 glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
620 glBindVertexArray(VAO);
621 glm::mat4 tmp = glm::mat4(1.0f); //Temp
622 //Carga de modelo
623 //Casa de Rick y Morty
624 view = camera.GetViewMatrix();
625 glm::mat4 model(1);
626 glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
627 CasaRyM.Draw(lightingShader);
628 //Rick
629 view = camera.GetViewMatrix();
630 tmp = model = glm::translate(model, glm::vec3(-150, 40, 380));
631 glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
632 Rick.Draw(lightingShader);
633 //BrazoIzq
634 view = camera.GetViewMatrix();
635 tmp = model = glm::translate(model, glm::vec3(3, 11.5, 1.5));
636 //model = glm::translate(model, glm::vec3(posX, posY, posZ));
637 //model = glm::rotate(model, glm::radians(rot), glm::vec3(0.0f, 1.0f,
638 0.0f));
639 model = glm::rotate(model, glm::radians(-rotBraDer), glm::vec3(1.0f,
640 0.0f, 0.0f));
641 glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
642 BrazoIzR.Draw(lightingShader);
643 //BrazoDer
644 view = camera.GetViewMatrix();
645 model = glm::translate(tmp, glm::vec3(-7, 0, 0));
646 model = glm::rotate(model, glm::radians(-45.0f), glm::vec3(1.0f, 0.0f,
647 0.0f));
648 model = glm::rotate(model, glm::radians(-rotBraIzq), glm::vec3(1.0f,
649 0.0f, 0.0f));
650 glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
651 BrazoDeR.Draw(lightingShader);
652 // Also draw the lamp object, again binding the appropriate shader
653 lampShader.Use();
654 // Get location objects for the matrices on the lamp shader (these could
655 // 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 // Set matrices
660 shader.Use();
661 view = camera.GetViewMatrix();
662 glUniformMatrix4fv(glGetUniformLocation(shader.Program, "projection"),
663 1, GL_FALSE, glm::value_ptr(projection));
664 glUniformMatrix4fv(glGetUniformLocation(shader.Program, "view"), 1,
665 GL_FALSE, glm::value_ptr(view));
666 // Draw the loaded model
667 //Portal y Alien
668 model = glm::rotate(model, glm::radians(45.0f), glm::vec3(1.0f, 0.0f,
669 0.0f));
670 model = glm::translate(model, glm::vec3(-10, 5, 100)); // Translate it
671 down a bit so it's at the center of the scene
672 model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f)); // It's a bit
673 too big for our scene, so scale it down

```

Se mantuvo la configuración de iluminación de las practicas

Carga del modelo de la casa

Carga del modelo de Rick

Carga de modelos de los brazos Rick

Carga de modelos: portal y alien


```

636 model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f,
637 0.0f));
638 glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1,
639 GL_FALSE, glm::value_ptr(model));
640 models[i].Draw(shader);
641 //Robot de la mantequilla
642 view = camera.GetViewMatrix();
643 model = glm::mat4(1);
644 model = glm::translate(model, glm::vec3(-22, 43.4, 150));
645 model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f,
646 0.0f));
647 model = glm::translate(model, PosIni + glm::vec3(movKitX, 0, movKitZ));
648 model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f, 1.0f,
649 0.0f));
650 glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
651 Robot.Draw(lightingShader);
652 //Brazos de robot
653 view = camera.GetViewMatrix();
654 model = glm::mat4(1);
655 model = glm::translate(model, glm::vec3(-22, 47.2, 150.5));
656 model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f,
657 0.0f));
658 model = glm::rotate(model, glm::radians(-BrazRob), glm::vec3(1.0f, 0.0f,
659 0.0f));
660 glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
661 BRobot.Draw(lightingShader);
662 //Nave
663 view = camera.GetViewMatrix();
664 model = glm::mat4(1);
665 model = glm::translate(model, glm::vec3(-250, 60, 450));
666 model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f,
667 0.0f));
668 model = glm::translate(model, PosIni + glm::vec3(movKitX2, movKitY2,
669 movKitZ2));
670 model = glm::rotate(model, glm::radians(rotKitX2), glm::vec3(0.0f, 1.0f,
671 0.0f));
672 model = glm::rotate(model, glm::radians(rotKitY2), glm::vec3(1.0f, 0.0f,
673 0.0f));
674 model = glm::rotate(model, glm::radians(rotKitZ2), glm::vec3(0.0f, 0.0f,
675 1.0f));
676 glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
677 Nave.Draw(lightingShader);
678 //Caja de Meeseeks
679 view = camera.GetViewMatrix();
680 model = glm::mat4(1);
681 model = glm::translate(model, glm::vec3(-248, 56.5, 50));
682 glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
683 BoxM.Draw(lightingShader);
684 //Meeseeks
685 view = camera.GetViewMatrix();
686 model = glm::mat4(1);
687 model = glm::translate(model, glm::vec3(-250, 48.1, 70));
688 model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f,
689 0.0f));
690 glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1,
691 GL_FALSE, glm::value_ptr(model));
692 modelsM[k].Draw(shader);
693 view = camera.GetViewMatrix();
694 ttp = model = glm::translate(model, glm::vec3(-11.2, 12, 12.2));
695 model = glm::rotate(model, glm::radians(-BrazM), glm::vec3(0.0f, 0.0f,
696 1.0f));
697 glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
698 BrazDM.Draw(lightingShader);
699 view = camera.GetViewMatrix();
700 model = glm::mat4(1);
701 model = glm::translate(model, glm::vec3(-250, 48.1, 30)); // Translate
702 it down a bit so it's at the center of the scene
703 glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1,
704 GL_FALSE, glm::value_ptr(model));
705 models2[j].Draw(shader);
706 glBindVertexArray(0);
707 // Draw skybox as last
708 glDepthFunc(GL_EQUAL); // Change depth function so depth test passes
709 when values are equal to depth buffer's content
710 SkyBoxshader.Use();
711 view = glm::mat4(camera.GetViewMatrix()); // Remove any
712 translation component of the view matrix
713 glUniformMatrix4fv(glGetUniformLocation(SkyBoxshader.Program, "view"),
714 1, GL_FALSE, glm::value_ptr(view));
715 glUniformMatrix4fv(glGetUniformLocation(SkyBoxshader.Program,
716 "projection"), 1, GL_FALSE, glm::value_ptr(projection));
717 // skybox cube
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

```

Carga de modelos:
robot y sus brazos

Carga de modelos:
Nave

Carga de modelos:
Meeseeks y su caja

```

719 // Swap the screen buffers
720 glfwSwapBuffers(window);
721 }
722 }
723 }
724 }
725 glDeleteVertexArrays(1, &VAO);
726 glDeleteVertexArrays(1, &lightVAO);
727 glDeleteBuffers(1, &VB0);
728 glDeleteBuffers(1, &EB0);
729 glDeleteVertexArrays(1, &skyboxVAO);
730 glDeleteBuffers(1, &skyboxVB0);
731 // Terminate GLFW, clearing any resources allocated by GLFW.
732 glfwTerminate();
733 }
734 }
735 }
736 }
737 return 0;
738 }
739 }
740 }
741 void animacion()
742 {
743 //Movimiento del personaje
744 if (play)
745 {
746 if (i_curr_steps >= i_max_steps) //end of animation between frames?
747 {
748 playIndex++;
749 if (playIndex > FrameIndex - 2) //end of total animation?
750 {
751 printf("termina anim\n");
752 playIndex = 0;
753 play = false;
754 }
755 else //Next frame interpolations
756 {
757 i_curr_steps = 0; //Reset counter
758 //Interpolation
759 interpolation();
760 }
761 }
762 else
763 {
764 //Draw animation
765 posX += KeyFrame[playIndex].incX;
766 posY += KeyFrame[playIndex].incY;
767 posZ += KeyFrame[playIndex].incZ;
768 rotRodIzq += KeyFrame[playIndex].rotInc;
769 rotRodDer += KeyFrame[playIndex].rotInc2;
770 rotBraIzq += KeyFrame[playIndex].rotInc3;
771 rotBraDer += KeyFrame[playIndex].rotInc4;
772 BrazM += KeyFrame[playIndex].rotInc5;
773 BrazRob += KeyFrame[playIndex].rotInc6;
774 i_curr_steps++;
775 }
776 }
777 //recorridos
778 if (circuit)
779 {
780 if (recorrido1)
781 {
782 movKitX += 0.1f;
783 if (movKitX > 30)
784 {
785 recorrido1 = false;
786 recorrido2 = true;
787 }
788 }
789 if (recorrido2)
790 {
791 rotKit = 320;
792 movKitX += 0.1f;
793 movKitZ += 0.15f;
794 if (movKitX < 0 && movKitZ > 47)
795 {
796 recorrido2 = false;
797 recorrido3 = true;
798 }
799 }
800 if (recorrido3)
801 {
802 rotKit = 90;
803 movKitX += 0.1f;
804 if (movKitX > 30)
805 {
806 recorrido3 = false;
807 recorrido4 = true;
808 }
809 }
810 if (recorrido4)
811 {
812 rotKit = 220;
813 movKitX += 0.1f;
814 movKitZ += 0.15f;
815 if (movKitX < 0 && movKitZ > 0)
816 {
817 }
818 }
819 }
820 }
821 }
822 }

```

Dibujando
KeyFrames de los
respectivos
modelos

Planteamiento de
los recorridos del
robot

```

823     {
824         recorrido4 = false;
825         recorrido5 = true;
826     }
827 }
828 if (recorrido5)
829 {
830     rotKit = 90;
831     movKitX += 0.1f;
832     if (movKitX > 0)
833     {
834         recorrido5 = false;
835         recorrido1 = true;
836     }
837 }
838 }
839 }
840 if (circuito2)
841 {
842     if (recorrido6)
843     {
844         movKitY2 += 1;
845         if (movKitY2 > 50)
846         {
847             recorrido6 = false;
848             recorrido7 = true;
849         }
850     }
851     if (recorrido7)
852     {
853         rotKitY2 = 330;
854         movKitY2 += 1;
855         movKitZ2 += 1;
856         if (movKitZ2 > 200 && movKitY2 > 200)
857         {
858             recorrido7 = false;
859             recorrido8 = true;
860         }
861     }
862 }
863 }
864 if (recorrido8)
865 {
866     rotKitY2 = 360;
867     rotKitX2 = 90;
868     movKitX2 += 1;
869     if (movKitX2 > 200)
870     {
871         recorrido8 = false;
872         recorrido9 = true;
873     }
874 }
875 if (recorrido9)
876 {
877     rotKitX2 = 45;
878     movKitX2 += 1;
879     movKitZ2 += 1;
880     if (movKitX2 > 500 && movKitZ2 > 500)
881     {
882         recorrido9 = false;
883         recorrido10 = true;
884     }
885 }
886 if (recorrido10)
887 {
888     rotKitX2 = 180;
889     movKitZ2 += 1;
890     if (movKitZ2 < 0)
891     {
892         recorrido10 = false;
893         recorrido11 = true;
894     }
895 }
896 if (recorrido11)
897 {
898     rotKitX2 = 270;
899     movKitX2 += 8;
900     movKitY2 += 1;
901     if (movKitX2 < 200 && movKitY2 < 200)
902     {
903         recorrido11 = false;
904         recorrido12 = true;
905     }
906 }
907 if (recorrido12)
908 {
909     movKitX2 += 2;
910     movKitY2 += 4;
911     if (movKitX2 < 0 && movKitY2 < 0)
912     {
913         recorrido12 = false;
914         recorrido13 = true;
915     }
916 }
917 if (recorrido13)
918 {
919     rotKitX2 = 0;
920     if (movKitX2 < 0 && movKitY2 < 0)
921     {
922         recorrido13 = false;
923         recorrido6 = true;
924     }
925 }
926 }

```

Planteamiento de los recorridos de la nave

```

927
928
929 }
930 }
931
932
933 // Is called whenever a key is pressed/released via GLFW
934 void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode)
935 {
936     if (GLFW_KEY_T == key && GLFW_PRESS == action) {
937         if (i == 0) {
938             i = 2;
939         }
940         else {
941             i = 1;
942         }
943     }
944     if (GLFW_KEY_M == key && GLFW_PRESS == action) {
945         if (j == 0) {
946             j = 3;
947         }
948         else {
949             j = 1;
950         }
951     }
952     if (GLFW_KEY_N == key && GLFW_PRESS == action) {
953         if (k == 0) {
954             k = 1;
955         }
956         else {
957             k = 1;
958         }
959     }
960     if (keys[GLFW_KEY_L])
961     {
962         if (play == false && (FrameIndex > 1))
963         {
964             resetElements();
965             //First Interpolation
966             interpolation();
967
968             play = true;
969             playIndex = 0;
970             i_curr_steps = 0;
971         }
972         else
973         {
974             play = false;
975         }
976     }
977 }
978
979 if (keys[GLFW_KEY_K])
980 {
981     if (FrameIndex < MAX_FRAMES)
982     {
983         saveFrame();
984     }
985 }
986
987
988
989
990
991 if (GLFW_KEY_ESCAPE == key && GLFW_PRESS == action)
992 {
993     glfwSetWindowShouldClose(window, GL_TRUE);
994 }
995
996 if (key >= 0 && key < 1024)
997 {
998     if (action == GLFW_PRESS)
999     {
1000         keys[key] = true;
1001     }
1002     else if (action == GLFW_RELEASE)
1003     {
1004         keys[key] = false;
1005     }
1006 }
1007
1008 if (keys[GLFW_KEY_SPACE])
1009 {
1010     active = !active;
1011     if (active)
1012         LightP1 = glm::vec3(1.0f, 0.0f, 0.0f);
1013     else
1014         LightP1 = glm::vec3(0.0f, 0.0f, 0.0f);
1015 }
1016 }
1017
1018 void MouseCallback(GLFWwindow* window, double xPos, double yPos)
1019 {
1020     if (firstMouse)
1021     {
1022         lastX = xPos;
1023         lastY = yPos;
1024         firstMouse = false;
1025     }
1026 }
1027
1028 GLfloat xOffset = xPos - lastX;
1029 GLfloat yOffset = lastY - yPos; // Reversed since y-coordinates go from

```

Asignando teclas para acciona las animaciones de transición

Asignando teclas para acciona las animaciones por KeyFrame

```

1030
1031 lastX = xPos;
1032 lastY = yPos;
1033
1034 camera.ProcessMouseMovement(xOffset, yOffset);
1035 }
1036
1037 // Moves/alters the camera positions based on user input
1038 void DoMovement()
1039 {
1040
1041     if (keys[GLFW_KEY_2])
1042     {
1043         if (rotBraDer < 80.0f)
1044             rotBraDer += 1.0f;
1045     }
1046
1047     if (keys[GLFW_KEY_3])
1048     {
1049         if (rotBraDer > -45)
1050             rotBraDer -= 1.0f;
1051     }
1052
1053     if (keys[GLFW_KEY_4])
1054     {
1055         if (rotBraIzq < 80.0f)
1056             rotBraIzq += 1.0f;
1057     }
1058
1059     if (keys[GLFW_KEY_5])
1060     {
1061         if (rotBraIzq > -45)
1062             rotBraIzq -= 1.0f;
1063     }
1064
1065     if (keys[GLFW_KEY_6])
1066     {
1067         if (BrazM < 80.0f)
1068             BrazM += 1.0f;
1069     }
1070
1071     if (keys[GLFW_KEY_7])
1072     {
1073         if (BrazM > -45)
1074             BrazM -= 1.0f;
1075     }
1076
1077     if (keys[GLFW_KEY_8])
1078     {
1079         if (BrazRob < 60.0f)
1080             BrazRob += 1.0f;
1081     }
1082
1083     if (keys[GLFW_KEY_9])
1084     {
1085         if (BrazRob > -15)
1086             BrazRob -= 1.0f;
1087     }
1088
1089     //Mov Personaje
1090     if (keys[GLFW_KEY_H])
1091     {
1092         posZ += 1;
1093     }
1094
1095     if (keys[GLFW_KEY_V])
1096     {
1097         posZ -= 1;
1098     }
1099
1100     if (keys[GLFW_KEY_G])
1101     {
1102         posX += 1;
1103     }
1104
1105     if (keys[GLFW_KEY_J])
1106     {
1107         posX -= 1;
1108     }
1109
1110     //CIRCUITOS
1111     if (keys[GLFW_KEY_U])
1112     {
1113         circuito = true;
1114     }
1115
1116     if (keys[GLFW_KEY_I])
1117     {
1118         circuito = false;
1119     }
1120
1121     if (keys[GLFW_KEY_O])
1122     {
1123         circuito2 = true;
1124     }
1125
1126 }

```

Asignando teclas de movimiento en animaciones por KeyFrame

```

1133
1134 if (keys[GLFW_KEY_P])
1135 {
1136     circuito2 = false;
1137 }
1138
1139
1140 // Camera controls
1141 if (keys[GLFW_KEY_W] || keys[GLFW_KEY_UP])
1142 {
1143     camera.ProcessKeyboard(FORWARD, deltaTime*0.5);
1144 }
1145
1146 if (keys[GLFW_KEY_S] || keys[GLFW_KEY_DOWN])
1147 {
1148     camera.ProcessKeyboard(BACKWARD, deltaTime*0.5);
1149 }
1150
1151 if (keys[GLFW_KEY_A] || keys[GLFW_KEY_LEFT])
1152 {
1153     camera.ProcessKeyboard(LEFT, deltaTime*0.5);
1154 }
1155
1156 if (keys[GLFW_KEY_D] || keys[GLFW_KEY_RIGHT])
1157 {
1158     camera.ProcessKeyboard(RIGHT, deltaTime*0.5);
1159 }
1160
1161
1162 }
1163
1164 }
1165
1166
1167
1168
1169
1170
1171
1172 }

```

Teclas de control para el manejo de la cámara

Asignando teclas para accionar las animaciones por recorridos