

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Computación Gráfica e Interacción Humano- Computadora

Manual Técnico

TORRES DE HANOI



Alumna: Carolina Téllez Gallardo

Número de cuenta: 316292896

Profesor: Ing. Luis Sergio Valencia Castro

INDICE

INTRODUCCIÓN.....	3
OBJETIVOS.....	4
DESCRIPCION DEL PROYECTO.....	5
ESPECIFICACIONES DE DESARROLLO.....	7
CODIFICACION DEL PROYECTO.....	8
SALIDAS.....	23
RECURSOS DE ACCESO.....	25
CONCLUSIONES.....	26
BIBLIOGRAFÍA DE CONSULTA.....	27

INTRODUCCIÓN

La Torre de Hanoi es un famoso rompecabezas matemático y de lógica que consta de tres postes y una serie de discos de diferentes tamaños que pueden deslizarse sobre los postes. La disposición inicial implica que todos los discos están apilados de mayor a menor tamaño en un solo poste, de manera que el disco más grande esté en la base y el más pequeño en la parte superior.

El objetivo del juego es mover toda la pila de discos al tercer poste, utilizando los otros dos postes como espacios auxiliares, manteniendo siempre la regla de que nunca se puede colocar un disco más grande sobre uno más pequeño.

Las reglas son simples:

1. Solo se puede mover un disco a la vez.
2. Nunca se puede colocar un disco más grande sobre uno más pequeño.
3. Se debe lograr mover toda la pila de discos del primer poste al tercero, utilizando el segundo como auxiliar, manteniendo el orden de los discos.

Ahora, la importancia de las Torres de Hanoi radica en varios aspectos:

1. **Ejemplo de recursión:** Este rompecabezas es un excelente ejemplo para enseñar y entender el concepto de recursión en programación. La solución al problema se basa en la lógica recursiva, dividiendo el problema en subproblemas más pequeños y resolviéndolos de manera recursiva.
2. **Aplicaciones en algoritmos:** La Torre de Hanoi es un caso de estudio común en algoritmos debido a su naturaleza recursiva. Ayuda a comprender la estructura de los algoritmos recursivos y su aplicación en la resolución de problemas complejos.
3. **Enseñanza de estrategias y resolución de problemas:** Al resolver las Torres de Hanoi, se fomenta el desarrollo de habilidades de pensamiento lógico y estratégico. Ayuda a los estudiantes a aprender a planificar y visualizar pasos hacia una solución.

En resumen, las Torres de Hanoi no solo son un desafío lógico interesante, sino que también son una herramienta educativa valiosa para comprender conceptos matemáticos y lógicos, así como para desarrollar habilidades en la resolución de problemas.

OBJETIVOS

- Recreación del comportamiento recursivo de las torres de Hanoi utilizando tres discos de diferentes tamaños (grande, pequeño y mediano) en un ambiente tridimensional.
- Implementación de la animación (movimiento) de cada disco mediante Keyframes.
- Modelado de los elementos necesarios (pilares, discos, etc) con un software de modelado para su posterior implementación en Visual Studio Code con el uso de las librerías de OpenGL.

DESCRIPCIÓN DEL PROYECTO

El proyecto consistirá en la creación de un escenario, compuesto por los siguientes elementos (Torres de Hanoy), dichos elementos deben ser modelos con extensión obj importados al programa en OpenGL:

a) Torres de Hanoy

1. Tres Cilindros.
2. Tres Toroides
3. Tres Prismas.
4. Un Plano

Características del escenario:

Los tres Cilindros y los tres Prismas formarán las figuras mostradas, es decir, los Prismas servirán como base para los cilindros.

Las dimensiones de los Prismas y los Cilindros, así como la posición de los Cilindros sobre los Prismas quedan a criterio del alumno. Tanto los Prismas como los Cilindros de las bases deben ser dibujados de Color GRIS (que tienda a ser plateado). El plano se encontrará del lado derecho del escenario y deberá estar cubierto por una TEXTURA con el nombre del alumno, las dimensiones del plano quedará a criterio del alumno.

Ya con las Bases construidas. La Base 1 tendrá los tres Toroides, cada uno de diferentes diámetros, uno sobre otro, a manera de formar una pirámide.

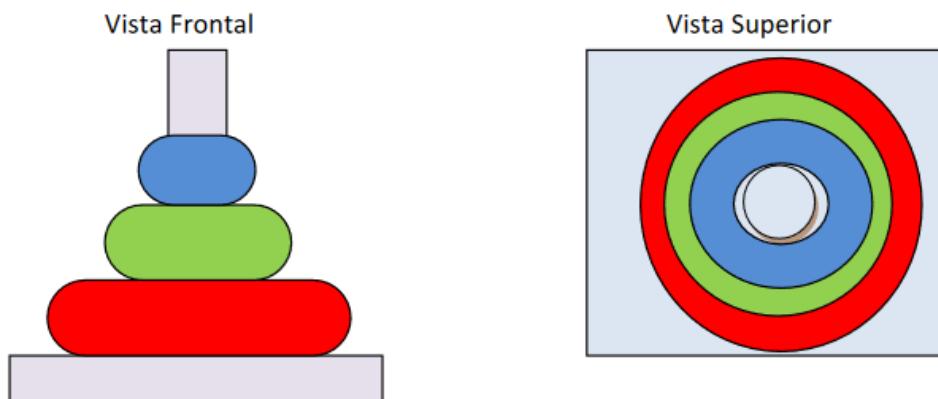


Figura 1: Vista de los cilindros.

Los colores de los Toroides serán los que se indican en los Dibujos, es decir el Toroide 1 (el más grande de diámetro) será color ROJO, el Toroide 2, intermedio de tamaño, será color VERDE, y por último el Toroide 3, el más chico, será color AZUL.

Estos colores serán colocados haciendo uso de Texturas. Se deberán crear las siguientes animaciones mediante código de programación que utilice las transformaciones geométricas básicas.

Paso “1” (uno) de la animación, el Toroide 3 (el más chico) estando en la Base 1, se trasladará sobre el eje Y, dando la impresión de elevarse, y será trasladado a la Base 3, donde será depositado. Ese será el fin de esta animación.

Paso “2” (dos) de la animación, el Toroide 2 (el mediano) estando en la Base 1, se trasladará sobre el eje Y, dando la impresión de elevarse, y será trasladado a la Base 2, donde será depositado. Ese será el fin de esta animación.

Paso “3” (tres) de la animación, el Toroide 3 (el más chico) estando en la Base 3, se trasladará sobre el eje Y, dando la impresión de elevarse, y será trasladado a la Base 2, donde será depositado. Ese será el fin de esta animación.

Paso “4” (cuatro) de la animación, el Toroide 1 (el más grande) estando en la Base 1, se trasladará sobre el eje Y, dando la impresión de elevarse, y será trasladado a la Base 3, donde será depositado. Ese será el fin de esta animación.

Paso “5” (cinco) de la animación, el Toroide 3 (el más chico) estando en la Base 2, se trasladará sobre el eje Y, dando la impresión de elevarse, y será trasladado a la Base 1, donde será depositado. Ese será el fin de esta animación.

Paso “6” (seis) de la animación, el Toroide 2 (el mediano) estando en la Base 2, se trasladará sobre el eje Y, dando la impresión de elevarse, y será trasladado a la Base 3, donde será depositado. Ese será el fin de esta animación.

Paso “7” (siete) de la animación, el Toroide 1 (el más chico) estando en la Base 1, se trasladará sobre el eje Y, dando la impresión de elevarse, y será trasladado a la Base 3, donde será depositado. Ese será el fin de esta animación.

Estas animaciones deberán realizarse en ese orden.

Al final de estas animaciones, los tres toroides se encontrarán, en orden, en la base 3. La animación debe poder repetirse al presionar alguna tecla definida por el alumno.

ESPECIFICACIONES DE DESARROLLO

Para el desarrollo de este proyecto fue necesario el uso de diferentes herramientas para la creación de modelado 3D y software necesario para su manipulación.

C++: C++ es un lenguaje de programación de propósito general que se utiliza ampliamente en el desarrollo de software. Es conocido por ser un lenguaje de programación de nivel medio que combina la programación orientada a objetos con características de programación estructurada. Es utilizado en una variedad de aplicaciones, incluyendo sistemas operativos, juegos, aplicaciones de escritorio, software empresarial y mucho más, debido a su eficiencia y capacidad para trabajar directamente con la memoria del sistema.

Blender 4.0: Blender es un software de código abierto y gratuito utilizado para crear contenido en 3D, incluyendo modelos, animaciones, efectos visuales y más. Es una herramienta versátil que permite modelado, texturizado, animación, simulación, renderizado, composición y edición de video. La versión 4.0 (o cualquier versión específica) indica la versión del software, con actualizaciones y mejoras que pueden incluir nuevas funciones y correcciones de errores.

Visual Studio Code 2022: Visual Studio Code (VS Code) es un entorno de desarrollo integrado (IDE) ligero y gratuito creado por Microsoft. Aunque es una herramienta liviana, ofrece funcionalidades avanzadas para la edición de código, depuración, control de versiones integrado y soporte para varios lenguajes de programación. Es altamente personalizable mediante la instalación de extensiones que amplían sus capacidades según las necesidades del desarrollador.

Open GL: OpenGL (Open Graphics Library) es una especificación estándar que define una API multiplataforma para la renderización de gráficos 2D y 3D. Se utiliza ampliamente en el desarrollo de aplicaciones gráficas, incluyendo juegos, software de diseño asistido por computadora (CAD), visualización científica y más. Proporciona funciones para dibujar primitivas gráficas, aplicar transformaciones, gestionar texturas, iluminación y efectos visuales, siendo compatible con varios sistemas operativos.

CODIFICACIÓN DEL PROYECTO

Para la codificación del proyecto fue necesario en primer lugar realizar los modelos 3D de cada uno de los objetos solicitados en el software Blender 4.0 en las siguientes imágenes se ilustrarán.

Modelo 1: Torre

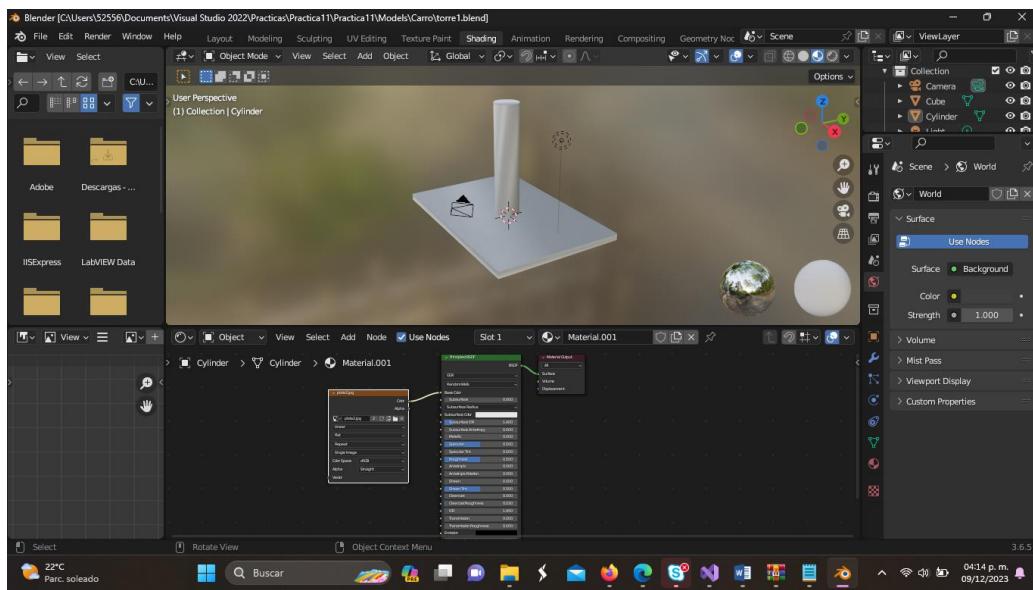


Figura 2: Modelo de Torre en Blender.

Modelo 2: Cilindro pequeño

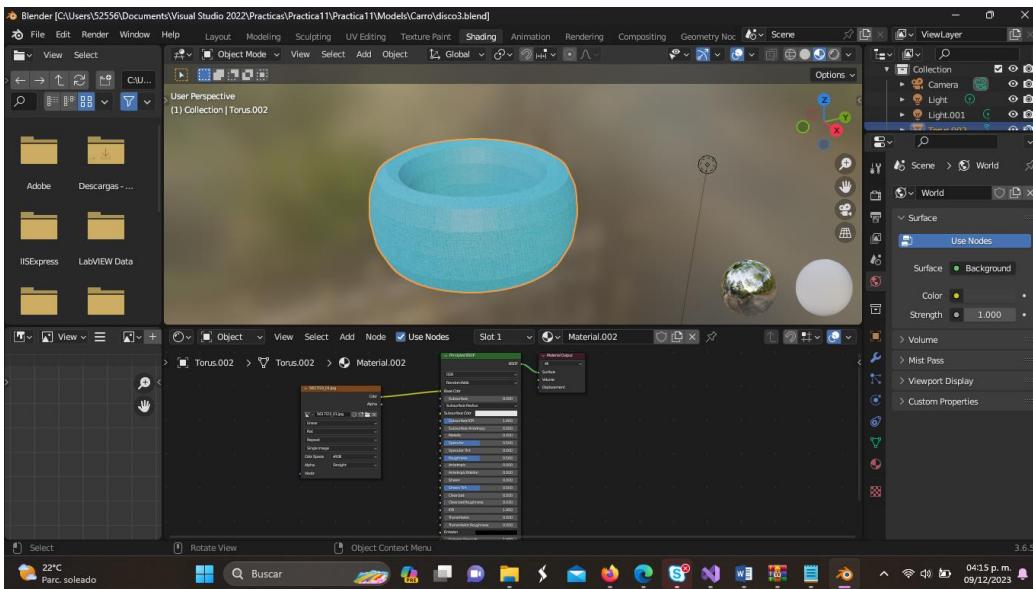


Figura 3: Modelo Cilindro pequeño en Blender.

Modelo 3: Cilindro mediano

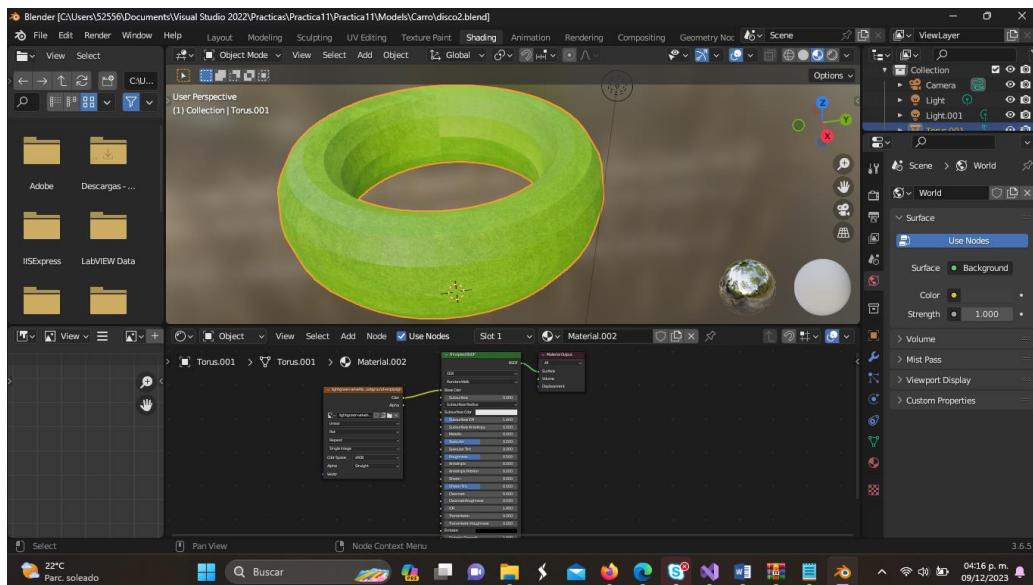


Figura 4: Modelo Cilindro mediano en blender.

Modelo 4: Cilindro grande

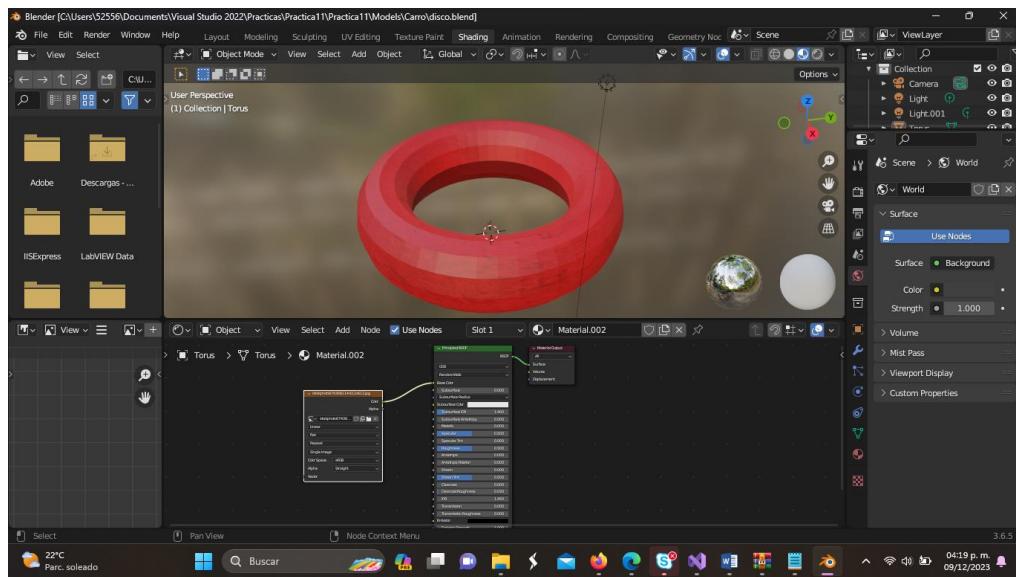


Figura 5: Modelo Cilindro grande en Blender.

Modelo 5: Plano con nombre de la alumna

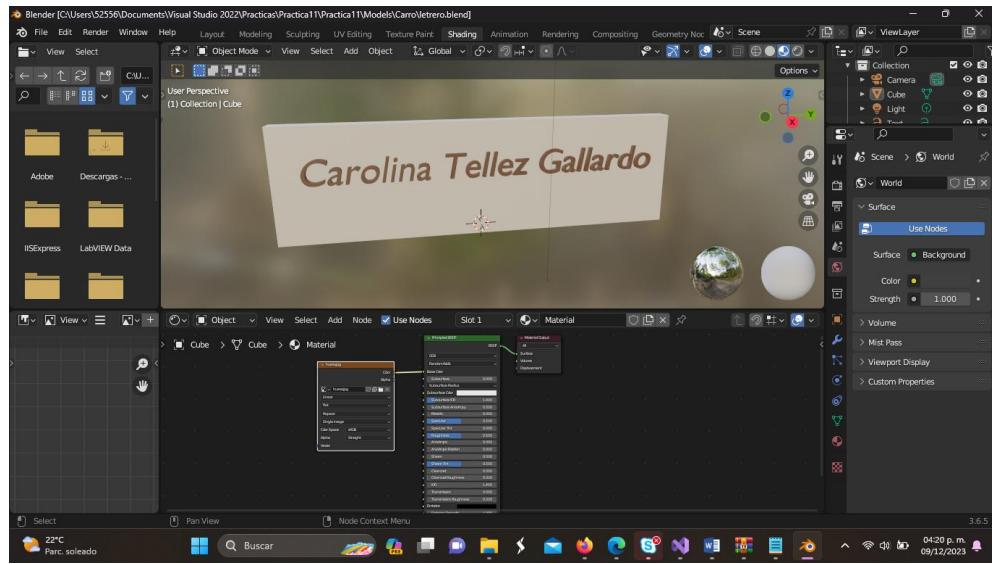


Figura 6: Modelo Plano con nombre en Blender.

Modelo 6: Plano para el escenario

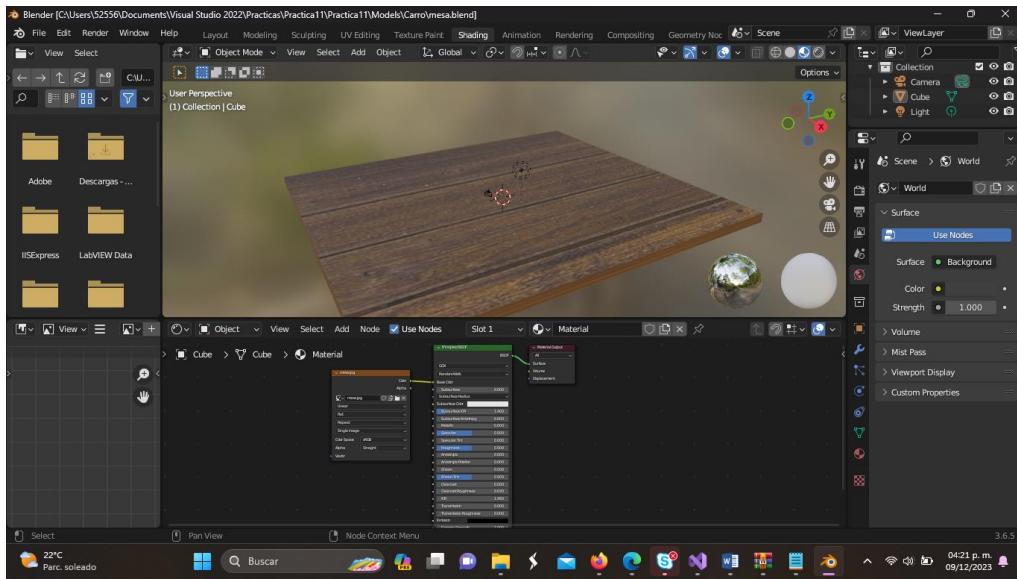


Figura 7: Modelo Plano del escenario en Blender.

El modelo de la figura 7 se agregó a consideración de la alumna ya que se quería dar un ambiente más armonioso al colocar un plano sobre el cual estarán las torres de hanoi con los respectivos discos.

Una vez que se crearon los modelos de los objetos necesarios se comenzó a codificar el proyecto con ayuda de las librerías de OpenGL y Visual Studio Code con el lenguaje de programación C++. A continuación se muestran las capturas del código empleado.

The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** Práctica11
- Editor Area:** The code is written in C++ and includes comments about the university, faculty, and student information. It also imports various OpenGL libraries like `<iostream>`, `<cmath>`, `<GL/glew.h>`, `<GLFW/glfw3.h>`, `<stb_image.h>`, and `<glm/glm.hpp>`.
- Status Bar:** Shows the current file is "Práctica11", the line number is 15, character position is 1, and the encoding is CRLF.
- Output Panel:** Shows the output "El subproceso 0x407c terminó con código 0 (0x0)."
- Taskbar:** Shows the system tray with weather (23°C), notifications, and the date/time (09/12/2023).

Figura 8: Codificación

The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** Práctica11
- Editor Area:** The code continues from the previous snippet, adding declarations for shaders, camera, and window dimensions. It includes prototypes for key and mouse callbacks, and logic for movement and animation.
- Status Bar:** Shows the current file is "Práctica11", the line number is 15, character position is 1, and the encoding is CRLF.
- Output Panel:** Shows the output "El subproceso 0x407c terminó con código 0 (0x0)."
- Taskbar:** Shows the system tray with weather (23°C), notifications, and the date/time (09/12/2023).

Figura 9: Codificación

```

Practica11 (Ámbito global)
54 float rot = 0.0f;
55 // Light attributes
56 glm::vec3 lightPos(0.0f, 0.0f, 0.0f);
57 glm::vec3 PosIni(-95.0f, 0.35f, -45.0f);
58 bool active;
59
60 //POSICIONAMIENTO DE LAS LUCES
61 // Positions of the point lights
62 glm::vec3 pointLightPositions[] = {
63     glm::vec3(0.7f, 0.2f, 2.0f),
64     glm::vec3(2.3f, -3.3f, -4.0f),
65     glm::vec3(-4.0f, 2.0f, -12.0f),
66     glm::vec3(0.0f, 0.0f, -3.0f)
67 };
68
69 glm::vec3 LightP1;
70
71 //En las siguientes lineas se definen las variables que se utilizaran para las animaciones de cada uno de los discos
72 // deltatime
73 GLfloat deltaTime = 0.0f;
74 GLfloat lastFrame = 0.0f;
75
76 //La animacion de las torres de hanoi se implementara mediante keyframes
77 // Keyframes
78 float posX = PosIni.x, posY = PosIni.y, posZ = PosIni.z;
79 float posX1 = PosIni.x, posY1 = PosIni.y, posZ1 = PosIni.z;
80 float posX2 = PosIni.x, posY2 = PosIni.y, posZ2 = PosIni.z;
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109

```

Salida
Mostrar salida de: Depurar
Lista de errores Salida
Lista Agregar al control de código fuente Seleccionar repositorio 04:55 p. m. 09/12/2023

Figura 10: Codificación

En las primeras líneas de código se declaran las bibliotecas que se van a utilizar, así como los shaders necesarios y las funciones que tendrá el proyecto para posteriormente declarar las variables y estructuras necesarias para los keyframes.

```

Practica11 (Ámbito global)
82 //Definimos un maximo de frames que sera guardado para capturar cada posicion de los discos
83 #define MAX_FRAMES 21
84 int i_max_steps = 190;
85 int i_curr_steps = 0;
86
87 //Declaramos una estructura para guardar las posiciones de los discos en los ejes X Y Z
88 typedef struct _frame
89 {
90     //Variables para GUARDAR Key Frames
91
92     float posX; //Variable para PosicionX
93     float posY; //Variable para PosicionY
94     float posZ; //Variable para PosicionZ
95
96     float incX; //Variable para IncrementoX
97     float incY; //Variable para IncrementoY
98     float incZ; //Variable para IncrementoZ
99
100
101
102     float posX1; //Variable para PosicionX
103     float posY1; //Variable para PosicionY
104     float posZ1; //Variable para PosicionZ
105
106     float incX1; //Variable para IncrementoX
107     float incY1; //Variable para IncrementoY
108     float incZ1; //Variable para IncrementoZ
109
110
111
112
113
114
115
116
117
118
119

```

Salida
Mostrar salida de: Depurar
Lista de errores Salida
Lista Agregar al control de código fuente Seleccionar repositorio 04:56 p. m. 09/12/2023

Figura 11: Codificación

```

Practica11 (Ámbito global)

108     float incY1;      //Variable para IncrementoY
109     float incZ1;      //Variable para IncrementoZ
110
111 //-----
112
113     float posX2;      //Variable para PosicionX
114     float posY2;      //Variable para PosicionY
115     float posZ2;      //Variable para PosicionZ
116
117     float incX2;      //Variable para IncrementoX
118     float incY2;      //Variable para IncrementoY
119     float incZ2;      //Variable para IncrementoZ
120
121 }FRAME;
122
123 //Con las siguientes variables vamos a inicializar los keyframes
124 FRAME KeyFrame[MAX_FRAMES]; //Asigno num total de frames
125 int FrameIndex = 0;          //introducir datos
126 bool play = false;
127 int playIndex = 0;
128
129 //La siguiente funcion guardara los keyframes de cada disco
130 void saveFrame(void)
131 {
132
133     printf("frameindex %d\n", FrameIndex);
134
135     KeyFrame[FrameIndex].posX = posX;
136 }
137
138 //No se encontraron problemas.

```

Salida

Mostrar salida de: Depurar

Lista de errores Salida

Lista

23°C Mayorm. nublado Buscar Agregar al control de código fuente Seleccionar repositorio 04:57 p. m. 09/12/2023

Figura 12: Codificación

```

Practica11 (Ámbito global)

153 void resetElements(void)
154 {
155     posX = KeyFrame[0].posX;
156     posY = KeyFrame[0].posY;
157     posZ = KeyFrame[0].posZ;
158
159 //-----
160
161     posX1 = KeyFrame[0].posX1;
162     posY1 = KeyFrame[0].posY1;
163     posZ1 = KeyFrame[0].posZ1;
164
165 //-----
166     posX2 = KeyFrame[0].posX2;
167     posY2 = KeyFrame[0].posY2;
168     posZ2 = KeyFrame[0].posZ2;
169 }
170
171 //Un paso importante de la animacion por keyframes es la interpolacion
172 //esto hara que cada uno de los frames se una para crear un movimiento constante
173 void interpolation(void)
174 {
175
176     KeyFrame[playIndex].incX = (KeyFrame[playIndex + 1].posX - KeyFrame[playIndex].posX) / i_max_steps;
177     KeyFrame[playIndex].incY = (KeyFrame[playIndex + 1].posY - KeyFrame[playIndex].posY) / i_max_steps;
178     KeyFrame[playIndex].incZ = (KeyFrame[playIndex + 1].posZ - KeyFrame[playIndex].posZ) / i_max_steps;
179
180 }
181
182 //No se encontraron problemas.

```

Salida

Mostrar salida de: Depurar

Lista de errores Salida

Lista

23°C Mayorm. nublado Buscar Agregar al control de código fuente Seleccionar repositorio 04:57 p. m. 09/12/2023

Figura 13: Codificación

Una vez declaradas las variables que se van a utilizar es necesario comenzar a codificar las funciones necesarias para realizar la interpolación de los frames así como nuestra función principal.

The screenshot shows a Windows desktop environment. In the center is a code editor window titled "Practica11" displaying C++ code. The code includes calculations for frame interpolation and the main function setup for GLFW. Below the code editor is a "Salida" (Output) window showing no errors or warnings. At the bottom is a taskbar with various icons and system status indicators.

```
181 KeyFrame[playIndex].incX1 = (KeyFrame[playIndex + 1].posX1 - KeyFrame[playIndex].posX1) / i_max_steps;
182 KeyFrame[playIndex].incY1 = (KeyFrame[playIndex + 1].posY1 - KeyFrame[playIndex].posY1) / i_max_steps;
183 KeyFrame[playIndex].incZ1 = (KeyFrame[playIndex + 1].posZ1 - KeyFrame[playIndex].posZ1) / i_max_steps;
184
185 //-
186 KeyFrame[playIndex].incX2 = (KeyFrame[playIndex + 1].posX2 - KeyFrame[playIndex].posX2) / i_max_steps;
187 KeyFrame[playIndex].incY2 = (KeyFrame[playIndex + 1].posY2 - KeyFrame[playIndex].posY2) / i_max_steps;
188 KeyFrame[playIndex].incZ2 = (KeyFrame[playIndex + 1].posZ2 - KeyFrame[playIndex].posZ2) / i_max_steps;
189
190 }
191
192 //DECLARACION DE LA FUNCION MAIN
193 int main()
194 {
195     // Init GLFW
196     glfwInit();
197     // Set all the required options for GLFW
198     glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
199     glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
200     glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
201     glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
202     glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
203
204     // Create a GLFWwindow object that we can use for GLFW's functions
```

Figura 14: Codificación

The screenshot shows a Windows desktop environment. In the center is a code editor window titled "Practica11" displaying C++ code for GLFW initialization. The code handles window creation, context setting, and GLEW initialization. Below the code editor is a "Salida" (Output) window showing no errors or warnings. At the bottom is a taskbar with various icons and system status indicators.

```
211 if (nullptr == window)
212 {
213     std::cout << "Failed to create GLFW window" << std::endl;
214     glfwTerminate();
215
216     return EXIT_FAILURE;
217 }
218
219 glfwMakeContextCurrent(window);
220
221 glfwGetFramebufferSize(window, &SCREEN_WIDTH, &SCREEN_HEIGHT);
222
223 // Set the required callback functions
224 glfwSetKeyCallback(window, KeyCallback);
225 glfwSetCursorPosCallback(window, MouseCallback);
226 printf("%f", glfwGetTime());
227
228 // GLFW Options
229 glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);
230
231 // Set this to true so GLEW knows to use a modern approach to retrieving function pointers and extensions
232 glewExperimental = GL_TRUE;
233 // Initialize GLEW to setup the OpenGL Function pointers
234 if (GLEW_OK != glewInit())
235 {
236     std::cout << "Failed to initialize GLEW" << std::endl;
237     return EXIT_FAILURE;
238 }
```

Figura 15: Codificación

```

239 // Define the viewport dimensions
240 glViewport(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT);
241
242 // OpenGL options
243 glEnable(GL_DEPTH_TEST);
244
245 //LLAMADO A LOS SHADERS
246 //Para este proyecto es necesario el uso de shaders
247 Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting.frag");
248 Shader lampShader("Shaders/lamp.vs", "Shaders/lamp.frag");
249
250 //A continuacion se declaran los modelos a utilizar
251 //Estos modelos fueron creados con la ayuda del software Blender
252 Model mesa((char*)"Models/Carro/mesa.obj");
253 Model torreI((char*)"Models/Carro/torreI.obj");
254 Model disco((char*)"Models/Carro/disco.obj");
255 Model disco2((char*)"Models/Carro/disco2.obj");
256 Model disco3((char*)"Models/Carro/disco3.obj");
257 Model letrero((char*)"Models/Carro/letrero.obj");
258
259
260 //Inicialización de KeyFrames
261 for (int i = 0; i < MAX_FRAMES; i++)
262 {
263     KeyFrame[i].posX = 0;
264     KeyFrame[i].incX = 0;
265     KeyFrame[i].incY = 0;
266 }

```

No se encontraron problemas.

Figura 16: Codificación

Una de las partes más importantes del código son las líneas donde se declaran los modelos que se van a utilizar, esto para posteriormente posicionar el modelo en el escenario 3D con ayuda de las transformaciones básicas.

```

269 //-
270 KeyFrame[i].posX1 = 0;
271 KeyFrame[i].incX1 = 0;
272 KeyFrame[i].incY1 = 0;
273 KeyFrame[i].incZ1 = 0;
274
275 //-----
276 KeyFrame[i].posX2 = 0;
277 KeyFrame[i].incX2 = 0;
278 KeyFrame[i].incY2 = 0;
279 KeyFrame[i].incZ2 = 0;
280
281
282
283 // Set up vertex data (and buffer(s)) and attribute pointers
284 GLfloat vertices[] =
285 {
286     // Positions           // Normals           // Texture Coords
287     -0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f,
288     0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f,
289     0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
290     0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f,
291     -0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
292     -0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f,
293     -0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f,
294
295     -0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
296     0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f,

```

No se encontraron problemas.

Figura 17: Codificación

```
Practica11                                     (Ámbito global)
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
};

GLuint indices[] =
{ // Note that we start from 0!
    0,1,2,3,
    4,5,6,7,
    8,9,10,11,
    12,13,14,15,
    16,17,18,19,
    20,21,22,23,
    24,25,26,27,
    28,29,30,31,
    32,33,34,35
};
```

Figura 18: Codificación

```
Practica11 (Ámbito global)
344 // Positions all containers
345 glm::vec3 cubePositions[] = {
346     glm::vec3(0.6f,  0.6f,  0.6f),
347     glm::vec3(-0.6f,  0.6f,  0.6f),
348     glm::vec3(0.6f, -0.6f,  0.6f),
349     glm::vec3(-0.6f, -0.6f,  0.6f),
350     glm::vec3(0.6f,  0.6f, -0.6f),
351     glm::vec3(-0.6f,  0.6f, -0.6f),
352     glm::vec3(0.6f, -0.6f, -0.6f),
353     glm::vec3(-0.6f, -0.6f, -0.6f),
354     glm::vec3(0.0f,  0.6f,  0.6f),
355     glm::vec3(0.0f, -0.6f,  0.6f),
356 };
357
358
359 // First, set the container's VAO (and VBO)
360 GLuint VBO, VAO, EBO;
361 glGenVertexArrays(1, &VAO);
362 glGenBuffers(1, &VBO);
363 glGenBuffers(1, &EBO);
364
365 glBindVertexArray(VAO);
366 glBindBuffer(GL_ARRAY_BUFFER, VBO);
367 glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
368
369 glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
370 glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);

Linea: 15 | Carácter: 1 | OVR | TABULACIONES | CRLF
98 %
No se encontraron problemas.
```

Figura 19: Codificación

```
Practica1 (Ámbito global)
388     glEnableVertexAttribArray(2);
389     glBindVertexArray(0);
390
391     // 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))
392     GLuint lightVAO;
393     glGenVertexArrays(1, &lightVAO);
394     glBindVertexArray(lightVAO);
395     // 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.
396     glBindBuffer(GL_ARRAY_BUFFER, VBO);
397     // Set the vertex attributes (only position data for the lamp)
398     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)0); // Note that we skip over the other data in our buffer object (we don't need the
399     glEnableVertexAttribArray(0);
400     glBindVertexArray(0);
401
402     // Load textures
403     GLuint texture1, texture2;
404     glGenTextures(1, &texture1);
405     glGenTextures(1, &texture2);
406
407     int textureWidth, textureHeight, nrChannels;
408     stbi_set_flip_vertically_on_load(true);
409     unsigned char* image;
410     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
411     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
412     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
413     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST_MIPMAP_NEAREST);
414
415     // Diffuse map
416     image = stbi_load("images/TexturesCom_GravelCobble0019_7.S.jpg", &textureWidth, &textureHeight, &nrChannels, 0);
417
98 % 98 % No se encontraron problemas. > Línea: 15 Carácter: 1 OVR TABULACIONES CRLF
```

Figura 20: Codificación

```
Practica11 (Ámbito global) main()
491 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"), LightP1.x, LightP1.y, LightP1.z);
492 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].constant"), 1.0f);
493 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].linear"), 0.09f);
494 glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"), 0.032f);
495
496 //DECLARACION DE LAS LUces POINLIGHTS Y SPOTLIGHT
497
498 // Point light 2
499 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].position"), pointLightPositions[1].x, pointLightPositions[1].y, pointLightPositions[1].z);
500 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].ambient"), 0.05f, 0.05f, 0.05f);
501 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].diffuse"), 1.0f, 1.0f, 0.0f);
502 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].specular"), 1.0f, 1.0f, 0.0f);
503 glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].constant"), 1.0f);
504 glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].linear"), 0.09f);
505 glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].quadratic"), 0.032f);
506
507 // Point light 3
508 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].position"), pointLightPositions[2].x, pointLightPositions[2].y, pointLightPositions[2].z);
509 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].ambient"), 0.05f, 0.05f, 0.05f);
510 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].diffuse"), 0.0f, 1.0f, 1.0f);
511 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].specular"), 0.0f, 1.0f, 1.0f);
512 glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].constant"), 1.0f);
513 glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].linear"), 0.09f);
514 glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].quadratic"), 0.032f);
515
516 // Point light 4
517 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].position"), pointLightPositions[3].x, pointLightPositions[3].y, pointLightPositions[3].z);
518 glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].ambient"), 0.05f, 0.05f, 0.05f);
```

Figura 21: Codificación

Una de las partes más importantes del diseño de nuestro escenario es la posición de las luces para que nuestros modelos se vean iluminados en este proyecto se utilizaron los tres tipos de luces spotlight, pointlight y specular.

```
Practica11 | (Ámbito global) | main()
```

```
525 // SpotLight
526 glUniform3f(glGetUniformLocation(lightingShader.Program, "spotlight.position"), camera.GetPosition().x, camera.GetPosition().y, camera.GetPosition().z);
527 glUniform3f(glGetUniformLocation(lightingShader.Program, "spotlight.direction"), camera.GetFront().x, camera.GetFront().y, camera.GetFront().z);
528 glUniform3f(glGetUniformLocation(lightingShader.Program, "spotlight.ambient"), 0.0f, 0.0f, 0.0f);
529 glUniform3f(glGetUniformLocation(lightingShader.Program, "spotlight.diffuse"), 0.0f, 0.0f, 0.0f);
530 glUniform3f(glGetUniformLocation(lightingShader.Program, "spotlight.specular"), 0.0f, 0.0f, 0.0f);
531 glUniform1f(glGetUniformLocation(lightingShader.Program, "spotlight.constant"), 1.0f);
532 glUniform1f(glGetUniformLocation(lightingShader.Program, "spotlight.linear"), 0.09f);
533 glUniform1f(glGetUniformLocation(lightingShader.Program, "spotlight.quadratic"), 0.032f);
534 glUniform1f(glGetUniformLocation(lightingShader.Program, "spotlight.cutOff"), glm::cos(glm::radians(12.5f)));
535 glUniform1f(glGetUniformLocation(lightingShader.Program, "spotlight.outerCutOff"), glm::cos(glm::radians(15.0f)));
536
537 // Set material properties
538 glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 32.0f);
539
540 // Create camera transformations
541 glm::mat4 view;
542 view = camera.GetViewMatrix();
543
544 // Get the uniform locations
545 GLint modelLoc = glGetUniformLocation(LightingShader.Program, "model");
546 GLint viewLoc = glGetUniformLocation(LightingShader.Program, "view");
547 GLint projLoc = glGetUniformLocation(LightingShader.Program, "projection");
548
549 // Pass the matrices to the shader
550 glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
551 glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
```

Figura 22: Codificación

Practica11

(Ámbito global) main()

```
556 //EN LAS SIGUIENTES LINEAS A TRAVES DE LAS TRANSFORMACIONES BASICAS SE COLOCAN LOS OBJETOS EN EL ESPACIO 3D
557 //Colocare una mesa para que el escenario se vea mas ordenado
558
559 //Colocar mesa
560 model = glm::mat4(1);
561 model = glm::translate(model, glm::vec3(-58.0f, 1.0f, -40.5f));
562 model = glm::scale(model, glm::vec3(3.5f, 1.0f, 3.5f));
563 glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
564 mesa.Draw(LightingShader);
565
566 //Colocar pilar
567 model = glm::mat4(1);
568 model = glm::translate(model, glm::vec3(-110.0f, 2.0f, -75.5f));
569 model = glm::scale(model, glm::vec3(2.3f, 3.2f, 2.3f));
570 glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
571 torre1.Draw(LightingShader);
572
573 //Colocar segundo pilar
574 model = glm::mat4(1);
575 model = glm::translate(model, glm::vec3(-80.0f, 2.0f, -75.5f));
576 model = glm::scale(model, glm::vec3(2.3f, 3.2f, 2.3f));
577 glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
578 torre1.Draw(LightingShader);
579
580 //Colocar tercer pilar
581 model = glm::mat4(1);
582 model = glm::translate(model, glm::vec3(-50.0f, 2.0f, -75.5f));
583
```

No se encontraron problemas.

Línea: 520 | Carácter: 84 | Columna: 90 | OVR | TABULACIONES | CR LF

Salida

Mostrar salida de: Depurar

Lista de errores Salida

Agregar al control de código fuente Seleccionar repositorio

16°C Nublado

Buscar

WPS Office

Google Chrome

Microsoft Edge

Microsoft Word

Microsoft Excel

Microsoft Powerpoint

Microsoft OneDrive

Microsoft Outlook

Microsoft Word

Microsoft Excel

Microsoft Powerpoint

Microsoft OneDrive

Microsoft Outlook

0845 p. m.
09/12/2023

Figura 23: Codificación

Como se mencionaba anteriormente, es necesario hacer uso de las transformaciones básicas para posicionar de forma correcta los objetos en el escenario tridimensional.

```

589 //Se colocaran los toroides en la base 1 en su posicion inicial
590 //Toroide 1 toroide rojo
591 model = glm::mat4(1);
592 model = glm::translate(model, glm::vec3(posX, posY, posZ));
593 model = glm::translate(model, glm::vec3(-15.5f, 3.0f, -30.5f));
594 //Con la siguiente transformacion vamos a mover el toroide 1
595 model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
596 model = glm::scale(model, glm::vec3(2.3f, 3.2f, 2.3f));
597 glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
598 disco1.Draw(lightingShader);
599
600 //Toroide2 toroide verde
601 model = glm::mat4(1);
602 model = glm::translate(model, glm::vec3(-15.5f, 4.0f, -30.5f));
603 //Con la siguiente transformacion vamos a mover el toroide 2
604 model = glm::translate(model, glm::vec3(posX1, posY1, posZ1));
605 model = glm::scale(model, glm::vec3(2.3f, 3.2f, 2.3f));
606 glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
607 disco2.Draw(lightingShader);
608
609 //Toroide 3 toroide azul
610 model = glm::mat4(1);
611 model = glm::translate(model, glm::vec3(posX2, posY2, posZ2));
612 model = glm::translate(model, glm::vec3(-15.5f, 5.0f, -30.5f));
613 //Con la siguiente transformacion vamos a mover el toroide 2
614 model = glm::scale(model, glm::vec3(2.3f, 3.2f, 2.3f));
615 glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
616 disco3.Draw(lightingShader);

```

No se encontraron problemas.

Figura 24: Codificación

```

619 //En las siguientes lineas se colocara el plano con el nombre de la alumna
620 model = glm::mat4(1);
621 model = glm::translate(model, glm::vec3(-30.0f, 35.0f, -75.5f));
622 model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
623 model = glm::scale(model, glm::vec3(2.3f, 3.2f, 2.3f));
624 glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
625 letrero.Draw(lightingShader);
626
627 glBindVertexArray(0);
628
629 // Also draw the lamp object, again binding the appropriate shader
630 lampShader.Use();
631 // Get location objects for the matrices on the lamp shader (these could be different on a different shader)
632 modelLoc = glGetUniformLocation(lampShader.Program, "model");
633 viewLoc = glGetUniformLocation(lampShader.Program, "view");
634 projLoc = glGetUniformLocation(lampShader.Program, "projection");
635
636 // Set matrices
637 glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
638 glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
639 model = glm::mat4(1);
640 model = glm::translate(model, lightPos);
641 //model = glm::scale(model, glm::vec3(0.2f)); // Make it a smaller cube
642 glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
643 // Draw the light object (using light's vertex attributes)
644 glBindVertexArray(lightVAO);
645 for (GLuint i = 0; i < 4; i++)

```

No se encontraron problemas.

Figura 25: Codificación

```
Practica11.cpp (Ámbito global) main()

645     for (GLuint i = 0; i < 4; i++)
646     {
647         model = glm::mat4(1);
648         model = glm::translate(model, pointLightPositions[i]);
649         model = glm::scale(model, glm::vec3(0.2f)); // Make it a smaller cube
650         glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
651         glDrawArrays(GL_TRIANGLES, 0, 36);
652     }
653     glBindVertexArray(0);

654     // Swap the screen buffers
655     glfwSwapBuffers(window);
656 }

657
658
659
660     glDeleteVertexArrays(1, &VAO);
661     glDeleteVertexArrays(1, &lightVAO);
662     glDeleteBuffers(1, &VBO);
663     glDeleteBuffers(1, &EBO);
664     // Terminate GLFW, clearing any resources allocated by GLFW.
665     glfwTerminate();
666
667
668
669
670     return 0;
671
672
```

Salida
Mostrar salida de: Depurar
Lista de errores: Salida
Lista
16°C Nublado Buscar Agregar al control de código fuente Seleccionar repositorio 08:46 p. m. 09/12/2023

Figura 26: Codificación

```
Practica11.cpp (Ámbito global) main()

672     //Esta función es la encargada de mover a través de los ejes correspondientes X Y Z cada uno de los toroides
673     void DoMovement()
674     {
675         //Movimiento Toroide2 azul C V E R
676
677         if (keys[GLFW_KEY_C])
678         {
679             posX2 -= 0.1;
680         }
681
682         if (keys[GLFW_KEY_V])
683         {
684             posX2 += 0.1;
685         }
686
687         if (keys[GLFW_KEY_E])
688         {
689             posY2 -= 0.1;
690         }
691
692         if (keys[GLFW_KEY_R])
693         {
694             posY2 += 0.1;
695         }
696
697         //Movimiento Toroide2 verde Z Y I U
698
699     }
```

Salida
Mostrar salida de: Depurar
Lista de errores: Salida
Lista
16°C Nublado Buscar Agregar al control de código fuente Seleccionar repositorio 08:47 p. m. 09/12/2023

Figura 27: Codificación

The screenshot shows a code editor window with the following code:

```
859 ////FUNCION ANIMACIÓN
860 //En esta función se va a hacer el conteo de los frames (pasos) de cada uno de los discos
861 void animacion()
862 {
863     //Movimiento del personaje
864
865     if (play)
866     {
867         if (i_curr_steps >= i_max_steps) //end of animation between frames?
868         {
869             playIndex++;
870             if (playIndex > FrameIndex - 2) //end of total animation?
871             {
872                 printf("termina anim\n");
873                 playIndex = 0;
874                 play = false;
875             }
876             else //Next frame interpolations
877             {
878                 i_curr_steps = 0; //Reset counter
879                 //Interpolation
880                 interpolation();
881             }
882         }
883     }
884 }
885
886 //draw_animation
```

The code is part of a C program named 'Practica11'. It defines a function 'animacion()' which handles character movement and animation logic. It uses global variables like 'play', 'i_curr_steps', 'i_max_steps', 'playIndex', 'FrameIndex', and 'interpolation()'.

Figura 28: Codificación

The screenshot shows a code editor window with the following code:

```
885
886
887     //Draw animation
888     posX += KeyFrame[playIndex].incX;
889     posY += KeyFrame[playIndex].incY;
890     posZ += KeyFrame[playIndex].incZ;
891
892     posX1 += KeyFrame[playIndex].incX1;
893     posY1 += KeyFrame[playIndex].incY1;
894     posZ1 += KeyFrame[playIndex].incZ1;
895
896     posX2 += KeyFrame[playIndex].incX2;
897     posY2 += KeyFrame[playIndex].incY2;
898     posZ2 += KeyFrame[playIndex].incZ2;
899
900     i_curr_steps++;
901
902 }
903
904 //En la siguiente función se establecen las teclas con las cuales se van a guardar cada uno de los frames
905 //Asimismo se establece la tecla con la que se va a ejecutar la animación
906
907 //void KeyCallback(GLFWwindow * window, int key, int scancode, int action, int mode)
908 //{
909 //    if (keys[GLFW_KEY_L])
910 //    {
911
912 }
```

The code is part of a C program named 'Practica11'. It includes a main loop that updates character positions based on keyframes and handles keyboard input via a key callback function.

Figura 29: Codificación

The screenshot shows a Windows operating system desktop. In the center is a code editor window titled "Practica11" with the file "main()". The code is written in C++ and includes comments explaining the purpose of certain sections. The code editor has a status bar at the bottom showing "Línea: 520" and "Carácter: 84". Below the code editor is a taskbar with various icons for applications like File Explorer, Microsoft Edge, and others. The system tray shows the date and time as "09/12/2023 08:48 p.m.". A weather widget indicates "16°C Nublado".

```
959     active = !active;
960     if (active)
961         LightP1 = glm::vec3(1.0f, 0.0f, 0.0f);
962     else
963         LightP1 = glm::vec3(0.0f, 0.0f, 0.0f);
964     }
965
966
967 //Por ultimo tenemos una funcion que movera la camara en cuanto a orientacion mediante el raton (mouse)
968 void MouseCallback(GLFWwindow* window, double xPos, double yPos)
969 {
970
971     if (firstMouse)
972     {
973         lastX = xPos;
974         lastY = yPos;
975         firstMouse = false;
976     }
977
978     GLfloat xOffset = xPos - lastX;
979     GLfloat yOffset = lastY - yPos; // Reversed since y-coordinates go from bottom to left
980
981     lastX = xPos;
982     lastY = yPos;
983
984     camera.ProcessMouseMovement(xOffset, yOffset);
985
986 }
```

Figura 30: Codificación

En las últimas líneas de código se declaran las funciones de movimiento de la cámara y las funciones de animación, en las cuales se utilizaron distintas teclas para guardar y ejecutar los frames de cada disco.

SALIDAS

Une vez explicado el código fuente del proyecto podemos ejecutar y visualizar la salida que obtenemos en las siguientes imágenes.

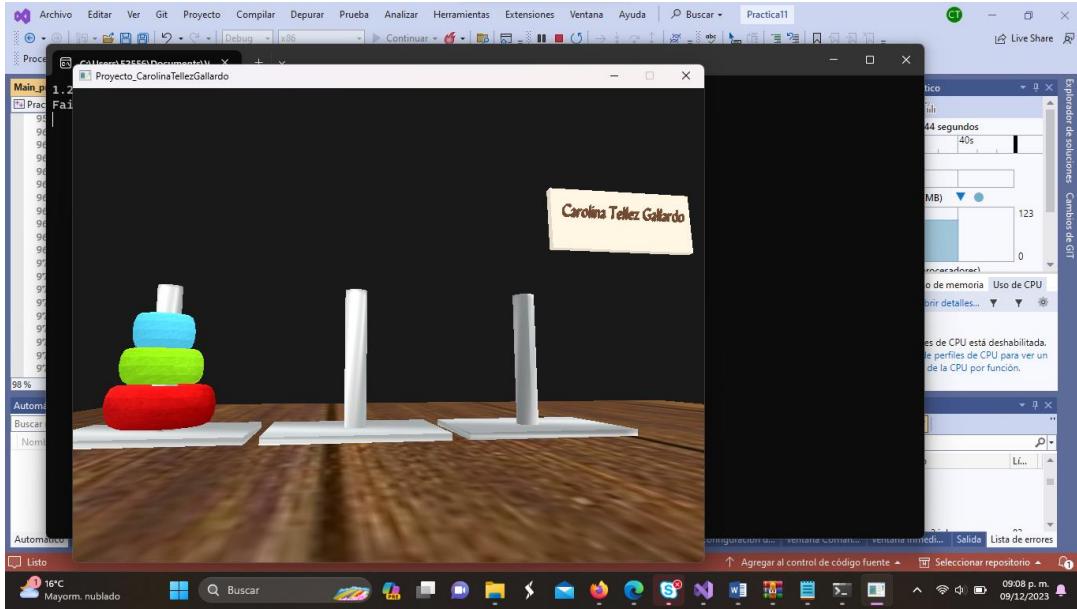


Figura 31: Salida del proyecto.

Como se puede apreciar en la figura 31 se tiene un escenario con las torres de hanoi y de nuestro lado derecho un cartel con el nombre completo de la alumna como se solicita en la descripción del proyecto.

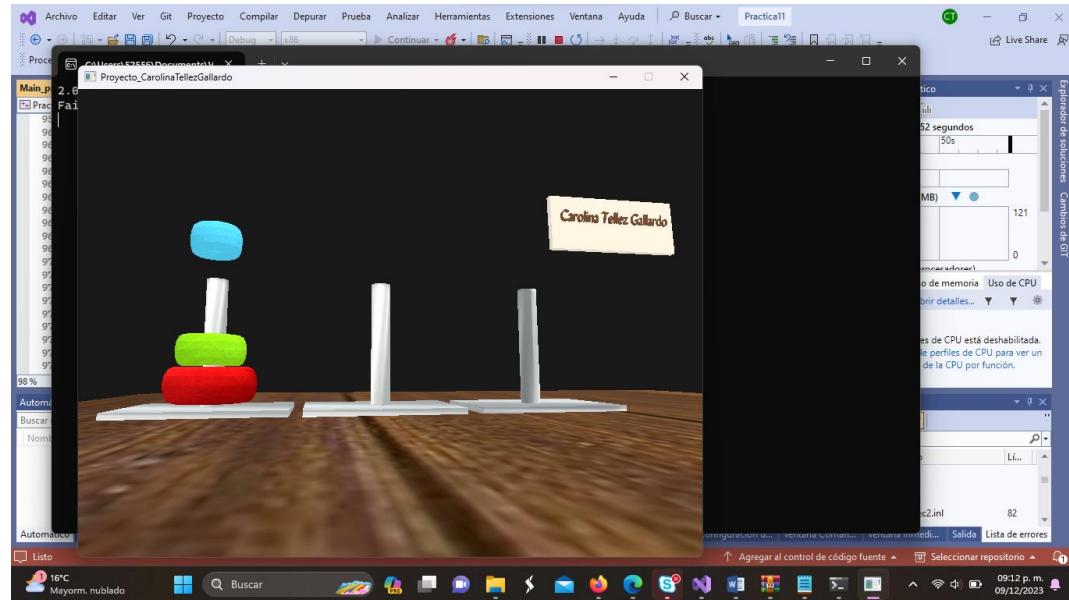


Figura 32: Movimiento del toroide azul.

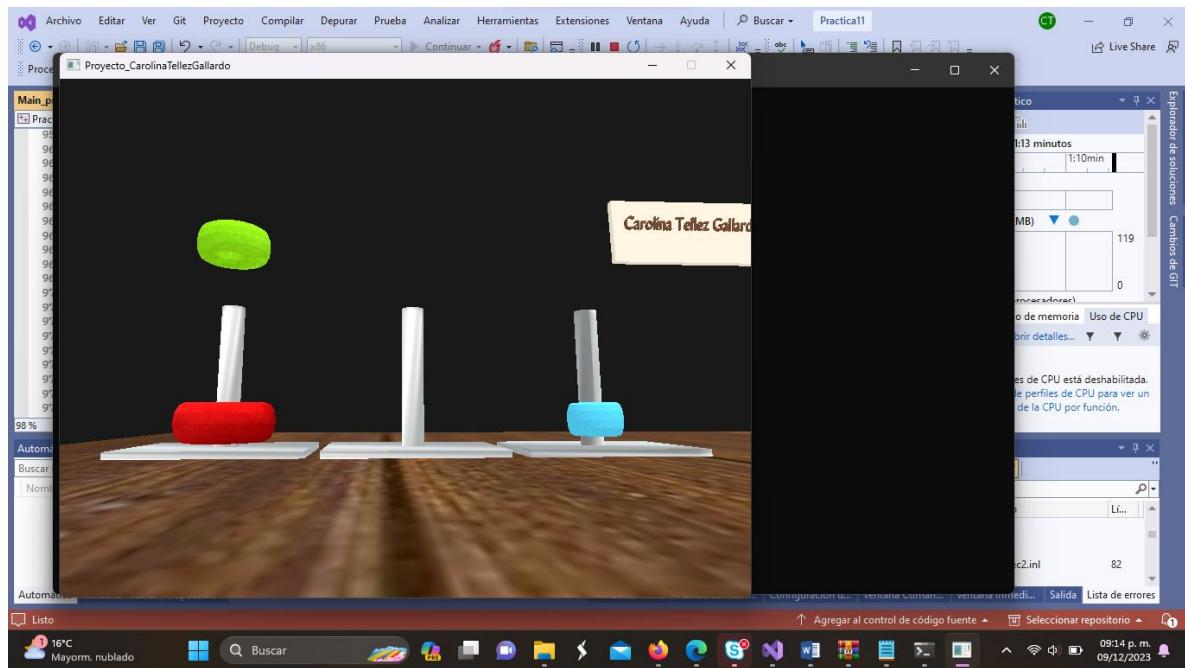


Figura 33: Movimiento del toroide verde.

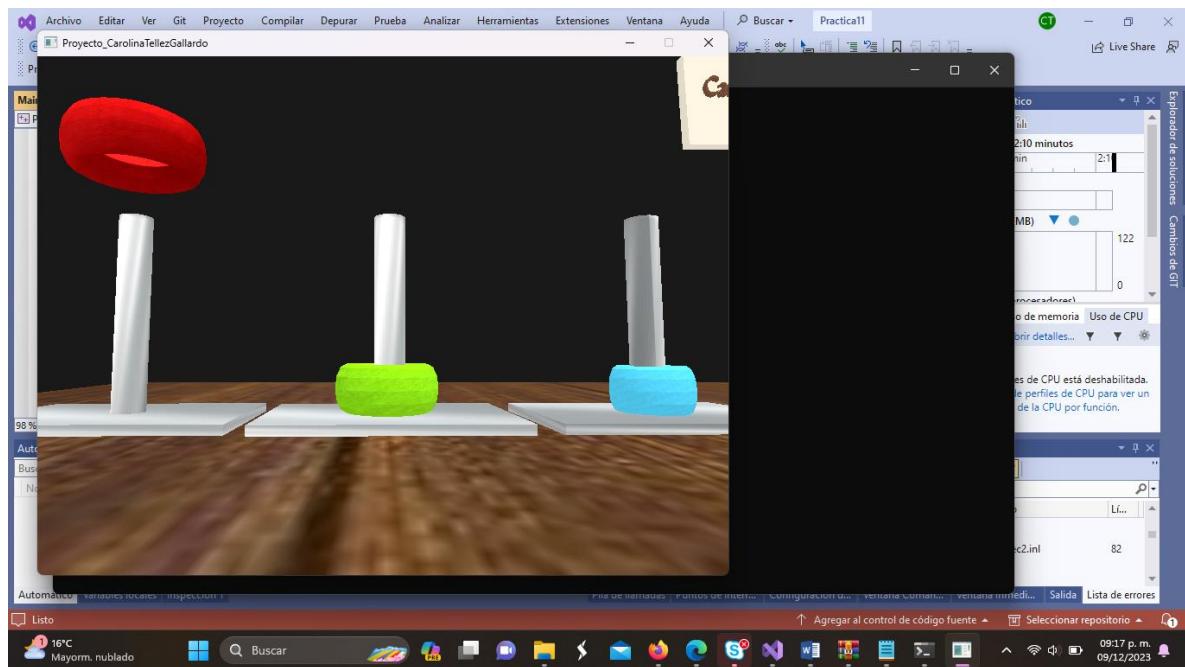


Figura 34: Movimiento del toroide rojo.

RECURSOS DE ACCESO

El proyecto se puede obtener desde Github a través del siguiente repositorio:

https://github.com/Caro1504/Proyecto_TorresDeHanoi_CarolinaTellezGallardo.git

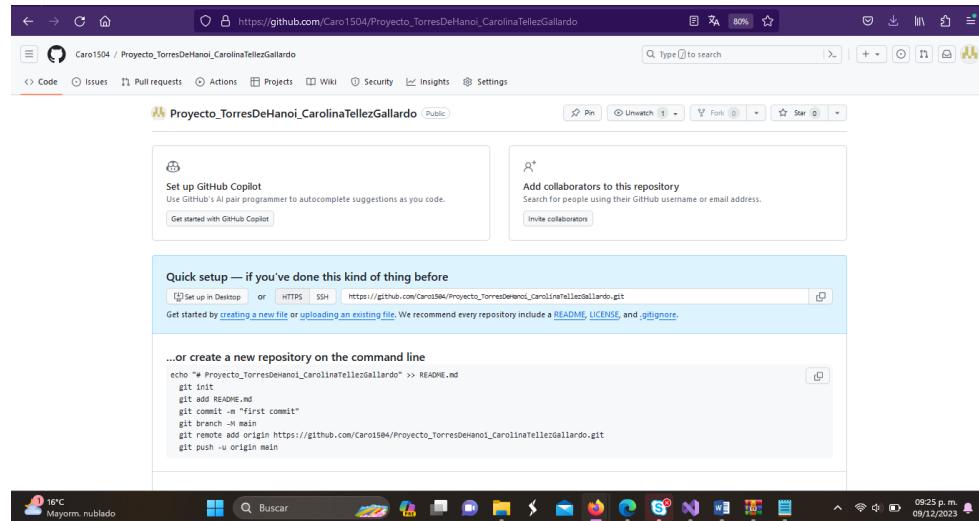


Figura 35: Repositorio de Github

La liga al video de la animación la podemos descargar desde Google Drive:

https://drive.google.com/drive/u/0/folders/1nkygDrnWAFTG9h-1wsxv_u-x74pOGbuZ

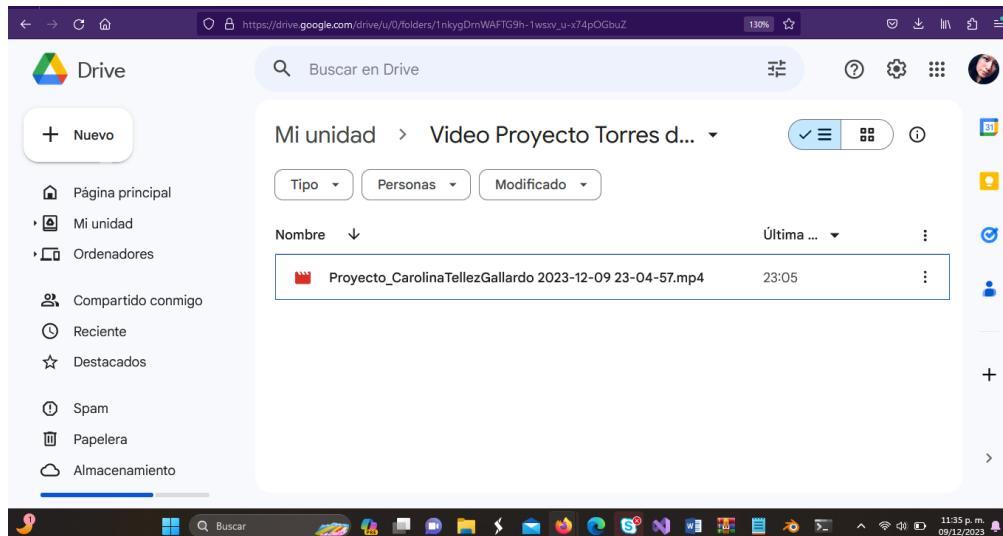


Figura 36: Video en Drive

CONCLUSIONES

En este proyecto, se buscó emular los movimientos asociados a las Torres de Hanoi, involucrando tres discos y creando un entorno tridimensional mediante el uso de herramientas de desarrollo gráfico y modelado 3D. Empleando los conocimientos adquiridos en el campo de la computación gráfica, se logró programar las secuencias de animación para los tres discos del rompecabezas de las Torres de Hanoi. Esta implementación se llevó a cabo utilizando el método de animación basado en Keyframes, el cual considero apropiado para representar estos movimientos, aunque existen diversas formas de animación disponibles. El resultado de este proyecto fue satisfactorio y permitirá al usuario comprender de manera más clara los movimientos asociados a este relevante problema matemático.

BIBLIOGRAFÍA DE CONSULTA

- [1] Sin Autor (Sin Fecha) Khan Academy ‘*Torres de Hanoi*’ Recuperado de: <https://es.khanacademy.org/computing/computer-science/algorithms/towers-of-hanoi/a/towers-of-hanoi>
- [2] Karmakar, D. (2023) freecode ‘*Como resolver el problema de la torre de Hanoi: Una guía ilustrada del algoritmo*’. Recuperado de: <https://www.freecodecamp.org/espanol/news/como-resolver-el-problema-de-la-torre-de-hanoi-una-guia-ilustrada-del-algoritmo/>