

Universidad Nacional Autónoma de México

Facultad de Ingeniería

Fossil Land



Desarrollador:

Mario Alberto Vásquez Cancino

v. 1.0.0

Índice

Objetivo	2
Plan de Trabajo	2
Alcance del Proyecto	3
Funcionamiento	4
Instalación y Ejecución	4
Código Fuente	4
Estructura	4
Fossil_Land.sln	4
External Libraries	5
Fossil_Land	5
Código	6
Costos	19
Conclusión	19

Agradecimientos especiales a los siguientes sitios donde se obtuvieron algunas texturas, modelos y demás materiales ocupados en este proyectos:

www.turbosquid.com

www.cgtrader.com

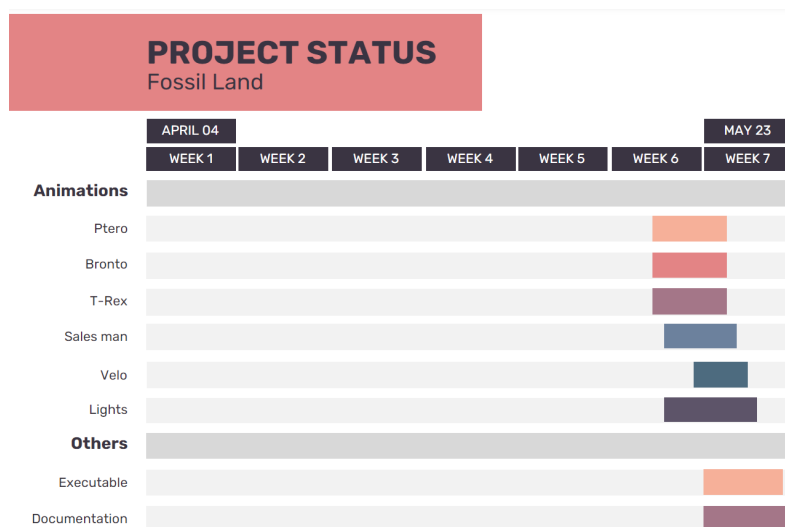
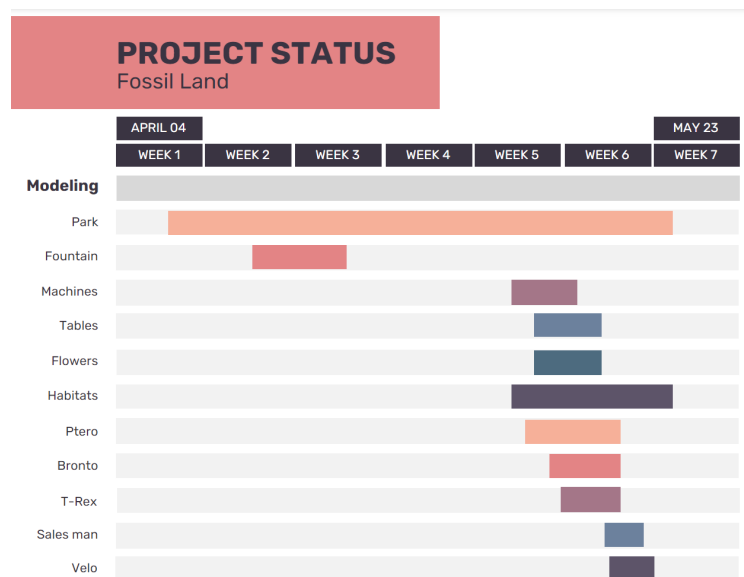
cpetry.github.io/NormalMap-Online

www.filterforge.com

Objetivo

Ofrecer un recorrido virtual en una sencilla recreación de un parque jurásico “Fossil Land”.

Plan de Trabajo



Alcance del Proyecto

Al finalizar este proyecto se debe obtener un programa que permita al usuario recorrer una recreación virtual sencilla del parque jurásico “Fossil Land”.

La recreación virtual del parque jurásico se basará según la propuesta realizada donde se menciona la siguiente distribución del parque:



Distribución del parque propuesta

De los dinosaurios seleccionados para el parque son:

- Brontosaurio
- Pterodáctilo
- Velociraptor
- T-Rex

También se contará con 5 animaciones, las cuales son:

- Pterodáctilo volando.
- Dinosaurio rondando.
- Brontosaurio comiendo.
- Velociraptor bailando la macarena.
- Persona vendiendo boletos.

Para que el usuario pueda tener una interacción con el programa, se debe desplazarse mediante las teclas W,S,A y D, mover la vista con el mouse, detener y reproducir las animaciones con la barra espaciadora y controlar la luz interna del cuarto.

El proyecto finalizado con su documentación, archivo ejecutable y binarios deberá ser entregado a más tardar el lunes 23 de mayo de 2022.

Limitaciones

Debido a que es una recreación, al tiempo de desarrollo y los recursos disponibles el parque jurásico, las animaciones y elementos modelados no tendrán un 100% de parecido con respecto a lo propuesto, pero serán muy similares y se acercaran lo mas posible a lo pensado.

Funcionamiento




Instalación y Ejecución

Para correr la simulación véase el Manual de Usuario.

Código Fuente

Estructura

El proyecto se realizó con Visual Studio por lo que en la carpeta raíz encontramos los siguientes elementos principales:



 External Libraries	30/03/2022 15:40	Carpeta de archivos	
 Fossil_Land	14/04/2022 11:59	Carpeta de archivos	
 Fossil_Land.sln	14/04/2022 11:59	Visual Studio Solution	2 KB

Fossil_Land.sln

Solución de Visual Studio para manejar el proyecto.

External Libraries



















Librerías ocupadas en el proyecto.

 assimp	30/03/2022 15:41	Carpeta de archivos
 GLEW	30/03/2022 15:40	Carpeta de archivos
 GLFW	30/03/2022 15:40	Carpeta de archivos
 glm	30/03/2022 15:41	Carpeta de archivos
 SOIL2	30/03/2022 15:41	Carpeta de archivos

- **ASSIMP:** Librería que nos servirá para cargar modelos o escenas 3D almacenados en gran variedad de formatos.
- **GLEW:** Es una librería multiplataforma de código abierto escrita en C/C++ la cual nos proveerá de un mecanismo en tiempo de ejecución para determinar cuál de las funciones opengl están soportadas en la plataforma de destino.
- **GLFW:** Librería que nos permitirá crear y administrar las ventanas donde desplegamos los gráficos OpenGL.
- **GLM:** Librería nos brinda acceso a funciones matemáticas comúnmente utilizada en los gráficos 3D
- **SOIL:** Es una pequeña librería para cargar imágenes en 6 tipos de formatos.

Fossil_Land

Carpeta que contiene los archivos y recursos principales del proyecto.

 Models	14/04/2022 12:03	Carpeta de archivos	
 Shaders	30/03/2022 15:41	Carpeta de archivos	
 SkyBox	30/03/2022 15:41	Carpeta de archivos	
 SOIL2	30/03/2022 15:41	Carpeta de archivos	
 assimp-vc140-mt.dll	07/04/2019 23:07	Extensión de la aplica...	15,705 KB
 Camera.h	31/03/2019 1:51	C Include File	5 KB
 Fossil_Land.vcxproj	14/04/2022 12:06	VC++ Project	11 KB
 Fossil_Land.vcxproj.filters	14/04/2022 12:06	VC++ Project Filters F...	1 KB
 Fossil_Land.vcxproj.user	14/04/2022 11:59	Per-User Project Opti...	1 KB
 glew32.dll	31/07/2017 21:42	Extensión de la aplica...	381 KB
 Main.cpp	21/05/2022 9:29	Archivo CPP	42 KB
 Mesh.h	08/04/2019 0:21	C Include File	4 KB
 meshAnim.h	15/03/2022 21:49	C Include File	7 KB
 Model.h	08/04/2019 3:15	C Include File	8 KB
 modelAnim.h	15/03/2022 21:49	C Include File	27 KB
 Shader.h	25/03/2019 3:38	C Include File	4 KB
 stb_image.h	09/01/2019 6:03	C Include File	249 KB
 Texture.h	15/03/2022 21:49	C Include File	3 KB

- **images:** Carpeta de imágenes que se ocuparon en el proyecto.
- **Models:** Carpeta con todos los modelos que se ocupan en el proyecto.
- **Shaders:** Carpeta que contiene todos los shaders para la luz, animaciones, etc.
- **Skybox:** Carpeta con las imágenes para el Skybox.
- **SOIL2:** Librería para cargar imágenes en 6 tipos de formatos.
- **Main.cpp:** Código principal del proyecto.
- **assimp-vc140-mt.dll:** Archivo para manejar la librería ASSIMP.
- **Camera.h:** Archivo de cabecera para controlar la cámara.
- **glew32.dll:** Librería de OpenGL Extension Wrangler.
- **Mesh.h y meshAnim.h:** Archivos de cabecera para manejar las mallas para las texturas de los modelos.
- **Model.h y modelAnim.h:** Archivos de cabecera para manejar los modelos juntos a sus mallas y texturas.
- **Shader.h:** Archivo de cabecera para manejar los shaders.
- **stb_image.h:** Librería para cargar imágenes hecha por Sean Barret.
- **Texture.h:** Archivo de cabecera para cargar las texturas del SkyBox

Código

Abriendo el archivo principal del proyecto (Main.cpp) encontramos:

Importación de las librerías que ocuparemos.

```
#include <iostream>
#include <cmath>

// GLEW
#include <GL/glew.h>

// GLFW
#include <GLFW/glfw3.h>

// Other Libs
#include "stb_image.h"

// GLM Mathematics
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>

//Load Models
#include "SOIL2/SOIL2.h"

// Other includes
#include "Shader.h"
#include "Camera.h"
#include "Model.h"
#include "Texture.h" /*SKYBOX*/
```

© 2006 The Authors

```

// Animation Params
glm::vec3 PositionVec(0.0f, 0.0f, 0.0f);
glm::vec3 directionVec(0.5f, 0.5f, 1.1f);
glm::vec3 correctionVec(0.5f, 0.5f, -1.1f);
float moveZ = 0.0;
float moveX12 = 0.0;
float rotateZ = 90.0f;
bool canMove = true;
bool canMove2 = false;
bool canMove3 = false;
bool patax = false;
float rotateX = 0.0f;
float rotateY = 0.0f;

// Animation Params
bool brace = false;
float outFrame = 0.0f;

// Positions of the point lights
// Definición de un Lightmap por el id num de point lights y agregar sus posiciones aquí
glm::vec3 pointPosition[10] = {
    {0.0f, 0.0f, 7.5f, 0.7f}
};

glm::vec3 Light = glm::vec3(0);

// Delatates
float delatates = 0.0f; // Time between current frame and last frame
float lastFrame = 0.0f; // Time of last frame

// Movement
float posX = PosMainID.x, posY = PosMainID.y, posZ = PosMainID.z, rotX = 0;
float posY2 = PosMainID.x, posY1 = PosMainID.y, posZ1 = PosMainID.z, rotY1 = 0, rotY2 = 0;
float posX2 = PosMainID.x, posY2 = PosMainID.y, posZ2 = PosMainID.z, rotX2 = 0, rotY2 = 0;
float posX3 = PosMainID.x, posY3 = PosMainID.y, posZ3 = PosMainID.z, rotX3 = 0, rotY3 = 0;

```

7


```

void resetElements(void)
{
    posX = KeyFrame[0].posX;
    posY = KeyFrame[0].posY;
    posZ = KeyFrame[0].posZ;

    rotHI = KeyFrame[0].rotHI;

    posXBI = KeyFrame[0].posXBI;
    posYBI = KeyFrame[0].posYBI;
    posZBI = KeyFrame[0].posZBI;

    rotBI = KeyFrame[0].rotBI;
    rotBIX = KeyFrame[0].rotBIX;

    posXHD = KeyFrame[0].posXHD;
    posYHD = KeyFrame[0].posYHD;
    posZHD = KeyFrame[0].posZHD;

    rotHD = KeyFrame[0].rotHD;

    posXBD = KeyFrame[0].posXBD;
    posYBD = KeyFrame[0].posYBD;
    posZBD = KeyFrame[0].posZBD;

    rotBD = KeyFrame[0].rotBD;
    rotBOX = KeyFrame[0].rotBOX;
}

void interpolation(void)
{
    KeyFrame[playIndex].incX = (KeyFrame[playIndex + 1].posX - KeyFrame[playIndex].posX) / i_max_steps;
    KeyFrame[playIndex].incY = (KeyFrame[playIndex + 1].posY - KeyFrame[playIndex].posY) / i_max_steps;
    KeyFrame[playIndex].incZ = (KeyFrame[playIndex + 1].posZ - KeyFrame[playIndex].posZ) / i_max_steps;

    KeyFrame[playIndex].rotIncHI = (KeyFrame[playIndex + 1].rotHI - KeyFrame[playIndex].rotHI) / i_max_steps;

    KeyFrame[playIndex].incXBI = (KeyFrame[playIndex + 1].posXBI - KeyFrame[playIndex].posXBI) / i_max_steps;
    KeyFrame[playIndex].incYBI = (KeyFrame[playIndex + 1].posYBI - KeyFrame[playIndex].posYBI) / i_max_steps;
    KeyFrame[playIndex].incZBI = (KeyFrame[playIndex + 1].posZBI - KeyFrame[playIndex].posZBI) / i_max_steps;

    KeyFrame[playIndex].rotIncBI = (KeyFrame[playIndex + 1].rotBI - KeyFrame[playIndex].rotBI) / i_max_steps;
    KeyFrame[playIndex].rotIncBIX = (KeyFrame[playIndex + 1].rotBIX - KeyFrame[playIndex].rotBIX) / i_max_steps;

    KeyFrame[playIndex].incXHD = (KeyFrame[playIndex + 1].posXHD - KeyFrame[playIndex].posXHD) / i_max_steps;
    KeyFrame[playIndex].incYHD = (KeyFrame[playIndex + 1].posYHD - KeyFrame[playIndex].posYHD) / i_max_steps;
    KeyFrame[playIndex].incZHD = (KeyFrame[playIndex + 1].posZHD - KeyFrame[playIndex].posZHD) / i_max_steps;

    KeyFrame[playIndex].rotIncHD = (KeyFrame[playIndex + 1].rotHD - KeyFrame[playIndex].rotHD) / i_max_steps;

    KeyFrame[playIndex].incXBD = (KeyFrame[playIndex + 1].posXBD - KeyFrame[playIndex].posXBD) / i_max_steps;
    KeyFrame[playIndex].incYBD = (KeyFrame[playIndex + 1].posYBD - KeyFrame[playIndex].posYBD) / i_max_steps;
    KeyFrame[playIndex].incZBD = (KeyFrame[playIndex + 1].posZBD - KeyFrame[playIndex].posZBD) / i_max_steps;

    KeyFrame[playIndex].rotIncBD = (KeyFrame[playIndex + 1].rotBD - KeyFrame[playIndex].rotBD) / i_max_steps;
    KeyFrame[playIndex].rotIncBOX = (KeyFrame[playIndex + 1].rotBOX - KeyFrame[playIndex].rotBOX) / i_max_steps;
}

```

Dentro de la función main encontramos:

Inicialización de GLFW, las opciones de OpenGL y creación de la ventana.

```

// Init GLFW
glfwInit();
// Set all the required options for GLFW
glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);

// Create a GLFWwindow object that we can use for GLFW's functions
GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "Fossil Land", nullptr, nullptr);

if (nullptr == window)
{
    std::cout << "Failed to create GLFW window" << std::endl;
    glfwTerminate();

    return EXIT_FAILURE;
}

glfwMakeContextCurrent(window);

glfwGetFramebufferSize(window, &SCREEN_WIDTH, &SCREEN_HEIGHT);

// Set the required callback functions
glfwSetKeyCallback(window, KeyCallback);
glfwSetCursorPosCallback(window, MouseCallback);

// GLFW Options
glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);

// Set this to true so GLEW knows to use a modern approach to retrieving function pointers and extensions
glewExperimental = GL_TRUE;
// Initialize GLEW to setup the OpenGL function pointers
if (GLEW_OK != glewInit())
{
    std::cout << "Failed to initialize GLEW" << std::endl;
    return EXIT_FAILURE;
}

// Define the viewport dimensions
glViewport(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT);

```

Importación de los modelos y shaders..

```
// Setup and compile our shaders
Shader lightingShader("Shaders/Lighting.vs", "Shaders/Lighting.frag");
Shader lampShader("Shaders/lamp.vs", "Shaders/lamp.frag");
Shader SkyBoxShader("Shaders/SkyBox.vs", "Shaders/SkyBox.frag"); /* SKYBOX */

/* Parque */
Model piso ((char*)"Models/Piso/Piso.obj");
Model habitats ((char*)"Models/Piso/habitats2.obj");
Model caja ((char*)"Models/Piso/Caja.obj");
Model barrotes ((char*)"Models/Piso/barrotes.obj");
Model lampara ((char*)"Models/Piso/lampara.obj");
Model lamparas ((char*)"Models/Piso/streetlamp.obj");
Model mateo ((char*)"Models/Piso/mateo.obj");

Model fuente ((char*)"Models/Sodas/Fuente/FuenteC.obj");
Model fuenteAgua ((char*)"Models/Sodas/Fuente/FuenteA.obj");
Model mesa ((char*)"Models/Sodas/Mesa/mesa.obj");
Model maquinas ((char*)"Models/Sodas/Maquina/maquinas.obj");

/* Dinosaurios */
Model brontoCa ((char*)"Models/Dinosaurios/Bronto/brontoCabeza.obj");
Model brontoCu ((char*)"Models/Dinosaurios/Bronto/brontoCuerpo.obj");

Model pteroCu ((char*)"Models/Dinosaurios/Ptero/pteroCu.obj");
Model pteroAlaD ((char*)"Models/Dinosaurios/Ptero/pteroAlaD.obj");
Model pteroAlaI ((char*)"Models/Dinosaurios/Ptero/pteroAlaI.obj");

Model veloCu ((char*)"Models/Dinosaurios/Velo/veloCu.obj");
Model veloHomD ((char*)"Models/Dinosaurios/Velo/veloHombreDer.obj");
Model veloBrad ((char*)"Models/Dinosaurios/Velo/veloBrazoDer.obj");
Model veloHomI ((char*)"Models/Dinosaurios/Velo/veloHombreIzq.obj");
Model veloBrai ((char*)"Models/Dinosaurios/Velo/veloBrazoIzq.obj");

Model tRexCu ((char*)"Models/Dinosaurios/Trex/T_rexCu.obj");
Model tRexPD ((char*)"Models/Dinosaurios/Trex/T_rexPiernaD.obj");
Model tRexPI ((char*)"Models/Dinosaurios/Trex/T_rexPiernaI.obj");

Model personaCu ((char*)"Models/Persona/personaCu.obj");
Model personaBra ((char*)"Models/Persona/personaBraI.obj");
```

Inicialización de los KeyFrames para la animación del velociraptor.

```
//Inicialización de KeyFrames
for (int i = 0; i < MAX_FRAMES; i++)
{
    KeyFrame[i].posX = 0;
    KeyFrame[i].incX = 0;
    KeyFrame[i].incY = 0;
    KeyFrame[i].incZ = 0;
    KeyFrame[i].rotHI = 0;
    KeyFrame[i].rotInHI = 0;
}

// Guardando animación KEYFRAMES 0
KeyFrame[0].posXHD = posXHD;
KeyFrame[0].posYHD = posYHD;
KeyFrame[0].posZHD = posZHD;
KeyFrame[0].rotHD = rotHD;
KeyFrame[0].rotBDX = rotBDX;

KeyFrame[0].posX = posX;
KeyFrame[0].posY = posY;
KeyFrame[0].posZ = posZ;
KeyFrame[0].rotHI = rotHI;

KeyFrame[0].posXBI = posXBI;
KeyFrame[0].posYBI = posYBI;
KeyFrame[0].posZBI = posZBI;
KeyFrame[0].rotBI = rotBI;
KeyFrame[0].rotBIX = rotBIX;

// Guardando animación KEYFRAMES 1
KeyFrame[1].posXHD = posXHD;
KeyFrame[1].posYHD = posYHD;
KeyFrame[1].posZHD = posZHD;
KeyFrame[1].rotHD = -90.0f;

KeyFrame[1].posXBD = posXBD;
KeyFrame[1].posYBD = posYBD;
KeyFrame[1].posZBD = posZBD;
KeyFrame[1].rotBD = -90.0f;
KeyFrame[1].rotBDX = 0.0f;

KeyFrame[1].posX = posX;
KeyFrame[1].posY = posY;
KeyFrame[1].posZ = posZ;
KeyFrame[1].rotHI = 0.0f;

KeyFrame[1].posXBI = posXBI;
KeyFrame[1].posYBI = posYBI;
KeyFrame[1].posZBI = posZBI;
KeyFrame[1].rotBI = 0.0f;
KeyFrame[1].rotBIX = 0.0f;
```

```
// Guardando animación KEYFRAMES 2
KeyFrame[2].posXHD = posXHD;
KeyFrame[2].posYHD = posYHD;
KeyFrame[2].posZHD = posZHD;
KeyFrame[2].rotHD = -90.0f;

KeyFrame[2].posXBD = posXBD;
KeyFrame[2].posYBD = posYBD;
KeyFrame[2].posZBD = posZBD;
KeyFrame[2].rotBD = -90.0f;
KeyFrame[2].rotBDX = 0.0f;

KeyFrame[2].posX = posX;
KeyFrame[2].posY = posY;
KeyFrame[2].posZ = posZ;
KeyFrame[2].rotHI = -90.0f;

KeyFrame[2].posXBI = posXBI;
KeyFrame[2].posYBI = posYBI;
KeyFrame[2].posZBI = posZBI;
KeyFrame[2].rotBI = -90.0f;
KeyFrame[2].rotBIX = 0.0f;

// Guardando animación KEYFRAMES 3
KeyFrame[3].posXHD = posXHD;
KeyFrame[3].posYHD = posYHD;
KeyFrame[3].posZHD = posZHD;
KeyFrame[3].rotHD = -90.0f;

KeyFrame[3].posXBD = posXBD;
KeyFrame[3].posYBD = posYBD;
KeyFrame[3].posZBD = posZBD - 0.1;
KeyFrame[3].rotBD = -90.0f;
KeyFrame[3].rotBDX = -15.0f;

KeyFrame[3].posX = posX;
KeyFrame[3].posY = posY;
KeyFrame[3].posZ = posZ;
KeyFrame[3].rotHI = -90.0f;

KeyFrame[3].posXBI = posXBI;
KeyFrame[3].posYBI = posYBI;
KeyFrame[3].posZBI = posZBI;
KeyFrame[3].rotBI = -90.0f;
KeyFrame[3].rotBIX = 0.0f;
```

```
// Guardando animación KEYFRAMES 4
KeyFrame[4].posXHD = posXHD;
KeyFrame[4].posYHD = posYHD;
KeyFrame[4].posZHD = posZHD;
KeyFrame[4].rotHD = -90.0f;

KeyFrame[4].posXBD = posXBD;
KeyFrame[4].posYBD = posYBD;
KeyFrame[4].posZBD = posZBD - 0.1;
KeyFrame[4].rotBD = -90.0f;
KeyFrame[4].rotBDX = -15.0f;

KeyFrame[4].posX = posX;
KeyFrame[4].posY = posY;
KeyFrame[4].posZ = posZ;
KeyFrame[4].rotHI = -90.0f;

KeyFrame[4].posXBI = posXBI;
KeyFrame[4].posYBI = posYBI;
KeyFrame[4].posZBI = posZBI + 0.1;
KeyFrame[4].rotBI = -90.0f;
KeyFrame[4].rotBIX = 15.0f;

// Guardando animación KEYFRAMES 5
KeyFrame[5].posXHD = posXHD;
KeyFrame[5].posYHD = posYHD;
KeyFrame[5].posZHD = posZHD;
KeyFrame[5].rotHD = -120.0f;

KeyFrame[5].posXBD = posXBD;
KeyFrame[5].posYBD = posYBD;
KeyFrame[5].posZBD = posZBD;
KeyFrame[5].rotBD = -120.0f;
KeyFrame[5].rotBDX = 0.0f;

KeyFrame[5].posX = posX;
KeyFrame[5].posY = posY;
KeyFrame[5].posZ = posZ;
KeyFrame[5].rotHI = -90.0f;

KeyFrame[5].posXBI = posXBI;
KeyFrame[5].posYBI = posYBI;
KeyFrame[5].posZBI = posZBI + 0.1;
KeyFrame[5].rotBI = -90.0f;
KeyFrame[5].rotBIX = 15.0f;
```

```
// Guardando animación KEYFRAMES 6
KeyFrame[6].posXHD = posXHD;
KeyFrame[6].posYHD = posYHD;
KeyFrame[6].posZHD = posZHD;
KeyFrame[6].rotHD = -120.0f;

KeyFrame[6].posXBD = posXBD;
KeyFrame[6].posYBD = posYBD;
KeyFrame[6].posZBD = posZBD;
KeyFrame[6].rotBD = -120.0f;
KeyFrame[6].rotBDX = 0.0f;

KeyFrame[6].posX = posX;
KeyFrame[6].posY = posY;
KeyFrame[6].posZ = posZ;
KeyFrame[6].rotHI = -120.0f;

KeyFrame[6].posXBI = posXBI;
KeyFrame[6].posYBI = posYBI;
KeyFrame[6].posZBI = posZBI;
KeyFrame[6].rotBI = -120.0f;
KeyFrame[6].rotBIX = 0.0f;

// Guardando animación KEYFRAMES 7
KeyFrame[7].posXHD = posXHD;
KeyFrame[7].posYHD = posYHD;
KeyFrame[7].posZHD = posZHD;
KeyFrame[7].rotHD = 0.0f;

KeyFrame[7].posXBD = posXBD;
KeyFrame[7].posYBD = posYBD;
KeyFrame[7].posZBD = posZBD;
KeyFrame[7].rotBD = -0.0f;
KeyFrame[7].rotBDX = 0.0f;

KeyFrame[7].posX = posX;
KeyFrame[7].posY = posY;
KeyFrame[7].posZ = posZ;
KeyFrame[7].rotHI = 0.0f;

KeyFrame[7].posXBI = posXBI;
KeyFrame[7].posYBI = posYBI;
KeyFrame[7].posZBI = posZBI;
KeyFrame[7].rotBI = 0.0f;
KeyFrame[7].rotBIX = 0.0f;
```


En main nos encontraremos con GAME LOOP que es una ciclo que se repetirá hasta que se cierre la ventana para poder refrescar la imagen y actualizar los cambios hechos, en pocas palabras poder movernos en la simulación, ver las luces y animaciones.

Configuración de la luz ambiental, de la luz que posee la cámara y de la luz interna de la casa.

[illegible]

Se dibujan los modelos en su posición específica y con las configuraciones necesarias para que sean transparentes o no y si van a estar en movimiento o fijos.

```

glm::mat4 model(1);

//Carga de modelo
view = camera.GetViewMatrix(1);

/* PARQUE */
/* Piso */
model = glm::mat4(1);
glm::UniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glm::Uniform1i(glm::UniformLocation(LightingShader.Program, "activaTransparencia"), 0);
piso.Draw(LightingShader);

/* Lampara */
model = glm::mat4(1);
glm::UniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glm::Uniform1i(glm::UniformLocation(LightingShader.Program, "activaTransparencia"), 0);
lampara.Draw(LightingShader);

/* Lamparas */
model = glm::mat4(1);
glm::UniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glm::Uniform1i(glm::UniformLocation(LightingShader.Program, "activaTransparencia"), 0);
lamparas.Draw(LightingShader);

/* Mateso */
model = glm::mat4(1);
glm::UniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glm::Uniform1i(glm::UniformLocation(LightingShader.Program, "activaTransparencia"), 0);
mateso.Draw(LightingShader);

/* Habitats */
model = glm::mat4(1);
glm::UniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glm::Uniform1i(glm::UniformLocation(LightingShader.Program, "activaTransparencia"), 0);
habitats.Draw(LightingShader);

/* Caja */
model = glm::mat4(1);
glm::UniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glm::Uniform1i(glm::UniformLocation(LightingShader.Program, "activaTransparencia"), 0);
caja.Draw(LightingShader);

/* Barrotes */
model = glm::mat4(1);
glm::UniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glm::Uniform1i(glm::UniformLocation(LightingShader.Program, "activaTransparencia"), 0);
barrotes.Draw(LightingShader);

```

```

/* Fuente */
glEnable(GL_BLEND); //Activa la funcionalidad para trabajar el canal alfa
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 1);
glUniform4f(glGetUniformLocation(LightingShader.Program, "colorAlpha"), 1.0, 1.0, 1.0, 1.0);
FuenteAqua.Draw(LightingShader);
glDisable(GL_BLEND);
glUniform4f(glGetUniformLocation(LightingShader.Program, "colorAlpha"), 1.0, 1.0, 1.0, 1.0);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
Fuente.Draw(LightingShader);

/* Mesas */
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-39.0f, 0.0f, 20.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
mesa.Draw(LightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-34.0f, 0.0f, 20.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
mesa.Draw(LightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-39.0f, 0.0f, 27.5f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
mesa.Draw(LightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-34.0f, 0.0f, 27.5f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
mesa.Draw(LightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-39.0f, 0.0f, 35.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
mesa.Draw(LightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-34.0f, 0.0f, 35.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
mesa.Draw(LightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-39.0f, 0.0f, 42.5f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
mesa.Draw(LightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-34.0f, 0.0f, 42.5f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
mesa.Draw(LightingShader);

```

```

/* Maquinas */
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-36.0f, 0.0f, 47.9f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
maquinas.Draw(LightingShader);

/* DINOSAURIOS */

/* Brontosaurio */
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(36.0f, 0.0f, movBronto));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
brontoCu.Draw(LightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(36.0f, 0.0f, 40.0f));
model = glm::rotate(model, glm::radians(rotCuello), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
brontoCa.Draw(LightingShader);

/* Pterodactilo */
model = glm::mat4(1);
model = glm::translate(model, PosIniPtero + glm::vec3(movKitX, 0, movKitZ));
model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
pteroCu.Draw(LightingShader);
model = glm::mat4(1);
model = glm::translate(model, PosIniPtero + glm::vec3(movKitX, 0, movKitZ));
model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(rotAlaD), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
pteroAlaD.Draw(LightingShader);
model = glm::mat4(1);
model = glm::translate(model, PosIniPtero + glm::vec3(movKitX, 0, movKitZ));
model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(rotAlaI), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
pteroAlaI.Draw(LightingShader);

```

```

/* Velociraptor */
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(36.0f, 0.0f, -40.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
veloCu.Draw(LightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(posXHD, posYHD, posZHD));
model = glm::rotate(model, glm::radians(rotHD), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
veloHomD.Draw(LightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(posXBD, posYBD, posZBD));
model = glm::rotate(model, glm::radians(rotBD), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::rotate(model, glm::radians(rotBDX), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
veloBraD.Draw(LightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(rotHI), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
veloHomI.Draw(LightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(posXBI, posYBI, posZBI));
model = glm::rotate(model, glm::radians(rotBI), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::rotate(model, glm::radians(rotBIX), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
veloBraI.Draw(LightingShader);

/* T-Rex */
model = glm::mat4(1);
model = glm::translate(model, PosIniTrex + glm::vec3(movKitX2, 0, movKitZ2));
model = glm::rotate(model, glm::radians(rotKit2), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
tRexCu.Draw(LightingShader);
model = glm::mat4(1);
model = glm::translate(model, PosIniTrex + correccionD + glm::vec3(movKitX2, 0, movKitZ2));
model = glm::rotate(model, glm::radians(rotKit2), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(rotPataD), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
tRexPD.Draw(LightingShader);
model = glm::mat4(1);
model = glm::translate(model, PosIniTrex + correccionI + glm::vec3(movKitX2, 0, movKitZ2));
model = glm::rotate(model, glm::radians(rotKit2), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(rotPataI), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
tRexPI.Draw(LightingShader);

```

```

/* Persona */
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 33.0f));
model = glm::scale(model, glm::vec3(2.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
personaCu.Draw(LightingShader);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.51f, 2.6f, 33.3f));
model = glm::scale(model, glm::vec3(2.0f));
model = glm::rotate(model, glm::radians(rotBrazo), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(LightingShader.Program, "activaTransparencia"), 0);
personaBra.Draw(LightingShader);

```

Cerrando los buffers ocupados.

```
glBindVertexArray(0);

// Also draw the lamp object, again binding the appropriate shader
lampShader.Use();
// Get location objects for the matrices on the lamp shader (these could be different on a different shader)
modelLoc = glGetUniformLocation(lampShader.Program, "model");
viewLoc = glGetUniformLocation(lampShader.Program, "view");
projLoc = glGetUniformLocation(lampShader.Program, "projection");

/*SKYBOX*/
// Also draw the lamp object, again binding the appropriate shader
lampShader.Use();
// Get location objects for the matrices on the lamp shader (these could be different on a different shader)
modelLoc = glGetUniformLocation(lampShader.Program, "model");
viewLoc = glGetUniformLocation(lampShader.Program, "view");
projLoc = glGetUniformLocation(lampShader.Program, "projection");

// Draw skybox as last
glDepthFunc(GL_EQUAL); // Change depth function so depth test passes when values are equal to depth buffer's content
SkyBoxShader.Use();
view = glm::mat3(glm::mat3(camera.GetViewMatrix())); // Remove any translation component of the view matrix
glm::uniformMatrix4fv(glGetUniformLocation(SkyBoxShader.Program, "view"), 1, GL_FALSE, glm::value_ptr(view));
glm::uniformMatrix4fv(glGetUniformLocation(SkyBoxShader.Program, "projection"), 1, GL_FALSE, glm::value_ptr(projection));

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

// Swap the screen buffers
glfwSwapBuffers(window);

/*SKYBOX*/
glDeleteVertexArrays(1, &VAO);
glDeleteVertexArrays(1, &lightVAO);
glDeleteBuffers(1, &VB0);
glDeleteBuffers(1, &VB1);
glDeleteVertexArrays(1, &skyboxVAO);
glDeleteBuffers(1, &skyboxVB0);
/*SKYBOX*/
// Terminate GLFW, clearing any resources allocated by GLFW.
glfwTerminate();
return 0;
```

Ya fuera del main, encontramos otras funciones que controlan las animaciones e interacción que tiene el usuario con el programa.

Controles de la cámara.

```
// Moves/alters the camera positions based on user input
void DoMovement()
{
    // Camera controls
    if (keys[GLFW_KEY_W] || keys[GLFW_KEY_UP])
    {
        camera.ProcessKeyboard(FORWARD, deltaTime);
    }

    if (keys[GLFW_KEY_S] || keys[GLFW_KEY_DOWN])
    {
        camera.ProcessKeyboard(BACKWARD, deltaTime);
    }

    if (keys[GLFW_KEY_A] || keys[GLFW_KEY_LEFT])
    {
        camera.ProcessKeyboard(LEFT, deltaTime);
    }

    if (keys[GLFW_KEY_D] || keys[GLFW_KEY_RIGHT])
    {
        camera.ProcessKeyboard(RIGHT, deltaTime);
    }
}
```

Controles para las animaciones, luz del cuarto y cerrar la ventana.

```
// Is called whenever a key is pressed/released via GLFW
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode)
{
    if (GLFW_KEY_ESCAPE == key && GLFW_PRESS == action)
    {
        glfwSetWindowShouldClose(window, GL_TRUE);
    }

    if (key >= 0 && key < 1024)
    {
        if (action == GLFW_PRESS)
        {
            keys[key] = true;
        }
        else if (action == GLFW_RELEASE)
        {
            keys[key] = false;
        }
    }

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

    if (keys[GLFW_KEY_L])
    {
        luces = !luces;
        // Luz
        if (luces)
        {
            Light1 = glm::vec3(1.0f, 1.0f, 1.0f);
        }
        else
        {
            Light1 = glm::vec3(0); //Cuado es solo un valor en los 3 vectores pueden dejar solo una componente
        }
    }
}
```

Controles del mouse para mover la vista de la cámara.

```
void MouseCallback(GLFWwindow* window, double xPos, double yPos)
{
    if (firstMouse)
    {
        lastX = xPos;
        lastY = yPos;
        firstMouse = false;
    }

    GLfloat xOffset = xPos - lastX;
    GLfloat yOffset = lastY - yPos; // Reversed since y-coordinates go from bottom to left

    lastX = xPos;
    lastY = yPos;

    camera.ProcessMouseMovement(xOffset, yOffset);
}
```

Función main de las animaciones.

```
void animacion()
{
    if (active)
    {
        animacionPtero();
        animacionBronto();
        animacionTrex();
        animacionPer();
        animKeyFrame();
    }
}
```


Función que controla las animación del Pterodáctilo.

```
void animacionPtero()
{
    //Movimiento del Ptero
    if (recorrido1)
    {
        movMitZ += 0.5f;
        if (movMitZ > 70)
        {
            recorrido1 = false;
            recorrido2 = true;
        }
    }
    if (recorrido2)
    {
        rotMit = 90;
        movMitX += 0.5f;
        if (movMitX > 70)
        {
            recorrido2 = false;
            recorrido3 = true;
        }
    }
    if (recorrido3)
    {
        rotMit = 180;
        movMitZ -= 0.5f;
        if (movMitZ < 0)
        {
            recorrido3 = false;
            recorrido4 = true;
        }
    }
    if (recorrido4)
    {
        rotMit = 270;
        movMitX -= 0.5f;
        if (movMitX < 0)
        {
            recorrido4 = false;
            recorrido5 = true;
        }
    }
    if (recorrido5)
    {
        rotMit = 0;
        movMitZ += 0.5f;
        if (movMitZ > 0)
        {
            recorrido5 = false;
            recorrido1 = true;
        }
    }
    // Alas del Ptero
    if (rotAlaI < 20 && !alas)
    {
        rotAlaI += 0.5f;
        rotAlaD -= 0.5f;
    }
    else
    {
        alas = true;
        if (rotAlaI > -20 && alas)
        {
            rotAlaI -= 0.5f;
            rotAlaD += 0.5f;
        }
    }
    else
    {
        alas = false;
    }
}
```

Función que controla las animación del Brontosaurio.

```
void animacionBronto()
{
    if (rotCuello < 5 && !cuello)
    {
        rotCuello += 0.1f;
        movBronto += 0.006f;
    }
    else
    {
        cuello = true;
        if (rotCuello > 0 && cuello)
        {
            rotCuello -= 0.1f;
            movBronto -= 0.006f;
        }
    }
    else
    {
        cuello = false;
    }
}
```

Función que controla las animación del T-Rex.

```
void animacionTrex()
{
    // Recorrido del T-Rex
    if (camino1)
    {
        movKitX2 += 0.1f;
        if (movKitX2 > 10)
        {
            camino1 = false;
            camino2 = true;
            correcionD = glm::vec3(-0.54f, 5.2f, -1.22f);
            correcionI = glm::vec3(1.22f, 5.2f, 0.54f);
        }
    }
    if (camino2)
    {
        rotKit2 = -45;
        movKitZ2 += 0.2f;
        movKitX2 -= 0.1f;
        if (movKitZ2 > 25)
        {
            camino2 = false;
            camino3 = true;
            correcionD = glm::vec3(1.0f, 5.2f, 0.5f);
            correcionI = glm::vec3(-1.0f, 5.2f, 0.5f);
        }
    }
    if (camino3)
    {
        rotKit2 = 180;
        movKitZ2 -= 0.1f;
        if (movKitZ2 < 0)
        {
            camino3 = false;
            camino1 = true;
            rotKit2 = 90;
            correcionD = glm::vec3(-0.5f, 5.2f, 1.1f);
            correcionI = glm::vec3(-0.5f, 5.2f, -1.1f);
        }
    }
    // Patas del T-Rex
    if (rotPataI < 20 && !patas)
    {
        rotPataI += 0.5f;
        rotPataD -= 0.5f;
    }
    else
    {
        patas = true;
        if (rotPataI > -20 && patas)
        {
            rotPataI -= 0.5f;
            rotPataD += 0.5f;
        }
    }
    else
    {
        patas = false;
    }
}
```

Función que controla las animación del vendedor.

```
void animacionPer()
{
    // Brazo de la persona persona
    if (rotBrazo < 30 && !brazo)
        rotBrazo += 0.5f;
    else
        brazo = true;
    if (rotBrazo > 0 && brazo)
        rotBrazo -= 0.5f;
    else
        brazo = false;
}
```

Función que controla las animación del Velociraptor mediante KeyFrames.

```
void animKeyFrame()
{
    if (play == false && (FrameIndex > 1))
    {
        resetElements();
        //First Interpolation
        interpolation();

        play = true;
        playIndex = 0;
        i_curr_steps = 0;
    }
    if (play)
    {
        if (i_curr_steps >= i_max_steps) //end of animation between frames?
        {
            playIndex++;
            if (playIndex > FrameIndex - 2) //end of total animation?
            {
                /*printf("Animaciones KeyFrames\n");*/
                playIndex = 0;
                play = false;
            }
            else //Next frame interpolations
            {
                i_curr_steps = 0; //Reset counter
                //Interpolation
                interpolation();
            }
        }
        else
        {
            //Draw animation
            posX += KeyFrame[playIndex].incX;
            posY += KeyFrame[playIndex].incY;
            posZ += KeyFrame[playIndex].incZ;

            rothI += KeyFrame[playIndex].rotIncHI;

            posXBI += KeyFrame[playIndex].incXBI;
            posYBI += KeyFrame[playIndex].incYBI;
            posZBI += KeyFrame[playIndex].incZBI;

            rotBI += KeyFrame[playIndex].rotIncBI;
            rotBIX += KeyFrame[playIndex].rotIncBIX;

            posXHD += KeyFrame[playIndex].incXHD;
            posYHD += KeyFrame[playIndex].incYHD;
            posZHD += KeyFrame[playIndex].incZHD;

            rothD += KeyFrame[playIndex].rotIncHD;

            posXBD += KeyFrame[playIndex].incXBD;
            posYBD += KeyFrame[playIndex].incYBD;
            posZBD += KeyFrame[playIndex].incZBD;

            rotBD += KeyFrame[playIndex].rotIncBD;
            rotBDX += KeyFrame[playIndex].rotIncBDX;

            i_curr_steps++;
        }
    }
}
```

Costos

Licencia MAYA	\$8,037 (al mes)
Internet	\$1,500.00
Energía Eléctrica	\$800.00
Gastos no previstos	\$1,500
Salario	\$8,037

Salario consultado de:

<https://mx.indeed.com/career/dise%C3%B1ador-gr%C3%A1fico-junior/salaries>

El total del proyecto considerando que se desarrollará en un periodo de 2 meses es de: \$38,248.

Conclusión

At the end of the project, apart from learning the concepts that we saw during the semester through repetition, the methodology in which it is thought to carry it out is the most similar to developing a real life project in the faculty. The simple fact of having to meet requirements, having a delivery date, advancing the project little by little, structuring it, talking with the professor (client) about the doubts that arose, having a simple planning to carry it out, documenting it and above all the problems that arise along the way and that have to be solved as soon as possible to avoid delays. This is what the development of the project left me with.