Manual Técnico

Proyecto Final

Hernández Hernández Alonso de Jesús

Computación Gráfica e Interacción Humano Computadora

Facultad de Ingeniería

U.N.A.M

26 de Mayo del 2022

Grupo: 05

Índice

1. Versión Español

1.1. Objetivo

El desarrollo de este proyecto se realizó para poder envolver al usuario en un recorrido gráfico sobre la fachada y cuarto virtual utilizando OpenGL en el cual se podrá navegar por el cuarto y observar los objetos de cerca, animaciones realizadas, de igual forma poder implementar lo aprendido en el curso, desarrollando nuestra imaginación para el proyecto, ya que es un entorno 3D en el cual la caricatura llamada "Los Picapiedra"se desenvuelve en un entorno 2D, de esta forma implementar el cuarto y su fachada, de igual forma ver el entorno y que el usuario pueda sentirse dentro del universo de los picapiedra. La creación de este proyecto fue gracias a los conocimientos obtenidos en el transcurso del curso, siendo así la implementación basada en los mismos, obteniendo como resultado este proyecto desarrollado con el esfuerzo, desempeño y compromiso basado para cumplir el objetivo base de este proyecto final, la implementación de todos los conocimientos.

1.2. Alcance

Lograr la implementación de una fachada cercana al realismo de la fachada con la imagen de referencia junto su textura cercana al mismo, de igual forma la interacción por medio de una cámara libre que nos permite movernos por el entorno, realizar interacciones con animaciones establecidas para lograr la interacción del mismo. Obteniendo un resultado favorable, en el cual se ve el entorno esencial de el proyecto, resultados de el objetivo principal de acuerdo a los tiempos establecidos y asignados, siendo así alcanzados en la entrega final, de acuerdo al diagrama de gantt establecido en este documento, en el cual pudimos avanzar con precisión y rapidez, siendo así un desarrollo satisfactorio, debido a todo lo que se logra implementar en esta sección, ya que es toda lo visto en curso.

1.3. Limitantes

Se lograron alcanzar los objetivos de la materia, a su vez por el tiempo se trabajo de forma a corto alcance,se vieron muchos temas, conocimientos, funciones, animaciones,etc, pero no a profundización máxima por cuestión del programa de estudio, a su vez el profesor nos brindó toda la información posible para poder emplear este proyecto, algunos conocimientos que no se encontraban, fueron combinados por el lenguaje de programación C++ y la implementación de OpenGL, así logrando obtener ciertas acciones o ambientación, ya que si buscabas directamente la duda, no se encontraba de manera sencilla y fácil. Otro limitante encontrado fueron los recursos de mi computador, que se tuvo que adaptar y disminuir a su vez los vértices creados para su ejecución, empleando el voxel art para que se representara ciertos objetos.

1.4. Diagrama de Gantt

Es el desarrollo de un diagrama en el cual se estableció fechas de entrega, tiempo y avance de acuerdo a los tiempos asignados por el desarrollo y su complejidad del mismo, a su vez de la urgencia que se tenía para evitar retrasos de entrega en el momento, siendo así el cumplimiento en fechas de entrega.

Nombre de la tarea	Fecha de inicio	Fecha de finalización	Asignado	Estado
Proyecto Picapiedra	07.03.2022	11.05.2022	Alonso	Terminado
Propuesta de Fachada	09.03.2022	14.03.2022	Alonso	Terminado
Primer Objeto	16.03.2022	22.03.2022	Alonso	Terminado
Dibujado del Primer Objeto	16.03.2022	18.03.2022	Alonso	Terminado
Texturizado del Primer Objeto	19.03.2022	20.03.2022	Alonso	Terminado
Programado del Primer Objeto en OpenGL	20.03.2022	21.03.2022	Alonso	Terminado
Segundo Objeto	23.03.2022	29.03.2022	Alonso	Terminado
Dibujado del Segundo Objeto	24.03.2022	25.03.2022	Alonso	Terminado
Texturizado del Segundo Objeto	26.03.2022	27.03.2022	Alonso	Terminado
Programado del Segundo Objeto en OpenGL	28.03.2022	29.03.2022	Alonso	Terminado
Tercer Objeto	30.03.2022	05.04.2022	Alonso	Terminado
Dibujado del Tercer Objeto	31.03.2022	01.04.2022	Alonso	Terminado
Texturizado del Tercer Objeto	02.04.2022	03.04.2022	Alonso	Terminado
Programado del Tercer Objeto en OpenGL	04.04.2022	05.04.2022	Alonso	Terminado
Cuarto Objeto	06.04.2022	12.04.2022	Alonso	Terminado
Dibujado del Cuarto Objeto	07.04.2022	08.04.2022	Alonso	Terminado
Texturizado del Cuarto Objeto	09.04.2022	10.04.2022	Alonso	Terminado
Programado del Cuarto Objeto en OpenGL	11.04.2022	11.04.2022	Alonso	Terminado
Quinto Objeto	13.04.2022	20.04.2022	Alonso	Terminado
Dibujado del Quinto Objeto	14.04.2022	15.04.2022	Alonso	Terminado
Texturizado del Quinto Objeto	16.04.2022	17.04.2022	Alonso	Terminado
Programado del Quinto Objeto en OpenGL	18.04.2022	18.04.2022	Alonso	Terminado
Sexto Objeto	20.04.2022	27.04.2022	Alonso	Terminado
Dibujado del Sexto Objeto	21.04.2022	22.04.2022	Alonso	Terminado
Texturizado del Sexto Objeto	23.04.2022	24.04.2022	Alonso	Terminado
Programado del Sexto Objeto en OpenGL	25.04.2022	25.04.2022	Alonso	Terminado
Septimo Objeto	27.04.2022	03.05.2022	Alonso	Terminado
Dibujado del Septimo Objeto	28.04.2022	28.04.2022	Alonso	Terminado
Texturizado del Septimo Objeto	29.04.2022	29.04.2022	Alonso	Terminado
Programado del Septimo Objeto en OpenGL	02.05.2022	02.05.2022	Alonso	Terminado
Dibujado del Fachada Objeto	11.04.2022	12.04.2022	Alonso	Terminado
Texturizado del Fachada Objeto	12.04.2022	13.04.2022	Alonso	Terminado
Programado del Fachada Objeto en OpenGL	14.04.2022	14.04.2022	Alonso	Terminado
Programado de Primer Animación en OpenGL	14.04.2022	15.04.2022	Alonso	Terminado
Programado de Segunda Animación en OpenGL	20.04.2022	21.04.2022	Alonso	Terminado
Programado de Tercer Animación en OpenGL	02.05.2022	02.05.2022	Alonso	Terminado
Programado de Cuarto Animación en OpenGL	04.05.2022	05.05.2022	Alonso	Terminado
Programado de Quinto Animación en OpenGL	05.05.2022	07.05.2022	Alonso	Terminado
Desarrollo de Propuesta de Animación Mejorado	17.05.2022	17.05.2022	Alonso	Terminado
Desarrollo de Bitacora	18.05.2022	19.05.2022	Alonso	Terminado
Boceto	19.05.2022	19.05.2022	Alonso	Terminado
Mejora de Proyecto Final	20.05.2022	24.05.2022	Alonso	Terminado
Desarrollo de Documentación	24.05.2022	25.05.2022	Alonso	Terminado

Figura 1: Diagram de Gantt Parte 1

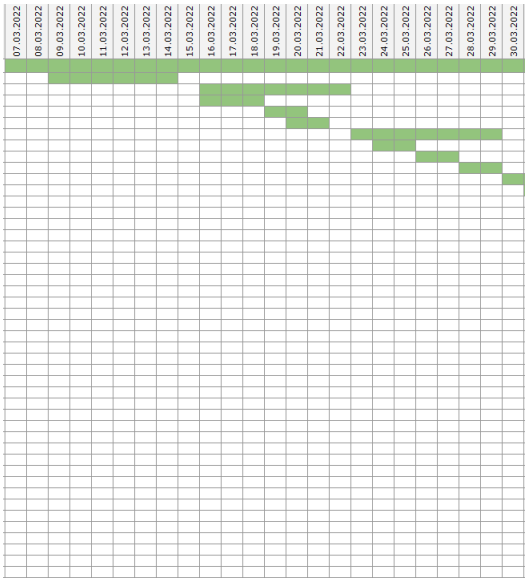


Figura 2: Diagram de Gantt Parte 2

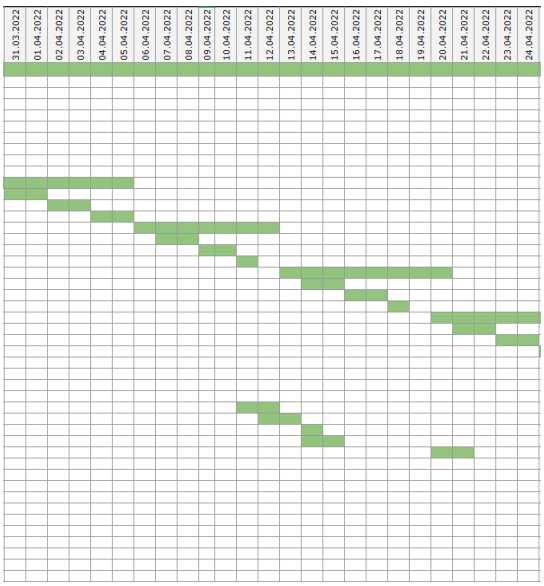


Figura 3: Diagrama de Gantt Parte 3

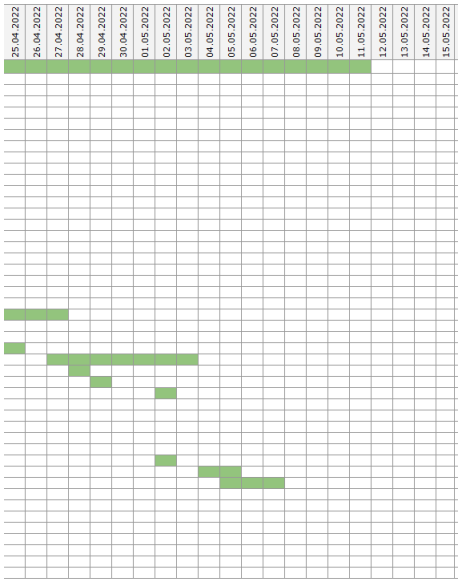


Figura 4: Diagrama de Gantt Parte 4

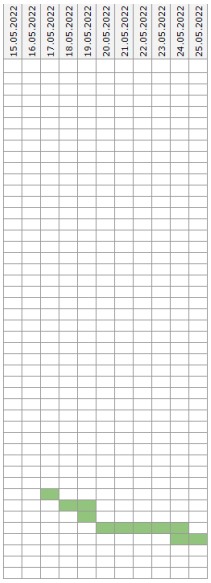


Figura 5: Diagrama de Gantt Parte 5

1.5. Documentación

- En la siguiente imagen se observa la declaración de librerías a usar, librerías de procesamiento en la primera sección, librerías para incluir código OpenGL, usadas para realizar la transformaciones a los modelos como lo es glm, a su vez encontramos la librería que nos va a permitir crear y cargar los modelos texturizados, como lo es SOIL2, y por último las librerías que son creadas independiente o por nosotros.

```
/*Sección de declaración de bibliotecas y dependecias a usar para su procesamiento*/
#include <iostream>
#include <cmath>
#include <Windows.h>
#include <mmsystem.h>

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

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

// Libreria de std image.h
#include "stb_image.h"

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

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


// Libreria creadas
#include "Shader.h"
#include "Camera.h"
#include "Model.h"
#include "Texture.h"
```

Figura 6: Declaración de Librerías

- La declaración de las funciones que vamos a utilizar, como es utilizar el teclado, el mouse y funciones especiales para la creación de animaciones y banderas.

```
/*Declaración de las funciones a utilizar*/
// Function prototypes
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode);
void MouseCallback(GLFWwindow* window, double xPos, double yPos);
void DoMovement();
void animacion();
void animacion2();
void animacion3();
void Sonido();
void Sonido2();
void Sonido3();
```

Figura 7: Declaración de Funciones

- Creación de la variables a usar para la inicialización del tamaño de ventana en que resolución se ejecutara la ventana.Declaración para que la pantalla quede a la mitad de la resolución, en el punto medio de la ventana.Se declara el arreglo para asignación total de las teclas a utilizar, en este, caso se asigna un total de memoria de 1024.Bandera de activación para el movimiento de ratón, enviando la bandera como positivo.Inicialización de la posición inicial, objeto 1 (Dinosaurio Cuello Largo), objeto 2 (Pedro Picapiedra) los cuales se moverán con Key Frame, iniciando su posición en (0,0,0). La primera variable funciona como bandera, en el cual activará la animación 1, la segunda será un contador para la animación 1, la tercer variable funciona la captura de tiempo o pulsos de procesador, y una bandera utilizada para prender y apagar la TV.

```
// Dimensiones de la ventana al crear
const GLuint WIDTH = 1920, HEIGHT = 1080;
int SCREEN_WIDTH, SCREEN_HEIGHT;

// Declaración de modelo de camara y su posición
Camera  camera(glm::vec3(15.0f, 1.0f, 0.0f));
GLfloat lastX = WIDTH / 2.0;
GLfloat lastY = HEIGHT / 2.0;

//Arreglo para teclas
bool keys[1024];

//Variable de uso de movimiento de ratón (periferico)
bool firstMouse = true;


// Atributos del uso de ilumincación, posición inicial
glm::vec3 lightPos(0.0f, 0.0f, 0.0f);

//Declaración de posición inical de objeto a mover con Key Frames
glm::vec3 PosIni(0.0f, 1.2f, 8.0f);
glm::vec3 PosIni2(-1.0f, 0.5f, 1.5f);
glm::vec3 PosIni3(8.0f, 0.9f, 0.0f);


/*Declaración de variables, para activar animaciones */
bool active;
float anim1=0.0f;
float tiempo;
int bandera;
```

Figura 8: Declaración Variables

- Arreglo de puntos iniciales para la posición de los point lights en el cual será la posición en donde se dibujarán desde un inicio.Variables a utilizar para animación 3 donde se utiliza la primera para el movimiento en X, la segunda para el movimiento en Z y la tercera para la rotación utilizada para movimiento en positivo y negativo.

```
unsigned int indices[48600];
GLuint VBO2, VAO2, EBO;


// Posición inicial del pointlight y cubo
glm::vec3 pointLightPositions[] = {
    glm::vec3(0.0f,3.0f, 0.0f),
    glm::vec3(10.0, 3.0, 15.0),
    glm::vec3(25.0, 3.0, 15.0),
    glm::vec3(40.0, 3.0, 15.0),
    glm::vec3(1.5f, 0.82f, 1.15f),
};

//Variables de movimiento y rotación para animación 3
float movKitX = 0.0;
float movKitZ = 8.0;
float movKitY = 12.0;
float rotKit = -90.0;


float movKitX2 = 0.0;
float movKitZ2 = 0.0;
float movKitY2 = 0.0;
float rotKit2 = -90.0;
float rot2 = 0.0f;
float rot3 = 0.0f;
float rot4 = 0.0f;
float rot5 = 0.0f;
```

Figura 9: Variables Animación

- Inicialización de la bandera que controlará la animación 3 del dragón. Variables bandera, para realizar el cambio de posición siendo así el recorrido cambia cuando se complete el proceso.

```
bool anim3 = false;
bool anim4 = false;
bool anim5 = false;
bool anim6 = false;
bool recorrido10 = true;
bool recorrido11 = false;


//Bandera a usar la activación de recorrido de movimiento
bool circuito = false;
bool circuito2 = false;

/*Banderas para activar movimiento y recorrido por secciones*/
bool recorrido1 = true;
bool recorrido2 = false;
bool recorrido3 = false;
bool recorrido4 = false;
bool recorrido5 = false;
bool recorrido6 = false;
bool recorrido7 = false;
bool recorrido8 = false;
bool recorrido9 = false;

bool anim = false;
bool anim2 = false;
float rot = 0.0f;
```

Figura 10: Banderas, recorrido y animaciones

- Inicialización de variables para el calculo de los frames usados, deltatime, será la diferencia entre frame actual y ultimo, lastframe almacenará el fame actual. Variables inicializadas a valores seguros, estas variables serán usadas para el incremento de posición que se almacenarán en el arreglo de frames, la última variable será usada para el movimiento del modelo. Definición como constante la variable max_frames, la cual será el limite de frames a almacenar. Variables para medición, el número total de posiciones, para recorrer el arreglo de posición, almacenando cada uno en una localidad independiente, así teniendo como máximo 190 y el actual, será para medir el recorrido actual.

```
// Declaración de medida de tiempo entre frames usado y el ultimo frame
GLfloat deltaTime = 0.0f;   // Diferencia de tiempo entre frames
GLfloat lastFrame = 0.0f;   // Tiempo del Ultimo Frame

// Declaración de los key frames, primer key frame
float posX = PosIni.x, posY = PosIni.y, posZ = PosIni.z, rotCuerpo = 0, rotPies=0;

// Declaración de los key frames, segundo key frame
float posX2 = PosIni.x, posY2 = PosIni.y, posZ2 = PosIni2.z, rotBrazoDer = 0;

//Definición de memoria de los key frames, valor 4
#define MAX_FRAMES 4

//Variables a usar de numero maximo de pasos y paso actual, uso para key frame
int i_max_steps = 190;
int i_curr_steps = 0;

//Variables a usar de numero maximo de pasos y paso actual, uso para key frame 2
int i_max_steps2 = 190;
int i_curr_steps2 = 0;
```

Figura 11: Declaración posición inicial y frames

- Definición de estructura frame 1 y frame 2, con el cual se contará con la siguientes propiedades, posición en X,Y,Z en variables diferentes, al igual el incremento en cada una de ellas, para realizar el aumento lineal, de igual forma el incremento del objeto a rotar por key frames, para su incremento de igual forma se crea una variable.

```
//Definición Estructural del Frame, para acceso a sus atributos
typedef struct _frame
{
    //Variables para GUARDAR Key Frames
    float posX;     //Variable para PosicionX
    float posY;     //Variable para PosicionY
    float posZ;     //Variable para PosicionZ
    float incX;     //Variable para IncrementoX
    float incY;     //Variable para IncrementoY
    float incZ;     //Variable para IncrementoZ
    float rotCuerpo; //Variable para Rotación de Cuerpo
    float rotInc; //Variable para guardar rotación
    float rotInc2;
    float rotPies;

}FRAME;

//Definición Estructural del Frame 2, para acceso a sus atributos
typedef struct _frame2
{
    //Variables para GUARDAR Key Frames
    float posX2;        //Variable para PosicionX
    float posY2;        //Variable para PosicionY
    float posZ2;        //Variable para PosicionZ
    float incX2;        //Variable para IncrementoX
    float incY2;        //Variable para IncrementoY
    float incZ2;        //Variable para IncrementoZ
    float rotBrazoDer; //Variable para guardar rotación de brazos
    float rotInc2;  //Variable para guardar incremento de brazos

}FRAME2;
```

Figura 12: Estructura Frame 1 y 2

- Declaración de variables para trabajar por medio de frames, declarando un arreglo llamado KeyFrame con memoria de 4, la siguiente variable usará el índice frame para el recorrido de cada frame, una bandera para la activación y recorrido y el índice de para el recorrer dentro de la estructura.

```
/*
* Declaración de arreglo Key Frame con tamaño 4
* Variable para introducir los datos de posición e interpolación al frame
* Bandera de activación de animación
* Indice de arreglo para recorrido
*/
FRAME KeyFrame[MAX_FRAMES];
int FrameIndex = 0;         //introducir datos
bool play = false;
int playIndex = 0;

/*
* Declaración de arreglo Key Frame 2 con tamaño 4
* Variable para introducir los datos de posición e interpolación al frame
* Bandera de activación de animación
* Indice de arreglo para recorrido
*/
FRAME2 KeyFrame2[MAX_FRAMES];
int FrameIndex2 = 0;            //introducir datos
bool play2 = false;
int playIndex2 = 0;
```

Figura 13: Variables para Key Frame 1 y 2

- Creación de arreglo, con puntos de los vertices para poder dibujar el cubo de iluminación, mostrado en para simular un foco y a su vez simular la TV encendida.

```
//Areglo de los vertices para dibujo de cubo de iluminación
float vertices[] = {
    -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
       0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
       0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
       0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
      -0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
      -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,

      -0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
       0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
       0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
       0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
      -0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
      -0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f,

      -0.5f,  0.5f,  0.5f, -1.0f,  0.0f,  0.0f,
      -0.5f,  0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
      -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
      -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
      -0.5f, -0.5f,  0.5f, -1.0f,  0.0f,  0.0f,
      -0.5f,  0.5f,  0.5f, -1.0f,  0.0f,  0.0f,

       0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f,
       0.5f,  0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
       0.5f, -0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
       0.5f, -0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
       0.5f, -0.5f,  0.5f,  1.0f,  0.0f,  0.0f,
       0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f,

      -0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,
       0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,
       0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
       0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
      -0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
      -0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,

      -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,
       0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,
       0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
       0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
      -0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
      -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f
};
```

Figura 14: Arreglo Vertices

- Inicialización de variable de iluminación a vector 3, para asignar transformaciones, posición,etc.Función para reiniciar los elementos al reproducir la animación, la posición inicial se guarda en la primera localidad y esta misma se envía a la posición actual, para reiniciar posición del objeto.

```
//Declaración de model de luz
glm::vec3 Light1 = glm::vec3(0);


//Función para reiniciar posición inicial de objeto de frame 1
void resetElements(void)
{
    //Variable regresando la posición inicial almacenada en el frame arreglo de posición 0
    posX = KeyFrame[0].posX;
    posY = KeyFrame[0].posY;
    posZ = KeyFrame[0].posZ;

    rotCuerpo = KeyFrame[0].rotCuerpo;
    rotPies = KeyFrame[0].rotPies;
}

//Función para reiniciar posición inicial de objeto
void resetElements2(void)
{
    //Variable regresando la posición inicial almacenada en el frame arreglo de posición 0
    posX2 = KeyFrame2[0].posX2;
    posY2 = KeyFrame2[0].posY2;
    posZ2 = KeyFrame2[0].posZ2;

    rotBrazoDer = KeyFrame2[0].rotBrazoDer;

}
```

Figura 15: Función Reset Elements

- Función interpolación, realiza un calculo para realizar el incremento, dicho incremento definirá la velocidad de reproducción de nuestra animación por key frame 1 y 2, donde se implementa la formula:

$$Incremento = \frac{posx.siguiente - posx.actual}{numeromaximodepasos}$$

```
//Función de calculo de interpolación para realizar la animación y su velocidad de incremento, siendo lineal.
void interpolation(void)
{

    KeyFrame[playIndex].incX = (KeyFrame[playIndex + 1].posX - KeyFrame[playIndex].posX) / i_max_steps; //Incremento en X
    KeyFrame[playIndex].incY = (KeyFrame[playIndex + 1].posY - KeyFrame[playIndex].posY) / i_max_steps; //Incremento en Y
    KeyFrame[playIndex].incZ = (KeyFrame[playIndex + 1].posZ - KeyFrame[playIndex].posZ) / i_max_steps; //Incremento en Z

    KeyFrame[playIndex].rotInc = (KeyFrame[playIndex + 1].rotCuerpo - KeyFrame[playIndex].rotCuerpo) / i_max_steps; //Incremeneto en rotación de objeto
    KeyFrame[playIndex].rotPies= (KeyFrame[playIndex + 1].rotPies - KeyFrame[playIndex].rotPies) / i_max_steps; //Incremeneto en rotación de objeto
}

//Función de calculo de interpolación para realizar la animación y su velocidad de incremento, siendo lineal.
void interpolation2(void)
{

    KeyFrame2[playIndex2].incX2 = (KeyFrame2[playIndex2 + 1].posX2 - KeyFrame2[playIndex2].posX2) / i_max_steps2; //Incremento en X
    KeyFrame2[playIndex2].incY2 = (KeyFrame2[playIndex2 + 1].posY2 - KeyFrame2[playIndex2].posY2) / i_max_steps2; //Incremento en Y
    KeyFrame2[playIndex2].incZ2 = (KeyFrame2[playIndex2 + 1].posZ2 - KeyFrame2[playIndex2].posZ2) / i_max_steps2; //Incremento en Z

    KeyFrame2[playIndex2].rotInc2 = (KeyFrame2[playIndex2 + 1].rotBrazoDer - KeyFrame2[playIndex2].rotBrazoDer) / i_max_steps2; // Incremeneto en rotación de objeto
}
```

Figura 16: Función Interpolación 1y 2

- Se inicializa los shader a usar en este caso el primero se utiliza para el manejo de iluminación, el segundo para iluminación en puntos y el último para construcción de skybox.

```
//Se llama la función toroide
toroide();


/*
* Inicializamos los shaders a usar
* Shader para iluminación
* Shader para iluminación
* Shader de Skybox
*/
Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting.frag");
Shader lampShader("Shaders/lamp.vs", "Shaders/lamp.frag");
Shader SkyBoxshader("Shaders/SkyBox.vs", "Shaders/SkyBox.frag");
```

Figura 17: Función toroide y shaders

- Se construye el modelo basado en el .obj para la construcción de Pedro Picapiedra, así establecemos todos los objetos en una variables especifica para su construcción en un modelo OpenGL. Se construye el modelo basado en el .obj para la construcción del Dragón, así establecemos todos los objetos en una variables especifica para su construcción en un modelo OpenGL. Se manda a llamar el shader de animación, establecido como libreria externa inicalizado en un variable local Anim.Se construye el modelo basado en el .obj para la construcción de Objetos dentro de la fachada: Sillón,Mesa,Libro,TV,Tapete,Cojín y Planta, así establecemos todos los objetos en variables especifica para su construcción en un modelo OpenGL.

```
//Pedro Picapiedra
//Modelo de la Cabeza
Model CabezaP((char*)"Models/Obj/Models/Pedro/Cabeza/Cabeza.obj");
//Modelo de el Cuerpo
Model CuerpoP((char*)"Models/Obj/Models/Pedro/Cuerpo/CuerpoP.obj");
//Modelo del Brazo Izquierdo
Model BrazoIzq((char*)"Models/Obj/Models/Pedro/BrazoIzq/BrazoIzq.obj");
//Modelo del Brazo Derecho
Model BrazoDer((char*)"Models/Obj/Models/Pedro/BrazoDer/BrazoDer.obj");
//Modelo de Pies
Model PiesP((char*)"Models/Obj/Models/Pedro/Pies/Pies.obj");

Model Carro((char*)"Models/Obj/Models/Carro/Carro.obj");
//Modelo de Dinosaurio
Model AlaD((char*)"Models/Obj/Models/Dragon/AlaDerecha/AlaDerecha.obj");
Model AlaI((char*)"Models/Obj/Models/Dragon/AlaIzquierda/AlaIzquierda.obj");
Model Dino((char*)"Models/Obj/Models/Dragon/Cuerpo/Cuerpo.obj");
```

```
//Shader creado para animacion del agua
Shader Anim("Shaders/anim.vs", "Shaders/anim.frag");
//Modelo de el agua
Model Agua((char*)"Models/Obj/Models/Agua/Agua.obj");
//Modelo del dinosaurio cuerpo
Model DinosaurioCuerpo((char*)"Models/Obj/Models/DinoPerson/Cuerpo/Cuerpo.obj");
//Modelo del dinosaurio pies traseros
Model DinosaurioPiesA((char*)"Models/Obj/Models/DinoPerson/PiesA/PiesA.obj");
//Modelo del dinosaurio pies delanteros
Model DinosaurioPiesD((char*)"Models/Obj/Models/DinoPerson/PiesD/PiesD.obj");
//Modelo del piso
Model Piso((char*)"Models/Obj/Models/Piso/Piso.obj");
//Modelo de la fachada
Model Fachada((char*)"Models/Obj/Models/Fachada/Fachada.obj");
//Modelo de Planta 1
Model Planta1((char*)"Models/Obj/Models/Planta1/Planta1.obj");
//Modelo de Planta 2
Model Planta2((char*)"Models/Obj/Models/Planta2/Planta2.obj");
//Modelo de Planta 3
Model Planta3((char*)"Models/Obj/Models/Planta3/Planta3.obj");
//Modelo de Planta 4
Model Planta4((char*)"Models/Obj/Models/Planta4/Planta4.obj");
```

Figura 18: Inicialización de Modelos de Objetos

```
//Modelo de Planta 5
Model Planta5((char*)"Models/Obj/Models/Planta5/Planta5.obj");
//Modelo de Planta 6
Model Planta6((char*)"Models/Obj/Models/Planta6/Planta6.obj");
//Modelo de Sillon1
Model Sillon1((char*)"Models/Obj/Models/Sillon1/Sillon.obj");
//Modelo de Sillon2
Model Sillon2((char*)"Models/Obj/Models/Sillon2/Sillon2.obj");
//Modelo de Cojines
Model Cojines((char*)"Models/Obj/Models/Cojines/Cojines.obj");
//Modelo de Mesa
Model Mesa((char*)"Models/Obj/Models/Mesa/Mesa.obj");
//Modelo de Mesa2
Model Mesa2((char*)"Models/Obj/Models/Mesa2/Mesa2.obj");
//Modelo de TV
Model TV((char*)"Models/Obj/Models/TV/TV.obj");
//Modelo de Tapete
Model Tapete((char*)"Models/Obj/Models/Tapete/Tapete.obj");
```

```
//Modelo de Tapete
Model Tapete((char*)"Models/Obj/Models/Tapete/Tapete.obj");
//Modelo de Libros
Model Libros((char*)"Models/Obj/Models/Libros/Libros.obj");
//Modelo de Montañas
Model Montana((char*)"Models/Obj/Models/Montaña/Montaña.obj");
//Modelo de Rocas
Model Roca((char*)"Models/Obj/Models/Roca/Roca.obj");
//Modelo de Pablo
Model Pablo((char*)"Models/Obj/Models/Pablo/Pablo.obj");
//Modelo de Betty
Model Betty((char*)"Models/Obj/Models/Betty/Betty.obj");
//Modelo de Vilma
Model Vilma((char*)"Models/Obj/Models/Vilma/Vilma.obj");
//Modelo de Rajuela
Model Rajuela((char*)"Models/Obj/Models/Rajuela/Rajuela.obj");
//Modelo de Cavernicola
Model Cavernicola((char*)"Models/Obj/Models/Cavernicola/Cavernicola.obj");
//Modelo de Trajeado
Model Trajeado((char*)"Models/Obj/Models/Trajeado/Trajeado.obj");
```

Figura 19: Inicialización de Modelos de Objetos

- Se inicializa a valores de posición para el keyframe 1 y keyframe 2 para su reproducción automática

```
KeyFrame[0].posX= 0;
KeyFrame[0].posY = 1.5;
KeyFrame[0].posZ= 8.0;
KeyFrame[1].posX = 0;
KeyFrame[1].posY = 1.0;
KeyFrame[1].posZ = 8.0;
KeyFrame[2].posX = 0;
KeyFrame[2].posY = 1.0;
KeyFrame[2].posZ = 8.0;
KeyFrame[3].posX = 0;
KeyFrame[3].posY = 1.0;
KeyFrame[3].posZ = 8.0;
KeyFrame[0].rotCuerpo = 0;
KeyFrame[1].rotCuerpo = 45;
KeyFrame[2].rotCuerpo = -1;
KeyFrame[0].rotPies= 0;
KeyFrame[1].rotPies = 45;
KeyFrame[2].rotPies= -1;

FrameIndex = 4;
```

```
KeyFrame2[0].rotBrazoDer = 0;
KeyFrame2[1].rotBrazoDer = -90;
KeyFrame2[2].rotBrazoDer = 0;
FrameIndex2 = 4;
```

Figura 20: Ciclos para Frames

- Creación de un arreglo para asignación de vértices del cubo a dibujar del skybox, así realizando cada vertices para su construcción.

```
//Arreglo de vertices para dibujo de cubo para skybox
GLfloat skyboxVertices[] = {
    // Positions
    -1.0f,  1.0f, -1.0f,
    -1.0f, -1.0f, -1.0f,
    1.0f, -1.0f, -1.0f,
    1.0f, -1.0f, -1.0f,
    1.0f,  1.0f, -1.0f,
    -1.0f,  1.0f, -1.0f,

    -1.0f, -1.0f,  1.0f,
    -1.0f, -1.0f, -1.0f,
    -1.0f,  1.0f, -1.0f,
    -1.0f,  1.0f, -1.0f,
    -1.0f,  1.0f,  1.0f,
    -1.0f, -1.0f,  1.0f,

    1.0f, -1.0f, -1.0f,
    1.0f, -1.0f,  1.0f,
    1.0f,  1.0f,  1.0f,
    1.0f,  1.0f,  1.0f,
    1.0f,  1.0f, -1.0f,
    1.0f, -1.0f, -1.0f,

    -1.0f, -1.0f,  1.0f,
    -1.0f,  1.0f,  1.0f,
    1.0f,  1.0f,  1.0f,
    1.0f,  1.0f,  1.0f,
    1.0f, -1.0f,  1.0f,
    -1.0f, -1.0f,  1.0f,

    -1.0f,  1.0f, -1.0f,
    1.0f,  1.0f, -1.0f,
    1.0f,  1.0f,  1.0f,
    1.0f,  1.0f,  1.0f,
    -1.0f,  1.0f,  1.0f,
    -1.0f,  1.0f, -1.0f,

    -1.0f, -1.0f, -1.0f,
    -1.0f, -1.0f,  1.0f,
    1.0f, -1.0f, -1.0f,
    1.0f, -1.0f, -1.0f,
    -1.0f, -1.0f,  1.0f,
    1.0f, -1.0f,  1.0f
};
```

Figura 21: Vertices cubo skybox

- Se inicializa la carga de texturas de el skybox, para siguiente hacer la carga de imágenes de textura de cara de los cubos del vertex.

```
//Carga de imagenes para texturizar el skybox
vector<const GLchar*> faces;
faces.push_back("SkyBox/right.tga");
faces.push_back("SkyBox/left.tga");
faces.push_back("SkyBox/top.tga");
faces.push_back("SkyBox/bottom.tga");
faces.push_back("SkyBox/front.tga");
faces.push_back("SkyBox/back.tga");
```

Figura 22: Textura Skybox

- Llamada de las funciones de sonido para su reproducción, la intro de sonido para programa y la segunda función de sonido ambiental.

```
//Inicialización de funciones de sonido ambiente e intro
Sonido();
Sonido2();
```

Figura 23: Invocación de función de Sonidos

- Invocación de las funciones para captura de teclado, movimiento constante, y funciones animación, desde esta parte del código se inicializa un ciclo infinito, en el cual cambiará el estado de cada función invocada y su acción de cuerdo a las banderas declaradas.

```
// Revisamos la inicialización de eventos para animaciones constantes y revisamos banderas
glfwPollEvents();
DoMovement();
animacion();
animacion2();
animacion3();
animacion4();
animacion5();
animacion6();
```

Figura 24: Inicialización de Funciones de Animación

- Se inicializa en la primera parte el shader de iluminación establecemos su propiedades de iluminación (especular,difusa,direccional,ambiental, a continuación establecemos a los ejes X,Y,Z que de acuerdo al ingreso de los valores los lea como color, en las siguientes lineas se establece posición del pointlight y spotlight, una usada para iluminación dentro de fachada la siguiente como animación, de igual manera se inicializan sus propiedades.

```
// Establecemos la iluminación direccional, dedsde dirección, escular, ambiental y difusa.
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.1f, 0.1f, 0.1f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.1f, 0.1f, 0.1f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.4f, 0.4f, 0.4f);
```

```
//Inicialización de pposición, e iluminación desde el arreglo de posiciones
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"), pointLightPositions[0].x, pointLightPositions[0].y, pointLightPositions[0].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"), 1.0f, 1.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), 1.0f, 1.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"), 1.0f, 1.0f, 0.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].linear"), 0.7f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"), 1.8f);

glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].position"), pointLightPositions[1].x, pointLightPositions[1].y, pointLightPositions[1].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].ambient"), 1.0f, 0.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].diffuse"), 1.0f, 1.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].specular"), 1.0f, 1.0f, 0.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].linear"), 0.7f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].quadratic"), 1.8f);
```

Figura 25: Inicialización de Iluminación

```
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].position"), pointLightPositions[2].x, pointLightPositions[2].y, pointLightPositions[2].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].ambient"), 0.0f, 1.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].diffuse"), 1.0f, 1.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].specular"), 1.0f, 1.0f, 0.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].linear"), 0.7f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].quadratic"), 1.8f);

glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].position"), pointLightPositions[3].x, pointLightPositions[3].y, pointLightPositions[3].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].ambient"), 0.0f, 0.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].diffuse"), 1.0f, 1.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].specular"), 1.0f, 1.0f, 0.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].linear"), 0.7f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].quadratic"), 1.8f);
```

```
// Iniciaización de posición de spotlight, usado como animación de encedido y apagado de TV
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[0].position"), 1.5f, 0.82f, 1.18f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[0].direction"), -5.0f, 0.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[0].ambient"), anim1, anim1, anim1);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[0].diffuse"), anim1, anim1, anim1);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[0].specular"), anim1, anim1, anim1);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[0].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[0].linear"), 0.35f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[0].quadratic"), 0.44f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[0].cutOff"), glm::cos(glm::radians(12.5f)));
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[0].outerCutOff"), glm::cos(glm::radians(15.0f)));

glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[1].position"), 10.0f, 3.0f, -15.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[1].direction"), 0.0f, -1.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[1].ambient"), 0.0f, 0.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[1].diffuse"), 1.0f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[1].specular"), 1.0f, 1.0f, 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[1].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[1].linear"), 0.35f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[1].quadratic"), 0.44f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[1].cutOff"), glm::cos(glm::radians(12.5f)));
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[1].outerCutOff"), glm::cos(glm::radians(15.0f)));
```

Figura 26: Inicialización de Iluminación

```
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[2].position"), 25.0f, 3.0f, -15.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[2].direction"), 0.0f, -1.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[2].ambient"), 1.0f, 0.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[2].diffuse"), 1.0f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[2].specular"), 1.0f, 1.0f, 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[2].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[2].linear"), 0.35f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[2].quadratic"), 0.44f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[2].cutOff"), glm::cos(glm::radians(12.5f)));
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[2].outerCutOff"), glm::cos(glm::radians(15.0f)));

glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[3].position"), 40.0f, 3.0f, -15.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[3].direction"), 0.0f, -1.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[3].ambient"), 0.0f, 1.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[3].diffuse"), 1.0f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[3].specular"), 1.0f, 1.0f, 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[3].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[3].linear"), 0.35f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[3].quadratic"), 0.44f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[3].cutOff"), glm::cos(glm::radians(12.5f)));
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[3].outerCutOff"), glm::cos(glm::radians(15.0f)));

// Iniciamos la propiedad de brillo sobre material
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
```

Figura 27: Inicialización de Iluminación

- A continuación se cargan los modelos establecido anteriormente para el dibujado y asignación a matriz del modelo anteriormente iniciada, así mismo en cada final de linea de parte de cuerpo de dinosaurio, se envía a dibujar con un parámetro a recibir el shader de iluminación.

```
//Carga de modelo de personaje de dinosaurio
view = camera.GetViewMatrix();//establecemos la vista
glm::mat4 model(1);//Declaración de model
model = glm::translate(model, glm::vec3(0.0f, 1.2f, 8.0f));//Transformación de traslación a posición inicial
model = glm::rotate(model, glm::radians(-rotCuerpo), glm::vec3(0.0f, 0.0f, 1.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Enviamos el modelo a la matriz
DinosaurioCuerpo.Draw(lightingShader);//Dibujamos el modelo con parametro recibido de shader de iluminación


view = camera.GetViewMatrix();//Establecemos vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.0f, 1.2f, 8.0f));//Transformación de traslación a posición inicial
model = glm::rotate(model, glm::radians(-rotPies), glm::vec3(0.0f, 0.0f, 1.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos matriz de modelo
DinosaurioPiesA.Draw(lightingShader);//Dibujamos el modelo  con parametro recibido de shader de iluminación

view = camera.GetViewMatrix();//establecemos la vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.0f, 1.2f, 8.0f));//Transformación de traslación a posición inicial
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Enviamos el modelo a la matriz
DinosaurioPiesD.Draw(lightingShader);//Dibujamos el modelo con parametro recibido de shader de iluminación

glBindVertexArray(0);//Finalizamos vertex array
```

```
view = camera.GetViewMatrix();//establecemos la vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.0f, 1.0f, -7.0f));//Transformación de traslación a posición inicial
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Enviamos el modelo a la matriz
Carro.Draw(lightingShader);//Dibujamos el modelo con parametro recibido de shader de iluminación
glBindVertexArray(0);//Finalizamos vertex array
```

Figura 28: Dibujado de Modelos

- Modelos

```
glBindVertexArray(VAO2);//Cargamos los valores a una matriz diferente

glm::mat4 modelTor = glm::mat4(1.0f);//Inicializamos el modelo
modelTor = glm::translate(glm::mat4(1.0f), glm::vec3(1.0, 0.2f, -8.0f));//Se realiza translación del modelo de toroide
modelTor = glm::rotate(modelTor, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 1.0));//Se realiza rotación del modelo de toroide
modelTor = glm::rotate(modelTor, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0));//Se realiza rotación del modelo de toroide
modelTor = glm::scale(modelTor, glm::vec3(0.2f, 0.2f, 0.2f));//Se realiza escalado del modelo de toroide
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(modelTor));//Se envia modelo de toroide en matriz

glDrawElements(GL_TRIANGLES, 1000, GL_UNSIGNED_INT, 0);//Se dibujan los elementos de la matriz

modelTor = glm::translate(glm::mat4(1.0f), glm::vec3(-1.0, 0.2f, -8.0f));//Se realiza translación del modelo de toroide
modelTor = glm::rotate(modelTor, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 1.0));//Se realiza rotación del modelo de toroide
modelTor = glm::rotate(modelTor, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0));//Se realiza rotación del modelo de toroide
modelTor = glm::scale(modelTor, glm::vec3(0.2f, 0.2f, 0.2f));//Se realiza escalado del modelo de toroide
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(modelTor));//Se envia modelo de toroide en matriz

glDrawElements(GL_TRIANGLES, 1000, GL_UNSIGNED_INT, 0);//Se dibujan los elementos de la matriz
```

```
modelTor = glm::translate(glm::mat4(1.0f), glm::vec3(-1.0, 0.2f, -6.0f));//Se realiza translación del modelo de toroide
modelTor = glm::rotate(modelTor, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 1.0));//Se realiza rotación del modelo de toroide
modelTor = glm::rotate(modelTor, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0));//Se realiza rotación del modelo de toroide
modelTor = glm::scale(modelTor, glm::vec3(0.2f, 0.2f, 0.2f));//Se realiza escalado del modelo de toroide
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(modelTor));//Se envia modelo de toroide en matriz

glDrawElements(GL_TRIANGLES, 1000, GL_UNSIGNED_INT, 0);//Se dibujan los elementos de la matriz

modelTor = glm::translate(glm::mat4(1.0f), glm::vec3(1.0, 0.2f, -6.0f));//Se realiza translación del modelo de toroide
modelTor = glm::rotate(modelTor, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 1.0));//Se realiza rotación del modelo de toroide
modelTor = glm::rotate(modelTor, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0));//Se realiza rotación del modelo de toroide
modelTor = glm::scale(modelTor, glm::vec3(0.2f, 0.2f, 0.2f));//Se realiza escalado del modelo de toroide
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(modelTor));//Se envia modelo de toroide en matriz

glDrawElements(GL_TRIANGLES, 1000, GL_UNSIGNED_INT, 0);//Se dibujan los elementos de la matriz



glBindVertexArray(0);
```

Figura 29: Dibujado de Modelos

- Modelos

```
//Carga de modelo del piso
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos a la matriz el modelo
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);//Ingresamos transparencia a objetos
Piso.Draw(lightingShader);//Dibujamos

//Envíamos Modelo de Fachada a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de fachada
Fachada.Draw(lightingShader);
```

```
//Envíamos Modelo de Planta1 a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Planta1
Planta1.Draw(lightingShader);
//Envíamos Modelo de Planta2 a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Planta1
Planta2.Draw(lightingShader);
//Envíamos Modelo de Planta3 a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Planta3
Planta3.Draw(lightingShader);
//Envíamos Modelo de Planta4 a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Planta4
Planta4.Draw(lightingShader);
//Envíamos Modelo de Planta5 a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Planta5
Planta5.Draw(lightingShader);
//Envíamos Modelo de Planta6 a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Planta6
Planta6.Draw(lightingShader);
//Envíamos Modelo de Sillon1 a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Sillon1
Sillon1.Draw(lightingShader);
//Envíamos Modelo de Sillon2 a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
```

Figura 30: Dibujado de Modelos

- Modelos

```
Sillon2.Draw(lightingShader);
//Envíamos Modelo de Cojines a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Cojines
Cojines.Draw(lightingShader);
//Envíamos Modelo de Mesa a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Mesa
Mesa.Draw(lightingShader);
//Envíamos Modelo de Mesa2 a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Mesa2
Mesa2.Draw(lightingShader);
//Envíamos Modelo de TV a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de TV
TV.Draw(lightingShader);
//Envíamos Modelo de Tapete a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Tapete
Tapete.Draw(lightingShader);
//Envíamos Modelo de Libros a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Libros
Libros.Draw(lightingShader);
```

```
//Enviamos el color alpha
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0, 1.0, 1.0, 1.0);
//Finalizamos vertex array
glBindVertexArray(0);
```

Figura 31: Dibujado de Modelos

- Modelos

```
//Carga de Modelo Pedro Picapiedra
view = camera.GetViewMatrix();//establecemos la vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.0f,0.5f,1.5f));//Transformación de traslación a posición inicial
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Envíamos el modelo a la matriz
CuerpoP.Draw(lightingShader);

view = camera.GetViewMatrix();//Establecemos vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.0f, 0.5f, 1.5f));//Transformación de traslación a posición inicial
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Envíamos el modelo a la matriz
CabezaP.Draw(lightingShader);

view = camera.GetViewMatrix();//Establecemos vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.0f, 0.5f, 1.5f));//Transformación de traslación a posición inicial
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Envíamos el modelo a la matriz
PiesP.Draw(lightingShader);//Dibujamos el modelo  con parametro recibido de shader de iluminación

view = camera.GetViewMatrix();//Establecemos vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.0f, 0.5f, 1.5f));//Transformación de traslación a posición inicial
//model = glm::translate(model, glm::vec3(posX2, posY2, posZ2));//Transformación de traslación a posición inicial
model = glm::rotate(model, glm::radians(-rotBrazoDer), glm::vec3(0.0f, 0.0f, 1.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Envíamos el modelo a la matriz
BrazoIzq.Draw(lightingShader);//Dibujamos el modelo  con parametro recibido de shader de iluminación
```

```
view = camera.GetViewMatrix();//Establecemos vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.0f, 0.5f, 1.5f));//Transformación de traslación a posición inicial
model = glm::rotate(model, glm::radians(-rotBrazoDer), glm::vec3(0.0f, 0.0f, 1.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Envíamos el modelo a la matriz
BrazoDer.Draw(lightingShader);//Dibujamos el modelo  con parametro recibido de shader de iluminación



//Finalizamos vertex array
glBindVertexArray(0);
```

Figura 32: Dibujado de Modelos

- Modelos

```
//Carga de Modelo Pedro Picapiedra
view = camera.GetViewMatrix();//establecemos la vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.0f,0.5f,1.5f));//Transformación de traslación a posición inicial
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Envíamos el modelo a la matriz
CuerpoP.Draw(lightingShader);

view = camera.GetViewMatrix();//Establecemos vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.0f, 0.5f, 1.5f));//Transformación de traslación a posición inicial
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Envíamos el modelo a la matriz
CabezaP.Draw(lightingShader);

view = camera.GetViewMatrix();//Establecemos vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.0f, 0.5f, 1.5f));//Transformación de traslación a posición inicial
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Envíamos el modelo a la matriz
PiesP.Draw(lightingShader);//Dibujamos el modelo  con parametro recibido de shader de iluminación

view = camera.GetViewMatrix();//Establecemos vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.0f, 0.5f, 1.5f));//Transformación de traslación a posición inicial
//model = glm::translate(model, glm::vec3(posX2, posY2, posZ2));//Transformación de traslación a posición inicial
model = glm::rotate(model, glm::radians(-rotBrazoDer), glm::vec3(0.0f, 0.0f, 1.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Envíamos el modelo a la matriz
BrazoIzq.Draw(lightingShader);//Dibujamos el modelo  con parametro recibido de shader de iluminación
```

```
view = camera.GetViewMatrix();//Establecemos vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.0f, 0.5f, 1.5f));//Transformación de traslación a posición inicial
model = glm::rotate(model, glm::radians(-rotBrazoDer), glm::vec3(0.0f, 0.0f, 1.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Envíamos el modelo a la matriz
BrazoDer.Draw(lightingShader);//Dibujamos el modelo  con parametro recibido de shader de iluminación



//Finalizamos vertex array
glBindVertexArray(0);
```

Figura 33: Dibujado de Modelos

- Modelos

```
//Shaggy
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, PosIni3 + glm::vec3(movKitX2,movKitY2, movKitZ2));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(rotKit2), glm::vec3(0.0f, 1.0f, 0.0));//asignamos transofmración de rotación al modelo, usado para cambio de dirección
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos a la matriz el modelo
ShaCuerpo.Draw(lightingShader);//Dibujamos el modelo del dinosaurio

view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, PosIni3 + glm::vec3(movKitX2, movKitY2, movKitZ2));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(rotKit2), glm::vec3(0.0f, 1.0f, 0.0));//asignamos transofmración de rotación al modelo, usado para cambio de dirección
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos a la matriz el modelo
ShaCabeza.Draw(lightingShader);//Dibujamos el modelo del dinosaurio

view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, PosIni3 + glm::vec3(movKitX2, movKitY2, movKitZ2));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(rotKit2), glm::vec3(0.0f, 1.0f, 0.0));//asignamos transofmración de rotación al modelo, usado para cambio de dirección
model = glm::rotate(model, glm::radians(-rot2), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos a la matriz el modelo
ShaBrazoDer.Draw(lightingShader);//Dibujamos el modelo del dinosaurio

view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, PosIni3 + glm::vec3(movKitX2, movKitY2, movKitZ2));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(rotKit2), glm::vec3(0.0f, 1.0f, 0.0));//asignamos transofmración de rotación al modelo, usado para cambio de dirección
model = glm::rotate(model, glm::radians(-rot3), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos a la matriz el modelo
ShaBrazoIzq.Draw(lightingShader);//Dibujamos el modelo del dinosaurio

view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, PosIni3 + glm::vec3(movKitX2, movKitY2, movKitZ2));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(rotKit2), glm::vec3(0.0f, 1.0f, 0.0));//asignamos transofmración de rotación al modelo, usado para cambio de dirección
model = glm::rotate(model, glm::radians(-rot3), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos a la matriz el modelo
ShaPiernaDer.Draw(lightingShader);//Dibujamos el modelo del dinosaurio

view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, PosIni3 + glm::vec3(movKitX2, movKitY2, movKitZ2));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(rotKit2), glm::vec3(0.0f, 1.0f, 0.0));//asignamos transofmración de rotación al modelo, usado para cambio de dirección
model = glm::rotate(model, glm::radians(-rot2), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos a la matriz el modelo
ShäPiernaIzq.Draw(lightingShader);//Dibujamos el modelo del dinosaurio
```

Figura 34: Dibujado de Modelos

- Modelos

```
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, PosIni + glm::vec3(movKitX, movKitY, movKitZ));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f, 1.0f, 0.0));//asignamos transofmración de rotación al modelo, usado para cambio de dirección
model = glm::scale(model, glm::vec3(3.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos a la matriz el modelo
Dino.Draw(lightingShader);//Dibujamos el modelo del dinosaurio

view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, PosIni + glm::vec3(movKitX, movKitY, movKitZ));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f, 1.0f, 0.0));//asignamos transofmración de rotación al modelo, usado para cambio de dirección
model = glm::rotate(model, glm::radians(-rot), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::scale(model, glm::vec3(3.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos a la matriz el modelo
AlaD.Draw(lightingShader);//Dibujamos el modelo del dinosaurio

view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, PosIni + glm::vec3(movKitX, movKitY, movKitZ));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f, 1.0f, 0.0));//asignamos transofmración de rotación al modelo, usado para cambio de dirección
model = glm::rotate(model, glm::radians(rot), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::scale(model, glm::vec3(3.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos a la matriz el modelo
AlaI.Draw(lightingShader);//Dibujamos el modelo del dinosaurio


glBindVertexArray(0);//Finalizamos vertex array
```

```
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, glm::vec3(10.0, 0.0, 15.0));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Fachada.Draw(lightingShader);

view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, glm::vec3(10.0, 0.0, 15.0));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Sillon2.Draw(lightingShader);

view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, glm::vec3(10.0, 0.0, 15.0));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
TV.Draw(lightingShader);

view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, glm::vec3(10.0, 0.0, 15.0));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Mesa.Draw(lightingShader);
```

Figura 35: Dibujado de Modelos

- Modelos

```
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, glm::vec3(12.0, -0.4, 15.3));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(0.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Pablo.Draw(lightingShader);
```

```
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, glm::vec3(27.0, -0.4, 15.3));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(0.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Betty.Draw(lightingShader);
```

Figura 36: Dibujado de Modelos

- Modelos

```
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, glm::vec3(42.0, -0.4, 15.3));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(0.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Vilma.Draw(lightingShader);
```

```
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, glm::vec3(8.0, -0.4, -15.3));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Rajuela.Draw(lightingShader);
```

Figura 37: Dibujado de Modelos

- Modelos

```
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, glm::vec3(23.0, -0.4, -15.3));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Cavernicola.Draw(lightingShader);
```

```
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, glm::vec3(38.0, -0.4, -15.3));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Trajeado.Draw(lightingShader);
```

Figura 38: Dibujado de Modelos

- Modelos

```
//Creación  de modelo de iluminación punto
model = glm::mat4(1);
//Transformación de traslación en iluminación punto
model = glm::translate(model, pointLightPositions[0]);
//Transformación de escalado en iluminación punto
model = glm::scale(model, glm::vec3(0.2f));
//Enviar modelo a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glBindVertexArray(VAO);
//Dibujado de cubo de point light
glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);

//Creación de modelo de iluminación punto
model = glm::mat4(1);
//Transformación de traslación en iluminación punto
model = glm::translate(model, pointLightPositions[1]);
//Transformación de escalado en iluminación punto
model = glm::scale(model, glm::vec3(0.2f));
//Enviar modelo a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glBindVertexArray(VAO);
//Dibujado de cubo de point light
glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);
```

```
//Creación de modelo de iluminación punto
model = glm::mat4(1);
//Transformación de traslación en iluminación punto
model = glm::translate(model, pointLightPositions[2]);
//Transformación de escalado en iluminación punto
model = glm::scale(model, glm::vec3(0.2f));
//Enviar modelo a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glBindVertexArray(VAO);
//Dibujado de cubo de point light
glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);

//Creación de modelo de iluminación punto
model = glm::mat4(1);
//Transformación de traslación en iluminación punto
model = glm::translate(model, pointLightPositions[3]);
//Transformación de escalado en iluminación punto
model = glm::scale(model, glm::vec3(0.2f));
//Enviar modelo a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glBindVertexArray(VAO);
//Dibujado de cubo de point light
glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);
```

Figura 39: Dibujado de Modelos

- Modelos

```
model = glm::mat4(1);
//Transformación de traslación en iluminación punto
model = glm::translate(model, pointLightPositions[4]);
//Transformación de escalado en iluminación punto
model = glm::scale(model, glm::vec3(0.2f, 0.5f, 0.7f));
//Enviar modelo a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glBindVertexArray(VAO);
//Dibujado de cubo de point light
glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);

model = glm::mat4(1);
//Transformación de traslación en iluminación punto
model = glm::translate(model, glm::vec3(10.0f, 3.0f, -15.0f));
//Transformación de escalado en iluminación punto
model = glm::scale(model, glm::vec3(0.2f));
//Enviar modelo a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glBindVertexArray(VAO);
//Dibujado de cubo de point light
glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);
```

```
model = glm::mat4(1);
//Transformación de traslación en iluminación punto
model = glm::translate(model, glm::vec3(25.0f, 3.0f, -15.0f));
//Transformación de escalado en iluminación punto
model = glm::scale(model, glm::vec3(0.2f));
//Enviar modelo a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glBindVertexArray(VAO);
//Dibujado de cubo de point light
glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);

model = glm::mat4(1);
//Transformación de traslación en iluminación punto
model = glm::translate(model, glm::vec3(40.0f, 3.0f, -15.0f));
//Transformación de escalado en iluminación punto
model = glm::scale(model, glm::vec3(0.2f));
//Enviar modelo a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glBindVertexArray(VAO);
//Dibujado de cubo de point light
glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);
// Dibujamos al ultimo el skybox
glDepthFunc(GL_LEQUAL);
//Iniciamos shader de sky box para su uso
SkyBoxshader.Use();
//Movemos el componente de vista de la matriz
view = glm::mat4(glm::mat3(camera.GetViewMatrix()));
//Enviamos la vista y proyección a la matriz
glUniformMatrix4fv(glGetUniformLocation(SkyBoxshader.Program, "view"), 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(glGetUniformLocation(SkyBoxshader.Program, "projection"), 1, GL_FALSE, glm::value_ptr(projection));
```

Figura 40: Dibujado de Modelos

- Declaración de funciones de sonidos en la primera es el intro de programa, segundo sonido ambiental, tercera sonido de animación de gol.

```
//Funcion de Insertado de sonido de intro
void Sonido()
{
    PlaySound(TEXT("picapiedra.wav"), NULL, SND_SYNC );
}

//Funcion de Insertado de sonido de ambientación
void Sonido2()
{

    PlaySound(TEXT("ambiente.wav"), NULL, SND_LOOP | SND_ASYNC);

}

//Función de insertado de sonido de animación gol
void Sonido3()
{

    PlaySound(TEXT("gol.wav"), NULL, SND_ASYNC);

}
```

Figura 41: Funciones de Sonido

- Declaración de función de teclas en el cual se activarán para movimiento de la cámara en la cual se ejecutará cuando se interactué, donde se ingresa las teclas de movimiento de dinosaurio para tomar agua y el movimiento de brazos de Pedro Picapiedra, animaciones definidas, movimiento del dragon, etc.

```
//Control de iniciado de animación de movimiento de dragon
if (keys[GLFW_KEY_I])
{
    circuito = true;
    anim = true;
}

if (keys[GLFW_KEY_N])
{
    circuito2 = true;
    anim3 = true;
}


//Control de pausado de movimiento de dragon
if (keys[GLFW_KEY_O])
{
    circuito = false;
}
```

```
/// Animación Dinosaurio
void animacion()
{
    //Bandera de activación
    if (circuito)
    {
        //Bandera de activación recorrido 1
        if (recorrido1)
        {
            //Desplazamiento en X
            movKitX += 0.1f;
            movKitY -= 0.02f;
            //Bandera de limite de 20 unidades
            if (movKitX > 40 )
            {
                //Cambio de recorrido
                recorrido1 = false;
                recorrido2 = true;
            }
        }
        //Bandera de activación recorrido 2
        if (recorrido2)
        {
            //Cambio de dirección
            rotKit += 1;
            //Bandera de limite de 90 grados
            if (rotKit > 0)
            {
                //Cambio de recorrido
                recorrido2 = false;
                recorrido3 = true;
            }
```

Figura 42: Funciones Animación

- Funciones Animaciones

```
}
//Bandera de activación recorrido 3
if (recorrido3)
{
    //Desplazamiento en Z negativo
    movKitZ -= 0.1f;
    movKitY += 0.03f;
    //Bandera de limite de 20 unidades menos
    if (movKitZ < -20)
    {
        //Cambio de recorrido
        recorrido3 = false;
        recorrido4 = true;
    }
}
//Bandera de activación recorrido 4
if (recorrido4)
{
    //Cambio de dirección
    rotKit += 1;
    //Bandera de limite en 180 grados
    if (rotKit > 90)
    {
        //Cambio de recorrido
        recorrido4 = false;
        recorrido5 = true;
    }
}
//Bandera de activación recorrido 5
if (recorrido5)
{
    //Desplazamiento en X negativo
    movKitX -= 0.1f;
    movKitY -= 0.015f;
    //Bandera de limite de 20 unidades menos
    if (movKitX < -20)
    {
        //Cambio de recorrido
        recorrido5 = false;
        recorrido6 = true;
    }
}
//Bandera de activación recorrido 6
if (recorrido6)
{
    //Cambio de dirección
    rotKit += 1;
    //Bandera de limite en 270 grados
    if (rotKit > 180)
    {
        //Cambio de recorrido
        recorrido6 = false;
        recorrido7 = true;
    }
}
```

Figura 43: Funciones Animación

- Funciones Animación

```
//Bandera de activación recorrido 7
if (recorrido7)
{
    //Desplazamiento en Z positivo
    movKitZ += 0.1f;
    movKitY += 0.03f;
    //Bandera de limite de 0 unidades
    if (movKitZ > 0)
    {
        //Cambio de recorrido
        recorrido7 = false;
        recorrido8 = true;
    }
}
//Bandera de activación recorrido 8
if (recorrido8)
{
    //Cambio de dirección
    rotKit += 1;
    //Bandera de limite en 360 grados
    if (rotKit > 270)
    {
        //Cambio de recorrido
        recorrido8 = false;
        recorrido9 = true;
    }
}
```

```
//Bandera de activación recorrido 9
if (recorrido9)
{
    //Desplazamiento en X positivo
    movKitX += 0.1f;
    movKitY += 0.03f;
    //Bandera de limite de 0 unidades
    if (movKitX > 0)
    {
        rotKit = -90;
        //Cambio de recorrido
        recorrido9 = false;
        recorrido1 = true;
    }
}
}
```

Figura 44: Funciones Animación

- Funciones Animación

```
//Bandera de play
if (play)
{
    //Bandera de pasos actaules mayor a pasos maximos
    if (i_curr_steps >= i_max_steps)
    {
        //Incremento de indice de actual
        playIndex++;
        //Bandera indice actual al indice almacenado no sea mayor, termina animación
        if (playIndex > FrameIndex - 2)
        {
            printf("termina anim\n");
            //Inicializamos a valores iniciales
            playIndex = 0;
            play = false;
        }
        else
        {
            //Reiniciamos contador
            i_curr_steps = 0;
            //interpolamos para realizar
            interpolation();
        }
    }
    else
    {
        //Recorrido de posición
        posX += KeyFrame[playIndex].incX;
        posY += KeyFrame[playIndex].incY;
        posZ += KeyFrame[playIndex].incZ;

        rotCuerpo += KeyFrame[playIndex].rotInc;
        rotPies += KeyFrame[playIndex].rotPies;

        i_curr_steps++;
    }
```

```
//Bandera de play
if (play2)
{
    //Bandera de pasos actaules mayor a pasos maximos
    if (i_curr_steps2 >= i_max_steps2)
    {
        //Incremento de indice de actual
        playIndex2++;
        //Bandera de posición 2
        if (playIndex2 == 1)
        {
            //Función de sonido 3, sonido de gol
            Sonido3();
        }
        //Bandera indice actual al indice almacenado no sea mayor, termina animación
        if (playIndex2 > FrameIndex2 - 2)
        {
            printf("termina anim\n");
            //Función de sonido 2, sonido de ambiente
            Sonido2();
            //Inicializamos a valores iniciales
            playIndex2 = 0;
            play2 = false;
        }
        else
        {
            //Reiniciamos contador
            i_curr_steps2 = 0;
            //interpolamos para realizar
            interpolation2();
        }
    }
```

Figura 45: Funciones Animación

- Funciones Animación

```
        else
        {
            //Reiniciamos contador
            i_curr_steps2 = 0;
            //interpolamos para realizar
            interpolation2();
        }
    }
    else
    {
        //Recorrido de posición
        posX2 += KeyFrame2[playIndex2].incX2;
        posY2 += KeyFrame2[playIndex2].incY2;
        posZ2 += KeyFrame2[playIndex2].incZ2;

        rotBrazoDer += KeyFrame2[playIndex2].rotInc2;

        i_curr_steps2++;
    }

}
```

```
if (anim3)
{
    if (rot2 <= 45.0f)
    {
        rot2 = rot2 + 0.5f;
        rot3 = rot3 - 0.5f;
    }
    else
    {
        anim3 = false;
        anim4 = true;
    }
}
if (anim4)
{
    if (rot2 >= -45.0f)
    {
        rot2 = rot2 - 0.5f;
        rot3 = rot3 + 0.5f;
    }
    else
    {
        anim4 = false;
        anim3 = true;
    }
}
```

Figura 46: Funciones Animación

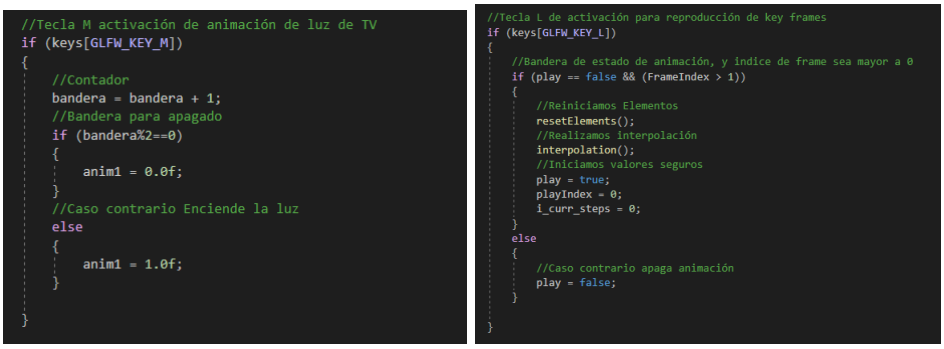
- Funciones Animación

```
//Bandera de activación
if (circuito2)
{
    //Bandera de activación recorrido 1
    if (recorrido10)
    {
        //Desplazamiento en X
        movKitX2 -= 0.05f;
        //Bandera de limite de 20 unidades
        if (movKitX2 < -8)
        {
            //Cambio de recorrido
            recorrido10 = false;
            recorrido11 = true;
        }
    }
    //Bandera de activación recorrido 2
    if (recorrido11)
    {
        //Cambio de dirección
        rotKit2 = 90;
        movKitX2 += 0.05f;
        //Bandera de limite de 90 grados
        if (movKitX2 > 8)
        {
            rotKit2 = -90;
            //Cambio de recorrido
            recorrido11 = false;
            recorrido10 = true;
        }
    }
}
```

```
if (anim)
{
    if (rot <= 45.0f)
    {
        rot = rot + 0.5f;
    }
    else
    {
        anim = false;
        anim2 = true;
    }
}
if (anim2)
{
    if (rot >= -45.0f)
    {
        rot = rot - 0.5f;
    }
    else
    {
        anim2 = false;
        anim = true;
    }
}
```

Figura 47: Funciones Animación

- Funciones Animación



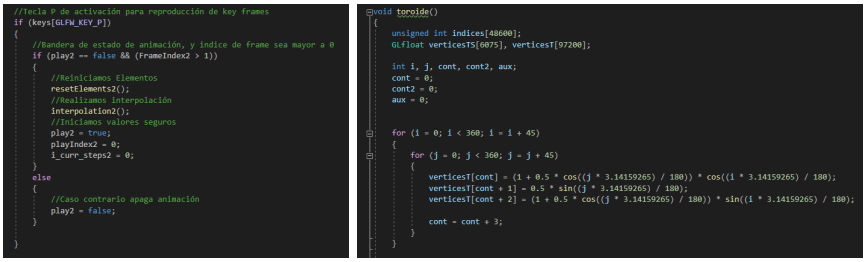
```
//Tecla M activación de animación de luz de TV
if (keys[GLFW_KEY_M])
{
    //Contador
    bandera = bandera + 1;
    //Bandera para apagado
    if (bandera%2==0)
    {
        anim1 = 0.0f;
    }
    //Caso contrario Enciende la luz
    else
    {
        anim1 = 1.0f;
    }

}
```

```
//Tecla L de activación para reproducción de key frames
if (keys[GLFW_KEY_L])
{
    //Bandera de estado de animación, y indice de frame sea mayor a 0
    if (play == false && (FrameIndex > 1))
    {
        //Reiniciamos Elementos
        resetElements();
        //Realizamos interpolación
        interpolation();
        //Iniciamos valores seguros
        play = true;
        playIndex = 0;
        i_curr_steps = 0;
    }
    else
    {
        //Caso contrario apaga animación
        play = false;
    }
}
```

Figura 48: Funciones Animación

- Funciones Animación



```
//Tecla P de activación para reproducción de key frames
if (keys[GLFW_KEY_P])
{
    //Bandera de estado de animación, y indice de frame sea mayor a 0
    if (play2 == false && (FrameIndex2 > 1))
    {
        //Reiniciamos Elementos
        resetElements2();
        //Realizamos interpolación
        interpolation2();
        //Iniciamos valores seguros
        play2 = true;
        playIndex2 = 0;
        i_curr_steps2 = 0;
    }
    else
    {
        //Caso contrario apaga animación
        play2 = false;
    }
}
```

```
void toroide()
{
    unsigned int indices[48600];
    GLfloat verticesTS[6075], verticesT[97200];

    int i, j, cont, cont2, aux;
    cont = 0;
    cont2 = 0;
    aux = 0;


    for (i = 0; i < 360; i = i + 45)
    {
        for (j = 0; j < 360; j = j + 45)
        {
            verticesT[cont] = (1 + 0.5 * cos((j * 3.14159265) / 180)) * cos((i * 3.14159265) / 180);
            verticesT[cont + 1] = 0.5 * sin((j * 3.14159265) / 180);
            verticesT[cont + 2] = (1 + 0.5 * cos((j * 3.14159265) / 180)) * sin((i * 3.14159265) / 180);

            cont = cont + 3;
        }
    }
```

Figura 49: Funciones Animación

- Función toroide

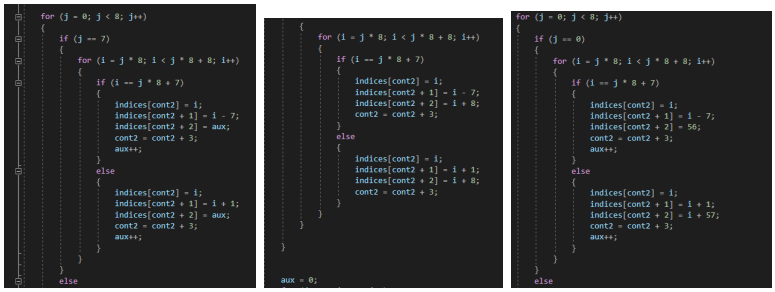


Figura 50: Funciones Animación

- Función toroide

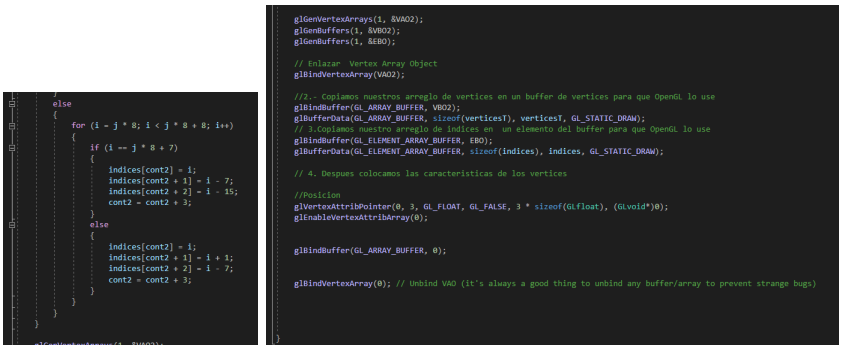


Figura 51: Funciones Animación

1.6. Conclusiones

- La realización del proyecto fue increíble debido a que no se conocía la forma de construcción gráfica, en este aspecto se conoció desde el momento creativo, creación y producción, por el cual se puede pasar una creación grafica fueron pocos metodos los que vimos, los basicos y al investigar dedsde los previos y cada funcion, se aprende demasiado de igual forma el realizar el proyecto reafirma cada conocimiento aprendido en el curso, ya que se implementa todo lo visto en clase, para este proyecto concluido me llevo mucho conocimiento construido y a su vez uso de programas que no conocía o sabía usar, por esta parte siento que fue un proyecto relacionado a lo laboral ya que se tenian fechas de entrega del mismo.

2. English Version

2.1. Factual

This project development was made to envolve the use with virtual route using OpenGL in wich could navigate to the virtual room, objects, animations and other side that i can develop this project with the knowledge of the course, development our imagination for this project because is an enviroment 3D and the cartoon of flintstones is unwrap in 2D, of this shape we being able to development the virtual room and his facade, in this case the user feels inside to the universe of flintstones that is the main reason of this project, so this project was made through all the knowledge gained in the course, and gain as a result of this project was develop with effort, perfomance and compromise for reach the main factual, use all knowledge of the course.

2.2. Scope

To achieve this develop of a facade very close to realism of the original facade, as his texture and objects, the main interaction is a free camera that allows us move inside in the virtual enviroment, interact with the animations, i see a great results, in which we can see the essential enviroment and achieve with interact with them. the result of this main factual is did it with the agreed times and assigned, have been achieve with the final delivery, acoording to the gantt diagram we established in the document that we were able to do with precision and quickness,this being a greatfuly develop, for everything learn in the progess.

2.3. Limitations

The main objectives of this subject we get it, but for the time to this project is short, we work with an short scope, if we got more time we are going to see more topics, get knowledge, learn functions, animations, etc. Which is the main reason for this limitations is the study program, in turn on the professor gave all the knowledge, information and some tips for did this project,some functions that we don't know, was combined with C++ and OpenGL, and this works. Another limitation was the resource of my computer, which had to be adapted and reduce the vertex created, and using voxel art to something objects.

2.4. Gantt Diagram

It is the development of a diagram in which put delivery dates, time and progress established to the assigned time, and complexity of them, in another side the urgency of delivery delays at the time, so this diagram works very well.

Nombre de la tarea	Fecha de inicio	Fecha de finalización	Asignado	Estado
Proyecto Picapiedra	07.03.2022	11.05.2022	Alonso	Terminado
Propuesta de Fachada	09.03.2022	14.03.2022	Alonso	Terminado
Primer Objeto	16.03.2022	22.03.2022	Alonso	Terminado
Dibujado del Primer Objeto	16.03.2022	18.03.2022	Alonso	Terminado
Texturizado del Primer Objeto	19.03.2022	20.03.2022	Alonso	Terminado
Programado del Primer Objeto en OpenGL	20.03.2022	21.03.2022	Alonso	Terminado
Segundo Objeto	23.03.2022	29.03.2022	Alonso	Terminado
Dibujado del Segundo Objeto	24.03.2022	25.03.2022	Alonso	Terminado
Texturizado del Segundo Objeto	26.03.2022	27.03.2022	Alonso	Terminado
Programado del Segundo Objeto en OpenGL	28.03.2022	29.03.2022	Alonso	Terminado
Tercer Objeto	30.03.2022	05.04.2022	Alonso	Terminado
Dibujado del Tercer Objeto	31.03.2022	01.04.2022	Alonso	Terminado
Texturizado del Tercer Objeto	02.04.2022	03.04.2022	Alonso	Terminado
Programado del Tercer Objeto en OpenGL	04.04.2022	05.04.2022	Alonso	Terminado
Cuarto Objeto	06.04.2022	12.04.2022	Alonso	Terminado
Dibujado del Cuarto Objeto	07.04.2022	08.04.2022	Alonso	Terminado
Texturizado del Cuarto Objeto	09.04.2022	10.04.2022	Alonso	Terminado
Programado del Cuarto Objeto en OpenGL	11.04.2022	11.04.2022	Alonso	Terminado
Quinto Objeto	13.04.2022	20.04.2022	Alonso	Terminado
Dibujado del Quinto Objeto	14.04.2022	15.04.2022	Alonso	Terminado
Texturizado del Quinto Objeto	16.04.2022	17.04.2022	Alonso	Terminado
Programado del Quinto Objeto en OpenGL	18.04.2022	18.04.2022	Alonso	Terminado
Sexto Objeto	20.04.2022	27.04.2022	Alonso	Terminado
Dibujado del Sexto Objeto	21.04.2022	22.04.2022	Alonso	Terminado
Texturizado del Sexto Objeto	23.04.2022	24.04.2022	Alonso	Terminado
Programado del Sexto Objeto en OpenGL	25.04.2022	25.04.2022	Alonso	Terminado
Septimo Objeto	27.04.2022	03.05.2022	Alonso	Terminado
Dibujado del Septimo Objeto	28.04.2022	28.04.2022	Alonso	Terminado
Texturizado del Septimo Objeto	29.04.2022	29.04.2022	Alonso	Terminado
Programado del Septimo Objeto en OpenGL	02.05.2022	02.05.2022	Alonso	Terminado
Dibujado del Fachada Objeto	11.04.2022	12.04.2022	Alonso	Terminado
Texturizado del Fachada Objeto	12.04.2022	13.04.2022	Alonso	Terminado
Programado del Fachada Objeto en OpenGL	14.04.2022	14.04.2022	Alonso	Terminado
Programado de Primer Animación en OpenGL	14.04.2022	15.04.2022	Alonso	Terminado
Programado de Segunda Animación en OpenGL	20.04.2022	21.04.2022	Alonso	Terminado
Programado de Tercer Animación en OpenGL	02.05.2022	02.05.2022	Alonso	Terminado
Programado de Cuarto Animación en OpenGL	04.05.2022	05.05.2022	Alonso	Terminado
Programado de Quinto Animación en OpenGL	05.05.2022	07.05.2022	Alonso	Terminado
Desarrollo de Propuesta de Animación Mejorado	17.05.2022	17.05.2022	Alonso	Terminado
Desarrollo de Bitacora	18.05.2022	19.05.2022	Alonso	Terminado
Boceto	19.05.2022	19.05.2022	Alonso	Terminado
Mejora de Proyecto Final	20.05.2022	24.05.2022	Alonso	Terminado
Desarrollo de Documentación	24.05.2022	25.05.2022	Alonso	Terminado

Figura 52: Gantt Diagram Part 1

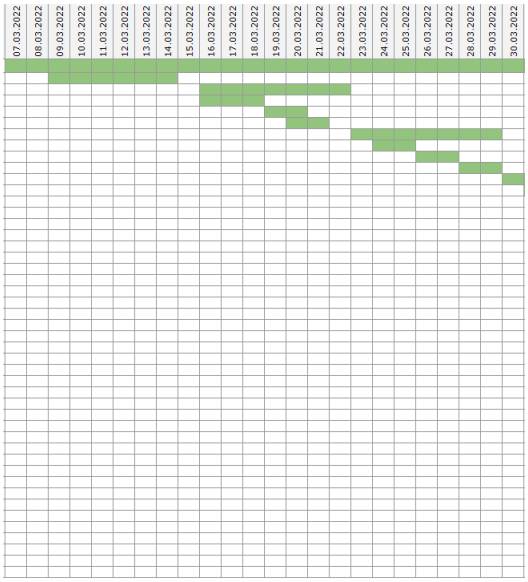


Figura 53: Gantt Diagram Part 2

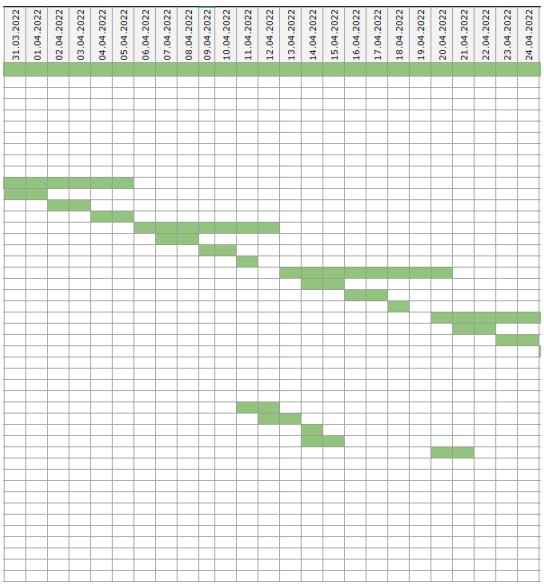


Figura 54: Gantt Diagram Part 3

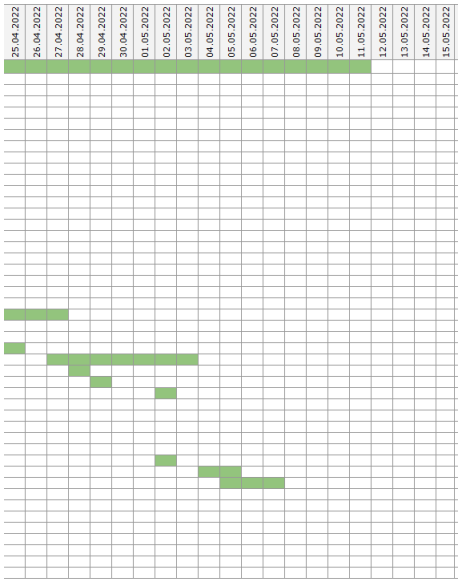


Figura 55: Gantt Diagram Part 4

2.5. Documentation

- The following image we can see the declaration of libreries to use, libreries to processing in the first section, libreries to include OpenGL, another way for used to make the transformation in the models, as glm, so we found another librery wich is allow us create and load textured in the models, as SOIL2, for the last the libreria created by us.

```
/*Sección de declaración de bibliotecas y dependecias a usar para su procesamiento*/
#include <iostream>
#include <cmath>
#include <Windows.h>
#include <mmsystem.h>

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

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

// Libreria de std image.h
#include "stb_image.h"

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

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


// Libreria creadas
#include "Shader.h"
#include "Camera.h"
#include "Model.h"
#include "Texture.h"
```

Figura 56: Librery Declaration

- The declaration of this functions that we are going to use, for using keyboard,mouse and special funcitions for creating animations and flags to sound functions.

```
/*Declaración de las funciones a utilizar*/
// Function prototypes
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode);
void MouseCallback(GLFWwindow* window, double xPos, double yPos);
void DoMovement();
void animacion();
void animacion2();
void animacion3();
void Sonido();
void Sonido2();
void Sonido3();
```

Figura 57: Functions Declaration

- We create this vairables to use for the initialization of a window size in wich resoultion and it will be executed.The declaration of this has a relation to the screen to be in the middle of resolution, as a mind point of window.The array is declared for the assignment of keys to be used, in this case is asigned a total of 1024 of memory.Activation flags for mouse movement, sending the flas as true. Lighting atributes that having a initial positions is (0,0,0).Initialization of the initial position, object 1 (Long Neck Dinosaur), object 2 (Flintstone Pedro) which they will move by key Frame, initial position is (0,0,0).The first variable works as a flag, which it will activate animation 1, the second will be a counter for animation 1, the third works as capture the processor pulses, and the last as a flag that works to turn on and off the TV.

```
// Dimensiones de la ventana al crear
const GLuint WIDTH = 1920, HEIGHT = 1080;
int SCREEN_WIDTH, SCREEN_HEIGHT;

// Declaración de modelo de camara y su posición
Camera  camera(glm::vec3(15.0f, 1.0f, 0.0f));
GLfloat lastX = WIDTH / 2.0;
GLfloat lastY = HEIGHT / 2.0;

//Arreglo para teclas
bool keys[1024];

//Variable de uso de movimiento de ratón (periferico)
bool firstMouse = true;


// Atributos del uso de ilumincación, posición inicial
glm::vec3 lightPos(0.0f, 0.0f, 0.0f);

//Declaración de posición inical de objeto a mover con Key Frames
glm::vec3 PosIni(0.0f, 1.2f, 8.0f);
glm::vec3 PosIni2(-1.0f, 0.5f, 1.5f);
glm::vec3 PosIni3(8.0f, 0.9f, 0.0f);


/*Declaración de variables, para activar animaciones */
bool active;
float anim1=0.0f;
float tiempo;
int bandera;
```

Figura 58: Variable Declaration

- Array of initial points for the position of the point light in which it will be draw from the beginning. Variable to use to the animation 3, the first is use to the movement in X, the second is use to the movement in Z, and the third is use to the object rotation.

```
unsigned int indices[48600];
GLuint VBO2, VAO2, EBO;


// Posición inicial del pointlight y cubo
glm::vec3 pointLightPositions[] = {
    glm::vec3(0.0f,3.0f, 0.0f),
    glm::vec3(10.0, 3.0, 15.0),
    glm::vec3(25.0, 3.0, 15.0),
    glm::vec3(40.0, 3.0, 15.0),
    glm::vec3(1.5f, 0.82f, 1.15f),
};

//Variables de movimiento y rotación para animación 3
float movKitX = 0.0;
float movKitZ = 8.0;
float movKitY = 12.0;
float rotKit = -90.0;


float movKitX2 = 0.0;
float movKitZ2 = 0.0;
float movKitY2 = 0.0;
float rotKit2 = -90.0;
float rot2 = 0.0f;
float rot3 = 0.0f;
float rot4 = 0.0f;
float rot5 = 0.0f;
```

Figura 59: Animation Variable 2

- Variable to use as a flag in which it will be activate the route of the dragon. Flags variable to make the change of position of the route when the process is completed.

```
bool anim3 = false;
bool anim4 = false;
bool anim5 = false;
bool anim6 = false;
bool recorrido10 = true;
bool recorrido11 = false;


//Bandera a usar la activación de recorrido de movimiento
bool circuito = false;
bool circuito2 = false;

/*Banderas para activar movimiento y recorrido por secciones*/
bool recorrido1 = true;
bool recorrido2 = false;
bool recorrido3 = false;
bool recorrido4 = false;
bool recorrido5 = false;
bool recorrido6 = false;
bool recorrido7 = false;
bool recorrido8 = false;
bool recorrido9 = false;

bool anim = false;
bool anim2 = false;
float rot = 0.0f;
```

Figura 60: Route Flags

- Initialization of variables to the calculate of frames used, deltatime, will be used to the difference between the current and last frame, and the lastframe will store the current frame.Initialization of variables to safe values, this variables will be use to increment the position and stored on the frame array, the last variable will be use to the model movement. Definition as a constant of the variable max_frame, which will be the limit of store frame . Variables for the total number of position, through of position array and sotring each one in an independent location, that the maximum is 190, and the last will be use to measur the current route.

```
// Declaración de medida de tiempo entre frames usado y el ultimo frame
GLfloat deltaTime = 0.0f;   // Diferencia de tiempo entre frames
GLfloat lastFrame = 0.0f;   // Tiempo del Ultimo Frame

// Declaración de los key frames, primer key frame
float posX = PosIni.x, posY = PosIni.y, posZ = PosIni.z, rotCuerpo = 0, rotPies=0;

// Declaración de los key frames, segundo key frame
float posX2 = PosIni.x, posY2 = PosIni.y, posZ2 = PosIni2.z, rotBrazoDer = 0;

//Definición de memoria de los key frames, valor 4
#define MAX_FRAMES 4

//Variables a usar de numero maximo de pasos y paso actual, uso para key frame
int i_max_steps = 190;
int i_curr_steps = 0;

//Variables a usar de numero maximo de pasos y paso actual, uso para key frame 2
int i_max_steps2 = 190;
int i_curr_steps2 = 0;
```

Figura 61: Max and Actual of frames

- Definition of frame structure 1 and 2, in which the following properties, position in X,Y,Z in different variables and increase of them, in the same way the object to rotate by key frame, for his increase we are going to created a variable.

```
//Definición Estructural del Frame, para acceso a sus atributos
typedef struct _frame
{
    //Variables para GUARDAR Key Frames
    float posX;     //Variable para PosicionX
    float posY;     //Variable para PosicionY
    float posZ;     //Variable para PosicionZ
    float incX;     //Variable para IncrementoX
    float incY;     //Variable para IncrementoY
    float incZ;     //Variable para IncrementoZ
    float rotCuerpo; //Variable para Rotación de Cuerpo
    float rotInc; //Variable para guardar rotación
    float rotInc2;
    float rotPies;

}FRAME;

//Definición Estructural del Frame 2, para acceso a sus atributos
typedef struct _frame2
{
    //Variables para GUARDAR Key Frames
    float posX2;        //Variable para PosicionX
    float posY2;        //Variable para PosicionY
    float posZ2;        //Variable para PosicionZ
    float incX2;        //Variable para IncrementoX
    float incY2;        //Variable para IncrementoY
    float incZ2;        //Variable para IncrementoZ
    float rotBrazoDer; //Variable para guardar rotación de brazos
    float rotInc2;  //Variable para guardar incremento de brazos

}FRAME2;
```

Figura 62: Frame Struct 1 and 2

- Declaration of variable to work throught frames, declaring an array called key frame, with memory of 4, the following variable will use the index frame for the route each frame, a flag for the activation and route, and the index for through the route.

```
/*
* Declaración de arreglo Key Frame con tamaño 4
* Variable para introducir los datos de posición e interpolación al frame
* Bandera de activación de animación
* Indice de arreglo para recorrido
*/
FRAME KeyFrame[MAX_FRAMES];
int FrameIndex = 0;         //introducir datos
bool play = false;
int playIndex = 0;

/*
* Declaración de arreglo Key Frame 2 con tamaño 4
* Variable para introducir los datos de posición e interpolación al frame
* Bandera de activación de animación
* Indice de arreglo para recorrido
*/
FRAME2 KeyFrame2[MAX_FRAMES];
int FrameIndex2 = 0;            //introducir datos
bool play2 = false;
int playIndex2 = 0;
```

Figura 63: Variables for Frame Key 1 and 2

- Array that it will have the vertex of triangles for draw the lighting box, show to simulate a spotlight by TV.

```
//Areglo de los vertices para dibujo de cubo de iluminación
float vertices[] = {
    -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
       0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
       0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
       0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
      -0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
      -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,

      -0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
       0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
       0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
       0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
      -0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
      -0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f,

      -0.5f,  0.5f,  0.5f, -1.0f,  0.0f,  0.0f,
      -0.5f,  0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
      -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
      -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
      -0.5f, -0.5f,  0.5f, -1.0f,  0.0f,  0.0f,
      -0.5f,  0.5f,  0.5f, -1.0f,  0.0f,  0.0f,

       0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f,
       0.5f,  0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
       0.5f, -0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
       0.5f, -0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
       0.5f, -0.5f,  0.5f,  1.0f,  0.0f,  0.0f,
       0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f,

      -0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,
       0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,
       0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
       0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
      -0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
      -0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,

      -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,
       0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,
       0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
       0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
      -0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
      -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f
};
```

Figura 64: Vertex Array

- Function to restart the element when playback the animation, the initial position is stored in the first location and sent to the current positon, to restart the object position .

```
//Declaración de model de luz
glm::vec3 Light1 = glm::vec3(0);


//Función para reiniciar posición inicial de objeto de frame 1
void resetElements(void)
{
    //Variable regresando la posición inicial almacenada en el frame arreglo de posición 0
    posX = KeyFrame[0].posX;
    posY = KeyFrame[0].posY;
    posZ = KeyFrame[0].posZ;

    rotCuerpo = KeyFrame[0].rotCuerpo;
    rotPies = KeyFrame[0].rotPies;
}

//Función para reiniciar posición inicial de objeto
void resetElements2(void)
{
    //Variable regresando la posición inicial almacenada en el frame arreglo de posición 0
    posX2 = KeyFrame2[0].posX2;
    posY2 = KeyFrame2[0].posY2;
    posZ2 = KeyFrame2[0].posZ2;

    rotBrazoDer = KeyFrame2[0].rotBrazoDer;


}
```

Figura 65: ResetElements Function

- Interpolation function, this works to calculate the increase, this increase will define the playback speed of our animation by key frame, the formula is:

$$Incremento = \frac{posx.siguiente - posx.actual}{numeromaximodepasos}$$

```
//Función de calculo de interpolación para realizar la animación y su velocidad de incremento, siendo lineal.
void interpolation(void)
{

    KeyFrame[playIndex].incX = (KeyFrame[playIndex + 1].posX - KeyFrame[playIndex].posX) / i_max_steps; //Incremento en X
    KeyFrame[playIndex].incY = (KeyFrame[playIndex + 1].posY - KeyFrame[playIndex].posY) / i_max_steps; //Incremento en Y
    KeyFrame[playIndex].incZ = (KeyFrame[playIndex + 1].posZ - KeyFrame[playIndex].posZ) / i_max_steps; //Incremento en Z

    KeyFrame[playIndex].rotInc = (KeyFrame[playIndex + 1].rotCuerpo - KeyFrame[playIndex].rotCuerpo) / i_max_steps; //Incremeneto en rotación de objeto
    KeyFrame[playIndex].rotPies= (KeyFrame[playIndex + 1].rotPies - KeyFrame[playIndex].rotPies) / i_max_steps; //Incremeneto en rotación de objeto
}

//Función de calculo de interpolación para realizar la animación y su velocidad de incremento, siendo lineal.
void interpolation2(void)
{

    KeyFrame2[playIndex2].incX2 = (KeyFrame2[playIndex2 + 1].posX2 - KeyFrame2[playIndex2].posX2) / i_max_steps2; //Incremento en X
    KeyFrame2[playIndex2].incY2 = (KeyFrame2[playIndex2 + 1].posY2 - KeyFrame2[playIndex2].posY2) / i_max_steps2; //Incremento en Y
    KeyFrame2[playIndex2].incZ2 = (KeyFrame2[playIndex2 + 1].posZ2 - KeyFrame2[playIndex2].posZ2) / i_max_steps2; //Incremento en Z

    KeyFrame2[playIndex2].rotInc2 = (KeyFrame2[playIndex2 + 1].rotBrazoDer - KeyFrame2[playIndex2].rotBrazoDer) / i_max_steps2; // Incremeneto en rotación de objeto
}
```

Figura 66: Interpolation Function

- The shaders will be initialized, the first is used to lighting management, the second is for pointlight and the last is for skybox construction.

```
//Se llama la función toroide
toroide();


/*
* Inicializamos los shaders a usar
* Shader para iluminación
* Shader para iluminación
* Shader de Skybox
*/
Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting.frag");
Shader lampShader("Shaders/lamp.vs", "Shaders/lamp.frag");
Shader SkyBoxshader("Shaders/SkyBox.vs", "Shaders/SkyBox.frag");
```

Figura 67: Shaders Initialized and Toroide

- The built of this objects to construction of Pedro Flintstone, this we created the object by a model for OpenGL. The built of this objects to construction of Pedro Flintstone, this we created the object by a model for OpenGL.The animation shader is called, set as an external library initialized in a local anim variable.The built of this for objects inside facade: Sofa, Table, Book, TV, Pillow and Plant. So we set all the objects created by a model for OpenGL.

```
//Pedro Picapiedra
//Modelo de la Cabeza
Model CabezaP((char*)"Models/Obj/Models/Pedro/Cabeza/Cabeza.obj");
//Modelo de el Cuerpo
Model CuerpoP((char*)"Models/Obj/Models/Pedro/Cuerpo/CuerpoP.obj");
//Modelo del Brazo Izquierdo
Model BrazoIzq((char*)"Models/Obj/Models/Pedro/BrazoIzq/BrazoIzq.obj");
//Modelo del Brazo Derecho
Model BrazoDer((char*)"Models/Obj/Models/Pedro/BrazoDer/BrazoDer.obj");
//Modelo de Pies
Model PiesP((char*)"Models/Obj/Models/Pedro/Pies/Pies.obj");

Model Carro((char*)"Models/Obj/Models/Carro/Carro.obj");
//Modelo de Dinosaurio
Model AlaD((char*)"Models/Obj/Models/Dragon/AlaDerecha/AlaDerecha.obj");
Model AlaI((char*)"Models/Obj/Models/Dragon/AlaIzquierda/AlaIzquierda.obj");
Model Dino((char*)"Models/Obj/Models/Dragon/Cuerpo/Cuerpo.obj");
```

```
//Shader creado para animacion del agua
Shader Anim("Shaders/anim.vs", "Shaders/anim.frag");
//Modelo de el agua
Model Agua((char*)"Models/Obj/Models/Agua/Agua.obj");
//Modelo del dinosaurio cuerpo
Model DinosaurioCuerpo((char*)"Models/Obj/Models/DinoPerson/Cuerpo/Cuerpo.obj");
//Modelo del dinosaurio pies traseros
Model DinosaurioPiesA((char*)"Models/Obj/Models/DinoPerson/PiesA/PiesA.obj");
//Modelo del dinosaurio pies delanteros
Model DinosaurioPiesD((char*)"Models/Obj/Models/DinoPerson/PiesD/PiesD.obj");
//Modelo del piso
Model Piso((char*)"Models/Obj/Models/Piso/Piso.obj");
//Modelo de la fachada
Model Fachada((char*)"Models/Obj/Models/Fachada/Fachada.obj");
//Modelo de Planta 1
Model Planta1((char*)"Models/Obj/Models/Planta1/Planta1.obj");
//Modelo de Planta 2
Model Planta2((char*)"Models/Obj/Models/Planta2/Planta2.obj");
//Modelo de Planta 3
Model Planta3((char*)"Models/Obj/Models/Planta3/Planta3.obj");
//Modelo de Planta 4
Model Planta4((char*)"Models/Obj/Models/Planta4/Planta4.obj");
```

Figura 68: Objects Model Initialize

```
//Modelo de Planta 5
Model Planta5((char*)"Models/Obj/Models/Planta5/Planta5.obj");
//Modelo de Planta 6
Model Planta6((char*)"Models/Obj/Models/Planta6/Planta6.obj");
//Modelo de Sillon1
Model Sillon1((char*)"Models/Obj/Models/Sillon1/Sillon.obj");
//Modelo de Sillon2
Model Sillon2((char*)"Models/Obj/Models/Sillon2/Sillon2.obj");
//Modelo de Cojines
Model Cojines((char*)"Models/Obj/Models/Cojines/Cojines.obj");
//Modelo de Mesa
Model Mesa((char*)"Models/Obj/Models/Mesa/Mesa.obj");
//Modelo de Mesa2
Model Mesa2((char*)"Models/Obj/Models/Mesa2/Mesa2.obj");
//Modelo de TV
Model TV((char*)"Models/Obj/Models/TV/TV.obj");
//Modelo de Tapete
Model Tapete((char*)"Models/Obj/Models/Tapete/Tapete.obj");
```

```
//Modelo de Tapete
Model Tapete((char*)"Models/Obj/Models/Tapete/Tapete.obj");
//Modelo de Libros
Model Libros((char*)"Models/Obj/Models/Libros/Libros.obj");
//Modelo de Montañas
Model Montana((char*)"Models/Obj/Models/Montaña/Montaña.obj");
//Modelo de Rocas
Model Roca((char*)"Models/Obj/Models/Roca/Roca.obj");
//Modelo de Pablo
Model Pablo((char*)"Models/Obj/Models/Pablo/Pablo.obj");
//Modelo de Betty
Model Betty((char*)"Models/Obj/Models/Betty/Betty.obj");
//Modelo de Vilma
Model Vilma((char*)"Models/Obj/Models/Vilma/Vilma.obj");
//Modelo de Rajuela
Model Rajuela((char*)"Models/Obj/Models/Rajuela/Rajuela.obj");
//Modelo de Cavernicola
Model Cavernicola((char*)"Models/Obj/Models/Cavernicola/Cavernicola.obj");
//Modelo de Trajeado
Model Trajeado((char*)"Models/Obj/Models/Trajeado/Trajeado.obj");
```

Figura 69: Objects Model Initialize

- We will assign the initial position for the both key frame, in this case is Key frame 1 and key frame 2

```
KeyFrame[0].posX= 0;
KeyFrame[0].posY = 1.5;
KeyFrame[0].posZ= 8.0;
KeyFrame[1].posX = 0;
KeyFrame[1].posY = 1.0;
KeyFrame[1].posZ = 8.0;
KeyFrame[2].posX = 0;
KeyFrame[2].posY = 1.0;
KeyFrame[2].posZ = 8.0;
KeyFrame[3].posX = 0;
KeyFrame[3].posY = 1.0;
KeyFrame[3].posZ = 8.0;
KeyFrame[0].rotCuerpo = 0;
KeyFrame[1].rotCuerpo = 45;
KeyFrame[2].rotCuerpo = -1;
KeyFrame[0].rotPies= 0;
KeyFrame[1].rotPies = 45;
KeyFrame[2].rotPies= -1;

FrameIndex = 4;
```

```
KeyFrame2[0].rotBrazoDer = 0;
KeyFrame2[1].rotBrazoDer = -90;
KeyFrame2[2].rotBrazoDer = 0;
FrameIndex2 = 4;
```

Figura 70: Frames Position

- The array is created for set the vertex of the box to draw the skybox, so do each vetex to built.

```
//Arreglo de vertices para dibujo de cubo para skybox
GLfloat skyboxVertices[] = {
    // Positions
    -1.0f,  1.0f, -1.0f,
    -1.0f, -1.0f, -1.0f,
    1.0f, -1.0f, -1.0f,
    1.0f, -1.0f, -1.0f,
    1.0f,  1.0f, -1.0f,
    -1.0f,  1.0f, -1.0f,

    -1.0f, -1.0f,  1.0f,
    -1.0f, -1.0f, -1.0f,
    -1.0f,  1.0f, -1.0f,
    -1.0f,  1.0f, -1.0f,
    -1.0f,  1.0f,  1.0f,
    -1.0f, -1.0f,  1.0f,

    1.0f, -1.0f, -1.0f,
    1.0f, -1.0f,  1.0f,
    1.0f,  1.0f,  1.0f,
    1.0f,  1.0f,  1.0f,
    1.0f,  1.0f, -1.0f,
    1.0f, -1.0f, -1.0f,

    -1.0f, -1.0f,  1.0f,
    -1.0f,  1.0f,  1.0f,
    1.0f,  1.0f,  1.0f,
    1.0f,  1.0f,  1.0f,
    1.0f, -1.0f,  1.0f,
    -1.0f, -1.0f,  1.0f,

    -1.0f,  1.0f, -1.0f,
    1.0f,  1.0f, -1.0f,
    1.0f,  1.0f,  1.0f,
    1.0f,  1.0f,  1.0f,
    -1.0f,  1.0f,  1.0f,
    -1.0f,  1.0f, -1.0f,

    -1.0f, -1.0f, -1.0f,
    -1.0f, -1.0f,  1.0f,
    1.0f, -1.0f, -1.0f,
    1.0f, -1.0f, -1.0f,
    -1.0f, -1.0f,  1.0f,
    1.0f, -1.0f,  1.0f
};
```

Figura 71: Skybox Vertex Box

- To load of texture for the skybox, so load the texture of the face of cubes in his vertex.

```
//Carga de imagenes para texturizar el skybox
vector<const GLchar*> faces;
faces.push_back("SkyBox/right.tga");
faces.push_back("SkyBox/left.tga");
faces.push_back("SkyBox/top.tga");
faces.push_back("SkyBox/bottom.tga");
faces.push_back("SkyBox/front.tga");
faces.push_back("SkyBox/back.tga");
```

Figura 72: Skybox Texture

- Call of the sound functions for playback as a progam intro, and second funcion is the ambient sound.

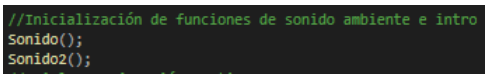


Figura 73: Sound Function Call

- Invocation of functions for keyboard capture, constant movement and animation function, from this part the code was in a infinite loop, in which his state will change according to the flags.

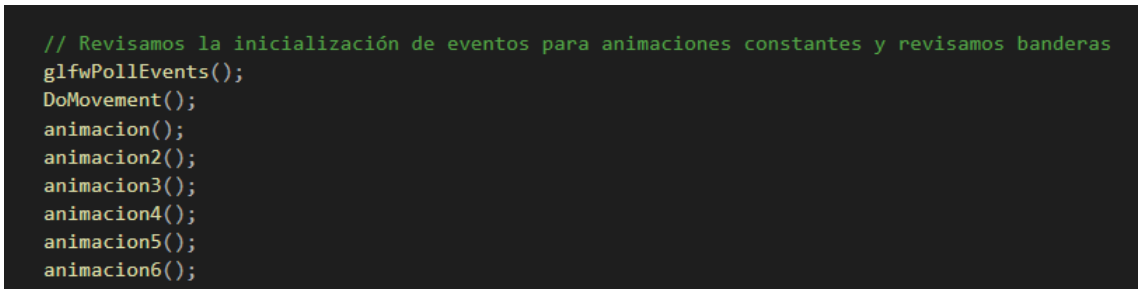


Figura 74: Animation Function Initialize

- The lighting shader is initialize in the first part, we establish his properties (specular,diffuse, directional and ambiental, so we set the axis x,y,z in according to input of values and read as a color, in the next lines, set the position of pointlight and spotlight, that it will be use inside facade, and initiliaze his properties.

Figura 75: lighting Function Initialize

```
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].position"), pointLightPositions[2].x, pointLightPositions[2].y, pointLightPositions[2].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].ambient"), 0.0f, 1.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].diffuse"), 1.0f, 1.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].specular"), 1.0f, 1.0f, 0.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].linear"), 0.7f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].quadratic"), 1.8f);

glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].position"), pointLightPositions[3].x, pointLightPositions[3].y, pointLightPositions[3].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].ambient"), 0.0f, 0.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].diffuse"), 1.0f, 1.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].specular"), 1.0f, 1.0f, 0.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].linear"), 0.7f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].quadratic"), 1.8f);
```

```
// Iniciaización de posición de spotlight, usado como animación de encedido y apagado de TV
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[0].position"), 1.5f, 0.82f, 1.18f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[0].direction"), -5.0f, 0.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[0].ambient"), anim1, anim1, anim1);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[0].diffuse"), anim1, anim1, anim1);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[0].specular"), anim1, anim1, anim1);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[0].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[0].linear"), 0.35f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[0].quadratic"), 0.44f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[0].cutOff"), glm::cos(glm::radians(12.5f)));
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[0].outerCutOff"), glm::cos(glm::radians(15.0f)));

glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[1].position"), 10.0f, 3.0f, -15.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[1].direction"), 0.0f, -1.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[1].ambient"), 0.0f, 0.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[1].diffuse"), 1.0f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[1].specular"), 1.0f, 1.0f, 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[1].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[1].linear"), 0.35f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[1].quadratic"), 0.44f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[1].cutOff"), glm::cos(glm::radians(12.5f)));
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[1].outerCutOff"), glm::cos(glm::radians(15.0f)));
```

Figura 76: lighting Function Initialize

```
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[2].position"), 25.0f, 3.0f, -15.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[2].direction"), 0.0f, -1.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[2].ambient"), 1.0f, 0.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[2].diffuse"), 1.0f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[2].specular"), 1.0f, 1.0f, 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[2].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[2].linear"), 0.35f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[2].quadratic"), 0.44f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[2].cutOff"), glm::cos(glm::radians(12.5f)));
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[2].outerCutOff"), glm::cos(glm::radians(15.0f)));

glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[3].position"), 40.0f, 3.0f, -15.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[3].direction"), 0.0f, -1.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[3].ambient"), 0.0f, 1.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[3].diffuse"), 1.0f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight[3].specular"), 1.0f, 1.0f, 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[3].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[3].linear"), 0.35f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[3].quadratic"), 0.44f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[3].cutOff"), glm::cos(glm::radians(12.5f)));
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight[3].outerCutOff"), glm::cos(glm::radians(15.0f)));

// Iniciamos la propiedad de brillo sobre material
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
```

Figura 77: lighting Function Initialize

- Next, the previously established models are loaded for the drawing and assignment to the previously started model matrix, likewise at each end of the line of the dinosaur body part, it is sent to draw with a parameter to receive the lighting shader.

```
//Carga de modelo de personaje de dinosaurio
view = camera.GetViewMatrix();//establecemos la vista
glm::mat4 model(1);//Declaración de model
model = glm::translate(model, glm::vec3(0.0f, 1.2f, 8.0f));//Transformación de traslación a posición inicial
model = glm::rotate(model, glm::radians(-rotCuerpo), glm::vec3(0.0f, 0.0f, 1.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Enviamos el modelo a la matriz
DinosaurioCuerpo.Draw(lightingShader);//Dibujamos el modelo con parametro recibido de shader de iluminación


view = camera.GetViewMatrix();//Establecemos vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.0f, 1.2f, 8.0f));//Transformación de traslación a posición inicial
model = glm::rotate(model, glm::radians(-rotPies), glm::vec3(0.0f, 0.0f, 1.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos matriz de modelo
DinosaurioPiesA.Draw(lightingShader);//Dibujamos el modelo  con parametro recibido de shader de iluminación

view = camera.GetViewMatrix();//establecemos la vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.0f, 1.2f, 8.0f));//Transformación de traslación a posición inicial
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Enviamos el modelo a la matriz
DinosaurioPiesD.Draw(lightingShader);//Dibujamos el modelo con parametro recibido de shader de iluminación

glBindVertexArray(0);//Finalizamos vertex array
```

```
view = camera.GetViewMatrix();//establecemos la vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.0f, 1.0f, -7.0f));//Transformación de traslación a posición inicial
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Enviamos el modelo a la matriz
Carro.Draw(lightingShader);//Dibujamos el modelo con parametro recibido de shader de iluminación
glBindVertexArray(0);//Finalizamos vertex array
```

Figura 78: Drawing Models

- Models

```
glBindVertexArray(VAO2);//Cargamos los valores a una matriz diferente

glm::mat4 modelTor = glm::mat4(1.0f);//Inicializamos el modelo
modelTor = glm::translate(glm::mat4(1.0f), glm::vec3(1.0, 0.2f, -8.0f));//Se realiza translación del modelo de toroide
modelTor = glm::rotate(modelTor, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 1.0));//Se realiza rotación del modelo de toroide
modelTor = glm::rotate(modelTor, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0));//Se realiza rotación del modelo de toroide
modelTor = glm::scale(modelTor, glm::vec3(0.2f, 0.2f, 0.2f));//Se realiza escalado del modelo de toroide
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(modelTor));//Se envia modelo de toroide en matriz

glDrawElements(GL_TRIANGLES, 1000, GL_UNSIGNED_INT, 0);//Se dibujan los elementos de la matriz

modelTor = glm::translate(glm::mat4(1.0f), glm::vec3(-1.0, 0.2f, -8.0f));//Se realiza translación del modelo de toroide
modelTor = glm::rotate(modelTor, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 1.0));//Se realiza rotación del modelo de toroide
modelTor = glm::rotate(modelTor, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0));//Se realiza rotación del modelo de toroide
modelTor = glm::scale(modelTor, glm::vec3(0.2f, 0.2f, 0.2f));//Se realiza escalado del modelo de toroide
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(modelTor));//Se envia modelo de toroide en matriz

glDrawElements(GL_TRIANGLES, 1000, GL_UNSIGNED_INT, 0);//Se dibujan los elementos de la matriz
```

```
modelTor = glm::translate(glm::mat4(1.0f), glm::vec3(-1.0, 0.2f, -6.0f));//Se realiza translación del modelo de toroide
modelTor = glm::rotate(modelTor, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 1.0));//Se realiza rotación del modelo de toroide
modelTor = glm::rotate(modelTor, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0));//Se realiza rotación del modelo de toroide
modelTor = glm::scale(modelTor, glm::vec3(0.2f, 0.2f, 0.2f));//Se realiza escalado del modelo de toroide
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(modelTor));//Se envia modelo de toroide en matriz

glDrawElements(GL_TRIANGLES, 1000, GL_UNSIGNED_INT, 0);//Se dibujan los elementos de la matriz

modelTor = glm::translate(glm::mat4(1.0f), glm::vec3(1.0, 0.2f, -6.0f));//Se realiza translación del modelo de toroide
modelTor = glm::rotate(modelTor, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 1.0));//Se realiza rotación del modelo de toroide
modelTor = glm::rotate(modelTor, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0));//Se realiza rotación del modelo de toroide
modelTor = glm::scale(modelTor, glm::vec3(0.2f, 0.2f, 0.2f));//Se realiza escalado del modelo de toroide
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(modelTor));//Se envia modelo de toroide en matriz

glDrawElements(GL_TRIANGLES, 1000, GL_UNSIGNED_INT, 0);//Se dibujan los elementos de la matriz



glBindVertexArray(0);
```

Figura 79: Drawing Models

- Models

```
//Carga de modelo del piso
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos a la matriz el modelo
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);//Ingresamos transparencia a objetos
Piso.Draw(lightingShader);//Dibujamos

//Envíamos Modelo de Fachada a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de fachada
Fachada.Draw(lightingShader);
```

```
//Envíamos Modelo de Planta1 a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Planta1
Planta1.Draw(lightingShader);
//Envíamos Modelo de Planta2 a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Planta1
Planta2.Draw(lightingShader);
//Envíamos Modelo de Planta3 a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Planta3
Planta3.Draw(lightingShader);
//Envíamos Modelo de Planta4 a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Planta4
Planta4.Draw(lightingShader);
//Envíamos Modelo de Planta5 a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Planta5
Planta5.Draw(lightingShader);
//Envíamos Modelo de Planta6 a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Planta6
Planta6.Draw(lightingShader);
//Envíamos Modelo de Sillon1 a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Sillon1
Sillon1.Draw(lightingShader);
//Envíamos Modelo de Sillon2 a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
```

Figura 80: Drawing Models

- Models

```
Sillon2.Draw(lightingShader);
//Envíamos Modelo de Cojines a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Cojines
Cojines.Draw(lightingShader);
//Envíamos Modelo de Mesa a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Mesa
Mesa.Draw(lightingShader);
//Envíamos Modelo de Mesa2 a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Mesa2
Mesa2.Draw(lightingShader);
//Envíamos Modelo de TV a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de TV
TV.Draw(lightingShader);
//Envíamos Modelo de Tapete a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Tapete
Tapete.Draw(lightingShader);
//Envíamos Modelo de Libros a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Libros
Libros.Draw(lightingShader);
```

```
//Enviamos el color alpha
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0, 1.0, 1.0, 1.0);
//Finalizamos vertex array
glBindVertexArray(0);
```

Figura 81: Drawing Models

- Models

```
//Carga de Modelo Pedro Picapiedra
view = camera.GetViewMatrix();//establecemos la vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.0f,0.5f,1.5f));//Transformación de traslación a posición inicial
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Envíamos el modelo a la matriz
CuerpoP.Draw(lightingShader);

view = camera.GetViewMatrix();//Establecemos vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.0f, 0.5f, 1.5f));//Transformación de traslación a posición inicial
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Envíamos el modelo a la matriz
CabezaP.Draw(lightingShader);

view = camera.GetViewMatrix();//Establecemos vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.0f, 0.5f, 1.5f));//Transformación de traslación a posición inicial
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Envíamos el modelo a la matriz
PiesP.Draw(lightingShader);//Dibujamos el modelo  con parametro recibido de shader de iluminación

view = camera.GetViewMatrix();//Establecemos vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.0f, 0.5f, 1.5f));//Transformación de traslación a posición inicial
//model = glm::translate(model, glm::vec3(posX2, posY2, posZ2));//Transformación de traslación a posición inicial
model = glm::rotate(model, glm::radians(-rotBrazoDer), glm::vec3(0.0f, 0.0f, 1.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Envíamos el modelo a la matriz
BrazoIzq.Draw(lightingShader);//Dibujamos el modelo  con parametro recibido de shader de iluminación
```

```
view = camera.GetViewMatrix();//Establecemos vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.0f, 0.5f, 1.5f));//Transformación de traslación a posición inicial
model = glm::rotate(model, glm::radians(-rotBrazoDer), glm::vec3(0.0f, 0.0f, 1.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Envíamos el modelo a la matriz
BrazoDer.Draw(lightingShader);//Dibujamos el modelo  con parametro recibido de shader de iluminación



//Finalizamos vertex array
glBindVertexArray(0);
```

Figura 82: Drawing Models

- Models

```
//Carga de Modelo Pedro Picapiedra
view = camera.GetViewMatrix();//establecemos la vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.0f,0.5f,1.5f));//Transformación de traslación a posición inicial
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Envíamos el modelo a la matriz
CuerpoP.Draw(lightingShader);

view = camera.GetViewMatrix();//Establecemos vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.0f, 0.5f, 1.5f));//Transformación de traslación a posición inicial
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Envíamos el modelo a la matriz
CabezaP.Draw(lightingShader);

view = camera.GetViewMatrix();//Establecemos vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.0f, 0.5f, 1.5f));//Transformación de traslación a posición inicial
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Envíamos el modelo a la matriz
PiesP.Draw(lightingShader);//Dibujamos el modelo  con parametro recibido de shader de iluminación

view = camera.GetViewMatrix();//Establecemos vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.0f, 0.5f, 1.5f));//Transformación de traslación a posición inicial
//model = glm::translate(model, glm::vec3(posX2, posY2, posZ2));//Transformación de traslación a posición inicial
model = glm::rotate(model, glm::radians(-rotBrazoDer), glm::vec3(0.0f, 0.0f, 1.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Envíamos el modelo a la matriz
BrazoIzq.Draw(lightingShader);//Dibujamos el modelo  con parametro recibido de shader de iluminación
```

```
view = camera.GetViewMatrix();//Establecemos vista
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.0f, 0.5f, 1.5f));//Transformación de traslación a posición inicial
model = glm::rotate(model, glm::radians(-rotBrazoDer), glm::vec3(0.0f, 0.0f, 1.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model)); //Envíamos el modelo a la matriz
BrazoDer.Draw(lightingShader);//Dibujamos el modelo  con parametro recibido de shader de iluminación



//Finalizamos vertex array
glBindVertexArray(0);
```

Figura 83: Drawing Models

- Models

```
//Shaggy
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, PosIni3 + glm::vec3(movKitX2,movKitY2, movKitZ2));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(rotKit2), glm::vec3(0.0f, 1.0f, 0.0));//asignamos transofmración de rotación al modelo, usado para cambio de dirección
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos a la matriz el modelo
ShaCuerpo.Draw(lightingShader);//Dibujamos el modelo del dinosaurio

view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, PosIni3 + glm::vec3(movKitX2, movKitY2, movKitZ2));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(rotKit2), glm::vec3(0.0f, 1.0f, 0.0));//asignamos transofmración de rotación al modelo, usado para cambio de dirección
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos a la matriz el modelo
ShaCabeza.Draw(lightingShader);//Dibujamos el modelo del dinosaurio

view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, PosIni3 + glm::vec3(movKitX2, movKitY2, movKitZ2));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(rotKit2), glm::vec3(0.0f, 1.0f, 0.0));//asignamos transofmración de rotación al modelo, usado para cambio de dirección
model = glm::rotate(model, glm::radians(-rot2), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos a la matriz el modelo
ShaBrazoDer.Draw(lightingShader);//Dibujamos el modelo del dinosaurio
```

```
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, PosIni3 + glm::vec3(movKitX2, movKitY2, movKitZ2));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(rotKit2), glm::vec3(0.0f, 1.0f, 0.0));//asignamos transofmración de rotación al modelo, usado para cambio de dirección
model = glm::rotate(model, glm::radians(-rot3), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos a la matriz el modelo
ShaBrazoIzq.Draw(lightingShader);//Dibujamos el modelo del dinosaurio

view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, PosIni3 + glm::vec3(movKitX2, movKitY2, movKitZ2));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(rotKit2), glm::vec3(0.0f, 1.0f, 0.0));//asignamos transofmración de rotación al modelo, usado para cambio de dirección
model = glm::rotate(model, glm::radians(-rot3), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos a la matriz el modelo
ShaPiernaDer.Draw(lightingShader);//Dibujamos el modelo del dinosaurio

view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, PosIni3 + glm::vec3(movKitX2, movKitY2, movKitZ2));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(rotKit2), glm::vec3(0.0f, 1.0f, 0.0));//asignamos transofmración de rotación al modelo, usado para cambio de dirección
model = glm::rotate(model, glm::radians(-rot2), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos a la matriz el modelo
ShäPiernaIzq.Draw(lightingShader);//Dibujamos el modelo del dinosaurio
```

Figura 84: Drawing Models

- Models

```
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, PosIni + glm::vec3(movKitX, movKitY, movKitZ));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f, 1.0f, 0.0));//asignamos transofmración de rotación al modelo, usado para cambio de dirección
model = glm::scale(model, glm::vec3(3.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos a la matriz el modelo
Dino.Draw(lightingShader);//Dibujamos el modelo del dinosaurio

view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, PosIni + glm::vec3(movKitX, movKitY, movKitZ));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f, 1.0f, 0.0));//asignamos transofmración de rotación al modelo, usado para cambio de dirección
model = glm::rotate(model, glm::radians(-rot), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::scale(model, glm::vec3(3.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos a la matriz el modelo
AlaD.Draw(lightingShader);//Dibujamos el modelo del dinosaurio

view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, PosIni + glm::vec3(movKitX, movKitY, movKitZ));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f, 1.0f, 0.0));//asignamos transofmración de rotación al modelo, usado para cambio de dirección
model = glm::rotate(model, glm::radians(rot), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::scale(model, glm::vec3(3.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos a la matriz el modelo
AlaI.Draw(lightingShader);//Dibujamos el modelo del dinosaurio


glBindVertexArray(0);//Finalizamos vertex array
```

```
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, glm::vec3(10.0, 0.0, 15.0));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Fachada.Draw(lightingShader);

view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, glm::vec3(10.0, 0.0, 15.0));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Sillon2.Draw(lightingShader);

view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, glm::vec3(10.0, 0.0, 15.0));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
TV.Draw(lightingShader);

view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, glm::vec3(10.0, 0.0, 15.0));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Mesa.Draw(lightingShader);
```

Figura 85: Drawing Models

- Models

```
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, glm::vec3(12.0, -0.4, 15.3));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(0.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Pablo.Draw(lightingShader);
```

```
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, glm::vec3(27.0, -0.4, 15.3));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(0.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Betty.Draw(lightingShader);
```

Figura 86: Drawing Models

- Models

```
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, glm::vec3(42.0, -0.4, 15.3));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(0.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Vilma.Draw(lightingShader);
```

```
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, glm::vec3(8.0, -0.4, -15.3));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Rajuela.Draw(lightingShader);
```

Figura 87: Drawing Models

- Models

```
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, glm::vec3(23.0, -0.4, -15.3));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Cavernicola.Draw(lightingShader);
```

```
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, glm::vec3(38.0, -0.4, -15.3));//asignamos transformación de traslación al modelo, usado para animación
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));//Transformación de rotación por variable
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Trajeado.Draw(lightingShader);
```

Figura 88: Drawing Models

- Models

```
//Creación  de modelo de iluminación punto
model = glm::mat4(1);
//Transformación de traslación en iluminación punto
model = glm::translate(model, pointLightPositions[0]);
//Transformación de escalado en iluminación punto
model = glm::scale(model, glm::vec3(0.2f));
//Enviar modelo a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glBindVertexArray(VAO);
//Dibujado de cubo de point light
glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);

//Creación de modelo de iluminación punto
model = glm::mat4(1);
//Transformación de traslación en iluminación punto
model = glm::translate(model, pointLightPositions[1]);
//Transformación de escalado en iluminación punto
model = glm::scale(model, glm::vec3(0.2f));
//Enviar modelo a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glBindVertexArray(VAO);
//Dibujado de cubo de point light
glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);
```

```
//Creación de modelo de iluminación punto
model = glm::mat4(1);
//Transformación de traslación en iluminación punto
model = glm::translate(model, pointLightPositions[2]);
//Transformación de escalado en iluminación punto
model = glm::scale(model, glm::vec3(0.2f));
//Enviar modelo a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glBindVertexArray(VAO);
//Dibujado de cubo de point light
glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);

//Creación de modelo de iluminación punto
model = glm::mat4(1);
//Transformación de traslación en iluminación punto
model = glm::translate(model, pointLightPositions[3]);
//Transformación de escalado en iluminación punto
model = glm::scale(model, glm::vec3(0.2f));
//Enviar modelo a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glBindVertexArray(VAO);
//Dibujado de cubo de point light
glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);
```

Figura 89: Dibujado de Modelos

- Models

```
model = glm::mat4(1);
//Transformación de traslación en iluminación punto
model = glm::translate(model, pointLightPositions[4]);
//Transformación de escalado en iluminación punto
model = glm::scale(model, glm::vec3(0.2f, 0.5f, 0.7f));
//Enviar modelo a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glBindVertexArray(VAO);
//Dibujado de cubo de point light
glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);

model = glm::mat4(1);
//Transformación de traslación en iluminación punto
model = glm::translate(model, glm::vec3(10.0f, 3.0f, -15.0f));
//Transformación de escalado en iluminación punto
model = glm::scale(model, glm::vec3(0.2f));
//Enviar modelo a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glBindVertexArray(VAO);
//Dibujado de cubo de point light
glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);
```

```
model = glm::mat4(1);
//Transformación de traslación en iluminación punto
model = glm::translate(model, glm::vec3(25.0f, 3.0f, -15.0f));
//Transformación de escalado en iluminación punto
model = glm::scale(model, glm::vec3(0.2f));
//Enviar modelo a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glBindVertexArray(VAO);
//Dibujado de cubo de point light
glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);

model = glm::mat4(1);
//Transformación de traslación en iluminación punto
model = glm::translate(model, glm::vec3(40.0f, 3.0f, -15.0f));
//Transformación de escalado en iluminación punto
model = glm::scale(model, glm::vec3(0.2f));
//Enviar modelo a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glBindVertexArray(VAO);
//Dibujado de cubo de point light
glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);
// Dibujamos al ultimo el skybox
glDepthFunc(GL_LEQUAL);
//Iniciamos shader de sky box para su uso
SkyBoxshader.Use();
//Movemos el componente de vista de la matriz
view = glm::mat4(glm::mat3(camera.GetViewMatrix()));
//Enviamos la vista y proyección a la matriz
glUniformMatrix4fv(glGetUniformLocation(SkyBoxshader.Program, "view"), 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(glGetUniformLocation(SkyBoxshader.Program, "projection"), 1, GL_FALSE, glm::value_ptr(projection));
```

Figura 90: Drawing Models

- Declaration of sound functions, in the first is the soundtrack intro, second ambient track, this goal track, animation sound.

```
//Funcion de Insertado de sonido de intro
void Sonido()
{
    PlaySound(TEXT("picapiedra.wav"), NULL, SND_SYNC );
}

//Funcion de Insertado de sonido de ambientación
void Sonido2()
{

    PlaySound(TEXT("ambiente.wav"), NULL, SND_LOOP | SND_ASYNC);

}

//Función de insertado de sonido de animación gol
void Sonido3()
{

    PlaySound(TEXT("gol.wav"), NULL, SND_ASYNC);

}
```

Figura 91: Sound Functions

- Function declaration of keys in which they will be activated for movement of the camera in which it will be executed when interacted, where the dinosaur movement keys are entered to drink water and the movement of Pedro Flintstone's arms, defined animations, movement of the dragon etc.

```
//Control de iniciado de animación de movimiento de dragon
if (keys[GLFW_KEY_I])
{
    circuito = true;
    anim = true;
}

if (keys[GLFW_KEY_N])
{
    circuito2 = true;
    anim3 = true;
}


//Control de pausado de movimiento de dragon
if (keys[GLFW_KEY_O])
{
    circuito = false;
}
```

```
/// Animación Dinosaurio
void animacion()
{
    //Bandera de activación
    if (circuito)
    {
        //Bandera de activación recorrido 1
        if (recorrido1)
        {
            //Desplazamiento en X
            movKitX += 0.1f;
            movKitY -= 0.02f;
            //Bandera de limite de 20 unidades
            if (movKitX > 40 )
            {
                //Cambio de recorrido
                recorrido1 = false;
                recorrido2 = true;
            }
        }
        //Bandera de activación recorrido 2
        if (recorrido2)
        {
            //Cambio de dirección
            rotKit += 1;
            //Bandera de limite de 90 grados
            if (rotKit > 0)
            {
                //Cambio de recorrido
                recorrido2 = false;
                recorrido3 = true;
            }
```

Figura 92: Animation Function

- Animation Function

```
}
//Bandera de activación recorrido 3
if (recorrido3)
{
    //Desplazamiento en Z negativo
    movKitZ -= 0.1f;
    movKitY += 0.03f;
    //Bandera de limite de 20 unidades menos
    if (movKitZ < -20)
    {
        //Cambio de recorrido
        recorrido3 = false;
        recorrido4 = true;
    }
}
//Bandera de activación recorrido 4
if (recorrido4)
{
    //Cambio de dirección
    rotKit += 1;
    //Bandera de limite en 180 grados
    if (rotKit > 90)
    {
        //Cambio de recorrido
        recorrido4 = false;
        recorrido5 = true;
    }
}
//Bandera de activación recorrido 5
if (recorrido5)
{
    //Desplazamiento en X negativo
    movKitX -= 0.1f;
    movKitY -= 0.015f;
    //Bandera de limite de 20 unidades menos
    if (movKitX < -20)
    {
        //Cambio de recorrido
        recorrido5 = false;
        recorrido6 = true;
    }
}
//Bandera de activación recorrido 6
if (recorrido6)
{
    //Cambio de dirección
    rotKit += 1;
    //Bandera de limite en 270 grados
    if (rotKit > 180)
    {
        //Cambio de recorrido
        recorrido6 = false;
        recorrido7 = true;
    }
}
```

Figura 93: Animation Function

- Animation Function

```
//Bandera de activación recorrido 7
if (recorrido7)
{
    //Desplazamiento en Z positivo
    movKitZ += 0.1f;
    movKitY += 0.03f;
    //Bandera de limite de 0 unidades
    if (movKitZ > 0)
    {
        //Cambio de recorrido
        recorrido7 = false;
        recorrido8 = true;
    }
}
//Bandera de activación recorrido 8
if (recorrido8)
{
    //Cambio de dirección
    rotKit += 1;
    //Bandera de limite en 360 grados
    if (rotKit > 270)
    {
        //Cambio de recorrido
        recorrido8 = false;
        recorrido9 = true;
    }
}
```

```
//Bandera de activación recorrido 9
if (recorrido9)
{
    //Desplazamiento en X positivo
    movKitX += 0.1f;
    movKitY += 0.03f;
    //Bandera de limite de 0 unidades
    if (movKitX > 0)
    {
        rotKit = -90;
        //Cambio de recorrido
        recorrido9 = false;
        recorrido1 = true;
    }
}
}
```

Figura 94: Animation Function

- Animation Function

```
//Bandera de play
if (play)
{
    //Bandera de pasos actaules mayor a pasos maximos
    if (i_curr_steps >= i_max_steps)
    {
        //Incremento de indice de actual
        playIndex++;
        //Bandera indice actual al indice almacenado no sea mayor, termina animación
        if (playIndex > FrameIndex - 2)
        {
            printf("termina anim\n");
            //Inicializamos a valores iniciales
            playIndex = 0;
            play = false;
        }
        else
        {
            //Reiniciamos contador
            i_curr_steps = 0;
            //interpolamos para realizar
            interpolation();
        }
    }
    else
    {
        //Recorrido de posición
        posX += KeyFrame[playIndex].incX;
        posY += KeyFrame[playIndex].incY;
        posZ += KeyFrame[playIndex].incZ;

        rotCuerpo += KeyFrame[playIndex].rotInc;
        rotPies += KeyFrame[playIndex].rotPies;

        i_curr_steps++;
    }
```

```
//Bandera de play
if (play2)
{
    //Bandera de pasos actaules mayor a pasos maximos
    if (i_curr_steps2 >= i_max_steps2)
    {
        //Incremento de indice de actual
        playIndex2++;
        //Bandera de posición 2
        if (playIndex2 == 1)
        {
            //Función de sonido 3, sonido de gol
            Sonido3();
        }
        //Bandera indice actual al indice almacenado no sea mayor, termina animación
        if (playIndex2 > FrameIndex2 - 2)
        {
            printf("termina anim\n");
            //Función de sonido 2, sonido de ambiente
            Sonido2();
            //Inicializamos a valores iniciales
            playIndex2 = 0;
            play2 = false;
        }
        else
        {
            //Reiniciamos contador
            i_curr_steps2 = 0;
            //interpolamos para realizar
            interpolation2();
        }
    }
```

Figura 95: Animation Function

- Animation Function

```
        else
        {
            //Reiniciamos contador
            i_curr_steps2 = 0;
            //interpolamos para realizar
            interpolation2();
        }
    }
    else
    {
        //Recorrido de posición
        posX2 += KeyFrame2[playIndex2].incX2;
        posY2 += KeyFrame2[playIndex2].incY2;
        posZ2 += KeyFrame2[playIndex2].incZ2;

        rotBrazoDer += KeyFrame2[playIndex2].rotInc2;

        i_curr_steps2++;
    }

}
```

```
if (anim3)
{
    if (rot2 <= 45.0f)
    {
        rot2 = rot2 + 0.5f;
        rot3 = rot3 - 0.5f;
    }
    else
    {
        anim3 = false;
        anim4 = true;
    }
}
if (anim4)
{
    if (rot2 >= -45.0f)
    {
        rot2 = rot2 - 0.5f;
        rot3 = rot3 + 0.5f;
    }
    else
    {
        anim4 = false;
        anim3 = true;
    }
}
```

Figura 96: Animation Function

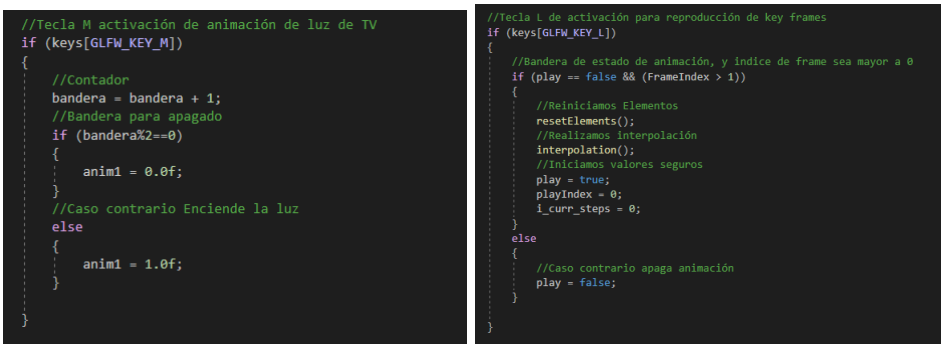
- Animation Function

```
//Bandera de activación
if (circuito2)
{
    //Bandera de activación recorrido 1
    if (recorrido10)
    {
        //Desplazamiento en X
        movKitX2 -= 0.05f;
        //Bandera de limite de 20 unidades
        if (movKitX2 < -8)
        {
            //Cambio de recorrido
            recorrido10 = false;
            recorrido11 = true;
        }
    }
    //Bandera de activación recorrido 2
    if (recorrido11)
    {
        //Cambio de dirección
        rotKit2 = 90;
        movKitX2 += 0.05f;
        //Bandera de limite de 90 grados
        if (movKitX2 > 8)
        {
            rotKit2 = -90;
            //Cambio de recorrido
            recorrido11 = false;
            recorrido10 = true;
        }
    }
}
```

```
if (anim)
{
    if (rot <= 45.0f)
    {
        rot = rot + 0.5f;
    }
    else
    {
        anim = false;
        anim2 = true;
    }
}
if (anim2)
{
    if (rot >= -45.0f)
    {
        rot = rot - 0.5f;
    }
    else
    {
        anim2 = false;
        anim = true;
    }
}
```

Figura 97: Animation Function

- Animation Function



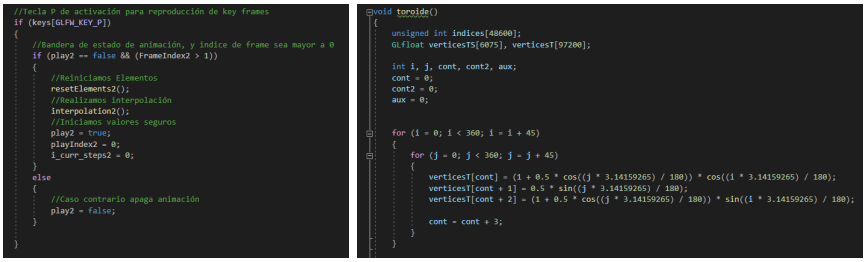
```
//Tecla M activación de animación de luz de TV
if (keys[GLFW_KEY_M])
{
    //Contador
    bandera = bandera + 1;
    //Bandera para apagado
    if (bandera%2==0)
    {
        anim1 = 0.0f;
    }
    //Caso contrario Enciende la luz
    else
    {
        anim1 = 1.0f;
    }

}
```

```
//Tecla L de activación para reproducción de key frames
if (keys[GLFW_KEY_L])
{
    //Bandera de estado de animación, y indice de frame sea mayor a 0
    if (play == false && (FrameIndex > 1))
    {
        //Reiniciamos Elementos
        resetElements();
        //Realizamos interpolación
        interpolation();
        //Iniciamos valores seguros
        play = true;
        playIndex = 0;
        i_curr_steps = 0;
    }
    else
    {
        //Caso contrario apaga animación
        play = false;
    }
}
```

Figura 98: Animation Function

- Animation Function



```
//Tecla P de activación para reproducción de key frames
if (keys[GLFW_KEY_P])
{
    //Bandera de estado de animación, y indice de frame sea mayor a 0
    if (play2 == false && (FrameIndex2 > 1))
    {
        //Reiniciamos Elementos
        resetElements2();
        //Realizamos interpolación
        interpolation2();
        //Iniciamos valores seguros
        play2 = true;
        playIndex2 = 0;
        i_curr_steps2 = 0;
    }
    else
    {
        //Caso contrario apaga animación
        play2 = false;
    }
}
```

```
void toroide()
{
    unsigned int indices[48600];
    GLfloat verticesTS[6075], verticesT[97200];

    int i, j, cont, cont2, aux;
    cont = 0;
    cont2 = 0;
    aux = 0;


    for (i = 0; i < 360; i = i + 45)
    {
        for (j = 0; j < 360; j = j + 45)
        {
            verticesT[cont] = (1 + 0.5 * cos((j * 3.14159265) / 180)) * cos((i * 3.14159265) / 180);
            verticesT[cont + 1] = 0.5 * sin((j * 3.14159265) / 180);
            verticesT[cont + 2] = (1 + 0.5 * cos((j * 3.14159265) / 180)) * sin((i * 3.14159265) / 180);

            cont = cont + 3;
        }
    }
```

Figura 99: Animation Function

- Toroid Function

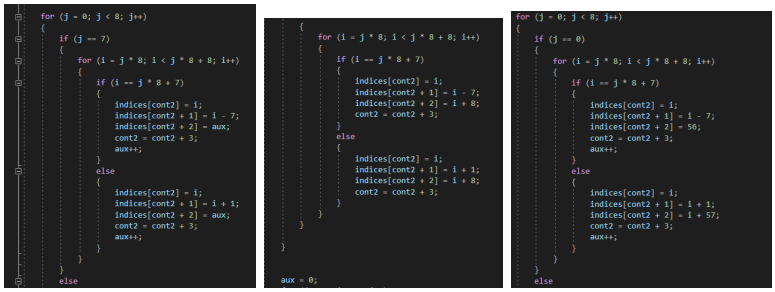


Figura 100: Toroid Function

- Toroid Function

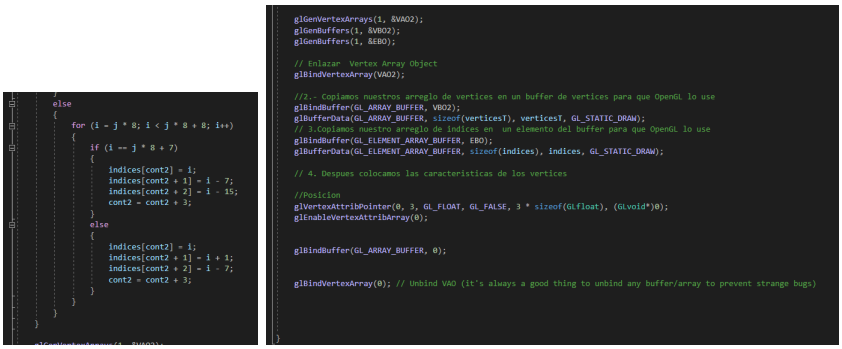


Figura 101: Toroid Function

2.6. Conclusions

- This project was incredible because the form of graph construction i was not know, in this case i was known from the moment creative, creation and production, through which a grapich creation, so there were few method that i saw, the basics at moment i know now, and when investigate from the previous and documentation what is the function of something, i learn too much in the class, for this finish project i learn each knowledge in the course, so is implemented for this completed and for this completed project i took a lot of knowledge in terms that used programs that a i did not know o knew how to use, and this part i feel that is was a project realted in real delivery cause has delivery dates.