Manual Técnico

Proyecto Final

315058765

Computación Gráfica e Interacción Humano Computadora

Facultad de Ingeniería

U.N.A.M

11 de Mayo del 2022

Grupo: 09

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	En progreso
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

Figura 1: Diagram de Gantt Parte 1

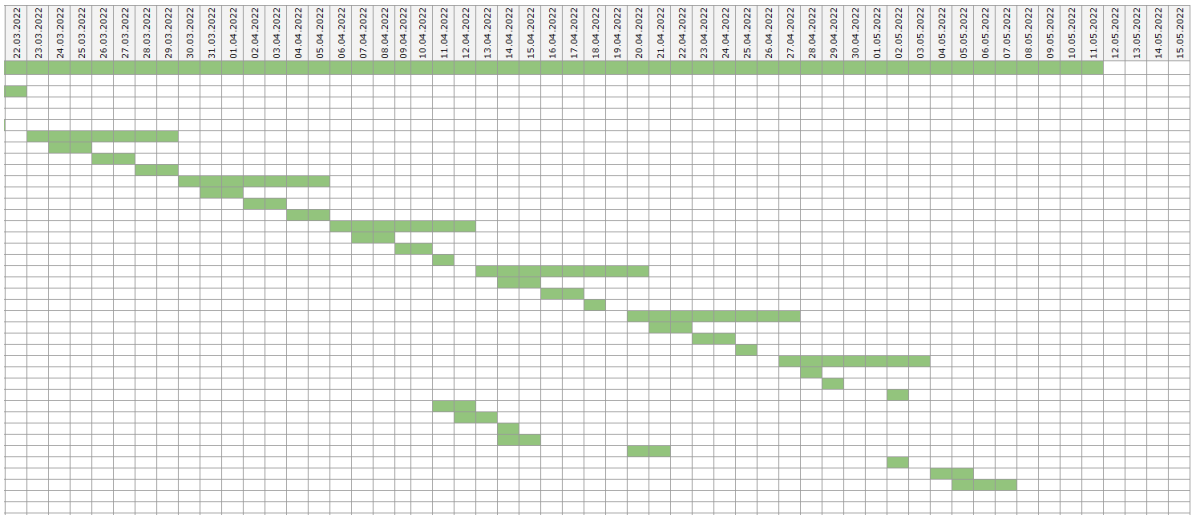

Figura 2: Diagram de Gantt Parte 2

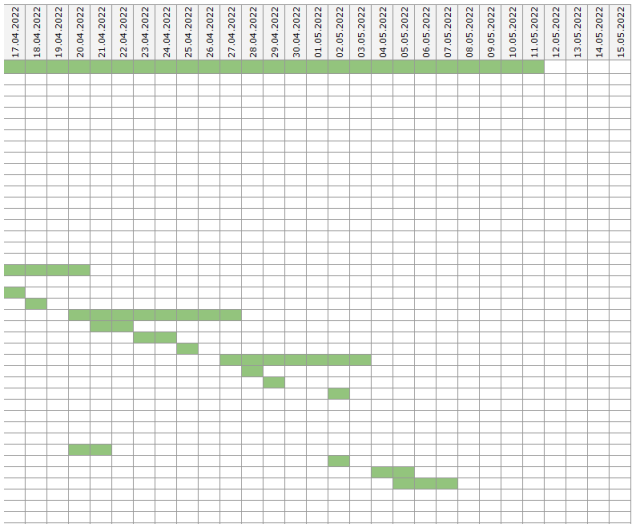

Figura 3: Diagrama de Gantt Parte 3

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 4: 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 5: Declaración de Funciones

- Creación de la variables a usar para la inicialización del tamaño de ventana en que resolución se ejecutara la ventana.

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

Figura 6: Declaración de Resolución

- Declaración para que la pantalla quede a la mitad de la resolución, en el punto medio de la ventana.

```
// Declaración de modelo de camara y su posición
Camera  camera(glm::vec3(4.0f, 1.0f, 0.0f));
GLfloat lastX = WIDTH / 2.0;
GLfloat lastY = HEIGHT / 2.0;
```

Figura 7: D

- Se declara el arreglo para asignación total de las teclas a utilizar, en este, caso se asigna un total de memoria de 1024.

```
//Arreglo para teclas
bool keys[1024];
```

Figura 8: Declaración de Memoria Teclas

- Bandera de activación para el movimiento de ratón, enviando la bandera como positivo.

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

Figura 9: Bandera Mouse

- Atributos del uso de iluminación, teniendo una posición en el cual se inicia (0,0,0).

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

Figura 10: Posición Iluminación

- 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).

```
//Declaración de posición inical de objeto a mover con Key Frames
glm::vec3 PosIni(0.0f, 0.0f, 0.0f);
glm::vec3 PosIni2(0.0f, 0.0f, 0.0f);
```

Figura 11: Posición Inicial Objetos

- 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.

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

Figura 12: Posición Iluminación

- Arreglo de puntos iniciales para la posición de los point lights en el cual será la posición en donde se dibujarán desde un inicio.

```
// Posición inicial del pointlight y cubo
glm::vec3 pointLightPositions[] = {
    glm::vec3(0.0f,3.0f, 0.0f),
    glm::vec3(1.5f, 0.82f, 1.15f)

};
```

Figura 13: Posición Point Lights

- 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,

```
//Variables de movimiento y rotación para animación 3
float movKitX = 0.0;
float movKitZ = 0.0;
float rotKit = 0.0;
```

Figura 14: Variables Animación 2

- Inicialización de la bandera que controlará la animación 3 del dragón

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

Figura 15: Bandera Circuito

- Variables bandera, para realizar el cambio de posición siendo así el recorrido cambia cuando se complete el proceso.

```
/*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;
```

Figura 16: Banderas recorrido

- Inicialización de variables para el calculo de los frames usados, deltatime, será la diferencia entre frame actual y ultimo, lastframe almacenará el fame 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
```

Figura 17: Variables Frames

- 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.

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

Figura 18: Variables Frames para Modelo

- Definición como constante la variable max_frames, la cual será el limite de frames a almacenar

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

Figura 19: Define Max Frames

- 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.

```
//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 20: Limite y Actual

- Definición de estructura frame 1, 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

}FRAME;
```

Figura 21: Estructura Frame 1

- Definición de estructura 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 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 22: Estructura Frame 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;
```

Figura 23: Variables para Key Frame 1

- 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 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 24: Variables para Key Frame 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 25: Arreglo Vertices

- Inicialización de variable de iluminación a vector 3, para asignar transformaciones, posición,etc.

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

Figura 26: Light 1

- Función encargada de almacenar los frames en cada posición del arreglo, asignando desde su posición inicial X,Y,Z a su vez asignando valores de rotar cuerpo.

```
//Función para salvar los frames
void saveFrame(void)
{
    //Imprime el numero de frame almacenado
    printf("frameindex %d\n", FrameIndex);

    //Arreglo de estructura para almacenar la posición en el frame en el elemento de posición
    KeyFrame[FrameIndex].posX = posX;
    KeyFrame[FrameIndex].posY = posY;
    KeyFrame[FrameIndex].posZ = posZ;

    KeyFrame[FrameIndex].rotCuerpo= rotCuerpo;

    FrameIndex++;
}
```

Figura 27: Función SaveFrame

- Función encargada de almacenar los frames en cada posición del arreglo, asignando desde su posición inicial X,Y,Z a su vez asignando valores de rotar cuerpo de objeto dos.

```
//Función para salvar los frames
void saveFrame2(void)
{
    //Arreglo de estructura para almacenar la posición en el frame en el elemento de posición
    printf("frameindex %d\n", FrameIndex2);

    KeyFrame2[FrameIndex2].posX2 = posX2;
    KeyFrame2[FrameIndex2].posY2 = posY2;
    KeyFrame2[FrameIndex2].posZ2 = posZ2;

    KeyFrame2[FrameIndex2].rotBrazoDer = rotBrazoDer;


    FrameIndex2++;
}
```

Figura 28: Función SaveFrame 2

- 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.

```
//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;
}
```

Figura 29: Función ResetElements

- 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.

```
//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 30: Función ResetElements 2

- Función interpolación, realiza un calculo para realizar el incremeneto, dicho incremento definirá la velocidad de reproducción de nuestra animación por key frame, 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

}
```

Figura 31: Función Interpolación

- Función interpolación, realiza un calculo para realizar el incremento, dicho incremento definirá la velocidad de reproducción de nuestra animación por key frame 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 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 32: Función Interpolación 2

- Inicialización de las bibliotecas y funciones para procesamiento gráfico, la siguiente función establece sugerencias para la siguiente llamada a glfwCreateWindow.

```
// Inicialización de  GLFW
glfwInit();
// Seteamos los valores requeridos por GLFW
glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
```

Figura 33: Inicialización de Ejecución

- Se inicializa el crear ventana asignando hacia la variable windows, en la cual recibe las variables de resolución y el título, anteriormente se inicializó las funciones para activar la ventana.

```
// Creamos el objeto de ventana por medio de la funcion de glfw, recibiendo resolución y nombre de ventana.
GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "Picapiedras", nullptr, nullptr);

//Comprobación de creación de venta correctamente.
if (nullptr == window)
{
    std::cout << "Failed to create GLFW window" << std::endl;
    glfwTerminate();

    return EXIT_FAILURE;
}
```

Figura 34: Inicialización de Ejecución

- Se inicializa todas las funciones a realizar y se establecen para recibirlo como métodos de entrada, a su vez se verifica que se haya inicializado todo el contexto en cuestión a GLFW, si es verdadero, se manda a viewport para construcción de ventana.

```
//Cambio de contexto anterior a nuevo
glfwMakeContextCurrent(window);

//Recibimos el tamaño de buffer de ventana
glfwGetFramebufferSize(window, &SCREEN_WIDTH, &SCREEN_HEIGHT);

// Inicializamos las funciones de callback de mouse y teclado
glfwSetKeyCallback(window, KeyCallback);
glfwSetCursorPosCallback(window, MouseCallback);

// Opciones de GLFW
glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);

// Inicializamos para recibir funciones de punteros y extensiones
glewExperimental = GL_TRUE;
// Inicializamos GLEW para los punteros y verificamos que se haya creado correctamente
if (GLEW_OK != glewInit())
{
    std::cout << "Failed to initialize GLEW" << std::endl;
    return EXIT_FAILURE;
}

// Definimos las dimensiones del viewport
glViewport(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT);

glEnable(GL_DEPTH_TEST);
```

Figura 35: Inicialización de Ejecución

- 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.

```
/*
* 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 36: Inicialización de Ejecución

- 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

```
//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");
```

Figura 37: Inicialización de Modelos Pedro Picapiedra

- 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

```
//Modelo de Dinosaurio
Model Dino((char*)"Models/Obj/Models/Dragon/Dragon.obj");
```

Figura 38: Inicialización de Modelos Dragón

- Se manda a llamar el shader de animación, establecido como libreria externa inicalizado en un variable local Anim.

```
//Shader creado para animacion del agua
Shader Anim("Shaders/anim.vs", "Shaders/anim.frag");
```

Figura 39: Inicialización de Shader Animación

- 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.

```
//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
Model DinosaurioPies((char*)"Models/Obj/Models/DinoPerson/Pies/Pies.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");
//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 Libros
Model Libros((char*)"Models/Obj/Models/Libros/Libros.obj");
```

Figura 40: Inicialización de Modelos de Objetos

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

```
KeyFrame[0].rotCuerpo = 0;
KeyFrame[1].rotCuerpo = 45;
KeyFrame[2].rotCuerpo = -1;
FrameIndex = 4;



KeyFrame2[0].rotBrazoDer = 0;
KeyFrame2[1].rotBrazoDer = 20;
KeyFrame2[2].rotBrazoDer = 0;
FrameIndex2 = 4;
```

Figura 41: 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 42: Vertices cubo skybox

- Se inicializa todas las propiedades desde los vertex arrays para la construcción de modelos, seteo de buffer, asignación de memoria para la posición y normal, por último inicializamos el uso del shader de iluminación declarando la luz difusa y especular.

```
// Inicialización de las funciones VAO y VBO
GLuint VBO, VAO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
glBindVertexArray(VAO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

//Establecemos el atributo de posición
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(0);
//Establecemos el atributo para la normal
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 * sizeof(float)));
glEnableVertexAttribArray(1);

// Inicilización de unidad de textura
lightingShader.Use();
glUniform1i(glGetUniformLocation(lightingShader.Program, "diffuse"), 0);
glUniform1i(glGetUniformLocation(lightingShader.Program, "specular"), 1);
```

Figura 43: inicialización de estancias básicas

- 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.

```
//Inicialización del SkyBox
GLuint skyboxVBO, skyboxVAO;
glGenVertexArrays(1, &skyboxVAO);
glGenBuffers(1, &skyboxVBO);
glBindVertexArray(skyboxVAO);
glBindBuffer(GL_ARRAY_BUFFER, skyboxVBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(skyboxVertices), &skyboxVertices, GL_STATIC_DRAW);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);

//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/back.tga");
faces.push_back("SkyBox/front.tga");
```

Figura 44: Textura Skybox

- Se cargan las texturas al cubo y sus caras, a su vez se inicializa la vista en perspectiva hacia el mismo modelo.

```
GLuint cubemapTexture = TextureLoading::LoadCubemap(faces);
//Creación de proyección en perspectiva
glm::mat4 projection = glm::perspective(camera.GetZoom(), (GLfloat)SCREEN_WIDTH / (GLfloat)SCREEN_HEIGHT, 0.1f, 100.0f);
```

Figura 45: Carga textura de 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 46: Invocación de función de Sonidos

- Inicialización del tiempo, con su calculo y asignación en la cual delta es la diferencia de tiempo de frame actual al ultimo, estableciendo el actual como último.

```
// Calculamos el ultimo frame y su diferentecía entre el actual y el ultimo
GLfloat currentFrame = glfwGetTime();
deltaTime = currentFrame - lastFrame;
lastFrame = currentFrame;
```

Figura 47: Inicialización de Tiempo

- 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();
```

Figura 48: Inicialización de Funciones de Animación

- Se inicializa a valores seguros el buffer de color y se establece.

```
// Limpiamos el buffer de ejecución y color
glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Figura 49: Limpiar Buffer Color

- Inicializa el shader de iluminación, a su vez se establece posición inicial a la cámara para su inicio.

```
// Declaramos el shader de iluminación a inicar uso
lightingShader.Use();
GLint viewPosLoc = glGetUniformLocation(lightingShader.Program, "viewPos");
//Iniciamos su posicionamiento
glUniform3f(viewPosLoc, camera.GetPosition().x, camera.GetPosition().y, camera.GetPosition().z);
```

Figura 50: Inicialización de Modelos

- 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);


// Vector de Iluminación de punto
glm::vec3 lightColor;
lightColor.x = abs(sin(glfwGetTime() * Light1.x));
lightColor.y = abs(sin(glfwGetTime() * Light1.y));
lightColor.z = sin(glfwGetTime() * Light1.z);

//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);

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

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

Figura 51: Inicialización de Iluminación

- Se realiza la inicialización de nuevo para enviar otra matriz de modelo, en este caso se enviarán nuevamente la cámara, se inicializa la proyección, modelo y vista y las propiedades del modelo.

```
// Creamos la creación de camara
glm::mat4 view;
view = camera.GetViewMatrix();

// Iniciamos la prespectica de model, view y proyección
GLint modelLoc = glGetUniformLocation(lightingShader.Program, "model");
GLint viewLoc = glGetUniformLocation(lightingShader.Program, "view");
GLint projLoc = glGetUniformLocation(lightingShader.Program, "projection");

// Realizamos la matreiz para la vista y proyección
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));

glBindVertexArray(VAO);
glm::mat4 tmp = glm::mat4(1.0f); //Temp
```

Figura 52: Inicialización de Matriz Model

- 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(posX, posY, posZ));//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)); //Envíamos el modelo a la matriz
DinosaurioCuerpo.Draw(lightingShader);//Dibujamos el modelo con parametro recibido de shader de iluminación

view = camera.GetViewMatrix();//Establecemos vista
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos matriz de modelo
DinosaurioPies.Draw(lightingShader);//Dibujamos el modelo  con parametro recibido de shader de iluminación

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

Figura 53: Dibujado de Dinosaurio

- Se carga ahora el modelo establecido del dragón para a su vez asignar las transformaciones de rotación y traslación, a la espera de recibir un valor cambiante que sería la animación 2, al final se envía a dibujar con un parámetro a recibir el shader de iluminación.

```
//Caragamos modelo del dragon volador
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, PosIni + glm::vec3(movKitX, 0, 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
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos a la matriz el modelo
Dino.Draw(lightingShader);//Dibujamos el modelo del dinosaurio
glBindVertexArray(0);//Finalizamos vertex array
```

Figura 54: Dibujado de Dragón

- Se carga el modelo del piso para su dibujado en este caso, se activa la transparencia del objeto para que no se vuelva transparente el piso y difuminado.

```
//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
```

Figura 55: Dibujado de Piso

- Se carga el modelo de todos los objetos dentro de la fachada: (Libros,Plantas,Mesa,Sillón,Cojines,TV,etc) para su dibujado dentro de la fachada, con posición inicial ya asignada.

```
//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));
//Dibujamos el modelo de Sillon2
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);
```

Figura 56: Dibujado de Objetos

- Inicialización del color alpha.

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

Figura 57: Color Alpha

- Inicialización de nuevo modelo para asignación de modelo de objetos creados para el dibujado de Pedro Picapiedra en el cual se asignará movimientos a los brazos para su movimiento por medio de key frames.

```
//Carga de Modelo Pedro Picapiedra
//Iniciamos Camara de vista
view = camera.GetViewMatrix();
//Enviamos modelo Cabeza de Pedro a la matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Cabeza
CabezaP.Draw(lightingShader);

//Iniciamos Camara de vista
view = camera.GetViewMatrix();
//Enviamos modelo Cuerpo de Pedro a la matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Cuerpo
CuerpoP.Draw(lightingShader);

//Iniciamos Camara de vista
view = camera.GetViewMatrix();
//Enviamos modelo Pies de Pedro a la matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Pies
PiesP.Draw(lightingShader);

//Iniciamos Camara de vista
view = camera.GetViewMatrix();
//Transformación de traslación al modelo
model = glm::translate(model, glm::vec3(posX2, posY2, posZ2));
//Transformación de rotación al modelo
model = glm::rotate(model, glm::radians(-rotBrazoDer), glm::vec3(0.0f, 0.0f, 1.0f));
//Enviamos modelo brazo de derecho de Pedro a la matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Brazo Derecho
BrazoDer.Draw(lightingShader);
```

```
//Iniciamos Camara de vista
view = camera.GetViewMatrix();
model = glm::translate(model, glm::vec3(posX2, posY2, posZ2));
//Enviamos modelo brazo de izquierdo de Pedro a la matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Brazo Izquierdo
BrazoIzq.Draw(lightingShader);

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

Figura 58: Dibujado de Pedro Picapiedra

- Inicialización de animación y objeto, en este caso se tiene establecida la animación como shader por el cual lo vamos a mandar a llamar y lo dibujamos.

```
//Iniciamos el shader animación para uso
Anim.Use();
//Utilizamos el tiempo generado por procesador para animación
tiempo = glfwGetTime();
//Inicamos modelo, vista y proyección del modelo
modelLoc = glGetUniformLocation(Anim.Program, "model");
viewLoc = glGetUniformLocation(Anim.Program, "view");
projLoc = glGetUniformLocation(Anim.Program, "projection");
//Enviamos vista a la matriz de modelo
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
//Enviamos proyección a la matriz de modelo
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
//Enviamos modelo a la matriz de modelo
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Enviamos el tiempo del procesador a la matriz de modelo
glUniform1f(glGetUniformLocation(Anim.Program, "time"), tiempo);
//Inicializamos modelo de objeto agua
model = glm::mat4(1);
//Enviamos modelo de objeto
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos objeto con parametro de animación recibido
Agua.Draw(Anim);
//Finalizamos vertex array
glBindVertexArray(0);
```

Figura 59: Dibujado de agua

- Inicialización de modelo para dibujado de cubos por medio de triángulos, a su vez igual forma de inicializa el shader de iluminación y se asigna la posición inicial establecida anteriormente.

```
// Inicializamos uso de shader de iluminación
lampShader.Use();

//Inicializamos objeto de modelo, vista y proyección
modelLoc = glGetUniformLocation(lampShader.Program, "model");
viewLoc = glGetUniformLocation(lampShader.Program, "view");
projLoc = glGetUniformLocation(lampShader.Program, "projection");

//Enviamos vista y proyección a la matriz
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
//Inicialización del modelo de iluminación
model = glm::mat4(1);
//Transformación de traslación en iluminación
model = glm::translate(model, lightPos);
//Transformación de escalado en iluminación
model = glm::scale(model, glm::vec3(0.2f));
//Enviar modelo a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
// Draw the light object (using light's vertex attributes)


//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);

//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,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);
```

Figura 60: Dibujado de Cubos Iluminación

- Dibujado del cubo y asignación de cada textura asignada a cada cara, al igual activación de textura y dibujado por triángulos del skybox.

```
// 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));

// Dibujamos el cubo para la asignación de skybox
glBindVertexArray(skyboxVAO);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_CUBE_MAP, cubemapTexture);
glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);
glDepthFunc(GL_LESS); // Set depth function back to default



// intercambiamos los buffer de ventana
glfwSwapBuffers(window);
```

Figura 61: Dibujado de Cubos Iluminación

- Liberamos la memoria asignada para todo el buffer, vertex shader y demás propiedades.

```
//Eliminamos memoria de skybox y objetos
glDeleteVertexArrays(1, &VAO);
glDeleteBuffers(1, &VBO);
glDeleteVertexArrays(1, &skyboxVAO);
glDeleteBuffers(1, &skyboxVBO);
// Terminamos GLFW para limpiar cualquier recurso almacenado
glfwTerminate();



return 0;
```

Figura 62: Liberación de Memoria

- 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 63: 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.

```
// Funciónde movimiento de teclas
void DoMovement()
{

    // Control de camara para enfrente
    if (keys[GLFW_KEY_W] || keys[GLFW_KEY_UP])
    {
        camera.ProcessKeyboard(FORWARD, deltaTime);
    }

    //Control de camara para atras
    if (keys[GLFW_KEY_S] || keys[GLFW_KEY_DOWN])
    {
        camera.ProcessKeyboard(BACKWARD, deltaTime);
    }

    //Control de camara para izquierda
    if (keys[GLFW_KEY_A] || keys[GLFW_KEY_LEFT])
    {
        camera.ProcessKeyboard(LEFT, deltaTime);
    }

    //Control de camara para derecha
    if (keys[GLFW_KEY_D] || keys[GLFW_KEY_RIGHT])
    {
        camera.ProcessKeyboard(RIGHT, deltaTime);
    }
```

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

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


    //Movimiento de cuerpo de dinosaurio positivo
    if (keys[GLFW_KEY_1])
    {
        if (rotCuerpo < 45.0)
            rotCuerpo += 1.0f;

    }

    //Movimiento de cuerpo de dinosaurio negativo
    if (keys[GLFW_KEY_2])
    {
        if (rotCuerpo > -1)
            rotCuerpo -= 1.0f;

    }

    //Movimiento de brazos de Pedro positivo
    if (keys[GLFW_KEY_3])
    {
        if (rotBrazoDer< 20.0)
            rotBrazoDer += 1.0f;
    }

    //Movimiento de brazos de Pedro negativo
    if (keys[GLFW_KEY_4])
    {
        if (rotBrazoDer > 0)
            rotBrazoDer -= 1.0f;

    }

}
```

Figura 64: Liberación de Memoria

- Declaración de función para primer animación la cual está asignada al dragón en la cual se representará el movimiento cuadrado de su recorrido, igual se activará cada recorrido por medio una bandera que una vez llegue al límite de recorrido cambiará la dirección de dicho recorrido.

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

```
//Bandera de activación recorrido 3
if (recorrido3)
{
    //Desplazamiento en Z negativo
    movKitZ -= 0.1f;
    //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 > 180)
    {
        //Cambio de recorrido
        recorrido4 = false;
        recorrido5 = true;
    }
}
//Bandera de activación recorrido 5
if (recorrido5)
{
    //Desplazamiento en X negativo
    movKitX -= 0.1f;
    //Bandera de limite de 20 unidades menos
    if (movKitX < -20)
    {
        //Cambio de recorrido
        recorrido5 = false;
        recorrido6 = true;
    }
}
```

Figura 65: Función de Animación Dragón

- Declaración de función para primer animación la cual está asignada al dragón en la cual se representará el movimiento cuadrado de su recorrido, igual se activará cada recorrido por medio una bandera que una vez llegue al límite de recorrido cambiará la dirección de dicho recorrido.

```
//Bandera de activación recorrido 6
if (recorrido6)
{
    //Cambio de dirección
    rotKit += 1;
    //Bandera de limite en 270 grados
    if (rotKit > 270)
    {
        //Cambio de recorrido
        recorrido6 = false;
        recorrido7 = true;
    }
}
//Bandera de activación recorrido 7
if (recorrido7)
{
    //Desplazamiento en Z positivo
    movKitZ += 0.1f;
    //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 > 360)
    {
        //Cambio de recorrido
        recorrido8 = false;
        recorrido9 = true;
    }
}
    //Bandera de activación recorrido 9
    if (recorrido9)
    {
        //Desplazamiento en X positivo
        movKitX += 0.1f;
        //Bandera de limite de 0 unidades
        if (movKitX > 0)
        {
            //Cambio de recorrido
            recorrido9 = false;
            rotKit = 0;
            recorrido1 = true;
        }
    }

  }
}
```

Figura 66: Función de Animación Dragón 2

- Declaración de función de segunda animación en el cual se hace la declaración y animación por medio de key frames con su recorrido por medio de una bandera, una vez terminado el recorrido por frames se inicializa a la posición inicial.

```
//Funcion de Animacion 2 por key frames
void animacion2()
{

    //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;

            i_curr_steps++;
        }

    }
}
```

Figura 67: Funciones de Animación Key Frames

- Declaración de función de segunda animación en el cual se hace la declaración y animación por medio de key frames 2 con su recorrido por medio de una bandera, una vez terminado el recorrido por frames se inicializa a la posición inicial.

```
//Funcion de Animacion 3 por key frames
void animacion3()
{
    //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();
            }
        }
        else
        {
            //Recorrido de posición
            posX2 += KeyFrame2[playIndex2].incX2;
            posY2 += KeyFrame2[playIndex2].incY2;
            posZ2 += KeyFrame2[playIndex2].incZ2;

            rotBrazoDer += KeyFrame2[playIndex2].rotInc2;

            i_curr_steps2++;
        }

    }
}
```

Figura 68: Funciones de Key Frames 2

- Declaración de función de teclas para uso de una sola vez, en esta sección se realizará la asignación de movimiento de una sola vez, y no constantes, se declara el uso de las teclas, prendido y apagado de TV, movimiento de Dragón.

```
    // Activación de una tecla de solo un momento
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode)
{
    if (GLFW_KEY_ESCAPE == key && GLFW_PRESS == action)
    {
        glfwSetWindowShouldClose(window, GL_TRUE);
    }

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

    //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;
        }

    }
```

Figura 69: Funciones de Tecla una vez

- Declaración para la función de teclas para el grabado de key frames, para la reproducción del mismo con la tecla L llamando interpolación y vemos el índice de frames.

```
//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 70: Funciones de reproducción y salvado

- Declaración para la función de teclas para el grabado de key frames, para la reproducción del mismo con la telca P llamando interpolación y vemos el índice de frames.

```
//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;
    }

}
```

Figura 71: Funciones de reproducción y salvado

- Declaración para el uso de mouse y a su vez regresar a la última posición almacenada lo que quiere decir usar la reciente y la anterior de movimiento.

```
//Función de llamada de mouse
void MouseCallback(GLFWwindow* window, double xPos, double yPos)
{
    //Bandera para lectura de posición de mouse
    if (firstMouse)
    {
        lastX = xPos;
        lastY = yPos;
        firstMouse = false;
    }

    //Almacenado de posición anterior de ambos ejes
    GLfloat xOffset = xPos - lastX;
    GLfloat yOffset = lastY - yPos;  // Reversed since y-coordinates go from bottom to left

    //Almacen de posición actual
    lastX = xPos;
    lastY = yPos;

    //enviado de posición a funcion y procesamiento de camara
    camera.ProcessMouseMovement(xOffset, yOffset);
}
```

Figura 72: Funciones de Mouse Lectura

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	En progreso
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

Figura 73: Gantt Diagram Part 1

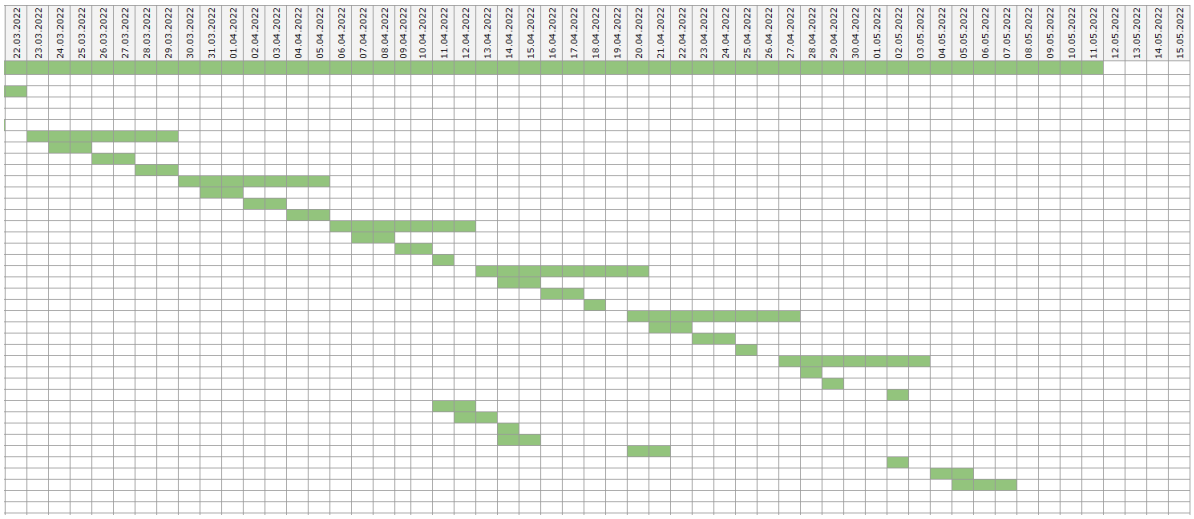

Figura 74: Gantt Diagram Part 2

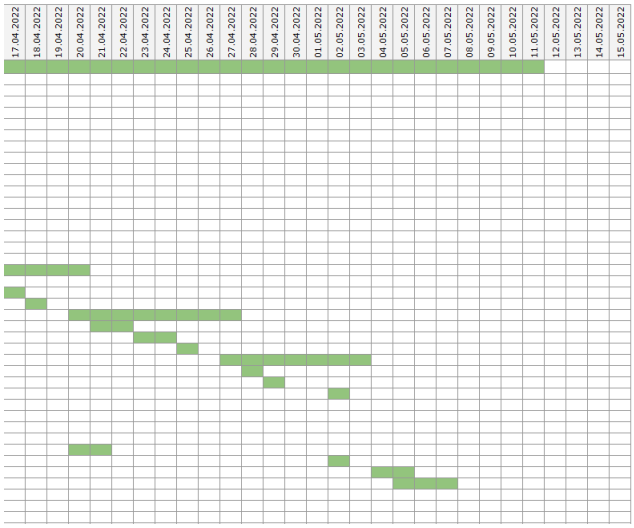

Figura 75: Gantt Diagrama Part 3

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 76: 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 77: Functions Declaration

- We create this vairables to use for the initialization of a window size in wich resoultion and it will be executed.

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

Figura 78: Resolution Declaration

- The declaration of this has a relation to the screen to be in the middle of resolution, as a mind point of window.

```
// Declaración de modelo de camara y su posición
Camera  camera(glm::vec3(4.0f, 1.0f, 0.0f));
GLfloat lastX = WIDTH / 2.0;
GLfloat lastY = HEIGHT / 2.0;
```

Figura 79: MindPoint Window

- The array is declared for the assignment of keys to be used, in this case is asigned a total of 1024 of memory.

```
//Arreglo para teclas
bool keys[1024];
```

Figura 80: Key Memory Declaration

- Activation flags for mouse movement, sending the flas as true.

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

Figura 81: Mouse Flag

- Lighting atributes that having a initial positions is (0,0,0)

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

Figura 82: Lighting Position

- 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)

```
//Declaración de posición inical de objeto a mover con Key Frames
glm::vec3 PosIni(0.0f, 0.0f, 0.0f);
glm::vec3 PosIni2(0.0f, 0.0f, 0.0f);
```

Figura 83: Objects Initial Position

- 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.

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

Figura 84: Variable Declaration

- Array of initial points for the position of the point light in which it will be draw from the beginning.

```
// Posición inicial del pointlight y cubo
glm::vec3 pointLightPositions[] = {
    glm::vec3(0.0f,3.0f, 0.0f),
    glm::vec3(1.5f, 0.82f, 1.15f)

};
```

Figura 85: Point Lights Position

- 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.

```
//Variables de movimiento y rotación para animación 3
float movKitX = 0.0;
float movKitZ = 0.0;
float rotKit = 0.0;
```

Figura 86: Animation Variable 2

- Variable to use as a flag in which it will be activate the route of the dragon.

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

Figura 87: Circuit Flag

- Flags variable to make the change of position of the route when the process is completed.

```
/*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;
```

Figura 88: 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.

```
// 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
```

Figura 89: Frames Variables

- 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.

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

Figura 90: Frames Variables For Model

- Definition as a constant of the variable max_frame, which will be the limit of store frame .

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

Figura 91: Max Frames Constant

- 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.

```
//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 92: Max and Actual

- Definition of frame structure 1, 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

}FRAME;
```

Figura 93: Frame Struct 1

- Definition of frame structure 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 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 94: Frame Struct 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;
```

Figura 95: Variables for Frame Key 1

- 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 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 96: Variables for Frame Key 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 97: Vertex Array

- Initialization of lighting variable in vector 3, to assing transformations, positions,etc

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

Figura 98: Light 1

- Function that works to save the frame in each position, assigning from initial position X,Y,Z for the rotate body.

```
//Función para salvar los frames
void saveFrame(void)
{
    //Imprime el numero de frame almacenado
    printf("frameindex %d\n", FrameIndex);

    //Arreglo de estructura para almacenar la posición en el frame en el elemento de posición
    KeyFrame[FrameIndex].posX = posX;
    KeyFrame[FrameIndex].posY = posY;
    KeyFrame[FrameIndex].posZ = posZ;

    KeyFrame[FrameIndex].rotCuerpo= rotCuerpo;

    FrameIndex++;
}
```

Figura 99: SaveFrame Function

- Function that works to save the frame in each position, assigning from initial position X,Y,Z for the rotate arms.

```
//Función para salvar los frames
void saveFrame2(void)
{
    //Arreglo de estructura para almacenar la posición en el frame en el elemento de posición
    printf("frameindex %d\n", FrameIndex2);

    KeyFrame2[FrameIndex2].posX2 = posX2;
    KeyFrame2[FrameIndex2].posY2 = posY2;
    KeyFrame2[FrameIndex2].posZ2 = posZ2;

    KeyFrame2[FrameIndex2].rotBrazoDer = rotBrazoDer;


    FrameIndex2++;
}
```

Figura 100: SaveFrame Function 2

- 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 .

```
//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;
}
```

Figura 101: ResetElements Function

- 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.

```
//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 102: ResetElements Function 2

- 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

}
```

Figura 103: Interpolation 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 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 104: Interpolation Function 2

- Initilization of the functions libraries for graph processing, the following function sets hints for the net call to glfwCreateWindown.

```
// Inicialización de  GLFW
glfwInit();
// Seteamos los valores requeridos por GLFW
glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
```

Figura 105: Initialization of Exec

- To create the window is initialized with de window variable, in which is receives the resolution variables and the title, previously the functioons to activate the windows was intiliazed.

```
// Creamos el objeto de ventana por medio de la funcion de glfw, recibiendo resolución y nombre de ventana.
GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "Picapiedras", nullptr, nullptr);

//Comprobación de creación de venta correctamente.
if (nullptr == window)
{
    std::cout << "Failed to create GLFW window" << std::endl;
    glfwTerminate();

    return EXIT_FAILURE;
}
```

Figura 106: Windows Exec

- All the fuctions was initliazed and they set to receive as input methos, in other way verified that all context was initialized correct, and if it is true, it is sent to the viewport for window construction.

```
//Cambio de contexto anterior a nuevo
glfwMakeContextCurrent(window);

//Recibimos el tamaño de buffer de ventana
glfwGetFramebufferSize(window, &SCREEN_WIDTH, &SCREEN_HEIGHT);

// Inicializamos las funciones de callback de mouse y teclado
glfwSetKeyCallback(window, KeyCallback);
glfwSetCursorPosCallback(window, MouseCallback);

// Opciones de GLFW
glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);

// Inicializamos para recibir funciones de punteros y extensiones
glewExperimental = GL_TRUE;
// Inicializamos GLEW para los punteros y verificamos que se haya creado correctamente
if (GLEW_OK != glewInit())
{
    std::cout << "Failed to initialize GLEW" << std::endl;
    return EXIT_FAILURE;
}

// Definimos las dimensiones del viewport
glViewport(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT);

glEnable(GL_DEPTH_TEST);
```

Figura 107: Initialized Exec

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

```
/*
* 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 108: Shaders Initialized

- The built of this objects to construction of Pedro Flintstone, this we created the object 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");
```

Figura 109: Pedro Picapiedra Model Initialize

- The built of this objects to construction of Pedro Flintstone, this we created the object by a model for OpenGL.

```
//Modelo de Dinosaurio
Model Dino((char*)"Models/Obj/Models/Dragon/Dragon.obj");
```

Figura 110: Dragon Model Initialize

- The animation shader is called, set as an external library initialized in a local anim variable.

```
//Shader creado para animacion del agua
Shader Anim("Shaders/anim.vs", "Shaders/anim.frag");
```

Figura 111: Animation Shader Initialize

- 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.

```
//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
Model DinosaurioPies((char*)"Models/Obj/Models/DinoPerson/Pies/Pies.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");
//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 Libros
Model Libros((char*)"Models/Obj/Models/Libros/Libros.obj");
```

Figura 112: 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].rotCuerpo = 0;
KeyFrame[1].rotCuerpo = 45;
KeyFrame[2].rotCuerpo = -1;
FrameIndex = 4;



KeyFrame2[0].rotBrazoDer = 0;
KeyFrame2[1].rotBrazoDer = 20;
KeyFrame2[2].rotBrazoDer = 0;
FrameIndex2 = 4;
```

Figura 113: 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 114: Skybox Vertex Box

- All properties of the vertex array for model building, buffer setting, memory allocation for position and normal. Finally we initialize the use of the lighting shader for diffuse and specular light.

```
// Inicialización de las funciones VAO y VBO
GLuint VBO, VAO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
glBindVertexArray(VAO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

//Establecemos el atributo de posición
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(0);
//Establecemos el atributo para la normal
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 * sizeof(float)));
glEnableVertexAttribArray(1);

// Inicilización de unidad de textura
lightingShader.Use();
glUniform1i(glGetUniformLocation(lightingShader.Program, "diffuse"), 0);
glUniform1i(glGetUniformLocation(lightingShader.Program, "specular"), 1);
```

Figura 115: Basic stay initialize

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

```
//Inicialización del SkyBox
GLuint skyboxVBO, skyboxVAO;
glGenVertexArrays(1, &skyboxVAO);
glGenBuffers(1, &skyboxVBO);
glBindVertexArray(skyboxVAO);
glBindBuffer(GL_ARRAY_BUFFER, skyboxVBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(skyboxVertices), &skyboxVertices, GL_STATIC_DRAW);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);

//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/back.tga");
faces.push_back("SkyBox/front.tga");
```

Figura 116: Skybox Texture

- The textures will be loaded in each faces of box, and set perspective view in the same model

```
GLuint cubemapTexture = TextureLoading::LoadCubemap(faces);
//Creación de proyección en perspectiva
glm::mat4 projection = glm::perspective(camera.GetZoom(), (GLfloat)SCREEN_WIDTH / (GLfloat)SCREEN_HEIGHT, 0.1f, 100.0f);
```

Figura 117: Skybox Textures Faces

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

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

Figura 118: Sound Function Call

- Time initialize, with his calculation and asiggment to delta that is the difference between the current frame to the last, setting the current as last.

```
// Calculamos el ultimo frame y su diferentecía entre el actual y el ultimo
GLfloat currentFrame = glfwGetTime();
deltaTime = currentFrame - lastFrame;
lastFrame = currentFrame;
```

Figura 119: Time Initialize

- 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.

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

Figura 120: Animation Function Initialize

- The color buffer is initialized to safe values and set.

```
// Limpiamos el buffer de ejecución y color
glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Figura 121: Color Buffer Clean

- Initialize the lighting shader and his initial position of camera will be start.

```
// Declaramos el shader de iluminación a inicar uso
lightingShader.Use();
GLint viewPosLoc = glGetUniformLocation(lightingShader.Program, "viewPos");
//Iniciamos su posicionamiento
glUniform3f(viewPosLoc, camera.GetPosition().x, camera.GetPosition().y, camera.GetPosition().z);
```

Figura 122: Models 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.

```
// 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);


// Vector de Iluminación de punto
glm::vec3 lightColor;
lightColor.x = abs(sin(glfwGetTime() * Light1.x));
lightColor.y = abs(sin(glfwGetTime() * Light1.y));
lightColor.z = sin(glfwGetTime() * Light1.z);

//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);

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

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

Figura 123: Lighting Initiliaze

- The initiliazation is start again, to send another model matrix, in this case is the camera that will be his properites previously initialized.

```
// Creamos la creación de camara
glm::mat4 view;
view = camera.GetViewMatrix();

// Iniciamos la prespectica de model, view y proyección
GLint modelLoc = glGetUniformLocation(lightingShader.Program, "model");
GLint viewLoc = glGetUniformLocation(lightingShader.Program, "view");
GLint projLoc = glGetUniformLocation(lightingShader.Program, "projection");

// Realizamos la matreiz para la vista y proyección
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));

glBindVertexArray(VAO);
glm::mat4 tmp = glm::mat4(1.0f); //Temp
```

Figura 124: Model Matriz Initialize

- Next the model will be loaded for the drawing and assing in the model matrix previously initialize, so in the draw line we sent 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(posX, posY, posZ));//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
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos matriz de modelo
DinosaurioPies.Draw(lightingShader);//Dibujamos el modelo  con parametro recibido de shader de iluminación

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

Figura 125: Dinosaur Drawing

- Now we load the dragon model to assing the transformationos of rotation and transalation, waiting to recieve a value that will be in animation 2,and drawn in the ending.

```
//Caragamos modelo del dragon volador
view = camera.GetViewMatrix();//Iniciamos vista
model = glm::mat4(1);//Iniciamos modelo
model = glm::translate(model, PosIni + glm::vec3(movKitX, 0, 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
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));//Enviamos a la matriz el modelo
Dino.Draw(lightingShader);//Dibujamos el modelo del dinosaurio
glBindVertexArray(0);//Finalizamos vertex array
```

Figura 126: Dragon Drawing

- The floor model is loaded for drawing, the transparency is activated for not be transpared and blurred.

```
//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
```

Figura 127: Floor Drawing

- The model of objects inside the facade is loaded: Books, plants, table, sofa, pillow, tv,etc. And initial position already assinged.

```
//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));
//Dibujamos el modelo de Sillon2
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);
```

Figura 128: Objects Drawing

- Alpha color initialization

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

Figura 129: Alpha Color

- Initialize a new model for assing the model of drawing to Pedro Flinstones, in wich assing movement Pedro's Arms for his movement through key frames.

```
//Carga de Modelo Pedro Picapiedra
//Iniciamos Camara de vista
view = camera.GetViewMatrix();
//Enviamos modelo Cabeza de Pedro a la matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Cabeza
CabezaP.Draw(lightingShader);

//Iniciamos Camara de vista
view = camera.GetViewMatrix();
//Enviamos modelo Cuerpo de Pedro a la matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Cuerpo
CuerpoP.Draw(lightingShader);

//Iniciamos Camara de vista
view = camera.GetViewMatrix();
//Enviamos modelo Pies de Pedro a la matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Pies
PiesP.Draw(lightingShader);

//Iniciamos Camara de vista
view = camera.GetViewMatrix();
//Transformación de traslación al modelo
model = glm::translate(model, glm::vec3(posX2, posY2, posZ2));
//Transformación de rotación al modelo
model = glm::rotate(model, glm::radians(-rotBrazoDer), glm::vec3(0.0f, 0.0f, 1.0f));
//Enviamos modelo brazo de derecho de Pedro a la matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Brazo Derecho
BrazoDer.Draw(lightingShader);
```

```
//Iniciamos Camara de vista
view = camera.GetViewMatrix();
model = glm::translate(model, glm::vec3(posX2, posY2, posZ2));
//Enviamos modelo brazo de izquierdo de Pedro a la matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos el modelo de Brazo Izquierdo
BrazoIzq.Draw(lightingShader);

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

Figura 130: Pedro's Flintstones Drawing

- Initialization of animation and objects, in this case the animation is assing as a shader so which we are going to call it and draw it.

```
//Iniciamos el shader animación para uso
Anim.Use();
//Utilizamos el tiempo generado por procesador para animación
tiempo = glfwGetTime();
//Inicamos modelo, vista y proyección del modelo
modelLoc = glGetUniformLocation(Anim.Program, "model");
viewLoc = glGetUniformLocation(Anim.Program, "view");
projLoc = glGetUniformLocation(Anim.Program, "projection");
//Enviamos vista a la matriz de modelo
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
//Enviamos proyección a la matriz de modelo
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
//Enviamos modelo a la matriz de modelo
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Enviamos el tiempo del procesador a la matriz de modelo
glUniform1f(glGetUniformLocation(Anim.Program, "time"), tiempo);
//Inicializamos modelo de objeto agua
model = glm::mat4(1);
//Enviamos modelo de objeto
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//Dibujamos objeto con parametro de animación recibido
Agua.Draw(Anim);
//Finalizamos vertex array
glBindVertexArray(0);
```

Figura 131: Water Drawing

- Initialize the model for drawing boxes by triangles, and the same way intialize the lighting shader and assing the initial position previously established.

```
// Inicializamos uso de shader de iluminación
lampShader.Use();

//Inicializamos objeto de modelo, vista y proyección
modelLoc = glGetUniformLocation(lampShader.Program, "model");
viewLoc = glGetUniformLocation(lampShader.Program, "view");
projLoc = glGetUniformLocation(lampShader.Program, "projection");

//Enviamos vista y proyección a la matriz
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
//Inicialización del modelo de iluminación
model = glm::mat4(1);
//Transformación de traslación en iluminación
model = glm::translate(model, lightPos);
//Transformación de escalado en iluminación
model = glm::scale(model, glm::vec3(0.2f));
//Enviar modelo a matriz
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
// Draw the light object (using light's vertex attributes)


//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);

//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,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);
```

Figura 132: Lighting Box Drawing

- Drawing the box and assing each texture in his faces, as well as the activation of texture and drawing by triangles of the skybox.

```
// 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));

// Dibujamos el cubo para la asignación de skybox
glBindVertexArray(skyboxVAO);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_CUBE_MAP, cubemapTexture);
glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);
glDepthFunc(GL_LESS); // Set depth function back to default



// intercambiamos los buffer de ventana
glfwSwapBuffers(window);
```

Figura 133: Lighting Box Drawing

- Free the allocated memory fot all buffers, vertex shader and other properties.

```
//Eliminamos memoria de skybox y objetos
glDeleteVertexArrays(1, &VAO);
glDeleteBuffers(1, &VBO);
glDeleteVertexArrays(1, &skyboxVAO);
glDeleteBuffers(1, &skyboxVBO);
// Terminamos GLFW para limpiar cualquier recurso almacenado
glfwTerminate();



return 0;
```

Figura 134: Memory Free

- 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 135: Sound Functions

- Declaration of function of keys in which we will be activated movement camera so it will be executed when interacted, where the movement key of dinosaur to drink water, and the movement of Pedro's arms movement.

```
// Funciónde movimiento de teclas
void DoMovement()
{

    // Control de camara para enfrente
    if (keys[GLFW_KEY_W] || keys[GLFW_KEY_UP])
    {
        camera.ProcessKeyboard(FORWARD, deltaTime);

    }

    //Control de camara para atras
    if (keys[GLFW_KEY_S] || keys[GLFW_KEY_DOWN])
    {
        camera.ProcessKeyboard(BACKWARD, deltaTime);


    }

    //Control de camara para izquierda
    if (keys[GLFW_KEY_A] || keys[GLFW_KEY_LEFT])
    {
        camera.ProcessKeyboard(LEFT, deltaTime);


    }

    //Control de camara para derecha
    if (keys[GLFW_KEY_D] || keys[GLFW_KEY_RIGHT])
    {
        camera.ProcessKeyboard(RIGHT, deltaTime);


    }
```

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

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


    //Movimiento de cuerpo de dinosaurio positivo
    if (keys[GLFW_KEY_1])
    {
        if (rotCuerpo < 45.0)
            rotCuerpo += 1.0f;

    }

    //Movimiento de cuerpo de dinosaurio negativo
    if (keys[GLFW_KEY_2])
    {
        if (rotCuerpo > -1)
            rotCuerpo -= 1.0f;

    }

    //Movimiento de brazos de Pedro positivo
    if (keys[GLFW_KEY_3])
    {
        if (rotBrazoDer< 20.0)
            rotBrazoDer += 1.0f;
    }

    //Movimiento de brazos de Pedro negativo
    if (keys[GLFW_KEY_4])
    {
        if (rotBrazoDer > 0)
            rotBrazoDer -= 1.0f;

    }
}
```

Figura 136: Movement Object and Camera

- Function declaration for first animation which is assigned to the dragon move as square route, so we will activate the route through a flag when the limit is maxium the route change dthe direction.

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

```
//Bandera de activación recorrido 3
if (recorrido3)
{
    //Desplazamiento en Z negativo
    movKitZ -= 0.1f;
    //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 > 180)
    {
        //Cambio de recorrido
        recorrido4 = false;
        recorrido5 = true;
    }
}
//Bandera de activación recorrido 5
if (recorrido5)
{
    //Desplazamiento en X negativo
    movKitX -= 0.1f;
    //Bandera de limite de 20 unidades menos
    if (movKitX < -20)
    {
        //Cambio de recorrido
        recorrido5 = false;
        recorrido6 = true;
    }
}
```

Figura 137: Dragon Animation Function

- Function declaration for first animation which is assigned to the dragon move as square route, so we will activate the route through a flag when the limit is in the route change dthe direction.

```
//Bandera de activación recorrido 6
if (recorrido6)
{
    //Cambio de dirección
    rotKit += 1;
    //Bandera de limite en 270 grados
    if (rotKit > 270)
    {
        //Cambio de recorrido
        recorrido6 = false;
        recorrido7 = true;
    }
}
//Bandera de activación recorrido 7
if (recorrido7)
{
    //Desplazamiento en Z positivo
    movKitZ += 0.1f;
    //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 > 360)
    {
        //Cambio de recorrido
        recorrido8 = false;
        recorrido9 = true;
    }
}
        //Bandera de activación recorrido 9
        if (recorrido9)
        {
            //Desplazamiento en X positivo
            movKitX += 0.1f;
            //Bandera de limite de 0 unidades
            if (movKitX > 0)
            {
                //Cambio de recorrido
                recorrido9 = false;
                rotKit = 0;
                recorrido1 = true;
            }
        }

    }
}
```

Figura 138: Dragon Animation Function 2

- Function declaration of second animation, in which the declaration and animation will be do it by key frame, the route is through a flag and when the route has finished by fram is initialize his inital position.

```
//Funcion de Animacion 2 por key frames
void animacion2()
{

    //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;

            i_curr_steps++;
        }

    }
}
```

Figura 139: Key Frames Animation Function

- Function declaration of third animation, in which the declaration and animation will be do it by key frame, the route is through a flag and when the route has finished by fram is initialize his inital position.

```
//Funcion de Animacion 3 por key frames
void animacion3()
{
    //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();
            }
        }
        else
        {
            //Recorrido de posición
            posX2 += KeyFrame2[playIndex2].incX2;
            posY2 += KeyFrame2[playIndex2].incY2;
            posZ2 += KeyFrame2[playIndex2].incZ2;

            rotBrazoDer += KeyFrame2[playIndex2].rotInc2;

            i_curr_steps2++;
        }

    }
}
```

Figura 140: Key Frames Animation Function 2

- Function declaration of the key call back, in this section will do the assign movemente only once, and not constant, declare his uses of key, turn on and off TV, and movement dragon.

```
// Activación de una tecla de solo un momento
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode)
{
    if (GLFW_KEY_ESCAPE == key && GLFW_PRESS == action)
    {
        glfwSetWindowShouldClose(window, GL_TRUE);
    }

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

    //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;
        }

    }
```

Figura 141: Key Callback Function

- Declaration function for recording key frames, in this the L key for his playbak and calling the interpolation function so we will se the frame index frame.

```
//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 142: Recording and Playback Function

- Declaration function for recording key frames, in this the P key for his playbak and calling the interpolation function so we will se the frame index frame.

```
//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;
    }

}
```

Figura 143: Recording and Playback Function 2

- Declaration for use the mouse and return his position stored, which means using the recent and previous movement

```
//Función de llamada de mouse
void MouseCallback(GLFWwindow* window, double xPos, double yPos)
{
    //Bandera para lectura de posición de mouse
    if (firstMouse)
    {
        lastX = xPos;
        lastY = yPos;
        firstMouse = false;
    }

    //Almacenado de posición anterior de ambos ejes
    GLfloat xOffset = xPos - lastX;
    GLfloat yOffset = lastY - yPos;  // Reversed since y-coordinates go from bottom to left

    //Almacen de posición actual
    lastX = xPos;
    lastY = yPos;

    //enviado de posición a funcion y procesamiento de camara
    camera.ProcessMouseMovement(xOffset, yOffset);
}
```

Figura 144: Reading Position Mouse 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.