UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Computación Gráfica e Interacción Humano Computadora

Grupo 04

Proyecto Final

Manual Técnico

Alumno: Corona Vera Logan Alejandro

Fecha de entrega: 26 de enero del 2021

INTRODUCCIÓN

El manual consta de la explicación técnica del proyecto, haciendo referencia a lo solicitado por el profesor y su solución en código C++.

CONTENIDO

IDE Y LENGUAJE DE PROGRAMACIÓN

Utilicé Visual Studio 2017 el cual es un entorno de desarrollo integrado para Windows, Linux y macOS. Es compatible con múltiples lenguajes de programación, tales como C++.

BIBLIOTECAS Y CABECERAS

Windows.h: Contiene todas las funciones, estructuras, macros y constantes numéricas que forman la API.

Glad/glad.h: Es una biblioteca que carga punteros a las funciones de OpenGL en tiempo de ejecución, el núcleo y las extensiones.

GLFW: es una biblioteca de utilidad ligera para uso con OpenGL. Proporciona a los programadores la capacidad de crear y dirigir ventanas y aplicaciones OpenGL, así como recibir la entrada de joystick, teclado y ratón.

Stdlib.h: Contiene los prototipos de funciones de C para gestión de memoria dinámica, control de procesos y otras.

GLM: es una librería matemática escrita en C++ para el desarrollo de software gráfico basado en OpenGL.

Time.h: Relacionado con formato de hora y fecha es un archivo de cabecera de la biblioteca estándar del lenguaje de programación C que contiene funciones para manipular y formatear la fecha y hora del sistema.

SDL: es una biblioteca multiplataforma (Linux, Windows, ...) para el control multimedia del ordenador

iostream: es un componente de la biblioteca estándar del lenguaje de programación C++ que es utilizado para operaciones de entrada/salida.

camera.h: Se encargará del funcionamiento adecuado de la cámara.

Model.h: Se encargará de la funcionalidad que se requiere para cargar los modelos 3D.

Texture.h: Se encargará de la carga de texturas para su uso en el ambiente gráfico.

```
#include <Windows.h>

#include <glad/glad.h>
#include <glfw3.h>  //main
#include <stdlib.h>
#include <glm/glm.hpp>  //camera y model
#include <glm/gtc/matrix_transform.hpp> //camera y model
#include <glm/gtc/type_ptr.hpp>
#include<time.h>


#define STB_IMAGE_IMPLEMENTATION
#include <stb_image.h>  //Texture

#define SDL_MAIN_HANDLED
#include <SDL/SDL.h>

#include <shader_m.h>
#include <camera.h>
#include <modelAnim.h>
#include <model.h>
#include <Skybox.h>
#include <iostream>
```

Configuraciones

En esta parte de la codificación de las configuraciones, estarán algunas declaraciones importantes dentro del proyecto.

Donde se encuentran el tamaño de la cámara, la velocidad del movimiento de la cámara, el tiempo, la dirección de la luz y el sonido.

```
// settings
const unsigned int SCR_WIDTH = 1920;
const unsigned int SCR_HEIGHT = 1200;

// camera
Camera camera(glm::vec3(90.0f, 25.0f, 420.0f));
float MovementSpeed = 1.5f;
float lastX = SCR_WIDTH / 2.0f;
float lastY = SCR_HEIGHT / 2.0f;
bool firstMouse = true;

// timing
const int FPS = 60;
const int LOOP_TIME = 1000 / FPS; // = 16 milisec // 1000 millisec == 1 sec
double  deltaTime = 0.0f,
        lastFrame = 0.0f;

//Lighting
glm::vec3 lightPosition(20.0f, 30.0f, -200.0f);
glm::vec3 lightDirection(-1.0f, -1.0f, -1.0f);

//Sound
void sound();
```

Variables

En esta parte de la documentación se muestran las declaraciones de las variables para las animaciones, tanto para el cuánto se van a mover cómo para, activar o desactivar el movimiento.

También están declaradas las variables para activar el sonido, así como la definición de cuantos FRAMES vamos a requerir para la animación por KEY FRAMES y sus variables para las posiciones

```
// posiciones
float   movBanJx = 0.0f,
        movBanJz = 0.0f,
        movCama1 = 0.0f,
        movCama2 = 0.0f,
        movMueble1 = 0.0f,
        movMueble2 = 0.0f,
        movMueble3 = 0.0f,
        orienta = 180.0f;
bool    animacion = false,
        recorrido1 = true,
        recorrido2 = false,
        recorrido3 = false,
        recorrido4 = false,
        recorrido5 = false,
        recorrido6 = false,
        recorridoBJ1 = false,
        recorridoBJ2 = false,
        animacionBJ = false,
        recorridoBJR1 = false,
        recorridoBJR2 = false,
        animacionBJR = false,
        animacionCM = false,
        recorridoM1 = false,
        recorridoM2 = false,
        recorridoC1 = false,
        recorridoC2 = false,
        recorridoM3 = false,
        animacionCMR = false,
        recorridoMR1 = false,
        recorridoMR2 = false,
        recorridoCR1 = false,
        recorridoCR2 = false,
```

```
        recorridoM3 = false,
        animacionCMR = false,
        recorridoMR1 = false,
        recorridoMR2 = false,
        recorridoCR1 = false,
        recorridoCR2 = false,
        recorridoMR3 = false;

float initBottle = 270.0f,
      initBottleY = 20.0f,
      rotateBottle = 0.0f,
      mariY = 13.0f,
      mariZ = 25.0f;
//Keyframes (Manipulación y dibujo)
float   movAuto_x = 0.0f,
        movAuto_y = 0.0f,
        movAuto_z = 0.0f,
        giroAuto_y = 0.0f;
float   incX = 0.0f,
        incY = 0.0f,
        incZ = 0.0f,
        rotInc = 0.0f,
        giroAuto_yInc = 0.0f;

// Sound

bool activeAnim = false,
     activeAnimM = false,
     displayGlass = false,
            mari1 = true,
            mari2 = false;

bool soundon = true;
```

```
// Sound

bool activeAnim = false,
     activeAnimM = false,
     displayGlass = false,
            mari1 = true,
            mari2 = false;

bool soundon = true;


#define MAX_FRAMES 9
int i_max_steps = 60;
int i_curr_steps = 0;
typedef struct _frame
{
    //Variables para GUARDAR Key Frames
    float movAuto_x;        //Variable para PosicionX
    float movAuto_y;        //Variable para PosicionY
    float movAuto_z;        //Variable para PosicionY
    float giroAuto_y;       //Variable para PosicionZ

}FRAME;

FRAME KeyFrame[MAX_FRAMES];
int FrameIndex = 5;         //introducir datos
bool play = false;
int playIndex = 0;
```

```
void saveFrame(void)
{
    //printf("frameindex %d\n", FrameIndex);
    std::cout << "Frame Index = " << FrameIndex << std::endl;
    std::cout << "X = " << movAuto_x << std::endl;
    std::cout << "Y = " << movAuto_y << std::endl;
    std::cout << "Z = " << movAuto_z << std::endl;
    std::cout << "GIRO Y = " << giroAuto_y << std::endl;

    KeyFrame[FrameIndex].movAuto_x = movAuto_x;
    KeyFrame[FrameIndex].movAuto_y = movAuto_y;
    KeyFrame[FrameIndex].movAuto_z = movAuto_z;
    KeyFrame[FrameIndex].giroAuto_y = giroAuto_y;

    FrameIndex++;
}

void resetElements(void)
{
    movAuto_x = KeyFrame[0].movAuto_x;
    movAuto_y = KeyFrame[0].movAuto_y;
    movAuto_z = KeyFrame[0].movAuto_z;
    giroAuto_y = KeyFrame[0].giroAuto_y;
}
```

Función de interpolación

Esta función nos va a servir para obtener los FRAMES intermedios entre cada KEYFRAME que se guarde.

```
void interpolation(void)
{
    incX = (KeyFrame[playIndex + 1].movAuto_x - KeyFrame[playIndex].movAuto_x) / i_max_steps;
    incY = (KeyFrame[playIndex + 1].movAuto_y - KeyFrame[playIndex].movAuto_y) / i_max_steps;
    incZ = (KeyFrame[playIndex + 1].movAuto_z - KeyFrame[playIndex].movAuto_z) / i_max_steps;
    giroAuto_yInc = (KeyFrame[playIndex + 1].giroAuto_y - KeyFrame[playIndex].giroAuto_y) / i_max_steps;

}
```

Función sound():

Esta función sirve para cargar el sonido que nos servirá para ambientar el paisaje, así como configurarlo como Loop para que no deje su función.

```
// Sound
void sound() {
    if (soundon) {
        bool played = PlaySound("birds.wav", NULL, SND_LOOP | SND_ASYNC);
        cout << "Ambient:" << played << endl;
        soundon = false;
    }
}
```

Función anímate()

Esta función nos servirá para programar las animaciones que existirán en el escenario

```
void animate(void)
{
    if (play)
    {
        if (i_curr_steps >= i_max_steps) //end of animation between frames?
        {
            playIndex++;
            if (playIndex > FrameIndex - 2) //end of total animation?
            {
                std::cout << "Animation ended" << std::endl;
                //printf("termina anim\n");
                playIndex = 0;
                play = false;
            }
            else //Next frame interpolations
            {
                i_curr_steps = 0; //Reset counter
                                  //Interpolation
                interpolation();
            }
        }
        else
        {
            //Draw animation
            movAuto_x += incX;
            movAuto_y += incY;
            movAuto_z += incZ;
            giroAuto_y += giroAuto_yInc;

            i_curr_steps++;
        }
    }
```

```
//Botella
if (activeAnim)
{
    if (initBottle <= 275.0f)
    {
        initBottle += 0.1;
        if (rotateBottle <= 50.0f)
            rotateBottle += 1.0;
    }
    if (initBottleY >= -5.0f) {
        initBottleY -= 1.0f;
        if (initBottleY <= -6.0f)
            displayGlass = true;
    }
}

//////Mariposa
if (activeAnimM) {

    if (mari1) {
        mariY += 1.0f;
        mariZ -= 0.8f;
        if (mariY >= 14.0f) {
            mari2 = true;
            mari1 = false;
        }
    }
    if (mari2) {
        mariY -= 1.0f;
        if (mariY <= -6.0f) {
            mari1 = true;
            mari2 = false;
```

```
//Animación Banco de Jaimie
if (animacionBJ)//x = 14.0f, z=-24.5f
{
    if (recorridoBJ1)
    {
        movBanJz += 1.0f;
        if (movBanJz > 20.0f)
        {
            recorridoBJ1 = false;
            recorridoBJ2 = true;

        }
    }

    if (recorridoBJ2)
    {
        movBanJx += 1.0f;
        if (movBanJx > 13.0f)
        {
            recorridoBJ2 = false;
        }

    }
}
//Animación Banco de Jaimie Regreso a la Cuna
if (animacionBJR)//x = 14.0f, z=-24.5f
{
    if (recorridoBJR1)
    {
        movBanJx -= 1.0f;
        if (movBanJx < 0.0f)
        {
            recorridoBJR1 = false;
```

```
    }

    if (recorridoBJR2)
    {
        movBanJz -= 1.0f;
        if (movBanJz < 0.0f)
        {
            recorridoBJR2 = false;
        }

    }
}

//Animación de mover los muebles
if (animacionCM)
{
    if (recorridoM1)
    {
        movMueble1 -= 1.0f;
        if (movMueble1 < -40.0f)
        {
            recorridoM1 = false;
            recorridoC1 = true;

        }
    }

    if (recorridoC1)
    {
        movCama1 -= 1.0f;
        if (movCama1 < -37.0f)
        {
            recorridoC1 = false;
```

Función main()

En esta función se muestra todo lo que lleva nuestro escenario, todos los objetos, las operaciones matemáticas, la declaración de vectores, la carga de los modelos, de las texturas, la animación por KEYFRAMES.

La construcción, compilación, carga del skybox, la configuración del shader

```
// build and compile shaders
// -------------------------
//Shader staticShader("Shaders/lightVertex.vs", "Shaders/lightFragment.fs");
Shader staticShader("Shaders/shader_Lights.vs", "Shaders/shader_Lights.fs");
Shader skyboxShader("Shaders/skybox.vs", "Shaders/skybox.fs");
//Shader animShader("Shaders/anim.vs", "Shaders/anim.fs");

vector<std::string> faces
{
    "resources/skybox/right.jpg",
    "resources/skybox/left.jpg",
    "resources/skybox/top.jpg",
    "resources/skybox/bottom.jpg",
    "resources/skybox/front.jpg",
    "resources/skybox/back.jpg"
};

Skybox skybox = Skybox(faces);

// Shader configuration
// --------------------
skyboxShader.use();
skyboxShader.setInt("skybox", 0);

// load models
// -----------
/*---------------- MODELOS CASA ----------------*/
Model floor_house("resources/objects/PisoCasa/floor_house.obj");
Model barda_exterior("resources/objects/BardaJardin/barda_jardin.obj");
Model pasto("resources/objects/Pasto/pasto.obj");
```

```
/*---------------- MODELOS COCINA ----------------*/
Model cereal("resources/objects/ArticulosCocina/cajas.obj");
Model alacena("resources/objects/Alacena/alacena.obj");
Model refri("resources/objects/Refri/refri.obj");
Model mesaCocina("resources/objects/Mesa/mesa00.obj");
Model estufa("resources/objects/Estufa/Estufa_LSVC.obj");
Model sink("resources/objects/Lavabo/Sink_LSVC.obj");
Model silla("resources/objects/Silla/silla.obj");
Model gabinete("resources/objects/Gabinetes/gabinete.obj");
Model gabinete2("resources/objects/Gabinetes/gabinete2.obj");
Model gabinete3("resources/objects/Gabinetes/gabinete3.obj");
Model broken("resources/objects/Glass/broken.obj");
Model bottle("resources/objects/Glass/bottle.obj");
Model agua("resources/objects/Glass/water.obj");
Model mari("resources/objects/Mariposa/mariposa.obj");

Model mesanueva("resources/objects/MesaNueva/mesanueva.obj");
Model refriNuevo("resources/objects/RefriNuevo/refriNuevo.obj");
Model platos("resources/objects/Platos/platos.obj");
Model micro("resources/objects/Micro/micro.obj");
Model dvd("resources/objects/DVD/dvd.obj");
Model tv("resources/objects/TV/tv.obj");
Model mtv("resources/objects/MTV/mtv.obj");
/*---------------- MODELOS BAÑO ----------------*/
Model ducha("resources/objects/Bathroom/ducha.obj");
Model espejo("resources/objects/Bathroom/espejo.obj");
Model lavabo("resources/objects/Bathroom/lavabo.obj");
Model papel("resources/objects/Bathroom/papel.obj");
Model paperholder("resources/objects/Bathroom/paperholder.obj");
Model tina("resources/objects/Bathroom/tina.obj");
```

```
KeyFrame[0].movAuto_x = -150;
KeyFrame[0].movAuto_y = 0;
KeyFrame[0].movAuto_z = 0;
KeyFrame[0].giroAuto_y = 0;

KeyFrame[1].movAuto_x = 310;
KeyFrame[1].movAuto_y = 40;
KeyFrame[1].movAuto_z = 0;
KeyFrame[1].giroAuto_y = 0;

KeyFrame[2].movAuto_x = 850;
KeyFrame[2].movAuto_y = 0;
KeyFrame[2].movAuto_z = 0;
KeyFrame[2].giroAuto_y = 0;

KeyFrame[3].movAuto_x = 850;
KeyFrame[3].movAuto_y = 0;
KeyFrame[3].movAuto_z = 0;
KeyFrame[3].giroAuto_y = 90;

KeyFrame[4].movAuto_x = 850;
KeyFrame[4].movAuto_y = 0;
KeyFrame[4].movAuto_z = -480;
KeyFrame[4].giroAuto_y = 90;
```

```
// render
// ------
glClearColor(0.3f, 0.3f, 0.3f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// don't forget to enable shader before setting uniforms
staticShader.use();
//Setup Advanced Lights
staticShader.setVec3("viewPos", camera.Position);
staticShader.setVec3("dirLight.direction", lightDirection);
staticShader.setVec3("dirLight.ambient", glm::vec3(0.2f, 0.2f, 0.2f));
staticShader.setVec3("dirLight.diffuse", glm::vec3(1.0f, 1.0f, 1.0f));
staticShader.setVec3("dirLight.specular", glm::vec3(0.0f, 0.0f, 0.0f));

staticShader.setVec3("pointLight[0].position", lightPosition);
staticShader.setVec3("pointLight[0].ambient", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setVec3("pointLight[0].diffuse", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setVec3("pointLight[0].specular", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setFloat("pointLight[0].constant", 0.8f);
staticShader.setFloat("pointLight[0].linear", 0.009f);
staticShader.setFloat("pointLight[0].quadratic", 0.032f);

staticShader.setVec3("pointLight[1].position", glm::vec3(0.0, 0.0f, 0.0f));
staticShader.setVec3("pointLight[1].ambient", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setVec3("pointLight[1].diffuse", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setVec3("pointLight[1].specular", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setFloat("pointLight[1].constant", 1.0f);
staticShader.setFloat("pointLight[1].linear", 0.009f);
staticShader.setFloat("pointLight[1].quadratic", 0.032f);
```

```
// Escenario
// -------------------------------------------------------------------------------------------
//staticShader.use();
staticShader.setMat4("projection", projection);
staticShader.setMat4("view", view);

/*---------------- EXTERIOR ----------------*/
// PASTO
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, -0.5f, 0.0f));
model = glm::scale(model, glm::vec3(2.0f, 1.0f, 2.0f));
staticShader.setMat4("model", model);
pasto.Draw(staticShader);

// BARDA EXTERIOR
model = glm::translate(glm::mat4(1.0f), glm::vec3(-200.0f, 0.5f, 0.0f));
model = glm::scale(model, glm::vec3(1000.0f, 1000.0f, 1000.0f));
staticShader.setMat4("model", model);
barda_exterior.Draw(staticShader);

model = glm::translate(glm::mat4(1.0f), glm::vec3(-100.0f, 0.5f, 0.0f));
model = glm::scale(model, glm::vec3(1000.0f, 1000.0f, 1000.0f));
staticShader.setMat4("model", model);
barda_exterior.Draw(staticShader);

model = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 0.5f, 0.0f));
model = glm::scale(model, glm::vec3(1000.0f, 1000.0f, 1000.0f));
staticShader.setMat4("model", model);
barda_exterior.Draw(staticShader);
```

```
model = glm::translate(glm::mat4(1.0f), glm::vec3(-150.0f, 0.0f, 600.0f));
tmp = model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
staticShader.setMat4("model", model);
arbol_cafe.Draw(staticShader);

model = glm::translate(tmp, glm::vec3(350.0f, 0.0f, -50.0f));
staticShader.setMat4("model", model);
arbol_cafe.Draw(staticShader);


//PAVIMENTO
model = glm::translate(glm::mat4(1.0f), glm::vec3(450.0f, 0.09f, 220.0f));
tmp = model = glm::scale(model, glm::vec3(2.0f, 0.01f, 5.0f));
staticShader.setMat4("model", model);
pavimento.Draw(staticShader);

tmp = model = glm::translate(tmp, glm::vec3(0.0f, 0.0f, 38.0f));
staticShader.setMat4("model", model);
pavimento.Draw(staticShader);

model = glm::translate(tmp, glm::vec3(0.0f, 0.0f, 38.0f));
staticShader.setMat4("model", model);
pavimento.Draw(staticShader);

model = glm::rotate(glm::mat4(1.0f), glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::translate(model, glm::vec3(-743.0f, 0.09f, 410.0f));
tmp = model = glm::scale(model, glm::vec3(2.0f, 0.01f, 5.0f));
staticShader.setMat4("model", model);
pavimento.Draw(staticShader);

tmp = model = glm::translate(tmp, glm::vec3(0.0f, 0.0f, -38.0f));
staticShader.setMat4("model", model);
pavimento.Draw(staticShader);
```

Función my_input()

Esta función nos sirve para poder activar las animaciones de los modelos, dependiendo de la tecla presionada en el teclado.

```
void my_input(GLFWwindow *window, int key, int scancode, int action, int mode)
{
    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);
    //Camera movement
    if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS)
        camera.ProcessKeyboard(FORWARD, (float)deltaTime);
    if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS)
        camera.ProcessKeyboard(BACKWARD, (float)deltaTime);
    if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS)
        camera.ProcessKeyboard(LEFT, (float)deltaTime);
    if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS)
        camera.ProcessKeyboard(RIGHT, (float)deltaTime);
    //To Configure Model

    //Charger
    if (glfwGetKey(window, GLFW_KEY_G) == GLFW_PRESS)
        movAuto_x +=10;
    if (glfwGetKey(window, GLFW_KEY_H) == GLFW_PRESS)
        movAuto_x -=10;
    if (glfwGetKey(window, GLFW_KEY_J) == GLFW_PRESS)
        movAuto_y += 10;
```

```
//Botella
if (glfwGetKey(window, GLFW_KEY_M) == GLFW_PRESS) {
    activeAnim = true;
    initBottle++;
    initBottleY--;
    rotateBottle++;
}
//Mariposa
if (glfwGetKey(window, GLFW_KEY_N) == GLFW_PRESS) {
    activeAnimM = true;
}



//Car animation
if (key == GLFW_KEY_SPACE && action == GLFW_PRESS)
    animacion ^= true;

// Sound
if (glfwGetKey(window, GLFW_KEY_0) == GLFW_PRESS)
    soundon = false;

//Banco de Jaimie guardado
if (key == GLFW_KEY_1 && action == GLFW_PRESS)
{
    animacionBJ ^= true;
    recorridoBJ1 = true;
}

//Banco de Jaimie de regreso a la cuna
```

```
if (key == GLFW_KEY_2 && action == GLFW_PRESS)
{
    animacionBJR ^= true;
    recorridoBJR1 = true;
}

//Mover muebles del cuarto de malcolm
if (key == GLFW_KEY_3 && action == GLFW_PRESS)
{
    animacionCM ^= true;
    recorridoM1 = true;
}

//Mover muebles del cuarto de malcolm
if (key == GLFW_KEY_4 && action == GLFW_PRESS)
{
    animacionCMR ^= true;
    recorridoCR1 = true;
}

//To play KeyFrame animation
if (key == GLFW_KEY_P && action == GLFW_PRESS)
{
    if (play == false && (FrameIndex > 1))
    {
        std::cout << "Play animation" << std::endl;
        resetElements();
        //First Interpolation
        interpolation();

        play = true;
        playIndex = 0;
        i_curr_steps = 0;
```