Universidad Nacional Autónoma de México

Facultad de Ingeniería

“Proyecto Final – Modelo Casa
Manual Técnico”

Asignatura: COMPUTACIÓN GRÁFICA

Grupo: 4

Profesor: ING. CARLOS ALDAIR ROMAN BALBUENA

Alumno:

García Quezada Cristian Gabriel

Semestre: 2022-1
Fecha de entrega:09/12/2021

Objetivo

El alumno deberá aplicar y demostrar los conocimientos adquiridos durante todo el curso.

Descripción

El alumno deberá seleccionar una fachada y un espacio que pueden ser reales o ficticios, y debe estar compuesto por dos cuartos, además presentar imágenes de referencia de dichos espacios para su recreación 3D en OpenGL.

En la imagen de referencia se deben visualizar objetos que el alumno va a recrear virtualmente y donde dichos objetos deben ser lo más parecido a su imagen de referencia, así como su ambientación.

Alcance del proyecto

Para el proyecto se espera tardarse un periodo de 4 o 5 semanas, ya que la primera se dedicará totalmente a elegir lo que se modelará y aprender un poco de maya, desde un inicio se sabe que será complejo y no se aprenderá todo lo que se puede hacer en Maya, además no poder hacerlo perfectamente ya que es algo que requiere practica constante, además que más seguramente habrá una semana en donde no se realizaran cambios ni se trabajara en el proyecto.

Como resultado se espera una casa similar a la de la propuesta, aunque claro se harán algunas modificaciones, después de todo solo es de referencia, para la cantidad de objetos modelados se crearan unos pocos más de los solicitados, con el fin de que se asemeje más a una casa real.

Limitantes

La principal limitante es maya, ya que hace falta practica para poder sacarle todo el provecho posible, en eso va incluido el modelado y el texturizado adecuado, también incluido que algunas veces desde mi punto de vista ya esta bien pero puede que no del todo. Otra limitante podría ser en las animaciones complejas ya que no sé exactamente como podrían hacerse, pero intentare que salgan lo mejor posible.

En caso de que se desee descargar el proyecto completo de github y ejecutar el visual, por razones desconocidas se modifica el tipo de arquitectura a x64, por lo que se debe poner en x86 (esta modificacion se encuentra al lado de Depurar)

****No es limitante en sí, pero se debe mencionar que debido a la cantidad y peso de los modelos e imágenes tarda un poco en cargar la ventana****

Propuesta

La casa que se usara de referencia es la que se ve a continuación, la cual fue sacada directo de internet, además debido a su forma se decidió implementar dos cuartos para aprovechar el espacio, los cuales también se dejan a continuación e igualmente fueron sacados de internet.

Casa parte exterior

dormitorio

sala

Cabe mencionar que en ningún lado de las imágenes se ve una puerta por lo que se pondrá en la parte posterior de la casa, además de otra pequeña entrada que haga la conexión entre los dos cuartos.

Los objetos que se decidió modelar son los siguientes :

Dormitorio

- cama

- librero

-estantería

-lampara (esta se modificará para que se vea diferente)

-sillón

- banco

-*** con posibilidad a los cuadros****

Sala

- lampará (pero se usará el modelo anterior)

-sillón pequeño (pero se usará el modelo anterior)

-sillón grande (este se hará con diseño similar y textura que el anterior)

-florero

-repisa

-estantería

-otra repisa aunque lo más seguro es que el diseño se cambie un poco

-tapete

*Además se crearán otros modelos como una mesa y sillas, pero de esos no hay referencia así que serán vistos en el ejecutable

Documentación

Primeramente se muestran los encabezados que se usaran en el programa para usar las librerías

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

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

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

// Other Libs
#include "stb_image.h"

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

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


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

Librerías propias de C++

-#include <iostream>

-#include <math>

Librerías para usar openGl

-#include <GL/glew.h>

-#include <GLFW/glfw3.h>

-#include <glm/glm.hpp>

-#include <glm/gtc/matrix_transform.hpp>

-#include <glm/gtc/type_ptr.hpp>

Librería para poder cargar los modelos desde maya

-#include "SOIL2/SOIL2.h"

Otras librerías que ocuparemos para insertar imágenes, para usar la cámara, los skybox, los modelos y texturas

-#include "stb_image.h"

-#include "Shader.h"

-#include "Camera.h"

-#include "Model.h"

-#include "Texture.h"

Posteriormente se configura el tamaño de la pantalla que será de 800*600, esto se indica en la sección “Window dimensions”

Además de algunas funciones que se usaran, incluida la cámara y variables que ocuparemos en el proyecto para la animacion.

```
// Function prototypes
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode);
void MouseCallback(GLFWwindow* window, double xPos, double yPos);
void DoMovement();

// Window dimensions
const GLuint WIDTH = 800, HEIGHT = 600;
int SCREEN_WIDTH, SCREEN_HEIGHT;

// Camera
Camera  camera(glm::vec3(-2.0f, 2.0f, -15.0f));//-50.0f, 1.0f, -50.0f));camx,camy,
GLfloat lastX = WIDTH / 2.0;
GLfloat lastY = HEIGHT / 2.0;
bool keys[1024];
bool firstMouse = true;
float range = 0.0f;
float spotAngle = 0.0f;
// Light attributes
glm::vec3 lightPos(0.0f, 0.0f, 0.0f);
bool active;
```

```
//Variables de animacion

bool anim = false;          ->control sobre animacion
float posicion = 0.0;       ->llevando a cero la posición cundo se vaya a ocupar
float rot = 0.0f;           ->se ocupará para la rotación iniciándola en 0°
float movx = 0.0f;          ->para controlar el movimiento de x, se iniciará en 0
float movz = 0.0f;          ->para controlar el movimiento de Z, se iniciará en 0
float rot1 = 90.0f;         -> se ocupará para la rotación iniciándola en 90°
float movsilla_x = 0.0f;    ->para controlar el desplazamiento en x de la silla, se
                            iniciará en 0
bool tec = false;           ->control sobre animacion de la silla
float movluz_y = 0.0f;      ->para controlar el desplazamiento en y de la lampara1,
                            se iniciará en 0
bool luz = false;           ->control sobre animacion de la lampara1
float movluz_z = 0.0f;      ->para controlar el desplazamiento en z de la lampara2,
                            se iniciará en 0
bool luz1 = false;          ->control sobre animacion de la lampara2
float movsab_y = 0.0f;      ->para controlar el desplazamiento en y de la sabana, se
                            iniciará en 0
bool sab = false;           ->control sobre animacion de la sabana
float movtech = 0.0f;       ->para controlar el desplazamiento en x del techo, se
                            iniciará en 0
bool tech = false;          ->control sobre animacion del techo
```

Los siguientes 3 arreglos servirán para indicar las coordenadas de origen de la luz, tanto para pointlight, como para spotlight

```
// Positions of the point lights
glm::vec3 pointLightPositions[] = {
       glm::vec3(0.0f,0.0f,0.0f),
       glm::vec3(0.0f,0.0f,0.0f),
       glm::vec3(0.0f,0.0f,0.0f),
       glm::vec3(0.0f,0.0f,0.0f)
};

//posicion spotlight
glm::vec3 spotLightPositions = { glm::vec3(0.0f,0.0f,0.0f) };

//direccion spotlight
glm::vec3 spotLightDirections = { glm::vec3(0.0f,0.0f,0.0f) };

glm::vec3 Light1 = glm::vec3(0);
```

```
// Deltatime
GLfloat deltaTime = 0.0f;  // Estas dos variables son para controlar la
GLfloat lastFrame = 0.0f;  // velocidad con la que iniciara la cámara
```

A continuación se muestra la función por medio de la cual se crea la ventana del proyecto, donde se mostrará, además de indicar que nombre recibirá, en este caso se llamara “proyecto”

```
// Create a GLFWwindow object that we can use for GLFW's functions
GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "Proyecto", nullptr, nullptr);//Iluminacion 2

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

    return EXIT_FAILURE;
}

glfwMakeContextCurrent(window);

glfwGetFramebufferSize(window, &SCREEN_WIDTH, &SCREEN_HEIGHT);
```

Con esta sección de código habilitaremos para poder mover la vista de la cámara conforme movamos el ratón, y que de ese modo sea mas cómodo para el usuario

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

Los shaders que se ocuparan, tanto para el control de la luz como también para el skybox que ocuparemos, para los fondos

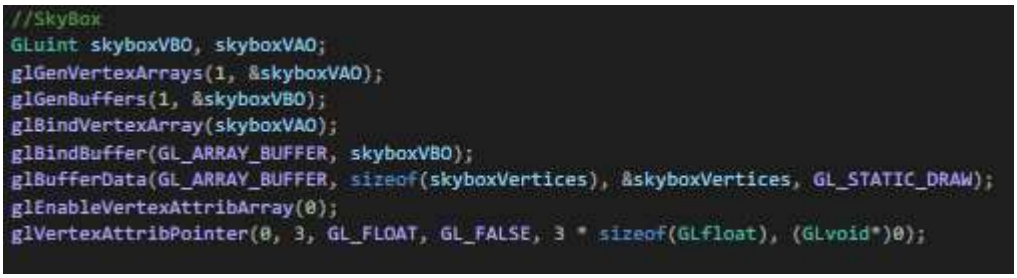
```
Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting.frag");
Shader lampShader("Shaders/lamp.vs", "Shaders/lamp.frag");
Shader SkyBoxshader("Shaders/SkyBox.vs", "Shaders/SkyBox.frag");
```

En cuanto a los modelos empleados para ser visualizados en visual son los siguiente, a la mayoría no hay necesidad de comentarlos por que su nombre indica tal cual que es lo que son, pero algunos como algunos no es del todo claro pondré un comentarios indicando que representa

```
Model Piso((char*)"Models/Esfera/Piso.obj");    ->hace referencia al pasto sobre el
                                                que se encuentra montada la casa
Model sabana((char*)"models/cama/sabana.obj"); ->sabana de la cama
Model cama((char*)"models/cama/cama.obj");      ->colchon de la cama
Model base((char*)"models/cama/base.obj");      ->base de la cama
Model lampara((char*)"models/lampara/lampara.obj");    ->lampara, tanto cuarto como
                                                       sala
Model librero((char*)"models/librero/librero.obj");->el que se encuentra en cuarto
Model mueble((char*)"models/mueble/mueble.obj");->el que se encuentra en el cuarto
Model banco((char*)"models/banco/banco.obj");   ->el que se encuentra en el cuarto
Model repisa((char*)"models/repisa/repisa.obj"); ->se encuentra en sala
Model sillonc((char*)"models/sillonc/sillonc.obj");    ->sillon pequeño de la sala y
                                                       cuarto
Model sillong((char*)"models/sillong/sillong.obj"); ->sillon grande de la sala
```

```
Model casa((char*)"models/cas/casa.obj");       ->cimientos de la casa
Model techo((char*)"models/techo/techo.obj");   ->techo montado de la casa
Model mesa((char*)"models/mesa/mesa.obj");      ->se encuentra en la sala
Model silla((char*)"models/silla/silla.obj");   -> se encuentra en la sala
Model silla2((char*)"models/silla2/silla2.obj"); -> se encuentra en la sala
Model arbol((char*)"models/arbol/arbol.obj");     ->esta afuera de la casa, se repite
                                                  3 veces
Model florero((char*)"models/florero/florero.obj");    ->hay 2, uno en el cuarto y
                                                       otro en la sala
Model estante((char*)"models/estante/estante.obj");    ->esta en la sala al lado del
                                                       sillon grande
Model tapete((char*)"models/tapete/tapete.obj");       ->esta en la sala
Model maceta((char*)"models/maceta/maceta.obj");       ->se encuentra junto a un
                                                       árbol debajo de la ventana
Model puerta((char*)"models/puerta/puerta.obj");       ->en la entrada de la casa
Model cuadro1((char*)"models/cuadro1/cuadro1.obj");    ->los 4 cuadros se
Model cuadro2((char*)"models/cuadro2/cuadro2.obj");    ->encuentran
Model cuadro3((char*)"models/cuadro3/cuadro3.obj");    ->en el
Model cuadro4((char*)"models/cuadro4/cuadro4.obj");    ->cuarto
Model libre((char*)"models/libre/libre.obj");          ->librero de la sala
Model botella((char*)"models/botella/botella.obj");    ->botella dentro de estante de
                                                       la sala
```

Seguido a lo anterior el VBO y el VAO para los skybox, este controlara como debe funcionar



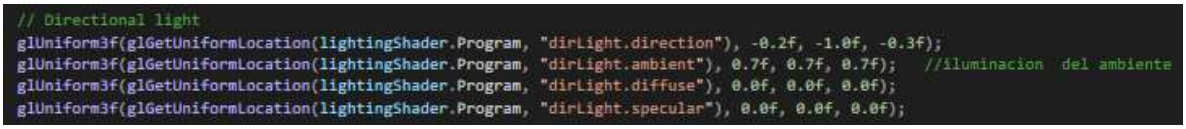
```
//SkyBox
GLuint skyboxVBO, skyboxVAO;
glGenVertexArrays(1, &skyboxVAO);
glGenBuffers(1, &skyboxVBO);
glBindVertexArray(skyboxVAO);
glBindBuffer(GL_ARRAY_BUFFER, skyboxVBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(skyboxVertices), &skyboxVertices, GL_STATIC_DRAW);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0,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

así mismo como la llamada a sus caras para texturizar **el skybox elegido fue principalmente porque se buscó varios pero todos se pixeleaban mucho, por lo que se veía mal, así que se decidió usar el mostrado más adelante **

```
// Load textures
vector<const GLchar*> faces;
faces.push_back("SkyBox/left.jpg");      ->cara izquierda
faces.push_back("SkyBox/right.jpg");     ->cara derecha
faces.push_back("SkyBox/top.jpg");       ->cara superior
faces.push_back("SkyBox/bottom.jpg");    ->cara inferior
faces.push_back("SkyBox/back.jpg");      ->cara trasera
faces.push_back("SkyBox/front.jpg");     ->cara frontal

GLuint cubemapTexture = TextureLoading::LoadCubemap(faces);  ->cargado de caras
```

Ahora se mostrara la sección de código donde muestra la direccional light desglosada en todas sus componentes



```
// Directional light
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.7f, 0.7f, 0.7f);   //iluminacion  del ambiente
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.0f, 0.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.0f, 0.0f, 0.0f);
```

Sección de código donde muestra el point light desglosada en todas sus componentes, además de las variables con las que se moverá de lugar, las cuales fueron declaradas anteriormente. *Cabe mencionar que hay 4 point light, pero solo se pondrá uno para ejemplificar*

```
// Point light 1
glm::vec3 lightColor;
lightColor.x = abs(sin(glfwGetTime() * Light1.x));
lightColor.y = abs(sin(glfwGetTime() * Light1.y));
lightColor.z = sin(glfwGetTime() * Light1.z);


glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"), pointLightPositions[0].x, pointLightPositions[0].y, pointLightPositions[0].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"), 0.0f, 0.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), 0.0f, 0.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"), 0.0f, 0.0f, 0.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].linear"), 0.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"), 0.0f);
```

Sección de código donde muestra el spot light, tando de posición como de dirección y si desglose de todas sus componentes, además de las variables con las que se moverá de lugar, las cuales fueron declaradas anteriormente

```
// SpotLight
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.position"), spotLightPositions.x, spotLightPositions.y, spotLightPositions.z);// camera.GetPosition().x
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.direction"), spotLightDirections.x, spotLightDirections.y + movluz_y, spotLightDirections.z + movluz_z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.ambient"), 0.4f, 0.4f, 0.4f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.diffuse"), 0.3f, 0.3f, 0.3f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.specular"), 0.5f, 0.5f, 0.5f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.linear"), 0.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.quadratic"), 0.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.cutOff"), glm::cos(glm::radians(10.0f)));
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.outerCutOff"), glm::cos(glm::radians(11.0f)));
```

Posteriormente se debe mencionar la configuracion de los modelos dentro de visual, ya que no solo basta con declararlos, sino también mandarlos a llamar y acomodarlos en la posición correcta, además de ajustar su tamaño. Pero solo se pondrá un ejemplo ya que son varios los modelos que se crearon.

Como podemos observar se muestra con todas las transformaciones a realizar para dicho modelo, las cuales consistirán en traslación, escala y rotación.

```
/////cama
1.- model = glm::mat4(1);
2.- model = glm::rotate(model, glm::radians(rot), glm::vec3(0.0f, 0.0f, 90.1f));
3.- model = glm::scale(model, glm::vec3(1.7f, 1.7f, 1.5f));
4.- model = glm::translate(model, glm::vec3(8.0f, 0.7f, -7.5f));
5.- glUniform1f(glGetUniformLocation(lightingShader.Program, "activatransparencia"),
0);
6.- glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
7.- cama.Draw(lightingShader);
```

1.- sirve para reiniciar la matriz y llevarla a condiciones iniciales, ya que de lo contrario se quedara con la ultima configuracion introducida, esto es recomendable sobre todo para poder llevar un mejor control sobre los objetos, aunque claro también en algunos casos resulta practico quedarse con la configuracion anterior y solo realizarle pequeño cambio, un ejemplo de ello es como se manejo para colocar la base de la cama, ya que se dejo la configuracion de cama (es el colchón) para solo mover ligeramente la base.

2.- indicaremos la rotación que deberá tener el objeto, para ello nos ayudaremos de la variable “rot” y a seguido indicaremos que tanto debe moverse y en qué dirección.

3.-La escala que tendrá el objeto, indicando hacia que eje debe hacerse más grande o más pequeño.

4.-Para indicar la posición que tendrá el objeto, tanto en x,y,z, ya que no solo basta con colocarlo, sino darle una ubicación para que tenga sentido.

5 y 6 .- son propios para que se pueda importar y mostrar adecuadamente

7.- mando a llamar al objeto para que se visualice

Ahora se muestran los controles de la cámara que servirán para controlar la visualización

```
// Camera controls
if (keys[GLFW_KEY_W] || keys[GLFW_KEY_UP])
{
    camera.ProcessKeyboard(FORWARD, deltaTime);
}

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

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

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

-el primer “ if “ servirá para que podamos acercar la camara hacia adelante con la tecla w

-el segundo “ if “ servirá para que podamos alejar la camara hacia atras con la tecla s

-el tercer “ if “ servirá para que podamos mover la camara hacia la izquierda con la tecla a

-el cuarto “ if “ servirá para que podamos mover la camara hacia la derecha con la tecla d

Posteriormente la parte del código donde se controla la posición de la point light

```
if (keys[GLFW_KEY_T])             ->con T se mueve en x positivo
{
	pointLightPositions[0].x += 0.01f;
}
if (keys[GLFW_KEY_G])             ->con G se mueve en x negativo
{
	pointLightPositions[0].x -= 0.01f;
}
if (keys[GLFW_KEY_Y])             ->con Y se mueve en y positivo
{
	pointLightPositions[0].y += 0.01f;
}
if (keys[GLFW_KEY_H])             ->con H se mueve en y negativo
{
	pointLightPositions[0].y -= 0.01f;
}
```

```
if (keys[GLFW_KEY_U])             ->con U se mueve en z positivo
{
	pointLightPositions[0].z += 0.1f;
}
if (keys[GLFW_KEY_J])             ->con J se mueve en z negativo
{
	pointLightPositions[0].z -= 0.01f;
}
```

Ahora la parte del código donde se controla la posición de la spot light

```
//Spothlight posicion
if (keys[GLFW_KEY_Y])             ->con Y se mueve en x positivo
{
	spotLightPositions.x += 0.01f;
}
if (keys[GLFW_KEY_H])             ->con H se mueve en x negativo
{
	spotLightPositions.x -= 0.01f;
}
if (keys[GLFW_KEY_T])             ->con T se mueve en y positivo
{
	spotLightPositions.y += 0.01f;
}
if (keys[GLFW_KEY_G])             ->con G se mueve en y negativo
{
	spotLightPositions.y -= 0.01f;
}
if (keys[GLFW_KEY_U])             ->con U se mueve en z positivo
{
	spotLightPositions.z += 0.01f;
}
if (keys[GLFW_KEY_J])             ->con J se mueve en y negativo
{
	spotLightPositions.z -= 0.01f;
}
```

Ahora la parte del código donde se controla la dirección de la spot light

```
//Spotlight direccion
if (keys[GLFW_KEY_E])             ->con E se mueve en dirección x positivo
{
	spotLightDirections.x += 0.1f;
}
if (keys[GLFW_KEY_D])             ->con D se mueve en dirección x negativo
{
	spotLightDirections.x -= 0.1f;
}
if (keys[GLFW_KEY_R])             ->con R se mueve en dirección y positivo
{
	spotLightDirections.y += 0.1f;
}
```

```
if (keys[GLFW_KEY_F])             ->con F se mueve en dirección y negativo
{
       spotLightDirections.y -= 0.1f;
}
if (keys[GLFW_KEY_N])             ->con N se mueve en dirección z positivo
{
       spotLightDirections.z += 0.1f;
}
if (keys[GLFW_KEY_M])             ->con M se mueve en dirección z negativo
{
       spotLightDirections.z -= 0.1f;
}
```

La siguiente función sirve para poder salir de la ventana, que sera presionando la tecla “ esc (escape)”, ademas de mantener el contenido en el tamaño de la ventana

```
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;
        }
    }
```

Ahora se mostrara la función que permite mover la cámara con respecto al movimiento del mouse

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

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

    lastX = xPos;
    lastY = yPos;

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

Finalmente esta la parte de las animaciones, en donde se comentara de que manera se realizaron, pero debido a que la manera es prácticamente la misma se pondrá un ejemplo para mostrarlo.

Por el lado de animacion sencilla se tiene el movimiento de una silla, el cual se hará de la siguiente forma

-primeramente en la parte de DoMovement, que es donde indicará el comportamiento que tendrá la animacion

```
//////////////////////animacion silla//////////////////////////////
if (tec) {          ->recibe la variable y cuando lo hace realiza lo siguiente
	if (movsilla_x >= -3.0f)   ->si la variable movsilla_x es mayor o igual a -
	                           3.0f entonces
	{
		movsilla_x = movsilla_x - 0.1f;   -> la variable movsilla_x se ira
		                                     decrementando en 0.1f
	}
}
else {                     ->si la variable tec no tiene el valor deseado entonces
	if (movsilla_x <= 0.0f)    ->si la variable movsilla_x es menor o igual a
	                           -0.0f  entonces
	{
		movsilla_x = movsilla_x + 0.1f;   -> la variable movsilla_x se ira
		                                     incrementando en 0.1f
	}
}
```

-mas abajo en la parte de keycallback, que es donde se corroborara que se quiera iniciar la animacion

```
//silla sala

if (keys[GLFW_KEY_X])      ->comprueba que se haya presionado la tecla X, para dar
                           inicio a la animacion
{
	if (!tec)              ->verifica que el estado de tec sea false
		tec = true;    ->cambia el estado de tec a true para indicar que comenzó
		               la animacion
	else               ->caso contrario y que tec sea true
		tec = false;   ->se modifica tec para valer false indicando que se
		               termino la animacion
}
```

-Y finalmente para llevarla a cabo, se indica en que parte del código se realizaran las modificaciones

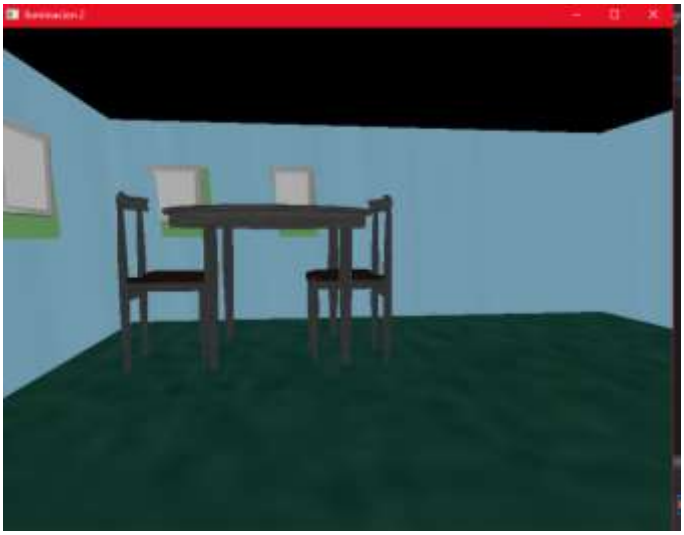
```
model = glm::mat4(1);                                                                    /////silla 2
model = glm::translate(model, glm::vec3(-5.0f + movsilla_x , 0.75f, 9.0f));
model = glm::scale(model, glm::vec3(0.88f, 0.88f, 0.88f));
//glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "activatransparencia"), 0);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
```

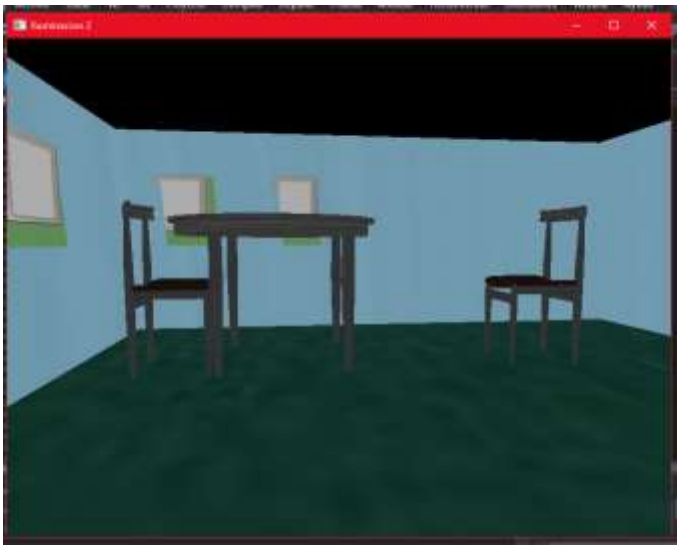
En este caso al ser la silla el ejemplo, nos vamos a donde dimos instrucciones para visualizar la silla, y en la parte donde ponemos su posición X agregamos que se le sume el valor de la variable, esto hará que se mueva.

Para las demás animaciones es algo similar, así que solo se mostrara este ejemplo, y para ver las otras es necesario entrar al código fuente.

Una muestra de lo realizado

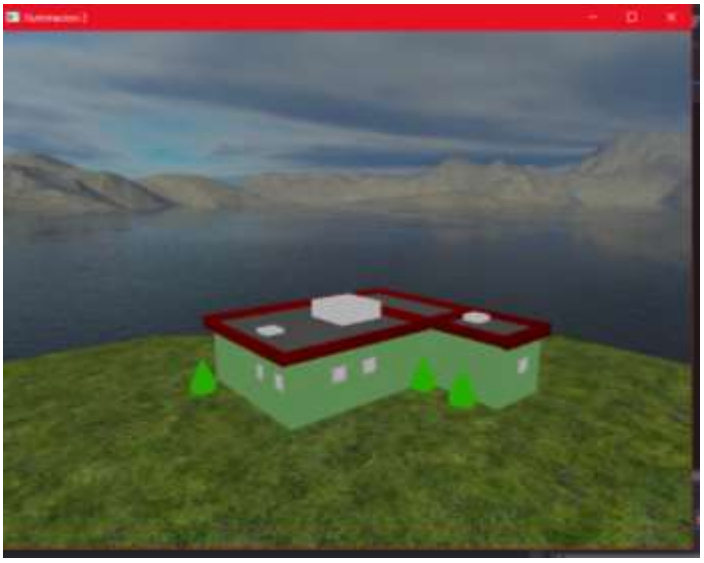
Para llevarlo a cabo es necesario presionar la tecla X para moverla, y después se puede volver a presionarla para regresar a su lugar original, esto simulara que alguien abre la silla para sentarse.


estado original

 silla movida de lugar

*Movimiento de la silla al presionar X *

** Se deja imagen de vista de la casa por fuera**



Muestra de la casa, vista por fuera además de fondo un skybox

Otras animaciones sencillas que se crearon fueron los movimientos de las luces, para la del cuarto es necesario presionar la tecla “Z”; para el movimiento de la luz de la lampara de la sala es con “C”; para el movimiento de la sabana de la cama es con “V”

Costos

Fases de proyecto	costos	precio venta
diseño	400	800
modelado	1000	3200
programacion	900	2100
pruebas	200	500
total	2500	6600

Partiendo de un costo de $150*hora y de un costo de luz de 1.051 $/kWh y un iva de 0.16%, los costos resultantes son los siguientes

Tiempo	
Horas	55
precio x hora	120
Total	6600
Servicios	
luz	57.805
Total	6657.805
iva	1065.2488
Total + iva	7723.0538

precio a cobrar para realizar el proyecto

Diagrama de Gantt

Finalmente se dejan los enlaces de descarga:

Liga github: https://github.com/kurohaven1999/Proyecto_Grafica_gpo4.git

Liga drive: https://drive.google.com/file/d/1zqIPH80EMvw6jFShOo3qgpA1m2YOfgJy/view?usp=sharing