Universidad Nacional Autónoma de México

Facultad de Ingeniería

División de Ingeniería Eléctrica (DIE)

Materia: Lab. De Computación Gráfica e Interacción Humano-Computadora

Semestre: 2022-2

Grupo: 12

Proyecto final: Manual técnico.

Alumno: Flores Licea Lars Alain

Fecha de entrega: 12 de mayo de 2022

Prof: Ing. Carlos Aldair Román Balbuena.

No. de cuenta: 417086611
Grupo: 12

Índice.

No. de cuenta: 417086611
Grupo: 12

Objetivos:

- El primer objetivo de este proyecto sin duda fue experimentar, aprender, conocer e incluso divertirme a lo largo de la construcción del proyecto.
- Familiarizarnos con herramientas de modelado en 3D como lo es: Maya, y de edición de imágenes como lo es: GIMP. Para la construcción de modelos y su posterior texturización.
- Poner en práctica los conceptos aprendidos a lo largo de las sesiones de laboratorio: transformaciones básicas, manejo de cámara, carga y dibujado de modelos, iluminación, animación sencilla y compleja, etc.

Diagrama de Gantt:

Para tener un control de los avances de nuestro proyecto se elaboró un sencillo Diagrama de Gantt.

Diagrama de Gannt									
Tiempo de duración									
	Fecha de inicio: 04/04/2022	Abril				Mayo			
Actividad	Fecha de entrega: 12/05/2022	1	2	3	4	1	2	3	4
Imagen de referencia	04/04/2022								
Modelo 1	15/04/2022								
Modelo 2	15/04/2022								
Modelo 3	22/04/2022								
Modelo 4	22/04/2022								
Modelo 5	29/04/2022								
Modelo 6	29/04/2022								
Modelo 7	06/05/2022								
Cuarto	07/05/2022								
Fachada	07/05/2022								
SkyBox	08/05/2022								
Iluminación	08/05/2022								
Animación	11/05/2022								
Creación de ejecutable	12/05/2022								
Creación de Manuales	12/05/20222								

En el proyecto, hubo ciertas peculiaridades, en lo personal me parece que fue un error ir haciendo modelo a modelo cada semana, en lugar de realizar la fachada y el cuarto lo más rápido posible, eso me quitó mucho tiempo, pues trataba de ser lo más minicuoso posible con mis modelos y cuando menos lo pensé; tenía el tiempo encima.

Alcance del proyecto.

Este proyecto tiene la finalidad de cubrir los rubros básicos vistos en las sesiones de laboratorio, desde la inclusión de múltiples modelos los cuales usando transformaciones básicas: traslación, rotación y escalamiento, se encuentren bien situados en el escenario dando así el mayor parecido posible con la imagen de referencia. Implementación de los tres tipos de iluminación: spotlight,

pointlight y directional light, y un efecto llamativo en el que las pointlights oscilen de un color a otro, y la inclusión de animaciones sencillas principalmente rotaciones y algunas complejas dando un efecto sumamente llamativo a algunos modelos. No se incluyeron animaciones de tipo circuito, o inclusive más complejas que tengan que ver con fenómenos físicos tipo: caída libre, tiro parábolico, etc, o la inclusión de texturas con transparencia o materiales del tipo Phong, que pueden mostrar otra imagen cuando les de la luz.

Limitantes.

En lo personal considero que fue mi poca planeación en el proyecto, sinceramente pensaba tener más tiempo pero entre el trabajo y la escuela todo se vino encima, es algo que lamento ya que tengo este reto de querer implementar animaciones complejas que tengan que ver con alguna ecuación matemática que describa cierto fenómeno físico, el haber usado más texturas con transparencias y así habilitar el canal alfa dando efectos más vistosos a mi proyecto.

Documentación del código.

El código tiene en total 1522 líneas de código, por lo que no explicaremos línea a línea sino abarcaremos los rubros más importantes: carga y dibujado de modelos, iluminación, y animación.

Carga de modelos.

```
230     Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting.frag");
231     Shader lampShader("Shaders/lamp.vs", "Shaders/lamp.frag");
232     Shader SkyBoxshader("Shaders/SkyBox.vs", "Shaders/SkyBox.frag");
233     Shader Anim("Shaders/anim.vs", "Shaders/anim.frag");
234     Shader Anim2("Shaders/anim2.vs", "Shaders/anim2.frag");
235     Shader Anim3("Shaders/anim3.vs", "Shaders/anim2.frag");
236     Shader Anim4("Shaders/anim4.vs", "Shaders/anim2.frag");
237     // Carga de modelos
238
239     Model desk((char*)"Models/MLars/escritoriolars.obj");
240     Model cajon((char*)"Models/MLars/cajon.obj");
241     Model espada1((char*)"Models/MLarsEspada/espada1lars.obj");
242     Model espada2((char*)"Models/MLarsEspada/espada2lars.obj");
243     Model espada3((char*)"Models/MLarsEspada/espada3lars.obj");
244     Model arco((char*)"Models/MLarsEspada/arco.obj");
245     Model escudo1((char*)"Models/MLarsEscudo/escudo1.obj");
246     Model sofa((char*)"Models/MLarsSofa/sofa.obj");
247     Model reloj((char*)"Models/MLarsReloj/reloj.obj");
248     Model pendulo((char*)"Models/MLarsReloj/pendulo.obj");
249     Model piso((char*)"Models/MLarsPiso/piso.obj");
250     Model pisoint((char*)"Models/MLarsPiso/pisoint.obj");
251     Model camino((char*)"Models/MLarsPiso/camino.obj");
252     Model chimenea((char*)"Models/MLarsChimenea/chimenea.obj");
253     Model cuenco((char*)"Models/MLarsChimenea/cuenco.obj");
254     Model flama((char*)"Models/MLarsChimenea/flama.obj");
255     Model letras((char*)"Models/MLarsLetras/letras.obj");
256     Model Room1((char*)"Models/MLarsCuartoint/Kitroom1.obj");
```

Los modelos se cargan en la función main (), en la clase Model se añaden todos los modelos a utilizar se les asigna una variable local, la cual puede ser llamada según tu elección, y se usa un apuntador indicando la ruta en la que se encuentra el modelo, importantísimo; el modelo tiene que estar en formato obj. Una vez cargados los modelos, procedemos a dibujarlos.

Dibujado de modelos.

```
//Carga de modelo
//Personaje
view = camera.GetViewMatrix();
glm::mat4 model(1);
tmp = model = glm::translate(model, glm::vec3(0, 1, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::scale(model, glm::vec3(2.75f, 0.0f, 2.0f));
model = glm::translate(model, glm::vec3(0.0f, -5.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
piso.Draw(lightingShader);
glBindVertexArray(0);

model = glm::mat4(1);
tmp = model = glm::translate(model, glm::vec3(0, 1, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0, 0.0f));
model = glm::scale(model, glm::vec3(0.70f, 1.0f, 1.0f));
model = glm::translate(model, glm::vec3(-27.25f, 0.1f, 7.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
camino.Draw(lightingShader);
glBindVertexArray(0);
```

El primer modelo a cargar debe de declarar el uso de la matriz de vista y modelo, se setea la matriz de modelo siempre, y el usar tmp nos sitúa en el centro del escenario, se pueden hacer las transformaciones que sean pero siempre al final hay que avisarle a la matriz de modelo los cambios que se hicieron ya sea: una rotación, traslación o escalamiento. Dibujamos el modelo con el nombre de variable que hayamos elegido e importantísimo la función Draw recibe como parámetro un Shader, en nuestro caso será el lightingShader. Replicamos lo mismo con cada modelo a excepción de algunos modelos.

Iluminación.

```
// Directional light
//Creación de la luz direccional
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.5f, 0.5f, 0.5f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.1f, 0.1f, 0.1f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 1.0f, 1.0f, 1.0f);
```

La luz direccional es una luz que afecta a todo es escenario y está compuesta por las componente: ambiental, difusa y especular.

```
// Point light 0
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"), lightColor.x, lightColor.y, lightColor.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), lightColor.x, lightColor.y, lightColor.z);
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"), 0.07f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"), 1.8f);
```

La pointlight tiene una variable especial llamada lightColor, esa variable nos permitirá dar el efecto de que la luz varíe su color. Los puntos de luz reciben la posición, la componente del ambiente,

difusa, especular y los valores constante, lineal y cuadrático. Según cambien estos valores la luz puede ser más tenue o intensa, abarcar un mayor rango, etc.

```
// SpotLight
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.position"), camera.GetPosition().x, camera.GetPosition().y, camera.GetPosition().z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.direction"), camera.GetFront().x, camera.GetFront().y, camera.GetFront().z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.ambient"), 1.0f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.diffuse"), 1.0f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.specular"), 2.0f, 2.0f, 2.0f);
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)));
```

La spotlight es similar a la pointlight sólo que esta limitida por un ángulo en específico, los dos primeras son importantes ya que van a dar la impresión de tenemos una linterna, es decir conforme nos movamos la spotlight también se moverá. Recibe casi los mismos parámetros que la pointlight sólo tiene dos nuevos: cutOff y outerCutOff .

Animaciones.

Rotación de la puerta.

```
model = glm::mat4(1);
tmp = model = glm::translate(model, glm::vec3(0, 1, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::translate(model, glm::vec3(7.35f, 0.085f, -1.35f));
model = glm::rotate(model, glm::radians(rot), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Puerta.Draw(lightingShader);
glBindVertexArray(0);
```

Dibujamos, la puerta que rotará ciertos radianes (rot) sobre el eje x, es sumamente importante en este tipo de animaciones colocar muy bien el pivote antes de exportar el modelo.

Para la parte de animación, necesitamos una variable rot de tipo float, que será el valor de rotación de la puerta, se inicia en 0.0f, y la segunda variable de tipo bool, me permitirá manejar la animación.

```
float rot = 0.0f;
bool abierto = false;
```

```
if (keys[GLFW_KEY_O])
{
    abierto = !abierto;
}
if (abierto) {
    if (rot < 30.0f)
        rot += deltaTime * 25;
}
else {
    if (rot > 0.0f)
        rot -= deltaTime * 25;
}
```

Esa parte es la más importante, cada vez que se presiona la tecla O, la puerta girará hasta 30 grados sobre el eje x, en un tiempo deltaTime * 25, es decir, lo hará 25 veces más rápido. Cuando alcance ese grado de rotación, entonces al volver a presionar la tecla O la puerta regresará a su posición original, la variable abierto me permite conocer cuando llegó a ese límite.

Rotación del cuadro.

```
model = glm::mat4(1);
tmp = model = glm::translate(model, glm::vec3(0, 1, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::scale(model, glm::vec3(2.0f, 2.0f, 1.0f));
model = glm::translate(model, glm::vec3(0.4f, 1.5f, -13.75f));
model = glm::rotate(model, glm::radians(rot1), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Cuadro.Draw(lightingShader);
glBindVertexArray(0);
```

Es muy similar a la animación de la puerta, Dibujamos, el cuadro que rotará ciertos radianes (rot) sobre el eje x, es sumamente importante en este tipo de animaciones colocar muy bien el pivote antes de exportar el modelo.

Para la parte de animación, necesitamos una variable rot de tipo float, que será el valor de rotación de la puerta, se inicia en 0.0f, y la segunda variable de tipo bool, me permitirá manejar la animación.

```
float rot = 0.0f;
float rot1 = 0.0f;

bool abierto = false;
bool abierto1 = false;
```

```
if (keys[GLFW_KEY_P])
{
    abierto1 = !abierto1;
}
if (abierto1) {
    if (rot1 < 60.0f)
        rot1 += deltaTime * 25;
}
else {
    if (rot1 > 0.0f)
        rot1 -= deltaTime * 25;

}
```

Esa parte es la más importante, cada vez que se presiona la tecla P, la puerta girará hasta 30 grados sobre el eje x, en un tiempo deltaTime * 25, es decir, lo hará 25 veces más rápido. Cuando alcance ese grado de rotación, entonces al volver a presionar la tecla P la puerta regresará a su posición original, la variable abierto me permite conocer cuando llegó a ese límite. Esta animación esconde una gran sorpresa.

No. de cuenta: 417086611
Grupo: 12

Animaciones complejas.

Flama.

```
//Cargando el Shader de Anim
Anim.Use();
tiempo = glfwGetTime();
modelLoc = glGetUniformLocation(Anim.Program, "model");
viewLoc = glGetUniformLocation(Anim.Program, "view");
projLoc = glGetUniformLocation(Anim.Program, "projection");
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
model = glm::mat4(1);
tmp = model = glm::translate(model, glm::vec3(0, 1, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::translate(model, glm::vec3(7.5f, 0.5f, -9.5f));
model = glm::scale(model, glm::vec3(2.5f, 4.0f, 2.5f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(Anim.Program, "time"), tiempo);
flama.Draw(Anim);
```

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoords;

const float amplitude = 0.7;
const float frequency = 3.0;
const float PI = 3.14159;
out vec2 TexCoords;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;
uniform float time;

void main()
{
  float distance = length(aPos);
  float effect = amplitude*cos(-PI*distance*frequency+time);
  gl_Position = projection*view*model*vec4(aPos.x,aPos.y, aPos.z,1);
  TexCoords=vec2(aTexCoords.x+effect,aTexCoords.y+effect);

}
```

Esta animación es compleja y la hacemos sobre el propio shader, dándonos un efecto sobre la textura del modelo, lo que simula que hay una flama. El efecto está dado por una ecuación y la animación al ser sobre el propio Shader es automática no hay necesidad de estar presionando una tecla, ya que se ejecuta en cada ciclo de reloj.

Péndulo.

```
Anim2.Use();
tiempo = glfwGetTime();
modelLoc = glGetUniformLocation(Anim.Program, "model");
viewLoc = glGetUniformLocation(Anim.Program, "view");
projLoc = glGetUniformLocation(Anim.Program, "projection");
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
model = glm::mat4(1);
tmp = model = glm::translate(model, glm::vec3(0, 1, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::translate(model, glm::vec3(-4.0f, 0.1f, -0.05f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(Anim.Program, "time"), tiempo);
pendulo.Draw(Anim2);
glBindVertexArray(0);
```

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoords;

const float amplitude = 0.25;
const float frequency = 1.0;
const float PI = 3.14159;
out vec2 TexCoords;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;
uniform float time;

void main()
{
  float distance = length(aPos);
  float effect = amplitude*cos(time);
  gl_Position = projection*view*model*vec4(aPos.x+effect,aPos.y, aPos.z,1); //modelo
  TexCoords=vec2(aTexCoords.x,aTexCoords.y); //Textura

}
```

Esta animación es compleja y la hacemos sobre el propio shader, dándonos un efecto sobre los vértices del modelo, lo que simula un movimiento sobre el eje x del péndulo. El efecto está dado por una ecuación y la animación al ser sobre el propio Shader es automática no hay necesidad de estar presionando una tecla, ya que se ejecuta en cada ciclo de reloj.

No. de cuenta: 417086611
Grupo: 12

Cajón.

```
Anim3.Use();
tiempo = glfwGetTime();
modelLoc = glGetUniformLocation(Anim.Program, "model");
viewLoc = glGetUniformLocation(Anim.Program, "view");
projLoc = glGetUniformLocation(Anim.Program, "projection");
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
model = glm::mat4(1);
tmp = model = glm::translate(model, glm::vec3(0, 1, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::scale(model, glm::vec3(2.5f, 2.0f, 2.5f));
model = glm::rotate(model, glm::radians(0.0f), glm::vec3(0.0f, 1.0, 0.0f));
model = glm::translate(model, glm::vec3(2.50f, 0.1f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(Anim.Program, "time"), tiempo);
cajon.Draw(Anim3);
glBindVertexArray(0);
```

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoords;

const float amplitude = 0.3;
const float frequency = 1.0;
const float PI = 3.14159;
out vec2 TexCoords;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;
uniform float time;

void main()
{
  float distance = length(aPos);
  float effect = amplitude*cos(time);
  gl_Position = projection*view*model*vec4(aPos.x+effect,aPos.y, aPos.z,1); //modelo
  TexCoords=vec2(aTexCoords.x,aTexCoords.y); //Textura

}
```

Esta animación es compleja y la hacemos sobre el propio shader, dándonos un efecto sobre los vértices del modelo, lo que simula un movimiento sobre el eje x del cajón. El efecto está dado por una ecuación y la animación al ser sobre el propio Shader es automática no hay necesidad de estar presionando una tecla, ya que se ejecuta en cada ciclo de reloj. Esta animación esconde un secreto.

Espada y arco.

```
Anim4.Use();
tiempo = glfwGetTime();
modelLoc = glGetUniformLocation(Anim.Program, "model");
viewLoc = glGetUniformLocation(Anim.Program, "view");
projLoc = glGetUniformLocation(Anim.Program, "projection");
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
model = glm::mat4(1);
tmp = model = glm::translate(model, glm::vec3(0, 1, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
/*model = glm::translate(model, PosIni + glm::vec3(movKitX, movKitY, 0));*/
model = glm::translate(model, glm::vec3(14.0f, 0.2f, -2.5f));
model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(Anim4.Program, "time"), tiempo);
espada3.Draw(Anim4);
model = glm::mat4(1);
tmp = model = glm::translate(model, glm::vec3(0, 1, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::scale(model, glm::vec3(1.5f, 1.5f, 1.5f));
model = glm::translate(model, glm::vec3(-1.5f, 2.5f, -8.530f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
arco.Draw(Anim4);
glBindVertexArray(0);
```

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoords;

const float amplitude = 0.7;
const float frequency = 3.0;
const float PI = 3.14159;
out vec2 TexCoords;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;
uniform float time;

void main()
{
  float distance = length(aPos);
  float effect = amplitude*cos(-PI*distance*frequency+time);
  gl_Position = projection*view*model*vec4(aPos.x,aPos.y, aPos.z,1);
  TexCoords=vec2(aTexCoords.x,aTexCoords.y+effect);

}
```

Esta animación es compleja y la hacemos sobre el propio shader, dándonos un efecto sobre la textura del modelo, lo que simula una espada mágica al igual que el arco. El efecto está dado por una ecuación y la animación al ser sobre el propio Shader es automática no hay necesidad de estar presionando una tecla, ya que se ejecuta en cada ciclo de reloj.

No. de cuenta: 417086611
Grupo: 12

Luces de color.

```
if (keys[GLFW_KEY_SPACE])
{
    active = !active;
    if (active)
    {
        Light1 = glm::vec3(1.0f, 0.0f, 1.0f);
        Light2 = glm::vec3(1.0f, 0.0f, 1.0f);
        Light3 = glm::vec3(1.0f, 0.0f, 1.0f);
        Light4 = glm::vec3(1.0f, 0.0f, 1.0f);
        Light5 = glm::vec3(1.0f, 0.0f, 1.0f);
        Light6 = glm::vec3(1.0f, 0.0f, 1.0f);
        Light7 = glm::vec3(1.0f, 0.0f, 0.0f);
        Light8 = glm::vec3(1.0f, 1.0f, 1.0f);

    }
    else
    {
        Light1 = glm::vec3(0);//Cuando es solo un valor en los 3 vectores pueden dejar solo una componente
        Light2 = glm::vec3(0);//Cuando es solo un valor en los 3 vectores pueden dejar solo una componente
        Light3 = glm::vec3(0);//Cuando es solo un valor en los 3 vectores pueden dejar solo una componente
        Light4 = glm::vec3(0);//Cuando es solo un valor en los 3 vectores pueden dejar solo una componente
        Light5 = glm::vec3(0);//Cuando es solo un valor en los 3 vectores pueden dejar solo una componente
        Light6 = glm::vec3(0);//Cuando es solo un valor en los 3 vectores pueden dejar solo una componente
        Light7 = glm::vec3(0);//Cuando es solo un valor en los 3 vectores pueden dejar solo una componente
        Light8 = glm::vec3(0);//Cuando es solo un valor en los 3 vectores pueden dejar solo una componente
    }
}
```

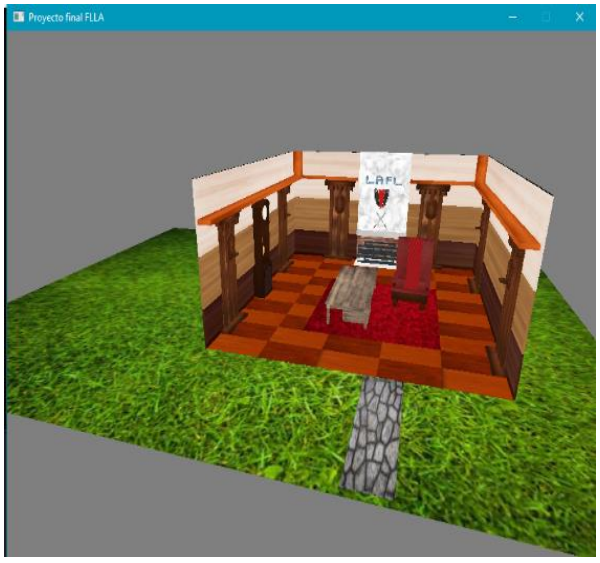
```
// Positions of the point lights
glm::vec3 pointLightPositions[] = {
    glm::vec3(-97.5f,7.55f, -38.65f),
    glm::vec3(-97.5f,7.55f, -52.65f),
    glm::vec3(-77.90f,7.55f,  -38.65f),
    glm::vec3(-77.90f,7.55f, -52.65f),
    glm::vec3(-82.15f, 7.55f, -56.9f),
    glm::vec3(-92.85f, 7.55f, -56.9f),
    glm::vec3(-87.45f,3.35f,-56.9f),
    glm::vec3(-87.65f,3.30f,-47.75f),

};
```

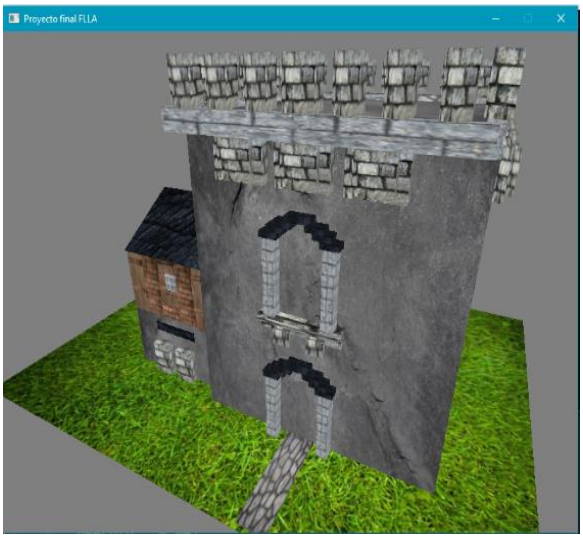
Los pointlights los ubicamos en puntos específicos de nuestro escenario, y a partir de la variable lightcolor oscilamos de un color a otro cuando presionamos la tecla espacio se activa el efecto, cuando volvemos a presionar detenemos el efecto. Es un efecto sumamente vistoso y le da buena presentación a nuestro escenario.

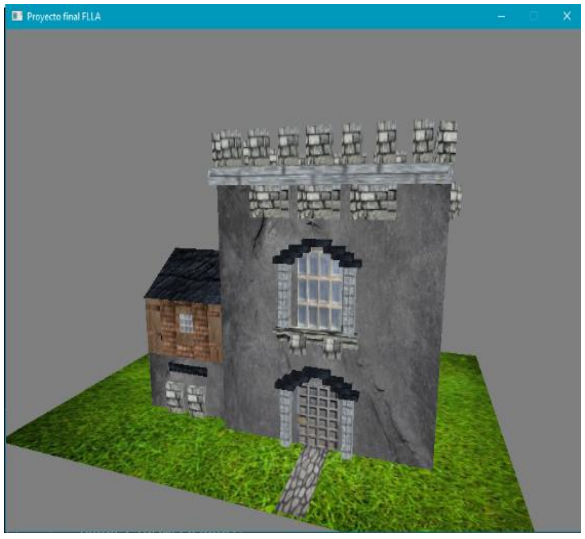
Imágenes.

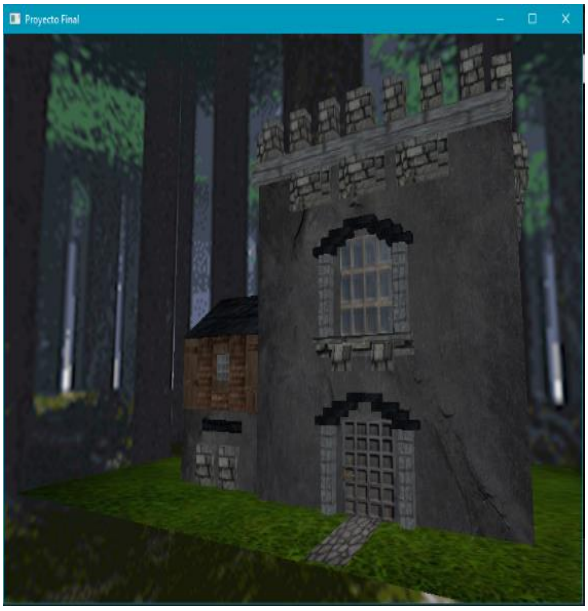
¡Este es el resultado final! :,) Con fotos de todo el proceso.



No. de cuenta: 417086611
Grupo: 12





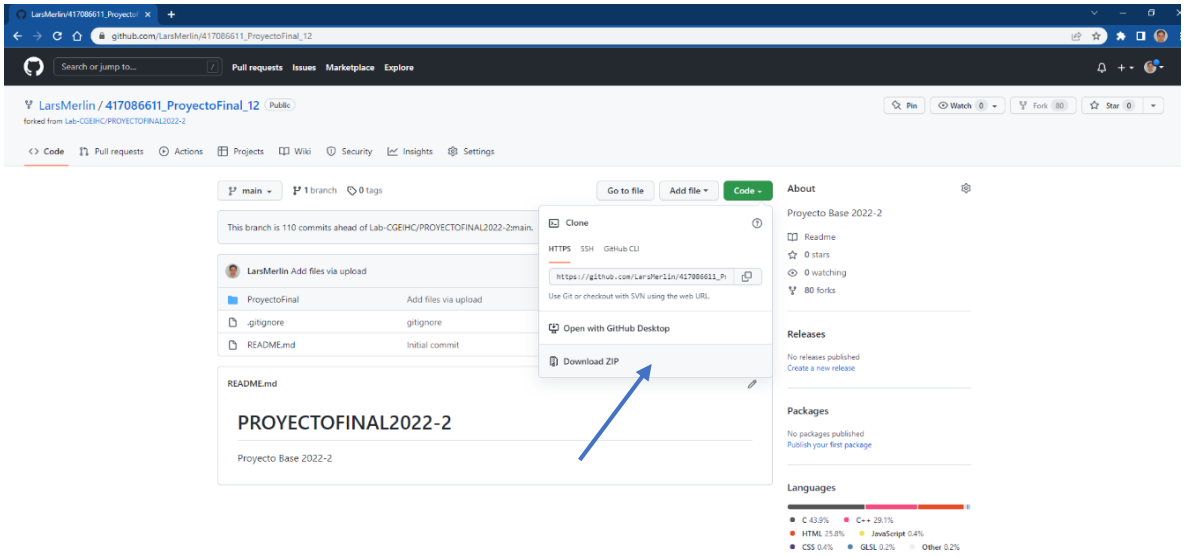


No. de cuenta: 417086611
Grupo: 12

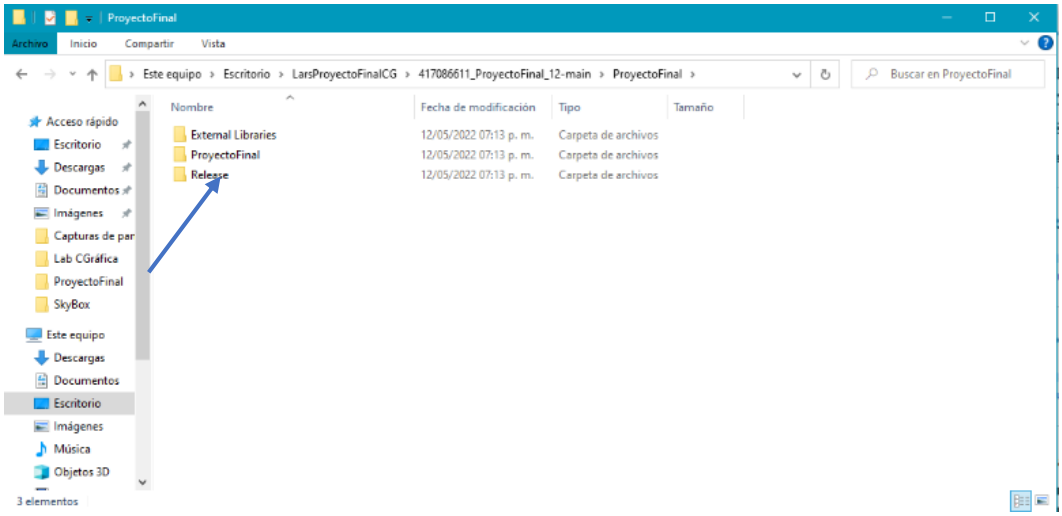
Manual de usuario.

Este es el manual de usuario de mi proyecto final de Laboratorio de Computación Gráfica e Interacción Humano-Computadora. Para ver e interactuar con el proyecto, sigue los siguientes pasos:

1. Descarga el proyecto desde el repositorio de GitHub.
   https://github.com/LarsMerlin/417086611_ProyectoFinal_12.git

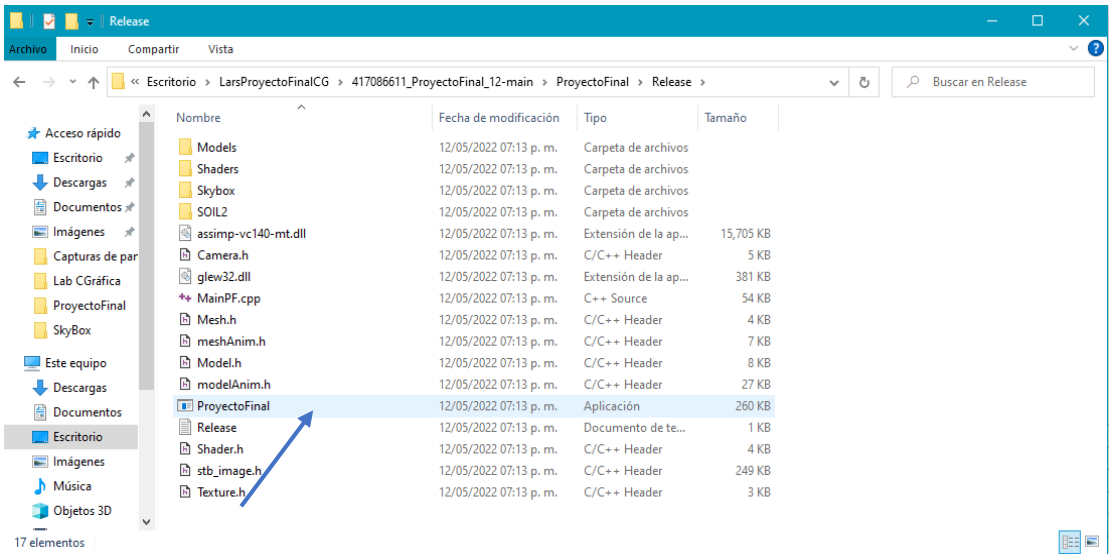


2. Ya una vez descargado el proyecto, descomprimelo y guárdalo.
3. Verás una carpeta llamada Release, da click en ella.



4. En Release verás un archivo que se llama ProyectoFinal.exe, da click y espera un poco.

No. de cuenta: 417086611
Grupo: 12



5. Ya una vez dentro del proyecto, te puedes mover con el mouse para mover la dirección de la cámara y con las teclas W, A, S y D te puedes mover de la siguiete manera:
   - W te mueves hacia arriba. ↑
   - S te mueves hacia abajo. ↓
   - A te mueves hacia la izquierda. ←
   - D te mueves hacia la derecha. →
6. Con la tecla O se abre la puerta y con la misma tecla O puedes cerrarla.
7. Con la tecla P se mueve el cuadro y con la misma tecla P puedes regresar a su posición original.
8. Con la tecla SPACE o la barra espaciadora se activan las luces con la misma tecla desactivas las luces.
9. Con la tecla ESC puedes cerrar el proyecto.

Conclusión.

Este proyecto representó todo un reto, tiempo, esfuerzo, asombro, en algunos momentos fue tedioso, pero a pesar de todo me siento feliz y a la vez con un sentimiento agridulce de querer tener más tiempo para conocer más, para aprender más, disfrute mucho la materia y me siento un poco triste por el fin, pero estoy agradecido con el profesor y mis compañeros por el hecho de que pude revivir ese sueño alejado por ciertas situaciones de la vida; de querer crear videojuegos.

De verdad, no lo creo, veo la imagen de referencia y que el proyecto se asemeje un poco me hace sentir alegría, yo incluí cosas que me gustaban a mí y no necesariamente estaban en la imagen pero confiaba en que darían un buen aspecto y así fue. Cada práctica fue sorprenderme, fue sumamente gratificante el poder resolver los ejercicios, cuando por primera vez vimos modelado o lo de iluminación y sombreado, me sentí emocionado, pensaba que ese mundo estaba muy distante, que era muy difícil y al pasar de las prácticas no fue así. También tengo que agradecer la paciencia y el empeño que tiene el profesor por la materia, sin ello no hubiese podido resolver muchas dudas que tuve a lo largo del curso y durante el proyecto.

No. de cuenta: 417086611
Grupo: 12

Es un proyecto básico, cumple con lo establecido, lamentablemente por el tiempo y demás circunstancias no es tan pulcro quisiera, pero tiene una gran margen de mejora como: en la creación de los materiales, que ya no sean de tipo Lambert y sean de tipo Phong, jugar más con las luces, incluir personajes y animarlos por KeyFrames, incluir animaciones más complejas que permitan interactuar a los personajes con el escenario, incluir música o sonidos, cambiar el teclado por algún control de PlayStation o Xbox, etc.