Universidad Nacional
Autónoma de México

Facultad de Ingeniería

División de Ingeniería Eléctrica

Cómputo Gráfico e Interacción Humano Computadora

Lara Sala Kevin Arturo

Proyecto final. Manual técnico.

Profesor: Arturo Pérez de la Cruz, Ing.

Grupo: 1
Fecha: 8 de Diciembre de 2021
Semestre 2022-1

Objetivo.

El objetivo de este documento es dar una explicación un tanto a fondo de las técnicas, modelos, y demás componentes utilizados para llevar a cabo este proyecto.

Introducción.

La computación gráfica es un área de la Ingeniería en Computación que ha venido teniendo un empuje muy grande en los últimos años, a pesar de existir desde que empezaron a existir las computadoras personales. Hoy en día, la computación gráfica está presente en aplicaciones de realidad virtual, realidad aumentada, desarrollo de videojuegos o incluso en área del día a día, por ejemplo, en cualquier situación en la que se usen dispositivos de despliegue y se haga uso de cálculos y métodos para mostrar información de manera entendible y llamativa al usuario.

En este proyecto se aplicaron todos los conceptos vistos durante el semestre y se explican a detalle en el desarrollo de este documento.

Desarrollo.

Se dividirá el documento en diferentes secciones. En primer lugar se hará un breve resumen de los modelos que se utilizaron. Posteriormente se platicarán de las técnicas que animación utilizadas y finalmente, se explicarán detalles relacionados al código y su estructura. Para acceder al proyecto, dar click en el siguiente link que lo llevará al repositorio en Github: https://github.com/KevinLaraSala/Proyecto_CGeIHC.git

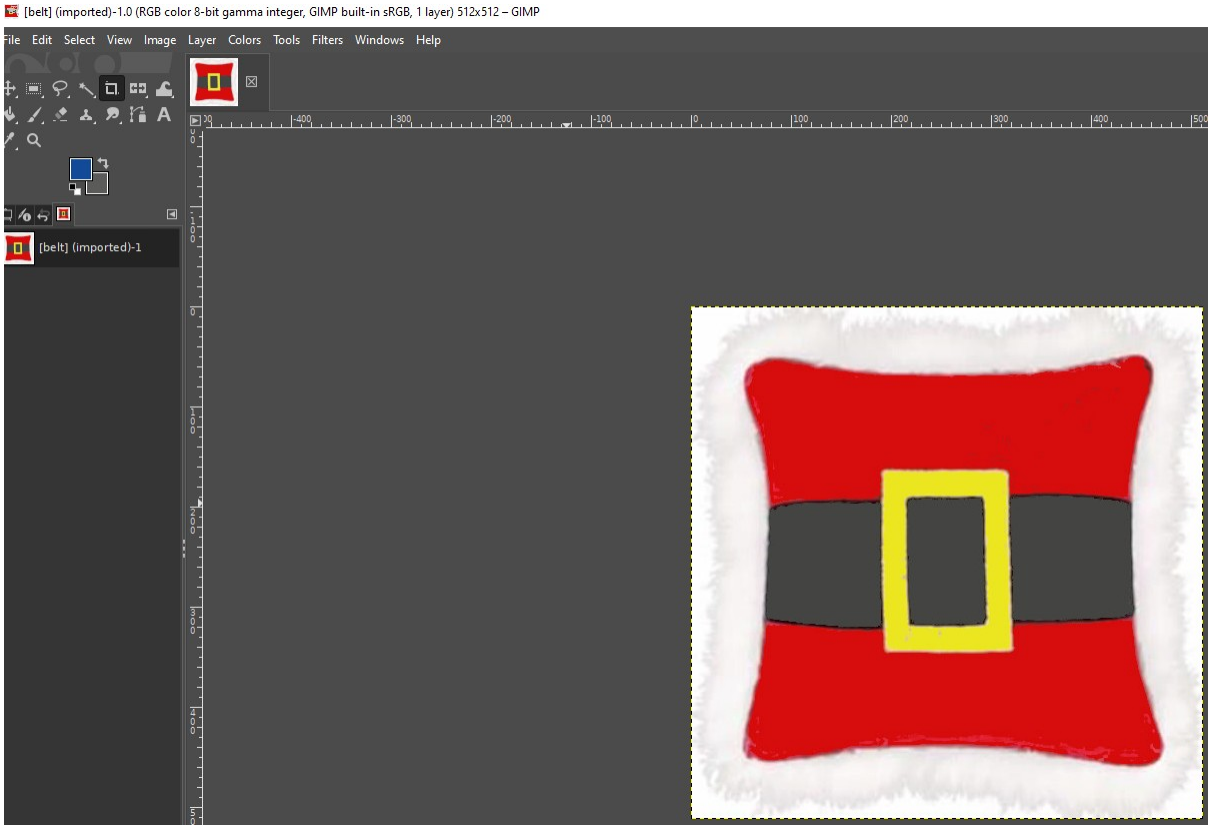
1. Modelado.

```
Model Trees((char*)"Models/Trees/trees.obj");
Model House((char*)"Models/House/house.obj");
Model LivingRoom((char*)"Models/LivingRoom/livingRoom.obj");
Model LivingRoom2((char*)"Models/LivingRoom/livingRoom2.obj");
Model Kitchen((char*)"Models/Kitchen/kitchen.obj");
Model Bathroom((char*)"Models/Bathroom/bathroom.obj");
Model Room((char*)"Models/Room/room.obj");
Model Presents((char*)"Models/Presents/presents.obj");
Model Present((char*)"Models/Presents/present1.obj");
Model Present2((char*)"Models/Presents/present2.obj");
Model Windows((char*)"Models/Windows/windows.obj");
Model DWindow((char*)"Models/Windows/doorWindow.obj");
Model Door((char*)"Models/Windows/door.obj");
Model MailBox((char*)"Models/Mailbox/mailbox.obj");
Model Sled((char*)"Models/Sled/sled.obj");
Model hSanta((char*)"Models/Santa/headSanta.obj");
Model bSanta((char*)"Models/Santa/bodySanta.obj");
Model raSanta((char*)"Models/Santa/rightArmSanta.obj");
Model laSanta((char*)"Models/Santa/leftArmSanta.obj");
Model rlSanta((char*)"Models/Santa/rightLegSanta.obj");
Model llSanta((char*)"Models/Santa/leftLegSanta.obj");
Model Sign((char*)"Models/Sign/sign.obj");
Model Bag((char*)"Models/Bag/sac.obj");
Model body((char*)"Models/Lego/body.obj");
Model head((char*)"Models/Lego/head.obj");
Model RightArm((char*)"Models/Lego/rightArm.obj");
Model LeftArm((char*)"Models/Lego/leftArm.obj");
Model RightLeg((char*)"Models/Lego/rightLeg.obj");
Model LeftLeg((char*)"Models/Lego/leftLeg.obj");
```

Se muestran todos los modelos utilizados en el proyecto. Se acomodaron en la carpeta “Models” dentro de la carpeta de la solución de Visual Studio, acomodados por concepto. Como es de esperarse, algunos modelos se separaron en componentes individuales para su correcta animación.

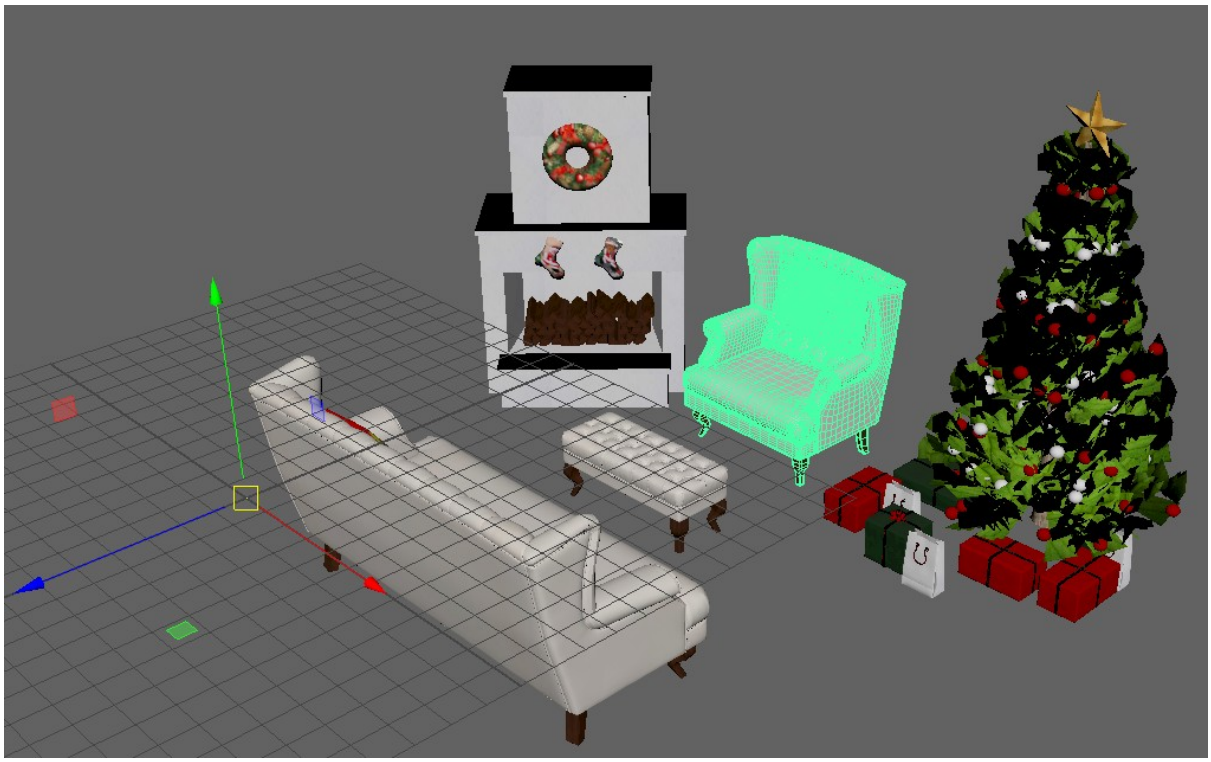
Algunos modelos de obtuvieron de internet, de la página de CGTrader, y algunos otros modelos (los más sencillos) se realizaron por el alumno en el software de Maya.
Por ejemplo, el modelo de la casa fue completamente modificado para cumplir las necesidades del proyecto, al que el personaje de Santa Claus fue modificado para poder llevar a cabo las animaciones. Algunos modelos más complejos como los sillones o la cama, se obtuvieron de internet.

Todos los modelos fueron texturizados por el alumno. Muchas de las texturas se crearon desde cero utilizando el software libre “GIMP”, y las demás se tuvieron que modificar para que OpenGL pudiera ser capaz de mostrarlas de manera correcta. Las texturas que no eran colores sólidos, se escalaron a un tamaño de 512x512, es decir, siempre siguiendo el principio de potencias de 2, y tamaños cuadrados. Por ejemplo:



Se muestra una imagen de tamaño 512x512 que será la textura de un cojín de la sala.
Las imágenes se manejaron en formato png para poder añadir canales alfa en caso de que sea necesarios eliminar secciones de la imagen o evitar tener problemas con el despliegue de los modelos en OpenGL.
Por otro lado, los colores sólidos simplemente se crearon desde cero en GIMP como imágenes PNG de tamaño 32x32 para optimizar espacio y tiempos de carga.

Ahora se muestra un ejemplo de modelado en Maya.



Se aprecia en la imagen, la manipulación del modelo que se encuentra en la sala, ya texturizado de la manera más parecida posible a la propuesta de proyecto.
En este modelo, se usaron algunos modelos individuales obtenidos de internet y simplemente se adecuaron a la forma que se muestra en la captura, para que el escenario tenga sentido.
Los modelos se tuvieron que trabajar de manera que los archivos no sobrepasaran los 100[MB] para no tener problemas a la hora de cargarlos en la plataforma de Github.

2. Animaciones.

En este proyecto se utilizaron 3 animaciones principales.

- Movimiento de Santa Claus.

  El modelo de Santa Claus se programó de manera que inicie en su habitación y camine hasta su saco de regalos, y una vez ahí, aviente el regalo a su saco y así poder partir a repartir.

  Se usó la técnica de variables booleanas para dictar las trayectorias de los modelos. Una vez que se cumplan las condiciones necesarias, el modelo cambiará de dirección o de posición:

  Se muestra una captura a continuación a manera de ejemplo, de las variables utilizadas en la animación de Santa Claus.

  El modelo del regalo y el modelo de Santa Claus van de la mano ya que se mueven juntos. En las variables se aprecia la declaración de variables respectivas al modelo de la puerta. Esto debido a que la puerta “interactúa” con el modelo de Santa Claus, una vez que sale, la puerta se cierra.

  Finalmente, se declara una variable llamada caminarSanta que se usará para la rotación de las piernas, simulando la caminata del modelo. El movimiento de Santa Claus se activa con la tecla F.

```
//Santa
float movSantaX = -5;
float movSantaY = 2;
float movSantaZ = 25;
float rotSanta = 180;
float caminarSanta;
bool trayectorySanta1 = true;
bool trayectorySanta2 = false;
bool trayectorySanta3 = false;
bool trayectorySanta4 = false;
bool trayectorySanta5 = false;
bool santaTrayectory;
float movDoorX = 7;
float movDoorY = -1.2;
float movDoorZ = 32.4;
bool openDoor;
//Regalo
float movRegaloX = movSantaX;
float movRegaloY = movSantaY;
float movRegaloZ = movSantaZ;
bool lanzamientoRegalo = false;
bool caidaRegalo = false;
int x = 0;
```

```
void animacion()
{
    if (play) { ... }
    if (sledTrayectory) { ... }
    if (santaTrayectory)
    {
        if (movDoorX > -2.3 && movSantaZ > 40)
            movDoorX -= 0.1;
        if (trayectorySanta1)
        {
            caminarSanta = sin(glfwGetTime() * 4) * 50;
            if (movSantaZ > 46)
            {
                trayectorySanta1 = false;
                trayectorySanta2 = true;
                movSantaY = 1;
                movRegaloY = movSantaY;
            }
            else
            {
                movSantaZ += 0.1;
                movRegaloZ = movSantaZ;
            }
        }
        if (trayectorySanta2) { ... }
        if (trayectorySanta3) { ... }
        if (trayectorySanta4) { ... }
        if (trayectorySanta5) { ... }
        if (lanzamientoRegalo) { ... }
        if (caidaRegalo) { ... }
    }
}
```

En la misma función de animación, se controla el movimiento de los tres elementos principales. En la parte de play, se maneja el control de KeyFrames, en SledTrayectory se controla el control del trineo y en Santa, el movimiento de SantaClaus.
Se usan 5 trayectorias para el movimiento de SantaClaus, y 2 variables booleanas para el movimiento del regalo cayendo en el saco. En la captura se muestra un ejemplo de la estructura de cada una de las trayectorias, modificando los desplazamientos y las rotaciones, así como el movimiento de las piernas.

- Movimiento de duendes.

```
//Lego 1
float movx = 0;
float movy = 0;
float movz = -130;
//Lego 2
float movLegoX, movLegoY = -7.3, movLegoZ = -40.05;
float rotLego = 90;
float caminar;
```

En este caso, se utilizaron menos variables, sin embargo, aquí es donde se empleó la técnica que animación por KeyFrames. El primer modelo del Lego (duende 1) es el que se mueve de la mano con el trineo, sin embargo, se programó la animación por KeyFrames simulando el movimiento de sus brazos, como si estuviera saludando. Esta animación se activa con la tecla L. Una vez terminada la animación, se puede volver a activar las veces que se quiera, presionando dicha tecla.
El movimiento de los Legos se activa también con la tecla F, al igual que Santa Claus y el trineo.

Adelante se muestran los métodos involucrados en la animación de KeyFrames.

```
// Keyframes
float rightArm1, leftArm1;

#define MAX_FRAMES 9
int i_max_steps = 190;
int i_curr_steps = 0;
typedef struct _frame
{
    float rightArm1;
    float incRightArm1;
    float leftArm1;
    float incLeftArm1;
}FRAME;

FRAME KeyFrame[MAX_FRAMES];
int FrameIndex = 0;
bool play = false;
int playIndex = 0;
```

```
for (int i = 0; i < MAX_FRAMES; i++)
{
    KeyFrame[i].rightArm1 = 0;
    KeyFrame[i].incRightArm1 = 0;
    KeyFrame[i].leftArm1 = 0;
    KeyFrame[i].incLeftArm1 = 0;
}
```

```
void saveFrame(void)
{

    printf("frameindex %d\n", FrameIndex);

    KeyFrame[0].rightArm1 = 0;
    KeyFrame[1].rightArm1 = 45;
    KeyFrame[2].rightArm1 = 0;
    KeyFrame[0].leftArm1 = 0;
    KeyFrame[1].leftArm1 = 45;
    KeyFrame[2].leftArm1 = 0;

    FrameIndex = 3;
}

void resetElements(void)
{
    rightArm1 = KeyFrame[0].rightArm1;
    leftArm1 = KeyFrame[0].leftArm1;
}

void interpolation(void)
{
    KeyFrame[playIndex].incRightArm1 = (KeyFrame[playIndex + 1].rightArm1 - KeyFrame[playIndex].rightArm1) / i_max_steps;
    KeyFrame[playIndex].incLeftArm1 = (KeyFrame[playIndex + 1].leftArm1 - KeyFrame[playIndex].leftArm1) / i_max_steps;
}
```

El funcionamiento básicamente es el siguiente: Se declaran las variables globales y las variables del struct.
Después se inicializan en el ciclo for, y para terminar, se modifican las tres funciones, asignando los valores de posición inicial y final. Se reinician los elementos en la posición inicial. Finalmente, se calcula la interpolación de las posiciones normalizando el cálculo del desplazamiento haciendo uso de las variables auxiliares de incrementos.

Captura de la función de animación en la sección de KeyFrames (play):

```
void animacion()
{
    if (play)
    {
        if (i_curr_steps >= i_max_steps) //end of animation between frames?
        {
            playIndex++;
            if (playIndex >= FrameIndex)    //end of total animation?
            {
                printf("termina anim\n");
                playIndex = 0;
                play = false;
            }
            else //Next frame interpolations
            {
                i_curr_steps = 0; //Reset counter
                                  //Interpolation
                interpolation();
            }
        }
        else
        {
            rightArm1 += KeyFrame[playIndex].incRightArm1*2;
            leftArm1 += KeyFrame[playIndex].incLeftArm1*2;

            i_curr_steps+=2;
        }
    }
    if (sledTrayectory) { ... }
    if (santaTrayectory) { ... }
}
```

- Movimiento del trineo.

Para finalizar, se utiliza el mismo principio del movimiento de Santa Claus y los duendes para el trineo.

Se adjunta captura de la sección de animación del trineo.

```
void animacion()
{
    if (play) { ... }
    if (sledTrayectory)
    {
        if (movy < -20.0f)
            subir = true;
        else if (movy > 12.0f)
            subir = false;
        if (subir)
            movy += 0.1;
        else
            movy -= 0.1;
        caminar = sin(glfwGetTime() * 4) * 70;

        if (trayectory1)
        {
            if (movx == 160.0f)
            {
                trayectory1 = false;
                trayectory2 = true;
                rotSled = -90.0f;
                rotLego = 0.0f;
            }
            else
            {
                movx += 0.5f;
                movLegoX += 0.3f;
            }
        }

        if (trayectory2) { ... }

        if (trayectory3) { ... }

        if (trayectory4) { ... }
    } //Animación trineo.
    if (santaTrayectory) { ... }
}
```

*Nota. Para calcular la rotación de las piernas, se usa la función seno, y se le pasa como argumento la hora del sistema, de manera que pueda alternar entre 1 y -1. Se multiplican por 4 para modificar la amplitud de las piernas y se multiplica por 70 para modificar la velocidad del movimiento.

- Iluminación.

Para la parte de la animación, se separó todo en una función aparte para no saturar al método principal. Se programaron diversas *pointLights* y se dividieron en dos grupos principales: Las luces de la casa y las luces de los árboles exteriores.

```
Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting.frag");
// Use cooresponding shader when setting uniforms/drawing objects
lightingShader.Use();
GLint viewPosLoc = glGetUniformLocation(lightingShader.Program, "viewPos");
glUniform3f(viewPosLoc, camera.GetPosition().x, camera.GetPosition().y, camera.GetPosition().z);
// Set material properties
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 10.0f);
// Directional light
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.4f, 0.4f, 0.4f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.4f, 0.4f, 0.4f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.5f, 0.5f, 0.5f);

/*****************************************************************************************************
*                                          Iluminación casa                                          *
*****************************************************************************************************/
glm::vec3 lightColor1;
lightColor1.x = abs(sin(glfwGetTime() * LightP1.x));
lightColor1.y = abs(sin(glfwGetTime() * LightP1.y));
lightColor1.z = sin(glfwGetTime() * LightP1.z);
glm::vec3 lightColor2;
lightColor2.x = abs(sin(glfwGetTime() * LightP2.x));
lightColor2.y = abs(sin(glfwGetTime() * LightP2.y));
lightColor2.z = sin(glfwGetTime() * LightP2.z);
// Point light 1
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"), houseLights[0].x, houseLights[0].y, houseLights[0].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), lightColor1.x, lightColor1.y, lightColor1.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"), lightColor1.x, lightColor1.y, lightColor1.z);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].linear"), 0.09f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"), 0.032f);
// Point light 2
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].position"), houseLights[1].x, houseLights[1].y, houseLights[1].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].diffuse"), lightColor2.x, lightColor2.y, lightColor2.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].specular"), lightColor2.x, lightColor2.y, lightColor2.z);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].linear"), 0.09f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].quadratic"), 0.032f);
// Point light 3
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].position"), houseLights[2].x, houseLights[2].y, houseLights[2].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].diffuse"), lightColor1.x, lightColor1.y, lightColor1.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].specular"), lightColor1.x, lightColor1.y, lightColor1.z);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].linear"), 0.09f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].quadratic"), 0.032f);
```

```
/*****************************************************************************************************
*                                          Iluminación árboles                                       *
*****************************************************************************************************/
glm::vec3 lightColor3;
lightColor3.x = abs(sin(glfwGetTime() * LightP3.x));
lightColor3.y = abs(sin(glfwGetTime() * LightP3.y));
lightColor3.z = sin(glfwGetTime() * LightP3.z);
// Point light 1
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[12].position"), treeLights[0].x, treeLights[0].y, treeLights[0].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[12].diffuse"), lightColor3.x, lightColor3.y, lightColor3.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[12].specular"), lightColor3.x, lightColor3.y, lightColor3.z);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[12].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[12].linear"), 0.09f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[12].quadratic"), 0.032f);
// Point light 2
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[13].position"), treeLights[1].x, treeLights[1].y, treeLights[1].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[13].diffuse"), lightColor3.x, lightColor3.y, lightColor3.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[13].specular"), lightColor3.x, lightColor3.y, lightColor3.z);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[13].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[13].linear"), 0.09f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[13].quadratic"), 0.032f);
// Point light 3
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[14].position"), treeLights[2].x, treeLights[2].y, treeLights[2].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[14].diffuse"), lightColor3.x, lightColor3.y, lightColor3.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[14].specular"), lightColor3.x, lightColor3.y, lightColor3.z);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[14].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[14].linear"), 0.09f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[14].quadratic"), 0.032f);
// Point light 4
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[15].position"), treeLights[3].x, treeLights[3].y, treeLights[3].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[15].diffuse"), lightColor3.x, lightColor3.y, lightColor3.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[15].specular"), lightColor3.x, lightColor3.y, lightColor3.z);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[15].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[15].linear"), 0.09f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[15].quadratic"), 0.032f);
// Point light 5
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[15].position"), treeLights[4].x, treeLights[4].y, treeLights[4].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[15].diffuse"), lightColor3.x, lightColor3.y, lightColor3.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[15].specular"), lightColor3.x, lightColor3.y, lightColor3.z);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[15].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[15].linear"), 0.09f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[15].quadratic"), 0.032f);
```

Además, se modificó el archivo de shaders: *lighting.frag* para poder cambiar el número de pointLights. La luz ambiental se dejó de manera que se aprecien los colores de manera correcta, y la SpotLight se dejó de manera que no sea muy invasiva.

Para terminar se usó la función seno para alternar el encendido y apagado de las luces decorativas. Las luces se activan con la tecla ESPACIO.

3. Código.

Se anexan algunos segmentos del código a manera de explicación breve.

```
int main()
{
    // Init GLFW
    glfwInit();

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

    if (nullptr == window) { ... }

    glfwMakeContextCurrent(window);

    glfwGetFramebufferSize(window, &SCREEN_WIDTH, &SCREEN_HEIGHT);

    // Set the required callback functions
    glfwSetKeyCallback(window, KeyCallback);
    glfwSetCursorPosCallback(window, MouseCallback);
    printf("%f", glfwGetTime());

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

    // Set this to true so GLEW knows to use a modern approach to retrieving function pointers and extensions
    glewExperimental = GL_TRUE;
    // Initialize GLEW to setup the OpenGL Function pointers
    if (GLEW_OK != glewInit()) { ... }

    // Define the viewport dimensions
    glViewport(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT);
```

Se inicia con el método Main, se mandan llamar funciones que establezcan diversos parámetros relacionados a la ventana, teclado y mouse.

```
// First, set the container's VAO (and VBO)
GLuint VBO, VAO, EBO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
glGenBuffers(1, &EBO);

glBindVertexArray(VAO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);

// Position attribute
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(0);
// Normals attribute
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)(3 * sizeof(GLfloat)));
glEnableVertexAttribArray(1);
// Texture Coordinate attribute
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)(6 * sizeof(GLfloat)));
glEnableVertexAttribArray(2);
glBindVertexArray(0);

// Then, we set the light's VAO (VBO stays the same. After all, the vertices are the same for the light object (also a 3D cube))
GLuint lightVAO;
glGenVertexArrays(1, &lightVAO);
glBindVertexArray(lightVAO);
// We only need to bind to the VBO (to link it with glVertexAttribPointer), no need to fill it; the VBO's data already contains a
glBindBuffer(GL_ARRAY_BUFFER, VBO);
// Set the vertex attributes (only position data for the lamp))
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)0); // Note that we skip over the other data in our
glEnableVertexAttribArray(0);
glBindVertexArray(0);


//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);

// Load textures
vector<const GLchar*> faces;
faces.push_back("SkyBox/Winter/winter-skyboxes/IceLake/posx.jpg");
faces.push_back("SkyBox/Winter/winter-skyboxes/IceLake/negx.jpg");
faces.push_back("SkyBox/Winter/winter-skyboxes/IceLake/posy.jpg");
faces.push_back("SkyBox/Winter/winter-skyboxes/IceLake/negy.jpg");
faces.push_back("SkyBox/Winter/winter-skyboxes/IceLake/posz.jpg");
faces.push_back("SkyBox/Winter/winter-skyboxes/IceLake/negz.jpg");
```

En la captura anterior, se establecen los apuntadores en la memoria y los espacios reservados para poder manejar el VBO y el EVO según corresponda. Y también se manejan los índices del SkyBox y se dibujan.

```
// Create camera transformations
glm::mat4 view;
view = camera.GetViewMatrix();
// Get the uniform locations
GLint modelLoc = glGetUniformLocation(lightingShader.Program, "model");
GLint viewLoc = glGetUniformLocation(lightingShader.Program, "view");
GLint projLoc = glGetUniformLocation(lightingShader.Program, "projection");

// Pass the matrices to the shader
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
glBindVertexArray(VAO);
glEnable(GL_DEPTH_TEST);


glm::mat4 model(1);
//Carga de modelo
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
House.Draw(lightingShader);

model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Kitchen.Draw(lightingShader);

model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Room.Draw(lightingShader);
```

Después se utiliza una matriz cuadrada 4x4 para poder manipular los modelos que se utilizarán en el proyecto y a partir de ahí se hacen las transformaciones correspondientes y se dibujan los modelos con sus texturas en el escenario. Se crean las matrices necesarias para la proyección ortogonal y el manejo de la cámara.

```
void animacion(){ ... }


// Is called whenever a key is pressed/released via GLFW
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode){ ... }

void MouseCallback(GLFWwindow* window, double xPos, double yPos){ ... }

// Moves/alters the camera positions based on user input
void DoMovement(){ ... }
```

Finalmente, se muestran las últimas 4 animaciones. La primera ya se tocó en la parte de las animaciones. La segunda se usa para manejar los eventos de teclado, como lo son la iniciación de las animaciones y la activación de las luces, además del cierre del proyecto con la tecla ESC.

La tercera se usa para manejar la rotación de la cámara con el mouse. Y la última función sirve para poder manejar el desplazamiento de la cámara mediante las teclas AWSD para un desplazamiento rápido, o las flechas de arriba, abajo, izquierda, derecha para un desplazamiento más lento.

Conclusión.

El presente documento establece los conceptos fundamentales para el correcto manejo y entendimiento del proyecto de la asignatura. Se tocan los elementos más importantes del código, las animaciones y los modelos para que cualquiera que quiera acceder al contenido de manera más técnica.