



# UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

## FACULTAD DE INGENIERÍA

---

## Manual Técnico: Gotham City

---

COMPUTACIÓN GRÁFICA E INTERACCIÓN HUMANO COMPUTADORA

Fecha límite de entrega: 25/05/2022

*Alumna:* LÁZARO MARTÍNEZ ANNETTE  
ARIADNA  
**No. CUENTA:** 316129189

*Profesor:* ING. JOSE ROQUE  
ROMAN GUADARRAMA  
**GRUPO 3**  
SEMESTRE 2022-2



# Índice

<b>1. Geometría</b>	<b>3</b>
1.1. Magica Voxel . . . . .	3
1.2. Modelos formato .vox . . . .	3
1.3. Modelos formato .obj . . . .	4
1.3.1. Texturas de una dimensión . . . . .	5
1.4. Optimización de modelos . . . . .	5
1.5. Carga de modelos . . . . .	5
1.6. Jerarquía de modelos . . . . .	7
1.7. Resultados . . . . .	10
1.7.1. Árboles . . . . .	11
1.7.2. Edificios . . . . .	11
1.7.3. Personas . . . . .	12
1.7.4. Transportes . . . . .	12
1.7.5. Tubos . . . . .	13
1.7.6. Otros de calle . . . . .	13
<b>2. Avatar</b>	<b>13</b>
2.1. Primitivas: cubos . . . . .	13
2.2. Textura . . . . .	15
2.2.1. Modelo en Magica Voxel . . . . .	15
2.2.2. Creación de la textura . . . . .	16
2.2.3. Posicionamiento de la textura . . . . .	16
2.2.4. Jerarquía . . . . .	17
2.3. Objetos adicionales . . . . .	18
2.4. Material para interactuar con luces . . . . .	18
2.5. Animación . . . . .	19
2.6. Resultados . . . . .	23
<b>3. Recorrido</b>	<b>24</b>
3.1. Cámara aérea . . . . .	25
3.2. Cámara ligada al piso en tercera persona . . . . .	25
3.3. Cambio de cámaras . . . . .	25
3.4. Control del movimiento . . . . .	26
3.5. Resultados . . . . .	28
<b>4. Iluminación</b>	<b>29</b>
4.1. Ciclo día-noche . . . . .	29
4.1.1. Cambio de Skybox . . . . .	29
4.1.2. Iluminación con base al ciclo . . . . .	30
4.1.3. Cambio de texturas con base al ciclo . . . . .	31
4.2. Iluminación por teclado . . . . .	33
4.3. Show de luces . . . . .	35
4.4. Resultados . . . . .	37



<b>5. Animación</b>	<b>37</b>
5.1. Básica . . . . .	38
5.1.1. Batmobile: Movimiento por teclado . . . . .	38
5.1.2. Batiseñal: Con base al ciclo de día-noche . . . . .	40
5.1.3. Transportes de las calles . . . . .	40
5.2. Compleja . . . . .	46
5.2.1. Red Hood . . . . .	46
5.2.2. Nightwing . . . . .	48
5.2.3. Ave azul . . . . .	50
5.3. Técnica de animación . . . . .	52
5.3.1. KeyFrames . . . . .	52
5.3.2. Dirigible . . . . .	55
5.3.3. Helicóptero . . . . .	58
<b>6. Audio</b>	<b>63</b>
6.1. Biblioteca implementada . . . . .	63
6.1.1. Modificaciones del proyecto . . . . .	63
6.2. Clase 'Sound' . . . . .	64
6.3. Sonido utilizado . . . . .	66
6.4. Utilización . . . . .	66
<b>7. Referencias</b>	<b>66</b>



# 1. Geometría

## 1.1. Magica Voxel

Debido a que se solicitó un estilo específico de Voxel Art para el proyecto, la mayoría de los modelos se realizaron utilizando el software de Magica Voxel, el cual permite personalizar diferentes modelos con este estilo, y guardarlos en diferentes formatos, incluyendo .obj

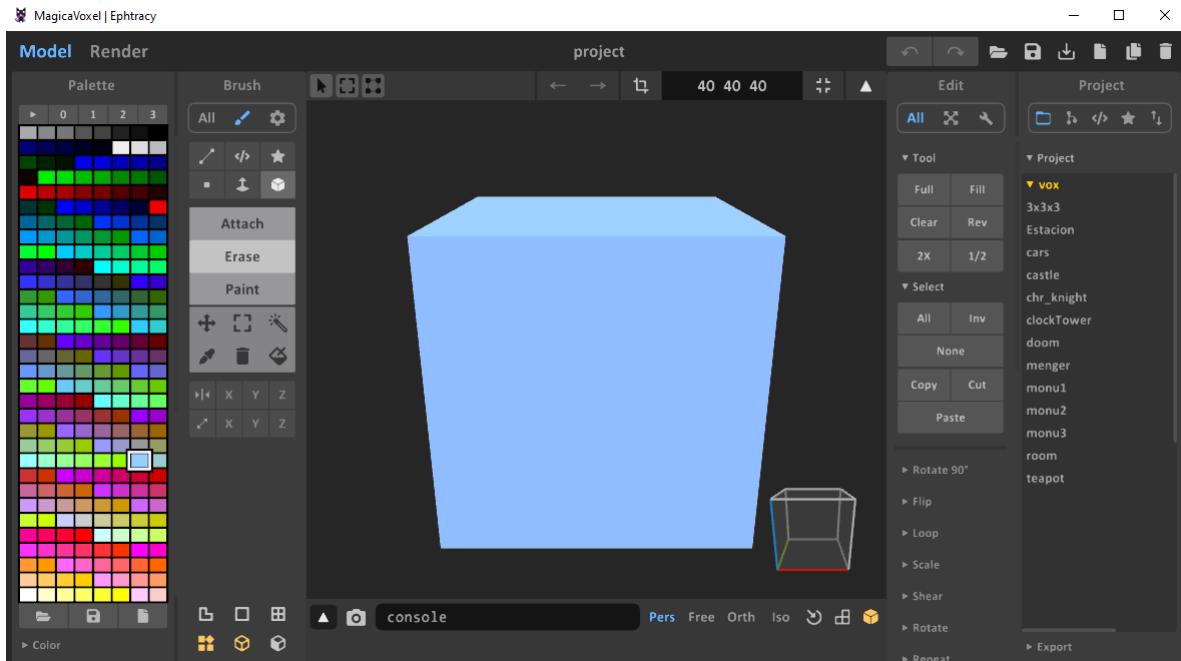


Figura 2: Software de modelado

## 1.2. Modelos formato .vox

Todos los modelos los almacené en formato .vox, el cual es editable desde Magica Voxel, por si era necesario modificarlos en un futuro. Estos modelos se encuentran en el repositorio.

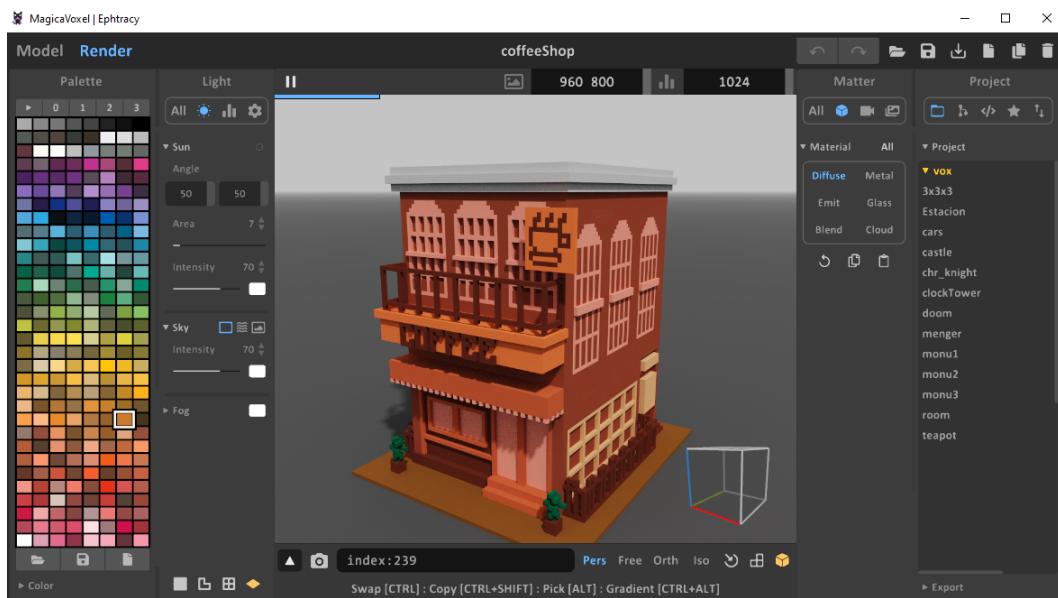


Figura 3: Modelo de ejemplo 1 en Magica Voxel

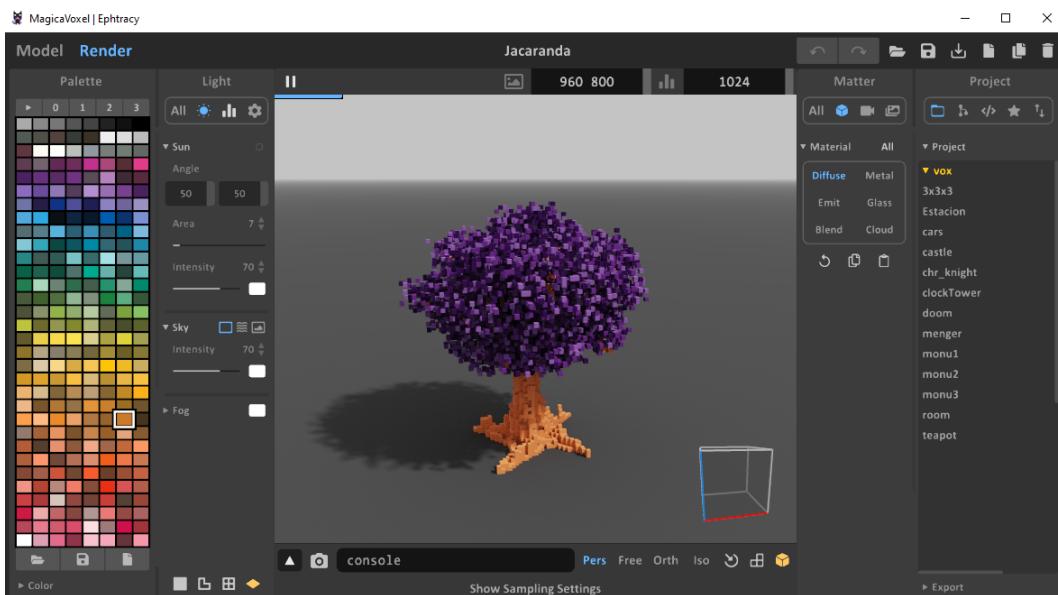


Figura 4: Modelo de ejemplo 2 en Magica Voxel

### 1.3. Modelos formato .obj

Para poder importar los modelos a OpenGL y realizar ajustes con los mismos, fue necesario exportar los modelos a formato .obj

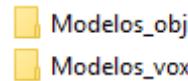


Figura 5: Carpeta de modelos

### 1.3.1. Texturas de una dimensión

Cabe mencionar que estos modelos, al importarse desde Magica Voxel, incluían texturas de una sola dimensión.

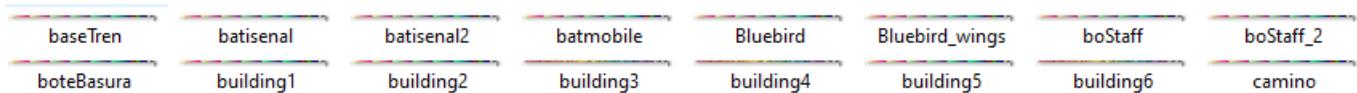


Figura 6: Ejemplo de Texturas utilizadas

## 1.4. Optimización de modelos

Fue necesario modificar algunos de los modelos .obj desde 3DS Max, principalmente cuando se tenía un modelo cuyo pivote debía modificarse.

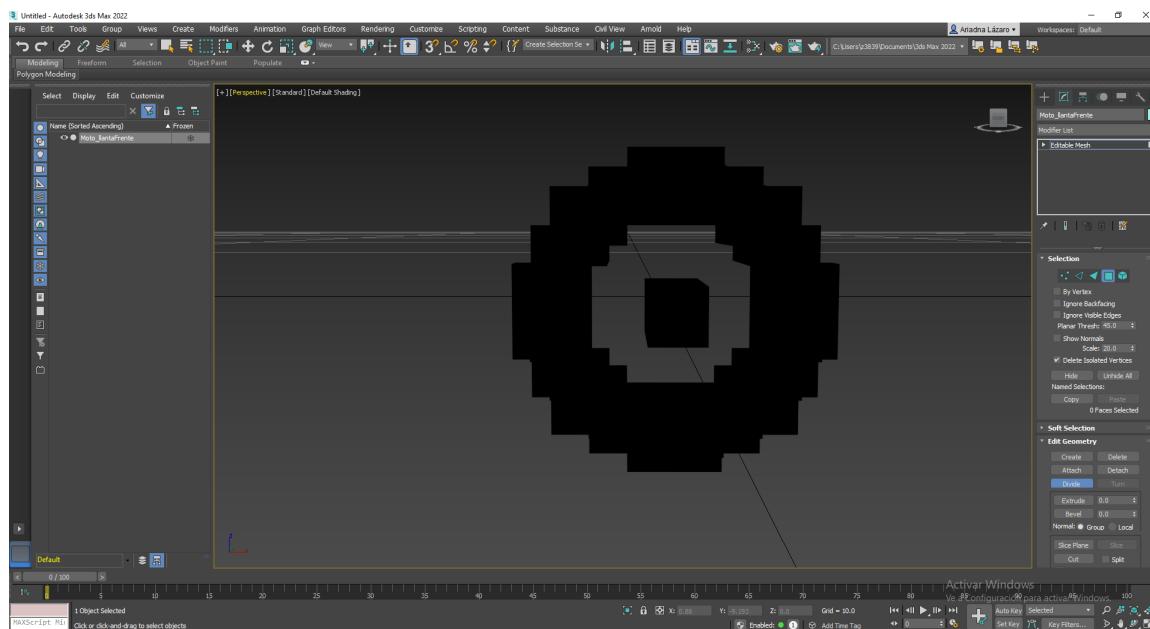


Figura 7: Ejemplo de modelo optimizado

## 1.5. Carga de modelos

La carga de modelo en OpenGL implicó crear una instancia de la clase 'Model' para cada uno, y asignarle la ruta correspondiente, asegurándonos de que su textura se encontrara en la carpeta 'Textures'.



```
Model WE_M;
Model BaseTren_M;
Model Coffee_M;
Model Building1_M;
Model Building2_M;
Model Building3_M;
Model Building4_M;
Model Building5_M;
Model Building6_M;
Model Tree_M;
Model Tree2_M;
Model Moto1_M;
Model tren_M;
Model BaseTren_M;
Model Estacion_M;
Model Auto1_M;
Model Auto2_M;
Model Auto_llanta;
Model Dirigible_M;
Model Patrulla_M;
Model Moto1_M;
Model Moto2_M;
Model Moto3_M;
Model Moto3_llanta;
Model Moto4_M;
Model helice_M;
Model Pole_M;
Model Banca_M;
Model Batisenall1_M;
Model Batisenall2_M;
Model BlueBird_M;
Model BlueBirdWings_M;
Model Bote_M;
Model Box_M;
Model Semaforo_M;
Model Semaforo_Pole_M;
Model Semaforo_Top_M;
Model Semaforo_V_M;
Model Semaforo_A_M;
Model Semaforo_R_M;
Model PlatTren_M;
Model PlatTren_luz_M;
```

Figura 8: Instancias creadas

```
//Calle
Camino_M = Model();
Camino_M.LoadModel("Modelos_obj/Others-street/camino.obj");
Tree_M = Model();
Tree_M.LoadModel("Modelos_obj/Arbol/Tree_obj.obj");
Tree2_M = Model();
Tree2_M.LoadModel("Modelos_obj/Arbol/arbo12.obj");
Tree3_M = Model();
Tree3_M.LoadModel("Modelos_obj/Arbol/Jacaranda.obj");
BlueBird_M = Model();
BlueBird_M.LoadModel("Modelos_obj/Others-street/Bluebird.obj");
BlueBirdWings_M = Model();
BlueBirdWings_M.LoadModel("Modelos_obj/Others-street/Bluebird_wings.obj");
Pole_M = Model();
Pole_M.LoadModel("Modelos_obj/Poles/pole2.obj");
Banca_M = Model();
Banca_M.LoadModel("Modelos_obj/Others-street/banca.obj");
Batisenall1_M = Model();
Batisenall1_M.LoadModel("Modelos_obj/Others-street/batisenal.obj");
Batisenall2_M = Model();
Batisenall2_M.LoadModel("Modelos_obj/Others-street/batisenal2.obj");
PhoneBox_M = Model();
PhoneBox_M.LoadModel("Modelos_obj/Others-street/phonebox.obj");
Bote_M = Model();
Bote_M.LoadModel("Modelos_obj/Others-street/boteBasura.obj");
Semaforo_M = Model();
Semaforo_M.LoadModel("Modelos_obj/Others-street/semaforo.obj");
Semaforo_Pole_M = Model();
Semaforo_Pole_M.LoadModel("Modelos_obj/Others-street/semaforo_pole.obj");
Semaforo_Top_M = Model();
Semaforo_Top_M.LoadModel("Modelos_obj/Others-street/Semaforo_top_apagado.obj");
Semaforo_V_M = Model();
Semaforo_V_M.LoadModel("Modelos_obj/Others-street/Semaforo_top_v.obj");
Semaforo_A_M = Model();
Semaforo_A_M.LoadModel("Modelos_obj/Others-street/Semaforo_top_a.obj");
Semaforo_R_M = Model();
Semaforo_R_M.LoadModel("Modelos_obj/Others-street/Semaforo_top_r.obj");
Escenario_M = Model();
Escenario_M.LoadModel("Modelos_obj/Others-street/Escenario.obj");
pista_M = Model();
pista_M.LoadModel("Modelos_obj/Transportes/pista.obj");
```

Figura 9: Asignación de las rutas de los archivos



```
Moto2_M = Model();
Moto2_M.LoadModel("Modelos_obj/Transportes/moto2.obj");
Moto3_M = Model();
Moto3_M.LoadModel("Modelos_obj/Transportes/moto3.obj");
Moto3_llanta = Model();
Moto3_llanta.LoadModel("Modelos_obj/Transportes/moto2_llanta.obj");
helicopter_M = Model();
helicopter_M.LoadModel("Modelos_obj/Transportes/helicopter.obj");
helice_M = Model();
helice_M.LoadModel("Modelos_obj/Transportes/helicopter_helice.obj");
Helipuerto_M = Model();
Helipuerto_M.LoadModel("Modelos_obj/Transportes/Helipuerto.obj");

//Edificios
WE_M = Model();
WE_M.LoadModel("Modelos_obj/Edificios/WE.obj");
ClockTower_M = Model();
ClockTower_M.LoadModel("Modelos_obj/Edificios/clockTower.obj");
Coffe_M = Model();
Coffe_M.LoadModel("Modelos_obj/Edificios/coffeeShop.obj");
Building1_M = Model();
Building1_M.LoadModel("Modelos_obj/Edificios/building1.obj");
Building2_M = Model();
Building2_M.LoadModel("Modelos_obj/Edificios/building2.obj");
Building3_M = Model();
Building3_M.LoadModel("Modelos_obj/Edificios/building3.obj");
Building4_M = Model();
Building4_M.LoadModel("Modelos_obj/Edificios/building4.obj");
Building5_M = Model();
Building5_M.LoadModel("Modelos_obj/Edificios/building5.obj");
Building6_M = Model();
Building6_M.LoadModel("Modelos_obj/Edificios/building6.obj");
```

Figura 10: Asignación de las rutas de los archivos

Finalmente, se realizaron las transformaciones necesarias con cada uno de los modelos para posicionarlos dentro del escenario de forma correcta.

```
//WE 0,0
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -3.3f, -60.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(5.0f, 7.0f, 5.0f));
model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
MaterialNormal.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
WE_M.RenderModel();
```

Figura 11: Ejemplo de carga de modelo

## 1.6. Jerarquía de modelos

Debido a la naturaleza de los modelos, fue necesario realizar la jerarquía por código de algunos objetos, principalmente los transportes como autos o motocicletas, sin embargo, también se utilizó la jerarquía con los humanoides, con el ave, e incluso con los modelos que rodean a cada uno de los edificios, porque de esta forma, al tomar a los edificios como centro de la jerarquía, fue sencillo mover a los demás objetos para acomodar todos los edificios dentro del escenario.



```
//RedHood
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(xRh, yRh, zRh));
model = glm::translate(model, glm::vec3(cuerpo1, 0.0f, cuerpo1 * cuerpo1));
model = glm::scale(model, glm::vec3(0.48f, 0.5f, 0.48f));
model = glm::rotate(model, rotCuerpo * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
modelaux2 = model;
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
Rh_Cuerpo_M.RenderModel();

//Brazo izq
model = modelaux2;
model = glm::translate(model, glm::vec3(-1.55f, 1.2f, 0.0f));
model = glm::rotate(model, brazo1 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
Rh_Brazo_M.RenderModel();

//Brazo der
model = modelaux2;
model = glm::translate(model, glm::vec3(1.55f, 1.2f, 0.0f));
model = glm::rotate(model, brazo2 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
Rh_Brazo_M.RenderModel();

//Pierna izq
model = modelaux2;
model = glm::translate(model, glm::vec3(-0.75f, -1.63f, 0.0f));
model = glm::rotate(model, pierna2 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
Rh_Pierna_M.RenderModel();

//Pierna der
model = modelaux2;
model = glm::translate(model, glm::vec3(0.75f, -1.63f, 0.0f));
model = glm::rotate(model, pierna1 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
Rh_Pierna_M.RenderModel();
```

Figura 12: Jerarquía de humanoides

```
//Bird
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(180.0f - xbird, 56.0f + ybird, 70.0f - zbird));
model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, rotBird * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
modelaux = model;
MaterialNormal.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
BlueBird_M.RenderModel();

//alas
model = modelaux;
model = glm::translate(model, glm::vec3(0.0f, -0.1f, -0.2f));
model = glm::rotate(model, rotWings * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
MaterialNormal.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
BlueBirdWings_M.RenderModel();
```

Figura 13: Jerarquía del ave



```
//Moto RedHood
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(140.0f, -1.9f, -135.0f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
modelaux2 = model;
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
MaterialParaluces.UseMaterial(uniformSpecularIntensity, uniformShininess);
Moto1_M.RenderModel();

//Llanta delantera
model = modelaux2;
model = glm::translate(model, glm::vec3(-2.1f, 1.15f, 0.0f));
MaterialNormal.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Moto1_LlantaDelantera.RenderModel();

//Llanta trasera
model = modelaux2;
model = glm::translate(model, glm::vec3(2.0f, 1.15f, 0.0f));
MaterialNormal.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Moto1_LlantaTrasera.RenderModel();
```

Figura 14: Jerarquía de transportes

```
//arbol
model = modelaux;
model = glm::translate(model, glm::vec3(20.0f, 1.4f, 20.0f));
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Tree_M.RenderModel();

model = modelaux;
model = glm::translate(model, glm::vec3(-20.0f, 1.4f, 20.0f));
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Tree_M.RenderModel();

model = modelaux;
model = glm::translate(model, glm::vec3(20.0f, 1.4f, -20.0f));
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Tree_M.RenderModel();

model = modelaux;
model = glm::translate(model, glm::vec3(-20.0f, 1.4f, -20.0f));
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Tree_M.RenderModel();

//Jacaranda
model = modelaux;
model = glm::translate(model, glm::vec3(40.0f, 1.4f, 0.0f));
model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Tree3_M.RenderModel();

model = modelaux;
model = glm::translate(model, glm::vec3(-40.0f, 1.4f, 0.0f));
model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Tree3_M.RenderModel();

//Phonebox
model = modelaux;
model = glm::translate(model, glm::vec3(-40.0f, 1.4f, 20.0f));
model = glm::scale(model, glm::vec3(1.0f, 1.2f, 1.0f));
model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
```

Figura 15: Jerarquía con edificios parte 1



```
//arbol
model = modelaux;
model = glm::translate(model, glm::vec3(20.0f, 1.4f, 20.0f));
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Tree_M.RenderModel();

model = modelaux;
model = glm::translate(model, glm::vec3(-20.0f, 1.4f, 20.0f));
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Tree_M.RenderModel();

model = modelaux;
model = glm::translate(model, glm::vec3(20.0f, 1.4f, -20.0f));
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Tree_M.RenderModel();

model = modelaux;
model = glm::translate(model, glm::vec3(-20.0f, 1.4f, -20.0f));
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Tree_M.RenderModel();

//Jacaranda
model = modelaux;
model = glm::translate(model, glm::vec3(40.0f, 1.4f, 0.0f));
model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Tree3_M.RenderModel();

model = modelaux;
model = glm::translate(model, glm::vec3(-40.0f, 1.4f, 0.0f));
model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Tree3_M.RenderModel();

//Phonebox
model = modelaux;
model = glm::translate(model, glm::vec3(-40.0f, 1.4f, 20.0f));
model = glm::scale(model, glm::vec3(1.0f, 1.2f, 1.0f));
model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
```

Figura 16: Jerarquía con edificios parte 2

## 1.7. Resultados

Para una mejor organización de los modelos, cree subcarpetas que me permitieran guardar los diferentes modelos creados.

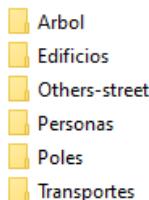


Figura 17: Categorías de los modelos

A continuación se muestra un ejemplo de los diferentes objetos instanciados dentro del escenario del proyecto final.

### 1.7.1. Árboles

En esta carpeta se almacenaron principalmente los árboles de hojas verdes y las Jacarandas.



Figura 18: Árboles

### 1.7.2. Edificios

En esta carpeta se almacenaron los 9 edificios diferentes utilizados en el proyecto, así como algunos de los objetos de dichos edificios que cambian de acuerdo al ciclo de día y noche.



Figura 19: Edificios 1

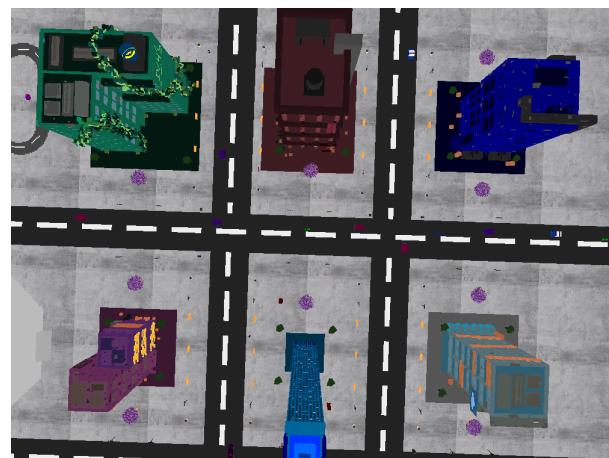


Figura 20: Edificios 2



### 1.7.3. Personas

En esta carpeta se almacenaron a los personajes principales y sus armas.



Figura 21: Personas 1



Figura 22: Personas 2

### 1.7.4. Transportes

En esta carpeta se almacenaron los autos, motocicletas, al helicóptero, al dirigible, al tren, y al Batimobile.

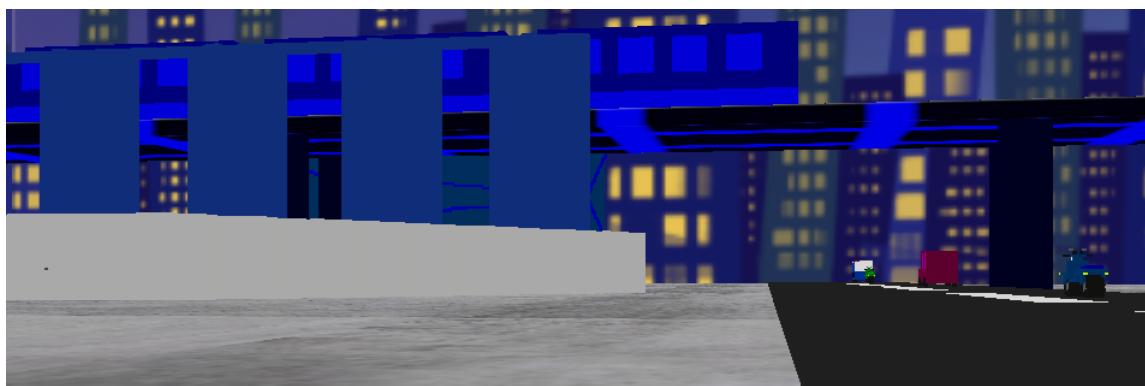


Figura 23: Transportes

### 1.7.5. Tubos

En esta carpeta se almacenaron principalmente los postes de luz y el arma del personaje principal.



Figura 24: Tubos

### 1.7.6. Otros de calle

En esta carpeta se almacenaron los modelos correspondientes a las bancas, los botes de basura, las cabinas telefónicas, los semáforos, etc.



Figura 25: Otros de calle

## 2. Avatar

### 2.1. Primitivas: cubos

Debido al estilo de dibujo, el tipo de primitivas que utilicé para el avatar fueron cubos, sin embargo, debido a que cada parte del avatar debería tener coordenadas de textura diferentes, tuve que realizar un cubo diferente para cada miembro de su cuerpo.

Cabe mencionar que se utilizaron cubos unitarios y posteriormente se usaron transformaciones para modificar el tamaño de los objetos, además, teniendo como referencia los cuadros de Magica Voxel, la escala de los cubos y su posterior posicionamiento fue relativamente sencillo.



```
void CrearPersonaje()
{
    unsigned int cara_indices[] = {
        // front
        0, 1, 2,
        2, 3, 0,
        // right
        4, 5, 6,
        6, 7, 4,
        // back
        8, 9, 10,
        10, 11, 8,
        // left
        12, 13, 14,
        14, 15, 12,
        // bottom
        16, 17, 18,
        18, 19, 16,
        // top
        20, 21, 22,
        22, 23, 20,
    };

    GLfloat cara_vertices[] = {
        // front
        //x      y      z      S      T      NX     NY     NZ
        -0.5f, -0.5f,  0.5f,  0.09375f,  0.87697f,  0.0f,  0.0f, -1.0f, //0
        0.5f, -0.5f,  0.5f,  0.26660f,  0.87697f,  0.0f,  0.0f, -1.0f, //1
        0.5f,  0.5f,  0.5f,  0.26660f,  0.99982f,  0.0f,  0.0f, -1.0f, //2
        -0.5f,  0.5f,  0.5f,  0.09375f,  0.99982f,  0.0f,  0.0f, -1.0f, //3
        // right
        //x      y      z      S      T
        0.5f, -0.5f,  0.5f,  0.46584f,  0.36994f,  -1.0f,  0.0f,  0.0f,
        0.5f, -0.5f, -0.5f,  0.61035f,  0.36994f,  -1.0f,  0.0f,  0.0f,
        0.5f, -0.5f, -0.5f,  0.61035f,  0.49611f,  -1.0f,  0.0f,  0.0f,
        0.5f,  0.5f,  0.5f,  0.46584f,  0.49611f,  -1.0f,  0.0f,  0.0f,
        // back
        -0.5f, -0.5f, -0.5f,  0.46584f,  0.36994f,  0.0f,  0.0f,  1.0f,
        0.5f, -0.5f, -0.5f,  0.61035f,  0.36994f,  0.0f,  0.0f,  1.0f,
        0.5f,  0.5f, -0.5f,  0.61035f,  0.49611f,  0.0f,  0.0f,  1.0f,
        -0.5f,  0.5f, -0.5f,  0.46584f,  0.49611f,  0.0f,  0.0f,  1.0f,
        // left
        //x      y      z      S      T
        -0.5f, -0.5f, -0.5f,  0.46584f,  0.36994f,  1.0f,  0.0f,  0.0f,
        -0.5f, -0.5f,  0.5f,  0.61035f,  0.36994f,  1.0f,  0.0f,  0.0f,
        -0.5f,  0.5f,  0.5f,  0.61035f,  0.49611f,  1.0f,  0.0f,  0.0f,
        -0.5f,  0.5f, -0.5f,  0.46584f,  0.49611f,  1.0f,  0.0f,  0.0f,
        // bottom
        //x      y      z      S      T
        -0.5f, -0.5f,  0.5f,  0.89551f,  0.87009f,  0.0f,  1.0f,  0.0f,
        0.5f, -0.5f,  0.5f,  0.99802f,  0.87009f,  0.0f,  1.0f,  0.0f,
        0.5f, -0.5f, -0.5f,  0.99802f,  0.77541f,  0.0f,  1.0f,  0.0f,
        -0.5f, -0.5f, -0.5f,  0.89551f,  0.77541f,  0.0f,  1.0f,  0.0f,
        //UP
        //x      y      z      S      T
        -0.5f,  0.5f,  0.5f,  0.88867f,  0.53320f,  0.0f,  -1.0f,  0.0f,
        0.5f,  0.5f,  0.5f,  1.0f,  0.53320f,  0.0f,  -1.0f,  0.0f,
        0.5f,  0.5f, -0.5f,  1.0f,  0.63085f,  0.0f,  -1.0f,  0.0f,
        -0.5f,  0.5f, -0.5f,  0.88867f,  0.63085f,  0.0f,  -1.0f,  0.0f,
```

Figura 26: Creación de los cubos parte 1

```
// left
//x      y      z      S      T
-0.5f, -0.5f, -0.5f,  0.46584f,  0.36994f,  1.0f,  0.0f,  0.0f,
-0.5f, -0.5f,  0.5f,  0.61035f,  0.36994f,  1.0f,  0.0f,  0.0f,
-0.5f,  0.5f,  0.5f,  0.61035f,  0.49611f,  1.0f,  0.0f,  0.0f,
-0.5f,  0.5f, -0.5f,  0.46584f,  0.49611f,  1.0f,  0.0f,  0.0f,
// bottom
//x      y      z      S      T
-0.5f, -0.5f,  0.5f,  0.89551f,  0.87009f,  0.0f,  1.0f,  0.0f,
0.5f, -0.5f,  0.5f,  0.99802f,  0.87009f,  0.0f,  1.0f,  0.0f,
0.5f, -0.5f, -0.5f,  0.99802f,  0.77541f,  0.0f,  1.0f,  0.0f,
-0.5f, -0.5f, -0.5f,  0.89551f,  0.77541f,  0.0f,  1.0f,  0.0f,
```

Figura 27: Creación de los cubos parte 2

```
unsigned int cuerpo_indices[] = [ ... ]
GLfloat cuerpo_vertices[] = [ ... ]
unsigned int brazoIzq_indices[] = [ ... ]
GLfloat brazoIzq_vertices[] = [ ... ]
unsigned int brazoDer_indices[] = [ ... ]
GLfloat brazoDer_vertices[] = [ ... ]
unsigned int piernaIza_indices[] = [ ... ]
GLfloat piernaIza_vertices[] = [ ... ]
unsigned int piernaDer_indices[] = [ ... ]
GLfloat piernaDer_vertices[] = [ ... ]
unsigned int pieIzq_indices[] = [ ... ]
GLfloat pieIzq_vertices[] = [ ... ]
unsigned int pieDer_indices[] = [ ... ]
GLfloat pieDer_vertices[] = [ ... ]
unsigned int cuello_indices[] = [ ... ]
GLfloat cuello_vertices[] = [ ... ]
```

Figura 28: Creación de todas las partes del cuerpo



```
Mesh* cara = new Mesh();
cara->CreateMesh(cara_vertices, cara_indices, 192, 36);
meshList.push_back(cara);

Mesh* cuerpo = new Mesh();
cuerpo->CreateMesh(cuerpo_vertices, cuerpo_indices, 192, 36);
meshList.push_back(cuerpo);

Mesh* b1 = new Mesh();
b1->CreateMesh(brazoIzq_vertices, brazoIzq_indices, 192, 36);
meshList.push_back(b1);

Mesh* b2 = new Mesh();
b2->CreateMesh(brazoDer_vertices, brazoDer_indices, 192, 36);
meshList.push_back(b2);

Mesh* p1 = new Mesh();
p1->CreateMesh(piernaIzq_vertices, piernaIzq_indices, 192, 36);
meshList.push_back(p1);

Mesh* p2 = new Mesh();
p2->CreateMesh(piernaDer_vertices, piernaDer_indices, 192, 36);
meshList.push_back(p2);

Mesh* pie1 = new Mesh();
pie1->CreateMesh(pieIzq_vertices, pieIzq_indices, 192, 36);
meshList.push_back(pie1);

Mesh* pie2 = new Mesh();
pie2->CreateMesh(pieDer_vertices, pieDer_indices, 192, 36);
meshList.push_back(pie2);

Mesh* cuello = new Mesh();
cuello->CreateMesh(cuello_vertices, cuello_indices, 192, 36);
meshList.push_back(cuello);
```

Figura 29: Envío de cada cubo al mesh

```
CreateObjects();
CrearPersonaje();
CreateShaders();
```

Figura 30: Llamado de las funciones que crean los cubos desde la función main

## 2.2. Textura

### 2.2.1. Modelo en Magica Voxel

Para la creación de la textura, fue necesario realizar un modelo en Magica Voxel primero.



Figura 31: Modelo de referencia en Magica Voxel



### 2.2.2. Creación de la textura

Después, se tomaron las capturas necesarias, se realizó el ajuste de color y tamaño correspondiente, se eliminó el fondo, se exportó en formato .tga y se obtuvo la textura del personaje desde GIMP.



Figura 32: Textura del avatar

### 2.2.3. Posicionamiento de la textura

Para aplicarle la textura a cada uno de los cubos, se tuvo que obtener una regla de 3 entre las dimensiones de la imagen de la textura (1024x1024), y el sistema de coordenadas S x T que tiene un rango entre 0 y 1, tal como se hizo en la práctica correspondiente a texturas.

```
GLfloat cuerpo_vertices[] = {
    // front
    //x   y      z      S      T      NX      NY      NZ
    -0.5f, -0.5f,  0.5f,  0.07812f,  0.65529f,  0.0f,  0.0f, -1.0f, //0
    0.5f, -0.5f,  0.5f,  0.28613f,  0.65420f,  0.0f,  0.0f, -1.0f, //1
    0.5f,  0.5f,  0.5f,  0.28613f,  0.85156f,  0.0f,  0.0f, -1.0f, //2
    -0.5f,  0.5f,  0.5f,  0.07812f,  0.85156f,  0.0f,  0.0f, -1.0f, //3
    // right
    //x   y      z      S      T      NX      NY      NZ
    0.5f, -0.5f,  0.5f,  0.18742f,  0.15234f, -1.0f,  0.0f,  0.0f,
    0.5f, -0.5f, -0.5f,  0.25390f,  0.15234f, -1.0f,  0.0f,  0.0f,
    0.5f,  0.5f, -0.5f,  0.25390f,  0.35056f, -1.0f,  0.0f,  0.0f,
    0.5f,  0.5f,  0.5f,  0.18742f,  0.35056f, -1.0f,  0.0f,  0.0f,
    // back
    -0.5f, -0.5f, -0.5f,  0.45019f,  0.65234f,  0.0f,  0.0f,  1.0f,
    0.5f, -0.5f, -0.5f,  0.65625f,  0.65234f,  0.0f,  0.0f,  1.0f,
    0.5f,  0.5f, -0.5f,  0.65625f,  0.84765f,  0.0f,  0.0f,  1.0f,
    -0.5f,  0.5f, -0.5f,  0.45019f,  0.84765f,  0.0f,  0.0f,  1.0f,
    // left
    //x   y      z      S      T      NX      NY      NZ
    -0.5f, -0.5f, -0.5f,  0.18742f,  0.15234f,  1.0f,  0.0f,  0.0f,
    -0.5f, -0.5f,  0.5f,  0.25390f,  0.15234f,  1.0f,  0.0f,  0.0f,
    -0.5f,  0.5f,  0.5f,  0.25390f,  0.35056f,  1.0f,  0.0f,  0.0f,
    -0.5f,  0.5f, -0.5f,  0.18742f,  0.35056f,  1.0f,  0.0f,  0.0f,
    // bottom
    //x   y      z      S      T      NX      NY      NZ
    -0.5f, -0.5f,  0.5f,  0.752929f,  0.99802f,  0.0f,  1.0f,  0.0f,
    0.5f, -0.5f,  0.5f,  0.87792f,  0.99902f,  0.0f,  1.0f,  0.0f,
    0.5f, -0.5f, -0.5f,  0.87792f,  0.89648f,  0.0f,  1.0f,  0.0f,
    -0.5f, -0.5f, -0.5f,  0.752929f,  0.89648f,  0.0f,  1.0f,  0.0f,
```

Figura 33: Posiciones de la textura para cada cara de un cubo



#### 2.2.4. Jerarquía

Finalmente, se mandaron a llamar a los objetos del mesh y se utilizaron las matrices auxiliares para realizar el modelado jerárquico de los miembros del cuerpo, tomando como centro el torso del personaje.

```
//Personaje
//cuerpo
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(-12.0f, movimientoY, -17.0f+ cuelpo3));
model = glm::rotate(model, rotCuerpo3 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(1.2f, 1.6f, 0.8f));
color = glm::vec3(1.0f, 1.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
TimTexture.UseTexture();
MaterialParaluces.UseMaterial(uniformSpecularIntensity, uniformShininess);
meshList[4]->RenderMesh();

//cuello
model = modelaux;
model = glm::translate(model, glm::vec3(0.0f, 0.8f, 0.0f));
modelaux2 = model;
model = glm::scale(model, glm::vec3(0.4f, 0.1f, 0.35f));
color = glm::vec3(1.0f, 1.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
TimTexture.UseTexture();
MaterialParaluces.UseMaterial(uniformSpecularIntensity, uniformShininess);
meshList[11]->RenderMesh();

//cara
model = modelaux2;
model = glm::translate(model, glm::vec3(0.0f, 0.55f, 0.0f));
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 0.8f));
color = glm::vec3(1.0f, 1.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
TimTexture.UseTexture();
MaterialParaluces.UseMaterial(uniformSpecularIntensity, uniformShininess);
meshList[3]->RenderMesh();

//Hombro izquierdo
model = modelaux;
model = glm::translate(model, glm::vec3(-0.775f, 0.5f, 0.0f));
model = glm::rotate(model, brazo5* toRadians, glm::vec3(1.0f, 0.0f, 0.0f));


```

Figura 34: Jerarquía del avatar parte 1

```
//brazo izq
model = glm::translate(model, glm::vec3(0.0f, -0.4f, 0.0f));
modelaux2 = model;
model = glm::scale(model, glm::vec3(0.35f, 1.2f, 0.4f));
color = glm::vec3(1.0f, 1.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
TimTexture.UseTexture();
MaterialParaluces.UseMaterial(uniformSpecularIntensity, uniformShininess);
meshList[5]->RenderMesh();

//bostaff
model = modelaux2;
model = glm::translate(model, glm::vec3(0.0f, -0.67f, 0.0f));
model = glm::scale(model, glm::vec3(1.0f, 1.0f, staffScale));
model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, staffRot * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
if (cambio == 3) {
    boStaff_M.RenderModel();
}
else {
    boStaff2_M.RenderModel();
}

//hombro derecho
model = modelaux;
model = glm::translate(model, glm::vec3(0.775f, 0.5f, 0.0f));
model = glm::rotate(model, brazo6 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));

//brazo der
model = glm::translate(model, glm::vec3(0.0f, -0.4f, 0.0f));
modelaux2 = model;
model = glm::scale(model, glm::vec3(0.35f, 1.2f, 0.4f));
color = glm::vec3(1.0f, 1.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
TimTexture.UseTexture();
MaterialParaluces.UseMaterial(uniformSpecularIntensity, uniformShininess);
meshList[6]->RenderMesh();
```

Figura 35: Jerarquía del avatar parte 2



```
//union pierna izq
model = modelaux;
model = glm::translate(model, glm::vec3(-0.375f, -0.625f, 0.0f));
model = glm::rotate(model, pierna6 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));

//pierna izq
model = glm::translate(model, glm::vec3(0.0f, -0.625f, 0.0f));
modelaux2 = model;
model = glm::scale(model, glm::vec3(0.45f, 0.95f, 0.4f));
color = glm::vec3(1.0f, 1.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
TmTexture.UseTexture();
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
meshList[7]->RenderMesh();

//pie izq
model = modelaux;
model = glm::translate(model, glm::vec3(0.0f, -0.6f, 0.125f));
model = glm::scale(model, glm::vec3(0.45f, 0.25f, 0.65f));
color = glm::vec3(1.0f, 1.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
TmTexture.UseTexture();
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
meshList[9]->RenderMesh();

//union pierna der
model = modelaux;
model = glm::translate(model, glm::vec3(0.375f, -0.625f, 0.0f));
model = glm::rotate(model, pierna5 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));

//pierna der
model = glm::translate(model, glm::vec3(0.0f, -0.625f, 0.0f));
modelaux2 = model;
model = glm::scale(model, glm::vec3(0.45f, 0.95f, 0.4f));
color = glm::vec3(1.0f, 1.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
TmTexture.UseTexture();
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
meshList[8]->RenderMesh();
```

Figura 36: Jerarquía del avatar parte 3

### 2.3. Objetos adicionales

También se utilizaron modelos adicionales para el avatar, como el gancho para escalar y su arma preferida, un bo staff retráctil.

```
//Gancho
model = modelaux2;
model = glm::translate(model, glm::vec3(0.0f, -0.67f - ganchoTras, 0.0f + ganchoMovZ));
model = glm::rotate(model, 90 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, ganchoRot * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
//model = glm::scale(model, glm::vec3(0.45f, 0.25f, 0.65f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
gancho_M.RenderModel();
```

Figura 37: Armas

### 2.4. Material para interactuar con luces

Se tuvieron que utilizar dos tipos de materiales diferentes, uno que fuera más brillante para interactuar con las luces y otro que fuera menos brillante.

```
Material MaterialParaLuces;
MaterialParaLuces = Material(5.0f, 300);
Material MaterialNormal;
MaterialNormal = Material(0.5f, 3);
```

Figura 38: Materiales



Después, se llamaron tanto las texturas correspondientes como el tipo de material para cada una de las partes del cuerpo del avatar.

```
//Personaje
//cuerpo
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-12.0f, movimientoY, -17.0f+ cuerpo3));
model = glm::rotate(model, rotCuerpo3 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(1.2f, 1.6f, 0.8f));
color = glm::vec3(1.0f, 1.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
TimTexture.UseTexture();
MaterialParaluces.UseMaterial(uniformSpecularIntensity, uniformShininess);
meshList[4]->RenderMesh();
```

Figura 39: Avatar

## 2.5. Animación

Primero, se modificaron los archivos Windows.h y Windows.cpp para tener una variable booleana llamada 'Tim' que modifique su valor dependiendo del estado de 2 teclas, así como las funciones necesarias para obtener su valor y regresar su valor a falso de ser necesario.

Las variables y funciones se declaran como se muestra a continuación:

```
GLboolean getTim() { return Tim; }
GLboolean getHelicoptero() { return helicoptero; }
void setTim(bool value) { Tim = value; }
bool getShouldClose() {
    return glfwWindowShouldClose(mainWindow);
}
bool* getKeys() { return keys; }
void swapBuffers() { return glfwSwapBuffers(mainWindow); }

~Window();
private:
    GLFWwindow *mainWindow;
    GLint width, height;
    bool keys[1024];
    GLint bufferWidth, bufferHeight;
    void createCallbacks();
    GLfloat lastX;
    GLfloat lastY;
    GLfloat xChange;
    GLfloat yChange;
    GLfloat mvevex;
    GLboolean CamAerea, spotlights, animacionBatmobile;
    GLboolean Tim, helicoptero;
```

Figura 40: Cambios en Windows.h

Y se agregan los eventos para cambiar su valor con las teclas '7' y '8'.

```
if (key == GLFW_KEY_7)
{
    theWindow->Tim = true;
}
if (key == GLFW_KEY_8)
{
    theWindow->Tim = false;
}
```

Figura 41: Cambios en Windows.cpp



Para iniciar la animación, el valor de la variable debe ser verdadero, lo que implica que solo inicia si el usuario presiona la tecla '7'. El cambio de la música de fondo al soundtrack del avatar inicia al instante

Para tener un control sobre el orden de la animación, se utilizó una variable llamada 'cambio', que controle el estado de la animación, así como variables auxiliares para la traslación, rotación y escala de las diferentes partes del avatar.

Al inicio de la animación se levanta el brazo correspondiente que está sujetando el bo staff, posteriormente, el segundo cambio implica el alargamiento del bo staff, debido a que es un arma retráctil. Luego, se realiza la rotación del arma durante 2 vueltas. Al terminar el movimiento, se retrae el bo staff y se baja el brazo.

```
// Animacion avatar
if (mainwindow.getTim()) {
    //Musica
    music.pause();
    Tim.play();
    Jason.pause();
    Dick.pause();
    Disparo.pause();

    if (cambio == 1 && brazo5 >= -90.0f) {
        brazo5 -= 2.0f;
    }
    else if (cambio == 1 && brazo5 < -90.0f) {
        cambio = 2;
    }
    else if (cambio == 2 && staffScale <= 4.0f) {
        staffScale += 0.2f;
    }
    else if (cambio == 2 && staffScale > 4.0f) {
        cambio = 3;
        staffScale = 1.0f;
    }
    else if (cambio == 3 && staffRot <= 360) {
        staffRot += 2.5f;
    }
    else if (cambio == 3 && staffRot > 360) {
        cambio = 4;
        staffRot = 0.0f;
        staffScale = 4.0f;
    }
    else if (cambio == 4 && staffScale > 1.0f) {
        staffScale -= 0.2f;
    }
    else if (cambio == 4 && staffScale <= 1.0f) {
        cambio = 5;
    }
    else if (cambio == 5 && brazo5 < 0.0f) {
        brazo5 += 2.0f;
    }
    else if (cambio == 5 && brazo5 >= 0.0f) {
        cambio = 6;
    }
}
```

Figura 42: Animación del avatar parte 1

En el cambio número 6 se realiza un movimiento continuo de los brazos y las piernas mientras se realiza un traslado, de forma que se simula el recorrido de un trayecto en línea recta. Al llegar a su destino, se regresa al valor inicial de los miembros.



```
else if (cambio == 6 && cuerpo3 <= 70.0f) {
    //Mov miembros
    if (brazo5 <= 30.0f && movBrazo5 == true) {
        brazo5 += 1.5f;
    }
    else if (brazo5 > 30.0f && movBrazo5 == true) {
        movBrazo5 = false;
    }
    else if (brazo5 >= -30.0f && movBrazo5 == false) {
        brazo5 -= 1.5f;
    }
    else if (brazo5 < -30.0f && movBrazo5 == false) {
        movBrazo5 = true;
    }

    if (brazo6 <= 30.0f && movBrazo6 == true) {
        brazo6 += 1.5f;
    }
    else if (brazo6 > 30.0f && movBrazo6 == true) {
        movBrazo6 = false;
    }
    else if (brazo6 >= -30.0f && movBrazo6 == false) {
        brazo6 -= 1.5f;
    }
    else if (brazo6 < -30.0f && movBrazo6 == false) {
        movBrazo6 = true;
    }

    if (pierna5 <= 30.0f && movPierna5 == true) {
        pierna5 += 1.5f;
    }
    else if (pierna5 > 30.0f && movPierna5 == true) {
        movPierna5 = false;
    }
    else if (pierna5 >= -30.0f && movPierna5 == false) {
        pierna5 -= 1.5f;
    }
    else if (pierna5 < -30.0f && movPierna5 == false) {
        movPierna5 = true;
    }
}
```

Figura 43: Animación del avatar parte 2

```
if (pierna6 <= 30.0f && movPierna6 == true) {
    pierna6 += 1.5f;
}
else if (pierna6 > 30.0f && movPierna6 == true) {
    movPierna6 = false;
}
else if (pierna6 >= -30.0f && movPierna6 == false) {
    pierna6 -= 1.5f;
}
else if (pierna6 < -30.0f && movPierna6 == false) {
    movPierna6 = true;
}

//Traslacion
cuerpo3 += 0.25f;
movimientoY += 0.005f;
}

else if (cambio == 6 && cuerpo3 > 70.0f) {
    pierna5 = 0.0f;
    pierna6 = 0.0f;
    brazo5 = 0.0f;
    brazo6 = 0.0f;
    cambio = 7;
}
```

Figura 44: Animación del avatar parte 3

El cambio 7 implica el movimiento del brazo que sostiene el gancho y el cambio 8 es el regreso de dicho brazo, de forma que se simula el lanzamiento del objeto. El cambio 9 realiza el lanzamiento y rotación del gancho, de forma que se queda sostenido en la orilla del edificio. El cambio 10 implica el movimiento del personaje debido al gancho y el 11 es cuando el avatar recupera su herramienta.



```
else if (cambio == 7 && brazo6 >= -210.0f) { } 7
} else if (cambio == 7 && brazo6 < -210.0f) {
    cambio = 8;
}
else if (cambio == 8 && brazo6 <= -180.0f) { } 8
} else if (cambio == 8 && brazo6 > -180.0f) {
    cambio = 9;
}
else if (cambio == 9 && ganchoTras <= 42.0f) { } 9
    ganchoTras += 0.5f;
    if (ganchoRot <= 110.0f) {
        ganchoRot += 5.0f;
    }
}
else if (cambio == 9 && ganchoTras > 42.0f) {
    cambio=10;
}
else if (cambio == 10 && movimientoY <= 43.8f) { } 10
    movimientoY+=0.2f;
    ganchoTras -= 0.2f;
    if (cuerpo3 <= 72.0f) {
        cuerpo3 += 0.1f;
    }
}
else if (cambio == 10 && movimientoY > 43.8f) {
    cambio=11;
}
else if (cambio == 11 && movimientoY <= 48.0f) { } 11
    movimientoY += 0.5f;
    if (cuerpo3 <= 76.0f) {
        cuerpo3 += 0.2f;
    }
}
else if (cambio == 11 && movimientoY > 48.0f) {
    brazo6 = 0.0f;
    ganchoRot = 0.0f;
    cambio = 12;
}
```

Figura 45: Animación del avatar parte 4

El cambio 12 es el giro del personaje y el 13 es el regreso al piso. A partir del cambio 14 se realiza el movimiento de los miembros y la traslación del cuerpo para regresar al lugar de inicio.

```
else if (cambio == 12 && rotCuerpo3 <=180.0f) {
    rotCuerpo3 += 5.0f;
}
else if (cambio == 12 && rotCuerpo3 > 180.0f) {
    cambio = 13;
}

else if (cambio == 13 && movimientoY >= 2.0f) { } 13
    movimientoY -= 0.5f;
    if (cuerpo3 >= 66.0f) {
        cuerpo3 -= 0.2f;
    }
}
else if (cambio == 13 && movimientoY < 2.0f) {
    cambio = 14;
}
else if (cambio == 14 && cuerpo3 >= 0.0f) { } 14
//Mov miembros
if (brazo5 < 30.0f && movBrazo5 == true) {
    brazo5 += 1.5f;
}
else if (brazo5 > 30.0f && movBrazo5 == true) {
    movBrazo5 = false;
}
else if (brazo5 >= -30.0f && movBrazo5 == false) {
    brazo5 -= 1.5f;
}
else if (brazo5 < -30.0f && movBrazo5 == false) {
    movBrazo5 = true;
}
```

Figura 46: Animación del avatar parte 5

Al terminar el último movimiento, se reinician las variables para regresar al estado inicial, y se llama a la función 'setTim' para reiniciar la variable booleana con un valor falso, de forma que el usuario debe presionar la tecla '7' nuevamente para ver la animación otra vez.



```
//Traslacion
cuerpo3 -= 0.25f;
movimientoY -= 0.003;
}
else if (cambio == 14 && cuerpo3 < 0.0f) {
    cambio=15;
}
else if (cambio == 15 && rotCuerpo3 > 0.0f) {
    rotCuerpo3 -= 5.0f;
}
else if (cambio == 15 && rotCuerpo3 <= 0.0f) {
    pierna5 = 0.0f;
    pierna6 = 0.0f;
    brazo5 = 0.0f;
    brazo6 = 0.0f;
    cuerpo3 = 0.0f;
    movimientoY = 0.0f;
    cambio = 1;
    mainWindow.setTim(false);
}
else{
    //Musica
    Tim.pause();
}
```

Figura 47: Animación del avatar parte 6

Finalmente, se toman los valores de las variables modificadas en la animación para cambiar las transformaciones del avatar.

```
//Personaje
//cuerpo
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-12.0f, movimientoY, -17.0f+ cuerpo3));
model = glm::rotate(model, rotCuerpo3 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(1.2f, 1.6f, 0.8f));
color = glm::vec3(1.0f, 1.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
TimTexture.UseTexture();
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
meshList[4]->RenderMesh();
```

Figura 48: Cambio de traslación del avatar

```
//Hombro izquierdo
model = modelaux;
model = glm::translate(model, glm::vec3(-0.775f, 0.5f, 0.0f));
model = glm::rotate(model, brazo5* toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
```

Figura 49: Rotación del hombro

## 2.6. Resultados

El personaje en el escenario se muestra a continuación, pero la animación se aprecia mejor en el video.



Figura 50: Avatar: Timothy Jackson Drake

### 3. Recorrido

El movimiento por el escenario se define de acuerdo a la cámara que se utilice.

Como el escenario fue un exterior, se utilizaron dos cámaras diferentes, una ligada al piso y otra aérea.

```
Camera cameraPiso;  
Camera cameraAerea;
```

Figura 51: Instancias para las cámaras

De acuerdo con la clase Camera, se configuraron los siguientes valores iniciales para cada una de las cámaras:

- Posición de la cámara
- Orientación hacia arriba.
- Sesgo en eje 'x'.
- Sesgo en eje 'y'.
- Velocidad de Movimiento
- Velocidad de giro.

```
cameraPiso = Camera(glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 1.0f, 0.0f), -50.0f, 0.0f, 0.5f, 0.5f);  
cameraAerea = Camera(glm::vec3(0.0f, 320.0f, 0.0f), glm::vec3(0.0f, 1.0f, 0.0f), 0.0f, -90.0f, 2.0f, 0.5f);
```

Figura 52: configuración inicial para las cámaras



### 3.1. Cámara aérea

Para la cámara aérea se declaró una posición inicial en el centro de coordenadas, pero a una altura en la que se pueden observar todos los demás objetos del escenario. Su orientación inicial es hacia el eje 'y' positivo, sin embargo, el sesgo agregado en el eje 'y' de -90 unidades da la sensación de estar observando hacia abajo.

Cabe mencionar que la velocidad de esta cámara es más rápida que la cámara de piso, con la intención de poder seguir algunos elementos del escenario, como al dirigible.

### 3.2. Cámara ligada al piso en tercera persona

La cámara ligada al piso tiene una posición inicial en el centro de coordenadas, su orientación es con el eje 'y' positivo y solo maneja un sesgo en el eje 'x', puesto que si se añade un sesgo en el eje 'y' es probable que el usuario termine viendo algún elemento por debajo del piso durante su recorrido.

### 3.3. Cambio de cámaras

Para el cambio de cámaras se utilizó una variable booleana que cambie de valor de acuerdo a las teclas '1' y '2'.

```
GLboolean getCamAerea() { return CamAerea; }
GLboolean getSpotlights() { return spotlights; }
GLboolean getAnimacionBatmobile() { return animacionBatmobile; }
GLboolean getTim() { return Tim; }
GLboolean getHelicoptero() { return helicoptero; }
void setTim(bool value) { Tim = value; }
bool getShouldClose() {
    return glfwWindowShouldClose(mainWindow);
}
bool* getsKeys() { return keys; }
void swapBuffers() { return glfwSwapBuffers(mainWindow); }

~Window();
vate:
GLFWwindow *mainWindow;
GLint width, height;
bool keys[1024];
GLint bufferWidth, bufferHeight;
void callbacks();
GLfloat lastX;
GLfloat lastY;
GLfloat xChange;
GLfloat yChange;
GLfloat muevex;
GLboolean CamAerea, spotlights, animacionBatmobile;
```

Figura 53: Cambio en Window.h



```
        if (key == GLFW_KEY_1)
    {
        theWindow->CamAerea = false;
    }
    if (key == GLFW_KEY_2)
    {
        theWindow->CamAerea = true;
    }
```

Figura 54: Cambio en Window.cpp

Además, las funciones que calculan la posición de las cámaras, su vista y el control tanto del ratón como del teclado son llamadas de acuerdo con el valor de la variable auxiliar, de forma que se obtiene la cámara de piso con la tecla '1' y la aérea con '2'.

```
if (mainWindow.getCamAerea()) {
    cameraAerea.keyControl(mainWindow.getKeys(), deltaTime, mainWindow.getCamAerea());
    cameraAerea.mouseControl(mainWindow.getXChange());
}
else {
    cameraPiso.keyControl(mainWindow.getKeys(), deltaTime, mainWindow.getCamAerea());
    cameraPiso.mouseControl(mainWindow.getXChange());
}
```

Figura 55: Control de la cámara

```
if (mainWindow.getCamAerea()) {
    glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(cameraAerea.calculateViewMatrix()));
    glUniform3f(uniformEyePosition, cameraAerea.getCameraPosition().x, cameraAerea.getCameraPosition().y, cameraAerea.getCameraPosition().z);
}
else {
    glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(cameraPiso.calculateViewMatrix()));
    glUniform3f(uniformEyePosition, cameraPiso.getCameraPosition().x, cameraPiso.getCameraPosition().y, cameraPiso.getCameraPosition().z);
}
```

Figura 56: Posición de la cámara

### 3.4. Control del movimiento

Al método 'Key Control' se le tiene que mandar la información de cuál cámara se está utilizando para poder seleccionar hacia donde se hace el movimiento con teclas.

Debido a las funciones de 'update' y 'calculateViewMatrix' de la clase cámara, se obtienen automáticamente los ejes de coordenadas de acuerdo a la posición y orientación actual de la misma, por lo que los valores de las variables 'derecha', 'enfrente' y 'arriba' se actualizan sin problemas, sin embargo, la cámara de piso debe poder moverse hacia 'enfrente' y hacia la 'derecha', mientras que la cámara aérea debe poder moverse hacia 'arriba' y hacia la 'derecha'.



```
void Camera::mouseControl(GLfloat xChange)
{
    xChange *= turnSpeed;

    yaw += xChange;
    update();
}

glm::mat4 Camera::calculateViewMatrix()
{
    return glm::lookAt(position, position + front, up);
}

glm::vec3 Camera::getCameraPosition()
{
    return position;
}

glm::vec3 Camera::getCameraDirection()
{
    return glm::normalize(front);
}

void Camera::update()
{
    front.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch));
    front.y = sin(glm::radians(pitch));
    front.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));
    front = glm::normalize(front);
    right = glm::normalize(glm::cross(front, worldUp));
    up = glm::normalize(glm::cross(right, front));
}
```

Figura 57: Actualización automática

Debido a lo anterior, se usa el valor de la variable booleana para decidir el movimiento de las cámaras.

- Tecla W: Mueve la cámara hacia enfrente en la cámara de piso y hacia arriba en la cámara aérea.
- Tecla A: Mueve la cámara hacia la izquierda ('derecha' negativo) en cualquiera de las cámaras.
- Tecla S: Mueve la cámara hacia atrás ('enfrente' negativo) en la cámara de piso y hacia abajo ('arriba' negativo) en la cámara aérea.
- Tecla D: Mueve la cámara hacia la derecha en cualquiera de las cámaras.

```
void Camera::keyControl(bool* keys, GLfloat deltaTime, GLboolean aerea)
{
    GLfloat velocity = moveSpeed * deltaTime;

    if ((keys[GLFW_KEY_W]) && (aerea == true))
    {
        position += up * velocity;
    }
    else if (keys[GLFW_KEY_W])
    {
        position += front * velocity;
    }

    if ((keys[GLFW_KEY_S]) && (aerea == true))
    {
        position -= up * velocity;
    }
    else if (keys[GLFW_KEY_S])
    {
        position -= front * velocity;
    }

    if (keys[GLFW_KEY_A])
    {
        position -= right * velocity;
    }
    if (keys[GLFW_KEY_D])
    {
        position += right * velocity;
    }
}
```

Figura 58: Movimiento con teclado de acuerdo con la cámara usada

Además, se puede mencionar que el hecho de utilizar dos instancias para la cámara permite el correcto almacenamiento de la posición previa en las cámaras.

### 3.5. Resultados

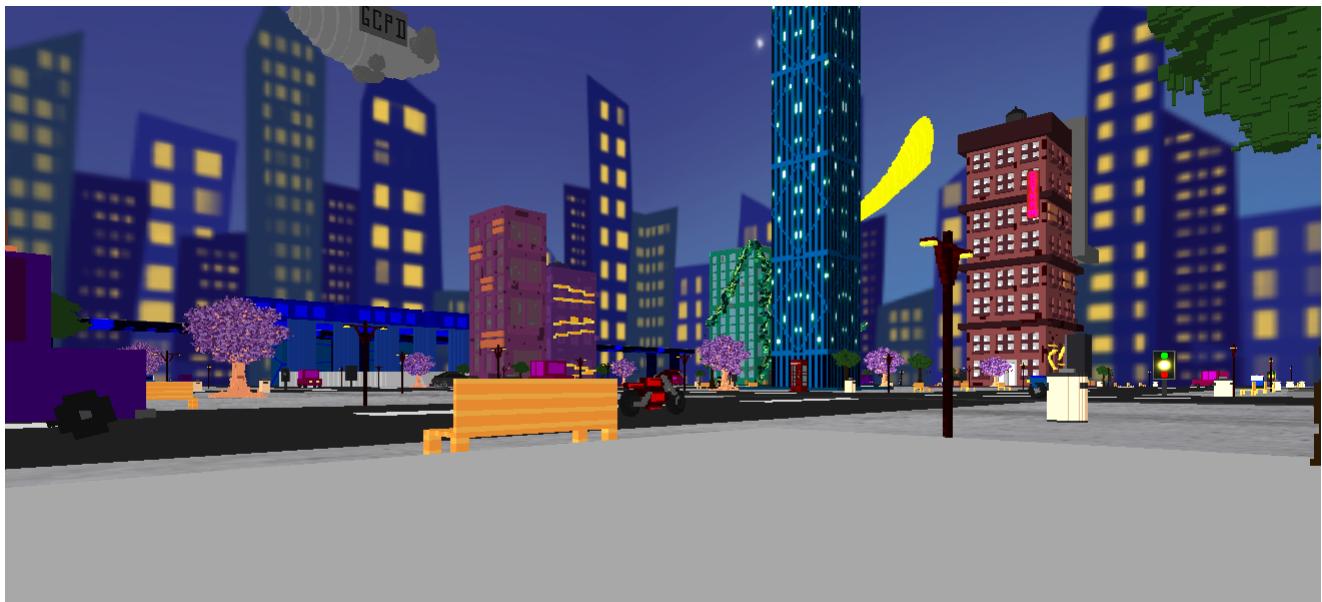


Figura 59: Cámara de piso

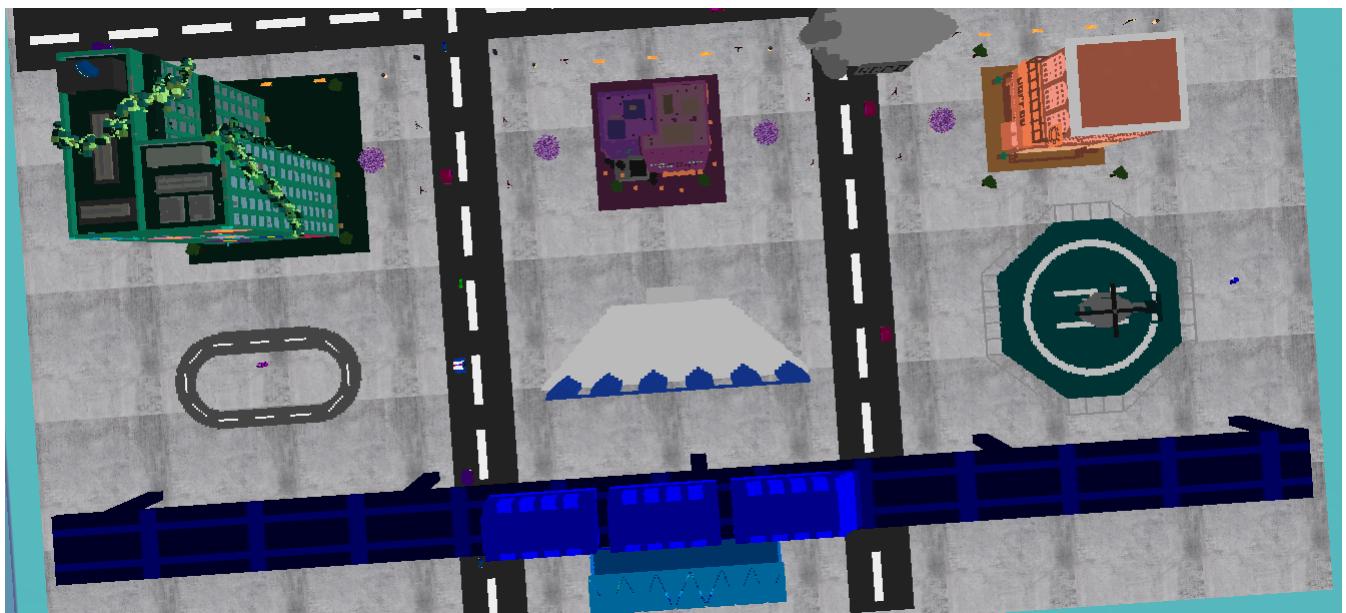


Figura 60: Cámara aérea



## 4. Iluminación

### 4.1. Ciclo día-noche

Para realizar el cambio, se utilizaron algunas variables auxiliares, un contador 'count' y dos límites, una variable que establezca el límite de la duración del día (2000 unidades) y otra del ciclo completo de día-noche (4000 unidades). Los cambios con base al ciclo se realizan con las variables CicloDia y CicloDiaNoche, mientras que la variable count incrementa su valor en una unidad mientras el programa está abierto y se reinicia al alcanzar el valor de CicloDiaNoche.

```
unsigned int count = 0, countSem=0, cicloDia=2000, cicloDiaNoche=4000;
```

Figura 61: Variables auxiliares

#### 4.1.1. Cambio de Skybox

Para este cambio de Skybox fue necesario tener los dos modelos correspondientes, almacenar las 6 caras de cada uno de ellos y generar dos instancias de la clase Skybox con sus correspondientes caras.

```
//skyboxes
std::vector<std::string> skyboxFaces;
skyboxFaces.push_back("Textures/Skybox/cupertin-lake_rt.tga");
skyboxFaces.push_back("Textures/Skybox/cupertin-lake_lf.tga");
skyboxFaces.push_back("Textures/Skybox/cupertin-lake_dn.tga");
skyboxFaces.push_back("Textures/Skybox/cupertin-lake_up.tga");
skyboxFaces.push_back("Textures/Skybox/cupertin-lake_bk.tga");
skyboxFaces.push_back("Textures/Skybox/cupertin-lake_ft.tga");

skybox = Skybox(skyboxFaces);

std::vector<std::string> skyboxFaces2;
skyboxFaces2.push_back("Textures/Skybox/cupertin-lake_rt_noche.tga");
skyboxFaces2.push_back("Textures/Skybox/cupertin-lake_lf_noche.tga");
skyboxFaces2.push_back("Textures/Skybox/cupertin-lake_dn_noche.tga");
skyboxFaces2.push_back("Textures/Skybox/cupertin-lake_up_noche.tga");
skyboxFaces2.push_back("Textures/Skybox/cupertin-lake_bk_noche.tga");
skyboxFaces2.push_back("Textures/Skybox/cupertin-lake_ft_noche.tga");

skybox2 = Skybox(skyboxFaces2);
```

Figura 62: Skyboxes en código

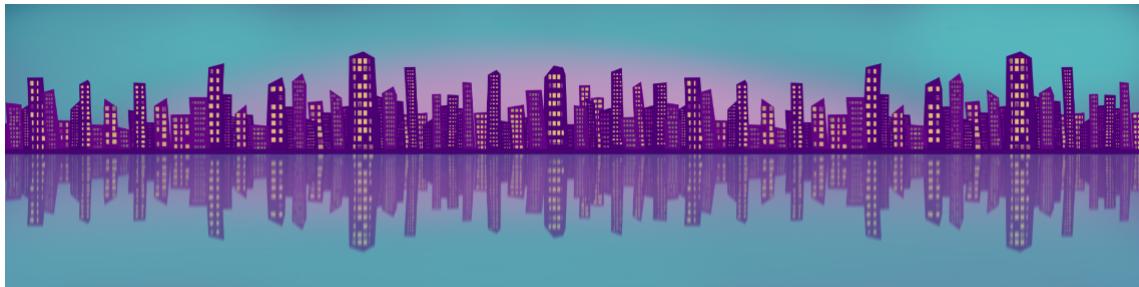


Figura 63: Skyboxes en imagen

El cambio de Skybox se realiza dentro del ciclo While del programa, mientras la ventana principal siga abierta, sin embargo, no solo se tienen 2 casos, sino que se tienen que considerar los cambios de cámara también, por lo que la función 'DrawSkybox' tiene 4 combinaciones posibles de acuerdo a la cámara actual y al valor de nuestro contador para saber si es de día o noche.

```
if ((count < cicloDia) && (mainWindow.getCamAerea())) {  
    skybox.DrawSkybox(cameraAerea.calculateViewMatrix(), projection);  
} else if ((count < cicloDia) && (mainWindow.getCamAerea()==false)) {  
    skybox.DrawSkybox(cameraPiso.calculateViewMatrix(), projection);  
    //skybox.DrawSkybox(camera.calculateViewMatrix(), projection);  
} else if ((count >= cicloDia) && (mainWindow.getCamAerea())) {  
    skybox2.DrawSkybox(cameraAerea.calculateViewMatrix(), projection);  
} else {  
    skybox2.DrawSkybox(cameraPiso.calculateViewMatrix(), projection);  
    //skybox2.DrawSkybox(camera.calculateViewMatrix(), projection);  
}  
count++;  
if (count >= cicloDiaNoche) {  
    count = 0;  
}
```

Figura 64: Cambio de Skybox

#### 4.1.2. Iluminación con base al ciclo

Algunas luces como la ligada a la batiseñal y la del dirigible, se activan únicamente durante la noche, por lo que es necesario saber el valor de la variable 'count' para declararlas o no.



```
if (count >= cicloDia) {  
    //luz direccional  
    mainLight = DirectionalLight(1.0f, 1.0f, 1.0f,  
        0.9f, 0.1f,  
        -1.0f, 0.0f, 0.0f);  
  
    if (mainWindow.getHelicoptero()) {  
        //Luces puntuales  
  
        //Dirigible  
        pointLights[0] = PointLight(0.0f, 0.0f, 1.0f,  
            0.9f, 10.0f,  
            posXd, 0.0f, posZd,  
            0.3f, 0.2f, 0.1f);  
  
        pointLights[0].SetPos(glm::vec3(posXd + movd_x, 0.0f, posZd + movd_y));  
  
        //Batiseñal  
        pointLights[1] = PointLight(1.0f, 1.0f, 0.0f,  
            0.9f, 20.0f,  
            -192.0f, 82.8f, 46.0f,  
            // -170.0f, 104.0f, 66.0f,  
            0.3f, 0.2f, 0.1f);  
  
        //Helicoptero  
        pointLights[2] = PointLight(1.0f, 0.0f, 0.0f,  
            0.9f, 10.0f,  
            (posXh + movh_x) + 10.0f, 5.0f, posZh,  
            0.3f, 0.2f, 0.1f);  
  
        pointLights[2].SetPos(glm::vec3((posXh + movh_x) + 10.0f, 5.0f, posZh));  
  
        pointLightCount = 3;  
    }  
}
```

Figura 65: Luces con el ciclo de noche

```
else {  
    mainLight = DirectionalLight(1.0f, 1.0f, 1.0f,  
        1.0f, 0.1f,  
        -1.0f, 0.0f, 0.0f);  
  
    if (mainWindow.getHelicoptero()) {  
  
        //Helicoptero  
        pointLights[0] = PointLight(1.0f, 0.0f, 0.0f,  
            0.9f, 10.0f,  
            (posXh + movh_x) + 10.0f, 5.0f, posZh,  
            0.3f, 0.2f, 0.1f);  
  
        pointLights[0].SetPos(glm::vec3((posXh + movh_x) + 10.0f, 5.0f, posZh));  
  
        pointLightCount = 1;  
    }  
    else {  
        //Luces puntuales  
        pointLightCount = 0;  
    }  
}
```

Figura 66: Luces con el ciclo de día

#### 4.1.3. Cambio de texturas con base al ciclo

Debido a que Magica Voxel permite renderizar algunos elementos, se pueden agregar luces a los modelos de esa forma, sin embargo, estos modelos no se pueden importar.

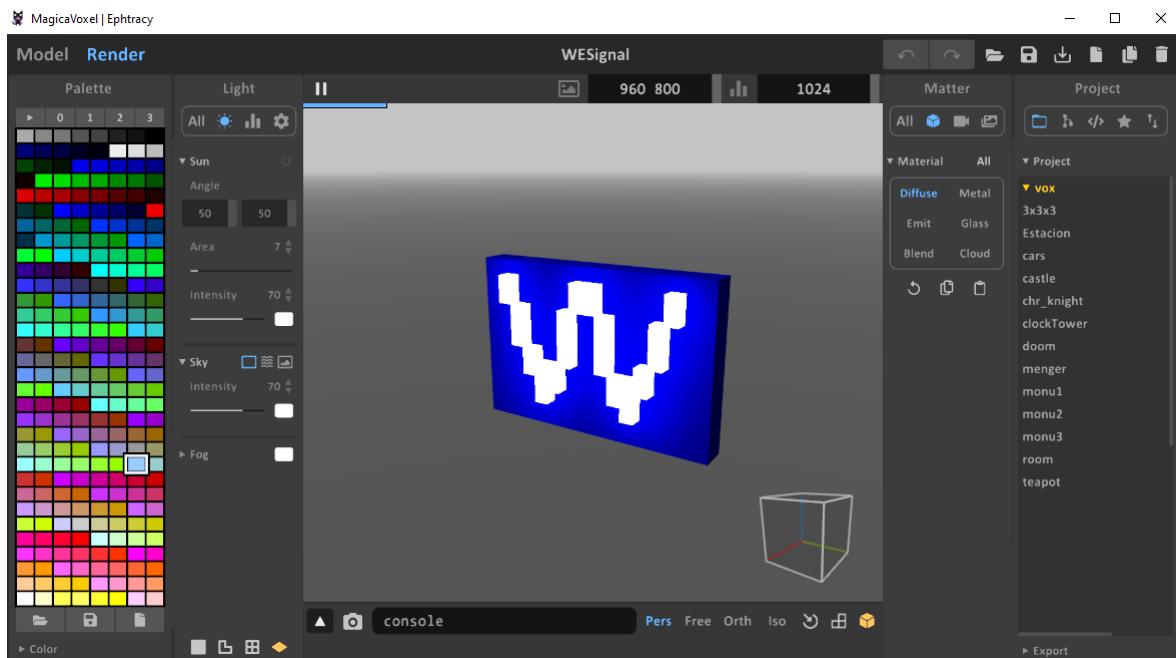


Figura 67: Renderizado en Magica Voxel

La solución para obtener más elementos de iluminación y cumplir el apartado de 'elementos de los edificios que se enciendan de acuerdo al ciclo de día noche' incluido en la propuesta del proyecto, tomé capturas de algunos objetos en Magica Voxel, con lo que hice texturas de algunos de los objetos con y sin luz.

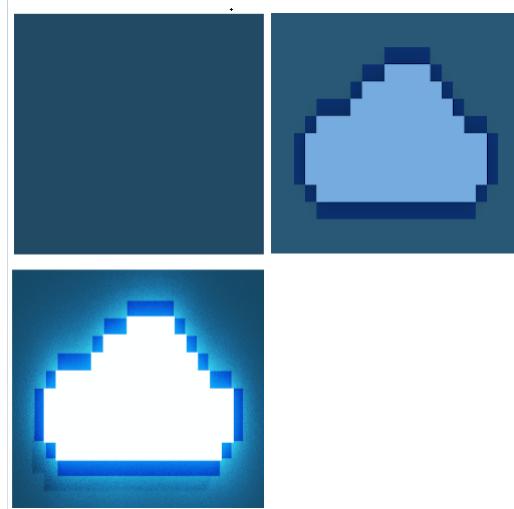


Figura 68: Texturas creadas

Después, utilizando Blender, asigné las texturas correspondientes como lo hicimos en clase, solo que la apliqué a dos objetos, uno con luz y otro sin luz.



Finalmente, modifiqué en el código el modelo que se mostraba cuando era de día o de noche en el programa, logrando el objeto de luminarias adicionales de algunas señales del escenario, la base del tren, los semáforos, entre otros.

```
//Señal
model = modelaux;
model = glm::translate(model, glm::vec3(10.5f, 58.8f, 21.5f));
model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(1.0f, 3.5f, 3.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
if (count < cicloDia) {
    CloudSignal_M.RenderModel();
}
else {
    CloudSignal_luz_M.RenderModel();
}
```

Figura 69: Cambio de modelo de acuerdo al ciclo día-noche

## 4.2. Iluminación por teclado

Al igual que otros objetos de este proyecto, y como se mencionó anteriormente en otros apartados, se agregó una variable auxiliar booleana para encender y apagar la luz ligada al helicóptero con las teclas '9' y '0', realizando los cambios correspondientes en la clase Window.

```
if (key == GLFW_KEY_9)
{
    theWindow->helicoptero = true;
}
if (key == GLFW_KEY_0)
{
    theWindow->helicoptero = false;
}
```

Figura 70: Teclas de cambio en Window.cpp

Con ayuda de esta variable, pueden cambiar las declaraciones de las luces, por ejemplo, en el ciclo de día, la luz puntual del helicóptero es la única que se declara si la variable es verdadera, mientras que no se declara ninguna luz de este tipo con la variable negativa (en el ciclo de noche).

```
if (mainWindow.getHelicoptero()) {

    //Helicoptero
    pointLights[0] = PointLight(1.0f, 0.0f, 0.0f,
        0.9f, 10.0f,
        (posXh + movh_x) + 10.0f, 5.0f, posZh,
        0.3f, 0.2f, 0.1f);

    pointLights[0].SetPos(glm::vec3((posXh + movh_x) + 10.0f, 5.0f, posZh));

    pointLightCount = 1;
}
else {
    //Luces puntuales
    pointLightCount = 0;
}
```

Figura 71: Declaración de la luz del helicóptero durante el día



Por otra parte, en el ciclo de noche, se deben declarar las luces correspondientes al dirigible y la batiseñal siempre, solo se modifica la declaración de la luz del helicóptero dependiendo de si el valor de su variable es verdadero o falso, por lo que se pueden tener 2 o 3 luces de tipo puntual durante la noche.

```
if (mainWindow.getHelicoptero()) {  
    //Luces puntuales  
  
    //Dirigible  
    pointLights[0] = PointLight(0.0f, 0.0f, 1.0f,  
        0.9f, 10.0f,  
        posXd, 0.0f, posZd,  
        0.3f, 0.2f, 0.1f);  
  
    pointLights[0].SetPos(glm::vec3(posXd + movd_x, 0.0f, posZd + movd_y));  
  
    //Batiseñal  
    pointLights[1] = PointLight(1.0f, 1.0f, 0.0f,  
        0.9f, 20.0f,  
        -192.0f, 82.8f, 46.0f,  
        // -170.0f, 104.0f, 66.0f,  
        0.3f, 0.2f, 0.1f);  
  
    //Helicóptero  
    pointLights[2] = PointLight(1.0f, 0.0f, 0.0f,  
        0.9f, 10.0f,  
        (posXh + movh_x) + 10.0f, 5.0f, posZh,  
        0.3f, 0.2f, 0.1f);  
  
    pointLights[2].SetPos(glm::vec3((posXh + movh_x) + 10.0f, 5.0f, posZh));  
  
    pointLightCount = 3;  
}
```

Figura 72: Declaración de la luz del helicóptero durante el día (helicóptero: verdadero)

```
else {  
    //Luces puntuales  
  
    //Dirigible  
    pointLights[0] = PointLight(0.0f, 0.0f, 1.0f,  
        0.9f, 10.0f,  
        posXd, 0.0f, posZd,  
        0.3f, 0.2f, 0.1f);  
  
    pointLights[0].SetPos(glm::vec3(posXd + movd_x, 0.0f, posZd + movd_y));  
  
    //Batiseñal  
    pointLights[1] = PointLight(1.0f, 1.0f, 0.0f,  
        0.9f, 20.0f,  
        -192.0f, 82.8f, 46.0f,  
        // -170.0f, 104.0f, 66.0f,  
        0.3f, 0.2f, 0.1f);  
  
    pointLightCount = 2;  
}
```

Figura 73: Declaración de la luz del helicóptero durante el día (helicóptero: falso)



### 4.3. Show de luces

Para el show de luces, también se deben activar de acuerdo con las teclas '3' y '4', por lo que se realizaron los cambios correspondientes en Window.h y Window.cpp

```
if (key == GLFW_KEY_3)
{
    theWindow->spotlights = true;
}
if (key == GLFW_KEY_4)
{
    theWindow->spotlights = false;
```

Figura 74: Teclas de inicialización del show de luces

En el método principal se declaran las luces spotlights solo si el valor de la variable auxiliar que se activa con la tecla '3' es verdadera.

```
//Show de luces
if (mainWindow.getSpotlights()) {
    //luz 1
    spotLights[0] = SpotLight(0.0f, 1.0f, 1.0f,
        1.0f, 2.0f,
        -35.0f, 20.0f, 160.0f,
        0.0f, -1.0f, 0.0f,
        1.0f, 0.0f, 0.0f,
        20.0f);

    //luz 2
    spotLights[1] = SpotLight(1.0f, 0.0f, 1.0f,
        1.0f, 2.0f,
        0.0f, 15.0f, 160.0f,
        0.0f, -1.0f, 0.0f,
        1.0f, 0.0f, 0.0f,
        20.0f);

    //luz 3
    spotLights[2] = SpotLight(1.0f, 1.0f, 0.0f,
        1.0f, 2.0f,
        35.0f, 25.0f, 160.0f,
        0.0f, -1.0f, 0.0f,
        1.0f, 0.0f, 0.0f,
        20.0f);

    spotLightCount = 3;
```

Figura 75: Inicialización de luces para el show

Para la animación de este ciclo se usan diferentes funciones y variables auxiliares, así como una variable auxiliar (luces) que controle el orden de la animación.

Los cambios 1 y 2 son movimientos senoidales independientes de las luces de los extremos, los cambios 3, 4 y 5 son los movimientos senoidales en conjunto de las luces de los extremos, mientras que el movimiento 6 y 7 es el movimiento en forma de espiral de la luz central.



```
xluz2 = (3 + 2 * movLuz * toRadians) * cos(movLuz * toRadians);
zluz2 = (3 + 2 * movLuz * toRadians) * sin(movLuz * toRadians);
if (xluz1<=75.0f && luces==1) {
    xluz1 += 0.2f;
    zluz1 += 1.0f;
}
else if (xluz1 > 75.0f && luces == 1) {
    luces = 2;
    xluz1 = 0.0f;
}
else if (xluz3 <= 75.0f && luces == 2) {
    xluz3 += 0.2f;
    zluz3 += 1.0f;
}
else if (xluz3 > 75.0f && luces == 2) {
    luces = 3;
    xluz3 = 0.0f;
}
else if (xluz1 <= 75.0f && luces == 3) {
    xluz1 += 0.2f;
    zluz1 += 1.0f;
}
else if (xluz1 > 75.0f && luces == 3) {
    luces = 4;
}
else if (xluz3 <= 75.0f && luces == 4) {
    xluz3 += 0.2f;
    zluz3 += 1.0f;
    xluz1 -= 0.2f;
    zluz1 += 1.0f;
}
else if (xluz3 > 75.0f && luces == 4) {
    luces = 5;
}
else if (xluz3 >= 0.0f && luces == 5) {
    xluz3 -= 0.2f;
    zluz3 += 1.0f;
    xluz1 += 0.2f;
    zluz1 += 1.0f;
}
```

Figura 76: Animación del show parte 1

```
else if (xluz3 < 0.0f && luces == 5) {
    luces = 6;
    xluz1 = 0.0f;
    xluz3 = 0.0f;
}
else if (movLuz <= 360.0f && luces == 6) {
    movLuz += 1.0f;
}
else if (movLuz > 360.0f && luces == 6) {
    luces = 7;
}
else if (movLuz >= 0.0f && luces == 7) {
    movLuz -= 1.0f;
}
else if (movLuz < 0.0f && luces == 7) {
    luces = 1;
    xluz1 = 0.0f;
    xluz2 = 0.0f;
    xluz3 = 0.0f;
    zluz1 = 0.0f;
    zluz2= 0.0f;
    zluz3 = 0.0f;
    movLuz = 0.0f;
}

if (zluz1 > 360.0f) {
    zluz1 = 0.0f;
}
if (zluz1 < -360.0f) {
    zluz1 = 0.0f;
}
if (zluz3 > 360.0f) {
    zluz3 = 0.0f;
}
if (zluz3 < -360.0f) {
    zluz3 = 0.0f;
}
```

Figura 77: Animación del show parte 2

Finalmente, se llama al método 'setPos' de las spotlights para modificar su posición de acuerdo

a las variables auxiliares de la animación.

```
spotLights[0].SetPos(glm::vec3(-35.0f + xluz1, 11.0f, 160.0f + 6.0f * cos(zluz1 * toRadians)));
spotLights[1].SetPos(glm::vec3(0.0f + xluz2, 11.0f, 160.0f + zluz2));
spotLights[2].SetPos(glm::vec3(35.0f - xluz3, 11.0f, 160.0f + 6.0f * cos(zluz3 * toRadians))');
```

Figura 78: Animación del show parte 3

#### 4.4. Resultados

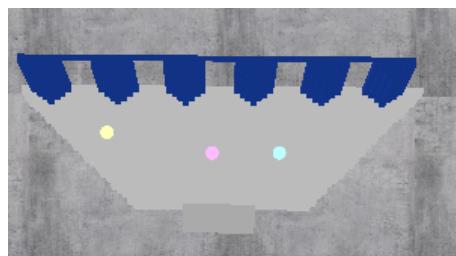


Figura 79: Vista aérea del show de luces



Figura 80: Vista aérea de la batiseñal

### 5. Animación

Para todas las animaciones se utilizaron variables auxiliares.



```

//Movimiento ave (compleja automática)
float xbird = 0.0f, ybird = 0.0f, zbird = 0.0f, rotBird = 0.0f, rotWings = 0.0f, movbird = 0.0f;
float countBird = 0.0f;
bool wings = true, sube = true;

//Movimiento batmobile (simple por teclado)
float movBatmobile = 0.0f, movBatmobile2 = 0.0f, rotllanta = 0.0f, rotBatmobile = 180.0f;
int batmobile = 1;

//Movimiento dirigible (compleja *Keyframe)
float movDirigible = 0.0f, a = 0.0f, b = 0.0f, countDirigible = 0.0f;

//Movimiento show de luces (compleja por teclado)
float xluz1 = 0.0f, xluz1 = 0.0f, xluz2 = 0.0f, xluz2 = 0.0f, xluz3 = 0.0f, xluz3 = 0.0f, movLuz=0.0f;
int luces = 1;

//Movimiento batiseñal (simple automática)
float rotBatsignal =0.0f, escalaBatsignal=1.0f;

//Movimiento RedHood (compleja) automática
float xRh= 125.0f, yRh= 0.0f, zRh= -250.0f;
float brazo1 = 0.0f, brazo2 = 0.0f, pierna1 = 0.0f, pierna2 = 0.0f, cuerpo1 = 0.0f, rotCuerpo = 0.0f;
bool movBrazo = true, movBrazo2 = false, movPierna1=false, movPierna2=true;
int countDisp = 0;
int cambioJ = 1;

//Movimiento Nightwing (compleja) automática
float xNw = -230.0f, yNw = 0.3f, zNw = 22.0f;
float brazo3 = 0.0f, brazo4 = 0.0f, pierna3 = 0.0f, pierna4 = 0.0f, cuerpo2 = 0.0f, movY=0.0f, rotCuerpo2 = 0.0f, movZ=0.0f;
bool movBrazo3 = true, movBrazo4 = false, movPierna3 = false, movPierna4 = true, movCuerpo2 = true, salto = false;
int countSalto = 0;
int cambioN = 1;

//Movimiento Tim (simple) por teclado
float brazo5 = 0.0f, brazo6 = 0.0f, pierna5 = 0.0f, pierna6 = 0.0f, cuerpo3 = 0.0f, rotCuerpo3=0.0f;
float staffScale = 1.0f, staffRot = 0.0f, movimientoY = 0.0f, ganchoTras = 0.0f, ganchoRot = 0.0f, ganchoMovZ=0.0f;
bool movBrazo5 = true, movBrazo6 = false, movPierna5 = false, movPierna6 = true, movCuerpo3 = true;
int cambio = 1;

//Variables semáforo
int verde1 = 200, amarillo1 = 300, rojo1 = 600;
int verde2 = 300, amarillo2 = 500;
int cambioAuto = 1;

```

Figura 81: Variables para animación

## 5.1. Básica

### 5.1.1. Batmobile: Movimiento por teclado

Este modelo se instanció con las llantas formando una jerarquía de modelos, de forma que se pudieran mover junto al batmobile, pero también pudieran rodar de forma independiente.

Al igual que en otros objetos, se utilizó una variable auxiliar booleana para que el movimiento empiece con la tecla '5' y se detenga con la tecla '6'.

```
if (key == GLFW_KEY_5)
{
    theWindow->animacionBatmobile = true;
}
if (key == GLFW_KEY_6)
{
    theWindow->animacionBatmobile = false;
}
```

Figura 82: Teclas para la animación

Para la animación, también se utilizó una variable auxiliar que lleva el control del movimiento, en este caso solo se realizan desplazamiento en los ejes 'x' y 'z', así como rotaciones sobre el eje



del batmobile y las rotaciones de las llantas.

```
if(mainWindow.getAnimacionBatmobile()){  
    if (movBatmobile <= 160.0f && batmobile == 1) {  
        movBatmobile += 0.25f;  
        rotllanta += 0.1f;  
        rotBatmobile = 180.0f;  
    }  
    else if (movBatmobile > 160.0f && batmobile == 1) {  
        batmobile = 2;  
        rotBatmobile = 180.0f;  
    }  
    else if (movBatmobile2 <= 108.0f && batmobile == 2) {  
        movBatmobile2 += 0.25f;  
        rotllanta += 0.1f;  
        rotBatmobile = 90.0f;  
    }  
    else if (movBatmobile2 > 108.0f && batmobile == 2) {  
        batmobile = 3;  
        rotBatmobile = 90.0f;  
    }  
    else if (movBatmobile >= 0.0f && batmobile == 3) {  
        movBatmobile -= 0.25f;  
        rotllanta += 0.1f;  
        rotBatmobile = 0.0f;  
    }  
    else if (movBatmobile < 0.0f && batmobile == 3) {  
        batmobile = 4;  
        rotBatmobile = 0.0f;  
    }  
    else if (movBatmobile2 >=0.0f && batmobile == 4) {  
        movBatmobile2 -= 0.25f;  
        rotllanta += 0.1f;  
        rotBatmobile = -90.0f;  
    }  
    else if (movBatmobile2 < 0.0f && batmobile == 4) {  
        batmobile = 1;  
        rotBatmobile = -90.0f;  
    }  
}
```

Figura 83: Animación del batmobile

También se declararon los límites de las variables que sirven para los giros, de forma que no lleguen a valores muy altos durante la ejecución del programa.

```
if (rotllanta > 359.0f) {  
    rotllanta = 0.0f;  
}  
if (rotllanta < -359.0f) {  
    rotllanta = 0.0f;  
}
```

Figura 84: Límites para variables

Finalmente, se usaron las variables que realizan los incrementos y decrementos para modificar las transformaciones del modelo, como se muestra a continuación.



```
//batmobile sin movimiento
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(78.0f - movBatmobile, 0.8f, -5.5f - movBatmobile2));
model = glm::rotate(model, rotBatmobile * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
modelaux = model;
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
Batmobile_M.RenderModel();

//Llanta delantera derecha
model = modelaux;
model = glm::translate(model, glm::vec3(-3.44f, -1.4f, -1.6f));
model = glm::rotate(model, -rotllanta, glm::vec3(0.0f, 0.0f, 1.0f));
color = glm::vec3(1.0f, 1.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
llanta_B_M.RenderModel();

//Llanta posterior derecha
model = modelaux;
model = glm::translate(model, glm::vec3(3.1f, -1.4f, -1.6f));
model = glm::rotate(model, -rotllanta, glm::vec3(0.0f, 0.0f, 1.0f));
color = glm::vec3(1.0f, 1.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
llanta_B_M.RenderModel();

//Llanta delantera izq
model = modelaux;
model = glm::translate(model, glm::vec3(-3.44f, -1.4f, 1.6f));
model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, rotllanta, glm::vec3(0.0f, 0.0f, 1.0f));
color = glm::vec3(1.0f, 1.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
llanta_B_M.RenderModel();
```

Figura 85: Transformaciones de traslación y rotación.

### 5.1.2. Batiseñal: Con base al ciclo de día-noche

Este modelo cambia cuando es de noche para aparentar que se 'encendió' la señal, además, realiza una rotación y un escalamiento durante la noche.

```
//Batiseñal
model = modelaux;
model = glm::translate(model, glm::vec3(-12.0f, 85.5f, -24.0f));
model = glm::scale(model, glm::vec3(1.5f, 1.5f, 1.5f));
model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, rotBatsignal * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(escalaBatsignal, escalaBatsignal, escalaBatsignal));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
if (count < cicloDia) {
    Batiseñal1_M.RenderModel();
    if (escalaBatsignal > 1.0f) {
        escalaBatsignal -= 0.001f;
    }
}
else {
    Batiseñal2_M.RenderModel();
    rotBatsignal += 1.0f;
    if (escalaBatsignal < 2.0f) {
        escalaBatsignal += 0.001f;
    }
}

if (rotBatsignal > 359.0f) {
    rotBatsignal = 0.0f;
}
if (rotBatsignal < -359.0f) {
    rotBatsignal = 0.0f;
}
```

Figura 86: Animación de la batiseñal

### 5.1.3. Transportes de las calles

El control de estos movimientos fue algo complicado, además de que se necesitaron variables para cada uno de los transportes utilizados y una variable extra para el movimiento de sus



llantas, además del posicionamiento con jerarquía de modelos.

```
//Animacion de los autos y motos de la calle.  
float movAuto1 = 0.0f, rotLlanta1 = 0.0f;  
float movAuto2 = 0.0f, rotLlanta2 = 0.0f;  
float movAuto3 = 0.0f, rotLlanta3 = 0.0f;  
float movAuto4 = 0.0f, rotLlanta4 = 0.0f;  
float movAuto5 = 0.0f, rotLlanta5 = 0.0f;  
float movAuto6 = 0.0f, rotLlanta6 = 0.0f;  
float movAuto7 = 0.0f, rotLlanta7 = 0.0f;  
float movAuto8 = 0.0f, rotLlanta8 = 0.0f;  
float movAuto9 = 0.0f, rotLlanta9 = 0.0f;  
float movAuto10 = 0.0f, rotLlanta10 = 0.0f;  
float movAuto11 = 0.0f, rotLlanta11 = 0.0f;  
float movAuto12 = 0.0f, rotLlanta12 = 0.0f;  
float movAuto13 = 0.0f, rotLlanta13 = 0.0f;  
float movAuto14 = 0.0f, rotLlanta14 = 0.0f;  
float movAuto15 = 0.0f, rotLlanta15 = 0.0f;  
float movAuto16 = 0.0f, rotLlanta16 = 0.0f;  
float movAuto17 = 0.0f, rotLlanta17 = 0.0f;  
float movAuto18 = 0.0f, rotLlanta18 = 0.0f;  
float moto1 = 0.0f, rotLlantaMoto1 = 0.0f;  
float moto2 = 0.0f, rotLlantaMoto2 = 0.0f;  
float moto3 = 0.0f, rotLlantaMoto3 = 0.0f;  
float moto4 = 0.0f, rotLlantaMoto4 = 0.0f;  
float moto5 = 0.0f, rotLlantaMoto5 = 0.0f;  
float moto6 = 0.0f, rotLlantaMoto6 = 0.0f;  
float moto7 = 0.0f, rotLlantaMoto7 = 0.0f;  
float moto8 = 0.0f, rotLlantaMoto8 = 0.0f;
```

Figura 87: Variables para animación

Esta animación es constante, por lo que la condición para detener o continuar los movimientos es el valor utilizado en los semáforos, coordinándolos para que los transportes avancen cuando su carril tiene verde o amarillo, pero se detengan cuando tengan rojo. De acuerdo al control, el primer movimiento es el que realizan los vehículos en los carriles de la izquierda y la derecha del avatar, por lo que se incrementan los valores de forma constante para todos los modelos.

```
//Carril izq  
if (-555.0f + movAuto7 == 30.0f) {  
    movAuto7 = 0.0f;  
    //printf("\nSe reinicio mov7");  
}  
movAuto7 += 0.5f;  
rotLlanta7 += 5.0f;  
  
if (-695.0f + movAuto8 == -110.0f) {  
    movAuto8 = 0.0f;  
    //printf("\nSe reinicio mov8");  
}  
movAuto8 += 0.5f;  
rotLlanta8 += 5.0f;  
  
if (-455.0f + movAuto9 == 130.0f) {  
    movAuto9 = 0.0f;  
    //printf("\nSe reinicio mov9");  
}  
movAuto9 += 0.5f;  
rotLlanta9 += 5.0f;  
  
if (-345.0f + movAuto10 == 240.0f) {  
    movAuto10 = 0.0f;  
    //printf("\nSe reinicio mov10");  
}  
movAuto10 += 0.5f;  
rotLlanta10 += 5.0f;  
  
if (-825.0f + movAuto11 == -240.0f) {  
    movAuto11 = 0.0f;  
    //printf("\nSe reinicio mov11");  
}  
movAuto11 += 0.5f;  
rotLlanta11 += 5.0f;  
  
if (-687.0f + movAuto12 == -22.0f) {  
    movAuto12 = 0.0f;  
    //printf("\nSe reinicio mov12");  
}  
movAuto12 += 0.5f;  
rotLlanta12 += 5.0f;  
  
if (-755.0f + moto5 == -60.0f) {  
    moto5 = 0.0f;  
    //printf("\nSe reinicio moto5");  
}  
moto5 += 0.5f;  
rotLlantaMoto5 += 5.0f;  
  
if (-515.0f + moto6 == 70.0f) {  
    moto6 = 0.0f;  
    //printf("\nSe reinicio moto6");  
}  
moto6 += 0.5f;  
rotLlantaMoto6 += 5.0f;  
  
if (-415.0f + moto7 == 170.0f) {  
    moto7 = 0.0f;  
    //printf("\nSe reinicio moto7");  
}  
moto7 += 0.5f;  
rotLlantaMoto7 += 5.0f;  
  
if (-755.0f + moto8 == -170.0f) {  
    moto8 = 0.0f;  
    //printf("\nSe reinicio moto8");  
}  
moto8 += 0.5f;  
rotLlantaMoto8 += 5.0f;
```

Figura 88: Incrementos para el carril izquierdo



Cabe mencionar que las condiciones que se agregan son para reiniciar el valor de las variables que realizan las traslaciones cuando llegan nuevamente a su posición inicial, esto para evitar el desbordamiento de las mismas debido a que el movimiento de todos los vehículos es constante durante todo el programa.

```
if ((countSem < amarillo1 + 1) && (cambioAuto == 1)) {
    //Movimiento izq y derecha

    //Carril derecho
    if (625.0f - movAuto1 == 40.0f) {
        movAuto1 = 0.0f;
        //printf("\nSe reinicio mov1");
    }
    movAuto1 += 0.5f;
    rotLlanta1 += 5.0f;

    if (485.0f - movAuto2 == -100.0f) {
        movAuto2 = 0.0f;
        //printf("\nSe reinicio mov2");
    }
    movAuto2 += 0.5f;
    rotLlanta2 += 5.0f;

    if (705.0f - movAuto3 == 120.0f) {
        movAuto3 = 0.0f;
        //printf("\nSe reinicio mov3");
    }
    movAuto3 += 0.5f;
    rotLlanta3 += 5.0f;

    if (825.0f - movAuto4 == 240.0f) {
        movAuto4 = 0.0f;
        //printf("\nSe reinicio mov4");
    }
    movAuto4 += 0.5f;
    rotLlanta4 += 5.0f;

    if (345.0f - movAuto5 == -240.0f) {
        movAuto5 = 0.0f;
        //printf("\nSe reinicio mov5");
    }
    movAuto5 += 0.5f;
    rotLlanta5 += 5.0f;

    if (563.0f - movAuto6 == -22.0f) {
        movAuto6 = 0.0f;
        //printf("\nSe reinicio mov6");
    }
    movAuto6 += 0.5f;
    rotLlanta6 += 5.0f;

    if (525.0f - moto1 == -60.0f) {
        moto1 = 0.0f;
        //printf("\nSe reinicio moto1");
    }
    moto1 += 0.5f;
    rotLlantaMoto1 += 5.0f;

    if (655.0f - moto2 == 70.0f) {
        moto2 = 0.0f;
        //printf("\nSe reinicio moto2");
    }
    moto2 += 0.5f;
    rotLlantaMoto2 += 5.0f;

    if (755.0f - moto3 == 170.0f) {
        moto3 = 0.0f;
        //printf("\nSe reinicio moto3");
    }
    moto3 += 0.5f;
    rotLlantaMoto3 += 5.0f;

    if (415.0f - moto4 == -170.0f) {
        moto4 = 0.0f;
        //printf("\nSe reinicio moto4");
    }
    moto4 += 0.5f;
    rotLlantaMoto4 += 5.0f;
```

Figura 89: Incrementos para el carril derecho

```
else if ((countSem == amarillo1 + 1) && (cambioAuto == 1)) {
    cambioAuto = 2;
    //printf("Val:%f", 40.0f - movAuto1);
}
```

Figura 90: Control de cambio para el siguiente movimiento

Después, continua el movimiento de los carriles de enfrente y atrás del avatar.



```
else if ((countSem > amarillo1 + 1 && countSem < rojo1) && (cambioAuto == 2)) {  
    //Mov enfrente y atrás  
  
    //Carril atrás  
    if (605.0f - movAuto13 == 20.0f) {  
        movAuto13 = 0.0f;  
        //printf("\nSe reinicio mov13");  
    }  
    movAuto13 += 0.5f;  
    rotLlanta13 += 5.0f;  
  
    if (755.0f - movAuto14 == 170.0f) {  
        movAuto14 = 0.0f;  
        //printf("\nSe reinicio mov14");  
    }  
    movAuto14 += 0.5f;  
    rotLlanta14 += 5.0f;  
  
    if (455.0f - movAuto15 == -130.0f) {  
        movAuto15 = 0.0f;  
        //printf("\nSe reinicio mov15");  
    }  
    movAuto15 += 0.5f;  
    rotLlanta15 += 5.0f;
```

Figura 91: Incrementos para el carril de atrás

```
//Carril enfrente  
if (-565.0f + movAuto16 == 20.0f) {  
    movAuto16 = 0.0f;  
    //printf("\nSe reinicio mov16");  
}  
movAuto16 += 0.5f;  
rotLlanta16 += 5.0f;  
  
if (-415.0f + movAuto17 == 170.0f) {  
    movAuto17 = 0.0f;  
    //printf("\nSe reinicio mov17");  
}  
movAuto17 += 0.5f;  
rotLlanta17 += 5.0f;  
  
if (-715.0f + movAuto18 == -130.0f) {  
    movAuto18 = 0.0f;  
    //printf("\nSe reinicio mov18");  
}  
movAuto18 += 0.5f;  
rotLlanta18 += 5.0f;
```

Figura 92: Incrementos para el carril de enfrente

```
else if ((countSem > amarillo1 + 1 && countSem == rojo1) && (cambioAuto == 2)) {  
    cambioAuto = 1;
```

Figura 93: Control de cambio para regresar al primer movimiento



Y se establecen los límites para las variables de las rotaciones.

```
    if (rotLlanta14 >= 360.0f) {
        rotLlanta14 = 0.0f;
    }
    if (rotLlanta15 >= 360.0f) {
        rotLlanta15 = 0.0f;
    }
    if (rotLlanta16 >= 360.0f) {
        rotLlanta16 = 0.0f;
    }
    if (rotLlanta17 >= 360.0f) {
        rotLlanta17 = 0.0f;
    }
    if (rotLlanta18 >= 360.0f) {
        rotLlanta18 = 0.0f;
    }
    if (rotLlantaMoto1 >= 360.0f) {
        rotLlantaMoto1 = 0.0f;
    }
    if (rotLlantaMoto2 >= 360.0f) {
        rotLlantaMoto2 = 0.0f;
    }
    if (rotLlantaMoto3 >= 360.0f) {
        rotLlantaMoto3 = 0.0f;
    }
    if (rotLlantaMoto4 >= 360.0f) {
        rotLlantaMoto4 = 0.0f;
    }
```

Figura 94: Límites

Finalmente, se modifican las traslaciones de los objetos de acuerdo a las variables que realizan los incrementos. Cabe mencionar que se tomaron en cuenta dos casos diferentes para la traslación, uno de ellos es el primer movimiento del vehículo hasta que llega al final del escenario, y el segundo de ellos es para comenzar desde el otro extremo del escenario, para tener un movimiento constante.

Debido a lo anterior, fue necesario realizar diferentes operaciones y asignarles valores diferentes a cada vehículo de acuerdo a su posición inicial.

```
//Auto x - 420 -285
model = modelaux;
if (120.0f - movAuto3 >= -300.0f) {
    model = glm::translate(model, glm::vec3(9.0f, 1.2f, 120.0f - movAuto3));
}
else if (120.0f - movAuto3 < -300.0f) {
    model = glm::translate(model, glm::vec3(9.0f, 1.2f, 705.0f - movAuto3));
}
model = glm::scale(model, glm::vec3(1.2f, 1.2f, 1.2f));
modelaux2 = model;
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Auto2_M.RenderModel();

//Llanta delantera der
model = modelaux2;
model = glm::translate(model, glm::vec3(1.4f, -0.4f, 1.5f));
model = glm::rotate(model, rotLlanta3 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Auto_llanta.RenderModel();

//Llanta delantera izq
model = modelaux2;
model = glm::translate(model, glm::vec3(-1.65f, -0.4f, -1.5f));
model = glm::rotate(model, -rotLlanta3 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Auto_llanta.RenderModel();

//Llanta posterior der
model = modelaux2;
model = glm::translate(model, glm::vec3(1.4f, -0.4f, 1.7f));
model = glm::rotate(model, rotLlanta3 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Auto_llanta.RenderModel();

//Llanta posterior izq
model = modelaux2;
model = glm::translate(model, glm::vec3(-1.65f, -0.4f, 1.5f));
model = glm::rotate(model, -rotLlanta3 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Auto_llanta.RenderModel();
```

Figura 95: Modelo de auto en el carril derecho



```
//Moto x + 345 = -300
model = modelaux;
if (-60.0f + moto5 <= 285.0f) {
    model = glm::translate(model, glm::vec3(-6.0f, 0.7f, -60.0f + moto5));
}
else if (-60.0f + moto5 > 285.0f) {
    model = glm::translate(model, glm::vec3(-6.0f, 0.7f, -645.0f + moto5));
}
model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
modelaux2 = model;
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Moto3_M.RenderModel();

//llanta delantera
model = modelaux2;
model = glm::translate(model, glm::vec3(0.0f, 0.05f, 0.72f));
model = glm::rotate(model, rotLlantaMoto5 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Moto3_llanta.RenderModel();

//llanta trasera
model = modelaux2;
model = glm::translate(model, glm::vec3(0.0f, 0.05f, -0.75f));
model = glm::rotate(model, rotLlantaMoto5 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Moto3_llanta.RenderModel();
```

Figura 96: Modelo de moto en el carril izquierdo

```
//Auto x - 320 =285
model = modelaux;
if (20.0f - movAuto13 >= -300.0f) {
    model = glm::translate(model, glm::vec3(20.0f - movAuto13, -0.3f, -9.0f));
}
else if (20.0f - movAuto13 < -300.0f) {
    model = glm::translate(model, glm::vec3(605.0f - movAuto13, -0.3f, -9.0f));
}
model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
modelaux2 = model;
model = glm::scale(model, glm::vec3(2.5f, 2.5f, 2.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Patrulla_M.RenderModel();

//llanta delantera der
model = modelaux2;
model = glm::scale(model, glm::vec3(1.2f, 1.2f, 1.2f));
model = glm::translate(model, glm::vec3(1.6f, 1.0f, -1.55f));
model = glm::rotate(model, -rotLlanta13 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Auto_llanta.RenderModel();

//llanta delantera izq
model = modelaux2;
model = glm::scale(model, glm::vec3(1.2f, 1.2f, 1.2f));
model = glm::translate(model, glm::vec3(-1.6f, 1.0f, -1.57f));
model = glm::rotate(model, rotLlanta13 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Auto_llanta.RenderModel();
```

Figura 97: Modelo de patrulla en el carril de atrás



```
//Auto x + 115 = -300
model = modelaux;
if (170.0f + movAuto17 <= 285.0f) {
    model = glm::translate(model, glm::vec3(170.0f + movAuto17, 1.5f, 9.0f));
}
else if (170.0f + movAuto17 > 285.0f) {
    model = glm::translate(model, glm::vec3(-415.0f + movAuto17, 1.5f, 9.0f));
}
model = glm::scale(model, glm::vec3(1.2f, 1.2f, 1.2f));
model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
modelaux2 = model;
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Auto2_M.RenderModel();

//llanta delantera der
model = modelaux2;
model = glm::translate(model, glm::vec3(1.4f, -0.4f, -1.5f));
model = glm::rotate(model, -rotLlanta17 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Auto_llanta.RenderModel();

//llanta delantera izq
model = modelaux2;
model = glm::translate(model, glm::vec3(-1.65f, -0.4f, -1.5f));
model = glm::rotate(model, -rotLlanta17 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Auto_llanta.RenderModel();
```

Figura 98: Modelo de auto en el carril de enfrente

## 5.2. Compleja

### 5.2.1. Red Hood

Esta animación se activa cuando la cámara de piso se acerca a cierta distancia del modelo. Los movimientos que realiza es una traslación parabólica en el plano XZ, movimiento de todos sus miembros, alza su brazo y activa el sonido del disparo para simular el disparo de una de sus armas, da media vuelta y regresa a su posición original para repetir la animación.

```
if ((cameraPiso.getCameraPosition().x- (xRh+cuerpo1) <= 100.0f) && (cameraPiso.getCameraPosition().z -(zRh+(cuerpo1* cuerpo1)) <= 100.0f)) {
    //Musica
    music.pause();
    Tim.pause();
    Jason.play();
    Dick.pause();
    Disparo.pause();

    if (cuerpo1 <= 10.0f && cambioJ==1) {
        //Mov miembros
        if (brazo1 < 30.0f && movBrazo == true) {
            brazo1 += 1.5f;
        }
        else if (brazo1 > 30.0f && movBrazo == true) {
            movBrazo = false;
        }
        else if (brazo1 >= -30.0f && movBrazo == false) {
            brazo1 -= 1.5f;
        }
        else if (brazo1 < -30.0f && movBrazo == false) {
            movBrazo = true;
        }
    }
```

Figura 99: Animación de Red Hood parte 1



```
//Traslacion
cuerpo1 += 0.01f;
}
else if (cuerpo1 > 10.0f && cambioJ == 1) {
    cambioJ = 2;
}
else if (brazo1 >= -90.0f && cambioJ == 2) {
    brazo1 -= 1.5f;
}
else if (brazo1 < -90.0f && cambioJ == 2 && countDisp < 150) {
    pierna1 = 0.0f;
    pierna2 = 0.0f;
    brazo2 = 0.0f;
    Disparo.play();
    countDisp += 1;
}
else if (brazo1 < -90.0f && cambioJ == 2 && countDisp >= 150) {
    Disparo.pause();
    cambioJ = 3;
}
else if (rotCuerpo<=180.0f && cambioJ == 3) {
    rotCuerpo += 5.0f;
}
else if (rotCuerpo > 180.0f && cambioJ == 3) {
    cambioJ = 4;
}
else if (cuerpo1 > 0.0f && cambioJ == 4) {
    //Mov miembros
    if (brazo1 <= 30.0f && movBrazo == true) {
        brazo1 += 1.5f;
    }
    else if (brazo1 > 30.0f && movBrazo == true) {
        movBrazo = false;
    }
    else if (brazo1 >= -30.0f && movBrazo == false) {
        brazo1 -= 1.5f;
    }
    else if (brazo1 < -30.0f && movBrazo == false) {
        movBrazo = true;
    }
}
```

Figura 100: Animación de Red Hood parte 2

```
//Traslacion
cuerpo1 -= 0.01f;
}
else if (cuerpo1 <= 0.0f && cambioJ == 4) {
    cambioJ = 5;
}
else if (rotCuerpo > 0.0f && cambioJ == 5) {
    rotCuerpo -= 5.0f;
}
else if (rotCuerpo <= 0.0f && cambioJ == 5) {
    cambioJ = 6;
}
else if (rotCuerpo > 0.0f && cambioJ == 6) {
    rotCuerpo -= 5.0f;
}
else if (rotCuerpo <= 180.0f && cambioJ == 6) {
    pierna1 = 0.0f;
    pierna2 = 0.0f;
    brazo1 = 0.0f;
    brazo2 = 0.0f;
    cuerpo1 = 0.0f;
    countDisp = 0;
    cambioJ = 1;
}

//Musica
Jason.pause();
}
```

Figura 101: Animación de Red Hood parte 3



```
//RedHood
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(xRh, yRh, zRh));
model = glm::translate(model, glm::vec3(cuerpo1, 0.0f, cuerpo1 * cuerpo1));
model = glm::scale(model, glm::vec3(0.48f, 0.5f, 0.48f));
model = glm::rotate(model, rotCuerpo * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
modelaux2 = model;
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
Rh_Cuerpo_M.RenderModel();

//Brazo izq
model = modelaux2;
model = glm::translate(model, glm::vec3(-1.55f, 1.2f, 0.0f));
model = glm::rotate(model, brazo1 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
Rh_Brazo_M.RenderModel();

//Brazo der
model = modelaux2;
model = glm::translate(model, glm::vec3(1.55f, 1.2f, 0.0f));
model = glm::rotate(model, brazo2 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
Rh_Brazo_M.RenderModel();

//Pierna izq
model = modelaux2;
model = glm::translate(model, glm::vec3(-0.75f, -1.63f, 0.0f));
model = glm::rotate(model, pierna2 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
Rh_Pierna_M.RenderModel();

//Pierna der
model = modelaux2;
model = glm::translate(model, glm::vec3(0.75f, -1.63f, 0.0f));
model = glm::rotate(model, pierna1 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
Rh_Pierna_M.RenderModel();
```

Figura 102: Animación de Red Hood parte 4

### 5.2.2. Nightwing

Este personaje realiza un recorrido en el plano XZ, gira 90 grados, da un salto en forma parabólica, da media vuelta para regresar al inicio de donde comenzó el salto, gira 90 grados nuevamente y regresa a su posición original para reiniciar la animación.

```
if ((cameraPiso.getCameraPosition().x - (xNw + cuerpo2 * cuerpo2) <= 100.0f) && (cameraPiso.getCameraPosition().z - (zNw + (cuerpo2) <= 100.0f)) {
    //Musica
    music.pause();
    Tim.pause();
    Jason.pause();
    Dick.play();
    Disparo.pause();

    if (cuerpo2 <= 100.0f && cambioN == 1) {
        //Mov miembros
        if (brazo3 <= 30.0f && movBrazo3 == true) {
            brazo3 += 1.5f;
        }
        else if (brazo3 > 30.0f && movBrazo3 == true) {
            movBrazo3 = false;
        }
        else if (brazo3 >= -30.0f && movBrazo3 == false) {
            brazo3 -= 1.5f;
        }
        else if (brazo3 < -30.0f && movBrazo3 == false) {
            movBrazo3 = true;
        }
    }
```

Figura 103: Animación de Nightwing parte 1



```
//Traslación
cuerpo2 += 0.5f;
}
else if (cuerpo2 > 10.0f && cambioN == 1) {
pierna3 = 0.0f;
pierna4 = 0.0f;
cambioN = 2;
}
else if (rotCuerpo2 <= 90.0f && cambioN == 2) {
rotCuerpo2 += 5.0f;
}
else if (rotCuerpo2 > 90.0f && cambioN == 2) {
cambioN = 3;
}
else if (pierna3 >= -90.0f && pierna4 <= 90.0f && cambioN == 3) {
pierna3 -= 1.0f;
pierna4 += 1.0f;
movY += 0.1f;
movZ -= 0.01f;
}
else if (pierna3 < -90.0f && pierna4 > 90.0f && cambioN == 3) {
cambioN = 4;
}
else if (movY > 0.0f && cambioN == 4) {
pierna3 += 1.0f;
pierna4 -= 1.0f;
movY -= 0.1f;
movZ -= 0.01f;
}
else if (movY <= 0.0f && cambioN == 4) {
cambioN = 5;
/*printf("Z final:%f", movZ);*/
}
else if (rotCuerpo2 <= 270.0f && cambioN == 5) {
rotCuerpo2 += 5.0f;
}
else if (rotCuerpo2 > 0.0f && cambioN == 5) {
cambioN = 6;
}
```

Figura 104: Animación de Nightwing parte 2

```
else if (movZ <= 0.0f && cambioN == 6) {
//Mov miembros
if (brazo3 <= 30.0f && movBrazo3 == true) {
brazo3 += 1.5f;
}
else if (brazo3 > 30.0f && movBrazo3 == true) {
movBrazo3 = false;
}
else if (brazo3 >= -30.0f && movBrazo3 == false) {
brazo3 -= 1.5f;
}
else if (brazo3 < -30.0f && movBrazo3 == false) {
movBrazo3 = true;
}
}
```

Figura 105: Animación de Nightwing parte 3

```
//Traslación
movZ += 0.01f;
}
else if (movZ > 0.0f && cambioN == 6) {
cambioN = 7;
}
else if (rotCuerpo2 >= 180.0f && cambioN == 7) {
rotCuerpo2 -= 5.0f;
}
else if (rotCuerpo2 < 360.0f && cambioN == 7) {
cambioN = 8;
}
else if (cuerpo2 > 0.0f && cambioN == 8) {
//Mov miembros
if (brazo3 <= 30.0f && movBrazo3 == true) {
brazo3 += 1.5f;
}
else if (brazo3 > 30.0f && movBrazo3 == true) {
movBrazo3 = false;
}
else if (brazo3 >= -30.0f && movBrazo3 == false) {
brazo3 -= 1.5f;
}
else if (brazo3 < -30.0f && movBrazo3 == false) {
movBrazo3 = true;
}
}
```

Figura 106: Animación de Nightwing parte 4



```
//Traslacion
cuerpo2 -= 0.5f;
}
else if (cuerpo2 <= 0.0f && cambioN == 8) {
    cambioN=9;
}
else if (rotCuerpo2 > 0.0f && cambioN == 9) {
    rotCuerpo2 -= 5.0f;
}
else if (rotCuerpo2 <= 0.0f && cambioN == 9) {
    cambioN = 1;
    pierna3 = 0.0f;
    pierna4 = 0.0f;
    brazo3 = 0.0f;
    brazo4 = 0.0f;
    movY = 0.0f;
    cuerpo2 = 0.0f;
    rotCuerpo2 = 0.0f;

    Dick.pause();
}
```

Figura 107: Animación de Nightwing parte 5

```
//Nightwing
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(xNw,yNw,zNw));
model = glm::translate(model, glm::vec3(cuerpo2, 0.3f+ movY, -movZ*movZ));
model = glm::scale(model, glm::vec3(0.48f, 0.49f, 0.48f));
model = glm::rotate(model, rotCuerpo2 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
modelaux2 = model;
model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
Nw_Cuerpo_M.RenderModel();

//Brazo izq
model = modelaux2;
model = glm::translate(model, glm::vec3(0.0f, 0.2f, -1.55f));
model = glm::rotate(model, brazo3 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
Nw_Brazo_M.RenderModel();

//Brazo der
model = modelaux2;
model = glm::translate(model, glm::vec3(0.0f, 0.2f, 1.55f));
model = glm::rotate(model, brazo4 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
Nw_Brazo_M.RenderModel();

//Pierna izq
model = modelaux2;
model = glm::translate(model, glm::vec3(0.0f, -2.5f, -0.75f));
model = glm::rotate(model, pierna4 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
Nw_Pierna_M.RenderModel();

//Pierna der
model = modelaux2;
model = glm::translate(model, glm::vec3(0.0f, -2.5f, 0.75f));
model = glm::rotate(model, pierna3 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
```

Figura 108: Animación de Nightwing parte 6

### 5.2.3. Ave azul

Este modelo realiza un movimiento en forma de espiral en el plano XZ mientras realiza un movimiento lineal en el eje Y, además de rotar sus alas y su cuerpo. Al llegar a un punto en el movimiento en espiral, regresa con el mismo movimiento.

Cabe mencionar que se usó una variable auxiliar como contador para iniciar el movimiento



después de cierto tiempo de que se inició el programa, y no desde el comienzo como otros modelos.

```
xbird = (10 + 15 * movbird * toRadians) * cos(movbird * toRadians);
zbird = (10 + 15 * movbird * toRadians) * sin(movbird * toRadians);
if (countBird >= 500.0f) {
    if (sube == true && movbird <= 359.0f) {
        movbird += 0.2f;
        rotBird += 0.3f;
        ybird += 0.02f;
        if (rotWings <= 45.0f && wings == true) {
            rotWings += 0.75f;
            //printf("\nRotWings: %f\n", rotWings);
        }
        else if (rotWings > 45.0f && wings == true) {
            wings = false;
            //printf("\nCambio a False\n");
        }
        else if (rotWings >= -45.0f && wings == false) {
            rotWings -= 0.75f;
            //printf("\nRotWings: %f\n", rotWings);
        }
        else if (rotWings < -45.0f && wings == false) {
            wings = true;
            //printf("\nCambio a True\n");
        }
    }
    else if (sube == true && movbird > 359.0f) {
        sube = false;
    }
    else if (sube == false && movbird >= 0.0f) {
        movbird -= 0.2f;
        rotBird += 0.3f;
        ybird -= 0.02f;
        if (rotWings <= 45.0f && wings == true) {
            rotWings += 0.75f;
            //printf("\nRotWings: %f\n", rotWings);
        }
        else if (rotWings > 45.0f && wings == true) {
            wings = false;
            //printf("\nCambio a False\n");
        }
        else if (rotWings >= -45.0f && wings == false) {
            rotWings -= 0.75f;
            //printf("\nRotWings: %f\n", rotWings);
        }
    }
}
```

Figura 109: Animación del ave parte 1

```
else if (rotWings < -45.0f && wings == false) {
    wings = true;
    //printf("\nCambio a True\n");
}
else if (sube == false && movbird < 0.0f) {
    sube = true;
}
else {
    countBird += 1.0f;
}

if (rotBird > 359.0f) {
    rotBird = 0.0f;
}
if (rotBird < -359.0f) {
    rotBird = 0.0f;
}
```

Figura 110: Animación del ave parte 2



```
//Bird
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(180.0f - xbird, 56.0f + ybird, 70.0f - zbird));
model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, rotbird * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
modelaux = model;
MaterialNormal.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
BlueBird_M.RenderModel();

//alias
model = modelaux;
model = glm::translate(model, glm::vec3(0.0f, -0.1f, -0.2f));
model = glm::rotate(model, rotWings * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
MaterialNormal.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
BlueBirdWings_M.RenderModel();
```

Figura 111: Animación del ave parte 3

## 5.3. Técnica de animación

### 5.3.1. KeyFrames

Para esta técnica de animación se declararon las variables, estructuras, funciones y demás necesarias, tanto para el helicóptero como para el dirigible. Algo importante que debo mencionar es que cada variable requiere una variable auxiliar para realizar el cálculo de los incrementos en la interpolación.

```
//Animacion Dirigible
bool animacion_Dirigible = false;
float reproduciranimacion_Dirigible, habilitaranimacion_Dirigible, guardoFrame_Dirigible, reinicioFrame_Dirigible, ciclo_Dirigible, ciclo2_Dirigible;
float leoFrame_Dirigible = 0.0f, reinicioLeoFrame_Dirigible = 0.0f, contador_Dirigible = 0;
bool xdokay = false, ydokay = false, zdokay = false;

glm::vec3 posd = glm::vec3(0.0f, 0.0f, 0.0f);
float posXd = 70.0f, posYd = 110.0f, posZd = 60.0f;

//NEW// Keyframes
float movd_x = 0.0f, movd_y = 0.0f;
float girod = 0;

#define MAX_FRAMES_Dirigible 30 //Cuantos cuadros se guardan
int i_max_steps_Dirigible = 100; //Valores intermedios, entre mayor, mas suave se ve, pero ocupa mas recursos
int i_curr_steps_Dirigible = 2; //Número de frames declarados
typedef struct _frame_Dirigible { ... } FRAMED;

FRAMED KeyFrame_Dirigible[MAX_FRAMES_Dirigible];
int FrameIndex_Dirigible = 2; //introducir datos
bool play_Dirigible = false;
int playIndex_Dirigible = 0;

void saveFrame_Dirigible(void) { ... }

void readFrame_Dirigible(void) { ... }

void resetElements_Dirigible(void) { ... }

void interpolation_Dirigible(void) { ... }

void animate_Dirigible(void) { ... }
```

Figura 112: Declaraciones para la animación del dirigible

La Función para guardar los frames muestra en consola el mensaje correspondiente y almacena los valores en el formato indicado, de forma que se facilita la lectura del archivo de texto.



```
void readFrame_Dirigible(void)
{
    using namespace std;
    string aux = "";
    fstream fichero;
    char linea[40];
    fichero.open("Frames_Dirigible.txt", ios::in);
    if (fichero.fail()) {
        cerr << "Error al abrir el archivo Frames_Dirigible.txt" << endl;
    }
    else {
        fichero >> linea;           // Primera linea
        while (!fichero.eof())
        {
            if (FrameIndex_Dirigible < 10) {
                //cout << texto << endl;   // Muestrar el contenido en terminal
                if (linea[0] == 'K') {
                    if (linea[17] == 'x') {
                        aux = aux + linea[19] + linea[20] + linea[21] + linea[22] + linea[23] + linea[24] + linea[25];
                        movd_x = stof(aux);
                        aux = "";
                        zdokay = true;
                        cout << "movd_x =" + to_string(movd_x) << endl;
                    }
                    else if (linea[17] == 'y') {
                        aux = aux + linea[19] + linea[20] + linea[21] + linea[22] + linea[23] + linea[24] + linea[25];
                        movd_y = stof(aux);
                        aux = "";
                        zdokay = true;
                        cout << "movd_y =" + to_string(movd_y) << endl;
                    }
                    else if (linea[12] == 'g') {
                        aux = aux + linea[18] + linea[19] + linea[20] + linea[21] + linea[22] + linea[23] + linea[24];
                        girod = stof(aux);
                        aux = "";
                        zdokay = true;
                        cout << "girod =" + to_string(girod) << endl;
                    }
                }
            }
        }
    }
}

void saveFrame_Dirigible(void)
{
    using namespace std;
    printf("Se guardo el frame (%d)\n", FrameIndex_Dirigible);

    KeyFrame_Dirigible[FrameIndex_Dirigible].movd_x = movd_x;
    KeyFrame_Dirigible[FrameIndex_Dirigible].movd_y = movd_y;
    KeyFrame_Dirigible[FrameIndex_Dirigible].girod = girod;

    ofstream archivo("Frames_Dirigible.txt", ios::app);
    if (archivo.is_open())
    {
        archivo << "*****" << endl;
        archivo << "KeyFrame[" + std::to_string(FrameIndex_Dirigible) + "].movd_x=" + std::to_string(movd_x) << endl;
        archivo << "KeyFrame[" + std::to_string(FrameIndex_Dirigible) + "].movd_y=" + std::to_string(movd_y) << endl;
        archivo << "KeyFrame[" + std::to_string(FrameIndex_Dirigible) + "].girod=" + std::to_string(girod) << endl;
        /*archivo << "\ni_curr_steps = " + std::to_string(i_curr_steps) << endl;*/
        archivo.close();
    }
    else cerr << "Error de apertura del archivo." << endl;
    FrameIndex_Dirigible++;
}
```

Figura 113: Función para almacenar valores en archivo de texto

Después, se declara la función para leer dichos valores almacenados de acuerdo a la posición de los valores. Algo importante que hay que destacar es que, al tener un frame mayor a 9, las posiciones dentro del archivo de texto se modifican, por eso se tomaron dos casos de lectura, además de que no se contempló un caso en donde se tuviera una cantidad de frames de 3 cifras (ej. 100), debido a la cantidad de recursos que se utilizarían.



```
void saveFrame_Dirigible(void)
{
    using namespace std;
    printf("Se guarda el frame (%d)\n", FrameIndex_Dirigible);

    KeyFrame_Dirigible[FrameIndex_Dirigible].movd_x = movd_x;
    KeyFrame_Dirigible[FrameIndex_Dirigible].movd_y = movd_y;
    KeyFrame_Dirigible[FrameIndex_Dirigible].girod = girod;

    ofstream archivo("Frames_Dirigible.txt", ios::app);
    if (archivo.is_open()) {
        archivo << "*****" << endl;
        archivo << "KeyFrame[" + std::to_string(FrameIndex_Dirigible) + "].movd_x=" + std::to_string(movd_x) << endl;
        archivo << "KeyFrame[" + std::to_string(FrameIndex_Dirigible) + "].movd_y=" + std::to_string(movd_y) << endl;
        archivo << "KeyFrame[" + std::to_string(FrameIndex_Dirigible) + "].girod=" + std::to_string(girod) << endl;
        /*archivo << "\ni_curr_steps= " + std::to_string(i_curr_steps) << endl;*/
        archivo.close();
    }
    else cerr << "Error de apertura del archivo." << endl;
}

FrameIndex_Dirigible++;
}
```

Figura 114: Función para leer valores desde el archivo de texto

Se tiene una función auxiliar para regresar los valores a 0.

```
void resetElements_Dirigible(void)
{
    movd_x = KeyFrame_Dirigible[0].movd_x;
    movd_y = KeyFrame_Dirigible[0].movd_y;
    girod = KeyFrame_Dirigible[0].girod;
}
```

Figura 115: Función para resetear variables

Y se agrega la función que realiza el cálculo de la interpolación con 2 cuadros y el total de pasos que se declaró en un inicio.

```
void interpolation_Dirigible(void)
{
    KeyFrame_Dirigible[playIndex_Dirigible].movd_xInc = (KeyFrame_Dirigible[playIndex_Dirigible + 1].movd_x - KeyFrame_Dirigible[playIndex_Dirigible].movd_x) / i_max_steps_Dirigible;
    KeyFrame_Dirigible[playIndex_Dirigible].movd_yInc = (KeyFrame_Dirigible[playIndex_Dirigible + 1].movd_y - KeyFrame_Dirigible[playIndex_Dirigible].movd_y) / i_max_steps_Dirigible;
    KeyFrame_Dirigible[playIndex_Dirigible].girodInc = (KeyFrame_Dirigible[playIndex_Dirigible + 1].girod - KeyFrame_Dirigible[playIndex_Dirigible].girod) / i_max_steps_Dirigible;
}
```

Figura 116: Función para interpolación

La función que realiza el control de la animación es la siguiente.



```
void animate_Dirigible(void)
{
    //Movimiento del objeto
    if (play_Dirigible)
    {
        if (i_curr_steps_Dirigible >= i_max_steps_Dirigible) //end of animation between frames
        {
            playIndex_Dirigible++;
            printf("Frame (%d) reproducido\n", playIndex_Dirigible);
            if (playIndex_Dirigible > FrameIndex_Dirigible - 2) //end of total animation
            {
                printf("El ultimo frame es [%d]\n", FrameIndex_Dirigible);
                printf("Termina animacion\n");
                playIndex_Dirigible = 0;
                play_Dirigible = false;
            }
            else //Next frame interpolations
            {
                i_curr_steps_Dirigible = 0; //Reset counter
                //Interpolation
                interpolation_Dirigible();
            }
        }
        else
        {
            //Draw animation
            movd_x += KeyFrame_Dirigible[playIndex_Dirigible].movd_xInc;
            movd_y += KeyFrame_Dirigible[playIndex_Dirigible].movd_yInc;
            girod += KeyFrame_Dirigible[playIndex_Dirigible].girodInc;
            i_curr_steps_Dirigible++;
        }
    }
}
```

Figura 117: Función para animación

Esta función es llamada desde el método principal, durante el ciclo while:

```
//para keyframes
inputKeyframes(mainWindow.getsKeys());
animate_Helicopter();
animate_Dirigible();
```

Figura 118: Llamada a función para animación

Las mismas funciones se declaran con la terminación '\_Helicoptero', adaptándolas al modelo del helicóptero para sus movimientos. La mayor diferencia son las teclas que realizan los cambios.

### 5.3.2. Dirigible

En el método principal también se declaran los Frames iniciales del modelo, en este caso solo se usan dos.

```
//KEYFRAMES DECLARADOS INICIALES DIRIGIBLE

KeyFrame_Dirigible[0].movd_x = 5.0f;
KeyFrame_Dirigible[0].movd_y = -5.0f;
KeyFrame_Dirigible[0].girod = 0;

KeyFrame_Dirigible[1].movd_x = 15.0f;
KeyFrame_Dirigible[1].movd_y = -15.0f;
KeyFrame_Dirigible[1].girod = 0;
```

Figura 119: Cuadros iniciales del dirigible



A continuación se muestran las diferentes teclas que se asignaron para realizar los movimientos del dirigible.

```
//Dirigible
if (keys[GLFW_KEY_TAB])
{
    if (reproduciranimacion_Dirigible < 1)
    {
        if (play_Dirigible == false && (FrameIndex_Dirigible > 1))
        {
            resetElements_Dirigible();
            //First Interpolation
            interpolation_Dirigible();
            play_Dirigible = true;
            playIndex_Dirigible = 0;
            i_curr_steps_Dirigible = 0;
            reproduciranimacion_Dirigible++;
            printf("\nPresiona BACKSPACE para habilitar reproducir de nuevo la animacion'\n");
            habilitaranimacion_Dirigible = 0;
        }
        else
        {
            play_Dirigible = false;
        }
    }
}
if (keys[GLFW_KEY_BACKSPACE])
{
    if (habilitaranimacion_Dirigible < 1)
    {
        reproduciranimacion_Dirigible = 0;
    }
}
```

Figura 120: Teclas para reproducir animación

```
if (keys[GLFW_KEY_PERIOD])
{
    if (guardoFrame_Dirigible < 1)
    {
        saveFrame_Dirigible();
        printf("\nPresiona , (coma) para habilitar guardar otro frame\n");
        guardoFrame_Dirigible++;
        reinicioFrame_Dirigible = 0;
    }
}
if (keys[GLFW_KEY_COMMA])
{
    if (reinicioFrame_Dirigible < 1)
    {
        guardoFrame_Dirigible = 0;
    }
}
```

Figura 121: Teclas para guardar los valores en el archivo de texto



```
if (keys[GLFW_KEY_LEFT])
{
    if (ciclo_Dirigible < 1)
    {
        movd_x -= 5.0f;
        printf("movd_x es: %f\n", movd_x);
        ciclo_Dirigible++;
        ciclo2_Dirigible = 0;
        printf("reinicia con F1\n");
    }
}

if (keys[GLFW_KEY_RIGHT])
{
    if (ciclo_Dirigible < 1)
    {
        movd_x += 5.0f;
        printf("movd_x es: %f\n", movd_x);
        ciclo_Dirigible++;
        ciclo2_Dirigible = 0;
        printf("reinicia con F1\n");
    }
}
```

Figura 122: Teclas para movimiento en eje X

Cabe mencionar que el movimiento que controla la variable 'movd\_y' en realidad es sobre el eje Z, solo que nunca cambié el nombre.

```
if (keys[GLFW_KEY_UP])
{
    if (ciclo_Dirigible < 1)
    {
        movd_y -= 5.0f;
        printf("movd_y es: %f\n", movd_y);
        ciclo_Dirigible++;
        ciclo2_Dirigible = 0;
        printf("reinicia con F1\n");
    }
}

if (keys[GLFW_KEY_DOWN])
{
    if (ciclo_Dirigible < 1)
    {
        movd_y += 5.0f;
        printf("movd_y es: %f\n", movd_y);
        ciclo_Dirigible++;
        ciclo2_Dirigible = 0;
        printf("reinicia con F1\n");
    }
}
```

Figura 123: Teclas para movimiento en eje Z

```
if (keys[GLFW_KEY_G])
{
    if (ciclo_Dirigible < 1)
    {
        girod += 90.0f;
        printf("girod es: %f\n", girod);
        ciclo_Dirigible++;
        ciclo2_Dirigible = 0;
        printf("reinicia con F1\n");
    }
}

if (keys[GLFW_KEY_F])
{
    if (ciclo_Dirigible < 1)
    {
        girod -= 90.0f;
        printf("girod es: %f\n", girod);
        ciclo_Dirigible++;
        ciclo2_Dirigible = 0;
        printf("reinicia con F1\n");
    }
}
```

Figura 124: Teclas para giro



```
if (keys[GLFW_KEY_F1])
{
    if (ciclo2_Dirigible < 1)
    {
        ciclo_Dirigible = 0;
    }
}

if (keys[GLFW_KEY_INSERT])
{
    if (leoFrame_Dirigible < 1)
    {
        printf("\nLeyendo Frames\n");
        readFrame_Dirigible();
        printf("\nPresiona DELETE para habilitar leer nuevamente el archivo\n");
        leoFrame_Dirigible++;
        reinicioLeoFrame_Dirigible = 0;
    }
}
if (keys[GLFW_KEY_DELETE])
{
    if (reinicioLeoFrame_Dirigible < 1)
    {
        leoFrame_Dirigible = 0;
    }
}
```

Figura 125: Teclas para leer los valores desde el archivo de texto

	Frames_Dirigible	13/05/2022 01:08 p. m.	Documento de te...	2 KB
	Frames_Helicoptero	13/05/2022 01:06 p. m.	Documento de te...	2 KB

Figura 126: Archivos generados por el programa

### 5.3.3. Helicóptero

```
//KEYFRAMES DECLARADOS INICIALES HELICOPTERO

KeyFrame_Helicopter[0].movh_x = 0;
KeyFrame_Helicopter[0].movh_y = 0;
KeyFrame_Helicopter[0].movh_z = 0;
KeyFrame_Helicopter[0].giroh = 0;
KeyFrame_Helicopter[0].rot_helice = 45.0f;

KeyFrame_Helicopter[1].movh_x = 0.0f;
KeyFrame_Helicopter[1].movh_y = 0.0f;
KeyFrame_Helicopter[1].movh_z = 0.0f;
KeyFrame_Helicopter[1].giroh = 0;
KeyFrame_Helicopter[1].rot_helice = 180.0f;
```

Figura 127: Frames iniciales



```
void inputKeyframes(bool* keys)
{
    //Helicoptero
    if (keys[GLFW_KEY_SPACE])
    {
        if (reproduciranimacion_Helicopter < 1)
        {
            if (play_Helicopter == false && (FrameIndex_Helicopter > 1))
            {
                resetElements_Helicopter();
                //First Interpolation
                interpolation_Helicopter();
                play_Helicopter = true;
                playIndex_Helicopter = 0;
                i_curr_steps_Helicopter = 0;
                reproduciranimacion_Helicopter++;
                printf("\nPresiona Enter para habilitar reproducir de nuevo la animación\n");
                habilitaranimacion_Helicopter = 0;
            }
            else
            {
                play_Helicopter = false;
            }
        }
    }
    if (keys[GLFW_KEY_ENTER])
    {
        if (habilitaranimacion_Helicopter < 1)
        {
            reproduciranimacion_Helicopter = 0;
        }
    }
}
```

Figura 128: Teclas para reproducir animación

```

if (keys[GLFW_KEY_F2])
{
    if (guardoFrame_Helicopter < 1)
    {
        saveFrame_Helicopter();
        printf("\nPresiona F3 para habilitar guardar otro frame\n");
        guardoFrame_Helicopter++;
        reinicioFrame_Helicopter = 0;
    }
}
if (keys[GLFW_KEY_F3])
{
    if (reinicioFrame_Helicopter < 1)
    {
        guardoFrame_Helicopter = 0;
    }
}
if (keys[GLFW_KEY_L])
{
    if (ciclo_Helicopter < 1)
    {
        movh_x -= 30.0f;
        printf("movh_x es: %f\n", movh_x);
        ciclo_Helicopter++;
        ciclo2_Helicopter = 0;
        printf("reinicia con F5\n");
    }
}
```

Figura 129: Teclas para guardar los valores en el archivo de texto y movimiento positivo en X



```
if (keys[GLFW_KEY_LEFT])
{
    if (ciclo_Dirigible < 1)
    {
        movd_x -= 5.0f;
        printf("movd_x es: %f\n", movd_x);
        ciclo_Dirigible++;
        ciclo2_Dirigible = 0;
        printf("reinicia con F1\n");
    }
}

if (keys[GLFW_KEY_RIGHT])
{
    if (ciclo_Dirigible < 1)
    {
        movd_x += 5.0f;
        printf("movd_x es: %f\n", movd_x);
        ciclo_Dirigible++;
        ciclo2_Dirigible = 0;
        printf("reinicia con F1\n");
    }
}
```

Figura 130: Teclas para movimiento en eje X

```
if (keys[GLFW_KEY_J])
{
    if (ciclo_Helicopter < 1)
    {
        movh_x += 30.0f;
        printf("movh_x es: %f\n", movh_x);
        ciclo_Helicopter++;
        ciclo2_Helicopter = 0;
        printf("reinicia con F5\n");
    }
}

if (keys[GLFW_KEY_I])
{
    if (ciclo_Helicopter < 1)
    {
        movh_y += 25.0f;
        printf("movh_y es: %f\n", movh_y);
        ciclo_Helicopter++;
        ciclo2_Helicopter = 0;
        printf("reinicia con F5\n");
    }
}

if (keys[GLFW_KEY_K])
{
    if (ciclo_Helicopter < 1)
    {
        movh_y -= 25.0f;
        printf("movh_y es: %f\n", movh_y);
        ciclo_Helicopter++;
        ciclo2_Helicopter = 0;
        printf("reinicia con F5\n");
    }
}
```

Figura 131: Teclas para movimiento en eje Y y movimiento negativo en X



```
if (keys[GLFW_KEY_N])
{
    if (ciclo_Helicopter < 1)
    {
        movh_z += 25.0f;
        printf("movh_z es: %f\n", movh_z);
        ciclo_Helicopter++;
        ciclo2_Helicopter = 0;
        printf("reinicia con F5\n");
    }
}
if (keys[GLFW_KEY_M])
{
    if (ciclo_Helicopter < 1)
    {
        movh_z -= 25.0f;
        printf("movh_z es: %f\n", movh_z);
        ciclo_Helicopter++;
        ciclo2_Helicopter = 0;
        printf("reinicia con F5\n");
    }
}
```

Figura 132: Teclas para movimiento en eje z

```
if (keys[GLFW_KEY_O])
{
    if (ciclo_Helicopter < 1)
    {
        giroh += 90.0f;
        printf("giroh es: %f\n", giroh);
        ciclo_Helicopter++;
        ciclo2_Helicopter = 0;
        printf("reinicia con F5\n");
    }
}
if (keys[GLFW_KEY_P])
{
    if (ciclo_Helicopter < 1)
    {
        giroh -= 90.0f;
        printf("giroh es: %f\n", giroh);
        ciclo_Helicopter++;
        ciclo2_Helicopter = 0;
        printf("reinicia con F5\n");
    }
}
if (keys[GLFW_KEY_U])
{
    if (ciclo_Helicopter < 1)
    {
        rot_helice += 90.0f;
        printf("rot_helice es: %f\n", rot_helice);
        ciclo_Helicopter++;
        ciclo2_Helicopter = 0;
        printf("reinicia con F5\n");
    }
}
```

Figura 133: Teclas para giro del helicóptero y rotación del hélice



```
if (keys[GLFW_KEY_Y])
{
    if (ciclo_Helicopter < 1)
    {
        rot_helice -= 90.0f;
        printf("rot_helice es: %f\n", rot_helice);
        ciclo_Helicopter++;
        ciclo2_Helicopter = 0;
        printf("reinicia con F5\n");
    }
}

if (keys[GLFW_KEY_F5])
{
    if (ciclo2_Helicopter < 1)
    {
        ciclo_Helicopter = 0;
    }
}

if (keys[GLFW_KEY_F6])
{
    if (leoFrame_Helicopter < 1)
    {
        printf("\nLeyendo Frames\n");
        readFrame_Helicopter();
        printf("\nPresiona F7 para habilitar leer nuevamente el archivo\n");
        leoFrame_Helicopter++;
        reinicioLeoFrame = 0;
    }
}
if (keys[GLFW_KEY_F7])
{
    if (reinicioLeoFrame < 1)
    {
        leoFrame_Helicopter = 0;
    }
}
```

Figura 134: Teclas para leer los valores desde el archivo de texto

```
//Helipuerto
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(190.0f, -2.0f, 160.0f));
color = glm::vec3(1.0f, 1.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
modelaux = model;
model = glm::scale(model, glm::vec3(8.0f, 5.0f, 8.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Helipuerto_M.RenderModel();

//punto de giro
model = glm::mat4(1.0);
posh = glm::vec3(posXh -10.0f, posYh, posZh);
model = glm::translate(model, posh);
model = glm::rotate(model, giroh * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));

//Helicóptero
model = glm::translate(model, glm::vec3(10.0f + movh_x, movh_y, +movh_z));
model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
modelaux2 = model;
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
helicoptero_M.RenderModel();

//helice
model = modelaux2;
model = glm::translate(model, glm::vec3(0.0f, 3.8f, 0.0f));
model = glm::rotate(model, rot_helice * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
MaterialNormal.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
helice_M.RenderModel();
```

Figura 135: Modificación de las transformaciones del modelo en código

Todos los resultados de esta sección se aprecian mejor desde el video.



## 6. Audio

### 6.1. Biblioteca implementada

La biblioteca de audio que se utilizó para este proyecto fue 'Bass', implementada al seguir un tutorial.

Los archivos para esta biblioteca se pueden conseguir desde la página oficial o desde un repositorio de GitHub de la persona que subió el tutorial de implementación.

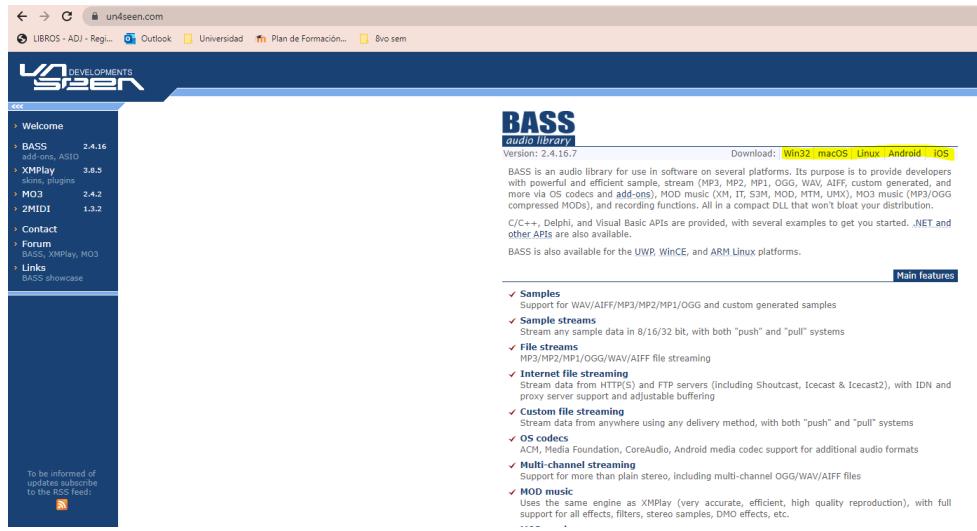


Figura 136: Descarga desde la página oficial

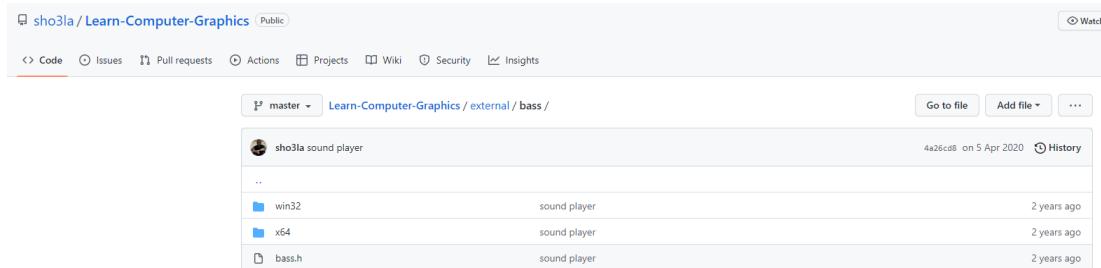


Figura 137: Descarga desde Github

#### 6.1.1. Modificaciones del proyecto

Para implementar la biblioteca fue necesario incluir los archivos correspondientes en las carpetas 'include' y 'lib'.



Nombre	Fecha de modificación	Tipo	Tamaño
assimp	24/04/2022 04:28 p. m.	Carpeta de archivos	
bass.h	24/06/2021 04:40 p. m.	C/C++ Header	44 KB
glew.h	23/03/2022 10:24 p. m.	C/C++ Header	104 KB

Figura 138: Archivo en 'include'

Nombre	Fecha de modificación	Tipo	Tamaño
assimp-vc140-mt.lib	23/03/2022 10:24 p. m.	Object File Library	310 KB
bass.lib	24/04/2022 04:23 p. m.	Object File Library	25 KB
glew32.lib	23/03/2022 10:24 p. m.	Object File Library	696 KB

Figura 139: Archivo en 'lib'

Además, se añadió la biblioteca en la configuración de la solución del proyecto

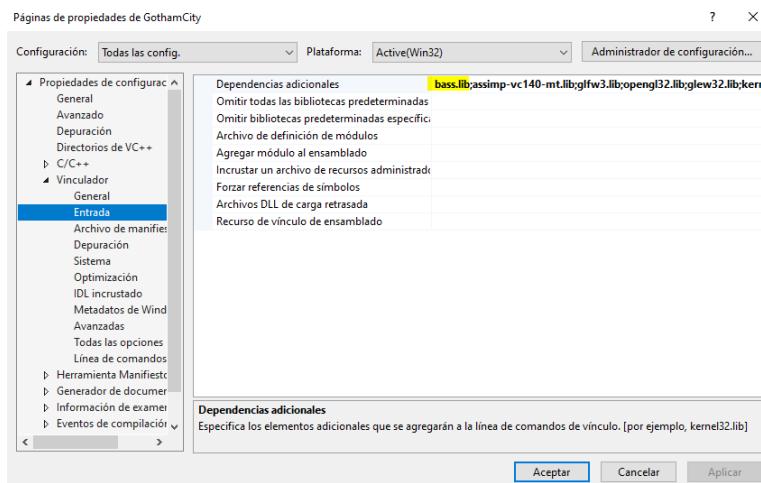


Figura 140: Dependencias adicionales del proyecto

## 6.2. Clase 'Sound'

Para implementar esta clase, se crearon los archivo .h y .cpp correspondientes. Se maneja un constructor que reciba la dirección del archivo de audio, 3 funciones para reproducir, pausar y detener el audio, y 2 variables auxiliares para detectar el dispositivo de audio y el canal por el que se reproduce el audio.



```
Sound.h  X Gotham.cpp
GothamCity
1 #pragma once
2
3 class Sound
4 {
5 public:
6     Sound(const char* filename);
7
8     ~Sound();
9
10    void play();
11
12    void pause();
13
14    void stop();
15
16 private:
17     unsigned int channel;
18     static bool audio_device;
19 };
20
```

Figura 141: Archivo Sound.h

El archivo Sound.cpp manda un mensaje de error en consola si no se encuentra algún dispositivo de reproducción de audio, y utiliza un método de Bass para obtener el canal para el audio correspondiente. Si no se puede obtener, se manda el mensaje de error a consola.

```
#include "Sound.h"
#include <iostream>
#include <bass.h>

bool Sound::audio_device = false;

|Sound::Sound(const char* filename)
{
|  if (!audio_device)
|  {
|    if (!BASS_Init(-1, 44100, 0, NULL, NULL))
|    {
|      printf("Error al cargar el archivo, no hay dispositivo de audio\n");
|    }
|    audio_device = true;
|  }

  channel = BASS_StreamCreateFile(false, filename, 0, 0, BASS_SAMPLE_LOOP);

  if (!channel)
  {
    printf("No se puede reproducir el audio %s\n",filename);
  }
}
```

Figura 142: Archivo Sound.cpp parte 1

Las funciones siguientes solo utilizan algunas de las funciones de la biblioteca de Bass para reproducir, pausar o detener el sonido.

```
|Sound::~Sound()
{
|  BASS_Free();
}

|void Sound::play()
{
|  BASS_ChannelPlay(channel, false);
}

|void Sound::pause()
{
|  BASS_ChannelPause(channel);
}

|void Sound::stop()
{
|  BASS_ChannelStop(channel);
}
```

Figura 143: Archivo Sound.cpp parte 2



### 6.3. Sonido utilizado

Se declararon diferentes instancias de música para este proyecto.

```
Sound music = Sound("Music/Nycteris.mp3");
Sound Tim = Sound("Music/twenty one pilots Stressed Out.mp3");
Sound Jason = Sound("Music/Bohnes - Middle Finger.mp3");
Sound Dick = Sound("Music/Fall Out Boy - Where Did The Party Go.mp3");
Sound Disparo = Sound("Music/Disparo.mp3");
```

Figura 144: Instancias de música

### 6.4. Utilización

La música de fondo se reproduce desde el inicio del programa, sin embargo, las otras pistas de audio son activadas junto a las animaciones del proyecto.

```
// Animacion avatar
if (mainWindow.getTim()) {
    //Musica
    music.pause();
    Tim.play();
    Jason.pause();
    Dick.pause();
    Disparo.pause();
```

Figura 145: Ejemplo de audio 1 en animación

```
else if (brazo1 < -90.0f && cambioJ == 2 && countDisp < 150) {
    pierna1 = 0.0f;
    pierna2 = 0.0f;
    brazo2 = 0.0f;
    Disparo.play();
    countDisp += 1;
}
```

Figura 146: Ejemplo de audio 2 en animación

## 7. Referencias

- [1] Ali deMorg. (2021, 6 septiembre). Base character in 1 MINUTE - Magica Voxel 2021 [Vídeo]. YouTube. [https://www.youtube.com/watch?v=gND2\\_m4Kx3I&list=WL&index=32](https://www.youtube.com/watch?v=gND2_m4Kx3I&list=WL&index=32)
- [2] Art Whit Flo. (2020, 3 julio). You Can Draw This SKYLINE in PROCREATE [Vídeo]. YouTube. <https://www.youtube.com/watch?v=2apW38bDVDg&list=WL&index=37>
- [3] ArtChanny. (2020, 26 diciembre). MagicaVoxel Tutorials: Simple Building Tutorial [Vídeo]. YouTube. <https://www.youtube.com/watch?v=Lc3bj87Oj7I&list=WL&index=13>
- [4] ArtChanny. (2021, 23 mayo). Magicavoxel Tutorials: Part 1: Creating a City (Asset Building) [Vídeo]. YouTube. <https://www.youtube.com/watch?v=j7OgBUaMeaY&list=WL&index=15>



- [5] Bohnes. (2015, 17 diciembre). Bohnes - Middle Finger [Vídeo]. YouTube. <https://www.youtube.com/watch?v=k2z34nkfA9Q>
- [6] C. (1970, 22 agosto). Fotos gratis: estructura, madera, blanco, textura, piso, ciudad, urbano, pared, línea, azulejo, monocromo, material, hormigón, fondo, yeso, Suelo laminado 3836x2153. Pxhere. <https://pxhere.com/es/photo/1173878>
- [7] Casavecchia, P. (2021, 23 mayo). Magicavoxel - Organic looking trees in 20 minutes. [Vídeo]. YouTube. <https://www.youtube.com/watch?v=83pvUjWMlEY&list=WL&index=18>
- [8] Classical and Relax. (2015, 1 mayo). Sonido pistola - sonido disparos [Vídeo]. YouTube. <https://www.youtube.com/watch?v=0JwWEzGuUuY>
- [9] Criscuolo, I. (2021, 8 julio). ¿Qué es el voxel art? Domestika. <https://www.domestika.org/es/blog/5529-que-es-el-voxel-art>
- [10] Developments, U. (s. f.). Bass Audio Library. Un4seen. Recuperado 13 de mayo de 2022, de <https://www.un4seen.com/>
- [11] F. (2021, 7 octubre). Voxel Art para el diseño de personajes. Factor3D. <https://factor3d.com/qubicle/voxel-art-para-el-diseno-de-personajes/>
- [12] Fajarnadril. (2021, 10 diciembre). Tutorial Dasar Magica Voxel dan Upload ke Sketchfab [Vídeo]. YouTube. <https://www.youtube.com/watch?v=COzxKK08pME&list=WL&index=34>
- [13] Fall Out Boy. (2013, 8 abril). Fall Out Boy - Where Did The Party Go [Vídeo]. YouTube. <https://www.youtube.com/watch?v=yvVpfo5dwQI>
- [14] Fueled By Ramen. (2015, 28 abril). twenty one pilots: Stressed Out [OFFICIAL VIDEO] [Vídeo]. YouTube. <https://www.youtube.com/watch?v=pXRviuL6vMY>
- [15] Game Level Arts . (2021, 5 diciembre). Voxel to the future /AE86 - VoxEdit/MagicaVoxel [Vídeo]. YouTube. <https://www.youtube.com/watch?v=0SbIeHT2Eis&list=WL&index=17>
- [16] Graphic Vision. (2021, 18 enero). 3D Isometric Bike model (Speed Art) Scene Bloom Effect & Renderer [Vídeo]. YouTube. <https://www.youtube.com/watch?v=wte8W8PFbP4&list=WL&index=20>
- [17] Illustration Party. (s. f.). a8e60eb06fc31e54f902072fb0ce72c2 [Ilustración]. i.pinimg.com. <https://i.pinimg.com/originals/a8/e6/0e/a8e60eb06fc31e54f902072fb0ce72c2.png>
- [18] Kaikina. (2018, 17 junio). Street - Voxel Art [Vídeo]. YouTube. <https://www.youtube.com/watch?v=xmf5vn2yEXE&list=WL&index=25>



- [19] LoudEyes. (2018, 15 diciembre). Voxel Art Timelapse - 1920s Inspired Street Lamp - MagicaVoxel [Vídeo]. YouTube. <https://www.youtube.com/watch?v=1MAo-YVTB5g&list=WL&index=29>
- [20] Low poly Big Ben. (s. f.). Dribbble. Recuperado 13 de mayo de 2022, de <https://dribbble.com/shots/2883582-Low-poly-Big-Ben>
- [21] Meebit #11748. (s. f.). meebits. Recuperado 13 de mayo de 2022, de <https://meebits.app/meebits/detail?index=11748>
- [22] Meg Wayne. (2019, 14 julio). My top 6 MagicaVoxel tips | 3D city illustration build [Vídeo]. YouTube. <https://www.youtube.com/watch?v=MnHRSRrqK9M&list=WL&index=27>
- [23] Mifik. (2021, 14 noviembre). [MagicaVoxel] Voxel art - CYBERPUNK STREET PART-1 [Vídeo]. YouTube. <https://www.youtube.com/watch?v=EBF0Tcp1bPo&list=WL&index=24>
- [24] Omegafoxx. (2017, 8 octubre). [MagicaVoxel] Building a building #1 [Vídeo]. YouTube. <https://www.youtube.com/watch?v=JvwqlkjBMTA&list=WL&index=31>
- [25] Pixel Real. (2021, 18 octubre). Patrones Hama Beads Halloween Murciélagos [Pixel Real]. <https://pixelreal.net/hama-beads/halloween/murcielagos-2/>
- [26] S. (2020, 5 abril). Learn-Computer-Graphics/external/bass at master sho3la/Learn-Computer-Graphics. GitHub. Recuperado 13 de mayo de 2022, de <https://github.com/sho3la/Learn-Computer-Graphics/tree/master/external/bass>
- [27] Santacruz, W. (2021, 26 junio). Glow City n Voxel Art [Vídeo]. YouTube. <https://www.youtube.com/watch?v=k7PlVxDJXvs&list=WL&index=28>
- [28] Shaalan, M. (2020, 5 abril). how to play sound with opengl c++ ? [Vídeo]. YouTube. [https://www.youtube.com/watch?v=hFwI8xQpM\\_E&list=WL&index=38&t=159s](https://www.youtube.com/watch?v=hFwI8xQpM_E&list=WL&index=38&t=159s)
- [29] Simple Voxel Helicopters Pack | 3D Air. (s. f.). Unity Asset Store. Recuperado 13 de mayo de 2022, de [https://assetstore.unity.com/packages/3d/vehicles/air/simple-voxel-helicopters-pack-109976?aid=1101l3b93&utm\\_campaign=unity\\_affiliate&utm\\_medium=affiliate&utm\\_source=partnerize-linkmaker](https://assetstore.unity.com/packages/3d/vehicles/air/simple-voxel-helicopters-pack-109976?aid=1101l3b93&utm_campaign=unity_affiliate&utm_medium=affiliate&utm_source=partnerize-linkmaker)
- [30] Sozidation Lab. (2021, 6 junio). MagicaVoxel - Toyota Land Cruiser 200 - tutorial for beginners [Vídeo]. YouTube. <https://www.youtube.com/watch?v=TQguJgVtgm0&list=WL&index=19>
- [31] Universo Voxel. (2021, 24 mayo). MAGICAVOXEL - Creating Simple Buildings 3D (Time Lapse) [Vídeo]. YouTube. <https://www.youtube.com/watch?v=MJ8WTSUG-Gk&list=WL&index=30>
- [32] Vo Thai. (2019, 25 octubre). Voxel Art - Street Light Poles - Magicavoxel [Vídeo]. YouTube. <https://www.youtube.com/watch?v=sY4767ESSu4&list=WL&index=21>



- [33] Vo Thai. (2021, 4 enero). Voxel Art - Phonebox - Magicavoxel [Vídeo]. YouTube. <https://www.youtube.com/watch?v=H6oq3TNVerU&list=WL&index=22>
- [34] Zimmer, H. (2019, 14 marzo). Nycteris [Vídeo]. YouTube. <https://www.youtube.com/watch?v=FtzPpelgPZ8>
- [35] Reportes de prácticas de Laboratorio y código proporcionado en clase.