Universidad Nacional Autónoma de México
Faculty of Engineering

Technical Manual - GothamCity

Laboratory of Computer Graphics and Human-Computer Interaction

Deadline: 14/05/2022

Student: Lázaro Martínez Annette Ariadna
Student ID: 316129189

Professor: Ing. Jose Roque Roman Guadarrama
Laboratory group 11
Semester 2022-2

Índice

1. Geometry

1.1. Magica Voxel

Since a specific style of Voxel Art was requested for the project, most of the models were made using Magica Voxel software, which allows you to customize different models with this style and save them in different formats, including .obj

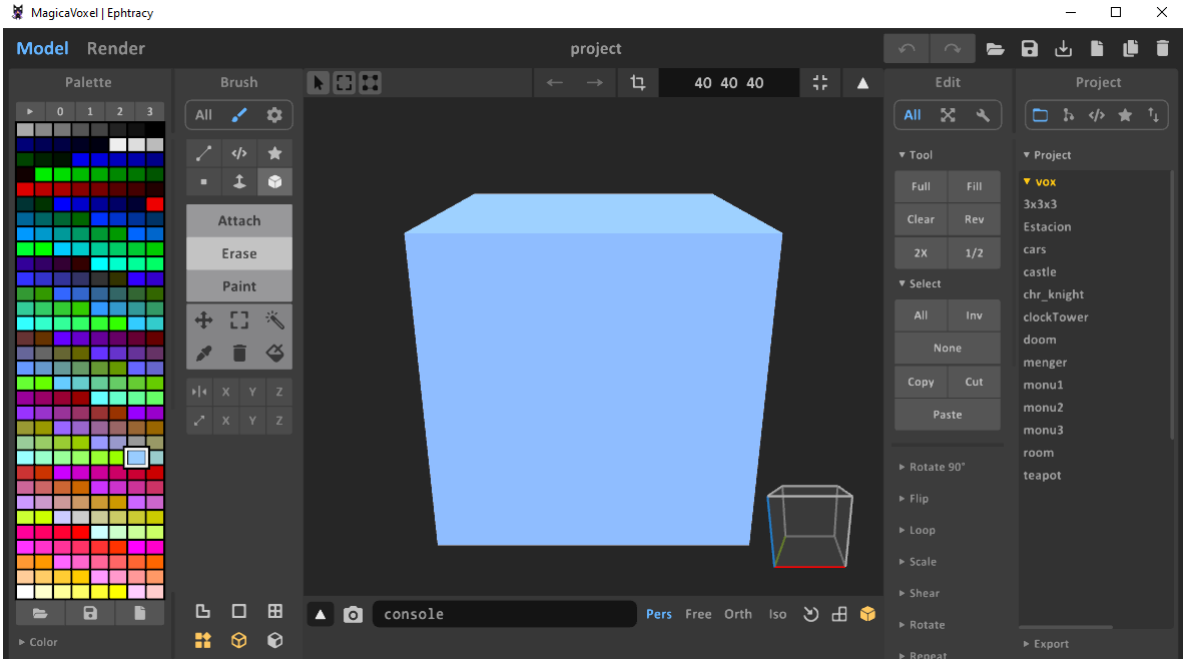


Figura 2: Modeling software

1.2. Models .vox format

I stored all the models in .vox format, which is editable from Magica Voxel, in case it was necessary to modify them in the future. These models are in the repository.

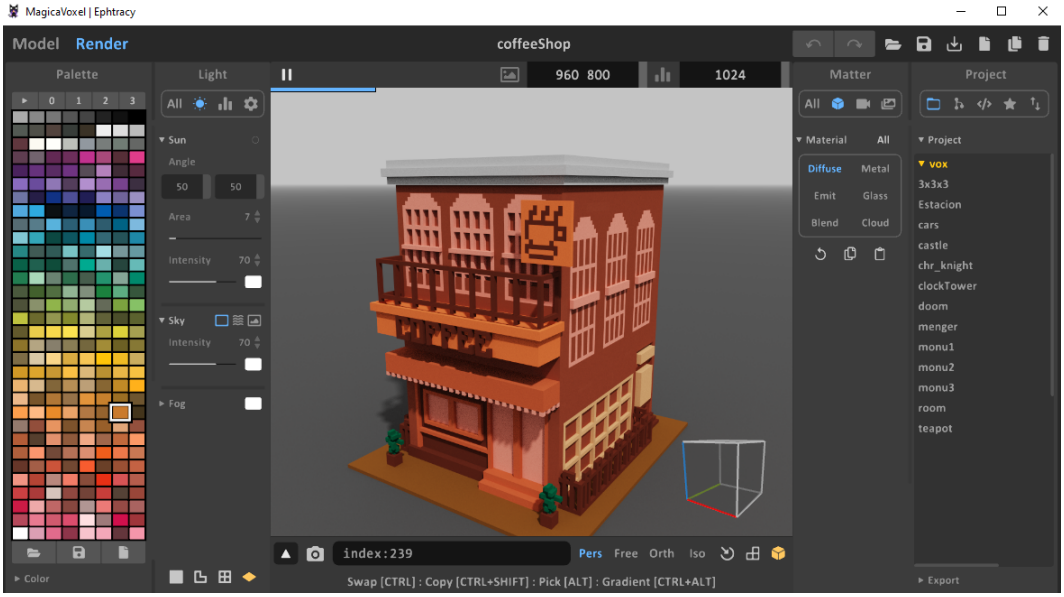


Figura 3: Example model 1 in Magica Voxel

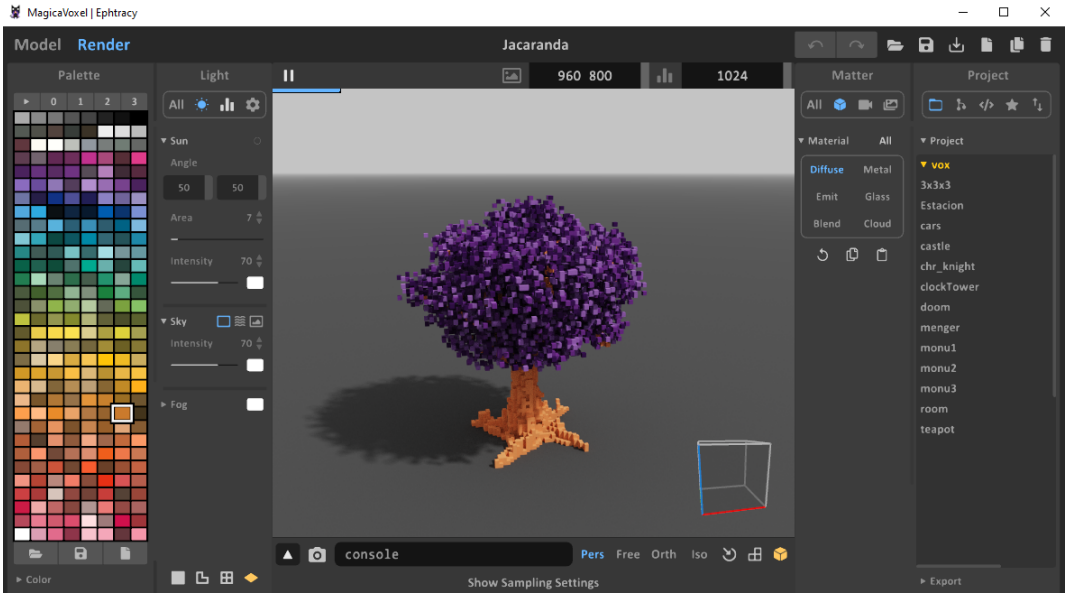


Figura 4: Example model 2 in Magica Voxel

1.3. Models .obj format

In order to import the models to OpenGL and make adjustments with them, it was necessary to export the models to .obj format

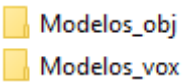


Figura 5: Models folder

1.3.1. One Dimensional Textures

It is worth mentioning that these models, when imported from Magica Voxel, included one-dimensional textures.

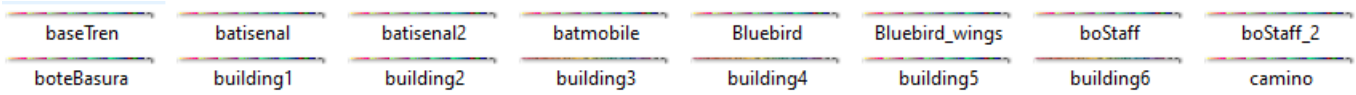


Figura 6: Example of Textures used

1.4. Model optimization

It was necessary to modify some .obj models from 3DS Max, mainly when I had a model whose pivot needed to be modified.

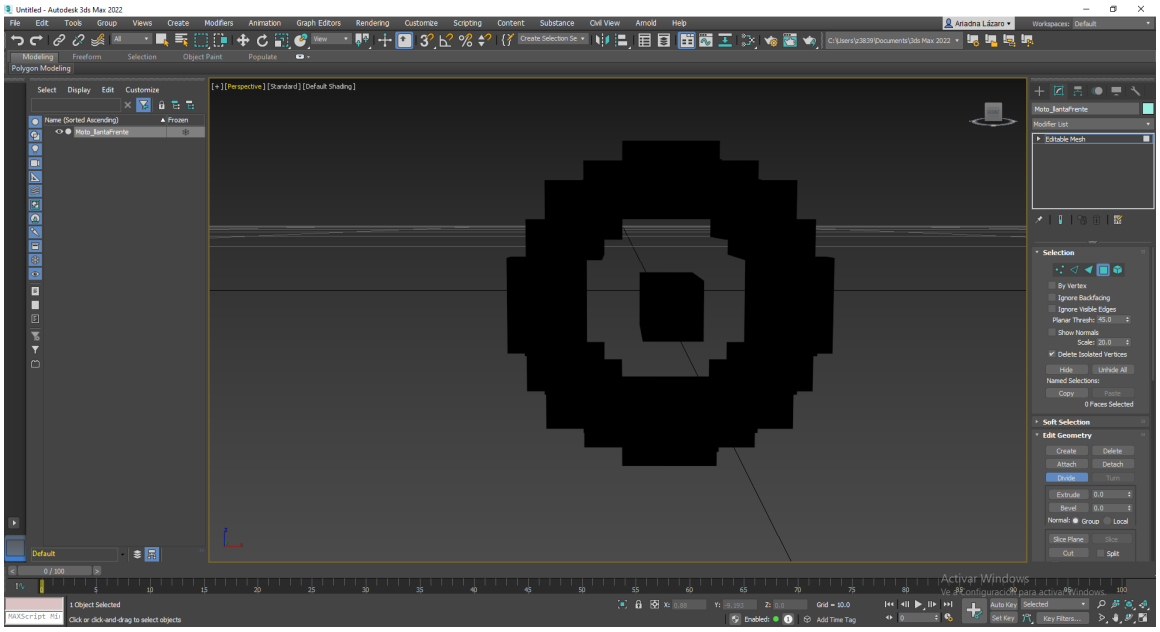
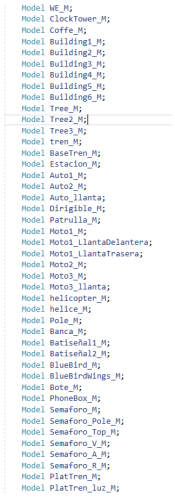


Figura 7: Optimized Model Example

1.5. Model load

Loading the models in OpenGL involved creating an instance of the 'Model' class for each one, and assigning it the corresponding path, making sure that its texture was located in the 'Textures' folder.



```
Model WE_M;
Model ClockTower_M;
Model Coffe_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 Tree3_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 Moto1_LlantaDelantera;
Model Moto1_LlantaTrasera;
Model Moto2_M;
Model Moto3_M;
Model Moto3_llanta;
Model helicopter_M;
Model helice_M;
Model Pole_M;
Model Banca_M;
Model Batiseñal1_M;
Model Batiseñal2_M;
Model BlueBird_M;
Model BlueBirdWings_M;
Model Bote_M;
Model PhoneBox_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: Instances created

```
//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/arbol2.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");
Batiseñal1_M = Model();
Batiseñal1_M.LoadModel("Modelos_obj/Others-street/batisenal.obj");
Batiseñal2_M = Model();
Batiseñal2_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: Assigning file paths

```
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: Assigning file paths

Finally, the necessary transformations were made with each of the models to position them correctly within the scenario.

```
//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: Model load example

1.6. Model Hierarchy

Due to the nature of the models, it was necessary to carry out the hierarchy by code of some objects, mainly to effect a translation such as cars or motorcycles, however, the hierarchy was also used with humanoids, with the bird, and even with the models that surround each of the buildings, because this way, by taking the buildings as the center of the hierarchy, it was easy to move the other objects to accommodate all the buildings within the stage.

```
//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: Hierarchy of humanoids

```
//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: Hierarchy of the bird

```
//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: Hierarchy of transports

```
//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: Hierarchy of buildings, part 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: Hierarchy of buildings, part 2

1.7. Results

For a better organization of the models, I created subfolders that would allow me to save the different models created.

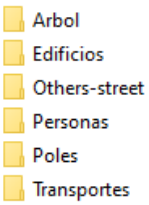


Figura 17: Model Categories

Below is an example of the different objects instantiated within the final project stage.

1.7.1. Trees

Green leaf trees and Jacarandas are mainly stored in this folder.

Figura 18: Trees

1.7.2. Buildings

In this folder, the 9 different buildings used in the project were stored, as well as some objects of said buildings that change according to the day and night cycle.

Figura 19: Buildings 1

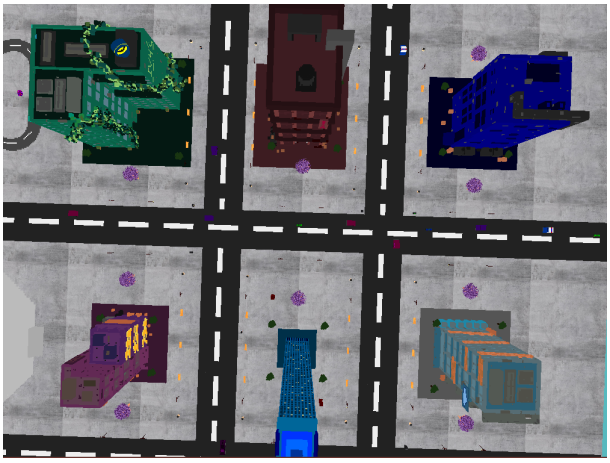


Figura 20: Buildings 2

1.7.3. People

The main characters and their weapons were stored in this folder.

Figura 21: People 1

Figura 22: People 2

1.7.4. Transports

In this folder the cars, motorcycles, the helicopter, the airship, the train, and the Batimobile were stored.

Figura 23: Transports

1.7.5. Poles

Mainly the light poles and the main character's weapon are stored in this folder.

Figura 24: Poles

1.7.6. Others-street

In this folder, the models corresponding to benches, garbage cans, telephone booths, traffic lights, etc. were stored.

Figura 25: Others-street

2. Avatar

2.1. Primitives: cubes

Due to the drawing style, the type of primitives I used for the avatar were cubes, however, since each part of the avatar should have different texture coordinates, I had to make a different cube for each member of its body.

It is worth mentioning that unitary cubes were used, and later transformations were used to modify the size of the objects, in addition, taking the Magica Voxel coordinates as a reference, the scale of the cubes and their subsequent positioning was relatively simple.

```
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,   0.99902f,       0.0f,   0.0f,   -1.0f,  //2
        -0.5f,  0.5f,  0.5f,    0.09375f,   0.99902f,       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,
```

Figura 26: Creating the cubes, part 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,

        //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 27: Creating the cubes, part 2

```
unsigned int cuerpo_indices[] = { ... }
GLfloat cuerpo_vertices[] = { ... }
unsigned int brazoIzq_indices[] = { ... }
GLfloat brazoIzq_vertices[] = { ... }
unsigned int brazoDer_indices[] = { ... }
GLfloat brazoDer_vertices[] = { ... }
unsigned int piernaIzq_indices[] = { ... }
GLfloat piernaIzq_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: Creation of all parts of the body

```
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: Sending each cube to the mesh

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

Figura 30: Calling the functions that create the cubes from the main function

2.2. Texture

2.2.1. Model at Magica Voxel

For the creation of the texture, it was necessary to make a model in Magica Voxel first.

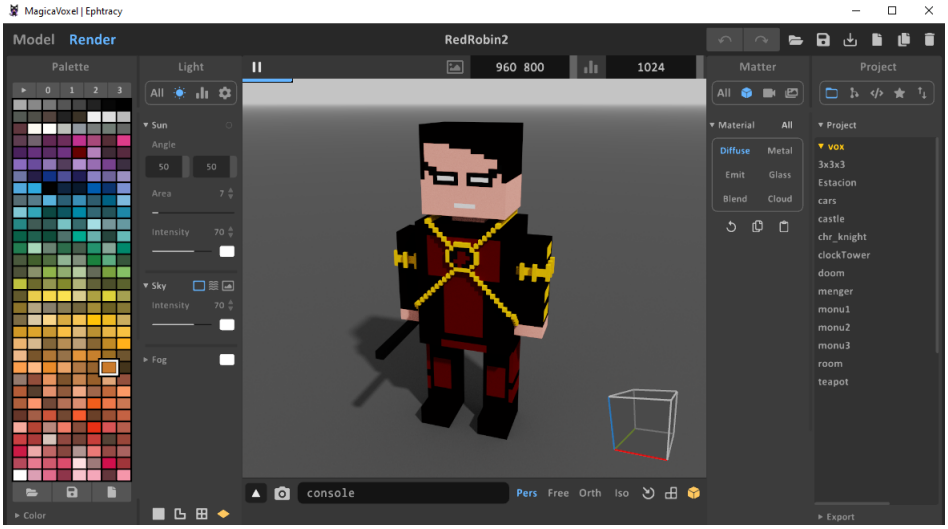


Figura 31: Reference model in Magica Voxel

2.2.2. Creation of the texture

Then, the necessary captures were taken, the corresponding color and size adjustment was made, the background was removed, it was exported in .tga format and the character's texture was obtained from GIMP.

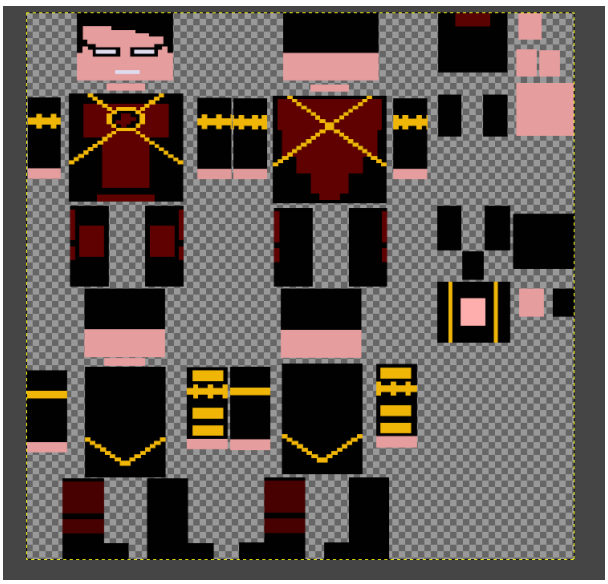


Figura 32: Avatar texture

2.2.3. Texture positioning

To apply the texture to each of the cubes, a rule of 3 had to be obtained between the dimensions of the image texture (1024x1024), and the coordinate system S x T that has a range between 0 and 1, such as it was done in the practice corresponding to textures.

```
GLfloat cuerpo_vertices[] = {
    // front
    //x     y       z       S       T           NX      NY      NZ
    -0.5f, -0.5f,  0.5f,    0.07812f,   0.65529,        0.0f,   0.0f,   -1.0f,  //0
    0.5f, -0.5f,  0.5f,     0.28613f,   0.65429f,       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
    0.5f, -0.5f,  0.5f,     0.10742f,   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,     0.10742f,   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
    -0.5f, -0.5f,  -0.5f,   0.10742f,   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,   0.10742f,   0.35056f,       1.0f,   0.0f,   0.0f,

    // bottom
    //x     y       z       S       T
    -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: Texture positions for each face of a cube

2.2.4. Hierarchy

Finally, the mesh objects were called and the auxiliary matrices were used to perform the hierarchical modeling of the body members, taking the character's torso as the center.

```
//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();

//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: Avatar Hierarchy, part 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: Avatar Hierarchy, part 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));
TimTexture.UseTexture();
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
meshList[7]->RenderMesh();

//pie izq
model = modelaux2;
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));
TimTexture.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));
TimTexture.UseTexture();
MaterialParaLuces.UseMaterial(uniformSpecularIntensity, uniformShininess);
meshList[8]->RenderMesh();
```

Figura 36: Avatar Hierarchy, part 3

2.3. Additional objects

Additional models for the avatar were also used, such as the climbing hook and his weapon of choice, a retractable bo staff.

```
//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: Weapons

2.4. Material to interact with lights

Two different types of materials had to be used, one that was brighter to interact with the lights and one that was less bright.

```
Material MaterialParaLuces;
Material MaterialNormal;
```

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

Figura 38: Materials

Afterward, both the corresponding textures and the type of material for each of the avatar's body parts were called up.

```
//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. Animation

First, the files Windows.h and Windows.cpp were modified to have a boolean variable called 'Tim' that modifies its value depending on the state of 2 keys, as well as the necessary functions to obtain its value and return its value to false if necessary.

Variables and functions are declared as shown below:

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

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

Figura 40: Changes on Windows.h

And the events were added to change its value with the keys '7' and '8'.

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

Figura 41: Changes on Windows.cpp

To start the animation, the value of the variable must be true, which means that it only starts if the user presses the '7' key. The change from the background music to the soundtrack of the avatar starts instantly.

To have control over the order of the animation, a variable called 'cambio' was used, which controls the state of the animation, as well as auxiliary variables for the translation, rotation and scale of the different parts of the avatar.

At the beginning of the animation, the corresponding arm that is holding the bo staff is raised, later, the second change involves the lengthening of the bo staff, since it is a retractable weapon. Then the rotation of the weapon is performed for 2 turns. At the end of the movement, the bo staff is retracted, and the arm is lowered.

```
// 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: Avatar animation, part 1

In state number 6, a continuous movement of the arms and legs is made while a translation is made, in such a way that the route of a straight line journey is simulated. Upon reaching its destination, it returns to the initial value of the members.

```
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: Avatar animation, part 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: Avatar animation, part 3

Change 7 implies the movement of the arm that holds the hook and change 8 is the return of said arm, in such a way that the launch of the object is simulated. Change 9 performs the launch and rotation of the hook, so that it remains supported on the edge of the building. Change 10 involves the character moving due to the hook and 11 is when the avatar retrieves their tool.

```
else if (cambio == 7 && brazo6 >= -210.0f) {
    brazo6 -= 2.0f;
}
else if (cambio == 7 && brazo6 < -210.0f) {
    cambio = 8;
}
else if (cambio == 8 && brazo6 <= -180.0f) {
    brazo6 += 5.0f;
}
else if (cambio == 8 && brazo6 > -180.0f) {
    cambio = 9;
}
else if (cambio == 9 && ganchoTras <= 42.0f) {
    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) {
    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) {
    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: Avatar animation, part 4

Change 12 is the turn of the character and 13 is the return to the ground. As of change 14, the movement of the limbs and the translation of the body are carried out to return to the starting point.

```
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) {
    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) {
    //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: Avatar animation, part 5

At the end of the last movement, the variables are reset to return to the initial state, and the 'setTim' function is called to reset the boolean variable with a false value, so the user must press the '7' key to see the animation again.

```
            //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: Avatar animation, part 6

Finally, the values of the modified variables in the animation are taken to change the transformations of the 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: Avatar translation change

```
//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: Shoulder rotation

2.6. Results

The character on stage is shown below, but the animation is better seen in the video.

Figura 50: Avatar: Timothy Jackson Drake

3. Routes

The movement around the stage is defined according to the camera used.

As the stage was outdoors, two different cameras were used, one linked to the floor and the other aerial.

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

Figura 51: Instances for the cameras

According to the Camera class, the following initial values were configured for each of the cameras:

- Camera position
- Upward orientation.
- Skew on 'x' axis.
- Skew on 'y' axis'.
- Speed of movement.
- Turning speed.

```
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: Initial setup for cameras

3.1. Aerial camera

For the aerial camera, an initial position was declared at the center of coordinates, but at a height where all other objects on the stage can be observed. Its initial orientation is towards the positive 'y' axis, however, the added skew on the 'y' axis of -90 units gives the feeling of looking down.

It is worth mentioning that the speed of this camera is faster than the floor camera, with the intention of being able to follow some elements of the scene, such as the airship.

3.2. Camera bound to the floor in third person

The camera linked to the floor has an initial position in the center of coordinates, its orientation is with the positive 'y' axis, and it only handles a skew in the 'x' axis, since if a skew is added in the 'y' axis, it is likely that the user will end up seeing something below the floor during their tour.

3.3. Change of cameras

To change cameras, a Boolean variable was used that changes value according to the '1' and '2' keys.

```
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 = false; }
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 createCallbacks();
GLfloat lastX;
GLfloat lastY;
GLfloat xChange;
GLfloat yChange;
GLfloat muevex;
GLboolean CamAerea, spotlights, animacionBatmobile;
```

Figura 53: Changes on Window.h

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

Figura 54: Changes onWindow.cpp

In addition, the functions that calculate the position of the cameras, their view and the control of both the mouse and the keyboard are called according to the value of the auxiliary variable, so that the floor camera is obtained with the '1' key and the aerial with '2'.

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

Figura 55: Camera control

```
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: Camera position

3.4. Movement control

We must be sent to the 'Key Control' method the information of which camera is being used in order to select the axis where the movement is made with keys.

Due to the 'update' and 'calculateViewMatrix' functions of the camera class, the coordinate axes are automatically obtained according to the current position and orientation of the camera, so the values of the variables 'right', 'in front' and 'up' are updated without issue, however the ground camera should be able to move 'forward' and 'right', while the aerial camera should be able to move 'up' and 'right'.

```
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: Automatic update

Due to the above, the value of the boolean variable is used to decide the movement of the cameras.

- W-Key: Moves the camera forward on the ground camera and up on the aerial camera.
- A-Key: Moves the camera to the left (negative 'right') on either camera.
- S-Key: Moves the camera backwards (negative 'front') on the ground camera and down (negative 'up') on the aerial camera.
- D-Key: Moves the camera to the right on any of the cameras.

```
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: Movement with keyboard according to the camera used

In addition, it can be mentioned that the fact of using two instances for the camera allows the correct storage of the previous position in the cameras.

3.5. Results

Figura 59: Floor camera

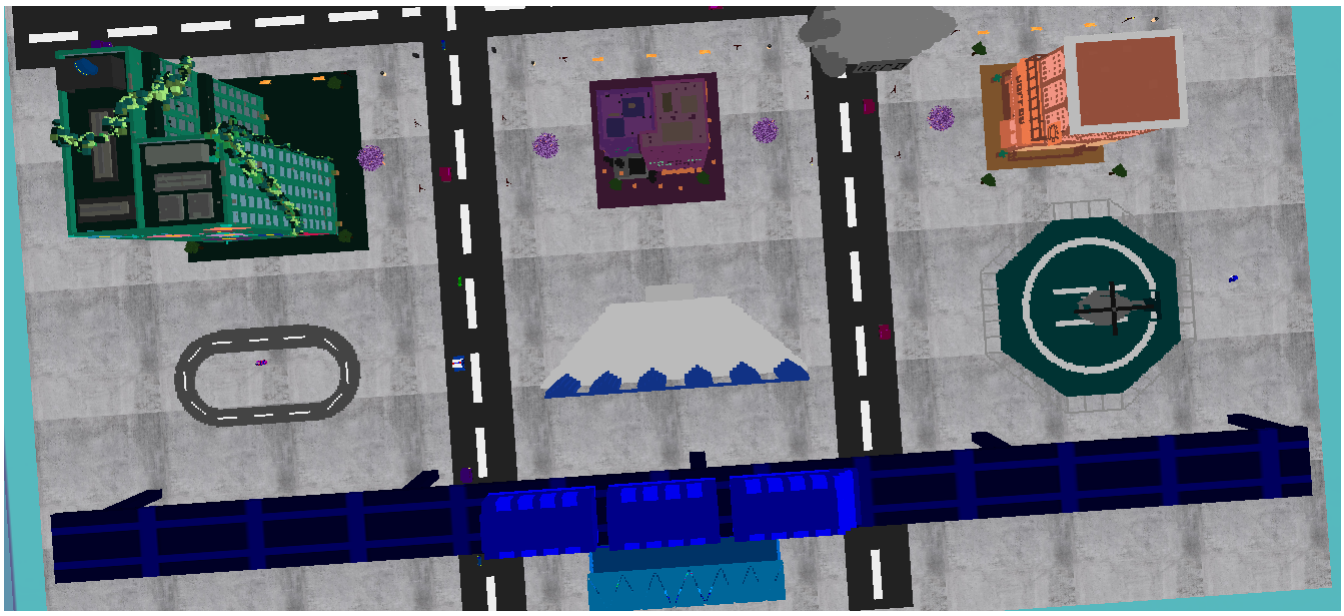


Figura 60: Aerial Camera

4. Illumination

4.1. Day-night cycle

To make the change, some auxiliary variables were used, a counter 'count' and two limits, a variable that establishes the limit of the duration of the day (2000 units) and another one of the complete day-night cycle (4000 units). Cycle-based changes are made with the variables CicloDia and CicloDiaNoche, while the variable count increments its value by one while the program is open and restarts when reaching the value of CicloDiaNoche.

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

Figura 61: Auxiliary variabless

4.1.1. Skybox Change

For the Skybox change it was necessary to have the two corresponding images, store the 6 faces of each of them and generate two instances of the Skybox class with their corresponding faces.

```
//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 in code

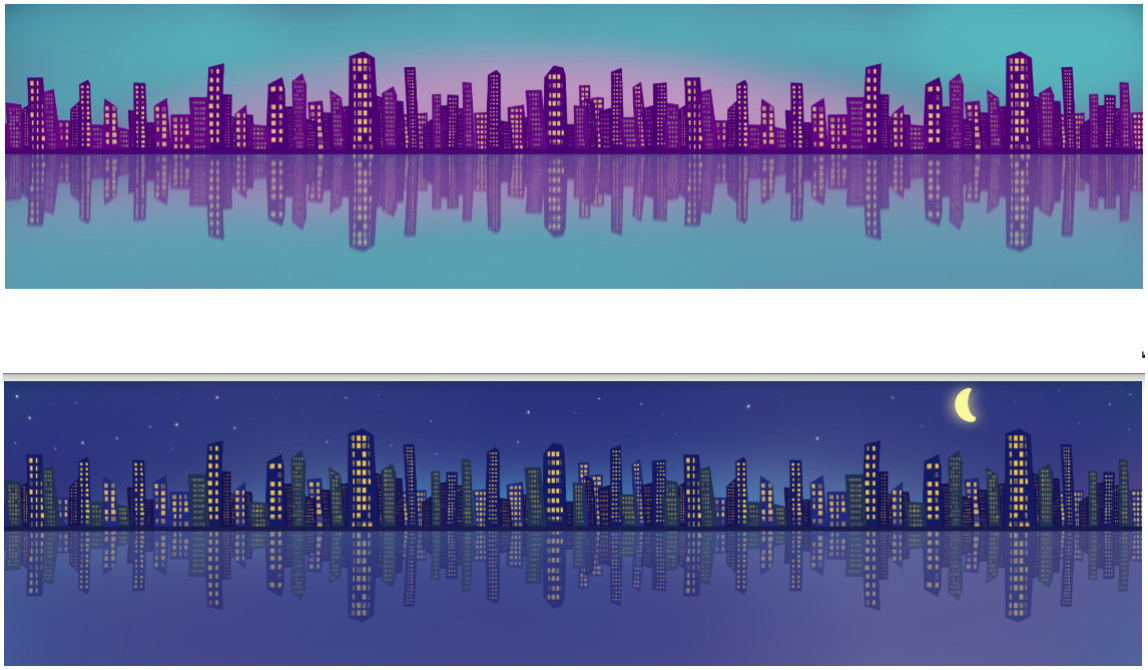


Figura 63: Skyboxes in image

The change of Skybox is done within the While loop of the program, while the main window is still open, however, not only are there 2 cases, camera changes have to be considered as well, so the 'DrawSkybox' function has 4 possible combinations according to the current camera and the value of our counter to know if it is day or night.

```
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: Skybox Change

4.1.2. Cycle Based Lighting

Some lights, such as the one linked to the bat-signal and the airship, are activated only at night, so it is necessary to know the value of the 'count' variable to declare them or not.

```
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: Night cycle lights

```
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: Day cycle lights

4.1.3. Change of textures based on the cycle

Since Magica Voxel allows some elements to be rendered, lights can be added to models that way, however these models cannot be imported.

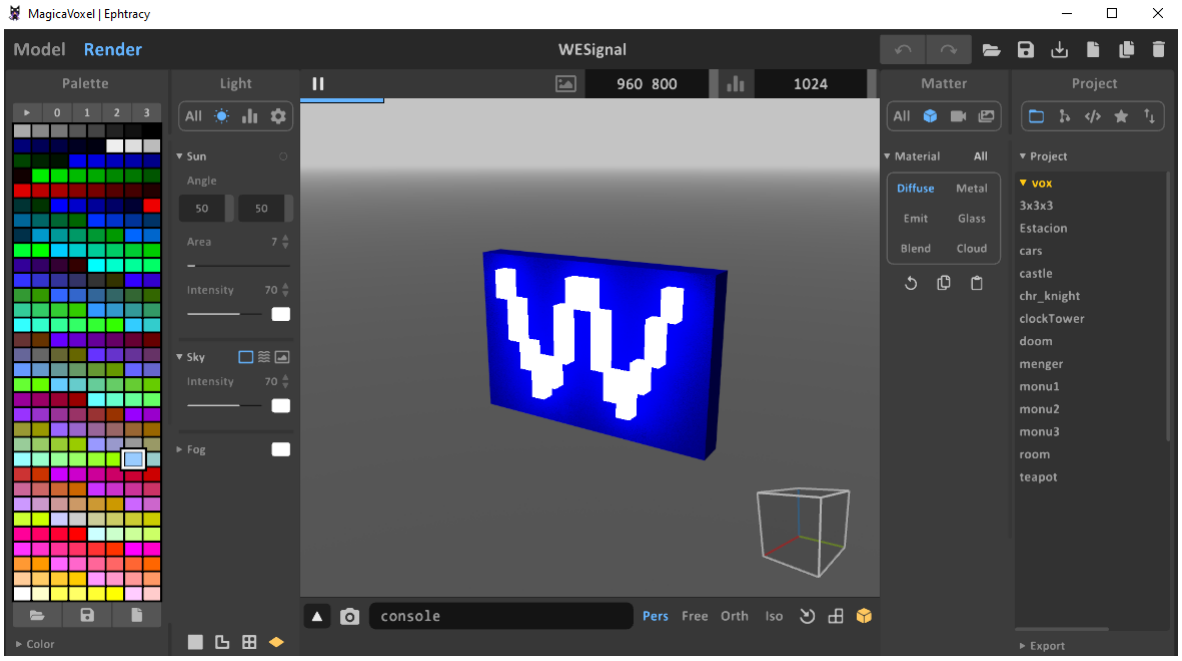


Figura 67: Rendering in Magica Voxel

In the solution to obtain more lighting elements and comply with the section on 'building elements that light up according to the day-night cycle' included in the project proposal, I took screenshots of some objects in Magica Voxel, with which I made textures of some objects with and without light.

Figura 68: Textures created

Later, using Blender, I assigned the corresponding textures as we did in class, only I applied it to two objects, one with light and one without light.

Finally, I modified in the code the model that was shown when it was day or night in the program, achieving the object of additional luminaires of some signals on the stage, the base of the train, the traffic lights, among others.

```
//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: Change of model according to the day-night cycle

4.2. Keyboard controlled illumination

Like other objects in this project, and as previously mentioned in other sections, a boolean auxiliary variable was added to turn on and off the light linked to the helicopter with the '9' and '0' keys, making the corresponding changes in the Window class.

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

}
```

Figura 70: Shift keys in Window.cpp

With the help of this variable, you can change the declarations of the lights. For example, in the day cycle, the spotlight of the helicopter is the only one that is declared if the variable is true, while no light of this type is declared with the negative value of the variable (in the day cycle).

```
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: Helicopter light during the day

On the other hand, in the night cycle, the lights corresponding to the airship and the bat signal must always be declared, only the declaration of the helicopter light is modified depending on whether the value of its variable is true or false, so they may have 2 or 3 spotlights at night.

```
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 72: Declaration of helicopter light during the day (helicopter: true)

```
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: Declaration of helicopter light during the day (helicopter: false)

4.3. Light show

For the light show, the lights must also be activated according to the keys '3' and '4', so the corresponding changes were made in Window.h and Window.cpp

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

Figura 74: Light Show Initialization Keys

Spotlights are declared in the main method only if the value of the auxiliary variable that is activated by the '3' key is true.

```
//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: Initialization of lights for the show

For the animation of this cycle, different functions and auxiliary variables are used, as well as an auxiliary variable (luces) that controls the order of the animation.

Shifts 1 and 2 are independent sinusoidal movements of the end lights, shifts 3, 4 and 5 are the joint sinusoidal movements of the end lights, while movement 6 and 7 is the spiral movement of the center light.

```
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: Animation of the show, part 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: Animation of the show, part 2

Finally, the 'setPos' method of the spotlights is called to modify its position according to the

auxiliary variables of the animation.

```
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: Animation of the show, part 3

4.4. Results

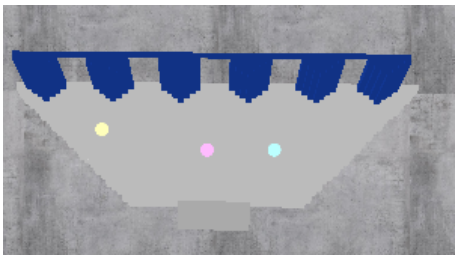


Figura 79: Aerial view of the light show

Figura 80: Aerial view of the batsignal

5. Animation

Auxiliary variables were used for all animations.

```
//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, zluz1 = 0.0f, xluz2 = 0.0f, zluz2 = 0.0f, xluz3 = 0.0f, zluz3 = 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 for animation

5.1. Basic

5.1.1. Batmobile: keyboard controlled movement

This model was instantiated with the wheels forming a hierarchy of models, so that they could move alongside the batmobile, but could also roll independently.

As in other objects, a boolean helper variable was used so that the movement starts with the '5' key and stops with the '6' key.

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

}
```

Figura 82: Keys for animation

For the animation, an auxiliary variable was also used to control the movement, in this case it was added only displacement in the 'x' and 'z' axes, as well as rotations on the axis of the

batmobile and the rotations of the tires.

```
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: Animation of the batmobile

The limits of the variables used for turns were also declared, so that they do not reach very high values during the execution of the program.

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

Figura 84: Limits for variables

Finally, the variables that perform the increments and decrements were used to modify the transformations of the model, as shown below.

```
//batmobile sin movimiento
model = glm::mat4(1.0);
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: Translation and rotation transformations.

5.1.2. Batsignal: Based on the day-night cycle

This model changes at night to appear to have 'turned on' the signal, plus it rotates and scales at night.

```
//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: Batsignal animation

5.1.3. Street transports

The control of these movements was somewhat complicated, in addition to the fact that variables were needed for each of the transports and an extra variable for the movement of their tires

was required, it was also added the positioning with a hierarchy of models.

```
//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 for animation

This animation is constant, so the condition to stop or continue the movements is the value used in the traffic lights, coordinating them so that the transports advance when their lane is green or yellow, but stop when they are red. According to the control, the first movement is the one made by the vehicles in the lanes to the left and right of the avatar, so the values are constantly increased for all models.

```
//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 (-607.0f + movAuto12 == -22.0f) {
    movAuto12 = 0.0f;
    //printf("\nSe reinicio mov12");
}
movAuto12 += 0.5f;
rotLlanta12 += 5.0f;
```

```
if (-645.0f + 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: Increases for the left lane

It is worth mentioning that the conditions that are added are to restart the value of the variables that carry out the translations when they arrive again at their initial position, this was implemented to avoid their overflow due to the fact that the movement of all the vehicles is constant throughout the entire Program.

```
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: Increases for the right lane

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

Figura 90: Change control for the next move

Then it continues the movement of the lanes in front of and behind the 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: Increases for the back lane

```
//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: Increases for the front lane

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

Figura 93: Change control to return to the first movement

And the limits for the variables of the rotations are established.

```
//
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: Limits

Finally, the translations of the objects are modified according to the variables that carry out the increments. It is worth mentioning that two different cases were taken into account for the translation, one of them is the first movement of the vehicle until it reaches the end of the stage, and the second one is to start from the other end of the stage, to have a constant movement.

Due to the above, it was necessary to perform different operations and assign different values to each vehicle according to its initial position.

```
//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: Car model in the right lane

```
//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: Bike model in the left lane

```
//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: Patrol model in the back lane

```
//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: Car model in the front lane

5.2. Complex

5.2.1. Red Hood

This animation is triggered when the floor camera approaches to a certain distance from the model. The movements he performs is a parabolic translation in the XZ plane, movement of all his limbs, he raises his arm and activates the sound of the shot to simulate the firing of one of his weapons, then he turns around and returns to his original position to repeat the animation.

```
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: Red Hood animation, part 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: Red Hood animation, part 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: Red Hood animation, part 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: Red Hood animation, part 4

5.2.2. Nightwing

This character walks in the XZ plane, turns 90 degrees, jumps in a parabolic shape, turns around to return to the start of where the jump started, turns 90 degrees again, and returns to its original position to restart the animation.

```
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: Nightwing animation, part 1

```
        //Traslacion
        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: Nightwing animation, part 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: Nightwing animation, part 3

```
        //Traslacion
        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: Nightwing animation, part 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: Nightwing animation, part 5

```
//Nigthwing
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: Nightwing animation, part 6

5.2.3. Blue bird

This model performs a spiral movement in the XZ plane while performing a linear movement in the Y axis, as well as rotating its wings and body. Upon reaching a point in the spiral movement, it returns with the same movement.

It is worth mentioning that an auxiliary variable was used as a counter to start the movement after a certain time after the program was started, and not from the beginning like other models.

```
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: Bird animation, part 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: Bird animation, part 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();

//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 111: Bird animation, part 3

5.3. Animation technique

5.3.1. KeyFrames

For this animation technique, the variables, structures, functions and other necessary variables were declared, both for the helicopter and for the airship. Something important that I must mention is that each variable requires an auxiliary variable to perform the calculation of the increments in the interpolation.

```
//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; //Numero 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: Declarations for airship animation

The function to save the frames shows the corresponding message in the console and stores the values in the indicated format, so that it is easier to read the text file.

```
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 = "";
                        xdokay = 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 = "";
                        ydokay = 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: Function to store values in the text file

Then, the function is declared to read these stored values according to the position of the values. Something important to note is that, by having a frame greater than 9, the positions within the text file are modified, which is why two reading cases were taken, in addition to the fact that there is not a case with 3-digit frames (eg 100), due to the amount of resources that would be used.

```
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 114: Function to read values from text file

There is a helper function to return the values to 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: Function to reset variables

And the function that performs the calculation of the interpolation with 2 frames and the total number of steps that was declared at the beginning is added.

```
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: Interpolation function

The function that performs the animation control is as follows.

```
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: Function for animation

This function is called from the main method, during the while loop:

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

Figura 118: Calling the function for animation

The same functions are declared with the ending '_Helicoptero', adapting them to the helicopter model for its movements. The biggest difference is the keys that make the changes.

5.3.2. Airship

In the main method the initial Frames of the model are also declared, in this case only two are used.

```
//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: Airship starting frames

Below are the different keys that were assigned to perform the airship's movements.

```
//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: Keys to play animation

```
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: Keys to save the values in the text file

```
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: Keys for movement in X axis

It is worth mentioning that the movement that controls the variable 'movd_y' is actually about the Z axis, only I never changed the name.

```
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: Keys for movement in Z axis

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

Figura 124: Turn keys

```
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: Keys to read the values from the text file

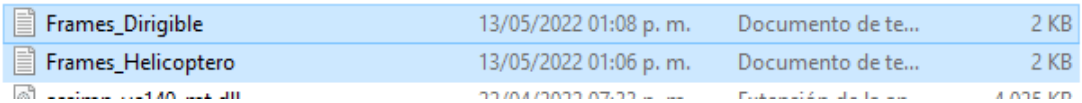


Figura 126: Files generated by the program

5.3.3. Helicopter

```
//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: Initial Frames

```
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;
        }
    }
```

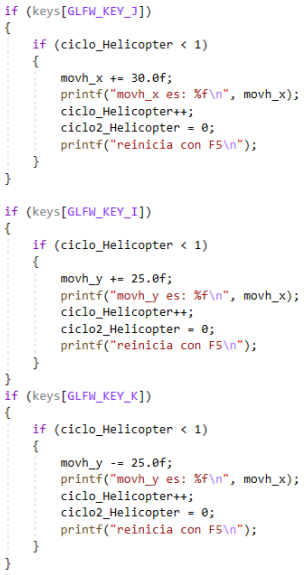
Figura 128: Keys to play animation

```
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: Keys to save the values in the text file and positive movement in X

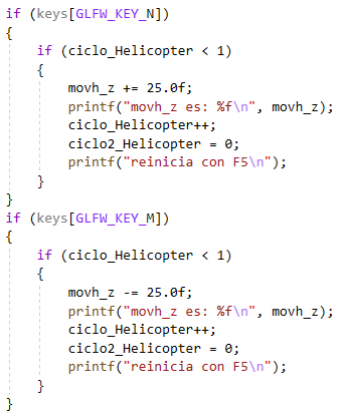
```
if (keys[GLFW_KEY_LEFT])
{
    if (ciclo_Dirigible < 1)
    {
        movd_x -= 5.0f;
        printf("movd_x es: %f\n", movd_x);
        ciclo_Dirigible++;
        ciclo2_Dirigible = 0;
        printf("reinicia con F1\n");
    }
}

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: Keys for movement in X axis



```
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_x);
        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_x);
        ciclo_Helicopter++;
        ciclo2_Helicopter = 0;
        printf("reinicia con F5\n");
    }
}
```

Figura 131: Keys for movement in Y axis and negative movement in 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: Keys for movement in Z axis

```
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: Keys for helicopter rotation and propeller rotation

```
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: Keys to read the values from the text file

```
//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));


//Helicopter
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);
helicopter_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: Modifying model transformations in code

All the results of this section are better appreciated from the video.

6. Audio

6.1. Implemented library

The audio library used for this project was 'Bass', implemented by following a tutorial.

The files for this library can be obtained from the official page or from a GitHub repository from the person who uploaded the implementation tutorial.

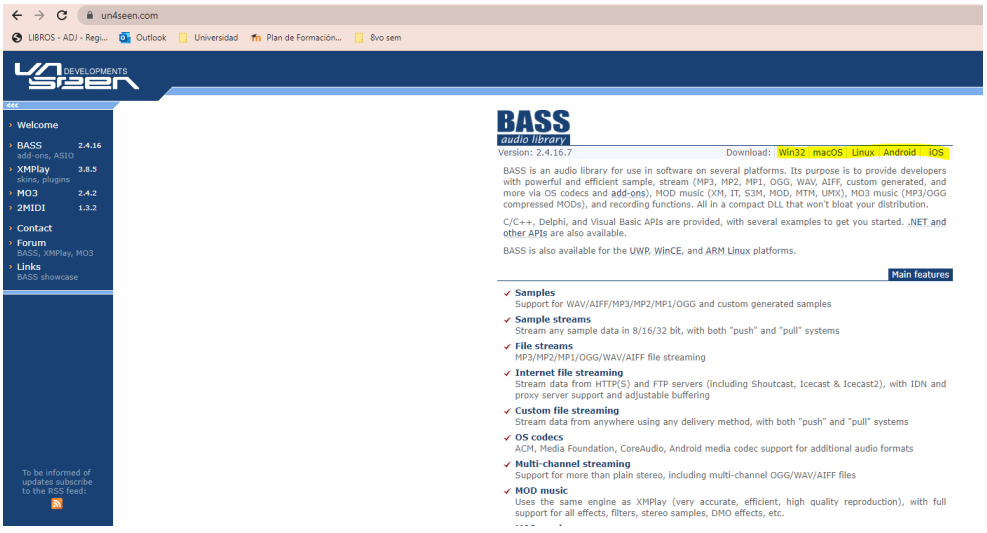


Figura 136: Download from the official page

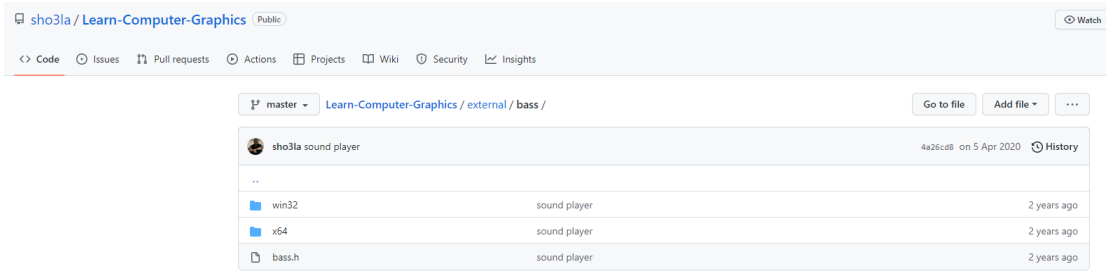


Figura 137: Download from Github

6.1.1. Project Modifications

To implement the library it was necessary to include the corresponding files in the 'include' and 'lib' folders.

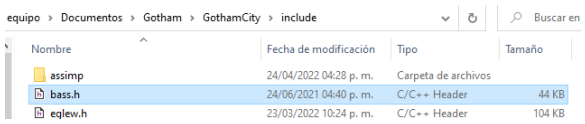


Figura 138: File in 'include'

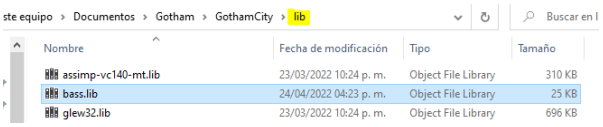


Figura 139: File in 'lib'

Also, it was added the library to the project solution settings.

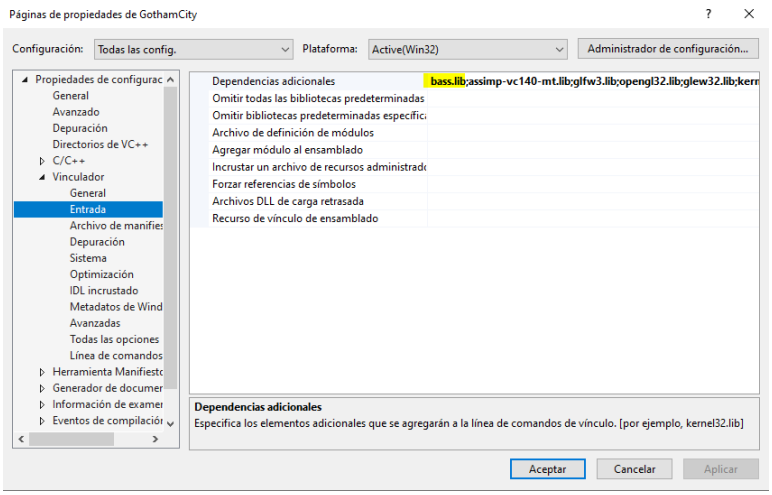


Figura 140: Additional project dependencies

6.2. 'Sound' class

To implement this class, the corresponding .h and .cpp files were created. There are handled a constructor that receives the address of the audio file, 3 functions to play, pause and stop the audio, and 2 auxiliary variables to detect the audio device and the channel through which the audio is played.

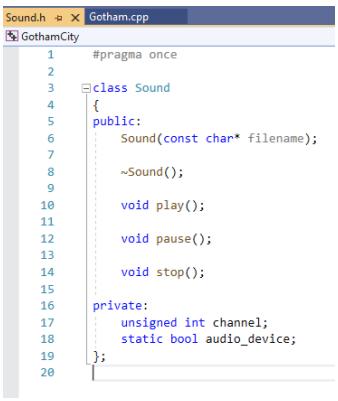


Figura 141: Sound.h file

The Sound.cpp file sends an error message to the console if no audio playback device is found, and uses a Bass method to get the channel for the corresponding audio. If it cannot be obtained, the error message is sent to the console.

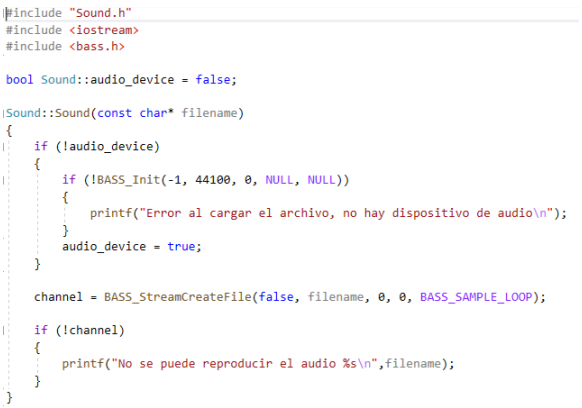


Figura 142: Sound.cpp file, part 1

The following functions only use some of the functions in the Bass library to play, pause, or stop the sound.

```
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: Sound.cpp file, part 2

6.3. Used sound

Different instances of music were declared for this project.

```
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: Instances of music

6.4. Utilization

The background music is played from the beginning of the program, however, the other audio tracks are activated together with the animations of the project.

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

Figura 145: Audio example 1 in animation

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

Figura 146: Audio example 2 in animation

7. References

[1] Ali deMorg. (2021, September 6). Base character in 1 MINUTE - Magica Voxel 2021 [Vídeo]. YouTube. https://www.youtube.com/watch?v=gND2_m4Kx3I&list=WL&index=32

[2] Art Whit Flo. (2020, July 3). You Can Draw This SKYLINE in PROCREATE [Vídeo]. YouTube. https://www.youtube.com/watch?v=2apW38bDVDg&list=WL&index=37

[3] ArtChanny. (2020, December 26). MagicaVoxel Tutorials: Simple Building Tutorial [Vídeo]. YouTube. https://www.youtube.com/watch?v=Lc3bj87Oj7I&list=WL&index=13

[4] ArtChanny. (2021, May 23). 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, December 17). Bohnes - Middle Finger [Vídeo]. YouTube. https://www.youtube.com/watch?v=k2z34nkfA9Q

[6] C. (1970, August 22). 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, May 23). 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, May 1). Sonido pistola - sonido disparos [Vídeo]. YouTube. https://www.youtube.com/watch?v=0JwWEzGuUuY

[9] Criscuolo, I. (2021, July 8). ¿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. Retrieved May 13, 2022, from https://www.un4seen.com/

[11] F. (2021, October 7). Voxel Art para el diseño de personajes. Factor3D. https://factor3d.com/qubicle/voxel-art-para-el-diseno-de-personajes/

[12] Fajarnadril. (2021, December 10). 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, April 8). Fall Out Boy - Where Did The Party Go [Vídeo]. YouTube. https://www.youtube.com/watch?v=yvVpfo5dwQI

[14] Fueled By Ramen. (2015, April 28). twenty-one pilots: Stressed Out [OFFICIAL VIDEO] [Vídeo]. YouTube. https://www.youtube.com/watch?v=pXRviuL6vMY

[15] Game Level Arts . (2021, December 5). Voxel to the future /AE86 - VoxEdit/MagicaVoxel [Vídeo]. YouTube. https://www.youtube.com/watch?v=0SbIeHT2Eis&list=WL&index=17

[16] Graphic Vision. (2021, January 18). 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, June 17). Street - Voxel Art [Vídeo]. YouTube. https://www.youtube.com/watch?v=xmf5vn2yEXE&list=WL&index=25

[19] LoudEyes. (2018, December 15). 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. Retrieved May 13, 2022, from https://dribbble.com/shots/2883582-Low-poly-Big-Ben

[21] Meebit #11748. (s. f.). meebits. Retrieved May 13, 2022, from https://meebits.app/meebits/detail?index=11748

[22] Meg Wayne. (2019, July 14). My top 6 MagicaVoxel tips | 3D city illustration build [Vídeo]. YouTube. https://www.youtube.com/watch?v=MnHRSLrqK9M&list=WL&index=27

[23] Mifik. (2021, November 14). [MagicaVoxel] Voxel art - CYBERPUNK STREET PART-1 [Vídeo]. YouTube. https://www.youtube.com/watch?v=EBF0Tcp1bPo&list=WL&index=24

[24] Omegafoxx. (2017, October 8). [MagicaVoxel] Building a building #1 [Vídeo]. YouTube. https://www.youtube.com/watch?v=JvwqlljBMTA&list=WL&index=31

[25] Pixel Real. (2021, October 18). Patrones Hama Beads Halloween Murciélagos [Pixel Real]. https://pixelreal.net/hama-beads/halloween/murcielagos-2/

[26] S. (2020, April 5). Learn-Computer-Graphics/external/bass at master sho3la/Learn-Computer-Graphics. GitHub. Retrieved May 13, 2022, from https://github.com/sho3la/Learn-Computer-Graphics/tree/master/external/bass

[27] Santacruz, W. (2021, June 26). Glow City n Voxel Art [Vídeo]. YouTube. https://www.youtube.com/watch?v=k7PlVxDJXvs&list=WL&index=28

[28] Shaalan, M. (2020, April 5). how to play sound with opengl c++ ? [Vídeo]. YouTube. 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. Retrieved May 13, 2022, from 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, June 6). 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, May 24). 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, October 25). Voxel Art - Street Light Poles - Magicavoxel [Vídeo]. YouTube. https://www.youtube.com/watch?v=sY4767ESSu4&list=WL&index=21

[33] Vo Thai. (2021, January 4). Voxel Art - Phonebox - Magicavoxel [Vídeo]. YouTube. https://www.youtube.com/watch?v=H6oq3TNVerU&list=WL&index=22

[34] Zimmer, H. (2019, March 14). Nycteris [Vídeo]. YouTube. https://www.youtube.com/watch?v=FtzPpelgPZ8

[35] Laboratory practice reports and code provided in class.