National Autonomous University of Mexico

Faculty of Engineering

Electrical Engineering Division

Computer Engineering

Advanced Computer Graphics

Final project. HALOMHOHGOKAA

Teacher: Ing. Reynaldo Martell Avila
Group: 01

Delivery date: Wednesday, February 10, 2021

Students: Lona López Alberto Alonzo
Santillán García Josué

Semester 2021-1

Introduction

OpenGL is a standard specification that defines a multi-language, cross-platform API for writing applications that produce 2D and 3D graphics. The interface consists of more than 250 different functions that can be used to draw complex three-dimensional scenes from simple geometric primitives, such as points, lines, and triangles. It was originally developed by Silicon Graphics Inc. (SGI) in 1992 and it is widely used in CAD, virtual reality, scientific representation, information visualization, and flight simulation. It is also used in game development, where it competes with Direct3D on Microsoft Windows platforms.

Objective

Create a first-person shooter video game to recreate part of a level from the first Halo in OpenGL with what you have learned throughout the course. In addition to this, recreate a chapter from the novel Halo the flood, in which Sergeant Marvin Mobuto dies while trying to reach the activation rate at the library level and as a tribute to the video game saga that I am so passionate about, to approve the subject and because my teammate gave me the green light to support me with this madness (although seeing the result in the end, perhaps we should have elaborated something simpler).

Hypothesis

With the concepts seen throughout the course, develop the different game mechanics to be able to do a FPS (First Person Shooter).

Analysis

In order to develop an FPS, it is necessary to modify the first person camera that was worked on throughout the semester by limiting the position it can reach on the Z axis, adding and adjusting a model / object type “weapon” that moves next to the camera and from which we can instantiate bullet-type models / objects. In addition to this, create the enemy type models and create a collision box for them, as well as the player's weapon to detect that if they collide with each other, so that if it is the case, destroy the model. All this together with textures and height maps to generate the virtual environment.

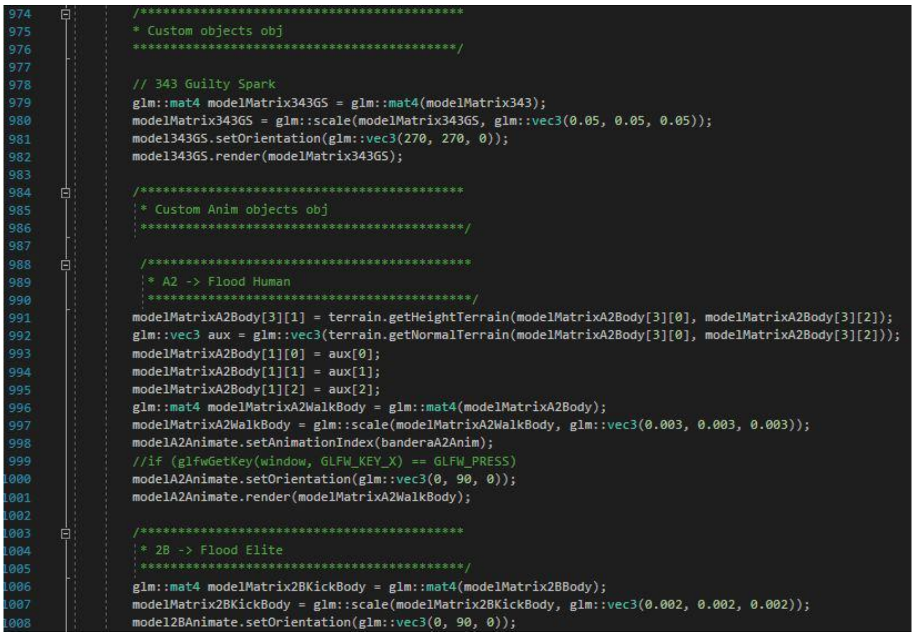
Workplan

Work simultaneously in a team designating complementary but separate tasks between team members and using Git to store the project in a repository, as already mentioned in the previous point, based on what has already been worked on throughout the course , one will begin to elaborate the pertinent adjustments to the code that we can reuse to adjust the camera, the height map, the textures, etc., while the other will search and generate the models and sound effects to use, adjusting them in Blender if necessary. necessary, importing them into the code adjusting their scale, position and rotation, etc. Mainly working in a dual way on separate but complementary rubrics.

Developing

In a way, some parts were developed as a set of deliverables of various practice reports, since at the beginning the models to be used were added, the weapon model and the two variants of the enemies, of the scenario, as well as a simple model of bullet, along with their corresponding arrays, the path in the file explorer to initialize the objects, as well as the buffers for the sound and the collision boxes. Then the code is created to remove the shaders and buffers created.

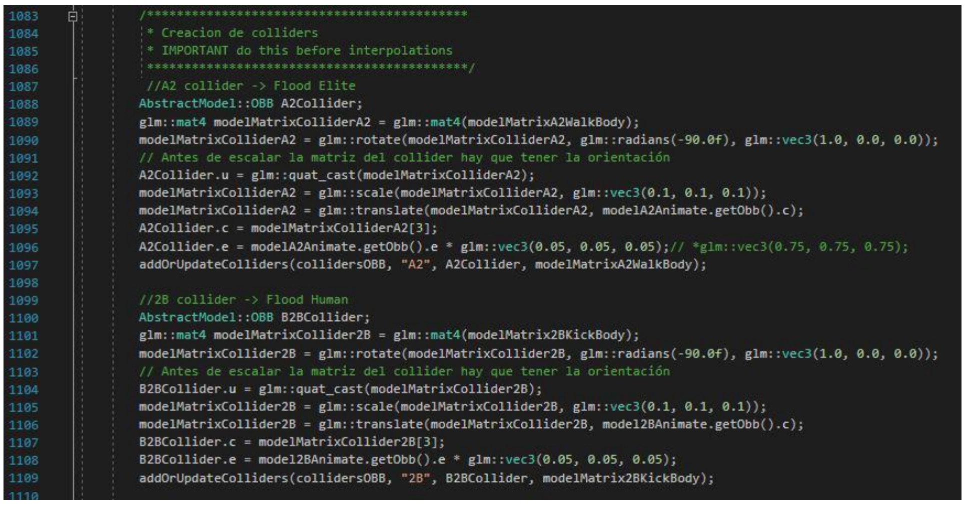
After this, the models began to be instantiated with their position, orientation and scale, to render them.



```
/*******************************************
* Custom objects obj
*******************************************/

// 343 Guilty Spark
glm::mat4 modelMatrix343GS = glm::mat4(modelMatrix343);
modelMatrix343GS = glm::scale(modelMatrix343GS, glm::vec3(0.05, 0.05, 0.05));
model343GS.setOrientation(glm::vec3(270, 270, 0));
model343GS.render(modelMatrix343GS);

/*******************************************
 * Custom Anim objects obj
 *******************************************/

 /*******************************************
  * A2 -> Flood Human
  *******************************************/
modelMatrixA2Body[3][1] = terrain.getHeightTerrain(modelMatrixA2Body[3][0], modelMatrixA2Body[3][2]);
glm::vec3 aux = glm::vec3(terrain.getNormalTerrain(modelMatrixA2Body[3][0], modelMatrixA2Body[3][2]));
modelMatrixA2Body[1][0] = aux[0];
modelMatrixA2Body[1][1] = aux[1];
modelMatrixA2Body[1][2] = aux[2];
glm::mat4 modelMatrixA2WalkBody = glm::mat4(modelMatrixA2Body);
modelMatrixA2WalkBody = glm::scale(modelMatrixA2WalkBody, glm::vec3(0.003, 0.003, 0.003));
modelA2Animate.setAnimationIndex(banderaA2Anim);
//if (glfwGetKey(window, GLFW_KEY_X) == GLFW_PRESS)
modelA2Animate.setOrientation(glm::vec3(0, 90, 0));
modelA2Animate.render(modelMatrixA2WalkBody);

/*******************************************
 * 2B -> Flood Elite
 *******************************************/
glm::mat4 modelMatrix2BKickBody = glm::mat4(modelMatrix2BBody);
modelMatrix2BKickBody = glm::scale(modelMatrix2BKickBody, glm::vec3(0.002, 0.002, 0.002));
model2BAnimate.setOrientation(glm::vec3(0, 90, 0));
```

Figure 1. Code to instantiate objects

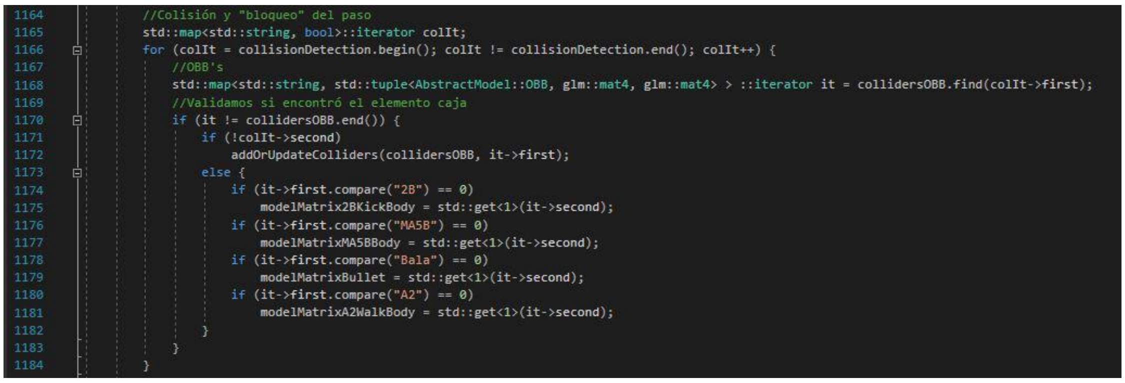
Colliders were created for each object to use.



```
/*******************************************
 * Creacion de colliders
 * IMPORTANT do this before interpolations
 *******************************************/
 //A2 collider -> Flood Elite
AbstractModel::OBB A2Collider;
glm::mat4 modelMatrixColliderA2 = glm::mat4(modelMatrixA2WalkBody);
modelMatrixColliderA2 = glm::rotate(modelMatrixColliderA2, glm::radians(-90.0f), glm::vec3(1.0, 0.0, 0.0));
// Antes de escalar la matriz del collider hay que tener la orientación
A2Collider.u = glm::quat_cast(modelMatrixColliderA2);
modelMatrixColliderA2 = glm::scale(modelMatrixColliderA2, glm::vec3(0.1, 0.1, 0.1));
modelMatrixColliderA2 = glm::translate(modelMatrixColliderA2, modelA2Animate.getObb().c);
A2Collider.c = modelMatrixColliderA2[3];
A2Collider.e = modelA2Animate.getObb().e * glm::vec3(0.05, 0.05, 0.05);// *glm::vec3(0.75, 0.75, 0.75);
addOrUpdateColliders(collidersOBB, "A2", A2Collider, modelMatrixA2WalkBody);

//2B collider -> Flood Human
AbstractModel::OBB B2BCollider;
glm::mat4 modelMatrixCollider2B = glm::mat4(modelMatrix2BKickBody);
modelMatrixCollider2B = glm::rotate(modelMatrixCollider2B, glm::radians(-90.0f), glm::vec3(1.0, 0.0, 0.0));
// Antes de escalar la matriz del collider hay que tener la orientación
B2BCollider.u = glm::quat_cast(modelMatrixCollider2B);
modelMatrixCollider2B = glm::scale(modelMatrixCollider2B, glm::vec3(0.1, 0.1, 0.1));
modelMatrixCollider2B = glm::translate(modelMatrixCollider2B, model2BAnimate.getObb().c);
B2BCollider.c = modelMatrixCollider2B[3];
B2BCollider.e = model2BAnimate.getObb().e * glm::vec3(0.05, 0.05, 0.05);
addOrUpdateColliders(collidersOBB, "2B", B2BCollider, modelMatrix2BKickBody);
```

Figure 2. Code to create some colliders

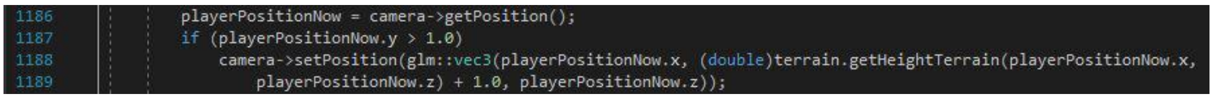
We took part of the code from what we saw in class to recreate the collisions of the cars as we learned it throughout the semester, but adjusting it to our needs.



```
//Colisión y "bloqueo" del paso
std::map<std::string, bool>::iterator colIt;
for (colIt = collisionDetection.begin(); colIt != collisionDetection.end(); colIt++) {
    //OBB's
    std::map<std::string, std::tuple<AbstractModel::OBB, glm::mat4, glm::mat4> > ::iterator it = collidersOBB.find(colIt->first);
    //Validamos si encontró el elemento caja
    if (it != collidersOBB.end()) {
        if (!colIt->second)
            addOrUpdateColliders(collidersOBB, it->first);
        else {
            if (it->first.compare("2B") == 0)
                modelMatrix2BKickBody = std::get<1>(it->second);
            if (it->first.compare("MA5B") == 0)
                modelMatrixMA5BBody = std::get<1>(it->second);
            if (it->first.compare("Bala") == 0)
                modelMatrixBullet = std::get<1>(it->second);
            if (it->first.compare("A2") == 0)
                modelMatrixA2WalkBody = std::get<1>(it->second);
        }
    }
}
```

Figure 3. Code for box-to-box collision

For the development of the camera, we developed the following code snippet.

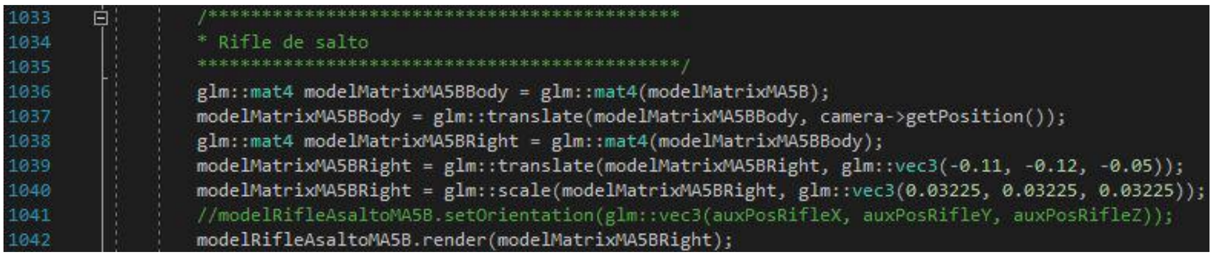


```
playerPositionNow = camera->getPosition();
if (playerPositionNow.y > 1.0)
    camera->setPosition(glm::vec3(playerPositionNow.x, (double)terrain.getHeightTerrain(playerPositionNow.x,
        playerPositionNow.z) + 1.0, playerPositionNow.z));
```

Figure 4. Code to position the camera.

It consists of positioning the camera according to the level of the ground, and constantly comparing the level of the camera, in order to establish a barrier and prevent the camera from becoming floating.

The following code states that the weapon is following the player



```
/*********************************************
* Rifle de salto
*********************************************/
glm::mat4 modelMatrixMA5BBody = glm::mat4(modelMatrixMA5B);
modelMatrixMA5BBody = glm::translate(modelMatrixMA5BBody, camera->getPosition());
glm::mat4 modelMatrixMA5BRight = glm::mat4(modelMatrixMA5BBody);
modelMatrixMA5BRight = glm::translate(modelMatrixMA5BRight, glm::vec3(-0.11, -0.12, -0.05));
modelMatrixMA5BRight = glm::scale(modelMatrixMA5BRight, glm::vec3(0.03225, 0.03225, 0.03225));
//modelRifleAsaltoMA5B.setOrientation(glm::vec3(auxPosRifleX, auxPosRifleY, auxPosRifleZ));
modelRifleAsaltoMA5B.render(modelMatrixMA5BRight);
```

Figure 5. Code to position the weapon based on the position of the camera.

For the development of the level, the following reference image was used.

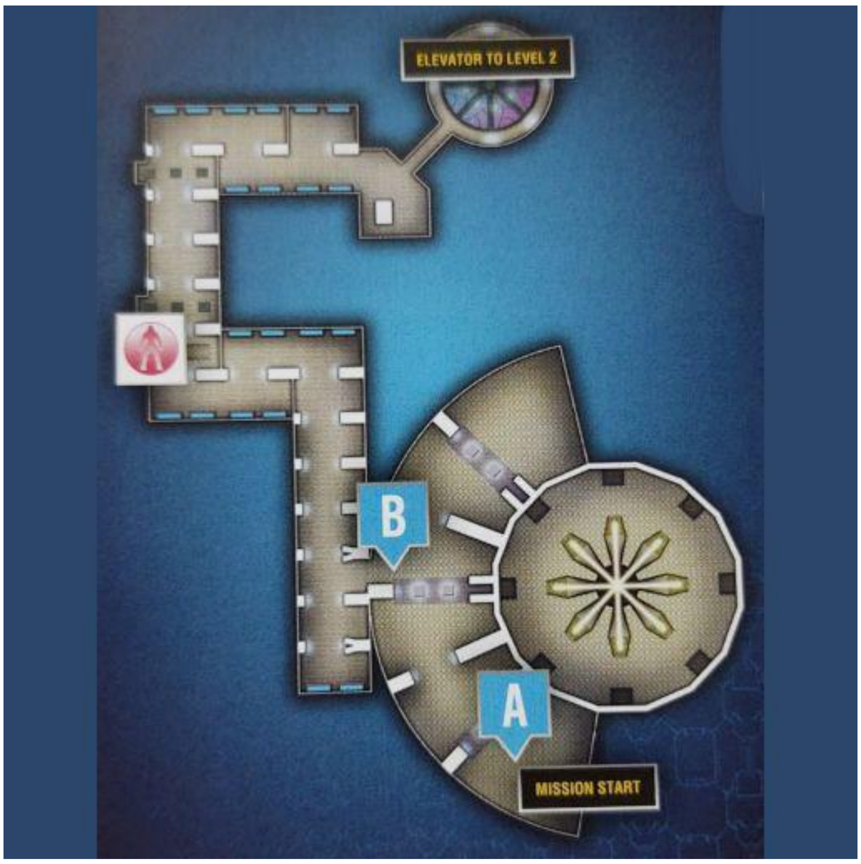


Figure 6. Image taken from the Halo: Combat Evolved book.

The following models were developed in the blender program.

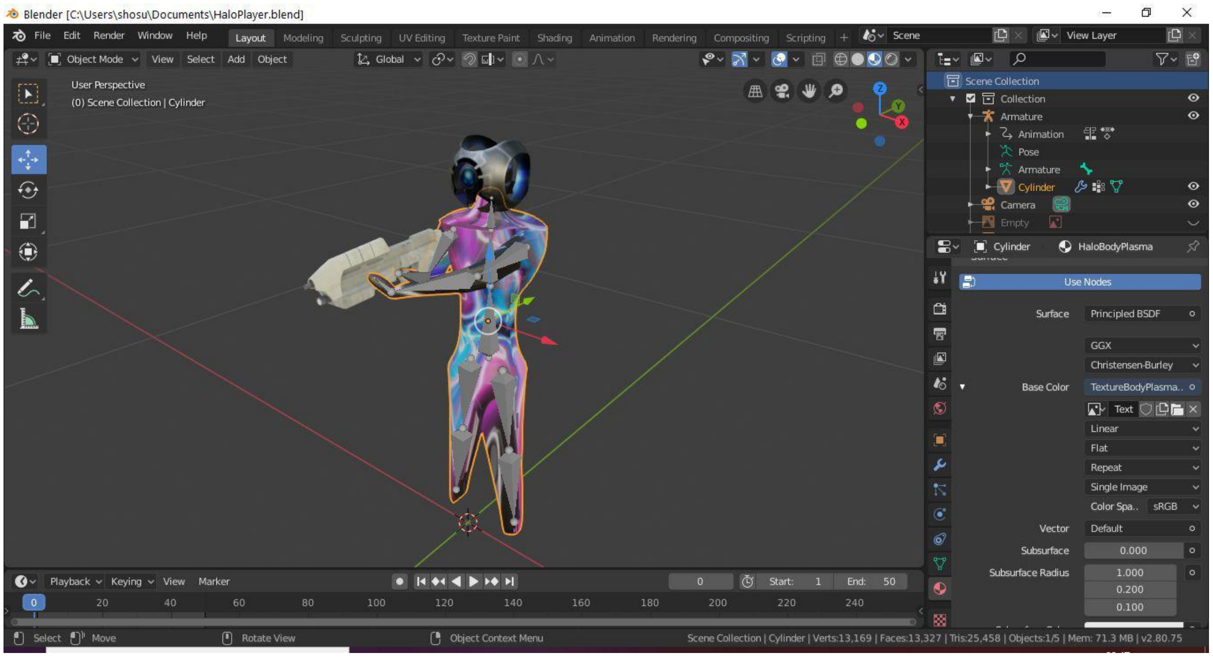
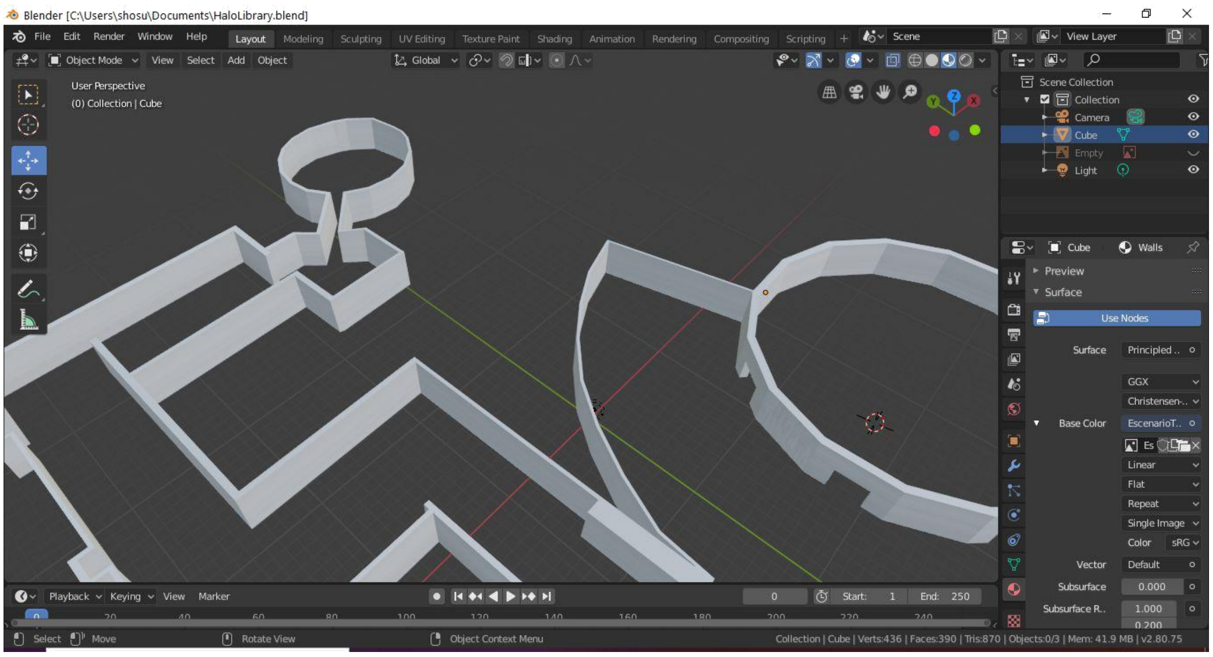

Blender [C:\Users\shosu\Documents\HaloPlayer.blend]
File Edit Render Window Help Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Scripting Scene View Layer
Object Mode View Select Add Object Global
User Perspective
(0) Scene Collection | Cylinder
Scene Collection
Collection
Armature
Animation
Pose
Armature
Cylinder
Camera
Empty
Cylinder HaloBodyPlasma
Use Nodes
Surface Principled BSDF
GGX
Christensen-Burley
Base Color TextureBodyPlasma..
Text
Linear
Flat
Repeat
Single Image
Color Spa.. sRGB
Vector Default
Subsurface 0.000
Subsurface Radius 1.000 0.200 0.100
Playback Keying View Marker
0 Start: 1 End: 50
0 20 40 60 80 100 120 140 160 180 200 220 240
Select Move Rotate View Object Context Menu
Scene Collection | Cylinder | Verts:13,169 | Faces:13,327 | Tris:25,458 | Objects:1/5 | Mem: 71.3 MB | v2.80.75

Figure 7. Model representing the player.



Surface: Principled ..
GGX
Christensen-..
Base Color: EscenarioT..
Linear
Flat
Repeat
Single Image
Color: sRG
Vector: Default
Subsurface: 0.000
Subsurface R..: 1.000, 0.200

Start: 1 End: 250

Collection | Cube | Verts:436 | Faces:390 | Tris:870 | Objects:0/3 | Mem: 41.9 MB | v2.80.75

Figure 8. Model representing the base of the game.

And with that same image, we developed the height map and the mix map of texture.

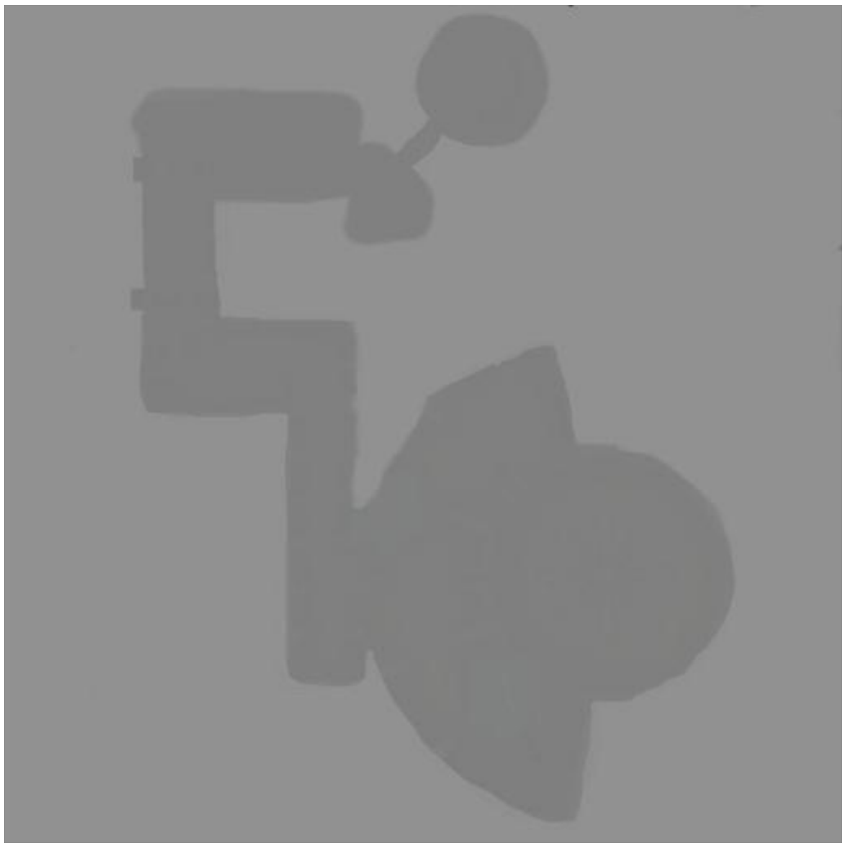


Figure 9. Height map of the game.

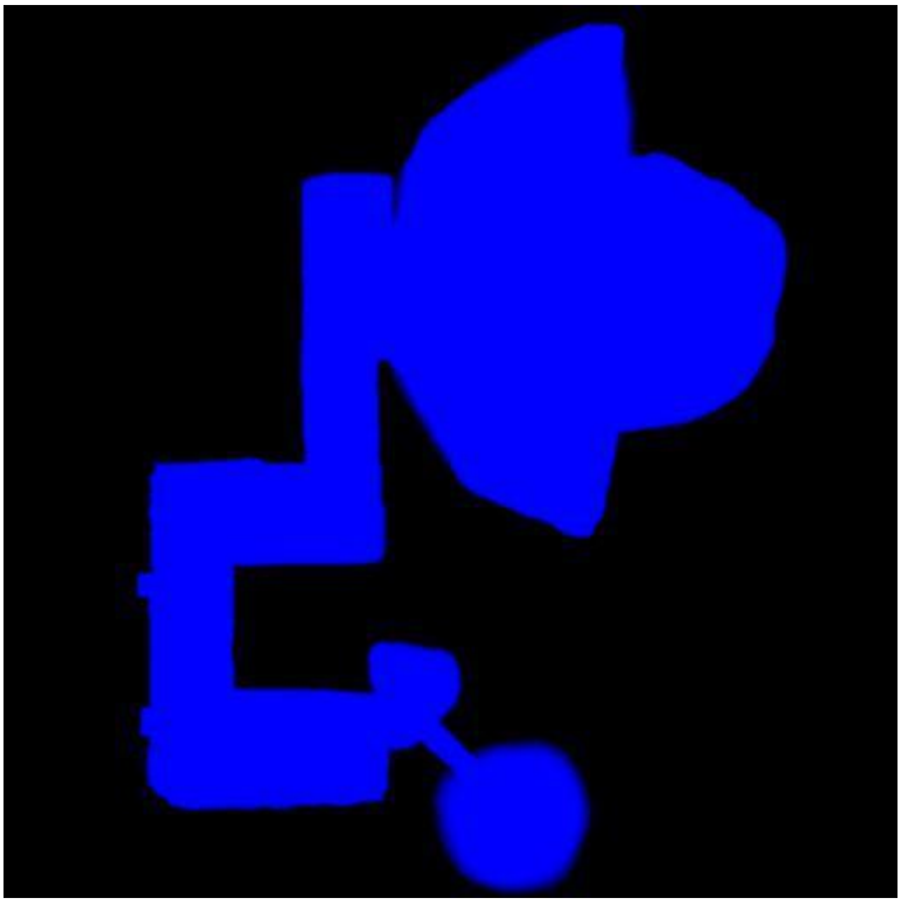


Figure 10. Game texture mix map.

Itemized Cost

How it was established in the objectives of the project, this project is really a tribute to a franchise and video game universe that one of the team members is passionate about, so the cost would be ruled out to some extent.

Discharge

Due to the particularity of the project, licenses belonging to 343 Industries and Microsoft are used, so if the project continues to focus on being a tribute, it is best not to try to license it.

Main challenges and difficulties

The main challenge we faced was time, although at the beginning a planning of time and activities was made, this was not fully respected due to external factors, such as the time allocated to other subjects, as well as the fact of working and distance study. Apart from that problem, another of the main challenges we encountered was having previously worked with video game development engines such as Unity or Unreal and when developing in OpenGL, there were things that we did not know how to solve or translate into the code.

Another difficulty was to capture or "print" text within the OpenGL screen since although there is documentation for it, when deciding whether or not to implement within the project we decided not to do so, giving priority to missing essential mechanics.

Upgrades

One of the main improvements and remaining mechanics is the life or respawn system since although it was possible to have a collision between the player (or the player's weapon) and the enemies, it was not possible to implement that it "died" or destroy the object. In addition to this, the playback times of the sounds could be improved since they are always in a loop and in the long run it is a bit annoying.

Another improvement would be a particle system for when the weapon is fired or add animations to the enemy models, since they only have the attack animation.

Future work

There are really quite a few things to add but the main feature would be to make the game "playable" or with a certain purpose, since in this state it is in an alpha (or pre-alpha) phase, not because of the subject but as evidence to feed a project portfolio for team members.

Conclusions

Lona López Alberto Alonzo: The project was quite ambitious and that's fine, part of life is doing things that have that touch of unattainable or difficult to strive to achieve it or fail miserably while you learn, which in this case but what was achieved at the time of Make this report and taking the real time in which it was prepared, I am satisfied with the result.

Santillán García Josué: This project was quite “studded” (so to speak) but I think it is a great example of how success was in the console industry at the time, and not only because of the environment where it takes place, but because it was a great example of how graphical computing can be very powerful. However, with this small project, if we compare it with the original Halo, this project does require a lot of time and with more people in different areas of video game development (programmers, modelers, post production, etc.).