Universidad Nacional Autónoma de México

Facultad de Ingeniería

División de Ingeniería Eléctrica

Ingeniería en Computación

Computación Gráfica Avanzada

Proyecto Final. HALOMHOHGOKAA

Profesor: Ing. Reynaldo Martell Avila

Grupo: 01

Fecha de entrega: Miércoles 10 de febrero de 2021

Alumnos: Lona López Alberto Alonzo

Santillán García Josué

Semestre 2021-1

Introducción

OpenGL es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. La interfaz consiste en más de 250 funciones diferentes que pueden usarse para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos. Fue desarrollada originalmente por Silicon Graphics Inc. (**SGI**) en 1992 y se usa ampliamente en CAD, realidad virtual, representación científica, visualización de información y simulación de vuelo. También se usa en desarrollo de videojuegos, donde compite con Direct3D en plataformas Microsoft Windows.

Objetivo

Elaborar un videojuego de disparos en primera persona para recrear parte de un nivel del primer Halo en OpenGL con lo aprendido a lo largo del curso. Además de esto recrear un capítulo de la novela Halo the flood, en el cual el sargento Marvin Mobuto muere al intentar alcanzar el índice de activación en el nivel de la biblioteca y como un tributo a la saga de videojuegos que tanto me apasiona, para aprobar la asignatura y porque mi compañero de equipo me dio luz verde para apoyarme con esta locura (aunque viendo el resultado al final quizás debimos elaborar algo más sencillo).

Hipótesis

Con los conceptos vistos a lo largo del curso, elaborar las distintas mecánicas de juego para poder hacer un FPS (First Person Shooter).

Análisis

Para poder elaborar un FPS, se necesita modificar la cámara en primera persona que se trabajó a lo largo del semestre limitando la posición que puede alcanzar en el eje Z, añadiendo y ajustando un modelo/objeto tipo “arma”que se mueva junto a la cámara y de la cual podamos instanciar modelos/objetos tipo bala. Además de esto, crear los modelos del tipo enemigo y crearles una caja de colisión, así como al arma del jugador para lograr detectar que si estos colisionan entre sí, para que si es el caso destruir el modelo. Todo esto junto con texturas y mapas de altura para generar el ambiente virtual.

Plan de trabajo

Trabajar de manera simultánea en equipo designando tareas complementarias pero separadas entre los miembros del equipo y haciendo uso de Git para almacenar el proyecto en un repositorio, como ya se mencionó en el punto anterior a partir de lo que ya se trabajó a lo largo del curso, uno empezará a elaborar los ajustes pertinentes al código que podemos reutilizar para ajustar la cámara, el mapa de alturas, las texturas, etc., mientras que el otro buscará y generará los modelos y efectos de sonido a utilizar, ajustándolos en Blender de ser necesario, importándolos al código ajustando su escala, posición y rotación, etc. Principalmente trabajando de manera dual en rúbricas separadas pero complementarias entre sí.

Desarrollo

De cierto modo algunas partes se desarrollaron como un conjunto de entregables de varios reportes de práctica, ya que al inicio se agregaron los modelos a utilizar, el modelo del arma y las dos variantes de los enemigos, del escenario, así como un modelo simple de bala, junto con sus matrices correspondientes, la ruta en el explorador de archivos para inicializar los objetos, al igual que los buffers para el sonido y las cajas de colisiones. Después se crea el código para eliminar los shaders y buffers creados.

Después de esto se empezó a instanciar los modelos con su posición, orientación y escala, para renderizarlos.

```
974     /*******************************************
975     * Custom objects obj
976     *******************************************/
977
978     // 343 Guilty Spark
979     glm::mat4 modelMatrix343GS = glm::mat4(modelMatrix343);
980     modelMatrix343GS = glm::scale(modelMatrix343GS, glm::vec3(0.05, 0.05, 0.05));
981     model343GS.setOrientation(glm::vec3(270, 270, 0));
982     model343GS.render(modelMatrix343GS);
983
984     /*******************************************
985     * Custom Anim objects obj
986     *******************************************/
987
988      /*******************************************
989      * A2 -> Flood Human
990      *******************************************/
991     modelMatrixA2Body[3][1] = terrain.getHeightTerrain(modelMatrixA2Body[3][0], modelMatrixA2Body[3][2]);
992     glm::vec3 aux = glm::vec3(terrain.getNormalTerrain(modelMatrixA2Body[3][0], modelMatrixA2Body[3][2]));
993     modelMatrixA2Body[1][0] = aux[0];
994     modelMatrixA2Body[1][1] = aux[1];
995     modelMatrixA2Body[1][2] = aux[2];
996     glm::mat4 modelMatrixA2WalkBody = glm::mat4(modelMatrixA2Body);
997     modelMatrixA2WalkBody = glm::scale(modelMatrixA2WalkBody, glm::vec3(0.003, 0.003, 0.003));
998     modelA2Animate.setAnimationIndex(banderaA2Anim);
999     //if (glfwGetKey(window, GLFW_KEY_X) == GLFW_PRESS)
1000    modelA2Animate.setOrientation(glm::vec3(0, 90, 0));
1001    modelA2Animate.render(modelMatrixA2WalkBody);
1002
1003    /*******************************************
1004    * 2B -> Flood Elite
1005    *******************************************/
1006    glm::mat4 modelMatrix2BKickBody = glm::mat4(modelMatrix2BBody);
1007    modelMatrix2BKickBody = glm::scale(modelMatrix2BKickBody, glm::vec3(0.002, 0.002, 0.002));
1008    model2BAnimate.setOrientation(glm::vec3(0, 90, 0));
```

Figura 1. Código para instanciar objetos

Se crearon los colliders para cada objeto a usar.

```
1083    /*******************************************
1084    * Creacion de colliders
1085    * IMPORTANT do this before interpolations
1086    *******************************************/
1087     //A2 collider -> Flood Elite
1088    AbstractModel::OBB A2Collider;
1089    glm::mat4 modelMatrixColliderA2 = glm::mat4(modelMatrixA2WalkBody);
1090    modelMatrixColliderA2 = glm::rotate(modelMatrixColliderA2, glm::radians(-90.0f), glm::vec3(1.0, 0.0, 0.0));
1091    // Antes de escalar la matriz del collider hay que tener la orientación
1092    A2Collider.u = glm::quat_cast(modelMatrixColliderA2);
1093    modelMatrixColliderA2 = glm::scale(modelMatrixColliderA2, glm::vec3(0.1, 0.1, 0.1));
1094    modelMatrixColliderA2 = glm::translate(modelMatrixColliderA2, modelA2Animate.getObb().c);
1095    A2Collider.c = modelMatrixColliderA2[3];
1096    A2Collider.e = modelA2Animate.getObb().e * glm::vec3(0.05, 0.05, 0.05);// *glm::vec3(0.75, 0.75, 0.75);
1097    addOrUpdateColliders(collidersOBB, "A2", A2Collider, modelMatrixA2WalkBody);
1098
1099    //2B collider -> Flood Human
1100    AbstractModel::OBB B2BCollider;
1101    glm::mat4 modelMatrixCollider2B = glm::mat4(modelMatrix2BKickBody);
1102    modelMatrixCollider2B = glm::rotate(modelMatrixCollider2B, glm::radians(-90.0f), glm::vec3(1.0, 0.0, 0.0));
1103    // Antes de escalar la matriz del collider hay que tener la orientación
1104    B2BCollider.u = glm::quat_cast(modelMatrixCollider2B);
1105    modelMatrixCollider2B = glm::scale(modelMatrixCollider2B, glm::vec3(0.1, 0.1, 0.1));
1106    modelMatrixCollider2B = glm::translate(modelMatrixCollider2B, model2BAnimate.getObb().c);
1107    B2BCollider.c = modelMatrixCollider2B[3];
1108    B2BCollider.e = model2BAnimate.getObb().e * glm::vec3(0.05, 0.05, 0.05);
1109    addOrUpdateColliders(collidersOBB, "2B", B2BCollider, modelMatrix2BKickBody);
1110
```

Figura 2. Código para crear algunos colliders

Tomamos parte del código de lo visto en clase para recrear las colisiones de los tal y como lo aprendimos a lo largo del semestre, pero ajustándolo a nuestras necesidades.

```
1164        //Colisión y "bloqueo" del paso
1165        std::map<std::string, bool>::iterator colIt;
1166        for (colIt = collisionDetection.begin(); colIt != collisionDetection.end(); colIt++) {
1167            //OBB's
1168            std::map<std::string, std::tuple<AbstractModel::OBB, glm::mat4, glm::mat4> > ::iterator it = collidersOBB.find(colIt->first);
1169            //Validamos si encontró el elemento caja
1170            if (it != collidersOBB.end()) {
1171                if (!colIt->second)
1172                    addOrUpdateColliders(collidersOBB, it->first);
1173                else {
1174                    if (it->first.compare("2B") == 0)
1175                        modelMatrix2BKickBody = std::get<1>(it->second);
1176                    if (it->first.compare("MA5B") == 0)
1177                        modelMatrixMA5BBody = std::get<1>(it->second);
1178                    if (it->first.compare("Bala") == 0)
1179                        modelMatrixBullet = std::get<1>(it->second);
1180                    if (it->first.compare("A2") == 0)
1181                        modelMatrixA2WalkBody = std::get<1>(it->second);
1182                }
1183            }
1184        }
```

Figura 3. Código para colisión caja con caja

Para el desarrollo de la cámara, desarrollamos el siguiente fragmento de código.

```
1186        playerPositionNow = camera->getPosition();
1187        if (playerPositionNow.y > 1.0)
1188            camera->setPosition(glm::vec3(playerPositionNow.x, (double)terrain.getHeightTerrain(playerPositionNow.x,
1189                playerPositionNow.z) + 1.0, playerPositionNow.z));
```

Figura 4. Código para posicionar la cámara.

Consiste en posicionar a la cámara según sea el nivel del terreno, y estar constantemente comparando el nivel de la cámara, para poder establecer una barrera y evitar que la cámara se vuelva flotante.

El siguiente código establece que el arma esté siguiendo al jugador

```
1033        /*******************************************
1034        * Rifle de salto
1035        *******************************************/
1036        glm::mat4 modelMatrixMA5BBody = glm::mat4(modelMatrixMA5B);
1037        modelMatrixMA5BBody = glm::translate(modelMatrixMA5BBody, camera->getPosition());
1038        glm::mat4 modelMatrixMA5BRight = glm::mat4(modelMatrixMA5BBody);
1039        modelMatrixMA5BRight = glm::translate(modelMatrixMA5BRight, glm::vec3(-0.11, -0.12, -0.05));
1040        modelMatrixMA5BRight = glm::scale(modelMatrixMA5BRight, glm::vec3(0.03225, 0.03225, 0.03225));
1041        //modelRifleAsaltoMA5B.setOrientation(glm::vec3(auxPosRifleX, auxPosRifleY, auxPosRifleZ));
1042        modelRifleAsaltoMA5B.render(modelMatrixMA5BRight);
```

Figura 5. Código para posicionar el arma en base a la posición de la camara.

Para el desarrollo del nivel se utilizó la siguiente imagen de referencia.

Figura 6. Imagen tomada del libro de Halo: Combat Evolved.

Los siguientes modelos fueron desarrollados en el programa de blender.

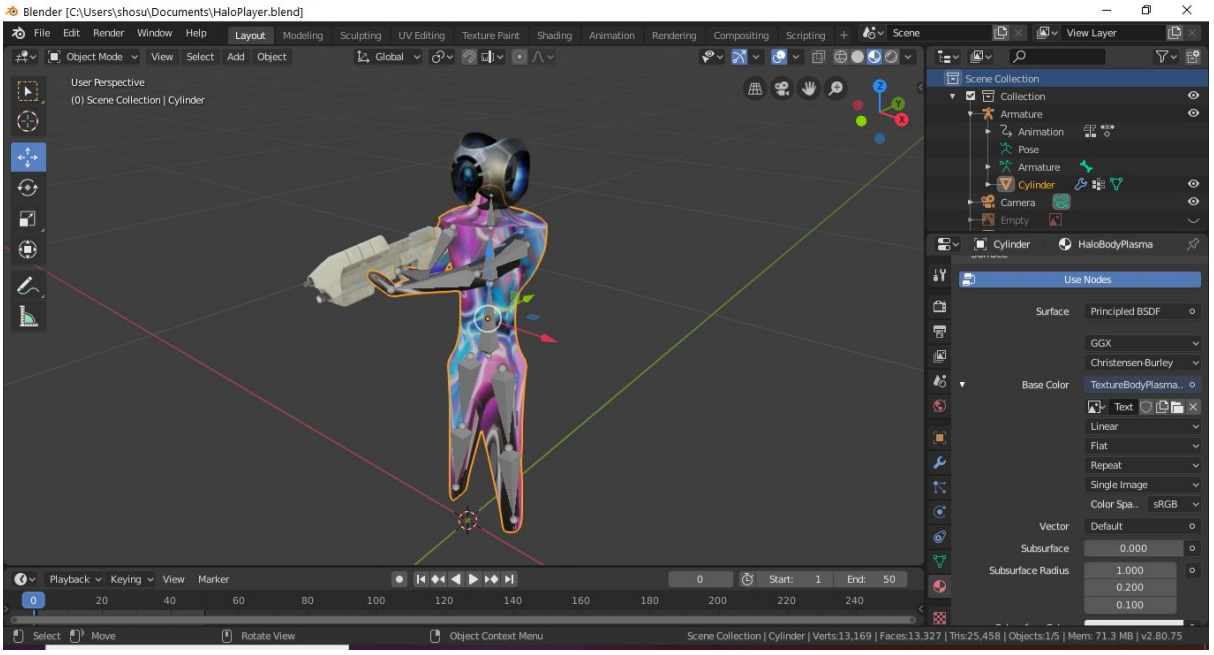

Figura 7. Modelo que representa al jugador.

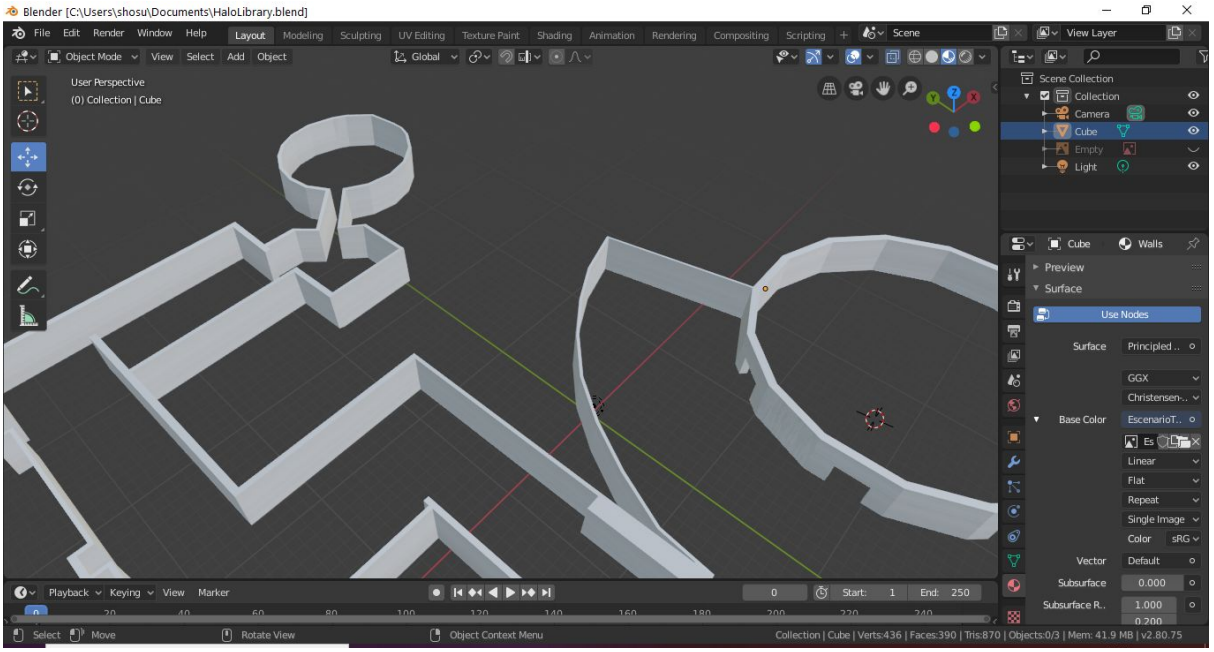

Figura 8. Modelo que representa a la base del juego.

Y con esa misma imagen, desarrollamos el mapa de alturas y el mapa de mezcla de textura.

Figura 9. Mapa de alturas del juego.

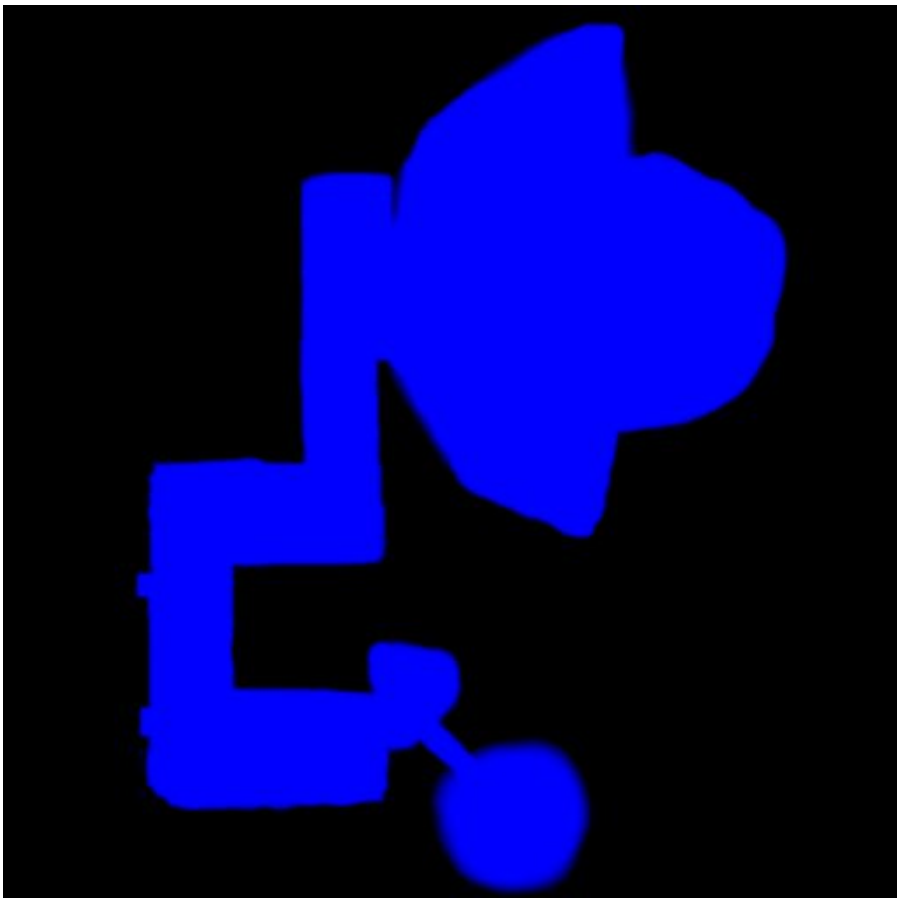

Figura 10. Mapa de mezcla de texturas del juego.

Costo Desglosado

Cómo se estableció en los objetivos del proyecto, realmente este proyecto es un tributo para una franquicia y universo de videojuegos que le apasiona a uno de los integrantes del equipo por lo que el costo quedaría descartado hasta cierto punto.

Licenciamiento

Debido a la particularidad del proyecto se usan licencias pertenecientes a 343 Industries y a Microsoft, por lo que si el proyecto sigue enfocándose en ser un tributo lo mejor es no intentar licenciarlo.

Principales retos y dificultades

El principal reto al cual nos enfrentamos fue el tiempo, si bien al inicio se hizo una planificación de tiempo y actividades esta no se respetó del todo debido a factores externos, como el tiempo asignado a las demás asignaturas, así como el hecho de trabajar y estudiar a distancia. Aparte de ese problema otro de los principales retos que nos encontramos fue el haber trabajado antes con motores de desarrollo de videojuegos como Unity o Unreal y al momento de desarrollar en OpenGL, había cosas que no supimos cómo solucionar o plasmar dentro del código.

Otra dificultad fue plasmar o “imprimir” texto dentro de la pantalla de OpenGL ya que si bien existe documentación para ello, al momento de decidir si implementar o no dentro del proyecto decidimos no hacerlo dándole prioridad a mecánicas esenciales faltantes.

Mejoras

Una de las principales mejoras y mecánicas restantes es el sistema de vida o de respawn ya que si bien se logró tener una colisión entre el jugador (o bien, el arma del jugador) y los enemigos, no se logró implementar que se “muriera” o destruyera el objeto. Además de esto se podrían mejorar los tiempos de reproducción de los sonidos ya que siempre están en loop y a la larga es un poco molesto.

Otra de las mejoras sería un sistema de partículas para cuando se dispara el arma o agregar animaciones a los modelos de los enemigos, ya que estos sólo tienen la animación de ataque.

Trabajo a futuro

Realmente hay bastantes cosas a añadir pero la característica principal sería hacer el juego “jugable” o con cierto propósito, ya que en este estado se encuentra en una fase alfa (o pre-alfa), no por la asignatura sino como evidencia para alimentar un portafolio de proyectos para los miembros del equipo.

Conclusiones

Lona López Alberto Alonzo: El proyecto fue bastante ambicioso y eso está bien, parte de la vida es hacer cosas que tengan ese toque de inalcanzable o difícil para esforzarse por lograrlo o fracasar estrepitosamente mientras aprendes, cosa que en este caso pero de lo que se logró hacer al momento de realizar este reporte y tomando el tiempo real en que se elaboró estoy satisfecho con el resultado.

Santillán García Josué: Este proyecto fue bastante “talachudo” (por decirlo así) pero creo que es un gran ejemplo de como en su momento fue el éxito en la industria de las consolas, y no solo por el ambiente en donde se desarrolla, sino fue porque era un gran ejemplo de cómo la computación gráfica puede ser muy poderosa. Sin embargo, con este pequeño proyecto, si lo comparamos con el original Halo, este proyecto si requiere de mucho tiempo y con más personas en diferentes áreas del desarrollo de videojuegos (programadores, modeladores, post producción, etc.).