UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Laboratorio

Computación Gráfica Interacción Humano Computadora

PROYECTO FINAL

LABORATORIO DE COMPUTACIÓN GRÁFICA

ESPINO DE HORTA JOAQUÍN GUSTAVO

315104271

Semestre 2023-1

12 de enero de 2022

Proyecto El pantano de Shrek

Introducción.

La computación gráfica es el campo de la informática visual, donde se utilizan computadoras tanto para generar imágenes visuales sintéticamente como integrar o cambiar la información visual y espacial probada del mundo real.

Este campo puede ser dividido en varias áreas: Interpretado 3D en tiempo real (a menudo usado en videojuegos), animación de computadora, captura de vídeo y creación de vídeo interpretado, edición de efectos especiales (a menudo usado para películas y televisión), edición de imagen, y modelado (a menudo usado para ingeniería y objetivos médicos). El desarrollo en la gráfica realizada por computadora fue primero alimentado por intereses académicos y patrocinio del gobierno. Sin embargo, cuando las aplicaciones verdaderas mundiales de la gráfica realizada por computadora (CG) en televisión y películas demostraron una alternativa viable a efectos especiales más a las tradicionales y las técnicas de animación, los comerciales han financiado cada vez más el avance de este campo.

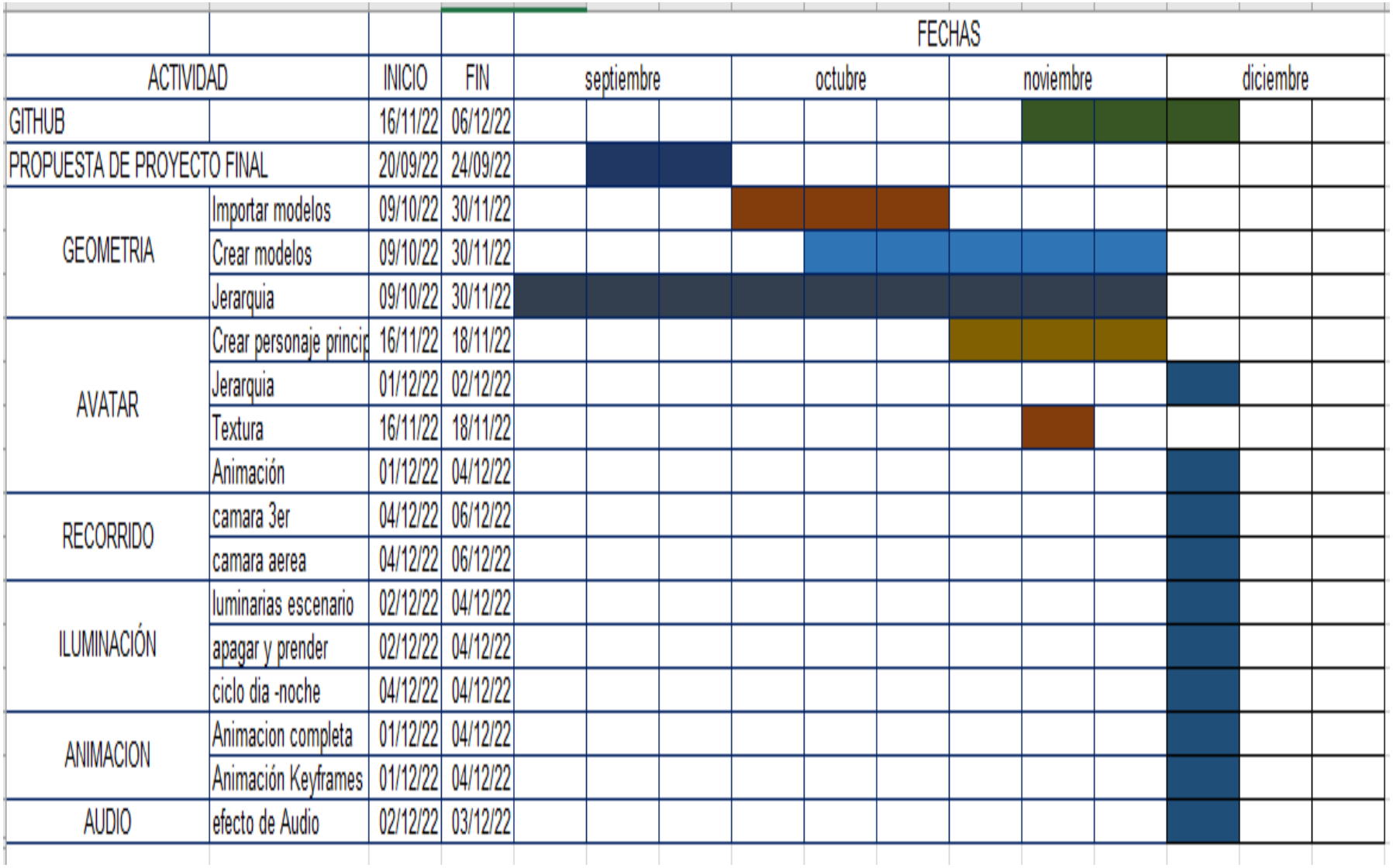
Objetivo

El estudiante combinará las técnicas básicas vistas dentro del laboratorio de cómputo grafica e interacción humano- computadora para el reforzamiento de lo aprendido en las sesiones presenciales.

Lineamientos

El alumno deberá seleccionar una fachada y un espacio que pueden ser reales o ficticios y presentar imágenes de referencia de dichos espacios para su recreación 3D en OpenGL. En la imagen de referencia se debe visualizar 7 objetos que el alumno va a recrear virtualmente y donde dichos objetos deben ser lo más parecido a su imagen de referencia, así como su ambientación.

Diagrama de Gantt



ACTIVIDAD		INICIO	FIN
GITHUB		16/11/22	06/12/22
PROPUESTA DE PROYECTO FINAL		20/09/22	24/09/22
GEOMETRIA	Importar modelos	09/10/22	30/11/22
	Crear modelos	09/10/22	30/11/22
	Jerarquia	09/10/22	30/11/22
AVATAR	Crear personaje princip	16/11/22	18/11/22
	Jerarquia	01/12/22	02/12/22
	Textura	16/11/22	18/11/22
	Animación	01/12/22	04/12/22
RECORRIDO	camara 3er	04/12/22	06/12/22
	camara aerea	04/12/22	06/12/22
ILUMINACIÓN	luminarias escenario	02/12/22	04/12/22
	apagar y prender	02/12/22	04/12/22
	ciclo dia -noche	04/12/22	04/12/22
ANIMACION	Animacion completa	01/12/22	04/12/22
	Animación Keyframes	01/12/22	04/12/22
AUDIO	efecto de Audio	02/12/22	03/12/22

Alcance del proyecto

Imágenes de referencia

Casa del personaje Shrek.

Siete elementos para reproducir

1. La Mesa,
2. Silla,
3. Sillón,
4. La fogata,
5. El Ataúd de Cristal
6. El Baño
7. El letrero por fuera

Documentación del código

Para empezar, el proyecto se purgó y optimizó retirando todo aquel material ajeno del proyecto, como las carpetas Images, Models/Personaje etcétera. Así como su inclusión y llamado en el código.

Se renombraron las shaders para dejar un entendimiento más claro de lo que estaba realizando en el proyecto, pues el nombre de lamp ó lightning si bien era para indicar la interacción de la luz, esto estaba referido a un modelo estandar.

Se sacaron muchas cosas del bucle principal, sobretodo por parte de la configuración de la iluminación, puesto que no se emplearían todos los puntos focales o modificar en cada ciclo los parámetros.

En cuanto características propias del programa. Se optimizó el uso de Keyframes de la siguiente manera

```
typedef struct _frame {//Variables para GUARDAR Key Frames
    float* var;
    struct _frame() { var = nullptr; }
    void setComplexity(const unsigned int complexity) {
        var = new float[complexity];
        for (unsigned int i = 0; i < complexity; i++) var[i] = 0;
    }
}FRAME;
```

La estructura cuenta con un vector flotante de tamaño dinámico para almacenar los parámetros que se requieran animar, desde 3 parámetros para los 3 ejes de una extremidad, 4 parámetros (vector de posición y angulo del eje Y) para la manipulación de un modelo. Todos los componentes del interior.

```
typedef struct _routine {
    FRAME* KF;
    unsigned int complexity, currDetail, nFrames, detail;
    int currFrame;
    float* delta;
    float* value;
    bool play, cicle;
    struct _routine(const unsigned int complexity, const unsigned int nFrames, const unsigned int detail, const bool cicle = false) {
        this->nFrames = nFrames;
        this->complexity = complexity;
        KF = new FRAME[this->nFrames];
        for (unsigned int i = 0; i < this->nFrames; i++) KF[i].setComplexity(this->complexity);
        delta = new float[this->complexity];
        value = new float[this->complexity];
        for (unsigned int i = 0; i < this->complexity; i++) { value[i] = 0; delta[i] = 0; }
        currFrame = 0;
        currDetail = 0;
        this->detail = detail;
        play = false; this->cicle = cicle;
    }
```

El verdadero cambio administrativo está en la estructura de rutina, pues esta contiene todos los valores necesarios que se pueden necesitar de la versión proporcionada por el laboratorio, las ventajas de esta administración es la elección de la complejidad de los KeyFrames, la cantidad de estos y a que detalle se realizan las operaciones de interpolación, como la posibilidad de repetirse en bucle. (Aunque el sistema no transiciona gradualmente entre la última posición y la primera).

Contiene un vector de Keyframes, un vector flotante para los cambios y uno para el valor actual de la animación, así como indicadores de la interpolación y cuadro actual, como banderas booleanas de reproducción o ciclo.

```
void interpolation() {
    for (unsigned int i = 0; i < complexity; i++)
        delta[i] = (KF[currFrame + 1].var[i] - KF[currFrame].var[i]) / detail;
}
void setAtCero() {
    for (unsigned int i = 0; i < complexity; i++) value[i] = KF[0].var[i];
    currFrame = 0;
    currDetail = 0;
    interpolation();
}
void animacion() {//Movimiento del personaje
    if (play)
        if (currDetail >= detail) {//end of animation between frames?
            if (currFrame < nFrames - 2) {//Next frame interpolations
                currDetail = 0; //Reset counter
                currFrame++;
                interpolation();
            }else if (cicle) //end of total animation?
                      setAtCero();
                  else
                      play = false;
            }else{
                for (unsigned int i = 0; i < complexity; i++)//Cambio de las variables
                    value[i] += delta[i];
                currDetail++;
            }
}
```

El método de interpolación es el mismo en cuanto a aritmética, tan solo se modificó para ser cíclico.

SetAtCero es el reinicio formal de la animación, devolviendo a la posición inicial sin transición, colocando los indicadores a cero y realizando la interpolación que especifica el método original.

En cuanto la animación, se conservó la variable de control play para que dejara de ejecutarse aún estando en el bucle principal. Se intercambió la lógica para resolver un problema de reproducción con el último cuadro, las pruebas iniciales arrojaban que hacía falta la transición al último cuadro, por lo que se llevó el aumento temprano del frame actual pero antes de su interpolación. Si es que la animación había terminado, se pregunta si es un bucle para reiniciar, sino se evita ejecutar este procedimiento cambiando la variable play a false.

Y la transición se suma. En un formato de bucle.

```
RT rt_Pos_Shrek(4, 13, 600, false);//Rutina Principal
RT rt_WK(10, 9, 90, true), rt_SB(10, 4, 240, false), rt_VM(10, 4, 450, false);//Rutinas Extremidades Shrek
RT rt_Puerta_Bano(4, 5, 120, false), rt_Puerta_Casa(1, 4, 300, false), rt_Ataud(3, 3, 300, false), rt_Sillas(2, 3, 300, false);//Rutinas del entorno
void setAnim() {
    ////////////////////////////////////////////////////////ANIMACION DE MOVIMIENTO SHREK/////////////////////////////////////////////////////////////
    rt_Pos_Shrek.KF[0].var[0] = 27.0f; rt_Pos_Shrek.KF[0].var[1] = 10.3f; rt_Pos_Shrek.KF[0].var[2] = 23.0f; rt_Pos_Shrek.KF[0].var[3] = -130.0f;//Inicio
    rt_Pos_Shrek.KF[1].var[0] = 26.5f; rt_Pos_Shrek.KF[1].var[1] = 10.3f; rt_Pos_Shrek.KF[1].var[2] = 22.5f; rt_Pos_Shrek.KF[1].var[3] = -130.0f;//Somebody
    rt_Pos_Shrek.KF[2].var[0] = 26.0f; rt_Pos_Shrek.KF[2].var[1] = 10.3f; rt_Pos_Shrek.KF[2].var[2] = 22.0f; rt_Pos_Shrek.KF[2].var[3] = -130.0f;
    rt_Pos_Shrek.KF[3].var[0] = 25.7f; rt_Pos_Shrek.KF[3].var[1] = 10.3f; rt_Pos_Shrek.KF[3].var[2] = 21.1f; rt_Pos_Shrek.KF[3].var[3] = -120.0f;//Caminata
    rt_Pos_Shrek.KF[4].var[0] = 19.0f; rt_Pos_Shrek.KF[4].var[1] =  8.7f; rt_Pos_Shrek.KF[4].var[2] = 17.0f; rt_Pos_Shrek.KF[4].var[3] = -90.0f;
    rt_Pos_Shrek.KF[5].var[0] = 14.0f; rt_Pos_Shrek.KF[5].var[1] =  5.7f; rt_Pos_Shrek.KF[5].var[2] = 13.0f; rt_Pos_Shrek.KF[5].var[3] = -100.0f;
    rt_Pos_Shrek.KF[6].var[0] =  9.0f; rt_Pos_Shrek.KF[6].var[1] =  3.7f; rt_Pos_Shrek.KF[6].var[2] = 10.0f; rt_Pos_Shrek.KF[6].var[3] = -130.0f;
    rt_Pos_Shrek.KF[7].var[0] = 4.34f; rt_Pos_Shrek.KF[7].var[1] =  2.6f; rt_Pos_Shrek.KF[7].var[2] =  8.4f; rt_Pos_Shrek.KF[7].var[3] = -160.0f;
    rt_Pos_Shrek.KF[8].var[0] = 0.02f; rt_Pos_Shrek.KF[8].var[1] =  2.4f; rt_Pos_Shrek.KF[8].var[2] =  7.0f; rt_Pos_Shrek.KF[8].var[3] = -175.0f;
    rt_Pos_Shrek.KF[9].var[0] =-0.03f; rt_Pos_Shrek.KF[9].var[1]  = 2.36f; rt_Pos_Shrek.KF[9].var[2]  = 0.76f; rt_Pos_Shrek.KF[9].var[3]  = -180.0f;//Entrada a la Casa
    rt_Pos_Shrek.KF[10].var[0] =-0.73f; rt_Pos_Shrek.KF[10].var[1] = 2.36f; rt_Pos_Shrek.KF[10].var[2] = 0.76f; rt_Pos_Shrek.KF[10].var[3] = -145.0f;
    rt_Pos_Shrek.KF[11].var[0] = -6.0f; rt_Pos_Shrek.KF[11].var[1] = 2.36f; rt_Pos_Shrek.KF[11].var[2] =-2.36f; rt_Pos_Shrek.KF[11].var[3] = -135.0f;
    rt_Pos_Shrek.KF[12].var[0] = -4.8f; rt_Pos_Shrek.KF[12].var[1] = 2.0f;  rt_Pos_Shrek.KF[12].var[2] = -6.5f; rt_Pos_Shrek.KF[12].var[3] = -270.0f;//Empujar
    rt_Pos_Shrek.setAtCero();////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////////ANIMACION DEL ENTORNO/////////////////////////////////////////////////////////////////////
    rt_Puerta_Bano.KF[0].var[0] = 26.55f; rt_Puerta_Bano.KF[0].var[1] = 10.0f; rt_Puerta_Bano.KF[0].var[2] = 21.16f; rt_Puerta_Bano.KF[0].var[3] = 0.0f;
    rt_Puerta_Bano.KF[1].var[0] = 13.27f; rt_Puerta_Bano.KF[1].var[1] = 15.0f; rt_Puerta_Bano.KF[1].var[2] = 10.30f; rt_Puerta_Bano.KF[1].var[3] = 180.0f;
    rt_Puerta_Bano.KF[2].var[0] =   0.0f; rt_Puerta_Bano.KF[2].var[1] = 20.0f; rt_Puerta_Bano.KF[2].var[2] =  0.18f; rt_Puerta_Bano.KF[2].var[3] = 360.0f;
    rt_Puerta_Bano.KF[3].var[0] =-13.27f; rt_Puerta_Bano.KF[3].var[1] = 15.0f; rt_Puerta_Bano.KF[3].var[2] =-10.30f; rt_Puerta_Bano.KF[3].var[3] = 440.0f;
    rt_Puerta_Bano.KF[4].var[0] =-23.77f; rt_Puerta_Bano.KF[4].var[1] = 10.0f; rt_Puerta_Bano.KF[4].var[2] =-21.01f; rt_Puerta_Bano.KF[4].var[3] = 800.0f;
    rt_Puerta_Bano.setAtCero();////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    rt_Puerta_Casa.KF[0].var[0] = 0.0f; rt_Puerta_Casa.KF[1].var[0] = -120.0f; rt_Puerta_Casa.KF[2].var[0] = -120.0f;
    rt_Puerta_Casa.setAtCero();////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    rt_Ataud.KF[0].var[0] = 0.0f; rt_Ataud.KF[0].var[1] = 0.0f; rt_Ataud.KF[0].var[2] = 0.0f;
    rt_Ataud.KF[1].var[0] = 0.0f; rt_Ataud.KF[1].var[1] = 0.0f; rt_Ataud.KF[1].var[2] = 0.0f;
    rt_Ataud.KF[2].var[0] = 0.0f; rt_Ataud.KF[2].var[1] = 0.0f; rt_Ataud.KF[2].var[2] = 0.0f;
    rt_Ataud.setAtCero();////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    rt_Sillas.KF[0].var[0] = 0.0f; rt_Sillas.KF[0].var[1] = 0.0f;
    rt_Sillas.KF[1].var[0] = 0.0f; rt_Sillas.KF[1].var[1] = 0.0f;
```

Así se realiza la carga de información, primero debemos definir nuestras estructuras con la dimensión de los Frames, el número de estos, la calidad del detalle y de forma opcional, un booleano que indique si esto se repetirá en bucle o no (Por defecto es false) y el proceso de carga es semejante a un arreglo bidimensional, donde se consulta el valor de cada keyframe.

Al final de cada declaración de valores, debe citarse el método setAtCero de cada subrutina para realizar la primera interpolación.

NOTA: Este procedimiento no está automatizado, por lo tanto, no hay algo que regule la cantidad de accesos a la memoria, por lo que si se solicita el acceso a un número igual o mayor al tamaño de la complejidad ó de la cantidad de keyframes, esta dará un error al inicio del programa, aunque todos aquellos valores que tengan asignación, tendrán por defecto 0.0f.

Esta es la forma en la que se maneja la animación (los valores numéricos pueden variar, se siguen haciendo ajustes), se tiene una bandera activada con la tecla “F” al tornarse True, siempre se ejecutará la animación maestra, se decidió usar la rutina de posicionamiento del avatar, específicamente el cuadro en el que realiza la acción para desencadenar las demás rutinas del entorno.

```
float random = 0.0f, deg = 0.025f;
float rot_limbs[10];
for (unsigned int i = 0; i < 10; i++)
    rot_limbs[i] = rt_SB.value[i];

// Game loop
while (!glfwWindowShouldClose(window)){
if (active) { //Animacion general de la escena
    rt_Pos_Shrek.animacion();//Animacion Maestra
    if (one_shot_0) {
        std::thread soundtrack(&playSoundTrack, 0);
        soundtrack.detach();
        one_shot_0 = false;
    }
    if (rt_Pos_Shrek.currFrame > 0) {
        rt_Puerta_Bano.animacion();
        rt_SB.animacion();
    }
    if (rt_Pos_Shrek.currFrame > 2) {
        rt_WK.animacion();
    }

    if (rt_Pos_Shrek.currFrame > 7) {
        rt_Puerta_Casa.animacion();
        rt_Pos_Shrek.detail = 300;
    }

    if (rt_Pos_Shrek.currFrame > 9) {
        if (one_shot_1) {
            std::thread quote(&playSoundTrack, 1);
            quote.detach();
            one_shot_1 = false;
        }
        rt_VM.animacion();
    }
    if (rt_VM.currFrame > 2) {
        rt_Sillas.animacion();
        rt_Ataud.animacion();
    }
```

Se implementó una biblioteca de sonido, existe una función para ejecutarse en paralelo (de otra manera no podría ejecutarse OpenGL) y se utiliza una bandera que solo permita una sola ejecución a la vez, con el fin de prevenir conflictos con el manejo irracional de hilos en el programa.

```
void resetScene() {
    active = false; one_shot_0 = true; one_shot_1 = true; currLight = 0.7f; targetLight = 0.7f;
    rt_Pos_Shrek.play = false; rt_Pos_Shrek.setAtCero(); rt_Pos_Shrek.detail = 600;
    rt_Puerta_Bano.play = false; rt_Puerta_Bano.setAtCero();
    rt_Puerta_Casa.play = false; rt_Puerta_Casa.setAtCero();
    rt_Sillas.play = false; rt_Sillas.setAtCero();
    rt_Ataud.play = false; rt_Ataud.setAtCero();
    rt_SB.play = false; rt_SB.setAtCero();
    rt_WK.play = false; rt_WK.setAtCero();
    rt_VM.play = false; rt_VM.setAtCero();
}
```

Este procedimiento esta hecho para reiniciar los parámetros al valor inicial, no hay forma de hacer esto un proceso general debido a la naturaleza particular de cada rutina, como algunos otros detalles como las banderas one_shot para el audio o los valores lumínicos.

ADVERTENCIA: Si se acciona la tecla destinada al reinicio y los audios de la escena siguen en reproducción, puede crearse un efecto de sobre exposición en el sonido o colapsar el programa, se recomienda esperar un par de segundos después de no escuchar absolutamente nada del programa.

```
const float degree(const float current, const float goal) {
    if (current > goal)
        return current - 0.0001f;
    else if (current < goal)
        return current + 0.0001f;
    return current;
}
```

La función degradado, esta planificada para llegar de un valor A a otro valor B en una prolongación de tiempo. Y detenerse (devolviendo uno de los valores sin alteración) al llegar al destino. Estas 2 funciones son ideales para usarse en la iluminación, especialmente en la simulación de la fogata.

```
if (abs((0.0125f + random) - degree(deg, 0.0125f + random)) < 0.00001)
    random = (float)(std::rand() % 1000) / 10000.0f;
else
    deg = degree(deg, 0.0125 + random);
glUniform1f(glGetUniformLocation(standar.Program, "pointLights[0].linear"), deg);
glUniform1f(glGetUniformLocation(standar.Program, "pointLights[0].quadratic"), deg / 2.0f);
// Global Light
currLight = degree(currLight, targetLight);
glUniform3f(glGetUniformLocation(standar.Program, "dirLight.ambient"), currLight, currLight, currLight);
```

Como podemos apreciar, dependemos de un valor aleatorio para el alcance del PointLight, para dar un acabado suave al efecto, se realiza este rango entre 0.0125f como mínimo (máxima luz) hasta un máximo de 0.1024f (mínima luz). Mientras que la luz de ambiente está sometida a los eventos de la escena, perfectamente podía estar sometida a una rutina como los demás eventos, pero decidió así por su simpleza de implementar.

```
////////////////////Dibujo de Shrek////////////////////
model = glm::translate(glm::mat4(1), glm::vec3(rt_Pos_Shrek.value[0], rt_Pos_Shrek.value[1], rt_Pos_Shrek.value[2]));
model = glm::rotate(model, glm::radians(rt_Pos_Shrek.value[3]), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Torso.Draw(standar);
temp = glm::translate(model, glm::vec3(0.0f, 0.9f, -0.1f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(temp));
Cabeza.Draw(standar);
///BRAZO IZQ Y//BRAZO IZQ Z//ANTBRAZO IZQ Z//BRAZO DER Y//BRAZO DER Z//ANTBRAZO DER Z///PIERNA IZQ X//ANTPIERNA IZQ X//PIERNA DER X//ANTPIERNA X///
//Brazo Izq
temp = glm::translate(model, glm::vec3(0.43f, 0.5f, -0.18f));
temp = glm::rotate(temp, glm::radians(rot_limbs[0]), glm::vec3(0.0f, 1.0f, 0.0f));
temp = glm::rotate(temp, glm::radians(rot_limbs[1]), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(temp));
Brazo.Draw(standar);
//Ant Brazo Izq
temp = glm::translate(temp, glm::vec3(0.75f, 0.0f, 0.0f));
temp = glm::rotate(temp, glm::radians(rot_limbs[2]), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(temp));
Antebrazo.Draw(standar);
//Brazo Der
temp = glm::translate(model, glm::vec3(-0.43f, 0.5f, -0.18f));
temp = glm::rotate(temp, glm::radians(rot_limbs[3]), glm::vec3(0.0f,-1.0f, 0.0f));
temp = glm::rotate(temp, glm::radians(rot_limbs[4]), glm::vec3(0.0f, 0.0f, 1.0f));
temp = glm::scale(temp, glm::vec3(-1.0f, 1.0f, 1.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(temp));
Brazo.Draw(standar);
//Ant Brazo Izq
temp = glm::translate(temp, glm::vec3(0.75f, 0.0f, 0.0f));
temp = glm::rotate(temp, glm::radians(rot_limbs[5]), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(temp));
Antebrazo.Draw(standar);
//Pierna Izq
temp = glm::translate(model, glm::vec3(0.25f, -0.7f, 0.0f));
temp = glm::rotate(temp, glm::radians(rot_limbs[6]), glm::vec3(1.0f, 0.0f, 0.0f));
```

Finalmente su implementación en un modelado jerárquico en la que se toma el value de la subrutina correspondiente. Por otra parte los elementos estáticos como la casa, el letrero o el baño se le aplican directamente las transformaciones de traslación para optimizar recursos. Para ayudar a esa práctica, se escalaron y orientaron todos los modelos en su exportación.

```
if (keys[GLFW_KEY_F]) {
    active = true;
    rt_Pos_Shrek.play = true;
    rt_Puerta_Bano.play = true;
    rt_Puerta_Casa.play = true;
    rt_Sillas.play = true;
    rt_Ataud.play = true;
    rt_SB.play = true;
    rt_WK.play = true;
    rt_VM.play = true;
}
if (keys[GLFW_KEY_R])
    resetScene();
```

Las primeras versiones del programa trataron de abarcar mas allá de los puntos establecidos, pensando en que este objeto podría llegar a contener una textura así como ser probado en programas externos como Blender o Maya mediante la creación de un archivo .obj, debido a la falta de tiempo y demasiadas dificultades técnicas en cuanto a las exigencias de la construcción de este objeto, se decidió recortar su alcance a la simple generación de vértices y los índices que daría lugar para dibujar al objeto.

```
unsigned int getRealSubDiv(const int n);
void constantCircle(std::vector<float>* vec, const float radius, const unsigned int inQual);

void getIndexArray(std::vector<int>* vec,const unsigned int IQ, const unsigned int FQ);
void  getVertexArray(std::vector<float>*  vec,const  float  inRad,  const  float  forRad,  const
unsigned int IQ, const unsigned int FQ);
```

En su estructura utilizaremos 4 métodos.
El primero es utilizado para obtener el número real de las calidades.

```
unsigned int getRealSubDiv(const int n) {
	if (n < 1)
		return 2;
	return 2*n + 2;
}
```

De tal forma que, el mínimo caso que nos pueda dar sea un rectángulo muy delgado en caso de darnos algo inferior a 1. Siendo una rotunda declaración de error por parte del parámetro. Aunque en un caso funcional, nuestro mínimo sería una figura de calidad 4x4 es decir 16 vértices en total y 30 triángulos por dibujar. Se hizo de esta manera para dar una figura mínima a comprender de la computadora y la más consistente en cuanto a su referente de circunferencia.

```
void constantCircle(std::vector<float>* vec, const float radius, const unsigned int IQ) {
	vec->resize(IQ);
	const float incX = PI * 2 / (float)IQ;
	for (int i = 0; i < IQ; i++) {
		const float zeta = radius * (float)std::sin(incX * (float)i);
		 vec->at(i) = 0.00001 < abs(zeta) ? zeta : 0.0f;
	}
}
```

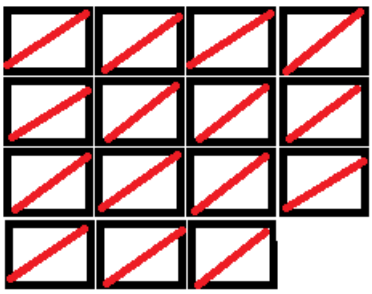
Esta función es concebida para ahorrar costos de procesamiento al momento de la creación, ya que la manera en que trabaja el programa es en replicar una circunferencia en el plano XY a través de los distintos ángulos que nos dicta la calidad exterior en el plano XZ.
Por lo tanto es muy efectivo calcular una sola vez el valor en Y del círculo y concederle los valores correspondientes en X y Z a posterior.

```
void getVertexArray(std::vector<float>* vec, const float inRad, const float forRad, const
unsigned int IQ, const unsigned int FQ) {
	vec->resize(IQ * FQ * 3);
	//std::cout << "Tamanio Vertex Array = " << vec->size() << std::endl;
	const float incAng = 2 * PI / (float)FQ;
	const float incIR = inRad / (float)IQ;
	std::vector<float> inCircle;
	constantCircle(&inCircle, inRad, IQ);
	for (unsigned int i = 0; i < FQ; i++) {
		float pivX = std::cos(incAng * (float)i);
		float pivZ = std::sin(incAng * (float)i);
		pivX = (0.00001 < abs(pivX) ? pivX : 0.0f); pivZ = (0.00001 < abs(pivZ) ?
pivZ : 0.0f);
		for (unsigned int j = 0; j < IQ; j++) {
			vec->at(i * IQ + j * 3) = (forRad + j * incIR) * pivX;
			vec->at(i * IQ + j * 3 + 1) = inCircle.at(j);
			vec->at(i * IQ + j * 3 + 2) = (forRad + j * incIR) * pivZ;
			std::cout << vec->at(i * IQ + j * 3) << "f, " << vec->at(i * IQ + j * 3
+ 1) << "f, " << vec->at(i * IQ + j * 3 + 2) << "f," << std::endl;
		}
	}
}
```

La función principal para la elaboración de los vértices en un inicio calcula las constantes de salto, determinadas por la calidad y las mediciones del radio así como el cálculo del círculo que conforma el perfil del toroide. En el primer nivel se determina mediante el seno y coseno el escalar de su correspondencia en el plano XZ Del cual se incrementará a medida el objeto sea impreso en el vector.

```
void getIndexArray(std::vector<int>* vec, const unsigned int IQ, const unsigned int FQ) {
	vec->resize((IQ * FQ - 1) * 6);
	//std::cout << "Tamanio Index Array = " << vec->size() << std::endl;
	int nx = 0, ny = 0, nz = 0, seq = 0;
	for (int i = 0; i < FQ; i++)
		for (int j = 0; j < IQ; j++)
			if(j != (IQ - 1) || i != (FQ - 1)) {
				nx = i * IQ + j;
				nz = ( i + 1 == FQ ? j : nx + IQ );
				ny = ( j + 1 == IQ ? IQ * (i + 1) :nz + 1);

				vec->at(i * IQ + j) = nx;
				vec->at(i * IQ + j + 1) = ny;
				vec->at(i * IQ + j + 2) = nz;
				vec->at(i * IQ + j + 3) = (nx + 1) % (IQ * (i + 1)) + (j + 1 == IQ
? IQ * i : 0);
				vec->at(i * IQ + j + 4) = ny;
				vec->at(i * IQ + j + 5) = nx;
				std::cout << vec->at(i * IQ + j) << ", " << vec->at(i * IQ + j +
1) << ", " << vec->at(i * IQ + j + 2) << "," << std::endl;
				std::cout << vec->at(i * IQ + j + 3) << ", " << vec->at(i * IQ +
j + 4) << ", " << vec->at(i * IQ + j + 5) << ", " << std::endl;
			}
```

}
Por otra parte, la función que da los índices no depende del valor de los radios sino la calidad del objeto, ya que esta desenvuelve el toroide en una malla y mediante los cálculos necesarios, determina las caras que deban ser colocadas.
Si vemos el punto de vista de una cuadrícula.
Podemos comprender que si seguimos el orden descendente y hacia la derecha, comprendemos que el primer triángulo del punto está conformado por nuestro punto "protagonista" el de la siguiente columna a su misma altura y su inmediato. De manera semejante se comporta el otro triángulo que conforma su celda, es el inmediato y sus 2 vecinos.



Por lo que este algoritmo se encarga de calcularlo con base en la calidad que representa encerrando a estos valores en una iteración adicional que conecta con el inicio como lo debe hacer el toroide por su definición dejando una celda libre.

Resultados:
Para implementarse en el proyecto, se requiere de una Shader primitiva que pueda únicamente los datos de posición e interprete un color constante sin alteraciones en los focos de luz.

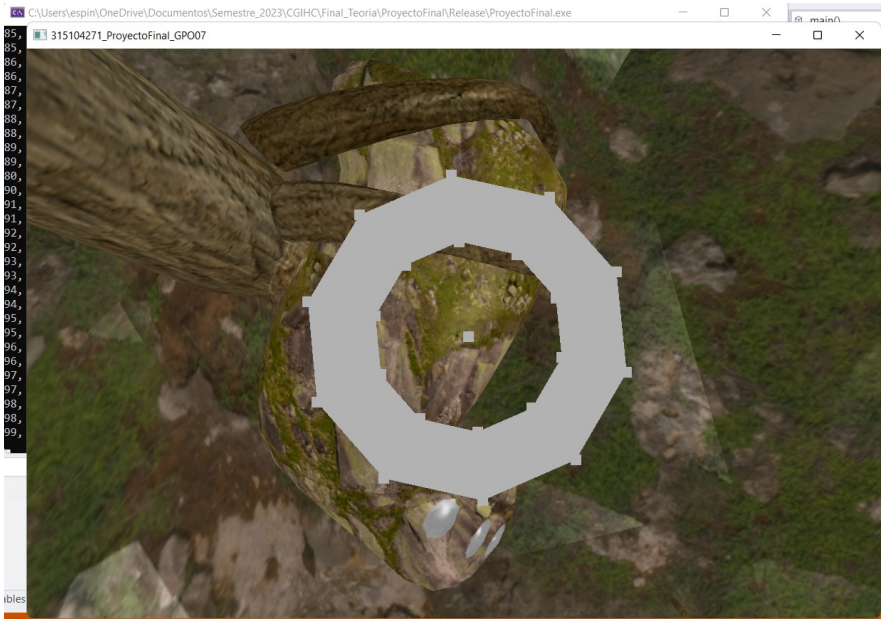
Así como volver a preparar los VertexArrayObjects, VertexBufferObjec y el ElementBufferObject con el tamaño de la función.

```
GLuint VBO, VAO, EBO;
	glGenVertexArrays(1, &VAO);
	glGenBuffers(1, &VBO);
	glGenBuffers(1, &EBO);
	glBindVertexArray(VAO);
	glBindBuffer(GL_ARRAY_BUFFER, VBO);
	glBufferData(GL_ARRAY_BUFFER, sizeof(pVert), pVert, GL_STATIC_DRAW);
	//glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(float) * 2, vertices.data(), 
GL_STATIC_DRAW);
	glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
	glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(pIndx), pIndx, GL_STATIC_DRAW);
	//glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(int) * 2, 
indices.data(), GL_STATIC_DRAW);

	glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);
	glEnableVertexAttribArray(0);

	glBindBuffer(GL_ARRAY_BUFFER, 0);

	glBindVertexArray(0);
```

Sin embargo…

Esta es una captura de lo que sucede, ilustra tanto los puntos como el enlace de sus caras. De una forma desconocida, la información aun cuando se use el objeto vector para almacenar la información, esta es corrompida al entrar al buffer de datos.

De tal manera que se implementaron exactamente los mismos valores en arreglos estáticos



y este es el resultado.

Por lo tanto se concluye que el algoritmo entrega valores correctos y debido a un error

desconocido y fuera del alcance a los conocimientos e interés del curso impide que el objetivo original se realice.

Debido al limitado tiempo y total falta de conocimiento en este tema, se optó por la siguiente modificación al código.

```
void playSoundTrack(int nOST) {
	switch (nOST) {
	case 0:
		PlaySound(TEXT("Audio/All_Star.wav"), NULL, SND_SYNC);
		PlaySound(TEXT("Audio/Ambientacion.wav"), NULL, SND_LOOP || SND_NOSTOP);
		break;
	case 1:
		PlaySound(TEXT("Audio/Vieja_Muerta.wav"), NULL, SND_SYNC);
	case 2:
		PlaySound(TEXT("Audio/Ambientacion.wav"), NULL, SND_LOOP||SND_NOSTOP);
		break;
	}
}
```

Ahora se reproducirá el sonido de fondo una vez termine cualquier otro sonido de la animación. Si se quisiera hacer un efecto más inmerso, se debería editar los audios para concordar en sus sonidos de fondo para no notar el cambio.

Configuración de las animaciones

Las teclas de control, solo se añadieron F para la reproducción de las animaciones y R para su reinicio.

Se puede activar y desactivar el ciclo de día con las teclas T Y respectivamente

Se puede encender y apagar la luz cercana al letrero de “Cuidado con el Ogro” con G H respectivamente

Configuración de la cámara

Las teclas WASD corresponden al movimiento de la cámara en el plano XY y el mouse controla la dirección de la vista.

Para cambiar la vista, a las cámaras fijas use los números 1 2 3 4 del teclado alfabético.

1.- Cámara libre

2.- Cámara Fija #1

3.- Cámara Fija #2

4.- Cámara de personaje

Limitantes

Como un segundo proyecto, se sabían de antemano muchos obstáculos a superar y los paradigmas que se debían de seguir para evitar los errores que llevaron al fracaso anterior. La visión tuvo que limitarse a un escenario ya conocido, no se contaba con el tiempo ni la experiencia suficiente para realizar un entorno original, ni mucho menos excederse en el número de edificios o cualquier requisito adicional para una decoración interior. Tomando en cuenta las restricciones como Los Simpson o Kame House (Dragon Ball) se optó por el Pantano de Shrek, popula, bien conocido sin pedir demasiados requisitos.

Como sus elementos y animaciones para presentar, se optó por un enfoque de comedia pues es el mejor de los casos en la que modelos tan feos y movimientos poco detallados en comparación a los productos profesionales pueden ofrecer. El sistema de keyframes se optimizó para realizar animaciones complejas en poco tiempo y aun así estamos limitados por una producción de ensayo y error, pues no contamos con una herramienta con la que podamos obtener los valores exactos con el comportamiento real de los modelos a cada transformación.

Por otra parte, la biblioteca de audio utilizada solo ofrece la opción de reproducir el audio más no de una administración de pausa o cancelación de la rutina.

Así como existe un defecto desconocido en as luces de tipo Spotlight pues aunque se encuentren bien configuradas, estas no reproducen ningún tipo de luz pese a estar implementadas.

El toroide debió ser colocado de una manera fija aunque sus valores estén correctos se desconoce la causa exacta de este defecto aunque se especulase debido a la impresión de los datos de tipo float.

Conclusiones

El proyecto se ha construido con todas y cada una de las herramientas enseñadas a lo largo del curso, siendo un camino relativamente simple debido a la construcción gradual de este ambiente virtual, existieron inconveniencias menores como las cargas incorrectas de la textura, el uso de diversas shaders o dificultades tan triviales pero difíciles de ver como un signo menos o una asignación incorrecta en una variable.

El modelado tuvo que ser menos de un día de trabajo, pues debí controlar el mal hábito de querer la perfección en cada detalle, pues se requería de un trabajo integral siendo un único integrante para realizar todo.

Y así se conservó durante el proyecto de Teoría que este permitió y favoreció el trabajo en equipo, por circunstancias ajenas no se pudieron implementar los cambios de las primeras propuestas, cosas que no estaban se tuvieron que dar de un día para otro y aun así se consiguió.