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

Shrek’s Swamp Proyect

Introduction

Computer graphics is the field of visual computing, where computers are used both to synthetically generate visual images and to integrate or change proven visual and spatial information from the real world.

This field can be divided into several areas: Real-time 3D rendering (often used in video games), computer animation, video capture and creation of rendered video, special effects editing (often used for film and television), editing imaging, and modeling (often used for engineering and medical purposes). The development in computer graphics was first fueled by academic interests and government patronage. However, as the truly worldwide applications of computer graphics (CG) in television and film proved a viable alternative to more traditional special effects and animation techniques, commercials have increasingly financed the advancement of this field. .

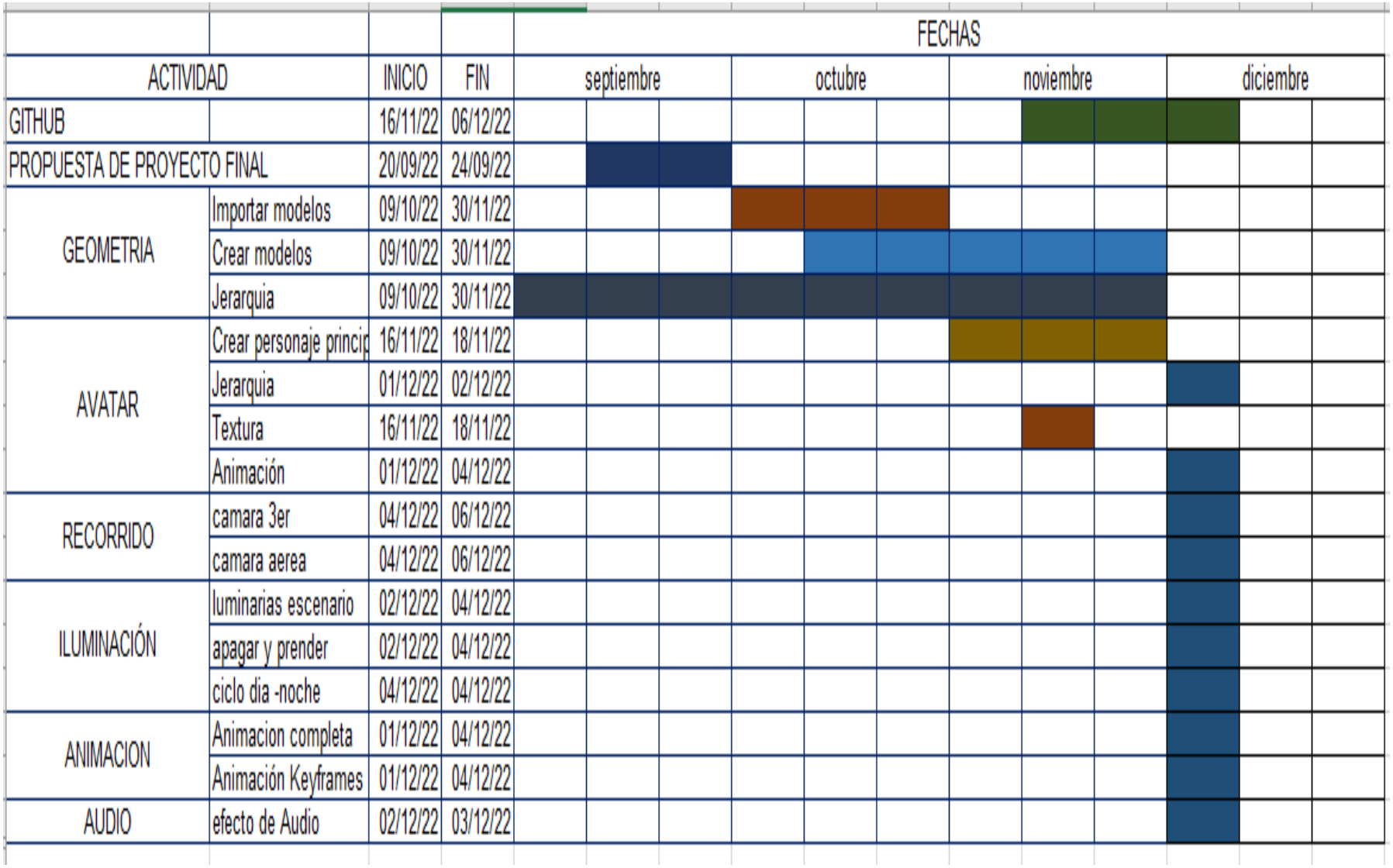
Objective

The student will combine the basic techniques seen in the computer graphics laboratory and human-computer interaction to reinforce what has been learned in the face-to-face sessions.

GuideLines

The student must select a façade and a space that can be real or fictitious and present reference images of said spaces for their 3D recreation in OpenGL. In the reference image, 7 objects must be visualized that the student is going to recreate virtually and where said objects must be the closest thing to their reference image, as well as their setting.

Diagramm 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

Proyect Reach

Reference Images

Shrek's House

Elements to make:

1. Table
2. Chair
3. Coach
4. Firepit
5. Coffin Glass
6. Outside restroom
7. Sight of Warning

Documentation

To begin with, the project was purged and optimized by removing all foreign material from the project, such as the folders Images, Models/Character and so on. As well as its inclusion and call in the code.

The shaders were renamed to leave a clearer understanding of what was being done in the project, since the name of lamp or lightning, although it was to indicate the interaction of light, this was referred to a standard model.

Many things were removed from the main loop, especially regarding the lighting configuration, since not all the focal points would be used or the parameters would be modified in each cycle.

Regarding characteristics of the program. Optimized the use of Keyframes as follows

```
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;
```

The structure has a floating vector of dynamic size to store the parameters that need to be animated, from 3 parameters for the 3 axes of a limb, 4 parameters (position vector and angle of the Y axis) for the manipulation of a model. All interior components.

```
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;
    }
```

The real administrative change is in the routine structure, since it contains all the necessary values that may be needed from the version provided by the laboratory. The advantages of this administration are the choice of the complexity of the KeyFrames, the number of these and which detail interpolation operations are performed, such as the possibility of repeating in a loop. (Although the system does not gradually transition between the last position and the first).

It contains a vector of Keyframes, a float vector for changes and one for the current value of the animation, as well as indicators of the tween and current frame, such as boolean play or cycle flags.

```
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++;
        }
}
```

The interpolation method is the same in terms of arithmetic, it was only modified to be cyclic.

SetAtZero is the formal restart of the animation, returning to the starting position without a transition, setting the flags to zero, and performing the interpolation specified by the original method.

As for the animation, the play control variable was kept so that it would stop executing while still in the main loop. Swapped logic to resolve a last frame playback issue, initial testing showed that the transition to the last frame was needed, so the current frame was ramped up early but before it was interpolated. If the animation had finished, it is asked if it is a loop to restart, otherwise it is prevented from executing this procedure by changing the play variable to false.

And the transition adds up. In a loop format.

```
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;
```

This is how the information is loaded, first we must define our structures with the dimension of the Frames, the number of these, the quality of the detail and optionally, a boolean that indicates if this will be repeated in a loop or not (By default it is false) and the loading process is similar to a two-dimensional array, where the value of each keyframe is queried.

At the end of each value declaration, the setAtCero method of each subroutine must be called to perform the first interpolation.

NOTE: This procedure is not automated, therefore, there is not something that regulates the number of memory accesses, so if access is requested to a number equal to or greater than the size of the complexity or the number of keyframes , this will give an error at the beginning of the program, although all those values that have an assignment will have 0.0f by default.

This is how the animation is handled (numerical values may vary, tweaks are still being made), you have a flag set with the “F” key turning True, the master animation will always run, it was decided to use the avatar positioning routine, specifically the frame in which you perform the action to trigger the other routines in the environment.

```
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();
    }
```

A sound library has been implemented, there is a function to run in parallel (OpenGL couldn't run otherwise) and a flag that only allows one execution at a time is used, in order to prevent conflicts with irrational thread handling. in the program.

```
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();
}
```

This procedure is made to reset the parameters to the initial value, there is no way to make this a general process due to the particular nature of each routine, such as some other details like the one_shot flags for the audio or lighting values.

WARNING: If the reset key is pressed and the scene audio continues to play, an overexposure effect may be created in the sound or the program may collapse. It is recommended to wait a couple of seconds after listening to absolutely nothing from the program .

```
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;
}
```

The degraded function is scheduled to go from a value A to another value B in a length of time. And stop (returning one of the values unchanged) when reaching the destination. These 2 functions are ideal for use in lighting, especially in campfire simulation.

```
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);
```

As we can see, we depend on a random value for the scope of the PointLight, to give a smooth finish to the effect, this range is made between 0.0125f as a minimum (maximum light) to a maximum of 0.1024f (minimum light). While the ambient light is subject to the events of the scene, it could perfectly well be subject to a routine like the other events, but it decided to do so because of its simplicity to implement.

```
////////////////////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));
```

Finally, its implementation in a hierarchical model in which the value of the corresponding subroutine is taken. On the other hand, the static elements such as the house, the sign or the bathroom are directly applied to the translation transformations to optimize resources. To aid that practice, all models were scaled and oriented on export.

```
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();
```

The first versions of the program tried to go beyond the established points, thinking that this object could contain a texture as well as be tested in external programs such as Blender or Maya by creating an .obj file, due to the lack of of time and too many technical difficulties in terms of the requirements of the construction of this object, it was decided to cut its scope to the simple generation of vertices and the indices that would lead to draw the object.

```
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);
```

In its structure we will use 4 methods.
The first is used to obtain the real number of qualities.

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

In such a way that, the minimum case that it can give us is a very thin rectangle in case of giving us something less than 1. Being a resounding error declaration by the parameter. Although in a functional case, our minimum would be a 4x4 quality figure, that is, 16 vertices in total and 30 triangles to draw. It was done this way to give a minimal figure to understand from the computer and the most consistent in terms of its circumference referent.

```
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;
	}
}
```

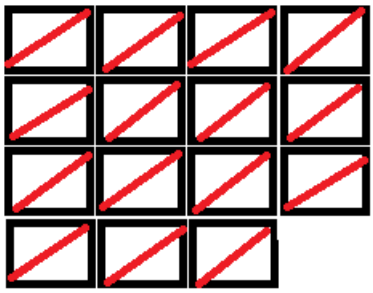
This function is designed to save processing costs at the time of creation, since the way the program works is to replicate a circumference in the XY plane through the different angles dictated by the exterior quality in the XZ plane.
Therefore it is very effective to calculate only once the Y value of the circle and assign the corresponding X and Z values to it later.

```
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;
		}
	}
}
```

The main function for the elaboration of the vertices initially calculates the jump constants, determined by the quality and the measurements of the radius as well as the calculation of the circle that makes up the profile of the toroid. In the first level, the scalar of its correspondence in the XZ plane is determined by means of sine and cosine. It will increase as the object is printed in the 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;
			}
}
```

On the other hand, the function that gives the indices does not depend on the value of the radii but on the quality of the object, since it unwraps the toroid in a mesh and, through the necessary calculations, determines the faces that must be placed.
If we see the point of view of a grid.
We can understand that if we follow the descending order and to the right, we understand that the first triangle of the point is made up of our "protagonist" point, that of the next column at its same height and its immediate. The other triangle that makes up its cell behaves in a similar way, it is the immediate one and its 2 neighbors.



Therefore, this algorithm is in charge of calculating it based on the quality that it represents, enclosing these values in an additional iteration that connects with the beginning as the toroid should do by its definition, leaving a free cell.

Summary:
To be implemented in the project, a Shader primitive is required that only captures the position data and interprets a constant color without alterations in the light sources.

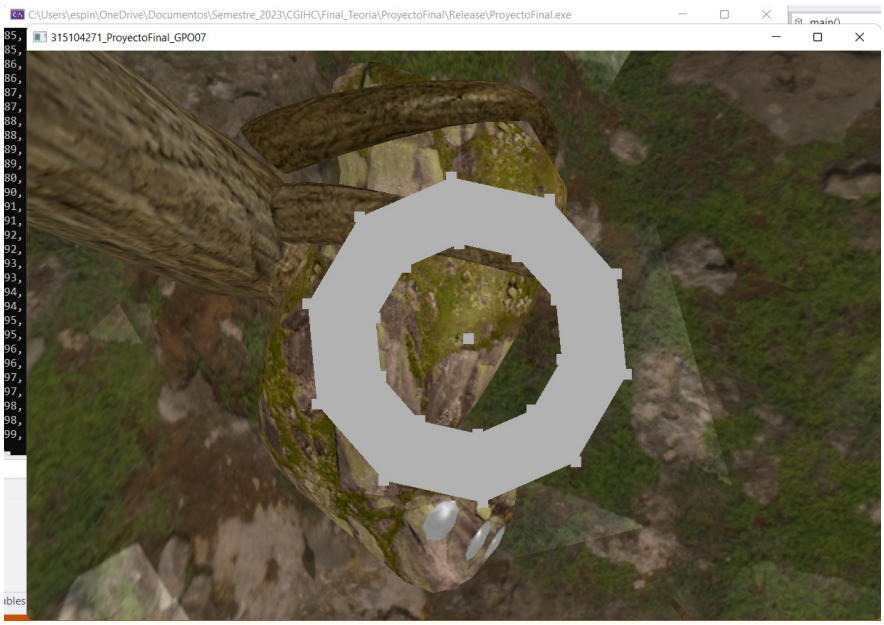
As well as re-preparing the Vertex Array Objects, Vertex BufferObjec and the ElementBufferObject with the size of the function.

```
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);
```

Nevertheless…

This is a screenshot of what happens, it illustrates both the points and the link of their faces. In an unknown way, the data even when the vector object is used to store the data, it is corrupted when it enters the data buffer.

Such that exactly the same values were implemented in static arrays and this is the result.



Therefore, it is concluded that the algorithm delivers correct values and due to an unknown error and outside the scope of the knowledge and interest of the course, it prevents the original objective from being achieved.

Due to the limited time and total lack of knowledge on this subject, the following modification to the code was chosen.

```
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;
	}
}
```

Background sound will now play after any other sound in the animation ends. If you want to make a more immersed effect, you should edit the audios to match their background sounds so you don't notice the change.

Animation settings

The control keys, only F for the playback of the animations and R for its restart were added.

The day cycle can be activated and deactivated with the T Y keys respectively

The light near the “Watch out for the Ogre” sign can be turned on and off with G H respectively.

camera settings
The WASD keys correspond to the movement of the camera in the XY plane and the mouse controls the direction of view.
To change the view, to fixed cameras use the numbers 1 2 3 4 on the alphabetic keyboard.

1.- Free camera
2.- Fixed Camera #1
3.- Fixed Camera #2
4.- Character camera

Limitations

As a second project, many obstacles to overcome and the paradigms that had to be followed to avoid the mistakes that led to the previous failure were known in advance. The vision had to be limited to an already known scenario, there was not enough time or experience to make an original environment, much less exceed the number of buildings or any additional requirements for interior decoration. Taking into account the restrictions such as The Simpsons or Kame House (Dragon Ball), the popular, well-known Shrek Swamp was chosen without asking for too many requirements.

As its elements and animations to present, a comedy approach was chosen as it is the best in which ugly models and poorly detailed movements can offer compared to professional products. The keyframe system was optimized to perform complex animations in a short time and even so we are limited by trial and error production, since we do not have a tool with which we can obtain the exact values with the real behavior of the models at each transformation. .

On the other hand, the audio library used only offers the option to play the audio but not to manage pause or cancel the routine.

Just as there is an unknown defect in Spotlight type lights because even if they are well configured, they do not reproduce any type of light despite being implemented.

The toroid must have been placed in a fixed way, although its values are correct, the exact cause of this defect is unknown, although it was speculated due to the printing of float type data.

Conclusions

The project has been built with each and every one of the tools taught throughout the course, being a relatively simple path due to the gradual construction of this virtual environment, there were minor inconveniences such as incorrect loading of the texture, the use of various shaders or difficulties as trivial but difficult to see as a minus sign or an incorrect assignment in a variable.

The modeling had to be less than a day's work, because I had to control the bad habit of wanting perfection in every detail, since it required a comprehensive job being a single member to do everything.

And so it was preserved during the Theory project that this allowed and favored teamwork, due to external circumstances the changes of the first proposals could not be implemented, things that were not there had to be given from one day to the next and even so got.