Universidad Nacional Autónoma de México

Facultad de Ingeniería

Laboratorio de Computación Gráfica e Interacción Humano Computadora

Profesor: Ing. José Roque Román Guadarrama

Proyecto Final

Manual técnico

Integrantes:

Brito Serrano Miguel Ángel

Muñoz García Arturo

Fecha:10/12/2021

Semestre 2022-2

Objetivo

- El alumno aplicara los conocimientos adquiridos durante todo el curso.
- El alumno deberá elaborar un proyecto en el que se demuestren los conceptos aprendidos a lo largo del curso.
- El alumno deberá seleccionar un entorno y recrearlo en 3D en OpenGL.

Enlace repositorio remoto GitHub

https://github.com/ArturoMu12018/ProyectoComputacion.git

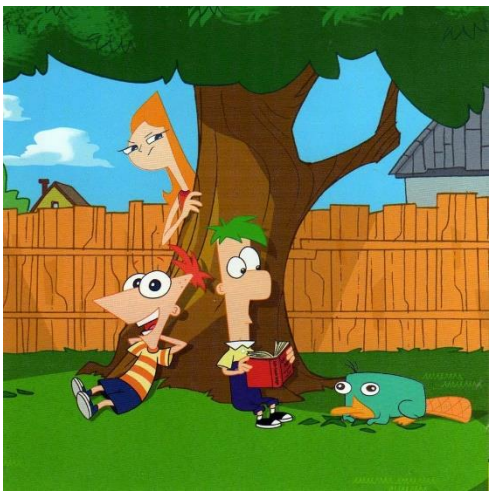
Alcance del Proyecto

Recrear un ambiente virtual con temática en el patio de Phineas y Ferb, en 3D OpenGl.

- Los elementos para recrear son:
- Phineas
- Ferb
- Candace
- Perry
- Santa Claus
- Árbol de Navidad
- Cerca de madera

Elementos extras y de ambientación son:

A continuación, se muestran imágenes de referencia:



Declaraicones de variables y inicialisacion inicial:

```
//Proyecto
float rotacionArbol;
float movXFerb;
float movZFerb;
float anguloYFerb;
float movXPhineas;
float movZPhineas;
float anguloYPhineas;

float movYFerb;
float anguloXFerb;
bool idaFerb;
float anguloZFerb;

float santaX;
float santaY;
float santaZ;
bool subidaSanta=true;
bool activaSanta = false;
float anguloYSanta = 0.0f;
float anguloXSanta;
float anguloZSanta;
```

```
//Animacion de Avatra
//Pie izquierdo
float rotacionPieIzq = 0.0f;
float posicionPieIzqX = 0.0f;
float posicionPieIzqY = 0.25f;
float timerPasosAvaatr = 0.0f;

//Pie derecho
float rotacionPieDer = 0.0f;
float posicionPieDerX = 0.0f;
float posicionPieDerY = 0.25f;

//Brazo Derecho
float rotacionBrazoDer = 0.0f;
float posicionBrazoDerX = 0.0f;
float posicionBrazoDerY = 0.25f;

//Brazo Izquierdo
float rotacionBrazoIz = 0.0f;
float posicionBrazoIzX = 0.0f;
float posicionBrazoIzY = 0.25f;
```

```
float tiempo1;
float tiempo2;

float rotaCamraRobot;

float movYCandase;
float movZCandase;
float anguloYCandase;
float contadorVueltasCandase;

float alturaArbol;
bool subidaArbolNavidad;
bool activaArbol=false;
```

```
//Avatar
float rotacionAvatraDerecha;
float rotacionAvatarIzquierda;
float posAvatarX;
float posAvatarZ;
int tipoCamara=0; //si es 0 camara global, si es 1 en 3ra persona y  si es 2 es camara global
```

```
//iluminaicon
bool dia = true;
bool lucesPorTeclado = false;
bool encendidasApagadas = true;
bool enciendeLucesShow = false;
bool showActivacion = false;
float timerShow = 0.0f;
float posLuz3X = 0.0f;
float posLuz3Y = 0.0f;
float posLuz3Z = 0.0f;
float posLuz4X = 0.0f;
float posLuz4Y = 0.0f;
float posLuz4Z = 0.0f;
float posLuz5X = 0.0f;
float posLuz5Y = 0.0f;
float posLuz5Z = 0.0f;
float posLuz6X = 0.0f;
float posLuz6Y = 0.0f;
float posLuz6Z = 0.0f;
```

```
//camara
float camaraAereaX;
float camaraAereaZ;
float camaraLibreX;
float camaraLibreY;
float camaraLibreZ;
```

```
//----Proyecto
Model Arbol;
Model Candace;
Model Santa_M;
Model Ferb_M;
Model Perry;
Model Phienas;
Model ArbolNavidad_M;

//Avatar
Model CabezaRobot_M;
Model BrazoDerechoRobot_M;
Model BrazoIzquierdoRobot_M;
Model CuerpoRobot_M;
Model PieDerechoRobot_M;
Model PieIzquierdoRobot_M;
```

Modelos creados en OpenGl:

Valla:

Creamos los indices y vertices. En los vertices pusimos la posicion de la textura

```
void CrearPrismaPoligonal()
{
    unsigned int prisma_indices[] = {
        // front
        0, 1, 2,
        2, 3, 0,
        2, 3, 4,

        //right bottom
        5,6,7,
        7,8,5,

        //right up
        9,10,11,
        11,12,9,

        //left bottom
        13,14,15,
        15,16,13,

        //left up
        17,18,19,
        19,20,17,
```

```
    //left up
    17,18,19,
    19,20,17,

    //button
    21,22,23,
    23,24,21,

    //back
    25,26,27,
    27,28,25,
    27,28,29
```

```
GLfloat prisma_vertices[] = {

    // front
    //x     y       z       S       T           NX      NY      NZ
    -0.0f, -0.0f,  0.0f,    0.01f,  0.01f,      0.0f,   0.0f,   -1.0f,  //0
    4.5f, -0.0f,  0.0f,     0.3f,   0.01f,      0.0f,   0.0f,   -1.0f,  //1
    4.5f,  7.5f,  0.0f,     0.3f,   0.5f,       0.0f,   0.0f,   -1.0f,  //2
    -0.0f,  7.5f,  0.0f,    0.01f,  0.5,        0.0f,   0.0f,   -1.0f,  //3
    2.25f,  9.5f,  0.0f,    0.15f,  0.7f,       0.0f,   0.0f,   -1.0f,  //4

    //right bottom
    //x     y       z       S       T           NX      NY      NZ
    4.5f, -0.0f,  0.0f,     0.31f,  0.01f,      0.0f,   0.0f,   -1.0f,  //5
    4.5f, -0.0f,  -1.0f,    0.4f,   0.01f,      0.0f,   0.0f,   -1.0f,  //6
    4.5f,  7.5f,  -1.0f,    0.4f,   0.5f,       0.0f,   0.0f,   -1.0f,  //7
    4.5f,  7.5f,  0.0f,     0.31f,  0.5f,       0.0f,   0.0f,   -1.0f,  //8

    //right up
    //x     y       z       S       T           NX      NY      NZ
    4.5f,  7.5f,  -1.0f,    0.21f,  0.51f,      0.0f,   0.0f,   -1.0f,  //9
    2.25f,  9.5f,  -1.0f,   0.4f,   0.7f,       0.0f,   0.0f,   -1.0f,  //10
    2.25f,  9.5f,  0.0f,    0.21f,  0.7f,       0.0f,   0.0f,   -1.0f,  //11
    4.5f,  7.5f,  0.0f,     0.4f,   0.51f,      0.0f,   0.0f,   -1.0f,  //12
```

```
//left bottom
//x     y       z       S       T           NX      NY      NZ
-0.0f, -0.0f,  0.0f,    0.71f,  0.01f,      0.0f,   0.0f,   -1.0f,  //13
-0.0f, -0.0f,  -1.0f,   0.8f,  0.01f,      0.0f,   0.0f,   -1.0f,  //14
-0.0f,  7.5f,  -1.0f,   0.8f,   0.5f,      0.0f,   0.0f,   -1.0f,  //15
-0.0f,  7.5f,  0.0f,    0.71,   0.5f,      0.0f,   0.0f,   -1.0f,  //16

//left up
//x     y       z       S       T           NX      NY      NZ
-0.0f,  7.5f,  0.0f,    0.1f,   0.6f,      0.0f,   0.0f,   -1.0f,  //17
2.25f,  9.5f,  0.0f,    0.1f,   0.8f,      0.0f,   0.0f,   -1.0f,  //18
2.25f,  9.5f,  -1.0f,   0.01f,  0.8f,      0.0f,   0.0f,   -1.0f,  //19
-0.0f,  7.5f,  -1.0f,   0.01f,  0.6f,      0.0f,   0.0f,   -1.0f,  //20

//botton
-0.0f, -0.0f,  0.0f,    0.27f,  0.35f,      0.0f,   0.0f,   -1.0f,  //21
4.5f, -0.0f,  0.0f,     0.48f,  0.35f,      0.0f,   0.0f,   -1.0f,  //22
4.5f, -0.0f,  -1.0f,    0.48f,  0.35f,      0.0f,   0.0f,   -1.0f,  //23
-0.0f, -0.0f,  -1.0f,   0.27f,  0.35f,      0.0f,   0.0f,   -1.0f,  //24

//back
//x     y       z       S       T           NX      NY      NZ
-0.0f, -0.0f,  -1.0f,   0.7f,  0.01f,      0.0f,   0.0f,   -1.0f,  //25
4.5f, -0.0f,  -1.0f,    0.41f,  0.01f,      0.0f,   0.0f,   -1.0f,  //26
4.5f,  7.5f,  -1.0f,    0.41f,  0.5f,       0.0f,   0.0f,   -1.0f,  //27
-0.0f,  7.5f,  -1.0f,   0.7f,   0.5f,       0.0f,   0.0f,   -1.0f,  //28
2.25f,  9.5f,  -1.0f,   0.55f,  0.7f,       0.0f,   0.0f,   -1.0f,  //29
```

```
Mesh* prisma = new Mesh();
prisma->CreateMesh(prisma_vertices, prisma_indices, 240, 48);
meshList.push_back(prisma);
```

Declaracion de textura

```
Texture Madera;
```

```
dadoTexture.LoadTextureA();
Madera= Texture("Textures/madera.tga");
Madera.LoadTextureA();
alfombra = Texture("Textures/alfombraRoja
```

Declaracion del modelo, Lo metimos a un for para que hubiera muchos de ellos entre -45 y 50

```
for (int x = -45; x <= 50; x += 4) {
    model = glm::mat4(1.0);
    model = glm::translate(model, glm::vec3(x, -2.0f, -1.0f));
    model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
    Madera.UseTexture();
    meshList[8]->RenderMesh();
    }
```

Base Prisma triangular:

Creamos los indices y vertices. En los vertices pusimos la posicion de la textura

```
void CreaPrismaTrian()
{
    unsigned int prismaTrian_indices[] = {
        // front
        0, 1, 2,

        //Back
        3,4,5,

        //right
        6,7,8,
        8,9,6,

        //left
        10,11,12,
        12,13,10,

        //botton
        14,15,16,
        16,17,14,
```

```
GLfloat prismaTrian_vertices[] = {

    // front
    //x     y       z       S       T           NX      NY      NZ
    -0.0f, -0.0f,  0.0f,    0.371f,  0.24f,     0.0f,   0.0f,   -1.0f,  //0
    4.0f, -0.0f,  0.0f,     0.661f, 0.24f,      0.0f,   0.0f,   -1.0f,  //1
    2.0f,  3.0f,  0.0f,     0.532f, 0.0f,       0.0f,   0.0f,   -1.0f,  //2

    // back
    //x     y       z       S       T           NX      NY      NZ
    -0.0f, -0.0f,  -5.0f,   0.38f,  0.723f,     0.0f,   0.0f,   -1.0f,  //3
    4.0f, -0.0f,  -5.0f,    0.66f,  0.723f,     0.0f,   0.0f,   -1.0f,  //4
    2.0f,  3.0f, -5.0f,     0.525f, 1.0f,       0.0f,   0.0f,   -1.0f,  //5

    //right
    4.0f, -0.0f,  0.0f,     0.661f, 0.25f,      0.0f,   0.0f,   -1.0f,  //6
    4.0f, -0.0f,  -5.0f,    0.661f, 0.723f,     0.0f,   0.0f,   -1.0f,  //7
    2.0f,  3.0f, -5.0f,     0.95f,  0.723f,     0.0f,   0.0f,   -1.0f,  //8
    2.0f,  3.0f,  0.0f,     0.95f,  0.25f,      0.0f,   0.0f,   -1.0f,  //9

    //left
    -0.0f, -0.0f,  0.0f,    0.37f,  0.25f,      0.0f,   0.0f,   -1.0f,  //10
    -0.0f, -0.0f,  -5.0f,   0.37f,  0.723f,     0.0f,   0.0f,   -1.0f,  //11
    2.0f,  3.0f, -5.0f,     0.1f,   0.723f,     0.0f,   0.0f,   -1.0f,  //12
    2.0f,  3.0f,  0.0f,     0.1f,   0.25f,      0.0f,   0.0f,   -1.0f,  //13
```

```
    //botton
    -0.0f, -0.0f,  0.0f,    0.371f,  0.25f,     0.0f,   0.0f,   -1.0f,  //14
    -0.0f, -0.0f,  -5.0f,   0.371f,  0.723f,        0.0f,   0.0f,   -1.0f,  //15
    4.0f, -0.0f,  -5.0f,    0.66f,  0.723f,     0.0f,   0.0f,   -1.0f,//16
    4.0f, -0.0f,  0.0f,     0.66f,  0.25f,      0.0f,   0.0f,   -1.0f,  //17

};

Mesh* prismaTrian = new Mesh();
prismaTrian->CreateMesh(prismaTrian_vertices, prismaTrian_indices, 144, 24);
meshList.push_back(prismaTrian);
```

Textura de alfombra

```
Texture Madera;
Texture alfombra;
```

```
Madera.LoadTextureA();
alfombra = Texture("Textures/alfombraRoja.tga");
alfombra.LoadTextureA();
```

Declaramos una gerarquia con el arbol de navidad para que suban y bajen al mismo tiempo. Usamos modelauxArbolNav para la gerarquia

```
glm::mat4 modelauxArbolNav(1.0);//matriz auxiliar para jerarquia de helicoptero


model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-20.0f, alturaArbol-2.0f, 5.0f));
modelauxArbolNav = model;
model = glm::scale(model, glm::vec3(0.07f, 0.07f, 0.07f));
model = glm::rotate(model, rotacionArbol * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
ArbolNavidad_M.RenderModel();


//Base arbol
model = modelauxArbolNav;
model = glm::translate(model, glm::vec3(4.1f, 0.0f, -2.0f));
model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, -90 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
alfombra.UseTexture();
meshList[9]->RenderMesh();
```

Carga de modelos de blender:

Cragamos los modelos los inicial nombre

```
//------Proyecto
Arbol = Model();
Arbol.LoadModel("Models/arbol.obj");
Candace = Model();
Candace.LoadModel("Models/candace.obj");

Santa_M = Model();
Santa_M.LoadModel("Models/santa.obj");
Ferb_M = Model();
Ferb_M.LoadModel("Models/ferbo.obj");
Perry = Model();
Perry.LoadModel("Models/perry.obj");
Phienas = Model();
Phienas.LoadModel("Models/phineas.obj");
ArbolNavidad_M = Model();
ArbolNavidad_M.LoadModel("Models/arbolNavidad.obj");


CabezaRobot_M = Model();
CabezaRobot_M.LoadModel("Models/Cabeza_Robot.obj");
BrazoDerechoRobot_M = Model();
BrazoDerechoRobot_M.LoadModel("Models/BrazoDerecho_Robot.obj");
BrazoIzquierdoRobot_M = Model();
BrazoIzquierdoRobot_M.LoadModel("Models/BrazoIzquierdo_Robot.obj");
CuerpoRobot_M = Model();
CuerpoRobot_M.LoadModel("Models/BrazoCuerpo_Robot.obj");
PieDerechoRobot_M = Model();
PieDerechoRobot_M.LoadModel("Models/PieDercho_Robot.obj");
PieIzquierdoRobot_M = Model();
PieIzquierdoRobot_M.LoadModel("Models/PieIzquierdo_Robot.obj");
```

```
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(15.0f, -2.0f, 5.0f));
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
Arbol.RenderModel();
```

```
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-1.0f, movYCandase+1.0f, movZCandase +10.0f));
model = glm::scale(model, glm::vec3(2.9f, 2.9f, 2.9f));
model = glm::rotate(model, anguloYCandase * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
Candace.RenderModel();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(santaX-10.0f, santaY-2.0f, santaZ+5.0f));
model = glm::scale(model, glm::vec3(3.0f, 3.0f, 3.0f));
model = glm::rotate(model, anguloYSanta * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, anguloXSanta * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, anguloZSanta * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
Santa_M.RenderModel();
```

```
model = glm::mat4(1.0);

model = glm::translate(model, glm::vec3(movXFerb + 7.0f, movYFerb -2.0f, movZFerb));
model = glm::scale(model, glm::vec3(0.23f, 0.23f, 0.23f));
model = glm::rotate(model, anguloYFerb * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, anguloXFerb * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, anguloZFerb * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
Ferb_M.RenderModel();



model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(movXPerry -5.0f, moYPerry -2.0f, moZPerry +15.0f));
model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));
model = glm::rotate(model, rotacionXPerry * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, rotacionYPerry * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, rotacionZPerry * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
Perry.RenderModel();


model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(movXPhineas + 10.0f, -2.0f, movZPhineas + 0.0f));
model = glm::scale(model, glm::vec3(0.9f, 0.9f, 0.9f));
model = glm::rotate(model, anguloYPhineas * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
Phienas.RenderModel();
```

```
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-20.0f, alturaArbol-2.0f, 5.0f));
model = glm::scale(model, glm::vec3(0.07f, 0.07f, 0.07f));
model = glm::rotate(model, rotacionArbol * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
ArbolNavidad_M.RenderModel();




//Avatar--Robot

//Cabeza cuerpo
glm::mat4 modelauxRobot(1.0);//matriz auxiliar para jerarquia de helicoptero


model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(posAvatarX +5.0f, -3.2f, posAvatarZ +0.0f));
model = glm::scale(model, glm::vec3(5.0f, 5.0f, 5.0f));
model = glm::rotate(model, rotaCamraRobot + 180 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
modelauxRobot = model;
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
CabezaRobot_M.RenderModel();
```

```
//Brazo derecho
model = modelauxRobot;
model = glm::translate(model, glm::vec3(0.0f, 0.25f, 0.0f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
//model = glm::rotate(model, mainWindow.getrotahelice() * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
//model = glm::rotate(model, rotacionAvatraDerecha * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
BrazoDerechoRobot_M.RenderModel();



//Brazo Izquierdp
model = modelauxRobot;
model = glm::translate(model, glm::vec3(0.0f, 0.25f, 0.0f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
//model = glm::rotate(model, mainWindow.getrotahelice() * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
//model = glm::rotate(model, rotacionAvatarIzquierda+180 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
BrazoIzquierdoRobot_M.RenderModel();


//Cuerpo
model = modelauxRobot;
model = glm::translate(model, glm::vec3(0.0f, 0.25f, 0.0f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
//model = glm::rotate(model, mainWindow.getrotahelice() * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
//model = glm::rotate(model, rotahelice * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
CuerpoRobot_M.RenderModel();
```

```
//Cuerpo
model = modelauxRobot;
model = glm::translate(model, glm::vec3(0.0f, 0.25f, 0.0f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
//model = glm::rotate(model, mainWindow.getrotahelice() * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
//model = glm::rotate(model, rotahelice * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
CuerpoRobot_M.RenderModel();


//Pie derecho
model = modelauxRobot;
model = glm::translate(model, glm::vec3(0.0f, 0.25f, 0.0f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
//model = glm::rotate(model, mainWindow.getrotahelice() * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
//model = glm::rotate(model, rotacionAvatraDerecha * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
PieDerechoRobot_M.RenderModel();


//Pie Izquierdo
model = modelauxRobot;
model = glm::translate(model, glm::vec3(0.0f, 0.25f, 0.0f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
//model = glm::rotate(model, mainWindow.getrotahelice() * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
//model = glm::rotate(model, rotacionAvatarIzquierda * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
PieIzquierdoRobot_M.RenderModel();
```

Nota: Los modelos se encuentran en /Models/ y sus texturas en /Texture/

Inicializacion de skybox

```
std::vector<std::string> skyboxFaces;

//patio phienas y ferb dia
skyboxFaces.push_back("Textures/Skybox/PatioIzq.jpg");
skyboxFaces.push_back("Textures/Skybox/patioDer.jpg");
skyboxFaces.push_back("Textures/Skybox/cupertin-lake_dn.tga");
skyboxFaces.push_back("Textures/Skybox/cielo.jpg");
skyboxFaces.push_back("Textures/Skybox/patioAtras.jpg");
skyboxFaces.push_back("Textures/Skybox/patioAdel.jpg");

skybox = Skybox(skyboxFaces);
```

Carga skybox dia y noche:

Declaramos una variable llamada “tiempo1” esta es nuestra bandera que nos ayuda para que haga el cambio de dia y noche. Cuando llega al tiempo entra a camvioSybox() y reinicia el “tiempo1”.

```
tiempo1 += deltaTime;
if (tiempo1 > 30) {
    camvioSkybox();
    tiempo1 = 0;
}
```

Tenemos una variable bool que inicial mente es true y esta cambia cuando se invoca la funcion al caso contrario de la actual y hace el cambio de skybox.

```
void camvioSkybox(){


    if (dia==false) {
        std::vector<std::string> skyboxFaces;
        skyboxFaces.push_back("Textures/Skybox/PatioIzq.jpg");
        skyboxFaces.push_back("Textures/Skybox/patioDer.jpg");
        skyboxFaces.push_back("Textures/Skybox/cupertin-lake_dn.tga");
        skyboxFaces.push_back("Textures/Skybox/cielo.jpg");
        skyboxFaces.push_back("Textures/Skybox/patioAtras.jpg");
        skyboxFaces.push_back("Textures/Skybox/patioAdel.jpg");
        skybox = Skybox(skyboxFaces);
        dia = true;
        return;
    }
    else if(dia) {
        std::vector<std::string> skyboxFaces;
        skyboxFaces.push_back("Textures/Skybox/PatioIzq_noche.jpg");
        skyboxFaces.push_back("Textures/Skybox/patioDer_noche.jpg");
        skyboxFaces.push_back("Textures/Skybox/cupertin-lake_dn.tga");
        skyboxFaces.push_back("Textures/Skybox/cielo_noche.jpg");
        skyboxFaces.push_back("Textures/Skybox/patioAtras_noche.jpg");
        skyboxFaces.push_back("Textures/Skybox/patioAdel_noche.jpg");
        skybox = Skybox(skyboxFaces);
        dia = false;

    }
    return;


}
```

Nota: Las imágenes se encuentran en la carpeta /Texture/Skybox

Camaras:

Son 3 camaras

0-libre boton C

1-avatar boton Z

2-aerea boton X

En camera.cpp modificamos los controles para cada camara. En la camara libre lo dejamos por defecto, la camara de avatar dejamos el mismo movimiento de teclado pero modificamos que el mouse solo se pueda mover en x limitado a 20 y quitando el Chanel y. Para la aerea modificamos que el teclado no vaya a adelante lo pusimos para que se recorra en eje Y y bloqueamos los movimientos mouse.

```
void Camera::keyControl(bool* keys, GLfloat deltaTime,int tipo)//Sirve para cambiar direccion
{

    if (tipo==0 || tipo == 1) {
        GLfloat velocity = moveSpeed * deltaTime;

        if (keys[GLFW_KEY_W])
        {
            position += front * velocity;
        }

        if (keys[GLFW_KEY_S])
        {
            position -= front * velocity;
        }

        if (keys[GLFW_KEY_A])
        {
            position -= right * velocity;
        }

        if (keys[GLFW_KEY_D])
        {
            position += right * velocity;
        }
    }
```

```
}
else if (tipo == 2) {
    GLfloat velocity = moveSpeed * deltaTime;

    if (keys[GLFW_KEY_W])
    {
        position += up * velocity;
    }

    if (keys[GLFW_KEY_S])
    {
        position -= up * velocity;
    }

    if (keys[GLFW_KEY_A])
    {
        position -= right * velocity;
    }

    if (keys[GLFW_KEY_D])
    {
        position += right * velocity;
    }
}
```

```
void Camera::mouseControl(GLfloat xChange, GLfloat yChange, int
{
    if (tipo==0) {
        xChange *= turnSpeed;
        yChange *= turnSpeed;

        yaw += xChange;
        pitch += yChange;

        if (pitch > 89.0f)
        {
            pitch = 89.0f;
        }

        if (pitch < -89.0f)
        {
            pitch = -89.0f;
        }

        update();
    }
```

```
if (tipo == 1) {
    xChange *= turnSpeed;
    //yChange *= turnSpeed;

    yaw += xChange;
    //pitch += yChange;

    if (yaw > 20.0f)
    {
        yaw = 20.0f;
    }

    if (yaw < -20.0f)
    {
        yaw = -20.0f;
    }

    update();
}
```

```
if (tipo==2) {
    //xChange *= turnSpeed;
    //yChange *= turnSpeed;

    //yaw += xChange;
    //pitch += yChange;

    if (pitch > 89.0f)
    {
        pitch = 89.0f;
    }

    if (pitch < -89.0f)
    {
        pitch = -89.0f;
    }

    update();
}
```

Cuando se cambia de camara suena el audio de bleep.mp3. Para las camaras le creamos variables para que se guarde la ubicación donde se quedaron. Para el avatar se usa posAvatarX y posAvatarZ ahí se guardan la ubicación del avatar, cameraAereaX y cameraAereaZ para la ubicación de la camara aerea y camaraLibreX, camaraLibreY y camaraLibreX para la ubicación de la camara libre.

```
//Camara Avatar
if (keys[GLFW_KEY_Z]) {
    camera = Camera(glm::vec3(posAvatarX-10.0f, 5.0f, posAvatarZ), glm::vec3(0.0f, 1.0f, 0.0f), 0.0f, 0.0f, 7.0f, 0.5f);
    tipoCamara=1;
    SoundEngine->play2D("audio/bleep.mp3", false);
}

//Camara Aerea
else if (keys[GLFW_KEY_X]) {
    camera = Camera(glm::vec3(camaraAereaX, 22.0f, camaraAereaZ), glm::vec3(0.0f, 1.0f, 0.0f), 0.0f, -90.0f, 7.0f, 0.5f);
    tipoCamara = 2;
    SoundEngine->play2D("audio/bleep.mp3", false);
}


//Camara libre
else if (keys[GLFW_KEY_C] ) {
    camera = Camera(glm::vec3(camaraLibreX, camaraLibreY, camaraLibreZ), glm::vec3(0.0f, 1.5f, 0.0f), -60.0f, 0.0f, 7.0f, 0.5f);
    tipoCamara = 0;
    SoundEngine->play2D("audio/bleep.mp3", false);
}
```

Declaramos un if para que el avatar rotara hacia la direccion que va. La funcion avatarPasos sirve para la animacion de extremidades del avatar y avaatarParado aniacion cuando esta parado.

```
//Giro cuerpo
if (keys[GLFW_KEY_D] || keys[GLFW_KEY_A] || keys[GLFW_KEY_S] || keys[GLFW_KEY_W]) {
    if (keys[GLFW_KEY_W]) {
        avatarPasos();
    }
    if (keys[GLFW_KEY_D]) {
        rotaCamraRobot = 80.0f;
        avatarPasos();

    }
    if (keys[GLFW_KEY_A]) {
        rotaCamraRobot = -80.0f;
        avatarPasos();
    }
    if (keys[GLFW_KEY_S]) {
        rotaCamraRobot = -160.0f;
        avatarPasos();
    }
    if (keys[GLFW_KEY_A] && keys[GLFW_KEY_W]) {
        rotaCamraRobot = 240.0f;
        //avatarPasos();
    }
    if (keys[GLFW_KEY_D] && keys[GLFW_KEY_W]) {
        rotaCamraRobot = -240.0f;
        //avatarPasos();
    }
}
```

```
    }
}
else {
    rotaCamraRobot = 0.0f;

    avaatarParado();
}
```

Animacion Avatar

La animacion de caminar tiene 3 estados, cuando esta parado y 2 de cuando levanta sus extremidades.

El timerPasosAvaatr sirve para llevar el tiempo de los cambios de estado. Cada extremidad tiene su ratacion en Z y la ubicación de la nueva posicion ya que al rotar se mueve la extremidad de lado.

Con los pies encontramos posicionPieIzqY, posicionPieIzqX, rotacionPieIzq nos srive para controlar el pie izquierdo, posicionPieDerY, posicionPieDerX, rotacionPieDer nos srive para controlar el pie derecho, rotacionBrazoIz, posicionBrazoIzX, posicionBrazoIzY sirve para manipular la extremidad del brazo izquierdo y rotacionBrazoDer, posicionBrazoDerX, posicionBrazoDerY sirve para manipular la extremidad del brazo derecho.

```
void avatarPasos() {

    timerPasosAvaatr += deltaTime * 6;

    if (timerPasosAvaatr < 1) {
        //Pie izquierdo
        posicionPieIzqY = 0.25f;
        posicionPieIzqX = 0.0f;
        rotacionPieIzq = 0;

        //Pie derecho
         rotacionPieDer = 0.0f;
         posicionPieDerX = 0.0f;
         posicionPieDerY = 0.25f;


         //Brazo Izquierdo
         rotacionBrazoIz = 0.0f;
         posicionBrazoIzX = 0.0f;
         posicionBrazoIzY = 0.25f;


         //Brazo Derecho
         rotacionBrazoDer = 0.0f;
         posicionBrazoDerX = 0.0f;
         posicionBrazoDerY = 0.25f;
```

```
if (timerPasosAvaatr < 2 && timerPasosAvaatr>1) {
    posicionPieIzqY = 0.41f;
    posicionPieIzqX = 0.43f;
    rotacionPieIzq = 45;

    posicionPieDerX = -0.415f;
    posicionPieDerY = 0.44f;
    rotacionPieDer = -45;

    //Brazo Izquierdo
    rotacionBrazoIz = -45;
    posicionBrazoIzX = -0.662f;
    posicionBrazoIzY = 0.52f;

    //Brazo Derecho
    rotacionBrazoDer = 45;
    posicionBrazoDerX = 0.662f;
    posicionBrazoDerY = 0.51f;
```

```
if (timerPasosAvaatr < 3 && timerPasosAvaatr>2) {
    posicionPieIzqY = 0.25f;
    posicionPieIzqX = 0.0f;
    rotacionPieIzq = 0;

    posicionPieDerY = 0.25f;
    posicionPieDerX = 0.0f;
    rotacionPieDer = 0;

    //Brazo Izquierdo
    rotacionBrazoIz = 0.0f;
    posicionBrazoIzX = 0.0f;
    posicionBrazoIzY = 0.25f;

    //Brazo Derecho
    rotacionBrazoDer = 0.0f;
    posicionBrazoDerX = 0.0f;
    posicionBrazoDerY = 0.25f;
}
```

```
if (timerPasosAvaatr < 4 && timerPasosAvaatr>3) {

    posicionPieIzqX = -0.415f;
    posicionPieIzqY = 0.44f;
    rotacionPieIzq = -45;


    posicionPieDerY = 0.41f;
    posicionPieDerX = 0.43f;
    rotacionPieDer = 45;


    //Brazo Izquierdo
    rotacionBrazoIz = 45;
    posicionBrazoIzX = 0.662f;
    posicionBrazoIzY = 0.51f;

    //Brazo Derecho
    rotacionBrazoDer = -45;
    posicionBrazoDerX = -0.662f;
    posicionBrazoDerY = 0.52f;
}
```

```
if (timerPasosAvaatr > 4) {
    timerPasosAvaatr = 0.0f;
}
```

Iluminación:

Tenemos dos variables “encendidasApagadas”, “lucesPorTeclado” y dos funciones “showIluminacion”y “iluminacionShow”. Para “encendidasApagadas” viene por defecto true que hace que estén prendidas para apagarlas debe ser false y no respeta el cambio del skybox. “Para lucesPorTeclado” es el que da que false es el que da que se respete el ciclo del skybox y se iluminan y en false se toma la configuración de “encendidasApagadas”. Para “showIluminacion” actica las luces que se usan en el show de iluminación por defecto vienen apagadas y “iluminacionShow” activa el show. Las luces del show son SpotLight que son de la 3 a la 6. Para aumentar nuestro numero de luces hay que modificar MAX_SPOT_LIGHTS en CommonValues.h y en shader_light.frag MAX_SPOT_LIGHTS también.

```
out vec4 color;

const int MAX_POINT_LIGHTS = 8;
const int MAX_SPOT_LIGHTS = 8;
```

```
void iluminacion(bool* keys){
    if (keys[GLFW_KEY_T] && encendidasApagadas) {
        encendidasApagadas = false;

    }
    else if (keys[GLFW_KEY_T] && encendidasApagadas == false) {
        encendidasApagadas = true;

    }

    else if (keys[GLFW_KEY_R] && lucesPorTeclado == false) {
        lucesPorTeclado = true;


    }
    else if (keys[GLFW_KEY_R] && lucesPorTeclado) {
        lucesPorTeclado = false;

    }
```

```
spotLights[5] = SpotLight(1.0f, 0.0f, 0.0f,
    0.0f, 2.0f,
    0.0f, 0.0f, 0.0f,
    0.0f, -1.0f, 0.0f,
    1.0f, 0.0f, 0.0f,
    20.0f);
spotLightCount++;

spotLights[6] = SpotLight(1.0f, 1.0f, 1.0f,
    0.0f, 2.0f,
    0.0f, 0.0f, 0.0f,
    0.0f, -1.0f, 0.0f,
    1.0f, 0.0f, 0.0f,
    20.0f);
spotLightCount++;
```

```
//Iluminacion para el show de luces

spotLights[3] = SpotLight(0.0f,0.0f, 1.0f,
    0.0f, 2.0f,
    0.0f, 0.0f, 0.0f,
    0.0f, -1.0f, 0.0f,
    1.0f, 0.0f, 0.0f,
    20.0f);
spotLightCount++;


spotLights[4] = SpotLight(0.0f, 1.0f, 0.0f,
    0.0f, 2.0f,
    0.0f, 0.0f, 0.0f,
    0.0f, -1.0f, 0.0f,
    1.0f, 0.0f, 0.0f,
    20.0f);
spotLightCount++;
```

```
//iluminacion para show
void iluminacionShow(bool* keys){
    if (keys[GLFW_KEY_Y] && enciendeLucesShow == false) {
        enciendeLucesShow = true;
    }
    else if (keys[GLFW_KEY_Y] && enciendeLucesShow) {
        enciendeLucesShow = false;
    }

    if ( enciendeLucesShow) {
        spotLights[3].SetFlash(glm::vec3(-5.0f, 10.0f, 5.0f), glm::vec3(0.0, -1.0f, 0.0f));
        spotLights[4].SetFlash(glm::vec3(-5.0f, 10.0f, 10.0f), glm::vec3(0.0, -1.0f, 0.0f));
        spotLights[5].SetFlash(glm::vec3(5.0f, 10.0f, 5.0f), glm::vec3(0.0, -1.0f, 0.0f));
        spotLights[6].SetFlash(glm::vec3(5.0f, 10.0f, 10.0f), glm::vec3(0.0, -1.0f, 0.0f));
    }
    else if (enciendeLucesShow == false) {
        spotLights[3].SetFlash(glm::vec3(0.0f, 10.0f, 15.0f), glm::vec3(0.0, 0.0f, 0.0f));
        spotLights[4].SetFlash(glm::vec3(0.0f, 10.0f, 10.0f), glm::vec3(0.0, 0.0f, 0.0f));
        spotLights[5].SetFlash(glm::vec3(0.0f, 10.0f, 5.0f), glm::vec3(0.0, 0.0f, 0.0f));
        spotLights[6].SetFlash(glm::vec3(0.0f, 10.0f, 5.0f), glm::vec3(0.0, 0.0f, 0.0f));
    }

}
```

Implementar 5 animaciones, dos complejas y 3 basicas:

Animación básica:

Animación árbol

“rotacionArbol” hace rotación en Y, “alturaArbol“se mueve en Y y “subidaArbolNavidad” es la bandera que decide si sube o baja si sube es true y si baja es false. También la base toma la altura del árbol ya que tiene jerarquía

```
void GiroArbol(bool* keys) {

    if (keys[GLFW_KEY_E] && activaArbol==false) {
        activaArbol = true;
    }


    if (activaArbol) {
        if (alturaArbol <= 10.0f && subidaArbolNavidad) {

            if (alturaArbol >= 9.0f && alturaArbol < 10.0f) {
                subidaArbolNavidad = false;
            }
            else
            {
                rotacionArbol+= deltaTime*15;
                alturaArbol += deltaTime;
            }
        }
        else if (alturaArbol >= 0.0f && subidaArbolNavidad == false) {

            if (alturaArbol <= 1.0f && alturaArbol > 0.0f) {
                subidaArbolNavidad = true;
            }
            else
            {
                rotacionArbol-= deltaTime*15;
                alturaArbol -= deltaTime;
            }
        }
    }

}
```

Animación santa

Se activa la animación con la Q. SantaY, SantaX y SantaZ son para la posición y anguloXsanta y anguloYsanta son para las rotaciones del modelo

```
void animaiconSanta(bool* keys){
    if (keys[GLFW_KEY_Q] && activaSanta == false) {
        activaSanta = true;
    }
    if (activaSanta) {
        if (santaY < 10 && subidaSanta) {
            if (santaY >= 9.7 && santaY < 10) {
                santaY = 10;
            }
            else {
                santaY += deltaTime;
            }

        }
        if (santaY == 10 && anguloXSanta< 360 && subidaSanta){
            if (anguloXSanta>=359.7 && anguloXSanta<360) {
                anguloXSanta = 360;
                subidaSanta = false;
            }
            else {
                anguloXSanta += deltaTime * 15;
                santaZ += deltaTime;
            }

        }
```

```
if (anguloXSanta == 360 && santaY > 0 && subidaSanta == false) {
    if (santaY <=0.2  && santaY >0) {
        santaY = 0;
        anguloYSanta = 180;
    }
    else {
        santaY -= deltaTime;
        anguloYSanta += deltaTime * 18;
    }
}


if (anguloYSanta <= 180 && santaY == 0 && subidaSanta == false && santaZ>0) {
    if (santaZ <= 0.2 && santaZ > 0) {
        santaZ = 0;
        anguloYSanta = 0;
        anguloXSanta = 0;
        subidaSanta = true;
    }
    else {
        santaZ -= deltaTime;
        anguloYSanta -= deltaTime * 8;
    }
}
```

Animación Compleja:

ContadorVueltasCandase sirve para que de 3 vueltas, movZCandase sirve para que se mueva en Z, anguloYCandase es el que da rotacion en Y y “movYCandase” es el que sirve para mover en Y a candase.

```
void animacionCandase() {

    if (contadorVueltasCandase<3 && movZCandase<36){

        if (contadorVueltasCandase == 1.0f && movYCandase<2.0f) {
            if (movYCandase >= 2.7 && movYCandase < 3.0f) {
                movYCandase = 3.0f;
            }
            else {
                movYCandase += 0.5 * deltaTime;
            }

        }
        if (contadorVueltasCandase == 2.0f && movYCandase >0.0f) {
            if (movYCandase >= 0.3 && movYCandase < 0.0f) {
                movYCandase = 0.0f;
            }
            else {
                movYCandase -= 0.5 * deltaTime;
            }

        }

        if (anguloYCandase >= 357 && anguloYCandase < 360) {
            anguloYCandase = 0.0f;
            contadorVueltasCandase++;
            if (contadorVueltasCandase == 3) {
                movZCandase = 36;
            }
        }
```

```
        else {
            anguloYCandase += 30 * 0.5 * deltaTime;
            movZCandase += 0.5 * deltaTime;
        }

    }


    if (contadorVueltasCandase == 3) {
            if (anguloYCandase >= 178 && anguloYCandase<180) {
                anguloYCandase = 180;
                contadorVueltasCandase++;
            }
            else {
                anguloYCandase += 30 * 0.5 * deltaTime;

            }
        }
    if (contadorVueltasCandase == 4 && movZCandase >0.0f) {
        if (movZCandase <= 0.3 && movZCandase > 0.0f) {
            movZCandase = 0.0f;
            contadorVueltasCandase++;
        }
        else {
            movZCandase -= 0.5 * deltaTime;
        }

    }
```

```
if (contadorVueltasCandase == 5 && movZCandase == 0.0f) {
    if (anguloYCandase <= 2 && anguloYCandase > 0.0f) {
        anguloYCandase = 0;
        contadorVueltasCandase=0.0f;
    }
    else {
        anguloYCandase -= 30 * 0.5 * deltaTime;

    }
}
```

Phineas

movXPhienas y movZPhienas sirven para que se muevan en X y Z respectivamente, anguloYPhineas sirve para la rotacion en Y de phineas, movZPhineas y movXPhineas sirven para mover a phienas en X y Z

```
void animacionPhineas() {
    if (movZPhineas < 20.0f && movXPhineas == 0.0f) {
        if (movZPhineas >= 19.7f && movZPhineas < 20.0f) {
            if (anguloYPhineas < 90) {
                anguloYPhineas += 0.5f * deltaTime * 50;
                movZPhineas = 19.9f;
            }
            else {
                anguloYPhineas = 90;
                movZPhineas = 20.0f;
            }
        }
        else {
            movZPhineas += 0.5f * deltaTime ;
        }

    }

    if (movXPhineas < 10 && movZPhineas == 20.0f) {
        if (movXPhineas >= 9.7 && movXPhineas < 10) {
            if (anguloYPhineas >= 90 && anguloYPhineas < 180) {
                anguloYPhineas += movOffset * deltaTime * 50;
                movXPhineas = 9.9;
            }
            else {
                anguloYPhineas = 180;
                movXPhineas = 10.0f;
            }
        }
        else {
            movXPhineas += movOffset * deltaTime ;
        }

    }
```

```
if (movZPhineas > 0 && movXPhineas == 10.0f) {
    if (movZPhineas <= 0.3 && movZPhineas > 0) {
        if (anguloYPhineas >= 180.0f && anguloYPhineas < 270.0f) {
            anguloYPhineas += movOffset * deltaTime * 50;
            movZPhineas = 0.1;
        }
        else {
            anguloYPhineas = 270;
            movZPhineas = 0;
        }
    }
    else {
        movZPhineas -= movOffset * deltaTime ;
    }

}

if (movZPhineas == 0.0f && movXPhineas > 0.0f) {
    if (movXPhineas <= 0.3 && movXPhineas > 0) {
        if (anguloYPhineas >= 270.0f && anguloYPhineas < 360.0f) {
            anguloYPhineas += movOffset * deltaTime * 50;
            movXPhineas = 0.1;
        }
        else {
            anguloYPhineas = 0;
            movXPhineas = 0;
        }
    }
    else {
        movXPhineas -= movOffset * deltaTime ;
    }

}
}
```

Animación ferb:

“idaFerb” es booleana sirve para saber si va llenado o regresando si es true va en Z positivo y si es false va en Z negativo. “movZFerb” y “movYFerb” sirven para el movimiento de ferb en Z y Y respectivamente. “anguloXFerb” y “anguloYFerb” sirven para la rotacion en X y Y respectivamente.

```
//Animacion Ferb se lanza
void animacionFerb(){

    if (movZFerb < 20 && idaFerb) {

        if (movZFerb >= 19.8 && movZFerb < 20  ) {
            movZFerb = 20;
        }
        else {
            movZFerb += movOffset * deltaTime;
        }
    }

    if (movZFerb >= 20 && movZFerb <26  && idaFerb) {

        if (anguloXFerb < 90) {
            movZFerb += movOffset * deltaTime;
            anguloXFerb += movOffset * deltaTime * 30;
            movYFerb += movOffset * deltaTime;

            if (anguloXFerb >= 88.5 && anguloXFerb < 90) {
                anguloXFerb = 90.0f;
            }
        }

            else if (movYFerb>0 && anguloXFerb==90)
            {
                movYFerb -= movOffset * deltaTime;
                movZFerb += movOffset * deltaTime;
                if (movYFerb <= 0.1 && movYFerb > 0) {
                    movYFerb = 0.0f;
                    movZFerb = 26.0f;
                }
            }
```

```
if (movZFerb==26.0f && movYFerb==0.0f && anguloXFerb > 0 && idaFerb) {
    if(anguloXFerb > 0.1)
        anguloXFerb -= movOffset * deltaTime * 30;
    else {
        anguloXFerb = 0.0f;
    }
}


if (movZFerb == 26.0f && movYFerb == 0.0f &&  anguloYFerb < 180 ) {

    //printf("%f\n", anguloYFerb);--si sale error es el angulo restar o aumentar
    if (anguloYFerb >= 178.0f && anguloYFerb < 180.0f) {
        anguloYFerb = 180.0f;
        idaFerb = false;
    }
    else {

        anguloYFerb += movOffset * deltaTime * 30;



    }
}


if (idaFerb == false && movZFerb>0.0f ) {

    if (movZFerb<=0.3f && movZFerb>0) {
        movZFerb = 0.0f;
    }
    else {
        movZFerb -= movOffset * deltaTime;
    }

}
```

```
        if (idaFerb == false && movZFerb==0.0f ) {
            if (anguloYFerb > 0.2f) {
                anguloYFerb -= movOffset * deltaTime*30;
            }
            else {
                idaFerb = true;
                anguloYFerb = 0.0f;
            }

        }


    }
```

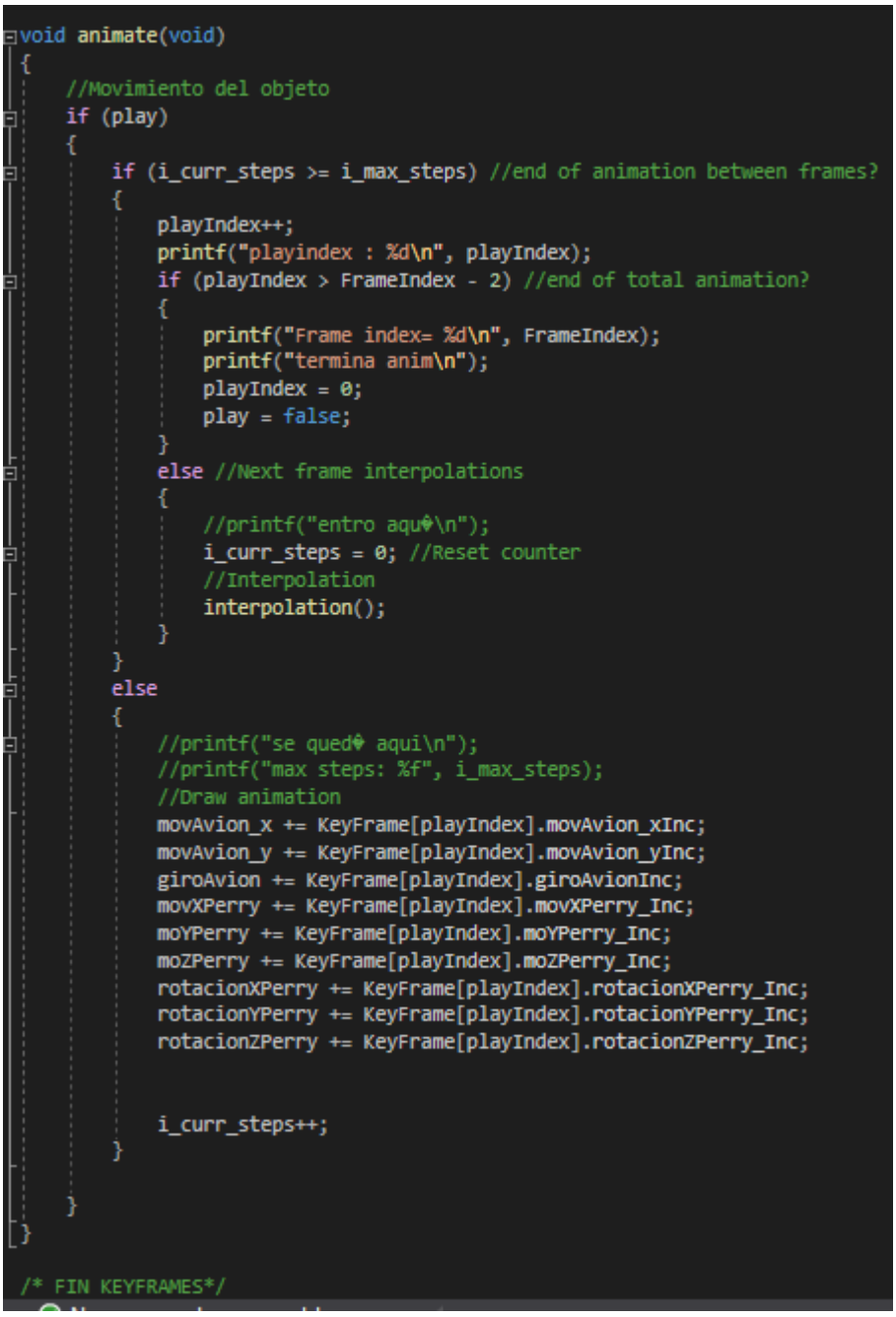
Animación Perry:

Con la implementación ya existente de keyFrame agregamos los movimientos de Perry

```
//NEW// Keyframes
float posXavion = 2.0, posYavion = 2.0, posZavion = 0, movXPerry=0.0f, moYPerry = 0.0f, moZPerry = 0.0f, rotacionXPerry=0.0f, rotacionYPerry = 0.0f, rotacionZPerry = 0.0f;
float   movAvion_x = 0.0f, movAvion_y = 0.0f, movXPerry_Inc = 0.0f, moYPerry_Inc = 0.0f, moZPerry_Inc = 0.0f, rotacionXPerry_Inc = 0.0f, rotacionYPerry_Inc = 0.0f, rotacionZPerry_Inc = 0.0f;
float giroAvion = 0;

#define MAX_FRAMES 30
int i_max_steps = 90;
int i_curr_steps = 5;
typedef struct _frame
{
    //Variables para GUARDAR Key Frames
    float movAvion_x;       //Variable para PosicionX
    float movAvion_y;       //Variable para PosicionY
    float movAvion_xInc;        //Variable para IncrementoX
    float movAvion_yInc;        //Variable para IncrementoY
    float giroAvion;
    float giroAvionInc;

    float movXPerry;
    float moYPerry;
    float moZPerry;
    float rotacionXPerry;
    float rotacionYPerry;
    float rotacionZPerry;

    float movXPerry_Inc;
    float moYPerry_Inc;
    float moZPerry_Inc;
    float rotacionXPerry_Inc;
    float rotacionYPerry_Inc;
    float rotacionZPerry_Inc;

}FRAME;
```

```
FRAME KeyFrame[MAX_FRAMES];
int FrameIndex = 13;            //introducir datos
bool play = false;
int playIndex = 0;

void saveFrame(void)
{

    printf("frameindex %d\n", FrameIndex);


    KeyFrame[FrameIndex].movAvion_x = movAvion_x;
    KeyFrame[FrameIndex].movAvion_y = movAvion_y;
    KeyFrame[FrameIndex].giroAvion = giroAvion;
    KeyFrame[FrameIndex].movXPerry = movXPerry;
    KeyFrame[FrameIndex].moYPerry = moYPerry;
    KeyFrame[FrameIndex].moZPerry = moZPerry;
    KeyFrame[FrameIndex].rotacionXPerry = rotacionXPerry;
    KeyFrame[FrameIndex].rotacionYPerry = rotacionYPerry;
    KeyFrame[FrameIndex].rotacionZPerry = rotacionZPerry;

    FrameIndex++;
}

void resetElements(void)
{




    movAvion_x = KeyFrame[0].movAvion_x;
    movAvion_y = KeyFrame[0].movAvion_y;
    giroAvion = KeyFrame[0].giroAvion;
    movXPerry = KeyFrame[0].movXPerry;
    moYPerry = KeyFrame[0].moYPerry;
    moZPerry = KeyFrame[0].moZPerry;
    rotacionXPerry = KeyFrame[0].rotacionXPerry;
    rotacionYPerry = KeyFrame[0].rotacionYPerry;
    rotacionZPerry = KeyFrame[0].rotacionZPerry;


}
```

```
void interpolation(void)
{


    KeyFrame[playIndex].movAvion_xInc = (KeyFrame[playIndex + 1].movAvion_x - KeyFrame[playIndex].movAvion_x) / i_max_steps;
    KeyFrame[playIndex].movAvion_yInc = (KeyFrame[playIndex + 1].movAvion_y - KeyFrame[playIndex].movAvion_y) / i_max_steps;
    KeyFrame[playIndex].giroAvionInc = (KeyFrame[playIndex + 1].giroAvion - KeyFrame[playIndex].giroAvion) / i_max_steps;
    KeyFrame[playIndex].movXPerry_Inc = (KeyFrame[playIndex + 1].movXPerry- KeyFrame[playIndex].movXPerry) / i_max_steps;
    KeyFrame[playIndex].moYPerry_Inc = (KeyFrame[playIndex + 1].moYPerry- KeyFrame[playIndex].moYPerry) / i_max_steps;
    KeyFrame[playIndex].moZPerry_Inc = (KeyFrame[playIndex + 1].moZPerry- KeyFrame[playIndex].moZPerry) / i_max_steps;
    KeyFrame[playIndex].rotacionXPerry_Inc = (KeyFrame[playIndex + 1].rotacionXPerry- KeyFrame[playIndex].rotacionXPerry) / i_max_steps;
    KeyFrame[playIndex].rotacionYPerry_Inc = (KeyFrame[playIndex + 1].rotacionYPerry- KeyFrame[playIndex].rotacionYPerry) / i_max_steps;
    KeyFrame[playIndex].rotacionZPerry_Inc = (KeyFrame[playIndex + 1].rotacionZPerry- KeyFrame[playIndex].rotacionZPerry) / i_max_steps;

}
```



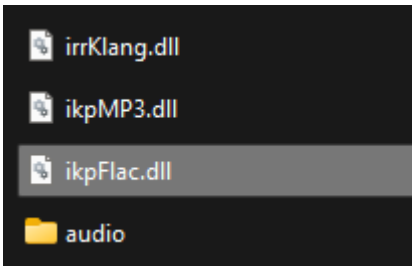
```
void animate(void)
{
    //Movimiento del objeto
    if (play)
    {
        if (i_curr_steps >= i_max_steps) //end of animation between frames?
        {
            playIndex++;
            printf("playindex : %d\n", playIndex);
            if (playIndex > FrameIndex - 2) //end of total animation?
            {
                printf("Frame index= %d\n", FrameIndex);
                printf("termina anim\n");
                playIndex = 0;
                play = false;
            }
            else //Next frame interpolations
            {
                //printf("entro aqu\n");
                i_curr_steps = 0; //Reset counter
                //Interpolation
                interpolation();
            }
        }
        else
        {
            //printf("se qued aqui\n");
            //printf("max steps: %f", i_max_steps);
            //Draw animation
            movAvion_x += KeyFrame[playIndex].movAvion_xInc;
            movAvion_y += KeyFrame[playIndex].movAvion_yInc;
            giroAvion += KeyFrame[playIndex].giroAvionInc;
            movXPerry += KeyFrame[playIndex].movXPerry_Inc;
            moYPerry += KeyFrame[playIndex].moYPerry_Inc;
            moZPerry += KeyFrame[playIndex].moZPerry_Inc;
            rotacionXPerry += KeyFrame[playIndex].rotacionXPerry_Inc;
            rotacionYPerry += KeyFrame[playIndex].rotacionYPerry_Inc;
            rotacionZPerry += KeyFrame[playIndex].rotacionZPerry_Inc;


            i_curr_steps++;
        }

    }
}

/* FIN KEYFRAMES*/
```

Declaración de la animación de Perry

```
//KEYFRAMES DECLARADOS INICIALES



KeyFrame[0].movAvion_x = 0.0f;
KeyFrame[0].movAvion_y = 0.0f;
KeyFrame[0].giroAvion = 0.0f;
KeyFrame[0].rotacionXPerry = 0.0f;
KeyFrame[0].moYPerry = 0.0f;
KeyFrame[0].moZPerry = 0.0f;
KeyFrame[0].rotacionYPerry = 0.0f;
KeyFrame[0].movXPerry = 0.0f;



KeyFrame[1].movAvion_x = 1.0f;
KeyFrame[1].movAvion_y = 2.0f;
KeyFrame[1].giroAvion = 0;
KeyFrame[1].rotacionXPerry = 90.0f;
KeyFrame[1].moYPerry = 2.5f;
KeyFrame[1].moZPerry = 2.5f;
KeyFrame[1].rotacionYPerry = 0.0f;
KeyFrame[1].movXPerry = 0.0f;


KeyFrame[2].movAvion_x = 2.0f;
KeyFrame[2].movAvion_y = 0.0f;
KeyFrame[2].giroAvion = 0;
KeyFrame[2].rotacionXPerry = 180.0f;
KeyFrame[2].moYPerry = 5.0f;
KeyFrame[2].moZPerry = 5.0f;
KeyFrame[2].rotacionYPerry = 0.0f;
KeyFrame[2].movXPerry = 0.0f;
```

Sonido:

Creamos en include una carpeta llamada “irrklang” y agregamos “irrKlang.dll”,” ikpMP3.dll” y “ikpFlac.dll”, en la carpeta lib agregamos “irrKlang.lib” y creamos una nueva carpeta llamada audio para los sonidos utilizados.



Biblioteca

```
#include <irrKlang/irrKlang.h>
```

Declaramos para usar la biblioteca

```
irrklang::ISoundEngine* SoundEngine = irrklang::createIrrKlangDevice();//audio
```

Declaramos para usar el sonido lo que esta entre comillas es el audio ya descargado en la carpeta audio.

```
movZFerb = 26.0f;
SoundEngine->play2D("audio/golpeFerb.mp3", false);
```

Terminamos para destruir el uso del audio

```
SoundEngine->drop();
return 0;
```

Referencias:

Biblioteca Audio

download irrKlang. (s. f.). Ambiera. Recuperado 22 de noviembre de 2021, de https://www.ambiera.com/irrklang/downloads.html

Imagen SkyBox:

Facebook - Meld je aan of registreer je. (s. f.). Facebook. Recuperado 22 de noviembre de 2021, de https://www.facebook.com/unsupportedbrowser

Algunos Modelos

Sketchfab - The best 3D viewer on the web. (s. f.). Sketchfab. Recuperado 22 de noviembre de 2021, de https://sketchfab.com/

Audio http://www.sonidosmp3gratis.com/download.php?id=15902&sonido=golpe%20cara