MANUAL TÉCNICO.

Proyecto: Feria.

Integrantes:

- **Zagoya Mellado Roberto Uriel**
  Número de cuenta: 416113778
- **Orozco Hernández Alexis**
  Número de cuenta: 313140255

Asignatura: Computación Grafica e interacción humana
Grupo teoría: 04
Grupo Laboratorio: 04

Nombre del profesor: **ING. José Roque Román Guadarrama**

Escuela: Universidad Nacional Autónoma de México
Facultad de ingeniería.

Índice

Introducción.

Este proyecto trata de la simulación de una feria de pueblo, o parque de diversiones como normalmente lo conocen, se llevó a cabo con el cuidado y principal prioridad de cumplir los alineamientos requeridos para la entrega del mismo y definidos en clase y por los requisitos abordados en las normas vistas en el documento de especificaciones, con el fin de que se pueda comprender este proyecto y presentarse ejemplos claros de cada aplicación, función, modelaje, etc. de esta feria se desarrollara este documento donde se mostrara elementos de la programación como la elaboración de las primitivas, como se exportaron los modelos, funciones que se emplean, etc. como del correcto manejado del programa en ejecución es decir como poder moverse dentro de la feria, ver las animaciones, etc. Aun sin saber de programación grafica ni nada por el estilo, es decir que va orientado a un usuario final que no necesita ser especialista en el área para poder ejecutar y controlar este programa de la feria. También el objetivo de este documento va guiado a un usuario final que si tenga conocimientos de la programación grafica para que en caso de requerirse se pueda dar seguimiento de forma sencilla y por ello se adjuntan capturas de pantalla y ejemplos para facilitar la comprensión de este proyecto. Pero principalmente se realiza para que se pueda observar la elaboración del programa paso a pasa y cómo es que se elaboró y verificar que este cumple con los objetivos. Primero partimos de una breve introducción como se está observando, después procedemos a mostrar los objetivos del proyecto y finalmente la muestra la programación de los modelos exportados, manipulación del programa, animaciones, y la elaboración de los juegos realizados por primitivas, así como sus animaciones, las funciones, iluminación y texturizado de estas últimas, todo esto con descripción y ejemplos de lo que se realizó.

Controles.

Los controles del programa en ejecución se definieron al inicio del código como se puede mostrar en la siguiente captura.

```
/*---------------------------------------------------------*/
/* ----------------  Proyecto Final ----------------------*/
/*-----------------    2019-2   ----------------------------*/
/*-- Computación gráfica e interacción humano computadora --*/
/*------------------Integrantes:----------------------------*/
/*------------- Orozco Hernandez Alexis  -----------------*/
/*------------- Zagoya Mellado Roberto Uriel-----------------*/
/*--------------  Version VS 2017  ----------------------*/
/*---------------------Grupo de teoria: 04----------------*/
/*-----------------  Grupo 04  --------------------------*/

/*--------------------Controles:
                      W: mover hacia delante
                      S: Mover hacia atras
                      D: Mover hacia la derecha
                      A: Mover hacia la izquierda
                      P: Poner de noche
                      O : poner de dia
                      R: Activar la aniacion de la rueda de la fortuna
                      T: Activar las tazas
                      D: Desactivar las tazas
                      K: Activar el carro de comida
                      J: Activar los carros chocones
                      L: Detiene los carros chocones
                      H: Activa los caballos
                      V: Activa animacion por keyframes de la montaña */
```

En esta parte se puede observar los controles que manejan el programa en ejecución, esto es gracias a los casos declarados en las líneas posteriores de código.

```
// ----------------------------------------------------------------------------
void my_input(GLFWwindow *window)
{
    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);

    if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS) {
        camera.ProcessKeyboard(FORWARD, (float)deltaTime);
    }
    if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS)
        camera.ProcessKeyboard(BACKWARD, (float)deltaTime);
    if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS)
        camera.ProcessKeyboard(LEFT, (float)deltaTime);
    if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS)
        camera.ProcessKeyboard(RIGHT, (float)deltaTime);
    //if (glfwGetKey(window, GLFW_KEY_P) == GLFW_PRESS)
    //  play = true;
    //if (glfwGetKey(window, GLFW_KEY_O) == GLFW_PRESS)
    //  play = false;
    /*if (glfwGetKey(window, GLFW_KEY_RIGHT) == GLFW_PRESS)
        direction_right = true;
    if (glfwGetKey(window, GLFW_KEY_LEFT) == GLFW_PRESS)
        direction_right = false;*/
```

```
2046    if (glfwGetKey(window, GLFW_KEY_P) == GLFW_PRESS) {//poner de noche
2047        ambient1 = 0.04f;
2048        ambient2 = 0.04f;
2049        ambient3 = 0.04f;
2050    }
2051    if (glfwGetKey(window, GLFW_KEY_O) == GLFW_PRESS) {//poner de dia
2052        ambient1 = 1.0f;
2053        ambient2 = 1.0f;
2054        ambient3 = 1.0f;
2055    }
2056    if (glfwGetKey(window, GLFW_KEY_R) == GLFW_PRESS)//PARA ACTIVAR LA NIMACION DE LA RUEDA
2057        activate_rueda = true;
2058        //----------para tasas giratorias
2059    if (glfwGetKey(window, GLFW_KEY_T) == GLFW_PRESS)//activar taas
2060        activate_tasas = true;
2061    if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS)//descativar tasas
2062        activate_tasas = false;
2063    if (glfwGetKey(window, GLFW_KEY_K) == GLFW_PRESS)//para activar el carro de comida
2064        activate_cc = true;
2065    if (glfwGetKey(window, GLFW_KEY_J) == GLFW_PRESS) {//para activar los carros chocones
2066        stop_cart1 = false;
2067        stop_cart2 = false;
2068        stop_cart3 = false;
```

```
2102    if (glfwGetKey(window, GLFW_KEY_L) == GLFW_PRESS) {//detiene los carros chocones
2103        stop_cart1 = true;
2104        stop_cart2 = true;
2105        stop_cart3 = true;
2106    }
2107
2108    if (glfwGetKey(window, GLFW_KEY_H) == GLFW_PRESS) {//activa los caballos
2109        activate_caballos = true;
2110    }
2111
2112    //-------------para activar la animacion por keyframes-------------------------
2113    if (glfwGetKey(window, GLFW_KEY_V) == GLFW_PRESS) {//activa animacion por keyframes de la montaña
2114        if (activate_coaster == false && (FrameIndex > 1))
2115        {
2116            resetElements();
2117            //First Interpolation
2118            interpolation();
2119
2120            activate_coaster = true;
2121            playIndex = 0;
2122            i_curr_steps = 0;
2123        }
2124        else
2125        {
2126            activate_coaster = false;
2127        }
2128    }
```

Donde cómo se puede observar dependiendo de la tecla presionada, valida las variables para que dependiendo del valor de cada variable en una función que se evalué, haga una determinada función.

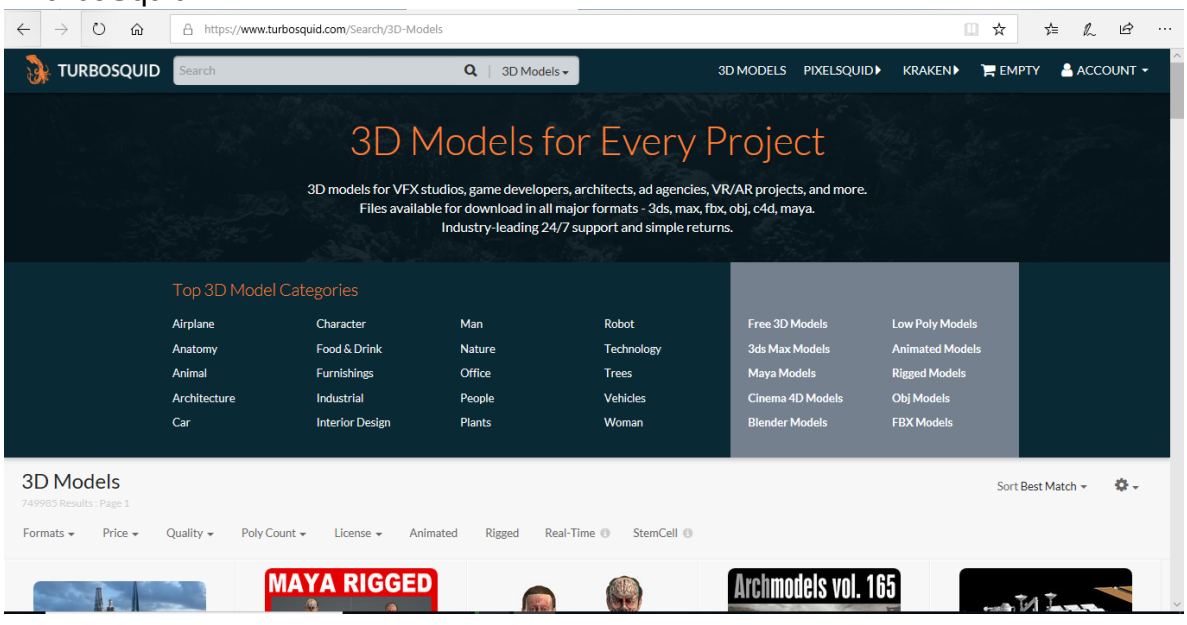
El ratón se controla gracias a la siguiente función y a donde lo muevas va a enfocar la cámara.

```
// glfw: whenever the mouse moves, this callback is called
// -------------------------------------------------------
void mouse_callback(GLFWwindow* window, double xpos, double ypos)
{
    if (firstMouse)
    {
        lastX = xpos;
        lastY = ypos;
        firstMouse = false;
    }

    double xoffset = xpos - lastX;
    double yoffset = lastY - ypos; // reversed since y-coordinates go from bottom to top

    lastX = xpos;
    lastY = ypos;

    camera.ProcessMouseMovement(xoffset, yoffset);
}
```

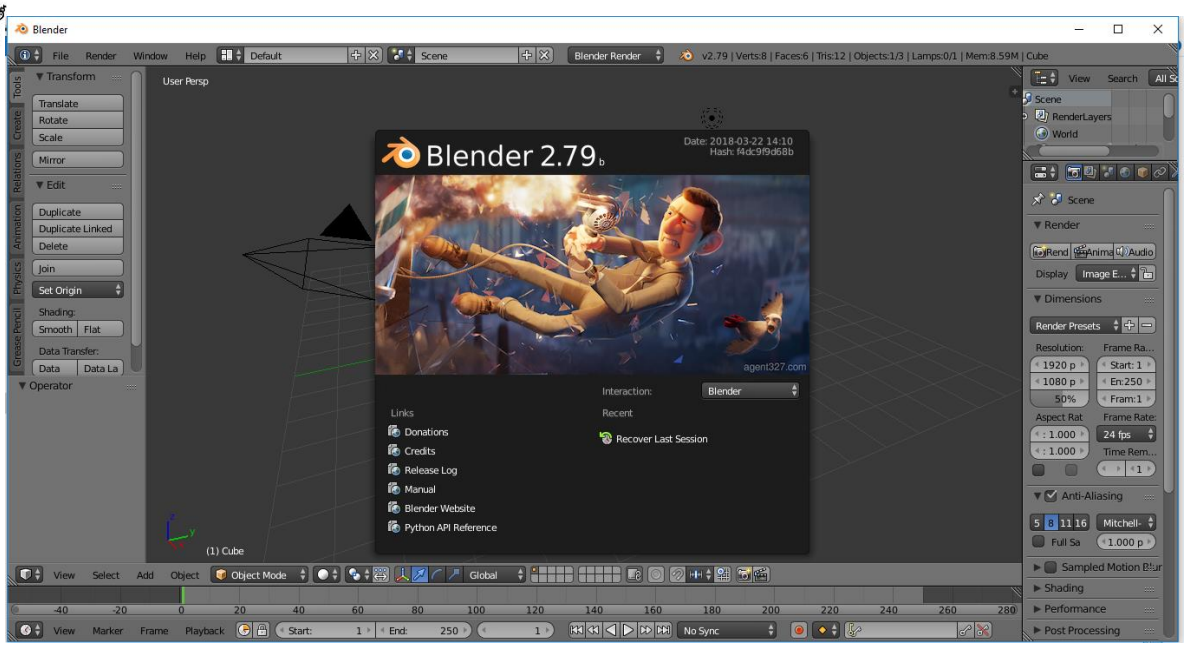
Modelos exportados

En esta sección se encuentran los modelos que se descargaron en la página de internet TurboSquid y posteriormente se colocó la textura en Blender.

TurboSquid



Blender



En esta parte se pasan los modelos como parámetros para poder colocarlo de la forma deseada en la feria

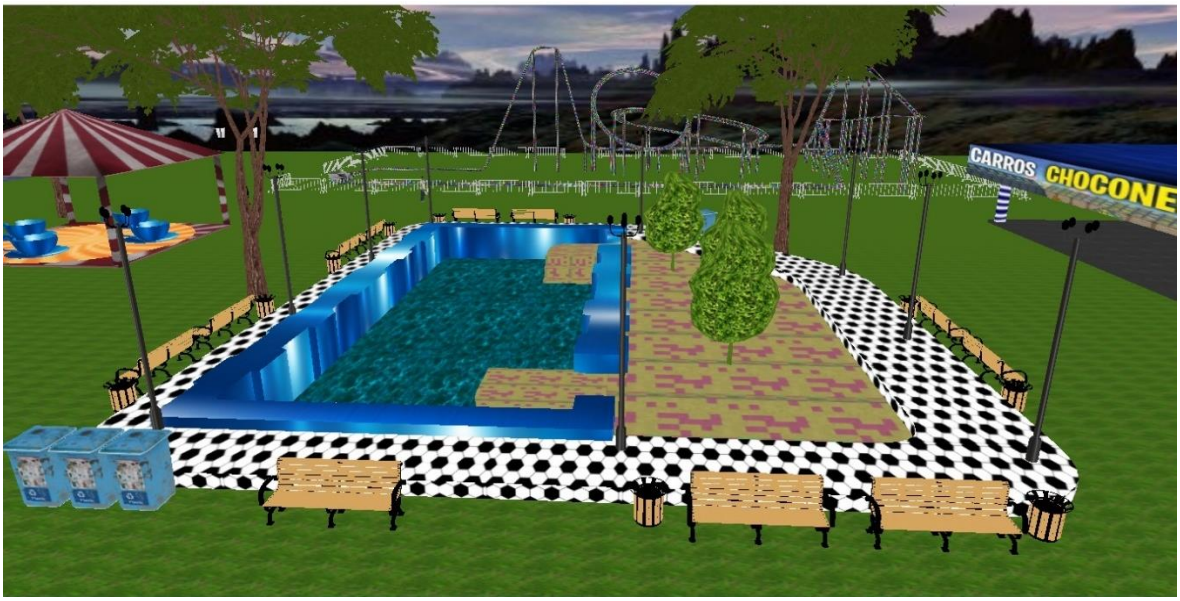
```
void display(Shader shader, Shader lampshader, Shader lightingshader, Shader textureshader, Shader lightposshader,
    Model carril, Model personaje, Model personaje2, Model futbolito, Model carpa_tasas, Model base_tasas, Model tasa, Model basketball, Model tiro_blanco,
    Model jugos, Model ambulante1, Model barrera, Model carpa_fut, Model bumper_car, Model pista, Model silla, Model trash, Model trash2, Model poste_luz1, Model arbol1
    Model premios, Model toilet, Model base_caballos, Model giro_caballos, Model caballo, Model coaster_cart)//ahora si resiven parametros: shader, carril y personaje
{
```

Los modelos instanciados se declaran de la siguiente forma.

```
2232    // Load models
2233    Model carril = ((char *)"Models/parque/parqueWL.obj");//para cargar el modelo del carril
2234    Model personaje = ((char *)"Models/personaje/muro.obj");//para cargar el modelo del personaje
2235    Model personaje2 = ((char *)"Models/man/man.obj");//para cargar el modelo del personaje
2236    Model carpa_tasas = ((char *)"Models/tasas_giratorias/carpa_tasas.obj");
2237    Model base_tasas = ((char *)"Models/tasas_giratorias/base_giro.obj");
2238    Model tasa = ((char *)"Models/tasas_giratorias/tasa.obj");
2239    Model futbolito = ((char *)"Models/futbolito/futbolito.obj");
2240    Model carpa_fut = ((char *)"Models/futbolito/carpa/carpa_futbolito.obj");
2241    Model basketball = ((char *)"Models/basketball/basketball.obj");
2242    Model jugos = ((char *)"Models/food_carts/jugos/jugos.obj");
2243    //Model jugos = ((char *)"Models/coaster_cart/coaster_cart.obj");
2244
2245    Model ambulante1 = ((char *)"Models/food_carts/jugos/jugos.obj");//carro de comida ambulante
2246    Model barrera = ((char *)"Models/barrera/Barrera.obj");
2247    Model bumper_car = ((char *)"Models/bumper/carrito_chocon.obj");
2248    Model pista = ((char *)"Models/bumper/pista_2.obj");
2249    Model silla = ((char *)"Models/elementos/silla1.obj");
2250    Model poste_luz1 = ((char *)"Models/elementos/poste_luz1.obj");
2251    Model trash = ((char *)"Models/elementos/trash.obj");
2252    Model trash2 = ((char *)"Models/elementos/trash/bote.obj");
2253    Model arbol1 = ((char *)"Models/elementos/arboles/bigtree.obj");
2254    Model tiro_blanco = ((char *)"Models/tiro_blanco/dart_board.obj");
2255    Model premios = ((char *)"Models/juguetes/premios/premios.obj");
2256    Model toilet = ((char *)"Models/toilet/toilet.obj");
2257
2258    Model base_caballos = ((char *)"Models/caballos/base.obj");
2259    Model giro_caballos = ((char *)"Models/caballos/c_giratorio.obj");
2260    Model caballo = ((char *)"Models/caballos/caballo.obj");
2261
2262    Model coaster_cart = ((char *)"Models/coaster_cart/coaster_cart.obj");
2263
2264    Model local = ((char *)"Models/ducks_game/local.obj");
2265    Model duck = ((char *)"Models/ducks_game/duck.obj");
2266    Model tubo = ((char *)"Models/ducks_game/tubo.obj");
2267    Model arma = ((char *)"Models/arma/arma.obj");
```

A continuación, se muestra las imágenes de los modelos empleados en la feria, cabe resaltar que estos modelos son como los hemos estado trabajando en las practicas, es decir se pueden rotar, trasladar y escalar cada modelo, para ponerlo en la posición deseada y la escala deseada de la feria.

Fuente:



Como se puede observar los modelos empleados en la fuente tiene texturas las cuales se agregaron usando Blender y además cabe resaltar que varios modelos como los arboles dentro de la misma son modelos incluidos, por ello es por lo que aquí no se describe a fondo, pero se pueden verificar en las líneas de código.

```
1672    //-------------------elementos clasicos--------------------------
1673    //sillas
1674    model = glm::translate(tmp, glm::vec3(-24.0f, 5.67f, 10.0));
1675    model = glm::scale(model, glm::vec3(0.017f, 0.0140f, 0.017f));
1676    model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
1677    //model = glm::rotate(model, glm::radians(rot_llanta), glm::vec3(1.0f, 0.0f, 0.0f));
1678    shader.setMat4("model", model);
1679    silla.Draw(shader);
1680
1681    model = glm::translate(tmp, glm::vec3(-23.0f, 5.67f, 13.0));
1682    model = glm::scale(model, glm::vec3(0.017f, 0.0140f, 0.017f));
1683    shader.setMat4("model", model);
1684    trash.Draw(shader); //bote de basura
1685
1686    model = glm::translate(tmp, glm::vec3(-24.0f, 5.67f, 5.0));
1687    model = glm::scale(model, glm::vec3(0.017f, 0.0140f, 0.017f));
1688    model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
1689    shader.setMat4("model", model);
1690    silla.Draw(shader);
1691
1692    model = glm::translate(tmp, glm::vec3(-23.0f, 5.67f, 2.0f));
1693    model = glm::scale(model, glm::vec3(0.017f, 0.0140f, 0.017f));
1694    shader.setMat4("model", model);
1695    trash.Draw(shader);//bote de basura
1696
1697    model = glm::translate(tmp, glm::vec3(-24.0f, 5.67f, -7.0));
1698    model = glm::scale(model, glm::vec3(0.017f, 0.0140f, 0.017f));
1699    model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
1700    shader.setMat4("model", model);
1701    silla.Draw(shader);
1702
```

Sujeto1

Sujeto2

Con las siguientes líneas código se llaman los shader adecuados para poder dibujar los modelos declarados anteriormente.

```
//personaje1
model = glm::translate(tmp, glm::vec3(-40.0f, 5.68f, 0.0));
model = glm::scale(model, glm::vec3(0.015f, 0.015f, 0.015f));
//model = glm::rotate(model, glm::radians(rot_personaje), glm::vec3(0.0f, 1.0f, 0.0f));

//model = glm::rotate(model, glm::radians(rot_llanta), glm::vec3(1.0f, 0.0f, 0.0f));
shader.setMat4("model", model);
personaje.Draw(shader); //Izq delantera

//personaje 2
model = glm::translate(tmp, glm::vec3(-10.0f, 5.68f, 40.0));
model = glm::scale(model, glm::vec3(0.015f, 0.015f, 0.015f));
//model = glm::rotate(model, glm::radians(rot_personaje), glm::vec3(0.0f, 1.0f, 0.0f));
```

Tazas giratorias

Como se puede observar este modelo se activa con la tecla T, con lo que realiza su función de girar, esto se debe gracias a la sección de animación. Que como se mencionó anteriormente al recibir un parámetro después de oprimir la tecla T, se activa la animación.

Con las siguientes líneas código se llaman los shader adecuados para poder dibujar los modelos declarados anteriormente.

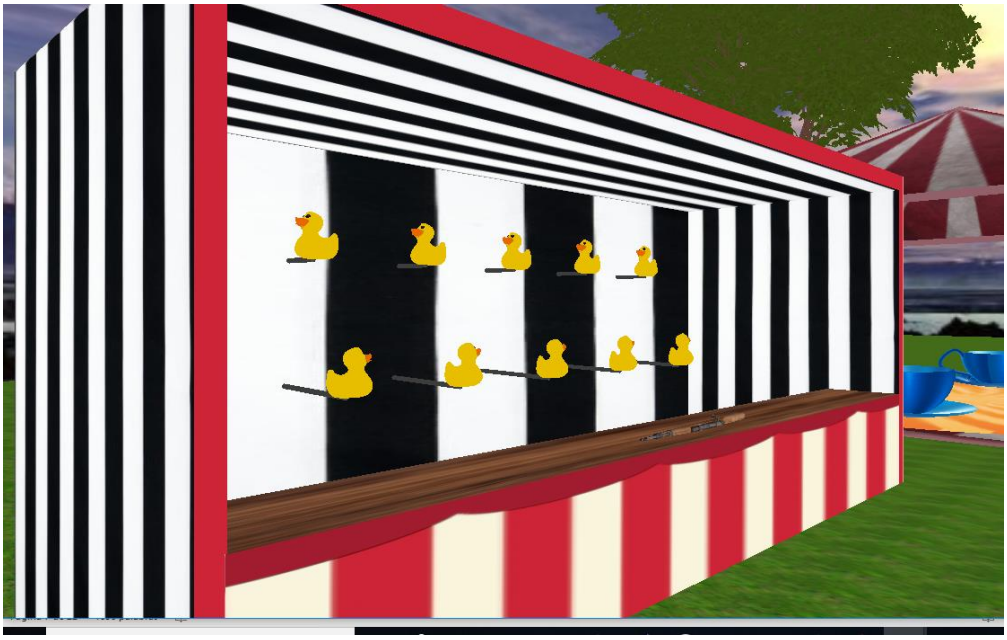
```
//----------------------tasas giratorias------------------------------
model = glm::translate(tmp, glm::vec3(12.0f, 5.7f, -40.0));
model = glm::scale(model, glm::vec3(0.9f, 0.8f, 0.9f));
//model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
shader.setMat4("model", model);
carpa_tasas.Draw(shader);

model = glm::translate(model, glm::vec3(0.0f, 0.05f, 0.0f));
model = glm::rotate(model, glm::radians(rot_base_tasas), glm::vec3(0.0f, 1.0f, 0.0f)); //para el movimie
shader.setMat4("model", model);
temp2 = model;
base_tasas.Draw(shader);

model = glm::translate(model, glm::vec3(0.0f, 0.0f, -5.5));
model = glm::rotate(model, glm::radians(rot_tasas), glm::vec3(0.0f, 1.0f, 0.0f));//para rotar la tasa
shader.setMat4("model", model);
tasa.Draw(shader);

model = temp2;
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 5.5));
model = glm::rotate(model, glm::radians(rot_tasas), glm::vec3(0.0f, 1.0f, 0.0f));//para rotar la tasa
shader.setMat4("model", model);
tasa.Draw(shader);

model = temp2;
model = glm::translate(model, glm::vec3(5.5f, 0.0f, 0.0));
model = glm::rotate(model, glm::radians(rot_tasas), glm::vec3(0.0f, 1.0f, 0.0f));//para rotar la tasa
shader.setMat4("model", model);
```

Caballos de feria.

Con las siguientes líneas código se llaman los shader adecuados para poder dibujar los modelos declarados anteriormente.

```
1631    //-----------------------------------caballos-----------------------------------------
1632    model = glm::translate(tmp, glm::vec3(-20.0f, 5.6f, 45.0f));
1633    model = glm::scale(model, glm::vec3(0.30f, 0.30f, 0.30f));
1634    //model = glm::rotate(model, glm::radians(rot_llanta), glm::vec3(1.0f, 0.0f, 0.0f));
1635    shader.setMat4("model", model);
1636    base_caballos.Draw(shader);
1637
1638    model = glm::translate(model, glm::vec3(0, -3.5f, 0.0));
1639    model = glm::scale(model, glm::vec3(1.0f, 1.15f, 1.0f));
1640    model = glm::rotate(model, glm::radians(rot_caballos), glm::vec3(0.0f, 1.0f, 0.0f));
1641    shader.setMat4("model", model);
1642    temp2 = model;
1643    giro_caballos.Draw(shader);
1644
1645    model = glm::translate(model, glm::vec3(18.0f, y_caballos, 0.0));
1646    model = glm::scale(model, glm::vec3(1.1f, 1.1f, 1.0f));
1647    model = glm::rotate(model, glm::radians(40.0f), glm::vec3(0.0f, 1.0f, 0.0f));
1648    shader.setMat4("model", model);
1649    caballo.Draw(shader);
1650
1651    model = temp2;
1652    model = glm::translate(model, glm::vec3(-18.0f, y_caballos, 0.0));
1653    model = glm::scale(model, glm::vec3(1.1f, 1.1f, 1.0f));
1654    model = glm::rotate(model, glm::radians(220.0f), glm::vec3(0.0f, 1.0f, 0.0f));
1655    shader.setMat4("model", model);
1656    caballo.Draw(shader);
```

Juego de azar.



Con las siguientes líneas código se llaman los shader adecuados para poder dibujar los modelos declarados anteriormente.

```
1291    //-------------------------------------------------------------------------------------------------------
1292    //------------------------------ juego de patos -----------------------------------------
1293    model = glm::translate(tmp, glm::vec3(-6.0f, 5.5f, -40.0));
1294    model = glm::scale(model, glm::vec3(0.55f, 0.55f, 0.55f));
1295    model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
1296    shader.setMat4("model", model);
1297    local.Draw(shader);
1298    tmp_game = model;
1299
1300    model = glm::translate(model, glm::vec3(00.0f, -0.5f, duck_pos1));
1301    model = glm::translate(model, glm::vec3(00.0f, duck_mov1, 0.0f));
1302    shader.setMat4("model", model);
1303    tmp_ducks = model;
1304    tubo.Draw(shader);
1305
1306    model = glm::translate(model, glm::vec3(-1.0f, 4.8f, 3.7f));
1307    model = glm::rotate(model, glm::radians(rotpato_inf), glm::vec3(0.0f, 1.0f, 0.0f));
1308    shader.setMat4("model", model);
1309    duck.Draw(shader);
1310
1311    //segundo tubo xD
1312    model = tmp_game;
1313    model = glm::translate(model, glm::vec3(00.0f, -0.5f, duck_pos1));
1314    model = glm::translate(model, glm::vec3(00.0f, duck_mov2, 3.0f));
1315    shader.setMat4("model", model);
1316    tubo.Draw(shader);
1317
1318    model = glm::translate(model, glm::vec3(-1.0f, 4.8f, 3.7f));
1319    model = glm::rotate(model, glm::radians(rotpato_inf), glm::vec3(0.0f, 1.0f, 0.0f));
1320    shader.setMat4("model", model);
1321    duck.Draw(shader);
1322
```

Carros chocones

Este modelo cuenta con animación de la cual se activa presionando la tecla J.

Con las siguientes líneas código se llaman los shader adecuados para poder dibujar los modelos declarados anteriormente.

```
1573    //--------------------carros chocones----------------------------------
1574    model = glm::translate(tmp, glm::vec3(10.0f, 5.7f, 45.0));
1575    model = glm::scale(model, glm::vec3(0.3f, 0.1f, 0.3f));
1576    //model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
1577
1578    shader.setMat4("model", model);
1579    pista.Draw(shader);
1580    tmp_carros = model;
1581    //bumper_car.Draw(shader);
1582
1583    model = glm::translate(model, glm::vec3(x_cart, z_cart, y_cart));//carro1
1584    model = glm::scale(model, glm::vec3(0.4f, 0.8f, 0.4f));
1585    model = glm::rotate(model, glm::radians(rot_cart), glm::vec3(0.0f, 1.0f, 0.0f));
1586    model = glm::rotate(model, glm::radians(rot_cart_z), glm::vec3(0.0f, 0.0f, 1.0f));
1587    shader.setMat4("model", model);
1588    bumper_car.Draw(shader);
1589
1590    model = tmp_carros;
1591    model = glm::translate(model, glm::vec3(x_cart3, y_cart3, z_cart3));//carro 3
1592    model = glm::scale(model, glm::vec3(0.4f, 0.8f, 0.4f));
1593    model = glm::rotate(model, glm::radians(rot_cart3), glm::vec3(0.0f, 1.0f, 0.0f));
1594    model = glm::rotate(model, glm::radians(rot_cart3_z), glm::vec3(0.0f, 0.0f, 1.0f));
1595    shader.setMat4("model", model);
1596    bumper_car.Draw(shader);
1597
1598    model = tmp_carros;
1599    model = glm::translate(model, glm::vec3(x_cart2, y_cart2, z_cart2));//carro2
1600    model = glm::scale(model, glm::vec3(0.4f, 0.8f, 0.4f));
1601    model = glm::rotate(model, glm::radians(rot_cart2), glm::vec3(0.0f, 1.0f, 0.0f));
1602    model = glm::rotate(model, glm::radians(rot_cart2_z), glm::vec3(0.0f, 0.0f, 1.0f));
1603    shader.setMat4("model", model);
1604    bumper_car.Draw(shader);
```

Carrito de comida.

Para la animación de este modelo se requiere oprimir la tecla K

Con las siguientes líneas código se llaman los shader adecuados para poder dibujar los modelos declarados anteriormente.

```
1607    //------------------ comida -------------------------------

1608
1609    //jugos
1610    model = glm::translate(tmp, glm::vec3(x_comida, 5.6f, z_comida));
1611    model = glm::scale(model, glm::vec3(0.80f, 0.80f, 0.80f));
1612    model = glm::rotate(model, glm::radians(rot_comida), glm::vec3(0.0f, 1.0f, 0.0f));
1613    //model = glm::rotate(model, glm::radians(rot_llanta), glm::vec3(1.0f, 0.0f, 0.0f));
1614    shader.setMat4("model", model);
1615    jugos.Draw(shader);
1616
1617
1618    //------------------ comida -------------------------------
```

Carpas de futbolito

Con las siguientes líneas código se llaman los shader adecuados para poder dibujar los modelos declarados anteriormente.

```
//-----------------------------------futbolitos-------------------------------------
//carpa
model = glm::translate(tmp, glm::vec3(-50.0f, 5.45f, -25.0));
model = glm::scale(model, glm::vec3(0.09f, 0.09f, 0.09f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
shader.setMat4("model", model);
carpa_fut.Draw(shader);
//-

model = glm::translate(model, glm::vec3(70.0f, 2.0f, 30.0));
model = glm::scale(model, glm::vec3(17.0f, 17.0f, 17.0f));
model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));

//model = glm::rotate(model, glm::radians(rot_llanta), glm::vec3(1.0f, 0.0f, 0.0f));
shader.setMat4("model", model);
futbolito.Draw(shader); //

model = glm::translate(model, glm::vec3(0.0f, 0.0f, 3.5));
shader.setMat4("model", model);
futbolito.Draw(shader); //

model = glm::translate(model, glm::vec3(0.0f, 0.0f, 3.5));
shader.setMat4("model", model);
futbolito.Draw(shader);

model = glm::translate(model, glm::vec3(0.0f, 0.0f, 3.5));
shader.setMat4("model", model);
futbolito.Draw(shader);

model = glm::translate(model, glm::vec3(-3.0f, 0.0f, 0.0));
shader.setMat4("model", model);
futbolito.Draw(shader);
```

Este modelo no cuenta con animación, sin embargo, cabe resaltar que se realizó a partir de dos modelos que son: el primero la carpa y el segundo las mesas, las cuales simplemente están dibujadas el número de veces que aparecen.

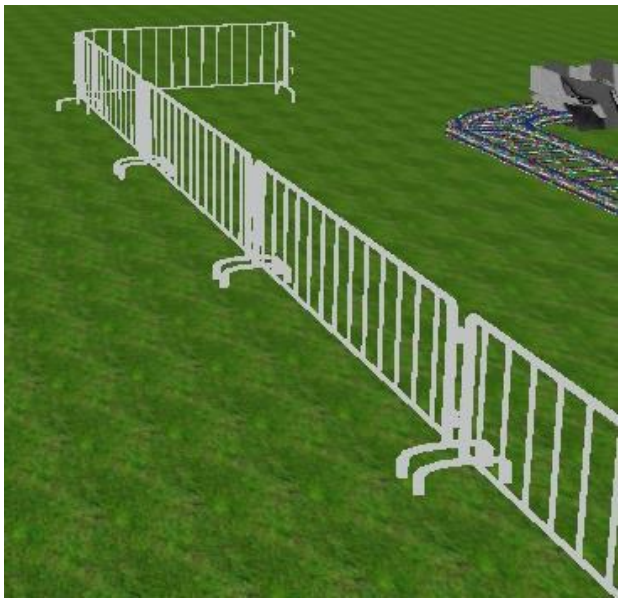
Árbol

De igual forma este modelo no cuenta con animación, pero se compone de un solo modelo.

Con las siguientes líneas código se llaman los shader adecuados para poder dibujar los modelos declarados anteriormente.

```
1826    //------------------------arboles-----------------------------------
1827    model = glm::translate(tmp, glm::vec3(-3.0f, 5.7f, -20.0)); // a la derecha de las canastas
1828    model = glm::scale(model, glm::vec3(1.5f, 1.5f, 1.5f));
1829    model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
1830    shader.setMat4("model", model);
1831    arbol1.Draw(shader);//arbol grande
1832
1833    model = glm::translate(tmp, glm::vec3(10.0f, 5.7f, 12.0)); // a la derecha de las canastas
1834    model = glm::scale(model, glm::vec3(1.5f, 1.5f, 1.5f));
1835    model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
1836    shader.setMat4("model", model);
1837    arbol1.Draw(shader);//arbol grande
1838
1839    model = glm::translate(tmp, glm::vec3(-45.0f, 5.7f, 60.0)); // a lado de los dardos
1840    model = glm::scale(model, glm::vec3(1.5f, 1.5f, 1.5f));
1841    model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
1842    shader.setMat4("model", model);
1843    arbol1.Draw(shader);//arbol grande
1844
1845    model = glm::translate(tmp, glm::vec3(-5.0f, 5.7f, 60.0)); // a la derecha de las canastas
1846    model = glm::scale(model, glm::vec3(1.5f, 1.5f, 1.5f));
1847    model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
1848    shader.setMat4("model", model);
1849    arbol1.Draw(shader);//arbol grande
1850
1851    model = glm::translate(tmp, glm::vec3(-72.0f, 5.7f, -37.0)); // a la izquierda de los baños
1852    model = glm::scale(model, glm::vec3(1.5f, 1.5f, 1.5f));
1853    model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
1854    shader.setMat4("model", model);
1855    arbol1.Draw(shader);//arbol grande
1856
```

Reja de seguridad



Estas rejas son las que se encuentran resguardado la montaña rusa, para darle realismo y esta no cuenta con animación pues no es necesaria.

Con las siguientes líneas código se llaman los shader adecuados para poder dibujar los modelos declarados anteriormente.

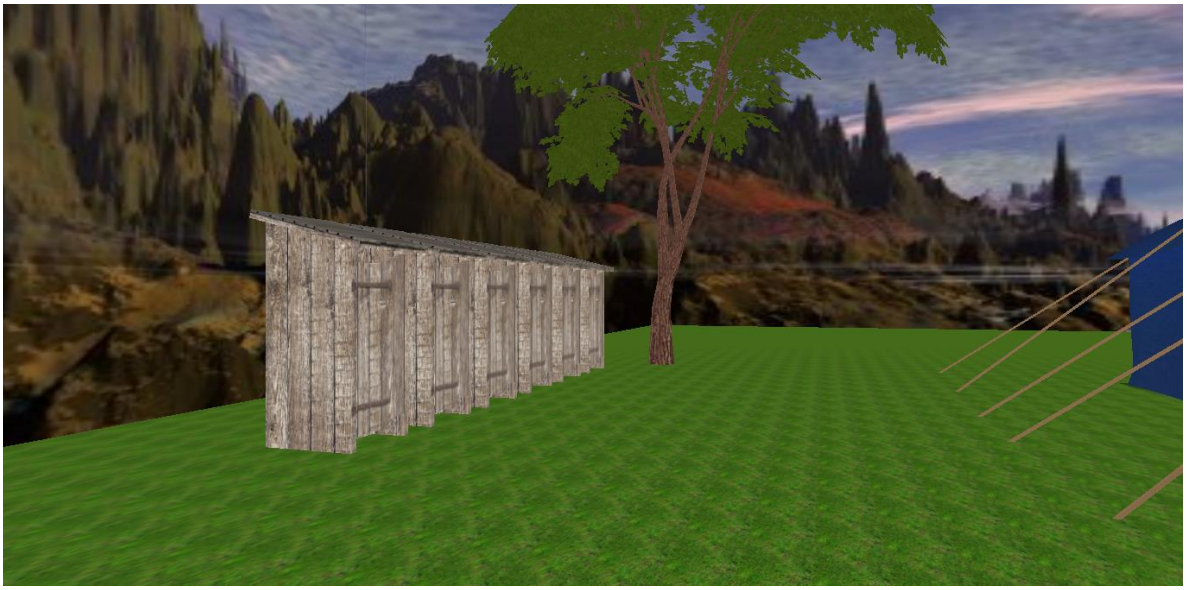
```
969     //--------------------barrera para montaña rusa-----------------------------
970     model = glm::translate(tmp, glm::vec3(35.0f, 5.65f, -30.0));
971     model = glm::scale(model, glm::vec3(0.4f, 0.2f, 0.5f));
972     model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
973
974     //model = glm::rotate(model, glm::radians(rot_llanta), glm::vec3(1.0f, 0.0f, 0.0f));
975     shader.setMat4("model", model);
976     barrera.Draw(shader);
977
978     model = glm::translate(model, glm::vec3(-11.4f, 0.0f, 0.0));
979     shader.setMat4("model", model);
980     barrera.Draw(shader);
981
982     model = glm::translate(model, glm::vec3(22.8f, 0.0f, 0.0));
983     shader.setMat4("model", model);
984     barrera.Draw(shader);
985
986     model = glm::translate(model, glm::vec3(11.4f, 0.0f, 0.0));
987     shader.setMat4("model", model);
988     barrera.Draw(shader);
989
990     model = glm::translate(model, glm::vec3(11.4f, 0.0f, 0.0));
991     shader.setMat4("model", model);
992     barrera.Draw(shader);
993
994     model = glm::translate(model, glm::vec3(11.4f, 0.0f, 0.0));
995     shader.setMat4("model", model);
996     barrera.Draw(shader);
997
998     model = glm::translate(model, glm::vec3(11.4f, 0.0f, 0.0));
999     shader.setMat4("model", model);
1000    barrera.Draw(shader);
1001
```

Este modelo no cuenta con animación, pero le da bastante realismo a la feria.

Con las siguientes líneas código se llaman los shader adecuados para poder dibujar los modelos declarados anteriormente.

```
1457    //----------------------------basketball-----------------------------------------------
1458    model = glm::translate(tmp, glm::vec3(-48.0f, 5.5f, 27.0));
1459    model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
1460    model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
1461    shader.setMat4("model", model);
1462    basketball.Draw(shader);
1463
1464    model = glm::translate(model, glm::vec3(0.0f, 0.0f, 15.0));
1465    shader.setMat4("model", model);
1466    basketball.Draw(shader);
1467
1468    model = glm::translate(model, glm::vec3(0.0f, 0.0f, 15.0));
1469    shader.setMat4("model", model);
1470    basketball.Draw(shader);
1471
1472    model = glm::translate(model, glm::vec3(0.0f, 0.0f, 15.0));
1473    shader.setMat4("model", model);
1474    basketball.Draw(shader);
1475
1476    model = glm::translate(model, glm::vec3(0.0f, 0.0f, 15.0));
1477    shader.setMat4("model", model);
1478    basketball.Draw(shader);
1479
1480    model = glm::translate(model, glm::vec3(-10.0f, 0.0f, 0.0));
1481    model = glm::rotate(model, glm::radians(-180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
1482    shader.setMat4("model", model);
1483    basketball.Draw(shader);
1484
1485    model = glm::translate(model, glm::vec3(0.0f, 0.0f, 15.0));
1486    shader.setMat4("model", model);
1487    basketball.Draw(shader);
1488
```

Baños.



Con las siguientes líneas código se llaman los shader adecuados para poder dibujar los modelos declarados anteriormente.

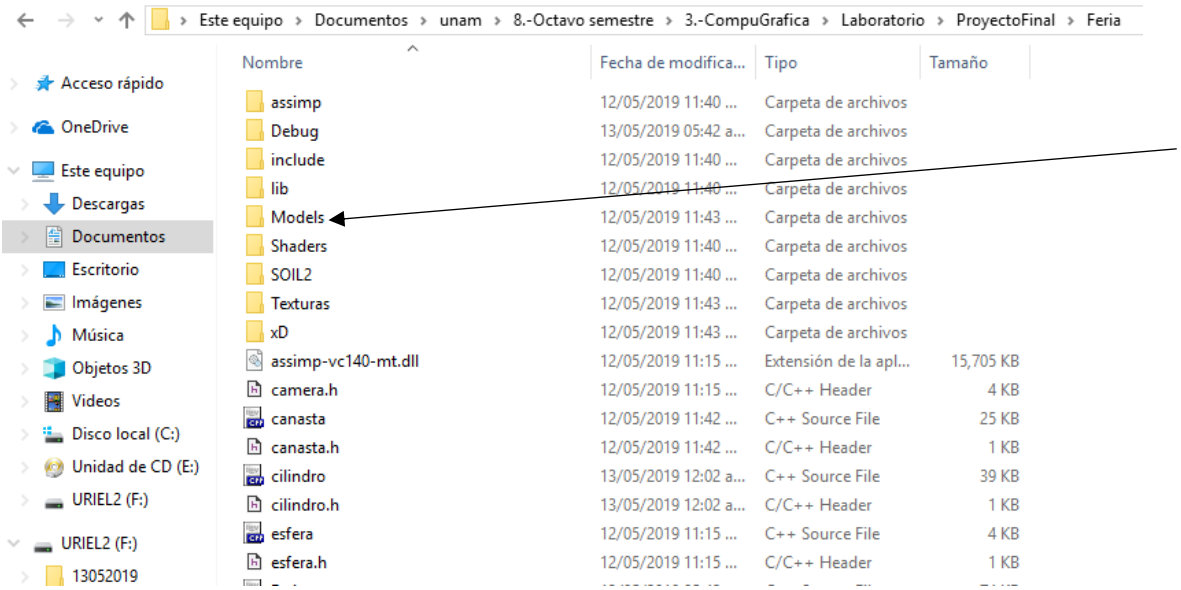
```
1909    //---------------------------------baños-------------------------------------------------
1910    model = glm::translate(tmp, glm::vec3(-75.0f, 5.68f, -20.0));//a la do de las tasas
1911    model = glm::scale(model, glm::vec3(0.6f, 0.6f, 0.6f));
1912    model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
1913    shader.setMat4("model", model);
1914    toilet.Draw(shader);
1915
1916    model = glm::translate(model, glm::vec3(10.0f, 0.0f, 0.0));//a la do de las tasas
1917    shader.setMat4("model", model);
1918    toilet.Draw(shader);
1919
1920    model = glm::translate(model, glm::vec3(10.0f, 0.0f, 0.0));//a la do de las tasas
1921    shader.setMat4("model", model);
1922    toilet.Draw(shader);
1923
1924    model = glm::translate(tmp, glm::vec3(-75.0f, 5.68f, 20.0));//a la do de las tasas
1925    model = glm::scale(model, glm::vec3(0.6f, 0.6f, 0.6f));
1926    model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
1927    shader.setMat4("model", model);
1928    toilet.Draw(shader);
1929
1930    model = glm::translate(model, glm::vec3(-10.0f, 0.0f, 0.0));//a la do de las tasas
1931    shader.setMat4("model", model);
1932    toilet.Draw(shader);
1933
1934    model = glm::translate(model, glm::vec3(-10.0f, 0.0f, 0.0));//a la do de las tasas
1935    shader.setMat4("model", model);
1936    toilet.Draw(shader);
```

Tiro al blanco.

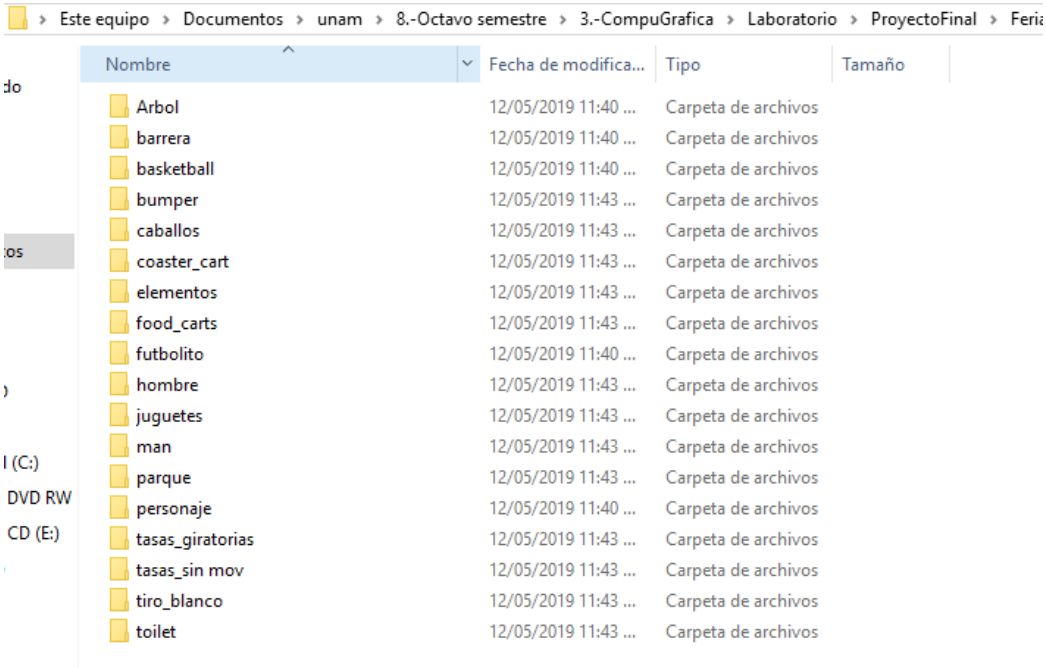
Con las siguientes líneas código se llaman los shader adecuados para poder dibujar los modelos declarados anteriormente.

```
1540        //premios
1541        model = temp2;
1542        model = glm::translate(model, glm::vec3(-140.0f, 2.5f, -30.0));
1543        model = glm::scale(model, glm::vec3(3.00f, 4.0f, 4.00f));
1544        //model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
1545        shader.setMat4("model", model);
1546        premios.Draw(shader);
1547
1548        model = glm::translate(model, glm::vec3(0.0f, 0.0f, 8.0));
1549        shader.setMat4("model", model);
1550        premios.Draw(shader);
1551
1552        model = glm::translate(model, glm::vec3(0.0f, 0.0f, 8.0));
1553        shader.setMat4("model", model);
1554        premios.Draw(shader);
1555
1556        model = temp2;
1557        model = glm::translate(model, glm::vec3(100.0f, 2.5f, -30.0));
1558        model = glm::scale(model, glm::vec3(3.00f, 4.0f, 4.00f));
1559        model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
1560        shader.setMat4("model", model);
1561        premios.Draw(shader);
1562
1563        model = glm::translate(model, glm::vec3(0.0f, 0.0f, -8.0));
1564        shader.setMat4("model", model);
1565        premios.Draw(shader);
1566
1567        model = glm::translate(model, glm::vec3(0.0f, 0.0f, -8.0));
1568        shader.setMat4("model", model);
1569        premios.Draw(shader);
1570        //-------------------------------------------------------------------
1571
```

Los modelos anteriores se encuentran en la carpeta de Models que se encuentra dentro del proyecto esto se realiza para poder mandar a llamarlos cuando se utilicen dentro del código.

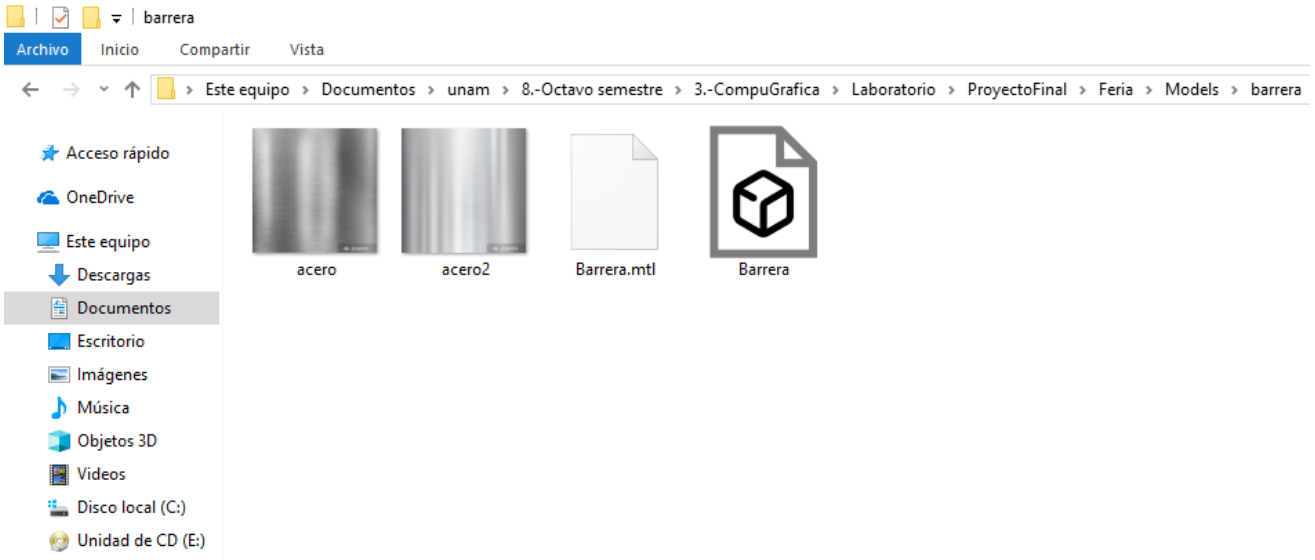


Donde dentro de la carpeta observamos los modelos:

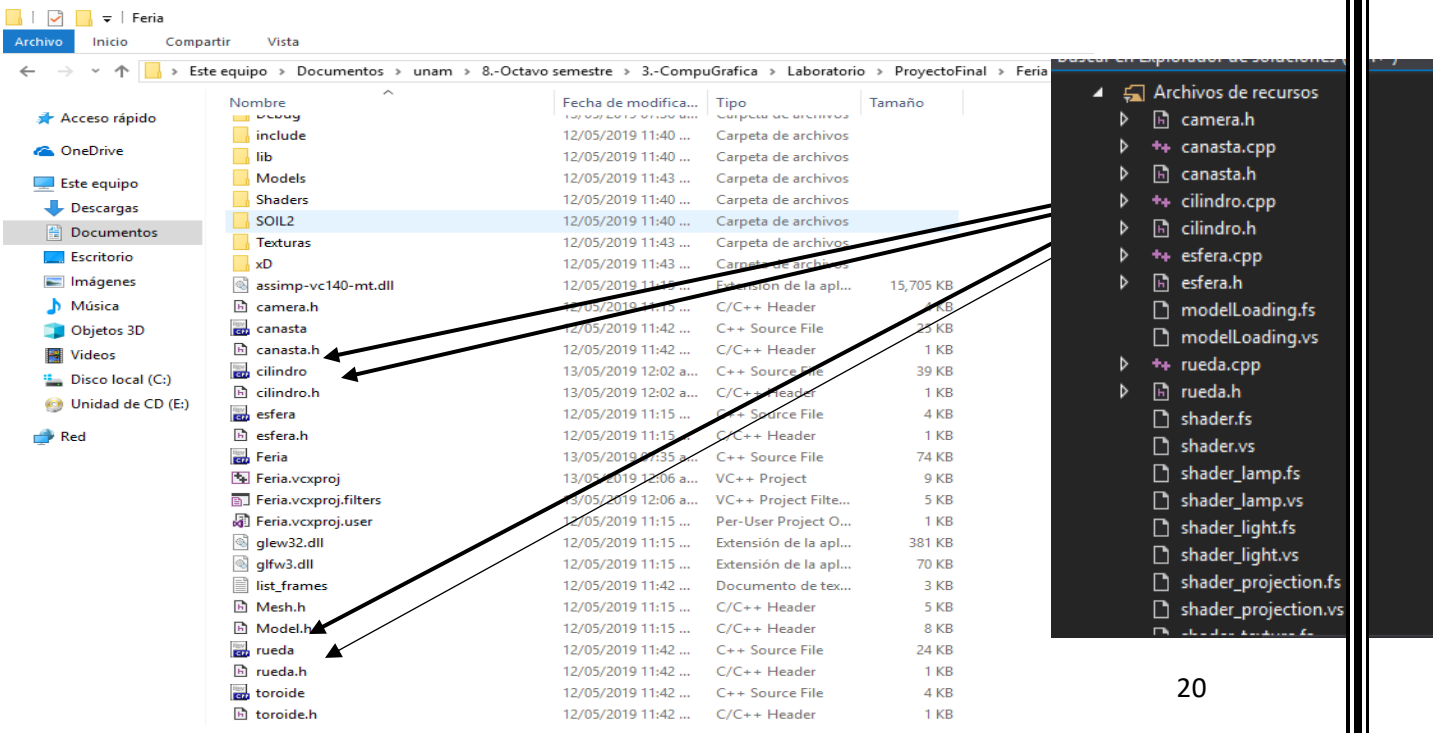


Dentro de estas carpetas se encuentran los modelos y las texturas correspondientes a cada una así como el archivo donde se controla cada textura que se usara en el modelo, es importante que se especifique la ruta donde se carga cada modelo, debido a que si no se especifica la ruta y se declara un modelo en una ruta donde ese modelo no existe, causara un error en el código.

Aquí solo se visualizará dentro de una carpeta de modelo para poder visualizar los archivos correspondientes, pero en general las demás carpetas de modelos se encuentran los mismos archivos.



primitivas y son lo que se conoce como clases que generan objetos y estos objetos simplemente llaman a métodos que son los que se encargan de dibujar o hacer una función en específico, estas “clases” se agregan a los archivos de recursos y se colocan físicamente en una carpeta como se muestra a continuación.



Dentro de estos archivos punto h y punto cpp se encuentran los métodos antes mencionados que se encargan de dibujar estos modelos y así por medio de estas funciones simplifican mucho el código, así como la optimización del programa, y la reutilización de este, a continuación, se muestra los ejemplos de las funciones que realizan las “Clases antes mencionadas”

Por ejemplo, para este caso se encuentra en el archivo de cilindro punto cpp, la función de roller coaster, la cual se encarga de dibujar la montaña rusa.

cilindro.cpp

Feria (Ámbito global)

```
178
179  void Cilindro::roller_coaster(Shader projectionShader)
180  {
181      //Shader projectionShader("shaders/shader_light.vs", "shaders/shader_light.fs");//ponerlo como parametro para evita
182      projectionShader.use();
```

rueda.cpp cilindro.cpp

Feria (Ámbito global)

```
265
266  void Rueda::dibujaRueda(unsigned int tex1, Shader ligthShader, Shader textureShader, glm::mat4 model, GLuint rueda_VBO, GLuint ru
267  {
268      my_cylinder2.init();
269      my_toroide2.init();
270      Shader projectionShader("shaders/shader_light.vs", "shaders/shader_light.fs");
271      //glBindVertexArray(lightVAO);
272      //glEnable(GL_CULL_FACE);
```

Y en el archivo de rueda.cpp se encuentra el de dibuja rueda que se encarga de dibujar la rueda en el lugar y posición especificada, esta recibe parámetros de

Como se puede observar estas funciones o métodos reciben parámetros los cuales son útiles para poder dibujar el modelo en este caso y para poder optimizar el código, así como la fácil manipulación de estos modelos

Modelo de montaña rusa.

Este modelo cuenta con animación por key frame y se puede activar con la tecla v, para poder realizar el recorrido sobre la montaña.

Primitivas de la montaña rusa.

Como se observó anteriormente las primitivas correspondientes a la montaña rusa eran el cilindro, pero dentro de estas son la base de lo que es la estructura total de como permite dibujar, dentro de cilindro cpp, se encuentra la inicialización de los vértices para poder dibujar el cilindro y se basa en las formulas que le da para cada eje x, y, z, que se muestra en la siguiente captura, esta es la función que permite trazar los vértice para posteriormente dibujar el cilindro y poder usarlo como una primitiva.

```
void Cilindro::init()
{
    const int nn = PARALELOS * MERIDIANOS * 3;
    GLfloat cylinder_nor[nn]; // normal
    // generate the sphere data
    GLfloat x, y, z, a, b, da, db, r = 1.0;
    int ia, ib, ix, iy;
    da = (GLfloat)2.0*M_PI / GLfloat(MERIDIANOS);
    db = (GLfloat)M_PI / GLfloat(PARALELOS - 1);

    // [Generate sphere point data]

    // spherical angles a,b covering whole sphere surface
    for (ix = 0, b = (GLfloat)-0.5*M_PI, ib = 0; ib < PARALELOS; ib++, b += db)
        for (a = 0.0, ia = 0; ia < MERIDIANOS; ia++, a += da, ix += 3)
        {
            // unit cylinder

            x = b;
            y = sin(a);
            z = cos(a);
            cilindro_pos[ix + 0] = x * r;
            cilindro_pos[ix + 1] = y * r;
            cilindro_pos[ix + 2] = z * r;
            cylinder_nor[ix + 0] = x;
            cylinder_nor[ix + 1] = y;
            cylinder_nor[ix + 2] = z;
        }
```

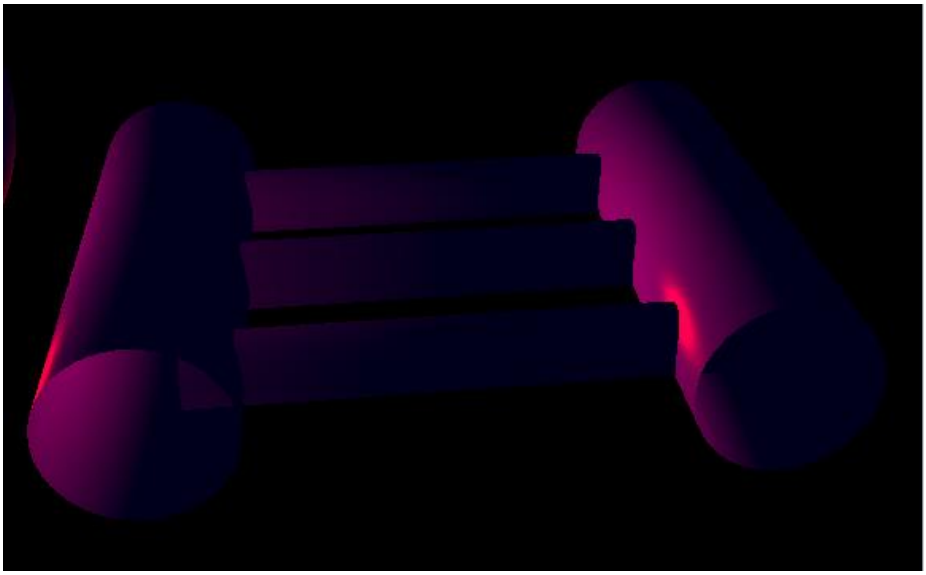
Posterior a esta función se encuentra cilindro.render que es la que manda a dibujar el cilindro una vez iniciados los vértices.

```
void Cilindro::render()
{
    //glEnable(GL_CULL_FACE);
    //glFrontFace(GL_CCW);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glBindVertexArray(cilindro_VAO[0]);
    //  glDrawArrays(GL_POINTS,0,sizeof(esfera_pos)/sizeof(GLfloat));                // POINTS ... no i
    glDrawElements(GL_TRIANGLES, sizeof(cilindro_index) / sizeof(GLuint), GL_UNSIGNED_INT, 0);    // indi
    glBindVertexArray(0);
}
```

Posterior a esta se encuentra la función de cilindro punto riel que es la que dibuja un riel de tres “durmientes” y las vias, esto es para facilitar la construcción y solo dibujar el riel el numero de veces que se requiere, se pasa el shader y model para poder identificar donde se ubica, sino se tendría que llamar n veces este shader y la ubicación del riel donde se quiere dibujar se complicaría.

```
void Cilindro::riel(glm::mat4 model, Shader projectionShader)
{
    //Shader projectionShader("shaders/shader_light.vs", "shaders/shader_light.fs");
    projectionShader.use();
    //glBindVertexArray(lightVAO);
    //glEnable(GL_CULL_FACE);
    //glFrontFace(GL_CCW);
    glm::mat4 temp01 = glm::mat4(1.0f); //Temp
    glm::mat4 temp02 = glm::mat4(1.0f); //Temp

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glBindVertexArray(cilindro_VAO[0]);
    //  glDrawArrays(GL_POINTS,0,sizeof(esfera_pos)/sizeof(GLfloat));     // POINTS ... no indices for debug
    temp01 = model;
    temp02 = model;

    model = temp02;
    model = glm::scale(model, glm::vec3(1.0f, 0.5f, 0.5f));
    projectionShader.setMat4("model", model);

    glDrawElements(GL_TRIANGLES, sizeof(cilindro_index) / sizeof(GLuint), GL_UNSIGNED_INT, 0);    // indices (choose just one line n

    model = glm::translate(model, glm::vec3(0.0, 0, 6.5));
    projectionShader.setMat4("model", model);
    glDrawElements(GL_TRIANGLES, sizeof(cilindro_index) / sizeof(GLuint), GL_UNSIGNED_INT, 0);
```

Básicamente lo que dibuja esta función cada que se llama es el siguiente modelo donde se puede escalar, trasladar y rotar.



Posterior a esto se encuentra la funcion `roller_coaster` que usa a riel para poder dibujar toda la estructura de la montaña rusa como tal.

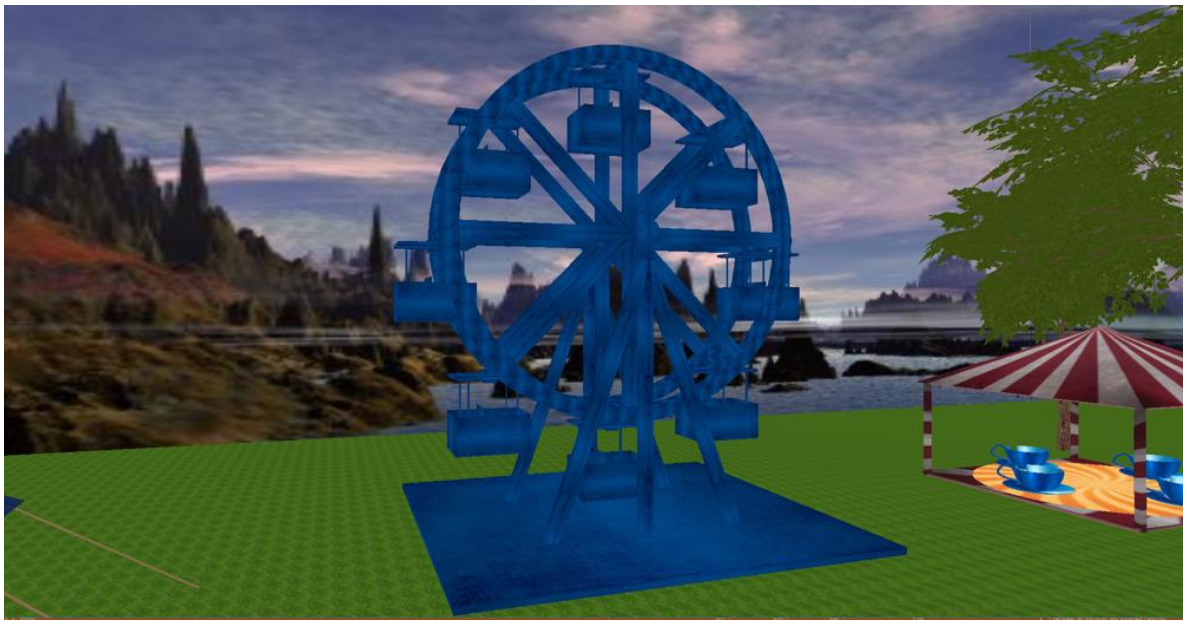
```
void Cilindro::roller_coaster(Shader projectionShader)
{
    //Shader projectionShader("shaders/shader_light.vs", "shaders/shader_light.fs");//ponerlo como parametro para
    projectionShader.use();
    //glBindVertexArray(lightVAO);
    //glEnable(GL_CULL_FACE);
    //glFrontFace(GL_CCW);
    glm::mat4 model = glm::mat4(1.0f);      // initialize Matrix, Use this matrix for individual models
    glm::mat4 temp04 = glm::mat4(1.0f);//Temp
    glm::mat4 temp03 = glm::mat4(1.0f);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glBindVertexArray(cilindro_VAO[0]);
    //  glDrawArrays(GL_POINTS,0,sizeof(esfera_pos)/sizeof(GLfloat));     // POINTS ... no indices for debug

    model = glm::mat4(1.0f);
    model = glm::translate(model, glm::vec3(-30.0, -4.15, -70.0));//era -4.0
    //model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0, 1, 0));
    model = glm::scale(model, glm::vec3(0.3f, 0.2f, 0.2f));
    temp04 = model;
    projectionShader.setMat4("model", model);
    //projectionShader.setVec3("ambientColor", 0.0f, 0.0f, 1.0f);
    //projectionShader.setVec3("diffuseColor", 0.6f, 0.0f, 0.3f);
    //projectionShader.setVec3("specularColor", 1.0f, 0.0f, 0.0f);

    Cilindro::riel(model, projectionShader);

    for (int i = 0; i < 11; i++) {
        model = glm::translate(model, glm::vec3(3.14, 0, 0));
        projectionShader.setMat4("model", model);
        Cilindro::riel(model, projectionShader);
    }
```

Como se mencionó usa esta función, para dibujar la montaña rusa con el rial antes mostrado, simplemente, traslada, rota y escala el riel n veces hasta completar la estructura de la montaña rusa que se mostró anteriormente.

```
//inicia curva de subida

model = glm::translate(model, glm::vec3(2.0, 0.3, 0.0));
model = glm::rotate(model, glm::radians(-30.0f), glm::vec3(0, 0, -1));
model = glm::scale(model, glm::vec3(0.5f, 1.0f, 1.0f));
projectionShader.setMat4("model", model);

Cilindro::riel(model, projectionShader);

model = glm::translate(model, glm::vec3(3.0, 0.4, 0.0));
model = glm::rotate(model, glm::radians(-15.0f), glm::vec3(0, 0, -1));
projectionShader.setMat4("model", model);
Cilindro::riel(model, projectionShader);

//model = glm::translate(model, glm::vec3(0, 0.55, 4.6));
model = glm::translate(model, glm::vec3(6.15, 0.81, 0));
model = glm::rotate(model, glm::radians(-10.0f), glm::vec3(0, 0, -1));
model = glm::scale(model, glm::vec3(3.0f, 1.0f, 1.0f));
projectionShader.setMat4("model", model);

Cilindro::riel(model, projectionShader);
```

Como en la anterior captura se muestra como es que se manda a llamar la función riel dentro de `roller_coaster` que se coloca estratégicamente, para formar la estructura de la montaña. Como es un modelo por primitiva en un archivo punto cpp, cuando se manda a llamar, toda la estructura se escala, rota y modela, en una sola línea, y no se tiene que hacer cada una de los rieles que la conforman.

Modelo de rueda de la fortuna.



Este modelo activa su animación con la tecla R.

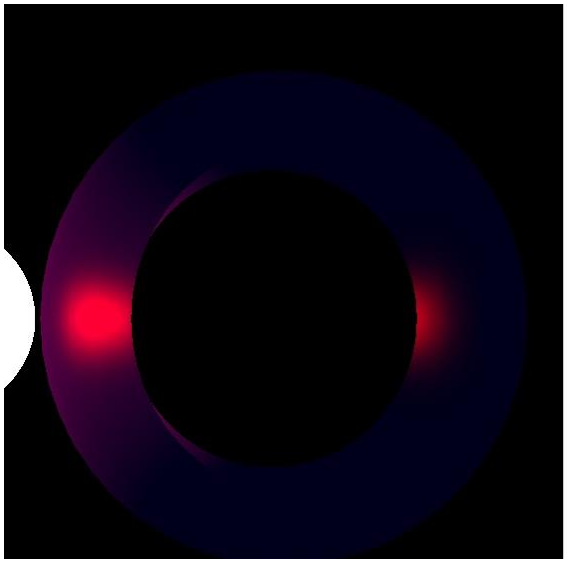
Primitivas de la rueda.

```
void Toroide::init()
{
    const int nn = PARALELOS * MERIDIANOS * 3;
    GLfloat toroider_nor[nn]; // normal
    // generate the sphere data
    GLfloat x, y, z, a, b, da, db, r = 1.0;
    int ia, ib, ix, iy;
    da = (GLfloat)2.0*M_PI / GLfloat(MERIDIANOS);
    db = (GLfloat)M_PI / GLfloat(PARALELOS - 1);

    // [Generate sphere point data]

    // spherical angles a,b covering whole sphere surface
    for (ix = 0, b = (GLfloat)-0.5*M_PI, ib = 0; ib < PARALELOS; ib++, b += db)
        for (a = 0.0, ia = 0; ia < MERIDIANOS; ia++, a += da, ix += 3)
        {
            // unit cylinder

            x = cos(a)*(10+cos(b));
            y = sin(a)*(10+cos(b));
            z = sin(b);
            toroide_pos[ix + 0] = x * r;
            toroide_pos[ix + 1] = y * r;
            toroide_pos[ix + 2] = z * r;
            toroider_nor[ix + 0] = x;
            toroider_nor[ix + 1] = y;
            toroider_nor[ix + 2] = z;
        }
    // [Generate GL_TRIANGLE indices]
```

De igual forma el toroide contiene a la rueda de la fortuna, la función init inicia los vértices que se calculan por fórmulas matemáticas que dibujan el toroide.

```
void Toroide::render()
{
    //glEnable(GL_CULL_FACE);
    //glFrontFace(GL_CCW);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glBindVertexArray(toroide_VAO[0]);
    //  glDrawArrays(GL_POINTS,0,sizeof(esfera_pos)/sizeof(GLfloat));                    // POINTS
    glDrawElements(GL_TRIANGLES, sizeof(toroide_index) / sizeof(GLuint), GL_UNSIGNED_INT, 0);
    glBindVertexArray(0);
}
```

Con toroide punto render dibuja un toroide.



Con toroide render se dibuja el toroide, el cual se puede rotar, trasladar y escalar.

En el archivo rueda cpp se usa tanto la funcion de toroide como de cilindtro para formar la rueda.

rueda.cpp | cilindro.cpp | Feria.cpp*

Feria | (Ámbito global)

```
1   #include "rueda.h"
2   #include"canasta.h"
3   #include"toroide.h"
4   #include "cilindro.h"
5
6   //glm::vec3 lightPos(0.0f, 0.0f, 0.0f);
7   //GLuint VBO, VAO, lightVAO;
8    Canasta my_canasta1(1.0f);
9    Rueda my_rueda1(1.0f);
10   Cilindro my_cylinder2(1.0f);
11   Toroide my_toroide2(1.0f);
12  //Camera camera(glm::vec3(0.0f, 0.0f, 3.0f));
13   //int SCR_WIDTH = 3800;
14   //int SCR_HEIGHT = 7600;
```

Como se puede observar estos se declaran para su posterior uso, esto se hace para que la rueda se pueda trasladar, rotar y escalar toda la estructura completa. Y no se tenga que declarar y llamar varias veces las mismas líneas de código, como las que se usan para dibujar las canastas de la rueda de la fortuna.

En la función dibuja rueda para poder dibujar toda la estructura de la rueda.

```
void Rueda::dibujaRueda(unsigned int tex1, Shader ligthShader, Shader textureShader, glm::mat4 model, GLuint rueda_VBO, GLuint rueda_V
{
    my_cylinder2.init();
    my_toroide2.init();
    Shader projectionShader("shaders/shader_light.vs", "shaders/shader_light.fs");
    //glBindVertexArray(lightVAO);
    //glEnable(GL_CULL_FACE);
    //glFrontFace(GL_CCW);
    //glm::mat4 model = glm::mat4(1.0f); //Temp
    glm::mat4 temp01 = glm::mat4(1.0f); //Temp
    glm::mat4 temp02 = glm::mat4(1.0f); //Temp
    glm::mat4 temp03 = glm::mat4(1.0f); //Temp
    //glm::mat4 projection = glm::mat4(1.0f);   //This matrix is for Projection
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glBindVertexArray(rueda_VAO);
    //model = glm::mat4(1.0f);
    //model = glm::rotate(model, glm::radians(90.0f), glm::vec3(1, 0, 0));
    //------------Rueda de la fortuna --------
    //-----------------Soporte-----------------------------------
    temp01 = model;
    model = glm::scale(model, glm::vec3(0.1f, 1.2f, 0.1f));
    model = glm::rotate(model, glm::radians(90.0f), glm::vec3(1, 0, 0));
    model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0, 1, 0));
    //model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0, 1, 0));
    textureShader.setMat4("model", model);
    my_cylinder2.render();
```

En los parámetros recibe shader que facilitan la colocación de texturas de la rueda e iluminación, y los vértices declarados para poder dibujar la canasta ya que esta formada de cubos de diferentes dimensiones y con normales ya declaradas, así como la base de la rueda.

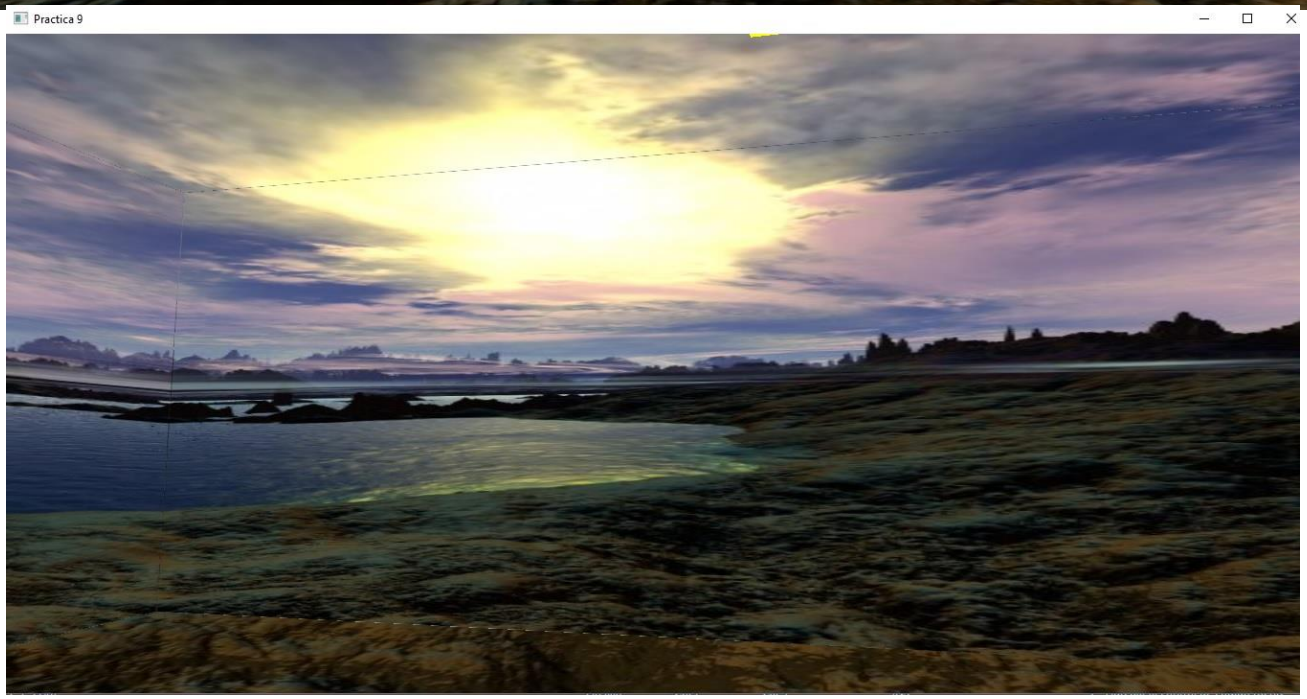
```
void Canasta::dibujaCanasta(unsigned int tex1, Shader ligthShader,Shader textureShader, glm::mat4 model, glm::mat4 temp01, GLuint cana
{
    //Shader projectionShader("shaders/shader_light.vs", "shaders/shader_light.fs");
    //glBindVertexArray(lightVAO);
    //glEnable(GL_CULL_FACE);
    //glFrontFace(GL_CCW);
    //glm::mat4 temp01 = glm::mat4(1.0f); //Temp
    glm::mat4 temp02 = glm::mat4(1.0f); //Temp
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glBindVertexArray(canasta_VAO);


    ligthShader.use();
    model = glm::translate(model, glm::vec3(0.0f, -1.7, 1.0f));
    model = glm::scale(model, glm::vec3(1.0f, 1.5f, 1.5f));
    //model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0, 1, 0));
    //textureShader.setMat4("model", model);
    ligthShader.setMat4("model", model);
    ligthShader.setInt("material_diffuse", tex1);
    //textureShader.setMat4("projection", projection);
    temp02 = model;
    /*projectionShader.setVec3("ambientColor", 0.0f, 0.0f, 1.0f);
    projectionShader.setVec3("diffuseColor", 0.6f, 0.0f, 0.3f);
    projectionShader.setVec3("specularColor", 1.0f, 0.0f, 0.0f);*/
```

La canasta se dibuja con la funcion dibujaCnasta dentro del archivo canasta punto cpp y el cual a su ves se manda a llamar igual dentro del archivo rueda cpp.

En conclusión rueda.cpp usa a toroide.cpp, canasta.cpp y cilindro cpp para la construcción adecuada de su estructura y cuando se manda a llamar rueda cpp simplemente basta con la llamada a su función dibujaRueda, para poder dibujar la rueda en la opción, Angulo y escala correspondiente.

SkyBox

Con el fin de mostrar el sky box de todo el lugar se muestran las siguientes imágenes donde son capturas de pantalla del sky box en ejecución que se muestra desde diferentes ángulos, como se muestra a continuación estas son imágenes que se colocan dentro de una caja de dimensiones grandes superiores al proyecto de la feria,



Dentro del proyecto se pueden mover y explorar a lo largo de las dimensiones del skyBox con las teclas indicadas en el control y el movimiento del mouse, para ver todos los angulos y colocarse dentro del espacio en la posición deseada o a la que se quiera recorrer o visualizar con mas detalle.

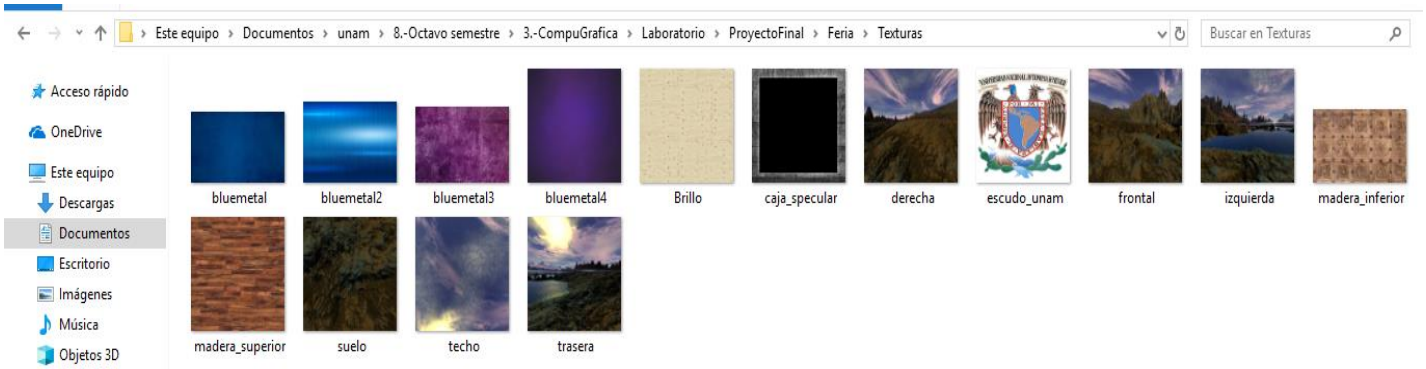
Los códigos para dibujar el skybox se usan las siguientes estructuras, para poder usar los vértices y normas declarados en la parte superior e indicados en cada uno de los siguientes for.

```
2065        //--------------------Sky Box--------------------
2066        lightingshader.setVec3("ambientColor", 1.0f, 1.0f, 1.0f);
2067        lightingshader.setVec3("diffuseColor", 0.8f, 0.8f, 0.8f);
2068        lightingshader.setVec3("specularColor", 1.0f, 1.0f, 1.0f);
2069        lightingshader.setInt("material_diffuse", 5);//parte trasera
2070        lightingshader.setMat4("model", model);
2071        //glDrawArrays(GL_QUADS, 24, 4);
2072
2073        for (int i = 0; i < 4; i++)
2074        {
2075            glDrawArrays(GL_TRIANGLE_FAN, 24, i + 1);
2076        }
2077
2078
2079        //lightingShader.setMat4("model", model);
2080        lightingshader.setInt("material_diffuse", 4);//Parte frontal
2081        //glDrawArrays(GL_QUADS, 28, 4);
2082        for (int i = 0; i < 4; i++)
2083        {
2084            glDrawArrays(GL_TRIANGLE_FAN, 28, i + 1);
2085        }
2086        //lightingShader.setMat4("model", model);
2087        lightingshader.setInt("material_diffuse", 7);//parte inferior
2088        //glDrawArrays(GL_QUADS, 40, 4);
2089        for (int i = 0; i < 4; i++)
2090        {
2091            glDrawArrays(GL_TRIANGLE_FAN, 40, i + 1);
2092        }
2093        //lightingShader.setMat4("model", model);
2094        lightingshader.setInt("material_diffuse", 6);//parte superior
2095        //glDrawArrays(GL_QUADS, 44, 4);
2096        for (int i = 0; i < 4; i++)
2097        {
2098            glDrawArrays(GL_TRIANGLE_FAN, 44, i + 1);
2099        }
```

```
2100        //lightingShader.setMat4("model", model);
2101        lightingshader.setInt("material_diffuse", 9);//parte izquierda
2102        /*Como aqui queremos agregar el material especular se lo pasamos al shader y como parametros nuestra textura
2103        que sera el especular, en esta parte en especifico lo tendral los dos costados, por lo que solo pasamos como parametro y
2104        dibujamos*/
2105        lightingshader.setInt("material_specular", 9);
2106        //glDrawArrays(GL_QUADS, 32, 4);
2107        for (int i = 0; i < 4; i++)
2108        {
2109            glDrawArrays(GL_TRIANGLE_FAN, 32, i + 1);
2110        }
2111
2112        lightingshader.setInt("material_diffuse", 8);
2113        lightingshader.setInt("material_specular", 10);
2114        //glDrawArrays(GL_QUADS, 36, 4);
2115        for (int i = 0; i < 4; i++)
2116        {
2117            glDrawArrays(GL_TRIANGLE_FAN, 36, i + 1);
2118        }
2119
2120        //-----------------Sky box---------------------
2121
```

Texturas.

Las texturas que se usan en este proyecto se definen por las de modelos que se encuentran especificadas en la sección de modelos exportados de este documento y las otras que están definidas en las carpetas de texturas.



Para después iniciarlas y utilizarlas adecuadamente estas se usan en elementos de modelos hechos con primitivas y por ejemplo par

```
293  void LoadTextures()
294  {
295      t_unam = generateTextures("Texturas/escudo_unam.png", 1);
296      t_caja_brillo = generateTextures("Texturas/caja_specular.png", 1)
297      t_coaster = generateTextures("Texturas/bluemetal.jpg", 0);
298      frontal = generateTextures("Texturas/frontal.jpg", 0);
299      trasera = generateTextures("Texturas/trasera.jpg", 0);
300      superior = generateTextures("Texturas/techo.jpg", 0);
301      inferior = generateTextures("Texturas/suelo.jpg", 0);
302      derecho = generateTextures("Texturas/derecha.jpg", 0);
303      izquierdo = generateTextures("Texturas/izquierda.jpg", 0);
304      brillo = generateTextures("Texturas/brillo.jpg", 0);
305  }
```

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, 0);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, t_unam);
glActiveTexture(GL_TEXTURE2);
glBindTexture(GL_TEXTURE_2D, t_caja_brillo);
glActiveTexture(GL_TEXTURE3);
glBindTexture(GL_TEXTURE_2D, t_coaster);
glActiveTexture(GL_TEXTURE4);
glBindTexture(GL_TEXTURE_2D, frontal);
glActiveTexture(GL_TEXTURE5);
glBindTexture(GL_TEXTURE_2D, trasera);
glActiveTexture(GL_TEXTURE6);
glBindTexture(GL_TEXTURE_2D, superior);
glActiveTexture(GL_TEXTURE7);
glBindTexture(GL_TEXTURE_2D, inferior);
glActiveTexture(GL_TEXTURE8);
glBindTexture(GL_TEXTURE_2D, derecho);
glActiveTexture(GL_TEXTURE9);
glBindTexture(GL_TEXTURE_2D, izquierdo);
glActiveTexture(GL_TEXTURE10);
glBindTexture(GL_TEXTURE_2D, brillo);
```

De noche es como se muestra a continuación, se puede apreciar que la iluminación se refleja en las texturas de la rueda de la fortuna y no en el sky box ya que esto le da realismo la luz común que usamos como son lámparas no ilumina todo como lo hace el sol de día, sino hasta su alcance.

Dia.

Como se puede observar de día si se ilumina todo por el sol que en nuestra simulación la iluminación ambiental es la que le da el efecto y por medio de animación podemos controlar día y noche con la letra o(amanece) y p(anochece).

Esto se puede observar de mejor forma en el video adjunto o directamente en la ejecución del programa.

Para los controles de este se usan:

```
2392    if (glfwGetKey(window, GLFW_KEY_P) == GLFW_PRESS) {//poner de noche
2393        ambient1 = 0.04f;
2394        ambient2 = 0.04f;
2395        ambient3 = 0.04f;
2396    }
2397    if (glfwGetKey(window, GLFW_KEY_O) == GLFW_PRESS) {//poner de dia
2398        ambient1 = 1.0f;
2399        ambient2 = 1.0f;
2400        ambient3 = 1.0f;
2401    }
```

```
2027        lightingshader.use();
2028
2029        lightingshader.setVec3("light.position", camera.Position);
2030        lightingshader.setVec3("light.direction", camera.Front);
2031        lightingshader.setFloat("light.cutOff", glm::cos(glm::radians(10.0f)));
2032
2033        lightingshader.setVec3("viewPos", camera.Position);
2034
2035        lightingshader.setVec3("light.ambient", ambient1, ambient2, ambient3);
2036        lightingshader.setVec3("light.diffuse", 0.4f, 0.8f, 0.8f);
2037        lightingshader.setVec3("light.specular", 0.0f, 0.0f, 2.0f);
2038        //For Positional and Spotlight
2039        lightingshader.setFloat("light.constant", 1.0f);
2040        lightingshader.setFloat("light.linear", 0.009f);
2041        lightingshader.setFloat("light.quadratic", 0.0032f);
2042
2043        // material properties
2044        lightingshader.setFloat("material_shininess", 35.0f);
2045
2046        projection = glm::perspective(glm::radians(camera.Zoom), (float)SCR_WIDTH / (float)SCR_HEIGHT, 0.1f, 500.0f);
2047        view = camera.GetViewMatrix();
2048
2049        model = glm::mat4(1.0f);
2050
2051        lightingshader.setMat4("view", view);
2052        // note: currently we set the projection matrix each frame, but since the projection matrix rarely changes it's of
2053        lightingshader.setMat4("projection", projection);
2054        lightingshader.setMat4("model", model);
```

Animación.

Como se puede observar los modelos e iluminación contaban con animación, esta se viene trabajando de dos formas en keyframe y en estados, como se vieron en laboratorio a lo largo de las practicas vistas y dependiendo de la tecla que se oprima valida una u otra animación.

Animación Carrucel.

Feria.cpp | rueda.cpp | cilindro.cpp

Feria | (Ámbito global)

```
576
577
578     void animate_horse(void) {
579         if (activate_caballos) {
580             if (vueltas_caballos == 0 || vueltas_caballos == 4) {
581                 rot_caballos += 7.0f;
582             }
583             if (vueltas_caballos > 0 && vueltas_caballos < 4) {
584                 rot_caballos += 10.0f;
585             }
586             if (vueltas_caballos == 5) {
587                 activate_caballos = false;
588                 vueltas_caballos = 0;
589             }
590             if (rot_caballos > 360.0) {
591                 rot_caballos -= 360.0f;
592                 vueltas_caballos += 1;
593             }
594             if (y_caballos < 6.0f && up_caballos) {
595                 y_caballos += 0.3f;
596                 if (y_caballos >= 6.0f) {
597                     up_caballos = false;
598                 }
599             }
600             if (y_caballos > 4.0f && !up_caballos) {
601                 y_caballos -= 0.3f;
602                 if (y_caballos <= 4.4f) {
603                     up_caballos = true;
604                 }
605             }
```

Un ejemplo de esto es la función que se definió para animar los caballos en el carrusel, pero, así como esta son varias funciones que se encargan de hacer la animación por el método correcto en el video estas se pueden apreciar mejor o directamente en la ejecución del programa.

Rueda de la fortuna.

```
819     //rueda de la fortuna
820     void animate(void)
821     {
822         if (activate_rueda) {
823             sol += 0.1f;
824             rotaRueda += 0.4f;
825             venus += 0.35f;
826             tierra += 0.3f;
827             marte += 0.25f;
828             jupiter += 0.2f;
829             saturno += 0.15f;
830             urano += 0.1f;
831             neptuno += 0.05f;
832 
833             angRotPuerta = angRotPuerta + bandera;
834             if (angRotPuerta >= 0)
835             {
836                 bandera = -0.5;
837             }
838             if (angRotPuerta <= -45)
839             {
840                 bandera = 0.5;
841             }
842         }
843     }
844 
```

Animación juego de azar.

A continuación, se muestra la animación para poder

```
845 void animate_ducks(void) {
846     if (activate_ducks) {
847         if (direction_ducks1 == 0) {
848             duck_pos1 -= 0.2f;
849         }
850         if (direction_ducks1 == 1) {
851             duck_pos1 += 0.2f;
852         }
853 
854         if (duck_pos1 <= -13.0f) {
855             direction_ducks1 = 1;
856             rotpato_inf = 0.0f;
857         }
858 
859         if (duck_pos1 >= -6.0f) {
860             direction_ducks1 = 0;
861             rotpato_inf = 180.0f;
862         }
863 
864         if (direction_ducks2 == 1) {
865             duck_pos2 += 0.2f;
866         }
867         if (direction_ducks2 == 0) {
868             duck_pos2 -= 0.2f;
869         }
870 
871         if (duck_pos2 >= -6.0f) {
872             direction_ducks2 = 0;
873             rotpato_sup = 180.0f;
874             //vueltas += 1;
875         }
876 
877         if (duck_pos2 <= -13.0f) {
878             direction_ducks2 = 1;
879             rotpato_sup = 0.0f;
880         }
```

```
883         if (duck_mov1 < 0.7f && up_ducks1) {
884             duck_mov1 += 0.1f;
885             if (duck_mov1 >= 0.7f) {
886                 up_ducks1 = false;
887             }
888         }
889 
890         if (duck_mov1 > -0.7f && !up_ducks1) {
891             duck_mov1 -= 0.1f;
892             if (duck_mov1 <= -0.7f) {
893                 up_ducks1 = true;
894             }
895         }
896 
897 
898 
899         if (duck_mov2 < 0.7f && up_ducks2) {
900             duck_mov2 += 0.1f;
901             if (duck_mov2 >= 0.7f) {
902                 up_ducks2 = false;
903             }
904         }
905 
906         if (duck_mov2 > -0.7f && !up_ducks2) {
907             duck_mov2 -= 0.1f;
908             if (duck_mov2 <= -0.7f) {
909                 up_ducks2 = true;
910             }
911         }
912 
913         if (duck_pos1 == -9.5f) {
914             vueltas += 1;
915         }
916         if (vueltas == 5) {
917             activate_ducks = false;
918             vueltas = 0;
919         }
920 
921 
922     }
923 }
```

Animación de las tasas.

```
void animated_tasas(void) {
    if (activate_tasas) {
        if (vueltas_tasas == 0 || vueltas_tasas == 6) {
            rot_base_tasas += 4.0f;
            rot_tasas += 6.0f;
        }
        if ((vueltas_tasas >= 1 && vueltas_tasas < 2) || (vueltas_tasas >= 5 && vueltas_tasas < 6)) {
            rot_base_tasas += 6.0f;
            rot_tasas += 8.0f;
        }
        if (vueltas_tasas >= 2 && vueltas_tasas <= 4) {
            rot_base_tasas += 10.0f;
            rot_tasas += 12.0f;
        }

        if (rot_base_tasas > 360.0) {
            rot_base_tasas -= 360.0f;
            vueltas_tasas += 1;
        }
        if (rot_tasas > 360.0) {
            rot_tasas -= 360.0f;
        }

        if (vueltas_tasas == 7) {
            activate_tasas = false;
            vueltas_tasas = 0;
        }
    }
}
```

Animación caros chocones.

```
void animate_cc(void) {
    if (activate_cc) {
        if (estado_cc == 0) {
            rot_comida = -90.0f;
            if (x_comida >= -30.0f) {
                x_comida -= 0.3f;
            }
            else {
                estado_cc = 1;
            }
        }
        if (estado_cc == 1) {
            rot_comida = 180.0f;
            if (z_comida >= -25) {
                z_comida -= 0.3f;
            }
            else {
                estado_cc = 2;
            }
        }
        if (estado_cc == 2) {
            rot_comida = -270;
            if (x_comida <= 25) {
                x_comida += 0.3;
            }
            else {
                estado_cc = 3;
            }
        }
        if (estado_cc == 3) {
            rot_comida = 0.0f;
            if (z_comida <= 23) {
                z_comida += 0.3;
            }
            else {
                estado_cc = 0;
```

Animacion por keyFrame.

A continuación, se muestra las funciones que se usaron para poder hacer la animación de la montaña rusa por este método.

```
714   void saveFrame(void) //pasa los datos a la estructura
715   {
716
717       printf("frameindex %d\n", FrameIndex);
718       printf("x %f ", x_coaster_cart);
719       printf("y %f ", y_coaster_cart);
720       printf("z %f ", z_coaster_cart);
721       printf("angx %f ", rotx_coaster_cart);
722       printf("angy %f ", roty_coaster_cart);
723       printf("angz %f\n ", rotz_coaster_cart);
724
725       KeyFrame[FrameIndex].x_coaster_cart = x_coaster_cart; //pasa las posiciones por cada estado (guarda el estado del monito)
726       KeyFrame[FrameIndex].y_coaster_cart = y_coaster_cart;
727       KeyFrame[FrameIndex].z_coaster_cart = z_coaster_cart;
728
729       KeyFrame[FrameIndex].rotx_coaster_cart = rotx_coaster_cart;
730       KeyFrame[FrameIndex].roty_coaster_cart = roty_coaster_cart;
731       KeyFrame[FrameIndex].rotz_coaster_cart = rotz_coaster_cart;
732
733       FrameIndex++;
734   }
```

```
738   void setFrame(void) {
739       FILE * frames;
740       errno_t err;
741       if((err = fopen_s(&frames, "list_frames.txt", "rb")) != 0){
742           perror("Error al intentar abrir el archivor");
743       }
744       else {
745           while (feof(frames) == 0) {
746               fscanf_s(frames, "%f%f%f%f%f%f", &KeyFrame[FrameIndex].x_coaster_cart, &KeyFrame[FrameIndex].y_coaster_cart, &KeyFrame[FrameIndex].z_coas
747                   &KeyFrame[FrameIndex].roty_coaster_cart, &KeyFrame[FrameIndex].rotz_coaster_cart);
748               //printf("frameindex %d\n", FrameIndex);//primera parte
749               FrameIndex++;
750           }
751       }
752
753       fclose(frames);
754       //printf("archivo leido correctamente...\n");
755   }
756
```

```
760   void resetElements(void)//coloca la animacion al inicio
761   {
762       x_coaster_cart = KeyFrame[0].x_coaster_cart;
763       y_coaster_cart = KeyFrame[0].y_coaster_cart;
764       z_coaster_cart = KeyFrame[0].z_coaster_cart;
765
766       rotx_coaster_cart = KeyFrame[0].rotx_coaster_cart;
767       roty_coaster_cart = KeyFrame[0].roty_coaster_cart;
768       rotz_coaster_cart = KeyFrame[0].rotz_coaster_cart;
769   }
770
771   void interpolation(void)//interpolacion entre keyframes
772   {
773
774       KeyFrame[playIndex].xInc_coaster_cart = (KeyFrame[playIndex + 1].x_coaster_cart - KeyFrame[playIndex].x_coaster_cart) / i_max_steps;//se divide
775       KeyFrame[playIndex].yInc_coaster_cart = (KeyFrame[playIndex + 1].y_coaster_cart - KeyFrame[playIndex].y_coaster_cart) / i_max_steps;
776       KeyFrame[playIndex].zInc_coaster_cart = (KeyFrame[playIndex + 1].z_coaster_cart - KeyFrame[playIndex].z_coaster_cart) / i_max_steps;
777
778       KeyFrame[playIndex].rotxInc_coaster_cart = (KeyFrame[playIndex + 1].rotx_coaster_cart - KeyFrame[playIndex].rotx_coaster_cart) / i_max_steps;
779       KeyFrame[playIndex].rotyInc_coaster_cart = (KeyFrame[playIndex + 1].roty_coaster_cart - KeyFrame[playIndex].roty_coaster_cart) / i_max_steps;
780       KeyFrame[playIndex].rotzInc_coaster_cart = (KeyFrame[playIndex + 1].rotz_coaster_cart - KeyFrame[playIndex].rotz_coaster_cart) / i_max_steps;
781
782   }
783
```

```
784  void animate_coaster(void) {
785      if (activate_coaster) {
786          if (i_curr_steps >= i_max_steps) //end of animation between frames?
787          {
788              playIndex++;
789              if (playIndex > FrameIndex - 2) //end of total animation?
790              {
791                  printf("termina anim\n");
792                  playIndex = 0;
793                  activate_coaster = false;
794              }
795              else //Next frame interpolations
796              {
797                  i_curr_steps = 0; //Reset counter
798                                    //Interpolation
799                  interpolation();
800              }
801          }
802          else
803          {
804              //Draw animation
805              x_coaster_cart += KeyFrame[playIndex].xInc_coaster_cart;//paso actual mas el paso siguiente
806              y_coaster_cart += KeyFrame[playIndex].yInc_coaster_cart;
807              z_coaster_cart += KeyFrame[playIndex].zInc_coaster_cart;
808
809              rotx_coaster_cart += KeyFrame[playIndex].rotxInc_coaster_cart;
810              roty_coaster_cart += KeyFrame[playIndex].rotyInc_coaster_cart;
811              rotz_coaster_cart += KeyFrame[playIndex].rotzInc_coaster_cart;
812
813              i_curr_steps++;
814          }
815      }
816  }
817
```

```
686  //------------------------animacion por keyframes-------------------------------------------
687
688  #define MAX_FRAMES 60 //numero maximo de keyframes se puede modificar
689  int i_max_steps = 7; //intervalo entre cvada uno de los keyframes (interpolacion)
690  int i_curr_steps = 0; //indice para saber en que keyframe estamos
691  typedef struct _frame //definicion de la estructura
692  {
693      float x_coaster_cart;
694      float xInc_coaster_cart;
695      float y_coaster_cart;
696      float yInc_coaster_cart;
697      float z_coaster_cart;
698      float zInc_coaster_cart;
699      float rotx_coaster_cart;
700      float rotxInc_coaster_cart;
701      float roty_coaster_cart;
702      float rotyInc_coaster_cart;
703      float rotz_coaster_cart;
704      float rotzInc_coaster_cart;
705
706  }FRAME;
707
708  FRAME KeyFrame[MAX_FRAMES];
709  int FrameIndex = 0;          //introducir datos
710  //bool play = false;
711  int playIndex = 0;
712
```

Donde se manda a llamar el archivo de la lista de frames.