



Technical manual

Computer Graphics and Human Interaction
Computer

Universidad Nacional
Autónoma de México



Group 4

11/29/2023



Students:

Herrera Alcantara Emilio Ramses

Martinez Ramirez Jose Angel

Professor: Jose Roque Roman Guadarrama

Technical manual

Pinball marbles. To cover the requirement:

- 1 marble that has keyframe animation and that collides with at least 3 objects
- 1 animated marble that follows a route around the board

We use the models called *marble1* and *marble2*, being animated in the indicated manner respectively.

The first marble is activated by pressing the 9 key, which will execute the arrangement of positions in the keyframes, and when finished we can execute it again by pressing the 0 key, followed by the 9 key again; while

that the second will be done by pressing the right click and we can execute it again by pressing the same key.

For the keyframe animation, a key structure was used to manipulate the position of the

marble at runtime and thus save frames. First we have variable declarations for

manage keyframes:

```
//variables para keyframes
float reproduciranimacion, habilitaranimacion, guardoFrame, reinicioFrame, ciclo13, ciclo14, ciclo15, ciclo16, ciclo17, ciclo18, ci
```

Next, we declare more variables to control the position and movement of the marble, in addition to defining the maximum number of frames:

```
//NEW// Keyframes
float posXcanica = -4.3, posYcanica = 0.3, posZcanica = -1.65;
float movCanica_z = 0.0f, movCanica_x = 0.0f;

#define MAX_FRAMES 100
int i_max_steps = 90;
int i_curr_steps = 10;//10
typedef struct _frame
{
    //Variables para GUARDAR Key Frames
    float movCanica_z;        //Canica PosX
    float movCanica_x;        //Canica PosZ
    float movCanica_zInc;     //Canica IncX
    float movCanica_xInc;     //Canica IncZ
}FRAME;

FRAME KeyFrame[MAX_FRAMES];
int FrameIndex = 10;//10      //introducir datos
bool play = false;
int playIndex = 0;
```

A frame saving function, which adds frames to the frame array:

```

void saveFrame(void) //tecla L
{
    printf("frameindex %d\n", FrameIndex);

    KeyFrame[FrameIndex].movCanica_z = movCanica_z;
    KeyFrame[FrameIndex].movCanica_x = movCanica_x;
    //no volatil, agregar una forma de escribir a un archivo para guardar los frames
    FrameIndex++;
}

```

Also a function to reset the frames and thus view the created animation:

```

void resetElements(void) //Tecla 0
{
    movCanica_x = KeyFrame[0].movCanica_z;
    movCanica_z = KeyFrame[0].movCanica_x;
}

```

Also, the interpolation function to get from one point to another in the animation:

```

void interpolation(void)
{
    KeyFrame[playIndex].movCanica_zInc = (KeyFrame[playIndex + 1].movCanica_z - KeyFrame[playIndex].movCanica_z) / i_max_steps;
    KeyFrame[playIndex].movCanica_xInc = (KeyFrame[playIndex + 1].movCanica_x - KeyFrame[playIndex].movCanica_x) / i_max_steps;
}

```

Finally, the function to draw the created animation:

```

void animate(void)
{
    //Movimiento del objeto con barra espaciadora
    if (play)
    {
        if (i_curr_steps >= i_max_steps) //fin de animación entre frames?
        {
            playIndex++;
            printf("playindex : %d\n", playIndex);
            if (playIndex > FrameIndex - 2) //Fin de toda la animación con último frame?
            {
                printf("Frame index= %d\n", FrameIndex);
                printf("Terminó la animación\n");
                playIndex = 0;
                play = false;
            }
        }
        else //Interpolación del próximo cuadro
        {
            i_curr_steps = 0; //Resetea contador
            //Interpolar
            interpolation();
        }
    }
    else
    {
        //Dibujar Animación
        movCanica_z += KeyFrame[playIndex].movCanica_zInc;
        movCanica_x += KeyFrame[playIndex].movCanica_xInc;
        i_curr_steps++;
    }
}

```

With this function, the keys are read from the keyboard to control the animation and save frames:

```

void inputKeyframes(void)
{
    if (keys[GLFW_KEY_9])
    {
        if (reproduciranimacion < 1)
        {
            if (play == false && (FrameIndex > 1))
            {
                resetElements();
                //First Interpolation
                interpolation();
                play = true;
                playIndex = 0;
                i_curr_steps = 0;
                reproduciranimacion++;
                printf("\n Presiona 0 para habilitar reproducir de nuevo la animación\n");
                habilitaranimacion = 0;
            }
            else
            {
                play = false;
            }
        }
    }
    if (keys[GLFW_KEY_0])
    {
        if (habilitaranimacion < 1 && reproduciranimacion>0)
        {
            printf("Ya puedes reproducir de nuevo la animación con 9\n");
            reproduciranimacion = 0;
        }
    }
}

```

All this helped us create the keyframe animation, the following frames were obtained:

frameindex 1 movCanica_z es: 0.000000 movCanica_x es: -0.500000 presiona P para habilitar	frameindex 2 movCanica_z es: 3.500000 movCanica_x es: -0.500000 presiona P para habilitar	frameindex 3 movCanica_z es: 2.000000 movCanica_x es: 2.000000 presiona P para habilitar
frameindex 6 movCanica_z es: 3.000000 movCanica_x es: 6.000000 presiona P para habilitar	frameindex 7 movCanica_z es: 3.000000 movCanica_x es: 7.500000 presiona P para habilitar	frameindex 4 movCanica_z es: 0.500000 movCanica_x es: 6.500000 presiona P para habilitar
frameindex 5 movCanica_z es: 0.500000 movCanica_x es: 8.000000 presiona P para habilitar	frameindex 8 movCanica_z es: 1.500000 movCanica_x es: 3.500000 presiona P para habilitar	frameindex 9 movCanica_z es: 0.000004 movCanica_x es: 0.000053 presiona P para habilitar

So these are placed as active frames:

```

//FRAME INICIAL
KeyFrame[0].movCanica_z = 0.0f;
KeyFrame[0].movCanica_x = 0.0f;

//FRAMES RESTANTES CALCULADOS
KeyFrame[1].movCanica_z = 0.0f;
KeyFrame[1].movCanica_x = -0.5f;

KeyFrame[2].movCanica_z = 3.5f;
KeyFrame[2].movCanica_x = -0.5f;

KeyFrame[3].movCanica_z = 2.0f;
KeyFrame[3].movCanica_x = 2.0f;

KeyFrame[4].movCanica_z = 0.5f;
KeyFrame[4].movCanica_x = 6.5f;

KeyFrame[5].movCanica_z = 0.5f;
KeyFrame[5].movCanica_x = 8.0f;

KeyFrame[6].movCanica_z = 3.0f;
KeyFrame[6].movCanica_x = 6.0f;

KeyFrame[7].movCanica_z = 3.0f;
KeyFrame[7].movCanica_x = 7.5f;

KeyFrame[8].movCanica_z = 1.5f;
KeyFrame[8].movCanica_x = 3.5f;

KeyFrame[9].movCanica_z = 0.000004f;
KeyFrame[9].movCanica_x = 0.000053f;

```

These are placed in a loop to repeat the same animation:

```

        movCanica_z += KeyFrame[playIndex].movCanica_zInc;
        movCanica_x += KeyFrame[playIndex].movCanica_xInc;
        i_curr_steps++;
    }
}
else {
    movCanica_x = KeyFrame[0].movCanica_z;
    movCanica_z = KeyFrame[0].movCanica_x;
    play = true;
}

```

In conjunction with the keyframe animation, the display changes with different textures and scores when

We advance in the animation:


```

if (playIndex <= 2) {
    displays = 0;
}
if (playIndex > 2 && playIndex <= 5) {
    displays = 1;
}
if (playIndex > 5 && playIndex <= 7) {
    displays = 2;
}
if (playIndex > 7 && playIndex <= 10) {
    displays = 3;
}

```

```

//display
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(8.0f, 4.75f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0f, 1.0f, 1.0f, 1.0f);
if (displays == 0) {
    display0.Draw(lightingShader);
}
if (displays == 1) {
    display1.Draw(lightingShader);
}
if (displays == 2) {
    display2.Draw(lightingShader);
}
if (displays == 3) {
    display3.Draw(lightingShader);
}

```

For the second marble, the flags that will control the path of the marble are placed in the following block, in addition to the variables to modify its position and the flag that will alternate between route 1 and 2:

```

//animacion canica 2
float xcanica2, ycanica2, zcanica2;
bool a1, a2, a3, a4, a5, a6, a7, a8, a9;
bool a10 = false;
bool a11 = false;
bool a12 = false;
bool ruta1 = true;
bool ruta2 = false;

```

Then the marble is placed in the slot where the route will begin:

```

//canica 2
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-6.3f, 0.25f, 3.75f));
model = glm::translate(model, glm::vec3(xcanica2, ycanica2, zcanica2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0f, 1.0f, 1.0f, 1.0f);
canica.Draw(lightingShader);

```

We create this void function so that it detects the right mouse button and thus the animation will be activated:

```
void MouseButtonCallback(GLFWwindow* window, int button, int action, int mods) {
    if (button == GLFW_MOUSE_BUTTON_RIGHT && action == GLFW_PRESS) {
        a1 = true; // Inicia la animación
    }
}
```

```
if (a1) {
    if (despPal > -0.3f) {
        despPal -= 0.05f;
    }
    if (despPal2 > -0.2f) {
        despPal2 -= 0.01f;
    }
    if (escPal < 1.6f) {
        escPal += 0.1f;
    }
    if (escPal2 > 0.8f) {
        escPal2 -= 0.01f;
    }
    if (xcanica2 > -0.45f) {
        xcanica2 -= 0.01f;
    }
    if (zcanica2 < 0.4f) {
        zcanica2 += 0.05f;
    }
    if (zcanica2 >= 0.4f) {
        a1 = false;
        a2 = true;
    }
}
```

```
if (a2) {
    if (xcanica2 < 14.0) {
        xcanica2 += 0.05;
    }
    if (despPal < 0.0f) {
        despPal += 0.05f;
    }
    if (despPal2 < 0.0f) {
        despPal2 += 0.01f;
    }
    escPal = escPal2 = 1.0f;
    if (xcanica2 >= 14.0) {
        a2 = false;
        a3 = true;
    }
}
```

```
if (a3) {
    if (zcanica2 > -7.6f) {
        zcanica2 -= 0.1f;
    }
    if (zcanica2 <= -7.6f) {
        a3 = false;
        a4 = true;
    }
    if (ycanica2 < 0.5f) {
        ycanica2 += 0.1f;
    }
}
if (a4) {
    if (zcanica2 < -0.6f) {
        zcanica2 += 0.08f;
    }
    if (xcanica2 > 0.0f) {
        xcanica2 -= 0.1f;
    }
    if (ycanica2 > 0.2f) {
        ycanica2 -= 0.1f;
    }
    if (zcanica2 >= -0.6) {
        a4 = false;
        a5 = true;
    }
}
```

At the end of animation 4, depending on the value of route1 and route2, it will perform one of the following codes:

```
if (ruta1) {
    if (a5) {
        if (xcanica2 < 8.0) {
            xcanica2 += 0.08;
        }
        if (xcanica2 >= 8.0f) {
            a5 = false;
            a6 = true;
        }
    }
    if (a6) {
        if (xcanica2 > -3.5) {
            xcanica2 -= 0.035;
        }
        if (zcanica2 > -5.0) {
            zcanica2 -= 0.06;
        }
        if (zcanica2 <= -5.0) {
            a6 = false;
            a7 = true;
        }
    }
    if (a7) {
        if (xcanica2 > -1.0) {
            xcanica2 -= 0.06;
        }
        if (zcanica2 < 1.0) {
            zcanica2 += 0.01;
        }
        if (xcanica2 <= -1.0) {
            a7 = false;
            xcanica2 = 0.0;
            zcanica2 = 0.0;
            ruta1 = false;
            ruta2 = true;
        }
    }
}
```

```
if (ruta2) {
    if (a5) {
        if (zcanica2 > -3.0) {
            zcanica2 -= 0.03;
        }
        if (xcanica2 < 7.0) {
            xcanica2 += 0.07;
        }
        if (xcanica2 >= 7.0) {
            a5 = false;
            a6 = true;
        }
    }
    if (a6) {
        if (xcanica2 > 1.0) {
            xcanica2 -= 0.02;
        }
        if (zcanica2 > -5.0) {
            zcanica2 -= 0.05;
        }
        if (xcanica2 <= 1.0) {
            a6 = false;
            a7 = true;
        }
    }
    if (a7) {
        if (zcanica2 < -2.3) {
            zcanica2 += 0.1;
        }
        if (zcanica2 >= -2.3) {
            a7 = false;
            a8 = true;
        }
    }
    if (a8) {
        if (zcanica2 > -4.0) {
            zcanica2 -= 0.04;
        }
        if (xcanica2 > -1.0) {
            xcanica2 -= 0.05;
        }
        if (xcanica2 <= -1.0) {
            a8 = false;
            xcanica2 = 0.0;
            zcanica2 = 0.0;
            ruta2 = false;
            ruta1 = true;
        }
    }
}
```

These routes will alternate, allowing you to view the animations in more detail.

Board lights:

For the dashboard lights, 4 were placed, one at the bottom of the dashboard in front of the flippers, 2 for bumpers one on the left bumper and one on the right bumper and one in the center of the machine. These lights are They can be turned on independently with the H, J, K and L keys.

```
119 // Positions of the point lights
120 glm::vec3 pointLightPositions[] = { // Cada vector se cambia para que haya un point
121     glm::vec3(-7.4f, -0.2f, -0.0f),
122     glm::vec3(3.4f, -0.2f, -1.5f),
123     glm::vec3(3.4f, -0.2f, 2.2f),
124     glm::vec3(0.f, -0.2f, 0.0f)
125 };
126
127 struct Spotlight {
128     glm::vec3 Pos;
129     glm::vec3 Dir;
130 };
131 Spotlight spotlight;
```

```
630 // Directional light
631 glUniform3f(glGetUniformLocation(lightningShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
632 glUniform3f(glGetUniformLocation(lightningShader.Program, "dirLight.ambient"), 0.3f, 0.3f, 0.3f);
633 glUniform3f(glGetUniformLocation(lightningShader.Program, "dirLight.diffuse"), 0.3f, 0.3f, 0.3f);
634 glUniform3f(glGetUniformLocation(lightningShader.Program, "dirLight.specular"), 1.0f, 1.0f, 1.0f);
635
636
637
638 // Point light 1
639 glm::vec3 lightColor;
640 lightColor.x = abs(2*sin(glfwGetTime() * Light1.x));
641 lightColor.y = abs(2*sin(glfwGetTime() * Light1.y));
642 lightColor.z = abs(2*sin(glfwGetTime() * Light1.z));
643 // Point light 1
644 glUniform3f(glGetUniformLocation(lightningShader.Program, "pointLights[0].position"), pointLightPositions[0].x, pointLightPosi
645 glUniform3f(glGetUniformLocation(lightningShader.Program, "pointLights[0].ambient"), lightColor.x, lightColor.y, lightColor.z)
646 glUniform3f(glGetUniformLocation(lightningShader.Program, "pointLights[0].diffuse"), lightColor.x, lightColor.y, lightColor.z)
647 glUniform3f(glGetUniformLocation(lightningShader.Program, "pointLights[0].specular"), 1.0f, 1.0f, 1.0f);
648 glUniform1f(glGetUniformLocation(lightningShader.Program, "pointLights[0].constant"), 1.0f); // Estas 3 en todas
649 glUniform1f(glGetUniformLocation(lightningShader.Program, "pointLights[0].linear"), 0.35f);
650 glUniform1f(glGetUniformLocation(lightningShader.Program, "pointLights[0].quadratic"), 0.44f);
651
```

Configur

```
705 // SpotLight
706 spotlight.Pos.y += desplazamientoY;
707 glUniform3f(glGetUniformLocation(lightningShader.Program, "spotLight.position"), spotlight.Pos.x, spotlight.Pos.y, spotlight.P
708 glUniform3f(glGetUniformLocation(lightningShader.Program, "spotLight.direction"), spotlight.Dir.x, spotlight.Dir.y, spotlight.
709 glUniform3f(glGetUniformLocation(lightningShader.Program, "spotLight.ambient"), 0.7f, 0.7f, 0.7f);
710 glUniform3f(glGetUniformLocation(lightningShader.Program, "spotLight.diffuse"), 0.7f, 0.7f, 0.7f);
711 glUniform3f(glGetUniformLocation(lightningShader.Program, "spotLight.specular"), 1.0f, 1.0f, 1.0f);
712 glUniform1f(glGetUniformLocation(lightningShader.Program, "spotLight.constant"), 1.0f);
713 glUniform1f(glGetUniformLocation(lightningShader.Program, "spotLight.linear"), 0.032f);
714 glUniform1f(glGetUniformLocation(lightningShader.Program, "spotLight.quadratic"), 0.045f);
715 glUniform1f(glGetUniformLocation(lightningShader.Program, "spotLight.cutOff"), glm::cos(glm::radians(12.5f)));
716 glUniform1f(glGetUniformLocation(lightningShader.Program, "spotLight.outerCutOff"), glm::cos(glm::radians(15.0f)));
717
718 // Set material properties
719 glUniform1f(glGetUniformLocation(lightningShader.Program, "material.shininess"), 16.0f);
```

Skybox:

To place the skybox, we first make the relevant Texture.h includes. You have the following:


```

Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting.frag");
Shader lampShader("Shaders/lamp.vs", "Shaders/lamp.frag");
Shader SkyBoxshader("Shaders/SkyBox.vs", "Shaders/SkyBox.frag");

```

```

// Load textures
vector<const GLchar*> faces;
faces.push_back("SkyBox/cupertin-lake_rt.tga");
faces.push_back("SkyBox/cupertin-lake_lf.tga");
faces.push_back("SkyBox/cupertin-lake_up.tga");
faces.push_back("SkyBox/cupertin-lake_dn.tga");
faces.push_back("SkyBox/cupertin-lake_bk.tga");
faces.push_back("SkyBox/cupertin-lake_ft.tga");
/*
faces.push_back("SkyBox/cupertin-lake-night_rt.tga");
faces.push_back("SkyBox/cupertin-lake-night_lf.tga");
faces.push_back("SkyBox/cupertin-lake-night_up.tga");
faces.push_back("SkyBox/cupertin-lake-night_dn.tga");
faces.push_back("SkyBox/cupertin-lake-night_bk.tga");
faces.push_back("SkyBox/cupertin-lake-night_ft.tga");
*/

GLuint cubemapTexture = TextureLoading::LoadCubemap(faces);

```

And until the end of the render we place the following:

```

// Draw skybox as last
glDepthFunc(GL_LEQUAL); // Change depth function so depth test passes when values are equal to depth buffer's content
SkyBoxshader.Use();
view = glm::mat4(glm::mat3(camera.GetViewMatrix())); // Remove any translation component of the view matrix
glUniformMatrix4fv(glGetUniformLocation(SkyBoxshader.Program, "view"), 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(glGetUniformLocation(SkyBoxshader.Program, "projection"), 1, GL_FALSE, glm::value_ptr(projection));

// skybox cube
glBindVertexArray(skyboxVAO);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_CUBE_MAP, cubemapTexture);
glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);
glDepthFunc(GL_LESS); // Set depth function back to default

```

Importing models has a similar structure in most cases, however, objects that have animation had different statements with individual variables to manipulate their movements either with functions or with basic transformations. The following is an example of the general drawing of fixed objects.

```

//base bumper adelante
model = glm::mat4(1.0);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0f, 1.0f, 1.0f, 1.0f);
basebumpatras.Draw(lightingShader);

```

And the following is an example of declaring and drawing an object with animation.

```
//miles con base
time += deltaTime;
heightmiles = (sin(time * speedmiles) + 1.0f) * (maxHeightmiles / 2.0f);
if (heightmiles < 0.0f) heightmiles = 0.0f;
model = glm::mat4(1.0);
model = glm::translate(glm::mat4(1.0f), glm::vec3(-0.5f, heightmiles, -1.5f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform4f(glGetUniformLocation(lightningShader.Program, "colorAlpha"), 1.0f, 1.0f, 1.0f, 1.0f);
milesbase.Draw(lightningShader);
```

As you can see, we have a heightmiles variable to control the movement of one of the bases, a sine function is used so that the movement is constant up and down, we can control the speed by declaring the speedmiles variable and also your maximum height with maxHeightmiles.

The following example is from the animation using basic transformations for spot bust rotation.

```
//Spot
rotspot += rotationSpeed * deltaTime;
if (rotspot > 360.0f) {
    ...
    rotspot -= 360.0f; // Para evitar un overflow de la variable
}
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(6.8f, 0.8f, 3.3f));
model = glm::rotate(model, glm::radians(rotspot), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform4f(glGetUniformLocation(lightningShader.Program, "colorAlpha"), 1.0f, 1.0f, 1.0f, 1.0f);
spot.Draw(lightningShader);
```

The rotate transformation is used with the rotspot variable on the Y axis so that it rotates 360 degrees infinitely.

Later we have a key activation code like the one used for flippers.

We know that flippers in a real machine are activated by pressing a physical button and when released it returns to its original position. So it had to be replicated in this project. To do this, this code was used.

```

//Flipper Izquierdo
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-6.5f, 0.25f, -1.6f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(rotFlipperIzq), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform4f(glGetUniformLocation(lightningShader.Program, "colorAlpha"), 1.0f, 1.0f, 1.0f, 1.0f);
flipizq.Draw(lightningShader);

//Flipper Derecho
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-6.6f, 0.25f, 1.55f));
model = glm::rotate(model, glm::radians(rotFlipperDer), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform4f(glGetUniformLocation(lightningShader.Program, "colorAlpha"), 1.0f, 1.0f, 1.0f, 1.0f);
flipder.Draw(lightningShader);

```

We can see that basic transformations are used to accommodate the Flipper in its position on the board, but particularly here there is a second rotate, this rotate uses the variable rotFlipperLeft and rotFlipperDer to rotate on the Y axis.

```

16 void DoMovement()
17 {
18
19     // Camera controls
20     if (keys[GLFW_KEY_Z]) // Si se presiona la tecla Z y la rotación es menor a 75 grados.
21     {
22         rotFlipperIzq = 45.0f; // Incrementa la rotación.
23         traslacionBotonZ = traslacionMaxima;
24     }
25     else if (!keys[GLFW_KEY_Z]) // Si se suelta la tecla Z y la rotación es mayor a 0 grados.
26     {
27         rotFlipperIzq = 0.0f; // Decrementa la rotación.
28         traslacionBotonZ = 0.0f;
29     }
30     if (keys[GLFW_KEY_X]) // Si se presiona la tecla X y la rotación es menor a 75 grados.
31     {
32         rotFlipperDer = -45.0f; // Incrementa la rotación.
33         traslacionBotonX = traslacionMaximader;
34     }
35     else if (!keys[GLFW_KEY_X]) // Si se suelta la tecla X y la rotación es mayor a 0 grados.
36     {
37         rotFlipperDer = 0.0f; // Decrementa la rotación.
38         traslacionBotonX = 0.0f;
39     }

```

Then we use a function called DoMovement that will be for everything related to the user's interaction with the keys. In this function we use the Z key and the pressed, if this condition is met then the variable remains at 0 and would return to its original position.

Avatar:

For this, we divide the character into 5 parts, these being his torso and his limbs, which will be placed

in the following way:

```
//O hara
glm::mat4 tmp = glm::mat4(1.0f); //Temp
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::rotate(model, glm::radians(-rot6), glm::vec3(0.0f, 1.0f, 0.0f));
tmp = model = glm::translate(model, glm::vec3(0, 1, 0));
model = glm::translate(model, glm::vec3(posX, 0.0, posZ));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
torsoAvatar.Draw(lightningShader);

//Pierna Izq
view = camera.GetViewMatrix();
model = glm::translate(tmp, glm::vec3(0.0f, 0.6f, 0.0f));
model = glm::translate(model, glm::vec3(posX, 0.0, posZ));
//model = glm::rotate(model, glm::radians(rot), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(-rot2), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
piernaderAvatar.Draw(lightningShader);

//Pierna Der
view = camera.GetViewMatrix();
model = glm::translate(tmp, glm::vec3(0.0f, 0.6f, 0.0f));
model = glm::translate(model, glm::vec3(posX, 0.0, posZ));
//model = glm::rotate(model, glm::radians(rot), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(rot2), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
piernaizqAvatar.Draw(lightningShader);

//Brazo derecho
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(tmp, glm::vec3(0.0f, 0.0f, -0.0f));
model = glm::translate(model, glm::vec3(posX, 0.0, posZ));
//model = glm::rotate(model, glm::radians(rot), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::translate(model, glm::vec3(0.0f, 0.85f, 0));
model = glm::rotate(model, glm::radians(rot2), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
brazoderAvatar.Draw(lightningShader);

//Brazo Izquierdo
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(tmp, glm::vec3(0.0f, 0.0f, -0.0f));
model = glm::translate(model, glm::vec3(posX, 0.0, posZ));
/*model = glm::rotate(model, glm::radians(rot2), glm::vec3(0.0f, 1.0f, 0.0f));*/
model = glm::translate(model, glm::vec3(0.0f, 0.85f, 0));
model = glm::rotate(model, glm::radians(-rot2), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
brazoizqAvatar.Draw(lightningShader);
```


Having these values, we will use the variable `rot2` within a boolean condition called `animation` to move the limbs and simulate walking, and they will move when we press any of the WASD keys within the `DoMovement` function, and the animation will stop when we stop pressing the keys:

```
if (keys[GLFW_KEY_W] || keys[GLFW_KEY_UP])
{
    /*camera.ProcessKeyboard(FORWARD, deltaTime);*/
    if (posZ > -7.5) {
        posZ -= .1;
    }
    animacion = true;
}

if (keys[GLFW_KEY_S] || keys[GLFW_KEY_DOWN])
{
    //camera.ProcessKeyboard(BACKWARD, deltaTime);
    if (posZ < 7.5) {
        posZ += .1;
    }
    animacion = true;
}

if (keys[GLFW_KEY_A] || keys[GLFW_KEY_LEFT])
{
    //camera.ProcessKeyboard(LEFT, deltaTime);
    if (posX > -4.0) {
        posX -= .1;
    }
    animacion = true;
}

if (keys[GLFW_KEY_D] || keys[GLFW_KEY_RIGHT])
{
    //camera.ProcessKeyboard(RIGHT, deltaTime);
    if (posX < 4.0f) {
        posX += .1;
    }
    animacion = true;
}
```

```
if (!keys[GLFW_KEY_W] && !keys[GLFW_KEY_D]) {
    if (!keys[GLFW_KEY_A] && !keys[GLFW_KEY_S]) {
        animacion = false;
        rot2 = 0.0f;
    }
}
```

It is observed that within the translation movement the values are limited so that it only travels inside the board.

Now to manage the swing of the limbs the following was used:

```
if (animacion) {
    if (anim2) {
        rot2 += 0.5;
        if (rot2 >= 30.0f) {
            anim2 = false;
        }
    }
    else {
        rot2 -= 0.5;
        if (rot2 <= -30.0f) {
            anim2 = true;
        }
    }
}
```

Audio:

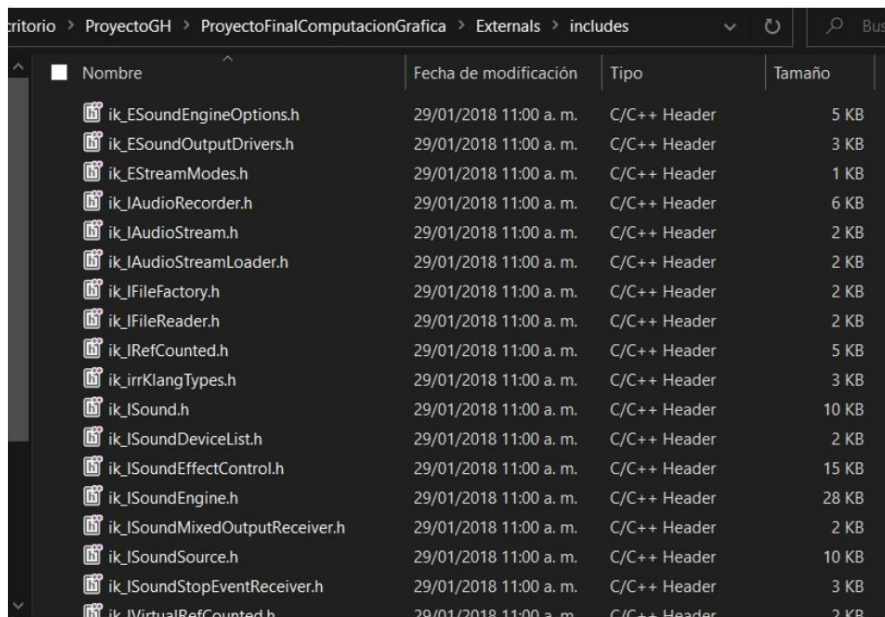
For the music section, the `Irrklang` library was used, which was our favorite because the library's documentation is clear in its explanations, in addition to having small

tutorials to implement audio tracks in our C++ and C# projects.

Two other libraries were also considered: the first (called OpenAL), which allowed us to handle 2D and 3D audio for video games specifically; However, the method to implement our audios was very complex due to the large number of arguments used by the functions of this library.

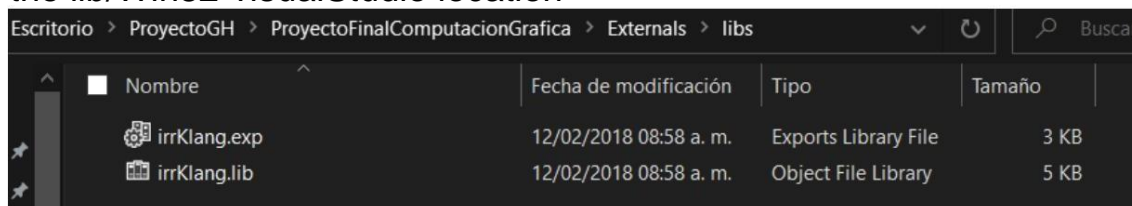
The second was Audiere, which has clean and clear documentation, but was discarded because only 2D audio can be implemented, in addition to the fact that no tutorials were found on the internet that suited our project.

To correctly configure our chosen library, we download the .zip file of the 32-bit version from the official Irrklang website. When opening the extracted file, the folder was selected include, which we place in a folder within the project called Externals:



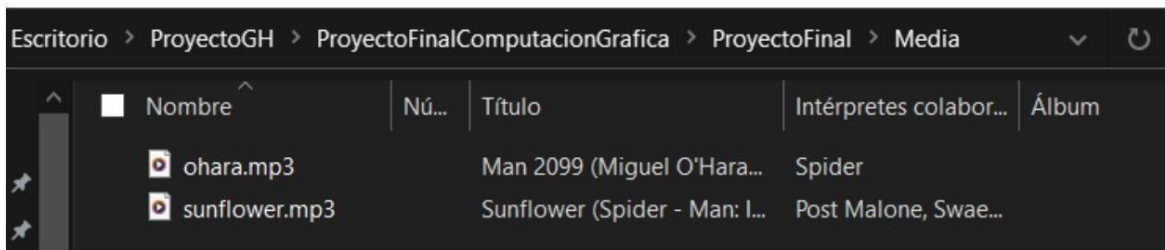
Nombre	Fecha de modificación	Tipo	Tamaño
ik_ESoundEngineOptions.h	29/01/2018 11:00 a. m.	C/C++ Header	5 KB
ik_ESoundOutputDrivers.h	29/01/2018 11:00 a. m.	C/C++ Header	3 KB
ik_EStreamModes.h	29/01/2018 11:00 a. m.	C/C++ Header	1 KB
ik_IAudioRecorder.h	29/01/2018 11:00 a. m.	C/C++ Header	6 KB
ik_IAudioStream.h	29/01/2018 11:00 a. m.	C/C++ Header	2 KB
ik_IAudioStreamLoader.h	29/01/2018 11:00 a. m.	C/C++ Header	2 KB
ik_IFileFactory.h	29/01/2018 11:00 a. m.	C/C++ Header	2 KB
ik_IFileReader.h	29/01/2018 11:00 a. m.	C/C++ Header	2 KB
ik_IRefCounted.h	29/01/2018 11:00 a. m.	C/C++ Header	5 KB
ik_irrKlangTypes.h	29/01/2018 11:00 a. m.	C/C++ Header	3 KB
ik_ISound.h	29/01/2018 11:00 a. m.	C/C++ Header	10 KB
ik_ISoundDeviceList.h	29/01/2018 11:00 a. m.	C/C++ Header	2 KB
ik_ISoundEffectControl.h	29/01/2018 11:00 a. m.	C/C++ Header	15 KB
ik_ISoundEngine.h	29/01/2018 11:00 a. m.	C/C++ Header	28 KB
ik_ISoundMixedOutputReceiver.h	29/01/2018 11:00 a. m.	C/C++ Header	2 KB
ik_ISoundSource.h	29/01/2018 11:00 a. m.	C/C++ Header	10 KB
ik_ISoundStopEventReceiver.h	29/01/2018 11:00 a. m.	C/C++ Header	3 KB
ik_VirtualRefCounted.h	29/01/2018 11:00 a. m.	C/C++ Header	2 KB

Inside the Externals folder we also inserted the libs folder, which was inside the .zip in the lib/Win32-visualStudio location

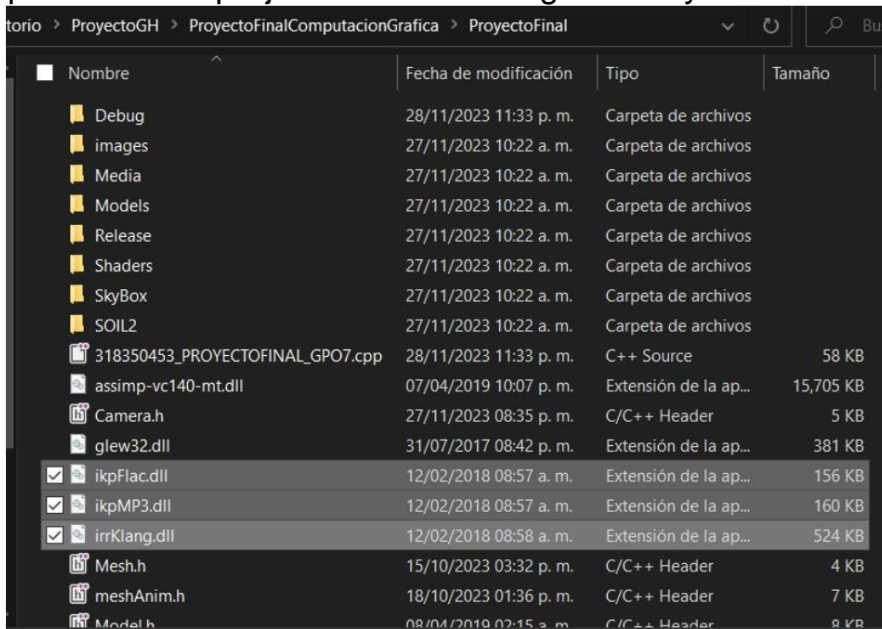


Nombre	Fecha de modificación	Tipo	Tamaño
irrKlang.exp	12/02/2018 08:58 a. m.	Exports Library File	3 KB
irrKlang.lib	12/02/2018 08:58 a. m.	Object File Library	5 KB

Within our project, the media folder was created, where we will insert our songs:

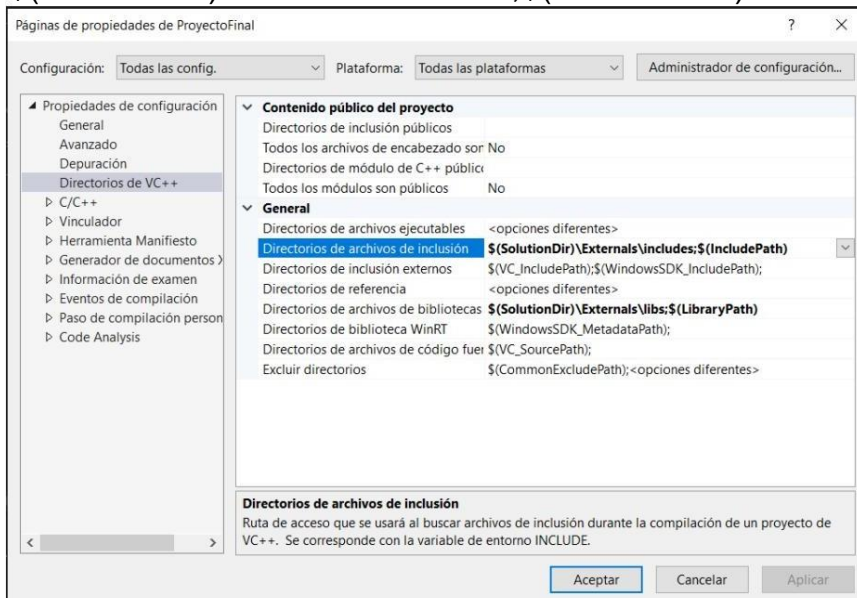


Finally, the .dll files are extracted from the .zip in bin/win32-visualStudio, which are placed in our project in the following directory:



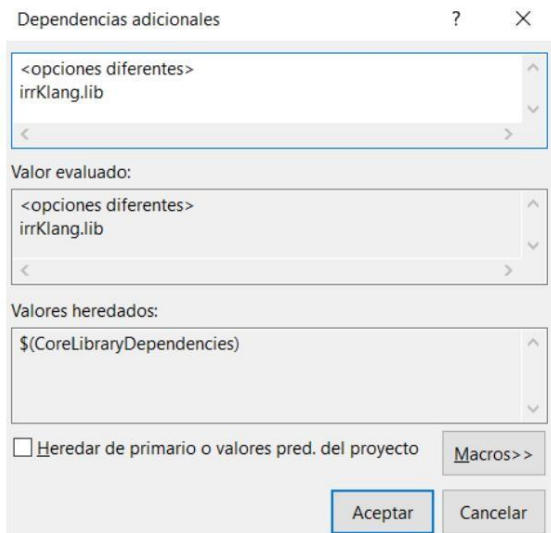
For the configuration within Visual, we open the project properties, select VC++ Directories, and then Include File Directories, and paste the following line:

`$(SolutionDir)\Externals\includes;$(IncludePath)`



Now we go to Linker/Input/Additional Dependencies, and paste the text:

irrKlang.lib



Click apply, and you will be ready to use Irrklang in your code.

To implement the music in the code, we import the .h and declare the namespace:

```
4 //Biblioteca de musica
5 #include <irrKlang.h>
6 using namespace irrklang;
```

Now we create 2 “devices”, one in 2D and one in 3D with the option enabled for them to play in a loop, and we will make the 3D audio be placed at the coordinate (3,0,0) in order to listen to it “a how far away”:

```
//configuracion del sonido
vec3df pos3d(3, 0, 0);
ISoundEngine* engine = createIrrKlangDevice();
ISoundEngine* engine2 = createIrrKlangDevice();
engine->play2D("media/ohara.mp3", true);
engine2->play3D("media/sunflower.mp3", vec3df(pos3d), true);
```

Now we simply place the position from where we listen to the audio, and the minimum distance at which the 3D audio will be heard:

```
vec3df listenerPos = vec3df(0, 0, 0);
vec3df listenerLookDir = vec3df(0, 0, 1);
engine->setListenerPosition(listenerPos, listenerLookDir);

if (engine2)
    engine2->setDefault3DSoundMinDistance(8.0f);
```

Finally, we free the memory of the devices:


```

glfwTerminate();

engine->drop();
engine2->drop();

return 0;

```

This is placed inside the main function in the window's while loop, so that the music starts when the pinball table loads, and ends when the window is closed.

Variables Dictionary

bool active; **It was used for the animation of the flippers**

float rotFlipperLzq = 0.0f; **Flipper rotation 1**

float rotFlipperDer = 0.0f; **Flipper rotation 2**

float rotFlipperArr = 0.0f; **Flipper rotation 3**

float rotspot = 0.0f; **Rotation initialization for the spot figure**

float rotationSpeed = 45.0f; **Degrees per second for spin speed**

bool anim; **It was used for the animation of the flippers**

//animation bumpers forward

float logoBumpHeight = 0.35f; **Maximum movement height of the bumpers** float

logoBumpSpeed = 1.5f; **Speed of movement**

const float maxYLimit = -0.005f; **Maximum and negative that can be reached**

const float maxYLimitd = -0.005f; **Same case but for the right bumper**

//back bumper animation

const float logoBumpSpeedatras = 2.0f; **Animation speed for the back bumper**

const float logoBumpHeightatras = 0.35f; **Maximum animation height**

const float initialPhase = 1.5f; **Initial phase to start at a different point**

Bool ruta1, ruta2; **Flag that allows us to change between the marble 2 routes**

Bool a1, a2, a3, a4, a5, a6, a7, a8 = false; **Phases of movement of the marble 2, animated without keyframes**

Bool anim2 = true; bool anim3 = false; animation that simulates the avatar's walking

Float rot2 = 0.0f; angle in radians to rotate the avatar's limbs, will be used positively and negative to make them move more realistically

//miles base animation

float timemiles = 0.0f; Time initialization for Miles base

float speedmiles = 0.5f; Oscillation speed

float maxHeightmiles = 1.0f; Maximum height the base can reach

float heightmiles = 0.0f; Start of the animation at that height

//gwen base animation

float timegwen = 0.0f; Initialization time for Gwen's base

float speedgwen = 0.5f; Oscillation speed

float maxHeightgwen = 1.0f; Maximum height for Gwen's base

float heightgwen = 0.0f; Start of animation at this point

//lever animation

float despPal = 0.0f;

float escPal = 1.0f;

float despPal2 = 0.0f;

float escPal2 = 1.0f;

//oHara animation

float posX; //Variable for PositionX

float posY; //Variable for PositionY

float posZ; //Variable for PositionZ

float incX; //Variable for IncrementX

float incY; //Variable for IncrementY

float incZ; //Variable for IncrementZ

bool anim2 = true, animacion = false; //flags that are used to determine whether the animation is activated or not, and how will it be activated

float rot = 90.0f, rot2 = 0.0f, rot3 = 0.0f; //the character is rotated 90 degrees to orient him towards the machine, and the another 2 function to simulate the movement of the avatar's limbs

//marble animation 2

```
float xcanica2, ycanica2, zcanica2; //they work to make the traditional marble animation
```

```
bool a1, a2, a3, a4, a5, a6;
```

```
float desplazamientoY = -1.0f; Variable to adjust the position of the SpotLight
```

```
float intensidadBrillo = 1.5f; Variable to increase the brightness of the PointLight
```

```
//Spider climbing animation
```

```
bool anim2 = true, anim3 = false; Boolean to activate spider animations
```

```
bool anim4 = true, anim5 = false;
```

```
bool anim6 = true, anim7 = false;
```

```
float rot = 30.0f, rot2 = 0.0f, rot3 = 0.0f, rot4 = 0.0, rot5 = 0.0, rot6 = -90.0, rot7 =
```

```
180.0; Declaration of the rotations for each of the parts of this spiderman
```

```
//button animation
```

```
Float traslacionBotonZ = 0.0f; Variable for the translation of the side buttons
```

```
of the machine
```

```
const float traslacionMaxima = 0.04f; Maximum translation that the button can have
```

```
float traslacionBotonX = 0.0f;
```

```
const float traslacionMaximader = -0.04f;
```