Manual Técnico

Computación Gráfica e Interacción Humano Computadora

Universidad Nacional
Autonoma De Mexico

Grupo: 4

29/11/2023

Alumnos:
Herrera Alcantara Emilio Ramsés
Martinez Ramirez Jose Angel

Profesor: Jose Roque Roman Guadarrama

Manual técnico

Canicas de pinball. Para cubrir el requisito:

- 1 canica que tenga animación por keyframes y que colisione con al menos 3 objetos
- 1 canica animada que siga un recorrido por el tablero

Utilizamos los modelos llamados *canica1* y *canica2*, siendo animadas de la manera indicada respectivamente. La primera canica es activada al presionar la tecla 9, lo que ejecutará el arreglo de posiciones en los keyframes, y que al terminar podremos volver a ejecutar presionando la tecla 0, seguida de la tecla 9 nuevamente; mientras que la segunda lo hará al presionar el click derecho y podremos ejecutarla de nuevo al presionar la misma tecla.

Para la animación por keyframes se utilizó una estructura de teclas para poder manipular la posición de la canica en tiempo de ejecución y así ir guardando frames. Primero tenemos declaraciones de variables para manejar los keyframes:

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

Luego, declaramos más variables para controlar la posición y movimiento de la canica, además de definir el máximo número de 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;
```

Una función para guardar frames, que añade frames al arreglo de frames:

```
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++;
}
```

También una función para resetear los frames y así visualizar la animación creada:

```
void resetElements(void) //Tecla 0
{

    movCanica_x = KeyFrame[0].movCanica_z;
    movCanica_z = KeyFrame[0].movCanica_x;
}
```

También, la función de interpolación para llegar de un punto a otro en la animación:

```
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;
}
```

Finalmente, la función para dibujar la animación creada:

```
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++;
```

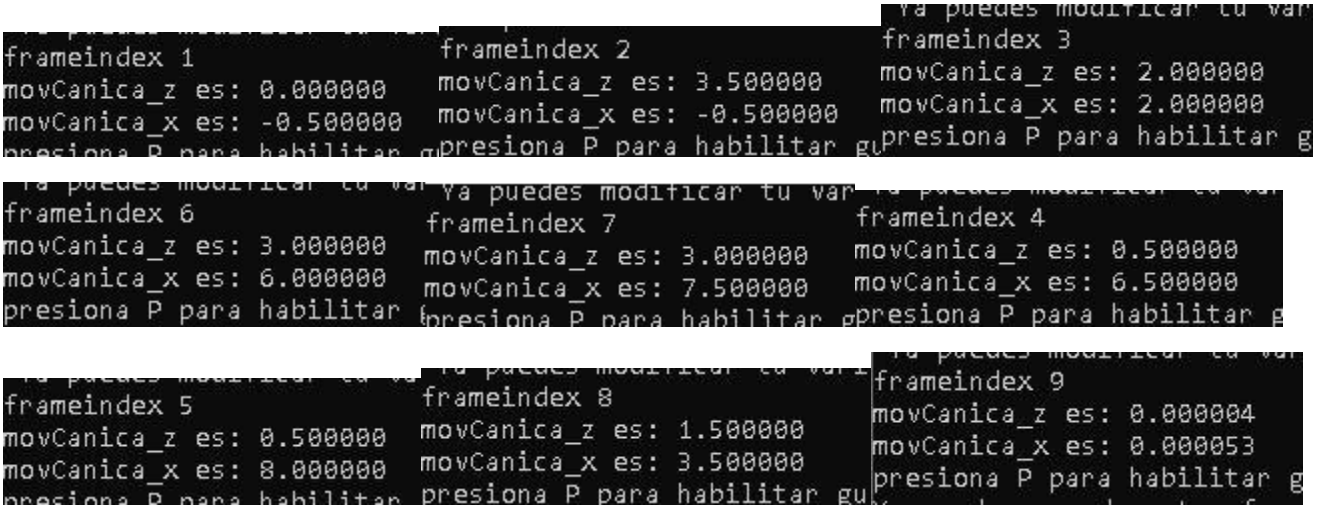
Con esta función se lee del teclado las teclas para controlar la animación e ir guardando frames:

```
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;
```

Todo esto nos ayudó a crear la animación por keyframes, se obtuvieron los siguientes frames:



De modo que estos mismos se colocan como frames activos:

```
//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;
```

Estos se colocan en bucle para estar repitiendo la misma animación:

```
            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;
    }
```

En conjunto con la animación por keyframes, se cambia el display con distintas texturas y puntajes cuando avanzamos en la animación:

```
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);
}
```

Para la segunda canica se coloca en el siguiente bloque las banderas que controlarán el camino de la canica, además de las variables para modificar su posición y la bandera que alternará entre la ruta 1 y la 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;
```

Luego es colocada la canica en el slot de donde comenzará la ruta:

```
//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);
```

Creamos esta función void para que detecte el botón derecho del mouse y así se activará la animación:

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

```
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;

    }
}
```

Al terminar la animación 4, dependiendo del valor de ruta1 y ruta2, realizará uno de los siguientes códigos:

```
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;
        }
    }
```

Estas rutas se irán alternando, permitiendo visualizar las animaciones más a detalle.

Luces del tablero:

Para las luces del tablero se colocaron 4, una en el inferior del tablero frente a los flippers, 2 para bumpers uno en el bumper izquierdo y otro en el bumper derecho y uno en el centro de la máquina. Estas luces se pueden encender de forma independiente con las teclas H, J, K y L.

```
// Positions of the point lights
glm::vec3 pointLightPositions[] = {//Cada vector se cambia para que haya un point
    glm::vec3(-7.4f, -0.2f, -0.0f),
    glm::vec3(3.4f, -0.2f, -1.5f),
    glm::vec3(3.4f, -0.2f, 2.2f),
    glm::vec3(0.f, -0.2f, 0.0f)
};

struct Spotlight {
    glm::vec3 Pos;
    glm::vec3 Dir;
};
Spotlight spotlight;
```

```
// Directional light
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.3f, 0.3f, 0.3f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.3f, 0.3f, 0.3f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 1.0f, 1.0f, 1.0f);



// Point light 1
glm::vec3 lightColor;
lightColor.x = abs(2*sin(glfwGetTime() * Light1.x));
lightColor.y = abs(2*sin(glfwGetTime() * Light1.y));
lightColor.z = abs(2*sin(glfwGetTime() * Light1.z));
// Point light 1
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"), pointLightPositions[0].x, pointLightPosi
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"), lightColor.x, lightColor.y, lightColor.z)
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), lightColor.x, lightColor.y, lightColor.z)
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"), 1.0f, 1.0f, 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].constant"), 1.0f);//Estas 3 en todas
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].linear"), 0.35f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"), 0.44f);
```

```
// SpotLight
spotlight.Pos.y += desplazamientoY;
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.position"), spotlight.Pos.x, spotlight.Pos.y, spotlight.P
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.direction"), spotlight.Dir.x, spotlight.Dir.y, spotlight.
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.ambient"), 0.7f, 0.7f, 0.7f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.diffuse"), 0.7f, 0.7f, 0.7f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.specular"), 1.0f, 1.0f, 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.linear"), 0.032f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.quadratic"), 0.045f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.cutOff"), glm::cos(glm::radians(12.5f)));
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.outerCutOff"), glm::cos(glm::radians(15.0f)));

// Set material properties
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);
```

Skybox:

Para colocar el skybox, primero realizamos los include pertinentes de Texture.h. Se tiene lo siguiente:

```
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);
```

Y hasta el final de los render colocamos lo siguiente:

```
// 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
```

La importación de modelos tiene una estructura similar en la mayoría de los casos, sin embargo, los objetos que tienen animación tuvieron diferentes declaraciones con variables individuales para manipular sus movimientos ya fuera con funciones o con transformaciones básicas. El siguiente es un ejemplo de el dibujado general de objetos fijos.

```
//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);
```

Y el siguiente es un ejemplo de una declaración y dibujado de un objeto con animación.

```
//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(lightingShader.Program, "colorAlpha"), 1.0f, 1.0f, 1.0f, 1.0f);
milesbase.Draw(lightingShader);
```

Como se puede apreciar, tenemos una variable heightmiles para controlar el movimiento de una de las bases, se utiliza una función senoidal para que el movimiento sea constante hacia arriba y hacia abajo, la velocidad la podemos controlar mediante la declaración de la variable speedmiles y también su altura máxima con maxHeightmiles.

El siguiente ejemplo es de la animación usando transformaciones básicas para la rotación del busto de spot.

```
//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(lightingShader.Program, "colorAlpha"), 1.0f, 1.0f, 1.0f, 1.0f);
spot.Draw(lightingShader);
```

Se utiliza la transformación de rotate con la variable rotspot sobre el eje Y para que se encuentre girando 360 grados infinitamente.

Posteriormente tenemos código de activación por tecla como el que se utiliza para los flippers. Sabemos que los flippers en una maquina real se activan al presionar un botón físico y al soltarlo regresa a su posición original. Por lo que había que replicarlo en este proyecto. Para realizarlo, se utilizo este código.

```
//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(lightingShader.Program, "colorAlpha"), 1.0f, 1.0f, 1.0f, 1.0f);
flipizq.Draw(lightingShader);

//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(lightingShader.Program, "colorAlpha"), 1.0f, 1.0f, 1.0f, 1.0f);
flipder.Draw(lightingShader);
```

Podemos ver que se utilizan transformaciones básicas para acomodar el Flipper en su posición del tablero, pero particularmente aquí hay un segundo rotate, este rotate utiliza la varibale rotFlipperIzq y rotFlipperDer para rotar en el eje Y.

```
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      }
```

Despues utilizamos una función llamada DoMovement que será para todo lo relacionado a la interacción del usuario con las teclas. En esta función utilizamos la tecla Z y la tecla X para que al presionarse la varibale rotFlipperIzq adquiera los valores de 45 y de -45 que serán los grados hasta los que puede girar el Flipper y tenemos un else if que verifica que la tecla no este presionada, en dado caso que esa condición se cumpla entonces la variable se queda en 0 y regresaría a su posición original.

Avatar:
Para esto, dividimos al personaje en 5 partes, siendo estas su torso y sus extremidades, las cuales serán colocadas

de la siguiente forma:

```
//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, 0.0));
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(lightingShader);
//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, 0.0));
model = glm::rotate(model, glm::radians(-rot2), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
piernaderAvatar.Draw(lightingShader);

//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(lightingShader);

//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(lightingShader);

//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(lightingShader);
```

Teniendo estos valores, utilizaremos la variable rot2 dentro de una condición booleana llamada animacion para mover las extremidades y simular que camina, y se moverán al presionar alguna de las teclas WASD dentro de la función DoMovement, y que la animación se detenga cuando dejamos de presionar las teclas:

```
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;
    }
}
```

Se observa que dentro del movimiento de traslación se limitan los valores para que recorra solo el interior del tablero.
Ahora para manejar la oscilación de las extremidades se utilizó lo siguiente:

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

Audio:

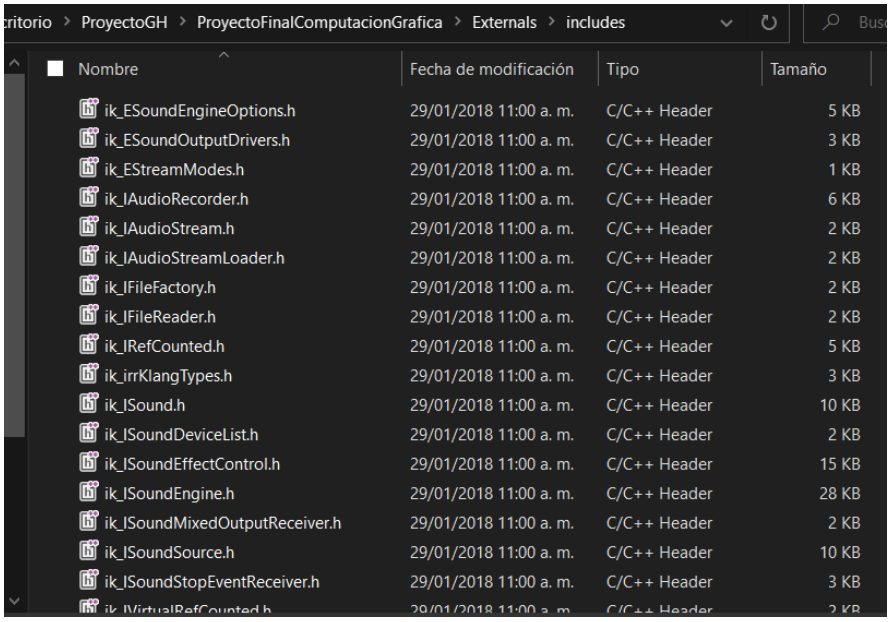
Para el apartado de la música se utilizó la librería Irrklang, la cual fue nuestra favorita debido a que la documentación de la librería es clara en sus explicaciones, además de contar con pequeños

tutoriales para implementar pistas de audio en nuestros proyectos de C++ y C#.

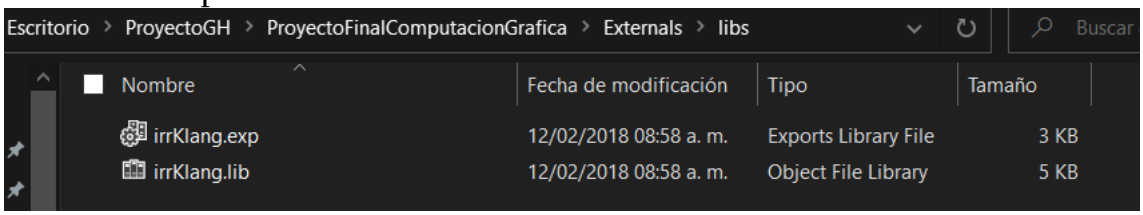
También se consideraron otras dos librerías: la primera (llamada OpenAL), la cual nos permitía manejar audio 2D y 3D para videojuegos específicamente; sin embargo, el método para implementar nuestros audios era muy complejo debido a la gran cantidad de argumentos que utilizan las funciones de esta librería.

La segunda fue Audiere, que tiene una documentación limpia y clara, pero que fue descartada debido a que solo se puede implementar audio 2D, además de que en internet no se encontraron tutoriales que se adecuaran a nuestro proyecto.

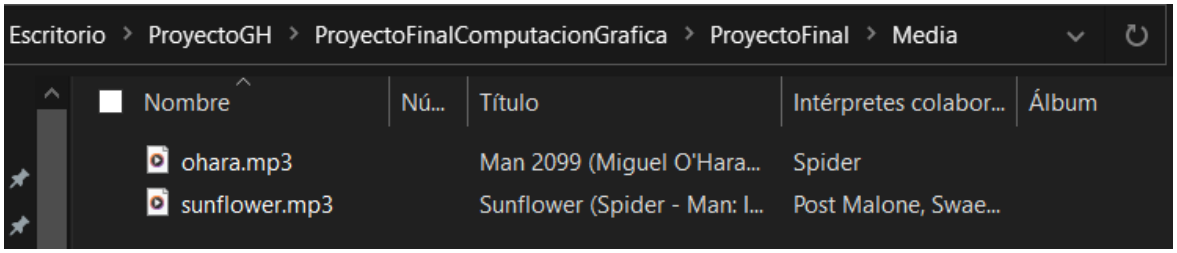
Para configurar correctamente nuestra librería escogida, descargamos el archivo .zip de la versión de 32 bits, en la página oficial de Irrklang. Al abrir el archivo extraído, se seleccionó la carpeta include, la cual colocamos en una carpeta dentro del proyecto llamada Externals:



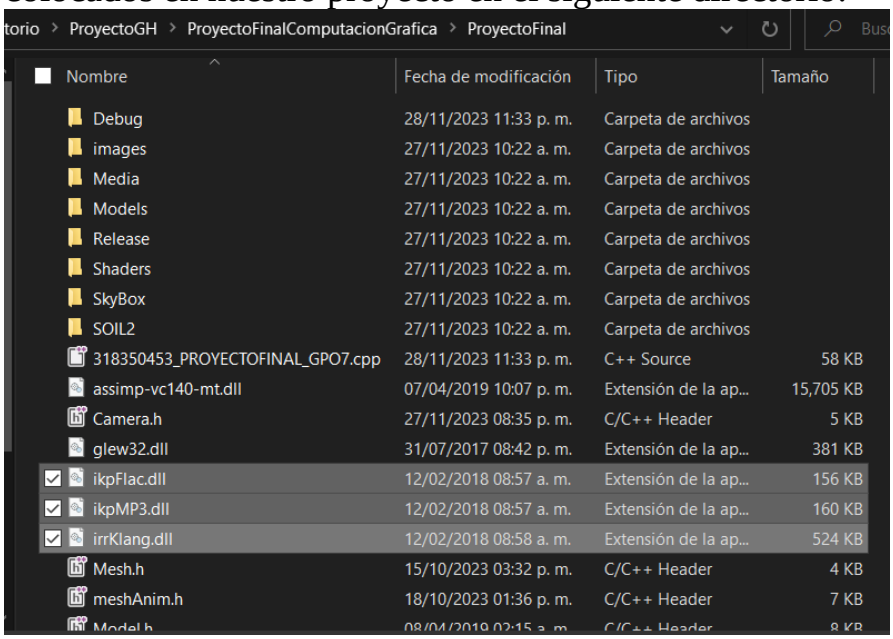
Dentro de la carpeta Externals también insertamos la carpeta libs, misma que se encontraba dentro del .zip en la ubicación lib/Win32-visualStudio



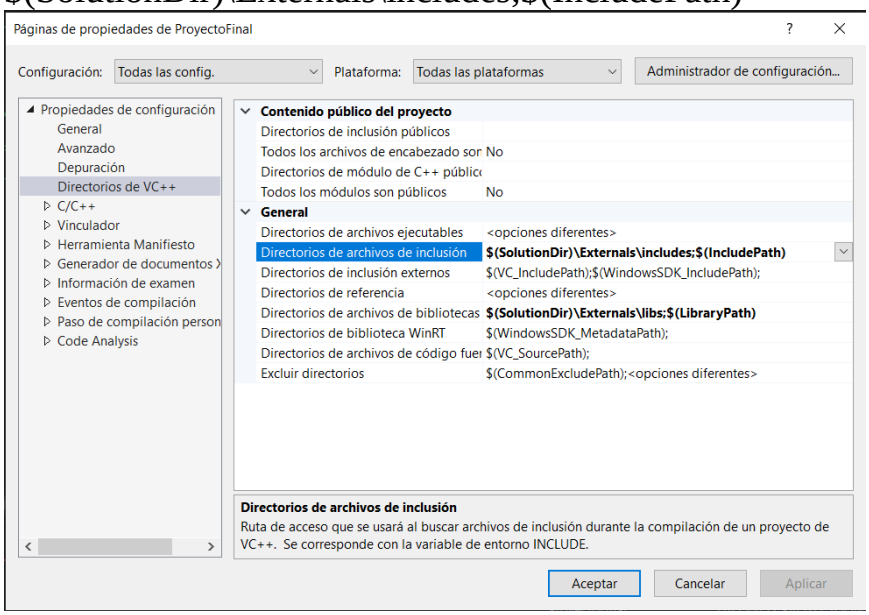
Dentro de nuestro proyecto se creó la carpeta media, donde insertaremos nuestras canciones:



Por último, del .zip en bin/win32-visualStudio se extraen los archivos .dll, los cuales son colocados en nuestro proyecto en el siguiente directorio:

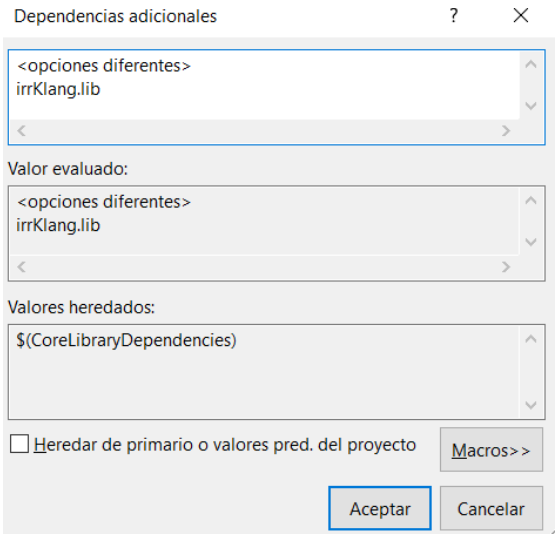


Para la configuración dentro de Visual, abrimos las propiedades del proyecto, seleccionamos Directorios de VC++, y luego en Directorios de archivos de inclusión, y se pega la siguiente línea:
$(SolutionDir)\Externals\includes;$(IncludePath)



Ahora vamos a Vinculador/Entrada/Dependencias adicionales, y pegamos el texto:

irrKlang.lib



Damos aplicar, y estará listo para usar Irrklang en el código.

Para implementar la música en el código, importamos los .h y declaramos el namespace:

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

Ahora creamos 2 “dispositivos”, uno en 2D y otro en 3D con la opción habilitada para que se reproduzcan en loop, y haremos que el audio 3D se coloque en la coordenada (3,0,0) con el fin de escucharla “a lo lejos”:

```
//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);
```

Ahora simplemente colocamos la posición desde donde escuchamos los audios, y la distancia mínima a la que se escuchará el audio 3D:

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

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

Por último, liberamos la memoria de los dispositivos:

```
glfwTerminate();

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

return 0;
```

Esto es colocado dentro de la función main en el ciclo while de la ventana, con el fin de que la música comience cuando se cargue la mesa de pinball, y termine cuando se cierre la ventana.

Diccionario de Variables

bool active; Se utilizó para la animación de los flippers

float rotFlipperIzq = 0.0f; Rotacion de flipper 1

float rotFlipperDer = 0.0f; Rotacion de flipper 2

float rotFlipperArr = 0.0f; Rotacion de flipper 3

float rotspot = 0.0f; Inicializacion de la rotacion para la figura de spot

float rotationSpeed = 45.0f; Grados por segundo para la velocidad del giro

bool anim; Se utilizó para la animación de los flippers

//animacion bumpers adelante

float logoBumpHeight = 0.35f; Altura máxima de movimiento de los bumpers

float logoBumpSpeed = 1.5f; Velocidad de movimiento

const float maxYLimit = -0.005f; Máxima y negativa a la que puede llegar

const float maxYLimitd = -0.005f; Mismo caso pero para el bumper derecho

//animacion bumper atras

const float logoBumpSpeedatras = 2.0f; Velocidad de la animación para el bumper de atrás

const float logoBumpHeightatras = 0.35f; Altura máxima de la animación

const float initialPhase = 1.5f; Fase inicial para empezar en un punto diferente

Bool ruta1, ruta2; Bandera que nos permiten cambiar entre las rutas de la canica 2

Bool a1, a2, a3, a4, a5, a6, a7, a8 = false; Fases de movimiento de la canica 2, animada sin keyframes

Bool anim2 = true; bool anim3 = false; animación que simula el caminar del avatar

Float rot2 = 0.0f; ángulo en radianes para rotar las extremidades del avatar, será utilizada en forma positiva y negativa para que se muevan de manera más realista

//animacion base miles

float timemiles = 0.0f; Inicialización del tiempo para la base de Miles

float speedmiles = 0.5f; Velocidad de oscilación

float maxHeightmiles = 1.0f; Máxima altura a la que puede llegar la base

float heightmiles = 0.0f; Inicio de la animación a dicha altura

//animacion base gwen

float timegwen = 0.0f; Inicialización tiempo para la base de Gwen

float speedgwen = 0.5f; Velocidad de oscilación

float maxHeightgwen = 1.0f; Máxima altura para la base de Gwen

float heightgwen = 0.0f; Inicio de la animación a esta altura

//animacion palanca

float despPal = 0.0f;

float escPal = 1.0f;

float despPal2 = 0.0f;

float escPal2 = 1.0;

//animacion oHara

float posX; //Variable para PosicionX
float posY; //Variable para PosicionY

float posZ; //Variable para PosicionZ

float incX; //Variable para IncrementoX

float incY; //Variable para IncrementoY

float incZ; //Variable para IncrementoZ

bool anim2 = true, animacion = false; //banderas que se utilizan para determinar si la animación se activa o no, y cómo se activará

float rot = 90.0f, rot2 = 0.0f, rot3 = 0.0f; //se rota al personaje 90 grados para orientarlo hacia la maquina, y las otras 2 funcionan para simular el movimiento de las extremidadws del avatar

//animacion canica 2

float xcanica2, ycanica2, zcanica2; //funcionan para realizar la animacion tradicional de la canica

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

float desplazamientoY = -1.0f; Variable para ajustar la posición de la SpotLight

float intensidadBrillo = 1.5f; Variable para aumentar el brillo de las PointLight

//Animacion Spider escalando

bool anim2 = true, anim3 = false; Booleano para activar las animaciones del spider

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; Declaracion de las rotaciones para cada una de las partes de este spiderman

//animacion botones

float traslacionBotonZ = 0.0f; Variable para la traslacion de los botones laterales de la maquina

const float traslacionMaxima = 0.04f; Maxima traslacion que podra tener el boton

float traslacionBotonX = 0.0f;

const float traslacionMaximader = -0.04f;