



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

INGENIERÍA EN COMPUTACIÓN

LABORATORIO DE COMPUTACIÓN GRÁFICA e
INTERACCIÓN HUMANO COMPUTADORA



REPORTE DE PRÁCTICA N° 09

NOMBRE COMPLETO: Gabriel Patricio Balam Flores

N° de Cuenta: 320280324

GRUPO DE LABORATORIO: 03

GRUPO DE TEORÍA: 04

SEMESTRE 2026-1

FECHA DE ENTREGA LÍMITE: 29/oct

CALIFICACIÓN: _____

REPORTE DE PRÁCTICA

1.- Ejecución de los ejercicios que se dejaron, comentar cada uno y capturas de pantalla de bloques de código generados y de ejecución del programa.

Ejercicio #01

Este ejercicio pedía separar el arco de las puertas y del letrero. En mi caso, exporté todo como un solo modelo, por lo que fue necesario separar el arco de las puertas en Blender. En cuanto al cartel, opté por crearlo directamente desde OpenGL en lugar de exportarlo desde Blender, ya que necesitaba modificar sus texturas en tiempo de ejecución.

Otro punto importante a mencionar es por qué importé tres puertas en lugar de una sola, si el modelo es el mismo para las tres. Aunque las puertas parecen idénticas, cada una tiene un origen distinto, lo cual permite que la rotación y la estética sean las adecuadas.

Importar / crear modelos a OpenGL

```
Model Puerta_reja_arco;  
Model Puerta_reja_derecha;  
Model Puerta_reja_central;  
Model Puerta_reja_piso;  
Model Puerta_reja_puerta_izquierda;  
Model Puerta_reja_puerta_derecha;
```

```
Puerta_reja_arco = Model();  
Puerta_reja_arco.LoadModel("Models/Puerta_reja_arco.obj");  
Puerta_reja_derecha = Model();  
Puerta_reja_derecha.LoadModel("Models/Puerta_reja_derecha.obj");  
Puerta_reja_central = Model();  
Puerta_reja_central.LoadModel("Models/Puerta_reja_puerta_central.obj");  
Puerta_reja_piso = Model();  
Puerta_reja_piso.LoadModel("Models/Puerta_reja_piso.obj");  
Puerta_reja_puerta_izquierda = Model();  
Puerta_reja_puerta_izquierda.LoadModel("Models/Puerta_reja_puerta_izquierda.obj");  
Puerta_reja_puerta_derecha = Model();  
Puerta_reja_puerta_derecha.LoadModel("Models/Puerta_reja_puerta_derecha.obj");
```

```
GLfloat verticesCartel[] = {  
    //  x      y      z      u      v      nx      ny      nz  
    -2.5f, -1.0f, 0.0f, 0.0f, 0.875f, 0.0f, 0.0f, -1.0f,  
    2.5f, -1.0f, 0.0f, 0.22f, 0.875f, 0.0f, 0.0f, -1.0f,  
    2.5f, 1.0f, 0.0f, 0.22f, 1.0f, 0.0f, 0.0f, -1.0f,  
    -2.5f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, -1.0f  
};
```

```
unsigned int cartelIndices[] = {  
    0, 1, 2,  
    0, 3, 2,  
};
```

```
Mesh* obj8 = new Mesh();  
obj8->CreateMesh(verticesCartel, cartelIndices, 32, 6);  
meshList.push_back(obj8);
```

Ejercicio #02

Este ejercicio consistía en modificar la textura aplicada al cartel para mostrar el mensaje “Proyecto CGEIHC” desplazándose en bucle. El primer paso fue crear la textura; en mi caso, elaboré la imagen en Word y la edité en GIMP para hacerla compatible con OpenGL.




El siguiente y último paso fue implementar la lógica para generar la animación de desplazamiento. Este proceso ya se había trabajado en el ejercicio de la práctica, por lo que no resultó complicado.

Resultado en el video 01

Animación del cartel	Implementación
<pre>tiempoAcumulado += deltaTime; if (tiempoAcumulado >= 11.0f) { toffsetcartelu += 0.06; if (toffsetcartelu >= 1.0) { toffsetcartelu = 0.0; } tiempoAcumulado = 0.0f; } toffsetcartelv = 0.0f;</pre>	<pre>// Para el cartel de la puerta ----- model = modelaux; toffset = glm::vec2(toffsetcartelu, toffsetcartelv); model = glm::translate(model, glm::vec3(0.0f, 17.19f, 0.0f)); glUniform2fv(uniformTextureOffset, 1, glm::value_ptr(toffset)); glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); hk_font_Proyecto_GCEIHC.UseTexture(); meshList[7]->RenderMesh();</pre>

Ejercicio #03

Este ejercicio consistía en agregar una animación de apertura y cierre a las puertas de la reja. En mi caso, mi reja cuenta con tres puertas; por estética, decidí que las dos laterales se abrieran mediante rotación hacia la izquierda y hacia la derecha, respectivamente, mientras que la puerta central se abría hacia abajo. Al plantearle al profesor el funcionamiento de esta última puerta, me indicó que debía agregar una textura en el piso para simular un agujero por donde entrara la puerta y evitar que pareciera que atraviesa el suelo. Por ello, el primer paso fue añadir este plano con su respectiva textura.

Textura	Implementación
	<pre>// Piso de la puerta ----- model = modelaux; model = glm::translate(model, glm::vec3(0.0f, 0.01f, 0.0f)); glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); Puerta_reja_piso.RenderModel();</pre>

La creación e implementación de la animación fue sencilla, ya que se siguió el mismo procedimiento que con el dragón, utilizando un par de banderas y condiciones anidadas.

Resultado en el video 02

Animación de las puertas [C]

```
// Animación de la puerta
if (mainWindow.getPuertaAbriendose())
{
    if (mainWindow.getPuertaCerrada())
    {
        if (bajadaPuerta >= -8.17f)
        {
            bajadaPuerta -= 0.1f * deltaTime;
            rotacionPuerta += 1.46879f * deltaTime;
        }
        else
        {
            mainWindow.setPuertaAbriendose();
            mainWindow.setPuertaCerrada();
        }
    }
}
```

```
// Puerta
GLboolean getPuertaCerrada() { return puertaCerrada; }
GLboolean getPuertaAbriendose() { return puertaAbriendose; }
void setPuertaCerrada() { puertaCerrada = !puertaCerrada; }
void setPuertaAbriendose() { puertaAbriendose = !puertaAbriendose; }

if (key == GLFW_KEY_C && action == GLFW_PRESS)
{
    if (!theWindow->puertaAbriendose)
    {
        theWindow->puertaAbriendose = true;
    }
}
```

```

else
{
    if (bajadaPuerta <= 0.0f)
    {
        bajadaPuerta += 0.1f * deltaTime;
        rotacionPuerta -= 1.46879f * deltaTime;
    }
    else
    {
        mainWindow.setPuertaAbriendose();
        mainWindow.setPuertaCerrada();
    }
}
}

```

Implementación

```

// Parte central de la puerta -----
model = modelaux;
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.0f));
model = glm::translate(model, glm::vec3(0.0f, 0.0f + bajadaPuerta, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Puerta_reja_central.RenderModel();

// Piso de la puerta -----
model = modelaux;
model = glm::translate(model, glm::vec3(0.0f, 0.01f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Puerta_reja_piso.RenderModel();

// Parte central derecha de la puerta -----
model = modelaux;
model = glm::translate(model, glm::vec3(8.25f, 0.0f, 0.0f));
model = glm::rotate(model, -1 * rotacionPuerta * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Puerta_reja_puerta_derecha.RenderModel();

```

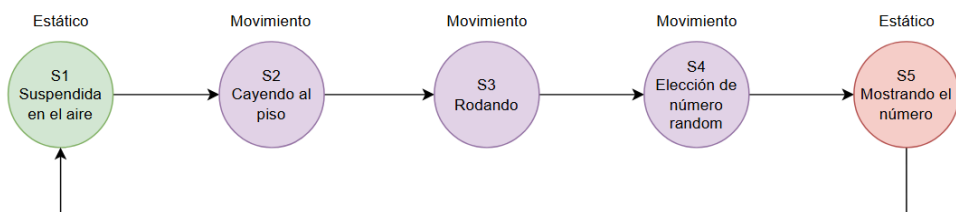
Importante:

- La puerta se abre con la tecla [C] y permanece abierta hasta que se vuelva a presionar la misma tecla.
- Para lograr la sincronización de las tres puertas, primero tomé como referencia la puerta que se abre hacia abajo, de la cual obtuve su desplazamiento y los incrementos por actualización. Posteriormente, para las puertas que rotan, calculé el desplazamiento en grados que debían realizar e hice una regla de tres para determinar el valor que debía sumarse a deltaTime, de modo que todas llegaran a su posición final al mismo tiempo.

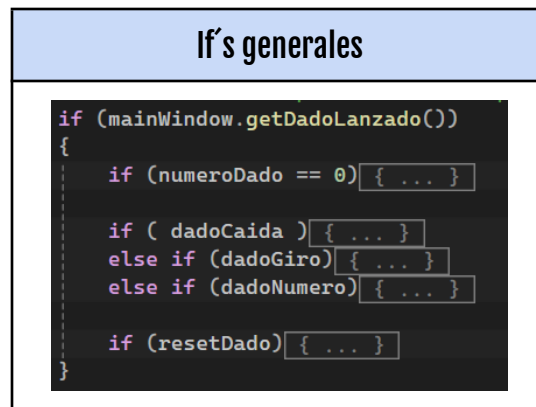
Ejercicio Extra

Sin duda, este fue el ejercicio más difícil y tardado de toda la materia. La premisa es sencilla: crear la animación de un dado de ocho caras que ruede y muestre una cara completamente aleatoria en cada lanzamiento. Sin embargo, la implementación resulta realmente laboriosa.

La animación puede interpretarse como un autómata de estados finitos:

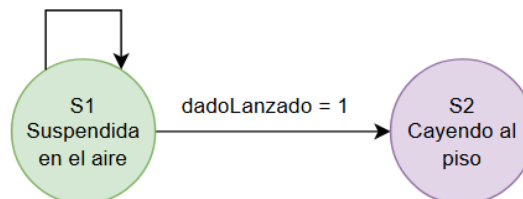


Estos estados se pueden traducir en los siguientes if's:



- El primer if detecta el momento en que se presiona la tecla [V], iniciando así la animación.

dadoLanzado = 0



- El segundo *if* determina el número aleatorio que se mostrará al final del lanzamiento.

Número random

```

if (numeroDado == 0)
{
    numeroDado = dist(gen);
    //numeroDado = 8;
}
```

- El tercer *if* controla la animación de caída. Esta parte es relativamente sencilla: consiste en una traslación en el eje -y acompañada de una ligera rotación del dado, de modo que parezca chocar con el piso de forma plana.

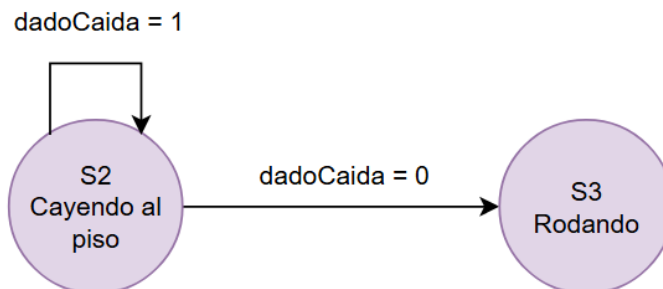
Cayendo al piso

```

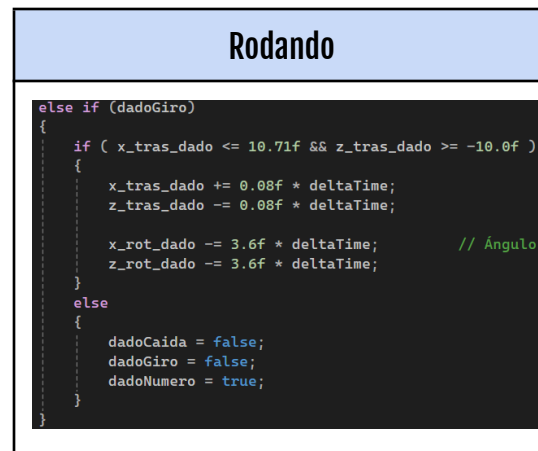
if ( dadoCaída )
{
    if ( y_tras_dado >= -0.3f)
    {
        x_tras_dado += 0.035f * deltaTime;
        y_tras_dado -= 0.5f * deltaTime;
        z_tras_dado += 0.035f * deltaTime;

        z_rot_dado += 2.25f * deltaTime;
    }
    else
    {
        dadoCaída = false;
        dadoGiro = true;
    }
}
```

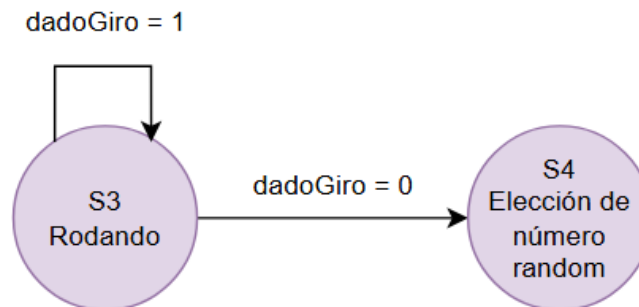
Para sincronizar ambas modificaciones, utilicé el mismo cálculo con regla de tres que en el ejercicio anterior.



- El tercer if controla la animación de rotación. Esta también es sencilla, consiste en una traslación en el eje x y en -z y una rotación en x y z.



De igual forma sincronicé ambas transformaciones con una regla de tres.



- El cuarto if fue el más tedioso de implementar. Intenté varias estrategias: calcular el ángulo y la rotación de cada número, obtener los ángulos en Blender y asignarlos a una traslación mediante regla de tres. Finalmente, la solución que funcionó fue determinar los ángulos manualmente en OpenGL y asignar traslaciones específicas para simular la sincronización y que la animación se viera natural.

Este método provoca que algunas rotaciones no se vean del todo naturales, ya que en ciertos números la rotación cambia completamente de sentido. Sin embargo, es la única solución que se me ocurrió para lograr el efecto deseado.

Elección del número random

```

else if (numeroDado == 1) // Se ve bien
{
    /* ... */

    if (numeroDado == 1) // Se ve bien
    {
        if (x_rot_dado >= -449.9711f)
        {
            x_rot_dado -= 1.0f * deltaTime;
        }
        else
            x_dado = true;

        if (y_rot_dado <= 385.0f)
        {
            x_tras_dado += 0.08f * deltaTime;
            z_tras_dado -= 0.08f * deltaTime;
            y_rot_dado += 5.0f * deltaTime;
        }
        else
            y_dado = true;

        if (z_rot_dado >= -483.531158f)
        {
            z_rot_dado -= 1.0f * deltaTime;
        }
        else
            z_dado = true;

        //x_rot_dado = -449.429535f;
        //y_rot_dado = -54.0f;
        //z_rot_dado = -482.97052f;
    }
    else if (numeroDado == 2) // Se ve bien
    {
        if (x_rot_dado <= -449.9711f)
            x_rot_dado += 1.0f * deltaTime;
        else
            x_dado = true;

        if (y_rot_dado <= 235.0f)
        {
            x_tras_dado += 0.08f * deltaTime;
            z_tras_dado -= 0.08f * deltaTime;
            y_rot_dado += 5.0f * deltaTime;
        }
        else
            y_dado = true;

        if (z_rot_dado >= -483.531158f)
            z_rot_dado -= 1.0f * deltaTime;
        else
            z_dado = true;

        //x_rot_dado = -449.545654f;
        //y_rot_dado = -125.0f;
        //z_rot_dado = -482.921082f;
    }
}

else if (numeroDado == 3) // Se ve bien
{
    if (x_rot_dado >= -684.532532f)
    {
        x_tras_dado += 0.08f * deltaTime;
        z_tras_dado -= 0.08f * deltaTime;
        x_rot_dado -= 5.0f * deltaTime;
    }
    else
        x_dado = true;

    y_dado = true; // no cambia Y en este

    if (z_rot_dado <= -483.163208f)
        z_rot_dado += 1.0f * deltaTime;
    else
        z_dado = true;

    //x_rot_dado = -324.548835f;
    //z_rot_dado = -483.163208f;
}
else if (numeroDado == 4) // Se ve bien
{
    if (x_rot_dado >= -759.619324f)
    {
        x_tras_dado += 0.08f * deltaTime;
        z_tras_dado -= 0.08f * deltaTime;
        x_rot_dado -= 5.0f * deltaTime;
    }
    else
        x_dado = true;

    y_dado = true;

    if (z_rot_dado <= -482.953613f)
        z_rot_dado += 1.0f * deltaTime;
    else
        z_dado = true;

    //x_rot_dado = -394.65f;
    //z_rot_dado = -483.2f;
}
else if (numeroDado == 5) // Se ve bien
{
    if (x_rot_dado >= -586.511292f)
    {
        x_tras_dado += 0.08f * deltaTime;
        z_tras_dado -= 0.08f * deltaTime;
        x_rot_dado -= 5.0f * deltaTime;
    }
    else
        x_dado = true;

    y_dado = true;

    if (z_rot_dado <= -482.999268f)
        z_rot_dado += 1.0f * deltaTime;
    else
        z_dado = true;

    //x_rot_dado = -586.511292f;
    //y_rot_dado = 0.0f;
    //z_rot_dado = -482.999268f;
}

else if (numeroDado == 6) // Se ve bien
{
    if (x_rot_dado >= -574.518498f)
    {
        x_tras_dado += 0.08f * deltaTime;
        z_tras_dado -= 0.08f * deltaTime;
        x_rot_dado -= 5.0f * deltaTime;
    }
    else
        x_dado = true;

    y_dado = true;

    if (z_rot_dado <= -483.035675f)
        z_rot_dado += 1.0f * deltaTime;
    else
        z_dado = true;

    //x_rot_dado = -574.518498f;
    //z_rot_dado = -483.035675f;
}
else if (numeroDado == 7) // Se ve bien
{
    if (x_rot_dado <= -449.93457f)
        x_rot_dado += 1.0f * deltaTime;
    else
        x_dado = true;

    if (y_rot_dado <= 55.0f)
    {
        x_tras_dado += 0.08f * deltaTime;
        z_tras_dado -= 0.08f * deltaTime;
        y_rot_dado += 2.5f * deltaTime;
    }
    else
        y_dado = true;

    if (z_rot_dado <= -483.0587324f)
        z_rot_dado += 1.0f * deltaTime;
    else
        z_dado = true;

    //x_rot_dado = -449.93457f;
    //y_rot_dado = 55.0f;
    //z_rot_dado = -483.0587324f;
}
else if (numeroDado == 8)
{
    if (x_rot_dado <= -489.787201f)
        x_rot_dado += 1.0f * deltaTime;
    else
        x_dado = true;

    if (y_rot_dado <= 125.0f)
    {
        x_tras_dado += 0.08f * deltaTime;
        z_tras_dado -= 0.08f * deltaTime;
        y_rot_dado += 4.0f * deltaTime;
    }
    else
        y_dado = true;

    if (z_rot_dado <= -483.076813f)
        z_rot_dado += 1.0f * deltaTime;
    else
        z_dado = true;

    //x_rot_dado = -489.787201f;
    //y_rot_dado = 125.0f;
    //z_rot_dado = -483.076813f;
}

if (x_dado && y_dado && z_dado)
{
    tiempoNumero += 0.5f * deltaTime;

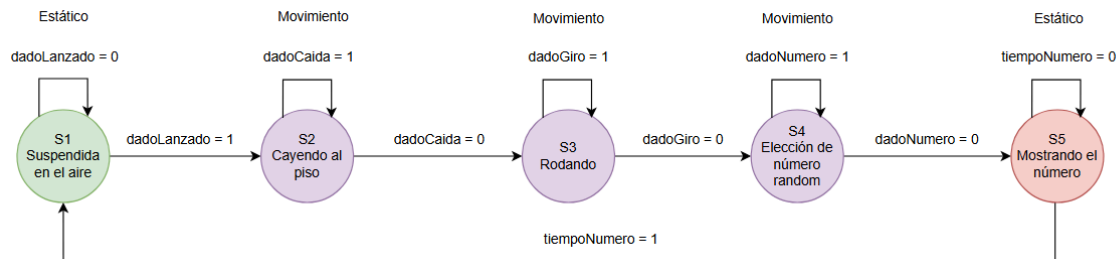
    if (tiempoNumero >= 50.0f)
    {
        dadoCaida = false;
        dadoGiro = false;
        dadoNumero = false;
        resetDado = true;
        tiempoNumero = 0.0f;
        x_dado = false;
        y_dado = false;
        z_dado = false;
    }
}

```

Como se puede ver en las imágenes, creé un `if` para cada cara del dado, lo que me permitió asignar la rotación y traslación específicas en cada caso. Como mencioné antes, no fue posible sincronizar todas las rotaciones y traslaciones de manera perfecta, por lo que, para determinar cuándo terminan, creé tres banderas que indican el fin de la rotación en cada eje y permiten avanzar al siguiente estado. El último `if` se encarga de dar una pequeña pausa para que se pueda observar el resultado del lanzamiento.

- El último if se encarga de reiniciar todos los valores a sus estados originales.

En el siguiente autómata se muestra en términos muy generales el proceso de la animación.



Resultado en el video 03

Variables y funciones auxiliares

```

// Dado
float x_rot_dado = 0.0f;
float y_rot_dado = 0.0f;
float z_rot_dado = 0.0f;

float x_tras_dado = 0.0f;
float y_tras_dado = 10.0f;
float z_tras_dado = 0.0f;

float tiempoInicio = 0.0f;
bool dadoCaída = true;
bool dadoGiro = false;
bool dadoNumero = false;
bool resetDado = false;

int numeroDado = 0;
float tiempoNumero = 0.0f;

bool x_dado = false;
bool y_dado = false;
bool z_dado = false;

```

```

// Dado
GLboolean getDadoInicio() { return dadoInicio; }
GLboolean getDadoLanzado() { return dadoLanzado; }
void setDadoInicio() { dadoInicio = !dadoInicio; }
void setDadoLanzado() { dadoLanzado = !dadoLanzado; }

```

```

if (key == GLFW_KEY_V && action == GLFW_PRESS)
{
    if (!theWindow->dadoLanzado)
    {
        theWindow->dadoLanzado = true;
    }
}

```

Implementación

```

// Dado 8 -----
model = glm::mat4(1.0);
toffset = glm::vec2(0.0, 0.0);
model = glm::translate(model, glm::vec3(x_tras_dado, y_tras_dado, z_tras_dado));
model = glm::scale(model, glm::vec3(4.0f, 4.0f, 4.0f));

model = glm::rotate(model, toRadians * x_rot_dado, glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, toRadians * y_rot_dado, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, toRadians * z_rot_dado, glm::vec3(0.0f, 0.0f, 1.0f));

glUniform2fv(uniformTextureOffset, 1, glm::value_ptr(toffset));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
dado8Texture.UseTexture();
meshList[8]->RenderMesh();

```

2.- Liste los problemas que tuvo a la hora de hacer estos ejercicios y si los resolvió explicar cómo fue, en caso de error adjuntar captura de pantalla.

Contratiempos

Para los ejercicios obligatorios, el único contratiempo fue pensar en la lógica para las animaciones; sin embargo, como ya se habían realizado animaciones similares en ejercicios anteriores, no resultó un problema mayor.

En el ejercicio opcional, el desafío fue lograr que el número elegido aleatoriamente apareciera de forma natural. Como mencioné antes, probé varias estrategias, utilizando herramientas como Blender y Excel para replicar la animación y facilitar el proceso, pero no tuve éxito. La solución que implementé no es la más estética, pero considero que es funcional y adecuada para mi nivel de conocimientos.

3.- Conclusión:

- a. Los ejercicios del reporte: Complejidad, Explicación.
 - b. Comentarios generales.
 - c. Conclusión
1. Bibliografía en formato APA

Conclusión

Esta práctica me ayudó a comprender la lógica detrás de las animaciones, tanto las sencillas como las que involucran varios estados. Además, me permitió entender cómo implementar las transformaciones de los distintos elementos y cómo sincronizarlas para que la animación se vea lo más natural posible.

Bibliografía

- Página principal. (26 de septiembre de 2018). *Wiki de OpenGL* . Recuperado el 29 de agosto de 2025 de http://www.khronos.org/opengl/wiki/opengl/index.php?title=Main_Page&oldid=14430 .