Reporte de Práctica de Laboratorio de Computación Gráfica e Interacción Humano-Computadora

Información de la práctica

Nombre del alumno:	Martínez Pérez Brian Erik		
Número de Cuenta:	319049792	Práctica número:	9
Título de la práctica:	Animación básica, procedural y Animación por keyframes		
Fecha de entrega:	3 de noviembre de 2025		

Introducción: (descripción inicial sobre la práctica)

En esta práctica se implementan las técnicas fundamentales de animación por computadora en OpenGL, tomando en cuenta el movimiento continuo y suave hasta el control de secuencias complejas mediante la gestión de estados.

El objetivo es pasar del renderizado estático a la creación de escenas dinámicas e interactivas. El archivo Animación_Básica.cpp crea la primera animación simple, utilizando el *render loop* para actualizar las transformaciones de los objetos de manera continua en función del tiempo.

La práctica aplica dos métodos esenciales para el control de animaciones. El código KeyFrames.cpp demuestra la representación de los valores (posición, rotación, escala) para cada cuadro clave y utiliza la Interpolación para calcular automáticamente los cuadros intermedios.

El archivo Maquina de estados.cpp implementa un sistema de control más sofisticado. Una Máquina de Estados se utiliza para gestionar transiciones lógicas y complejas, permitiendo que la animación pase de un estado a otro solo cuando se cumplen ciertas condiciones, en lugar de ser una secuencia lineal.

Resumen Teórico: (introducción sobre los fundamentos teóricos en los que se basa la práctica y cuáles son los objetivos teóricos que persigue.)

deltaTime: es esencial para asegurar que la actualización de la escena sea independiente de la tasa de fotogramas (FPS), garantizando la fluidez del movimiento.

Keyframes: Es un punto de referencia en la línea de tiempo de una animación donde el artista define explícitamente el estado de un objeto. Un keyframe contiene todos los valores esenciales: posición, ángulo de rotación y escala.

Interpolación Lineal (Lerp): se utiliza para calcular automáticamente los valores intermedios (*in-betweening*) entre los estados de posición, rotación y escala definidos por el usuario, creando transiciones suaves.

Máquina de Estados: controla secuencias de animación complejas y transiciones lógicas entre diferentes acciones del objeto, basadas en condiciones o *inputs*, en lugar de un *script* fijo.

Objetivos:

- Implementar una animación básica para la pelota.
- Lograr dividir el modelo del perro, en sus diferentes extremidades.
- Implementar la animación por maquinas de estado, para el movimiento total del perro.
- Lograr crear una animación fluida con keyframes.

Reporte de Práctica de Laboratorio de Computación Gráfica e Interacción Humano-Computadora

Descripción de la práctica: (En esta sección se deben describir todos los pasos ejecutados durante la sesión práctica realizada en el laboratorio.)

Animación básica

Partiendo del proyecto de la práctica anterior, lo primero que realizamos fue el cargado de los archivos necesarios para poder aplicar las animaciones a los modelos que utilicemos, mantenemos los mismos archivos “lighting.frag”, “lighting.vs, ”lamp.frag“, “lamp.vs”, “modelLoading.frag” y “modelLoading.vs” en la sección de archivos de origen que se utilizaran como shaders. Además de agregar el código principal “Animacion_Basica.cpp” en la sección de Archivos de origen.

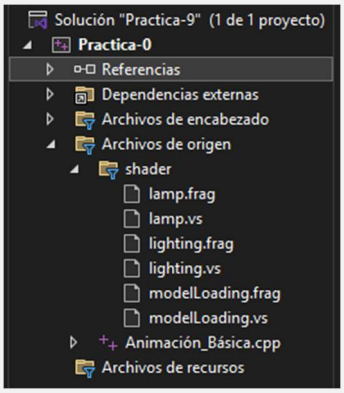

Imagen 1.1 - organización de los archivos utilizados en la solución del proyecto.

Ya dentro del código tenemos las variables de animación, las cuales nos ayudaran a realizar la rotación automática de la pelota, y el control para activar o desactivar animaciones.

```
//Anim
float rotBall = 0;
bool AnimBall = false;
```

Imagen 1.2 – variables de animación.

Se declara deltaTime, para un movimiento consistente en cualquier hardware y animaciones suaves independientes del framerate.

```
// Deltatime
GLfloat deltaTime = 0.0f;   // Time between current frame and last frame
GLfloat lastFrame = 0.0f;   // Time of last frame
```

Imagen 1.3 – declaración de deltaTime.

Cargamos los modelos que vamos a usar dentro de nuestro entorno, en este caso utilizamos un plano que será nuestro pasto, el perro rojo, y una pelota semitransparente, la pelota será el objeto que animaremos.

La transformación de la pelota se aplica en el eje Y (0.0f, 1.0f, 0.0f) con una rotación horizontal, el angulo (rotBall) se incrementa automáticamente y la conversión: “glm::radians()” de grados a radianes.

```
//Carga de modelo
view = camera.GetViewMatrix();
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Piso.Draw(lightingShader);

model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "transparency"), 0);
Dog.Draw(lightingShader);

model = glm::mat4(1);
glEnable(GL_BLEND);//Avtiva la funcionalidad para trabajar el canal alfa
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "transparency"), 1);
model = glm::rotate(model, glm::radians(rotBall), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Ball.Draw(lightingShader);
glDisable(GL_BLEND);  //Desactiva el canal alfa
glBindVertexArray(0);
```

Imagen 1.4 – carga de modelos.

Función de animación, donde se define la lógica del giro de la pelota. Además, se agrega una línea de impresión en la consola, donde observamos los valores que tomara la posición de la pelota.

```
void Animation() {
    if (AnimBall)
    {
        rotBall += 0.2f;
        printf("%f", rotBall);
    }
    else
    {
        //rotBall = 0.0f;
    }
}
```

Imagen 1.5 – definición del movimiento de la pelota

Se declaran las teclas para las 2 animaciones generadas, ESPACIO, alterna animación de color de luz, y N para alternar animación de rotación de pelota.

```
if (keys[GLFW_KEY_SPACE])
{
    active = !active;
    if (active)
    {
        Light1 = glm::vec3(1.0f, 1.0f, 0.0f);

    }
    else
    {
        Light1 = glm::vec3(0);//Cuado es solo un valor en los 3 vectores pueden dejar solo una componente
    }
}
if (keys[GLFW_KEY_N])
{
    AnimBall = !AnimBall;

}
```

Imagen 1.6 – teclas de animación.

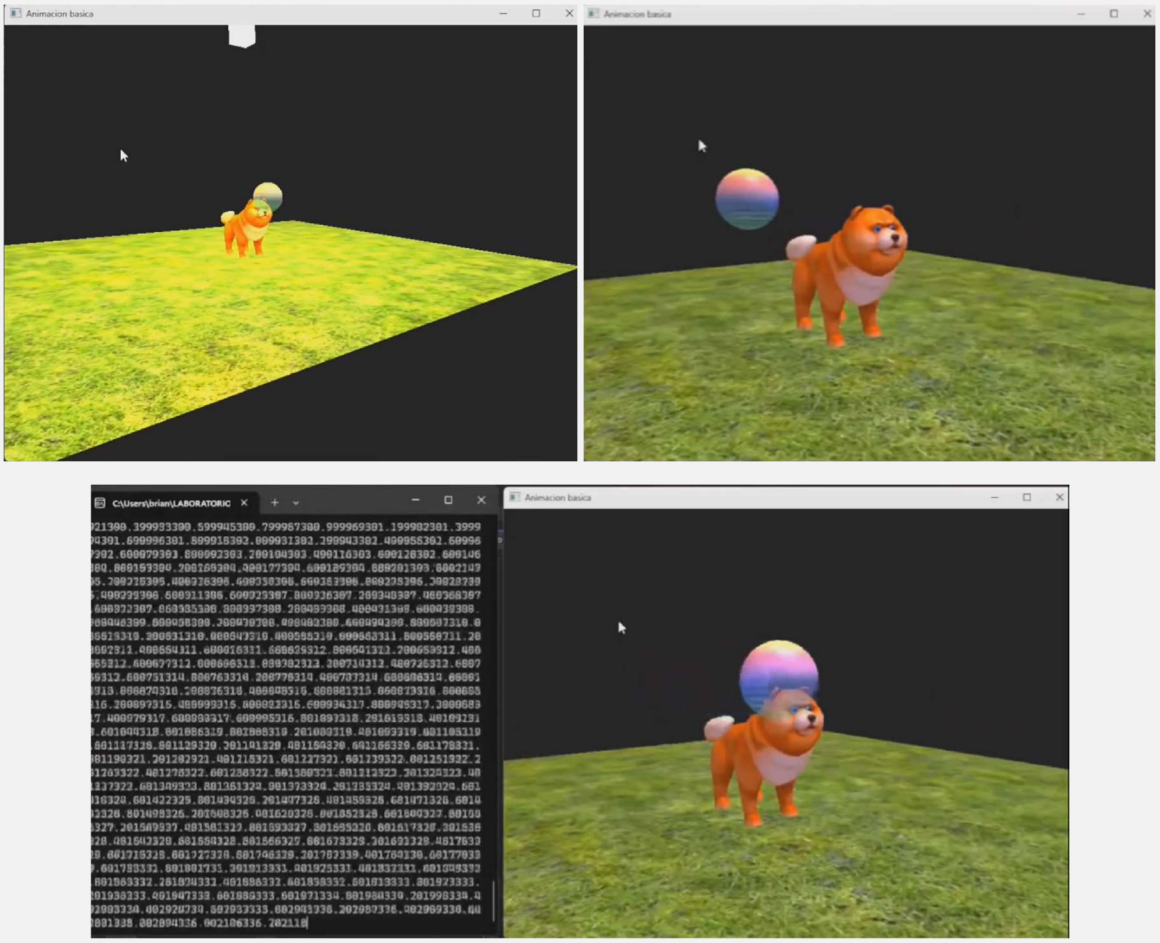

Imagen 1.7 – imágenes de la animación básica ejecutada

Animación con máquinas de estado

La siguiente animación se basa en sistemas de estado del modelo del perro cargado, lo que se resume en tener estados de animación (Reposo y caminata), transiciones entre estados y animación cíclica para caminar.

Para poder aplicar esta metodología, es necesario cargar el modelo del perro, por partes, ya que para este caso cada movimiento será independiente de cada extremidad del perro, el modelo se dividió en el cuerpo del perro que será el padre en el modelado jerárquico y los demás hijos, que serán la cabeza, cola y las 4 patas.

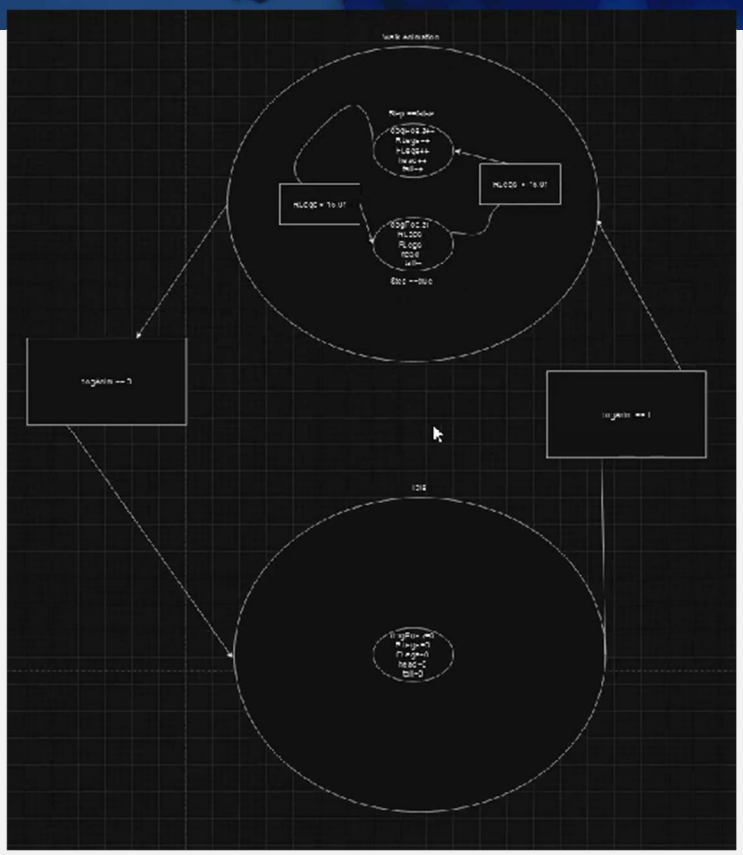

Imagen 2.1 – máquina de estados de la animación

```
//Anim
float rotBall = 0.0f;
bool AnimBall = false;
bool AnimDog = false;
float rotDog = 0.0f;
int dogAnim = 0;
float FLegs = 0.0f;
float RLegs = 0.0f;
float head = 0.0f;
float tail = 0.0f;
glm::vec3 dogPos (0.0f,0.0f,0.0f);
float dogRot = 0.0f;
bool step = false;
```

Imagen 2.2 – variables de animación para el control de los estados.

```
//models
Model DogBody((char*)"Models/DogBody.obj");
Model HeadDog((char*)"Models/HeadDog.obj");
Model DogTail((char*)"Models/TailDog.obj");
Model F_RightLeg((char*)"Models/F_RightLegDog.obj");
Model F_LeftLeg((char*)"Models/F_LeftLegDog.obj");
Model B_RightLeg((char*)"Models/B_RightLegDog.obj");
Model B_LeftLeg((char*)"Models/B_LeftLegDog.obj");
Model Piso((char*)"Models/piso.obj");
Model Ball((char*)"Models/ball.obj");
```

Imagen 2.3 – separación de las partes del perro en cada modelo.

```
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "transparency"), 0);
//Body
modelTemp= model = glm::translate(model, dogPos);
modelTemp= model = glm::rotate(model, glm::radians(dogRot), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
DogBody.Draw(lightingShader);
```

Imagen 2.4 – dibujo del cuerpo del perro (padre).

```
//Head
model = modelTemp;
model = glm::translate(model, glm::vec3(0.0f, 0.093f, 0.208f));
model = glm::rotate(model, glm::radians(head), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
HeadDog.Draw(lightingShader);
//Tail
model = modelTemp;
model = glm::translate(model, glm::vec3(0.0f, 0.026f, -0.288f));
model = glm::rotate(model, glm::radians(tail), glm::vec3(0.0f, 0.0f, -1.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
DogTail.Draw(lightingShader);
//Front Left Leg
model = modelTemp;
model = glm::translate(model, glm::vec3(0.112f, -0.044f, 0.074f));
model = glm::rotate(model, glm::radians(FLegs), glm::vec3(-1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
F_LeftLeg.Draw(lightingShader);
```

```
//Front Right Leg
model = modelTemp;
model = glm::translate(model, glm::vec3(-0.111f, -0.055f, 0.074f));
model = glm::rotate(model, glm::radians(FLegs), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
F_RightLeg.Draw(lightingShader);
//Back Left Leg
model = modelTemp;
model = glm::translate(model, glm::vec3(0.082f, -0.046, -0.218));
model = glm::rotate(model, glm::radians(RLegs), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
B_LeftLeg.Draw(lightingShader);
//Back Right Leg
model = modelTemp;
model = glm::translate(model, glm::vec3(-0.083f, -0.057f, -0.231f));
model = glm::rotate(model, glm::radians(RLegs), glm::vec3(-1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
B_RightLeg.Draw(lightingShader);
```

Imagen 2.5 – dibujo de las demás extremidades del perro (hijos).

```
if (keys[GLFW_KEY_B])
{
    dogAnim = 1;
}
```

Imagen 2.6 – tecla “B” para activar animación del movimiento del perro.

```
if (dogAnim==1) //walk animation
{
    if (!step) //state 1
    {
        RLegs += 0.03f;
        FLegs += 0.03f;
        head += 0.03f;
        tail += 0.03f;

        if (RLegs > 15.0f){ //condition
            step = true;
        }
    }
    else
    {
        RLegs -= 0.03f;
        FLegs -= 0.03f;
        head -= 0.03f;
        tail -= 0.03f;

        if (RLegs < -15.0f) { //condition
            step = false;
        }
    }
    dogPos.z += 0.0001;
    printf("/n%f", RLegs);

}
```

Imagen 2.7 – implementación de la lógica de la máquina de estados de la animación.

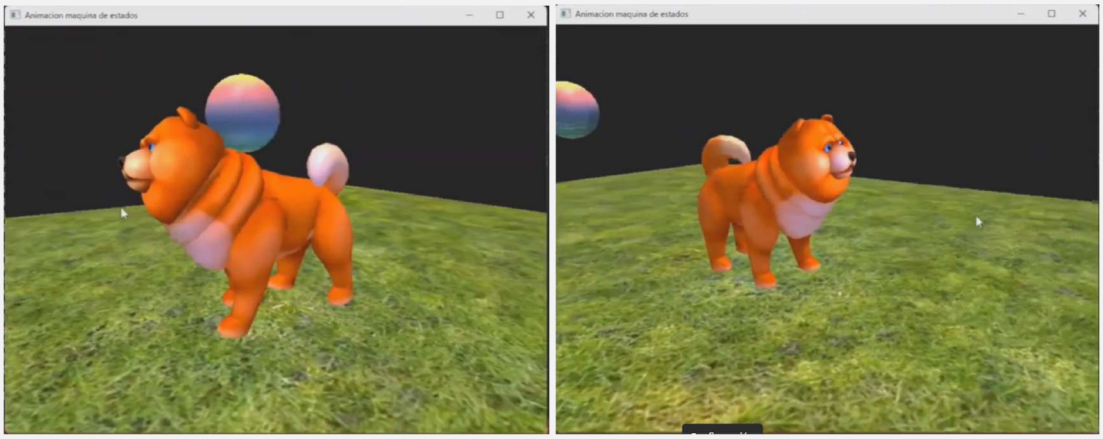

Imagen 2.8 – imágenes de la animación por máquina de estados ejecutada.

Animación por Keyframes

Este código implementa un sistema avanzado de keyframes que permite crear animaciones complejas mediante interpolación entre posiciones y rotaciones predefinidas.

Se definen las variables de control de los Keyframes. El índice para guardar keyframes, estado de reproducción, Keyframe actual en reproducción, pasos totales entre keyframes, paso actual entre keyframes.

```
#define MAX_FRAMES 9
int i_max_steps = 190;
int i_curr_steps = 0;
```

```
FRAME KeyFrame[MAX_FRAMES];
int FrameIndex = 0;
bool play = false;
int playIndex = 0;
```

Imagen 3.1 - variables de control de los Keyframes.

```
typedef struct _frame {

    float rotDog;
    float rotDogInc;
    float dogPosX;
    float dogPosY;
    float dogPosZ;
    float incX;
    float incY;
    float incZ;
    float head;
    float headInc;


}FRAME;
```

Imagen 3.2 – estructura de datos para los movimientos del perro.

```
void saveFrame(void)
{

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

    KeyFrame[FrameIndex].dogPosX = dogPosX;
    KeyFrame[FrameIndex].dogPosY = dogPosY;
    KeyFrame[FrameIndex].dogPosZ = dogPosZ;

    KeyFrame[FrameIndex].rotDog = rotDog;
    KeyFrame[FrameIndex].head = head;

    FrameIndex++;
}
```

Imagen 3.3 – función para guardar los valores del keyframe.

```
void resetElements(void)
{
    dogPosX = KeyFrame[0].dogPosX;
    dogPosY = KeyFrame[0].dogPosY;
    dogPosZ = KeyFrame[0].dogPosZ;
    head = KeyFrame[0].head;

    rotDog = KeyFrame[0].rotDog;


}
```

Imagen 3.4 – función para reiniciar a la posición inicial.

```
void interpolation(void)
{

    KeyFrame[playIndex].incX = (KeyFrame[playIndex + 1].dogPosX - KeyFrame[playIndex].dogPosX) / i_max_steps;
    KeyFrame[playIndex].incY = (KeyFrame[playIndex + 1].dogPosY - KeyFrame[playIndex].dogPosY) / i_max_steps;
    KeyFrame[playIndex].incZ = (KeyFrame[playIndex + 1].dogPosZ - KeyFrame[playIndex].dogPosZ) / i_max_steps;
    KeyFrame[playIndex].headInc = (KeyFrame[playIndex + 1].head - KeyFrame[playIndex].head) / i_max_steps;

    KeyFrame[playIndex].rotDogInc = (KeyFrame[playIndex + 1].rotDog - KeyFrame[playIndex].rotDog) / i_max_steps;

}
```

Imagen 3.5 – función para calcular la interpolación.

```
void Animation() {

    if (play)
    {
        if (i_curr_steps >= i_max_steps) //end of animation between frames?
        {
            playIndex++;
            if (playIndex > FrameIndex - 2) //end of total animation?
            {
                printf("termina anim\n");
                playIndex = 0;
                play = false;
            }
            else //Next frame interpolations
            {
                i_curr_steps = 0; //Reset counter
                //Interpolation
                interpolation();
            }
        }
        else
        {
            //Draw animation
            dogPosX += KeyFrame[playIndex].incX;
            dogPosY += KeyFrame[playIndex].incY;
            dogPosZ += KeyFrame[playIndex].incZ;
            head += KeyFrame[playIndex].headInc;
            rotDog += KeyFrame[playIndex].rotDogInc;

            i_curr_steps++;
        }

    }
}
```

Imagen 3.6 - Flujo de Animación con KeyFrames.

Para poder tener control en la ejecución de keyframes se utilizaron 2 teclas, la tecla “K” para guardar el keyframe actual, y la tecla “L” para reproducir la animación con los keyframes guardados hasta el momento.

```
if (keys[GLFW_KEY_L])
{
    if (play == false && (FrameIndex > 1))
    {

        resetElements();
        //First Interpolation
        interpolation();

        play = true;
        playIndex = 0;
        i_curr_steps = 0;
    }
    else
    {
        play = false;
    }

}

if (keys[GLFW_KEY_K])
{
    if (FrameIndex < MAX_FRAMES)
    {
        saveFrame();
    }

}
```

Imagen 3.7 - Teclas para KeyFrames.

```
//Dog Controls
if (keys[GLFW_KEY_4])
{

    head += 0.1f;

}
if (keys[GLFW_KEY_5])
{

    head -= 0.1f;

}

if (keys[GLFW_KEY_2])
{

        rotDog += 0.1f;

}

if (keys[GLFW_KEY_3])
{

        rotDog -= 0.1f;

}
```

```
if (keys[GLFW_KEY_H])
{
    dogPosZ += 0.001;
}

if (keys[GLFW_KEY_Y])
{
    dogPosZ -= 0.001;
}

if (keys[GLFW_KEY_G])
{
    dogPosX -= 0.001;
}

if (keys[GLFW_KEY_J])
{
    dogPosX += 0.001;
}
```

Imagen 3.8 – teclas para el movimiento del perro.

Pasos para generar los keyframes y la animación.

1. Posicionar manualmente el perro usando teclas 2,3,4,5 (giro de la cabeza), G,H,J,Y (movimiento del perro).
2. Guardar keyframe con tecla K (máximo 9 keyframes)
3. Mover a nueva posición y guardar otro keyframe
4. Repetir hasta tener todos los keyframes deseados
5. Reproducir animación con tecla L

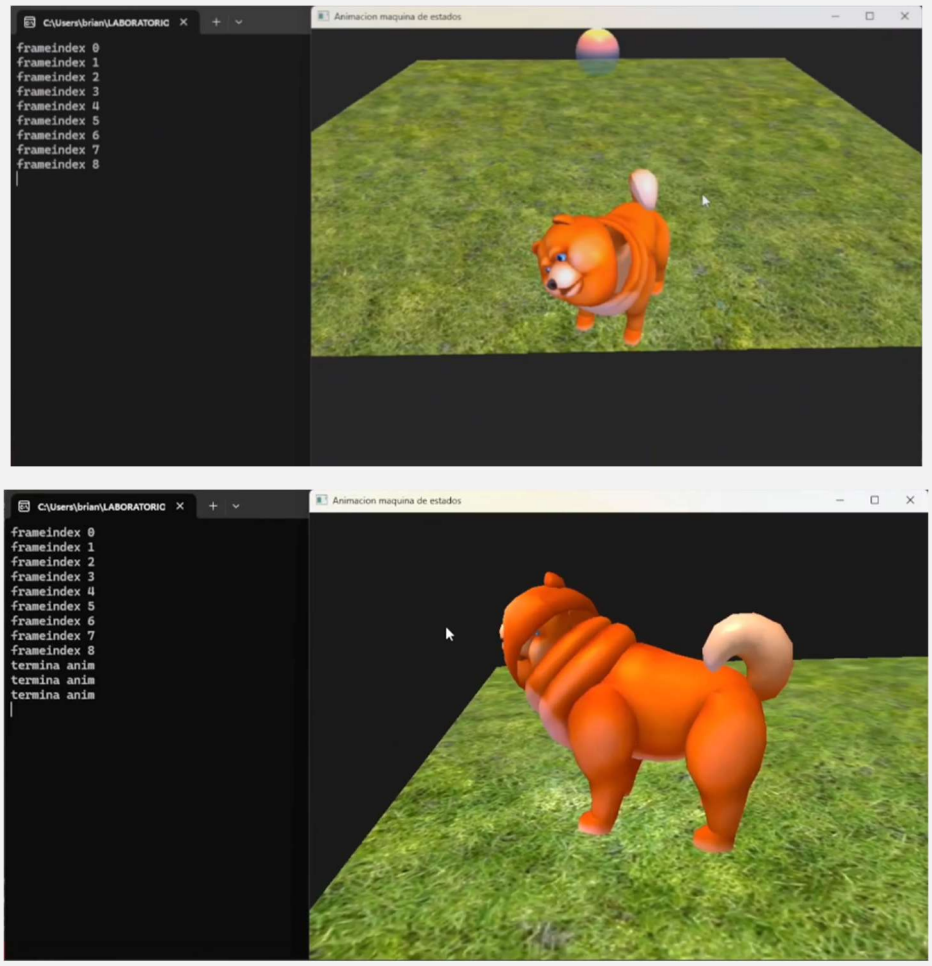

Imagen 3.9 - imágenes de la animación por keyframes ejecutada.

Video de la demostración de todas las animaciones: https://www.youtube.com/watch?v=ugK_sEF172A

Resultados: (explicar lo que se logró o aquello que no lograron realizar o comprender)

- Se logro implementar la primera animación básica fluida, mediante la actualización de la posición de la pelota. Creando la ilusión de que gira alrededor del perro.
- Se logro dividir el modelo del perro, para poder animar cada una de sus extremidades con los métodos restantes.
- Se logro implementar la animación por maquina de estados, simulando la acción de que el perro puede caminar.
- Por ultimo se logro comprender como funciona la animación por keyframes, además se logro implementar en open gl y utilizarla para generar una animación fluida con el movimiento del perro.

Conclusiones:

En esta practica logre comprender las características de las distintas técnicas de animación, para ello se implementaron en open gl, para generar animaciones fluidas. Esto nos ayuda a poder generar escenarios más dinámicos en nuestros proyectos, ya que los mismos usuarios pueden tener interacción con el entorno generado. Por último, encontré la utilidad en estas animaciones para mi proyecto de la materia. Donde puede crear la animación de una puerta que abre y se cierra. Además, para el giro como otra animación para el sol artificial, que se puede crear. En esta última animación pensada, se me ocurre como el giro que da el sol a la tierra, creando el día y la noche en mi escenario.

Bibliografía consultada:

Interactivas y Computación Gráfica, T. [@ArturoVMS]. (s/f-a). *Animación básica en OpenGL* [[Object Object]]. Youtube. Recuperado el 2 de noviembre de 2025, de https://www.youtube.com/watch?v=Iuybf6bTzhc

Interactivas y Computación Gráfica, T. [@ArturoVMS]. (s/f-b). *Animación con Máquinas de Estados en OpenGL: Movimiento de Objetos 3D [Tutorial Completo]* [[Object Object]]. Youtube. Recuperado el 2 de noviembre de 2025, de https://www.youtube.com/watch?v=B-mqD317HHM

Interactivas y Computación Gráfica, T. [@ArturoVMS]. (s/f-c). *Animación por Keyframes en OpenGL Movimiento y Rotación Suaves [Tutorial Completo]* [[Object Object]]. Youtube. Recuperado el 2 de noviembre de 2025, de https://www.youtube.com/watch?v=5l1A6Lxa3GQ