Línea 1: import pygame

- Importa la biblioteca pygame completa. Pygame es un conjunto de módulos de Python diseñados para escribir videojuegos, pero aquí se utiliza principalmente para crear una ventana, manejar eventos (como cerrar la ventana) y gestionar el bucle principal de la aplicación junto con OpenGL.

Línea 2: from pygame.locals import *

- Importa todas las constantes y nombres definidos en el módulo pygame.locals. Esto permite usar nombres como QUIT, DOUBLEBUF, OPENGL directamente sin el prefijo pygame.locals..

Línea 3: from OpenGL.GL import *

- Importa todos los nombres del módulo OpenGL.GL. Este módulo proporciona acceso a las funciones principales de la API de OpenGL, que se utilizan para el renderizado gráfico 3D (por ejemplo, glEnable, glClearColor, glBegin, glVertex3f, etc.).

Línea 4: from OpenGL.GLU import *

- Importa todos los nombres del módulo OpenGL.GLU. GLU (OpenGL Utility Library) proporciona funciones de utilidad de nivel superior construidas sobre OpenGL, como gluPerspective (para configurar la proyección de la cámara) y gluNewQuadric, gluSphere (para dibujar primitivas como esferas).

Línea 5: import numpy as np

- Importa la biblioteca numpy y le asigna el alias np. Aunque en este código no se utiliza numpy de forma explícita y directa como en el ejemplo anterior (por ejemplo, para matrices de vértices), es una biblioteca común en gráficos y podría ser útil para futuras expansiones o cálculos más complejos. En este código específico, su uso es implícito o no visible.

Línea 6: import math

- Importa el módulo math, que proporciona acceso a funciones matemáticas básicas, como sin (seno) y radians (para convertir grados a radianes), utilizadas aquí para calcular los ángulos de animación.

Líuna 7: (En blanco)

- Línea en blanco: Separa la sección de importaciones del resto del código para mayor claridad.

Línea 8: # ------- Utilidades OPENGL -------

- Comentario: Indica que las siguientes funciones son utilidades relacionadas con la configuración o el dibujo en OpenGL.

Línea 9: (En blanco)

- Línea en blanco: Espacio antes de la definición de la primera función de utilidad.

Línea 10: def init_opengl():

- Define una función llamada init_opengl que no toma argumentos. Esta función se encarga de la configuración inicial de algunos estados de OpenGL.

Línea 11: glEnable(GL_DEPTH_TEST)

- Dentro de la función init_opengl: Llama a glEnable con el argumento GL_DEPTH_TEST. Esto habilita el test de profundidad (o z-buffering), que asegura que los objetos más cercanos a la cámara oculten a los más lejanos, logrando una correcta representación 3D.

Línea 12: glEnable(GL_COLOR_MATERIAL)

- Dentro de la función init_opengl: Llama a glEnable con el argumento GL_COLOR_MATERIAL. Esta opción permite que las llamadas a glColor3f (y similares) afecten directamente el color del material de los objetos que se están dibujando, simplificando la asignación de colores.

Línea 13: glClearColor(0.1, 0.1, 0.1, 1)

- Dentro de la función init_opengl: Llama a glClearColor con los argumentos 0.1, 0.1, 0.1, 1. Esto establece el color que se usará para limpiar el buffer de color (el fondo de la pantalla). Los valores son (R, G, B, Alpha), por lo que define un color gris oscuro (0.1, 0.1, 0.1) y completamente opaco (1).

Línea 14: glLineWidth(5)

- Dentro de la función init_opengl: Llama a glLineWidth con el argumento 5. Esto establece el ancho de las líneas rasterizadas (como las que se dibujan con GL_LINES) a 5 píxeles.

Línea 15: (En blanco)

- Línea en blanco: Separa la función init_opengl de la siguiente función.

Línea 16: def draw_bone(length=1.0):

- Define una función llamada draw_bone que toma un argumento opcional length con un valor por defecto de 1.0. Esta función dibujará una representación simple de un "hueso".

Línea 17: glBegin(GL_LINES)

- Dentro de la función draw_bone: Llama a glBegin con el argumento GL_LINES. Esto inicia la definición de una o más primitivas de línea. Cada par de vértices definidos entre glBegin(GL_LINES) y glEnd() formará una línea.

Línea 18: glVertex3f(0, 0, 0)

- Dentro de la función draw_bone y del bloque glBegin(GL_LINES): Llama a glVertex3f para definir el primer vértice de la línea en las coordenadas (0, 0, 0) del sistema de coordenadas local actual.

Línea 19: glVertex3f(0, length, 0)

- Dentro de la función draw_bone y del bloque glBegin(GL_LINES): Llama a glVertex3f para definir el segundo vértice de la línea en las coordenadas (0, length, 0). Esto crea una línea vertical a lo largo del eje Y, desde el origen hasta la altura especificada por length.

Línea 20: glEnd()

- Dentro de la función draw_bone: Llama a glEnd(). Esto finaliza la definición de las primitivas de línea iniciadas con glBegin(GL_LINES).

Línea 21: # Dibuja una pequeña esfera en el extremo

- Dentro de la función draw_bone: Comentario que indica que la siguiente línea dibujará una esfera, probablemente para marcar el final o la articulación del hueso.

Línea 22: draw_sphere(0.1, 0, length, 0)

- Dentro de la función draw_bone: Llama a la función draw_sphere (definida más adelante) para dibujar una esfera.
    - 0.1: Es el radio de la esfera.
    - 0, length, 0: Son las coordenadas (x, y, z) donde se centrará la esfera. Esto la coloca en el extremo superior del hueso que se acaba de dibujar.

Línea 23: (En blanco)

- Línea en blanco: Separa la función draw_bone de la siguiente función.

Línea 24: def draw_sphere(radius, x, y, z):

- Define una función llamada draw_sphere que toma cuatro argumentos: radius (el radio de la esfera) y x, y, z (las coordenadas del centro de la esfera).

Línea 25: """ Esfera en una posición"""

- Dentro de la función draw_sphere: Un docstring simple que describe brevemente lo que hace la función.

Línea 26: glPushMatrix()

- Dentro de la función draw_sphere: Llama a glPushMatrix(). Esto guarda el estado actual de la matriz de transformación (modelview matrix). Es importante para que las transformaciones aplicadas dentro de esta función (como glTranslatef) no afecten a otros objetos dibujados posteriormente fuera de ella.

Línea 27: glTranslatef(x, y, z)

- Dentro de la función draw_sphere: Llama a glTranslatef(x, y, z). Esto traslada (mueve) el sistema de coordenadas local actual a la posición (x, y, z). Cualquier objeto dibujado después de esto se posicionará relativo a este nuevo origen.

Línea 28: quad = gluNewQuadric()

- Dentro de la función draw_sphere: Llama a gluNewQuadric() de la librería GLU. Esto crea un nuevo objeto "cuádrica", que es un objeto de utilidad usado por GLU para dibujar superficies curvas como esferas, cilindros y discos. El objeto cuádrica se asigna a la variable quad.

Línea 29: gluSphere(quad, radius, 8, 8)

- Dentro de la función draw_sphere: Llama a gluSphere de la librería GLU para dibujar una esfera.
    - quad: El objeto cuádrica creado anteriormente.
    - radius: El radio de la esfera, pasado como argumento a la función.
    - 8: El número de subdivisiones alrededor del eje Z (longitudes).

    - 8: El número de subdivisiones a lo largo del eje Z (latitudes).
    - Números más altos de subdivisiones producen esferas más suaves pero con mayor coste computacional.

Línea 30: glPopMatrix()

- Dentro de la función draw_sphere: Llama a glPopMatrix(). Esto restaura la matriz de transformación al estado en que estaba antes de la llamada a glPushMatrix() correspondiente. Así, la traslación aplicada para posicionar la esfera se deshace para el resto del código.

Línea 31: (En blanco)

- Línea en blanco: Separa las funciones de utilidad de dibujo de la función que dibuja el esqueleto.

Línea 32: # ------- ANIMACIÓN ESQUELÉTICA -------

- Comentario: Indica que la siguiente sección de código se relaciona con el dibujo de la animación del esqueleto.

Línea 33: (En blanco)

- Línea en blanco: Espacio antes de la definición de la función draw_skeleton.

Línea 34: def draw_skeleton(shoulder_angle, elbow_angle, wrist_angle):

- Define una función llamada draw_skeleton que toma tres argumentos: shoulder_angle, elbow_angle, y wrist_angle. Estos ángulos controlarán la rotación de las diferentes articulaciones del esqueleto.

Línea 35: glColor3f(0.5, 0.8, 1.0)

- Dentro de la función draw_skeleton: Llama a glColor3f para establecer el color actual de dibujo a un azul claro (RGB: 0.5, 0.8, 1.0). Todos los huesos dibujados por esta función tendrán este color, a menos que se cambie internamente.

Línea 36: (En blanco)

- Línea en blanco: Separa el establecimiento del color del dibujo de la primera articulación.

Línea 37: # Articulación 1: hombre

- Dentro de la función draw_skeleton: Comentario que identifica la primera articulación como el "hombro". "hombre" parece ser un error tipográfico, debería ser "hombro".

Línea 38: glPushMatrix()

- Dentro de la función draw_skeleton: Guarda la matriz de transformación actual. Esto es crucial para la animación jerárquica, ya que las transformaciones del hombro afectarán al codo y la muñeca, pero no queremos que afecten a otros objetos dibujados fuera de esta llamada a draw_skeleton (o antes del primer glPushMatrix de este bloque).

Línea 39: glRotatef(shoulder_angle, 0, 0, 1)

- Dentro de la función draw_skeleton: Aplica una rotación.
    - shoulder_angle: El ángulo de rotación en grados.
    - 0, 0, 1: El vector (x, y, z) que define el eje de rotación. En este caso, (0, 0, 1) significa que la rotación se realizará alrededor del eje Z.
- Esto rota el sistema de coordenadas para el primer hueso (hombro).

Línea 40: draw_bone(2.0)

- Dentro de la función draw_skeleton: Llama a la función draw_bone con una longitud de 2.0. Esto dibuja el primer segmento del esqueleto (el hueso del "hombro" o brazo superior) con la rotación aplicada previamente.

Línea 41: (En blanco)

- Línea en blanco: Separa la lógica del hombro de la del codo.

Línea 42: # Articulación 2: codo (relativa al hombro)

- Dentro de la función draw_skeleton: Comentario que indica que se va a dibujar la articulación del codo y que su posición y orientación son relativas al hombro.

Línea 43: glTranslatef(0, 2.0, 0)

- Dentro de la función draw_skeleton: Traslada el sistema de coordenadas local.
    - 0, 2.0, 0: Mueve el origen 2.0 unidades a lo largo del eje Y local. Dado que el primer hueso (draw_bone(2.0)) se dibujó desde (0,0,0) hasta (0,2.0,0) en su propio sistema de coordenadas (rotado), esta traslación

mueve el origen al extremo de ese primer hueso. Esto es la base de la animación jerárquica: la posición del codo depende de la posición y orientación del hombro.

Línea 44: glRotatef(elbow_angle, 0, 0, 1)

- Dentro de la función draw_skeleton: Aplica una rotación para la articulación del codo.
    - elbow_angle: El ángulo de rotación para el codo.
    - 0, 0, 1: Rota alrededor del eje Z local (que ya ha sido afectado por la rotación del hombro y la traslación).
- Esta rotación se aplica después de haberse trasladado al extremo del hueso anterior.

Línea 45: draw_bone(1.5)

- Dentro de la función draw_skeleton: Llama a draw_bone con una longitud de 1.5. Esto dibuja el segundo segmento del esqueleto (el hueso del "codo" o antebrazo) con la rotación y traslación relativas al hombro.

Línea 46: (En blanco)

- Línea en blanco: Separa la lógica del codo de la de la muñeca.

Línea 47: # Articulación 3: muñeca (relativa al codo)

- Dentro de la función draw_skeleton: Comentario que indica que se va a dibujar la articulación de la muñeca y que su posición y orientación son relativas al codo.

Línea 48: glTranslatef(0, 1.5, 0)

- Dentro de la función draw_skeleton: Traslada el sistema de coordenadas local.
    - 0, 1.5, 0: Mueve el origen 1.5 unidades a lo largo del eje Y local. Dado que el segundo hueso (draw_bone(1.5)) se dibujó desde (0,0,0) hasta (0,1.5,0) en su propio sistema de coordenadas, esta traslación mueve el origen al extremo de ese segundo hueso. Esto es la base de la animación jerárquica: la posición de la muñeca depende de la posición y orientación del codo.

Línea 49: glRotatef(wrist_angle, 0, 0, 1)

- Dentro de la función draw_skeleton: Aplica una rotación para la articulación de la muñeca.
    - wrist_angle: El ángulo de rotación para la muñeca.
    - 0, 0, 1: Rota alrededor del eje Z local.
- Esta rotación se aplica después de haberse trasladado al extremo del hueso anterior.

Línea 50: draw_bone(1.0)

- Dentro de la función draw_skeleton: Llama a draw_bone con una longitud de 1.0. Esto dibuja el tercer segmento del esqueleto (el hueso de la "muñeca" o mano) con la rotación y traslación relativas al codo.

Línea 51: (En blanco)

- Línea en blanco: Separa el último dibujo de hueso del cierre de la pila de matrices.

Línea 52: glPopMatrix()

- Dentro de la función draw_skeleton: Llama a glPopMatrix(). Esto restaura la matriz de transformación al estado en que estaba al entrar a draw_skeleton (antes del primer glPushMatrix). Esto asegura que las transformaciones aplicadas dentro de esta función (rotaciones y traslaciones jerárquicas) solo afecten al esqueleto y no a otros objetos dibujados fuera de esta función.

Línea 53: (En blanco)

- Línea en blanco: Separa la función draw_skeleton del bucle principal del programa.

Línea 54: # ------- LOOP PRINCIPAL -------

- Comentario: Indica que la siguiente sección de código contiene la función principal del programa que inicializa Pygame/OpenGL y ejecuta el bucle de renderizado.

Línea 55: (En blanco)

- Línea en blanco: Espacio antes de la definición de la función main.

Línea 56: def main():

- Define la función main, que encapsula la lógica principal de la aplicación.

Línea 57: pygame.init()

- Dentro de la función main: Llama a pygame.init(). Esta función inicializa todos los módulos de Pygame necesarios para que la aplicación funcione.

Línea 58: display = (900, 700)

- Dentro de la función main: Define una tupla display con las dimensiones de la ventana (ancho 900 píxeles, alto 700 píxeles).

Línea 59: pygame.display.set_mode(display, DOUBLEBUF|OPENGL)

- Dentro de la función main: Llama a pygame.display.set_mode().
    - display: Establece el tamaño de la ventana.
    - DOUBLEBUF|OPENGL: Son banderas (flags) que configuran el modo de visualización. DOUBLEBUF habilita el doble buffer (para renderizado suave sin parpadeos), y OPENGL le indica a Pygame que se va a usar OpenGL para el renderizado en esta ventana.

Línea 60: pygame.display.set_caption('Skeletal Animation 3D - Esqueleto Jerárquico')

- Dentro de la función main: Llama a pygame.display.set_caption() para establecer el título de la ventana de la aplicación.

Línea 61: (En blanco)

- Línea en blanco: Separa la configuración inicial de Pygame de la configuración de la cámara de OpenGL.

Línea 62: gluPerspective(45, display[0] / display[1], 0.1, 100)

- Dentro de la función main: Llama a gluPerspective() de la librería GLU para configurar la matriz de proyección (cámara) en OpenGL.
    - 45: El ángulo de campo de visión (FOV) vertical en grados.
    - display[0] / display[1]: La relación de aspecto (ancho/alto) de la ventana.
    - 0.1: El plano de recorte cercano (near clipping plane). Los objetos más cerca que esta distancia no se renderizarán.
    - 100: El plano de recorte lejano (far clipping plane). Los objetos más lejos que esta distancia no se renderizarán.

Línea 63: glTranslatef(0.0, -2.0, -10)

- Dentro de la función main: Llama a glTranslatef(). Esto aplica una traslación a la matriz de modelado-vista (modelview matrix).
    - 0.0, -2.0, -10: Mueve la cámara (o el mundo en dirección opuesta a la cámara) 0 unidades en X, -2.0 unidades en Y (hacia abajo) y -10 unidades en Z (alejándola del origen del mundo, lo que hace que los objetos sean visibles).

Línea 64: (En blanco)

- Línea en blanco: Separa la configuración de la cámara de la inicialización de OpenGL.

Línea 65: init_opengl()

- Dentro de la función main: Llama a la función init_opengl() definida anteriormente para configurar los estados globales de OpenGL (test de profundidad, color de material, color de fondo, ancho de línea).

Línea 66: (En blanco)

- Línea en blanco: Separa la inicialización de OpenGL de las variables del bucle principal.

Línea 67: angle = 0

- Dentro de la función main: Inicializa la variable angle a 0. Esta variable se utilizará para controlar la progresión de la animación a lo largo del tiempo.

Línea 68: clock = pygame.time.Clock()

- Dentro de la función main: Crea un objeto Clock de Pygame y lo asigna a la variable clock. Este objeto se utiliza para controlar la velocidad de fotogramas (FPS) de la animación.

Línea 69: running = True

- Dentro de la función main: Inicializa una variable booleana running a True. Esta variable controla si el bucle principal de la aplicación debe seguir ejecutándose.

Línea 70: (En blanco)

- Línea en blanco: Separa la inicialización de variables del comienzo del bucle principal.

Línea 71: while running:

- Dentro de la función main: Inicia un bucle while que continuará ejecutándose mientras la variable running sea True. Este es el bucle principal del juego/animación.

Línea 72: glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

- Dentro del bucle while: Llama a glClear().
    - GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT: Son banderas que indican qué buffers deben limpiarse. Limpia el buffer de color (el que almacena los píxeles visibles de la pantalla, pintándolo con el glClearColor definido) y el buffer de profundidad (el que almacena la información de profundidad para el test de profundidad). Esto se hace al inicio de cada fotograma para dibujar una escena limpia.

Línea 73: (En blanco)

- Línea en blanco: Separa la limpieza de buffers del manejo de eventos.

Línea 74: for event in pygame.event.get():

- Dentro del bucle while: Inicia un bucle for que itera sobre la lista de eventos que Pygame ha detectado desde la última llamada a pygame.event.get().

Línea 75: if event.type == QUIT:

- Dentro del bucle for: Comprueba si el tipo de evento actual (event.type) es igual a la constante QUIT. Esto ocurre cuando el usuario hace clic en el botón de cerrar la ventana.

Línea 76: running = False

- Dentro del bloque if: Si el evento es QUIT, establece la variable running a False. Esto hará que el bucle while principal termine en la próxima iteración.

Líne 77: (En blanco)

- Línea en blanco: Separa el manejo de eventos del cálculo de los ángulos de animación.

Línea 78: # Animación de cada hueso

- Dentro del bucle while: Comentario que indica que las siguientes líneas calcularán los ángulos de rotación para cada articulación del esqueleto.

Línea 79: shoulder = 30 * math.sin(math.radians(angle))

- Dentro del bucle while: Calcula el ángulo de la articulación del hombro.
    - math.radians(angle): Convierte el angle actual (que está en grados) a radianes, ya que math.sin espera radianes.
    - math.sin(...): Calcula el seno del ángulo en radianes. El seno varía entre -1 y 1.
    - 30 * ...: Multiplica el resultado por 30, haciendo que el shoulder angle oscile entre -30 y +30 grados.

Línea 80: elbow = 45 * math.sin(math.radians(angle * 2))

- Dentro del bucle while: Calcula el ángulo de la articulación del codo.
    - angle * 2: Duplica la velocidad de cambio del ángulo en comparación con el hombro.
    - math.radians(...), math.sin(...): Similar al shoulder, pero con el doble de frecuencia.
    - 45 * ...: Hace que el elbow angle oscile entre -45 y +45 grados.

Línea 81: wrist = 60 * math.sin(math.radians(angle * 3))

- Dentro del bucle while: Calcula el ángulo de la articulación de la muñeca.
    - angle * 3: Triplica la velocidad de cambio del ángulo en comparación con el hombro.
    - math.radians(...), math.sin(...): Similar a los anteriores, pero con el triple de frecuencia.
    - 60 * ...: Hace que el wrist angle oscile entre -60 y +60 grados.
- La combinación de diferentes frecuencias crea un movimiento más complejo y orgánico.

Línea 82: (En blanco)

- Línea en blanco: Separa el cálculo de los ángulos individuales de la rotación global y el dibujo del esqueleto.

Línea 83: # Rotación global del modelo

- Dentro del bucle while: Comentario que indica que se aplicará una rotación a todo el esqueleto como un conjunto.

Línea 84: glPushMatrix()

- Dentro del bucle while: Guarda la matriz de modelado-vista actual. Esto permite que la rotación global del esqueleto no afecte a la posición de la cámara establecida al principio ni a otros posibles objetos futuros dibujados fuera de este bloque.

Línea 85: glRotatef(angle * 0.3, 0, 1, 0)

- Dentro del bucle while: Aplica una rotación global a todo el esqueleto.
    - angle * 0.3: Utiliza el angle general de la animación, pero lo reduce para una rotación más lenta.
    - 0, 1, 0: Rota el esqueleto alrededor del eje Y (lo hace girar sobre sí mismo).

Línea 86: draw_skeleton(shoulder, elbow, wrist)

- Dentro del bucle while: Llama a la función draw_skeleton con los ángulos de hombro, codo y muñeca calculados para el fotograma actual. La rotación global aplicada en la línea anterior afectará a la posición inicial de todo el esqueleto dibujado por esta función.

Línea 87: glPopMatrix()

- Dentro del bucle while: Restaura la matriz de modelado-vista al estado en que estaba antes de la rotación global (glPushMatrix). Esto asegura que la rotación global solo afecte al esqueleto.

Línea 88: (En blanco)

- Línea en blanco: Separa el dibujo de la escena de la actualización de la pantalla.

Línea 89: pygame.display.flip()

- Dentro del bucle while: Llama a pygame.display.flip(). Esto actualiza el contenido de toda la pantalla. En el modo DOUBLEBUF, esto intercambia los buffers, mostrando lo que se ha dibujado en el buffer "oculto" mientras el

buffer "visible" se convierte en el nuevo buffer oculto para el siguiente fotograma.

Línea 90: clock.tick(60)

- Dentro del bucle while: Llama al método tick() del objeto clock con el argumento 60. Esto pausa el programa el tiempo suficiente para asegurar que el bucle no se ejecute a más de 60 fotogramas por segundo (FPS).

Línea 91: angle += 2

- Dentro del bucle while: Incrementa la variable angle en 2. Este angle es el valor que se utiliza para calcular los ángulos de rotación de los huesos y la rotación global en el siguiente fotograma, impulsando la animación.

Línea 92: (En blanco)

- Línea en blanco: Separa el bucle principal de la finalización de Pygame.

Línea 93: pygame.quit()

- Después del bucle while: Llama a pygame.quit(). Esto desinicializa todos los módulos de Pygame que se inicializaron con pygame.init(), liberando recursos y cerrando la ventana.

Línea 94: (En blanco)

- Línea en blanco: Separa la función main de la comprobación para ejecutarla.

Línea 95: if **name** == '**main**':

- Esta es una construcción estándar en Python. Comprueba si el script se está ejecutando directamente (no importado como un módulo en otro script).

Línea 96: main()

- Dentro del bloque if: Si el script se está ejecutando directamente, llama a la función main(). Esto inicia la ejecución de todo el programa.

Resumen del Código

Este código Python utiliza las bibliotecas `pygame` y `PyOpenGL` para crear una animación 3D de un modelo esquelético simple.

1. **Configuración e Inicialización:**

   * Importa las bibliotecas necesarias: `pygame` para la gestión de la ventana y eventos, `OpenGL.GL` y `OpenGL.GLU` para el renderizado 3D, y `math` para cálculos trigonométricos.

   * La función `init_opengl` configura estados básicos de OpenGL como el test de profundidad, el material de color, el color de fondo y el ancho de línea.

2. **Funciones de Dibujo de Primitivas:**

   * `draw_bone`: Dibuja un segmento de "hueso" como una línea gruesa, y añade una pequeña esfera en su extremo para representar una articulación.

   * `draw_sphere`: Dibuja una esfera en una posición dada, utilizando las utilidades de GLU. Ambas funciones usan `glPushMatrix()` y `glPopMatrix()` para aislar sus transformaciones y evitar que afecten al resto de la escena.

3. **Animación Esquelética Jerárquica:**

   * La función `draw_skeleton` es el corazón de la animación jerárquica. Toma tres ángulos (`shoulder_angle`, `elbow_angle`, `wrist_angle`) como parámetros.

   * Utiliza `glPushMatrix()` y `glPopMatrix()` junto con `glTranslatef()` y `glRotatef()` de manera encadenada para dibujar un sistema de huesos (hombro, codo, muñeca) donde cada hueso se posiciona y rota en relación con el hueso anterior. Esto simula una cadena cinemática (brazo).

4. **Bucle Principal de la Aplicación:**

   * La función `main` inicializa Pygame y configura una ventana para el renderizado OpenGL.

   * Establece la perspectiva de la cámara (`gluPerspective`) y una traslación inicial para posicionar la vista.

   * Entra en un bucle continuo (`while running`):

      * Limpia los buffers de color y profundidad en cada fotograma.

      * Procesa eventos de Pygame, permitiendo cerrar la ventana.

* Calcula dinámicamente los ángulos de rotación para el hombro, codo y muñeca utilizando funciones seno basadas en un contador `angle` que avanza. Esto crea un movimiento cíclico y coordinado de las articulaciones.

* Aplica una rotación global a todo el esqueleto (`glRotatef(angle * 0.3, 0, 1, 0)`) para que el modelo gire sobre su propio eje.

* Llama a `draw_skeleton` con los ángulos calculados para dibujar el esqueleto en su estado animado actual.

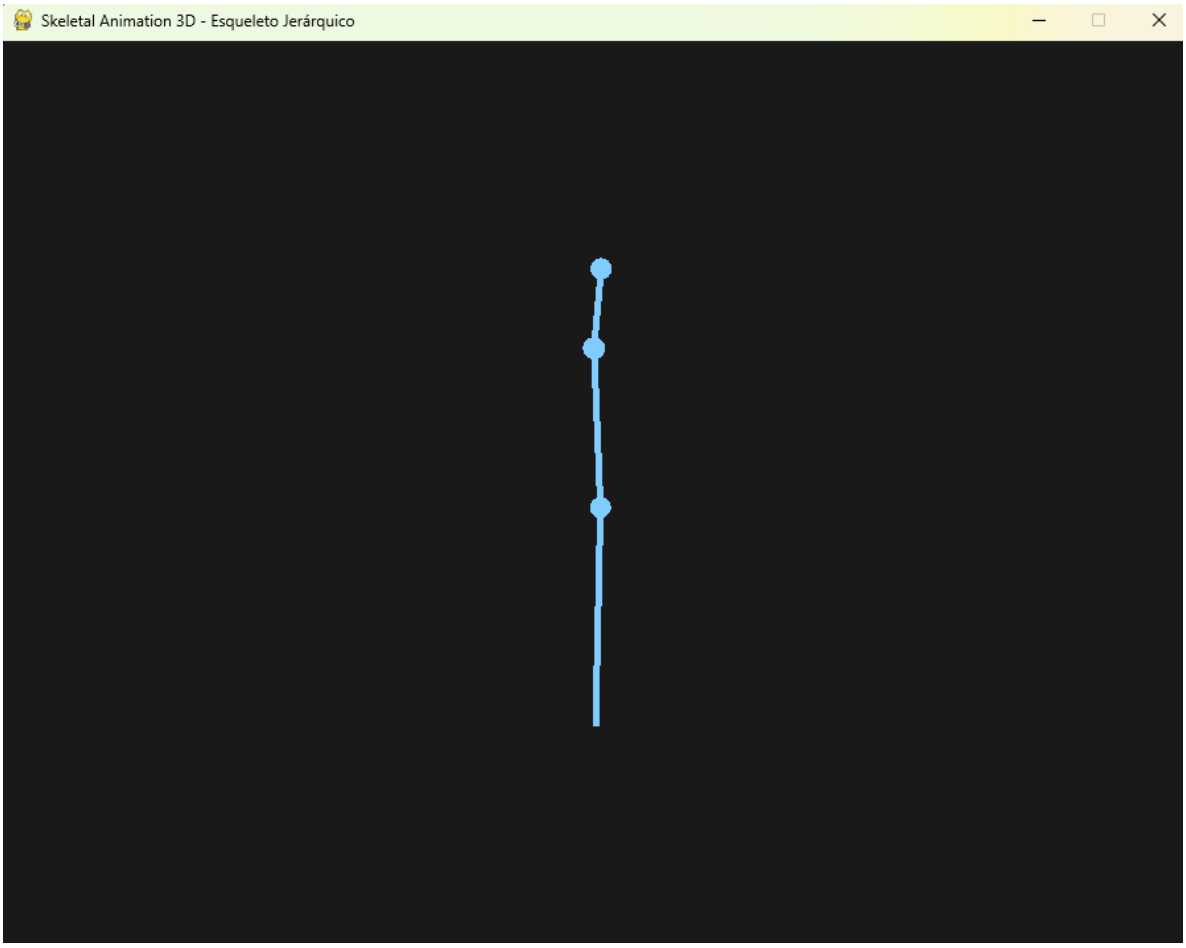
* Actualiza la pantalla (`pygame.display.flip()`) para mostrar el fotograma renderizado.

* Controla la velocidad de fotogramas a un máximo de 60 FPS (`clock.tick(60)`).

* Incrementa el contador `angle` para el siguiente fotograma.

* Una vez que el bucle termina (por ejemplo, si el usuario cierra la ventana), `pygame.quit()` se encarga de cerrar Pygame.

En esencia, el código crea una aplicación interactiva que muestra un brazo articulado en 3D, donde los segmentos se mueven de forma independiente pero coordinada, y todo el modelo gira, demostrando los principios de la animación esquelética jerárquica en OpenGL.


Skeletal Animation 3D - Esqueleto Jerárquico