Línea 1: import pygame

- Importa la biblioteca `pygame`. Pygame es un conjunto de módulos de Python diseñado para facilitar la creación de videojuegos y aplicaciones multimedia. En este código, se utiliza para la gestión de la ventana de visualización, el bucle principal de la aplicación, la captura de eventos del usuario (como cerrar la ventana) y para proporcionar un contexto de OpenGL.

Línea 2: from pygame.locals import *

- Importa todas las constantes y atributos públicos directamente desde el módulo `pygame.locals`. Esto permite utilizar nombres de constantes como `DOUBLEBUF`, `OPENGL`, y `QUIT` directamente en el código sin tener que prefijarlos con `pygame.locals.`. Esta es una práctica común en el desarrollo con Pygame para mayor comodidad.

Línea 3: from OpenGL.GL import *

- Importa todas las funciones y constantes de bajo nivel de la biblioteca OpenGL directamente desde el módulo `OpenGL.GL`. Estas funciones son las que interactúan directamente con la tarjeta gráfica para realizar operaciones de renderizado 3D, como dibujar primitivas (puntos, líneas), manipular matrices, configurar colores y habilitar funcionalidades de renderizado.

Línea 4: from OpenGL.GLU import *

- Importa todas las funciones y constantes de la biblioteca de utilidades de OpenGL (GLU) directamente desde el módulo `OpenGL.GLU`. GLU proporciona funciones de más alto nivel que simplifican tareas comunes de OpenGL, como la configuración de la perspectiva de la cámara (`gluPerspective`) o la creación de formas geométricas complejas.

Línea 5: import numpy as np

- Importa la biblioteca `numpy` y le asigna el alias `np`. NumPy es una biblioteca fundamental en Python para la computación numérica, especialmente para trabajar con arrays multidimensionales y realizar operaciones matemáticas eficientes. En este

código, se utiliza para representar las coordenadas 3D de los puntos del esqueleto y para realizar cálculos trigonométricos.

Línea 6: (En blanco)

- Línea en blanco: Actúa como un separador visual entre la sección de importaciones y el resto del código, mejorando la legibilidad.

Línea 7: # _____ DEFINICIÓN DEL ESQUELETO _____

- Comentario que sirve como un encabezado de sección. Indica que el bloque de código que sigue se dedicará a definir la estructura lógica y de conexión del esqueleto.

Línea 8: # Cada par (i, j) representa un hueso que conecta el punto i al punto j

- Comentario explicativo: Describe la convención que se utilizará en la variable `bones` a continuación. Un "hueso" se representa como una conexión directa entre dos puntos numerados del esqueleto.

Línea 9: bones = [

- Inicia la definición de la variable `bones`, que es una lista. Esta lista contendrá tuplas, donde cada tupla especifica los índices de dos puntos que están conectados por un hueso.

Línea 10: (0, 1), # pelvis -> torso

- Primer elemento de la lista `bones`: Una tupla `(0, 1)`. Esto significa que hay un "hueso" que conecta el punto con índice 0 y el punto con índice 1. El comentario `# pelvis -> torso` aclara que estos puntos representan la conexión entre la pelvis y el torso.

Línea 11: (1, 2), # torso -> cuello

- Segundo hueso: Conecta el punto 1 (torso) con el punto 2 (cuello).

Línea 12: (2, 3), # cuello -> cabeza

- Tercer hueso: Conecta el punto 2 (cuello) con el punto 3 (cabeza).

Línea 13: (1, 4), # torso -> brazo izq

- Cuarto hueso: Conecta el punto 1 (torso) con el punto 4 (brazo izquierdo).

Línea 14: (4, 5), # brazo izq -> antebrazo izq

- Quinto hueso: Conecta el punto 4 (brazo izquierdo) con el punto 5 (antebrazo izquierdo).

Línea 15: (1, 6), # torso -> brazo der

- Sexto hueso: Conecta el punto 1 (torso) con el punto 6 (brazo derecho).

Línea 16: (6, 7), # brazo der -> antebrazo der

- Séptimo hueso: Conecta el punto 6 (brazo derecho) con el punto 7 (antebrazo derecho).

Línea 17: (0, 8), # pelvis -> pierna izq

- Octavo hueso: Conecta el punto 0 (pelvis) con el punto 8 (pierna izquierda).

Línea 18: (8, 9), # pierna izq -> pie izq

- Noveno hueso: Conecta el punto 8 (pierna izquierda) con el punto 9 (pie izquierdo).

Línea 19: (0, 10), # pelvis -> pierna der

- Décimo hueso: Conecta el punto 0 (pelvis) con el punto 10 (pierna derecha).

Línea 20:   (10, 11) # pierna der -> pie der

- Undécimo y último hueso: Conecta el punto 10 (pierna derecha) con el punto 11 (pie derecho).

Línea 21:]

- Cierre de la definición de la lista `bones`.

Línea 22: (En blanco)

- Línea en blanco: Separa la definición de la estructura del esqueleto de la función que generará los datos de movimiento simulados.

Línea 23: # FRAMES DE MOCAP SIMULADOS (12 puntos por frame)

- Comentario que sirve como encabezado de sección. Indica que el siguiente bloque de código se ocupa de generar datos de captura de movimiento (mocap) de forma simulada. También especifica que cada "frame" (fotograma) de estos datos contendrá 12 puntos 3D.

Línea 24: def generate_mock_mocap_frames(num_frames=120):

- Define una función llamada `generate_mock_mocap_frames`. Esta función es responsable de crear los datos de animación. Toma un parámetro opcional `num_frames`, que por defecto es 120, especificando cuántos fotogramas de animación se deben generar.

Línea 25:   frames = []

- Dentro de `generate_mock_mocap_frames`: Inicializa una lista vacía llamada `frames`. Esta lista se utilizará para almacenar cada fotograma de datos de puntos del esqueleto a medida que se generen.

Línea 26: for t in range(num_frames):

- Dentro de `generate_mock_mocap_frames`: Inicia un bucle `for` que iterará `num_frames` veces. La variable `t` actuará como el índice del fotograma actual, variando desde 0 hasta `num_frames - 1`.

Línea 27: angle = np.radians(t * 3)

- Dentro del bucle `for`: Calcula un `angle` (ángulo) para el fotograma actual. `t * 3` significa que el ángulo aumenta 3 grados por cada fotograma. `np.radians()` convierte este valor de grados a radianes, ya que las funciones trigonométricas de NumPy (como `np.sin` y `np.cos`) operan con radianes.

Línea 28: points = np.array([

- Dentro del bucle `for`: Inicia la definición de un array NumPy llamado `points`. Este array representará las coordenadas 3D de los 12 puntos (articulaciones) para el esqueleto en el fotograma `t`.

Línea 29: [0, 0, 0], # 0 pelvis

- Define las coordenadas `(x, y, z)` para el punto con índice 0 (pelvis). Aquí, la pelvis se mantiene fija en el origen `[0, 0, 0]`.

Línea 30: [0, 1, 0], # 1 torso

- Coordenadas del punto 1 (torso): `[0, 1, 0]`.

Línea 31: [0, 2, 0], # 2 cuello

- Coordenadas del punto 2 (cuello): `[0, 2, 0]`.

Línea 32: [0, 2.5, 0.2*np.sin(angle)], # 3 cabeza

- Coordenadas del punto 3 (cabeza): Su coordenada X e Y son fijas, pero la coordenada Z (`0.2*np.sin(angle)`) oscila sinusoidalmente con el `angle` del fotograma. Esto simula un movimiento de balanceo de la cabeza.

Línea 33: [-0.5, 1.8, 0.2*np.sin(angle)], # 4 brazo izq

- Coordenadas del punto 4 (brazo izquierdo): Su coordenada Z también oscila con `angle`, igual que la cabeza, creando un movimiento coordinado.

Línea 34: [-1, 1.6, 0.4*np.sin(angle)], # 5 antebrazo izq

- Coordenadas del punto 5 (antebrazo izquierdo): Su coordenada Z oscila con el doble de amplitud (`0.4*np.sin(angle)`) que la del brazo izquierdo, lo que amplifica el movimiento.

Línea 35: [0.5, 1.8, -0.2*np.sin(angle)], # 6 brazo der

- Coordenadas del punto 6 (brazo derecho): Su coordenada Z oscila en dirección opuesta a la del brazo izquierdo (`-0.2*np.sin(angle)`), simulando un movimiento de brazos alterno.

Línea 36: [1, 1.6, -0.4*np.sin(angle)], # 7 antebrazo der

- Coordenadas del punto 7 (antebrazo derecho): Su coordenada Z oscila en dirección opuesta y con el doble de amplitud que el antebrazo izquierdo.

Línea 37: [-0.3, -1, 0.1*np.cos(angle)], # 8 pierna izq

- Coordenadas del punto 8 (pierna izquierda): Su coordenada Z oscila utilizando la función coseno (`0.1*np.cos(angle)`). El uso de coseno en lugar de seno introduce un desfase en el movimiento de las piernas con respecto a los brazos.

Línea 38: [-0.3, -2, 0.2*np.cos(angle)], # 9 pie izq

- Coordenadas del punto 9 (pie izquierdo): Su coordenada Z oscila con el doble de amplitud que la pierna izquierda, usando coseno.

Línea 39: [0.3, -1, -0.1*np.cos(angle)], # 10 pierna der

- Coordenadas del punto 10 (pierna derecha): Su coordenada Z oscila en dirección opuesta a la pierna izquierda (`-0.1*np.cos(angle)`).

Línea 40: [0.3, -2, -0.2*np.cos(angle)], # 11 pie der

- Coordenadas del punto 11 (pie derecho): Su coordenada Z oscila en dirección opuesta y con el doble de amplitud que el pie izquierdo.

- El `]])` cierra la definición del array NumPy `points`.

Línea 41: frames.append(points)

- Dentro del bucle `for`: Añade el array `points` (que contiene las coordenadas 3D de todas las articulaciones para el fotograma `t`) a la lista `frames`.

Línea 42: return frames

- Dentro de `generate_mock_mocap_frames`: Después de que el bucle `for` ha terminado de generar todos los fotogramas, la función devuelve la lista `frames`, que ahora contiene una secuencia completa de arrays NumPy, cada uno representando una pose del esqueleto en un momento dado de la animación.

Línea 43: (En blanco)

- Línea en blanco: Sirve como separador visual entre la función de generación de datos y las funciones de dibujo de OpenGL.

Línea 44: # FUNCIONES DE DIBUJO

- Comentario que sirve como encabezado de sección. Indica que el siguiente bloque de código contiene funciones dedicadas a renderizar el esqueleto utilizando OpenGL.

Línea 45: def draw_skeleton(points):

- Define una función llamada `draw_skeleton`. Esta función es la encargada de dibujar el esqueleto en la pantalla de OpenGL para un fotograma específico. Acepta un parámetro `points`, que se espera sea un array o una secuencia de coordenadas 3D de los puntos del esqueleto para el fotograma actual.

Línea 46: glColor3f(0.6, 0.9, 1.0)

- Dentro de `draw_skeleton`: Establece el color actual para el dibujo en OpenGL. `glColor3f(r, g, b)` especifica un color utilizando componentes de rojo, verde y azul con valores flotantes entre 0.0 y 1.0. Aquí, se establece un color azul claro o cian para los "huesos".

Línea 47: glBegin(GL_LINES)

- Dentro de `draw_skeleton`: Inicia la definición de un conjunto de primitivas de tipo línea. Todo lo que se especifique con `glVertex3fv` entre esta llamada y `glEnd()` será interpretado como segmentos de línea. Cada dos vértices definen una línea.

Línea 48: for i, j in bones:

- Dentro de `draw_skeleton`: Inicia un bucle `for` que itera sobre cada tupla `(i, j)` en la lista global `bones`. Cada tupla representa un par de índices de puntos que deben ser conectados por una línea.

Línea 49: glVertex3fv(points[i])

- Dentro del bucle `for`: Especifica el primer vértice de un segmento de línea. `glVertex3fv(vector)` toma un array o tupla de 3 elementos como las coordenadas `(x, y, z)` del vértice. `points[i]` accede a las coordenadas 3D del punto `i` del fotograma actual.

Línea 50: glVertex3fv(points[j])

- Dentro del bucle `for`: Especifica el segundo vértice del mismo segmento de línea. `points[j]` accede a las coordenadas 3D del punto `j`. Una línea se dibujará desde las coordenadas de `points[i]` hasta las de `points[j]`.

Línea 51: glEnd()

- Dentro de `draw_skeleton`: Finaliza la definición del conjunto de primitivas de líneas que comenzó con `glBegin(GL_LINES)`.

Línea 52: (En blanco)

- Dentro de `draw_skeleton`: Línea en blanco para separar visualmente el dibujo de las líneas (huesos) del dibujo de los puntos (articulaciones).

Línea 53: glPointSize(6)

- Dentro de `draw_skeleton`: Establece el tamaño para los puntos que se dibujarán a continuación. `glPointSize(size)` especifica el diámetro de los puntos en píxeles. Aquí, los puntos que representan las articulaciones tendrán un tamaño de 6 píxeles.

Línea 54: glColor3f(1, 0.5, 0.2)

- Dentro de `draw_skeleton`: Cambia el color actual para el dibujo. Se establece un color naranja/marrón claro (`1.0` rojo, `0.5` verde, `0.2` azul) para los puntos de las articulaciones.

Línea 55: glBegin(GL_POINTS)

- Dentro de `draw_skeleton`: Inicia la definición de un conjunto de primitivas de tipo punto. Cada `glVertex3fv` especificado entre esta llamada y `glEnd()` definirá un punto individual en la pantalla.

Línea 56: for p in points:

- Dentro de `draw_skeleton`: Inicia un bucle `for` que itera directamente sobre cada conjunto de coordenadas 3D (`p`) presente en el array `points` (el fotograma actual).

Lí línea 57: glVertex3fv(p)

- Dentro del bucle `for`: Especifica las coordenadas 3D para un punto. Dado que `p` ya es un array de 3 elementos `(x, y, z)`, se pasa directamente a `glVertex3fv`.

Línea 58: glEnd()

- Dentro de `draw_skeleton`: Finaliza la definición del conjunto de primitivas de puntos que comenzó con `glBegin(GL_POINTS)`.

Línea 59: (En blanco)

- Línea en blanco: Separa la sección de funciones de dibujo de la sección principal de configuración de OpenGL y el bucle del programa.

Línea 60: # _____ OPENGL Y LOOP PRINCIPAL _____

- Comentario que sirve como encabezado de sección. Indica que el bloque de código que sigue se refiere a la inicialización de OpenGL y al bucle principal de la aplicación.

Línea 61: def init_opengl():

- Define una función llamada `init_opengl`. Esta función encapsula la configuración inicial mínima necesaria para el contexto de OpenGL en esta aplicación.

Línea 62: glEnable(GL_DEPTH_TEST)

- Dentro de `init_opengl`: Habilita la funcionalidad de prueba de profundidad (depth testing) en OpenGL. Esto es crucial para el renderizado 3D, ya que asegura que los objetos más cercanos a la cámara se dibujen por encima de los objetos más lejanos,

evitando que objetos distantes se "superpongan" visualmente a objetos más próximos.

Línea 63: glClearColor(0.05, 0.05, 0.05, 1)

- Dentro de `init_opengl`: Establece el color que se utilizará para borrar el búfer de color (el fondo de la escena) en cada fotograma. Los valores `(0.05, 0.05, 0.05, 1)` representan un color gris muy oscuro (casi negro) con una opacidad total (1.0).

Línea 64: (En blanco)

- Línea en blanco: Separa la función `init_opengl` de la función principal `main`.

Línea 65: def main():

- Define la función `main`. Esta es la función principal del programa, donde se inicializa todo, se configura el bucle de juego y se gestiona la lógica principal de la aplicación.

Línea 66: pygame.init()

- Dentro de `main`: Inicializa todos los módulos de Pygame necesarios para su funcionamiento. Es una llamada obligatoria al principio de cualquier programa Pygame.

Línea 67: display = (1000, 700)

- Dentro de `main`: Define una tupla `display` que contiene el ancho (1000 píxeles) y el alto (700 píxeles) deseados para la ventana de visualización del programa.

Línea 68: pygame.display.set_mode(display, DOUBLEBUF | OPENGL)

- Dentro de `main`: Crea la superficie de visualización (la ventana donde se mostrará el contenido gráfico) de Pygame.

- `display`: Establece el tamaño de la ventana.

- `DOUBLEBUF`: Habilita el doble búfer. Esto significa que el dibujo se realiza en un búfer de memoria "oculto" mientras el búfer "visible" se muestra. Una vez que el dibujo está completo, los búferes se intercambian, lo que resulta en animaciones más fluidas y sin parpadeo.

- `OPENGL`: Le indica a Pygame que el modo de visualización debe ser compatible con OpenGL, creando un contexto OpenGL en la ventana.

Línea 69: pygame.display.set_caption("Motion Capture 3D - Demo Skeleton")

- Dentro de `main`: Establece el texto que aparecerá en la barra de título de la ventana creada por Pygame.

Línea 70: (En blanco)

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

Línea 71: gluPerspective(45, display[0] / display[1], 0.1, 50.0)

- Dentro de `main`: Configura la matriz de proyección en OpenGL para una perspectiva de cámara.

- `45`: El ángulo del campo de visión (FOV) vertical en grados.

- `display[0] / display[1]`: La relación de aspecto (ancho dividido por alto) de la ventana. Esto es importante para evitar distorsiones en la imagen.

- `0.1`: La distancia al plano de recorte cercano (near clipping plane). Los objetos más cercanos que esta distancia no se renderizarán.

- `50.0`: La distancia al plano de recorte lejano (far clipping plane). Los objetos más lejanos que esta distancia no se renderizarán.

Línea 72: glTranslatef(0, -1.0, -8)

- Dentro de `main`: Aplica una traslación (movimiento) a la matriz de modelo-vista actual de OpenGL. Esto es equivalente a mover la "cámara" virtual. Mueve la vista -8 unidades en el eje Z (alejándola del origen, haciendo que los objetos sean visibles) y -

1.0 unidades en el eje Y (moviéndola ligeramente hacia abajo), para que el esqueleto aparezca bien centrado en la vista.

Línea 73: (En blanco)

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

Línea 74: init_opengl()

- Dentro de `main`: Llama a la función `init_opengl()` definida anteriormente para aplicar las configuraciones iniciales de OpenGL, como habilitar la prueba de profundidad y establecer el color de fondo.

Línea 75: mocap_frames = generate_mock_mocap_frames()

- Dentro de `main`: Llama a la función `generate_mock_mocap_frames()` para obtener la lista de todos los fotogramas de datos de movimiento simulados. La lista resultante de arrays NumPy se almacena en la variable `mocap_frames`.

Línea 76: frame_count = len(mocap_frames)

- Dentro de `main`: Calcula el número total de fotogramas generados (la longitud de la lista `mocap_frames`) y lo almacena en la variable `frame_count`. Este valor se utilizará para ciclar la animación.

Línea 77: clock = pygame.time.Clock()

- Dentro de `main`: Crea una instancia del objeto `Clock` de Pygame. Este objeto es esencial para controlar la velocidad de fotogramas (FPS) del bucle principal del programa, asegurando que la animación se ejecute a una velocidad consistente.

Línea 78: angle = 0

- Dentro de `main`: Inicializa la variable `angle` a 0. Esta variable se utilizará para controlar la rotación global del esqueleto alrededor de su eje vertical en la escena.

Línea 79: frame_index = 0

- Dentro de `main`: Inicializa la variable `frame_index` a 0. Esta variable se utilizará para llevar un seguimiento del fotograma actual de la animación de mocap que debe dibujarse en cada iteración del bucle.

Línea 80: (En blanco)

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

Línea 81: running = True

- Dentro de `main`: Inicializa una variable booleana `running` a `True`. Esta variable actúa como una bandera de control para el bucle principal, manteniendo la aplicación en ejecución mientras su valor sea `True`.

Línea 82: while running:

- Dentro de `main`: Inicia el bucle principal del programa. Este bucle se ejecutará repetidamente (creando cada fotograma de la animación) mientras la variable `running` sea `True`.

Línea 83: glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

- Dentro del bucle `while`: Borra el contenido de los búferes de color y profundidad de OpenGL en cada fotograma. `GL_COLOR_BUFFER_BIT` borra todos los píxeles del búfer de color al color de fondo predefinido (`glClearColor`).
`GL_DEPTH_BUFFER_BIT` borra el búfer de profundidad, lo que es vital para que la prueba de profundidad funcione correctamente en cada nueva renderización del fotograma.

Línea 84: (En blanco)

- Dentro del bucle `while`: Línea en blanco para separar la limpieza de los búferes de la sección de manejo de eventos.

Línea 85: for event in pygame.event.get():

- Dentro del bucle `while`: Inicia un bucle `for` que itera sobre todos los eventos que Pygame ha detectado (como movimientos del ratón, pulsaciones de teclas, clics de ventana, etc.) desde la última llamada a `pygame.event.get()`.

Línea 86: if event.type == pygame.QUIT:

- Dentro del bucle `for` (manejo de eventos): Comprueba si el tipo de evento actual (`event.type`) es igual a la constante `pygame.QUIT`. Este evento se genera cuando el usuario intenta cerrar la ventana (por ejemplo, haciendo clic en el botón 'X' de la ventana).

Línea 87: running = False

- Dentro del condicional `if`: Si se detecta el evento `pygame.QUIT`, se establece la variable `running` a `False`. Esto provocará que el bucle `while running` termine en su próxima evaluación, cerrando la aplicación de forma ordenada.

Línea 88: (En blanco)

- Dentro del bucle `while`: Línea en blanco para separar el manejo de eventos del proceso de dibujo.

Línea 89: glPushMatrix()

- Dentro del bucle `while`: Guarda la matriz de modelo-vista actual (la que define la posición de la cámara y el mundo) en una pila interna de matrices. Esto permite aplicar transformaciones subsiguientes (como la rotación del esqueleto) que solo afectarán a los objetos dibujados antes de un `glPopMatrix()`, sin modificar la configuración de la cámara de forma permanente.

Línea 90: glRotatef(angle, 0, 1, 0)

- Dentro del bloque `glPushMatrix`/`glPopMatrix`: Aplica una rotación a la matriz de modelo-vista actual.

- `angle`: El ángulo de rotación en grados. Esta variable se incrementa en cada fotograma, lo que hace que el esqueleto gire continuamente.

- `0, 1, 0`: Los componentes del vector sobre el cual se realiza la rotación. `(0, 1, 0)` indica que la rotación se hará alrededor del eje Y (el eje vertical en la escena).

Línea 91: draw_skeleton(mocap_frames[frame_index])

- Dentro del bloque `glPushMatrix`/`glPopMatrix`: Llama a la función `draw_skeleton()` definida anteriormente. Le pasa `mocap_frames[frame_index]`, que es el array NumPy que contiene las coordenadas 3D de todos los puntos del esqueleto para el fotograma actual de la animación de mocap. Esta llamada es la que realmente dibuja el esqueleto en su pose y orientación para el fotograma.

Línea 92: glPopMatrix()

- Dentro del bucle `while`: Restaura la matriz de modelo-vista a su estado anterior a la última llamada a `glPushMatrix()`. Esto deshace las transformaciones aplicadas dentro del bloque, como la rotación global del esqueleto, para que el siguiente dibujo (si hubiera otro) comience con la matriz base.

Línea 93: (En blanco)

- Dentro del bucle `while`: Línea en blanco para separar el dibujo de la escena de la actualización de las variables de animación.

Línea 94: frame_index = (frame_index + 1) % frame_count

- Dentro del bucle `while`: Actualiza el `frame_index` para avanzar al siguiente fotograma de la animación mocap.

- `(frame_index + 1)`: Incrementa el índice en 1.

- `% frame_count`: El operador módulo (`%`) asegura que el `frame_index` "se reinicie" a 0 una vez que alcanza el `frame_count` total. Esto crea un bucle continuo de la animación de mocap.

Línea 95: angle += 0.5

- Dentro del bucle `while`: Incrementa la variable `angle` en 0.5 grados en cada fotograma. Esto asegura que la rotación global del esqueleto sea continua y constante a lo largo del tiempo.

Línea 96: (En blanco)

- Dentro del bucle `while`: Línea en blanco para separar la actualización de variables de la actualización de la pantalla.

Línea 97: pygame.display.flip()

- Dentro del bucle `while`: Intercambia los búferes de visualización (cuando se usa `DOUBLEBUF`). El contenido del búfer "oculto" (donde se ha dibujado el fotograma actual) se muestra en la pantalla, y el búfer visible se convierte en el búfer oculto para el próximo ciclo de dibujo. Esto es fundamental para una animación fluida sin efectos de "tearing" (desgarro de imagen).

Línea 98: clock.tick(30)

- Dentro del bucle `while`: Limita la velocidad de fotogramas (FPS) del bucle principal a un máximo de 30 fotogramas por segundo. Pygame pausará brevemente el bucle si es necesario para no exceder esta tasa, lo que ayuda a garantizar que la animación se ejecute a una velocidad consistente en diferentes sistemas.

Línea 99: (En blanco)

- Línea en blanco: Separa el bucle principal del código de limpieza final.

Línea 100: pygame.quit()

- Después del bucle `while`: Una vez que el bucle principal termina (porque `running` se ha establecido en `False`), esta línea desinicializa todos los módulos de Pygame que se habían inicializado con `pygame.init()`. Es una buena práctica para liberar los recursos del sistema de forma adecuada.

Línea 101: (En blanco)

- Línea en blanco: Separa la definición de la función `main` del bloque de ejecución condicional.

Línea 102: if __name__ == "__main__":

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

Línea 103:   main()

- Si el script se está ejecutando directamente, esta línea llama a la función `main()`, iniciando así todo el programa.

Nota sobre la repetición del código:

Como mencioné anteriormente, el código desde la línea 105 hasta el final del archivo es una copia exacta e idéntica de las líneas 1 a 104. En un escenario real, esta duplicación no sería funcionalmente correcta ni recomendable, ya que el intérprete de Python simplemente procesaría la primera instancia del código. La segunda instancia sería redundante y un ejemplo de violación del principio DRY (Don't Repeat Yourself - No te repitas). Para este análisis, he explicado la primera instancia completa.

Resumen del Código

Este programa Python utiliza las bibliotecas `pygame`, `OpenGL` y `numpy` para simular y visualizar una animación 3D de un esqueleto humano simplificado.

1. **Definición de la Estructura del Esqueleto:** Se define la estructura del esqueleto a través de una lista de "huesos" (`bones`), donde cada hueso es una conexión entre dos puntos (articulaciones) específicos representados por índices.

2. **Generación de Datos de Animación (Mocap Simulada):** La función `generate_mock_mocap_frames` crea una serie de "fotogramas" de datos. Cada fotograma contiene las coordenadas 3D de 12 puntos clave del esqueleto (como pelvis, torso, cabeza, brazos y piernas). Estas coordenadas se modifican con funciones trigonométricas (`sin` y `cos`) en función del tiempo para simular un movimiento de balanceo y oscilación, creando una animación de captura de movimiento básica.

3. **Funciones de Dibujo 3D (OpenGL):** La función `draw_skeleton` toma las coordenadas de los puntos de un fotograma y las renderiza en 3D utilizando OpenGL. Dibuja líneas para representar los huesos y puntos para representar las articulaciones, con colores y tamaños definidos.

4. **Inicialización y Configuración:**

   * La función `init_opengl` establece configuraciones básicas de OpenGL, como habilitar la prueba de profundidad (para que los objetos se oculten correctamente unos detrás de otros) y definir el color de fondo.

   * La función `main` inicializa Pygame y crea una ventana de visualización en modo OpenGL. Configura la perspectiva de la cámara 3D (`gluPerspective`) y traslada la vista para que el esqueleto sea visible en la pantalla.

5. **Bucle Principal de Animación:**

   * El programa entra en un bucle continuo (`while running`) que gestiona la lógica de la animación y el renderizado en cada fotograma.

   * En cada iteración, se limpian los búferes de color y profundidad de OpenGL para preparar el nuevo fotograma.

* Se procesan los eventos del usuario (por ejemplo, cerrar la ventana) para controlar la ejecución del programa.

* Se aplica una rotación global al esqueleto en la escena (alrededor del eje Y) para que el modelo gire continuamente.

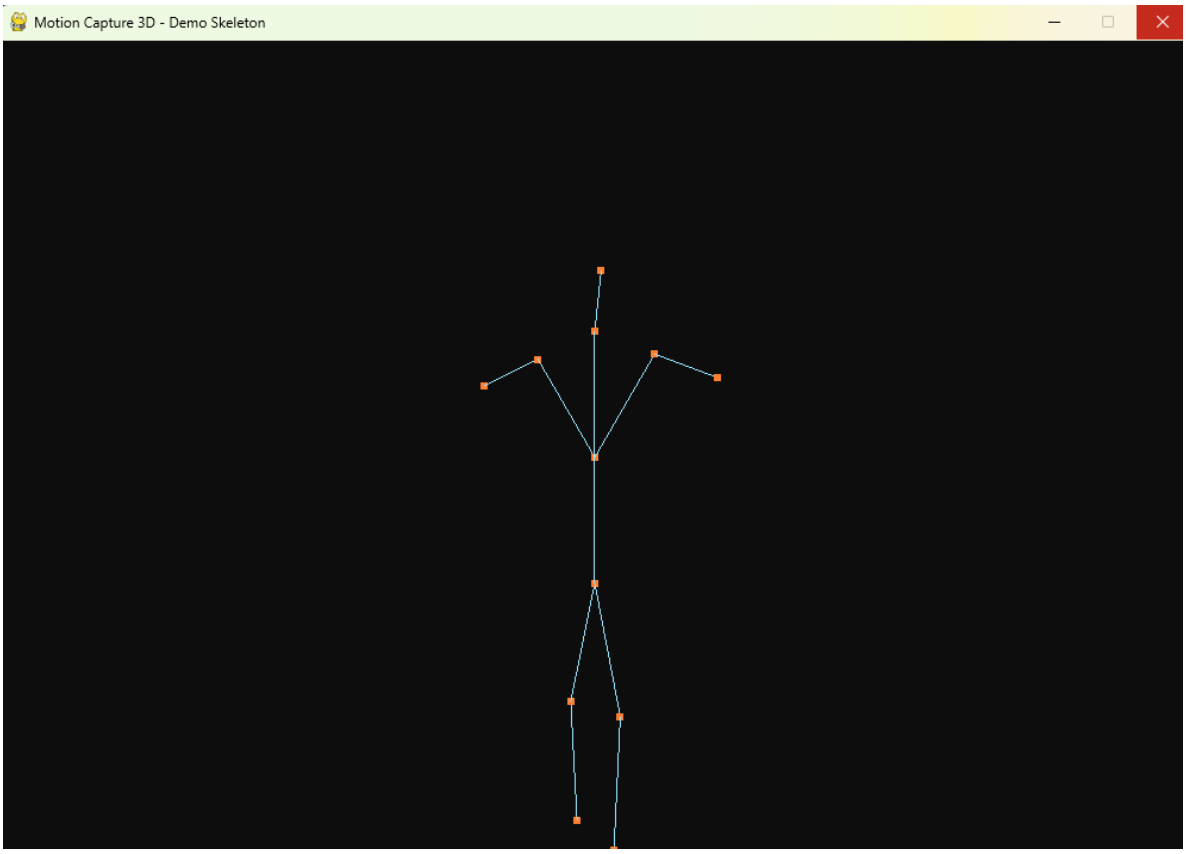
* Se llama a `draw_skeleton` para dibujar el esqueleto en la pose correspondiente al fotograma actual de los datos de mocap.

* El índice del fotograma se actualiza y se cicla (`% frame_count`) para asegurar que la animación de mocap se repita sin fin.

* Finalmente, `pygame.display.flip()` actualiza la pantalla para mostrar el fotograma recién renderizado, y `clock.tick(30)` limita la velocidad de fotogramas a 30 por segundo para una animación fluida y consistente.

6. **Ejecución:** El bloque `if __name__ == "__main__":` garantiza que la función `main()` se ejecute cuando el script se inicia directamente.

En resumen, el código genera una animación 3D de un esqueleto humano virtual realizando un movimiento oscilatorio, todo ello renderizado y controlado mediante las librerías Pygame y OpenGL, permitiendo una visualización interactiva en tiempo real.


Motion Capture 3D - Demo Skeleton