Línea 1: import sys

- Importa el módulo sys, que proporciona acceso a funciones y variables que interactúan fuertemente con el intérprete de Python, como sys.exit() para salir del programa.

Línea 2: import pygame

- Importa la biblioteca pygame, que es un conjunto de módulos Python diseñados para escribir videojuegos. Se utiliza aquí para manejar la ventana, los eventos del teclado/ratón y la configuración de OpenGL.

Línea 3: from pygame.locals import *

- Importa todas las constantes del módulo pygame.locals directamente al espacio de nombres actual. Esto permite usar constantes como QUIT, DOUBLEBUF, OPENGL, K_UP, etc., sin prefijarlas con pygame.locals..

Línea 4: from OpenGL.GL import *

- Importa todas las funciones y constantes de la biblioteca OpenGL para Python (PyOpenGL). El GL es el módulo principal de OpenGL, proporcionando funciones para dibujar primitivas, configurar colores, matrices, etc.

Línea 5: from OpenGL.GLU import *

- Importa todas las funciones y constantes del módulo OpenGL Utility Library (GLU) para Python. GLU proporciona funciones de utilidad de nivel superior que se basan en OpenGL, como gluPerspective para configurar la perspectiva de la cámara.

Línea 6: import numpy as np

- Importa la biblioteca numpy y le asigna el alias np. Se utiliza para operaciones numéricas, como la generación de números aleatorios en una sección del código, aunque el resultado no se usa activamente en la lógica de rotación/zoom.

Línea 7: (En blanco)

- Línea en blanco: Separa las declaraciones de importación de la definición de las variables globales que representan la geometría del cubo.

Línea 8: vertices = (

- Define una tupla llamada vertices. Esta tupla contendrá las coordenadas tridimensionales de cada uno de los 8 vértices del cubo.

Línea 9: (-1, -1, -1),

- Primer elemento de la tupla vertices: Las coordenadas (x, y, z) del vértice 0 del cubo.

Línea 10: (1, -1, -1),

- Segundo elemento de la tupla vertices: Las coordenadas del vértice 1.

Línea 11: (1, 1, -1),

- Tercer elemento de la tupla vertices: Las coordenadas del vértice 2.

Línea 12: (-1, 1, -1),

- Cuarto elemento de la tupla vertices: Las coordenadas del vértice 3.

Línea 13: (-1, -1, 1),

- Quinto elemento de la tupla vertices: Las coordenadas del vértice 4.

Línea 14: (1, -1, 1),

- Sexto elemento de la tupla vertices: Las coordenadas del vértice 5.

Línea 15: (1, 1, 1),

- Séptimo elemento de la tupla vertices: Las coordenadas del vértice 6.

Línea 16: (-1, 1, 1)

- Octavo y último elemento de la tupla vertices: Las coordenadas del vértice 7.

Línea 17:)

- Cierra la definición de la tupla vertices.

Línea 18: (En blanco)

- Línea en blanco: Separa la definición de vertices de la definición de edges.

Línea 19: edges = (

- Define una tupla llamada edges. Esta tupla contiene sub-tuplas, donde cada sub-tupla representa una arista del cubo especificando los índices de los dos vértices que la conectan, basándose en la tupla vertices.

Línea 20: (0,1), (1,2), (2,3), (3,0),

- Define las primeras cuatro aristas que forman la cara "trasera" (o inferior en la configuración inicial) del cubo.

Línea 21: (4,5), (5,6), (6,7), (7,4),

- Define las siguientes cuatro aristas que forman la cara "frontal" (o superior en la configuración inicial).

Línea 22: (0,4), (1,5), (2,6), (3,7)

- Define las últimas cuatro aristas que conectan las dos caras anteriores, formando los lados del cubo.

Línea 23:)

- Cierra la definición de la tupla edges.

Línea 24: (En blanco)

- Línea en blanco: Separa la definición de edges de la definición de faces.

Línea 25: faces = (

- Define una tupla llamada faces. Cada elemento de esta tupla es una tupla de índices de vértices que definen una cara cuadrada del cubo, en un orden específico para que OpenGL las dibuje correctamente.

Línea 26: (0,1,2,3),

- Define la primera cara del cubo utilizando los vértices con índices 0, 1, 2 y 3.

Línea 27: (4,5,6,7),

- Define la segunda cara del cubo utilizando los vértices con índices 4, 5, 6 y 7.

Línea 28: (0,1,5,4),

- Define la tercera cara del cubo utilizando los vértices con índices 0, 1, 5 y 4.

Línea 29: (2,3,7,6),

- Define la cuarta cara del cubo utilizando los vértices con índices 2, 3, 7 y 6.

Línea 30: (1,2,6,5),

- Define la quinta cara del cubo utilizando los vértices con índices 1, 2, 6 y 5.

Línea 31: (0,3,7,4)

- Define la sexta y última cara del cubo utilizando los vértices con índices 0, 3, 7 y 4.

Línea 32:)

- Cierra la definición de la tupla faces.

Línea 33: (En blanco)

- Línea en blanco: Separa la definición de faces de la definición de colors.

Línea 34: colors = [

- Define una lista llamada colors. Esta lista contiene tuplas RGB (rojo, verde, azul) que representan los colores que se aplicarán a cada una de las 6 caras del cubo.

Línea 35: (1,0,0), (0,1,0), (0,0,1),

- Primeros tres colores: rojo puro (1,0,0), verde puro (0,1,0), azul puro (0,0,1).

Línea 36: (1,1,0), (1,0,1), (0,1,1)

- Últimos tres colores: amarillo (1,1,0), magenta (1,0,1), cian (0,1,1).

Línea 37:]

- Cierra la definición de la lista colors.

Línea 38: (En blanco)

- Línea en blanco: Separa la definición de las variables globales de la definición de la función draw_cube.

Línea 39: def draw_cube(intensity):

- Define una función llamada draw_cube que toma un argumento intensity. Esta función se encargará de dibujar el cubo en la escena OpenGL. El parámetro intensity se usará para ajustar el brillo de las caras.

Línea 40: glBegin(GL_QUADS)

- Dentro de draw_cube: Inicia la definición de un grupo de primitivas de tipo "quads" (cuadriláteros). Los vértices especificados a partir de ahora se interpretarán como grupos de cuatro que forman una cara.

Línea 41: for i, face in enumerate(faces):

- Dentro de draw_cube: Inicia un bucle que itera sobre la lista faces. enumerate proporciona tanto el índice i (0 a 5) como la face (la tupla de vértices de la cara) para cada iteración.

Línea 42: color = colors[i]

- Dentro del bucle for face: Asigna el color de la lista colors correspondiente al índice i de la cara actual a la variable color.

Línea 43: r = min(max(color[0] * intensity, 0.0), 1.0)

- Dentro del bucle for face: Calcula el componente rojo (r) del color.
- color[0] * intensity: Multiplica el componente rojo original por el factor intensity.
- max(..., 0.0): Asegura que el valor resultante no sea menor que 0.0 (lo clampa a 0.0 si es negativo).
- min(..., 1.0): Asegura que el valor resultante no sea mayor que 1.0 (lo clampa a 1.0 si excede). Esto mantiene el valor dentro del rango válido de 0 a 1 para componentes de color RGB.

Línea 44: g = min(max(color[1] * intensity, 0.0), 1.0)

- Dentro del bucle for face: Realiza el mismo cálculo de clamping para el componente verde (g).

Línea 45: b = min(max(color[2] * intensity, 0.0), 1.0)

- Dentro del bucle for face: Realiza el mismo cálculo de clamping para el componente azul (b).

Línea 46: glColor3f(r, g, b)

- Dentro del bucle for face: Establece el color actual para dibujar. Todos los vértices que se definan después de esta llamada tendrán este color hasta que se cambie.

Línea 47: for vertex in face:

- Dentro del bucle for face: Inicia un bucle anidado que itera sobre los índices de los vértices que componen la face actual.

Línea 48: glVertex3fv(vertices[vertex])

- Dentro del bucle for vertex: Especifica un vértice. vertices[vertex] accede a las coordenadas (x, y, z) reales del vértice usando el índice vertex de la face. glVertex3fv envía estas coordenadas a OpenGL para que sean parte de la primitiva actual (el cuádruple).

Línea 49: glEnd()

- Dentro de draw_cube: Finaliza la definición del grupo de primitivas de tipo GL_QUADS. A partir de este punto, OpenGL procesará los vértices especificados como un conjunto de caras.

Línea 50: (En blanco)

- Dentro de draw_cube: Línea en blanco: Separa el dibujo de las caras del dibujo de las aristas.

Línea 51: glColor3f(0, 0, 0)

- Dentro de draw_cube: Establece el color actual para dibujar a negro puro (0,0,0). Este color se usará para las aristas del cubo.

Línea 52: glBegin(GL_LINES)

- Dentro de draw_cube: Inicia la definición de un grupo de primitivas de tipo "lines" (líneas). Los vértices especificados a partir de ahora se interpretarán como pares que forman segmentos de línea.

Línea 53: for edge in edges:

- Dentro de draw_cube: Inicia un bucle que itera sobre la lista edges. Cada edge es una tupla de dos índices de vértice.

Línea 54: for vertex in edge:

- Dentro del bucle for edge: Inicia un bucle anidado que itera sobre los dos índices de vértice que componen la edge actual.

Línea 55: glVertex3fv(vertices[vertex])

- Dentro del bucle for vertex: Especifica un vértice. vertices[vertex] accede a las coordenadas reales del vértice para la arista actual. glVertex3fv envía estas coordenadas a OpenGL para que sean parte de la primitiva de línea actual.

Línea 56: glEnd()

- Dentro de draw_cube: Finaliza la definición del grupo de primitivas de tipo GL_LINES. OpenGL procesará los vértices especificados como un conjunto de aristas.

Línea 57: (En blanco)

- Línea en blanco: Separa la definición de la función draw_cube de la definición de la función main.

Línea 58: def main():

- Define la función main, que será el punto de entrada principal del programa y contendrá el bucle de juego y la lógica de OpenGL.

Línea 59: pygame.init()

- Dentro de main: Inicializa todos los módulos importados de Pygame. Esto es necesario antes de usar la mayoría de las funciones de Pygame.

Línea 60: display = (800, 600)

- Dentro de main: Define una tupla display que especifica el ancho (800 píxeles) y el alto (600 píxeles) de la ventana de visualización.

Línea 61: pygame.display.set_mode(display, DOUBLEBUF | OPENGL)

- Dentro de main: Configura el modo de visualización de Pygame.
- display: Establece las dimensiones de la ventana.
- DOUBLEBUF: Habilita el doble búfer, lo que significa que el dibujo se realiza en un búfer "oculto" y luego se muestra todo de una vez (pygame.display.flip()) para evitar parpadeos.
- OPENGL: Le dice a Pygame que cree una superficie de visualización compatible con OpenGL, permitiendo que las funciones de OpenGL se dibujen en ella.

Línea 62: pygame.display.set_caption("Cubo 3D Interactivo - PyOpenGL")

- Dentro de main: Establece el título de la ventana del juego a la cadena "Cubo 3D Interactivo - PyOpenGL".

Línea 63: (En blanco)

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

Línea 64: glEnable(GL_DEPTH_TEST)

- Dentro de main: Habilita el test de profundidad (Z-buffering). Esto asegura que los objetos más cercanos a la cámara ocluyan correctamente a los objetos más lejanos, incluso si se dibujan en un orden diferente, creando una ilusión de profundidad 3D.

Línea 65: glMatrixMode(GL_PROJECTION)

- Dentro de main: Establece el modo de la matriz actual a la matriz de proyección. Las operaciones de matriz que sigan afectarán a cómo los objetos 3D se "proyectan" en el espacio 2D de la pantalla.

Línea 66: gluPerspective(45, display[0]/display[1], 0.1, 50.0)

- Dentro de main: Configura una matriz de proyección de perspectiva.
- 45: El ángulo de campo de visión vertical (FOV) en grados.
- display[0]/display[1]: La relación de aspecto (ancho/alto) de la ventana de visualización, para evitar distorsiones.
- 0.1: La distancia al plano de recorte cercano (near clipping plane). Los objetos más cercanos que esta distancia no se dibujarán.
- 50.0: La distancia al plano de recorte lejano (far clipping plane). Los objetos más lejanos que esta distancia no se dibujarán.

Línea 67: glMatrixMode(GL_MODELVIEW)

- Dentro de main: Establece el modo de la matriz actual a la matriz de modelado y vista. Las operaciones de matriz que sigan afectarán a la posición y orientación de los objetos en el mundo 3D y a la posición y orientación de la cámara.

Línea 68: (En blanco)

- Dentro de main: Línea en blanco: Separa la configuración inicial de OpenGL de la inicialización de las variables de estado del cubo y la animación.

Línea 69: rot_x = rot_y = rot_z = 0

- Dentro de main: Inicializa las variables rot_x, rot_y, rot_z a 0. Estas variables controlarán los ángulos de rotación del cubo alrededor de los ejes X, Y y Z, respectivamente.

Línea 70: zoom = -7

- Dentro de main: Inicializa la variable zoom a -7. Esta variable representa la traslación del cubo a lo largo del eje Z (distancia de la cámara al cubo). Un valor negativo significa que el cubo está "alejado" de la cámara.

Línea 71: color_intensity = 1.0

- Dentro de main: Inicializa la variable color_intensity a 1.0. Esta variable controlará la intensidad del color de las caras del cubo (brillo), con 1.0 siendo el color original.

Línea 72: (En blanco)

- Dentro de main: Línea en blanco: Separa las variables de estado de las variables de control del bucle de juego.

Línea 73: clock = pygame.time.Clock()

- Dentro de main: Crea un objeto Clock de Pygame. Se utilizará para controlar la velocidad de fotogramas del juego.

Línea 74: running = True

- Dentro de main: Inicializa una variable booleana running a True. Esta variable controlará la ejecución del bucle principal del juego. Cuando se establezca en False, el bucle terminará.

Línea 75: ticks_ago = 0

- Dentro de main: Inicializa una variable ticks_ago a 0. Esta variable se utilizará para controlar un comportamiento automático de rotación y zoom que alterna cada cierto número de fotogramas.

Línea 76: while running:

- Dentro de main: Inicia el bucle principal del juego. Todo el código indentado bajo esta línea se ejecutará repetidamente mientras running sea True.

Línea 77: dt = clock.tick(60) / 1000

- Dentro del bucle while: Calcula dt (delta time).
- clock.tick(60): Pausa el bucle hasta que haya transcurrido suficiente tiempo para mantener un máximo de 60 fotogramas por segundo (FPS). Devuelve el número de milisegundos transcurridos desde la última llamada.

- / 1000: Convierte los milisegundos a segundos. dt representa el tiempo transcurrido desde el último fotograma en segundos, lo cual es crucial para hacer los movimientos y animaciones independientes de la velocidad de fotogramas.

Línea 78: ticks_ago += 1

- Dentro del bucle while: Incrementa el contador ticks_ago en 1 en cada fotograma.

Línea 79: print(ticks_ago)

- Dentro del bucle while: Imprime el valor actual de ticks_ago en la consola.

Línea 80: (En blanco)

- Dentro del bucle while: Línea en blanco: Separa el cálculo de dt y ticks_ago del manejo de eventos.

Línea 81: for event in pygame.event.get():

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

Línea 82: if event.type == QUIT:

- Dentro del bucle for event: Comprueba si el tipo de evento actual es QUIT. El evento QUIT se genera cuando el usuario hace clic en el botón de cerrar la ventana.

Línea 83: running = False

- Dentro de la condición if event.type == QUIT: Si el evento es QUIT, establece la variable running a False, lo que hará que el bucle principal del juego termine después de esta iteración.

Línea 84: (En blanco)

- Dentro del bucle while: Línea en blanco: Separa el manejo de eventos del comportamiento automático del cubo.

Línea 85: random = np.random.randint(0, 1)

- Dentro del bucle while: Llama a np.random.randint(0, 1). Esta función genera un número entero aleatorio en el intervalo [0, 1), lo que en la práctica significa que siempre generará el número 0. El valor resultante se asigna a la variable

random, pero esta variable no se utiliza en ninguna parte posterior del código, haciendo esta línea ineficaz en su propósito aparente de introducir aleatoriedad.

Línea 86: if ticks_ago > 140:

- Dentro del bucle while: Inicia una condición que comprueba si el contador ticks_ago es mayor que 140. Esto controla la dirección del movimiento automático del cubo.

Línea 87: rot_x -= 90 * dt

- Dentro de la condición if ticks_ago > 140: Si ticks_ago es mayor que 140, disminuye la rotación del cubo alrededor del eje X. La rotación se ajusta en 90 grados por segundo (90 * dt).

Línea 88: rot_y += 90 * dt

- Dentro de la condición if ticks_ago > 140: Aumenta la rotación del cubo alrededor del eje Y en 90 grados por segundo.

Línea 89: zoom += 2 * dt

- Dentro de la condición if ticks_ago > 140: Aumenta la variable zoom en 2 unidades por segundo, lo que acerca el cubo a la cámara (ya que zoom es negativo).

Línea 90: if ticks_ago > 280:

- Dentro de la condición if ticks_ago > 140: Inicia una condición anidada que comprueba si ticks_ago es mayor que 280.

Línea 91: ticks_ago = 0

- Dentro de la condición anidada if ticks_ago > 280: Si ticks_ago supera 280, lo reinicia a 0. Esto hace que el ciclo de movimiento automático se repita.

Línea 92: else:

- Dentro del bucle while: La parte else de la condición if ticks_ago > 140. Esto se ejecuta si ticks_ago es menor o igual a 140.

Línea 93: rot_x += 90 * dt

- Dentro del bloque else: Aumenta la rotación del cubo alrededor del eje X en 90 grados por segundo.

Línea 94: rot_y -= 90 * dt

- Dentro del bloque else: Disminuye la rotación del cubo alrededor del eje Y en 90 grados por segundo.

Línea 95: zoom -= 2 * dt

- Dentro del bloque else: Disminuye la variable zoom en 2 unidades por segundo, lo que aleja el cubo de la cámara.

Línea 96: (En blanco)

- Dentro del bucle while: Línea en blanco: Separa el control de movimiento automático del manejo de la entrada del teclado.

Línea 97: keys = pygame.key.get_pressed()

- Dentro del bucle while: Obtiene el estado actual de todas las teclas del teclado. keys es una lista booleana donde keys[key_constant] es True si la tecla está presionada.

Línea 98: if keys[K_UP]:

- Dentro del bucle while: Comprueba si la tecla "flecha arriba" (K_UP) está presionada.

Línea 99: rot_x -= 90 * dt

- Dentro de la condición if keys[K_UP]: Si K_UP está presionada, disminuye la rotación del cubo alrededor del eje X (rota hacia arriba).

Línea 100: if keys[K_DOWN]:

- Dentro del bucle while: Comprueba si la tecla "flecha abajo" (K_DOWN) está presionada.

Línea 101: rot_x += 90 * dt

- Dentro de la condición if keys[K_DOWN]: Si K_DOWN está presionada, aumenta la rotación del cubo alrededor del eje X (rota hacia abajo).

Línea 102: if keys[K_LEFT]:

- Dentro del bucle while: Comprueba si la tecla "flecha izquierda" (K_LEFT) está presionada.

Línea 103: rot_y -= 90 * dt

- Dentro de la condición if keys[K_LEFT]: Si K_LEFT está presionada, disminuye la rotación del cubo alrededor del eje Y (rota hacia la izquierda).

Línea 104: if keys[K_RIGHT]:

- Dentro del bucle while: Comprueba si la tecla "flecha derecha" (K_RIGHT) está presionada.

Línea 105: rot_y += 90 * dt

- Dentro de la condición if keys[K_RIGHT]: Si K_RIGHT está presionada, aumenta la rotación del cubo alrededor del eje Y (rota hacia la derecha).

Línea 106: if keys[K_q]:

- Dentro del bucle while: Comprueba si la tecla 'q' (K_q) está presionada.

Línea 107: rot_z -= 90 * dt

- Dentro de la condición if keys[K_q]: Si 'q' está presionada, disminuye la rotación del cubo alrededor del eje Z (rota en sentido antihorario).

Línea 108: if keys[K_e]:

- Dentro del bucle while: Comprueba si la tecla 'e' (K_e) está presionada.

Línea 109: rot_z += 90 * dt

- Dentro de la condición if keys[K_e]: Si 'e' está presionada, aumenta la rotación del cubo alrededor del eje Z (rota en sentido horario).

Línea 110: if keys[K_w]:

- Dentro del bucle while: Comprueba si la tecla 'w' (K_w) está presionada.

Línea 111: zoom += 5 * dt

- Dentro de la condición if keys[K_w]: Si 'w' está presionada, aumenta la variable zoom, acercando el cubo a la cámara.

Línea 112: if keys[K_s]:

- Dentro del bucle while: Comprueba si la tecla 's' (K_s) está presionada.

Línea 113: zoom -= 5 * dt

- Dentro de la condición if keys[K_s]: Si 's' está presionada, disminuye la variable zoom, alejando el cubo de la cámara.

Línea 114: if keys[K_a]:

- Dentro del bucle while: Comprueba si la tecla 'a' (K_a) está presionada.

Línea 115: color_intensity = max(0.1, color_intensity - 0.5 * dt)

- Dentro de la condición if keys[K_a]: Si 'a' está presionada, disminuye la color_intensity (oscurece el cubo) en 0.5 unidades por segundo, pero asegura que no baje de 0.1 usando max().

Línea 116: if keys[K_d]:

- Dentro del bucle while: Comprueba si la tecla 'd' (K_d) está presionada.

Línea 117: color_intensity = min(2.0, color_intensity + 0.5 * dt)

- Dentro de la condición if keys[K_d]: Si 'd' está presionada, aumenta la color_intensity (aclara el cubo) en 0.5 unidades por segundo, pero asegura que no supere 2.0 usando min().

Línea 118: if keys[K_r]:

- Dentro del bucle while: Comprueba si la tecla 'r' (K_r) está presionada.

Línea 119: rot_x = rot_y = rot_z = 0

- Dentro de la condición if keys[K_r]: Si 'r' está presionada, reinicia los ángulos de rotación rot_x, rot_y, rot_z a 0.

Línea 120: zoom = -7

- Dentro de la condición if keys[K_r]: Reinicia el zoom a su valor inicial de -7.

Línea 121: color_intensity = 1.0

- Dentro de la condición if keys[K_r]: Reinicia la color_intensity a su valor inicial de 1.0.

Línea 122: (En blanco)

- Dentro del bucle while: Línea en blanco: Separa el manejo del teclado del bloque de renderizado de OpenGL.

Línea 123: glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

- Dentro del bucle while: Limpia los búferes de color y profundidad.
- GL_COLOR_BUFFER_BIT: Borra el búfer de color (la pantalla actual) con el color de fondo establecido por glClearColor() (por defecto, negro).

- GL_DEPTH_BUFFER_BIT: Borra el búfer de profundidad, reiniciando la información de profundidad para el nuevo fotograma.

Línea 124: glLoadIdentity()

- Dentro del bucle while: Carga la matriz identidad en la matriz actual (que está en modo GL_MODELVIEW). Esto "reinicia" la transformación para el objeto, eliminando cualquier rotación, traslación o escalado anterior.

Línea 125: glTranslatef(0, 0, zoom)

- Dentro del bucle while: Aplica una traslación a la matriz GL_MODELVIEW. Mueve el origen del sistema de coordenadas a la posición (0, 0, zoom). Esto efectivamente aleja o acerca el cubo de la cámara.

Línea 126: glRotatef(rot_x, 1, 0, 0)

- Dentro del bucle while: Aplica una rotación alrededor del eje X. El cubo rotará rot_x grados alrededor del eje X. Estas rotaciones se concatenan en el orden inverso al que se aplican (última llamada, primera rotación).

Línea 127: glRotatef(rot_y, 0, 1, 0)

- Dentro del bucle while: Aplica una rotación alrededor del eje Y. El cubo rotará rot_y grados alrededor del eje Y.

Línea 128: glRotatef(rot_z, 0, 0, 1)

- Dentro del bucle while: Aplica una rotación alrededor del eje Z. El cubo rotará rot_z grados alrededor del eje Z.

Línea 129: (En blanco)

- Dentro del bucle while: Línea en blanco: Separa las transformaciones de la llamada a la función de dibujo.

Línea 130: draw_cube(color_intensity)

- Dentro del bucle while: Llama a la función draw_cube que se definió anteriormente, pasándole el color_intensity actual para ajustar el brillo del cubo.

Línea 131: (En blanco)

- Dentro del bucle while: Línea en blanco: Separa la llamada al dibujo del intercambio de búferes.

Línea 132: pygame.display.flip()

- Dentro del bucle while: Actualiza toda la pantalla para mostrar lo que se ha dibujado en el búfer de fondo. Esto es parte del doble búfer y evita el parpadeo.

Línea 133: (En blanco)

- Línea en blanco: Separa el bucle principal de juego de las funciones de limpieza.

Línea 134: pygame.quit()

- Dentro de main: Desinicializa todos los módulos de Pygame. Libera los recursos que Pygame estaba utilizando.

Línea 135: sys.exit()

- Dentro de main: Termina el programa Python. Esto asegura una salida limpia de la aplicación.

Línea 136: (En blanco)

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

Línea 137: if __name__ == "__main__":

- Este es un bloque estándar en Python que garantiza que el código indentado debajo solo se ejecute cuando el script se ejecuta directamente (no cuando se importa como un módulo en otro script).

Línea 138: main()

- Dentro de la condición if __name__ == "__main__":: Si el script se ejecuta directamente, llama a la función main(), iniciando así la aplicación.

Resumen del Código

Este código Python crea una aplicación de escritorio interactiva que muestra un cubo en 3D utilizando Pygame para la gestión de ventanas y eventos, y PyOpenGL para el renderizado gráfico.

1. **Definición del Modelo 3D:** Al inicio del script, se definen las coordenadas de los vertices del cubo, las edges (aristas) que conectan esos vértices y las faces (caras) que forman la superficie del cubo. También se define una lista de colors para pintar cada una de las seis caras.

2. **Función de Dibujo (draw_cube):** Una función draw_cube toma un factor de intensity y se encarga de renderizar el cubo. Utiliza primitivas OpenGL (GL_QUADS para las caras y GL_LINES para las aristas) para dibujar el cubo. Los colores de las caras se ajustan multiplicándolos por la intensity y se aseguran de permanecer dentro del rango RGB válido (0.0 a 1.0).
3. **Configuración de Pygame y OpenGL:** La función main inicializa Pygame, crea una ventana de 800x600 píxeles configurada para renderizado OpenGL con doble búfer. Luego, configura el entorno de OpenGL habilitando el test de profundidad y definiendo una proyección de perspectiva para la cámara virtual.
4. **Bucle Principal del Juego:** El corazón de la aplicación es un bucle while running. En cada iteración:
    - Controla la velocidad de fotogramas a 60 FPS y calcula dt (delta tiempo) para movimientos independientes del framerate.
    - Gestiona los eventos de Pygame, como cerrar la ventana.
    - Implementa un movimiento automático del cubo (rotación y zoom) que alterna su dirección cada 140 fotogramas.
    - Responde a la entrada del teclado, permitiendo al usuario:
        - Rotar el cubo alrededor de los ejes X, Y y Z (flechas, Q/E).
        - Acercar o alejar el cubo (W/S).
        - Ajustar la intensidad del color de las caras (A/D).
        - Reiniciar la posición, rotación, zoom y color a sus valores iniciales (R).
    - **Renderizado:** Borra la pantalla, reinicia la matriz de modelado y vista de OpenGL a la identidad, y luego aplica las transformaciones de traslación (zoom) y rotación (en los tres ejes) según los valores actuales de las variables de estado. Finalmente, llama a draw_cube para dibujar el cubo con la intensidad de color actual.
    - pygame.display.flip(): Muestra el contenido del búfer de dibujo en la pantalla.
5. **Finalización:** Una vez que el bucle termina (por ejemplo, al cerrar la ventana), Pygame y el programa se cierran limpiamente.

En resumen, el código crea una demostración interactiva de un cubo 3D que los usuarios pueden manipular en tiempo real (rotar, hacer zoom, ajustar color) y que también tiene un movimiento automático predefinido.