```python
"""
Description:
This program draws and rotates a cube according to user input using Pygame.
The user may change the direction of rotation with their mouse.
Friction is also added to slow the cube down over time.
"""


# ----- IMPORTS -----


import math as m
import pygame as py


# ----- CONSTANTS -----


WIDTH        =  600       # Screen width
HEIGHT       =  600       # Screen height
BGCOLOR      =  'black'   # Background color
TEXTCOLOR    =  'white'   # Color of the cube lines
THICK        =  3         # Line thickness
DISTANCE     =  600       # Distance from cube to camera
K1           =  600       # Perspective constant
FPS          =  20        # Speed of animation
SENSITIVITY  =  0.0008    # Sensitivity for user input
CUBESIZE     =  100       # Size of the cube on the screen


# ----- OTHER VARIABLES -----


running       = True   # Allows for pausing
mouseDownPos  = None   # Starting point for rotation by the user

# Define initial rotation angles
A = 0.00 # x-axis
B = 0.00 # y-axis

# Dictionary of different shapes with their corresponding vertices and edges
# 'vertices' represent the 3D coordinates of a given shape's points
# 'edges' represents the lines connecting those points, where the two numbers represent the indices of the vertices
shapes = {
  'triangle': {
    'vertices': [
      [+0, +1.1547, 0],
      [-1, -0.5774, 0],
      [+1, -0.5774, 0],
    ],
    'edges': [
      [0, 1], [1, 2], [2, 0]
    ],
    'color': 'red'
  },
  'square': {
    'vertices': [
      [+1, +1, 0],
      [-1, +1, 0],
      [-1, -1, 0],
      [+1, -1, 0],
    ],
    'edges': [
      [0, 1], [1, 2],
      [2, 3], [3, 0]
    ],
    'color': 'blue'
  },
  'cube': {
    'vertices': [
      [+1, +1, +1],
      [-1, +1, +1],
      [+1, -1, +1],
      [+1, +1, -1],
      [-1, -1, +1],
      [+1, -1, -1],
      [-1, +1, -1],
      [-1, -1, -1],
    ],
```

```
    'edges': [
      [0, 1], [3, 6],
      [1, 4], [6, 7],
      [2, 4], [5, 7],
      [0, 2], [3, 5],
      [2, 5], [1, 6],
      [0, 3], [4, 7]
    ],
    'color': 'green'
  },
  'octahedron': {
    'vertices': [
      [+1.5, +0.0, +0.0],
      [+0.0, +1.5, +0.0],
      [-1.5, +0.0, +0.0],
      [+0.0, -1.5, +0.0],
      [+0.0, +0.0, +1.5],
      [+0.0, +0.0, -1.5]
    ],
    'edges': [
      [0, 1], [1, 2],
      [2, 3], [3, 0],
      [0, 4], [1, 4],
      [2, 4], [3, 4],
      [0, 5], [1, 5],
      [2, 5], [3, 5]
    ],
    'color': 'orange'
  },
}

# Current shape
curShapes = ['cube']


# ----- TRANSFORMATION FUNCTIONS -----


# Rotates the given point using angles A, B, and C
def rot(x, y, z):
  # Rotation around x-axis
  xAxis_x = x
  xAxis_y = y * m.cos(A) - z * m.sin(A)
  xAxis_z = y * m.sin(A) + z * m.cos(A)

  # Rotation aroudn y-axis
  yAxis_x = xAxis_x * m.cos(B) + xAxis_z * m.sin(B)
  yAxis_y = xAxis_y
  yAxis_z = xAxis_z * m.cos(B) - xAxis_x * m.sin(B)

  return round(yAxis_x, 6), round(yAxis_y, 6), round(yAxis_z, 6)


# Projects the given point using perspective
def project(x, y, z):
  # Transform (x, y, z) coordinates into (x, y) coordinates with perspective
  factor = K1 / (z + DISTANCE)
  projX = int(WIDTH / 2 - x * factor)
  projY = int(HEIGHT / 2 + y * factor)

  return projX, projY


# Function that rotates (and projects) the cube and draws it
# PERSONAL PROJECT REFERENCE FUNCTION
def update(curShapes):
  # Reset the Screen
  screen.fill(BGCOLOR)

  # Draw the buttons
  drawButtons()

  global shapes
  for shape in shapes:
    # Iterate through all shapes. If a shape is in the list of current shapes, draw and rotate it
    if shape in curShapes:
      # Temporary Points list to replace the points in vertices
      Points = []
      vertices = shapes[shape]['vertices']
      edges = shapes[shape]['edges']
```

```
      color = shapes[shape]['color']

      # Iteration to acquire rotated point
      for v in range(len(vertices)):
        vertex = vertices[v] # Define the point
        x, y, z = rot(vertex[0], vertex[1], vertex[2]) # Rotate the point
        # Avoid dividing by 0 for when the point runs through the projection function
        if (z + DISTANCE == 0):
          z += 0.1
        vertices[v] = [x,y,z] # Change vertices list with new point
        Points.append(project(x * CUBESIZE, y * CUBESIZE, z * CUBESIZE)) # Add rotated point to the 'Points' list

      # Iteration to draw projected point
      for edge in edges:
        py.draw.line(screen, color, Points[edge[0]], Points[edge[1]], THICK)

  # Update the screen
  py.display.flip()
  clock.tick(FPS)


# ----- INITIATION -----


# Initiate Pygame
py.init()
py.display.set_caption("Rotating Cube")
screen = py.display.set_mode((WIDTH, HEIGHT))
clock = py.time.Clock()


# ----- SHAPE BUTTONS -----


# Button variables
buttons   = []
yPos      = HEIGHT - 100
btnWidth  = 120 # Width of each button
btnHeight = 50  # Height of each button
spacing   = (WIDTH - len(shapes) * btnWidth) // (len(shapes) + 1)

# Create a button for each shape
for idx, shape in enumerate(shapes.keys()):
  xPos = spacing + idx * (btnWidth + spacing) # Calculate x position of each button
  rect = py.Rect(xPos, yPos, btnWidth, btnHeight) # Create rectangle object
  buttons.append((shape, rect)) # Add the rectangle object to the list of buttons

# Function to draw the buttons
def drawButtons():
  font = py.font.SysFont('Courier', 15) # Creates a font object
  for shape, rect in buttons:
    color = shapes[shape]['color']
    py.draw.rect(screen, color, rect, 2) # Draws the button outlinect
    text = font.render(shape, True, 'white') # Creates a text object for each shape
    textRect = text.get_rect(center=rect.center) # Creates the text rectangle
    screen.blit(text, textRect) # Draws the text in the button


# ----- MAINLOOP -----


# Animate the movement through continuous calling of the cube() function
while True:
  # Friction-like Force
  A *= 0.96
  B *= 0.96

  # Quit the game if window closed
  for event in py.event.get():
    if event.type == py.QUIT:
      py.quit()

    # Pause the animation
    elif event.type == py.KEYDOWN:
      if event.key == py.K_SPACE:
        running = not running

    # Implementing user input for rotation speed
    elif event.type == py.MOUSEBUTTONDOWN:
      # NOTE: Google's AI Overview provided the knowledge that left-click is represented by event.button == 1
```

```
240       if event.button == 1: # left-click
241         # Differentiate between hitting a button and changing the rotation
242         buttonClicked = False
243 
244         # Check if the mouse is over a button
245         for shape, rect in buttons:
246           # If so, change the current shape
247           if rect.collidepoint(event.pos):
248             buttonClicked = True
249             if shape in curShapes:
250               curShapes.remove(shape)
251             else:
252               curShapes.append(shape)
253             # Redraw the buttons
254             screen.fill(BGCOLOR)
255 
256         # Otherwise, begin changing the rotation speed
257         if not buttonClicked:
258           mouseDownPos = event.pos
259 
260     elif event.type == py.MOUSEBUTTONUP:
261       if event.button == 1 and mouseDownPos is not None:
262         # Acquire points
263         x1, y1 = mouseDownPos
264         x2, y2 = event.pos
265         # Calculate the differences
266         changeX = x2 - x1
267         changeY = y2 - y1
268         # Change Rotation speed
269         # Order is reversed because a horizontal swipe (change in x) should change the y-axis
270         B += changeX * SENSITIVITY
271         A += changeY * SENSITIVITY
272         # Resent position where mouse is initially pressed
273         mouseDownPos = None
274 
275   # Update to next frame
276   if running:
277     update(curShapes)
278 
279 # hashtag best code i've ever written
```