```
# Importing the necessary Modules
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *

# Initial vertical positions of the triangles
yellow_triangle_y = 0.0  # Vertical position for the yellow triangle
red_triangle_y = 0.0  # Vertical position for the red triangle

# Movement directions for both triangles
direction_yellow = 0.01  # Yellow triangle initially rises
direction_red = -0.01  # Red triangle initially descends

# Disply callback function
def display():

    glClear(GL_COLOR_BUFFER_BIT)  # Reset background
    render_Scene()  # Call function to draw the scene
    glutSwapBuffers()  # Update the display by swapping buffers

# Scene render function
def render_Scene():

    # Apply a translation for the red triangle's position
    glPushMatrix()  # Save the current matrix state
    glTranslatef(0.0, red_triangle_y, 0)  # Modify the red triangle's position
vertically

    # Render the first triangle in red
    glColor3f(1.0,0.0,0.0);
    glBegin(GL_TRIANGLES);
    glVertex3f(-0.3,-0.1,-0.8);
    glVertex3f(0.3,-0.1,-1);
    glVertex3f(0,0.3,-0.9);
    glEnd();
    glPopMatrix()  # Restore the matrix state

    # Apply a translation for the yellow triangle's position
    glPushMatrix()  # Save the current matrix state
    glTranslatef(0.0, yellow_triangle_y, 0)  # Shift the yellow triangle
vertically

    # Render the second triangle in yellow
    glColor3f(1.0,1.0,0.0);
    glBegin(GL_TRIANGLES);
    glVertex3f(-0.3,0.2,0.0);
    glVertex3f(-0.7,0.5,0.0);
    glVertex3f(-0.5,0.7,0.0);
    glEnd();
    glPopMatrix()  # Restore the matrix state

# Function to update positions of the triangles for animation
```

```
def update_position(value):
    global red_triangle_y, yellow_triangle_y, direction_red, direction_yellow

    # Update the yellow triangle's Y (vertical) position
    yellow_triangle_y += direction_yellow

    # Reset yellow triangle to the bottom when it moves above the screen
    if yellow_triangle_y <= -1:
        yellow_triangle_y = 1.0  # Reset to the top
    elif yellow_triangle_y >= 1:
        yellow_triangle_y = -1.0  # Reset to the bottom


    # Update the red triangle's Y (vertical) position
    red_triangle_y += direction_red

    # Reset red triangle to the top when it moves below the screen
    if red_triangle_y <= -1:
        red_triangle_y = 1.0  # Reset to the top
    elif red_triangle_y >= 1:
        red_triangle_y = -1.0  # Reset to the bottom

    # Redisplay the window after updates
    glutPostRedisplay()  # Redraw the scene
    glutTimerFunc(16, update_position, 0)  # Continue calling the update function

# Initialize GLUT
glutInit()
glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH)  # Double buffer and RGB
glutInitWindowSize(500, 500)  # Set window size
glutCreateWindow("Red and Yellow Triangles with Continuous Movement")  # Create
window with title
glutInitWindowPosition(50, 50)  # Set window position

# Set the projection matrix to set orthographic view (standard for 2D)
glMatrixMode(GL_PROJECTION)
glLoadIdentity()
glOrtho(-1, 1, -1, 1, -1, 1)  # Set orthographic projection from -1 to 1

# Set the display callback function
glutDisplayFunc(display)

# Set the update callback function with a timer for continuous movement
glutTimerFunc(25, update_position, 0)  # Start the update loop

# Start the event loop
glutMainLoop()
```

Red and Yellow Triangle Animation with OpenGL and GLUT

Introduction

This report details an OpenGL and GLUT program that creates an animation of two continuously moving triangles, one red and one yellow. The animation utilizes basic concepts of graphics programming, including:

- **Triangle Drawing:** The program defines and draws two triangles using OpenGL functions.
- **Color Specification:** Different colors are assigned to each triangle using `glColor3f`.
- **Transformation Matrix:** The `glPushMatrix` and `glPopMatrix` functions manage the transformation matrix to translate (move) the triangles independently.
- **Animation Loop:** The `glutTimerFunc` function creates an animation loop that continuously updates the positions of the triangles.

Functionality

The program operates by:

1. **Importing Libraries:** Necessary modules from OpenGL libraries are imported for graphics, windowing, and utility functions.
2. **Triangle Initialization:** Initial vertical positions and movement directions are defined for both triangles.
3. **Callback Functions:**
    - `display()`: Clears the background, renders the scene, and swaps buffers for smooth animation.
    - `render_Scene()`: Applies translations, sets drawing colors, and defines the triangles using `glBegin` and `glEnd`.
    - `update_position(value)`: Updates the vertical positions of the triangles based on their movement directions. It also resets the positions when they move off-screen. Finally, it triggers a redraw and schedules itself for continuous updates.
4. **GLUT Configuration:**
    - The program initializes GLUT for window management.
    - It sets the display mode, defines window size and title, and configures the projection matrix for an orthographic view.
5. **Main Loop:**
    - `glutDisplayFunc` defines the callback function for redrawing the scene.
    - `glutTimerFunc` starts the animation loop by calling `update_position` after an initial delay.
    - `glutMainLoop` initiates the main event loop for handling user input and continuous redrawing.

Code Analysis

The provided code showcases the following key aspects:

- **Clear variable naming:** Variable names like `yellow_triangle_y`, `direction_red`, etc., enhance code readability.
- **Comments:** While minimal, comments explain certain sections, improving code understanding.
- **Modularization:** Separate functions for rendering and position updates promote code organization.

- **Double Buffering:** The program uses double buffering with `glutSwapBuffers` for smooth animation without flickering.

Conclusion

This program demonstrates the fundamentals of animation in OpenGL and GLUT. It provides a basic example of creating and animating simple shapes like triangles. Further exploration could involve incorporating more complex shapes, user interaction, and advanced animation techniques.