Lab File

For

Computer Graphics

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

GALGOTIAS UNIVERSITY, GREATER NOIDA

UTTAR PRADESH

SUBMITTED BY:

Sarvesh Pratap Yadav 23SCSE1011213

.

SUBMITTED TO:

MR. VIMAL SINGH

3D Object Rotation Product Showcase using Python

Problem Understanding:

The goal of this project is to create a visual representation of a car that can rotate in a 3D-like manner using 2D graphics. The main objectives are:

1. To simulate the rotation of a car model, showcasing its various parts (body, windows, wheels).
2. To provide a clear and engaging visual experience for users, similar to what is seen in online storefronts for car brands.
3. To understand and apply basic 3D transformations in a 2D environment.

2D Transformation Used:

2D transformations are mathematical operations that move or change graphical objects in the two-dimensional plane. They are fundamental in computer graphics to simulate motion, scaling, rotation, and more. In this project, the following transformations were primarily used:

1. Translation

Translation involves moving every point of an object by a certain offset in the x and y directions. The car's parts are drawn relative to a central point (the car's center). After rotation, the points are translated back to their original position on the screen.

2. Rotation

The car's parts are defined as points in a 2D plane. By applying a rotation transformation, we can simulate the effect of the car rotating around its center. The rotation is achieved using the following formulas:

- For a point (x,y) rotated by an angle θ: $x' = x \cdot \cos(\theta) - y \cdot \sin(\theta)$

Implementation:

The implementation consists of the following key components:

1. **Car Representation**: The car is represented using polygons for the body and windows, and circles for the wheels. Each part is defined by its coordinates relative to the car's center.

2. **Rotation Function**: A function is created to rotate points around the origin using the rotation formulas mentioned above.

3. **Drawing Functions**: Functions are defined to draw the car's body, windows, and wheels. The wheels also include spokes that simulate rotation.

4. **Animation Loop**: A main loop continuously updates the angle of rotation, redraws the car, and introduces a delay to control the speed of rotation.

Code:

```
import turtle
import math
import time
screen = turtle.Screen()
screen.title("3D Car Rotation Simulation")
screen.bgcolor("white")
screen.setup(width=800, height=600)
screen.tracer(0)
pen = turtle.Turtle()
pen.hideturtle()
pen.speed(0)
pen.pensize(3)
center_x = 0
center_y = 0
def rotate_point(x, y, angle_deg):
    """Rotate point (x,y) by angle_deg around origin."""
    angle_rad = math.radians(angle_deg)
    cos_ang = math.cos(angle_rad)
    sin_ang = math.sin(angle_rad)
    x_rot = x * cos_ang - y * sin_ang
    y_rot = x * sin_ang + y * cos_ang
    return x_rot, y_rot
body_points = [
    (-150, -40),
    (-150, 40),
    (-60, 70),
    (50, 70),
    (120, 40),
    (120, -40),
]
left_window_points = [
    (-50, 50),
    (-50, 65),
    (0, 65),
    (0, 50),
]
right_window_points = [
```

```python
    (10, 50),
    (10, 65),
    (70, 65),
    (70, 50),
]
wheel_centers = [
    (-90, -40),
    (90, -40),
]
wheel_radius = 30
body_color ="blue"
window_color = "green"
wheel_color = "black"
rim_color = "red"
def draw_polygon(points, fill_color, outline_color="black"):
    pen.penup()
    first = points[0]
    pen.goto(first[0], first[1])
    pen.pendown()
    pen.color(outline_color, fill_color)
    pen.begin_fill()
    for point in points[1:]:
        pen.goto(point[0], point[1])
    pen.goto(first[0], first[1])
    pen.end_fill()
def draw_wheel(center_x, center_y, radius, angle_deg=0):
    pen.penup()
    pen.goto(center_x, center_y - radius)
    pen.setheading(0)
    pen.pendown()
    pen.color("black", wheel_color)
    pen.begin_fill()
    pen.circle(radius)
    pen.end_fill()
    pen.color(rim_color)
    for i in range(6):
        spoke_angle = angle_deg + (360 / 6) * i
```

```python
        rad = math.radians(spoke_angle)
        x_start = center_x + radius * 0.1 * math.cos(rad)
        y_start = center_y + radius * 0.1 * math.sin(rad)
        x_end = center_x + radius * 0.85 * math.cos(rad)
        y_end = center_y + radius * 0.85 * math.sin(rad)
        pen.penup()
        pen.goto(x_start, y_start)
        pen.pendown()
        pen.goto(x_end, y_end)
def rotate_and_draw_car(angle_deg):
    pen.clear()
    rotated_body = [rotate_point(x, y, angle_deg) for x, y in body_points]
    rotated_left_window = [rotate_point(x, y, angle_deg) for x, y in left_window_points]
    rotated_right_window = [rotate_point(x, y, angle_deg) for x, y in right_window_points]
    rotated_wheels = [rotate_point(x, y, angle_deg) for x, y in wheel_centers]
    draw_polygon(rotated_body, body_color)
    draw_polygon(rotated_left_window, window_color)
    draw_polygon(rotated_right_window, window_color)
    spoke_angle = (angle_deg * 5) % 360
    for (wx, wy) in rotated_wheels:
        draw_wheel(wx, wy, wheel_radius, spoke_angle)
    screen.update()
def main():
    angle = 0
    while True:
        rotate_and_draw_car(angle)
        angle = (angle + 2) % 360
        time.sleep(0.05)
if __name__ == "__main__":
    main()
```

Challenges and Solutions:

Several challenges were encountered during the development of this project:

1. **Simulating 3D Rotation**: Since Turtle graphics is 2D, simulating a 3D effect required careful manipulation of points and understanding of rotation.

2. **Performance**: Drawing and redrawing the car in each frame could lead to performance issues if not managed properly.

3. **Mathematical Understanding**: A solid understanding of 2D rotation and transformation helped in accurately simulating the rotation effect.

4. **Efficient Redrawing**: By clearing the screen and redrawing only the necessary parts, performance was optimized. The use of `screen.tracer(0)` and `screen.update()` helped manage the drawing process efficiently.

Screenshots: