Department of Computer Science and Engineering

CSE 422: Computer Graphics Lab

Session 6 : 2D Transformation and animation

Part 1: Translation (Moving Objects)

Theory

Translation moves an object from one position to another by adding translation factors to coordinates.

- **New X = Old X + Tx**
- **New Y = Old Y + Ty**

Example 1.1: Translating a Single Triangle

```
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *


# Original triangle vertices
triangle_vertices = [
    [200, 200],
    [300, 200],
    [250, 300]
]


# Translation values
tx = 100  # Move 100 pixels right
ty = 50   # Move 50 pixels up


def draw_triangle(vertices):
```

```
    glBegin(GL_TRIANGLES)
    # glBegin(GL_LINES)
    # glVertex2f(200, 200)
    # glVertex2f(300, 200)
    # glVertex2f(250, 300)
    # glVertex2f(200, 200)
    # glVertex2f(300, 200)
    # glVertex2f(250, 300)
    for vertex in vertices:

        glVertex2f(vertex[0], vertex[1])
    glEnd()


def translate_point(x, y, tx, ty):
    return [x + tx, y + ty]


def translate_shape(vertices, tx, ty):
    translated = []
    for vertex in vertices:
        new_point = translate_point(vertex[0], vertex[1], tx, ty)
        translated.append(new_point)
    return translated


def iterate():
    glViewport(0, 0, 500, 500)
    glMatrixMode(GL_PROJECTION)
```

```
    glLoadIdentity()
    glOrtho(0.0, 500, 0.0, 500, 0.0, 1.0)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()


def showScreen():
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()
    iterate()


    # Draw original triangle in yellow
    glColor3f(1.0, 1.0, 0.0)
    draw_triangle(triangle_vertices)


    # Translate and draw new triangle in red
    glColor3f(1.0, 0.0, 0.0)
    translated_triangle = translate_shape(triangle_vertices, tx, ty)
    # draw_triangle(translated_triangle)


    glutSwapBuffers()


glutInit()
glutInitDisplayMode(GLUT_RGBA)
glutInitWindowSize(500, 500)
glutInitWindowPosition(0, 0)
wind = glutCreateWindow(b"Translation Example")
```

```
glutDisplayFunc(showScreen)

glutMainLoop()
```

Example 1.2: Animated Translation

```
from OpenGL.GL import *

from OpenGL.GLUT import *

from OpenGL.GLU import *


triangle_vertices = [

    [50, 200],

    [150, 200],

    [100, 300]

]


# Animation variables

tx = 0

ty = 0

direction = 1  # 1 for right, -1 for left


def draw_triangle(vertices):

    glBegin(GL_TRIANGLES)

    for vertex in vertices:

        glVertex2f(vertex[0], vertex[1])

    glEnd()
```

```
def translate_shape(vertices, tx, ty):
    translated = []
    for vertex in vertices:
        translated.append([vertex[0] + tx, vertex[1] + ty])
    return translated


def iterate():
    glViewport(0, 0, 500, 500)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    glOrtho(0.0, 500, 0.0, 500, 0.0, 10.0)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()


def animate(value):
    global tx, ty, direction


    # Move the triangle
    tx += direction * 2
    ty += direction * 2


    # Reverse direction at boundaries
    if tx > 300 or tx < 0:
        direction *= -1


    if ty > 300 or ty < 0:
```

```
        direction *= -1

    glutPostRedisplay()

    glutTimerFunc(16, animate, 3)  # ~60 FPS


def showScreen():

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

    glLoadIdentity()

    iterate()


    glColor3f(0.0, 1.0, 1.0)

    translated_triangle = translate_shape(triangle_vertices, tx, ty)

    draw_triangle(translated_triangle)


    glutSwapBuffers()


glutInit()

glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE)

glutInitWindowSize(500, 500)

glutInitWindowPosition(0, 0)

wind = glutCreateWindow(b"Animated Translation")

glutDisplayFunc(showScreen)

glutTimerFunc(0, animate, 0)

glutMainLoop()
```

Part 2: Rotation (Spinning Objects)

Theory

Rotation rotates an object around a pivot point by an angle θ (theta).

- **New X = (X - Xc) * cos(θ) - (Y - Yc) * sin(θ) + Xc**
- **New Y = (X - Xc) * sin(θ) + (Y - Yc) * cos(θ) + Yc**

Where (Xc, Yc) is the center of rotation.

Example 2.1: Rotating a Triangle by 45 Degrees

```
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *
import math


triangle_vertices = [
    [250, 200],
    [300, 200],
    [275, 250]
]


def draw_triangle(vertices):
    glBegin(GL_TRIANGLES)
    for vertex in vertices:
        glVertex2f(vertex[0], vertex[1])
    glEnd()


def rotate_point(x, y, cx, cy, angle_deg):
```

```
    # Convert angle to radians
    angle_rad = math.radians(angle_deg)

    # Translate point to origin
    temp_x = x - cx
    temp_y = y - cy

    # Apply rotation
    rotated_x = temp_x * math.cos(angle_rad) - temp_y *
math.sin(angle_rad)
    rotated_y = temp_x * math.sin(angle_rad) + temp_y *
math.cos(angle_rad)

    # Translate back
    new_x = rotated_x + cx
    new_y = rotated_y + cy

    return [new_x, new_y]

def rotate_shape(vertices, cx, cy, angle):
    rotated = []
    for vertex in vertices:
        new_point = rotate_point(vertex[0], vertex[1], cx, cy, angle)
        rotated.append(new_point)
    return rotated

def calculate_center(vertices):
```

```
    cx = sum(v[0] for v in vertices) / len(vertices)
    cy = sum(v[1] for v in vertices) / len(vertices)
    return cx, cy


def iterate():
    glViewport(0, 0, 500, 500)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    glOrtho(0.0, 500, 0.0, 500, 0.0, 1.0)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()


def showScreen():
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()
    iterate()


    # Calculate center of triangle
    cx, cy = calculate_center(triangle_vertices)
    cx = cy = 0  # Rotate around origin (0,0)
    # Draw original triangle in yellow
    glColor3f(1.0, 1.0, 0.0)
    draw_triangle(triangle_vertices)


    # Rotate by 45 degrees and draw in red
    glColor3f(1.0, 0.0, 0.0)
```

```
    rotated_triangle = rotate_shape(triangle_vertices, cx, cy, 45)
    draw_triangle(rotated_triangle)


    glutSwapBuffers()


glutInit()
glutInitDisplayMode(GLUT_RGBA)
glutInitWindowSize(500, 500)
glutInitWindowPosition(0, 0)
wind = glutCreateWindow(b"Rotation Example")
glutDisplayFunc(showScreen)
glutMainLoop()
```

Example 2.2: Continuously Rotating Triangle

```
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *
import math


triangle_vertices = [
    [250, 200],
    [300, 200],
    [275, 250]
]
```

```
# Animation variable
angle = 0


def draw_triangle(vertices):
    glBegin(GL_TRIANGLES)
    for vertex in vertices:
        glVertex2f(vertex[0], vertex[1])
    glEnd()


def rotate_point(x, y, cx, cy, angle_deg):
    angle_rad = math.radians(angle_deg)
    temp_x = x - cx
    temp_y = y - cy
    rotated_x = temp_x * math.cos(angle_rad) - temp_y *
math.sin(angle_rad)
    rotated_y = temp_x * math.sin(angle_rad) + temp_y *
math.cos(angle_rad)
    return [rotated_x + cx, rotated_y + cy]


def rotate_shape(vertices, cx, cy, angle):
    rotated = []
    for vertex in vertices:
        new_point = rotate_point(vertex[0], vertex[1], cx, cy, angle)
        rotated.append(new_point)
    return rotated


def calculate_center(vertices):
```

```
    cx = sum(v[0] for v in vertices) / len(vertices)
    cy = sum(v[1] for v in vertices) / len(vertices)
    return cx, cy


def iterate():
    glViewport(0, 0, 500, 500)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    glOrtho(0.0, 500, 0.0, 500, 0.0, 1.0)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()


def animate(value):
    global angle
    angle = (angle - 5) % 360  # Increment angle and wrap at 360
    glutPostRedisplay()
    glutTimerFunc(16, animate, 0)


def showScreen():
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()
    iterate()


    cx, cy = calculate_center(triangle_vertices)
    cx = cy = 0
    glColor3f(0.0, 1.0, 0.0)
```

```
    rotated_triangle = rotate_shape(triangle_vertices, cx, cy, angle)
    draw_triangle(rotated_triangle)


    glutSwapBuffers()


glutInit()
glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE)
glutInitWindowSize(500, 500)
glutInitWindowPosition(0, 0)
wind = glutCreateWindow(b"Continuous Rotation")
glutDisplayFunc(showScreen)
glutTimerFunc(0, animate, 0)
glutMainLoop()
```

Part 3: Scaling (Resizing Objects)

Theory

Scaling changes the size of an object relative to a fixed point.

- **New X = (X - Xc) * Sx + Xc**
- **New Y = (Y - Yc) * Sy + Yc**

Where Sx and Sy are scaling factors (1.0 = no change, >1 = enlarge, <1 = shrink).

Example 3.1: Scaling a Rectangle

```
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *
```

```
rectangle_vertices = [
    [200, 200],
    [300, 200],
    [300, 300],
    [200, 300]
]

def draw_rectangle(vertices):
    glBegin(GL_QUADS)
    for vertex in vertices:
        glVertex2f(vertex[0], vertex[1])
    glEnd()

def scale_point(x, y, cx, cy, sx, sy):
    # Translate to origin
    temp_x = x - cx
    temp_y = y - cy

    # Apply scaling
    scaled_x = temp_x * sx
    scaled_y = temp_y * sy

    # Translate back
    new_x = scaled_x + cx
    new_y = scaled_y + cy
```

```
    return [new_x, new_y]


def scale_shape(vertices, cx, cy, sx, sy):
    scaled = []
    for vertex in vertices:
        new_point = scale_point(vertex[0], vertex[1], cx, cy, sx, sy)
        scaled.append(new_point)
    return scaled


def calculate_center(vertices):
    cx = sum(v[0] for v in vertices) / len(vertices)
    cy = sum(v[1] for v in vertices) / len(vertices)
    return cx, cy


def iterate():
    glViewport(0, 0, 500, 500)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    glOrtho(0.0, 500, 0.0, 500, 0.0, 1.0)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()


def showScreen():
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()
```

```
    iterate()


    cx, cy = calculate_center(rectangle_vertices)


    # Original rectangle in yellow
    glColor3f(1.0, 1.0, 0.0)
    draw_rectangle(rectangle_vertices)


    # Scaled up (1.5x) in green
    glColor3f(0.0, 1.0, 0.0)
    scaled_up = scale_shape(rectangle_vertices, cx, cy, 1.5, 1.5)
    draw_rectangle(scaled_up)


    # Scaled down (0.5x) in red
    glColor3f(1.0, 0.0, 0.0)
    scaled_down = scale_shape(rectangle_vertices, cx, cy, 0.5, 0.5)
    draw_rectangle(scaled_down)


    glutSwapBuffers()


glutInit()
glutInitDisplayMode(GLUT_RGBA)
glutInitWindowSize(500, 500)
glutInitWindowPosition(0, 0)
wind = glutCreateWindow(b"Scaling Example")
glutDisplayFunc(showScreen)
```

```
glutMainLoop()
```

Example 3.2: Pulsating (Breathing) Animation

```
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *
import math


rectangle_vertices = [
    [200, 200],
    [300, 200],
    [300, 300],
    [200, 300]
]


# Animation variables
scale_factor = 1.0
scale_direction = 1


def draw_rectangle(vertices):
    glBegin(GL_QUADS)
    for vertex in vertices:
        glVertex2f(vertex[0], vertex[1])
    glEnd()
```

```
def scale_point(x, y, cx, cy, sx, sy):
    temp_x = x - cx
    temp_y = y - cy
    scaled_x = temp_x * sx
    scaled_y = temp_y * sy
    return [scaled_x + cx, scaled_y + cy]

def scale_shape(vertices, cx, cy, sx, sy):
    scaled = []
    for vertex in vertices:
        new_point = scale_point(vertex[0], vertex[1], cx, cy, sx, sy)
        scaled.append(new_point)
    return scaled

def calculate_center(vertices):
    cx = sum(v[0] for v in vertices) / len(vertices)
    cy = sum(v[1] for v in vertices) / len(vertices)
    return cx, cy

def iterate():
    glViewport(0, 0, 500, 500)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    glOrtho(0.0, 500, 0.0, 500, 0.0, 1.0)
    glMatrixMode(GL_MODELVIEW)
```

```
    glLoadIdentity()


def animate(value):
    global scale_factor, scale_direction


    scale_factor += scale_direction * (0.01)


    # Reverse direction at boundaries
    if scale_factor > 1.5 or scale_factor < 0.5:
        scale_direction *= -1


    glutPostRedisplay()
    glutTimerFunc(16, animate, 0)


def showScreen():
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()
    iterate()


    cx, cy = calculate_center(rectangle_vertices)


    glColor3f(1.0, 0.0, 1.0)
    scaled_rect = scale_shape(rectangle_vertices, cx, cy, scale_factor,
scale_factor)
    draw_rectangle(scaled_rect)


    glutSwapBuffers()
```

```
glutInit()
glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE)
glutInitWindowSize(500, 500)
glutInitWindowPosition(0, 0)
wind = glutCreateWindow(b"Pulsating Animation")
glutDisplayFunc(showScreen)
glutTimerFunc(0, animate, 0)
glutMainLoop()
```

Part 4: Combined Transformations

Example 4.1: Rotating and Translating Circle

```
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *
import math

# Animation variables
angle = 0
orbit_angle = 0

def draw_point(x, y):
    glPointSize(2)
    glBegin(GL_POINTS)
    glVertex2f(x, y)
    glEnd()
```

```
def draw_circle_points(x, y, cx, cy):
    draw_point(x + cx, y + cy)
    draw_point(y + cx, x + cy)
    draw_point(y + cx, -x + cy)
    draw_point(x + cx, -y + cy)
    draw_point(-x + cx, -y + cy)
    draw_point(-y + cx, -x + cy)
    draw_point(-y + cx, x + cy)
    draw_point(-x + cx, y + cy)

def draw_circle(radius, cx, cy):
    d = 1 - radius
    x = 0
    y = radius

    draw_circle_points(x, y, cx, cy)

    while x < y:
        if d < 0:
            d = d + 2 * x + 3
        else:
            d = d + 2 * x - 2 * y + 5
            y -= 1
        x += 1
        draw_circle_points(x, y, cx, cy)
```

```
def rotate_point(x, y, cx, cy, angle_deg):
    angle_rad = math.radians(angle_deg)
    temp_x = x - cx
    temp_y = y - cy
    rotated_x = temp_x * math.cos(angle_rad) - temp_y *
math.sin(angle_rad)
    rotated_y = temp_x * math.sin(angle_rad) + temp_y *
math.cos(angle_rad)
    return rotated_x + cx, rotated_y + cy


def iterate():
    glViewport(0, 0, 500, 500)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    glOrtho(0.0, 500, 0.0, 500, 0.0, 1.0)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()


def animate(value):
    global orbit_angle
    orbit_angle = (orbit_angle - 2) % 360
    glutPostRedisplay()
    glutTimerFunc(16, animate, 0)


def showScreen():
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
```

```
    glLoadIdentity()
    iterate()

    # Center of screen
    center_x, center_y = 250, 250
    orbit_radius = 100

    # Draw center circle (sun)
    glColor3f(1.0, 1.0, 0.0)
    draw_circle(30, center_x, center_y)

    # Calculate orbiting position
    orbit_x = center_x + orbit_radius
    orbit_y = center_y
    new_x, new_y = rotate_point(orbit_x, orbit_y, center_x, center_y,
orbit_angle)

    # Draw orbiting circle (planet)
    glColor3f(0.0, 0.5, 1.0)
    draw_circle(15, int(new_x), int(new_y))

    glutSwapBuffers()

glutInit()
glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE)
glutInitWindowSize(500, 500)
glutInitWindowPosition(0, 0)
```

```
wind = glutCreateWindow(b"Orbit Animation")

glutDisplayFunc(showScreen)

glutTimerFunc(0, animate, 0)

glutMainLoop()
```

Example 4.2: Complex Animation - Rotating Triangle with Scaling

```
from OpenGL.GL import *

from OpenGL.GLUT import *

from OpenGL.GLU import *

import math


triangle_vertices = [

    [250, 200],

    [300, 200],

    [275, 260]

]


# Animation variables

angle = 0

scale = 1.0

scale_direction = 1


def draw_triangle(vertices):

    glBegin(GL_TRIANGLES)

    for vertex in vertices:
```

```
        glVertex2f(vertex[0], vertex[1])
    glEnd()


def rotate_point(x, y, cx, cy, angle_deg):
    angle_rad = math.radians(angle_deg)
    temp_x = x - cx
    temp_y = y - cy
    rotated_x = temp_x * math.cos(angle_rad) - temp_y *
math.sin(angle_rad)
    rotated_y = temp_x * math.sin(angle_rad) + temp_y *
math.cos(angle_rad)
    return [rotated_x + cx, rotated_y + cy]


def scale_point(x, y, cx, cy, s):
    temp_x = x - cx
    temp_y = y - cy
    return [temp_x * s + cx, temp_y * s + cy]


def transform_shape(vertices, cx, cy, angle, scale_factor):
    transformed = []
    for vertex in vertices:
        # First rotate
        rotated = rotate_point(vertex[0], vertex[1], cx, cy, angle)
        # Then scale
        scaled = scale_point(rotated[0], rotated[1], cx, cy, scale_factor)
        transformed.append(scaled)
    return transformed
```

```
def calculate_center(vertices):
    cx = sum(v[0] for v in vertices) / len(vertices)
    cy = sum(v[1] for v in vertices) / len(vertices)
    return cx, cy

def iterate():
    glViewport(0, 0, 500, 500)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    glOrtho(0.0, 500, 0.0, 500, 0.0, 1.0)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()

def animate(value):
    global angle, scale, scale_direction

    angle = (angle + 3) % 360
    scale += scale_direction * 0.01

    if scale > 1.5 or scale < 0.7:
        scale_direction *= -1

    glutPostRedisplay()
    glutTimerFunc(16, animate, 0)
```

```
def showScreen():
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()
    iterate()

    cx, cy = calculate_center(triangle_vertices)

    glColor3f(1.0, 0.5, 0.0)
    transformed = transform_shape(triangle_vertices, cx, cy, angle, scale)
    draw_triangle(transformed)

    glutSwapBuffers()

glutInit()
glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE)
glutInitWindowSize(500, 500)
glutInitWindowPosition(0, 0)
wind = glutCreateWindow(b"Combined Transformations")
glutDisplayFunc(showScreen)
glutTimerFunc(0, animate, 0)
glutMainLoop()
```

Practice Exercises

Exercise 1: Bouncing Ball

Department of Computer Science and Engineering

CSE 422: Computer Graphics Lab

Create a circle that bounces around the screen, reversing direction when it hits the edges.

Exercise 2: Spinning Square

Create a square that continuously rotates around its center while also moving in a circular path around the screen center.

Exercise 3: Growing and Shrinking Star

Draw a star (using triangles) that grows and shrinks while rotating.

Exercise 4: Multiple Objects

Create 3 triangles that rotate at different speeds around different centers.