Lecture: 2D Transformations - Translation and Rotation

Session 1: Translation

1.1 Theory

What is Translation?

- Moving an object from one position to another
- Preserves shape and size
- Requires displacement values (tx, ty)

Translation Matrix

```
[x']   [1  0  tx] [x]
[y'] = [0  1  ty] [y]
[1 ]   [0  0   1] [1]
```

Where:

- (x,y) is the original position
- (x',y') is the new position
- tx is displacement along x-axis
- ty is displacement along y-axis

Key Points:

- Translation is additive
- Order doesn't matter in pure translation
- Easily reversible (use negative values)

1.2 Practical Implementation

```
#include <GL/glut.h>
#include <math.h>

// Global variables for translation
float translateX = 0.0f;
float translateY = 0.0f;

// Initialize window and OpenGL settings
void init() {
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f); // White background
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-10.0, 10.0, -10.0, 10.0);
}

// Draw a colored square
void drawSquare() {
    glBegin(GL_POLYGON);
        glColor3f(1.0f, 0.0f, 0.0f); // Red
        glVertex2f(-1.0f, -1.0f);
        glVertex2f( 1.0f, -1.0f);
        glVertex2f( 1.0f,  1.0f);
        glVertex2f(-1.0f,  1.0f);
    glEnd();
}

// Draw coordinate axes
void drawAxes() {
    glColor3f(0.0f, 0.0f, 0.0f); // Black
    glBegin(GL_LINES);
        // X-axis
        glVertex2f(-10.0f, 0.0f);
        glVertex2f(10.0f, 0.0f);
        // Y-axis
        glVertex2f(0.0f, -10.0f);
        glVertex2f(0.0f, 10.0f);
    glEnd();
}

// Display function
void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    // Draw axes
    drawAxes();

    // Apply translation and draw square
    glPushMatrix();
        glTranslatef(translateX, translateY, 0.0f);
        drawSquare();
    glPopMatrix();
```

```
    // Display current translation values
    glColor3f(0.0f, 0.0f, 0.0f);
    glRasterPos2f(-9.5f, 9.0f);
    char buffer[50];
    sprintf(buffer, "Translation: (%.1f, %.1f)", translateX, translateY);

    glutSwapBuffers();
}

// Keyboard control
void keyboard(unsigned char key, int x, int y) {
    switch(key) {
        case 'w': translateY += 0.5f; break; // Move up
        case 's': translateY -= 0.5f; break; // Move down
        case 'a': translateX -= 0.5f; break; // Move left
        case 'd': translateX += 0.5f; break; // Move right
        case ' ': // Reset position
            translateX = 0.0f;
            translateY = 0.0f;
            break;
    }
    glutPostRedisplay();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(800, 800);
    glutCreateWindow("Translation Demo");

    init();
    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);
    glutMainLoop();
    return 0;
}
```

Session 2: Rotation

2.1 Theory

What is Rotation?

- Turning an object around a fixed point (pivot)
- Defined by an angle θ
- Preserves shape and size

Rotation Matrix

```
[x']   [cos θ  -sin θ  0] [x]
[y'] = [sin θ   cos θ  0] [y]
[1 ]   [  0      0     1] [1]
```

Where:

- θ is the rotation angle (positive = counterclockwise)
- (x,y) is the original position
- (x',y') is the new position

Key Points:

- Rotation is around the origin (0,0)
- Order matters in rotation
- 360° = full rotation

2.2 Practical Implementation

```
#include <GL/glut.h>
#include <math.h>

// Global variables for rotation
float rotateAngle = 0.0f;
bool autoRotate = false;

void init() {
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-10.0, 10.0, -10.0, 10.0);
}

// Draw a triangle with different colored vertices
void drawTriangle() {
    glBegin(GL_TRIANGLES);
        glColor3f(1.0f, 0.0f, 0.0f); // Red
        glVertex2f(-1.0f, -1.0f);

        glColor3f(0.0f, 1.0f, 0.0f); // Green
        glVertex2f(1.0f, -1.0f);

        glColor3f(0.0f, 0.0f, 1.0f); // Blue
        glVertex2f(0.0f, 1.0f);
    glEnd();
}
```

```
// Draw rotation center indicator
void drawCenter() {
    glPointSize(5.0f);
    glColor3f(0.0f, 0.0f, 0.0f);
    glBegin(GL_POINTS);
        glVertex2f(0.0f, 0.0f);
    glEnd();
}

void drawAxes() {
    glColor3f(0.5f, 0.5f, 0.5f); // Gray
    glBegin(GL_LINES);
        glVertex2f(-10.0f, 0.0f);
        glVertex2f(10.0f, 0.0f);
        glVertex2f(0.0f, -10.0f);
        glVertex2f(0.0f, 10.0f);
    glEnd();
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    // Draw axes
    drawAxes();

    // Draw center point
    drawCenter();

    // Apply rotation and draw triangle
    glPushMatrix();
        glRotatef(rotateAngle, 0.0f, 0.0f, 1.0f);
        drawTriangle();
    glPopMatrix();

    // Display current rotation angle
    glColor3f(0.0f, 0.0f, 0.0f);
    glRasterPos2f(-9.5f, 9.0f);
    char buffer[50];
    sprintf(buffer, "Rotation Angle: %.1f degrees", rotateAngle);

    glutSwapBuffers();
}

void keyboard(unsigned char key, int x, int y) {
    switch(key) {
        case 'r': rotateAngle += 5.0f; break;  // Rotate clockwise
        case 'R': rotateAngle -= 5.0f; break;  // Rotate counter-clockwise
        case ' ': rotateAngle = 0.0f; break;   // Reset rotation
        case 'a': autoRotate = !autoRotate; break; // Toggle auto-rotation
    }

    // Keep angle between 0 and 360
    if(rotateAngle >= 360.0f) rotateAngle -= 360.0f;
    if(rotateAngle < 0.0f) rotateAngle += 360.0f;
```

```
    glutPostRedisplay();
}

void update(int value) {
    if(autoRotate) {
        rotateAngle += 2.0f;
        if(rotateAngle >= 360.0f)
            rotateAngle -= 360.0f;
        glutPostRedisplay();
    }
    glutTimerFunc(16, update, 0); // ~60 FPS
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(800, 800);
    glutCreateWindow("Rotation Demo");

    init();
    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);
    glutTimerFunc(0, update, 0);
    glutMainLoop();
    return 0;
}
```

Interactive Controls:

Translation Demo:

- W: Move up
- S: Move down
- A: Move left
- D: Move right
- Space: Reset position

Rotation Demo:

- R: Rotate clockwise
- Shift+R: Rotate counter-clockwise
- A: Toggle auto-rotation
- Space: Reset rotation