RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Assignment

Course Code	CSC309A
Course Name	Computer Graphics
Programme	B.Tech
Department	CSE
Faculty	FET

Name of the Student	Satyajit Ghana
Reg. No.	17ETCS002159
Semester/Year	06/2020
Course Leader(s)	Dr. Subarna Chatterjee

Declaration Sheet

Student Name	Satyajit Ghana		
Reg. No	17ETCS002159		
Programme	B.Tech	Semester/Year	06/2020
Course Code	CSC309A		
Course Title	Computer Graphics		
Course Date		to	
Course Leader	Dr. Subarna Chatterjee		

Declaration

The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

Signature of the Student		Date	
Submission date stamp (by Examination & Assessment Section)			

Signature of the Course Leader and date	Signature of the Reviewer and date

Contents

Declaration Sheet ii

Contents iii

List of Figures iv

1 Question 1 5

1.1 Introduction 5

1.1.1 Translation 5

1.1.2 Rotation 5

1.1.3 Scaling 5

1.1.4 Order of Transformations 6

1.2 Implementation of Transformation 6

1.3 Results with Screenshot and Discussion 11

Bibliography 18

List of Figures

Figure 1 Order of Transformations..... 6
Figure 2 The OpenGL Pipeline 11
Figure 3 Original Polygon 12
Figure 4 Rotated Polygon 13
Figure 5 Translated Polygon 14
Figure 6 Scaled Polygon.....15
Figure 7 Final Transformed Polygon.....16

1 Question 1

Solution to Question No. 1

1.1 Introduction

1.1.1 Translation

Offset `(tx, ty, tz)` is applied to all subsequent coordinates. Effectively moves the origin of coordinate system.

- `x' = x + tx , y' = y + ty, z' = z + tz`
- OpenGL function is `glTranslate`
- `glTranslatef( tx, ty, tz );`

1.1.2 Rotation

Expressed as rotation through angle `θ` about an axis direction `(x,y,z)` .

- OpenGL function is `glRotatef(θ,x,y,z)`. So `glRotatef(30.0, 0.0, 1.0, 0.0)` rotates by 30° about y-axis.
- Note carefully:
  - `glRotate` wants angles in degrees.
  - C math library (sin, cos etc.) wants angles in radians.
  - `degs = rads * 180/π; rads = degs * π / 180`
- Positive angle? Right hand rule: if the thumb point along the vector of rotation, a positive angle has the fingers curling towards the palm.

1.1.3 Scaling

- Multiply subsequent coordinates by scale factors `sx, sy, sz`. (Note: these are not a point, not a vector, just 3 numbers)

  `x' = sx * x , y' = sy * y, z' = sz * z`

- Often `sx = sy = sz` for a uniform scaling effect. If the factors are different, the scaling is called anamorphic.
- OpenGL function – `glScale` For example, `glScalef(0.5,0.5,0.5);` would cause all objects drawn subsequently to be half as big.

1.1.4 Order of Transformations

Call order is the reverse of the order the transforms are applied. Different call orders result in different transforms! Each transform multiplies the object by a matrix that does the corresponding transformation so the transform closest to the object gets multiplied first.

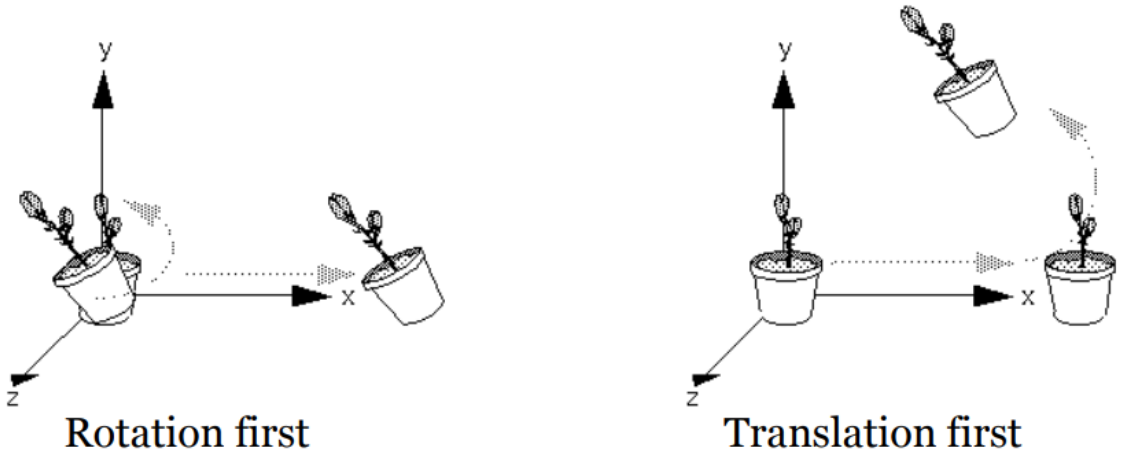

Figure 1 Order of Transformations

Let,

```
glTranslate = Mat Trans
glRotate = Mat Rot
glScale = Mat Scale
DrawCube = v
```

Modelview matrix:

```
Identity -> Trans -> Trans*Rot -> Trans*Rot*Scale -> Trans*Rot*Scale*v
```

Or, `Trans(Rot(Scale*v)))`

So, Scale is applied first, then Rot, then Trans

1.2 Implementation of Transformation

`main.cpp`

```
#include <iostream>
#include <string>
#include <GL/freeglut.h>
```

```c
#define WIN_WIDTH 1000
#define WIN_HEIGHT 1000
#define GRID_X_MIN -8
#define GRID_X_MAX 60
#define GRID_Y_MIN -8
#define GRID_Y_MAX 60

#define WIN_TITLE "Graphics Assignement"

int state = 0;
const unsigned char state_title[][20] = { "ORIGINAL", "ROTATED", "TRANSLATED", "SCALE
D", "FINAL TRANSFORMED" };
int total_states = 5;

void render();
void reshape(int, int);
void keyboard(unsigned char, int, int);
void special_keys(int key, int x, int y);
void init();

int main(int argc, char* argv[]) {
    glutInit(&argc, argv);
    // initialize the display mode
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA | GLUT_MULTISAMPLE);
    // for anti-aliasing
    glEnable(GLUT_MULTISAMPLE);
    // set the window position
    glutInitWindowPosition(100, 100);
    // set the window size
    glutInitWindowSize(WIN_WIDTH, WIN_HEIGHT);
    // now create the window
    glutCreateWindow(WIN_TITLE);

    // register the core functions
    glutDisplayFunc(render);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutSpecialFunc(special_keys);

    init();
    glutMainLoop();

    return 0;
}

void init() {
    glClearColor(164 / 255.0, 176 / 255.0, 190 / 255.0, 0.8);
    glFlush();
}
```

```
void reshape(int w, int h) {
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluOrtho2D(GRID_X_MIN, GRID_X_MAX, GRID_Y_MIN, GRID_Y_MAX);
}

void show_grid() {

    glBegin(GL_LINES);
    const GLubyte prestigeBlue[] = { 47.0, 53.0, 66.0, 1.0 };
    const GLubyte pureRed[] = { 255.0, 0.0, 0.0, 1.0 };

    for (float i = GRID_X_MIN; i <= GRID_X_MAX; i += 1.0) {
        glColor4ubv(prestigeBlue);
        if (i == 0)
            glColor4ubv(pureRed);
        glVertex2f(i, GRID_Y_MIN);
        glVertex2f(i, GRID_Y_MAX);
    }

    for (float i = GRID_Y_MIN; i <= GRID_Y_MAX; i += 1.0) {
        glColor4ubv(prestigeBlue);
        if (i == 0)
            glColor4ubv(pureRed);
        glVertex2f(GRID_X_MIN, i);
        glVertex2f(GRID_X_MAX, i);
    }

    glEnd();
}

void polygon() {
    glBegin(GL_POLYGON);

    const GLubyte jalapenoRed[] = { 183.0, 21.0, 64.0, 1.0 };
    const GLubyte darkSapphire[] = { 12.0, 36.0, 97.0, 1.0 };
    const GLubyte forestBlues[] = { 10.0, 61.0, 98.0, 1.0 };
    const GLubyte reefEncounter[] = { 7.0, 153.0, 146.0, 1.0 };

    glColor4ubv(jalapenoRed);
    glVertex2f(-1.0, 1.0);
    glColor4ubv(darkSapphire);
    glVertex2f(-1.0, -1.0);
    glColor4ubv(forestBlues);
    glVertex2f(1.0, -1.0);
    glColor4ubv(reefEncounter);
    glVertex2f(1.0, 1.0);
    glEnd();
}

void show_original() {
```

```
    glPushMatrix();
    polygon();
    glPopMatrix();
}

void show_transformed() {
    glPushMatrix();
    glScalef(5.0, 6.0, 1.0);
    glTranslatef(4.0, 8.0, 0.0);
    glRotatef(90, 0.0, 0.0, 1.0);
    polygon();
    glPopMatrix();
}

void show_text() {
    glPushMatrix();
    glColor3f(0.0, 0.0, 0.0);
    glTranslatef(30.0, 30.0, 1.0);
    glRasterPos2f(0.0, 0.0);
    const unsigned char *string = state_title[state];
    glutBitmapString(GLUT_BITMAP_HELVETICA_18, string);
    glPopMatrix();
}

void render() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    show_text();
    show_grid();

    switch (state) {
        //ORIGINAL
    case 0:
        glPushMatrix();
        polygon();
        glPopMatrix();
        break;
        // ROTATED
    case 1:
        glPushMatrix();
        glRotatef(90, 0.0, 0.0, 1.0);
        polygon();
        glPopMatrix();
        break;
        // TRANSLATED
    case 2:
        glPushMatrix();
        glTranslatef(4.0, 8.0, 0.0);
        polygon();
        glPopMatrix();
        break;
```

```
        // SCALED
    case 3:
        glPushMatrix();
        glScalef(5.0, 6.0, 1.0);
        polygon();
        glPopMatrix();
        break;
        // FINAL TRANSFORMED
    case 4:
        show_grid();
        show_original();
        show_transformed();
        break;
    }

    glFlush();

    glutSwapBuffers();

}

void keyboard(unsigned char c, int x, int y) {
    switch (c) {
    case 13:
    case 'q':
    case 'Q':
        exit(EXIT_SUCCESS);
        break;
    }
}

void special_keys(int key, int x, int y) {
    switch (key) {
    case GLUT_KEY_RIGHT:
        state = ((state + 1) % total_states + total_states) % total_states;
        glutPostRedisplay();
        break;
    case GLUT_KEY_LEFT:
        state = ((state - 1) % total_states + total_states) % total_states;
        glutPostRedisplay();
        break;
    }
}
```

1.3 Results with Screenshot and Discussion

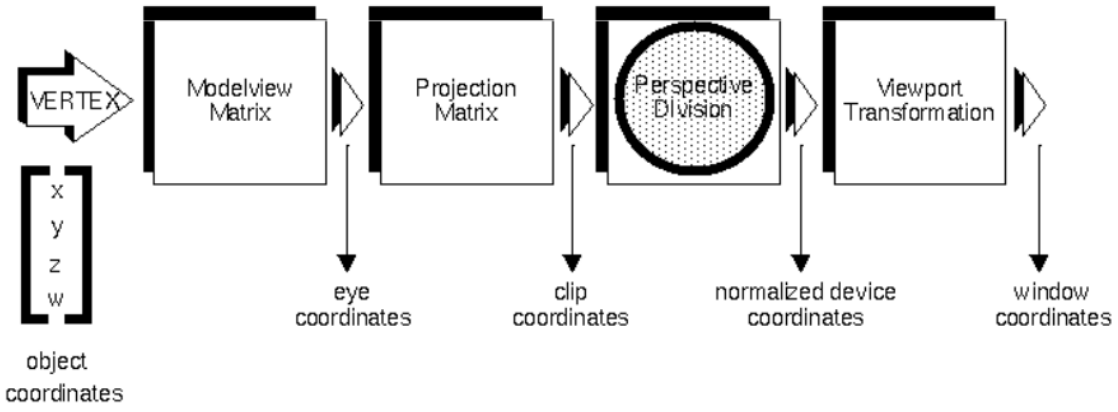

Figure 2 The OpenGL Pipeline

Each of the transformations above (Model View Matrix, Projection Matrix etc.) is maintained by OpenGL as part of the graphics state. (Current Transformation Matrix CTM)

`glLoadIdentity` sets the CTM to the identity matrix, for a "fresh start". When `glRotate` or similar command is issued, the appropriate transformation matrix is updated.

Note carefully that the rotation matrix doesn't overwrite the old CTM. It updates CTM by matrix multiplication. In fact, the CTM is so important that OpenGL can keep several of them in a stack. By popping the stack, you can recover an old and possibly still-useful CTM.

- `glPushMatrix();`
  - Save the state.
  - Push a copy of the CTM onto the stack.
  - The CTM itself is unchanged.
- `glPopMatrix();`
  - Restore the state, as it was at the last Push.
  - Overwrite the CTM with the matrix at the top of the stack.
- `glLoadIdentity();`
  - Overwrite the CTM with the identity matrix.

1. Original Square

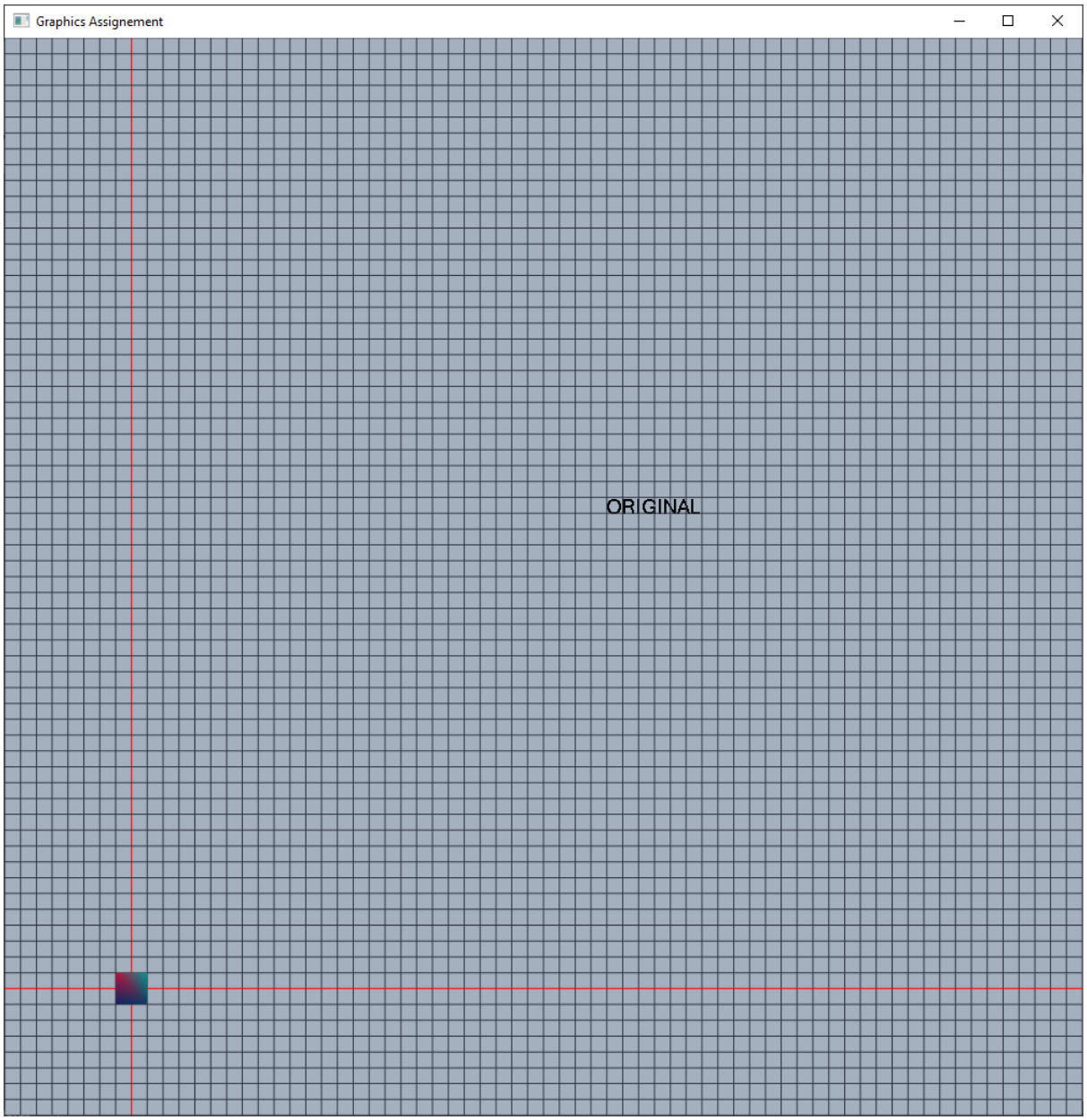

Figure 3 Original Polygon

$$\text{Polygon} = \begin{pmatrix} -1 & 1 & 1 & -1 \\ -1 & -1 & 1 & 1 \end{pmatrix}$$

2. Rotated Square

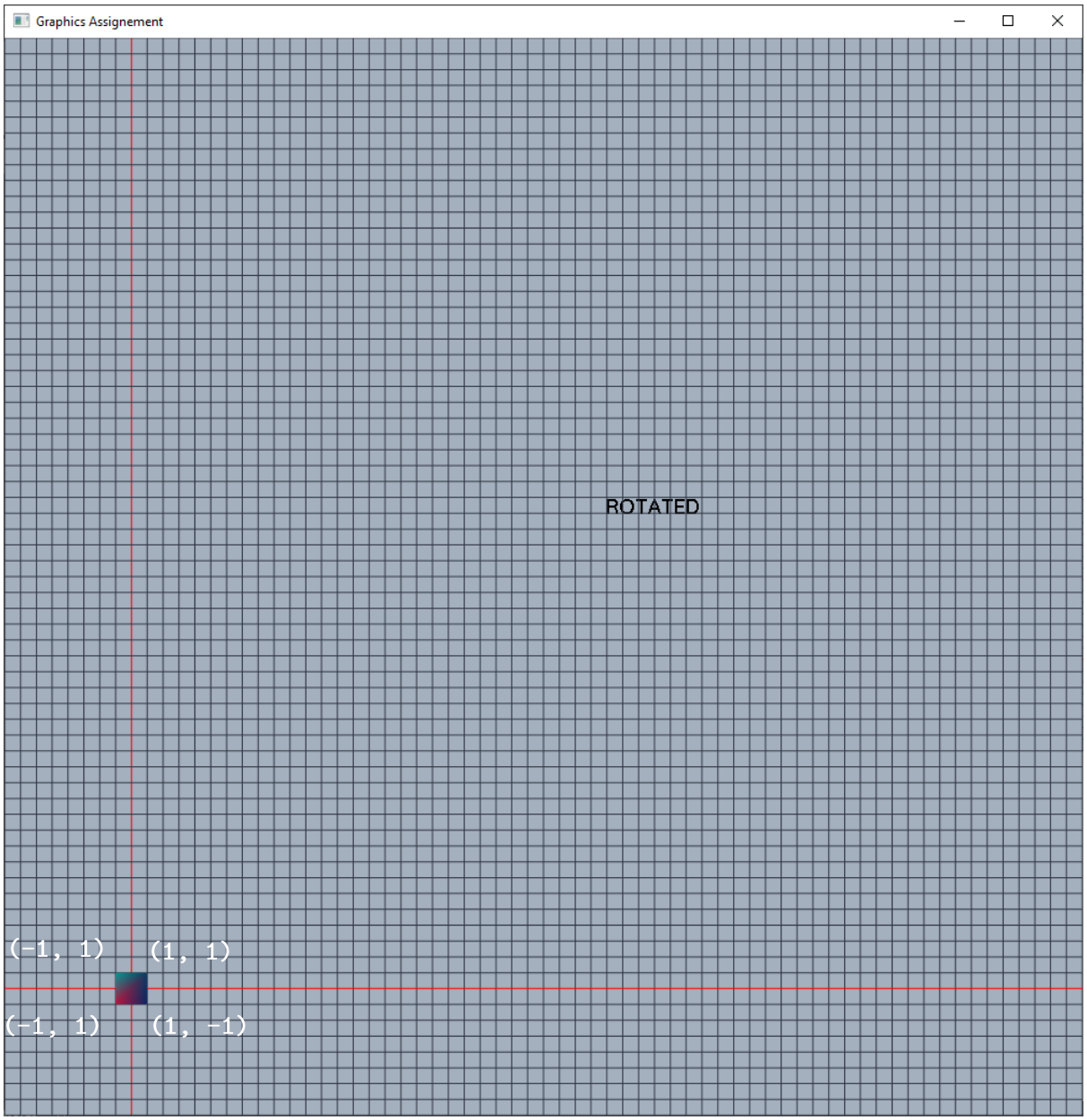

Figure 4 Rotated Polygon

$$\text{Polygon} = \begin{pmatrix} -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & 1 \end{pmatrix}$$

$$\text{Rot} = \begin{pmatrix} \cos 90 & -\sin 90 \\ \sin 90 & \cos 90 \end{pmatrix}$$

$$\text{Rot} \times \text{Polygon} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} -1 & 1 & 1 & -1 \\ -1 & -1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & -1 & -1 \\ -1 & 1 & 1 & -1 \end{pmatrix}$$

3. Translated Square

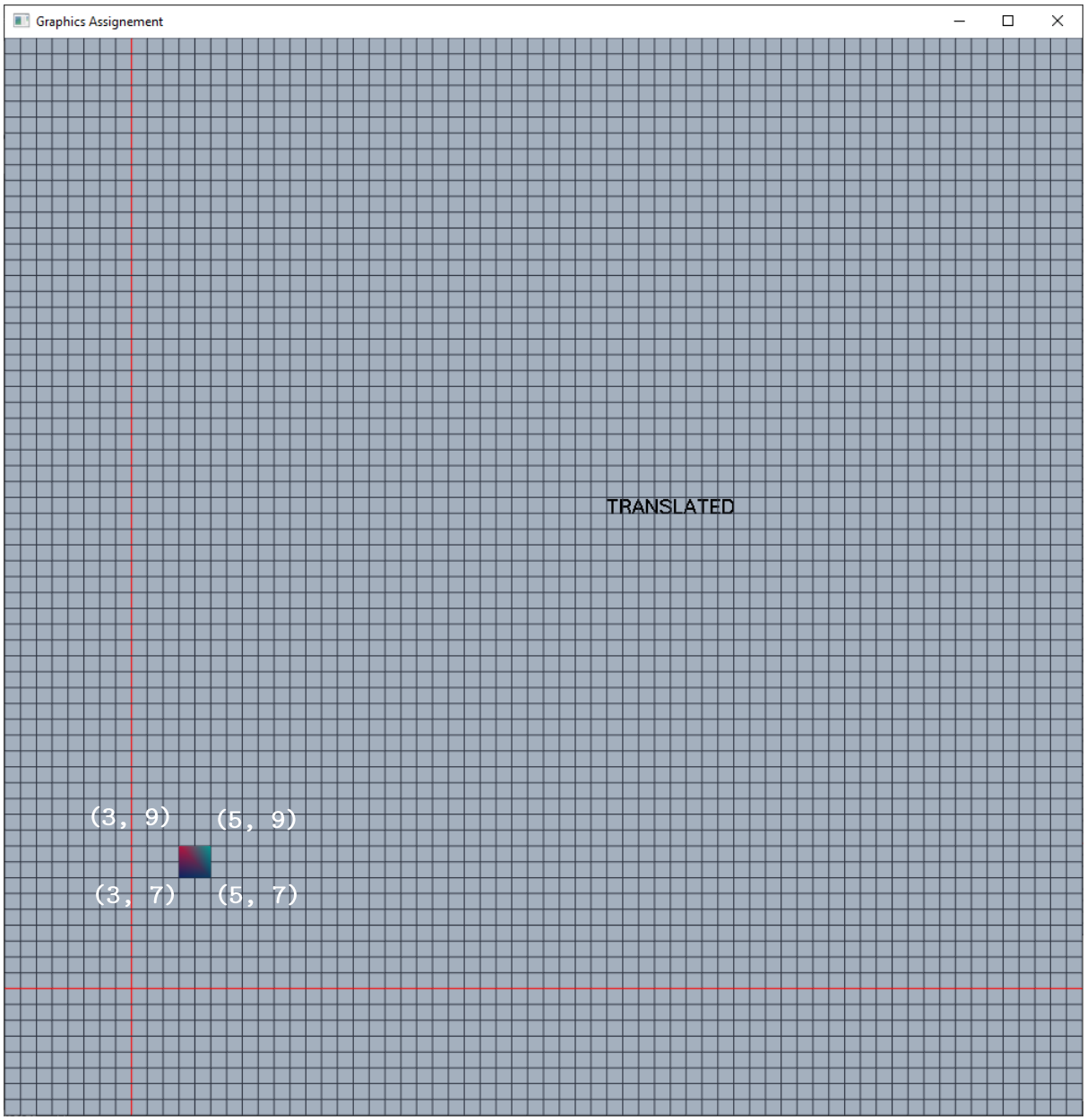

Figure 5 Translated Polygon

$$\text{Polygon} = \begin{pmatrix} -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & 1 \end{pmatrix}$$

$$\text{Translate} = \begin{pmatrix} t_x = 4 \\ t_y = 8 \end{pmatrix}$$

$$\text{Polygon} + \text{Translate} = \begin{pmatrix} -1+4 & 1+4 & 1+4 & -1+4 \\ -1+8 & -1+8 & 1+8 & 1+8 \end{pmatrix} = \begin{pmatrix} 3 & 5 & 5 & 3 \\ 7 & 7 & 9 & 9 \end{pmatrix}$$

4. Scaled Square

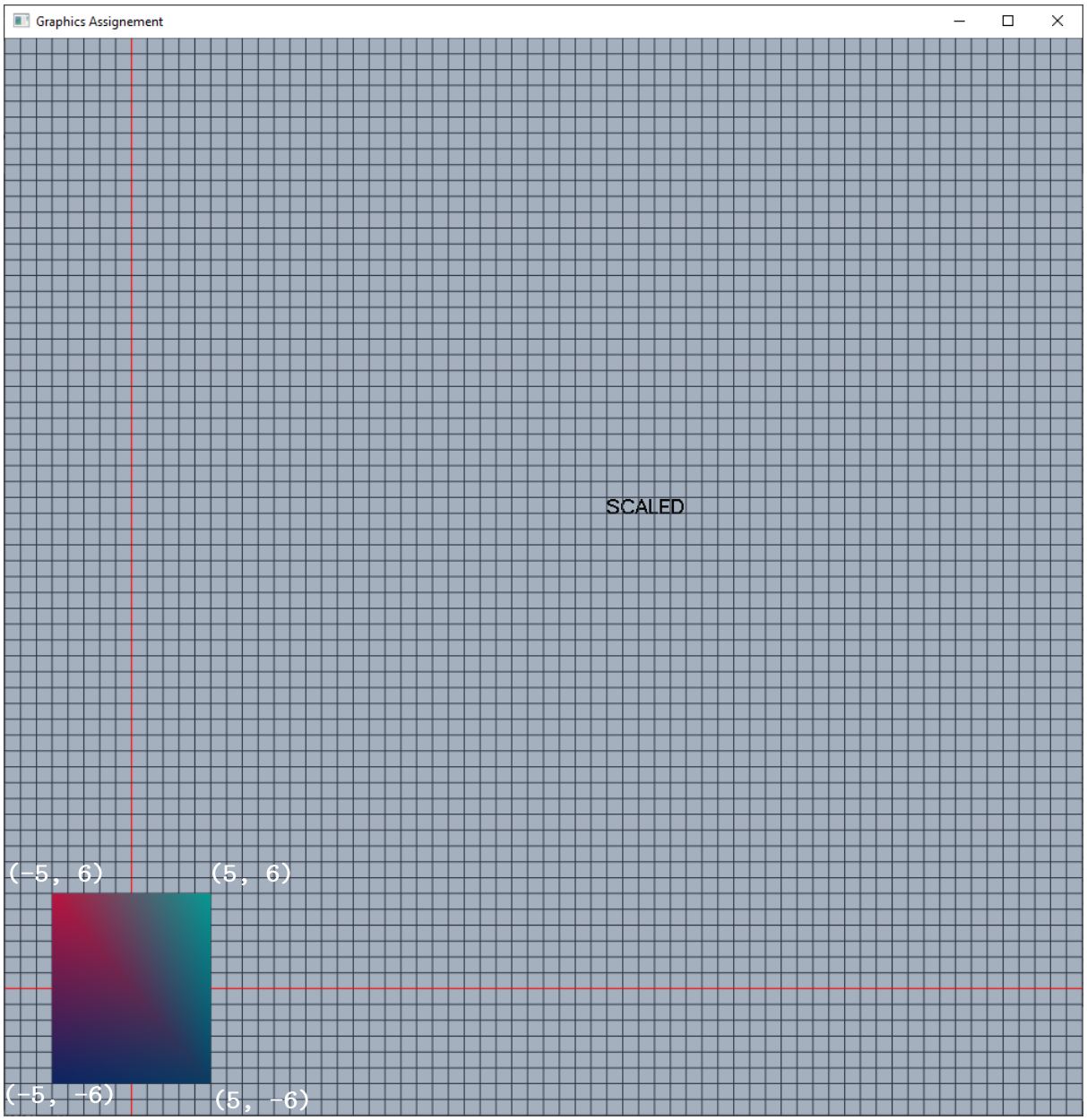

Figure 6 Scaled Polygon

$$\text{Polygon} = \begin{pmatrix} -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & 1 \end{pmatrix}$$

$$\text{Scale} = \begin{pmatrix} s_x = 5 & 0 \\ 0 & s_y = 6 \end{pmatrix}$$

$$\text{Scale} \times \text{Polygon} = \begin{pmatrix} 5 & 0 \\ 0 & 6 \end{pmatrix} \times \begin{pmatrix} -1 & 1 & 1 & -1 \\ -1 & -1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} -5 & 5 & 5 & -5 \\ -6 & -6 & 6 & 6 \end{pmatrix}$$

5. Final Transformed Polygon

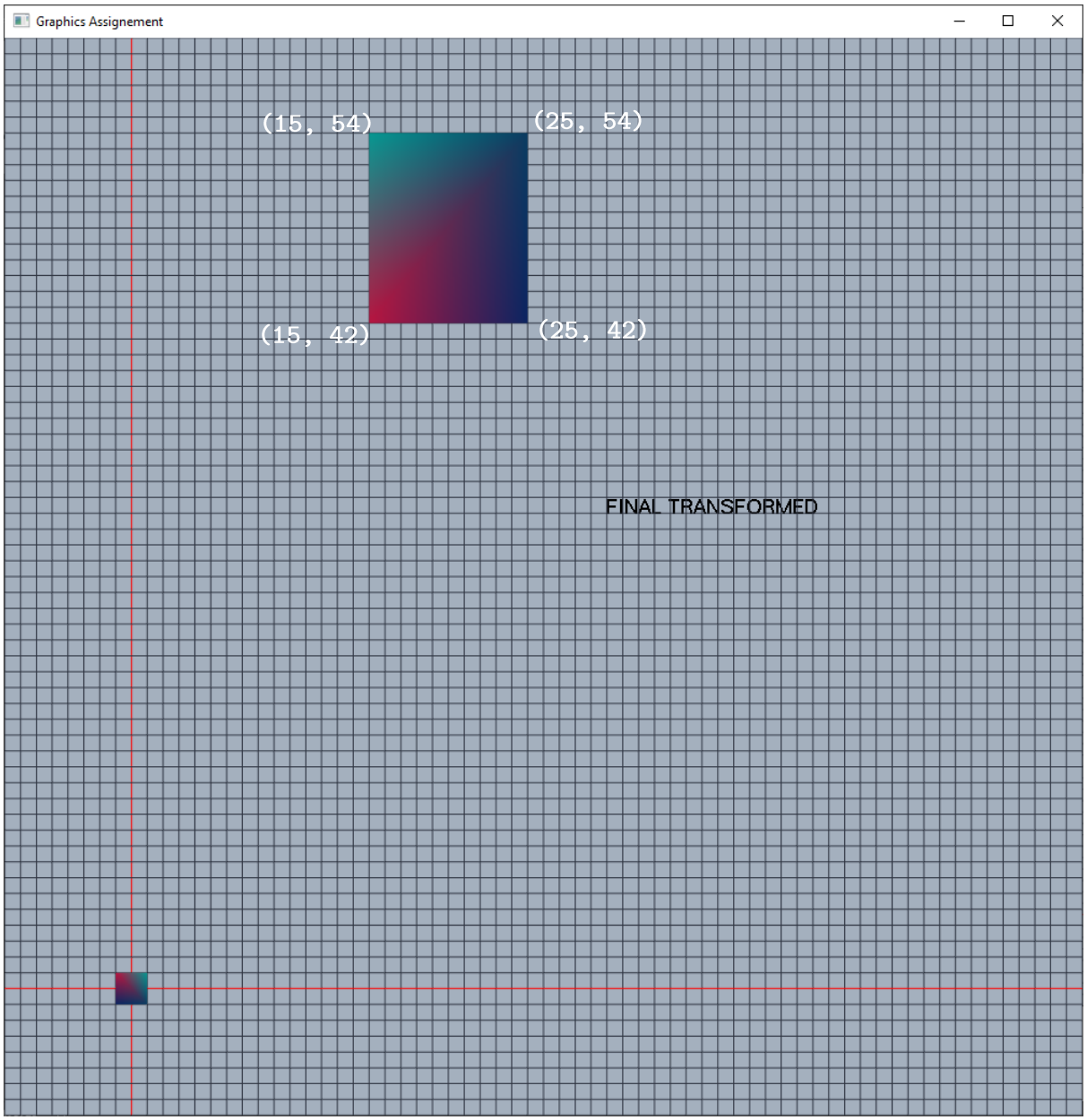

Figure 7 Final Transformed Polygon

All the transformed applied sequentially,

$$\text{Polygon} = \begin{pmatrix} -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & 1 \end{pmatrix}$$

Rotate:

$$\text{Rot} \times \text{Polygon} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} -1 & 1 & 1 & -1 \\ -1 & -1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & -1 & -1 \\ -1 & 1 & 1 & -1 \end{pmatrix}$$

Translate:

$$\text{Polygon} + \text{Translate} = \begin{pmatrix} 1+4 & 1+4 & -1+4 & -1+4 \\ -1+8 & 1+8 & 1+8 & -1+8 \end{pmatrix} = \begin{pmatrix} 5 & 5 & 3 & 3 \\ 7 & 9 & 9 & 7 \end{pmatrix}$$

Scale:

$$\text{Scale} \times \text{Polygon} = \begin{pmatrix} 5 & 0 \\ 0 & 6 \end{pmatrix} \times \begin{pmatrix} 5 & 5 & 3 & 3 \\ 7 & 9 & 9 & 7 \end{pmatrix} = \begin{pmatrix} 25 & 25 & 15 & 15 \\ 42 & 54 & 54 & 42 \end{pmatrix}$$

Final Polygon:

$$\begin{pmatrix} 25 & 25 & 15 & 15 \\ 42 & 54 & 54 & 42 \end{pmatrix}$$

Bibliography

1. https://www2.cs.duke.edu/courses/compsci344/spring15/classwork/07_pipeline/cox_04transformations.pdf
2. https://www.cs.cmu.edu/afs/cs/academic/class/15462-s09/www/lec/03/lec03a.pdf