|  |  |
| --- | --- |
| **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](#_Toc38200282)

[Contents iii](#_Toc38200283)

[List of Figures iv](#_Toc38200284)

[1 Question 1 5](#_Toc38200285)

[1.1 Introduction 5](#_Toc38200286)

[1.1.1 Translation 5](#_Toc38200287)

[1.1.2 Rotation 5](#_Toc38200288)

[1.1.3 Scaling 5](#_Toc38200289)

[1.1.4 Order of Transformations 6](#_Toc38200290)

[1.2 Implementation of Transformation 6](#_Toc38200291)

[1.3 Results with Screenshot and Discussion 11](#_Toc38200292)

[Bibliography 18](#_Toc38200293)

# List of Figures

[Figure 1 Order of Transformations 6](#_Toc38200294)

[Figure 2 The OpenGL Pipeline 11](#_Toc38200295)

[Figure 3 Original Polygon 12](#_Toc38200296)

[Figure 4 Rotated Polygon 13](#_Toc38200297)

[Figure 5 Translated Polygon 14](#_Toc38200298)

[Figure 6 Scaled Polygon 15](#_Toc38200299)

[Figure 7 Final Transformed Polygon 16](#_Toc38200300)

# Question 1

Solution to Question No. 1

## Introduction

### 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 );

### 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.

### 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.

### 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.

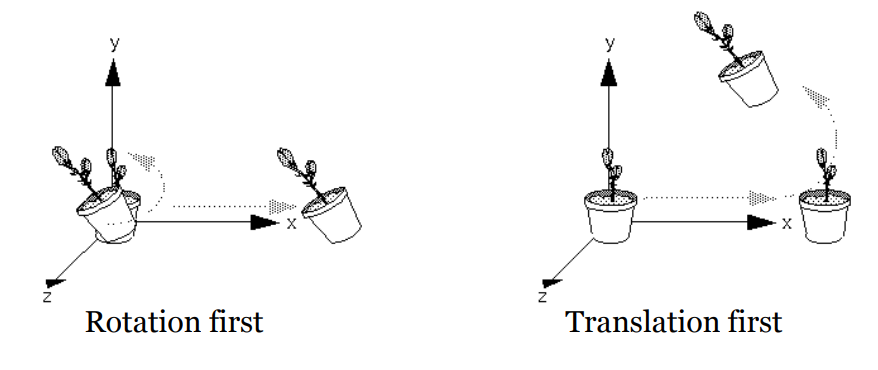


Figure Order of Transformations

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

## Implementation of Transformation

**main.cpp**

#include <iostream>

#include <string>

#include <GL/freeglut.h>

#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", "SCALED", "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;

    }

}

## Results with Screenshot and Discussion

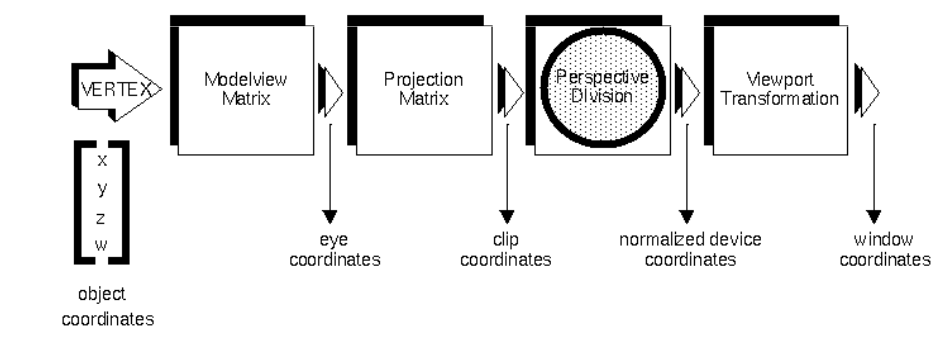


Figure The OpenGL Pipeline

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

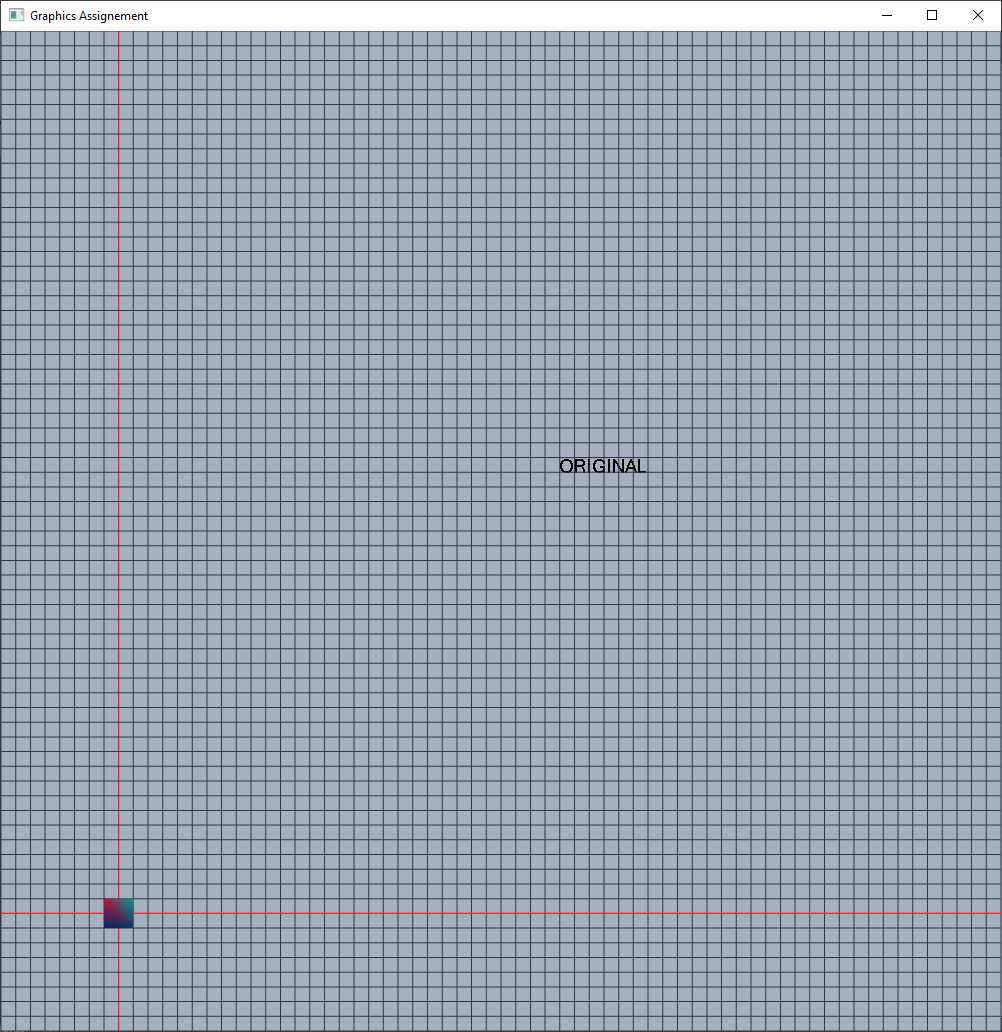
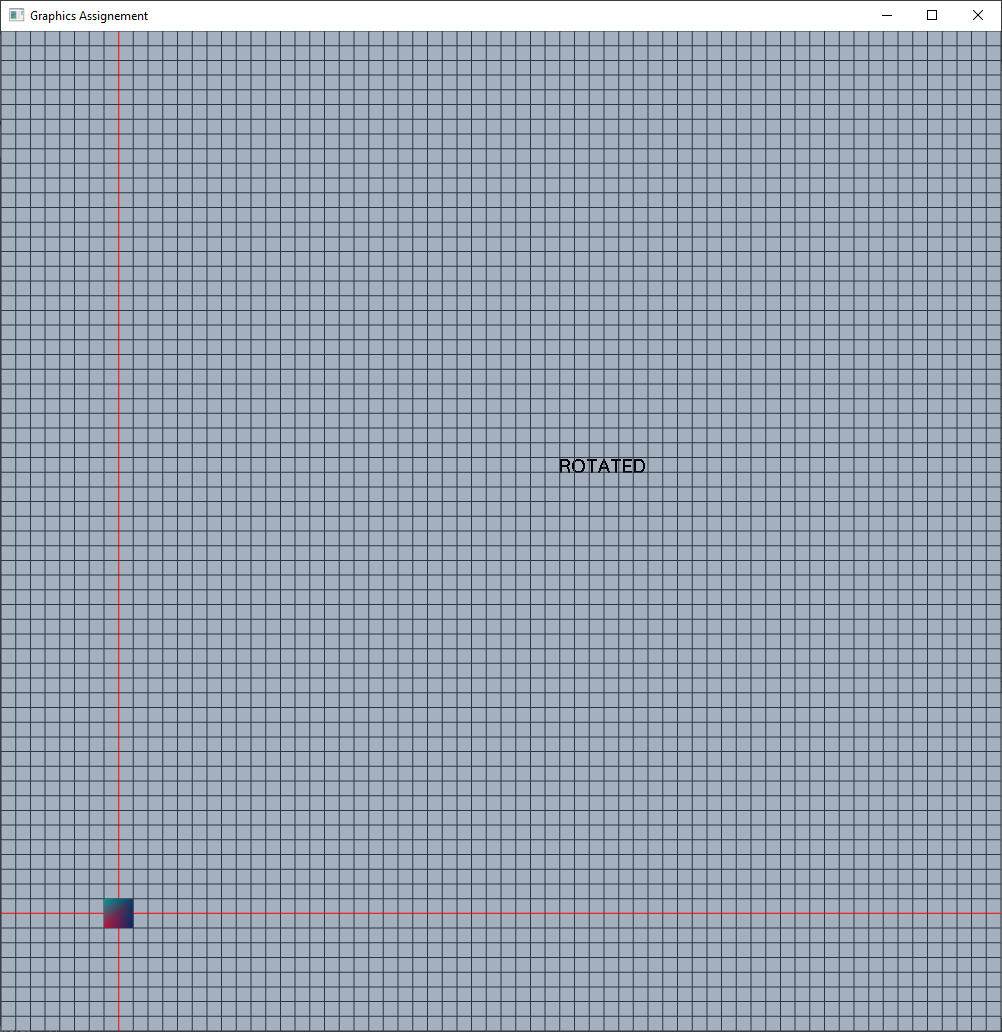


Figure 3 Original Polygon

1. Rotated Square



**(-1, 1)**

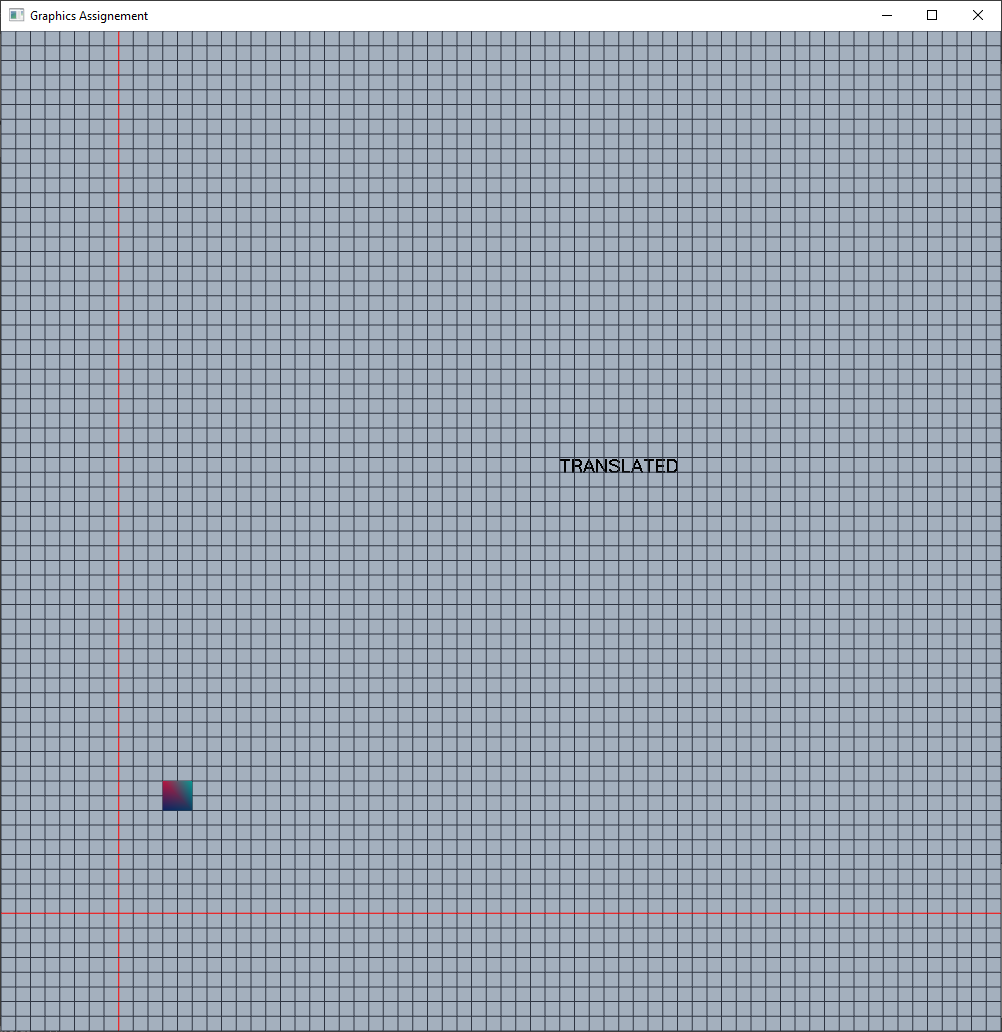
**(1, -1)**

**(-1, 1)**

**(1, 1)**

Figure 4 Rotated Polygon

1. Translated Square



**(3, 9)**

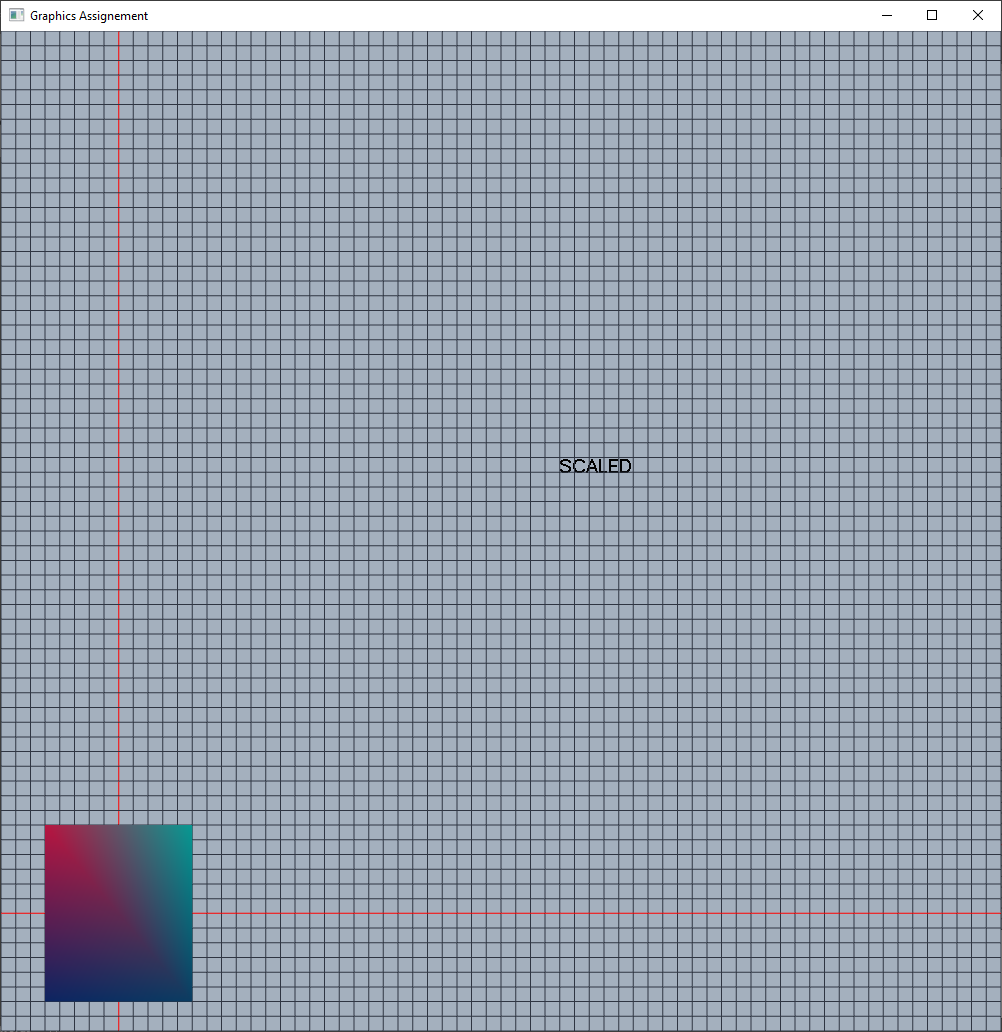
**(5, 9)**

**(5, 7)**

**(3, 7)**

Figure 5 Translated Polygon

1. Scaled Square



**(5, -6)**

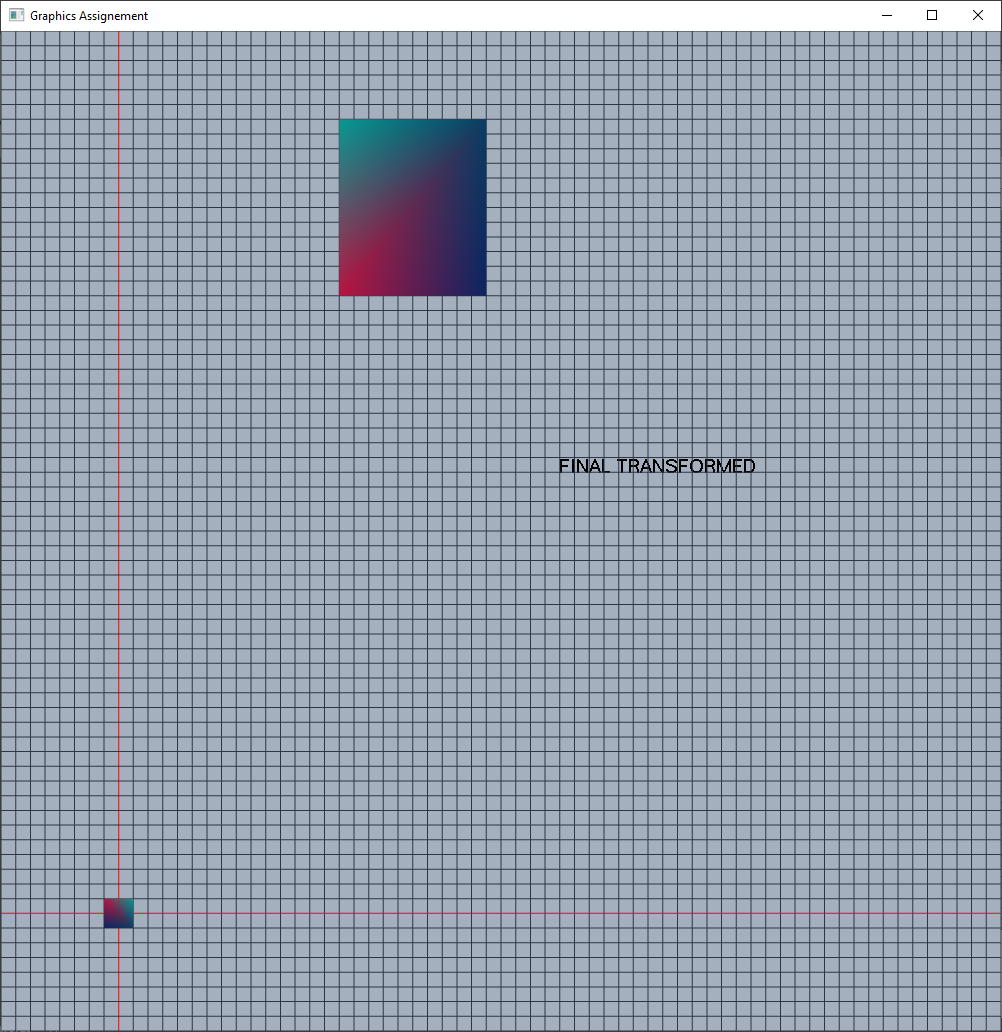
**(-5, -6)**

**(5, 6)**

**(-5, 6)**

Figure 6 Scaled Polygon

1. Final Transformed Polygon



**(25, 42)**

**(25, 54)**

**(15, 42)**

**(15, 54)**

Figure 7 Final Transformed Polygon

All the transformed applied sequentially,

Rotate:

Translate:

Scale:

Final Polygon:

# 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>