RAMAIAH UNIVERSITY OF APPLIED SCIENCES

# ASSIGNMENT

| | |
|---|---|
| **Course Code** | CSC309A |
| **Course Name** | COMPUTER GRAPHICS |
| **Programme** | B.TECH |
| **Department** | CSE |
| **Faculty** | FET |

| | |
|---|---|
| **Name of the Student** | AISHWARYA |
| **Reg. No** | 17ETCS002015 |
| **Semester/Year** | 6TH SEM/ 3RD YEAR |
| **Course Leader/s** | Deepak V. |

| Declaration Sheet | | | |
|---|---|---|---|
| Student Name | AISHWARYA | | |
| Reg. No | 17ETCS002015 | | |
| Programme | B.TECH | Semester/Year | 6TH SEM/ 3RD YEAR |
| Course Code | CSC309A | | |
| Course Title | COMPUTER GRAPHICS | | |
| Course Date | | to | |
| Course Leader | Deepak V. | | |

**Declaration**

| | | | |
|---|---|---|---|
| Signature of the Student | | Date | 19/04/2020 |
| Submission date stamp (by Examination & Assessment Section) | | | |

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

AISHWARYA

| Engineering and Technology | | | |
|---|---|---|---|
| **Ramaiah University of Applied Sciences** | | | |
| Department | Computer Science and Engineering | Programme | B. Tech. |
| Semester/Batch | 6th/2017 | | |
| Course Code | 19CSC309A | Course Title | Computer Graphics |
| Course Leader(s) | Deepak V. and Dr. Subarna Chatterjee | | |

| **Questions** | **Marking Scheme** | | **Marks** | | |
|---|---|---|---|---|---|
| | | | **Max Marks** | **First Examiner Marks** | **Moderator** |
| **1** | 1.1 | Introduction | 3 | | |
| | 1.2 | Implementation of transformation | 5 | | |
| | 1.3 | Results with screenshots and discussion | 2 | | |
| | | **Question 1 Max Marks** | **10** | | |
| | | **Total Assignment Marks** | **10** | | |

| **Course Marks Tabulation** | | | | |
|---|---|---|---|---|
| **Question** | **First Examiner** | **Remarks** | **Moderator** | **Remarks** |
| 1 | | | | |
| **Marks (Max 10 )** | | | | |

**Signature of First Examiner** **Signature of Moderator**

## 1.1 Introduction to the problem

According to the given problem statement 2D transformation is to be done including rotation, translation and scaling of a square with the help of a library file “glut.h “. Here, with the help of this library file objects can be created, also it allows a variety of operations on the object created.

The problem statement states the following:

First, Rotation of square with 90 degree as specified in the question. After that, Translate the resultant transformed square 4 units in X- direction and 8- units in Y direction. Next, scaling is done with scaling factor Sx=5 and Sy=6.Since, the coordinates for square is to be assumed, so here the coordinated of the square are P(1,1) , Q(1,-1), R(-1,-1), S(-1,1).

Open Graphics Library (OpenGL) is a cross-language (language independent), cross-platform (platform independent) API for rendering 2D and 3D Vector Graphics (use of polygons to represent image). Transformations play a very important role in manipulating objects on screen.
**There are three basic kinds of Transformations:**

**Translation**: A translation is applied to an object by repositioning it along a straight-line path from one coordinate location to another. We translate a two-dimensional point by adding translation distances, tx and ty to the original coordinate position (x, y) to move the point to a new position (x', y'). Polygons are translated by adding the translation vector to the coordinate position of each vertex and regenerating the polygon using the new set of vertex coordinates and the current attribute settings

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$ where tx and ty are translation coordinates

The OpenGL function is glTranslatef( tx, ty, tz )

**Rotation**: A two-dimensional rotation is applied to an object by repositioning it along a circular path in the xy plane. To generate a rotation, we specify a rotation angle 0 and the position (x, y) of the rotation point (or pivot point) about which the object is to be rotated, transformation can also be described as a rotation about a rotation axis that is perpendicular to the xy plane and passes through the pivot point. For anti-clockwise rotation

$$\begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ where, the angle of rotation is θ.

The above formula will rotate the point around the origin.
The OpenGL function is glRotatef (θ, x, y, z).

**Scaling**: A scaling transformation alters the size of an object. This operation can be carried out for polygons by multiplying the coordinate values (x, y) of each vertex by scaling factors s, and s, to produce the transformed coordinates (x', y'):

$$\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ $S_X$, $S_Y$ being scaling factors.

The OpenGL function is glScalef(float x, float y, float z)

For the above problem given, since various transformations such as rotation, translation and scaling is done on the square hence, composite transformation is used here to solve the problem.

1. Counter clockwise rotation 90 deg about origin

$$\boldsymbol{T1} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} cos90 & -sin90 & 0 \\ sin90 & cos90 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2. Translation of square 4 units along x-axis and 8 units along y-axis.

$$\boldsymbol{T2} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 4 \\ 0 & 1 & 8 \\ 0 & 0 & 1 \end{bmatrix}$$

3. Scaling of square where, Sx=5 and Sy=6

$$\boldsymbol{T3} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

4. Composite function

The order of the transformation (multiplication) in composite transformation is reversed.

$$\boldsymbol{T_c = T_3 * T_2 * T_1}$$

$$T_c = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 4 \\ 0 & 1 & 8 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_c = \begin{bmatrix} 5 & 0 & 20 \\ 0 & 6 & 48 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{T_c} = \begin{bmatrix} \mathbf{0} & \mathbf{-5} & \mathbf{20} \\ \mathbf{6} & \mathbf{0} & \mathbf{48} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

Now the new coordinates obtained for:

For P (1,1) $\begin{bmatrix} 0 & -5 & 20 \\ 6 & 0 & 48 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 15 \\ 54 \\ 1 \end{bmatrix}$

For Q (1,-1) $\begin{bmatrix} 0 & -5 & 20 \\ 6 & 0 & 48 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 25 \\ 54 \\ 1 \end{bmatrix}$

For R (-1,-1) $\begin{bmatrix} 0 & -5 & 20 \\ 6 & 0 & 48 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 25 \\ 42 \\ 1 \end{bmatrix}$

For S (-1,1) $\begin{bmatrix} 0 & -5 & 20 \\ 6 & 0 & 48 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 15 \\ 42 \\ 1 \end{bmatrix}$

Hence, the obtained coordinates of the square are:

P (15,54), Q (25,54), R (25,42), S (15,42)

Hence, the resultant square to be obtained along with the actual square is represented below

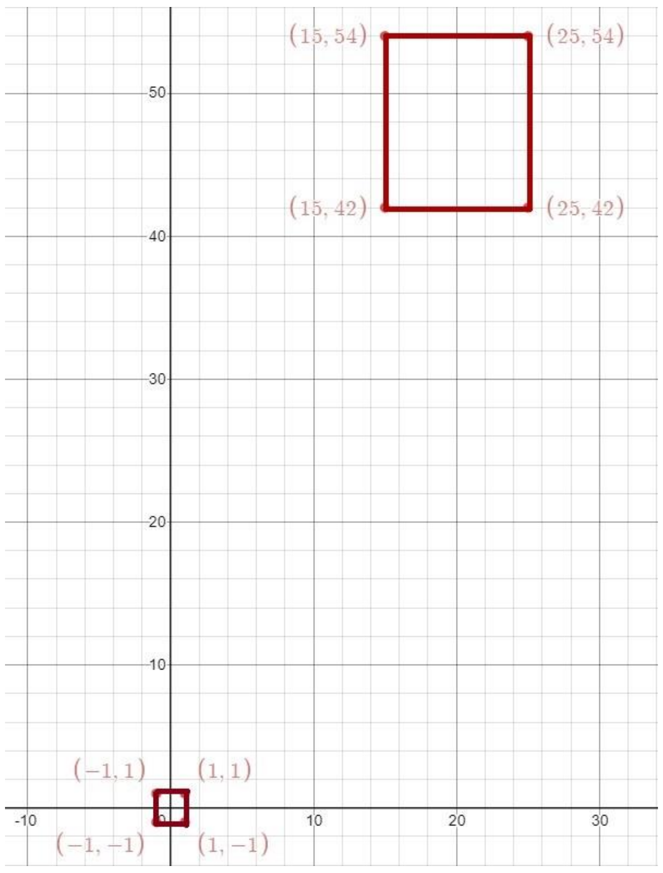
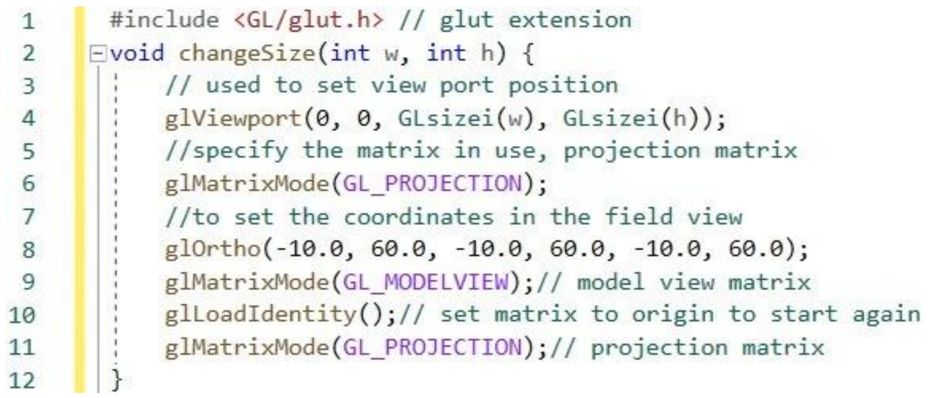


*Figure 1 Figure after manual calculation*

## 1.2 Implementation of transformation



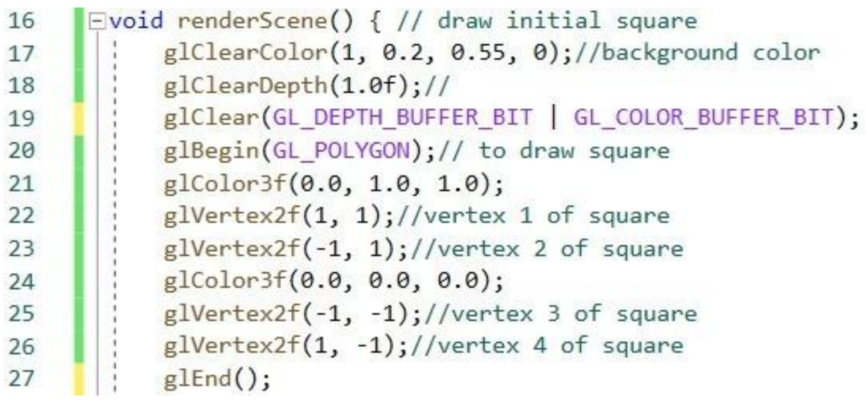
```
 #include <GL/glut.h> // glut extension
void changeSize(int w, int h) {
    // used to set view port position
    glViewport(0, 0, GLsizei(w), GLsizei(h));
    //specify the matrix in use, projection matrix
    glMatrixMode(GL_PROJECTION);
    //to set the coordinates in the field view
    glOrtho(-10.0, 60.0, -10.0, 60.0, -10.0, 60.0);
    glMatrixMode(GL_MODELVIEW);// model view matrix
    glLoadIdentity();// set matrix to origin to start again
    glMatrixMode(GL_PROJECTION);// projection matrix
}
```

*Fig.1 code for transformation*

The above figure shows the glut extension added in the program. It performs system-level I/O with the host operating system, functons performed inclides window control, defining of the window and to monitor keyboard and mouse input.

The change size function is an argument provideddefined in the main function. This function mainly focuses on the characteristics as well as functionalities of the output screen. Here, glViewport is used to set position of the view port whereas the glMatrixMode is used to show the matrix currently in use os used by **GL_MODELVIEW** for modelview and **GL_PROJRCTION** for projection.

Before the viewing transformation can be specified, the current matrix is set to the identity matrix with **glLoadIdentity()**. This step is necessary since most of the transformation commands multiply the current matrix by the specified matrix and then set the result to be the current matrix.

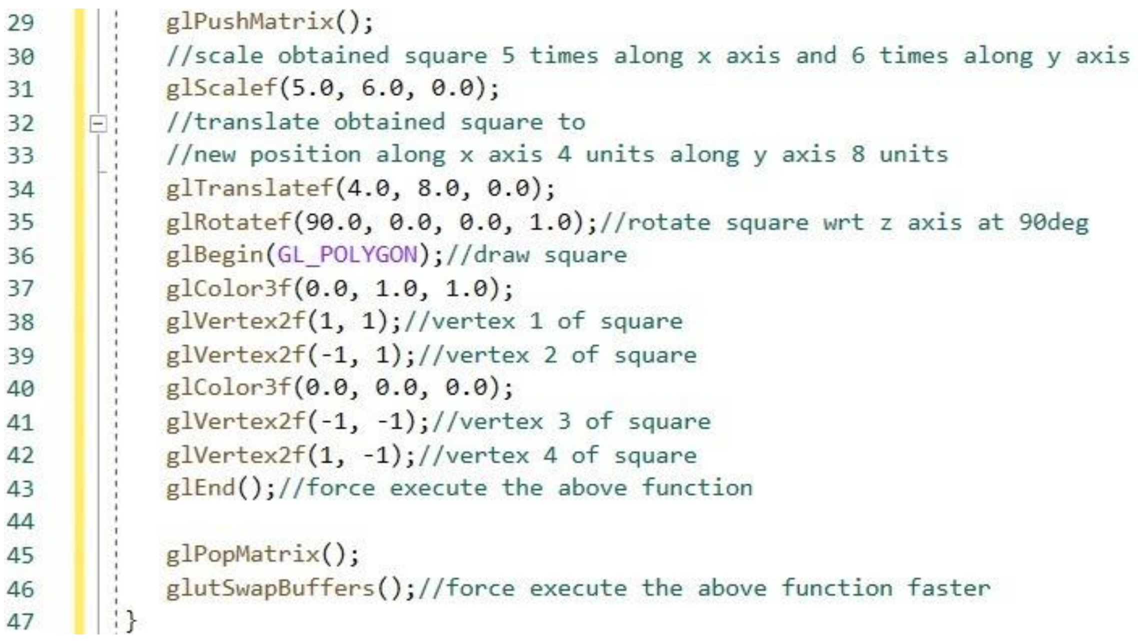


```
void renderScene() { // draw initial square
    glClearColor(1, 0.2, 0.55, 0);//background color
    glClearDepth(1.0f);//
    glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);// to draw square
    glColor3f(0.0, 1.0, 1.0);
    glVertex2f(1, 1);//vertex 1 of square
    glVertex2f(-1, 1);//vertex 2 of square
    glColor3f(0.0, 0.0, 0.0);
    glVertex2f(-1, -1);//vertex 3 of square
    glVertex2f(1, -1);//vertex 4 of square
    glEnd();
```

*Fig.2 code for transformation*

The render function is for the initial square to be drawn. Here, “glClearColor” function is called which takes four arguments as RGBA to set the background colour of the window to pink. Colour of the upcoming is specified to gradient of cyan and black using “glColor3f” function which also takes RGB as argument. The glBegin() is to start the polygon that is to draw the square. The vertices are defined by glVertex2f, the vertices are (1,1), (-1,1), (-1, -1) and (1, -1).

Here, for the representation of the square developed and transformed, gradient colour is used. In computer graphics, gradient colour specifies the range of position dependent colour used to represent a (square) object. The colours used here for the representation of the square is “cyan” and “black”.



```
    glPushMatrix();
    //scale obtained square 5 times along x axis and 6 times along y axis
    glScalef(5.0, 6.0, 0.0);
    //translate obtained square to
    //new position along x axis 4 units along y axis 8 units
    glTranslatef(4.0, 8.0, 0.0);
    glRotatef(90.0, 0.0, 0.0, 1.0);//rotate square wrt z axis at 90deg
    glBegin(GL_POLYGON);//draw square
    glColor3f(0.0, 1.0, 1.0);
    glVertex2f(1, 1);//vertex 1 of square
    glVertex2f(-1, 1);//vertex 2 of square
    glColor3f(0.0, 0.0, 0.0);
    glVertex2f(-1, -1);//vertex 3 of square
    glVertex2f(1, -1);//vertex 4 of square
    glEnd();//force execute the above function

    glPopMatrix();
    glutSwapBuffers();//force execute the above function faster
}
```

*Fig.3 code for transformation*

The above snippet, represents the transformation of the square, glScalef for the scaling, glTranslatef for the translation of square and glRotatef for the rotation, of the square. The square is to be rotated $90^o$

with respect to z axis and then translated 4 units along x-axis and 8 units along y-axis, the obtained square is then scaled 5-times along the x-axis and 6-times along the y-axis.

Three OpenGL routines for modelling transformations are glTranslatef(), glRotatef(), and glScalef().

**glTranslatef(TYPE x, TYPE y, TYPE z);**

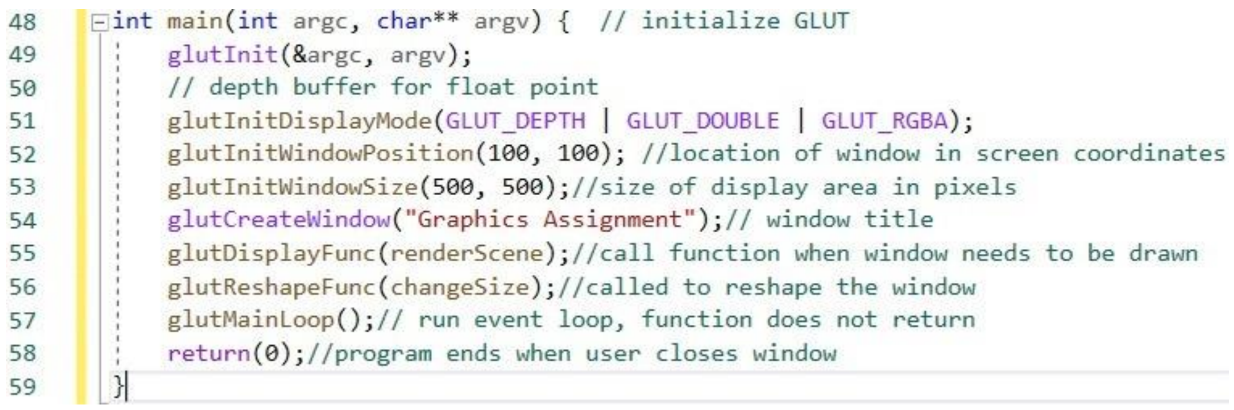
Multiplies the current matrix by a matrix that moves (translates) an object by the given x, y, and z values (or moves the local coordinate system by the same amounts).

**glRotatef(TYPE angle, TYPE x, TYPE y, TYPE z);**

Multiplies the current matrix by a matrix that rotates an object (or the local coordinate system) in a counter clockwise direction about the ray from the origin through the point (x, y, z). The angle parameter specifies the angle of rotation in degrees.

**glScalef(TYPE x, TYPE y, TYPE z);**

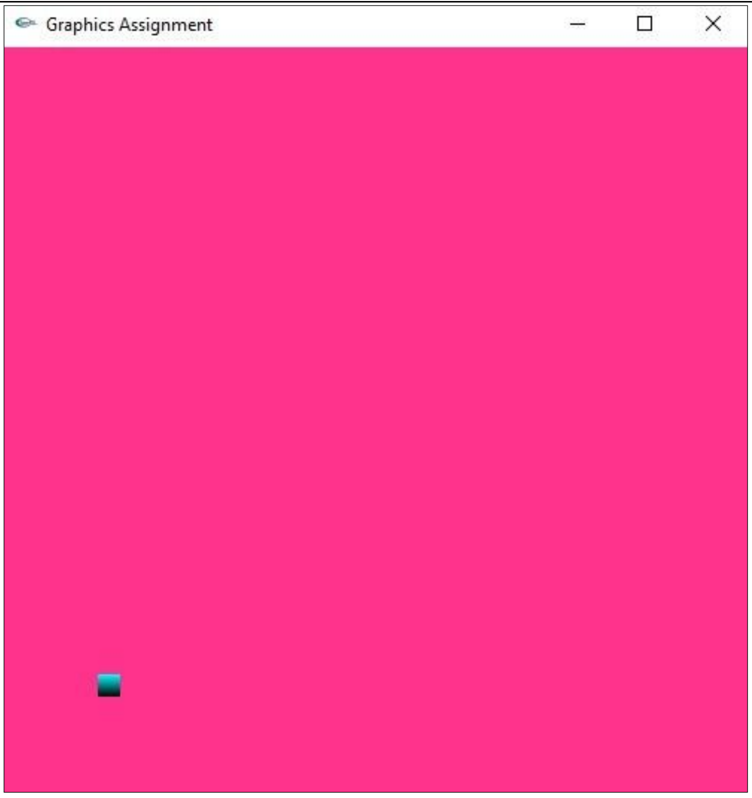
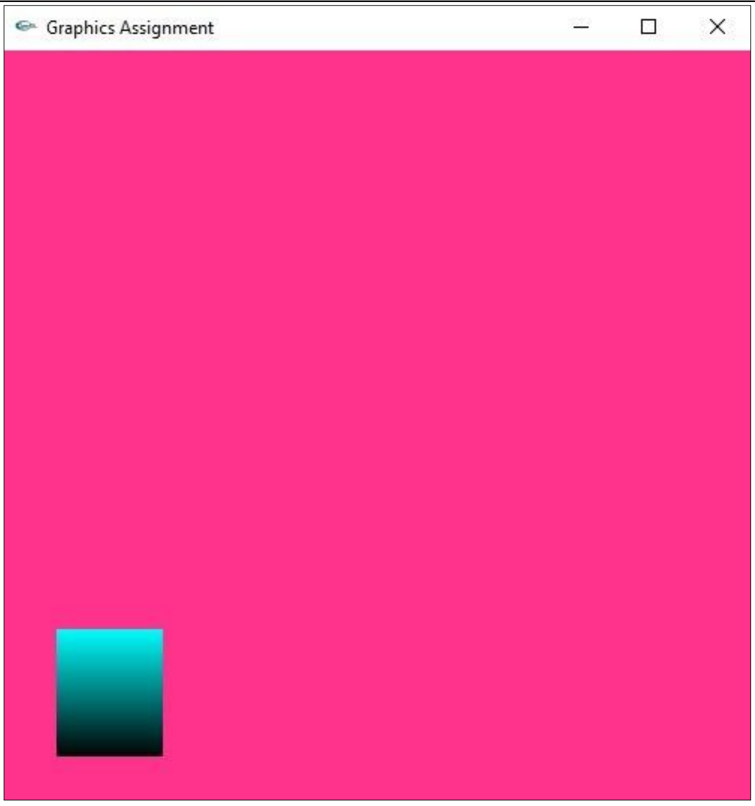
Multiplies the current matrix by a matrix that stretches, shrinks, or reflects an object along the axes. Each x, y, and z coordinate of every point in the object is multiplied by the corresponding argument x, y, or z. With the local coordinate system approach, the local coordinate axes are stretched, shrunk, or reflected by the x, y, and z factors, and the associated object is transformed with them.

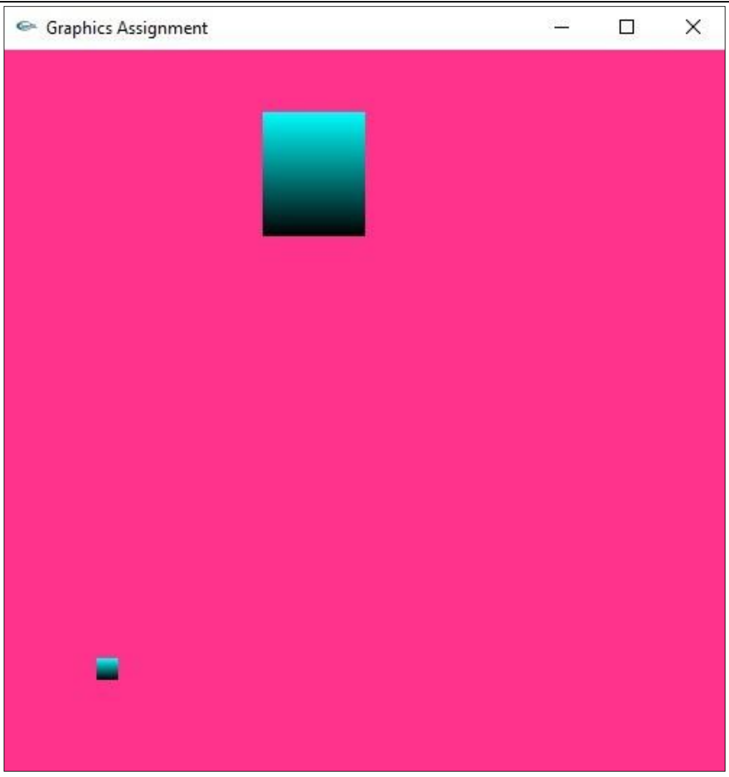
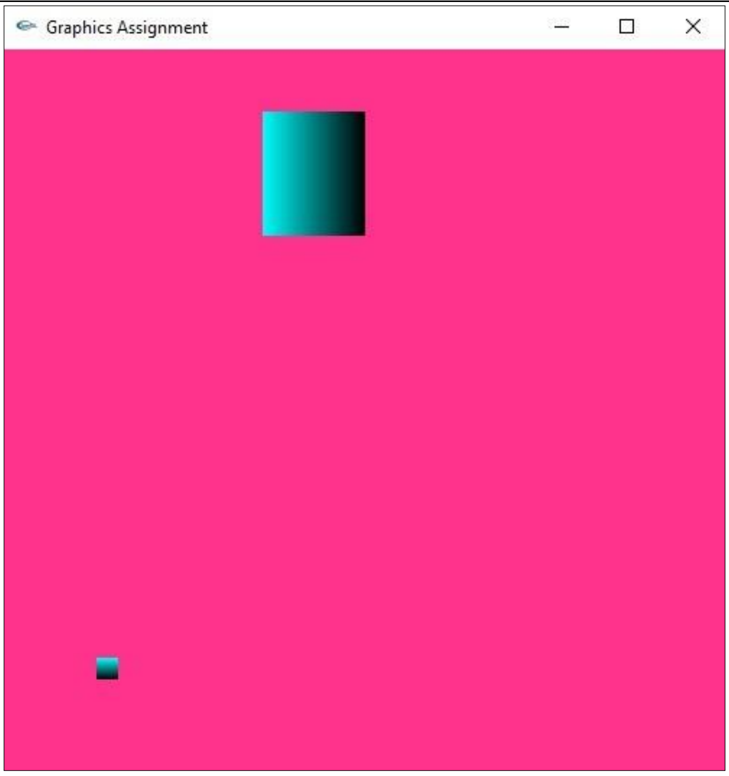


```
int main(int argc, char** argv) {  // initialize GLUT
    glutInit(&argc, argv);
    // depth buffer for float point
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowPosition(100, 100); //location of window in screen coordinates
    glutInitWindowSize(500, 500);//size of display area in pixels
    glutCreateWindow("Graphics Assignment");// window title
    glutDisplayFunc(renderScene);//call function when window needs to be drawn
    glutReshapeFunc(changeSize);//called to reshape the window
    glutMainLoop();// run event loop, function does not return
    return(0);//program ends when user closes window
}
```

The above snippet is of the main function, here GLUT library is initialized and gradient colour is used, it is set as an argument to **glutInitDisplayMode**, **glutInitWindowPosition** represents the location of window in the screen coordinates, and **glutInitWindowSize** represents the size of display area in pixels. The main function calls the function **renderScene** when window needs to be drawn and **changeSize** to reshape the window.

### 1.3 Results with screenshots and discussion

| OUTPUT | DISCRIPTION |
|---|---|
|  Graphics Assignment | In the snippet shown here, initial square can be seen that is with the coordinates (1, 1), (-1, 1), (-1, -1) and (1, -1), the square above has no transformation performed on it. |
|  Graphics Assignment | The snippet represents the square as discussed above, which is scaled with Sx value 5 and Sy value 6, which actually zooms the square. Here, according to the given problem the values provided for scaling the square are different, hence the resultant square is of bigger size. |

| | |
|---|---|
|  | The snippet represents the initial square taken with the coordinated (1, 1), (-1, 1), (-1, -1) and (1, -1) and along with is the square translated 4 units along the x-axis and 8 units along the y-axis.<br>The formula for translation is:<br>**Along x-axis**: X' = X + tx , where tx is the units translated here tx provided is 4 units**.**<br>**Along x-axis**: Y' = Y + ty , where ty is the units translated here ty provided is 8 units**.** |
|  | The snippet represents the initial square as well as the square obtained after the transformation carried out, the final square obtain after the translation, that is the square obtained after is rotated about the origin with an of angle 90 degree, the rotation is done along z axis as the square is a 2-dimensional figure placed in XY- plane which makes 2 possible rotations for 2D figures which is clockwise and anti-clockwise. Here, anti-clockwise rotation is used as the type of rotation is not specified in the question. |

From the above figures, as it can be seen that initially before rotation of 90 degrees along the z-axis the black was the base, after the rotation of 90degree in counter-clockwise direction, the base color changes accordingly.
Hence, the obtained square is transformed that is it is rotated 90 degree, translated 4 units along x-axis and 8 units along y axis also scaling is done for Sx= 5 and Sy=6.
The above result obtained through the implementation can be verified by the manual calculations done above in the introduction part.