Batch: D-2	**Roll No.:** 16010122151
Experiment No. 06	

TITLE: Write a program to perform 2D and 3D transformation

AIM:

Write a program to perform 2D and 3D transformation

a. Translation

b. Scaling

c. Rotation

d. Shear

e. Reflection

Expected OUTCOME of Experiment:

Implement Fill area Primitives, 2D Geometric Transformations and 2D viewing

Implement Clipping,3D Geometric Transformations and 3D viewing

Books/ Journals/ Websites referred:

https://cse18-iiith.vlabs.ac.in/exp/transformations-rotation/
https://cse18-iiith.vlabs.ac.in/exp/transformations-scaling/
https://cse18-iiith.vlabs.ac.in/exp/transformations-translation/
https://cse18-iiith.vlabs.ac.in/exp/2d-demo/
https://cse18-iiith.vlabs.ac.in/exp/3d-articulated-arm/

Algorithm/ Pseudo code for each process:

2-Dimension Translation

- Get points of object (here, a triangle) as (x_1,y_1), (x_2,y_2) … (x_n,y_n)
- Get translation of object in each direction as transX and transY
- Calculate transformation matrix for translation in the following manner:

$$T = \begin{bmatrix} 1 & 0 & transX \\ 0 & 1 & transY \\ 0 & 0 & 1 \end{bmatrix}$$

- Make matrix for points of the object

$$P = \begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

- Calculate product of transformation matrix and points matrix

$$P' = \begin{bmatrix} 1 & 0 & transX \\ 0 & 1 & transY \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

- Draw points on screen

2-Dimension Scaling

- Get points of object (here, a triangle) as (x_1,y_1), (x_2,y_2) … (x_n,y_n)
- Get scaling of object in each direction as scaleX and scaleY
- Calculate transformation matrix for scaling in the following manner:

$$T = \begin{bmatrix} scaleX & 0 & 0 \\ 0 & scaleY & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Make matrix for points of the object

$$P = \begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

- Calculate product of transformation matrix and points matrix

$$P' = \begin{bmatrix} scaleX & 0 & 0 \\ 0 & scaleY & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

- Draw points on screen

2-Dimension Rotation

- Get points of object (here, a triangle) as (x_1,y_1), (x_2,y_2) … (x_n,y_n)
- Get rotation of object in degrees as theta θ
- Calculate transformation matrix for rotation in the following manner:

$$P' = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Make matrix for points of the object

$$P = \begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

- Calculate product of transformation matrix and points matrix

$$P' = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

- Draw points on screen

3-D Transformation

1. Translation:

If P is a point having co-ordinates in three directions (x, y, z) is translated, then after translation its coordinates will be (x1 y1 z1) after translation. Tx, Ty, Tz are translation vectors in x, y, and z directions respectively.

x1 = x + Tx
y1 = y + Ty
z1 = z + Tz

Matrix for Translation

$$\begin{Bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

2. Rotation:

Matrix for representing three-dimensional rotations about the Z axis,

$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Matrix for representing three-dimensional rotations about the X axis,

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Matrix for representing three-dimensional rotations about the Y axis,

$$\begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3. Scaling

Scaling Factors –

Sx = Scaling factor in x-direction

Sy = Scaling factor in y-direction

Sz = Scaling factor in z-direction

Matrix -

$$\begin{Bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

4. Shear:

Matrix for shear

$$\begin{pmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

5. Reflection

Matrix for Reflection relative to XY plane,

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Matrix for Reflection relative to YZ plane,

$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Matrix for Reflection relative to

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Implementation details:

Translation:

```
#include <GL/glut.h>
#include <iostream>
#include <vector>
#include <Eigen/Dense> // For matrix operations

using namespace Eigen;
using namespace std;

const int width = 1000;
const int height = 1000;

vector<Vector3f> points;
vector<Vector3f> newPoints;

void iterate() {
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, width, 0.0, height, 0.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void translate() {
    glColor3f(0.0, 1.0, 1.0); // Cyan
    glBegin(GL_LINE_LOOP);
    for (const auto& point : points) {
        glVertex2f(point.x(), point.y());
    }
    glEnd();

    glColor3f(1.0, 1.0, 0.0); // Yellow
```

```
    glBegin(GL_LINE_LOOP);
    for (const auto& point : newPoints) {
        glVertex2f(point.x(), point.y());
    }
    glEnd();
}

void showScreen() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    iterate();
    translate();
    glutSwapBuffers();
}

int main(int argc, char** argv) {
    for (int i = 0; i < 3; ++i) {
        float pointX, pointY;
        cout << "Enter point " << i + 1 << " of triangle (x y): ";
        cin >> pointX >> pointY;
        points.push_back(Vector3f(pointX, pointY, 1));
    }

    float transX, transY;
    cout << "Enter translation in x-direction: ";
    cin >> transX;
    cout << "Enter translation in y-direction: ";
    cin >> transY;

    Matrix3f translateMat;
    translateMat << 1, 0, transX,
                    0, 1, transY,
                    0, 0, 1;

    newPoints.resize(points.size());
    for (size_t i = 0; i < points.size(); ++i) {
        newPoints[i] = translateMat * points[i];
    }

    cout << "\nOld position: cyan" << endl;
    for (const auto& point : points) {
        cout << point.transpose() << endl;
    }

    cout << "\nTranslation Matrix: " << endl;
    cout << translateMat << endl;
```

```
    cout << "\nNew Position: yellow" << endl;
    for (const auto& point : newPoints) {
        cout << point.transpose() << endl;
    }

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA);
    glutInitWindowSize(width, height);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Translation");
    glutDisplayFunc(showScreen);
    glutIdleFunc(showScreen);
    glutMainLoop();

    return 0;
}
```

Rotation:

```
#include <GL/glut.h>
#include <iostream>
#include <vector>
#include <cmath>
#include <Eigen/Dense>

using namespace std;
using namespace Eigen;

const int width = 1000;
const int height = 1000;

vector<Vector3f> points;
vector<Vector3f> newPoints;

void iterate() {
    glViewport(0, 0, 2 * width, 2 * height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-width, width, -height, height, 0.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void draw() {
    glColor3f(0.0, 1.0, 1.0); // Cyan
```

```
    glLineWidth(3.0);
    glBegin(GL_LINE_LOOP);
    for (const auto& point : points) {
        glVertex2f(point.x(), point.y());
    }
    glEnd();

    glColor3f(1.0, 1.0, 0.0); // Yellow
    glBegin(GL_LINE_LOOP);
    for (const auto& point : newPoints) {
        glVertex2f(point.x(), point.y());
    }
    glEnd();
}

void showScreen() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    iterate();
    draw();
    glutSwapBuffers();
}

int main(int argc, char** argv) {
    for (int i = 0; i < 3; ++i) {
        float pointX, pointY;
        cout << "  Enter point " << i + 1 << " of triangle (x y): ";
        cin >> pointX >> pointY;
        points.push_back(Vector3f(pointX, pointY, 1));
    }

    float rotAngle;
    cout << "  Enter rotation angle in degrees: ";
    cin >> rotAngle;

    float rotAngleRad = rotAngle * (M_PI / 180.0);

    Matrix3f rotMat;
    rotMat << cos(rotAngleRad), -sin(rotAngleRad), 0,
              sin(rotAngleRad), cos(rotAngleRad), 0,
              0, 0, 1;

    newPoints.resize(points.size());
    for (size_t i = 0; i < points.size(); ++i) {
        newPoints[i] = rotMat * points[i];
    }
```

```
    cout << "\n  Old position: cyan" << endl;
    for (const auto& point : points) {
        cout << point.transpose() << endl;
    }

    cout << "\n  Rotation Matrix: " << endl;
    cout << rotMat << endl;

    cout << "\n  New Position: yellow" << endl;
    for (const auto& point : newPoints) {
        cout << point.transpose() << endl;
    }

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA);
    glutInitWindowSize(2 * width, 2 * height);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Rotation");
    glutDisplayFunc(showScreen);
    glutIdleFunc(showScreen);
    glutMainLoop();

    return 0;
}
```

Scaling:

```
#include <GL/glut.h>
#include <iostream>
#include <vector>
#include <Eigen/Dense>

using namespace std;
using namespace Eigen;

const int width = 1000;
const int height = 1000;

vector<Vector3f> points;
vector<Vector3f> newPoints;

void iterate() {
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
```

```
    glOrtho(0.0, width, 0.0, height, 0.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void draw() {
    glColor3f(0.0, 1.0, 1.0); // Cyan
    glBegin(GL_LINE_LOOP);
    for (const auto& point : points) {
        glVertex2f(point.x(), point.y());
    }
    glEnd();

    glColor3f(1.0, 1.0, 0.0); // Yellow
    glBegin(GL_LINE_LOOP);
    for (const auto& point : newPoints) {
        glVertex2f(point.x(), point.y());
    }
    glEnd();
}

void showScreen() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    iterate();
    draw();
    glutSwapBuffers();
}

int main(int argc, char** argv) {
    for (int i = 0; i < 3; ++i) {
        float pointX, pointY;
        cout << "  Enter point " << i + 1 << " of triangle (x y): ";
        cin >> pointX >> pointY;
        points.push_back(Vector3f(pointX, pointY, 1));
    }

    float scale;
    cout << "  Enter scaling factor: ";
    cin >> scale;

    Matrix3f scaleMat;
    scaleMat << scale, 0, 0,
                0, scale, 0,
                0, 0, 1;

    newPoints.resize(points.size());
```

```
    for (size_t i = 0; i < points.size(); ++i) {
        newPoints[i] = scaleMat * points[i];
    }

    cout << "\n  Old position: cyan" << endl;
    for (const auto& point : points) {
        cout << point.transpose() << endl;
    }

    cout << "\n  Scaling Matrix: " << endl;
    cout << scaleMat << endl;

    cout << "\n  New Position: yellow" << endl;
    for (const auto& point : newPoints) {
        cout << point.transpose() << endl;
    }

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA);
    glutInitWindowSize(width, height);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Scaling");
    glutDisplayFunc(showScreen);
    glutIdleFunc(showScreen);
    glutMainLoop();

    return 0;
}
```

3-D transformation

```
#include <GL/glut.h>
#include <cmath>
#include <iostream>

using namespace std;

const int numVertices = 8;
float theMatrix[numVertices][3] = {
    {40, 40, 50}, {90, 40, 50}, {90, 90, 50}, {40, 90, 50},
    {30, 30, 0}, {80, 30, 0}, {30, 80, 0}
};
float m[4][4] = {0};
float output[numVertices][3] = {0};

void setIdentityM() {
```

```
    for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 4; ++j) {
            m[i][j] = (i == j) ? 1 : 0;
        }
    }
}

void translate(float tx, float ty, float tz) {
    for (int i = 0; i < numVertices; ++i) {
        output[i][0] = theMatrix[i][0] + tx;
        output[i][1] = theMatrix[i][1] + ty;
        output[i][2] = theMatrix[i][2] + tz;
    }
}

void scale(float sx, float sy, float sz) {
    m[0][0] = sx;
    m[1][1] = sy;
    m[2][2] = sz;
}

void rotateX(float angle) {
    angle = angle * (M_PI / 180.0);
    m[1][1] = cos(angle);
    m[1][2] = -sin(angle);
    m[2][1] = sin(angle);
    m[2][2] = cos(angle);
}

void rotateY(float angle) {
    angle = angle * (M_PI / 180.0);
    m[0][0] = cos(angle);
    m[0][2] = -sin(angle);
    m[2][0] = sin(angle);
    m[2][2] = cos(angle);
}

void rotateZ(float angle) {
    angle = angle * (M_PI / 180.0);
    m[0][0] = cos(angle);
    m[0][1] = sin(angle);
    m[1][0] = -sin(angle);
    m[1][1] = cos(angle);
}

void multiply() {
    for (int i = 0; i < numVertices; ++i) {
```

```
        for (int j = 0; j < 3; ++j) {
            output[i][j] = 0;
            for (int k = 0; k < 3; ++k) {
                output[i][j] += theMatrix[i][k] * m[k][j];
            }
        }
    }
}

void axes() {
    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_LINES);
    glVertex2f(-1000, 0);
    glVertex2f(1000, 0);
    glVertex2f(0, -1000);
    glVertex2f(0, 1000);
    glEnd();
}

void draw(float vertices[numVertices][3]) {
    glBegin(GL_QUADS);
    float colors[6][3] = {
        {0.7, 0.4, 0.5}, {0.8, 0.2, 0.4}, {0.3, 0.6, 0.7},
        {0.2, 0.8, 0.2}, {0.7, 0.7, 0.2}, {1.0, 0.1, 0.1}
    };
    for (int i = 0; i < 6; ++i) {
        glColor3f(colors[i][0], colors[i][1], colors[i][2]);
        for (int j = 0; j < 4; ++j) {
            glVertex3fv(vertices[(i * 4 + j) % numVertices]);
        }
    }
    glEnd();
}

void init() {
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glOrtho(-454.0, 454.0, -250.0, 250.0, -250.0, 250.0);
    glEnable(GL_DEPTH_TEST);
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    axes();
    draw(theMatrix);
    setIdentityM();

    int choice;
```

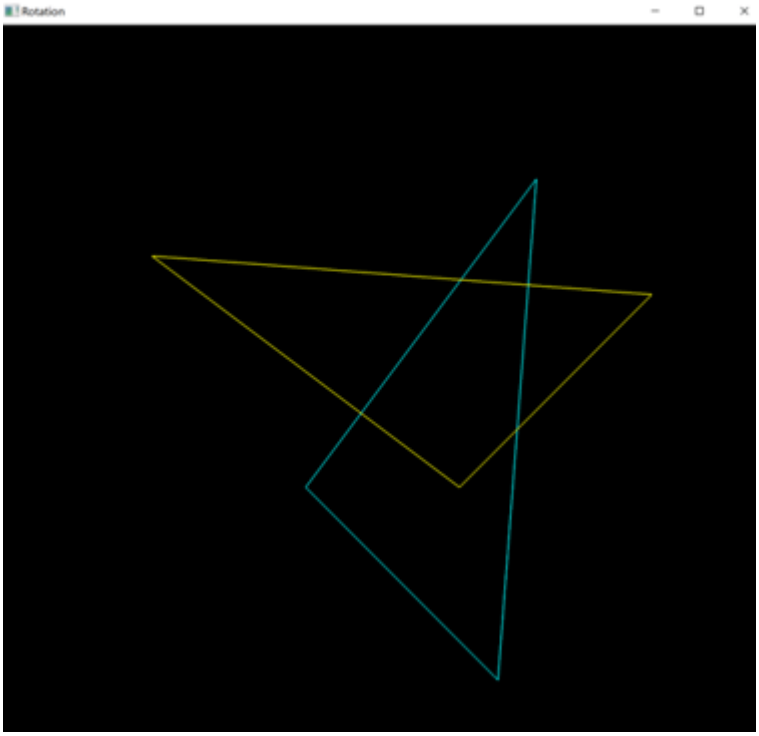
```
    cout << "Enter your choice number:\n1. Translation\n2. Scaling\n3.
Rotation\n: ";
    cin >> choice;

    if (choice == 1) {
        float tx, ty, tz;
        cout << "Enter Tx: ";
        cin >> tx;
        cout << "Enter Ty: ";
        cin >> ty;
        cout << "Enter Tz: ";
        cin >> tz;
        translate(tx, ty, tz);
    } else if (choice == 2) {
        float sx, sy, sz;
        cout << "Enter Sx: ";
        cin >> sx;
        cout << "Enter Sy: ";
        cin >> sy;
        cout << "Enter Sz: ";
        cin >> sz;
        scale(sx, sy, sz);
        multiply();
    } else if (choice == 3) {
        int choicerot;
        float angle;
        cout << "Enter your choice for Rotation about axis:\n1. X-
axis\n2. Y-axis\n3. Z-axis\n: ";
        cin >> choicerot;
        cout << "Enter Rotation Angle: ";
        cin >> angle;

        if (choicerot == 1) {
            rotateX(angle);
        } else if (choicerot == 2) {
            rotateY(angle);
        } else if (choicerot == 3) {
            rotateZ(angle);
        }
    }

    multiply();
    draw(output);
    glFlush();
}

int main(int argc, char** argv) {
```

```
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(1362, 750);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("3D Transformations");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

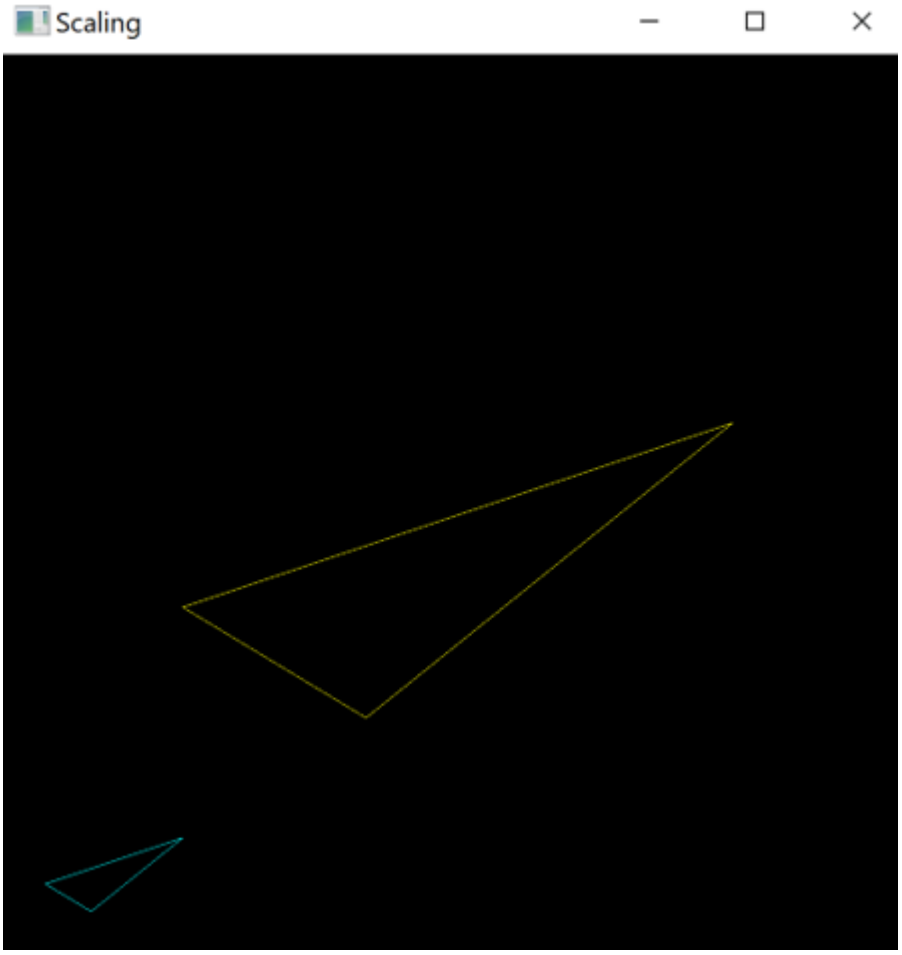
Output(s) (Screen Shot):

Translation:

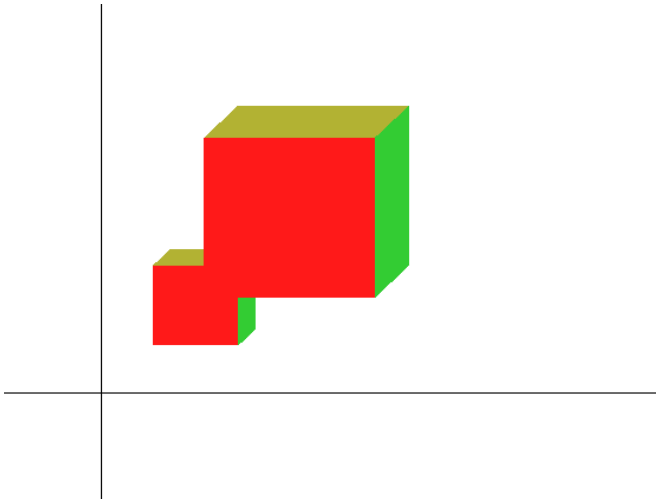
Rotation:

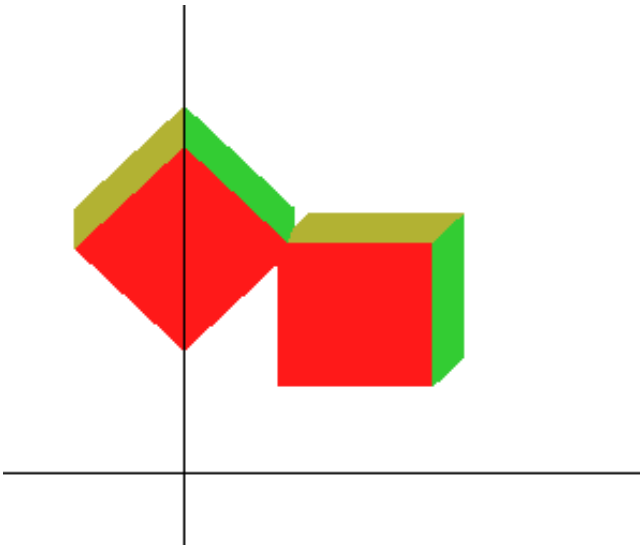


Scaling:



3-D Transformation





Screenshots from VLab(if any):

Pre Test

If the object is moved along positive X- direction then the new x coordinate of the object will be greater than the old x coordinate.

- (●) a : True
- ( ) b : False

If the object is moved along negative X- direction then the new x coordinate of the object will be greater than the old x coordinate.

- (●) a : True
- ( ) b : False

If the object is moved along positive Y- direction then the new y coordinate of the object will be greater than the old y coordinate.

- ( ) a : True
- (●) b : False

If the object is moved along negative Y- direction then the new y coordinate of the object will be greater.

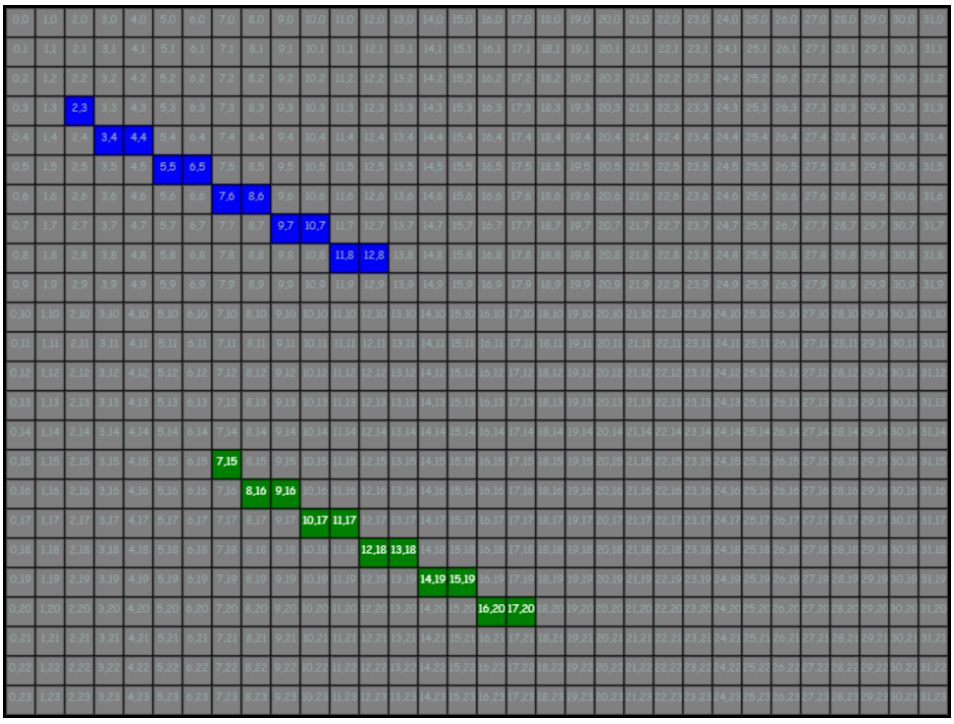
- (●) a : True
- ( ) b : False

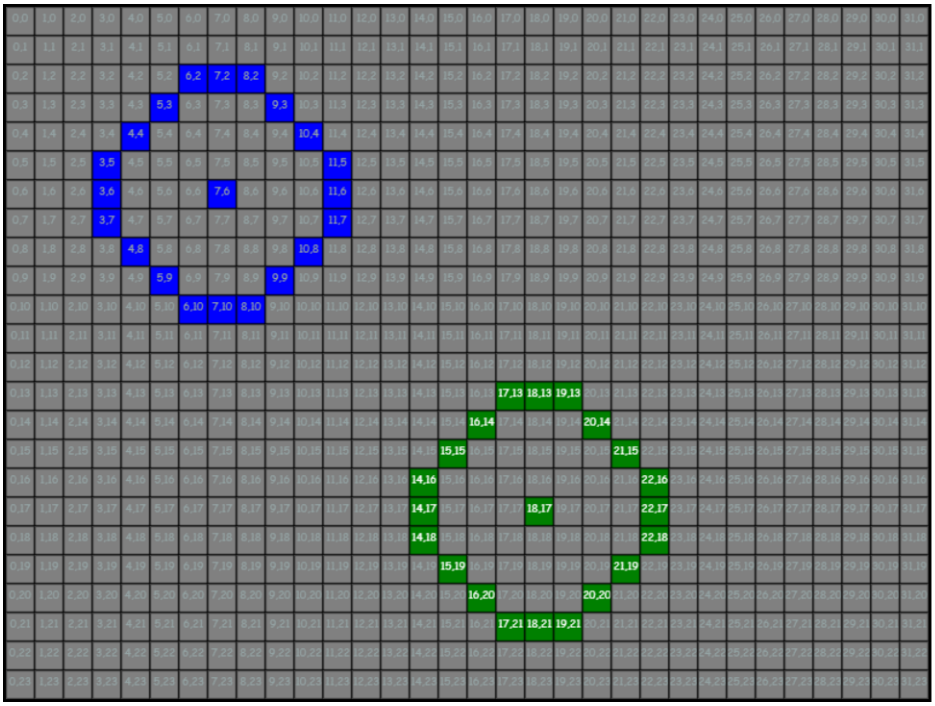
If the object is moved along positive X and Y directions, then the new (x', y') coordinate of the object will be greater than old (x, y) coordinate.

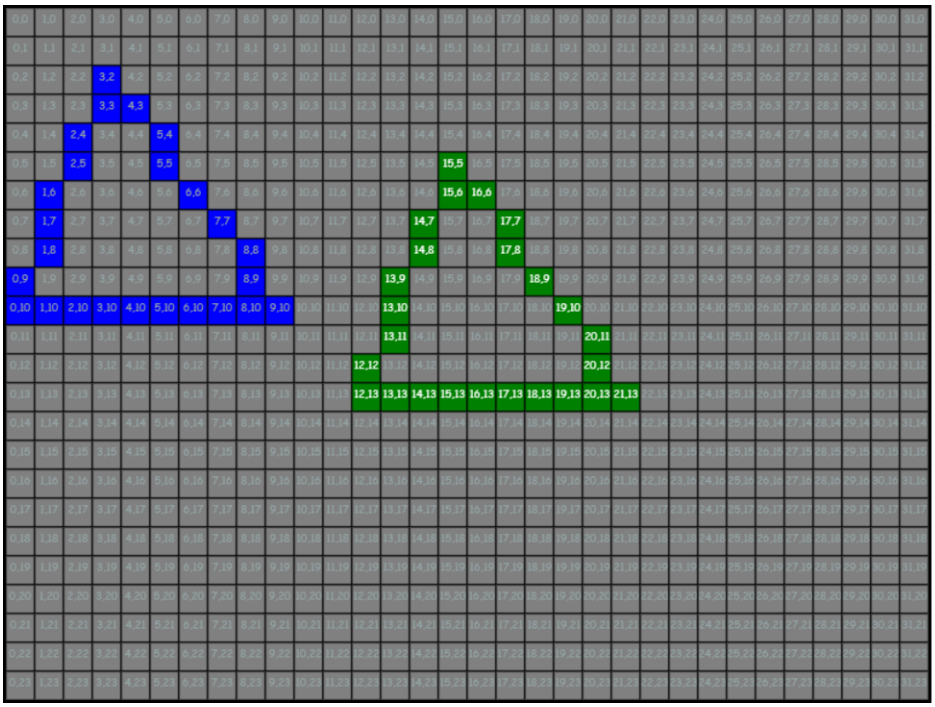
- (●) a : True
- ( ) b : False

Submit Quiz

5 out of 5







Post Test

If the point P(10,10) is translated along X-axis and Y-axis by translation factor Tx and Ty as 10, in respective directions, then what will be the new coordinate of P'?
- a : (0,0)
- b : (20,10)
- c : (10,20)
- d : (20,20) ✓

If the point P(10,10) is translated along X-axis and Y-axis by translation factor Tx =10 Ty=0; then what will be the new coordinate of P'?
- a : (10,20)
- b : (20,10) ✓
- c : (20,20)
- d : (0,0)

If the point P(10,10) is translated along X-axis and Y-axis by translation factor Tx =0 Ty=10; then what will be the new coordinate of P'?
- a : (20,10)
- b : (10,20) ✓
- c : (20,20)
- d : (0,0)

If the point P(10,10) is translated along X-axis and Y-axis by translation factor Tx = -10 Ty= -10; then what will be the new coordinate of P'?
- a : (0,0) ✓
- b : (10,20)
- c : (20,10)
- d : (20,20)

If the point P(10,10) is translated along X-axis and Y-axis by translation factor Tx = -10 Ty= 10; then what will be the new coordinate of P'?
- a : (0,20) ✓
- b : (10,20)
- c : (20,20)
- d : (0,0)

Submit Quiz

5 out of 5

Pre Test

1. If the object size is enlarged i.e doubled, then the coordinates of the object will be doubled.
- a : True ✓
- b : False

2. If the object size is reduced by half, then the coordinates of the object will also be reduced by half.
- a : False
- b : True ✓

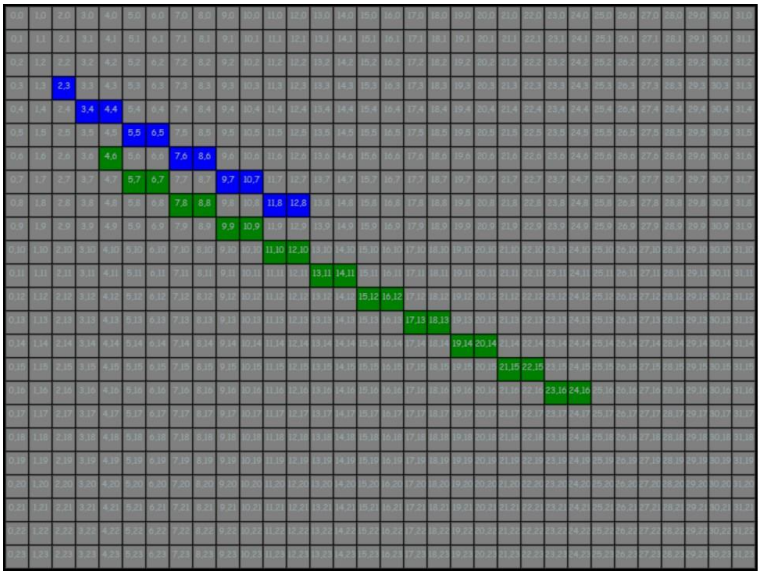
3. If the object is scaled in x direction, then the object will be enlarged in X-direction only.
- a : False
- b : True ✓

4. If the object is scaled in x direction, then the object will be reduced in Y-direction.
- a : False ✓
- b : True

5. The object square can be scaled in x and y direction to convert the same into a rectangle.
- a : True ✓
- b : False

Submit Quiz

5 out of 5



1. Enter Scaling matrix : [2 0 / 0 2]

CHECK MATRIX

2. Enter the scaled co-ordinates for the given points :

$\begin{bmatrix}2\\3\end{bmatrix}\begin{bmatrix}2&0\\0&2\end{bmatrix} = \begin{bmatrix}4\\6\end{bmatrix}$

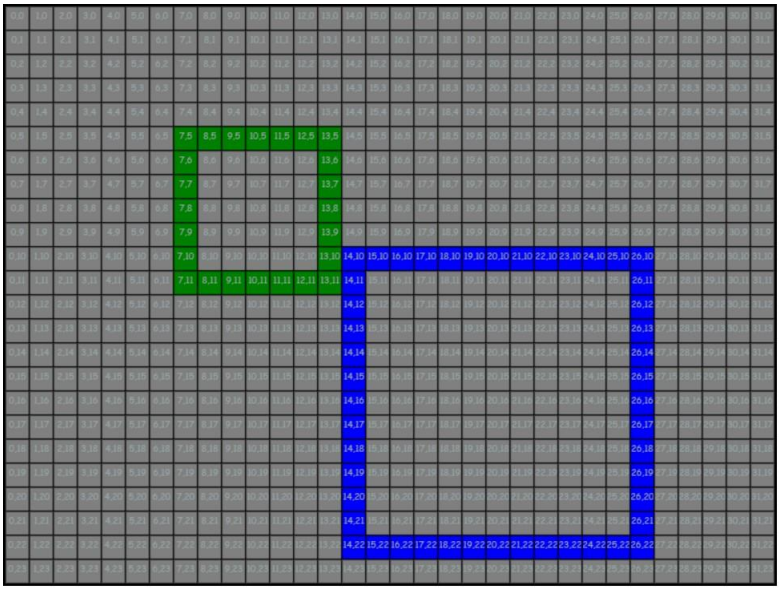
$\begin{bmatrix}12\\8\end{bmatrix}\begin{bmatrix}2&0\\0&2\end{bmatrix} = \begin{bmatrix}24\\16\end{bmatrix}$

CHECK SCALED CO-ORDINATES

3. Plot the scaled points on the canvas

4. Click the button below to join the points

JOIN POINTS



1. Enter Scaling matrix : [0.5 0 / 0 0.5]

CHECK MATRIX

2. Enter the scaled co-ordinates for the given points :

$\begin{bmatrix}14\\10\end{bmatrix}\begin{bmatrix}0.5&0\\0&0.5\end{bmatrix} = \begin{bmatrix}7\\5\end{bmatrix}$

$\begin{bmatrix}14\\22\end{bmatrix}\begin{bmatrix}0.5&0\\0&0.5\end{bmatrix} = \begin{bmatrix}7\\11\end{bmatrix}$

$\begin{bmatrix}26\\10\end{bmatrix}\begin{bmatrix}0.5&0\\0&0.5\end{bmatrix} = \begin{bmatrix}13\\5\end{bmatrix}$

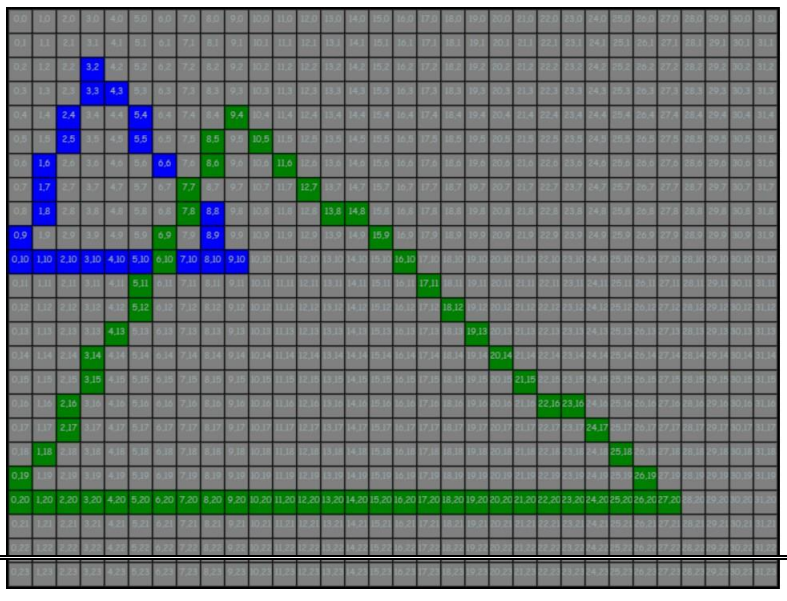
$\begin{bmatrix}26\\22\end{bmatrix}\begin{bmatrix}0.5&0\\0&0.5\end{bmatrix} = \begin{bmatrix}13\\11\end{bmatrix}$

CHECK SCALED CO-ORDINATES

3. Plot the scaled points on the canvas

4. Click the button below to join the points

JOIN POINTS



1. Enter Scaling matrix : [3 0 / 0 2]

CHECK MATRIX

2. Enter the scaled co-ordinates for the given points :

$\begin{bmatrix}3\\2\end{bmatrix}\begin{bmatrix}3&0\\0&2\end{bmatrix} = \begin{bmatrix}9\\4\end{bmatrix}$

$\begin{bmatrix}0\\10\end{bmatrix}\begin{bmatrix}3&0\\0&2\end{bmatrix} = \begin{bmatrix}0\\20\end{bmatrix}$

$\begin{bmatrix}9\\10\end{bmatrix}\begin{bmatrix}3&0\\0&2\end{bmatrix} = \begin{bmatrix}27\\20\end{bmatrix}$

CHECK SCALED CO-ORDINATES

3. Plot the scaled points on the canvas

4. Click the button below to join the points

Post Test

1. If the point P(10,10) is scaled along X-axis and Y-axis by scaling factor Sx=3 and Sy=3, in respective directions, then what will be the new coordinate of point P'?
- a : (0,0)
- b : (30,10)
- c : (10,30)
- d : (30,30) ✓

2. If the point P(10,10) is scaled along X-axis and Y-axis by scaling factor Sx=3 and Sy=1, in respective directions, then what will be the new coordinate of point P'?
- a : (10,30)
- b : (30,10) ✓
- c : (30,30)
- d : (0,0)

3. If the point P(10,10) is scaled along X-axis and Y-axis by scaling factor Sx=1 and Sy=3, in respective directions, then what will be the new coordinate of point P'?
- a : (30,10)
- b : (10,30) ✓
- c : (30,30)
- d : (0,0)

4. If the point P(10,10) is scaled along X-axis and Y-axis by scaling factor Sx=0.5 and Sy=2, in respective directions, then what will be the new coordinate of point P'?
- a : (5,20) ✓
- b : (0,0)
- c : (5,5)
- d : (20,20)

5. If the point P(10,10) is scaled along X-axis and Y-axis by scaling factor Sx=1.5 and Sy=0.5, in respective directions, then what will be the new coordinate of P'?
- a : (15,5) ✓
- b : (5,20)
- c : (20,20)
- d : (0,0)

Submit Quiz

5 out of 5

Pre Test

1. If the object is rotated, then the coordinates of the object will be changed.
- a : True ✓
- b : False

2.If the rectangle is rotated in either clockwise or anticlockwise direction; then it will change its shape.
- a : True
- b : False ✓

3. The shape of the object will remains same after performing the rotation.
- a : True ✓
- b : False

4. If the object is rotated in clockwise direction, then all the point will be rotated in clockwise direction.
- a : True ✓
- b : False

5. The point P(0,0) is rotated with some angle , then it will retains its original coordinates P'(0,0').
- a : True ✓
- b : False

Submit Quiz

5 out of 5

Conclusion and discussion:

Successfully learnt and implemented Scaling, Transformation and Rotation. Linear transformations can be applied to objects by representing points within a coordinate system and using matrices for the transformations. By multiplying the points with the transformation matrix, the resulting matrix provides the coordinates for the transformed object. Varying the values of the transformation matrices produces different effects, such as rotation, translation, scaling, shearing, and more. Additionally, multiple transformation matrices can be combined through multiplication to create a single matrix that represents several transformations, which is particularly useful when applying the same set of transformations to multiple objects.

Date: 18-10-2024

Signature of faculty in-charge