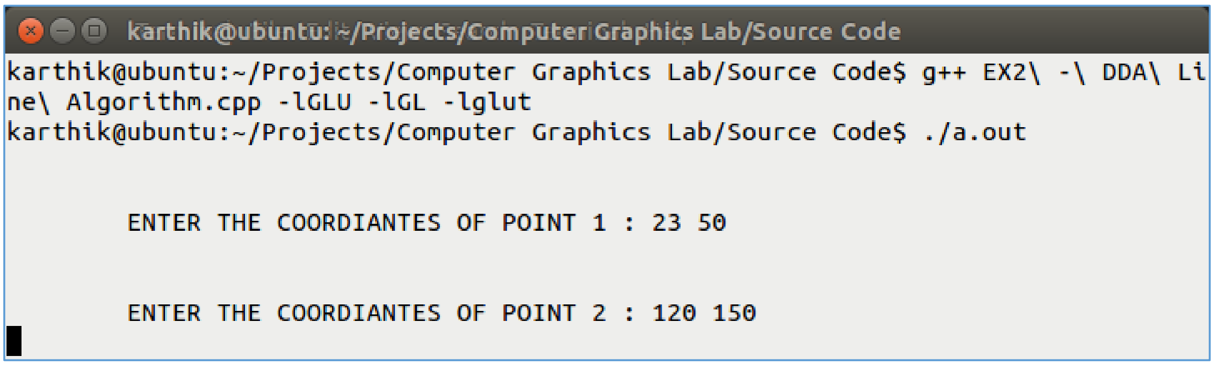
EX 2:

PROGRAM: DDA

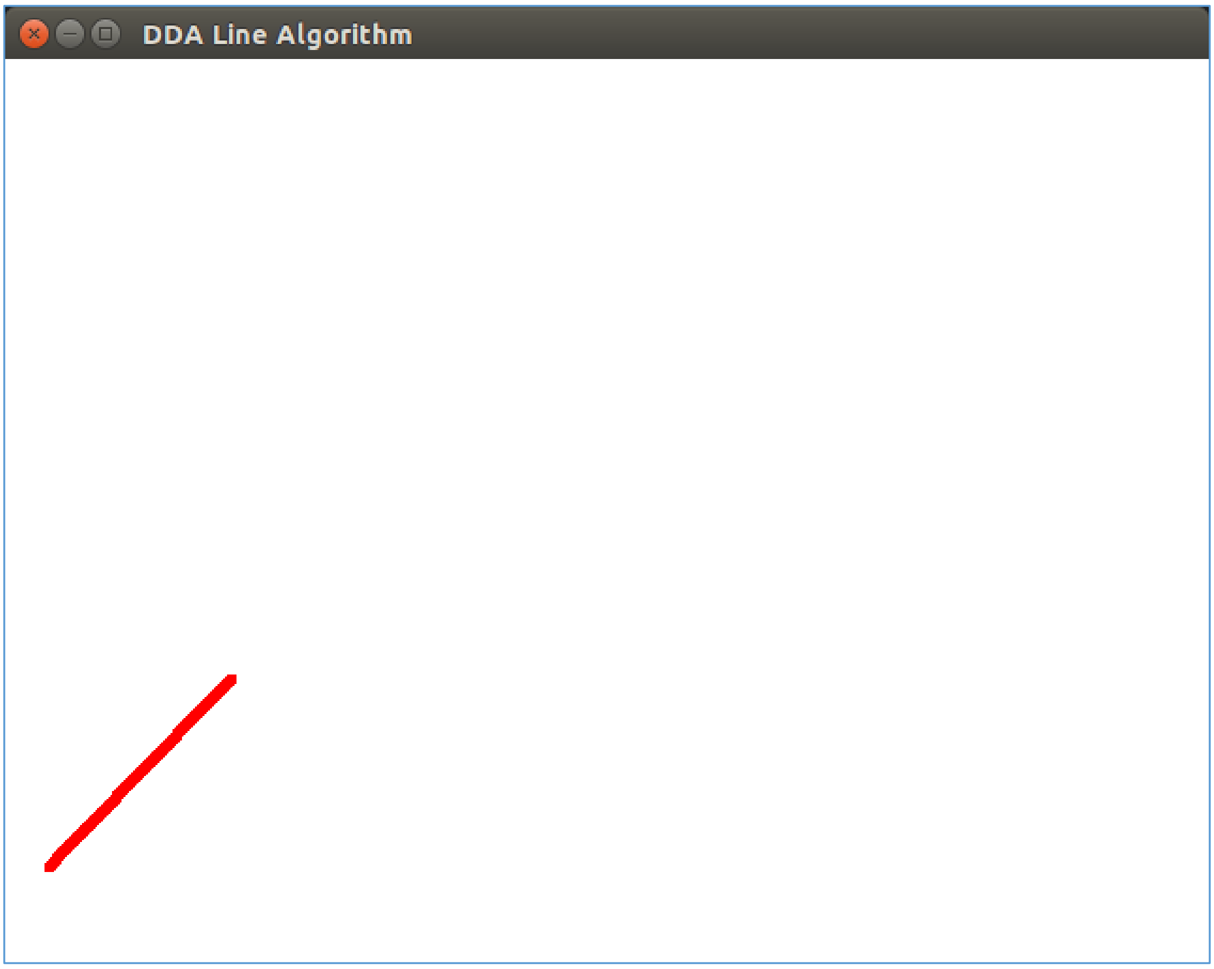
```
#include <GL/glut.h>
#include <iostream>

using namespace std;

//Window Parameters
const int WINDOW_WIDTH = 640;
const int WINDOW_HEIGHT = 480;
const char WINDOW_TITLE[] = "DDA Line Algorithm";

//Canvas Parameters
const float PLANE_LEFT = 0.0;
const float PLANE_RIGHT = WINDOW_WIDTH;
const float PLANE_BOTTOM = 0.0;
const float PLANE_TOP = WINDOW_HEIGHT;
const int POINT_SIZE = 5;

//Condtions
float x1, y1, x2, y2;
float m, inc;
float x, y;

void canvasInit() {
    //Initialize the canvas
    glClearColor(1, 1, 1, 0);
    glColor3f(1, 0, 0);
    glPointSize(POINT_SIZE);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(PLANE_LEFT, PLANE_RIGHT, PLANE_BOTTOM, PLANE_TOP);

}

void myDisplay() {
    glClear(GL_COLOR_BUFFER_BIT);

    /* Implement your drawing logic here */
    glBegin(GL_POINTS);
        //Calculate slope
        m = (y2 - y1) / (x2 - x1);
        if(m <= 1 && m >= -1) {
            //Set increment variable
            inc = m;

            //Swap the points if we have to traverse from right to left
            if(x1 > x2) {
                int temp = x1; x1 = x2; x2 = temp;
                temp = y1; y1 = y2; y2 = temp;
            }

            //Digital Differential Algorithm for x+=1
            y = y1;
            for(int x=x1; x<=x2; x++) {
                glVertex2d(x, y);
                y += inc;
            }
        } else {
            //Set increment variable
            inc = 1/m;

            //Swap the point if we have to traverse from right to left
            if (y1 > y2) {
                int temp = x1; x1 = x2; x2 = temp;
                temp = y1; y1 = y2; y2 = temp;
            }

            //Digital Differential Algorithm for y+=1
            x = x1;
            for(int y=y1; y<=y2; y++) {
                glVertex2d(x, y);
                x += inc;
            }
        }
    glEnd();
    glFlush();
}

int main(int argc, char** argv) {
    /* Get any input you need here */
    cout<<"\n\n\tENTER THE COORDIANTES OF POINT 1 : ";
    cin>>x1>>y1;
    cout<<"\n\n\tENTER THE COORDIANTES OF POINT 2 : ";
    cin>>x2>>y2;

    //Initialize the main window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    glutCreateWindow(WINDOW_TITLE);
    glutDisplayFunc(myDisplay);

    //Initialize the canvas and start the main loop
    canvasInit();
    glutMainLoop();

    return 0;
}
```

OUTPUT:



karthik@ubuntu: ~/Projects/Computer Graphics Lab/Source Code

```
karthik@ubuntu:~/Projects/Computer Graphics Lab/Source Code$ g++ EX2\ -\ DDA\ Li
ne\ Algorithm.cpp -lGLU -lGL -lglut
karthik@ubuntu:~/Projects/Computer Graphics Lab/Source Code$ ./a.out


        ENTER THE COORDIANTES OF POINT 1 : 23 50


        ENTER THE COORDIANTES OF POINT 2 : 120 150
```

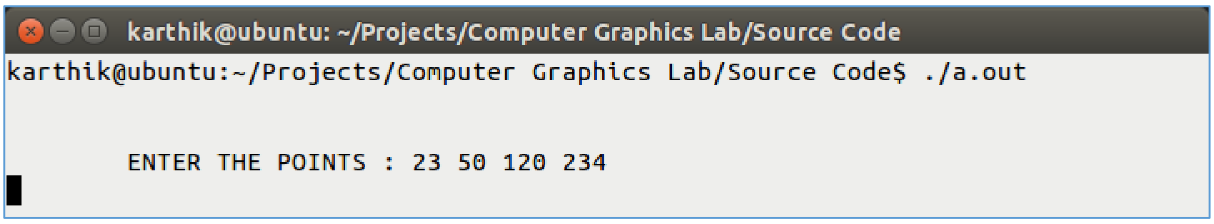

DDA Line Algorithm

PROGRAM: BRESHENAM'S

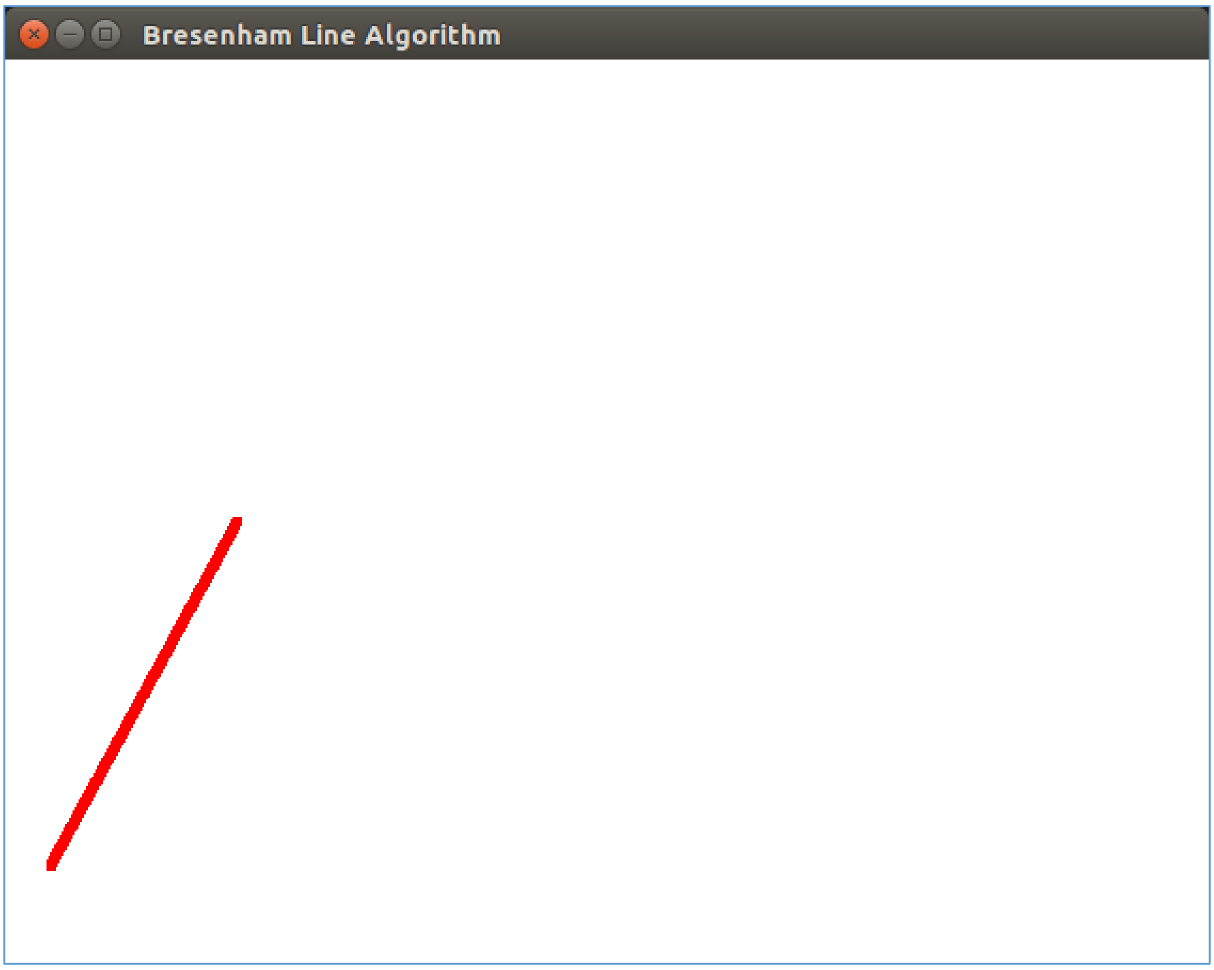
```
#include <GL/glut.h>
#include <iostream>

using namespace std;

//Window Parameters
const int WINDOW_WIDTH = 640;
const int WINDOW_HEIGHT = 480;
const char WINDOW_TITLE[] = "Bresenham Line Algorithm";

//Canvas Parameters
const float PLANE_LEFT = 0.0;
const float PLANE_RIGHT = WINDOW_WIDTH;
const float PLANE_BOTTOM = 0.0;
const float PLANE_TOP = WINDOW_HEIGHT;
const int POINT_SIZE = 5;

/*Create the variables here*/
float x1, y1, x2, y2;
float m;

void canvasInit() {
       //Initialize the canvas
       glClearColor(1, 1, 1, 0);
       glColor3f(1, 0, 0);
       glPointSize(POINT_SIZE);
       glMatrixMode(GL_PROJECTION);
       glLoadIdentity();
       gluOrtho2D(PLANE_LEFT, PLANE_RIGHT, PLANE_BOTTOM, PLANE_TOP);
}

void myDisplay() {
       glClear(GL_COLOR_BUFFER_BIT);

       if (m <= 1) {
              //Swap the points to start from left to right
              int fx = x1, fy = y1;
              if (x1 > x2) {
                     fx = x1; fy = y1;
                     x1 = x2; y1 = y2;
                     x2 = fx; y2 = fy;
              }

              //Calculate values for computation
              int dx = x2 - x1;
              int dy = y2 - y1;
              int pk = 2 * dy - dx;

              //Draw the actual line
              glBegin(GL_POINTS);
              while (fx < x2) {
                     if (pk < 0) {
                            fx++;
                            pk += 2 * dy;
                     }
                     else {
                            fx++; fy++;
                            pk += 2 * (dy - dx);
                     }
```

```
                        glVertex2d(fx, fy);
                }
                glEnd();
        }
        else {
                //Swap the points to start from left to right
                int fx = x1, fy = y1;
                if (y1 > y2) {
                        fx = x1; fy = y1;
                        x1 = x2; y1 = y2;
                        x2 = fx; y2 = fy;
                }

                //Calculate values for computation
                int dx = x2 - x1;
                int dy = y2 - y1;
                int pk = 2 * dx - dy;

                //Draw the actual line
                glBegin(GL_POINTS);
                while (fy < y2) {
                        if (pk < 0) {
                                fy++;
                                pk += 2 * dx;
                        }
                        else {
                                fx++; fy++;
                                pk += 2 * (dx - dy);
                        }

                        glVertex2d(fx, fy);
                }
                glEnd();
        }

        glFlush();
}

int main(int argc, char** argv) {
        /* Get any input you need here */
        cout << "\n\n\tENTER THE POINTS : ";
        cin >> x1 >> y1 >> x2 >> y2;
        m = (y2 - y1) / (x2 - x1);

        //Initialize the main window
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
        glutCreateWindow(WINDOW_TITLE);
        glutDisplayFunc(myDisplay);

        //Initialize the canvas and start the main loop
        canvasInit();
        glutMainLoop();
        return 0;
}
```

OUTPUT:



```
karthik@ubuntu: ~/Projects/Computer Graphics Lab/Source Code
karthik@ubuntu:~/Projects/Computer Graphics Lab/Source Code$ ./a.out


        ENTER THE POINTS : 23 50 120 234
```



RESULT:

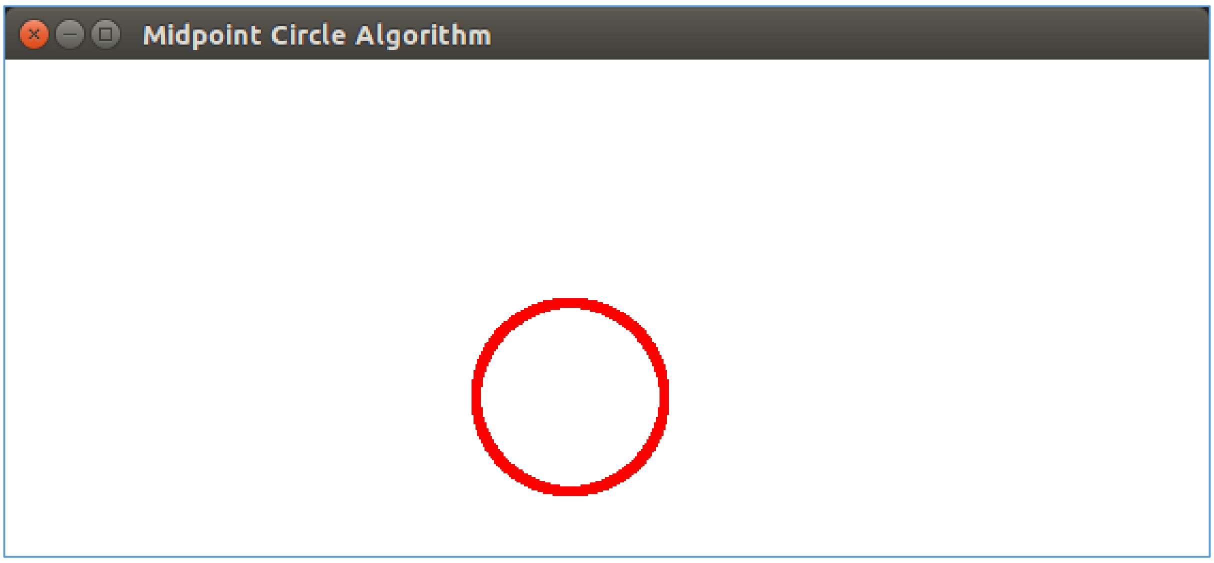
THE LINE DRAWING ALGORITHM WAS SUCCESSFULLY IMPLEMENTED.

EX 3:

PROGRAM: MIDPOINT CIRCLE

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

using namespace std;

//Window Parameters
const int WINDOW_WIDTH = 640;
const int WINDOW_HEIGHT = 480;
const char WINDOW_TITLE[] = "Midpoint Circle Algorithm";

//Canvas Parameters
const float PLANE_LEFT = 0.0;
const float PLANE_RIGHT = WINDOW_WIDTH;
const float PLANE_BOTTOM = 0.0;
const float PLANE_TOP = WINDOW_HEIGHT;
const int POINT_SIZE = 5;

/*Conditional Parameters*/
int radius;
int centerX, centerY;

void canvasInit() {
    //Initialize the canvas
    glClearColor(1, 1, 1, 0);
    glColor3f(1, 0, 0);
    glPointSize(POINT_SIZE);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(PLANE_LEFT, PLANE_RIGHT, PLANE_BOTTOM, PLANE_TOP);
}

void myDisplay() {
    glClear(GL_COLOR_BUFFER_BIT);

    /* Implement your drawing logic here */
    int x = radius;
    int y = 0;
    int pk = 1 - x;

    glBegin(GL_POINTS);
        while (y <= x) {
            //Since the circle is symmetric about both x-axis, y-axis and origin
            //The circle can be considered as 8 equal parts
            //So Calculate for one part and extends it to the other parts
            glVertex2d( x + centerX,  y + centerY);
            glVertex2d( y + centerX,  x + centerY);
            glVertex2d(-x + centerX,  y + centerY);
            glVertex2d(-y + centerX,  x + centerY);
            glVertex2d(-x + centerX, -y + centerY);
            glVertex2d(-y + centerX, -x + centerY);
            glVertex2d( x + centerX, -y + centerY);
            glVertex2d( y + centerX, -x + centerY);


            //Increment x (if required), y, pk(based on the condition)
            y++;
            if (pk <= 0) { pk += 2 * y + 1; }
            else {
                x--;
                pk += 2 * (y - x) + 1;
            }
        }
    glEnd();

    glFlush();
}

int main(int argc, char** argv) {
    /* Get any input you need here */
    cout<<"\n\n\tENTER THE RADIUS : ";
    cin>>radius;
    cout<<"\n\tENTER THE CENTER : ";
    cin>>centerX>>centerY;

    //Initialize the main window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    glutCreateWindow(WINDOW_TITLE);
    glutDisplayFunc(myDisplay);

    //Initialize the canvas and start the main loop
    canvasInit();
    glutMainLoop();
    return 0;
}
```

OUTPUT:

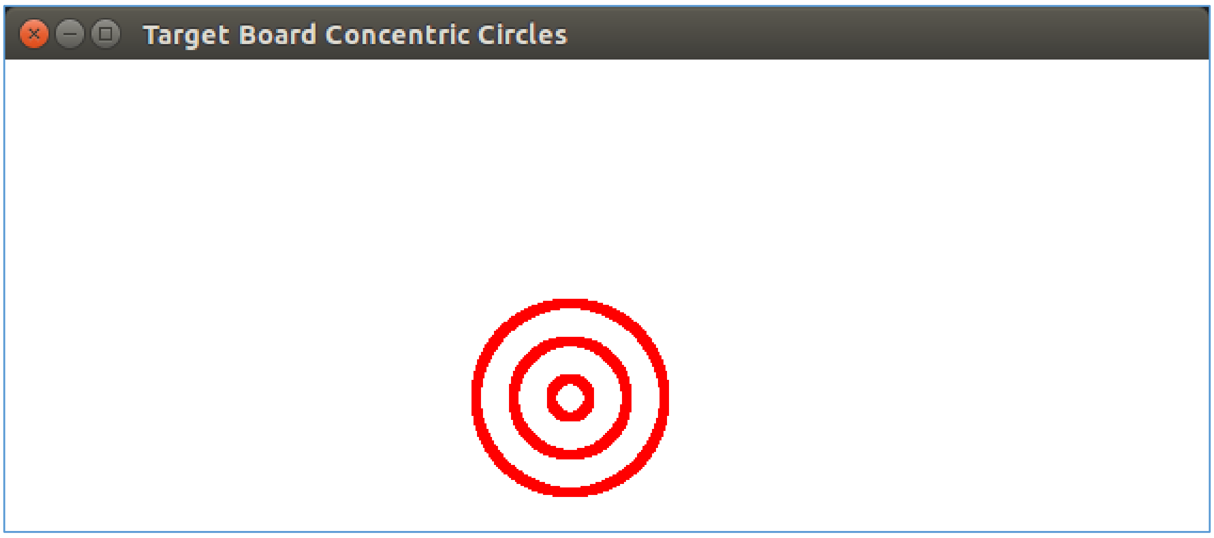


PROGRAM: TARGET BOARD

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

using namespace std;

//Window Parameters
const int WINDOW_WIDTH = 640;
const int WINDOW_HEIGHT = 480;
const char WINDOW_TITLE[] = "Target Board Concentric Circles";

//Canvas Parameters
const float PLANE_LEFT = 0.0;
const float PLANE_RIGHT = WINDOW_WIDTH;
const float PLANE_BOTTOM = 0.0;
const float PLANE_TOP = WINDOW_HEIGHT;
const int POINT_SIZE = 5;

/*Conditional Parameters*/
int radius;
int centerX, centerY;

void canvasInit() {
    //Initialize the canvas
    glClearColor(1, 1, 1, 0);
    glColor3f(1, 0, 0);
    glPointSize(POINT_SIZE);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(PLANE_LEFT, PLANE_RIGHT, PLANE_BOTTOM, PLANE_TOP);
}

void circle(int centerX, int centerY, int radius) {
    //Initialize the required variables
    int x = radius;
    int y = 0;
    int pk = 1 - x;

    while (y <= x) {
        //Since the circle is symmetric about both x-axis, y-axis and origin
        //The circle can be considered as 8 equal parts
        //So Calculate for one part and extends it to the other parts
        glVertex2d( x + centerX,  y + centerY);
        glVertex2d( y + centerX,  x + centerY);
        glVertex2d(-x + centerX,  y + centerY);
        glVertex2d(-y + centerX,  x + centerY);
        glVertex2d(-x + centerX, -y + centerY);
        glVertex2d(-y + centerX, -x + centerY);
        glVertex2d( x + centerX, -y + centerY);
        glVertex2d( y + centerX, -x + centerY);


        //Increment x (if required), y, pk(based on the condition)
        y++;
        if (pk <= 0) { pk += 2 * y + 1; }
        else {
            x--;
            pk += 2 * (y - x) + 1;
        }
    }
}
```

```
void myDisplay() {
    glClear(GL_COLOR_BUFFER_BIT);

    /* Implement your drawing logic here */
    glBegin(GL_POINTS);
    for(int r = radius; r >= 0; r-=20) {
        circle(centerX, centerY, r);
    }
    glEnd();

    glFlush();
}

int main(int argc, char** argv) {
    /* Get any input you need here */
    cout<<"\n\n\tENTER THE RADIUS : ";
    cin>>radius;
    cout<<"\n\tENTER THE CENTER : ";
    cin>>centerX>>centerY;

    //Initialize the main window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    glutCreateWindow(WINDOW_TITLE);
    glutDisplayFunc(myDisplay);

    //Initialize the canvas and start the main loop
    canvasInit();
    glutMainLoop();
    return 0;
}
```

OUTPUT:



RESULT:

THE CIRCLE ALGORITHMS WERE SUCCESSFULLY DEMONSTRATED.

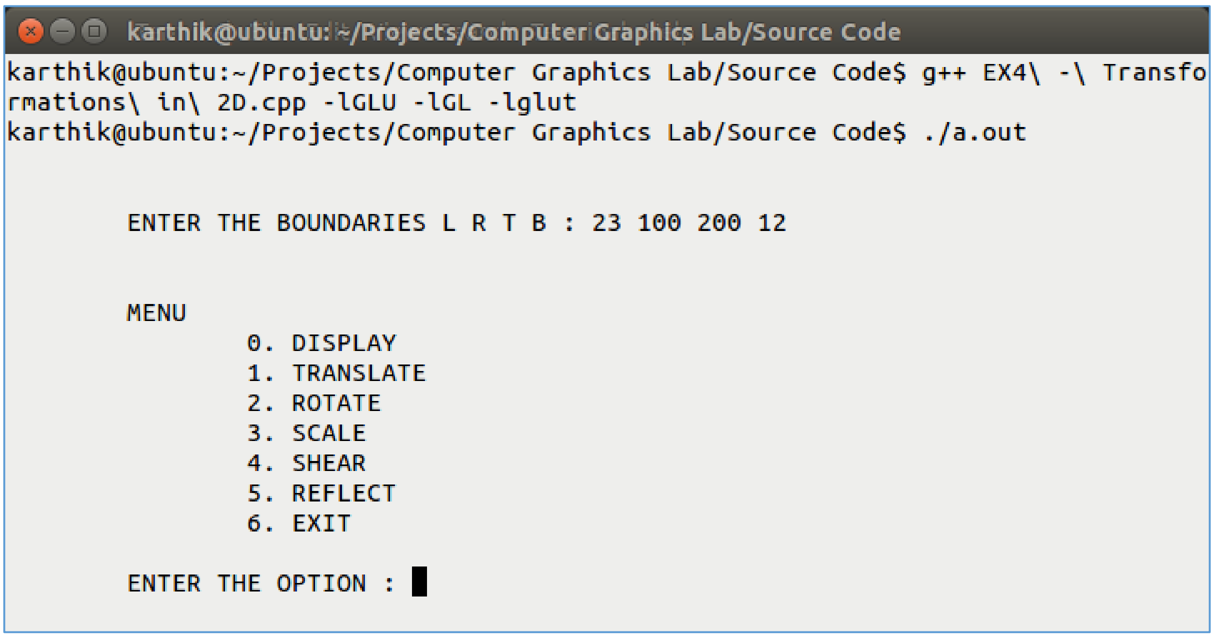
EX 4:

PROGRAM: TRANSFORMATIONS IN 2D

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

using namespace std;

//Window Parameters
const int WINDOW_WIDTH = 640;
const int WINDOW_HEIGHT = 480;
const char WINDOW_TITLE[] = "2D  Transformations in Homogeneous coordinates";

//Canvas Parameters
const float PLANE_LEFT = 0.0;
const float PLANE_RIGHT = WINDOW_WIDTH;
const float PLANE_BOTTOM = 0.0;
const float PLANE_TOP = WINDOW_HEIGHT;
const int POINT_SIZE = 5;

/*Create the variables here*/
int quadLeft, quadRight, quadTop, quadBottom;
enum TRANSFORMATIONS { NORMAL, TRANSLATE, ROTATE, SCALE,  SHEAR, REFLECT};

struct Matrix {
    float a[3][3];
    int m, n;
};

Matrix multMatrix(Matrix mat1, Matrix mat2) {
    Matrix res = {{0}, mat1.m, mat2.n};

    //Multiplication of the matrices
    if (mat1.n == mat2.m) {
        for(int i=0; i<mat1.m; i++) {
            for(int j=0; j<mat2.n; j++) {
                for(int k=0; k<3; k++) {
                    res.a[i][j] += mat1.a[i][k] * mat2.a[k][j];
                }
            }
        }
    }

    return res;
}

void canvasInit() {
    //Initialize the canvas
    glClearColor(1, 1, 1, 0);
    glColor3f(1, 0, 0);
    glPointSize(POINT_SIZE);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(PLANE_LEFT, PLANE_RIGHT, PLANE_BOTTOM, PLANE_TOP);
}

Matrix transformPoint(int x, int y, int op, float p1, float p2) {
    Matrix trans = {
        {   /*The transformation Matrix*/
```

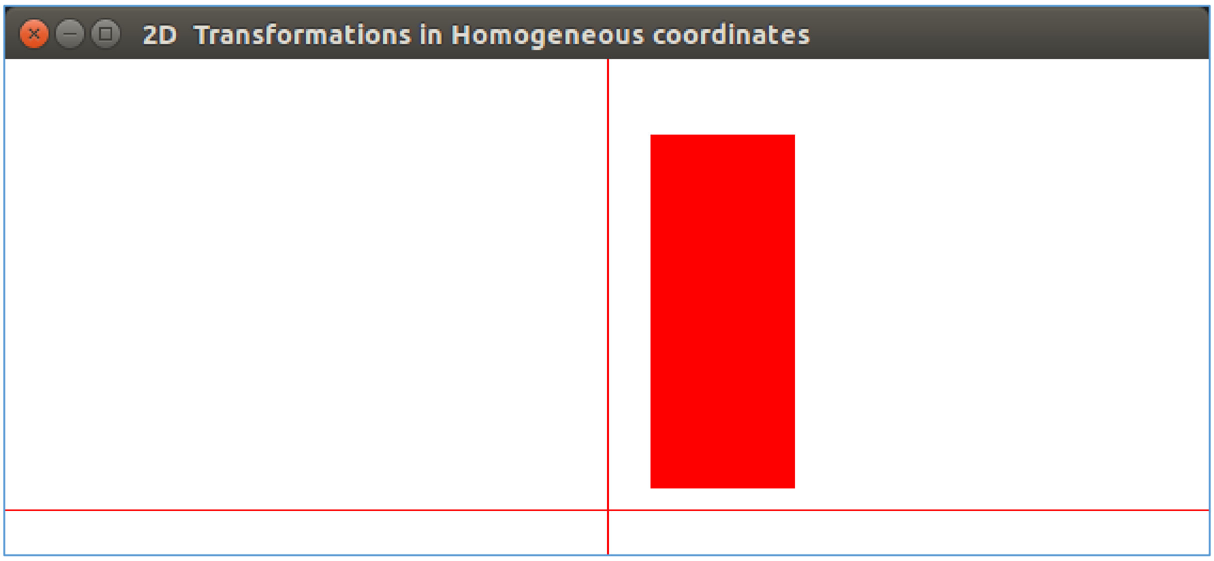
```
            {1, 0, 0},
            {0, 1, 0},
            {0, 0, 1}
        }, 3, 3
    };
    Matrix point = {
        {
            /*The point in homogeneous coordinates*/
            {x},
            {y},
            {1}
        }, 3, 1
    };

    switch (op) {
        case 0: {
            //Display as it is without any transformation
            break;
        }
        case 1: {
            //Translation of the given point
            //p1 is the x-translation factor
            //p2 is the y-translation factor
            trans.a[0][2] = p1;
            trans.a[1][2] = p2;
            break;
        }
        case 2: {
            //Rotation of the given point
            //p1 is the rotation angle
            //p2 is a place holder
            float angle = p1 * M_PI / 180;
            trans.a[0][0] = cos(angle);
            trans.a[0][1] = -sin(angle);
            trans.a[1][0] = -trans.a[0][1];
            trans.a[1][1] = trans.a[0][0];
            break;
        }
        case 3: {
            //Scaling of the given point
            //p1 is the scaleX factor
            //p2 is the scaleY factor
            trans.a[0][0] = p1;
            trans.a[1][1] = p2;
            break;
        }
        case 4: {
            //Shearing of the given point
            //p1 is the shearX factor
            //p2 is the shearY factor
            trans.a[0][1] = p1;
            trans.a[1][0] = p2;
            break;
        }
        case 5: {
            //Reflection of the given point
            //p1 is the Reflection about point
            //0 is no Reflection
            //1 is Reflection about y-axis
            //3 is Reflection about x-axis
            float angle = (90 * M_PI / 180) * p1;
            trans.a[0][0] = cos(angle);
            trans.a[0][1] = -sin(angle);
            trans.a[1][0] = -trans.a[0][1];
            trans.a[1][1] = trans.a[0][0];
            break;
        }
    }

    return multMatrix(trans, point);
}

void myDisplay() {
    int op;
    float p1, p2;

    while(true) {
        //Read the data from user
        cout<<"\n\n\tENTER THE OPTION : ";
        cin>>op;
        cout<<"\n\tENTER THE PARAMETERS IF ANY : ";
        cin>>p1>>p2;

        //Stop if option is stop
        if (op == 6) {
            break;
        }

        //Clear the canvas
        glClear(GL_COLOR_BUFFER_BIT);

        //Find the mid-point offset
        int xOffset = WINDOW_WIDTH / 2;
        int yOffset = WINDOW_HEIGHT / 2;

        //Draw the Cartesian coordinates
        glBegin(GL_LINES);
            glVertex2d(0, yOffset);
            glVertex2d(WINDOW_WIDTH, yOffset);
            glVertex2d(xOffset, 0);
            glVertex2d(xOffset, WINDOW_HEIGHT);
        glEnd();

        //Transform the points and display them as a quad
        glBegin(GL_QUADS);
            //Transformed points
            Matrix point1 = transformPoint(quadLeft, quadBottom, op, p1 , p2);
            Matrix point2 = transformPoint(quadLeft, quadTop, op, p1, p2);
            Matrix point3 = transformPoint(quadRight, quadTop, op, p1, p2);
            Matrix point4 = transformPoint(quadRight, quadBottom, op, p1, p2);

            //Original untouched quad
            glColor3f(0, 1, 0);
            glVertex2d(xOffset + quadLeft, yOffset + quadBottom);
            glVertex2d(xOffset + quadLeft, yOffset + quadTop);
            glVertex2d(xOffset + quadRight, yOffset + quadTop);
            glVertex2d(xOffset + quadRight, yOffset + quadBottom);

            //Transformed quad
            glColor3f(1, 0, 0);
            glVertex2d(xOffset + point1.a[0][0], yOffset + point1.a[1][0]);
            glVertex2d(xOffset + point2.a[0][0], yOffset + point2.a[1][0]);
            glVertex2d(xOffset + point3.a[0][0], yOffset + point3.a[1][0]);
            glVertex2d(xOffset + point4.a[0][0], yOffset + point4.a[1][0]);
        glEnd();

        //Flush the buffer to the screen
        glFlush();
    }
}

int main(int argc, char** argv) {
    /* Get any input you need here */
    cout<<"\n\n\tENTER THE BOUNDARIES L R T B : ";
    cin>>quadLeft>>quadRight>>quadTop>>quadBottom;
    cout<<"\n\n\tMENU"
        <<"\n\t\t"<<"0. DISPLAY"
        <<"\n\t\t"<<"1. TRANSLATE"
        <<"\n\t\t"<<"2. ROTATE"
        <<"\n\t\t"<<"3. SCALE"
        <<"\n\t\t"<<"4. SHEAR"
        <<"\n\t\t"<<"5. REFLECT"
        <<"\n\t\t"<<"6. EXIT";

    //Initialize the main window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    glutCreateWindow(WINDOW_TITLE);
    glutDisplayFunc(myDisplay);

    //Initialize the canvas and start the main loop
    canvasInit();
    glutMainLoop();
    return 0;
}
```

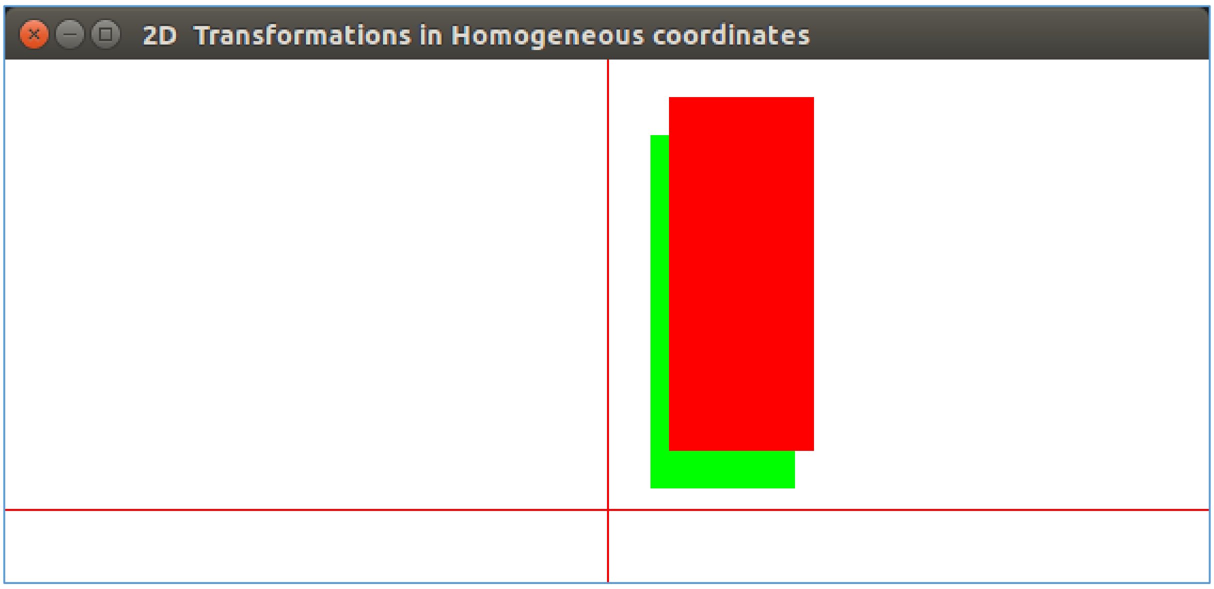
OUTPUT:

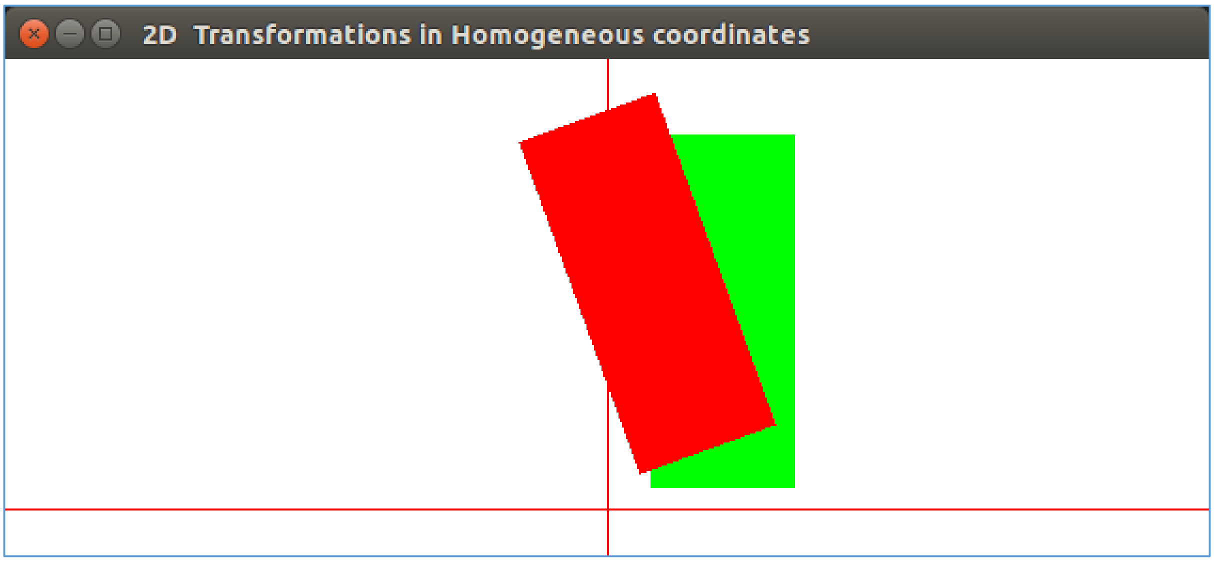


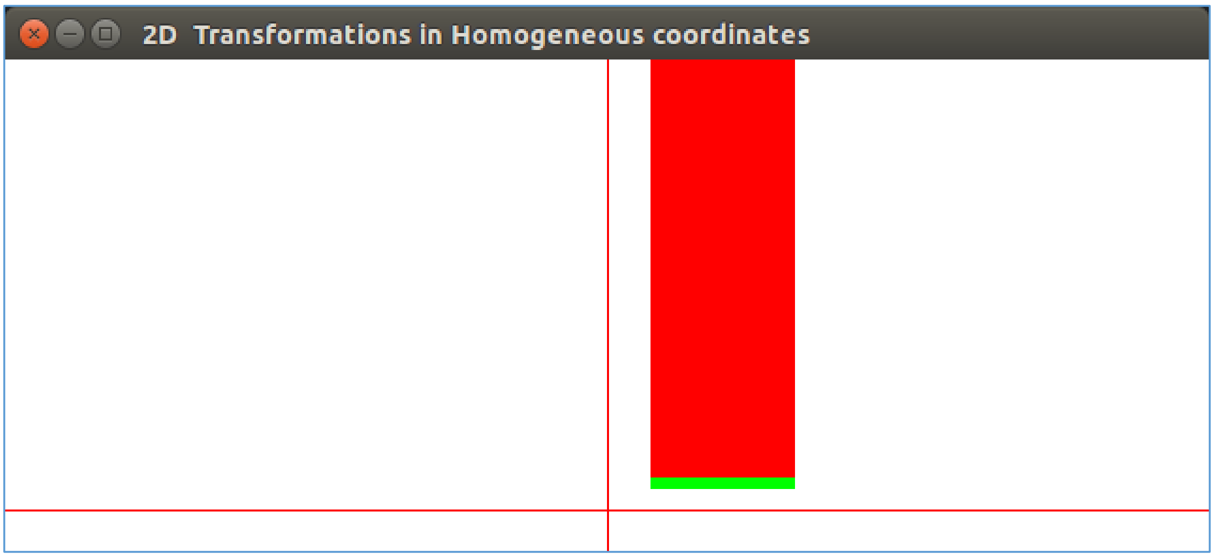
karthik@ubuntu: ~/Projects/Computer Graphics Lab/Source Code

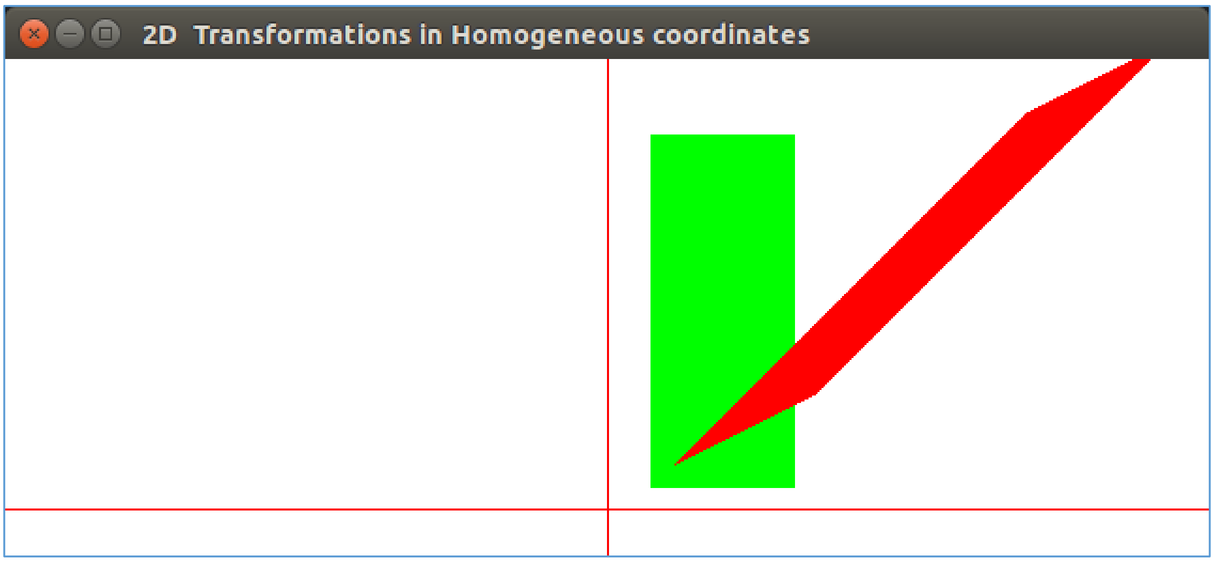
```
karthik@ubuntu:~/Projects/Computer Graphics Lab/Source Code$ g++ EX4\ -\ Transfo
rmations\ in\ 2D.cpp -lGLU -lGL -lglut
karthik@ubuntu:~/Projects/Computer Graphics Lab/Source Code$ ./a.out


        ENTER THE BOUNDARIES L R T B : 23 100 200 12


        MENU
                0. DISPLAY
                1. TRANSLATE
                2. ROTATE
                3. SCALE
                4. SHEAR
                5. REFLECT
                6. EXIT

        ENTER THE OPTION :
```

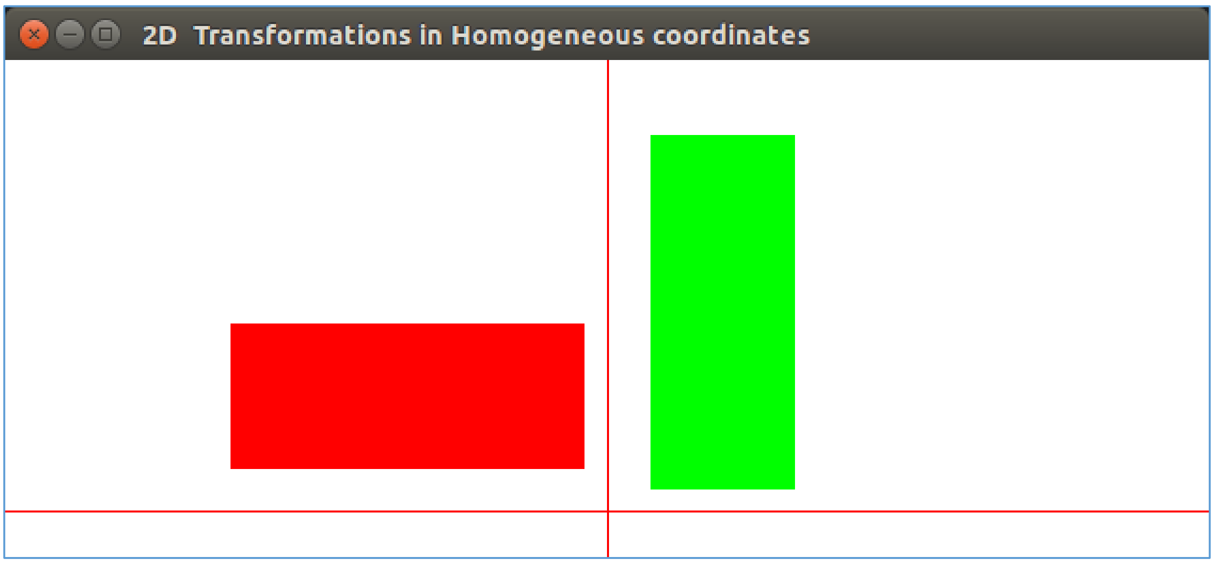

2D Transformations in Homogeneous coordinates


2D Transformations in Homogeneous coordinates


2D Transformations in Homogeneous coordinates







RESULT:

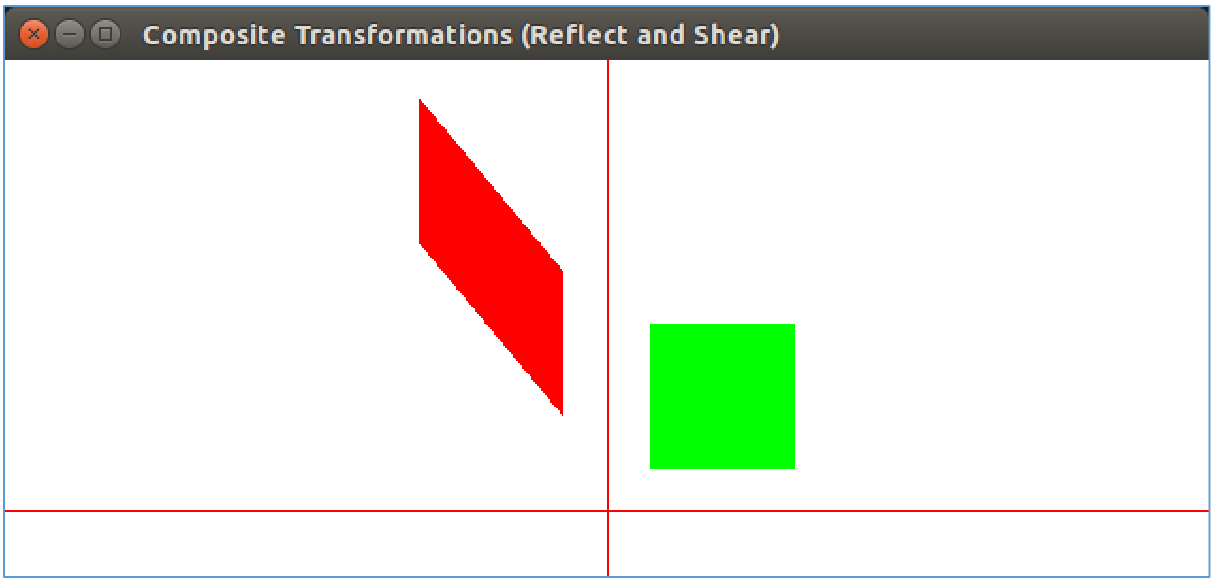
THE 2D TRANSFORMATIONS WERE SUCCESSFULLY DEMONSTRATED.

EX 5: COMPOSITE TRANSFORMATIONS

PROGRAM: REFLECT AND SHEAR

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

using namespace std;

//Window Parameters
const int WINDOW_WIDTH = 640;
const int WINDOW_HEIGHT = 480;
const char WINDOW_TITLE[] = "Composite Transformations (Reflect and Shear)";

//Canvas Parameters
const float PLANE_LEFT = 0.0;
const float PLANE_RIGHT = WINDOW_WIDTH;
const float PLANE_BOTTOM = 0.0;
const float PLANE_TOP = WINDOW_HEIGHT;
const int POINT_SIZE = 5;

/*Create the variables here*/
int quadLeft, quadRight, quadTop, quadBottom;
enum TRANSFORMATIONS { NORMAL, TRANSLATE, ROTATE, SCALE,  SHEAR, REFLECT};
int reflectAxis;
float shearX, shearY;

struct Matrix {
    float a[3][3];
    int m, n;
};


Matrix pointMatrix(int a, int b) {
    Matrix x = {{{a}, {b}, {1}}, 3, 1};
    return x;
}

Matrix multMatrix(Matrix mat1, Matrix mat2) {
    Matrix res = {{0}, mat1.m, mat2.n};

    //Multiplication of the matrices
    if (mat1.n == mat2.m) {
        for(int i=0; i<mat1.m; i++) {
            for(int j=0; j<mat2.n; j++) {
                for(int k=0; k<3; k++) {
                    res.a[i][j] += mat1.a[i][k] * mat2.a[k][j];
                }
            }
        }
    }

    return res;
}

void canvasInit() {
    //Initialize the canvas
    glClearColor(1, 1, 1, 0);
    glColor3f(1, 0, 0);
    glPointSize(POINT_SIZE);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(PLANE_LEFT, PLANE_RIGHT, PLANE_BOTTOM, PLANE_TOP);
}

Matrix transformPoint(Matrix point, int op, float p1, float p2) {
    Matrix trans = {
        {   /*The transformation Matrix*/
            {1, 0, 0},
            {0, 1, 0},
            {0, 0, 1}
        }, 3, 3
    };

    switch (op) {
        case 0: {
            //Display as it is without any transformation
            break;
        }
        case 1: {
            //Translation of the given point
            //p1 is the x-translation factor
            //p2 is the y-translation factor
            trans.a[0][2] = p1;
            trans.a[1][2] = p2;
            break;
        }
        case 2: {
            //Rotation of the given point
            //p1 is the rotation angle
            //p2 is a place holder
            float angle = p1 * M_PI / 180;
            trans.a[0][0] = cos(angle);
            trans.a[0][1] = -sin(angle);
            trans.a[1][0] = -trans.a[0][1];
            trans.a[1][1] = trans.a[0][0];
            break;
        }
        case 3: {
            //Scaling of the given point
            //p1 is the scaleX factor
            //p2 is the scaleY factor
            trans.a[0][0] = p1;
            trans.a[1][1] = p2;
            break;
        }
        case 4: {
            //Shearing of the given point
            //p1 is the shearX factor
            //p2 is the shearY factor
            trans.a[0][1] = p1;
            trans.a[1][0] = p2;
            break;
        }
        case 5: {
            //Reflection of the given point
            //p1 is the Reflection about point
            //0 is no Reflection
            //1 is Reflection about y-axis
            //3 is Reflection about x-axis
            float angle = (90 * M_PI / 180) * p1;
            trans.a[0][0] = cos(angle);
            trans.a[0][1] = -sin(angle);
            trans.a[1][0] = -trans.a[0][1];
            trans.a[1][1] = trans.a[0][0];
            break;
        }
    }

    return multMatrix(trans, point);
}

Matrix compositeTransform(int pointX, int pointY) {
    Matrix point = {{{pointX}, {pointY}, {1}}, 3, 1};

    //1. Reflected about the given axis
    //2. Sheared by the given factors
    Matrix result = transformPoint(
        transformPoint(point, REFLECT, reflectAxis, 0),
        SHEAR, shearX, shearY
    );

    return result;
}

void myDisplay() {
    //Clear the canvas
    glClear(GL_COLOR_BUFFER_BIT);

    //Find the mid-point offset
    int xOffset = WINDOW_WIDTH / 2;
    int yOffset = WINDOW_HEIGHT / 2;

    //Draw the Cartesian coordinates
    glBegin(GL_LINES);
        glVertex2d(0, yOffset);
        glVertex2d(WINDOW_WIDTH, yOffset);
        glVertex2d(xOffset, 0);
        glVertex2d(xOffset, WINDOW_HEIGHT);
    glEnd();

    //Transform the points and display them as a quad
    glBegin(GL_QUADS);
        //Transformed points
        Matrix point1 = compositeTransform(quadLeft, quadBottom);
        Matrix point2 = compositeTransform(quadLeft, quadTop);
        Matrix point3 = compositeTransform(quadRight, quadTop);
        Matrix point4 = compositeTransform(quadRight, quadBottom);

        //Original untouched quad
        glColor3f(0, 1, 0);
        glVertex2d(xOffset + quadLeft, yOffset + quadBottom);
        glVertex2d(xOffset + quadLeft, yOffset + quadTop);
        glVertex2d(xOffset + quadRight, yOffset + quadTop);
        glVertex2d(xOffset + quadRight, yOffset + quadBottom);

        //Transformed quad
        glColor3f(1, 0, 0);
        glVertex2d(xOffset + point1.a[0][0], yOffset + point1.a[1][0]);
        glVertex2d(xOffset + point2.a[0][0], yOffset + point2.a[1][0]);
        glVertex2d(xOffset + point3.a[0][0], yOffset + point3.a[1][0]);
        glVertex2d(xOffset + point4.a[0][0], yOffset + point4.a[1][0]);
    glEnd();
    //Flush the buffer to the screen
    glFlush();
}

int main(int argc, char** argv) {
    /* Get any input you need here */
    cout<<"\n\n\tENTER THE BOUNDARIES L R T B : ";
    cin>>quadLeft>>quadRight>>quadTop>>quadBottom;
    cout<<"\n\n\tENTER THE AXIS TO REFLECT : ";
    cin>>reflectAxis;
    cout<<"\n\n\tENTER THE SHEAR FACTORS : ";
    cin>>shearX>>shearY;

    //Initialize the main window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    glutCreateWindow(WINDOW_TITLE);
    glutDisplayFunc(myDisplay);

    //Initialize the canvas and start the main loop
    canvasInit();
    glutMainLoop();
    return 0;
}
```

OUTPUT:

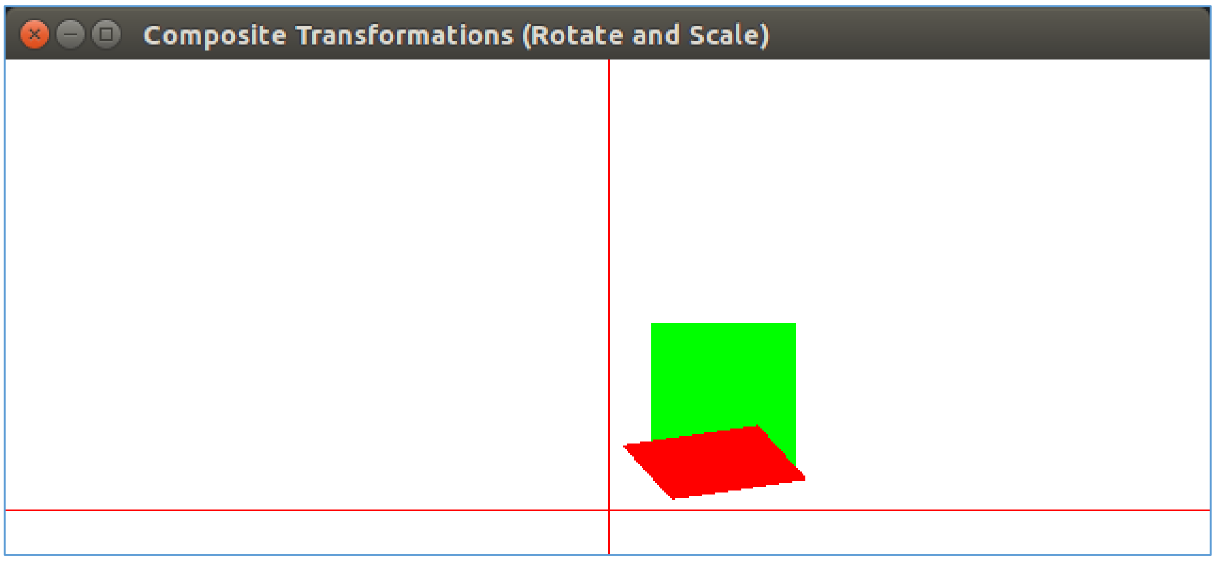


PROGRAM: ROTATE AND SCALE

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

using namespace std;

//Window Parameters
const int WINDOW_WIDTH = 640;
const int WINDOW_HEIGHT = 480;
const char WINDOW_TITLE[] = "Composite Transformations (Rotate and Scale)";

//Canvas Parameters
const float PLANE_LEFT = 0.0;
const float PLANE_RIGHT = WINDOW_WIDTH;
const float PLANE_BOTTOM = 0.0;
const float PLANE_TOP = WINDOW_HEIGHT;
const int POINT_SIZE = 5;

/*Create the variables here*/
int quadLeft, quadRight, quadTop, quadBottom;
enum TRANSFORMATIONS { NORMAL, TRANSLATE, ROTATE, SCALE,  SHEAR, REFLECT};
float angle, scaleX, scaleY, transX, transY;

struct Matrix {
    float a[3][3];
    int m, n;
};

Matrix multMatrix(Matrix mat1, Matrix mat2) {
    Matrix res = {{0}, mat1.m, mat2.n};

    //Multiplication of the matrices
    if (mat1.n == mat2.m) {
        for(int i=0; i<mat1.m; i++) {
            for(int j=0; j<mat2.n; j++) {
                for(int k=0; k<3; k++) {
                    res.a[i][j] += mat1.a[i][k] * mat2.a[k][j];
                }
            }
        }
    }

    return res;
}

void canvasInit() {
    //Initialize the canvas
    glClearColor(1, 1, 1, 0);
    glColor3f(1, 0, 0);
    glPointSize(POINT_SIZE);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(PLANE_LEFT, PLANE_RIGHT, PLANE_BOTTOM, PLANE_TOP);
}

Matrix transformPoint(Matrix point, int op, float p1, float p2) {
    Matrix trans = {
        {   /*The transformation Matrix*/
            {1, 0, 0},
            {0, 1, 0},
            {0, 0, 1}
        }, 3, 3
    };

    switch (op) {
        case 0: {
            //Display as it is without any transformation
            break;
        }
        case 1: {
            //Translation of the given point
            //p1 is the x-translation factor
            //p2 is the y-translation factor
            trans.a[0][2] = p1;
            trans.a[1][2] = p2;
            break;
        }
        case 2: {
            //Rotation of the given point
            //p1 is the rotation angle
            //p2 is a place holder
            float angle = p1 * M_PI / 180;
            trans.a[0][0] = cos(angle);
            trans.a[0][1] = -sin(angle);
            trans.a[1][0] = -trans.a[0][1];
            trans.a[1][1] = trans.a[0][0];
            break;
        }
        case 3: {
            //Scaling of the given point
            //p1 is the scaleX factor
            //p2 is the scaleY factor
            trans.a[0][0] = p1;
            trans.a[1][1] = p2;
            break;
        }
        case 4: {
            //Shearing of the given point
            //p1 is the shearX factor
            //p2 is the shearY factor
            trans.a[0][1] = p1;
            trans.a[1][0] = p2;
            break;
        }
        case 5: {
            //Reflection of the given point
            //p1 is the Reflection about point
            //0 is no Reflection
            //1 is Reflection about y-axis
            //3 is Reflection about x-axis
            float angle = (90 * M_PI / 180) * p1;
            trans.a[0][0] = cos(angle);
            trans.a[0][1] = -sin(angle);
            trans.a[1][0] = -trans.a[0][1];
            trans.a[1][1] = trans.a[0][0];
            break;
        }
    }

    return multMatrix(trans, point);
}
```

```
Matrix compositeTransform(int pointX, int pointY) {
    Matrix point = {{{pointX}, {pointY}, {1}}, 3, 1};

    //1. First translated to origin about the given point
    //2. Rotated about the origin
    //3. Translated back to the given point
    //4. Scaled by the given scale factors
    Matrix result = transformPoint(
        transformPoint(
            transformPoint(
                transformPoint(point, TRANSLATE, -transX, -transY),
                ROTATE, angle, 0
            ),
            TRANSLATE, transX, transY
        ),
        SCALE, scaleX, scaleY
    );

    return result;
}

void myDisplay() {
    //Clear the canvas
    glClear(GL_COLOR_BUFFER_BIT);

    //Find the mid-point offset
    int xOffset = WINDOW_WIDTH / 2;
    int yOffset = WINDOW_HEIGHT / 2;

    //Draw the Cartesian coordinates
    glBegin(GL_LINES);
        glVertex2d(0, yOffset);
        glVertex2d(WINDOW_WIDTH, yOffset);
        glVertex2d(xOffset, 0);
        glVertex2d(xOffset, WINDOW_HEIGHT);
    glEnd();

    //Transform the points and display them as a quad
    glBegin(GL_QUADS);
        //Transformed points
        Matrix point1 = compositeTransform(quadLeft, quadBottom);
        Matrix point2 = compositeTransform(quadLeft, quadTop);
        Matrix point3 = compositeTransform(quadRight, quadTop);
        Matrix point4 = compositeTransform(quadRight, quadBottom);

        //Original untouched quad
        glColor3f(0, 1, 0);
        glVertex2d(xOffset + quadLeft, yOffset + quadBottom);
        glVertex2d(xOffset + quadLeft, yOffset + quadTop);
        glVertex2d(xOffset + quadRight, yOffset + quadTop);
        glVertex2d(xOffset + quadRight, yOffset + quadBottom);

        //Transformed quad
        glColor3f(1, 0, 0);
        glVertex2d(xOffset + point1.a[0][0], yOffset + point1.a[1][0]);
        glVertex2d(xOffset + point2.a[0][0], yOffset + point2.a[1][0]);
        glVertex2d(xOffset + point3.a[0][0], yOffset + point3.a[1][0]);
        glVertex2d(xOffset + point4.a[0][0], yOffset + point4.a[1][0]);
    glEnd();

    //Flush the buffer to the screen
    glFlush();
}

int main(int argc, char** argv) {
    /* Get any input you need here */
    cout<<"\n\n\tENTER THE BOUNDARIES L R T B : ";
    cin>>quadLeft>>quadRight>>quadTop>>quadBottom;
    cout<<"\n\tENTER THE ANGLE AND SCALEX AND SCALEY : ";
    cin>>angle>>scaleX>>scaleY;
    cout<<"\n\tENTER THE FIEXD POINT : ";
    cin>>transX>>transY;

    //Initialize the main window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    glutCreateWindow(WINDOW_TITLE);
    glutDisplayFunc(myDisplay);

    //Initialize the canvas and start the main loop
    canvasInit();
    glutMainLoop();
    return 0;
}
```

OUTPUT:



RESULT:

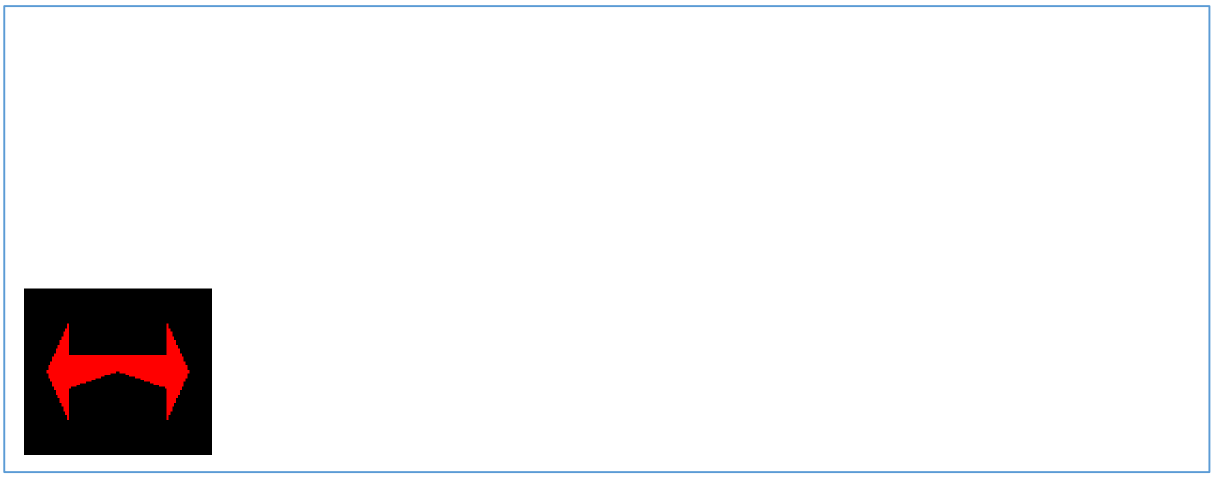
THE COMPOSITE TRANSFORMATION WAS SUCCESSFULLY DEMONSTRATED.

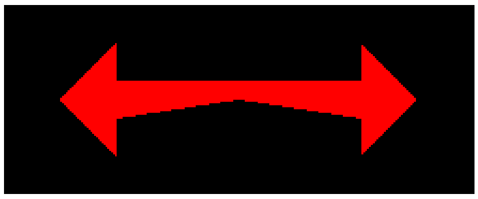
EX 6: WINDOW TO VIEWPORT TRANSFORMATION

PROGRAM:

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

using namespace std;

//Window Parameters
const int WINDOW_WIDTH = 640;
const int WINDOW_HEIGHT = 480;
const char WINDOW_TITLE[] = "Window to Viewport animation";

//Canvas Parameters
const float PLANE_LEFT = 0.0;
const float PLANE_RIGHT = WINDOW_WIDTH;
const float PLANE_BOTTOM = 0.0;
const float PLANE_TOP = WINDOW_HEIGHT;
const int POINT_SIZE = 5;

/*Create the variables here*/
float viewLeft, viewRight, viewTop, viewBottom;
float winLeft, winRight, winTop, winBottom;
enum TRANSFORMATIONS { NORMAL, TRANSLATE, ROTATE, SCALE,  SHEAR, REFLECT};

struct Matrix {
    float a[3][3];
    int m, n;
};

Matrix multMatrix(Matrix mat1, Matrix mat2) {
    Matrix res = {{0}, mat1.m, mat2.n};

    //Multiplication of the matrices
    if (mat1.n == mat2.m) {
        for(int i=0; i<mat1.m; i++) {
            for(int j=0; j<mat2.n; j++) {
                for(int k=0; k<3; k++) {
                    res.a[i][j] += mat1.a[i][k] * mat2.a[k][j];
                }
            }
        }
    }

    return res;
}

void canvasInit() {
    //Initialize the canvas
    glClearColor(1, 1, 1, 0);
    glColor3f(1, 0, 0);
    glPointSize(POINT_SIZE);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(PLANE_LEFT, PLANE_RIGHT, PLANE_BOTTOM, PLANE_TOP);
}

Matrix transformPoint(int x, int y, int op, float p1, float p2) {
    Matrix trans = {
        {   /*The transformation Matrix*/
            {1, 0, 0},
            {0, 1, 0},
            {0, 0, 1}
        }, 3, 3
    };
    Matrix point = {
        {
            /*The point in homogeneous coordinates*/
            {x},
            {y},
            {1}
        }, 3, 1
    };

    switch (op) {
        case 0: {
            //Display as it is without any transformation
            break;
        }
        case 1: {
            //Translation of the given point
            //p1 is the x-translation factor
            //p2 is the y-translation factor
            trans.a[0][2] = p1;
            break;
        }
            trans.a[1][2] = p2;
        case 2: {
            //Rotation of the given point
            //p1 is the rotation angle
            //p2 is a place holder
            float angle = p1 * M_PI / 180;
            trans.a[0][0] = cos(angle);
            trans.a[0][1] = -sin(angle);
            trans.a[1][0] = -trans.a[0][1];
            trans.a[1][1] = trans.a[0][0];
            break;
        }
        case 3: {
            //Scaling of the given point
            //p1 is the scaleX factor
            //p2 is the scaleY factor
            trans.a[0][0] = p1;
            trans.a[1][1] = p2;
            break;
        }
        case 4: {
            //Shearing of the given point
            //p1 is the shearX factor
            //p2 is the shearY factor
            trans.a[0][1] = p1;
            trans.a[1][0] = p2;
            break;
        }
        case 5: {
            //Reflection of the given point
            //p1 is the Reflection about point
            //0 is no Reflection
            //1 is Reflection about y-axis
            //3 is Reflection about x-axis
            float angle = (90 * M_PI / 180) * p1;
            trans.a[0][0] = cos(angle);
            trans.a[0][1] = -sin(angle);
            trans.a[1][0] = -trans.a[0][1];
            trans.a[1][1] = trans.a[0][0];
            break;
        }
    }

    return multMatrix(trans, point);
}

void myDisplay() {
    while(true) {
        //Clear the canvas
        glClear(GL_COLOR_BUFFER_BIT);

        //Transform Parameters
        int op;
        float p1 = (viewRight - viewLeft) / (winRight - winLeft);
        float p2 = (viewTop - viewBottom) / (winTop - winBottom);
        int xOffset = 10;
        int yOffset = 10;

        //Get user data
        cout<<"\n\n\tDISPLAY VIEW PORT : ";
        cin>>op;
        if (op == -1) { break; }
        op = op == 0 ? NORMAL : SCALE;

        //Draw the viewport or the window based on the user selection
        glBegin(GL_QUADS);
            glColor3f(0,0,0);
            if(op) {
                glVertex2d(xOffset + viewLeft, yOffset + viewBottom);
                glVertex2d(xOffset + viewLeft, yOffset + viewTop);
                glVertex2d(xOffset + viewRight, yOffset + viewTop);
                glVertex2d(xOffset + viewRight, yOffset + viewBottom);
            } else {
                glVertex2d(xOffset + winLeft, yOffset + winBottom);
                glVertex2d(xOffset + winLeft, yOffset + winTop);
                glVertex2d(xOffset + winRight, yOffset + winTop);
                glVertex2d(xOffset + winRight, yOffset + winBottom);
            }
        glEnd();

        //Transformed points to viewport or window based on user selection
        Matrix point1 = transformPoint(30, 50, op, p1 , p2);
        Matrix point2 = transformPoint(60, 80, op, p1, p2);
        Matrix point3 = transformPoint(60, 20, op, p1, p2);
        Matrix point4 = transformPoint(220, 50, op, p1 , p2);
        Matrix point5 = transformPoint(190, 80, op, p1, p2);
        Matrix point6 = transformPoint(190, 20, op, p1, p2);
        Matrix point7 = transformPoint(60, 40, op, p1, p2);
        Matrix point8 = transformPoint(60, 60, op, p1 , p2);
        Matrix point9 = transformPoint(190, 40, op, p1, p2);
        Matrix point10 = transformPoint(190, 60, op, p1, p2);

        //Transform the points and display them as a figure
        glBegin(GL_TRIANGLES);
            glColor3f(1,0,0);
            glVertex2d(xOffset + point1.a[0][0], yOffset + point1.a[1][0]);
            glVertex2d(xOffset + point2.a[0][0], yOffset + point2.a[1][0]);
            glVertex2d(xOffset + point3.a[0][0], yOffset + point3.a[1][0]);
            glVertex2d(xOffset + point4.a[0][0], yOffset + point4.a[1][0]);
            glVertex2d(xOffset + point5.a[0][0], yOffset + point5.a[1][0]);
            glVertex2d(xOffset + point6.a[0][0], yOffset + point6.a[1][0]);
        glEnd();
        glBegin(GL_QUADS);
            glVertex2d(xOffset + point7.a[0][0], yOffset + point7.a[1][0]);
            glVertex2d(xOffset + point8.a[0][0], yOffset + point8.a[1][0]);
            glVertex2d(xOffset + point9.a[0][0], yOffset + point9.a[1][0]);
            glVertex2d(xOffset + point10.a[0][0], yOffset + point10.a[1][0]);
        glEnd();

        //Flush the buffer to the screen
        glFlush();
    }
}

int main(int argc, char** argv) {
    /* Get any input you need here */
    cout<<"\n\n\tENTER THE VIEW BOUNDARIES L R T B : ";
    cin>>viewLeft>>viewRight>>viewTop>>viewBottom;
    winLeft = 0;
    winRight = 250;
    winTop = 100;
    winBottom = 0;

    //Initialize the main window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    glutCreateWindow(WINDOW_TITLE);
    glutDisplayFunc(myDisplay);

    //Initialize the canvas and start the main loop
    canvasInit();
    glutMainLoop();
    return 0;
}
```

OUTPUT:





RESULT:

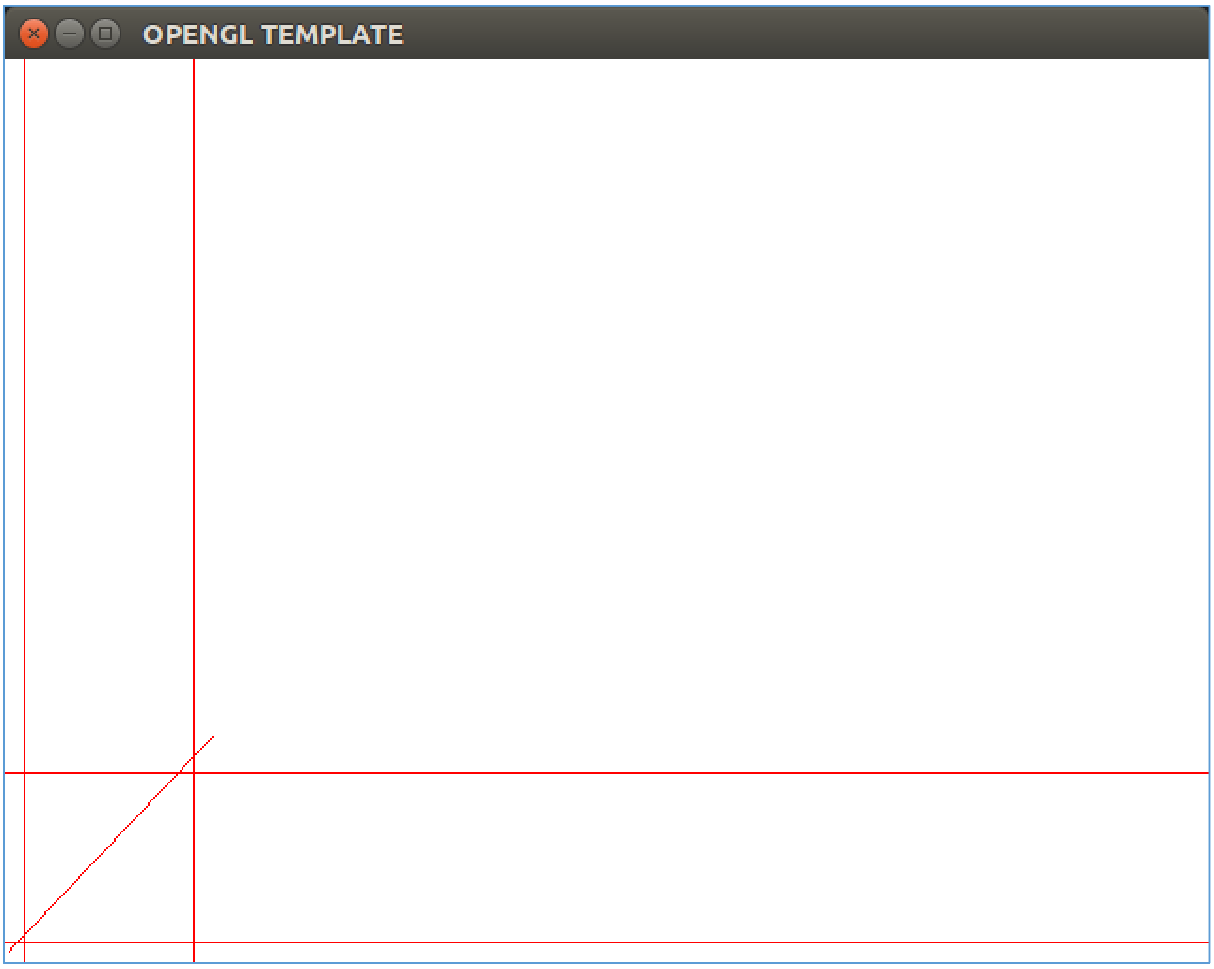
THE TRANSFORMATION WAS SUCCESSFUL.

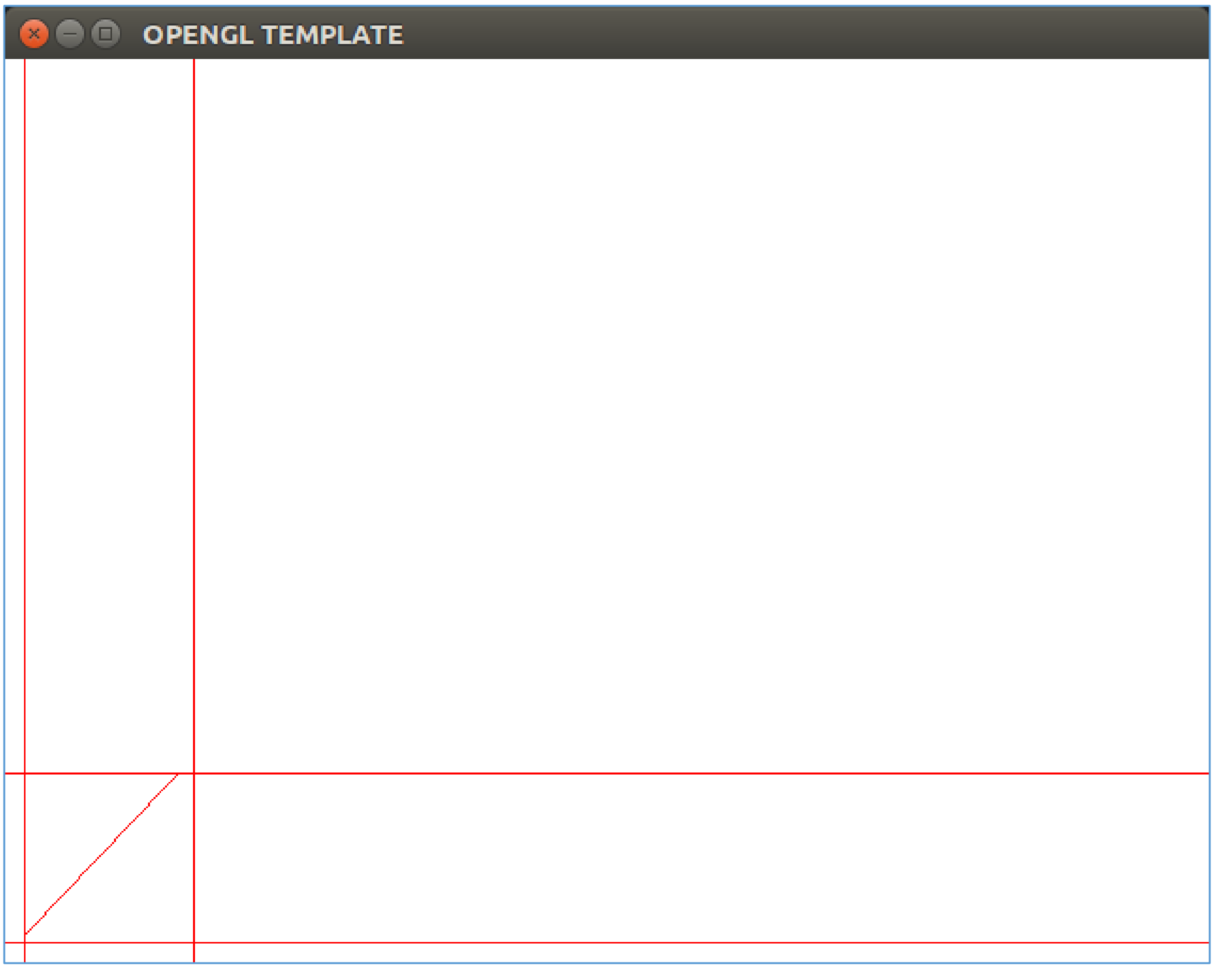
EX 7: LINE CLIPPING

PROGRAM:

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

using namespace std;

//Window Parameters
const int WINDOW_WIDTH = 640;
const int WINDOW_HEIGHT = 480;
const char WINDOW_TITLE[] = "OPENGL TEMPLATE";

//Canvas Parameters
const float PLANE_LEFT = 0.0;
const float PLANE_RIGHT = WINDOW_WIDTH;
const float PLANE_BOTTOM = 0.0;
const float PLANE_TOP = WINDOW_HEIGHT;
const int POINT_SIZE = 5;

/*Create the variables here*/
int winLeft, winRight, winTop, winBottom;
float px[2], py[2];
float fx[2], fy[2];
bool tbrl[4];

void canvasInit() {
    //Initialize the canvas
    glClearColor(1, 1, 1, 0);
    glColor3f(1, 0, 0);
    glPointSize(POINT_SIZE);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(PLANE_LEFT, PLANE_RIGHT, PLANE_BOTTOM, PLANE_TOP);
}

bool setTBRL(int x, int y) {
    tbrl[0] = y > winTop;
    tbrl[1] = y < winBottom;
    tbrl[2] = x > winRight;
    tbrl[3] = x < winLeft;

    for(int i=0; i<4; i++) {
        if(tbrl[i]) {
            return false;
        }
    }
    return true;
}

void myDisplay() {
    //Compute the slope of the line
    float m = (py[1] - py[0]) / (px[1] - px[0]);

    //Do the calculations and display
    for(int i = 0; i<2; i++) {
        for(int j=0; j<4; j++) {
            //Clear the canvas
            glClear(GL_COLOR_BUFFER_BIT);
            //Draw the clipping window
            glBegin(GL_LINES);
                glVertex2d(0, winTop);
                glVertex2d(WINDOW_WIDTH, winTop);
                glVertex2d(0, winBottom);
                glVertex2d(WINDOW_WIDTH, winBottom);
                glVertex2d(winLeft, 0);
                glVertex2d(winLeft, WINDOW_HEIGHT);
                glVertex2d(winRight, 0);
                glVertex2d(winRight, WINDOW_HEIGHT);
            glEnd();

            //Draw the clipped line
            glBegin(GL_LINES);
                glVertex2d(fx[0], fy[0]);
                glVertex2d(fx[1], fy[1]);
            glEnd();

            //Flush the buffer to the screen
            glFlush();

            //If it is a trivial accept then break
            if(setTBRL(fx[i], fy[i])) { break; }

            //If there is an intersection possible
            //Then find the point of intersection
            if(tbrl[j] == true) {
                switch (j) {
                    case 0:{
                        fx[i] = px[i] + (winTop - py[i]) / m;
                        fy[i] = winTop;
                        break;
                    }
                    case 1: {
                        fx[i] = px[i] + (winBottom - py[i]) / m;
                        fy[i] = winBottom;
                        break;
                    }
                    case 2: {
                        fx[i] = winRight;
                        fy[i] = py[i] + m * (winRight - px[i]);
                        break;
                    }
                    case 3: {
                        fx[i] = winLeft;
                        fy[i] = py[i] + m * (winLeft - px[i]);
                        break;
                    }
                }

                //Wait for user input
                int a;
                cin>>a;
            }
        }
    }
}

int main(int argc, char** argv) {
    /* Get any input you need here */
    cout<<"\n\n\tENTER THE TBRL : ";
    cin>>winTop>>winBottom>>winRight>>winLeft;
    cout<<"\n\n\tENTER THE POINTS : ";
    cin>>px[0]>>py[0]>>px[1]>>py[1];
    fx[0] = px[0];
    fy[0] = py[0];
    fx[1] = px[1];
    fy[1] = py[1];

    //Initialize the main window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    glutCreateWindow(WINDOW_TITLE);
    glutDisplayFunc(myDisplay);

    //Initialize the canvas and start the main loop
    canvasInit();
    glutMainLoop();
    return 0;
}
```

OUTPUT:





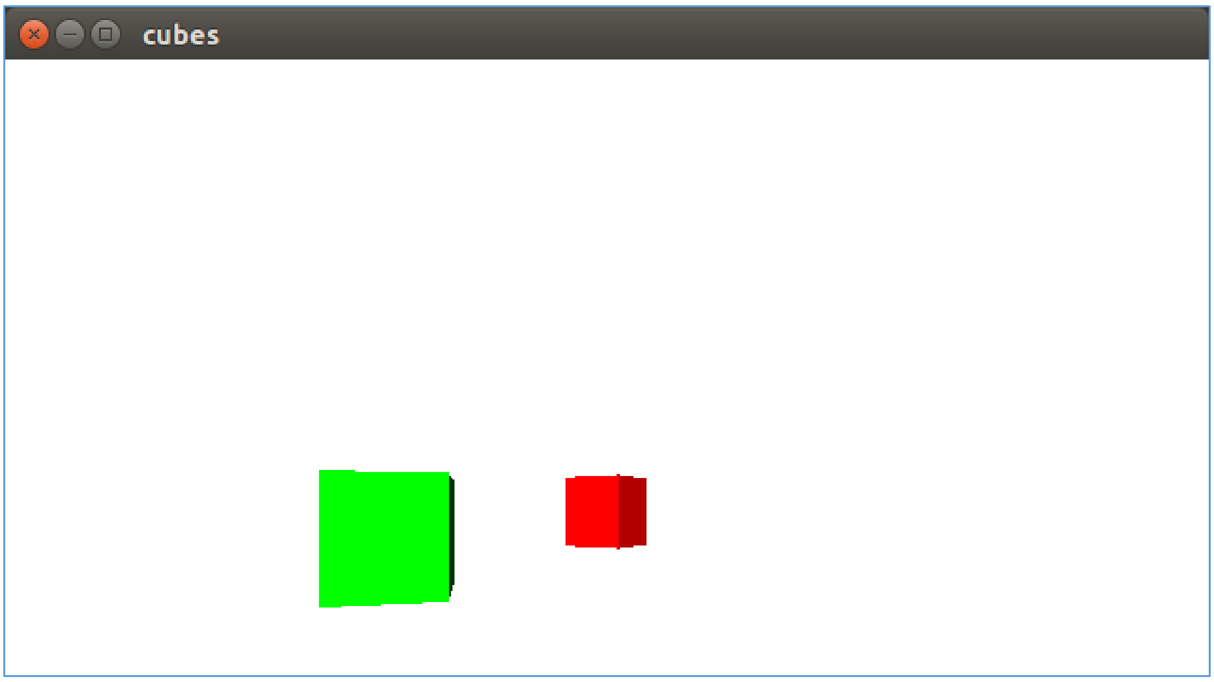
RESULT:

EX 8: TRANSFORMATIONS IN 3D

PROGRAM:

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

void init() {
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_COLOR_MATERIAL);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(80, 1.5, 1, 100);
    glClearColor(1, 1, 1, 0);
}

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

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    glTranslatef(0, 0, -40);
    glRotatef(-30, 0, 0.1, 0);
    glColor3ub(255, 0, 0);
    glutSolidCube(5);

    glTranslatef(0, -1, 20);
    glRotatef(-45, 0, 0.1, 0);
    glColor3ub(0, 255, 0);
    glutSolidCube(5);

    glutSwapBuffers();
}

int main(int argc, char *argv[]) {
    glutInit(&argc, argv);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(20, 30);
    glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);
    glutCreateWindow("cubes");

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

OUTPUT:



RESULT:

THE 3D TRANSFORMATIONS WERE DEMONSTRATED

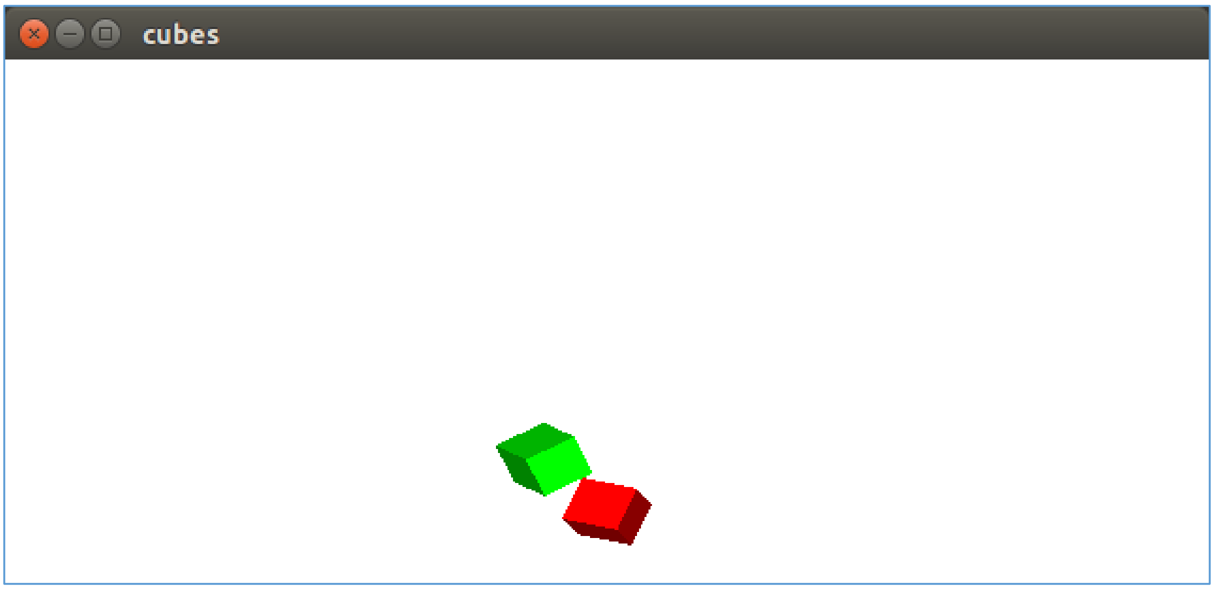
EX 9: 3D PROJECTIONS

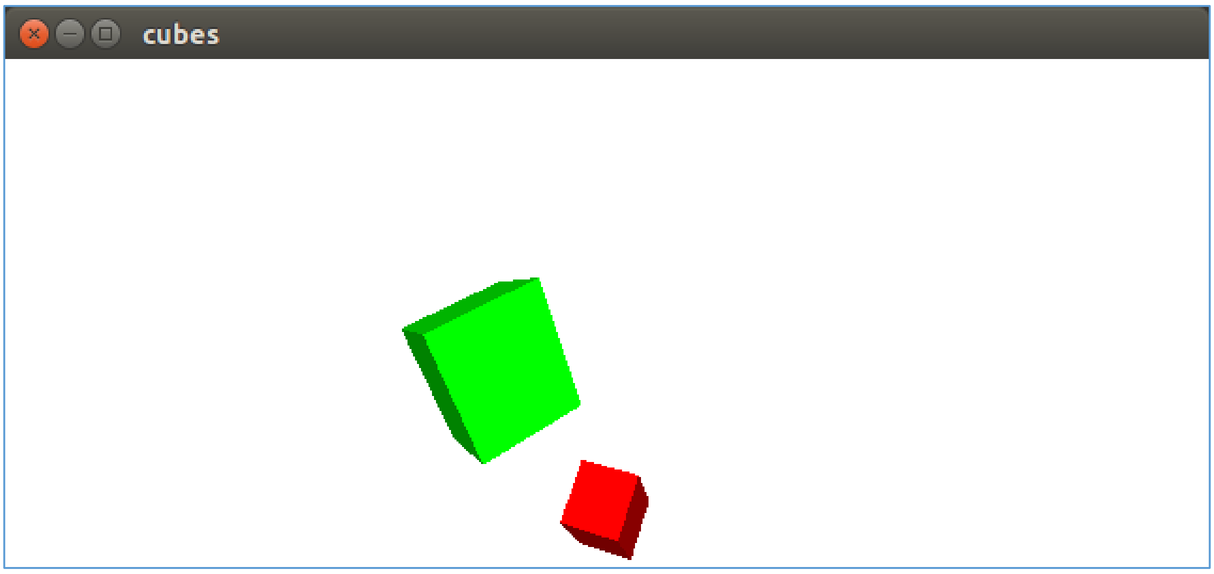
PROGRAM:

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

void init() {
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_COLOR_MATERIAL);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    gluPerspective(80, 1.5, 1, 100);
    //glOrtho(50, -50, 50, -50);

    glClearColor(1, 1, 1, 0);
}

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

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    glTranslatef(0, 0, -40);
    glRotatef(-30, 0, 0.1, 0);
    glColor3ub(255, 0, 0);
    glutSolidCube(5);

    glTranslatef(0, -1, 20);
    glRotatef(-45, 0, 0.1, 0);
    glColor3ub(0, 255, 0);
    glutSolidCube(5);

    glutSwapBuffers();
}

int main(int argc, char *argv[]) {
    glutInit(&argc, argv);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(20, 30);
    glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);
    glutCreateWindow("cubes");

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

OUTPUT:

THE LINE CLIPPING ALGORITHM WAS IMPLEMENTED





RESULT:

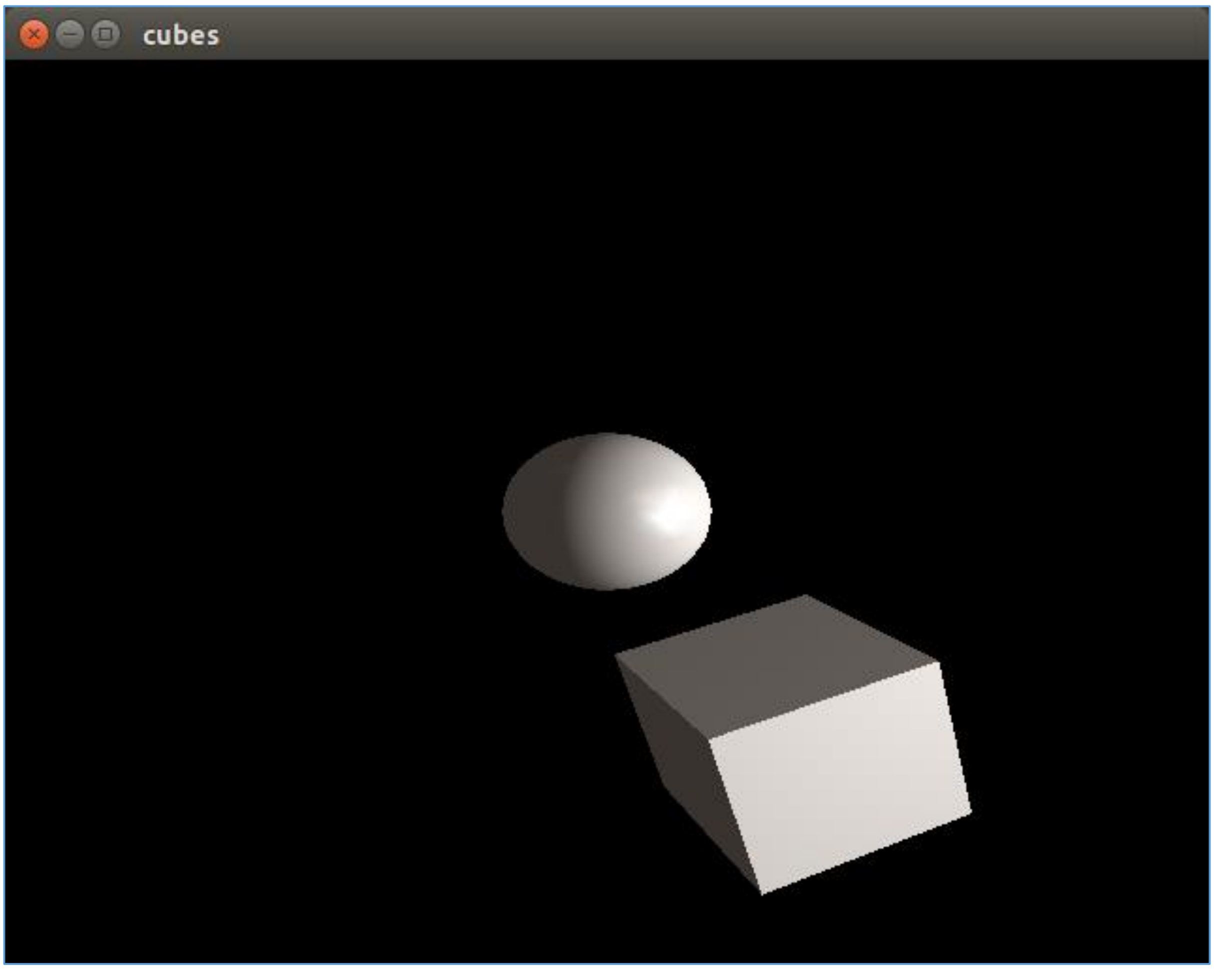
THE 3D PROJECTIONS WERE SUCCESSFULLY DEMONSTRATED

EX10: 3D SCENES

PROGRAM:

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

float matSpecular[] = { 1, 1, 1, 1 };
float matShininess[] = { 70 };
float lightPos[] = { 20, 0, 0, 1 };
float lightAmb[] = { .9, .8, .7, 1 };

void init() {
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);

    glShadeModel(GL_SMOOTH);
    glMaterialfv(GL_FRONT, GL_SPECULAR, matSpecular);
    glMaterialfv(GL_FRONT, GL_SHININESS, matShininess);

    glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
    glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmb);
}

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

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60, 1, 1, 100);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    glTranslatef(0, 0, -10);
    glRotatef(-45, 0.1, 0.1, 0.1);
    glutSolidSphere(1, 40, 16);

    glTranslatef(0, 0, -10);
    glRotatef(-45, 0.1, 0.1, 0.1);
    glutSolidCube(4);

    glutSwapBuffers();
}

int main(int argc, char *argv[]) {
    glutInit(&argc, argv);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(20, 30);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE);
    glutCreateWindow("cubes");

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

OUTPUT:

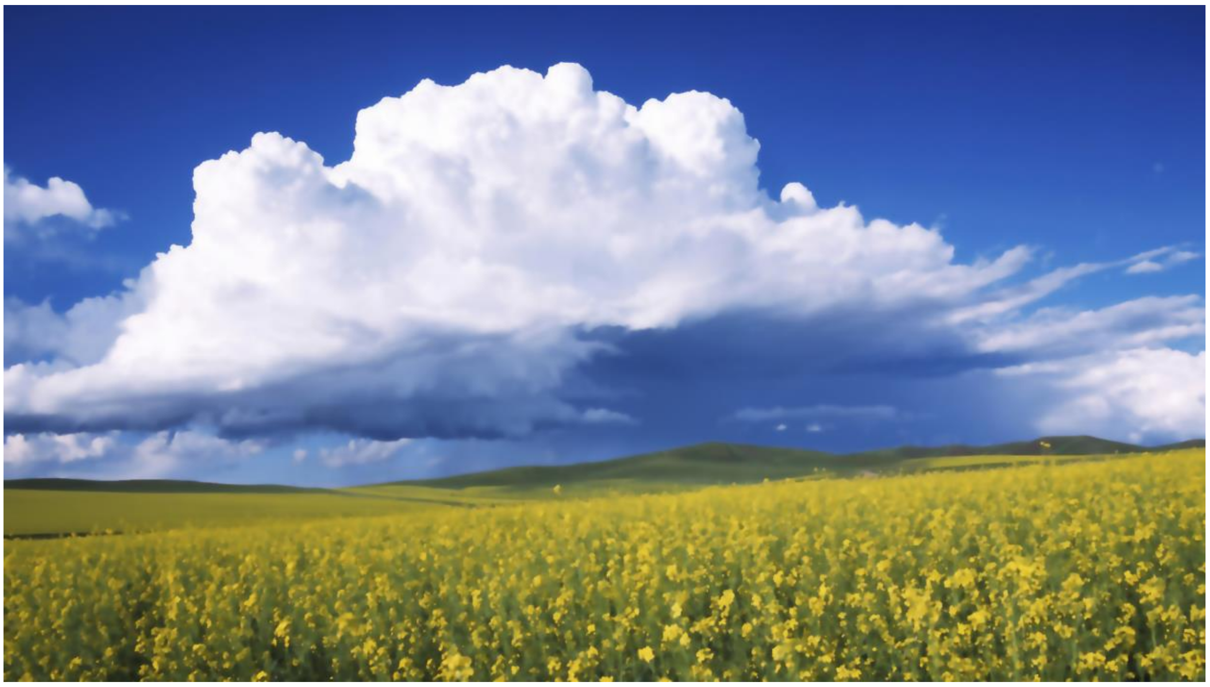


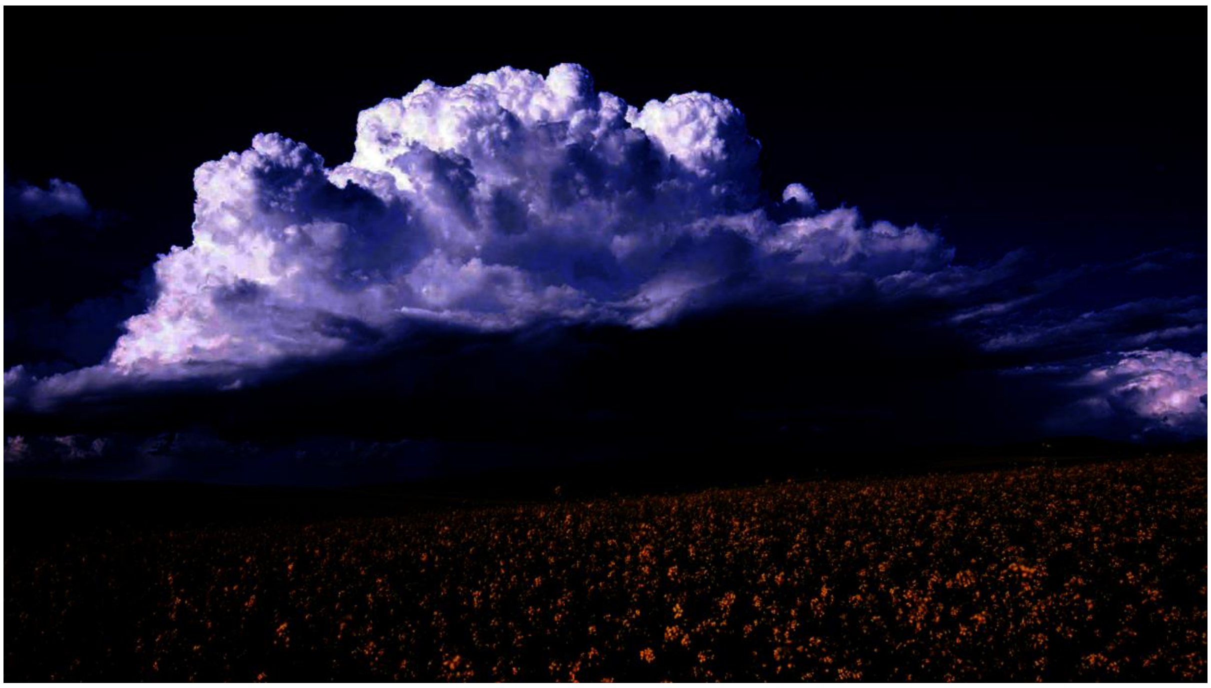
RESULT:

THE 3D SCENE WAS SUCCESSFULLY CREATED.

EX 11: IMAGE MANIPULATION USING GIMP

OUTPUT:

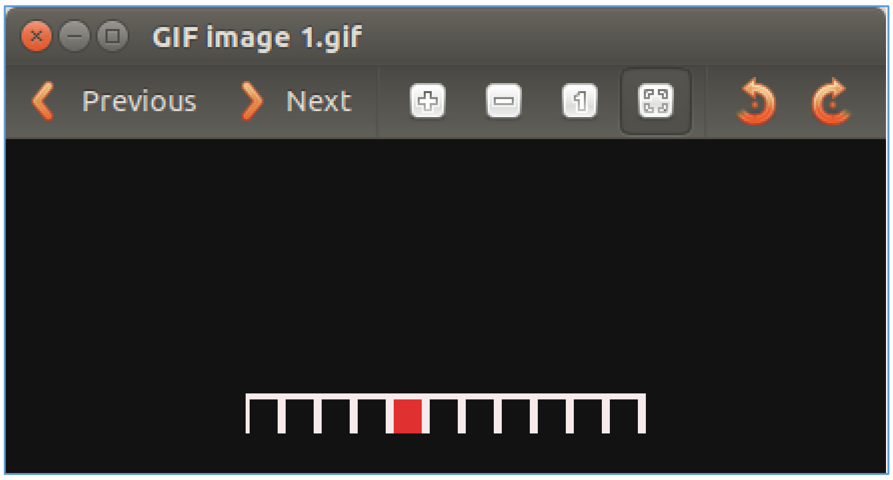


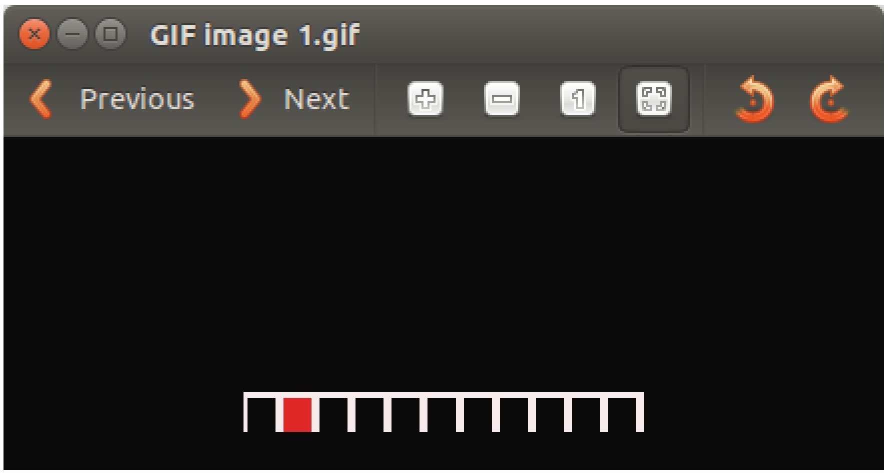


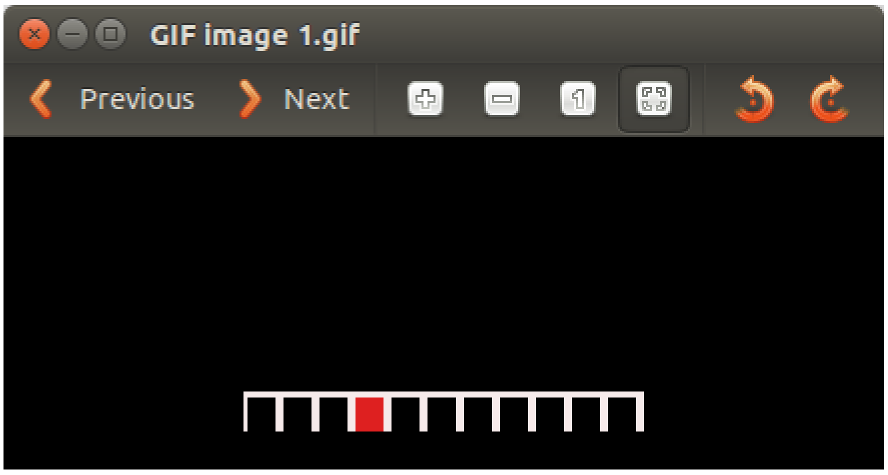
RESULT:

THE IMAGE WAS MANIPULATED USING GIMP.

EX 12: GIF IMAGE





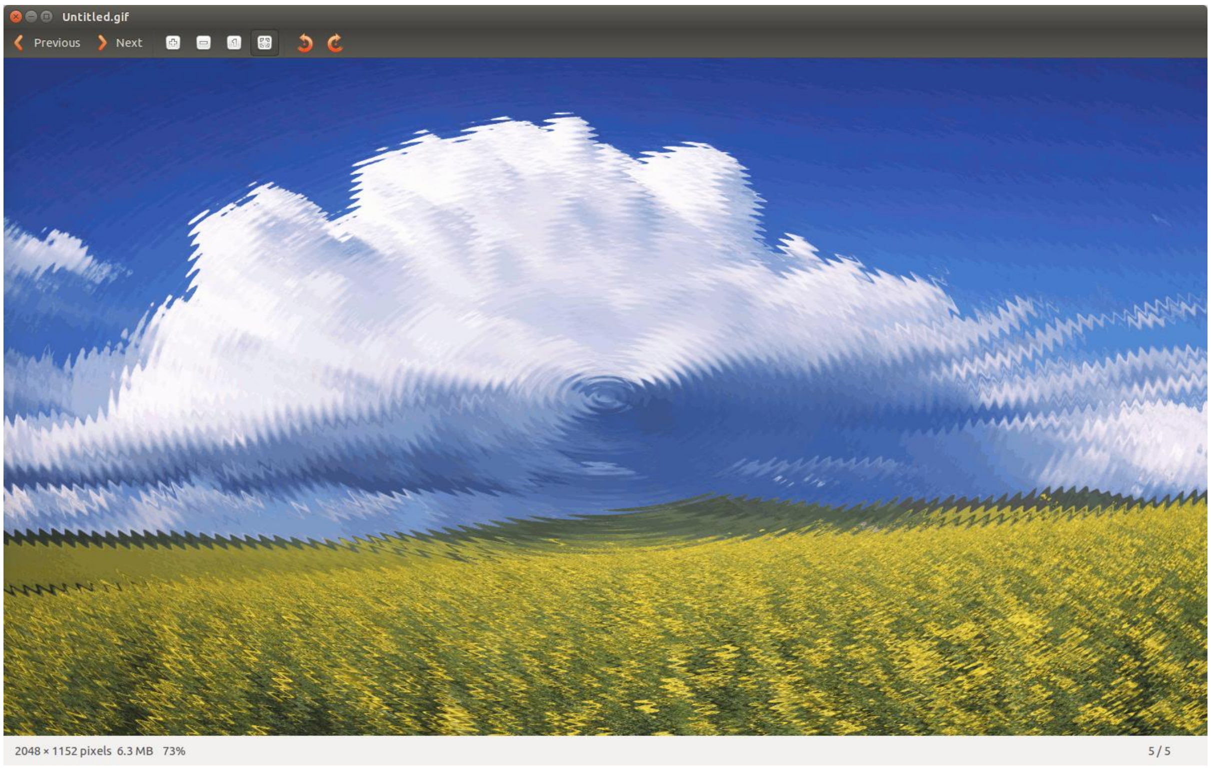


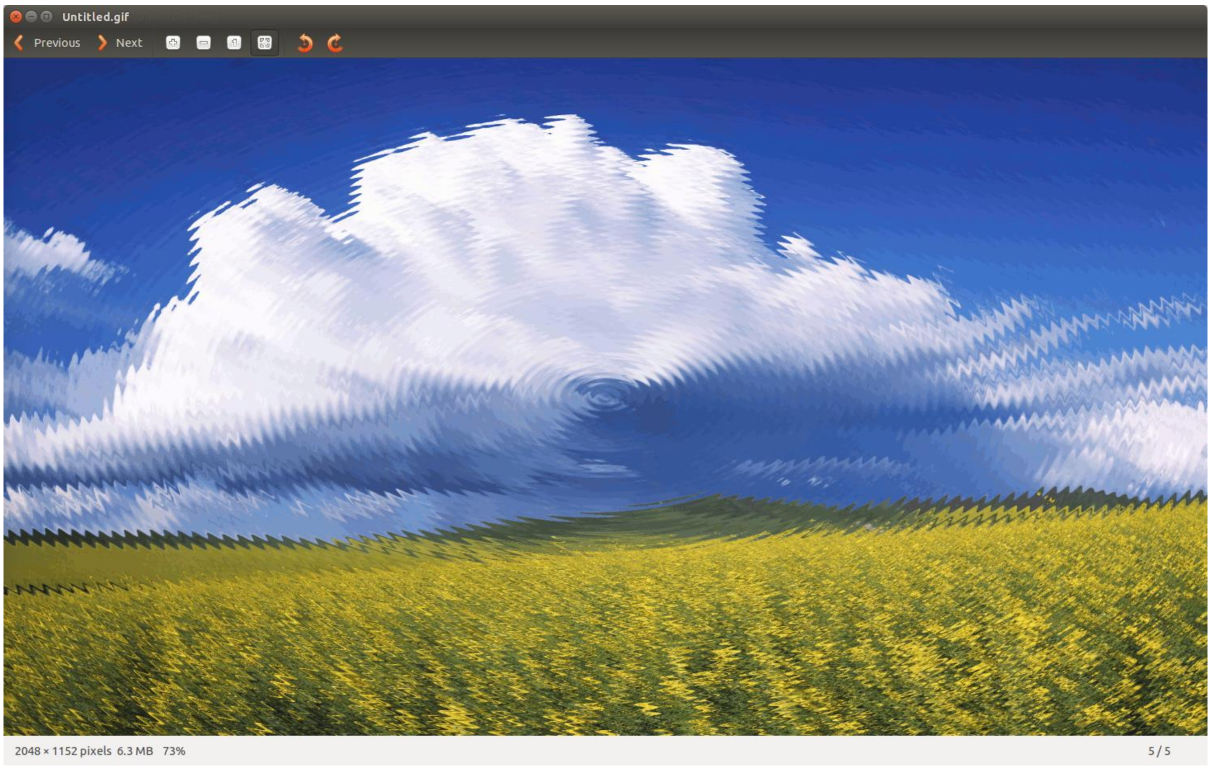
RESULT:

THE GIF ANIMATION WAS CREATED

EX 13: CREATING ANIMATION

OUTPUT:





RESULT:

THE ANIMATION WAS CREATED SUCCESSFULLY.