Exercise 1
Study of Basic Output Primitives

AIM:

a) To create an output window using OPENGL and to draw the following basic output primitives – POINTS, LINES, LINE_STRIP, LINE_LOOP, TRIANGLES, QUADS, QUAD_STRIP, POLYGON.

b) To create an output window and draw a checkerboard using OpenGL.

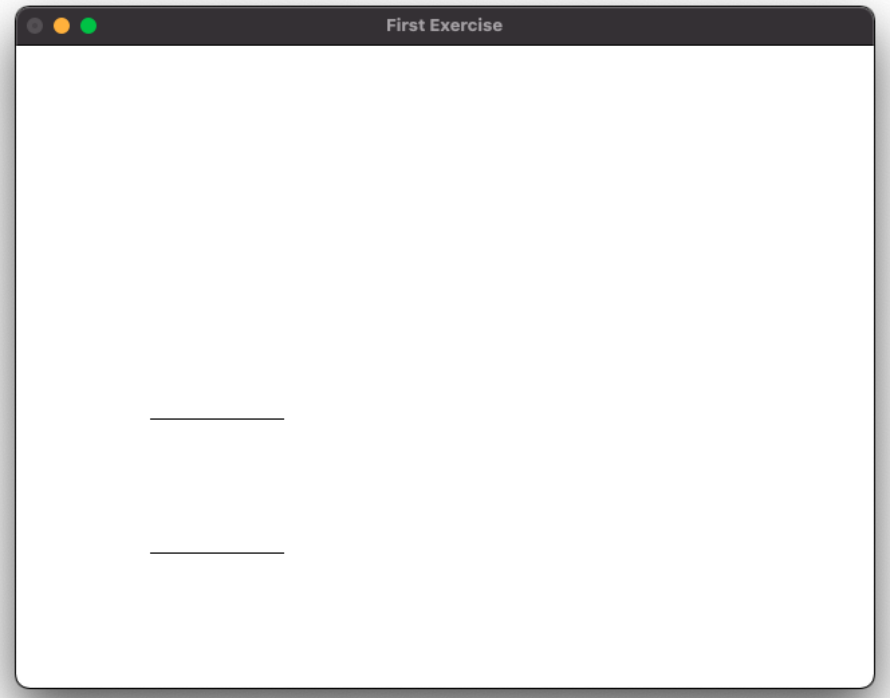
c) To create an output window and draw a house using POINTS, LINES, TRAINGLES and QUADS/POLYGON.

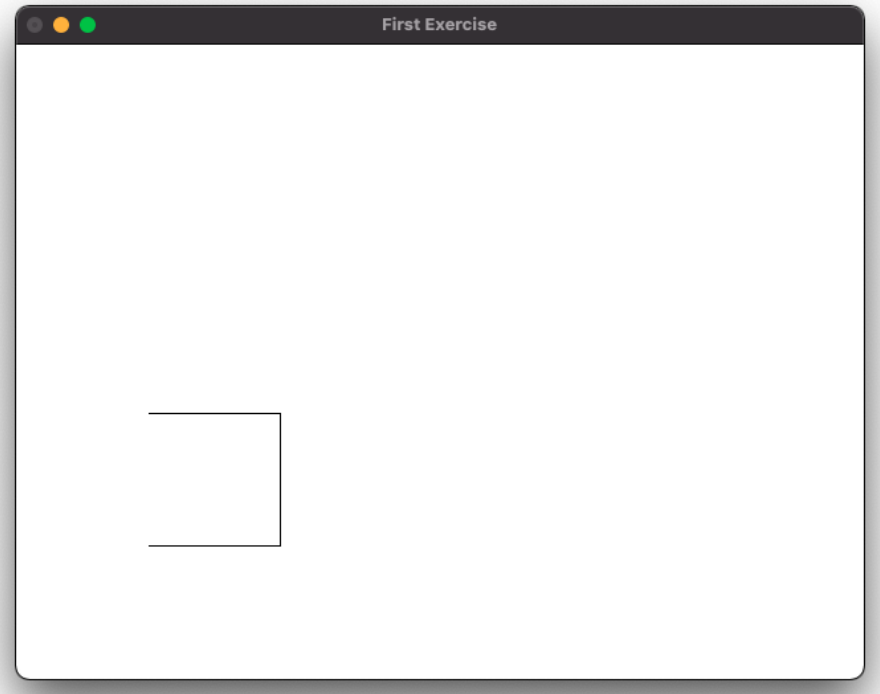
CODE:

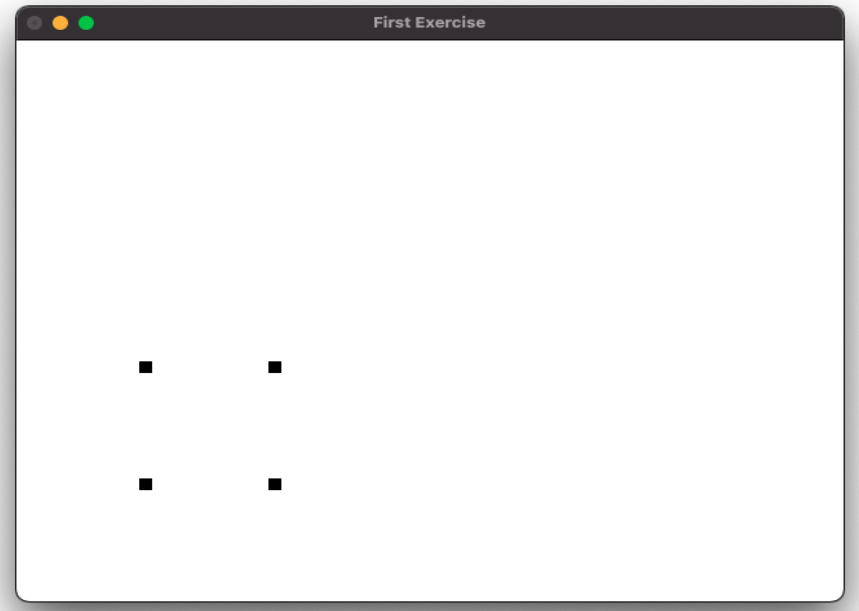
```
#include<GLUT/glut.h>

void myInit() {
   glClearColor(1.0,1.0,1.0,0.0);
   glColor3f(0.0f,0.0f,0.0f);
   //glPointSize(10);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(0.0,640.0,0.0,480.0);
}

void myDisplay() {
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_QUAD_STRIP);
   glVertex2d(150,100);
   glVertex2d(250,100);
   glVertex2d(250,200);
   glVertex2d(150,200);
   glVertex2d(300,300);
   glVertex2d(400,300);
   //glVertex2d(630,400);
   //glVertex2d(530,400);
   glEnd();
   glFlush();
}

void buildboard(int size){
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_QUADS);
```

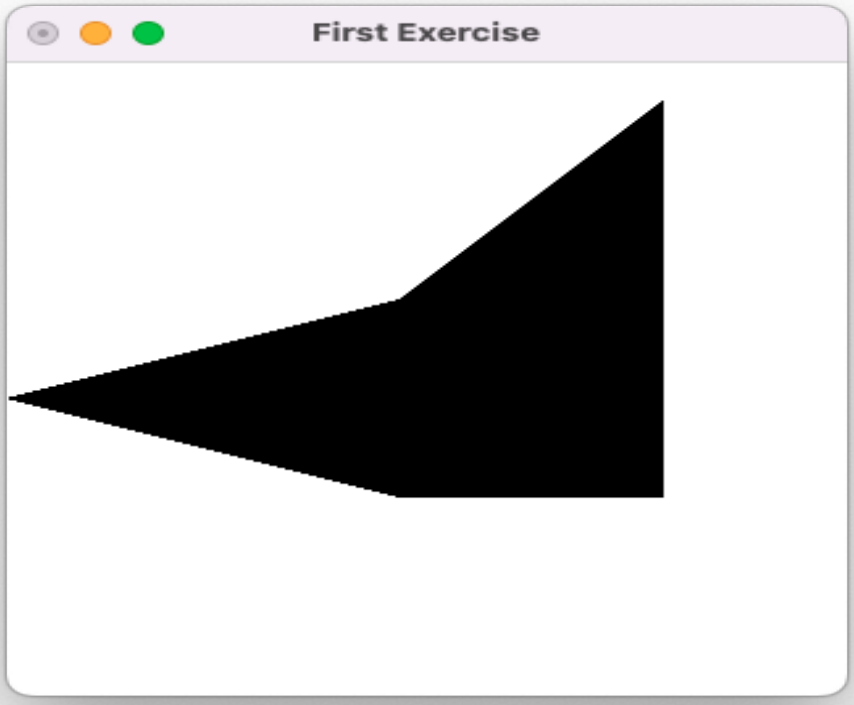
```
    for(int i=0;i<8;i++){
        for(int j=0;j<8;j++){
            if((i+j)%2==0){
                glColor3f(0.0f,0.0f,0.0f);
            }
            else{
                glColor3f(1.0f,1.0f,1.0f);
            }
            glVertex2d(i*size,j*size);
            glVertex2d(i*size+size,j*size);
            glVertex2d(i*size+size,j*size+size);
            glVertex2d(i*size,j*size+size);
        }
    }
    glEnd();
    glFlush();
}

void board(){
    int size=40;
    buildboard(size);
}

int main(int argc,char* argv[]) {
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(640,480);
    glutCreateWindow("First Exercise");
    glutDisplayFunc(myDisplay);
    myInit();
    glutMainLoop();
    return 1;
}
```

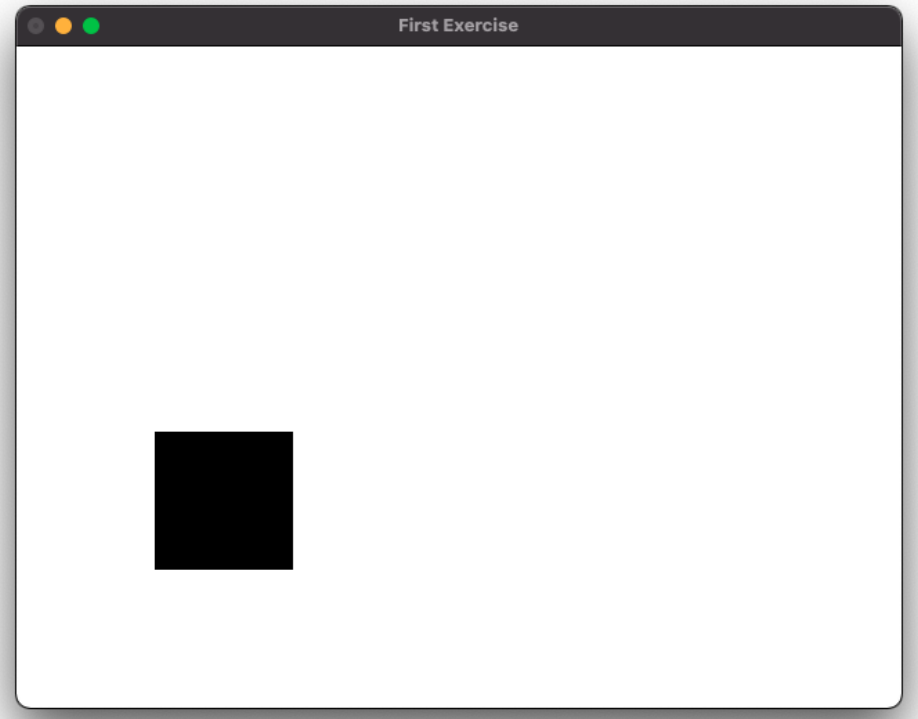
SAMPLE OUTPUT:

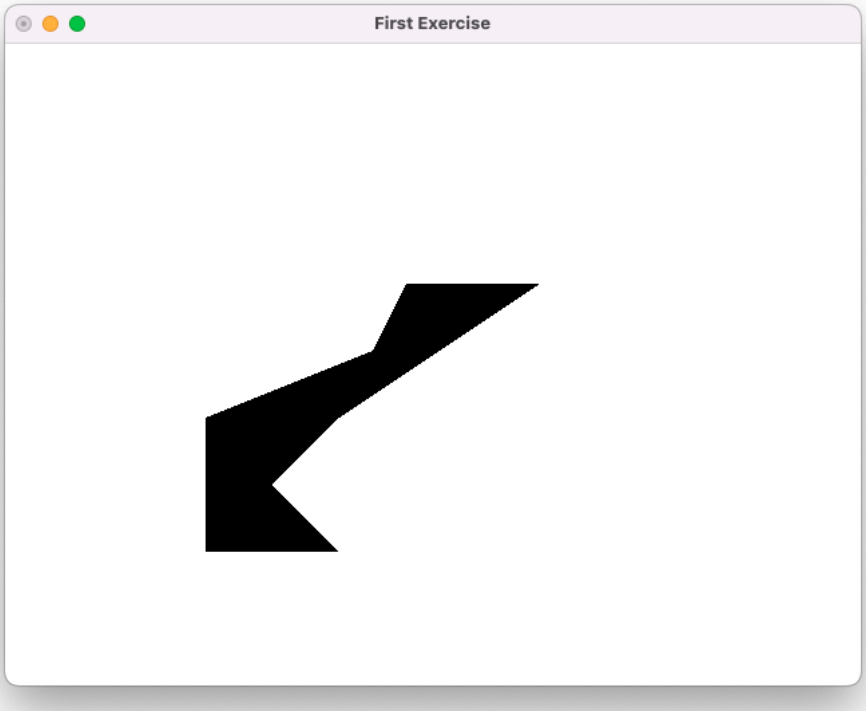


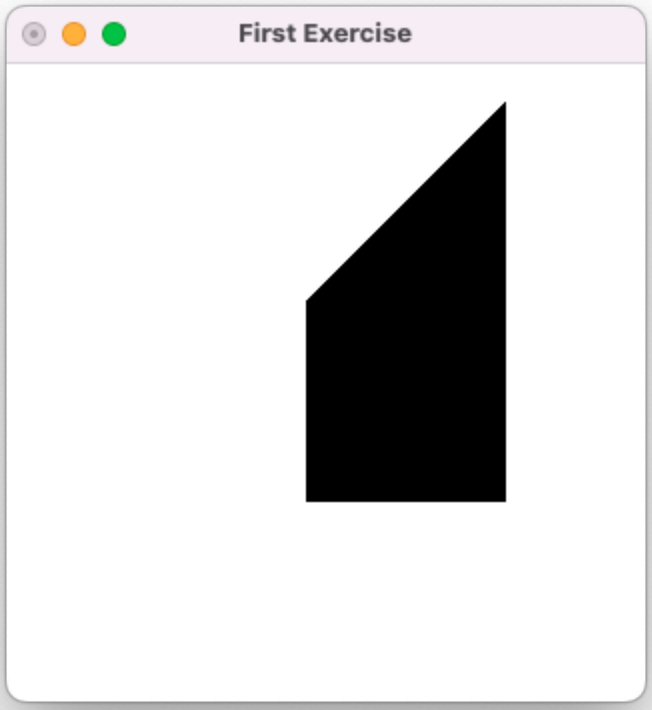


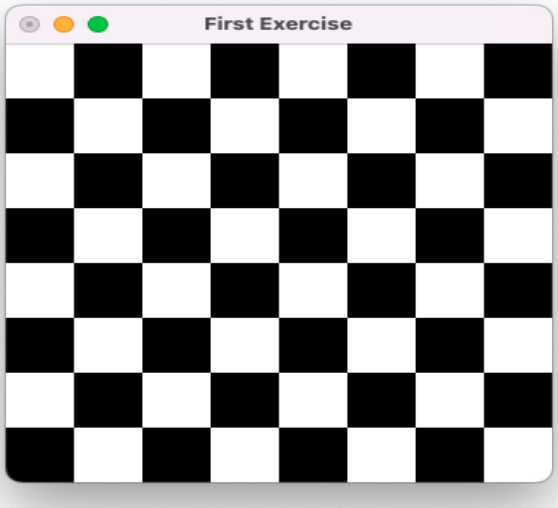

First Exercise

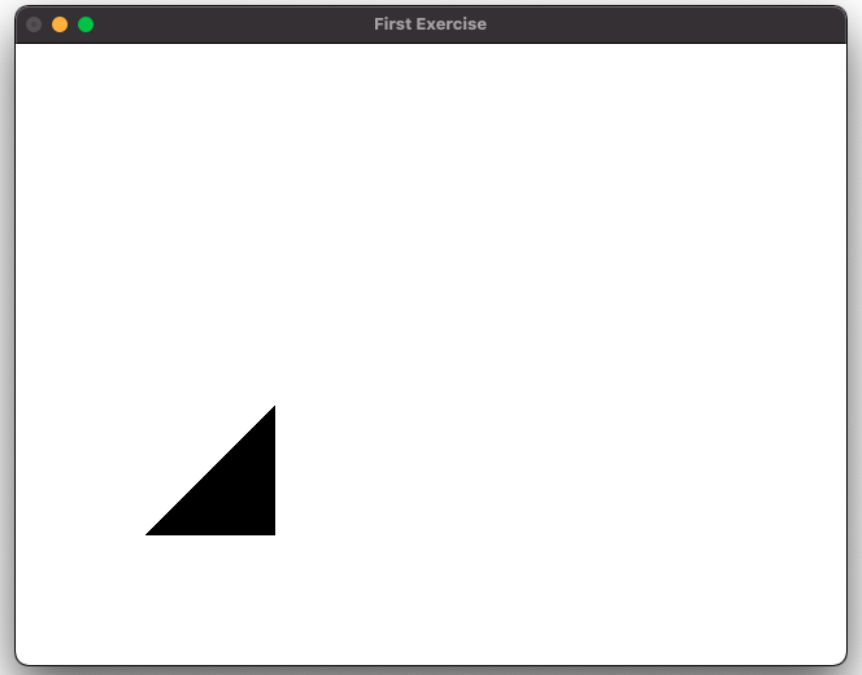

First Exercise

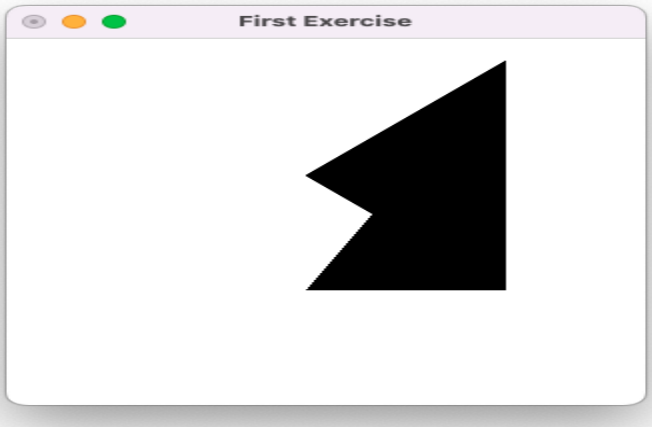

First Exercise

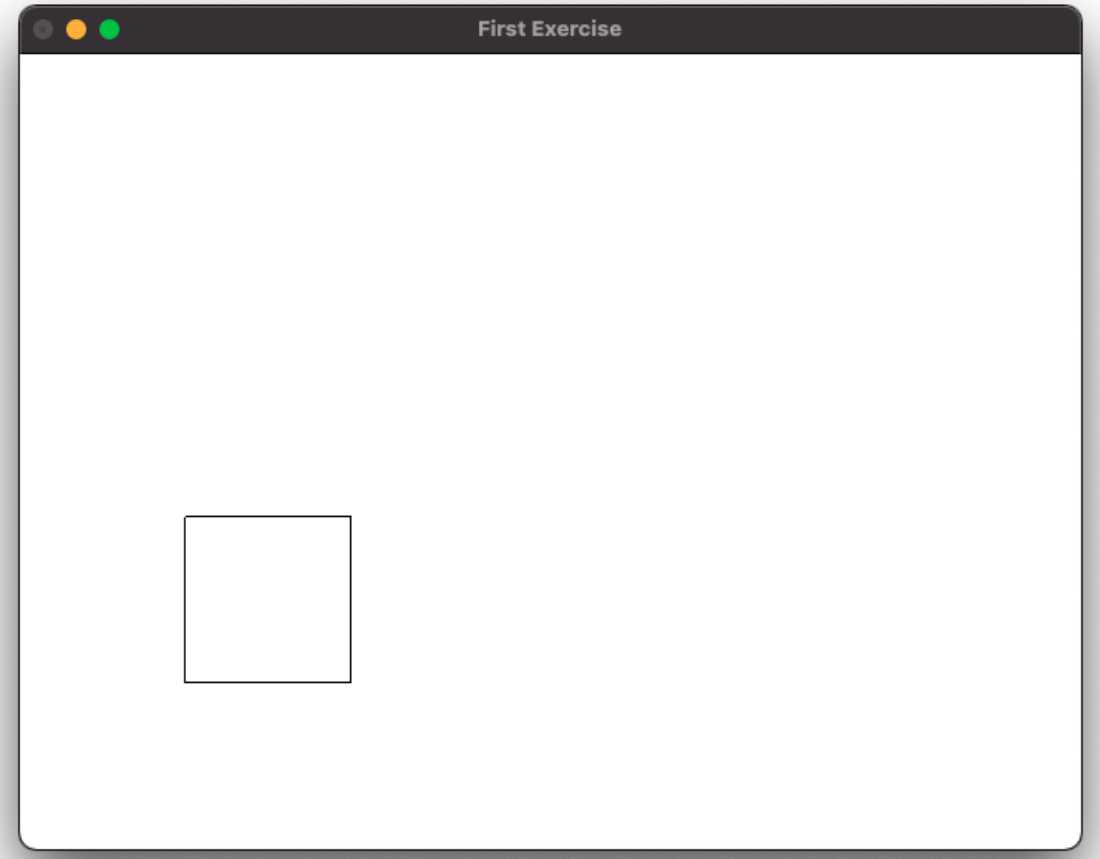

First Exercise


First Exercise


First Exercise


First Exercise





RESULT:

Basic Primitives were successfully drawn in C++ using OpenGL.

Exercise 2

DDA Line Drawing Algorithm

AIM:

To plot points that make up the line with endpoints (x0,y0) and (xn,yn) using DDA line drawing algorithm.

Case 1: +ve slope Left to Right line

Case 2: +ve slope Right to Left line

Case 3: -ve slope Left to Right line

Case 4: -ve slope Right to Left line

Each case has two subdivisions (i) |m|<= 1 (ii) |m|>1

Note that all four cases of line drawing must be given as test cases.

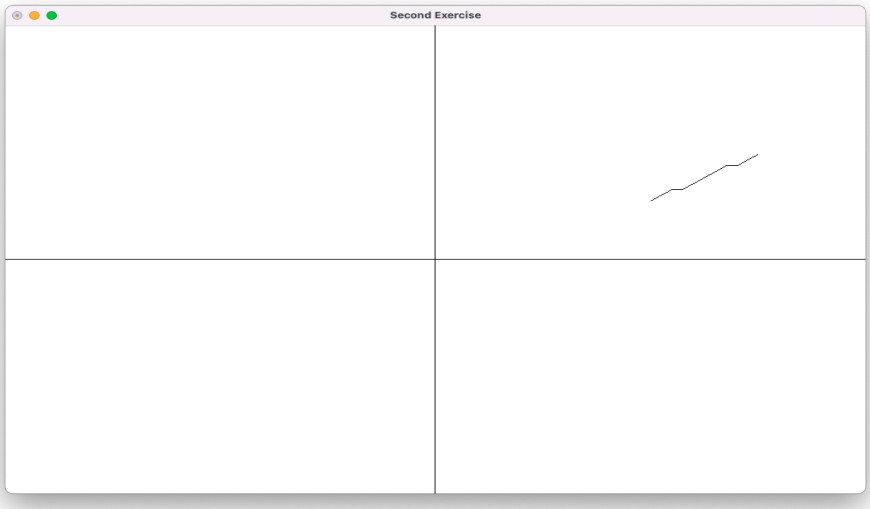
CODE:

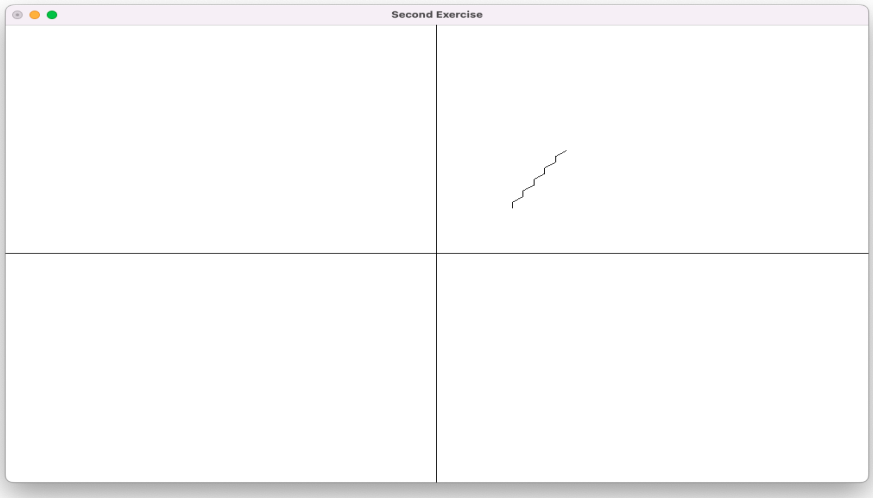
```
#include<GLUT/glut.h>

void myInit() {
   glClearColor(1.0,1.0,1.0,0.0);
   glColor3f(0.0f,0.0f,0.0f);
   //glPointSize(10);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(-40.0,40.0,-40.0,40.0);
}

void myDisplay(int x1,int y1,int x2,int y2) {
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_LINES);
   glVertex2d(0,-40);
   glVertex2d(0,40);
   glVertex2d(-40,0);
   glVertex2d(40,0);
   glEnd();
   int dx=x2-x1;
   int dy=y2-y1;
   float m=dy/dx;
   int p0;
   if(m<1){
```

```
        p0=2*dy-dx;
    }
    else{
        p0=2*dx-dy;
    }

    glBegin(GL_LINE_STRIP);
    glVertex2d(x1,y1);
    while(x1!=x2 && y1!=y2){
        if(m<1){
            if(p0>0){
                y1+=1;
                p0=p0-2*dx+2*dy;
            }
            else{
                p0=p0+2*dy;
            }
            x1+=1;
        }
        else{
            if(p0>0){
                x1+=1;
                p0=p0+2*dx-2*dy;
            }
            else{
                p0=p0+2*dx;
            }
            y1+=1;
        }
        glVertex2d(x1,y1);
    }
    glVertex2d(x1,y1);
    glEnd();
    glFlush();
}

void helper(){
    myDisplay(16,30,25,20);
}

int main(int argc,char* argv[]) {
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(1000,1000);
```

```
    glutCreateWindow("Second Exercise");
    glutDisplayFunc(helper);
    myInit();
    glutMainLoop();
    return 1;
}
```

SAMPLE OUTPUT:





RESULT:

DDA Line Drawing Algorithm was successfully implemented in C++ using OpenGL.

UCS1712 - GRAPHICS AND MULTIMEDIA LAB
ASSIGNMENT 3

QUESTION:

Bresenham's Line Drawing Algorithm in C++ using OpenGL

To plot points that make up the line with endpoints (x0,y0) and (xn,yn) using Bresenham's line drawing algorithm.

Case 1: +ve slope Left to Right line

Case 2: +ve slope Right to Left line

Case 3: -ve slope Left to Right line

Case 4: -ve slope Right to Left line

Each case has two subdivisions

(i) |m|<= 1 (ii) |m|>1

Note that all four cases of line drawing must be given as test cases.

AIM:

To plot points that make up the line with endpoints (x0,y0) and (xn,yn) using Bresenham's line drawing algorithm including all test cases.

ALGORITHM:

- Input line endpoints, (x0,y0) and (xn,yn)
- Load (x0,y0) into the frame buffer that is first point
- Calculate the constants Δx, Δy, $2\Delta y$ and $2\Delta y - 2\Delta x$
- Calculate parameter $p0 = 2\Delta y - \Delta x$
- Set pixel at position (x0,y0)
- Repeat the following steps until (xn,yn) is reached:
    - if $pk < 0$
        - Set the next pixel at position (xk +1, yk)
        - Calculate new $pk+1 = pk + 2\Delta y$
    - if $pk \geq 0$
        - Set the next pixel at position (xk +1, yk + 1)
        - Calculate new $pk+1 = pk + 2(\Delta y - \Delta x)$
- Repeat last step Δx times

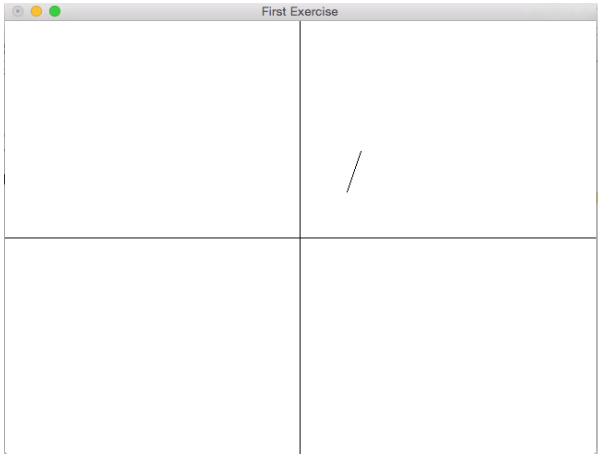
SOURCE CODE:

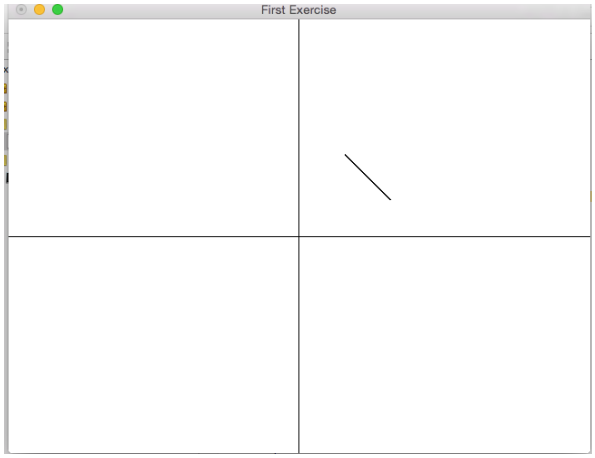
```
#include<iostream>
#include<GLUT/glut.h>
#include<OPENGL/OpenGL.h>
using namespace std;
void myInit() {
   glClearColor(1.0,1.0,1.0,0.0);
   glColor3f(0.0f,0.0f,0.0f);
   glPointSize(1);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(-640.0,640.0,-480.0,480.0);}
void myDisplay()
{
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_LINES);
   glVertex2f(-640,0);
   glVertex2f(640,0);
   glEnd();
   glBegin(GL_LINES);
   glVertex2f(0,-480);
   glVertex2f(0,480);
   glEnd();
   glBegin(GL_POINTS);
   int x1,x2,y1,y2;
   cout<<"Enter the coordinates of the starting points";
   cin>>x1>>y1;
   cout<<"Enter the coordinates of the ending points";
   cin>>x2>>y2;
   int m = (y2-y1)/(x2-x1);
   if(m>0){
      if(m<1)
      {
      int m_new = 2 * (y2 - y1);
      int slope_error_new = m_new - (x2 - x1);
      for (int x = x1, y = y1; x <= x2; x++)
      {
         glVertex2d(x,y);
         slope_error_new += m_new;
         if (slope_error_new >= 0)
```

```
        {
            y++;
            slope_error_new  -= 2 * (x2 - x1);
        }
    }
    }
else{
    int m_new = 2 * (x2 - x1);
    int slope_error_new = m_new - (y2 - y1);
    for (int x = x1, y = y1; y <= y2; y++)
    {
        glVertex2d(x,y);
        slope_error_new += m_new;
        if (slope_error_new >= 0)
        {
            x++;
            slope_error_new  -= 2 * (y2 - y1);
        }
    }
}
}
else
{ if(m>-1)
{
    int m_new = 2 * (y2 - y1);
    int slope_error_new = m_new - (x2 - x1);
    for (int x = x1, y = y1; x <= x2; x++)
    {
        glVertex2d(x,y);
        slope_error_new -= m_new;
        if (slope_error_new <= 0)
        {
            y--;
            slope_error_new  -= 2 * (x2 - x1);
        }
    }
}
    else
    {
        int m_new = 2 * (x2 - x1);
```

```
            int slope_error_new = m_new - (y2 - y1);
            for (int x = x1, y = y1; y <= y2; y++)
            {
               glVertex2d(x,y);
               slope_error_new -= m_new;
               if (slope_error_new <= 0)
               {
                  x--;
                  slope_error_new  -= 2 * (y2 - y1);
               }
            }
         }
      }
      glEnd();
      glFlush();
}
int main(int argc,char* argv[]) {
         glutInit(&argc,argv);
         glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
         glutInitWindowSize(640,480);
         glutCreateWindow("First Exercise");
         glutDisplayFunc(myDisplay);
         myInit();
         glutMainLoop();
         return 1;
      }
```

OUTPUT:





LEARNING OUTCOMES:

- Inclusion and incorporation of OpenGL and GLUT frameworks.
- Application of Bresenham's line algorithm to plot a line given 2 endpoints.
- Implementation of Bresenham's line drawing algorithm in C++.
- Testing of all the cases concerned with Bresenham's line algorithm.

UCS 1712 – GRAPHICS AND MULTIMEDIA LAB

Lab Exercise 4 – Midpoint Circle Drawing Algorithm in C++ using OpenGL

Aim :

a) To plot points that make up a circle with center (x_c, y_c) and radius r using midpoint circle drawing algorithm.

Algorithm :

a) Initialize GLUT parameters and set the display mode. Set the window size, name the window, and set the display function.
b) Clear the background (set it to the color of choice) and set the pen color and size. Initialize the bottom-left and top-right endpoints of the window.
c) Take as input the coordinates of the center and the radius of the circle.
d) Set the starting point to be (0,r) and the initial decision parameter p as 1-r.
e) While x<y, choose the next point as either y or y-1 depending upon the value of p. f) Plot the next point, as well its symmetry points in all the remaining 7 octants. Update the value of the decision parameter accordingly

Code :

a) Draw a circle

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

#include<iostream>

#include<cmath>


using namespace std;


void myInit() {

 glClearColor(1.0,1.0,1.0,0.0);

 glColor3f(0.0f,0.0f,0.0f);

 glPointSize(1);
```

```
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 glOrtho(-320.0,320.0,-240.0,240.0,1,-1);
}

void myDisplay() {

glClear(GL_COLOR_BUFFER
_BIT);  glBegin(GL_LINES);
 glVertex2d(-320,0);
 glVertex2d(320,0);
 glVertex2d(0,-240);
 glVertex2d(0,240);
 glEnd();
 glBegin(GL_POINTS);
 int x,y,xc,yc,r;
 cout<<"Enter center coordinates :
";  cin>>xc>>yc;
 cout<<"Enter radius : ";
 cin>>r;
 x=0, y=r;
 int p=1-r;
 while(x<y){
 glVertex2d(xc+x, yc+y);
 glVertex2d(xc-x, yc+y);
 glVertex2d(xc+x, yc-y);
 glVertex2d(xc-x, yc-y);
 glVertex2d(xc+y, yc+x);
```

```
glVertex2d(xc-y, yc+x);
glVertex2d(xc+y, yc-x);
glVertex2d(xc-y, yc-x);
x+=1;
if(p<0){
p+=(2*x)+1;
}
else{
y-=1;
p+=(2*x)+1-(2*y);
}
}
glEnd();
glFlush();
}

int main(int argc,char* argv[]) {
glutInit(&argc,argv);

glutInitDisplayMode(GLUT_SINGLE|GLUT
_RGB);  glutInitWindowSize(640,480);
glutCreateWindow("Ex 3");
glutDisplayFunc(myDisplay);
myInit();
glutMainLoop();
return 1;
}
```

b) Draw an object using line and circle drawing algorithms

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

using namespace std;
void myInit() {
 glClearColor(1.0,1.0,1.0,0.0);
 glColor3f(0.0f,0.0f,0.0f);
 glPointSize(1);

glMatrixMode(GL_PROJECTI
ON);  glLoadIdentity();

glOrtho(-320.0,320.0,-240.0,240.0,1,-
1); }

void myDisplay() {

glClear(GL_COLOR_BUFFER
_BIT);  glBegin(GL_LINES);
 glVertex2d(-320,0);
 glVertex2d(320,0);
 glVertex2d(0,-240);
 glVertex2d(0,240);
 glEnd();
 glBegin(GL_POINTS);
 int x,y,xc,yc,r;
```

```
 cout<<"Enter center coordinates :
";  cin>>xc>>yc;
 cout<<"Enter radius : ";
 cin>>r;
 x=0, y=r;
 int p=1-r;
 while(x<y){
 glVertex2d(xc+x, yc+y);
 glVertex2d(xc-x, yc+y);
 glVertex2d(xc+x, yc-y);
 glVertex2d(xc-x, yc-y);
glVertex2d(xc+y, yc+x);
glVertex2d(xc-y, yc+x);
glVertex2d(xc+y, yc-x);
glVertex2d(xc-y, yc-x);
x+=1;
 if(p<0){
 p+=(2*x)+1;
 }
 else{
 y-=1;
 p+=(2*x)+1-(2*y);  }
 }

 cout<<"Enter radius :
";  cin>>r;
 x=0, y=r;
 p=1-r;
```

```
while(x<y){
glVertex2d(xc+x, yc+y);
glVertex2d(xc-x,   yc+y);
glVertex2d(xc+x,   yc-y);
glVertex2d(xc-x, yc-y);
glVertex2d(xc+y, yc+x);
glVertex2d(xc-y,   yc+x);
glVertex2d(xc+y,   yc-x);
glVertex2d(xc-y, yc-x);
x+=1;
if(p<0){
p+=(2*x)+1;
}
else{
y-=1;
p+=(2*x)+1-(2*y);
}
}
int x1,y1,x2,y2;
cout<<"Enter starting point coordinates :
";  cin>>x1>>y1;
cout<<"Enter end point coordinates :
";  cin>>x2>>y2;
bool isLeft2Right;
if(x1<x2){
isLeft2Right=true;
}
else{
```

```
 isLeft2Right=false;
 }
 int dx=x2-x1, dy=y2-y1;
 float m=float(dy)/dx;
 float xl=x1/1.0, yl=y1/1.0;
 if(m>=-1 && m<=1){
 if(isLeft2Right){
 while(xl!=x2){
 glVertex2d(round(xl),round(yl));
xl+=1;
 yl+=m;
 }
 }
 else{
 while(xl!=x2){
 glVertex2d(round(xl),round(yl));
xl-=1;
 yl-=m;
 }
 }
 }
 else if(m>1){
 if(isLeft2Right){
 while(yl!=y2){
 glVertex2d(round(xl),round(yl));
yl+=1;
 xl+=1.0/m;
 }
```

```
 }
 else{
 while(yl!=y2){
 glVertex2d(round(xl),round(yl));
yl-=1;
 xl-=1.0/m;
 }
 }
 }
 else if(m<-1){
 if(isLeft2Right){
 while(yl!=y2){
 glVertex2d(round(xl),round(yl));
yl-=1;
 xl-=1.0/m;
 }
 }
 else{
 while(yl!=y2){
 glVertex2d(round(xl),round(yl));
yl+=1;
 xl+=1.0/m;
 }
 }
 }

 cout<<"Enter starting point coordinates :
";  cin>>x1>>y1;
```

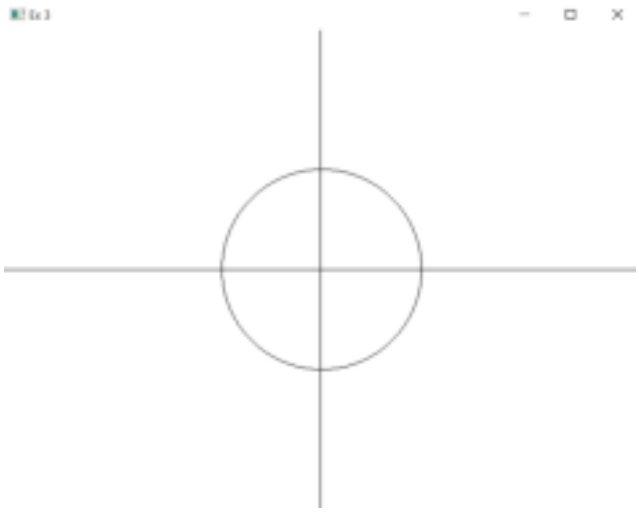
```
cout<<"Enter end point coordinates :
";  cin>>x2>>y2;
if(x1<x2){
isLeft2Right=true;
}
else{
isLeft2Right=false;
}
dx=x2-x1, dy=y2-y1;
m=float(dy)/dx;
xl=x1/1.0, yl=y1/1.0;
if(m>=-1 && m<=1){
if(isLeft2Right){
while(xl!=x2){
glVertex2d(round(xl),round(yl));
xl+=1;
yl+=m;
}
}
else{
while(xl!=x2){
glVertex2d(round(xl),round(yl));
xl-=1;
yl-=m;
}
}
}
else if(m>1){
```

```
 if(isLeft2Right){
 while(yl!=y2){
 glVertex2d(round(xl),round(yl));
yl+=1;
 xl+=1.0/m;
 }
 }
 else{
 while(yl!=y2){
 glVertex2d(round(xl),round(yl));
yl-=1;
 xl-=1.0/m;
 }
 }
 }
 else if(m<-1){
 if(isLeft2Right){
 while(yl!=y2){
 glVertex2d(round(xl),round(yl));
yl-=1;
 xl-=1.0/m;
 }
 }
 else{
 while(yl!=y2){
 glVertex2d(round(xl),round(yl));
yl+=1;
 xl+=1.0/m;
```

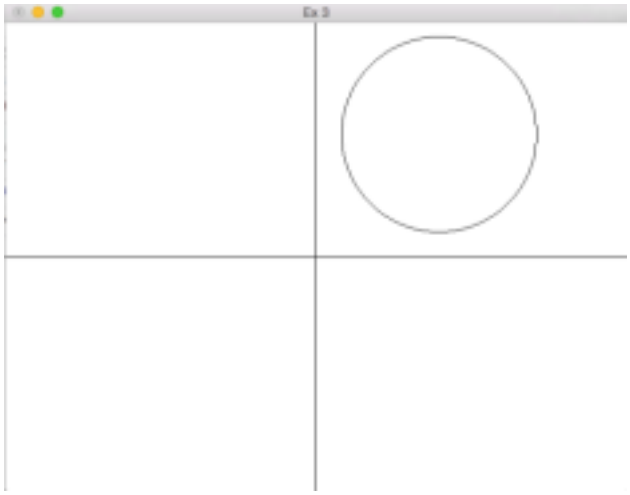
```
}
}
}

glEnd();
glFlush();
}

int main(int argc,char* argv[]) {
glutInit(&argc,argv);

glutInitDisplayMode(GLUT_SINGLE|GLUT
_RGB);  glutInitWindowSize(640,480);
glutCreateWindow("Ex 3");
glutDisplayFunc(myDisplay);
myInit();
glutMainLoop();
return 1;
}
```

Sample Output :

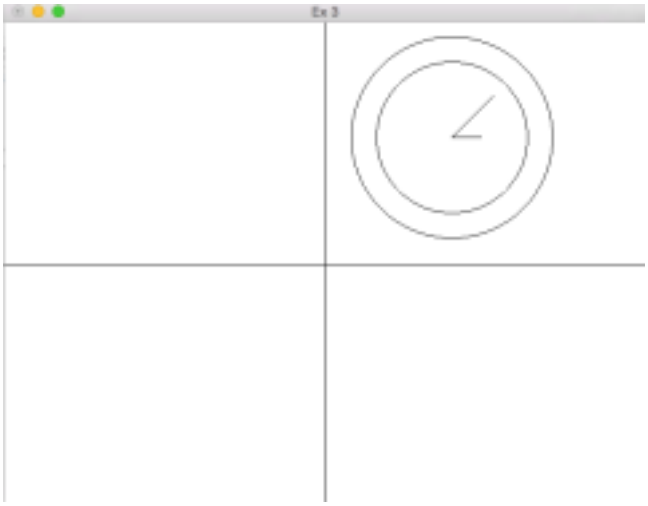
a) Draw a circle with center (0,0)



b) Draw a circle with center (125,125) and radius 100



c) Draw an object with line and circle drawing algorithms



Learning Outcome :

a) Learnt about the frameworks and libraries required to implement graphics in C++ b) Learnt to implement the ‘GL_POINTS’ output primitive in OpenGL

c) Learnt to draw a circle given the center and radius using midpoint circle drawing algorithm

Exercise 5
2D Transformations in C++ using OpenGL

To apply the following 2D transformations on objects and to render the final output along with the original object.
1) Translation
2) Rotation a) about origin b) with respect to a fixed point (xr,yr)
3) Scaling with respect to a) origin - Uniform Vs Differential Scaling b) fixed point (xf,yf)
4) Reflection with respect to a) x-axis b) y-axis c) origin d) the line x=y
5) Shearing a) x-direction shear b) y-direction shear
Note: Use Homogeneous coordinate representations and matrix multiplication to perform transformations. Divide the output window into four quadrants. (Use LINES primitive to draw x and y axis.

Source Code:

```
//#include <iostream>
//
//double main(double argc, const char * argv[]) {
//    // insert code here...
//    std::cout << "Hello, World!\n";
//    return 0;
//}


#include<iostream>

#include<cmath>

#include<GLUT/GLUT.h>

#include<vector>

using namespace std;
```

```
typedef vector<vector<double>> vvi;

double c;

double x[5000];

double y[5000];

double pt = 0;

void myDisplay()

{

   glClear(GL_COLOR_BUFFER_BIT);

   glBegin(GL_POINTS);

   for(int i=0; i<=pt; i++){

      glVertex2d(x[i],y[i]);
```

```
    }

    glEnd();

    glFlush();

}

void myInit()
{
    glClearColor(1.0,1.0,1.0,0.0);
//    glColor3f(0.0f,0.0f,0.0f);
    glPointSize(1);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-300.0,300.0,-300.0,300.0);
}
vector<double> multiplication(vector<vector<double>> a,vector<vector<double>> b){
    double r1=a.size();double c2=b[0].size();double c1=a[0].size();
```

```
    vector<vector<double>> mult(r1,vector<double>(c2,0));

    for(double i = 0; i < r1; ++i)

            for(double j = 0; j < c2; ++j)

                for(double k = 0; k < c1; ++k)

                {

                    mult[i][j] += a[i][k] * b[k][j];

                }

//    cout << mult.size();

//    for(double i=0;i<3;i++) for(double j=0;j<1;j++) cout << mult[i][j] << " ";

    return {mult[0][0],mult[1][0]};

}

void polygonDraw(vector<double>podoubles) {

    glColor3f(0.0f,0.0f,0.0f);

    glBegin(GL_POLYGON);

    for(double i=0;i<podoubles.size()-1;i+=2)

        glVertex2d(podoubles[i],podoubles[i+1]);

    glEnd();

}

void shear(double shh,double yref){

    vector<vector<double>> shearMat={{1,shh,-shh*yref},{0,1,0},{0,0,1}};
```

```
    vector<double> input={0,0,100,0,100,100,0,100};

    vector<double> output(input.size(),0);

    for(double i=0;i<input.size()-1;i+=2){

        vector<double> temp=multiplication(shearMat, {{input[i]},{input[i+1]},{1}});

//        cout << temp[0];

        output[i]=temp[0];output[i+1]=temp[1];

    }

    polygonDraw(output);

}



void refl(){

    vector<vector<double>> reflMat={{0,1,0},{1,0,0},{0,0,1}};

    vector<double> input={200,50,250,50,250,100,200,100};

    vector<double> output(input.size(),0);

    for(double i=0;i<input.size()-1;i+=2){

        vector<double> temp=multiplication(reflMat, {{input[i]},{input[i+1]},{1}});

//        cout << temp[0];

        output[i]=temp[0];output[i+1]=temp[1];

    }

    polygonDraw(output);
}
```

```
vector<double> scale(double sx, double sy, vector<double> input){
    vector<vector<double>> scaleMat={{sx,0,0},{0,sy,0},{0,0,1}};

    vector<double> output(input.size(),0);

    for(double i=0;i<input.size()-1;i+=2){

        vector<double> temp=multiplication(scaleMat, {{input[i]},{input[i+1]},{1}});
        output[i]=temp[0];output[i+1]=temp[1];

    }
    return output;
}

vector<double> translate(double x,double y,vector<double> input){

    vector<vector<double>> transMat={{1,0,x},{0,1,y},{0,0,1}};

//    vector<double> input={200,50,250,50,250,100,200,100};

    vector<double> output(input.size(),0);

    for(double i=0;i<input.size()-1;i+=2){

        vector<double> temp=multiplication(transMat, {{input[i]},{input[i+1]},{1}});

//        cout << temp[0];

        output[i]=temp[0];output[i+1]=temp[1];

    }
//    polygonDraw(input);
//    polygonDraw(output);
    return output;

}

vector<double> rotate(double angle,vector<double> input){
    angle*= M_PI/180;
```

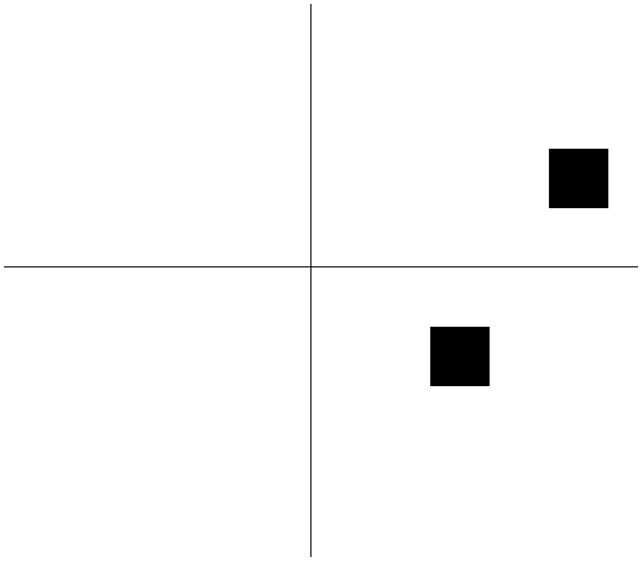
```
    vector<vector<double>>
rotMat={{cos(angle),-sin(angle),0},{sin(angle),cos(angle),0},{0,0,1}};
    vector<double> output(input.size(),0);

    for(double i=0;i<input.size()-1;i+=2){

        vector<double> temp=multiplication(rotMat, {{input[i]},{input[i+1]},{1}});

//        cout << temp[0];

        output[i]=temp[0];output[i+1]=temp[1];

    }
//    polygonDraw(input);
//    polygonDraw(output);
    return output;
}
void draw() {

    double x1, y1, x2, y2;

//    cin >> x1 >> y1;

//    cin >> x2 >> y2;

    x1=70,y1=20,x2=300,y2=20;



    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(0.0f,0.0f,0.0f);

    glBegin(GL_LINES);

        glVertex2f(-600.0,0);

        glVertex2f(600.0,0);

    glEnd();
```

```
    glBegin(GL_LINES);

        glVertex2f(0,-600.0);

        glVertex2f(0,600.0);

    glEnd();

    vector<double> input={150,50,200,50,200,100,150,100};
    vector<double> temp={0,0,100,0,100,100,0,100};
//    temp.append(3);
//    polygonDraw(temp);

//    shear(1,-50);
//    refl();
//    translate(-100,-150);
    int x=0,y=50;
    int sx=2.5,sy=3;
    polygonDraw(input);
    polygonDraw(translate(x,y,rotate(60,translate(-x,-y,input))));
//    polygonDraw(translate(x,y,scale(sx,sy,translate(-x,-y,input))));
    glFlush();









}

int main(int argc, char* argv[]) {

//    double a, b;
```

```
//    cin >> a >> b;

//    cout << a << " " << b;

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);

    glutInitWindowSize(600, 600);

    glutCreateWindow("First Ex");



    glutDisplayFunc(draw);

    myInit();

    glutMainLoop();

    return 1;

}
```

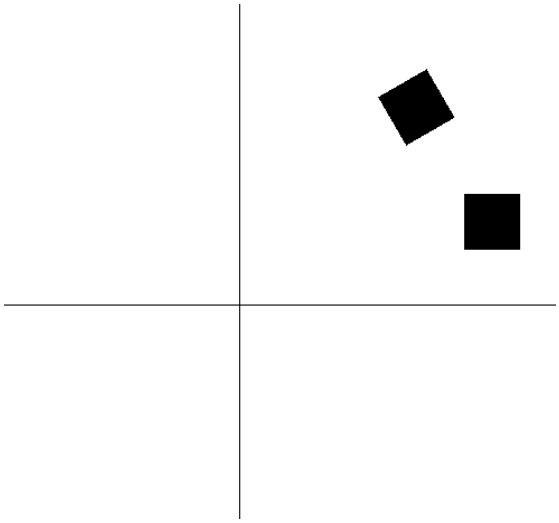
Output:

Translation:

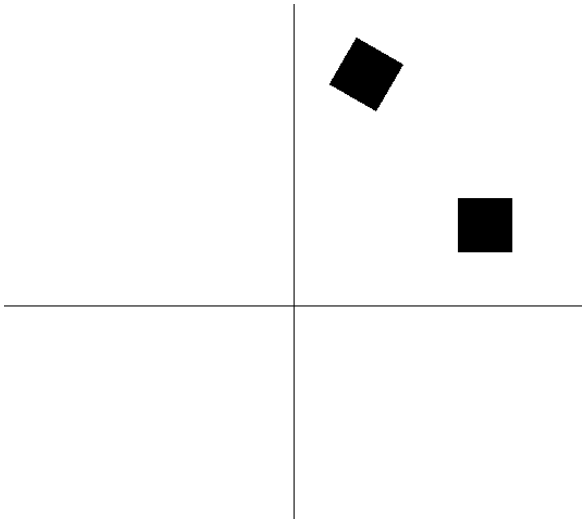


Rotation

About Origin (30 deg):

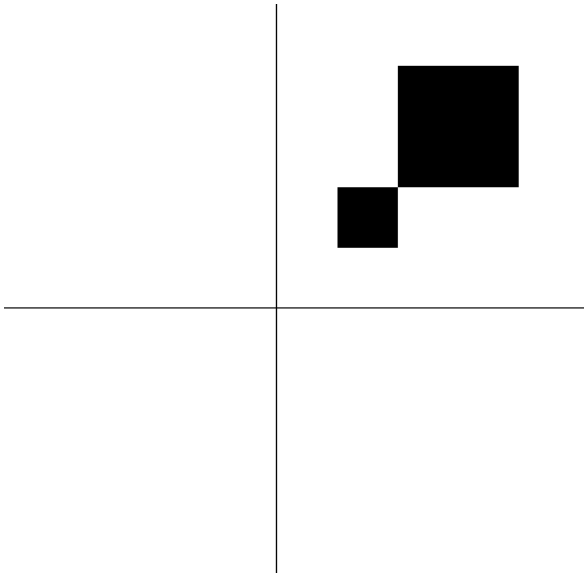


w.r.t. fixed point(0,50) at an Angle 60 deg:

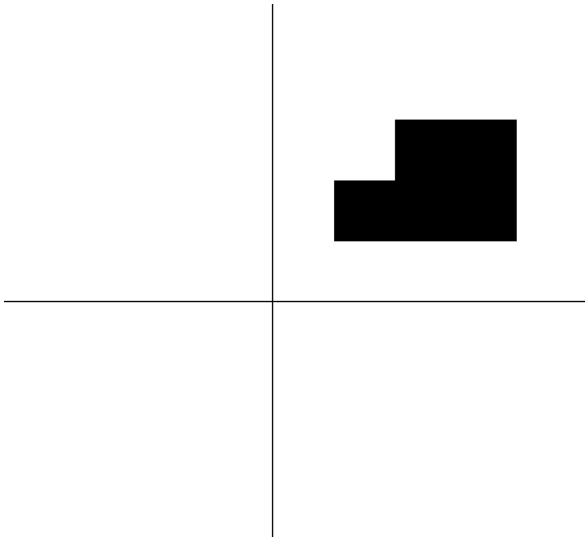


Scaling

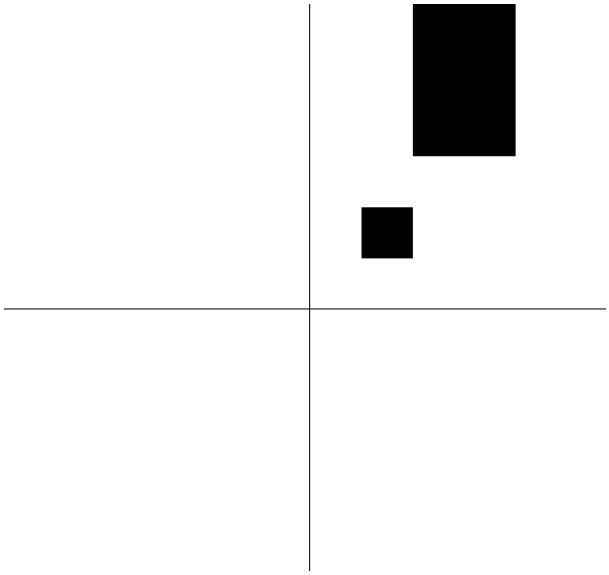
W.r.t origin



W.r.t fixed point (0,50)

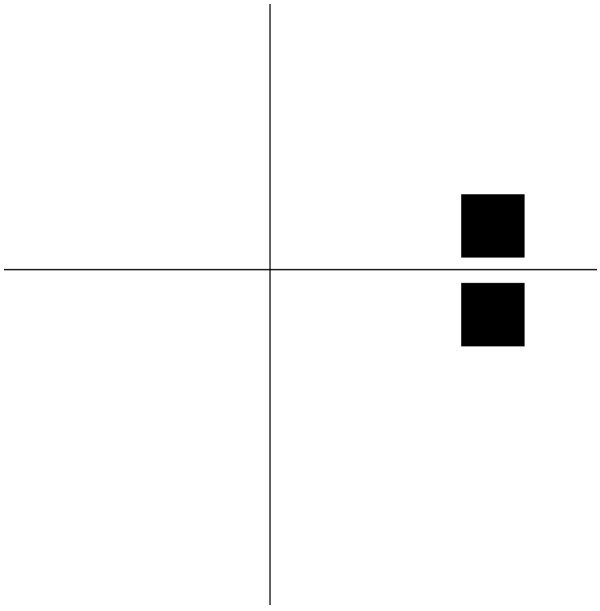


Differential scaling with Sx and Sy unequal

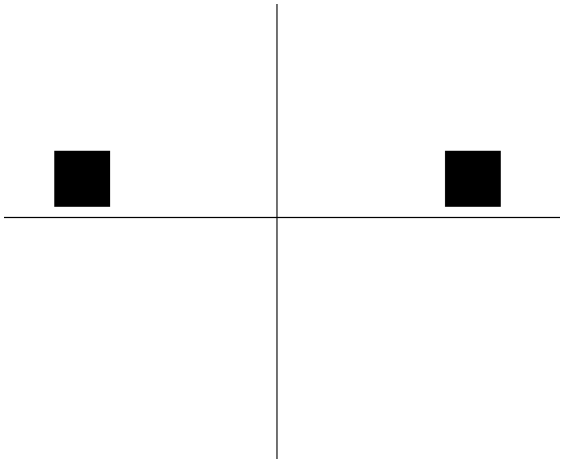


Reflection

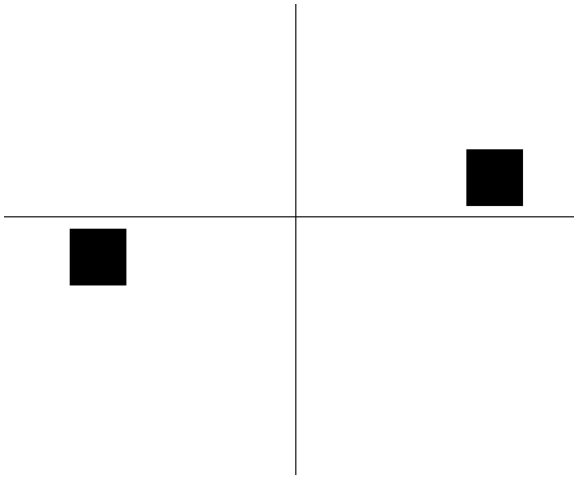
X-Axis



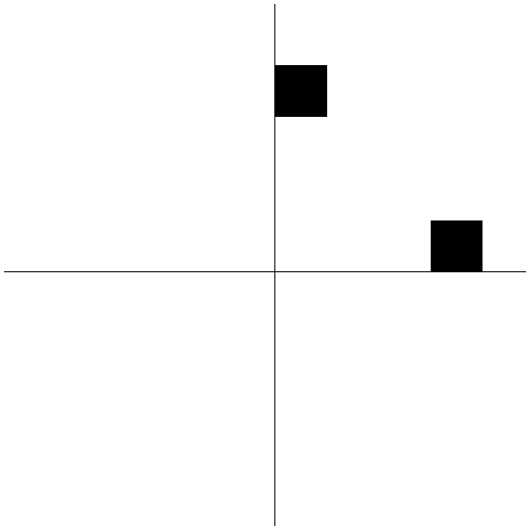
Y-Axis



Origin

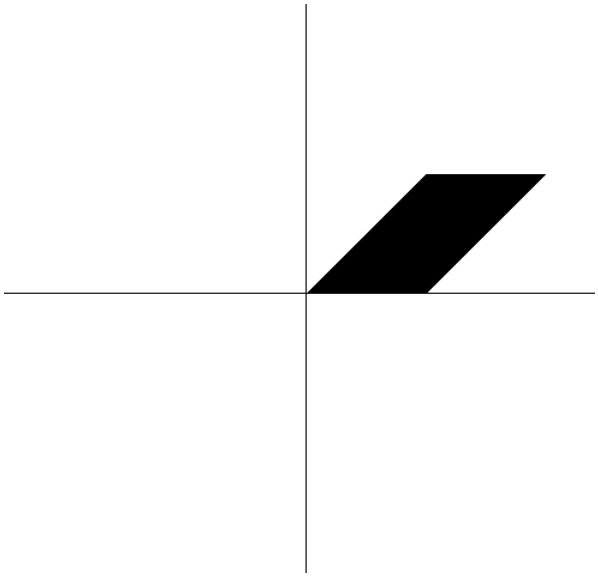


X=Y

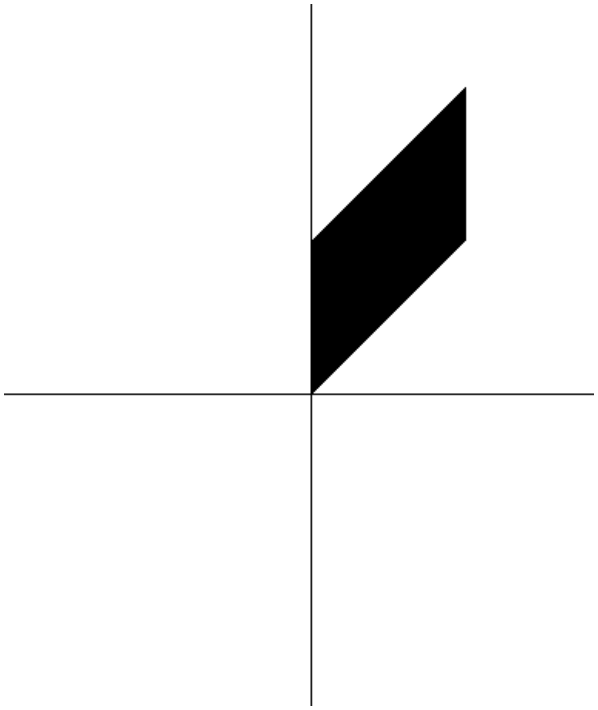


Shearing

X-Direction Shear



Y-Direction Shear



Learning Outcomes:

- Inclusion and incorporation of OpenGL and GLUT frameworks.
- Learning about 2D transformations.
- Addressing how translation, rotation, scaling, shearing, reflection transformations work.
- Learning about homogeneous coordinate representation and to represent transformations as matrices, use matrix multiplication for composite transformations.

UCS1712 - GRAPHICS AND MULTIMEDIA LAB
ASSIGNMENT 6

Aim:

a) To exhibit composite transformations for any 2 transformations given as input on an object.
b) To apply window to viewport transformation on an object and to display both window and viewport.

Algorithm:

a) Composite transformations
- Input the object coordinates.
- Form input matrix using the coordinates
- Draw the initial polygon.

- Input user choice from the list of available transformations.
    - According to the choice, compute the corresponding transformation matrix and return the resultant matrix.
    - If choice is invalid, prompt the user.
    - Use the resultant matrix of the first choice transformation as the input for the second choice transformation and compute the final composite transformation matrix.
    - Draw the final polygon after 2 transformations.
    - While drawing the object on the screen, display the coordinates of the object vertices.

b) Window to viewport transformations

- Input the coordinates of the window,viewport and the object in the window. ● Draw the window, viewport and window object on the screen.
- For computing the corresponding coordinates of the object in the viewport (window to viewport transformation):
    - Obtain the minimum and maximum values of x and y coordinates from both window and viewport.
    - Compute sx and sy using the formula
- sx = xvmax-xvmin / xwmax-xwmin
- sy = yvmax-yvmin / ywmax-ywmin

- For each vertex of the window object (xw,yw), calculate the corresponding viewport object coordinates using the following formula
- xv = xvmin + (xw - xwmin) * sx;
- yv = yvmin + (yw - ywmin) * sy;

- Draw the transformed viewport object on the screen.
    - While drawing the object on the screen, display the coordinates of the object vertices.

Source Code:

a) Composite transformations

```
#include <vector>
#include <iostream>
#include <GL/glut.h>
#include <cmath>
#include<string>
using namespace std;
vector<vector<double>> multiply(vector<vector<double>> A,
vector<vector<double>> B) {
 double r1 = A.size();
```

```
 double c2 = B[0].size();
 double c1 = A[0].size();
 vector<vector<double>> res(r1, vector<double>(c2, 0.0));
 for (int i = 0; i < r1; i++)
 for (int j = 0; j < c2; j++)
 for (int k = 0; k < c1; k++)
 {
 res[i][j] += A[i][k] * B[k][j];
 }
 return res;
}
vector<vector<double>> translation(vector<vector<double>> inp)
{
 double tx = 250, ty = 150;
 vector<vector<double>> T = { {1,0,tx},{0,1,ty},{0,0,1} };
 vector<vector<double>> res = multiply(T, inp);
 return res;
}
vector<vector<double>> rotation(vector<vector<double>> inp)
{
 double theta = 45 * 3.142 / 180;
 double sinT = sin(theta), cosT = cos(theta);
 double xr = 50, yr = 50;
 vector<vector<double>> T = { {cosT,-1 * sinT,((xr * (1 - cosT)) + (yr *
sinT))},{sinT,cosT,((yr * (1  - cosT)) -
 (xr * sinT))},{0,0,1} };
 vector<vector<double>> res = multiply(T, inp);
 return res;
}
vector<vector<double>> scaling(vector<vector<double>> inp)
{
 double sx = 2, sy = 2;
 //double sx=2,sy=0.5;
 int xf = 50, yf = 50;
 vector<vector<double>> T = { {sx,0,(xf * (1 - sx))},{0,sy,(yf *
(1 - sy))},{0,0,1} };  vector<vector<double>> res = multiply(T,
inp);
 return res;
}
```

```
vector<vector<double>> reflection(vector<vector<double>> inp)
{
 vector<vector<double>> T = { {1,0,0},{0,-1,0},{0,0,1} };
 vector<vector<double>> res = multiply(T, inp);
 return res;
}
vector<vector<double>> shearing(vector<vector<double>> inp)
{
 double shx = 0, shy = 0.5;
 double xref = 5, yref = 0;
 vector<vector<double>> T = { {1,0,0},{shy,1,(-shy * xref)},{0,0,1} };
 vector<vector<double>> res = multiply(T, inp);
 return res;
}
```

main.cpp

```
#include"transformations.h"
vector<vector<double>> form_inp(vector<double> arr)
{
 vector<vector<double>> inp;
 vector<double> x;
 vector<double> y;
 vector<double> ones(arr.size() / 2, 1);
 for (int i = 0; i < arr.size(); i++)
 {
 if (i % 2 == 0)
 {
 x.push_back(arr[i]);
 }
 else
 {
 y.push_back(arr[i]);
 }
 }
 inp.push_back(x);
 inp.push_back(y);
 inp.push_back(ones);
 return inp;
}
void write_points(int x1, int y1)
```

```
{
 string p1 = "(" + to_string(x1) + "," +
to_string(y1) + ")";  glRasterPos2i(x1 -
25, y1 - 15);
 for (int i = 0; p1[i] != '\0'; i++) {

glutBitmapCharacter(GLUT_BITMAP_8_
BY_13, p1[i]);  }
}
void point_writer(vector<int> w)
{
 for (int i = 0; i < w.size(); i = i + 2)
 {
 write_points(w[i], w[i + 1]);
 }
}
void
draw_polygon(vector<vector<dou
ble>> res) {
 vector<int> obj;
 glBegin(GL_LINE_LOOP);
 for (int i = 0; i < res[0].size(); i++)
 {
 int x = 0;
 float temp[2];
 for (int j = 0; j < 2; j++)
 {
 temp[x] = res[j][i];
 obj.push_back(temp[x]);
 x++;
 }
 glVertex2d(temp[0], temp[1]);
 }
 glEnd();
 point_writer(obj);
}
vector<vector<double>> choices(vector<vector<double>>
inp, char choice) {
 switch (choice)
```

```
 {
 case 'a': inp = translation(inp); break;
 case 'b': inp = rotation(inp); break;
 case 'c': inp = scaling(inp); break;
 case 'd': inp = reflection(inp); break;
 case 'e': inp = shearing(inp); break;
 default: cout << "Invalid choice! Make appropriate choice!\n";
 }
 return inp;
}
void operations()
{
 vector<double> arr = { 10,10,110,10,110,110,10,110 };
 vector<vector<double>> inp = form_inp(arr);
 char choice;
 draw_polygon(inp);
 cout << "\t2D
transformations:\na.Translation\nb.Rotation\nc.Scaling\nd.Reflection\ne.
Shearing\n";  for (int i = 0; i < 2; i++)
 {
 cout << "Enter choice "<< i+1 <<": ";
 cin >> choice;
 inp=choices(inp, choice);
 }
 draw_polygon(inp);
 return;
}
void myInit()
{
 glClearColor(1.0, 1.0, 1.0, 0.0);
 glColor3f(0.0f, 0.0f, 0.0f);
 glPointSize(1);
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 gluOrtho2D(-640.0, 640.0,
-480.0, 480.0); }
void myDisplay()
{
 glClear(GL_COLOR_BUFFER_BIT);
```

```
 glBegin(GL_LINES);
 glVertex2f(-640, 0);
 glVertex2f(640, 0);
 glVertex2f(0, -480);
 glVertex2f(0, 480);
 glEnd();
 operations();
 glFlush();
}
int main(int argc, char* argv[])
{
 glutInit(&argc, argv);

glutInitDisplayMode(GLUT_SINGL
E | GLUT_RGB);
glutInitWindowSize(640, 480);
 glutCreateWindow("First Exercise");
 glutDisplayFunc(myDisplay);
 myInit();
 glutMainLoop();
 return 1;
}
```

b) Window to viewport transformations **main.cpp**

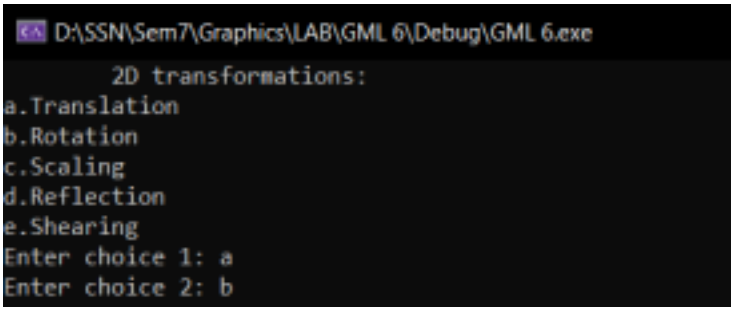
```
#include <vector>
#include <iostream>
#include <GL/glut.h>
#include <cmath>
#include<string>
using namespace std;
void write_points(int x1, int y1)
{
 string p1 = "(" + to_string(x1) + "," + to_string(y1) + ")";
 glRasterPos2i(x1 - 25, y1 - 15);
 for (int i = 0; p1[i] != '\0'; i++) {
 glutBitmapCharacter(GLUT_BITMAP_8_BY_13, p1[i]);
 }
}
void point_writer(vector<int> w)
```

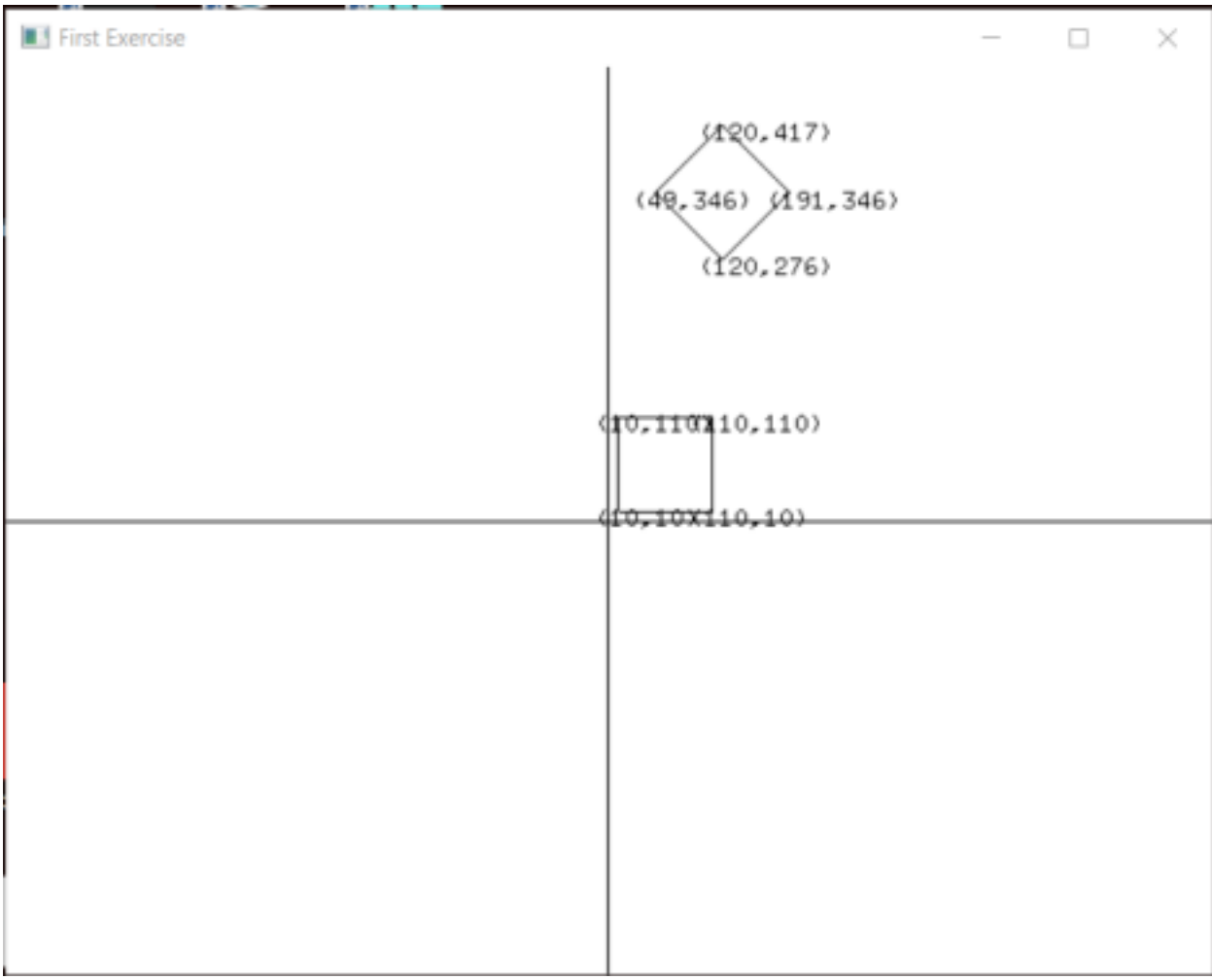
```
{
 for (int i = 0; i < w.size(); i = i + 2)
 {
 write_points(w[i], w[i + 1]);
 }
}
void plot_area(vector<int> w)
{
 glBegin(GL_LINE_LOOP);
 for (int i = 0; i < w.size(); i=i+2)
 {
 glVertex2d(w[i], w[i + 1]);
 }
 glEnd();
 point_writer(w);
}
void viewport_point(vector<int> window,vector<int> viewport,vector<int>
object_window) {
 int xwmin = window[0]; int ywmin = window[1];
 int xwmax = window[4]; int ywmax = window[5];
 int xvmin = viewport[0]; int yvmin = viewport[1];
 int xvmax = viewport[4]; int yvmax = viewport[5];
 float sx,sy;
 float xv, yv;
 vector<int> object_viewport;
 sx = float(xvmax-xvmin) /
float(xwmax-xwmin);  sy =
float(yvmax-yvmin) /
float(ywmax-ywmin);  for (int i =
0; i < object_window.size(); i = i +
2)  {
 xv = xvmin + (object_window[i] -
xwmin) * sx;  yv = yvmin +
(object_window[i+1] - ywmin) * sy;
object_viewport.push_back(xv);
 object_viewport.push_back(yv);
 }
 plot_area(object_viewport);
}
```

```
void operations()
{
 vector<int> window =
{50,100,350,100,350,400,50,400};
plot_area(window);
 vector<int> viewport =
{400,150,600,150,600,350,400,350};
plot_area(viewport);
 vector<int> object_window = {
100,200,200,200,200,300};
plot_area(object_window);

 viewport_point(window,viewport,obje
ct_window); }
void myInit()
{
 glClearColor(1.0, 1.0, 1.0, 0.0);
 glColor3f(0.0f, 0.0f, 0.0f);
 glPointSize(10);
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
void myDisplay()
{
 glClear(GL_COLOR_BUFFER_BIT);
 operations();
 glFlush();
}
int main(int argc, char* argv[])
{
 glutInit(&argc, argv);

glutInitDisplayMode(GLUT_SINGL
E | GLUT_RGB);
glutInitWindowSize(640, 480);
 glutCreateWindow("First Exercise");
 glutDisplayFunc(myDisplay);
 myInit();
```

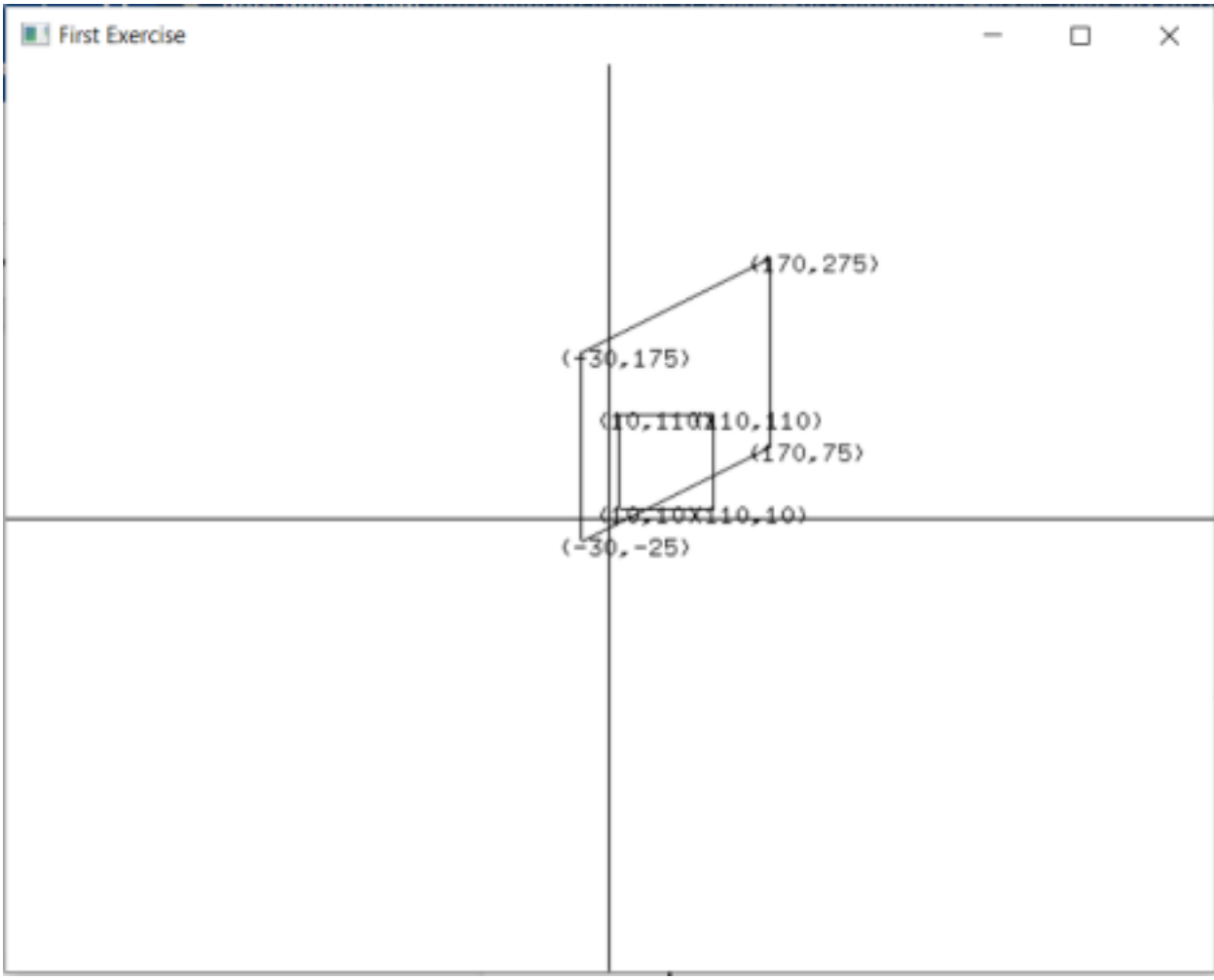
```
glutMainLoop();
return 1;
}
```

OUTPUT:

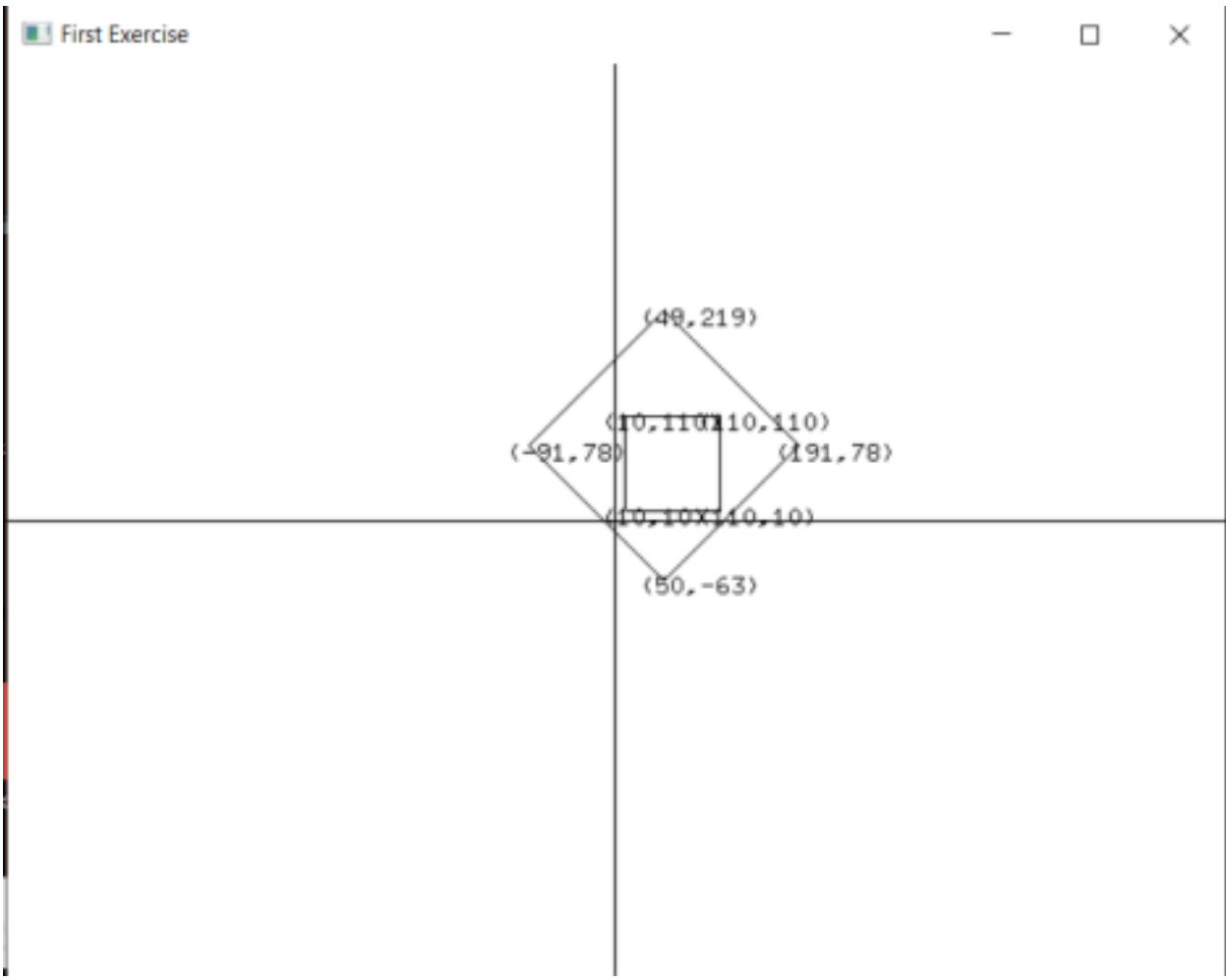




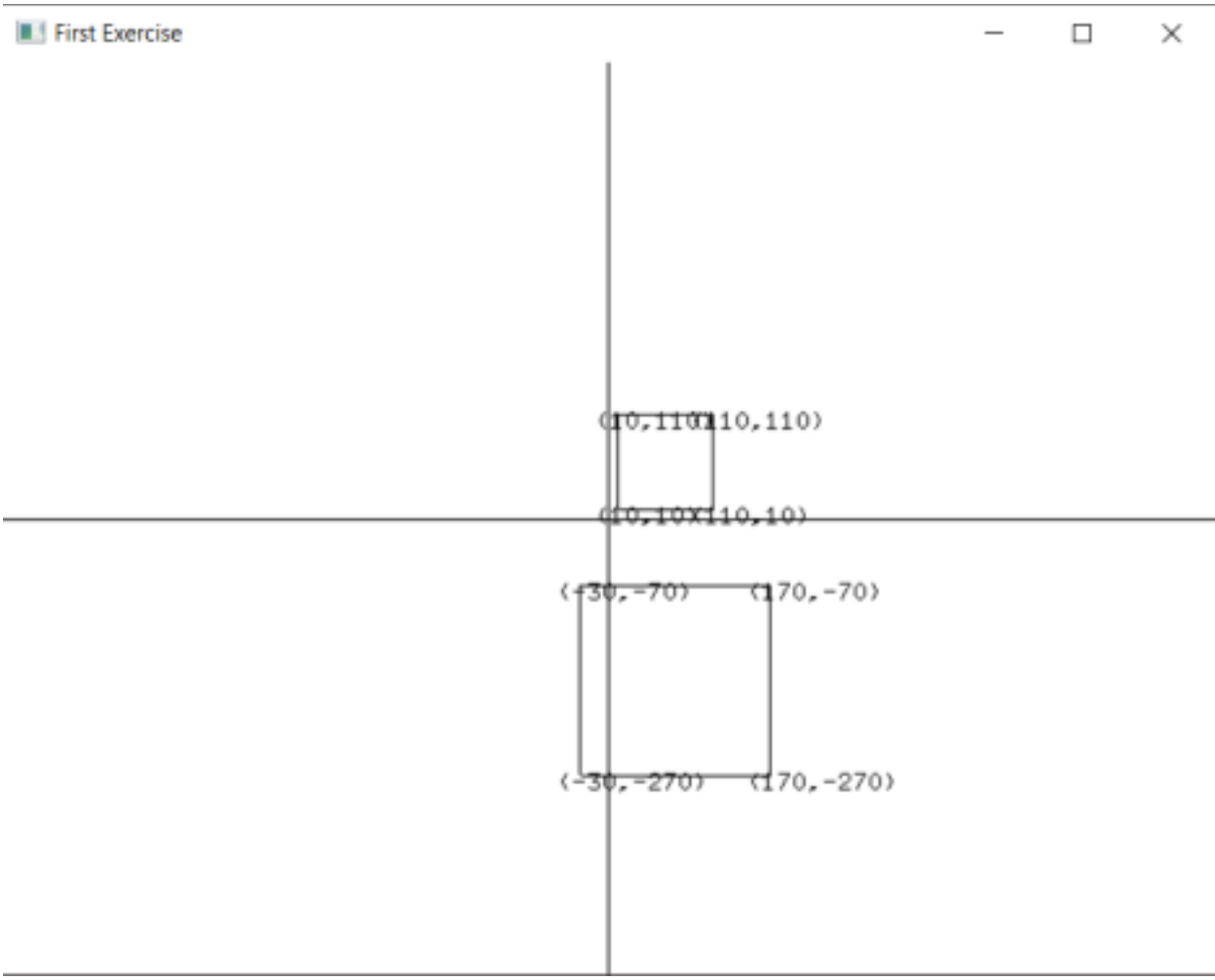
```
D:\SSN\Sem7\Graphics\LAB\GML 6\Debug\GML 6.exe
        2D transformations:
a.Translation
b.Rotation
c.Scaling
d.Reflection
e.Shearing
Enter choice 1: e
Enter choice 2: c
```



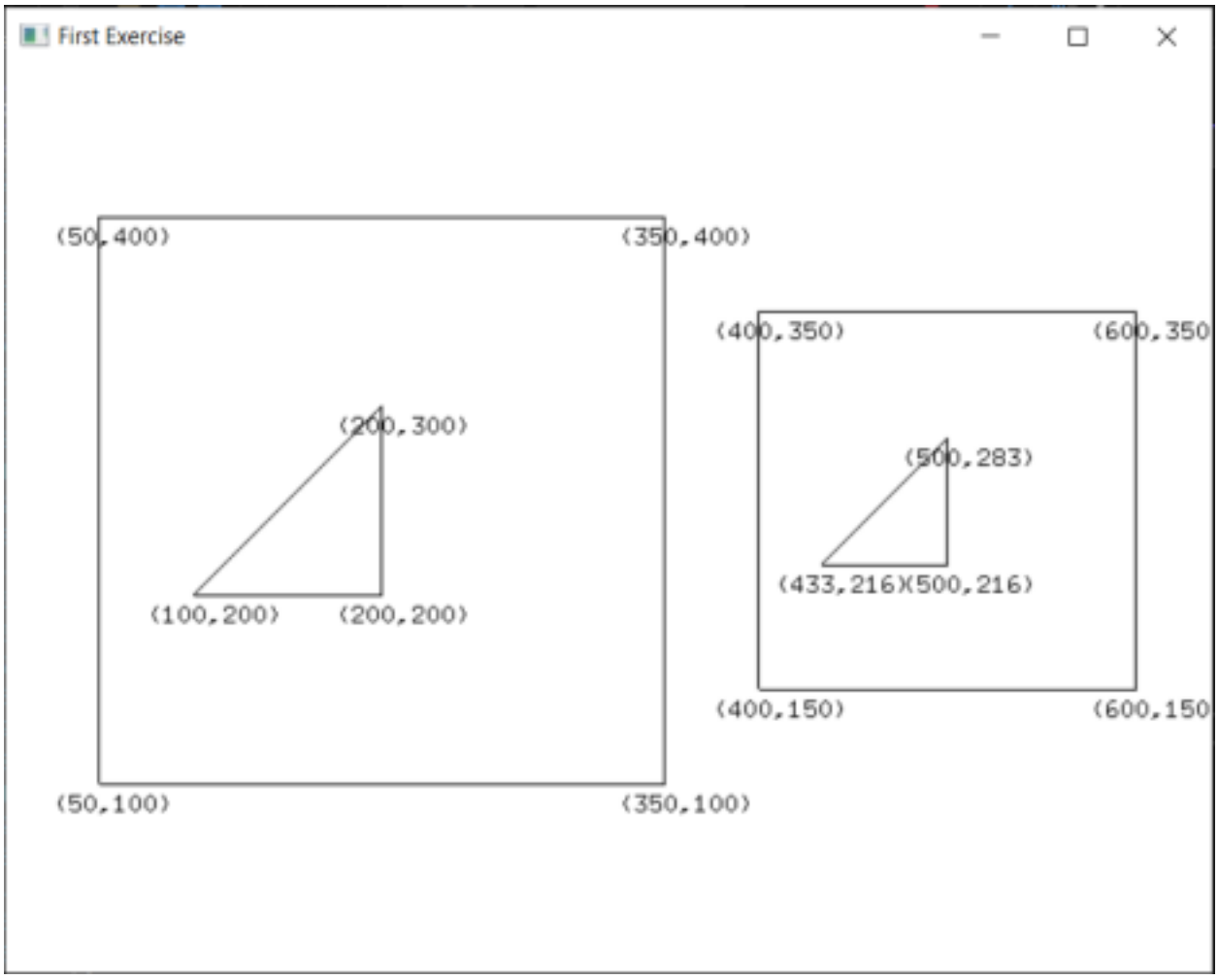
```
        2D transformations:
a.Translation
b.Rotation
c.Scaling
d.Reflection
e.Shearing
Enter choice 1: c
Enter choice 2: b
```




```
        2D transformations:
a.Translation
b.Rotation
c.Scaling
d.Reflection
e.Shearing
Enter choice 1: d
Enter choice 2: c
```



b) Window to viewport transformations


First Exercise
(50,400)
(350,400)
(200,300)
(100,200)
(200,200)
(50,100)
(350,100)
(400,350)
(600,350
(500,283)
(433,216)(500,216)
(400,150)
(600,150

UCS1712 - GRAPHICS AND MULTIMEDIA LAB
ASSIGNMENT 7

QUESTION:

Cohen Sutherland Line clipping in C++ using OpenGL

AIM:

To implement Cohen Sutherland Line clipping algorithm in C++ using OpenGL.

ALGORITHM:

1. Input the coordinates of the clipping window and the endpoints of the line.
2. Draw the clipping window and the line on the screen with color differentiation.
3. Calculate the slope m of the line using m=y2-y1/x2-x1.
4. Find the region code of the 2 end points.
5. Perform bitwise OR operation for the 2 region codes.
    a. If the result is 0000, then it is the case of trivial accept (i.e.) the line is completely inside the clipping window.
    b. Else, if the result is not 0000,
        i. Perform bitwise AND operation for the 2 region codes.
            1. If the result is not 0000, it is the case of trivial reject (i.e.) the line is completely outside the clipping window.
            2. Else, if the result is 0000, the line can be clipped. Follow the clipping procedure stated below.
6. Perform bitwise AND for the 2 points with each of the 4 edges (left,right,bottom,top in that order) and determine the intersection points of the line with the corresponding edge.
7. Clip the part of the line from the initial point to the intersection point and now consider the intersection point as the new point.
8. Repeat steps 5 to 7 until the trivial accept condition becomes true.
9. Draw the points used for clipping on the screen and show the final clipped line segment with color differentiation.
10. For all points on the screen, write the coordinates near the point on the screen

SOURCE CODE:

```
#include <vector>
#include <iostream>
#include <GL/glut.h>
#include <cmath>
#include<string>
#include<Windows.h>
using namespace std;
typedef struct point
```

```
{
   int x, y;
   vector<int> RC{0,0,0,0};
}point;
void write_points(int x1, int y1)
{
   string p1 = "(" + to_string(x1) + "," + to_string(y1) + ")";
   glRasterPos2i(x1 - 25, y1 - 15);
   for (int i = 0; p1[i] != '\0'; i++) {
      glutBitmapCharacter(GLUT_BITMAP_8_BY_13, p1[i]);
   }
}
void point_writer(vector<int> w)
{
   for (int i = 0; i < w.size(); i = i + 2)
   {
      write_points(w[i], w[i + 1]);
   }
}
void plot_area(vector<int> w)
{
   glBegin(GL_LINE_LOOP);
   glColor3f(1,0,0);
   for (int i = 0; i < w.size(); i = i + 2)
   {
      glVertex2d(w[i], w[i + 1]);
   }
   glEnd();
   point_writer(w);
}
void draw_line(vector<int> line)
{
   glBegin(GL_LINES);
   glColor3f(0,1,0);
   glVertex2d(line[0],line[1]);
   glVertex2d(line[2],line[3]);
   glEnd();
   point_writer(line);
}
point regioncode(vector<int> clipwindow,point p)
```

```
{
   int xmin = clipwindow[0], xmax = clipwindow[4];
   int ymin = clipwindow[1], ymax = clipwindow[5];
   if (p.y > ymax)
   {
      p.RC[0] = 1;
   }
   if (p.y < ymin)
   {
      p.RC[1] = 1;
   }
   if (p.x > xmax)
   {
      p.RC[2] = 1;
   }
   if (p.x < xmin)
   {
      p.RC[3] = 1;
   }
   return p;
}
int bitwise_ANDcheck(vector<int> RC, vector<int> edge)
{
   vector<int> rcop(4, 0);
   for (int i = 0; i < 4; i++)
   {
      rcop[i] = RC[i] & edge[i];
      if (rcop[i] == 1)
      {
         return 1;
      }
   }
   return 0;
}
void cohen_sutherland(vector<int> clipwindow, vector<int> line)
{
   int flag = 0;
   int xmin = clipwindow[0], xmax = clipwindow[4];
   int ymin = clipwindow[1], ymax = clipwindow[5];
   point p1, p2;
```

```
    p1.x = line[0]; p1.y = line[1];
    p2.x = line[2]; p2.y = line[3];
    float m = (float)(p2.y - p1.y) / (float)(p2.x - p1.x);
repeat:
    glBegin(GL_POINTS);
    glColor3f(0,0,0);
    glVertex2d(p1.x,p1.y);
    glVertex2d(p2.x, p2.y);
    glEnd();
    write_points(p1.x,p1.y);
    write_points(p2.x, p2.y);
    p1.RC = { 0,0,0,0 };
    p2.RC = { 0,0,0,0 };
    p1 = regioncode(clipwindow, p1);
    p2 = regioncode(clipwindow, p2);
    vector<int> rcop(4, 0);
    for (int i = 0; i < 4; i++)
    {
       rcop[i] = p1.RC[i] | p2.RC[i];
       if (rcop[i] == 1)
       {
          flag = 1;
       }
    }
    if (flag == 1)
    {
       flag = 0;
       for (int i = 0; i < 4; i++)
       {
          rcop[i] = p1.RC[i] & p2.RC[i];
          if (rcop[i] == 1)
          {
             flag = 1;
          }
       }
       if (flag == 1)
       {
          cout << "(" << p1.x << "," << p1.y << ") to (" << p2.x << "," << p2.y << ") is " <<
"COMPLETELY OUTSIDE\n";
       }
```

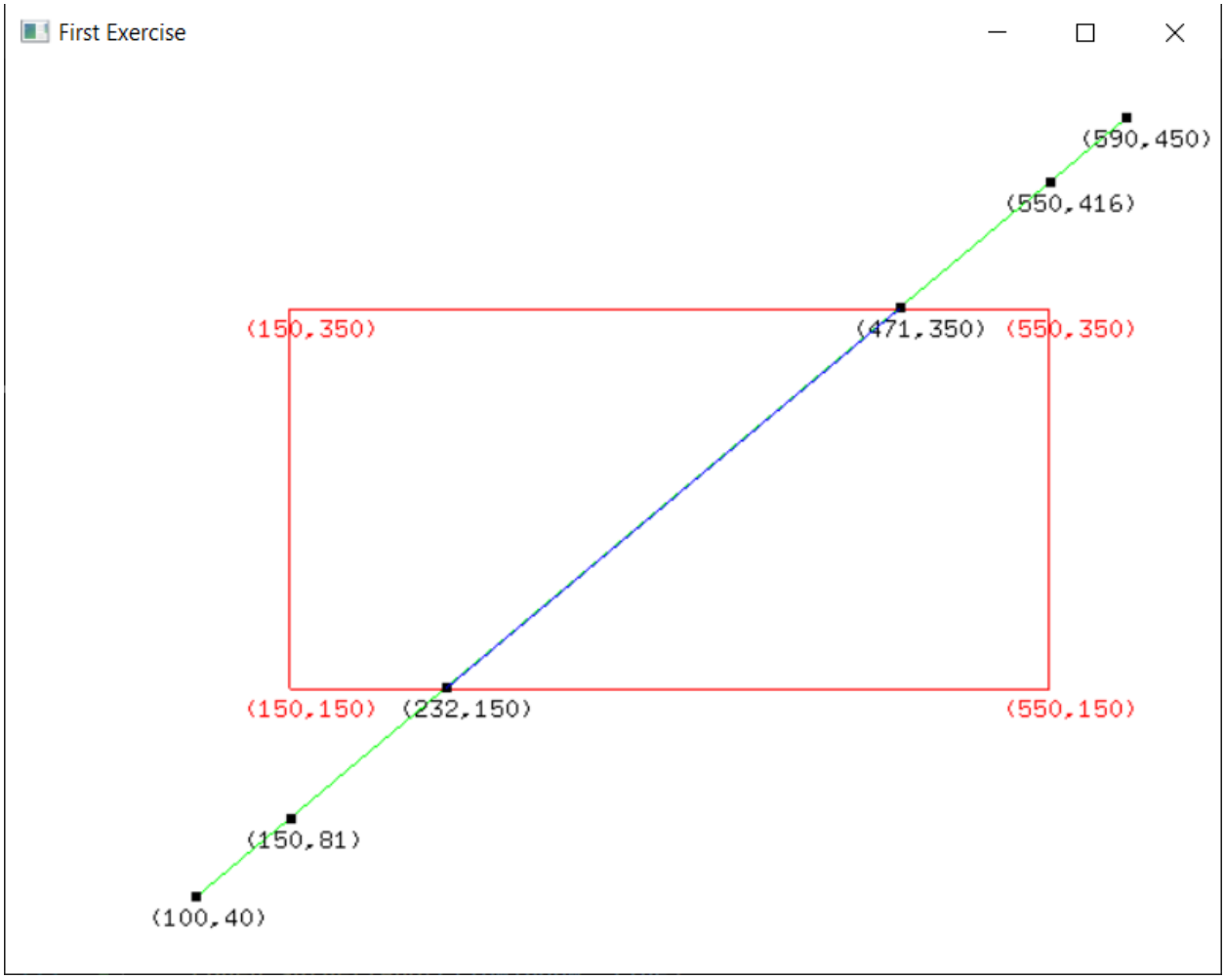
```
        else
        {
            cout << "(" << p1.x << "," << p1.y << ") to (" << p2.x << "," << p2.y << ")  " << "CAN
BE CLIPPED\n";
            //LRBT
            float x, y;
            vector<int> left = { 0,0,0,1 };
            vector<int> right = { 0,0,1,0 };
            vector<int> top = { 1,0,0,0 };
            vector<int> bottom = { 0,1,0,0 };
            if (bitwise_ANDcheck(p1.RC, left) == 1)
            {
                x = xmin;
                y = p1.y + m * (x - p1.x);
                p1.x = x;
                p1.y = y;
                goto repeat;
            }
            if (bitwise_ANDcheck(p1.RC, right) == 1)
            {
                x = xmax;
                y = p1.y + m * (x - p1.x);
                p1.x = x;
                p1.y = y;
                goto repeat;
            }
            if (bitwise_ANDcheck(p1.RC, bottom) == 1)
            {
                y = ymin;
                x = p1.x + (float)(y - p1.y) / (float)m;
                p1.x = x;
                p1.y = y;
                goto repeat;
            }
            if (bitwise_ANDcheck(p1.RC, top) == 1)
            {
                y = ymax;
                x = p1.x + (float)(y - p1.y) / (float)m;
                p1.x = x;
                p1.y = y;
```

```
            goto repeat;
        }
        if (bitwise_ANDcheck(p2.RC, left) == 1)
        {
            x = xmin;
            y = p2.y + m * (x - p2.x);
            p2.x = x;
            p2.y = y;
            goto repeat;
        }
        if (bitwise_ANDcheck(p2.RC, right) == 1)
        {
            x = xmax;
            y = p2.y + m * (x - p2.x);
            p2.x = x;
            p2.y = y;
            goto repeat;
        }
        if (bitwise_ANDcheck(p2.RC, bottom) == 1)
        {
            y = ymin;
            x = p2.x + (float)(y - p2.y) / (float)m;
            p2.x = x;
            p2.y = y;
            goto repeat;
        }
        if (bitwise_ANDcheck(p2.RC, top) == 1)
        {
            y = ymax;
            x = p2.x + (float)(y - p2.y) / (float)m;
            p2.x = x;
            p2.y = y;
            goto repeat;
        }
      }
    }
    else
    {
      cout << "(" << p1.x << "," << p1.y << ") to (" << p2.x << "," << p2.y << ") is " <<
"COMPLETELY INSIDE\n";
```

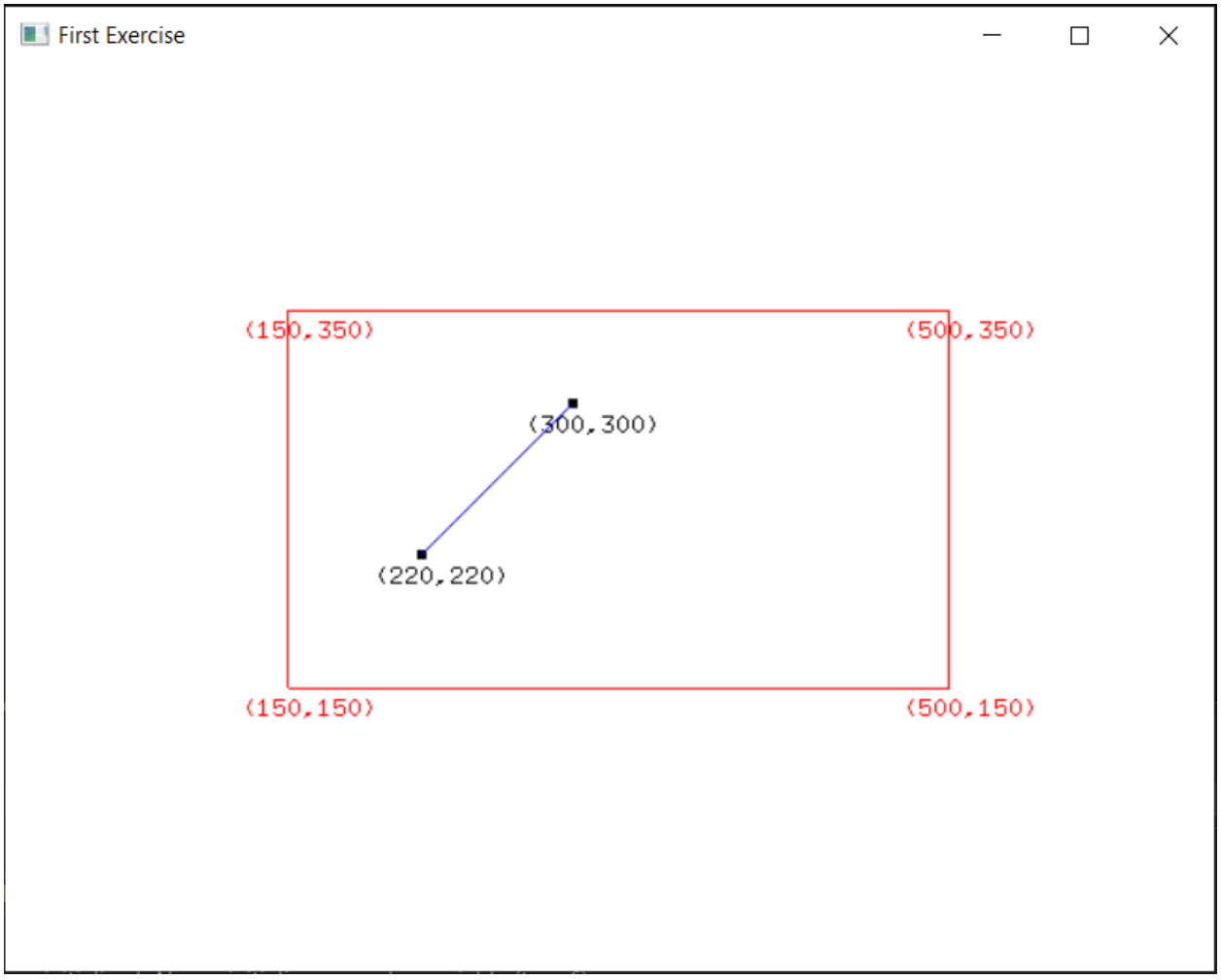
```
        glBegin(GL_LINES);
        glColor3f(0, 0, 1);
        glVertex2d(p1.x,p1.y);
        glVertex2d(p2.x,p2.y);
        glEnd();
    }
}
void operations()
{
    //vector<int> clipwindow={150,150,500,150,500,350,150,350};
    //vector<int> line={220,220,300,300};
    //vector<int> clipwindow={150,150,500,150,500,350,150,350};
    //vector<int> line={0,0,10,10};
    vector<int> clipwindow = { 150,150,550,150,550,350,150,350 };
    plot_area(clipwindow);
    vector<int> line = { 100,40,590,450 };
    draw_line(line);
    cohen_sutherland(clipwindow, line);
}
void myInit()
{
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glColor3f(0.0f, 0.0f, 0.0f);
    glPointSize(5);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
void myDisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);
    operations();
    glFlush();
}
int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutCreateWindow("First Exercise");
```

```
    glutDisplayFunc(myDisplay);
    myInit();
    glutMainLoop();
    return 1;
}
```

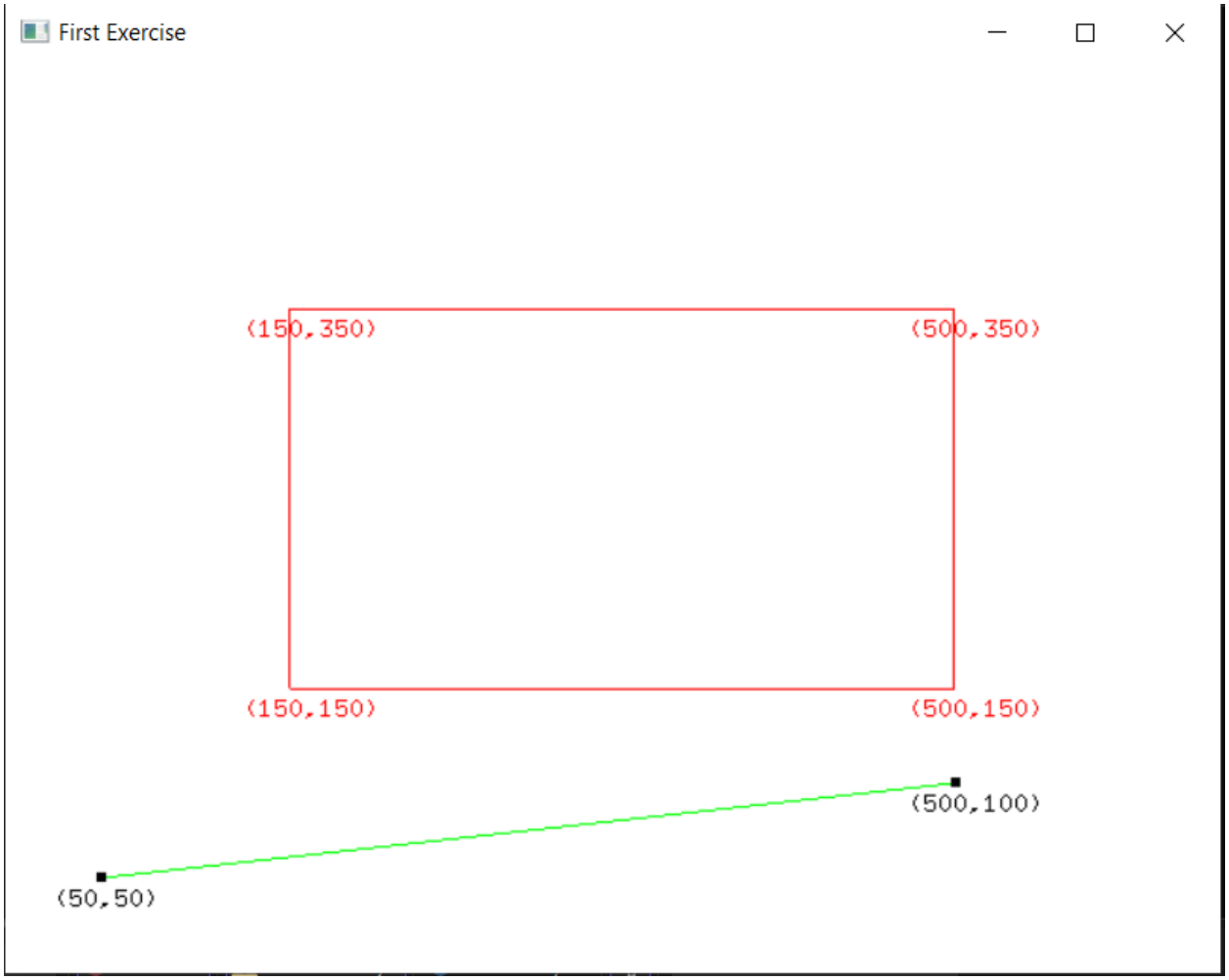
OUTPUT:




```
(100,40) to (590,450)  CAN BE CLIPPED
(150,81) to (590,450)  CAN BE CLIPPED
(232,150) to (590,450)  CAN BE CLIPPED
(232,150) to (550,416)  CAN BE CLIPPED
(232,150) to (471,350) is COMPLETELY INSIDE
```




```
(220,220) to (300,300) is COMPLETELY INSIDE
```




```
(50,50) to (500,100) is COMPLETELY OUTSIDE
```

LEARNING OUTCOMES:

- Inclusion and incorporation of OpenGL and GLUT frameworks.
- Implementation of Cohen Sutherland Line clipping algorithm in C++.
- Usage of vectors for extensive dynamic computations and manipulations.
- Carried out bitwise operations and region code determination for the algorithm.
- Exhibition of all cases of a line clipping scenario with suitable test cases.
- Printing of point coordinates near the point.

Assignmemnt 8:

3-Dimensional Transformations in C++ using OpenGL

Aim:

Perform the following basic 3D Transformations on any 3D Object.

1) Translation
2) Rotation
3) Scaling

Use only homogeneous coordinate representation and matrix multiplication to perform transformations. Set the camera to any position on the 3D space. Have (0,0,0) at the center of the screen. Draw X , Y and Z axis.

Algorithm:

1. Initialize the toolkit using the function glutInit (&argc, argv).
2. Set the display mode and specify the color scheme using the function glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB).
3. Specify the window size using the function glutInitWindowSize(640, 480). 4. Set the starting position for the window using the function glutInitWindowPosition(100,150).
5. Initialize the window and set the title using the function glutCreateWindow("Basic 3d transformation").
6. Clear the screen using the function glClear(GL_COLOR_BUFFER_BIT). 7. SCALING:
    a. glVertex2i(round(pntX[i] * x), round(pntY[i] * y));
8. TRANSLATION:
    a. glVertex2i(pntX[i] + x, pntY[i] + y);
9. ROTATION:
    a. glVertex2i(round((pntX[i] * cos(angleRad)) - (pntY[i] * sin(angleRad))), round((pntX[i] * sin(angleRad)) + (pntY[i] * cos(angleRad))));

Program:

```
#include<iostream>
#include<math.h>
#include<GL/GLUT.h>
using namespace std;
```

```
typedef float Matrix4[4][4];
Matrix4 theMatrix;
static GLfloat input[8][3] =
{
{-50,-50,40},{-100,-50,40},{-100,-100,40},{-50,-100,40
}, {-40,-40,0},{-90,-40,0},{-90,-90,0},{-40,-90,0} };
float output[8][3]; float
tx, ty, tz; float sx, sy,
sz; float angle;
int choice, choiceRot;
void setIdentityM(Matrix4 m)
{
        for (int i = 0; i < 4; i++)
                for (int j = 0; j < 4; j++)
                        m[i][j] = (i == j);
}
void translate(int tx, int ty, int tz)
{
        for (int i = 0; i < 8; i++)
        {
                output[i][0] = input[i][0] + tx;
                output[i][1] = input[i][1] + ty;
                output[i][2] = input[i][2] + tz;
        }
}
void scale(int sx, int sy, int sz)
{
        theMatrix[0][0] = sx;
        theMatrix[1][1] = sy;
        theMatrix[2][2] = sz;
        for (int i = 0; i < 8; i++)
        {
                output[i][0] = input[i][0];
                output[i][1] = input[i][1];
                output[i][2] = input[i][2];
        }
}
void RotateX(float angle) //Parallel to x
{
        angle = angle * 3.142 / 180;
        theMatrix[1][1] = cos(angle);
```

```
	theMatrix[1][2] = -sin(angle);
	theMatrix[2][1] = sin(angle);
	theMatrix[2][2] = cos(angle);
}
void RotateY(float angle) //parallel to y
{
	angle = angle * 3.14 / 180;
	theMatrix[0][0] = cos(angle);
	theMatrix[0][2] = -sin(angle);
	theMatrix[2][0] = sin(angle);
	theMatrix[2][2] = cos(angle);
}
void RotateZ(float angle) //parallel to z
{
	angle = angle * 3.14 / 180;
	theMatrix[0][0] = cos(angle);
	theMatrix[0][1] = sin(angle);
	theMatrix[1][0] = -sin(angle);
	theMatrix[1][1] = cos(angle);
}
void multiplyM()
{
	for (int i = 0; i < 8; i++)
	{
		for (int j = 0; j < 3; j++)
		{
			output[i][j] = 0;
			for (int k = 0; k < 3; k++)
			{
				output[i][j] = output[i][j] + input[i][k] * theMatrix[k][j];
			}
		}
	}
}
void draw(float a[8][3])
{
	glBegin(GL_QUADS);
	glColor3f(0.7, 0.4, 0.5); //behind
	glVertex3fv(a[0]); glVertex3fv(a[1]);
	glVertex3fv(a[2]); glVertex3fv(a[3]);
	glColor3f(0.8, 0.2, 0.4); //bottom
```

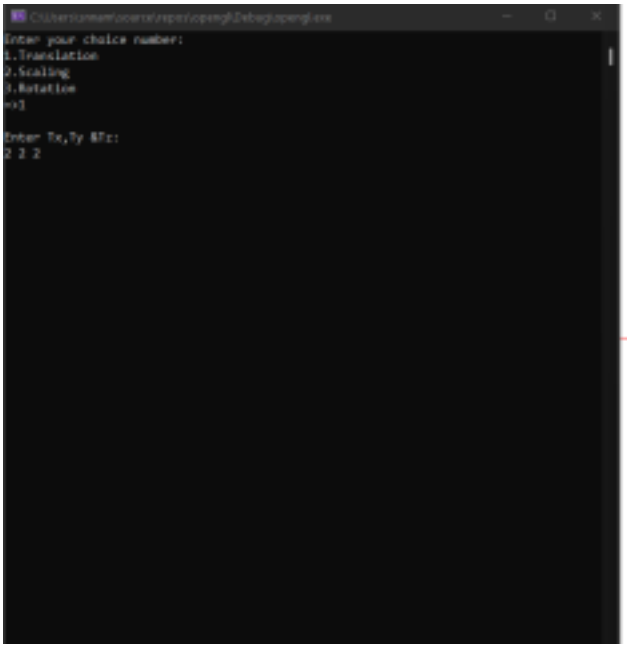
```
        glVertex3fv(a[0]); glVertex3fv(a[1]);
        glVertex3fv(a[5]); glVertex3fv(a[4]);
        glColor3f(0.3, 0.6, 0.7); //left
        glVertex3fv(a[0]); glVertex3fv(a[4]);
        glVertex3fv(a[7]); glVertex3fv(a[3]);
        glColor3f(0.2, 0.8, 0.2); //right
        glVertex3fv(a[1]); glVertex3fv(a[2]);
        glVertex3fv(a[6]); glVertex3fv(a[5]);
        glColor3f(0.7, 0.7, 0.2); //up
        glVertex3fv(a[2]);
        glVertex3fv(a[3]);
        glVertex3fv(a[7]);
        glVertex3fv(a[6]);
        glColor3f(1.0, 0.1, 0.1);
        glVertex3fv(a[4]);
        glVertex3fv(a[5]);
        glVertex3fv(a[6]);
        glVertex3fv(a[7]);
        glEnd();
}
void init()
{
        glClearColor(1.0, 1.0, 1.0, 1.0); //set backgrond color to
        white glOrtho(-500.0, 500.0, -500.0, 500.0, -500.0, 500.0);
        // Set the no. of Co-ordinates along X & Y axes and their
        gappings glEnable(GL_DEPTH_TEST);
        // To Render the surfaces Properly according to their
depths }
void display()
{
        glClear(GL_COLOR_BUFFER_BIT |
        GL_DEPTH_BUFFER_BIT); glColor3f(1.0, 0.0, 0.0);
        glBegin(GL_LINES);
        glVertex3d(0, 0, -500);
        glVertex3d(0, 0, 500);
        glVertex3d(-500, 0, 0);
        glVertex3d(500, 0, 0);
        glVertex3d(0, -500, 0);
        glVertex3d(0, 500, 0);
        glEnd();
```

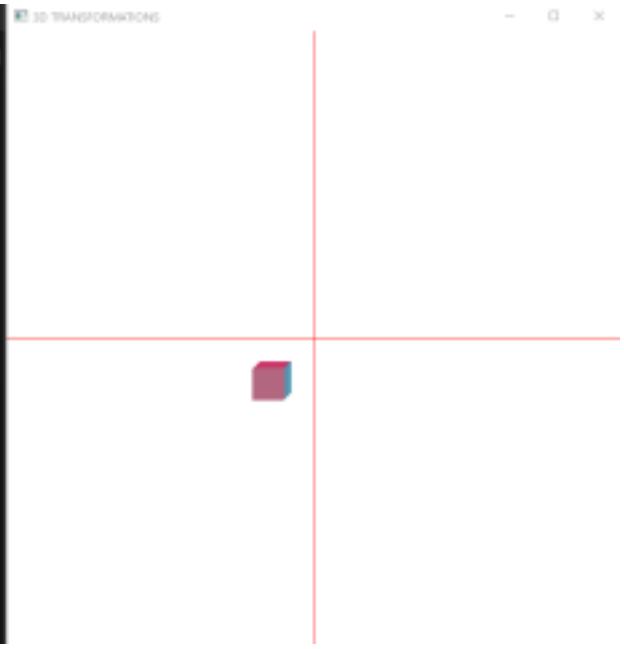
```
        draw(input);
        setIdentityM(theMatrix);
        switch (choice)
        {
        case 1:
                translate(tx, ty, tz); break;
        case 2:
                scale(sx, sy, sz);
                multiplyM(); break;
        case 3:
                switch (choiceRot) {
                case 1:
                        RotateX(angle);
                        break;
                case 2: RotateY(angle);
                        break;
                case 3:
                        RotateZ(angle);
                        break;
                default:
                        break;
                }
                multiplyM();
                break;
        }
        draw(output);
        glFlush();
}
int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
                                                glutInitWindowPosition(0, 0);
                                                glutCreateWindow("3D
                                                TRANSFORMATIONS");
                                                init();
                                                cout << "Enter your choice
                                                number:\n1.Translation\n2.Scaling\n3.Rotation\
                                                n=>";
cin >>
glutInitWindowSize(1362,
750);                                           choice;
```

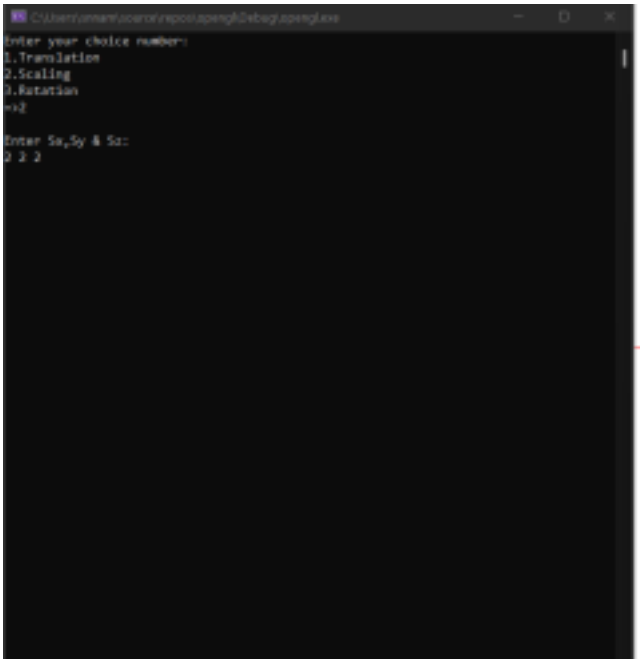
```
switch (choice) {
case 1:
cout << "\nEnter Tx,Ty &Tz: \n"; cin
>> tx >> ty >> tz;
break;
case 2:
cout << "\nEnter Sx,Sy & Sz: \n"; cin
>> sx >> sy >> sz;
break;
case 3:
cout << "Enter your choice for Rotation about
axis:\n1.parallel to X-axis." << "(y&
z)\n2.parallel to Y-axis.(x& z)\n3.parallel to
Z-axis."
<< "(x& y)\n =>";
cin >> choiceRot; switch
(choiceRot) {
                    break;
          }
          glutDisplayFunc(display);
          glutMainLoop();
          return 0;
}
```

```
case 1:
cout << "\nENter Rotation angle: "; cin
>> angle;
break;
case 2:
cout << "\nENter Rotation angle: "; cin
>> angle;
break;
case 3:
cout << "\nENter Rotation angle: "; cin
>> angle;
break;
default:
break;
}
break;
default:
```

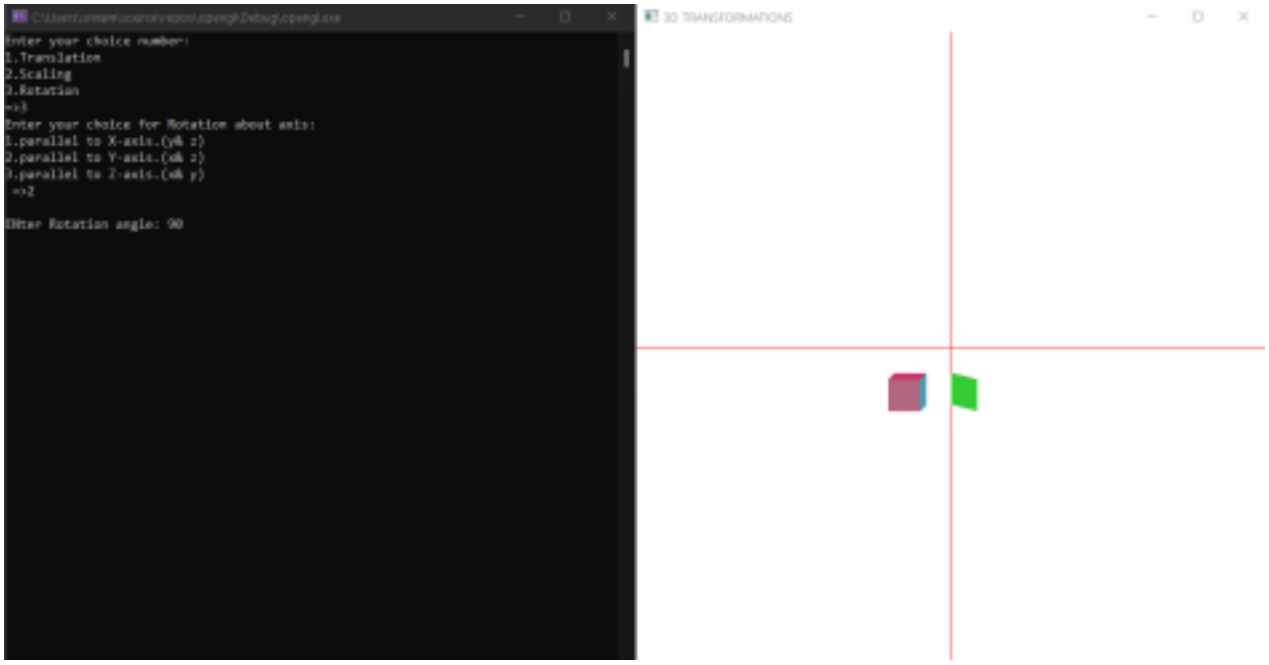
Sample input output for all cases:


C:\Users\vmam\source\repos\opengl\Debug\opengl.exe
Enter your choice number:
1.Translation
2.Scaling
3.Rotation
=>1
Enter Tx,Ty &Tz:
2 2 2


3D TRANSFORMATIONS


C:\Users\vmam\source\repos\opengl\Debug\opengl.exe
Enter your choice number:
1.Translation
2.Scaling
3.Rotation
=>2
Enter Sx,Sy & Sz:
2 2 2


3D TRANSFORMATIONS



Learning outcomes:

- Learnt and implemented 3D transformations in C++ and OpenGL.
- Used OPENGL and GLUT frameworks to visualize the result.

Exercise 9: 3-Dimensional Projections in C++ using OpenGL

AIM:

To write a menu driven program to perform Orthographic parallel projection and Perspective projection on any 3D object.

SOURCE CODE:

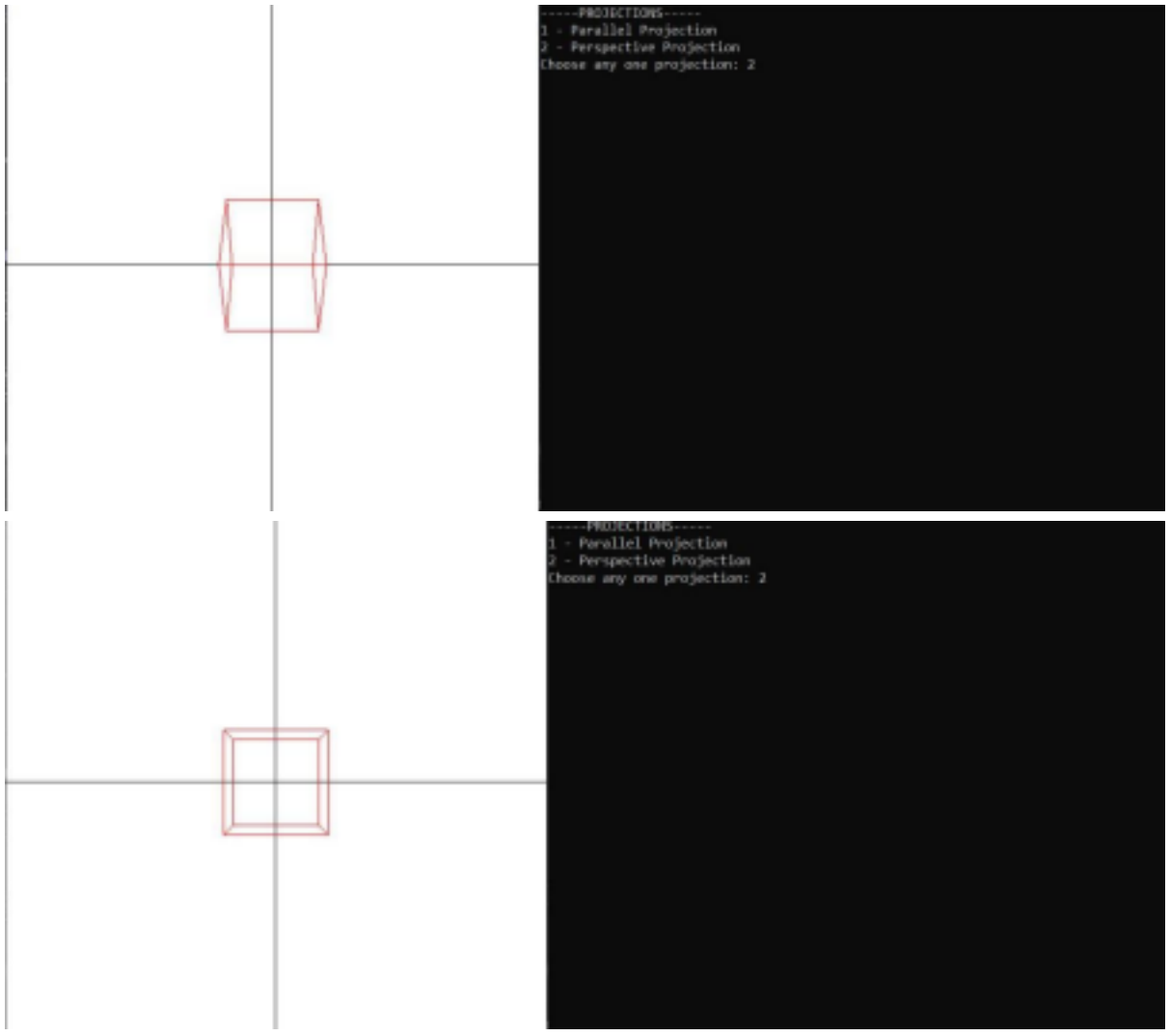
```
#pragma warning(disable : 4996) #include <GL/glut.h>
#include <stdio.h>
#include <stdlib.h>

#include <iostream> using namespace std;

bool* keyStates = new bool[256]; int x_angle = 0, y_angle = 0, z_angle = 0;

void drawAxis() {
glBegin(GL_LINES); glVertex3d(-

500, 0, 0);
glVertex3d(500, 0, 0);

glEnd(); glBegin(GL_LINES); glVertex3d(0, 500, 0); glVertex3d(0,
-500, 0); glEnd();

}
void drawTeapot() {
glClear(GL_COLOR_BUFFER_BIT |

GL_DEPTH_BUFFER_BIT);  glColor3f(0.0, 0.0, 0.0); // X,

Y and Z axis glBegin(GL_LINES);

glVertex3d(-5, 0, 0); glVertex3d(5, 0, 0); glVertex3d(0, 5, 0); glVertex3d(0,
```

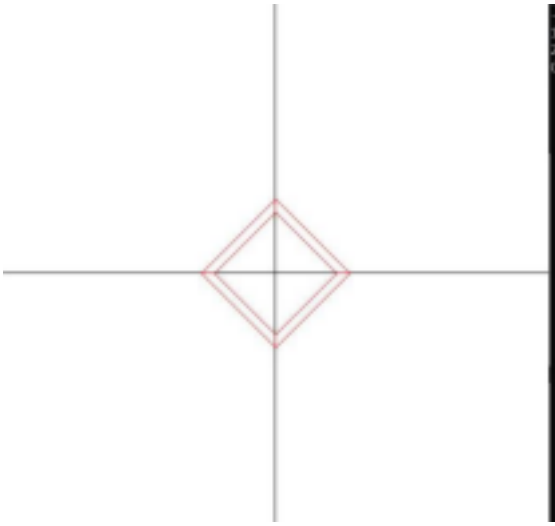
```
0, 1);  glEnd();

glColor3f(1.0, 0.0, 0.0); glLoadIdentity(); glTranslatef(0.0f, 0.0f, -5.0f);
glPushMatrix();  glRotatef(x_angle, 1, 0, 0); glRotatef(y_angle, 0, 1, 0);
glRotatef(z_angle, 0, 0, 1);  glutWireCube(1); glPopMatrix();

glFlush(); }

void keyOperations(void) { int ANGLE_INC = 45; if

(keyStates['w']) {
x_angle += ANGLE_INC;

glVertex3d(0, -5, 0); glVertex3d(0, 0, 100);

}
else if (keyStates['s']) {

x_angle -= ANGLE_INC; }

else if (keyStates['a']) {
y_angle -= ANGLE_INC;

}
else if (keyStates['d']) {

y_angle += ANGLE_INC; }

else if (keyStates[' ']) {
z_angle += ANGLE_INC;

}
x_angle %= 360; y_angle %= 360; z_angle %= 360;

drawTeapot(); }
```

```
void initialize() {
int WIDTH = 500, HEIGHT = 500, choice=1;
cout << "-----PROJECTIONS-----\n1 - Parallel Projection\n2 -

Perspective  Projection\nChoose any one projection: "; cin >> choice;

glClearColor(1.0f, 1.0f, 1.0f, 0.0f);

glViewport(0, 0, WIDTH, HEIGHT);

glMatrixMode(GL_PROJECTION);  glLoadIdentity(); if (choice

== 1) {

glOrtho(-2.0, 2.0, -2.0, 2.0, 1, 100); }

else {
gluPerspective(60, (GLfloat)WIDTH / (GLfloat)HEIGHT, 1,

100.0);  }

glMatrixMode(GL_MODELVIEW); for (int i = 0; i <

256; i++) {  keyStates[i] = false; }

}
void keyPressed(unsigned char key, int x, int y) { keyStates[key] =

true;  keyOperations(); }

void keyUp(unsigned char key, int x, int y) { keyStates[key] =
false; }
int main(int argc, char** argv) {

glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
```

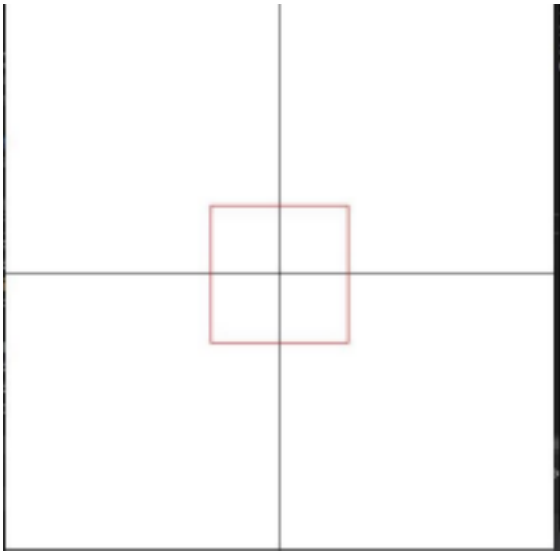
```
    glutInitWindowSize(500, 500);  glutCreateWindow("Projections");
    glutDisplayFunc(drawTeapot);
    drawAxis(); initialize();
glutKeyboardFunc(keyPressed); glutKeyboardUpFunc(keyUp); glutMainLoop(); }
```

INPUT/OUTPUT:

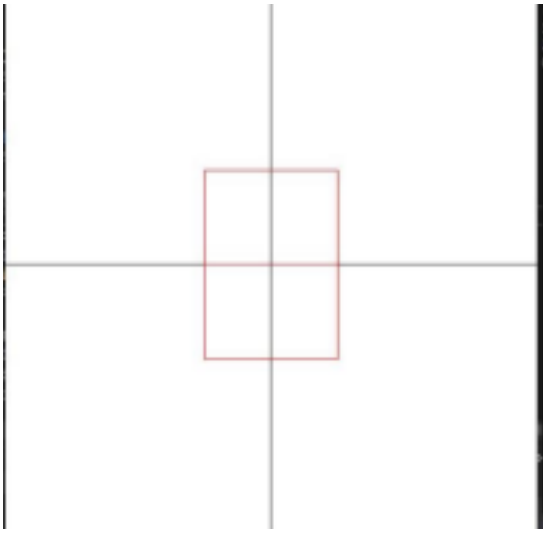




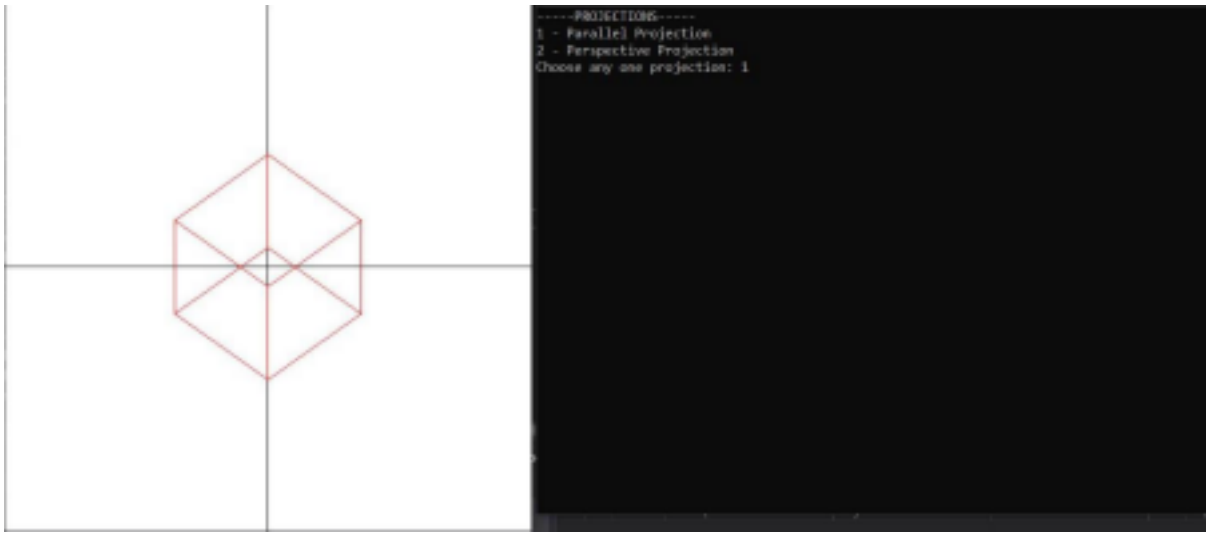
```
-----PROJECTIONS-----
1 - Parallel Projection
2 - Perspective Projection
Choose any one projection: 2
```



```
-----PROJECTIONS-----
1 - Parallel Projection
2 - Perspective Projection
Choose any one projection: 1
```



```
-----PROJECTIONS-----
1 - Parallel Projection
2 - Perspective Projection
Choose any one projection: 1
```


-----PROJECTIONS-----
1 - Parallel Projection
2 - Perspective Projection
Choose any one projection: 1

UCS1712 - GRAPHICS AND MULTIMEDIA LAB
ASSIGNMENT 10

QUESTION:
Write a C++ program using Opengl to draw atleast four 3D objects. Apply lighting and texture and render the scene. Apply transformations to create a simple 3D animation. [Use built-in transformation functions].

AIM:
To write a C++ program using Opengl to draw atleast four 3D objects. Apply lighting and texture and render the scene. Apply transformations to create a simple 3D animation.

SOURCE CODE:

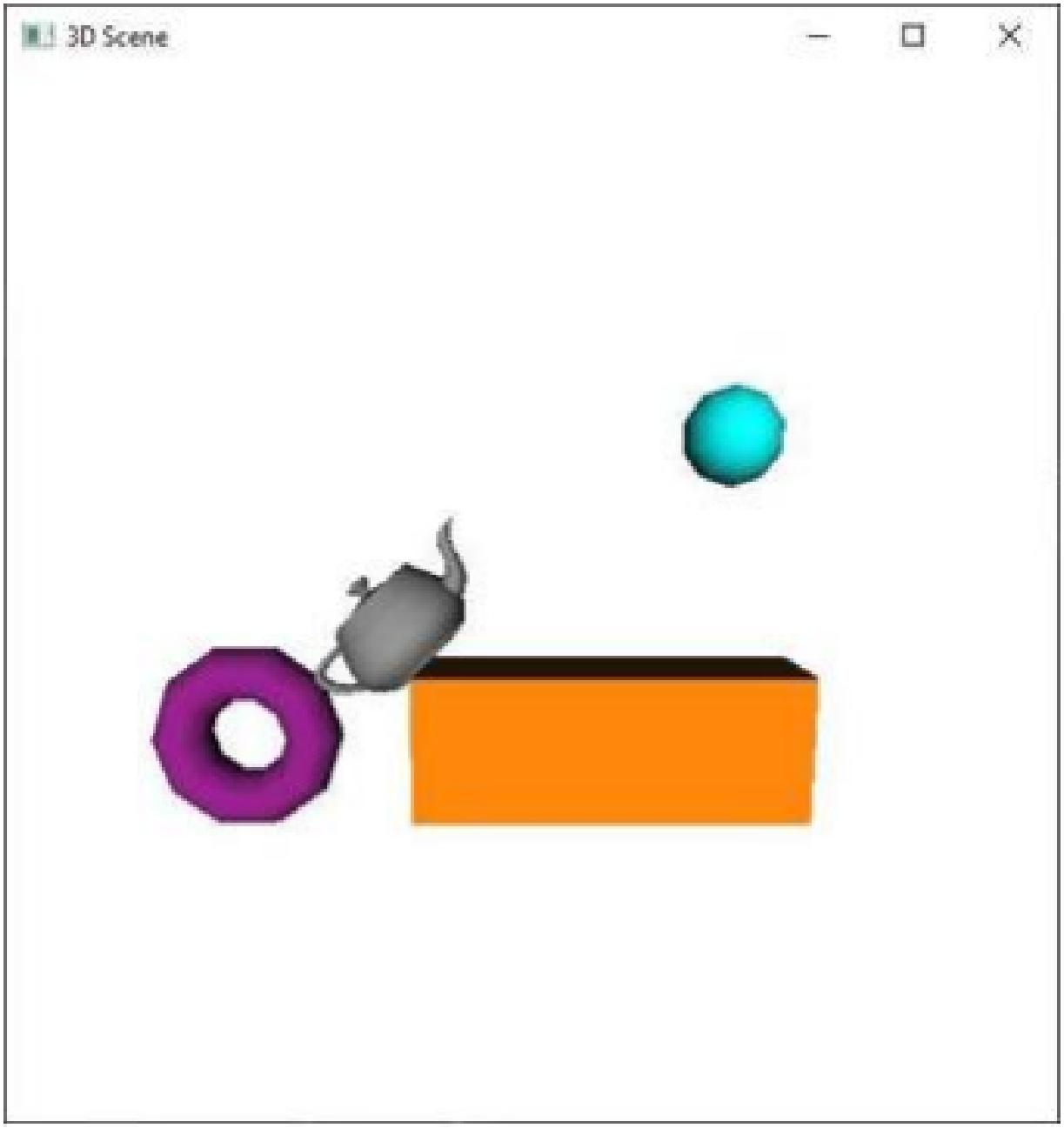
```
#include <vector>
#include <iostream>
#include <GL/glut.h>
#include <cmath>
#include<string>
#include <GL/glu.h>
#include <stdlib.h>
#include <stdio.h>
#include<Windows.h>
using namespace std;
int INC = 1;
void initialize(void)
{
	glClearColor(1.0, 1.0, 1.0, 0.0);
	glShadeModel(GL_SMOOTH);
	GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
	GLfloat light_position[] = { 0, 0, 1, 0 };
	glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
	glLightfv(GL_LIGHT0, GL_POSITION, light_position);
	glEnable(GL_LIGHTING);
	//glEnable(GL_LIGHTi);
	glEnable(GL_DEPTH_TEST);
}
GLuint LoadTexture(const char* filename)
{
```

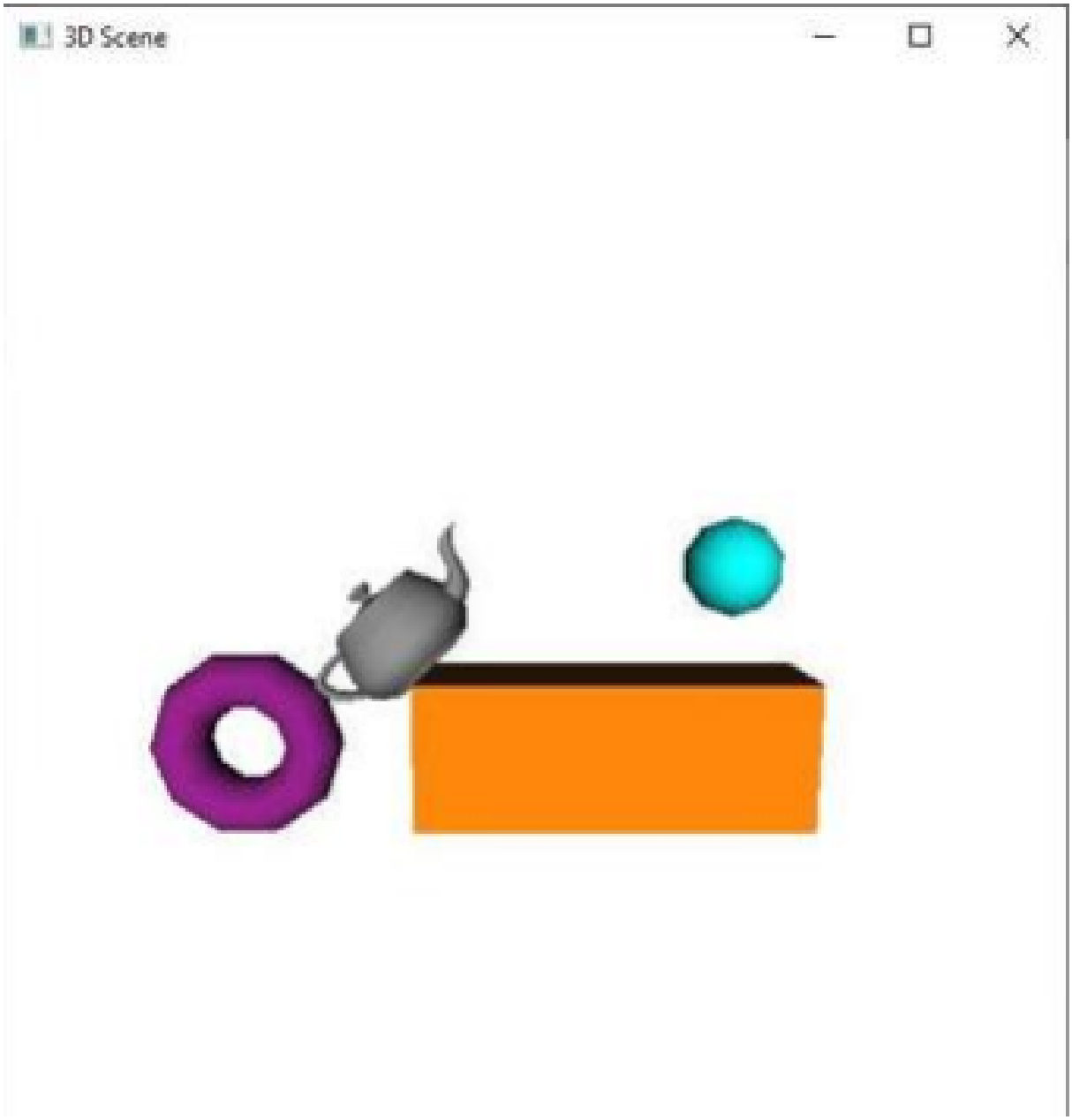
```
	GLuint texture;
	int width, height;
	unsigned char* data;
	FILE* file;
	file = fopen(filename, "rb");
	if (file == NULL) return 0;
	width = 1024;
	height = 512;
	data = (unsigned char*)malloc(width * height * 3);
	//int size = fseek(file,);
	fread(data, width * height * 3, 1, file);
	fclose(file);
	for (int i = 0; i < width * height; ++i)
	{
		int index = i * 3;
		unsigned char B, R; B = data[index];
		R = data[index + 2];
		data[index] = R;
		data[index + 2] = B;
	}
	glGenTextures(1, &texture);
	glBindTexture(GL_TEXTURE_2D, texture);
	glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
	glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_NEAREST);
	glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
	glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	gluBuild2DMipmaps(GL_TEXTURE_2D, 3, width, height, GL_RGB,
GL_UNSIGNED_BYTE, data);
	free(data);
}

void drawScene(int state)
{
	if (state == 0)
		INC = 1;
	else if (state == 10)
		INC = -1;
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

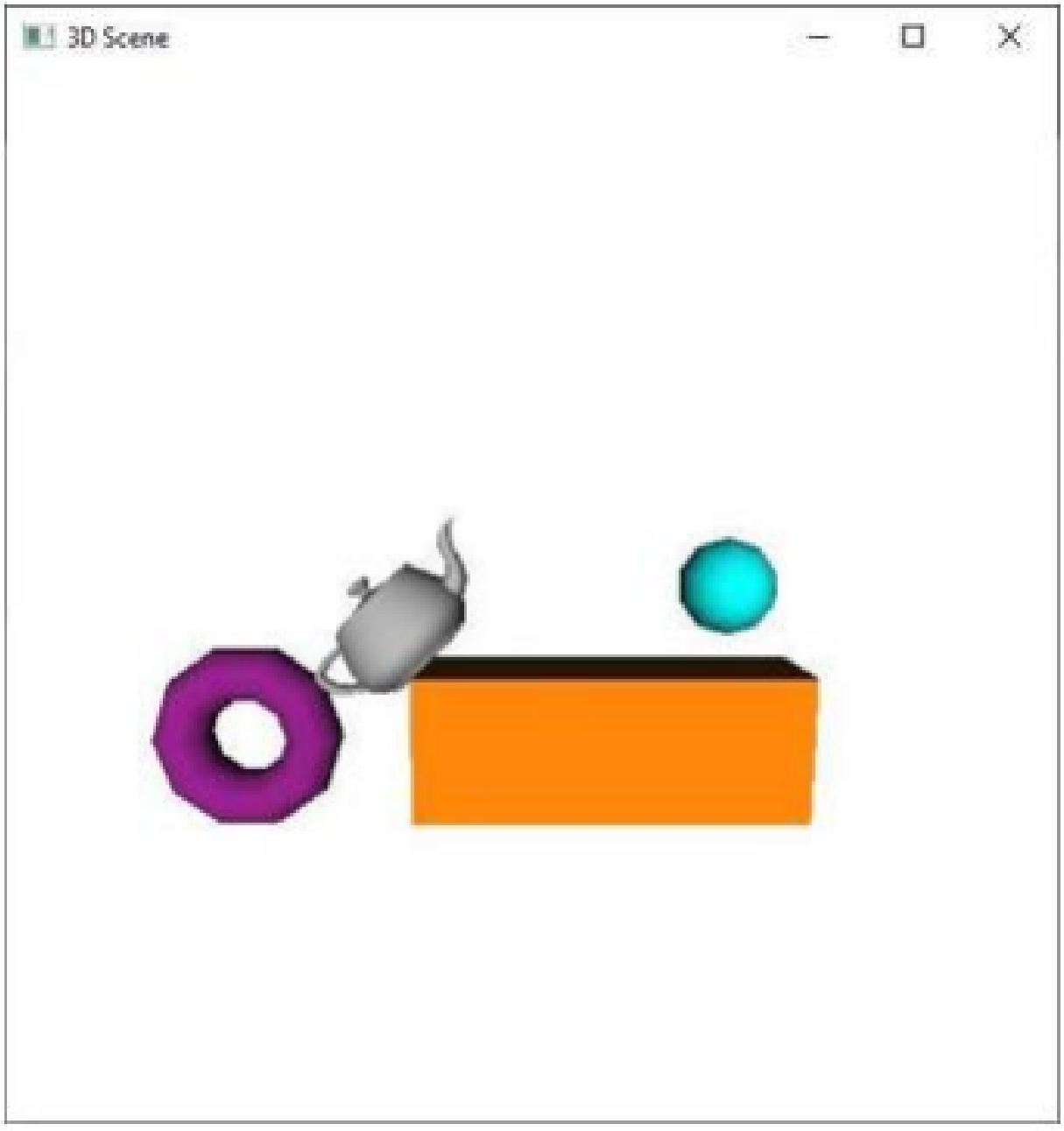
```
GLuint texture;
texture = LoadTexture("silver.bmp");
glBindTexture(GL_TEXTURE_2D, texture);
glLoadIdentity();
gluLookAt(0.0, 1.0, 7.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
glMatrixMode(GL_MODELVIEW);
// Cube
glPushMatrix();
GLfloat cube_color[] = { 1, 0.5, 0.0, 0.0 };
glMaterialfv(GL_FRONT, GL_DIFFUSE, cube_color);
glScalef(4, 1.5, 1.0);
glTranslatef(0.2, -1.0, 0.0);
glutSolidCube(1.0);
glPopMatrix();
// Torus
glPushMatrix();
GLfloat torus_color[] = { 0.59, 0.1, 0.55, 1.0 };
glMaterialfv(GL_FRONT, GL_DIFFUSE, torus_color);
glTranslatef(-3, -1.5, 0.0);
glutSolidTorus(0.3, 0.7, 10, 10);
glPopMatrix();
// Teapot glPushMatrix();
glEnable(GL_TEXTURE_2D);
GLfloat teapot_color[] = { 0.7, 0.7, 0.7, 0.0 };
GLfloat mat_shininess[] = { 100 };
glMaterialfv(GL_FRONT, GL_DIFFUSE, teapot_color);
glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
glRotatef(45, 0, 0, 1);
glTranslatef(-1.2, 0.8, 0.0);
glutSolidTeapot(0.7);
glDisable(GL_TEXTURE_2D);
glPopMatrix();
// Sphere
glPushMatrix();
GLfloat ball_color[] = { 0.0, 1, 1, 1.0 };
glMaterialfv(GL_FRONT, GL_DIFFUSE, ball_color);
glTranslatef(2, 2.1 - 0.25 * state, 0);
glutSolidSphere(0.5, 10, 10);
glPopMatrix();
glutSwapBuffers();
```

```
	glutTimerFunc(1000 / 60, drawScene, state + INC);
}
void reshape(int w, int h)
{
	glViewport(0, 0, (GLsizei)w, (GLsizei)h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluPerspective(75, 1, 1, 20);
}
void sceneDemo() {
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glutTimerFunc(1000 / 60, drawScene, 0);
}
int main(int argc, char** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
	glutInitWindowSize(500, 500);
	glutCreateWindow("3D Scene");
	initialize();
	glutDisplayFunc(sceneDemo);
	glutReshapeFunc(reshape);
	glutMainLoop();
	return 1;
}
```

OUTPUT:


3D Scene


3D Scene



LEARNING OUTCOMES:

- Inclusion and Incorporation of OPENGL and GLUT frameworks and libraries.
- Drawing 3D objects.
- Applying lighting and texture using required functions to render the scene.
- Implementing transformations to create a simple 3D animation.

UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Ex-11: Image Editing and Manipulation

AIM

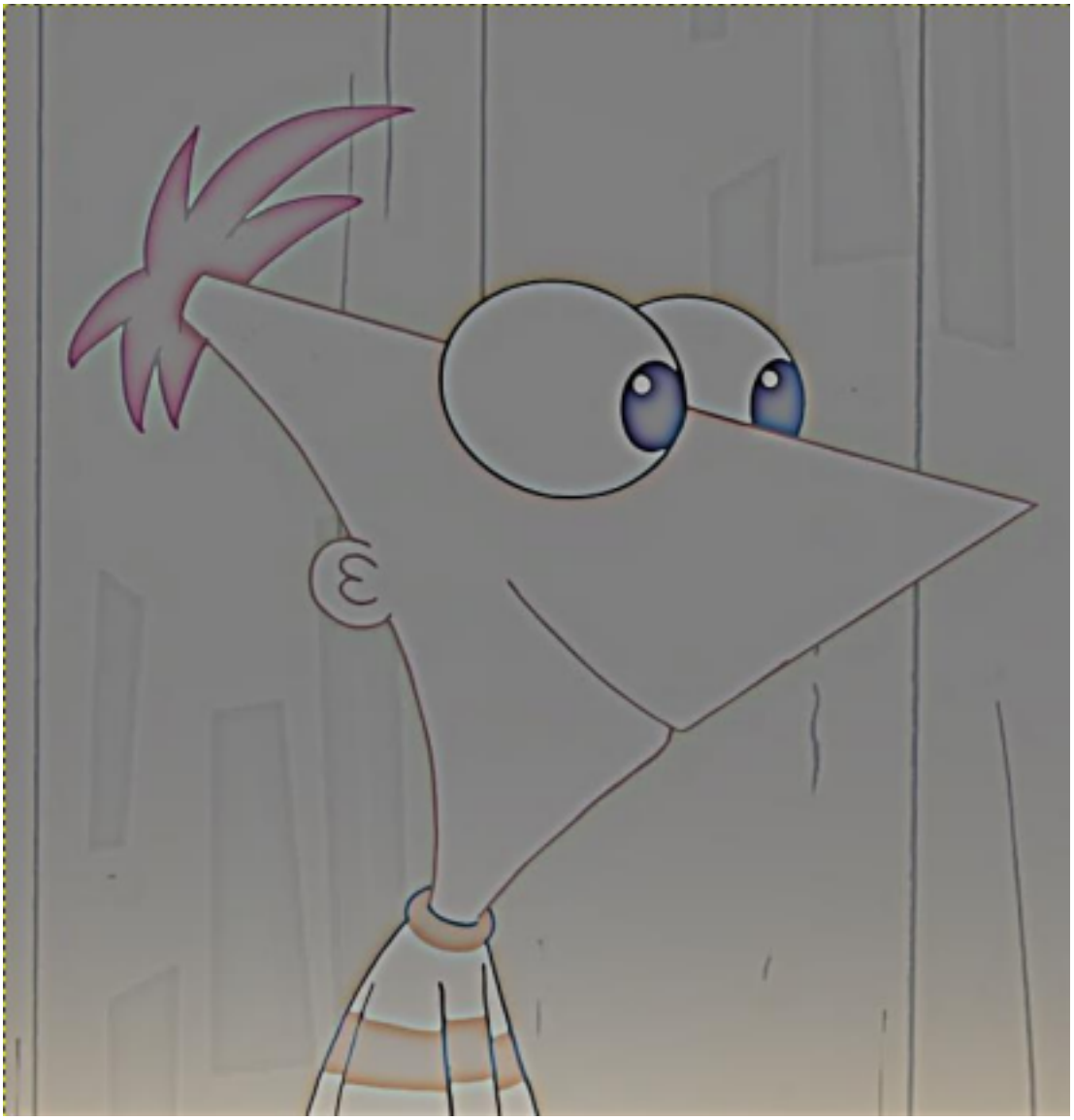
To Perform basic operations on an image (retouch image, add filters, masks, etc) using GIMP software to enhance it.

OUTPUT:

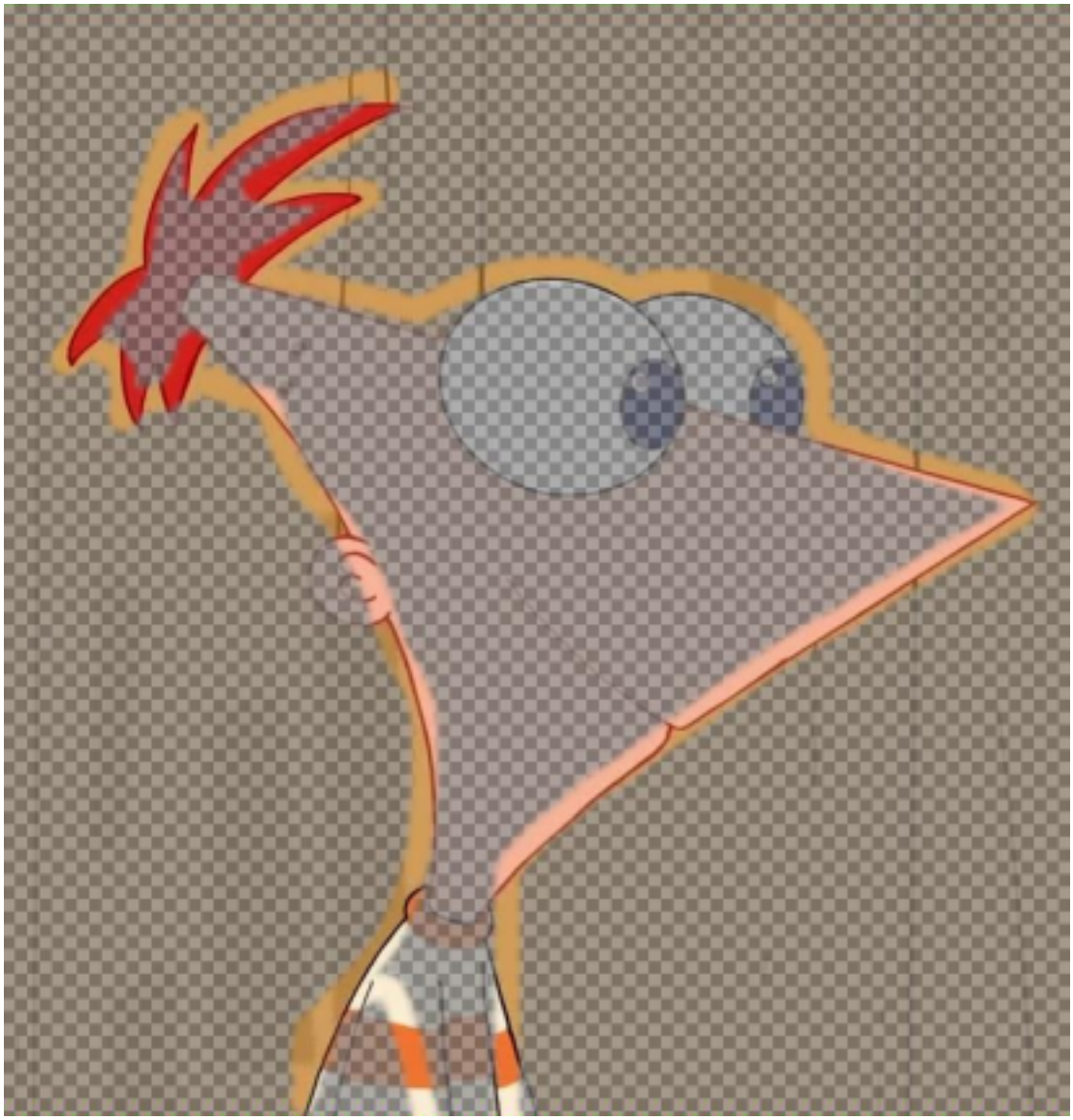
ORGINAL IMAGE

RETOUCH

FILTER



LAYER MASK



RESULT:
Thus, Image Editing and Manipulation was implemented successfully.

1)b.

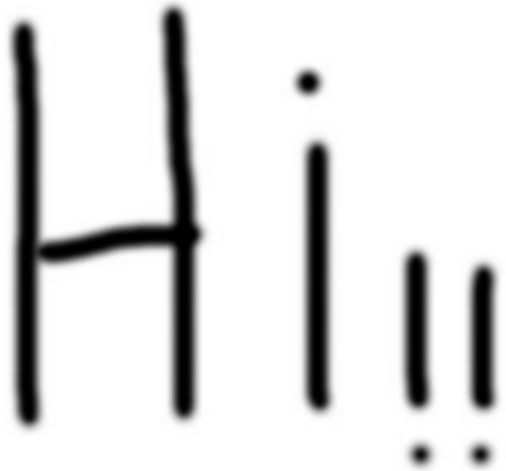
AIM

To Perform basic operations on an image (retouch image, add filters, masks, etc) using GIMP software to enhance it.

OUTPUT:

FRAME-1

FRAME-2


Hi!!

RESULT:

Thus, Simple gif was animated successfully.

UCS1712 – GRAPHICS AND MULTIMEDIA LAB

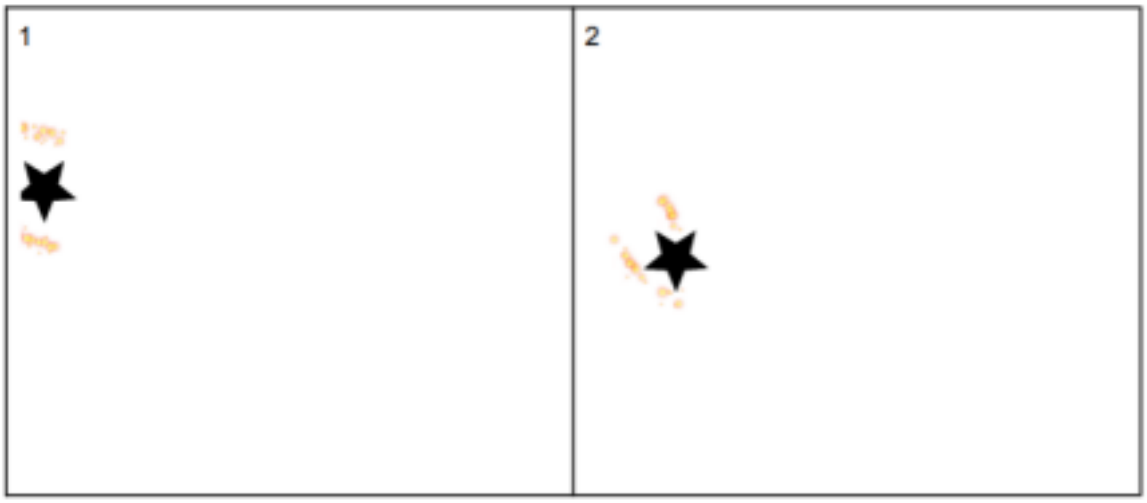
Ex-12: 2D Animation

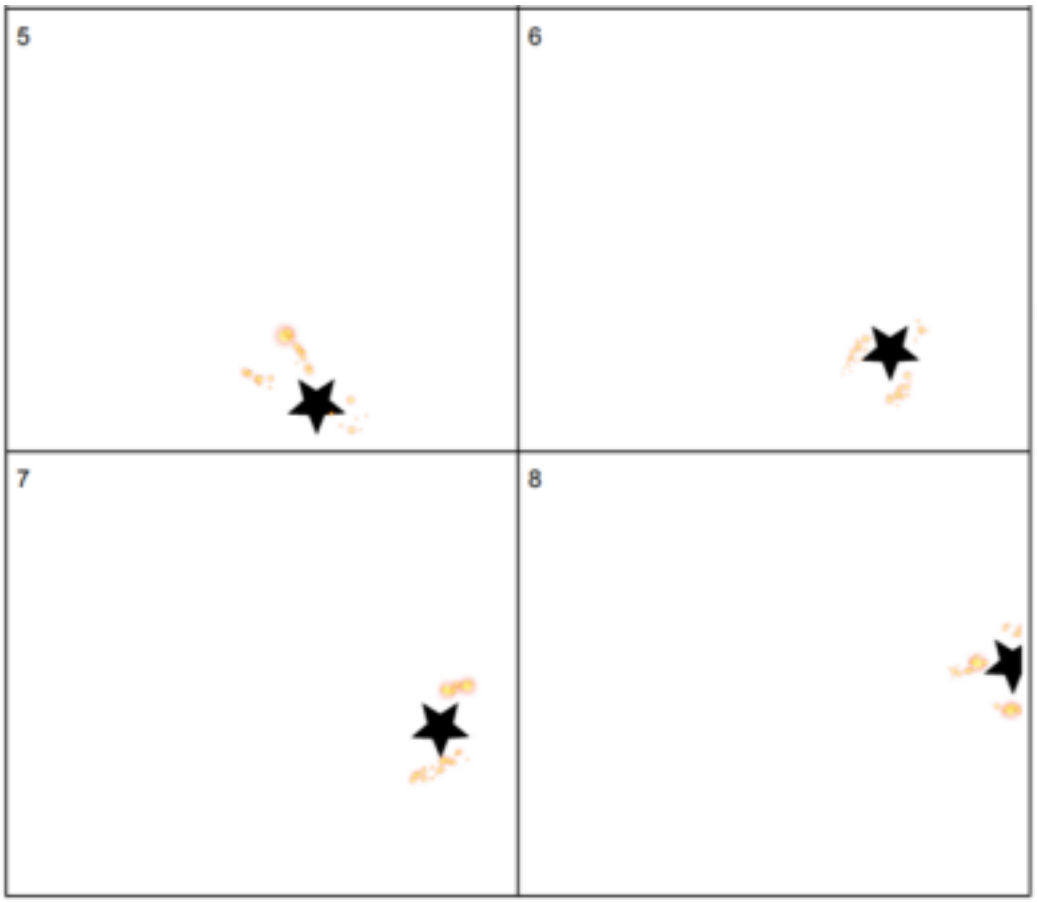
AIM

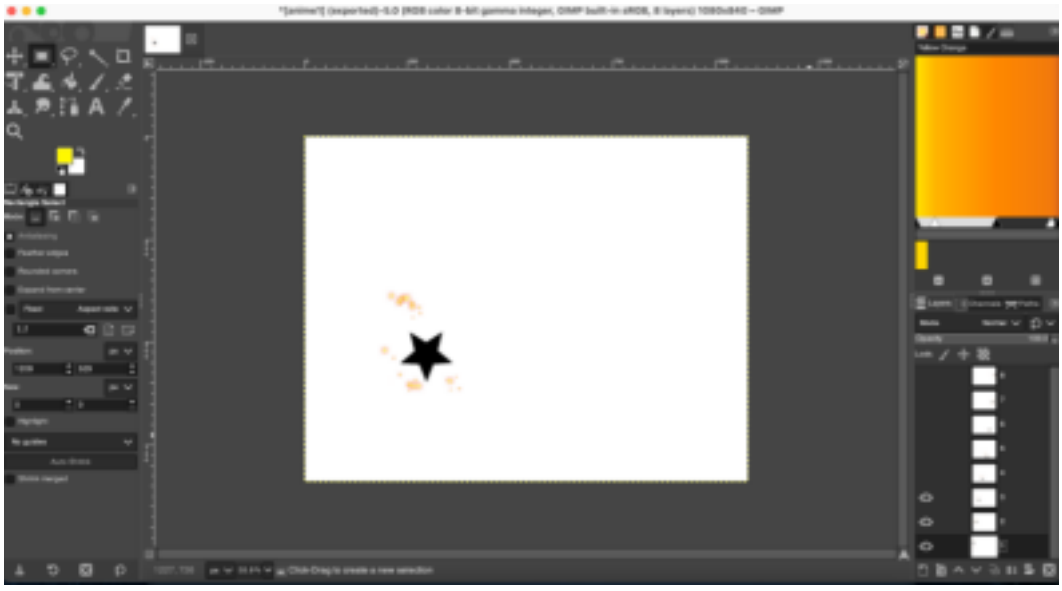
To create a simple 2D animation sequence video with minimum of 2 objects using GIMP Software.

OUTPUT:

FRAMES







RESULT:

Thus, 2D animation was done using gimp

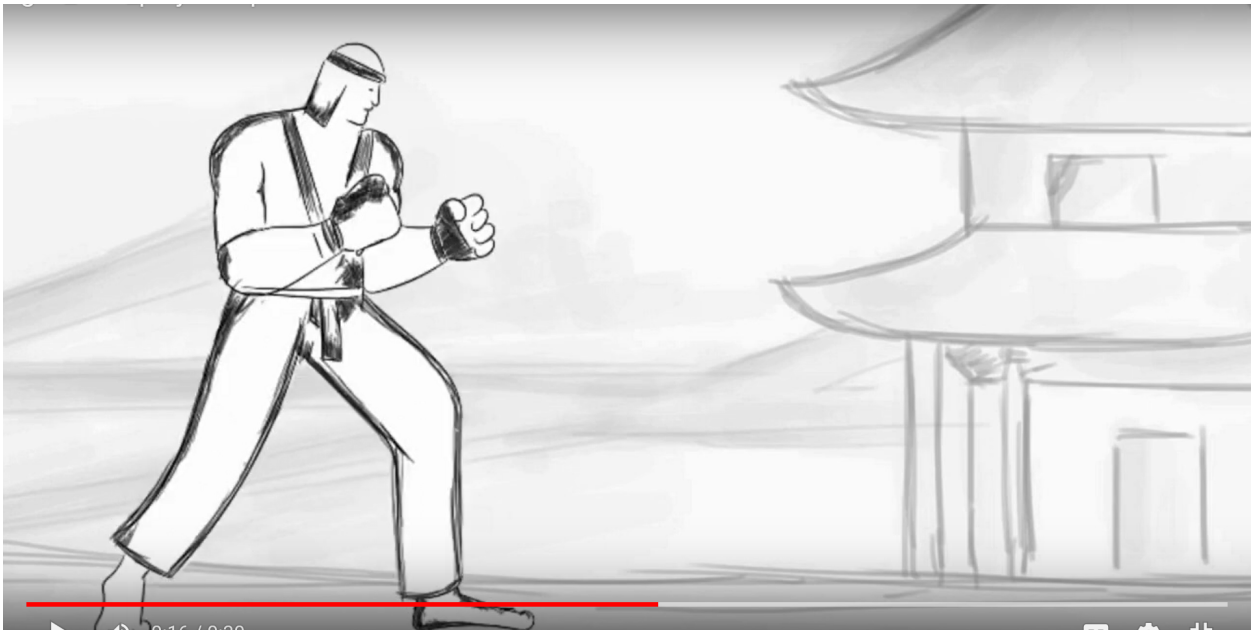
GML - Mini Project Report

Project Title: Martial Art Warrior
Software used: Adobe After Effects

Steps:

1. Create a new project in Adobe After Effects.
2. Create/download drawn sketches for a static martial art warrior.
3. Create a set of 8 images for the waiting position.
4. Create a set of 8 images for the kicking position
5. Create a set of 8 images for going backward action.
6. Create a set of 8 images for moving forward action.
7. Create a set of 8 images for punching pose.
8. Open the control options and create initial settings for the animation.
9. Create background image
10. Compile the order of the images correctly to maintain the sequence.
11. Add the images to the timeline such that each of them acts as a frame
12. Adjust the positions of the keyframes such that aspect ratio is maintained.
13. Drag the pointer along the timeline to check for ease of the animation.
14. Set the timer along the timeline for each action.
15. Adjust the keyframes and check for lags in the animation.
16. Add music (optional)

Screenshots




gml_mini_project.mp4