Computer Graphics Lab

Submitted By -

Name: Arjav Jain
SAP ID: 500083556
Roll No.: R214220227
Batch: B5-H (AIML)
Specialization: B.tech(CSE Hons.) Spl. Artificial Intelligence and ML

Submitted to –
Dr Niharika Singh(Course Instructor)

School of Computer Science
University of Petroleum and Energy Studies
Dehradun - 248007
Year 2020-24

Experiment 1

Introduction to OpenGL: [Virtual Lab Environment Setup]

• What is OpenGL?
OpenGL is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering.
• What is GLU/GLUT?
GLU is the OpenGL Utility Library. This is a set of functions to create texture mipmaps from a base image, map coordinates between screen and object space, and draw quadric surfaces.
• What is OpenGL Architecture?
The architecture of OpenGL is based on a client-server model. An application program written to use the OpenGL API is the "client" and runs on the CPU. The implementation of the OpenGL graphics engine (including the GLSL shader programs you will write) is the "server" and runs on the GPU. Geometry and many other types of attributes are stored in buffers called Vertex Buffer Objects (or VBOs).

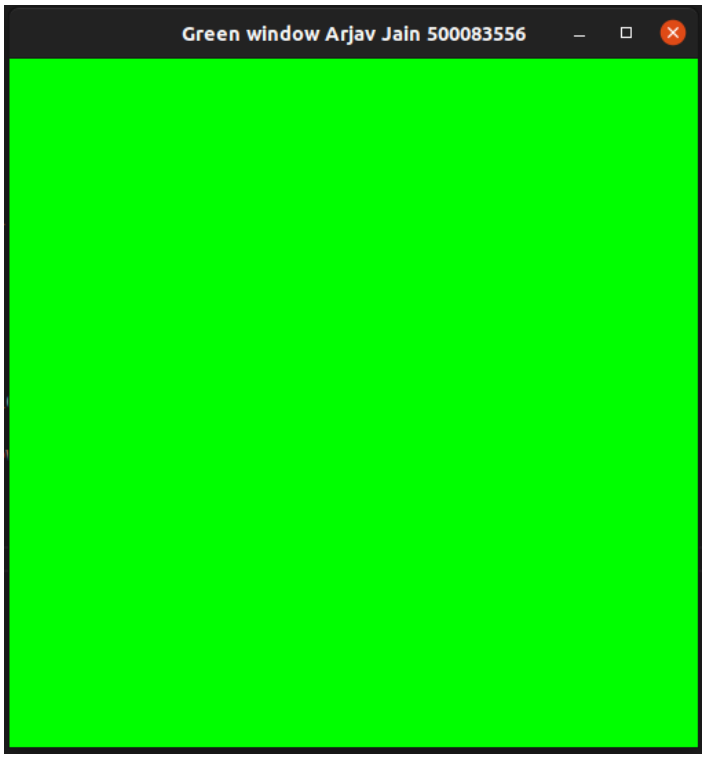
Q) First OpenGL Program: This initialises a window of Green colour.

Code:

```
#include<GL/glut.h>
#include<stdio.h>
void init(){
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(640,480);
glutInitWindowPosition(1000,200);
glutCreateWindow("Simple Window");
}
void display()
{
glClearColor(0.0,1.0,0.0,0.0);
glClear(GL_COLOR_BUFFER_BIT);
glFlush();
}
int main(int argc,char **argv)
```

```
{
glutInit(&argc,argv);
init();
glutDisplayFunc(display);
glutMainLoop();
return 0;
}
```

Output:



Experiment 2

Drawing a line

Q) Draw a line using the equation of line Y=m*X+C.

Code:

```
#include <stdio.h>
#include <GL/glut.h>
void init()
{
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(640, 480);
   glutInitWindowPosition(100, 200);
   glutCreateWindow("Straight Line Arjav Jain 500083556");
   glClearColor(1.0, 1.0, 1.0, 1.0);
   glClear(GL_COLOR_BUFFER_BIT);
   gluOrtho2D(0, 100, 0, 100);
   glLoadIdentity();
   glMatrixMode(GL_PROJECTION);
}

void display()
{
   glColor3f(0, 0, 0);
   glClear(GL_COLOR_BUFFER_BIT);

   float x1, x2, y1, y2;

   printf("Enter x1:");
   scanf("%f", &x1);

   printf("Enter y1:");
   scanf("%f", &y1);

   printf("Enter x2:");
   scanf("%f", &x2);
```

```
    printf("Enter y2:");
    scanf("%f", &y2);

    float m, c, y;
    m = (y2 - y1) / (x2 - x1);
    c = (y1 - m) * x1;

    glBegin(GL_POINTS);
    for (float x = x1; x <= x2; x = x + 0.001)
    {
        y = m * x + c;
        glVertex2f(x, y);
    }
    glEnd();
    glFlush();
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Output:

Q) Draw a line using DDA algorithm for slope m<1 and m>1.

Code:

```
#include <stdio.h>
#include <GL/glut.h>
void init()
{
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(640, 480);
   glutInitWindowPosition(100, 200);
   glutCreateWindow("DDA Arjav Jain 500083556");

   glClearColor(1.0, 1.0, 1.0, 1.0);
   glClear(GL_COLOR_BUFFER_BIT);
   gluOrtho2D(0, 100, 0, 100);
   glLoadIdentity();
   glMatrixMode(GL_PROJECTION);
}

void display()
{
   glColor3f(0, 0, 0);
   glClear(GL_COLOR_BUFFER_BIT);

   float x1, x2, y1, y2;

   printf("Enter x1:");
   scanf("%f", &x1);

   printf("Enter y1:");
   scanf("%f", &y1);

   printf("Enter x2:");
   scanf("%f", &x2);

   printf("Enter y2:");
   scanf("%f", &y2);

   float m, dx, dy, x_inc, y_inc;
```

```
    dx = x2 - x1;
    dy = y2 - y1;
    m = dy / dx;

    int steps = (abs(dx) > abs(dy)) ? dx : dy;

    x_inc = dx / (float)steps;
    y_inc = dy / (float)steps;

    float x, y;

    x = x1;
    y = y1;
    glBegin(GL_POINTS);

    for (int i = 0; i < steps; i++)
    {
        glVertex2f(x, y);
        x += x_inc;
        y += y_inc;
    }
    glEnd();

    glFlush();
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    init();
    glutDisplayFunc(display);
    glutMainLoop();

    return 0;
}
```

Output:

Q) Draw a line using Bresenham algorithm for slope m<1 and m>1.

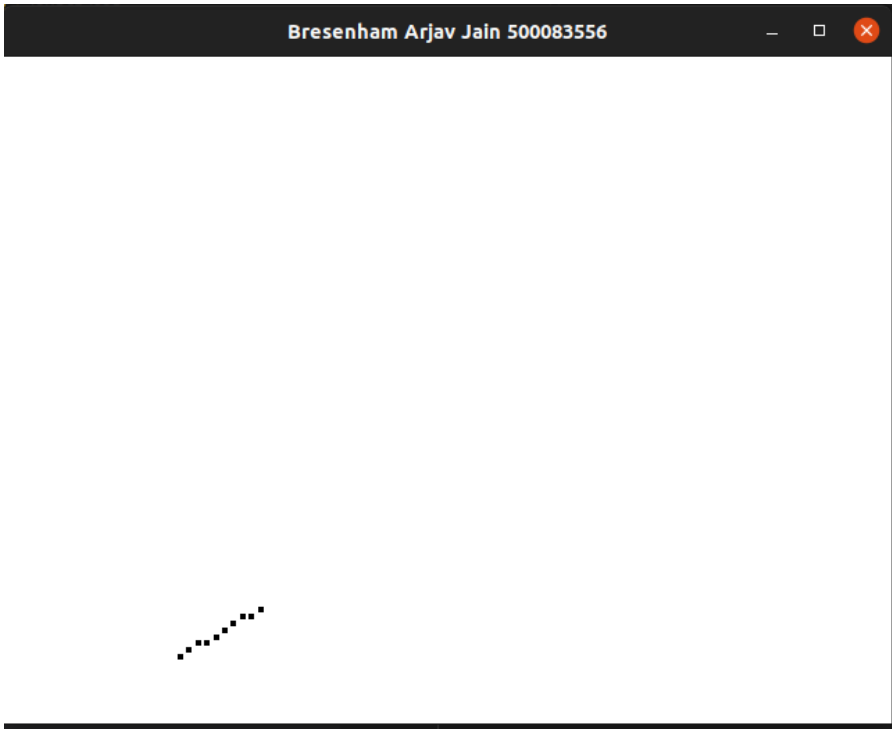
Code:

```
#include <stdio.h>
#include <GL/glut.h>
void init()
{
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(640, 480);
   glutInitWindowPosition(100, 200);
   glutCreateWindow("Bresenham Arjav Jain 500083556");

   glClear(GL_COLOR_BUFFER_BIT);
   glClearColor(1.0, 1.0, 1.0, 1.0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(0, 100, 0, 100);
}

void display()
{
   glColor3f(0, 0, 0);
   glClear(GL_COLOR_BUFFER_BIT);

   float x1, x2, y1, y2;

   printf("Enter x1:");
   scanf("%f", &x1);

   printf("Enter y1:");
   scanf("%f", &y1);

   printf("Enter x2:");
   scanf("%f", &x2);

   printf("Enter y2:");
   scanf("%f", &y2);

   int m, dx, dy, x_inc, y_inc, p;
```

```
dx = x2 - x1;
dy = y2 - y1;
m = dy / dx;

p = (2 * dy) - dx;
glBegin(GL_POINTS);

if (m < 1)
{
   while (x1 < x2)
   {
      if (p > 0)
      {
         glVertex2i(x1, y1);
         y1 += 1;
         p = p + (2 * dy) - (2 * dx);
      }
      else
      {
         glVertex2i(x1, y1);
         p = p + (2 * dy);
      }
      x1 += 1;
   }
}
else
{
   while (y1 < y2)
   {
      if (p > 0)
      {
         glVertex2i(x1, y1);
         x1 += 1;
         p = p + (2 * dx) - (2 * dy);
      }
      else
      {
         glVertex2i(x1, y1);
         p = p + (2 * dx);
      }
```

```
            y1 += 1;
        }
    }
    glEnd();
    glFlush();
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    init();
    glutDisplayFunc(display);
    glutMainLoop();

    return 0;
}
```

Output:



Experiment 3
Drawing a Circle and an Ellipse

Q)Draw the circle with the help of polar equations

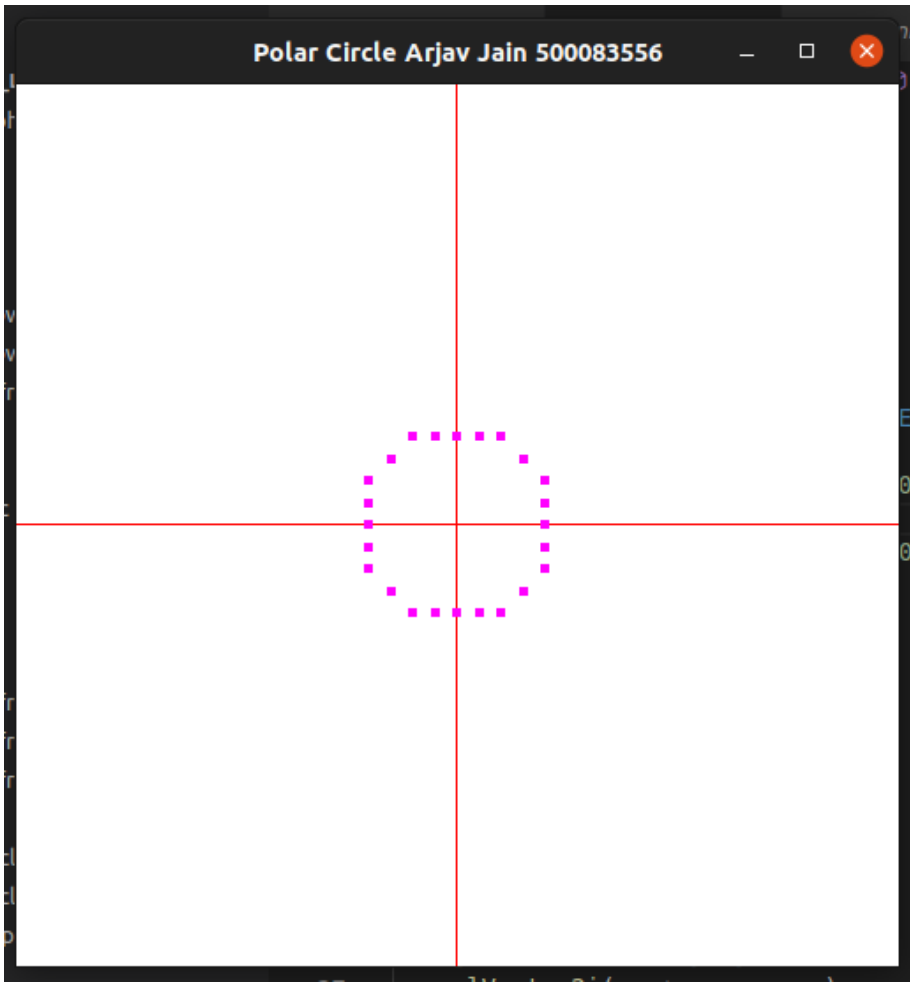
Code:

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

int xc, yc, r;

void init()
{

   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(500, 500);
   glutInitWindowPosition(200, 100);
   glutCreateWindow("Polar Circle Arjav Jain 500083556");
   glClearColor(1.0, 1.0, 1.0, 1.0);
   gluOrtho2D(-20, 20, -20, 20);
}

void putpixel(int x, int y)
{

   glPointSize(5.0);
   glColor3f(1.0, 0.0, 1.0);
   glBegin(GL_POINTS);
   glVertex2i(xc + x, yc + y);
   glVertex2i(xc + x, yc - y);
   glVertex2i(xc + y, yc + x);
   glVertex2i(xc + y, yc - x);
   glVertex2i(xc - x, yc - y);
   glVertex2i(xc - y, yc - x);
   glVertex2i(xc - x, yc + y);
   glVertex2i(xc - y, yc + x);
```

```
    glEnd();
}

void display1()
{

    float x, y;
    x = 0;
    y = r;
    float theta = 0;
    float inc = (float)1 / r;

    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_LINES);
    glVertex2i(-20, 0);
    glVertex2i(20, 0);
    glEnd();
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_LINES);
    glVertex2i(0, -20);
    glVertex2i(0, 20);
    glEnd();

    float end = 3.14 / 4;
    float C = cos(inc);
    float S = sin(inc);

    while (theta <= end)
    {

        float xtemp = x;
        x = x * C - y * S;
        y = y * C + xtemp * S;
        putpixel(x, y);
        theta = theta + inc;
    }
    glFlush();
}
int main(int argc, char **argv)
```

```
{

   printf("Enter the coordinates of the circle's center:");
   scanf("%d %d", &xc, &yc);
   printf("Enter the radius of the circle:");
   scanf("%d", &r);

   glutInit(&argc, argv);
   init();
   glutDisplayFunc(display1);
   glutMainLoop();

   return 0;
}
```

Output:



Q) Draw the circle with the help of mid-point method.

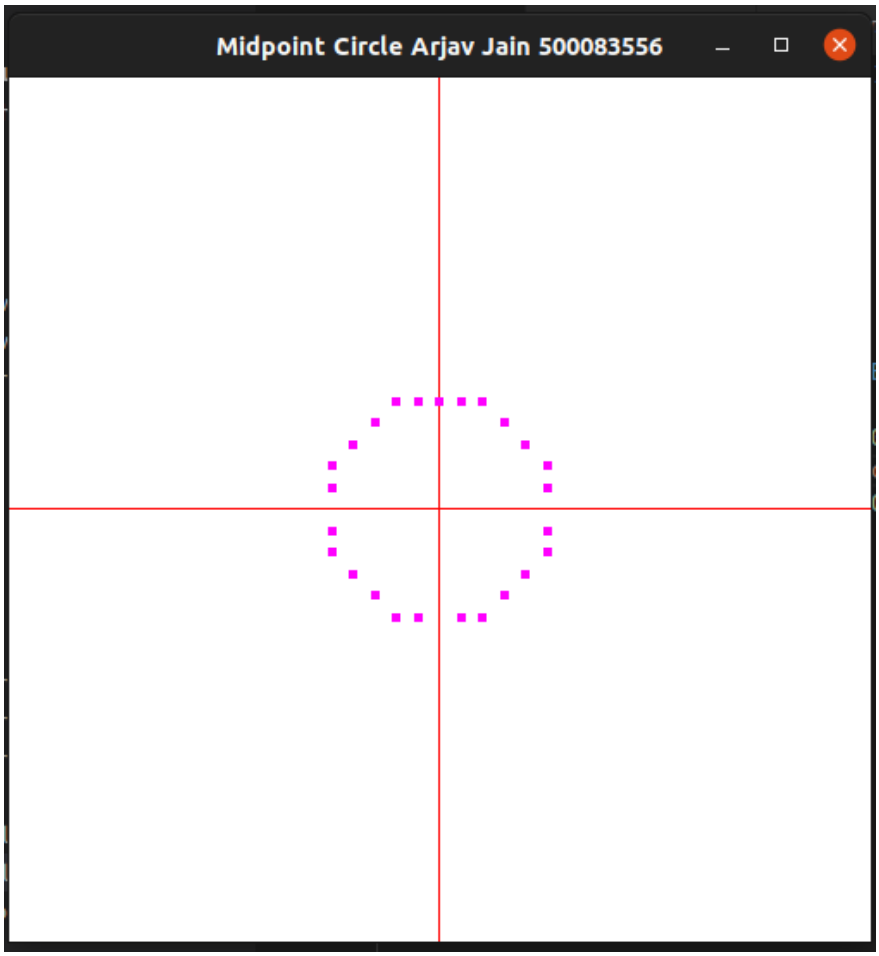
Code:

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

int xc;
int yc;
int r;
void init()
{
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(500, 500);
   glutInitWindowPosition(200, 100);
   glutCreateWindow("Midpoint Circle Arjav Jain 500083556");
   glClearColor(1.0, 1.0, 1.0, 1.0);
   gluOrtho2D(-20, 20, -20, 20);
}
void putpixel(int x, int y)
{
   glPointSize(5.0);
   glColor3f(1.0, 0.0, 1.0);
   glBegin(GL_POINTS);
   glVertex2i(x + xc, y + yc);
   glEnd();
}
void display()
{
   int x = 0;
   int y = r;
   float dec = 1 - r;
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 0.0, 0.0);
   glBegin(GL_LINES);
   glVertex2i(-20, 0);
   glVertex2i(20, 0);
   glEnd();
   glColor3f(1.0, 0.0, 0.0);
   glBegin(GL_LINES);
```

```
    glVertex2i(0, -20);
    glVertex2i(0, 20);
    glEnd();

    putpixel(x, y);

    while (y > x)
    {
        if (dec < 0)
        {
            x++;
            dec += 2 * x + 1;
        }
        else
        {
            y--;
            x++;
            dec += 2 * (x - y) + 1;
        }
        putpixel(x, y);
        putpixel(x, -y);
        putpixel(-x, y);
        putpixel(-x, -y);
        putpixel(y, x);
        putpixel(-y, x);
        putpixel(y, -x);
        putpixel(-y, -x);
    }

    glFlush();
}
int main(int argc, char **argv)
{

    printf("Enter the coordinates of the circle's center:");
    scanf("%d %d", &xc, &yc);
    printf("Enter the radius of the circle:");
    scanf("%d", &r);
    glutInit(&argc, argv);
    init();
```

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

Output:



Q) Draw the Ellipse with the mid-point method.

Code:

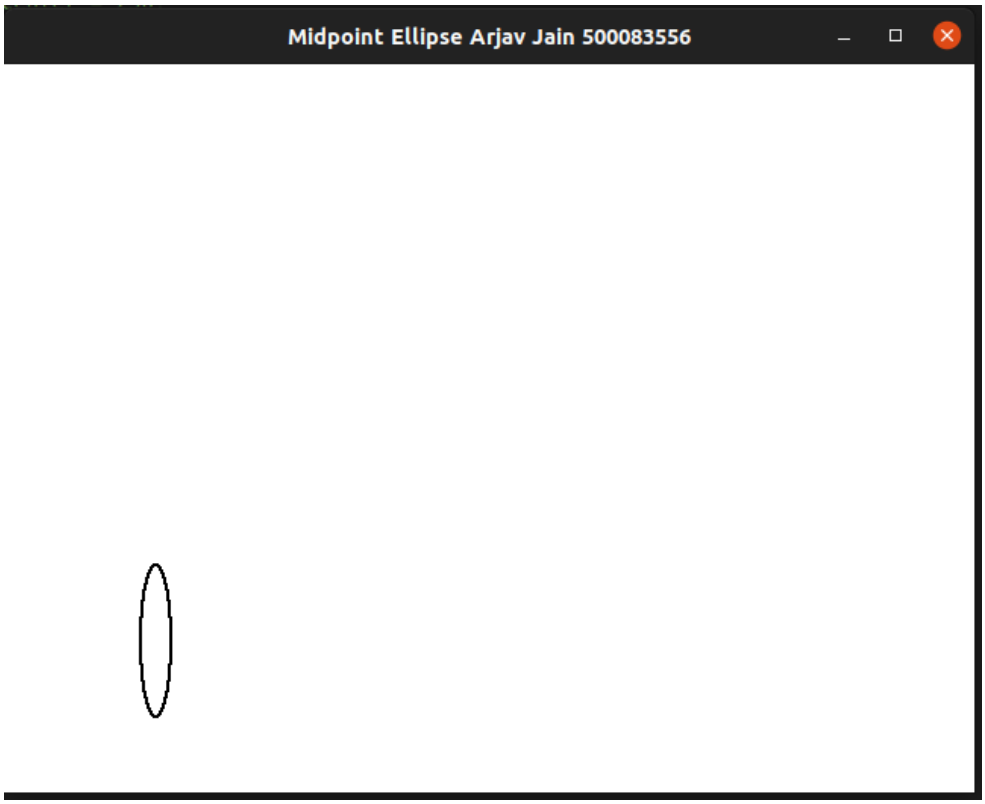
```
#include <GL/glut.h>
#include <stdio.h>

int rx, ry;          //semi-Major axis & semi Minor Axis
int xCenter, yCenter; //center of ellipse
void myinit(void)
{
   glClearColor(1.0, 1.0, 1.0, 1.0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
void setPixel(GLint x, GLint y)
{
   glBegin(GL_POINTS);
   glVertex2i(x, y);
   glEnd();
}

void ellipseMidPoint()
{
   //int xCenter = 250;
   //int yCenter = 300;
   //int rx = 200;
   //int ry = 200;

   //plotting for 1st region of 1st quardant and the slope will be < -1
   //----------------------Region-1-----------------------//
   float x = 0;
   float y = ry; //(0,ry) ---
   float p1 = ry * ry - (rx * rx) * ry + (rx * rx) * (0.25);
   //slope
   float dx = 2 * (ry * ry) * x;
   float dy = 2 * (rx * rx) * y;
   while (dx < dy)
   {
```

```
    //plot (x,y)
    setPixel(xCenter + x, yCenter + y);
    setPixel(xCenter - x, yCenter + y);
    setPixel(xCenter + x, yCenter - y);
    setPixel(xCenter - x, yCenter - y);
    if (p1 < 0)
    {
        x = x + 1;
        dx = 2 * (ry * ry) * x;
        p1 = p1 + dx + (ry * ry);
    }
    else
    {
        x = x + 1;
        y = y - 1;
        dx = 2 * (ry * ry) * x;
        dy = 2 * (rx * rx) * y;
        p1 = p1 + dx - dy + (ry * ry);
    }
}
//ploting for 2nd region of 1st quardant and the slope will be > -1
//----------------------Region-2-----------------------//
float p2 = (ry * ry) * (x + 0.5) * (x + 0.5) + (rx * rx) * (y - 1) * (y - 1) - (rx * rx) * (ry * ry);

while (y > 0)
{
    //plot (x,y)
    setPixel(xCenter + x, yCenter + y);
    setPixel(xCenter - x, yCenter + y);
    setPixel(xCenter + x, yCenter - y);
    setPixel(xCenter - x, yCenter - y); //glEnd();
    if (p2 > 0)
    {
        x = x;
        y = y - 1;
        dy = 2 * (rx * rx) * y;
        //dy = 2 * rx * rx *y;
        p2 = p2 - dy + (rx * rx);
    }
    else
```

```
        {
            x = x + 1;
            y = y - 1;
            dy = dy - 2 * (rx * rx);
            dx = dx + 2 * (ry * ry);
            p2 = p2 + dx -
                dy + (rx * rx);
        }
    }
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT); // clear the screen
    glColor3f(0.0, 0.0, 0.0);
    glPointSize(2.0);           // size of points to be drawin (in pixel)

    //establish a coordinate system for the image

    ellipseMidPoint();
    glFlush(); // send all output to the display
}

int main(int argc, char **argv)
{
    printf("Enter Center Of Ellipse\n");
    scanf("%d %d", &xCenter, &yCenter);
    // cin >> xCenter;

    // cout << "\n y = ";
    // cin >> yCenter;

    printf("Enter a Semi Major Axix\n");
    scanf("%d", &rx);
    printf("Enter a Semi Minor Axis\n");
    scanf("%d", &ry);
    glutInit(&argc, argv);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(10, 10);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow("Midpoint Ellipse Arjav Jain 500083556");
```

```
    myinit();
    glutDisplayFunc(display); // set the gl display callback function
    glutMainLoop();           // enter the GL event loop
}
```

Output:



Experiment 4
Filling –Area

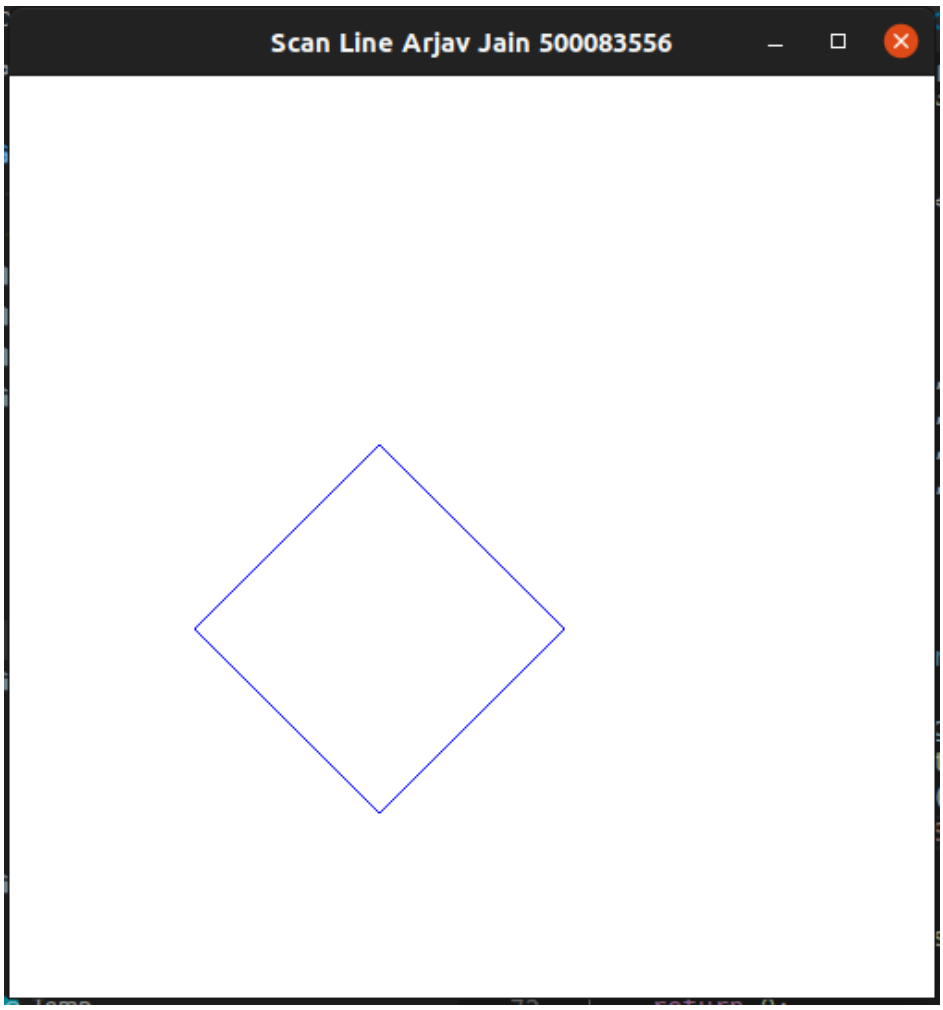
Q) WAP to fill the polygon using scan lines.

Code:

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

int le[500], re[500], m;

void init()
{
   gluOrtho2D(0, 500, 0, 500);
}

void edge(int x0, int y0, int x1, int y1)
{
   if (y1 < y0)
   {
      int tmp;
      tmp = y1;
      y1 = y0;
      y0 = tmp;
      tmp = x1;
      x1 = x0;
      x0 = tmp;
   }
   int x = x0;
   m = (y1 - y0) / (x1 - x0);
   for (int i = y0; i < y1; i++)
   {
      if (x < le[i])
         le[i] = x;
      if (x > re[i])
         re[i] = x;
      x += (1 / m);
   }
```

```
}

void display()
{
   glClearColor(1, 1, 1, 1);
   glClear(GL_COLOR_BUFFER_BIT);

   glColor3f(0, 0, 1);
   glBegin(GL_LINE_LOOP);
   glVertex2f(200, 100);
   glVertex2f(100, 200);
   glVertex2f(200, 300);
   glVertex2f(300, 200);
   glEnd();

   for (int i = 0; i < 500; i++)
   {
      le[i] = 500;
      re[i] = 0;
   }
   edge(200, 100, 100, 200);
   edge(100, 200, 200, 300);
   edge(200, 300, 300, 200);
   edge(300, 200, 200, 100);
   glFlush();
}


int main(int argc, char **argv)
{
   glutInit(&argc, argv);
   glutInitWindowPosition(100, 100);
   glutInitWindowSize(500, 500);
   glutCreateWindow("Scan Line Arjav Jain 500083556");
   init();

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

Output:



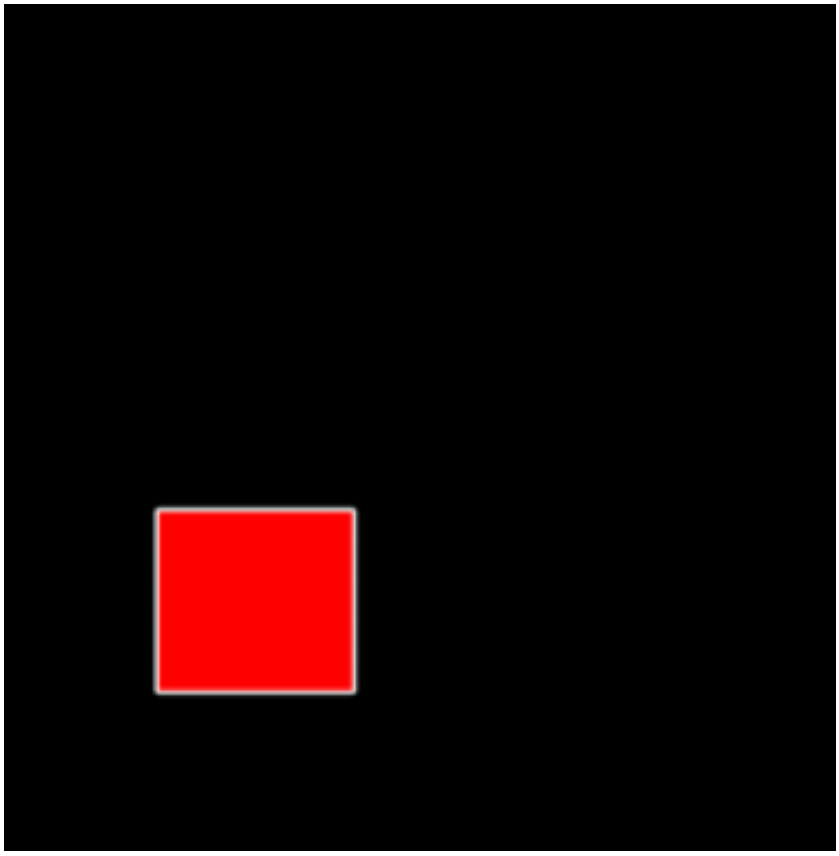
Q) WAP to fill a region using boundary fill algorithm using 4 or 8 connected approaches.

Code:

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

int x, y;

void init()
{
   gluOrtho2D(0, 500, 0, 500);
}

void boundaryFill(int x, int y, float ll[3], float boundary[3])
{
   float color[3];
   glReadPixels(x, y, 1.0, 1.0, GL_RGB, GL_FLOAT, color);
   if ((color[0] != boundary[0] || color[1] != boundary[1] || color[2] != boundary[2]) && (color[0]
!= ll[0] ||
                                                                color[1] != ll[1] || color[2] != ll[2]))
   {
      glColor3f(ll[0], ll[1], ll[2]);
      glBegin(GL_POINTS);
      glVertex2i(x, y);
      glEnd();
      glFlush();
      boundaryFill(x + 1, y, ll, boundary);
      boundaryFill(x - 1, y, ll, boundary);
      boundaryFill(x, y + 1, ll, boundary);
      boundaryFill(x, y - 1, ll, boundary);
      boundaryFill(x - 1, y - 1, ll, boundary);
      boundaryFill(x + 1, y - 1, ll, boundary);
      boundaryFill(x + 1, y + 1, ll, boundary);
      boundaryFill(x - 1, y + 1, ll, boundary);
   }
}

void display()
{
```

```
    int n, a, b, i;
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter the vertices: ");
    glLineWidth(2);
    glBegin(GL_LINE_LOOP);
    glColor3f(1.0f, 1.0f, 1.0f);
    for (i = 0; i < n; i++)
    {
        scanf("%d %d", &a, &b);
        glVertex2f(a, b);
    }
    glEnd();
    glFlush();

    printf("Enter seed vertex: ");
    scanf("%d %d", &x, &y);
    float boundary[3] = {1.0f, 1.0f, 1.0f};
    float color[3] = {1.0f, 0.0f, 0.0f};
    boundaryFill(x, y, color, boundary);
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(200, 200);
    glutCreateWindow("Boundary Fill Arjav Jain 500083556");
    glutDisplayFunc(display);
    init();
    glutMainLoop();
    return 0;
}
```

Output:



Q) WAP to fill a region using flood fill algorithm using 4 or 8 connected approaches.

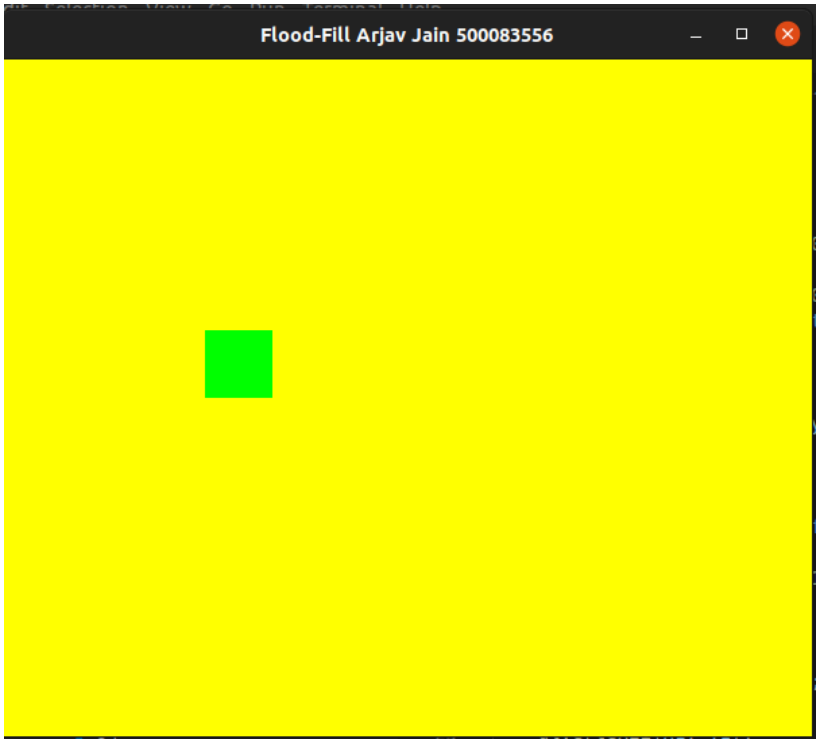
Code:

```
#include <stdio.h>
#include <math.h>
#include <GL/glut.h>
int ww = 600, wh = 500;
float bgCol[3] = {1.0, 0.9, 0.7};
float intCol[3] = {1.0, 0.0, 0.0};
float llCol[3] = {0.4, 0.3, 0.0};
void setPixel(int pointx, int pointy, float f[3])
{
   glBegin(GL_POINTS);
   glColor3fv(f);
   glVertex2i(pointx, pointy);
   glEnd();
   glFlush();
}
void getPixel(int x, int y, float pixels[3])
{
   glReadPixels(x, y, 1.0, 1.0, GL_RGB, GL_FLOAT, pixels);
}
void drawPolygon(int x1, int y1, int x2, int y2)
{
   glColor3f(0.0, 1.0, 0.0);
   glBegin(GL_POLYGON);
   glVertex2i(x1, y1);
   glVertex2i(x1, y2);
   glVertex2i(x2, y2);
   glVertex2i(x2, y1);
   glEnd();
   glFlush();
}
void display()
{
   glClearColor(1.2, 1.4, 0.0, 0.6);
   glClear(GL_COLOR_BUFFER_BIT);
   drawPolygon(150, 250, 200, 300);
   glFlush();
```

```
}
void floodfill4(int x, int y, float oldcolor[3], float newcolor[3])
{
   float color[3];
   getPixel(x, y, color);
   if (color[0] == oldcolor[0] && (color[1]) == oldcolor[1] && (color[2]) == oldcolor[2])
   {
      setPixel(x, y, newcolor);
      floodfill4(x + 1, y, oldcolor, newcolor);
      floodfill4(x - 1, y, oldcolor, newcolor);
      floodfill4(x, y + 1, oldcolor, newcolor);
      floodfill4(x, y - 1, oldcolor, newcolor);
   }
}
void mouse(int btn, int state, int x, int y)
{
   if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
   {
      int xi = x;
      int yi = (wh - y);
      floodfill4(xi, yi, intCol, llCol);
   }
}
void myinit()
{
   glViewport(0, 0, ww, wh);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(0.0, (GLdouble)ww, 0.0, (GLdouble)wh);
   glMatrixMode(GL_MODELVIEW);
}
int main(int argc, char **argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(ww, wh);
   glutCreateWindow("Flood-Fill Arjav Jain 500083556");
   glutDisplayFunc(display);
   myinit();
   glutMouseFunc(mouse);
```

```
    glutMainLoop();
    return 0;
}
```

Output:



Experiment 7
Basic Two Dimensional Transformations

Q) Write an interactive program for following basic transformation.

- **Translation**
- **Rotation**
- **Scaling**
- **Reflection about axis.**
- **Reflection about a line Y=mX+c and aX+bY+c=0.**
- **Shear about an edge and about a vertex.**

Code:

```
#include <GL/glut.h>
#include <stdio.h>
#include <math.h>
#define pi 3.1415
float x[100], y[100];
float p[100], m[100];
int n, k, c;
double angle;
float a, b;
void init()
{
   glClear(GL_COLOR_BUFFER_BIT);
   glClearColor(1.0, 1.0, 1.0, 1.0);
   glLoadIdentity();
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D(-200, 200, -200, 200);
}
void translation()
{
   printf("\nEnter Translation Distance along X and Y axix:");
   scanf("%f%f", &a, &b);
   for (int i = 0; i < n; i++)
   {
      p[i] += a;
      m[i] += b;
   }
}
```

```
void rotation()
{
   printf("\nEnter The Angle Of Rotation :");
   scanf("%lf", &angle);
   printf("\n1.Anticlockwise\n2.Clockwise");
   printf("\nEnter Your Choice :");
   scanf("%d", &c);
   printf("\nEnter the point about which Rotation is to be done:");
   scanf("%f%f", &a, &b);
   if (c == 1)
      angle = (angle * pi) / 180;
   if (c == 2)
      angle = (-angle * pi) / 180;
   for (int i = 0; i < n; i++)
   {
      p[i] -= a;
      m[i] -= b;
   }
   for (int i = 0; i < n; i++)
   {
      float t1, t2;
      t1 = p[i];
      t2 = m[i];
      p[i] = (t1 * cos(angle)) - (t2 * sin(angle));
      m[i] = (t1 * sin(angle)) + (t2 * cos(angle));
   }
   for (int i = 0; i < n; i++)
   {
      p[i] += a;
      m[i] += b;
   }
}
void scaling()
{
   printf("\nEnter Scaling Factor along X and Y:");
   scanf("%f%f", &a, &b);
   for (int i = 0; i < n; i++)
   {
      p[i] *= a;
      m[i] *= b;
```

```
    }
}
void reflection()
{
    printf("Reflection about:\n1. X-axis\n2. Y-axis\n3. X=Y\n4. X=-Y\n5. Origin\n6. An arbitrary
line");
    printf("\nEnter Your Choice :");
    scanf("%d", &c);
    if (c == 1)
    {
        for (int i = 0; i < n; i++)
        {
            m[i] = -m[i];
        }
    }
    else if (c == 2)
    {
        for (int i = 0; i < n; i++)
        {
            p[i] = -p[i];
        }
    }
    else if (c == 3)
    {
        for (int i = 0; i < n; i++)
        {
            float temp;
            temp = p[i];
            p[i] = m[i];
            m[i] = temp;
        }
    }
    else if (c == 4)
    {
        for (int i = 0; i < n; i++)
        {
            float temp;
            temp = -p[i];
            p[i] = -m[i];
            m[i] = temp;
```

```
    }
}
else if (c == 5)
{
    for (int i = 0; i < n; i++)
    {
        p[i] = -p[i];
        m[i] = -m[i];
    }
}
else if (c == 6)
{
    float m1;
    printf("\nEnter Slope And Intercept Of The Line : ");
    scanf("%f%d", &m1, &c);
    for (int i = 0; i < n; i++)
    {
        m[i] -= c;
    }
    angle = -atan(m1);
    for (int i = 0; i < n; i++)
    {
        float t1, t2;
        t1 = p[i];
        t2 = m[i];
        p[i] = (t1 * cos(angle)) - (t2 * sin(angle));
        m[i] = (t1 * sin(angle)) + (t2 * cos(angle));
    }
    for (int i = 0; i < n; i++)
    {
        m[i] = -m[i];
    }
    angle = -angle;
    for (int i = 0; i < n; i++)
    {
        float t1, t2;
        t1 = p[i];
        t2 = m[i];
        p[i] = (t1 * cos(angle)) - (t2 * sin(angle));
        m[i] = (t1 * sin(angle)) + (t2 * cos(angle));
```

```
        }
        for (int i = 0; i < n; i++)
        {
            m[i] += c;
        }
    }
    else
        printf("Wrong Choice\n");
}
void shearing()
{
    printf("\nEnter X and Y Shearing Factor :");
    scanf("%f%f", &a, &b);
    for (int i = 0; i < n; i++)
    {
        p[i] += a * m[i];
        m[i] += b * p[i];
    }
}
void display(void)
{
    printf("Enter the Number of Vertices :");
    scanf("%d", &n);
    printf("\nEnter the Vertices : ");
    for (int i = 0; i < n; i++)
    {
        scanf("%f%f", &x[i], &y[i]);
        p[i] = x[i];
        m[i] = y[i];
    }
    glBegin(GL_POLYGON);
    glColor3f(0.8f, 0.1f, 0.1f);
    for (int i = 0; i < n; i++)
    {
        glVertex2f(x[i], y[i]);
    }
    glEnd();
    glFlush();
    while (1)
    {
```

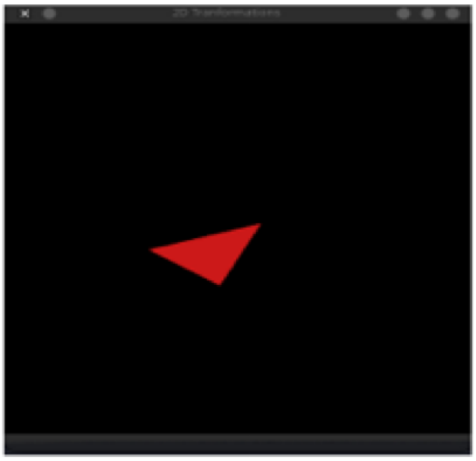
```
        printf("1.Translate\n2.Rotate\n3.Scale\n4.Reflect\n5.Shear\n0.Exit");
        printf("\nEnter Your Choice : ");
        scanf("%d", &k);
        if (k == 1)
            translation();
        else if (k == 2)
            rotation();
        else if (k == 3)
            scaling();
        else if (k == 4)
            reflection();
        else if (k == 5)
            shearing();
        else if (k == 0)
            break;
        else
            printf("Wrong Choice\n");
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_POLYGON);
        glColor3f(0.8f, 0.1f, 0.1f);
        for (int i = 0; i < n; i++)
        {
            glVertex2f(p[i], m[i]);
        }
        glEnd();
        glFlush();
    }
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(200, 200);
    glutCreateWindow("2D Tranformations");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
```

}

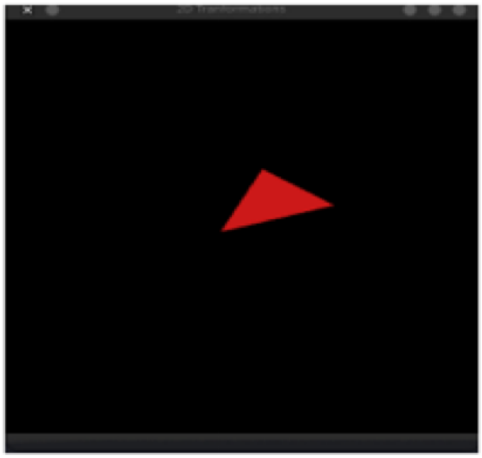
Output:

Reflect along x-axis

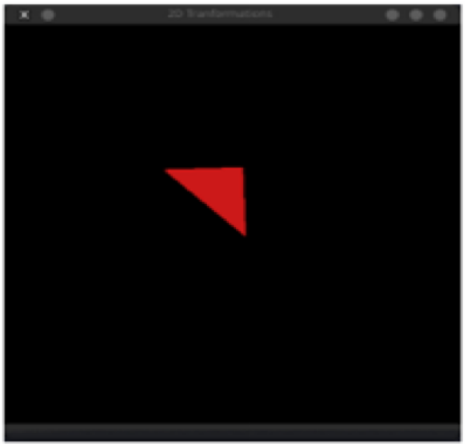
Reflect along y-axis



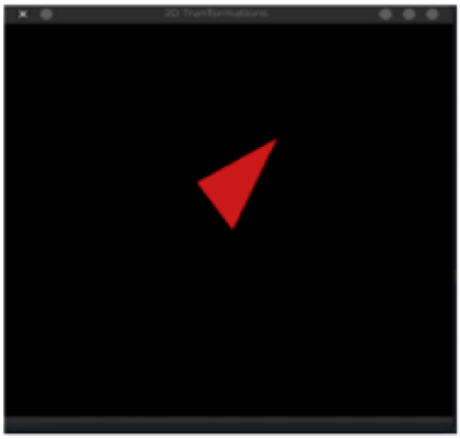
Reflect along x=y



Reflect along origin



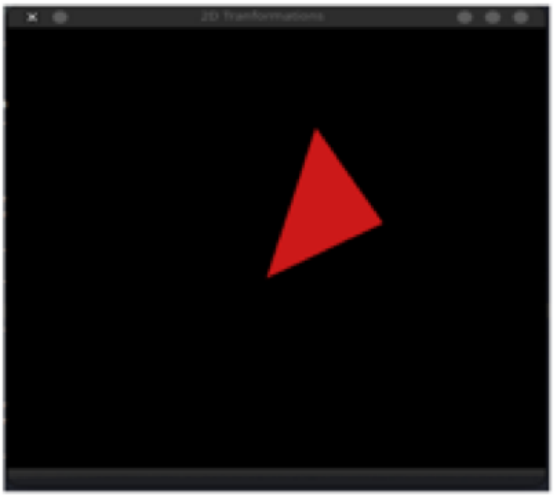
Reflect Along y=4x+5



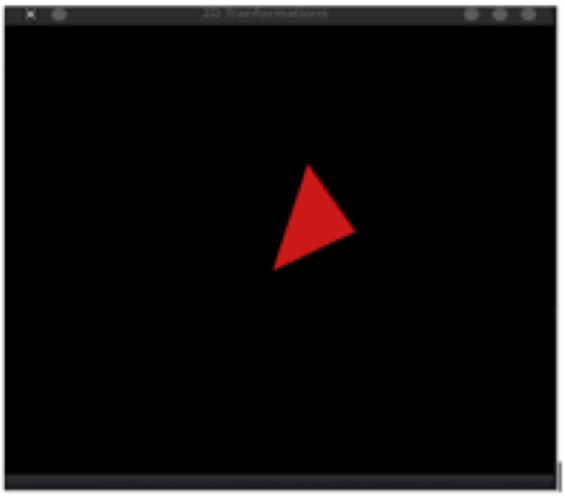
Shear 0.6 0.6 about x and y

Original Figure

Translate by 30,30



30°anticlock rotation along origin



Scale by 0.7 0.7 along x and y

Experiment 8
Basic Three Dimensional Transformations

Q) Write an interactive program for following basic transformation.

- **Translation**
- **Rotation**
- **Scaling**
- **Reflection about axis.**
- **Reflection about a line Y=mX+c and aX+bY+c=0.**
- **Shear about an edge and about a vertex.**

Code:

```
#include <GL/glut.h>
#include <stdio.h>
#include <math.h>
#define pi 3.1415
float A[8][3] = {{0, 0, 0}, {25, 0, 0}, {0, 25, 0}, {0, 0, 25}, {25, 25, 0}, {25, 0, 25}, {0, 25, 25},
{25, 25, 25}};
float B[8][3] = {{0, 0, 0}, {25, 0, 0}, {0, 25, 0}, {0, 0, 25}, {25, 25, 0}, {25, 0, 25}, {0, 25, 25},
{25, 25, 25}};
float x, y, z;
double angle;
int k, c, p;
void init()
{
   glEnable(GL_DEPTH_TEST);
   glOrtho(-100.0, 100.0, -100.0, 100.0, -100.0, 100.0);
}
void translation()
{
   printf("\nEnter Translation Distance along X, Y and Z axes:");
   scanf("%f%f%f", &x, &y, &z);
   for (int i = 0; i < 8; i++)
   {
      B[i][0] = B[i][0] + x;
      B[i][1] = B[i][1] + y;
      B[i][2] = B[i][2] + z;
   }
```

```
}
void rotation()
{
   printf("\nEnter The Angle Of Rotation :");
   scanf("%lf", &angle);
   printf("\n1.Anticlockwise\n2.Clockwise");
   printf("\nEnter Your Choice :");
   scanf("%d", &p);
   if (p == 1)
      angle = (angle * pi) / 180;
   if (p == 2)
      angle = (-angle * pi) / 180;
   printf("\nRotation about:\n1.X-axis\n2.Y-axis\n3.Z-axis");
   printf("\nEnter Your Choice :");
   scanf("%d", &c);
   if (c == 1)
   {
      for (int i = 0; i < 8; i++)
      {
         float t;
         t = B[i][1];
         B[i][1] = (B[i][1] * cos(angle)) - (B[i][2] * sin(angle));
         B[i][2] = (t * sin(angle)) + (B[i][2] * cos(angle));
      }
   }
   else if (c == 2)
   {
      for (int i = 0; i < 8; i++)
      {
         float t;
         t = B[i][0];
         B[i][0] = (B[i][0] * cos(angle)) - (B[i][2] * sin(angle));
         B[i][2] = (t * sin(angle)) + (B[i][2] * cos(angle));
      }
   }
   else if (c == 3)
   {
      for (int i = 0; i < 8; i++)
      {
         float t;
```

```
            t = B[i][0];
            B[i][0] = (B[i][0] * cos(angle)) - (B[i][1] * sin(angle));
            B[i][1] = (t * sin(angle)) + (B[i][1] * cos(angle));
        }
    }
    else
        printf("Wrong choice");
}
void scaling()
{
    printf("\nEnter Scaling Factor along X, Y and Z:");
    scanf("%f%f%f", &x, &y, &z);
    for (int i = 0; i < 8; i++)
    {

        B[i][0] = B[i][0] * x;
        B[i][1] = B[i][1] * y;
        B[i][2] = B[i][2] * z;
    }
}
void reflection()
{
    printf("Reflection about:\n1. XY plane\n2. YZ plane\n3. ZX plane");
    printf("\nEnter Your Choice :");
    scanf("%d", &c);
    if (c == 1)
        for (int i = 0; i < 8; i++)
            B[i][2] = -B[i][2];
    else if (c == 2)
        for (int i = 0; i < 8; i++)
            B[i][0] = -B[i][0];
    else if (c == 3)
        for (int i = 0; i < 8; i++)
            B[i][1] = -B[i][1];
    else
        printf("Wrong Choice\n");
}
void shearing()
{
    printf("\nEnter X, Y and Z Shearing Factor :");
```

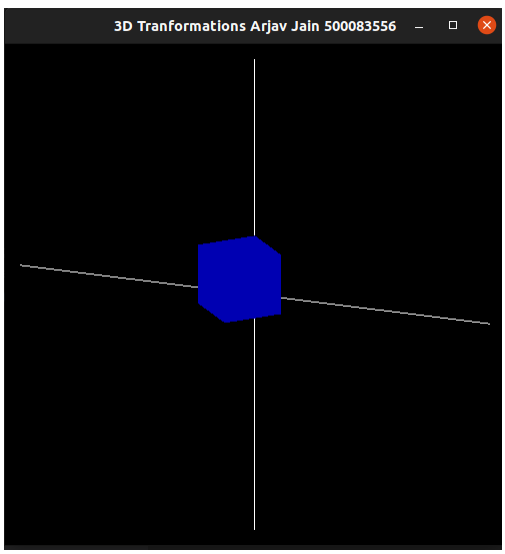
```
    scanf("%f%f%f", &x, &y, &z);
    for (int i = 0; i < 8; i++)
    {
        B[i][0] = B[i][0] + y * B[i][1] + z * B[i][2];
        B[i][1] = B[i][1] + x * B[i][0] + z * B[i][2];
        B[i][2] = B[i][2] + x * B[i][0] + y * B[i][1];
    }
}
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glBegin(GL_LINES);
    glColor3f(1.0f, 1.0f, 1.0f);
    glVertex2f(0, 100);
    glVertex2f(0, -100);
    glVertex2f(100, 0);
    glVertex2f(-100, 0);
    glEnd();
    glRotatef(20, 1.0, 0.0, 0.0);
    glRotatef(-20, 0.0, 1.0, 0.0);
    glBegin(GL_QUADS);
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3fv(A[6]);
    glVertex3fv(A[7]);
    glVertex3fv(A[5]);
    glVertex3fv(A[3]);
    glColor3f(0.7f, 0.0f, 0.0f);
    glVertex3fv(A[2]);
    glVertex3fv(A[4]);
    glVertex3fv(A[1]);
    glVertex3fv(A[0]);
    glColor3f(0.7f, 0.0f, 0.0f);
    glVertex3fv(A[6]);
    glVertex3fv(A[2]);
    glVertex3fv(A[0]);
    glVertex3fv(A[3]);
    glColor3f(0.7f, 0.0f, 0.0f);
    glVertex3fv(A[7]);
    glVertex3fv(A[4]);
    glVertex3fv(A[1]);
```

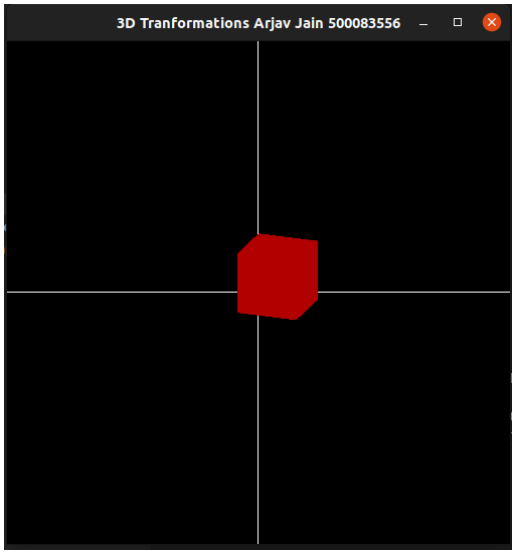
```
glVertex3fv(A[5]);
glColor3f(0.7f, 0.0f, 0.0f);
glVertex3fv(A[6]);
glVertex3fv(A[2]);
glVertex3fv(A[4]);
glVertex3fv(A[7]);
glColor3f(0.7f, 0.0f, 0.0f);
glVertex3fv(A[3]);
glVertex3fv(A[0]);
glVertex3fv(A[1]);
glVertex3fv(A[5]);
glEnd();
glFlush();
glutSwapBuffers();
while (1)
{
   printf("1.Translate\n2.Rotate\n3.Scale\n4.Reflect\n5.Shear\n0.Exit");
   printf("\nEnter Your Choice : ");
   scanf("%d", &k);
   if (k == 1)
      translation();
   else if (k == 2)
      rotation();
   else if (k == 3)
      scaling();
   else if (k == 4)
      reflection();
   else if (k == 5)
      shearing();
   else if (k == 0)
      break;
   else
      printf("Wrong Choice\n");
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glBegin(GL_LINES);
   glColor3f(1.0f, 1.0f, 1.0f);
   glVertex2f(0, 100);
   glVertex2f(0, -100);
   glVertex2f(100, 0);
   glVertex2f(-100, 0);
```

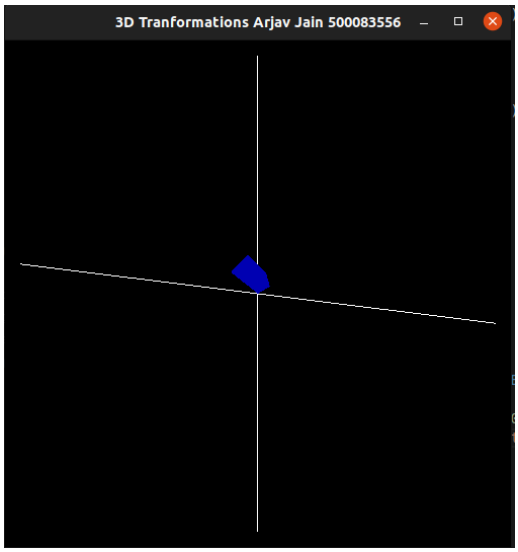
```
        glEnd();
        glBegin(GL_QUADS);
        glColor3f(0.0f, 0.0f, 1.0f);
        glVertex3fv(B[6]);
        glVertex3fv(B[7]);
        glVertex3fv(B[5]);
        glVertex3fv(B[3]);
        glColor3f(0.0f, 0.0f, 0.7f);
        glVertex3fv(B[2]);
        glVertex3fv(B[4]);
        glVertex3fv(B[1]);
        glVertex3fv(B[0]);
        glColor3f(0.0f, 0.0f, 0.7f);
        glVertex3fv(B[6]);
        glVertex3fv(B[2]);
        glVertex3fv(B[0]);
        glVertex3fv(B[3]);
        glColor3f(0.0f, 0.0f, 0.7f);
        glVertex3fv(B[7]);
        glVertex3fv(B[4]);
        glVertex3fv(B[1]);
        glVertex3fv(B[5]);
        glColor3f(0.0f, 0.0f, 0.7f);
        glVertex3fv(B[6]);
        glVertex3fv(B[2]);
        glVertex3fv(B[4]);
        glVertex3fv(B[7]);
        glColor3f(0.0f, 0.0f, 0.7f);
        glVertex3fv(B[3]);
        glVertex3fv(B[0]);
        glVertex3fv(B[1]);
        glVertex3fv(B[5]);
        glEnd();
        glFlush();
        glutSwapBuffers();
    }
}

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

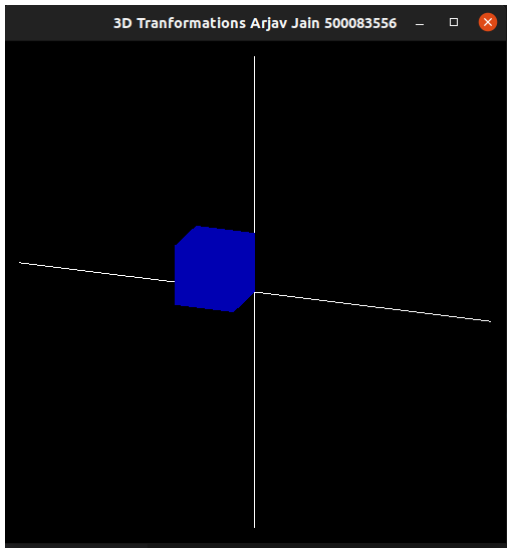
```
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(200, 200);
    glutCreateWindow("3D Tranformations Arjav Jain 500083556");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

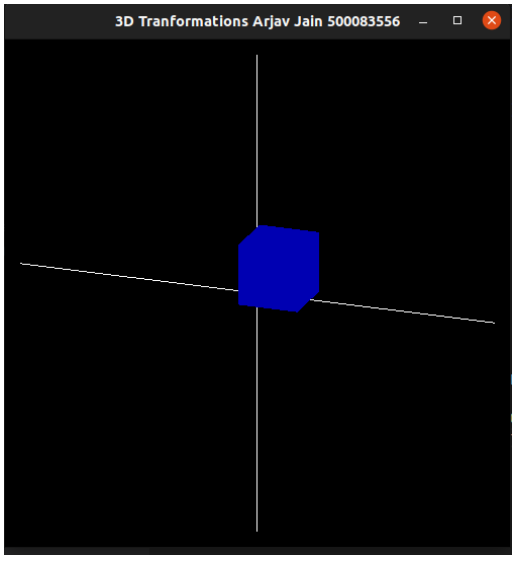
Output:

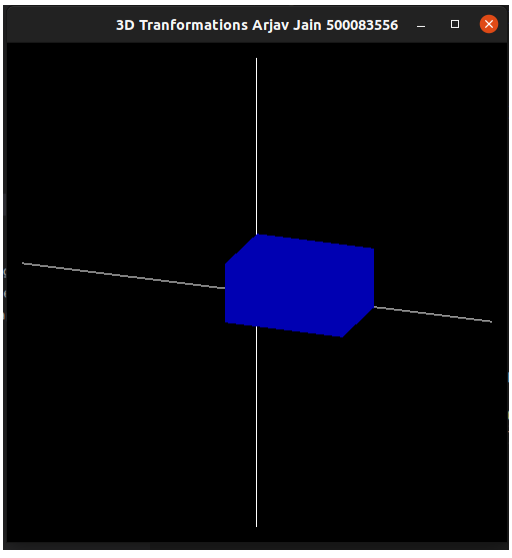

3D Tranformations Arjav Jain 500083556


3D Tranformations Arjav Jain 500083556


3D Tranformations Arjav Jain 500083556


3D Tranformations Arjav Jain 500083556


3D Tranformations Arjav Jain 500083556


3D Tranformations Arjav Jain 500083556