COMPUTER GRAPHICS lab

Program of study – B. Tech CSE (H)DevOps (2021-25)

IVth Semester



Submitted to SUBMITTED BY

Dr. Rohit Shrivastav Harshit Pandey

School Of Computer Science 500095944

UPES Batch – DevOps HB1

University Of Petroleum And

Energy Studies, Dehradun

**Index**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sr No. | Experiment | Page No. | Date Of Experiment | Remarks |
|  | Introduction to OpenGL – Drawing Basic Triangles with Mouse Function | 3 - 7 |  |  |
|  | Drawing Line using DDA, Bresenham Algorithm | 8 - 11 |  |  |
|  | Drawing Circle using Bresenham, Mid Point Algorithm | 12 - 15 |  |  |
|  | Drawing Ellipse using Mid Point theorem | 15 - 17 |  |  |
|  | Filling the objects flood fill and boundary fill algorithm | 18 - 22 |  |  |
|  | Write a program for 2D & 3D transformation: translation, scaling, rotation. | 22 – 42 |  |  |
|  | Line Clipping Algorithm | 43 - 47 |  |  |

**Experiment 1**

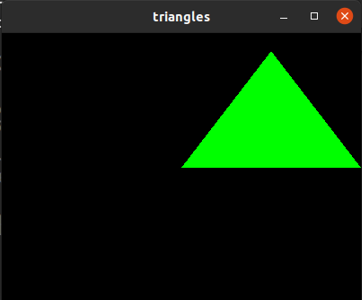
**Introduction to OpenGL**

1. **Simple OpenGL program to draw a triangle**

**Code:**

|  |
| --- |
| #include <GL/glut.h>  #include<math.h>  void disp(void)  {  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  glColor3f(0.0, 1.0, 0.0);  glLoadIdentity();  glBegin(GL\_POLYGON);  float p=sqrt(0.75);  glVertex3f(0.0, 0.0, 0.0);  glVertex3f(1.0, 0.0, 0.0);  glVertex3f(0.5, p, 0.0);  glEnd();  glutSwapBuffers();  }  int main(int argc, char\*\* argv)  {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE);  glutInitWindowSize(400, 300);  glutInitWindowPosition(100, 100);  glutCreateWindow("triangles");  glutDisplayFunc(disp);  glutMainLoop();  return 0;  } |

**Output:**

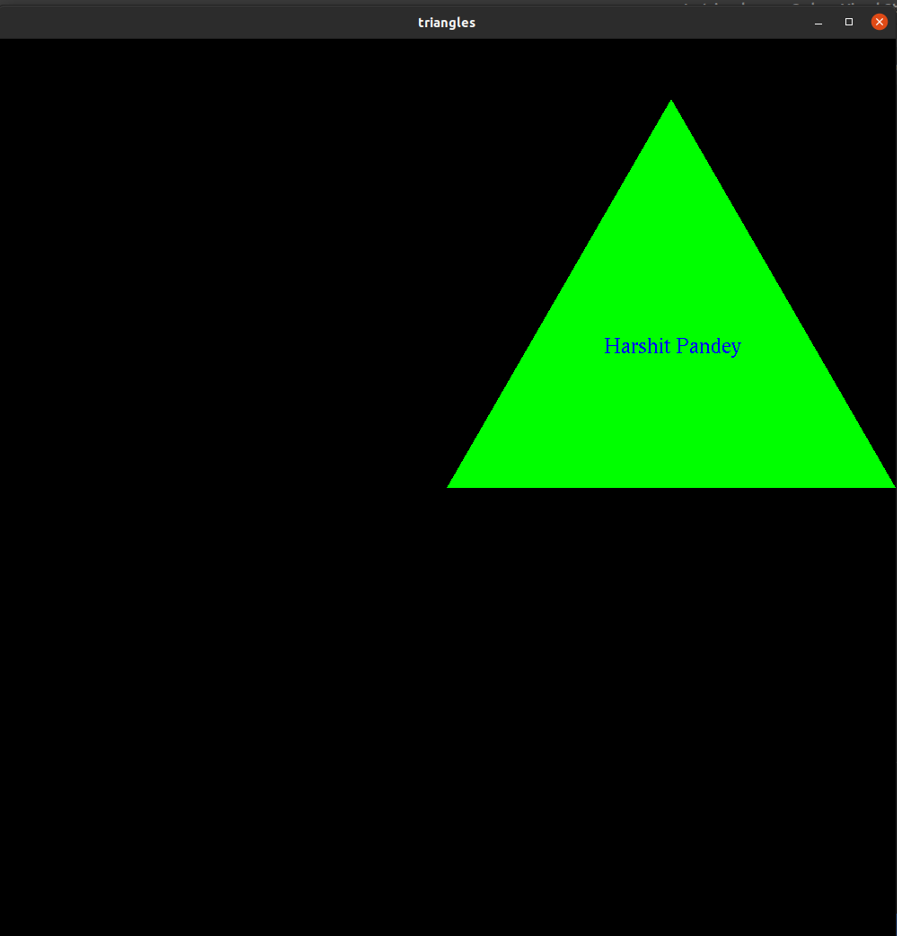


1. **Program to write name on the triangle**

**Code:**

|  |
| --- |
| #include<GL/glut.h>  #include<math.h>  void draw\_triangle()  {  glBegin(GL\_POLYGON);  //glColor3d(255.0, 255.0, 255.0);  glColor3ub(1,1,1);  float p=sqrt(0.75);  glVertex3f(0.0, 0.0, 0.0);  glVertex3f(1.0, 0.0, 0.0);  glVertex3f(0.5, p, 0.0);  glEnd();  }  void drawline(float x1,float y1,float x2,float y2)  {  glVertex2f(x1,y1);  glVertex2f(x2,y2);  }  void draw\_name()  {  glColor3f(1.0,1.0,1.0);  // glLineWidth(4.1);  // drawline(0.0,0.0,0.0,1.0);  // drawline(0.1,0.0,0.1,0.1);  // drawline(0.1,0.05,0.15,0.05);  // drawline(0.15,0.0,0.15,0.1);  // drawline(0.17,0.0,0.20,0.1);  // drawline(0.18,0.05,0.22,0.05);  // drawline(0.20,0.1,0.23,0.0);  // drawline(0.25,0.0,0.25,.1);  // drawline(0.25,0.1,0.28,0.1);  // drawline(0.28,0.1,0.28,0.05);  // drawline(0.28,0.05,0.25,0.05);  // drawline(0.25,0.05,0.28,0.0);  // not feasable...  glRasterPos2f(0.35,0.3);  char name[]="Harshit Pandey";  for(int i=0;name[i]!='\0';i++)  glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24,name[i]);  }  void disp()  {  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  draw\_triangle();  draw\_name();  glutSwapBuffers();  }  int main(int argc, char\*\* argv)  {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);  glutInitWindowSize(1000, 1000);  glutInitWindowPosition(100, 100);  glutCreateWindow("triangles");  glutDisplayFunc(disp);  //disp();  glutMainLoop();  return 0;  } |

**Output:**

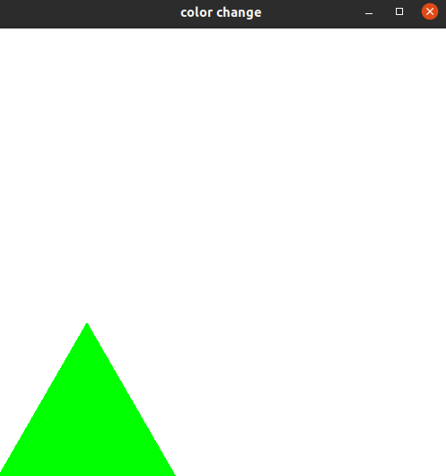
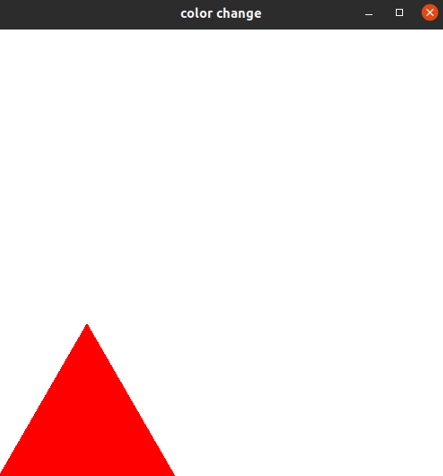


1. **Program to change color of the triangle using glMouseFunc**

**Code:**

|  |
| --- |
| #include<GL/glut.h>  #include<math.h>  #include<stdio.h>  float p=1,q=0;  float p1;  int x=0;  void myInit(void)  {  glClearColor(1.0,1.0,1.0,1.0);  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3d(0,0,0);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluOrtho2D(0,500,0,500);  }  void drawTriangle()  {  x=1;  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  glBegin(GL\_TRIANGLES);  glColor3f(p, q, 0.0);  p1=sqrt(200\*200 - 100\*100);  glVertex3d(0,0,0 );  glVertex3d(200,0 ,0 );  glVertex3d(100, p1,0);  glEnd();  glFlush();  }  void disp(void)  {  glClear(GL\_COLOR\_BUFFER\_BIT);  drawTriangle();  glFlush();  }  void onClick(int button,int state,int x,int y)  {  float m1=p1/100;  float m2=0;  float m3=p1/(-100);  float mc=(500-float(y))/float(x);  float mc2=(500-float(y))/(float(x)-200);    if(mc<m1 && mc>m2 && mc2>m3)  {  if(m3<0 && mc2<0)  if(button==GLUT\_LEFT\_BUTTON)  {  //printf("%d and %d -> %f -> %f -> %f -> %f -> %f -> %f\n",x,y,mc,m1,mc,m2,mc2,m3);  float temp=p;  p=q;  q=temp;  drawTriangle();  }  }  }  int main(int argc,char\*\* argv)  {  glutInit(&argc,argv);  glutInitDisplayMode(GLUT\_SINGLE);  glutInitWindowSize(500,500);  glutInitWindowPosition(400,400);  glutCreateWindow("color change");  glutDisplayFunc(disp);  glutMouseFunc(onClick);  myInit();  glutMainLoop();  return 0;  } |

**Output:**



**Experiment 2**

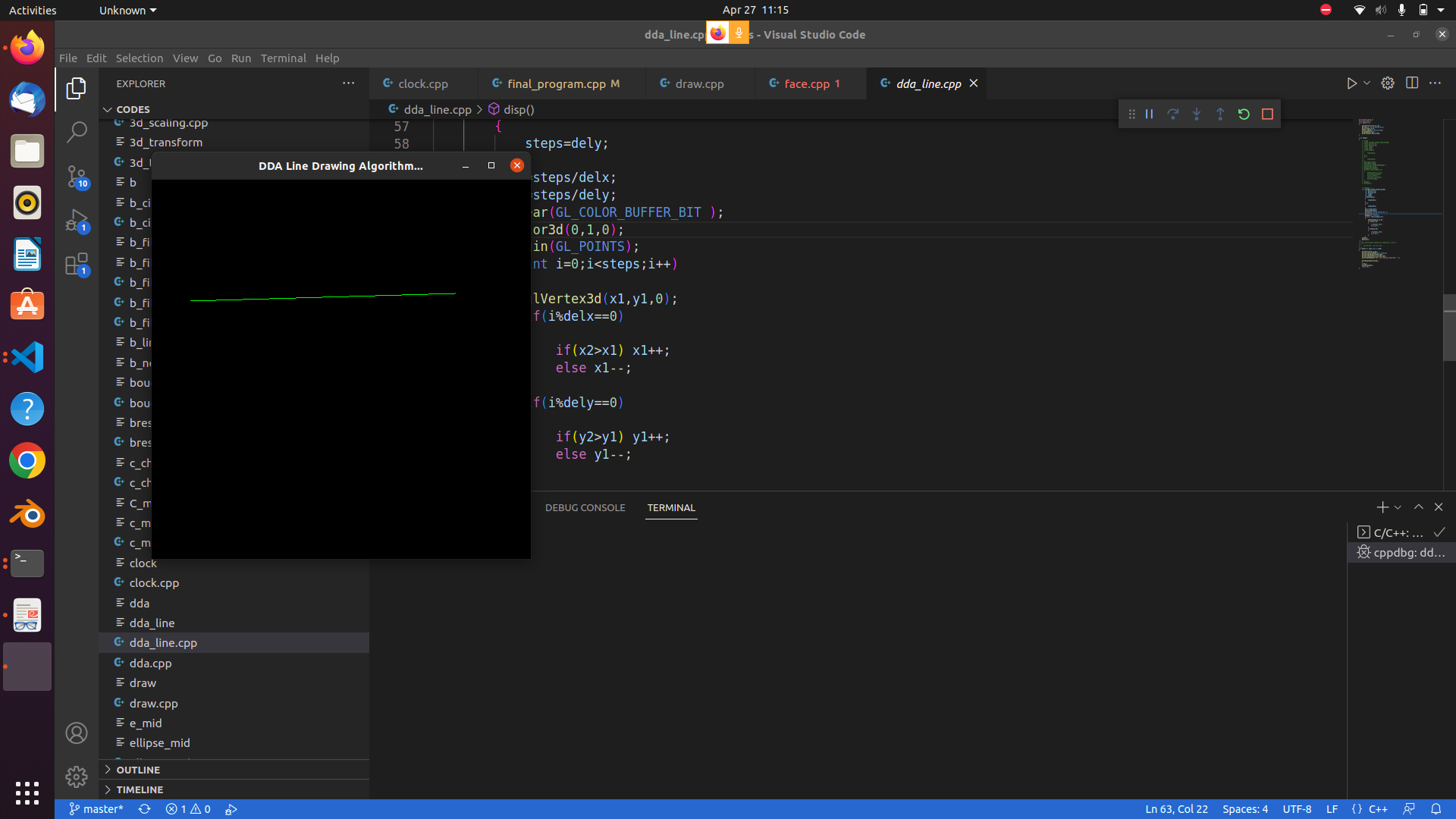
**Drawing Line**

1. **Drawing line using DDA algorithm.**

**Code:**

|  |
| --- |
| =#include<GL/glut.h>  #include<stdio.h>  void init(void)  {  glClearColor(1.0,1.0,1.0,1.0);  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3d(0,0,0);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluOrtho2D(0,500,0,500);  }  void disp()  {  // wrong  // float x1=100,y1=350,x2=50,y2=100;  // float delx=x1-x2;  // float dely=y1-y2;  // float steps;  // float temp;  // if(delx>dely)  // {  // steps=delx;  // }  // else  // {  // steps=dely;  // }  // delx=delx/steps;  // dely=dely/steps;  // glClear(GL\_COLOR\_BUFFER\_BIT );  // glColor3d(1,0,0);  // glBegin(GL\_POINTS);  // for(int i=0;i<steps;i++)  // {  // glVertex3f(x1,y1,0);  // if(x2>x1)x1+=delx;  // else x1-=delx;  // if(y2>y1) y1+=dely;  // else y1-=dely;  // }  // glEnd();  // glFlush();  // method 2 ...  int x1=400,y1=350,x2=50,y2=340;  int delx=x1-x2;  int dely=y1-y2;  int steps;  int temp;  if(delx>dely)  {  steps=delx;  }  else  {  steps=dely;  }  delx=steps/delx;  dely=steps/dely;  glClear(GL\_COLOR\_BUFFER\_BIT );  glColor3d(1,0,0);  glBegin(GL\_POINTS);  for(int i=0;i<steps;i++)  {  glVertex3d(x1,y1,0);  if(i%delx==0)  {  if(x2>x1) x1++;  else x1--;  }  if(i%dely==0)  {  if(y2>y1) y1++;  else y1--;  }  }  glEnd();  glFlush();  }  // void onClick(int button,int state,int x,int y)  // {  // printf("%d , %d \n",x,y);  // }  int main(int argc,char\*\* argv)  {  glutInit(&argc,argv);  glutInitDisplayMode(GLUT\_SINGLE);  glutInitWindowSize(500,500);  glutInitWindowPosition(200,200);  glutCreateWindow("DDA Line Drawing Algorithm...");    glutDisplayFunc(disp);  // `````````````````` glutMouseFunc(onClick);  init();  glutMainLoop();  return 0;  } |

**Output:**

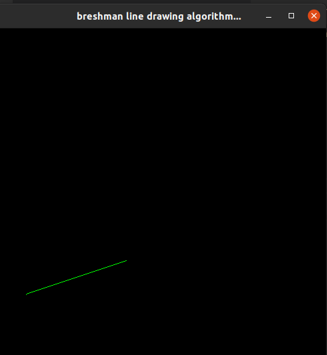
****

1. **Drawing line using Bresenham algorithm.**

**Code:**

|  |
| --- |
| #include<GL/glut.h>  #include<iostream>  void init(void)  {  glClearColor(1,1,1,1);  glClear(GL\_COLOR\_BUFFER\_BIT);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluOrtho2D(0,500,0,500);  }  void disp(void)  {  int x1=50,y1=100,x2=200,y2=150;  int dx=x2-x1,dy=y2-y1;  int p=2\*dx-dy;  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT );  glColor3d(1,0,0);  // glBegin(GL\_POINTS);  // glVertex2d(50,100);  // glEnd();  // if(steps==dy)  // {  // glRasterPos2d(50,100);  // char text[]="IN MAINTANENCE";  // std::cout<<"HEllo ";  // int i=0;  // while(text[i]!='\0')  // {  // glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24,text[i]);  // i++;  // }  // return;  // }  glBegin(GL\_POINTS);  while(x1<=x2)  {  glVertex2d(x1,y1);  x1++;  if(p<0)  {  p+=2\*abs(dy);  }  else  {  p+=2\*abs(dy)-2\*abs(dx);  y1++;  }  }  glEnd();  glFlush();  }  int main(int argc,char\*\* argv)  {  glutInit(&argc,argv);  glutInitDisplayMode(GLUT\_SINGLE);  glutInitWindowSize(500,500);  glutInitWindowPosition(100,100);  glutCreateWindow("breshman line drawing algorithm...");  glutDisplayFunc(disp);  init();  glutMainLoop();  return 0;  } |

**Output:**



**Experiment 3**

**Drawing circle**

1. **Drawing circle using bresenham algorithm.**

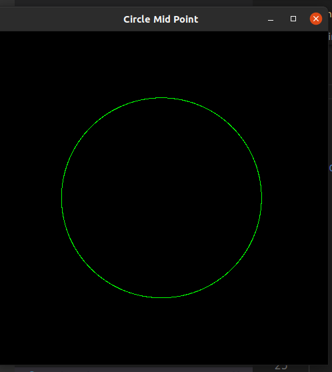
|  |
| --- |
| #include<GL/glut.h>  void init()  {  glClearColor(1,1,1,1);  glClear(GL\_COLOR\_BUFFER\_BIT);  glLoadIdentity();  glMatrixMode(GL\_PROJECTION);  gluOrtho2D(-500,500,-500,500);  }  void disp()  {  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3d(1,0,0);  int r=300;  int y1=r,x1=0;  int p=1-r;  glBegin(GL\_POINTS);  while(x1<y1)  {  glVertex2d(x1,y1);  glVertex2d(y1,x1);  glVertex2d(-x1,y1);  glVertex2d(-y1,x1);  glVertex2d(-x1,-y1);  glVertex2d(-y1,-x1);  glVertex2d(y1,-x1);  glVertex2d(x1,-y1);  x1++;  if(p<0)  {  p=p+4\*(x1)+6;  }  else  {  y1=y1-1;  p=p+4\*x1-4\*y1+10;  }  }  glEnd();  glFlush();  }  int main(int argc,char\*\* argv)  {  glutInit(&argc,argv);  glutInitDisplayMode(GLUT\_SINGLE);  glutInitWindowSize(500,500);  glutInitWindowPosition(100,100);  glutCreateWindow("Circle Mid Point");  glutDisplayFunc(disp);  init();  glutMainLoop();  return 0;  } |

**Output:**

1. **Drawing circle using Mid point algorithm.**

|  |
| --- |
| #include<GL/glut.h>  void init()  {  glClearColor(1,1,1,1);  glClear(GL\_COLOR\_BUFFER\_BIT);  glLoadIdentity();  glMatrixMode(GL\_PROJECTION);  gluOrtho2D(-500,500,-500,500);  }  void disp()  {  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3d(1,0,0);  int r=300;  int y1=r,x1=0;  int p=1-r;  glBegin(GL\_POINTS);  while(x1<y1)  {  glVertex2d(x1,y1);  glVertex2d(y1,x1);  glVertex2d(-x1,y1);  glVertex2d(-y1,x1);  glVertex2d(-x1,-y1);  glVertex2d(-y1,-x1);  glVertex2d(y1,-x1);  glVertex2d(x1,-y1);  x1++;  if(p<0)  {  p=p+2\*(x1)+1;  }  else  {  y1=y1-1;  p=p+2\*x1-2\*y1+1;  }  }  glEnd();  glFlush();  }  int main(int argc,char\*\* argv)  {  glutInit(&argc,argv);  glutInitDisplayMode(GLUT\_SINGLE);  glutInitWindowSize(500,500);  glutInitWindowPosition(100,100);  glutCreateWindow("Circle Mid Point");  glutDisplayFunc(disp);  init();  glutMainLoop();  return 0;  } |

**Output:**



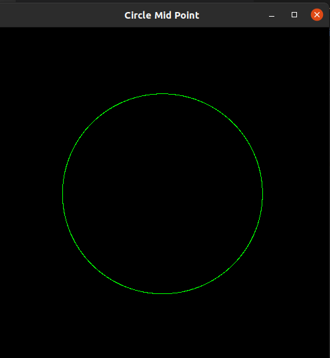
**Experiment 4**

1. **Drawing ellipse using mid point theorem.**

**Code:**

|  |
| --- |
| #include<GL/glut.h>  void init()  {  glClearColor(1,1,1,1);  glClear(GL\_COLOR\_BUFFER\_BIT);  glLoadIdentity();  glMatrixMode(GL\_PROJECTION);  gluOrtho2D(-500,500,-500,500);  }  void disp()  {  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3d(1,0,0);  int rx=300,ry=350;  int y1=ry,x1=0;  int p=(ry\*ry)+((rx\*rx)/4)-(ry\*rx\*rx);  glBegin(GL\_POINTS);  int dy=2\*y1\*rx\*rx,dx=2\*x1\*ry\*ry;  while(dx<dy)  {  dy=2\*y1\*rx\*rx;  dx=2\*x1\*ry\*ry;  glVertex2d(x1,y1);  glVertex2d(-x1,y1);  glVertex2d(-x1,-y1);  glVertex2d(x1,-y1);  x1++;  if(p<0)  {  p=p+ry\*ry\*2\*(x1)+(ry\*ry);  }  else  {  y1=y1-1;  p=p+ry\*ry\*2\*x1+ry\*ry-rx\*rx\*2\*y1;  }  }  while(y1>=0)  {  glVertex2d(x1,y1);  glVertex2d(-x1,y1);  glVertex2d(-x1,-y1);  glVertex2d(x1,-y1);  y1--;  if(p>=0)  {  p=p-rx\*rx\*2\*(y1)+rx\*rx;  }  else  {  x1=x1+1;  p=p-(rx\*rx\*2\*y1)+(ry\*ry\*2\*x1)+rx\*rx;  }  }  glEnd();  glFlush();  }  int main(int argc,char\*\* argv)  {  glutInit(&argc,argv);  glutInitDisplayMode(GLUT\_SINGLE);  glutInitWindowSize(500,500);  glutInitWindowPosition(100,100);  glutCreateWindow("Circle Mid Point");  glutDisplayFunc(disp);  init();  glutMainLoop();  return 0;  } |

**Output:**



**Experiment 5**

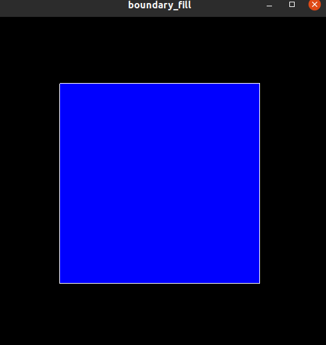
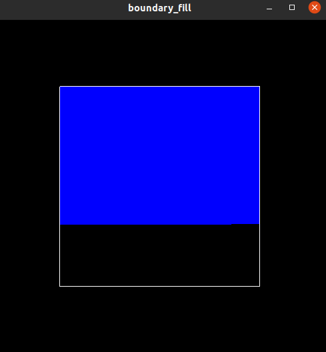
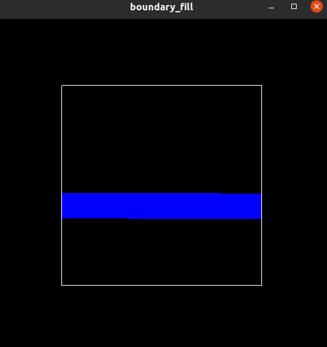
**Filling the objects**

1. **Filling the square using boundary fill algorithm.**

**Code:**

|  |
| --- |
| #include<GL/glut.h>  #include<iostream>  using namespace std;  typedef struct color{GLubyte red;GLubyte green;GLubyte blue;} color;  void init()  {  glClearColor(0,0,0,1);  glClear(GL\_COLOR\_BUFFER\_BIT);  glLoadIdentity();  glMatrixMode(GL\_PROJECTION);  gluOrtho2D(0,500,0,500);  }  void b\_fill(float x,float y,color b,color f)  {  color e;  glReadPixels(x,y,1,1,GL\_RGB,GL\_UNSIGNED\_BYTE,&e);  //cout<<x<<" "<<y<<" "<<(int)e.red<<" "<<(int)e.green<<" "<<(int)e.blue<<endl;  int count=0;  //cout<<"hohoho...";  if((int)b.red==(int)e.red) count++;  if((int)b.green==(int)e.green) count++;  if((int)b.blue == (int)e.blue) count++;  //cout<<count<<endl;  if(count!=3)  {  int count=0;  if((int)f.red==(int)e.red) count++;  if((int)f.green==(int)e.green) count++;  if((int)f.blue == (int)e.blue) count++;  //cout<<count+1<<endl;  //cout<<"COmes here ,,,";  if(count!=3)  {  glBegin(GL\_POINTS);  glColor3ub(f.red,f.green,f.blue);  glVertex2f(x,y);  glEnd();  glFlush();  // glReadPixels(x,y,1,1,GL\_RGB,GL\_UNSIGNED\_BYTE,&e);  // cout<<x<<" "<<y<<" "<<(int)e.red<<" "<<(int)e.green<<" "<<(int)e.blue<<endl;  b\_fill(x+1,y,b,f);  b\_fill(x-1,y,b,f);  b\_fill(x,y+1,b,f);  b\_fill(x,y-1,b,f);  }  // cout<<"yoyoyo";  }  // cout<<"yoyoyo";  }  void disp()  {  glClear(GL\_COLOR\_BUFFER\_BIT);  color b\_color,f\_color;  b\_color.red=255;  b\_color.green=255;  b\_color.blue=255;  f\_color.red=0;  f\_color.green=0;  f\_color.blue=255;  glBegin(GL\_LINES);  glColor3ub(255,255,255);  glVertex2f(100,100);  glVertex2f(100,400);  glVertex2f(100,400);  glVertex2f(400,400);  glVertex2f(400,400);  glVertex2f(400,100);  glVertex2f(400,100);  glVertex2f(100,100);  glEnd();  b\_fill(200,200,b\_color,f\_color);  glFlush();  }  int main(int a,char\*\* b)  {  glutInit(&a,b);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB );  glutInitWindowSize(500,500);  glutCreateWindow("boundary\_fill");  glutDisplayFunc(disp);  init();  glutMainLoop();  return 0;  } |

**Output:**

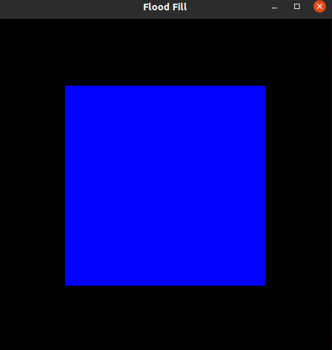
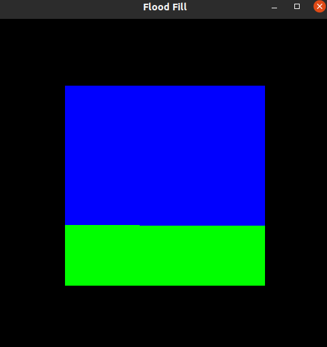
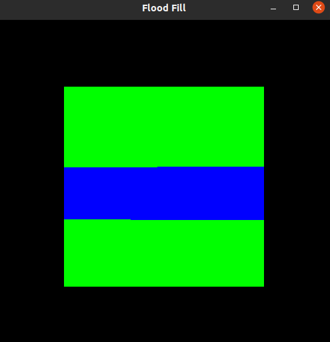


1. **Filling the rectangle using flood fill algorithm.**

**Code:**

|  |
| --- |
| #include<GL/glut.h>  typedef struct color{GLubyte red; GLubyte green; GLubyte blue;}color;  void init()  {  glClearColor(0,0,0,1);  glClear(GL\_COLOR\_BUFFER\_BIT);  glLoadIdentity();  glMatrixMode(GL\_PROJECTION);  gluOrtho2D(0,500,0,500);  }  void f\_fill(int x,int y,color e,color n)  {  color o;  glReadPixels(x,y,1,1,GL\_RGB,GL\_UNSIGNED\_BYTE,&o);  if(o.red==e.red && o.green==e.green && o.blue==e.blue)  {  glBegin(GL\_POINTS);  glColor3ub(n.red,n.green,n.blue);  glVertex2f(x,y);  glEnd();  glFlush();  f\_fill(x+1,y,e,n);  f\_fill(x-1,y,e,n);  f\_fill(x,y+1,e,n);  f\_fill(x,y-1,e,n);  }  }  void disp()  {  color e\_color;  e\_color.red=0;  e\_color.green=255;  e\_color.blue=0;  glBegin(GL\_POLYGON);  glColor3ub(e\_color.red,e\_color.green,e\_color.blue);  glVertex2f(100,100);  glVertex2f(100,400);  glVertex2f(400,400);  glVertex2f(400,100);  glEnd();  color n\_color;  n\_color.red=0;  n\_color.green=0;  n\_color.blue=255;  f\_fill(200,200,e\_color,n\_color);  glFlush();  }    int main(int a,char \*\* b)  {  glutInit(&a,b);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB );  glutInitWindowSize(500,500);  glutCreateWindow("Flood Fill");  glutDisplayFunc(disp);  init();  glutMainLoop();  return 0;  } |

**Output:**



**Experiment 6**

1. **Write a program for 2D transformation:**

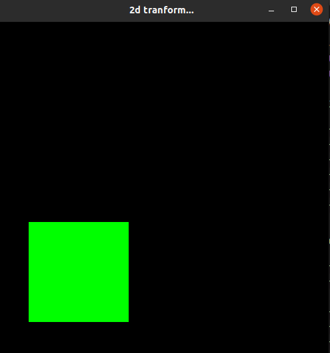
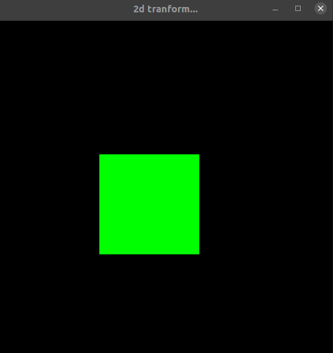
**a. translation, b. scaling, c. rotation,**

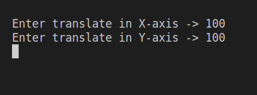
**Code:**

**2D - Translation:**

|  |
| --- |
| #include<GL/glut.h>  #include<iostream>  using namespace std;  void init()  {  glClearColor(0,0,0,1);  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3ub(0,255,0);  glLoadIdentity();  glMatrixMode(GL\_PROJECTION);  gluOrtho2D(0,500,0,500);  }  void draw\_square(int x1,int y1,int x2,int y2)  {  glClear(GL\_COLOR\_BUFFER\_BIT);  glBegin(GL\_POLYGON);  glVertex2f(x1,y1);  glVertex2f(x1,y2);  glVertex2f(x2,y2);  glVertex2f(x2,y1);  glEnd();  glFlush();  }  void disp()  {  int x1=50,y1=50,x2=200,y2=200;  draw\_square(x1,y1,x2,y2);  int transx,transy;  cout<<"Enter translate in X-axis -> ";  cin>>transx;  cout<<"Enter translate in Y-axis -> ";  cin>>transy;  draw\_square(x1+transx,y1+transy,x2+transx,y2+transy);  }  int main(int a,char \*\* b)  {  glutInit(&a,b);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB );  glutInitWindowSize(500,500);  glutCreateWindow("2d tranform...");  glutDisplayFunc(disp);  init();  glutMainLoop();  return 0;  } |

**Output:**

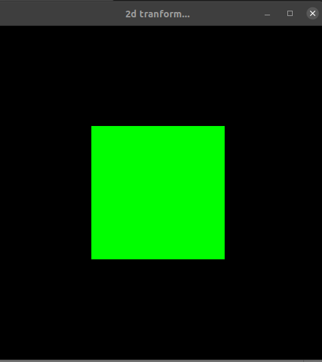
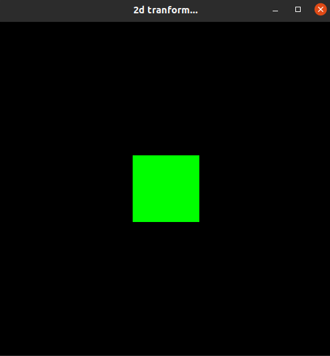
 

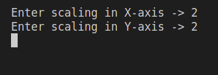


**2D - Scaling**

|  |
| --- |
| #include<GL/glut.h>  #include<iostream>  using namespace std;  void init()  {  glClearColor(0,0,0,1);  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3ub(0,255,0);  glLoadIdentity();  glMatrixMode(GL\_PROJECTION);  gluOrtho2D(-500,500,-500,500);  }  void draw\_square(int x1,int y1,int x2,int y2)  {  glClear(GL\_COLOR\_BUFFER\_BIT);  glBegin(GL\_POLYGON);  glVertex2f(x1,y1);  glVertex2f(x1,y2);  glVertex2f(x2,y2);  glVertex2f(x2,y1);  glEnd();  glFlush();  }  void mat\_mul(int a[3][3],int b[3][1],int r[3][1])  {  for(int i=0;i<3;i++)  {  for(int j=0;j<1;j++)  {  r[i][j]=0;  }  }  for(int i=0;i<3;i++)  {  for(int j=0;j<1;j++)  {  for(int k=0;k<3;k++)  {  r[i][j]+=(a[i][k]\*b[k][j]);  }  }  }  }  void disp()  {  int x1=100,y1=100,x2=-100,y2=-100;  draw\_square(x1,y1,x2,y2);  int s1,s2;  cout<<"Enter scaling in X-axis -> ";  cin>>s1;  cout<<"Enter scaling in Y-axis -> ";  cin>>s2;  int mat[3][3];  for(int i=0;i<3;i++)  {  for(int j=0;j<3;j++)  {  if(i==j)  {  mat[i][j]=1;  }  else  {  mat[i][j]=0;  }  }  }  mat[0][0]=s1;  mat[1][1]=s2;  int coordinate[3][1];  coordinate[0][0]=x1;  coordinate[1][0]=y1;  coordinate[2][0]=1;  int multiply[3][1];  mat\_mul(mat,coordinate,multiply);  x1=multiply[0][0];  y1=multiply[1][0];  cout<<x1<<" "<<y1;  coordinate[0][0]=x2;  coordinate[1][0]=y2;  mat\_mul(mat,coordinate,multiply);  x2=multiply[0][0];  y2=multiply[1][0];  draw\_square(x1,y1,x2,y2);  }  int main(int a,char \*\* b)  {  glutInit(&a,b);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB );  glutInitWindowSize(500,500);  glutCreateWindow("2d tranform...");  glutDisplayFunc(disp);  init();  glutMainLoop();  } |

**Output:**

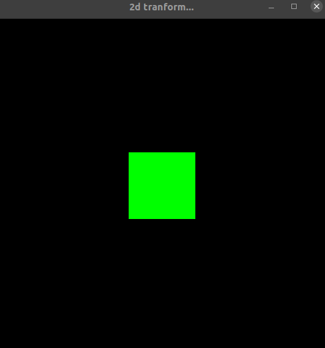
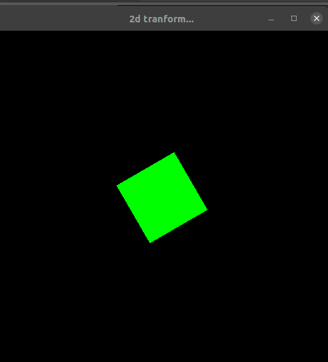




**2D - Rotation**

|  |
| --- |
| #include<GL/glut.h>  #include<math.h>  #include<iostream>  using namespace std;  float pi=3.14;  void init()  {  glClearColor(0,0,0,1);  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3ub(0,255,0);  glLoadIdentity();  glMatrixMode(GL\_PROJECTION);  gluOrtho2D(-500,500,-500,500);  }  void draw\_square(float x1,float y1,float x2,float y2,float x3,float y3,float x4,float y4)  {  glClear(GL\_COLOR\_BUFFER\_BIT);  glBegin(GL\_POLYGON);  glVertex2f(x1,y1);  glVertex2f(x2,y2);  glVertex2f(x3,y3);  glVertex2f(x4,y4);  glEnd();  glFlush();  }  void mat\_mul(float a[3][3],float b[3][1],float r[3][1])  {  for(int i=0;i<3;i++)  {  for(int j=0;j<1;j++)  {  r[i][j]=0;  }  }  for(int i=0;i<3;i++)  {  for(int j=0;j<1;j++)  {  for(int k=0;k<3;k++)  {  r[i][j]+=(a[i][k]\*b[k][j]);  }  }  }  }  void disp()  {  float x1=100,y1=100,x2=-100,y2=-100;  draw\_square(x1,y1, x1,y2, x2,x2, x2,y1);  float theta;  cout<<"Enter angle of rotation -> ";  cin>>theta;  theta=theta\*(pi/180);  float mat[3][3];  for(int i=0;i<3;i++)  {  for(int j=0;j<3;j++)  {  mat[i][j]=0;  }  }  mat[0][0]=cos(theta);  mat[1][1]=cos(theta);  mat[0][1]=sin(theta);  mat[1][0]=(-1)\*sin(theta);  mat[2][2]=1;  float coordinate[3][1];  coordinate[0][0]=x1;  coordinate[1][0]=y1;  coordinate[2][0]=1;  float multiply[3][1];  mat\_mul(mat,coordinate,multiply);  float x1f=multiply[0][0];  float y1f=multiply[1][0];  //cout<<x1<<" "<<y1;  coordinate[0][0]=x1;  coordinate[1][0]=y2;  mat\_mul(mat,coordinate,multiply);  float x2f=multiply[0][0];  float y2f=multiply[1][0];  coordinate[0][0]=x2;  coordinate[1][0]=y2;  mat\_mul(mat,coordinate,multiply);  float x3f=multiply[0][0];  float y3f=multiply[1][0];  coordinate[0][0]=x2;  coordinate[1][0]=y1;  mat\_mul(mat,coordinate,multiply);  float x4f=multiply[0][0];  float y4f=multiply[1][0];    draw\_square(x1f,y1f,x2f,y2f,x3f,y3f,x4f,y4f);  }  int main(int a,char \*\* b)  {  glutInit(&a,b);  glutInitDisplayMode( GLUT\_SINGLE | GLUT\_RGB );  glutInitWindowSize(500,500);  glutCreateWindow("2d tranform...");  glutDisplayFunc(disp);  init();  glutMainLoop();  } |

**Output:**

1. **Write a program for 3D transformation:**

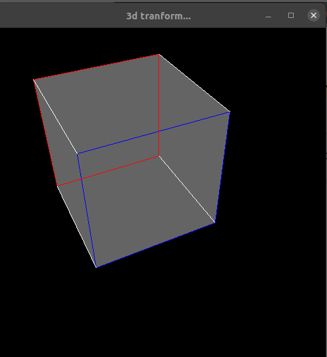
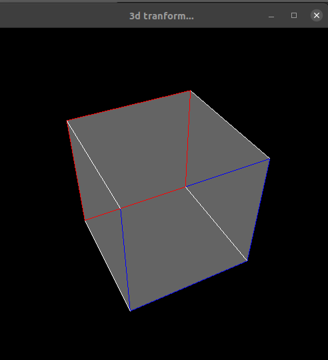
**a. translation, b. scaling, c. rotation, d. reflection**

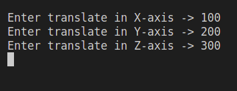
**Code:**

**3D - Translation**

|  |
| --- |
| #include<GL/glut.h>  #include<iostream>  using namespace std;  int height=500,width=500;  void init()  {  glClearColor(0,0,0,1);  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3ub(0,255,0);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluPerspective(45.0f,float(width)/height,1.0f,100.0f);  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  gluLookAt(1,2,-2,  0,0,0,  0,1,0);  //gluOrtho2D(0,500,0,500);  }  void boundary(float x1,float y1,float z1,float x2,float y2,float z2)  {  glBegin(GL\_LINES);  glColor3ub(255,255,255);  //e1  glColor3ub(0,0,255);  glVertex3f(x1,y1,z1);  glVertex3f(x2,y1,z1);  //e2  glVertex3f(x2,y1,z1);  glVertex3f(x2,y2,z1);  //e3  glVertex3f(x2,y2,z1);  glVertex3f(x1,y2,z1);  //e4  glVertex3f(x1,y2,z1);  glVertex3f(x1,y1,z1);  glColor3ub(255,0,0);  //e5  glVertex3f(x1,y1,z2);  glVertex3f(x2,y1,z2);  //e6  glVertex3f(x2,y1,z2);  glVertex3f(x2,y2,z2);  //e7  glVertex3f(x2,y2,z2);  glVertex3f(x1,y2,z2);  //e8  glVertex3f(x1,y2,z2);  glVertex3f(x1,y1,z2);  glColor3ub(255,255,255);  //e9  glVertex3f(x1,y2,z1);  glVertex3f(x1,y2,z2);  //e10  glVertex3f(x2,y2,z1);  glVertex3f(x2,y2,z2);  //e11  glVertex3f(x2,y1,z1);  glVertex3f(x2,y1,z2);  //e12  glVertex3f(x1,y1,z1);  glVertex3f(x1,y1,z2);      glEnd();  }  void draw\_square(float x1,float y1,float z1,float x2,float y2,float z2)  {  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  glBegin(GL\_QUADS);  glColor3ub(100,100,100);  //front  glVertex3f(x1,y1,z1);  glVertex3f(x1,y2,z1);  glVertex3f(x2,y2,z1);  glVertex3f(x2,y1,z1);  //back  glVertex3f(x1,y1,z2);  glVertex3f(x1,y2,z2);  glVertex3f(x2,y2,z2);  glVertex3f(x2,y1,z2);  //left  glVertex3f(x1,y1,z1);  glVertex3f(x1,y2,z1);  glVertex3f(x1,y2,z2);  glVertex3f(x1,y1,z2);  //right  glVertex3f(x2,y2,z1);  glVertex3f(x2,y1,z1);  glVertex3f(x2,y1,z2);  glVertex3f(x2,y2,z2);  //top  glVertex3f(x1,y2,z1);  glVertex3f(x2,y2,z1);  glVertex3f(x2,y2,z2);  glVertex3f(x1,y2,z2);  //bottom  glVertex3f(x1,y1,z1);  glVertex3f(x2,y1,z1);  glVertex3f(x2,y1,z2);  glVertex3f(x1,y1,z2);  glEnd();  boundary(x1,y1,z1,x2,y2,z2);  glFlush();  }  void disp()  {  float x1=-500,y1=-500,z1=-500,x2=500,y2=500,z2=500;  x1/=1000;  y1/=1000;  z1/=1000;  x2/=1000;  y2/=1000;  z2/=1000;    draw\_square(x1,y1,z1,x2,y2,z2);  float transx,transy,transz;  cout<<"Enter translate in X-axis -> ";  cin>>transx;  cout<<"Enter translate in Y-axis -> ";  cin>>transy;  cout<<"Enter translate in Z-axis -> ";  cin>>transz;  transx/=1000;  transy/=1000;  transz/=1000;  draw\_square(x1+transx,y1+transy,z1+transz,x2+transx,y2+transy,z2+transz);  }  int main(int a,char \*\* b)  {  glutInit(&a,b);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB );  glutInitWindowSize(500,500);  glutCreateWindow("3d tranform...");  glutDisplayFunc(disp);  init();  glutMainLoop();  } |

**Output:**

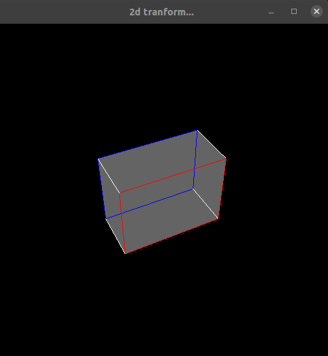
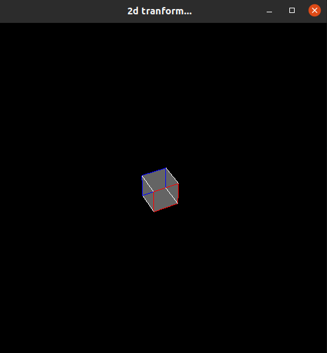


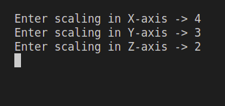


**3D - Scaling**

|  |
| --- |
| #include<GL/glut.h>  #include<iostream>  using namespace std;  int width=500,height=500;  void init()  {  glClearColor(0,0,0,1);  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3ub(0,255,0);    glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluPerspective(45.0f,float(width)/height,1.0f,100.0f);  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  gluLookAt(1,2,-2,  0,0,0,  0,1,0);  }  void boundary(float x1,float y1,float z1,float x2,float y2,float z2)  {  glBegin(GL\_LINES);  glColor3ub(255,255,255);  //e1  glColor3ub(0,0,255);  glVertex3f(x1,y1,z1);  glVertex3f(x2,y1,z1);  //e2  glVertex3f(x2,y1,z1);  glVertex3f(x2,y2,z1);  //e3  glVertex3f(x2,y2,z1);  glVertex3f(x1,y2,z1);  //e4  glVertex3f(x1,y2,z1);  glVertex3f(x1,y1,z1);  glColor3ub(255,0,0);  //e5  glVertex3f(x1,y1,z2);  glVertex3f(x2,y1,z2);  //e6  glVertex3f(x2,y1,z2);  glVertex3f(x2,y2,z2);  //e7  glVertex3f(x2,y2,z2);  glVertex3f(x1,y2,z2);  //e8  glVertex3f(x1,y2,z2);  glVertex3f(x1,y1,z2);  glColor3ub(255,255,255);  //e9  glVertex3f(x1,y2,z1);  glVertex3f(x1,y2,z2);  //e10  glVertex3f(x2,y2,z1);  glVertex3f(x2,y2,z2);  //e11  glVertex3f(x2,y1,z1);  glVertex3f(x2,y1,z2);  //e12  glVertex3f(x1,y1,z1);  glVertex3f(x1,y1,z2);      glEnd();  }  void draw\_cube(float x1,float y1,float z1,float x2,float y2,float z2)  {  glClear(GL\_COLOR\_BUFFER\_BIT);  glBegin(GL\_POLYGON);  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  glBegin(GL\_QUADS);  glColor3ub(100,100,100);  //front  glVertex3f(x1,y1,z1);  glVertex3f(x1,y2,z1);  glVertex3f(x2,y2,z1);  glVertex3f(x2,y1,z1);  //back  glVertex3f(x1,y1,z2);  glVertex3f(x1,y2,z2);  glVertex3f(x2,y2,z2);  glVertex3f(x2,y1,z2);  //left  glVertex3f(x1,y1,z1);  glVertex3f(x1,y2,z1);  glVertex3f(x1,y2,z2);  glVertex3f(x1,y1,z2);  //right  glVertex3f(x2,y2,z1);  glVertex3f(x2,y1,z1);  glVertex3f(x2,y1,z2);  glVertex3f(x2,y2,z2);  //top  glVertex3f(x1,y2,z1);  glVertex3f(x2,y2,z1);  glVertex3f(x2,y2,z2);  glVertex3f(x1,y2,z2);  //bottom  glVertex3f(x1,y1,z1);  glVertex3f(x2,y1,z1);  glVertex3f(x2,y1,z2);  glVertex3f(x1,y1,z2);  glEnd();  boundary(x1,y1,z1,x2,y2,z2);  glFlush();  glEnd();  glFlush();  }  void mat\_mul(float a[4][4],float b[4][1],float r[4][1])  {  for(int i=0;i<3;i++)  {  for(int j=0;j<1;j++)  {  r[i][j]=0;  }  }  for(int i=0;i<3;i++)  {  for(int j=0;j<1;j++)  {  for(int k=0;k<3;k++)  {  r[i][j]+=(a[i][k]\*b[k][j]);  }  }  }  }  void disp()  {  float x1=100,y1=100,z1=100,x2=-100,y2=-100,z2=-100;  x1/=1000;  y1/=1000;  z1/=1000;  x2/=1000;  y2/=1000;  z2/=1000;  draw\_cube(x1,y1,z1,x2,y2,z2);  int s1,s2,s3;  cout<<"Enter scaling in X-axis -> ";  cin>>s1;  cout<<"Enter scaling in Y-axis -> ";  cin>>s2;  cout<<"Enter scaling in Z-axis -> ";  cin>>s3;  float mat[4][4];  for(int i=0;i<3;i++)  {  for(int j=0;j<3;j++)  {  if(i==j)  {  mat[i][j]=1;  }  else  {  mat[i][j]=0;  }  }  }  mat[0][0]=s1;  mat[1][1]=s2;  mat[2][2]=s3;  float coordinate[4][1];  coordinate[0][0]=x1;  coordinate[1][0]=y1;  coordinate[2][0]=z1;  coordinate[3][0]=1;    float multiply[4][1];  mat\_mul(mat,coordinate,multiply);  x1=multiply[0][0];  y1=multiply[1][0];  z1=multiply[2][0];  cout<<x1<<" "<<y1;  coordinate[0][0]=x2;  coordinate[1][0]=y2;  coordinate[2][0]=z2;  mat\_mul(mat,coordinate,multiply);  x2=multiply[0][0];  y2=multiply[1][0];  z2=multiply[2][0];  draw\_cube(x1,y1,z1,x2,y2,z2);  }  int main(int a,char \*\* b)  {  glutInit(&a,b);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB );  glutInitWindowSize(500,500);  glutCreateWindow("2d tranform...");  glutDisplayFunc(disp);  init();  glutMainLoop();  } |

**Output:**

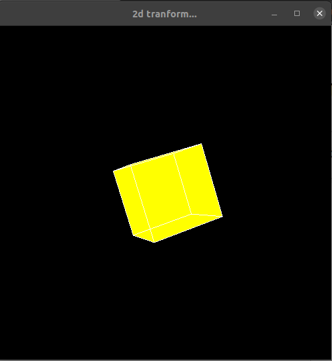
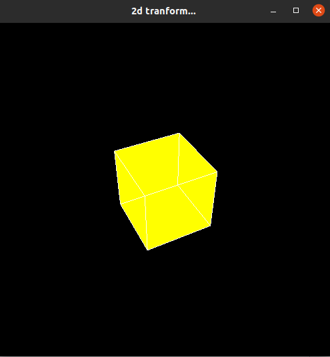




**3D - Rotation**

|  |
| --- |
| #include<GL/glut.h>  #include<math.h>  #include<iostream>  using namespace std;  float pi=3.14;  int width=500,height=500;  void init()  {  glClearColor(0,0,0,1);  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3ub(0,255,0);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluPerspective(50.0f,float(width)/height,1.0f,100.0f);  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  gluLookAt(1,2,-2,  0,0,0,  0,1,0);  }  void boundary(float vertices[][3])  {  glBegin(GL\_LINES);  glColor3ub(255,255,255);  //e1  glVertex3f(vertices[0][0],vertices[0][1],vertices[0][2]);  glVertex3f(vertices[1][0],vertices[1][1],vertices[1][2]);  //e2  glVertex3f(vertices[1][0],vertices[1][1],vertices[1][2]);  glVertex3f(vertices[2][0],vertices[2][1],vertices[2][2]);  //e3  glVertex3f(vertices[2][0],vertices[2][1],vertices[2][2]);  glVertex3f(vertices[3][0],vertices[3][1],vertices[3][2]);  //e4  glVertex3f(vertices[3][0],vertices[3][1],vertices[3][2]);  glVertex3f(vertices[0][0],vertices[0][1],vertices[0][2]);  //e5  glVertex3f(vertices[4][0],vertices[4][1],vertices[4][2]);  glVertex3f(vertices[5][0],vertices[5][1],vertices[5][2]);  //e6  glVertex3f(vertices[5][0],vertices[5][1],vertices[5][2]);  glVertex3f(vertices[6][0],vertices[6][1],vertices[6][2]);  //e7  glVertex3f(vertices[6][0],vertices[6][1],vertices[6][2]);  glVertex3f(vertices[7][0],vertices[7][1],vertices[7][2]);  //e8  glVertex3f(vertices[7][0],vertices[7][1],vertices[7][2]);  glVertex3f(vertices[4][0],vertices[4][1],vertices[4][2]);  //e9  glVertex3f(vertices[1][0],vertices[1][1],vertices[1][2]);  glVertex3f(vertices[5][0],vertices[5][1],vertices[5][2]);  //e10  glVertex3f(vertices[0][0],vertices[0][1],vertices[0][2]);  glVertex3f(vertices[4][0],vertices[4][1],vertices[4][2]);  //e11  glVertex3f(vertices[3][0],vertices[3][1],vertices[3][2]);  glVertex3f(vertices[7][0],vertices[7][1],vertices[7][2]);  //e12  glVertex3f(vertices[2][0],vertices[2][1],vertices[2][2]);  glVertex3f(vertices[6][0],vertices[6][1],vertices[6][2]);  glEnd();  }  void draw\_cube(float vertices[8][3])  {  glClear(GL\_COLOR\_BUFFER\_BIT);  glColor3ub(255,255,0);  glBegin(GL\_POLYGON);  //front  glVertex3f(vertices[0][0],vertices[0][1],vertices[0][2]);  glVertex3f(vertices[1][0],vertices[1][1],vertices[1][2]);  glVertex3f(vertices[2][0],vertices[2][1],vertices[2][2]);  glVertex3f(vertices[3][0],vertices[3][1],vertices[3][2]);  glEnd();  //back  glBegin(GL\_POLYGON);  glVertex3f(vertices[4][0],vertices[4][1],vertices[4][2]);  glVertex3f(vertices[5][0],vertices[5][1],vertices[5][2]);  glVertex3f(vertices[6][0],vertices[6][1],vertices[6][2]);  glVertex3f(vertices[7][0],vertices[7][1],vertices[7][2]);  glEnd();  //left  glBegin(GL\_POLYGON);  glVertex3f(vertices[1][0],vertices[1][1],vertices[1][2]);  glVertex3f(vertices[2][0],vertices[2][1],vertices[2][2]);  glVertex3f(vertices[6][0],vertices[6][1],vertices[6][2]);  glVertex3f(vertices[5][0],vertices[5][1],vertices[5][2]);  glEnd();  //top  glBegin(GL\_POLYGON);  glVertex3f(vertices[0][0],vertices[0][1],vertices[0][2]);  glVertex3f(vertices[1][0],vertices[1][1],vertices[1][2]);  glVertex3f(vertices[5][0],vertices[5][1],vertices[5][2]);  glVertex3f(vertices[4][0],vertices[4][1],vertices[4][2]);  glEnd();  //right  glBegin(GL\_POLYGON);  glVertex3f(vertices[0][0],vertices[0][1],vertices[0][2]);  glVertex3f(vertices[3][0],vertices[3][1],vertices[3][2]);  glVertex3f(vertices[7][0],vertices[7][1],vertices[7][2]);  glVertex3f(vertices[4][0],vertices[4][1],vertices[4][2]);  glEnd();  //bottom  glBegin(GL\_POLYGON);  glVertex3f(vertices[0][0],vertices[0][1],vertices[0][2]);  glVertex3f(vertices[2][0],vertices[2][1],vertices[2][2]);  glVertex3f(vertices[3][0],vertices[3][1],vertices[3][2]);  glVertex3f(vertices[7][0],vertices[7][1],vertices[7][2]);  glEnd();  boundary(vertices);  glFlush();  }  void mat\_mul(float a[4][4],float b[4][1],float r[4][1])  {  for(int i=0;i<3;i++)  {  for(int j=0;j<1;j++)  {  r[i][j]=0;  }  }  for(int i=0;i<3;i++)  {  for(int j=0;j<1;j++)  {  for(int k=0;k<3;k++)  {  r[i][j]+=(a[i][k]\*b[k][j]);  }  }  }  }  void disp()  {  float vertices[][3]={  {300,300,300},  {-300,300,300},  {-300,-300,300},  {300,-300,300},  {300,300,-300},  {-300,300,-300},  {-300,-300,-300},  {300,-300,-300}  };  for(int i=0;i<8;i++)  {  for(int j=0;j<3;j++)  {  vertices[i][j]/=1000;  }  }  draw\_cube(vertices);  float theta;  cout<<"Enter angle of rotation -> ";  cin>>theta;  theta=theta\*(pi/180);  float mat[4][4];  for(int i=0;i<4;i++)  {  for(int j=0;j<4;j++)  {  mat[i][j]=0;  }  }  mat[0][0]=1;  mat[1][1]=cos(theta);  mat[1][2]=sin(theta);  mat[2][1]=(-1)\*sin(theta);  mat[2][2]=cos(theta);  mat[3][3]=1;  float coordinates[4][1];  coordinates[3][0]=1;  for(int i=0;i<8;i++)  {  float result[4][1];  coordinates[0][0]=vertices[i][0];  coordinates[1][0]=vertices[i][1];  coordinates[2][0]=vertices[i][2];  mat\_mul(mat,coordinates,result);  vertices[i][0]=result[0][0];  vertices[i][1]=result[1][0];  vertices[i][2]=result[2][0];  }  draw\_cube(vertices);  }  int main(int a,char \*\* b)  {  glutInit(&a,b);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB );  glutInitWindowSize(500,500);  glutCreateWindow("2d tranform...");  glutDisplayFunc(disp);  init();  glutMainLoop();  } |

**Output:**





**Experiment 7**

1. **Write a program for Line Clipping Algorithm:**

**Code -:**

|  |
| --- |
| #include<GL/glut.h>  #include<iostream>  using namespace std;  #define size 4  int winx1=100,winy1=100,winx2=400,winy2=400;  int total\_vertex=4;  int v[][2]={  {50,50},  {350,350},  {500,100},  {100,500}  };  void init()  {  glClearColor(0,0,0,1);  glColor3ub(255,255,255);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluOrtho2D(0,500,0,500);  }  int and\_operator(int a[],int b[])  {  for(int i=0;i<size;i++)  {  if(a[i]==1 && b[i]==1)  {  return 0;  }  }  return 1;  }  int yOnX(int x1,int y1,int x2,float m)  {  // cout<<"slope ->"<<x1<<" "<<y1<<" "<<x2<<" "<<" "<<m<<" " <<float(y1)+m\*float(x2-x1)<<endl;  return float(y1)+m\*float(x2-x1);  }  int xOnY(int x1,int y1,int y2,float m)  {  // cout<<"slope ->"<<x1<<" "<<y1<<" "<<y2<<" "<<" "<<m<<" "<<float(x1)+(float(y2-y1)/m)<<endl;  return float(x1)+(float(y2-y1)/m);  }  void check(int v[2],int pos[4])  {  for(int i=0;i<size;i++)  {  pos[i]=0;  }  if(v[0]<winx1)  {  pos[0]=1;  }  else if(v[0]>winx2)  {  pos[1]=1;  }  if(v[1]<winy1)  {  pos[2]=1;  }  else if(v[0]>winy2)  {  pos[3]=1;  }  }  void drawline(int x1,int y1,int x2,int y2)  {  glBegin(GL\_LINES);  glVertex2f(x1,y1);  glVertex2f(x2,y2);  glEnd();  }  void disp()  {    glClear(GL\_COLOR\_BUFFER\_BIT);  for(int i=0;i<total\_vertex;i+=2)  {  drawline(v[i][0],v[i][1],v[i+1][0],v[i+1][1]);  }  glFlush();    }  void onClick(int button,int state,int x,int y)  {  if ( button==GLUT\_LEFT\_BUTTON )  {  glClear(GL\_COLOR\_BUFFER\_BIT);  int pos[4][4];  for(int i=0;i<total\_vertex;i++)  {  check(v[i],pos[i]);  }  //drawline(0,250,250,0);  for(int i=0;i<total\_vertex;i=i+2)  {  //cout<<i<<" Here..."<<endl;  int x1,y1,x2,y2;  //cout<<v[i][0]<<" "<<v[i][1]<<" - ";  // for(int j=0;j<4;j++)  // {  // cout<<pos[i][j]<<" "<<pos[i+1][j]<<endl;  // }  float m=float(v[i+1][1]-v[i][1])/float(v[i+1][0]-v[i][0]);  float delx=v[i+1][0]-v[i][0],dely=v[i+1][1]-v[i][1];  //cout<<and\_operator(pos[i],pos[i+1])<<endl;  if(and\_operator(pos[i],pos[i+1]))  {  //cout<<"Inside...";  if(v[i][0]>winx1 && v[i][0]<winx2 && v[i][1]>winy1 && v[i][1]<winy2)  {  x1=v[i][0];  y1=v[i][1];  }  else  {    int temp=xOnY(v[i][0],v[i][1],winy1,m);  if(delx>0 && temp>x2) temp=winx2+1;  if(delx<0 && temp<x2) temp=winx2+1;  if(temp>=winx1 && temp<=winx2)  {  x1=temp;  y1=winy1;  }  else{  temp=xOnY(v[i][0],v[i][1],winy2,m);  if(delx>0 && temp>x2) temp=winx2+1;  if(delx<0 && temp<x2) temp=winx2+1;  if(temp>=winx1 && temp<=winx2)  {  x1=temp;  y1=winy2;  }  else  {  temp=yOnX(v[i][0],v[i][1],winx1,m);  if(dely>0 && temp>y2) temp=winy2+1;  if(dely<0 && temp<y2) temp=winy2+1;  if(temp>=winy1 && temp<=winy2)  {  x1=winx1;  y1=temp;  }  else  {  temp=yOnX(v[i][0],v[i][1],winx2,m);  if(dely>0 && temp>y2) temp=winy2+1;  if(dely<0 && temp<y2) temp=winy2+1;  if(temp>=winy1 && temp<=winy2)  {  x1=winx2;  y1=temp;  }  else  {  x1=0;  y1=0;  }  }  }  }  }  if(v[i+1][0]>winx1 && v[i+1][0]<winx2 && v[i+1][1]>winy1 && v[i+1][1]<winy2)  {  x2=v[i+1][0];  y2=v[i+1][1];  }  else  {    int temp=xOnY(v[i+1][0],v[i+1][1],winy1,m);  if(temp>=winx1 && temp<=winx2 && temp!=x1)  {  x2=temp;  y2=winy1;  }  else{  temp=xOnY(v[i+1][0],v[i+1][1],winy2,m);  if(temp>=winx1 && temp<=winx2 && temp!=x1)  {  x2=temp;  y2=winy2;  }  else  {  temp=yOnX(v[i+1][0],v[i+1][1],winx1,m);  if(temp>=winy1 && temp<=winy2 && temp!=y1)  {  x2=winx1;  y2=temp;  }  else  {  temp=yOnX(v[i+1][0],v[i+1][1],winx2,m);  if(temp>=winy1 && temp<=winy2 && temp!=y1)  {  x2=winx2;  y2=temp;  }  else  {  x2=0;  y2=0;  }  }  }  }  }  //cout<<"Drawline "<<x1<<" "<<y1<<" "<<x2<<" "<<y2<<endl;  drawline(x1,y1,x2,y2);  }  }  glFlush();  }  else if( button==GLUT\_RIGHT\_BUTTON )  {  disp();  }  }  int main(int a,char \*\*b)  {  glutInit(&a,b);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB );  glutInitWindowPosition(100,100);  glutInitWindowSize(500,500);  glutCreateWindow("Line Clipping...");  glutDisplayFunc(disp);  glutMouseFunc(onClick);  init();  glutMainLoop();  return 0;  } |

**Output -:**

