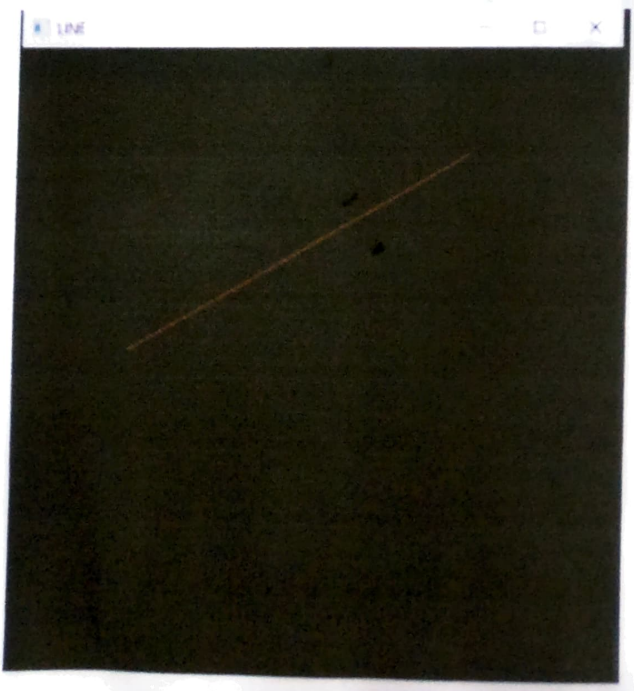


Program1: Line

Date :

Experiment No. : 1

Page No : 1

Write a program to generate a line using Bresenham's line drawing technique. Consider slopes greater than one and slopes less than one. User must be provided a menu and option input through keyboard/mouse.

```
#include<iostream>
#include<GL/glut.h>
#include<stdlib.h>
using namespace std;
int x1, x2, y1, y2;
int flag = 0;
void draw_pixel(int x, int y)
{ glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
    glFlush(); }
void draw_line()
{ int dx, dy, i, e;
  int incx, incy, inc1, inc2;
  int x, y;
  dx = x2 - x1;
  dy = y2 - y1;
  if (dx < 0) dx = -dx;
  if (dy < 0) dy = -dy;
  incx = 1;
  if (x2 < x1)
  incx = -1;
```

Date :.....

Experiment No. :.....1.....

Page No :.....2.....

```
x= x1;
y= y1;
if (dx>dy)
  { dum->pixel(x,y);
  e= 2*dy-dx;
    inc1= 2*(y-dx);
    inc2= 2*dy;
for(i=0; i<dx; i++)
   { if (e>0)
      { y+=inc;
        e+=inc1;
   } else
        e+=inc2;
      x+=inc2;
   }
} else {
dum->pixel(x,y);
    e= 2*dx-dy;
     inc1= 2*(dx-dy);
     inc2= 2*dx;
for(i=0; i<dy; i++) {
  if (e>0) {
     x+=incx;
      e+=inc1;
   } else
    e+=inc2;
      y+=incy;
     dum->pixel(x,y); }
```

Reva

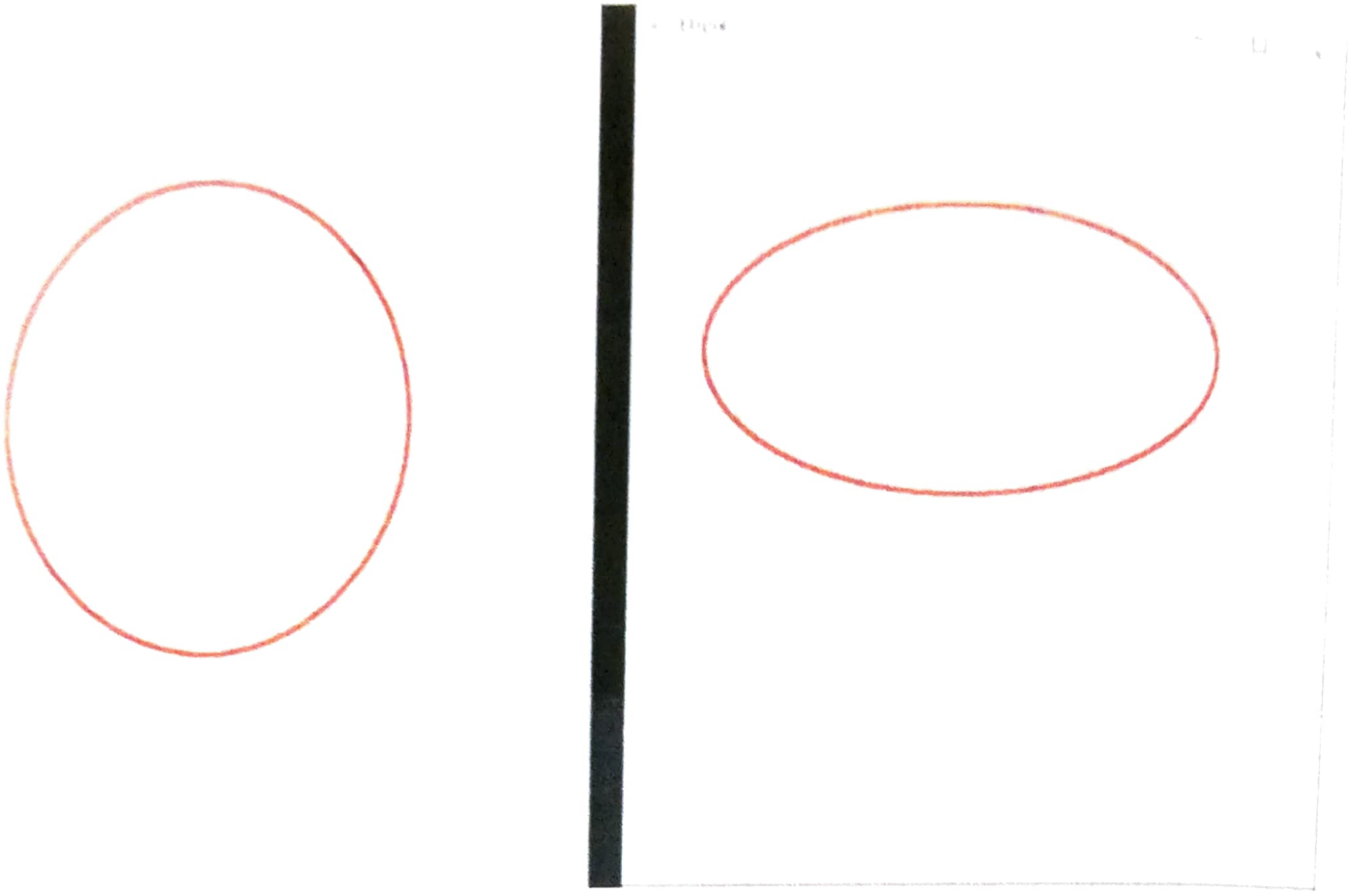
Date :.....

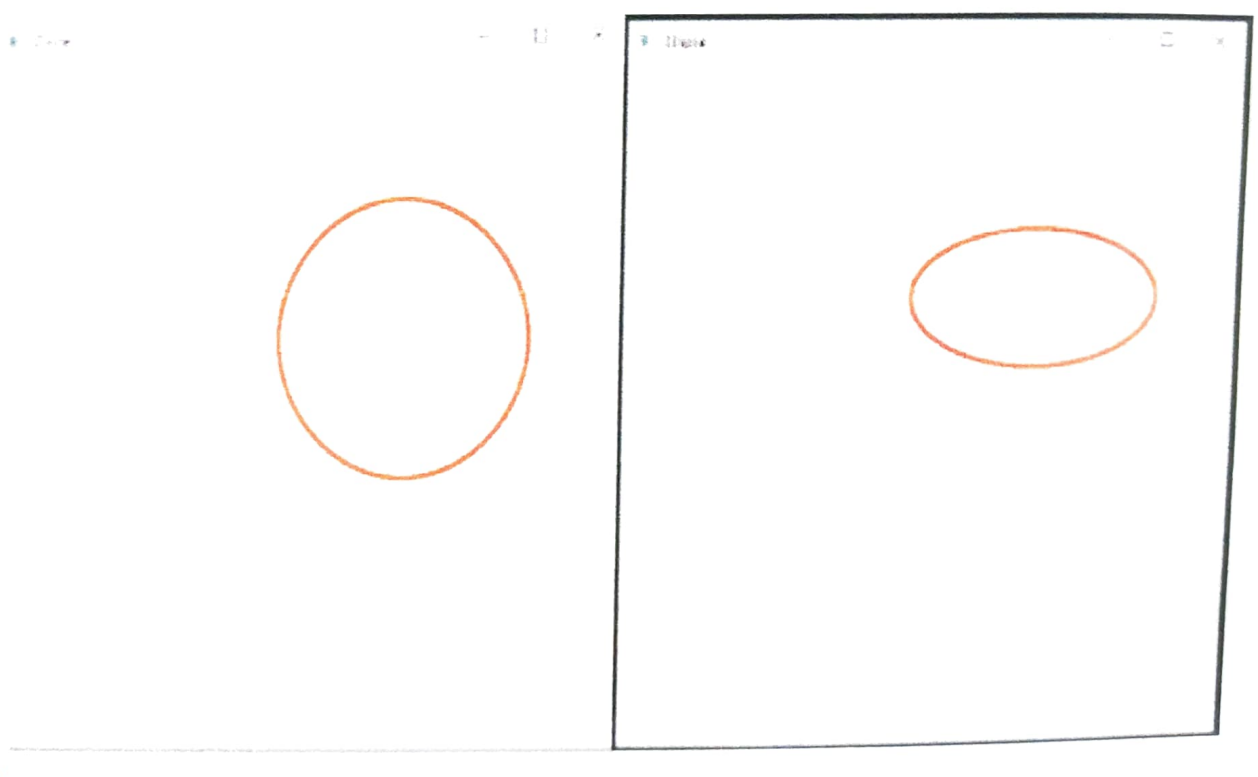
Experiment No. :.....1.....

Page No :.....3.....

```
void display()
{ }
int main(int argc, char *argv[])
{
cout << "enter x1 & y1 \n";
    cin >> x1 >> y1;
cout << "enter x2 & y2 \n";
    cin >> x2 >> y2;
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
 glutInitWindowSize(500, 500);
 glutInitWindowPosition(100, 200);
   glutCreateWindow("line");
   myinit();
glutReshapeFunc(myreshape);
   dda_line();
   glutDisplayFunc(display);
     glutMainLoop();
}
```

Reva





Program2: Circle and Ellipse

Date :

Experiment No. : 2

Page No : 4

Write a program to generate a circle & ellipse using Bresenham's circle drawing and ellipse drawing techniques. Use 2 windows to draw circle in one window & ellipse in the other window.

```
#include<GL/glut.h>
#include<stdio.h>
using namespace std;
int xc, yc, r;
int rx, ry, xcc, ycc;
void draw_circle(int xc, int yc, int x, int y)
{ glBegin(GL_POINTS);
  glVertex2i(xc+x, yc+y);
  glVertex2i(xc-x, yc+y);
  glVertex2i(xc+x, yc-y);
  glVertex2i(xc+y, yc+x);
  glVertex2i(xc-y, yc+x);
  glVertex2i(xc-y, yc-x);
  glEnd(); }
void circlebres()
{ glClear(GL_COLOR_BUFFER_BIT);
  int x=0, y=r;
  int d=3-2*r;
  while(x<=y) {
  draw_circle(xc, yc, x, y);
    x++;
  if(d<0)
    d=d+4*x+6;
```

Reva

Date :

Experiment No. :

Page No : 5

```
else {
    y--; d = d + 4 * (x-4) + 10*8;
    } draw_circle (xc, yc, x, y);
} glFlush();
} int h1_x, h2_x, h1_y, h2_y;
   int first_done = 0;
void mymouse fun(int button, int state, int x, int y)
{ if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN
  && first_done == 0) {
  h1_x = x - 250;
     h1_y = 250 - y; }
else if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
 { h2_x = x - 250; h2_y = 250 - y;
     xc = h1_x; yc = h1_y;
   float rad = (h2_x - h1_x) * (h2_x - h1_x) +
   (h2_y * - h1_y) * (h2_y - h1_y);
     r = (int) (sqrt (rad));
      display(); first_done = 0; }
void draw_ellipse (int xcc, int ycc, int x, int y)
 { glBegin (GL_POINTS);
   glVertex2i (x + xcc, y + ycc);
   glVertex2i (-x + xcc, y + ycc);
   glEnd(); }
void midEllipse()
{ glClear (GL_COLOR_BUFFER_BIT);
   float dx, dy, d1, d2, x, y;
   x = 0; y = (ry);
d1 = (ry * ry) - (rx * rx * ry) + (rx * rx * 0.25);
```

Date :.....

Experiment No. :.....

Page No :.....6.....

```
while (dx<dy) {
draw_ellipse (xcc, ycc, rx, ry);
  if (d1<0)
{  x++;  dx=dx+(2*ry*ry);
   d1=d1+dx+(ry*ry); }
else {
    x++;  y--;
  dx=dx+(2*ry*ry);
   dy=dy-(2*rx*rx);
   d1=d1+dx-dy+(ry*ry); }}
d2=((ry*ry)*((x+0.5)*(x+0.5))+((rx*rx)*
 (y-1)*(y-1))-(rx*rx*ry*ry);
 while(y>=0) {
draw_ellipse(xcc, ycc, x, y);
  if (d2>0) {
        y--;  dy=dy-(2*ry*rx);
     d2=d2+(rx*rx)-dy;
}} glFlush(); }
void mymouse(int button, int state, int x, int y) {
 if (button==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
   { h1c=x-250;  h1c-y=250-y;
   xcc=h1c-x;
     ycc=h1c-y;
      point(c-down=1;) }
else if (button==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
{ h2c-x=x-250;
   h2c-y=250-y;
   float eth=(h2c-x-h1c-x)*(h2c-y-h1c-y);
```

Date :

Experiment No. :

Page No : 7

```
void myinit()
{
  glClearColor(1, 1, 1, 1);
   glColor3f(1.0, 0.0, 0.0);
   glPointSize(3.0);
    gluOrtho2D(-250, 250, -250, 250);
}
void main(int argc, char *argv[])
 { glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
 glutInitWindowSize(500, 500);
 glutInitWindowPosition(0,0);
int id1 = glutCreateWindow("circle");
 glutSetWindow(id1);
glutMouseFunc(myMouseFunc);
  glutDisplayFunc(myDisplay);
 myinit();
glutInitWindowSize(500, 500);
 glutInitWindowPosition(600,100);
 glutSetWindow(id2);
 glutMouseFunc(myMouseFunc);
  glutDisplayFunc(myDisplay);
 myinit();
   glutMainLoop();
}
```

Reva

Program 3: Seirpinski Gasket

Date :

Experiment No. : 3

Page No : 8

Write a program to recursively subdivide a tetrahedron to form 3D sierpinski gesket. The no of recursive steps is to be specified at execution time.

```
#include <gl/glut.h>
#include <stdio.h>
int n;
typedef float point[3];
point tetra[4] = {{0,100,-100},{0,0,100},{100,-100,-100},{-100,-100,-100}};
void tetrahedron(void)
{
  glclear(GL_COLOR_BUFFER_BIT);
  glcolor3f(1.0,0.0,0.0);
  divide_triangle(tetra[0],tetra[1],tetra[2],n);
  glcolor3f(1.0,0.0,0.0);
  divide_triangle(tetra[3],tetra[2],tetra[1],n);
  glcolor3f(0.0,0.0,1.0);
  divide_triangle(tetra[0],tetra[2],tetra[3],n);
  glflush();
}
void div_triangle(point a,point b,point c)
{
  glbegin(GL_triangles);
    glvertex3fv(a);
    glvertex3fv(b);
    glvertex3fv(c);
    glend();
}
void divide_triangle(point a,point b,point c,int m)
{
  point v1,v2,v3;
  int j;
```

Reva

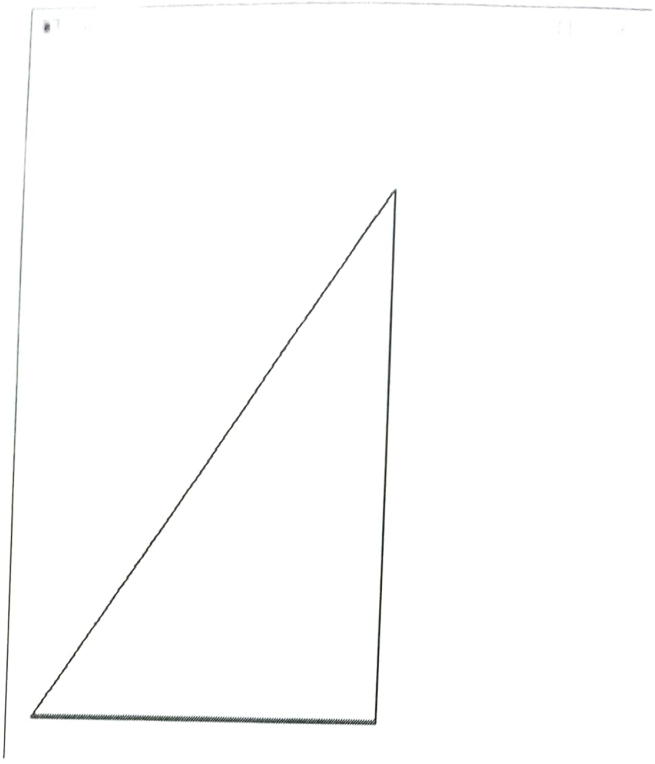
Date :

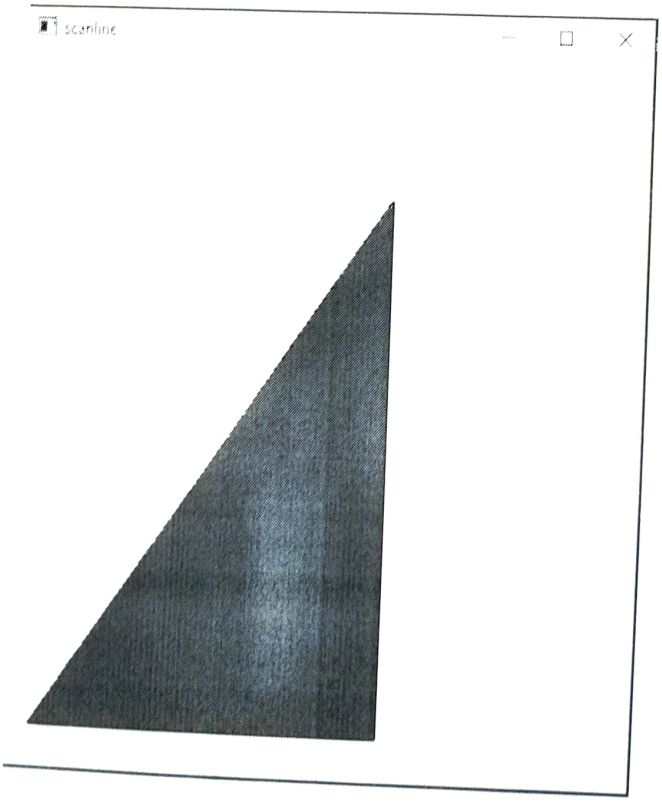
Experiment No. : 3

Page No : 9

```
if (n>0) {
for (j=0; j<3; j++)
    v1[j] = (a[j] + b[j])/2;
for (j=0; j<3; j++)
    v3[j] = (b[j] + c[j])/2;
draw_trigle(a, v1, v2, n-1);
draw_trigle(c, v2, v3, n-1);
} else
draw_trigle(a, b, c); }
int main(int argv, char **argc)
{ printf("enter the no of words");
scanf("%d", &n);
glutInit(&argv, argc);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowPosition(100, 200);
glutInitWindowSize(500, 500);
glutCreateWindow("Sierpinski basket");
glutDisplayFunc(triangle);
glClearColor(GL_DEPTH_TEST);
myinit();
glutMainLoop();
}
```

Reva





Program4: Scan fill

Date :

Experiment No. : 4

Page No : 10

Write a program to fill any given polygon using scan-line area filling algorithm.

```
#include<gl/glut.h>
#include<stdlib.h>
using namespace std;
float x[100], y[100];
int n, m;
static float int x[10]={0};
void display_lines( float x1, float y1, float x2, float y2){
    glClear(100);
    glColor3f(1,0,0);
    glVertex2f(x1, y1);
    glVertex2f(x2, y2);
    glEnd(); glFlush(); }
void edgedetect (float x1, float y1, float x2, float y2,
int scanline) {
float temp;
  if (y2<y1) {
temp=x1; x1=x2; x2=temp;
  temp=y1; y1=y2; y2=temp; }
if (scanline>y1 && scanline<y2)
    intx[m++]=x1+(scanline-y1)*(x2-x1)/(y2-y1);
void scanfill( float x[], float y[]) {
for(int s1=0; s1<=wy; s1++) {
  m=0;
for(int i=0; i<n; i++)
  edgedetect(x[i], y[i], x[(i+1)%n], y[(i+1)%n]);
```

3

Experiment No. : 4 Date :

Page No : 41

```
void (int *, (int *)+n);
if (n>=2)
for(int i=0; i<n; i=i+2)
dm-bm(int x[i], s1, int x[i+1], s1); }
void main( int ac, char *av[]) {
glutInit(&ac, av);
printf("\n Enter no of verts: \n");
cin >> n;
printf("\n Enter the coordinates of vertex: \n");
for(int i=0; i<n; i++) {
printf("\n x-cord y-cord for %d vertex: ", i+1);
cin >> x[i] >> y[i]; }
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(500, 500);
glutInitWindowPosition(0, 0);
glutCreateWindow("simple");
glutDisplayFunc(display_filled_polygon);
myInit();
glutMainLoop();
}
```

Reva

```
Enter the rotation angle
90
Enter c and m value for line y=mx+c
3
5
```

```
Enter the rotation angle
270
Enter c and m value for line y=mx+c
-2
3
```

Program5: House Rotation

Date :

Experiment No. : 5

Page No : 12

Write a program to create house like figure & perform the following
i) Rotate it about a given fixed point using OpenGL transformation
2) Reflect it about a given line using OpenGL transformation functions

```
#include<GL/glut.h>
float house[11][2]={{100,200},{200,250},{300,200},
{100,200},{100,100},{125,100},{125,150},{225,150}
{300,100},{300,200}};
int angle; float m,c,theta;
void display(){
glClearColor(1,1,1,0);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glMatrixMode(GL_PROJECTION);
gluOrtho2D(-450,450,-450,450);
for(int i=0;i<11;i++)
glVertex2fv(house[i]);
glEnd(); glFlush();
glPushMatrix();
glTranslatef(100,100,0);
glRotatef(angle,0,0,1);
for(int i=0;i<11;i++)
glVertex2fv(house[i]);
glEnd();
glPopMatrix();
glFlush();
void display2(){
glClearColor(1,1,1,0);
```

Reva

Date :

Experiment No. : 5

Page No : 12

```
glmatrixmode(GL_PROJECTION);
  glloadidentity();
  glortho2D(-450, 450, -450, 450);
   glmatrixmode(GL_MODELVIEW);
    glloadidentity();
  float x1=0, x2=500;
    float y1 = m * x1 + c;
     float y2 = m * x2 + c;
     glcolor3f(1, 1, 0);
     glbegin(GL_LINES);
      glvertex2f(x1, y1);
       glvertex2f(x2, y2);
       glend(); glflush();
glpushmatrix();
 gltranslatef(0, c, 0);
  theta = atan(m);
  theta = theta * 180/3.14;
  glrotatef(1, -1, 1);
   glrotatef(-theta, 0, 0, 1);
    for(int i=0; i<1; i++
          glvertex2fv(house[i]);
   glend();
    glpopmatrix(2);
    glflush(); }
void main(){
  glclearcolor(1.0, 1.0, 1.0, 1.0);
    glcolor3f(1.0, 0.0, 0.0);
   glpointsize(2.0);
```

Reva

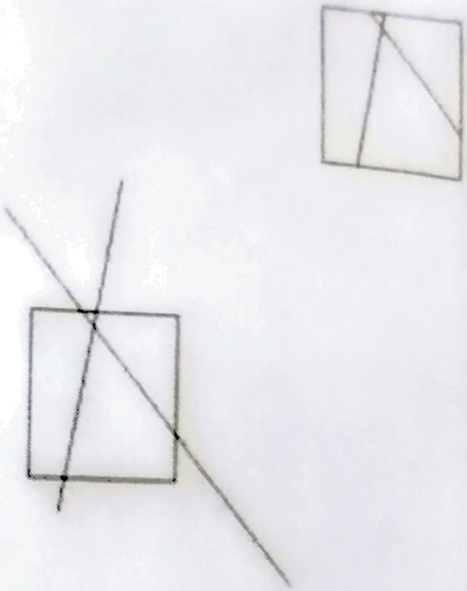
Date :

Experiment No. : 5

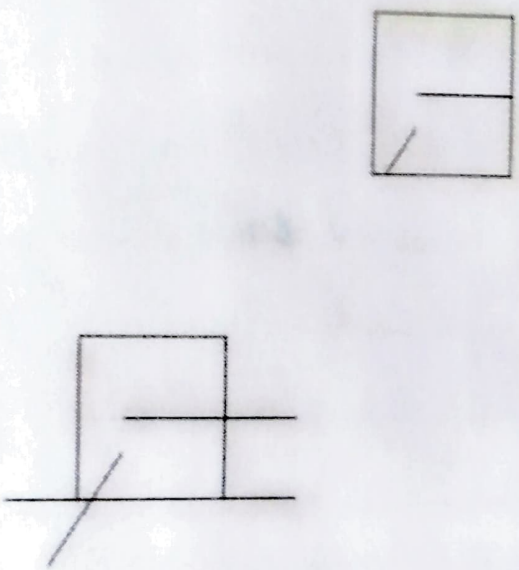
Page No : 14

```
glLoadIdentity();
gluOrtho2D(-450, 450, -450, 450); }
void mouse(int btn, int state, int x, int y) {
if(btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN){
    display();
else display2();
}}
int main(int argc, char** argv)
{ printf("Enter the rotation angle\n");
cin>> angle;
printf("Enter C & m such that y = mx+C\n");
cin>>C>>m;
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(900,900);
glutInitWindowPosition(100,100);
glutCreateWindow("Hand Rotation");
glutDisplayFunc(display);
    glutMouseFunc(mouse);
    myinit();
glutMainLoop();
}
```

```
Enter window coordinates (xmin ymin xmax ymax):
100 100 200 200
Enter viewport coordinates (xvmin yvmin xvmax yvmax) :
300 300 400 400
Enter no. of lines:
2
Enter line endpoints (x1 y1 x2 y2):
80 260 200 40
Enter line endpoints (x1 y1 x2 y2):
160 280 120 80
```



```
Enter window coordinates (xmin ymin xmax ymax):
100 100 200 200
Enter viewport coordinates (xvmin yvmin xvmax yvmax) :
300 300 400 400
Enter no. of lines:
2
Enter line endpoints (x1 y1 x2 y2):
50 100 250 100
Enter line endpoints (x1 y1 x2 y2):
80 60 130 130
Enter line endpoints (x1 y1 x2 y2):
130 150 250 150
```



Date :

Experiment No. : 8

Page No : 15

Write a program to implement the cohen-sutherland line clipping algorithm. Make provision to specify the input for multiple lines, window for clipping & input for defining clipping window.

```
#include<gl/glut.h>
const int RIGHT=4;
const int LEFT=8;
const int TOP=1;
const int BOTTOM=2;
int computecode(double x, double y)
{ int code=0;
  if(y>ymax)
      code|=TOP;
  else if(y<ymin)
      code|=BOTTOM;
  if(x>xmax)
      code|=RIGHT;
  return code; }
void cohensutherland(double x0, double y0, double x1, double y1)
{ int outcode0, outcode1, outcodeout;
  bool accept=false, done=false;
  outcode0=computecode(x0,y0);
  outcode1=computecode(x1,y1);
do {
  if(!(outcode0|outcode1))
  { accept=true;
    done=true; }
else if(outcode0&outcode1)
```

Reva

Date :

Experiment No. : 8

Page No : 16

```
y = y0 + (y1 - y0) * (xm - x0) / (x1 - x0);
}
else {
  x1 = x;
  y1 = y;
   outcode1 = computecode(x1, y1);
}} while(!done);
if (accept) {
double sx = (xvmax - xvmin) / ((xmax - xmin));
 double sy = (yvmax - yvmin) / (ymax - ymin);
 double vx1 = xvmin + (x1 - xmin) * sx;
 double vy1 = ymin + (y1 - ymin) * sy;
   glColor3f(1, 0, 0);
    glVertex2f(xvmin, yvmin);
      glVertex2f(vmax, ymin);
     glEnd();
     glColor3f(0, 0, 1)
     glBegin(GL_LINES)
       glVertex2D(vx0, vy0))
         glEnd(); }
void display()
{  glClear(GL_COLOR_BUFFER_BIT);
 glColor3f (0, 0, 1);
     glBegin(GL_LINE_LOOP);
glVertex2f(xmin, ymin);
    glVertex2f(xmin, ymax);
for (int i = 0; i < n; i++) {
 glBegin(GL_LINES);
```

Reva

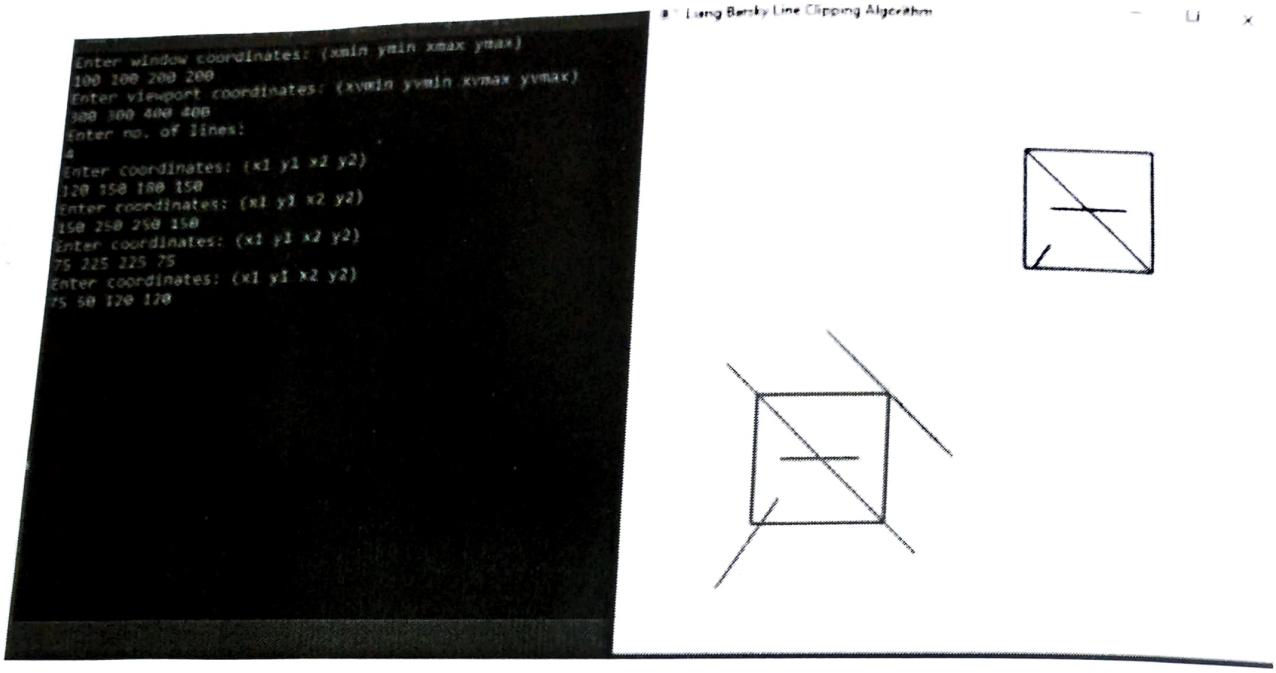
Date :

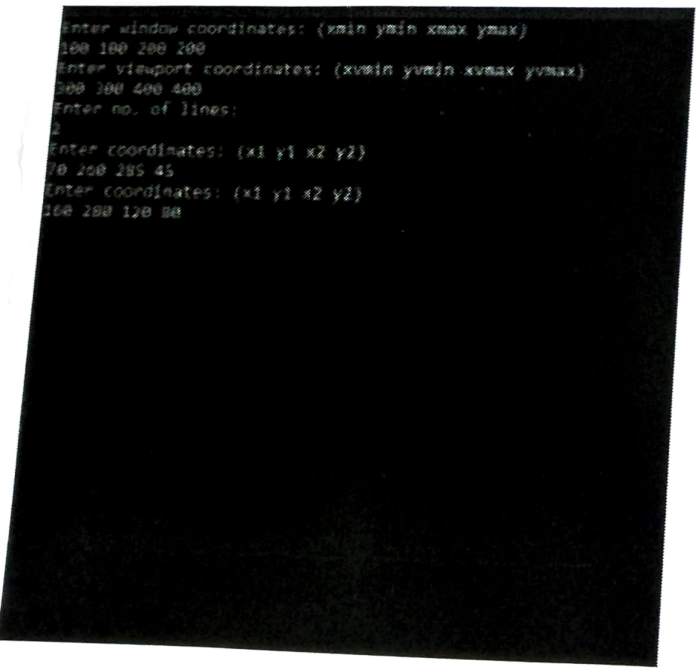
Experiment No. : 8

Page No : 17

```
glutMouseFunc(2D(ls[i].x1, ls[i].y1);
    glutMotionFunc(2D(ls[i].x2, ls[i].y2);
        glutIdle();
} for(int i=0; i<n; i++)
    construction(ls[i].x1, ls[i].y1, ls[i].x2, ls[i].y2);
    glFlush(); }
    void main(int argc, char *argv)
{ printf("enter the initial coordinates");
scanf("%f%f%f%f", &x1, &y1, &x2, &y2);
printf("enter the terminal coordinates");
scanf("%f%f%f%f", &x1, &y1, &x2, &y2);
printf("enter the no of lines:\n");
    scanf("%d", &n);
for(i=0; i<n; i++)
    scanf("%f%f%f%f", &ls[i].x1, &ls[i].y1,
    &ls[i].x2, &ls[i].y2); }
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(500, 500);
glutInitWindowPosition(0, 0);
    glutCreateWindow("clip");
    myInit();
glutDisplayFunc(display);
    glutMainLoop();
}
```

Reva





Program7: LiangBarsky Line Clipping

Date :.....

Experiment No. :.....7..... Page No :.....12.....

Write a program to implement the Liang Barsky line clipping algorithm. Make provisions to specify the input line, window for clipping and viewport for displaying the clipped image.

```
#include<GL/glut.h>
double xmin, ymin, xmax, ymax;
double x1, y1, x2, y2;
int n;
struct line_segment {
    int x1;
    int y1;
    int x2;
    int y2;
};
struct line_segment ls[10];
int clip(double h, double q, double *u1, double *u2)
{
  double r;
if (h) r = q/p;
  if (h<0.0)
  { if (r > *u1) *u1 = r;
  if (r > *u2) return(false);
} else
if (h>0.0) {
 if (r<*u2) *u2 = r;
   if (r<*u1) return(false);
}
```

Date :.....

Experiment No. :.....7.....

Page No :.....19.....

```
else if(h==0.0)
{   if(d<0.0) return(false);
    } return(true);
return bool LiangBarskyClip(double x0, double y0, double x1, double y1)
{ double dx=x1-x0, dy=y1-y0, u1=0.0, u2=1.0;
glColor3f(1.0,0.0,0.0);
glBegin(GL_LINE_LOOP);
glVertex2f(xwmin, ywmin);
  glVertex2f(xwmax, ywmax);
  glEnd();
if(clipTest(-dx,x0-xwmin,&u1,&u2))
if(clipTest(dx,xwmax-x0,&u1,&u2))
if(clipTest(-dy,y0-ywmin,&u1,&u2))
if(clipTest(dy,ywmax-y0,&u1,&u2))
{ if(u2<1.0)
    x1=x0+u2*dx;
      y1=y0+u2*dy; }
glColor3f(0.0,0.0,1.0);
  glBegin(GL_LINES);
  glVertex2d(vx0,vy0);
  glVertex2d(vx1,vy1);
    glEnd(); }
void display()
{ glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0,0.0,0.0);
  for(int i=0;i<n;i++){
  glBegin(GL_LINES);
    glVertex2d(ds[i].x1,ds[i].y1);
    glEnd();
```

Reva

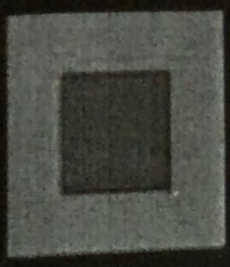
Date :

Experiment No. : 7 Page No : 20

```
for(int i=0; i<n; i++)
using Bresenhams line algorithm(ls[i].x1, ls[i].y1, ls[i].x2, ls[i]
    glFlush(); }
    int main(int argc, char **argv)
{ glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(500, 500);
    glutInitWindowPosition(0,0);
printf("enter window coordinates: ");
  scanf("%d %d %d %d", &xmin, &ymin, &xmax, &ymax);
printf("enter view port coordinates: ");
scanf("%d %d %d %d", &vxmin, &vymin, &vxmax, &vymax);
  printf("enter the no of lines \n");
    scanf("%d", &n);
    for(int i=0; i<n; i++)
    { printf("enter coordinates: ");
scanf("%d %d %d %d", &ls[i].x1, &ls[i].y1, &ls[i].x2, &ls[i].y2);
}
glutCreateWindow("Liang Barsky line clipping algorithm");
    glutDisplayFunc(display);
      myinit();
  glutMainLoop();
}
```

Reva

```
Enter no. of
4
Polygon Vertex:
100 100
Polygon Vertex:
200 100
Polygon Vertex:
200 200
Polygon Vertex:
100 200
Enter no. of vertices of clipping window:
4
Clip Vertex:
50 50
Clip Vertex:
250 50
Clip Vertex:
250 250
Clip Vertex:
50 250
```



```
Enter no. of vertices:
3
Polygon Vertex:
150 200
Polygon Vertex:
100 100
Polygon Vertex:
200 100
Enter no. of vertices of clipping window:
4
Clip Vertex:
110 100
Clip Vertex:
190 100
Clip Vertex:
190 180
Clip Vertex:
110 180
```

Polygon Clipping!

```
Enter no. of vertices:
5
Polygon Vertex:
300 100
Polygon Vertex:
100 200
Polygon Vertex:
100 100
Polygon Vertex:
400 100
Polygon Vertex:
400 200
Enter no. of vertices of clipping window:
4
Clip Vertex:
80 120
Clip Vertex:
420 120
Clip Vertex:
420 250
Clip Vertex:
80 250
```

Polygon Clipping!

Program8: Polygon Clipping

Date :

Experiment No. : 8

Page No : 21

Write a program to implement the Sutherland Hodgman polygon clipping algorithm. Make provision to specify the input polygon & window for clipping.

```
#include<stdio.h>
#include<GL/glut.h>
int poly_sz, poly_points[20][2], org_poly_sz,
  org_poly_points[20][2], clip_sz, clip_points[20][2];
const int MAX_POINTS = 20;
void display(int p[][2], int n)
{
  glClear(GL_COLOR_BUFFER_BIT);
  for(int i=0; i<n; i++)
  glVertex2f(p[i][0], p[i][1]);
  glEnd(); }
int x_intersect(int x1, int y1, int x2, int y2,
  int x3, int y3, int x4, int y4)
{ int num = (x1*y2 - y1*x2) * (x3-x4) - (x1-x2)
  * (x3*y4 - y3*x4);
int den = (x1-x2) * (y3-y4) - (y1-y2) * (x3-x4);
  return num/den; }
void clip( int poly_in[][2], int &poly_sz,
    int x1, int y1, int x2, int y2)
{
  int new_points[MAX_POINTS][2], new_poly_size=0;
  for(int i=0; i<poly_sz; i++)
  { int k = (i+1) % poly_sz;
  int ix = poly_points[i][0], iy = poly_points[i][1],
  int kx = poly_points[k][0], ky = poly_points[k][1];
```

Rev

Date :

Experiment No. : 8

Page No : 22

```
int k_pos = (x2-x1) * (ky-y1) - (y2-y1) * (kx-x1);
if (i_pos >= 0 && k_pos >= 0)
 { new_points[new_poly_size][0] = kx;
  new_points[new_poly_size][1] = ky;
   new_poly_size++; }
else if ( i_pos >= 0 && k_pos < 0)
 {
  new_points [new_poly_size] [0] = x_intersect(x1, y1, x2,
y2, ix, iy, kx, ky);
 new_points [new_poly_size] [1] = y_intersect (x1, y1, x2, y2,
   ix, iy, kx, ky);
  new_poly_size++; }
  else
   {
       }  }
  void display()
{ init();
 glColor3f (1.0f, 0.0f, 0.0f);
  drawpoly(clipper_points, clipper_size);
 glColor3f (0.0f, 1.0f, 0.0f);
   drawpoly(org_poly_points, org_poly_size);
 for (int i=0; i< clipper_size; i++) {
  int k = (i+1) % clipper_size;
  clip (poly_points, poly_size, clipper_points[i][0],
   clipper_points[i][1], clipper_points [k][0],
    clipper_points [k] [1]); }
 glColor3f (0.0f, 0.0f, 1.0f);
  drawpoly (poly_points, poly_size);
   glFlush();
  }
```

Date :

Experiment No. : 8

Page No : 27

```
}
int main(int argc, char *argv[])
{
printf("enter no of vertex: \n");
scanf("%d", &poly_size);
argc_poly_sz = poly_size;
for(int i=0; i<poly_sz; i++){
printf("polygon vertex: \n");
scanf("%d %d", &poly_bounds[i][0], &poly_bounds[i][1]);
org_poly_bounds[i][0] = poly_bounds[i][0];
org_poly_bounds[i][1] = poly_bounds[i][1]; }
printf("\n enter no of vertex of clipping window:");
scanf("%d", &clipper_size);
for(int i=0; i<clip_size; i++)
{
printf("\n clip vertex: \n");
scanf("%d %d", &clipper_bounds[i][0], &clipper_
bounds[i][1]); }
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(400, 400);
glutInitWindowPosition(100, 100);
glutCreateWindow("Polygon clipping");
glutDisplayFunc(display);
glutMainLoop();
return 0;
}
```

Reva

Date :

Experiment No. : 9

Page No : 24

Write a prgm to model a car like figure using display lists & move a car from one end of the screen to other end. Using mouse click to move the wheel with mouse.

```
#include<glut.h>
#define CAR 1      #define WHEEL 2
float s=1;
void carlist() {
glNewList(CAR,GL_COMPILE);
 glColor3f(1,1,1);
glBegin(GL_POLYGON);
 glVertex3f(0,25,0);
  glVertex3f(90,25,0);
 glVertex3f(0,155,0);
 glEnd();
   glEndList(); }
void wheellist() {
glNewList(WHEEL,GL_COMPILE_AND_EXECUTE);
  glColor3f(0,1,1);
 glutSolidSphere(10,25,25); glEndList(); }
void movecar (float s) {
 glTranslatef(S,0.0,0.0);
 glCallList(CAR);
  glPushMatrix();
 glTranslatef(25,25,0.0);
  glCallList(wheel);
  glTranslatef(75,25,00);
    glPopMatrix();
}
```

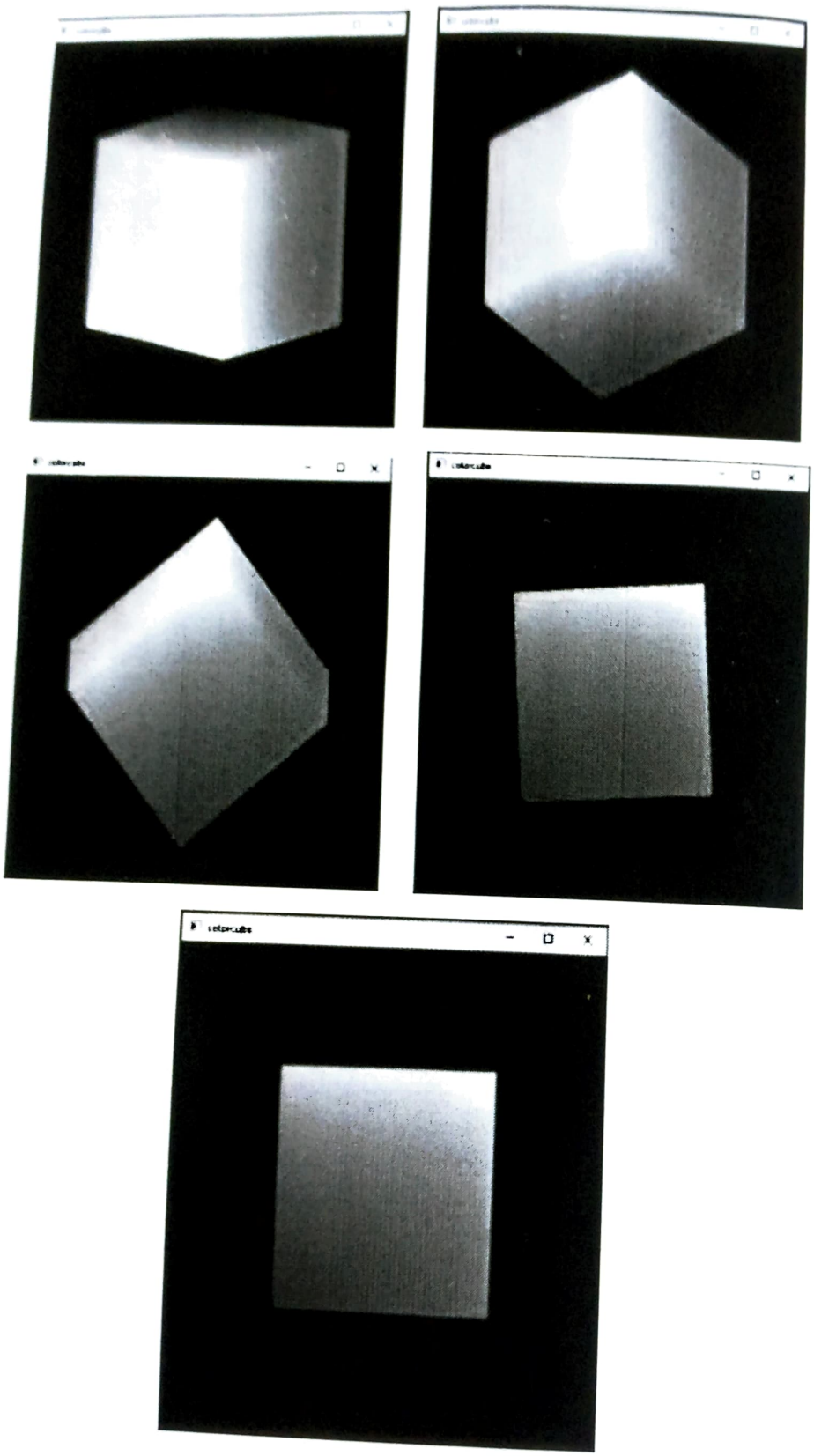
Date :

Experiment No. : 1

Page No : 25

```
void mouse(int btn, int state, int x, int y)
  if (btn == GLUT_LEFT_BUTTON && state == GLUT_
  DOWN) {
   st=25;
    Mydisplay(); }
else if (btn==GLUT_RIGHT_BUTTON && state==GLUT_DOWN) {
      st=2;
       Mydisplay(); }
  int main(int argc, char *argv[]) {
  glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(600, 500);
 glutInitWindowPosition(100, 100);
   glutCreateWindow("Car");
   myinit();
 glutDisplayFunc(MyDisplay);
  glutMouseFunc(mouse);
  glutKeyboardFunc(Mykeyboard);
  glutMainLoop(); }
void display(){
 glClear(GL_COLOR_BUFFER_BIT);
    loadIdentity();
     move(st);
      checkst();
}
```

Reva



Program10: Colour Cube

Date :

Experiment No. : 10

Page No : 26

Create a program to create colorful cube and rotate it using OpenGL transformations.

```
#include <GL/glut.h>
GLfloat vertices[] = {-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, 1.0, 1.0, -1.0,
-1.0, 1.0, 1.0, 1.0, 1.0, -1.0, 1.0, -1.0, 1.0, 1.0, 1.0};
GLfloat normals[] = {-1.0, -1.0, -1.0, 1.0, 1.0, -1.0, 1.0, -1.0,
1.0, -1.0, 1.0, 1.0, -1.0, 1.0, 1.0};
GLfloat colors[] = {0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0,
1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0};
GLubyte cubeIndices[] = {0, 3, 2, 1, 2, 3, 7, 6, 0, 4, 7, 3, 1, 2, 6, 5, 4,
void display(void)
{ glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();
  glRotatef(theta[0], 1.0, 0.0, 0.0);
  glRotatef(theta[1], 0.0, 1.0, 0.0);
  glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE, cubeIndices);
  glBegin(GL_LINES);
  glVertex3f(1.0, 1.0, 1.0); glEnd();
  glFlush(); }
void spinCube()
{ delay(0.01);
  theta[axis] += 2.0;
  if (theta[axis] > 360.0) theta[axis] -= 360.0;
  glutPostRedisplay(); }
void mouse(int btn, int state, int x, int y)
{ if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN) axis = 0;
if (btn == GLUT_RIGHT_BUTTON && state == GLUT
}
```

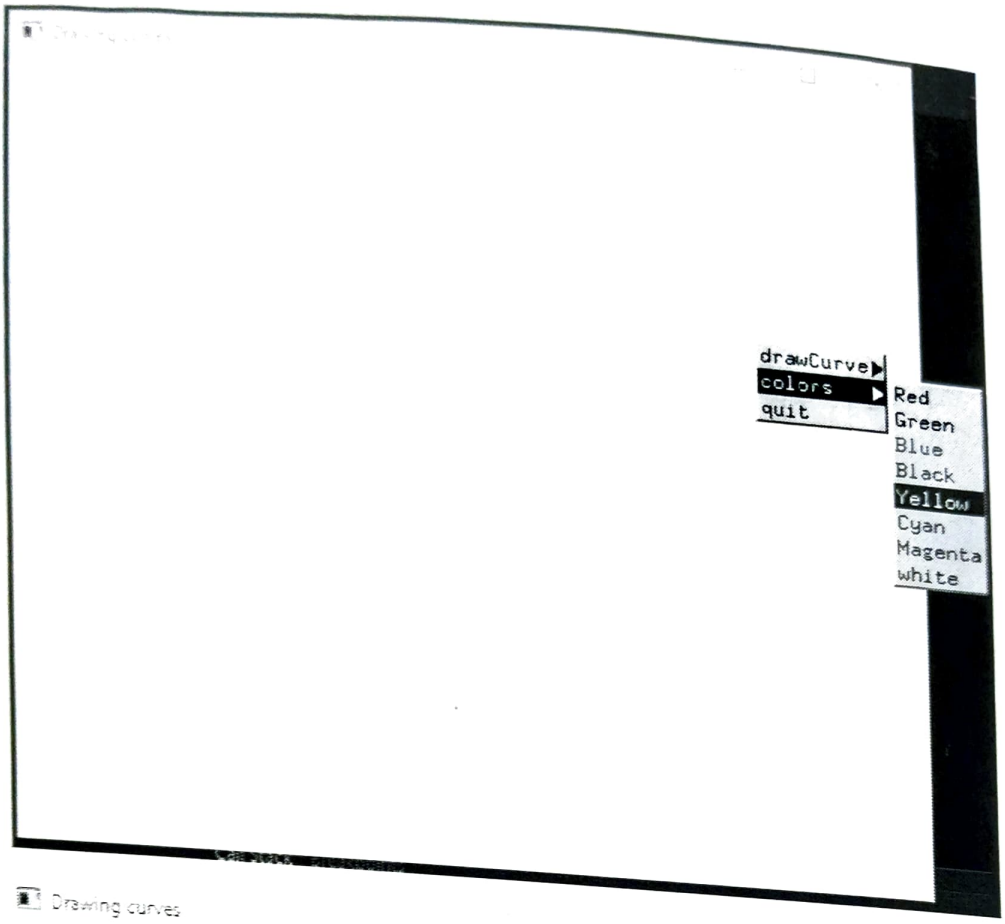
Date :

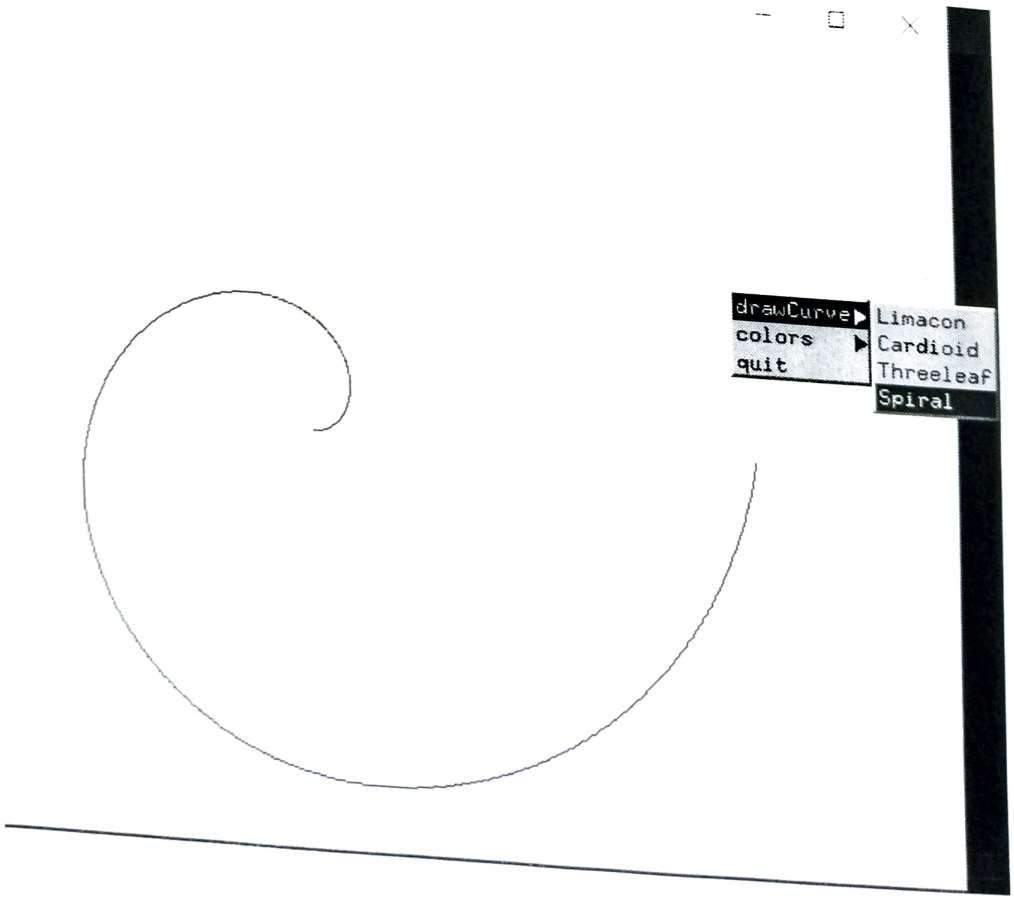
Experiment No. : 10

Page No : 27

```
void myreshape(int w, int h)
{ glViewport(0,0,w,h);
 if (w<=h)
    glOrtho(-2.0,2.0,-2.0*(GLfloat)h/(GLfloat)w,2.0*(GLfloat)h/(GLfloat)w,-10.0,10.0);
else
glOrtho(-2.0*(GLfloat)w/(GLfloat)h,2.0*(GLfloat)w/(GLfloat)h,-2.0,2.0);
}
void main(int argc, char **argv)
{glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
 glutInitWindowPosition(100,100);
 glutInitWindowSize(500,500);
 glutCreateWindow("window1");
 glutReshapeFunc(myreshape);
 glutDisplayFunc(display);
 glutIdleFunc(spinCube);
 glutMouseFunc(mouse);
 glEnable(GL_DEPTH_TEST);
 glEnableClientState(GL_COLOR_ARRAY);
 glEnableClientState(GL_NORMAL_ARRAY);
 glEnableClientState(GL_VERTEX_ARRAY);
 glVertexPointer(3,GL_FLOAT,0,vertices);
 glColorPointer(3,GL_FLOAT,0,colors);
 glNormalPointer(GL_FLOAT,0,normals);
    glClearColor(1.0,1.0,1.0,1.0);
    glutMainLoop();
}
```

Reva





Program11: Drawing Curves

Date :

Experiment No. : 4 Page No : 58

~~#~~ Write a menu with 3 entries named lines, colors & quit. The lines entry has a sub menu which has 4 entries namely limacon, cardioid, three leaf & spiral. The colors menu has sub menu with all 8 colors of RGB color model. Write a program to create the above hierarchical menu & attach appropriate servings for each menu entry with mouse buttons.

```
#include <GL/glut.h>
typedef enum{limacon=1, cardioid=2, threeleaf=3,
    spiral=4} curveName;
int w=600, h=500;
void lineSegment(scrPt p1, scrPt p2){
  glBegin(GL_LINES);
  glVertex2i(p1.x, p1.y);
   glVertex2i(p2.x, p2.y);
    glEnd();
     glFlush(); }
void drawCurve(int curveNum){
  const double twoPi=6.283185;
  const int a=175, b=60;
  int x0=200, y0=250;
  scrPt curvePt[2];
  glColor3f(red, green, blue);
   curvePt[0].x = x0;
  curvePt[0].y = y0;
glClear(GL_COLOR_BUFFER_BIT);
 switch(curveNum){
```

Reva

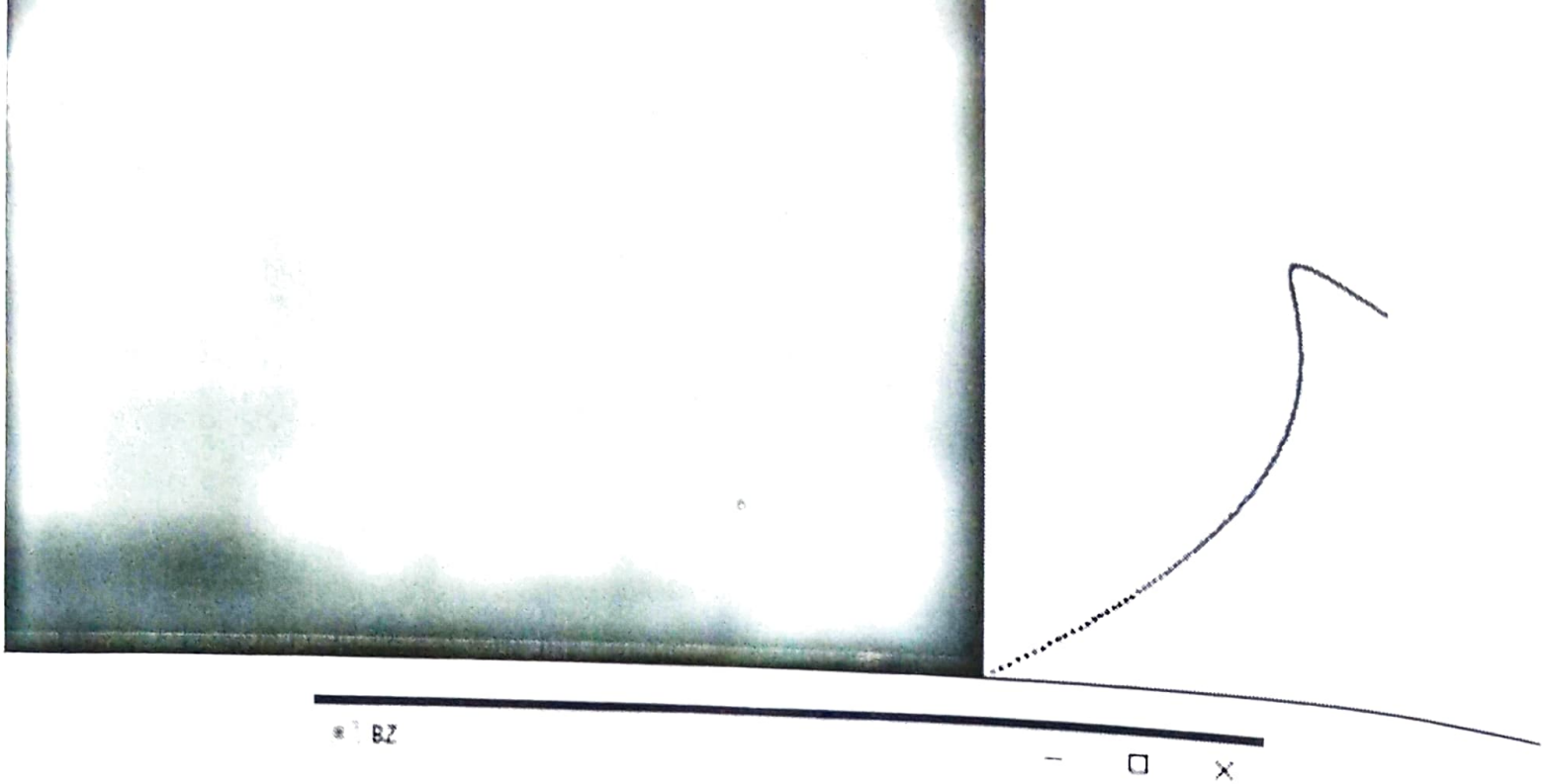
```
case Limacon: currpt[0].x += a+b; break;
case cardioid: currpt[0].x += a+a; break;
case threeleaf: currpt[0].x += a; break;
case spiral: break;
default: break; }
curvept[0].x = curpt[1].x;
currpt[0].y = currpt[1].y;
theta += dtheta; }}
void colormenu(int id){
switch(id){
case 0:
break;
case 1:
red=0;
green=0;
blue=1; break;
case 2: red=0; green=1; blue=0; break;
case 3: red=0; green=1; blue=1; break;
case 4: red=1; green=0; blue=0; break;
case 5: red=1; green=0; blue=1; break;
case 6: red=1; green=1; blue=0; break;
case 7: red=1; green=1; blue=1; break;
default: Break;

draw curve(curve); }
void main_menu(int id) {
switch(id){
case 3: exit(0);
```

Date :

Experiment No. : 4 Page No : 30

```
default: break; }}
void main(int argc, char **argv){
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(w, h);
glutInitWindowPosition(100, 100);
glutCreateWindow("Drawing Curve");
int curve = glutCreateMenu(drawCurve);
glutAddMenuEntry("Linear", 1);
glutAddMenuEntry("Cardinal", 2);
glutAddMenuEntry("Bezier", 3);
glutAddMenuEntry("Bspline", 4);
int color = glutCreateMenu(colorMenu);
glutAddMenuEntry("Red", 4);
glutAddMenuEntry("Green", 2);
glutAddMenuEntry("Blue", 1);
glutAddMenuEntry("Black", 0);
glutAddMenuEntry("Yellow", 6);
glutAddMenuEntry("Cyan", 3);
glutAddMenuEntry("Magenta", 5);
glutAddMenuEntry("White", 7);
glutAttachMenu(GLUT_LEFT_BUTTON);
glutCreateMenu(main_menu);
glutAddSubMenu("drawing", curve);
glutAddMenuEntry("Quit", 3);
// glutAttachMenu(GLUT_LEFT_BUTTON);
glutMainLoop();
}
```

Reva



Program12: BZ

Date :

Experiment No. : 12

Page No : 31

Write a program to construct a Bezier curve. Control points are entered through keyboard/ mouse.

```
#include<gl.glut.h>
float x[4], y[4], xc[4], yc[4];
    int flag=0;
void myInit() {
glClearColor(1.1,1.1,1.1);
    glColor3f(1.1,1.1);
    glPointSize(5);
      gluOrtho2D(0.0,500.0,0.500.0);
}

void drawPixel(float x, float y) {
      glBegin(GL_POINTS);
        glVertex2f(x,y);
          glEnd(); }
void display() {
 glClear(GL_COLOR_BUFFER_BIT);
    int i;
    double t;
for(t=0;t<1;t=t+0.005) {
  double xt = pow(1-t,3)*x[0]+3*t*pow(1-t,2)*x[1]+3*t*t*pow(1-t,1)*x[2]+pow(t,3)*x[3];
double yt = pow(1-t,3)*y[0]+3*t*pow(1-t,2)*y[1]+3*t*t*pow(1-t,1)*y[2]+pow(t,3)*y[3];
      glVertex2f(xt,yt); }
 glColor3f(1.0,1.0);
 for(i=0;i<4;i++)
     glVertex2f(x[i],y[i]);
       glEnd();
```

Reva

Date :

Experiment No. : 12

Page No : 32

```
void mouse(int btn, int state, int x, int y)
{ if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
{ x[flag]=x;
  yc[flag]=500-y;
cout<<"x:"<<x<<"y:"<<500-y;
  glPointSize(3);
  glColor3f(1,1,0);
   glBegin(GL_POINTS);
   glVertex2i(x,500-y);
    glEnd();
     flag++; }
if(flag>=4&&btn==GLUT_LEFT_BUTTON)
{ glColor3f(0,0,1);
      display();
    flag=0;
  }}
int main(int argc, char *argv[])
{ glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
 glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
 glutCreateWindow("B2");
 glutDisplayFunc(display);
 myInit();
 glutMouseFunc();
}
```