PANJAB UNIVERSITY

COMPUTER GRAPHICS

Submitted to:

Dr. RAVINDER KUMAR SINGLA

(Dept. of Computer science and Application)

Submitted by:

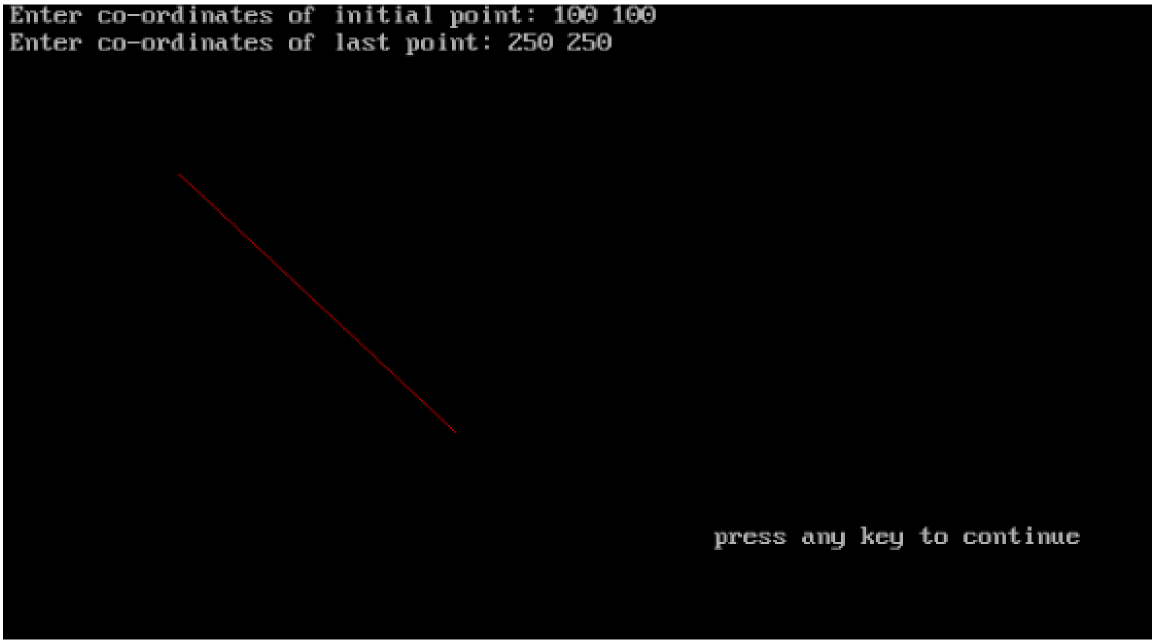
Naveen kumar

ROLL NO. 12

Computer Graphics Lab Programs (Dr R K Singla)

1. DDA Line Drawing Algorithm

```
#include<stdio.h>

#include<iostream.h>

#include<graphics.h>

#include<math.h>

#include<dos.h>

#include<conio.h>

int abs (int n){

return ( (n > 0) ? n : ( n * (-1)));

}

int round(float z)

{

int y;

y = z > 0.0 ? int(z + 0.5) : int(z - 0.5);

return y;

}

void DDA(int X0, int Y0, int X1, int Y1){

int dx = X1 - X0;

int dy = Y1 - Y0;


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


float Xinc = dx / (float) steps;

float Yinc = dy / (float) steps;


float X = X0;

float Y = Y0;

for (int i = 0; i <= steps; i++)

{

  putpixel (round(X),round(Y),RED); // put pixel at (X,Y)

  X += Xinc; // increment in x at each step

  Y += Yinc; // increment in y at each step
  delay(50); // for visualization of line-
// generation step by step
}
}

int main()
{
int gd = DETECT, gm;
initgraph (&gd, &gm, "C:\\turboc3\\bgi");
int X0, Y0, X1, Y1;
cout<<" Enter co-ordinates of initial point: ";
cin>>X0>>Y0;
cout<<" Enter co-ordinates of last point: ";
cin>>X1>>Y1;
line(0, 0, 639, 0); //x-axis
line(0, 0, 0, 479); //y-axis
DDA(X0, Y0, X1, Y1);
gotoxy(50, 20); cout<<"press any key to continue";
getchar();
return 0;
}
```

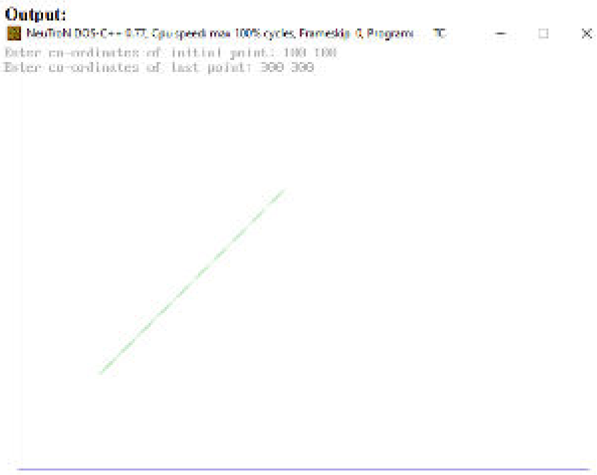


```
Enter co-ordinates of initial point: 100 100
Enter co-ordinates of last point: 250 250





                                                 press any key to continue
```

2. DDA Line Drawing (with origin on lower-left corner)

```
#include<iostream.h>

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

struct point {

  float x; float y;

};

void main() {

  clrscr();

  int gd = DETECT, gm;

  initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");

  point beg, end, Line;

  float slope;

  int j, i;

  cout<<"Enter co-ordinates of initial point: ";

  cin>>beg.x>>beg.y;

  cout<<"Enter co-ordinates of last point: ";

  cin>>end.x>>end.y;

  Line = beg;

  ////////////////////////////////////////////////

  setbkcolor(WHITE); setcolor(BLUE);

  line(20, 25, 20, 454); //x-axis

  line(15, 449, 619, 449); //y-axis

  ////////////////////////////////////////////////

  slope = fabs((end.y-beg.y)/(end.x-beg.x));

  if(slope <= 1) {

  for(i=0; i<int(end.x-beg.x); i++) {

  putpixel(Line.x, 449-Line.y, GREEN);

  Line.x += 1;

  Line.y += slope;

  }

  } else {

  for(i=0; i<int(end.y-beg.y); i++) {
```

```
  putpixel(Line.x, 449-Line.y, GREEN);

  Line.x += 1/slope;

  Line.y += 1;

  }

  }

getch(); closegraph();

}
```



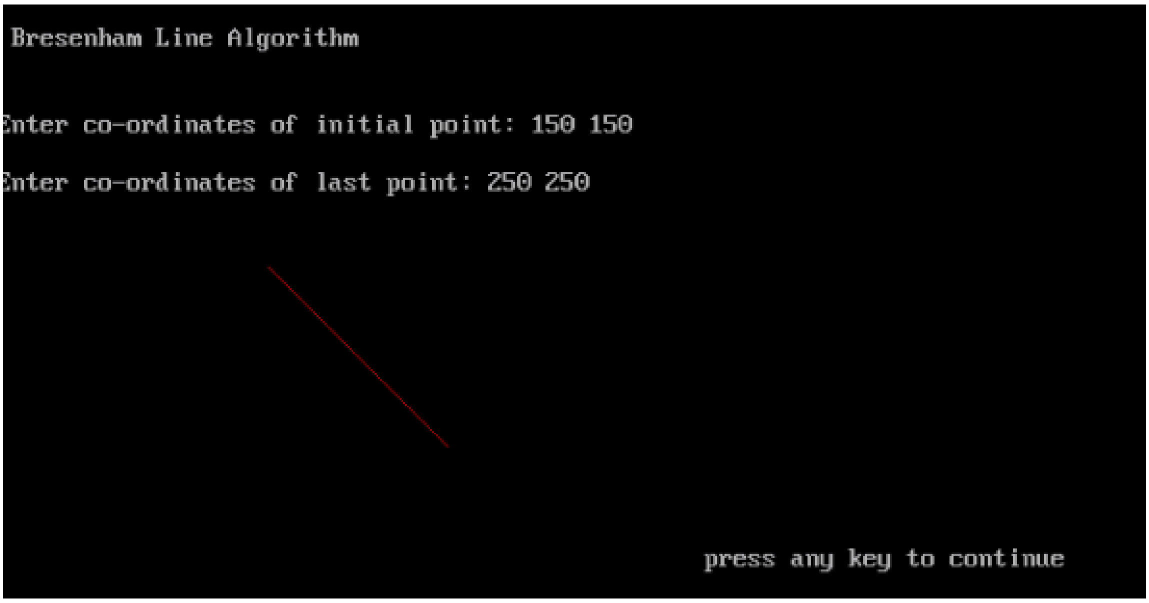
Output:

NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter co-ordinates of initial point: 100 100
Enter co-ordinates of last point: 300 300
```

3. Bresenham’s Line Drawing Algorithm

```
#include<stdio.h>

#include<iostream.h>

#include<graphics.h>

#include<dos.h>

#include<conio.h>

//Bresenham's Function for line generation

void drawline(int x0, int y0, int x1, int y1) {


int dx, dy, p, x, y;

dx = x1 - x0;

dy = y1 - y0;

x = x0;

y = y0;

p=2 * dy - dx;

while(x < x1) {

if(p >= 0) {

putpixel(x, y, RED);
y = y + 1;

p = p + 2 * dy - 2 * dx;

}

else {

putpixel(x, y, RED);

p = p + 2 * dy;

}

x = x + 1;

delay(50);// for step-by-step visualization

}

}

// Driver program

int main() {

int gd = DETECT, gm;

// Initialize graphics function

initgraph (&gd, &gm, "C:\\turboc3\\bgi");

int X0, Y0, X1, Y1;

cout<<"\n Bresenham Line Algorithm\n\n";

cout<<" \nEnter co-ordinates of initial point: ";

cin>>X0>>Y0;

cout<<" \nEnter co-ordinates of last point: ";

cin>>X1>>Y1;

line(0, 0, 639, 0); //x-axis

line(0, 0, 0, 479); //y-axis

drawline(X0, Y0, X1, Y1);

gotoxy(50, 20); cout<<"press any key to continue";

getchar();

return 0;

}
```

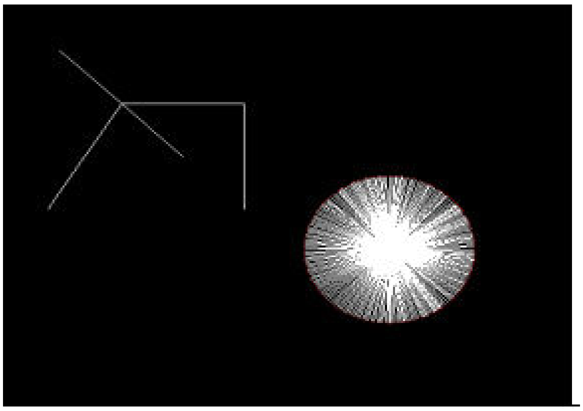


```
Bresenham Line Algorithm

Enter co-ordinates of initial point: 150 150

Enter co-ordinates of last point: 250 250

                                        press any key to continue
```

4. Bresenham's Line Drawing Generalized Algorithm (All slopes)

```
#include<stdio.h>

#include<graphics.h>

#include<math.h>

#include<dos.h>

#include<conio.h>

//int sign(x) {

// if(x>0) return 1; else if(x<0) return -1; else return 0;

//}

int sign(int x) {

return (x > 0) ? (1) : ( (x < 0) ? (-1) : (0));

}

void bres(int x1,int y1, int x2, int y2){

int x, y, dx, dy, swap, temp, s1, s2, p, i;

x = x1; y = y1;

dx = abs(x2-x1); dy = abs(y2-y1);

s1 = sign(x2-x1); s2 = sign(y2-y1);

swap = 0;

putpixel(x1, y1, RED);

if(dy > dx){

temp = dx; dx = dy; dy = temp;

swap=1;

}
p=2 * dy - dx;

for(i = 0; i < dx; i++){

putpixel(x, y, getcolor());

while(p >= 0){

p = p – 2 * dx;

if(swap) x += s1; else y += s2;

}

p = p + 2 * dy;

if(swap) y += s2; else x += s1;

}

putpixel(x2, y2, RED);

}

//driver

void main(){

int gd=DETECT,gm;

int x1, y1, x2, y2, theta = 0, midx, midy;

initgraph(&gd,&gm,"\\turboc3\\BGI\\");

midx=getmaxx()/2;

midy=getmaxy()/2;

// testing of bresenham's bres() line function above

bres(150,150, 50, 50);

bres(100, 100, 200, 100);

bres(200, 100, 200, 200);

bres(100, 100, 40, 200);

getchar();

// drawing radial lines for a circle.

while(!kbhit()){

bres(midx,midy,midx+70*sin(3.14f*theta/180),midy+70*cos(3.14f*theta/180));

delay(50);

cleardevice();

theta+=2;

}

}
```

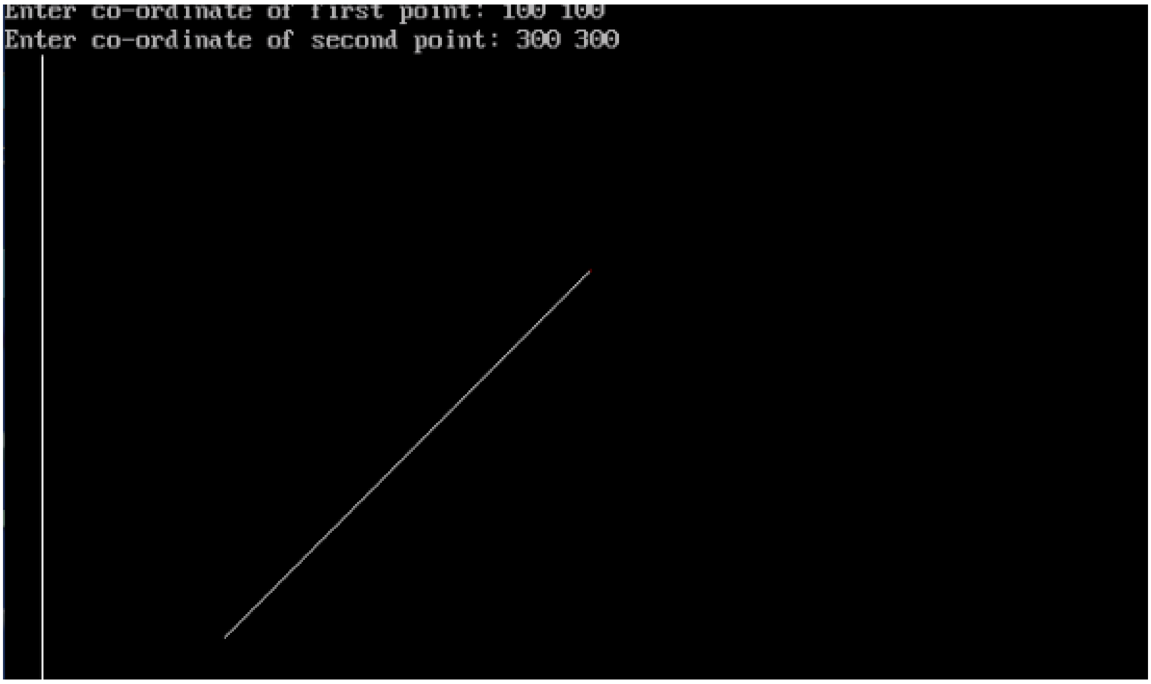
Output:



5. Bresenham's Line Drawing Generalized Algorithm (All slopes) with origin on lower-left corner

```
#include<stdio.h>
#include<iostream.h>
#include<graphics.h>
#include<math.h>
#include<dos.h>
#include<conio.h>
int sign(int x) {
return (x > 0) ? (1) : ( (x < 0) ? (-1) : (0));
}
void bres(int x1,int y1, int x2, int y2){
int x,y,dx,dy,swap,temp,s1,s2,p,i;
x=x1; y=y1;
dx=abs(x2-x1);
dy=abs(y2-y1);
s1=sign(x2-x1);
s2=sign(y2-y1);
swap=0;
putpixel(20+x1,449-y1,RED);
if(dy>dx){
temp=dx; dx=dy; dy=temp;
swap=1;
}
p=2*dy-dx;
for(i=0;i<dx;i++){
```

```
putpixel(20+x,449-y,getcolor());

while(p>=0){

p=p-2*dx;

if(swap) x+=s1; else y+=s2;

}

p=p+2*dy;

if(swap) y+=s2; else x+=s1;

}

putpixel(20+x2,449-y2,RED);

}

//driver function

void main(){

int gd=DETECT,gm;

int x1,y1,x2,y2;

initgraph(&gd,&gm,"\\turboc3\\BGI\\");

line(20, 25, 20, 454); //x-axis

line(15, 449, 619, 449); //y-axis

cout<<"Enter co-ordinate of first point: ";

cin>>x1>>y1;

cout<<"Enter co-ordinate of second point: ";

cin>>x2>>y2;

bres(x1,y1,x2,y2);

getch();

}
```



```
Enter co-ordinate of first point: 100 100
Enter co-ordinate of second point: 300 300
```

6.program uses 3D Animation to display 3 wire-frame rotating Cubes.

```
#include <graphics.h>

#include <math.h>

#include <stdio.h>

/* define global variables */

int rho=11000,d=12300;

/* define the 8 corners of the cube */

int point[8][3]=

     {

       {50,50,50},      {-50,50,50},    {-50,50,-50},    {50,50,-50},

       {50,-50,-50}, {50,-50,50},    {-50,-50,50}, {-50,-50,-50}

     };

int sx[8],sy[8],rot=1;

int originx=320,originy=150, page=0;

/* define the end points of the 12 edges of thr cube */

int edge[12][2]=

    {

        {0,1},    {1,2},    {2,3},    {3,0},

        {3,4},    {4,5},    {5,6},    {6,7},

        {7,4},    {7,2},    {6,1},    {5,0}

    };

char c;

double theta=3.141/4,phi=0;
```

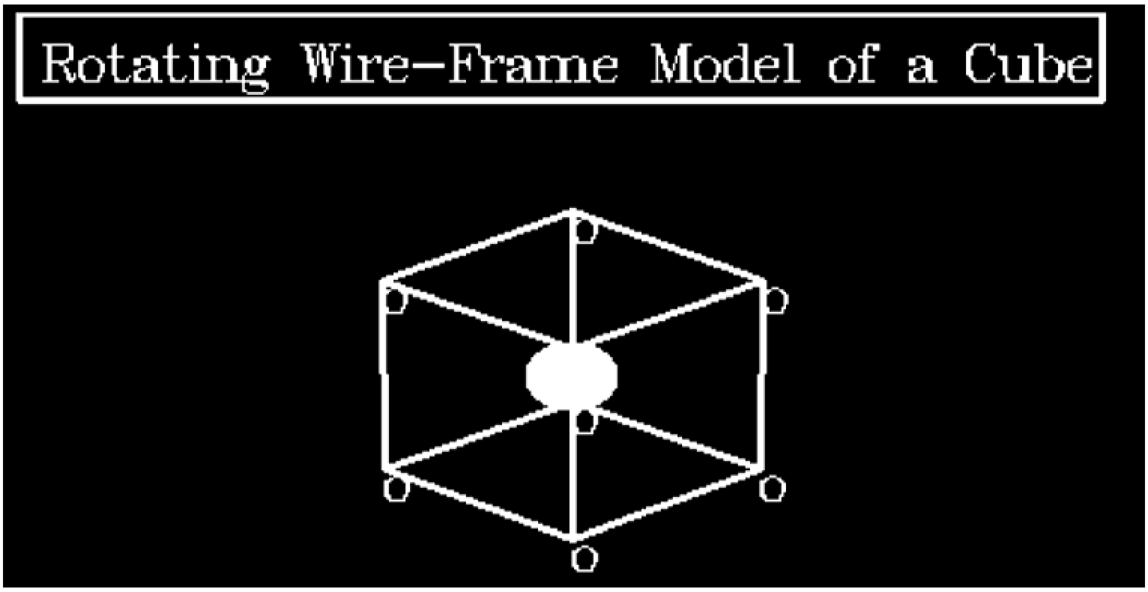
```
float s1,s2,c1,c2;

main()

{

    int gd=VGA,gm=VGAMED,k;

    initgraph(&gd,&gm,"c:\\turboc3\\bgi");

    for(;;)

      {

              if(kbhit()) keypressed();

              rotation();

              pageflip();

              for(k=1;k<=3;k++)

                  {

                    generatepoint(k);

                    if(k==1){originx=320;originy=100;}

                    if(k==2){originx=360;originy=150;}

                    if(k==3){originx=400;originy=200;}


                    drawbox();

                  }

              //getch();

              fflush(stdin);

      }

}/* end of main */


screenxy(int x, int y, int z, int *scx, int *scy)

{    float xe,ye,ze;

      xe=-x*s1 + y*c1;

      ye=-x*c1*c2 - y*s1*c2 + z*s2;

      ze=-x*s2*c1 - y*s1*s2 - z*c2 + rho;

      *scx = (d*xe)/ze;

      *scy=.8*(d*ye)/ze;return 0;

}

drawbox()

{ int i,x1,y1,x2,y2;
```

```
    settextstyle(TRIPLEX_FONT,HORIZ_DIR,3);
    outtextxy(100,5,"Rotating Wire-Frame Model of a Cube");
    rectangle(90,0,540,36);
    outtextxy(250,270,"By");outtextxy(210,300,"R.K. Singla");
    for (i=0;i<19;i++)circle(originx,originy,i);
    for(i=0;i<12;i++)
    { setlinestyle(SOLID_LINE,0,THICK_WIDTH);
        x1=originx+sx[edge[i][0]];
        y1=originy-sy[edge[i][0]];
        x2=originx+sx[edge[i][1]];
        y2=originy-sy[edge[i][1]];
        setcolor(WHITE);
        line(x1,y1,x2,y2);
        outtextxy(x1,y1-10,"o");
      }
      return 0;
}
rotation()
{    switch(rot)
      {
          case 1 : phi      +=.1;break;
          case 2 : theta +=.1;break;
          case 3 : phi      -=.1;break;
          case 4 : theta -=.1;break;
          case 5 : rho      -=800;break;
          case 6 : rho      +=800;break;
          case 7 : d -=800;break;
      }
      s1=sin(theta); s2=sin(phi);
      c1=cos(theta); c2=sin(phi);
      return 0;
}

keypressed()
```

```
{ c=getch();
  if(c==27) { restorecrtmode();exit(0);}
  if(c==0)
  { c=getch();
    switch(c)
    {     case 72 : rot=1;break;
          case 77 : rot=2;break;
          case 80 : rot=3;break;
          case 75 : rot=4;break;
          case 59 : rot=5;break;
          case 60 : rot=6;break;
          case 61 : rot=7;break;
    }
  }
  return 0;
}


pageflip()
{
   setvisualpage(page);
   page=1-page;
   setactivepage(page);
   clearviewport();
   return 0;
}
generatepoint(int j)
{
  int i,x,y,z,x1,y1,z1;
  switch(j)
  {
     case 1 :


        for(i=0;i<8;i++)
         {
```

```
        x=point[i][0];
        y=point[i][1];
        z=point[i][2];
        screenxy(x,y,z,&sx[i],&sy[i]);
    }
  break;

case 2 :

  for(i=0;i<8;i++)
    {
        x=point[i][0];
        y=point[i][1];
        z=point[i][2];
        x1=.866*x-.5*y-1;
        y1=.5*x+.866*y+6;
        z1=3*z+2;
        x=x1;y=y1;z=z1;
        screenxy(x,y,z,&sx[i],&sy[i]);
    }
  break;

case 3 :

  for(i=0;i<8;i++)
    {
        x=point[i][0];
        y=point[i][1];
        z=point[i][2];
        x1=1.93*x-.259*y+.224*z+4;
        y1=.518*x+.966*y-.836*z+2;
        z1=1.732*y+.5*z-2;
        x=x1;y=y1;z=z1;
        screenxy(x,y,z,&sx[i],&sy[i]);
```

```
            }
          break;
  }
  return 0;
}
```

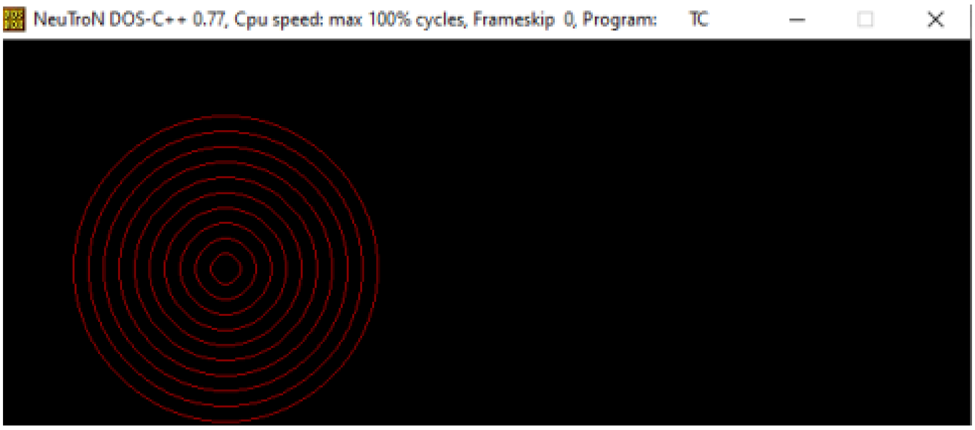


Rotating Wire–Frame Model of a Cube

7. Bresenham's Circle Drawing Algorithm

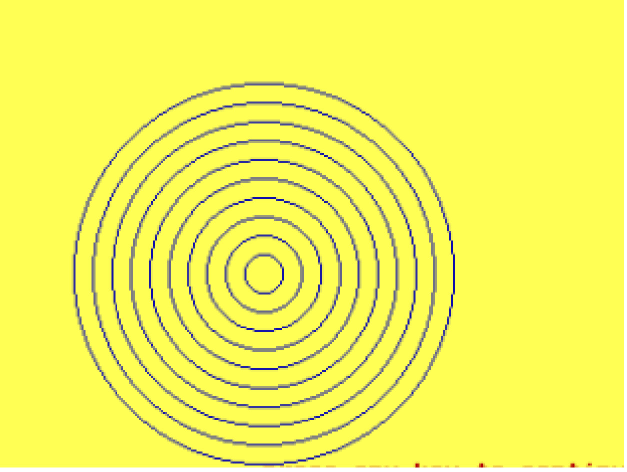
```
#include <iostream.h>
#include<stdio.h>
#include <dos.h>
#include <graphics.h>
#include<conio.h>
// Function to put 8-pixels at different points
void drawSymmetricPoints(int xc, int yc, int x, int y)
{
putpixel(xc+x, yc+y, RED);
putpixel(xc-x, yc+y, RED);
putpixel(xc+x, yc-y, RED);
putpixel(xc-x, yc-y, RED);
putpixel(xc+y, yc+x, RED);
putpixel(xc-y, yc+x, RED);
putpixel(xc+y, yc-x, RED);
putpixel(xc-y, yc-x, RED);
}
```

```
// Function for circle-generation using Bresenham's algorithm
void circleBres(int xc, int yc, int r)
{
int x = 0, y = r;
int p = 3 - 2 * r;
drawSymmetricPoints(xc, yc, x, y);
while (y >= x)
{
// for each pixel we will draw all eight pixels
x++;
// check for decision parameter and correspondingly
// update p, x, y
if (p > 0)
{
y--;
p = p + 4 * (x - y) + 10;
}
else
p = p + 4 * x + 6;
drawSymmetricPoints(xc, yc, x, y);
delay(50);
}
}
// Driver code
int main()
{
int xc = 150, yc = 150, r;
int gd = DETECT, gm;
initgraph(&gd, &gm, "C:\\turboc3\\bgi"); // initialize graphics
for(r=10; r<=100; r=r+10)
  circleBres(xc, yc, r); // function call
gotoxy(50, 20); cout<<"press any key to continue";
getchar();
return 0;
}
```


NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

8. Bresenham's Midpoint Circle Drawing Algorithm

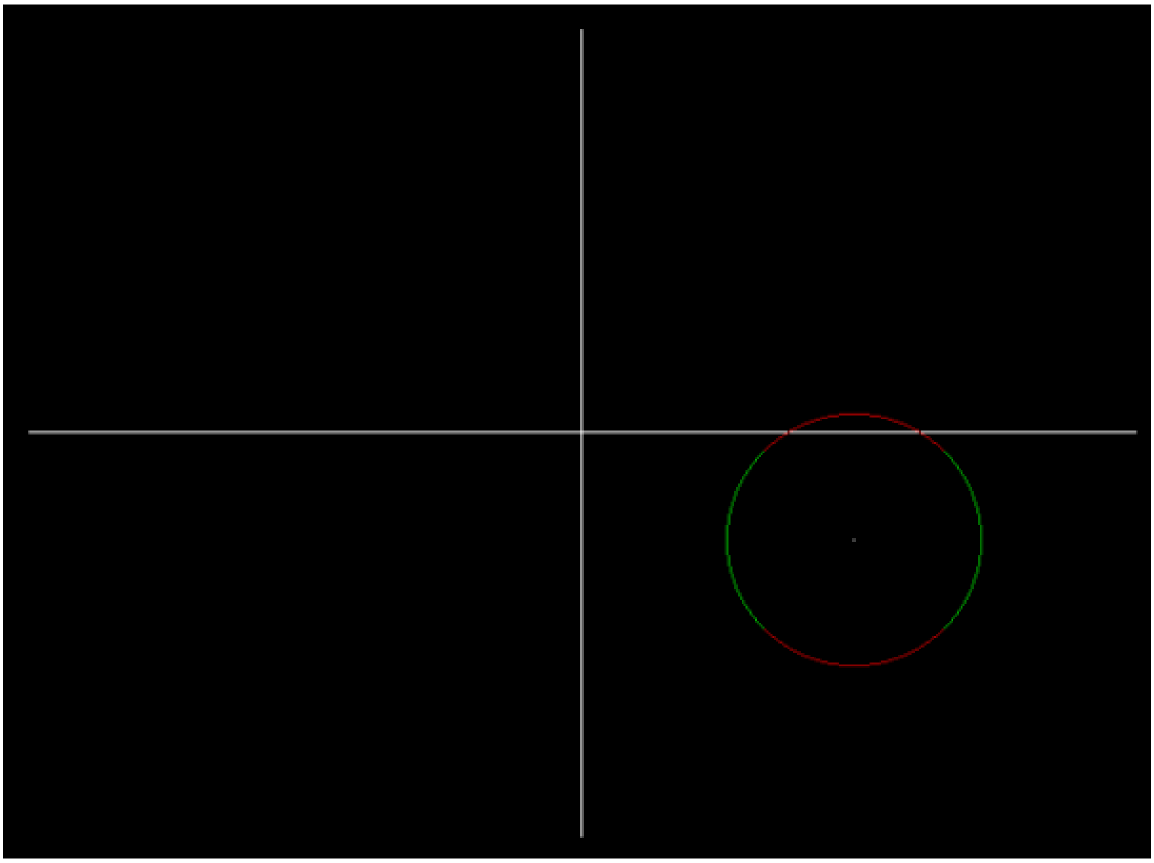
```
#include <iostream.h>

#include<stdio.h>

#include <dos.h>

#include <graphics.h>

#include<conio.h>

// Function to put 8-pixels at different points

void drawSymmetricPoints(int xc, int yc, int x, int y)

{

putpixel(xc+x, yc+y, BLUE);

putpixel(xc-x, yc+y, BLUE);

putpixel(xc+x, yc-y, BLUE);

putpixel(xc-x, yc-y, BLUE);

putpixel(xc+y, yc+x, BLUE);

putpixel(xc-y, yc+x, BLUE);

putpixel(xc+y, yc-x, BLUE);

putpixel(xc-y, yc-x, BLUE);

}

// Function for circle-generation using Bresenham's algorithm

void midpointcircleBres(int xc, int yc, int r)

{

int x = 0, y = r;

int p = 1 - r;

drawSymmetricPoints(xc, yc, x, y);

while (y >= x)
{

// for each pixel we will draw all eight pixels

x++;

// check for decision parameter and correspondingly

// update p, x, y

if (p > 0)

{

y--;

p += 2 * (x - y) + 1;

}

else

p += 2 * x + 1;

drawSymmetricPoints(xc, yc, x, y);

delay(50);

}

}

// Driver code

int main()

{

int xc = 150, yc = 150, r;

int gd = DETECT, gm;

initgraph(&gd, &gm, "C:\\turboc3\\bgi"); // initialize graphics

setbkcolor(YELLOW);

for(r=10; r<=100; r=r+10)

midpointcircleBres(xc, yc, r); // function call

setcolor(RED);

outtextxy(150, 250,"press any key to continue");

getchar();

return 0;

}
```



9. Midpoint circle Algo [origin of coordinate axis= (319, 239)]

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
int main() {
  int h, k, r, p, x, y;
  int centrex = 319, centrey = 239;
  cout<<"Enter co-ordinates of center(h, k): ";
  cin>>h>>k;
  cout<<"Radius: ";
  cin>>r;
  clrscr();
  int gd = DETECT,gm;
  initgraph(&gd, &gm, "C:\\turboc3\\BGI");
  line(319, 15, 319, 464); // y-axis
  line(15, 239, 624, 239); // x-axis
  centrex += h;
  centrey -= k;
  putpixel(centrex, centrey, WHITE); // centre (h, k)
  x = 0;
  y = r;
  p = 1 - r;
  while(x <= y) {
  /* REFLECTIONS */
  putpixel(centrex+x, centrey-y, RED); // O1
  putpixel(centrex+x, centrey+y, RED); // O4
  putpixel(centrex-x, centrey-y, RED); // O5
  putpixel(centrex-x, centrey+y, RED); // O8
  putpixel(centrex+y, centrey-x, GREEN); // O2
  putpixel(centrex+y, centrey+x, GREEN); // O3
  putpixel(centrex-y, centrey-x, GREEN); // O6
  putpixel(centrex-y, centrey+x, GREEN); // O7
  if(p < 0) {
p += 2 * x + 3;
  } else {
p += 2 * x - 2 * y + 5;
y--;
  }
  x++;
  }
  getch();
  closegraph();
  return 0;
}
```

Output:



10.2D Transformations

```
#include<iostream.h>

#include<conio.h>

#include<graphics.h>

#include<stdio.h>

#include<dos.h>

#include<math.h>

#define PI 3.14159265

double cm[4][3]; //globally declared coordinate-matrix

void plot()

{

int x=getmaxx()/2; //getting x axis midpoint

int y=getmaxy()/2; //getting y axis midpoint

line(x,0,x,y*2); //drawing x axis

line(0,y,x*2,y); //drawing y axis

int a[10]={x+cm[0][0],y-cm[0][1],x+cm[1][0],y-cm[1][1],x+cm[2][0],
ycm[2][1],x+cm[3][0],y-cm[3][1],x+cm[0][0],y-cm[0][1]};

drawpoly(5, a);

}//draw the shape(rectangle in this case)

//new=coordi matrix * homogeneous matrix

void mul(double mat[3][3]) //matrix multiplication function, passing homogenous

coordinate matrix

{

double temp_cm[4][3];//temporary storage

for(int i=0;i<4;i++)//4= no of rows in coordinate matrix

{

for(int j=0;j<3;j++)//3=no of columns in homo matrix

{

temp_cm[i][j]=0; //initializing all values to zero

for(int k=0;k<3;k++)//3=no. of rows in homo mat

{

temp_cm[i][j]+=cm[i][k]*mat[k][j];

}
//cout<<"i="<<i<<"j="<<j<<"\t\t"<<temp_cm[i][j]<<"\n";
}
}
for(int m=0;m<4;m++)
{
for(int j=0;j<3;j++)
{
cm[m][j]=temp_cm[m][j];
}
}
}//mat function closed
void translate(double tx,double ty)
{
double t[3][3]={{1.0,0.0,0.0},{0.0,1.0,0.0},{tx,ty,1.0}};
mul(t); //passing this mat for multiplication with coordinate matrix
}
void rotate(double angle)
{
double s=sin(angle*PI/180);
double c=cos(angle*PI/180);
double rot[3][3]={{c,s,0.0},{-1.0*s,c,0.0},{0.0,0.0,1.0}};
double xb=cm[0][0];//saving the values for backup since cm gets updated after
every fn call
double yb=cm[0][1];
translate(-xb,-yb);//bringing one vertex to the origin
mul(rot); //then rotating
translate(xb,yb); //translating back to org position
}
void scale(double x,double y)
{
double sca[3][3]={{x,0.0,0.0},{0.0,y,0.0},{0.0,0.0,1.0}};
double midx=(cm[0][0]+cm[1][0])/2;
double midy=(cm[0][1]+cm[3][1])/2;
translate(-midx,-midy); //moving center to origin
```

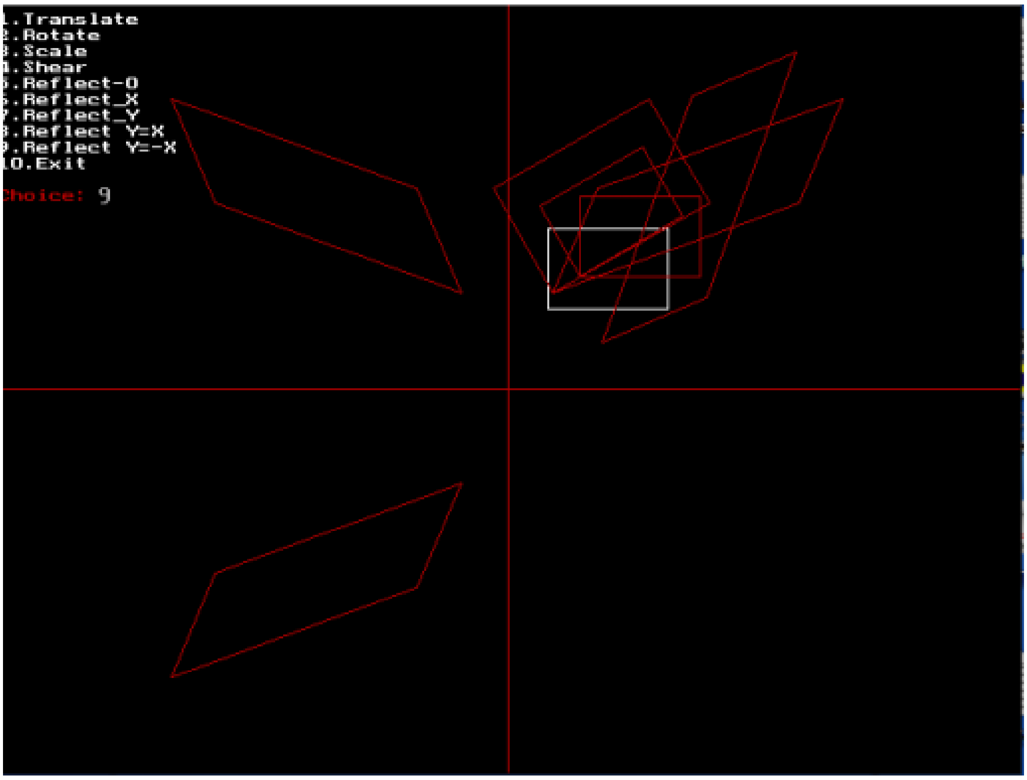
```
mul(sca); //then scaling

translate(midx,midy); //translating center back to org pos

}

void shear(double sx,double sy)

{

double she[3][3]={{1.0,sy,0.0},{sx,1.0,0.0},{0.0,0.0,1.0}};

double xb=cm[0][0];

double yb=cm[0][1];

translate(-xb,-yb); //moving one vertex to origin

mul(she);

translate(xb,yb);

}

void reflect_origin()

{

double ref_mat[3][3]={{-1.0,0.0,0.0},{0.0,-1.0,0.0},{0.0,0.0,1.0}};

mul(ref_mat);

}

void reflect_x()

{

double ref_mat[3][3]={{1.0,0.0,0.0},{0.0,-1.0,0.0},{0.0,0.0,1.0}};

mul(ref_mat);

}

void reflect_y()

{

double ref_mat[3][3]={{-1.0,0.0,0.0},{0.0,1.0,0.0},{0.0,0.0,1.0}};

mul(ref_mat);

}

void reflect_y_equalto_x()

{

double ref_mat[3][3]={{0.0,1.0,0.0},{1.0,0.0,0.0},{0.0,0.0,1.0}};

mul(ref_mat);

}

void reflect_y_equalto_minus_x()

{
```

```
double ref_mat[3][3]={{0.0,-1.0,0.0},{-1.0,0.0,0.0},{0.0,0.0,1.0}};

mul(ref_mat);

}

void main()

{

clrscr();

int gd=DETECT,gm,choice;

initgraph(&gd,&gm,"C:\\TurboC3\\BGI");

double tx,ty,angle,x,y,sx,sy;

//initializing the coordinate matrix for a rectangle

cm[0][0]=25; cm[0][1]=50; cm[0][2]=1;

cm[1][0]=100; cm[1][1]=50; cm[1][2]=1;

cm[2][0]=100; cm[2][1]=100; cm[2][2]=1;

cm[3][0]=25; cm[3][1]=100; cm[3][2]=1;

outtextxy(1, 5,"1.Translate");

outtextxy(1, 15,"2.Rotate");

outtextxy(1, 25,"3.Scale");

outtextxy(1, 35,"4.Shear");

outtextxy(1, 45,"5.Reflect-O");

outtextxy(1, 55,"6.Reflect_X");

outtextxy(1, 65,"7.Reflect_Y");

outtextxy(1, 75,"8.Reflect Y=X");

outtextxy(1, 85,"9.Reflect Y=-X");

outtextxy(1, 95,"10.Exit");

plot();//displaying the rectangle in initial position

delay(500);

//outtextxy(1, 10,"Select:");

while(1)

{

outtextxy(1, 115,"Choice:");

gotoxy(9, 8);cin>>choice;

if(choice==10) { break; }

switch(choice)

{
```

```
case 1:
{
tx=20;
ty=20;
setcolor(RED);
translate(tx,ty);
plot();
break;
}
case 2:
{
angle=30;
rotate(angle);
setcolor(RED);
plot();
break;
}
case 3:
{
x=1.5; y=1.5;
scale(x,y);
setcolor(RED);
plot();
break;
}
case 4:
{
setcolor(RED);
sx=1;
sy=0;
shear(sx,sy);
plot();
break;
}
```

```
case 5:

{

setcolor(RED);

reflect_origin();

plot();

break;

}

case 6:

{

setcolor(RED);

reflect_x();

plot();

break;

}

case 7:

{ setcolor(RED);

reflect_y();

plot();

break;

}

case 8:

{ setcolor(RED);

reflect_y_equalto_x();

plot();

break;

}

case 9:

{ setcolor(RED);

reflect_y_equalto_minus_x();

plot();

break;

}

case 10:

{ break;}
```

```
default:

{ cout<<"Invalid choice"; }

}//end of switch

} //end of while

getch();

closegraph();

}
```

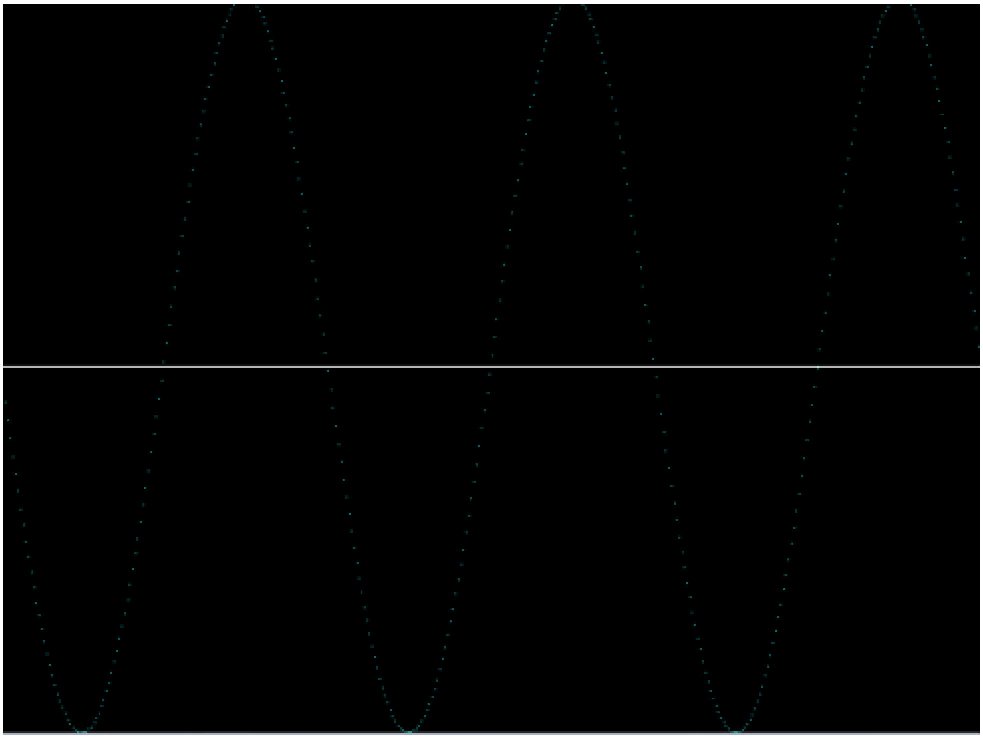


```
1.Translate
2.Rotate
3.Scale
4.Shear
5.Reflect-O
6.Reflect_X
7.Reflect_Y
8.Reflect Y=X
9.Reflect Y=-X
10.Exit

Choice: 9
```

11.PROGRAM TO DRAW SINE & COSINE CURVE

```
#include<graphics.h>

#include<math.h>

#include<conio.h>

#define pie 3.1412

void transform (float minang, float maxang, int option, int l, int b, int t, int r);

void main(){

int option, l, b, t, r;

float ang2, iang, ang1, maxang, maxy;

int gd = VGA, gm = VGAHI, ec = 0;

clrscr();

gotoxy(20, 4); printf("sin/cosine curve drawing");

gotoxy(20, 6); printf("Press 1->SINE CURVE");
gotoxy(20, 8); printf("Press 2->COSINE CURVE");

gotoxy(20, 10); printf("Enter the option:");

gotoxy(40, 10);scanf("%d", &option);

gotoxy(20, 12);printf ("enter the initial angle in degrees:");

gotoxy(56, 12);scanf("%f", &iang); ang1=(iang * pie) / 180;

gotoxy(20, 14);printf("Enter the maximum value of angle in degrees:");

gotoxy(70, 14);scanf("%f", &maxang); ang2=(maxang * pie)/180;

printf("\n\n\n Press any key to conitune"); getch();

initgraph(&gd, &gm, "C:\\turboc3\\bgi");

ec=graphresult();

if(ec!=0)printf("Graphics system error\n",grapherrormsg(ec));

//printf("Enter the coordinates for the viewport:\n");

//scanf("%d %d %d %d", &l, &t, &r, &b);

l=0; t=0; r=639; b=479;

setviewport(l, t, r, b, 0);

//printf("\n%d %d", getmaxx(), getmaxy());

cleardevice();

rectangle (l, t, r, b);

line(l, b - ((b - t)/2), r, b-((b - t)/2));

transform (ang1, ang2, option, l, t, r, b);

getch();

}

/***********************************************************/

/*Transformation of world coordinates to screen coordinates**/

/*World coordinates: xmin, ymin, xmax, y max **/

/*screen coordinates: sxmin, symin, sxmax, symax **/

/***********************************************************/

void transform (float minang, float maxang, int option,int l, int t, int r, int b){

float temp, x, y, xmax, ymax, xmin, ymin, sxmax, symax, sxmin, symin;

double sx, sy;

sxmin = l; sxmax = r;

symin = t; symax = b;

xmin = minang; xmax = maxang;

ymin = -1.0; ymax = 1.0;
```

```
while (minang <= maxang){

if (option == 1) temp = sin(minang); else temp = cos(minang);

y = -(temp);

x = minang;

sx = (double)((sxmin)+(((sxmax-sxmin)/(xmax- xmin)) *(x-xmin)));

sy = (double)((symin)+(((symax-symin)/(ymax- ymin)) *(y-ymin)));

putpixel((long)sx, (long)sy,3);

minang += 0.05;

}

}
```

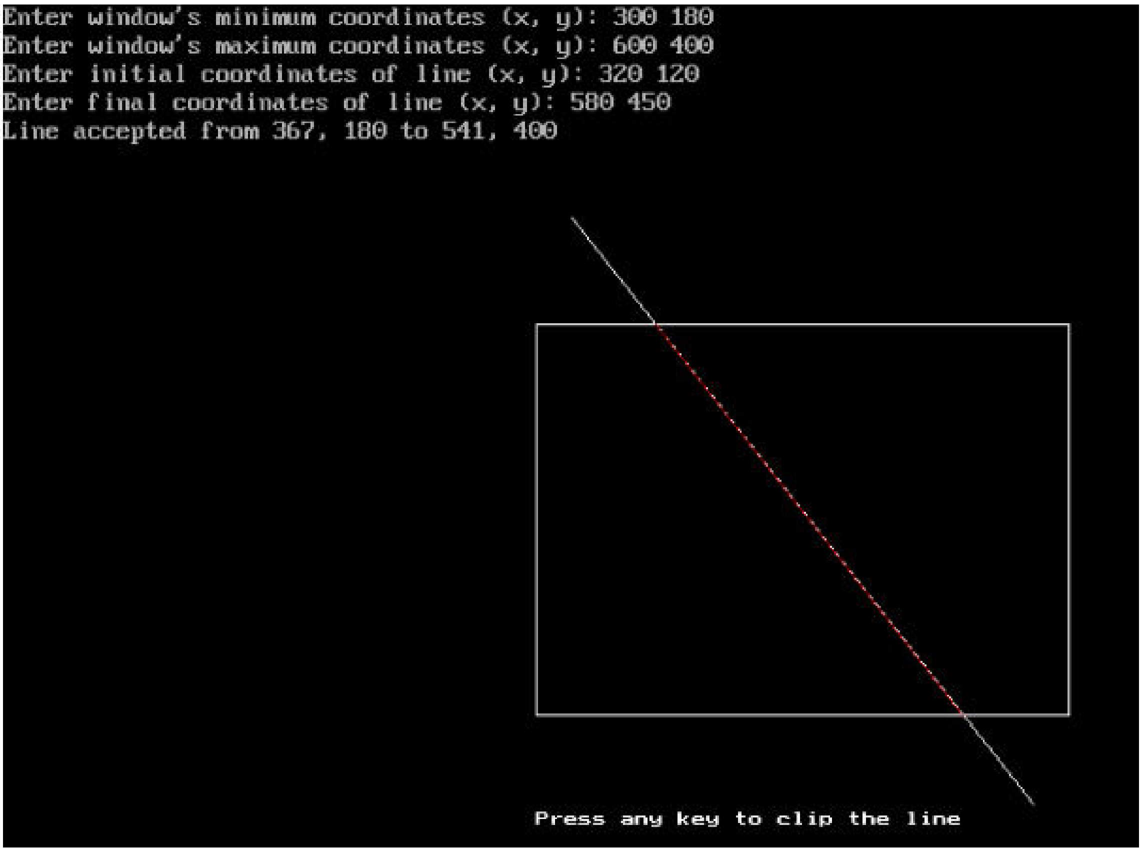


12. Cohen-Sutherland Line Clipping Algorithm

```
#include<iostream.h>

#include<conio.h>

#include<graphics.h>

// Defining region codes

const int INSIDE = 0; // 0000

const int LEFT = 1, RIGHT = 2, BOTTOM = 4, TOP = 8;

//int x_max = 10, y_max = 8, x_min = 4, y_min = 4;

// assigning region code

int computeCode(double x, double y)

{
  int code = INSIDE;
  if (x < x_min)
code |= LEFT;
  else if (x > x_max)
code |= RIGHT;
  if (y < y_min)
code |= BOTTOM;
  else if (y > y_max)
code |= TOP;
  return code;
}
void cohenSthAlgo(double x1, double y1, double x2, double y2)
{
  // region codes (P1 & P2)
  int code1 = computeCode(x1, y1);
  int code2 = computeCode(x2, y2);
  int accept = 0;
  while (1)
  {
if ((code1 == 0) && (code2 == 0))
{
  accept = 1;
  break;
}
else if (code1 & code2)
  break;
else
{
  int code_out;
  double x, y;
  if (code1 != 0)
code_out = code1;
  else
code_out = code2;
```

```
  // y = y1 + slope * (x - x1),
  // x = x1 + (1 / slope) * (y - y1)
  if (code_out & TOP)
  {
x = x1 + (x2 - x1) * (y_max - y1) / (y2 - y1);
y = y_max;
  }
  else if (code_out & BOTTOM)
  {
x = x1 + (x2 - x1) * (y_min - y1) / (y2 - y1);
y = y_min;
  }
  else if (code_out & RIGHT)
  {
y = y1 + (y2 - y1) * (x_max - x1) / (x2 - x1);
x = x_max;
  }
  else if (code_out & LEFT)
  {
y = y1 + (y2 - y1) * (x_min - x1) / (x2 - x1);
x = x_min;
  }
  if (code_out == code1)
  { x1 = x;
y1 = y;
code1 = computeCode(x1, y1);
  }
  else
  { x2 = x;
y2 = y;
code2 = computeCode(x2, y2);
  }
}//end of else
  } //end of while
  if (accept)
  {
cout <<"Line accepted from " << (int)(x1+0.5) << ", "
  << (int)(y1+0.5) << " to "<< (int)(x2+0.5)
  << ", " << (int)(y2+0.5) << endl;
line(x1, y1, x2, y2);
  }
  else
cout << "Line rejected" << endl;
}
void main()
{
  clrscr();
  double px1, py1, px2, py2, left, top, right, bottom;
  int gd = DETECT,gm;
  initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
  cout<<"Enter window\'s minimum coordinates (x, y): ";
  cin>>x_min>>y_min;
  cout<<"Enter window\'s maximum coordinates (x, y): ";
  cin>>x_max>>y_max;
  cout<<"Enter initial coordinates of line (x, y): ";
  cin>>px1>>py1;
  cout<<"Enter final coordinates of line (x, y): ";
  cin>>px2>>py2;
  rectangle(x_min, y_max, x_max, y_min);
  line(px1, py1, px2, py2);
  outtextxy(300, 455,"Press any key to clip the line");
  getch();
  setcolor(RED);
  cohenSthAlgo(px1, py1, px2, py2);
  getch();
}
```

Output:



```
Enter window's minimum coordinates (x, y): 300 180
Enter window's maximum coordinates (x, y): 600 400
Enter initial coordinates of line (x, y): 320 120
Enter final coordinates of line (x, y): 580 450
Line accepted from 367, 180 to 541, 400
```

```
Press any key to clip the line
```

13. Rubber Line and Inking Algorithm for Interaction

06-mouse.h

```
#include<dos.h>
#include<graphics.h>
#include<stdlib.h>
#define LFTCLICK (1)
union REGS ii,oo;
int InitMouse( void )
{
  ii.x.ax=0;
  int86(0x33,&ii,&oo);
  return oo.x.ax;
} /*--InitMouse( )---*/
/*-----------------------------------------------
ShowMousePtr - Shows Mouse Pointer. */
void ShowMousePtr( void )
{
  ii.x.ax=1;
  int86(0x33,&ii,&oo);
```

```
} /*--ShowMousePtr( )----*/

/*-----------------------------------------------

HideMousePtr - Hide Mouse Pointer. */

void HideMousePtr( void )

{

  ii.x.ax=2;

  int86(0x33,&ii,&oo);

} /*--HideMousePtr( )-----*/

/*-----------------------------------------------

MoveMousePtr - Move Mouse Pointer to (x, y). */

void MoveMousePtr( int x, int y )

{

  ii.x.ax=4;

  ii.x.cx=x;

  ii.x.dx=y;

  int86(0x33,&ii,&oo);

} /*--MoveMousePtr( )-----*/

/*-----------------------------------------------

RestrictMousePtr-Restrict Mouse Pointer to the specified coordinates */

void RestrictMousePtr( int x1, int y1, int x2, int y2 )

{ ii.x.ax=7;

  ii.x.cx=x1;

  ii.x.dx=x2;

  int86(0x33,&ii,&oo);

  ii.x.ax=8;

  ii.x.cx=y1;

  ii.x.dx=y2;

  int86(0x33,&ii,&oo);

} /*--RestrictMousePtr( )--------*/

/*-----------------------------------------------

GetMousePos - Gets Mouse position & mouse button value. */

void GetMousePos( int *mbutton, int *mx, int *my )

{

  ii.x.ax=3;
  int86(0x33,&ii,&oo);
  *mx=oo.x.cx;
  *my=oo.x.dx;
  *mbutton=oo.x.bx;
} /*--GetMousePos( )------*/
```

06-paint.cpp

```
/*-----------------------------------------------------------------
Mini Paintbrush demo
*---
*/
#include <dos.h>
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include <graphics.h>
#include "c:\turboc3\source\06-mouse.h"
#define ESC (27)
#define ISDRAWBOX(x, y) ( x>141 && x<498 && y>131 && y<298 )
typedef int BOOLEAN;
#define FALSE (0)
#define TRUE (1)
#define PRESS (0)
#define NORMAL (1)
#define MAXCMDBUTTON (3)
#define BRUSH (0)
#define LINE (1)
#define QUIT (2)
struct RecButtonCoord
{
int x1
int x2;
int y2;
};
struct RecButtonCoord RecBut_Cd[MAXCMDBUTTON];
```

```
void far MyOuttextxy( int x, int y, char far *str, int color );
void MyRectangle( int x1, int y1, int x2, int y2, int upcolor, int
lowcolor ); ;
int y1;
void InitVB( void );
void InitScreen( void );
void VBForm( int x1, int y1, int x2, int y2, char *title );
void VBFrame( int x1, int y1, int x2, int y2 );
void VBDrawBox( int x1, int y1, int x2, int y2 );
void CmdButton( int cmdno, int status );
int CmdButtonVal( int x, int y );
void ShowStatus( int msgno );
/*-----------------------------------------------
MyOuttextxy - Prints text with specified color */
void far MyOuttextxy( int x, int y, char far *str, int color )
{
setcolor( color );
outtextxy( x, y, str );
} /*--MyOuttextxy( )-----------*/
/*-----------------------------------------------
MyRectangle - Rectangle with upcolor for Ú, lowcolor for Ù.
It's for Command Button effect. */
void MyRectangle( int x1, int y1, int x2, int y2, int upcolor, int
lowcolor )
{
setcolor( upcolor );
line( x1, y1, x2, y1 );
line( x1, y1, x1, y2 );
setcolor( lowcolor );
line( x1, y2, x2, y2 );
line( x2, y1, x2, y2);
} /*--MyRectangle( )-------------*/
/*-----------------------------------------------
InitVB - Initializes VB. ie, Checks errors. */
```

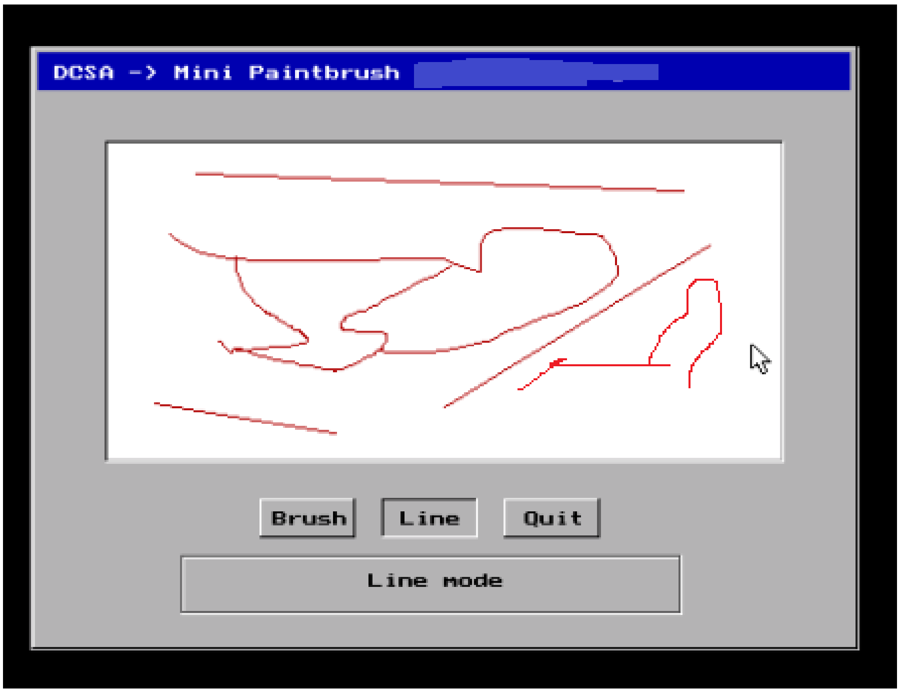
```
void InitVB( void )
{
int gdriver = VGA, gmode = VGAHI, error;
if ( !InitMouse( ) )
{
cprintf( "Mouse support needed! \r\n\a" );
exit( 1 );
}
initgraph( &gdriver, &gmode, "c:\\turboc3\\bgi" );
error = graphresult( );
if ( error != grOk )
{
closegraph( );
cprintf( "Graphics error: %s \r\n\a", grapherrormsg( error ) );
exit( 1 );
}
} /*--InitVB( )-------*/
/*-----------------------------------------------
InitScreen - Initializes Screen. */
void InitScreen( void )
{
int i, x, y;
VBForm( 100,80,540,400, "DCSA -> Mini Paintbrush -> Dr R K Singla" );
VBFrame( 180, 350, 445, 380 );
VBDrawBox( 140, 130, 500, 300 );
for( i= 0, x = 222, y = 320 ; i < 3 ; x += 65, ++i )
{
RecBut_Cd[i].x1 = x;
RecBut_Cd[i].y1 = y;
RecBut_Cd[i].x2 = x + 50;
RecBut_Cd[i].y2 = y + 20;
CmdButton( i, NORMAL );
}
/* Labels for Command Button... */
MyOuttextxy( 229, 327, "Brush", BLACK );
MyOuttextxy( 297, 327, "Line", BLACK );
MyOuttextxy( 363, 327, "Quit", BLACK );
} /*--InitScreen( )------*/
/*----------------------------------------------
VBForm - Creates a Window with the given title. */
void VBForm( int x1, int y1, int x2, int y2, char *title )
{
setfillstyle( SOLID_FILL, LIGHTGRAY );
bar( x1, y1, x2, y2 );
setfillstyle( SOLID_FILL, BLUE );
bar( x1+4, y1+3, x2-5, y1+22 );
MyOuttextxy( x1+13, y1+10, title, WHITE );
MyRectangle( x1+1, y1, x2-1, y2-1, WHITE, BLACK );
} /*--VBForm( )-----------*/
/*----------------------------------------------
VBFrame - Creates VB like Frame. */
void VBFrame( int x1, int y1, int x2, int y2 )
{
MyRectangle( x1+1, y1+1, x2, y2, WHITE, DARKGRAY );
MyRectangle( x1, y1, x2+1, y2+1, DARKGRAY, WHITE );
} /*--VBFrame( )--------------*/
/*----------------------------------------------
VBDrawBox - Creates Drawing Box. */
void VBDrawBox( int x1, int y1, int x2, int y2 )
{
setfillstyle( SOLID_FILL, WHITE );
bar( x1+1, y1+1, x2-2, y2-2 );
MyRectangle( x1, y1, x2, y2, BLACK, WHITE);
} /*--VBDrawBox( )--------*/
/*----------------------------------------------
CmdButton - Draws Command Button for specified status.
status are NORMAL, PRESS */
void CmdButton( int cmdno, int status )
```

```
{
if ( status==NORMAL )
MyRectangle( RecBut_Cd[cmdno].x1, RecBut_Cd[cmdno].y1,
RecBut_Cd[cmdno].x2, RecBut_Cd[cmdno].y2, WHITE, BLACK
);
else
MyRectangle( RecBut_Cd[cmdno].x1, RecBut_Cd[cmdno].y1,
RecBut_Cd[cmdno].x2, RecBut_Cd[cmdno].y2, BLACK, WHITE );
} /*--CmdButton( )----------*/
/*---------------------------------------------
CmdButtonVal - Returns Command Button value. */
int CmdButtonVal( int x, int y )
{
BOOLEAN found = FALSE;
int i;
for( i= 0; !found && i < MAXCMDBUTTON ; ++i )
found = ( x > RecBut_Cd[i].x1 && x < RecBut_Cd[i].x2
&& y > RecBut_Cd[i].y1 && y < RecBut_Cd[i].y2);
if ( found )
--i;
return( i );
} /*--CmdButtonVal( )----------*/
/*---------------------------------------------
ShowStatus - Display messages. */
void ShowStatus( int msgno )
{
char *message[] = {
"Brush mode",
"Line mode"
};
if ( msgno==0 || msgno==1 )
{
setfillstyle( SOLID_FILL, LIGHTGRAY );
bar( 280, 360, 438, 370 );
```

```
MyOuttextxy( 280, 360, message[msgno], BLACK );

}

} /*--ShowStatus( )--------*/

/*-----------------------------------------------

main - Main of VB */

int main( void )

{

int mx, my, x1, x2, y1, y2, mbutton, cmdno, prevcmdno=0;

const int brushcolor = RED; /* choose default brush color */

BOOLEAN stayin = TRUE;

InitVB( );

InitScreen( );

CmdButton( BRUSH, PRESS ); /* Force <Brush> button to default */

ShowStatus( BRUSH );

ShowMousePtr( );

while( stayin )

{

/* if ESC is pressed, then quit! */

if ( kbhit( ) )

stayin = ( getch( )!=ESC );

GetMousePos( &mbutton, &mx, &my );

if (mbutton==LFTCLICK )

{

cmdno = CmdButtonVal( mx, my );

if ( cmdno!=MAXCMDBUTTON && cmdno != prevcmdno )

{

HideMousePtr( );

CmdButton( cmdno, PRESS );

CmdButton( prevcmdno, NORMAL );

ShowStatus( cmdno );

prevcmdno = cmdno;

ShowMousePtr( );

stayin = (cmdno!=QUIT);

}
```

```
if (ISDRAWBOX( mx, my ) )

{

RestrictMousePtr( 142, 132, 497, 297 );

switch (prevcmdno)

{

case BRUSH:

x1 = mx;

y1 = my;

setcolor( brushcolor );

HideMousePtr( );

putpixel( mx, my, brushcolor );

ShowMousePtr( );

do

{

GetMousePos( &mbutton, &mx, &my );

if ( x1!=mx || y1!=my )

{

HideMousePtr( );

line( x1, y1, mx, my );

ShowMousePtr( );

x1 = mx;

y1 = my;

}

} while(mbutton==LFTCLICK);

break;

case LINE:

x2 = x1 = mx;

y2 = y1 = my;

/* Note! in XOR_PUT mode, you must setcolor to 'WHITE-brushcolor'

*/

setwritemode( XOR_PUT );

setcolor(WHITE-brushcolor );

do

{
```

```
GetMousePos( &mbutton, &mx, &my );
if (mx!=x2 || my!= y2)
{
HideMousePtr( );
line( x1, y1, x2, y2 );
line( x1, y1, mx, my );
ShowMousePtr( );
x2 = mx;
y2 = my;
}
} while(mbutton==LFTCLICK);
setwritemode( COPY_PUT );
/* Note! in COPY_PUT mode, you must setcolor to 'brushcolor'*/
setcolor( brushcolor );
HideMousePtr( );
line( x1, y1, mx, my );
ShowMousePtr( );
} // end of switch
RestrictMousePtr( 0, 0, 640, 480 );
} //end of ISDRAWBOX
} //end of if mbutton==LFTCLICK
} //end of while with stayin
closegraph( );
return( 0 );
} /*--main( )---------*/
```



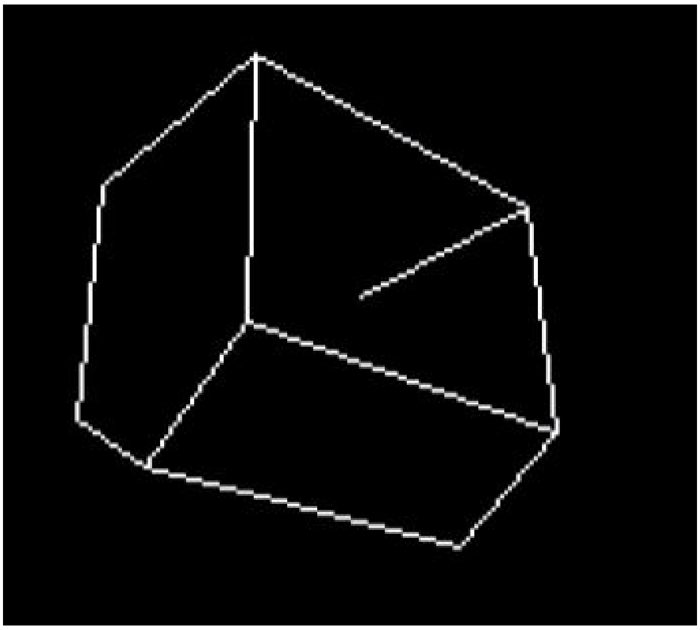
14.program uses 3D Animation to display a house with hidden lines.

```
#include <graphics.h>

#include <math.h>

#include <stdio.h>

#include <dos.h>

/* define global variables */

int rho=40,d=250;

/* define the 10 corners of the house */

int v[10][3]=

        {

          {5,7,-5}, {5,7,5}, {5,-7,5}, {5,-7,-5},

          {-5,7,-5}, {-5,-7,-5},    {-5,-7,5}, {-5,7,5},

          {0,7,8}, {0,-7,8}

        };

int sv[10][2],rot=1;

int nps[7]={5,6,5,6,5,5,5};

static int s[7][6] = {

                    {0,1,2,3,0}, {0,4,7,8,1,0}, {4,5,6,7,4},

                    {3,2,9,6,5,3}, {2,1,8,9,2}, {6,9,8,7,6},

                    {0,3,5,4,0}

                 };

int n[7][3], e[12][3];

int originx=320,originy=150, page=0;
```

```
char c;

double theta=.7,phi=0.9;

float s1,s2,c1,c2;

main()

{

    int gd=VGA,gm=VGAMED;

    initgraph(&gd,&gm,"c:\\turboc3\\bgi");

    for(;;)

      {

              if(kbhit()) keypressed();

              rotation();

              pageflip();

              generatepoint();

              normvector();

              visibility();

              drawedges();

      }

}/* end of main */

screenxy(int x, int y, int z, int *scx, int *scy)

{

      float xe,ye,ze;

      xe=-x*s1 + y*c1;

      ye=-x*c1*c2 - y*s1*c2 + z*s2;

      ze=-x*s2*c1 - y*s1*s2 - z*c2 + rho;

      *scx = (d*xe)/ze;

      *scy = (d*ye)/ze;

      return 0;

}


rotation()

{

    switch(rot)

      { case 1 : phi      +=.1;break;

          case 2 : theta +=.1;break;
          case 3 : phi      -=.1;break;
          case 4 : theta -=.1;break;
          case 5 : rho      -=800;break;
          case 6 : rho      +=800;break;
          case 7 : d -=800;break;
      }
      s1=sin(theta); s2=sin(phi);
      c1=cos(theta); c2=cos(phi);
      return 0;
}
keypressed()
{
    c=getch();
    if(c==27) { restorecrtmode();exit(0);}
    if(c==0)
    {
        c=getch();
        switch(c)
        {     case 72 : rot=1;break;
             case 77 : rot=2;break;
             case 80 : rot=3;break;
             case 75 : rot=4;break;
             case 59 : rot=5;break;
             case 60 : rot=6;break;
             case 61 : rot=7;break;
        }
    }
    return 0;
}
pageflip()
{    setvisualpage(page);
      page=1-page;
      setactivepage(page);
      clearviewport();delay(200);
```

```
     return 0;
}
generatepoint()
{
    int i,x,y,z;
    for(i=0;i<10;i++)
    {      x=v[i][0];
           y=v[i][1];
           z=v[i][2];
           screenxy(x,y,z,&sv[i][0],&sv[i][1]);
           sv[i][0]=originx+sv[i][0];
           sv[i][1]=originy-sv[i][1];
     }
     return 0;
}
normvector()
{
    int i,j1,j2,k,u1,u2,u3,v1,v2,v3;
    for(i=0;i<7;i++)
        { j1=s[i][1]; j2=s[i][2]; k=s[i][0];
           u1=v[j1][0]-v[k][0];
           u2=v[j1][1]-v[k][1];
           u3=v[j1][2]-v[k][2];
           v1=v[j2][0]-v[k][0];
           v2=v[j2][1]-v[k][1];
           v3=v[j2][2]-v[k][2];
           n[i][0]=u2*v3-v2*u3;
           n[i][1]=u3*v1-v3*u1;
           n[i][2]=u1*v2-v1*u2;
        }
        return 0;
}
visibility()
{
```

```
int m,i,e2,e1,j,k,flag;

float xe,ye,ze,wx,wy,wz,dot;


xe=rho*s2*c1; ye=rho*s2*s1; ze=rho*c2;

m=0;

for(i=0;i<7;i++)

    {

        e2=s[i][0];

        wx=xe-v[e2][0];

        wy=ye-v[e2][1];

        wz=ze-v[e2][2];

        dot=n[i][0]*wx+n[i][1]*wy+n[i][2]*wz;

        if(dot<=0)continue;

        e1=s[i][0];

        for(j=1;j<=nps[i];j++)

        {

        e2=s[i][j];

        for(k=0;k<=m;k++)

            { flag=0;

                if(e[k][0]==e2 && e[k][1]==e1)

                {

                    e[k][2]=2;

                    flag=1;

                }

                if(flag==1)break;

            }/* end of k*/

        if(flag==0){

                        e[m][0]=e1;

                        e[m][1]=e2;

                        e[m][2]=1;

                        m=m+1;

                    }

        e1=e2;

        }/* end of j */
        }/* end of i */

        return 0;

}/* end of function */


drawedges()

{

    int i,j,k;

    for(i=0;i<12;i++)

          {

            if(e[i][2]==0)continue;

            j=e[i][0];k=e[i][1];

            setcolor(WHITE);

            line(sv[j][0],sv[j][1],sv[k][0],sv[k][1]);

          }

          return 0;

}
```

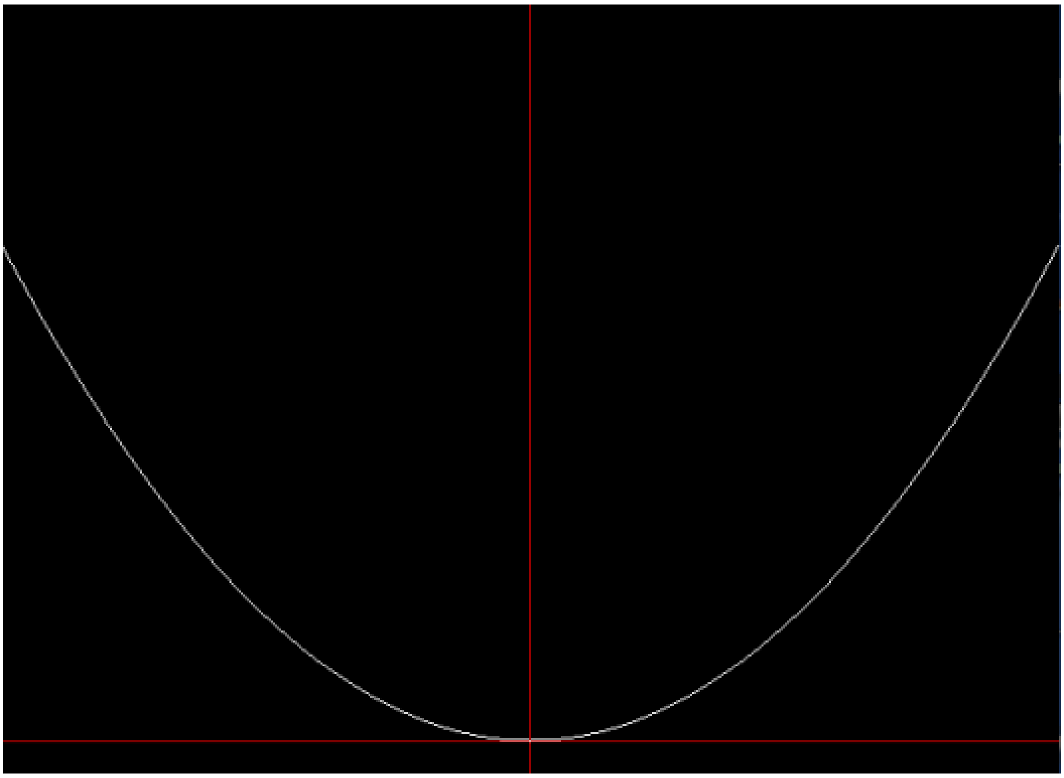


15. Plotting the Quadratic function y=ax2+bx+c

```
#include <graphics.h>

#include <conio.h>

/** Change the values below to suit your screen resolution.

  These values will probably suit most users. ***/
```

```
#define MAXX 640

#define MAXY 480

/* Co-ordinate conversion functions, prototypes and declarations */

float screen_y(float, float, float);

float screen_y(float, float, float);

float screen_x(float xb, float x, float xe)

{

return ((x-xb)/(xe-xb)*MAXX);

}

float screen_y(float yb, float y, float ye)

{

return (MAXY-(y-yb)/(ye-yb)*MAXY);

}

void main(void) {

/* Change the values of 'drv' and 'mode' if you change MAXX and MAXY */

int drv=VGA, mode=VGAHI;

float a,b,c,xb,xe,yb,ye,x,y;

/* You can play with the parameters below */

a=1; b=0; c=0;

xb=-10;

xe=10;

yb=-10;

ye=150;

/* Stop playing!!! */

initgraph(&drv, &mode,"C:\\turboc3\\bgi");

/* Draw the X and Y axes */

setcolor(4);

line(0,screen_y(yb,0,ye),MAXX,screen_y(yb,0,ye));

line(screen_x(xb,0,xe),0,screen_x(xb,0,xe),MAXY);

setcolor(15);

moveto(0,screen_y(yb,a*xb*xb+b*xb+c,ye));

for(x=xb;x<=xe;x+=(xe-xb)/MAXX)

{
```

```
y=a*x*x+b*x+c;

lineto(screen_x(xb,x,xe),screen_y(yb,y,ye));

}

getch(); closegraph();

return;

}
```



16. Zooming and Panning-graphs in action

```
#include <graphics.h>

#include <conio.h>

#include <math.h>

/* Change the values below to suit your screen resolution.

  These values will probably suit most users. */

#define MAXX 640

#define MAXY 480

/* Co-ordinate conversion functions, prototypes and declarations */

float screen_y(float, float, float);

float screen_y(float, float, float);

/* The user-defined function to be plotted */

float my_func(float);
```

```
/* You can use all the parameters below as coeffecients

  to whatever function you wish to plot. You should assign them

  values in main() and define your function in my_func() */

float a,b,c,d,e,f,g,h,i,j;

float screen_x(float xb, float x, float xe)

{

return ((x-xb)/(xe-xb)*MAXX);

}

float screen_y(float yb, float y, float ye)

{

return (MAXY-(y-yb)/(ye-yb)*MAXY);

}

float my_func(float x)

{

/* Define your function here */

return a * sin( ( b * x + c ) * 3.1415 / 180.0) ;

}

void main(void)

{

/* Change the values of 'drv' and 'mode' if you change MAXX and MAXY */

int drv=VGA, mode=VGAHI;

float xb,xe,yb,ye,x,y,potx,poty;

char key;

/* You can play with the parameters below */

a = 1; b = 1; c = 0;

xb = -360;

xe = 360;

yb = -10;

ye = 10;

/* Stop playing !!! */

initgraph(&drv,&mode,"c:\\turboc3\\bgi");

/* Draw the X and Y axes */

setcolor(4);
```

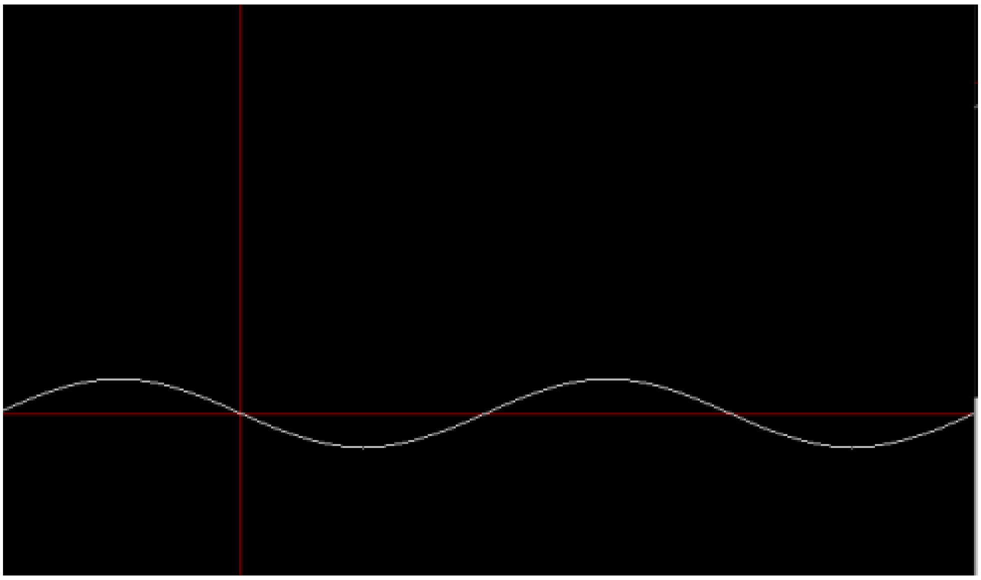
```
line(0,screen_y(yb,0,ye),MAXX,screen_y(yb,0,ye));
line(screen_x(xb,0,xe),0,screen_x(xb,0,xe),MAXY);
setcolor(15);
moveto(0,screen_y(yb,my_func(xb),ye));
for(x=xb;x<=xe;x+=(xe-xb)/MAXX)
{
y = my_func(x);
lineto(screen_x(xb,x,xe),screen_y(yb,y,ye));
}
while((key=getch())!=27) /* Repeat the loop until [ESC] pressed */
{
potx = xb;
poty = yb;
switch(key)
{
case 'z': /* pan left */
xb -= (xe-xb)/4.0;
xe -= (xe-potx)/4.0;
break;
case 'x': /* pan right */
xb += (xe-xb)/4.0;
xe += (xe-potx)/4.0;
break;
case 'q': /* pan up */
yb += (ye-yb)/4.0;
ye += (ye-poty)/4.0;
break;
case 'w': /* pan down */
yb -= (ye-yb)/4.0;
ye -= (ye-poty)/4.0;
break;
case '+': /* Y-axis zoom out */
yb -= (yb-ye)/4.0;
```

```
ye += (poty-ye)/4.0;

break;

case '-': /* Y-axis zoom in */

yb += (yb-ye)/4.0;

ye -= (poty-ye)/4.0;

break;

case '[': /* X-axis zoom out */

xb += (xb-xe)/4.0;

xe -= (potx-xe)/4.0;

break;

case ']': /* X-axis zoom in */

xb -= (xb-xe)/4.0;

xe += (potx-xe)/4.0;

break;

case '/': /* Home (initial view) */

xb = -360;

xe = 360;

yb = 10;

ye = -10;

break;

default:

//continue;

break;

}

cleardevice();

/* Draw the X and Y axes */

setcolor(4);

line(0,screen_y(yb,0,ye),MAXX,screen_y(yb,0,ye));

line(screen_x(xb,0,xe),0,screen_x(xb,0,xe),MAXY);

setcolor(15);

moveto(0,screen_y(yb,my_func(xb),ye));

for(x=xb;x<=xe;x+=(xe-xb)/MAXX)

{
```

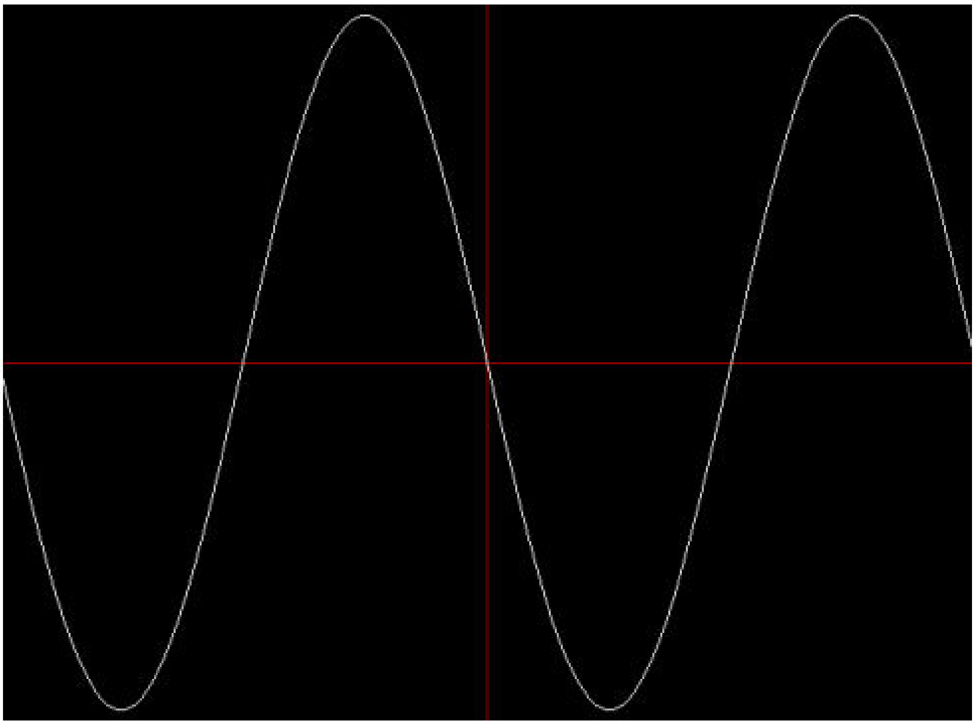
```
y = my_func(x);
lineto(screen_x(xb,x,xe),screen_y(yb,y,ye));
}
}
closegraph();
return;
}
```

Output:

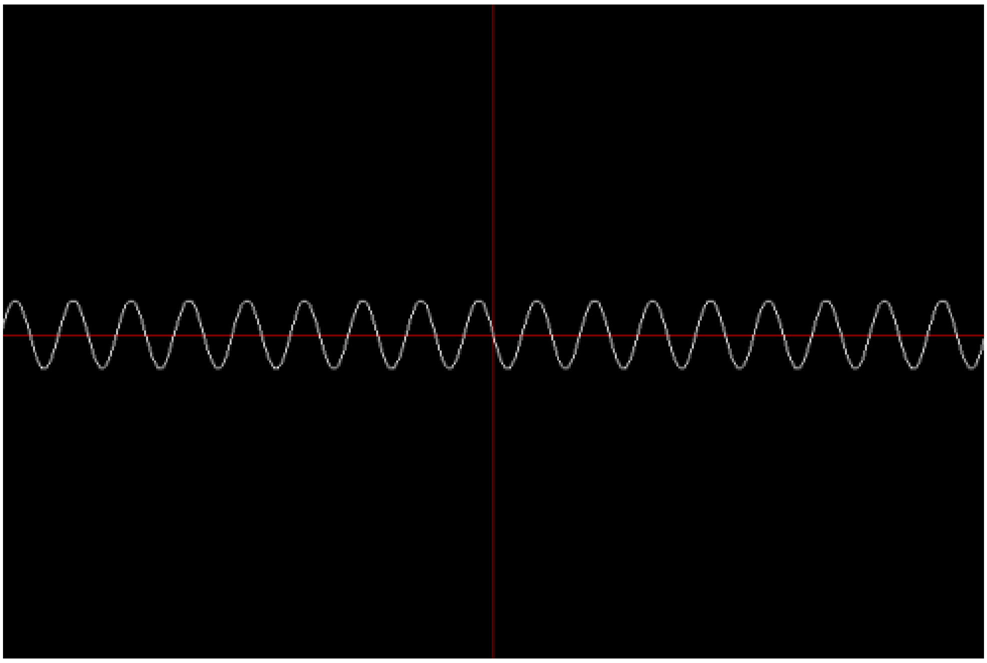
Panning (Left/Right/Up/Down) Demo:



Zooming in/out About Y-axis Demo:



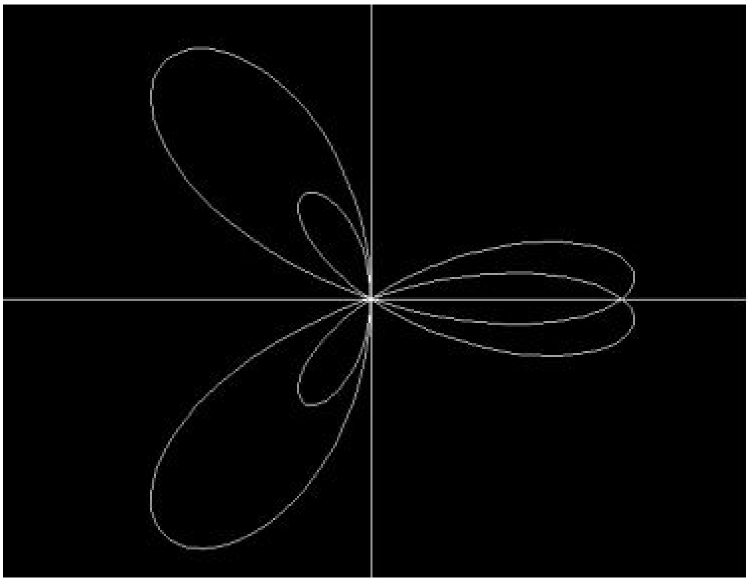
Zooming in/out About X-axis Demo:



17. Plotting Polar Functions

```
#include <graphics.h>
#include <conio.h>
#include <math.h>
/** Change the values below to suit your screen resolution.
  These values will probably suit most users. ***/
#define MAXX 640
#define MAXY 480
/* Co-ordinate conversion functions, prototypes and declarations */
float screen_y(float, float, float);
float screen_y(float, float, float);
float P2Q_x(float, float);
float P2Q_y(float, float);
/*********************************************************/
float screen_x(float xb, float x, float xe)
{
return ( (x - xb) / (xe - xb) * MAXX );
}
float screen_y(float yb, float y, float ye)
{
return ( MAXY - (y - yb) / (ye - yb) * MAXY);
```

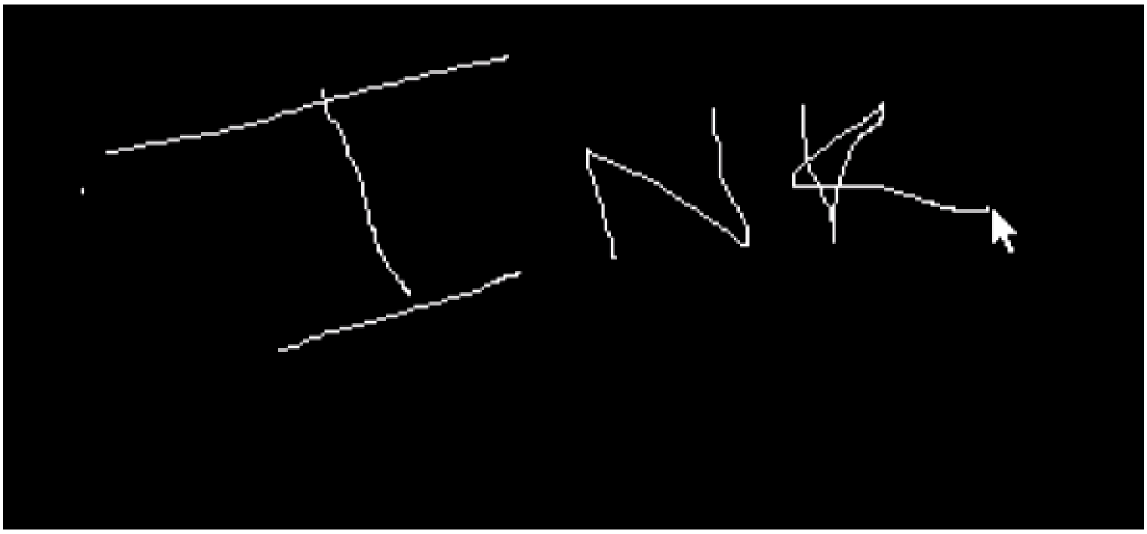
```
}
/***********************************************************/
/****** The two functions below convert a polar **************
*******point into a quadratic point ***********************/
float P2Q_x(float r, float phi)
{
return r * cos(phi);
}
float P2Q_y(float r, float phi)
{
return r * sin(phi);
}
/**********************************************************/
void main(void) {
/* Change the values of 'drv' and 'mode' if you change MAXX and MAXY */
int drv = VGA, mode = VGAHI;
float phi, r, xb, xe, yb, ye, x, y;
/* You can play with the parameters below to determine the viewable
  area of your graph */
xb = -3;
xe = 3;
yb = -3;
ye = 3;
/* Stop playing !!! */
initgraph(&drv, &mode, "C:\\TURBOC3\\BGI");
/* Draw the X and Y axes */
setcolor(15);
line( 0, screen_y(yb, 0, ye), MAXX, screen_y(yb, 0, ye) );
line( screen_x(xb, 0, xe), 0, screen_x(xb, 0, xe), MAXY );
moveto( screen_x(xb, 0, xe), screen_y(yb, 0, ye) );
for(phi = 0; phi < 2 * 3.1415; phi += 0.02)
{ r = sin(2 * phi) + 2 * cos(3 * phi);
x = P2Q_x(r, phi);
y = P2Q_y(r, phi);
lineto(screen_x(xb, x, xe), screen_y(yb, y, ye) );

}

getch();

closegraph();

return;

}
```



18. Interactive Graphical Technique: Inking/ Free Hand Drawing

```
#include <dos.h>

#include<conio.h> // for kbhit()

#include <graphics.h>

#include <stdio.h>

union REGS in, out;

// Function to initialize the mouse pointer

int initMouse()

{

in.x.ax = 0;

int86(0X33, &in, &out);

return out.x.ax;

}

// Function to display the mouse pointer

void showMouse()

{

in.x.ax = 1;
```

```
int86(0X33, &in, &out);

}

// Function to hide the mouse pointer

void hideMouse()

{

in.x.ax = 2;

int86(0X33, &in, &out);

}

// Function to get the exact mouse position

void getMousePosition(int* x, int* y, int* click)

{

in.x.ax = 3;

// Get the coordinates

int86(0x33, &in, &out);

// Update the coordinates

*x = out.x.cx;

*y = out.x.dx;

*click = out.x.bx & 1;

}

// Driver Code

int main()

{

int click, status, i, gd = DETECT, gm,x1,y1,x2,y2;

// Initialize graphics

initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

initMouse();

// kbhit If a key has been pressed then it returns a non zero value

// otherwise returns zero(false)

while (!kbhit()) {

// Show the mouse pointer

showMouse();

// Get the mouse position

getMousePosition(&x1, &y1, &click);

x2 = x1;
y2 = y1;

// When mouse is clicked

while (click == 1) {

hideMouse();

// Draw line

line(x1, y1, x2, y2);

x1 = x2;

y1 = y2;

// Get the mouse position

getMousePosition(&x2, &y2, &click);

}

}

getch();

closegraph(); // Close the graphics

return 0;

}
```



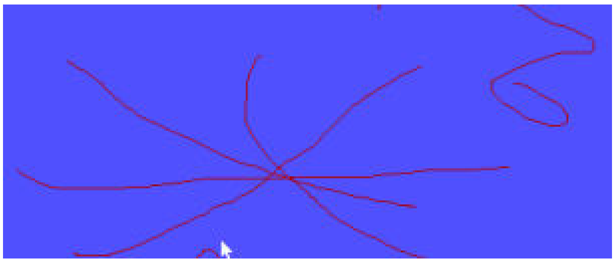
19. Interactive Graphical Technique: Inking/ Free Hand Drawing

```
#include <dos.h>

#include<conio.h>

#include <graphics.h>

#include <stdio.h>

union REGS in, out;

// Function to initialize the mouse pointer

int initMouse()
```

```
{
in.x.ax = 0;
int86(0X33, &in, &out);
return out.x.ax;
}
// Function to display the mouse pointer
void showMouse()
{
in.x.ax = 1;
int86(0X33, &in, &out);
}
// Function to hide the mouse pointer
void hideMouse()
{
in.x.ax = 2;
int86(0X33, &in, &out);
}
// Function to get the exact mouse position
void getMousePosition(int* x, int* y, int* click)
{
in.x.ax = 3;
// Get the coordinates
int86(0x33, &in, &out);
// Update the coordinates
*x = out.x.cx;
*y = out.x.dx;
*click = out.x.bx & 1;
}
// Driver Code
int main()
{
int click, gd = DETECT, gm, mx, my, x1,y1;
const int brushcolor = RED;
// Initialize graphics
```

```
initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

initMouse();

// Show the mouse pointer

showMouse();

setbkcolor(LIGHTGRAY);

// kbhit If a key has been pressed then it returns a non-zero value

// otherwise returns zero(false)

while (!kbhit()) {

// Get the mouse position

getMousePosition(&mousex, &mousey, &click);

if(click==1)

{

startx = mousex;

starty = mousey;

setcolor(brushcolor);

hideMouse();

putpixel(startx, starty, brushcolor); //start free hand drawing

showMouse();

// Till the mouse is kept left clicked

do{

getMousePosition(&mousex, &mousey, &click);

if(mousex != startx || mousey != starty)

{

hideMouse();

// Draw line for free hand drawing

line(startx, starty, mousex, mousey);

showMouse();

startx = mousex;

starty = mousey;

}

} while (click == 1);

} //end of if

} //end of while

getch();
```

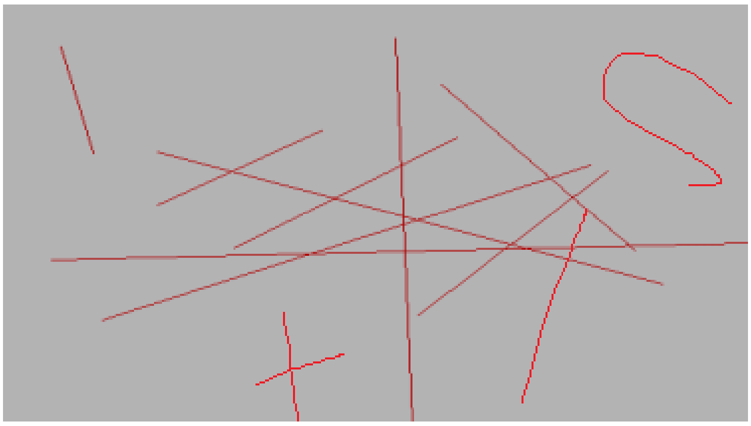
```
// Close the graphics
closegraph();
return 0;
}// end of main
```



20. Elastic or Rubber Band Interactive System to realize straight line

```
#include <dos.h>
#include<conio.h>
#include <graphics.h>
#include <stdio.h>
union REGS in, out;
// Function to initialize the mouse pointer
int initMouse()
{ in.x.ax = 0;
int86(0X33, &in, &out);
return out.x.ax;
}
// Function to display the mouse pointer
void showMouse()
{ in.x.ax = 1;
int86(0X33, &in, &out);
}
// Function to hide the mouse pointer
void hideMouse()
{ in.x.ax = 2;
int86(0X33, &in, &out);
}
// Function to get the exact mouse position
void getMousePosition(int* x, int* y, int* click)
```

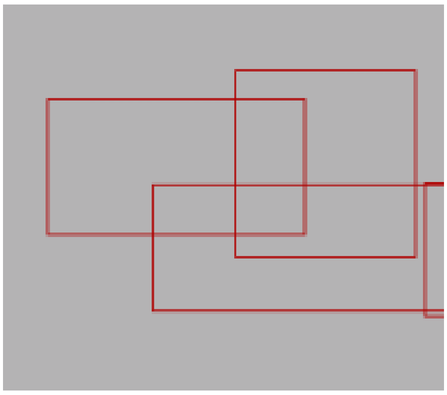
```
{ in.x.ax = 3;
// Get the coordinates
int86(0x33, &in, &out);
// Update the coordinates
*x = out.x.cx;
*y = out.x.dx;
*click = out.x.bx & 1;
}
// Driver Code
int main()
{
int gd = DETECT, gm;
int click, mousex, mousey, startx, starty, endx, endy;
const int brushcolor = RED;
// Initialize graphics
initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
setbkcolor(LIGHTGRAY);
// Initialize mouse and show the mouse pointer
initMouse();
showMouse();
// kbhit If a key has been pressed then it returns a non-zero value
// otherwise returns zero(false)
while (!kbhit()) {
// Get the mouse position
getMousePosition(&mousex, &mousey, &click);
if(click==1) //left click
{ //Anchor point on line
startx = endx = mousex;
starty = endy = mousey;
setwritemode(XOR_PUT); //to erase and redraw line
setcolor(WHITE-brushcolor);
line(startx, starty, endx, endy); //start drawing line
// till the mouse is kept left clicked
do{ //loop until left click released
getMousePosition(&mousex, &mousey, &click);
if(mousex !=endx || mousey !=endy)
{
hideMouse();
line(startx, starty, endx, endy); //erase previous line
endx = mousex; //new end point of line
endy = mousey;
line(startx, starty, endx, endy); //draw new line
showMouse();
}
} while (click == 1);
//left click released and therefore draw permanent line
setwritemode(COPY_PUT);
setcolor(brushcolor);
endx = mousex;
endy = mousey;
hideMouse();
// Draw Final line
line(startx, starty, endx, endy);
showMouse();
} //end of if
} //end of while
getch();
// Close the graphics
closegraph();
return 0;
}
```



21. Realizing Rubber Band Rectangle

```
#include <dos.h>

#include<conio.h>

#include <graphics.h>

#include <stdio.h>

union REGS in, out;

// Function to initialize the mouse pointer

int initMouse()

{

in.x.ax = 0;

int86(0X33, &in, &out);

return out.x.ax;

}

// Function to display the mouse pointer

void showMouse()

{

in.x.ax = 1;

int86(0X33, &in, &out);

}

// Function to hide the mouse pointer

void hideMouse()

{

in.x.ax = 2;

int86(0X33, &in, &out);

}

// Function to get the exact mouse position

void getMousePosition(int* x, int* y, int* click)

{

in.x.ax = 3;

// Get the coordinates

int86(0x33, &in, &out);

// Update the coordinates

*x = out.x.cx;
```

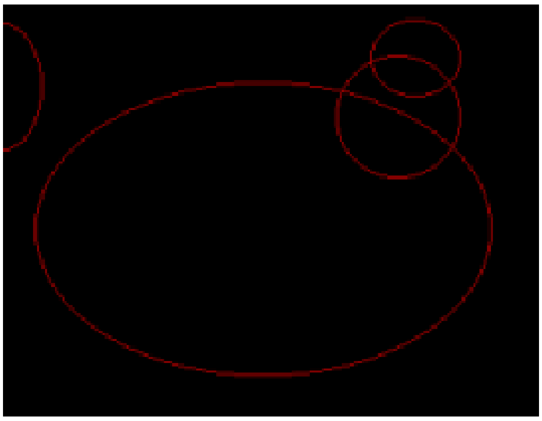
```
*y = out.x.dx;

*click = out.x.bx & 1;

}

// Driver Code

int main()

{

int gd = DETECT, gm;

int click, mousex, mousey, startx, starty, endx, endy;

const int brushcolor = RED;

// Initialize graphics

initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

setbkcolor(LIGHTGRAY);

// Initialize mouse and show the mouse pointer

initMouse();

showMouse();

// kbhit If a key has been pressed then it returns a non-zero value

// otherwise returns zero(false)

while (!kbhit()) {

// Get the mouse position

getMousePosition(&mousex, &mousey, &click);

if(click==1)

{

startx = endx = mousex;

starty = endy = mousey;

setwritemode(XOR_PUT);

setcolor(WHITE-brushcolor);

rectangle(startx, starty, endx, endy); //start drawing rectangle

// till the mouse is kept left clicked

do{

getMousePosition(&mousex, &mousey, &click);

if(mousex !=endx || mousey !=endy)

{

hideMouse();

rectangle(startx, starty, endx, endy); //erase previous
endx = mousex; //new end points
endy = mousey;
rectangle(startx, starty, endx, endy); //draw new
showMouse();
}
} while (click == 1);
//left click released
setwritemode(COPY_PUT);
setcolor(brushcolor);
endx = mousex;
endy = mousey;
hideMouse();
// Draw Final line
rectangle(startx, starty, endx, endy);
showMouse();
} //end of if
} //end of while
getch();
closegraph(); // Close the graphics
return 0;
}
```



22. Realizing Rubber Band Ellipse

```
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
```

```
#include<math.h>

#define PI 3.14159

union REGS in,out;

void initmouse(){ in.x.ax=0; int86(0x33,&in,&out);}

void showmouse(){ in.x.ax=1; int86(0x33,&in,&out);}

void hidemouse(){ in.x.ax=2; int86(0x33,&in,&out);}

void getmousepos(int &button,int &x,int &y) {

in.x.ax=3; int86(0x33,&in,&out);

button=out.x.bx; x=out.x.cx; y=out.x.dx;

}

const int colour = RED;

void myellipse(int cenx,int ceny,int xrad,int yrad)

{

float cx,cy,angle=0;

while(angle<360)

{

float THETA=PI/180.0 * angle;

cx=cenx+xrad*cos(THETA);

cy=ceny-yrad*sin(THETA);

line(cx,cy,cx,cy);

angle+=.5;

}

}

void rub_circle()

{

int button,mousex,mousey,startx,starty,dx,dy;

while(!kbhit())

{

getmousepos(button,mousex,mousey);

if(button & 1==1)

{ getmousepos(button,mousex,mousey);

setwritemode(XOR_PUT);

setcolor(WHITE-colour);

startx=mousex;
starty=mousey;
while((button & 1)==1)
{
getmousepos(button,mousex,mousey);
hidemouse();
dx=abs(mousex-startx); dy=abs(mousey-starty);
//draw
myellipse((startx+mousex)/2,(starty+mousey)/2,dx/2,dy/2);
//erase
myellipse((startx+mousex)/2,(starty+mousey)/2,dx/2,dy/2);
showmouse();
}
setwritemode(COPY_PUT);
setcolor(colour);
hidemouse();
//final draw
ellipse((startx+mousex)/2,(starty+mousey)/2,0,360,dx/2,dy/2);
showmouse();
} //end of if
showmouse();
} //end of while
} //end of rubber band ellipse
void main()
{
int gdriver = DETECT, gmode;
initgraph(&gdriver, &gmode, "C:\\turboc3\\bgi");
initmouse(); showmouse();
setcolor(BLACK);
rub_circle();
getch();
closegraph();
}
```

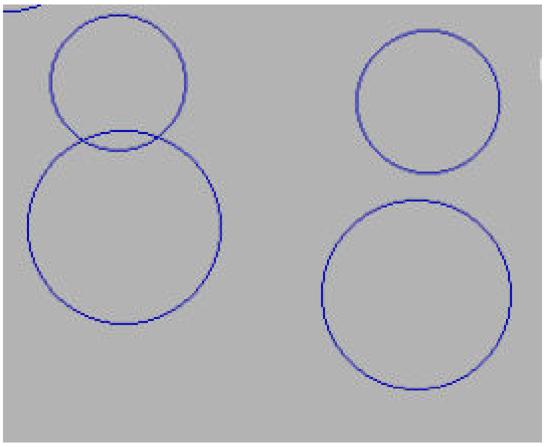


23. Realizing Rubber Band Circle

```
#include<dos.h>
#include<conio.h>
#include<graphics.h>
#include<stdio.h>
#include<math.h>
union REGS in, out;
#define PI 3.14159
// Function to initialize the mouse pointer
int initMouse()
{
in.x.ax = 0;
int86(0X33, &in, &out);
return out.x.ax;
}
// Function to display the mouse pointer
void showMouse()
{
in.x.ax = 1;
int86(0X33, &in, &out);
}
// Function to hide the mouse pointer
void hideMouse()
{
in.x.ax = 2;
int86(0X33, &in, &out);
```

```
}
// Function to get the exact mouse position
void getMousePosition(int* x, int* y, int* click)
{
in.x.ax = 3;
// Get the coordinates
int86(0x33, &in, &out);
// Update the coordinates
*x = out.x.cx;
*y = out.x.dx;
*click = out.x.bx & 1;
}
void mycircle(int cenx,int ceny,int radius)
{
float cx,cy,angle=0;
while(angle<360)
{
float THETA=PI/180.0 * angle;
cx=cenx+radius*cos(THETA);
cy=ceny-radius*sin(THETA);
line(cx,cy,cx,cy);
angle+=.5;
}
}
// Driver Code
int main()
{
int gd = DETECT, gm;
int click, mousex, mousey, startx, starty, endx, endy;
int dx, dy, centrex, centrey, diameter, radius;
const int brushcolor = BLUE;
// Initialize graphics
initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
setbkcolor(LIGHTGRAY);
```

```
// Initialize mouse and show the mouse pointer
initMouse();
showMouse();
// kbhit If a key has been pressed then it returns a non zero value
// otherwise returns zero(false)
while (!kbhit()) {
// Get the mouse position
getMousePosition(&mousex, &mousey, &click);
if(click==1)
{
startx = endx = mousex;
starty = endy = mousey;
setwritemode(XOR_PUT);
setcolor(WHITE-brushcolor);
// till the mouse is kept left clicked
do{
getMousePosition(&mousex, &mousey, &click);
//if(mousex !=endx || mousey !=endy)
//{
  endx=mousex; endy=mousey;
  dx=abs(startx-endx); dy=abs(starty-endy);
  // (centrex, centrey) is the centre
  centrex = (startx+endx)/2;
  centrey = (starty+endy)/2;
  diameter = (int)(sqrt(dx * dx + dy * dy)); //x^2 + y^2 = r^2
  radius = diameter/2;
  hideMouse();
  mycircle(centrex, centrey, radius); //draw circle
  mycircle(centrex, centrey, radius); //erase circle
  showMouse();
//}//end of if
} while (click == 1);
//left click released
setwritemode(COPY_PUT);
setcolor(brushcolor);

hideMouse();

// Draw Final Circle

circle(centrex, centrey, radius);

showMouse();

} //end of if

} //end of while

getch();

// Close the graphics

closegraph();

return 0;

}
```



24. Paint System Simulator

```
#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <dos.h>

#include <math.h>

#include <graphics.h>

#define PI 3.14159265

#define TRUE 1

union REGS in,out;

//mouse interrupt with functions 0, 1, 2, 3, 7

void initmouse() {

in.x.ax=0;
```

```
int86(0x33,&in,&out);

}

void showmouse() {

in.x.ax=1;

int86(0x33,&in,&out);

}

void hidemouse() {

in.x.ax=2;

int86(0x33,&in,&out);

}

void getmousepos(int &button,int &x,int &y) {

in.x.ax=3;

int86(0x33,&in,&out);

button=out.x.bx;

x=out.x.cx;

y=out.x.dx;

}

void restrictmouse(int x1,int y1,int x2,int y2) {

in.x.ax=7;

in.x.cx=x1;

in.x.dx=x2;

int86(0x33,&in,&out);

in.x.ax=8;

in.x.cx=y1;

in.x.dx=y2;

int86(0x33,&in,&out);

}

//(52,30,628,418) is the drawing area with lightgray background

int posx, posy, control_button, colx,coly, col_button; //global variables

void selectcolour(int, int, int); //prototype

int colour, col2; //global variables in selectcolour()

//save image in picture.dat in /bin directory of Turbo-C++

void save_image(){

//setting impression of all button presses
```

```
for(int k=80;k<212;k+=21) {
setcolor(BLACK);
rectangle(5,k,25,k+20); rectangle(26,k,46,k+20);
setcolor(WHITE);
line(5,k,5,k+20); line(5,k,25,k);
line(26,k,26,k+20); line(26,k,46,k);
}
//giving impression of save button press
setcolor(BLACK);
line(5,80,5,100); line(5,80,25,80);
setcolor(WHITE);
line(25,100,25,80); line(25,100,5,100);
hidemouse();
FILE *fp=fopen("picture.dat","wb");
for(int x=52+1;x<=628-1;x++){
for(int y=30+1;y<=418-1;y++)
{
int cr=getpixel(x,y);
fwrite(&cr,2,1,fp);
}
}
fclose(fp);
showmouse();
} //end of save_image()
//load image file picture.dat from /bin
void load_image() {
//setting impression of all button press
for(int k=80;k<212;k+=21) {
setcolor(BLACK);
rectangle(5,k,25,k+20); rectangle(26,k,46,k+20);
setcolor(WHITE);
line(5,k,5,k+20); line(5,k,25,k);
line(26,k,26,k+20); line(26,k,46,k);
}
```

```
//giving impression of load button press

setcolor(BLACK);

line(26,80,26,100); line(26,80,46,80);

setcolor(WHITE);

line(46,100,46,80); line(46,100,26,100);

FILE *fp=fopen("picture.dat","rb");

int cr;

for(int x=52+1;x<=628-1;x++){

for(int y=30+1;y<=418-1;y++)

{

fread(&cr,2,1,fp);

putpixel(x,y,cr);

}

}

fclose(fp);

}//end of load_image()

//create pencil button for pencil();

void *pb; //global

void createpb() {

//filled lightgray rectangle for backgroud of buttons

setcolor(LIGHTGRAY);

rectangle(3,3,300,300);

setfillstyle(SOLID_FILL,LIGHTGRAY);

floodfill(40,40,LIGHTGRAY);

//pencil from end to before nose

setcolor(RED);

line(60,50,50,57); line(63,54,53,61);

line(50,57,53,61); line(60,50,63,54);

setfillstyle(SOLID_FILL,RED);

floodfill(60,52,RED);

//nose of pencil

setcolor(BLACK);

line(50,57,47,62); line(47,62,53,61);

line(60,50,50,57); line(63,54,53,61);
line(50,57,53,61); line(60,50,63,54);

int size=imagesize(45,48,65,65);

pb= new int [size];

getimage(45,48,65,65,pb);

cleardevice();

} // end of createpb()

void *eraser; //create eraser button for rub()

void createeraser() {

setcolor(LIGHTGRAY);

rectangle(10,10,200,200);

setfillstyle(SOLID_FILL,LIGHTGRAY);

floodfill(50,50,LIGHTGRAY);

setcolor(RED);

rectangle(30,112,37,115);

line(30,112,35,105); line(35,105,42,105);

line(42,105,37,112); line(42,105,42,109);

line(42,109,37,115);

setfillstyle(SOLID_FILL,RED);

floodfill(33,110,RED); floodfill(40,110,RED);

setcolor(WHITE);

rectangle(30,112,37,115);

setfillstyle(SOLID_FILL,WHITE);

floodfill(33,113,WHITE);

int size = imagesize(28,102,45,118);

eraser = new int [size];

getimage(28,102,45,118,eraser);

cleardevice();

} //end of createeraser()

//create fill button for getcolour()

void *fill;

void createfill() {

setcolor(LIGHTGRAY);

rectangle(3,3,300,300);

setfillstyle(SOLID_FILL,LIGHTGRAY);
```

```
floodfill(40,40,LIGHTGRAY);

setcolor(BLACK);

line(50,57,47,62); line(47,62,53,61);

line(60,50,50,57); line(63,54,53,61);

line(60,50,63,54);

int size=imagesize(45,48,65,65);

fill= new int [size];

getimage(45,48,65,65,fill);

cleardevice();

} //end of createfill()

//graphical user interface

void gui() {

int i,x=getmaxx(),y=getmaxy();

createpb();

createfill();

createeraser();

putimage(6,102,pb,COPY_PUT); //pencil button

putimage(6,123,fill,COPY_PUT); //colour filler button

putimage(27,103,eraser,COPY_PUT); //eraser button

setcolor(BLUE);

rectangle(0,0,x,15); //top blue

setfillstyle(SOLID_FILL,BLUE);

floodfill(10,10,BLUE);

//plain yellow square/sheet on top blue

setcolor(YELLOW);

rectangle(5,3,12,12);

setfillstyle(SOLID_FILL,YELLOW);

floodfill(9,9,YELLOW);

settextstyle(12,0,5);

outtextxy(20,4,"Paint Simulator");

setcolor(LIGHTGRAY);

rectangle(0,15,x-1,30); //top

rectangle(0,30,50,y); // left

rectangle(50,y-60,x-1,y); // bottom
rectangle(x-1,30,x-10,y-60); // right
setfillstyle(SOLID_FILL,LIGHTGRAY); // fill above borders
floodfill(20,20,LIGHTGRAY);
floodfill(40,60,LIGHTGRAY);
floodfill(60,y-20,LIGHTGRAY);
floodfill(x-5,y-100,LIGHTGRAY);
setcolor(BLACK);
line(0,31,50,31);
line(50,y-61,0,y-61);
line(x-1,31,x-10,31);
line(x-1,y-61,x-10,y-61);
line(0,y-15,x-1,y-15); // filler lines
line(x-100,y-15,x-100,y);
line(x-200,y-15,x-200,y);
rectangle(10,y-50,37,y-23); // Outer box of selected colour
setcolor(WHITE);
line(37,y-23,37,y-50);
line(37,y-23,10,y-23);
//16 color boxes at the bottom
setcolor(BLACK);
int k,c=0;
for(k=40;k<157;k+=15) {
setcolor(c);
rectangle(k,y-50,k+12,y-38);
setfillstyle(SOLID_FILL,c);
floodfill(k+5,y-40,c);
setcolor(BLACK);
line(k,y-50,k,y-38);//drawing and filling of colour boxes(1-8)
line(k,y-50,k+12,y-50);
setcolor(WHITE);
line(k+12,y-38,k+12,y-50); line(k+12,y-38,k,y-38);
c++;
}
for(k=40;k<157;k+=15) {
```

```
setcolor(c);

rectangle(k,y-35,k+12,y-23);

setfillstyle(SOLID_FILL,c);

floodfill(k+5,y-25,c);

setcolor(BLACK);

line(k,y-35,k,y-23); line(k,y-35,k+12,y-35);

setcolor(WHITE); //drawing and filling of colour boxes(9-16)

line(k+12,y-23,k+12,y-35); line(k+12,y-23,k,y-23);

c++;

}

setcolor(WHITE);

rectangle(52,32,x-12,y-62);

setfillstyle(SOLID_FILL,WHITE); //Drawing area (52,32) to (628,418)

floodfill(200,200,WHITE);

setcolor(BLACK);

for(k=80;k<212;k+=21) {

rectangle(5,k,25,k+20);

rectangle(26,k,46,k+20);

setcolor(WHITE);

line(5,k,5,k+20);

line(5,k,25,k); // Function Buttons

line(26,k,26,k+20);

line(26,k,46,k);

setcolor(BLACK);

}

rectangle(8,k+8,43,k+68); // button option box

setcolor(WHITE);

line(43,k+68,8,k+68); line(43,k+68,43,k+8);

setcolor(BLACK);

rectangle(13,y-45,25,y-33); // selected foreground colour box

setfillstyle(SOLID_FILL,BLACK);

floodfill(15,y-40,BLACK);

setcolor(WHITE);

line(26,y-40,32,y-40);
line(20,y-28,20,y-32);

line(32,y-40,32,y-28);

line(32,y-28,20,y-28); //selected background colour box

line(26,y-40,26,y-33);

line(26,y-32,20,y-32);

setfillstyle(SOLID_FILL,WHITE);

floodfill(30,y-37,WHITE);

line(13,y-45,13,y-33); line(13,y-45,25,y-45);

setcolor(BLACK);

line(26,y-40,32,y-40); line(20,y-28,20,y-32);

setcolor(LIGHTGRAY);

rectangle(x-16,2,x-5,12); // close icon

setfillstyle(SOLID_FILL,LIGHTGRAY);

floodfill(x-10,8,LIGHTGRAY);

setcolor(BLACK);

line(x-13,4,x-7,10); line(x-13,10,x-7,4);

line(x-5,12,x-5,2); line(x-5,12,x-16,12);

setcolor(WHITE);

line(x-16,2,x-16,12); line(x-16,2,x-5,2);

setcolor(BLACK);

ellipse(15,153,0,360,8,4); //ellipse button

rectangle(9,168,22,178); // rectangle button

line(9,190,20,201); //line button

// For polygon button

line(33,127,30,136); line(30,136,39,136);

line(33,127,38,127); line(38,127,36,132);

line(36,132,40,132); line(40,132,39,136);

// For Paint Brush button

line(35,145,35,151); line(38,145,38,151);

line(35,145,38,145); line(35,151,32,153);

line(38,151,41,153); line(32,153,32,159);

line(41,153,41,159); line(32,159,41,159);

line(41,154,32,154); line(41,155,32,155);

line(35,159,35,157); line(38,159,38,157);
```

```
// For curved line button
ellipse(32,173,180,0,4,2);
ellipse(39,173,0,180,4,2);
// For bucket button for fill
line(32,200,40,200); line(32,200,32,193);
line(40,200,40,193); line(32,193,40,193);
line(36,193,38,189); line(38,193,40,189);
line(38,189,40,190); line(37,193,39,189);
line(32,195,40,195); line(32,196,40,196);
// For spray can button
line(7,223,7,215); line(7,215,12,215);
line(12,215,12,223); line(12,223,7,223);
line(8,215,8,213); line(11,215,11,213);
line(11,213,8,213); line(11,213,19,210);
line(11,213,19,216); line(11,213,19,215);
line(11,213,19,213); line(11,213,19,211);
//For text button
settextstyle(1,0,1);
outtextxy(30,203,"A");
//buttons for save and load
setusercharsize(7,25,7,25);//more than size 1/4 and less than 1/3
outtextxy(7, 83,"Save");
outtextxy(28, 83,"Load");
settextstyle(12,0,5);
outtextxy(8,y-10,"DCSA - Dr R K Singla");
} //end of gui()
//pencil: free hand drawing after left-clicking on pencil icon
void pencil() {
//setting impression of all button press
for(int k=80;k<212;k+=21) {
setcolor(BLACK);
rectangle(5,k,25,k+20); rectangle(26,k,46,k+20);
setcolor(WHITE);
line(5,k,5,k+20); line(5,k,25,k);
line(26,k,26,k+20); line(26,k,46,k);

}

//giving impression of pencil button press

setcolor(BLACK);

line(5,101,5,121); line(5,101,25,101);

setcolor(WHITE);

line(25,121,25,101); line(25,121,5,121);

int button,x,y,prevx,prevy; //local

while(TRUE) {

getmousepos(button,x,y);

if((button & 1==1 && x>5 && x<46 && y>80 && y<225)||(button &

1==1 && x>625 && x<635 && y>2 && y<12)) {

control_button=1;

posx=x; posy=y;

break;

}

if(button && x>=40 && x<=157 && y>=430 && y<=457){

if(button==1) col_button=1;

if(button==2) col_button=2;

colx=x; coly=y;

selectcolour(col_button,colx,coly);

}

setcolor(colour);

if(button & 1 ==1 && x>52 && x<628 && y>32 && y<418){

prevx=x; prevy=y;

while((button & 1) ==1){

restrictmouse(52,32,627,417);

hidemouse();

line(prevx,prevy,x,y);

showmouse();

prevx=x; prevy=y;

getmousepos(button,x,y);

}

restrictmouse(0,0,getmaxx(),getmaxy());
```

```
}
}
} // end of pencil()
void getcolour(){
for(int k=80;k<212;k+=21){
setcolor(BLACK);
rectangle(5,k,25,k+20); rectangle(26,k,46,k+20);
setcolor(WHITE);
line(5,k,5,k+20); line(5,k,25,k);
line(26,k,26,k+20); line(26,k,46,k);
}
setcolor(BLACK);
line(5,122,25,122); line(5,122,5,142);
setcolor(WHITE);
line(25,142,25,122); line(25,142,5,142);
int button,x,y,prevx,prevy,gcolour,e,f,u; //local
k=226;
while(!kbhit()){
getmousepos(button,x,y);
if((button & 1==1 && x>5 && x<46 && y>80 && y<225)||(button &
1==1 && x>625 && x<635 && y>2 && y<12)){
control_button=1;
posx=x;
posy=y;
break;
}
if(button & 1==1 && x>52 && x<627 && y>32 && y<417){
while(button & 1==1){
restrictmouse(52,32,627,417);
hidemouse();
gcolour=getpixel(x,y);
showmouse();
setcolor(gcolour);
for(e=9;e<43;e++)
for(f=k+9;f<k+68;f++)

putpixel(e,f,gcolour);

getmousepos(button,x,y);

}

for(e=9;e<43;e++)

for(f=k+9;f<k+68;f++)

putpixel(e,f,LIGHTGRAY);

colour=gcolour;

restrictmouse(0,0,getmaxx(),getmaxy());

colour=gcolour;

u=getmaxy();

rectangle(13,u-45,25,u-33); // selected foreground colour box

setfillstyle(SOLID_FILL,colour);

floodfill(15,u-40,colour);

setcolor(BLACK);

line(25,u-33,25,u-45); line(25,u-33,13,u-33);

setcolor(WHITE);

line(13,u-45,25,u-45); line(13,u-45,13,u-33);

} //end of if

} //end of while

}//end of getcolour()

//int colour, col2; //global vars in selectcolour() already defined above

void selectcolour(int col_button, int colx, int coly) {

if(colx>40 && colx<52 && coly>430 && coly<442) {

if(col_button==2) col2=0;

if(col_button==1) colour=0;

}

if(colx>55 && colx<67 && coly>430 && coly<442) {

if(col_button==2) col2=1;

if(col_button==1) colour=1;

}

if(colx>70 && colx<82 && coly>430 && coly<442) {

if(col_button==2) col2=2;

if(col_button==1) colour=2;
```

```
}

if(colx>85 && colx<97 && coly>430 && coly<442) {

if(col_button==2) col2=3;

if(col_button==1) colour=3;

}

if(colx>100 && colx<112 && coly>430 && coly<442) {

if(col_button==2) col2=4;

if(col_button==1) colour=4;

}

if(colx>115 && colx<127 && coly>430 && coly<442) {

if(col_button==2) col2=5;

if(col_button==1) colour=5;

}

if(colx>130 && colx<142 && coly>430 && coly<442) {

if(col_button==2) col2=6;

if(col_button==1) colour=6;

}

if(colx>145 && colx<157 && coly>430 && coly<442) {

if(col_button==2) col2=7;

if(col_button==1) colour=7;

}

if(colx>40 && colx<52 && coly>445 && coly<457) {

if(col_button==2) col2=8;

if(col_button==1) colour=8;

}

if(colx>55 && colx<67 && coly>445 && coly<457) {

if(col_button==2) col2=9;

if(col_button==1) colour=9;

}

if(colx>70 && colx<82 && coly>445 && coly<457) {

if(col_button==2) col2=10;

if(col_button==1) colour=10;

}

if(colx>85 && colx<97 && coly>445 && coly<457) {
```

```
if(col_button==2) col2=11;

if(col_button==1) colour=11;

}

if(colx>100 && colx<112 && coly>445 && coly<457) {

if(col_button==2) col2=12;

if(col_button==1) colour=12;

}

if(colx>115 && colx<127 && coly>445 && coly<457) {

if(col_button==2) col2=13;

if(col_button==1) colour=13;

}

if(colx>130 && colx<142 && coly>445 && coly<457) {

if(col_button==2) col2=14;

if(col_button==1) colour=14;

}

if(colx>145 && colx<157 && coly>445 && coly<457) {

if(col_button==2) col2=15;

if(col_button==1) colour=15;

}

int y=getmaxy();

if(col_button==1) {

setcolor(colour);

rectangle(13,y-45,25,y-33); // selected foreground colour box

setfillstyle(SOLID_FILL,colour);

floodfill(15,y-40,colour);

setcolor(BLACK);

line(25,y-33,25,y-45); line(25,y-33,13,y-33);

} //****************

//Right Click

if(col_button==2) {

setcolor(col2);

line(26,y-40,32,y-40);

line(20,y-28,20,y-32);

line(32,y-40,32,y-28);
line(32,y-28,20,y-28); //selected background colour box

line(26,y-40,26,y-32);

line(26,y-32,20,y-32);

setfillstyle(SOLID_FILL,col2);

floodfill(30,y-37,col2);

setcolor(WHITE);

line(32,y-40,32,y-28); line(32,y-28,20,y-28);

setcolor(BLACK);

line(26,y-40,32,y-40); line(20,y-28,20,y-32);

}

} // end of selectcolor()

int check_rub=0; //global

struct point {

int xc;

int yc;

};

point p1,p2,p3,p4; //global

void four_pt_bez(point p1,point p2,point p3,point p4) {

int x0,y0,x1,y1,x2,y2,x3,y3,vx,vy,x01,y01,x12,y12,x23,y23;

int x012,y012,x123,y123,x0123,y0123,x,y;

x0=p1.xc; y0=p1.yc;

x3=p2.xc; y3=p2.yc;

x1=p3.xc; y1=p3.yc;

x2=p4.xc; y2=p4.yc;

x01=x1-x0; y01=y1-y0;

x12=x2-x1; y12=y2-y1;

x23=x3-x2; y23=y3-y2;

x012=x12-x01; y012=y12-y01;

x123=x23-x12; y123=y23-y12;

x0123=x123-x012; y0123=y123-y012;

for(float t=0;t<1;t+=0.0001) {

x=t*t*t*x0123+3*t*t*x012+3*t*x01+x0;

y=t*t*t*y0123+3*t*t*y012+3*t*y01+y0;

if(x>52 && x<628 && y>32 && y<417) line(x,y,x,y);
```

```
}
} //end of four_pt_bez()
void thr_pt_bez(point p1,point p2,point p3) {
int x0,y0,x1,y1,x2,y2,x3,y3,vx,vy,x01,y01,x12,y12,x23,y23;
int x012,y012,x123,y123,x0123,y0123,x,y;
x0=p1.xc; y0=p1.yc;
x3=p2.xc; y3=p2.yc;
x1=x2=p3.xc; y1=y2=p3.yc;
x01=x1-x0; y01=y1-y0;
x12=x2-x1; y12=y2-y1;
x23=x3-x2; y23=y3-y2;
x012=x12-x01; y012=y12-y01;
x123=x23-x12; y123=y23-y12;
x0123=x123-x012; y0123=y123-y012;
for(float t=0;t<1;t+=0.0001) {
x=t*t*t*x0123+3*t*t*x012+3*t*x01+x0;
y=t*t*t*y0123+3*t*t*y012+3*t*y01+y0;
if(x>52 && x<628 && y>32 && y<417) line(x,y,x,y);
}
} //end tr_pt_bez()
void bez2();
void bez() {
int button1 ,x1,y1,k=0;
while(!k) {
getmousepos(button1,x1,y1);
if(button1==1) {
setwritemode(XOR_PUT);
line(p1.xc,p1.yc,p2.xc,p2.yc);
while(button1==1) {
// restrictmouse(0,0,getmaxx(),getmaxy());
setcolor(15-colour);
p3.xc=x1;
p3.yc=y1;
hidemouse();
```

```
thr_pt_bez(p1,p2,p3);

thr_pt_bez(p1,p2,p3);

showmouse();

getmousepos(button1,x1,y1);

}

// restrictmouse(52,32,627,417);

// setcolor(colour);

hidemouse();

thr_pt_bez(p1,p2,p3);

k=1;

showmouse();

bez2();

}

}

} //end of bez()

void bez2() {

int button1,x1,y1,k=0;

while(!k) {

showmouse();

getmousepos(button1,x1,y1);

if(button1==1) {

// setcolor(0);

setwritemode(XOR_PUT);

thr_pt_bez(p1,p2,p3);

setwritemode(XOR_PUT);

while(button1==1) {

// restrictmouse(0,0,getmaxx(),getmaxy());

p4.xc=x1;

p4.yc=y1;

setcolor(15-colour);

hidemouse();

four_pt_bez(p1,p2,p3,p4);

four_pt_bez(p1,p2,p3,p4);

showmouse();
```

```
getmousepos(button1,x1,y1);
}
// restrictmouse(0,0,getmaxx(),getmaxy());
setwritemode(COPY_PUT);
setcolor(colour);
hidemouse();
four_pt_bez(p1,p2,p3,p4);
k=1;
showmouse();
}
}
} //end of bez2()
void curve(){
int button,x,y,prevx,prevy;
for(int k=80;k<212;k+=21){
setcolor(BLACK);
rectangle(5,k,25,k+20);
rectangle(26,k,46,k+20);
setcolor(WHITE);
line(5,k,5,k+20); line(5,k,25,k);
line(26,k,26,k+20); line(26,k,46,k);
}
setcolor(BLACK);
line(26,164,26,184); line(26,164,46,184);
setcolor(WHITE);
line(46,184,46,164); line(46,184,26,184);
while(TRUE){
getmousepos(button,x,y);
if((button & 1==1 && x>5 && x<46 && y>80 && y<225)||(button &
1==1 && x>625 && x<635 && y>2 && y<12)){
control_button=1;
posx=x; posy=y;
break;
}
```

```
if(button && x>=40 && x<=157 && y>=430 && y<=457){

if(button==1) col_button=1;

if(button==2) col_button=2;

colx=x; coly=y;

selectcolour(col_button,colx,coly);

}

if(button & 1 == 1 && x>52 && x<628 && y>32 && y<418){

prevx=x; prevy=y;

setwritemode(XOR_PUT);

while((button & 1) == 1 && x>120){

restrictmouse(52,32,627,417);

setcolor(15-colour);

hidemouse();

line(prevx,prevy,x,y);

line(prevx,prevy,x,y);

showmouse();

getmousepos(button,x,y);

}

restrictmouse(0,0,getmaxx(),getmaxy());

hidemouse();

// setcolor(colour);

line(prevx,prevy,x,y);

showmouse();

p1.xc=prevx; p1.yc=prevy;

p2.xc=x; p2.yc=y;

bez();

} //end of if

} //end of while

} //end of curve()

struct coordinate {

int x,y; //structure

coordinate *next;

};

coordinate *control,*last,*temp; //global pointer varables holding coordinate
```

```
void insert(int x,int y) { //data type

coordinate *new_coord;

new_coord=new coordinate;

new_coord->x=x;

new_coord->y=y; //insert link function

new_coord->next=NULL;

last->next=new_coord;

last=new_coord;

}//end of insert()

void bucket(int x,int y,int backcolour,int colour){

if(backcolour==colour)return;

last=control=new coordinate; //last & control points to new blocks of memory

control->x=x; // large enough to store a coordinate variable

control->y=y;

control->next=NULL; //Null pointer value

while(control!=NULL){

putpixel(x,y,colour);

if(y-1>=32 && getpixel(x,y-1)==backcolour){

putpixel(x,y-1,colour);

insert(x,y-1);

}

if(x+1<628 && getpixel(x+1,y)==backcolour){

putpixel(x+1,y,colour);

insert(x+1,y);

}

if(y+1<418 && getpixel(x,y+1)==backcolour){

putpixel(x,y+1,colour);

insert(x,y+1);

}

if(x-1>=52 && getpixel(x-1,y)==backcolour){

putpixel(x-1,y,colour);

insert(x-1,y);

}

temp=control;
```

```
control=temp->next; //change the control to next link

delete temp;

x=control->x;

y=control->y;

} //end of while

} //end of bucket()

void ffill(){

int button,x,y,backcolour;

for(int k=80;k<212;k+=21){

setcolor(BLACK);

rectangle(5,k,25,k+20); rectangle(26,k,46,k+20);

setcolor(WHITE);

line(5,k,5,k+20); line(5,k,25,k);

line(26,k,26,k+20); line(26,k,46,k);

}

setcolor(BLACK);

line(26,185,26,205); line(26,185,46,185);

setcolor(WHITE);

line(46,205,46,185); line(46,205,26,205);

while(TRUE){

getmousepos(button,x,y);

if((button & 1==1 && x>5 && x<46 && y>80 && y<225)||(button &

1==1 && x>625 && x<635 && y>2 && y<12)){

control_button=1;

posx=x; posy=y;

break;

}

if(button && x>=40 && x<=157 && y>=430 && y<=457){

if(button==1)col_button=1;

if(button==2)col_button=2;

colx=x;

coly=y;

selectcolour(col_button,colx,coly);

}
```

```
if(button & 1 ==1 && x>52 && x<628 && y>32 && y<418){
hidemouse();
backcolour=getpixel(x,y);
bucket(x,y,backcolour,colour);
}
showmouse();
} //end of while
} //end of ffill()
void text(){
int button,x,y,m,n,col[200],v,u,g=0,i,j;
char ch,a[100];
void *bk;
for(int k=80;k<212;k+=21){
setcolor(BLACK);
rectangle(5,k,25,k+20); rectangle(26,k,46,k+20);
setcolor(WHITE);
line(5,k,5,k+20); line(5,k,25,k);
line(26,k,26,k+20); line(26,k,46,k);
}
setcolor(BLACK);
line(26,206,26,226); line(26,206,46,206);
setcolor(WHITE);
line(46,226,46,206); line(46,226,26,226);
while(TRUE){
getmousepos(button,x,y);
if((button & 1==1 && x>5 && x<46 && y>80 && y<225)||(button &
1==1 && x>625 && x<635 && y>2 && y<12)){
control_button=1;
posx=x; posy=y;
break;
}
if(button && x>=40 && x<=157 && y>=430 && y<=457){
if(button==1) col_button=1;
if(button==2) col_button=2;
```

```
colx=x; coly=y;
selectcolour(col_button,colx,coly);
}
if(button & 1==1 && x>52 && x<627 && y>32 && y<417){
u=m=x;
n=y;
setcolor(colour);
hidemouse();
settextstyle(0,0,1);
for(int i=0;i<100;i++)
a[i]=' ';
a[0]='_';
g=0;
v=imagesize(m,n,getmaxx()-10,n+9);
bk=new int[v];
getimage(m,n,getmaxx()-10,n+9,bk);
outtextxy(m,n,a);
while((ch=getch())!=13){
if(ch==8) { //backspace
if(g>0){
putimage(m,n,bk,COPY_PUT);
a[g]=' ';
a[g-1]='_';
outtextxy(m,n,a);
g--;
u-=8;
}
}
else
{
if(u<(getmaxx()-24)){
putimage(m,n,bk,COPY_PUT);
a[g]=ch;
a[g+1]='_';
g++;
u+=8;
outtextxy(m,n,a);
}
else
{
putimage(m,n,bk,COPY_PUT);
a[g-1]=ch;
a[g]=' '; //if it reaches the end
outtextxy(m,n,a);
}
}//else
}// end of while
a[g]=' ';
putimage(m,n,bk,COPY_PUT);
outtextxy(m,n,a);
showmouse();
} //end of if
} //end of while
}//end of text()
void poly(){
X: int button,x,y,prevx,prevy,j=0,orix,oriy;
for(int k=80;k<212;k+=21){
setcolor(BLACK);
rectangle(5,k,25,k+20); rectangle(26,k,46,k+20);
setcolor(WHITE);
line(5,k,5,k+20); line(5,k,25,k);
line(26,k,26,k+20); line(26,k,46,k);
}
setcolor(BLACK);
line(26,122,46,122); line(26,122,26,142);
setcolor(WHITE);
line(46,122,26,122); line(46,122,46,142);
while(TRUE){
```

```
getmousepos(button,x,y);

if((button & 1==1 && x>5 && x<46 && y>80 && y<225)||(button &

1==1 && x>625 && x<635 && y>2 && y<12)){

control_button=1;

posx=x; posy=y;

break;

}

if(button && x>=40 && x<=157 && y>=430 && y<=457){

if(button==1) col_button=1;

if(button==2) col_button=2;

colx=x; coly=y;

selectcolour(col_button,colx,coly);

}

if(button==1 && x<orix+2 && x>orix-2 && y<oriy+2 && y>oriy-2){

hidemouse();

setcolor(colour);

line(orix,oriy,prevx,prevy);

showmouse();

goto X;

}

if(button==1 && x>=100 && x<=120 && y>=140 && y<=180)break;

if(button & 1 ==1 && x>52 && x<627 && y>32 && y<417){

hidemouse();

if(j==0){ orix=prevx=x; oriy=prevy=y; }

setwritemode(XOR_PUT);

while((button & 1) ==1){

restrictmouse(52,32,627,417);

if(j==0) {

getmousepos(button,x,y);

prevx=x; prevy=y;

j=1;

}

getmousepos(button,x,y);

setcolor(15-colour);
hidemouse();

line(prevx,prevy,x,y); line(prevx,prevy,x,y);

showmouse();

}

restrictmouse(0,0,640,480);

setcolor(colour);

setwritemode(COPY_PUT);

if(x>52&&x<627&&y>32&&y<417){

hidemouse();

line(prevx,prevy,x,y);

showmouse();

}

prevx=x; prevy=y;

}//end of if

showmouse();

} //end of while

} //end of poly()

void spray(){

for(int k=80;k<212;k+=21){

setcolor(BLACK);

rectangle(5,k,25,k+20); rectangle(26,k,46,k+20);

setcolor(WHITE);

line(5,k,5,k+20); line(5,k,25,k);

line(26,k,26,k+20); line(26,k,46,k);

}

setcolor(BLACK);

line(5,206,25,206); line(5,206,5,226);

setcolor(WHITE);

line(25,226,25,206); line(25,226,5,226);

int button,x,y,prevx,prevy;

while(TRUE){

getmousepos(button,x,y);

if((button & 1==1 && x>5 && x<46 && y>80 && y<225)||(button &

1==1 && x>625 && x<635 && y>2 && y<12)){
```

```
control_button=1;

posx=x; posy=y;

break;

}

if(button && x>=40 && x<=157 && y>=430 && y<=457){

if(button==1) col_button=1;

if(button==2) col_button=2;

colx=x; coly=y;

selectcolour(col_button,colx,coly);

}

if(button & 1==1 && x>52 && x<627 && y>32 && y<417) {

hidemouse();

int i,j;

while((button & 1) ==1){

restrictmouse(52,32,627,417);

setcolor(colour);

for(i=x,j=y;i<x+5,j<y+5;i+=random(20),j+=random(20)){

if(i<x-5)break;

if(j<y-5) break;

i-=random(20);

j-=random(20);

if(i>52&&i<627&&j>32&&j<417)putpixel(i,j,colour);

getmousepos(button,x,y);

}

break;

} //end of while

}//end of if

restrictmouse(0,0,640,480);

showmouse();

} //end of while

} //end of spray()

void thickline(int prevx,int prevy,int x,int y){

int radius =3;

if(check_rub==1) {
```

```
setcolor(WHITE);

setfillstyle(SOLID_FILL,WHITE);

}

else

{

setcolor(colour);

setfillstyle(SOLID_FILL,colour);

}

int lefx,upy,rigx,lowy;

if(prevx>x){ lefx=x; rigx=prevx;}

else { lefx=prevx; rigx=x;}

if(prevy>y){ upy=y; lowy=prevy;}

else { upy=prevy; lowy=y;}

int s1,s2,q1,q2;

s1=prevx; s2=prevy;

q1=x; q2=y;

if(lefx!=rigx) for(int i=lefx;i<=rigx;i++) fillellipse(i,(((q2-s2)*(i-s1))/(q1-

s1))+s2,radius,radius);

if(upy!=lowy) for(int j=upy;j<=lowy;j++)fillellipse((((j-s2)*(q1-s1))/(q2-

s2))+s1,j,radius,radius);

showmouse();

} //end of thickline()

void brush(){

check_rub=0;

int button,x,y,prevx,prevy;

for(int k=80;k<212;k+=21) {

setcolor(BLACK);

rectangle(5,k,25,k+20); rectangle(26,k,46,k+20);

setcolor(WHITE);

line(5,k,5,k+20); line(5,k,25,k);

line(26,k,26,k+20); line(26,k,46,k);

}

setcolor(BLACK);

line(26,143,46,143); line(26,143,26,163);
```

```
setcolor(WHITE);

line(46,163,46,143); line(46,163,26,163);

while(TRUE){

getmousepos(button,x,y);

if((button &1==1 && x>5 && x<46 && y>80 && y<225)||(button &

1==1 && x>625 && x<635 && y>2 && y<12)){

control_button=1;

posx=x; posy=y;

break;

}

if(button && x>=40 && x<=157 && y>=430 && y<=457){

if(button==1) col_button=1;

if(button==2) col_button=2;

colx=x; coly=y;

selectcolour(col_button,colx,coly);

}

int q=3; //q=radius of thick line -refer thickline function

if(button & 1 ==1 && x>52+q && x<628-q && y>32+q && y<418-q) {

setcolor(colour);

prevx=x; prevy=y;

while((button==1)) {

restrictmouse(52+q,32+q,627-q,417-q);

getmousepos(button,x,y);

setcolor(colour);

hidemouse();

thickline(prevx,prevy,x,y);

showmouse();

prevx=x; prevy=y;

}

restrictmouse(0,0,getmaxx(),getmaxy());

} //end of if

} // end of while

} //end of brush()

void rub(){
```

```
check_rub=1;

int button,x,y,prevx,prevy;

for(int k=80;k<212;k+=21){

setcolor(BLACK);

rectangle(5,k,25,k+20); rectangle(26,k,46,k+20);

setcolor(WHITE);

line(5,k,5,k+20); line(5,k,25,k);

line(26,k,26,k+20); line(26,k,46,k);

}

setcolor(BLACK);

line(26,101,46,101); line(26,101,26,121);

setcolor(WHITE);

line(46,121,46,101); line(46,121,26,121);

while(TRUE){

getmousepos(button,x,y);

if((button &1==1 && x>5 && x<46 && y>80 && y<225)||(button &

1==1 && x>625 && x<635 && y>2 && y<12)) {

control_button=1;

posx=x; posy=y;

break;

}

if(button && x>=40 && x<=157 && y>=430 && y<=457){

if(button==1) col_button=1;

if(button==2) col_button=2;

colx=x; coly=y;

selectcolour(col_button,colx,coly);

}

int q=3; //q=radius of thick line - refer thickline function

if(button & 1 ==1 && x>52+q && x<628-q && y>32+q && y<418-q){

setcolor(WHITE);

prevx=x; prevy=y;

while((button==1)) {

restrictmouse(52+q,32+q,627-q,417-q);

getmousepos(button,x,y);
setcolor(WHITE);

hidemouse();

thickline(prevx,prevy,x,y);

showmouse();

prevx=x; prevy=y;

}

restrictmouse(0,0,getmaxx(),getmaxy());

}//end of if

} //end of while

} //end of rub()

void line() {

for(int k=80;k<212;k+=21) {

setcolor(BLACK);

rectangle(5,k,25,k+20); rectangle(26,k,46,k+20);

setcolor(WHITE);

line(5,k,5,k+20); line(5,k,25,k);

line(26,k,26,k+20); line(26,k,46,k);

}

setcolor(BLACK);

line(5,185,5,205); line(5,185,25,185);

setcolor(WHITE);

line(25,205,25,185); line(25,205,5,205);

int button,x,y,prevx,prevy;

while(TRUE){

getmousepos(button,x,y);

if((button & 1==1 && x>5 && x<46 && y>80 && y<225)||(button &

1==1 && x>625 && x<635 && y>2 && y<12)) {

control_button=1;

posx=x; posy=y;

break;

}

if(button && x>=40 && x<=157 && y>=430 && y<=457){

if(button==1) col_button=1;

if(button==2) col_button=2;
```

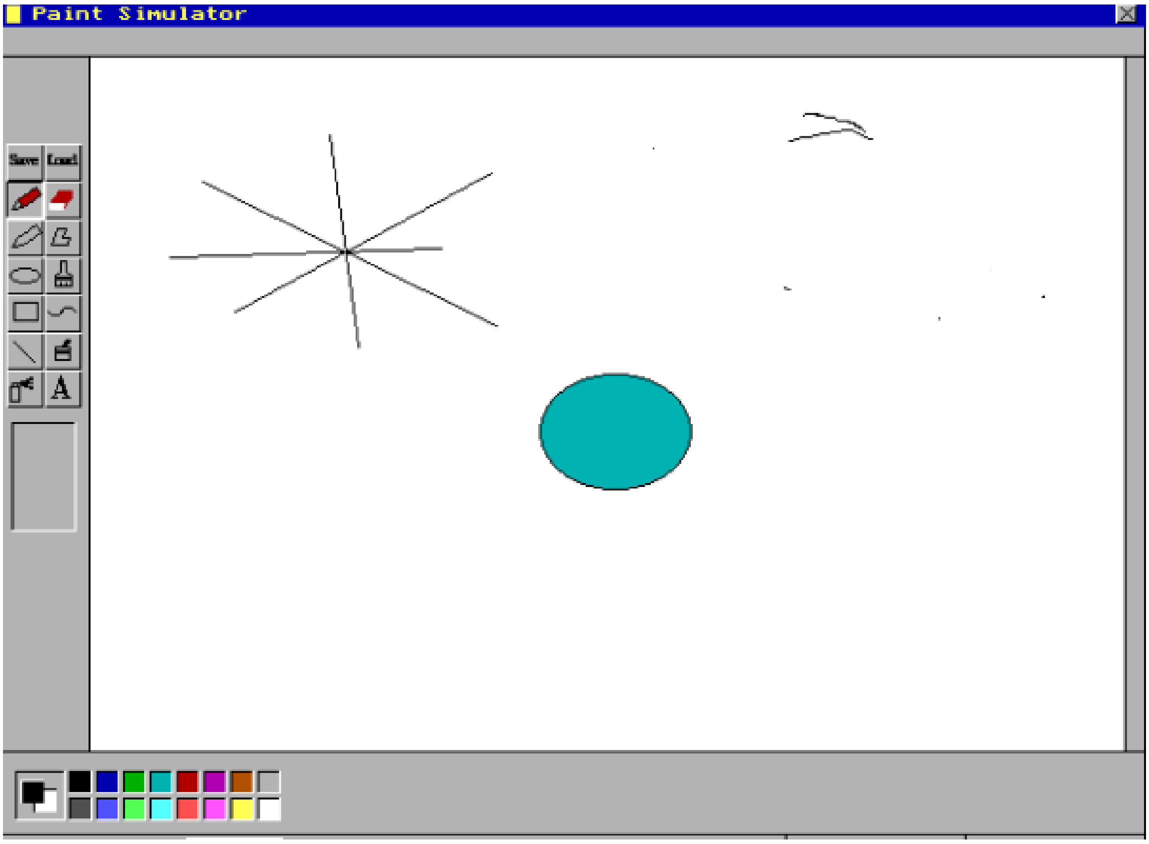
```
colx=x; coly=y;
selectcolour(col_button,colx,coly);
}
if(button & 1 ==1 && x>52 && x<628 && y>32 && y<418){
getmousepos(button,x,y);
setwritemode(XOR_PUT);
prevx=x; prevy=y;
while((button & 1) == 1){
restrictmouse(52,32,627,417);
getmousepos(button,x,y);
setcolor(15-colour);
hidemouse();
line(prevx,prevy,x,y);
line(prevx,prevy,x,y);
showmouse();
}
setwritemode(COPY_PUT);
setcolor(colour);
hidemouse();
line(prevx,prevy,x,y);
showmouse();
restrictmouse(0,0,getmaxx(),getmaxy());
} //end of if
} //end of while
} // end of line()
void myellipse(int cenx,int ceny,int xrad,int yrad) {
float cx,cy,angle=0;
while(angle<360) {
float THETA=PI/180.0 * angle;
cx=cenx+xrad*cos(THETA);
cy=ceny-yrad*sin(THETA);
line(cx,cy,cx,cy);
angle+=.5;
}
```

```
}//end of myellipse()
void circle() {
for(int k=80;k<212;k+=21) {
setcolor(BLACK);
rectangle(5,k,25,k+20); rectangle(26,k,46,k+20);
setcolor(WHITE);
line(5,k,5,k+20); line(5,k,25,k);
line(26,k,26,k+20); line(26,k,46,k);
}
setcolor(BLACK);
line(5,143,5,163); line(5,143,25,143);
setcolor(WHITE);
line(25,163,5,163); line(25,163,25,143);
int button,x,y,prevx,prevy,i,j;
while(TRUE) {
getmousepos(button,x,y);
if((button & 1==1 && x>5 && x<46 && y>80 && y<225)||(button &
1==1 && x>625 && x<635 && y>2 && y<12)){
control_button=1;
posx=x; posy=y;
break;
}
if(button && x>=40 && x<=157 && y>=430 && y<=457){
if(button==1) col_button=1;
if(button==2) col_button=2;
colx=x; coly=y;
selectcolour(col_button,colx,coly);
}
if(button & 1 ==1 && x>52 && x<628 && y>32 && y<418){
getmousepos(button,x,y);
setwritemode(XOR_PUT);
prevx=x; prevy=y;
while((button & 1) == 1){
restrictmouse(52,32,627,417);
```

```
getmousepos(button,x,y);
setcolor(15-colour);
hidemouse();
i=abs(x-prevx); j=abs(y-prevy);
myellipse((prevx+x)/2,(prevy+y)/2,i/2,j/2);
myellipse((prevx+x)/2,(prevy+y)/2,i/2,j/2);
showmouse();
}
setwritemode(COPY_PUT);
setcolor(colour);
hidemouse();
ellipse((prevx+x)/2,(prevy+y)/2,0,360,i/2,j/2);
showmouse();
restrictmouse(0,0,getmaxx(),getmaxy());
} //end of if
showmouse();
} //end of while
} //end of circle()
void rectangle() {
for(int k=80;k<212;k+=21) {
setcolor(BLACK);
rectangle(5,k,25,k+20); rectangle(26,k,46,k+20);
setcolor(WHITE);
line(5,k,5,k+20); line(5,k,25,k);
line(26,k,26,k+20); line(26,k,46,k);
}
setcolor(BLACK);
line(5,164,5,184); line(5,164,25,164);
setcolor(WHITE);
line(25,184,25,164); line(25,184,5,184);
int button,x,y,prevx,prevy;
while(TRUE) {
getmousepos(button,x,y);
if((button & 1 == 1 && x>5 && x<46 && y>80 && y<225)||(button &
1==1 && x>625 && x<635 && y>2 && y<12)){

control_button=1;

posx=x; posy=y;

break;

}

if(button && x>=40 && x<=157 && y>=430 && y<=457) {

if(button==1) col_button=1;

if(button==2) col_button=2;

colx=x;

coly=y;

selectcolour(col_button,colx,coly);

}

if(button & 1 == 1 && x>52 && x<628 && y>32 && y<418) {

getmousepos(button,x,y);

setwritemode(XOR_PUT);

prevx=x; prevy=y;

while((button & 1) == 1) {

restrictmouse(52,32,627,417);

getmousepos(button,x,y);

setcolor(15-colour);

hidemouse();

rectangle(prevx,prevy,x,y);

rectangle(prevx,prevy,x,y);

showmouse();

}

setwritemode(COPY_PUT);

setcolor(colour);

hidemouse();

rectangle(prevx,prevy,x,y);

showmouse();

restrictmouse(0,0,getmaxx(),getmaxy());

}

}

} //end of tectangle()
```

```
void main() {

int gdriver = DETECT, gmode;

initgraph(&gdriver, &gmode, "C:\\turboc3\\bgi");

gui();

//(52,30,628,418) is the drawing area with lightgray background

initmouse();

showmouse();

setcolor(BLACK);

colour=0;

pencil(); //to start with by default

while(TRUE) {

// Action after left-clicking on first row of icons

if(control_button==1 && posx>5 && posx<25 && posy>100 && posy<120) pencil();

if(control_button==1 && posx>5 && posx<25 && posy>121 && posy<141) getcolour();

if(control_button==1 && posx>5 && posx<25 && posy>142 && posy<162) circle();

if(control_button==1 && posx>5 && posx<25 && posy>163 && posy<183) rectangle();

if(control_button==1 && posx>5 && posx<25 && posy>184 && posy<204) line();

if(control_button==1 && posx>5 && posx<25 && posy>205 && posy<225) spray();

// Acion after left-clicking second row of icons

if(control_button==1 && posx>26 && posx<46 && posy>100 && posy<120) rub();

if(control_button==1 && posx>26 && posx<46 && posy>121 && posy<141) poly();

if(control_button==1 && posx>26 && posx<46 && posy>142 && posy<162) brush();

if(control_button==1 && posx>26 && posx<46 && posy>163 && posy<183) curve();

if(control_button==1 && posx>26 && posx<46 && posy>184 && posy<204) ffill();

if(control_button==1 && posx>26 && posx<46 && posy>205 && posy<225) text();

// binary file save and load

if(control_button==1 && posx>5 && posx<25 && posy>80 && posy<100)

{

save_image();

outtextxy(4,335,"SAVED");

pencil(); //focus on pencil button

}

if(control_button==1 && posx>26 && posx<46 && posy>80 && posy<100)

{
```

```
load_image();

pencil(); //focus on pencil button

}

// Acion after left-clicking on exit icon

if(control_button==1 && posx>625 && posx<635 && posy>2 && posy<12) exit(0);

}

}//end of main()
```



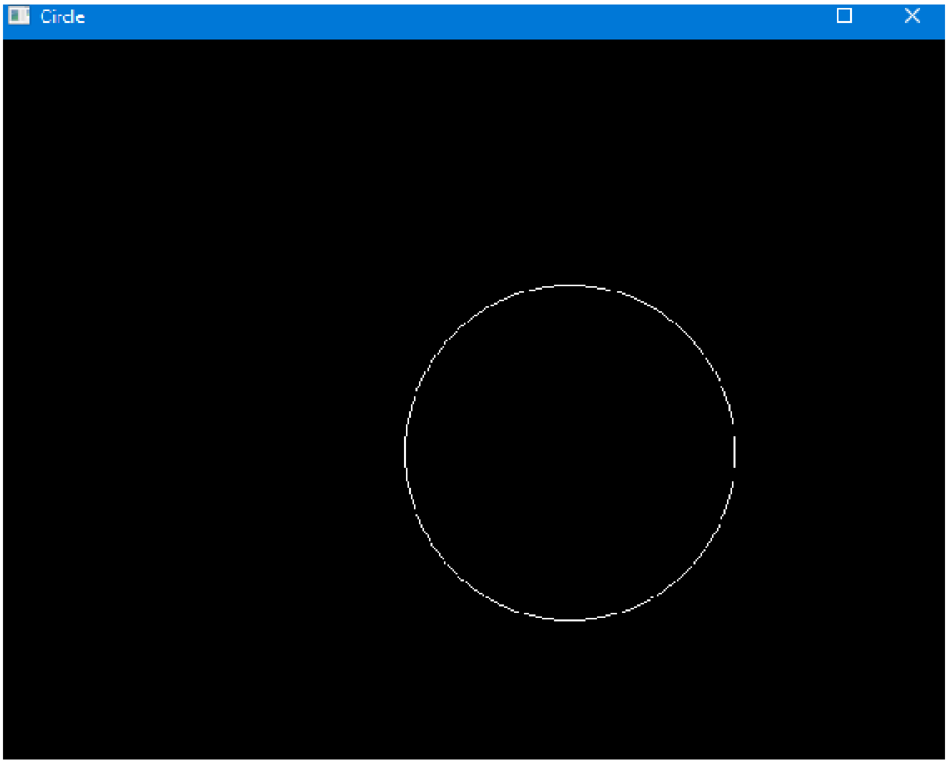
Computer Graphics: Introduction to OpenGL (Dr R K Singla)

PROGRAM – 1 - TO CREATE A WINDOW AND DRAW A CIRCLE

```
//circle drawing

#include <GL/glut.h>


#include <stdlib.h>
```

```c
#include <stdio.h>
#include <math.h>

int valid = 0;
int radius = 0;
int xi, yi, xf, yf;

void mouse_func(int button, int state, int x, int y) {

    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        valid = state == GLUT_DOWN;
        radius = 0;
        xi = x - 320;
        yi = 240 - y;
    }
    if (button == GLUT_LEFT_BUTTON && state == GLUT_UP) {
        valid = state == GLUT_DOWN;
    }

}

int get_radius(int dx, int dy) {
    return sqrt((double)((dx * dx) + (dy * dy)));
}

void motion_func(int x, int y) {

    if (valid) {
        int dx = xi - (x - 320);
        int dy = yi - (240 - y);
        radius = get_radius(dx, dy);
    }
    xf = x - 320;
    yf = 240 - y;

    //printf("%d %d \n",x,y );
}
```

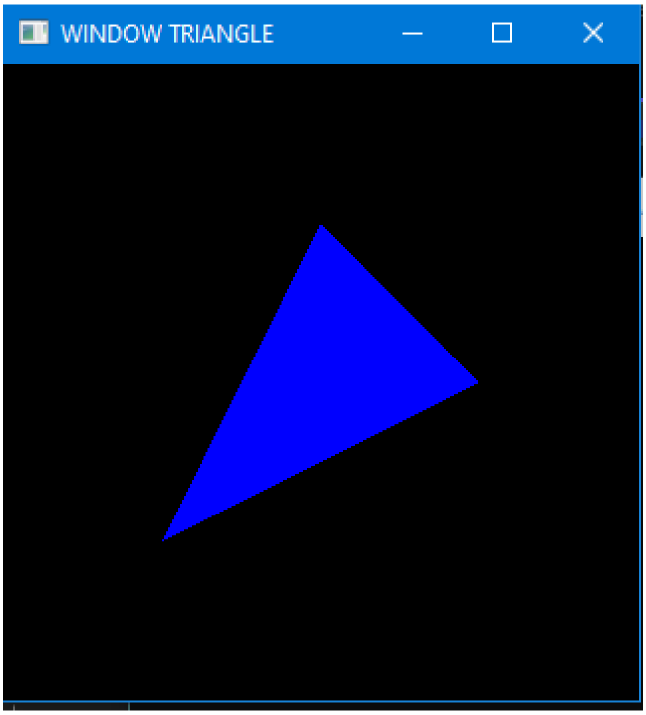
```
static void resize(int width, int height) {
    const float ar = (float)width / (float)height;
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-320, 319, -240, 239, -1, 1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void draw8way(int c_x, int c_y, int x, int y) {
    glVertex2i(c_x + x, c_y + y);
    glVertex2i(c_x - x, c_y + y);

    glVertex2i(c_x + y, c_y + x);
    glVertex2i(c_x + y, c_y - x);

    glVertex2i(c_x - y, c_y + x);
    glVertex2i(c_x - y, c_y - x);

    glVertex2i(c_x + x, c_y - y);
    glVertex2i(c_x - x, c_y - y);

}

void drawCircle(int c_x, int c_y, int r)
{
    int x = 0, y = r;
    int d = 5 - 4 * r;

    draw8way(c_x, c_y, x, y);

    while (x < y)
    {
        if (d < 0)//dE
        {
            d += (8 * x + 12);
            x++;
        }
        else//dSE
        {
            d += (8 * x - 8 * y + 20);
            x++;
            y--;
        }
        draw8way(c_x, c_y, x, y);
    }

}

int get_dis(int xo, int yo, int xt, int yt) {
    int ddd = ((xt - xo) * (xt - xi)) + ((yt - yo) * (yt - yo));
    return ddd;
}

void Circle_drawing()
{

    drawCircle(xi, yi, radius);
}

static void display(void) {
    int x = 10, y = 20;
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3d(1, 1, 1);
    // glBegin(GL_LINES);
    // glVertex2i(-320, 0);
    // glVertex2i(319, 0);
    // glVertex2i(0, -240);
    // glVertex2i(0, 239);
    // glEnd();
    glBegin(GL_POINTS);
    Circle_drawing();
    glEnd();
    glFlush();
}

static void key(unsigned char key, int x, int y)
{
    switch (key)
    {
    case 27:
    case 'q':
        exit(0);
        break;
    }
    //glutPostRedisplay();
}

static void idle(void)
{
    glutPostRedisplay();
}

/* Program entry point */

int main(int argc, char* argv[]) {
    glutInit(&argc, argv);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(10, 10);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE | GLUT_DEPTH);

    glutCreateWindow("Circle");

    glutReshapeFunc(resize);
    glutDisplayFunc(display);
    glutKeyboardFunc(key);
    glutIdleFunc(idle);
    glutMouseFunc(mouse_func);

    glutMotionFunc(motion_func);

    glutMainLoop();

    return EXIT_SUCCESS;

}
```



PROGRAM – 2 - TO CREATE A WINDOW AND DRAW A TRIANGLE

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

void renderScene(void)

{

glClear(GL_COLOR_BUFFER_BIT);

glBegin(GL_TRIANGLES);

glColor3f(0.0f, 0.0f, 1.0f); // set the drawing color to blue

glVertex3f(-0.5,-0.5,0.0);

glVertex3f(0.5,0.0,0.0);

glVertex3f(0.0,0.5,0.0);

glEnd();

glFlush();

}

int main(int argc, char** argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT_DEPTH | GLUT_SINGLE |GLUT_RGBA);
glutInitWindowPosition(100,100);

glutInitWindowSize(320,320);

glutCreateWindow("WINDOW TRIANGLE");


glutDisplayFunc(renderScene);

glutMainLoop();

return 0;

}
```

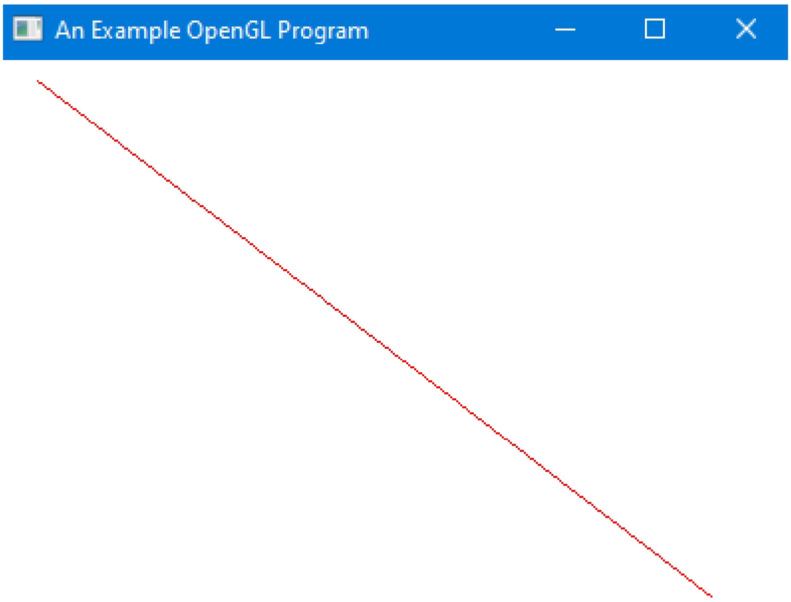
OUTPUT:



PROGRAM – 3 - TO DRAW A LINE IN THE DISPLAY WINDOW

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

void init(void)

{

  glClearColor(1.0,1.0,1.0,0.0); // set the display window color to white

  glMatrixMode(GL_PROJECTION); // set the projection parameters

  gluOrtho2D(0.0, 200.0, 0.0, 150.0);

}

void lineSegment(void)

{

  glClear(GL_COLOR_BUFFER_BIT); // clear the display window

  glColor3f(1.0,0.0,0.0); //set line segment color to red

  glBegin(GL_LINES);
  glVertex2i(180,15); // specify the line segment geometry


  glVertex2i(10,145);

  glEnd();

  glFlush(); // process all OpenGL functions as quickly as possible

}

int main(int argc, char** argv)

{

  glutInit(&argc, argv); // initialize glut

  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); //set display mode

  glutInitWindowPosition(50,100); //set top left display window position

  glutInitWindowSize(400,300); //set display window height and width

  glutCreateWindow("An Example OpenGL Program"); //create display window

  init(); // execute initialization procedure

  glutDisplayFunc(lineSegment); //send graphics to display window

  glutMainLoop(); // display everything and wait

}
```

OUTPUT:



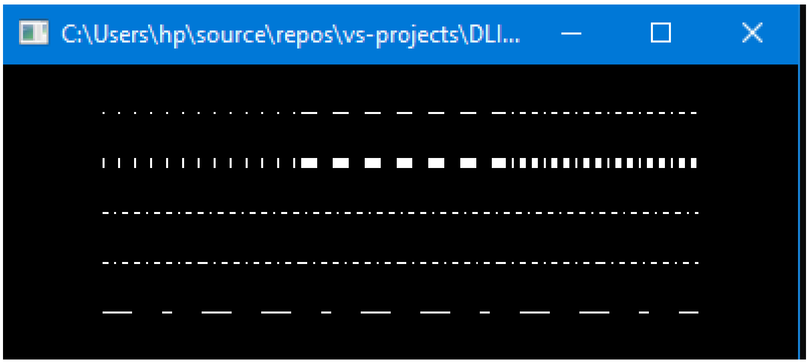
PROGRAM – 4 - TO DRAW DIFFERENT LINE PATTERNS

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

void drawOneLine(GLfloat x1,GLfloat y1,GLfloat x2,GLfloat y2)

{ glBegin(GL_LINES);

glVertex2f ((x1),(y1));
```

```
glVertex2f ((x2),(y2));

glEnd();

}

void init(void)

{ glClearColor (0.0, 0.0, 0.0, 0.0); }

void display(void)

{ int i;

  glClear (GL_COLOR_BUFFER_BIT);

  /* select white color for all lines */

  glColor3f (1.0, 1.0, 1.0);

/* in 1st row, 3 lines, each with a different stipple */

  glEnable (GL_LINE_STIPPLE);

  glLineStipple (1, 0x0101); /* dotted */

drawOneLine (50.0, 125.0, 150.0, 125.0);

glLineStipple (1, 0x00FF); /* dashed */

drawOneLine (150.0, 125.0, 250.0, 125.0);

glLineStipple (1, 0x1C47); /* dash/dot/dash */

drawOneLine (250.0, 125.0, 350.0, 125.0);

  /* in 2nd row, 3 wide lines, each with different stipple */

glLineWidth (5.0);

glLineStipple (1, 0x0101); /* dotted */

drawOneLine (50.0, 100.0, 150.0, 100.0);

  glLineStipple (1, 0x00FF); /* dashed */

  drawOneLine (150.0, 100.0, 250.0, 100.0);

glLineStipple (1, 0x1C47); /* dash/dot/dash */

drawOneLine (250.0, 100.0, 350.0, 100.0);

glLineWidth (1.0);


/*in 3rd row, 6 lines, with dash/dot/dash stipple as part of a single connected line strip */

glLineStipple (1, 0x1C47); /* dash/dot/dash */

glBegin (GL_LINE_STRIP);

for (i = 0; i < 7; i++)

  glVertex2f (50.0 + ((GLfloat) i * 50.0), 75.0);

glEnd ();
```

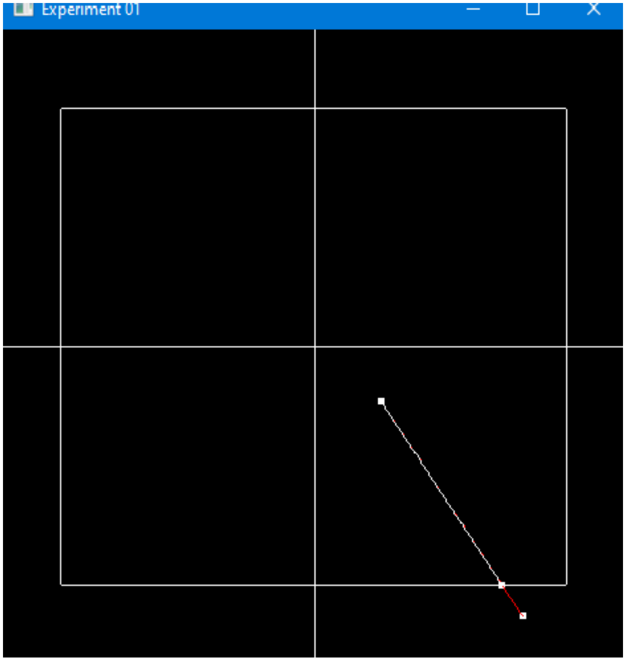
```
/* in 4th row, 6 independent lines with same stipple */
for (i = 0; i < 6; i++)
{
  drawOneLine (50.0 + ((GLfloat) i * 50.0), 50.0,50.0 + ((GLfloat)(i+1) * 50.0), 50.0);
}
/* in 5th row, 1 line, with dash/dot/dash stipple and a stipple repeat factor of 5 */
glLineStipple (5, 0x1C47); /* dash/dot/dash */
drawOneLine (50.0, 25.0, 350.0, 25.0);
glDisable (GL_LINE_STIPPLE);
glFlush ();
}
void reshape (int w, int h)
{
glViewport (0, 0, (GLsizei) w, (GLsizei) h);
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
  gluOrtho2D (0.0, (GLdouble) w, 0.0, (GLdouble) h);
}
int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize (400, 150);
  glutInitWindowPosition (100, 100);
  glutCreateWindow (argv[0]);
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMainLoop();
return 0;
}
```

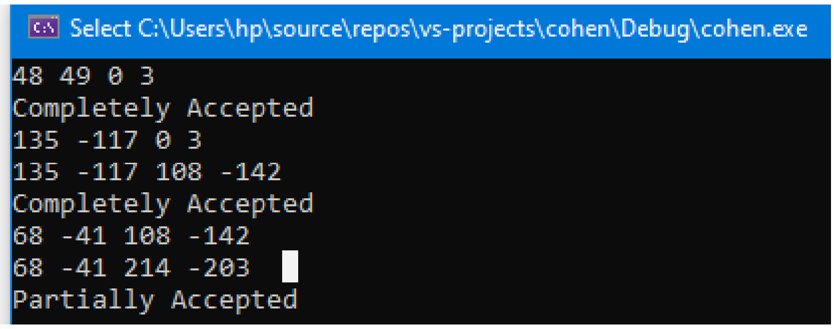
OUTPUT:



PROGRAM – 5 - TO Cohen-Sutherland Line Clipping

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

#include <stdlib.h>
#include <stdio.h>

int click = 0; int nxyget = 0;
bool check = true;
int zone;
int x0, y0, x1, y1;
int x0Click, y0Click, x1Click, y1Click;
int gp = 60;
int xl = -320, yl = -240, xr = 319, yr = 239;
int xmax = xr - gp, xmin = xl + gp, ymax = yr - gp, ymin = yl + gp;
bool writen = false;

int makeCode(int x, int y) {
    int code = 0;
    if (y > ymax)code += 8;
    else if (y < ymin)code += 4;

    if (x > xmax)code += 2;
    else if (x < xmin)code += 1;

    return code;
}

void get_zone(int x0, int y0, int x1, int y1) {
    int dX = x1 - x0;
    int dY = y1 - y0;

    if (dX >= 0 && dY >= 0)
    {
        if (dX > dY) zone = 0; else zone = 1;
    }
    else if (dX < 0 && dY >= 0)
    {
        if (abs(dX) > dY) zone = 3; else zone = 2;
    }
    else if (dX < 0 && dY < 0)
    {
        if (abs(dX) > abs(dY)) zone = 4; else zone = 5;
    }
    else if (dX >= 0 && dY < 0)
    {
        if (dX > abs(dY)) zone = 7; else zone = 6;
    }
}

void drawLine(int x0, int y0, int x1, int y1, int zone)
{
    int dX = x1 - x0;
    int dY = y1 - y0;

    int x = x0, y = y0;
    int d = 2 * dY - dX;

    int dE = 2 * dY, dNE = 2 * (dY - dX);

    // glVertex2i(x,y);

    while (x < x1)
    {

        if (d < 0)
        {
            x++;
            d += dE;
        }
        else
        {
            x++;
            y++;
            d += dNE;
        }
        if (zone == 0)
            glVertex2i(x, y);
        else if (zone == 1)
            glVertex2i(y, x);
        else if (zone == 2)
            glVertex2i(-y, x);
        else if (zone == 3)
            glVertex2i(-x, y);
        else if (zone == 4)
            glVertex2i(-x, -y);
        else if (zone == 5)
            glVertex2i(-y, -x);
        else if (zone == 6)
            glVertex2i(y, -x);
        else if (zone == 7)
            glVertex2i(x, -y);

    }

}

void lineDraw(int x0, int y0, int x1, int y1)
{
    get_zone(x0, y0, x1, y1);
    if (zone == 0) {
        drawLine(x0, y0, x1, y1, zone);
    }
    else if (zone == 1) {
        drawLine(y0, x0, y1, x1, zone);
    }
    else if (zone == 2) {
        // glColor3f(0,0,1);
        drawLine(y0, -x0, y1, -x1, zone);
    }
    else if (zone == 3) {
        // glColor3f(1,1,0);
        drawLine(-x0, y0, -x1, y1, zone);
    }
    else if (zone == 4) {
        // glColor3f(1,0,1);
        drawLine(-x0, -y0, -x1, -y1, zone);
    }
    else if (zone == 5) {
        // glColor3f(0,1,1);
        drawLine(-y0, -x0, -y1, -x1, zone);
    }
    else if (zone == 6) {
        // glColor3f(1,1,1);
        drawLine(-y0, x0, -y1, x1, zone);
    }
    else if (zone == 7) {
        // glColor3f(0.6,0.5,0.4);
        drawLine(x0, -y0, x1, -y1, zone);
    }

}

void mouse(int button, int state, int mousex, int mousey) {
    int x, y;
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        check = true;
        x = mousex - 320;
        y = 240 - mousey;

        if (click == 0) {
            x0Click = x0 = x;
            y0Click = y0 = y;
            click = 1;
            nxyget = 0;
        }
        else if (click == 1) {
            x1Click = x1 = x;
            y1Click = y1 = y;
            click = 2;

            writen = false;
        }
        else {
            x0Click = x0 = x;
            y0Click = y0 = y;
            click = 1;
            nxyget = 0;
        }
        printf("%d %d %d %d\n", x0, y0, x1, y1);
    }
    else if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
        glClearColor(1, 1, 1, 0);
        glClear(GL_COLOR_BUFFER_BIT);
        check = false;
    }
    glutPostRedisplay();
}

void CohenSutherLand() {
    int code0, code1, code;
    int x, y;
    int top = 8, bottom = 4, left = 1, right = 2;
    code0 = makeCode(x0, y0);
    code1 = makeCode(x1, y1);

    while (1) {
        if (!(code0 | code1)) {
            nxyget = 1;
            lineDraw(x0, y0, x1, y1);
            if (!writen)
                printf("Completely Accepted\n"), writen = true;
            break;
        }
        else if (code0 & code1) {
            if (!writen)
                printf("Completely Rejected\n"), writen = true;
            break;
        }
        else {
            if (!writen)
                printf("Partially Accepted\n"), writen = true;
            if (code0)code = code0;
            else code = code1;

            if (code & top) {
                y = ymax;
                x = x0 + ((y - y0) * (x1 - x0)) / (y1 - y0);

            }
            else if (code & bottom) {
                y = ymin;
                x = x0 + ((y - y0) * (x1 - x0)) / (y1 - y0);
            }
            else if (code & left) {
                x = xmin;
                y = y0 + ((x - x0) * (y1 - y0)) / (x1 - x0);
            }
            else {
                x = xmax;
                y = y0 + ((x - x0) * (y1 - y0)) / (x1 - x0);
            }

            if (code == code0) {
                x0 = x; y0 = y;
                code0 = makeCode(x0, y0);
            }
            else if (code == code1) {
                x1 = x; y1 = y;
                code1 = makeCode(x1, y1);
            }
        }
    }
}

static void resize(int width, int height) {
    const float ar = (float)width / (float)height;
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-320, 319, -240, 239, -1, 1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

static void display(void) {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3d(1, 1, 1);
    glBegin(GL_LINES);
    glVertex2i(-320, 0);
    glVertex2i(319, 0);
    glVertex2i(0, -240);
    glVertex2i(0, 239);

    glVertex2i(xl + gp, yl + gp);
    glVertex2i(xl + gp, yr - gp);

    glVertex2i(xl + gp, yr - gp);
    glVertex2i(xr - gp, yr - gp);

    glVertex2i(xr - gp, yr - gp);
    glVertex2i(xr - gp, yl + gp);
    glVertex2i(xr - gp, yl + gp);
    glVertex2i(xl + gp, yl + gp);
    glEnd();
    glPointSize(5.0);
    glBegin(GL_POINTS);
    if (click == 1) {
        glVertex2i(x0Click, y0Click);
    }
    else if (nxyget == 0 and click == 2) {
        glVertex2i(x0Click, y0Click);
        glVertex2i(x1Click, y1Click);
    }
    else if (nxyget == 1 and click == 2) {
        glVertex2i(x0Click, y0Click);
        glVertex2i(x1Click, y1Click);
        glVertex2i(x0, y0);
        glVertex2i(x1, y1);
    }

    glEnd();
    glPointSize(1.0);
    glBegin(GL_POINTS);
    if (click == 2) {
        glColor3f(1, 0, 0);
        lineDraw(x0Click, y0Click, x1Click, y1Click);
        glColor3f(1, 1, 1);
        CohenSutherLand();
    }

    glEnd();
    glutSwapBuffers();
}


static void key(unsigned char key, int x, int y)
{
    switch (key)
    {
    case 27:
    case 'q':
        exit(0);
        break;
    }

    //glutPostRedisplay();
}

static void idle(void)
{
    glutPostRedisplay();
}

/* Program entry point */

int main(int argc, char* argv[]) {
    glutInit(&argc, argv);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(10, 10);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

    glutCreateWindow("Experiment 01");

    glutReshapeFunc(resize);
    glutDisplayFunc(display);
    glutKeyboardFunc(key);
    glutIdleFunc(idle);
    glutMouseFunc(mouse);
    glutMainLoop();
    return EXIT_SUCCESS;
}
```



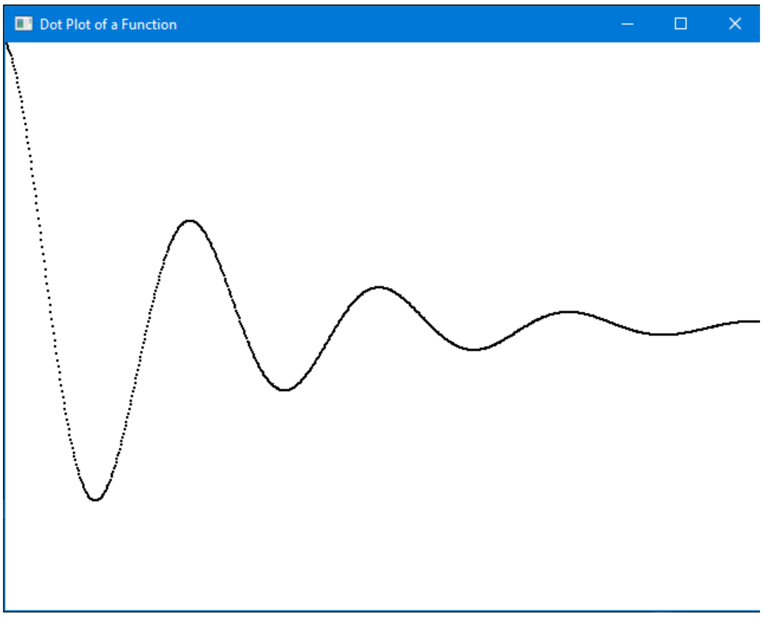

Select C:\Users\hp\source\repos\vs-projects\cohen\Debug\cohen.exe

```
48 49 0 3
Completely Accepted
135 -117 0 3
135 -117 108 -142
Completely Accepted
68 -41 108 -142
68 -41 214 -203
Partially Accepted
```

PROGRAM – 6 - TO DOT Plot

```
#include <windows.h>

#include <math.h>

#include <gl/glut.h>

const int screenWidth = 640; // width of screen window in pixels

const int screenHeight = 480; // height of screen window in pixels

GLdouble A, B, C, D; // values used for scaling and shifting

void myInit(void){

        glClearColor(1.0,1.0,1.0,0.0); // background color is white

        glColor3f(0.0f, 0.0f, 0.0f); // drawing color is black

        glPointSize(2.0); // a 'dot' is 2 by 2 pixels

        glMatrixMode(GL_PROJECTION); // set "camera shape"

        glLoadIdentity();

        gluOrtho2D(0.0, (GLdouble)screenWidth, 0.0, (GLdouble)screenHeight);

        A = screenWidth / 4.0; // set values used for scaling and shifting
        B = 0.0;

        C = D = screenHeight / 2.0;

}

//<<<<<<<<<<<<<<<<<<<<<<<<< myDisplay >>>>>>>>>>>>>>>>>

void myDisplay(void){

        glClear(GL_COLOR_BUFFER_BIT); // clear the screen

        glBegin(GL_POINTS);

        for(GLdouble x = 0; x < 4.0 ; x += 0.005){

                GLdouble func = exp(-x) * cos(2 * 3.14159265 * x);

                glVertex2d(A * x + B, C * func + D);

        }

        glEnd();

        glFlush(); // send all output to display

}

//<<<<<<<<<<<<<<<<<<<<<<<<< main >>>>>>>>>>>>>>>>>>>>>>

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

        glutInit(&argc, argv); // initialize the toolkit

        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); // set display mode

        glutInitWindowSize(screenWidth, screenHeight); // set window size

        glutInitWindowPosition(100, 150); // set window position on screen

        glutCreateWindow("Dot Plot of a Function"); // open the screen window

        glutDisplayFunc(myDisplay); // register redraw function

        myInit();

        glutMainLoop(); // go into a perpetual loop

}
```

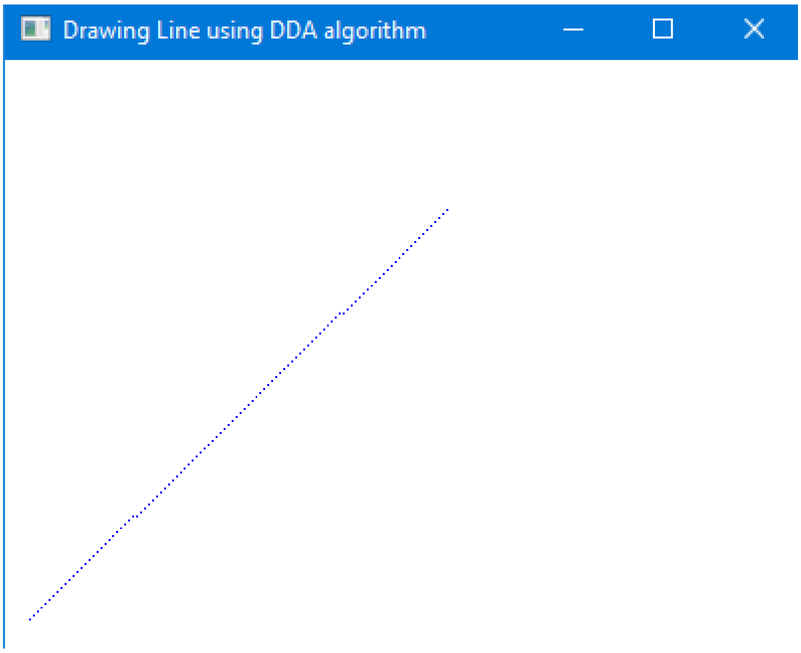
output:-



PROGRAM – 7 - To draw line using DDA Algorithm

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

#include <GL/gl.h>

#include <stdlib.h>

#include <math.h>

void init (void)

{

  glClearColor (1.0, 1.0, 1.0, 0.0); // Set display-window color to white.

  glMatrixMode (GL_PROJECTION); // Set projection parameters.

  gluOrtho2D (0.0, 200.0, 0.0, 150.0);

}

int round (const float a)

{ return int (a + 0.5); }

void setPixel(GLint xCoordinate, GLint yCoordinate)

{ glBegin(GL_POINTS);

  glVertex2i(xCoordinate, yCoordinate);

  glEnd();

  glFlush(); //process all OpenGL functions as quickly as possible

}

void lineDDA (void)

{

glClear (GL_COLOR_BUFFER_BIT); // Clear display window.

  glColor3f (0.0, 0.0, 1.0); // Set line segment color to blue.

  int x0 = 6;
  int y0 = 9;
  int xEnd = 111;
  int yEnd = 112;
  int dx = xEnd - x0, dy = yEnd - y0, steps, k;
  float xIncrement, yIncrement, x = x0, y = y0;
  if (fabs (dx) > fabs (dy))

  steps = abs (dx);
  else
  steps = abs (dy);
  xIncrement = float (dx) / float (steps);
  yIncrement = float (dy) / float (steps);
  setPixel(round(x),round(y));
  for (k = 0; k < steps; k++)
  { x += xIncrement;
  y += yIncrement;
  setPixel(round(x),round(y));
  }
}
int main (int argc, char** argv)
{
  glutInit (&argc, argv); // Initialize GLUT.
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB); // Set display mode.
  glutInitWindowPosition (50, 100); // Set top-left display-window position.
  glutInitWindowSize (400, 300); // Set display-window width and height
  // Create display window.
glutCreateWindow ("Drawing Line using DDA algorithm");
init ( ); // Execute initialization procedure.
  glutDisplayFunc (lineDDA);
  glutMainLoop ( ); // Display everything and wait.
  return 0;
  }
```

OUTPUT:



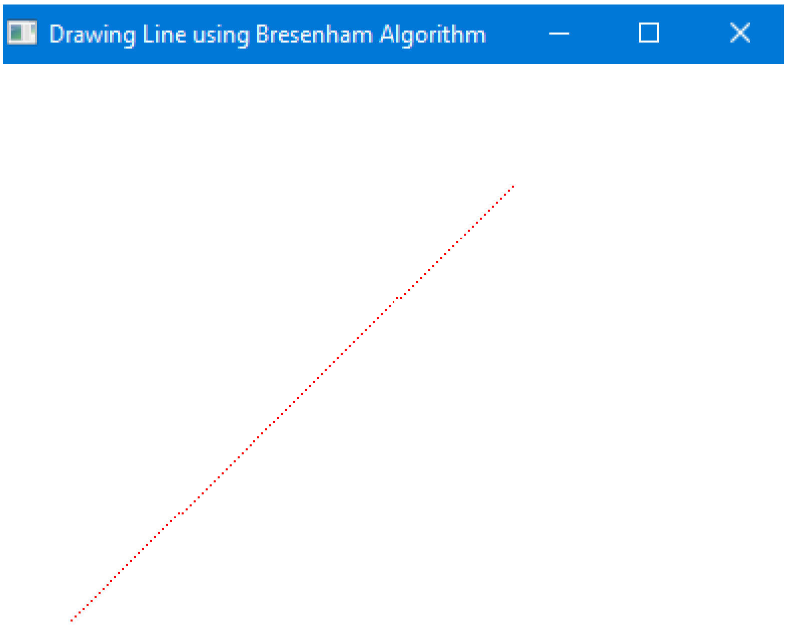
PROGRAM – 8 - To draw line using Bresenham's Algorithm

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

#include <GL/gl.h>

#include <stdlib.h>

#include <math.h>

void init (void)

{

glClearColor (1.0, 1.0, 1.0, 0.0); // Set display-window color to white.

  glMatrixMode (GL_PROJECTION); // Set projection parameters.

  gluOrtho2D (0.0, 200.0, 0.0, 150.0);

}

void setPixel(int xCoordinate, int yCoordinate)

{

glBegin(GL_POINTS);

glVertex2i(xCoordinate, yCoordinate);

glEnd();

glFlush(); //process all OpenGL functions as quickly as possible

}

void lineBres (void)

{

glClear (GL_COLOR_BUFFER_BIT); // Clear display window.

glColor3f (1.0, 0.0, 0.0); // Set line segment color to red.
```

```
int x0 = 20; int y0 = 10;

int xEnd = 130; int yEnd = 118;

int dx = abs(xEnd - x0), dy = abs(yEnd - y0);

int p = 2 * dy - dx;

int twoDy = 2 * dy,

twoDyMinusDx = 2 * (dy - dx);

  int x, y;

  if ((x0 > xEnd) || (y0 > yEnd))

  {

  x = xEnd; y = yEnd;

  xEnd = x0; yEnd = y0;

}

  else

  {

  x = x0;

  y = y0;

  }

setPixel(x,y);

  while ((x <= xEnd) && (y <= yEnd))

{

If (dx >0)

  x++;

  if (p < 0)

  p += twoDy;

  else

  {

  y++;

  p += twoDyMinusDx;

  }

  setPixel(x,y);

}

}

int main (int argc, char** argv)

{
```

```
glutInit (&argc, argv); // Initialize GLUT.
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB); // Set display mode.
  glutInitWindowPosition (50, 100); // Set top-left display-window position.
  glutInitWindowSize (400, 300); // Set display-window width and height.
  glutCreateWindow ("Drawing Line using Bresenham Algorithm");
init ( ); // Execute initialization procedure.
glutDisplayFunc (lineBres);
glutMainLoop ( ); // Display everything and wait.
return 0;
}
```

OUTPUT:



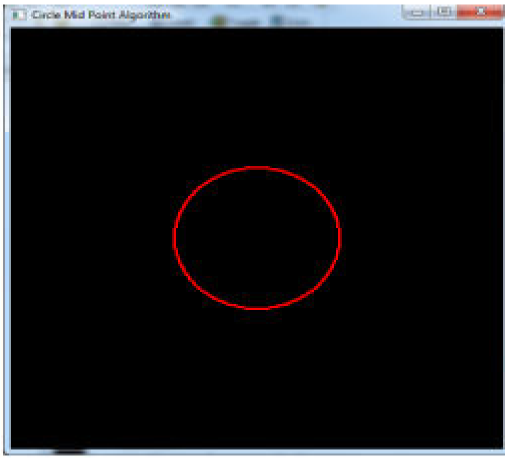
PROGRAM – 9 - To draw cirlce using Bresenham’s MidpointAlgorithm

```
#include<GL/glut.h>
#include<stdio.h>
#include<stdlib.h>
void setPixel(GLint xCoordinate, GLint yCoordinate)
{
glBegin(GL_POINTS);
glVertex2i(xCoordinate, yCoordinate);
glEnd();
glFlush(); }
void circleMidPoint(GLint xc, GLint yc, GLint radius)
{
```

```
int x=0,y=radius;
GLint p = 1-radius;
//function prototype
void circlePlotPoints(GLint, GLint, GLint,GLint);
circlePlotPoints(xc, yc,x,y);
while(x < y)
{ x++;
if(p<0) p += 2 * x + 1;
else
{ y--; p += 2 * (x-y) + 1; }
circlePlotPoints(xc, yc, x, y);
  } }
void circlePlotPoints(GLint xc, GLint yc, GLint x, GLint y)
{
setPixel(xc + x , yc + y);
setPixel(xc - x , yc + y);
setPixel(xc + x , yc - y);
setPixel(xc - x , yc - y);
setPixel(xc + y , yc + x);

setPixel(xc - y , yc + x);
setPixel(xc + y , yc - x);
setPixel(xc - y , yc - x);
}
void init(void)
{
glClearColor(0.0,0.0,0.0,0.0);
glMatrixMode(GL_PROJECTION);
gluOrtho2D(0.0,300.0,0.0,300.0);
}
void drawMyCircle(void)
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0,0.0,0.0);
```

```
glPointSize(3.0);

GLint xCenter=150;

GLint yCenter=150;

GLint radius=50;

circleMidPoint(xCenter, yCenter, radius);

}

int main(int argc, char**argv)

{

glutInit(&argc, argv);

glutInitWindowPosition(10,10);

glutInitWindowSize(500,500);

glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

glutCreateWindow("Circle Mid Point Algorithm ");

init();

glutDisplayFunc(drawMyCircle);

glutMainLoop();

return 0;

}
```

OUTPUT:

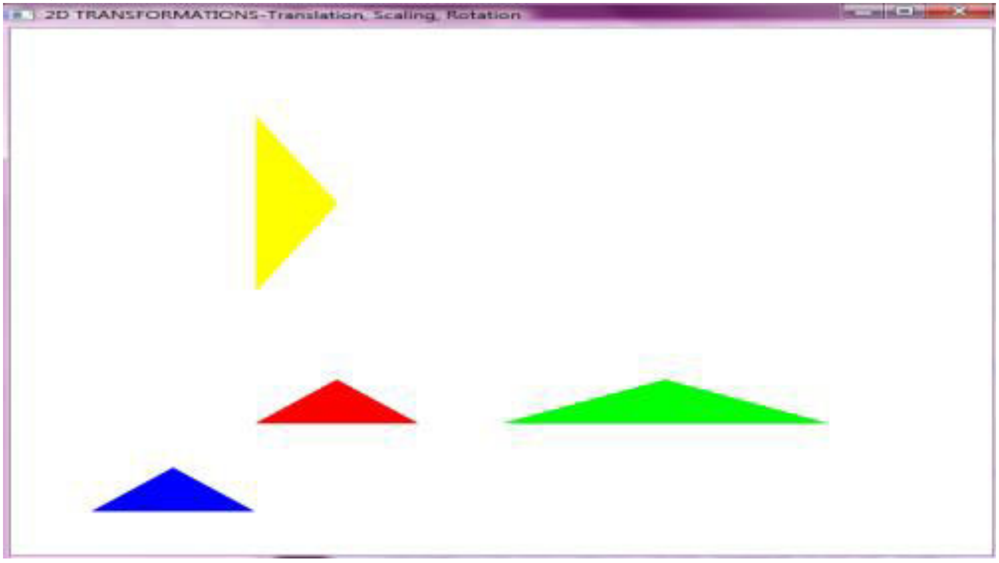


/ PROGRAM – 10 - 2D Transformations without using gl functions

```
#include <GL/glut.h>
#include <stdlib.h>
#include <iostream.h>
#include <math.h>
#include <windows.h> // use as needed for your system
#include <gl/gl.h>
class ppoints
{ public:
GLfloat x,y;
};
const GLdouble pi = 3.14159;
```

```
void init (void)
{
  glClearColor (1.0, 1.0, 1.0, 0.0); // Set display-window color to white.
glMatrixMode (GL_PROJECTION); // Set projection parameters.
gluOrtho2D (0.0, 600.0, 0.0, 600.0);
}
void translate(ppoints *verts, GLint nVerts, GLfloat tx, GLfloat ty)
{
GLint k;
for(k=0;k<nVerts; k++)
{
verts[k].x = verts[k].x+tx;
verts[k].y = verts[k].y+ty;
}
}

void scale(ppoints *verts,GLint nVerts,GLfloat sx, GLfloat sy)
{
GLint k;
for(k=0;k<nVerts; k++)
{
verts[k].x = verts[k].x * sx;
verts[k].y = verts[k].y * sy;
}
}
void rotate(ppoints *verts,GLint nVerts,GLdouble theta,ppoints *n)
{
GLint k;
for(k=0;k<nVerts;k++)
{
n[k].x = fabs(verts[k].x * cos(theta) - verts[k].y * sin(theta));
  n[k].y = fabs(verts[k].x * sin(theta) + verts[k].y * cos(theta));
}
}
void triangle(ppoints *verts)
{
GLint k;
glBegin(GL_TRIANGLES);
for (k =0;k<3; k++)
glVertex2f(verts[k].x,verts[k].y);
glEnd ( ) ;
glFlush();
}
void myDisplay(void)
{
glClear (GL_COLOR_BUFFER_BIT); // Clear display window.
glColor3f (0.0, 0.0, 1.0); // original triangle in blue color
GLint nVerts=3;
ppoints verts[3] = { {50.0,50.0},{100.0,100.0},{150.0,50.0}};
ppoints n[3]= {{1.0,1.0},{1.0,1.0},{1.0,1.0}};
GLfloat tx = 100.0,ty = 100.0;
GLfloat sx = 2.0,sy = 1.0;
GLdouble theta = pi/2.0;
triangle(verts);
translate(verts,nVerts,tx,ty);
glColor3f (1.0, 0.0, 0.0); // translated triangle in red color
triangle(verts);
scale(verts,nVerts,sx,sy);
glColor3f (0.0, 1.0, 0.0); // Scaled triangle in green color
triangle(verts);
glColor3f (0.0, 1.0, 1.0);
rotate(verts,nVerts,theta,n);
glColor3f (1.0, 1.0, 0.0); // Rotated triangle in Yellow color
triangle(n);
}
int main(int argc, char** argv)
{
glutInit(&argc, argv); // initialize the toolkit
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); // set display mode
glutInitWindowPosition (0,0); // Set top-left display-window position.
glutInitWindowSize (600,600); // Set display-window width and height
glutCreateWindow("2D TRANSFORMATIONS-Translation, Scaling, Rotation");
// open the screen window
init();
glutDisplayFunc(myDisplay); // register redraw function
glutMainLoop(); // go into a perpetual loop
return 0;
}
```

Output:



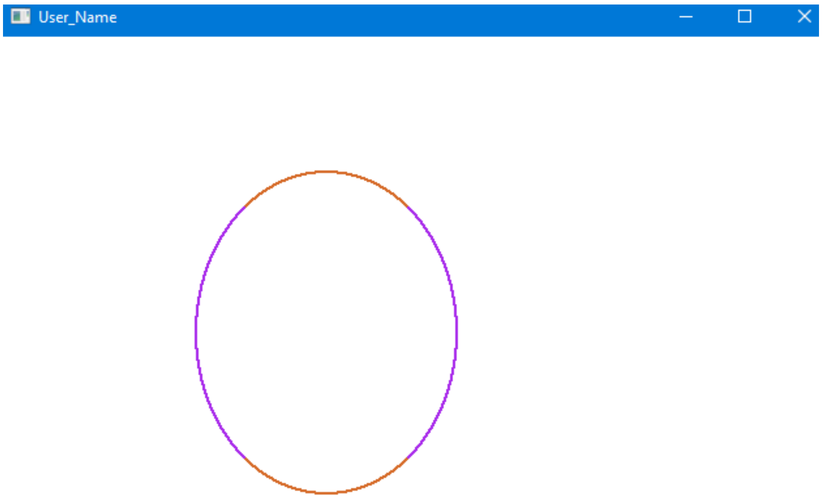
PROGRAM – 11 Ellipse using gl functions

```
//Ellipse
#include <GL/glut.h>
#include<iostream>
using namespace std;
int rx = 100, ry = 125;
int xCenter = 250, yCenter = 250;

void myinit(void)
{
	glClearColor(1.0, 1.0, 1.0, 0.0);
	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()
{
    float x = 0;
    float y = ry;
    float p1 = ry * ry - (rx * rx) * ry + (rx * rx) * (0.25);
    float dx = 2 * (ry * ry) * x;
    float dy = 2 * (rx * rx) * y;
    glColor3ub(rand() % 255, rand() % 255, rand() % 255);
    while (dx < dy)
    {
        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);
        }
    }
    glFlush();

    float p2 = (ry * ry) * (x + 0.5) * (x + 0.5) + (rx * rx) * (y
        - 1) * (y - 1) - (rx * rx) * (ry * ry);
    glColor3ub(rand() % 255, rand() % 255, rand() % 255);
    while (y > 0)
    {
        setPixel(xCenter + x, yCenter + y);
        setPixel(xCenter - x, yCenter + y);
        setPixel(xCenter + x, yCenter - y);
        setPixel(xCenter - x, yCenter - y);
        if (p2 > 0)
        {
            x = x;
            y = y - 1;
            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);
        }
    }
    glFlush();
}
void display()
```

```
{
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(1.0, 0.0, 0.0);
        glPointSize(2.0);
        ellipseMidPoint();
        glFlush();
}
int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitWindowSize(640, 480);
        glutInitWindowPosition(10, 10);
        glutCreateWindow("User_Name");
        myinit();
        glutDisplayFunc(display);
        glutMainLoop();
        return 0;
}
```

output:-

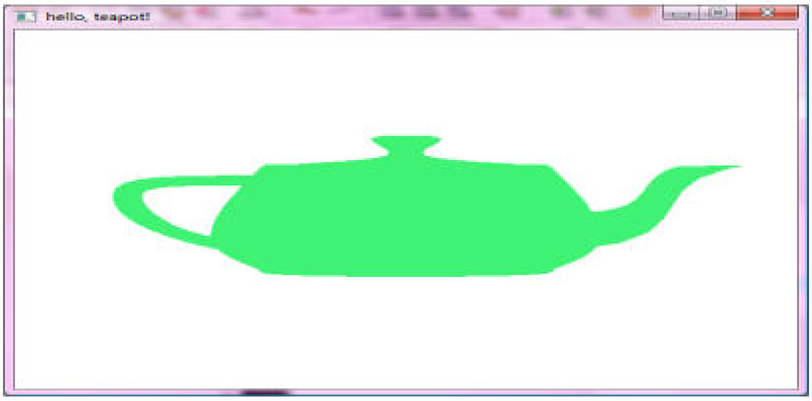


PROGRAM – 12 -To display Three Dimensional Object (Teapot)

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

#include <GL/glut.h>

void display ()

{

/* clear window */

  glClear(GL_COLOR_BUFFER_BIT);

  glColor3f(0.4,0.9,0.2);

/* draw scene */

  glutSolidTeapot(.5);

/* flush drawing routines to the window */

  glFlush();

}

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

```
/* initialize GLUT, using any commandline parameters passed to the program */
  glutInit(&argc,argv);
/* setup the size, position, and display mode for new windows */
  glutInitWindowSize(500,500);
  glutInitWindowPosition(0,0);
  glutInitDisplayMode(GLUT_RGB);
/* create and set up a window */
  glutCreateWindow("hello, teapot!");
  glutDisplayFunc(display);
/* tell GLUT to wait for events */
  glutMainLoop();
}
```

output:-



Solid Teapot

PROGRAM – 13 - To display Three Dimensional Objects and use of mouse events

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <GL/glut.h>
#define ROT_INC 0.1
void drawSphere(void);
static GLfloat g_rotate = 0;
static GLfloat g_rotInc = ROT_INC;
static void (*drawPrimP)(void) = drawSphere;
void drawSphere(void)
{
```

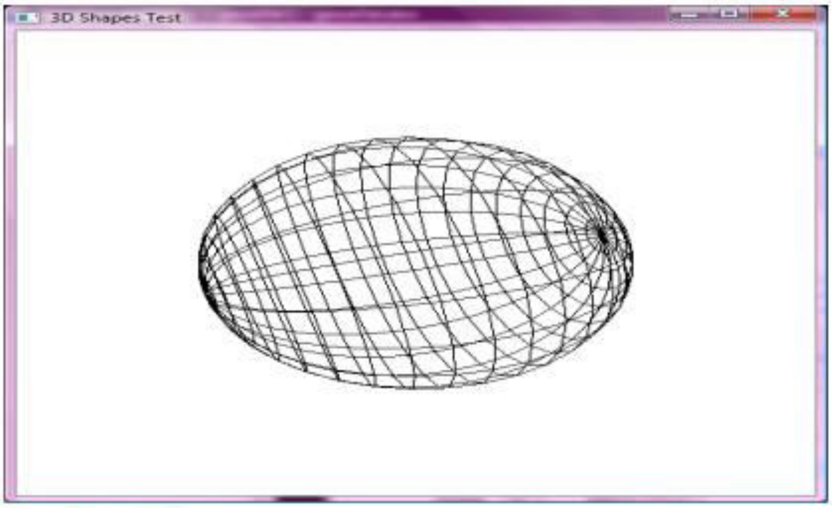
```
glutWireSphere(6.0,20,20);

}

void drawCube(void)

{

glutWireCube(6.0);

}

void drawCone(void)

{

glutWireCone(6.0, 8.0, 10, 20);

}

void drawTorus(void)

{

glutWireTorus(1.0, 6.0, 10, 20);

}

void drawTeapot(void)

{

glutWireTeapot(6.0);

}

void setPrim(int value)

{

switch(value)

  {

case 1:

drawPrimP = drawSphere;

break;

case 2:

drawPrimP = drawCube;

break;

case 3:

drawPrimP = drawCone;

break;

case 4:

drawPrimP = drawTorus;

break;
```

```
case 5:
drawPrimP = drawTeapot;
break;
}
}
void display(void)
{
  glClear( GL_COLOR_BUFFER_BIT );
glMatrixMode(GL_MODELVIEW); /* set matrix mode to modelview */
glPushMatrix(); /* save matrix */
glRotatef(g_rotate,1.0,1.0,1.0); /* global rotation */
(*drawPrimP)(); /* draw the geometry */
glPopMatrix(); /* restore matrix */
glutSwapBuffers(); /* swap buffers to display the frame */
}
void myReshape(int w, int h)
{
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity(); /* init projection matrix */
/* perspective parameters: field of view, aspect, near clip, far clip */
gluPerspective( 60.0, (GLdouble)w/(GLdouble)h, 0.1, 40.0);
  glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(0.0, 0.0, 20.0, /* eye at (0,0,20) */
  0.0, 0.0, 0.0, /* lookat point */
  0.0, 1.0, 0.0); /* up is in +ive y */
}
void myKey(unsigned char k, int x, int y)
{
switch (k) {
case 'q':
case 'Q': exit(0);
break;
```

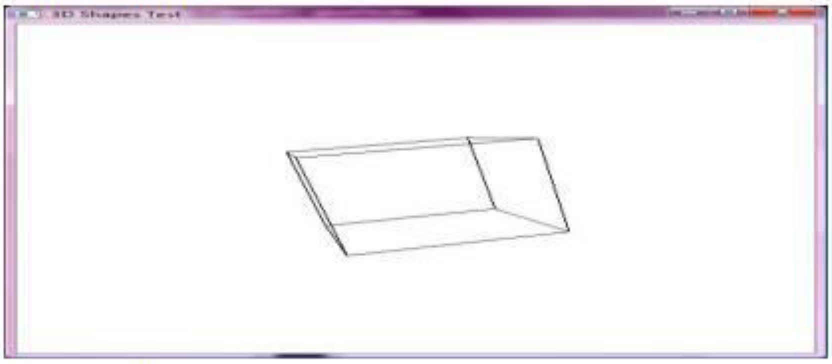
```
default:

printf("Unknown keyboard command \'%c\'.\n", k);

break;

  } }

void myMouse(int btn, int state, int x, int y)

{

  if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN) g_rotInc += ROT_INC;

  if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) g_rotInc = ROT_INC;

  glutPostRedisplay();

}

void myIdleFunc(void)

{

g_rotate += g_rotInc;

glutPostRedisplay();

}

int main(int argc, char **argv)

{

  glutInit(&argc, argv);

  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);

  glutInitWindowSize(500, 500);

  glutCreateWindow("3D Shapes Test");

  glutReshapeFunc(myReshape);

  glutDisplayFunc(display);

glutIdleFunc(myIdleFunc);

glutKeyboardFunc(myKey);

glutMouseFunc(myMouse);

/* set up right menu */

glutCreateMenu(setPrim);

glutAddMenuEntry("Sphere", 1);

glutAddMenuEntry("Cube", 2);

glutAddMenuEntry("Cone", 3);

glutAddMenuEntry("Torus", 4);

glutAddMenuEntry("Teapot", 5);

glutAttachMenu(GLUT_RIGHT_BUTTON);
```

```
glClearColor(1.0, 1.0, 1.0, 1.0); /* set clear colour */

glColor3f(0.0, 0.0, 0.0); /* set current colour to black */

glutMainLoop();

return 0;

}
```

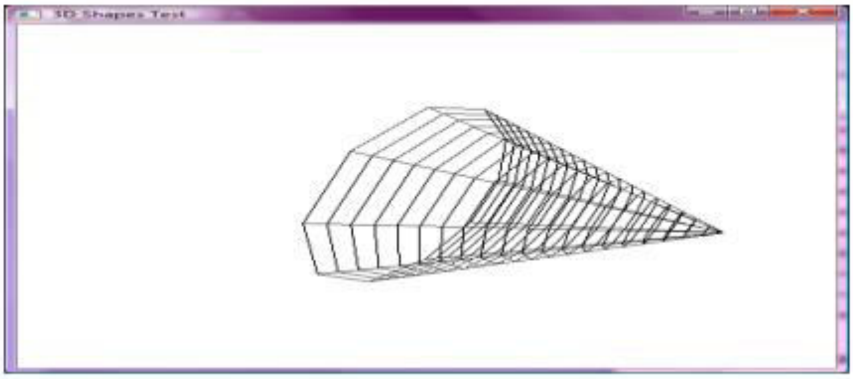
Output:-



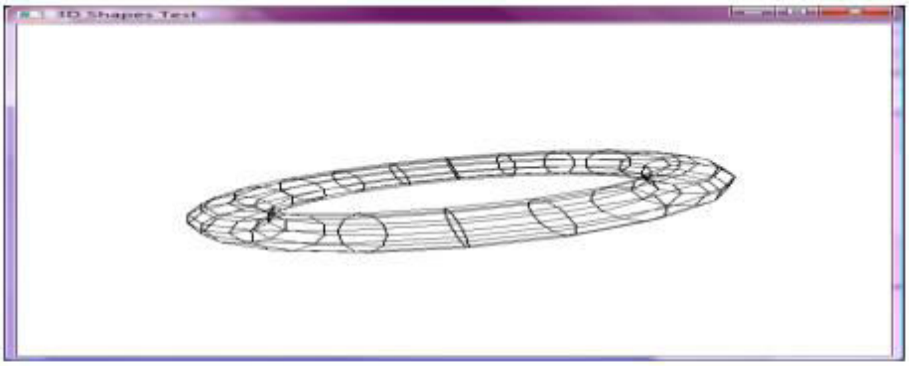
Case 1: Sphere



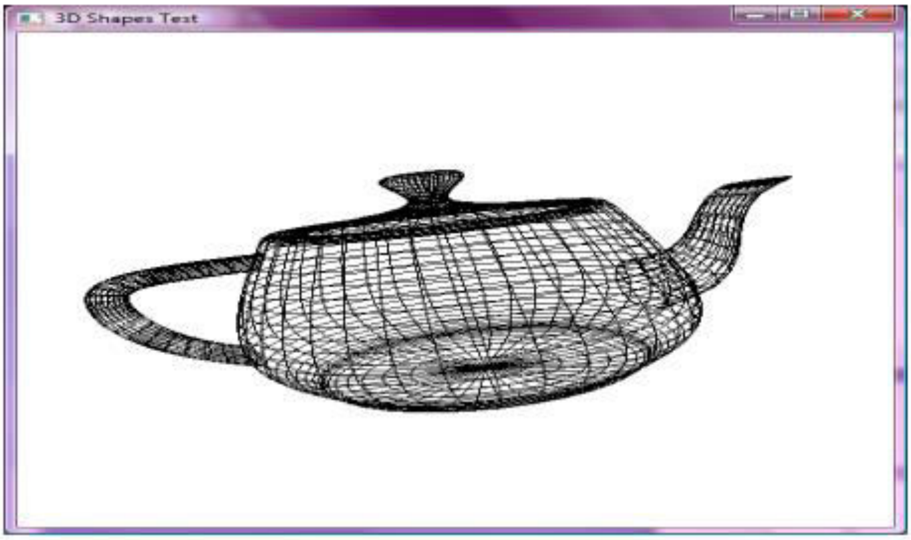
Case 2: Cube



Case 3: Cone



Case 4: Torus



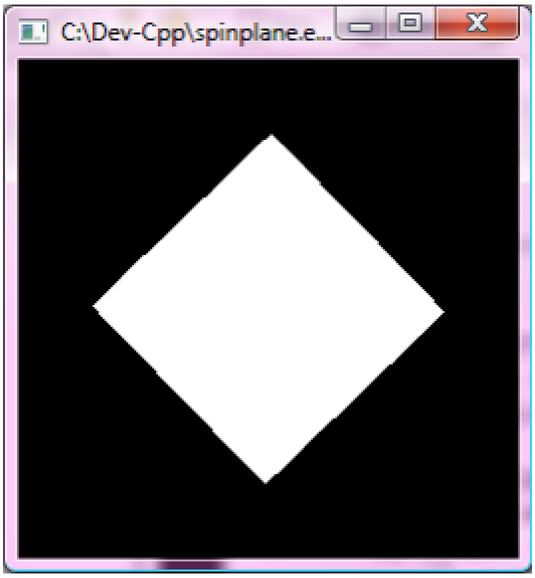
Case 5: Teapot

PROGRAM – 14 - To display Three Dimensional Spinning Plane

```
#include <GL/glut.h>
#include <stdlib.h>
static GLfloat spin = 0.0;
void display(void)
{
  glClear(GL_COLOR_BUFFER_BIT);
glPushMatrix();
  glRotatef(spin, 0.0, 0.0, 1.0);
  glColor3f(1.0, 1.0, 1.0);
  glRectf(-25.0, -25.0, 25.0, 25.0);
  glPopMatrix();
  glutSwapBuffers();
}
void spinDisplay(void)
{
  spin = spin + 2.0;
  if (spin > 360.0)
  spin = spin - 360.0;
  glutPostRedisplay();
}
void init(void)
{
  glClearColor (0.0, 0.0, 0.0, 0.0);
  glShadeModel (GL_FLAT);
}
void reshape(int w, int h)
{
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(-50.0, 50.0, -50.0, 50.0, -1.0, 1.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}
void mouse(int button, int state, int x, int y)
{
  switch (button)
  {
  case GLUT_LEFT_BUTTON:
  if (state == GLUT_DOWN)
  glutIdleFunc(spinDisplay);
  break;
  case GLUT_MIDDLE_BUTTON:
  case GLUT_RIGHT_BUTTON:
  if (state == GLUT_DOWN)
  glutIdleFunc(NULL);
  break;
  default:
  break;
  }
```

```
}

/* Request double buffer display mode. Register mouse input callback functions */

int main(int argc, char** argv)

{

  glutInit(&argc, argv);

  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);

  glutInitWindowSize (250, 250);

  glutInitWindowPosition (100, 100);

  glutCreateWindow (argv[0]);

  init ();

  glutDisplayFunc(display);

  glutReshapeFunc(reshape);

  glutMouseFunc(mouse);

  glutMainLoop();

  return 0; /* ANSI C requires main to return int. */

}
```

Output:



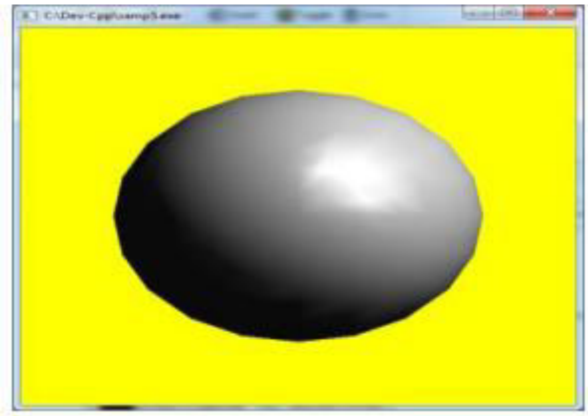
PROGRAM – 15 – To display 3D sphere with to show the illumination models

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

#include <GL/glu.h>

#include <GL/glut.h>

void init(void)

{
```

```
GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };

GLfloat mat_shininess[] = { 50.0 };

GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };

glClearColor (0.0, 0.0, 0.0, 0.0);

glShadeModel (GL_SMOOTH);

glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);

glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

glLightfv(GL_LIGHT0, GL_POSITION, light_position);

glEnable(GL_LIGHTING);

glEnable(GL_LIGHT0);

glEnable(GL_DEPTH_TEST);

}

void display(void)

{ glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glutSolidSphere (1.0, 20, 16);

  glFlush ();

}

void reshape (int w, int h)

{

glViewport (0, 0, (GLsizei) w, (GLsizei) h);

glMatrixMode (GL_PROJECTION);

glLoadIdentity();

if (w <= h)

  glOrtho (-1.5, 1.5, -1.5*(GLfloat)h/(GLfloat)w,1.5*(GLfloat)h/(GLfloat)w, -10.0, 10.0);

  else

  glOrtho (-1.5*(GLfloat)w/(GLfloat)h,1.5*(GLfloat)w/(GLfloat)h, -1.5, 1.5, -10.0, 10.0);

glMatrixMode(GL_MODELVIEW);

glLoadIdentity();

}

int main(int argc, char** argv)

{

glutInit(&argc, argv);

glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);

glutInitWindowSize (500, 500);
```

```
glutInitWindowPosition (100, 100);

glutCreateWindow (argv[0]);

init ();

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutMainLoop();

return 0;

}
```

Output:

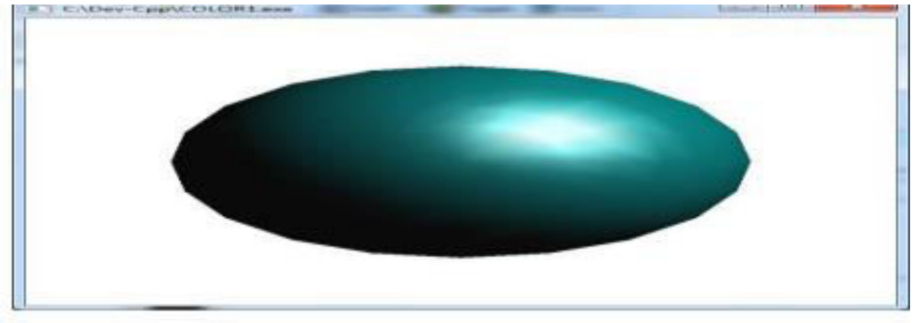


PROGRAM – 16 - To display 3D sphere with varying light and color intensities

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

#include <GL/glu.h>

#include <GL/glut.h>

GLfloat diffuseMaterial[4] = { 0.5, 0.5, 0.5, 1.0 };

void init(void)

{

GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };

GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };

glClearColor (0.0, 0.0, 0.0, 0.0);

glShadeModel (GL_SMOOTH);

glEnable(GL_DEPTH_TEST);

glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuseMaterial);

glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);

glMaterialf(GL_FRONT, GL_SHININESS, 25.0);

glLightfv(GL_LIGHT0, GL_POSITION, light_position);

glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glColorMaterial(GL_FRONT, GL_DIFFUSE);
glEnable(GL_COLOR_MATERIAL);
}
void display(void)
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glutSolidSphere(1.0, 20, 16);
glFlush ();
}
void reshape (int w, int h)
{
glViewport (0, 0, (GLsizei) w, (GLsizei) h);
glMatri xMode (GL_PROJECTION);
glLoadIdentity();
  if (w <= h)
  glOrtho (-1.5, 1.5, -1.5*(GLfloat)h/(GLfloat)w,1.5*(GLfloat)h/(GLfloat)w, -10.0, 10.0);
  else
  glOrtho (-1.5*(GLfloat)w/(GLfloat)h,1.5*(GLfloat)w/(GLfloat)h, -1.5, 1.5, -10.0, 10.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
}
void mouse(int button, int state, int x, int y)
{
switch (button)
{
case GLUT_LEFT_BUTTON:
if (state == GLUT_DOWN) /* change red */
{
  diffuseMaterial[0] += 0.1;
if (diffuseMaterial[0] > 1.0)
diffuseMaterial[0] = 0.0;
glColor4fv(diffuseMaterial);
glutPostRedisplay();
}
break;
case GLUT_MIDDLE_BUTTON:
if (state == GLUT_DOWN) /* change green */
{
diffuseMaterial[1] += 0.1;
  if (diffuseMaterial[1] > 1.0)
diffuseMaterial[1] = 0.0;
glColor4fv(diffuseMaterial);
glutPostRedisplay();
}
break;
case GLUT_RIGHT_BUTTON:
  if (state == GLUT_DOWN) /* change blue */
{
diffuseMaterial[2] += 0.1;
  if (diffuseMaterial[2] > 1.0)
diffuseMaterial[2] = 0.0;
glColor4fv(diffuseMaterial);
glutPostRedisplay();
}
break;
default:
break; } }
int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowSize (500, 500);
glutInitWindowPosition (100, 100);
glutCreateWindow (argv[0]);
init ();
glutDisplayFunc(display);
glutReshapeFunc(reshape);
```

```
glutMouseFunc(mouse);

glutMainLoop();

return 0;

}
```

Output:

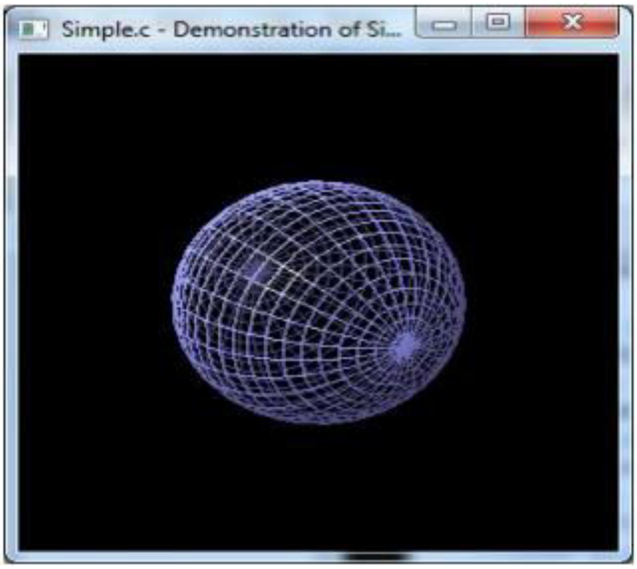


PROGRAM – 17 - To demonstrates simple object drawing with perspective

```
#include <stdio.h>

#include <gl/glut.h> // GLUT toolkit

// Define initial camera position and viewing window values

#define INIT_VIEW_X 0.0

#define INIT_VIEW_Y 0.0

#define INIT_VIEW_Z -4.5

#define VIEW_LEFT -2.0

#define VIEW_RIGHT 2.0

#define VIEW_BOTTOM -2.0

#define VIEW_TOP 2.0

#define VIEW_NEAR 1.0

#define VIEW_FAR 200.0

// My initialization values for lighting

GLfloat AmbientLight[] = { 0.3f, 0.3f, 0.3f, 1.0f };

GLfloat DiffuseLight[] = { 0.8f, 0.8f, 0.8f, 1.0f };

GLfloat SpecularLight[] = { 1.0f, 1.0f, 1.0f, 1.0f };

GLfloat SpecRef[] = { 0.7f, 0.7f, 0.7f, 1.0f };

GLfloat LightPos[] = {-50.0f,50.0f,100.0f,1.0f};

GLubyte Shine = 128;

//Display the current object using My Image's data

void DisplayObject(void)
```

```
{
// Clear the window
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glColor3ub(150, 150, 250); // Change the draw color to slate blue
glPushMatrix(); //Save viewing matrix state
glRotatef(45.0,1,1,1); // Rotate 45 degrees on each axis to show a little depth
glutWireSphere(1.0f, 30, 30); //Draw WireFrame Sphere
glPopMatrix(); // Restore matrix state
glutSwapBuffers(); // Flush drawing commands
} // End of DisplayObject
void SetupRend()
{
glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set background to black
glEnable(GL_DEPTH_TEST); // Enable depth testing
glEnable(GL_LIGHTING); // Enable lighting
glLightfv(GL_LIGHT0, GL_AMBIENT, AmbientLight);
glLightfv(GL_LIGHT0, GL_DIFFUSE, DiffuseLight);
glLightfv(GL_LIGHT0, GL_SPECULAR, SpecularLight);
glEnable(GL_LIGHT0);
glEnable(GL_COLOR_MATERIAL); // Enable color tracking
// Set material to folow glColor values
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
// Set specular reflectivity and shine
glMaterialfv(GL_FRONT, GL_SPECULAR, SpecRef);
glMateriali(GL_FRONT, GL_SHININESS, Shine);
}
void ChangeWindow(GLsizei w, GLsizei h) // Set a perspective window
{
GLfloat Ratio;
if (h==0) // Prevent division by zero
  h=1;
glViewport(0,0,w,h); // Set viewport to window dimensions
Ratio = (GLfloat)w/(GLfloat)h; // Set the perspective ratios
glMatrixMode( GL_PROJECTION ); // Reset coordinate system
glLoadIdentity();
gluPerspective(50.0f, Ratio, VIEW_NEAR, VIEW_FAR); // Set the viewing
perspective
glMatrixMode( GL_MODELVIEW ); // Set viewing translation
glLoadIdentity();
glTranslatef( INIT_VIEW_X, INIT_VIEW_Y, INIT_VIEW_Z );
glLightfv(GL_LIGHT0, GL_POSITION, LightPos);
}
// Main program module
int main(void)
{
// Set the buffers we want for best viewing
glutInitDisplayMode ( GLUT_DOUBLE // double-buffered pixel format for flicker-free movement
| GLUT_RGB // use RGB pixel format (easy to color)
| GLUT_DEPTH); // use depth buffer (better perspective)
// Create the viewing window
glutCreateWindow ("Simple.c - Demonstration of Simple C++ OpenGL Code");
glutReshapeFunc (ChangeWindow); // Set function for resizing window
glutDisplayFunc (DisplayObject); // Set function for redisplaying
SetupRend(); // Initialize and get ready to draw
glutMainLoop(); // Keep on going until user gets tired
return 0;
}
```

Output:



PROGRAM – 18 - To draw multiple spheres with different illumination

models

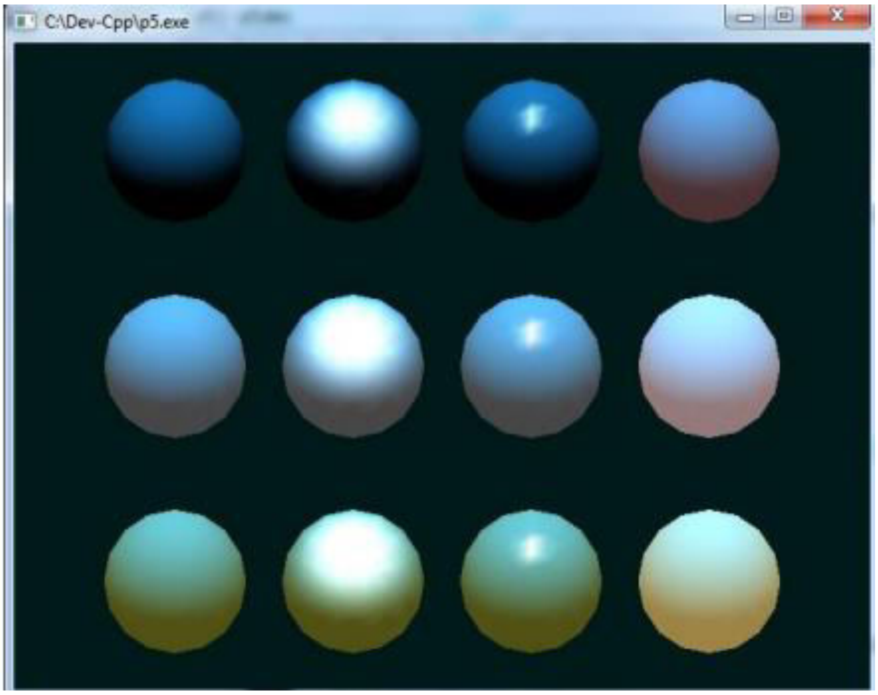
```
#include <stdlib.h>

#include <GL/glut.h>

void init(void)

{

  GLfloat ambient[] = { 0.0, 0.0, 0.0, 1.0 };

  GLfloat diffuse[] = { 1.0, 1.0, 1.0, 1.0 };

  GLfloat specular[] = { 1.0, 1.0, 1.0, 1.0 };

  GLfloat position[] = { 0.0, 3.0, 2.0, 0.0 };

  GLfloat lmodel_ambient[] = { 0.4, 0.4, 0.4, 1.0 };

  GLfloat local_view[] = { 0.0 };

  glClearColor(0.0, 0.1, 0.1, 0.0);

  glEnable(GL_DEPTH_TEST);

  glShadeModel(GL_SMOOTH);

glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);

  glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);

  glLightfv(GL_LIGHT0, GL_POSITION, position);

  glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodel_ambient);

  glLightModelfv(GL_LIGHT_MODEL_LOCAL_VIEWER, local_view);

  glEnable(GL_LIGHTING);

  glEnable(GL_LIGHT0); }

void display(void)

{

  GLfloat no_mat[] = { 0.0, 0.0, 0.0, 1.0 };

  GLfloat mat_ambient[] = { 0.7, 0.7, 0.7, 1.0 };

  GLfloat mat_ambient_color[] = { 0.8, 0.8, 0.2, 1.0 };

  GLfloat mat_diffuse[] = { 0.1, 0.5, 0.8, 1.0 };

  GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };

  GLfloat no_shininess[] = { 0.0 };

  GLfloat low_shininess[] = { 5.0 };

  GLfloat high_shininess[] = { 100.0 };

  GLfloat mat_emission[] = {0.3, 0.2, 0.2, 0.0};

  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glPushMatrix();
```

```
glTranslatef (-3.75, 3.0, 0.0);

glMaterialfv(GL_FRONT, GL_AMBIENT, no_mat);

glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);

glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);

glMaterialfv(GL_FRONT, GL_SHININESS, no_shininess);

glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);

glutSolidSphere(1.0, 16, 16);

glPopMatrix();

glPushMatrix();

glTranslatef (-1.25, 3.0, 0.0);

glMaterialfv(GL_FRONT, GL_AMBIENT, no_mat);

glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);

glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);

glMaterialfv(GL_FRONT, GL_SHININESS, low_shininess);

glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);

glutSolidSphere(1.0, 16, 16);

glPopMatrix();

glPushMatrix();

glTranslatef (1.25, 3.0, 0.0);

glMaterialfv(GL_FRONT, GL_AMBIENT, no_mat);

glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);

glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);

glMaterialfv(GL_FRONT, GL_SHININESS, high_shininess);

glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);

glutSolidSphere(1.0, 16, 16);

glPopMatrix();

glPushMatrix();

glTranslatef (3.75, 3.0, 0.0);

glMaterialfv(GL_FRONT, GL_AMBIENT, no_mat);

glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);

glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);

glMaterialfv(GL_FRONT, GL_SHININESS, no_shininess);

glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);

glutSolidSphere(1.0, 16, 16);
```

```
glPopMatrix();
glPushMatrix();
glTranslatef (-3.75, 0.0, 0.0);
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);
glMaterialfv(GL_FRONT, GL_SHININESS, no_shininess);
glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);
glutSolidSphere(1.0, 16, 16);
glPopMatrix();
glPushMatrix();
glTranslatef (-1.25, 0.0, 0.0);
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, low_shininess);
glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);
glutSolidSphere(1.0, 16, 16);
glPopMatrix();
glPushMatrix();
glTranslatef (1.25, 0.0, 0.0);
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, high_shininess);
glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);
glutSolidSphere(1.0, 16, 16);
glPopMatrix();
glPushMatrix();
glTranslatef (3.75, 0.0, 0.0);
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);
glMaterialfv(GL_FRONT, GL_SHININESS, no_shininess);
glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
glutSolidSphere(1.0, 16, 16);
glPopMatrix();
glPushMatrix();
glTranslatef (-3.75, -3.0, 0.0);
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient_color);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);
glMaterialfv(GL_FRONT, GL_SHININESS, no_shininess);
glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);
glutSolidSphere(1.0, 16, 16);
glPopMatrix();
glPushMatrix();
glTranslatef (-1.25, -3.0, 0.0);
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient_color);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, low_shininess);
glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);
glutSolidSphere(1.0, 16, 16);
glPopMatrix();
glPushMatrix();
glTranslatef (1.25, -3.0, 0.0);
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient_color);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, high_shininess);
glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);
glutSolidSphere(1.0, 16, 16);
glPopMatrix();
glPushMatrix();
glTranslatef (3.75, -3.0, 0.0);
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient_color);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
```

```
  glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);
  glMaterialfv(GL_FRONT, GL_SHININESS, no_shininess);
  glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
  glutSolidSphere(1.0, 16, 16);
  glPopMatrix();
  glFlush();
}
void reshape(int w, int h)
{
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  if (w <= (h * 2))
  glOrtho (-6.0, 6.0, -3.0*((GLfloat)h*2)/(GLfloat)w, 3.0*((GLfloat)h*2)/(GLfloat)w, -10.0, 10.0);
  else
  glOrtho (-6.0*(GLfloat)w/((GLfloat)h*2), 6.0*(GLfloat)w/((GLfloat)h*2), -3.0, 3.0, -10.0, 10.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}
void keyboard(unsigned char key, int x, int y)
{
  switch (key)
{
  case 27:
  exit(0);
  break;
  }
}
int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowSize (600, 450);
  glutCreateWindow(argv[0]);
  init();
  glutReshapeFunc(reshape);
  glutDisplayFunc(display);
  glutKeyboardFunc (keyboard);
  glutMainLoop();
  return 0;
}
```

Output:



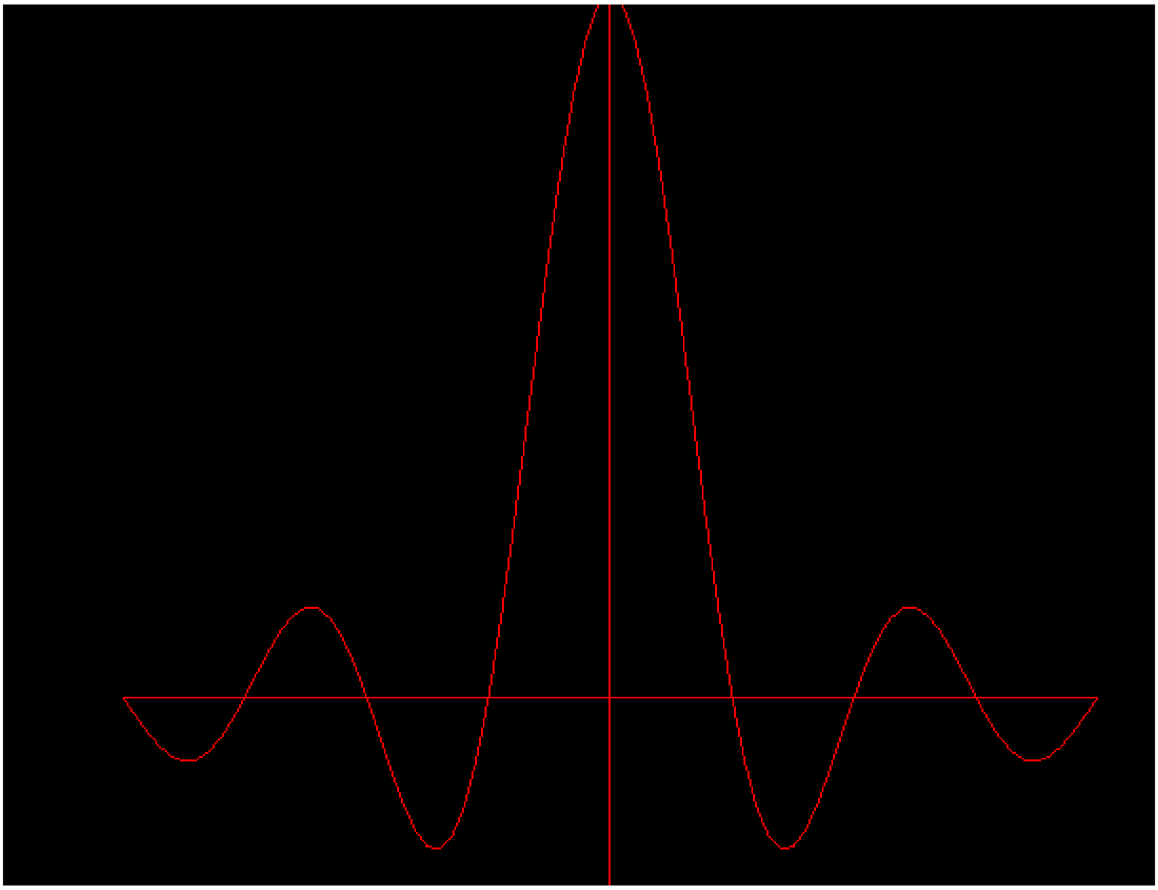
PROGRAM – 19-OPENGL PROGRAM TO DISPLAY SINC FUNCTION

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

const int screenWidth = 640;
const int screenHeight = 480;

void myInit()
{
        glClearColor(0.0, 0.0, 0.0, 0.0); //black background color
        glColor3f(1.0, 0.0, 0.0); // red foreground color
        glPointSize(3.0); // 3x3 pixel size
}
```

```
//set window
void setWindow(float left, float right, float bottom, float top)
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(left, right, bottom, top);
}

//set view port
void setViewport(float left, float right, float bottom, float top)
{
        glViewport(left, bottom, right - left, top - bottom);
}
void drawCoordinates() {

        // draw some lines
        //glColor3f(1.0, 1.0, 0.0); // yellow x
        glBegin(GL_LINES);
        // x axis

        glVertex3f(-4.0, 0.0f, 0.0f);
        glVertex3f(4.0, 0.0f, 0.0f);

        // y axis
        //glColor3f(0.0, 1.0, 0.0); // green y
        glBegin(GL_LINES);
        glVertex3f(0.0, -4.0f, 0.0f);
        glVertex3f(0.0, 4.0f, 0.0f);

        glEnd();
}
void myDisplay()    // plot the sinc function, using world coordinates
{
        glClear(GL_COLOR_BUFFER_BIT); //clear the screen
```

```
    setWindow(-5.0, 5.0, -0.3, 1.0);    //set the window
    setViewport(0, screenWidth,0,screenHeight); //set the viewport
    glBegin(GL_LINE_STRIP);
    for (GLfloat x = -4.0; x < 4.0; x += 0.1)    //draw the plot
        glVertex2f(x, sin(3.14159 * x) / (3.14159 * x));
    glEnd();
    drawCoordinates();
    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow("sinc function = sin(PI*X)/PI*X");
    glutDisplayFunc(myDisplay);
    myInit();
    glutMainLoop();
}
```

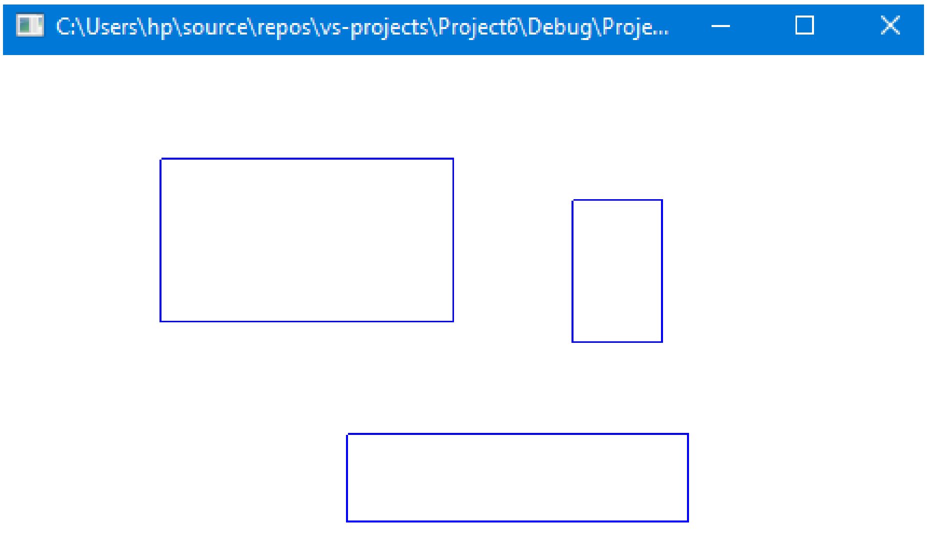
output:-



PROGRAM – 20- Demo program showing rubber rectangular drawing using xor raster operation.

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

#include <stdlib.h>

#include <stdio.h>


int window;

int screenWidth = 500, screenHeight = 500;

int left, bottom, right, top;                               //rectangular area


void init(void)

{

      glClearColor (1, 1, 1, 0.0);

      glPolygonMode( GL_FRONT, GL_LINE );

      glPolygonMode( GL_BACK, GL_LINE );

      glMatrixMode( GL_PROJECTION );

      glLoadIdentity();

      gluOrtho2D( 0.0, screenWidth, 0.0, screenHeight );   //set to screen values to simplify

}


void display(void)

{

      glClear(GL_COLOR_BUFFER_BIT);

      glColor3f ( 1, 0, 0 );

      //glBegin( GL_TRIANGLES);               //arbitrarily draw something

      //    glVertex2i ( 100, 100 );

      //    glVertex2i ( 200, 100 );

      //    glVertex2i ( 150, 150 );

      //glEnd();

}


void keyboard(unsigned char key, int x, int y)

{

    switch(key) {
        case 27: /* escape */
                glutDestroyWindow(window);
                exit(0);
    }
}

void myMouse( int button, int state, int x, int y )
{
    if ( button == GLUT_LEFT_BUTTON && state == GLUT_DOWN ) {
        glEnable( GL_COLOR_LOGIC_OP );  //enable logical operations
        glLogicOp ( GL_XOR );           //set it to XOR mode
        left = right = x;               //set the pivot
        top = bottom = screenHeight - y;
    } else if ( button == GLUT_LEFT_BUTTON && state == GLUT_UP ) {
        glDisable( GL_COLOR_LOGIC_OP ); //disable logical operations
        glColor3f ( 0.0, 0.0, 1.0 );
        glRecti ( left, bottom, right, top );//draw the final rectangle
        glFlush();
    }
}

void mouseMove ( int mx, int my )
{
    glRecti ( left, bottom, right, top );           //erase old rectangle
    right = mx;
    bottom = screenHeight - my;                     //flip y-coordinates
    glColor3f ( 0, 1, 1 );
    glRecti ( left, bottom, right, top );           //draw the new rectangle
    glFlush();
}

int main(int argc, char** argv)
{
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB );

      glutInitWindowSize(screenWidth, screenHeight);

      glutInitWindowPosition(100, 100);

      window = glutCreateWindow(argv[0]);

      init();

      glutDisplayFunc(display);

      glutMouseFunc( myMouse );

      glutMotionFunc( mouseMove );

      glutKeyboardFunc(keyboard);

      glutMainLoop();

      return 0;

}
```

output:-


C:\Users\hp\source\repos\vs-projects\Project6\Debug\Proje...

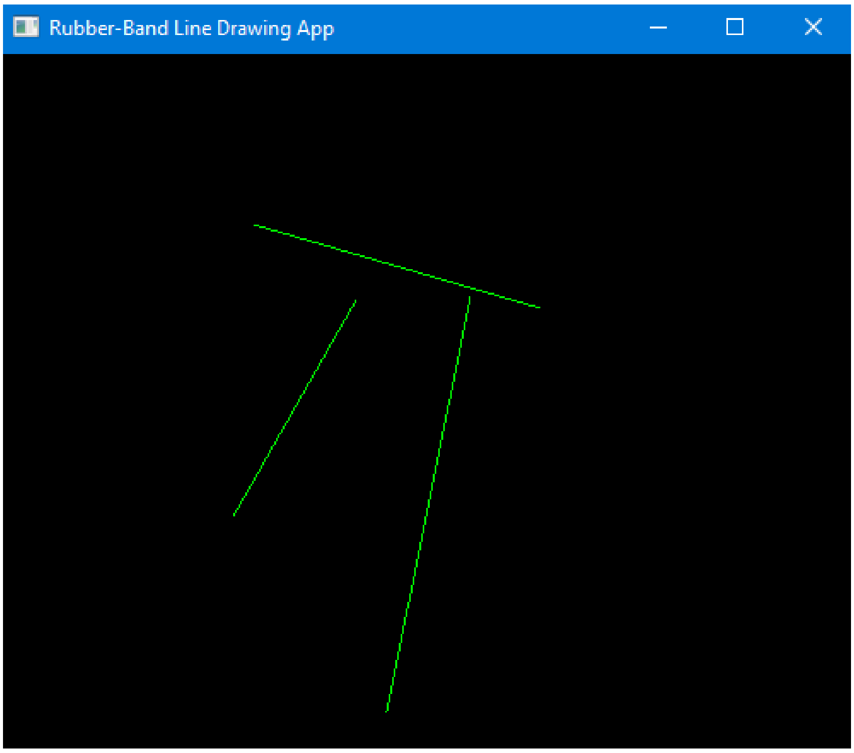
PROGRAM – 21-Rubber-band line draw with open_gl & glut

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

#include <stdio.h>

int numLines;

enum state

{

        waitingforclick,

        clickedone,

};

typedef struct point

{
        int x;
        int y;
}point;

point lines[256][2];

int gState = waitingforclick;
bool lineisvalid = false;
int gHeight;
float gColor[3] = { 0,1,0 };
void drawlines()
{
        glColor3fv(gColor);
        glBegin(GL_LINES);
        for (int i = 0; i <= numLines; i++)
        {
                glVertex2i(lines[i][0].x, gHeight - lines[i][0].y);
                glVertex2i(lines[i][1].x, gHeight - lines[i][1].y);
        }
        glEnd();
}
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        drawlines();
        glutSwapBuffers();
}
void menufunc(int val)
{
        switch (val)
        {
        case 0:
                gColor[0] = 1;
                gColor[1] = 0;
```

```
            gColor[2] = 0;
            break;
        case 1:
            gColor[0] = 0;
            gColor[1] = 1;
            gColor[2] = 0;
            break;
        case 2:
            gColor[0] = 0;
            gColor[1] = 0;
            gColor[2] = 1;
            break;
        }
}
void createMenu()
{
        int menu = glutCreateMenu(menufunc);
        glutAddMenuEntry("Red", 0);
        glutAddMenuEntry("Green", 1);
        glutAddMenuEntry("Blue", 2);
        glutAttachMenu(GLUT_RIGHT_BUTTON);
}
void init()
{
        glClearColor(0, 0, 0, 1);
        glMatrixMode(GL_PROJECTION);
        glOrtho(-1, 1.0, -1, 1.0, -1.0, 1.0);
        numLines = -1;
        glMatrixMode(GL_MODELVIEW);
        createMenu();
}
void reshape(int width, int height)
{
        gHeight = height;
```

```
	glMatrixMode(GL_PROJECTION);

	glLoadIdentity();

	gluOrtho2D(0, width, 0, height);

	glMatrixMode(GL_MODELVIEW);

}


void mouseclick(int button, int state, int x, int y)

{

	if (button == GLUT_LEFT_BUTTON && state == GLUT_UP)

	{

		switch (gState)

		{


		case waitingforclick:

			printf("     1st click");

			++numLines;

			lines[numLines][0].x = x;

			lines[numLines][0].y = y;

			lines[numLines][1].x = x;

			lines[numLines][1].y = y;

			gState++;

			break;

		case clickedone:

			printf("     2nd click");

			lines[numLines][1].x = x;

			lines[numLines][1].y = y;

			gState = waitingforclick;

			break;

		}

	}

	glutPostRedisplay();

}

void mousedrag(int x, int y)

{
        if (gState == clickedone)
        {
                lines[numLines][1].x = x;
                lines[numLines][1].y = y;
        }
        glutPostRedisplay();
}
int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
        glutInitWindowPosition(100, 100);
        glutInitWindowSize(500, 500);
        glutCreateWindow("Rubber-Band Line Drawing App");
        glutReshapeFunc(reshape);
        glutDisplayFunc(display);
        glutMouseFunc(mouseclick);
        glutPassiveMotionFunc(mousedrag);
        glutPostRedisplay(); //added function for speed and clarity
        init();
        glutMainLoop();
        return 0;
}
```

output:-



PROGRAM – 22- Program for simulation of Towers of Hanoi

```
#include <iostream>

#include<stdio.h>

#include <GL/glut.h>

#include <cmath>

#include <list>


#define NUM_DISCS 5

#define ROD_HEIGHT 4

#define WINDOW_WIDTH 1350

#define WINDOW_HEIGHT 690

#define PI 22/7.0f

#define DISC_SPACING 0.35

#define BOARD_X 10

#define BOARD_Y 5


#include<GL/glut.h>



//Keep DISC_SPACING between 0.3 to 0.6 for best results


using namespace std;
```

```
struct Vector3 {
        double x, y, z;
        Vector3() { x = y = z = 0.0; }
        Vector3(double x, double y, double z) : x(x), y(y), z(z) { }
        Vector3(Vector3 const& rhs) { *this = rhs; }
        Vector3& operator= (Vector3 const& rhs)
        {
                x = rhs.x;
                y = rhs.y;
                z = rhs.z;
                return *this;
        }
};

struct Disc {
        Disc() { normal = Vector3(0.0, 0.0, 1.0); }

        Vector3 position; //location
        Vector3 normal;    //orientation
};

struct ActiveDisc {     //Active Disc to be moved [later in motion]
        int disc_index;
        Vector3 start_pos, dest_pos;
        double u;                    // u E [0, 1]
        double step_u;
        bool is_in_motion;
        int direction;      // +1 for Left to Right & -1 for Right to left, 0 = stationary
};


// Rods and Discs Globals - Can be changed for different levels
```

```
struct Rod {
        Vector3 positions[NUM_DISCS];
        int occupancy_val[NUM_DISCS];
};

struct GameBoard {
        double x_min, y_min, x_max, y_max; //Base in XY-Plane
        double rod_base_rad;                       //Rod's base radius
        Rod rods[3];
};

struct solution_pair {
        size_t f, t;            //f = from, t = to
};

//Game Globals
Disc discs[NUM_DISCS];
GameBoard t_board;
ActiveDisc active_disc;
list<solution_pair> sol;
bool to_solve = false;

//Globals for window, time, FPS, moves
float SPEED = 2 ;
int FPS = int(30 * SPEED);
int moves = 0;
int prev_time = 0;
int window_width = WINDOW_WIDTH, window_height = WINDOW_HEIGHT;

void initialize();
void initialize_game();
void display_handler();
void reshape_handler(int w, int h);
void keyboard_handler(unsigned char key, int x, int y);
void anim_handler();
void move_disc(int from_rod, int to_rod);
Vector3 get_inerpolated_coordinate(Vector3 v1, Vector3 v2, double u);
void move_stack(int n, int f, int t);

int main2()
{

        // Initialize GLUT Window
            //glutInit(&argc, argv);
            glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
            glutInitWindowSize(window_width, window_height);
            glutInitWindowPosition(0, 0);
            glutCreateWindow("Towers of Hanoi");
            glutDestroyWindow(1);

            initialize();
            cout << "Towers of Hanoi" << endl;
            cout << "Press H for Help" << endl;

            //Callbacks
            glutDisplayFunc(display_handler);
            glutReshapeFunc(reshape_handler);
            glutKeyboardFunc(keyboard_handler);

            glutIdleFunc(anim_handler);

            glutMainLoop();
            return 0;
}

void initialize()
{
            glClearColor(0,0,0,0);
            //Setting the clear color
            //glShadeModel(GL_SMOOTH);
            //SMOOTH Shading

            glEnable(GL_DEPTH_TEST);                //Enabling Depth Test

            //Setting Light0 parameters
            GLfloat light0_pos[] = { 0.0f, 0.0f, 0.0f, 1.0f };
            // A positional light
            glLightfv(GL_LIGHT0, GL_POSITION, light0_pos);

            glEnable(GL_LIGHTING);
            //Enabling Lighting
            glEnable(GL_LIGHT0);
            //Enabling Light0



            //Globals initializations
            //prev_time = glutGet(GLUT_ELAPSED_TIME);

            //Initializing Game State
            initialize_game();

}

void initialize_game()
{
            //Initializing 1)GameBoard t_board 2) Discs discs   3) ActiveDisc active_disc State

            //1) Initializing GameBoard
            t_board.rod_base_rad = 1.0;
            t_board.x_min = 0.0;
            t_board.x_max = BOARD_X * t_board.rod_base_rad;
```

```
    t_board.y_min = 0.0;
    t_board.y_max = BOARD_Y * t_board.rod_base_rad;

    double x_center = (t_board.x_max - t_board.x_min) / 2.0;
    double y_center = (t_board.y_max - t_board.y_min) / 2.0;

    double dx = (t_board.x_max - t_board.x_min) / 3.0; //Since 3 rods
    double r = t_board.rod_base_rad;

//Initializing Rods Occupancy value
    for (int i = 0; i < 3; i++)
    {
            for (int h = 0; h < NUM_DISCS; h++)
            {
                    if (i == 0)
                    {
                            t_board.rods[i].occupancy_val[h] = NUM_DISCS - 1 - h;

                    }
                    else
                            t_board.rods[i].occupancy_val[h] = -1;

                            //printf("%d\t",t_board.rods[i].occupancy_val[h]);
            }
            //printf("\n");
    }



    //Initializing Rod positions
    for (int i = 0; i < 3; i ++)
    {
            for (int h = 0; h < NUM_DISCS; h++)
            {
                    double x = x_center + ((int)i - 1) * dx;
            double y = y_center;
            double z = (h + 1) * DISC_SPACING;
            Vector3& pos_to_set = t_board.rods[i].positions[h];
            pos_to_set.x = x;
            pos_to_set.y = y;
            pos_to_set.z = z;
            printf("%f %f %f \n",x,y,z);
        }
    }


    //2) Initializing Discs
    for (size_t i = 0; i < NUM_DISCS; i++)
    {
        discs[i].position = t_board.rods[0].positions[NUM_DISCS - i - 1];
        //Normals are initialized whie creating a Disc object - ie in constructor of Disc
    }

    //3) Initializing Active Disc
    active_disc.disc_index = -1;
    active_disc.is_in_motion = false;
    active_disc.step_u = 0.015;
    active_disc.u = 0.0;
    active_disc.direction = 0;
}

//Draw function for drawing a cylinder given position and radius and height
void draw_solid_cylinder(double x, double y, double r, double h)
{
    GLUquadric* q = gluNewQuadric();
    GLint slices = 50;
    GLint stacks = 10;

    glPushMatrix();
    glTranslatef(x, y, 0.0f);
    gluCylinder(q, r, r, h, slices, stacks);
    glTranslatef(0, 0, h);
    gluDisk(q, 0, r, slices, stacks);
    glPopMatrix();
    gluDeleteQuadric(q);
}

//Draw function for drawing rods on a given game board i.e. base
void draw_board_and_rods(GameBoard const& board)
{
    //Materials,
    GLfloat mat_white[] = { 1.0f, 1.0f, 1.0f, 1.0f };
    GLfloat mat_yellow[] = { 1.0f, 1.0f, 0.0f, 1.0f };

    glPushMatrix();
    //Drawing the Base Rectangle [where the rods are placed]
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, mat_white);
    glBegin(GL_QUADS);
        glNormal3f(0.0f, 0.0f, 1.0f);
        glVertex2f(board.x_min, board.y_min);
        glVertex2f(board.x_min, board.y_max);
        glVertex2f(board.x_max, board.y_max);
        glVertex2f(board.x_max, board.y_min);
    glEnd();

    //Drawing Rods and Pedestals
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, mat_yellow);

    double r = board.rod_base_rad;
    for (int i = 0; i < 3; i++)
    {
        Vector3 const& p = board.rods[i].positions[0];
```

```
                    draw_solid_cylinder(p.x, p.y, r * 0.1, ROD_HEIGHT - 0.1);

                    draw_solid_cylinder(p.x, p.y, r, 0.1);

          }


          glPopMatrix();

}




// Draw function for drawing discs

void draw_discs()

{

          int slices = 100;

          int stacks = 10;


          double rad;


          GLfloat r, g, b;

          GLfloat emission[] = { 0.4f, 0.4f, 0.4f, 1.0f };

          GLfloat no_emission[] = { 0.0f, 0.0f, 0.0f, 1.0f };

          GLfloat material[] = { 1.0f, 1.0f, 1.0f, 1.0f };

          for (size_t i = 0; i < NUM_DISCS; i++)

          {

                    switch (i)

                    {

                    case 0: r = 0; g = 0; b = 1;

                              break;

                    case 1: r = 0; g = 1; b = 0;

                              break;

                    case 2: r = 0, g = 1; b = 1;

                              break;

                    case 3 : r = 1; g = 0; b =0 ;

                              break ;

                    case 4 : r = 1; g = 0; b = 1;

                              break;
```

```
case 5 : r = 1; g = 1; b = 0;
        break;
case 6 : r = 1 ; g = 1 ; b = 1 ;
        break ;
default: r = g = b = 1.0f;
        break;
};

material[0] = r;
material[1] = g;
material[2] = b;
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, material);

GLfloat u = 0.0f;
//This part is written to highlight the disc in motion
if (i == active_disc.disc_index)
{
        glMaterialfv(GL_FRONT, GL_EMISSION, emission);
        u = active_disc.u;
}

GLfloat factor = 1.0f;
switch (i) {
        case 0: factor = 0.2;
                break;
        case 1: factor = 0.4;
                break;
        case 2: factor = 0.6;
                break;
        case 3: factor = 0.8;
                break;
        case 4 : factor = 1.2 ;
                break ;
        case 5 : factor = 1.4 ;
```

```
                break ;
            case 6 : factor = 1.6 ;
                break ;
            case 7 : factor = 1.8 ;
                break ;
            default: break;
        };
        rad = factor * t_board.rod_base_rad;
        int d = active_disc.direction;


        glPushMatrix();
        glTranslatef(discs[i].position.x, discs[i].position.y, discs[i].position.z);
        double theta = acos(discs[i].normal.z);
        theta *= 180.0f / PI;
        glRotatef(d * theta , 0.0f, 1.0f, 0.0f);
        glutSolidTorus(0.2 * t_board.rod_base_rad, rad, stacks, slices);
        glPopMatrix();

        glMaterialfv(GL_FRONT, GL_EMISSION, no_emission);
    }
}

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

    double x_center = (t_board.x_max - t_board.x_min) / 2.0;
    double y_center = (t_board.y_max - t_board.y_min) / 2.0;
    double r = t_board.rod_base_rad;

    static float view[] = {0,0,0} ;
    view[0]     = x_center ;
    view[1] = y_center - 10 ;
```

```
        view[2]    = 3 * r ;

        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        gluLookAt(      view[0] , view[1], view[2] ,
                                x_center, y_center, 3.0,
                                0.0, 0.0, 1.0    );

        glPushMatrix();
                draw_board_and_rods(t_board);
                draw_discs();
        glPopMatrix();
        glFlush();
        glutSwapBuffers();
}

void reshape_handler(int w, int h)
{
        window_width = w;
        window_height = h;

        glViewport(0, 0, (GLsizei)w, (GLsizei)h);

        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(45.0, (GLfloat)w / (GLfloat)h, 0.1, 20.0);

        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
}

void move_stack(int n, int f, int t)
{
        if (n == 1) {
```

```
                        solution_pair s;

                        s.f = f;

                        s.t = t;

                        sol.push_back(s);                    //pushing the (from, to) pair of solution to a list [so that it can be animated
later]

                        moves++;

                        cout << "From rod " << f << " to Rod " << t << endl;

                        return;

            }

            move_stack(n - 1, f, 3 - t - f);

            move_stack(1, f, t);

            move_stack(n - 1, 3 - t - f, t);

}


//Solve from 1st rod to 2nd

void solve()

{

            move_stack(NUM_DISCS, 0, 2);

}


void keyboard_handler(unsigned char key, int x, int y)

{


        //Console Outputs

            switch (key)

            {

            case 27:

            case 'q' :

            case 'Q' :

                        exit(0);

                        break;


            case 'h':

            case 'H':

                        cout << "ESC: Quit" << endl;
```

```
                    cout << "S: Solve from Initial State" << endl ;

                    cout << "H: Help" << endl;

                    break;


            case 's':

            case 'S':

                    if (t_board.rods[0].occupancy_val[NUM_DISCS - 1] < 0)

                            break;


                    solve();

                    to_solve = true;

                    break;


            case '+' : if(SPEED < 50)SPEED += 0.2 ; break ;

            case '-' : if(SPEED > 1)SPEED -= 0.2 ; break ;


            default:

                    break;

            };

}



void reshape(int w,int h)

{


        glViewport(0,0,w,h);

        glMatrixMode(GL_PROJECTION);

        glLoadIdentity();

        gluOrtho2D(0,w,h,0);

        glMatrixMode(GL_MODELVIEW);

        glLoadIdentity();


}

void drawStrokeText(const char* string,int x,int y,int z)
```

```
{

        const char *c;
        glPushMatrix();
        glTranslatef(x, y+8,z);
        glScalef(0.49f,-0.508f,z);

        for (c=string; *c !='\0'; c++)
        {
            glutStrokeCharacter(GLUT_STROKE_ROMAN , *c);
        }
        glPopMatrix();
}

void render(void)
{
      glClear(GL_COLOR_BUFFER_BIT);
      glLoadIdentity();

      glColor3f(1,1,0);
      drawStrokeText("Towers of Hanoi",100,50,0);

      drawStrokeText("Press:",200,130,0);
      drawStrokeText("H -> Help (in console window)", 100,200, 0);
      drawStrokeText("ESC or Q -> Quit", 100,300, 0);
      drawStrokeText("C -> Continue",100,400,0);

      glutSwapBuffers();
}

void keyboard_handler_for_intro(unsigned char key, int x, int y)
{

    //Console Outputs
```

```
	switch (key)
	{
	case 27:
	case 'q' :
	case 'Q' :
		exit(0);
		break;

	case 'h':
	case 'H':
		cout << "ESC: Quit" << endl;
		cout << "S: Solve from Initial State" << endl ;
		cout << "H: Help" << endl;
		break;


	case 'c':
	case 'C' :
		glutDisplayFunc(display_handler);
		glutReshapeFunc(reshape_handler);
		glutKeyboardFunc(keyboard_handler);
		glutIdleFunc(display_handler);
		main2();


	default:
		break;
	};
}



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

```
            // initialize glut
        glutInit(&argc, argv);

        // specify the display mode to be RGB and single buffering
        // we use single buffering since this will be non animated
        glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);

        // define the size
        glutInitWindowSize(1350,690);
            //glutFullScreen();
        // the position where the window will appear
        glutInitWindowPosition(0,0);
        glutCreateWindow("Towers Of Hanoi");
            glutKeyboardFunc(keyboard_handler_for_intro);
        glutDisplayFunc(render);

        glutReshapeFunc(reshape);
            //glutCreateMenu(select);




        // enter the main loop
        glutMainLoop();
      return 0;
}


void move_disc(int from_rod, int to_rod)
{

      int d = to_rod - from_rod;

      if (d > 0)
            active_disc.direction = 1;
```

```
    else if (d < 0)
        active_disc.direction = -1;


    if ((from_rod == to_rod  ) || (from_rod < 0) || (to_rod < 0) || (from_rod > 2) || (to_rod > 2) )
        return;


    int i;
    for (i = NUM_DISCS - 1; i >= 0 && t_board.rods[from_rod].occupancy_val[i] < 0; i--);
    if ( (i < 0) || (i == 0 && t_board.rods[from_rod].occupancy_val[i] < 0) )
        return;
    // Either the index < 0 or index at 0 and occupancy < 0 => it's an empty rod


    active_disc.start_pos = t_board.rods[from_rod].positions[i];


    active_disc.disc_index = t_board.rods[from_rod].occupancy_val[i];
    active_disc.is_in_motion = true;
    active_disc.u = 0.0;



    int j;
    for (j = 0; j < NUM_DISCS - 1 && t_board.rods[to_rod].occupancy_val[j] >= 0; j++);
    active_disc.dest_pos = t_board.rods[to_rod].positions[j];


    t_board.rods[from_rod].occupancy_val[i] = -1;
    t_board.rods[to_rod].occupancy_val[j] = active_disc.disc_index;
}



Vector3 get_inerpolated_coordinate(Vector3 sp, Vector3 tp, double u)
{
    //4 Control points
    Vector3 p;
    double x_center = (t_board.x_max - t_board.x_min) / 2.0;
    double y_center = (t_board.y_max - t_board.y_min) / 2.0;
double u3 = u * u * u;

double u2 = u * u;


Vector3 cps[4]; //P1, P2, dP1, dP2




//Hermite Interpolation

//Check Reference for equation of spline

{

        //P1

        cps[0].x = sp.x;

        cps[0].y = y_center;

        cps[0].z = ROD_HEIGHT + 0.2 * (t_board.rod_base_rad);


        //P2

        cps[1].x = tp.x;

        cps[1].y = y_center;

        cps[1].z = ROD_HEIGHT + 0.2 * (t_board.rod_base_rad);


        //dP1

        cps[2].x = (sp.x + tp.x) / 2.0 - sp.x ;

        cps[2].y = y_center;

        cps[2].z = 2 * cps[1].z; //change 2 * ..


        //dP2

        cps[3].x = tp.x - (tp.x + sp.x)/2.0;

        cps[3].y = y_center;

        cps[3].z = -cps[2].z; //- cps[2].z;




        double h0 = 2 * u3 - 3 * u2 + 1;

        double h1 = -2 *u3 + 3 * u2;

        double h2 = u3 - 2 * u2 + u;
```

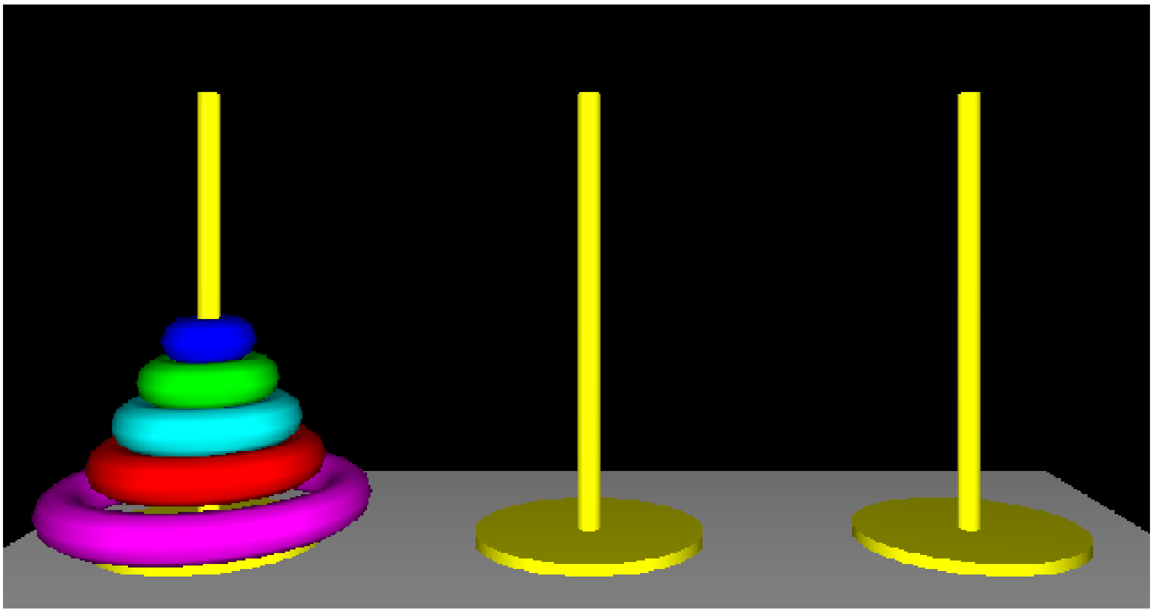
```
                double h3 = u3 - u2;


                p.x = h0 * cps[0].x + h1 * cps[1].x + h2 * cps[2].x +    h3 * cps[3].x;
                p.y = h0 * cps[0].y + h1 * cps[1].y + h2 * cps[2].y +    h3 * cps[3].y;
                p.z = h0 * cps[0].z + h1 * cps[1].z + h2 * cps[2].z +    h3 * cps[3].z;


        }


        return p;
}

//Normalize function for a vector
void normalize(Vector3& v)
{
        double length = sqrt(v.x * v.x + v.y * v.y + v.z * v.z);
        if (length == 0.0) return;
        v.x /= length;
        v.y /= length;
        v.z /= length;
}



Vector3 operator-(Vector3 const& v1, Vector3 const& v2)
{
        return Vector3(v1.x - v2.x, v1.y - v2.y, v1.z - v2.z);
}





void anim_handler()
{
        FPS = int(30 * SPEED);
        int curr_time = glutGet(GLUT_ELAPSED_TIME);
        int elapsed = curr_time - prev_time; // in ms
```

```
if (elapsed < 1000 / FPS) return;


prev_time = curr_time;


if (to_solve && active_disc.is_in_motion == false) {
            solution_pair s = sol.front();


   cout << s.f << ", " << s.t << endl;


            sol.pop_front();
            int i;
            for (i = NUM_DISCS; i >= 0 && t_board.rods[s.f].occupancy_val[i] < 0; i--);
            int ind = t_board.rods[s.f].occupancy_val[i];


            if (ind >= 0)
            active_disc.disc_index = ind;


            move_disc(s.f, s.t);
            if (sol.size() == 0)
                        to_solve = false;
}


if (active_disc.is_in_motion)
{
            int ind = active_disc.disc_index;
            ActiveDisc& ad = active_disc;


            if (ad.u == 0.0 && (discs[ind].position.z < ROD_HEIGHT + 0.2 * (t_board.rod_base_rad)) )
            {
                        discs[ind].position.z += 0.05;
                        glutPostRedisplay();
                        return;
            }
```

```
static bool done = false;

if (ad.u == 1.0 && discs[ind].position.z > ad.dest_pos.z )

{

        done = true;

        discs[ind].normal = Vector3(0, 0, 1);

        discs[ind].position.z -= 0.05;

        glutPostRedisplay();

        return;

}


ad.u += ad.step_u;

if (ad.u > 1.0) {

        ad.u = 1.0;

}


if (!done) {

        Vector3 prev_p = discs[ind].position;

        Vector3 p = get_inerpolated_coordinate(ad.start_pos, ad.dest_pos, ad.u);

        discs[ind].position = p;

        discs[ind].normal.x = (p - prev_p).x;

        discs[ind].normal.y = (p - prev_p).y;

        discs[ind].normal.z = (p - prev_p).z;

        normalize(discs[ind].normal);

}


if (ad.u >= 1.0 && discs[ind].position.z <= ad.dest_pos.z) {

        discs[ind].position.z = ad.dest_pos.z;

        ad.is_in_motion = false;

        done = false;

        ad.u = 0.0;

        discs[ad.disc_index].normal = Vector3(0, 0, 1);

        ad.disc_index = -1;


}
```

```
                glutPostRedisplay();
        }
}
```

output:-

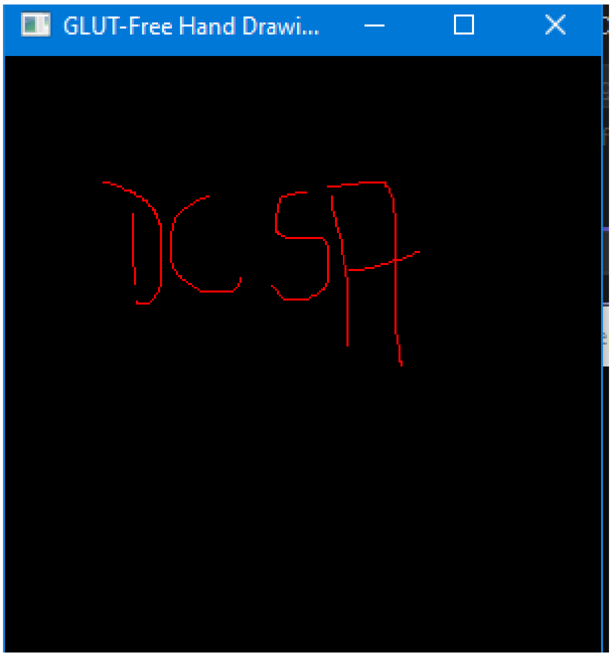


PROGRAM – 23-Free hand drawing Algorithm: Append mouse positions to a std::vector & use GL_LINE_STRIP

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


std::vector< int > points;
void draw()
{
    glBegin(GL_LINE_STRIP);
    glColor3ub(255, 0, 0);
    for (size_t i = 0; i < points.size(); i += 2)
    {
        glVertex2i(points[i + 0], points[i + 1]);
    }
    glEnd();


    glutSwapBuffers();
}
void mouse( int button, int state, int x, int y )
{
    if( state == GLUT_DOWN )
        points.clear();
    points.push_back( x );
    points.push_back( y );
    draw();
}

void motion( int x, int y )
{
    points.push_back( x );
    points.push_back( y );
    draw();
}

void display()
{
    glClearColor( 0, 0, 0, 1 );
    glClear(GL_COLOR_BUFFER_BIT);

    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    double w = glutGet( GLUT_WINDOW_WIDTH );
    double h = glutGet( GLUT_WINDOW_HEIGHT );
    glOrtho( 0, w, h, 0, -1, 1 );

    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    draw();
}

int main( int argc, char** argv )
{
    glutInit( &argc, argv );
    glutInitDisplayMode( GLUT_RGBA | GLUT_DOUBLE );
```

```
        glutCreateWindow( "GLUT-Free Hand Drawing" );

        glutMouseFunc( mouse );

        glutMotionFunc( motion );

        glutDisplayFunc( display );

        glutMainLoop();

        return 0;

}
```

output:-



PROGRAM – 24-Rotating cube with color interpolation

```
#include <stdlib.h>

#include <GL/glut.h>


            GLfloat vertices[][3] = {{-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}, {1.0,1.0,1.0}, {-1.0,1.0,1.0}};


            GLfloat normals[][3] = {{-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}, {1.0,1.0,1.0}, {-1.0,1.0,1.0}};


            GLfloat colors[][3] = {{0.0,0.0,0.0},{1.0,0.0,0.0},

            {1.0,1.0,0.0}, {0.0,1.0,0.0}, {0.0,0.0,1.0},

            {1.0,0.0,1.0}, {1.0,1.0,1.0}, {0.0,1.0,1.0}};
```

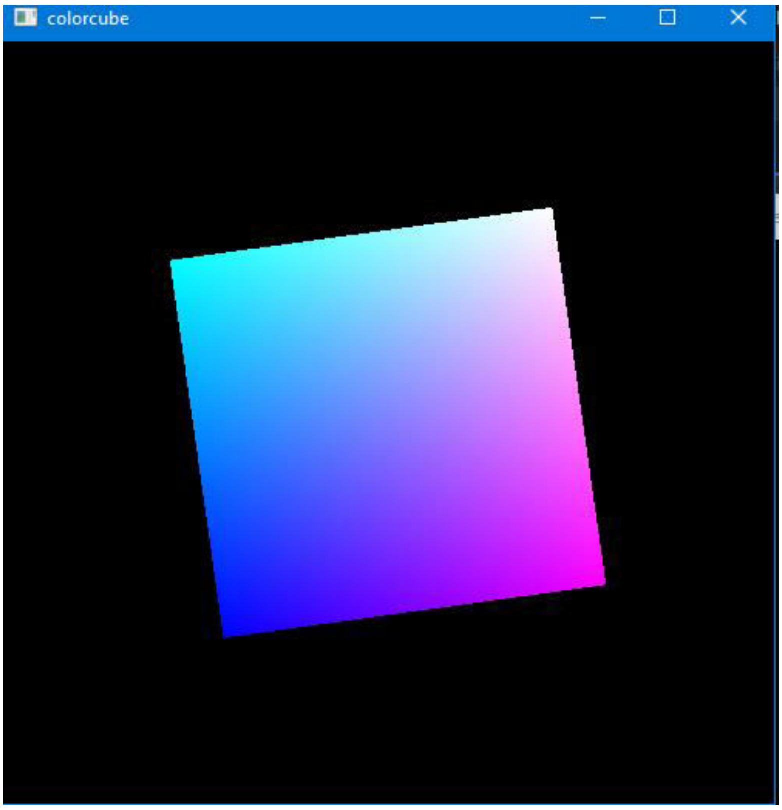
```
void polygon(int a, int b, int c , int d)
{

/* draw a polygon via list of vertices */

        glBegin(GL_POLYGON);
                glColor3fv(colors[a]);
                glNormal3fv(normals[a]);
                glVertex3fv(vertices[a]);
                glColor3fv(colors[b]);
                glNormal3fv(normals[b]);
                glVertex3fv(vertices[b]);
                glColor3fv(colors[c]);
                glNormal3fv(normals[c]);
                glVertex3fv(vertices[c]);
                glColor3fv(colors[d]);
                glNormal3fv(normals[d]);
                glVertex3fv(vertices[d]);
        glEnd();

                                                                                        }

void colorcube(void)
{

/* map vertices to faces */

        polygon(0,3,2,1);
        polygon(2,3,7,6);
        polygon(0,4,7,3);
        polygon(1,2,6,5);
        polygon(4,5,6,7);
        polygon(0,1,5,4);
}
```

```
static GLfloat theta[] = {0.0,0.0,0.0};
static GLint axis = 2;

void display(void)
{
/* display callback, clear frame buffer and z buffer,
    rotate cube and draw, swap buffers */

  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);
        glRotatef(theta[2], 0.0, 0.0, 1.0);

  colorcube();

  glFlush();
        glutSwapBuffers();
}

void spinCube()
{

/* Idle callback, spin cube 2 degrees about selected axis */

        theta[axis] += 2.0;
        if( theta[axis] > 360.0 ) theta[axis] -= 360.0;
        /* display(); */
        //glutPostRedisplay();
}

void mouse(int btn, int state, int x, int y)
{
```

```
/* mouse callback, selects an axis about which to rotate */

      if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN) axis = 0;
      if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) axis = 1;
      if(btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN) axis = 2;
}

void myReshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    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, -10.0, 10.0);
    glMatrixMode(GL_MODELVIEW);
}
void Timer(int iUnused)
{
      glutPostRedisplay();
      glutTimerFunc(160, Timer, 0);
}
int main(int argc, char **argv)
{
    glutInit(&argc, argv);

/* need both double buffering and z buffer */

    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
```

```
        glutCreateWindow("colorcube");

        glutReshapeFunc(myReshape);

        glutDisplayFunc(display);

                glutIdleFunc(spinCube);

                glutMouseFunc(mouse);

                glEnable(GL_DEPTH_TEST); /* Enable hidden--surface--removal */

                Timer(0);

        glutMainLoop();

            return 0;

}
```

output:-



PROGRAM – 25-Paint Program in 2D using OpenGL/GLUT [select color and shape before start of painting]

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

#include <iostream>

#include <cmath>

using namespace std;


// The following statement is to hide the console window

#pragma comment (linker,"/subsystem:\"windows\" /entry:\"mainCRTStartup\"")
```

```
// Maximum primitives
const int MAX = 500;

// Window width and height
int window_w;
int window_h;

bool dragging = false;

// Color structure
struct Color
{
        float r = 0.0;
        float g = 0.0;
        float b = 0.0;
};

// Shape structure
struct Shape
{
        string type;
        float startX, startY;
        float endX, endY;
        float pointSize;
        float lineWidth;
        bool isFilled = false;
        bool isActivated = false;
        Color color;
};

Color color;
Shape shapeList[MAX];
int shapeCount = 0;
```

```
// Draw mode enum
enum DRAW_MODE
{
        NONE, POINT_MODE, LINE_MODE,
        W_TRIANGLE_MODE, F_TRIANGLE_MODE, W_RECTANGLE_MODE, F_RECTANGLE_MODE, W_CIRCLE_MODE, F_CIRCLE_MODE
};

// Right-click menu enum
enum RIGHT_CLICK_MENU
{
        CLEAR, EXIT
};

// Color code enum
enum COLOR_CODE
{
        RED, GREEN, BLUE, CYAN, MAGENTA, YELLOW, WHITE, BLACK
};

DRAW_MODE mode = NONE;

float point_size = 1.0;
float line_width = 1.0;

const float PI = 3.142;

// Handle start drawing process
void handleStartDraw(float x, float y)
{
        if (mode == NONE)
        {
                return;
        }
        else
```

```
{
        // Start drawing
        dragging = true;

        // Active current shape
        shapeList[shapeCount].isActivated = true;

        if (mode == POINT_MODE) // Point mode
        {
                shapeList[shapeCount].type = "Point";
        }
        if (mode == LINE_MODE) // Line mode
        {
                shapeList[shapeCount].type = "Line";
        }
        if (mode == W_TRIANGLE_MODE || mode == F_TRIANGLE_MODE) // Triangle mode
        {
                if (mode == F_TRIANGLE_MODE) // Filled triangle
                {
                        shapeList[shapeCount].isFilled = true;
                }
                else // Wireframe triangle
                {
                        shapeList[shapeCount].isFilled = false;
                }

                shapeList[shapeCount].type = "Triangle";
        }
        if (mode == W_RECTANGLE_MODE || mode == F_RECTANGLE_MODE) // Rectangle mode
        {
                if (mode == F_RECTANGLE_MODE) // Filled rectangle
                {
                        shapeList[shapeCount].isFilled = true;
                }
```

```
    else // Wireframe triangle
    {
        shapeList[shapeCount].isFilled = false;
    }

    shapeList[shapeCount].type = "Rectangle";
}
if (mode == W_CIRCLE_MODE || mode == F_CIRCLE_MODE) // Circle mode
{
    if (mode == F_CIRCLE_MODE) // Filled circle
    {
        shapeList[shapeCount].isFilled = true;
    }
    else // Wireframe circle
    {
        shapeList[shapeCount].isFilled = false;
    }

    shapeList[shapeCount].type = "Circle";
}

// Set shape coordinates
shapeList[shapeCount].startX = x;
shapeList[shapeCount].startY = y;
shapeList[shapeCount].endX = x;
shapeList[shapeCount].endY = y;

// Set shape point size and line width
shapeList[shapeCount].pointSize = point_size;
shapeList[shapeCount].lineWidth = line_width;

// Set shape color
shapeList[shapeCount].color.r = color.r;
shapeList[shapeCount].color.g = color.g;
```

```
		shapeList[shapeCount].color.b = color.b;

		shapeCount++;
	}
}

// Handle continue drawing process
void handleContinueDraw(float x, float y)
{
	if (!dragging)
	{
		return;
	}

	// Get current position of end X and Y
	int current = shapeCount - 1;

	shapeList[current].endX = x;
	shapeList[current].endY = y;

	glutPostRedisplay();
}

// Handle finish drawing process
void handleFinishDraw(float x, float y)
{
	if (!dragging)
	{
		return;
	}

	// Finish drawing
	dragging = false;
```

```
        // Get current position of all coordinates
        int current = shapeCount - 1;

        if (shapeList[current].startX == shapeList[current].endX &&
                shapeList[current].startX == shapeList[current].endY)
        {
                shapeCount--;
        }

        glutPostRedisplay();
}

// Clear all primivites on drawing area
void clearPrimitives()
{
        // Reset shape count to zero
        shapeCount = 0;

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glFlush();
        glutPostRedisplay();
}

// Color menu
void colorMenu(int code)
{
        if (code == RED) // Red
        {
                color.r = 1.0;
                color.g = 0.0;
                color.b = 0.0;
        }
        else if (code == GREEN) // Green
        {
```

```
        color.r = 0.0;
        color.g = 1.0;
        color.b = 0.0;
}
else if (code == BLUE) // Blue
{
        color.r = 0.0;
        color.g = 0.0;
        color.b = 1.0;
}
else if (code == CYAN) // Cyan
{
        color.r = 0.0;
        color.g = 1.0;
        color.b = 1.0;
}
else if (code == MAGENTA) // Magenta
{
        color.r = 1.0;
        color.g = 0.0;
        color.b = 1.0;
}
else if (code == YELLOW) // Yellow
{
        color.r = 1.0;
        color.g = 1.0;
        color.b = 0.0;
}
else if (code == WHITE) // White
{
        color.r = 1.0;
        color.g = 1.0;
        color.b = 1.0;
}
```

```
	else if (code == BLACK) // Black
	{
		color.r = 0.0;
		color.g = 0.0;
		color.b = 0.0;
	}

	int current = shapeCount - 1;

	// Change color of current shape
	if (shapeList[current].isActivated)
	{
		shapeList[current].color.r = color.r;
		shapeList[current].color.g = color.g;
		shapeList[current].color.b = color.b;
	}

	glutPostRedisplay();
}

// Point size menu
void pointSizeMenu(int size)
{
	if (size == 1.0) // 1.0
	{
		point_size = 1.0;
	}
	else if (size == 2.0) // 2.0
	{
		point_size = 2.0;
	}
	else if (size == 3.0) // 3.0
	{
		point_size = 3.0;
```

```
        }
        else if (size == 4.0) // 4.0
        {
                point_size = 4.0;
        }
        else if (size == 5.0) // 5.0
        {
                point_size = 5.0;
        }
        else if (size == 6.0) // 6.0
        {
                point_size = 6.0;
        }
}

// Line width menu
void lineWidthMenu(int width)
{
        if (width == 1.0) // 1.0
        {
                line_width = 1.0;
        }
        else if (width == 2.0) // 2.0
        {
                line_width = 2.0;
        }
        else if (width == 3.0) // 3.0
        {
                line_width = 3.0;
        }
        else if (width == 4.0) // 4.0
        {
                line_width = 4.0;
        }
```

```
    else if (width == 5.0) // 5.0
    {
        line_width = 5.0;
    }
    else if (width == 6.0) // 6.0
    {
        line_width = 6.0;
    }

    int current = shapeCount - 1;

    // Change line width of current shape
    if (shapeList[current].isActivated)
    {
        shapeList[current].lineWidth = line_width;
    }

    glutPostRedisplay();
}

// Right-click menu
void rightClickMenu(int option)
{
    switch (option)
    {
    case CLEAR:     // Clear
        clearPrimitives();
        break;

    case EXIT: // Exit
        exit(0);
    }
}
```

```
// Create right-click mouse menu
void createMouseMenu()
{
	int c_menu, ps_menu, lw_menu;

	// Create color sub-menu
	c_menu = glutCreateMenu(colorMenu);
	glutAddMenuEntry("Red", RED);
	glutAddMenuEntry("Green", GREEN);
	glutAddMenuEntry("Blue", BLUE);
	glutAddMenuEntry("Cyan", CYAN);
	glutAddMenuEntry("Magenta", MAGENTA);
	glutAddMenuEntry("Yellow", YELLOW);
	glutAddMenuEntry("White", WHITE);
	glutAddMenuEntry("Black", BLACK);

	// Create point size sub-menu
	ps_menu = glutCreateMenu(pointSizeMenu);
	glutAddMenuEntry("1.0", 1.0);
	glutAddMenuEntry("2.0", 2.0);
	glutAddMenuEntry("3.0", 3.0);
	glutAddMenuEntry("4.0", 4.0);
	glutAddMenuEntry("5.0", 5.0);
	glutAddMenuEntry("6.0", 6.0);

	// Create point size sub-menu
	lw_menu = glutCreateMenu(lineWidthMenu);
	glutAddMenuEntry("1.0", 1.0);
	glutAddMenuEntry("2.0", 2.0);
	glutAddMenuEntry("3.0", 3.0);
	glutAddMenuEntry("4.0", 4.0);
	glutAddMenuEntry("5.0", 5.0);
	glutAddMenuEntry("6.0", 6.0);
	// Create main menu
	glutCreateMenu(rightClickMenu);
	glutAddSubMenu("Colors", c_menu);
	glutAddSubMenu("Point size", ps_menu);
	glutAddSubMenu("Line width", lw_menu);
	glutAddMenuEntry("Clear", CLEAR);
	glutAddMenuEntry("Exit", EXIT);
	glutAttachMenu(GLUT_RIGHT_BUTTON);
}

// Select drawing mode from mouse click
void selectMode(int x, int y, int modifiers)
{
	if (y > window_h - 50) // Point mode
	{
		mode = POINT_MODE;
	}
	else if (y > window_h - 100) // Line mode
	{
		mode = LINE_MODE;
	}
	else if (y > window_h - 150) // Wireframe triangle mode
	{
		mode = W_TRIANGLE_MODE;
	}
	else if (y > window_h - 200) // Filled triangle mode
	{
		mode = F_TRIANGLE_MODE;
	}
	else if (y > window_h - 250) // Wireframe rectangle mode
	{
		mode = W_RECTANGLE_MODE;
	}
	else if (y > window_h - 300) // Filled rectangle mode
```

```
    {
        mode = F_RECTANGLE_MODE;
    }
    else if (y > window_h - 350) // Wireframe circle mode
    {
        mode = W_CIRCLE_MODE;
    }
    else if (y > window_h - 400) // Filled circle mode
    {
        mode = F_CIRCLE_MODE;
    }
}

// Draw outline around selected mode
void drawSelected()
{
    if (mode == NONE)
    {
        return;
    }
    if (mode == POINT_MODE) // Point mode
    {
        glColor3f(1.0, 1.0, 1.0);
        glBegin(GL_LINE_LOOP);
        glVertex2i(2, window_h - 2);
        glVertex2i(49, window_h - 2);
        glVertex2i(49, window_h - 49);
        glVertex2i(2, window_h - 49);
        glEnd();
    }
    else if (mode == LINE_MODE) // Line mode
    {
        glColor3f(1.0, 1.0, 1.0);
        glBegin(GL_LINE_LOOP);
```

```
		glVertex2i(2, window_h - 51);
		glVertex2i(49, window_h - 51);
		glVertex2i(49, window_h - 99);
		glVertex2i(2, window_h - 99);
		glEnd();
	}
	else if (mode == W_TRIANGLE_MODE) // Wireframe triangle mode
	{
		glColor3f(1.0, 1.0, 1.0);
		glBegin(GL_LINE_LOOP);
		glVertex2i(2, window_h - 101);
		glVertex2i(49, window_h - 101);
		glVertex2i(49, window_h - 149);
		glVertex2i(2, window_h - 149);
		glEnd();
	}
	else if (mode == F_TRIANGLE_MODE) // Filled triangle mode
	{
		glColor3f(1.0, 1.0, 1.0);
		glBegin(GL_LINE_LOOP);
		glVertex2i(2, window_h - 151);
		glVertex2i(49, window_h - 151);
		glVertex2i(49, window_h - 199);
		glVertex2i(2, window_h - 199);
		glEnd();
	}
	else if (mode == W_RECTANGLE_MODE) // Wireframe rectangle mode
	{
		glColor3f(1.0, 1.0, 1.0);
		glBegin(GL_LINE_LOOP);
		glVertex2i(2, window_h - 201);
		glVertex2i(49, window_h - 201);
		glVertex2i(49, window_h - 249);
		glVertex2i(2, window_h - 249);
```

```
			glEnd();
		}
		else if (mode == F_RECTANGLE_MODE) // Filled rectangle mode
		{
			glColor3f(1.0, 1.0, 1.0);
			glBegin(GL_LINE_LOOP);
			glVertex2i(2, window_h - 251);
			glVertex2i(49, window_h - 251);
			glVertex2i(49, window_h - 299);
			glVertex2i(2, window_h - 299);
			glEnd();
		}
		else if (mode == W_CIRCLE_MODE) // Wireframe circle mode
		{
			glColor3f(1.0, 1.0, 1.0);
			glBegin(GL_LINE_LOOP);
			glVertex2i(2, window_h - 301);
			glVertex2i(49, window_h - 301);
			glVertex2i(49, window_h - 349);
			glVertex2i(2, window_h - 349);
			glEnd();
		}
		else if (mode == F_CIRCLE_MODE) // Filled circle mode
		{
			glColor3f(1.0, 1.0, 1.0);
			glBegin(GL_LINE_LOOP);
			glVertex2i(2, window_h - 351);
			glVertex2i(49, window_h - 351);
			glVertex2i(49, window_h - 399);
			glVertex2i(2, window_h - 399);
			glEnd();
		}
}
```

```
// Draw menu icon
void drawMenuIcon()
{
        // Draw point icon
        glColor3f(0.0, 0.0, 0.0);
        glPointSize(4);
        glBegin(GL_POINTS);
        glVertex2i(25, window_h - 25);
        glEnd();

        // Draw line icon
        glColor3f(0.0, 0.0, 0.0);
        glBegin(GL_LINES);
        glVertex2i(10, window_h - 90);
        glVertex2i(40, window_h - 60);
        glEnd();

        // Draw wireframe triangle icon
        glColor3f(0.0, 0.0, 0.0);
        glBegin(GL_LINE_LOOP);
        glVertex2i(25, window_h - 110);
        glVertex2i(10, window_h - 135);
        glVertex2i(40, window_h - 135);
        glEnd();

        // Draw filled triangle icon
        glColor3f(0.0, 0.0, 0.0);
        glBegin(GL_TRIANGLES);
        glVertex2i(25, window_h - 160);
        glVertex2i(10, window_h - 185);
        glVertex2i(40, window_h - 185);
        glEnd();

        // Draw wireframe rectangle icon
```

```
glColor3f(0.0, 0.0, 0.0);

glBegin(GL_LINE_LOOP);

glVertex2i(10, window_h - 210);

glVertex2i(40, window_h - 210);

glVertex2i(40, window_h - 240);

glVertex2i(10, window_h - 240);

glEnd();


// Draw filled rectangle icon

glColor3f(0.0, 0.0, 0.0);

glBegin(GL_QUADS);

glVertex2i(10, window_h - 260);

glVertex2i(40, window_h - 260);

glVertex2i(40, window_h - 290);

glVertex2i(10, window_h - 290);

glEnd();


int segment = 300;

float twoPI = PI * 2.0;

float radius = 15.0;


// Draw wireframe circle icon

glColor3f(0.0, 0.0, 0.0);

glBegin(GL_LINE_LOOP);


for (int i = 0 ; i < segment; i++)

{

        glVertex2f(25 + (radius * cos(i *    twoPI / segment)),

                (window_h - 325) + (radius * sin(i * twoPI / segment)));

}


glEnd();


// Draw filled circle icon
```

```
    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_TRIANGLE_FAN);

    for (int i = 0; i < segment; i++)
    {
        glVertex2f(25 + (radius * cos(i *    twoPI / segment)),
            (window_h - 375) + (radius * sin(i * twoPI / segment)));
    }

    glEnd();

    // Draw color status box
    glColor3f(color.r, color.g, color.b);
    glBegin(GL_QUADS);
    glVertex2i(2, window_h - 401);
    glVertex2i(49, window_h - 401);
    glVertex2i(49, window_h - 449);
    glVertex2i(2, window_h - 449);
    glEnd();
}

// Draw paint menu
void drawPaintMenu()
{
    // Draw menu bar
    glColor3f(0.4, 0.4, 0.4);
    glRectf(0, 0, 50, window_h);

    glColor3f(0.0, 0.0, 0.0);
    glLineWidth(2.0);
    glBegin(GL_LINES);

    // Draw bottom line
    glVertex2i(1, 1);
```

```
        glVertex2i(50, 1);


        // Draw left line

        glVertex2i(1, 1);

        glVertex2i(1, window_h);


        // Draw right line

        glVertex2i(50, 1);

        glVertex2i(50, window_h);


        // Draw top line

        glVertex2i(1, window_h - 1);

        glVertex2i(50, window_h - 1);


        int distance = 50;


        // Draw 9 lines for each icon

        for (int i = 1; i <= 9; i++)

        {

                glVertex2i(1, window_h - distance);

                glVertex2i(50, window_h - distance);


                distance = distance + 50;

        }


        glEnd();

}


// Draw menu interface

void drawInterface()

{

        // Draw paint menu

        drawPaintMenu();
```

```
    // Draw outline around selected mode
    drawSelected();

    // Draw menu icon
    drawMenuIcon();

    glutPostRedisplay();
}

// Draw all primitives on drawing area
void drawPrimitives()
{
    for (int i = 0; i < shapeCount; i++)
    {
        // Set primitives color based on selected color
        glColor3f(shapeList[i].color.r, shapeList[i].color.g, shapeList[i].color.b);

        if (shapeList[i].type == "Point") // Point
        {
            // Draw point
            glPointSize(shapeList[i].pointSize);
            glBegin(GL_POINTS);
            glVertex2f(shapeList[i].startX, shapeList[i].startY);
            glEnd();
        }
        if (shapeList[i].type == "Line") // Line
        {
            // Draw line
            glLineWidth(shapeList[i].lineWidth);
            glBegin(GL_LINES);
            glVertex2f(shapeList[i].startX, shapeList[i].startY);
            glVertex2f(shapeList[i].endX, shapeList[i].endY);
            glEnd();
        }
```

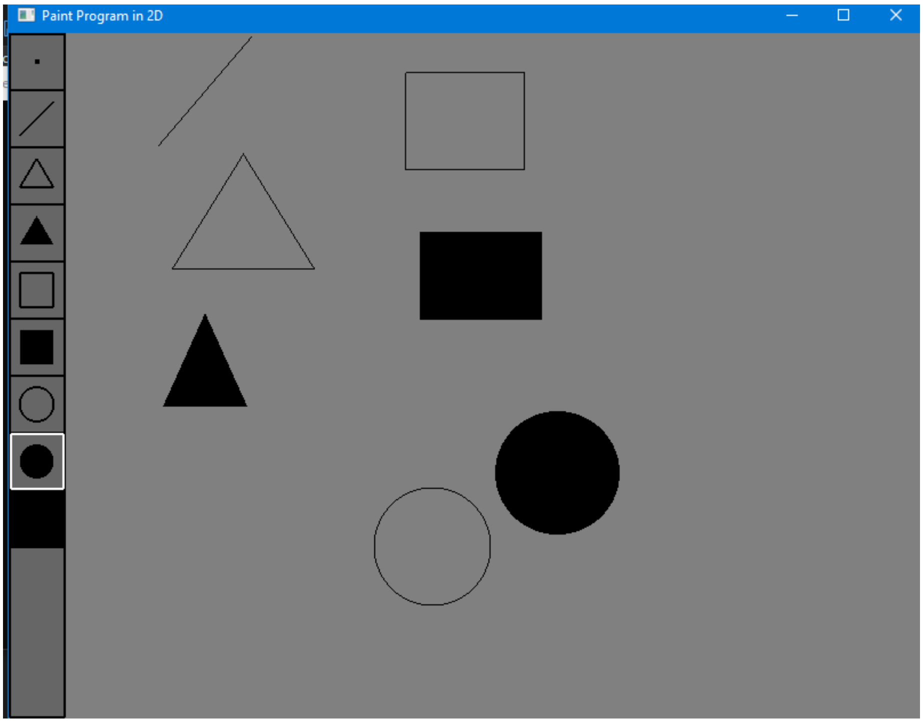
```
if (shapeList[i].type == "Triangle") // Triangle
{
        if (shapeList[i].isFilled) // Filled
        {
                glBegin(GL_TRIANGLES);
        }
        else // Wireframe
        {
                glLineWidth(shapeList[i].lineWidth);
                glBegin(GL_LINE_LOOP);
        }

        // Draw triangle
        glVertex2f(shapeList[i].startX, shapeList[i].startY);
        glVertex2f(((shapeList[i].endX - shapeList[i].startX) / 2) +
                shapeList[i].startX, shapeList[i].endY);
        glVertex2f(shapeList[i].endX, shapeList[i].startY);
        glEnd();
}
if (shapeList[i].type == "Rectangle") // Rectangle
{
        if (shapeList[i].isFilled) // Filled
        {
                glBegin(GL_QUADS);
        }
        else // Wireframe
        {
                glLineWidth(shapeList[i].lineWidth);
                glBegin(GL_LINE_LOOP);
        }

        // Draw rectangle
        glVertex2f(shapeList[i].startX, shapeList[i].startY);
```

```
    glVertex2f(shapeList[i].endX, shapeList[i].startY);
    glVertex2f(shapeList[i].endX, shapeList[i].endY);
    glVertex2f(shapeList[i].startX, shapeList[i].endY);
    glEnd();
}
if (shapeList[i].type == "Circle") // Circle
{
    if (shapeList[i].isFilled) // Filled
    {
        glBegin(GL_TRIANGLE_FAN);
    }
    else // Wireframe
    {
        glLineWidth(shapeList[i].lineWidth);
        glBegin(GL_LINE_LOOP);
    }

    // Calculate half width and height
    float halfWidth = (shapeList[i].endX - shapeList[i].startX) / 2;
    float halfHeight = (shapeList[i].endY - shapeList[i].startY) / 2;

    // Calculate center X and Y
    float centerX = shapeList[i].startX + halfWidth;
    float centerY = shapeList[i].startY + halfHeight;

    // Draw circle
    for (int j = 0; j < 360; j++)
    {
        float angle = j * PI / 180.0;
        float x = centerX + (cos(angle) * halfWidth);
        float y = centerY + (sin(angle) * halfHeight);
        glVertex2f(x, y);
    }
```

```
            glEnd();
        }

        // Create frame around primitives when drawing
        if (shapeList[i].isActivated)
        {
            glColor3f(1.0, 1.0, 1.0);
            glLineWidth(1.0);
            glEnable(GL_LINE_STIPPLE);
            glLineStipple(1.0, 0xF0F0);
            glBegin(GL_LINE_LOOP);

            // Line frame
            if (shapeList[i].type == "Line")
            {
                glVertex2f(shapeList[i].startX, shapeList[i].startY);
                glVertex2f(shapeList[i].endX, shapeList[i].endY);
            }
            // Triangle/Rectangle/Circle frame
            if (shapeList[i].type == "Triangle" || shapeList[i].type == "Rectangle" ||
                shapeList[i].type == "Circle")
            {
                glVertex2f(shapeList[i].startX, shapeList[i].startY);
                glVertex2f(shapeList[i].endX, shapeList[i].startY);
                glVertex2f(shapeList[i].endX, shapeList[i].endY);
                glVertex2f(shapeList[i].startX, shapeList[i].endY);
            }

            glEnd();
            glDisable(GL_LINE_STIPPLE);
        }
    }
}
```

```
// Initialize menu area and drawing area
void initArea(float x1, float x2, float y1, float y2)
{
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(x1, x2, y1, y2);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
}

// Initialize OpenGL/GLUT settings
void init()
{
	glClearColor(0.0, 0.0, 0.0, 0.0);
	glColor3f(1.0, 1.0, 1.0);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(0, window_w, 0, window_h);
}

// Display callback
void display()
{
	glClearColor(0.5, 0.5, 0.5, 0.5);
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	glViewport(50, 0, window_w - 50, window_h);

	// Initialize drawing area
	initArea(0, 1, 0, 1);

	drawPrimitives();

	glViewport(0, 0, 50, window_h);
}
        // Initialize paint menu area
        initArea(0, 51, 0, window_h);

        drawInterface();

        glutSwapBuffers();
}

// Reshape callback
void reshape(int w, int h)
{
        // Adjust clipping
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(0, w, 0, h, -1.0, 1.0);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();

        // Adjust viewport and clear
        glViewport(0, 0, w, h);
        glClearColor(0.8, 0.8, 0.8, 0.8);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        display();
        glFlush();

        // Set new window width and height value
        window_w = w;
        window_h = h;
}

// Mouse click callback
void mouse(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON)
    {
        if (x < 50 && state == GLUT_DOWN)
        {
            // Select drawing mode
            selectMode(x, window_h - y, glutGetModifiers());
        }

        float wx, wy;

        wx = (float)(x - 50) / (window_w - 50);
        wy = (float)(window_h - y) / window_h;

        if (state == GLUT_DOWN)
        {
            // Deactivate previus shape
            shapeList[shapeCount - 1].isActivated = false;

            // Handle start draw
            handleStartDraw(wx, wy);
        }
        else
        {
            // Handle finish draw
            handleFinishDraw(wx, wy);
        }
    }
}

// Mouse motion callback
void motion(int x, int y)
{
    if (dragging)
    {
        float wx, wy;

}       wx = (float)(x - 50) / (window_w - 50);
        wy = (float)(window_h - y) / window_h;

        // Handle continue draw
        handleContinueDraw(wx, wy);
    }
}


int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(800, 600); // Set window size
    glutInitWindowPosition(150, 50); // Set window position
    glutCreateWindow("Paint Program in 2D");
    init();

    // Create right-click mouse menu
    createMouseMenu();
    // Callback functions
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMouseFunc(mouse);
    glutMotionFunc(motion);

    glutMainLoop();

    return 0;
}
```

output:-



Paint Program in 2D