COMPUTER GRAPHICS

ALL SOURCE CODES WITH AN IMAGE OF THE O/P

1. CONCAVE POLYGON FILLING USING SCAN FILL ALGORITHM

```
#include <conio.h>
#include <iostream>
#include <graphics.h>
#include <stdlib.h>
using namespace std;
class point
{
 public:
 int x,y;
};
class poly
{
 private:
 point p[20];
 int inter[20],x,y;
 int v,xmin,ymin,xmax,ymax;
 public:
 int c;
 void read();
 void calcs();
 void display();
 void ints(float);
 void sort(int);
};
void poly::read()
{
 int i;
 cout<<"\n Scan Fill Algorithm ";
 cout<<"\n Enter Number Of Vertices Of Polygon: ";
 cin>>v;
 if(v>2)
 {
 for(i=0;i<v; i++) //ACCEPT THE VERTICES
 {
 cout<<"\nEnter co-ordinate no. "<<i+1<<" : ";
 cout<<"\n\tx"<<(i+1)<<"=";
 cin>>p[i].x;
 cout<<"\n\ty"<<(i+1)<<"=";
 cin>>p[i].y;
 }
 p[i].x=p[0].x;
 p[i].y=p[0].y;
 xmin=xmax=p[0].x;
 ymin=ymax=p[0].y;
 }
```

```
 else
 cout<<"\n Enter valid no. of vertices.";
}
void poly::calcs()
{
 for(int i=0;i<v;i++)
 {
 if(xmin>p[i].x)
 xmin=p[i].x;
 if(xmax<p[i].x)
 xmax=p[i].x;
 if(ymin>p[i].y)
 ymin=p[i].y;
 if(ymax<p[i].y)
 ymax=p[i].y;
 }
}
void poly::display()
{
 int ch1;
 char ch='y';
 float s,s2;
 do
 {
 cout<<"\n\nMENU:";
 cout<<"\n\n\t1 . Scan line Fill ";
 cout<<"\n\n\t2 . Exit ";
 cout<<"\n\nEnter your choice:";
 cin>>ch1;
 switch(ch1)
 {
 case 1:
 s=ymin+0.01;
 delay(100);
 cleardevice();
 while(s<=ymax)
 {
 ints(s);
 sort(s);
 s++;
 }
 break;
 case 2:
 exit(0);
 }
 cout<<"Do you want to continue?: ";
 cin>>ch;
 }while(ch=='y' || ch=='Y');
}
void poly::ints(float z)
```

```
{
 int x1,x2,y1,y2,temp;
 c=0;
 for(int i=0;i<v;i++)
 {
 x1=p[i].x;
 y1=p[i].y;
 x2=p[i+1].x;
 y2=p[i+1].y;
 if(y2<y1)
 {
 temp=x1;
 x1=x2;
 x2=temp;
 temp=y1;
 y1=y2;
 y2=temp;
 }
 if(z<=y2&&z>=y1)
 {
 if((y1-y2)==0)
 x=x1;
 else
 {
 x=((x2-x1)*(z-y1))/(y2-y1);
 x=x+x1;
 }
 if(x<=xmax && x>=xmin)
 inter[c++]=x;
 }
 }
}
void poly::sort(int z) // sorting
{
 int temp,j,i;
 for(i=0;i<v;i++)
 {
 line(p[i].x,p[i].y,p[i+1].x,p[i+1].y);
 }
 delay(100);
 for(i=0; i<c;i+=2)
 {
 delay(100);
 line(inter[i],z,inter[i+1],z);
 }
}
int main() //main
{
 int cl;
 initwindow(500,600);
```

```
 cleardevice();
 poly x;
 x.read();
 x.calcs();
 cleardevice();
 cout<<"\n\tEnter The Color You Want :(In Range 0 To 15 )->"; //selecting color
 cin>>cl;
 setcolor(cl);
 x.display();

 closegraph(); //closing graph
 getch();
 return 0;
}
```

2. POLYGON CLIPPING USING COHEN SUTHERLAND LINE CLIPPING ALGORITHM

```
#include <iostream>
#include <conio.h>
#include <graphics.h>
#include <math.h>
void Window()
{
	line(200, 200, 350, 200);
	line(350, 200, 350, 350);
	line(200, 200, 200, 350);
	line(200, 350, 350, 350);
}
void Code(char c[4], float x, float y)
{
	c[0] = (x < 200) ? '1' : '0';
	c[1] = (x > 350) ? '1' : '0';
	c[2] = (y < 200) ? '1' : '0';
	c[3] = (y > 350) ? '1' : '0';
}
void Clipping(char c[], char d[], float &x, float &y, float m)
{
	int flag = 1, i = 0;
	for (i = 0; i < 4; i++)
	{
		if (c[i] != '0' && d[i] != '0')
		{
```

```
                        flag = 0;
                        break;
                }
                if (flag)
                {
                        if (c[0] != '0')
                        {
                                y = m * (200 - x) + y;
                                x = 200;
                        }
                        else if (c[1] != '0')
                        {
                                y = m * (350 - x) + y;
                                x = 350;
                        }
                        else if (c[2] != '0')
                        {
                                x = ((200 - y) / m) + x;
                                y = 200;
                        }
                        else if (c[3] != '0')
                        {
                                x = ((350 - y) / m) + x;
                                y = 350;
                        }
                }
                if (flag == 0)
                        std::cout << "Line lying outside";
        }
}
int main()
{
        int gdriver = DETECT, gmode, errorcode;
        float x1, y1, x2, y2;
        float m;
        char c[4], d[4];
        //clrscr();
        initgraph(&gdriver, &gmode, " ");
        std::cout << "Enter coordinates";
        std::cin >> x1 >> y1 >> x2 >> y2;
        std::cout << "Before clipping";
        Window();
        line(x1, y1, x2, y2);
        getch();
        cleardevice();
        m = float((y2 - y1) / (x2 - x1));
        Code(c, x1, y1);
        Code(d, x2, y2);
        Clipping(c, d, x1, y1, m);
        Clipping(d, c, x2, y2, m);
```

```
        std::cout << "After Clipping";
        Window();
        line(x1, y1, x2, y2);
        getch();
        closegraph();

        return 0;
}
```

3.PATTERN DRAWING USING LINE AND CIRCLE

```
#include <iostream>
#include <math.h>
#include <graphics.h> //graphics.h is used to include graphical operater in a program.
using namespace std;

void DDALine(int x1, int y1, int x2, int y2, int color); // declare function

int main()
{
  int x1, y1, x2, y2, r, r1, Color;
  int gd, gm;
  gd = DETECT; // initialize the varialble for the graphics mode
  // gm is graphic mode which is a computer display mode that genetates image using
pixle
  // DETECT is a micro defined in "graphic.h" header file
  initgraph(&gd, &gm, NULL); // initgraph initialize the graphics system by loading a
graphics driver from disk

  cleardevice();                // the header file graphics.h contains cleardevice() function
which clear the screem in graphics mode and set the current position to (0,0)
                        // call the functions
  DDALine(100, 113, 50, 200, 10); // x1,y1,x2,y2,color
  DDALine(50, 200, 150, 200, 10);
  DDALine(150, 200, 100, 113, 10);
  r = 50 / sqrt(3); // formulae to find out radius of small circle
  x1 = (100 + 50 + 150) / 3;
  y1 = (113 + 200 + 200) / 3;

  circle(x1, y1, r); // draw small circle

  r1 = 100 / sqrt(3);
  circle(x1, y1, r1); // draw outer circle
  delay(10000);       // delay() function is used to hold execution of program
```

```
  return 0;
}
void DDALine(int x1, int y1, int x2, int y2, int Color)
{
  float dX, dY, Steps;
  float xinc, yinc, i, x, y;

  dX = x2 - x1;
  dY = y2 - y1;

  if (abs(dX) > abs(dY))
  {

    Steps = abs(dX);
  }
  else
  {
    Steps = abs(dY);
  }
  xinc = dX / Steps;
  yinc = dY / Steps;

  x = x1;
  y = y1;

  for (i = 1; i <= Steps; i++)
  {
    putpixel(x, y, Color);
    x = x + xinc;
    y = y + yinc;
  }
}
```

4.BASIC 2 DIMENSIONAL TRANSFORMATIONS

```
#include<iostream>

#include<stdlib.h>

#include<graphics.h>

#include<math.h>
```

```
using namespace std;

class POLYGON
{
   private:
      int p[10][10],Trans_result[10][10],Trans_matrix[10][10];
      float Rotation_result[10][10],Rotation_matrix[10][10];
      float Scaling_result[10][10],Scaling_matrix[10][10];
      float Shearing_result[10][10],Shearing_matrix[10][10];
      int Reflection_result[10][10],Reflection_matrix[10][10];

   public:
         int accept_poly(int [][10]);
         void draw_poly(int [][10],int);
         void draw_polyfloat(float [][10],int);
         void matmult(int [][10],int [][10],int,int,int,int [][10]);
         void matmultfloat(float [][10],int [][10],int,int,int,float [][10]);
         void shearing(int [][10],int);
         void scaling(int [][10],int);
         void rotation(int [][10],int);
         void translation(int [][10],int);
         void reflection(int [][10],int);
};

int POLYGON :: accept_poly(int p[][10])
{
         int i,n;
         cout<<"\n\nEnter number of vertices : ";
         cin>>n;
         for(i=0;i<n;i++)
         {
                  cout<<"\n\nEnter (x,y) Co-ordinate of point P"<<i<<" : ";
```

```
            cin >> p[i][0] >> p[i][1];
            p[i][2] = 1;
        }

        for(i=0;i<n;i++)
        {
            cout<<"\n";
            for(int j=0;j<3;j++)
            {
                cout<<p[i][j]<<"\t\t";
            }
        }

        return n;
}
void POLYGON :: draw_poly(int p[][10], int n)
{
        int i,gd = DETECT,gm;
        initgraph(&gd,&gm,NULL);
        line(320,0,320,480);
        line(0,240,640,240);

        for(i=0;i<n;i++)
        {
            if(i<n-1)
            {
                line(p[i][0]+320, -p[i][1]+240, p[i+1][0]+320, -p[i+1][1]+240);

            }
            else
                line(p[i][0]+320, -p[i][1]+240, p[0][0]+320, -p[0][1]+240);
        }
```

```
}

void POLYGON :: draw_polyfloat(float p[][10], int n)
{
	int i,gd = DETECT,gm;
	initgraph(&gd,&gm,NULL);
	line(320,0,320,480);
	line(0,240,640,240);

	for(i=0;i<n;i++)
	{
		if(i<n-1)
		{
			line(p[i][0]+320, -p[i][1]+240, p[i+1][0]+320, -p[i+1][1]+240);

		}
		else
			line(p[i][0]+320, -p[i][1]+240, p[0][0]+320, -p[0][1]+240);
	}


}


void POLYGON :: translation(int p[10][10],int n)
{
	int tx,ty,i,j; int i1,j1,k1,r1,c1,c2;
      r1=n;c1=c2=3;
	cout << "\n\nEnter X-Translation tx : ";
```

```
        cin >> tx;
        cout << "\n\nEnter Y-Translation ty : ";
        cin >> ty;
        for(i=0;i<3;i++)
        for(j=0;j<3;j++)
                Trans_matrix[i][j] = 0;
        Trans_matrix[0][0] = Trans_matrix[1][1] = Trans_matrix[2][2] = 1;
        Trans_matrix[2][0] = tx;
        Trans_matrix[2][1] = ty;

        for(i1=0;i1<10;i1++)
        for(j1=0;j1<10;j1++)
                Trans_result[i1][j1] = 0;
        for(i1=0;i1<r1;i1++)
        for(j1=0;j1<c2;j1++)
        for(k1=0;k1<c1;k1++)
                Trans_result[i1][j1] = Trans_result[i1][j1]+(p[i1][k1] * Trans_matrix[k1][j1]);
        cout << "\n\nPolygon after Translation : ";
        draw_poly(Trans_result,n);
}

void POLYGON :: rotation(int p[][10],int n)
{
        float type,Ang,Sinang,Cosang;
      int i,j; int i1,j1,k1,r1,c1,c2;
      r1=n;c1=c2=3;
        cout << "\n\nEnter the angle of rotation in degrees : ";
        cin >> Ang;
        cout << "\n\n* * * * Rotation Types * * * *";
        cout << "\n\n1.Clockwise Rotation \n\n2.Anti-Clockwise Rotation ";
        cout << "\n\nEnter your choice(1-2): ";
        cin >> type;
```

```
    Ang = (Ang * 6.2832)/360;
    Sinang = sin(Ang);
    Cosang = cos(Ang);
  cout<<"Mark1";
    for(i=0;i<3;i++)
    for(j=0;j<3;j++)
            Rotation_matrix[i][j] = 0;
  cout<<"Mark2";
    Rotation_matrix[0][0] = Rotation_matrix[1][1] = Cosang;
    Rotation_matrix[0][1] = Rotation_matrix[1][0] = Sinang;
    Rotation_matrix[2][2] = 1;
    if(type == 1)
            Rotation_matrix[0][1] = -Sinang;
    else
            Rotation_matrix[1][0] = -Sinang;

  for(i1=0;i1<10;i1++)
    for(j1=0;j1<10;j1++)
            Rotation_result[i1][j1] = 0;
    for(i1=0;i1<r1;i1++)
    for(j1=0;j1<c2;j1++)
    for(k1=0;k1<c1;k1++)
            Rotation_result[i1][j1] = Rotation_result[i1][j1]+(p[i1][k1] *
Rotation_matrix[k1][j1]);

    cout << "\n\nPolygon after Rotation : ";
  for(i=0;i<n;i++)
    {
            cout<<"\n";
            for(int j=0;j<3;j++)
            {
                    cout<<Rotation_result[i][j]<<"\t\t";
```

```
                }
        }
        draw_polyfloat(Rotation_result,n);
}

void POLYGON :: scaling(int p[][10],int n)
{
        float Sx,Sy;
      int i,j; int i1,j1,k1,r1,c1,c2;
      r1=n;c1=c2=3;
        cout<<"\n\nEnter X-Scaling Sx : ";
        cin>>Sx;
        cout<<"\n\nEnter Y-Scaling Sy : ";
        cin>>Sy;

        for(i=0;i<3;i++)
        {
                for(j=0;j<3;j++)
                {
                        Scaling_matrix[i][j] = 0;
                }
        }

        Scaling_matrix[0][0] = Sx;
        Scaling_matrix[0][1] = 0;
        Scaling_matrix[0][2] = 0;
        Scaling_matrix[1][0] = 0;
        Scaling_matrix[1][1] = Sy;
        Scaling_matrix[1][2] = 0;
        Scaling_matrix[2][0] = 0;
        Scaling_matrix[2][1] = 0;
        Scaling_matrix[2][2] = 1;
```

```
	for(i1=0;i1<10;i1++)
		for(j1=0;j1<10;j1++)
			Scaling_result[i1][j1] = 0;
		for(i1=0;i1<r1;i1++)
		for(j1=0;j1<c2;j1++)
		for(k1=0;k1<c1;k1++)
			Scaling_result[i1][j1] = Scaling_result[i1][j1]+(p[i1][k1] *
Scaling_matrix[k1][j1]);


		cout<<"\n\nPolygon after Scaling : ";
		draw_polyfloat(Scaling_result,n);
}


void POLYGON :: shearing(int p[][10],int n)
{
		float Sx,Sy,type; int i,j;
	int i1,j1,k1,r1,c1,c2;
	r1=n;c1=c2=3;
		for(i=0;i<3;i++)
		for(j=0;j<3;j++)
		{
			if(i == j)
				Shearing_matrix[i][j] = 1;
			else
				Shearing_matrix[i][j] = 0;
		}
		cout << "\n\n* * * * Shearing Types * * * *";
		cout << "\n\n1.X-Direction Shear \n\n2.Y-Direction Shear ";
		cout << "\n\nEnter your choice(1-2) : ";
		cin >> type;
		if(type == 1)
```

```
        {
                cout << "\n\nEnter X-Shear Sx : ";
                cin >> Sx;
                Shearing_matrix[1][0] = Sx;
        }
        else
        {
                cout << "\n\nEnter Y-Shear Sy : ";
                cin >> Sy;
                Shearing_matrix[0][1] = Sy;
        }



      for(i1=0;i1<10;i1++)
        for(j1=0;j1<10;j1++)
        Shearing_result[i1][j1] = 0;
        for(i1=0;i1<r1;i1++)
        for(j1=0;j1<c2;j1++)
        for(k1=0;k1<c1;k1++)
                Shearing_result[i1][j1] = Shearing_result[i1][j1]+(p[i1][k1] *
Shearing_matrix[k1][j1]);


        cout << "\n\nPolygon after Shearing : ";
        draw_polyfloat(Shearing_result,n);
}

void POLYGON :: reflection(int p[][10],int n)
{
        int type,i,j;


      int i1,j1,k1,r1,c1,c2;
 r1=n;c1=c2=3;
```

```
	cout << "\n\n* * * * Reflection Types * * * *";
	cout << "\n\n1.About X-Axis \n\n2.About Y-Axis \n\n3.About Origin\n\n4.About Line y = x \n\n5.About Line y = -x \n\nEnter your choice(1-5) : ";
	cin >> type;
	for(i=0;i<3;i++)
	for(j=0;j<3;j++)
	{
			Reflection_matrix[i][j] = 0;
	}
	switch(type)
	{
		case 1:
			Reflection_matrix[0][0] = 1;
		Reflection_matrix[1][1] = -1;
		Reflection_matrix[2][2] = 1;
			break;
		case 2:
			Reflection_matrix[0][0] = -1;
		Reflection_matrix[1][1] = 1;
		Reflection_matrix[2][2] = 1;
			break;
		case 3:
			Reflection_matrix[0][0] = -1;
		Reflection_matrix[1][1] = -1;
		Reflection_matrix[2][2] = 1;
			break;
		case 4:
			Reflection_matrix[0][1] = 1;
			Reflection_matrix[1][0] = 1;
			Reflection_matrix[2][2] = 1;
			break;
		case 5:
```

```
                        Reflection_matrix[0][1] = -1;
                        Reflection_matrix[1][0] = -1;
                        Reflection_matrix[2][2] = 1;
                        break;
        }

    for(i1=0;i1<10;i1++)
        for(j1=0;j1<10;j1++)
                Reflection_result[i1][j1] = 0;
        for(i1=0;i1<r1;i1++)
        for(j1=0;j1<c2;j1++)
        for(k1=0;k1<c1;k1++)
                Reflection_result[i1][j1] = Reflection_result[i1][j1]+(p[i1][k1] *
Reflection_matrix[k1][j1]);

        cout << "\n\n\t\tPolygon after Reflection : ";
//cout << "\n\n\t\tPolygon after Rotation…";
    for(i=0;i<n;i++)
        {
                cout<<"\n";
                for(int j=0;j<3;j++)
                {
                        cout<<Reflection_result[i][j]<<"\t\t";
                }
        }
        draw_poly(Reflection_result,n);
//closegraph();
}



int main()
```

```
{
        int ch,n,p[10][10];
        POLYGON p1;
        cout<<"\n\n* * * * 2-D TRANSFORMATION * * * *";
        n= p1.accept_poly(p);

        cout <<"\n\nOriginal Polygon : ";
        p1.draw_poly(p,n);
        do
        {

           int ch;
              cout<<"\n\n* * * * 2-D TRANSFORMATION * * * *";
                 cout<<"\n\n1.Translation \n\n2.Scaling \n\n3.Rotation \
              \n\n4.Reflection \n\n5.Shearing \n\n6.Exit";
              cout<<"\n\nEnter your choice(1-6) : ";
              cin>>ch;

                 switch(ch)
                 {
                          case 1:

                                   p1.translation(p,n);
                                   break;

                          case 2:

                                   p1.scaling(p,n);
                                   break;

                          case 3:
```

```
                    p1.rotation(p,n);
                    break;

                case 4:

                    p1.reflection(p,n);
                    break;

                case 5:

                    p1.shearing(p,n);
                    break;

                case 6:
                    exit(0);
            }
        }while(1);
        return 0;
}
```

5. CURVES AND FRACTALS

```
#include <iostream>
#include <stdlib.h>
#include <graphics.h>
#include <math.h>

using namespace std;
```

```
void move(int j,int h,int &x,int &y)
{
if(j==1)
y-=h;
else if(j==2)
x+=h;
else if(j==3)
y+=h;
else if(j==4)
x-=h;
lineto(x,y);
}

void hilbert(int r,int d,int l,int u,int i,int h,int &x,int &y)
{
if(i>0)
{
i--;
hilbert(d,r,u,l,i,h,x,y);
move(r,h,x,y);
hilbert(r,d,l,u,i,h,x,y);
move(d,h,x,y);
hilbert(r,d,l,u,i,h,x,y);
move(l,h,x,y);
hilbert(u,l,d,r,i,h,x,y);
}
}

int main()
{
int n,x1,y1;
```

```
int x0=50,y0=150,x,y,h=10,r=2,d=3,l=4,u=1;

cout<<"\nGive the value of n: ";
cin>>n;
x=x0;y=y0;
int gm,gd=DETECT;
initgraph(&gd,&gm,NULL);
moveto(x,y);
hilbert(r,d,l,u,n,h,x,y);
delay(10000);

closegraph();

return 0;
}
```

6. IMPLEMENTATION OF OPENGL LIBRARY & FUNCTIONS

```
#include<iostream>
#include<graphics.h>
#include<cstdlib>
#include<dos.h>
#include<cmath>
using namespace std;

int main()
{
```

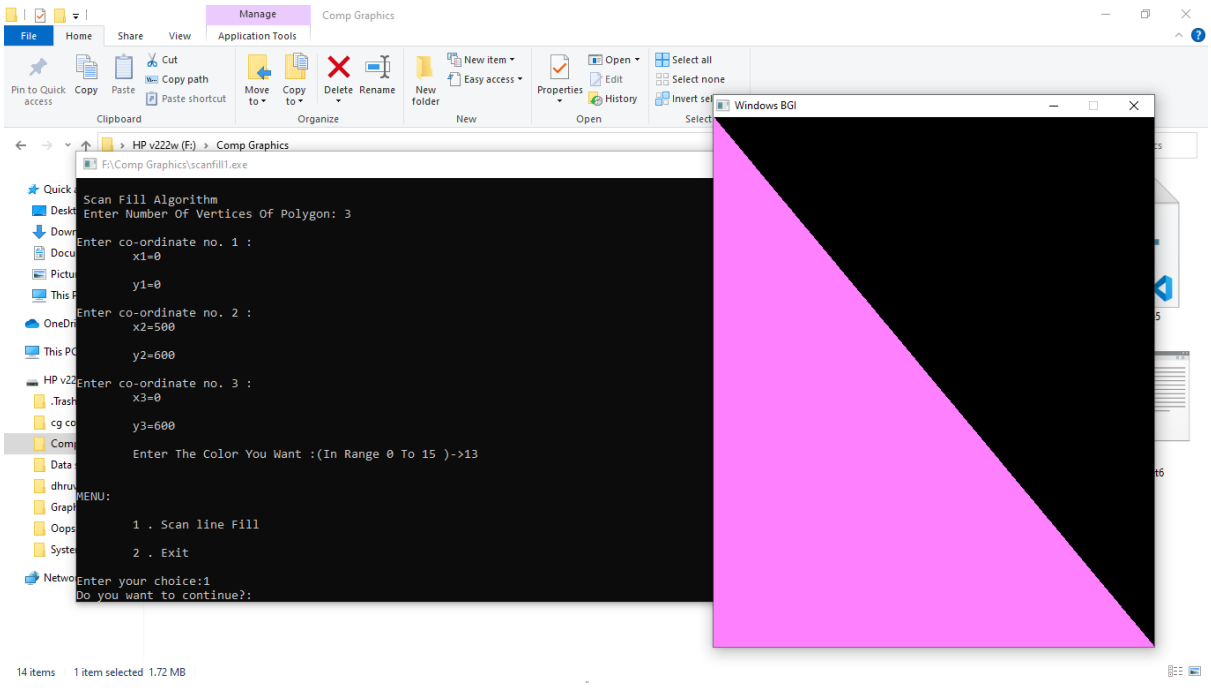
```
initwindow(800,500);
int x0,y0;
int gdriver = DETECT,gmode,errorcode;
int xmax,ymax;

errorcode=graphresult();

if(errorcode!=0)
{
   cout<<"Graphics error:"<<grapherrormsg(errorcode);
   cout<<"Press any ket to halt";
   exit(1);
}
int i,j;
setbkcolor(BLUE);
setcolor(RED);
rectangle(0,0,getmaxx(),getmaxy());

outtextxy(250,240,"::::PRESS ANY KEY TO CONTINUE:::::");
while(!kbhit());
for(i=50,j=0;i<=250,j<=250;i+=5,j+=5)
{
   delay(120);
   cleardevice();
   if(i<=150)
   {
      setcolor(YELLOW);
      setfillstyle(1,YELLOW);
      fillellipse(i,300-j,20,20);
   }
   else
   {
```

```
            setcolor(GREEN^RED);
            setfillstyle(1,GREEN^RED);
            fillellipse(i,300-j,20,20);
        }
    }
    delay(1000);
    cleardevice();
    setcolor(RED);
    setfillstyle(1,RED);
    fillellipse(300,50,20,20);
    delay(150);
    int k,l;
    for(k=305,l=55;k<=550,l<=300;k+=5,l+=5)
    {
        delay(120);
        cleardevice();
        if(k<=450)
        {

            setcolor(GREEN^RED);
            setfillstyle(1,GREEN^RED);
            fillellipse(k,l,20,20);
        }
        else
        {
            setcolor(YELLOW);
            setfillstyle(1,YELLOW);
            fillellipse(k,l,20,20);
        }
    }
    return 0;
}
```

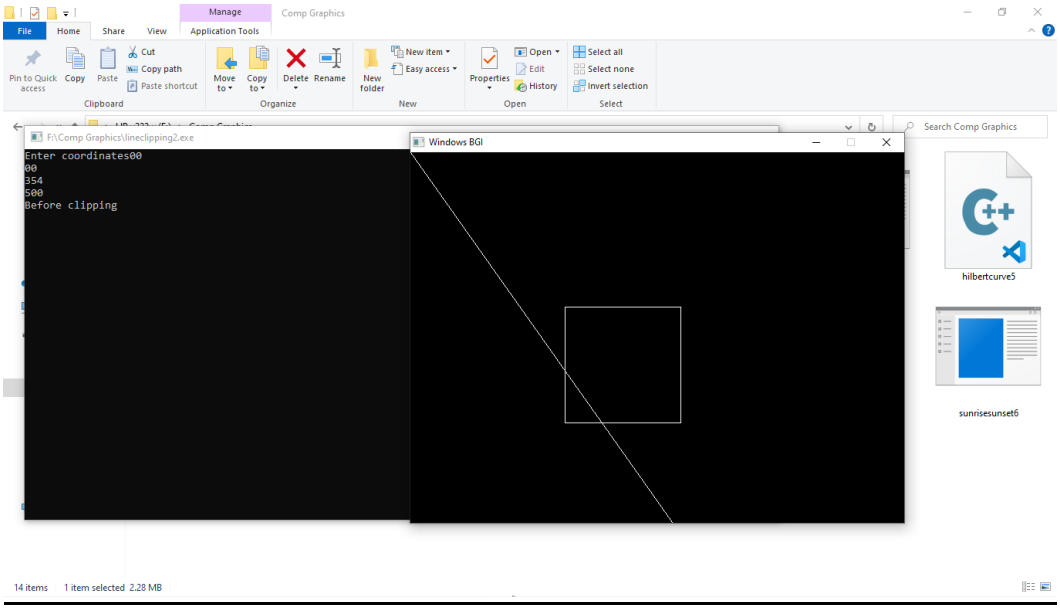
7. ANIMATION USING C++ PROGRAMMING

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

int main(){
   int gd=DETECT,gm;
   initgraph(&gd,&gm,"");
   float x=1,y,i,j=0.5,k,temp=0.5;
   setcolor(BLUE);
   for(k=0;k<7;k++){
      for(i=90;i<270;i+=10){
         y=cos(i*3.14/180);
         if(y>0)
         y=-y;
         x+=5;
         setfillstyle(SOLID_FILL,WHITE);
         circle(x,y*100+200,25);
         floodfill(x,y*100+200,BLUE);
         line(0,225,800,225);
         delay(50);
         cleardevice();
      }
      j+=temp;
      temp+=0.1;
   }
   getch();
}
```

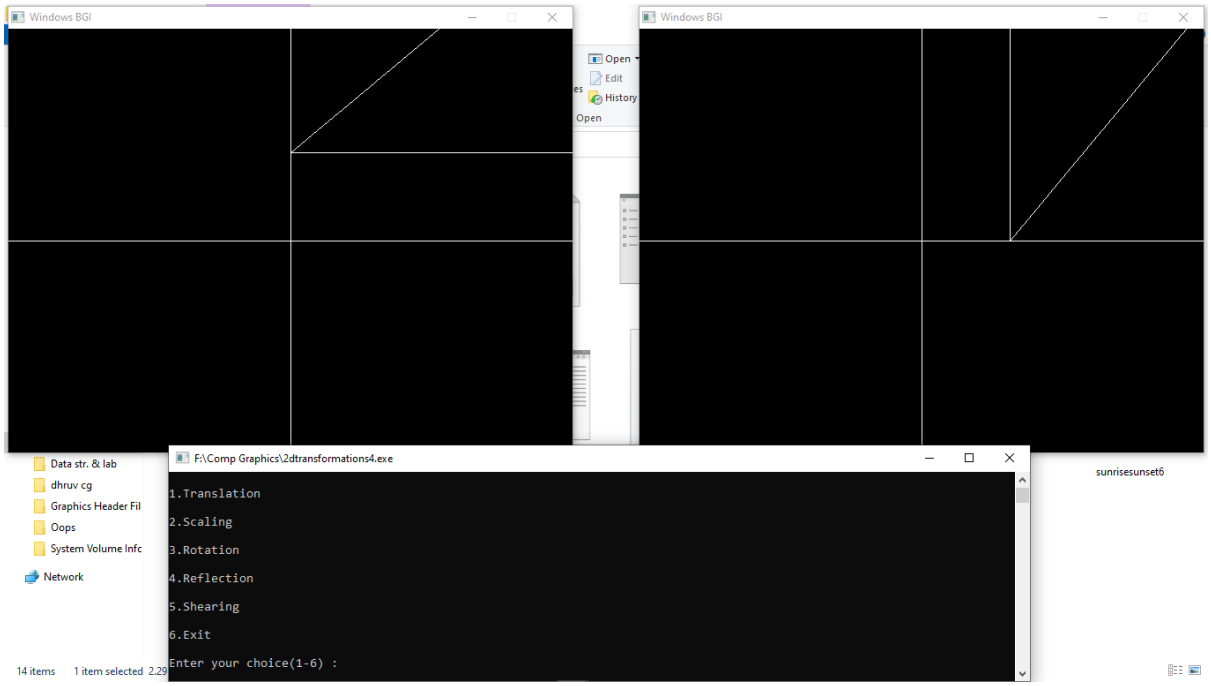
CONCAVE POLYGON FILLING USING SCAN FILL ALGORITHM



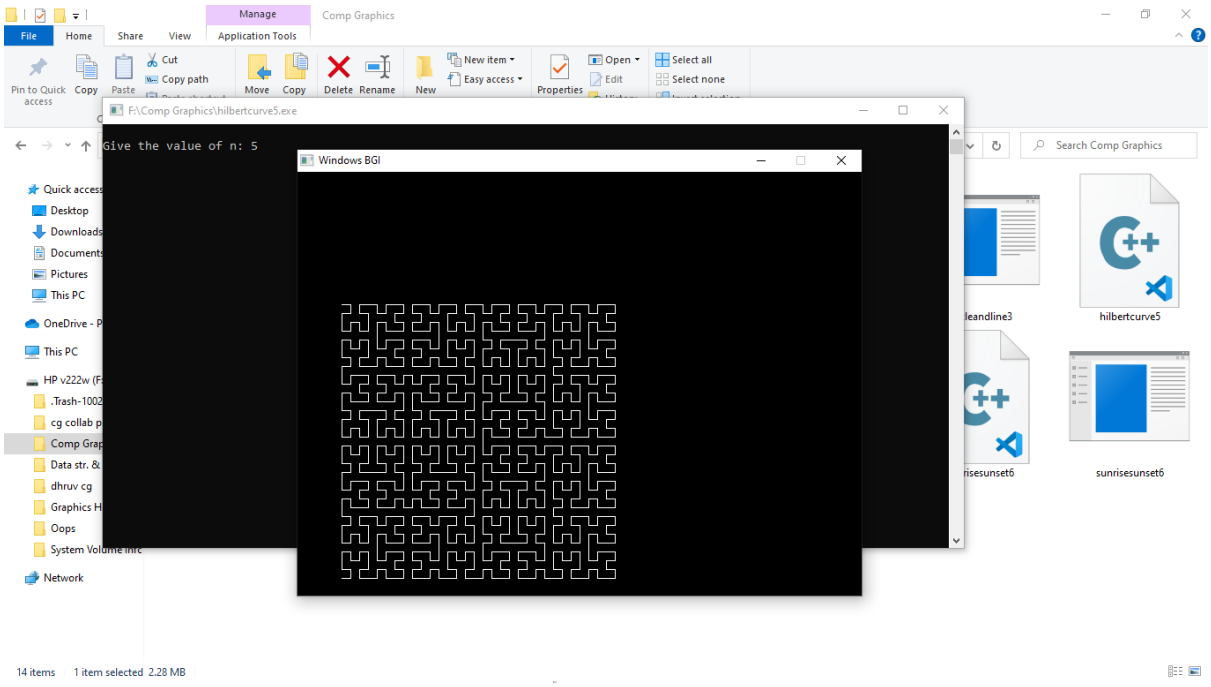
POLYGON CLIPPING USING COHEN SUTHERLAND LINE CLIPPING ALGORITHM



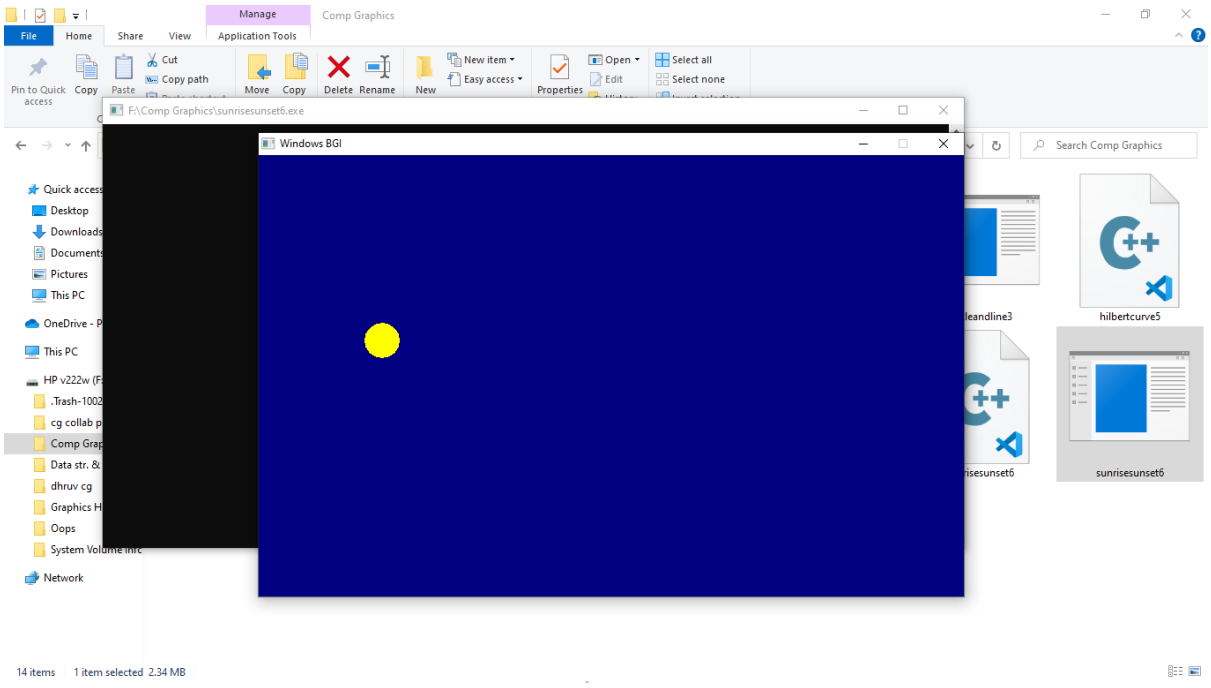
BASIC 2 DIMENSIONAL TRANSFORMATIONS

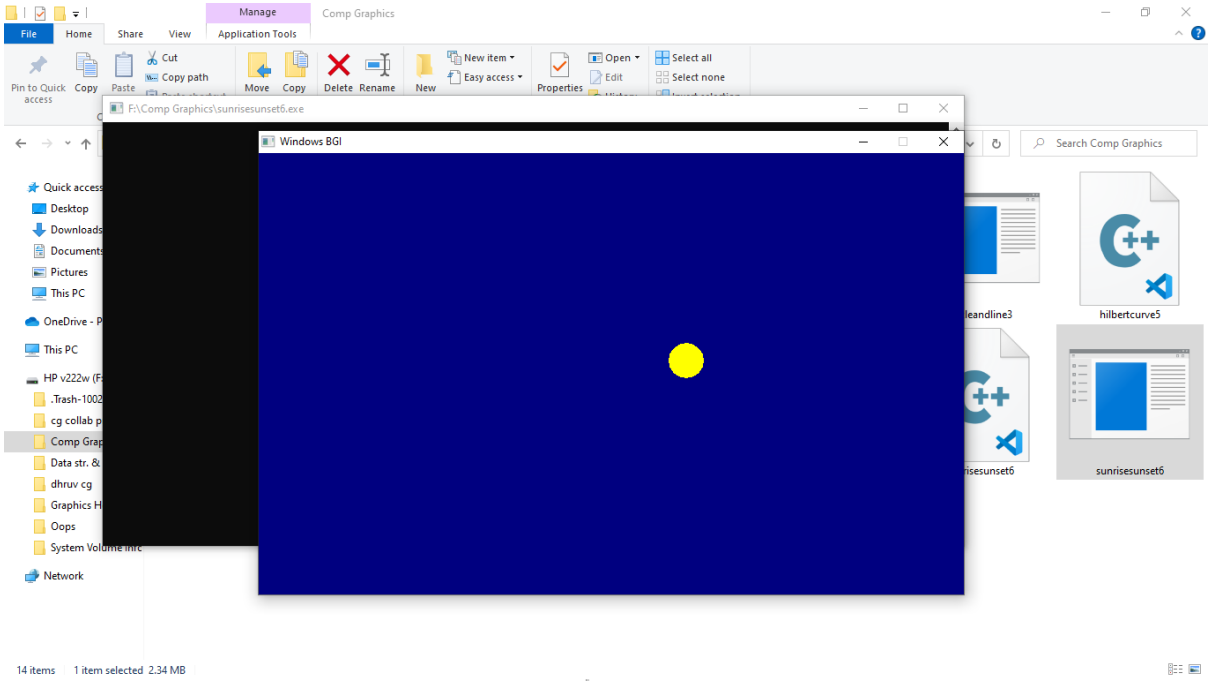


CURVES AND FRACTALS



IMPLEMENTATION OF OPENGL LIBRARY & FUNCTIONS





ANIMATION USING C++ PROGRAMMING