1. **Flag**

#include<graphics.h>

int main()

{

initwindow(1000, 1000);

setcolor(GREEN);

rectangle(100, 100, 500, 300);

setfillstyle(SOLID\_FILL, GREEN);

floodfill(200, 104, GREEN);

setcolor(RED);

circle(300, 200, 50);

setfillstyle(SOLID\_FILL, RED);

floodfill(300, 204, RED);

getch();

}

1. **2D transformation**

#include<graphics.h>

#include<iostream>

#include<math.h>

#define PI 3.14

using namespace std;

void translation(int x[], int y[], int tx, int ty)

{

//x1+=tx; y1+=ty; x2+=tx; y2+=ty;

setcolor(RED);

for(int i=0; i<4; i++)

{

line(x[i]+tx, y[i]+ty, x[(i+1)%4]+tx, y[(i+1)%4]+ty);

}

}

void scaling(int x[], int y[], float sx, float sy)

{

setcolor(BLUE);

for(int i=0; i<4; i++)

{

line(x[i]\*sx, y[i]\*sy, x[(i+1)%4]\*sx, y[(i+1)%4]\*sy);

}

}

void rotation(int x[], int y[], float angle)

{

setcolor(GREEN);

for(int i=0; i<4; i++)

{

int rx1 = x[i]\*cos(angle) - y[i]\*sin(angle);

int ry1 = x[i]\*sin(angle) + y[i]\*cos(angle);

int rx2 = x[(i+1)%4]\*cos(angle) - y[(i+1)%4]\*sin(angle);

int ry2 = x[(i+1)%4]\*sin(angle) + y[(i+1)%4]\*cos(angle);

line(rx1, ry1, rx2, ry2);

}

}

int main()

{

initwindow(1000, 1000);

int x[] = {200, 200, 500, 500};

int y[] = {200, 500, 500, 200};

for(int i=0; i<4; i++)

{

line(x[i], y[i], x[(i+1)%4], y[(i+1)%4]);

}

// int x1 = 200, y1 = 200, x2 = 500, y2 = 500;

// rectangle(x1, y1, x2, y2);

//translation vector

int tx = 295, ty = 295;

translation(x, y, tx, ty);

//

//scaling factors

float sx = 0.7, sy = 0.7;

scaling(x, y, sx, sy);

//

float angle = 15\*PI/180;

rotation(x, y, angle);

getch();

return 0;

}

1. **Bezier Curve**

#include<graphics.h>

#include<iostream>

#include<math.h>

using namespace std;

void combination(int n, int c[])

{

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

{

c[i] = 1;

for(int j=n; j>=i+1; j--)

c[i]\*=j;

for(int k=n-i; k>=2; k--)

c[i]/=k;

}

}

void draw\_curve(int points, int x\_points[], int y\_points[])

{

int degree = points-1;

int combinations[points];

combination(degree, combinations);

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

cout << combinations[i] << " ";

cout << endl;

freopen("output\_curve.txt", "w", stdout);

for(float u=0; u<=1; u+=0.002)

{

int px = 0, py = 0;

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

{

px+=x\_points[i]\*combinations[i]\*pow(u, i)\*pow(1-u, degree-i);

py+=y\_points[i]\*combinations[i]\*pow(u, i)\*pow(1-u, degree-i);

}

cout << px << " " << py << endl;

putpixel(px, py, WHITE);

}

}

int main()

{

int points;

freopen("input\_curve.txt","r", stdin);

cin >> points;

int x\_points[points];

int y\_points[points];

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

{

cin >> x\_points[i] >> y\_points[i];

}

initwindow(1000, 1000);

for(int i=0; i<4; i++)

putpixel(x\_points[i], y\_points[i], WHITE);

draw\_curve(points, x\_points, y\_points);

// int x\_points[] = {100, 200, 400, 600};

// int y\_points[] = {100, 300, 300, 400};

getch();

}

1. **Bresenham Line Drawing Algorithm**

#include<graphics.h>

#include<math.h>

void draw\_line(int x1, int y1, int x2, int y2)

{

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

int p = 2\*dy - dx;

putpixel(x1, y1, WHITE);

int y = y1;

while(x1<x2)

{

x1++;

if(p<0)

{

p+=2\*dy;

}

else

{

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

y+=1;

}

putpixel(x1, y, WHITE);

}

}

int main()

{

initwindow(1000, 1000);

// line(100, 100, 500, 100);

int x1=100, y1=100, x2=500, y2=300;

draw\_line(x1, y1, x2, y2);

getch();

return 0;

}

1. **Midpoint Circle Algorithm**

#include<graphics.h>

#include<iostream>

using namespace std;

void plotcirclepixels(int x\_center, int y\_center, int px, int py)

{

cout << px << " " << py << endl;

putpixel(x\_center+px, y\_center+py, WHITE);

putpixel(x\_center+py, y\_center+px, WHITE);

putpixel(x\_center+py, y\_center-px, WHITE);

putpixel(x\_center+px, y\_center-py, WHITE);

putpixel(x\_center-px, y\_center+py, WHITE);

putpixel(x\_center-py, y\_center+px, WHITE);

putpixel(x\_center-py, y\_center-px, WHITE);

putpixel(x\_center-px, y\_center-py, WHITE);

}

void findmidpoint(int x, int y, int r)

{

int p = 1 - r;

int px = 0, py = r;

plotcirclepixels(x, y, px, py);

while(px<py)

{

px++;

if(p<0)

{

p+=2\*px+1;

}

else

{

p+=2\*(px-py)+1;

py-=1;

}

plotcirclepixels(x, y, px, py);

}

}

int main()

{

initwindow(1000, 1000);

int x = 300, y =300, r = 100;

findmidpoint(x, y, r);

getch();

}

1. **Fractal Koch Curve**

#include<graphics.h>

#include<math.h>

#define PI 3.14

void draw\_koch\_curve(int x1, int y1, int x2, int y2, int no\_of\_iteration, bool outer\_direction)

{

float angle = 60\*PI/180;

int mid\_first\_x = x1+(x2-x1)/3;

int mid\_first\_y = y1 + (y2-y1)/3;

int mid\_sec\_x = x1+(x2-x1)\*2/3;

int mid\_sec\_y = y1 + (y2-y1)\*2/3;

int mid\_x, mid\_y;

if(outer\_direction)

{

mid\_x = mid\_first\_x+cos(angle)\*(mid\_sec\_x-mid\_first\_x) - sin(angle)\*(mid\_sec\_y-mid\_first\_y);

mid\_y =mid\_first\_y+sin(angle)\*(mid\_sec\_x-mid\_first\_x) + cos(angle)\*(mid\_sec\_y-mid\_first\_y);

}

else

{

mid\_x = mid\_first\_x+cos(angle)\*(mid\_sec\_x-mid\_first\_x) + sin(angle)\*(mid\_sec\_y-mid\_first\_y);

mid\_y =mid\_first\_y-sin(angle)\*(mid\_sec\_x-mid\_first\_x) + cos(angle)\*(mid\_sec\_y-mid\_first\_y);

}

if(no\_of\_iteration>0)

{

draw\_koch\_curve(x1, y1, mid\_first\_x, mid\_first\_y, no\_of\_iteration-1, outer\_direction);

draw\_koch\_curve(mid\_first\_x, mid\_first\_y, mid\_x, mid\_y, no\_of\_iteration-1, outer\_direction);

draw\_koch\_curve(mid\_x, mid\_y, mid\_sec\_x, mid\_sec\_y, no\_of\_iteration-1, outer\_direction);

draw\_koch\_curve(mid\_sec\_x, mid\_sec\_y, x2, y2, no\_of\_iteration-1, outer\_direction);

}

else

{

line(x1, y1, mid\_first\_x, mid\_first\_y);

line(mid\_first\_x, mid\_first\_y, mid\_x, mid\_y);

line(mid\_x, mid\_y, mid\_sec\_x, mid\_sec\_y);

line(mid\_sec\_x, mid\_sec\_y, x2, y2);

}

}

int main()

{

initwindow(1000, 1000);

int x1 = 300, y1 = 250, x2 = 700, y2 = 450;

float angle = 60\*PI/180;

int x3 = x1+cos(angle)\*(x2-x1) + sin(angle)\*(y2-y1), y3 =y1- sin(angle)\*(x2-x1) + cos(angle)\*(y2-y1);

draw\_koch\_curve(x1, y1, x2, y2, 3, false);

draw\_koch\_curve(x1, y1, x3, y3, 3, false);

draw\_koch\_curve(x2, y2, x3, y3, 3, true);

getch();

}

1. **Cohen Sutherland Line Clipping**

#include<graphics.h>

#include<iostream>

using namespace std;

void sutherland\_line\_clipping(int x1, int y1, int x2, int y2, int xwmin, int ywmin, int xwmax, int ywmax)

{

int region\_begin[] = {0, 0, 0, 0}, region\_end[] = {0, 0, 0, 0}, region\_code[4];

bool inside = true;

//begin point region code

if(x1<xwmin)

{

region\_begin[0] = 1;

inside = false;

}

if(x1>xwmax)

{

region\_begin[1]=1;

inside = false;

}

if(y1<ywmin)

{

region\_begin[2] = 1;

inside = false;

}

if(y1>ywmax)

{

region\_begin[3] = 1;

inside = false;

}

//end point region code

if(x2<xwmin)

{

region\_end[0] = 1;

inside = false;

}

if(x2>xwmax)

{

region\_end[1]=1;

inside = false;

}

if(y2<ywmin)

{

region\_end[2] = 1;

inside = false;

}

if(y2>ywmax)

{

region\_end[3] = 1;

inside = false;

}

if(inside)

return;

for(int i=0; i<4; i++)

{

region\_code[i] = region\_begin[i] && region\_end[i];

if(region\_code[i]==1)

{

return;

}

}

int slope = (y2-y1)/(x2-x1);

//find intersection of begin point

// left side

if(region\_begin[0]==1 && region\_begin[1]==0)

{

y1+=slope\*(xwmin-x1);

x1 = xwmin;

}

//right side

else if(region\_begin[0]==0 && region\_begin[1]==1)

{

y1+=slope\*(xwmax-x1);

x1 = xwmax;

}

//below side

if(region\_begin[2]==1 && region\_begin[3]==0)

{

x1+=(ywmin-y1)/slope;

y1 = ywmin;

}

else if(region\_begin[2]==0 && region\_begin[3]==1)

{

x1+=(ywmax-y1)/slope;

y1 = ywmax;

}

//find intersection of end point

//left side

if(region\_end[0]==1 && region\_end[1]==0)

{

y2+=(xwmin-x2)\*slope;

x2 = xwmin;

}

//right side

else if(region\_end[0]==0 && region\_end[1]==1)

{

y2+=(xwmax-x2)\*slope;

x2 = xwmax;

}

//below

if(region\_end[2]==1 && region\_end[3]==0)

{

x2+=(ywmin-y2)/slope;

y2 = ywmin;

}

//above

else if(region\_end[2]==0 && region\_end[3]==1)

{

x2+=(ywmax-y2)/slope;

y2=ywmax;

}

cout << x1 << " " << y1 << endl;

setcolor(RED);

line(x1, y1, x2, y2);

}

int main()

{

initwindow(1000, 1000);

int x1 = 200, y1 = 250, x2 = 500, y2 = 550;

line(x1, y1, x2, y2);

int xwmin = 220, ywmin = 270, xwmax = 510, ywmax = 500;

rectangle(xwmin, ywmin, xwmax, ywmax);

sutherland\_line\_clipping(x1, y1, x2, y2, xwmin, ywmin, xwmax, ywmax);

getch();

}

1. **Hidden Surface Elimination**

#include<graphics.h>

void circles()

{

setcolor(RED);

circle(100, 300, 50);

setfillstyle(SOLID\_FILL, RED);

floodfill(100, 305, RED);

}

void triangle()

{

setcolor(GREEN);

line(115, 265, 115, 335);

line(115, 265, 215, 300);

line(115, 335, 215, 300);

setfillstyle(SOLID\_FILL, GREEN);

floodfill(120, 300, GREEN);

}

void square()

{

setcolor(BLUE);

rectangle(80, 230, 170, 300);

setfillstyle(SOLID\_FILL, BLUE);

floodfill(90, 270, BLUE);

}

int main()

{

printf("Enter a number: ");

int number;

scanf("%d", &number);

initwindow(1000, 1000);

if(number==1)

{

square();

circles();

triangle();

}

else if(number==2)

{

circles();

square();

triangle();

}

else if(number==3)

{

circles();

triangle();

square();

}

else if(number==4)

{

triangle();

circles();

square();

}

else if(number==5)

{

triangle();

square();

circles();

}

else if(number==6)

{

square();

triangle();

circles();

}

getch();

}