**Experiment 1: Bresenham’s Line Drawing**

**Code:**

#include<stdio.h>

#include<graphics.h>

int x0 , y0 , x1 , y1;

void drawline()

{

int del\_x, del\_y, p, x, y;

del\_x=x1-x0;

del\_y=y1-y0;

X = x0 , y = y0;

p=2\*del\_y-del\_x;

while(x<x1){

if(p>=0){

putpixel(x,y,7);

y=y+1;

p=p+2\*del\_y-2\*del\_x;

}

else{

putpixel(x,y,7);

p=p+2\*del\_y;

}

x=x+1;

}

}

int main(){

int gd = DETECT, gm;

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

printf("Enter co-ordinates of first point: ");

scanf("%d%d", &x0, &y0);

printf("Enter co-ordinates of second point: ");

scanf("%d%d", &x1, &y1);

drawline();

closegraph();

return 0;

}

**Experiment 2: Midpoint Circle Drawing**

**Code:**

#include<stdio.h>

#include<graphics.h>

int x0 , y0 , r;

void drawcircle(){

int x = r , y = 0, p = 0;

while (x >= y){

putpixel(x0 + x, y0 + y, 7);

putpixel(x0 + y, y0 + x, 7);

putpixel(x0 - y, y0 + x, 7);

putpixel(x0 - x, y0 + y, 7);

putpixel(x0 - x, y0 - y, 7);

putpixel(x0 - y, y0 - x, 7);

putpixel(x0 + y, y0 - x, 7);

putpixel(x0 + x, y0 - y, 7);

if (p <= 0){

y += 1;

p += 2\*y + 1;

}

else{

x -= 1;

p -= 2\*x + 1;

}

}

}

int main(){

int gd = DETECT, gm;

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

printf("Enter radius of circle: ");

scanf("%d", &r);

printf("Enter co-ordinates of center(x and y): ");

scanf("%d%d", &x0, &y0);

drawcircle();

closegraph();

return 0;

}

**Experiment 3: 2D Translation, Rotation and Scaling**

**2D Translation:**

#include<stdio.h>

#include<graphics.h>

void translateLine (int P[][2], int T[])

{

setcolor (2);

line(P[0][0], P[0][1], P[1][0], P[1][1]);

P[0][0] = P[0][0] + T[0];

P[0][1] = P[0][1] + T[1];

P[1][0] = P[1][0] + T[0];

P[1][1] = P[1][1] + T[1];

setcolor(3);

line(P[0][0], P[0][1], P[1][0], P[1][1]);

closegraph();

}

int main()

{

int gd = DETECT, gm, errorcode;

initgraph (&gd, &gm, NULL);

int P[2][2] = {5, 8, 12, 18},T[] = {2, 1};

translateLine (P, T);

closegraph();

return 0;

}

**2D Rotation(triangle):**

#include<stdio.h>

#include<graphics.h>

float r,t;

void linerotate (float a, float b, float c, float d){

line (a\*cos(t)-b\*sin(t),a\*sin(t)+b\*cos(t),c\*cos(t)-d\*sin(t),c\*sin(t)+d\*cos(t));

}

int main()

{

int gd = DETECT, gm;

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

int X,Y,X1,Y1,X2,Y2;

printf("Enter the value of first coordinate:");

scanf("%d %d",&X,&Y);

printf("Enter the value of second coordinate:");

scanf("%d %d",&X1,&Y1);

printf("Enter the value of third coordinate:");

scanf("%d %d",&X2,&Y2);

printf("Enter rotation factor\n");

scanf("%f",&r);

t = (3.14/180)\*r;

setcolor(GREEN);

line(X,Y,X1,Y1);

line(X1,Y1,X2,Y2);

line(X2,Y2,X,Y);

setcolor(BLUE);

linerotate(X,Y,X1,Y1);

linerotate(X1,Y1,X2,Y2);

linerotate(X2,Y2,X,Y);

getch();

closegraph();

}

**2D Scaling(Triangle):**

#include<stdio.h>

#include<graphics.h>

void findNewCoordinate(int s[][2], int p[][1])

{

int temp[2][1] = { 0 };

for (int i = 0; i < 2; i++)

for (int j = 0; j < 1; j++)

for (int k = 0; k < 2; k++)

temp[i][j] += (s[i][k] \* p[k][j]);

p[0][0] = temp[0][0];

p[1][0] = temp[1][0];

}

void scale(int x[], int y[], int sx, int sy)

{

setcolor(2);

line(x[0], y[0], x[1], y[1]);

line(x[1], y[1], x[2], y[2]);

line(x[2], y[2], x[0], y[0]);

int s[2][2] = { sx, 0, 0, sy };

int p[2][1];

for (int i = 0; i < 3; i++)

{

p[0][0] = x[i];

p[1][0] = y[i];

findNewCoordinate(s, p);

x[i] = p[0][0];

y[i] = p[1][0];

}

setcolor(3);

line(x[0], y[0], x[1], y[1]);

line(x[1], y[1], x[2], y[2]);

line(x[2], y[2], x[0], y[0]);

}

int main()

{

int x[3] , y[3] , sx, sy;

printf("Enter Three Coordinates: ");

scanf("%d%d%d%d%d%d",&x[0],y[0],x[1],y[1],x[2],y[2]);

printf("Enter Scaling Factor (Sx and Sy): ");

scanf("%d%d",&sx,&sy);

int gd = DETECT, gm;

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

scale(x, y, sx,sy);

getch();

closegraph();

return 0;

}

**Experiment 4: Line Clipping and Polygon Clipping**

**Line Clipping(Using Cohen Sutherland Algorithm):**

#include<graphics.h>

#include<stdio.h>

#include<math.h>

void main()

{

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

int W\_xmax,W\_ymax,W\_xmin,W\_ymin,flag=0;

float slope;

int x,y,x1,y1,i, xc,yc;

int gr=DETECT,gm;

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

printf("\n\*\*\*\*\*\* Cohen Sutherland Line Clipping algorithm \*\*\*\*\*\*\*");

printf("\n Now, enter XMin, YMin =");

scanf("%d %d",&W\_xmin,&W\_ymin);

printf("\n First enter XMax, YMax =");

scanf("%d %d",&W\_xmax,&W\_ymax);

printf("\n Please enter intial point x and y= ");

scanf("%d %d",&x,&y);

printf("\n Now, enter final point x1 and y1= ");

scanf("%d %d",&x1,&y1);

cleardevice();

rectangle(W\_xmin,W\_ymin,W\_xmax,W\_ymax);

line(x,y,x1,y1);

line(0,0,600,0);

line(0,0,0,600);

if(y>W\_ymax) {

rcode\_begin[0]=1; // Top

flag=1 ;

}

if(y<W\_ymin) {

rcode\_begin[1]=1; // Bottom

flag=1;

}

if(x>W\_xmax) {

rcode\_begin[2]=1; // Right

flag=1;

}

if(x<W\_xmin) {

rcode\_begin[3]=1; //Left

flag=1;

}

//end point of Line

if(y1>W\_ymax){

rcode\_end[0]=1; // Top

flag=1;

}

if(y1<W\_ymin) {

rcode\_end[1]=1; // Bottom

flag=1;

}

if(x1>W\_xmax){

rcode\_end[2]=1; // Right

flag=1;

}

if(x1<W\_xmin){

rcode\_end[3]=1; //Left

flag=1;

}

if(flag==0)

printf("No need of clipping as it is already in window");

flag=1;

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

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

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

flag=0;

}

if(flag==0){

printf("\n Line is completely outside the window");

}

else{

slope=(float)(y1-y)/(x1-x);

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

y=y+(float) (W\_xmin-x)\*slope ;

x=W\_xmin;

}

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

y=y+(float) (W\_xmax-x)\*slope ;

x=W\_xmax;

}

if(rcode\_begin[0]==1 && rcode\_begin[1]==0){

x=x+(float) (W\_ymax-y)/slope ;

y=W\_ymax;

}

if(rcode\_begin[0]==0 && rcode\_begin[1]==1){

x=x+(float) (W\_ymin-y)/slope ;

y=W\_ymin;

}

// end points

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

y1=y1+(float) (W\_xmin-x1)\*slope ;

x1=W\_xmin;

}

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

y1=y1+(float) (W\_xmax-x1)\*slope ;

x1=W\_xmax;

}

if(rcode\_end[0]==1 && rcode\_end[1]==0){

x1=x1+(float) (W\_ymax-y1)/slope ;

y1=W\_ymax;

}

if(rcode\_end[0]==0 && rcode\_end[1]==1){

x1=x1+(float) (W\_ymin-y1)/slope ;

y1=W\_ymin;

}

}

delay(1000);

clearviewport();

rectangle(W\_xmin,W\_ymin,W\_xmax,W\_ymax);

line(0,0,600,0);

line(0,0,0,600);

setcolor(RED);

line(x,y,x1,y1);

getch();

closegraph();

}

**Polygon Clipping(Using Sutherland Hodgeman Algorithm):**

#include<stdio.h>

#include<graphics.h>

#include<stdlib.h>

int main()

{

int gd,gm,n,\*x,i,k=0,wx1=220,wy1=140,wx2=420,wy2=140,wx3=420,wy3=340;

int wx4=220,wy4=340;

int w[]={220,140,420,140,420,340,220,340,220,140};//array for drawing window

int gr=DETECT,gm;

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

printf("Window:-");

setcolor(RED); //red colored window

drawpoly(5,w); //window drawn

printf("Enter the no. of vertices of polygon: ");

scanf("%d",&n);

x = malloc(n\*2+1);

printf("Enter the coordinates of points:\n");

k=0;

for(i=0;i<n\*2;i+=2) //reading vertices of polygon

{

printf("(x%d,y%d): ",k,k);

scanf("%d,%d",&x[i],&x[i+1]);

k++;

}

x[n\*2]=x[0];

x[n\*2+1]=x[1];

setcolor(WHITE);

drawpoly(n+1,x);

printf("\nPress a button to clip a polygon..");

getch();

setcolor(RED);

drawpoly(5,w);

setfillstyle(SOLID\_FILL,BLACK);

floodfill(2,2,RED);

gotoxy(1,1); //bringing cursor at starting position

printf("\nThis is the clipped polygon..");

getch();

cleardevice();

closegraph();

}

**Experiment 5: Bezier Curves**

**Code:**

#include<graphics.h>

#include<math.h>

#include<conio.h>

#include<stdio.h>

void main()

{

int x[4],y[4],i;

double put\_x,put\_y,t;

int gr=DETECT,gm;

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

printf("\n\*\*\*\*\*\* Bezier Curve \*\*\*\*\*\*\*\*\*\*\*");

printf("\n Please enter x and y coordinates ");

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

{

scanf("%d%d",&x[i],&y[i]);

putpixel(x[i],y[i],3);

}

for(t=0.0;t<=1.0;t=t+0.001)

{

put\_x = pow(1-t,3)\*x[0] + 3\*t\*pow(1-t,2)\*x[1] + 3\*t\*t\*(1-t)\*x[2] + pow(t,3)\*x[3];

put\_y = pow(1-t,3)\*y[0] + 3\*t\*pow(1-t,2)\*y[1] + 3\*t\*t\*(1-t)\*y[2] + pow(t,3)\*y[3];

putpixel(put\_x,put\_y, WHITE);

}

getch();

closegraph();

return 0;

}

**Experiment 6: Koch Curve(Fractal)**

**Code:**

#include<graphics.h>

#include<stdlib.h>

#include<stdio.h>

#include<math.h>

#define pi M\_PI

typedef struct{

double x,y;

}point;

void kochCurve(point p1,point p2,int times){

point p3,p4,p5;

double theta = pi/3;

if(times>0){

p3 = (point){(2\*p1.x+p2.x)/3,(2\*p1.y+p2.y)/3};

p5 = (point){(2\*p2.x+p1.x)/3,(2\*p2.y+p1.y)/3};

p4 = (point){p3.x + (p5.x - p3.x)\*cos(theta)

+ (p5.y - p3.y)\*sin(theta),p3.y - (p5.x - p3.x)\*sin(theta)

+ (p5.y - p3.y)\*cos(theta)};

kochCurve(p1,p3,times-1);

kochCurve(p3,p4,times-1);

kochCurve(p4,p5,times-1);

kochCurve(p5,p2,times-1);

}

else

line(p1.x,p1.y,p2.x,p2.y);

}

int main(int argC, char\*\* argV)

{

int w,h,r;

point p1,p2;

if(argC!=4){

printf("Usage : %s <window width> <window height> <recursion level>",argV[0]);

}

else{

w = atoi(argV[1]);

h = atoi(argV[2]);

r = atoi(argV[3]);

initwindow(w,h,"Koch Curve");

p1 = (point){10,h-10};

p2 = (point){w-10,h-10};

kochCurve(p1,p2,r);

getch();

closegraph();

}

return 0;

}