NOT DONE QUESTION- 15, 16,22,32,33

**1. Write a Program to implement DDA Line drawing algorithm. (dda)**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

void main(){

int x,y,x1,y1,x2,y2,dx,dy,step,xinc,yinc,i,gd=DETECT,gm;

clrscr();

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

printf("ENTER START CO-ORDINATES:\n");

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

printf("ENTER END CO-ORDINATES:\n");

scanf("%d%d",&x2,&y2);

dx=x2-x1;

dy=y2-y1;

if(abs(dx)>=abs(dy))

step=abs(dx);

else

step=abs(dy);

xinc=dx/step;

yinc=dy/step;

x=x1;

y=y1;

putpixel(x,y,4);

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

x=x+xinc;

y=y+yinc;

putpixel(x,y,4);

}

getch();

closegraph();

}

**2. Write a Program to implement Bresenham’s Line drawing algorithm. (bline)**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<stdlib.h>

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

int dx=abs(x2-x1);

int dy=abs(y2-y1);

int sx=(x1 < x2)?1:-1;

int sy=(y1 < y2)?1:-1;

int err=dx-dy;

int e2;

while(x1!=x2 || y1!=y2){

putpixel(x1,y1,WHITE);

e2=2\*err;

if(e2 > -dy){

err-=dy;

x1+=sx;

}

if(e2<dx){

err+=dx;

y1+=sy;

}

}

}

void main(){

int x1,y1,x2,y2;

int gd=DETECT,gm;

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

while(1){

printf("enter x1,y1,x2,y2:");

if(scanf("%d,%d,%d,%d",&x1,&y1,&x2,&y2)!=4){

printf("invalid input");

break;

}

bline(x1,y1,x2,y2);

}

getch();

closegraph();

return 0;

}

**3. Write a Program to implement Bresenham’s Circle drawing algorithm. (bcircle)**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

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

{

int i=0,j=r,p=3-2\*r;

while(i<=j)

{

putpixel(x+i,y+j,WHITE);

putpixel(x-i,y-j,WHITE);

putpixel(x+j,y+i,WHITE);

putpixel(x-j,y-i,WHITE);

putpixel(x-i,y+j,WHITE);

putpixel(x+i,y-j,WHITE);

putpixel(x-j,y+i,WHITE);

putpixel(x+j,y-i,WHITE);

if(p<0)

{

i++;

p=p+4\*i+6;

}

else

{

i++;

j--;

p=p+4\*(i-j)+10;

}

}

}

int main()

{

int gd=DETECT,gm;

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

cir(250, 110, 50);

cir(300, 120, 50);

cir(350, 130, 50);

cir(275, 150, 50);

cir(325, 170, 50);

cir(300, 180, 50);

getch();

closegraph();

return 0;

}

**4. Write a Program to implement Mid-point Circle drawing algorithm. (mcircle)**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void midpointCircle(int x, int y, int radius);

void midpointCircle(int x, int y, int radius) {

int p = 1 - radius;

int i = 0;

int j = radius;

while (i <= j) {

putpixel(x + i, y + j, WHITE);

putpixel(x + j, y + i, WHITE);

putpixel(x - j, y + i, WHITE);

putpixel(x + j, y - i, WHITE);

putpixel(x - i, y - j, WHITE);

putpixel(x - j, y - i, WHITE);

putpixel(x + i, y - j, WHITE);

putpixel(x - i, y + j, WHITE);

if (p < 0) {

i++;

p += 2 \* i + 1;

}

else {

i++;

j--;

p += 2 \* (i - j) + 1;

}

}

}

int main() {

int gd=DETECT,gm;

initgraph(&gd,&gm,(char\*)"C:\\Turboc3\\BGI");

midpointCircle(200,300,50);

delay(5000);

closegraph();

return 0;

}

**5. Write a Program to draw a face of Teddy bear using midpoint algorithm only. (L5)**

#include <stdio.h>

#include <conio.h>

#include <graphics.h>

void midpointCircle(int x, int y, int radius);

void midpointCircle(int x, int y, int radius) {

int p = 1 - radius;

int i = 0;

int j = radius;

while (i <= j) {

putpixel(x + i, y + j, WHITE);

putpixel(x + j, y + i, WHITE);

putpixel(x - j, y + i, WHITE);

putpixel(x + j, y - i, WHITE);

putpixel(x - i, y - j, WHITE);

putpixel(x - j, y - i, WHITE);

putpixel(x + i, y - j, WHITE);

putpixel(x - i, y + j, WHITE);

if (p < 0) {

i++;

p += 2 \* i + 1;

} else {

i++;

j--;

p += 2 \* (i - j) + 1;

}

}

}

int main() {

int r,gd = DETECT, gm;

initgraph(&gd, &gm, "C:\\Turboc3\\BGI");

// Head

midpointCircle(300, 300, 100);

// Ears

midpointCircle(220, 200, 50); // Left ear

midpointCircle(380, 200, 50); // Right ear

// Eyes

midpointCircle(260, 280, 10); // Left eye

midpointCircle(340, 280, 10); // Right eye

// Nose

midpointCircle(300, 320, 15);

// Mouth

// You can add a simple mouth using another circle or a series of points/lines.

// Here we add a simple arc to represent the mouth using circles with decreasing radius

for ( r = 20; r > 10; r--) {

midpointCircle(300, 350, r);

}

delay(5000);

closegraph();

return 0;

}

**6. Write a Program to draw a car using Bresenham’s algorithm only. (L6)**

#include <graphics.h>

#include <stdio.h>

#include <conio.h>

#include <math.h>

// Bresenham's line algorithm

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

int dx = abs(x2 - x1);

int dy = abs(y2 - y1);

int sx = (x1 < x2) ? 1 : -1;

int sy = (y1 < y2) ? 1 : -1;

int err = dx - dy;

int e2;

while (x1 != x2 || y1 != y2) {

putpixel(x1, y1, WHITE);

e2 = 2 \* err;

if (e2 > -dy) {

err -= dy;

x1 += sx;

}

if (e2 < dx) {

err += dx;

y1 += sy;

}

}

}

// Midpoint circle algorithm

void cir(int x, int y, int r) {

int i = 0, j = r;

int p = 3 - 2 \* r;

while (i <= j) {

putpixel(x + i, y + j, WHITE);

putpixel(x - i, y + j, WHITE);

putpixel(x + i, y - j, WHITE);

putpixel(x - i, y - j, WHITE);

putpixel(x + j, y + i, WHITE);

putpixel(x - j, y + i, WHITE);

putpixel(x + j, y - i, WHITE);

putpixel(x - j, y - i, WHITE);

if (p < 0) {

p = p + 4 \* i + 6;

} else {

p = p + 4 \* (i - j) + 10;

j--;

}

i++;

}

}

int main() {

int gd = DETECT, gm;

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

// Car body using lines

bresnln(100, 200, 300, 200); // Top side

bresnln(100, 300, 300, 300); // Bottom side

bresnln(100, 200, 100, 300); // Left side

bresnln(300, 200, 300, 300); // Right side

// Car roof using lines

bresnln(150, 150, 250, 150); // Top side of roof

bresnln(150, 150, 100, 200); // Left slope

bresnln(250, 150, 300, 200); // Right slope

// Wheels using circles

cir(150, 320, 20); // Left wheel

cir(250, 320, 20); // Right wheel

getch();

closegraph();

return 0;

}

**7. Write a Program to implement Flood fill algorithm for a convex polygon. Draw polygon edges by DDA / Bresenham line algorithm. (L7)**

#include <stdio.h>

#include <conio.h>

#include <graphics.h>

#include <math.h>

int vertices[][2] = {{100, 100}, {130, 90}, {160, 100}, {160, 130}, {130, 140}, {100, 130}};

int n = sizeof(vertices) / sizeof(vertices[0]);

// DDA Line Algorithm

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

int k;

int dx = x2 - x1;

int dy = y2 - y1;

int steps;

float xIncrement, yIncrement;

float x = x1, y = y1;

if (abs(dx) > abs(dy)) {

steps = abs(dx);

} else {

steps = abs(dy);

}

xIncrement = dx / (float) steps;

yIncrement = dy / (float) steps;

putpixel(x, y, WHITE);

for (k = 0; k < steps; k++) {

x += xIncrement;

y += yIncrement;

putpixel(x, y, WHITE);

}

}

// Flood Fill Algorithm

void ffill(int x, int y, int fillcolor, int boundarycolor) {

if (getpixel(x, y) != boundarycolor && getpixel(x, y) != fillcolor) {

putpixel(x, y, fillcolor);

ffill(x + 1, y, fillcolor, boundarycolor);

ffill(x - 1, y, fillcolor, boundarycolor);

ffill(x, y + 1, fillcolor, boundarycolor);

ffill(x, y - 1, fillcolor, boundarycolor);

}

}

int main() {

int i,gd = DETECT, gm;

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

// Draw the polygon using DDA Line Algorithm

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

int x1 = vertices[i][0];

int y1 = vertices[i][1];

int x2 = vertices[(i + 1) % n][0];

int y2 = vertices[(i + 1) % n][1];

ddaLine(x1, y1, x2, y2);

}

// Perform Flood Fill inside the polygon

ffill(120, 120, YELLOW, WHITE); // A point inside the polygon

getch();

closegraph();

return 0;

}

**8. Write a Program to implement Boundary fill algorithm for a convex polygon. Draw polygon edges by DDA / Bresenham line algorithm. (L8)**

#include <stdio.h>

#include <conio.h>

#include <graphics.h>

#include <math.h>

int vertices[][2] = {{100, 100}, {130, 90}, {160, 100}, {160, 130}, {130, 140}, {100, 130}};

int n = sizeof(vertices) / sizeof(vertices[0]);

// DDA Line Algorithm

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

int k;

int dx = x2 - x1;

int dy = y2 - y1;

int steps;

float xIncrement, yIncrement;

float x = x1, y = y1;

if (abs(dx) > abs(dy)) {

steps = abs(dx);

} else {

steps = abs(dy);

}

xIncrement = dx / (float) steps;

yIncrement = dy / (float) steps;

putpixel(x, y, WHITE);

for (k = 0; k < steps; k++) {

x += xIncrement;

y += yIncrement;

putpixel(x, y, WHITE);

}

}

// Flood Fill Algorith;pm

void bfill(int x,int y, int newcolor,int boundarycolor)

{

int current=getpixel(x,y);

if(current!=newcolor&&current!=boundarycolor)

{

putpixel(x,y,newcolor);

bfill(x+1,y,newcolor,boundarycolor);

bfill(x,y+1,newcolor,boundarycolor);

bfill(x-1,y,newcolor,boundarycolor);

bfill(x,y-1,newcolor,boundarycolor);

}

}

int main() {

int i,gd = DETECT, gm;

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

// Draw the polygon using DDA Line Algorithm

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

int x1 = vertices[i][0];

int y1 = vertices[i][1];

int x2 = vertices[(i + 1) % n][0];

int y2 = vertices[(i + 1) % n][1];

ddaLine(x1, y1, x2, y2);

}

// Perform Flood Fill inside the polygon

bfill(120, 120, YELLOW, WHITE); // A point inside the polygon

getch();

closegraph();

return 0;

}

**9. Write a Program to implement Fence fill algorithm for a concave polygon. Draw polygon edges by DDA / Bresenham line algorithm. (L9)**

#include <stdio.h>

#include <conio.h>

#include <graphics.h>

#include <math.h>

int vertices[][2] = {{100, 100}, {150, 50}, {200, 100}, {175, 125}, {125, 125}};

int n = sizeof(vertices) / sizeof(vertices[0]);

// DDA Line Algorithm

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

int k;

int dx = x2 - x1;

int dy = y2 - y1;

int steps;

float xIncrement, yIncrement;

float x = x1, y = y1;

if (abs(dx) > abs(dy)) {

steps = abs(dx);

} else {

steps = abs(dy);

}

xIncrement = dx / (float) steps;

yIncrement = dy / (float) steps;

putpixel(x, y, WHITE);

for (k = 0; k < steps; k++) {

x += xIncrement;

y += yIncrement;

putpixel(x,y, WHITE);

}

}

// Fence Fill Algorithm

void fenceFill(int x, int y, int fillcolor, int boundarycolor) {

int xLeft = x - 1;

if (getpixel(x, y) != boundarycolor && getpixel(x, y) != fillcolor) {

while (getpixel(x, y) != boundarycolor) {

putpixel(x, y, fillcolor);

x++;

}

x = xLeft;

while (getpixel(x, y) != boundarycolor) {

putpixel(x, y, fillcolor);

x--;

}

x = xLeft;

y--;

while (getpixel(x, y) != boundarycolor) {

if (getpixel(x, y) != fillcolor) {

fenceFill(x, y, fillcolor, boundarycolor);

}

x++;

}

x = xLeft;

y += 2;

while (getpixel(x, y) != boundarycolor) {

if (getpixel(x, y) != fillcolor) {

fenceFill(x, y, fillcolor, boundarycolor);

}

x++;

}

}

}

int main() {

int i,gd = DETECT, gm;

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

// Draw the polygon using DDA Line Algorithm

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

int x1 = vertices[i][0];

int y1 = vertices[i][1];

int x2 = vertices[(i + 1) % n][0];

int y2 = vertices[(i + 1) % n][1];

ddaLine(x1, y1, x2, y2);

}

// Perform Fence Fill inside the polygon

fenceFill(150, 100, YELLOW, WHITE); // A point inside the polygon

getch();

closegraph();

return 0;

}

**10. Write a Program to implement Edge fill algorithm for a convex polygon. Draw polygon edges by DDA / Bresenham line algorithm. (L10)**

#include <stdio.h>

#include <conio.h>

#include <graphics.h>

#include <math.h>

int vertices[][2] = {{100, 100}, {130, 90}, {160, 100}, {160, 130}, {130, 140}, {100, 130}};

int n = sizeof(vertices) / sizeof(vertices[0]);

// DDA Line Algorithm

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

int k;

int dx = x2 - x1;

int dy = y2 - y1;

int steps;

float xIncrement, yIncrement;

float x = x1, y = y1;

if (abs(dx) > abs(dy)) {

steps = abs(dx);

} else {

steps = abs(dy);

}

xIncrement = dx / (float) steps;

yIncrement = dy / (float) steps;

putpixel(x, y, WHITE);

for (k = 0; k < steps; k++) {

x += xIncrement;

y += yIncrement;

putpixel(x, y, WHITE);

}

}

// Flood Fill Algorithm

void efill(int x, int y, int fillcolor, int boundarycolor) {

if (getpixel(x, y) != boundarycolor && getpixel(x, y) != fillcolor) {

putpixel(x, y, fillcolor);

efill(x + 1, y, fillcolor, boundarycolor);

efill(x - 1, y, fillcolor, boundarycolor);

efill(x, y + 1, fillcolor, boundarycolor);

efill(x, y - 1, fillcolor, boundarycolor);

}

}

int main() {

int i,gd = DETECT, gm;

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

// Draw the polygon using DDA Line Algorithm

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

int x1 = vertices[i][0];

int y1 = vertices[i][1];

int x2 = vertices[(i + 1) % n][0];

int y2 = vertices[(i + 1) % n][1];

ddaLine(x1, y1, x2, y2);

}

// Perform Flood Fill inside the polygon

efill(120, 120, YELLOW, WHITE); // A point inside the polygon

getch();

closegraph();

return 0;

}

**11. Write a Program to implement Scan line fill algorithm for a concave polygon. Draw polygon edges by DDA / Bresenham line algorithm. (L11)**

#include <stdio.h>

#include <conio.h>

#include <graphics.h>

#include <math.h>

int vertices[][2] = {{100, 100}, {150, 50}, {200, 100}, {175, 125}, {125, 125}};

int n = sizeof(vertices) / sizeof(vertices[0]);

// DDA Line Algorithm

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

int k;

int dx = x2 - x1;

int dy = y2 - y1;

int steps;

float xIncrement, yIncrement;

float x = x1, y = y1;

if (abs(dx) > abs(dy)) {

steps = abs(dx);

} else {

steps = abs(dy);

}

xIncrement = dx / (float) steps;

yIncrement = dy / (float) steps;

putpixel(x, y, WHITE);

for (k = 0; k < steps; k++) {

x += xIncrement;

y += yIncrement;

putpixel(x,y, WHITE);

}

}

//scanline function

void scanLineFill(int sides, int \*points, int fill\_color) {

int i, j, x, y, count, temp, xi, x1, y1, x2, y2;

for (y = 0; y < getmaxy(); y++) {

count = 0;

for (i = 0, j = sides \* 2 - 2; i < sides \* 2; j = i, i += 2) {

y1 = points[i + 1], y2 = points[j + 1];

x1 = points[i], x2 = points[j];

if ((y1 <= y && y2 > y) || (y2 <= y && y1 > y)) {

xi = (int)((float)(y - y1) / (y2 - y1) \* (x2 - x1) + x1);

if (count == 0) {

count++;

x = xi;

} else {

count = 0;

if (x > xi) {

temp = x;

x = xi;

xi = temp;

}

for (; x <= xi; x++) {

putpixel(x, y, fill\_color);

}

}

}

}

}

}

int main() {

int i,gd = DETECT, gm;

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

// Draw the polygon using DDA Line Algorithm

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

int x1 = vertices[i][0];

int y1 = vertices[i][1];

int x2 = vertices[(i + 1) % n][0];

int y2 = vertices[(i + 1) % n][1];

ddaLine(x1, y1, x2, y2);

}

// Perform Fence Fill inside the polygon

scanLineFill(n, vertices, YELLOW); // A point inside the polygon

getch();

closegraph();

return 0;

}

**12. Write a Program to implement 2D Scaling and rotation of a triangle.**(L12)

#include <graphics.h>

#include <stdio.h>

#include <math.h>

// Function to draw a triangle

void drawTriangle(int x1, int y1, int x2, int y2, int x3, int y3) {

line(x1, y1, x2, y2);

line(x2, y2, x3, y3);

line(x3, y3, x1, y1);

}

// Function to perform 2D scaling on a triangle

void scaleTriangle(int \*x1, int \*y1, int \*x2, int \*y2, int \*x3, int \*y3, float sx, float sy) {

\*x1 = \*x1 \* sx;

\*y1 = \*y1 \* sy;

\*x2 = \*x2 \* sx;

\*y2 = \*y2 \* sy;

\*x3 = \*x3 \* sx;

\*y3 = \*y3 \* sy;

}

// Function to perform 2D rotation on a triangle

void rotateTriangle(int \*x1, int \*y1, int \*x2, int \*y2, int \*x3, int \*y3, float angle) {

float rad = angle \* (M\_PI / 180.0);

float cosTheta = cos(rad);

float sinTheta = sin(rad);

int tempX1 = \*x1, tempY1 = \*y1;

int tempX2 = \*x2, tempY2 = \*y2;

int tempX3 = \*x3, tempY3 = \*y3;

\*x1 =(tempX1 \* cosTheta - tempY1 \* sinTheta);

\*y1 = (tempX1 \* sinTheta + tempY1 \* cosTheta);

\*x2 =(tempX2 \* cosTheta - tempY2 \* sinTheta);

\*y2 = (tempX2 \* sinTheta + tempY2 \* cosTheta);

\*x3 = (tempX3 \* cosTheta - tempY3 \* sinTheta);

\*y3 = (tempX3 \* sinTheta + tempY3 \* cosTheta);

}

int main() {

int choice,gd = DETECT, gm, x1 = 100, y1 = 100,x2 = 150, y2 = 50,x3 = 200, y3 = 100;

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

// Draw the original triangle

setcolor(WHITE);

drawTriangle(x1, y1, x2, y2, x3, y3);

printf("Choose operation:\n1. Scaling\n2. Rotation\nEnter choice: ");

scanf("%d", &choice);

// Perform selected operation

switch (choice) {

case 1: {

// Scaling factors

float sx = 1.5, sy = 1.5;

scaleTriangle(&x1, &y1, &x2, &y2, &x3, &y3, sx, sy);

break;

}

case 2: {

// Rotation angle

float angle = 45;

rotateTriangle(&x1, &y1, &x2, &y2, &x3, &y3, angle);

break;

}

default:

printf("Invalid choice.\n");

closegraph();

return 0;

}

// Draw the transformed triangle

setcolor(YELLOW);

drawTriangle(x1, y1, x2, y2, x3, y3);

getch();

closegraph();

return 0;

}

**13. Write a Program to implement 2D Scaling and translation of a triangle. (L13)**

#include <graphics.h>

#include <stdio.h>

#include <math.h>

// Function to draw a triangle

void drawTriangle(int x1, int y1, int x2, int y2, int x3, int y3) {

line(x1, y1, x2, y2);

line(x2, y2, x3, y3);

line(x3, y3, x1, y1);

}

// Function to perform 2D scaling on a triangle

void scaleTriangle(int \*x1, int \*y1, int \*x2, int \*y2, int \*x3, int \*y3, float sx, float sy) {

\*x1 = \*x1 \* sx;

\*y1 = \*y1 \* sy;

\*x2 = \*x2 \* sx;

\*y2 = \*y2 \* sy;

\*x3 = \*x3 \* sx;

\*y3 = \*y3 \* sy;

}

// Function to perform 2D translation on a triangle

void translateTriangle(int \*x1, int \*y1, int \*x2, int \*y2, int \*x3, int \*y3, int tx, int ty) {

\*x1 += tx;

\*y1 += ty;

\*x2 += tx;

\*y2 += ty;

\*x3 += tx;

\*y3 += ty;

}

int main() {

int choice, gd = DETECT, gm, x1 = 100, y1 = 100, x2 = 150, y2 = 50, x3 = 200, y3 = 100;

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

// Draw the original triangle

setcolor(WHITE);

drawTriangle(x1, y1, x2, y2, x3, y3);

printf("Choose operation:\n1. Scaling\n2. Translation\nEnter choice: ");

scanf("%d", &choice);

// Perform selected operation

switch (choice) {

case 1: {

// Scaling factors

float sx = 1.5, sy = 1.5;

scaleTriangle(&x1, &y1, &x2, &y2, &x3, &y3, sx, sy);

break;

}

case 2: {

// Translation distances

int tx = 50, ty = 50;

translateTriangle(&x1, &y1, &x2, &y2, &x3, &y3, tx, ty);

break;

}

default:

printf("Invalid choice.\n");

closegraph();

return 0;

}

// Draw the transformed triangle

setcolor(YELLOW);

drawTriangle(x1, y1, x2, y2, x3, y3);

getch();

closegraph();

return 0;

}

**14. Write a Program to implement 2D rotation and translation of a triangle. (L14)**

#include <graphics.h>

#include <stdio.h>

#include <math.h>

// Function to draw a triangle

void drawTriangle(int x1, int y1, int x2, int y2, int x3, int y3) {

line(x1, y1, x2, y2);

line(x2, y2, x3, y3);

line(x3, y3, x1, y1);

}

// Function to perform 2D rotation on a point

void rotatePoint(int \*x, int \*y, float angle) {

float rad = angle \* (M\_PI / 180.0);

float cosTheta = cos(rad);

float sinTheta = sin(rad);

int tempX = \*x, tempY = \*y;

\*x = (tempX \* cosTheta - tempY \* sinTheta);

\*y = (tempX \* sinTheta + tempY \* cosTheta);

}

// Function to perform 2D translation on a point

void translatePoint(int \*x, int \*y, int tx, int ty) {

\*x += tx;

\*y += ty;

}

int main() {

int choice, gd = DETECT, gm;

int x1 = 100, y1 = 100, x2 = 200, y2 = 100, x3 = 150, y3 = 50; // Initial coordinates of the triangle

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

// Draw the original triangle

setcolor(WHITE);

drawTriangle(x1, y1, x2, y2, x3, y3);

// Prompt user for operation choice

printf("Choose operation:\n1. Rotation\n2. Translation\nEnter choice: ");

scanf("%d", &choice);

// Perform selected operation

switch (choice) {

case 1: {

// Rotation angle

float angle;

printf("Enter rotation angle (in degrees): ");

scanf("%f", &angle);

// Rotate each vertex of the triangle

rotatePoint(&x1, &y1, angle);

rotatePoint(&x2, &y2, angle);

rotatePoint(&x3, &y3, angle);

break;

}

case 2: {

// Translation distances

int tx, ty;

printf("Enter translation distances along x and y axes: ");

scanf("%d %d", &tx, &ty);

// Translate each vertex of the triangle

translatePoint(&x1, &y1, tx, ty);

translatePoint(&x2, &y2, tx, ty);

translatePoint(&x3, &y3, tx, ty);

break;

}

default:

printf("Invalid choice.\n");

closegraph();

return 0;

}

// Draw the transformed triangle

setcolor(YELLOW);

drawTriangle(x1, y1, x2, y2, x3, y3);

getch();

closegraph();

return 0;

}

**15. Write a C program to show that R(θ1) . R(θ2) = R(θ1 + θ2)**

**16. Write a C program to show that R(θ1) . R(θ2) = R(θ2) . R(θ1) ‘**

**17. Write a C program to show that two successive translations are additive in nature. (L17)**

#include <stdio.h>

// Function to perform translation by dx along x-axis and dy along y-axis

void translate(float \*x, float \*y, float dx, float dy) {

\*x += dx;

\*y += dy;

}

int main() {

float x, y;

float dx1, dy1,dx2, dy2;

float dx\_sum,dy\_sum,newX ,newY;

// Initial point

printf("Enter the initial coordinates (x y): ");

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

// Translation distances for the first translation

printf("Enter the translation distances for the first translation (dx1 dy1): ");

scanf("%f %f", &dx1, &dy1);

// Translation distances for the second translation

printf("Enter the translation distances for the second translation (dx2 dy2): ");

scanf("%f %f", &dx2, &dy2);

// Perform first translation

translate(&x, &y, dx1, dy1);

// Perform second translation

translate(&x, &y, dx2, dy2);

// Calculate the sum of translation distances

dx\_sum = dx1 + dx2;

dy\_sum = dy1 + dy2;

// Perform a single translation by the sum of translation distances

newX = x + dx\_sum;

newY = y + dy\_sum;

// Display the results

printf("Result of two successive translations being additive:\n");

printf("Final coordinates after two successive translations: (%.2f, %.2f)\n", x, y);

printf("Final coordinates after translations again by sum of distances: (%.2f, %.2f)\n", newX, newY);

return 0;

}

**18. Write a C program to show that two successive rotations are commutative in nature. (l18)**

#include <stdio.h>

#include <math.h>

// Function to perform rotation by angle theta

void rotate(float \*x, float \*y, float theta) {

float rad = theta \* (M\_PI / 180.0); // Convert degrees to radians

float cosTheta = cos(rad);

float sinTheta = sin(rad);

float newX = (\*x) \* cosTheta - (\*y) \* sinTheta;

float newY = (\*x) \* sinTheta + (\*y) \* cosTheta;

// Update the coordinates

\*x = newX;

\*y = newY;

}

int main() {

float x, y;

float newX1 = x, newY1 = y;

float newX2 = x, newY2 = y;

float theta1, theta2;

// Initial point

printf("Enter the initial coordinates (x y): ");

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

printf("Enter the rotation angles (theta1 theta2 in degrees): ");

scanf("%f %f", &theta1, &theta2);

// First order of rotations: R(theta1) . R(theta2)

rotate(&newX1, &newY1, theta1);

rotate(&newX1, &newY1, theta2);

// Second order of rotations: R(theta2) . R(theta1)

rotate(&newX2, &newY2, theta2);

rotate(&newX2, &newY2, theta1);

// Display the results

printf("Result of two successive rotations being commutative:\n");

printf("Final coordinates after first order of rotations: (%.2f, %.2f)\n", newX1, newY1);

printf("Final coordinates after second order of rotations: (%.2f, %.2f)\n", newX2, newY2);

return 0;

}

**19. Write a C program to show that two successive translations are commutative in nature.**(L19)

#include <stdio.h>

// Function to perform translation by dx along x-axis and dy along y-axis

void translate(float \*x, float \*y, float dx, float dy) {

\*x += dx;

\*y += dy;

}

int main() {

float x, y,dx1, dy1,dx2, dy2;

float x1 = x, y1 = y;

float x2 = x, y2 = y;

// Initial point

printf("Enter the initial coordinates (x y): ");

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

// Translation distances for the first translation

printf("Enter the translation distances for the first translation (dx1 dy1): ");

scanf("%f %f", &dx1, &dy1);

// Translation distances for the second translation

printf("Enter the translation distances for the second translation (dx2 dy2): ");

scanf("%f %f", &dx2, &dy2);

// Perform two translations in the order: T(dx1, dy1) followed by T(dx2, dy2)

translate(&x1, &y1, dx1, dy1);

translate(&x1, &y1, dx2, dy2);

// Perform two translations in the order: T(dx2, dy2) followed by T(dx1, dy1)

translate(&x2, &y2, dx2, dy2);

translate(&x2, &y2, dx1, dy1);

// Display the results

printf("Result of two successive translations being commutative:\n");

printf("Final coordinates after T(dx1, dy1) followed by T(dx2, dy2): (%.2f, %.2f)\n", x1, y1);

printf("Final coordinates after T(dx2, dy2) followed by T(dx1, dy1): (%.2f, %.2f)\n", x2, y2);

return 0;

}

**20. Write a C Program to show that Reflection about a line Y=X is equivalent to reflection relative to X-axis followed by anticlockwise rotation of 900. (l20)**

#include <graphics.h>

#include <stdio.h>

#include <math.h>

// Function to perform reflection about the X-axis

void reflectX(int \*x, int \*y) {

\*y = -\*y;

}

// Function to perform 90-degree anticlockwise rotation

void rotate90(int \*x, int \*y) {

int temp = \*x;

\*x = -\*y;

\*y = temp;

}

// Function to perform reflection about the line Y = X

void reflectYX(int \*x, int \*y) {

int temp = \*x;

\*x = \*y;

\*y = temp;

}

// Function to draw a triangle

void drawTriangle(int x1, int y1, int x2, int y2, int x3, int y3, int color) {

setcolor(color);

line(x1 + getmaxx() / 2, getmaxy() / 2 - y1, x2 + getmaxx() / 2, getmaxy() / 2 - y2);

line(x2 + getmaxx() / 2, getmaxy() / 2 - y2, x3 + getmaxx() / 2, getmaxy() / 2 - y3);

line(x3 + getmaxx() / 2, getmaxy() / 2 - y3, x1 + getmaxx() / 2, getmaxy() / 2 - y1);

}

int main() {

int gd = DETECT, gm;

int orig\_x1, orig\_y1 ,orig\_x2 ,orig\_y2 ,orig\_x3 , orig\_y3 ;

int x1 = 50, y1 = 50;

int x2 = 100, y2 = 50;

int x3 = 75, y3 = 100;

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

// Draw the original triangle in white

drawTriangle(x1, y1, x2, y2, x3, y3, WHITE);

// Reflect about the X-axis followed by 90-degree anticlockwise rotation

reflectX(&x1, &y1);

rotate90(&x1, &y1);

reflectX(&x2, &y2);

rotate90(&x2, &y2);

reflectX(&x3, &y3);

rotate90(&x3, &y3);

// Draw the transformed triangle in yellow

drawTriangle(x1, y1, x2, y2, x3, y3, YELLOW);

delay(2000);

// Reset to original coordinates

x1 = orig\_x1; y1 = orig\_y1;

x2 = orig\_x2; y2 = orig\_y2;

x3 = orig\_x3; y3 = orig\_y3;

// Reflect about the line Y = X

reflectYX(&x1, &y1);

reflectYX(&x2, &y2);

reflectYX(&x3, &y3);

// Draw the triangle reflected about Y = X in green

drawTriangle(x1, y1, x2, y2, x3, y3, GREEN);

delay(2000);

// Display instructions to the user

outtextxy(10, 10, "WHITE: Original Triangle");

outtextxy(10, 50, "GREEN: Reflection about line Y=X");

outtextxy(10, 30, "YELLOW: Reflection about X-axis followed by 90-degree rotation");

getch();

closegraph();

return 0;

}

**21. Write a Program to implement all type of reflections about X axis and about Y axis of a triangle.(l21)**

#include <graphics.h>

#include <stdio.h>

#include <math.h>

// Function to draw a triangle

void drawTriangle(int x1, int y1, int x2, int y2, int x3, int y3, int color) {

setcolor(color);

line(x1, y1, x2, y2);

line(x2, y2, x3, y3);

line(x3, y3, x1, y1);

}

// Function to perform reflection about the X-axis

void reflectX(int \*x, int \*y) {

\*y = -\*y;

}

// Function to perform reflection about the Y-axis

void reflectY(int \*x, int \*y) {

\*x = -\*x;

}

int main() {

int choice,cx,cy,gd = DETECT, gm;

int rx1, ry1, rx2, ry2, rx3, ry3;

int x1 = 100, y1 = 100;

int x2 = 150, y2 = 50;

int x3 = 200, y3 = 100;

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

// Center points for the four parts of the screen

cx = getmaxx() / 2;

cy = getmaxy() / 2;

// Draw the original triangle in the first quadrant (top-left)

drawTriangle(cx + x1, cy - y1, cx + x2, cy - y2, cx + x3, cy - y3, WHITE);

printf("Choose the type of reflection:\n");

printf("1. Reflection about X-axis\n");

printf("2. Reflection about Y-axis\n");

printf("Enter your choice: ");

scanf("%d", &choice);

// Perform the selected reflection

rx1 = x1, ry1 = y1, rx2 = x2, ry2 = y2, rx3 = x3, ry3 = y3;

switch (choice) {

case 1:

reflectX(&rx1, &ry1);

reflectX(&rx2, &ry2);

reflectX(&rx3, &ry3);

outtextxy(3 \* cx / 2, cy / 2, "Reflection about X-axis");

break;

case 2:

reflectY(&rx1, &ry1);

reflectY(&rx2, &ry2);

reflectY(&rx3, &ry3);

outtextxy(cx / 2, 3 \* cy / 2, "Reflection about Y-axis");

break;

default:

printf("Invalid choice.\n");

closegraph();

return 0;

}

// Draw the reflected triangle

setcolor(YELLOW);

drawTriangle(cx + rx1, cy - ry1, cx + rx2, cy - ry2, cx + rx3, cy - ry3, YELLOW);

// Draw the dividing lines

setcolor(WHITE);

line(cx, 0, cx, getmaxy());

line(0, cy, getmaxx(), cy);

getch();

closegraph();

return 0;

}

**22. Write a Program to implement all type of reflections about origin and about a line Y = X for a triangle.**

**23. Write a Program to implement X and Y shear transformation (l23)**

#include <graphics.h>

#include <stdio.h>

#include <math.h>

// Function to draw a triangle

void drawTriangle(int x1, int y1, int x2, int y2, int x3, int y3, int color, const char\* label) {

setcolor(color);

line(x1, y1, x2, y2);

line(x2, y2, x3, y3);

line(x3, y3, x1, y1);

outtextxy((x1 + x2 + x3) / 3, (y1 + y2 + y3) / 3, label);

}

// Function to perform X shear transformation

void shearX(int \*x1, int \*y1, int \*x2, int \*y2, int \*x3, int \*y3, float shx) {

\*x1 = \*x1 + shx \* (\*y1);

\*x2 = \*x2 + shx \* (\*y2);

\*x3 = \*x3 + shx \* (\*y3);

}

// Function to perform Y shear transformation

void shearY(int \*x1, int \*y1, int \*x2, int \*y2, int \*x3, int \*y3, float shy) {

\*y1 = \*y1 + shy \* (\*x1);

\*y2 = \*y2 + shy \* (\*x2);

\*y3 = \*y3 + shy \* (\*x3);

}

int main() {

int choice,gd = DETECT, gm,shx,shy;

int x1 = 100, y1 = 100;

int x2 = 150, y2 = 50;

int x3 = 200, y3 = 100;

int sx1, sy1, sx2 , sy2, sx3, sy3;

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

// Draw the original triangle

drawTriangle(x1, y1, x2, y2, x3, y3, WHITE, "Original Triangle");

printf("Choose the type of shear transformation:\n");

printf("1. X-Shear\n");

printf("2. Y-Shear\n");

printf("Enter your choice: ");

scanf("%d", &choice);

// Perform the selected shear transformation

sx1 = x1, sy1 = y1, sx2 = x2, sy2 = y2, sx3 = x3, sy3 = y3;

switch (choice) {

case 1: {

// X-Shear factor

shx = 1.5;

shearX(&sx1, &sy1, &sx2, &sy2, &sx3, &sy3, shx);

drawTriangle(sx1, sy1, sx2, sy2, sx3, sy3, YELLOW, "X-Shear");

break;

}

case 2: {

// Y-Shear factor

shy = 1.5;

shearY(&sx1, &sy1, &sx2, &sy2, &sx3, &sy3, shy);

drawTriangle(sx1, sy1, sx2, sy2, sx3, sy3, GREEN, "Y-Shear");

break;

}

default:

printf("Invalid choice.\n");

closegraph();

return 0;

}

getch();

closegraph();

return 0;

}

**24. Write a Program to implement rotation about arbitrary point. (l24)**

#include <graphics.h>

#include <stdio.h>

#include <math.h>

// Function to draw a triangle

void drawTriangle(int x1, int y1, int x2, int y2, int x3, int y3, int color, const char\* label) {

setcolor(color);

line(x1, y1, x2, y2);

line(x2, y2, x3, y3);

line(x3, y3, x1, y1);

outtextxy((x1 + x2 + x3) / 3, (y1 + y2 + y3) / 3, label);

}

// Function to perform rotation about an arbitrary point

void rotateAboutPoint(int \*x, int \*y, int xp, int yp, float angle) {

int newX,newY;

float rad = angle \* (M\_PI / 180.0);

float cosTheta = cos(rad);

float sinTheta = sin(rad);

int tempX = \*x, tempY = \*y;

// Translate point to origin

\*x = tempX - xp;

\*y = tempY - yp;

// Rotate point

newX = \*x \* cosTheta - \*y \* sinTheta;

newY = \*x \* sinTheta + \*y \* cosTheta;

// Translate point back

\*x = newX + xp;

\*y = newY + yp;

}

int main() {

int xp,yp,gd = DETECT, gm;

int x1 = 100, y1 = 100;

int x2 = 150, y2 = 50;

int x3 = 200, y3 = 100;

int rx1, ry1, rx2, ry2, rx3, ry3;

float angle;

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

// Draw the original triangle

drawTriangle(x1, y1, x2, y2, x3, y3, WHITE, "Original Triangle");

// Ask the user for the arbitrary point and the angle of rotation

printf("Enter the coordinates of the arbitrary point (xp, yp): ");

scanf("%d %d", &xp, &yp);

printf("Enter the angle of rotation (in degrees): ");

scanf("%f", &angle);

// Perform the rotation about the arbitrary point

rx1 = x1, ry1 = y1, rx2 = x2, ry2 = y2, rx3 = x3, ry3 = y3;

rotateAboutPoint(&rx1, &ry1, xp, yp, angle);

rotateAboutPoint(&rx2, &ry2, xp, yp, angle);

rotateAboutPoint(&rx3, &ry3, xp, yp, angle);

// Draw the rotated triangle

drawTriangle(rx1, ry1, rx2, ry2, rx3, ry3, YELLOW, "Rotated Triangle");

getch();

closegraph();

return 0;

}

**25. Write a Program to implement Cohen Sutherland line clipping algorithm. (l25)**

**TAKE USER INPUT**

#include<conio.h>

#include<stdio.h>

#include<graphics.h>

#include<math.h>

void main()

{

int a[4],b[4];

float m,xnew,ynew;

float xl=100,yl=100,xh=300,yh=300,xa=10,ya=200,xb=250,yb=150;

int gd = DETECT,gm;

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

setcolor(5);

line(xa,ya,xb,yb);

setcolor(12);

rectangle(xl,yl,xh,yh);

m = (yb-ya)/(xb-xa);

if(xa < xl)

a[3] = 1;

else a[3] = 0;

if(xa>xh)

a[2] = 1;

else a[2] = 0;

if(ya < yl)

a[1] = 1;

else a[1] = 0;

if (ya > yh)

a[0] = 1;

else a[0] = 0;

if(xb < xl)

b[3] = 1;

else b[3] = 0;

if(xb>xh)

b[2] = 1;

else b[2] = 0;

if(yb < yl)

b[1] = 1;

else b[1] = 0;

if (yb > yh)

b[0] = 1;

else b[0] = 0;

printf("press a key to continue");

getch();

if(a[0] == 0 && a[1] == 0 && a[2] == 0 && a[3] == 0 && b[0] == 0 && b[1] == 0 && b[2] == 0 && b[3] == 0 )

{

printf("no clipping");

line(xa,ya,xb,yb);

}

else if(a[0]&&b[0] || a[1]&&b[1] || a[2]&&b[2] || a[3]&&b[3])

{

clrscr();

printf("line discarded");

rectangle(xl,yl,xh,yh);

}

else

{

if(a[3] == 1 && b[3]==0)

{

ynew = (m \* (xl-xa)) + ya;

setcolor(12);

rectangle(xl,yl,xh,yh);

setcolor(0);

line(xa,ya,xb,yb);

setcolor(15);

line(xl,ynew,xb,yb);

}

else if(a[2] == 1 && b[2] == 0)

{

ynew = (m \* (xh-xa)) + ya;

setcolor(12);

rectangle(xl,yl,xh,yh);

setcolor(0);

line(xa,ya,xb,yb);

setcolor(15);

line(xl,ynew,xb,yb);

}

else if(a[1] == 1 && b[1] == 0)

{

xnew = xa + (yl-ya)/m;

setcolor(0);

line(xa,ya,xb,yb);

setcolor(15);

line(xnew,yh,xb,yb);

}

else if(a[0] == 1 && b[0] == 0)

{

xnew = xa + (yh-ya)/m;

setcolor(0);

line(xa,ya,xb,yb);

setcolor(15);

line(xnew,yh,xb,yb);

}

}

getch();

closegraph();

}

**26. Write a Program to implement midpoint line clipping algorithm.**

**TAKE USER INPUT**

#include <graphics.h>

#include <stdio.h>

#include <stdlib.h>

#define LEFT 1

#define RIGHT 2

#define BOTTOM 4

#define TOP 8

int xmin, ymin, xmax, ymax;

int computeCode(int x, int y) {

int code = 0;

if (x < xmin) code |= LEFT;

if (x > xmax) code |= RIGHT;

if (y < ymin) code |= BOTTOM;

if (y > ymax) code |= TOP;

return code;

}

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

int code1 = computeCode(x1, y1);

int code2 = computeCode(x2, y2);

int accept = 0;

while (1) {

if (!(code1 | code2)) {

accept = 1;

break;

} else if (code1 & code2) {

break;

} else {

int codeOut;

int x, y;

if (code1 != 0) {

codeOut = code1;

} else {

codeOut = code2;

}

if (codeOut & TOP) {

x = x1 + (x2 - x1) \* (ymax - y1) / (y2 - y1);

y = ymax;

} else if (codeOut & BOTTOM) {

x = x1 + (x2 - x1) \* (ymin - y1) / (y2 - y1);

y = ymin;

} else if (codeOut & RIGHT) {

y = y1 + (y2 - y1) \* (xmax - x1) / (x2 - x1);

x = xmax;

} else if (codeOut & LEFT) {

y = y1 + (y2 - y1) \* (xmin - x1) / (x2 - x1);

x = xmin;

}

if (codeOut == code1) {

x1 = x;

y1 = y;

code1 = computeCode(x1, y1);

} else {

x2 = x;

y2 = y;

code2 = computeCode(x2, y2);

}

}

}

if (accept) {

setcolor(GREEN);

line(x1, y1, x2, y2);

}

}

void drawRectangle(int xmin, int ymin, int xmax, int ymax) {

setcolor(WHITE);

rectangle(xmin, ymin, xmax, ymax);

}

int main() {

int gd = DETECT, gm;

int x1 = 50, y1 = 150, x2 = 350, y2 = 250;

int xmin = 100; ymin = 100; xmax = 300; ymax = 300;

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

// Drawing the clipping rectangle

drawRectangle(xmin, ymin, xmax, ymax);

// Draw original line in red

setcolor(RED);

line(x1, y1, x2, y2);

// Perform midpoint line clipping

midpointLineClipping(x1, y1, x2, y2);

getch();

closegraph();

return 0;

}

**27. Write a Program to implement Sutherland-Hodgeman Polygon clipping algorithm. (l27)**

**TAKE USER INPUT**

#include<stdio.h>

#include<graphics.h>

#include<conio.h>

#include<stdlib.h>

int main()

{

int gd,gm,n,\*x,i,k=0;

//window coordinates int wx1=220,wy1=140,wx2=420,wy2=140,wx3=420,wy3=340,wx4=220,wy4=340;

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

detectgraph(&gd,&gm);

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

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]; //assigning the coordinates of first vertex to last additional vertex for drawpoly method.

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();

return 0;

}

**28. Write a Program to implement Generalized Polygon clipping algorithm. (l27)**

#include<stdio.h>

#include<graphics.h>

#include<conio.h>

#include<stdlib.h>

int main()

{

int gd,gm,n,\*x,i,k=0;

//window coordinates int wx1=220,wy1=140,wx2=420,wy2=140,wx3=420,wy3=340,wx4=220,wy4=340;

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

detectgraph(&gd,&gm);

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

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]; //assigning the coordinates of first vertex to last additional vertex for drawpoly method.

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();

return 0;

}

**29. Write a Program to draw a Koch curve upto ‘n’ iterations (l29)**

#include <graphics.h>

#include <stdio.h>

#include <math.h>

// Function to draw the Koch Curve

void kochCurve(int x1, int y1, int x2, int y2, int n) {

int dx,dy,x3,y3,x5,y5,x4,y4;

if (n == 0) {

line(x1, y1, x2, y2);

return;

}

// Calculate points

dx = x2 - x1;

dy = y2 - y1;

x3 = x1 + dx / 3;

y3 = y1 + dy / 3;

x5 = x1 + 2 \* dx / 3;

y5 = y1 + 2 \* dy / 3;

x4 = (int)(0.5 \* (x1 + x2) + sqrt(3) \* (y1 - y2) / 6);

y4 = (int)(0.5 \* (y1 + y2) + sqrt(3) \* (x2 - x1) / 6);

// Recursively draw the four segments

kochCurve(x1, y1, x3, y3, n - 1);

kochCurve(x3, y3, x4, y4, n - 1);

kochCurve(x4, y4, x5, y5, n - 1);

kochCurve(x5, y5, x2, y2, n - 1);

}

int main() {

int gd = DETECT, gm, n;

int x1 = 50, y1 = 200, x2 = 450, y2 = 200;

// Initialize graphics mode

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

// Ask user for the number of iterations

printf("Enter the number of iterations: ");

scanf("%d", &n);

setcolor(WHITE);

kochCurve(x1, y1, x2, y2, n);

// Wait for user input before closing

getch();

closegraph();

return 0;

}

**30. Write a Program to draw a Hilbert curve upto ‘n’ iterations. (l30)**

#include <graphics.h>

#include <stdio.h>

#include <math.h>

#include <conio.h>

#include <dos.h>

// Function to move to the next point based on the direction

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);

}

// Recursive function to draw the Hilbert Curve

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);

delay(100);

hilbert(r, d, l, u, i, h, x, y);

move(d, h, x, y);

delay(100);

hilbert(r, d, l, u, i, h, x, y);

move(l, h, x, y);

delay(100);

hilbert(u, l, d, r, i, h, x, y);

}

}

int main() {

int n, x1, y1;

int x0 = 50, y0 = 150, x, y, h = 10;

int r = 2, d = 3, l = 4, u = 1;

printf("Give the value of n: ");

scanf("%d", &n);

x = x0;

y = y0;

int gd = DETECT, gm;

initgraph(&gd, &gm, "C:\\Turboc3\\bgi");

moveto(x, y);

hilbert(r, d, l, u, n, h, &x, &y);

delay(100);

getch();

closegraph();

return 0;

}

**31. Write a Program to draw a Bezier curve upto ‘n’ iterations using midpoint method.(l31,l31a)**

#include <graphics.h>

#include <stdio.h>

// Structure to represent a point

typedef struct {

int x;

int y;

} Point;

// Function to calculate a point on the Bezier curve

Point calculateBezierPoint(Point \*controlPoints, int n, double t) {

int i;

Point \*newControlPoints = (Point \*)malloc((n - 1) \* sizeof(Point));

if (n == 1) {

return controlPoints[0];

}

for ( i = 0; i < n - 1; i++) {

newControlPoints[i].x = (int)((1 - t) \* controlPoints[i].x + t \* controlPoints[i + 1].x);

newControlPoints[i].y = (int)((1 - t) \* controlPoints[i].y + t \* controlPoints[i + 1].y);

}

return calculateBezierPoint(newControlPoints, n - 1, t);

}

// Function to draw the Bezier curve

void drawBezierCurve(Point \*controlPoints, int n, int numSegments) {

double t;

double dt = 1.0 / numSegments;

Point previousPoint = controlPoints[0];

for ( t = dt; t <= 1.0; t += dt) {

Point currentPoint = calculateBezierPoint(controlPoints, n, t);

line(previousPoint.x, previousPoint.y, currentPoint.x, currentPoint.y);

previousPoint = currentPoint;

}

}

int main() {

int i,gd = DETECT, gm;

int numSegments, numControlPoints;

Point \*controlPoints;

controlPoints = (Point \*)malloc(numControlPoints \* sizeof(Point));

printf("Enter the number of control points: ");

scanf("%d", &numControlPoints);

if (controlPoints == NULL) {

printf("Memory allocation failed.");

return 1;

}

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

printf("Enter x-coordinate of control point %d: ", i + 1);

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

printf("Enter y-coordinate of control point %d: ", i + 1);

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

}

printf("Enter the number of segments: ");

scanf("%d", &numSegments);

initgraph(&gd, &gm, "C:\\Turboc3\\bgi");

drawBezierCurve(controlPoints, numControlPoints, numSegments);

getch();

closegraph();

free(controlPoints);

return 0;

}

**32. Write a Program to draw a coastline using Fractal line upto ‘n’ iterations.**

**33. Write a Program to draw a mountain using Fractal surface upto ‘n’ iterations.**

**34. Write a program to achieve various animations without using any readymade line or**

**circle function. Use DDA or Bresenham algorithm for implementation of line and circle. ( for sample animations refer attached sheet). (l34)**

#include <stdio.h>

#include <conio.h>

#include <graphics.h>

#include <math.h>

#include <dos.h>

void drawDDALine(int x1, int y1, int x2, int y2, int color);

void drawBresenhamCircle(int xc, int yc, int r, int color);

void clearScreen(int color);

void drawDDALine(int x1, int y1, int x2, int y2, int color) {

int i;

int dx = x2 - x1;

int dy = y2 - y1;

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

float xInc = dx / (float)steps;

float yInc = dy / (float)steps;

float x = x1;

float y = y1;

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

putpixel(x,y, color);

x += xInc;

y += yInc;

}

}

void drawBresenhamCircle(int xc, int yc, int r, int color) {

int x = 0, y = r;

int p = 3 - 2 \* r;

while (x <= y) {

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

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

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

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

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

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

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

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

if (p < 0) {

p += 4 \* x + 6;

} else {

p += 4 \* (x - y) + 10;

y--;

}

x++;

}

}

void clearScreen(int color) {

int i,j;

int maxX = getmaxx();

int maxY = getmaxy();

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

for ( j = 0; j <= maxY; j++) {

putpixel(i, j, color);

}

}

}

int main() {

int i,gd = DETECT, gm;

int x1 = 100, y1 = 100, x2 = 200, y2 = 200;

int xc = 300, yc = 200, r = 50;

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

for ( i = 0; i < 100; i++) {

clearScreen(BLACK);

drawDDALine(x1 + i, y1, x2 + i, y2, WHITE);

drawBresenhamCircle(xc + i, yc, r, WHITE);

drawBresenhamCircle(xc - i, yc, r, WHITE);

delay(50);

}

getch();

closegraph();

return 0;

}

10)Write a program to implement of different 2D transformations:

a) Translate a polygon by 2 units.

b)Rotate a polygon about 90 degree.

#include <stdio.h>

#include <graphics.h>

#include <math.h>

int xt = 0, yt = 0;

int points[] = {300,100, 400,300, 200,300}; // Declare points array for triangle

// Drawing polygon (triangle in this case)

void drawPolygon(int sides, int \*points) {

int i;

for (i = 0; i < sides \* 2; i += 2) {

line(points[i], points[i + 1], points[(i + 2) % (sides \* 2)], points[(i + 3) % (sides \* 2)]);

}

}

// Rotate point function

void rotatePoint(int \*x, int \*y, int cx, int cy, float theta) {

float radian = theta \* (M\_PI / 180);

int new\_x = cx + (\*x - cx) \* cos(radian) - (\*y - cy) \* sin(radian);

int new\_y = cy + (\*x - cx) \* sin(radian) + (\*y - cy) \* cos(radian);

\*x = new\_x;

\*y = new\_y;

}

int main() {

int gd = DETECT, gm, i;

int choice;

float angle;

initgraph(&gd, &gm, "C:\\Turboc3\\BGI");

// Draw the object as it is

drawPolygon(3, points);

delay(2000);

printf("Choose an operation:\n");

printf("1. Translation\n");

printf("2. Rotation\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter translation values (x y): ");

scanf("%d %d", &xt, &yt);

break;

case 2:

printf("Enter rotation angle (in degrees): ");

scanf("%f", &angle);

angle \*= -1; // Reverse angle for clockwise rotation

break;

default:

printf("Invalid choice\n");

return 0;

}

cleardevice();

// Apply the chosen transformation to the object and redraw it

switch (choice) {

case 1:

for (i = 0; i < 6; i += 2) {

points[i] += xt;

points[i + 1] += yt;

}

drawPolygon(3, points);

break;

case 2: {

int cx = (points[0] + points[2] + points[4]) / 3;

int cy = (points[1] + points[3] + points[5]) / 3;

for (i = 0; i < 6; i += 2) {

rotatePoint(&points[i], &points[i + 1], cx, cy, angle);

}

drawPolygon(3, points);

break;

}

}

getch();

closegraph();

return 0;

}

11) Write a program to implement of different 2D transformations:

a)Scale a polygon by 3 units

b)Reflect a polygon about x.

#include<graphics.h>

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

int points[6] = {100, 100, 200, 100, 150, 50};

void drawPolygon(int \*points, int sides, int color) {

int i,x1,x2,y1,y2;

setcolor(color);

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

x1 = points[2 \* i];

y1 = points[2 \* i + 1];

x2 = points[2 \* ((i + 1) % sides)];

y2 = points[2 \* ((i + 1) % sides) + 1];

line(x1, y1, x2, y2);

}

}

void scalePolygon(int \*points, int sides, float scale) {

int i;

for (i = 0; i < sides \* 2; i++) {

points[i] = points[i] \* scale;

}

}

void reflectPolygon(int \*points, int sides, char axis) {

int i;

int max\_x = getmaxx() / 2;

int max\_y = getmaxy() / 2;

// Translate to center

for (i = 0; i < sides \* 2; i += 2) {

points[i] -= max\_x;

}

for (i = 1; i < sides \* 2; i += 2) {

points[i] -= max\_y;

}

if (axis == 'x' || axis == 'X') {

for (i = 1; i < sides \* 2; i += 2) {

points[i] = -points[i];

}

} else if (axis == 'y' || axis == 'Y') {

for (i = 0; i < sides \* 2; i += 2) {

points[i] = -points[i];

}

}

// Translate back to positive coordinates

for (i = 0; i < sides \* 2; i += 2) {

points[i] += max\_x;

}

for (i = 1; i < sides \* 2; i += 2) {

points[i] += max\_y;

}

}

int main() {

int i,gd = DETECT, gm;

char axis;

int sides = 3;

int scaledPoints[6];

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

// Draw original polygon

drawPolygon(points, sides, WHITE);

getch();

cleardevice();

// Scale polygon by 3 units

for ( i = 0; i < 6; i++) {

scaledPoints[i] = points[i];

}

scalePolygon(scaledPoints, sides, 3);

// Draw scaled polygon

drawPolygon(scaledPoints, sides, GREEN);

getch();

cleardevice();

// Reflect polygon about x or y axis

printf("Enter the axis of reflection (x or y): ");

scanf(" %c", &axis);

reflectPolygon(scaledPoints, sides, axis);

// Draw reflected polygon

drawPolygon(scaledPoints, sides, RED);

getch();

closegraph();

return 0;

}

12)Write a program to implement of different 3D transformations:

a) Translate a polygon by 2 units.

b)Rotate a polygon about 90 degree.

13) Write a program to implement of different 2D transformations:

a)Scale a polygon by 3 units

b)Reflect a polygon about z.====

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

int maxx,maxy,midx,midy; //global variables

void axis()

{

getch();

cleardevice();

line(midx,0,midx,maxy); // horizontal line I.e x axis

line(0,midy,maxx,midy); //vertical line I.e y axis

}

void main()

{

int gd,gm,x,y,z,ang,x1,x2,y1,y2;

detectgraph(&gd,&gm);

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

setfillstyle(3,25);

maxx = getmaxx();

maxy = getmaxy();

midx=maxx/2;

midy=maxy/2;

outtextxy(100,100,"ORIGINAL OBJECT");

line(midx,0,midx,maxy);

line(0,midy,maxx,midy);

bar3d(midx+100,midy-20,midx+60,midy-90,20,5);

axis();

outtextxy(100,20,"SCALING");

printf("\nEnter the Scaling Factor: ");

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

bar3d(midx+100,midy-20,midx+60,midy-90,20,5);

bar3d(midx+(x\*100),midy-(y\*20),midx+(x\*60),midy-(y\*90),20\*z,5);

axis();

outtextxy(100,20,"REFLECTION");

printf("\n Enter the Reflection angle: ");

scanf("%d",&ang);

x1=100\*cos(ang\*3.14/180)-20\*sin(ang\*3.14/180);

y1=100\*sin(ang\*3.14/180)+20\*sin(ang\*3.14/180);

x2=60\*cos(ang\*3.14/180)-90\*sin(ang\*3.14/180);

y2=60\*sin(ang\*3.14/180)+90\*sin(ang\*3.14/180);

axis();

printf("\n After reflection about z-axis\n");

bar3d(midx+100,midy-20,midx+60,midy-90,20,5);

bar3d(midx+x1,midy-y1,midx+x2,midy-y2,20,5);

axis();

closegraph();

}