1. Write a Program to implement DDA Line drawing algorithm

#include <graphics.h>

#include <stdio.h>

#include <conio.h>

#include <math.h>

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

int dx = x1 - x0;

int dy = y1 - y0;

int steps, k;

float xIncrement, yIncrement, x = x0, y = y0;

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

steps = abs(dx);

else

steps = abs(dy);

xIncrement = dx / (float) steps;

yIncrement = dy / (float) steps;

putpixel(round(x), round(y), WHITE);

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

x += xIncrement;

y += yIncrement;

putpixel(abs(x), abs(y), WHITE);

delay(2);

}

}

int main() {

int gm, gd = DETECT;

int x0, y0, x1, y1;

// Initialize the graphics mode

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

printf("Enter the coordinates of the first point of the first line (x0, y0): ");

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

printf("Enter the coordinates of the second point of the first line (x1, y1): ");

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

drawLine(x0, y0, x1, y1);

getch();

closegraph(); // Close the graphics mode before exiting

return 0;

}

1. Write a Program to implement Bresenham’s Line drawing algorithm

#include <graphics.h>

#include <stdio.h>

#include <conio.h>

#include <math.h>

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

int dx = abs(x2 - x1);

int dy = abs(y2 - y1);

int slope\_x = x1 < x2 ? 1 : -1;

int slope\_y = y1 < y2 ? 1 : -1;

int x, y;

if (dy <= dx) {

int d = 2 \* dy - dx;

int y = y1;

for (x = x1; x != x2; x += slope\_x) {

putpixel(x, y, BLUE);

delay(1);

if (d > 0) {

y += slope\_y;

d -= 2 \* dx;

}

d += 2 \* dy;

}

} else {

int d = 2 \* dx - dy;

int x = x1;

for (y = y1; y != y2; y += slope\_y) {

putpixel(x, y, BLUE);

delay(1);

if (d > 0) {

x += slope\_x;

d -= 2 \* dy;

}

d += 2 \* dx;

}

}

}

int main() {

int gm, gd = DETECT;

int x0, y0, x1, y1;

// Initialize the graphics mode

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

printf("Enter the coordinates of the first point of the first line (x0, y0): ");

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

printf("Enter the coordinates of the second point of the first line (x1, y1): ");

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

drawLine(x0, y0, x1, y1);

getch();

closegraph(); // Close the graphics mode before exiting

return 0;

}

1. Write a Program to implement Bresenham’s Circle drawing algorithm

#include <stdio.h>

#include <conio.h>

#include <graphics.h>

#include <math.h>

// Function to draw a circle using Bresenham's Circle Drawing Algorithm

void brescir(int x, int y, int r, int color) {

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

while (i <= j) {

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

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

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

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

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

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

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

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

if (p < 0) {

i++;

p = p + 4 \* i + 6;

} else {

i++;

j--;

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

}

}

}

int main() {

int gd = DETECT, gm;

int x, y, r;

// Initialize the graphics mode

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

// Take user input for circles

printf("Enter the center coordinates of the first circle (x, y) and radius: ");

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

brescir(x, y, r, MAGENTA); //any color you want

getch();

closegraph(); // Close the graphics mode before exiting

return 0;

}

1. Write a Program to implement Mid-point Circle drawing algorithm.

#include <stdio.h>

#include <stdlib.h>

#include <graphics.h>

#include <conio.h>

// Function to draw a circle using Midpoint Circle Drawing Algorithm

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

int p = 1 - r;

int i = 0;

int j = r;

while (i <= j) {

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

delay(1);

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

delay(1);

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

delay(1);

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

delay(1);

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

delay(1);

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

delay(1);

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

delay(1);

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

delay(1);

if (p < 0) {

i++;

p = p + 2 \* i + 1;

} else {

i++;

j--;

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

}

}

}

int main() {

int gd = DETECT, gm;

int x, y, r;

// Initialize the graphics mode

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

// Take user input for circles

printf("Enter the center coordinates of the first circle (x, y) and radius: ");

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

midcir(x, y, r);

getch();

closegraph(); // Close the graphics mode before exiting

return 0;

}

1. Write a Program to draw a face of Teddy bear using midpoint algorithm only.

#include <graphics.h>

#include <conio.h>

#include <stdio.h>

void drawCircle(int xc, int yc, int r) {

int x = 0, y = r;

int p = 1 - r;

while (x <= y) {

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

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

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

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

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

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

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

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

delay(1);

if (p < 0) {

p = p + 2 \* x + 3;

} else {

p = p + 2 \* (x - y) + 5;

y--;

}

x++;

}

}

void drawTeddyFace() {

// Head

drawCircle(250, 250, 100);

// Ears

drawCircle(180, 180, 50);

drawCircle(320, 180, 50);

// Eyes

drawCircle(220, 220, 15);

drawCircle(280, 220, 15);

// Nose

drawCircle(250, 270, 10);

// Mouth (using 2 circles for simplicity)

drawCircle(235, 300, 20);

drawCircle(265, 300, 20);

}

int main() {

int gd = DETECT, gm;

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

drawTeddyFace();

getch();

closegraph();

return 0;

}

1. Write a Program to draw a car using Bresenham’s algorithm only.

#include <graphics.h>

#include <stdio.h>

#include <conio.h>

#include <math.h>

// Function to draw a line using Bresenham's Line Drawing Algorithm

void drawLine(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;

while (1) {

putpixel(x1, y1, WHITE);

if (x1 == x2 && y1 == y2)

break;

int e2 = 2 \* err;

if (e2 > -dy) {

err -= dy;

x1 += sx;

}

if (e2 < dx) {

err += dx;

y1 += sy;

}

}

}

// Function to draw a circle using Bresenham's Circle Drawing Algorithm

void drawCircle(int xc, int yc, int r) {

int x = 0, y = r;

int d = 3 - 2 \* r;

while (y >= x) {

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

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

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

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

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

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

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

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

if (d < 0)

d = d + 4 \* x + 6;

else {

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

y--;

}

x++;

}

}

int main() {

int gd = DETECT, gm;

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

// Draw car body

drawLine(150, 300, 450, 300); // Bottom

drawLine(150, 200, 450, 200); // Top

drawLine(150, 200, 150, 300); // Left

drawLine(450, 200, 450, 300); // Right

// Draw car roof

drawLine(200, 200, 250, 150); // Left slope

drawLine(250, 150, 350, 150); // Top

drawLine(350, 150, 400, 200); // Right slope

// Draw car windows

drawLine(220, 200, 260, 160); // Front left window bottom

drawLine(260, 160, 340, 160); // Front window top

drawLine(340, 160, 380, 200); // Front right window bottom

drawLine(260, 200, 260, 160); // Left window side

drawLine(340, 200, 340, 160); // Right window side

// Draw wheels using circles

drawCircle(200, 310, 25); // Left wheel

drawCircle(400, 310, 25); // Right wheel

getch();

closegraph(); // Close the graphics mode before exiting

return 0;

}

1. Write a Program to implement Flood fill algorithm for a convex polygon. Draw polygon edges by DDA / Bresenham line algorithm.

#include <graphics.h>

#include <stdio.h>

#include <conio.h>

// Function to draw a line using Bresenham's Line Drawing Algorithm

void drawLine(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 (1) {

putpixel(x1, y1, WHITE);

if (x1 == x2 && y1 == y2)

break;

e2 = 2 \* err;

if (e2 > -dy) {

err -= dy;

x1 += sx;

}

if (e2 < dx) {

err += dx;

y1 += sy;

}

}

}

// Function to implement Flood Fill Algorithm

void floodFill(int x, int y, int fill\_color, int boundary\_color) {

int current = getpixel(x, y);

if (current != boundary\_color && current != fill\_color) {

putpixel(x, y, fill\_color);

floodFill(x + 1, y, fill\_color, boundary\_color);

floodFill(x - 1, y, fill\_color, boundary\_color);

floodFill(x, y + 1, fill\_color, boundary\_color);

floodFill(x, y - 1, fill\_color, boundary\_color);

}

}

int main() {

int gd = DETECT, gm;

// Vertices of the convex polygon

int poly[8]= {40, 40, 60, 20, 80, 40, 60, 60};;

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

// Draw the polygon using Bresenham's line algorithm

drawLine(poly[0], poly[1], poly[2], poly[3]);

drawLine(poly[2], poly[3], poly[4], poly[5]);

drawLine(poly[4], poly[5], poly[6], poly[7]);

drawLine(poly[6], poly[7], poly[0], poly[1]);

// Apply flood fill algorithm to fill the polygon

floodFill(60, 40, YELLOW, WHITE);

getch();

closegraph(); // Close the graphics mode before exiting

return 0;

}

1. Write a Program to implement Boundary fill algorithm for a convex polygon. Draw polygon edges by DDA / Bresenham line algorithm.

#include <graphics.h>

#include <stdio.h>

#include <conio.h>

// Function to draw a line using DDA Line Drawing Algorithm

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

int dx = x2 - x1;

int dy = y2 - y1;

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

float xIncrement = (float)dx / steps;

float yIncrement = (float)dy / steps;

float x = x1;

float y = y1;

int i;

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

putpixel(x, y, WHITE);

x += xIncrement;

y += yIncrement;

}

}

// Function to implement Boundary Fill Algorithm

void boundaryFill(int x, int y, int fill\_color, int boundary\_color) {

int current = getpixel(x, y);

if (current != boundary\_color && current != fill\_color) {

putpixel(x, y, fill\_color);

boundaryFill(x + 1, y, fill\_color, boundary\_color);

boundaryFill(x - 1, y, fill\_color, boundary\_color);

boundaryFill(x, y + 1, fill\_color, boundary\_color);

boundaryFill(x, y - 1, fill\_color, boundary\_color);

}

}

int main() {

int gd = DETECT, gm;

// Vertices of the convex polygon

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

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

// Draw the polygon using DDA line algorithm

drawLineDDA(poly[0], poly[1], poly[2], poly[3]);

drawLineDDA(poly[2], poly[3], poly[4], poly[5]);

drawLineDDA(poly[4], poly[5], poly[6], poly[7]);

drawLineDDA(poly[6], poly[7], poly[0], poly[1]);

// Apply boundary fill algorithm to fill the polygon

boundaryFill(150, 100, YELLOW, WHITE);

getch();

closegraph(); // Close the graphics mode before exiting

return 0;

}

1. Write a Program to implement Fence fill algorithm for a concave polygon. Draw polygon edges by DDA / Bresenham line algorithm

#include <graphics.h>

#include <stdio.h>

#include <conio.h>

#include <math.h>

// Function to draw a line using DDA Line Drawing Algorithm

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

int dx, dy, steps, i;

float xIncrement, yIncrement, x, y;

dx = x2 - x1;

dy = y2 - y1;

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

xIncrement = (float)dx / steps;

yIncrement = (float)dy / steps;

x = x1;

y = y1;

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

putpixel(x, y, color);

x += xIncrement;

y += yIncrement;

}

}

// Function to implement Fence Fill Algorithm for a concave polygon

void fenceFill(int x[], int y[], int n, int fill\_color, int boundary\_color) {

int yMin, yMax, scanline, xIntersections[20], xCount, i, j, k;

float slope;

yMin = y[0];

yMax = y[0];

// Find the minimum and maximum y-coordinates of the polygon

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

if (y[i] < yMin)

yMin = y[i];

if (y[i] > yMax)

yMax = y[i];

}

// Scan each horizontal line within the polygon bounds

for (scanline = yMin + 1; scanline < yMax; scanline++) {

xCount = 0;

// Find intersections of the scanline with polygon edges

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

j = (i + 1) % n;

if ((y[i] <= scanline && y[j] > scanline) || (y[j] <= scanline && y[i] > scanline)) {

slope = (float)(x[j] - x[i]) / (y[j] - y[i]);

xIntersections[xCount++] = x[i] + slope \* (scanline - y[i]);

}

}

// Sort the intersections in ascending order

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

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

if (xIntersections[j] > xIntersections[j + 1]) {

float temp = xIntersections[j];

xIntersections[j] = xIntersections[j + 1];

xIntersections[j + 1] = temp;

}

}

}

// Fill the space between pairs of intersections

for (k = 0; k < xCount - 1; k += 2) {

drawLineDDA(xIntersections[k], scanline, xIntersections[k + 1], scanline, fill\_color);

}

}

// Draw the boundary of the polygon

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

j = (i + 1) % n;

drawLineDDA(x[i], y[i], x[j], y[j], boundary\_color);

}

}

int main() {

int gd = DETECT, gm;

int x[] = {200, 400, 300, 250, 175, 125, 200}; // Vertices of the concave polygon

int y[] = {100, 100, 200, 225, 275, 225, 200};

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

int i;

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

// Apply fence fill algorithm to fill the polygon and draw its boundary

fenceFill(x, y, n, YELLOW, WHITE);

getch();

closegraph(); // Close the graphics mode before exiting

return 0;

}

1. Write a Program to implement Edge fill algorithm for a convex polygon. Draw polygon edges by DDA / Bresenham line algorithm.

#include <graphics.h>

#include <stdio.h>

#include <conio.h>

#include <math.h>

// Function to draw a line using DDA Line Drawing Algorithm

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

int dx, dy, steps, i;

float xIncrement, yIncrement, x, y;

dx = x2 - x1;

dy = y2 - y1;

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

xIncrement = (float)dx / steps;

yIncrement = (float)dy / steps;

x = x1;

y = y1;

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

putpixel(x, y, color);

x += xIncrement;

y += yIncrement;

}

}

// Function to implement Edge Fill Algorithm for a convex polygon

void edgeFill(int x[], int y[], int n, int fill\_color, int boundary\_color) {

int yMin, yMax, scanline, xIntersections[20], xCount, i, j, k;

float slope;

yMin = y[0];

yMax = y[0];

// Find the minimum and maximum y-coordinates of the polygon

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

if (y[i] < yMin)

yMin = y[i];

if (y[i] > yMax)

yMax = y[i];

}

// Scan each horizontal line within the polygon bounds

for (scanline = yMin + 1; scanline < yMax; scanline++) {

xCount = 0;

// Find intersections of the scanline with polygon edges

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

j = (i + 1) % n;

if ((y[i] <= scanline && y[j] > scanline) || (y[j] <= scanline && y[i] > scanline)) {

slope = (float)(x[j] - x[i]) / (y[j] - y[i]);

xIntersections[xCount++] = x[i] + slope \* (scanline - y[i]);

}

}

// Sort the intersections in ascending order

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

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

if (xIntersections[j] > xIntersections[j + 1]) {

float temp = xIntersections[j];

xIntersections[j] = xIntersections[j + 1];

xIntersections[j + 1] = temp;

}

}

}

// Fill the space between pairs of intersections

for (k = 0; k < xCount - 1; k += 2) {

drawLineDDA(xIntersections[k], scanline, xIntersections[k + 1], scanline, fill\_color);

}

}

// Draw the boundary of the polygon

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

j = (i + 1) % n;

drawLineDDA(x[i], y[i], x[j], y[j], boundary\_color);

}

}

int main() {

int gd = DETECT, gm;

// Modified polygon shape

int x[] = {200, 300, 400, 350, 250};

int y[] = {100, 150, 100, 200, 250};

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

int i;

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

// Apply edge fill algorithm to fill the polygon and draw its boundary

edgeFill(x, y, n, YELLOW, WHITE);

getch();

closegraph(); // Close the graphics mode before exiting

return 0;

}

1. Write a Program to implement Scan line fill algorithm for a concave polygon. Draw polygon edges by DDA / Bresenham line algorithm.

#include <graphics.h>

#include <stdio.h>

#include <conio.h>

#include <math.h>

// Function to draw a line using DDA Line Drawing Algorithm

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

int dx, dy, steps, i;

float xIncrement, yIncrement, x, y;

dx = x2 - x1;

dy = y2 - y1;

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

xIncrement = (float)dx / steps;

yIncrement = (float)dy / steps;

x = x1;

y = y1;

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

putpixel(x, y, color);

x += xIncrement;

y += yIncrement;

}

}

// Function to implement Scanline Fill Algorithm for a concave polygon

void scanlineFill(int x[], int y[], int n, int fill\_color, int boundary\_color) {

int yMin, yMax, scanline, i, j, k;

float slope, xIntersections[20];

yMin = y[0];

yMax = y[0];

// Find the minimum and maximum y-coordinates of the polygon

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

if (y[i] < yMin)

yMin = y[i];

if (y[i] > yMax)

yMax = y[i];

}

// Scan each horizontal line within the polygon bounds

for (scanline = yMin + 1; scanline < yMax; scanline++) {

int xCount = 0;

// Find intersections of the scanline with polygon edges

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

j = (i + 1) % n;

if ((y[i] <= scanline && y[j] > scanline) || (y[j] <= scanline && y[i] > scanline)) {

if (y[i] == y[j]) continue; // Horizontal line, skip

slope = (float)(x[j] - x[i]) / (y[j] - y[i]);

xIntersections[xCount++] = x[i] + slope \* (scanline - y[i]);

}

}

// Sort the intersections in ascending order

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

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

if (xIntersections[j] > xIntersections[j + 1]) {

float temp = xIntersections[j];

xIntersections[j] = xIntersections[j + 1];

xIntersections[j + 1] = temp;

}

}

}

// Fill the space between pairs of intersections

for (k = 0; k < xCount - 1; k += 2) {

drawLineDDA(xIntersections[k], scanline, xIntersections[k + 1], scanline, fill\_color);

}

}

// Draw the boundary of the polygon

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

j = (i + 1) % n;

drawLineDDA(x[i], y[i], x[j], y[j], boundary\_color);

}

}

int main() {

int gd = DETECT, gm;

// Vertices of the concave polygon

int x[] = {200, 300, 250, 250, 150, 100};

int y[] = {100, 150, 250, 200, 200, 150};

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

int i;

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

// Apply scanline fill algorithm to fill the polygon and draw its boundary

scanlineFill(x, y, n, YELLOW, WHITE);

getch();

closegraph(); // Close the graphics mode before exiting

return 0;

}

12. Write a Program to implement 2D Scaling and rotation of a triangle.

#include <stdio.h>

#include <graphics.h>

#include <conio.h>

#include <math.h>

int customRound(float num) {

return (num >= 0) ? (int)(num + 0.5) : (int)(num - 0.5);

}

void drawPolygon(int a[][2], int n) {

int i;

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

line(a[i][0], a[i][1], a[i + 1][0], a[i + 1][1]);

}

line(a[n - 1][0], a[n - 1][1], a[0][0], a[0][1]);

}

void scanLineFill(int a[][2], int n) {

int dy, dx;

int x, y, i, j;

int xi[20];

float slope[20];

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

dy = a[(i + 1) % n][1] - a[i][1];

dx = a[(i + 1) % n][0] - a[i][0];

if (dy != 0) {

slope[i] = (float)dx / dy;

} else {

slope[i] = 0;

}

}

for (y = 0; y < 480; y++) {

int k = 0;

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

int ymin = a[i][1] < a[(i + 1) % n][1] ? a[i][1] : a[(i + 1) % n][1];

int ymax = a[i][1] > a[(i + 1) % n][1] ? a[i][1] : a[(i + 1) % n][1];

if (y >= ymin && y < ymax) {

xi[k++] = customRound(a[i][0] + slope[i] \* (y - a[i][1]));

}

}

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

for (j = 0; j < k - 1; j++) {

if (xi[j] > xi[j + 1]) {

int temp = xi[j];

xi[j] = xi[j + 1];

xi[j + 1] = temp;

}

}

}

setcolor(LIGHTBLUE);

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

if (xi[i] < xi[i + 1]) {

line(xi[i], y, xi[i + 1] + 1, y);

}

}

}

}

void drawAxes() {

setcolor(WHITE);

line(0, 0, 630, 0);

line(0, 475, 630, 475);

line(0, 0, 0, 480);

line(630, 0, 630, 475);

line(0, 242, 630, 242);

line(320, 0, 320, 475);

}

void clearScreen() {

cleardevice();

drawAxes();

}

void drawMenu() {

printf("\nMenu:\n");

printf("1. Rotate\n");

printf("2. Scale\n");

printf("3. Exit\n");

}

void rotate(int a[][2], int n, float angle) {

int i;

float radians = angle \* M\_PI / 180.0; // Convert angle to radians

// Calculate the center of the polygon

int centerX = 0, centerY = 0;

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

centerX += a[i][0];

centerY += a[i][1];

}

centerX /= n;

centerY /= n;

// Translate polygon to origin

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

a[i][0] -= centerX;

a[i][1] -= centerY;

}

// Perform rotation

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

int x = a[i][0];

int y = a[i][1];

a[i][0] = centerX + (int)(x \* cos(radians) - y \* sin(radians));

a[i][1] = centerY + (int)(x \* sin(radians) + y \* cos(radians));

}

}

void scale(int a[][2], int n, float sx, float sy) {

int i;

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

a[i][0] = (int)(a[i][0] \* sx + 0.5);

a[i][1] = (int)(a[i][1] \* sy + 0.5);

}

}

void getTriangleInput(int a[][2]) {

printf("Enter coordinates for vertex 1 (x y): ");

scanf("%d %d", &a[0][0], &a[0][1]);

printf("Enter coordinates for vertex 2 (x y): ");

scanf("%d %d", &a[1][0], &a[1][1]);

printf("Enter coordinates for vertex 3 (x y): ");

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

}

int main() {

int n = 3, i; // Change to represent a triangle

int gd = DETECT, gm;

int choice;

float angle, sx, sy;

int a[3][2]; // Coordinates of a triangle

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

drawAxes();

getTriangleInput(a);

scanLineFill(a, n);

getch();

do {

cleardevice();

drawAxes();

scanLineFill(a, n);

drawMenu();

printf("Enter choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter angle of rotation: ");

scanf("%f", &angle);

rotate(a, n, angle);

break;

case 2:

printf("Enter scaling factors (sx sy): ");

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

scale(a, n, sx, sy);

break;

case 3:

break;

default:

printf("Invalid choice\n");

break;

}

} while (choice != 3);

closegraph();

return 0;

}

13. Write a Program to implement 2D Scaling and translation of a triangle.

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

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);

}

int main() {

int gd = DETECT, gm;

int x1, y1, x2, y2, x3, y3;

float sx, sy, tx, ty;

int x1\_new, y1\_new, x2\_new, y2\_new, x3\_new, y3\_new;

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

cleardevice();

// Input the coordinates of the triangle

printf("Enter the coordinates of the first vertex (x1 y1): ");

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

printf("Enter the coordinates of the second vertex (x2 y2): ");

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

printf("Enter the coordinates of the third vertex (x3 y3): ");

scanf("%d %d", &x3, &y3);

// Draw the original triangle

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

// Input the scaling factors

printf("Enter the scaling factors (sx sy): ");

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

// Scale the triangle

x1\_new = x1 \* sx;

y1\_new = y1 \* sy;

x2\_new = x2 \* sx;

y2\_new = y2 \* sy;

x3\_new = x3 \* sx;

y3\_new = y3 \* sy;

// Draw the scaled triangle

setcolor(RED);

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

// Input the translation factors

printf("Enter the translation factors (tx ty): ");

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

// Translate the triangle

x1\_new += tx;

y1\_new += ty;

x2\_new += tx;

y2\_new += ty;

x3\_new += tx;

y3\_new += ty;

// Draw the translated triangle

setcolor(GREEN);

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

getch();

closegraph();

return 0;

}

14. Write a Program to implement 2D rotation and translation of a triangle.

#include <stdio.h>

#include <graphics.h>

#include <conio.h>

#include <math.h>

int customRound(float num) {

return (num >= 0) ? (int)(num + 0.5) : (int)(num - 0.5);

}

void drawPolygon(int a[][2], int n) {

int i;

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

line(a[i][0], a[i][1], a[i + 1][0], a[i + 1][1]);

}

line(a[n - 1][0], a[n - 1][1], a[0][0], a[0][1]);

}

void scanLineFill(int a[][2], int n) {

int dy, dx;

int x, y, i, j;

int xi[20];

float slope[20];

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

dy = a[(i + 1) % n][1] - a[i][1];

dx = a[(i + 1) % n][0] - a[i][0];

if (dy != 0) {

slope[i] = (float)dx / dy;

} else {

slope[i] = 0;

}

}

for (y = 0; y < 480; y++) {

int k = 0;

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

int ymin = a[i][1] < a[(i + 1) % n][1] ? a[i][1] : a[(i + 1) % n][1];

int ymax = a[i][1] > a[(i + 1) % n][1] ? a[i][1] : a[(i + 1) % n][1];

if (y >= ymin && y < ymax) {

xi[k++] = customRound(a[i][0] + slope[i] \* (y - a[i][1]));

}

}

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

for (j = 0; j < k - 1; j++) {

if (xi[j] > xi[j + 1]) {

int temp = xi[j];

xi[j] = xi[j + 1];

xi[j + 1] = temp;

}

}

}

setcolor(LIGHTBLUE);

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

if (xi[i] < xi[i + 1]) {

line(xi[i], y, xi[i + 1] + 1, y);

}

}

}

}

void drawAxes() {

setcolor(WHITE);

line(0, 0, 630, 0);

line(0, 475, 630, 475);

line(0, 0, 0, 480);

line(630, 0, 630, 475);

line(0, 242, 630, 242);

line(320, 0, 320, 475);

}

void clearScreen() {

cleardevice();

drawAxes();

}

void translate(int a[][2], int n, int tx, int ty) {

int i;

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

a[i][0] += tx;

a[i][1] += ty;

}

}

void rotate(int a[][2], int n, float angle) {

int i;

float radians = angle \* M\_PI / 180.0; // Convert angle to radians

// Calculate the center of the polygon

int centerX = 0, centerY = 0;

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

centerX += a[i][0];

centerY += a[i][1];

}

centerX /= n;

centerY /= n;

// Translate polygon to origin

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

a[i][0] -= centerX;

a[i][1] -= centerY;

}

// Perform rotation

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

int x = a[i][0];

int y = a[i][1];

a[i][0] = centerX + (int)(x \* cos(radians) - y \* sin(radians));

a[i][1] = centerY + (int)(x \* sin(radians) + y \* cos(radians));

}

}

void drawMenu() {

printf("\nMenu:\n");

printf("1. Translate\n");

printf("2. Rotate\n");

printf("3. Exit\n");

}

void getTriangleInput(int a[][2]) {

printf("Enter coordinates for vertex 1 (x y): ");

scanf("%d %d", &a[0][0], &a[0][1]);

printf("Enter coordinates for vertex 2 (x y): ");

scanf("%d %d", &a[1][0], &a[1][1]);

printf("Enter coordinates for vertex 3 (x y): ");

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

}

int main() {

int n = 3, i; // Change to represent a triangle

int gd = DETECT, gm;

int choice;

int tx, ty;

float angle;

int a[3][2]; // Coordinates of a triangle

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

drawAxes();

getTriangleInput(a);

scanLineFill(a, n);

getch();

do {

cleardevice();

drawAxes();

scanLineFill(a, n);

drawMenu();

printf("Enter choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter translation values (tx ty): ");

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

translate(a, n, tx, ty);

break;

case 2:

printf("Enter angle of rotation: ");

scanf("%f", &angle);

rotate(a, n, angle);

break;

case 3:

break;

default:

printf("Invalid choice\n");

break;

}

} while (choice != 3);

closegraph();

return 0;

}

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

#include <graphics.h>  
#include <conio.h>  
#include <math.h>  
#include <stdio.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 rotate a point (x, y) by an angle theta  
void rotatePoint(int \*x, int \*y, float theta) {  
 float rad = theta \* M\_PI / 180;  
 int x\_new = \*x \* cos(rad) - \*y \* sin(rad);  
 int y\_new = \*x \* sin(rad) + \*y \* cos(rad);  
 \*x = x\_new;  
 \*y = y\_new;  
}  
  
// Function to rotate a triangle by an angle theta  
void rotateTriangle(int \*x1, int \*y1, int \*x2, int \*y2, int \*x3, int \*y3, float theta) {  
 rotatePoint(x1, y1, theta);  
 rotatePoint(x2, y2, theta);  
 rotatePoint(x3, y3, theta);  
}  
  
int main() {  
 int gd = *DETECT*, gm;  
 int x1 = 100, y1 = 100, x2 = 150, y2 = 50, x3 = 200, y3 = 100;  
 float theta1, theta2;  
  
 initgraph(&gd, &gm, "C:\\Turboc3\\BGI");  
  
 // Draw the original triangle  
 setcolor(*WHITE*);  
 drawTriangle(x1, y1, x2, y2, x3, y3);  
 getch();  
  
 // Prompt for two rotation angles  
 printf("Enter first rotation angle (theta1): ");  
 scanf("%f", &theta1);  
 printf("Enter second rotation angle (theta2): ");  
 scanf("%f", &theta2);  
  
 // Rotate the triangle by theta1  
 rotateTriangle(&x1, &y1, &x2, &y2, &x3, &y3, theta1);  
 cleardevice();  
 setcolor(*YELLOW*);  
 drawTriangle(x1, y1, x2, y2, x3, y3);  
 getch();  
  
 // Rotate the triangle by theta2  
 rotateTriangle(&x1, &y1, &x2, &y2, &x3, &y3, theta2);  
 cleardevice();  
 setcolor(*GREEN*);  
 drawTriangle(x1, y1, x2, y2, x3, y3);  
 getch();  
  
 // Reset triangle points for direct rotation by (theta1 + theta2)  
 x1 = 100; y1 = 100;  
 x2 = 150; y2 = 50;  
 x3 = 200; y3 = 100;  
  
 // Rotate the triangle directly by (theta1 + theta2)  
 rotateTriangle(&x1, &y1, &x2, &y2, &x3, &y3, theta1 + theta2);  
 cleardevice();  
 setcolor(*CYAN*);  
 drawTriangle(x1, y1, x2, y2, x3, y3);  
 getch();  
  
 closegraph();  
 return 0;  
}

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

#include <graphics.h>  
#include <conio.h>  
#include <math.h>  
#include <stdio.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 rotate a point (x, y) by an angle theta  
void rotatePoint(int \*x, int \*y, float theta) {  
 float rad = theta \* M\_PI / 180;  
 int x\_new = \*x \* cos(rad) - \*y \* sin(rad);  
 int y\_new = \*x \* sin(rad) + \*y \* cos(rad);  
 \*x = x\_new;  
 \*y = y\_new;  
}  
  
// Function to rotate a triangle by an angle theta  
void rotateTriangle(int \*x1, int \*y1, int \*x2, int \*y2, int \*x3, int \*y3, float theta) {  
 rotatePoint(x1, y1, theta);  
 rotatePoint(x2, y2, theta);  
 rotatePoint(x3, y3, theta);  
}  
  
int main() {  
 int gd = *DETECT*, gm;  
 int x1 = 100, y1 = 100, x2 = 150, y2 = 50, x3 = 200, y3 = 100;  
 float theta1, theta2;  
  
 initgraph(&gd, &gm, "C:\\Turboc3\\BGI");  
  
 // Draw the original triangle  
 setcolor(*WHITE*);  
 drawTriangle(x1, y1, x2, y2, x3, y3);  
 getch();  
  
 // Prompt for two rotation angles  
 printf("Enter first rotation angle (theta1): ");  
 scanf("%f", &theta1);  
 printf("Enter second rotation angle (theta2): ");  
 scanf("%f", &theta2);  
  
 // Reset triangle points for rotation sequence R(theta1) . R(theta2)  
 int x1\_1 = 100, y1\_1 = 100, x2\_1 = 150, y2\_1 = 50, x3\_1 = 200, y3\_1 = 100;  
 rotateTriangle(&x1\_1, &y1\_1, &x2\_1, &y2\_1, &x3\_1, &y3\_1, theta1);  
 rotateTriangle(&x1\_1, &y1\_1, &x2\_1, &y2\_1, &x3\_1, &y3\_1, theta2);  
  
 // Reset triangle points for rotation sequence R(theta2) . R(theta1)  
 int x1\_2 = 100, y1\_2 = 100, x2\_2 = 150, y2\_2 = 50, x3\_2 = 200, y3\_2 = 100;  
 rotateTriangle(&x1\_2, &y1\_2, &x2\_2, &y2\_2, &x3\_2, &y3\_2, theta2);  
 rotateTriangle(&x1\_2, &y1\_2, &x2\_2, &y2\_2, &x3\_2, &y3\_2, theta1);  
  
 // Clear the screen and draw both results  
 cleardevice();  
 setcolor(*YELLOW*);  
 drawTriangle(x1\_1, y1\_1, x2\_1, y2\_1, x3\_1, y3\_1);  
 setcolor(*CYAN*);  
 drawTriangle(x1\_2, y1\_2, x2\_2, y2\_2, x3\_2, y3\_2);  
 getch();  
  
 closegraph();  
 return 0;  
}

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

#include <graphics.h>

#include <conio.h>

#include <stdio.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 translate a point (x, y) by translation factors tx and ty

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

\*x = \*x + tx;

\*y = \*y + ty;

}

// Function to translate a triangle by translation factors tx and ty

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

translatePoint(x1, y1, tx, ty);

translatePoint(x2, y2, tx, ty);

translatePoint(x3, y3, tx, ty);

}

int main() {

int gd = DETECT, gm;

int x1 = 100, y1 = 100, x2 = 150, y2 = 50, x3 = 200, y3 = 100;

int tx1, ty1, tx2, ty2;

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

// Draw the original triangle

setcolor(WHITE);

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

getch();

// Prompt for two translation factors

printf("Enter first translation factors (tx1 ty1): ");

scanf("%d %d", &tx1, &ty1);

printf("Enter second translation factors (tx2 ty2): ");

scanf("%d %d", &tx2, &ty2);

// Translate the triangle by (tx1, ty1) followed by (tx2, ty2)

int x1\_1 = x1, y1\_1 = y1, x2\_1 = x2, y2\_1 = y2, x3\_1 = x3, y3\_1 = y3;

translateTriangle(&x1\_1, &y1\_1, &x2\_1, &y2\_1, &x3\_1, &y3\_1, tx1, ty1);

translateTriangle(&x1\_1, &y1\_1, &x2\_1, &y2\_1, &x3\_1, &y3\_1, tx2, ty2);

// Reset triangle points for single translation by (tx1 + tx2, ty1 + ty2)

int x1\_2 = x1\_1, y1\_2 = y1\_1, x2\_2 = x2\_1, y2\_2 = y2\_1, x3\_2 = x3\_1, y3\_2 = y3\_1;

translateTriangle(&x1\_2, &y1\_2, &x2\_2, &y2\_2, &x3\_2, &y3\_2, tx1 + tx2, ty1 + ty2);

// Clear the screen and draw both results

cleardevice();

setcolor(YELLOW);

drawTriangle(x1\_1, y1\_1, x2\_1, y2\_1, x3\_1, y3\_1);

setcolor(CYAN);

drawTriangle(x1\_2, y1\_2, x2\_2, y2\_2, x3\_2, y3\_2);

getch();

closegraph();

return 0;

}

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

|  |
| --- |
| #include <graphics.h>  #include <conio.h>  #include <math.h>  #include <stdio.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 rotate a point (x, y) around (cx, cy) by an angle theta  void rotatePoint(int \*x, int \*y, int cx, int cy, float theta) {  float rad = theta \* M\_PI / 180;  int x\_shifted = \*x - cx;  int y\_shifted = \*y - cy;  float x\_new = x\_shifted \* cos(rad) - y\_shifted \* sin(rad);  float y\_new = x\_shifted \* sin(rad) + y\_shifted \* cos(rad);  \*x = cx + (int)(x\_new + 0.5);  \*y = cy + (int)(y\_new + 0.5);  }  // Function to rotate a triangle around its centroid by an angle theta  void rotateTriangle(int \*x1, int \*y1, int \*x2, int \*y2, int \*x3, int \*y3, float theta) {  int cx = (\*x1 + \*x2 + \*x3) / 3;  int cy = (\*y1 + \*y2 + \*y3) / 3;  rotatePoint(x1, y1, cx, cy, theta);  rotatePoint(x2, y2, cx, cy, theta);  rotatePoint(x3, y3, cx, cy, theta);  }  int main() {  int gd = DETECT, gm;  int x1 = 100, y1 = 100, x2 = 150, y2 = 50, x3 = 200, y3 = 100;  float theta1, theta2;  initgraph(&gd, &gm, "C:\\Turboc3\\BGI");  // Draw the original triangle  setcolor(WHITE);  drawTriangle(x1, y1, x2, y2, x3, y3);  getch();  // Prompt for two rotation angles  printf("Enter first rotation angle (theta1): ");  scanf("%f", &theta1);  printf("Enter second rotation angle (theta2): ");  scanf("%f", &theta2);  // Reset triangle points for rotation sequence R(theta1) followed by R(theta2)  int x1\_1 = x1, y1\_1 = y1, x2\_1 = x2, y2\_1 = y2, x3\_1 = x3, y3\_1 = y3;  rotateTriangle(&x1\_1, &y1\_1, &x2\_1, &y2\_1, &x3\_1, &y3\_1, theta1);  rotateTriangle(&x1\_1, &y1\_1, &x2\_1, &y2\_1, &x3\_1, &y3\_1, theta2);  // Reset triangle points for rotation sequence R(theta2) followed by R(theta1)  int x1\_2 = x1, y1\_2 = y1, x2\_2 = x2, y2\_2 = y2, x3\_2 = x3, y3\_2 = y3;  rotateTriangle(&x1\_2, &y1\_2, &x2\_2, &y2\_2, &x3\_2, &y3\_2, theta2);  rotateTriangle(&x1\_2, &y1\_2, &x2\_2, &y2\_2, &x3\_2, &y3\_2, theta1);  // Clear the screen and draw both results  cleardevice();  setcolor(YELLOW);  drawTriangle(x1\_1, y1\_1, x2\_1, y2\_1, x3\_1, y3\_1);  setcolor(CYAN);  drawTriangle(x1\_2, y1\_2, x2\_2, y2\_2, x3\_2, y3\_2);  getch();  closegraph();  return 0; |

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

|  |
| --- |
| #include <graphics.h>  #include <stdlib.h>  #include <stdio.h>  #include<dos.h>  #include<conio.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 translate a triangle  void translateTriangle(int &x1, int &y1, int &x2, int &y2, int &x3, int &y3, int dx, int dy) {  x1 += dx;  y1 += dy;  x2 += dx;  y2 += dy;  x3 += dx;  y3 += dy;  }  int main() {  int gd = DETECT, gm;  initgraph(&gd, &gm, "C://TURBOC3//BGI");  // Original triangle vertices  int x1 = 100, y1 = 100;  int x2 = 150, y2 = 200;  int x3 = 50, y3 = 200;  // Draw original triangle  drawTriangle(x1, y1, x2, y2, x3, y3, WHITE);  delay(1000); // Delay to show the original triangle  // Translation values  int dx1 = 50, dy1 = 50;  int dx2 = 100, dy2 = 100;  // First translation sequence: T(dx1, dy1) followed by T(dx2, dy2)  int x1\_t1 = x1, y1\_t1 = y1;  int x2\_t1 = x2, y2\_t1 = y2;  int x3\_t1 = x3, y3\_t1 = y3;  translateTriangle(x1\_t1, y1\_t1, x2\_t1, y2\_t1, x3\_t1, y3\_t1, dx1, dy1);  drawTriangle(x1\_t1, y1\_t1, x2\_t1, y2\_t1, x3\_t1, y3\_t1, RED);  delay(1000); // Delay to show the first translation  translateTriangle(x1\_t1, y1\_t1, x2\_t1, y2\_t1, x3\_t1, y3\_t1, dx2, dy2);  drawTriangle(x1\_t1, y1\_t1, x2\_t1, y2\_t1, x3\_t1, y3\_t1, BLUE);  delay(1000); // Delay to show the second translation  // Second translation sequence: T(dx2, dy2) followed by T(dx1, dy1)  int x1\_t2 = x1, y1\_t2 = y1;  int x2\_t2 = x2, y2\_t2 = y2;  int x3\_t2 = x3, y3\_t2 = y3;  translateTriangle(x1\_t2, y1\_t2, x2\_t2, y2\_t2, x3\_t2, y3\_t2, dx2, dy2);  drawTriangle(x1\_t2, y1\_t2, x2\_t2, y2\_t2, x3\_t2, y3\_t2, GREEN);  delay(1000); // Delay to show the first translation  translateTriangle(x1\_t2, y1\_t2, x2\_t2, y2\_t2, x3\_t2, y3\_t2, dx1, dy1);  drawTriangle(x1\_t2, y1\_t2, x2\_t2, y2\_t2, x3\_t2, y3\_t2, YELLOW);  delay(1000); // Delay to show the second translation  // Check if both triangles coincide  if (x1\_t1 == x1\_t2 && y1\_t1 == y1\_t2 && x2\_t1 == x2\_t2 && y2\_t1 == y2\_t2 && x3\_t1 == x3\_t2 && y3\_t1 == y3\_t2) {  outtextxy(10, getmaxy() - 20, "Both final triangles coincide, demonstrating commutativity.");  } else {  outtextxy(10, getmaxy() - 20, "Translations do not coincide. There is an error.");  }  // Wait for a key press  getch();  // Close the graphics mode  closegraph();  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.

|  |
| --- |
| #include <graphics.h>  #include <stdio.h>  #include <math.h>  #include<dos.h>  #include<conio.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 reflect a point about the line y=x  void reflectAboutLineYX(int &x, int &y) {  int temp = x;  x = y;  y = temp;  }  // Function to reflect a point about the x-axis  void reflectAboutXAxis(int &x, int &y) {  y = -y;  }  // Function to rotate a point 90 degrees counterclockwise  void rotate90Counterclockwise(int &x, int &y) {  int temp = x;  x = -y;  y = temp;  }  int main() {  int gd = DETECT, gm;  initgraph(&gd, &gm, "C://TURBOC3//BGI");  // Original triangle vertices  int x1 = 100, y1 = 100;  int x2 = 200, y2 = 100;  int x3 = 150, y3 = 200;  // Draw original triangle  drawTriangle(x1, y1, x2, y2, x3, y3, WHITE);  delay(1000); // Delay to show the original triangle  // Reflection about the line y = x  int x1\_reflect = x1, y1\_reflect = y1;  int x2\_reflect = x2, y2\_reflect = y2;  int x3\_reflect = x3, y3\_reflect = y3;  reflectAboutLineYX(x1\_reflect, y1\_reflect);  reflectAboutLineYX(x2\_reflect, y2\_reflect);  reflectAboutLineYX(x3\_reflect, y3\_reflect);  // Draw triangle after reflection about y = x (RED)  drawTriangle(x1\_reflect, y1\_reflect, x2\_reflect, y2\_reflect, x3\_reflect, y3\_reflect, RED);  delay(1000); // Delay to show the reflection about y = x  // Reflection about the x-axis followed by 90 degrees counterclockwise rotation  int x1\_transform = x1, y1\_transform = y1;  int x2\_transform = x2, y2\_transform = y2;  int x3\_transform = x3, y3\_transform = y3;  reflectAboutXAxis(x1\_transform, y1\_transform);  reflectAboutXAxis(x2\_transform, y2\_transform);  reflectAboutXAxis(x3\_transform, y3\_transform);  rotate90Counterclockwise(x1\_transform, y1\_transform);  rotate90Counterclockwise(x2\_transform, y2\_transform);  rotate90Counterclockwise(x3\_transform, y3\_transform);  // Draw triangle after reflection about x-axis and rotation (GREEN)  drawTriangle(x1\_transform, y1\_transform, x2\_transform, y2\_transform, x3\_transform, y3\_transform, GREEN);  delay(1000); // Delay to show the combined transformation  // Wait for a key press  getch();  // Close the graphics mode  closegraph();  return 0;  } |

21 :Write a Program to implement all type of reflections about X axis and about Y axis of a triangle.

#include <graphics.h>

#include <conio.h>

#include <stdlib.h>

#include <stdio.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);

}

int main() {

    int gd = DETECT, gm;

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

    int x1, y1, x2, y2, x3, y3;

    // Ask the user to enter the coordinates of the triangle

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

    printf("Point 1 (x1, y1): ");

    scanf("%d %d", &x1, &y1);

    printf("Point 2 (x2, y2): ");

    scanf("%d %d", &x2, &y2);

    printf("Point 3 (x3, y3): ");

    scanf("%d %d", &x3, &y3);

    // Draw the original triangle

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

    outtextxy(100, 20, "Original Triangle");

    // Reflect about X-axis

    drawTriangle(x1, getmaxy() - y1, x2, getmaxy() - y2, x3, getmaxy() - y3, RED);

    outtextxy(100, getmaxy() - 40, "Reflected about X-axis");

    // Reflect about Y-axis

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

    outtextxy(getmaxx() - 300, 20, "Reflected about Y-axis");

    // Reflect about both X-axis and Y-axis

    drawTriangle(getmaxx() - x1, getmaxy() - y1, getmaxx() - x2, getmaxy() - y2, getmaxx() - x3, getmaxy() - y3, BLUE);

    outtextxy(getmaxx() - 300, getmaxy() - 40, "Reflected about Both axes");

    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

#include <graphics.h>

#include <conio.h>

#include <stdlib.h>

#include <stdio.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);

}

int main() {

    int gd = DETECT, gm;

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

    int x1, y1, x2, y2, x3, y3;

    // Ask the user to enter the coordinates of the triangle

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

    printf("Point 1 (x1, y1): ");

    scanf("%d %d", &x1, &y1);

    printf("Point 2 (x2, y2): ");

    scanf("%d %d", &x2, &y2);

    printf("Point 3 (x3, y3): ");

    scanf("%d %d", &x3, &y3);

    // Draw the original triangle

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

    outtextxy(100, 20, "Original Triangle");

    // Reflect about the origin

    drawTriangle(-x1 + getmaxx() / 2, -y1 + getmaxy() / 2,

                 -x2 + getmaxx() / 2, -y2 + getmaxy() / 2,

                 -x3 + getmaxx() / 2, -y3 + getmaxy() / 2, YELLOW);

    outtextxy(100, 40, "Reflected about Origin");

    // Reflect about the line y = x

    drawTriangle(y1 + getmaxx() / 2, x1 + getmaxy() / 2,

                 y2 + getmaxx() / 2, x2 + getmaxy() / 2,

                 y3 + getmaxx() / 2, x3 + getmaxy() / 2, CYAN);

    outtextxy(100, 60, "Reflected about line y=x");

    getch();

    closegraph();

    return 0;

}

23:Write a Program to implement X and Y shear transformation.

#include <stdio.h>

#include <conio.h>

#include <graphics.h>

#include <math.h>

int main() {

int gm, gr, x, y, az, w, ch2;

float x1, y1, az1, w1, x1s, y1s;

clrscr();

detectgraph(&gm, &gr);

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

printf("Enter the upper left corner of the rectangle:\n");

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

printf("Enter the lower right corner of the rectangle:\n");

scanf("%d%d", &az, &w);

rectangle(x, y, az, w);

printf("Shearing\n\n");

printf("1.x-direction shear\n2.y-direction shear\nEnter your choice:\n");

scanf("%d", &ch2);

switch (ch2) {

case 1:

printf("Enter the value of shear in x-direction:\n");

scanf("%f", &x1s);

x1 = x + (y \* x1s);

y1 = y;

az1 = az + (w \* x1s);

w1 = w;

rectangle(x1, y1, az1, w1);

break;

case 2:

printf("Enter the value of shear in y-direction:\n");

scanf("%f", &y1s);

x1 = x;

y1 = y + (x \* y1s);

az1 = az;

w1 = w + (az \* y1s);

rectangle(x1, y1, az1, w1);

break;

default:

printf("Invalid choice\n");

break;

}

getch();

closegraph();

return 0;

}

24 :Write a Program to implement rotation about arbitrary point.

#include <graphics.h>

#include <conio.h>

#include <stdlib.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 translation

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

    \*x = \*x + tx;

    \*y = \*y + ty;

}

// Function to rotate a point around the origin

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

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

    int nx = (int)(\*x \* cos(rad) - \*y \* sin(rad));

    int ny = (int)(\*x \* sin(rad) + \*y \* cos(rad));

    \*x = nx;

    \*y = ny;

}

int main() {

    int gd = DETECT, gm;

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

    int x1, y1, x2, y2, x3, y3, cx, cy;

    float angle;

    // Ask the user to enter the coordinates of the triangle

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

    printf("Point 1 (x1, y1): ");

    scanf("%d %d", &x1, &y1);

    printf("Point 2 (x2, y2): ");

    scanf("%d %d", &x2, &y2);

    printf("Point 3 (x3, y3): ");

    scanf("%d %d", &x3, &y3);

    // Ask the user to enter the coordinates of the center of rotation

    printf("Enter the coordinates of the center of rotation (cx, cy): ");

    scanf("%d %d", &cx, &cy);

    // Ask the user to enter the angle of rotation

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

    scanf("%f", &angle);

    // Draw the original triangle

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

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

    // Translate points to the origin relative to the center of rotation

    int tx1 = x1 - cx, ty1 = y1 - cy;

    int tx2 = x2 - cx, ty2 = y2 - cy;

    int tx3 = x3 - cx, ty3 = y3 - cy;

    // Rotate the translated points around the origin

    rotateAroundOrigin(&tx1, &ty1, angle);

    rotateAroundOrigin(&tx2, &ty2, angle);

    rotateAroundOrigin(&tx3, &ty3, angle);

    // Translate points back to the original position

    translatePoint(&tx1, &ty1, cx, cy);

    translatePoint(&tx2, &ty2, cx, cy);

    translatePoint(&tx3, &ty3, cx, cy);

    // Draw the rotated triangle

    drawTriangle(tx1, ty1, tx2, ty2, tx3, ty3, YELLOW);

    outtextxy(10, 30, "Rotated Triangle");

    getch();

    closegraph();

    return 0;

}

25:Write a Program to implement Cohen Sutherland line clipping algorithm.

#include <graphics.h>

#include <stdio.h>

#include<conio.h>

// Define region codes

#define INSIDE 0 // 0000

#define LEFT 1   // 0001

#define RIGHT 2  // 0010

#define BOTTOM 4 // 0100

#define TOP 8    // 1000

// Define screen boundaries

#define X\_MIN 100

#define Y\_MIN 100

#define X\_MAX 500

#define Y\_MAX 400

// Function to compute region code for a point (x, y)

int computeCode(int x, int y) {

    int code = INSIDE; // Initialize code as inside

    if (x < X\_MIN)      // to the left of rectangle

        code |= LEFT;

    else if (x > X\_MAX) // to the right of rectangle

        code |= RIGHT;

    if (y < Y\_MIN)      // below rectangle

        code |= BOTTOM;

    else if (y > Y\_MAX) // above rectangle

        code |= TOP;

    return code;

}

// Cohen-Sutherland clipping algorithm

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

    int code1 = computeCode(x1, y1);

    int code2 = computeCode(x2, y2);

    // Initially assume both points are inside the clip window

    int accept = 0;

    while (1) {

        if (!(code1 | code2)) { // Both endpoints are inside

            accept = 1;

            break;

        } else if (code1 & code2) { // Both endpoints are outside on same side

            break;

        } else {

            // Clip the line from an outside point to the edge of the window

            int x, y;

            // Select an endpoint that is outside

            int codeOut = code1 ? code1 : code2;

            // Find intersection point

            if (codeOut & TOP) {        // Point is above the clip window

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

                y = Y\_MAX;

            } else if (codeOut & BOTTOM) { // Point is below the clip window

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

                y = Y\_MIN;

            } else if (codeOut & RIGHT) {  // Point is to the right of clip window

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

                x = X\_MAX;

            } else if (codeOut & LEFT) {   // Point is to the left of clip window

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

                x = X\_MIN;

            }

            // Replace the outside point with the intersection point

            if (codeOut == code1) {

                x1 = x;

                y1 = y;

                code1 = computeCode(x1, y1);

            } else {

                x2 = x;

                y2 = y;

                code2 = computeCode(x2, y2);

            }

        }

    }

    // If line is accepted, draw it

    if (accept) {

        setcolor(RED);

        line(x1, y1, x2, y2);

    }

}

int main() {

    int gd = DETECT, gm;

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

    // Draw the clipping window

    rectangle(X\_MIN, Y\_MIN, X\_MAX, Y\_MAX);

    // Input the endpoints of the line

    int x1, y1, x2, y2;

    printf("Enter endpoints of line (x1 y1 x2 y2): ");

    scanf("%d %d %d %d", &x1, &y1, &x2, &y2);

    // Clip and draw the line

    cohenSutherland(x1, y1, x2, y2);

    getch();

    closegraph();

    return 0;

}

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

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<dos.h>

#include<math.h>

#include<graphics.h>

#define TOP 1

#define BOTTOM 2

#define RIGHT 4

#define LEFT 8

void drawwindow(int x1, int y1, int x2, int y2);

void drawline (int x1, int y1, int x2, int y2, int cl);

int setcode(int x, int y, int x1, int y1, int x2, int y2);

int visibility (int code1, int code2);

void midsub(int x1, int y1, int x2, int y2, int win\_x1, int win\_y1, int

win\_x2, int win\_y2);

int main()

{

int gd=DETECT, gm, v, x1, y1, x2, y2, win\_x1, win\_y1, win\_x2,

win\_y2;

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

cleardevice();

printf("\n\n\t\tENTER WINDOW COORDINATES (x1, y1, x2, y2):");

scanf("%d %d %d %d", &win\_x1, &win\_y1, &win\_x2, &win\_y2);

drawwindow(win\_x1, win\_y1, win\_x2, win\_y2);

getch();

printf("\n\n\t\tENTER END-POINT 1 (x,y): ");

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

printf("\n\n\t\tENTER END-POINT 2 (x,y): ");

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

cleardevice();

drawwindow(win\_x1, win\_y1, win\_x2, win\_y2);

getch();

drawline(x1, y1, x2, y2, 15);

getch();

cleardevice();

drawwindow(win\_x1, win\_y1, win\_x2, win\_y2);

midsub(x1, y1, x2, y2, win\_x1, win\_y1, win\_x2, win\_y2);

getch();

closegraph();

return(0);

}

void midsub(int x1, int y1, int x2, int y2, int win\_x1, int win\_y1, int

win\_x2, int win\_y2)

{

int code1, code2, v;

code1 = setcode(x1, y1, win\_x1, win\_y1, win\_x2, win\_y2);

code2 = setcode(x2, y2, win\_x1, win\_y1, win\_x2, win\_y2);

v = visibility(code1, code2);

switch(v)

{

case 0: /\* Line completely visible \*/

drawline(x1, y1, x2, y2, 15);

break;

case 1: /\* Line completely invisible \*/

break;

case 2: /\* Line partly visible \*/

{

int midx = (x1 + x2) / 2;

int midy = (y1 + y2) / 2;

if ((x2 - x1) \* (x2 - x1) + (y2 - y1) \* (y2 - y1) < 2) {

// Stop recursion if the line segment is very small

break;

}

midsub(x1, y1, midx, midy, win\_x1, win\_y1, win\_x2,

win\_y2);

midx++;

midy++;

midsub(midx, midy, x2, y2, win\_x1, win\_y1, win\_x2,

win\_y2);

break;

}

}

}

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

{

setcolor(RED);

line(x1, y1, x2, y1);

line(x2, y1, x2, y2);

line(x2, y2, x1, y2);

line(x1, y2, x1, y1);

}

void drawline (int x1, int y1, int x2, int y2, int cl)

{

setcolor(cl);

line(x1, y1, x2, y2);

}

int setcode(int x, int y, int x1, int y1, int x2, int y2)

{

int code = 0;

if(y <= y1)

code |= TOP;

if(y >= y2)

code |= BOTTOM;

if(x >= x2)

code |= RIGHT;

if(x <= x1)

code |= LEFT;

return code;

}

int visibility (int code1, int code2)

{

if((code1 & code2) != 0)

return 1; // Line completely invisible

else if((code1 | code2) == 0)

return 0; // Line completely visible

else

return 2; // Line partly visible

}

27. Write a Program to implement Sutherland-Hodgeman Polygon clipping algorithm.

#include <stdio.h>

#include <graphics.h>

#include <conio.h>

#include <stdlib.h>

typedef struct {

int x, y;

} Point;

void clipLeft(Point \*input, int inputCount, Point \*output, int \*outputCount, int xmin) {

int i;

\*outputCount = 0;

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

int k = (i + 1) % inputCount;

Point current = input[i];

Point next = input[k];

int x, y;

if (current.x >= xmin && next.x >= xmin) {

output[\*outputCount] = next;

(\*outputCount)++;

} else if (current.x >= xmin && next.x < xmin) {

x = xmin;

y = current.y + (next.y - current.y) \* (xmin - current.x) / (next.x - current.x);

output[\*outputCount].x = x;

output[\*outputCount].y = y;

(\*outputCount)++;

} else if (current.x < xmin && next.x >= xmin) {

x = xmin;

y = current.y + (next.y - current.y) \* (xmin - current.x) / (next.x - current.x);

output[\*outputCount].x = x;

output[\*outputCount].y = y;

(\*outputCount)++;

output[\*outputCount] = next;

(\*outputCount)++;

}

}

}

void clipRight(Point \*input, int inputCount, Point \*output, int \*outputCount, int xmax) {

int i;

\*outputCount = 0;

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

int k = (i + 1) % inputCount;

Point current = input[i];

Point next = input[k];

int x, y;

if (current.x <= xmax && next.x <= xmax) {

output[\*outputCount] = next;

(\*outputCount)++;

} else if (current.x <= xmax && next.x > xmax) {

x = xmax;

y = current.y + (next.y - current.y) \* (xmax - current.x) / (next.x - current.x);

output[\*outputCount].x = x;

output[\*outputCount].y = y;

(\*outputCount)++;

} else if (current.x > xmax && next.x <= xmax) {

x = xmax;

y = current.y + (next.y - current.y) \* (xmax - current.x) / (next.x - current.x);

output[\*outputCount].x = x;

output[\*outputCount].y = y;

(\*outputCount)++;

output[\*outputCount] = next;

(\*outputCount)++;

}

}

}

void clipBottom(Point \*input, int inputCount, Point \*output, int \*outputCount, int ymin) {

int i;

\*outputCount = 0;

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

int k = (i + 1) % inputCount;

Point current = input[i];

Point next = input[k];

int x, y;

if (current.y >= ymin && next.y >= ymin) {

output[\*outputCount] = next;

(\*outputCount)++;

} else if (current.y >= ymin && next.y < ymin) {

x = current.x + (next.x - current.x) \* (ymin - current.y) / (next.y - current.y);

y = ymin;

output[\*outputCount].x = x;

output[\*outputCount].y = y;

(\*outputCount)++;

} else if (current.y < ymin && next.y >= ymin) {

x = current.x + (next.x - current.x) \* (ymin - current.y) / (next.y - current.y);

y = ymin;

output[\*outputCount].x = x;

output[\*outputCount].y = y;

(\*outputCount)++;

output[\*outputCount] = next;

(\*outputCount)++;

}

}

}

void clipTop(Point \*input, int inputCount, Point \*output, int \*outputCount, int ymax) {

int i;

\*outputCount = 0;

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

int k = (i + 1) % inputCount;

Point current = input[i];

Point next = input[k];

int x, y;

if (current.y <= ymax && next.y <= ymax) {

output[\*outputCount] = next;

(\*outputCount)++;

} else if (current.y <= ymax && next.y > ymax) {

x = current.x + (next.x - current.x) \* (ymax - current.y) / (next.y - current.y);

y = ymax;

output[\*outputCount].x = x;

output[\*outputCount].y = y;

(\*outputCount)++;

} else if (current.y > ymax && next.y <= ymax) {

x = current.x + (next.x - current.x) \* (ymax - current.y) / (next.y - current.y);

y = ymax;

output[\*outputCount].x = x;

output[\*outputCount].y = y;

(\*outputCount)++;

output[\*outputCount] = next;

(\*outputCount)++;

}

}

}

void sutherlandHodgmanClip(Point \*polygon, int \*vertexCount, int xmin, int ymin, int xmax, int ymax) {

Point temp1[20], temp2[20];

int i, tempCount1, tempCount2;

clipLeft(polygon, \*vertexCount, temp1, &tempCount1, xmin);

clipRight(temp1, tempCount1, temp2, &tempCount2, xmax);

clipBottom(temp2, tempCount2, temp1, &tempCount1, ymin);

clipTop(temp1, tempCount1, temp2, &tempCount2, ymax);

\*vertexCount = tempCount2;

for ( i = 0; i < tempCount2; i++) {

polygon[i] = temp2[i];

}

}

int main() {

int n, i;

int xmin, ymin, xmax, ymax;

Point polygon[20];

int gd = DETECT, gm;

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

printf("Enter number of vertices of the polygon: ");

scanf("%d", &n);

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

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

printf("Vertex %d: ", i + 1);

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

}

printf("Enter the clipping window (xmin, ymin, xmax, ymax): ");

scanf("%d %d %d %d", &xmin, &ymin, &xmax, &ymax);

setcolor(WHITE);

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

int j = (i + 1) % n;

line(polygon[i].x, polygon[i].y, polygon[j].x, polygon[j].y);

}

rectangle(xmin, ymin, xmax, ymax);

getch();

cleardevice();

sutherlandHodgmanClip(polygon, &n, xmin, ymin, xmax, ymax);

setcolor(WHITE);

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

int j = (i + 1) % n;

line(polygon[i].x, polygon[i].y, polygon[j].x, polygon[j].y);

}

rectangle(xmin, ymin, xmax, ymax);

getch();

closegraph();

return 0;

}

28. Write a Program to implement Generalized Polygon clipping algorithm.

#include <graphics.h>

#include <conio.h>

#include <stdio.h>

#include <stdlib.h>

#define MAX\_POINTS 20

typedef struct {

int x, y;

} Point;

typedef struct {

Point points[MAX\_POINTS];

int n;

} Polygon;

typedef struct {

Point p1, p2;

} Edge;

Point intersection(Point p1, Point p2, Point cp1, Point cp2) {

Point i;

float a1 = p2.y - p1.y;

float b1 = p1.x - p2.x;

float c1 = a1 \* p1.x + b1 \* p1.y;

float a2 = cp2.y - cp1.y;

float b2 = cp1.x - cp2.x;

float c2 = a2 \* cp1.x + b2 \* cp1.y;

float determinant = a1 \* b2 - a2 \* b1;

if (determinant == 0) {

i.x = p1.x;

i.y = p1.y;

} else {

i.x = (b2 \* c1 - b1 \* c2) / determinant;

i.y = (a1 \* c2 - a2 \* c1) / determinant;

}

return i;

}

int inside(Point p, Point cp1, Point cp2) {

return (cp2.x - cp1.x) \* (p.y - cp1.y) > (cp2.y - cp1.y) \* (p.x - cp1.x);

}

void addPoint(Polygon \*poly, Point p) {

if (poly->n < MAX\_POINTS) {

poly->points[poly->n] = p;

poly->n++;

}

}

void weilerAtherton(Polygon \*subject, Polygon \*clipper, Polygon \*result) {

int i, j, k;

Point p, s, iP;

Polygon temp;

Point cp1 = clipper->points[j];

Point cp2 = clipper->points[(j + 1) % clipper->n];

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

s = subject->points[i];

for (j = 0; j < clipper->n; j++) {

temp.n = 0;

for (k = 0; k < subject->n; k++) {

p = subject->points[k];

if (inside(p, cp1, cp2)) {

if (!inside(s, cp1, cp2)) {

iP = intersection(s, p, cp1, cp2);

addPoint(&temp, iP);

}

addPoint(&temp, p);

} else if (inside(s, cp1, cp2)) {

iP = intersection(s, p, cp1, cp2);

addPoint(&temp, iP);

}

s = p;

}

\*subject = temp;

}

}

\*result = \*subject;

}

void drawPolygon(Polygon \*poly, int color) {

int i;

setcolor(color);

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

line(poly->points[i].x, poly->points[i].y, poly->points[(i + 1) % poly->n].x, poly->points[(i + 1) % poly->n].y);

}

}

void inputPolygon(Polygon \*poly) {

int i;

printf("Enter number of vertices: ");

scanf("%d", &poly->n);

if (poly->n > MAX\_POINTS) {

printf("Number of vertices should not exceed %d\n", MAX\_POINTS);

exit(1);

}

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

printf("Enter vertex %d (x y): ", i + 1);

scanf("%d %d", &poly->points[i].x, &poly->points[i].y);

}

}

int main() {

Polygon subject, clipper, result;

int gd = DETECT, gm;

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

result.n = 0;

printf("Enter the subject polygon:\n");

inputPolygon(&subject);

printf("Enter the clipping polygon:\n");

inputPolygon(&clipper);

setbkcolor(WHITE);

cleardevice();

drawPolygon(&subject, RED);

drawPolygon(&clipper, BLUE);

getch();

cleardevice();

weilerAtherton(&subject, &clipper, &result);

drawPolygon(&result, GREEN);

drawPolygon(&clipper, BLUE);

getch();

closegraph();

return 0;

}

29 Write a Program to draw a Koch curve upto ‘n’ iterations.

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <graphics.h>

// Function to draw a line

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

line(x1, y1, x2, y2);

}

// Function to draw the Koch curve

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

int deltaX = x2 - x1;

int deltaY = y2 - y1;

int x3 = x1 + deltaX / 3;

int y3 = y1 + deltaY / 3;

int x4 = x1 + deltaX \* 2 / 3;

int y4 = y1 + deltaY \* 2 / 3;

int x = x3 + (x4 - x3) / 2 - (int)(deltaY \* sqrt(3) / 6);

int y = y3 + (y4 - y3) / 2 + (int)(deltaX \* sqrt(3) / 6);

if (depth == 0) {

drawLine(x1, y1, x2, y2);

return;

}

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

kochCurve(x3, y3, x, y, depth - 1);

kochCurve(x, y, x4, y4, depth - 1);

kochCurve(x4, y4, x2, y2, depth - 1);

}

int main() {

int choice, gd = DETECT, gm;

int x1, y1, x2, y2, depth;

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

printf("Enter the starting point (x1, y1): ");

//200 200

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

printf("Enter the length of the side: ");

// 400

scanf("%d", &depth);

kochCurve(x1, y1, x1 + depth, y1, 5);

getch();

closegraph();

return 0;

}

30 Write a Program to draw a Hilbert curve upto ‘n’ iterations.

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <graphics.h>

// Function to draw a line

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

line(x1, y1, x2, y2);

}

// Function to draw a Hilbert curve

void hilbertCurve(int x, int y, int xi, int xj, int yi, int yj, int depth) {

if (depth <= 0) {

drawLine(x, y, x + xi, y + xj);

return;

}

hilbertCurve(x, y, yi / 2, yj / 2, xi / 2, xj / 2, depth - 1);

hilbertCurve(x + xi / 2, y + xj / 2, xi / 2, xj / 2, yi / 2, yj / 2, depth - 1);

hilbertCurve(x + xi / 2 + yi / 2, y + xj / 2 + yj / 2, xi / 2, xj / 2, yi / 2, yj / 2, depth - 1);

hilbertCurve(x + xi / 2 + yi, y + xj / 2 + yj, -yi / 2, -yj / 2, -xi / 2, -xj / 2, depth - 1);

}

int main() {

int choice, gd = DETECT, gm;

int x1, y1, x2, y2, depth;

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

printf("Enter the starting point (x1, y1): ");

// 100 100

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

printf("Enter the depth of recursion: ");

// 3

scanf("%d", &depth);

hilbertCurve(x1, y1, getmaxx() / 2, 0, 0, getmaxy() / 2, depth);

getch();

closegraph();

return 0;

}

31. Write a Program to draw a Bezier curve upto ‘n’ iterations using midpoint method.

#include <stdio.h>

#include <stdlib.h>

#include <graphics.h>

typedef struct {

double x;

double y;

} Point;

Point midpoint(Point p1, Point p2) {

Point mid;

mid.x = (p1.x + p2.x) / 2;

mid.y = (p1.y + p2.y) / 2;

return mid;

}

void drawBezier(Point p0, Point p1, Point p2, Point p3, int iterations) {

if (iterations == 0) {

line((int)p0.x, (int)p0.y, (int)p3.x, (int)p3.y);

} else {

Point p01 = midpoint(p0, p1);

Point p12 = midpoint(p1, p2);

Point p23 = midpoint(p2, p3);

Point p012 = midpoint(p01, p12);

Point p123 = midpoint(p12, p23);

Point p0123 = midpoint(p012, p123);

drawBezier(p0, p01, p012, p0123, iterations - 1);

drawBezier(p0123, p123, p23, p3, iterations - 1);

}

}

void drawBezierCurve() {

int gd = DETECT, gm;

Point p0, p1, p2, p3;

int iterations;

printf("Enter x and y coordinates for Point 1: ");

scanf("%lf %lf", &p0.x, &p0.y);

printf("Enter x and y coordinates for Point 2: ");

scanf("%lf %lf", &p1.x, &p1.y);

printf("Enter x and y coordinates for Point 3: ");

scanf("%lf %lf", &p2.x, &p2.y);

printf("Enter x and y coordinates for Point 4: ");

scanf("%lf %lf", &p3.x, &p3.y);

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

scanf("%d", &iterations);

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

setcolor(WHITE);

drawBezier(p0, p1, p2, p3, iterations);

getch();

closegraph();

}

int main() {

drawBezierCurve();

return 0;

}

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

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <graphics.h>

// Function to draw a line

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

line(x1, y1, x2, y2);

}

// Function to draw a fractal line

void fractalLine(int x1, int y1, int x2, int y2, int depth) {

int midX = (x1 + x2) / 2;

int midY = (y1 + y2) / 2;

int newX = (x1 + midX) / 2 + (x2 + midX) / 2 - midX;

int newY = (y1 + midY) / 2 + (y2 + midY) / 2 - midY;

if (depth == 0) {

drawLine(x1, y1, x2, y2);

return;

}

fractalLine(x1, y1, midX, midY, depth - 1);

fractalLine(midX, midY, newX, newY, depth - 1);

fractalLine(newX, newY, x2, y2, depth - 1);

}

int main() {

int choice, gd = DETECT, gm;

int x1, y1, x2, y2, depth;

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

printf("Enter the starting point (x1, y1): ");

//100 100

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

printf("Enter the end point (x2, y2): ");

//700 200

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

printf("Enter the depth of recursion: ");

// 5

scanf("%d", &depth);

fractalLine(x1, y1, x2, y2, depth);

getch();

closegraph();

return 0;

}

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

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <graphics.h>

// Function to draw a line

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

line(x1, y1, x2, y2);

}

// Function to draw a fractal curve

void fractalCurve(int x1, int y1, int x2, int y2, int depth) {

int deltaX = x2 - x1;

int deltaY = y2 - y1;

int x3 = x1 + deltaX / 3;

int y3 = y1 + deltaY / 3;

int x4 = x1 + deltaX \* 2 / 3;

int y4 = y1 + deltaY \* 2 / 3;

int x = x3 + (x4 - x3) / 2 - (int)(deltaY \* sqrt(3) / 6);

int y = y3 + (y4 - y3) / 2 + (int)(deltaX \* sqrt(3) / 6);

if (depth == 0) {

drawLine(x1, y1, x2, y2);

return;

}

fractalCurve(x1, y1, x3, y3, depth - 1);

fractalCurve(x3, y3, x, y, depth - 1);

fractalCurve(x, y, x4, y4, depth - 1);

fractalCurve(x4, y4, x2, y2, depth - 1);

}

int main() {

int choice, gd = DETECT, gm;

int x1, y1, x2, y2, depth;

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

printf("Enter the starting point (x1, y1): ");

// 100 200

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

printf("Enter the end point (x2, y2): ");

// 700 600

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

printf("Enter the depth of recursion: ");

// 4

scanf("%d", &depth);

fractalCurve(x1, y1, x2, y2, depth);

getch();

closegraph();

return 0;

}

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).

34.1 pendulum

#include <graphics.h>

#include <stdio.h>

#include <math.h>

#include <conio.h>

#define PI 3.14

int gd = DETECT, gm;

int pivotx, pivoty;

double thetamax, theta;

double len = 260;

int x, y, ymax, xmax;

int bobradius = 30;

int xsign = -1, ysign = 1;

double omega;

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

    int dx = x2 - x1;

    int dy = y2 - y1;

    int steps, k;

    float xIncrement, yIncrement, x = x1, y = y1;

    if (abs(dx) > abs(dy))

        steps = abs(dx);

    else

        steps = abs(dy);

    xIncrement = dx / (float) steps;

    yIncrement = dy / (float) steps;

    for (k = 0; k < steps; k++) {

        x += xIncrement;

        y += yIncrement;

        if ((int)(x + 0.5) == x2 && (int)(y + 0.5) == y2) {

            break; // Stop drawing at the endpoint

        }

        putpixel((int)(x + 0.5), (int)(y + 0.5), WHITE);

    }

}

void brescir(int x, int y, int r, int color) {

    int i = 0, j = r, p = 3 - 2 \* r;

    while (i <= j) {

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

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

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

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

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

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

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

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

        if (p < 0) {

            i++;

            p = p + 4 \* i + 6;

        } else {

            i++;

            j--;

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

        }

    }

}

void paint() {

    // Clear the screen

    cleardevice();

    // Draw the pivot

    setcolor(WHITE);

    brescir(pivotx, pivoty, 8,WHITE);

    setfillstyle(SOLID\_FILL, WHITE);

    floodfill(pivotx, pivoty, WHITE);

    // Draw the pendulum line and bob

    drawLine(pivotx, pivoty, x, y);

    setcolor(WHITE);

    brescir(x, y, bobradius,WHITE);

      setfillstyle(SOLID\_FILL, WHITE);

    floodfill(x,y, WHITE);

}

void main() {

    double decr;

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

    thetamax = 60 \* PI / 180;

    pivotx = getmaxx() / 2;

    pivoty = 30;

    ymax = (int)(pivoty + len \* cos(thetamax));

    xmax = (int)(pivotx + len \* sin(thetamax));

    x = xmax;

    y = ymax;

    theta = thetamax;

    while (1) {

        if (kbhit() && getch() == 13) { // Check if Enter key (ASCII 13) is pressed

            break;

        }

        if (x >= pivotx + abs(len \* sin(thetamax))) {

            xsign = -1;

            ysign \*= -1;

            x = xmax - 1;

            delay(40);

        } else if (x <= pivotx - abs(len \* sin(thetamax))) {

            ysign \*= -1;

            xsign = 1;

            x = (int)(pivotx - abs(len \* sin(thetamax)) + 2);

            delay(40);

        } else if (y >= pivoty + len) {

            ysign \*= -1;

        }

        omega = y / 60 \* PI / 180;

        decr = xsign \* omega;

        theta = theta + decr;

        x = (int)(pivotx + len \* sin(theta));

        y = (int)(pivoty + len \* cos(theta));

        paint();

        delay(40);

    }

    closegraph();

}

34.2. circle in circle

#include <graphics.h>

#include <dos.h>

#include <stdio.h>

#include <conio.h>

#include <math.h>

void brescir(int x, int y, int r, int color) {

    int i = 0, j = r, p = 3 - 2 \* r;

    while (i <= j) {

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

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

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

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

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

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

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

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

        if (p < 0) {

            i++;

            p = p + 4 \* i + 6;

        } else {

            i++;

            j--;

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

        }

    }

}

#define PI 3.14159265

int main()

{

    // Radius and center of the main circle

    int centerX = 150;

    int centerY = 150;

    int radius\_main = 100; // Radius of the main circle

    int radius\_moving = 12; // Radius of the moving circle

    // Initial angle for the moving circle

    float angle = 0;

    int gd = DETECT, gm, i, x, y;

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

    while (1)

    {

        // Clear the screen

        cleardevice();

        // Draw the main circle using Bresenham's algorithm

        brescir(centerX, centerY, radius\_main, WHITE);

        // Calculate position of moving circle

        x = centerX + (int)((radius\_main - radius\_moving) \* cos(angle \* PI / 180));

        y = centerY + (int)((radius\_main - radius\_moving) \* sin(angle \* PI / 180));

        // Draw the moving circle using Bresenham's algorithm

        brescir(x, y, radius\_moving, WHITE);

        // Delay to slow down the speed of moving circle

        delay(50);

        // Update the angle for next position

        angle += 1;

        // Reset angle when a complete circle is completed

        if (angle >= 360)

            angle = 0;

        // Check if Enter key is pressed to exit

        if (kbhit()) {

            if (getch() == '\r')  // '\r' represents Enter key

                break;

        }

    }

    getch();

    closegraph();

    return 0;

}

34.3 circle outside circle

#include <graphics.h>

#include <dos.h>

#include <stdio.h>

#include <conio.h>

#include <math.h>

void brescir(int x, int y, int r, int color) {

    int i = 0, j = r, p = 3 - 2 \* r;

    while (i <= j) {

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

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

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

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

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

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

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

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

        if (p < 0) {

            i++;

            p = p + 4 \* i + 6;

        } else {

            i++;

            j--;

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

        }

    }

}

#define PI 3.14159265

int main()

{

    // Radius and center of the main circle

    int centerX = 150;

    int centerY = 150;

    int radius\_main = 100; // Radius of the main circle

    int radius\_moving = 8; // Radius of the moving circle

    // Initial angle for the moving circle

    float angle = 0;

    int gd = DETECT, gm, i, x, y;

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

    while (1)

    {

        // Clear the screen

        cleardevice();

        // Draw the main circle using Bresenham's algorithm

        brescir(centerX, centerY, radius\_main, WHITE);

        // Calculate position of moving circle

        x = centerX + (int)((radius\_main + radius\_moving) \* cos(angle \* PI / 180));

        y = centerY + (int)((radius\_main + radius\_moving) \* sin(angle \* PI / 180));

        // Draw the moving circle using Bresenham's algorithm

        brescir(x, y, radius\_moving, WHITE);

        // Delay to slow down the speed of moving circle

        delay(50);

        // Update the angle for next position

        angle += 1;

        // Reset angle when a complete circle is completed

        if (angle >= 360)

            angle = 0;

        // Check if Enter key is pressed to exit

        if (kbhit()) {

            if (getch() == '\r')  // '\r' represents Enter key

                break;

        }

    }

    getch();

    closegraph();

    return 0;

}

34.4 circle inside rectangle

#include <graphics.h>

#include <dos.h>

#include <stdio.h>

#include <conio.h>

#include <math.h>

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

    int dx = x2 - x1;

    int dy = y2 - y1;

    int steps, k;

    float xIncrement, yIncrement, x = x1, y = y1;

    if (abs(dx) > abs(dy))

        steps = abs(dx);

    else

        steps = abs(dy);

    xIncrement = dx / (float) steps;

    yIncrement = dy / (float) steps;

    for (k = 0; k < steps; k++) {

        x += xIncrement;

        y += yIncrement;

        if ((int)(x + 0.5) == x2 && (int)(y + 0.5) == y2) {

            break; // Stop drawing at the endpoint

        }

        putpixel((int)(x + 0.5), (int)(y + 0.5), WHITE);

    }

}

void brescir(int x, int y, int r, int color) {

    int i = 0, j = r, p = 3 - 2 \* r;

    while (i <= j) {

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

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

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

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

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

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

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

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

        if (p < 0) {

            i++;

            p = p + 4 \* i + 6;

        } else {

            i++;

            j--;

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

        }

    }

}

int main()

{

    int gd = DETECT, gm, i, x, y, j = 0;

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

    drawLine(50, 50, 250, 50);

    drawLine(250, 50, 250, 250);

    drawLine(250, 250, 50, 250);

    drawLine(50, 250, 50, 50);

    // Initial coordinates of ball

    x = 58;

    y = 242;

    while (1)

    {

        // Clear the screen

        cleardevice();

        // Draw the square

        drawLine(50, 50, 250, 50);

        drawLine(250, 50, 250, 250);

        drawLine(250, 250, 50, 250);

        drawLine(50, 250, 50, 50);

        // Draw the ball

    brescir(x, y,10, WHITE); // Corrected the function name and added the color parameter

        // Delay to slow down the speed of ball

        delay(50);

        // Check for Enter key press to exit

        if (kbhit() && getch() == 13) {

            break;

        }

        // Logic to move the ball in square path

        if (x <= 58 && y > 58)

            y = y - 5;

        if (x < 242 && y <= 58)

            x = x + 5;

        if (x >= 242 && y < 242)

            y = y + 5;

        if (x > 58 && y >= 242)

            x = x - 5;

        j++;

    }

    getch();

    closegraph();

    return 0;

}

34.8 circle wave

#include <graphics.h>

#include <dos.h>

#include <stdio.h>

#include <conio.h>

#include <math.h>

#define PI 3.14159265

void brescir(int x, int y, int r, int color) {

    int i = 0, j = r, p = 3 - 2 \* r;

    while (i <= j) {

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

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

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

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

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

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

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

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

        if (p < 0) {

            i++;

            p = p + 4 \* i + 6;

        } else {

            i++;

            j--;

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

        }

    }

}

int main()

{

    int centerX = 100; // Adjusted centerX for larger view

    int centerY = 200; // Adjusted centerY for larger view

    int amplitude = 100; // Increased amplitude for larger wave

    int wavelength = 200; // Increased wavelength for larger wave

    int i;

    float angle = 0;

    int gd = DETECT, gm, x, y;

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

    while (1)

    {

    cleardevice();

    // Draw the wave with two wavelengths

    for (i = 0; i < 4 \* wavelength; i++) { // Increased range to display larger wave

        int waveY = centerY + (int)(amplitude \* sin(i \* 2 \* PI / wavelength));

        putpixel(centerX + i, waveY, WHITE);

    }

    // Calculate position of moving brescir along wave pattern

    x = centerX + (int)(angle);

    y = centerY + (int)(amplitude \* sin(angle \* 2 \* PI / wavelength)) - 8;

    // Draw the moving brescir

    brescir(x, y, 10,WHITE);

    // Delay to control the speed of moving brescir

    delay(10);

    // Update the angle for next position

        angle += 0.1;

        // Check if Enter key is pressed to exit

        if (kbhit()) {

            if (getch() == '\r')  // '\r' represents Enter key

                break;

        }

    }

    getch();

    closegraph();

    return 0;

}

34.13 clock

#include <stdio.h>

#include <graphics.h>

#include <conio.h>

void brescir(int x, int y, int r, int color) {

    int i = 0, j = r, p = 3 - 2 \* r;

    while (i <= j) {

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

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

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

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

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

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

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

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

        if (p < 0) {

            i++;

            p = p + 4 \* i + 6;

        } else {

            i++;

            j--;

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

        }

    }

}

void main() {

    int gd = DETECT, gm, i, j = 96;

    float k = 90.5;

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

    brescir(320, 240, 105,WHITE);

    settextstyle(TRIPLEX\_FONT, HORIZ\_DIR, 3);

    outtextxy(245, 85, "     Clock");

    setfillstyle(SOLID\_FILL, 15);

    floodfill(320, 240, 15);

    while (kbhit() == 0) {

        k -= 0.1;

        j -= 6;

        for (i = 90; i >= 1; i -= 6) {

            if (kbhit() != 0) exit();

            setcolor(BLACK);

            setfillstyle(SOLID\_FILL, BLACK);

            pieslice(320, 240, i, i - 1, 80);

            pieslice(320, 240, j, j - 1, 65);

            pieslice(320, 240, k, k - 1, 50);

            delay(1000); // Adjusted delay for seconds line

            setcolor(WHITE);

            setfillstyle(SOLID\_FILL, WHITE);

            pieslice(320, 240, i, i - 1, 80);

            pieslice(320, 240, j, j - 1, 65);

            pieslice(320, 240, k, k - 1, 50);

        }

        for (i = 359; i >= 91; i -= 6) {

            if (kbhit() != 0) exit();

            setcolor(BLACK);

            setfillstyle(SOLID\_FILL, BLACK);

            pieslice(320, 240, i, i - 1, 80);

            pieslice(320, 240, j, j - 1, 65);

            pieslice(320, 240, k, k - 1, 50);

            delay(1000); // Adjusted delay for seconds line

            setcolor(WHITE);

            setfillstyle(SOLID\_FILL, WHITE);

            pieslice(320, 240, i, i - 1, 80);

            pieslice(320, 240, j, j - 1, 65);

            pieslice(320, 240, k, k - 1, 50);

        }

        if (j <= 6) j = 365;

        if (k <= 0.5) k = 359;

    }

    closegraph();

}