B.Sc(H) Computer Sc. Sem VI 2023-24

Computer Graphics Assignment

Name:- Abhishek Kumar

Roll No. :- 10814

Section:- A

**Q1.** Consider two raster systems with the resolutions of 640 x 480 and 1280 x 1024.

**(a)** How many pixels could be accessed per second in each of these systems by a display controller that refreshes the screen at a rate of 60 frames per second?

Ans:-

Given , screen A is of 640 x 480

Screen B is 1280 x 1024

And each has 60 fps refresh rate,

That means,

60 frames in ---> 1 second

60 x 640 x 480 pixels in. --> 1 second

So for Display1 display controller can access 18,432,000 pixels in 1 second

Now for Display2

We get,

60 x 1280 x 1024 ,

That is 76,643,200 pixels in 1 second is accessed by display controller.

**(b)** What is the access time per pixel in each system?

Ans:-

since , Display1 display controller can access 18,432,000 pixels in 1 second

therefore , the access time per pixel will be seconds.

now , Display2 display controller can access 76,643,200 pixels in 1 second

therefore , the access time per pixel will be seconds

**Q2.** Consider a raster system with the resolution of 1024 x 768 pixels and the color palette calls for 65,536 colors. What is the minimum amount of video RAM that the computer must have to support the above-mentioned resolution and number of colors?

Ans:-

since 2^16 = 65,536

therefore we require 16 bits for color palette per pixel

so MIMIMUM video RAM will be = bits per pixel x total pixels

that is, 16 x (1024 x 768) bits

= 1,572,864 bytes

= 1.5 MB

**Q3.** Modify the line drawing algorithm (DDA/Bresenhams) to draw a line i) of thickness t

Ans:-

#include<iostream>

#include<conio.h>

#include<math.h>

#include<graphics.h>

usingnamespace std;

void swap(int\*&p1 , int\*&p2){

int \*temp = p2;

p2 = p1;

p1 = temp;

}

double slope(int\*p1, int\*p2,int&flag){

if(p2[1] - p1[1] ==0){

flag =0;

return0;

}

if(p2[0] - p1[0]==0){

flag =1;

return0;

}

double m = double(double(p2[1] - p1[1] ) / double(p2[0] - p1[0]));

return m;

}

void draw\_h(int\*p1, int\*p2,intthickness){

if (p2[0]<p1[0])

{ swap(p1,p2);

}

int x1 = p1[0];

int x2 = p2[0];

int y= p1[1];

int t1 = thickness/2;

int t2 = thickness/2;

while(x1 < x2){

putpixel(x1,getmaxy()-y,255);

for(int i=1;i<=t1;i++){

putpixel(x1,getmaxy()-y+i,255);

}

for(int i=1;i<=t2;i++){

putpixel(x1,getmaxy()-y-i,255);

}

x1 +=1;

}

}

void draw\_v(int\*p1, int\*p2,intthickness){

if (p2[1]<p1[1])

{ swap(p1,p2);

}

int y1 = p1[1];

int y2 = p2[1];

int x= p1[0];

int t1 = thickness/2;

int t2 = thickness/2;

while(y1 < y2){

putpixel(x,getmaxy()-y1,255);

for(int i=1;i<=t1;i++){

putpixel(x-i,getmaxy()-y1,255);

}

for(int i=1;i<=t2;i++){

putpixel(x+i,getmaxy()-y1,255);

}

y1 +=1;

}

}

void dda(int\*p1, int\*p2, intthickness){

int flag =-1;

double m = slope(p1,p2,flag);

if(flag==0){

draw\_h(p1,p2,thickness);

return ;

}

if(flag==1){

draw\_v(p1,p2,thickness);

return ;

}

if( abs(m) >1){

cout<<"steepe slope"<<endl;

if (p2[1]<p1[1])

{ swap(p1,p2);

}

int y2 = p2[1];

int y = p1[1];

double x = p1[0];

for(;y<=y2; y++){

x = x + double(1.0/m);

putpixel(round(x),getmaxy()-y,255);

int t1 = thickness/2;

int t2 = thickness/2;

for(int i=1;i<=t1;i++){

putpixel(round(x)-i,getmaxy()-y,255);

}

for(int i=1;i<=t2;i++){

putpixel(round(x)+i,getmaxy()-y,255);

}

}

}else{

cout<<"gradual slope"<<endl;

if (p1[0]>p2[0])

{swap(p1,p2);

}

int x2 = p2[0];

int x = p1[0];

double y = p1[1];

for(;x<=x2; x++){

y = y + double(m);

int t1 = thickness/2;

int t2 = thickness/2;

int flag = 0;

putpixel(round(x),getmaxy()-y,255);

for(int i=1;i<=t1;i++){

putpixel(round(x),getmaxy()-y-i,255);

}

for(int i=1;i<=t2;i++){

putpixel(round(x),getmaxy()-y+i,255);

}

}

}

}

void print\_point(int\*p){

cout<<" x : "<<p[0]<<" y : "<<p[1]<<endl;

}

int main()

{

int a,b;

int \*p1 = newint(2);

int \*p2 = newint(2);

p1[0] = 300;

p1[1] = 150;

p2[0] = 200;

p2[1] = 400;

int gd = DETECT, gm;

char pathtodriver[] = "";

initgraph(&gd, &gm, pathtodriver);

print\_point(p1);

print\_point(p2);

int t =4;

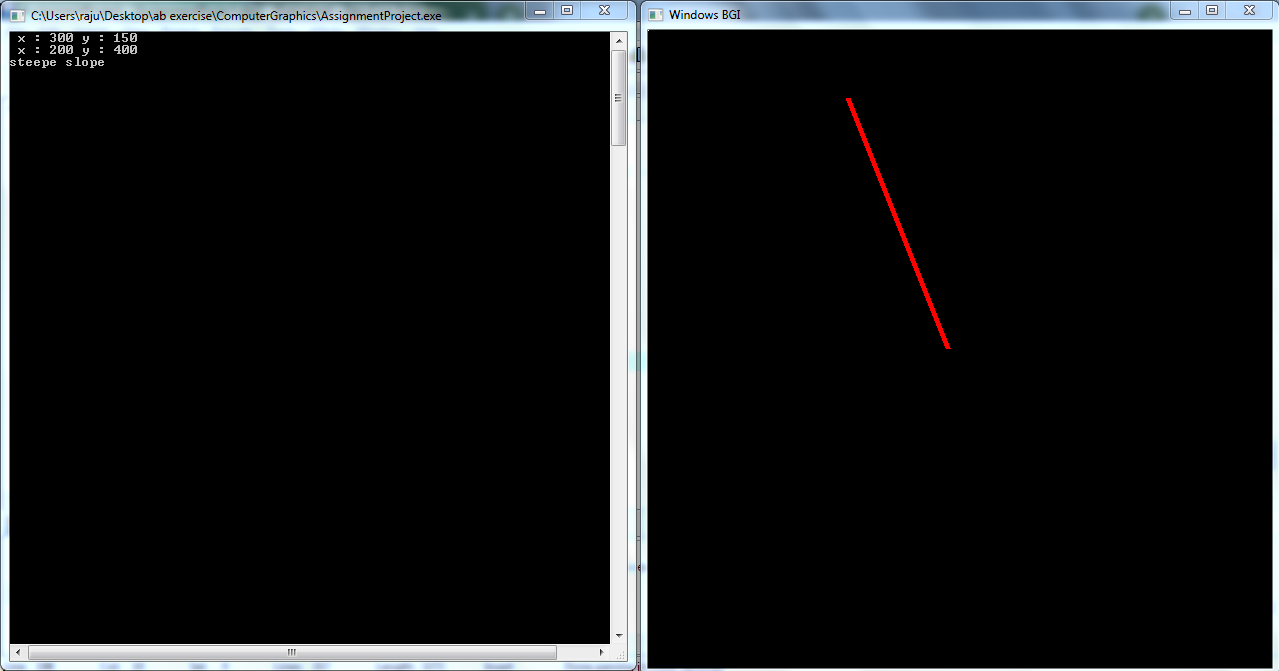
dda(p2,p1,t);

getch();

closegraph();

}

**OUTPUT**:-

****

ii) of a particular pattern( dotted, dashed, or any other)

SOURCE CODE FOR DASHED LINE :-

#include<iostream>

#include<conio.h>

#include<math.h>

#include<graphics.h>

Using namespace std;

void swap(int\*&p1 , int\*&p2){

int \*temp = p2;

p2 = p1;

p1 = temp;

}

double slope(int\*p1, int\*p2,int&flag){

if(p2[1] - p1[1] ==0){

flag =0;

return0;

}

if(p2[0] - p1[0]==0){

flag =1;

return0;

}

double m = double(double(p2[1] - p1[1] ) / double(p2[0] - p1[0]));

return m;

}

void draw\_h(int\*p1, int\*p2,int thickness, int dashed,int dashed\_space){

if (p2[0]<p1[0])

{ swap(p1,p2);

}

int x1 = p1[0];

int x2 = p2[0];

int y= p1[1];

int t1 = thickness/2;

int t2 = thickness/2;

int flag =dashed;

while(x1 < x2){

if(flag >0){

putpixel(x1,getmaxy()-y,255);

for(int i=1;i<=t1;i++){

putpixel(x1,getmaxy()-y+i,255);

}

for(int i=1;i<=t2;i++){

putpixel(x1,getmaxy()-y-i,255);

}

flag -=1;

if(flag==0){

flag = -1\*dashed\_space;

}

}else{

flag+=1;

if(flag==0){

flag = dashed;

}

}

x1 +=1;

}

}

void draw\_v(int\*p1, int\*p2,int thickness, int dashed, int dashed\_space){

if (p2[1]<p1[1])

{ swap(p1,p2);

}

int y1 = p1[1];

int y2 = p2[1];

int x= p1[0];

int t1 = thickness/2;

int t2 = thickness/2;

int flag =dashed;

while(y1 < y2){

if(flag >0){

putpixel(x,getmaxy()-y1,255);

for(int i=1;i<=t1;i++){

putpixel(x-i,getmaxy()-y1,255);

}

for(int i=1;i<=t2;i++){

putpixel(x+i,getmaxy()-y1,255);

}

flag -=1;

if(flag==0){

flag = -1\*dashed\_space;

}

}else{

flag+=1;

if(flag==0){

flag = dashed;

}

}

y1 +=1;

}

}

void dda(int\*p1, int\*p2, int thickness, int dashed, int dashed\_space){

int flag = dashed;

int t1 = thickness/2;

int t2 = thickness/2;

int flag\_x =-1;

double m = slope(p1,p2,flag\_x);

if(flag\_x==0){

draw\_h(p1,p2,thickness, dashed, dashed\_space);

return ;

}

if(flag\_x==1){

draw\_v(p1,p2,thickness, dashed, dashed\_space);

return ;

}

if( abs(m) >1){

cout<<"steepe slope"<<endl;

if (p2[1]<p1[1])

{ swap(p1,p2);

}

int y2 = p2[1];

int y = p1[1];

double x = p1[0];

for(;y<=y2; y++){

x = x + double(1.0/m);

if(flag >0){

putpixel(round(x),getmaxy()-y,255);

for(int i=1;i<=t1;i++){

putpixel(round(x)-i,getmaxy()-y,255);

}

for(int i=1;i<=t2;i++){

putpixel(round(x)+i,getmaxy()-y,255);

}

flag -=1;

if(flag==0){

flag = -1\*dashed\_space;

}

}else{

flag+=1;

if(flag==0){

flag = dashed;

}

}

}

}else{

cout<<"gradual slope"<<endl;

if (p1[0]>p2[0])

{swap(p1,p2);

}

int x2 = p2[0];

int x = p1[0];

double y = p1[1];

int flag = dashed;

for(;x<=x2; x++){

y = y + double(m);

if(flag >0){

putpixel(round(x),getmaxy()-y,255);

for(int i=1;i<=t1;i++){

putpixel(round(x)-i,getmaxy()-y,255);

}

for(int i=1;i<=t2;i++){

putpixel(round(x)+i,getmaxy()-y,255);

}

flag -=1;

if(flag==0){

flag = -1\* dashed\_space;

}

}else{

flag+=1;

if(flag==0){

flag = dashed;

}

}

}

}

}

void print\_point(int\*p){

cout<<" x : "<<p[0]<<" y : "<<p[1]<<endl;

}

int main()

{

int a,b;

int \*p1 = new int(2);

int \*p2 = new int(2);

p1[0] = 300;

p1[1] = 300;

p2[0] = 550;

p2[1] = 300;

int gd = DETECT, gm;

char pathtodriver[] = "";

initgraph(&gd, &gm, pathtodriver);

print\_point(p1);

print\_point(p2);

int t =6;

int dashed\_space\_size=10;

dda(p2,p1,t,6,6);

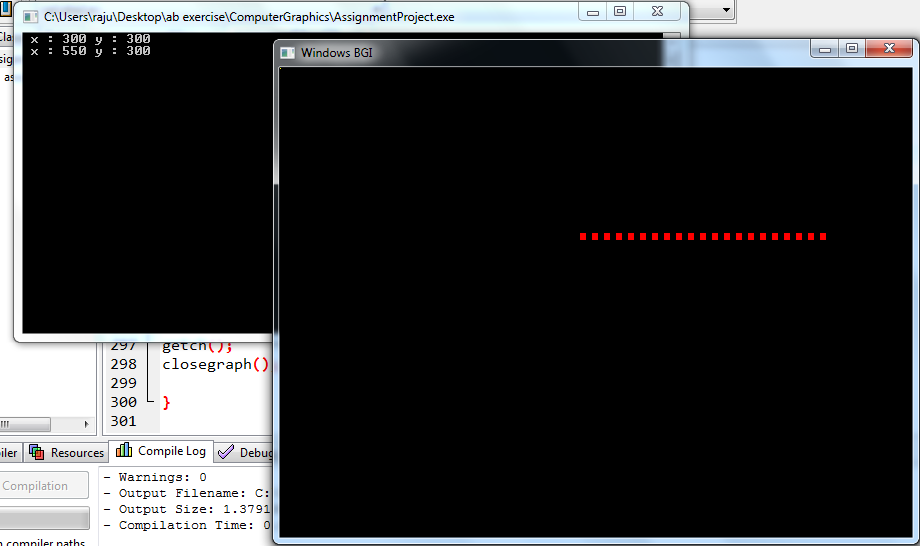
getch();

closegraph();

}

**OUTPUT**:-

**a.**This is representation of dashed line parallel to x-axis

  
  
**SOURCE CODE FOR DOTTED LINE :-**

#include<iostream>

#include<conio.h>

#include<math.h>

#include<graphics.h>

Using namespace std;

void swap(int\*&p1, int\*&p2)

{

int \*temp = p2;

p2 = p1;

p1 = temp;

}

double slope(int\*p1, int\*p2, int&flag)

{

if (p2[1] - p1[1] == 0)

{

flag = 0;

return0;

}

if (p2[0] - p1[0] == 0)

{

flag = 1;

return0;

}

double m = double(double(p2[1] - p1[1]) / double(p2[0] - p1[0]));

return m;

}

void draw\_h(int\*p1, int\*p2, intthickness, intdashed, intdashed\_space)

{

if (p2[0] <p1[0])

{

swap(p1, p2);

}

int x1 = p1[0];

int x2 = p2[0];

int y = p1[1];

int t1 = thickness / 2;

int t2 = thickness / 2;

int flag = dashed;

while (x1 < x2)

{

if (flag >0)

{

fillellipse(round(x1), getmaxy() - y, thickness, thickness);

flag -= 1;

if (flag == 0)

{

flag = -1 \* dashed\_space;

}

}

else

{

flag += 1;

if (flag == 0)

{

flag = dashed;

}

}

x1 += 1;

}

}

void draw\_v(int\*p1, int\*p2, intthickness, intdashed, intdashed\_space)

{

if (p2[1] <p1[1])

{

swap(p1, p2);

}

int y1 = p1[1];

int y2 = p2[1];

int x = p1[0];

int t1 = thickness / 2;

int t2 = thickness / 2;

int flag = dashed;

while (y1 < y2)

{

if (flag >0)

{

putpixel(x, getmaxy() - y1, 255);

fillellipse(round(x), getmaxy() - y1, thickness, thickness);

flag -= 1;

if (flag == 0)

{

flag = -1 \* dashed\_space;

}

}

else

{

flag += 1;

if (flag == 0)

{

flag = dashed;

}

}

y1 += 1;

}

}

void dda(int\*p1, int\*p2, intthickness, intdashed, intdashed\_space)

{

int flag = dashed;

int t1 = thickness / 2;

int t2 = thickness / 2;

int flag\_x = -1;

double m = slope(p1, p2, flag\_x);

if (flag\_x == 0)

{

draw\_h(p1, p2, thickness, dashed, dashed\_space);

return;

}

if (flag\_x == 1)

{

draw\_v(p1, p2, thickness, dashed, dashed\_space);

return;

}

if (abs(m) >1)

{

cout <<"steepe slope"<< endl;

if (p2[1] <p1[1])

{

swap(p1, p2);

}

int y2 = p2[1];

int y = p1[1];

double x = p1[0];

for (; y <= y2; y++)

{

x = x + double(1.0 / m);

if (flag >0)

{

fillellipse(round(x), getmaxy() - y, thickness, thickness);

flag -= 1;

if (flag == 0)

{

flag = -1 \* dashed\_space;

}

}

else

{

flag += 1;

if (flag == 0)

{

flag = dashed;

}

}

}

}

else

{

cout <<"gradual slope"<< endl;

if (p1[0] >p2[0])

{

swap(p1, p2);

}

int x2 = p2[0];

int x = p1[0];

double y = p1[1];

int flag = dashed;

for (; x <= x2; x++)

{

y = y + double(m);

if (flag >0)

{

putpixel(round(x), getmaxy() - y, 255);

fillellipse(round(x), getmaxy() - y, thickness, thickness);

flag -= 1;

if (flag == 0)

{

flag = -1 \* dashed\_space;

}

}

else

{

flag += 1;

if (flag == 0)

{

flag = dashed;

}

}

}

}

}

void print\_point(int\*p)

{

cout <<" x : "<<p[0] <<" y : "<<p[1] << endl;

}

int main()

{

int a, b;

int \*p1 = newint(2);

int \*p2 = newint(2);

p1[0] = 300;

p1[1] = 300;

p2[0] = 550;

p2[1] = 300;

int gd = DETECT, gm;

char pathtodriver[] = "";

initgraph(&gd, &gm, pathtodriver);

print\_point(p1);

print\_point(p2);

int t = 5;

int dashed\_space\_size = 10;

dda(p2, p1, t, 1, 4);

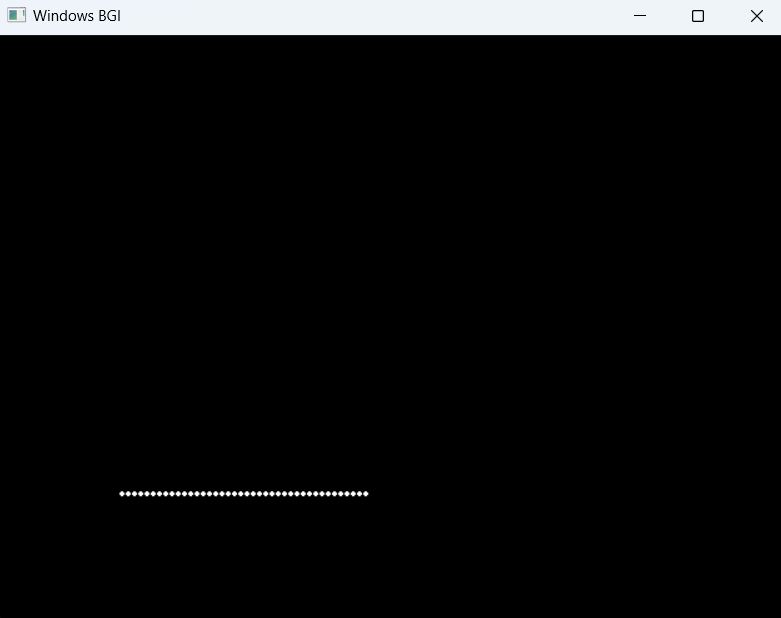
getch();

closegraph();

}

**OUTPUT**:-

**a.**This is representation of dotted line parallel to x-axis



**Q4.** Use line drawing and circle drawing algorithm to draw a circle divided into 8 sectors as shown below:

Ans:-

#include<iostream>

#include<graphics.h>

#include<conio.h>

#include<math.h>

#include"circle.cpp"

#include"p3.cpp"

#include<cstdlib>

usingnamespace std;

#defineN2

void multiply(doublearr1[][N], intarr2[N][1], int\*\*arr3)

{

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

{

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

{

arr3[i][j] = 0;

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

{

arr3[i][j] += arr1[i][k] \* arr2[k][j];

}

}

}

}

void zero\_matrix(intarr3[N][1])

{

arr3[0][0] = 0;

arr3[1][0] = 0;

}

void draw\_lines(intr, int\*p, int\*p1,int\*p2,int flag )

{

flag--;

if(flag<=0){

return;

}

double arr1[N][N] = {

{0.707, -0.707},

{0.707, 0.707},

};

int arr2[N][1] = {{p1[0] - p[0]}, {(p1[1] - p[0])}};

int arr22[N][1] = {{p2[0] - p[0]}, {(p2[1] - p[0])}};

int \*\*arr3 = new int\*[2];

for(int i=0;i<N;i++){

arr3[i] = new int[1];

}

multiply(arr1, arr2, arr3);

int \*\*arr33 = new int\*[2];

for(int i=0;i<N;i++){

arr33[i] = new int[1];

}

cout << arr33[1][0] <<" ,check tt first "<< arr33[1][0] << endl;

multiply(arr1, arr22, arr33);

cout << arr33[1][0] <<" ,check tt "<< arr33[1][0] << endl;

int \*i = new int(2);

int \*j = new int(2);

i[0] = abs(arr33[0][0] + p[0]);

i[1] = abs(arr33[1][0] + p[1]);

j[0] = abs(arr3[0][0] + p[0]);

j[1] = abs(arr3[1][0] + p[1]);

cout << i[0] <<" , "<< i[1] << endl;

cout << j[0] <<" , "<< j[1] << endl;

dda(j, i);

draw\_lines(r,p,j,i,flag);

}

int main()

{

int gd = DETECT, gm;

char pathtodriver[] = "";

initgraph(&gd, &gm, pathtodriver);

int \*p = newint(2);

int r = 50;

p[0] = 100;

p[1] = 100;

circle(p, r);

int \*p1 = newint(2);

int \*p2 = newint(2);

p1[0] = p[0];

p1[1] = p[1] + r;

p2[0] = p[0];

p2[1] = p[1] - r;

int f =5;

draw\_lines(r, p,p1,p2,f);

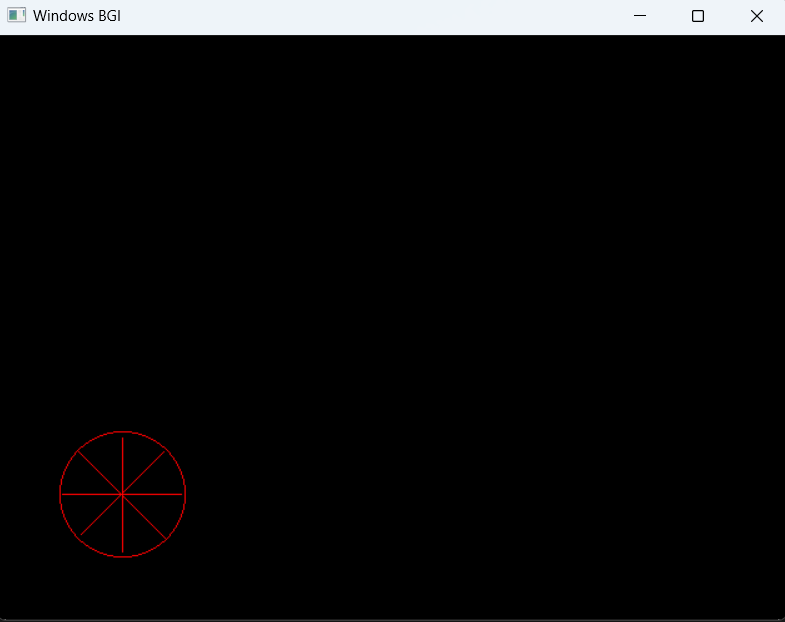
getch();

closegraph();

return0;

}

**OUTPUT**:-



**Q5.** Consider a square ABCD of size 2 (side is 2 units) positioned at origin. Transform it by applying following transformations one by one:

i) Rotate it w.r.t the centre of the square by an angle of 45 degrees in clockwise direction.

Ans :-

#include<iostream>

#include<conio.h>

#include<math.h>

#include<cmath>

#include"p3.cpp"

#include<graphics.h>

usingnamespace std;

#defineM\_PI3.14159265358979323846

void matrixMultiplication(double\*\*mat1, introws1, intcols1,

double\*\*mat2, introws2, intcols2,

double\*\*result) {

if (cols1 != rows2) {

std::cerr <<"Error: Matrix dimensions \n";

return;

}

for (int i = 0; i <rows1; ++i) {

for (int j = 0; j <cols2; ++j) {

result[i][j] = 0;

for (int k = 0; k <cols1; ++k) {

result[i][j] += mat1[i][k] \* mat2[k][j];

}

}

}

}

double degreesToRadians(doubledegrees) {

returndegrees \* (M\_PI / 180);

}

void create\_rotation\_matrix(double\*\*arr ,doubledegrees ){

arr[0][0] = 0.707;

arr[0][1] = 0.707;

arr[1][0] = -0.707;

arr[1][1] = 0.707;

arr[2][2] =1.0;

}

void print\_sq\_arr(double\*\*arr ,introw, intcol){

for(int i=0;i<row;i++){

for(int j=0;j<col;j++){

cout<<arr[i][j]<<" , ";

}

cout<<endl;

}

}

double\*\*make\_hom\_2d\_array\_zero(introw,intcol){

double \*\*arr = newdouble\*[row];

for(int i=0;i<row;i++){

arr[i] = newdouble[col];

}

for(int i=0;i<row;i++){

for(int j=0;j<col;j++){

arr[i][j]=0.0;

}

}

return&\*arr;

}

double\*\*make\_hom\_2d\_array\_ones(introw,intcol){

double \*\*arr = newdouble\*[row];

for(int i=0;i<row;i++){

arr[i] = newdouble[col];

}

for(int i=0;i<row;i++){

for(int j=0;j<col;j++){

arr[i][j]=1.0;

}

}

return&\*arr;

}

void draw\_square(double\*\*arr,intr,intc){

dda(arr[0][0], arr[0][1], arr[1][0], arr[1][1]);

dda( arr[2][0], arr[2][1],arr[1][0], arr[1][1]);

dda(arr[2][0], arr[2][1], arr[3][0], arr[3][1]);

dda( arr[0][0], arr[0][1],arr[3][0], arr[3][1]);

}

void subtract\_from\_each(double\*\*arr,introw, intcol,doublex,doubley){

for(int i=0;i<row;i++){

arr[i][0] = arr[i][0] - x;

arr[i][1] = arr[i][1] - y;

}

}

void add\_from\_each(double\*\*arr,introw, intcol,doublex,doubley){

for(int i=0;i<row;i++){

arr[i][0] = arr[i][0] + x;

arr[i][1] = arr[i][1] + y;

}

}

void find\_centroid(double\*\*arr, introw,intcol,double\*point){

double x=0.0;

double y=0.0;

for(int i=0;i<row;i++){

x += arr[i][0];

y += arr[i][1];

}cout<<endl;

x = x/4.0;

y = y/4.0;

point[0] = x;

point[1] = y;

}

void rotate\_square(double\*\*arr,introw,intcol){

double \*temp = newdouble(2);

temp[0] = 0.0;

temp[1] = 0.0;

find\_centroid(arr,row,col,temp);

cout<<"centroid : ";

cout<<temp[0]<<" , "<<temp[1]<<endl;

subtract\_from\_each(arr,row,col,temp[0],temp[1]);

cout<<endl;

print\_sq\_arr(arr,row,col);

cout<<endl;

double \*\*res = make\_hom\_2d\_array\_zero(4,3);

double \*\*mat = make\_hom\_2d\_array\_zero(3,3);

create\_rotation\_matrix(mat,45);

cout<<"rotation matrix---------------------------"<<endl;

print\_sq\_arr(mat,3,3);

cout<<"---------------------------------------------"<<endl;

matrixMultiplication(arr,row,col,mat,3,3,res);

add\_from\_each(res,row,col,temp[0],temp[1]);

cout<<endl;

print\_sq\_arr(res,row,col);

draw\_square(res,row,col);

}

int main()

{

int gd = DETECT, gm;

char pathtodriver[] = "";

initgraph(&gd, &gm, pathtodriver);

double \*\*mat = make\_hom\_2d\_array\_zero(3,3);

print\_sq\_arr(mat,3,3);

int row=4;

int col=3;

double \*\*arr = make\_hom\_2d\_array\_ones(4,3);

arr[0][0] = 100;

arr[1][0] = 200;

arr[2][0] = 200;

arr[3][0] =100;

arr[0][1] =100;

arr[1][1] =100;

arr[2][1] =200;

arr[3][1] =200;

draw\_square(arr,4,3);

rotate\_square(arr,row,col);

getch();

closegraph();

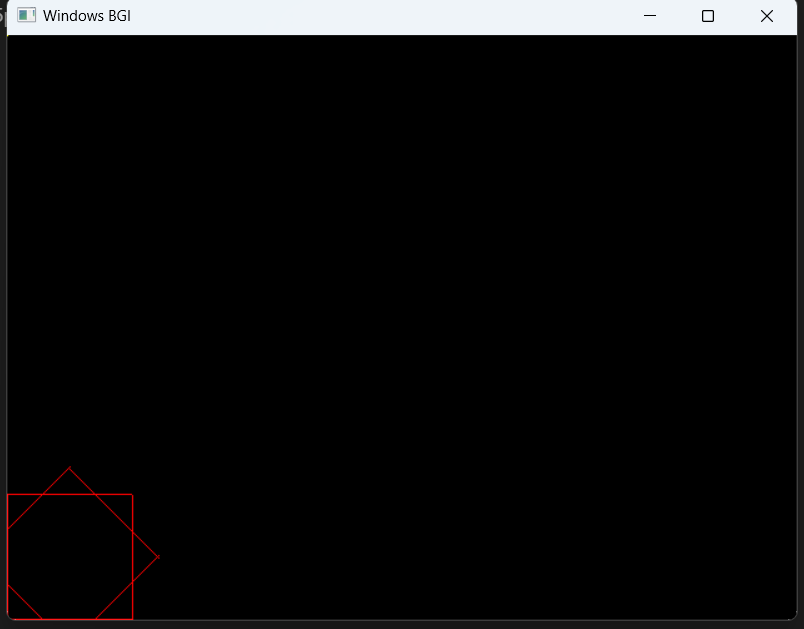
return 0;

}

Note :- for better visibility , here I have used square of side length =100 pixel

And its centroid of square is at (50,50)

**OUTPUT**:-



ii) Reflect it about a line y = 2x + 3.

Ans:-

#include<iostream>

#include<conio.h>

#include<math.h>

#include"p3.cpp"

#include<graphics.h>

Using namespace std;

void matrixMultiplication(double\*\*mat1, introws1, intcols1,

double\*\*mat2, introws2, intcols2,

double\*\*result) {

if (cols1 != rows2) {

std::cerr <<"Error: Matrix dimensions \n";

return;

}

for (int i = 0; i <rows1; ++i) {

for (int j = 0; j <cols2; ++j) {

result[i][j] = 0;

for (int k = 0; k <cols1; ++k) {

result[i][j] += mat1[i][k] \* mat2[k][j];

}

}

}

}

void print\_sq\_arr(double\*\*arr ,introw, intcol){

for(int i=0;i<row;i++){

for(int j=0;j<col;j++){

cout<<arr[i][j]<<" , ";

}

cout<<endl;

}

}

double\*\*make\_hom\_2d\_array\_zero(introw,intcol){

double \*\*arr = newdouble\*[row];

for(int i=0;i<row;i++){

arr[i] = newdouble[col];

}

for(int i=0;i<row;i++){

for(int j=0;j<col;j++){

arr[i][j]=0.0;

}

}

return&\*arr;

}

double\*\*make\_hom\_2d\_array\_ones(introw,intcol){

double \*\*arr = newdouble\*[row];

for(int i=0;i<row;i++){

arr[i] = newdouble[col];

}

for(int i=0;i<row;i++){

for(int j=0;j<col;j++){

arr[i][j]=1.0;

}

}

return&\*arr;

}

void fill\_composite\_matrix(double\*\*arr,doublem, doublec){

arr[0][0] = (1 - pow(m,2) )/ (1 + pow(m,2));

arr[1][0] = (2\*m )/ (1 + pow(m,2));

arr[2][0] = (-2\*c\*m )/ (1 + pow(m,2));

arr[0][1] = (2\*m )/ (1 + pow(m,2));

arr[1][1] = ( pow(m,2) -1 )/ (1 + pow(m,2));

arr[2][1] = (2\*c )/ (1 + pow(m,2));

arr[2][2] = 1;

}

void draw\_square(double\*\*arr,intr,intc){

dda(arr[0][0], arr[0][1], arr[1][0], arr[1][1]);

dda( arr[2][0], arr[2][1],arr[1][0], arr[1][1]); //200, 100 200, 0

dda(arr[2][0], arr[2][1], arr[3][0], arr[3][1]);

dda( arr[0][0], arr[0][1],arr[3][0], arr[3][1]);

}

int main()

{

int gd = DETECT, gm;

char pathtodriver[] = "";

initgraph(&gd, &gm, pathtodriver);

double \*\*mat = make\_hom\_2d\_array\_zero(3,3);

print\_sq\_arr(mat,3,3);

double m =2.0;

double c =-300.0;

dda(0,200,300,200);

fill\_composite\_matrix(mat,m,c);

double \*\*arr = make\_hom\_2d\_array\_ones(4,3);

arr[0][0] = 100;

arr[1][0] = 200;

arr[2][0] = 200;

arr[3][0] =100;

arr[0][1] =0;

arr[1][1] =0;

arr[2][1] =100;

arr[3][1] =100;

draw\_square(arr,4,3);

double \*\*res = make\_hom\_2d\_array\_zero(4,3);

matrixMultiplication(arr,4,3,mat,3,3,res);

print\_sq\_arr(res,4,3);

draw\_square(res,4,3);

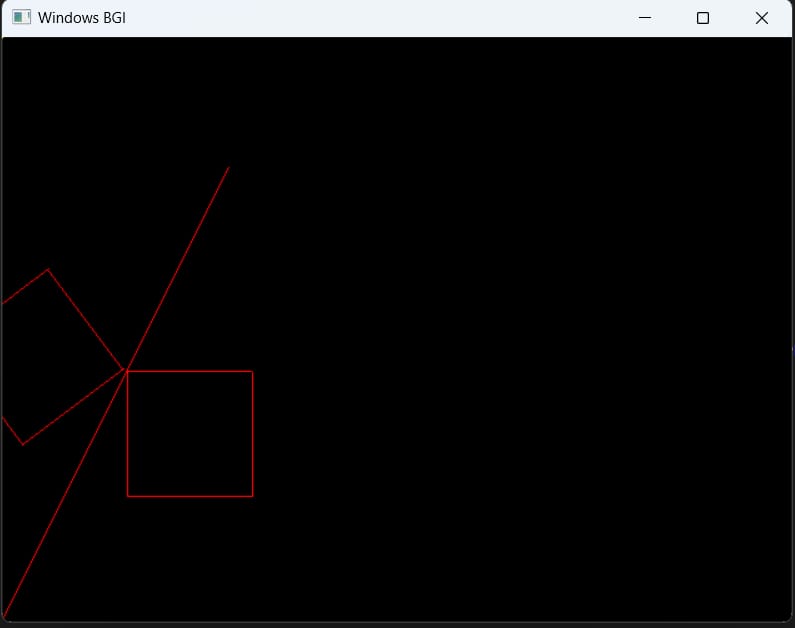
getch();

closegraph();

return 0;

}

**OUTPUT**:-



For better visibility , I used y = 200; around which a square is reflected

