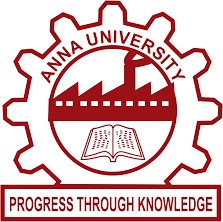
**MADRAS INSTITUTE OF TECHNOLOGY ANNA UNIVERSITY CHENNAI - 600044**



**DEPARTMENT OF INFORMATION TECHNOLOGY**

**IT7711** **GRAPHICS AND MULTIMEDIA LABORATORY**

**RECORD NOTE BOOK**

**7/8 B. TECH INFORMATION TECHNOLOGY**

**NAME**: **SANTHOSH Y**

**REGISTER NO**: **2018506107**

**SEMESTER**: **VII**

**YEAR**: **JUL 2021 - NOV 2021**

**MADRAS INSTITUTE OF TECHNOLOGY ANNA UNIVERSITY**

**CHENNAI - 600044.**

**BONAFIDE CERTIFICATE**

**NAME: SANTHOSH Y**

**REGISTER NO: 2018506107**

**SUBJECT: IT7711 GRAPHICS AND MULTIMEDIA LABORATORY**

**DEPARTMENT: INFORMATION TECHNOLOGY**

Certified that the bonafide record of Practical work done by **Santhosh Y** in the **Graphics and Multimedia Laboratory** subject code **IT7711** during the period July 2021 – November 2021.

**Date:** **Course-in-charge**

Submitted for the Practical Examination held on .

Internal Examiner External Examiner

**LIST OF EXPERIMENTS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.NO** | **DATE** | **TITLE** | **PAGE NO.** | **T. signature** |
| 1A |  |  |  |  |
| 1B |  |  |  |  |
| 1C |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |
| 5 |  |  |  |  |
| 6 |  |  |  |  |
| 7 |  |  |  |  |
| 8 |  |  |  |  |
| 9 |  |  |  |  |
| 10 |  |  |  |  |
| 11 |  |  |  |  |
| 12 |  |  |  |  |

**EXP NO: 1A DATE:**

**STUDY OF 2D PRIMITIVES IN GRAPHICS**

**AIM:**

To study about various 2D- graphical primitives involved in Computer graphics.

**DEFINITION:**

**GRAPHICAL PRIMITIVE:**

In graphics, primitives are basic elements, such as lines, curves, and polygons, which you can combine to create more complex graphical images. In programming, primitives are the basic operations supported by the programming language.

In reality, Computer graphics refer different things in different contexts: Pictures, scenes that are generated by a computer.

* Tools used to make such pictures, software and hardware, input/output devices.
* The whole field of study that involves these tools and the pictures they produce.

**TYPES OF 2D GRAPHICS:**

2D graphics come in two flavours

1. Raster - Raster graphics are the most common and are used for digital photos, Web graphics, icons, and other types of images. They are composed of a simple grid of pixels, which can each be a different colour.
2. Vector - Made up of paths, which may be lines, shapes, letters, or other scalable objects. They are often used for creating logos, signs, and other types of drawings. Unlike raster graphics, vector graphics can be scaled to a larger size without losing quality.

**APPLICATIONS OF COMPUTER:**

Graphics Pictures, Drawings, etc. Mathematical or Geometrical Models Computer Graphics Image Processing Overview of graphics system and output primitives Computer graphics is used today in many different areas of science, engineering, industry, business, education, entertainment, medicine, art and training. All of these are included in the following categories.

1. User interfaces
2. Plotting
3. Office automation and electronic publishing
4. Computer Aided Drafting and Design
5. Scientific and business Visualization
6. Simulation and modelling
7. Entertainment i.e. Disney movies such as Lion Kings and The Beauty of Beast, and other scientific movies like Jurassic Park, The lost world etc are the best example of the application of computer graphics in the field of entertainment.
8. Art and commerce
9. Cartography

**PRIMITIVES:**

A basic non-divisible graphical element for input or output within a computer graphics system.

**INPUT PRIMITIVE:**

* Typical output primitives are polyline, poly marker, and fill area. Clipping of an output primitive cannot be guaranteed to produce another output primitive.
* Typical input primitives are locator, choice, and valuator.
* Input primitives often have a style of echoing associated with them

**OUTPUT PRIMITIVES:**

* Output primitives are the geometric structures that has attributes such as straight line segments (pixel array) and patterns i.e.; polygon colour areas, used to describe the shapes and colours of the objects.
* Points and straight line segments are the simplest geometric components of pictures.
* Additional output primitive includes: circles and other conic sections, quadric surfaces, spline curves and surfaces, polygon colour areas and character strings. This includes picture generation algorithm by examining device-level algorithms for displaying two-dimensional output primitives, with emphasis on scan-conversion methods for raster graphics system.

**DIFFERENT 2D GRAPHICAL PRIMITIVES:**

**Point**

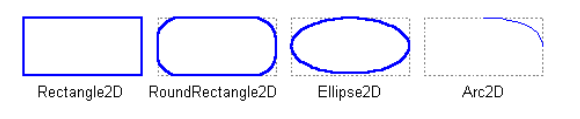
Point plotting is accomplished by converting a single coordinate position furnished by an application program into appropriate operations for the output device. With a CRT monitor, for example, the electron beam is turned on to illuminate the screen phosphor at the selected location.

**Line**

Line drawing is accomplished by calculating intermediate positions along the line path between two specified end points positions. An output device is then directed to fill in these positions between the end points.

**Rectangular Shapes**

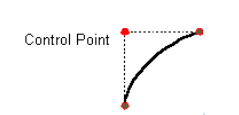
The Rectangle2D, RoundRectangle2D, Arc2D, and Ellipse2D primitives are all derived from the RectangularShape class. This class defines methods for Shape objects that can be described by a rectangular bounding box. The geometry of a Rectangular Shaped object can be extrapolated from a rectangle that completely encloses the outline of the Shape.



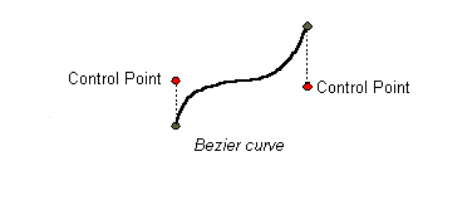
**Quadratic and cubic curves**

The QuadCurve2D enables you to create quadratic parametric curve segments. A quadratic curve is defined by two endpoints and one control point. The CubicCurve2D class enables you to create cubic parametric curve segments. A cubic curve is defined by two endpoints and two control points. The following are examples of quadratic and cubic curves. See Stroking and Filling Graphics Primitives for implementations of cubic and quadratic curves.

This figure represents a quadratic curve.

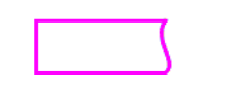


This figure represents a cubic curve.



**Arbitrary Shapes**

The GeneralPath class enables you to construct an arbitrary shape by specifying a series of positions along the shape’s boundary. These positions can be connected by line segments, quadratic curves, or cubic (Bézier) curves. The following shape can be created with three line segments and a cubic curve.



**Areas**

With the Area class, you can perform boolean operations, such as union, intersection, and subtraction, on any two Shape objects. This technique, often referred to as *constructive area geometry*, enables you to quickly create complex Shape objects without having to describe each line segment or curve.

**Line Drawing Algorithms**

* Digital Differential Analyzer (DDA) Algorithm
* Bresenham’s Line Algorithm
* Parallel Line Algorithm

**RESULT:**

Thus, the experiment has been successfully completed.

**EXP NO: 1B DATE:**

**2D PRIMITIVES INBUILT FUNCTIONS IN GRAPHICS**

**Aim**:

To study about various graphics functions in C.

**Functions with Syntax:**

* arc - it used to draw an arc with center (x, y) and stangle specifies starting angle, endangle specifies the end angle and last parameter specifies the radius of the arc. arc function can also be used to draw a circle but for that starting angle and end angle should be 0 and 360 respectively.

Declaration: void arc(int x, int y, int stangle, int endangle, int radius);

* bar - Bar function is used to draw a 2-dimensional, rectangular filled in bar . Coordinates of left top and right bottom corner are required to draw the bar. Left specifies the X-coordinate of top left corner, top specifies the Y-coordinate of top left corner, right specifies the X-coordinate of right bottom corner, bottom specifies the Y-coordinate of right bottom corner.

Declaration: void bar(int left, int top, int right, int bottom);

* bar3d - bar3d function is used to draw a 2-dimensional, rectangular filled in bar. Coordinates of left top and right bottom corner of bar are required to draw the bar. left specifies the X-coordinate of top left corner, top specifies the Y-coordinate of top left corner, right specifies the X-coordinate of right bottom corner, bottom specifies the Y-coordinate of right bottom corner, depth specifies the depth of bar in pixels, topflag determines whether a 3-dimensional top is put on the bar or not (if it's non-zero then it's put otherwise not).

Declaration: void bar3d(int left, int top, int right, int bottom, int depth, int topflag);

* circle - Circle function is used to draw a circle with center (x,y) and third parameter specifies the radius of the circle.

Declaration: void circle(int x, int y, int radius);

* cleardevice - cleardevice function clears the screen in graphics mode and sets the current position to (0,0). Clearing the screen consists of filling the screen with current background color.

Declaration: void cleardevice();

* closegraph - closegraph function closes the graphics mode, deallocates all memory allocated by graphics system and restores the screen to the mode it was in before you called initgraph.

Declaration: void closegraph();

* drawpoly - Drawpoly function is used to draw polygons i.e. triangle, rectangle, pentagon, hexagon etc.

Declaration: void drawpoly( int num, int \*polypoints );

* Ellipse - Ellipse is used to draw an ellipse (x,y) are coordinates of center of the ellipse, stangle is the starting angle, end angle is the ending angle, and fifth and sixth parameters specifies the X and Y radius of the ellipse.

Declarations: void ellipse(int x, int y, int stangle, int endangle, int xradius, int yradius);

* fillellipse - x and y are coordinates of center of the ellipse, xradius and yradius are x and y radius of ellipse respectively.

Declaration: void fillellipse(int x, int y, int xradius, int yradius);

* fillpoly - Fillpoly function draws and fills a polygon. It requires same arguments as drawpoly.

Declaration: void drawpoly(int num, int \*polypoints);

* floodfill - floodfill function is used to fill an enclosed area.

Declaration: void floodfill(int x, int y, int border);

* getarccords - getarccoords function is used to get coordinates of arc which is drawn most recently.

Declaration: void getarccoords(struct arccoordstype \*var);

* getbkcolor - Function getbkcolor returns the current background-color.

Declaration: int getbkcolor()

* getcolor - getcolor function returns the current drawing color.

Declaration: int getcolor();

* getdrivername - Function getdrivername returns a pointer to the current graphics driver.

Declaration: char \*name = getdrivername();

* getimage - getimage function saves a bit image of specified region into memory, region can be any rectangle.

Declaration: void getimage(int left, int top, int right, int bottom, void \*bitmap);

* getmaxcolor - getmaxcolor function returns maximum color value for current graphics mode and driver. Total number of colors available for current graphics mode and driver are ( getmaxcolor() + 1 ) as color numbering starts from zero.

Declaration: int getmaxcolor();

* getmaxx - getmaxx function returns the maximum X coordinate for current graphics mode and driver.

Declaration: int getmaxx();

* getmaxy - getmaxy function returns the maximum Y coordinate for current graphics mode and driver.

Declaration: int getmaxy();

* getpixel - getpixel function returns color of pixel present at location(x, y).

Declaration: int getpixel(int x, int y);

* getx - The function getx returns the X coordinate of the current position.

Declaration: int getx();

* gety - gety function returns the y coordinate of current position.

Declaration: int gety();

* graphdefaults - graphdefaults function resets all graphics settings to their defaults.

Declaration: void graphdefaults();

* grapherrormsg - grapherrormsg function returns an error message string.

Declaration: char \*grapherrormsg( int errorcode );

* imagesize - imagesize function returns the number of bytes required to store a bitimage. This function is used when we are using getimage.

Declaration: unsigned int imagesize(int left, int top, int right, int bottom);

* line - line function is used to draw a line from a point(x1,y1) to point(x2,y2) i.e. (x1,y1) and (x2,y2) are end points of the line.

Declaration: void line(int x1, int y1, int x2, int y2);

* moveto -moveto function changes the current position (CP) to (x, y)

Declaration: void moveto(int x, int y);

* moverel - Function moverel moves the current position to a relative distance.

Declaration: void moverel(int x, int y);

* outtext - outtext function displays text at current position.

Declaration: void outtext(char \*string);

* outtextxy - outtextxy function display text or string at a specified point(x,y) on the screen.

Declaration: void outtextxy(int x, int y, char \*string);

* putimage - putimage function outputs a bit image onto the screen.

Declaration: void putimage(int left, int top, void \*ptr, int op);

* putpixel - putpixel function plots a pixel at location (x, y) of specified color.

Declaration: void putpixel(int x, int y, int color);

* Rectangle - rectangle function is used to draw a rectangle. Coordinates of left top and right bottom corner are required to draw the rectangle. left specifies the X-coordinate of top left corner, top specifies the Y-coordinate of top left corner, right specifies the X-coordinate of right bottom corner, bottom specifies the Y-coordinate of right bottom corner.

Declaration: void rectangle(int left, int top, int right, int bottom);

* Sector -Sector function draws and fills an elliptical pie slice.

Declaration: void sector (int x, int y, int stangle, int endangle, int xradius, int yradius);

* setbkcolor - setbkcolor function changes current background color

Declaration: void setbkcolor(int color);

* setcolor – setcolor function is used to change the current drawing color.

Declaration: void setcolor(int color);

* setfillstyle -setfillstyle function sets the current fill pattern and fill color.

Declaration: void setfillstyle( int pattern, int color);

* setlinestyle – used to declare style of a line

Declaration: void setlinestyle(int linestyle, unsigned pattern, int thickness);

* settextstyle -Settextstyle function is used to change the way in which text appears, using it we can modify the size of text, change direction of text and change the font of text.

Declaration: void settextstyle( int font, int direction, int charsize);

* setviewport -setviewport function sets the current viewport for graphics output.

Declaration: void setviewport(int left, int top, int right, int bottom, int clip);

* textheight -textheight function returns the height of a string in pixels.

Declaration: int textheight(char \*string);

* textwidth -Textwidth function returns the width of a string in pixels.

Declaration: int textwidth(char \*string)

**Result**:

Thus, the experiment has been successfully completed and various graphics functions has been studied.

**EXP NO: 1C DATE:**

**SIMPLE GRAPHICS PROGRAM IN C**

**Aim**:

To write simple graphics program in C using basic graphics.h library functions.

**Source Code:**

1. #include<graphics.h>

#include<conio.h>

main(){

int gd=DETECT,gm;

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

setbkcolor(BLACK);

line(50,40,190,40);

rectangle(70,115,215,165);

arc(120,200,180,0,30);

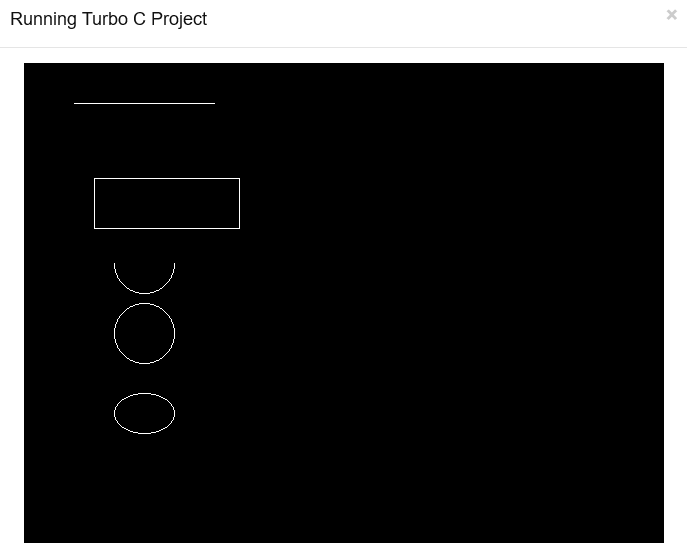
circle(120,270,30);

ellipse(120,350,0,360,30,20);

getch();

}

**Output:**



2. #include<graphics.h>

#include<conio.h>

#include<dos.h>

main()

{

int gd = DETECT, gm, x, y, color, angle = 0;

struct arccoordstype a, b;

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

delay(2000);

while(angle<=360)

{

setcolor(BLACK);

arc(getmaxx()/2,getmaxy()/2,angle,angle+2,100);

setcolor(BLUE);

getarccoords(&a);

circle(a.xstart,a.ystart,25);

setcolor(BLACK);

arc(getmaxx()/2,getmaxy()/2,angle,angle+2,150);

getarccoords(&a);

setcolor(YELLOW);

circle(a.xstart,a.ystart,25);

angle = angle+5;

delay(50);

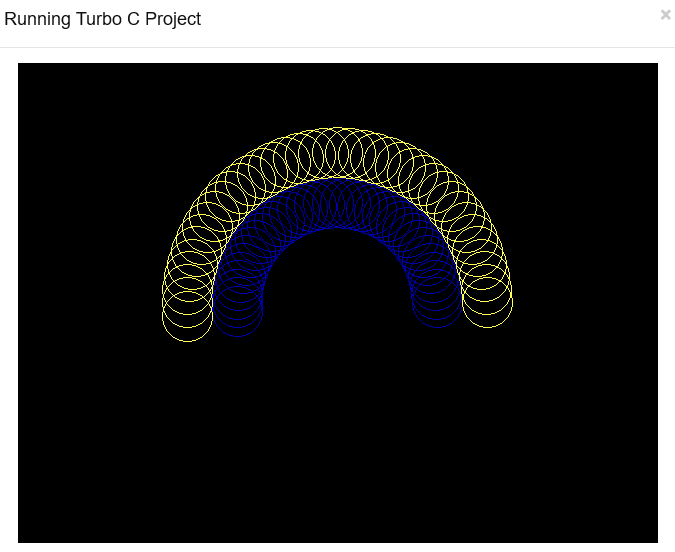
}

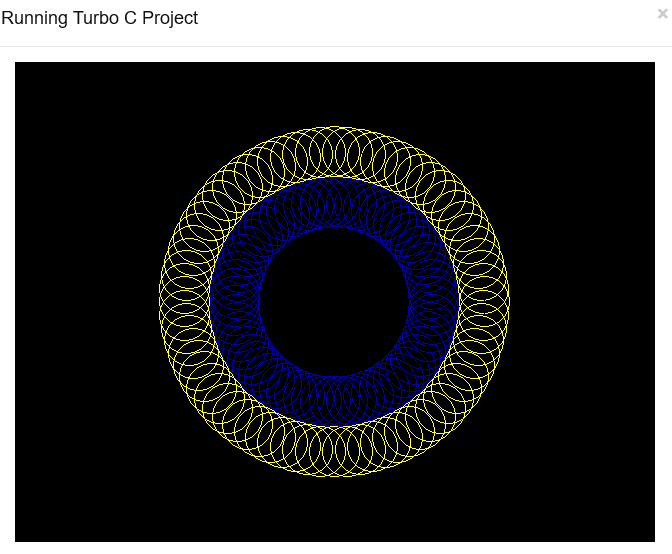
getch();

closegraph();

}

**Output:**





**Result**:

Thus, the experiment has been successfully completed and the output is verified

**EXP NO: 2 DATE:**

**DIGITAL DIFFERENTIAL ANALYZER ALGORITHM (DDA)**

**Aim**:

To write a program to implement line drawing Digital Differential Analyzer Algorithm (DDA) in C++.

**Algorithm:**

1. Get two endpoint pixel positions as Input.
2. Horizontal and vertical differences between the endpoint positions are assigned to parameters dx and dy. (Calculate dx=xb-xa and dy=yb-ya).
3. The difference with the greater magnitude determines the value of parameter steps.
4. Starting with pixel position (xa, ya), determine the offset needed at each step to generate the next pixel position along the line path.
5. Loop the following process for a number of times. Use a unit of increment or decrement in the x and y direction b. if xa is less than xb the values of increment in the x and y directions are 1 and m c. if xa is greater than xb then the decrements -1 and – m are used.

**Source Code:**

#include<graphics.h>

#include<conio.h>

#include<stdio.h>

main() {

int gd = DETECT ,gm, i;

float x, y,dx,dy,steps;

int x0, x1, y0, y1;

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

setbkcolor(WHITE);

x0 = 100 , y0 = 200, x1 = 500, y1 = 300;

dx = (float)(x1 - x0);

dy = (float)(y1 - y0);

if(dx>=dy){

steps = dx; }

else{

steps = dy; }

dx = dx/steps;

dy = dy/steps;

x = x0; y = y0; i = 1;

while(i<= steps){

putpixel(x, y, BLUE);

x += dx; y += dy;

i=i+1;

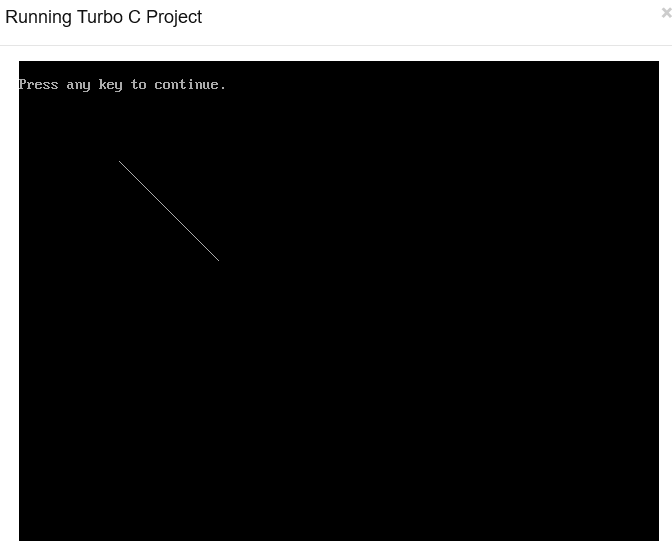
}

getch();

closegraph();

}

**Output:**



**Result:**

Thus, the experiment has been successfully completed and the output is verified.

**EXP NO: 3 DATE:**

**BRESENHAM’S LINE DRAWING ALGORITHM**

**Aim**:

To write a program to implement line drawing Bresenham’s Algorithm in C++.

**Algorithm:**

1. Input the two line endpoints and store the left end point in (x0, y0).
2. Load (x0, y0) into frame buffer, ie. Plot the first point.
3. Calculate the constants ∆x, ∆y, 2∆y and obtain the starting value for the decision parameter as

P0 = 2∆y-∆x

1. At each xk along the line, starting at k=0 perform the following test If Pk < 0, the next point to plot is (xk+1, yk) and Pk+1 = Pk + 2∆y otherwise, the next point to plot is (xk+1, yk+1) and Pk+1 = Pk + 2∆y - 2∆x
2. Perform step4 ∆x times.

**Source Code:**

#include<iostream.h>

#include<graphics.h>

void drawline(int x0, int y0, int x1, int y1{

int dx, dy, p, x, y;

dx =x1-x0;

dy =y1-y0;

x=x0;

y=y0;

p=2\*dy-dx

while(x<x1){

if(p>=0){

putpixel(x,y,7);

y=y+1;

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

}

else{

putpixel(x,y,7);

p=p+2\*dy;

}

x=x+1;

}

}

int main(){

int gdriver=DETECT, gmode, error, x0=100, y0=100, x1=200, y1=200;

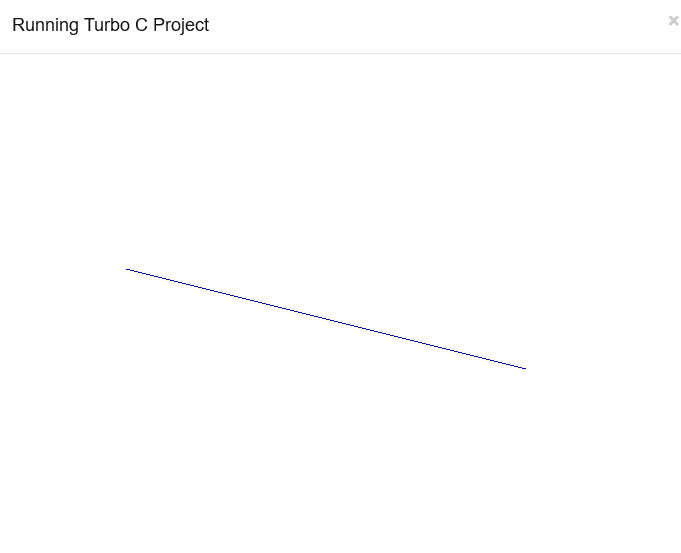
initgraph(&gdriver, &gmode, "c:\\tc\\bgi");

drawline(x0, y0, x1, y1);

return 0;

}

**Output:**

****

**Result:**

Thus, the experiment has been successfully completed and the output is verified.

**EXP NO: 4 DATE:**

**MID-POINT CIRCLE ALGORITHM**

**Aim**:

To implement Mid-Point Circle Algorithm in C++.

**Algorithm:**

1. Initially assume x =0, y =r

we have p =1-r

1. Repeat the steps while x ≤ y

Plot (x, y)

If (p<0)

Then have p = p + 2x + 3

Else

p = p + 2(x-y) +5

y =y - 1 (end if)

x =x+1 (end loop)

1. End

**Source Code:**

#include<graphics.h>

#include<stdio.h>

void pixel(int xc,int yc,int x,int y);

int main(){

int gd,gm,xc,yc,r,x,y,p;

detectgraph(&gd,&gm);

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

printf("Enter center of circle :");

scanf("%d%d",&xc,&yc);

printf("Enter radius of circle :");

scanf("%d",&r);

x=0;

y=r;

p=1-r;

pixel(xc,yc,x,y);

while(x<y)

{ if(p<0) {

x++;

p=p+2\*x+1;

}

else{

x++;

y--;

p=p+2\*(x-y)+1;

}

pixel(xc,yc,x,y);

}

getch();

closegraph();

return 0;

}

void pixel(int xc,int yc,int x,int 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);

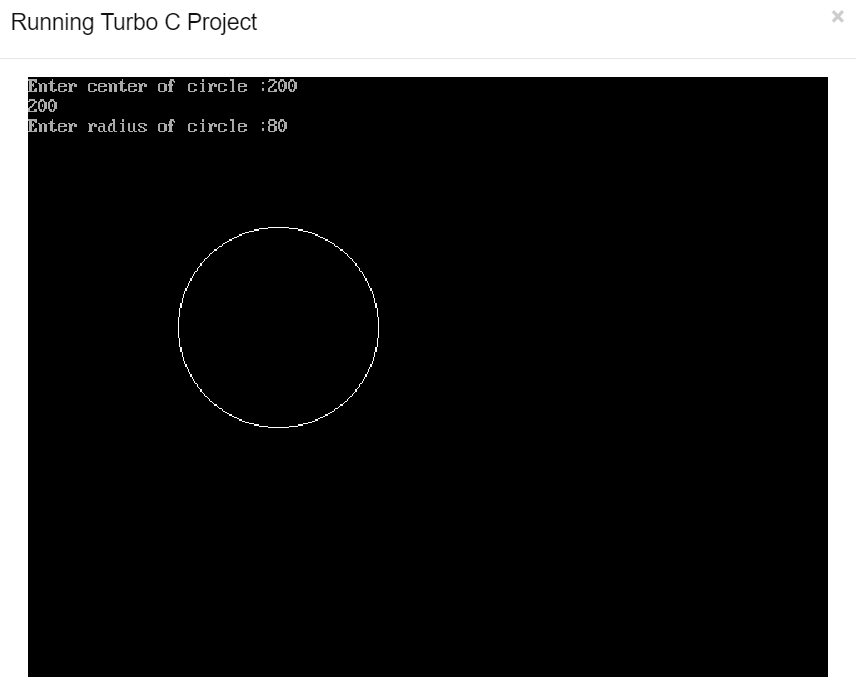
putpixel(xc+y,yc+x,WHITE);

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

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

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

}

**Output:**

**Result:**

Thus, the experiment has been successfully completed and the output is verified.

**EXP NO: 5 DATE:**

**MID-POINT ELLIPSE ALGORITHM**

**Aim**:

To implement Mid-Point Ellipse Algorithm in C++.

**Algorithm:**

1. Get input radius along x axis and y axis and find centre of ellipse.
2. We assume the centre of ellipse to be at origin and the first point as:

(x, y0) = (0, ry).

1. Find the initial decision parameter for region 1.
2. For every xk position in region 1:

If p1k<0 then the next point along the is (xk+1, yk)

Else, the next point is (xk+1, yk-1)

1. Obtain the initial value in region 2 using the last point (x0, y0) of region 1.
2. At each yk in region 2 starting at k =0 check the condition

If p2k>0 the next point is (xk, yk-1)

Else, the next point is (xk+1, yk -1)

1. Now obtain the symmetric points in the three quadrants and plot the coordinate value as: x=x+xc, y=y+yc
2. Repeat the steps for region 1 until 2ry2x & gt =2rx2y

**Source Code:**

#include <graphics.h>

#include <stdlib.h>

#include <math.h>

#include <stdio.h>

#include <conio.h>

main (){

float x,y,a, b,r,p,h,k,p1,p2;

h=319; k=239; a=50; b=40;

int gdriver = DETECT,gmode, errorcode;

int midx, midy, i;

initgraph (&gdriver, &gmode, "c:\\tc\\bgi");

x=0; y=b;

p1 = ((b \* b)-(a \* a \* b) + (a \* a)/4);

{

putpixel (x+h, y+k, RED);

putpixel (-x+h, -y+k, RED);

putpixel (x+h, -y+k, RED);

putpixel (-x+h, y+k, RED);

if (p1 < 0)

p1 += ((2 \*b \* b) \*(x+1))-((2 \* a \* a)\*(y-1)) + (b \* b);

else

{

p1+= ((2 \*b \* b) \*(x+1))-((2 \* a \* a)\*(y-1))-(b \* b);

y--;

}

x++;

}

p2 =((b \* b)\* (x + 0.5))+((a \* a)\*(y-1) \* (y-1))-(a \* a \*b \* b);

while (y>=0){

if (p2>0)

p2=p2-((2 \* a \* a)\* (y-1))+(a \*a);

else{

p2=p2-((2 \* a \* a)\* (y-1))+((2 \* b \* b)\*(x+1))+(a \* a);

x++;

}

y--;

putpixel (x+h, y+k, RED);

putpixel (-x+h, -y+k, RED);

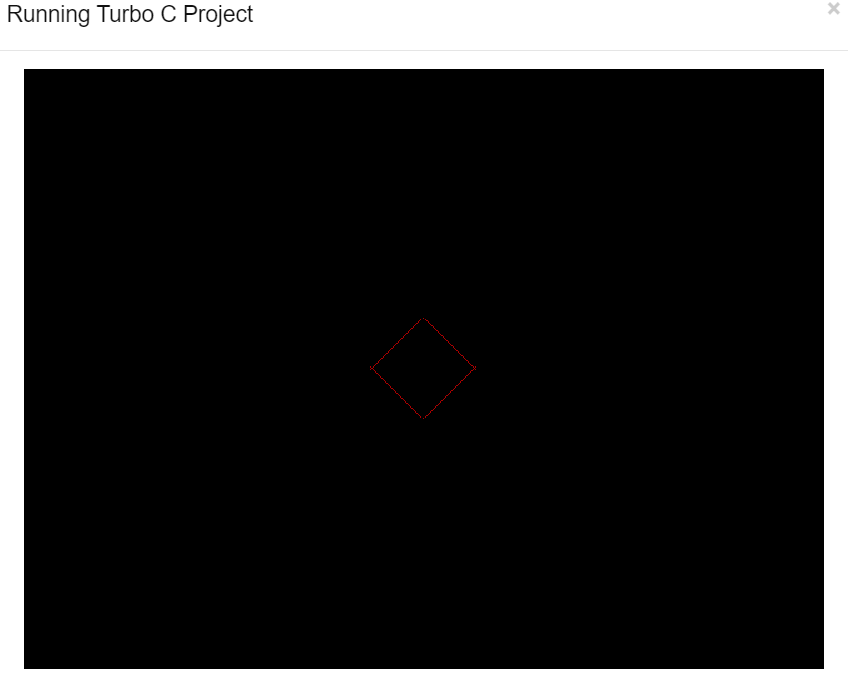
putpixel (x+h, -y+k, RED);

putpixel (-x+h, y+k, RED);

}

getch(); }

**Output:**



**Result**:

Thus, the experiment has been successfully completed and the output is verified.

**EXP NO: 6 DATE:**

**WINDOW TO VIEWPORT TRANSFORMATION**

**Aim**:

To implement window to viewport transformation in C.

**Source Code:**

#include <graphics.h>

#include<conio.h>

#include <stdio.h>

main(){

int wxmax,wymax,wxmin,wymin;

int vxmax,vymax,vxmin,vymin;

float sx,sy;

int x,x1,x2,y,y1,y2;

int gr=DETECT ,gm;

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

printf("Enter coordinates for triangle x,y");

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

printf("\n x1 and y1");

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

printf("\n x2 and y2");

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

printf("Enter window coordinates for triangle wxmax,wymax");

scanf("%d %d",&wxmax,&wymax);

printf("Enter window coordinates for triangle wxmin,wymin");

scanf("%d %d",&wxmin,&wymin);

cleardevice();

delay(50);

rectangle(wxmin,wymin,wxmax,wymax);

outtextxy(wxmin,wymin-10,"Window");

line(x,y,x1,y1);

line(x1,y1, x2,y2);

line(x2,y2,x,y);

vxmin=300; vymin=30; vxmax=550; vymax=350;

rectangle(vxmin,vymin,vxmax,vymax);

outtextxy(vxmin,vymin-10,"Viewport");

sx=(float)(vxmax-vxmin)/(wxmax-wxmin);

sy=(float)(vymax-vymin)/(wymax-wymin);

x=vxmin+(float)((x-wxmin)\*sx);

y=vymin+(float)((y-wymin)\*sy);

x1=vxmin+(float)((x1-wxmin)\*sx);

x2=vxmin+(float)((x2-wxmin)\*sx);

y1=vymin+(float)((y1-wymin)\*sx);

y2=vymin+(float)((y2-wymin)\*sx);

line(x,y,x1,y1);

line(x1,y1, x2,y2);

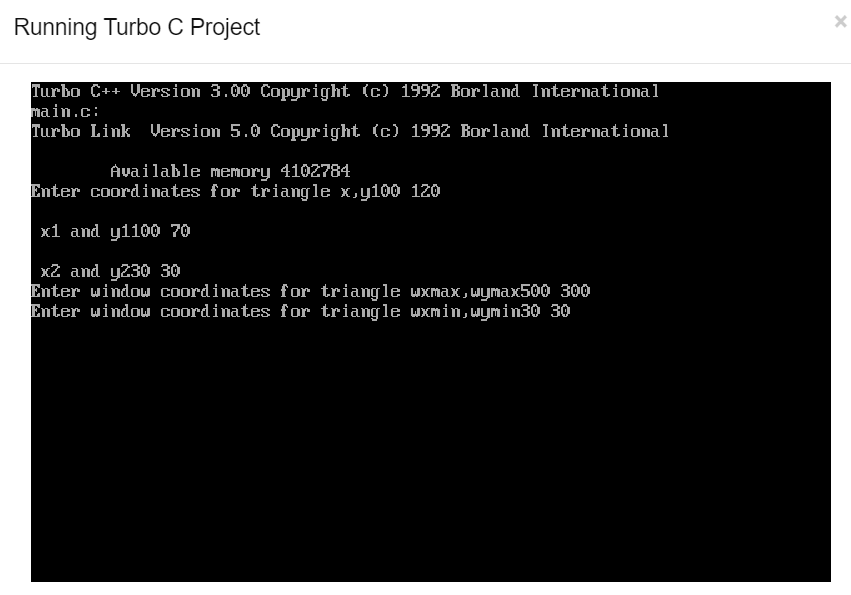
line(x2,y2,x,y);

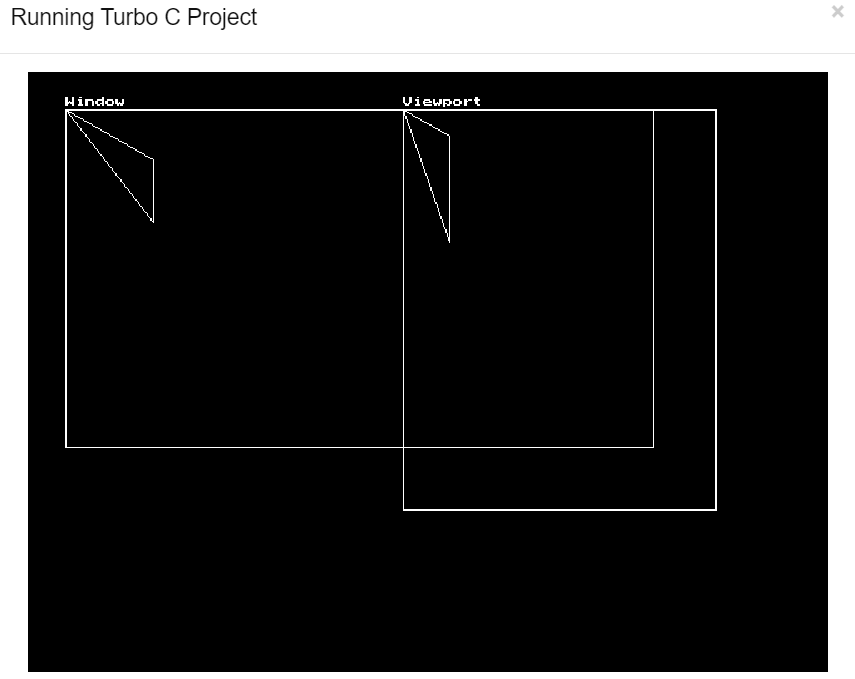
getch();

closegraph();

}

**Output:**





**Result**:

Thus, the experiment has been successfully completed and the output is verified.

**EXP NO: 7 DATE:**

**LIANG BARSKY CLIPPING ALGORITHM**

**Aim**:

To implement Liang Barsky line clipping algorithm in C.

**Algorithm:**

1. Assume tmin = 0 and tmax = 1.
2. Calculate the values of tL, tR, tT and tB. If t < tmin ot t < tmax ignore it, and go to the next edge. Otherwise classify the t value as entering or exiting value. If t is entering value set tmin = t if t is exiting value set tmax = t.
3. If tmin < tmax then draw a line from (x1 + dx \* tmin, y1 + dy \* tmin) to (x1 + dx \* tmax, y1 + dy \* tmax)
4. If the line crosses over the window, you will see (x1 + dx \* tmin, y1 + dy \* tmin) and (x1 + dx \* tmax, y1 + dy \* tmax) are intersection between line and edge.

**Source Code:**

#include<stdio.h>

#include<graphics.h>

#include<math.h>

#include<dos.h>

main(){

int i, gd=DETECT, gm;

int x1=120, y1=120, x2=300, y2=300, xmin=100, xmax=250, ymin=100, ymax=250, xx1, xx2, yy1, yy2, dx, dy;

float t1, t2, p[4], q[4], temp;

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

rectangle (xmin, ymin, xmax, ymax);

dx=x2-x1; dy=y2-y1;

p[0]=-dx; p[1]=dx; p[2]=-dy; p[3]=dy; q[0]=x1-xmin; q[1]=xmax-x1; q[2]=y1-ymin; q[3]=ymax-y1;

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

if(p[i]==0) {

printf("line is parallel to one of the clipping boundary");

if(q[i]>=0){

if(i<2) {

if(y1<ymin) {

y1=ymin; }

if(y2>ymax) {

y2=ymax; }

line(x1,y1,x2,y2); }

if(i>1) {

if(x1<xmin) {

x1=xmin; }

if(x2>xmax) {

x2=xmax; }

line(x1,y1,x2,y2);

}}}}

t1=0;

t2=1;

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

temp=q[i]/p[i];

if(p[i]<0) {

if(t1<=temp)

t1=temp; }

else{

if(t2>temp)

t2=temp; }}

if(t1<t2) {

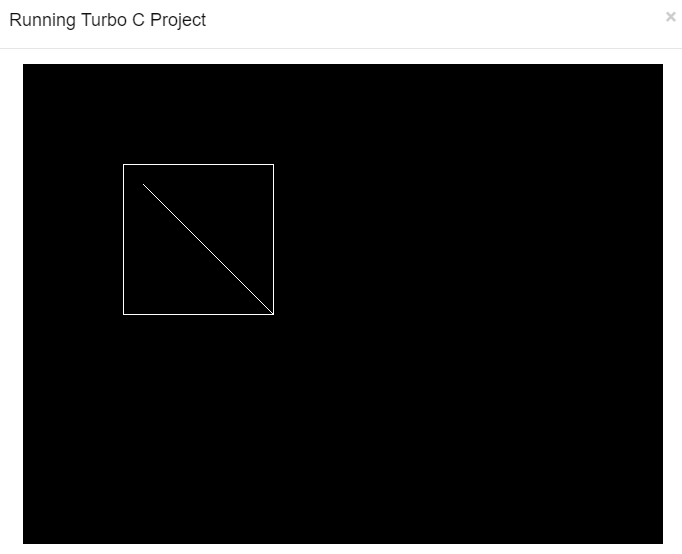
xx1 = x1 + t1 \* p[1]; xx2 = x1 + t2 \* p[1]; yy1 = y1 + t1 \* p[3]; yy2 = y1 + t2 \* p[3];

line(xx1,yy1,xx2,yy2); }

delay(3000);

closegraph(); }

**Output:**



**Result**:

Thus, the experiment has been successfully completed and the output is verified.

**EXP NO: 8 DATE:**

**COHEN SUTHERLAND CLIPPING ALGORITHM**

**Aim**:

To implement Cohen Sutherland line clipping algorithm in C++.

**Algorithm:**

1. Assign a region code for two endpoints of given line.
2. If both endpoints have a region code 0000 then given line is completely inside and we will keep this line.
3. If step 2 fails, perform the logical AND operation for both region codes.
   1. If the result is not 0000, then given line is completely outside.
   2. Else line is partially inside.
      1. Choose an endpoint of the line that is outside the given rectangle.
      2. Find the intersection point of the rectangular boundary (based on region code).
      3. Replace endpoint with the intersection point and update the region code.
      4. Repeat step 2 until we find a clipped line either trivially accepted or rejected.
4. Repeat step 1 for new inputs.

**Source Code:**

#include <iostream>

using namespace std;

class CohenSutherLandAlgo {

private:

double x1,y1,x2,y2;

double x\_max,y\_max,x\_min,y\_min;

const int INSIDE = 0;

const int LEFT = 1;

const int RIGHT = 2;

const int BOTTOM = 4;

const int TOP = 8;

public:

CohenSutherLandAlgo() {

x1 = 0.0;

x2 = 0.0;

y1 = 0.0;

y2 = 0.0;

}

void getCoordinates();

void getClippingRectangle();

int generateCode(double x, double y);

void cohenSutherland();

};

void CohenSutherLandAlgo::getCoordinates(){

cout<<"\nEnter Co-ordinates of P1(X1,Y1) of Line Segment : ";

cout<<"\nEnter X1 Co-ordinate : ";

cin>>x1;

cout<<"\nEnter Y1 Co-ordinate : ";

cin>>y1;

cout<<"\nEnter Co-ordinates of P2(X2,Y2) of Line Segment : ";

cout<< "\nEnter X2 Co-ordinate : ";

cin>>x2;

cout<<"\nEnter Y2 Co-ordinate : ";

cin>>y2;

}

void CohenSutherLandAlgo::getClippingRectangle(){

cout<< "\nEnter the Co-ordinates of Interested Rectangle.";

cout<<"\nEnter the X\_MAX : ";

cin>>x\_max;

cout<<"\nEnter the Y\_MAX : ";

cin>>y\_max;

cout<<"\nEnter the X\_MIN : ";

cin>>x\_min;

cout<<"\nEnter the Y\_MIN : ";

cin>>y\_min;

}

int CohenSutherLandAlgo::generateCode(double x, double y) {

int code = INSIDE; // intially initializing as being inside

if (x < x\_min) // lies to the left of rectangle

code |= LEFT;

else if (x > x\_max) // lies to the right of rectangle

code |= RIGHT;

if (y < y\_min) // lies below the rectangle

code |= BOTTOM;

else if (y > y\_max) // lies above the rectangle

code |= TOP;

return code;

}

void CohenSutherLandAlgo::cohenSutherland() {

int code1 = generateCode(x1, y1);

int code2 = generateCode(x2, y2);

bool accept = false;

while (true) {

if ((code1 == 0) && (code2 == 0)) {

accept = true;

break;

}

else if (code1 & code2) {

break;

}

else {

int code\_out;

double x, y;

if (code1 != 0)

code\_out = code1;

else

code\_out = code2;

if (code\_out & TOP) {

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

y = y\_max;

}

else if (code\_out & BOTTOM) {

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

y = y\_min;

}

else if (code\_out & RIGHT) {

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

x = x\_max;

}

else if (code\_out & LEFT) {

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

x = x\_min;

}

if (code\_out == code1) {

x1 = x;

y1 = y;

code1 = generateCode(x1, y1);

}

else{

x2 = x;

y2 = y;

code2 = generateCode(x2, y2);

}

}

}

if (accept) {

cout<<"Line accepted from ("<<x1<<","<<y1<<") to ("<<x2<<","<<y2<<")";

}

else

cout<<"Line can't be drawn";

}

int main() {

CohenSutherLandAlgo c;

c.getCoordinates();

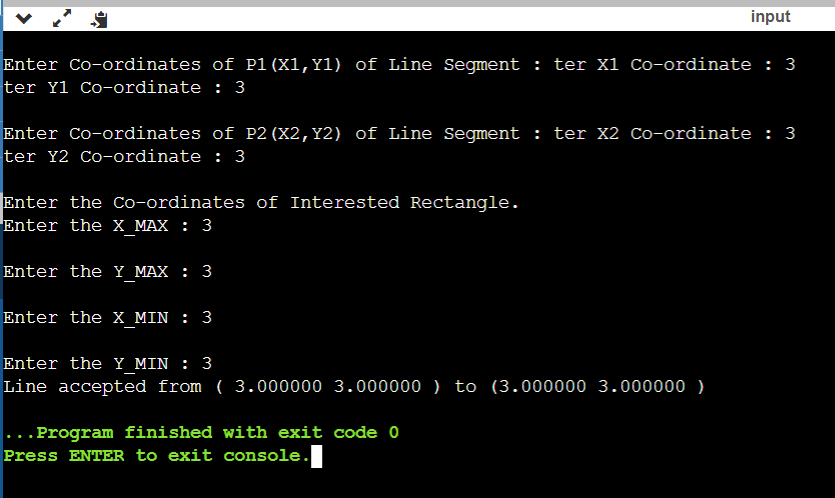
c.getClippingRectangle();

c.cohenSutherland();

return 0;

}

**Output:**



**Result**:

Thus, the experiment has been successfully completed and the output is verified.

**EXP NO: 9 DATE:**

**PERFORM 2D TRANSFORMATIONS OF AN OBJECT**

**Aim**:

To perform 2D Transformations of an object in C.

**Algorithm:**

1.Take 2\*2 matrix

2. For each point of (1) make a 2\*1 matrix P where P[0][0] equals to x coordinate of the point and P[1][0] equals to y coordinate of the point. (2) Multiply the matrix with the point P to get new co-ordinate.

3. Draw the final result using the new points.

**Source Code:**

**Scaling:**

#include<stdio.h>

#include<graphics.h>

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

{

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

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

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

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

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

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

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

}

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

{

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

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

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

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

int p[2][1];

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

{

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

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

findNewCoordinate(s, p);

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

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

}

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

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

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

}

int main()

{

int x[] = { 100, 200, 400 };

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

int sx = 2, sy = 2;

int gd, gm;

detectgraph(&gd, &gm);

initgraph(&gd, &gm," ");

scale(x, y, sx,sy);

getch();

return 0;

}

**Rotation:**

#include<stdio.h>

#include<graphics.h>

#include<math.h>

void main()

{

int gd=DETECT, r, gm, d, x1, y1, x2, y2, xn1, yn1, xn2, yn2;

float ra, si, co;

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

printf("Enter the value of X1 and Y1: ");

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

printf("Enter the value of X2 and Y2: ");

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

line(x1, y1, x2, y2);

printf("Enter the degree of rotation: ");

scanf("%d", &d);

//Starting point would be same

xn1 = x1;

yn1 = y1;

//Convert Degree into radian

r = x2-x1;

ra = 0.0175 \* d;

si = sin(ra);

co = cos(ra);

//second point

xn2 = x1 + r\*co + 1;

yn2 = y1 + r\*si + 1;

line(xn1, yn1, xn2, yn2);

getch();

closegraph();

}

**Translation:**

#include<stdio.h>

#include<graphics.h>

// function to translate rectangle

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

{

int gd = DETECT, gm, errorcode;

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

setcolor (2);

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

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

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

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

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

setcolor(3);

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

}

int main()

{

int P[2][2] = {50, 80, 120, 180};

int T[] = {20, 10};

translateRectangle (P, T);

return 0;

}

**Shear:**

#include<iostream.h>

#include<graphics.h>

#include<math.h>

#include<conio.h>

#include<dos.h>

void mul(int mat[3][3],int vertex[10][3],int n);

void shear(int vertex[10][3],int n);

void init(int vertex[10][3],int n);

int main()

{

int i,x,y;

int vertex[10][3],n;

clrscr();

cout<<"\nEnter the no. of vertex : ";

cin>>n;

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

{

cout<<"Enter the points (x,y): ";

cin>>x>>y;

vertex[i][0]=x;

vertex[i][1]=y;

vertex[i][2]=1;

}

shear(vertex,n);

getch();

return 0;

}

void init(int vertex[10][3],int n)

{

int gd=DETECT,gm,i;

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

setcolor(10);

line(0,240,640,240); //drawing X axis

line(320,0,320,480); //drawing Y axis

setcolor(3);

line(450,20,490,20);

setcolor(15);

line(450,50,490,50);

setcolor(6);

outtextxy(500,20,"Original");

outtextxy(500,50,"Transformed");

setcolor(3);

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

{

line(320+vertex[i][0],240-vertex[i][1],320+vertex[i+1][0],240-vertex[i+1][1]);

}

line(320+vertex[n-1][0],240-vertex[n-1][1],320+vertex[0][0],240-vertex[0][1]);

}

void mul(int mat[3][3],int vertex[10][3],int n)

{

int i,j,k;

int res[10][3];

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

{

for(j=0;j<3;j++)

{

res[i][j]=0;

for(k=0;k<3;k++)

{

res[i][j] = res[i][j] + vertex[i][k]\*mat[k][j];

}

}

}

setcolor(15);

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

{

line(320+res[i][0],240-res[i][1],320+res[i+1][0],240-res[i+1][1]);

}

line(320+res[n-1][0],240-res[n-1][1],320+res[0][0],240-res[0][1]);

}

void shear(int vertex[10][3],int n)

{

int opt;

int shear\_array[3][3];

cout<<"\n1.x-shear\n2.y-shear\nYour Choice: ";

cin>>opt;

switch(opt)

{

case 1: int xsh;

cout<<"\nEnter the x shear : ";

cin>>xsh;

shear\_array[0][0]=1;

shear\_array[1][0]=xsh;

shear\_array[2][0]=0;

shear\_array[0][1]=0;

shear\_array[1][1]=1;

shear\_array[2][1]=0;

shear\_array[0][2]=0;

shear\_array[1][2]=0;

shear\_array[2][2]=1;

init(vertex,n);

mul(shear\_array,vertex,n);

break;

case 2:int ysh;

cout<<"\nEnter the y shear : ";

cin>>ysh;

shear\_array[0][0]=1;

shear\_array[1][0]=0;

shear\_array[2][0]=0;

shear\_array[0][1]=ysh;

shear\_array[1][1]=1;

shear\_array[2][1]=0;

shear\_array[0][2]=0;

shear\_array[1][2]=0;

shear\_array[2][2]=1;

init(vertex,n);

mul(shear\_array,vertex,n);

break;

}

}

**Reflection:**

#include <conio.h>

#include <graphics.h>

#include <stdio.h>

void main()

{

int gm, gd = DETECT, ax, x1 = 100;

int x2 = 100, x3 = 200, y1 = 100;

int y2 = 200, y3 = 100;

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

cleardevice();

line(getmaxx() / 2, 0, getmaxx() / 2,

getmaxy());

line(0, getmaxy() / 2, getmaxx(),

getmaxy() / 2);

printf("Before Reflection Object"

" in 2nd Quadrant");

setcolor(14);

line(x1, y1, x2, y2);

line(x2, y2, x3, y3);

line(x3, y3, x1, y1);

getch();

printf("\nAfter Reflection");

setcolor(4);

line(getmaxx() - x1, getmaxy() - y1,

getmaxx() - x2, getmaxy() - y2);

line(getmaxx() - x2, getmaxy() - y2,

getmaxx() - x3, getmaxy() - y3);

line(getmaxx() - x3, getmaxy() - y3,

getmaxx() - x1, getmaxy() - y1);

setcolor(3);

line(getmaxx() - x1, y1,

getmaxx() - x2, y2);

line(getmaxx() - x2, y2,

getmaxx() - x3, y3);

line(getmaxx() - x3, y3,

getmaxx() - x1, y1);

setcolor(2);

line(x1, getmaxy() - y1, x2,

getmaxy() - y2);

line(x2, getmaxy() - y2, x3,

getmaxy() - y3);

line(x3, getmaxy() - y3, x1,

getmaxy() - y1);

getch();

closegraph();

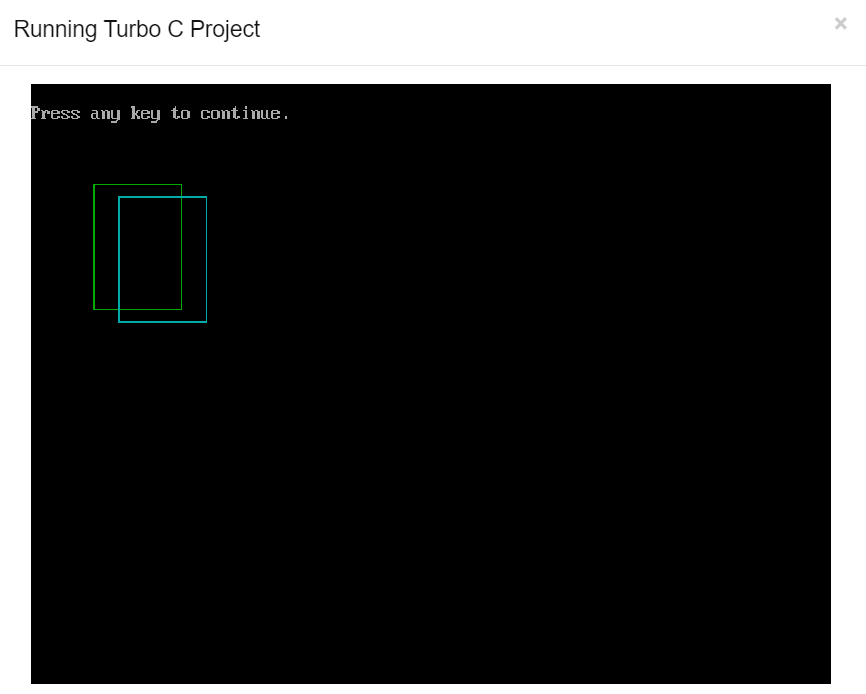
}

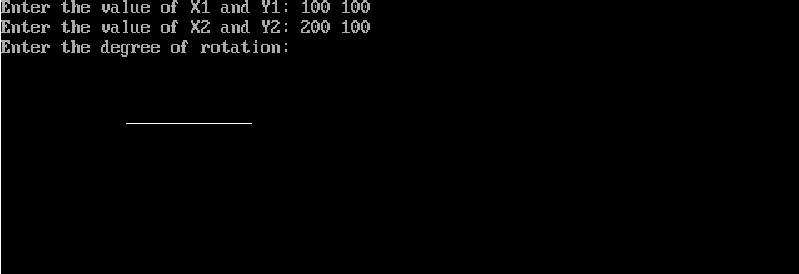
**Output:**

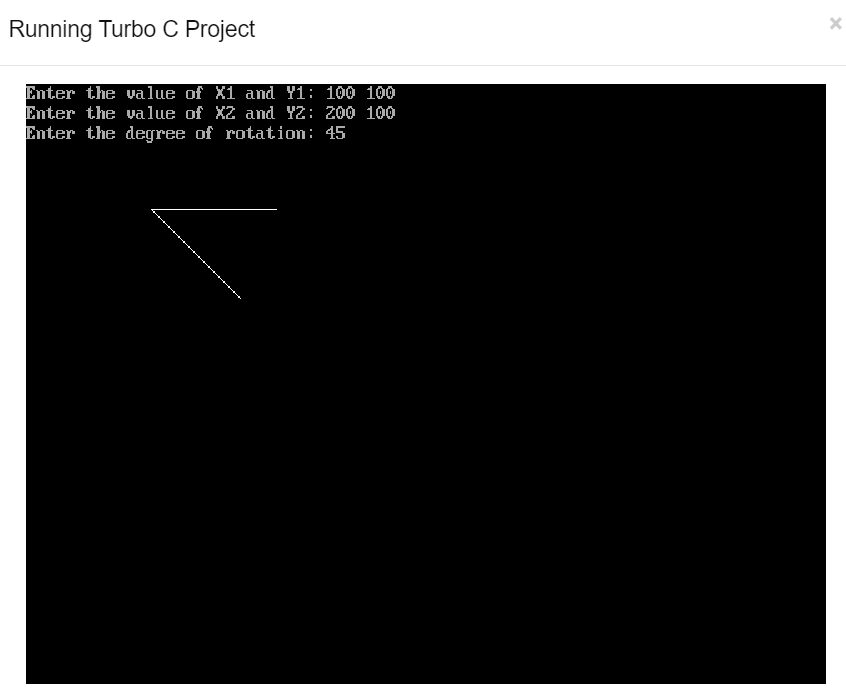
**Scaling of Objects:**



**Translation of Objects:**

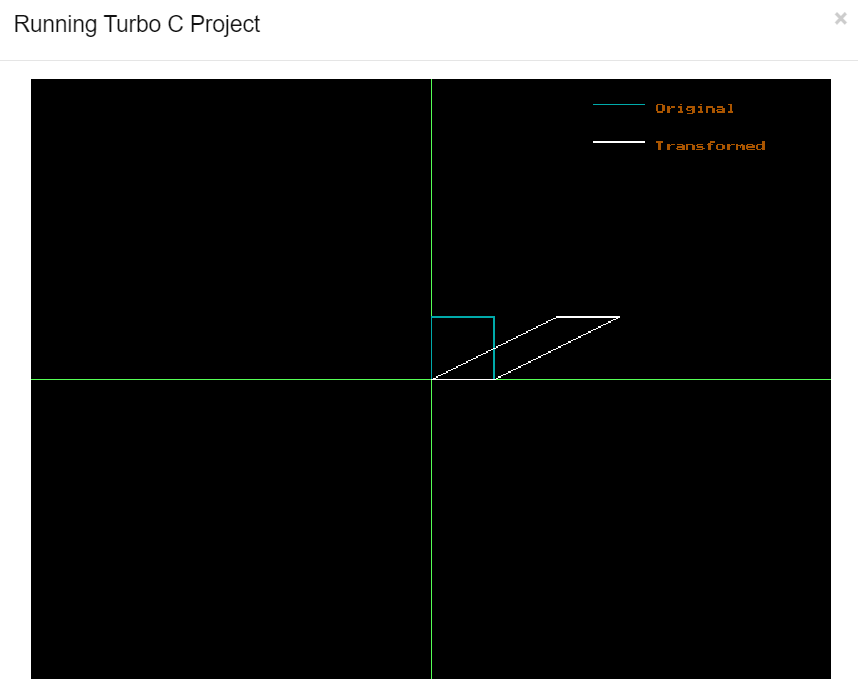


**Rotation of Objects:**

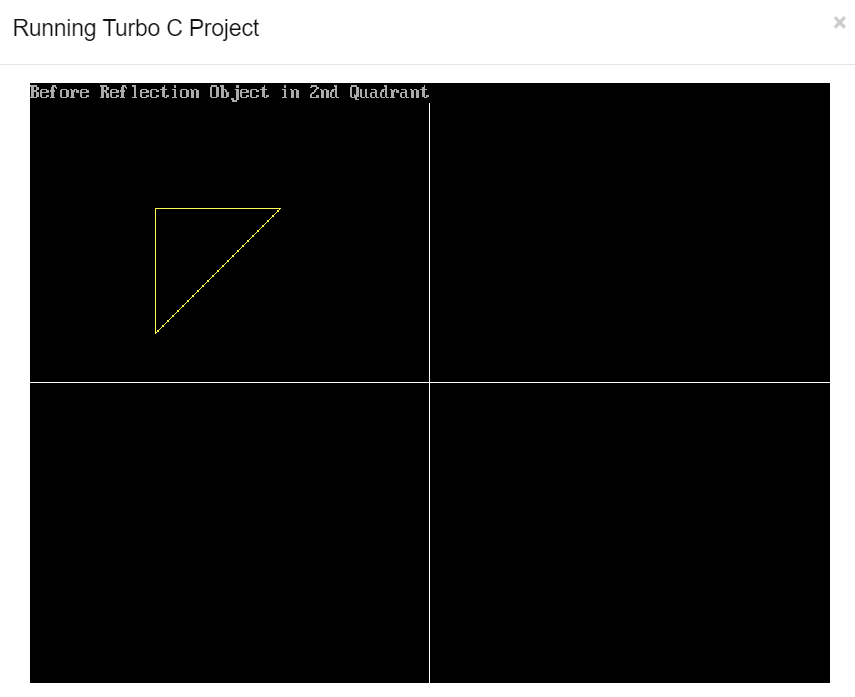


**Shearing:**





**Reflection of Objects:**



**Result**:

Thus, the experiment has been successfully completed and the output is verified.

**EXP NO: 10 DATE:**

**CREATE 2D AND 3D OBJECTS USING OPENGL**

**Aim**:

To create 2D and 3D objects using OpenGL.

**Source Code:**

#include <windows.h> #include <GL/glut.h> char title[] = "3D Shapes"; void initGL() {

glClearColor(0.0f, 0.0f, 0.0f, 1.0f); glClearDepth(1.0f); glEnable(GL\_DEPTH\_TEST); glDepthFunc(GL\_LEQUAL); glShadeModel(GL\_SMOOTH);

glHint(GL\_PERSPECTIVE\_CORRECTION\_HINT, GL\_NICEST);

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslatef(1.5f,0.0f,-7.0f); glBegin(GL\_QUADS);

glColor3f(0.0f, 1.0f, 0.0f); // Green

glVertex3f( 1.0f, 1.0f, -1.0f);

glVertex3f(-1.0f, 1.0f, -1.0f);

glVertex3f(-1.0f, 1.0f, 1.0f);

glVertex3f( 1.0f, 1.0f, 1.0f);

glColor3f(1.0f, 0.5f, 0.0f); // Orange

glVertex3f( 1.0f, -1.0f, 1.0f);

glVertex3f(-1.0f, -1.0f,1.0f);

glVertex3f(-1.0f, -1.0f, -1.0f);

glColor3f(1.0f, 0.0f, 0.0f); // Red

glVertex3f( 1.0f, 1.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 1.0f);

glVertex3f(-1.0f, -1.0f, 1.0f); glVertex3f( 1.0f, -1.0f, 1.0f); glColor3f(1.0f, 1.0f, 0.0f); // Yellow

glVertex3f( 1.0f, -1.0f, -1.0f); glVertex3f(-1.0f, -1.0f, -1.0f);

glVertex3f(-1.0f, 1.0f, -1.0f); glVertex3f( 1.0f, 1.0f, -1.0f); glColor3f(0.0f, 0.0f, 1.0f); // Blue

glVertex3f(-1.0f, 1.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);

glVertex3f(-1.0f, -1.0f, -1.0f); glVertex3f(-1.0f, -1.0f, 1.0f);

// Right face (x = 1.0f)

glColor3f(1.0f, 0.0f, 1.0f); // Magenta

glVertex3f(1.0f, 1.0f, -1.0f); glVertex3f(1.0f, 1.0f, 1.0f);

glVertex3f(1.0f, -1.0f, 1.0f); glVertex3f(1.0f,-1.0f,-1.0f); glEnd();

glLoadIdentity();

glTranslatef(-1.5f, 0.0f,6.0f);

glBegin(GL\_TRIANGLES);

glColor3f(1.0f, 0.0f, 0.0f);

glVertex3f( 0.0f, 1.0f, 0.0f);

glColor3f(0.0f, 1.0f, 0.0f);

glVertex3f(-1.0f, -1.0f, 1.0f);

glColor3f(0.0f, 0.0f, 1.0f);

glVertex3f(1.0f, -1.0f, 1.0f);

glColor3f(1.0f, 0.0f, 0.0f); // Red

glVertex3f(0.0f, 1.0f, 0.0f);

glColor3f(0.0f, 0.0f, 1.0f); // Blue

glVertex3f(1.0f, -1.0f, 1.0f);

glColor3f(0.0f, 1.0f, 0.0f); // Green

glVertex3f(1.0f, -1.0f, -1.0f);

glColor3f(1.0f, 0.0f, 0.0f); // Red

glVertex3f(0.0f, 1.0f, 0.0f);

glColor3f(0.0f, 1.0f, 0.0f); // Green

glVertex3f(1.0f, -1.0f, -1.0f);

glColor3f(0.0f, 0.0f, 1.0f); // Blue

glVertex3f(-1.0f, -1.0f, -1.0f);

glColor3f(1.0f,0.0f,0.0f); // Red

glVertex3f( 0.0f, 1.0f, 0.0f);

glColor3f(0.0f,0.0f,1.0f); // Blue

glVertex3f(-1.0f,-1.0f,-1.0f);

glColor3f(0.0f,1.0f,0.0f); // Green

glVertex3f(-1.0f,-1.0f, 1.0f);

glEnd();

glutSwapBuffers(); // Swap the front and back frame buffers (double buffering)

}

void reshape(GLsizei width, GLsizei height) {

if (height == 0) height = 1;

GLfloat aspect = (GLfloat)width / (GLfloat)height;

glViewport(0, 0, width, height);

glMatrixMode(GL\_PROJECTION); // To operate on the Projection matrix glLoadIdentity(); // Reset

gluPerspective(45.0f, aspect, 0.1f, 100.0f);

}

int main(int argc, char\*\* argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE);

glutInitWindowSize(640, 480);

glutInitWindowPosition(50, 50);

glutCreateWindow(title);

glutDisplayFunc(display);

glutReshapeFunc(reshape);

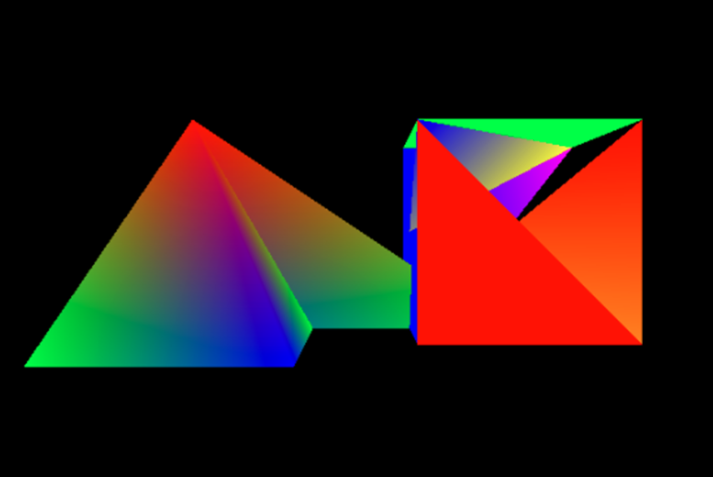
initGL();

glutMainloop();

return 0;

}

**Output:**



**Result**:

Thus, the experiment has been successfully completed and the output is verified.

**EXP NO: 11 DATE:**

**CREATE 3D PROJECTIONS USING OPENGL**

**Aim**:

To create 3D Projections using OpenGL.

**Source Code:**

#ifdef

#include <GLUT/glut.h>

#else

#include <GL/glut.h>

#endif

static bool spinning = true;

static const int FPS = 60;

static GLfloat currentAngleOfRotation = 0.0;

void reshape(GLint w, GLint h)

{ glViewport(0, 0, w, h);

GLfloat aspect = (GLfloat)w / (GLfloat)h;

glMatrixMode(GL\_PROJECTION); glLoadIdentity();

if (w <= h) {

glOrtho(-50.0, 50.0, -50.0/aspect, 50.0/aspect, -1.0, 1.0);

}

else {

glOrtho(-50.0\*aspect, 50.0\*aspect, -50.0, 50.0, -1.0, 1.0);

}}

void display() { glClear(GL\_COLOR\_BUFFER\_BIT);

glMatrixMode(GL\_MODELVIEW); glLoadIdentity();

glRotatef(currentAngleOfRotation, 0.0, 0.0, 1.0);

glRectf(-25.0, -25.0, 25.0, 25.0);

glFlush(); glutSwapBuffers();

}

void timer(int v) {

if (spinning) {

currentAngleOfRotation += 1.0;

if (currentAngleOfRotation > 360.0)

{ currentAngleOfRotation -= 360.0;

}

glutPostRedisplay();

}

glutTimerFunc(1000/FPS, timer, v);

}

void mouse(int button, int state, int x, int y) {

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN)

{ spinning = true;}

else if (button == GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN) { spinning = false;

}

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB); glutInitWindowPosition(80, 80);

glutInitWindowSize(800, 500);

glutCreateWindow("Spinning Square");

glutReshapeFunc(reshape);

glutDisplayFunc(display);

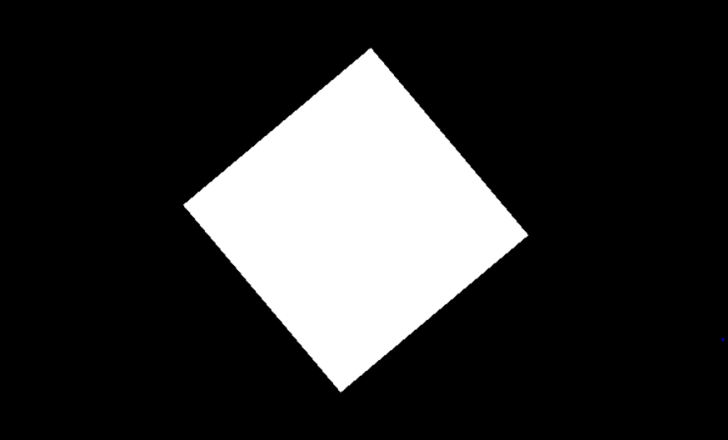
glutTimerFunc(100, timer, 0);

glutMouseFunc(mouse);

glutMainLoop();

}

**Output:**



**Result**:

Thus, the experiment has been successfully completed and the output is verified.