Experiment No : 1

Aim : Running your first graphics program on ubuntu.

Theory :
libgraph : libgraph is an implementation of the TurboC graphics API (graphics.h) on GNU/Linux using SDL. The library provides easy-to-use 2D graphics interface that can be used for simple prototyping, visualization or studying graphics algorithms.

Installation of libgraph:
1. First, we need to install the build-essentials package to use make, g++ and other compile tools. We can do this with the command

sudo apt-get install build-essential

2. Next we need to install the dependencies of libgraph:

sudo apt-get install libsdl-image1.2 libsdl-image1.2-dev guile-1.8 guile-1.8-dev libart-2.0-dev libaudiofile-dev libesd0-dev libdirectfb-dev libdirectfb-extra libfreetype6-dev libxext-dev x11proto-xext-dev libfreetype6 libaa1 libaa1-dev libslang2-dev libasound2 libasound2-dev

3. Now we need to download libgraph from http://download.savannah.gnu.org/releases/libgraph/libgraph1.0.2.tar.gz and untar the file:

wget http://download.savannah.gnu.org/releases/libgraph/libgraph-1.0.2.tar.gz tar -xf libgraph-1.0.2.tar.gz

You should now have the libgraph folder

4. Before compiling and installing we need to run the configure script in the libgraph folder. So we first give it execution permissions using chmod and then run it. The configure script runs preinstallation checks and sets up the system for installation.

cd libgraph-1.0.2
chmod +x configure
./configure

5. To install we run certain make commands and copy the compiled binaries from /usr/local/lib/ to /usr/lib/tar

sudo make
sudo make install
sudo cp /usr/local/lib/libgraph.* /usr/lib/

Execution :
1. Write a simple graphics program and add a delay to prevent the program from closing immediately.

```
 #include<graphics.h>
int main(){
int gd=DETECT,gm;
initgraph(&gd,&gm,"C:\\TC\\BGI");
circle(100,100,30);
delay(100);
closegraph();
}
```

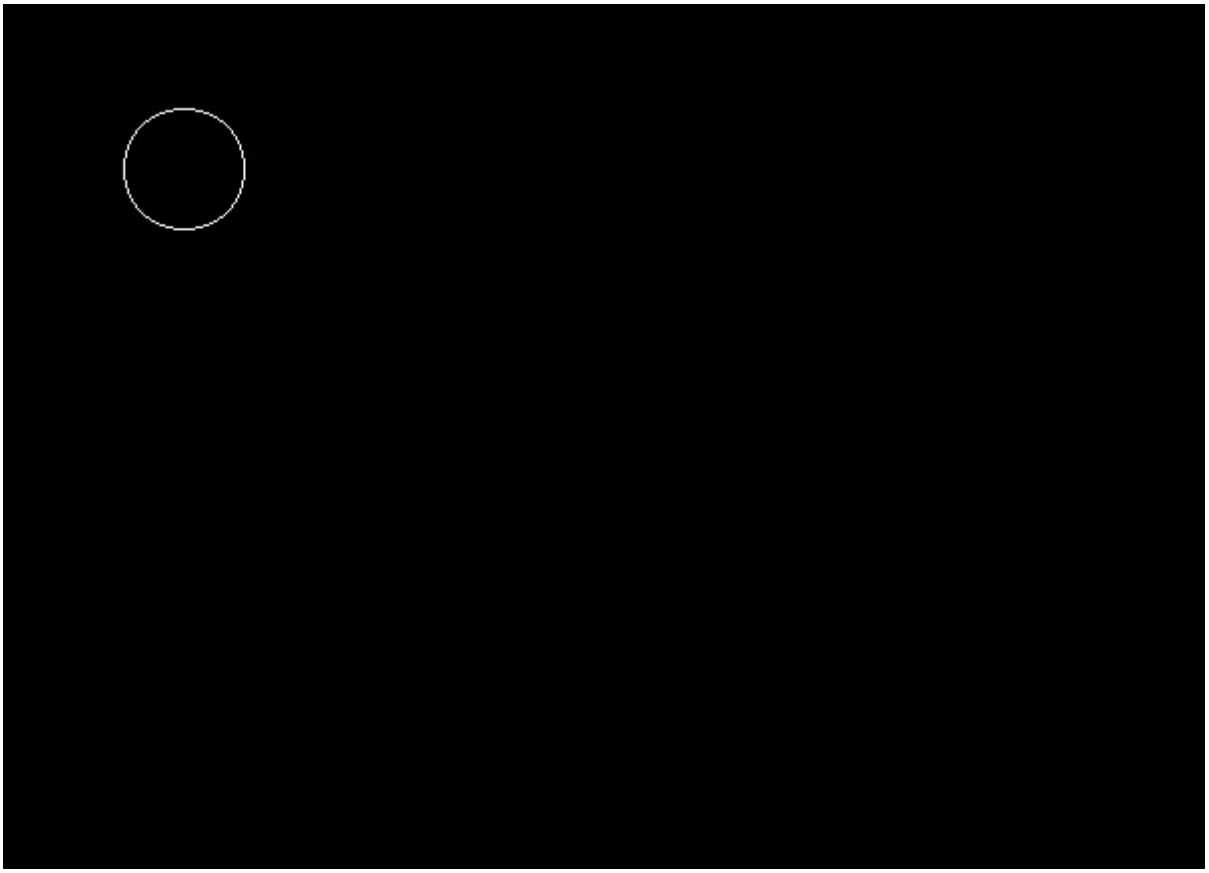
2. To compile the program use the following command:

g++ -g filename.cpp -o filename -lgraph

3. To execute the binary produced from compilation run:

./filename

Output :



Conclusion : libgraph was installed and example graphics program was compiled and installed.

Deepraj Bhosale
181105016

Experiment No : 2

Aim : Draw your name in English and Devnagiri using the graphics library

Theory :
The libgraph library : Libgraph is an implementation of the TurboC graphics API (graphics.h) on GNU/Linux using SDL. It was created as an easy-to-use tool for learning 2D graphics and for prototyping rather than as a powerful or flexible graphics library. The library Faraz Shahbazker, an engineering student from Mumbai. Some of the useful functions provided include:
1. initgraph and closegraph to create a close the canvas
2. drawpoly to draw a series of connected lines from a given list of points
3. circle and ellipse, to draw circles and eliipses
4. setbkcolor and setcolor to set the color of the canvas and its elements repsectively.

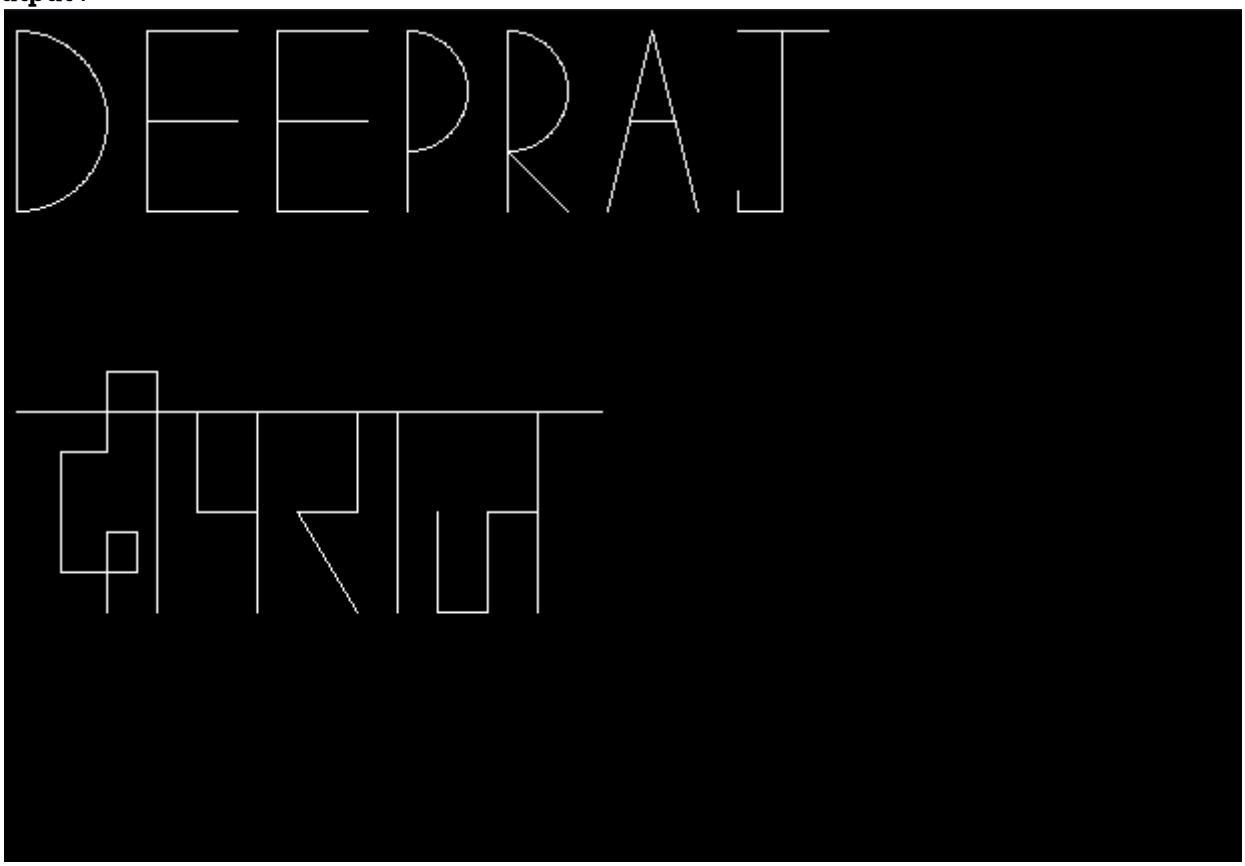
Code & Output :

```
#include<graphics.h>
#include<iostream>
using namespace std;
int main()
{
int gd=DETECT,gm;

initgraph(&gd,&gm,NULL);


line(10,10,10,100);
delay(500);
ellipse(10,55,270,90,45,45);
delay(500);


line(75,10,75,100);
delay(500);
line(75,10,120,10);
delay(500);
line(75,55,120,55);
delay(500);
line(75,100,120,100);
delay(500);

line(140,10,140,100);
delay(500);
line(140,10,185,10);
delay(500);
line(140,55,185,55);
delay(500);
line(140,100,185,100);
delay(500);

line(205,10,205,100);
delay(500);
ellipse(205,40,270,90,30,30);
delay(500);

line(255,10,255,100);
delay(500);
```

```
ellipse(255,40,270,90,30,30);
delay(500);
line(255,70,285,100);
delay(500);

line(305,100,327.5,10);
delay(500);
line(350,100,327.5,10);
delay(500);
line(316.25,55,338.75,55);
delay(500);

line(370,10,415,10);
delay(500);
line(392.5,10,392.5,100);
delay(500);
line(392.5,100,370,100);
delay(500);
line(370,100,370,90);
delay(500);


line(10,200,302.5,200);
delay(500);

line(55,200,55,220);
delay(500);
line(55,220,32.5,220);
delay(500);
line(32.5,220,32.5,280);
delay(500);
line(32.5,280,70,280);
delay(500);
line(70,280,70,260);
delay(500);
line(70,260,55,260);
delay(500);
line(55,260,55,300);
delay(500);
line(55,200,55,180);
delay(500);
line(55,180,80,180);
delay(500);
line(80,180,80,300);
delay(500);

line(100,200,100,250);
delay(500);
line(100,250,130,250);
delay(500);
line(130,200,130,300);
delay(500);

line(180,200,180,250);
delay(500);
line(180,250,150,250);
delay(500);
```

```
line(150,250,180,300);
delay(500);

line(200,200,200,300);
delay(500);

line(220,250,220,300);
delay(500);
line(220,300,245,300);
delay(500);
line(245,300,245,250);
delay(500);
line(245,250,270,250);
delay(500);
line(270,200,270,300);


delay(10000);
closegraph();
cout<<"Experiment 2"<<endl;
}
```

Output :



Conclusion : Program to write name in english and devnagiri were successfully written and executed

Deepraj Bhosale
181105016

Experiment No : 3

Aim : To create a simple animation using functions defined in the libgraph library

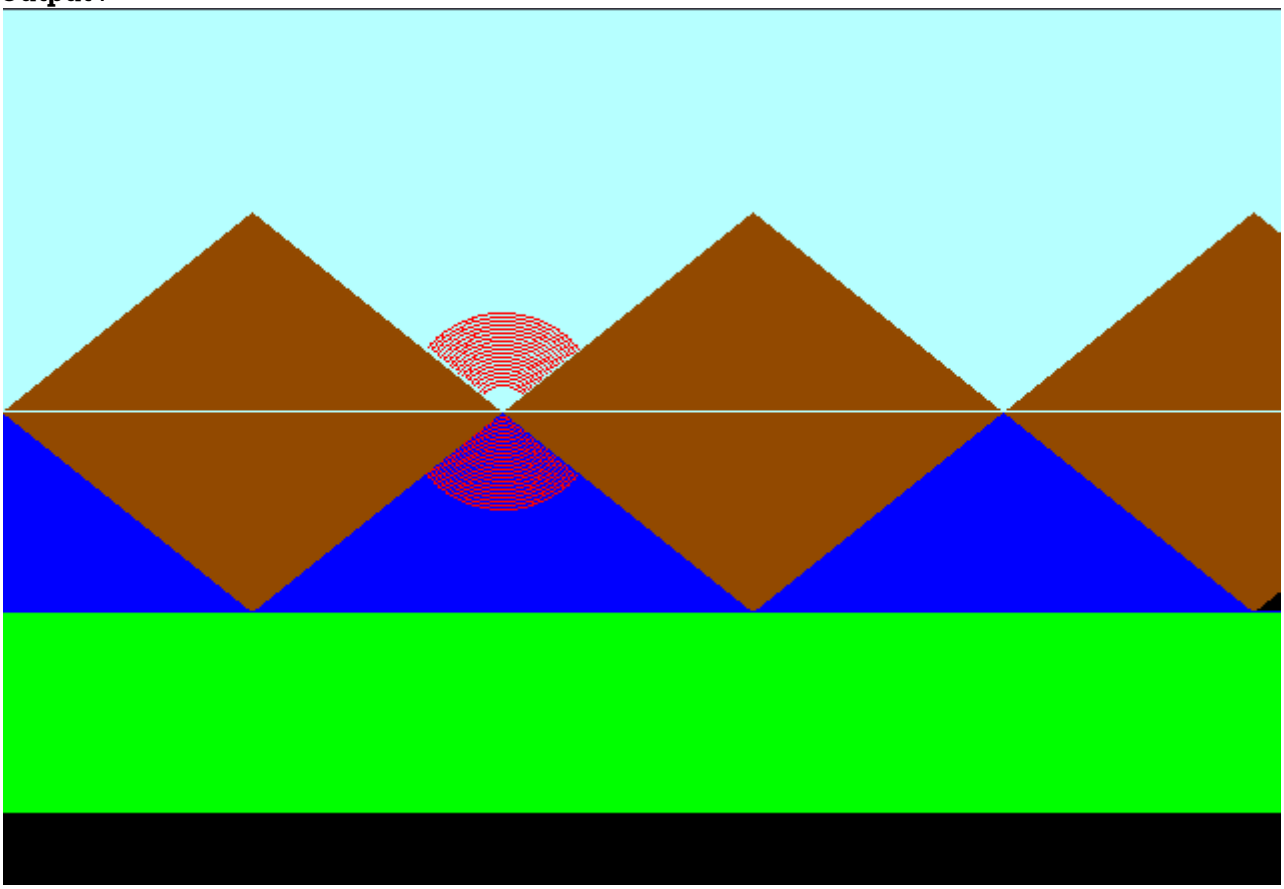
Theory :
Animation is a method in which figures are manipulated to appear as moving images. Generally the effect of animation is achieved by a rapid succession of sequential images that minimally differ from each other. In computer animation this is achieved by changing the image every few milliseconds. These changes act like frames.

Code & Output :

```
#include<graphics.h>
#include<iostream>
using namespace std;
int main()
{
int gd=DETECT,gm;

initgraph(&gd,&gm,NULL);
int r=1;

while (r<=50){

//Mountains
setcolor(BROWN);
line(0,200,1000,200);
line(0,200,125,100);
line(125,100,250,200);
line(250,200,375,100);
line(375,100,500,200);
line(500,200,625,100);
line(625,100,750,200);
setcolor(BROWN);
floodfill(50,190,WHITE);
floodfill(270,190,WHITE);
floodfill(525,190,WHITE);

//Mountain Reflection
line(0,200,125,300);
line(125,300,250,200);
line(250,200,375,300);
line(375,300,500,200);
line(500,200,625,300);
line(625,300,750,200);
floodfill(50,210,WHITE);
floodfill(270,210,WHITE);
floodfill(525,210,WHITE);

//Animated Sun between a mountain
setcolor(RED);
arc(250,200,220,320,r);
arc(250,200,40,140,r);
 //floodfill(250,201,WHITE);
```

```
//Water
 setcolor(BLUE);
 line(0,300,1000,300);
 floodfill(50,250,BROWN);
 floodfill(300,250,BROWN);
 floodfill(550,250,BROWN);

 //Grass
 setcolor(GREEN);
 line(0,400,1000,400);
 floodfill(50,350,BLUE);

if (r>10){
   setcolor(LIGHTCYAN);
   line(0,0,1000,0);
   line(0,0,0,200);
   line(1000,0,1000,200);
   line(0,200,1000,200);
   line(0,200,125,100);
   line(125,100,250,200);
   line(250,200,375,100);
   line(375,100,500,200);
   line(500,200,625,100);
   line(625,100,750,200);
   floodfill(10,10,GREEN);
}

r+=2;
delay(100);
}
delay(1500);
closegraph();
cout<<"Experiment 3"<<endl;
}
}
```

Output :



Conclusion : Program to create an animation using libgraph was written and executed successfully

Deepraj Bhosale
181105016

Experiment No : 4

Aim : Using DDA algorithm draw a house.

Theory :

Digital Differential Analyzer

The digital differential analyzer (DDA) is a scan-conversion line algorithm. A line is sampled at unit intervals in one coordinate and the corresponding integer values nearest the line path are determined for the other coordinate.

The DDA algorithm is a faster method for calculating pixel positions than one that directly implements the line equation. It eliminates the multiplication in the line equation by using raster characteristics, so that appropriate increments are applied in the x or y directions to step from one pixel position to another along the line path. The accumulation of round-off error in successive additions of the floating-point.

DDA Algorithm :

Consider start point of the line as (X0,Y0) and the end point of the line as (X1,Y1). Then

1. Calculate dx,dy using: dx = X1 – X0; dy = Y 1 – Y 0;
2. Depending upon absolute value of dx and dy choose number of steps to put pixel as:

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

3. Calculate increment in x and y for each steps:

Xinc = dx/(float)steps;
Yinc = dy/(float)steps;

4. Using the above 3 steps, draw pixels as :

```
X = X0;
Y = Y0;
for (int i = 0; i <= steps; i++)
{
        putpixel (round(X),round(Y),WHITE);
        X += Xinc;
        Y += Yinc;
}
```

Code & Output :

```
#include<graphics.h>
#include<iostream>
#include<cmath>
using namespace std;

void positiveLeftToRight(int x1, int y1, int x2, int y2, float m);
void positiveRightToLeft(int x1, int y1, int x2, int y2, float m);
void negativeLeftToRight(int x1, int y1, int x2, int y2, float m);
void negativeRightToLeft(int x1, int y1, int x2, int y2, float m);
void DDA(int x1, int y1, int x2, int y2);

int main(){
   int gd=DETECT,gm;
   initgraph(&gd,&gm,NULL);

   DDA(40,205,120,100);
   DDA(120,100,200,205);
   DDA(40,205,200,205);


   DDA(50,205,50,300);
   DDA(190,205,190,300);
   DDA(50,300,190,300);


   DDA(100,260,100,300);
   DDA(100,260,125,260);
   DDA(125,260,125,300);

   DDA(135,120,300,120);
   DDA(300,120,350,200);
   DDA(200,200,350,200);

   DDA(290,120,340,200);
   DDA(280,120,330,200);
   DDA(270,120,320,200);
   DDA(260,120,310,200);
   DDA(250,120,300,200);
   DDA(240,120,290,200);
   DDA(230,120,280,200);
   DDA(220,120,270,200);
   DDA(210,120,260,200);
   DDA(200,120,250,200);
   DDA(190,120,240,200);
   DDA(180,120,230,200);
   DDA(170,120,220,200);
   DDA(160,120,210,200);
   DDA(150,120,200,200);

   DDA(350,200,350,300);
   DDA(190,300,350,300);


   DDA(240,230,300,230);
   DDA(240,230,240,270);
   DDA(240,270,300,270);
   DDA(300,230,300,270);
   DDA(270,230,270,270);
   DDA(240,250,300,250);
```

```
   delay(100000);
   closegraph();
   return 0;
}

void DDA(int x1, int y1, int x2, int y2){
float m;
m = float(y2 - y1)/float(x2 - x1);
if((m > 0.0) && (x1 < x2))
   positiveLeftToRight(x1,y1,x2,y2,m);
else if((m > 0.0) && (x1 > x2))
   positiveRightToLeft(x1,y1,x2,y2,m);
else if(((1/m)==0) && (y1 < y2))
   positiveLeftToRight(x1,y1,x2,y2,m);
else if(((1/m)==0) && (y1 > y2))
   negativeLeftToRight(x1,y1,x2,y2,m);
else if((m < 0.0) && (x1 < x2))
   negativeLeftToRight(x1,y1,x2,y2,m);
else if((m < 0.0) && (x1 > x2))
   negativeRightToLeft(x1,y1,x2,y2,m);
else if((m == 0) && (x1 < x2))
   positiveLeftToRight(x1,y1,x2,y2,m);
else if((m == 0) && (x1 > x2))
   positiveRightToLeft(x1,y1,x2,y2,m);

}

void positiveLeftToRight(int x1, int y1, int x2, int y2, float m){
float X = x1, Y = y1;
int plotX = x1, plotY = y1;
if(m <= 1){
   while(plotX <= x2){
         putpixel(plotX,plotY,WHITE);
         X = X + 1;
         Y = Y + m;
         plotX = X;
         plotY = round(Y);

   }
}
else if (m > 1){
   while(plotY <= y2){
         putpixel(plotX,plotY,WHITE);
         X = X + (1/m);
         Y = Y + 1;
         plotX = round(X);
         plotY = Y;

   }
}
}

void positiveRightToLeft(int x1, int y1, int x2, int y2, float m){
float X = x1, Y = y1;
int plotX = x1, plotY = y1;
if(m <= 1){
   while(plotX >= x2){
         putpixel(plotX,plotY,WHITE);
```

```
        X = X - 1;
        Y = Y - m;
        plotX = X;
        plotY = round(Y);

    }
}
else if (m > 1){
    while(plotY >= y2){
        putpixel(plotX,plotY,WHITE);
        X = X - (1/m);
        Y = Y - 1;
        plotX = round(X);
        plotY = Y;

    }
}
}

void negativeLeftToRight(int x1, int y1, int x2, int y2, float m){
float X = x1, Y = y1;
int plotX = x1, plotY = y1;
if(abs(m) <= 1){
    while(plotX <= x2){
        putpixel(plotX,plotY,WHITE);
        X = X + 1;
        Y = Y + m;
        plotX = X;
        plotY = round(Y);

    }
}
else if (abs(m) > 1){
    while(plotY >= y2){
        putpixel(plotX,plotY,WHITE);
        X = X - (1/m);
        Y = Y - 1;
        plotX = round(X);
        plotY = Y;

    }
}
}

void negativeRightToLeft(int x1, int y1, int x2, int y2, float m){
float X = x1, Y = y1;
int plotX = x1, plotY = y1;
if(abs(m) <= 1){
    while(plotX >= x2){
        putpixel(plotX,plotY,WHITE);
        X = X - 1;
        Y = Y - m;
        plotX = X;
        plotY = round(Y);

    }
}
else if (abs(m) > 1){
    while(plotY <= y2){
        putpixel(plotX,plotY,WHITE);
```

```
        X = X + (1/m);
        Y = Y + 1;
        plotX = round(X);
        plotY = Y;

    }
}

}
```

Output :

Conclusion : Program to draw a house using DDA was successfully written and executed

Deepraj Bhosale
181105016

Experiment No : 5

Aim : Using Bresenham's Line Drawing algorithm, draw a cupboard

Theory :
Bresenham's Line Drawing algorithm is an accurate and efficient raster line-generating algorithm, developed by Jack Elton Bresenham, that uses only incremental integer calculations. In addition, Bresenham's line algorithm can be adapted to display circles and other curves.

Bresenham's Line Drawing Algorithm for m <1:

1. Input the two line endpoints and store the left endpoint in (x0, y0).
2. Set the color for frame-buffer position (x0, y0); i.e., plot the first point.
3. Calculate the constants Δ x, Δ y, 2Δ y, and 2Δ y - 2Δ x, and obtain the starting value for the decision parameter as p0 = 2Δy − Δx
4. 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 + 2Δy
Otherwise, the next point to plot is (xk + 1, yk + 1) and pk+1 = pk + 2Δy − 2Δx
5. Repeat step 4, Δx − 1 more times.

Code & Output :

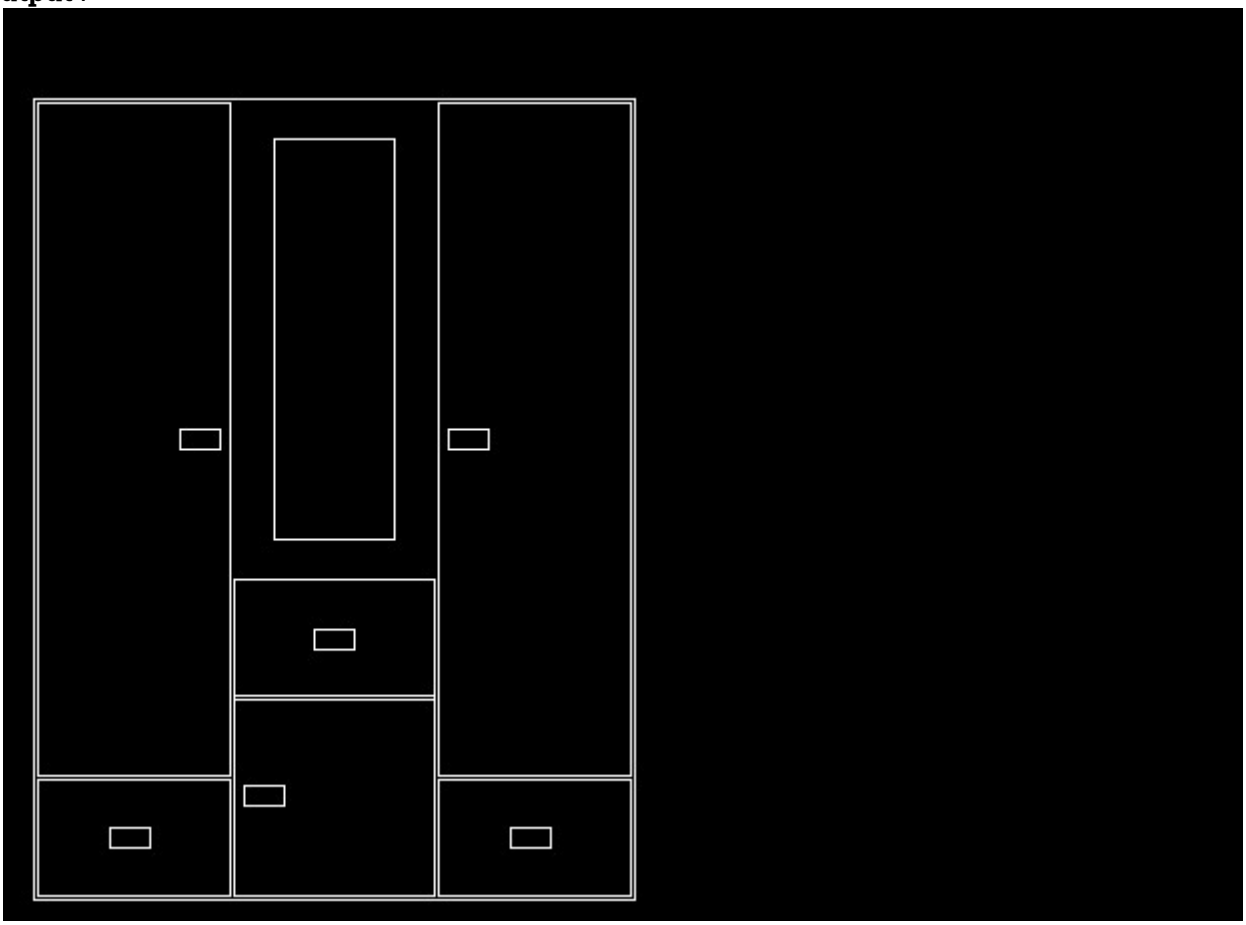
```
#include<graphics.h>
#include<iostream>
#include<cmath>
using namespace std;

void bresenham(int a1, int b1, int a2, int b2);
void forPositiveSlope(int x1, int y1, int x2, int y2, float m);
void forNegativeSlope(int x1, int y1, int x2, int y2, float m);

int main()
{
  int gd=DETECT,gm;
  initgraph(&gd,&gm,NULL);
  bresenham(20,50,320,50);
  bresenham(20,50,20,450);
  bresenham(320,50,320,450);
  bresenham(20,450,320,450);

  bresenham(22,448,118,448);
  bresenham(22,390,22,448);
  bresenham(22,390,118,390);
  bresenham(118,390,118,448);
  bresenham(58,414,78,414);
  bresenham(58,414,58,424);
  bresenham(58,424,78,424);
  bresenham(78,414,78,424);

  bresenham(222,448,318,448);
  bresenham(222,390,222,448);
  bresenham(222,390,318,390);
  bresenham(318,390,318,448);
  bresenham(258,414,278,414);
  bresenham(258,414,258,424);
  bresenham(258,424,278,424);
  bresenham(278,414,278,424);
```

```
   bresenham(120,448,220,448);
   bresenham(120,348,120,448);
   bresenham(120,348,220,348);
   bresenham(220,348,220,448);
   bresenham(125,393,145,393);
   bresenham(125,393,125,403);
   bresenham(125,403,145,403);
   bresenham(145,393,145,403);

   bresenham(120,290,120,350);
   bresenham(120,350,220,350);
   bresenham(120,290,220,290);
   bresenham(220,290,220,350);
   bresenham(160,315,180,315);
   bresenham(160,315,160,325);
   bresenham(160,325,180,325);
   bresenham(180,315,180,325);

   bresenham(22,388,118,388);
   bresenham(22,52,22,388);
   bresenham(22,52,118,52);
   bresenham(118,52,118,388);
   bresenham(93,215,113,215);
   bresenham(93,215,93,225);
   bresenham(93,225,113,225);
   bresenham(113,215,113,225);

   bresenham(222,388,318,388);
   bresenham(222,52,222,388);
   bresenham(222,52,318,52);
   bresenham(318,52,318,388);
   bresenham(227,215,247,215);
   bresenham(227,215,227,225);
   bresenham(227,225,247,225);
   bresenham(247,215,247,225);

   bresenham(140,70,200,70);
   bresenham(140,70,140,270);
   bresenham(140,270,200,270);
   bresenham(200,70,200,270);


   delay(100000);
   closegraph();
    return 0;
}

void bresenham(int a1, int b1, int a2, int b2)
{
    float m;
    if(a1 < a2)
    {
        m = float(b2 - b1)/float(a2 - a1);
        if(m >= 0)
            forPositiveSlope(a1,b1,a2,b2,m);
        else if(m < 0)
            forNegativeSlope(a1,b1,a2,b2,m);

    }
    else if(a2 < a1)
```

```
    {
       m = float(b1 - b2)/float(a1 - a2);
        if(m >= 0)
          forPositiveSlope(a2,b2,a1,b1,m);
       else if(m < 0)
          forNegativeSlope(a2,b2,a1,b1,m);

    }
    else if(a1 == a2)
    {
       m = float(b2 - b1)/float(a2 - a1);
       if(b1 < b2)
          forPositiveSlope(a1,b1,a2,b2,m);
       else if(b2 > b1)
          forNegativeSlope(a1,b1,a2,b2,m);
    }

}

void forPositiveSlope(int x1, int y1, int x2, int y2, float m)
{      int p0, pk, deltaX, deltaY;
       deltaX = x2 - x1;
       deltaY = y2 - y1;
       if(m < 1)
       {
          p0 = 2*deltaY - deltaX;
          pk = p0;
          while((x1 <= x2) && (y1 <= y2))
          {
             putpixel(x1,y1,WHITE);
             if(pk < 0)
             {
                x1 = x1 + 1;
                y1 = y1;
                pk = pk + 2*deltaY;
             }
             else
             {
                x1 = x1 + 1;
                y1 = y1 + 1;
                pk = pk + (2*deltaY) - (2*deltaX);
             }

          }
       }
       else if(m >= 1)
       {
          p0 = 2*deltaX - deltaY;
          pk = p0;
          while((x1 <= x2) && (y1 <= y2))
          {
             putpixel(x1,y1,WHITE);
             if(pk < 0)
             {
                x1 = x1;
                y1 = y1 + 1;
                pk = pk + 2*deltaX;
             }
             else
             {
```

```
                x1 = x1 + 1;
                y1 = y1 + 1;
                pk = pk + (2*deltaX) - (2*deltaY);
            }
        }
    }
}

void forNegativeSlope(int x1, int y1, int x2, int y2, float m)
{    int p0, pk, deltaX, deltaY;
    deltaX = x2 - x1;
    deltaY = y2 - y1;
    if(abs(m) < 1)
    {
        p0 = (-2*deltaY) - deltaX;
        pk = p0;
        while((x1 <= x2) && (y1 >= y2))
        {
            putpixel(x1,y1,WHITE);
            if(pk < 0)
            {
                x1 = x1 + 1;
                y1 = y1;
                pk = pk - 2*deltaY;
            }
            else
            {
                x1 = x1 + 1;
                y1 = y1 - 1;
                pk = pk - (2*deltaY) - (2*deltaX);
            }
        }
    }
    else if(abs(m) >= 1)
    {
        p0 = (-2*deltaX) - deltaY;
        pk = p0;
        while((x1 <= x2) && (y1 >= y2))
        {
            putpixel(x1,y1,WHITE);
            if(pk > 0)
            {
                x1 = x1;
                y1 = y1 - 1;
                pk = pk - 2*deltaX;
            }
            else
            {
                x1 = x1 + 1;
                y1 = y1 - 1;
                pk = pk - (2*deltaY) - (2*deltaX);
            }
        }
    }
}
```

Output :



Conclusion : Program to draw a cupboard using Bresenham's Line Drawing Algorithm was successfully written and executed

Deepraj Bhosale
181105016

Experiment No : 6

Aim : Using Mid-Point Circle algorithm, draw any 5 animated facial emoji

Theory :
Since the circle is a frequently used component in pictures and graphs, a procedure for generating either full circles or circular arcs is included in many graphics packages. Hence various algorithms for drawing rasterized circles have been formulated. One such algorithm is the Mid-Point Circle algorithm. This algorithm takes advantage of the symmetry of the circle and only computes points for 1 quadrant of the circle.

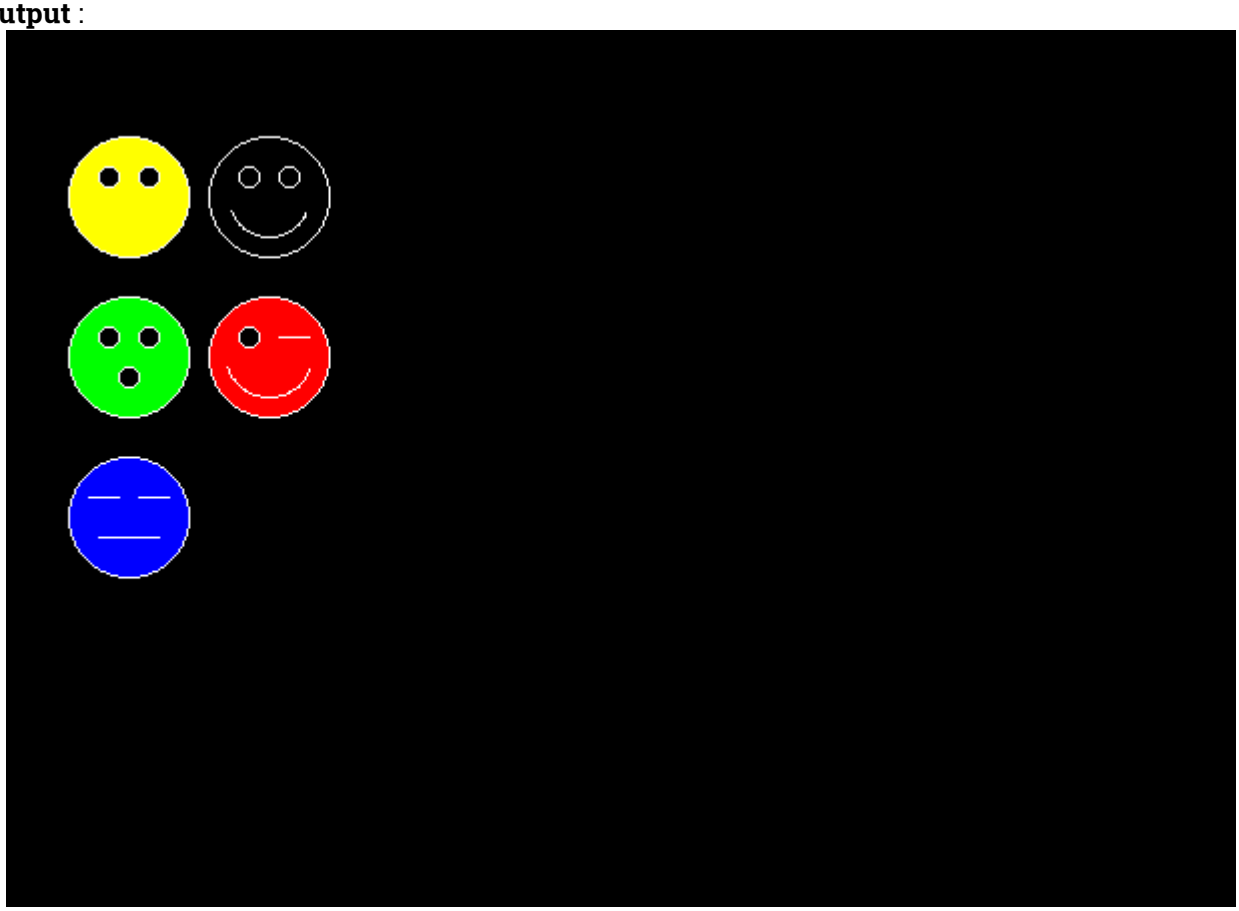
Algorithm
1. Input radius r and circle center (xc, yc), then set the coordinates for the first point on the circumference of a circle centered on the origin as (x0, y0) = (0, r)
2. Calculate the initial value of the decision parameter as p0 = 5/4 − r
3. At each xk position, starting at k = 0, perform the following test: If pk < 0, the next point along the circle centered on (0, 0) is (xk + 1, yk) and pk+1 = pk + 2xk+1 + 1
Otherwise, the next point along the circle is (xk + 1, yk − 1) and pk + 1 = pk + 2xk + 1 + 1 − 2yk + 1 where 2xk+1 = 2xk + 2 and 2yk+1 = 2yk − 2
4. Determine symmetry points in the other seven octants.
5. Move each calculated pixel position (x, y) onto the circular path

Code & Output :
```
#include<graphics.h>
#include<iostream>
using namespace std;
void mid_point_circle(int xc, int yc, int r);
void find_for_all_octants_shift(int x, int y, int xc, int yc);
void emoji_1(int i);
void emoji_2(int i);
void emoji_3(int i);
void emoji_4(int i);
void emoji_5(int i);

int main()
{
  int gd=DETECT,gm;
  initgraph(&gd,&gm,NULL);
  for(int i=0;i<=50;i++){
   cleardevice();
   emoji_1(i);
   emoji_2(i);
   emoji_3(i);
   emoji_4(i);
   emoji_5(i);
   delay(100);
  }
  closegraph();
  return 0;
}

void mid_point_circle(int xc, int yc, int r)
{
   int x=0, y=r;
   float pk = (5/4) - r;

   while(true)
```

```
    {
        find_for_all_octants_shift(x,y,xc,yc);
        cout<<endl;
        if(x >= y)
            break;
        if(pk < 0)
        {
            x = x + 1;
            y = y;
            pk = pk + (2*x) + 1;
        }
        else
        {
            x = x + 1;
            y = y - 1;
            pk = pk + (2*x) + 1 - (2*y);
        }
    }
}

void find_for_all_octants_shift(int x, int y, int xc, int yc)
{
    int x1,x2,x3,x4,x5,x6,x7,x8;
    int y1,y2,y3,y4,y5,y6,y7,y8;

    x1 = x + xc; y1 = y + yc;
    putpixel(x1,y1,WHITE);

    x2 = x + xc; y2 = (-1*y) + yc;
    putpixel(x2,y2,WHITE);

    x3 = y + xc; y3 = (-1*x) + yc;
    putpixel(x3,y3,WHITE);

    x4 = (-1*y) + xc; y4 = (-1*x) + yc;
    putpixel(x4,y4,WHITE);

    x5 = (-1*x) + xc; y5 = (-1*y) + yc;
    putpixel(x5,y5,WHITE);

    x6 = (-1*x) + xc; y6 = y + yc;
    putpixel(x6,y6,WHITE);

    x7 = (-1*y) + xc; y7 = x + yc;
    putpixel(x7,y7,WHITE);

    x8 = y + xc; y8 = x + yc;
    putpixel(x8,y8,WHITE);

}
void emoji_1(int i) // : )
{
    mid_point_circle(50+i,100,30);
    mid_point_circle(40+i,90,5);
    mid_point_circle(60+i,90,5);
    setcolor(YELLOW);
    floodfill(50+i,100,WHITE);
}

void emoji_2(int i) // :|
```

```
{
   mid_point_circle(50+i,180,30);
   mid_point_circle(40+i,170,5);
   mid_point_circle(60+i,170,5);
   mid_point_circle(50+i,190,5);
   setcolor(GREEN);
   floodfill(50+i,180,WHITE);


}

void emoji_3(int i) // -_-
{
   mid_point_circle(50+i,260,30);
   setcolor(WHITE);
   line(30+i,250,45+i,250);
   line(55+i,250,70+i,250);
   line(35+i,270,65+i,270);
   setcolor(BLUE);
   floodfill(50+i,260,WHITE);

}

void emoji_4(int i) // :')
{
   mid_point_circle(120+i,100,30);
   mid_point_circle(110+i,90,5);
   mid_point_circle(130+i,90,5);
   setcolor(BLACK);
   floodfill(120+i,100,WHITE);
   setcolor(WHITE);
   arc(120+i,100,20,160,20);
}

void emoji_5(int i) // B)
{
   mid_point_circle(120+i,180,30);
   mid_point_circle(110+i,170,5);
   setcolor(RED);
   floodfill(120+i,180,WHITE);
   setcolor(WHITE);
   line(125+i,170,140+i,170);
   arc(120+i,178,20,160,22);
}
```

Output :



Conclusion : Program to draw 5 emojis using mid point circle algorithm was successfully written and executed

Deepraj Bhosale
181105016

Experiment No : 7

Aim : Using Mid-point Ellipse algorithm, draw an animated solar system

Theory :
Ellipses are frequently used component in pictures and graphs, a procedure for generating either full ellipses or arcs is included in many graphics packages. Hence various algorithms for drawing rasterised ellipses have been formulated. One such algorithm is the Mid-Point Ellipse algorithm. This algorithm takes advantage of the symmetry of the ellipse and only computes points for 1 quadrant of the ellipse.

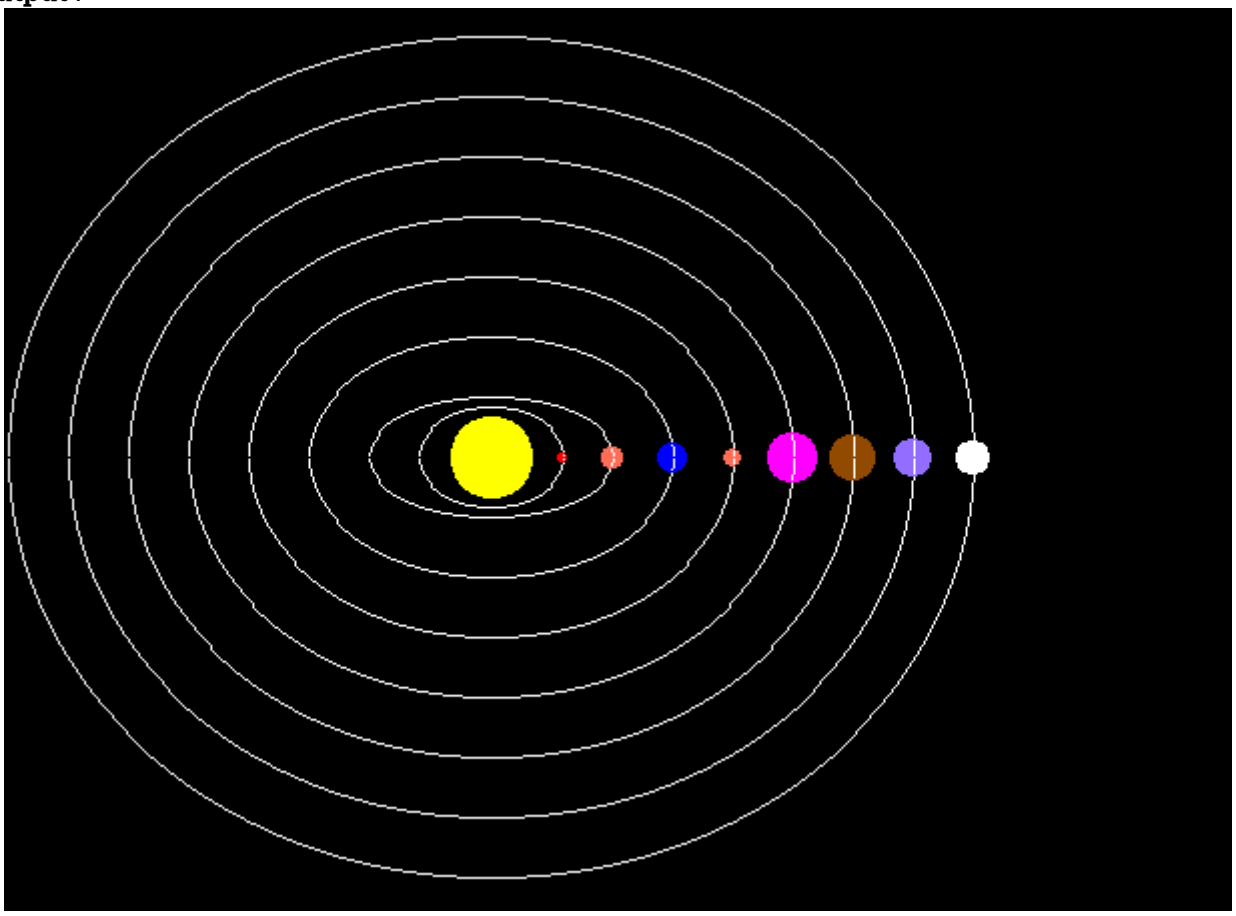
Code & Output :

```
#include<stdio.h>
#include<graphics.h>

void midPtEllipseAlgo(long x_center,long y_center,long a,long b);

int main(){
   int gd=DETECT,gm;
   initgraph(&gd,&gm,NULL);

   midPtEllipseAlgo(250,230,35,25);
   midPtEllipseAlgo(250,230,60,30);
   midPtEllipseAlgo(250,230,90,60);
   midPtEllipseAlgo(250,230,120,90);
   midPtEllipseAlgo(250,230,150,120);
   midPtEllipseAlgo(250,230,180,150);
   midPtEllipseAlgo(250,230,210,180);
   midPtEllipseAlgo(250,230,240,210);


   for(int i=0;i<12;i++)
   {
   //sun
   //setfillstyle(SOLID_FILL,YELLOW);
   setcolor(YELLOW);
   circle(250,230,20);
   floodfill(250,230,15);
   //mercury
   //setfillstyle(SOLID_FILL,RED);
   setcolor(RED);
   circle(285,230,2);
   floodfill(285,230,15);
   delay(100);
   //venus
   //setfillstyle(SOLID_FILL,LIGHTRED);
   setcolor(LIGHTRED);
   circle(310,230,5);
   floodfill(310,230,15);
   //earth
   //setfillstyle(SOLID_FILL,BLUE);
   setcolor(BLUE);
   circle(340,230,7);
   floodfill(340,230,15);
   //mars
   //setfillstyle(SOLID_FILL,LIGHTRED);
   setcolor(LIGHTRED);
   circle(370,230,4);
```

```
   floodfill(370,230,15);
   //jupiter
   //setfillstyle(SOLID_FILL,MAGENTA);
   setcolor(MAGENTA);
   circle(400,230,12);
   floodfill(400,230,15);
   //SATURN
   //setfillstyle(SOLID_FILL,BROWN);
   setcolor(BROWN);
   circle(430,230,11);
   floodfill(430,230,15);
   //uranus
   //setfillstyle(SOLID_FILL,LIGHTBLUE);
   setcolor(LIGHTBLUE);
   circle(460,230,9);
   floodfill(460,230,15);
   //neptune
   //setfillstyle(SOLID_FILL,WHITE);
   setcolor(WHITE);
   circle(490,230,8);
   floodfill(490,230,15);
   }
   getch();
  closegraph();
}

void midPtEllipseAlgo(long x_center,long y_center,long a,long b)
{
   long x,y,a_sqr,b_sqr, fx,fy, d,tmp1,tmp2;
   x=0;
   y=b;
   a_sqr=a*a;
   b_sqr=b*b;
   fx=2*b_sqr*x;
   fy=2*a_sqr*y;
  d=b_sqr-(a_sqr*b)+(a_sqr*0.25);
  do
   {
  putpixel(x_center+x,y_center+y,WHITE);
  putpixel(x_center-x,y_center-y,WHITE);
  putpixel(x_center+x,y_center-y,WHITE);
  putpixel(x_center-x,y_center+y,WHITE);

   if(d<0)
    {
  d=d+fx+b_sqr;
    }
   else
  {
  y=y-1;
  d=d+fx+-fy+b_sqr;
  fy=fy-(2*a_sqr);
  }
  x=x+1;
  fx=fx+(2*b_sqr);
  delay(10);

   }
   while(fx<fy);
   tmp1=(x+0.5)*(x+0.5);
```

```
 tmp2=(y-1)*(y-1);
 d=b_sqr*tmp1+a_sqr*tmp2-(a_sqr*b_sqr);
 do
 {
putpixel(x_center+x,y_center+y,WHITE);
putpixel(x_center-x,y_center-y,WHITE);
putpixel(x_center+x,y_center-y,WHITE);
putpixel(x_center-x,y_center+y,WHITE);

 if(d>=0)
d=d-fy+a_sqr;
 else

{
x=x+1;
d=d+fx-fy+a_sqr;
fx=fx+(2*b_sqr);
}
 y=y-1;
 fy=fy-(2*a_sqr);
 }
 while(y>0);

}
```

Output :



Conclusion : Program to draw a solar system using mid point ellipse algorithm was succesffully written and executed

Deepraj Bhosale
181105016

Experiment No : 8

Aim : To draw and colour animated boat using Boundaryfill and Floodfill Algorithm

Theory :

Boundaryfill

If the boundary of some region is specified in a single color, we can fill the interior of this region, pixel by pixel, until the boundary color is encountered. This method, called the boundary-fill algorithm, is employed in interactive painting packages, where interior points are easily selected. Using a graphics tablet or other interactive device, an artist or designer can sketch a figure outline, select a fill color from a color menu, specify the area boundary color, and pick an interior point. The figure interior is then painted in the fill color. Both inner and outer boundaries can be set up to define an area for boundary fill.

Floodfill

Sometimes we want to fill in (or recolor) an area that is not defined within a single color boundary. We can paint such areas by replacing a specified interior color instead of searching for a particular boundary color. This fill procedure is called a flood-fill algorithm. We start from a specified interior point (x, y) and reassign all pixel values that are currently set to a given interior color with the desired fill color. If the area that we want to paint has more than one interior color, we can first reassign pixel values so that all interior points have the same color. Using either a 4-connected or 8-connected approach, we then step through pixel positions until all interior points have been repainted

Algorithm

Boundaryfill

1. Add a pixel of the interior region on stack
2. If the stack is empty exit
3. Pop a pixel from stack
4. If color of pixel is not borderColor and not fillColor go to step 3
5. color the pixel to fillColor
6. push the 4 neighbours of the pixel on the stack.
7. Go to step 3

Floodfill

1. Add a pixel of the interior region on stack
2. If the stack is empty exit
3. Pop a pixel from stack
4. If color of pixel is interiorColor go to step 3
5. color the pixel to fillColor
6. push the 4 neighbours of the pixel on the stack.
7. Go to step 3

Code & Output :

```
#include <stdlib.h>
#include<graphics.h>
#include <math.h>
#include<iostream>
using namespace std;
void floodFill(int xi,int yi, int interiorColor, int fillColor){
if(getpixel(xi,yi)==interiorColor){
putpixel(xi,yi,fillColor);
floodFill(xi+1,yi,interiorColor,fillColor);
floodFill(xi-1,yi,interiorColor,fillColor);
floodFill(xi,yi+1,interiorColor,fillColor);
floodFill(xi,yi-1,interiorColor,fillColor);
}
}
void boundaryFill(int xi, int yi, int borderColor, int fillColor){
int color=getpixel(xi,yi);
if(color!=borderColor && color!=fillColor){
putpixel(xi,yi,fillColor);
boundaryFill(xi+1,yi,borderColor,fillColor);
boundaryFill(xi-1,yi,borderColor,fillColor);
boundaryFill(xi,yi+1,borderColor,fillColor);
boundaryFill(xi,yi-1,borderColor,fillColor);
}
}
int main(){
int gd=DETECT,gm;
initgraph(&gd,&gm,NULL);
setcolor(BLACK);
int x=100,y=100;
// Boat params
int slope=60,length=200,total=slope+length;
for(int i=0;i<5;i++){
setbkcolor(WHITE);
int boat[]={x,y, x+slope,y+slope, x+length,y+slope, x+total,y, x,y
};
drawpoly(5,boat);
floodFill(x+3,y+1,WHITE,BROWN);
//boundaryFill(x+3,y+1,BLACK,BROWN);
x+=15;
delay(500);
}
closegraph();
}
```

Output :

Conclusion : Program to color a boat using boundary and floodfill algorithms was successfully written and executed

Deepraj Bhosale
181105016

Experiment No : 9

Aim : To implement Cohen-Sutherland Line Clipping Algorithm.

Theory :
In computer graphics, line clipping is the process of removing lines or portions of lines outside an area of interest. Typically, any line or part there of which is outside of the viewing area is removed. A line-clipping method consists of various parts. Tests are conducted on a given line segment to find out whether it lies outside the view volume. Afterwards, intersection calculations are carried out with one or more clipping boundaries. Determining which portion of the line is inside or outside of the clipping volume is done by processing the endpoints of the line with regards to the intersection.

Algorithm
1. Initially, every line endpoint in a picture is assigned a four-digit binary value, called a region code, and each bit position is used to indicate whether the point is inside or outside one of the clipping-window boundaries
2. Using region codes determine which lines are completely outside the clipping boundary and eliminate them.
3. Using region codes determine which lines are completely inside the clipping boundary and save them.
4. To determine whether a line crosses a selected clipping boundary, we can check corresponding bit values in the two endpoint region codes. If one of these bit values is 1 and the other is 0, the line segment crosses that boundary.
5. Use line equations to determine intersection of line and boundary.
6. Clip the line to the intersection point and save it

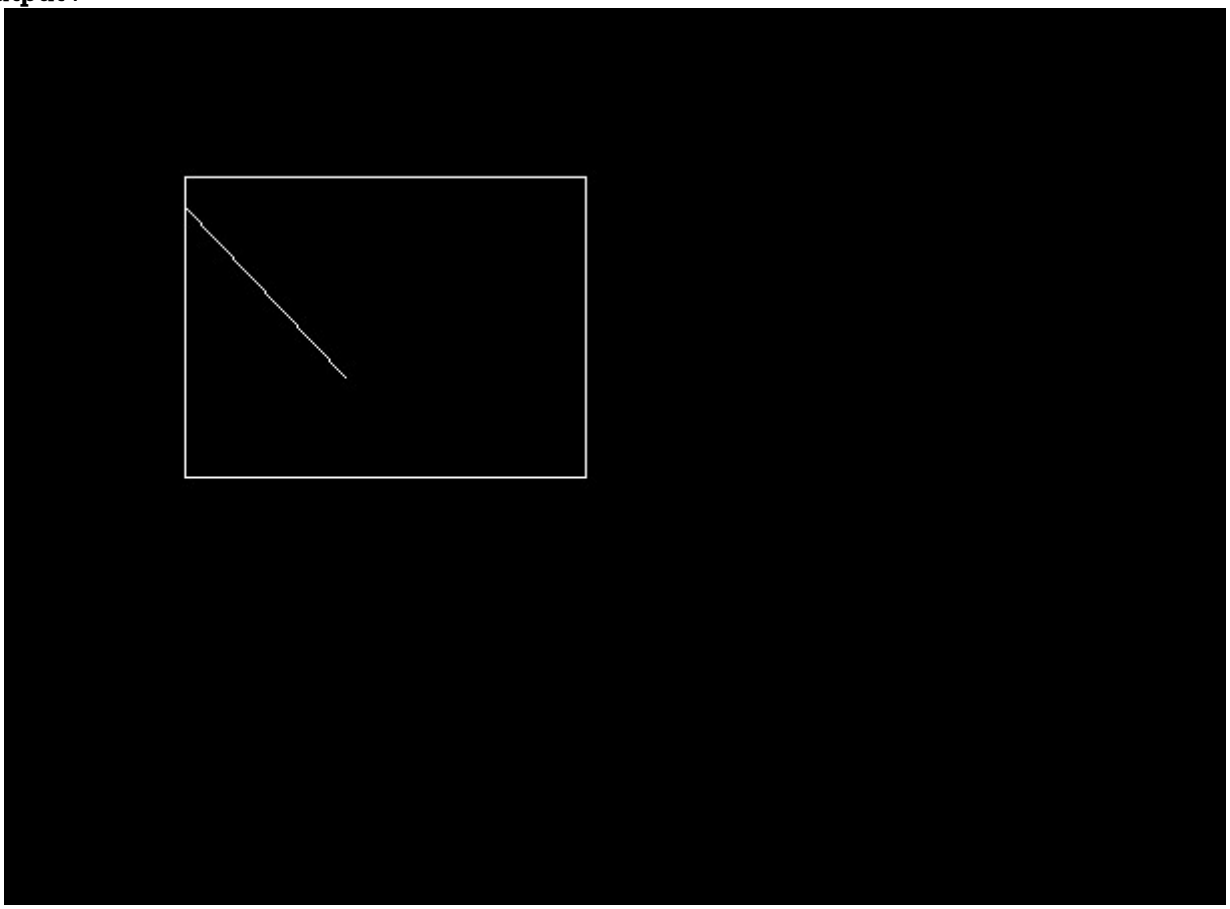
Code & Output :
```
#include<graphics.h>
#include<iostream>
using namespace std;
void get_region_code(float x1, float y1, float x2, float y2, int rc0[], int rc1[]);
void cohen_sutherland(float x1, float y1, float x2, float y2);
void clip_top(float &x,float &y, float a, float b);
void clip_bottom(float &x,float &y, float a, float b);
void clip_right(float &x,float &y, float a, float b);
void clip_left(float &x,float &y, float a, float b);
bool find_and_operation(int rc0[], int rc1[]);
float m = 0;
int XWmin = 100;
int YWmin = 100;
int XWmax = 300;
int YWmax = 250;

int main()
{
  float x1,y1,x2,y2;

  cout<<"Program to demonstrate Cohen Sutherland Line clipping algorithm"<<endl;
  cout<<endl;

  cout<<"Enter x1 : ";
  cin>>x1;
  cout<<"Enter y1 : ";
  cin>>y1;
  cout<<endl;
```

```
  cout<<"Enter x2 : ";
  cin>>x2;
  cout<<"Enter y2 : ";
  cin>>y2;
  m = (y2 - y1)/(x2 - x1);
  cohen_sutherland(x1,y1,x2,y2);


  return 0;
}

void cohen_sutherland(float x1,float y1, float x2, float y2)
{
   int rc0[4];
   int rc1[4];
   float a0,b0,a1,b1;
   a0 = x1;
   b0 = y1;
   a1 = x2;
   b1 = y2;
   get_region_code(a0,b0,a1,b1,rc0,rc1);

if((rc0[0]==0)&&(rc0[1]==0)&&(rc0[2]==0)&&(rc0[3]==0)&&(rc1[0]==0)&&(rc1[1]==0)&&(rc1[2]==0)&&(rc1[
3]==0))
   {
      int gd=DETECT,gm;
      initgraph(&gd,&gm,NULL);
      line(100,100,100,250); // Xmin = 100
      line(100,100,300,100); // Ymin = 100
      line(100,250,300,250); // Xmax = 300
      line(300,100,300,250); // Ymax = 250

      line(a0,b0,a1,b1);

      delay(1000000);
      closegraph();
   }
   else
   {
      if(find_and_operation(rc0,rc1)==true)
      {
         cout<<endl;
         cout<<"Line is discarded"<<endl;
      }
      else
      {
         if((rc0[0] != 0) || (rc0[1] != 0) || (rc0[2] != 0) || (rc0[3] != 0))
         {
            if(rc0[3] != 0)
            {
               clip_top(a0,b0,a1,b1);
            }
            else if(rc0[2] != 0)
            {
               clip_bottom(a0,b0,a1,b1);
            }
            else if(rc0[1] != 0)
            {
               clip_right(a0,b0,a1,b1);
            }
```

```
            else if(rc0[0] != 0)
            {
               clip_left(a0,b0,a1,b1);
            }
         }
         if((rc1[0] != 0) || (rc1[1] != 0) || (rc1[2] != 0) || (rc1[3] != 0))
         {
            if(rc1[3] != 0)
            {
               clip_top(a1,b1,a0,b0);
            }
            else if(rc1[2] != 0)
            {
               clip_bottom(a1,b1,a0,b0);
            }
            else if(rc1[1] != 0)
            {
               clip_right(a1,b1,a0,b0);
            }
            else if(rc1[0] != 0)
            {
               clip_left(a1,b1,a0,b0);
            }
         }

         cohen_sutherland(a0,b0,a1,b1);
      }
   }
}

void clip_top(float &x,float &y, float a, float b)
{
   y = YWmax;
   x = a + ((y-b)/m);
}

void clip_bottom(float &x,float &y, float a, float b)
{
   y = YWmin;
   x = a + ((y-b)/m);
}

void clip_right(float &x,float &y, float a, float b)
{
   x = XWmax;
   y = b + m*(x-a);
}

void clip_left(float &x,float &y, float a, float b)
{
   x = XWmin;
   y = b + m*(x-a);
}

bool find_and_operation(int rc0[], int rc1[])
{
   if((rc0[0]==1)&&(rc1[0]==1))
      return true;
   else if((rc0[1]==1)&&(rc1[1]==1))
      return true;
```

```
    else if((rc0[2]==1)&&(rc1[2]==1))
        return true;
    else if((rc0[3]==1)&&(rc1[3]==1))
        return true;
    else
        return false;

}

void get_region_code(float x1, float y1, float x2, float y2, int rc0[], int rc1[]){   // For first point
    if(x1 - XWmin >= 0.0)
      rc0[0] = 0;
    else
      rc0[0] = 1;

    if(XWmax - x1 >= 0.0)
      rc0[1] = 0;
    else
      rc0[1] = 1;

    if(y1 - YWmin >= 0.0)
      rc0[2] = 0;
    else
      rc0[2] = 1;

    if(YWmax - y1 >= 0.0)
      rc0[3] = 0;
    else
      rc0[3] = 1;

    //For second point
    if(x2 - XWmin >= 0.0)
      rc1[0] = 0;
    else
      rc1[0] = 1;

    if(XWmax - x2 >= 0.0)
      rc1[1] = 0;
    else
      rc1[1] = 1;

    if(y2 - YWmin >= 0.0)
      rc1[2] = 0;
    else
      rc1[2] = 1;

    if(YWmax - y2 >= 0.0)
      rc1[3] = 0;
    else
      rc1[3] = 1;

}
```

Output :



Conclusion : Program to implement Cohen-Sutherland line clipping was successfully written and executed

Deepraj Bhosale
181105016

Experiment No : 10

Aim : To draw a polygon and clip it using Sutherland Hodgeman Polygon Clipping Algorithm

Theory :
Polygon clipping is the process of finding the portion of a give convex polygon inside of a given convex polygon. It is widely used in 2D and 3D animation. The Sutherland Hodgeman Polygon Clipping Algorithm is one such algorithm.

Algorithm :
1. Begin with an input list of all vertices in the subject polygon
2. One side of the clipping boundary is extended infinitely in both directions, and the path of the subject polygon is traversed.
3. Vertices from the input list are inserted into an output list if they lie on the visible side of the extended clip polygon line, and new vertices are added to the output list where the subject polygon path crosses the extended clip polygon line.

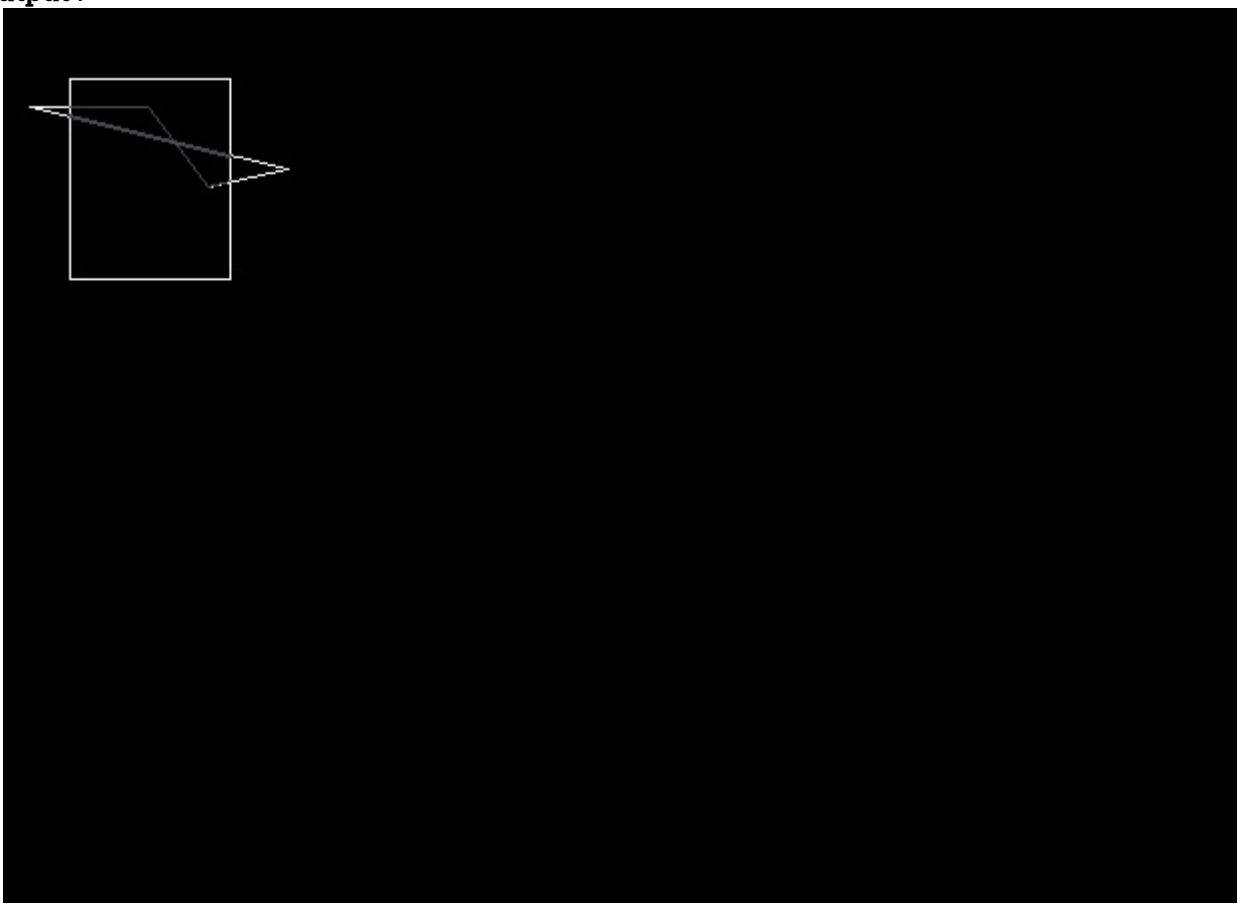
Code & Output :
```
#include <math.h>
#include<iostream>
#include<graphics.h>
#define ROUND(a) ((int)(a+0.5))
#define n 4

#define LEFT_EDGE 0x1
#define RIGHT_EDGE 0x2
#define BOTTOM_EDGE 0x4
#define TOP_EDGE 0x8

#define INSIDE(a)  (!a)
#define REJECT(a,b) (a&b)
#define ACCEPT(a,b) (!(a|b))

typedef struct wcpt2
{
   int x,y;
}wcpt2;

typedef struct dcpt
{
   int x,y;
}dcpt;

int main()
{
   int gd=DETECT,gm;
   int left,top,right,bottom;
   int x1,x2,y1,y2;
   int maxx, maxy;

   int poly[10];
   void clipline(dcpt,dcpt,wcpt2,wcpt2);

   initgraph(&gd,&gm,NULL);
   maxx = getmaxx()/4;
   maxy = getmaxy()/4;
```

```
    poly[0] = 20;
    poly[1] = maxy / 2;

    poly[2] = maxx - 10;
    poly[3] = 90;

    poly[4] = maxx - 50;
    poly[5] = maxy - 20;

    poly[6] = maxx / 2;
    poly[7] = maxy / 2;


    poly[8] = poly[0];
    poly[9] = poly[1];

    drawpoly(5, poly);

    rectangle(40,45,120,145);
    wcpt2 pt1,pt2;
    dcpt winmin,winmax;

    winmin.x=40;
    winmin.y=45;
    winmax.x=120;
    winmax.y=145;

    pt1.x=20;
    pt1.y=maxy/2;
    pt2.x=maxx-10;
    pt2.y=10;

    int i=0;
    for(int index=0;index<n;index++)
    {
       if(index==n-1)
       {
          pt1.x=poly[i];
          pt1.y=poly[i+1];
          i=0;
          pt2.x=poly[i];
          pt2.y=poly[i+1];
          clipline(winmin,winmax,pt1,pt2);
       }
       else
       {
          pt1.x=poly[i];
          pt1.y=poly[i+1];
          pt2.x=poly[i+2];
          pt2.y=poly[i+3];
          clipline(winmin,winmax,pt1,pt2);
       }
       i+=2;
    }
    pt1.x=poly[i];
    pt1.y=poly[i+1];
    clipline(winmin,winmax,pt1,pt2);
    getch();
}
```

```
unsigned char encode(wcpt2 pt,dcpt winmin,dcpt winmax)
{
   unsigned char code=0x00;
   if(pt.x < winmin.x)
      code=code | LEFT_EDGE;
   if(pt.x > winmax.x)
      code=code | RIGHT_EDGE;
   if(pt.y < winmin.y)
      code=code | TOP_EDGE;
   if(pt.y > winmax.y)
      code=code | BOTTOM_EDGE;
   return code;
}


void swappts(wcpt2 *p1,wcpt2 *p2)
{
   wcpt2 tmp;
   tmp = *p1;
   *p1 = *p2;
   *p2 = tmp;
}


void swapcode(unsigned char *c1,unsigned char *c2)
{
   unsigned char tmp;
   tmp = *c1;
   *c1 = *c2;
   *c2 = tmp;
}


void clipline(dcpt winmin,dcpt winmax,wcpt2 p1,wcpt2 p2)
{
   unsigned char encode(wcpt2,dcpt,dcpt);
   unsigned char code1,code2;
   int done = 0 , draw = 0;
   float m;
   void swapcode(unsigned char *c1,unsigned char *c2);
   void swappts(wcpt2 *p1,wcpt2 *p2);

   while(!done)
   {
      code1 = encode(p1,winmin,winmax);
      code2 = encode(p2,winmin,winmax);
      if(ACCEPT(code1,code2))
      {
         draw = 1;
         done = 1;
      }
      else if(REJECT(code1,code2))
         done = 1;
      else if(INSIDE(code1))
      {
            swappts(&p1,&p2);
            swapcode(&code1,&code2);
      }
      if(code1 & LEFT_EDGE)
```

```
        {
            p1.y += (winmin.x - p1.x) *  (p2.y - p1.y) / (p2.x - p1.x);
            p1.x = winmin.x;
        }
        else if(code1 & RIGHT_EDGE)
        {
            p1.y += (winmax.x - p1.x) *  (p2.y - p1.y) / (p2.x - p1.x);
            p1.x = winmax.x;
        }
        else if(code1 & TOP_EDGE)
        {
           if(p2.x != p1.x)
            p1.x += (winmin.y - p1.y) *  (p2.x - p1.x) / (p2.y - p1.y);
            p1.y = winmin.y;
        }
        else if(code1 & BOTTOM_EDGE)
        {
           if(p2.x != p1.x)
            p1.x += (winmax.y - p1.y) *  (p2.x - p1.x) / (p2.y - p1.y);
            p1.y = winmax.y;
        }
    }
    if(draw)
    {
    setcolor(8);
    line(p1.x,p1.y,p2.x,p2.y);
    }
}
```

Output :



Conclusion : Program to draw a polygon and clip it using Sutherland Hodgeman Polygon Clipping Algorithm was successfully written and executed

Deepraj Bhosale
181105016

Experiment No : 11

Aim : To implement the 2D transformation on a polygon.

Theory :
Geometric Transformations :
Operations that are applied to the geometric description of an object to change its position, orientation, or size are called geometric transformations. The geometric-transformation functions that are available in all graphics packages are those for translation, rotation, and scaling. Other useful transformation routines that are sometimes included in a package are reflection and shearing operations. In computer graphics, geometric transforms are represented as matrices that the original points are multiplied by or added to.

Homogeneous Coordinates :
Homogeneous coordinates are ubiquitous in computer graphics because they allow common vector operations such as translation, rotation, scaling and perspective projection to be represented as a matrix by which the vector is multiplied. By the chain rule, any sequence of such operations can be multiplied out into a single matrix, allowing simple and efficient processing. By contrast, using Cartesian coordinates, translations and perspective projection cannot be expressed as matrix multiplications, though other operations can. Modern OpenGL and Direct3D graphics cards take advantage of homogeneous coordinates to implement a vertex shader efficiently using vector processors with 4-element registers.

Code & Output :
→ Matrix

```
#include<iostream>

using namespace std;
class matrix;
matrix matmul(matrix a, matrix b);

class matrix{
	public:
	int i,j;
	float **values;

	matrix(int I, int J){
		i=I;j=J;
		int x,y;
		values=(float**)malloc(sizeof(float*)*i);
		for(x=0;x<i;x++){
			values[x]=(float*)malloc(sizeof(float)*j);
			for(y=0;y<j;y++){
				values[x][y]=0;
			}
		}
	}

	matrix(int I, int J, float vals[]){
		i=I;
		j=J;
		int x,y;
		values=(float**)malloc(sizeof(float*)*i);
		for(x=0;x<i;x++){
			values[x]=(float*)malloc(sizeof(float)*j);
			for(y=0;y<j;y++){
```

```
                                        values[x][y]=vals[x*j+y];
                                }
                        }
                }
                void display(){
                        int x,y;
                        for(x=0;x<i;x++){
                                for(y=0;y<j;y++){
                                        cout<<values[x][y]<<", ";
                                }
                                cout<<"\b\b"<<endl;
                        }
                }
                float* operator[](int index){
                        return values[index];
                }
};

matrix matmul(matrix a, matrix b){
        if(a.j!=b.i){
                cout<<"Matmul error: Matrices have incorrect dimensions"<<endl;
                exit(0);
        }
        matrix c(a.i,b.j);
        for (int i = 0; i < a.i; i++)
                for (int k = 0; k < a.j; k++)
                        for (int j = 0; j < b.j; j++)
                                c.values[i][j] += a.values[i][k] * b.values[k][j];
        return c;
}
```

→ **Transformations**

```
#include <iostream>
#include<math.h>
#include <list>
#include<graphics.h>
#include "matrix.h"

#define PI 3.14159
#define to_rad(x)  ((x/180)*PI)
using namespace std;

int thickness=4;

void eg(){
        int i=5,j=2;
        float values[]={1,2,3,4,5,6,7,8,9,10};
        matrix a=       matrix(i,j,values);
        matrix b=       matrix(j,i,values);
        /* matmul(a,b).display(); */
}
typedef struct Coordinates{
        int x,y;
}Coordinates;

typedef struct Point{
        matrix h; // homogenous matrix representation
        Point(int x,int y):h(3,1){
                h[0][0]=x; h[1][0]=y; h[2][0]=1;
        }
```

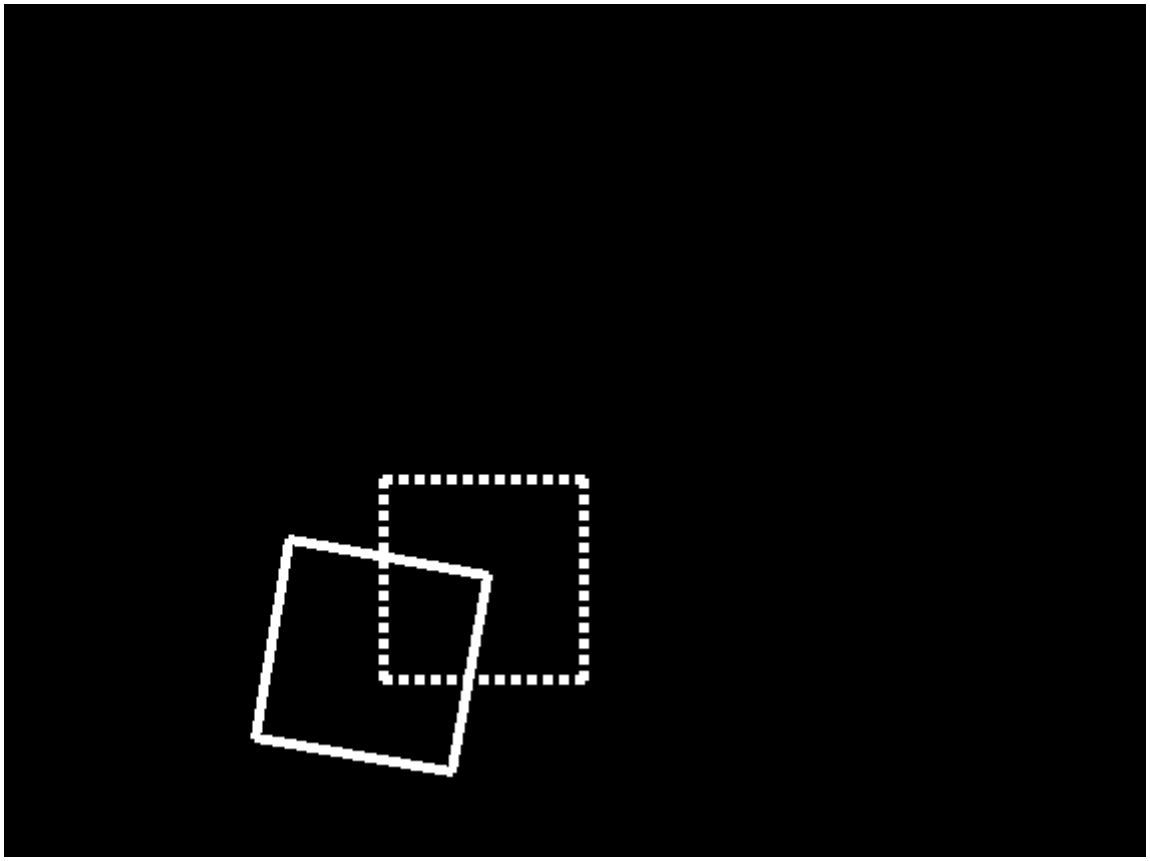
```
	Point(matrix m):h(3,1){
		if(m.i!=3 || m.j!=1){
			cout<<"Cannot convert matrix with dimensions ";
			cout<<m.i<<" "<<m.j<<"into Point(x,y)"<<endl;
			exit(1);
		}
		h[0][0]=m[0][0]; h[1][0]=m[1][0]; h[2][0]=1;
	}
	Coordinates getCoor(){
		return {(int)h[0][0],(int)h[1][0]};
	}
	void display(){
		Coordinates p=this->getCoor();
		cout<<p.x<<" "<<p.y;
	}
}Point;


typedef struct Polygon{
	int sides;
	list<Point> points;
	Polygon(){
		sides=0;
	}
	Polygon(int n, int polypoint[]){
		int i;
		for(i=0;i<n*2;i+=2){
			points.emplace_back(polypoint[i],polypoint[i+1])	;
		}
	}
	void display(){
		Coordinates c;
		for(Point p:points){
			c=p.getCoor();
			cout<<c.x<<" "<<c.y<<endl;
		}
	}
	void draw(int style=SOLID_LINE){
		Coordinates start,end;
		list<Point>::iterator it=points.begin();
		setlinestyle(style,2,thickness)	;
		start=it->getCoor();
		it++;
		while(it!=points.end()){
			end=it->getCoor();
			line(start.x,start.y,end.x,end.y);
			/* cout<<(start.x)<<" "<<start.y<<endl; */
			/* cout<<end.x<<" "<<end.y<<endl; */
			/* cout<<endl; */
			start=end;
			it++;
		}
		end=points.front().getCoor();
		line(start.x,start.y,end.x,end.y);
		/* cout<<start.x<<" "<<start.y<<endl; */
		/* cout<<end.x<<" "<<end.y<<endl<<endl; */
	}
}Polygon;

Polygon translate(Polygon poly, float tx, float ty){
```

```
	Polygon result=Polygon();
	float values[]={
		1,0,tx,
		0,1,ty,
		0,0,1
	};
	matrix transMatrix=matrix(3,3,values);

	for(Point p:poly.points){
		result.points.push_back(matmul(transMatrix,p.h));
	}
	return result;
	/* return Point(matmul(transMatrix,p.h)); */
}

Polygon rotate(Polygon poly, float theta){
	theta=to_rad(theta);
	Polygon result=Polygon();
	float values[]={
		cos(theta),-sin(theta),0,
		sin(theta),cos(theta),0,
		0,	0,	1
	};

	matrix rotMatrix=matrix(3,3,values);
	for(Point p:poly.points){
		result.points.push_back(matmul(rotMatrix,p.h));
	}
	return result;
	/* return Point(matmul(transMatrix,p.h)); */
}

Polygon scale(Polygon poly, float sx, float sy){
	Polygon result=Polygon();
	float values[]={
		sx,0,0,
		0,sy,0,
		0,0,1
	};
	matrix transMatrix=matrix(3,3,values);

	for(Point p:poly.points){
		result.points.push_back(matmul(transMatrix,p.h));
	}
	return result;
}

Polygon pivot(Polygon poly, float xr, float yr, float theta){
	return translate(rotate(translate(poly,-xr, -yr),theta),xr,yr);
}

Polygon fixedPointScale(Polygon poly, float xr, float yr, float sx, float sy){
	return translate(scale(translate(poly,-xr, -yr),sx,sy),xr,yr);
}

Polygon reflect(Polygon poly, float xr,float yr, float theta){
	Polygon result=Polygon();
	theta=to_rad(theta);
	Polygon p1=rotate(translate(poly,-xr,-yr),theta);
	float values[]={
```

```
                1,0,0,
                0,-1,0,
                0,0,1
        };
        matrix refMatrix=matrix(3,3,values);
        for(Point p:p1.points){
                result.points.push_back(matmul(refMatrix,p.h));
        }
        return translate(rotate(result,-theta),xr,yr);
}
Polygon shearx(Polygon poly,float yref,float shx){
        Polygon result=Polygon();
        float values[]={
                1,shx,-shx*yref,
                0,1,0,
                0,0,1
        };
        matrix shearMatrix=matrix(3,3,values);
        for(Point p:poly.points){
                result.points.push_back(matmul(shearMatrix,p.h));
        }
        return result;
}
Polygon sheary(Polygon poly,float xref,float shy){
        Polygon result=Polygon();
        float values[]={
                1,0,0,
                shy,1,-shy*xref,
                0,0,1
        };
        matrix shearMatrix=matrix(3,3,values);
        for(Point p:poly.points){
                result.points.push_back(matmul(shearMatrix,p.h));
        }
        return result;
}
int main(){
        int gd=DETECT,gm;
        initgraph(&gd,&gm,NULL);
        setcolor(WHITE);
        int n=4;
        int dt=3000;
        int polypoint[]={100,200, 100,300, 200,300, 200,200};
        Polygon p=Polygon(n,polypoint);
        p.draw(DASHED_LINE);
        delay(dt/2);
        p=translate(p, 100, 50);
        p.draw();
        delay(dt);
        setbkcolor(BLACK);
        p.draw(DASHED_LINE);

        delay(dt/2);
        Polygon p1=rotate(p,10);
        p1.draw();
        delay(dt);
        setbkcolor(BLACK);

        Polygon p2=scale(p1,1.2,1.2);
        p1.draw(DASHED_LINE);
```

```
        delay(dt/2);
        p2.draw();
        delay(dt);
        setbkcolor(BLACK);


        Polygon p3=reflect(p2,0,320,0);
        p2.draw(DASHED_LINE);
        delay(dt/2);
        p3.draw();
        delay(dt);
        setbkcolor(BLACK);

        p3.draw(DASHED_LINE);
        delay(dt/2);
        Polygon p4=shearx(p3,90,0.5);
        p4.draw();
        delay(dt);
        setbkcolor(BLACK);


        p3.draw(DASHED_LINE);
        Polygon p5=sheary(p3,90,0.5);
        delay(dt/2);
        p5.draw();
        delay(dt);
        setbkcolor(BLACK);

        // shearx(p,200,0.5).draw();
        // sheary(p,100,0.5).draw();
        closegraph();
}
```

Output :



Conclusion : Program to implement 2D transformation on a polygon was successfully written and executed

Deepraj Bhosale
181105016