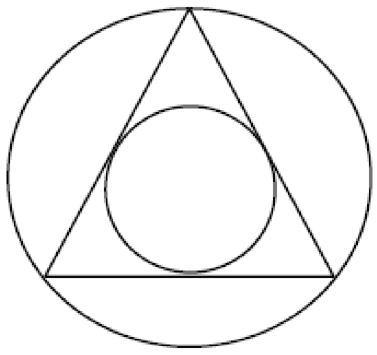
Assignment No:5

Title/ Problem Statement:

Write C++/Java program to draw inscribed and Circumscribed circles in the triangle as shown as an example below. (Use any Circle drawing and Line drawing algorithms)



Objectives:

1. To understand Bresenham‘s circle drawing algorithms used for computer graphics.
2. To understand Equilateral Triangle concept

Theory :

In computer graphics, the midpoint circle algorithm is an algorithm used to determine the points needed for drawing a circle. The algorithm is a variant of Bresenham's line algorithm, and is thus sometimes known as Bresenham's circle algorithm, although not actually invented by Jack E. Bresenham. The Midpoint circle drawing algorithm works on the same midpoint concept as the Bresenham’s line algorithm. In the Midpoint circle drawing algorithm, you determine the next pixel to be plotted based on the position of the midpoint between the current and next consecutive pixel.

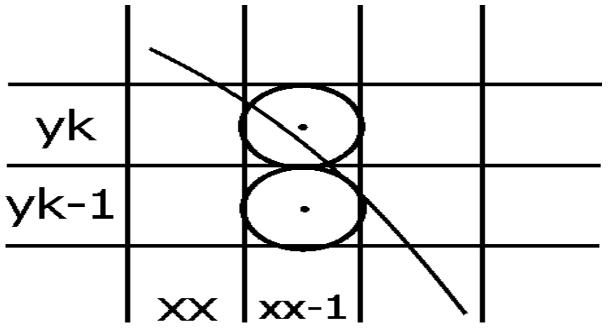


Fig 6.1 Midpoint circle Algorithm

Platform Required:

- Microsoft Windows 7/ Windows 8 Operating System onwards or 64-bit Open source Linux or its derivative.

Algorithm :

Bresenham's Algorithm to Draw a Circle.

1. Start.
2. Declare variables x,y,p and also declare gdriver=DETECT,gmode.
3. Initialise the graphic mode with the path location in TC folder.
4. Input the radius of the circle r.
5. Load x-0,y=r,initial decision parameter p=1-r.so the first point is (0,r).
6. Repeat Step 7 while (x<y) and increment x-value simultaneously.

7. If (p>0),

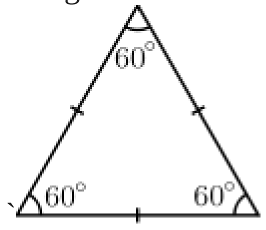
   do p=p+2*(x-y)+1.

8. Otherwise
   p=p+2*x+1 and
   y is decremented simultaneously.
9. Then calculate the value of the function circlepoints() with p.arameters (x,y).
10. Place pixels using putpixel at points (x+300,y+300) in specified colour in

    circlepoints() function shifting the origin to 300 on both x-axis and y-axis.
11. Close Graph.
12. Stop.

Equilateral triangle

In geometry, an equilateral triangle is a triangle in which all three sides are equal. In the familiar Euclidean geometry, equilateral triangles are also equiangular; that is, all three internal angles are also congruent to each other and are each 60°. They are regular polygons, and can therefore also be referred to as regular triangles.



Equal cevians
Three kinds of cevians are equal for (and only for) equilateral triangles:

- The three altitudes have equal lengths.
- The three medians have equal lengths.
- The three angle bisectors have equal lengths.

Coincident triangle centers

Every triangle center of an equilateral triangle coincides with its centroid, which implies that the equilateral triangle is the only triangle with no Euler line connecting some of the centers. For some pairs of triangle centers, the fact that they coincide is enough to ensure that the triangle is equilateral. In particular:

- A triangle is equilateral if any two of the circumcenter, incenter, centroid, or orthocenter coincide.
- It is also equilateral if its circumcenter coincides with the Nagel point, or if its incenter coincides with its nine-point center.

Six triangles formed by partitioning by the medians

For any triangle, the three medians partition the triangle into six smaller triangles.

- A triangle is equilateral if and only if any three of the smaller triangles have either the same perimeter or the same inradius.
- A triangle is equilateral if and only if the circumcenters of any three of the smaller triangles have the same distance from the centroid.

Algorithm:

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

class Pixel
{
  public:
     void plotPixel(int x,int y)
     {
           putpixel(x,y,WHITE);
     }
};


class Line : public Pixel
{
public:
     void drawLine(int x1,int y1,int x2,int y2,int d)//dda
     {
           int steps,i;
           float dx,dy,xinc,yinc,x,y;
           dx = abs(x2-x1);
           dy = abs(y2-y1);
           if(dx>dy) //gentle
           {
                 steps = dx;
                         xinc = dx/steps;
                 yinc = dy/steps;
                    plotPixel(x1,y1);
                 x = x1;
                 y = y1;
                 for(i=1 ;i <= steps; i++)
```

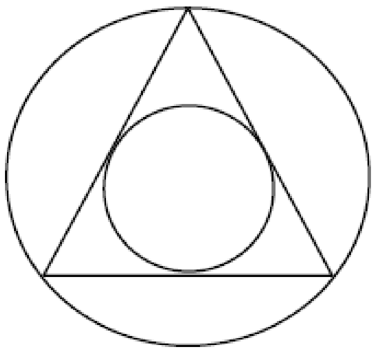
```
            {
              if(x1<x2)  //forward
              {
                         x = x + xinc;
                         if(y1>y2)
                     y = y - yinc;
                 else
                   y = y + yinc;
                 plotPixel(x,y+0.5);
              }
              else   //backward
                 {
                 x = x - xinc;
                         if(y1>y2)
                      y = y - yinc;
                 else
                    y = y + yinc;
                 plotPixel(x,y+0.5);
                      }
              delay(50);
            }

       }
       else   //steep
       {
            steps = dy;
                    xinc = dx/steps;
            yinc = dy/steps;
              plotPixel(x1,y1);
            x = x1;
            y = y1;
            for(i=1 ;i <= steps; i++)
            {
              if(y1<y2)  //forward
              {
                         if(x1<x2)
                             x = x + xinc;
                 else
                   x = x - xinc;
                 y = y + yinc;
                 plotPixel(x+0.5,y);
              }
              else   //backward
                 {
                        if(x1<x2)
                  x = x + xinc;
                else
                    x = x - xinc;
                 y = y - yinc;
                 plotPixel(x+0.5,y);
                      }
              delay(50);
            }
       }
}//end
void drawLine(int x1,int y1,int x2,int y2,char d)//bresenham
{
```

```
int x,y,i;
float dx,dy,g;
x = x1;
y = y1;
plotPixel(x,y);
dx = abs(x2-x1);
dy = abs(y2-y1);

if(dx>dy) //gentle
{
        g = 2*dy-dx;
      for(i = 1; i<= dx ; i++)
      {
            if(x1<x2)  //forward
            {
                              x = x + 1;
                  if(g>0)
                  {
                                          if(y1<y2)
                           y = y + 1;
                        else
                           y = y - 1;
                        g = g + 2 * dy - 2 * dx;
                  }
                  else
                  {
                                          g = g + 2 * dy;
                  }
                  plotPixel(x,y);
                  delay(50);
            }
            else  //backward
            {
                  x = x - 1;
                  if(g>0)
                  {
                        y = y - 1;
                        g = g + 2 * dy - 2 * dx;
                  }
                  else
                  {
                                          g = g + 2 * dy;
                  }
                  plotPixel(x,y);
                  delay(50);
            }

      }

}
else  //steep
{
        g = 2*dx-dy;
      for(i = 1; i<= dy ; i++)
      {
            if(y1<y2)  //forward
            {
```

```
                                     y = y + 1;
                         if(g>0)
                         {
                                             if(x1<x2)
                                  x = x + 1;
                              else
                                  x = x - 1;
                              g = g + 2 * dx - 2 * dy;
                         }
                         else
                         {
                                             g = g + 2 * dx;
                         }
                         plotPixel(x,y);
                         delay(50);
                    }
                    else  //backward
                    {
                         y = y - 1;
                         if(g>0)
                         {
                                             if(x1<x2)
                                 x = x + 1;
                              else
                                 x = x - 1;
                              g = g + 2 * dx - 2 * dy;
                         }
                         else
                         {
                                             g = g + 2 * dx;
                         }
                         plotPixel(x,y);
                                     delay(50);
                    }

               }
          }

     }//end

};

class MyCircle : public Pixel
{
public:
     void drawCircle(int xc,int yc,int r)
     {
          int x,y;
          float s;
          x = 0;
          y = r;
          s = 3-2*r;
          while(x < y)
          {
               if(s<=0)
               {
                    s = s + 4 * x + 6;
                    x = x + 1;
                }
                else
                {
                    s = s + 4 * (x-y) + 10;
                    x = x + 1;
                    y = y - 1;
                }
                display(xc,yc,x,y);
                delay(100);
            }

        }
        void display(int xc,int yc,int x,int y)
        {
            plotPixel(xc+x,yc+y);
            plotPixel(xc+y,yc+x);
            plotPixel(xc+y,yc-x);
            plotPixel(xc+x,yc-y);
            plotPixel(xc-x,yc-y);
            plotPixel(xc-y,yc-x);
            plotPixel(xc-y,yc+x);
            plotPixel(xc-x,yc+y);
        }
};

int main()
{
            int gd = DETECT, gm,x1,x2,y1,y2,xc,yc,r;
            float x3,y3,xm,ym,t1,t2,t,d;
            initgraph(&gd,&gm,NULL);
            Line l;
            MyCircle c;
            x1=200;y1=200;d=200;
            t=(sqrt(3)/2)*d;
            xm=x1;
            ym=y1+t;
            x2=xm-(d/2);
            y2=ym;
            x3=xm+(d/2);
            y3=ym;
            xc=(x1+x2+x3)/3;
            yc=(y1+y2+y3)/3;
            t1=sqrt(pow(xm-xc,2)+pow(ym-yc,2));
            t2=t-t1;
            l.drawLine(x2,y2,x1,y1,1);
            l.drawLine(x2,y2,x3,y3,1);
            l.drawLine(x1,y1,x3,y3,1);
            c.drawCircle(xc,yc,t2);
            c.drawCircle(xc,yc,t1);
            getch();
            closegraph();
            return 0;

}
```

Input: length of edge of triangle and starting point of line.

Output:



Conclusion: Thus we have Studied inscribed and Circumscribed circles in the triangle.