Computer Graphics 2004 p5q5

(a)(i) The trick is to generate one octant by a simple algorithm, and then to reflect the generated pixels into the other seven octants. So, if the single octant algorithm requires you to draw pixel (x, y) you instead draw all eight pixels $(\pm x, \pm y)$ $(\pm x, \mp y)$, $(\pm y, \pm x)$, $(\pm y, \mp x)$

This algorithm generates pixels in the second octant and then does the above to generate the other octant. It assumes that the origin is at the centre of a pixel.

```
int x = 0;
int y = R;
```
} start at (0, R)

~~drawPixel (0, R);~~
~~drawPixel (0, -R);~~
~~drawPixel (R, 0);~~
~~drawPixel (-R, 0);~~
} ~~draw the first pixels in all eight quadrants (in this case, pairs of them are equal)~~

$d = (x+1)^2 + (y-\frac{1}{2})^2 - R^2;$ } set up a decision variable at $(x+1, y-\frac{1}{2})$. If $d > 0$ this point is outside the circle. If $d < 0$ it is inside the circle. This variable is used to select whether the next pixel to turn on is $(x+1, y)$ or $(x+1, y-1)$.

```
while ( x ≤ y ) {
    drawPixel ( x, y);
    drawPixel ( y, x);
    drawPixel (-x, y);
    drawPixel (-y, x);
    drawPixel (-x, -y);
    drawPixel (-y, -x);
    drawPixel ( x, -y);
    drawPixel ( y, -x);
```
} stop once we have passed the 45° line (& don't draw any pixels past that line).

} draw all eight pixels

```
if (d < 0)                          } d<0 ⇒ move one pixel right
then {
        d = d + 2x - 3;             } update d to be (x+2)^2 + (y-½)^2 - R^2
     }
else {
        d = d + 2x - 2y - 1;        } update d to be (x+2)^2 + (y-3/2)^2 - R^2
        y = y - 1;                  } update y
     }
  x = x + 1;                        } update x
}                                   } end of while loop.
```

That's it. One can make it even simpler by calculating $(x+1)^2 + (y-\frac{1}{2})^2 - R^2$ explicitely in the if statement, thus obviating the need to use the variable d at all!

(a)(ii) You need to change those eight drawPixel statements. Let us assert that we have a simple drawHorizLine function:

```
drawHorizLine ( int y; int start_x; int end_x)
{ int i;
  for i = start_x to end_x
      drawPixel (i, y);
}
```

then we simply replace the eight drawPixel statements in part (a)(i) by:

```
drawHorizLine( y, -x, x );
drawHorizLine( x, -y, y );
drawHorizLine(-y, -x, x );
drawHorizLine(-x, -y, y );
```

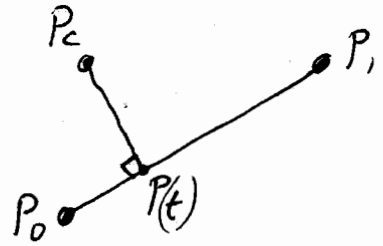
This is not the most efficient way to implement a filled circle but, for 3 marks, it is the most efficient way to get full marks for this part.

(b) The "level of accuracy" is a maximum allowed distance between the drawn curve and the true curve, often taken to be half a pixel's width. Call it ρ. We use a divide and conquer algorithm: if a straight line is within ρ of the entire curve then draw it, otherwise split the curve into two pieces and recurse. Assume the straight line drawing algorithm exists and it called by drawLine (x_1, y_1, x_2, y_2).

```
drawBezier( x0,y0,  x1,y1,  x2,y2,  x3,y3 ) {
    if ( perpDist( x1,y1,  x0,y0, x3,y3 ) < ρ
     and perpDist( x2,y2,  x0,y0, x3,y3 ) < ρ )
    then drawLine( x0,y0, x3,y3 );
    else {
        drawBezier( x0, y0,
                    ½(x0+x1),  ½(y0+y1),
                    ¼(x0+2x1+x2),  ¼(y0+2y1+y2),
                    ⅛(x0+3x1+3x2+x3),  ⅛(y0+3y1+3y2+y3) );
```

```
drawBezier( 1/8(x0 + 3x1 + 3x2 + x3), 1/8(y0 + 3y1 + 3y2 + y3),
            1/4(x1 + 2x2 + x3), 1/4(y1 + 2y2 + y3),
            1/2(x2 + x3), 1/2(y2 + y3),
            x3, y3 );
    }
}
```

The function perpDist (x_c, y_c, x_0, y_0, x_1, y_1) needs to calculate the perpendicular distance from point $P_c = (x_c, y_c)$ to the line segment between $P_0 = (x_0, y_0)$ and $P_1 = (x_1, y_1)$



$$P(t) = (1-t)P_0 + tP_1$$

$$(P_c - P(t)) \cdot (P_1 - P_0) = 0$$

solve this equation for t *

```
if ( 0 ≤ t ≤ 1)   return | Pc - P(t) |
else if ( t < 0)  return | Pc - P0 |
else              return | Pc - P1 |     /* t > 1 */
```

* The above level of detail is sufficient, but the actual equation for t is:

$$t = \frac{(P_c - P_0) \cdot (P_1 - P_0)}{(P_1 - P_0) \cdot (P_1 - P_0)}$$

$$= \frac{(x_c - x_0)(x_1 - x_0) + (y_1 - y_0)(y_c - y_0)}{(x_1 - x_0)^2 + (y_1 - y_0)^2}$$

2004 p5 q5 Marking Scheme

(a)(i)

does it draw all eight octants	2
does it start in the correct place (not one pixel early or one pixel late)	2
does it stop in the correct place (not one pixel early nor one pixel late)	2
does it update the pixel location correctly inside the loop	2
does it draw a correct, complete circle	2
	10

(ii)

for appropriate modifications	2
does it draw a correct, complete, filled circle	1
	3

(b)

check tolerances of (x_1, y_1) and (x_2, y_2)	1
if within tolerance drawLine (x_0, y_0, x_1, y_1)	1
correct recursion (two new Beziers)	1
correct coefficients in the calls to the two new Beziers	1
explanation of how to compute perpDist	1
which correctly handles the cases $t<0$ and $t>1$	1
overall clarity and correctness of algorithm	1
	7
	20