Scan Converting Circles

Given:
Center: (h,k)
Radius: r
Equation:
\[(x-h)^2 + (y-k)^2 = r^2\]

To simplify we’ll translate origin to center
Simplified Equation:
\[x^2 + y^2 = r^2\]

Circle has 8-fold symmetry
So only need to plot points in 1st octant
? x > y so step in x direction

Brute Force Circle Algorithm
Suppose we have a Set8pixel() routine
\[x_{\text{fin}} = 0.707 \times r\]
For (x=0; x<=x_{\text{fin}} ; x++)
  
  \[y = \sqrt{r^2 - x^2};\]
  Set8Pixel(round(x), round(y));
}

TOO SLOW!!

The Set8Pixel(x,y) routine

SetPixel(x,y);
SetPixel(x,-y);
SetPixel(-x,y);
SetPixel(-x,-y);
SetPixel(y,x);
SetPixel(y,-x);
SetPixel(-y,x);
SetPixel(-y,-x);
Could Use Parametric Equations

```c
for (theta=90; theta>=45; theta- -)
{
    x = r*cos(theta);
    y = r*sin(theta);
    Set8Pixel(round(x), round(y));
}
EVEN SLOWER!
```

DDA Circle Approximation

```c
x^2 + y^2 = r^2
```
Take Derivative:

2x+2y(dy/dx) = 0
dy = (-x/y)*dx
Step in x direction (dx=1)
dy = -xy
y = y + dy (approximation)

DDA Circle Algorithm

```c
x=0; y=r;
xfin=0.707*r;
while (x<=xfin)
{
    Set8Pixel(round(x), round(y));
    y = y - (x/y);
    x = x + 1;
}
Floating Pt. Divide--STILL TOO SLOW!
```

Midpoint Circle Algorithm

```
Extension of Bresenham ideas
Circle equation:  x^2 + y^2 = r^2
Define a circle function:
f = x^2 + y^2 - r^2
f=0 ==> (x,y) is on circle
f<0 ==> (x,y) is inside circle
f>0 ==> (x,y) is outside circle
```

```
We've just plotted (x_k,y_k)
(?, ?), so we're stepping in x
Next pixel is either:
(x_k + 1, y_k) -- the “top” case or
(x_k + 1, y_k - 1) -- the “bottom” case
```

Midpoint Circle Choices

```
Inside:  f<0 ==> choose upper pixel
Outside:  f>0 ==> choose lower pixel
```
Evaluate \( f \) at midpoint
\[(x=x_k+1, \ y=y_k-1/2)\]

Predictor:
\[P_k = f(x_k+1, y_k-1/2)\]
\[P_k < 0 \Rightarrow \text{inside (choose top pixel)}\]
\[P_k > 0 \Rightarrow \text{outside (choose bottom pixel)}\]
\[P_k = (x_k+1)^2 + (y_k-1/2)^2 - r^2\]
As for Bresenham, try to get a recurrence relation for \( P \)

Top Case \((x_{k+1} = x_k + 1, \ y_{k+1} = y_k)\):
\[P_{k+1} = f(x_k+1, y_k-1/2)\]
\[P_{k+1} = ((x_k+1) + 1)^2 + ((y_k - 1/2) - 1/2)^2 - r^2\]
\[P_{k+1} = x_k^2 + 2x_k + 5/4 + y_k^2 - y_k - r^2\]
But, \[P_k = x_k^2 + 2x_k + 5/4 + y_k^2 - y_k - r^2\]
\[P_k = P_{k+1} - P_k\]
So \[P_k = 2x_k + 3\]

Bottom Case \((x_{k+1} = x_k + 1, \ y_{k+1} = y_k - 1)\):
\[P_{k+1} = f(x_k+1, y_k-1)\]
\[P_{k+1} = ((x_k+1) + 1)^2 + ((y_k - 1) - 1)^2 - r^2\]
\[P_{k+1} = x_k^2 + 4x_k + 4 + y_k^2 - 3y_k + 9/4 - r^2\]
But, \[P_k = x_k^2 + 2x_k + 5/4 + y_k^2 - y_k - r^2\]
\[P_k = P_{k+1} - P_k\]
So \[P_k = 2x_k - 2y_k + 5\]

Initial \( P \):
\[P_0 = (x_0 + 1)^2 + (y_0 - 1/2)^2 - r^2\]
\[P_0 = 5/4 - r \rightarrow \ 1-r \ (\text{rounding to integer})\]

Midpoint Circle Algorithm
\[x=0; \ y=r; \ P=1-r;\]
\[\text{Set8Pixel}(x, y)\];
while \((x<y)\)
\[
\{ \ x = x + 1; \ \text{Set8Pixel}(x, y); \ 
\text{if} \ (P < 0) \ 
\ P = P + x<<1 + 1; \ 
\text{else} \ 
\ { \ y = y - 1; \ P = P + (x-y)<<1 + 1; } \}
\]

Circles not centered on origin
Need to redo the Set8Pixel() function
New Set8Pixel() Function

Set8Pixel(x, y, h, k)
{
    SetPixel(x+h, y+k);
    SetPixel(x+h, -y+k);
    SetPixel(-x+h, y+k);
    SetPixel(-x+h, -y+k);
    SetPixel(y+h, x+k);
    SetPixel(y+h, -x+k);
    SetPixel(-y+h, x+k);
    SetPixel(-y+h, -x+k);
}

Adjusting for Aspect Ratio

- One way—adjust at pixel level
- If pixel width = w, height = h
- A.R. = h/w
- So either:
  - Multiply each x by A.R.
  - or Divide each y by A.R.

Scan Converting an Ellipse

Midpoint Ellipse Algorithm: (x, y), just plotted...

Evaluate Ellipse function at midpoint (x, y): $E(x, y) = (x^2 - a^2) + (y^2 - b^2)$

The ellipse function at (x, y) equals 0 when:

$x = \frac{aj}{b}$

where $a = \frac{2a}{2}$ and $b = \frac{2b}{2}$

Bottom cases:

$E(x, y) = E(x, y) + E(x+b, y)$

$x = \frac{aj}{b}$

$E(x, y) = E(x, y) + E(x+b, y)$

Scan Converting an Ellipse
Midpoint Ellipse Alg. (Region I)
DPx = 2*ry*ry; Dpy = 2*rx*rx;
Initial values of Px, Py, Pq:

\[
\begin{align*}
P_x &= r_x^2 (x_1 - x_0) - r_y^2 (y_2 - y_1) \\
P_y &= r_x^2 (y_2 - y_1) + r_y^2 (x_1 - x_0)
\end{align*}
\]

Also need initial values of Px, Py:
P_x = 2*r_x^2 x_0; P_y = 2*r_y^2 y_0;

Also need recurrence relations for Pq:
P_q = 2*r_x r_y (x_1 - x_0)

So \( \Delta P_x = 2r_x^2 \) (constant)
So \( \Delta P_y = 2r_y^2 \) (constant)

Set4Pixel(x, y);
while (px < py)  // Region I
{  
x = x + 1; Px = Px + DPx;
if (f > 0)      // Bottom case
{  
y = y - 1; Py = Py - Dpy; f = f + ry^2 + Px - Py;
}
else          // Top case
{  
f = f + ry^2 + Px;
}
Set4Pixel(x, y);
}

Scan Converting other 2D Curves

DDA:
y = f(x); If we can differentiate it:
dy/dx = f'(x)
Step in x for parts of curve where dy/dx < 1
x = x + 1
y = y + f'(x)
Step in y for parts of curve where dy/dx > 1
y = y + 1
x = x + 1/f'(x)

Scan Converting other 2D Curves

Midpoint Algorithm
– Use techniques like those used for the ellipse
– Often we can devise a midpoint algorithm
  - Most efficient result could have several levels of recurrence

Text and Characters
– Very important output primitive
– Many pictures require text
– Two general techniques used
  - Bitmapped (raster)
  - Stroked (outline)
Bitmapped Characters
- Each character represented (stored) as a 2-D array
- Each element corresponds to a pixel in a rectangular “character cell”
- Simplest: each element is a bit (1=pixel on, 0=pixel off)

```
00111000
01101100
11000110
11000110
11111110
11000110
11000110
00000000
```

Stroked Characters
- Each character represented (stored) as a series of line segments
- Sometimes as more complex primitives
- Parameters needed to draw each stroke
  - Endpoint coordinates for line segments

```
  \( (0,0), (0,10) \)
  \( (0,0), (10,0) \)
  \( (0,5), (6,5) \)
```

Characteristics of Bitmapped Characters
- Each character in set requires same amount of memory to store
- Characters can only be scaled by integer scaling factors
- "Blocky" appearance
- Difficult to rotate characters by arbitrary angles
- Fast (BitBLT)

Characteristics of Stroked Characters
- Number of strokes (storage space) depends on complexity of character
- Each stroke must be scan converted ==> more time to display
- Easily scaled and rotated arbitrarily
  - Just transform each stroke

Example Character-Display Algorithms
- See CS-460/560 Notes Web Pages:
  - An illustration of how to display bitmapped characters
  - An illustration of how to display stroked characters

Algorithm for Bitmapped Characters--an Example
- Define bitmap for the letter--e.g. ‘T’
- 
  ```
  int T[7][7] = {
    0,0,0,0,0,0,0,
    0,1,1,1,1,1,0,
    0,0,1,0,0,0,0,
    0,0,1,0,0,0,0,
    0,0,1,0,0,0,0,
    0,0,0,0,0,0,0,
    0,0,0,0,0,0,0
  } // bitmap for ‘T’
  ```
- [Could have a file with the bitmap descriptions of each character in the character set to be displayed]
- Not the most efficient way of doing it
  - Could have used individual bits
  - Algorithm would be more complex
2. Define function to display bitmap letter at pixel coordinates (x,y)
   disp_letter (int x, int y, int letter[7][7])
   { int i, j;
     for (i=0; i<7; i++)
       for (j=0; j<7; j++)
         if (letter[i][j] == 1)
           Setpixel(x+j,y+i);  // plot from bitmap
   }

3. Call function, passing it desired bitmap
   disp_letter (50,100,t);  // draw a 'T' at (50,100)

1. Define a character (CH) type:
   typedef struct tagCH
   {
     int n;
     POINT * pts;
   } CH;

   pts is an array of stroke endpoint vertices
   n is the number of vertices

2. Define generic display-character function
   – Strokes are specified in variable c (type CH)
   – Display at pixel coordinates (xx,yy):
     disp_char (int xx , int yy, CH c)
     { int i, n_strokes;
       n_strokes=c.n/2;     // n points ==> n/2 strokes
       for (i=0; i<n_strokes; i++)
         line(xx+c.pts[2*i].x, yy +c.pts[2*i].y,
             xx+c.pts[2*i+1].x, yy+c.pts[2*i+1].y);
     }

   Strokes are specified in variable c (type CH)
   Display at pixel coordinates (xx,yy):
   n_strokes = c.n/2; // n points -> n/2 strokes

   for (i = 0; i < n_strokes; i++)
   {
     line(xx + c.pts[2*i].x, yy + c.pts[2*i].y,
          xx + c.pts[2*i+1].x, yy + c.pts[2*i+1].y);
   }

3. Define the character's CH structure
   The following could be for an 'F':
   POINT p[6];   CH f;
   p[0].x=0;   p[0].y=0;   p[1].x=0;   p[1].y=10;
   p[2].x=0;   p[2].y=0;   p[3].x=10;   p[3].y=0;
   p[4].x=0;   p[4].y=5;   p[5].x=6;   p[5].y=5;
   f.n = 6;   f.pts = p;

   Descriptions of each character in the character set could be stored in a file

4. Call the character-display function, passing it the desired character (CH)
   disp_char (50,100,f); // draw 'F' at (50,100)

OpenGL Character Functions
   Only low-level support in basic OpenGL library
   – Explicitly define characters as bitmaps
   – Display by mapping selected sequence of
     bitmaps to adjacent positions in frame buffer (BitBLTing)
OpenGL GLUT Text Support

Some predefined character sets in GLUT

1. GLUT Bitmapped:
   - Display with `glutBitmapCharacter(font, ch);`
     - `font`: constant type face to be used
       - GLUT_BITMAP_8_BY_13 (fixed-width)
       - GLUT_BITMAP_TIMES_ROMAN_10 (variable width)
       - Others are available
     - `ch`: ASCII code of character
   - Position with `glRasterPosition2i(x, y);`
   - Example:
     - `glRasterPosition2i(20, 10);`
     - `glutBitmapCharacter(GLUT_BITMAP_8_13, 'A');`
     - x coordinate is incremented by width of character after display

2. GLUT Stroked Characters:
   - `glutStrokeCharacter(font, ch);`
   - Font:
     - GLUT_STROKE_ROMAN (proportional spacing)
     - GLUT_STROKE_MONO_ROMAN (constant spacing)
   - Ch: ASCII code of character
   - Size & position determined by specifying transformation operations
   - We'll see these later