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

Special Case - Ellipse aligned with x-y axes

More origin to center:

\[
\frac{x^2}{r_x^2} + \frac{y^2}{r_y^2} = 1
\]
Ellipses have 4-fold symmetry. So use a Set4 Pixel function. Only traverse 1st quadrant.

Step in X until \( \frac{dy}{dx} < -1 \). Then Step in Y.

**DDA Algorithm (Region I)** -- Step in X:
\[ \Delta x = 1, \quad \Delta y = -\frac{xr_2}{yr_2} \]
Each iteration:
\[ x = x+1 \]
\[ y = y - \frac{xr_2}{yr_2} \]

**DDA Algorithm (Region II)** -- Step in Y:
\[ \Delta y = 1, \quad \Delta x = -\frac{yr_2}{xr_2} \]

**Midpoint Ellipse Algorithm:** \( (x_k, y_k) \) just plotted.

**Region I:** \( \frac{dy}{dx} > -1 \) \( \Rightarrow 2r_2 \times < 2r_2 \)

Next point:
\[ \begin{cases} (x_{k+1}, y_{k}) \text{ Top} \\ (x_{k+1}, y_{k-1}) \text{ Bottom} \end{cases} \]

**Region II:** \( \frac{dy}{dx} < -1 \)

Next point:
\[ \begin{cases} (x_{k+1}, y_{k-1}) \text{ Left} \\ (x_{k+1}, y_{k+1}) \text{ Right} \end{cases} \]

Define: \( P_x = 2r_2 x, \quad P_y = 2r_2 y \)

**Ellipse functions**:
\[ f = x^2/r_1^2 + y^2/r_2^2 - 1 \]
Evaluate at \( (x_k, y_k) \):
\[ \begin{align*} f &< 0 \Rightarrow \text{inside, choose top} \\ f &> 0 \Rightarrow \text{outside, choose bot} \end{align*} \]
Evaluate Ellipse function at midpoint $(x_{k+1}, y_{k+1})$:

$$f_{k+1} = f_k + \Delta f$$

$$x_{k+1} = x_k + 1$$
$$y_{k+1} = \frac{x_k}{y_k} \text{ (top)}$$
$$y_{k+1} = -\frac{x_k}{y_k} \text{ (bottom)}$$

To simplify, try to get recurrence relation:

$$f_{k+1} = f_k + \Delta f$$

Top case:

$$f_{k+1} = f_k + \alpha x_k + \beta$$

Result: $\Delta f = f_k - f_{k-1}$

Bottom case:

$$f_{k+1} = f_k + \alpha x_k + \beta$$

Result: $\Delta f = f_k - f_{k-1}$

Initial Values of $f_0, P_x, P_y$ when $x=0, y=\frac{y_0}{y_0}$

$$f_0 = \frac{y_0}{y_0} (0) - 1 - \frac{y_0}{y_0}$$

$$P_x = 2y_0 x_0 = 0$$
$$P_y = 2y_0 y_0 = 2y_0^2$$

Also need initial values of $P_x, P_y$

$$P_{x0} = 2y_0 x_0 = 0$$
$$P_{y0} = 2y_0 y_0 = 2y_0^2$$

Also need recurrence relations for $P_x, P_y$

$$P_{xk+1} = 2y_k^2 x_k + \left( \frac{2y_k^2}{y_k} \right)$$
$$P_{yk+1} = \left( \frac{2y_k^2}{y_k} \right)$$

So $\Delta P_x = 2y_k^2 \text{ (constant)}$

$\Delta P_y = 0 \text{ (top)}$
$$x_{k+1} = -\frac{x_k}{y_k} \text{ (bottom)}$$

$$\Delta P_y = -2y_k^2 \text{ (bottom)}$$
Midpoint Ellipse Alg. (Region I)

\[ \text{DPx} = 2 \cdot ry \cdot ry; \quad \text{DPy} = 2 \cdot rx \cdot rx; \quad x = 0; \quad y = ry; \quad Px = 0; \]
\[ \text{Py} = 2 \cdot rx \cdot rx; \quad f = ry^2 + rx^2 \cdot (0.25 - ry); \quad ry^2 = ry \cdot ry; \]

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 - Py + Px; \}
    else // Top case
        f = f + ry2 + Px;
    Set4Pixel(x, y);
}
Scan Converting other 2D Curves

DDA:

\[ y = f(x); \text{ If we can differentiate it:} \]
\[ \frac{dy}{dx} = f'(x) \]
Step in x for parts of curve where \( \frac{dy}{dx} < 1 \)
\[ x = x + 1 \]
\[ y = y + f'(x) \]
Step in y for parts of curve where \( \frac{dy}{dx} > 1 \)
\[ y = y + 1 \]
\[ x = x + \frac{1}{f'(x)} \]

Plotting Implicit Functions

- Explicit function: \( y = f(x) \)
  - Can always plot using DDA or Midpoint Algorithms
- Implicit function: \( g(x,y) = 0, \text{ e.g.:} \)
  - Ovals of Casini
  - \( g(x,y) = (x^2+y^2+a^2)^2 - 4a^2x^2 - b^4 \)
  - Often can’t be converted to explicit form
  - No solution \( y = f(x) \)
  - How do we plot such functions?
3D Surfaces

- A related more general implicit function
- \( z = f(x, y) \)
  - \( z \) could represent the height of point \((x, y)\)
- Contour curves
  - Want to plot points that have the same height
  - \( f(x, y) = h \), a constant
  - Gives curves like on a topographic map
  - Need to compute points \((x, y)\) that satisfy \( f(x, y) = h \)

Marching Squares

- Approximation technique for solving contour curve problem
- Suppose we sample \( f(x, y) \) at evenly-spaced points on a rectangular array
  \[
  f_{ij} = f(x_i, y_j), \quad x_i = x_0 + i \cdot dx, \quad i = 0,1,\ldots,N-1 \\
  y_j = y_0 + j \cdot dy, \quad j = 0,1,\ldots,M-1
  \]
- Want to find an approximation to curve \( z = f(x, y) \) for a particular value of \( z = h \)
  - For a given \( c \) there may be 0, 1, or many contour curves
Constructing Piecewise Linear Curve

- Start with rectangular cell
- Algorithm will find line segments for each cell using corner z values to determine if contour passes through cell

\[
\begin{align*}
(x_i, y_j) &\quad (x_{i+1}, y_{j+1}) \\
(x_i, y_{j+1}) &\quad (x_{i+1}, y_j)
\end{align*}
\]

- In general, sampled values are not equal to contour values
- But curve could still go through the cell
- One possible case:
  - \( f(i,j) > h \)
  - \( f(i+1,j) < h \)
  - \( f(i+1,j+1) < h \)
  - \( f(i,j+1) < h \)

- If \( f(x,y) - h > 0 \) at one vertex
- And \( f(x,y) - h < 0 \) at adjacent vertex,
  - It must be 0 somewhere in between

\[ f(x,y) - h = 0 \] between segments
Line Segments between intersection pts

- Estimate where contour intersects two edges and join points with line segment
  - Simplest approximation to curve

- Use interpolation to get intersection pts.
  \[ f(x_i, y_j) = a, \quad a < h; \quad f(x_{i+1}, y_j) = b, \quad b > h \]
  \[ \frac{x-x_i}{dx} = \frac{a-h}{a-b} \quad x = x_i + dx \cdot \frac{a-h}{a-b} \]

Other Types of Cells

- There are 16 possible combinations of cell vertex labelings
Only 4 Unique Vertex Labelings

- Rotational symmetry (e.g. 1 & 2)
- Exchange (black & white) symmetry (e.g. 0 & 15)
- So there are only 4 unique cases:

![Diagram of vertex labelings]

Four unique cases of vertex labelings.

How to draw Line Segments for each Case

- 1\textsuperscript{st} case: trivial (contour doesn’t intersect cell) \(
\rightarrow\) no line segments drawn
- 2\textsuperscript{nd} case: adjacent edges, as above, generates one line segment between adjacent edges
- 3\textsuperscript{rd} case: also draw one line segment that goes between opposite edges
- 4\textsuperscript{th} case: has an ambiguity
4th Case Ambiguity

Which one to use? Break or join contour?
- Pick one at random
- Subdivide into smaller cells & repeat
- Or ignore since no solution w/o more data

Subdivision

But we can ignore them if we want to keep the edges closed
Marching Squares Algorithm

- Form cell array data[i][j] from implicit function
  - For each cell i,j
    - Compute data[i][j] from f(x,y)

- Process cells to generate line segments
  - “March” through the cells
    - For each cell
      - Call code for single-cell processing: cell(…)
      - Compute & draw appropriate lines for that cell
      - Call helper functions for each of 4 cases

Code for Single Cell (i, j)

int cell(double a, double b, double c, double d)
{
    int n=0;
    if(a>h) n+=1; if (b>h)n+=8; if(c>h)n+=4; if(d>h)n+=2;
    switch(n) {
        // cases 1, 2, 4, 7, 8, 11, 13, 14: // contour cuts 1 corner
        draw_one(n, i, j, a, b, c, d); break
        // cases 3, 6, 9, 12: // contour crosses cell
        draw_opposite(n, i, j, a, b, c, d); break;
        // cases 0, 15: break; // nothing to draw
    }
}
draw_one ftn: adjacent edges

```c
void draw_one(n, i, j, a, b, c, d) {
    Switch(n)
    {
        case 1: case 14:
            x1=ox; y1=oy+dy*(h-a)/(d-a);
            x2=ox+dx*(h-a)/(b-a); y2=oy;
            break;
        // other cases here
    }
    glBegin(GL_LINES);
    glVertex2d(x1,y1); glVertex(x2,y2);
    glEnd();
}
```

Other “draw” function

☞ Draw_opposite(n,i,j,a,b,c,d)
    – For opposite-edge case
Extension to 3D

Marching Squares is easily extended to handle 3D volumetric data

- Represent “iso-surfaces” instead of contours
  - \( f(x,y,z) = \text{constant} \)
  - Display as 3D contour plots
- Use 3D grid cells instead of 2D cells
- “Marching Cubes” algorithm
  - Check data values at 8 corners of a cell
  - Interpolate to find best polygon surface element passing through a cell
  - Result: polygon mesh approximation to the surface
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

```
  x
 /|
/ | 
/  |
  y
```

**Strokes:**
- \((0,0), (0,10)\)
- \((0,0), (10,0)\)
- \((0,5), (6,5)\)

**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 stokes (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:
- Links to:
  - An illustration of how to display bitmapped characters
  - An illustration of how to display stroked characters
Algorithm for Bitmapped Characters--an Example

1. 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},
                  {0,0,0,1,0,0,0}, {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} };  // 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

Bitmapped Character Algorithm, Continued

2. Define a 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 the function, passing desired bitmap
   disp_letter (50,100,t);  // draw a ‘T’ at (50,100)
Algorithm for Stroked Characters

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

Stroked Character Algorithm, Continued

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);
   }
3. Define the character's CH structure

The following could be for an 'F':

```c
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)

```c
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_15, '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

Character Fonts in Windows

- FONT--Typeface, style, size of characters in a character set
- Three kinds of Windows Fonts
  - Stock Fonts
  - Logical or GDI Fonts
  - Device Fonts
Windows Stock Fonts

- Built into Windows
- Always available

```
Font = ANSI_FIXED_FONT
Font = ANSI_VAR_FONT
Font = DEVICE_DEFAULT_FONT
Font = OEM_FIXED_FONT
Font = SYSTEM_FONT
Font = SYSTEM_FIXED_FONT
```

Windows Logical or GDI Fonts

- Defined in separate font resource files on disk
  - .fon file
    - (Stroke or Raster)
  - .fot/.ttf file
    - (TrueType)
- Specific instance must be “created”
Windows Stroke Fonts

- Consist of line/curve segments
- Continuously scalable
- Slow to draw
- Legibility not too good

Windows Raster Fonts

- Bitmaps so:
  - Scaling by non-integer factors difficult
  - Fast to display
  - Legibility very good
Windows TrueType Fonts

Rasterized stroke fonts so:
– Stored as strokes with hints to convert to bitmap
– Conversion called rasterization
– Continuously scalable
– Fast to display
– Legibility very good
– Combine best of both stroke and raster fonts

Windows TrueType Fonts

Courier New AaBbCcDdEe
Courier New Bold AaBbCcDdEe
Courier New Italic AaBbCcDdEe
Courier New Bold Italic AaBbCcDdEe
Times New Roman AaBbCcDdEe
Times New Roman Bold AaBbCcDdEe
Times New Roman Italic AaBbCcDdEe
Times New Roman Bold Italic AaBbCcDdEe
Arial AaBbCcDdEe
Arial Bold AaBbCcDdEe
Arial Italic AaBbCcDdEe
Arial Bold Italic AaBbCcDdEe
∑ψμβλ ΑνβδΧχΔδΕε
 tutte
Device Fonts

- Native to output device
- E.g., built-in printer fonts
  - Postscript

Using Windows Stock Fonts

- Like stock pens, brushes
- Accessed with:
  
  GetStockObject(font_name);
  
  • Returns a handle to a font
  • Use by selecting into DC with SelectObject():

  Or --
  
  CDC::SelectStockObject(font_name);
Using Windows Logical Fonts

- Instantiate a CFont object
- Use CFont::CreateFont(14 params!!)
  - Specify characteristics
  - Interpolates data from font file
  - \(\rightarrow\) new sizes, bold, rotated, etc.
- Select CFont object into the DC
- Called logical since determined by program logic not just file contents
- See online help

Windows Text Metrics

- CreateFont() may not give you exactly what you ask for
- Can use CDC::GetTextMetrics() to find out font details
  - Gives lots of information in a TEXTMETRIC structure
  - Commonly used to determine font size
    - can be used to set line spacing, caret size, sizes of buttons, etc.
Windows Text Metrics

- External leading
- Internal leading
- Ascent
- Height
- Descent