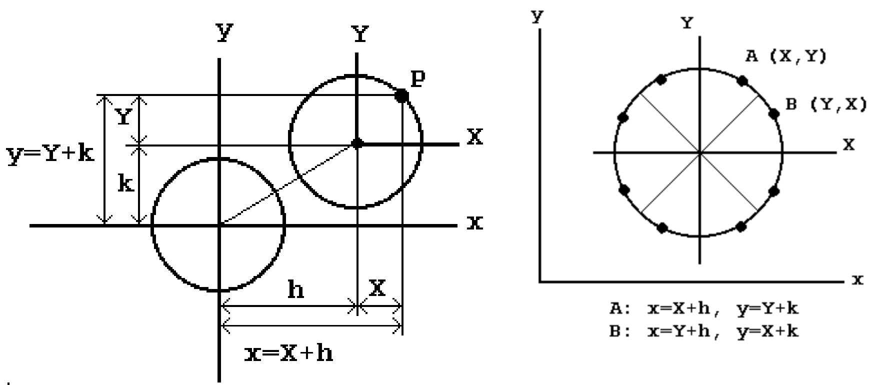
Circles not centered on origin



Need to redo the Set8Pixel() function

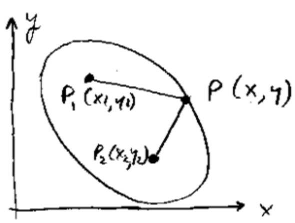
New Set8Pixel() Function

```
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

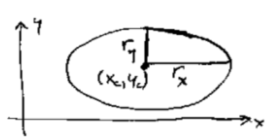


$$\sqrt{(x-x_1)^2+(y-y_1)^2} + \sqrt{(x-x_2)^2+(y-y_2)^2} = K$$

Square twice & rearrange

$$\rightarrow Ax^2+By^2+Cxy+Dx+Ey+F=0$$

Special Case – Ellipse aligned with x-y axes



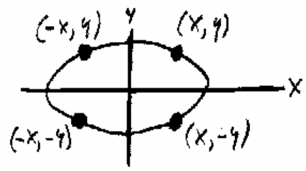
r_x, r_y : semi-major, semi-minor axes

(x_c, y_c): coordinates of center

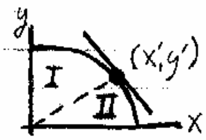
$$\left(\frac{x-x_c}{r_x}\right)^2 + \left(\frac{y-y_c}{r_y}\right)^2 = 1$$

Move origin to center:

$$\frac{x^2}{r_x^2} + \frac{y^2}{r_y^2} = 1$$



Ellipse has 4-fold Symmetry
So use a Set4Pixel function
Only traverse 1st quadrant



Step in x until $\frac{dy}{dx} < -1$
Then Step in y

$\frac{dy}{dx}(x',y') = -1$

$\frac{x^2}{r_x^2} + \frac{y^2}{r_y^2} = 1$

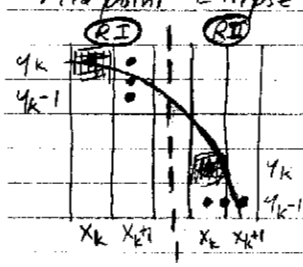
DDA Algorithm (Region I) -- Step in x:
$\Delta x = 1$, $\Delta y = -x r_y^2 / y r_x^2$
Each iteration: $x = x+1$
$y = y - x r_x^2 / y r_x^2$

$\frac{dy}{dx} = -\frac{x r_y^2}{y r_x^2}$

$\frac{dy}{dx} = -1$ at (x',y')
So Solve for (x',y')

DDA Algorithm (Region II) -- Step in y:
$\Delta y = 1$, $\Delta x = -y r_x^2 / x r_y^2$

Midpoint Ellipse Algorithm - (x_k, y_k) just plotted



Region I: $\frac{dy}{dx} > -1 \Rightarrow 2r_y^2 x < 2r_x^2 y$

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: $\begin{cases} (x_k, y_k-1), & \text{Left} \\ (x_k+1, y_k-1), & \text{Right} \end{cases}$

Define: $P_x \equiv 2r_y^2 x$, $P_y \equiv 2r_x^2 y$
$\Rightarrow$ Region I when $P_x < P_y$

Ellipse function:

$f \equiv x^2 r_y^2 + y^2 r_x^2 - r_x^2 r_y^2$ $\begin{cases} = 0 \Rightarrow (x,y) \text{ on curve} \\ < 0 \Rightarrow \text{inside, choose TOP} \\ > 0 \Rightarrow \text{outside, choose BOT} \end{cases}$

Evaluate at $(x_k+1, y_k - \frac{1}{2})$
(midpoint)

Evaluate Ellipse function at midpoint $(x_k+1, y_k-\frac{1}{2})$:

$$f_k = r_y^2(x_k+1)^2 + r_x^2(y_k-\tfrac{1}{2})^2 - r_x^2 r_y^2$$

Too complex -- Try to get recurrence relation:

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

$$x_{k+1} = x_k + 1$$

$$y_{k+1} = \begin{cases} y_k \text{ (top)} \\ y_k - 1 \text{ (Bottom)} \end{cases}$$

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

Top case: $f_{k+1} = r_y^2((x_k+1)+1)^2 + r_x^2(y_k-\frac{1}{2})^2 - r_x^2 r_y^2$

Result: $\Delta f = r_y^2(2x_k+3)$

But $x_{k+1} = (x_k+1)$, so $\Delta f = r_y^2(2x_{k+1}+1)$

$\Delta f = P_x + r_y^2$

Bottom case: $f_{k+1} = r_y^2((x_k+1)+1)^2 + r_x^2((y_k-1)-\frac{1}{2})^2 - r_x^2 r_y^2$

Result: $\Delta f = r_y^2(2x_k+3) + r_x^2(-2y_k+2)$

But $x_{k+1} = x_k+1$ & $y_{k+1} = y_k-1$, So $\Delta f = r_y^2 + P_x - P_y$

Initial Values of f_k, P_x, P_y, when $x=0$, $y=r_y$

$$f_0 = r_y^2(0+1)^2 + r_x^2(r_y-\tfrac{1}{2})^2 - r_x^2 r_y^2$$

$$f_0 = r_y^2 + r_x^2(\tfrac{1}{4} - r_y)$$

Also need initial values of P_x & P_y

$P_{x_0} = 2r_y^2 x_0 = 0 \qquad P_{y_0} = 2r_x^2 y_0 = 2r_x^2 r_y$

Also need recurrence relations for P_x & P_y

$P_{x_k} = 2r_y^2 x_k \qquad P_{y_k} = 2r_x^2 y_k$

$P_{x_{k+1}} = 2r_y^2(x_k+1)$

$$P_{y_{k+1}} = \begin{cases} 2r_x^2 y_k \text{ (top)} \\ 2r_x^2(y_k-1) \text{ (Bottom)} \end{cases}$$

So $\Delta P_x = 2r_y^2$ (constant)

$$\text{So } \Delta P_y = \begin{cases} 0 \text{ (Top)} \\ -2r_x^2 \text{ (Bottom)} \end{cases}$$

Region II - Just plotted (x_k, y_k)

Next point $\begin{cases} (x_k, y_k-1), & \text{Left case} \\ (x_k+1, y_k-1), & \text{Right case} \end{cases}$

Midpoint: $(x_k+\frac{1}{2}, y_k-1) \Rightarrow f = r_y^2(x+\frac{1}{2})^2 + r_x^2(y-1)^2 + r_x^2 r_y^2$

(predictor fn)

Assume last point in Region I was (x', y') -

Results: $f_{init} = r_y^2(x'+\frac{1}{2})^2 + r_x^2(y'-1)^2 - r_x^2 r_y^2$

Next point: $y = y-1$

$\Delta P_y = 2r_x^2$

$f > 0 \Rightarrow \Delta f = r_x^2 - P_y$

$f < 0 \Rightarrow x = x+1,\ \Delta f = r_x^2 - P_y + P_x$

Midpoint Ellipse Alg. (Region I)

```
DPx=2*ry*ry; DPy=2*rx*rx; x=0; y=ry; Px=0;
Py=2*rx*rx*ry; f=ry*ry+rx*rx(0.25-ry); ry2=ry*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+ry2+Px-Py;}
     else         // Top case
         f=f+ry2+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)

Plotting Implicit Functions

- Explicit function: y = f(x)
  - Can always plot using DDA or Midpoint Algorithms
- Implicit function: g(x,y) = 0, 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

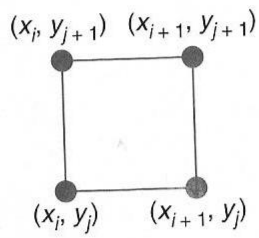
  fij = f(xi,yj), xi = x0+i*dx, i = 0,1,…,N-1

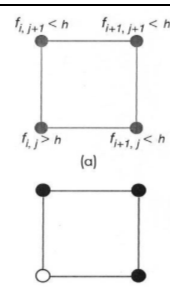
  yj = y0+j*dy, j = 0,1,…,M-1

  - Want to find an approximation to curve z=f(x,y) for a particular value of z = h
    - For a given c there may 0, 1, or many contour curves

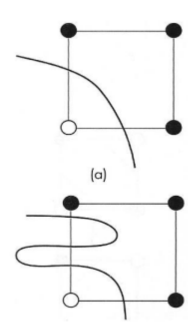
Constructing Piecewise Linear Curve

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





- 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



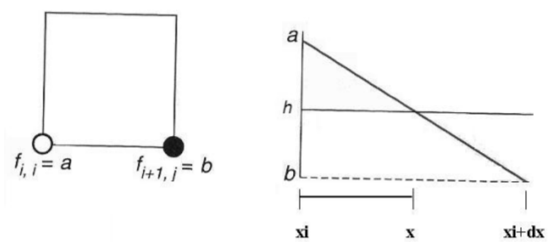
- If f(x,y)-h > 0 at one vertex
- And f(x,y)-h < 0 at adjacent vertex,
  - It must be 0 somewhere in between ✍ contour passes through that segment

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.

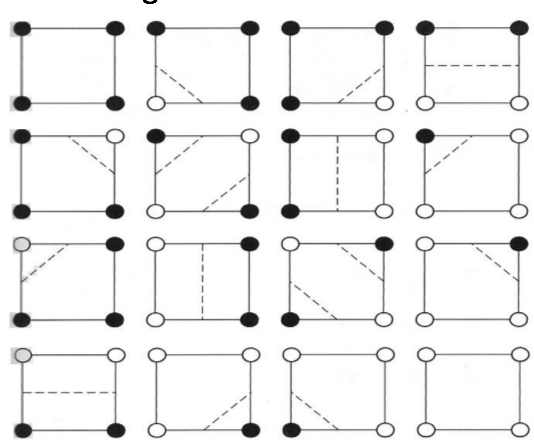
f(xi,yj) = a, a < h;   f(xi+1,yj) = b, b > h

(x-xi)/dx = (a-h)/(a-b) ⇒ x = xi +dx*(a-h)/(a-b)



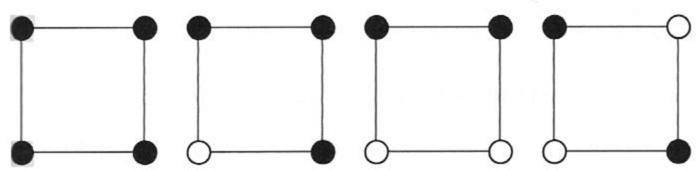
Other Types of Cells

- 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:

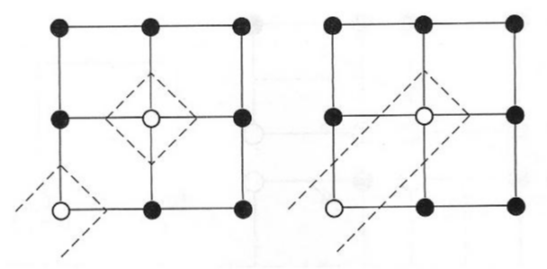


Four unique cases of vertex labelings.

How to draw Line Segments for each Case

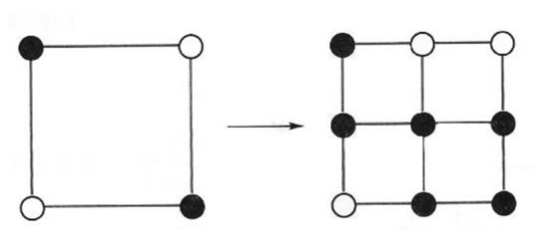
- 1st case: trivial (contour doesn't intersect cell) ✍ no line segments drawn
- 2nd case: adjacent edges, as above, generates one line segment between adjacent edges
- 3rd case: also draw one line segment that goes between opposite edges
- 4th case: has an ambiguity

4^{th} 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

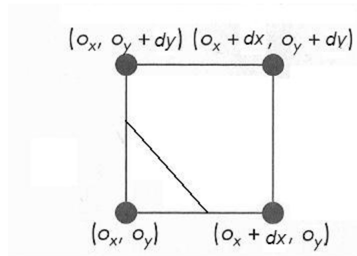
- Form cell array data[ ][ ] from implicit function
  - For each cell i,j
    - Compute data[i][j] from f(x,y)
- Process cells to generate line segments
  - "March" through the cells
    - For each cell
      - Call code for single-cell processing: cell(…)
      - Compute & draw appropriate lines for that cell
        - Call helper functions for each of 4 cases

Code for Single Cell (i, j) vertices a, b, c, d

```
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

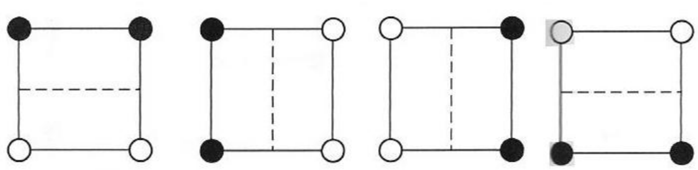
```
void draw_one(n, i, j, a, b, c, d)  {
Switch(n)
{
  case 1: case 14:
     x1=ox; y1=oy+dy*(h-a)/(d-a);
     x2=ox+dx*(h-a)/(b-a); y2=oy;
     break;
// other cases here
}
glBegin(GL_LINES);
     glVertex2d(x1,y1); glVertex(x2,y2);
glEnd();   }
```



Drawing the line segment for adjacent case 1

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) = 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

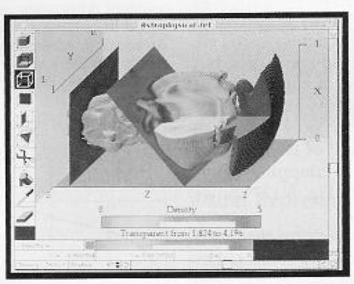


FIGURE 8–128 Cross-sectional slices of a three-dimensional data set. (*Courtesy of Spyglass, Inc.*)

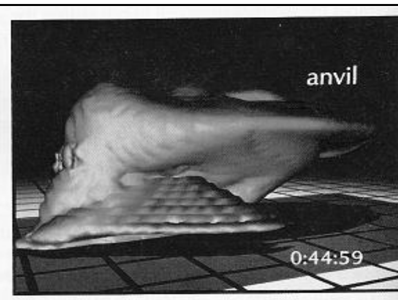


FIGURE 8–129 An isosurface generated from a set of water-content values obtained from a numerical model of a thunderstorm. (*Courtesy of Bob Wilhelmson, Department of Atmospheric Sciences and the National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign.*)

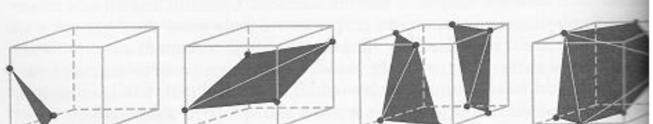


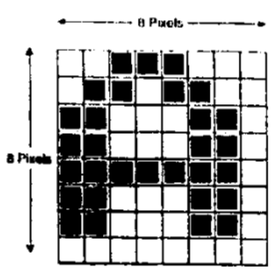
FIGURE 8–130 Isosurface intersections with grid cells, modeled with triangle patches.

Text and Characters

- Very important output primitive
- Many pictures require text
- Two general techniques used
  - Bitmapped (raster)
  - Stroked (outline)

Bitmapped Characters

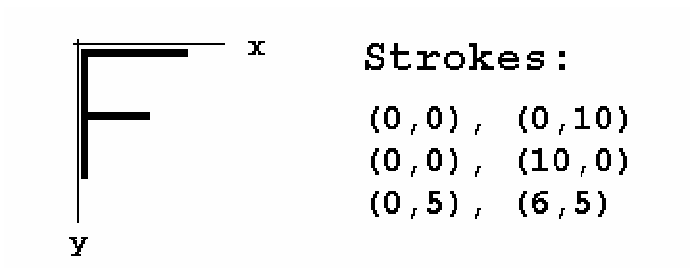
- Each character represented (stored) as a 2-D array
  - Each element corresponds to a pixel in a rectangular “character cell”
  - Simplest: each element is a bit (1=pixel on, 0=pixel off)



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



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);
}
```

Stroked Character Algorithm, Continued

- 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]

Stroked Character Algorithm, Continued

- 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_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

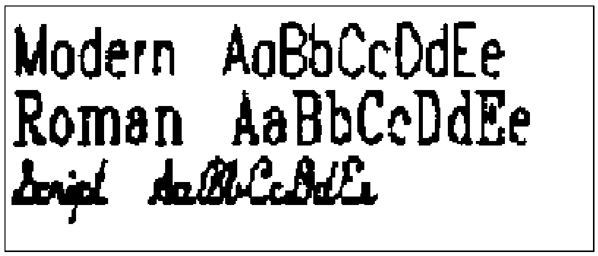
Windows Stock Fonts

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

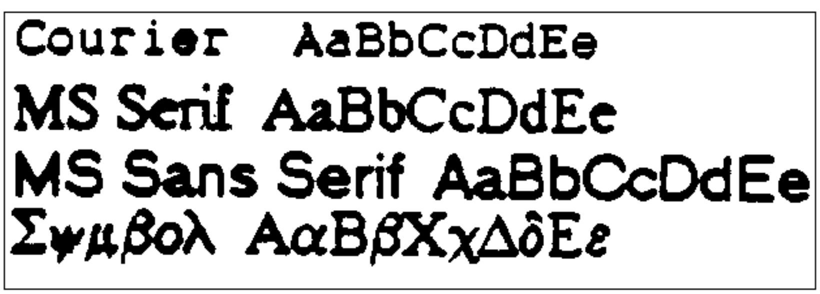
- Consist of line/curve segments
- Continuously scalable
- Slow to draw
- Legibility not too good



Windows Stroke Fonts

Windows Raster Fonts

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



Windows Raster Fonts

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

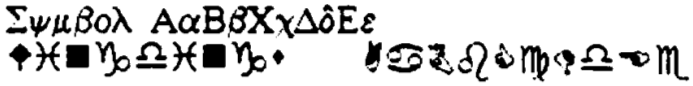
Arial AaBbCcDdEe

Arial Bold AaBbCcDdEe

Arial Italic AaBbCcDdEe

Arial Bold Italic AaBbCcDdEe

Σψµβολ ΑαΒβΧχΔδΕε



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
  - --> 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