Fonts

Steven R. Bagley

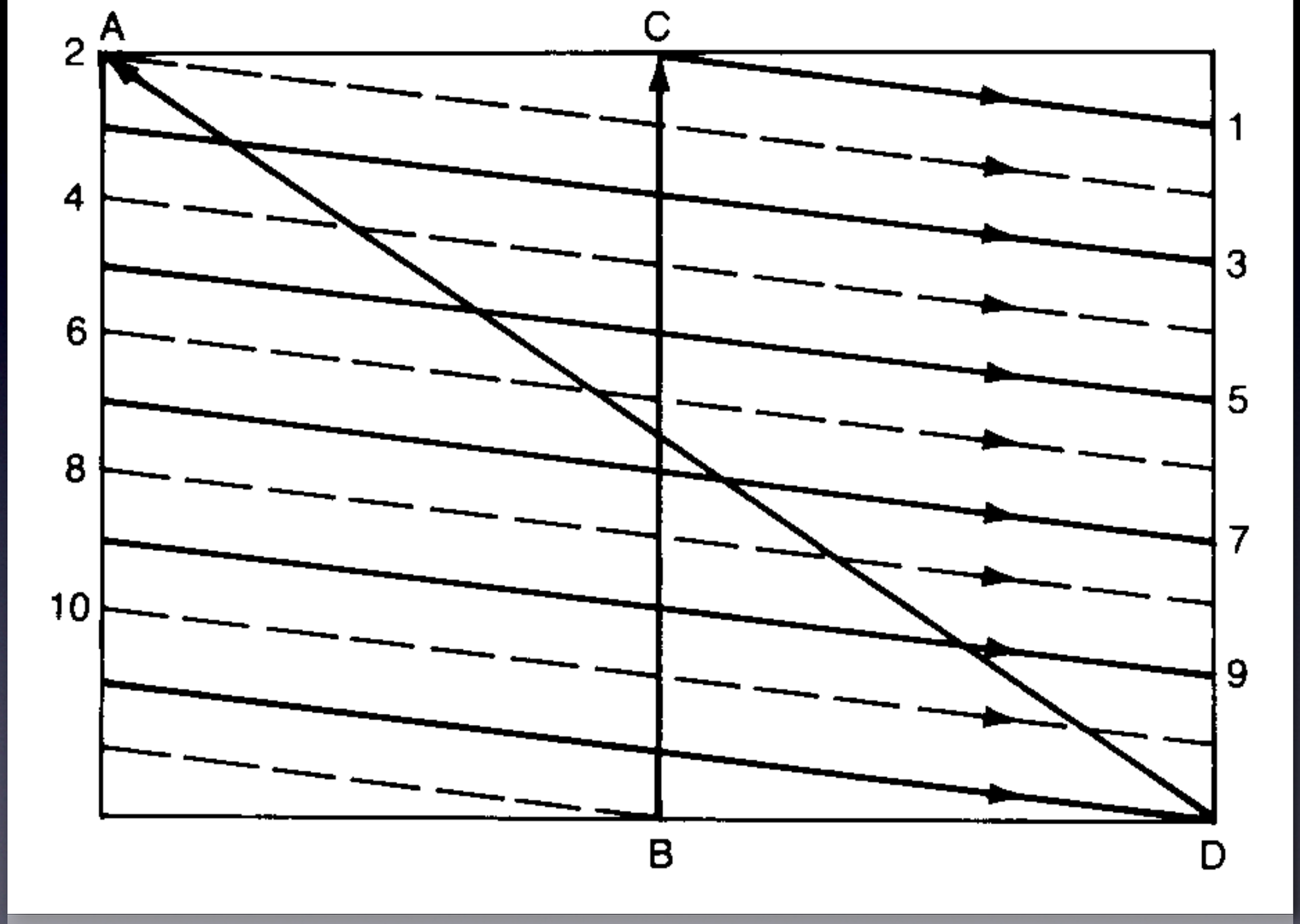
Introduction

- Considered how a computer stores text
  And how to lay it out…
- Also, how it can be described in a page description language
- How do we go from that to ink on a page?
- Look at how we draw text on screen…

(Or should that be pixels on the screen?)

Raster Display

- Need to understand how computers display images
- Very much tied to the way CRT monitors work…



Explain what happens

CRT Displays

- CRT beam swipes across screen from left to right
- Then jumps back to start of the next line on the left
- Eventually reaches the bottom of the display where it flies back to the top
- And repeats the process…

CRT Displays

- Happens so faster that the brain interprets it as a solid image
- Modulating the brightness of the CRT spot enables to construct the image, line-by-line
- By regularly reading bits from memory, the computer can modulate the CRT spot and produce an image

Raster Display

- Pixel clock specifies how fast bits are read from memory
- Providing the pixel clock is synchronized with the horizontal sync rate
- A fixed number of bits will appear per line
- By manipulating the bits in memory we can build up the image

Raster Display

- If the bit is on, the CRT spot is on (white)
- If the bit is off, the CRT spot is off (black)
- All bits are laid out sequential in memory
- But we know the line length and so can find the relevant line

Raster Displays

- All modern computer displays work like this
- Vary the number of bits used and arrangement to support color and greyscale
- Even printers use similar techniques…

Mention how dot-matrix printers scan using 9/24 pins vertically aligned

Displaying Text

- Displaying text is as simple as setting the relevant bits in the shape of the glyphs
- But how do we know which bits to set?

Fonts

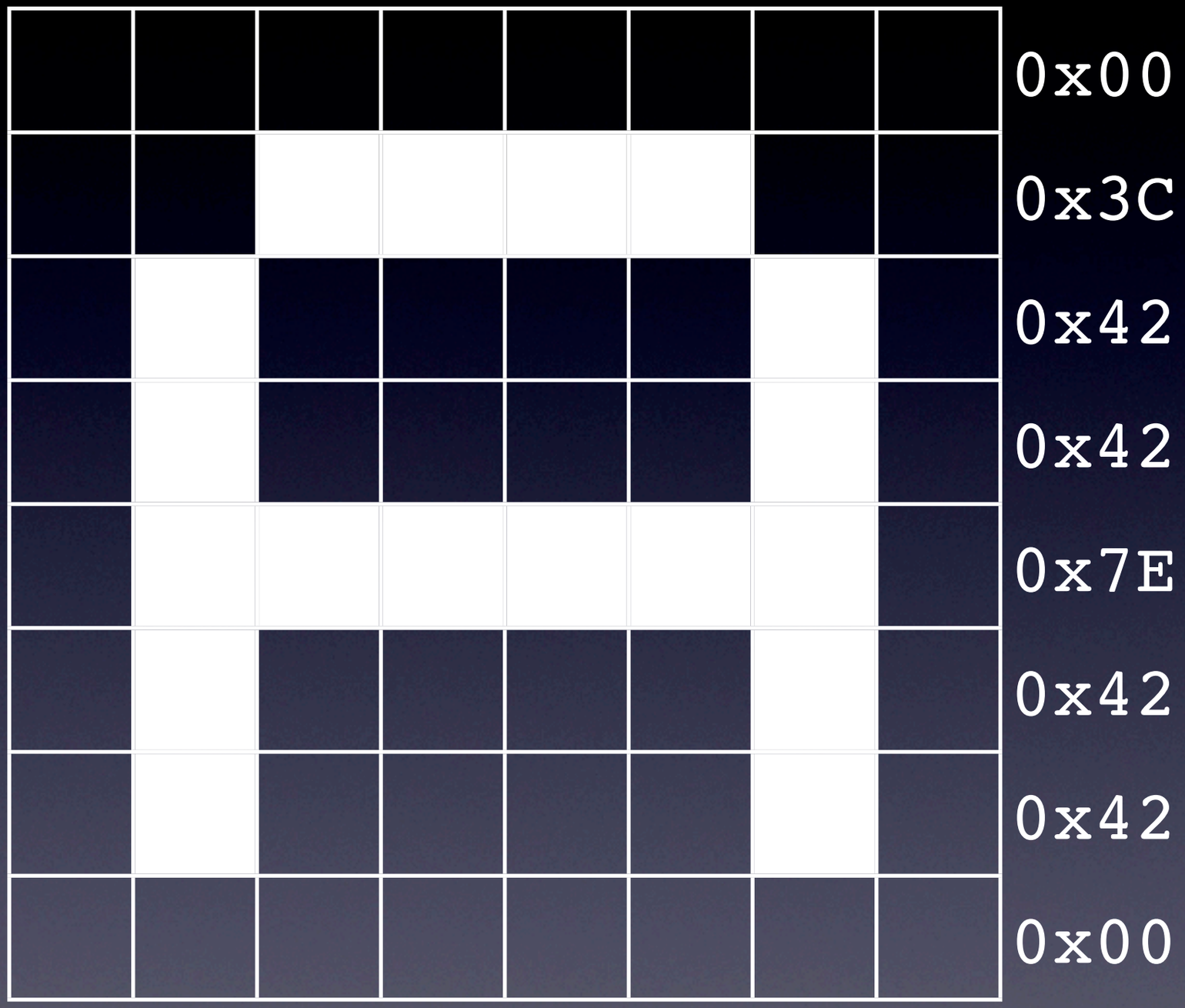
- Two main forms
  - Bitmap fonts — store the bit patterns for each glyph
  - Outline fonts — store instructions on how to draw the fonts

Bitmap fonts

- Stores the bit patterns for each glyph
- Computer then copies these bits into the relevant part of the screen memory
- Displaying the character on screen

Representing Glyphs

- Early computers had very simple font routines
- Each character was defined in (say) an 8x8 grid
- Therefore, each character takes up 8 bytes


0x00
0x3C
0x42
0x42
0x7E
0x42
0x42
0x00

Displaying Glyphs

- Just write those 8 bytes into screen memory — line width bytes apart
- Alternatively, OR the bytes into screen memory (preserving any backgrodun)
- AND and Exclusive-OR can be used for alternative effects

Hardware Characters

- Possible to implement all this in hardware
- Some systems would just write character codes into screen memory
- And let the hardware look up the bit patterns

Bitmap Font

- This approach works but has two problems
- All characters are the same width
  - A monospaced font
- Characters have to be imaged on a byte boundary (limiting the positions we can draw characters)

Shifting Glyphs

- Second problem is easy to solve
- If we bit shift the bytes containing the font data to the right
- Then when the bytes are written into the screen memory the character appears shifted to start at a different pixel
- But the right-hand edge will be cut-off

Shifting Glyphs

- Can stop the bits being chopped off by moving the byte into a 16-word
- Put the graphic data in the top 8 bits
- Shift to the right by *n* bits as needed…
- Then write the 16-bit word to the screen

Probably actually quicker to place in the bottom 8-bits and shift left

Proportionally Spaced

- Use the same approach to allow for proportionally spaced fonts
- Design the characters on a grid that is a multiple of 8 pixels
- But also encode a width value
- Advance that distance and draw the next character (shifting the pixels as needed)

Need to OR the bits into screen or we'll chop off parts of the display

Bitmap Fonts

- Relatively simple to implement and fast…
- Can have varying sizes by using multiple bytes per character width
- But every point size needs its own set of bitmap
- And every resolution (so separate fonts for screen and printer)

Bitmap Fonts

- Offer the best quality
- Designer can hand tweak every pixel so the characters look right
- But they have to create a lot of them…
- Don Knuth created METAFONT which generates bitmap fonts from outline descriptions…

Outline Fonts

- An alternative is to describe the font in terms of how it's drawn
- Vector-based
- Then scaled on demand to the right size
- Slower than a bitmap font since we need to 'rasterize' each glyph

Font Caching

- Most good font engines actually cache the bitmap representations of glyphs
- Then paint the cached bitmap rather than rasterizing the glyphs each time it is used
- Speed things up massively

Type 3 fonts

- PostScript Type 3 fonts are a simple example
- Each character's outline is stored as a simple PostScript procedure
- For each character that is imaged — the relevant procedure is called…

Talk them through the implementation of the chess font

Outline Font Problem

- There's a problem with this
- As we get smaller and smaller point sizes
- There's an increased risk that parts of the font will disappear (e.g. the cross piece of a letter 'e')
- Because they become less than one pixel wide

Outline Font Problem

- Dealing with 1-bit pixels
- No greyscale
- The result is text that is difficult to read
- Need an approach that stops this happening
- Type 1 fonts offer ‘hinting’ to solve this problem…