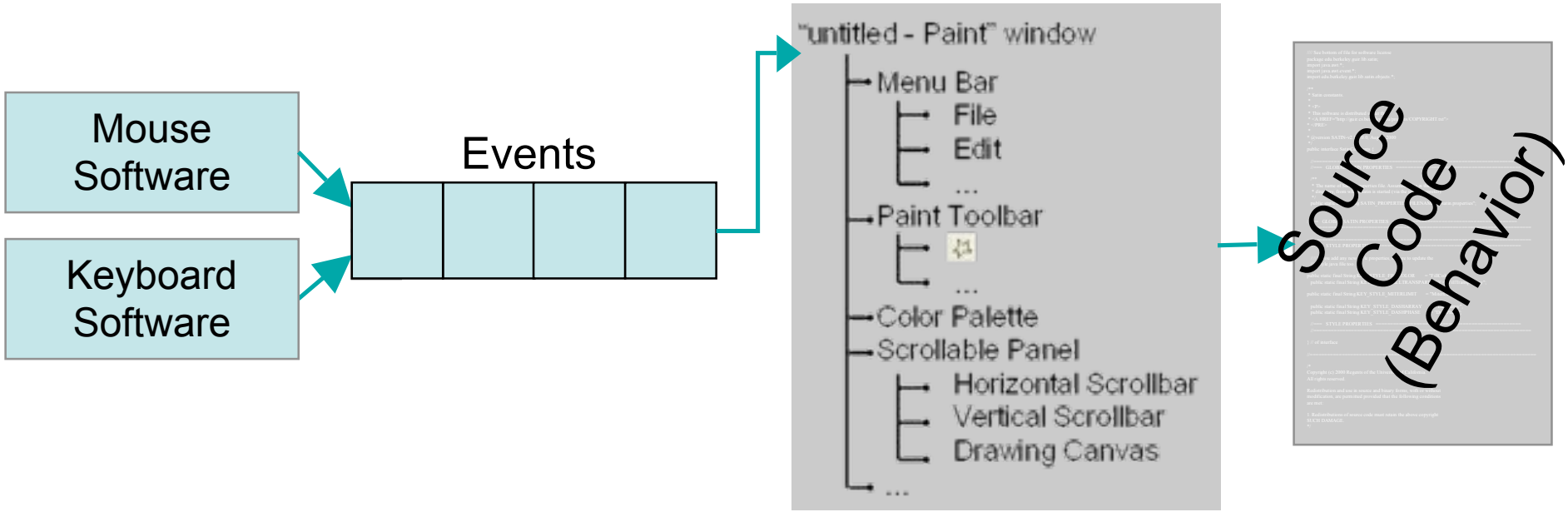
Basic GUI Output

Recap

- Three big ideas that glue GUIs together
  - Events
  - Widgets
  - Interactor Trees

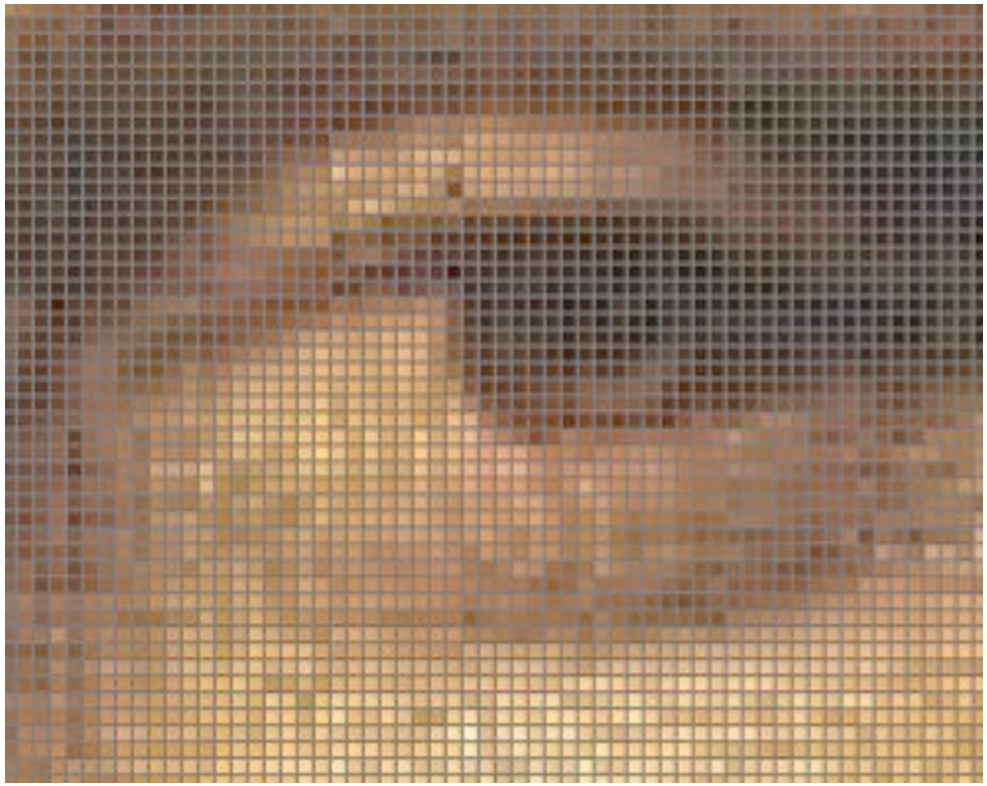


Outline

- Low-level graphical output models
  - Hardware: CRTs, LCDs, and displays
  - Color models
  - Raster (bitmap) operations
  - Lines, curves
  - Fonts
  - Affine Transforms
- Next lecture
  - Graphical output models for windows
- Later on in course (possibly)
  - Non-graphical output (sound, haptic / touch)

Human Perception and Displays

- Split a picture into a collection of small dots (pixels) and we can reconstruct it
  - resolution (ex. 1024x768) and pixels per inch (ex. 50ppi)



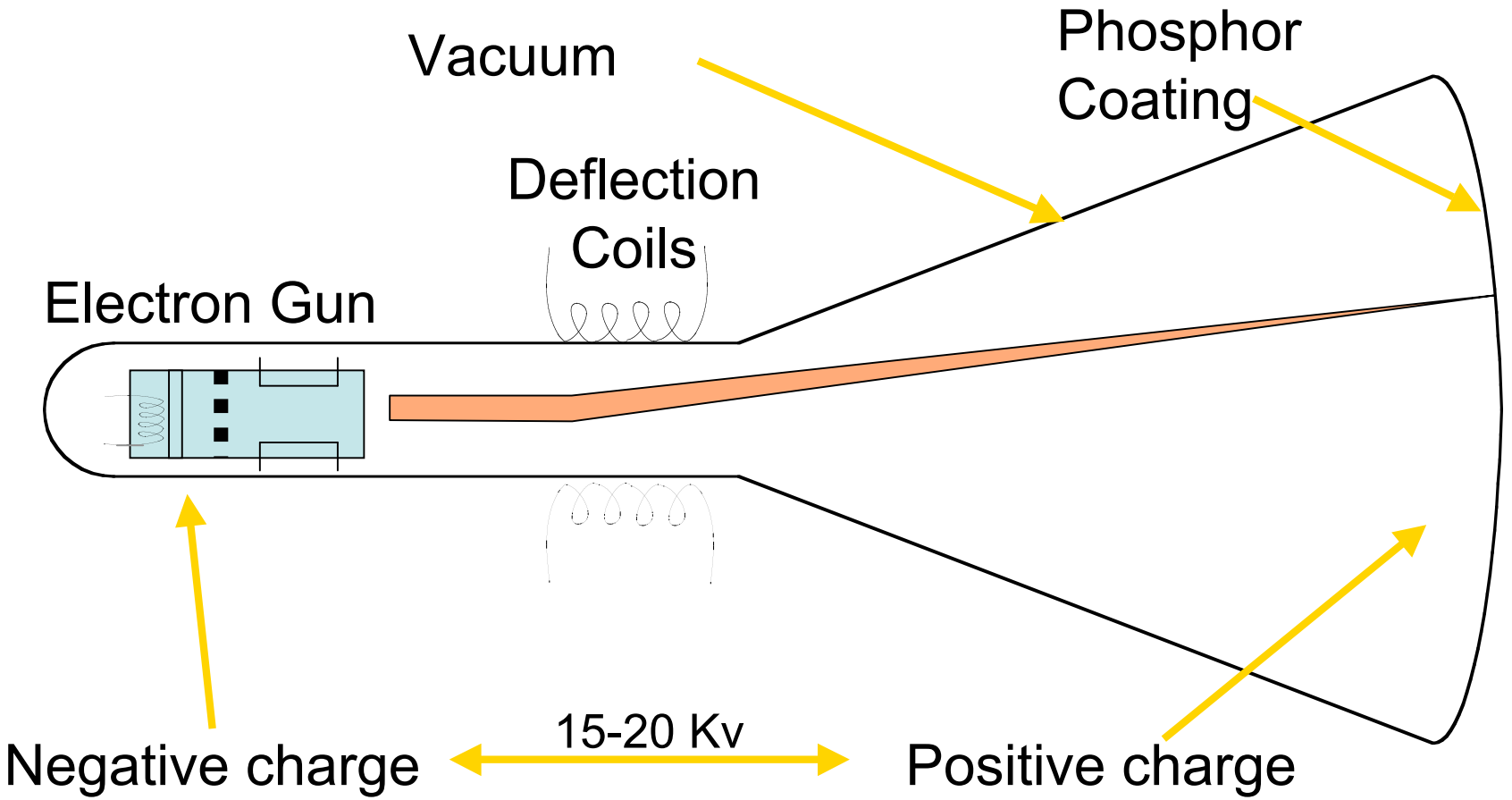
Display Devices

- How to display images?
  - Cathode ray tube (CRT)
  - Liquid crystal display (LCD)
  - Gas plasma
  - Light emitting diodes (LED)
- Most prevalent device: CRT
  - Cathode Ray Tube
  - AKA TV tube

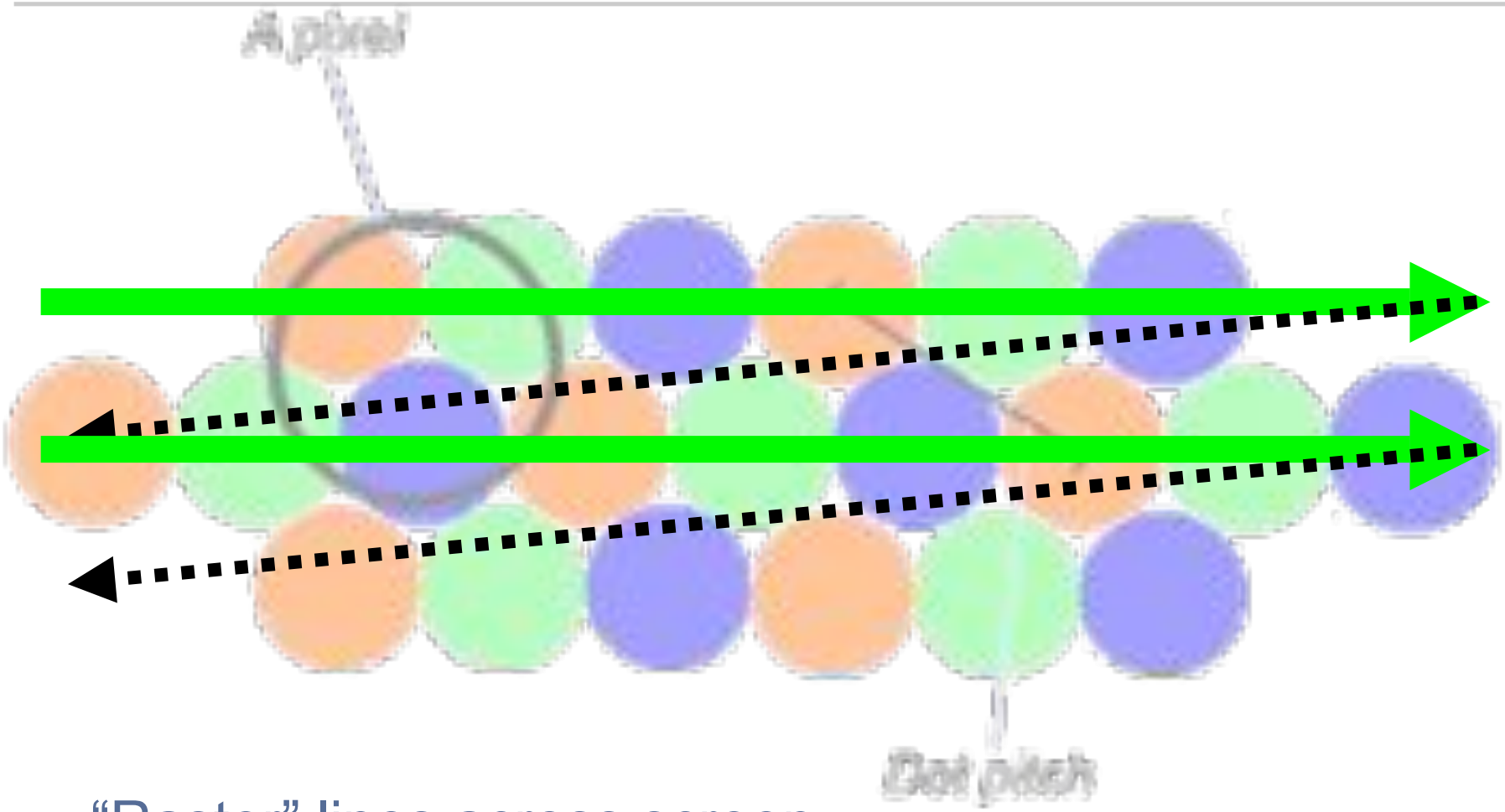
Cathode Ray Tubes

- Cutting edge 1930's technology
  - Invented in 1897
  - Uses a vacuum tube (big, power hog, …)
  - Refined some, but few fundamental changes
- But still dominant
  - Because TVs are consumer item
  - Many many years of work to make cheap
  - LCD's only recently a real challenger

How a CRT works



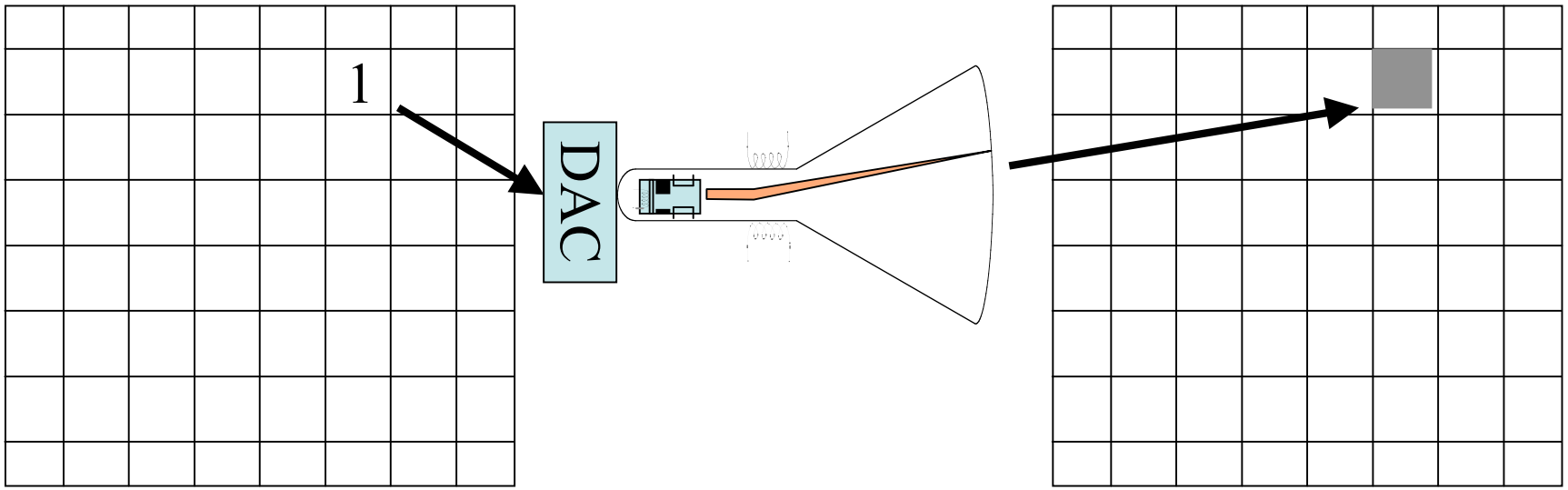
Phosphors on Screen



- “Raster” lines across screen
- Modulate intensity along line (in spots) to get pixels

Pixels Determined by Frame Buffer

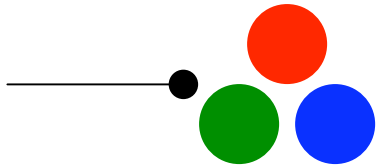
- "Frame buffer"
  - 2D array of memory of intensity values
  - Each memory cell controls 1 pixel



- All drawing done by placing desired values in memory
  - Odds low you will ever interact at this layer (drivers)

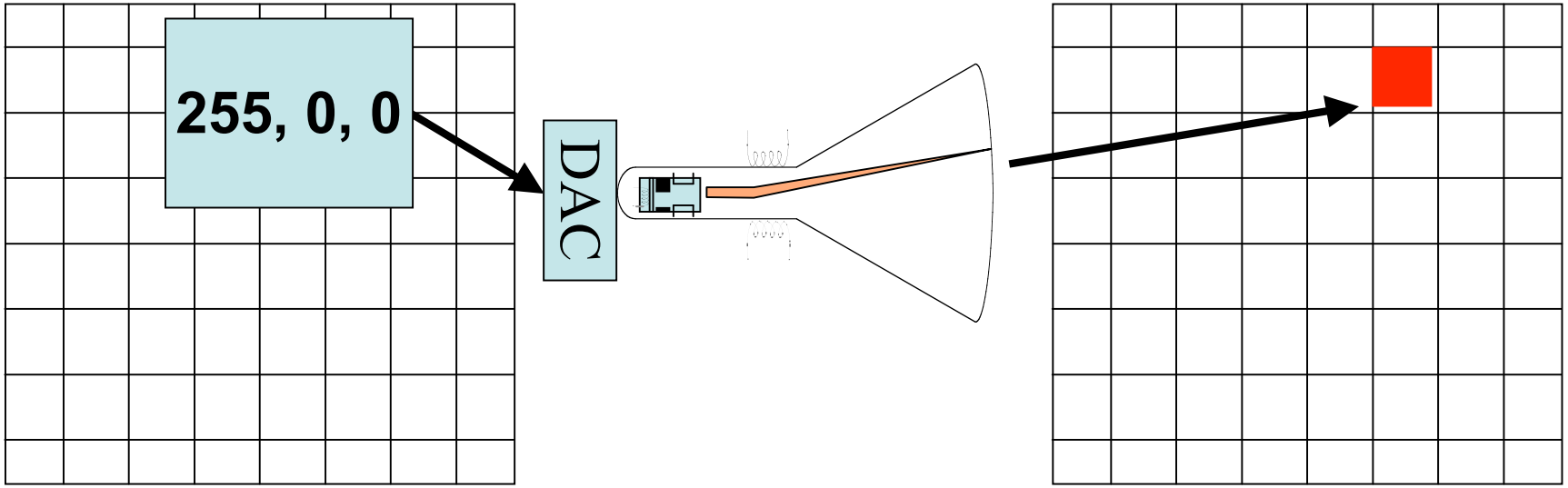
Adding Color

- Use 3 electron guns
- For each pixel place 3 spots of phosphor (glowing R, G, & B)
- Arrange for red gun to hit red spot, etc.
  - Requires a lot more precision than simple B/W



Color Frame Buffer

- Frame buffer now has 3 byte values for each pixel
  - each value drives one electron gun
  - can only see ~ 2^8 gradations of intensity for each of RGB
  - 1 byte each → 24 bits/pixel → full color
    - 16,777,216 color depth



Limitations of CRTs

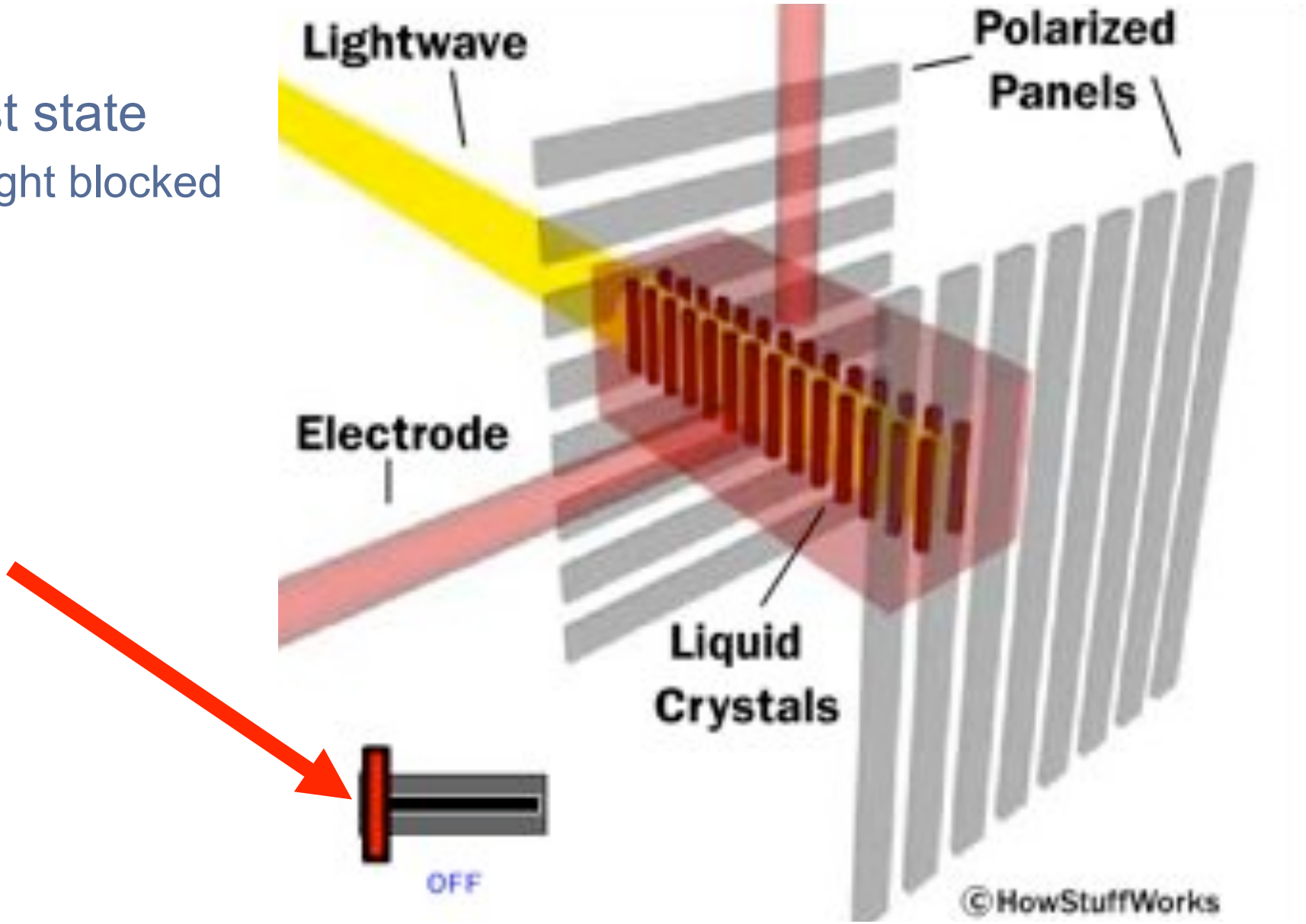
- Screen size limitation
  - 36” diagonal (why?)
- Bulky
  - LCD and Plasma display alternatives

Other Display Technologies: LCD

- Liquid Crystal Display
- Liquid crystal has unusual physical properties
  - rest state: rotates polarized light 90°
  - voltage applied: passes as is

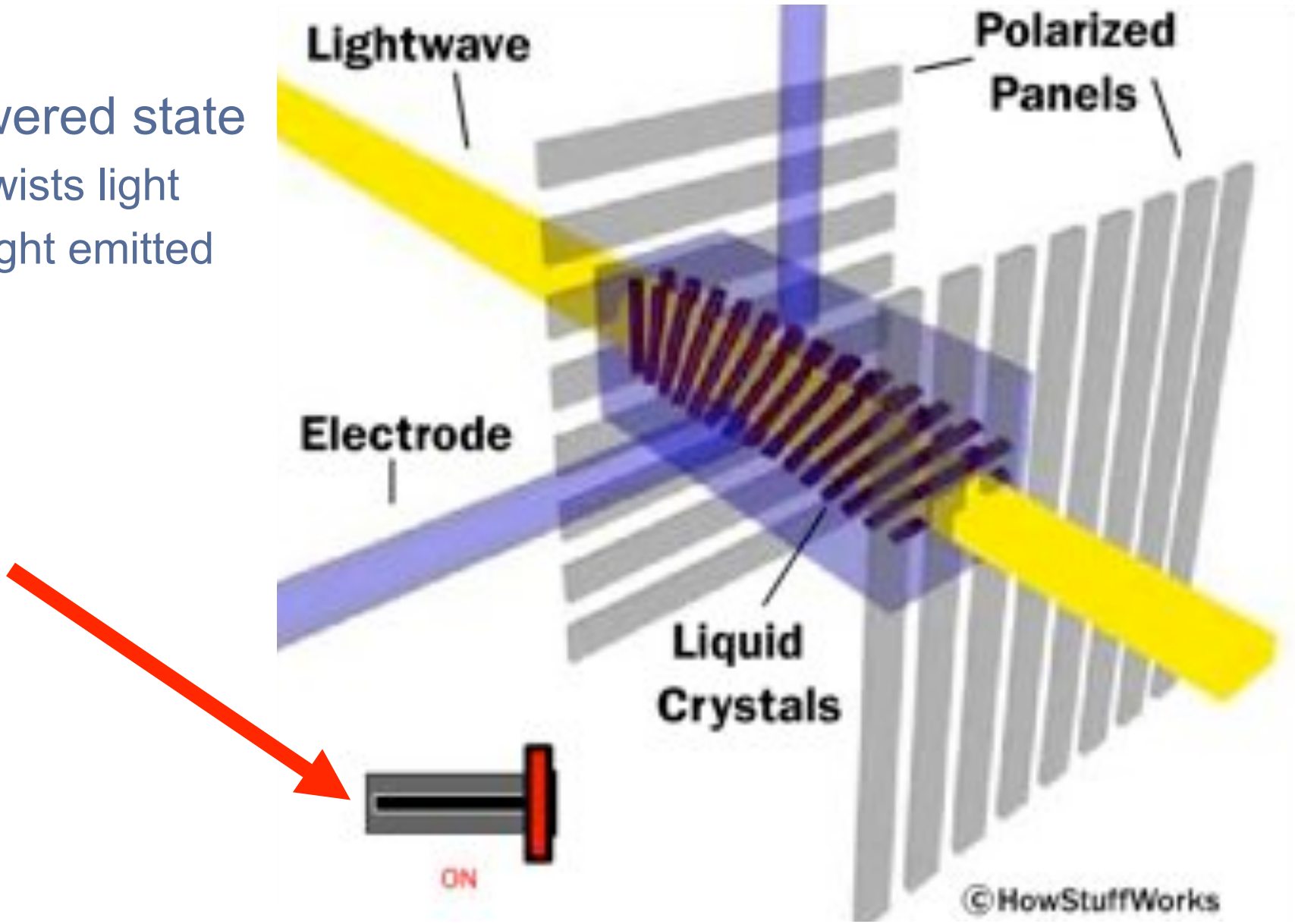
Layered display

- Rest state
  - light blocked



Layered display

- Powered state
  - twists light
  - light emitted



Types of LCDs

- Reflective vs. Backlit
- Passive matrix vs. Active matrix
  - Passive uses grid of conductive metal to charge each pixel
  - Active uses a capacitor for each pixel
  - Passive-matrix cheaper, but slower and lower contrast

CRTs vs LCDs

CRT

- Less expensive
- More accurate color
- More responsive (ghosting, flicker)
- Better resolution
- Harder to damage

LCD

- Less power
- Weigh less
- Easier to adjust
- Less eye strain

- But LCDs are gaining on CRTs

How Plasma Displays Work

- Plasma is a gas of xenon and neon atoms
  - Add electrons to excite atoms and produce ultraviolet
- Small fluorescent tubes providing RGB
  - UV hits RGB phosphors
  - Vary the intensities of tubes to produce full range of colors
- Pros
  - Thin
  - Very good color
  - Very large screens
- Cons
  - Price
  - Quality diminishes over time

How an LED works

- Invented in 1962 by Nick Holonyak, Jr.
- Electrons moving through semiconductor diode emit light
- Long-lasting, durable, and efficient
- Some examples
  - Digital clocks, Jumbo TVs, traffic lights, optical mouse, remote controls, entertainment devices
  - Blue LEDs since late 1990s

Other Interesting / Cool Technologies
Hi-res Displays

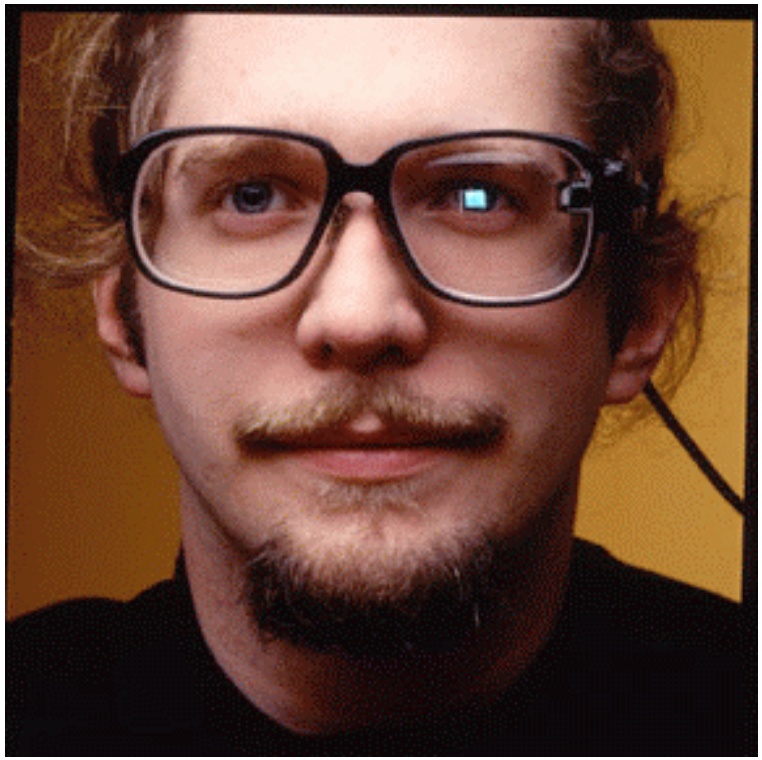
- IBM Roentgen
  - 200 ppi 16 inch display
  - 2560x2048 pixels (5.2 full color pixels)

Other Interesting / Cool Technologies
Hi-res Displays

- IBM Bertha
  - 204 ppi display
  - 3840x2560 pixels (9.8M full color pixels)
  - $9,000

Other Interesting / Cool Technologies
Wearable Displays

- Small displays that can be easily worn



Other Interesting / Cool Technologies

Direct Retinal Displays

- Direct retinal displays
  - University of Washington HIT lab
- Set of 3 color lasers scan image directly onto retinal surface
  - Scary but it works
  - Very high contrast, all in focus
  - Potential for very very high resolution
  - Has to be head mounted

Other Interesting / Cool Technologies
Wooden Mirror

- Art piece by Daniel Rozin

Other Interesting / Cool Technologies

Multiple Displays

Other Interesting / Cool Technologies

Seam-awareness

- Mackinlay, Heer 2004
- Observation:
  - Seams in between LCDs distorts views
- Idea:
  - Make apps seam-aware

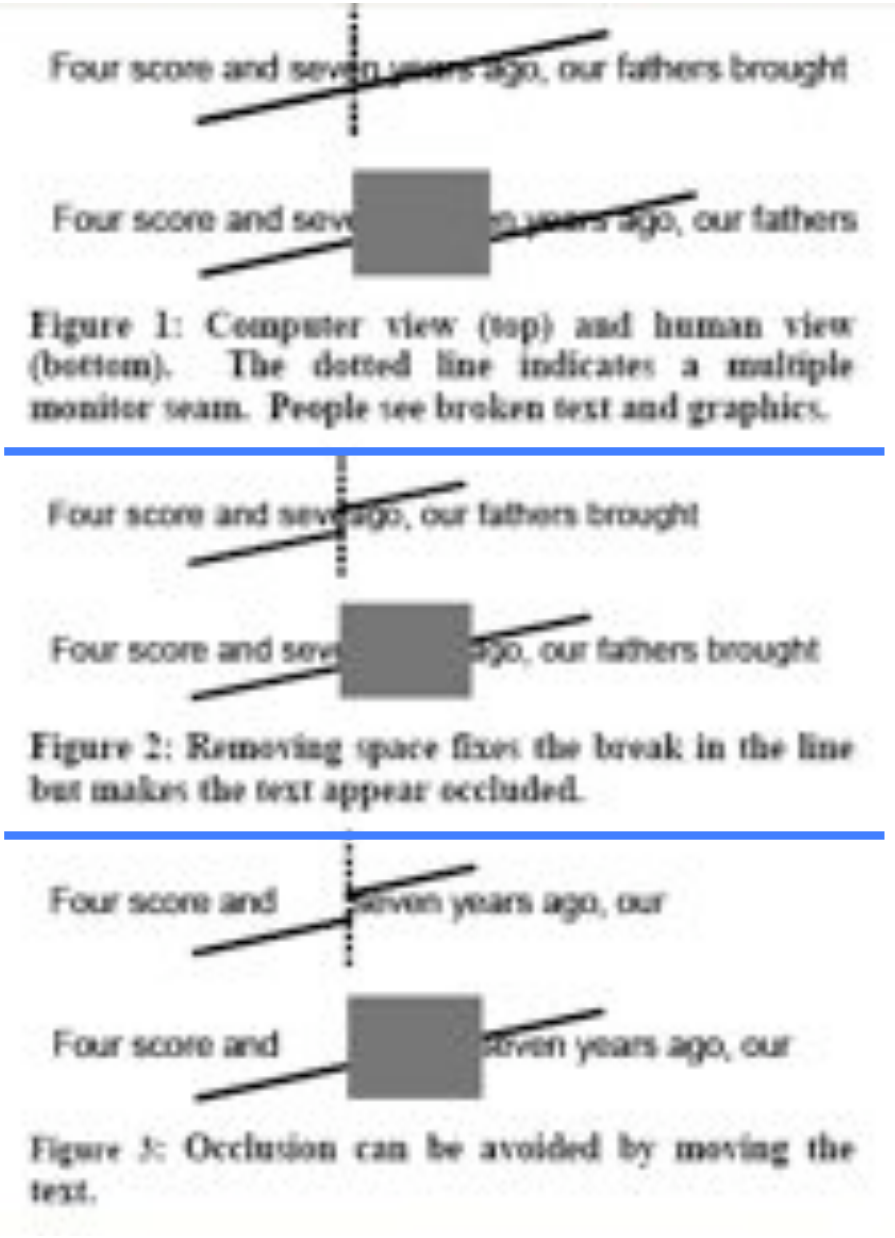


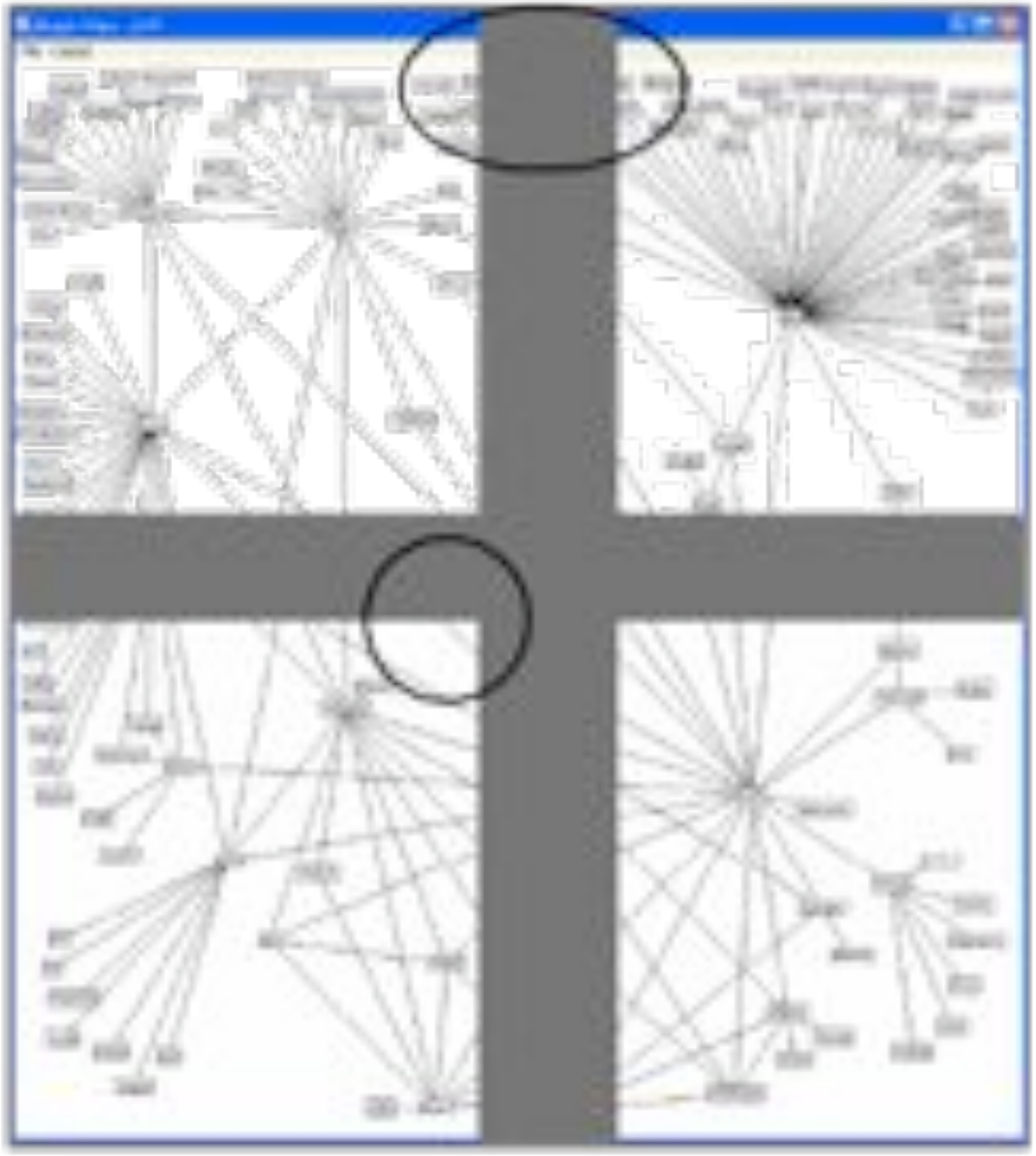
Figure 1: Computer view (top) and human view (bottom). The dotted line indicates a multiple monitor seam. People see broken text and graphics.

Figure 2: Removing space fixes the break in the line but makes the text appear occluded.

Figure 3: Occlusion can be avoided by moving the text.

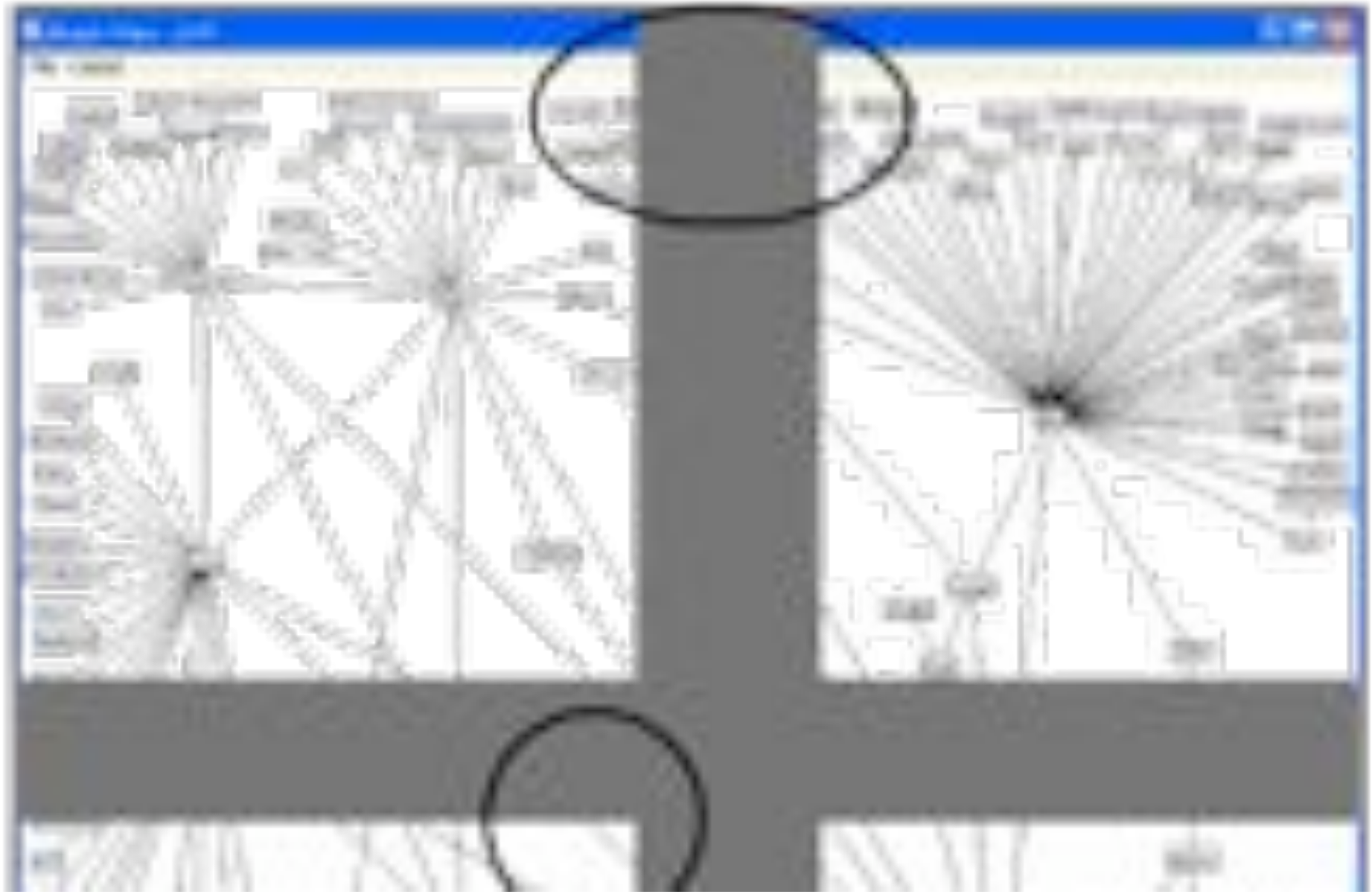
Other Interesting / Cool Technologies

Seam-awareness



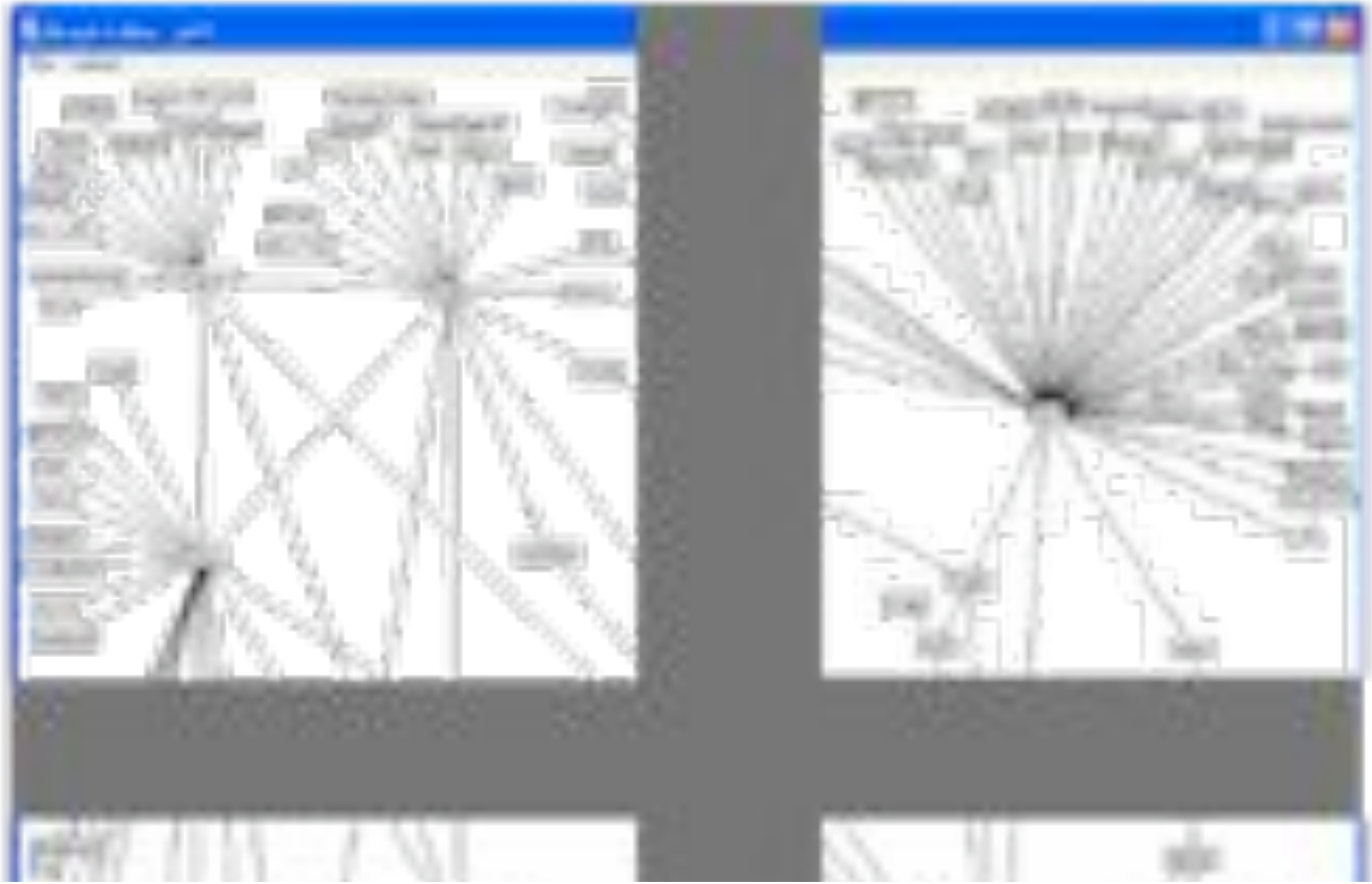
Other Interesting / Cool Technologies

Seam-awareness



Other Interesting / Cool Technologies

Seam-awareness

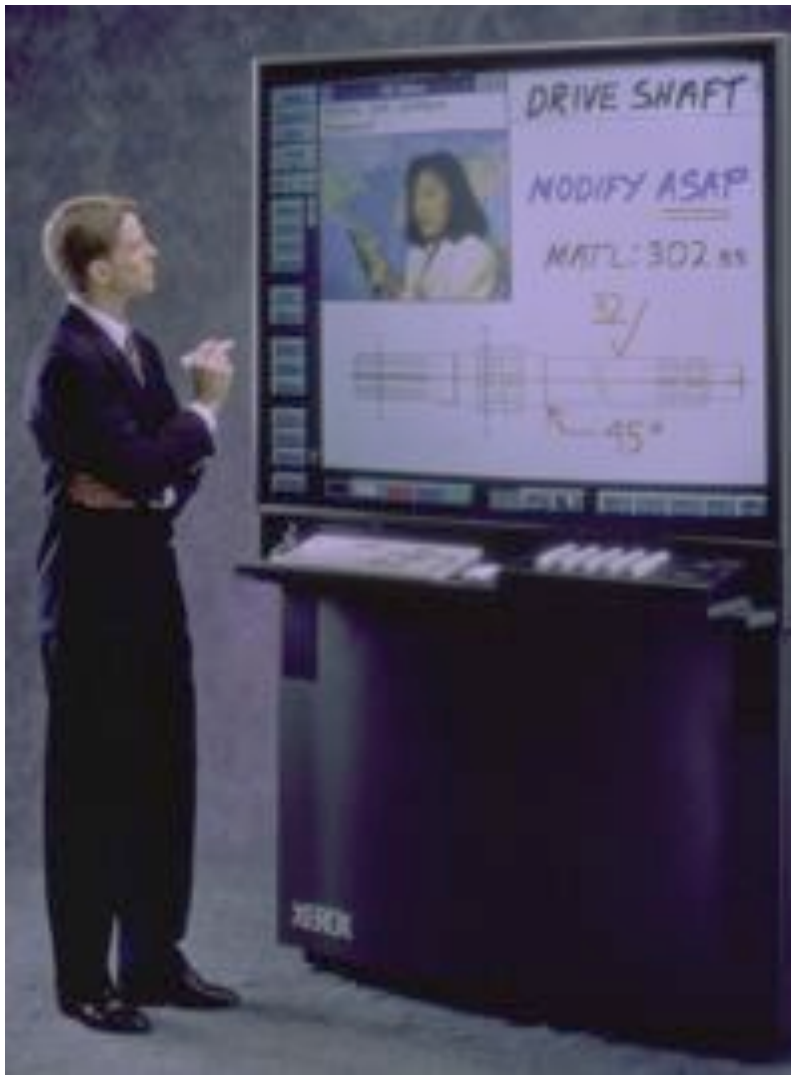


Other Interesting / Cool Technologies
Very Widescreen Displays

- Tan, Czerwinski, Robertson 2003
- Women do not do as well as men in 3D navigation on regular displays
- But performed comparable to men with wider screens and better 3D animations

Other Interesting / Cool Technologies
Very Large Displays

- Notifications and Start menu?
- How to reach menu bar?



Break

- Any questions / comments?

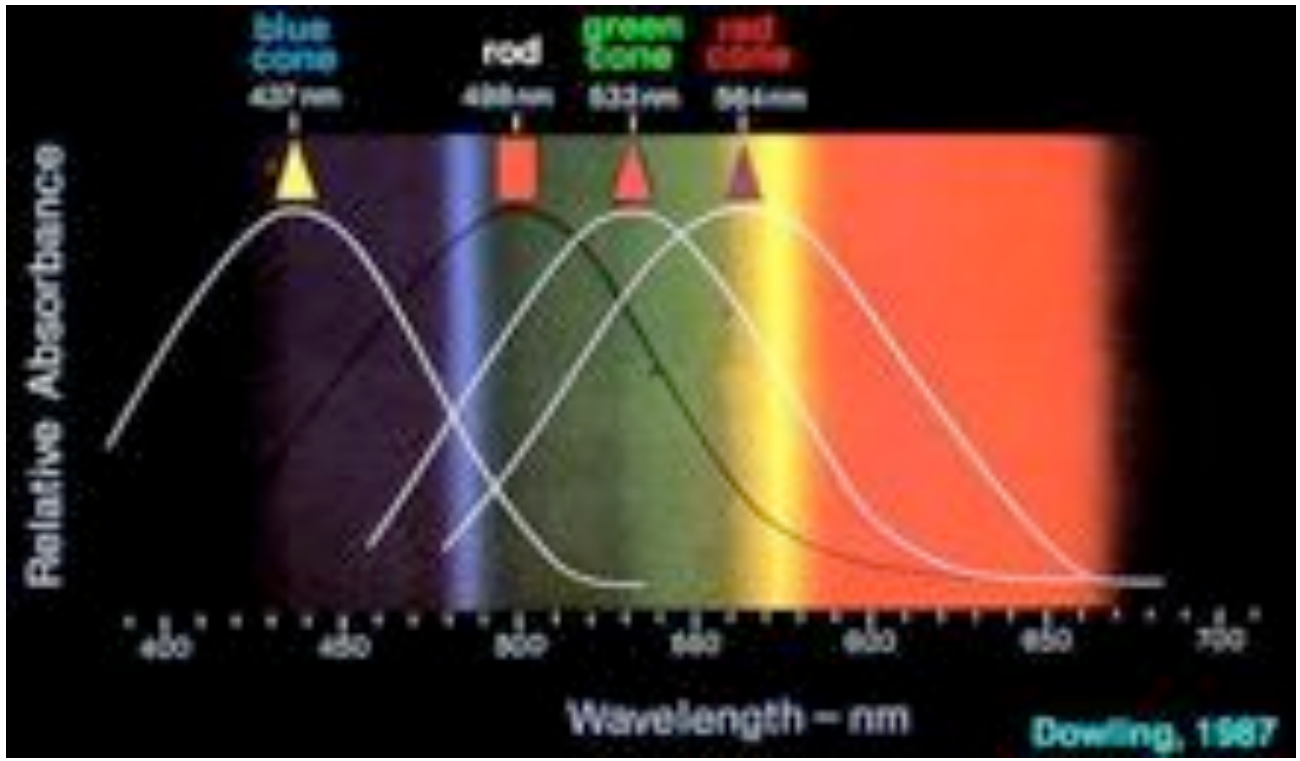
All these systems use a frame buffer

- Each pixel has 3 values
  - Red, Green Blue
  - 1 byte each → 24 bits/pixel → full color
    - 16,777,216 color depth

- Why RGB?
  - R, G, and B are particular frequencies of light
  - Actual light is a mix of lots of frequencies
  - Why is this enough?

Why RGB are enough

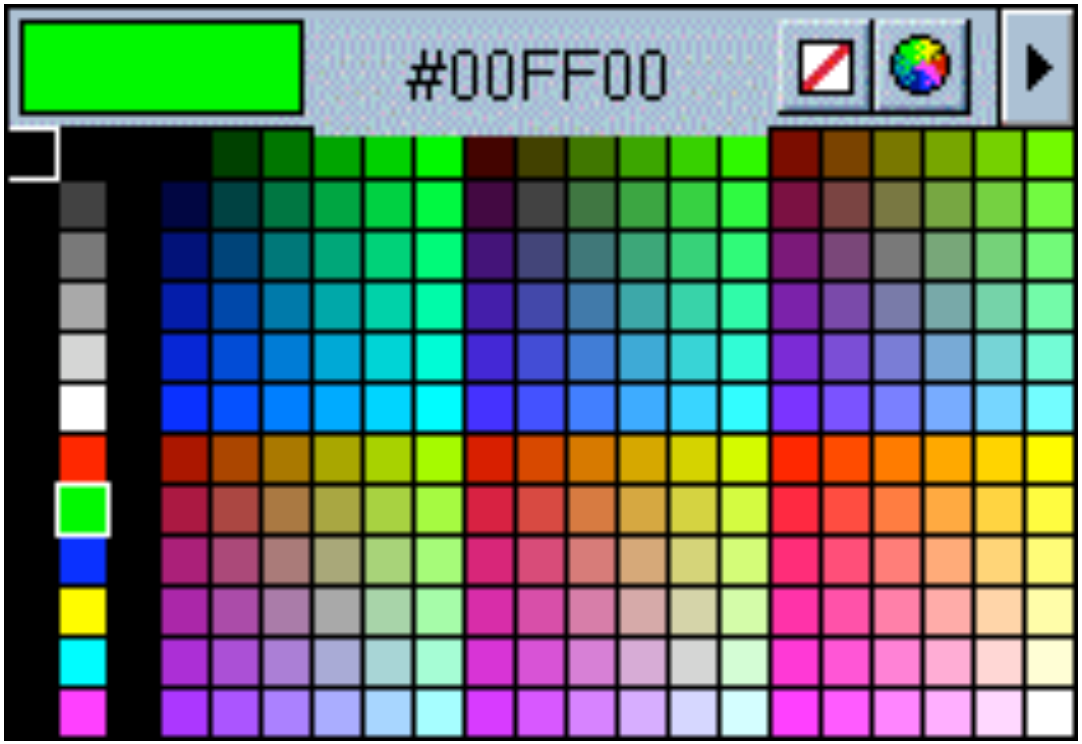
- Eye has receptors (cones) that are sensitive to (one of) these
  - Eye naturally quantizes/samples frequency distribution



from http://insight.med.utah.edu/Webvision/index.html

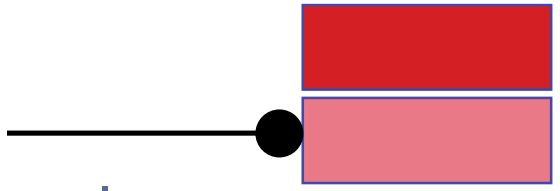
Technology-Centered Color Model

- Color usually programmed through RGB
- However, does not match how we think about colors
  - Especially artists and interior designers



HSV Color Model

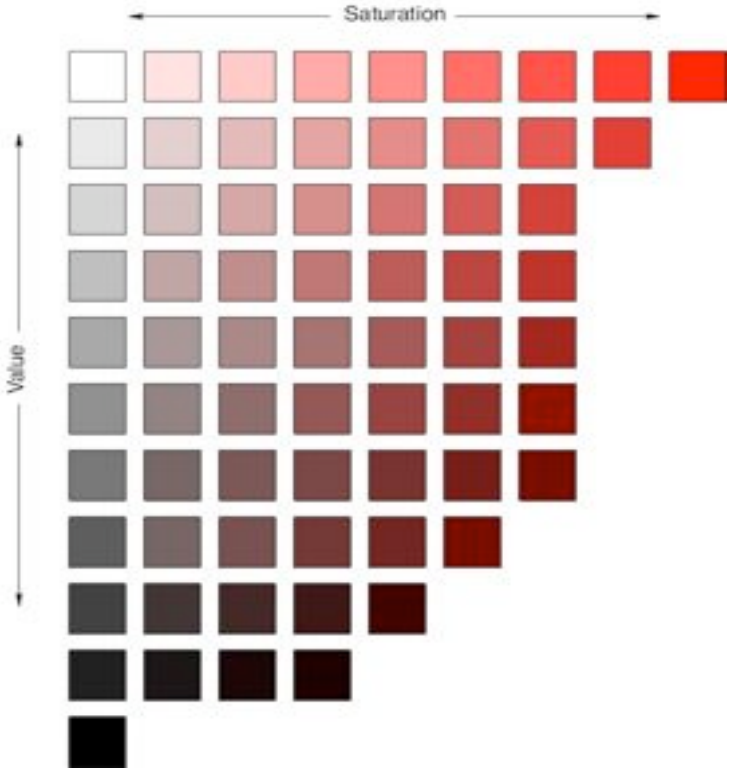
- Hue
  - property of the wavelengths of light (i.e., “color”)
- Saturation
  - purity of the hue
    - e.g., red is more saturated than pink



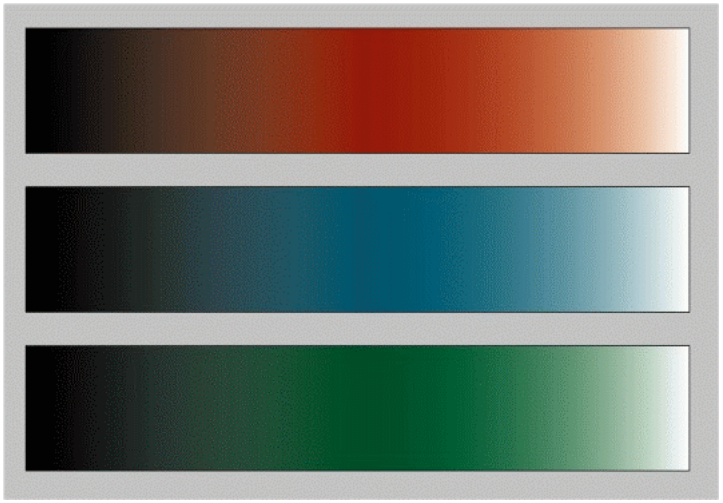
  - color is mixture of pure hue & achromatic color
    - achromatic: a color lacking hue; white, gray, or black
    - portion of pure hue is the degree of saturation
- Value (or Lightness or Brightness)
  - how much light appears to be reflected from a surface
  - some hues are inherently lighter (yellow) or darker (blue)

Color Components (cont.)

- Saturation



- Value



from http://www2.ncsu.edu/scivis/lessons/colormodels/color_models2.html#saturation.

Color Components (cont.)

Color Model Summary

- RGB easy to program
  - Close to hardware
  - Pretty much universally supported on all platforms

- HSV is easier for people to use
  - Uses people's intuition of what color is
  - There is a direct conversion to RGB

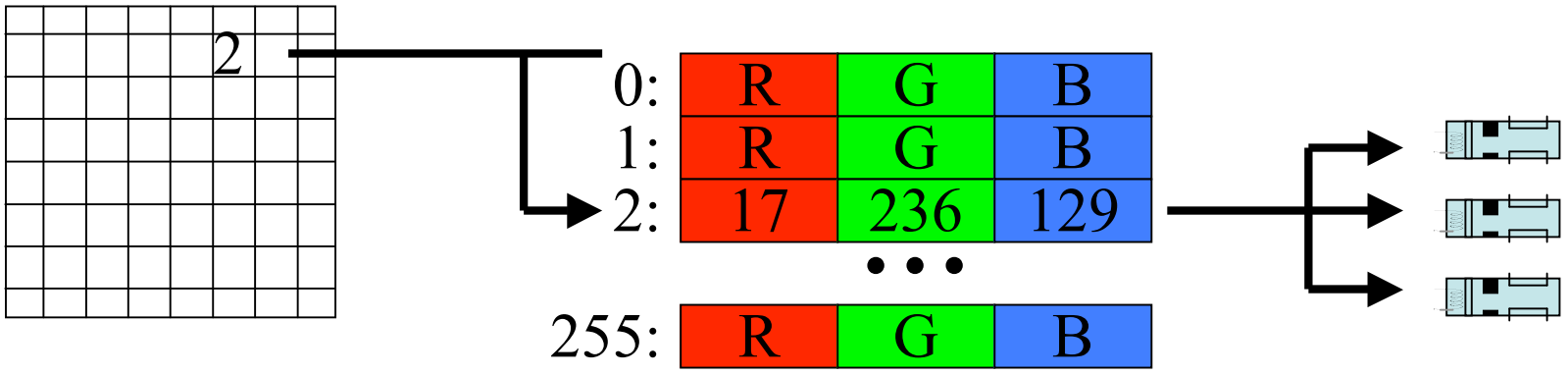
- Other colors models:
  - CMYK: mixing pigments cyan, magenta, & yellow (printing)

What if we have less than 24 bit color?

- Back to RGB…
- If 16 bits/pixel…
  - Can have 5 each in RGB with 1 pixel left over
  - Decent range (32 gradations each, 32K colors)
- If 8 bits/pixel…
  - 3 bits for GB, 2 for R
  - not enough for anything useful
  - Use a “trick” instead
- Thoughts?

Color lookup tables (CLUTs)

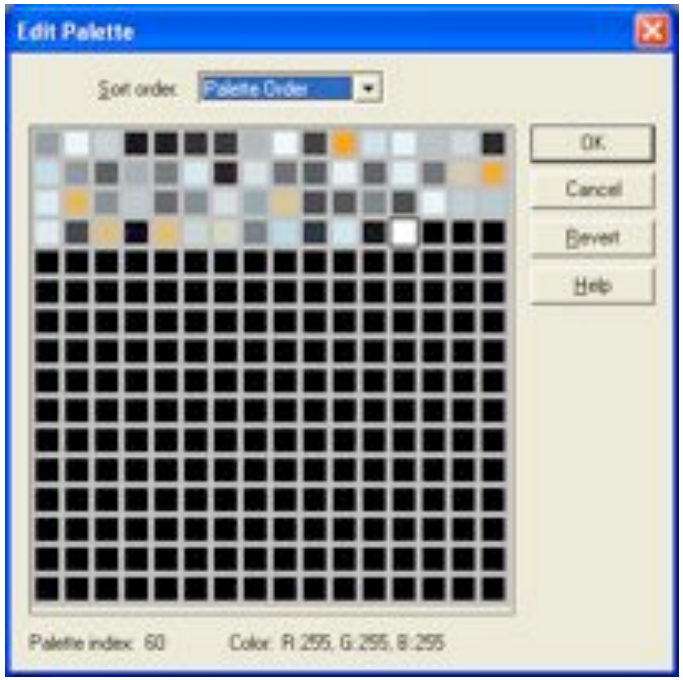
- aka Color Mapping
  - aka Level of indirection
- Extra piece of hardware
  - Use value in Frame Buffer as index into a CLUT
    - e.g. 8 bit pixel → entries 0…255



  - Each entry in CLUT has full RBG value used to drive 3 guns

Palettes

- 8 bits / pixel with Color Lookup Table
  - Gives “palette” of 256 different colors
  - Chosen from 16M colors
  - Can do a lot better than uniform by picking a good palette for the image to be displayed (nice algorithms for doing this)
- Same basic idea behind GIF images as well
  - 256 color palette selected from 16M



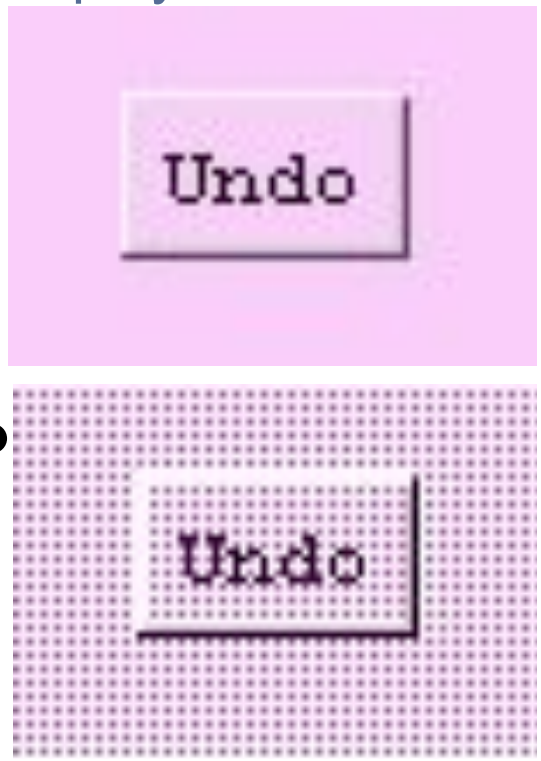
Color Maps with Window Systems?

- Recall that every window is virtual device
  - Thus has its own set of colors
  - But physical device may be limited
    - Ex. can only render 256 colors total
- What to do if there are not enough colors?
  - Ex. Window A uses one color map, Window B another?

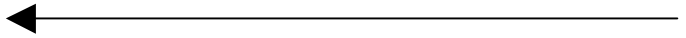
2-Minute Discussion

Color Maps with Window Systems?

- Fail (return with error)
  - Bad option
- Swap CLUTs based on active window
  - Ugly, and non-active windows still need to display
  - Also, still runs out of colors
- Add to color map (if possible)
  - Keep some slots in reserve for this
- Pick closest color
- Dither using available colors
  - Trade spatial resolution for color resolution



Outline

- Low-level graphical output models
  - CRTs, LCDs, and displays
  - Colors
  - Raster operations 
  - Lines, curves
  - Fonts
  - Affine Transforms

Raster-oriented Programming Model

- Raster == Bitmap
- This model pretty close to actual frame buffer HW
  - Integer coordinate system
  - 0,0 typically at top-left with Y down

  - All drawing primitives equivalent to filling in pixel color values in frame buffer

Most Primitive Raster Operation: Copy

- Copy an area of the screen

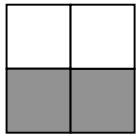
```
copyArea( int srcx,  int srcy,
          int w,     int h,
          int destx, int desty)
```

- Copies a rectangular area of the screen
  - Source rectangle to destination rectangle

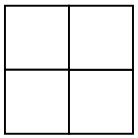
More sophisticated: BitBlt

- Fast BitBlt key to evolution of modern GUIs
  - Would not have been able to have effective graphics!
- Basic idea: combine pixels with values already there
- RasterOp (BitBlt)
  - First used for B/W only (1 bit color)
  - Boolean combination operators

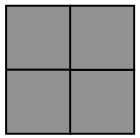
Src



Dest

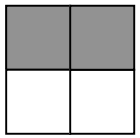


Clear (0)

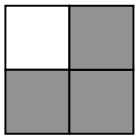


Set (1)

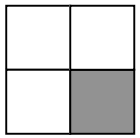
Copy



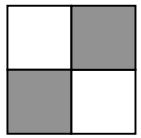
Not



OR

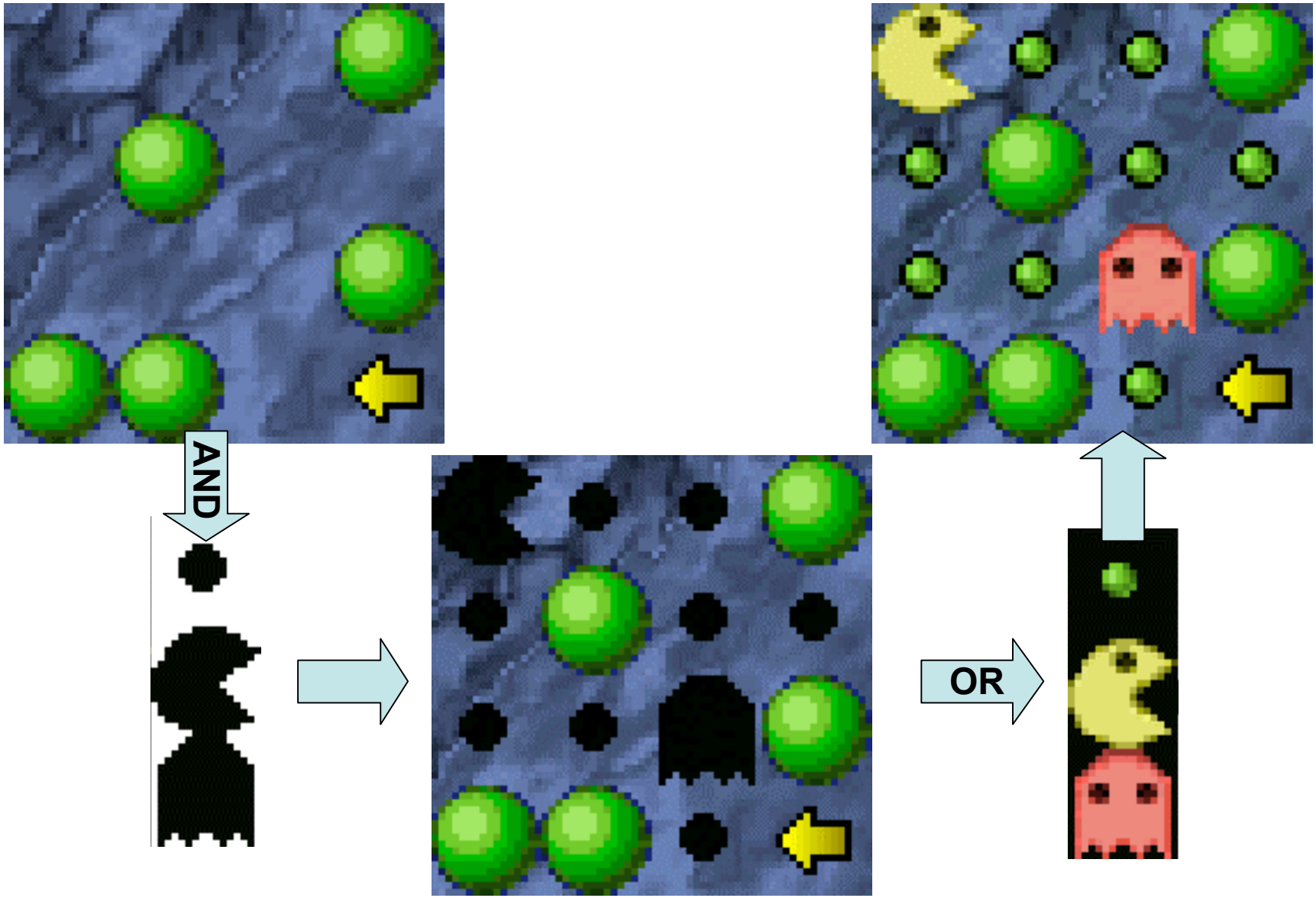


AND



XOR

More sophisticated: BitBlt



RasterOp Continued

- Other combination operators
  - 16 total including “not and”, “not or”
- XOR is particularly useful
  - A ^ 1(Black) == ~A
  - A ^ 0(White) == A
  - Selective inversion
  - A ^ B ^ B == A (basically, undo for any A and B)
  - Older displays, this was how mouse was drawn
- Digression: XOR swap trick for swapping in place
  - x := x ^ y
  - y := x ^ y
  - x := x ^ y

RasterOp Continued

- Note, XOR doesn't work as well in color
  - XOR well-defined (operates on bits)
  - But: Blue ^ Violet == ??

- Other combination ops make more sense for color
  - Transparency
    - weighted average of colors
  - "Alpha" values (RBGA) determine how much of source is "mixed" with existing destination colors
  - Leads to 32 bits (4 bytes) for colors
  - See `java.awt.Color`

Alpha Compositing

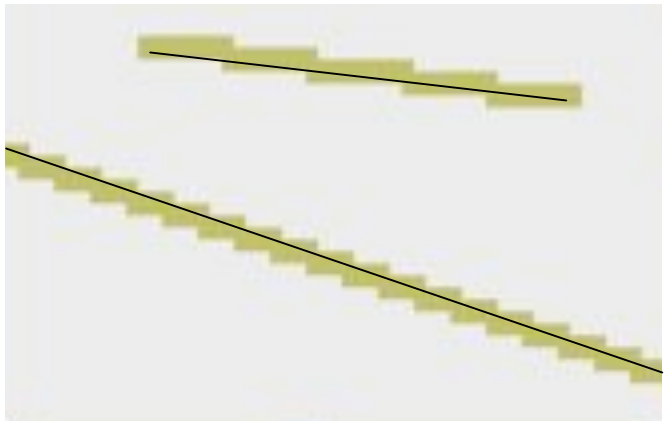
- Transparency trivial in Java

Outline

- Don’t want to program at bit-level, so…
- Low-level graphical output models
  - CRTs, LCDs, and displays
  - Colors
  - Raster operations
  - Lines, curves
  - Fonts
  - Affine Transforms

Drawing Primitives

- Support drawing primitives
  - Lines, rectangle, ovals, polylines, polygons, curves
  - "Scan conversion" algorithms to decide what pixels to set (won't cover here)
    - see e.g., Foley, van Dam, Feiner, & Hughes

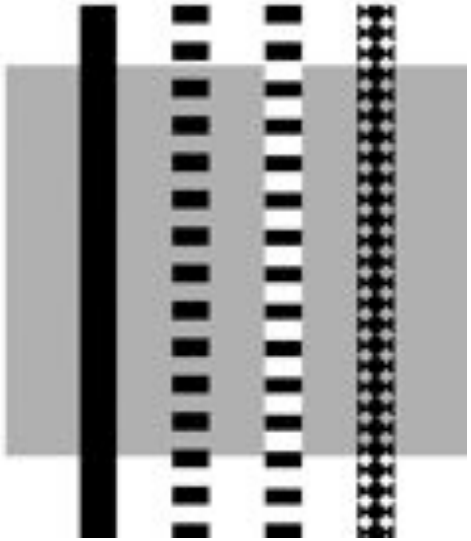


- Begin to abstract beyond "just pixels"

Line Properties

- Width

- Line styles
  - Solid, dashed 11100011100011000, "double-dashed", patterned

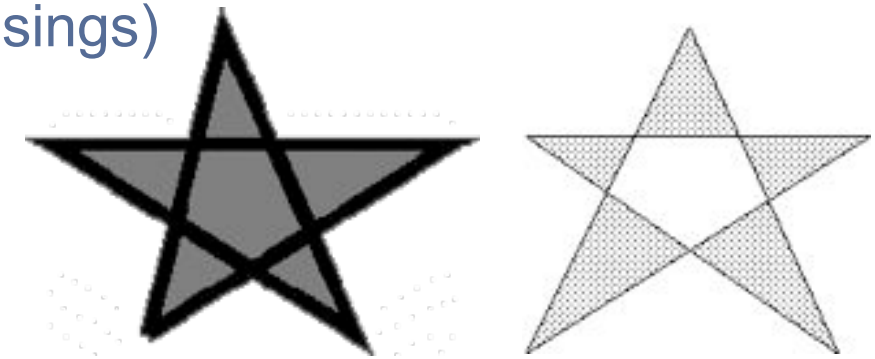


- Cap-style
  - butt, round, projecting (aka squared, by 1/2 line width)

Polylines and Polygons

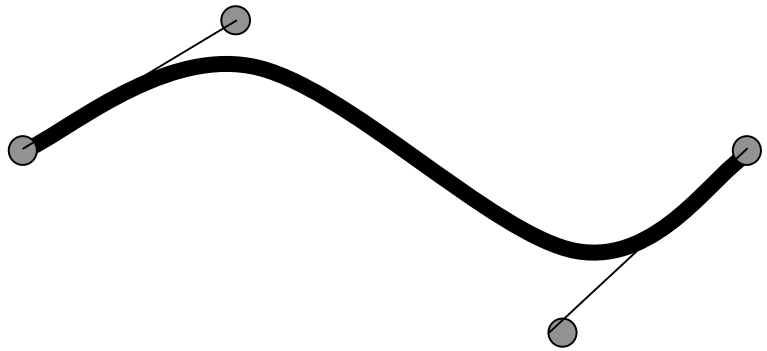
- End-caps:
  - Miter = point
  - Round = circle of the line width
  - Bevel = fill in notch with straight line

- Filled, what parts?
  - “Winding rule” determines what is “inside”
    - Non-zero
    - Parity / even-odd (#crossings)



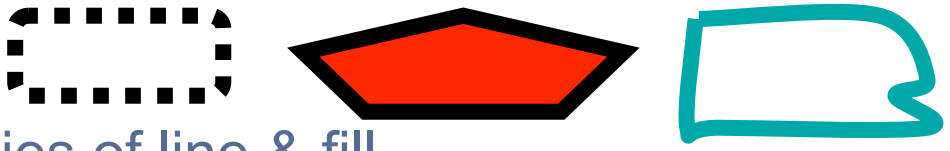
Curves (Splines)

- Curves defined by cubic equations
  - $x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$
    $y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$
  - Well-defined techniques from graphics (see e.g., Foley et al)
- Bézier curve defined by “control” points
  - Goes through 2 end pts
  - Other 2 define tangents



PostScript, PDF, Java2D Path Model

- Some models (ex. AWT) draw by drawing fixed shapes (drawRect, drawEllipse, etc.)
- Path model unifies:
  - Define a path first
    - General ops: moveTo, lineTo's, curveTo (etc.)
  - Then draw it
    - Stroke or fill
    - With various properties of line & fill

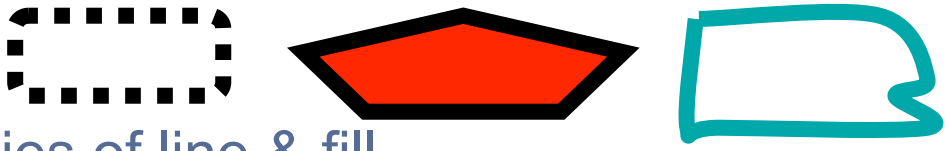


- Advantages of this approach(?)

3-Minute Discussion

PostScript, PDF, Java2D Path Model

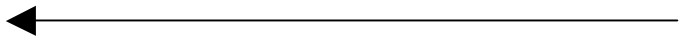
- Some models (ex. AWT) draw by drawing fixed shapes (drawRect, drawEllipse, etc.)
- Path model unifies:
  - Define a path first
    - General ops: moveTo, lineTo's, curveTo (etc.)
  - Then draw it
    - Stroke or fill
    - With various properties of line & fill



- Advantages of this approach(?)
  - Higher level abstraction providing more control and flexibility
  - Can handle higher resolutions better
  - Fewer bits to send (send a description vs bitmap)
  - If same model used for screen and printing, debugging

Outline

- Low-level graphical output models
  - CRTs, LCDs, and displays
  - Colors
  - Raster operations
  - Lines, curves
  - Fonts
  - Affine Transforms



Fonts and Drawing Strings

- Font provides description of the shape of a collection of chars
  - These shapes are called glyphs
- Plus information e.g. about how to advance after drawing a glyph
- Plus aggregate info for the whole collection

Fonts

- Typically specified by:
  - Family or typeface
    - e.g., courier, helvetica, times roman
    - Some fonts are sans serif
    - Some fonts are serif
    - Some fonts are monospaced
  - Size (normally in “points”)
  - Style
    - e.g., plain, *italic*, **bold**, ***bold & italic***
    - other styles: underline, ~~strikethrough~~, emboss, shadow

Points

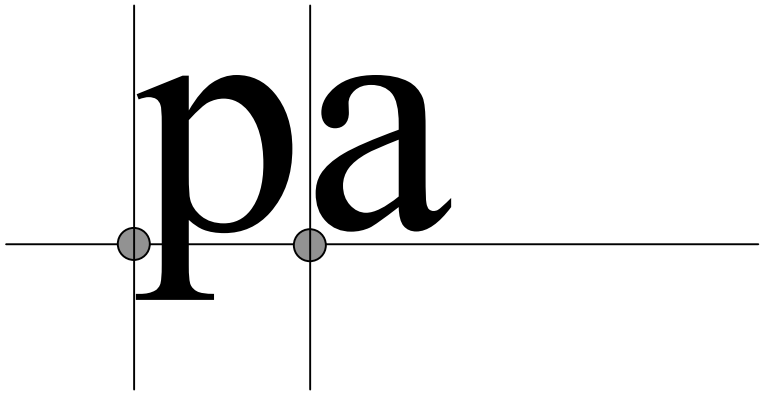
- An odd and archaic unit of measurement
  - 72.27 points per inch
    - Origin: 72 per French inch (!)
  - Postscript rounded to 72/inch most have followed
  - Early Macintosh: point == pixel

Reference point and baseline

- Each glyph has a reference point
  - Draw a character at (x,y) reference point will end up at (x,y) (not top-left)
  - Reference point defines a baseline

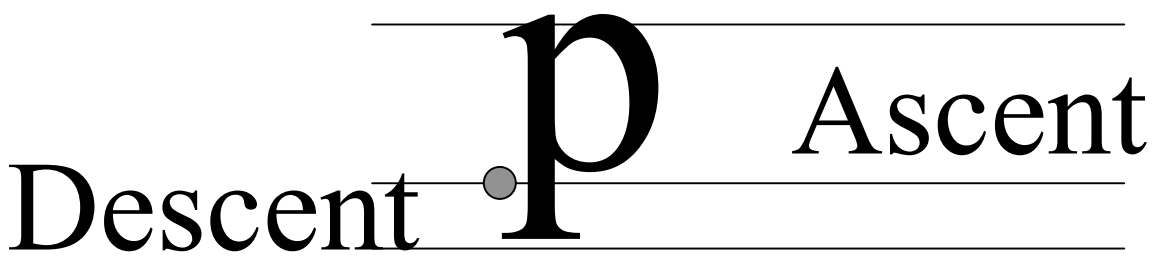
Advance width

- Each glyph has an “advance width”
  - Where reference point of next glyph goes along baseline



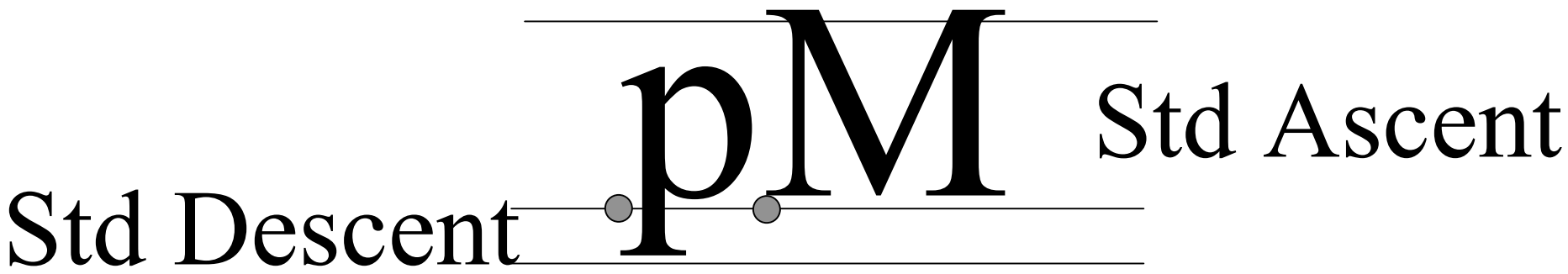
Ascent and Descent

- Glyphs are drawn both above and below baseline
  - Distance below: “descent” of glyph
  - Distance above: “ascent” of glyph



Standard Ascent and Descent

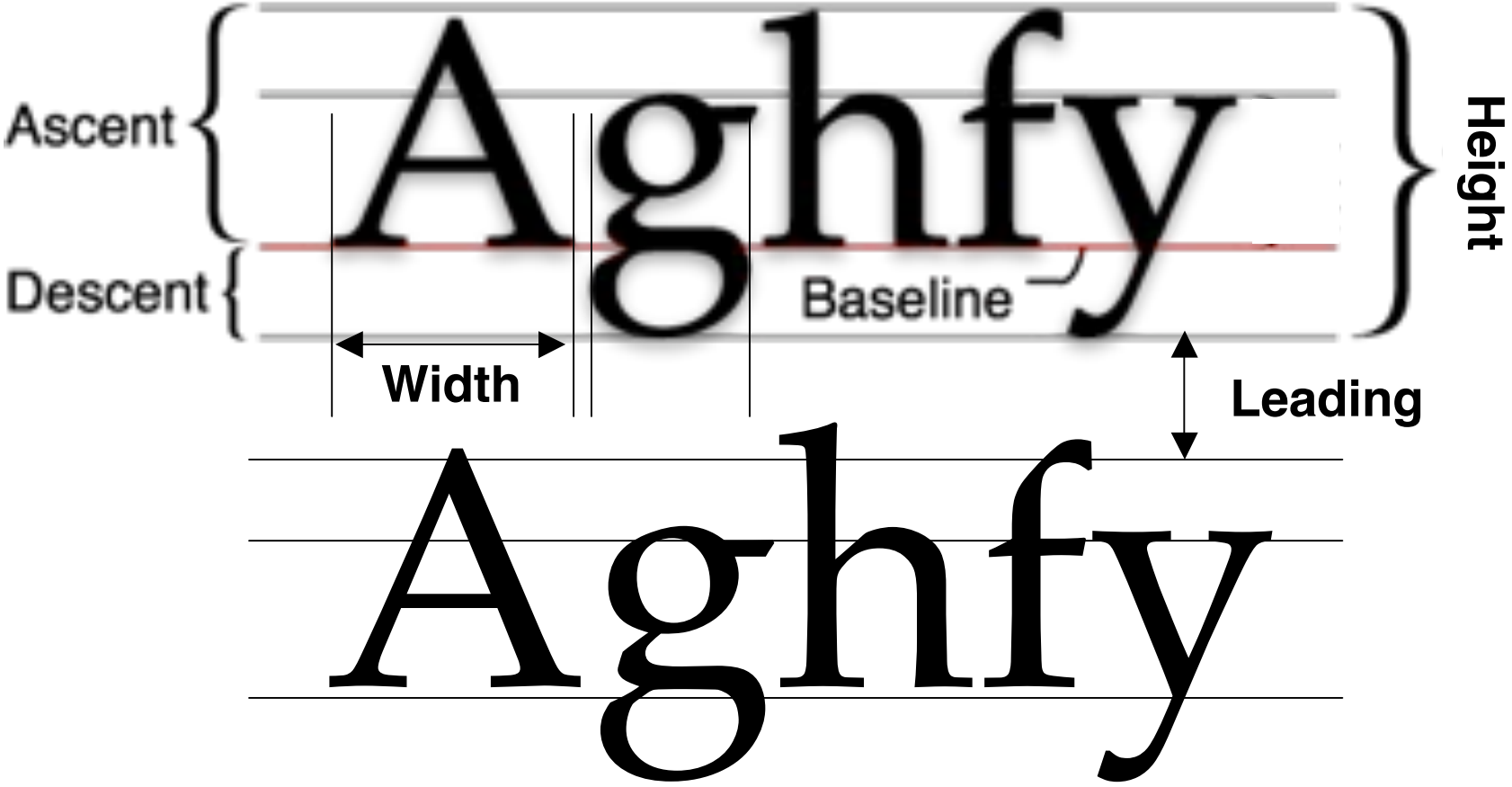
- Font as a whole has a standard ascent and standard descent



  - AWT has separate notion of Max ascent and descent, but these are usually the same

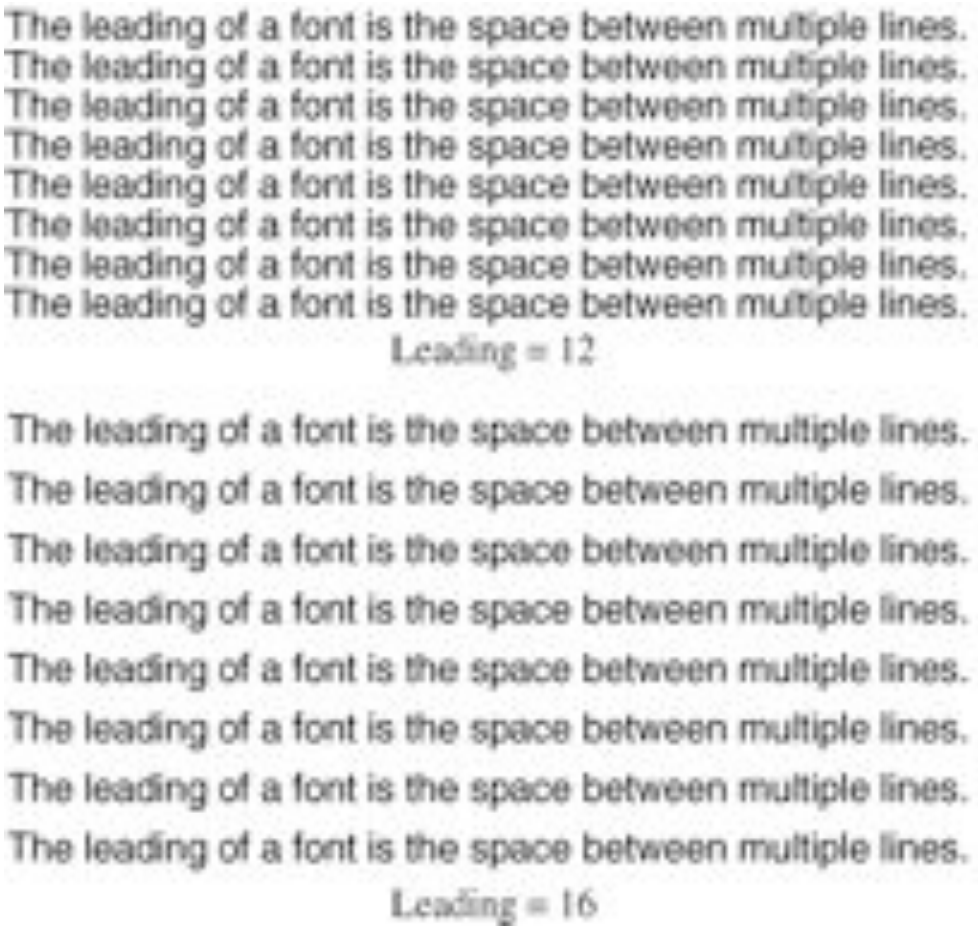
Height

- Height of character or font
  - ascent + descent + leading



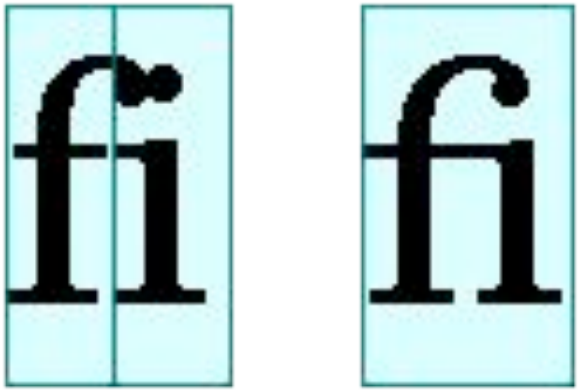
Leading

- Leading = space between lines of text
  - Pronounce “led”-ing after lead strips that used to provide it
  - Space between bottom of standard descent and top of standard ascent
    - i.e. interline spacing



Ligatures

- Merging two glyphs together



FontMetrics

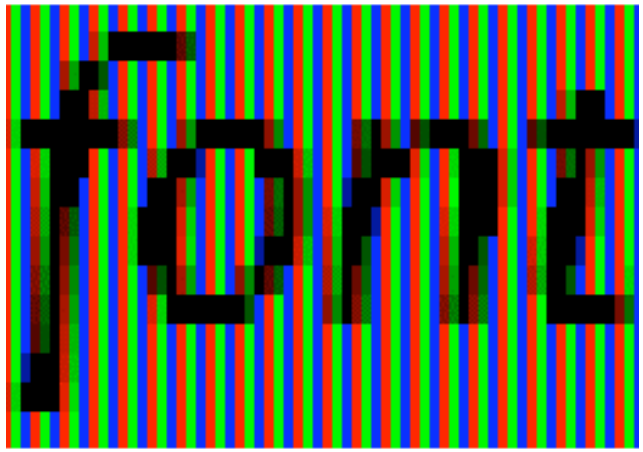
- Objects that allow you to measure characters, strings, and properties of whole fonts
  - See `java.awt.FontMetrics`
- `FontMetrics` objects give you all of above measurements
  - for chars & Strings
  - also char and byte arrays
  - for whole fonts
- In Java, `Graphics.getFontMetrics(f)` method gives `FontMetrics` for a given font

Aside: Microsoft's ClearType

- Subpixel rendering for fonts on LCD screens
  - Relies on how LCD RGB is arranged

The popularity of laptops shows that people are eager to use mobile technology. Windows XP Professional is designed to make mobile computing easier. New features for mobile computing will help you accomplish as much on the road or at home as you do in the office, so you can be productive no matter where you are.

Black and White



The popularity of laptops shows that people are eager to use mobile technology. Windows XP Professional is designed to make mobile computing easier. New features for mobile computing will help you accomplish as much on the road or at home as you do in the office, so you can be productive no matter where you are.

ClearType

Aside: Screen Fonts vs Print Fonts

- Some fonts designed for printing
  - Times New Roman, Helvetica
- Some fonts designed for screen
  - Verdana, Arial, Comic Sans, Trebuchet

- See FontBlog for more info on fonts
  - http://blogs.msdn.com/fontblog/

Anti-Aliasing

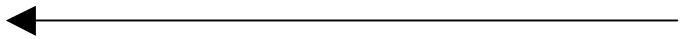
- Making edges appear smooth by using blended colors
- Useful for text (but not too small) as well as lines, etc.
- Supported by Java via `RenderingHints` parameter to `Graphics2D` object
- Can get to `Graphics2D` object in method `paintComponent(Graphics)`

Clipping

- Can also limit the effective area of drawing
  - Any pixels outside “clip area” are left unchanged
  - Like stencils in crafts
- May be limited set of shapes
  - Historically a single rectangle
- Java2D, etc. now support arbitrary shape clipping
  - Interesting drawing effects

  - Much more expensive than a single rect

Outline

- Low-level graphical output models
  - CRTs, LCDs, and displays
  - Colors
  - Raster operations
  - Lines, curves
  - Fonts
  - Affine Transforms 

Coordinate Transformations

- Linear ("affine") transformation
  - Translate, Scale, Rotate, Shear, plus any combination
- Can modify any shape, including text
- To fully understand, need matrix algebra:
  - Affine transformations are based on two-dimensional matrices of the following form:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & c & t_x \\ b & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \text{where} \quad \begin{aligned} x' &= ax + cy + t_x \\ y' &= bx + dy + t_y \end{aligned}$$

- See `java.awt.geom.AffineTransform`

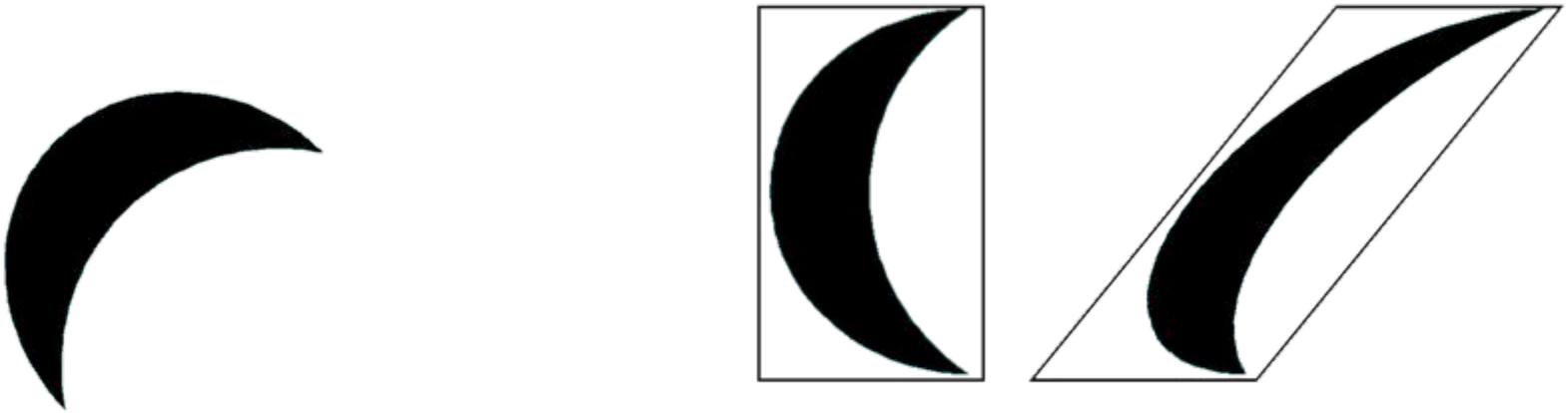
Translate

- Move with respect to origin
- Equivalent to changing the coordinate system
  - After translate(10,50) new origin is where (10,50) used to be
  - Used to implement hierarchical coordinates (child object gets own origin)

Scale

- Not necessarily uniform
- Get flip by negative scale

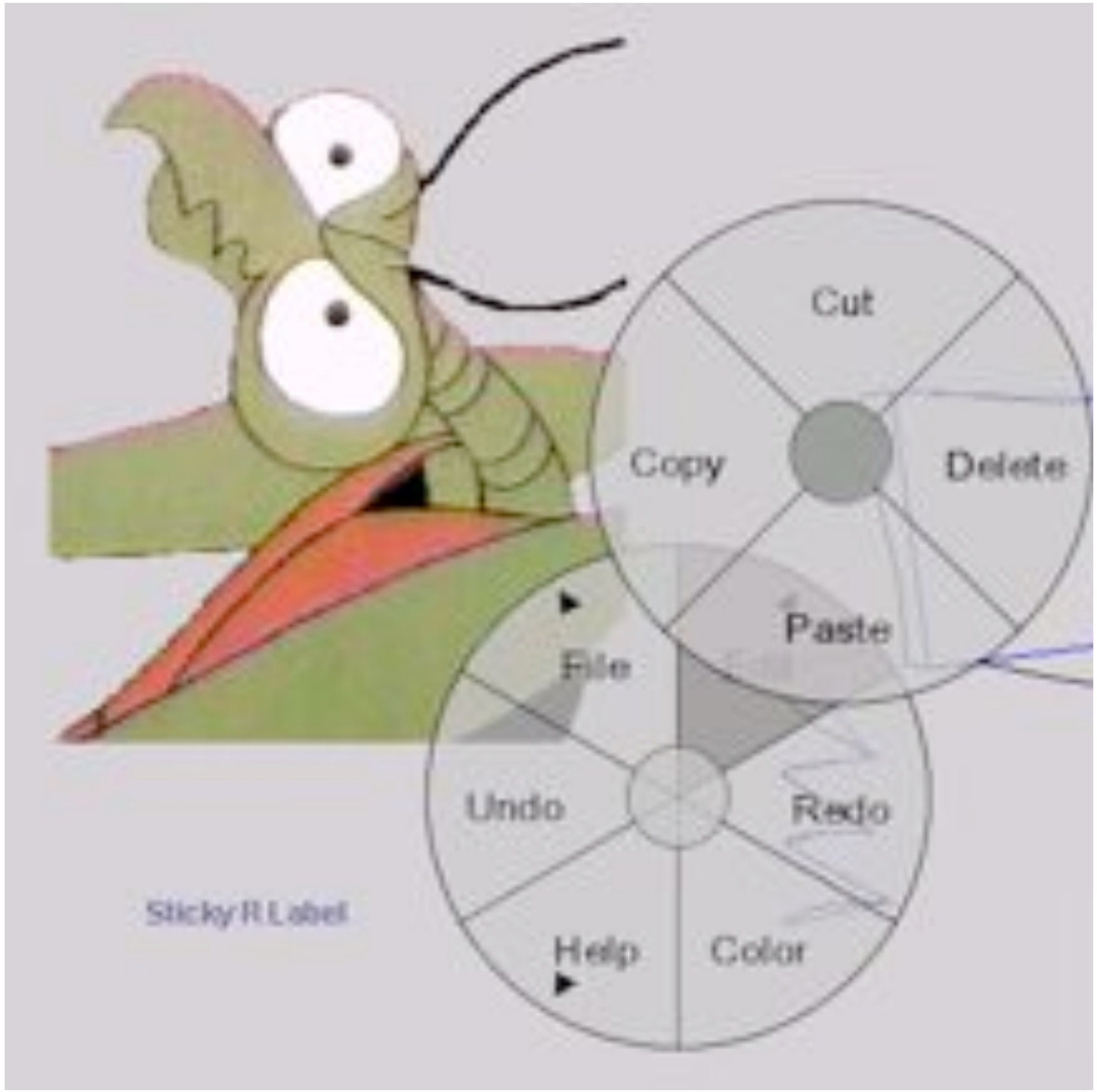
Rotate and Shear



- Used much less in UI work
  - Note that axis no longer aligned

Rotations and Alpha in Java

- Trivial in Java



Why are Affine Transforms Useful?

Rotating UIs

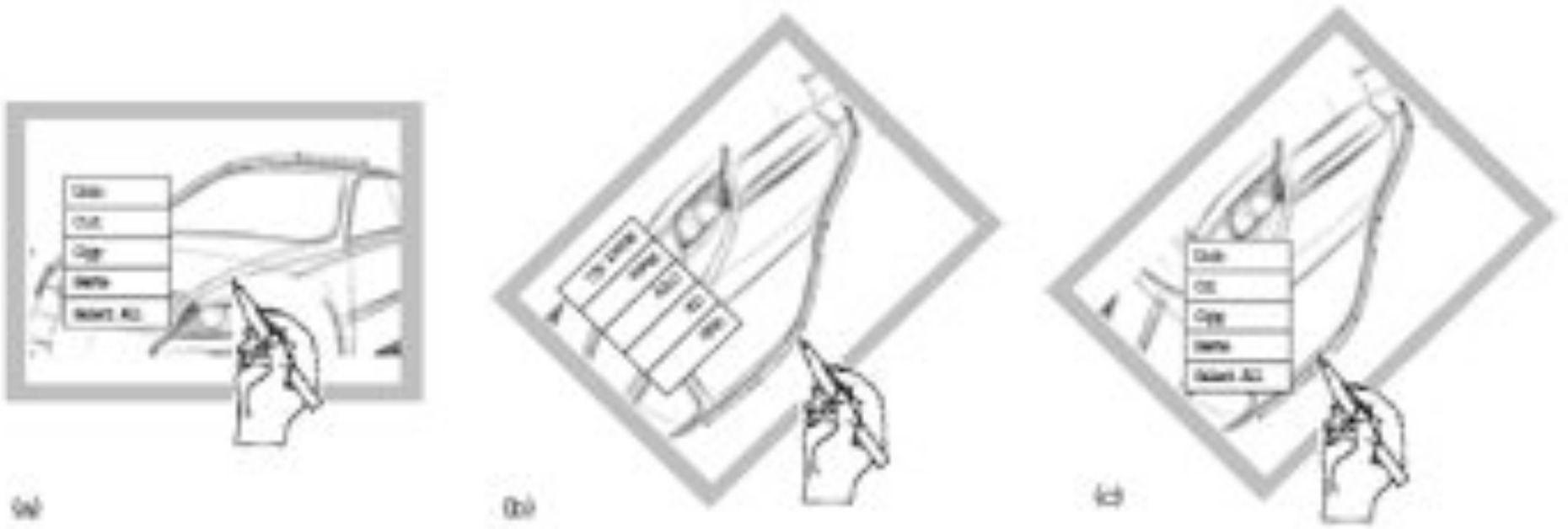
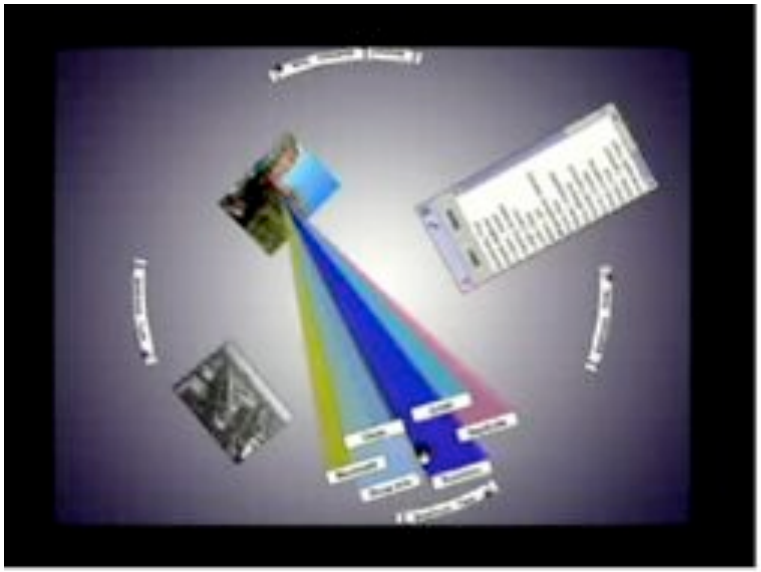
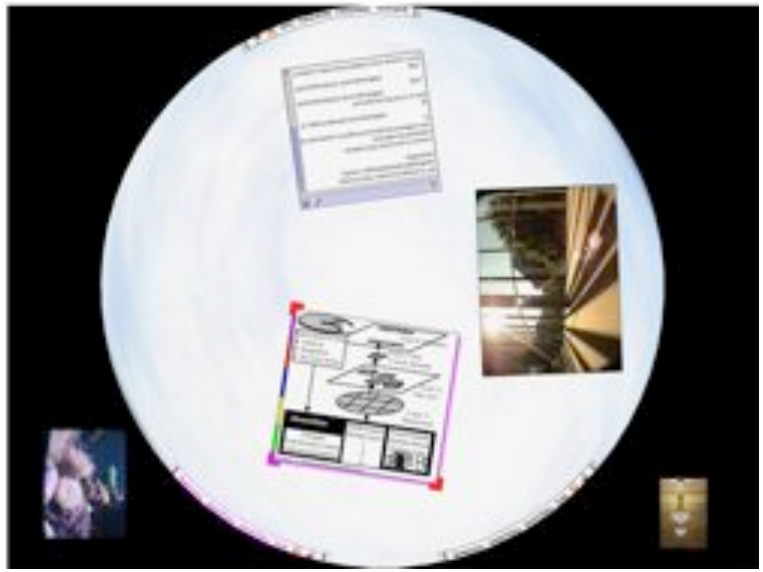


Figure 9: The concept of a rotating UI. a) shows a pen-based computer at normal orientation. Rotating the artwork (b) may give a drawing advantage but the UI becomes hard to use. (c) shows how rotating the UI relative to the user solves the problem.

Why are Affine Transforms Useful?

Rotating UIs





Why are Affine Transforms Useful?

Semantic Zooming UIs

Graphics Context Objects

- Same object often also provides access to drawing operations
  - Java: `Graphics` or `Graphics2D` object
  - Maintains graphical "state"
    - Color, font, transformation, clipping etc.
    - Also gives access to drawing
      - `drawLine()`, `fillRect()`, `drawString()`, etc.
  - Important: two Graphics objects may draw on the same part of the screen (but have different settings, e.g., current color)
  - Rendering Hints go here too

- See `paintComponent(Graphics2D)`

Summary

- Low-level graphical output models
  - CRTs, LCDs, and displays
  - Colors
  - Raster operations
  - Lines, curves
  - Fonts
  - Affine Transforms
- Homework assignment:
  - Read Sun's Java2D tutorial
  - http://java.sun.com/docs/books/tutorial/2d/index.html
  - No summary needed
- P2 out soon