Output Concepts

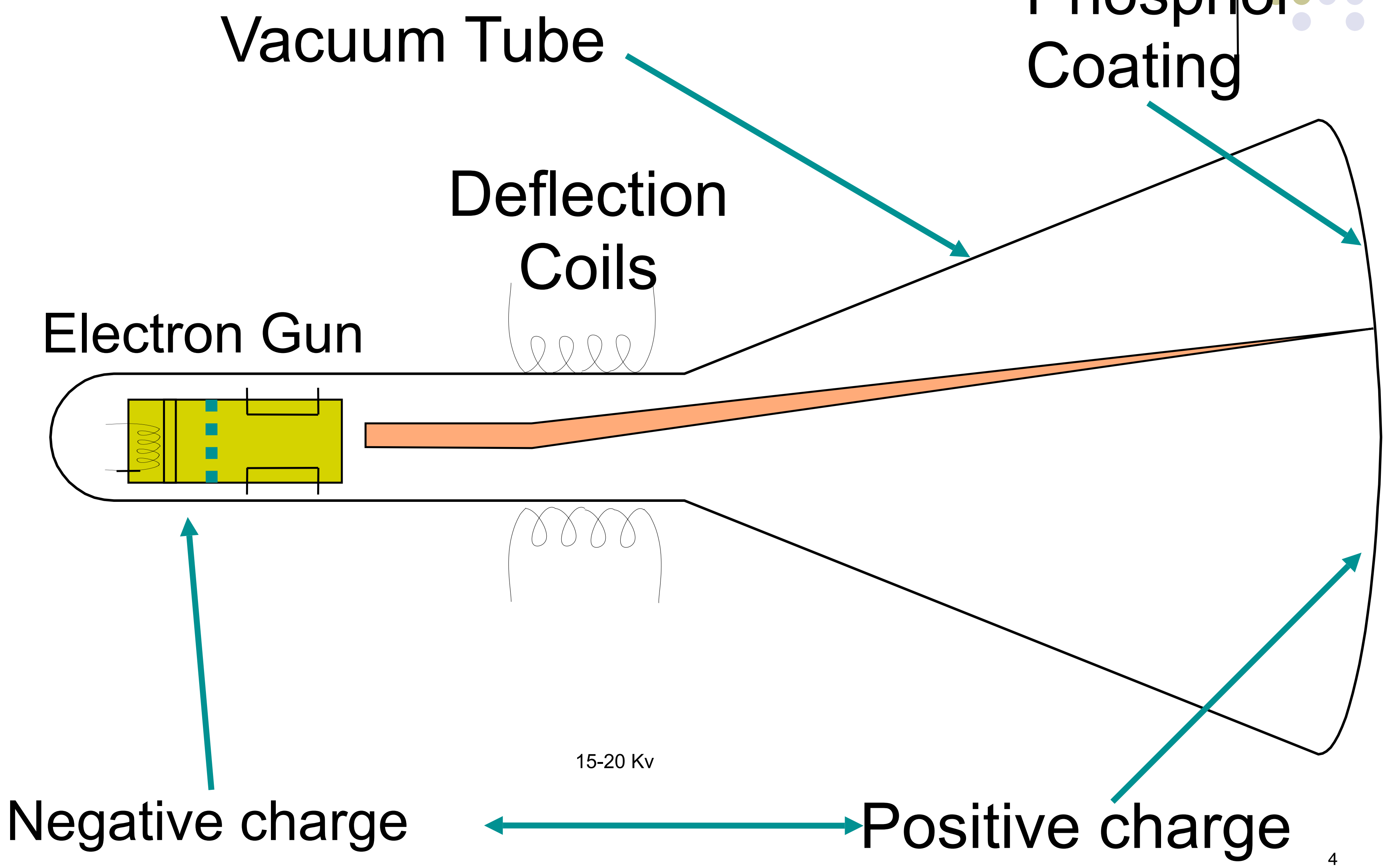
Start with some basics: display devices

- Just how do we get images onto a screen?
- Most prevalent device: CRT
  - Cathode Ray Tube
  - AKA TV tube

Cathode Ray Tubes

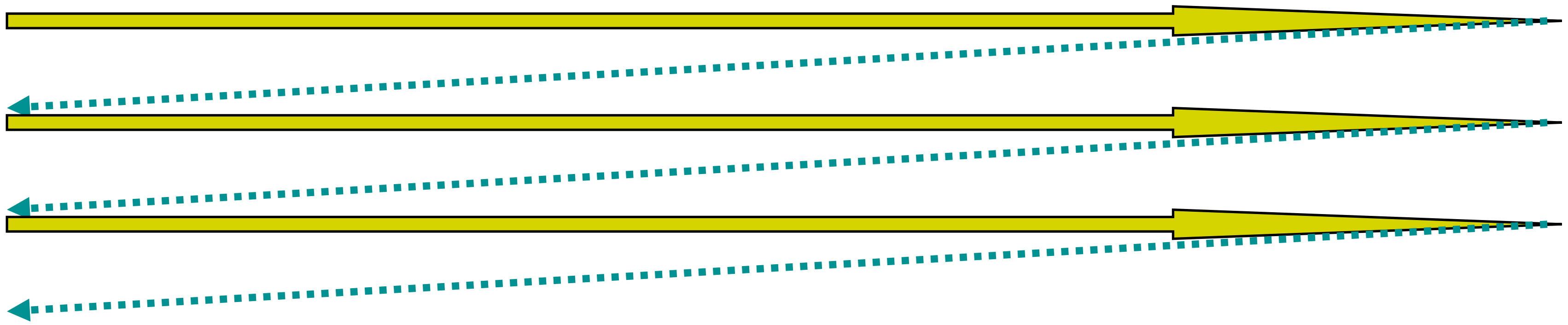
- Cutting edge 1930's technology
  - (basic device actually 100 yrs old)
  - Vacuum tube (big, power hog, …)
  - Refined some, but no fundamental changes
- But still dominant
  - Because TVs are consumer item
  - LCD's just starting to challenge

How a CRT works (B/W)



Move electron beam in fixed scanning pattern

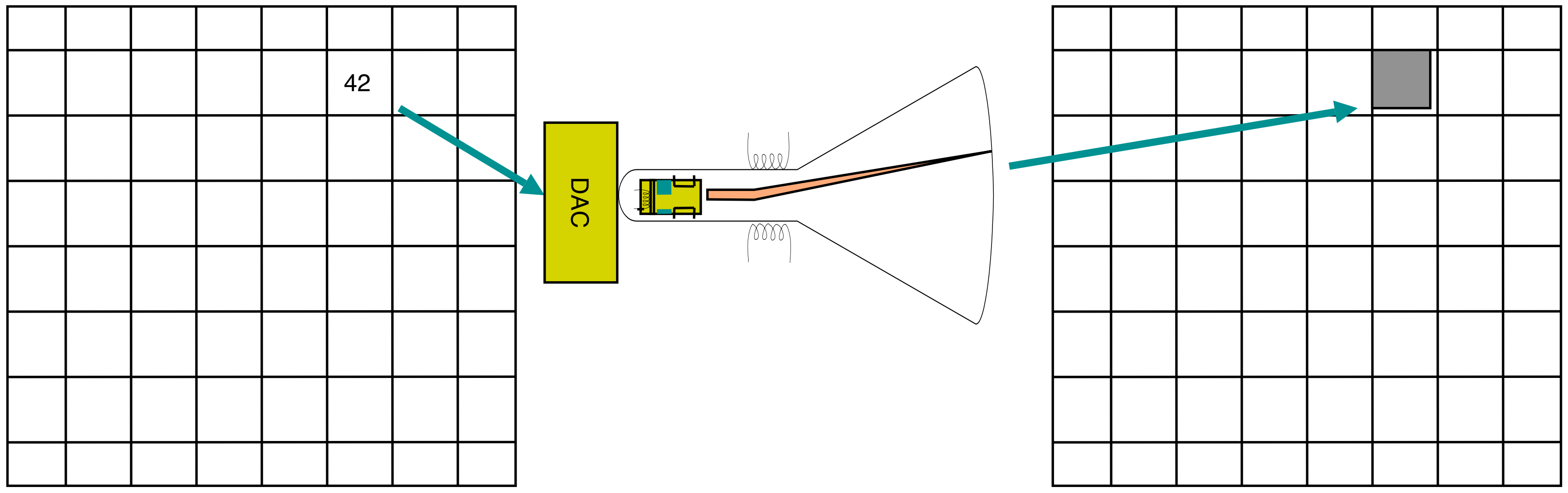
- “Raster” lines across screen



- Modulate intensity along line (in spots) to get pixels

Pixels determined by 2D array of intensity values in memory

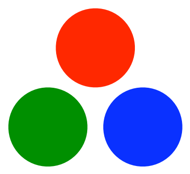
- "Frame buffer"
  - Each memory cell controls 1 pixel



  - All drawing by placing values in memory

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
  - Use "shadow mask" behind phosphor spots to help



Color frame buffer

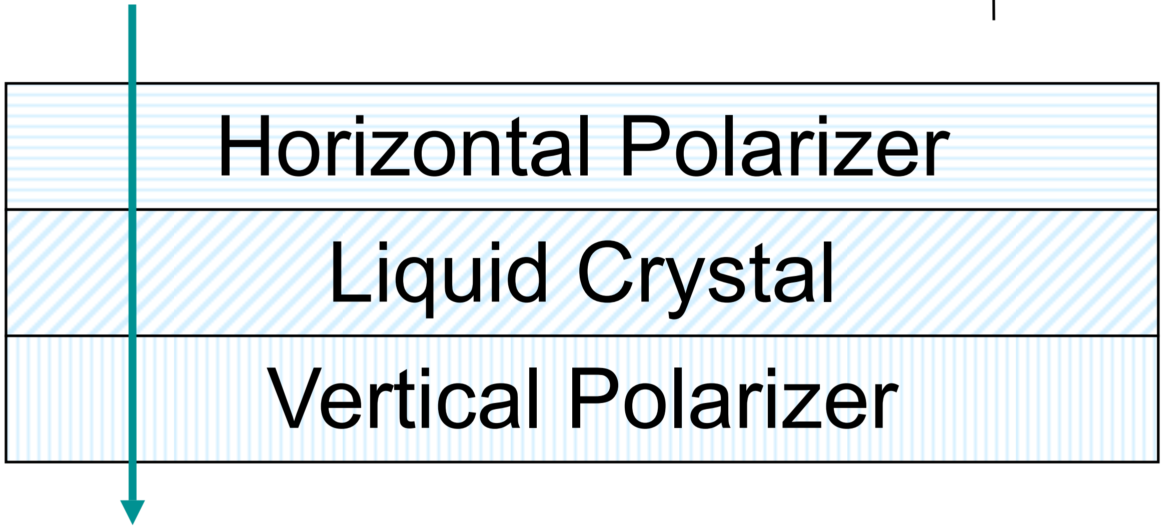
- Frame buffer now has 3 values for each pixel
    - each value drives one electron gun
    - can only see ~ 2^8 gradations of intensity for each of R,G,&B
    - 1 byte ea => 24 bits/pixel => full color

Other display technologies: LCD

- Liquid Crystal Display
- Discovered in 1888 (!) by Reinitzer
- Uses material with unusual physical properties: liquid crystal
    - rest state: rotates polarized light 90°
    - voltage applied: passes as is

Layered display

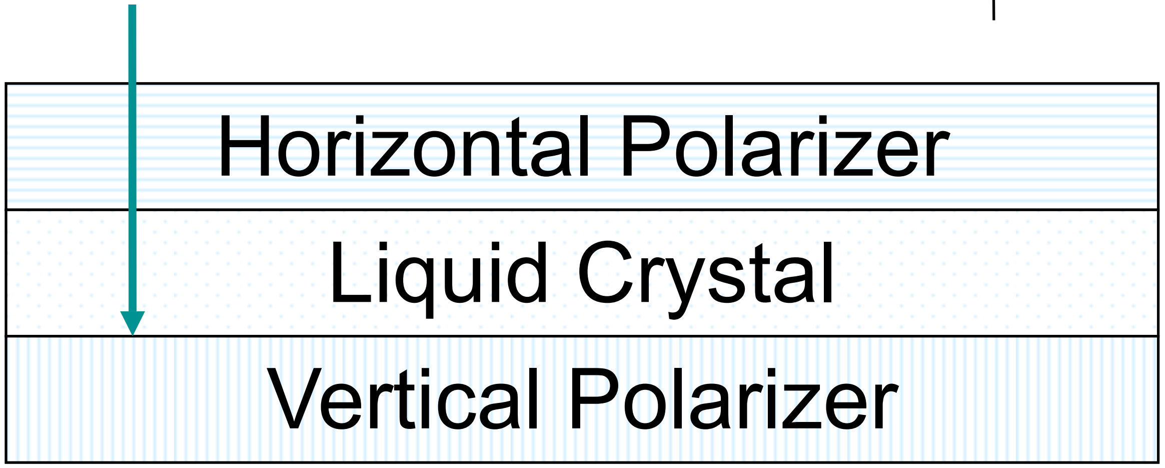
- Layers



- In rest state: light gets through
  - Horizontally polarized, LC flips 90°, becomes vertically polarized
  - Passes through

Layered display

- Layers



- In powered state: light stopped
  - Horizontally polarized, LC does nothing, stopped by vertical filter

Lots of other interesting/cool technologies

- 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

All these systems use a frame buffer

- Again, each pixel has 3 values
  - Red, Green Blue
- Why R, G, B?
  - R, G, and B are particular freq of light
  - Actual light is a mix of lots of frequencies
  - Why is just these 3 enough?

Why R, G, & B are enough

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

- 8-bit of each does a pretty good job, but… some complications

Complications

- Eye's perception is not linear (logarithmic)
- CRT's (etc.) do not respond linearly
- Different displays have different responses
  - different dynamic ranges
  - different color between devices!
- Need to compensate for all of this

Gamma correction

- Response of all parts understood (or just measured)
- Correct: uniform perceived color
  - Normally table driven
    - 0…255 in (linear intensity scale)
    - 0…N out to drive guns
      - N=1024 or 2048 typical

Unfortunately, gamma correction not always done

- E.g., TV is not gamma corrected

➡ Knowing RGB values does not tell you what color you will get!

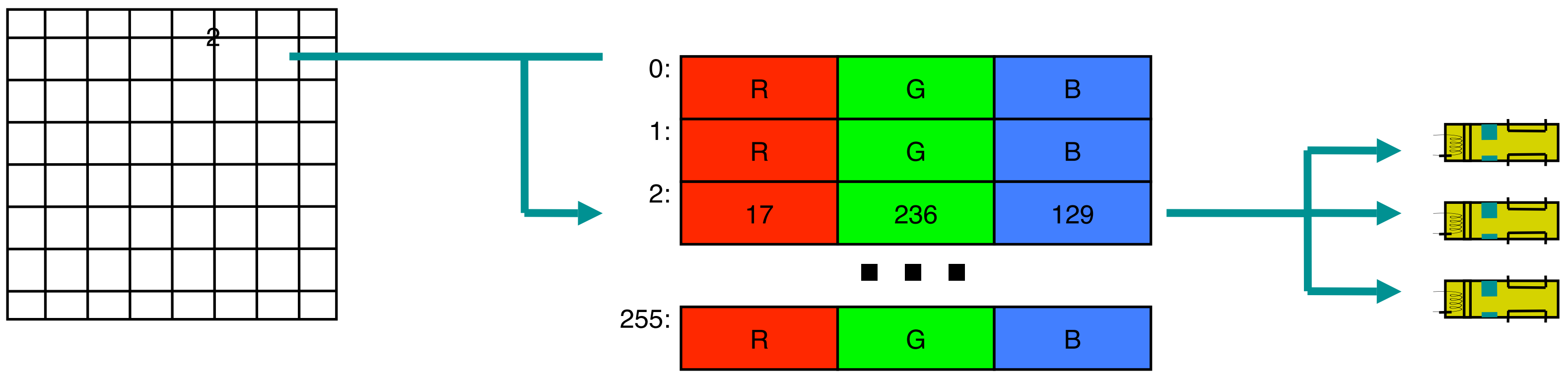
- For systems you control: do gamma correction

24 bits/pixel => "true color," but what if we have less?

- 16 bits/pixel
  - 5 each in RGB with 1 left over
  - decent range (32 gradations each)
- Unfortunately often only get 8
  - 3 bits for GB, 2 for R
  - not enough
  - Use a "trick" instead

Color lookup tables (CLUTs)

- Extra piece of hardware
  - Use value in FB as index into 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 CLUT
  - Gives "palette" of 256 different colors
  - Chosen from 16M
  - Can do a lot better than uniform by picking a good palette for the image to be displayed (nice algorithms for doing this)

Software models of output (Imaging models)

- Start out by abstracting the HW
- Earliest imaging models abstracted early hardware: vector refresh
  - stroke or vector (line only) models

Vector models

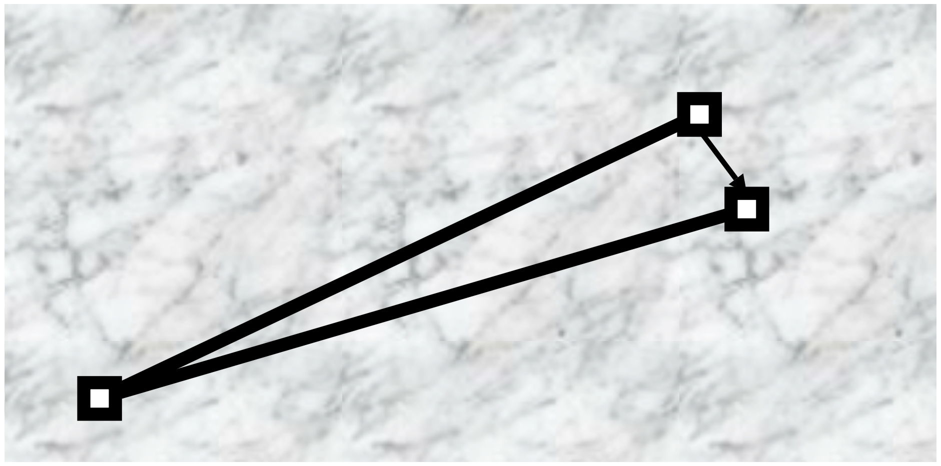
- Advantages
    - can freely apply mathematical xforms
        - Scale rotate, translate
        - Only have to manipulate endpoints
- Disadvantages
    - limited / low fidelity images
        - wireframe, no solids, no shading

Current dominant: Raster models

- Most systems provide model pretty close to raster display HW
  - integer coordinate system
  - 0,0 typically at top-left with Y down
  - all drawing primitives done by filling in pixel color values (values in FB)

Issue: Dynamics

- Suppose we want to "rubber-band" a line over complex background



- Drawing line is relatively easy
- But how do we "undraw" it?

Undrawing things in raster model

- Ideas?

(red, su, xo, pal, fwd)

Undrawing things in raster models

- Four solutions:
- 1) Redraw method
    - Redraw all the stuff under
    - Then redraw the line
- Relatively expensive (but HW is fast)
    - Note: don't have to redraw all, just "damaged" area
- Simplest and most robust (back)

How to undraw

- 2) “Save-unders”
  - When you draw the line, remember what pixel values were “under” it
  - To undraw, put back old values
  - Issue: (what is it?)

How to undraw

- 2) “Save-unders”
  - When you draw the line, remember what pixel values were “under” it
  - To undraw, put back old values
  - Issue: what if “background” changes
- Tends to either be complex or not robust
  - Typically used only in special cases

(back)

How to undraw

- 3) Use bit manipulation of colors
  - Colors stored as bits
  - Instead of replacing bits XOR with what is already there
    - A ^ B ^ B == ?

How to undraw

- 3) Use bit manipulation of colors
  - Colors stored as bits
  - Instead of replacing bits XOR with what is already there
    - A ^ B ^ B == A (for any A and B)
  - Draw line by XOR with some color
  - Undraw line by XOR with same color

Issue with XOR?

- What is it?

Issue with XOR

- Colors unpredictable
  - SomeColor ^ Blue == ??
    - Don’t know what color you will get
    - Not assured of good contrast
      - Ways to pick 2nd color to maximize contrast, but still get “wild” colors

Undraw with XOR

- Advantage of XOR undraw
  - Fast
  - Don't have to worry about what is "under" the drawing, just draw
- In the past used a lot where dynamics needed
  - May not be justified on current HW

(back)

How to undraw

- 4) Simulate independent bit-planes using CLUT "tricks"
  - Won't consider details, but can use tricks with CLUT to simulate set of transparent layers
  - Probably don't want to use this solution, but sometimes used for special cases like cursors (back)

Higher level imaging models

- Simple pixel/raster model is somewhat impoverished
  - Integer coordinate system
  - No rotation (or good scaling)
  - Not very device independent

Higher level imaging models

- Would like:
  - Real valued coordinate system
    - oriented as Descarte intended?
  - Support for full transformations
    - real scale and rotate
  - Richer primitives
    - curves

Stencil and paint model

- All drawing modeled as placing paint on a surface through a “stencil”
  - Stencil modeled as closed curves (e.g., splines)
- Issue: how do we draw lines?

Stencil and paint model

- All drawing modeled as placing paint on a surface through a "stencil"
  - Modeled as closed curves (splines)
- Issue: how do we draw lines?
  - (Conceptually) very thin stencil along direction of line
  - Actually special case & use line alg.

Stencil and paint model

- Original model used only opaque paint
  - Modeled hardcopy devices this was developed for (at Xerox PARC)
- Current systems now support "paint" that combines with "paint" already under it
  - e.g., translucent paint ("alpha" values)

Stencil and paint model(s)

- Postscript model is based on this approach
  - Dominant model for hardcopy, but not screen
- New Java drawing model (Java2D) also takes this approach
- Mac OS X
  - derived from NeXTstep, which used Display Postscript
- Windows Vista?

Stencil and paint model(s)

- Advantages
    - Resolution & device independent
        - does best job possible on avail HW
        - Don’t need to know size of pixels
    - Can support full transformations
        - rotate & scale

Stencil and paint model(s)

- Disadvantages
  - Slower
    - Less and less of an issue
    - But interactive response tends to be dominated by redraw time
  - Much harder to implement

Stencil and paint model(s)

- Stencil and paint type models generally the way to go
    - But have been slow to catch on
        - Market forces tend to keep us with old models
        - Much harder to implement
    - But starting to see these models for screen based stuff (esp. w/ Java2D)

Object-oriented abstractions for drawing

- Most modern systems provide uniform access to all graphical output capabilities / devices
  - Treated as abstract drawing surface
    - “Canvas” abstraction
    - subArctic: drawable
    - Macintosh: grafPort
    - Windows: device context
    - X Windows: GC (GraphicsContext)
    - Java: Graphics/Graphics2D classes

Object-oriented abstractions for drawing

- Abstraction provides set of drawing primitives
  - Might be drawing on…
    - Window, direct to screen, in-memory bitmap, printer, …
  - Key point is that you can write code that doesn't have to know which one

Object-oriented abstractions for drawing

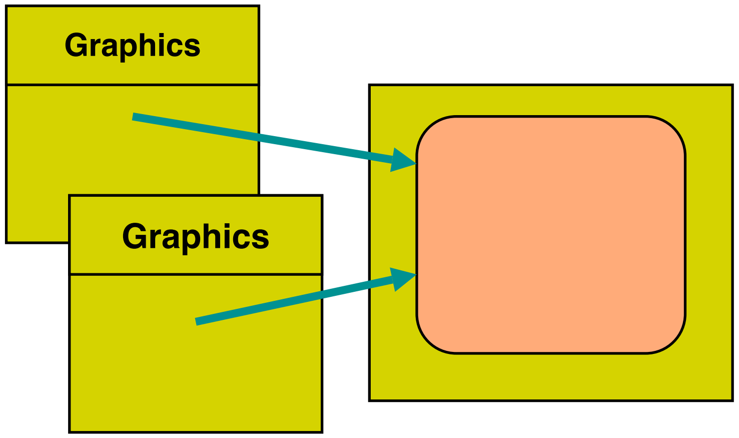
- Generally don't want to depend on details of device but sometimes need some:
  - How big is it
  - Is it resizable
  - Color depth (e.g., B/W vs. full color)
  - Pixel resolution (for fine details only)

A particular drawing abstraction: java.awt.Graphics

- Fairly typical raster-oriented model
- More recent version: Graphics2D

java.awt.Graphics

- Gives indirect access to drawing surface / device
  - Contains
    - Reference to screen
    - Drawing "state"
      - Current clipping, color, font, etc.



  - Multiple graphics instances may reference the same drawing surface (but hold different state information)

Fonts and drawing strings

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

- More recent formats (OpenType) can specify *lots* more
  - E.g., ligatures, alternates

ﬀ aﬀect

ﬃ aﬃne

ﬄ aﬄict

Fonts

- Typically specified by:
  - A family or typeface
    - e.g., courier, helvetica, times roman
  - A size (normally in “points”)
  - A style
    - e.g., plain, italic, bold, bold & italic
    - other possibles (from mac): underline, outline, shadow
- See java.awt.Font

Points

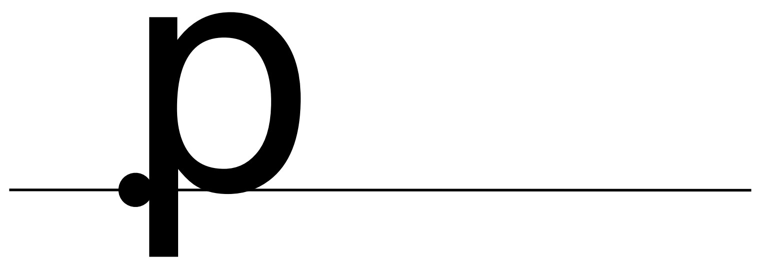
- An odd and archaic unit of measurement
  - 72.27 points per inch
    - Origin: 72 per French inch (!)
  - Postscript rounded to 72/inch most have followed
  - Early Macintosh: point==pixel (1/75th)

FontMetrics

- Objects that allow you to measure characters, strings, and properties of whole fonts
- java.awt.FontMetrics
- Get it by using:
    - Graphics.getFontMetrics()

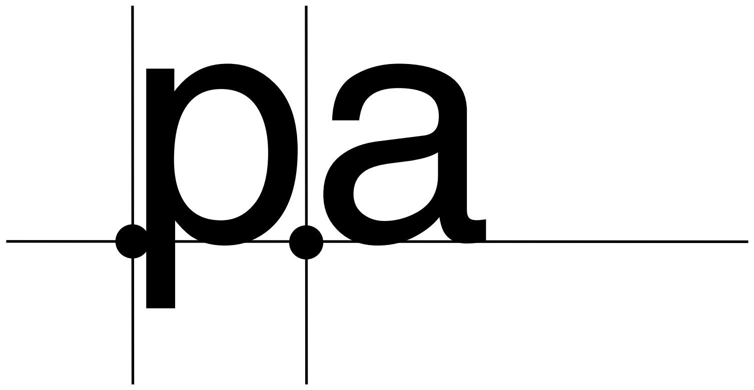
Reference point and baseline

- 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

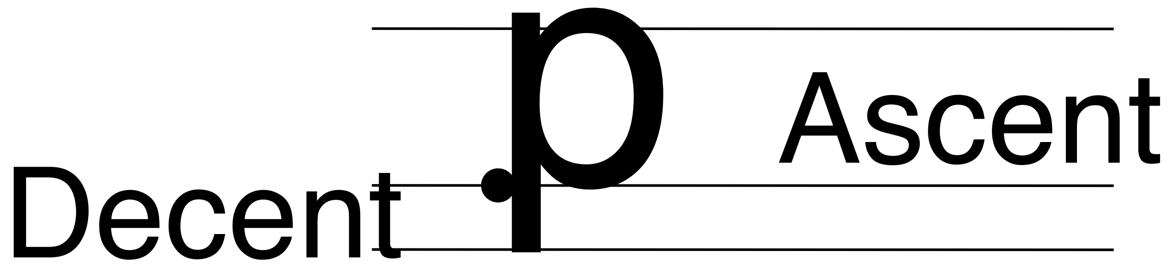


Widths

- Each character also has a bounding box width
  - May be different from advance width in some cases
  - Don’t get this with AWT FontMetrics, so there “width” means “advance width”

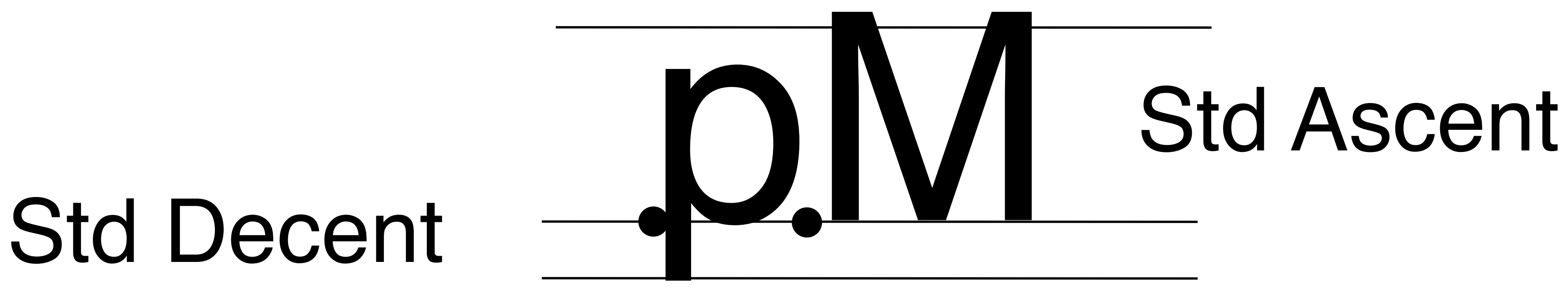
Ascent and decent

- Glyphs are drawn both above and below baseline
  - Distance below: "decent" of glyph
  - Distance above: "ascent" of glyph



Standard ascent and decent

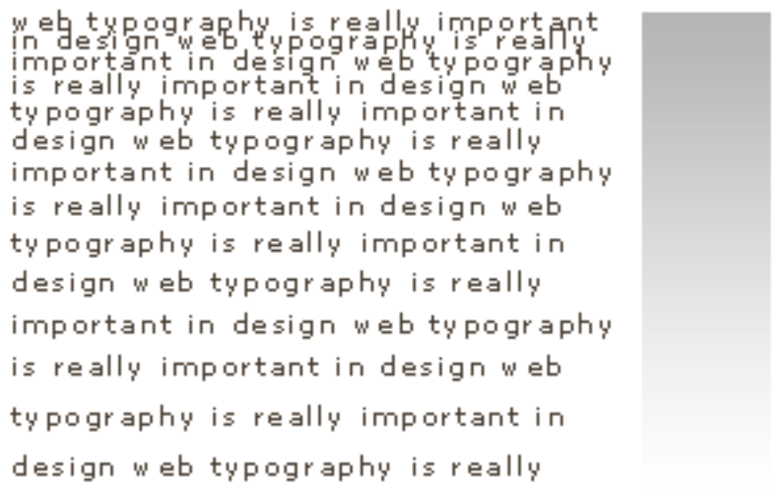
- Font as a whole has a standard ascent and standard decent



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

Leading

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



w eb typography is really important
in design w eb typography is really
important in design w eb typography
is really important in design w eb
typography is really important in
design w eb typography is really
important in design w eb typography
is really important in design w eb
typography is really important in
design w eb typography is really
important in design w eb typography
is really important in design w eb
typography is really important in
design w eb typography is really

Height

- Height of character or font
  - ascent + decent + leading
  - not standard across systems: on some systems doesn't include leading (but does in AWT)

FontMetrics

- FontMetrics objects give you all of above measurements
  - for chars & Strings
  - also char and byte arrays
  - for whole fonts
- Graphics method will get you FontMetrics for a given font