Output: Interaction Styles and Graphics

Overview

- Output styles / interaction techniques
- Output hardware
- Raster operations
- Fonts

Bad Day

Reminder

- **Don’t let your users have a bad day**
- **Don’t let yourselves have a bad day**

Input Devices

- QWERTY keyboard (other types)
- Mouse (1, 2, or 3 buttons)
- Other pointing devices:
  - Pens or pucks on tablets
  - Light pens on screens
  - DataGloves, eye tracking, etc.
  - Stylus on PDA
- Speech input
- Computer connected camera
  - Presence
  - Free-space gestures (Wii)
  - Eye-tracking
- Sensors

Output Devices

- Older
  - TTY on paper
  - 24x80 terminals “glass-TTY”
  - Vector screens
- Raster-scan screens
  - Color, monochrome
- LCD panels
- Tiny, room-size, portables, normal-size
- 3-D devices
  - Head-mounted displays
  - Stereo
- Speech output
- Non-speech audio

Examples of different types of applications

- Word processors
- Drawing programs: CAD/CAM
- Hierarchy displays (file browsers)
- Mail readers
- Spreadsheets
- Hypertext document reading
  - Web browsing
  - Form-based entry
- ATMs
- Virtual Reality
- Multimedia: video/animation
- Interactive, real-time games
- Controlling machinery

Diversity, diversity, diversity

Huge numbers of input devices

X

Huge numbers of output devices

X

Huge numbers of applications

= large numbers of ways to interact with systems

Metaphors

- **Content Metaphors**
  - **Desktop**
  - **Paper document**
  - **Notebook with tabs**
  - **Score sheet, stage with actors**
  - **Accounting ledger**
  - **Stereo (media players)**
  - **Phone keypad**
  - **Calculator**
  - **Web: shopping carts**
  - **Quicken: Checkbook**
- **Interaction Metaphors**
  - **Tools, agents: electronic secretary**

User Interface Styles

- Method for getting information from the user or interfacing with a user
- Some styles:
  1. Question and answer
  2. Single character commands and/or function keys
  3. Command language
  4. Menus
  5. Forms/dialogue boxes
  6. Direct manipulation
  7. WYSIWYG
     - Really sub-class of DM, not another style
  8. Gestures
  9. Natural language

Interaction Styles

- Usually, interfaces provide more than one style
  - Command language for experts with menus for novices
  - Menus plus single characters
- Appropriate style depends on type of user and task
- Important issues
  - Who has control?
  - Ease of use for novices
  - Learning time to become proficient
  - Speed of use (efficiency) once proficient
  - Generality/flexibility/power (how much of user interface does technique cover?)
  - Ability to show default, current values, etc.
  - Skill requirements (e.g. typing)

1) Question and Answer

- Nielsen describes 1, 2, and 3 as line-oriented
- Computer asks questions, user answers
- Used by some simple programs and also expert systems
- Wizards in many products
- Telephone interfaces: press 1 for sales, 2 for support, …
- Pros and cons
  + easy to implement (writeln, readln)
  + easy for novices
  - can't correct previous errors or change your mind (Wizards often have previous button)
  - can be slower for experts

2) Single Character Commands and/or Function Keys

- **Function keys can be labeled**

- **Pros and cons**
  - **+ fastest method for experts**
  - **+ easy to learn how to do things**
  - **+ so easier to provide telephone support (just hit the F1 key now)**
  - **+ usually very simple to implement**
  - **- hardest to remember which key does what**
  - **- easy to hit wrong key by mistake**

3) Command Language

- **User types instructions to computer in formal language**

- **Pros**
  - **+ most flexible**
  - **+ supports user initiative**
  - **+ fast for experts**
  - **+ possible to provide programming language capabilities for macros, customization, etc.**
  - **+ takes less space on screen**

3) Command Language

- **Cons**
  - **hardest for novices**
  - **requires substantial training and memorization**
  - **error rates usually high**
  - **syntax usually very strict**
  - **poor error handling**
  - **hard for user to tell what they can do**

- **Implementation difficulty depends on availability of tools like LEX and YACC and the complexity of the language**

4) Menus

- **Pros**
  - **+ very little training needed**
  - **+ shows available options**
  - **+ allows use of recognition memory (easier than generation)**
  - **+ hierarchy can expand selection**
  - **+ default or current selection can be shown**
  - **+ ability to show when parts are not relevant (e.g. greyed out)**
  - **+can be used for commands and arguments**
  - **+ reduces keystrokes (compared to command languages)**
  - **+ clear structure to decision making**

4) Menus

- Cons
  - usable only if there are a few choices
  - slow for experienced users (need accelerators)
  - if big hierarchy, commands can be hard to find
  - uses screen space
- Most effective with pointing devices
- Zoomable, adaptive, ...

5) Form Filling

- Like menus, except have text/ number fields that can be filled in
- Often used on character terminals (e.g. for data entry)
- E.g. Mac/Windows dialog boxes
- Most effective with pointing devices
- Most user interfaces are of this form

5) Form Filling

- **Pros and cons (similar to menus)**
  - **+ simplifies data entry**
  - **+ very little training needed**
  - **+ shows available options**
  - **+ allows use of recognition memory (easier than generation)**
  - **+ ability to show defaults and current values**
  - **+ ability to show when parts are not relevant (e.g. greyed out)**
  - **- consumes screen space**
  - **- expensive to internationalize**

6) Direct Manipulation

- [WIMP (Windows, Icons, Menus, Pointing Devices) Interfaces include 6 and 7]
- Definition
  - Visual model of the world
  - Visual objects that can be operated on
  - Results of actions are reflected in the objects immediately
  - Objects, once operated on, can be further operated on
- Term coined by Ben Shneiderman
- Original system: Sketchpad from 1962
- "Object-oriented" from user's point of view
  - As opposed to "function-oriented"
  - Usually select object, then give command

6) Direct Manipulation

- **Pros and Cons**
  - **+ user initiated**
  - **+ easy to learn, intuitive, analogical**
  - **+ fast to use for objects that are on the display**
  - **+ easily augmented with menus and forms**
  - **+ provides closure of actions and gestures**
  - **+ errors can be avoided**
  - **+ high subjective satisfaction (fun)**
  - **– can be inconvenient and slow if user knows the name of an undisplayed object, but must find it anyway**
  - **– limited power; not all desired actions have DM analog**
  - **– difficult to provide macros, other user extensible/ customizable features**
  - **– difficult to implement**

7) WYSIWYG

- "What you see is what you get"
- Like direct manipulation, but more so
- Pros and cons: similar to DM
  - + can always tell what final result will be
  - + reflects the state of the object
  - - screen image may be hard to read/interpret, especially if screen resolution is too low
  - - cannot show hidden structure (how picture was made)
  - - May be very slow at run-time (e.g. page breaks)
  - - Extremely difficult to implement
  - - WYSIATI: what you see is all there is: lack of structure; no ability to show structure

"Non-command" or "Next-generation" interfaces

- "Natural" actions invoke computer response
- 8) gestures, 9) speech, and 10) natural behavior
- Issues: mis-interpretation, feedback

8) Gestures

- Like user would mark on paper
- With a pen, stylus or watched by camera
- Pros and cons:
  - + can be very natural to learn
  - + often faster to execute than other techniques
  - + give command and parameters together
  - - many gestures are hard to do with a mouse
  - - users must memorize gestures: no affordances

9) Natural Language

- **E.g. a subset of normal English**
- **Includes speech**
- **Pros and cons:**
  - **+ theoretically easiest for learning**
  - **+ speaking is the fastest input technique**
  - **- rather slow for typing**
  - **- requires clarification dialog**
  - **- unpredictable**
  - **- general systems are impossible with today's technology**
- **Research shows that if you factor in correction times, speech input may be *slower* and *less natural* than typing, etc.**

10. Natural Behavior

- No direct input to computer system: intention
- Location-based services
- Context-based services
- Pros and cons:
  - + fast and can be very useful, if correct interpretations made
  - +pro-active, user doesn't have to do anything out of ordinary
  - - hard to infer user intent, needs
  - - unpredictable and difficult to control
  - - feedback difficult

Apple's Knowledge Navigator

- 1987, vision of the future
- Many different interaction styles
- Think about prototyping with lots of different input and output devices

Prototyping These Visions

- Useful?
- How many styles of input/ output?
- How do you prototype?

- **Break 15 minutes**

Basic graphics: display devices

- How do we get stuff on a screen?

Stroke models

- Describe an image as strokes (possibly with color & thickness)
  - Line ((10,4), (17,4), thick 2, red)
  - Circle ((19,13), radius 3, blue)
- Maps to early vector displays and plotters
- Essentially all window systems and toolkits support this kind of drawing (due to early CG roots)

Problems with pure stroke drawing models?

Problems with pure stroke drawing models?

- How do you draw this?

Modern display devices are typically *pixelated*

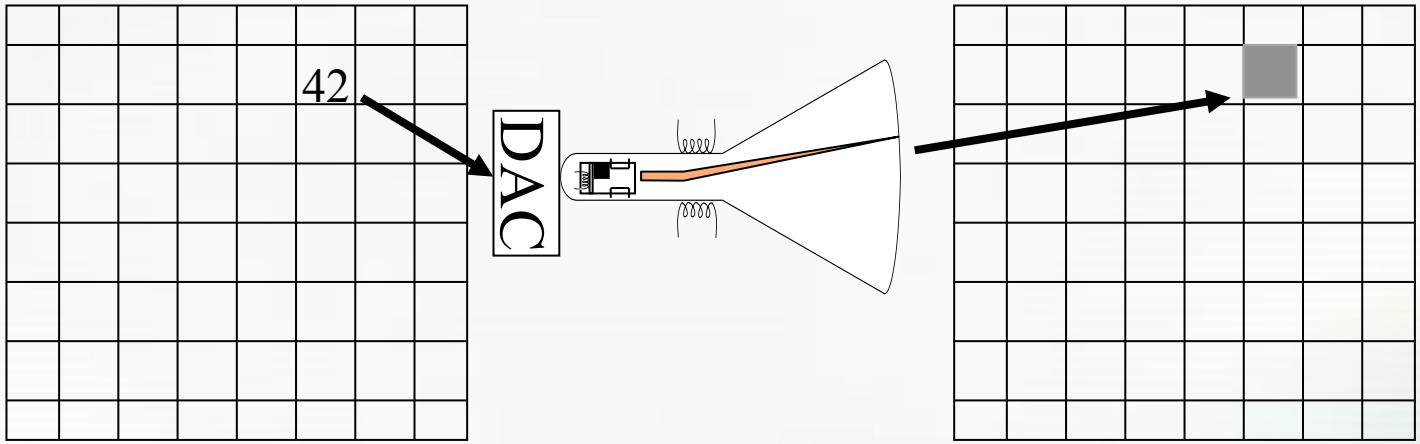
- Break display into a set of discrete picture elements (pixels) and store color for each
  - Typically small squares
  - Image depth: number of bits per pixel

- Until recently most prevalent device: Cathode Ray Tube (CRT)
- Now common: Liquid Crystal Display (LCD)
- Also: Plasma, direct retinal display, …

- All work from a "Frame Buffer"

Frame buffers

- All of these devices work from a "Frame buffer"
  - A piece of memory which holds values for the colors of pixels
  - Each memory cell controls 1 pixel



  - All drawing by placing values in memory

More on frame Buffers

- Each pixel actually 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-bits of each does a pretty good job
  - "Full color" → 3x8 = 24 bits

Other color models

- **CMY – Cyan, Magenta, Yellow**
  - **Subtractive primaries**
  - **Colors indicating what is removed from white rather than added to black (no light) as in RGB**
  - **For pigments rather than light emitters → printing**
  - **Pigment gets color from light it absorbs and doesn't reflect**
  - **Used by printers and artists**

Other color models

- **HSV** **Hue (primary wavelength)**
  Saturation (purity of light)
  Value (brightness)
  - **User-oriented, intuitive appear of artist's hint, shade, tone**
  - **Closer to people's intuitions of what color is**

- **Note interpolation between colors in different models gives different intermediate results**

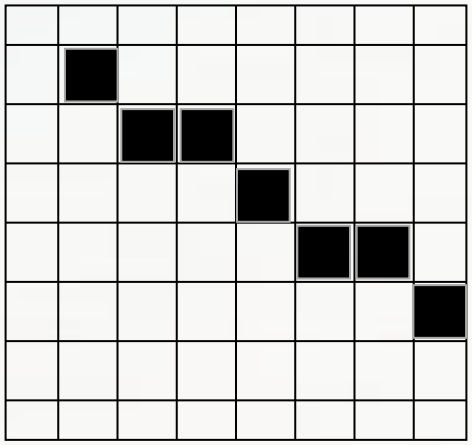
Aspects of pixelated displays

- Resolution
  - How many pixels on the display
    - E.g. 1280x1024
  - Also physical size of pixels
    - Pixels per inch (or dots per inch: DPI)
- Color depth
  - Bitmap (1 bit B/W)
  - Gray scale (2-8 bits monochrome)
  - Color mapped (typically 8 bits)
    - Mapped through a lookup table
    - At most 256 different colors, but you can pick which 256
  - Full Color (3x8 = 24 bits)

Issues with pixelated displays

How do you draw this:

with this?:



Resulting roughness is *Aliasing*

Solution: Anti-Aliasing

- Making edges appear smooth by using blended colors
  - Pixel is not just "off" or "on"
- Useful for text as well as lines, etc.

Region-based models

- Use the stroke model to define the outline (infinitely thin) of a region
  - Think of it as a stencil
- Fill the region with
  - Colors, patterns, blendings
  - Think of it as paint within the stencil
- Postscript (or PDF) are primary examples

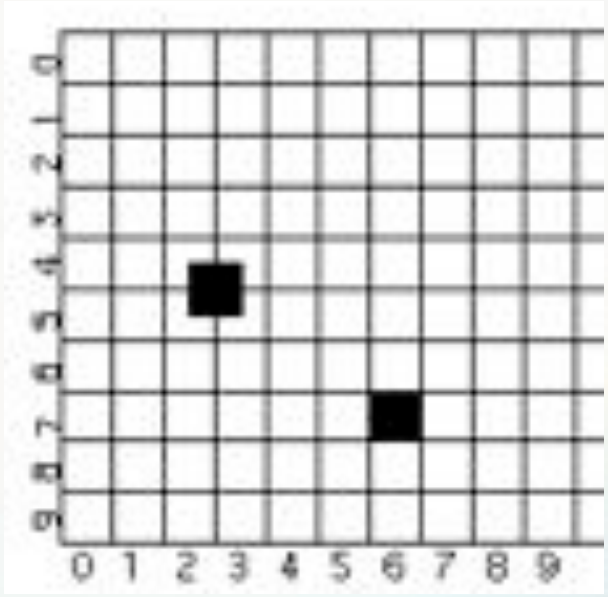
- Advantage
  - Unified model (including text)
  - Independent of pixel size
  - Lower levels of system can automatically adapt to device
    - Can be slower, but modern GPUs have plenty of speed

Back to pixelated or "raster-oriented" models

- Typically pretty close to display HW
  - Integer coordinate system
  - 0,0 typically at top-left with Y down
    - From text operations & raster scan
  - All drawing primitives equivalent to filling in pixel color values in frame buffer

Coordinate Systems for Drawing

- 0,0 in top left: different from conventional axes
- Coordinates of pixels:
  - Center of pixel?
  - Corner of pixel?
- Matters for lines



Most Primitive Raster Operation: Copy Values

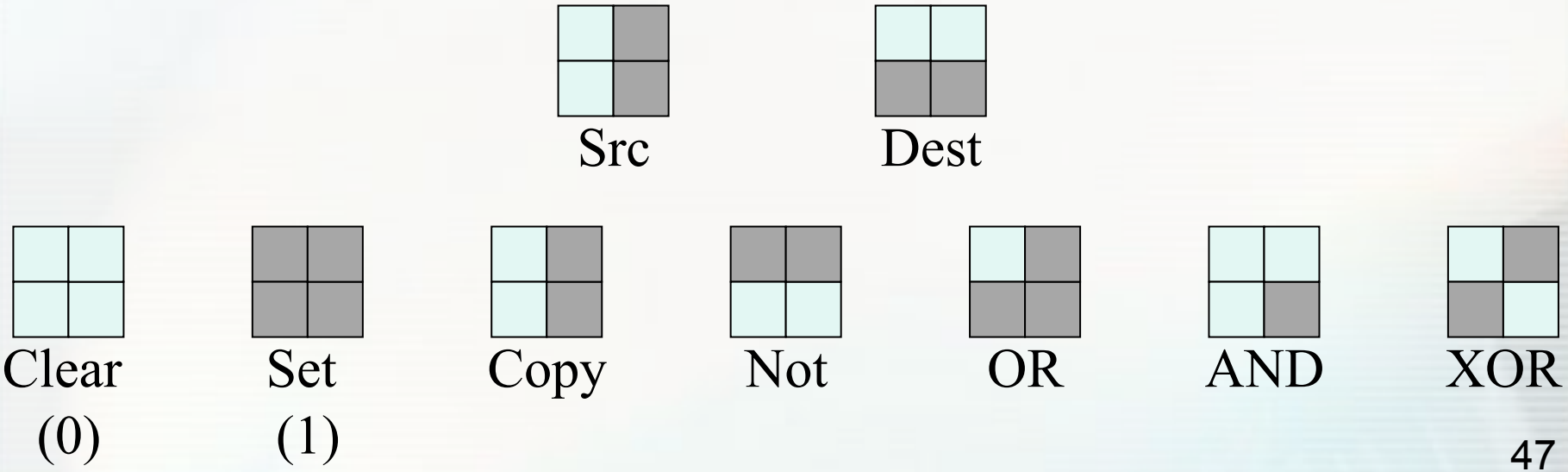
- Copy an area of the screen

```
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, combine pixels with values already there

- RasterOP (BitBlt)
  - First used for B/W only (1 bit color)
  - Boolean combination operators



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 (for any A and B)

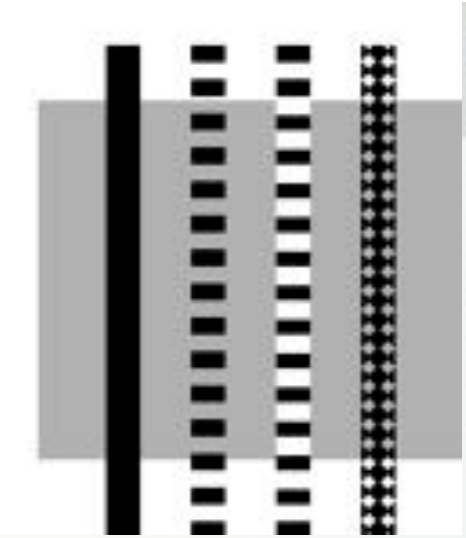
RasterOp Continued

- Doesn't work as well in color
  - Well defined (operates on bits)
  - But: Blue ^ Violet == ??
- Other combination operators make more sense for color
  - Transparency
    - weighted average of colors
  - "Alpha" values (RBGA) determine how much of source is "mixed" with existing destination colors

Drawing Primitives

- Don't want to do everything based on memory cells
- 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"

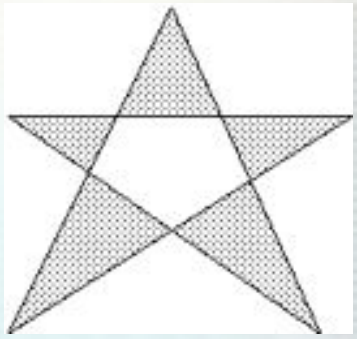
Line Properties



- Width
- Line styles
  - Solid, dashed 1110001110001110000, "double-dashed", patterned
- Cap-style: butt, round, projecting (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
  - which parts?

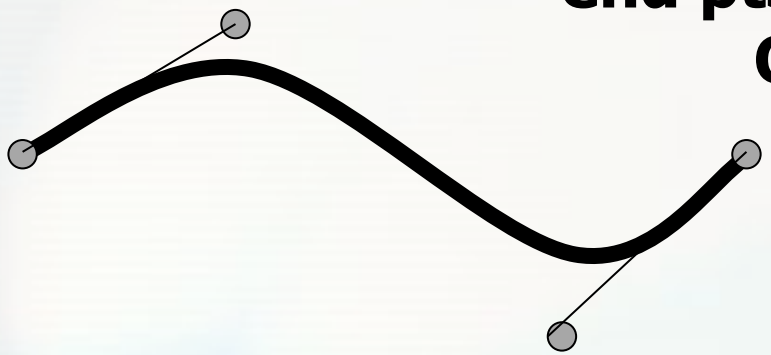


Curves (Splines)

- **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., FvDF& H)**

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., FvDF& H)**
- **Bézier defined by "control" points**



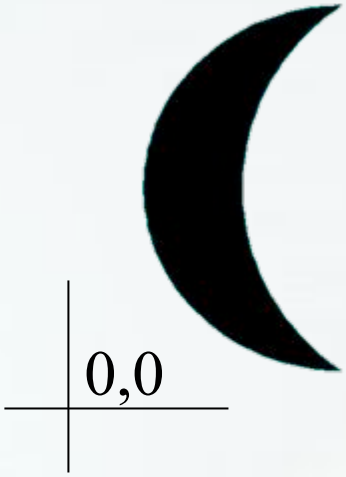
Path or region models (again)

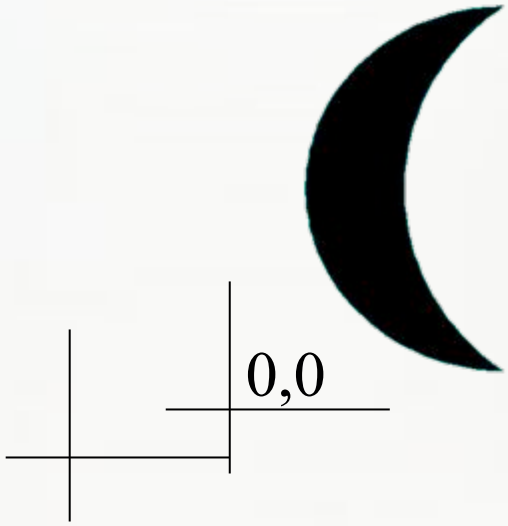
- Instead of drawing via fixed shapes (rectangle, etc.)
- Path model unifies:
  - Define a path first
    - General ops: moveTo, lineTo's, curveTo (etc.)
  - Then draw it
    - Stroke or fill
    - With various properties of line & fill

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
- Many systems now support arbitrary shape
  - Interesting drawing effects
  - Much more expensive than a single rectangle (but we can afford it on modern systems)

Transformations of primitives
Translate





- **Move with respect to origin**

Scale

- **Not necessarily uniform**
- **Get flip by negative scale**

Rotate and Shear

- Used much less in UI work
  - Axis no longer aligned

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

Fonts

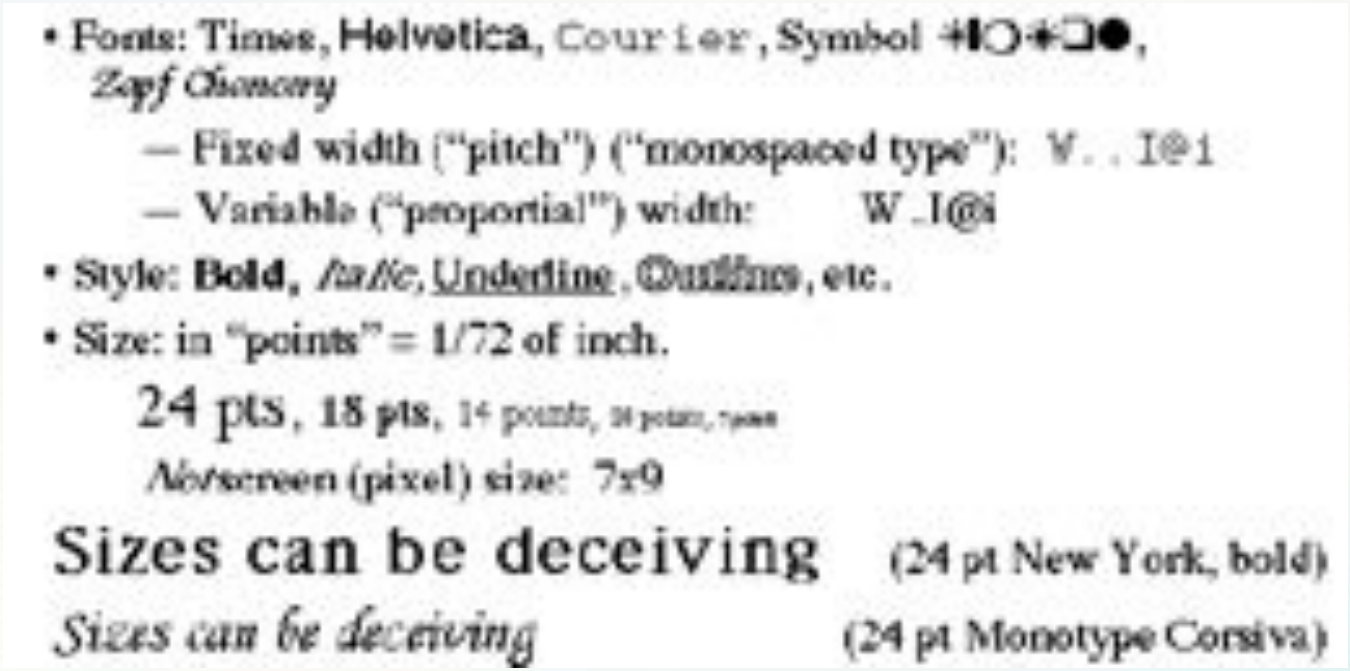
- Typically specified by:
  - A family or typeface
    - e.g., courier, helvetica, times roman
  - A size (normally in "points")
  - A style
    - e.g., plain, italic, bold, bold & italic
    - other possible styles: underline, strikethrough, outline, shadow

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 (1/75th)

FontMetrics

- **Objects that allow you to measure characters, strings, and properties of whole fonts**



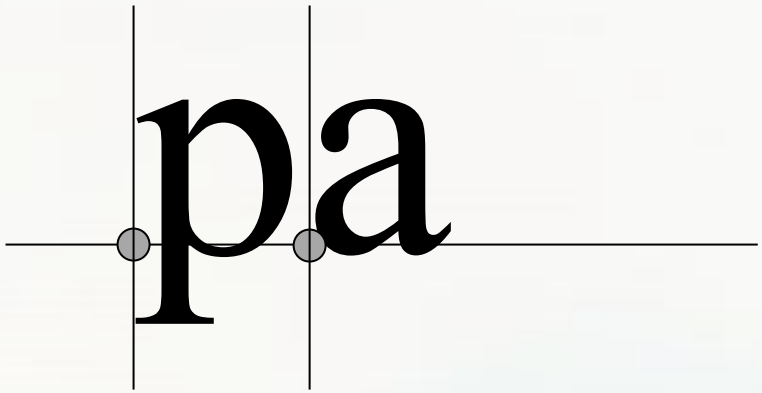
Reference point and baseline

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

  - Reference point defines a baseline

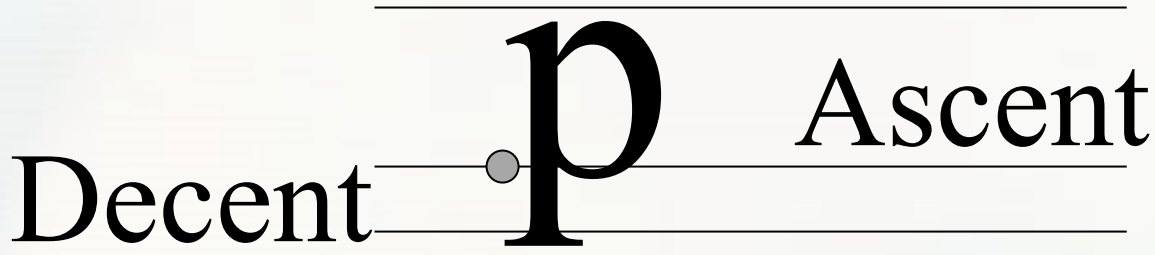
Advance width

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



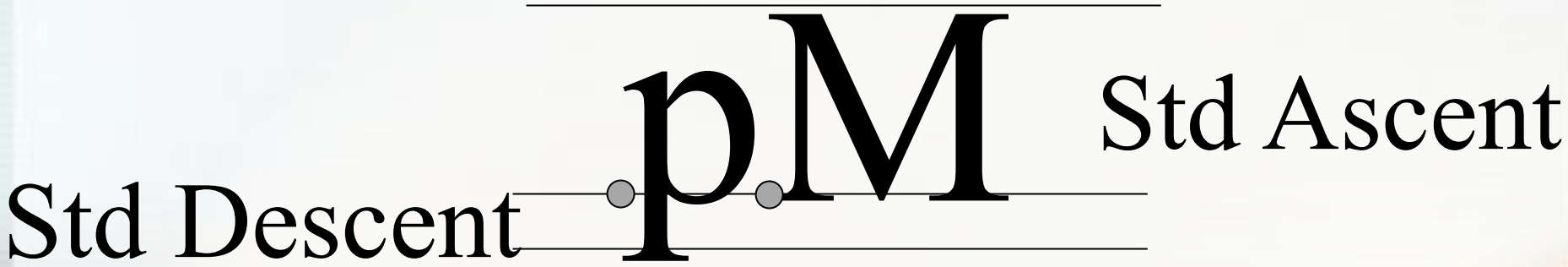
Ascent and decent

- **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**



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

Height

- Height of character or font
  - ascent + descent + leading
  - But not standard across systems: on some systems doesn't include leading

- **Questions?**
- **Admin…**

- **HW2b due today**
- **HW3 assigned today**

Questions?