2D GRAPHICS

CMPT 381

Overview

Colour models

Models for representing images

Coordinate systems

Abstract drawing capability

Node-based and Canvas-based Java Graphics

Redrawing and Clipping

Colour Models

RGB and HSB

Transparency

RGB

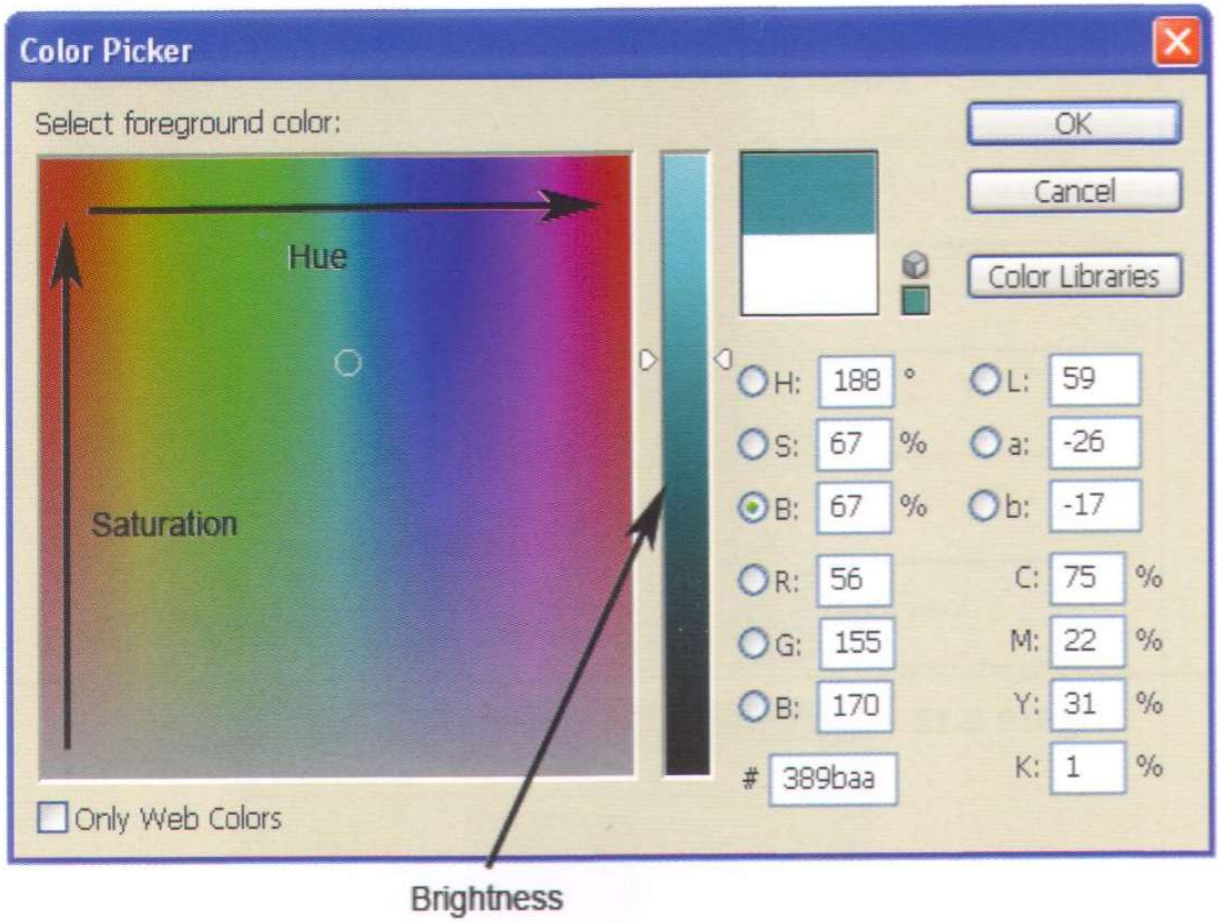
- specify color by Red, Green, & Blue components
- In 24-bit colour, 8 bits per primary
  - Only 6 bits per colour are needed
- In 32-bit colour, extra 8-bits for alpha
- Usually represented as a triple
  - 0-255: (155, 155, 46), #FFDD99
  - 0-1.0: (.5,.3,1.0)

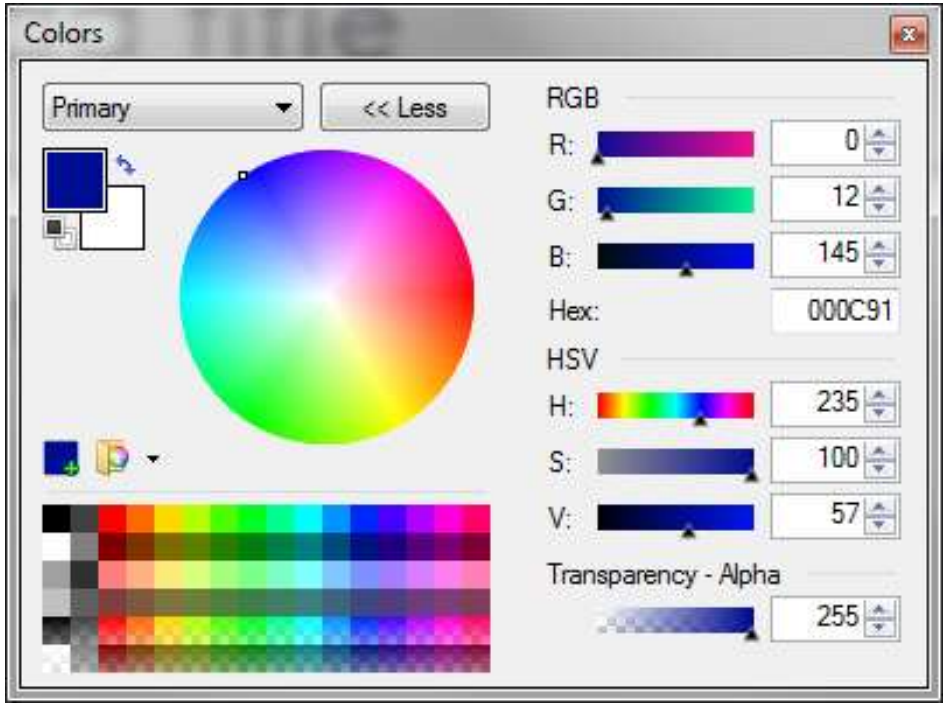
HSB/HSL/HSV/HSI

- Hue Saturation Brightness/Luminance/Value/Intensity
- HSV model: Hue, Saturation, Value
  - Hue: the primary wavelength
  - Saturation: a measure of purity
    - high is pure, low means mixed with grey or white
  - Value: intensity (dark vs. light)
  - HSV is easier for people to use
  - There is a direct conversion to RGB
  - H has 360 values (degrees), S 100 values (%), V 100 values(%)

Transparency

- Adds another byte to determine how much opacity a color has
  - How much light it lets through from behind
- Transparency is the fraction of light that should be allowed to show through
  - 0 shows nothing
  - 100% - completely see through
- Many systems use RGBA
  - Is the opacity (opposite to transparency value)


Color Picker
Select foreground color:
OK
Cancel
Color Libraries
Hue
Saturation
H: 188 °
S: 67 %
B: 67 %
R: 56
G: 155
B: 170
389baa
L: 59
a: -26
b: -17
C: 75 %
M: 22 %
Y: 31 %
K: 1 %
Only Web Colors
Brightness


Colors
Primary
<< Less
RGB
R: 0
G: 12
B: 145
Hex: 000C91
HSV
H: 235
S: 100
V: 57
Transparency - Alpha
255

Drawing Models

Pixels

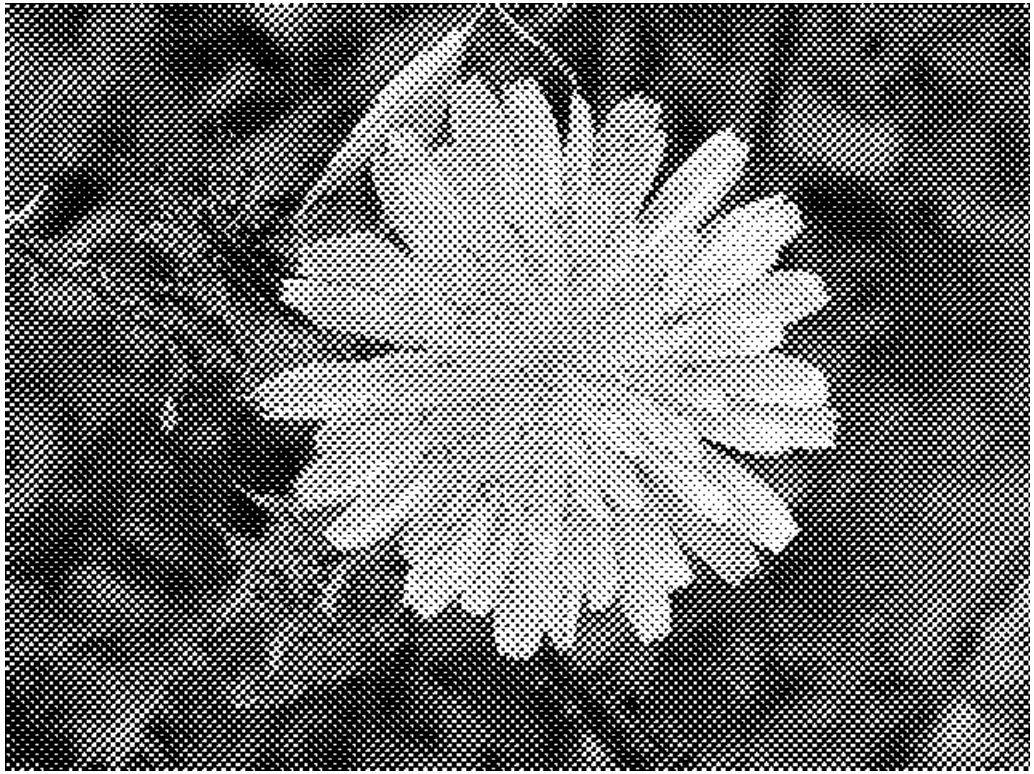
Stroke

Region Model

Pixel Model

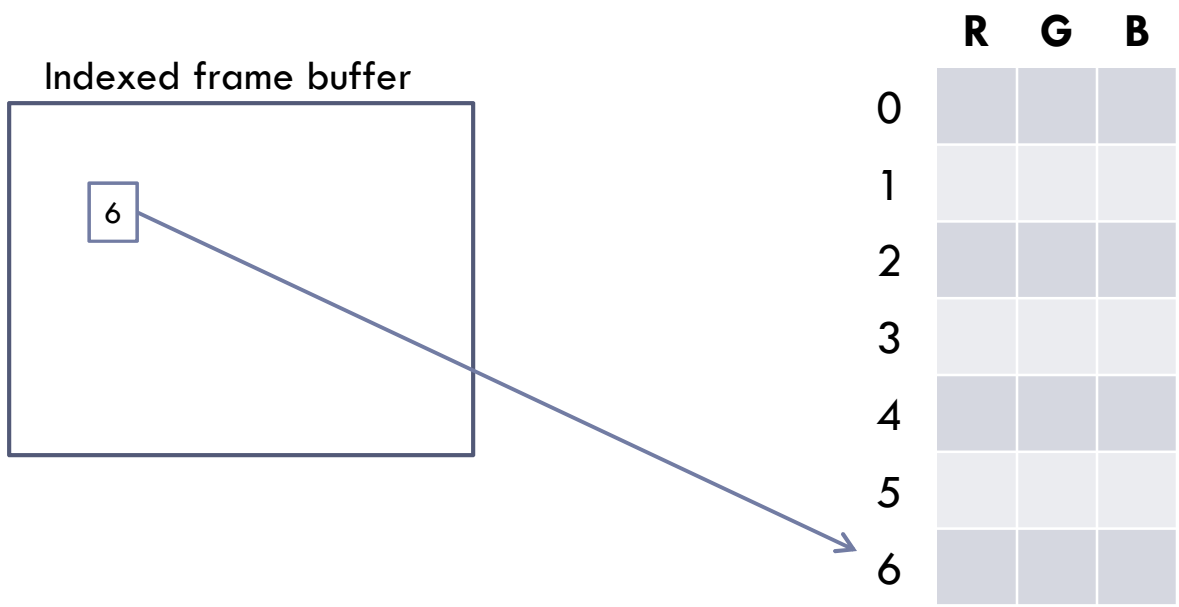
- "Pixel" = PICture ELement
- Basic way to display on a screen
  - A frame buffer contains color value for each spot
- Divide complex images into discrete units
  - Store a color value for each
- Quality determined by Resolution
  - Spatial resolution: number of rows by columns
    - e.g. 1280 x 1024 (monitor), 1440x900 (laptop screen)
    - e.g. 6000 x 4800 (laser printer)
  - Colour resolution (image depth)
    - number of bits per pixel
    - 1-bit, 8-bit, 24-bit, 32-bit colour

Image depth 1bit/pixel



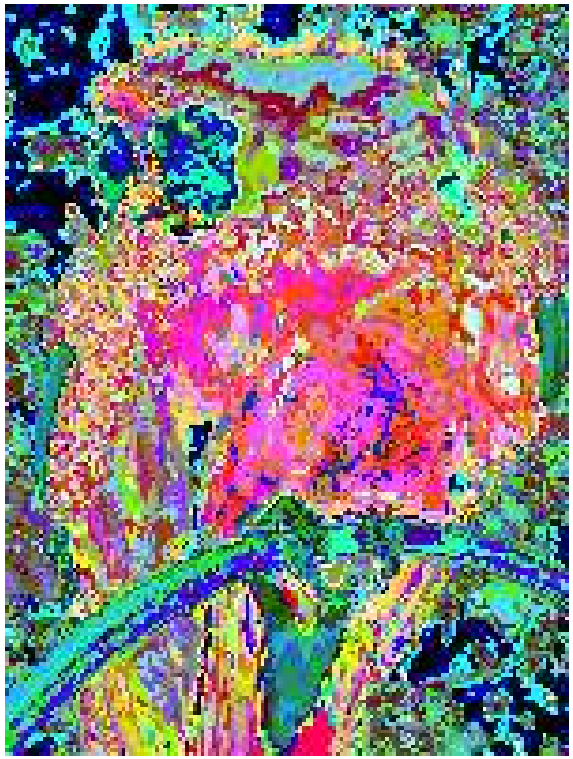
Bit mapped

- Color mapped
  - store index at each pixel into table with 24 bit colours
  - cuts space & computation
  - GIF Limited to 256 colours, 8 bits/pixel instead of 24

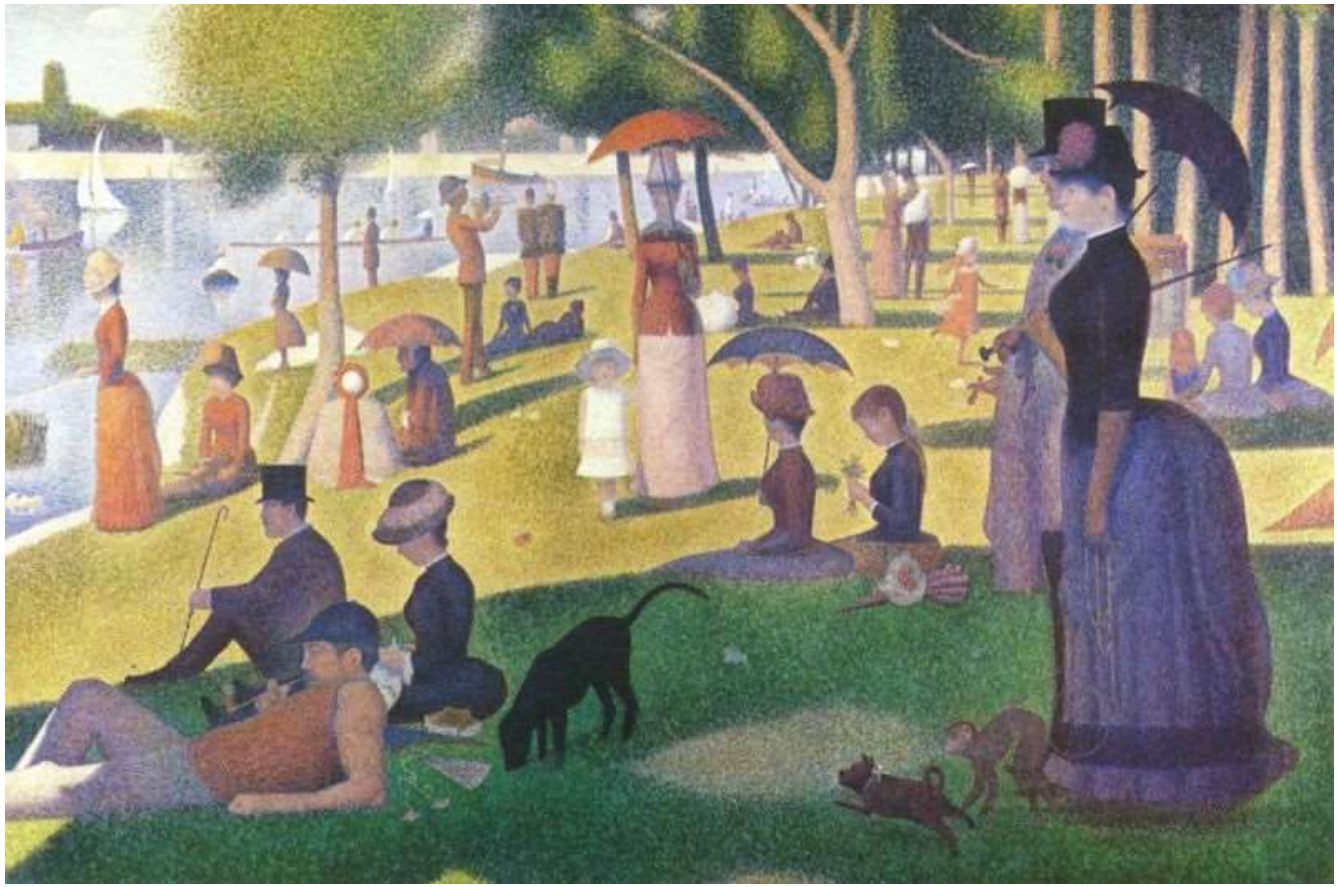


Color-mapped image and palette

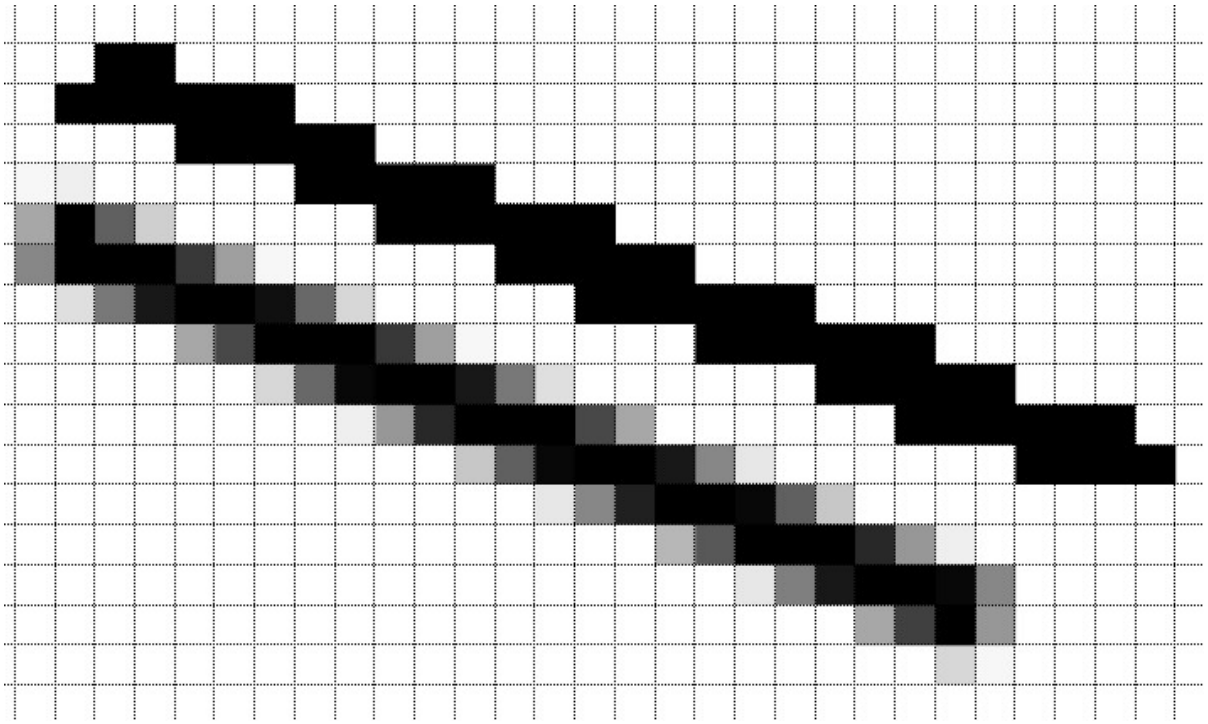
Color-mapped with wrong palette

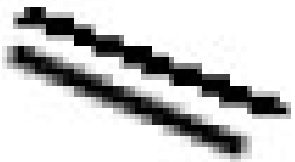


Pixels: good for images, bad for lines



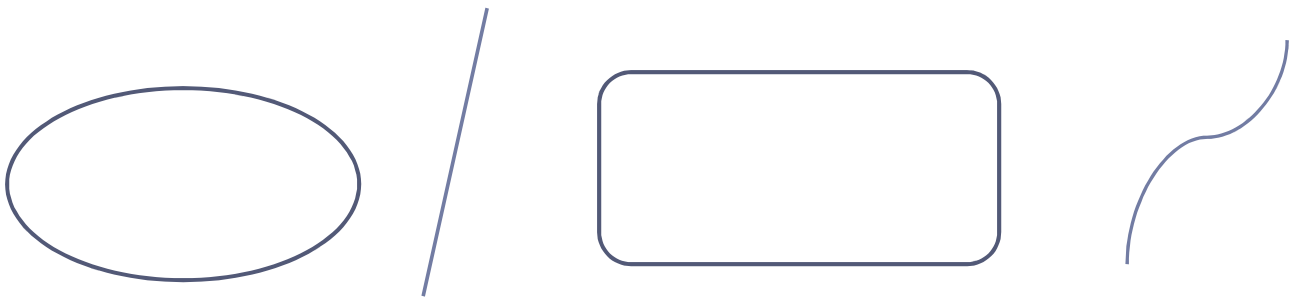
Aliasing and Anti-aliasing





Stroke Model

- Describes image as strokes
  - Position, width, colour
    - Line ((10, 4), (17,4), thick 2, red)
    - Circle ((19, 13), radius 3, thick 3, white)
- Used in early vector displays and plotters
- Most UI toolkits have stroke objects
  - arcs, ellipses, rounded rectangles, etc.



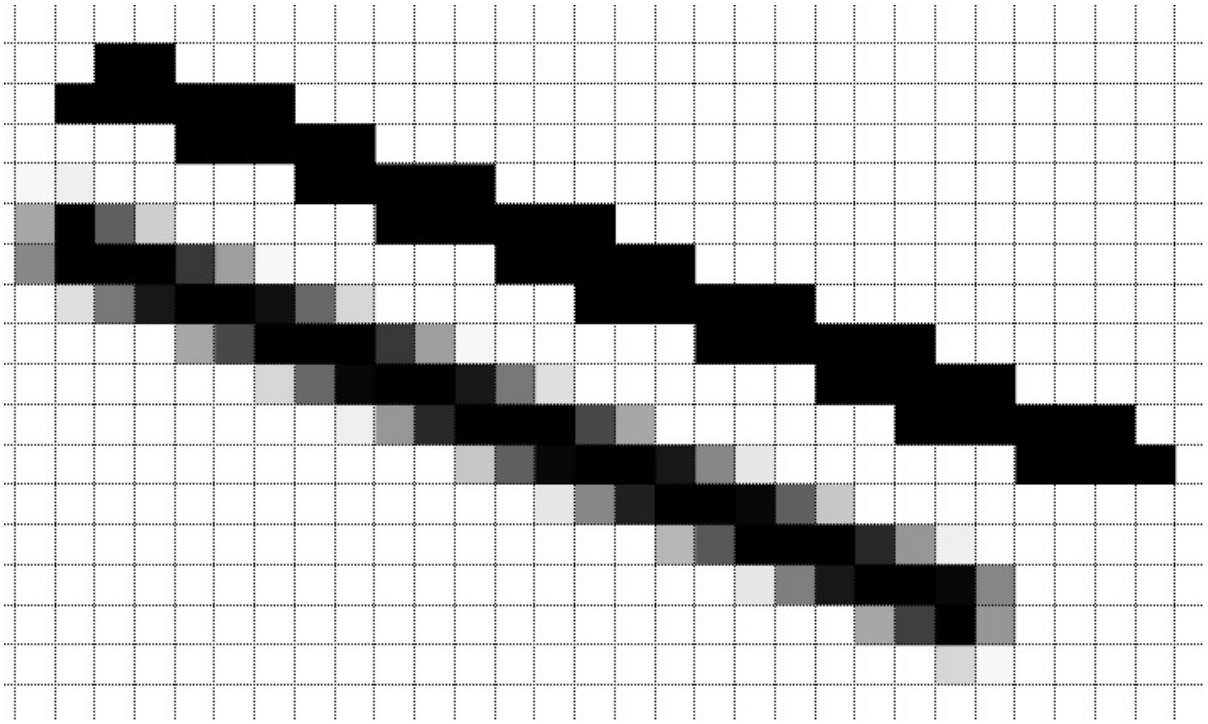
Vector Graphics

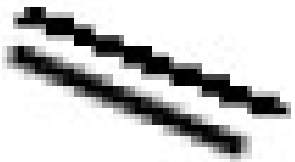
http://en.wikipedia.org/wiki/File:Phone.svg

Region Model

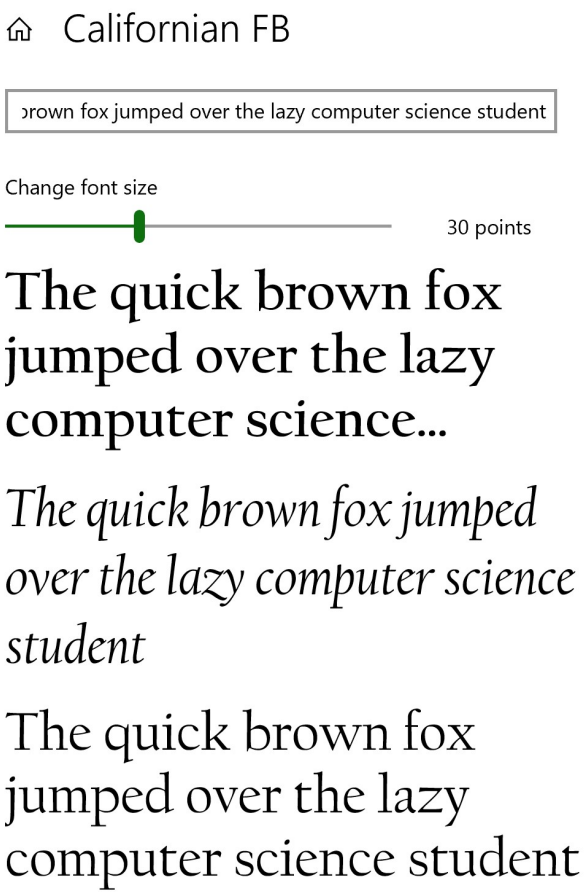
- Use the stroke model to outline a region
- Regions can then be filled
- Advantages
  - Requires very little memory
  - Can be drawn at any resolution
  - PostScript, PDF, TrueType fonts
- Disadvantages?

Stroke/Region on raster display



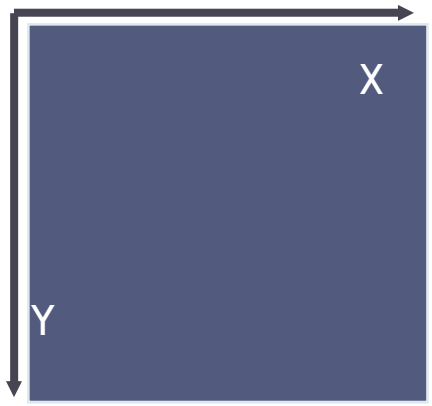


Pixel vs stroke model: fonts



Coordinates

- Device coordinates
  - coordinates of the display screen
  - origin at upper left
- Window coordinates
  - Toolkit presents window as a virtual display in a frame
  - Origin at upper left
  - coordinates expressed in pixels
  - what about scrolling windows?



Coordinates

- Physical coordinates
  - devices with different resolutions
  - specify coordinates in physical units (cm, px, points)
- Model coordinates
  - coordinate system of model objects
  - scale-independent (e.g., [0.0 .. 1.0])
  - conversions may be necessary
- View coordinates
  - Model coordinates multiplied by view extents

Coordinates

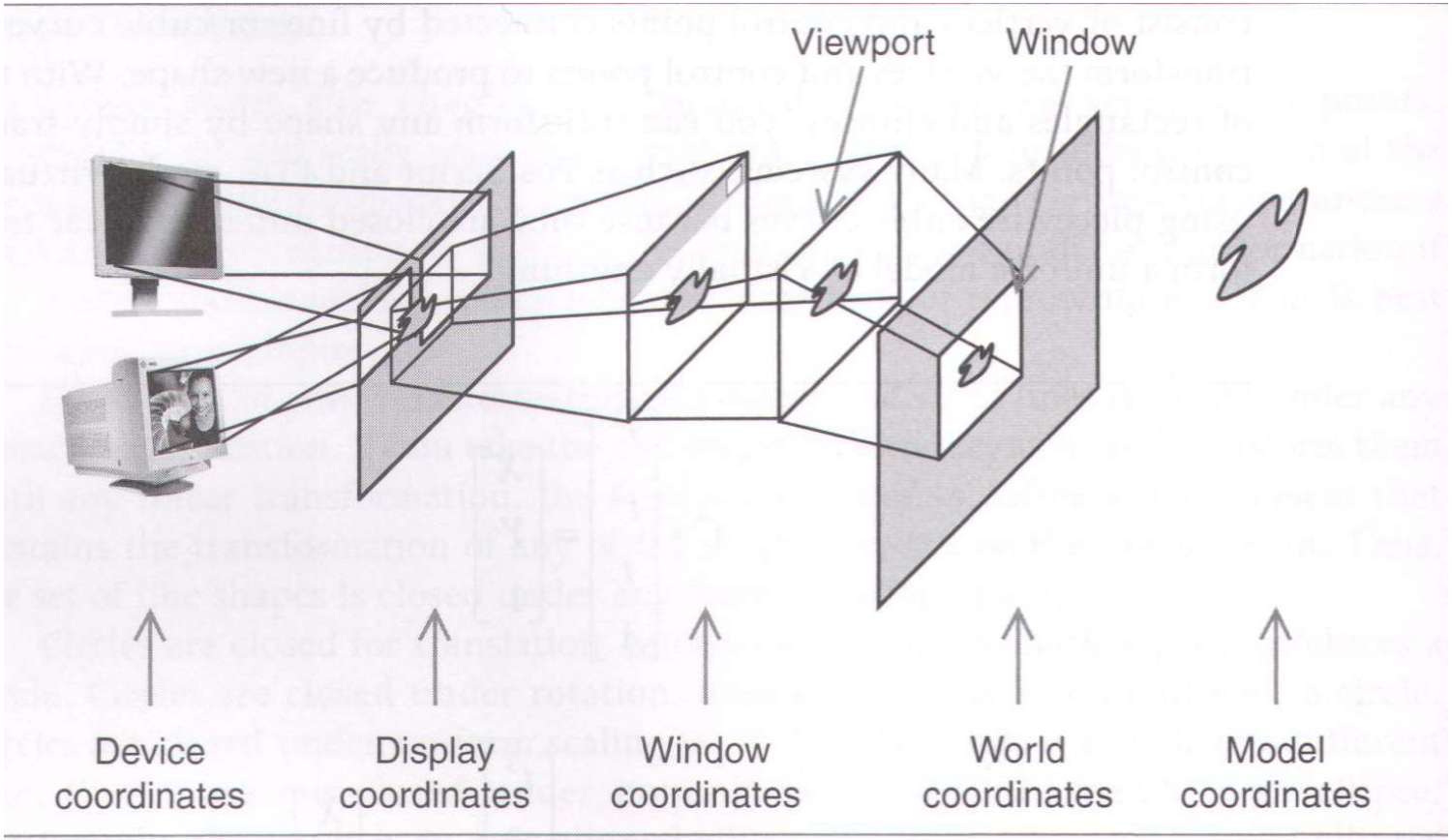
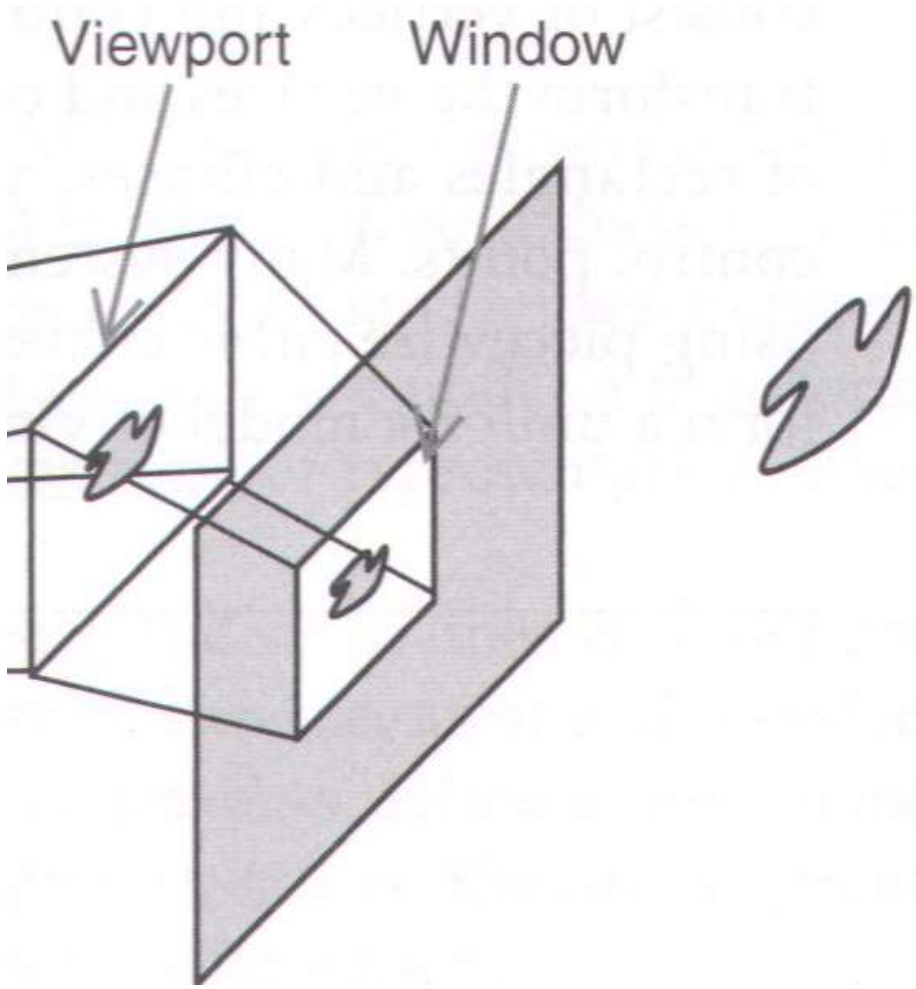
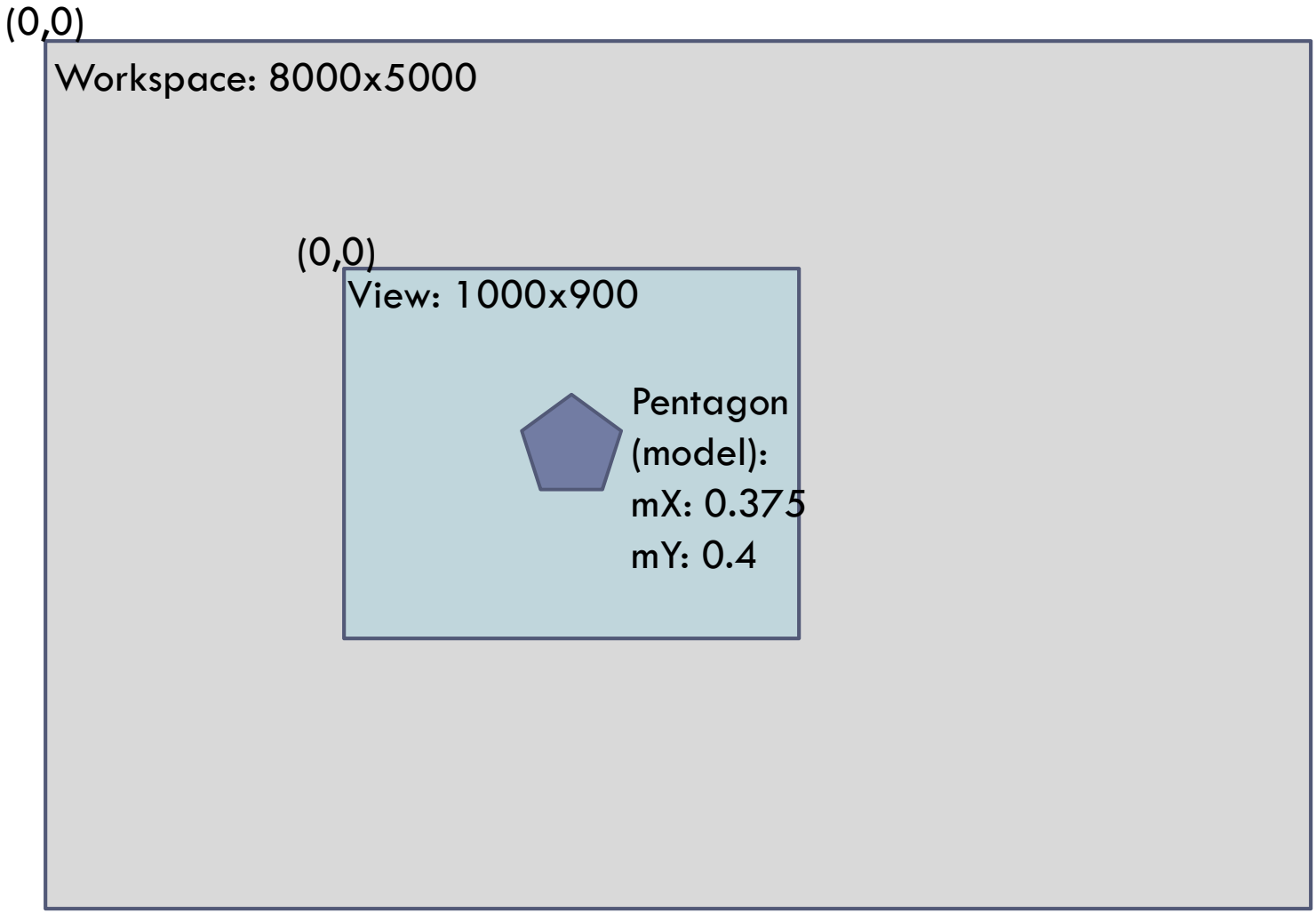


Figure 13.16 – Coordinate systems

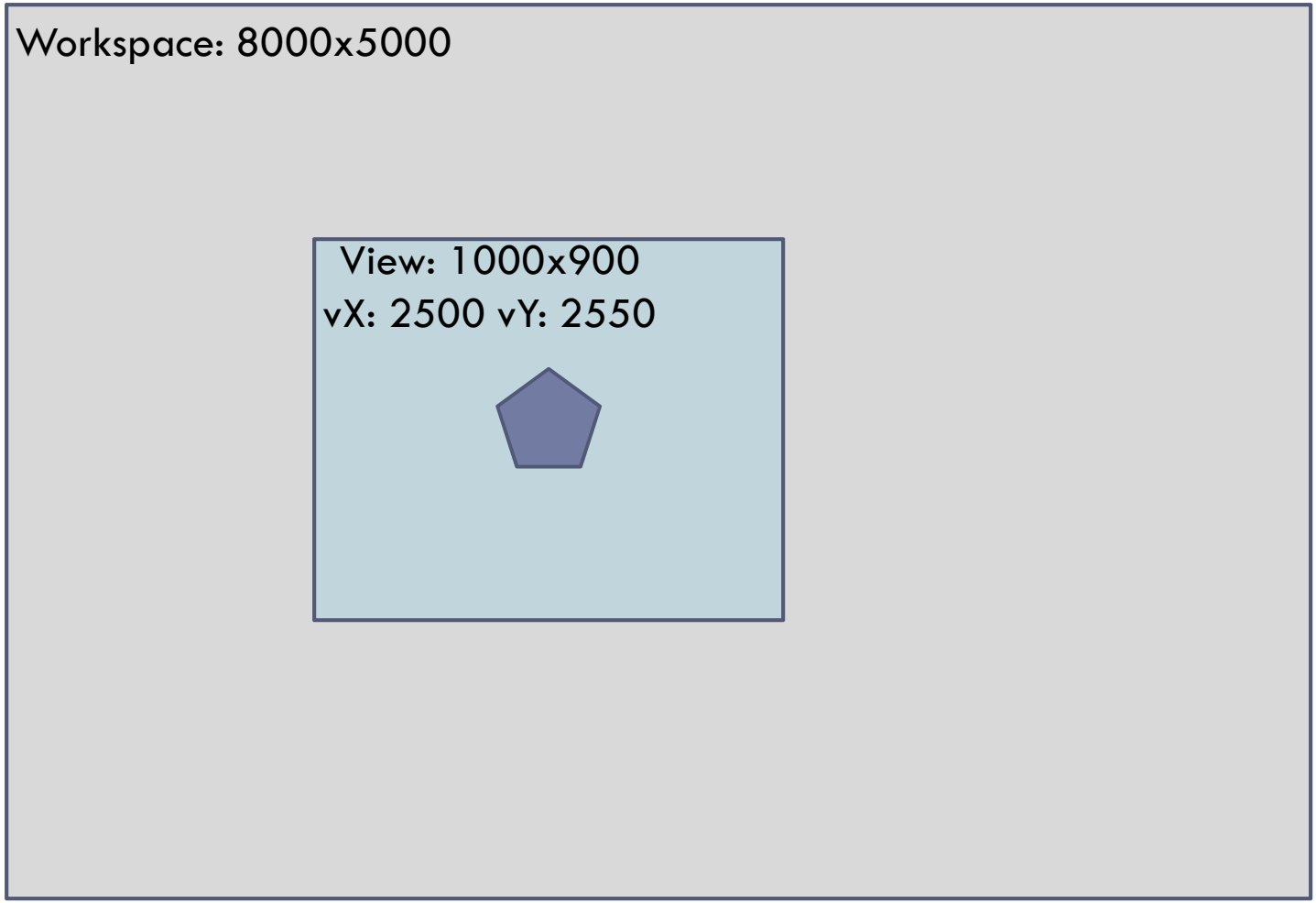
Viewports



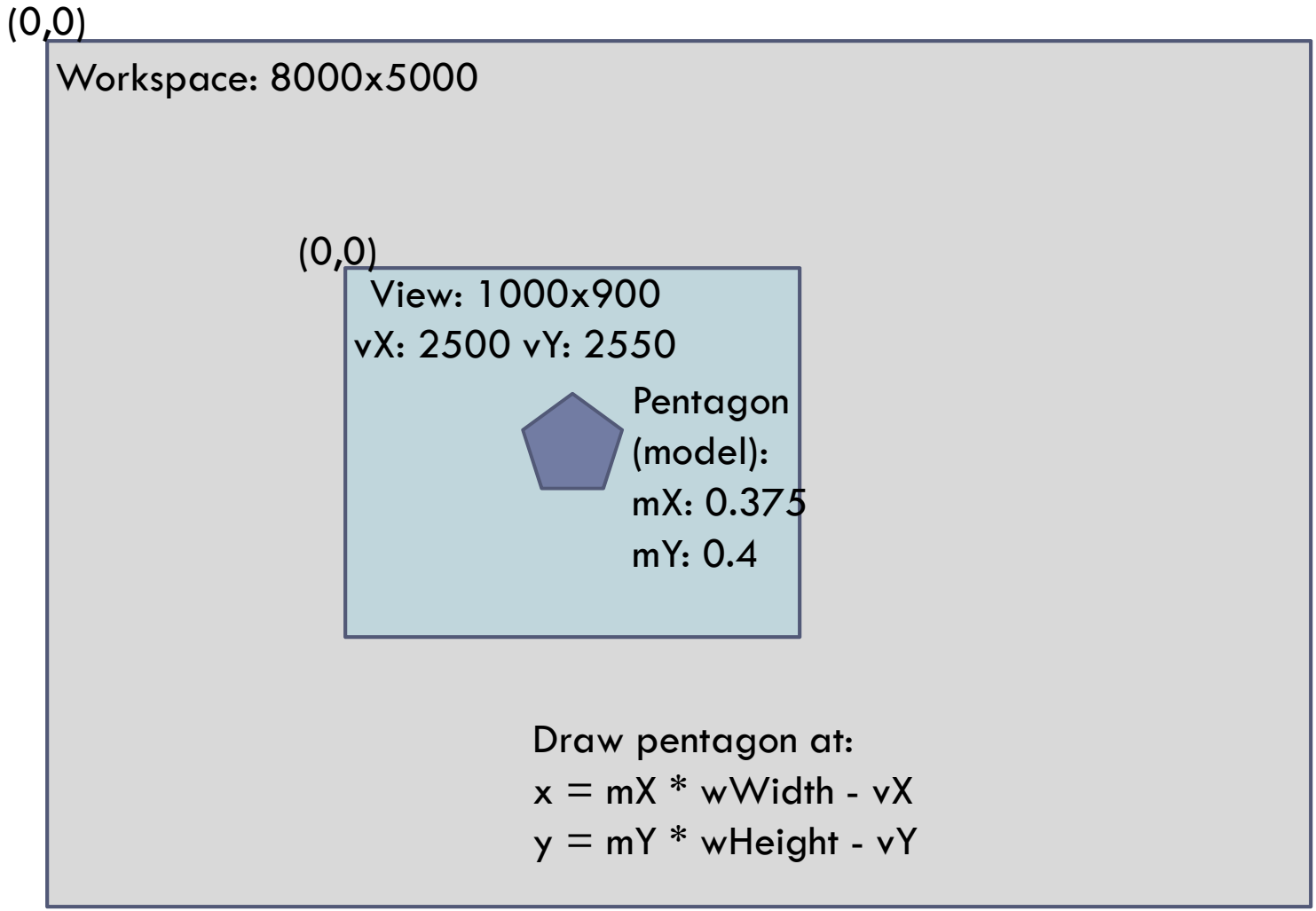
Viewports


(0,0)
Workspace: 8000x5000
(0,0)
View: 1000x900
Pentagon
(model):
mX: 0.375
mY: 0.4

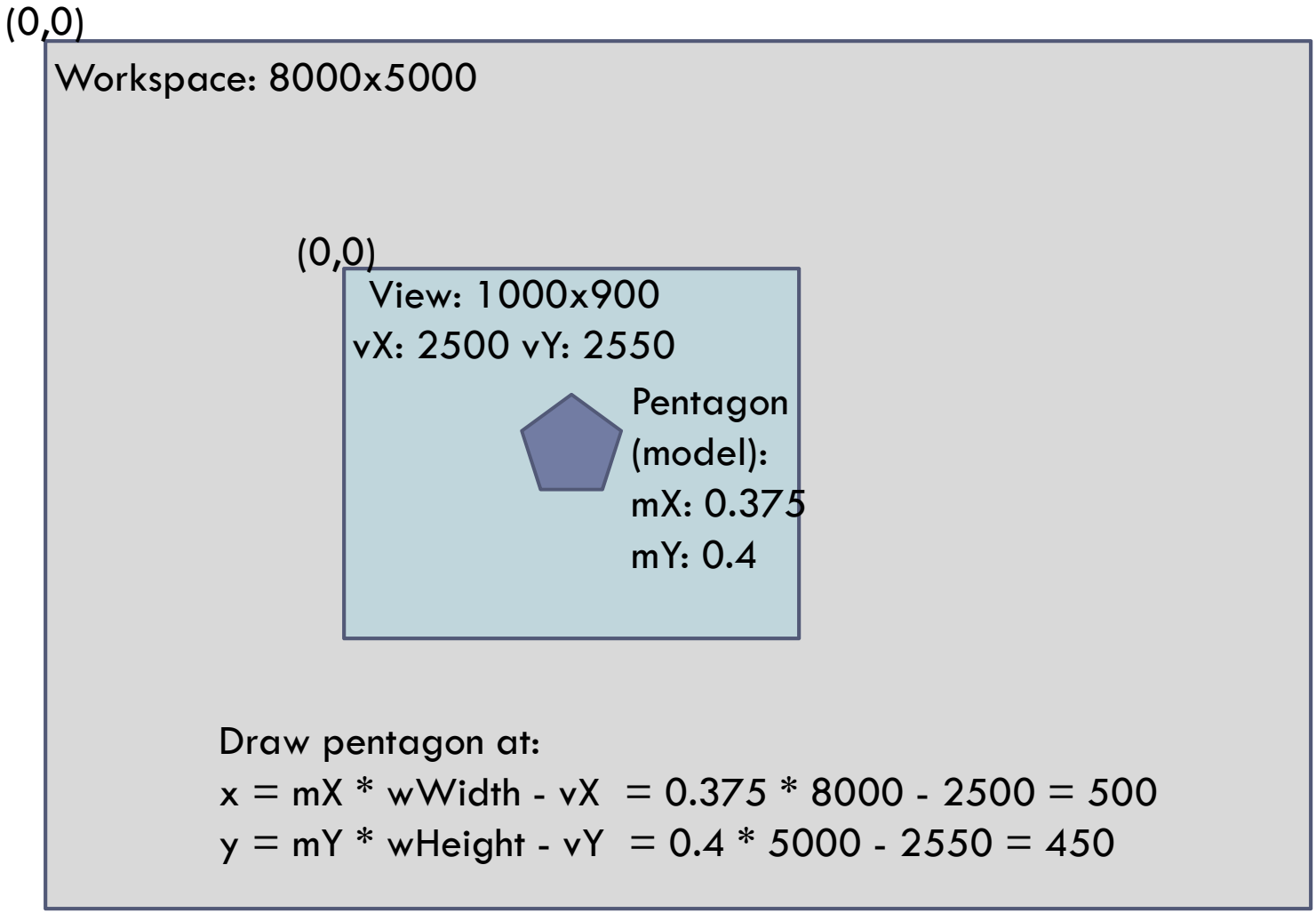
Viewports



Viewports


(0,0)
Workspace: 8000x5000
(0,0)
View: 1000x900
vX: 2500 vY: 2550
Pentagon
(model):
mX: 0.375
mY: 0.4
Draw pentagon at:
x = mX * wWidth - vX
y = mY * wHeight - vY

Viewports


(0,0)
Workspace: 8000x5000
(0,0)
View: 1000x900
vX: 2500 vY: 2550
Pentagon
(model):
mX: 0.375
mY: 0.4
Draw pentagon at:
x = mX * wWidth - vX = 0.375 * 8000 - 2500 = 500
y = mY * wHeight - vY = 0.4 * 5000 - 2550 = 450

Drawing in 2D with UI Toolkits

Graphics Object

- An abstraction for the drawing surface
  - Free us from worrying about other windows management, screen controllers or graphics cards
- Defines methods used for drawing
- Use the same method interface for drawing in
  - windows
  - image in memory
  - printed output
- Called the Canvas in both JavaFX and Android

Drawing with the abstract canvas

- Could have methods for different image types
  - void Canvas::Rectangle (x1, y1, x2, y2, lineWidth, lineColor, fillColor)
- Lots of parameters!
  - shapes have properties in common
    - geometry, line/border width, line/fill color, pattern
- Use current settings of canvas
  - void Canvas::Rectangle (x1, y1, x2, y2)
  - void Canvas::SetLineWidth (lw)
  - int Canvas::GetLineWidth ()

Types of images in the canvas

- Paths and Shapes
  - 1-d objects drawn in a 2-d space
  - no inside or outside, infinitely thin
  - Types (e.g.):
    - lines (end points)
    - rectangles (top left right bottom; or top left height width)
    - circles (center, radius)
    - ellipses (bounding box)
    - splines (control points)
- Bitmaps
  - usually loaded from a file
- Text

Using text in the canvas

- Font family
  - Garamond, Arial, Modern, Times Roman, Courier
  - defines the general shape of the characters
    - some are mono-spaced ("i" gets same space as "G")
    - serif (e.g., Times) vs. sans serif (e.g., Arial)
      - serifs have "feet" at baseline → easier to track
- Style
  - normal, **bold**, *italic*, ***bold italic***
- Size in points (1 point = 1/72 inch)
- Usually simple to draw
    - Canvas.SetFont ("Times", Bold, 10);
    - Canvas.Text (10, 20, "This is the text");

Retained-mode (Node-based) graphics

- Shapes are Nodes in JavaFX
  - predefined item types
  - can be added to any container
  - you can bind events to item types
  - actual painting and double-buffering handled for you

Canvas object in JavaFX

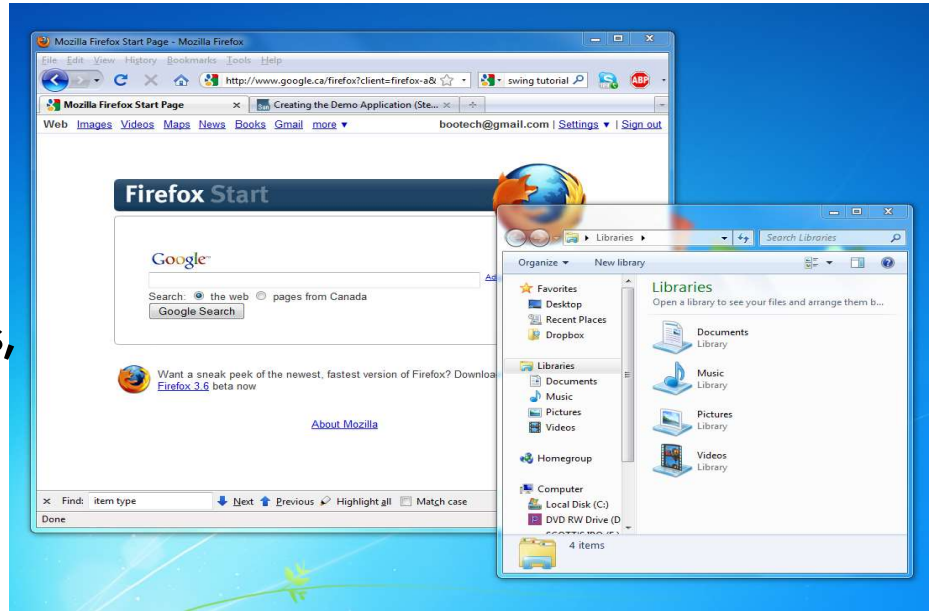
- Similar to abstract canvas
  - GraphicsContext class (retrieved from canvas object)
  - Properties set up the pen
- 2D Graphics
  - Similar capabilities to Java2D

Redrawing

- Only Firefox knows how to redraw a webpage, but it did nothing to cause the situation
- The windowing system decides when a window should be redrawn

Redrawing needs to happen for many reasons:

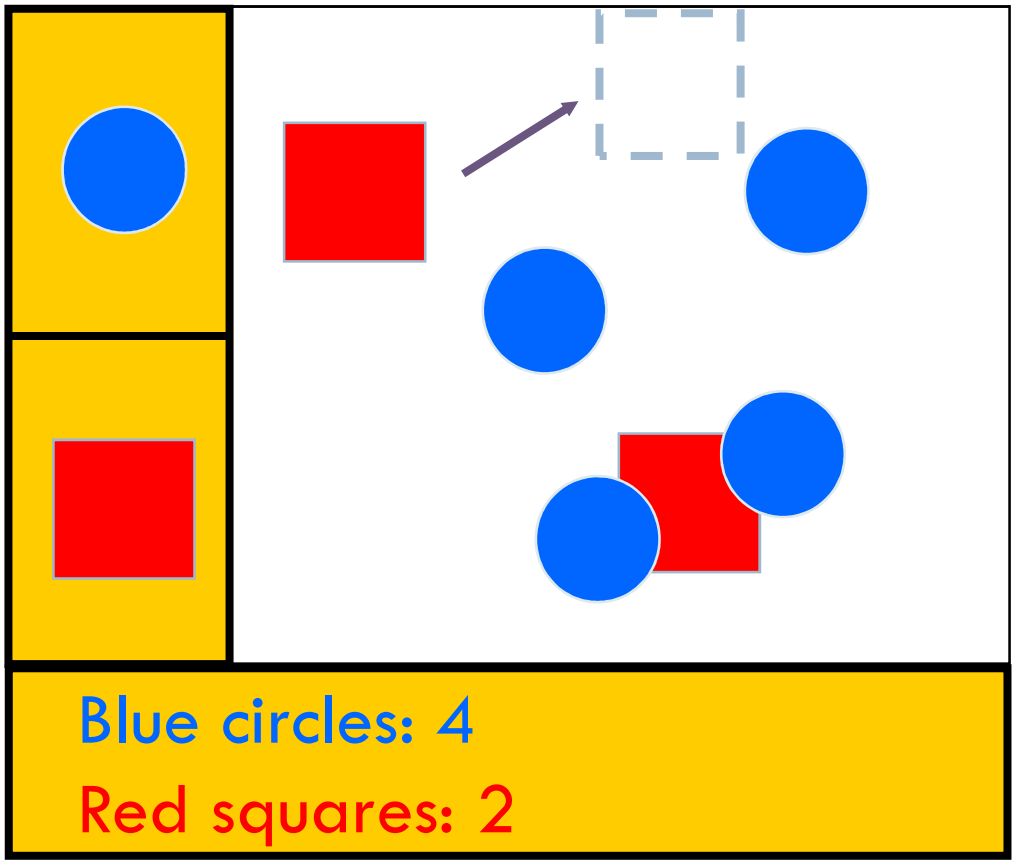
- Window resized
- minimized
- overlapping window changes, moves, closed
- or the model of the application changes



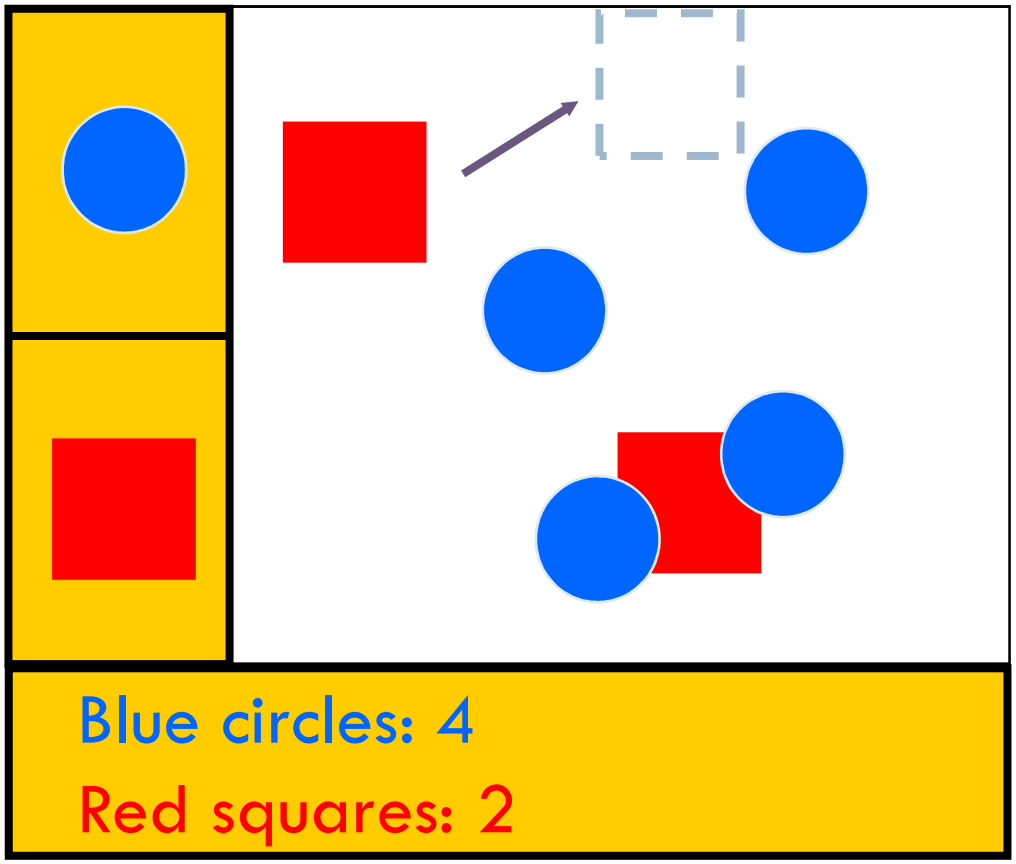
Redraw and Damage

- All GUI widgets (buttons, text fields, etc) need to implement a redraw method
- Windowing system decides when a region needs to be redrawn
- Android:
  - invalidate() called on a widget: without parameters
  - the entire widget must be redrawn
  - invalidate() called on a widget: with an area
  - invalidate(x1,y1,x2,y2)

Moving the red square

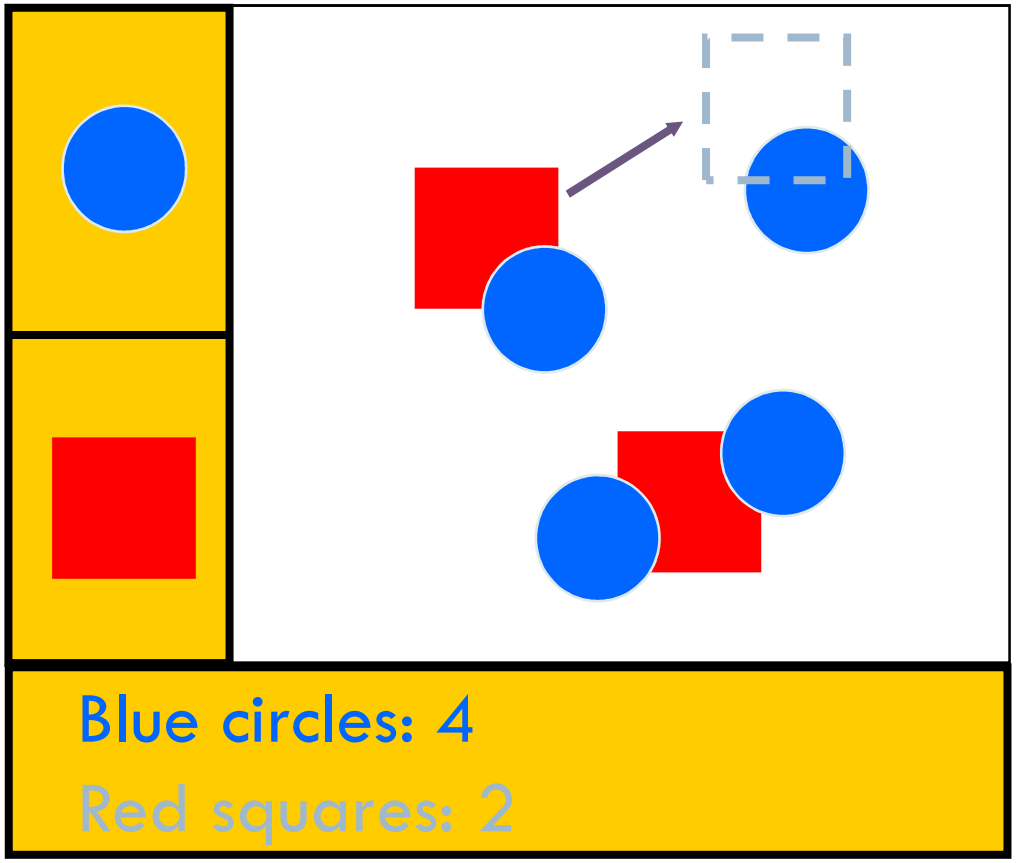


Erase & redraw method

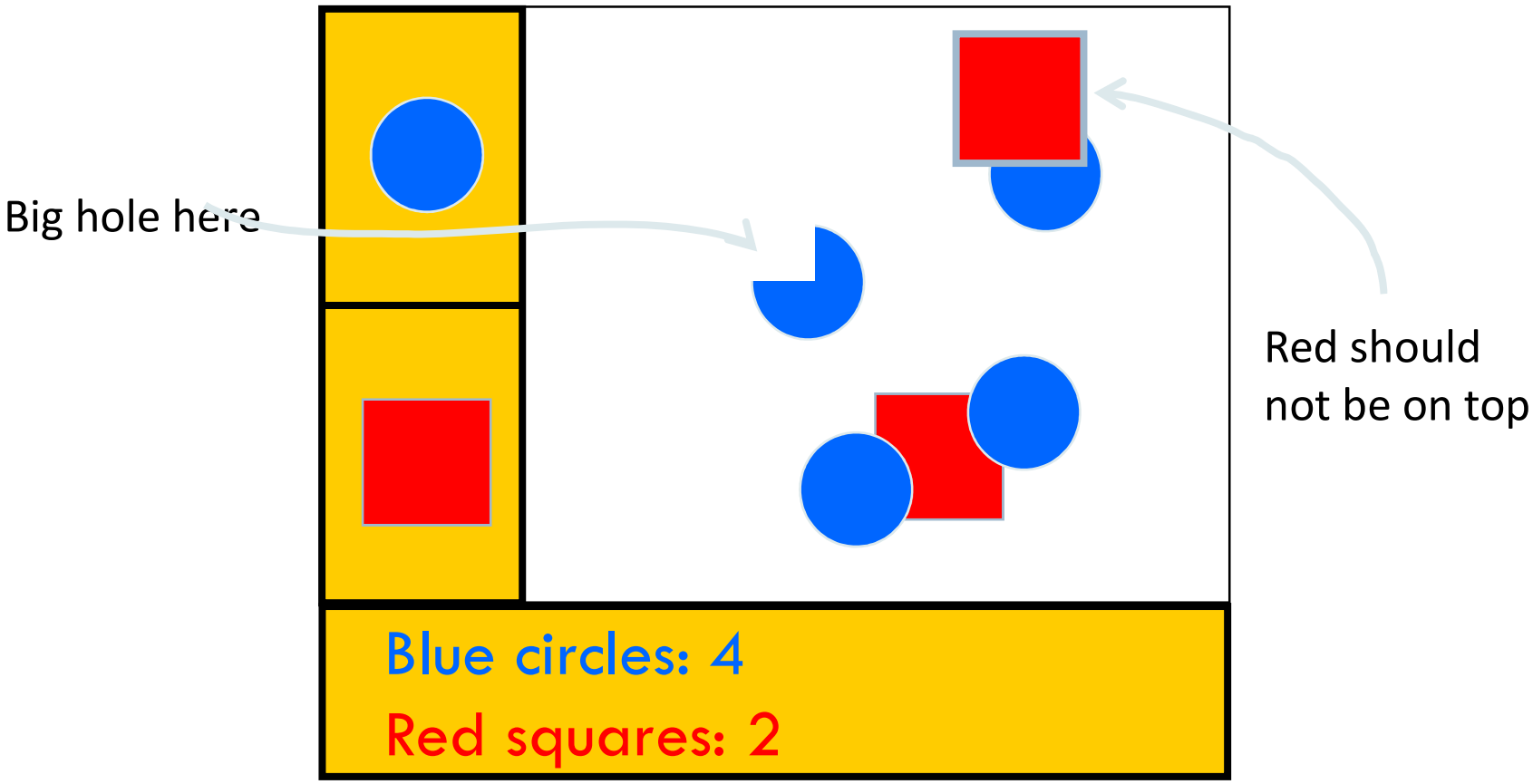


Works fine in this case

Moving the red square, case 2



Erase and redraw



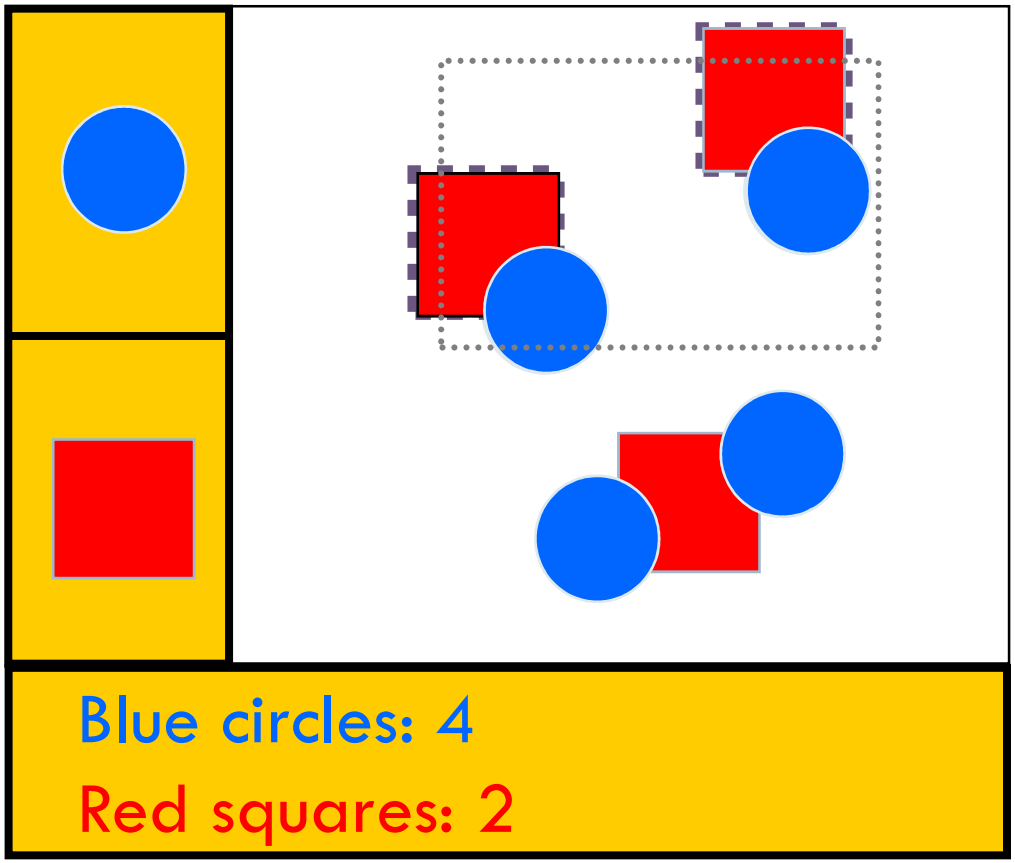
Redrawing the display

- Erase and redraw fails
  - using background color to erase leaves holes
  - drawing shape in new position loses ordering
- Move in model, then redraw view
  - change position of shapes in model
  - model keeps shapes in a desired order
  - tell all views to redraw themselves in order
  - drawbacks?
    - slow for large or complex drawings

Damage / Redraw method

- View informs toolkit or window system of ‘damage’
  - Clip rectangle: area that needs to be repainted
  - view does not redraw immediately
- Toolkit / windowing system
  - batches updates
  - clips them to visible portions of window
- Next time there is no event to process:
  - windowing system calls Redraw method for window

Damage and redraw



Clipping in JavaFX (class BlobView)

```
public void modelChanged(Blob changedBlob, double dx, double dy) {
    gc.save();
    clipLeft = Math.min(changedBlob.x, changedBlob.x - dx) - 5;
    clipWidth = changedBlob.width + Math.abs(dx) + 6;
    clipTop = Math.min(changedBlob.y, changedBlob.y - dy) - 5;
    clipHeight = changedBlob.height + Math.abs(dy) + 6;

    gc.beginPath();
    gc.rect(clipLeft, clipTop, clipWidth, clipHeight);
    gc.clip();
    draw();
    gc.restore();
}
```