Lecture 6: Graphics Basics

Onwards and downwards...

- What goes on underneath the GUI?
- How does the toolkit draw / maintain the interface?
- This class: Intro to basic graphics
  - image models
  - coordinate systems
  - canvas
  - drawing
  - clipping

Models for images

- How can we represent / store these images?

Stroke model

- Image = set of strokes
  Stroke = { start, end, thickness, color }
  - Line ((10,4), (17,4), thick 2, red)
  - Circle ((19,13), radius 3, thick 3, black)

- Easy to display, print (?), plot
- Commonly used in GUI toolkits, in variations
  - arcs, ellipses, rounded rectangles, etc.
Stroke model

• Can we represent this picture as strokes?

Pixel model

• “Pixel” = picture element
  - discrete 2D unit of a picture grid

• Pixel image components
  - spatial resolution = { # rows, # columns }
    - e.g., your favorite monitor or printer resolution
  - image depth = number of bits per pixel
    - why do we need more than one bit per pixel?

Pixel model: Image depth

• Black & white: 1 bit / pixel
• Grayscale: ~8 bits / pixel
  - gray from 0 (black) to 255 (white)
• Full color: ~24 bits / pixel
  - primary colors (red, green, blue) from 0-255
    - but... lots of memory needed!
• Color-mapped: ~8 bits / pixel
  - pixel = index into color table
  - color table elsewhere for lookup
  - some extra work, but lots less space

Pixel model: Aliasing

• Smooth objects are mapped to closest pixels, which can result in jagged edges
• Antialiasing attempts to reconcile with intermediate colors
Region model

- Region = closed stroke with defined fill
  - filled with color or blending pattern
- Advantages
  - little memory for a solid shape
    (unlike other two models)
  - independent of screen/printer resolution
    - shapes
    - text fonts
    - lines?

Coordinate systems

- Device coordinates
  - \textit{global} coordinates
    - coordinates of the display device
    - origin usually at upper left
  - \textit{local} coordinates
    - window is the center of attention
    - essentially, a rectangular virtual display within another
    - can ignore actual location of window in display
    - in Swing, top-level containers like JFrame & JApplet
    - choice depends on association with windows
      - mouse events?
    - generally, all coordinates are in pixels

Coordinate systems

- Physical coordinates
  - display coords are pixel-based, so they scale
different for different resolutions
  - solution: specify coordinates in physical units
    (e.g., inches, centimeters, or printer points)
  - necessary for printers, not used much for GUIs
- Model coordinates
  - coordinate system relative to drawn objects
    - book example: architecture scales (1” = 20’)
  - requires conversion to other coord systems

Canvas

- The first step to presenting a GUI to the user
- Canvas = abstraction of the drawing surface
- Aspects of a canvas
  - width & height
  - memory for drawn image
  - all methods used for drawing
- Canvas is the basis for all subclasses used for
  - defining images
  - creating on-screen windows
  - printing output
Canvas

- What functionality does the canvas provide?
  - drawing of lines & shapes
    - with stroke & region models
  - drawing of text
    - subset of lines & shapes, but highly specialized
  - “clipping”
    - drawing only what you need and/or what’s visible
  - controlling color & texture

Drawing

- Q: How do you draw a rectangle?
  - void Canvas::Rectangle (x1, y1, x2, y2, lineWidth, lineColor, fillColor)
- Do I have to do all that every time?!?
  - shapes have properties in common
    - geometry, line/border width, line/fill color, pattern
  - we can make use of this!
  - Maintain settings within system that generally stay the same within logical blocks
    - void Canvas::SetLineWidth (lw)
    - void Canvas::SetFillColor (color)
    - void Canvas::Rectangle (x1, y1, x2, y2)

Paths

- 1D objects drawn on the 2D canvas
  - no inside or outside
  - infinitely thin
  - can have associated “stroke”, “fill”
- Examples
  - lines (end points)
  - circles (center, radius)
  - ellipses (center, width, height)
  - arcs (circle plus restrictions)
  - splines (parametric cubic equations)

Regions = Closed shapes

- Closed path + fill color/pattern
- If path is not closed, system can “close” it for you
Text

• Back in the old days...
  Text was drawn within blocks.
  - easy storage: 1 byte / character
  - map character --> display representation

• The WYSIWYG revolution
  - “what you see is what you get”
  - suddenly, text was drawn like everything else
  - in hindsight, makes sense
  - but it takes work...

Text

• Font family
  - general shape of characters
  - e.g., Times, Palatino, Helvetica, Courier
  - serif vs. sans-serif (without serif)
    A
    Helvetica
    Palatino
  - fixed-width vs. proportionally-spaced
    every letter has the same width
    every letter has a different width
    Courier
    Times

Text

• Commodore 64 screen shot

Text

• Font style
  - normal, bold, italic, bold italic

• Font size
  - measured in points, 1 point = 1/72 inch
  - generally accurate for printed output
  - on displays, things can vary a great deal

• Font information
  - height (for multiple lines of text)
  - ascenders & descenders (e.g., f & g)
  - leading (space between lines)
  - kerning (space between letters) (Ha vs. Ta)
Text
• Outline vs. bitmapped fonts
  - bitmapped: character stored as pixels
  - outline: character stored as paths/regions
  - why are both useful?

Clipping
• Draw / update only specific areas
  - reason: performance!!!
• Rectangular areas are the most obvious
• Are they sufficient?

Clipping
• Rectilinear regions
  - closed shapes with edges entirely defined by horizontal and vertical lines
  
  ![Rectilinear Region Diagram]

• Pixel mask
  - updated drawing canvas, “masked” with 0/1 pattern --> draw only 1’s

Color models
• RGB model
  - specify color by red, green, & blue components
  - 8 bits/component = 24 bits total per pixel
  - in the end, all others reduce to RGB for display
• HSV model - hue, saturation, & value
  - easier for people to think about
  - hue = primary wavelength (i.e., basic color)
  - saturation = measure of how pure light is
    - high is pure, low means it is mixed w/ white/gray
  - value = intensity or brightness (dark vs. light)
  - direct conversion to RGB
Color models

- **CMY model**
  - represent in terms of mixtures of pigments
    - gets color from light it absorbs & doesn’t reflect
  - mix Cyan, Magenta, Yellow
    - “subtractive primaries”
      - cyan = no red,
        magenta = no green,
        yellow = no blue
  - used by printers and artists

- Color “matching” is quite difficult for printed output

Research Question of the Day

- Users are human — how does this affect how we can/should do graphics?
- Let’s ask two questions...
- **Question #1:**
  - Often, graphics today involves animation.
  - Animation involves re-drawing frames many times per second to “create” motion.
  - How quickly do we need to draw new frames to get smooth motion?
    - << Jumping Bug >>

- Answer to Question #1: No easy answer.
- Some common rates:
  - movies: 24 fps
  - television: 30 fps
    - actually, 1/2 frame @ 60 Hz (interlaced)
  - CRT (cathode-ray tube) monitor: ~60-100 fps
- Generally agreed-to rates:
  - minimal rate for smooth motion: ~20-30 fps
  - limits of the eye: probably ~200 fps (??)
  - medium rates may not be noticeable, but may cause eye strain

- In the context of GUIs, though, lower redraw rates may be OK... sometimes
  - dragging a window?
  - displaying typed text?
  - finding search text?
  - virtual reality simulation?
Research Question of the Day

• Question #2:
  - Today’s computers can display millions of colors.
  - How many colors can people distinguish?
  - How many colors are “enough”?

• Answer to Question #2: No easy answer.
  • Total colors = 256^3 = 16,777,216
  • Discriminable colors probably > 10,000
    - could theoretically be millions by some definition
    - for practical purposes, though, << 16 million
  • This is useful to know!
    - e.g., some “lossy” image compression algorithms
      (e.g., JPEG) take advantage of our inability to discriminate

Research Question of the Day

• But what about different individuals??
  - ~5% of people (!) have some color blindness
  - men 20x more likely than women
  - different types: red-green, yellow-blue
  - what does this mean for GUI design?

Graphics in Swing

• Drawing centers on the Graphics component
  - every component in Swing has a method
    ```java
    public void paintComponent(Graphics g)
    ```
  - that draws the “view” part of the component
  - you can override this method and draw whatever you’d like!
• Let’s look at the JPanel component, which is just a panel with no “stuff” in it
Graphics in Swing

- Drawing in a JPanel

```java
class ShapesPanel extends JPanel {
    final Color bg = Color.lightGray;
    final Color fg = Color.black;

    public ShapesPanel() {
        setBackground(bg);
        setForeground(fg);
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g); // handle default drawing
        << draw panel stuff >>
    }
}
```

- Drawing objects

```java
// Draw a line
g.drawLine(x, y+rectHeight-1, x+rectWidth, y);

// Draw a rectangle
g.drawRect(x, y, rectWidth, rectHeight);

// Draw a 3D rectangle
g.setColor(fg3D);
g.draw3DRect(x, y, rectWidth, rectHeight, true);

// Draw a filled rectangle
g.setColor(fg);
g.drawRoundRect(x, y, rectWidth, rectHeight, 10, 10);

// Draw an oval
g.drawOval(x, y, rectWidth, rectHeight);

// Draw an arc
g.drawArc(x, y, rectWidth, rectHeight, 90, 135);
```

- Drawing text

- pretty straightforward...

```java
g.drawString("Hello World!", x, y);
```

- useful method

```java
int stringWidth (String);
```

---

Handling multiple windows

- Passing information, modal dialog -> frame

- in DayWindow, we'll create the EventWindow

- when the user opens an event...

```java
button.addActionListener (new ActionListener () {
    public void actionPerformed (ActionEvent e) {
        event = new Event (); // or edit selected event, etc.
        EventWindow dialog = new EventWindow (frame, event);
        dialog.pack ();
        dialog.setVisible (true);
        // dialog interaction happens here while dialog is visible;
        // when user hits "OK", following code continues
        if (dialog.event != null) // that is, event actually created
            day.addEvent (event);
    }
});
```
Handling multiple windows

- Passing information, modal dialog -> frame
  - in EventWindow, we’ll manage Event info
  - when the user hits “OK”...

```java
okButton.addActionListener (new ActionListener () {
    public void actionPerformed (ActionEvent e) {
        // if not created already, create new event
        event = new Event ();
        // transfer info from components to data structure
        event.title = titleTextField.getText ();
        // etc.

        // now make window invisible, thus passing control back to parent
        self.setVisible (false);
    }
});
```