Today’s Topics

- Design principles
- Frame animation
- Palette animation
- Property animation
- Pacing & path

Why Animation?

- Purpose of application
  - Games, simulations, tutorials, video players
- Feedback
  - Visualizing changes not made by user
  - Keeping the user oriented during transitions
  - Displaying progress
- Help
  - “Animated icons”
  - Moving mouse around to show how to use UI
- Reinforcing illusion of direct manipulation
- Aesthetic appeal and engagement
Animation Isn’t Always Needed

- Existing events are often enough to provide incremental screen changes
  - User’s mouse events drive scrolling
  - Program events can drive a progress bar
- But bursty or slow events may need animation
- Short distances and short time periods
  - time < 100 ms
  - distance < width of the moving object

Design Principles

- Frame rate > 20 frames per second
  - 10 fps is convincing but looks jerky
  - Film is 24 fps, TV (NTSC) 30 fps
- Big jumps are disruptive
  - Use motion blur if frame rate can’t keep up with object speed
  - Rule of thumb: if object moves more than its width between frames, fill in with motion blur (smear of color or multiple images)
- Animation in direct manipulation
  - Solidity (motion blur, fading in/out)
  - Anticipation (wind-up before starting to move)
  - Slow-in/slow-out
  - Follow through (wiggle back and forth when stopping)
- Keep feedback animation short
  - Many users will wait for it to stop before continuing
- Use animation sparingly
  - Constant motion is distracting and agitating

Pixel Model: Frame Animation

- Frame animation
  - Animated GIF
  - Graphics.drawImage(..., this) automatically animates GIFs by calling this.repaint() when it’s time to show the next frame

Pixel Model: Palette Animation

- Palette animation
  - Split color index into layers
  - Double-buffering by making only one layer visible while drawing into the other
  - Objects can be moved around in one layer without need to redraw underlying layer
  - Fade-in by interpolating colors between layers
**Pixel/Stroke Model: Event Loop Approach**

- **Approach**
  - Set a periodic timer for 1/frame rate
  - Repaint every timer tick
  - Paint method uses current clock time to compute positions/sizes/etc to draw animated objects
  - Stop timer when animation complete or interrupted
- **May be hard to achieve smooth animation**
  - Event-handling may be bursty
  - Getting from timer tick to paint method requires two passes through event queue
  - Processing user input events has priority over animation repaints

**Pixel/Stroke Model: Animation Loop Approach**

- **Tight animation loop approach**
  - Repeat as fast as possible,
  - Check and handle input events
  - Paint everything for current clock time
  - (Optional: sleep a bit to yield to other processes)

**Component Model: Property Animation**

- Set periodic timer
- Every timer tick, update component properties as a function of current clock time
  - Position, size, color, opacity

**Pacing and Path**

- **Pacing function maps time t to parameter s [0,1]**
  - Linear: $s = t / \text{duration}$
  - Slow-in/slow-out: $s \sim \text{atan}(t)$
- **Path function maps s to property value v**
  - Linear: $(x,y) = (1-s)^2(x_0,y_0) + 2s(1-s)(x_1,y_1) + s^2(x_2,y_2)$
  - Quadratic Bezier curve:
  - Color: HSV vs. RGB
Declarative Animation in Avalon

```xml
<Rectangle Fill="Black"
    Height="100px" Width="100px"
    Canvas.Bottom="5px"
    Canvas.Right="5px">
    <Rectangle.Height>
        <LengthAnimationCollection>
            <LengthAnimation From="100" To="50"
                Duration="3"
                RepeatDuration="Indefinite" />
        </LengthAnimationCollection>
    </Rectangle.Height>
</Rectangle>
```