core WEB programming

Multithreaded Graphics

Agenda

- **Approaches for multithreaded graphics**
  - Redraw everything in paint
  - Have routines other than paint draw directly on window
  - Override update and have paint do incremental updating
  - Double buffering
- **Reducing flicker in animations**
- **Implementing double buffering**
- **Animating images**
- **Controlling timers**

Multithreaded Graphics: Alternative Approaches

- **Redraw everything in paint**
  - Simple and easy, but if things change quickly it is slow and can result in a flickering display
- **Have routines other than paint directly do drawing operations**
  - Easy, efficient, and flicker-free, but results in "transient" drawing that is lost next time the screen is redrawn
- **Override update and have paint do incremental updating**
  - Eliminates the flicker and improves efficiency somewhat, but requires the graphics to be non-overlapping
- **Double buffering**
  - Most efficient option and has no problem with overlapping graphics.
  - More complex and requires additional memory resources

Redraw Everything in `paint`

- **Idea**
  - Have user actions change non-graphical data structures, then call repaint.
  - The repaint method sets a flag that tells the event-handling process to call update.
  - The standard update method clears the screen and then calls paint.
  - The paint method completely redraws everything.
- **Advantage**
  - Easy
- **Disadvantages**
  - Flickers, slow.

Redrawing Everything in `paint`: Example

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;

/** Applet that draws a small circle where you click. */

public class DrawCircles extends Applet {
  private Vector circles;

  public void init() {
    circles = new Vector();
    addMouseListener(new CircleDrawer());
    setBackground(Color.white);
  }
  ...
```


Redrawing Everything in paint: Example (Continued)

```
/** When you click the mouse, create a SimpleCircle,
  *  put it in the Vector, and tell the system
  *  to repaint (which calls update, which clears
  *  the screen and calls paint).
  */

 private class CircleDrawer extends MouseAdapter {
   public void mousePressed(MouseEvent event) {
     circles.addElement(new SimpleCircle(event.getX(),
                                         event.getY(),
                                         25));
     repaint();
   }
 }
```

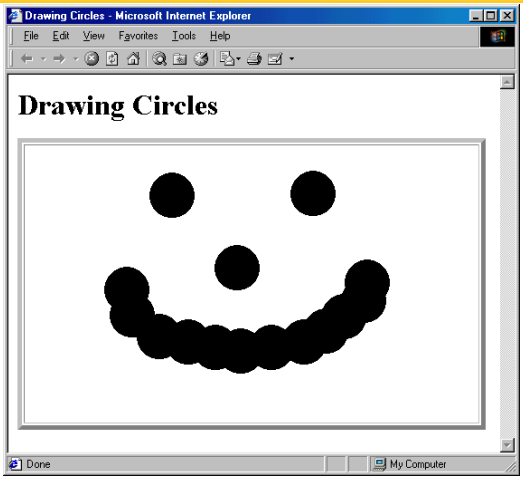
Redrawing Everything in paint: Example (Continued)

```
  ...

  /** This loops down the available SimpleCircle
   *  objects, drawing each one.
   */
  public void paint(Graphics g) {
    SimpleCircle circle;
    for(int i=0; i<circles.size(); i++) {
      circle = (SimpleCircle)circles.elementAt(i);
      circle.draw(g);
    }
  }
}
```


Redrawing Everything in `paint`: Example (Continued)

```
public class SimpleCircle {
  private int x, y, radius;

  public SimpleCircle(int x, int y, int radius) {
    setX(x);
    setY(y);
    setRadius(radius);
  }

  /** Given a Graphics, draw the SimpleCircle
   *  centered around its current position.
   */
  public void draw(Graphics g) {
    g.fillOval(x - radius, y - radius,
               radius * 2, radius * 2);
  }
  ...
}
```

Redrawing everything in paint: Result



By storing results in a permanent data structure and redrawing the whole structure every time paint is invoked, you cause the drawing to persist even after the window is covered up and reexposed

Have Other Routines Draw Directly on Window

- **Idea**
  - Arbitrary methods (i.e., other than paint) can call getGraphics to obtain the window's Graphics object
  - Use that Graphics object to draw
  - Drawing lost if
    - Window covered up and reexposed
    - The update method called (e.g., via repaint)
- **Advantage**
  - Fast
- **Disadvantage**
  - Temporary

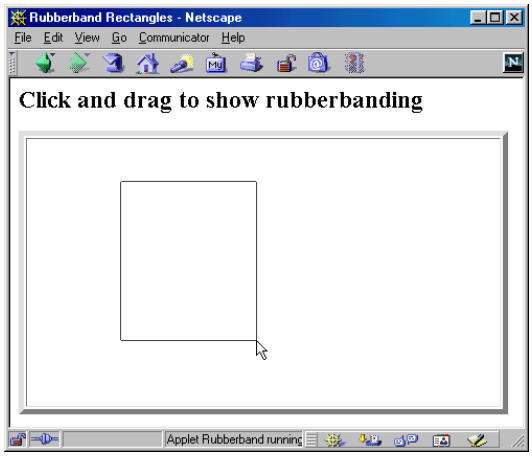
Drawing Directly on Window: Example

```
public class Rubberband extends Applet {
  private int startX, startY, lastX, lastY;
  ...
  private void drawRectangle(Graphics g, int startX,
                 int startY, int stopX, int stopY ) {
    int x, y, w, h;
    x = Math.min(startX, stopX);
    y = Math.min(startY, stopY);
    w = Math.abs(startX - stopX);
    h = Math.abs(startY - stopY);
    g.drawRect(x, y, w, h);
  }
  ...

  private class RectRecorder extends MouseAdapter {
    public void mousePressed(MouseEvent event) {
      startX = event.getX();
      startY = event.getY();
      lastX = startX;
      lastY = startY;
    }
```


Drawing Directly on Window: Example (Continued)

```
  public void mouseReleased(MouseEvent event) {
      Graphics g = getGraphics();
      g.setXORMode(Color.lightGray);
      drawRectangle(g, startX, startY, lastX, lastY);
    }
  }

  private class RectDrawer extends MouseMotionAdapter {
    public void mouseDragged(MouseEvent event) {
      int x = event.getX();
      int y = event.getY();
      Graphics g = getGraphics();
      g.setXORMode(Color.lightGray);
      drawRectangle(g, startX, startY, lastX, lastY);
      drawRectangle(g, startX, startY, x, y);
      lastX = x;
      lastY = y;
    }
  }
}
```

Drawing Directly on Window: Result



By retrieving the Graphics object, methods other than paint can draw directly on the window

Override `update` and Have `paint` do Incremental Updating

- **Idea**
  - Have repaint (which triggers update) avoid clearing the screen each time by overriding update as follows:

```
public void update(Graphics g) {
  paint(g);
}
```

  - Then, assuming objects don't overlap, erase each object at its old location by drawing over it in the background color then drawing it at the new location
- **Advantages**
  - No flicker, faster
- **Disadvantage**
  - Fails for overlapping images

Incremental Updating: Bounce Applet

```
public class Bounce extends Applet
                    implements Runnable, ActionListener {
  private Vector circles;
  private int width, height;
  private Button startButton, stopButton;
  private Thread animationThread = null;
  ...
  public void actionPerformed(ActionEvent event) {
    if (event.getSource() == startButton) {
      if (circles.size() == 0) {
        // Erase any circles from previous run.
        getGraphics().clearRect(0, 0, getSize().width,
                                      getSize().height);
        animationThread = new Thread(this);
        animationThread.start();
      }
      int radius = 25;
      int x = radius + randomInt(width - 2 * radius);
      int y = radius + randomInt(height - 2 * radius);
      int deltaX = 1 + randomInt(10);
      int deltaY = 1 + randomInt(10);
      circles.addElement(new MovingCircle(x, y, radius,
                                          deltaX, deltaY));
```


Bounce Applet (Continued)

```
  } else if (event.getSource() == stopButton) {
      if (animationThread != null) {
        animationThread = null; // Stop animation
        circles.removeAllElements();
      }
    }
    repaint();
  }

  public void run() {
    Thread myThread = Thread.currentThread();
    // Really while animationThread not null
    while(animationThread == myThread) {
      repaint();
      pause(100);
    }
  }
  ...
```

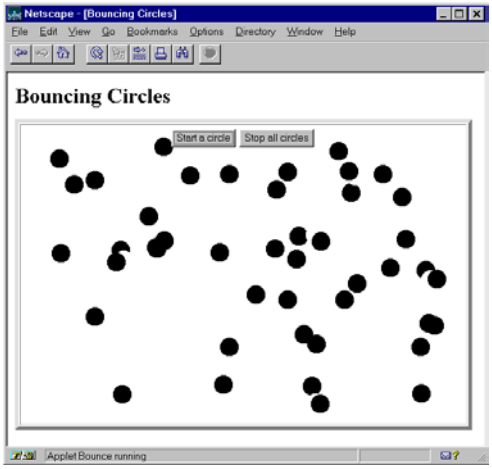
Bounce Applet (Continued)

```
  public void update(Graphics g) {
    paint(g);
  }

  public void paint(Graphics g) {
    MovingCircle circle;
    for(int i=0; i<circles.size(); i++) {
      circle = (MovingCircle)circles.elementAt(i);
      g.setColor(getBackground());
      circle.draw(g); // Old position
      circle.move(width, height);
      g.setColor(getForeground());
      circle.draw(g); // New position
    }
  }

  ...
}
```


Incremental Updating: MovingCircle Class

```
public class MovingCircle extends SimpleCircle {
  private int deltaX, deltaY;
  ...
  public void move(int windowWidth, int windowHeight) {
    setX(getX() + getDeltaX());
    setY(getY() + getDeltaY());
    bounce(windowWidth, windowHeight);
  }

  private void bounce(int windowWidth, int windowHeight) {
    int x = getX(), y = getY(), radius = getRadius(),
        deltaX = getDeltaX(), deltaY = getDeltaY();
    if ((x - radius < 0) && (deltaX < 0))
      setDeltaX(-deltaX);
    else if ((x + radius > windowWidth) && (deltaX > 0))
      setDeltaX(-deltaX);
    if ((y -radius < 0) && (deltaY < 0))
      setDeltaY(-deltaY);
    else if((y + radius > windowHeight) && (deltaY > 0))
      setDeltaY(-deltaY);
  }
  ...
}
```

Incremental Updating, Result



Incremental updating from paint can be flicker free and relatively fast, but it does not easily handle overlapping items

Option 4: Double Buffering

- **Idea**
  - Draw into an off-screen pixmap, then draw that pixmap on window
- **Outline**
  1. Override update to simply call paint
     - This prevents the flicker that would normally occur each time update clears the screen before calling `paint`
  2. Allocate an Image using createImage
     - Note that since this image uses native window-system support, it cannot be done until a window actually appears
  3. Look up its `Graphics` object using `getGraphics`
     - Unlike with windows, where you need to look up the `Graphics` context each time you draw, with images it is reliable to look it up once, store it, and reuse the same reference thereafter
  4. For each step, clear the image and redraw all objects onto it
     - Dramatically faster than drawing onto a visible window
  5. Draw the offscreen image onto the window
     - Use drawImage

Double Buffering: Pros & Cons

- **Advantages**
  - Much faster
  - Can easily handle overlapping objects
- **Disadvantages**
  - More complex
  - Memory requirements for offscreen pixmap
  - Sometimes less incremental update of display

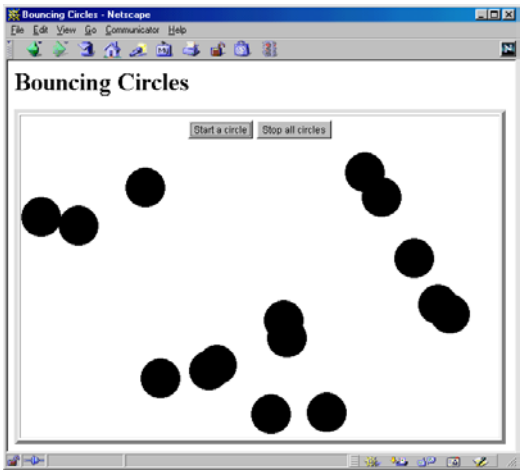
Double Buffering: Example

```
public class DoubleBufferBounce extends Applet implements
                                        Runnable, ActionListener {
  private Vector circles;
  private int width, height;
  private Image offScreenImage;
  private Graphics offScreenGraphics;
  private Button startButton, stopButton;
  private Thread animationThread = null;

  public void init() {
    setBackground(Color.white);
    width = getSize().width;
    height = getSize().height;
    offScreenImage = createImage(width, height);
    offScreenGraphics = offScreenImage.getGraphics();
    // Automatic in some systems, not in others.
    offScreenGraphics.setColor(Color.black);
    circles = new Vector();
    ...
  }
```

Double Buffering: Example

```
public void run() {
  MovingCircle circle;
  Thread myThread = Thread.currentThread();
  // Really while animationThread not null.
  while(animationThread == myThread) {
    for(int j=0; j<circles.size(); j++) {
      circle = (MovingCircle)circles.elementAt(j);
      circle.move(width, height);
    }
    repaint();
    pause(100);
  }
}
public void paint(Graphics g) {
  offScreenGraphics.clearRect(0, 0, width, height);
  MovingCircle circle;
  for(int i=0; i<circles.size(); i++) {
    circle = (MovingCircle)circles.elementAt(i);
    circle.draw(offScreenGraphics);
  }
  g.drawImage(offScreenImage, 0, 0, this);
}
}
```


Double Buffering: Result



At the expense of memory and some complexity, double buffering allows fast, flicker-free updating of possibly overlapping images

Array-Based Animation

- **Idea**
  - Load a sequence of images into an array
  - Start a thread to cycle through the images and draw to the graphics object
    - Each time the thread loops through the while loop, the array index is incremented and repaint (which triggers update) is called to update the images on the screen
  - Stop the animation by setting a flag
    - In an applet, end the animation from the applet's stop method

Array-Based Animation: Example

```
public class ImageAnimation extends Applet {
  private static final int NUMDUKES  = 2;
  private Duke[] dukes;  // Duke has array of images
  private int i;

  public void init() {
    dukes = new Duke[NUMDUKES];
    setBackground(Color.white);
  }

  public void start() {
    int tumbleDirection;
    for (int i=0; i<NUMDUKES ; i++) {
      tumbleDirection = (i%2 == 0) ? 1 :-1;
      dukes[i] = new Duke(tumbleDirection, this);
      dukes[i].start();
    }
  }

  ...
```

Animation Example (Continued)

```
public void update(Graphics g) {
  paint(g);
}

public void paint(Graphics g) {
  for (i=0 ; i<NUMDUKES ; i++) {
    if (dukes[i] != null) {
      g.drawImage(Duke.images[dukes[i].getIndex()],
                  200*i, 0, this);
    }
  }
}

public void stop() {
  for (int i=0; i<NUMDUKES ; i++) {
    if (dukes[i] != null) {
      dukes[i].setState(Duke.STOP);
    }
  }
}
```

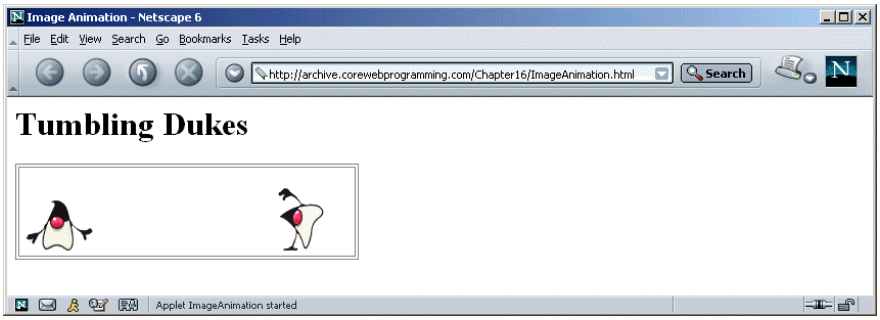

Animation Example (Continued)

```
public class Duke extends Thread {
  ...
  public static Image[] images;
  private static final int NUMIMAGES = 15;
  private static Object lock = new Object();
  private int state = RUN;

  public Duke(int tumbleDirection, Applet parent) {
    this.tumbleDirection = tumbleDirection;
    this.parent = parent;
    synchronized(lock) {
      if (images == null) {  // If not previously loaded.
        images = new Image[ NUMIMAGES ];
        for (int i=0; i<NUMIMAGES; i++) {
          images[i] = parent.getImage( parent.getCodeBase(),
                                       "images/T" + i + ".gif");
        }
      }
    }
  }
  ...
```

Animation Example (Continued)

```
  public void run() {
    while (checkState()!=STOP) {
      index += tumbleDirection;
      if (index < 0) {
        index = NUMIMAGES - 1;
      } else if (index >= NUMIMAGES) {
        index = 0;
      }

      parent.repaint();

      try {
        Thread.sleep(100);
      } catch (InterruptedException e) {
        break;   // Break while loop.
      }
    }
  }
}
```


Animation: Result



Timers

- **Swing defines a `Timer` class**
  - A Timer can ring for a single cycle or fire periodically
  - At each ring an ActionEvent is fired
  - Useful for animations, simulations, and timing out secure network connections
- **Approach**

```
Timer timer = new Timer(milliseconds, listener);
timer.start();
...
...
timer.stop();
```


Useful Timer Methods

- **start/stop**
  - Starts or stops the timing sequence
- **restart**
  - Cancels any undelivered time events and starts the timer again
- **setCoalesce**
  - Turns coalescing off or on
  - By default, if a timer event is in the event queue (coalesce true), a new ActionEvent is not created at the next firing interval
- **setRepeats**
  - Sets the timer to ring once (false) or to ring periodically (true)
  - Default behavior is to ring periodically

Timer: Example

```
import java.awt.*;
import javax.swing.*;

public class TimedAnimation extends JApplet {
  private static final int NUMDUKES = 2;
  private TimedDuke[] dukes;
  private int i, index;

  public void init() {
    dukes = new TimedDuke[NUMDUKES];
    setBackground(Color.white);
    dukes[0] = new TimedDuke( 1, 100, this);
    dukes[1] = new TimedDuke(-1, 500, this);
  }

  //  Start each Duke timer.
  public void start() {
    for (int i=0; i<NUMDUKES ; i++) {
      dukes[i].start();
    }
  } ...
```


Timer Example (Continued)

```
  ...
  public void paint(Graphics g) {
    for (i=0 ; i<NUMDUKES ; i++) {
      if (dukes[i] != null) {
        index = dukes[i].getIndex();
        g.drawImage(TimedDuke.images[index], 200*i, 0, this);
      }
    }
  }

  //  Stop each Duke timer.
  public void stop() {
    for (int i=0; i<NUMDUKES ; i++) {
      dukes[i].stop();
    }
  }
}
```

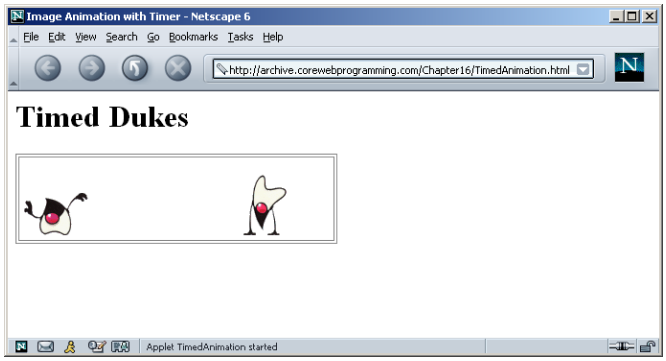
Timer Example (Continued)

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class TimedDuke extends Timer
                       implements ActionListener {
  private static final int NUMIMAGES = 15;
  private static boolean loaded = false;
  private static Object lock = new Object();
  private int tumbleDirection;
  private int index = 0;
  private Applet parent;
  public static Image[] images = new Image[NUMIMAGES];

  public TimedDuke(int tumbleDirection, int msec,
                                        Applet parent) {
    super(msec, null);
    addActionListener(this);
    this.tumbleDirection = tumbleDirection;
    this.parent = parent; ...
```


Timer Example (Continued)

```
    synchronized (lock) {
      if (!loaded) {
        // Load images using MediaTracker
        ...
      }
    }
  }

  // Return current index into image array.
  public int getIndex() { return index; }

  // Receives timer firing event.  Increments the index into
  // image array and forces repainting of the new image.
  public void actionPerformed(ActionEvent event) {
    index += tumbleDirection;
    if (index < 0){
      index = NUMIMAGES - 1;
    }
    if (index >= NUMIMAGES) {
      index = 0;
    }
    parent.repaint();
  }
```

Timer Example: Result



Each Duke moves at a different speed

Summary

- **Options**
  - Redraw everything in paint
  - Have routines other than paint directly do drawing operations
  - Override update and have paint do incremental updating
  - Double buffering
- **Animation can be achieved by cycling through a sequence of images**
- **Timers are restartable threads that fire an ActionEvent once or periodically**

core WEB programming

Questions?