Today's Lecture

(Java) Graphics

1. A Useful Identity

2. System-Level Graphics

   `java.awt`

3. Low-level Graphics
   - Frame Buffers
   - Line Drawing
   - Region Filling

A Useful Identity

Suppose a and b are arbitrary strings of bits (*i.e.* bitmaps) of length n

Then,

$$(a \texttt{ xor } b) \texttt{ xor } b = a$$

Example: Suppose a is a piece of a graphical display (*e.g.* an icon) and b is the bitmap corresponding to the cursor; the above identity can be used to display the cursor at a and to later restore a when the cursor leaves it.

How would you prove the identity?

Can get the same effect by copying; copying is sometimes necessary or desirable

A Useful Identity: An Application


```
/* Scribble.java */
/* modified to demonstrate XORing */

import java.applet.*;
import java.awt.*;

public class Scribble extends Applet {
    private int last_x = 0;
    private int last_y = 0;
    private Color cur_color = Color.black;

    private Button clear_button;
    private Choice color_choices;

    public void init() {
        this.setBackground(Color.white);

        clear_button = new Button("Clear");
        clear_button.setForeground(Color.black);
        clear_button.setBackground(Color.lightGray);
        this.add(clear_button);

        color_choices = new Choice();
        color_choices.addItem("black");
        color_choices.addItem("red");
        color_choices.addItem("yellow");
        color_choices.addItem("green");
        color_choices.setForeground(Color.black);
        color_choices.setBackground(Color.lightGray);
        this.add(new Label("Color"));
        this.add(color_choices);
    }

    /* XOR demo: track a filled circle at the cursor in XOR-mode */
    private int cursor_x = -100;
    private int cursor_y = -100;

    public boolean mouseMove(Event e,int x,int y) {
        Graphics g = this.getGraphics();
        Color c = g.getColor();

        g.setXORMode(Color.orange);
```



```
        g.fillOval(cursor_x-10,cursor_y-10,20,20);
        g.fillOval(x-10,y-10,20,20);
        g.setPaintMode();
        cursor_x = x;
        cursor_y = y;
        return false;
    }

    public boolean mouseDown(Event e,int x,int y) {
        last_x = x; last_y = y;
        return true;
    }

    public boolean mouseDrag(Event e,int x,int y) {
        Graphics g = this.getGraphics();
        g.setColor(cur_color);
        g.drawLine(last_x,last_y,x,y);
        last_x=x;
        last_y=y;
        return true;
    }

    public boolean action(Event e,Object a) {
        if (e.target == clear_button) {
            Graphics g = this.getGraphics();
            Rectangle r = this.bounds();
            g.setColor(this.getBackground());
            g.fillRect(r.x,r.y,r.width,r.height);
            return true;
        }
        else if (e.target == color_choices) {
            if (a.equals("black")) cur_color = Color.black;
            else if (a.equals("red")) cur_color = Color.red;
            else if (a.equals("yellow")) cur_color = Color.yellow;
            else if (a.equals("green")) cur_color = Color.green;
            return true;
        }
        else return super.action(e,a);
    }
}

/* end of Scribble.java */
```

System-Level Graphics: `java.awt`

We'll look at the following pieces of `java.awt`:

- `public class Graphics`
- `public class Scrollbar extends Component`
- `public interface LayoutManager`
- `public class FlowLayout implements LayoutManager`
  (`BorderLayout` is another Java-supplied layout manager)

`java.awt: class Graphics`

Constructors:

- `protected Graphics()`

Methods:

- `public abstract Graphics create()`
- `public Graphics create(int x,int y,int width,int height)`
- `public abstract void translate(int x,int y);`
- `public abstract Color getColor()`
- `public abstract void setColor()`
- `public abstract void setPaintMode()`
- `public abstract void setXORMode(Color c)`
- `public abstract Font getFont()`
- `public abstract void setFont(Font f)`
- `public FontMetrics getFontmetrics()`
- `public abstract Rectangle getClipRect()`
- `public abstract void clipRect(int x,int y,int width,int height)`

- `public abstract void copyArea(int x,int y,int w,int h,int dx, int dy)`
- `public abstract void drawLine(int x1,int y1,int x2,int y2)`
- `public abstract void fillRect(int x,int y,int width,int height)`
- `public abstract void drawRect(int x,int y,int width,int height)`
- `public abstract void clearRect(int x,int y,int width,int height)`
- Similar draw, fill, & clear methods for RoundRects, 3DRects, Ovals, Arcs and Polygons
- `public abstract void drawString(String s,int x,int y)`
- `public abstract void drawString(String s,int x,int y)`
- `public abstract void drawChars(char data[],int off,int len,int x,int y)`
- `public abstract void dispose()`
- `public void finalize()`
- `public String toString()`

`java.awt: class Scrollbar`

`public class Scrollbar extends Component`

Variables:

- `HORIZONTAL`
- `VERTICAL`

Constructors:

- `public Scrollbar()`
- `public Scrollbar(int orientation)`
- `public Scrollbar(int o,int value,int visible,int min,int max)`

Methods:

- `public int getOrientation()`
- `public int getValue()`
- `public void setValue(int value)`
- `public int getMinimum()`
- `public int getMaximum()`
- `public int getVisible()`
- `public int setLineIncrement(int i)`
- `public void getLineIncrement()`
- `public int setPageIncrement(int i)`
- `public void getPageIncrement()`

Methods:

- `public abstract void addLayoutComponent(String name, Component comp);`
- `public abstract void removeLayoutComponent(Component comp);`
- `public abstract Dimension preferredLayoutSize(Container parent);`
- `public abstract Dimension minimumLayoutSize(Container parent);`
- `public abstract void LayoutContainer(Container parent);`

`public class FlowLayout implements LayoutManager`

Variables:

- `CENTER`
- `LEFT`
- `RIGHT`

Constructors:

- `public FlowLayout()`
- `public FlowLayout(int align)`
- `public FlowLayout(int align, int hgap, int vgap)`

Methods:

- `public void addLayoutComponent(String name, Component comp);`
- `public void removeLayoutComponent(Component comp);`
- `public Dimension preferredLayoutSize(Container parent);`
- `public Dimension minimumLayoutSize(Container parent);`
- `public void LayoutContainer(Container parent);`
- `public String toString();`

Low-Level Graphics

Very basic algorithms for:

1. managing a frame buffer
2. drawing a line
3. filling a region (seed fill)
   - simple recursive fill
   - better scan-based fill

and their Java implementation

Frame Buffer

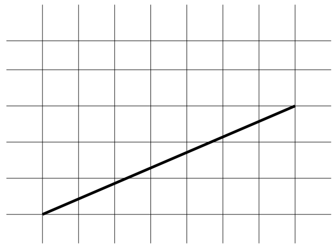
This Java class mimics common operations on a frame buffer:

```
/* FrameBuffer.java */

import java.awt.Dimension;
import java.awt.Point;

public class FrameBuffer {
    private Dimension d;
    private int buffer[][];

    public FrameBuffer(Dimension dRequest) {
        d = dRequest;
        /* origin: upper left-hand corner */
        buffer = new int[d.height][d.width];
    }

    public Dimension size() {
        return d;
    }

    final int REPLACE_MODE = 0;
    final int XOR_MODE = 1;
    private int mode = REPLACE_MODE;

    public void setMode(int m) {
        mode = m;
    }

    private int color = 0;
    public void setColor(int c) {
        color = c;
    }

    public int getPixel(Point p) {
        return buffer[p.y][p.x];
    }

    public void setPixel(Point p) {
        switch (mode) {
          case REPLACE_MODE:
```

```
            buffer[p.y][p.x] = color;
            break;
          case XOR_MODE:
            buffer[p.y][p.x] ^= color;
            break;
          /** handle yet-to-be-defined modes here **/
          default:
            /** handle unknown mode here **/
        }
    }

    public void dump() {
        for (int row = 0; row < d.height; row++) {
            for (int col = 0; col < d.width; col++) {
                System.out.print(buffer[row][col]+" ");
            }
            System.out.println();
        }
    }
}

/* end of FrameBuffer.java */
```


Bresenham's Algorithm

Idea: draw a line between two integer coordinates without doing any expensive math calculations.



Note: pixels are at grid intersections

Bresenham's Algorithm (cont'd)

Recall two equations for a line:

$$ax + by + c = 0 \tag{1}$$

$$y = \frac{\Delta y}{\Delta x}x + C \tag{2}$$

Combining (1) and (2) gives a function F:

$$F(x, y) = x\Delta y - y\Delta x + C\Delta x \tag{3}$$

with the properties

$$F(x, y) > 0 \Rightarrow (x, y) \text{ is above the line}$$

$$F(x, y) < 0 \Rightarrow (x, y) \text{ is below the line}$$

$$F(x, y) = 0 \Rightarrow (x, y) \text{ is on the line}$$

We can use F to decide where to place the next pixel

Bresenham's Algorithm (cont'd)

Suppose we know we placed a pixel at (x_0, y_0)

We can then *incrementally* compute $F(x_0 + 1, y_0 + \frac{1}{2})$ from $F(x_0, y_0)$ by adding

$$\begin{aligned} F(x_0+1, y_0+\frac{1}{2}) - F(x_0, y_0) &= \Delta y(x_0+1) - \Delta x(y_0+\frac{1}{2}) + C\Delta x - \\ &\quad (\Delta y x_0 - \Delta x y_0 + C\Delta x) \qquad (4) \\ &= \Delta y - \frac{\Delta x}{2} \qquad (5) \end{aligned}$$

to $F(x_0, y_0)$ giving

$$F(x_0+1, y_0+\frac{1}{2}) = F(x_0, y_0) + \Delta y - \frac{\Delta x}{2} \tag{6}$$

If equation (6) < 0, we place a pixel at $(x_0 + 1, y_0)$, otherwise we place a pixel at $(x_0 + 1, y_0 + 1)$

Bresenham's Algorithm (cont'd)

Furthermore, note that we can compute $F(x_0+1, y_0)$ from $F(x_0, y_0)$ by adding

$$\begin{aligned} F(x_0+1,y_0) - F(x_0,y_0) &= \Delta y(x_0+1) - \Delta x y_0 + C\Delta x - \\ (\Delta y x_0 - \Delta x y_0 + C\Delta x) & \qquad (7) \\ &= \Delta y \qquad (8) \end{aligned}$$

to $F(x_0, y_0)$

Similarly, we can compute $F(x_0+1, y_0+1)$ from $F(x_0, y_0)$ by adding

$$\begin{aligned} F(x_0+1,y_0+1) - F(x_0,y_0) &= \Delta y(x_0+1) - \Delta x(y_0+1) + C\Delta x - \\ (\Delta y x_0 - \Delta x y_0 + C\Delta x) & \qquad (9) \\ &= \Delta y - \Delta x \qquad (10) \end{aligned}$$

to $F(x_0, y_0)$

The incremental value of F at each x value is a *decision variable* and often denoted d in the algorithm.

Bresenham's Algorithm in Java


```
/* Bresenham.java */

import java.awt.Point;
import FrameBuffer;

public class Bresenham {
    private FrameBuffer fb;

    public Bresenham(FrameBuffer frameBuf) {
        fb = frameBuf;
    }

    /* draw a line from point p1 to point p2 using Bresenham's algorithm */
    /* assumes p1 and p2 lie in FrameBuffer fb */
    public void line(Point p1, Point p2) {
        int dx = p2.x - p1.x;
        int ax = Math.abs(dx)*2;
        int sx = sign(dx);
        int dy = p2.y - p1.y;
        int ay = Math.abs(dy)*2;
        int sy = sign(dy);

        int x = p1.x;
        int y = p1.y;
        int d;

        if (ax > ay) {
            d = ay - ax/2;
            while (true) {
                fb.setPixel(new Point(x, y));
                if (x == p2.x)
                    return;
                if (d >= 0) {
                    y += sy;
                    d -= ax;
                }
                x += sx;
                d += ay;
            }
        } else {
            d = ax - ay/2;
            while (true) {
```



```
                fb.setPixel(new Point(x, y));
                if (y == p2.y)
                    return;
                if (d >= 0) {
                    x += sx;
                    d -= ay;
                }
                y += sy;
                d += ax;
            }
        }
    }

    private int sign(int i) {
        return i > 0 ? 1 : -1;
    }
}

/* end of Bresenham.java */
```


Seed Fills

"Paint programs" often need to fill an arbitrary region of the frame buffer

A common fill takes a seed point (pixel) and a fill value (color) and proceeds to color all transitively *connected* points that are not of the fill value with the fill value

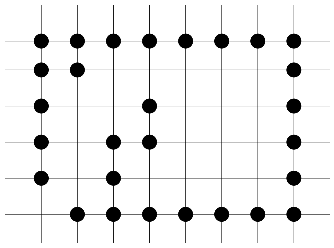
Four points *connected* to point (x, y):

$$(x, y+1),\ (x, y-1),\ (x-1, y),\ (x+1, y)$$

We'll look at two seed fills for arbitrary regions:

1. Recursive
2. Scan-Based

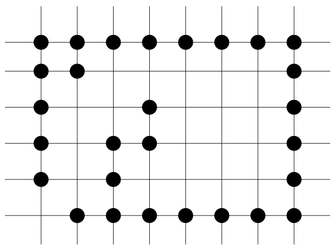
Recursive Seed Fills



Recursive Seed Fill in Java


```
/* SimpleFill.java */

import FrameBuffer;
import java.awt.Point;
import java.util.Stack;

public class SimpleFill {
    private FrameBuffer fb;

    private int fbMaxX, fbMaxY;

    public SimpleFill(FrameBuffer frameBuf) {
        fb = frameBuf;
        fbMaxX = fb.size().width;
        fbMaxY = fb.size().height;
    }

    public void fill(Point seedPt, int fillVal) {
        int x = seedPt.x;
        int y = seedPt.y;

        if (x >= fbMaxX || x < 0 || y >= fbMaxY || y < 0)
            return;
        if (fb.getPixel(seedPt) != fillVal) {
            fb.setPixel(seedPt);
            fill(new Point(x, y+1), fillVal);
            fill(new Point(x, y-1), fillVal);
            fill(new Point(x+1, y), fillVal);
            fill(new Point(x-1, y), fillVal);
        }
    }
}

/* end of SimpleFill.java */
```


Scan-based Seed Fill



Scan-Based Seed Fill in Java


```
/* ScanFill.java */

import FrameBuffer;
import java.awt.Point;
import java.util.Stack;

public class ScanFill {
    private FrameBuffer fb;
    private Stack s = new Stack();

    private int fbMaxX, fbMaxY;

    public ScanFill(FrameBuffer frameBuf) {
        fb = frameBuf;
        fbMaxX = fb.size().width;
        fbMaxY = fb.size().height;
    }

    private void scanLeft(Point p, int fillVal) {
        int x = p.x;
        int y = p.y;
        boolean inTop = false;
        boolean inBottom = false;

        while (x >= 0 && fb.getPixel(p) != fillVal) {
            fb.setPixel(p);
            /* check the line above for scan segments */
            if (y+1 < fbMaxY) {
                Point q = new Point(x, y+1);
                if (!inTop) {
                    if (fb.getPixel(q) != fillVal) {
                        /* found a new segment; push it on the stack */
                        s.push(q);
                        inTop = true;
                    }
                } else if (fb.getPixel(q) == fillVal) {
                    inTop = false;
                }
            }
            /* check the line below for scan segments */
            if (y-1 >= 0) {
```

```
                Point q = new Point(x, y-1);
                if (!inBottom) {
                    if (fb.getPixel(q) != fillVal) {
                        /* found a new segment; push it on the stack */
                        s.push(q);
                        inBottom = true;
                    }
                } else if (fb.getPixel(q) == fillVal) {
                    inBottom = false;
                }
            }
            /* move left */
            p = new Point(--x,y);
        }
    }

    private void scanRight(Point p, int fillVal) {
        int x = p.x;
        int y = p.y;
        boolean inTop = false;
        boolean inBottom = false;

        while (x <= fbMaxX && fb.getPixel(p) != fillVal) {
            /* fill the current pixel */
            fb.setPixel(p);

            /* check the line above for scan segments */
            if (y+1 < fbMaxY) {
                Point q = new Point(x, y+1);
                if (inTop) {
                    if (fb.getPixel(q) == fillVal) {
                        /* found a new segment; push it on the stack */
                        s.push(new Point(x-1, y+1));
                        inTop = false;
                    }
                } else if (fb.getPixel(q) == fillVal) {
                    inTop = true;
                }
            }
            /* check the line below for scan segments */
            if (y-1 >= 0) {
                Point q = new Point(x, y-1);
                if (inBottom) {
                    if (fb.getPixel(q) == fillVal) {
                        /* found a new segment; push it on the stack */
```



```
                        s.push(new Point(x-1, y-1));
                        inBottom = false;
                    }
                } else if (fb.getPixel(q) == fillVal) {
                    inBottom = true;
                }
            }
            /* move right */
            p = new Point(++x,y);
        }
    }

    public void fill(Point seedPt, int fillVal) {
        s.push(seedPt);
        while (!s.empty()) {
            Point p = (Point) s.pop();

            scanLeft(p, fillVal);
            scanRight(new Point(p.x+1,p.y), fillVal);
        }
    }
}

/* end of ScanFill.java */
```


A Driver for Our Primitives


```
/* GraphicsDriver.java */

import java.awt.Point;
import java.awt.Dimension;

import FrameBuffer;
import Bresenham;
import ScanFill;
import SimpleFill;

public class GraphicsDriver {
    private static FrameBuffer fb = new FrameBuffer(new Dimension(10, 10));
    private static Bresenham bh = new Bresenham(fb);
    private static ScanFill sf = new ScanFill(fb);
    private static SimpleFill simple = new SimpleFill(fb);

    public static void main(String args[]) {
        fb.setMode(fb.REPLACE_MODE);
        fb.setColor(1);
        System.out.println("drawing some lines:");
        bh.line(new Point(0, 0), new Point(9, 9));
        bh.line(new Point(0, 9), new Point(9, 0));
        fb.dump();
        System.out.println("slow fill:");
        simple.fill(new Point(5, 0), 1);
        fb.dump();
        System.out.println("faster fill:");
        sf.fill(new Point(0,5), 1);
        fb.dump();
    }
}

/* end of GraphicsDriver.java */
```


The Driver's Output


```
$ java GraphicsDriver
drawing some lines:
1 0 0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0 1 0
0 0 1 0 0 0 0 1 0 0
0 0 0 1 0 0 1 0 0 0
0 0 0 0 1 1 0 0 0 0
0 0 0 0 1 1 0 0 0 0
0 0 0 1 0 0 1 0 0 0
0 0 1 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0 0 1
slow fill:
1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 0
0 0 1 1 1 1 1 1 0 0
0 0 0 1 1 1 1 0 0 0
0 0 0 0 1 1 0 0 0 0
0 0 0 0 1 1 0 0 0 0
0 0 0 1 0 0 1 0 0 0
0 0 1 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0 0 1
faster fill:
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1 0 0
1 1 1 1 1 1 1 0 0 0
1 1 1 1 1 1 0 0 0 0
1 1 1 1 1 1 0 0 0 0
1 1 1 1 0 0 1 0 0 0
1 1 1 0 0 0 0 1 0 0
1 1 0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0 0 1
$
```

Next Lecture

Guest: Dave MacQueen, Bell Labs

ML (Meta Language)