# Pixels, Rasters, Sprites, and BitBlts

## Experiencing **JAVA**phobia?

- Exercise #1
- A Graphical Hello World
- Rasters
- Pixels
- Sprites
- BitBlts







# Exercise #1

Set up your course homepage by next Tuesday 9/19

The Brain's 6.837 Homepage

MIT's Graphics Group
Avalon
My *REAL* homepage
6.837's homepage

*Narf!*

Homepage Requirements:

- Locate on imagery in `/mit/imagery2/6.837/F00/username` (you must execute "**add imageory2**" first)
- Something about you
- Links to all 5 projects
- Area for links to computer graphics sites that you find interesting
- A link back to the 6.837 homepage







# A First Java Program

We start by writing what is called a Java *applet*. This is a form of a Java program that can be embedded in a HTML document and accessed from the web. An example HTML document that calls our example applet is shown below:

```
<HTML>
<HEAD>
<TITLE>Demo Applet</TITLE>
</HEAD>
<BODY>

<H1>Demo Applet</H1>
<P>My favorite class is 6.837</P>
<HR>
<CENTER>
<APPLET code="Demo.class" width=200 height=200>
</APPLET>
</CENTER>
<HR>
</BODY>
</HTML>
```

The highlighted lines add the Java code to our document. Notice the *.class* extension. All Java source code files should end with a *.java* extension. The source is here.







# Java Example

Next, the source of the *demo.java* file is shown.

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

public class Demo extends Applet {
        Image image;
        int count;

        public void init() {
            image = getImage(getDocumentBase(), "World.jpg");
            count = 1;
        }

        public void paint(Graphics g) {
            g.drawImage(image, 0, 0, this);
            g.setColor(Color.red);
            for (int y = 15; y < size().height; y += 15) {
                int x = (int) (size().width/2 + 30*Math.cos(Math.PI*y/75));
                g.drawString("Hello", x, y);
            }
            showStatus("Paint called "+count+" time"+((count > 1)?"s":""));
            count += 1;
        }
}
```

You can get the source here and the World.jpg image here.

# The Applet in Action

**Demo Applet**

My favorite class is 6.837

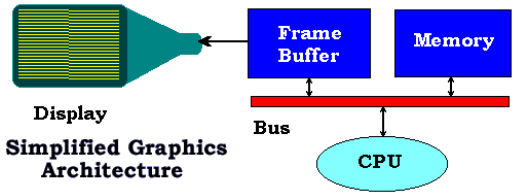
This is all of the Java programming language that we will specifically cover in class.

If you have never used Java before, you might want to consider buying one of the books discussed on the course's home page.





# Review of Raster Displays

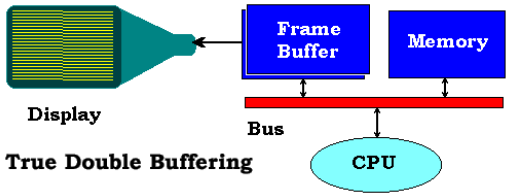



- Display synchronized with CRT sweep
  - Special memory for screen update
- Pixels are the discrete elements displayed
  - Generally, updates are visible





# High-End Graphics Display System

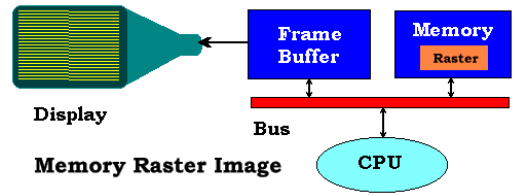



- Adds a second frame buffer
- Swaps during vertical blanking
  - Updates are invisible
    - Costly





# A Memory Raster




- Maintains a copy of the screen (or some part of it) in memory
  - Relies on a fast copy
  - Updates are *nearly* invisible
- Conceptual model of a physical object

# A Java Model of a Memory Raster

```
class Raster implements ImageObserver {
    ////////////////////////// Constructors ////////////////////
    public Raster();               // allows class to be extended
    public Raster(int w, int h);    // specify size
    public Raster(Image img);       // set to size and contents of image

    ////////////////////////// Interface Method /////////////////
    public boolean imageUpdate(Image img, int flags, int x, int y, int w, int h);

    ////////////////////////// Accessors ///////////////////////////
    public int getSize( );          // pixels in raster
    public int getWidth( );         // width of raster
    public int getHeight( );        // height of raster
    public int[] getPixelBuffer( ); // get array of pixels

    ////////////////////////// Methods ////////////////////////////
    public void fill(int argb);     // fill with packed argb
    public void fill(Color c);          // fill with Java color
    public Image toImage(Component root);
    public int getPixel(int x, int y);
    public Color getColor(int x, int y);
    public boolean setPixel(int pix, int x, int y);
    public boolean setColor(Color c, int x, int y);
}
```

Download Raster.java here.







# Example Usage: Rastest.java

The code on the right demonstrates the use of a Raster object. The running Applet is shown below. Clicking on the image will cause it to be negated.

The source code for this applet can be downloaded here: Rastest.java.

```
import java.applet.*;
import java.awt.*;
import Raster;

public class Rastest extends Applet {
  Raster raster;
  Image output;
  int count = 0;

  public void init() {
    String filename = getParameter("image");
    output = getImage(getDocumentBase(), filename);
    raster = new Raster(output);
    showStatus("Image size: " + raster.getWidth() + " x " +
raster.getHeight());
  }

  public void paint(Graphics g) {
    g.drawImage(output, 0, 0, this);
    count += 1;
    showStatus("paint() called " + count + " time" + ((count > 1) ? "s":""));
  }

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

  public boolean mouseUp(Event e, int x, int y) {
    int s = raster.getSize();
    int [] pixel = raster.getPixelBuffer();
    for (int i = 0; i < s; i++) {
      raster.pixel[i] ^= 0x00ffffff;
    }
    output = raster.toImage(this);
    repaint();
    return true;
  }
}
```





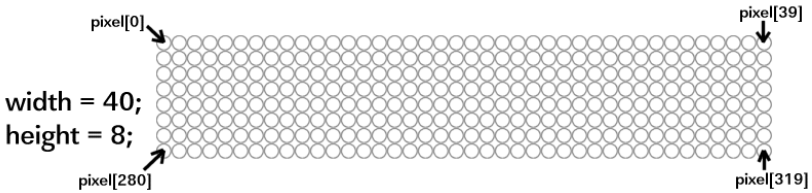


# Lets Talk About Pixels

- Pixels are stored as a 1-dimensional array of *int*s
- Each *int* is formatted according to Java's standard pixel model


**The 4 bytes of a 32-bit *Pixel* int.**
**if Alpha is 0 the pixel is transparent.**
**if Alpha is 255 the pixel is opaque.**

- Layout of the pixel array on the display:



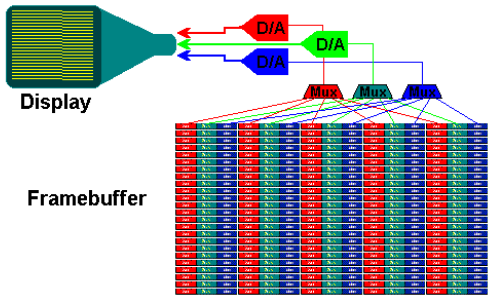

- This is the image format used internally by Java





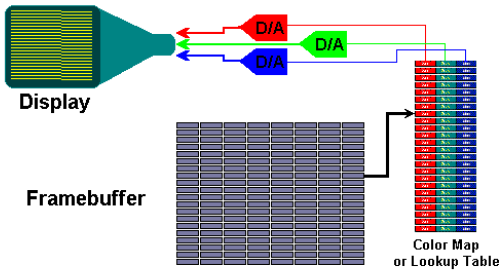


# True-Color Frame Buffers




- Each pixel requires at least 3 bytes. One byte for each primary color.
- Sometimes combined with a look-up table per primary
- Each pixel can be one of 2^24 colors
- Worry about your *Endians*

# Indexed-Color Frame Buffers




- Each pixel uses one byte
- Each byte is an index into a color map
- If the color map is not updated synchronously then *Color-map flashing* may occcur.
- Color-map Animations
- Each pixel may be one of 2^24 colors, but only 256 color be displayed at a time

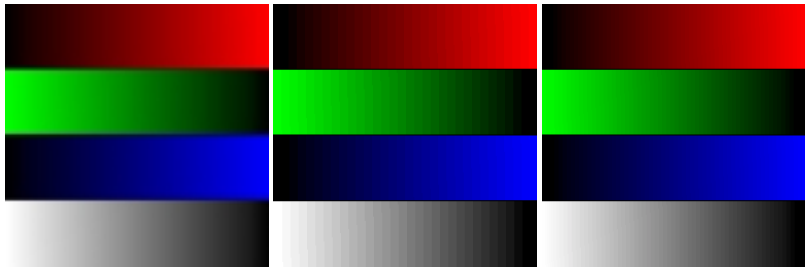




# High-Color Frame Buffers


**Pixels are packed in a short.**
**Each primary uses 5 bits.**

- Popular *PC/(SVGA)* standard (popular with Gamers)
- Each pixel can be one of 2^15 colors
- Can exhibit worse quantization (banding) effects than Indexed-color







# Sprites

Sprites are rasters that can be overlaid onto a background raster called a playfield.

A sprite can be animated, and it generally can be repositioned freely any where within the playfield.





# A Sprite is a Raster

```
class Sprite extends Raster {
    int x, y;                          //
position of sprite on playfield
    public void Draw(Raster bgnd);  //
draws sprite on a Raster
}
```

Things to consider:


**The Draw( ) method must handle transparent pixels, and it must also handle all cases where the sprite overhangs the playfield.**

# An Animated Sprite is a Sprite

```
class AnimatedSprite extends Sprite {
    int frames;         // frames in sprite
                        // there are other private variables

    public AnimatedSprite(Image images, int frames);
    public void addState(int track, int frame, int ticks, int dx, int dy);
    public void Draw(Raster bgnd);
    public void nextState();
    public void setTrack();
}
```

**A track is a series of frames. The *frame* is displayed for *ticks* periods. The sprite's position is then moved (*dx, dy*) pixels.**

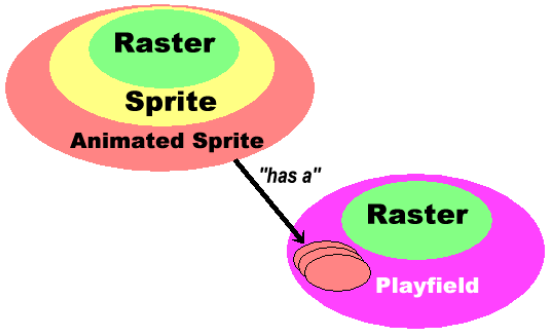




# A Playfield is a Raster and has Animated Sprites

```
class Playfield extends Raster {
    Raster background;          // background image
    AnimatedSprite sprite[];    // list of sprites on this playfield

    public Playfield(Image bgnd, int numSprites);
    public void addSprite(int i, AnimatedSprite s);
    public void Draw( );
}
```








# Other Image Formats

What if we want to read in some other image format?

*.bmp, .ppm, .tif, .tga, .rgb, .ras, .psd, ...*

We must implement an imageProducer to read pixels, an imageConsumer to make the image, and keep imageObservers updated.

Where?
How?





# Luckily, We Already Have

From our Raster class:

```
    public final Image toImage(Component root) {
        return root.createImage(new MemoryImageSource(width, height, pixel, 0,
width));
    }
```

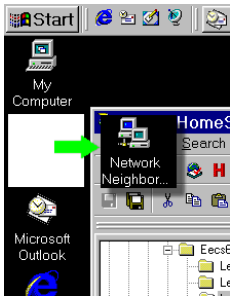
The `MemoryImageSource` method is an imageProducer (`root` is an imageObserver).

```
    public class ppmDecoder {
        Raster imgRaster;       // declare a Raster

        public ppmDecoder() { }
        public Image getppmImage(URL url) {
            // open url and read image header
                .
                .
                .
            // create imgRaster
                .
                .
                .
            // copy data from file into imgRaster
                .
                .
                .
            return (imgRaster.toImage());
        }
    }
```

# PixBlts

PixBlts are raster methods for moving and clearing sub-blocks of pixels from one region of a raster to another




Very heavily used by window systems:

- moving windows around
- scrolling text
- copying and clearing







# Seems Easy

Here's a PixBlt method:

```
    public void PixBlt(int xs, int ys, int w, int
h, int xd, int yd)
    {
        for (int j = 0; j < h; j++) {
            for (int i = 0; i < w; i++) {

this.setPixel(raster.getPixel(xs+i, ys+j), xd+i,
yd+j);
            }
        }
    }
```

But does this work?
What are the issues?
How do you fix it?







# The Tricky Blits


Our PixBlt Method works fine when the source and destination regions do not overlap. But, when these two regions do overlap we need to take special care to avoid copying the wrong pixels. The best way to handle this situation is by changing the iteration direction of the copy loops to avoid problems.


The iteration direction must always be *opposite* of the direction of the block's movement for each axis. You could write a fancy loop which handles all four cases. However, this code is usually so critical to system performace that, generally, code is written for all 4 cases and called based on a test of the source and destination origins. In fact the code is usually unrolled.







# Graphics System Architecture

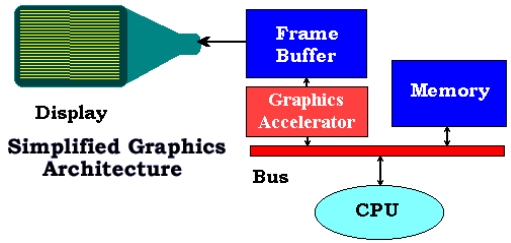
Computer Graphics is one of the few computer system functions where ***specialized H/W*** is still commonly used.

Why?

**The numbers:**
1M pixels * 60 f/s * 1 op/(pixel frame) = 60M ops/s

**Wouldn't you like to compute while your screen scrolls?**

**Isn't graphics why you have a computer?**

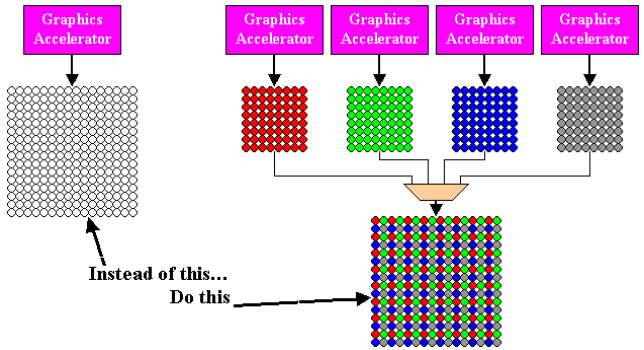



# The Problem is Bandwidth

Graphics is one of the most bandwidth intensive tasks that a computer does. For graphics, caching is not the effective solution that it is for traditional computing. Since we are always operating at or near the maximum capabilities of the current processing technology, graphics relys on architectural innovations to achieve the required performance. The most popular solution is parallelism.

Lets discuss how parallelism is exploited in graphics architectures.

**Pixel Interleaving:**







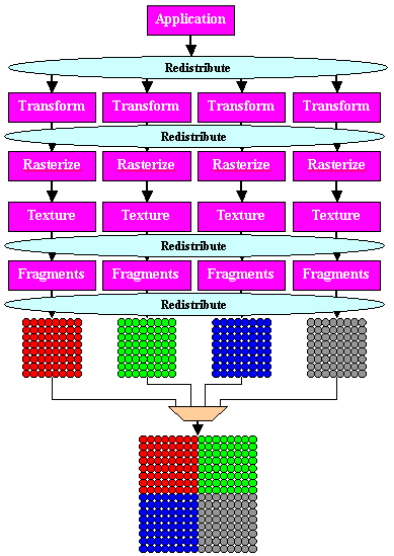

# Finer Grain Parallelism

In the absence of pixel-level interleaving, we need to constantly shuffle primitives in order to achieve a good load balance. There are many possible options:

- Sort-First
- Sort-Middle
- Sort-Last

Lately, new architures have been suggested...
Sort-Everywhere

Eldridge, Ighey, and Hanrahan,
**Pomegranate: A Fully Scalable Graphics Architecture**, SIGGRAPH '00








# Next Time

How do we see?

How do we percieve color?

Why do we only need red, green, and blue channels?

Is gamma greek to you?

What is a Just-noticable diffference anyway?