CS324e - Elements of Graphics and Visualization

Images

Images

- Treated as just another Graphic Primitive in Java 2D
- `Image` class in Java library
- Hold the contents of an actual image file
- OR can be drawn on like a panel

Image Formats

- Image files store the colors of each pixel
- Other information stored such as
  - dimensions
  - colors
- Popular image file formats:
  - GIF, JPEG, PNG, BMP, TIFF, and many more

Image Files

- Just numbers

Sunflower_as_gif_small.gif - Hex

```
     0: 47 49 46 38 39 61 FA 00  29 01 E7 FF 00 06 09 0D  GIF89aú.).çÿ....
    10: 0F 0A 08 19 0D 0C 20 0E  0A 2C 12 0B 22 18 17 1F  ...... ..,.."...
    20: 1A 16 22 1B 12 26 18 18  26 21 17 25 21 1B 2A 20  ..".."&..&!.%!.* 
    30: 1D 2C 20 18 2A 23 14 45  19 0A 33 23 27 36 24 21  .,  .*#.E..3#'6$!
    40: 3A 24 18 32 2B 19 32 2A  22 38 29 1A 31 2A 2A 3B  :$.2+.2*"8).1**;
    50: 34 20 62 27 09 41 31 30  43 32 27 3E 38 30 2E 45  4 b'.A10C2'>80.E
    60: 1F 2F 49 1B 37 48 1A 32  4C 18 46 3F 29 45 44 1A  ./I.7H.2L.F?)ED.
    70: 4B 3B 3A 4D 3C 32 36 4E  14 4D 41 1C 76 35 07 36  K;:M<26N.MA.v5.6
    80: 55 21 39 53 24 3C 53 1F  38 56 1A 41 52 22 5B 45  U!9S$<S.8V.AR"[E
    90: 2B 52 48 36 58 46 39 56  46 3F 41 58 1D 40 56 2F  +RH6XF9VF?AX.@V/
    A0: 47 56 1D 3F 5C 1A 3E 5C  20 47 5A 19 40 5B 27 54  GV.?\.>\ GZ.@['T
    B0: 53 1E 59 4E 33 58 4A 4D  5E 4B 38 43 58 3E 50 53  S.YN3XJM^K8CX>PS
    C0: 37 53 54 2C 56 4E 44 5C  50 2D 4A 5A 2A 87 45 05  7ST,VND\P-JZ*‡E.
    D0: 5C 52 40 7C 4D 08 60 50  48 48 67 27 4A 66 30 51  \R@|M.`PHHg'Jf0Q
    E0: 64 28 5A 63 1C 4A 66 38  4F 69 1D 52 64 35 5B 62  d(Zc.Jf8Oi.Rd5[b
    F0: 31 59 63 39 60 61 3A 5B  62 44 67 5E 3A 67 5E 42  1Yc9`a:[bDg^:g^B
   100: 68 5C 4A 6E 5A 48 64 5F  49 67 5C 51 68 5C 59 6E  h\JnZHd_Ig\Qh\Yn
   110: 62 2D 54 69 4F 55 6A 47  62 68 2F 52 71 31 97 58  b-TiOUjGbh/Rq1—X
   120: 02 55 71 3A 5B 69 56 54  71 41 5C 70 33 5E 6F 3B  .Uq:[iVTqA\p3^o;
   130: 66 6C 3A 5C 70 43 76 6B  20 63 6E 44 8A 67 07 6A  fl:\pCvk cnDŠg.j
   140: 6C 44 66 6D 4E 72 69 44  75 66 4C 70 6A 4C 6E 6A  lDfmNriDufLpjLnj
   150: 54 54 7B 41 79 65 53 70  67 61 70 68 5C 70 66 67  TT{AyeSpgaph\pfg
   160: A0 61 00 74 68 54 6B 75  2A 62 7A 2D 63 7B 3B A9   a.thTku*bz-c{;©
   170: 64 01 61 7C 45 A5 68 00  60 7D 4C 66 7A 4D 69 7B  d.a|E¥h.`}LfzMi{
   180: 45 66 7A 58 6F 79 4E 74  79 46 6A 79 65 71 78 59  EfzXoyNtyFjyeqxY
   190: 78 74 5D 7E 71 5D 6F 7E  42 77 78 50 7D 75 50 7B  xt]~q]o~BwxP}uP{
   1A0: 75 57 79 72 6D 7A 73 66  80 76 47 82 72 57 84 70  uWyrmzsf€vG‚rW„p
   1B0: 5E AC 6F 00 79 7E 44 6C  87 3B AA 74 00 8B 74 55  ^¬o.y~Dl‡;ªt.‹tU
   1C0: 6C 87 46 69 87 57 6F 86  4F 89 7B 45 B3 75 00 72  l‡Fi‡Wo†O‰{E³u.r
   1D0: 86 57 70 85 62 A3 7C 0A  7C 84 50 74 83 6E 7B 84  †Wp…b£|.|„Pt‚n{„
   1E0: 56 7A 84 5B 7C 82 63 84  82 50 80 82 5B 82 82 56  Vz„[|‚c„‚P€‚[‚‚V
   1F0: 85 80 61 87 80 5C 89 80  57 B3 7B 00 8C 80 51 84  …€a‡€\‰€W³{.Œ€Q„
   200: 80 67 8F 7D 5A 90 7B 64  8C 7D 65 B0 7E 00 95 7D  €g}Z{dŒ}e°~.•}
   210: 5C BB 7C 00 AE 83 00 B5  82 00 B9 80 00 77 92 5C  \»|.®ƒ.µ‚.¹€.w'\
   220: A0 87 29 7D 91 54 7E 90  5C 99 88 42 A7 89 18 88   ‡)}'T~\™ˆB§‰.ˆ
   230: 8E 54 7C 8F 74 86 8E 60  84 8E 67 87 8F 5A 8D 8C  ŽT|t†Ž`„Žg‡ZŒ
   240: 5E 91 8A 5E 8B 8C 65 9B  86 5A 96 88 5E 94 8B 54  ^'Š^‹Œe›†Z–ˆ^”‹T
   250: BB 87 00 9F 8B 37 BF 85  00 98 86 68 83 95 51 89  »‡.Ÿ‹7¿….˜†hƒ•Q‰
   260: 8C 72 90 8B 65 90 89 70  8C 8C 6B B8 8A 00 94 89  Œr‹ep‰kŒk¸Š.”‰
   270: 68 7C 95 6A B0 8A 1C B5  8D 02 96 88 74 9A 8E 52  h|•j°Š.µ.–ˆtšŽR
```

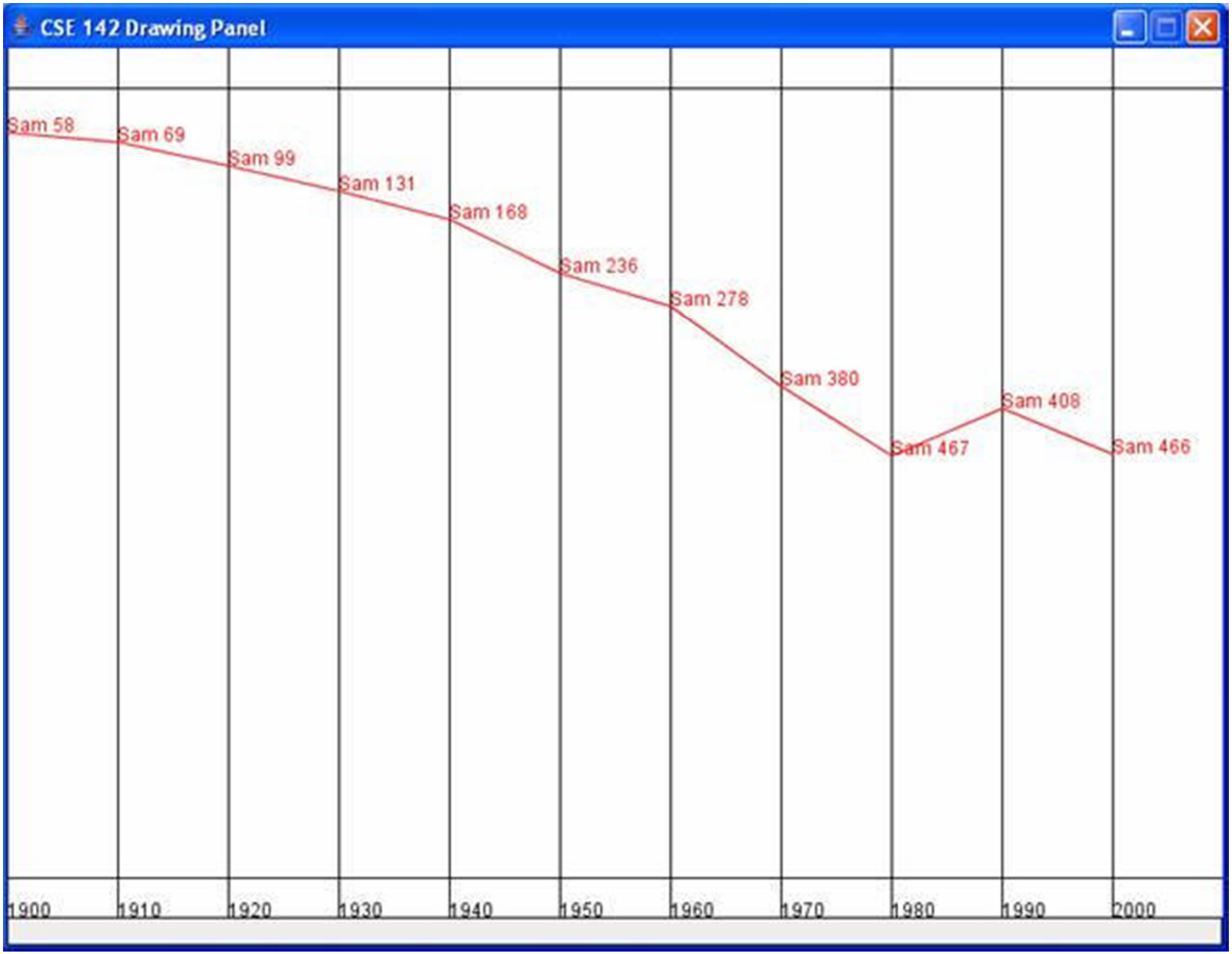
GIF

- Graphics Interchange Format
- 8 bits per pixel for color
- 256 colors
- file stores color palette or table
  - chose from 2^{24} colors
- One of the "colors" can be labeled transparent
  - displayed as white and gray grid in most image editors

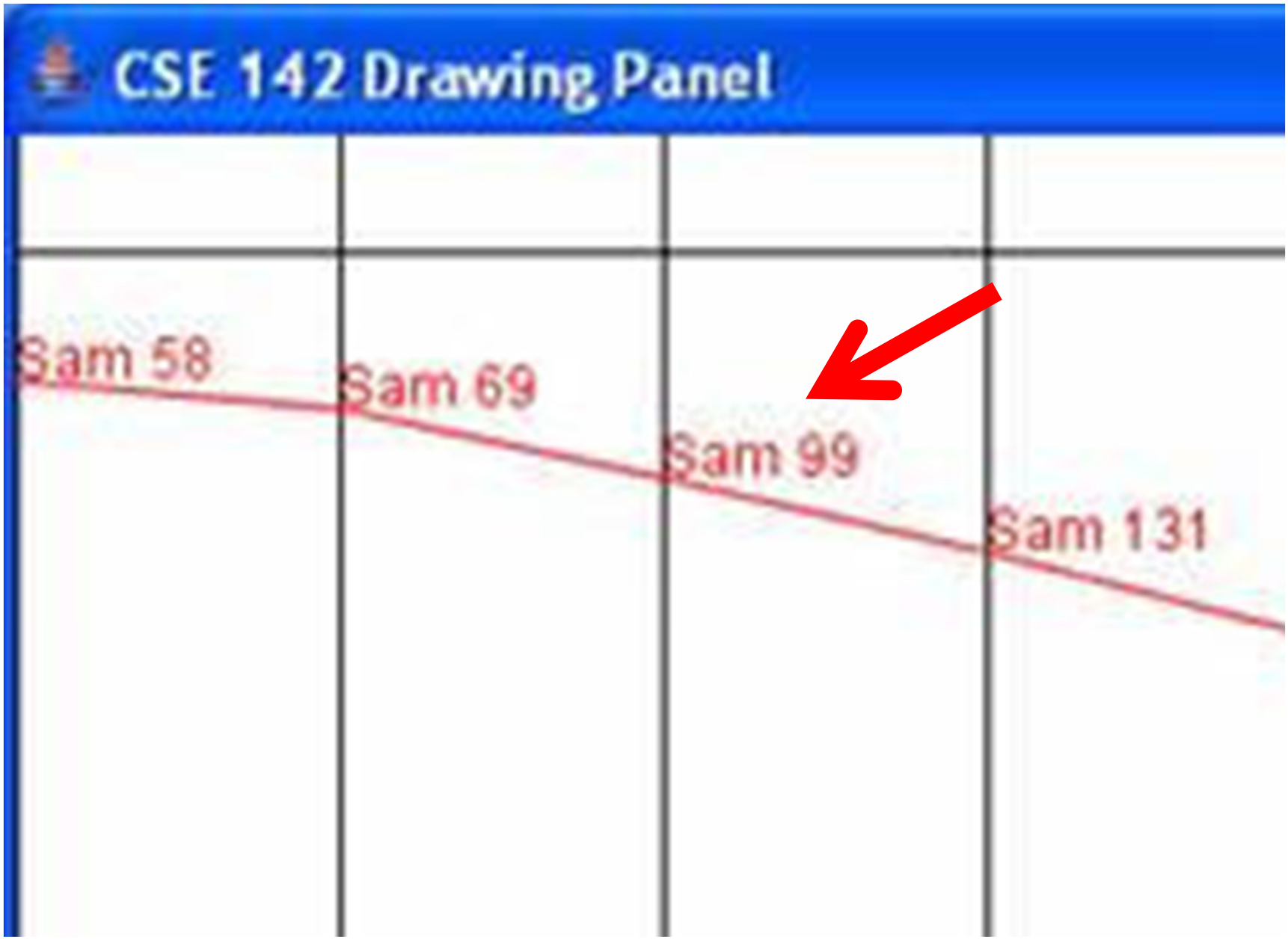
JPEG

- Joint Photographic Experts Group
  - most common file format for digital cameras and other image cpature devices
- Multiple color spaces possible
- typically RGB model with 1 byte per pixel per channel
  - 2^{24} = 16,777,216 colors
- JPEG files are typically compressed to save space
- compression is *lossy* meaning the uncompressed version is not guaranteed to match the original
  - some details lost
- No transparent pixels

JPEG Compression "Artifacts"



JPEG Compression "Artifacts"



PNG

- Portable Network Graphics
- RGB color spaces
- Typically 8 bits per channel plus an 8 bit alpha channel
  - transparent pixels possible
  - other resolutions possible
- 32 bits per pixels
- uses lossless compression

Images in Java

- Image: abstract class. super class to other image classes
- VolatileImage: designed for use with hardware acceleration and video memory storage
  - not used in our course
- BufferedImage: represents a rectangular image in memory
  - contains a color model and *raster*
  - workhorse for our class

Loading Images

- Old way:

```
public void readImage(File f){
    Toolkit tk = Toolkit.getDefaultToolkit();
    Image img = tk.getImage(f.getPath());
    MediaTracker tracker = new MediaTracker(new Component() {});
    tracker.addImage(img, 0);
    try {
      tracker.waitForID(0);
    } catch (InterruptedException ex) {}
    BufferedImage picture;
    picture = new BufferedImage(img.getWidth(this),
            img.getHeight(this), BufferedImage.TYPE_INT_RGB);

    picture.getGraphics().drawImage(img, 0, 0, this);
```

Loading Images

- new way:

```
private BufferedImage picture;

private void loadImage() {
    picture = null;
    try {
        picture = ImageIO.read(new File("peloton.jpg"));
    } catch (IOException e) {
        // what happens if file not there?
        System.out.println("Unable to load image: " + e);
    }
}
```

- path to file must be known

Loading Images from the Web

- Load from url on web:

```
private void loadImageFromURL() {
    try {
        URL url = new URL("http://i.nflcdn.com/static/site" +
                "/3.12/img/teams/SF/SF_logo-80x90.gif");
        picture = ImageIO.read(url);
    } catch (Exception e) {
        // what happens if file not there?
        System.out.println("Unable to load image: " + e);
    }
}
```

- better method would be to have String as parameter to method

Loading Images as Resources

- If creating a stand alone application images may be included as resources
- Java stand alone applications typically packaged as jar files
- images store in directory

```
try {

    imageA = ImageIO.read(getClass().getResource("images/A.jpg"));
    imageB = ImageIO.read(getClass().getResource("images/B.jpg"));
} catch (IOException e) {
    e.printStackTrace();
}
```

Displaying Images

- Images are similar to other graphic primitives (shapes, areas, paths)
- Multiple methods to display the images in Graphics and Graphics2D class
  - Any transform that has been applied to the graphics object affects the image as well
- Highlight a few of them

drawImage methods

- simplest version:

```
public abstract boolean drawImage(Image img,
                  int x,
                  int y,
                  ImageObserver observer)
```

Draws as much of the specified image as is currently available. The image is drawn with its top-left corner at (*x*, *y*) in this graphics context's coordinate space. Transparent pixels in the image do not affect whatever pixels are already there.

- ImageObserver is an object that is notified as image is constructed, changed, or drawn
- We will always send in null for observer

drawImage Methods

- draw scaled version of image

```
public abstract boolean drawImage(Image img,
                  int x,
                  int y,
                  int width,
                  int height,
                  ImageObserver observer)
```

Draws as much of the specified image as has already been scaled to fit inside the specified rectangle.

drawImage Methods

- draw image and supply different background color for transparent pixels (instead of what is already on the panel)

```
public abstract boolean drawImage(Image img,
                  int x,
                  int y,
                  Color bgcolor,
                  ImageObserver observer)
```

Draws as much of the specified image as is currently available. The image is drawn with its top-left corner at (*x*, *y*) in this graphics context's coordinate space. Transparent pixels are drawn in the specified background color.

drawImage Methods

- draw only part of an image
- d = destination, s = source

```
public abstract boolean drawImage(Image img,
                  int dx1,
                  int dy1,
                  int dx2,
                  int dy2,
                  int sx1,
                  int sy1,
                  int sx2,
                  int sy2,
                  ImageObserver observer)
```

Draws as much of the specified area of the specified image as is currently available, scaling it on the fly to fit inside the specified area of the destination drawable surface. Transparent pixels do not affect whatever pixels are already there.

drawImage Methods

- draw using a BufferedImageOp
- BufferedImageOp is another class that applies a filter to the image
  - like image editing software
  - multiple types of BufferedImageOps

```
public abstract void drawImage(BufferedImage img,
                               BufferedImageOp op,
                               int x,
                               int y)
```

Renders a `BufferedImage` that is filtered with a `BufferedImageOp`. The rendering attributes applied include the `Clip`, `Transform` and `Composite` attributes. This is equivalent to:

Demo

- Simple Image Op Program loads an image from a URL and then draws it
- position only
- scaled
- translate and rotate graphics objects and draw image again

Altering Images

- digital images are just a bunch of numbers that represent the color at each pixel

Image Processing

- Image processing and filtering is the result of mathematical operations on the image data, numbers representing colors at each pixel which has a location in the image
- BufferedImageOp
  - Java interface with several implementations already completed
- We will also create our own custom filters

BufferedImageOp

Method Summary

Methods

Modifier and Type	Method and Description
`BufferedImage`	`createCompatibleDestImage(BufferedImage src, ColorModel destCM)` Creates a zeroed destination image with the correct size and number of bands.
`BufferedImage`	`filter(BufferedImage src, BufferedImage dest)` Performs a single-input/single-output operation on a `BufferedImage`.
`Rectangle2D`	`getBounds2D(BufferedImage src)` Returns the bounding box of the filtered destination image.
`Point2D`	`getPoint2D(Point2D srcPt, Point2D dstPt)` Returns the location of the corresponding destination point given a point in the source image.
`RenderingHints`	`getRenderingHints()` Returns the rendering hints for this operation.

- Most important method for us is BufferedImage filter(BufferedImage src, BufferedImage dest)

Example Program

- Examples in ImageExamples program
- two menus
  - one for buffered image ops
  - one for our custom filters
- Button to load new image
- original image displayed on left, filtered on right
- rescales if images too big for display
  - doesn't scale images up (yet)
- Assignment 5, you will add menu options

Built in Buffered Image Ops

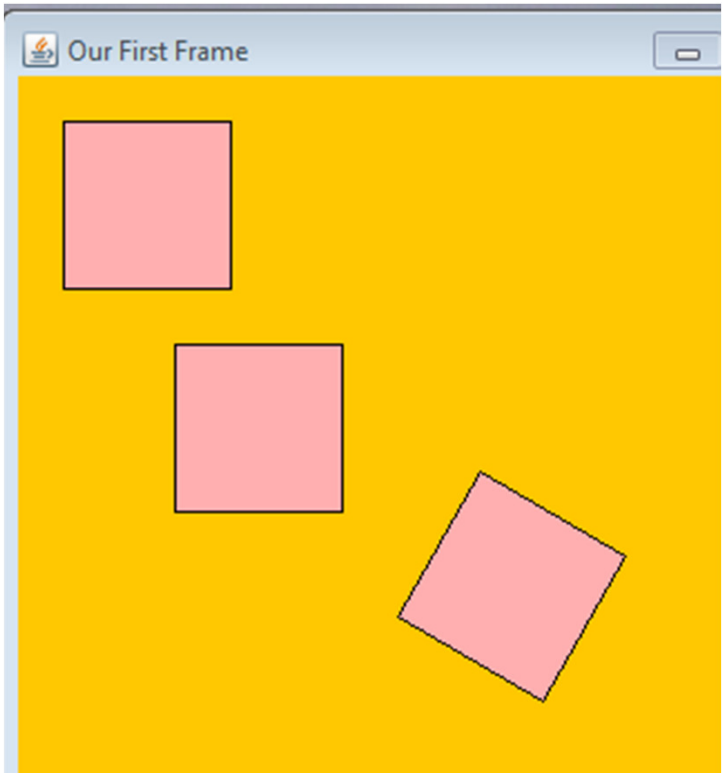
- AffineTransformOp
- RescaleOp
- LookupOp
- ConvolveOp
- ColorConvertOp

AffineTransformOp

- A geometry filter
- Doesn't change color of image
- Applies an affine transform
- Something of a convenience, don't have to apply transform to graphics object and undo
  - or maybe we want to process images before drawing

Affine Transforms

- Methods to apply an *affine transformation*
  - affine transformations alter the coordinates of an object but preserve straight lines and proportions between points on a straight line



Example - Reverse Image

- Along horizontal access
- Create an AffineTransform
- Translate the width of the image in the x direction (or we get negative coordinates for image)
- set scale to (-1, 1)
- filter image

Mirror Image Code

```
private BufferedImage mirrorImage() {
    AffineTransform at = new AffineTransform();
    at.translate(picture.getWidth(), 0);
    at.scale(-1, 1);
    AffineTransformOp result
        = new AffineTransformOp(at,
                AffineTransformOp.TYPE_BICUBIC);
    return result.filter(picture, null);
}
```

- AffineTransformOp.TYPE_BICUBIC is a constant for how to interpolate between if image size changes

RescaleOp

- Implements BufferedImageOp interface
- Specify *values*
  - either one for every channel
  - or scale value for each channel
- and *offsets*
  - offset added to each result after multiplied by scale
- final values clamped at 0 and 255 (for standard RGB)

RescaleOp example

scale, offset, rendering hints

```
new RescaleOp(-1, 255, null);
```

- example: given color (255, 255, 0)
- red = 255 * -1 + 255 = 0
- green = 255 * -1 + 255 = 0
- blue = 0 * -1 + 255 = 255
- result (0, 0, 255)
- What about (0, 0, 0) ? (255, 255, 255)

Sample RescaleOp

- Invert

Rescale Op

- What will this rescale op do?

```
float[] scales = {2, 1.5f, 1.5f};
float[] offsets = {20, 5, 0};
imageOps[0] = new RescaleOp(scales, offsets, null);
```

- f for float.
  - literals with a decimal are doubles
  - RescaleOp constructor expects floats. not doubles

RescaleOp Brighten

RescaleOp Washout

- Scale of 3, offset of 30

Rescale Op GUI

- Make a GUI with slides for scales and offsets
- Build RescaleOp on the fly as sliders adjusted
  - ChangeListener and stateChanged method instead of ActionListener and actionPerformed

LookupOp

- Color changes based on a look up table
- Essentially an array
- original color is input and the index
- value in the table (array) is the resulting color
- for RGB can be a single array that all colors refer to
- or 3 arrays, one for each channel

Creating a LookupOp

- Must first create a LookupTable
  - either a ByteLookupTable or a ShortLookupTable
  - includes an offset that is subtracted from values before indexing into array

LookupOp Simple Example

- Assume grayscale image with 10 shades
  - colors are 0 to 9

index	0	1	2	3	4	5	6	7	8	9
value	1	1	2	2	3	3	3	7	8	9

original image

0	0	0	0	1	1
0	0	0	1	1	1
0	4	4	4	4	7
0	0	0	7	7	7
0	0	1	4	4	4
0	4	8	9	9	6

resulting image

1	1	1	1	1	1
1	1	1	1	1	1
1	3	3	3	3	7
1	1	1	7	7	7
1	1	1	3	3	3
1	3	8	9	9	3

LookupOp

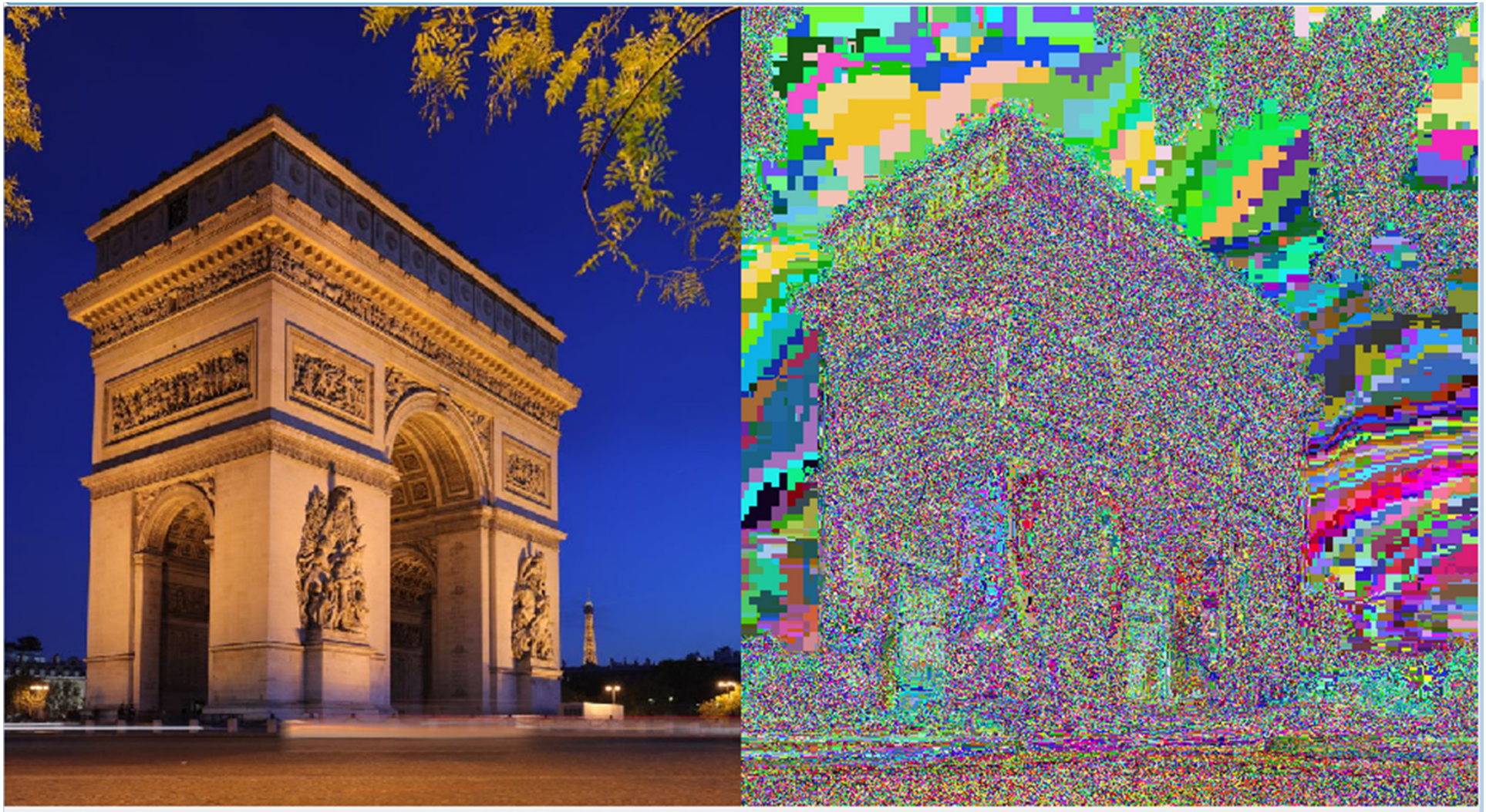
- Randomize table

```
private LookupOp createRandom() {
    short[][] values = new short[3][256];
    ArrayList<Short> list = new ArrayList<Short>();
    for(int i = 0; i < values[0].length; i++)
        list.add((short) i);

    for(int r = 0; r < values.length; r++) {
        Collections.shuffle(list);
        for(int c = 0; c < values[r].length; c++) {
            values[r][c] = list.get(c);
        }
    }

    ShortLookupTable table = new ShortLookupTable(0, values);
    return new LookupOp(table, null);
}
```

Randomize Result



Threshold LookupOp

- For all color values less than some threshold set the intensity to 0
- For all color values greater than or equal to the same threshold values set intensity to 255
- for threshold 70
- (140, 198, 65) -> (255, 255, 0)
- surprising result (Image reduced to 8 colors)

Threshold

Bands LookupOp

- bands of intensities that are unchanged followed by bands of intensities that are reduced
- Bands are input intensity raised to the 0.75 power and truncated

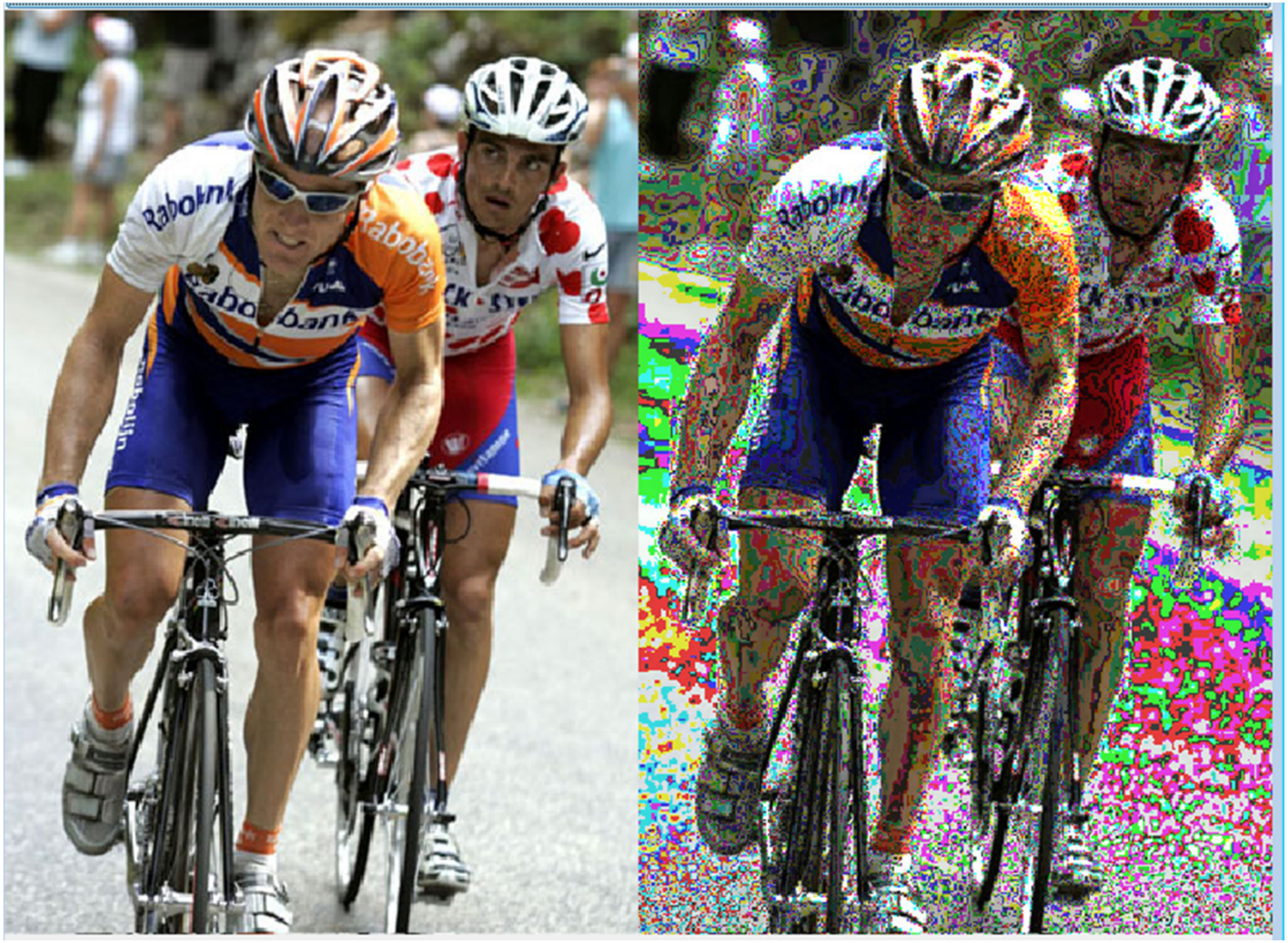
index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
value	0	1	2	3	2	3	3	4	8	9	10	11	6	6	7	7

Bands Code

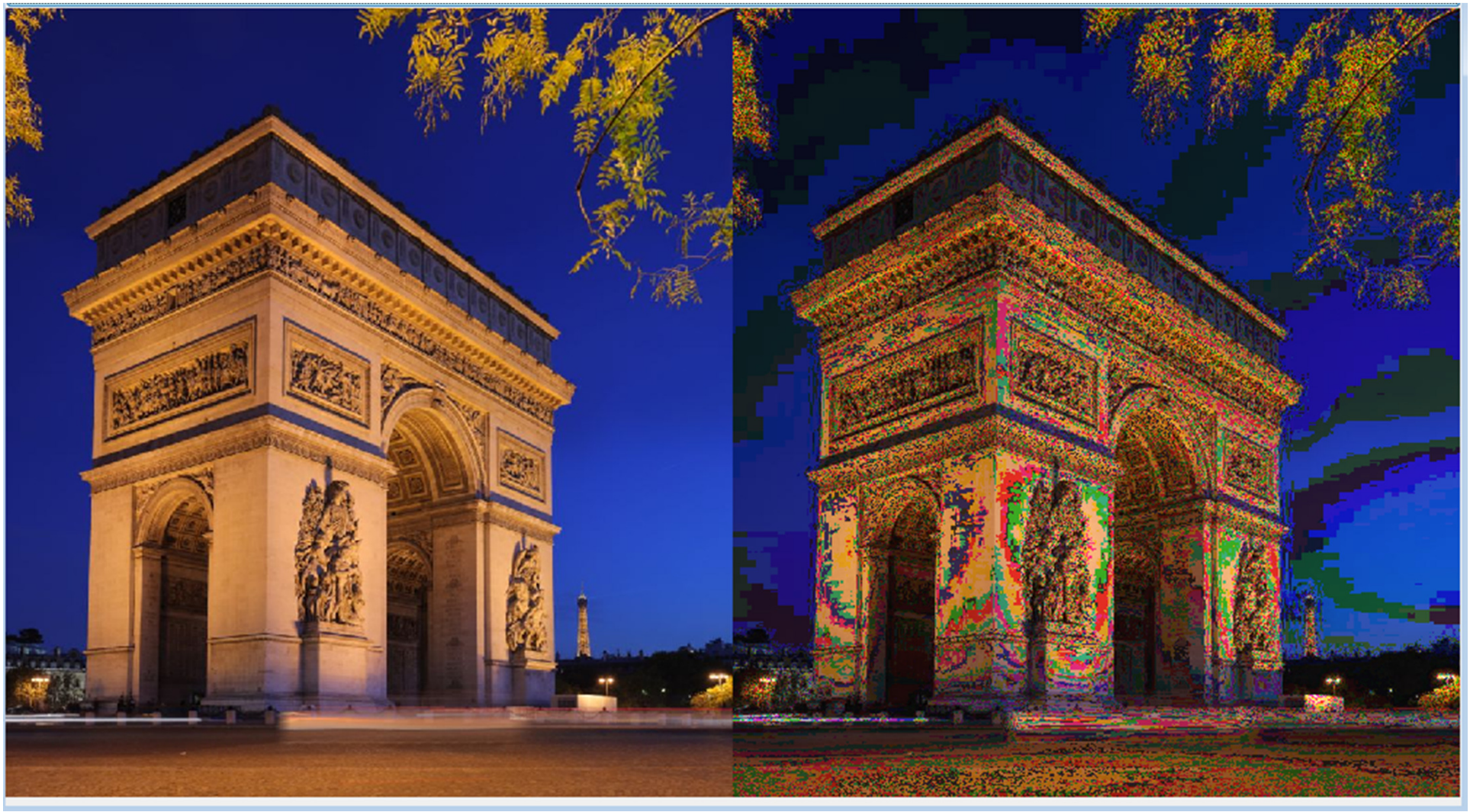
```
private LookupOp createBandsLookup() {
    final int NUM_COLORS = 256;
    short[] table = new short[NUM_COLORS];
    final int BAND_SIZE = 8;
    boolean makeBand = false;
    int count = 0;
    for(int shade = 0; shade < NUM_COLORS; shade++) {
        for(int row = 0; row < table.length; row++)
            if(makeBand)
                table[shade] = (short) Math.pow(shade, 0.75);
            else
                table[shade] = (short) shade;
        count++;
        if( (makeBand && count == BAND_SIZE) ||
                (!makeBand && count == BAND_SIZE * 2)) {
            count = 0;
            makeBand = !makeBand;
        }
    }
    return new LookupOp(new ShortLookupTable(0, table), null);
}
```

Bands Result

Band size = 8



Bands Result



Bands Changed

- Band Size = 16, exponent = 1.25

Bands Changed

ConvolveOp

- A neighborhood filter
- The color of a pixel depends on the color of the pixels around it
- define a matrix called a kernel
- usually a square matrix with an odd number of rows (and thus columns)
- color of resulting pixel obtained by laying kernel over original matrix and multiplying kernel values by original colors

Kernel and Convolve Example

kernel

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

original image

0	0	0	0	1	1
0	0	0	1	1	1
0	5	5	4	4	7
0	0	0	7	7	7
0	0	1	4	4	4
0	4	8	9	9	6

pixel in result at location
(1,1) = 1/9 * 0 + 1/9 * 0 + 1/9 * 0
1/9 * 0 + 1/9 * 0 + 1/9 * 0
1/9 * 0 + 1/9 * 5 + 1/9 * 5
= 10 / 9 = 1

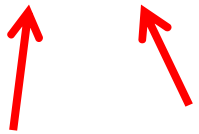
Kernel and Convolve Example

- What does the kernel on the previous slide do?
- if components of kernel sum to less than 1 image will be darkened
- if components of kernel sum to more than 1 image will be brightened

Example Code

- kernel actually a 1d array

```
// blur
float[] kernelValues = new float[49];
Arrays.fill(kernelValues, 1 / 49.0f);
imageOps[2] = new ConvolveOp(new Kernel(7, 7, kernelValues));
```



rows, columns

- values in kernelValues in *row major order*

Result

- Border due to ConvolveOp not handling edge cells gracefully

Another Convolve Op

kernel

-1	-1	-1
2	2	2
-1	-1	-1

original image

0	0	0	0	0	0
0	7	7	7	7	0
0	7	0	0	7	0
0	7	0	0	7	0
0	7	7	7	7	0
0	0	0	0	0	0

Calculate Resulting image.

Border cells set to 0, results clamped between 0 an 9

Result

0	0	0	0	0	0
0	9	9	9	9	0
0	0	0	0	0	0
0	0	0	0	0	0
0	9	9	9	9	0
0	0	0	0	0	0

- What does this filter do?
- Will this brighten or darken an image?

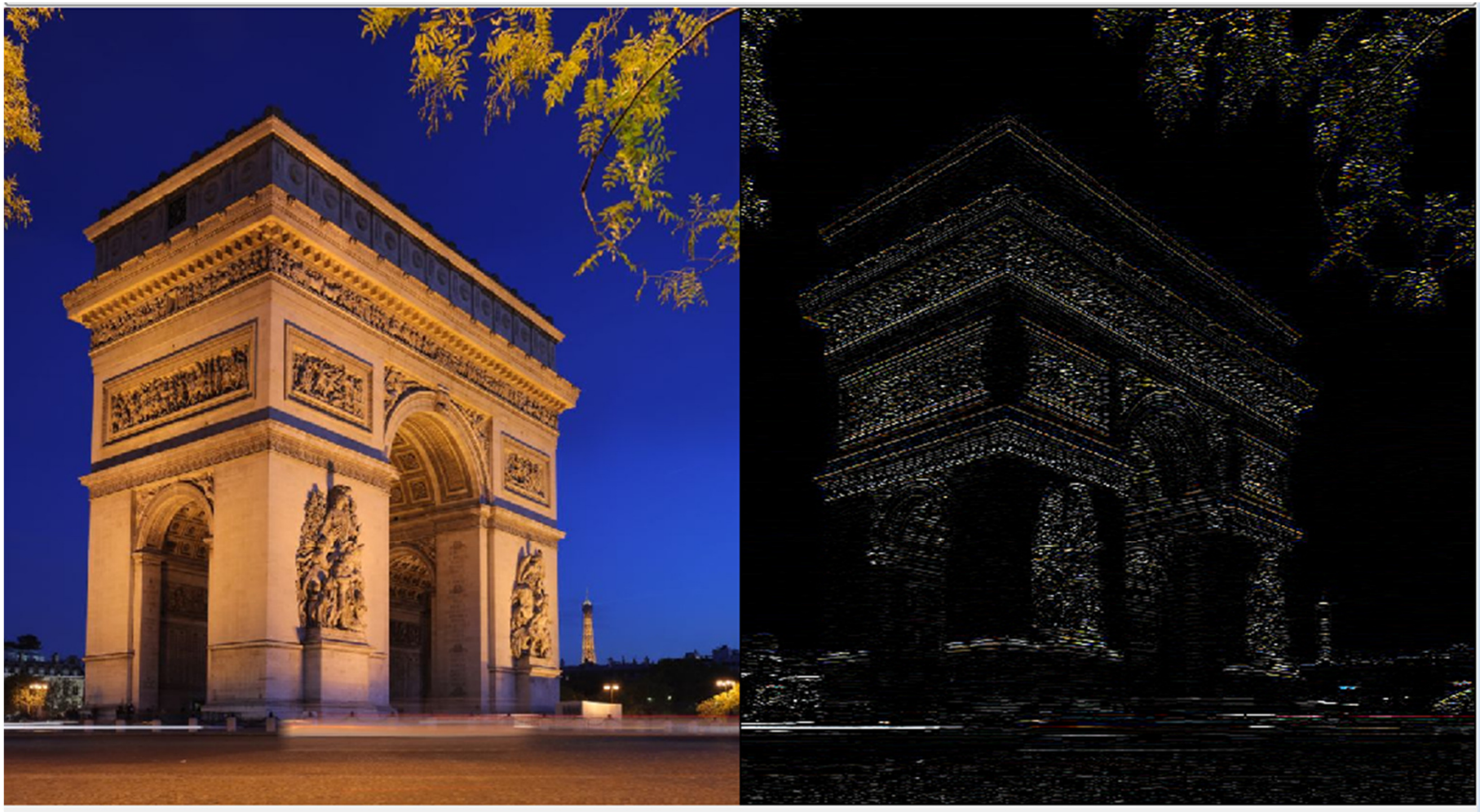
Edge Detection

- Vertical Lines

-1	2	-1
-1	2	-1
-1	2	-1

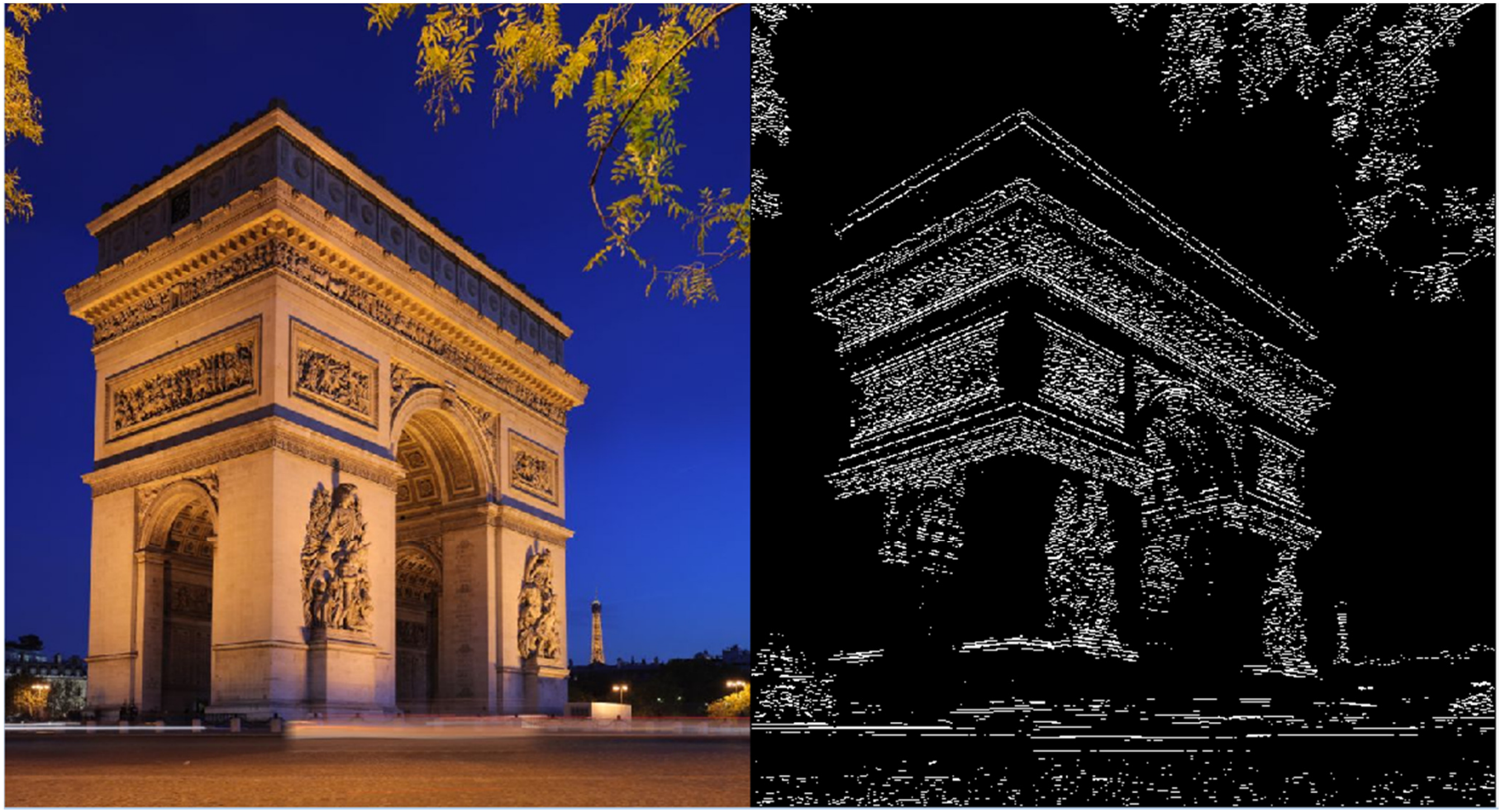
to see lines more cleanly:
-apply kernel
-set all pixels above some threshold to white

Horizontal Line



Horizontal Line

- Threshold at 125 total (sum of rgb after convolve op



Convolve Op

- Emboss
- Will this brighten or darken the image?

-2	-1	0
-1	1	1
0	1	2

Emboss

Making Our Own Filters

- Create our own abstract class to create filters
- *Filthy Rich Clients* shows how to create a new class that implements BufferedImageOp
- For assignment use our own FilterOp class

FilterOp

- abstract class
- some methods defined, some abstract
  - abstract method declared, but no implementation
  - class that extends FilterOp must implement the abstract methods or be abstract itself

FilterOp

- methods:
- public BufferedImage filter(BufferedImage src)

// pull out part of color

- public static int getRed(int pixel)
- public static int getGreen(int pixel)
- public static int getBlue(int pixel)

// get all components

- public static int[] getRGB(int pixel)
- public static int makePixel(int red, int green, int blue)

FilterOp

- public abstract int filterOp(int pixel,
  BufferedImage src);
- abstract method
- no body or implementation
- class that extends FilterOp must implement the abstract methods or be abstract itself

FilterOp

```
public BufferedImage filter(BufferedImage src){
    // create result (all pixels black initially)
    BufferedImage result = new BufferedImage(src.getWidth(),
            src.getHeight(), src.getType());

    // process every pixel in the source image
    for(int x = 0; x < src.getWidth(); x++)
        for(int y = 0; y < src.getHeight(); y++)
            result.setRGB(x, y, filterOp(src.getRGB(x, y), src));

    return result;
}
```

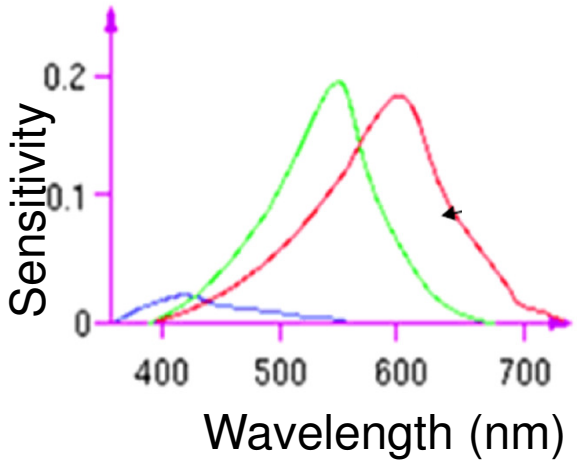
- Some filters will required overriding the filter method in the FilterOp class

FilterOp Example

- Grayscale
- naïve:

Grayscale

- Naïve approach: average together red, green, and blue to get a shade of gray
- problem: our eye is more sensitive to green and red than blue
- (0,255,0) should be "brighter" than (0,0,255)



Grayscale

- Typical Grayscale conversion
- gray = Red * .3 + Green * .59 + Blue * .11
- color = (gray, gray, gray)

Grayscale Side by Side

Grayscale Side by Side

Grayscale Class

```
class Grayscale extends FilterOp {

    public int filterOp(int pixel, BufferedImage src){
        int r = getRed(pixel);
        int g = getGreen(pixel);
        int b = getBlue(pixel);
        int gray = (int)(0.3 * r + 0.59 * g + 0.11 * b);
        return makePixel(gray, gray, gray);
    }
} // end of Grayscale class
```

- What BufferedImageOp could we have used instead?

HotMetal FilterOp

- Converts color at pixel to grayscale then

```
class HotMetal extends FilterOp {

    private int[] values;

    public HotMetal() {
        values = new int[256];
        final int MAX_RED = 170;
        for(int i = 0; i < values.length; i++) {
            int red = (int) Math.min(1.0 * i / MAX_RED * 255, 255);
            int green = i - MAX_RED;
            green = green <= 0
                    ? 0 :
                    (short) Math.min(1.0 * green / 85 * 255, 255);
            values[i] = makePixel(red, green, 0);
        }
    }

    public int filterOp(int pixel, BufferedImage src){
        int r = getRed(pixel);
        int g = getGreen(pixel);
        int b = getBlue(pixel);
        int gray = (int)(0.3 * r + 0.59 * g + 0.11 * b);
        return values[gray];
```

HotMetal