CS 1124
Media Computation

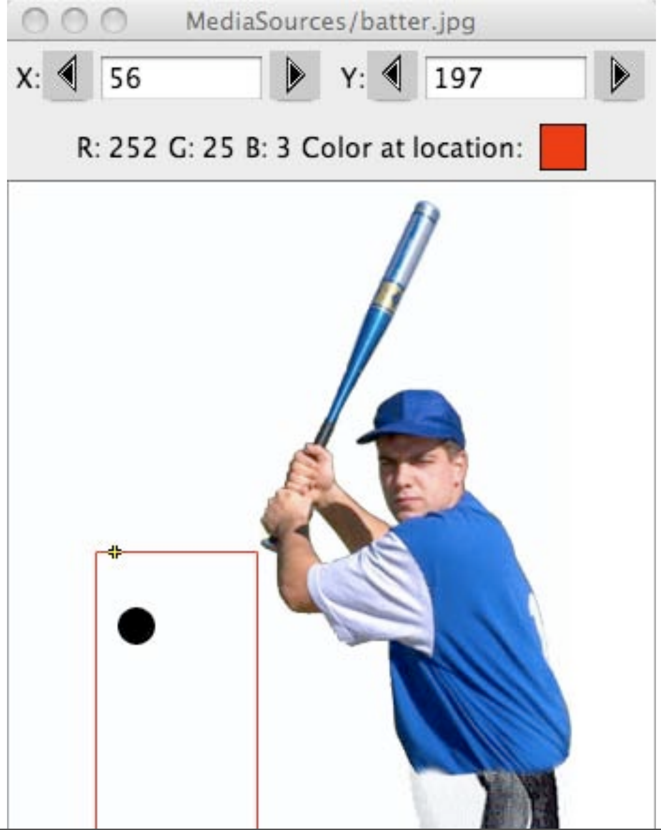
Steve Harrison

Lecture 5.2 (September 25, 2008)

Before we get to today's main events ...

Remember our JPEG problem in Project 4?

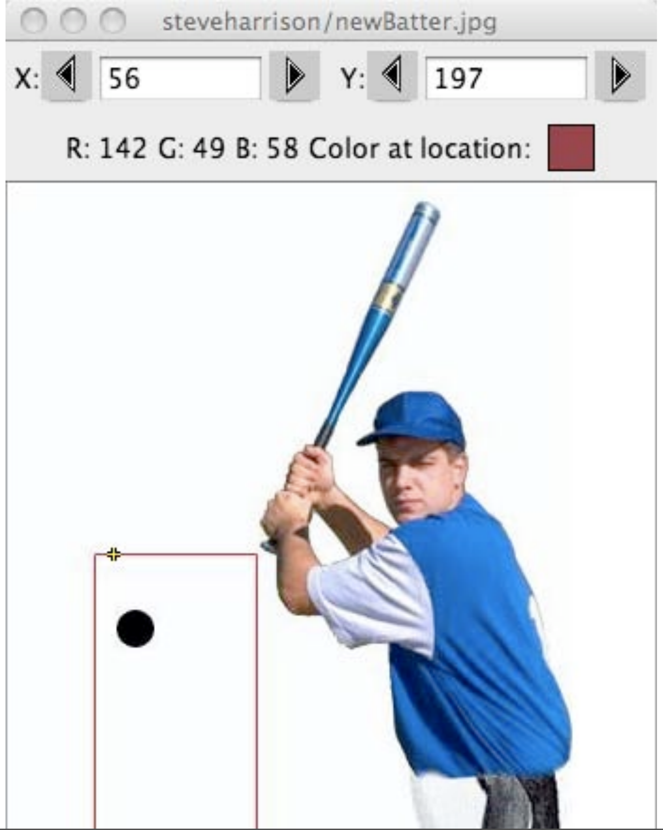
```
>>> batterFile = pickAFile()
>>> batterPic = makePicture( batterFile )
>>> writePictureTo( batterPic, "newBatter.jpg" )
>>> newBatterPic = makePicture( pickAFile() )
```



MediaSources/batter.jpg

X: 56 Y: 197

R: 252 G: 25 B: 3 Color at location:



steveharrison/newBatter.jpg

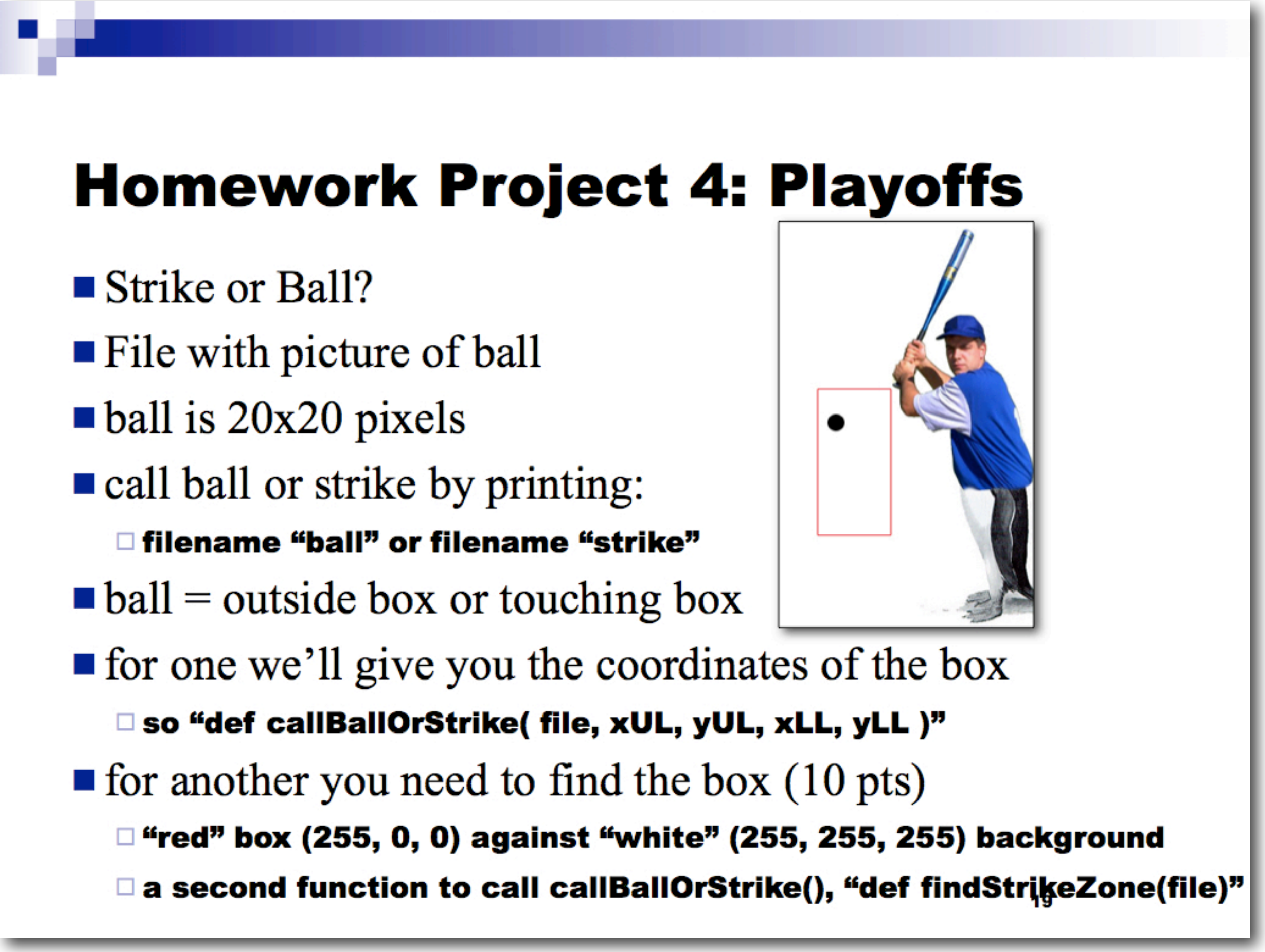
X: 56 Y: 197

R: 142 G: 49 B: 58 Color at location:

Look at the red line of the strike zone.
And neither are (255,0,0) !

Simple solution -- use .png format

- Red (255,0,0)
- White (255, 255, 255)
- Black (0,0,0)



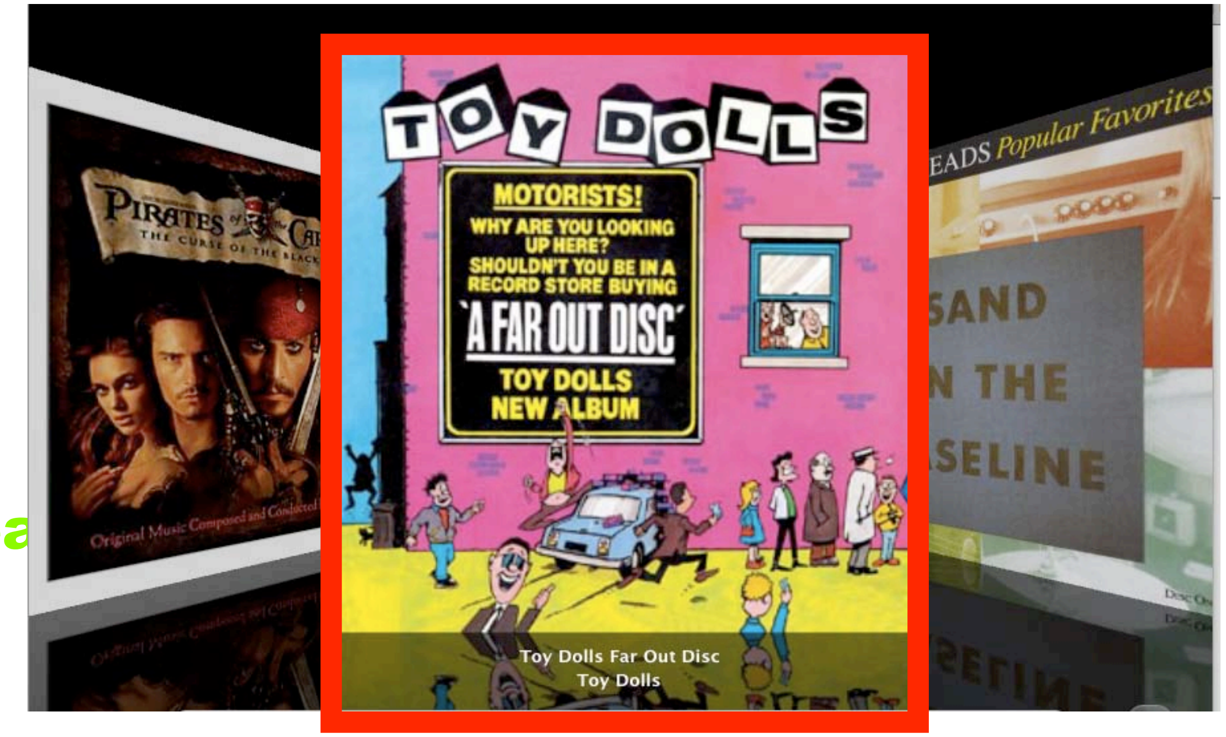
Homework Project 4: Playoffs

- Strike or Ball?
- File with picture of ball
- ball is 20x20 pixels
- call ball or strike by printing:
  - **filename “ball” or filename “strike”**
- ball = outside box or touching box
- for one we’ll give you the coordinates of the box
  - **so “def callBallOrStrike(file, xUL, yUL, xLL, yLL)”**
- for another you need to find the box (10 pts)
  - **“red” box (255, 0, 0) against “white” (255, 255, 255) background**
  - **a second function to call callBallOrStrike(), “def findStrikeZone(file)”**

Today

- iTunes effect
  - **who has the mirror effect ?**
- Really transforming pictures....
  - **swapping backgrounds**
  - **chromaKey (or the art of the Weather Channel)**
- Drawing graphics
  - **Drawing graphics by changing lots of pixels**
  - **Graphics functions that are built in to JES**
  - **Programmed graphics**

High level



```
def iTunesEffect(fileName):
    # get the picture, its height and crea
    source = makePicture( fileName )
    sourceHeight = getHeight( source )
    target = makeEmptyPicture( getWidth(source), int( sourceHeight*1.5 ) )
    # copy the picture
    target = copyPicture( source, target, 1, 1 )
    # now put fading mirror image below picture
    target = mirrorFade( source, target, 1, sourceHeight )
    show( target )
    return target
```

Lower level: mirrorFade(s,t,x,y) alternatives

```
def mirrorFade(src, trgt, startX, startY):
    # set source y to last row so that we copy from bottom to top for mirror effect
    srcHeight = getHeight( src ) * 1.0
    srcY = srcHeight
    # for each y in the target from the startY to the height of the target
    for trgtY in range(startY, getHeight( trgt ) + 1 ) :
        # figure out how much to fade to black for this row
        fade = (srcY  / srcHeight) - 0.25 <== subtracting a factor
        # for each x in the target and the source from the startX to the width of the pictures
        for x in range( startX, getWidth( src ) + 1 ) :
            # get the pixel from the source picture
            srcPixel = getPixel( src, x, int(srcY ) )
            # multiply each color by the fade factor
            trgtRed = int( getRed( srcPixel ) * fade)
            trgtGreen = int( getGreen( srcPixel ) * fade )
            trgtBlue = int( getBlue( srcPixel ) * fade )
            # put the pixel into the target
            setColor( getPixel( trgt, x, trgtY ), makeColor( trgtRed, trgtGreen, trgtBlue ) )
        # decrement the row in the source file to move towards the top of the source
        srcY = srcY - 2.0 <== stepping by twos makes floor seem more oblique to viewer
        if srcY < 1.0 :
            srcY = 1.0
    return trgt
```

Who made a good mirrored floor?

- Fade function?
- Step?

Today

- iTunes effect
- Really transforming pictures….
  - swapping backgrounds
  - chromaKey (or the art of the Weather Channel)
- Drawing graphics
  - Drawing graphics by changing lots of pixels
  - Graphics functions that are built in to JES
  - Programmed graphics

New Stuff - chromakey and pixel replacement

- This is really cool....
- If pixel color is in certain range replace with pixel from another picture

Swap the background

- If this pixel is nearly the same as the pixel in a background-only picture, then substitute a pixel from a new background picture

```
def swapBackground( src, background, newBackground ):
    # src, and background must be the same size
    # newBackground must be at least as big as src and background
    for x in range(1, getWidth( src ) + 1 ) :
        for y in range( 1, getHeight( src ) + 1 ) :
            srcPxl = getPixel( src, x, y )
            backgroundPxl = getPixel( background, x, y )
            if (distance(getColor( srcPxl ),  getColor( backgroundPxl )) < 15.0):
                setColor( srcPxl, getColor( getPixel( newBackground, x, y )  )  )
    return src
```

Swap the background

- If this pixel is nearly the same as the pixel in a background-only picture, then substitute a pixel from a new background picture

```
def swapBackground( src, background, newBackground ):
    # src, and background must be the same size
    # newBackground must be at least as big as src and background
    for x in range(1, getWidth( src ) + 1 ) :
        for y in range( 1, getHeight( src ) + 1 ) :
            srcPxl = getPixel( src, x, y )
            backgroundPxl = getPixel( background, x, y )
            if (distance(getColor( srcPxl ),  getColor( backgroundPxl )) < 15.0):
                setColor( srcPxl, getColor( getPixel( newBackground, x, y )  )  )
    return src
```

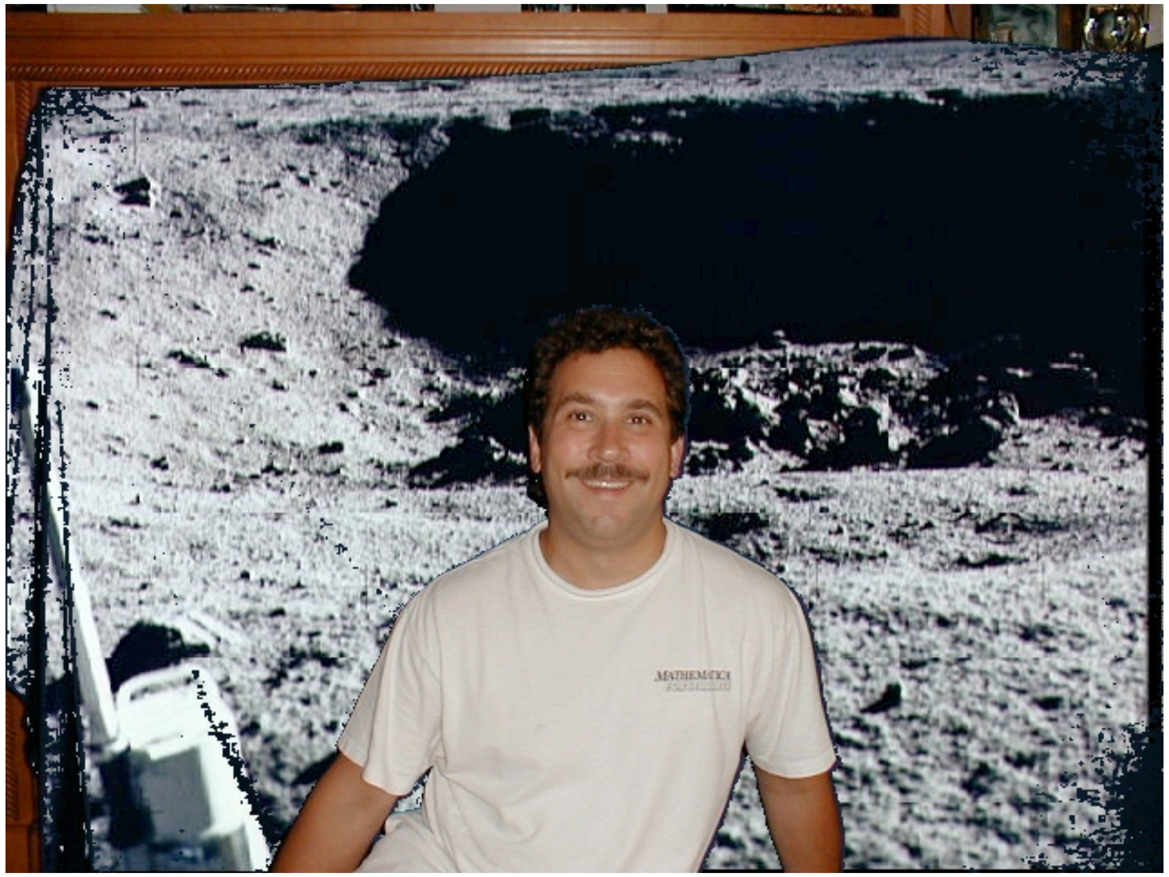
Is JPEG compression a problem?

Chromakey - just like the Weather Channel

```
def chromaKey( src, background ):
    # src, background, newBackground must be the same size
    for x in range(1, getWidth( src ) + 1 ) :
        for y in range( 1, getHeight( src ) + 1 ) :
            srcPxl = getPixel( src, x, y )
            backgroundPxl = getPixel( background, x, y )
            if (getRed( srcPxl ) + getGreen( srcPxl ) < getBlue( srcPxl )):
                setColor( srcPxl, getColor( getPixel( background, x, y )  )  )
    return src
```

Chromakey

- Now that's really cool!
- Unrealistic because:
  - Mark lit from front, moon lit from back right
  - wood frame shows
  - folds
  - Mark in focus, equipment not
  - edge around Mark:
    - flash makes shadow on screen
    - jpeg compression emphasizes changes in luminance



Today

- iTunes effect
- Really transforming pictures....
  - **swapping backgrounds**
  - **chromaKey (or the art of the Weather Channel)**
- Drawing graphics
  - **Drawing graphics by changing lots of pixels**
  - **Graphics functions that are built in to JES**
  - **Programmed graphics**

Drawing Graphics

- **Drawing graphics by changing lots of pixels**
- Graphics functions that are built in to JES
- Programmed graphics

We can make whatever we want on pictures already

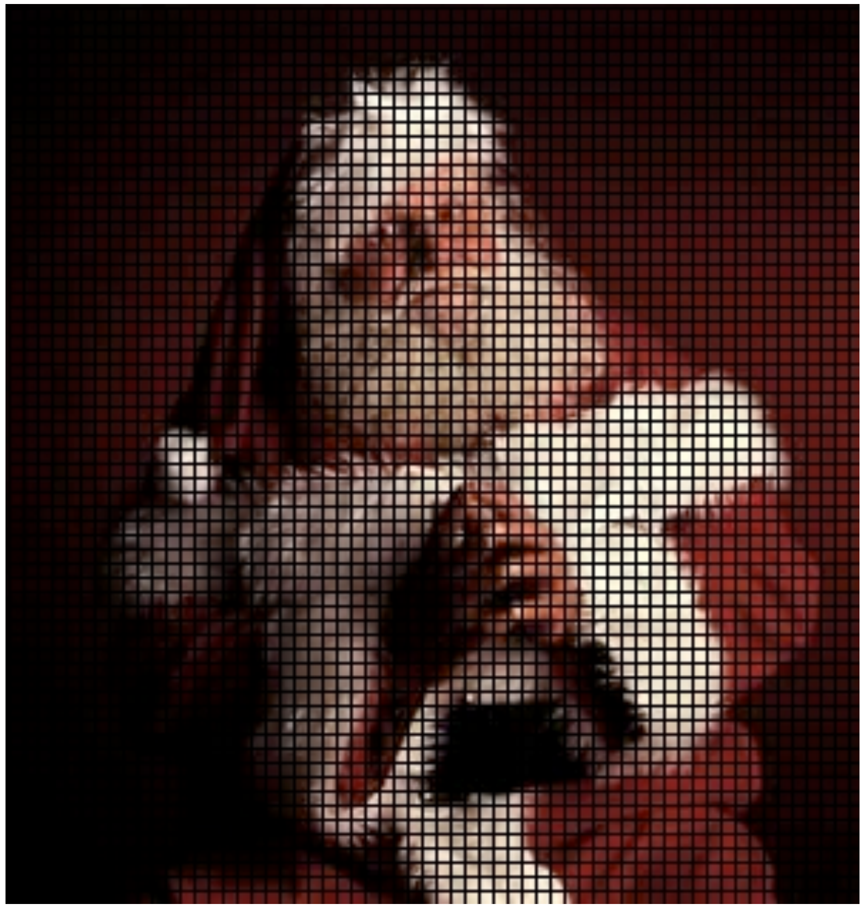
- *All* drawing on pictures comes down to changing pixel values.
- By directly changing values to black (or whatever else we want), we can draw whatever we want.

Drawing lines on Santa

```
def lineExample():
  img = makePicture(pickAFile())
  new = verticalLines(img)
  new2 = horizontalLines(img)
  show(new2)
  return new2

def horizontalLines(src):
  for x in range(1,getHeight(src),5):
    for y in range(1,getWidth(src)):
      setColor(getPixel(src,y,x),black)
  return src

def verticalLines(src):
  for x in range(1,getWidth(src),5):
    for y in range(1,getHeight(src)):
      setColor(getPixel(src,x,y),black)
  return src
```



Nested loops (one loop inside another loop):

Colors defined for you already: black, white, blue, red, green, gray, lightGray, darkGray, yellow, orange, pink, magenta & cyan

But that’s tedious

- It’s slow and tedious to set every pixel you want.
- What you really want to do is to think in terms of your desired effect (think about “requirements” and “design”)
    - **E.g. Instead of “change the color of all the pixels that happen to be in a line to black”, say “draw a black line”**

Drawing Graphics

- Drawing graphics by changing lots of pixels
- **Graphics functions that are built in to JES**
- Programmed graphics

New functions

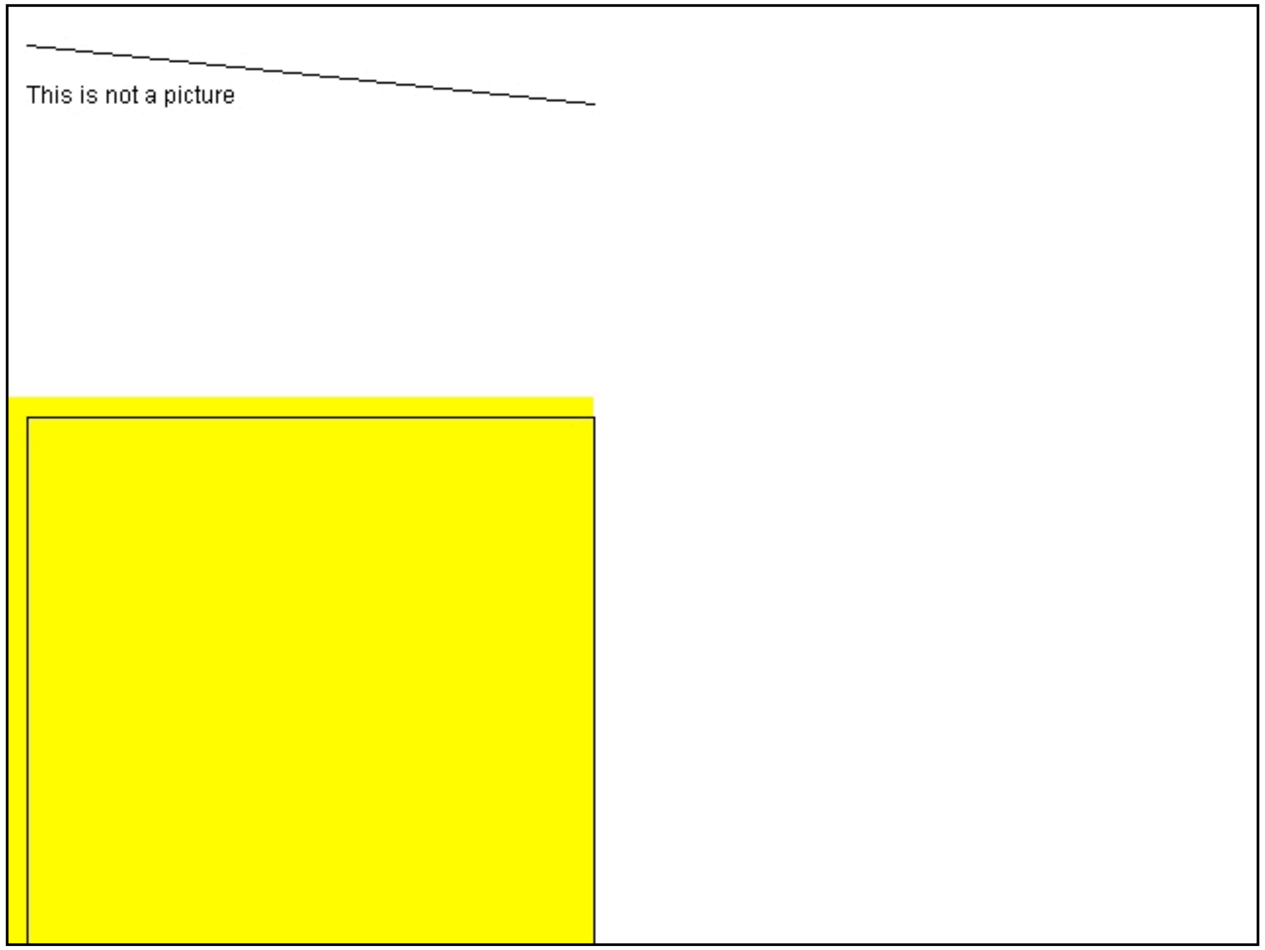
- **addText(pict,x,y,string)** puts the string starting at position (x,y) in the picture
- **addLine(picture,x1,y1,x2,y2)** draws a line from position (x1,y1) to (x2,y2)
- **addRect(pict,x1,y1,w,h)** draws a black rectangle (unfilled) with the upper left hand corner of (x1,y1) and a width of w and height of h. Same as:

```
addLine(pict, x1, y1, x1+w, y1)
addLine(pict, x1+w, y1, x1+w, y1+h)
addLine(pict, x1+w, y1+h, x1, y1+h)
addLine(pict, x1, y1+h, x1, y1)
```

- **addRectFilled(pict,x1,y1,w,h,color)** draws a rectangle filled with the color of your choice with the upper left hand corner of (x1,y1) and a width of w and height of h

Example picture

```
def littlepicture():
  canvas=makePicture(getMediaPath("640x480.jpg"))
  addText(canvas,10,50,"This is not a picture")
  addLine(canvas,10,20,300,50)
  addRectFilled(canvas,0,200,300,500,yellow)
  addRect(canvas,10,210,290,490)
  return canvas
```



A thought experiment

- Look at that previous page: Which is a fewer number of bytes?
  - **The program that drew the picture**
  - **The pixels in the picture itself.**

A thought experiment

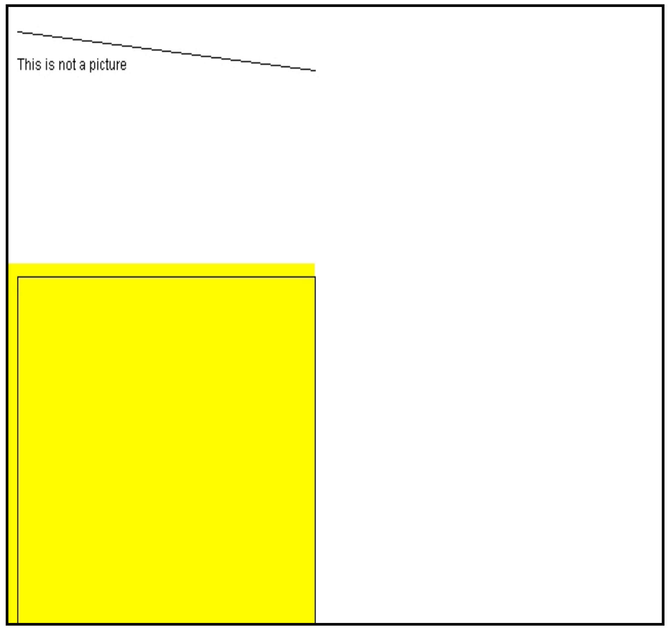
- Look at that previous page: Which is a fewer number of bytes?
    - The program that drew the picture
    - The pixels in the picture itself.

```
def littlepicture():
  canvas=makePicture(getMediaPath("640x480.jpg"))
  addText(canvas,10,50,"This is not a picture")
  addLine(canvas,10,20,300,50)
  addRectFilled(canvas,0,200,300,500,yellow)
  addRect(canvas,10,210,290,490)
  return canvas
```

A thought experiment

- Look at that previous page: Which is a fewer number of bytes?
  - **The program that drew the picture**
  - **The pixels in the picture itself.**

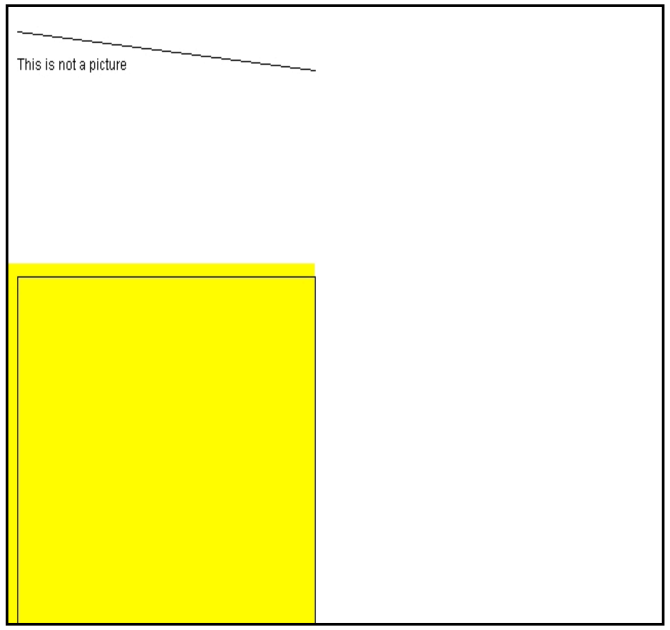
```
def littlepicture():
  canvas=makePicture(getMediaPath("640x480.jpg"))
  addText(canvas,10,50,"This is not a picture")
  addLine(canvas,10,20,300,50)
  addRectFilled(canvas,0,200,300,500,yellow)
  addRect(canvas,10,210,290,490)
  return canvas
```



A thought experiment

- Look at that previous page: Which is a fewer number of bytes?
  - □ **The program that drew the picture**
  - □ **The pixels in the picture itself.**

```
def littlepicture():
  canvas=makePicture(getMediaPath("640x480.jpg"))
  addText(canvas,10,50,"This is not a picture")
  addLine(canvas,10,20,300,50)
  addRectFilled(canvas,0,200,300,500,yellow)
  addRect(canvas,10,210,290,490)
  return canvas
```



- It’s a no-brainer
  - □ **The program is less than 300 characters (100 bytes)**
  - □ **The picture is stored on disk at about 15,000 bytes**

Drawing Graphics

- Drawing graphics by changing lots of pixels
- Graphics functions that are built in to JES
- **Programmed graphics**

Vector-based vs. Bitmap Graphical representations

- Vector-based graphical representations are basically *executable programs* that generate the picture on demand.
  - Postscript, Illustrator, Flash, and AutoCAD use vector-based representations
  - Editors change the vector representation which changes the picture
- Bitmap graphical representations (like JPEG, BMP, GIF) store individual pixels or representations of those pixels.
  - JPEG and GIF are actually compressed picture representations

Vector-based representations can be smaller

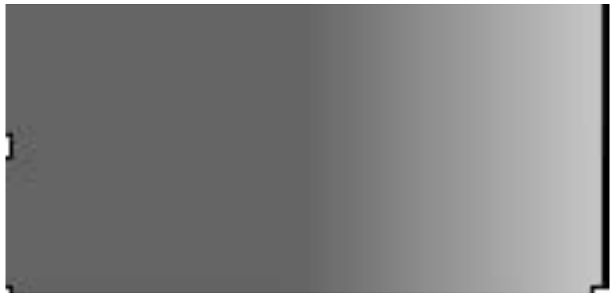
- Vector-based representations can be much smaller than bit-mapped representations
  - **Smaller means faster transmission (Flash and Postscript)**
  - **If you want all the detail of a complex picture, no, it’s not.**

But vector-based has more value than that

- Imagine that you're editing a picture with lines on it.
  - **If you edit a bitmap image and extend a line, it's just more bits.**
    - There's no way to really realize that you've *extended* or *shrunk* the line.
  - **If you edit a vector-based image, it's possible to just change the specification**
    - Change the numbers saying where the line is
    - Then it *really is* the same line
- That's important when the picture drives the creation of the product, like in automatic cutting machines

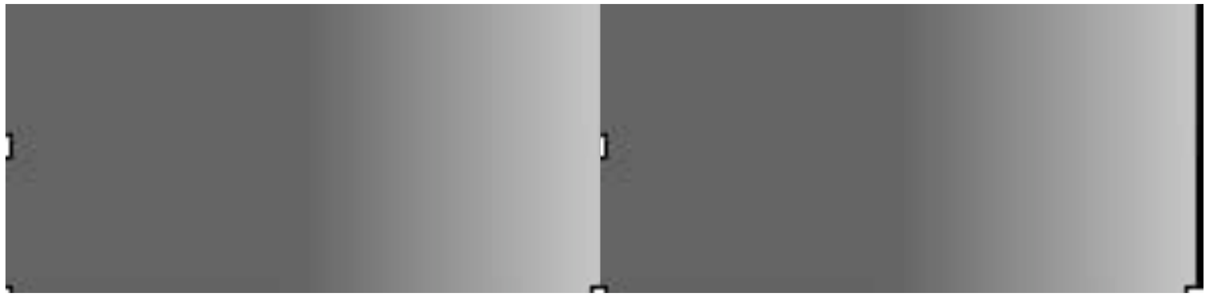
Example programmed graphic

- If I did this right, we perceive the left half as lighter than the right half
- In reality, the end quarters are actually the same colors.



Example programmed graphic

- If I did this right, we perceive the left half as lighter than the right half
- In reality, the end quarters are actually the same colors.

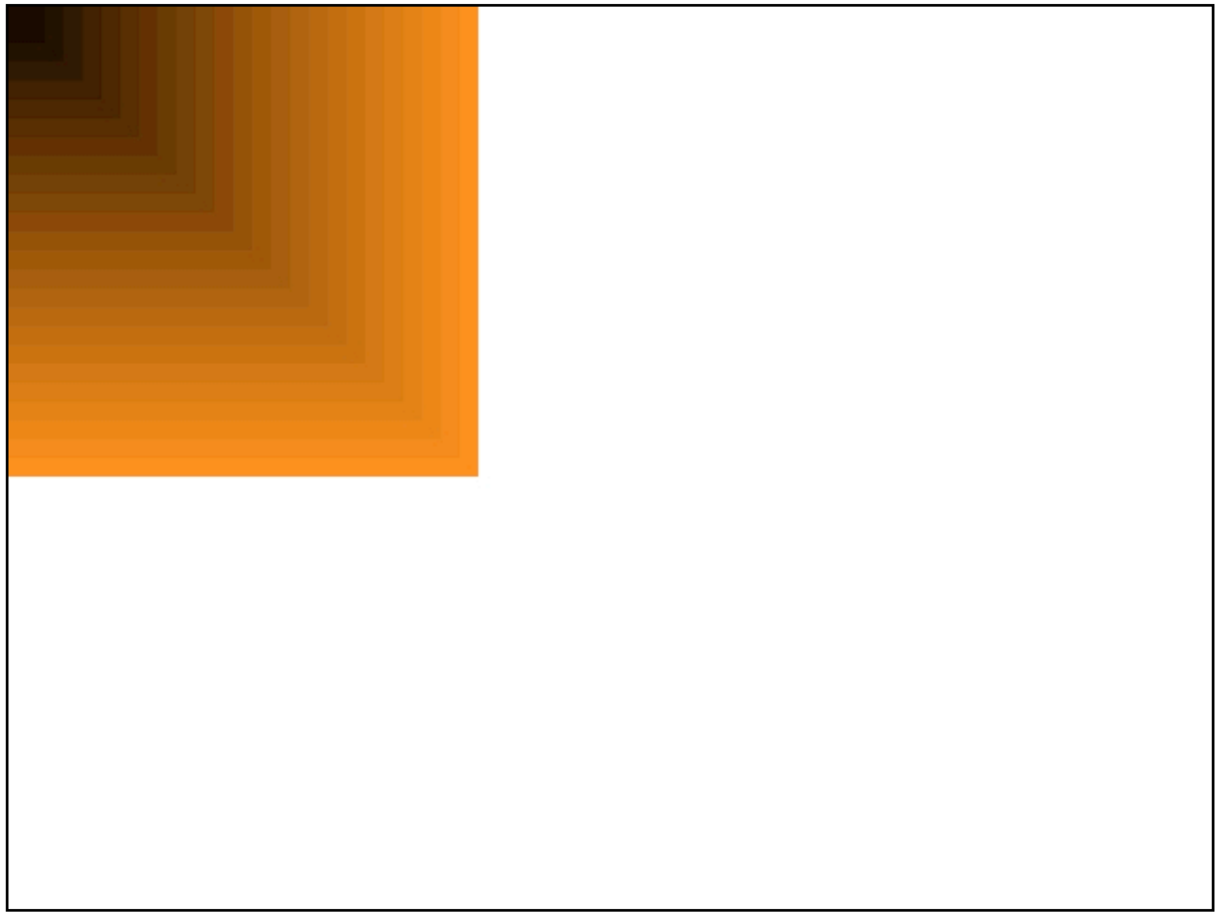


Building a programmed graphic

```
def greyEffect():
 file = getMediaPath("640x480.jpg")
 pic = makePicture(file)
 # First, 100 columns of 100-grey
 grey = makeColor(100,100,100)
 for x in range(1,100):
  for y in range(1,100):
   setColor(getPixel(pic,x,y),grey)
 # Second, 100 columns of increasing greyness
 greyLevel = 100
 for x in range(100,200):
  grey = makeColor(greyLevel, greyLevel, greyLevel)
  for y in range(1,100):
   setColor(getPixel(pic,x,y),grey)
  greyLevel = greyLevel + 1
# Third, 100 colums of increasing greyness, from 0
 greyLevel = 0
 for x in range(200,300):
  grey = makeColor(greyLevel, greyLevel, greyLevel)
  for y in range(1,100):
   setColor(getPixel(pic,x,y),grey)
  greyLevel = greyLevel + 1
 # Finally, 100 columns of 100-grey
 grey = makeColor(100,100,100)
 for x in range(300,400):
  for y in range(1,100):
   setColor(getPixel(pic,x,y),grey)
 return pic
```

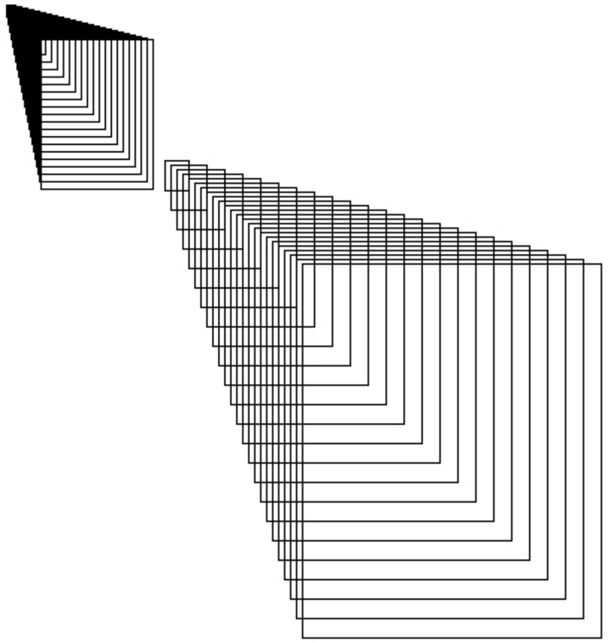
Another Programmed Graphic

```
def coolpic():
  canvas=makePicture(getMediaPath("640x480.jpg"))
  for index in range(25,1,-1):
    color = makeColor(index*10,index*5,index)
    addRectFilled(canvas,0,0,index*10,index*10,color)
  show(canvas)
  return canvas
```



And another

```
def coolpic2():
  canvas=makePicture(getMediaPath("640x480.jpg"))
  for index in range(25,1,-1):
    addRect(canvas,index,index,index*3,index*4)
    addRect(canvas,100+index*4,100+index*3,index*8,index*10)
  show(canvas)
  return canvas
```



Why do we write programs?

- Could we do this in Photoshop? Maybe
  - **I’m sure that you can, but you need to know how.**
  - **Illustrator is probably better, but still need to learn.**
- Could I teach you to do this in Photoshop? Maybe
  - **Might take a lot of demonstration**
- But this program is an *exact* definition of the process of generating this picture
  - **It works for anyone who can run the program, without knowing Photoshop**

Today

- iTunes effect
- Really transforming pictures....
  - **swapping backgrounds**
  - **chromaKey (or the art of the Weather Channel)**
- Drawing graphics
  - **Drawing graphics by changing lots of pixels**
  - **Graphics functions that are built in to JES**
  - **Programmed graphics**

Coming Attractions

- This Friday (9/26)
  - Group project due 2:00 PM
  - e-mail .zip file to srh@vt.edu
  - Bring to Lab!
- Next Monday (9/29)
  - Assignment 4 due 10:00 AM
- Next Wednesday (10/1)
  - midterm
  - midterm practice quiz available -- NOT GRADED