YEAH: Assignment 3

Images and Graphics

with Kara & Will!

Overview: Images

We use a SimpleImage module to help us visualize and manipulate images

We can do things like:

- Read image from a file
- Loop over pixels of an image
- Access color data inside a pixel

For more detailed information,
check out the Image Reference Guide & Lecture 9: Images

Part 1: Images

Finding forest fires

(Sandcastle Problem!)

Detecting Wildfires

Goal: Highlight areas where a forest fire is active

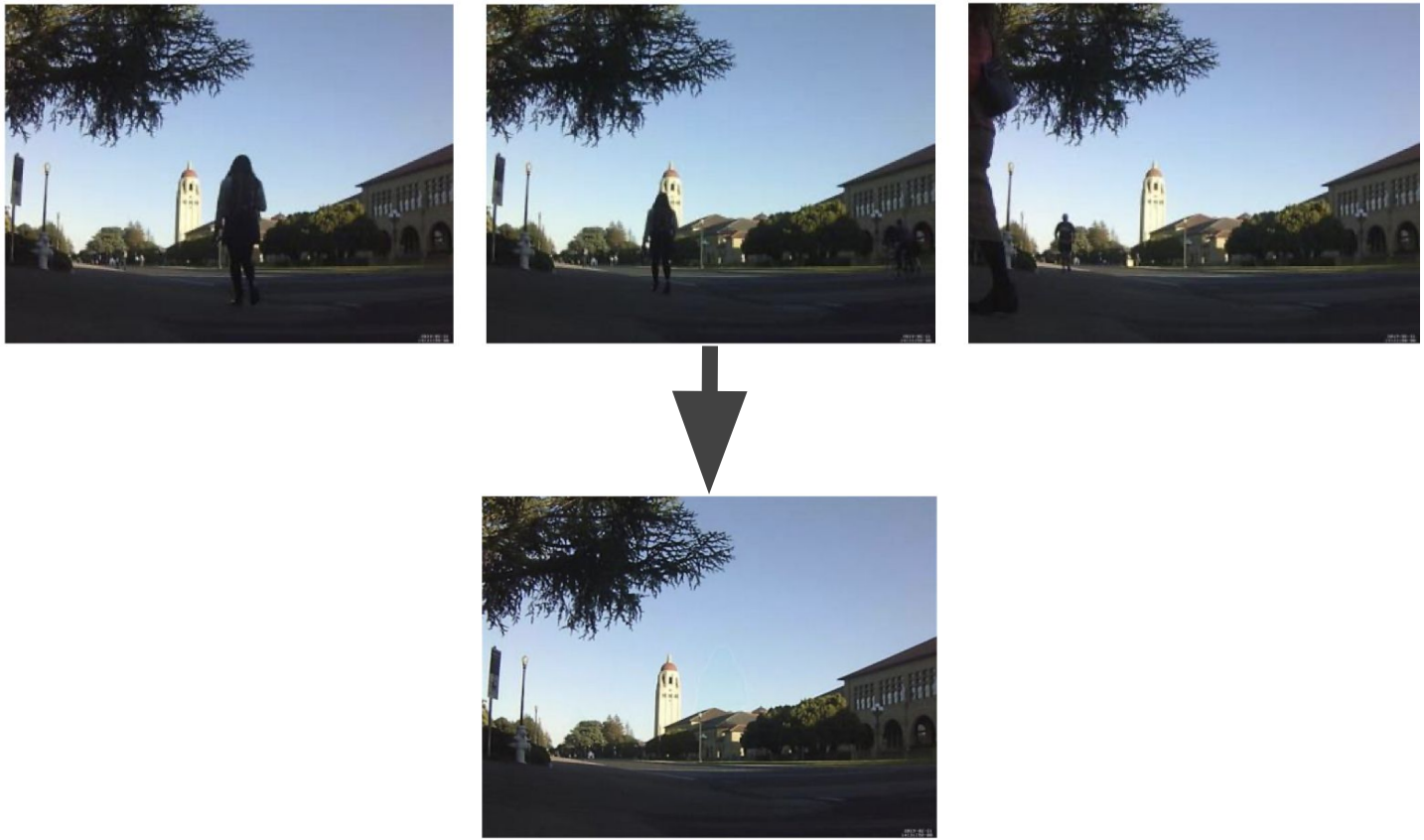
- Step 1: Determine if pixel is “sufficiently red”
- Step 2: If sufficiently red, set its red value to 255 (and green/blue to 0). If not sufficient, convert to gray scale value.

From lecture...

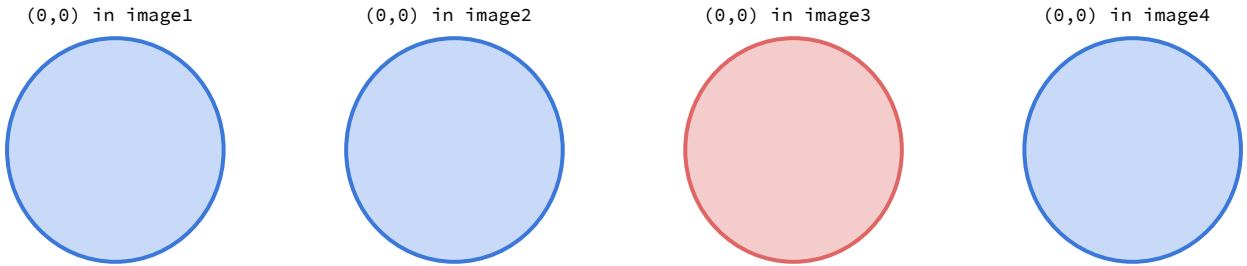
```
def redscreen(main_filename, back_filename):
    """
    Implements the notion of "redscreening".  That is,
    the image in the main_filename has its "sufficiently"
    red pixels replaced with pized from the corresponding x,y
    location in the image in the file back_filename.
    Returns the resulting "redscreened" image.
    """
    image = SimpleImage(main_filename)
    back = SimpleImage(back_filename)
    for pixel in image:
        average = (pixel.red + pixel.green + pixel.blue) // 3
        # See if this pixel is "sufficiently" red
        if pixel.red >= average * INTENSITY_THRESHOLD:
            # If so, we get the corresponding pixel from the
            # back image and overwrite the pixel in
            # the main image with that from the back image.
            x = pixel.x
            y = pixel.y
            image.set_pixel(x, y, back.get_pixel(x, y))
    return image
```

Ghost

Problem: How do we convert these three images with people in them into one image without anyone in it?



If there are 4 images and the pixel at (0,0) in each of the four looks like this:



then the red pixel (image3) is probably an error/outlier

Basic idea

- For each (x,y) coordinate , we are going to find the "best" pixel and put that pixel in our solution's (x,y) pixel location
- "Best" pixel is the pixel that has the shortest distance between itself and the average pixel
- The average pixel has the average red, green, and blue values from each input pixel at that coordinate. So given image1, image2, and image3, the average pixel should be the average of image1’s, image2’s, and image3's RGB values

Computing distance

- Use the euclidean distance formula
- Distance between points (x_1, y_1, z_1) and (x_2, y_2, z_2)

$$distance^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2$$

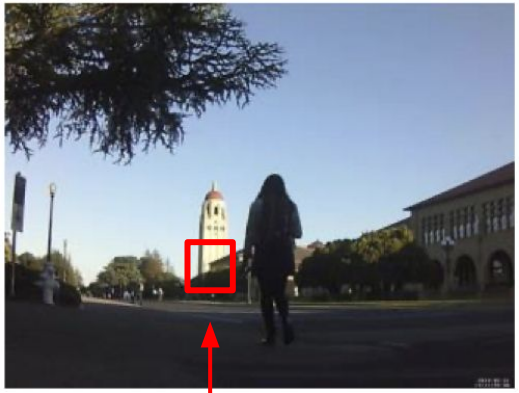


Image1 pixel at (10, 15)
pixel.red → 220
pixel.green → 240
pixel.blue → 190

Image2 pixel at (10, 15)
pixel.red → 0
pixel.green → 10
pixel.blue → 20

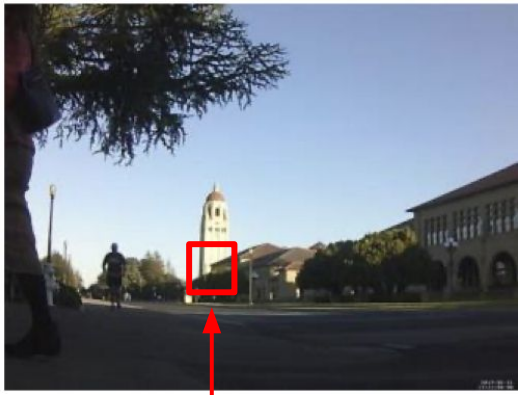
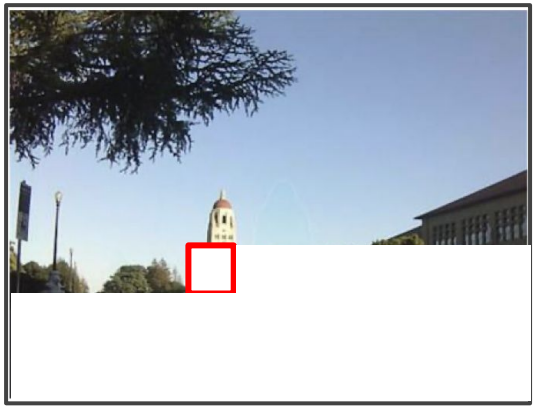


Image3 pixel at (10, 15)
pixel.red → 210
pixel.green → 220
pixel.blue → 140



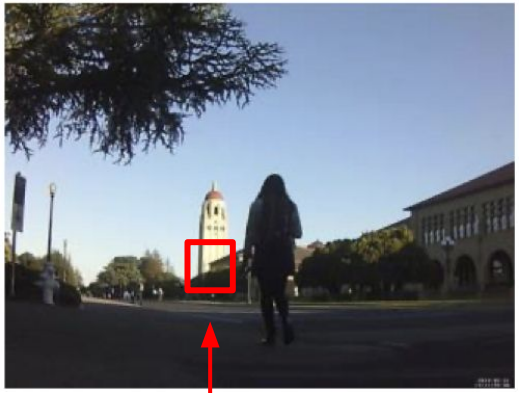


Image1 pixel at (10, 15)
pixel.red → 220
pixel.green → 240
pixel.blue → 190

Image2 pixel at (10, 15)
pixel.red → 0
pixel.green → 10
pixel.blue → 20

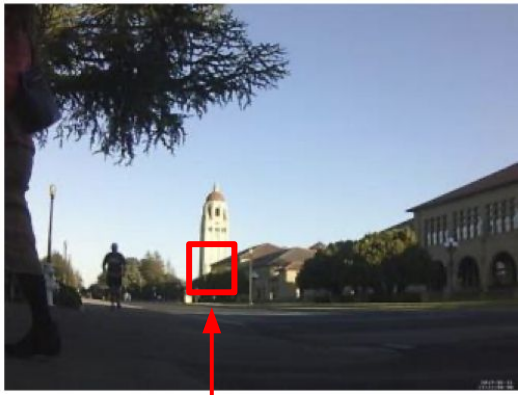
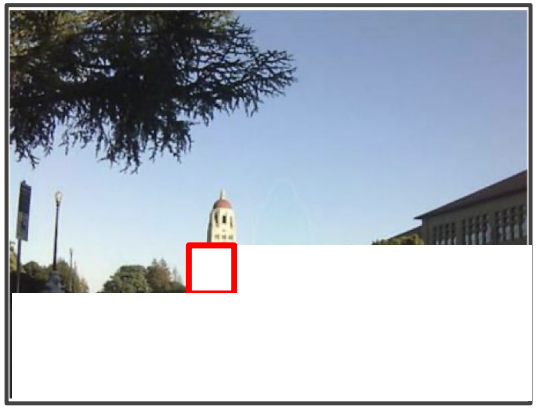


Image3 pixel at (10, 15)
pixel.red → 210
pixel.green → 220
pixel.blue → 140

Average pixel of (10, 15):
pixel.red → 143.33
pixel.green → 156.67
pixel.blue → 116.67



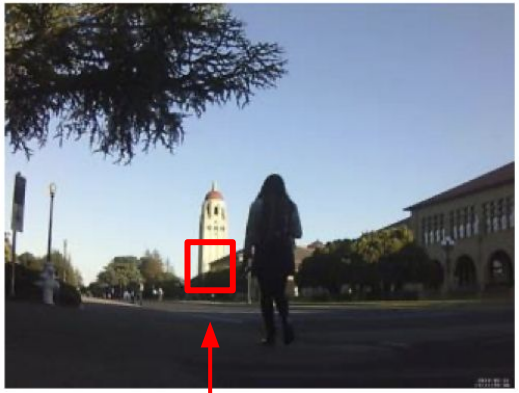


Image1 pixel at (10, 15)
pixel.red → 220
pixel.green → 240
pixel.blue → 190

distance from average: 18200

Image2 pixel at (10, 15)
pixel.red → 0
pixel.green → 10
pixel.blue → 20

distance from average: 51400

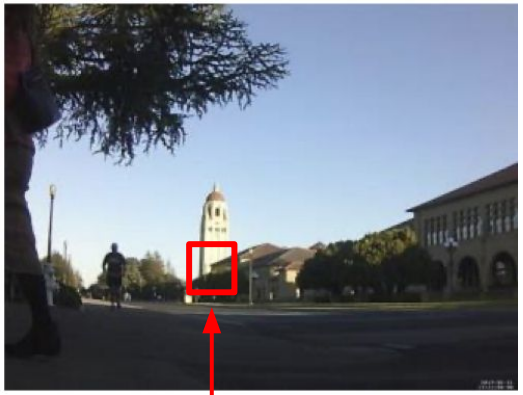
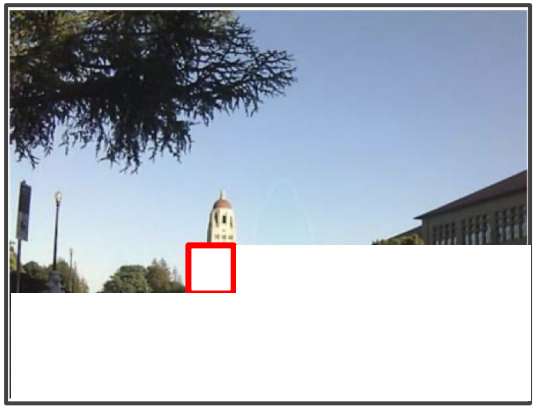


Image3 pixel at (10, 15)
pixel.red → 210
pixel.green → 220
pixel.blue → 140

distance from average: 9000



Average pixel of (10, 15):
pixel.red → 143.33
pixel.green → 156.67
pixel.blue → 116.67

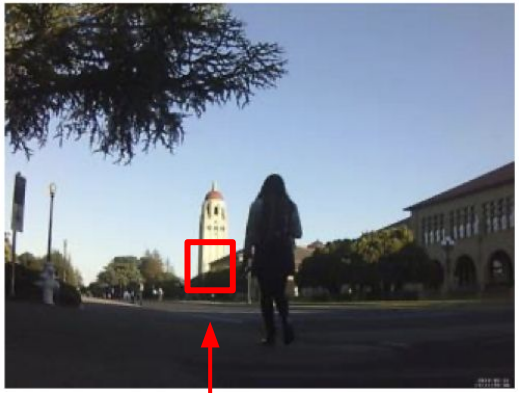


Image1 pixel at (10, 15)
pixel.red → 220
pixel.green → 240
pixel.blue → 190

distance from average: 18200

Image2 pixel at (10, 15)
pixel.red → 0
pixel.green → 10
pixel.blue → 20

distance from average: 51400

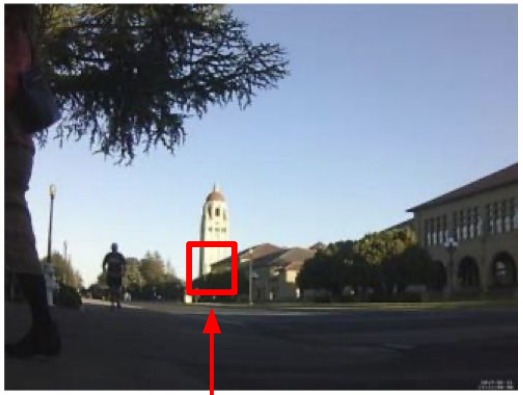
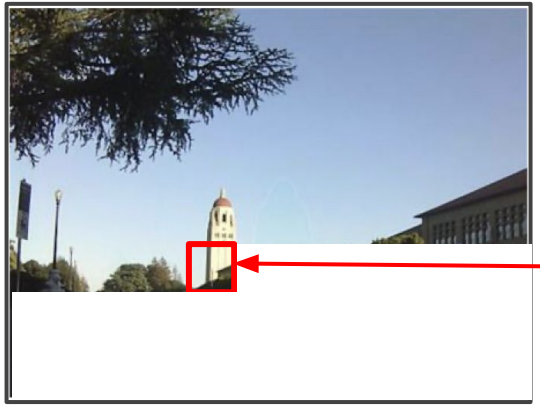


Image3 pixel at (10, 15)
pixel.red → 210
pixel.green → 220
pixel.blue → 140

distance from average: 9000

Average pixel of (10, 15):
pixel.red → 143.33
pixel.green → 156.67
pixel.blue → 116.67



Solution pixel at (10, 15)
pixel.red → 210
pixel.green → 220
pixel.blue → 140

Part 2: Graphics

Overview: Graphics

We utilize a Canvas and can create our drawings and images, unlike reading in an outside image.

We can do things like:

- Draw shapes
- Create patterns

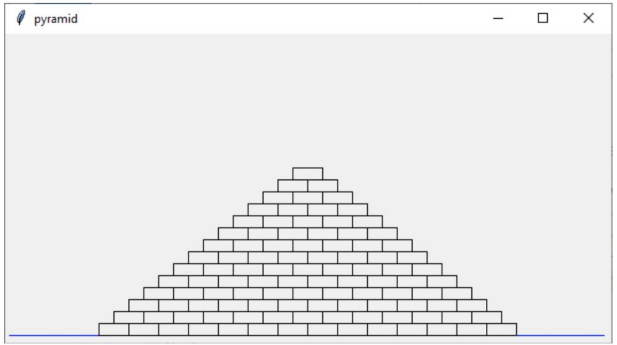
For more detailed information,
check out the Graphics Reference Guide & Lecture 10:Graphics

Pyramid

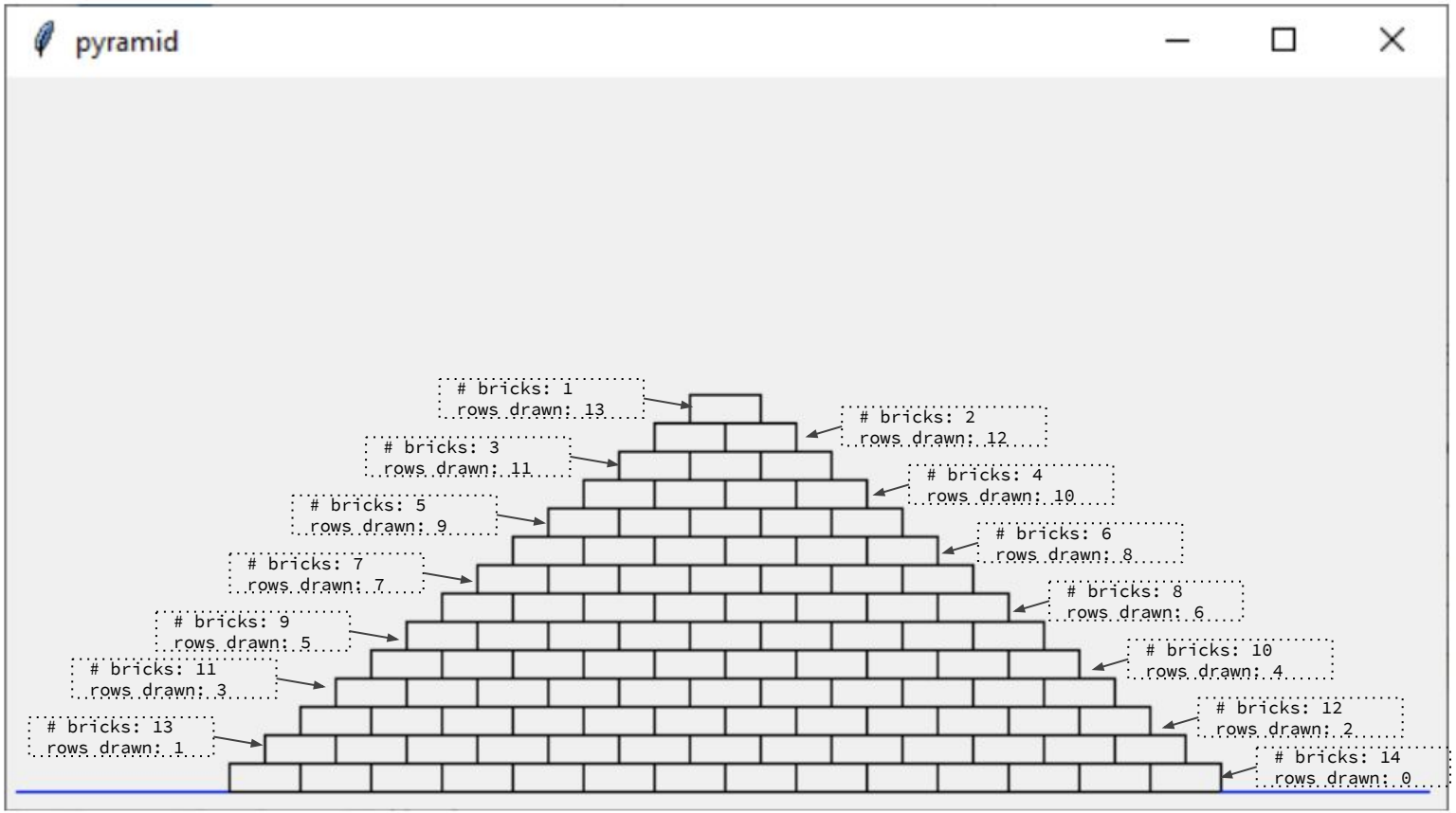
Drawing a Pyramid

Goal: Draw a pyramid with any number of bricks in its base

- Step 1: Draw a single brick
- Step 2: Starting at the bottom of the pyramid, draw *BRICKS_IN_BASE* - *n* bricks, where n is how many rows you've already drawn.



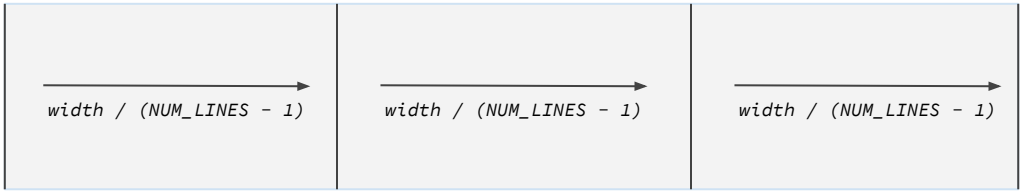
Example with BRICKS_IN_BASE = 14


pyramid
bricks: 1
rows drawn: 13
bricks: 2
rows drawn: 12
bricks: 3
rows drawn: 11
bricks: 4
rows drawn: 10
bricks: 5
rows drawn: 9
bricks: 6
rows drawn: 8
bricks: 7
rows drawn: 7
bricks: 8
rows drawn: 6
bricks: 9
rows drawn: 5
bricks: 10
rows drawn: 4
bricks: 11
rows drawn: 3
bricks: 12
rows drawn: 2
bricks: 13
rows drawn: 1
bricks: 14
rows drawn: 0

Quilt

Task 1: Drawing Bars

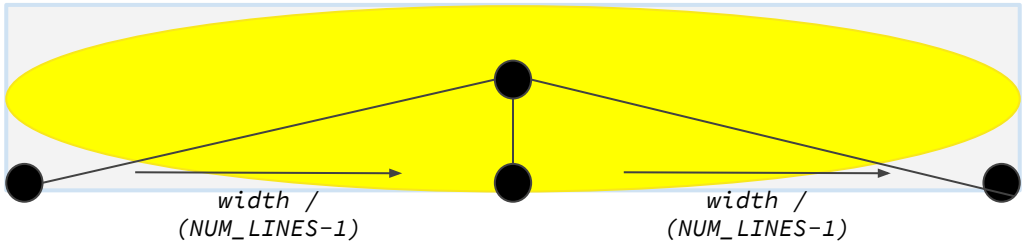
- Step 1: Draw a rectangle of size **width * height** that has its upper left corner at the pixel **(x, y)** with the color **light blue**
- Step 2: Draw **num_lines** evenly spaced lines in the rectangle
  - Starting at the left, each line should be drawn **width / (num_lines - 1)** to the right of the line drawn before it.



NUM_LINES = 4

Task 2: Drawing Eye

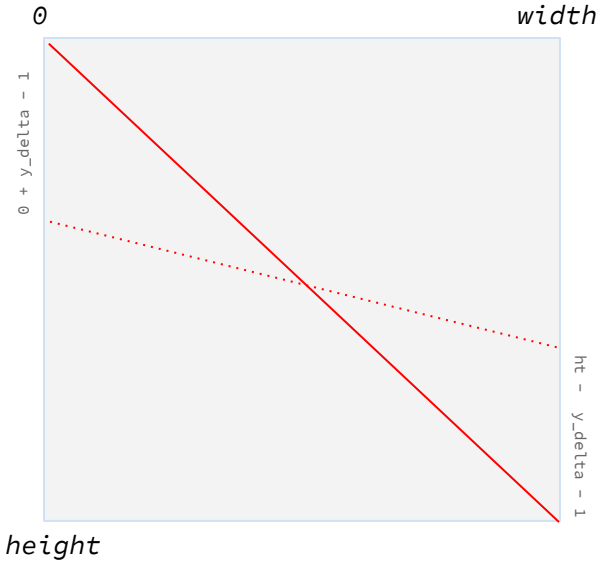
- Step 1: Draw a rectangle of size **width * height** that has its upper left corner at the pixel **(x, y)** with the color **light blue**
- Step 2: Draw a **yellow** oval **width** wide and **height** high, with its top left corner at **(x,y)**
- Step 3: Draw **num_lines** from the center of oval to **num_lines** points, evenly spaced, at bottom of the rectangle
  - Starting at the left, each line's ending point should be drawn **width / (num_lines-1)** to the right of the line drawn before it.



Note: all 3 lines have the same (x_1,y_1)

Task 3: Drawing Bowtie

- Step 1: Draw a light blue rectangle
- Step 2: Draw **num_lines** red lines.
  - All lines have the same x_1 and x_2 (they all start at 0 and end at width - 1!)
  - The height is evenly divided by red lines. For each line, calculate a y_delta distance from the start point.
  - y_1 should be *y_delta* from the top (or *0*), while y_2 should be *y_delta* from the bottom (or *height - 1*)
  - Remember that we subtract one to get the actual final pixel values!

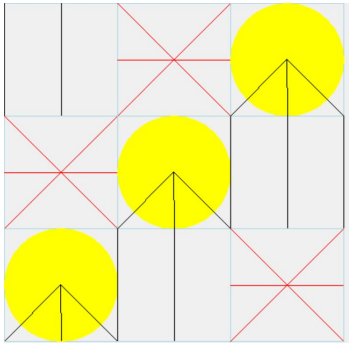


The first line goes from upper left to lower right. The next line will start *y_delta* lower and end *y_delta* higher

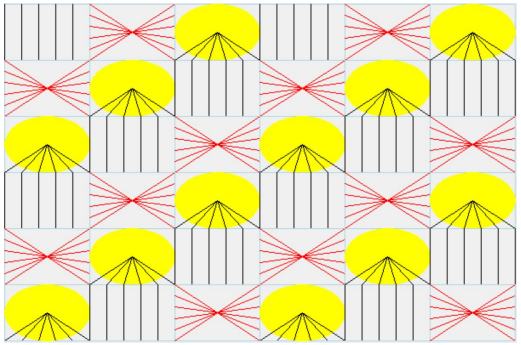
Build the Quilt

- Step 1: Compute the **sub_width** and the **sub_height** of each quilt rectangle. Given **n**, the number of patches per row/column, **sub_width** will be **width** // **n**
- Step 2: Use a double for loop to go through each quilt patch, calculating each patch's top left corner (x,y)
- Step 3: At each patch, draw the bars, the eye, or the bowtie (in rotation)

n=3



n=6



Good luck! :)