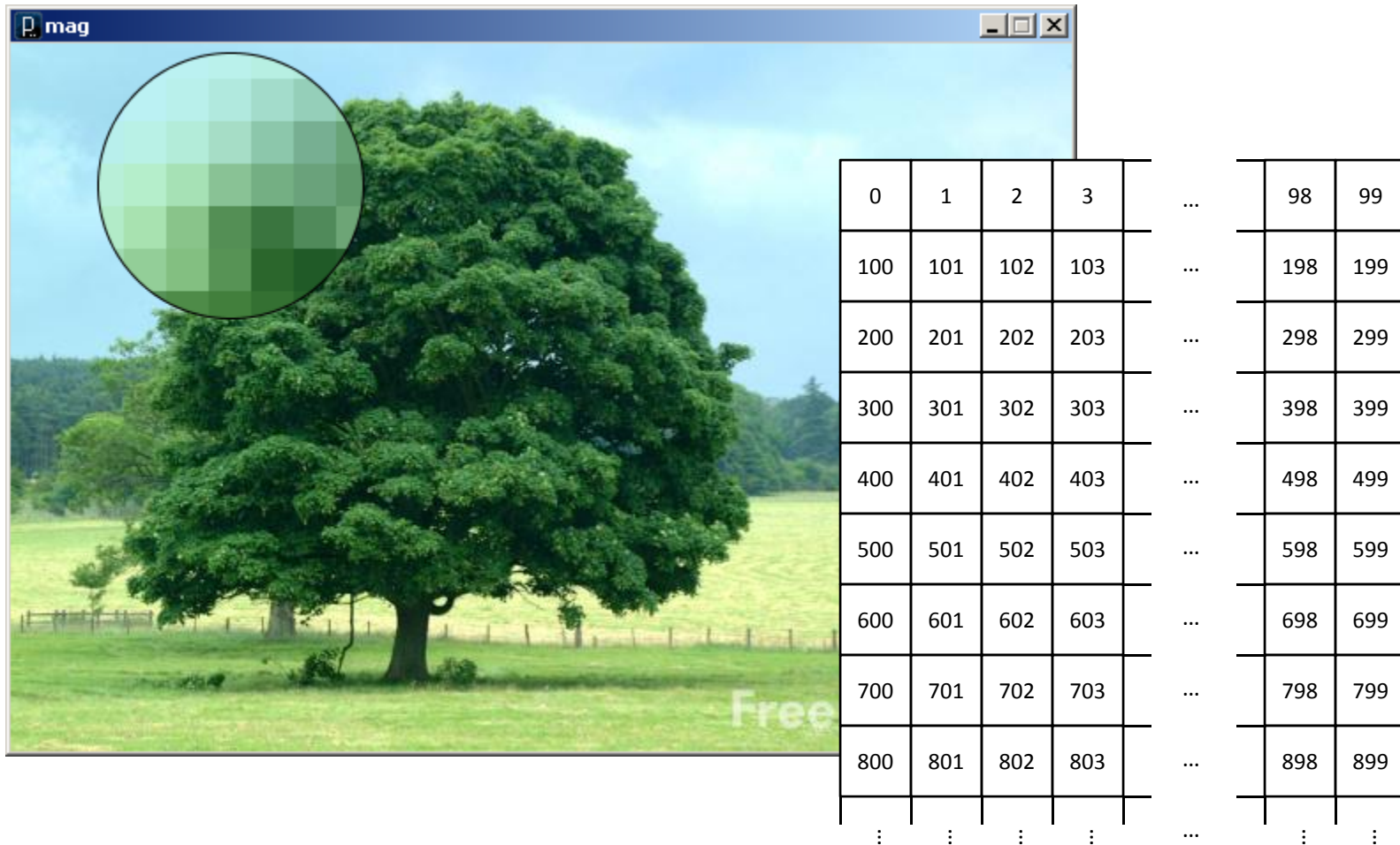


# Image Processing

... computing with and about data,

... where "data" includes the values and relative locations of the colors that make up an image.

# An image is an array of colors



The image shows a window titled "mag" containing a landscape photograph of a large, leafy tree in a green field under a blue sky. A circular inset in the top-left corner of the window shows a magnified view of a small section of the tree, illustrating its pixelated nature. To the right of the window is a grid representing the image's data as an array of colors. The grid consists of 10 rows and 10 columns. The first four columns are numbered 0, 1, 2, and 3. The fifth column contains three dots "...". The sixth and seventh columns are numbered 98 and 99. The eighth, ninth, and tenth columns contain three dots "...". Below the grid, there are three dots "..." under the fifth column, and three dots "..." under the eighth and ninth columns.

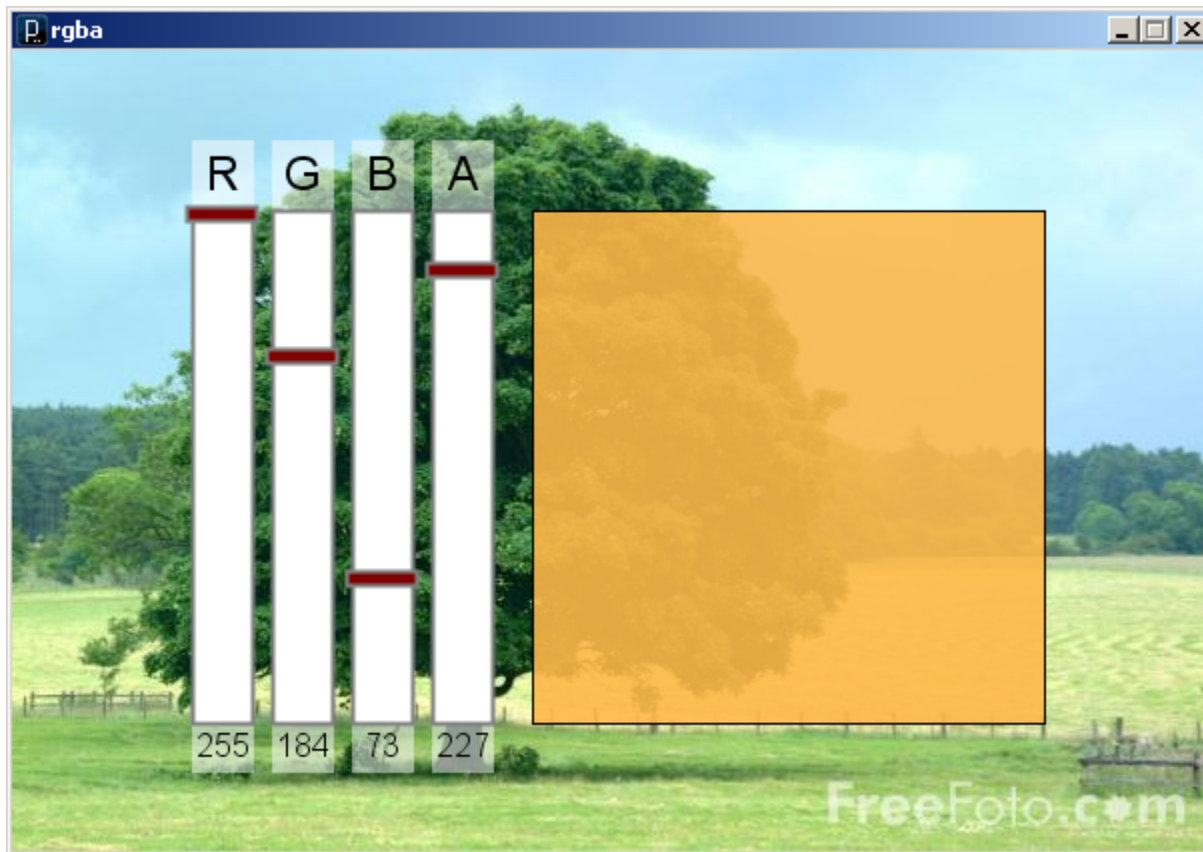
0	1	2	3	...	98	99
100	101	102	103	...	198	199
200	201	202	203	...	298	299
300	301	302	303	...	398	399
400	401	402	403	...	498	499
500	501	502	503	...	598	599
600	601	602	603	...	698	699
700	701	702	703	...	798	799
800	801	802	803	...	898	899
:	:	:	:	...	:	:

Pixel : Picture Element

mag.pde

# Color

- A triple of bytes [0, 255]
  - RGB or HSB
- Transparency (alpha)
  - How to blend a new pixel color with an existing pixel color



rgba.pde

## Accessing the pixels of a sketch

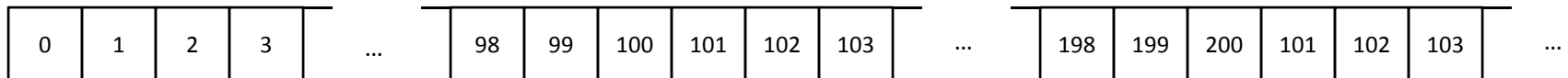
- `loadPixels()`
  - Loads the color data out of the sketch window into a 1D array of colors named `pixels[]`
  - The `pixels[]` array can be modified
- `updatePixels()`
  - Copies the color data from the `pixels[]` array back to the sketch window

# A 100-pixel wide image

- First pixel at index 0
- Right-most pixel in first row at index 99
- First pixel of second row at index 100

0	1	2	3	...	98	99
100	101	102	103	...	198	199
200	201	202	203	...	298	299
300	301	302	303	...	398	399
400	401	402	403	...	498	499
500	501	502	503	...	598	599
600	601	602	603	...	698	699
700	701	702	703	...	798	799
800	801	802	803	...	898	899
⋮	⋮	⋮	⋮	...	⋮	⋮

The pixels[] array is one-dimensional



```
// whiteNoise

void setup() {
  size(400, 300);
}

void draw() {
  float b;

  // Load colors into the pixels array
  loadPixels();

  // Fill pixel array with a random
  // grayscale value
  for (int i=0; i<pixels.length; i++) {
    b = random(0, 255);
    pixels[i] = color(b);
  }

  // Update the sketch with pixel data
  updatePixels();
}
```



See also [colorNoise.pde](#)

## Accessing Pixels as a 2D Array

- Pixels can be accessed as a 2D array using the following formula:

```
index = r * width + c
```

```
index = y * width + x
```

- Using 0-based indices

```
int idx = width*r + c;
```

```
pixels[idx] = color(b);
```

```
// cone
void setup() {
  size(400, 400);

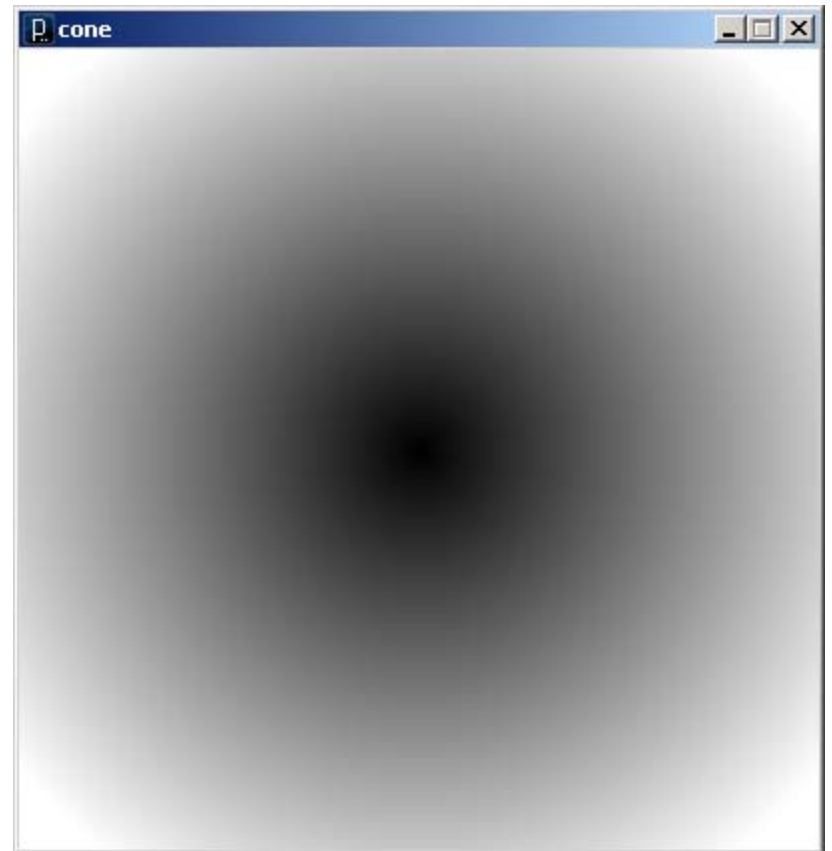
  // Load colors into the pixels array
  loadPixels();

  // Access pixels as a 2D array
  for (int y=0; y<height; y++) {
    for (int x=0; x<width; x++) {

      // Compute distance to center point
      float d = dist(x, y, width/2, height/2);

      // Set pixel as distance to center
      pixels[y*height+x] = color(d);
    }
  }

  // Update the sketch with pixel data
  updatePixels();
}
```





## Rendering Images in a Sketch

- Image data can be loaded from a file using `loadImage()` method, and drawn on a sketch with the `image()` command

```
PImage img = loadImage( "myImage.jpg" );  
image(img, 0, 0);
```

- The `PImage` object also permits individual pixel color data to be modified.
  - like the sketch window

# PImage

## Fields

- width - the width of the image
- height - the height of the image
- pixels[] - the image pixel colors  
(after a call to loadPixels())

# PImage

## Methods

loadPixels()

Loads the color data out of the PImage object into a 1D array of colors named pixels[].

updatePixels()

Copies the color data from the pixels[] array back to the PImage object.

## Also

red(color)      extract the red component of from color

blue(color)     extract the green component from a color

green(color)    extract the blue component from a color

# Basic Filters

- Color
  - Extracting Red/Green/Blue colors
    - `pixels[i] = color(red(c), 0, 0);`
    - `pixels[i] = color(0, 0, blue(c));`
  - Grayscale
    - `pixels[i] = color(0.3*red(c)+ 0.59*green(c)+ 0.11*blue(c));`
  - Negative
    - `pixels[i] = color(255-red(c), 255-green(c), 255-blue(c));`

# Sepia

- Technique for archiving BW photos
  - float r =  
 $\text{red}(c)*0.393+\text{green}(c)*0.769+\text{blue}(c)*0.189;$
  - float g =  
 $\text{red}(c)*0.349+\text{green}(c)*0.686+\text{blue}(c)*0.168;$
  - float b =  
 $\text{red}(c)*0.272+\text{green}(c)*0.534+\text{blue}(c)*0.131;$
  - pixels[i] = color(r, g, b);

# PImage

## Methods (Cont'd)

- get(...) Reads the color of any pixel or grabs a rectangle of pixels
- set(...) Writes a color to any pixel or writes an image into another
- copy(...) Copies pixels from one part of an image to another
- mask(...) Masks part of the image from displaying
- save(...) Saves the image to a TIFF, TARGA, PNG, or JPEG file
- resize(...) Changes the size of an image to a new width and height
- blend(...) Copies a pixel or rectangle of pixels using different blending modes
- filter(...) Processes the image using one of several algorithms

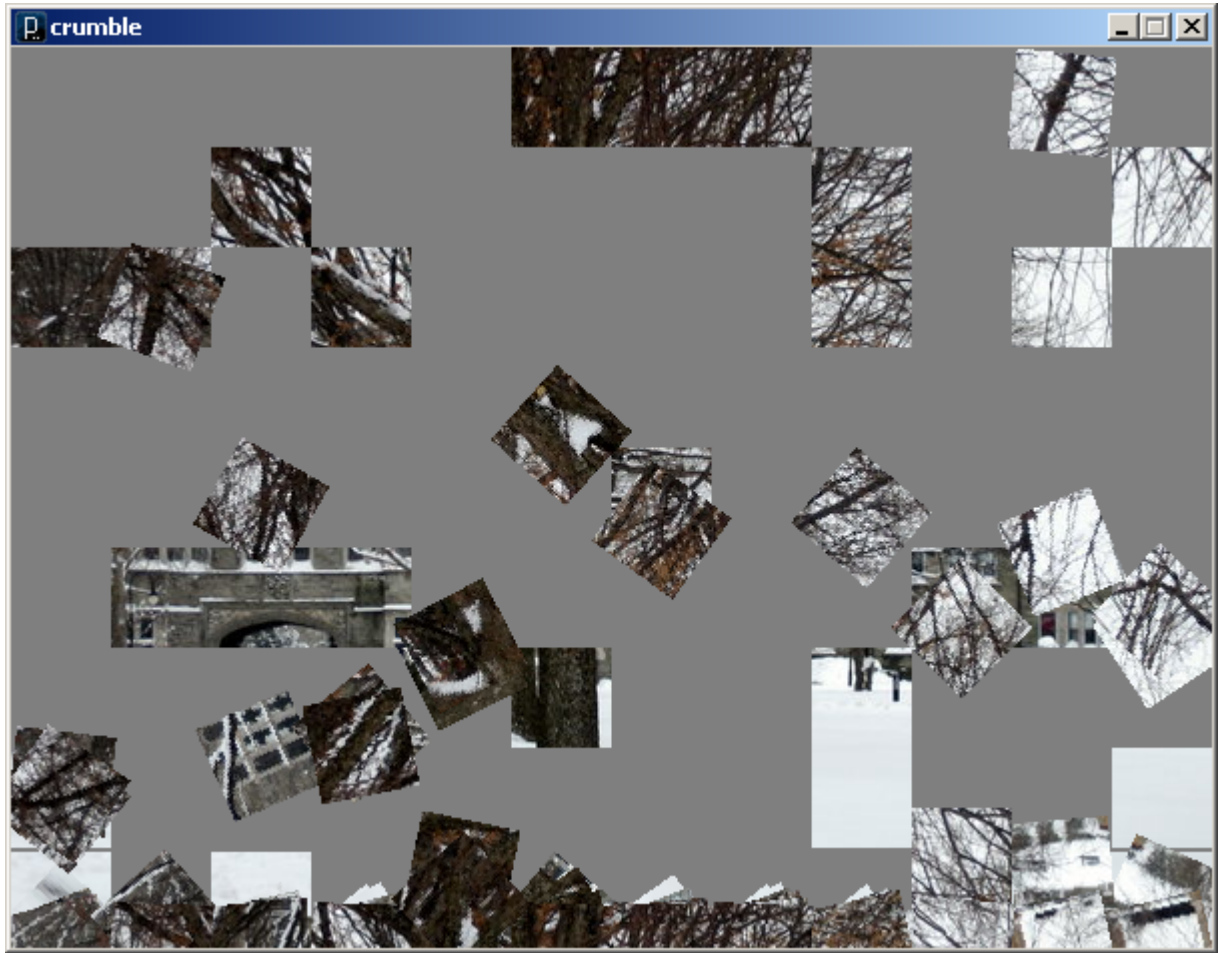
## get(...)

- Get a single pixel (very slow)

```
Color c = img.get(x, y);
```

- Get a rectangular range of pixels

```
PImage img2 = img.get(x, y, w, h);
```





## tint(...) / noTint()

- tint() modifies the fill value for images

```
tint( gray );
```

```
tint( gray, alpha );
```

```
tint( red, green, blue );
```

```
tint( red, green, blue, alpha );
```

- Turn off applied tint() values with noTint()

```
// warhol

void setup() {
  // Load the image three times
  PImage warhol = loadImage("andy-warhol2.jpg");
  size(warhol.width*3, warhol.height);

  // Draw modified images
  tint(255, 0, 0);
  image(warhol, 0, 0);

  tint(0, 255, 0);
  image(warhol, 250, 0);

  tint(0, 0, 255);
  image(warhol, 500, 0);
}
```

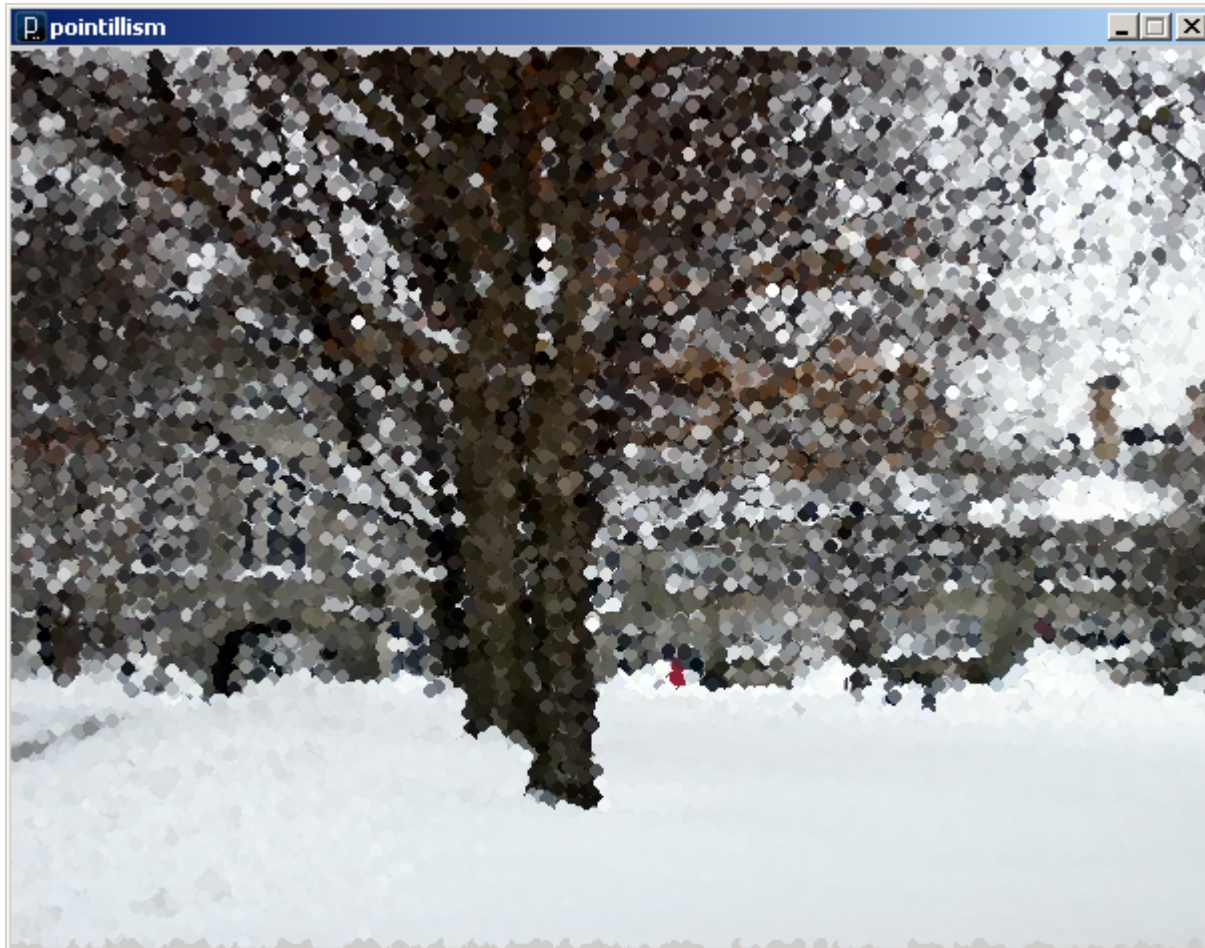


```
// pointillism
PImage img;

void setup() {
  img = loadImage("bmc3.jpg");
  image(img.width, img.height);
  imageMode(CENTER);
  image(img, width/2, height/2);
  noStroke();
  ellipseMode(CENTER);
  loadPixels();          // Cover with random circles
  for (int i=0; i<20000; i++) addPoint();
}

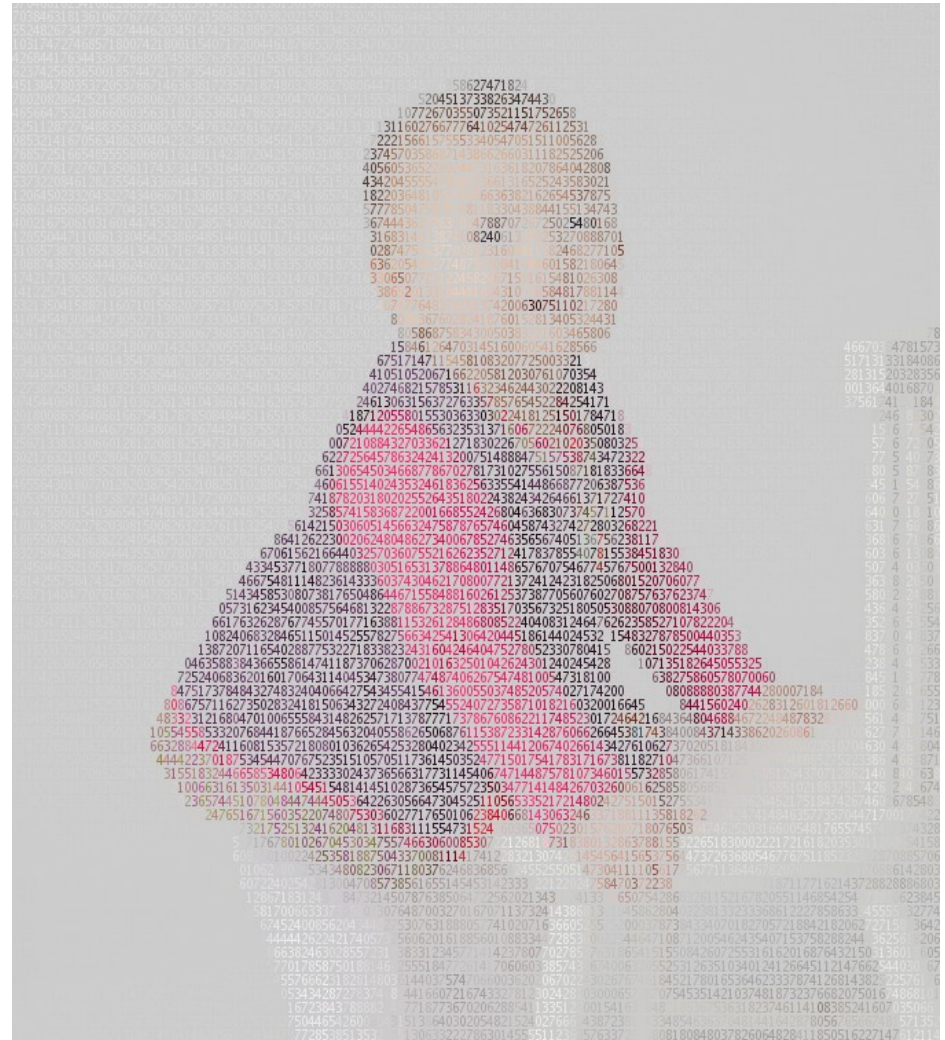
void addPoint() {
  // Add a random filled circle to image
  int x = (int)random(width);
  int y = (int)random(height);
  int i = x + width*y;
  color c = pixels[i];
  fill(c);
  ellipse(x, y, 7, 7);
}

void draw() {
  //addPoint();
}
```



# Simple Image Visualization

- Sample pixel colors every n pixels
- Draw a grid of basic shapes (ellipse, rect, line, triangle, text, etc) using the sampled color as fill color or stroke color



# Obamicon

