# Images & Pixels (processing)

Yue Mao

2012.5.29

# - How are images organized?

- We are familiar with the idea of each pixel on the screen having an X and Y position in a two dimensional window. However, the array pixels has only one dimension, storing color values in linear sequence.
- The processing has only one value for each pixel, and always start with zero.

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 | 9 |
| 10 | 11 | 12 | 13 | 14 |
| 15 | 16 | 17 | 18 | 19 |
| 20 | 21 | 22 | 23 | 24 |

How the pixels are stored:

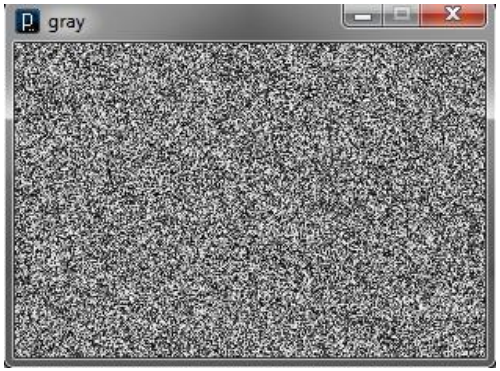
| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | . | . | . | | |

# How can we access individual pixels?

## Any Visual information on a computer is comprised of pixel information.

### Basic coding structure Template (example1)

```
size(300,200);   // Before we deal with pixels
loadPixels();  //this function is called before we access the pixel array

// Loop through every pixel
for (int i = 0; i < pixels.length; i++ ) { // We can get the length of the pixels array just like with any array.

  // Pick a random number, 0 to 255
  float rand = random(255);

  // Create a grayscale color based on random number
  color c = color(rand);

  // Set pixel at that location to random color
  pixels[i] = c; // We can access individual elements of the pixels array via an index, just like with any other array.
}

// When we are finished dealing with pixels
updatePixels(); //this function is called after we finish with the pixel array.
```



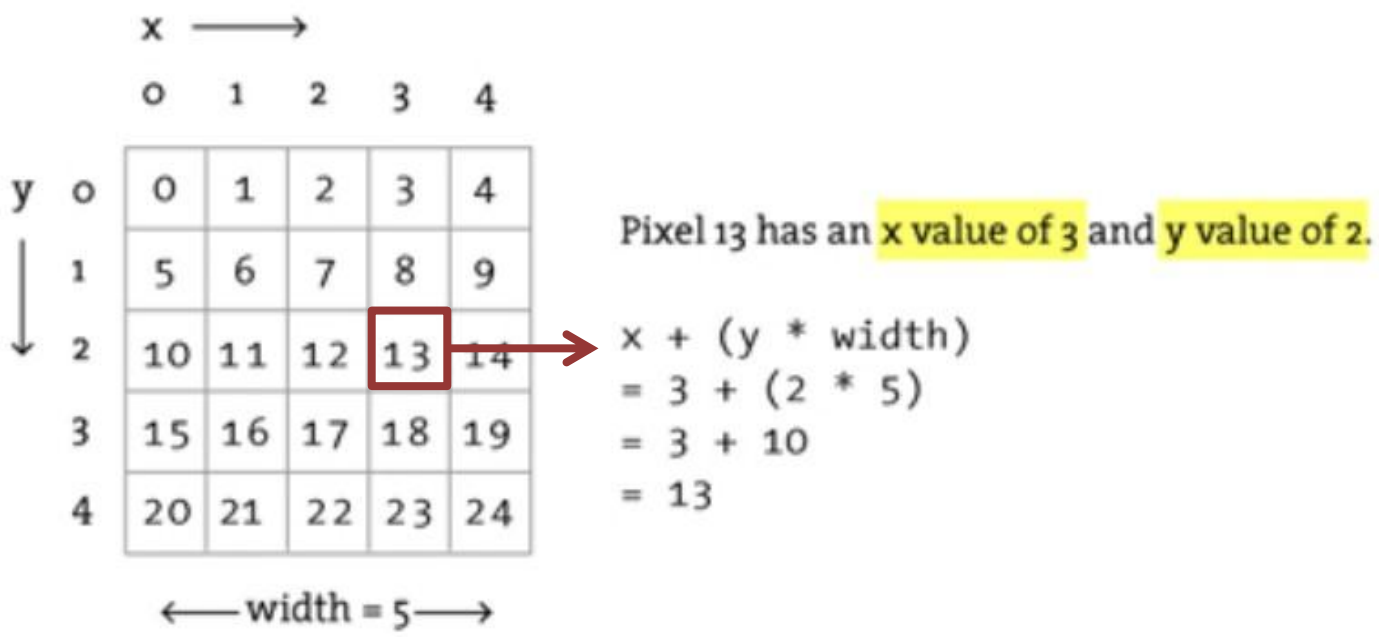
**Tips:**

**loadPixels()** This function is called before you access the pixel array, saying "load the pixels, I would like to speak with them!“

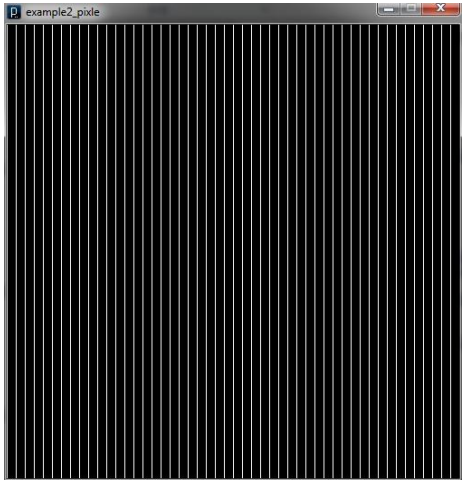
**updatePixels()** This function is called after you finish with the pixel array saying "Go ahead and update the pixels, I'm all done!“

**Formula: What is a pixel's house number?**
**Every pixel in an image is living in its exclusive house.**

**LOCATION = X + Y*WIDTH**



# Every pixel in an image is living in its exclusive house.(Example 2)




```
size(500,500);
loadPixels();
 // Two loops allow us to visit every column (x) and every row (y).

// Loop through every pixel column
for (int x = 0; x < width; x++ ) {
  // Loop through every pixel row
  for (int y = 0; y < height; y++ ) {
    // Use the formula to find the 1D location
    int loc = x + y * width;
// The location in the pixel array is calculated via our formula: 1D pixel location = x + y * width

    if (x%10== 0) { // If we are an even column..%→modulo,相除后的余数为零，即可除尽
      pixels[loc] = color(255); //set color as white
    } else { // If we are an odd column
      pixels[loc] = color(0); //   We use the column number (x) to determine whether the color should be black or white.
    }
  }
}
updatePixels();
```

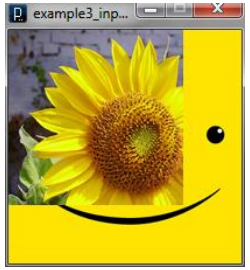
# loadImage VS loadPixel

## (Example 3)


```
PImage a;
void setup() {
  size(200,200);
  // Make a new instance of a PImage by loading an image file
  a = loadImage("smile.jpg");
}

void draw() {
  background(0);
  // Draw the image to the screen at coordinate (0,0)
  image(a,0,0);  //(pic,Xposition,Yposition)
}
```

```
PImage img;

void setup() {
  size(200, 200);
  img = loadImage("smile.jpg");
}
void draw() {
  loadPixels();
img.loadPixels();
  for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
      int loc = x + y*width;
      // The functions red(), green(), and blue() pull out the 3 color components from a pixel.
      float r = red(img.pixels[loc]);
      float g = green(img.pixels[loc]);
      float b = blue(img.pixels[loc]);

      // Image Processing would go here
      // If we were to change the RGB values, we would do it here, before setting the pixel in the display window.
      // Set the display pixel to the image pixel
      pixels[loc] =  color(r,g,b);
    }
  }
  updatePixels();  }
```

# To understand the difference




```
PImage img;
PImage bg;  //2 layers

void setup() {
  size(200,200);
  img = loadImage("sunflower.jpg");
  bg=loadImage("smile.jpg");
}
void draw() {
  background(bg);
  tint(255,127);

loadPixels();
```

```
 img.loadPixels();
  for (int y = 0; y < 150; y++ ) {
    for (int x = 0; x < 150; x++ ) {
      int loc = x + y*width;
      // The functions red(), green(), and
blue() pull out the three color
components from a pixel.
      float r = red(img.pixels [loc]);
      float g = green(img.pixels[loc]);
      float b = blue(img.pixels[loc]);
      pixels[loc] = color(r,g,b);
    }
  }
  updatePixels();
}
```

**Since the pixels are editable, it gives us much more room to try, such as filter.**

# Example4: Brightness Threshold


```
PImage source;     // Source image
PImage destination; // Destination image

void setup() {
 size(200,200);
 source = loadImage("sunflower.jpg");
 destination = createImage(source.width, source.height, RGB);
}

void draw() {
 float threshold = 200;//设定一个阈值或者叫临界值

 // We are going to look at both image's pixels
 source.loadPixels();
 destination.loadPixels();

 for (int x = 0; x < source.width; x++ ) {
  for (int y = 0; y < source.height; y++ ) {
    int loc = x + y*source.width;
    // Test the brightness against the threshold
    if (brightness(source.pixels[loc]) > threshold){
      destination.pixels[loc] = color(255); // White
    } else {
      destination.pixels[loc] = color(0);   // Black
    }
  }
 }

 // We changed the pixels in destination
 destination.updatePixels();
 // Display the destination，显示去色后的图片
 image(destination,0,0);
}
```

# Example5: Interactive Brightness

```
PImage img;

void setup() {
  size(200,200);
  img = loadImage("sunflower.jpg");
}

void draw() {
  loadPixels();

  for (int x = 0; x < img.width; x++ ) {
    for (int y = 0; y < img.height; y++ ) {

      // Calculate the 1D pixel location
      int loc = x + y*img.width;
      // Get the R,G,B values from image
      float r = red (img.pixels[loc]);
      float g = green (img.pixels[loc]);
      float b = blue (img.pixels[loc]);

      // We calculate a multiplier ranging from 0.0 to 8.0 based on mouseX position.
      // That multiplier changes the RGB value of each pixel.
      float adjustBrightness = ((float) mouseX / width) * 8.0;
      r *= adjustBrightness;
      g *= adjustBrightness;
      b *= adjustBrightness;
```


```
      // The RGB values are constrained between 0 and 255 before being set as a new color.
      r = constrain(r,0,255); //constrain(value, min, max)
      g = constrain(g,0,255);
      b = constrain(b,0,255);
    // Make a new color and set pixel in the window
      color c = color(r,g,b);
      pixels[loc] = c;
    }
  }
  updatePixels();
}
```

# Example6:Edge detector

```
PImage img;       // Source image
PImage destination;   // Destination image

void setup() {
 size(200,200);
 img = loadImage("sunflower.jpg");
 destination = createImage(img.width, img.height, RGB);
}

void draw() {
 // We are going to look at both image's pixels
 img.loadPixels();
 destination.loadPixels();  // Since we are looking at left neighbors
 // We skip the first column
 for (int x = 1; x < width; x++ ) {
  for (int y = 0; y < height; y++ ) {

   // Pixel location and color
   int loc = x + y*img.width;
   color pix = img.pixels[loc];

   // Pixel to the left location and color
   int leftLoc = (x - 1) + y*img.width;
   color leftPix = img.pixels[leftLoc];

   // New color is difference between pixel and left neighbor
   float diff = abs(brightness(pix) - brightness(leftPix)); //abs:绝对值absolute value
   destination.pixels[loc] = color(diff);
  }
 }

 // We changed the pixels in destination
 destination.updatePixels();
 // Display the destination
 image(destination,0,0);
}
```


This example is a simple horizontal edge detection algorithm算法. When pixels differ greatly from their neighbors, they are most likely "edge" pixels.

## Relative location of a pixel
## --a much more sophisticated algorithms to define a pixel.

A Pixel and its Friendly Neighbors:

| | | |
|---|---|---|
| | | |
| (x-1,y) | (x,y) | |
| | | (x+1,y+1) |

- Each pixel has 8 immediate neighbors: top left, top, top right, right, bottom right, bottom, bottom left, left.
- These image processing algorithms are often referred to as a "spatial convolution.“
- The process uses a weighted average of an input pixel and its neighbors to calculate an output pixel.

- **For example,**

- Sharpen:
- -1 -1 -1
- -1 9 -1
- -1 -1 -1

we "sharpen" an image by subtracting the neighboring pixel values and increasing the center point pixel.

- Blur:
- 1/9 1/9 1/9
- 1/9 1/9 1/9
- 1/9 1/9 1/9

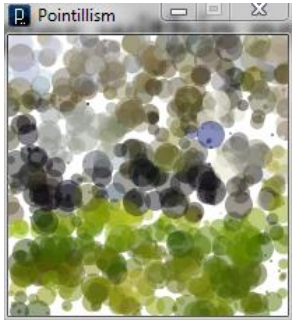
A blur is achieved by taking the average of all neighboring pixels.

# Example7： sharpen


```
PImage img;
int w = 80;
// The convolution matrix for a "sharpen" effect stored as a 3 x 3 two-dimensional array.
float[][] matrix = { { -1, -1, -1 } ,
                     { -1, 9, -1 } ,
                     { -1, -1, -1 } } ;//矩阵

void setup() {
  size(200,200);
  img = loadImage( "sunflower.jpg" );
}
void draw() {
   // We're only going to process a portion of the image
  // so let's set the whole image as the background first
  image(img,0,0);

  // In this example we are only processing a section of the image-an 80 x 80 rectangle around the mouse location.
  int xstart = constrain(mouseX-w/2,0,img.width); //mouse minus
  int ystart = constrain(mouseY-w/2,0,img.height);
  int xend = constrain(mouseX + w/2,0,img.width); //mouse plus
  int yend = constrain(mouseY + w/2,0,img.height);
  int matrixsize = 3; //矩阵大小为3*3
  loadPixels();
  // Begin our loop for every pixel
  for (int x = xstart; x < xend; x++ ) {
    for (int y = ystart; y < yend; y++ ) {
      // Each pixel location (x,y) gets passed into a function called convolution()
      // The convolution() function returns a new color to be displayed.
      color c = convolution(x,y,matrix,matrixsize,img);
      int loc = x + y*img.width;
//to draw this rectangle with the original picture
      pixels[loc] = c;
    }
  }
  updatePixels();
  stroke(0);  //black stroke
  noFill();
  rect(xstart,ystart,w,w);
}
```

```
color convolution(int x, int y, float[][] matrix, int matrixsize, PImage img) { //define the related parameters
 float rtotal = 0.0;
 float gtotal = 0.0;
 float btotal = 0.0;
 int offset = matrixsize / 2;

 // Loop through convolution matrix
 for (int i = 0; i < matrixsize; i++ ) {
  for (int j = 0; j < matrixsize; j++ ) {

   // What pixel are we testing
   int xloc = x + i-offset;
   int yloc = y + j-offset;
   int loc = xloc + img.width*yloc;

   // Make sure we haven't walked off the edge of the pixel array
   // It is often good when looking at neighboring pixels to make sure we have not gone off the edge of the pixel array by accident.
   loc = constrain(loc,0,img.pixels.length-1);

   // Calculate the convolution
   // We sum all the neighboring pixels multiplied by the values in the convolution matrix.
   rtotal += (red(img.pixels[loc]) * matrix[i][j]);
   gtotal += (green(img.pixels[loc]) * matrix[i][j]);
   btotal += (blue(img.pixels[loc]) * matrix[i][j]);
  }
 }

 // Make sure RGB is within range
 rtotal = constrain(rtotal,0,255);
 gtotal = constrain(gtotal,0,255);
 btotal = constrain(btotal,0,255);

 // Return the resulting color
 return color(rtotal,gtotal,btotal);
}
```

# Examples within processing: Pointillism




Processing—File—Standard examples– Image—Pointillism

```
PImage img;
int smallPoint = 2;
int largePoint;
int top, left;

void setup() {
  size(200, 200);
  img = loadImage("eames.jpg");  // an alternative image
  noStroke();
  background(255);
  smooth();
  largePoint = min(width, height) / 10;   // center the image on the screen
  left = (width - img.width) / 2;
  top = (height - img.height) / 2;
}

void draw() {
  float pointillize = map(mouseX, 0, width, smallPoint, largePoint);
  int x = int(random(img.width));
  int y = int(random(img.height));
  color pix = img.get(x, y);
  fill(pix, 128);   //alpha=128
  ellipse(left + x, top + y, pointillize, pointillize);
}
```

```
//Re-maps a number from one range to another.
 //map(value, low1, high1, low2, high2)
 //value   float: The incoming value to be converted
 //low1    float: Lower bound of the value's current range
 //high1   float: Upper bound of the value's current range
 //low2    float: Lower bound of the value's target range
 //high2   float: Upper bound of the value's target range
```

Thanks