+

Image Processing

+

Review

- function parts:
  - return type
  - name
  - parameters
  - body
- return type
  - void
  - int, float, boolean, etc.
  - int[], float[], etc.
- name
  - describes the function purpose
- parameters
  - no parameters
  - multiple parameters
  - one array parameter
  - array parameter with a non-array parameter
- body
  - does the work
  - no parameters means the caller has no control of how the body executes
  - as a rule: parameters should be used by the body, not assigned in the body.

+

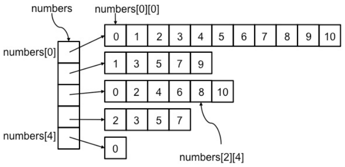
2D Array as an array of arrays

- Each element of a 2D array is a 1D array
- Thus each element of a 2D array has a length
- Declaration can be tiered:
  - `float[][] vals;`
  - `float[20][] vals;`
  - `float[20][300] vals;`
- Each element array does not have to be the same length

+

Ragged Arrays

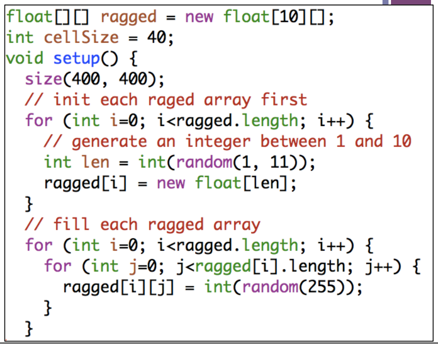
```
int[][] numbers = {
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
{1, 3, 5, 7, 9},
{0, 2, 4, 6, 8, 10},
{2, 3, 5, 7},
{0},
};
```



+

Example

- ragged



```
float[][] ragged = new float[10][];
int cellSize = 40;
void setup() {
  size(400, 400);
  // init each raged array first
  for (int i=0; i<ragged.length; i++) {
    // generate an integer between 1 and 10
    int len = int(random(1, 11));
    ragged[i] = new float[len];
  }
  // fill each ragged array
  for (int i=0; i<ragged.length; i++) {
    for (int j=0; j<ragged[i].length; j++) {
      ragged[i][j] = int(random(255));
    }
  }
```

+

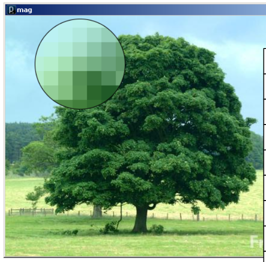
Challenge

- Recall the graySquares example
- Modify to plot black squares whenever both the row and column indices of a cell are even and white otherwise.

Image Processing

- … computing with and about data,
- … where "data" includes the values and relative locations of the colors that make up an image.

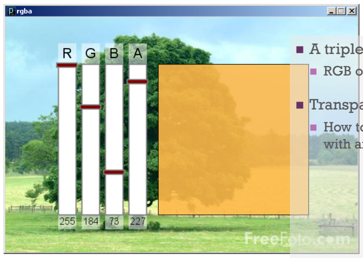
An image is an array of colors



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

Color



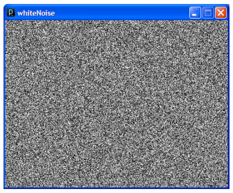
- A triple of bytes [0, 255]
  - RGB or HSB
- Transparency (alpha)
  - How to blend a new pixel color with an existing pixel color

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

Your Canvas as an Image

```
// 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

Useful Color functions

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

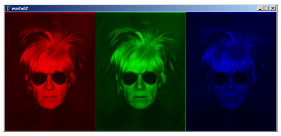
tint/noTint()

- 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()

```
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);
}
```



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 = red(c)*0.393+green(c)*0.769+blue(c)*0.189;
  - float g = red(c)*0.349+green(c)*0.686+blue(c)*0.168;
  - float b = red(c)*0.272+green(c)*0.534+blue(c)*0.131;
  - pixels[i] = color(r, g, b);

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

0	1	2	3	…	98	99	100	101	102	103	…	198	199	200	101	102	103	…

Accessing Pixels as a 2D Array

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

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

- Using 0-based indices

```
int idx      = width * row + column;
pixels[idx] = color(b);
```