Review

- We can declare an array of any type, even other arrays
- A 2D array is an “array of arrays”

```
float[][] myFloats = new float[10][20];
```

- All elements of a 2D array can be accessed using nested loops

```
for (int i=0; i<10; i++) {
  for (int j=0; j<20; j++) {
    myFloats[i][j] = random(100);
  }
}
```

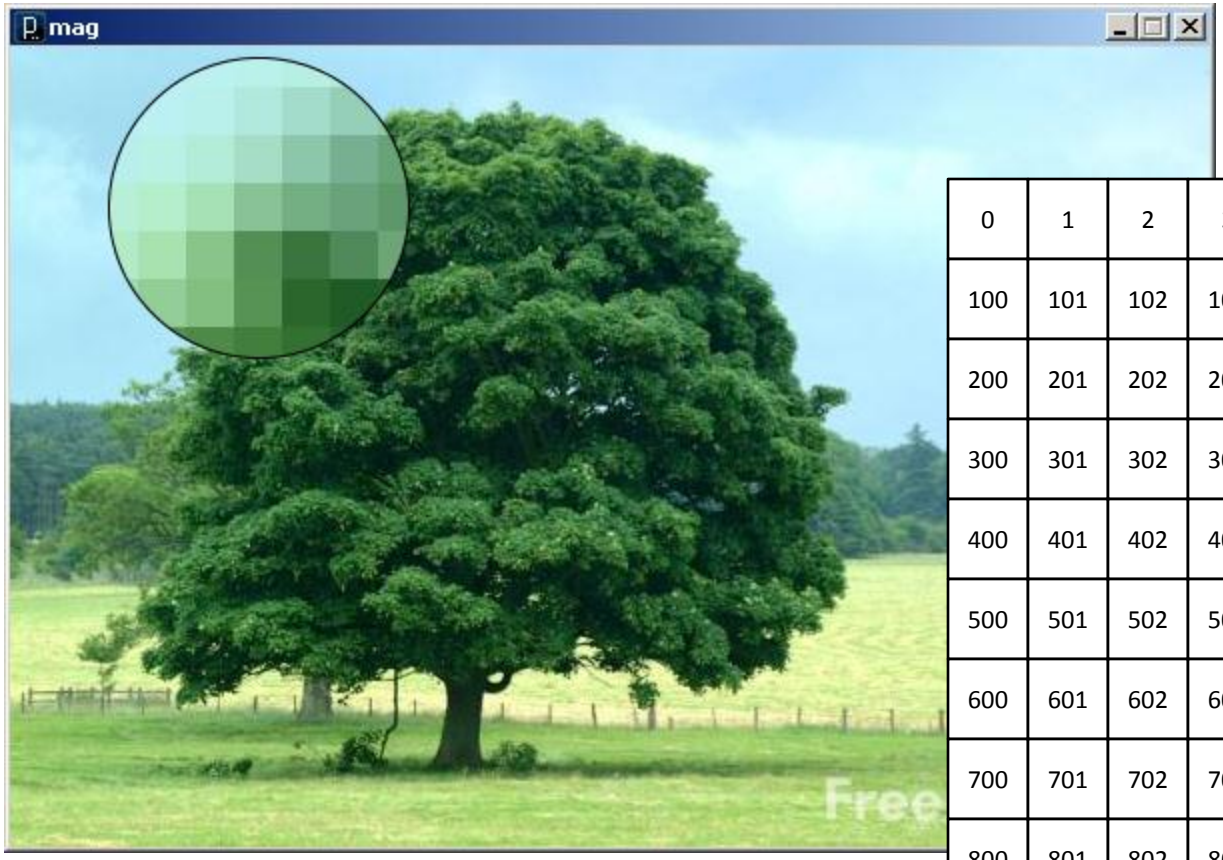
- A ragged array is an array of nonuniformly sized arrays

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



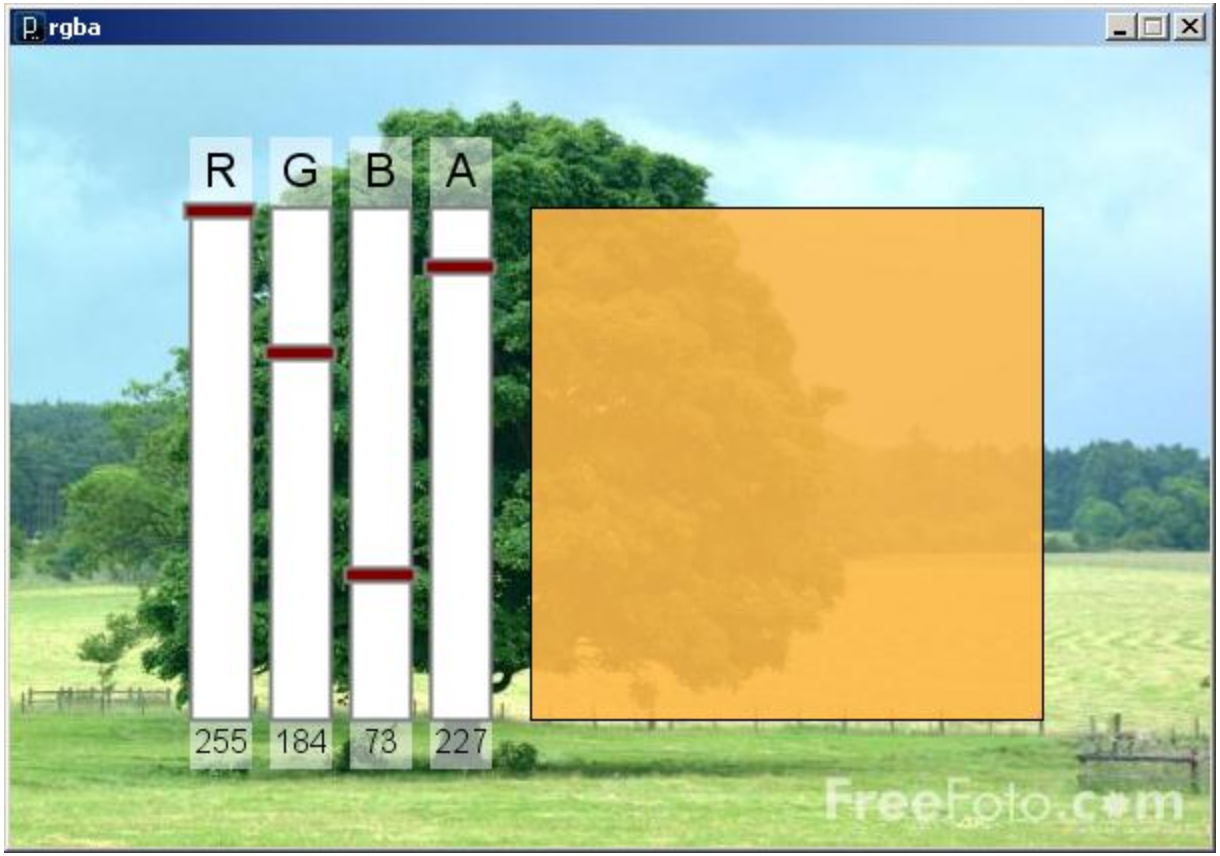
0	1	2	3		…	98	99
100	101	102	103		…	198	199
200	201	202	203		…	298	299
300	301	302	303		…	398	399
400	401	402	403		…	498	499
500	501	502	503		…	598	599
600	601	602	603		…	698	699
700	701	702	703		…	798	799
800	801	802	803		…	898	899
⋮	⋮	⋮	⋮		…	⋮	⋮

Pixel : Picture Element

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

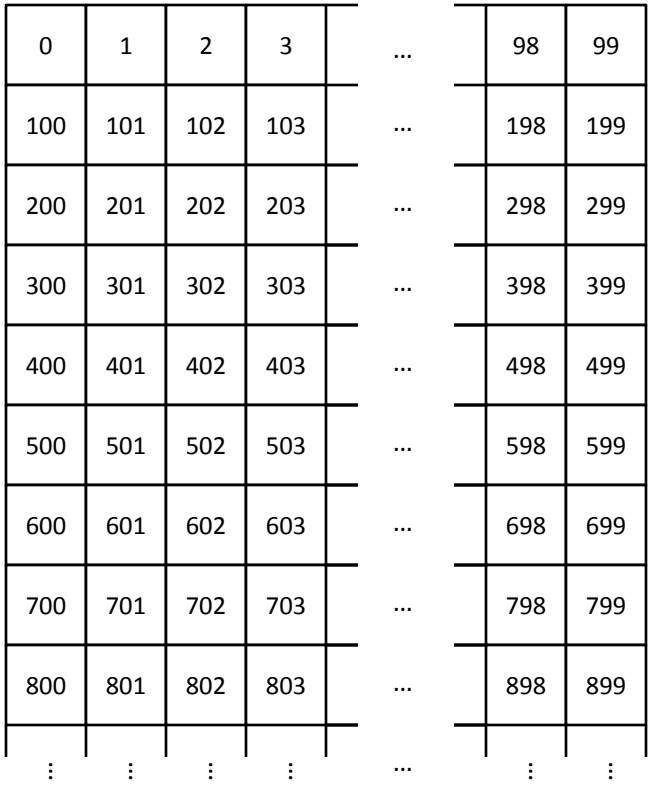
Accessing the pixels of a sketch

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

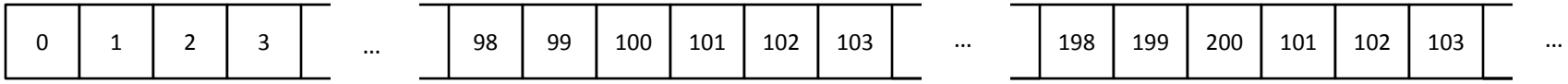
- updatePixels()
  - Copies the color data from the pixels[] array back to the sketch window

A 100-pixel wide image

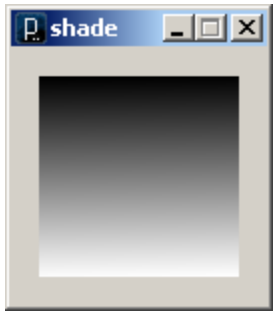
- First pixel at index 0
- Right-most pixel in first row at index 99
- First pixel of second row at index 100



The pixels[] array is one-dimensional



```
// shade
void setup() {
  size(100, 100);

  float b = 0.0;

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

  // Fill pixel array with a grayscale value
  // based on pixel array index
  for (int i=0; i<10000; i++) {
    b = map(i, 0, 10000, 0, 255);
    pixels[i] = color(b);
  }

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



```
// whiteNoise
int nPixels;

void setup() {
  size(400, 300);
  nPixels = width*height;
}

void draw() {
  float b;

  // Load colors into pixels array
  loadPixels();

  // Fill pixel array with a random
  // grayscale value
  for (int i=0; i<nPixels; 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:

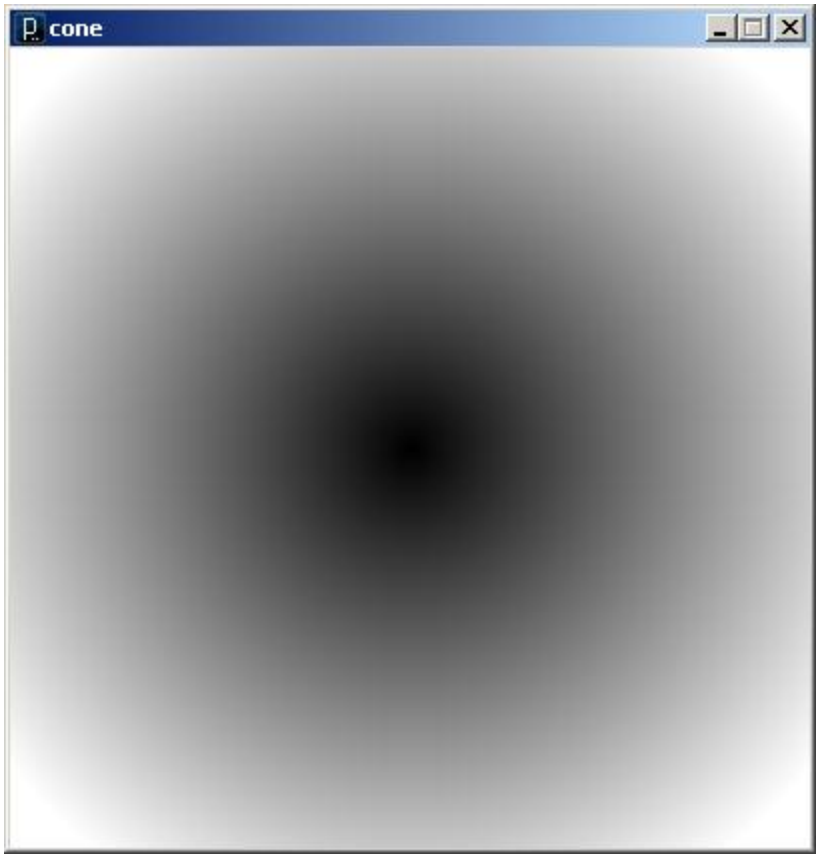
```
Index = (Col-1) + #Columns * (Row-1)
```

Check it …

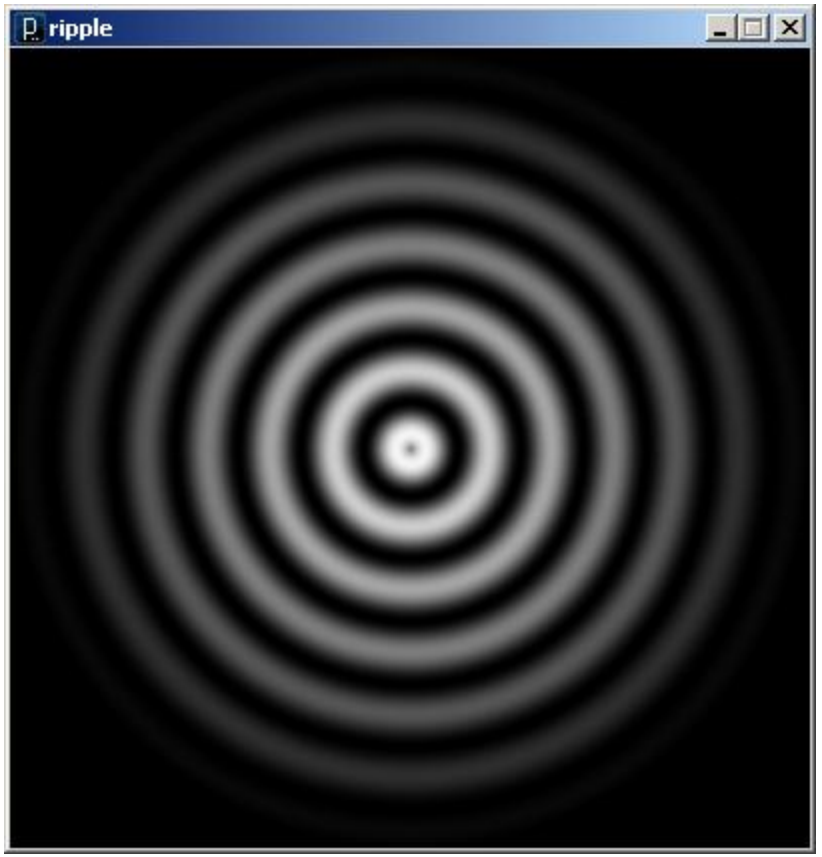
- Using 0-based indexes…

```
int i = c + width*r;
pixels[i] = color(b);
```

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

  float b = 0.0;

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

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

      // Compute distance to center
      b = dist(c, r, 200, 200);

      // Set pixel as distance to center
      int idx = c + width*r;
      pixels[idx] = color(b);
    }
  }

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



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

  float d;
  float b;

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

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

      // Compute distance to center point
      d = dist(c, r, 200, 200);

      // Compute ripple
      b = sin(d/5.0);
      b = ((200.0-d)/200.0)*map(b, -1.0, 1.0, 0, 255);

      // Set pixel as distance to center
      int idx = c + width*r;
      pixels[idx] = color(b);
    }
  }

  // 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

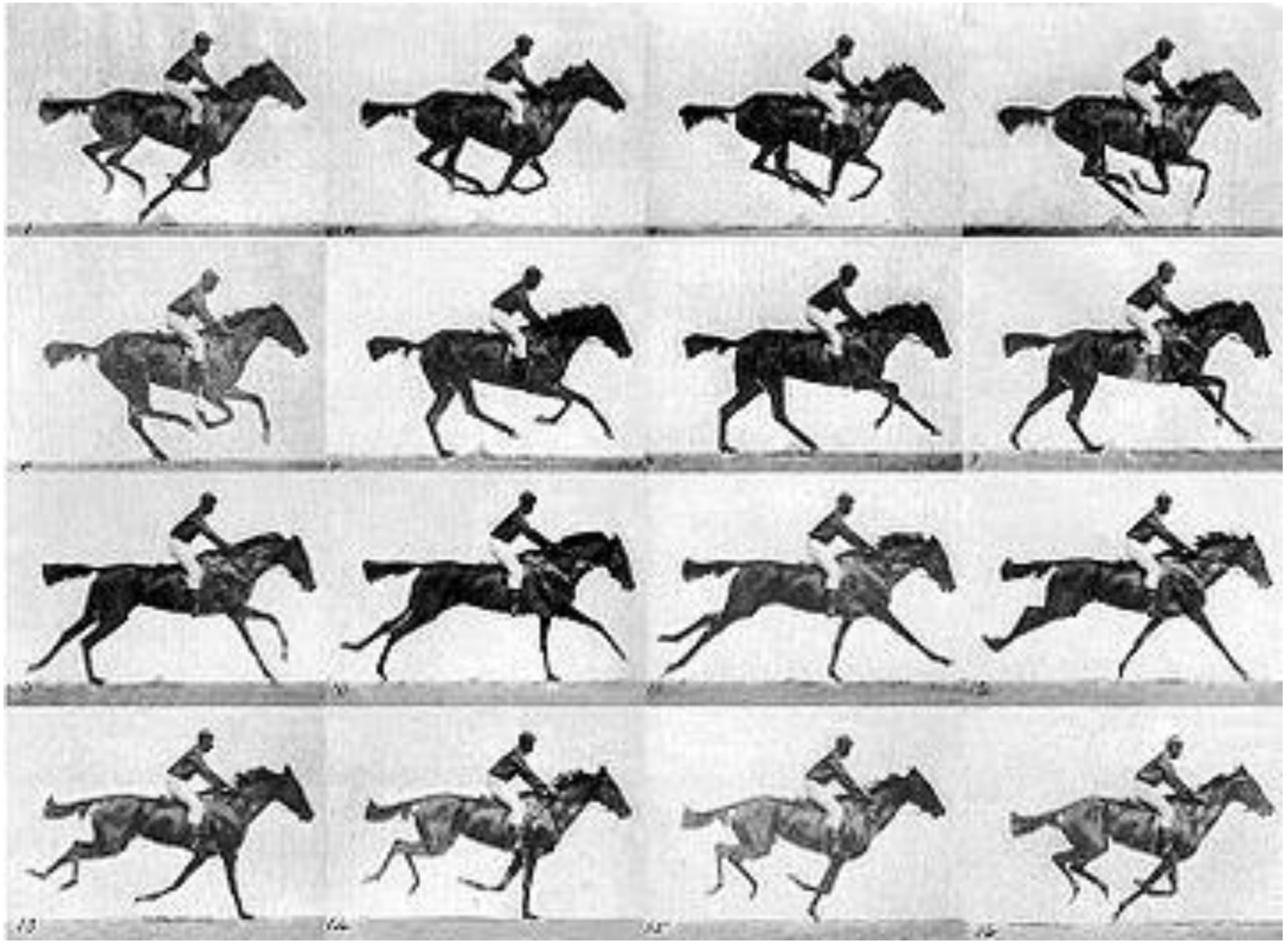
```
PImage img = loadImage("myImage.jpg");
image(img, 0, 0);
```

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

Animating with Images

- Animations can be created using
  - Arrays of PImage objects, and
  - Transformations

Image Sequence



http://commons.wikimedia.org/wiki/File:Muybridge_race_horse_animated_184px.gif

```
// sequence

PImage[] sequence = new PImage[15];
int seqNum = 0;

void setup() {
  size(180, 138);

  // Load images into array
  for (int i=0; i<15; i++) {
    String fileName = "horse" + (i+1) + ".gif";
    sequence[i] = loadImage(fileName);
  }
  // Set frame rate
  frameRate(16);
}

void draw() {
  // Draw a new image on each draw
  image(sequence[seqNum], 0, 0);
  seqNum = (seqNum + 1) % 15;
}
```

```
// sequence2

PImage[] sequence = new PImage[15];
int seqNum = 0;
float xOffset = 0.0;

void setup() {
  size(800, 138);

  // Load images into array
  for (int i=0; i<15; i++) {
    String fileName = "horse" + (i+1) + ".gif";
    sequence[i] = loadImage(fileName);
  }
  // Set frame rate
  frameRate(16);
}

void draw() {
  // translate and draw a new image on each draw
  background(230);
  translate(xOffset, 0);
  image(sequence[seqNum], 0, 0);
  seqNum = (seqNum + 1) % 15;
  xOffset = (xOffset + 20.0) % width;
}
```

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

```
// warhol
void setup() {
  size(750, 327);
  // Load the image three times
  PImage warhol_r = loadImage("andy-warhol2.jpg");
  PImage warhol_g = loadImage("andy-warhol2.jpg");
  PImage warhol_b = loadImage("andy-warhol2.jpg");

  // Load pixels
  warhol_r.loadPixels();
  warhol_g.loadPixels();
  warhol_b.loadPixels();

  // Remove color components
  color c;
  for (int i=0; i<warhol_r.pixels.length; i++) {
    c = warhol_r.pixels[i];
    warhol_r.pixels[i] = color(red(c), 0, 0);

    c = warhol_g.pixels[i];
    warhol_g.pixels[i] = color(0, green(c), 0);

    c = warhol_b.pixels[i];
    warhol_b.pixels[i] = color(0, 0, blue(c));
  }

  // Draw modified images
  image(warhol_r, 0, 0);
  image(warhol_g, 250, 0);
  image(warhol_b, 500, 0);
}
```

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)

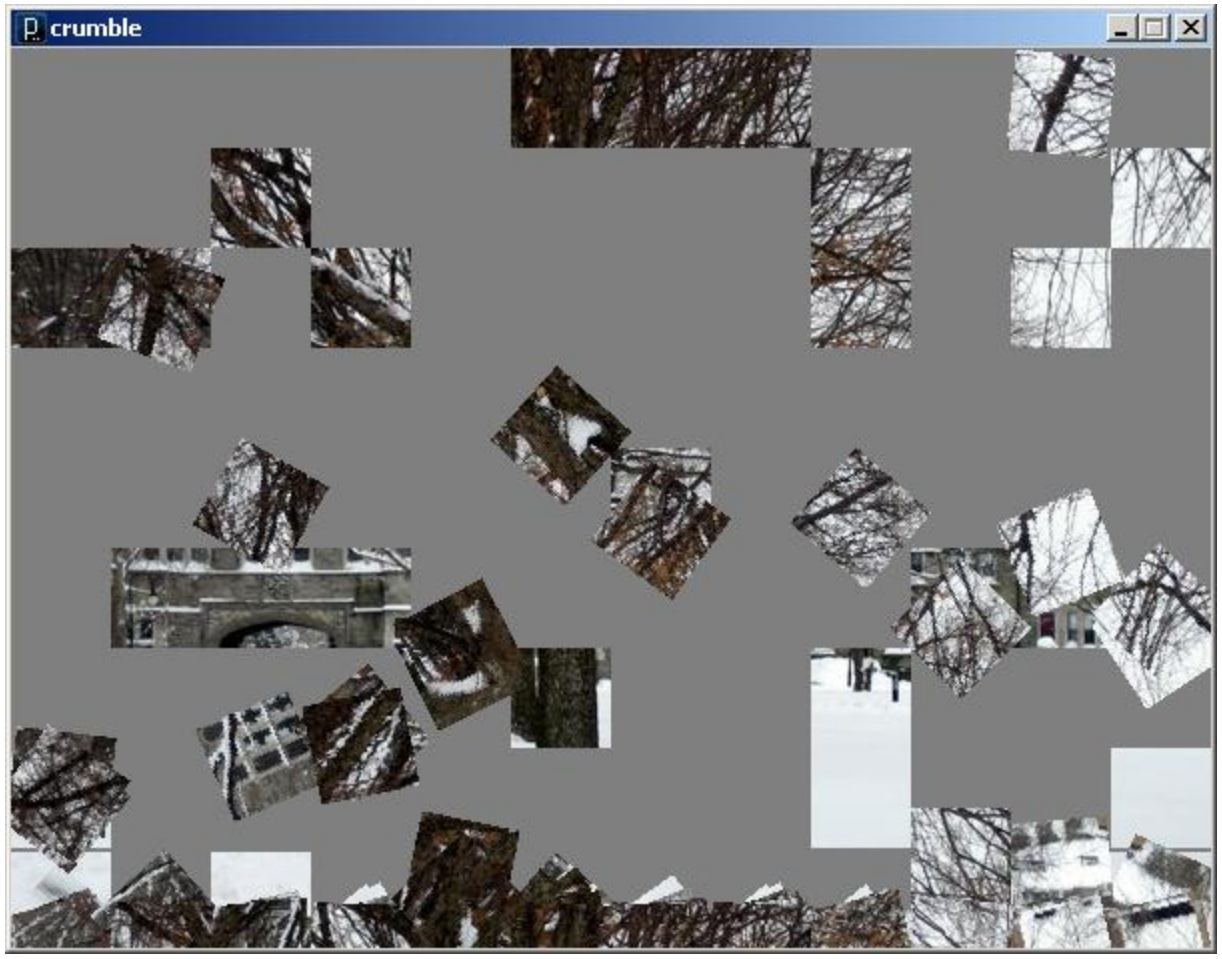
```
Color c = img.get(x, y);
```

- Get a rectangular range of pixels

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

```
// crumble
int nTiles = 12*9;
Tile[] tiles = new Tile[nTiles];

void setup() {
  size(600, 450);    // Load and display image
  imageMode(CENTER);
  PImage img = loadImage("bmc3.jpg");

  int c = 0;         // Init all image tiles
  for (int y=25; y<450; y=y+50) {
    for (int x=25; x<600; x=x+50) {
      tiles[c] = new Tile(
          img.get(x-25, y-25, 50, 50), x, y);
      c++;
    }
  }
}

void draw() {
  background(127);

  // Randomly start tiles falling
  int j = (int)random(nTiles);
  tiles[j].falling = true;
  for (int i=0; i<nTiles; i++)
    tiles[i].update();

  // Draw tiles from the bottom to the top
  for (int i=nTiles-1; i>=0; i--)
    tiles[i].draw();
}
```

```
class Tile {
  // Tile class
  float x;
  float y;
  PImage img;
  float angle = 0;
  boolean falling = false;

  Tile(PImage timg, float tx, float ty) {
    img = timg;
    x = tx;
    y = ty;
  }

  // Move and rotate tile to current location
  // and draw
  void draw() {
    resetMatrix();
    translate(x, y);
    rotate(angle);
    image(img, 0, 0);
  }

  // Update tile location and angle of rotation
  void update() {
    if (!falling) return;
    if (y > height) return;
    angle = (angle + 0.1) % TWO_PI;
    y += 3.0;
  }
}
```


crumble

tint(…) / noTint()

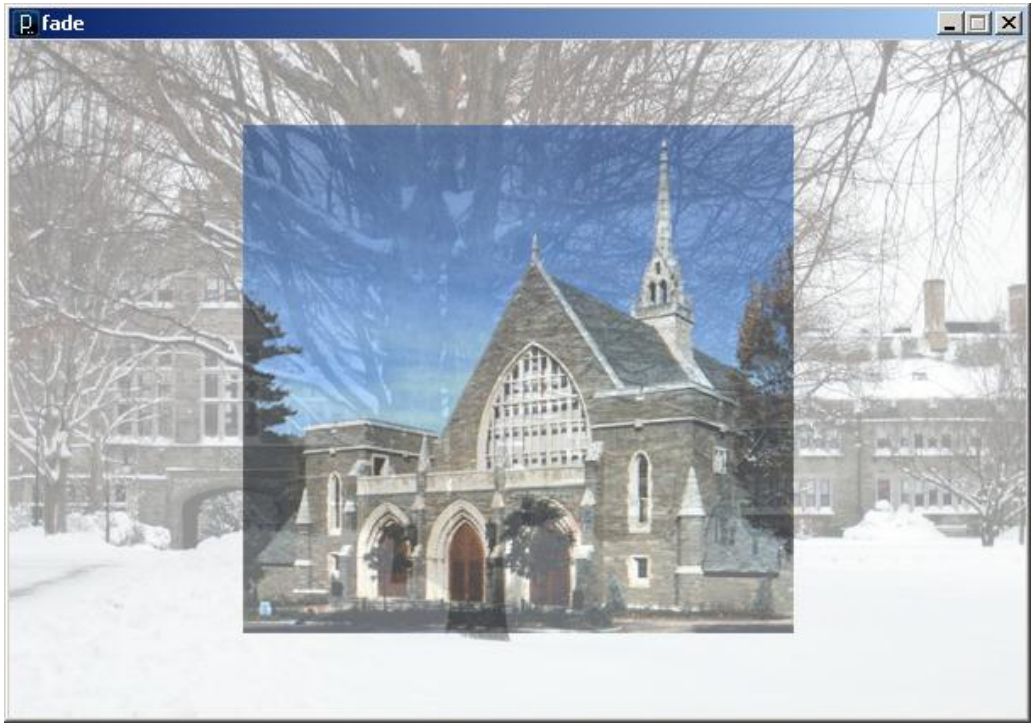
- tint() modifies the fill value for images

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

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

```
// warhol2

void setup() {
  size(750, 327);

  // Load the image three times
  PImage warhol = loadImage("andy-warhol2.jpg");

  // 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);
}
```

```
// fade
PImage[] img = new PImage[5];
int alpha = 255;
int i1 = 0, i2 = 1;

void setup() {
  size(600,400);
  imageMode(CENTER);
  for (int i=0; i<img.length; i++) { // Load images
    img[i] = loadImage("bmc"+i+".jpg");
  }
}

void draw() {
  background(255);

  // Fade out current image
  tint(255, alpha);
  image(img[i1], 300, 200);

  // Fade in next image
  tint(255, 255-alpha);
  image(img[i2], 300, 200);

  // Swap images when fade complete
  alpha--;
  if (alpha < 0) {
    i1 = (i1 + 1) % img.length;
    i2 = (i2 + 1) % img.length;
    alpha = 255;
  }
}
```



```
// pointillism
PImage img;

void setup() {
  size(600, 450);         // Load and display image
  img = loadImage("bmc3.jpg");
  imageMode(CENTER);
  image(img, 300, 225);
  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();
}
```


pointillism

Extending Processing with Libraries

- New objects can be added to Processing by "importing" libraries of prewritten code
- An extensive set of libraries are available
  - Video
  - Networking
  - Hardware Interfaces (Serial, Arduino)
  - Graphics (OpenGL)
  - Sound
  - Animation
  - User Interfaces
  - …

http://processing.org/reference/libraries/

Video Library

- Classes
  - Capture : Grabs images/frames from a camera
  - Movie : Read movie frames and play movies
  - MovieMaker : Create movies from scratch

- Importing …
  - Add the following line to the top of your program

```
import processing.video.*;
```

Creating a Video of Your Animation

- MovieMaker object
  - A class used to create QuickTime movies

```
MovieMaker mm = new MovieMaker(
          this,                          // Parent sketch
          width, height,                 // Sketch size
          "myMovie.mov",                 // Video file
          30,                            // Frames per second
          MovieMaker.ANIMATION,          // Codec
          MovieMaker.HIGH);              // Quality
```

- addFrame() method
  - Adds a snapshot of the sketch to the move as a new frame.

- finish() method
  - Stops recording and closes the video file.

http://processing.org/reference/libraries/video/MovieMaker.html

A generic framework for recording animated sketches as a movie

```
// Capture
// Import Video library
import processing.video.*;

// Declare MovieMaker object
MovieMaker mm;

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

  // Create new MovieMaker object with target movie file
  mm = new MovieMaker(this, width, height, "myMovie.mov", 30,
                      MovieMaker.ANIMATION, MovieMaker.HIGH);

  println("Recording...");

  // Put other setup here
  smooth();
}

void draw() {
  // Do all drawing ...
  // ...
  // Add current sketch display to movie as new frame
  mm.addFrame();
}

// Stop recording when space bar is pressed
void keyPressed() {
  if (key == ' ') {
    mm.finish();
    println("Finished");
  }
}
```

```
// captureCrumble
// Generating a video of the crumble

import processing.video.*;
MovieMaker mm;

int nTiles = 12*9;
Tile[] tiles = new Tile[nTiles];

void setup() {
  size(600, 450);    // Load and display image
  imageMode(CENTER);
  PImage img = loadImage("bmc3.jpg");

  int c = 0;         // Init all image tiles
  for (int y=25; y<450; y=y+50) {
    for (int x=25; x<600; x=x+50) {
      tiles[c] = new Tile(
          img.get(x-25, y-25, 50, 50), x, y);
      c++;
    }
  }

  // Create new MovieMaker object
  mm = new MovieMaker(this, width, height,
             "myMovie.mov", 30,
             MovieMaker.ANIMATION,
             MovieMaker.LOSSLESS);
  println("Recording...");
}

void draw() {
  background(127);

  // Randomly start tiles falling
  int j = (int)random(nTiles);
  tiles[j].falling = true;
  for (int i=0; i<nTiles; i++)
    tiles[i].update();

  // Draw tiles from bottom to top
  for (int i=nTiles-1; i>=0; i--)
    tiles[i].draw();

  // Add current sketch to movie
  mm.addFrame();
}

// Stop recording on space bar
void keyPressed() {
  if (key == ' ') {
    mm.finish();
    println("Finished");
  }
}
```