Review

- Images – an array of colors
- Color – RGBA
- Loading, modifying, updating pixels
- pixels[] as a 2D array
- Animating with arrays of images + transformations
- PImage class, fields and methods
- get() method and crumble
- tint() function – color and alpha filtering
- Creative image processing – Pointillism
- Video Library
- Recording animated sketches as movie files

Thresholding for Image Segmentation

- Pixels below a cutoff value are set to black
- Pixels above a cutoff value are set to white

threshold.pde

Obamicon

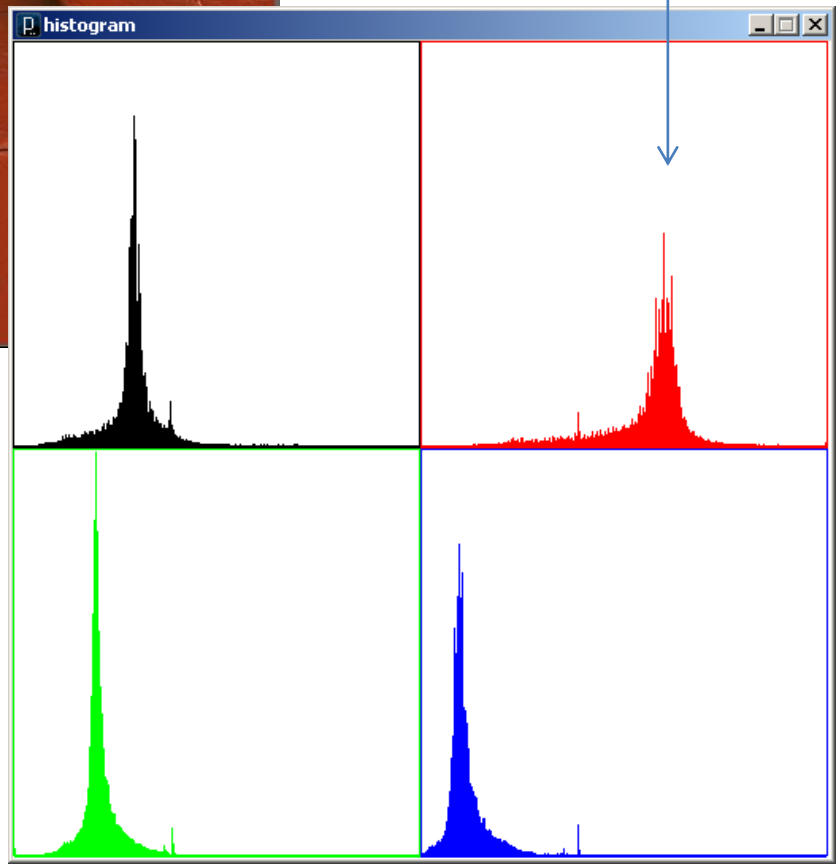
obamicon.pde

```
// obamicon

void setup() {

  // Load image
  PImage img = loadImage("head.jpg");

  // Define colors
  color darkBlue = color(0, 51, 76);
  color reddish = color(217, 26, 33);
  color lightBlue = color(112, 150, 158);
  color yellow = color(252, 227, 166);

  // Size sketch window
  size(img.width, img.height);

  // Draw picture on sketch
  image(img, 0, 0);

  // Posterize image
  loadPixels();

  for (int i = 0; i < pixels.length; i++) {
    // Get pixel color
    color c = pixels[i];

    // Total color components
    float total = red(c)+green(c)+blue(c);

    // Remap to new color
    if (total < 182) {
      pixels[i] = darkBlue;
    }
    else if (total < 364) {
      pixels[i] = reddish;
    }
    else if (total < 546) {
      pixels[i] = lightBlue;
    }
    else {
      pixels[i] = yellow;
    }
  }
  updatePixels();
}
```

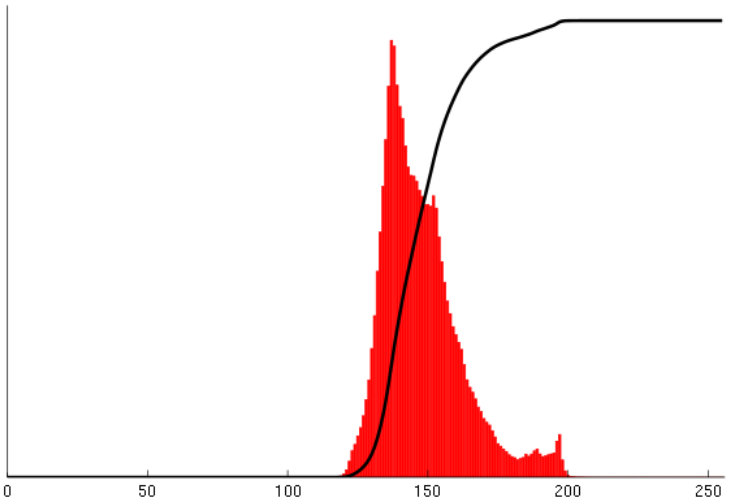
Histogram Equalization

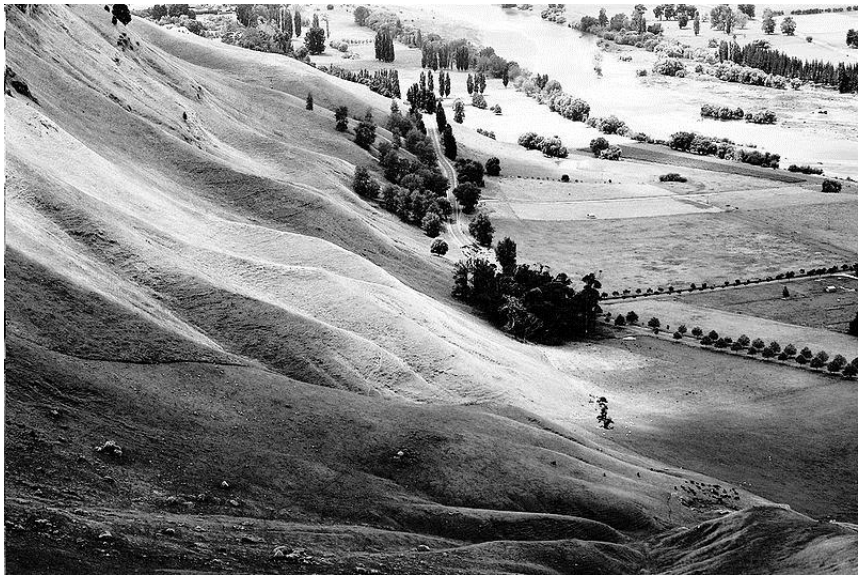
- Increase the global contrast of images
- So that intensities are better distributed
- Reveal more details in photos that are over or under exposed
- Better views of bone structure in X-rays

Shift to the right
implies brighter reds



histogram.pde


0
50
100
150
200
250




0
50
100
150
200
250

Histogram Equalization

- Calculate color frequencies - count the number of times each pixel color appear in the image
- Calculate the cumulative distribution function (cdf) for each pixel color – the number of times all smaller color values appear in the image
- Normalize over (0, 255)

Spatial Filtering (aka Area-Based Filters)

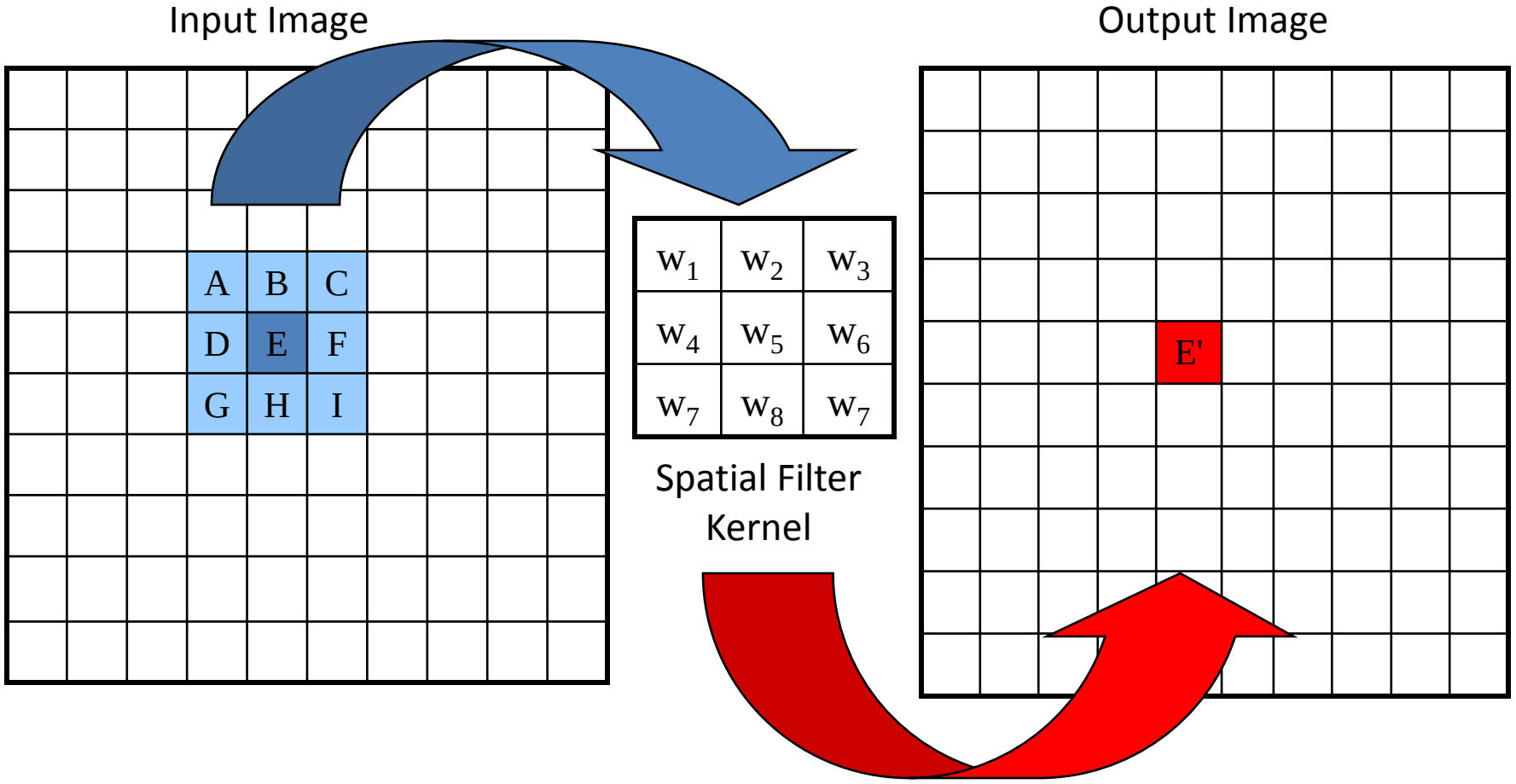
Sharpen

Edge Detection

Gaussian Blur

spatial.pde

Spatial Filtering (aka Area-Based Filters)



$$E' = w_1A+w_2B+w_3C+w_4D+w_5E+w_6F+w_7G+w_8H+w_7I$$

Spatial Kernel Filters - Identity

- No change

0	0	0
0	1	0
0	0	0

Average – smooth

- Set pixel to the average of all colors in the neighborhood
- Smoothes out areas of sharp changes.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Blur – Low Pass Filter

- Softens significant color changes in image
- Creates intermediate colors

1/16	2/16	1/16
2/16	4/16	2/16
1/16	2/16	4/16

Sharpen – High Pass Filter

- Enhances the difference between neighboring pixels
- The greater the difference, the more change in the current pixel

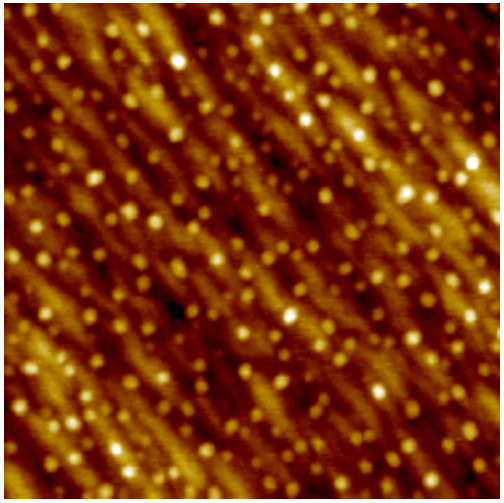
-1	-1	-1
-1	9	-1
-1	-1	-1

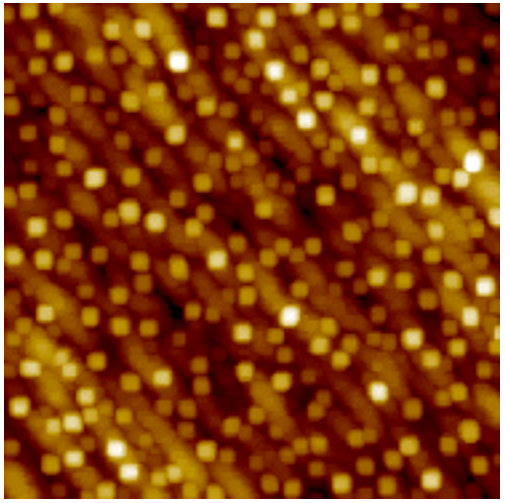
0	-2/3	0
-2/3	11/3	-2/3
0	-2/3	0

```
// Spatial Filtering

PImage img;
PImage filt;
int w = 100;
int msize = 3;

// Sharpen
float[][] matrix = {{ -1., -1., -1.},
                    { -1.,  9., -1.},
                    { -1., -1., -1.}};

// Laplacian Edge Detection
//float[][] matrix = {{  0.,  1.,  0. },
//                    {  1., -4.,  1. },
//                    {  0.,  1.,  0. }};

// Average
//float[][] matrix = {{ 1./9., 1./9., 1./9.},
//                    { 1./9., 1./9., 1./9.},
//                    { 1./9., 1./9., 1./9.}};

// Gaussian Blur
//float[][] matrix = {{  1./16.,  2./16.,  1./16. },
//                    {  2./16.,  4./16.,  2./16. },
//                    {  1./16.,  2./16.,  1./16. }};

void setup() {
  //img = loadImage("bmc3.jpg");
  img = loadImage("moon.jpg");
  size( img.width, img.height );
  filt = createImage(w, w, RGB);
}

void draw() {
  // Draw the image on the background
  image(img,0,0);

  // Get current filter rectangle location
  int xstart =
      constrain(mouseX-w/2,0,img.width);
  int ystart =
      constrain(mouseY-w/2,0,img.height);

  // Filter rectangle
  loadPixels();
  filt.loadPixels();

  for (int i=0; i<w; i++ ) {
    for (int j=0; j<w; j++) {
      int x = xstart + i;
      int y = ystart + j;
      color c =
          spatialFilter(x, y, matrix, msize, img);
      int loc = i+j*w;
      filt.pixels[loc] = c;
    }
  }

  filt.updatePixels();
  updatePixels();

  // Add rectangle around convolved region
  stroke(0);
  noFill();
  image(filt, xstart, ystart);
  rect(xstart, ystart, w, w);
}

// Perform spatial filtering on one pixel location
color spatialFilter(int x, int y, float[][] matrix,
                    int msize, PImage img) {
  float rtotal = 0.0;
  float gtotal = 0.0;
  float btotal = 0.0;
  int offset = msize/2;

  // Loop through filter matrix
  for (int i=0; i<msize; i++) {
    for (int j=0; j<msize; 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
      loc = constrain(loc,0,img.pixels.length-1);

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

Dilation - Morphology

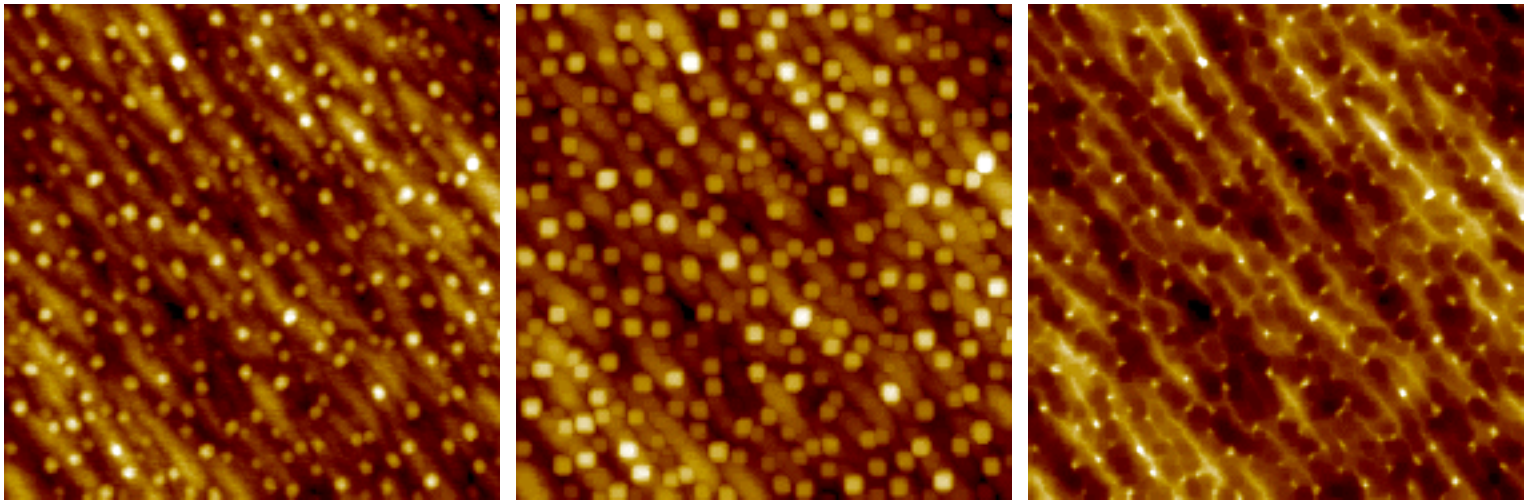
- Set pixel to the maximum color value within a 3x3 window around the pixel
- Causes objects to grow in size.
- Brightens and fills in small holes



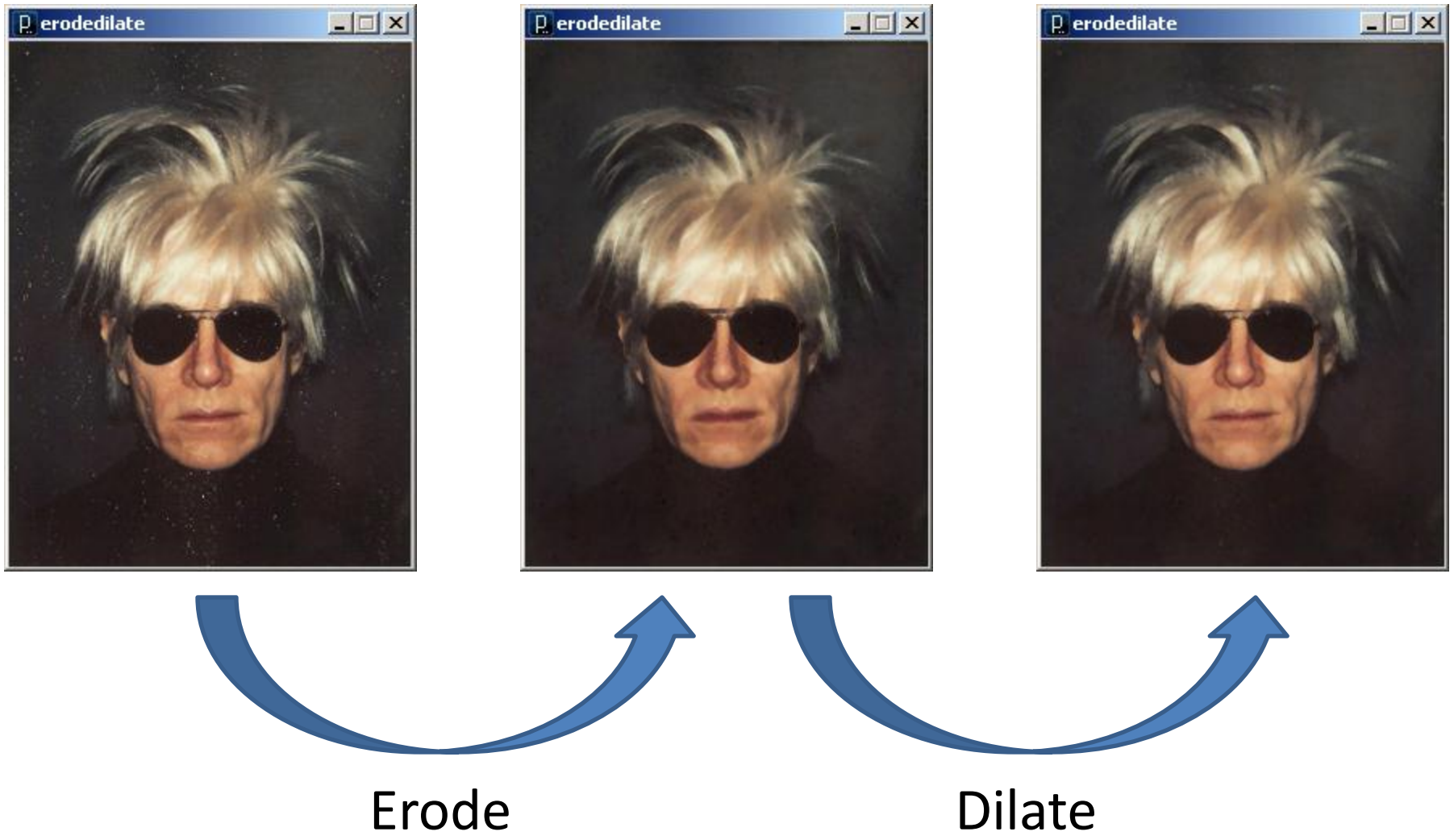


Erosion - Morphology

- Set pixel to the minimum color value within a 3x3 window around the pixel
- Causes objects to shrink.
- Darkens and removes small objects



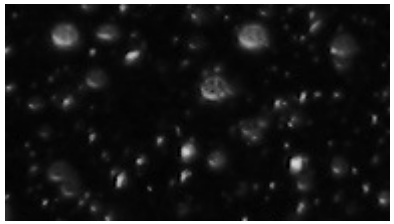
Erode + Dilate to Despeckle

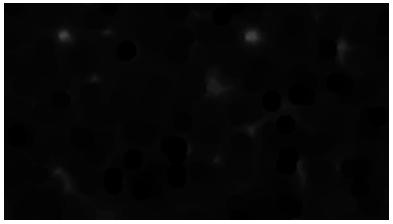


erodedilate.pde

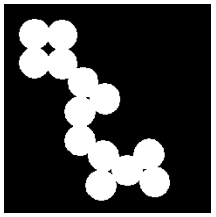
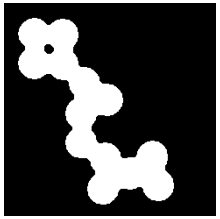
Feature Extraction

- Region detection – morphology manipulation
  - Dilate and Erode





  - Open
    - Erode → Dilate
    - Small objects are removed
  - Close
    - Dilate → Erode
    - Holes are closed

  - Skeleton and perimeter

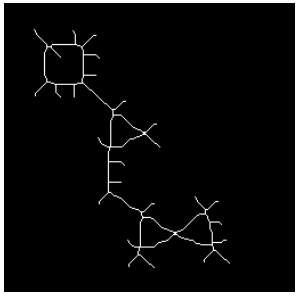
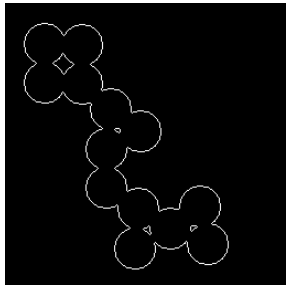
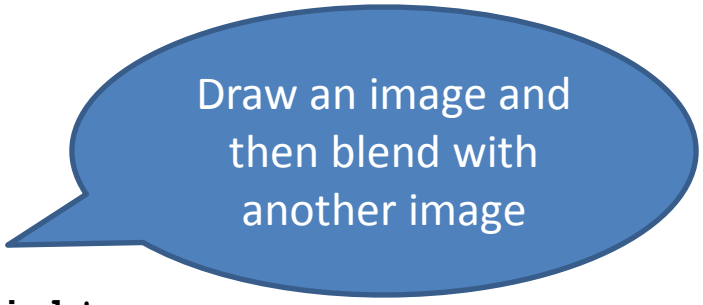




Image Processing in Processing

tint()	modulate individual color components
blend()	combine the pixels of two images in a given manner
filter()	apply an image processing algorithm to an image

blend()

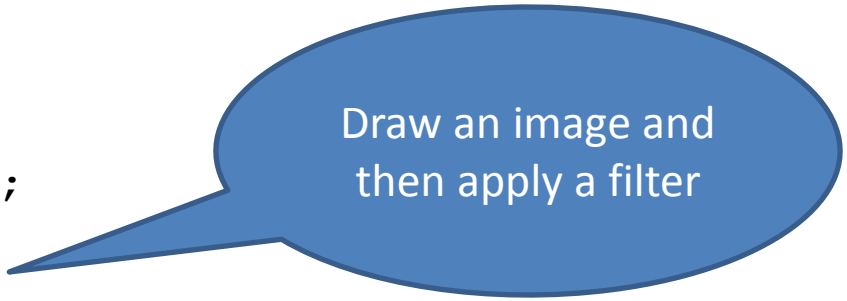
```
img = loadImage("colony.jpg");
mask = loadImage("mask.png");
image(img, 0, 0);
blend(mask, 0, 0, mask.width, mask.height,
              0, 0, img.width, img.height, SUBTRACT);
```



BLEND	linear interpolation of colours:	C = A*factor + B
ADD	additive blending with white clip:	C = min(A*factor + B, 255)
SUBTRACT	subtractive blending with black clip:	C = max(B - A*factor, 0)
DARKEST	only the darkest colour succeeds:	C = min(A*factor, B)
LIGHTEST	only the lightest colour succeeds:	C = max(A*factor, B)
DIFFERENCE	subtract colors from underlying image.	
EXCLUSION	similar to DIFFERENCE, but less extreme.	
MULTIPLY	Multiply the colors, result will always be darker.	
SCREEN	Opposite multiply, uses inverse values of the colors.	
OVERLAY	A mix of MULTIPLY and SCREEN. Multiplies dark values, and screens light values.	
HARD_LIGHT	SCREEN when greater than 50% gray, MULTIPLY when lower.	
SOFT_LIGHT	Mix of DARKEST and LIGHTEST. Works like OVERLAY, but not as harsh.	
DODGE	Lightens light tones and increases contrast, ignores darks.	
BURN	Darker areas are applied, increasing contrast, ignores lights.	

filter()

```
PImage b;
b = loadImage("myImage.jpg");
image(b, 0, 0);
filter(THRESHOLD, 0.5);
```



THRESHOLD	converts the image to black and white pixels depending if they are above or below the threshold defined by the level parameter. The level must be between 0.0 (black) and 1.0 (white). If no level is specified, 0.5 is used.
GRAY	converts any colors in the image to grayscale equivalents
INVERT	sets each pixel to its inverse value
POSTERIZE	limits each channel of the image to the number of colors specified as the level parameter
BLUR	executes a Gaussian blur with the level parameter specifying the extent of the blurring. If no level parameter is used, the blur is equivalent to Gaussian blur of radius 1.
OPAQUE	sets the alpha channel to entirely opaque.
ERODE	reduces the light areas with the amount defined by the level parameter.
DILATE	increases the light areas with the amount defined by the level parameter.

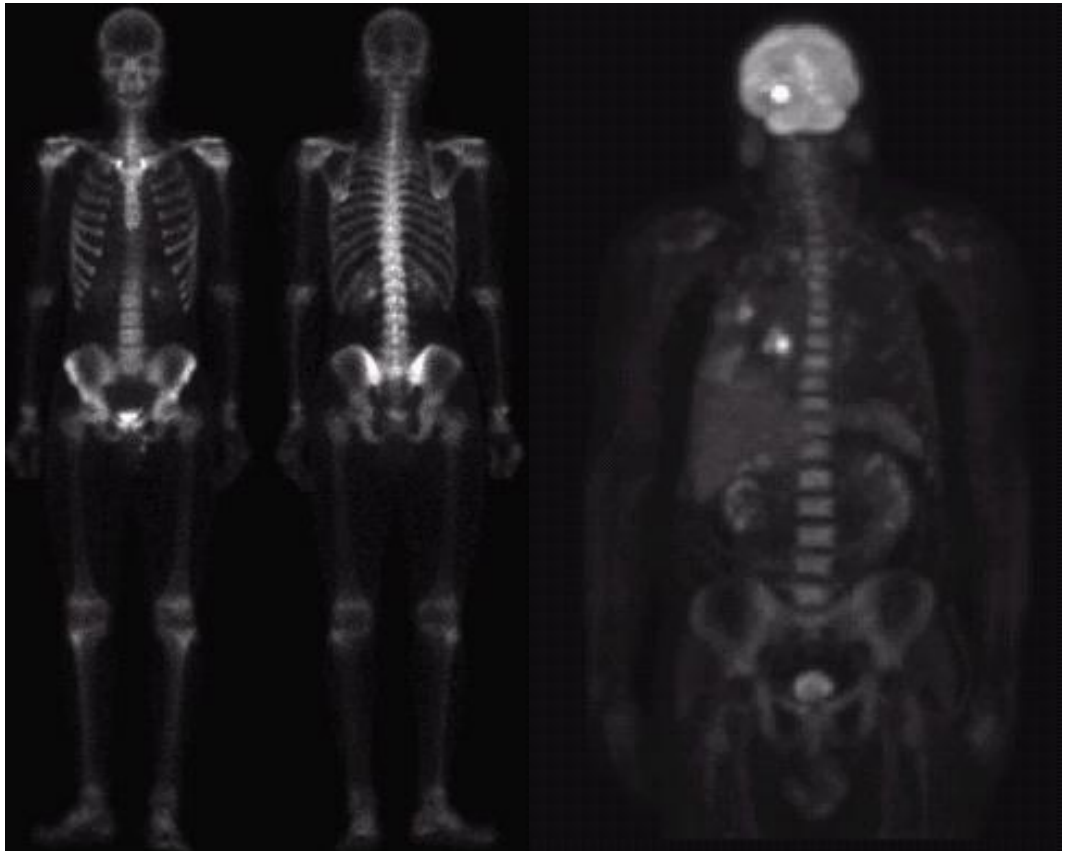
```
// Threshold
PImage img;

void setup() {
  img = loadImage("kodim01.png");
  size(img.width, img.height);
  image(img, 0, 0);
}

void draw() {}

void drawImg(float thresh) {
  image(img, 0, 0);
  filter(THRESHOLD, thresh);
}

void mouseDragged() {
  float thresh = map(mouseY, 0, height, 0.0, 1.0);
  println(thresh);
  drawImg(thresh);
}
```

threshold.pde

```
// Posterize
PImage img;

void setup() {
  img = loadImage("andy-warhol2.jpg");
  size(img.width, img.height);
  image(img, 0, 0);
}

void draw() {}

void drawImg(float val  {
  image(img, 0, 0);
  filter(POSTERIZE, val);
}

void mouseDragged() {
  float val = int(map(mouseY, 0, height, 2, 10));
  val = constrain(val, 2, 10);
  println(val);
  drawImg(val);
}
```

posterize.pde

Medical Images



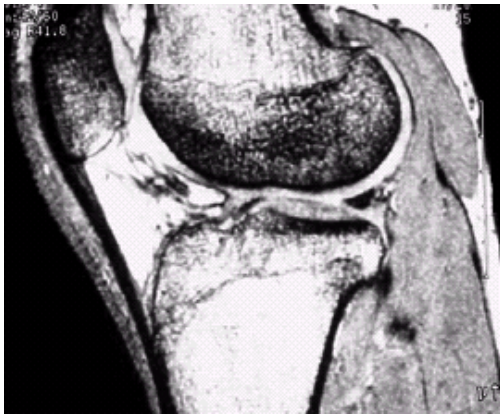
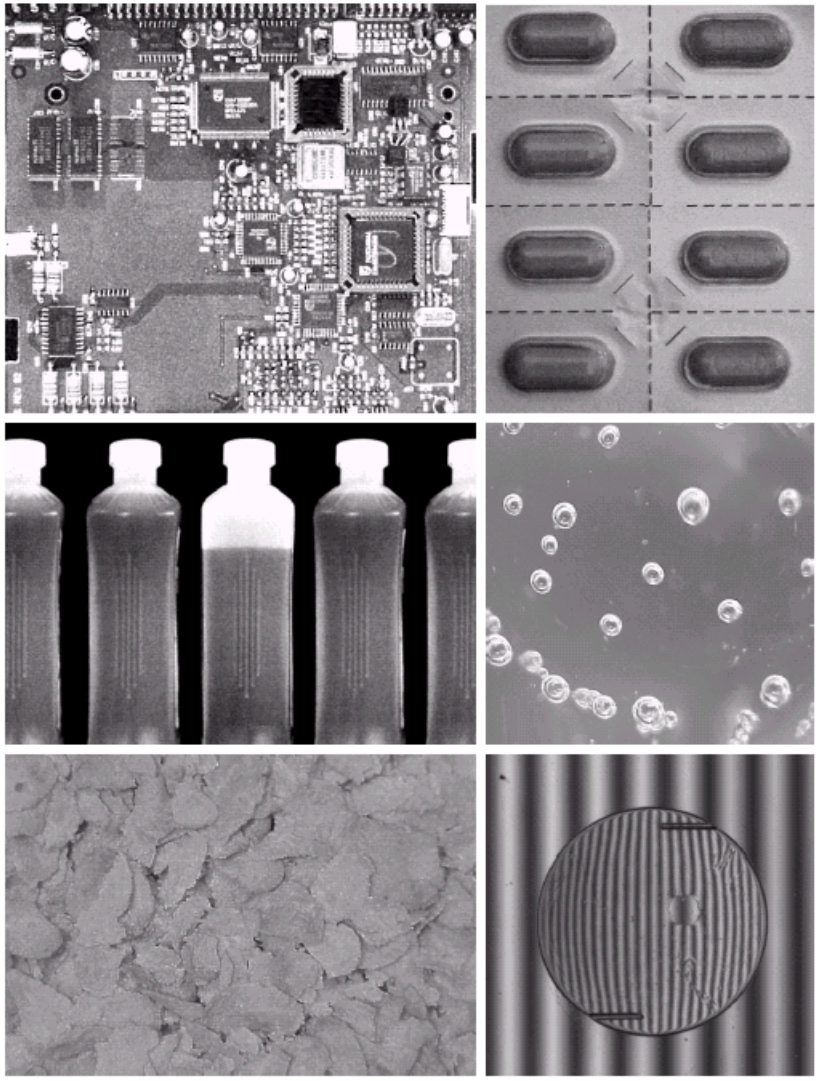


Image Processing in Manufacturing



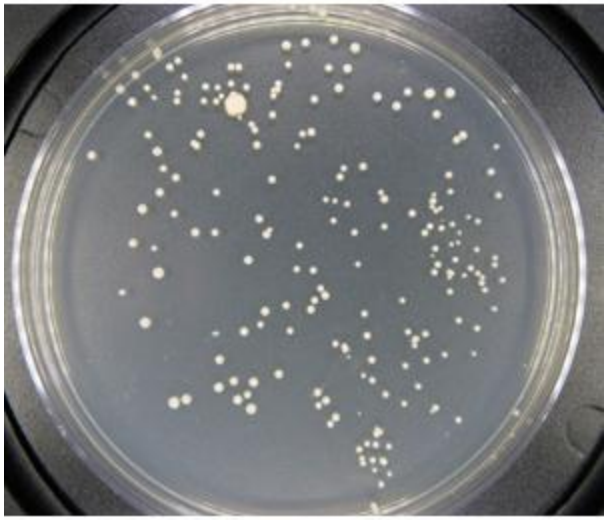
a b
c d
e f

FIGURE 1.14
Some examples of manufactured goods often checked using digital image processing. (a) A circuit board controller. (b) Packaged pills. (c) Bottles. (d) Bubbles in clear-plastic product. (e) Cereal. (f) Image of intraocular implant. (Fig. (f) courtesy of Mr. Pete Sites, Perceptics Corporation.)

Measuring Confluency in Cell Culture Biology

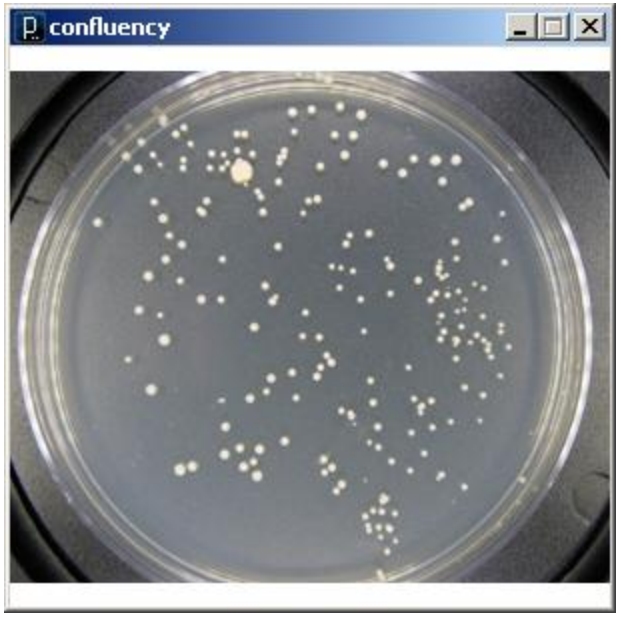
- Refers to the coverage of a dish or flask by the cells
- 100% confluency = completely covered

- Image Processing Method
  1. Mask off unimportant parts of image
  2. Threshold image
  3. Count pixels of certain color

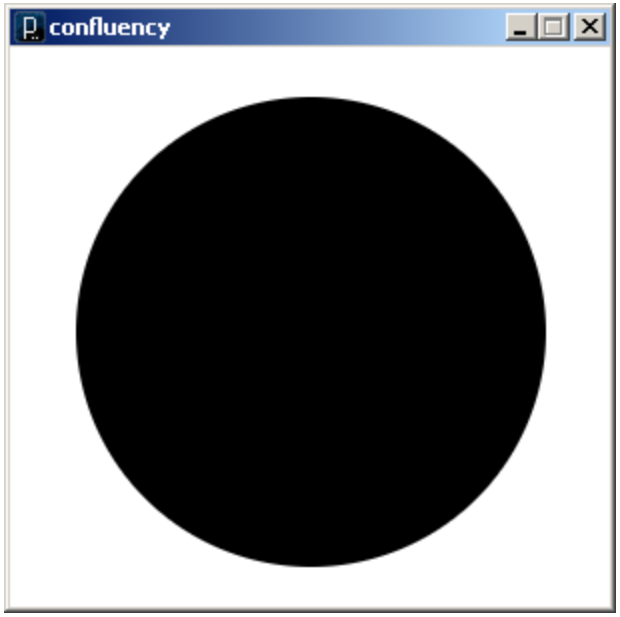


Blend: Subtract

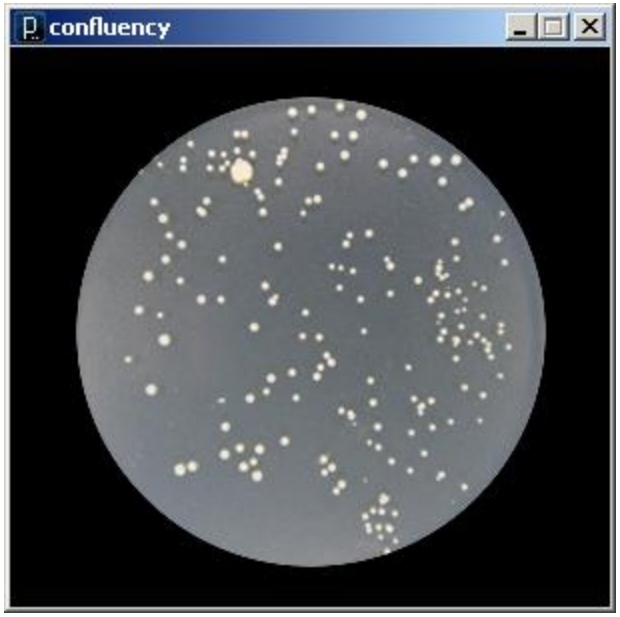
Original



Mask

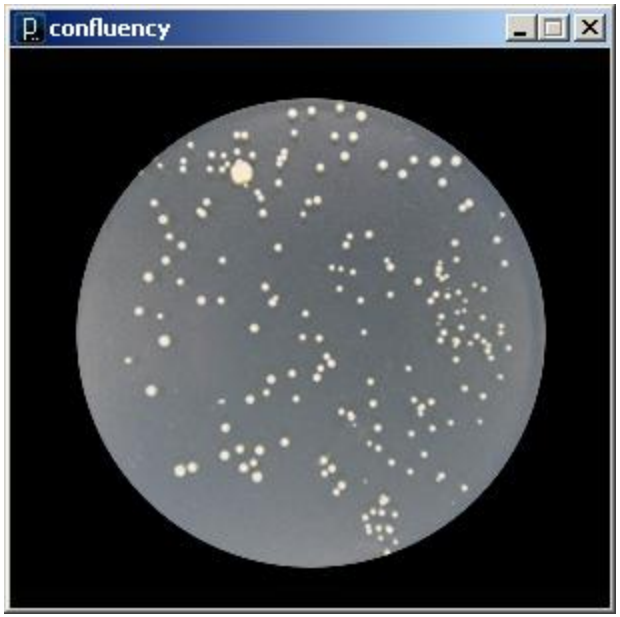


Subtracted

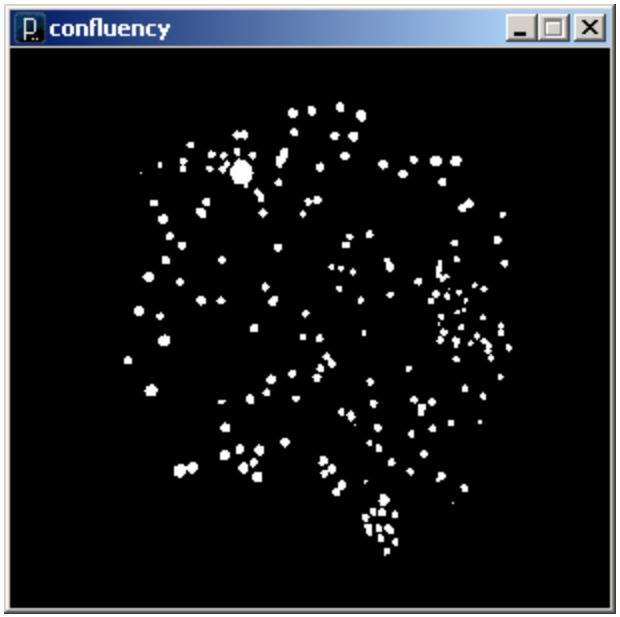


Filter: Theshold

Subtracted

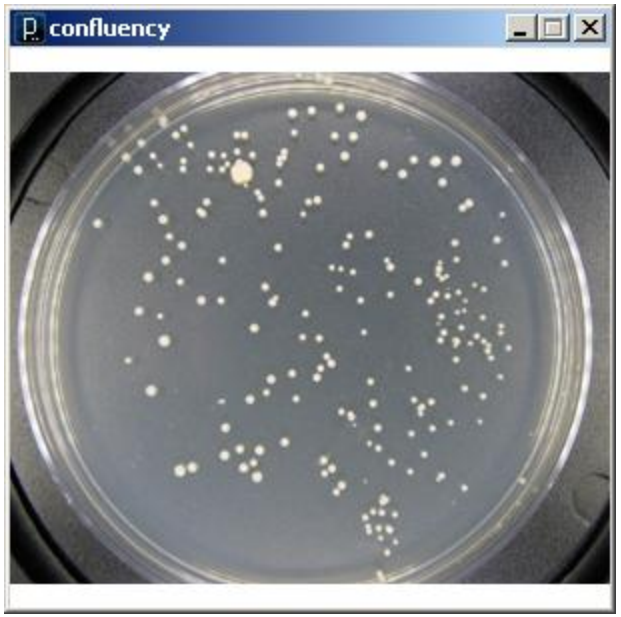


Threshold



Count Fraction of Pixels to Quantify

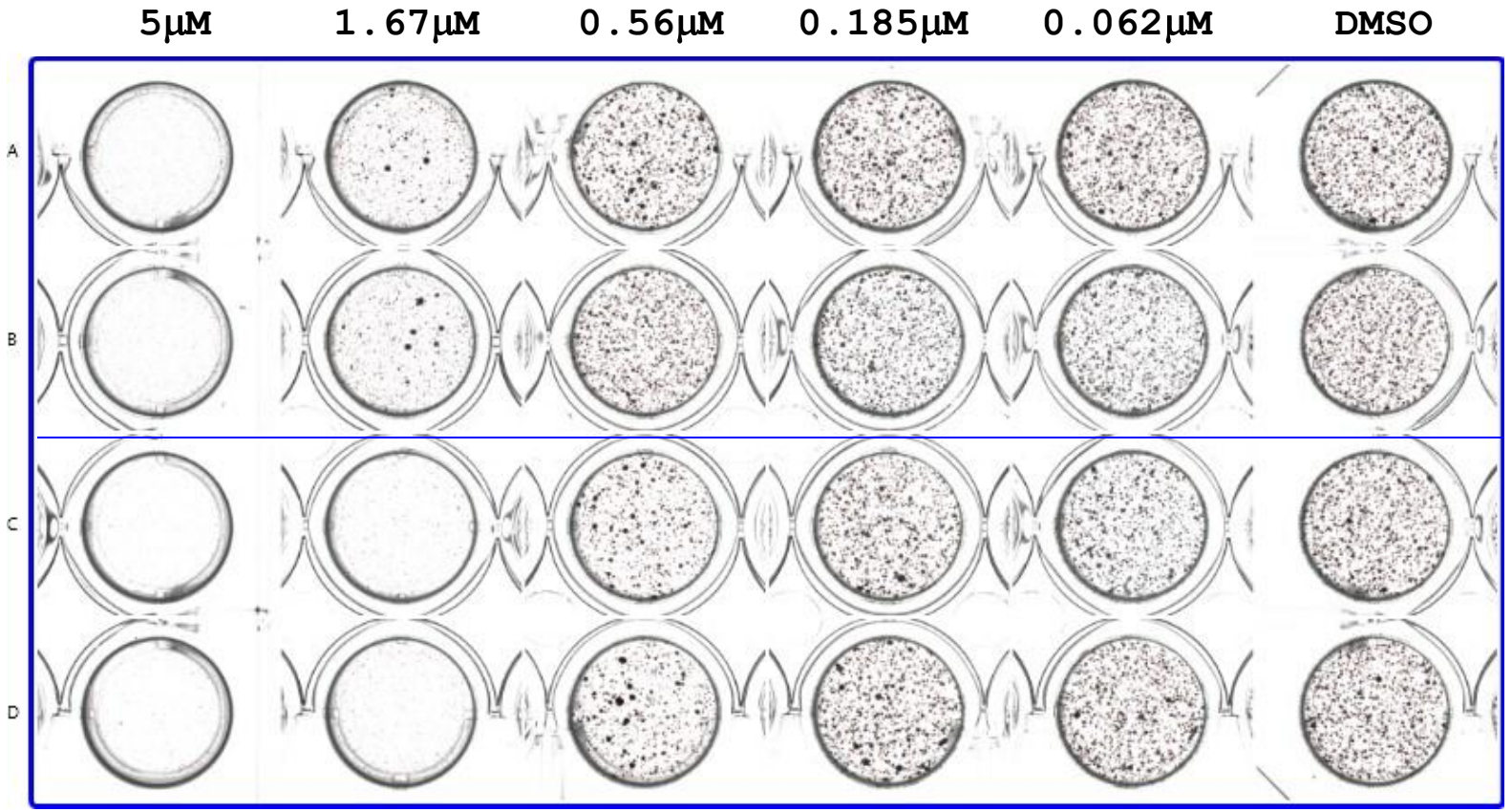
```
// Colony Confluency
PImage img;
PImage mask;

void setup() {
  img = loadImage("colony.jpg");
  mask = loadImage("mask.png");
  size(img.width, img.height);
}

void draw() {
  image(img, 0, 0);
  blend(mask, 0, 0, mask.width, mask.height,
              0, 0, img.width, img.height, SUBTRACT);
  filter(THRESHOLD, 0.6);
}

void mousePressed() {
  loadPixels();
  int count = 0;
  for (int i=0; i<pixels.length; i++)
    if (red(pixels[i]) == 255) count++;
  println(count/42969.0);
}
```



5.3 % Confluency

confluency.pde

IC_{50} determination



Vision Guided Robotics
Colony Picking

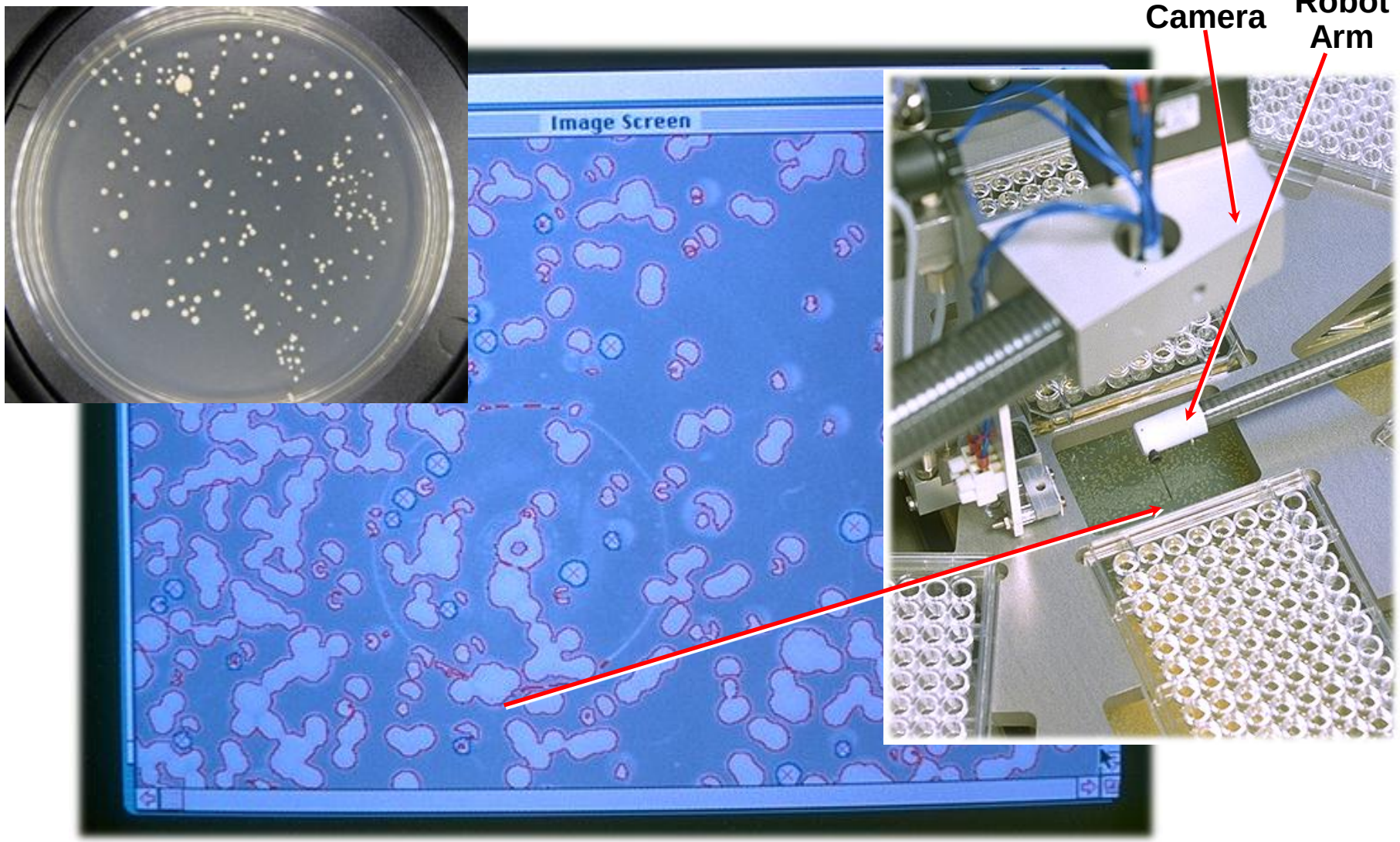
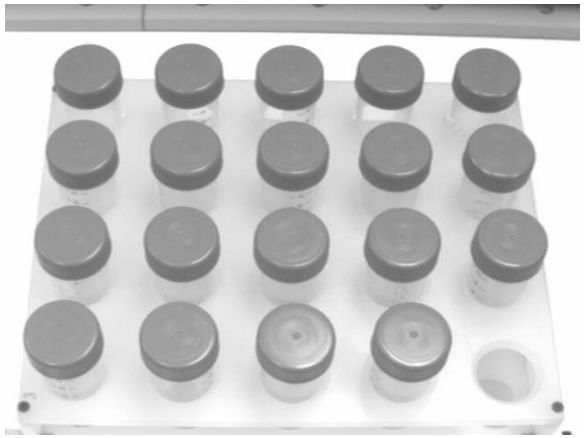
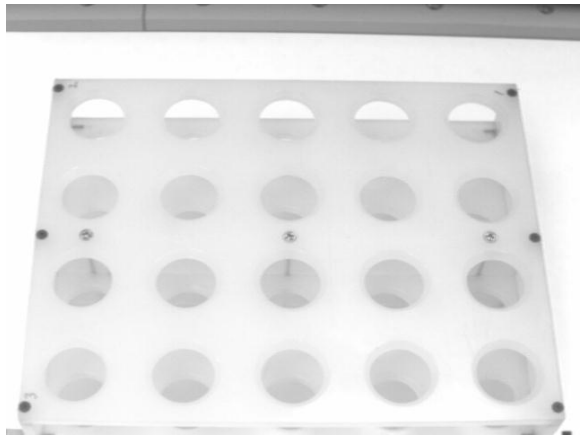


Image Processing





Compute the presence of objects or “particles”

Image Processing

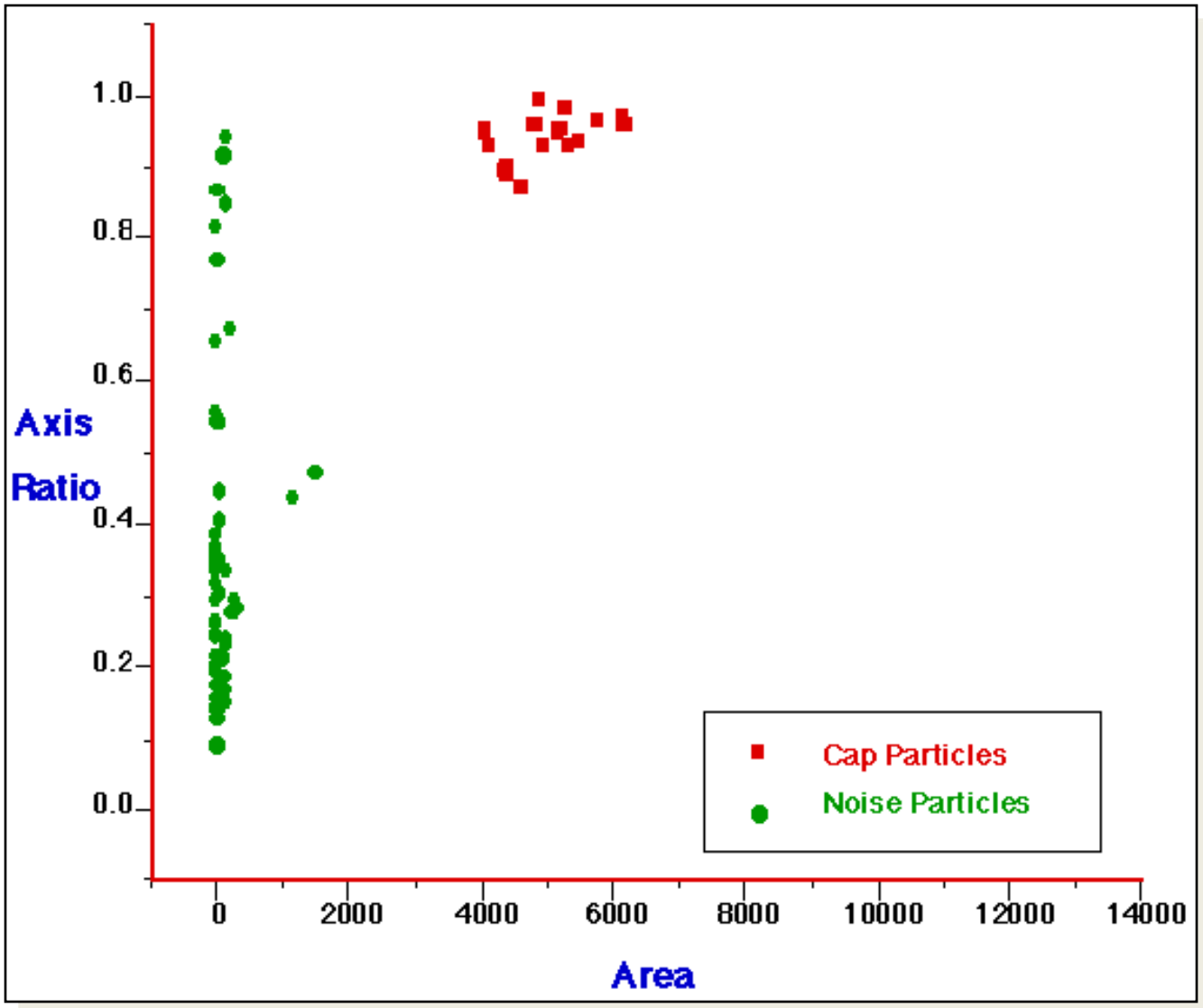


Image Processing

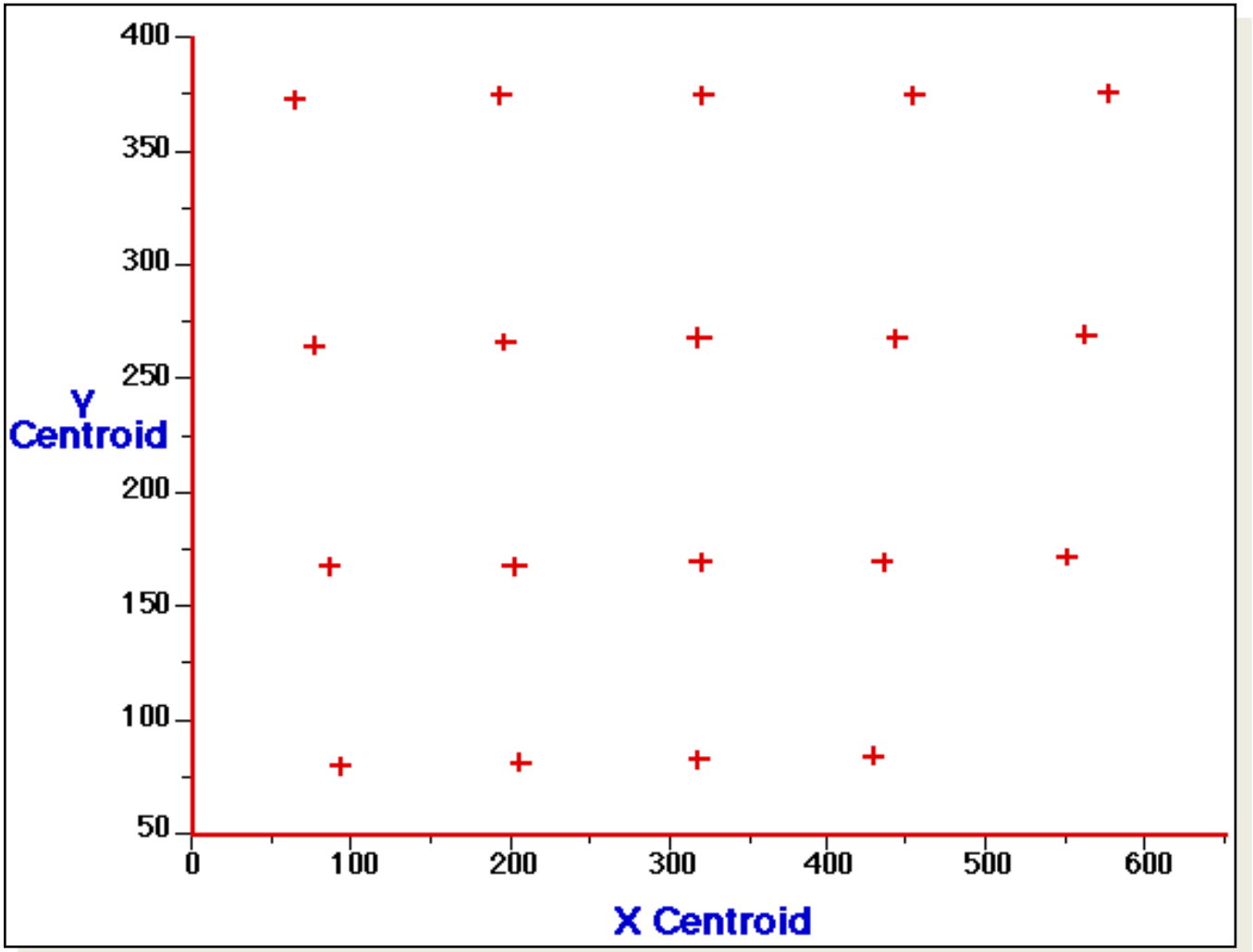
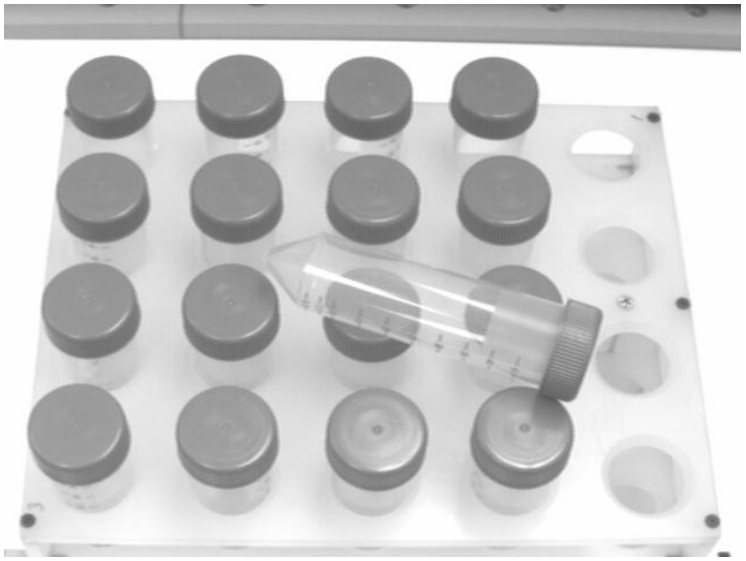


Image Processing



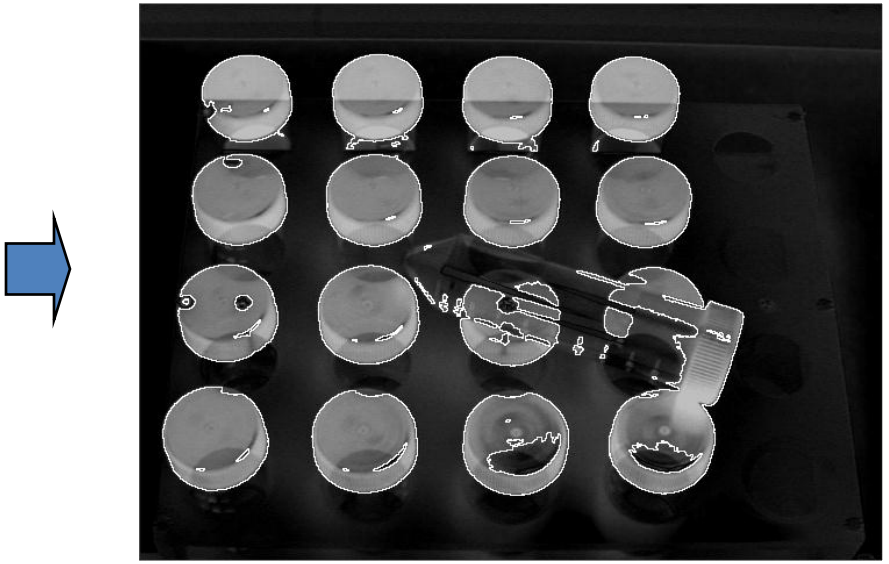
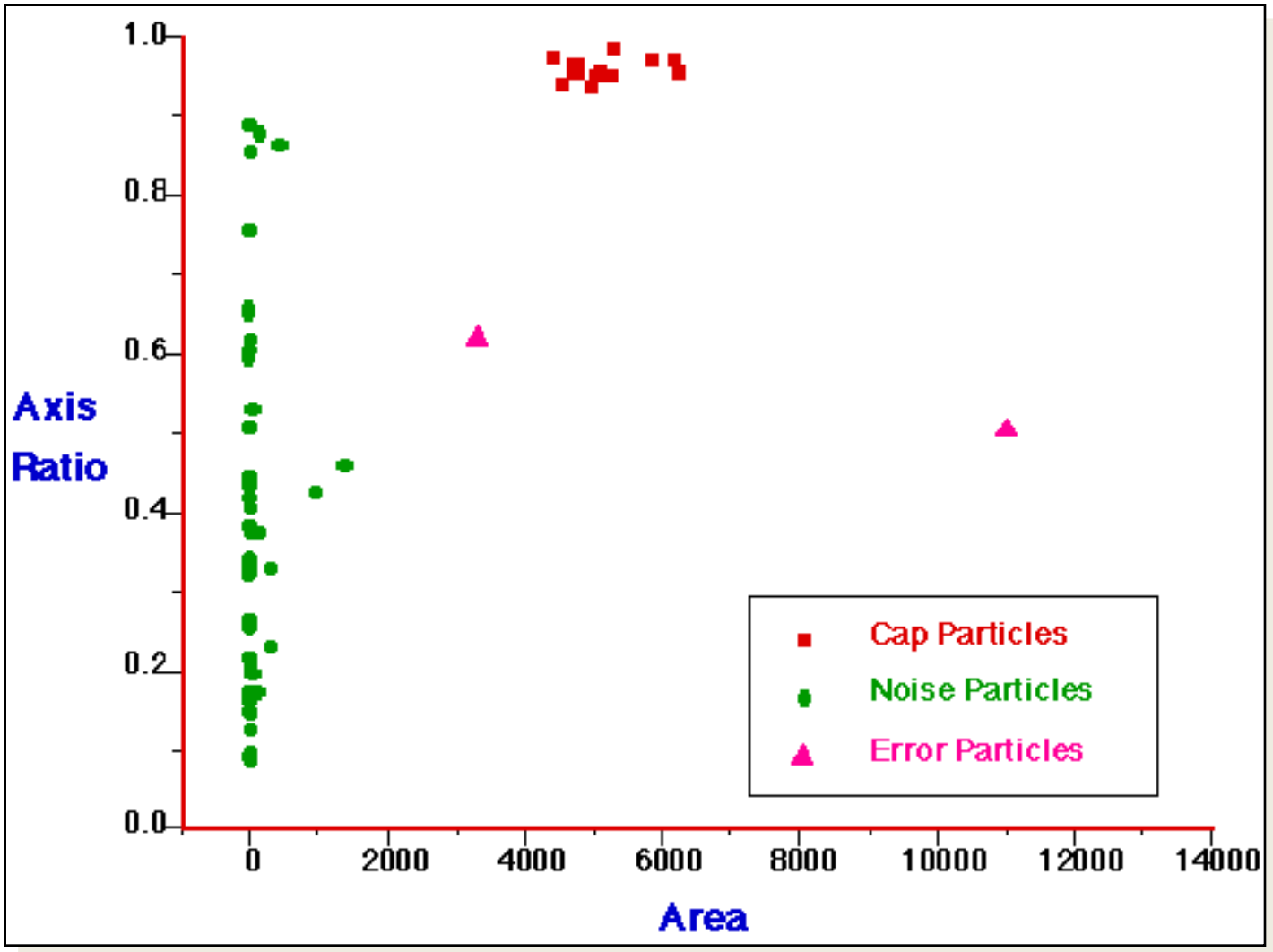


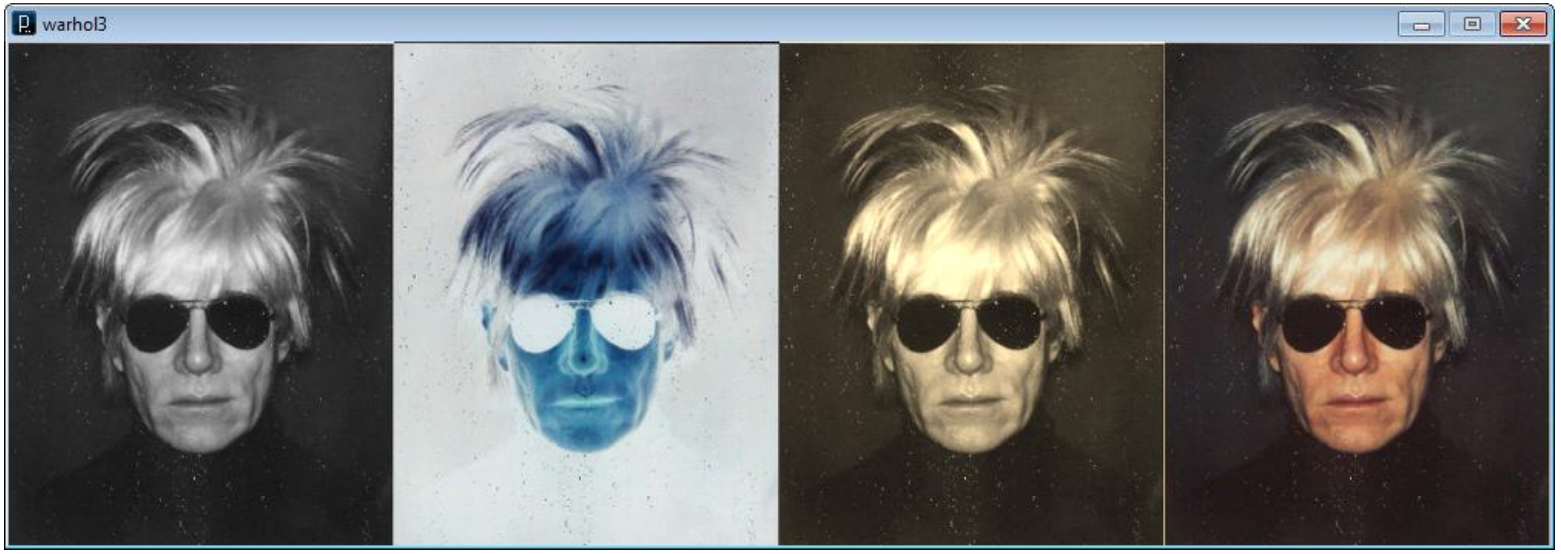
Image Processing



Implementing Basic Image Filtering

red green blue

grayscale negative sepia



warhol1.pde, warhol3.pde

Black and White, Negative and Sepia Filters

```
void setup() {
  size(1000, 327);

  // Load the image four times
  PImage warhol_bw  = loadImage("andy-warhol2.jpg");
  PImage warhol_neg = loadImage("andy-warhol2.jpg");
  PImage warhol_sep = loadImage("andy-warhol2.jpg");
  PImage warhol_a   = loadImage("andy-warhol2.jpg");

  // Load pixels into pixels array
  warhol_bw.loadPixels();
  warhol_neg.loadPixels();
  warhol_sep.loadPixels();
  warhol_a.loadPixels();

  // …
```

warhol3.pde

Black and White, Negative and Sepia Filters

```
// Continued …

// Remove color components
color c;
for (int i=0; i<warhol_bw.pixels.length; i++) {

  // Black and white filter
  c = warhol_bw.pixels[i];
  warhol_bw.pixels[i] = color(0.3*red(c)+ 0.59*green(c)+ 0.11*blue(c));

  // Negative filter
  c = warhol_neg.pixels[i];
  warhol_neg.pixels[i] = color(255-red(c), 255-green(c), 255-blue(c));

  // Sepia filter
  c = warhol_sep.pixels[i];
  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;
  warhol_sep.pixels[i] = color(r, g, b);
}
```

warhol3.pde

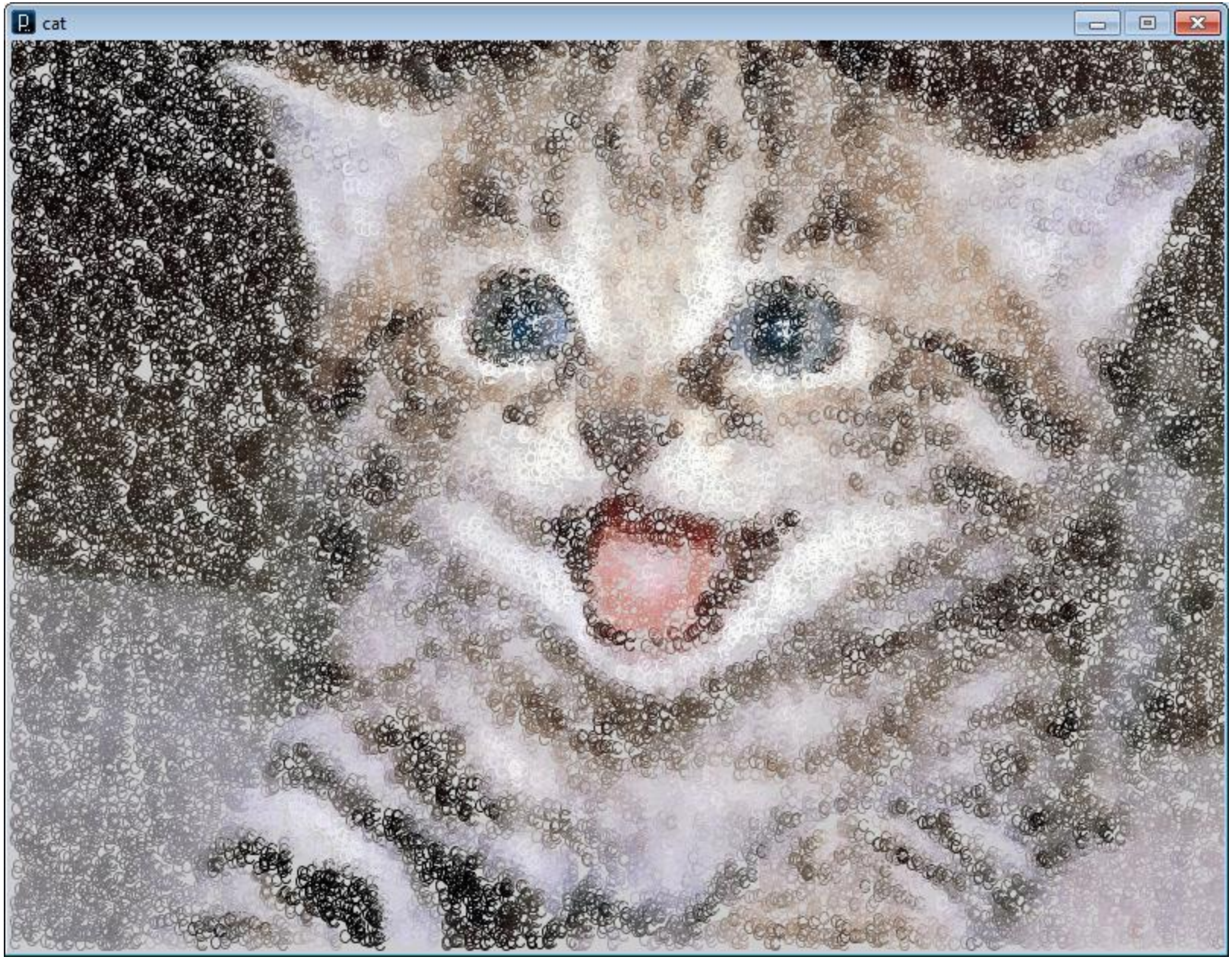
Black and White, Negative and Sepia Filters

```
  // Continued …

  // Draw modified images
  image(warhol_bw, 0, 0);
  image(warhol_neg, 250, 0);
  image(warhol_sep, 500, 0);
  image(warhol_a, 750, 0);
}
```

warhol3.pde

Cat made of various glyphs

```
// cat
PImage img;

void setup() {
  size(800, 600);
  img = loadImage("cat.jpg");    // Load image
  noStroke();
  ellipseMode(CENTER);
  img.loadPixels();             // Cover with random shapes
  for (int i=0; i<30000; i++) {
    addGlyph();
  }
}
void addGlyph() {
 // Add a random colored glyps to recreate the image
  int x = (int)random(width);
  int y = (int)random(height);
  int i = x + width*y;
  color c = img.pixels[i];
  fill(c);
  text("C", x, y);
  //ellipse(x, y, 7, 7);
}
```


cat

What can you do with Image Processing?

Inspect, Measure, and Count using Photos and Video
http://www.youtube.com/watch?v=KsTtNWVhpgI

Image Processing Software
http://www.youtube.com/watch?v=1WJp9mGnWSM

Manual Colony Counter
http://www.youtube.com/watch?v=7B-9Wf6pENQ

Automated Colony counter
http://www.youtube.com/watch?v=qtJmQqRHHag

Predator algorithm for object tracking with learning
http://www.youtube.com/watch?v=1GhNXHCQGsM

Video Processing, with Processing
http://www.niklasroy.com/project/88/my-little-piece-of-privacy/
http://www.youtube.com/watch?v=rKhbUjVyKIc