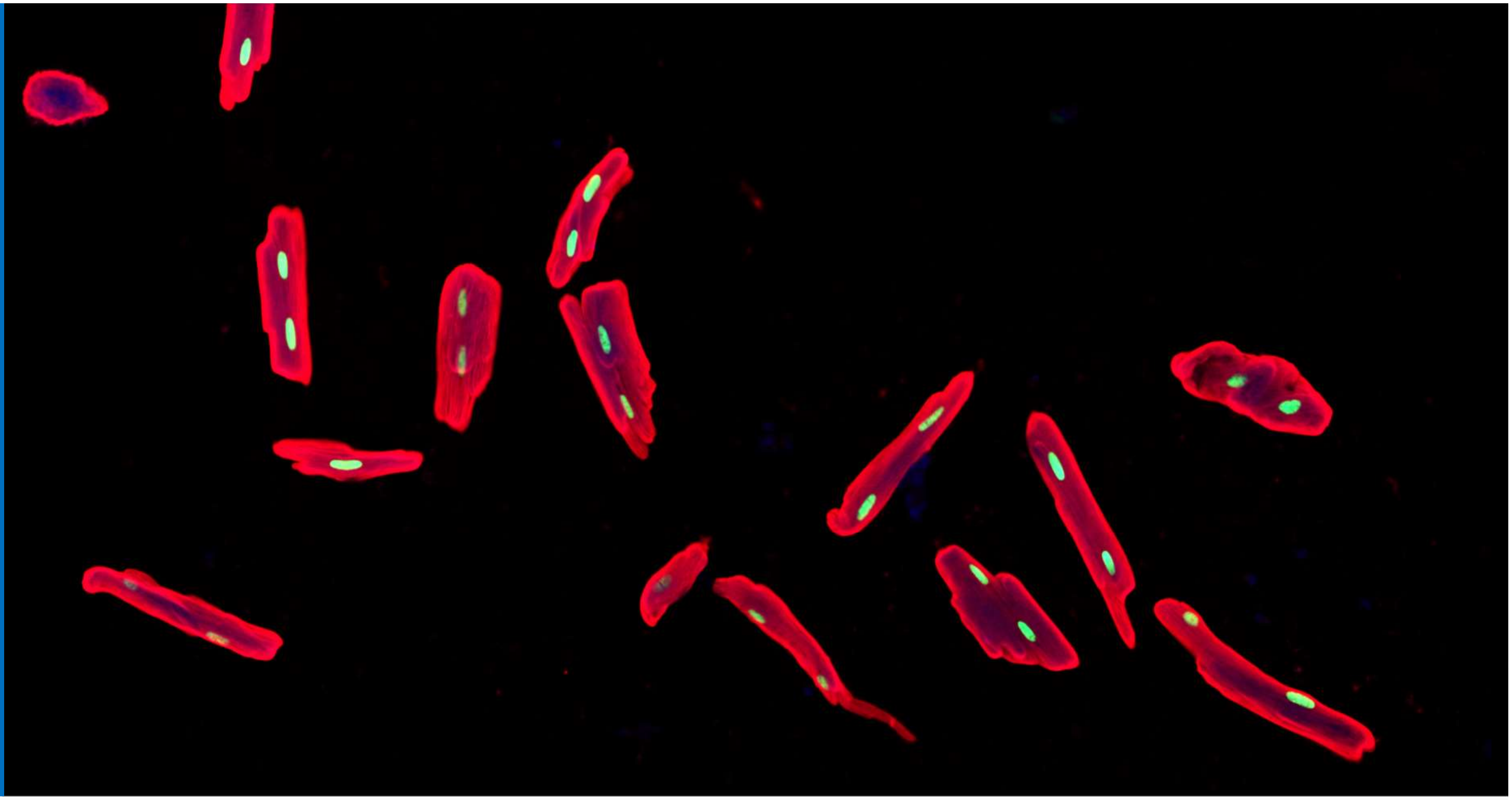


Week 3: Logical operations and Images

MCDB-BCHM 4312-5312

Learning goals

- Logical operations
  - Comparison operators
  - Logical indexing
- Images
  - Reading and displaying images in MATLAB
  - How image data is visualized
  - Numerical data types
  - Brightness and contrast adjustments
- Measuring the radius of circular objects in an image
  - Interactively using imdistline
  - Converting from pixels to physical units
  - Finding circular objects using the circular Hough transform

Logical operations in MATLAB

- Logical data can only have two possible values

`true` or `false`

Example:

```
>> 10 < 2
```

List of logical operators

Operator	Description
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Is equal to
~	Not (flips true to false and vice versa)
~=	Not equal to

Which of the following statements are false?

Operator	Description
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Is equal to
~	Not (flips true to false and vice versa)
~=	Not equal to

a) 5 >= 1

b) 5 >= 5

c) 10 ~= 5

d) ~(10 == 10)

e) ~(5 ~= 10)

Using a logical comparison on a matrix

```
>> A = [1 2; 3 4];
>> A > 2

ans =
  2×2 logical array
   0   0
   1   1
```

Number of elements matching a condition

- The function `nnz` (**N**umber of **N**on-**Z**eros) gives the number of true elements in a logical array

- Use this function to count how many elements satisfy a logical comparison

- Example:

```
>> A = [1 2; 3 4];
>> nnz(A > 2)
```

Logical indexing

You can index elements using a logical array

```
>> A = [1 2 3 4 5 6]
```

Use logical comparison to see which elements are less than 4

```
>> k = A < 4
k =
  1×6 logical array
   1   1   1   0   0   0
```

Use the logical array as an index

```
>> A(k)
ans =
     1     2     3
```

A more concise form:

```
>> A(A < 4)
```

There will be an example of using this in your homework

Images

Reading and displaying images

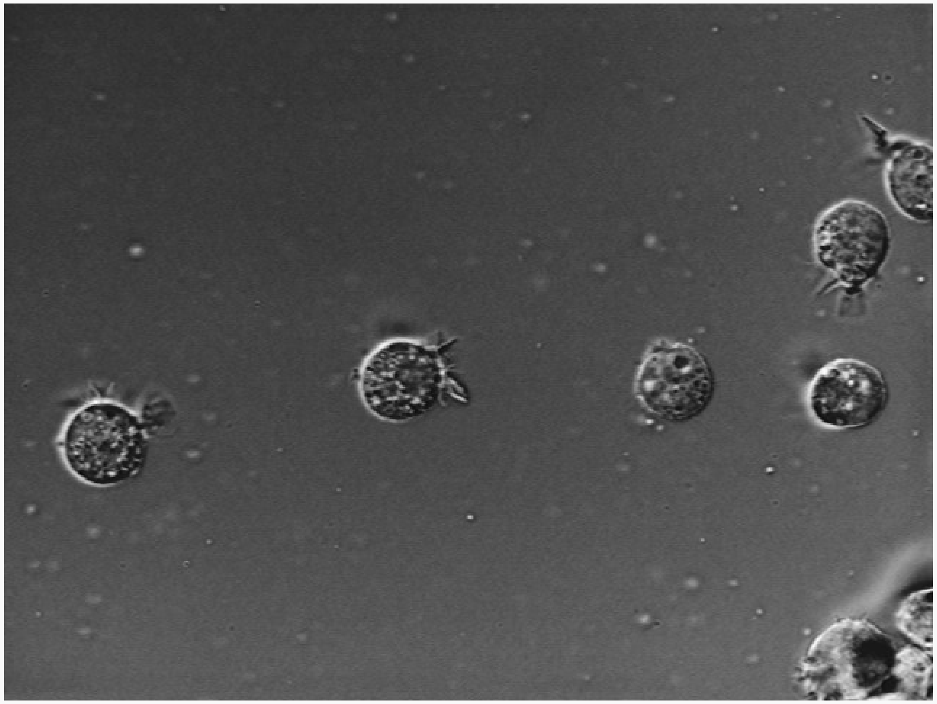
Use `imread` to load image data into a variable

```
>> I = imread('AT3_1m4_01.tif');
```

Demo image that ships with MATLAB

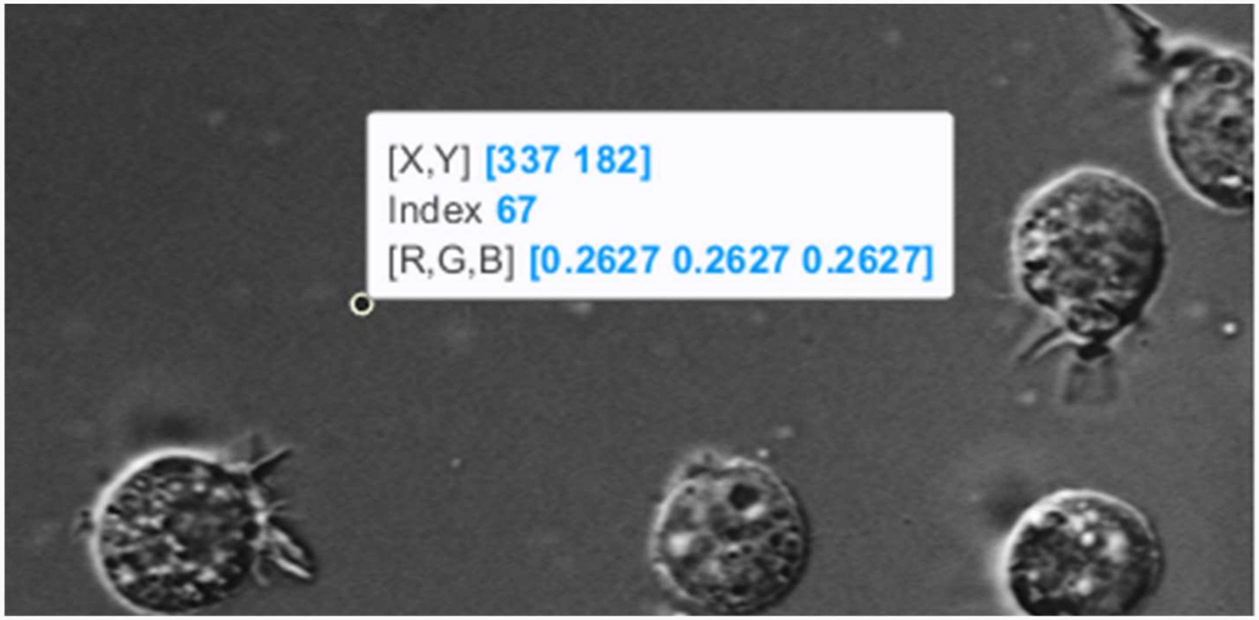
Display the image

```
>> imshow(I)
```



Can use data tips tool to get (x, y) location and intensity (index) of pixel

Reading and displaying images



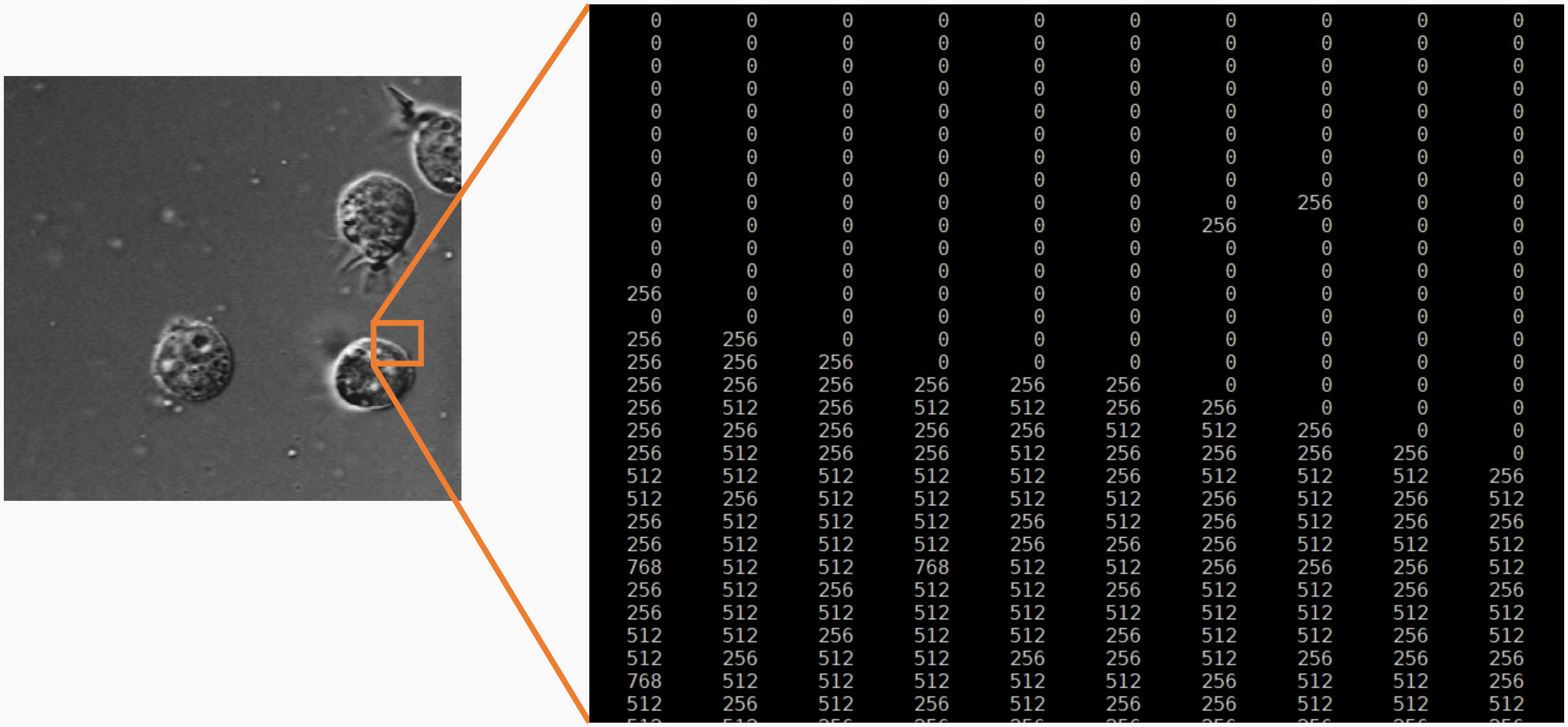
[X,Y] [337 182]
Index 67
[R,G,B] [0.2627 0.2627 0.2627]

Can use data tips tool to get:

- [x, y] - location
- Index – Pixel value (i.e. intensity)
- [R, G, B] – Displayed color value

Image data is proportional to intensity

- **RAW** image data is the intensity of light arriving at the camera

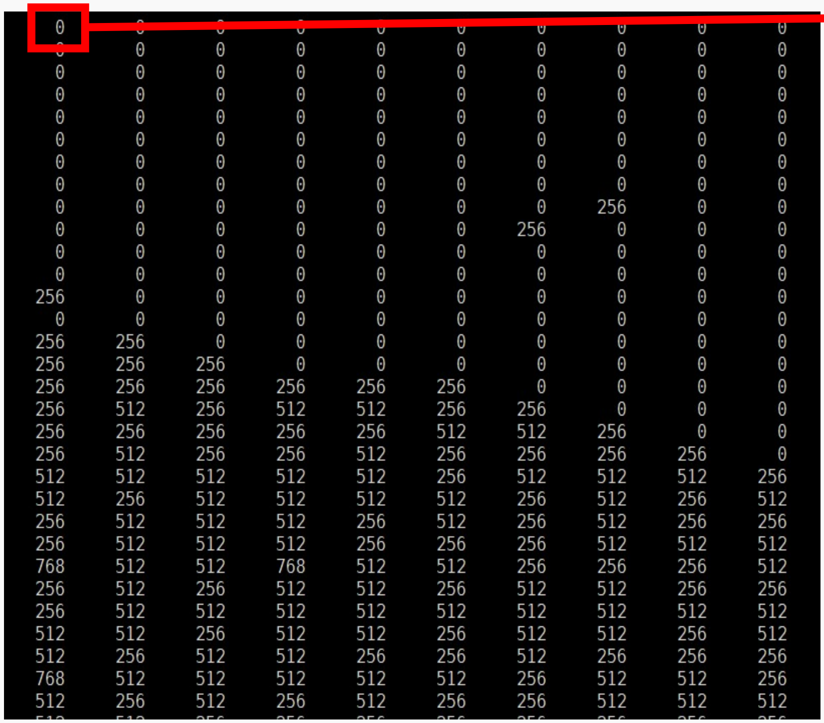


```
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0   256     0     0
     0     0     0     0     0     0   256     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
   256     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
   256   256     0     0     0     0     0     0     0     0
   256   256   256     0     0     0     0     0     0     0
   256   256   256   256   256   256     0     0     0     0
   256   512   256   512   512   256   256     0     0     0
   256   256   256   256   256   512   512   256     0     0
   256   512   256   256   512   256   256   256   256     0
   512   512   512   512   512   256   512   512   512   256
   512   256   512   512   512   512   256   512   256   512
   256   512   512   512   256   512   256   512   256   256
   256   512   512   512   256   256   256   512   512   512
   768   512   512   768   512   512   256   256   256   512
   256   512   256   512   512   256   512   512   256   256
   256   512   512   512   512   512   512   512   512   512
   512   512   256   512   512   256   512   512   256   512
   512   256   512   512   256   256   512   256   256   256
   768   512   512   512   512   512   256   512   512   256
   512   256   512   256   512   256   256   512   512   512
```

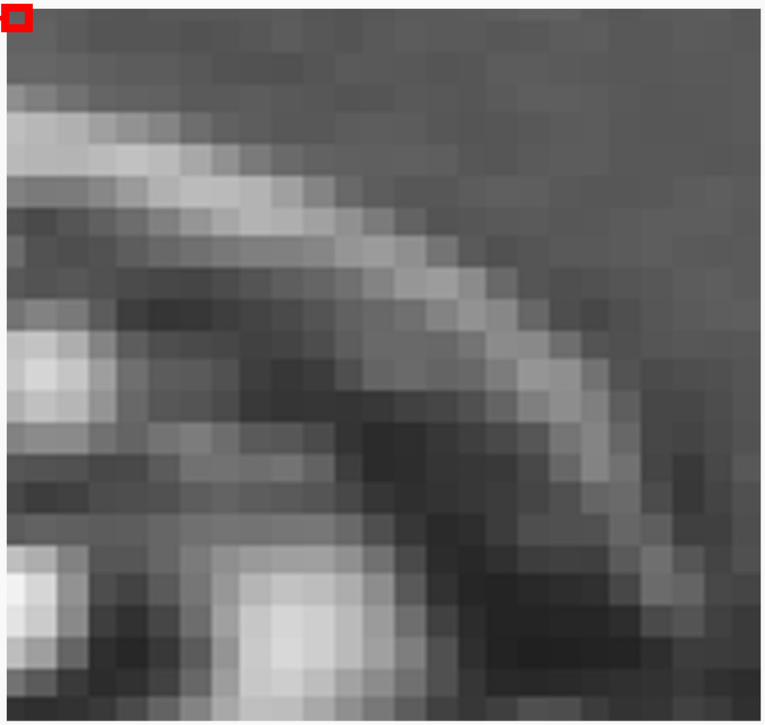
- Higher pixel values = more light detected by camera pixel
- Orientation of image matches matrix

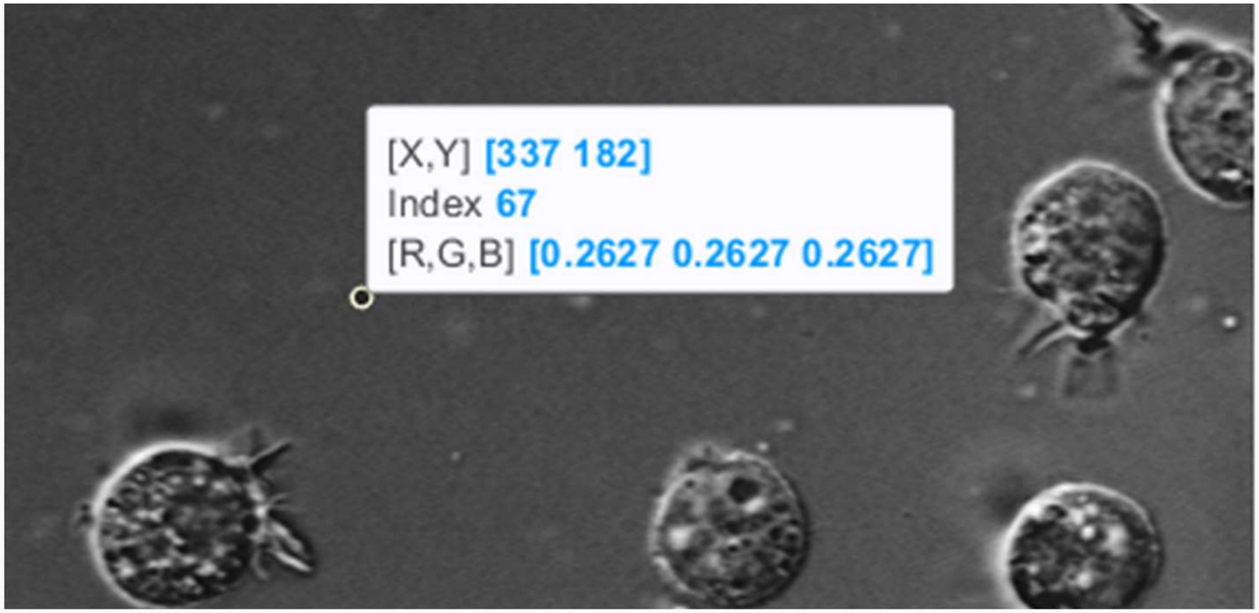
How are numbers transformed into a picture?

- Images are **<u>visual reconstructions</u>** of intensity data
- MATLAB takes this data and draws colored squares when displaying the image
- 1 matrix element = 1 tiny square



```
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0   256     0     0
     0     0     0     0     0     0   256     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
   256     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
   256   256     0     0     0     0     0     0     0     0
   256   256   256     0     0     0     0     0     0     0
   256   256   256   256   256   256     0     0     0     0
   256   512   256   512   512   256   256     0     0     0
   256   256   256   256   256   512   512   256     0     0
   256   512   256   256   512   256   256   256   256     0
   512   512   512   512   512   256   512   512   512   256
   512   256   512   512   512   512   256   512   256   512
   256   512   512   512   256   512   256   512   256   256
   256   512   512   512   256   256   256   512   512   512
   768   512   512   768   512   512   256   256   256   512
   256   512   256   512   512   256   512   512   256   256
   256   512   512   512   512   512   512   512   512   512
   512   512   256   512   512   256   512   512   256   512
   512   256   512   512   256   256   512   256   256   256
   768   512   512   512   512   512   256   512   512   256
   512   256   512   256   512   256   256   512   512   512
   512   512   256   256   256   256   256   256   256   256
```





[X,Y] [337 182]
Index 67
[R,G,B] [0.2627 0.2627 0.2627]

Which of the following statements correspond to the pixel with the (x, y) coordinates shown above?

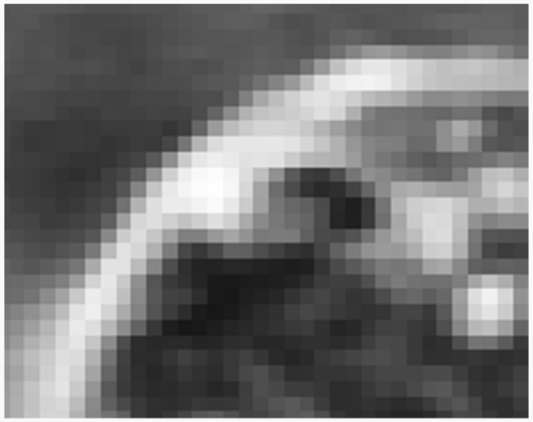
a) `I(182, 337)`

b) `I(337, 182)`

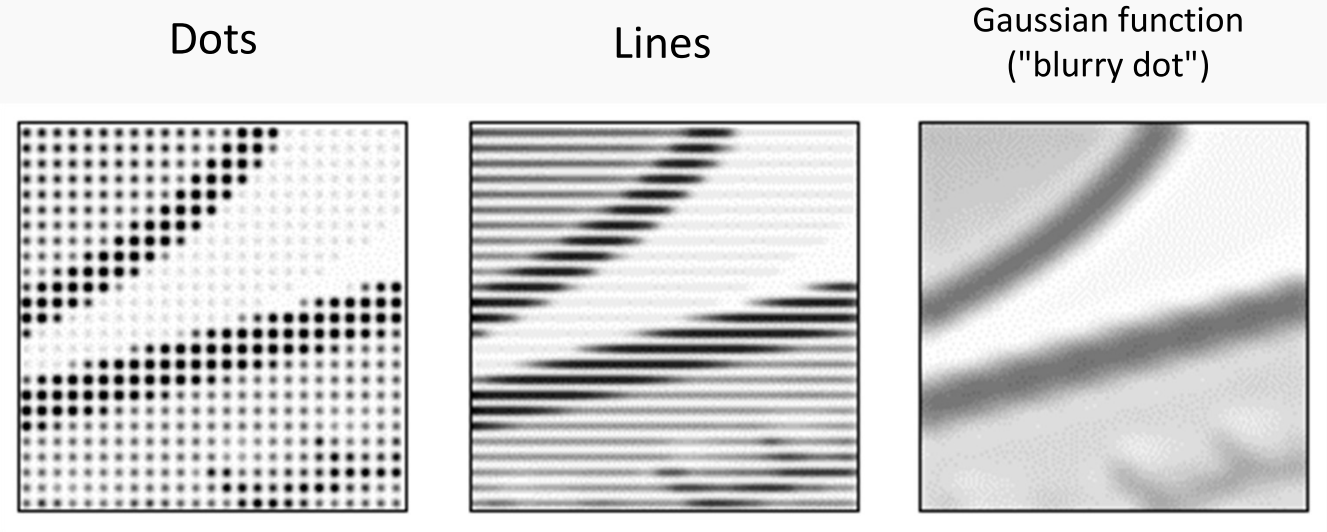
Are pixels actually little squares?

NO!

- Pixels are the measurement of intensity at a point in space



Most common is to display as squares



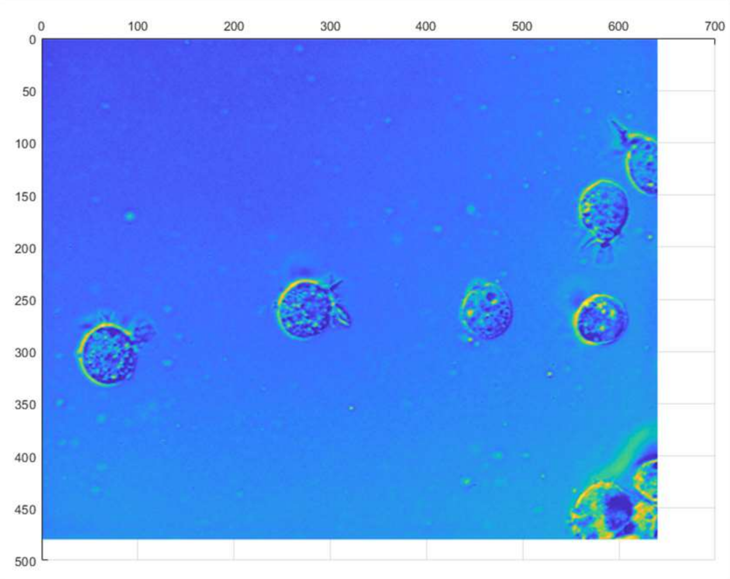
But other drawing methods exist

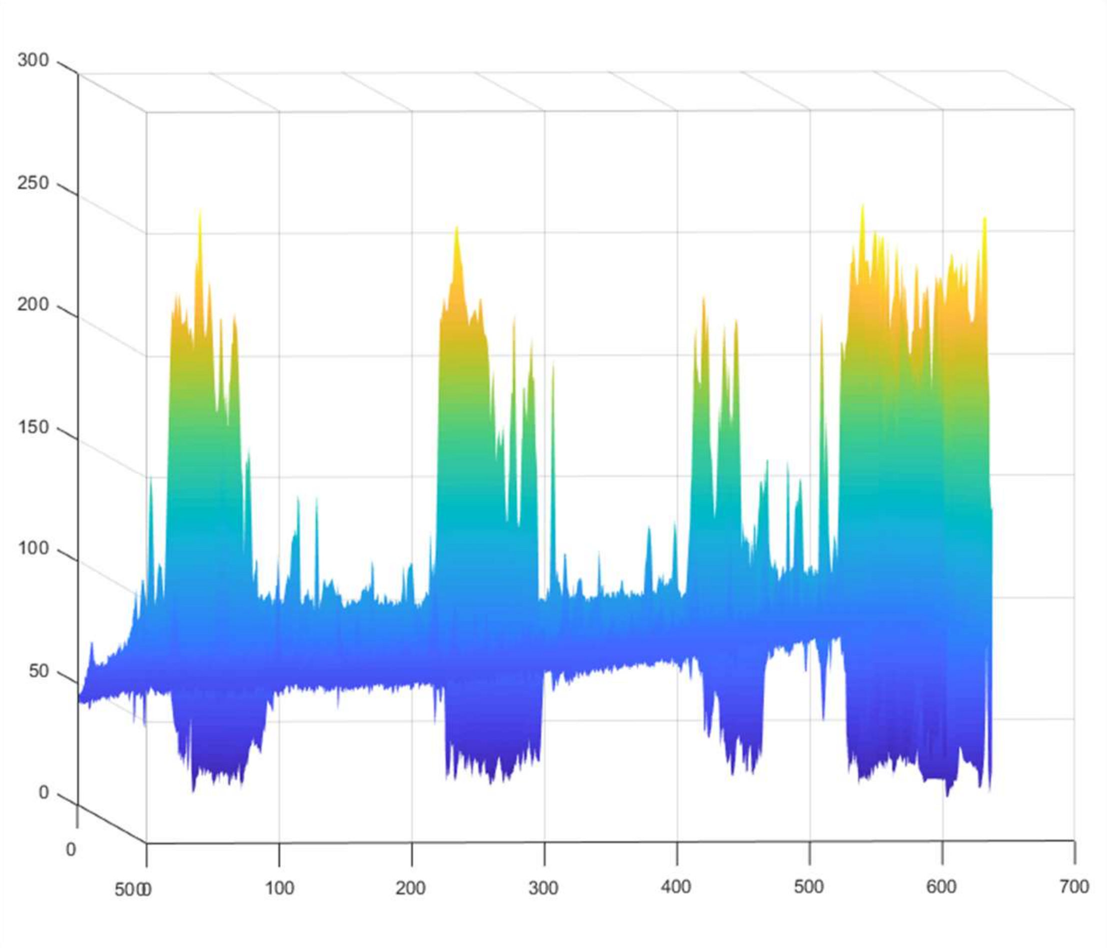
A pixel is not a little square by Alvy Ray Smith

Image credit: Wikipedia

Images can also be plotted as 3D surfaces

```
>> surf(I)
>> shading interp
```





We'll see this again in Lecture 5

Changing how bright images appear

```
>> I = imread('mri.tif');
>> imshow(I)
```

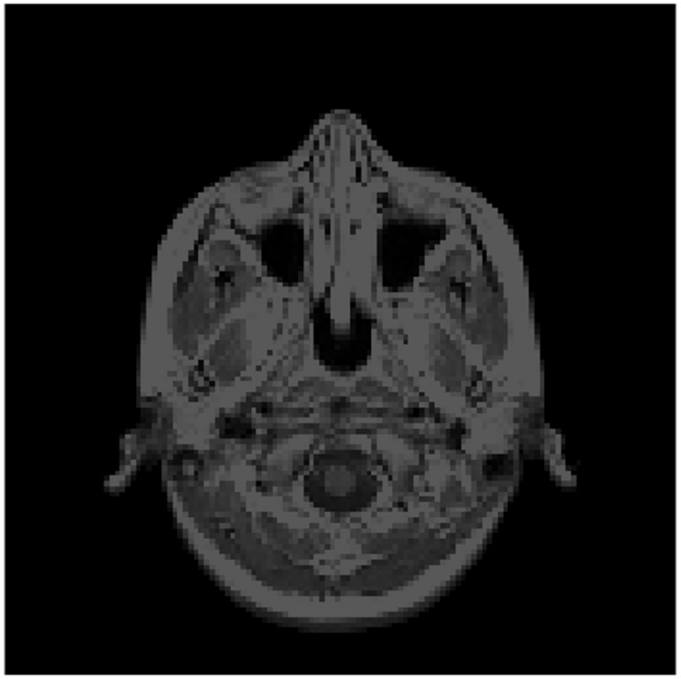


Image looks dark… why?

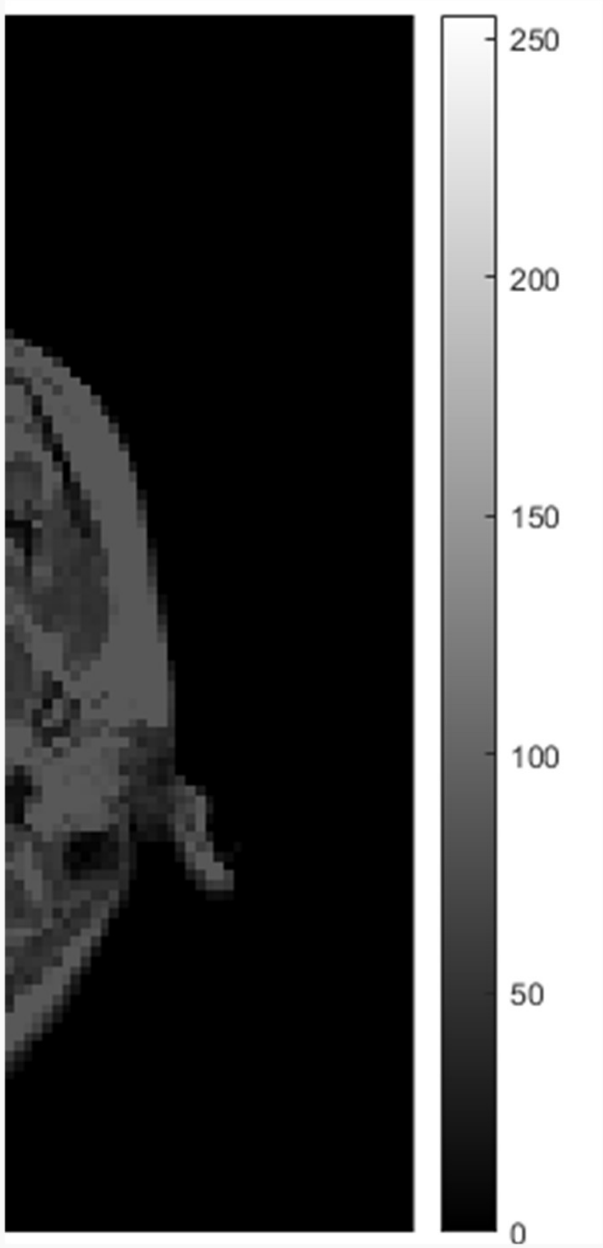
What is the highest pixel value in the image?

The colorbar

```
>> colorbar
```

Shows which color is used to represents which pixel value

Important to show when displaying quantitative data



Pixel value

Numerical data classes

- The colorbar is scaled depending on the **data type or "class"** of the image

- You can check the class of a variable in the Workspace

- Image data are typically **u**nsigned **int**egers (`uint8` or `uint16`)

- The number after `uint` tells you the **bit depth**

Example of unsigned integer

`uint8` means 8-bit integer

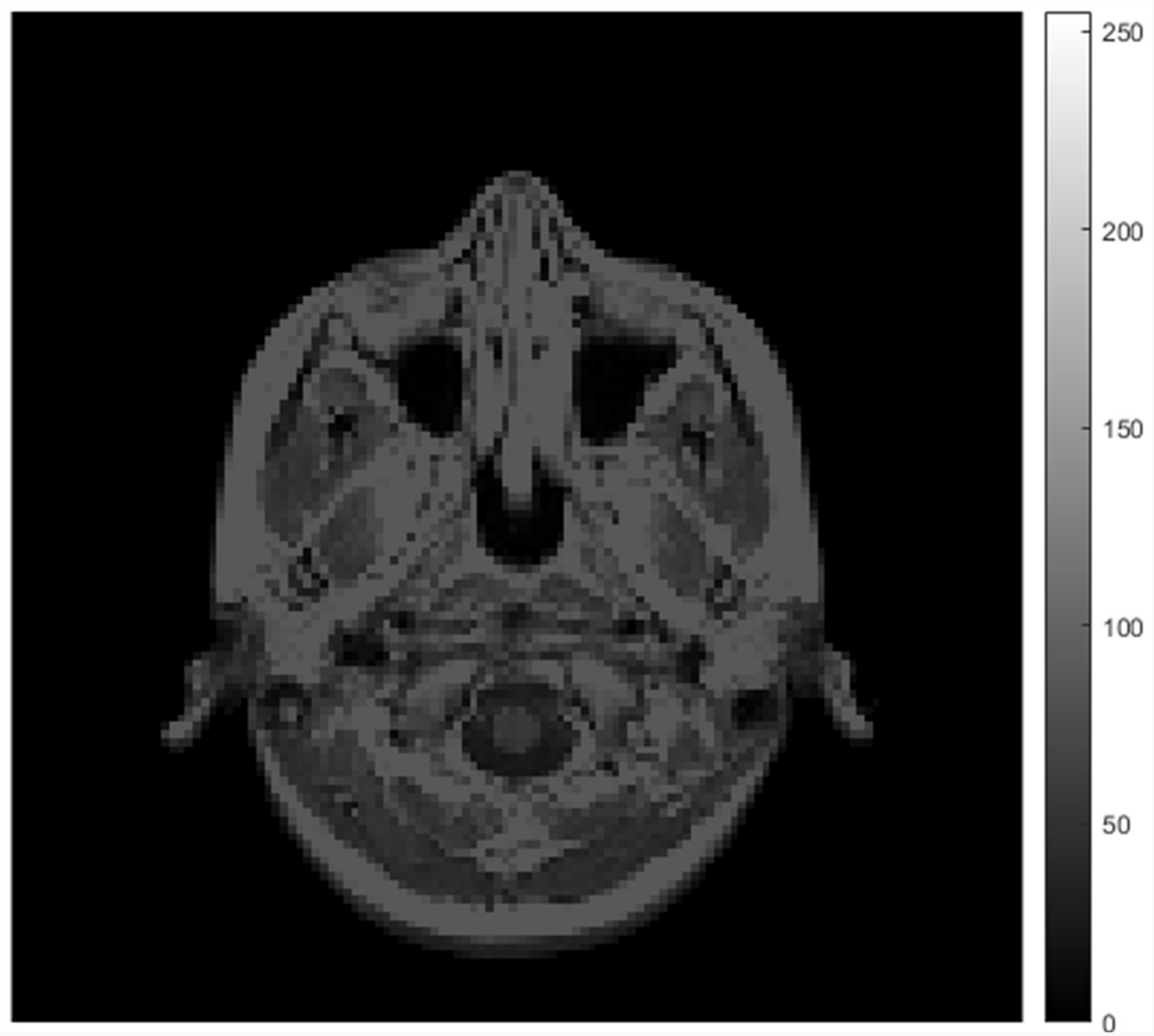
Numbers go from 0 → $(2^8 - 1)$ = 0 → 255

Positive numbers only

No decimal places

Default scaling using imshow

uint8 image



255

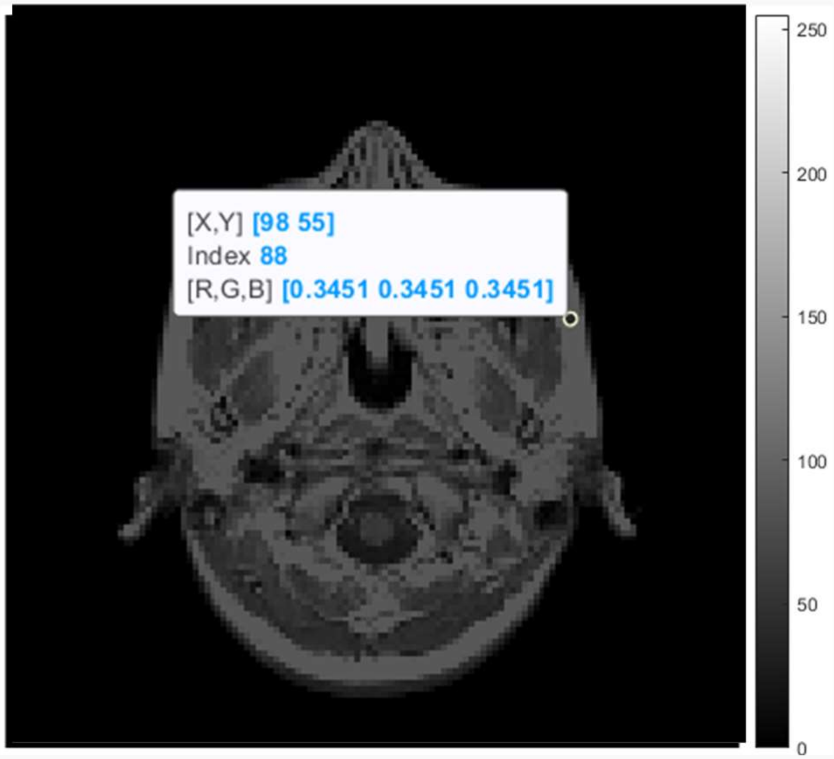
0

Changing the displayed color scale

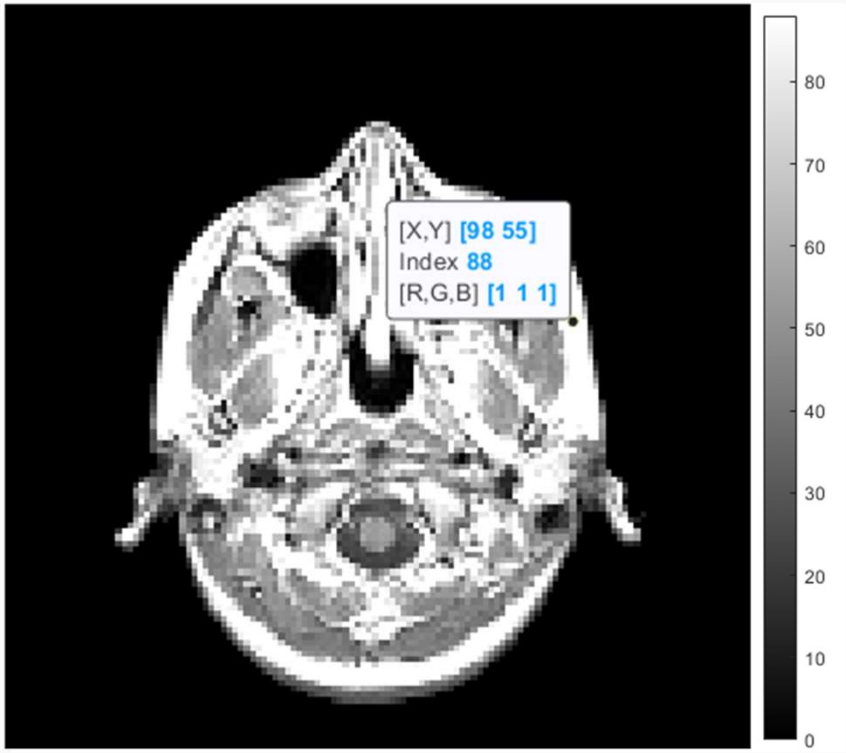
```
imshow(image, [low, high])
```

```
>> imshow(I, [0, 88])
>> colorbar
```

Original intensity values have not been changed



```
[X,Y] [98 55]
Index 88
[R,G,B] [0.3451 0.3451 0.3451]
```



```
[X,Y] [98 55]
Index 88
[R,G,B] [1 1 1]
```

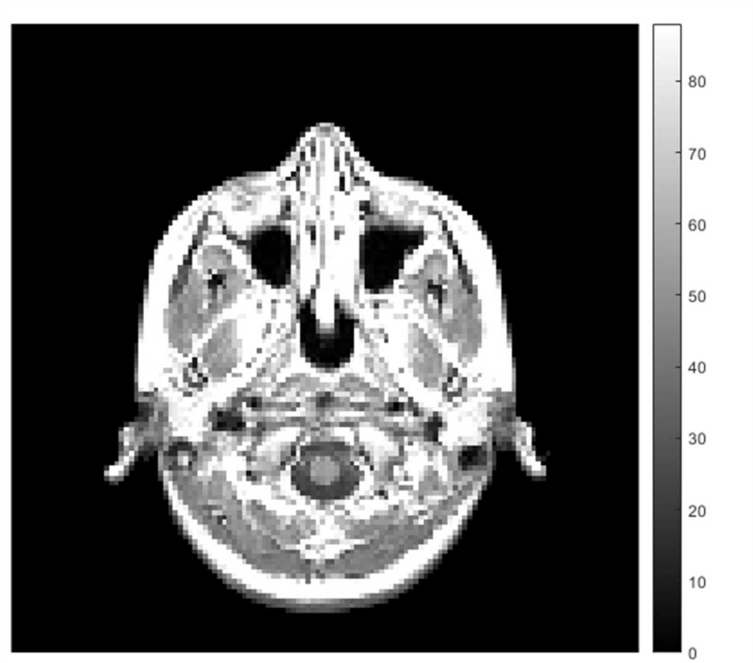
Automatic display scaling

```
imshow(I, [])
```

Empty matrix

is equivalent to

```
imshow(I, [min(I(:)), max(I(:))])
```



TIP: You might need this for your homework

The `double` class

- One other common number format is the `double` (short for "double-precision") – 64-bits

- Unlike unsigned integers, `double` can have decimal places and negative numbers

- `double` is the default data class for numbers in MATLAB

- To convert from unsigned integers to double

```
Idbl = double(I)
```

Default scaling for double

```
imshow(I)
```

What is the default color bar scale for `double`?

a) -1e32 to +1e32

b) min(image) to max(image)

c) 0 to 1

d) 0 to max(image)

Default scaling for double

```
imshow(image)
```

What is the default color bar scale for `double`?

a) -1e32 to +1e32

b) min(image) to max(image)

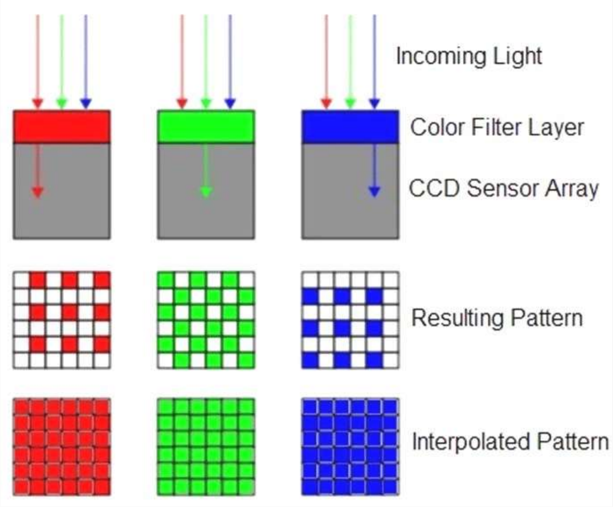
c) 0 to 1

d) 0 to max(image)

What about color images?

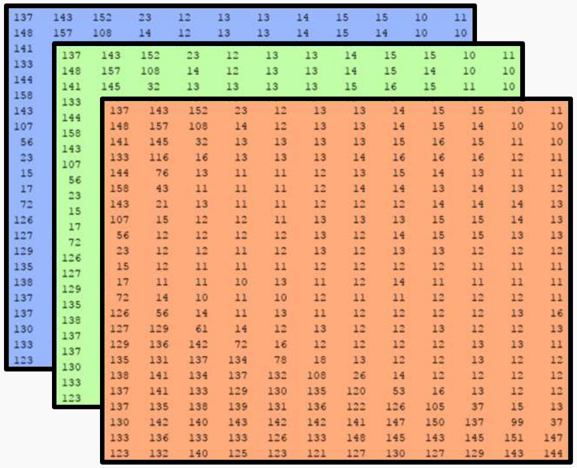
```
>> rgb = imread('tissue.png');
```

Color cameras



What is the size of the image?

The 3rd dimension is color



137	143	152	23	12	13	13	14	15	15	10	11
148	157	108	14	12	13	13	14	15	14	10	10
141	145	32	13	13	13	13	15	16	15	11	10
133	116	16	13	13	13	14	16	16	16	12	11
144	76	13	11	11	12	13	15	14	13	11	11
158	43	11	11	11	12	14	14	13	14	13	12
143	21	13	11	11	12	12	12	14	14	14	13
107	15	12	12	11	13	13	13	15	15	14	13
56	12	12	12	12	13	12	14	15	15	13	13
23	12	12	11	12	13	12	13	13	12	12	12
15	12	11	11	11	12	12	12	12	11	11	11
17	11	11	10	13	11	12	14	11	11	11	11
72	14	10	11	10	12	11	11	12	12	12	11
126	56	14	11	13	11	12	12	12	12	13	16
127	129	61	14	12	13	12	12	13	12	12	13
129	136	142	72	16	12	12	12	12	13	13	11
135	131	137	134	78	18	13	12	12	13	12	12
138	141	134	137	132	108	26	14	12	12	12	12
137	141	133	129	130	135	120	53	16	13	12	12
137	135	138	139	131	136	122	126	105	37	15	13
130	142	140	143	142	142	141	147	150	137	99	37
133	136	133	133	126	133	148	145	143	145	151	147
123	132	140	125	123	121	127	130	127	129	143	144

Order: Red, Green, Blue

aka RGB images

Which of these commands retrieves the BLUE channel?

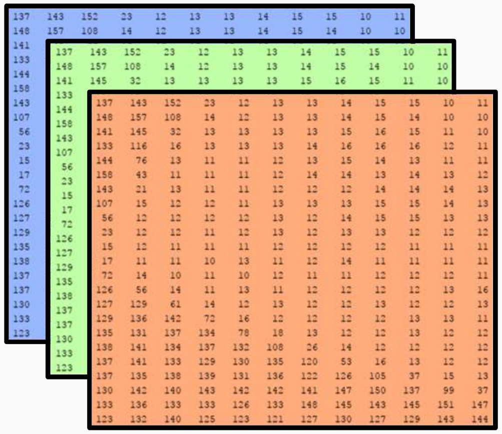
```
>> rgb = imread('tissue.png');
```

```
a) blue = rgb(:, :, 1)

b) blue = rgb(3, :, :)

c) blue = rgb(:, :, 3)

d) blue = rgb(:, 3)
```



Just an extension of matrix indexing

rgb(row, column, color)

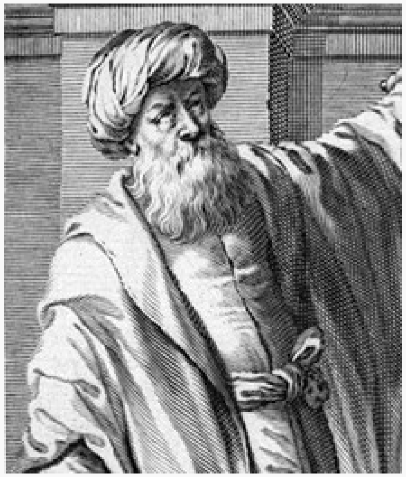
Does light change wavelengths when mixing?

For example:

Red photon (650 nm) + blue photon (400 nm)
= magenta photon (500 nm)

a) Yes

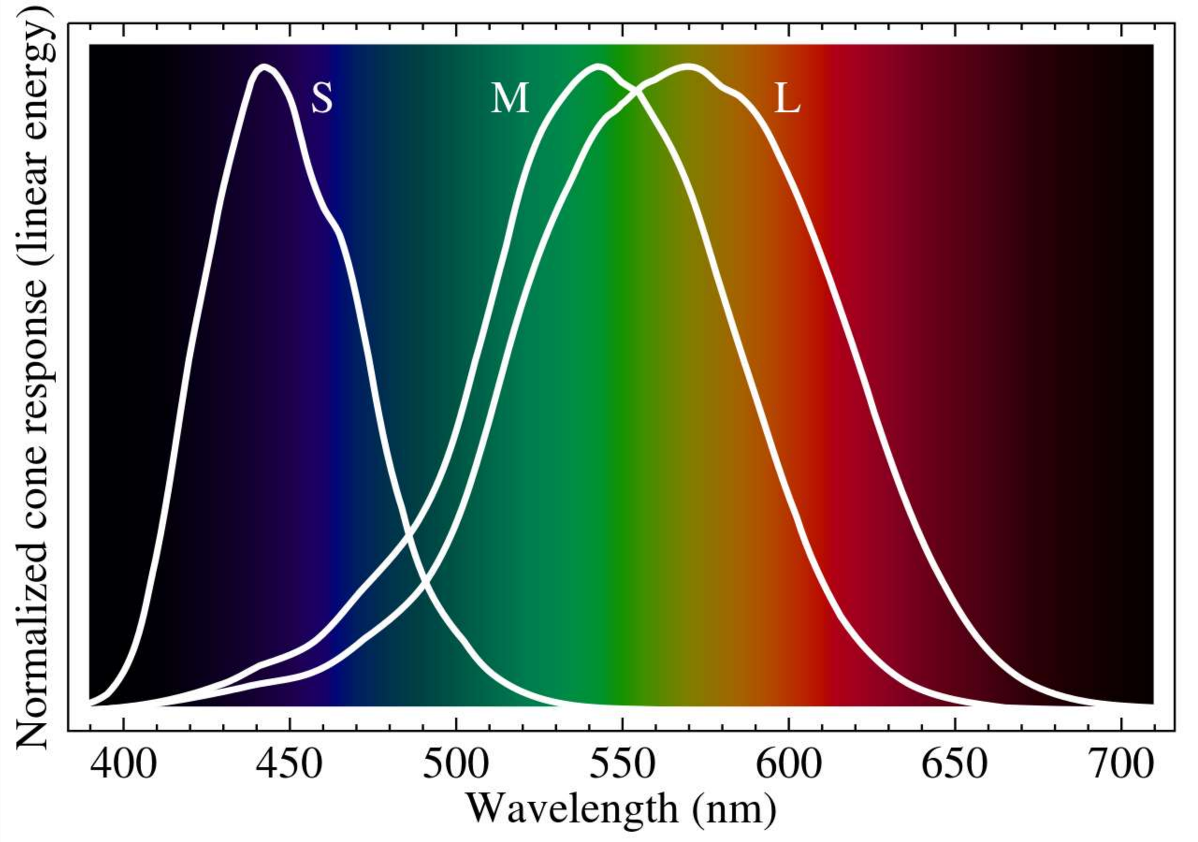
b) No

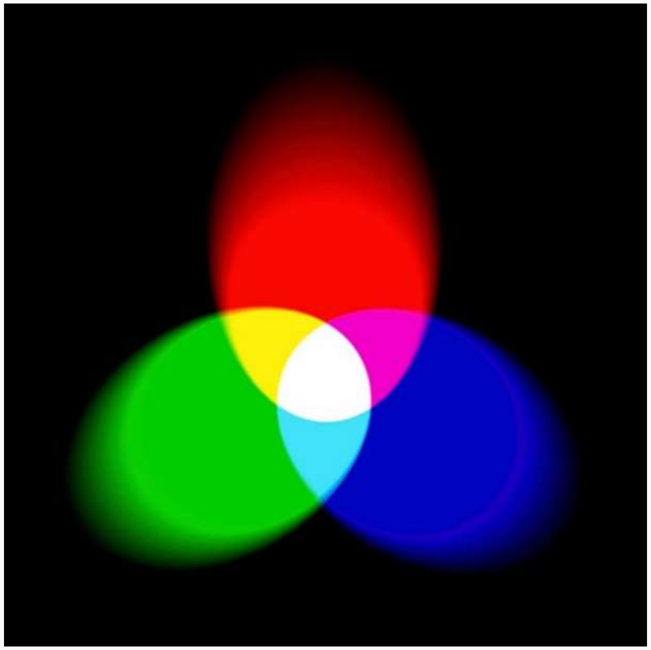


Ibn al-Haytham

Vision happens in the brain

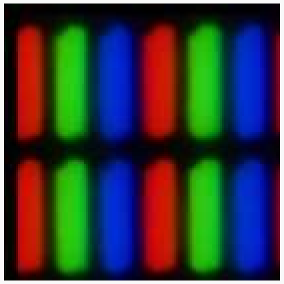
How humans perceive color





The pixel value in each color plane

- Each monitor pixel is made up of three elements



- The pixel value in each color plane tells the computer how bright each display element should be

Red = 100%
Green = 0%
Blue = 0%

Red = 50%
Green = 0%
Blue = 0%

Red = 3%
Green = 0%
Blue = 0%

Try this at home

```
>> rgbImg = zeros(50, 50, 3)
>> rgbImg(:, :, 1) = 0.5;
>> imshow(rgbImg)
```

The pixel value in each color plane

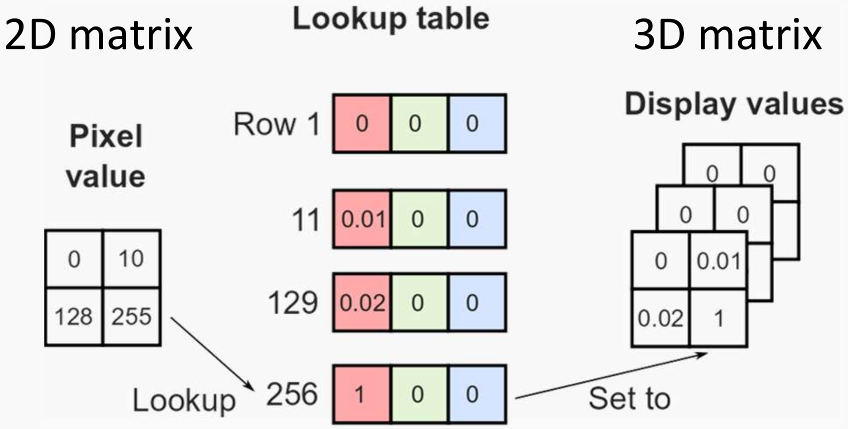
- The scale of the pixel value depends on the ***data type*** of the images

	Red = 100%
	Green = 0%
	Blue = 0%
		Red = 50%
		Green = 0%
		Blue = 0%
			Red = 3%
			Green = 0%
			Blue = 0%

double	[1, 0, 0]	[0.5, 0, 0]	[0.03, 0, 0]
uint8	[255, 0, 0]	[127, 0, 0]	[8, 0, 0]

False color images

- Microscope cameras are just CCD arrays
- Color microscope images are usually **false colored**
- The imaging software converts the original 2D matrix to a 3D matrix



2D matrix

Lookup table

3D matrix

Pixel value

0	10
128	255

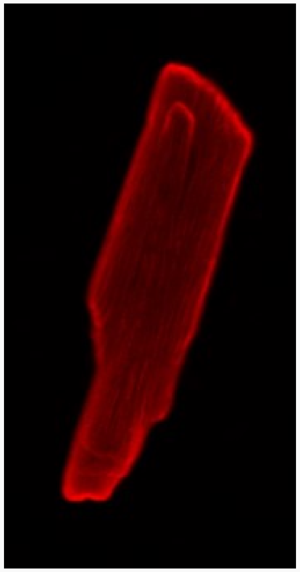
Lookup

Row 1	0	0	0
11	0.01	0	0
129	0.02	0	0
256	1	0	0

Set to

Display values

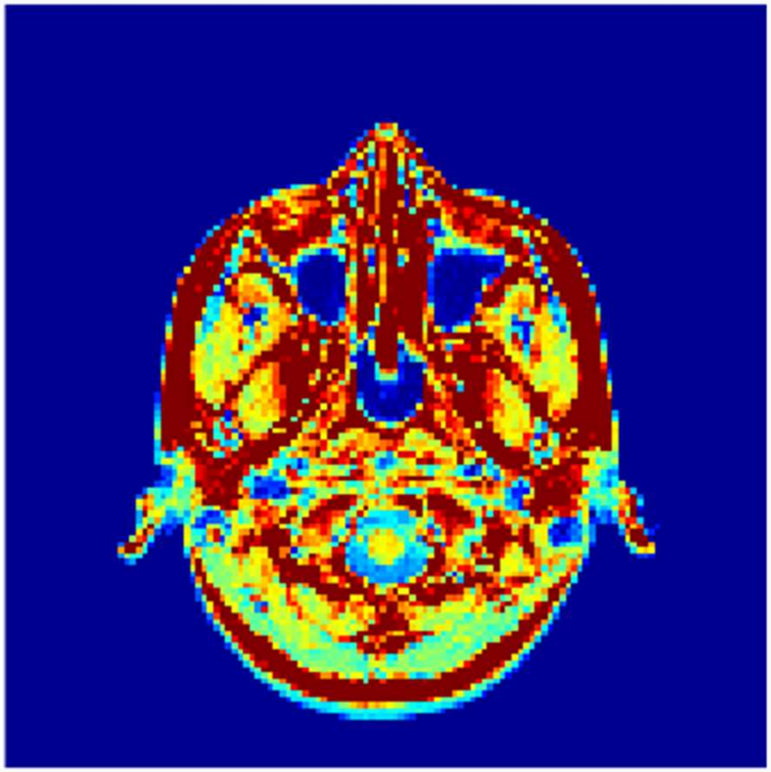
0	0.01
0.02	1



Changing the displayed color map

- You can change the color map of displayed greyscale images

```
>> imshow(I, [])
>> colormap('jet')
```



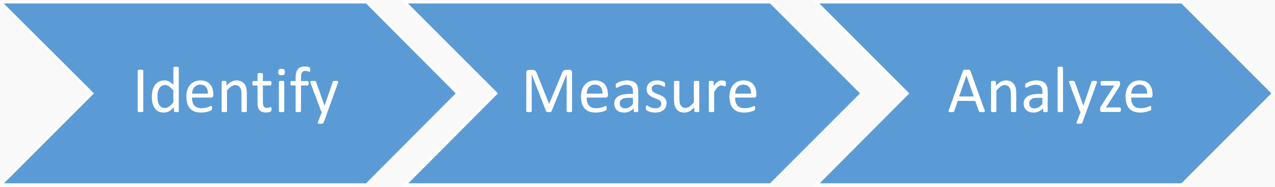
Summary

- Using imread and imshow
- Image data is measured intensity
- Images are reconstructions of the intensity data
- Color in microscope images are (generally) false

Questions?

Image analysis

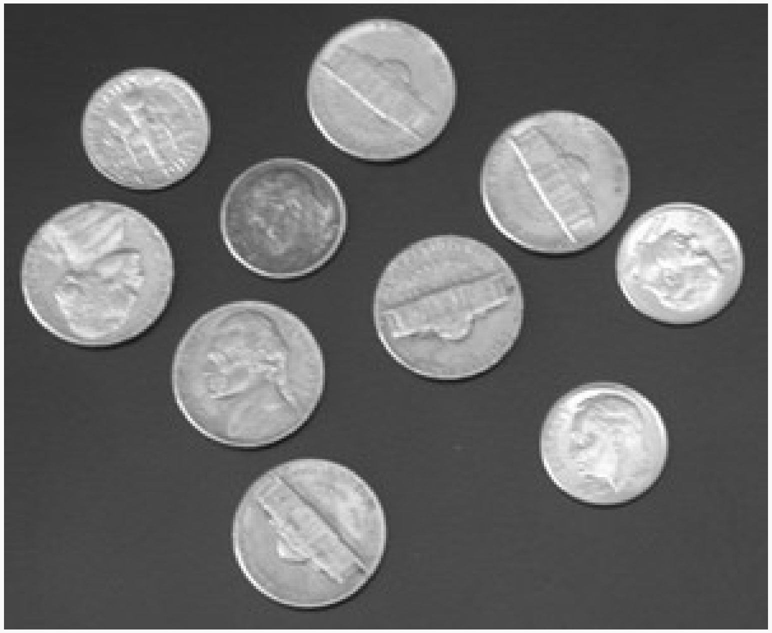
- Image analysis is the process of extracting quantitative information from images



Identify

Measure

Analyze



- Count the number of coins
- Measure the diameter of the coins

Workflow

Identify

Measure

Analyze

1. Load image and display

2. Identify the coins

3. Measure the diameter

4. Count the total value of coins (in homework)

Step 1:

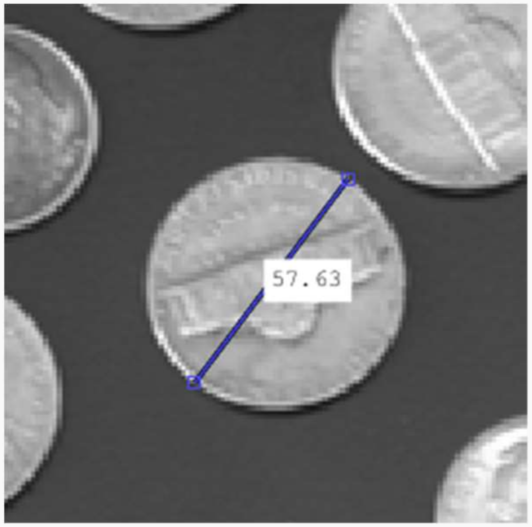
Read in and display the image `'coins.png'`

Measuring the distance between two points

1. Display the image and make sure that the figure is selected (MATLAB keeps track of the last active figure)

2. Use `imdistline` to measure the diameter of a coin in pixels (you might find it easier if you maximize the figure window)

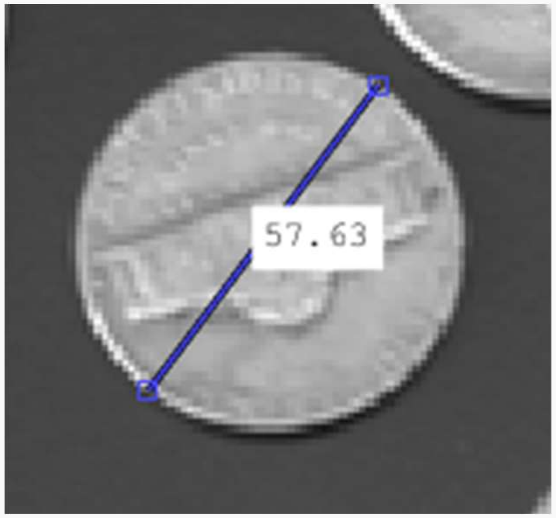
A manual approach – useful for getting quick estimates and for sanity checks



Converting from pixels to microns

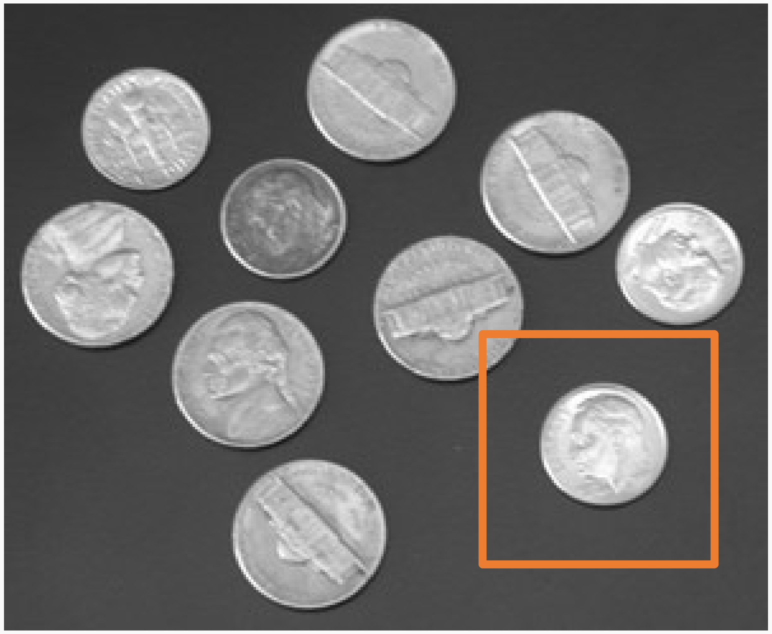
- `imdistline` displays distance in pixels
- For this image, each pixel represents a length 0.368 mm

$$\text{Length(mm)} = \text{Length(pixels)} \times \frac{\text{mm}}{\text{pixels}}$$



```
>> diam_in_mm = 57.63 * 0.368
```

Practice: Measure the dimeter of a dime



Px size = 0.368 mm

- According to the US Treasury, the diameter of a dime should be

17.91 mm

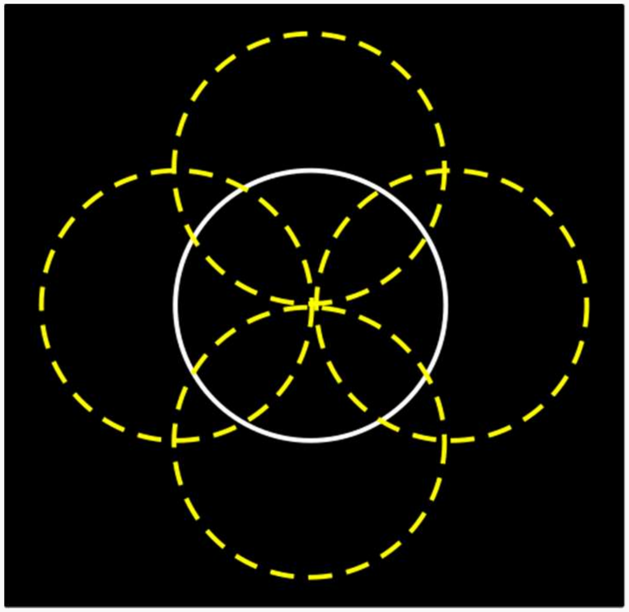
What did you get?

Computationally detecting and counting circular objects

- Detecting circular objects can be achieved using the **circular Hough transform (CHT)**

Basic principle:

If you draw circles around the edge of a circle, the point where the drawn circles intersect will be the center of the original circle

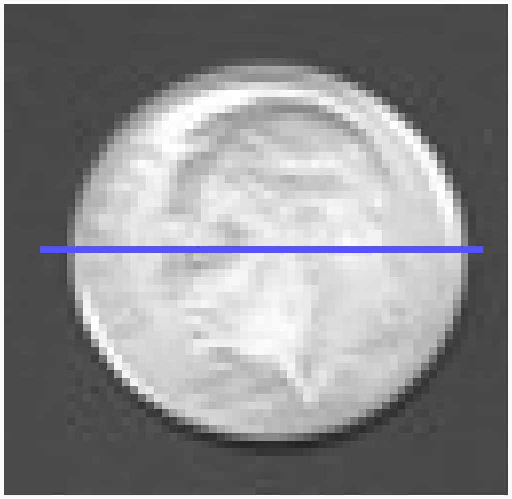


Steps in the Circular Hough Transform

1. Find the edges of objects in an image

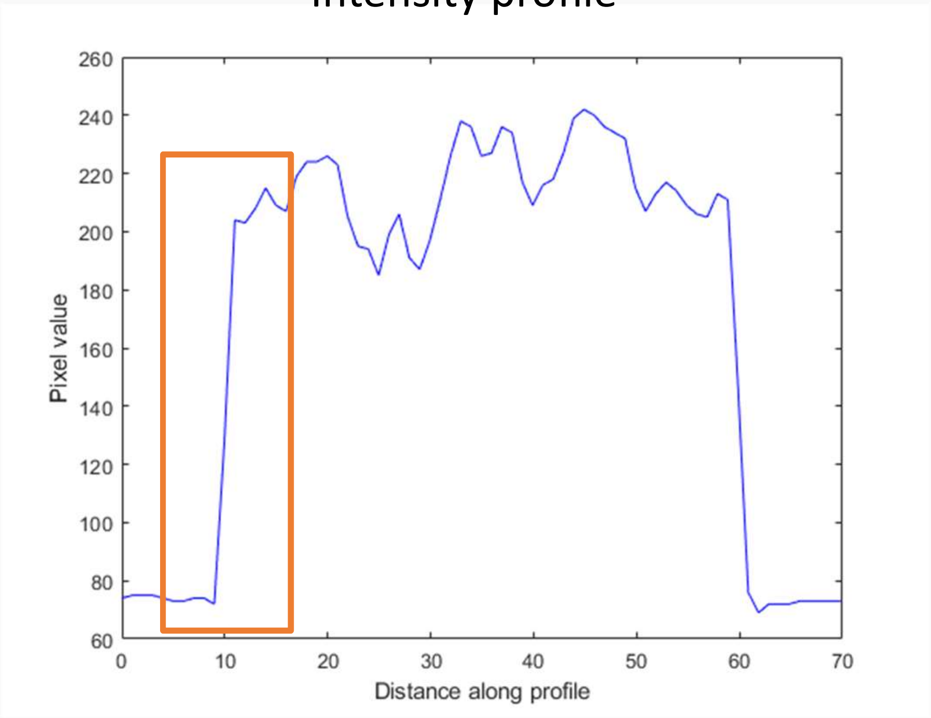
You don't need to know how to program the algorithm, but you do need to know conceptually how it works, and its advantages and disadvantages

How are edges defined?



`improfile`

Intensity profile

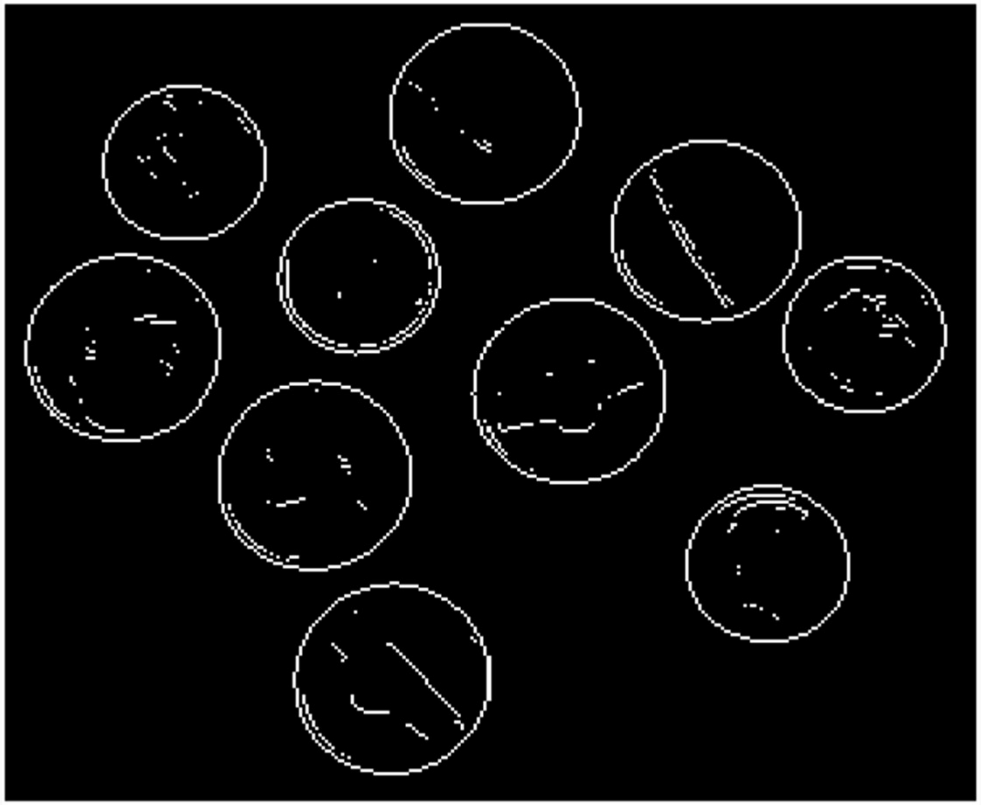


Edges of objects have steep changes in intensity

Example of detected edges

```
>> M = edge(I);
>> imshow(M)
```

Sobel operator: Computes the difference between neighboring pixels



← Logical array

Steps in the Circular Hough Transform

1. Find the edges of objects in an image

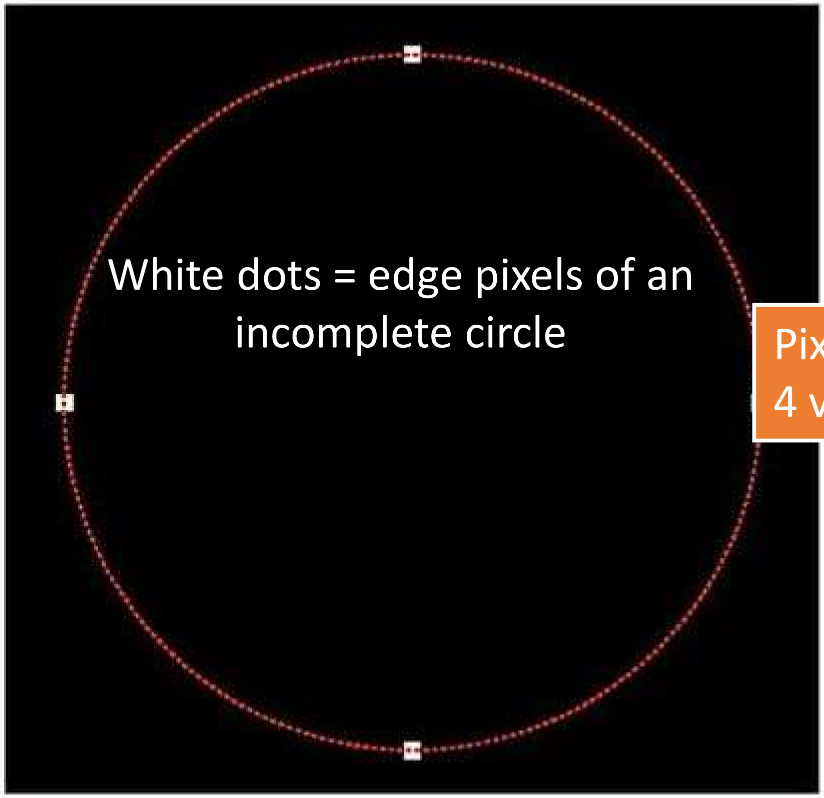
2. Draw circles along each edge pixel found

3. Look for intercept point – usually set a threshold i.e. must have at least 4 overlapping circles

You don't need to know how to program the algorithm, but you do need to know conceptually how it works

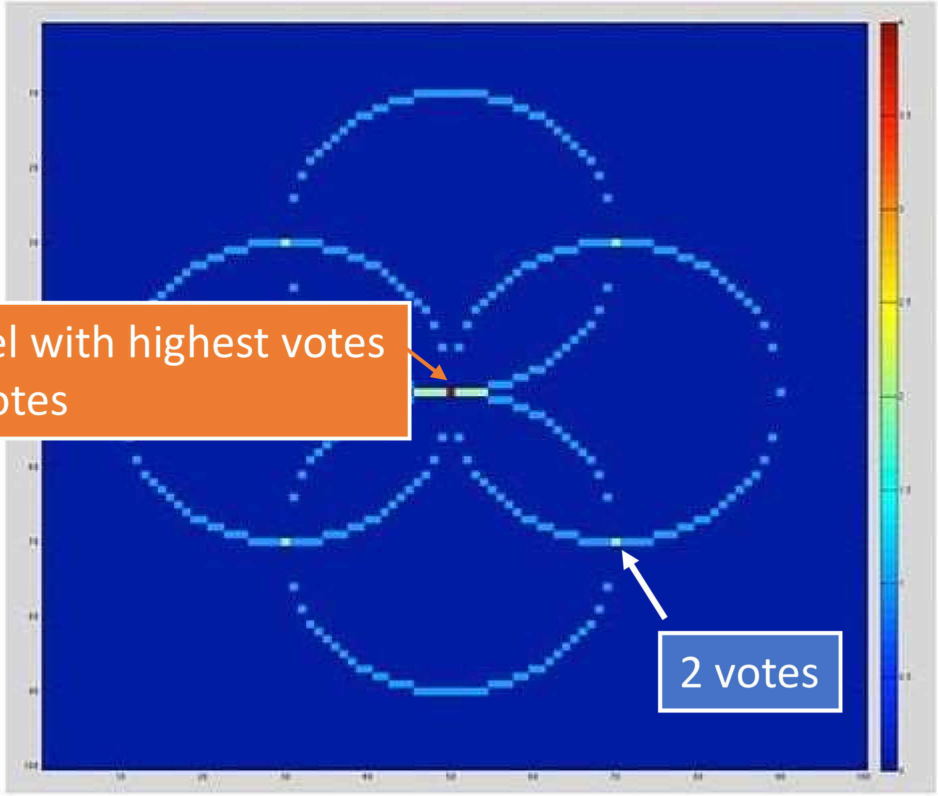
Voting procedure for the Hough Transform

- A "vote" = number of times a line is drawn on a pixel



White dots = edge pixels of an incomplete circle

Number of votes (accumulation array)



Pixel with highest votes
4 votes

2 votes

Image credit: Wikipedia

Applying the CHT in MATLAB

Syntax:

```
[centers, radii] = ...
    imfindcircles(image, [min_radius, max_radius])
```

Procedure:

1. Use `imdistline` to estimate the diameter of the object in pixels

2. Run `imfindcircles` to find circles

Please save your code in a script – you will need it for Question 1 of the homework

Output of imfindcircles

`centers` = Each row is (x, y) position of circle

`radii` = Each row is radius of circle

- How to check if the correct objects were found?

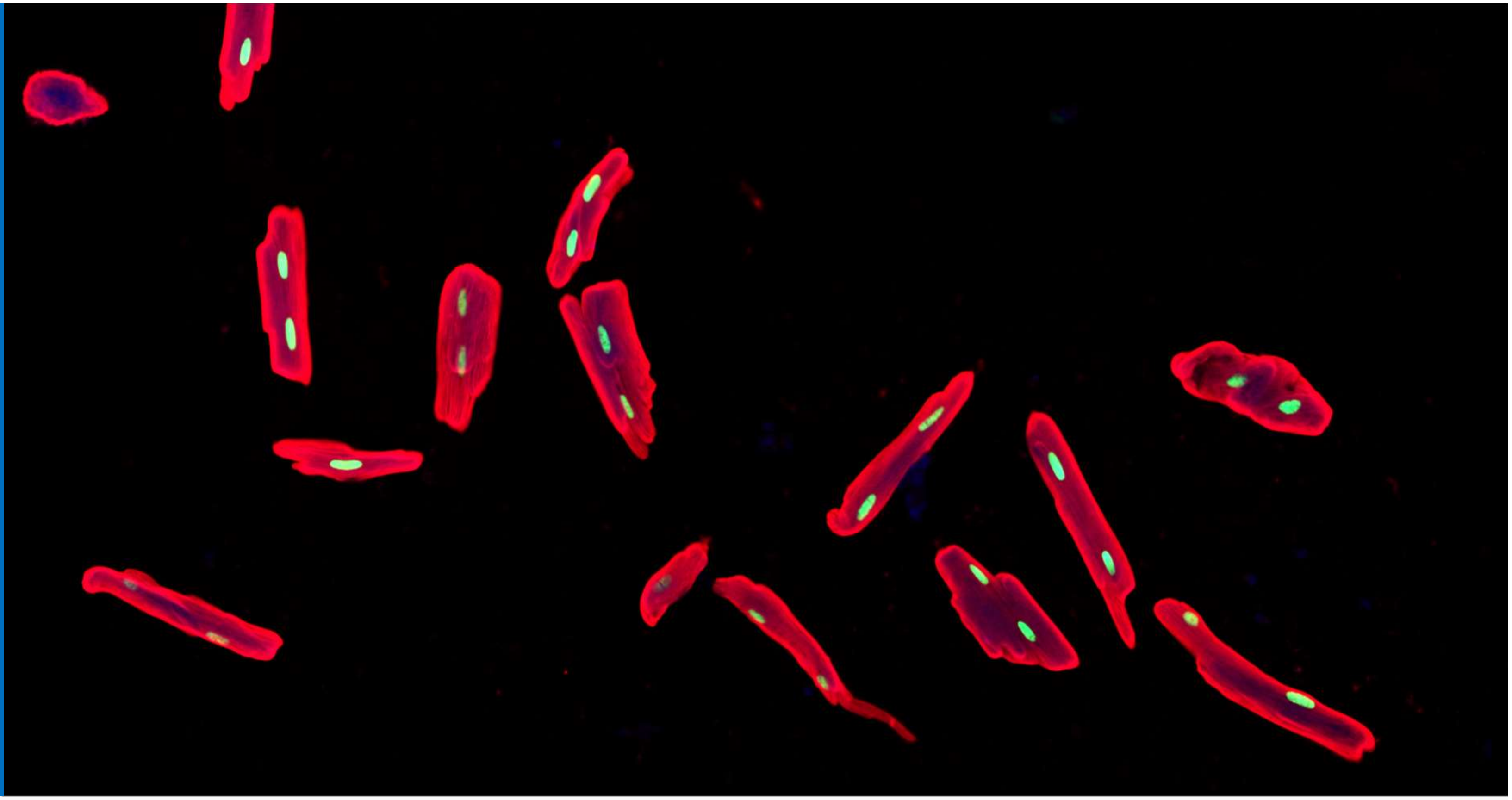
Show circles using the function `viscircles`

```
imshow(I)
viscircles(centers, radii)
```

Applications of the circular Hough transform

- Useful technique for detecting circular cells such as yeast cells and cell nuclei

- Homework will have you working on brightfield images of yeast



Week 3: Logical operations and Images

MCDB-BCHM 4312-5312