

ENGG1801

Engineering Computing

Lecture 8

Images & Movies

School of Information Technologies
The University of Sydney, Australia
canvas.sydney.edu.au

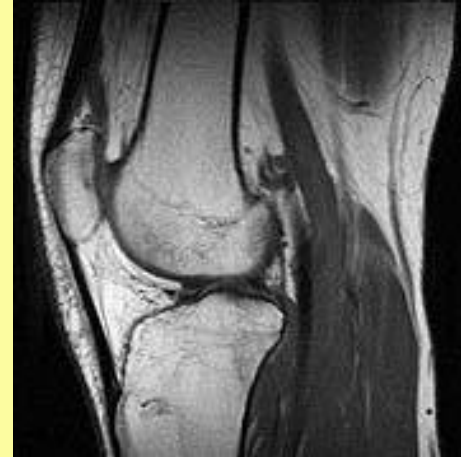
Images

- Often, our data consists of raw images
- We can process digital images in a similar fashion to matrices



Photo of Opera House

Ref: <http://www.sydneyoperahouse.com/about/photogallery.aspx>

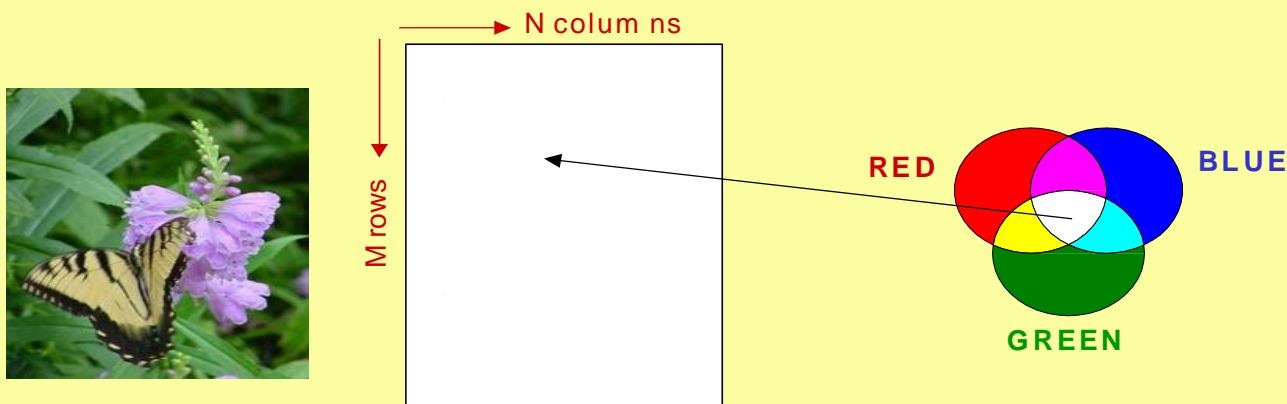


MRI image

Ref: http://en.wikipedia.org/wiki/Magnetic_resonance_imaging

Images

- A digital image is viewed as an $m \times n$ matrix of picture elements called *pixels*
- For a color image, a pixel is a mixture of 3 primary colors: red, green and blue
 - The more of each color, the closer to white



Ref: D. Smith,
Engineering
Computation with
Matlab (2008)

Images

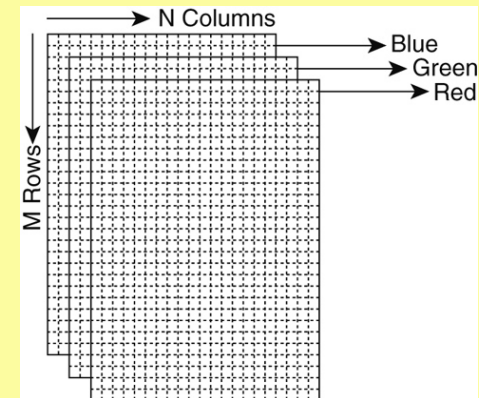
- A color is typically represented with 8 bits
 - 2^8 possible values for each primary color
 - Values range from **0** to **255** inclusive
 - When combining the 3 primary colors [R,G,B], there are **$256 \times 256 \times 256 = 2^{24} \approx 17$ million** possible color combinations (more than the human eye can distinguish)

Image representations

- Images can be represented in 3 ways:

1) True color image

- An $m \times n \times 3$ array
- 3 primary colors for each pixel
- Used for photo quality images



Ref: D. Smith,
Engineering
Computation with
Matlab (2008)

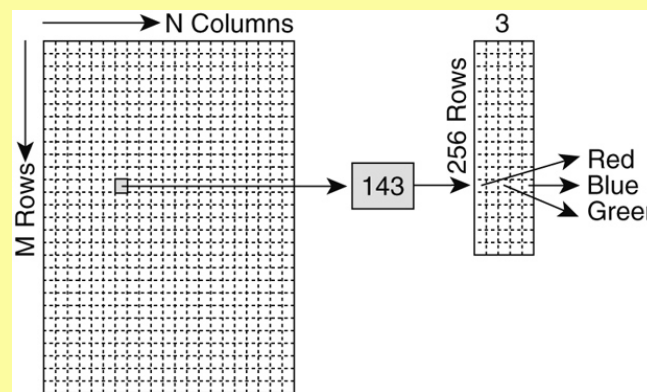
2) Gray scale image

- An $m \times n \times 1$ array
- Pixel values from **0** (black) to **255** (white)

Image representations

3) Color mapped (indexed) image

- An $m \times n \times 1$ array
- Each value corresponds to a specific color, as predefined by a color map (e.g. 256 colors)
- Used for images with limited colors (e.g. cartoons)

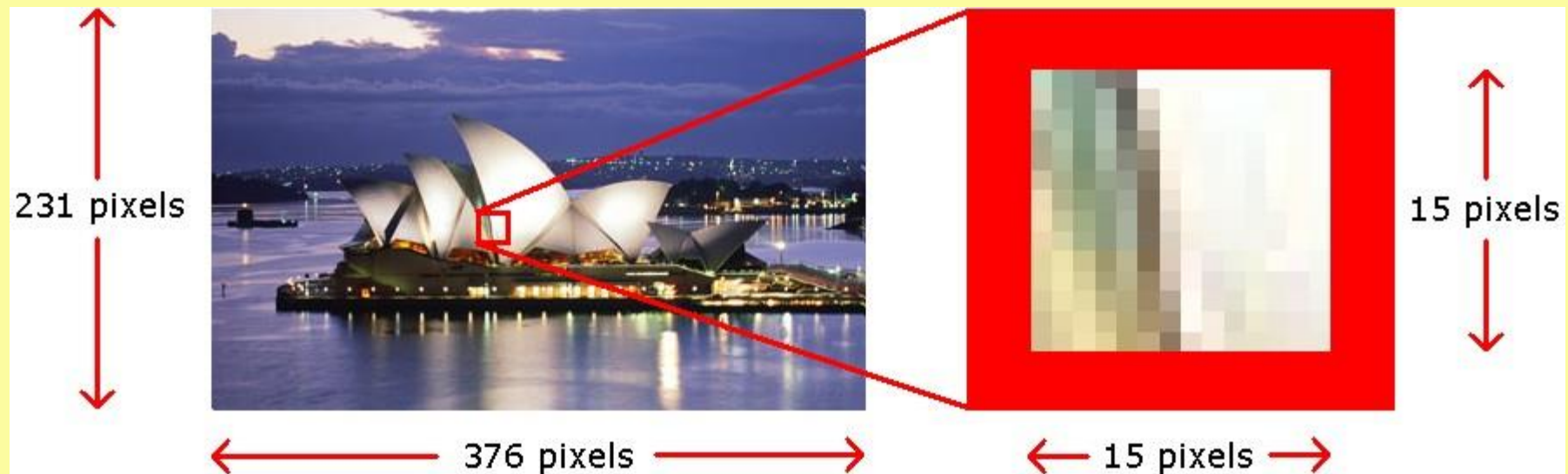


Ref: D. Smith,
Engineering
Computation with
Matlab (2008)

Images

- In all 3 representations, an image is represented as an **m x n x color** (3-D) matrix
 - **m** is the number of rows of pixels
 - **n** is the number of columns of pixels
 - **color** is the number of 2-D matrices
 - **color** is 3 for true color images since there are 3 primary colors (red, green, blue)
 - **color** is 1 for gray scale and color-mapped images

Practical Example



- How many bytes do we need to store this image as a true color image?

Practical Example

- Answer = 231×376 (pixels) $\times 3$ (primary colors) $\times 8$ (bits per color value)
= 2,084,544 bits
= 260,568 bytes (8 bits per byte)
= 254 kilobytes (KB) ($2^{10} = 1024$ bytes per KB)

Compression

- Images can be stored in compressed format, e.g. JPEG, GIF, TIFF, etc.
- JPEG is best for photos with a lot of fine colors
 - True color image
 - High compression, but loss of image quality
 - The previous example would reduce from 254 KB to 16.4 KB (94% reduction) for only a slight visual degradation

Compression

- GIF is better for images with few colors
 - Color mapped (indexed) image

Reading and Writing Images

```
% Read an image, specifying the format
pic = imread('opera1.jpg');

% Determine dimensions of image
[rows, cols, colors] = size(pic);

% Display image in normal sized window
% (image stretches or shrinks as needed)
image(pic);

% Display image in actual size
% (without stretching or shrinking)
imshow(pic);

% Write image to file, specifying the format
imwrite(pic, 'opera1.jpg', 'jpg');
```

Image Representations

- We can convert from one image representation to another:

rgb2gray()

rgb2ind()

ind2rgb()

gray2ind()

ind2gray()

*The **gray2rgb** function does not exist, but you will not be required to convert from gray scale image to true color image in this course

Image Representations

```
% Read an image, specifying the format
pic = imread('operal.jpg');

% Convert from true color image to gray scale image
gray_pic = rgb2gray(pic);

% Display image in actual size
% (without stretching or shrinking)
imshow(gray_pic);
```

Image Representations

```
% Control what colors will be shown;  
% color values less than low will be shown as black;  
% color values greater than high will be shown as white;  
% color values in between low and high will be shown as  
% shades of gray  
low = 0;  
high = 150;  
imshow(gray_pic, [low, high]);
```

- Top image is **gray_pic**
- Bottom image is **gray_pic**, but with all color values above 150 shown as white



Concatenating & Slicing Images

- Since images are represented as matrices, performing actions on matrices corresponds to modifying the image
- Combining images side-by-side is done by concatenating matrices
- Cropping (cutting) images is done by extracting rows and columns of matrices (slicing matrices)

Practical Example

- Start off with original image (shown on the left), and combine the left third and right third to form the new image (shown on the right)



Practical Example

```
% Read the picture from file and show what it looks like
pic = imread('opera1.jpg', 'jpg');
imshow(pic);

% How big is the image?
[rows, cols, colors] = size(pic);

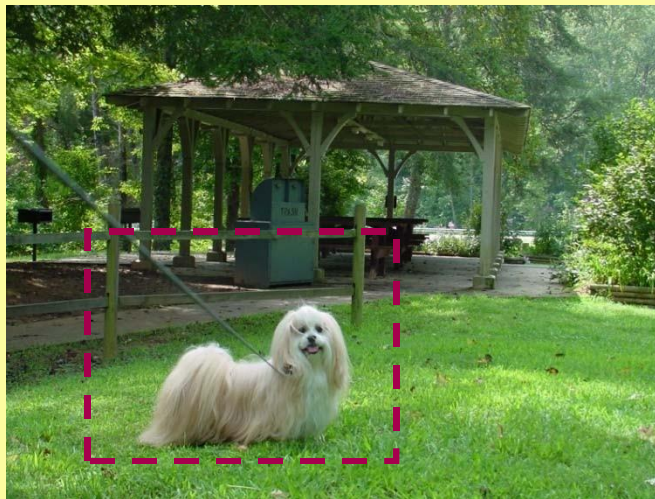
% Which column numbers divide the image into thirds?
end1 = round(cols/3);
end2 = round(2*cols/3);

% Get the left third and right third of the image
left_part = pic(:, 1:end1, :);
right_part = pic(:, end2+1:cols, :);

% Combine these 2 parts and show what it looks like
new_pic = [left_part right_part];
imshow(new_pic);
```

Another Practical Example

- Start off with original image (shown on the left), and create the new image showing only a part of the original image (shown on the right)



Ref: D. Smith,
Engineering
Computation with
Matlab (2008)



Another Practical Example

```
% Read image from file
pic = imread('dog.jpg', 'jpg');

% Specify where we want to crop
minX = 50;
maxX = 190;
minY = 120;
maxY = 230;

% Take only the columns and rows between
% the values above, note y before x
pic_cropped = pic(minY:maxY, minX:maxX, :);

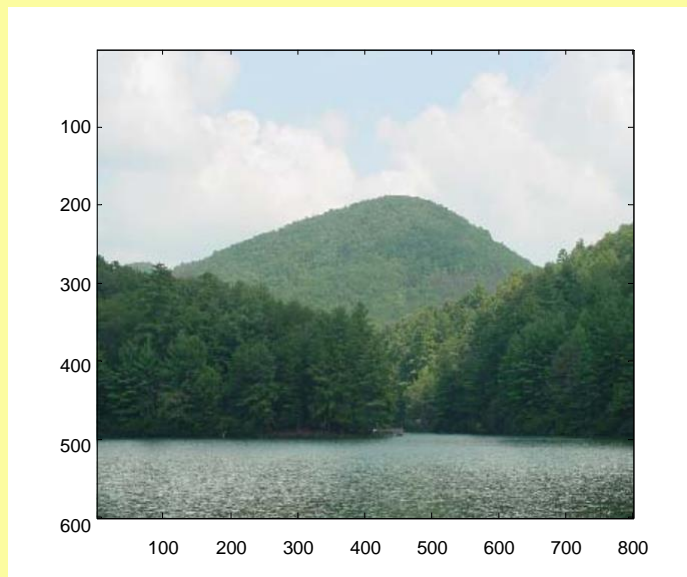
% Show the new cropped image
imshow(pic_cropped);
```

Shrinking & Expanding Images

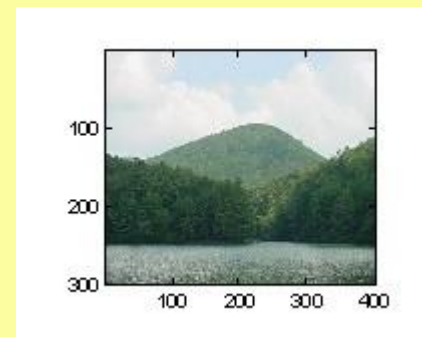
- We can shrink an image (make it smaller) by only sampling some rows and columns at regular intervals
- Similarly, we can expand an image (make it bigger) by repeating pixel values many times – the image may appear blurred because of this

Practical Example

- Start off with original image (shown on the left), and shrink it in half on both axes (shown on the right)



Ref: D. Smith,
Engineering
Computation with
Matlab (2008)



Practical Example

```
% Read image from file
pic = imread('lake.jpg', 'jpg');

% Get every 2nd row and column;
% note that we are therefore losing some data,
% so shrinking images will deteriorate image quality
pic_small = pic(1:2:end, 1:2:end, :);

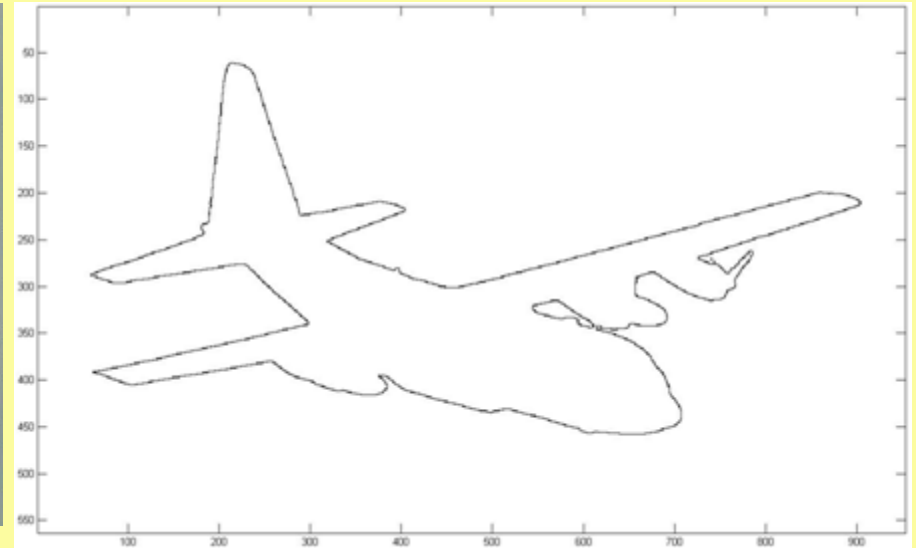
% Show the new smaller image
imshow(pic_small);
```

Dealing with Images

- There are many other tasks we can perform on images
- Humans are very good at understanding images
- Computers are very good at performing simple and repetitive tasks with images
 - Complex problems need to be broken down into many simpler steps

Edge Detection

- Edge-detection involves finding the outline of images



Ref: D. Smith, Engineering Computation with Matlab (2008)

Edge Detection

- The idea is to detect big changes in pixel values that are next to each other
 - Define a threshold (minimum amount) for what is considered a big change in pixel values
 - Neighboring pixels with a large difference in their pixel values has an edge between them
- There are many specific edge-detection algorithms, all using this general idea

Edge Detection

- The **edge()** function takes a gray scale image and detects its images
 - Edges set to white; everything else is black

```
% Read image from file
pic = imread('c-130.jpg', 'jpg');

% Convert to shades of gray
gray_pic = rgb2gray(pic);

% Generate an image showing just the edges using the
% Sobel method; use help for a list of other methods
pic_edge = edge(gray_pic, 'sobel');
```

Edge Detection



Sobel



Canny

Looping through a 2-D matrix

- We have already seen how to loop through a 2-D matrix – **you must know it!**
- The following slide shows you an example we used previously

Looping through a 2-D matrix (1)

```
% Create a matrix (3 rows, 4 columns)
matrix = [1 2 3 4; 5 6 7 8; 9 10 11 12];

% Determine dimensions of matrix (lecture 4-2, slide 70)
[rows, cols] = size(matrix);

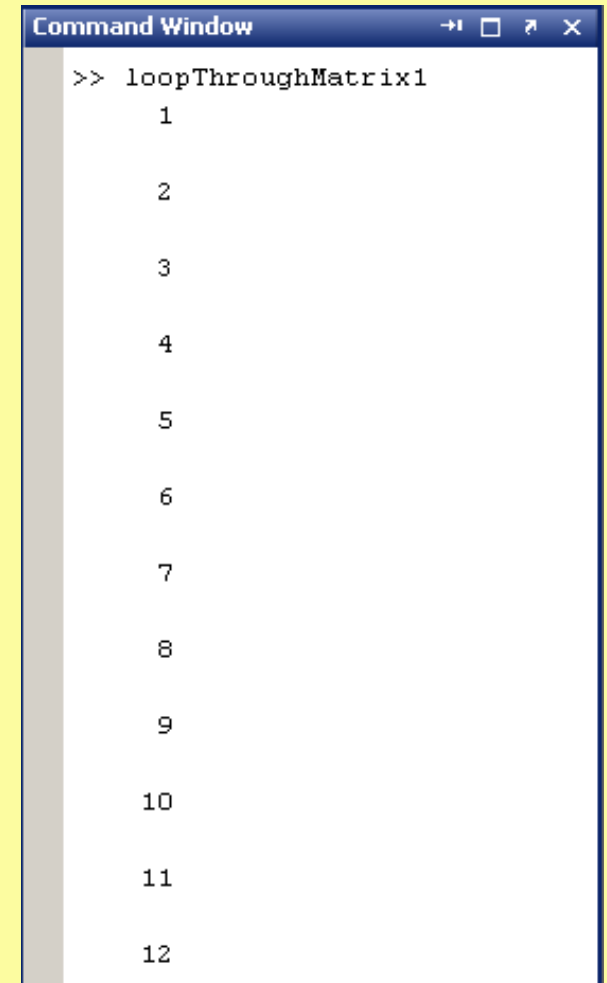
% Go through each row (r is row number)
% (compare to lecture 4-2, slide 59)
for r = 1:rows

    % Go through each column (c is column number)
    for c = 1:cols

        % Print element in matrix at row r, column c
        % (compare to lecture 4-2, slide 55)
        disp(matrix(r,c));
    end
end
```

Looping through a 2-D matrix (1)

- When the previous slide is run, this is the result:

A screenshot of a MATLAB Command Window. The title bar reads "Command Window" with standard window controls. The command prompt ">>" is followed by the function call "loopThroughMatrix1". Below this, the numbers 1 through 12 are listed vertically, representing the output of the loop.

```
>> loopThroughMatrix1
1
2
3
4
5
6
7
8
9
10
11
12
```

Looping through a 2-D matrix

- The following slide does the same thing as before, except that elements on the same row are printed beside each other instead of above each other:

Looping through a 2-D matrix (2)

```
% Create a matrix (3 rows, 4 columns)
numbers = [1 2 3 4; 5 6 7 8; 9 10 11 12];

% Determine dimensions of matrix (lecture 4-2, slide 70)
[rows, cols] = size(numbers);

% Go through each row (r is row number)
% (compare to lecture 4-2, slide 59)
for r = 1:rows

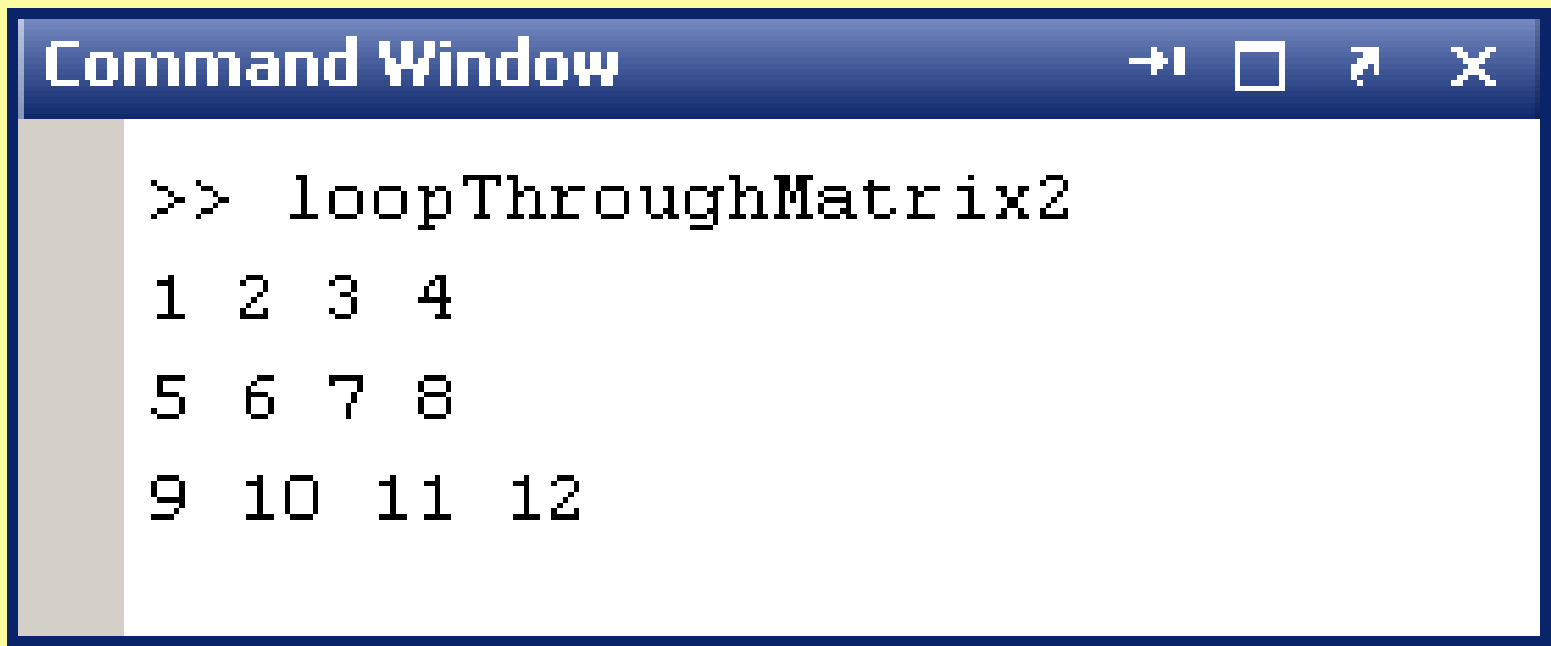
    % Go through each column (c is column number)
    for c = 1:cols

        % Print the value of an integer, which is stored in the
        % matrix at row r, column c, followed by a space
        % (compare to the slides on text output in lecture 8-2)
        fprintf('%d ', numbers(r,c));
    end

    % Print the newline character, so that next time we print on the
    % next line below
    fprintf('\n');
end
```

Looping through a 2-D matrix (2)

- When the previous slide is run, this is the result:



A screenshot of a MATLAB Command Window. The window has a dark blue title bar with the text "Command Window" and standard window control icons (minimize, maximize, close). The main area is white and contains the following text:

```
>> loopThroughMatrix2  
1 2 3 4  
5 6 7 8  
9 10 11 12
```

Looping through an image

- Similarly, you can loop through an image (since an image is just a 3-D matrix) to visit each pixel
 - E.g. Find maximum pixel value
 - You should be able to do this, and any other similar exercise, using what you learnt in previous lectures
 - More on this in this week's lab, and in the next few slides...

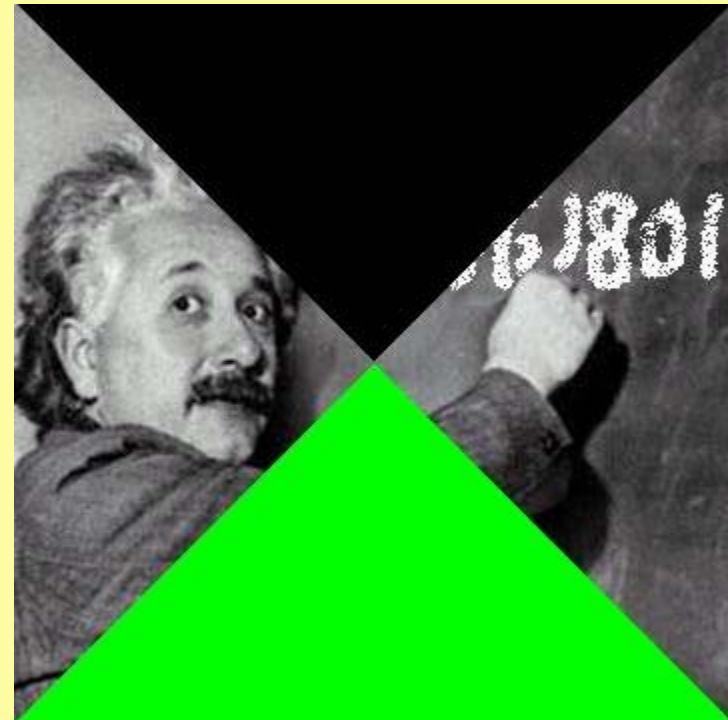
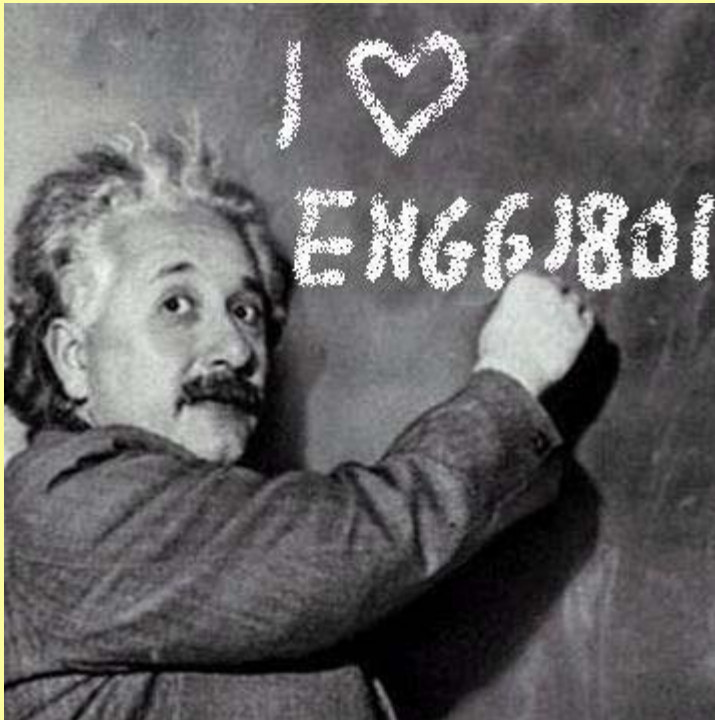
Practical Example

Write a Matlab program to:

- a) Read a file containing a true color image; you may assume that the image is square
- b) Each pixel that is closer to the top of the image than any other edge should turn black
- c) Each pixel that is closer to the bottom of the image than any other edge should turn green
- d) Display the original and resultant images

Practical Example

- For example, if the original image is on the left, the resultant image is on the right



Practical Example

```
% Read a photo-quality image from file
image = imread('einstein_blackboard.jpg');

% Show the original image
figure(1);
imshow(image);

% Determine dimensions of image (3-D matrix)
[rows, cols, colors] = size(image);

% Go thru each element in the image (3-D matrix)
for r = 1:rows
    for c = 1:cols
        for thisColor = 1:colors

            % Continued on next slide...
```

Practical Example

```
% If we are closer to the top edge,  
% turn the pixel black  
if r < c && r+c < rows+1  
    image(r,c,thisColor) = 0;  
  
% If closer to bottom edge, turn pixel green  
elseif r > c && r+c > rows+1  
  
    % Second of RGB primary colors is green  
    GREEN = 2  
  
    % Continued on next slide...
```

Practical Example

```
% If we are in the green sheet (the green  
% green 'slice' of the 3-D matrix), then  
% put the full color value in  
if thisColor == GREEN  
    FULL_COLOR = 255;  
    image(r,c,thisColor) = FULL_COLOR;  
  
% Otherwise, we're in another color  
% sheet (another 'slice' of the 3-D  
% matrix), so put no color value in  
else  
    image(r,c,thisColor) = 0;  
end  
end  
end  
end  
end
```


Practical Example

```
% Show the resultant image  
figure(2);  
imshow(image);
```

To Know

- Images
 - Image representations
 - Concatenating images
 - Slicing images
 - Shrinking & expanding images
 - Edge detection

Movies

- Creating movies
 - How to creating moving pictures with matrices

Images vs color / heat maps

- All of these images were designed to be saved into files
- Now, we will deal with color / heat maps
 - These are also images, but they are not designed to be saved into files
 - They are mainly used to help us create movies

Creating color / heat maps

- **imagesc(m)** creates a color / heat map, where different colors are used to represent each number in the matrix
 - We can also use some of the commands that we saw in previous lectures, e.g.
 - **figure()** – to choose in which window the color / heat map will be shown
 - **pause** – wait until user presses something
 - **colormap** – to control what colors will be shown
 - **colorbar** – to show what each color represents

Practical Example

- Given a matrix saved in the file **matrix.txt**,
 - Visualize the values of the matrix as a color / heat map

Practical Example

- The file **matrix.txt** looks like this:

```
8,2,7,7,4,3,8,8,4,1
9,10,0,0,4,7,3,3,8,1
1,10,8,3,8,7,5,8,6,5
9,5,9,0,8,2,7,2,5,8
6,8,7,1,2,1,9,9,9,9
1,1,8,8,5,5,10,3,3,1
3,4,7,7,4,10,5,2,8,6
5,9,4,3,6,3,1,3,8,5
10,8,7,10,7,6,1,6,4,0
10,10,2,0,8,2,3,5,6,3
```

Creating color / heat maps

```
% Read the matrix from file
myMatrix = csvread('matrix.txt');

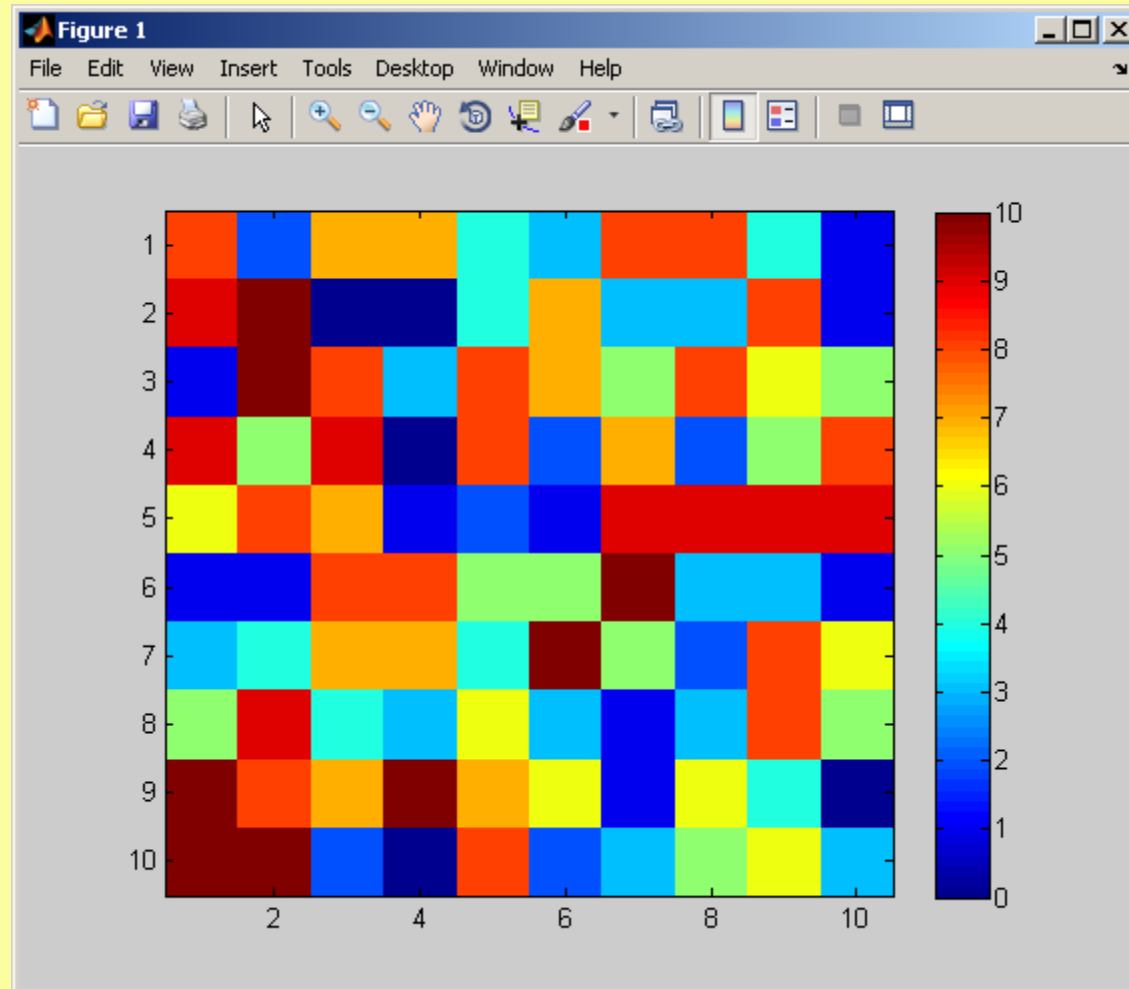
% We're going to plot in figure 1
figure(1);

% Plot the matrix as a color/heat map
imagesc(myMatrix);

% Show the legend which shows what each color represents
colorbar;

% Continued on next few slides...
```


Creating color / heat maps



ENGG1801 Engineering Computing

Creating a movie

- **getframe()** returns a frame
 - A frame is just 1 image in a movie
 - A movie consists of many frames
 - Use **getframe()** after you have already called **imagesc()** to create an image
 - That image will be stored in a frame

Creating a movie

- To create the actual movie (**.avi** file), you need 3 more parts
- **1)** This part goes right at the start, before you make any images:

```
% Prepare to create the movie (.avi) file  
video = VideoWriter('nameOfMovie.avi');  
open(video);
```

Creating a movie

- **2)** This code appears each time you create a frame:

```
% Create the next image  
imagesc(matrix);
```

```
% Get a frame (same as the image shown above),  
% and store into the video  
frame = getframe();  
writeVideo(video, frame);
```

Creating a movie

- **3)** This part goes right at the end, after you have made all your images:

```
% Close the file so that it saves properly  
close(video);
```

Creating a movie

- When you have these 3 parts, then an **.avi** (movie) file will be created
 - The movie is saved in the same folder as where your Matlab program is
 - You can use many standard programs to watch the movie (e.g. VLC Media Player)
 - So someone can watch the movie, even if they don't have Matlab!

Practical Example

- Create a random movie, where on each frame, 1 pixel (dot on the screen) is selected at random and changes color

Creating a random movie

```
% Prepare to create the movie (.avi) file
video = VideoWriter('random.avi');
open(video);

% Create a 32x32 matrix; all values are 0
matrix = zeros(32);

% For each of the 500 frames in the movie
for step = 1:500

    % Let x and y each be a random integer between 1 and 32
    x = randi([1,32]);
    y = randi([1,32]);

    % Change the value stored at the random point (x,y);
    % note that y is before x in (y,x) because y is row number
    matrix(y, x) = matrix(y, x) + 1;

    % Continued on the next slide...
```


Creating a random movie

```
% ... continued from the previous slide

% Create the next image
imagesc(matrix);

% Show the legend which shows what each color represents
colorbar;

% Get a frame (same as the image shown above), and store into the video
frame = getframe();
writeVideo(video, frame);
end

% Close the file so that it saves properly
close(video);
```

Interrupting the program

- If you interrupt the program and close the figure (window) while the program is running, you will get an error message

Command Window

```
Struct contents reference from a non-struct array object.  
  
Error in alternateGetframe  
  
Error in getframe (line 111)  
x = alternateGetframe(parentFig, offsetRect, includeDecorations);  
  
Error in randomMovie (line 45)  
    frame = getframe();
```

fx >>

Practical Example

- Create a movie, with a ball bouncing off the edge of the edges of the image

Creating a bouncing ball movie

```
% Create a movie with a bouncing ball
%
% bouncingBallMovie()
function bouncingBallMovie()

    % Prepare to create the movie (.avi) file
    video = VideoWriter('random.avi');
    open(video);

    % Create a 32x32 matrix; all values are 0
    matrix = zeros(32,32);

    % Find out how many rows and columns are in the matrix;
    % note y before x
    [maxY, maxX] = size(matrix);

    % Note the minimum possible x and y values
    minX = 1;
    minY = 1;

    % Initial position of active pixel (ball)
    x = 1;
    y = 1;                                     % Continued on next slide...
```

Creating a bouncing ball movie

```
% Initial rate of change of ball along x and y axes
dx = 1;
dy = 2;

% Current color for all pixels in the matrix
BG_COLOR = 0;

% Color for active pixel in matrix (where the ball is)
BALL_COLOR = 1;

% Choose the colors to be used in the movie
colormap hot;

% For each of the 500 frames in the movie
for step = 1:500

    % Put the ball at location (x,y); note y before x
    matrix(y, x) = BALL_COLOR;

    % Get the next frame (image)
    imagesc(matrix);

    % Change color of where the ball is back to background color
    matrix(y, x) = BG_COLOR;

    % Continued on next slide.
```

Creating a bouncing ball movie

```
% Get a frame (same as the image shown above), and store into the video
frame = getframe();
writeVideo(video, frame);

% If the next change in coordinates will put us outside of the range,
then
% negate the appropriate dx or dy (change directions = bounce)
if (x + dx > maxX && dx > 0) || (x + dx < minX && dx < 0)
    dx = -1 * dx;
end
if (y + dy > maxY && dy > 0) || (y + dy < minY && dy < 0)
    dy = -1 * dy;
end

% Move the ball
x = x + dx;
y = y + dy;
end

% Close the file so that it saves properly
close(video);
end
```

Creating a bouncing ball movie

- In this example:
 - (x,y) is the location of the bouncing ball
 - dx and dy are the rate of change of the ball along the x and y axes

Practical Example

- **This Q was used recently in Lab Exam 3**
- Modify the file `bouncingBallMovie`.
Change the program so that:
 - The ball is initially moving diagonally, bottom-right
 - In addition to the ball, there is also a goalkeeper of 5 horizontal pixels in a row; each pixel of the goalkeeper is always on the screen

Practical Example

- The goalkeeper is always on the 30th row of the image, directly below the ball, with the center of the goalkeeper directly below the ball wherever possible (if not possible, then as close as possible)
- The ball can bounce off the goalkeeper just like how it bounces off the edge of the screen

Your program should continue to do everything else the same as in the bouncing ball movie example

Practical Example

- The code will be available for download at the end of this week
 - It is one of the lab exercises; you should attempt to do it yourself

To Know

- Creating movies
 - Create a matrix with numbers which will be represented with different colors
 - Creating color / heat maps with **imagesc()**
 - Creating frames with **getframe()**
 - Store these frames in an array
 - Give this array to the function **movie2avi()**