Image Processing (and Intro to MATLAB)

Computer Science Intensive Program 2017

Iraklis Tsekourakis
Lieb 213
itsekour@stevens.edu
Agenda

1. Overview of Imaging Fields
2. Introduction to MATLAB Programming
3. Introduction to Image Processing (IP)
4. Some IP Algorithms
5. Play time!
6. Custom processing implementation
Why work with Images?

- Many interesting applications
  - Photoshop
  - Computer Vision
  - Computer Graphics
Graphics vs Vision

• My research is in Computer Vision

• Graphics vs Vision
  – Inverse Processes?
Computer Graphics

- Build a world by specifying
  - Geometry
  - Textures
  - Materials
  - Lights
  - Properties

Shape

Lights

Viewpoint
Computer Vision

- Shape
- Lights
- Viewpoint
Eyes for the Blind

- Cameras mounted on glasses
- Electrically stimulate visual pathway
- Resolution still limited

- Boston Retinal Implant Project
  - Harvard,
  - Berkley
  - MIT
The DARPA Grand Challenge

• 2004: goal drive 150 miles in the Mojave desert autonomously
  – Longest any participant traveled: 7.4 miles
• 2005: 5 teams completed race
  – Won by Stanley from Stanford (VW Touareg)
Graphics + Vision

- Fluid simulation interacting with real video
  - Dangerous/Impossible to use real fluid
  - Modeling the scene by hand is cumbersome
  - Interaction with *moving objects* in the scene (one-way)
Graphics + Vision
In the Last Few Years

- Google Maps, Street View and Google Earth
- Wii, XBox 360, Playstation 3, Kinect
Graphics + Vision
In the Last Few Years

- Snapchat
- Autonomous Vehicles
- Pokemon GO?
Image Processing

• All the areas have something in common, they involve image processing
  – Extracting information from images or frames of videos
  – Often involve “cleaning” up images
  – May involve generation of new images

• And obviously there’s a lot of interest in just being able to manipulate images for aesthetic fun
  – Greyscale
  – Sepia
  – Flat
  – Crop
  – Change colors
Intro to Matlab
What is MatLab

• Matlab (Matrix Laboratory) is a popular programming language for fields like
  – Research (general)
  – Engineering
• Built on C with a Java interface
  – But interpreted so slow 😞
• Why Matlab?
  – Lots of built in capabilities (the basic installation is ~4 gigs)
  – Tons of built-in functions and optional add-on packages
  – Cross-platform (different installer, but code pretty much works in each)
  – Quick and easy to use/learn
  – Built-in stuff for matrix operations/manipulation
Matlab Interface

• Command Line
• Workspace (variable list)
• Command History
• Directory Browser

• Can type *commands* in the command line
  – *Scripts* to run
  – Basic math operations
  – Run functions on existing variables (or make new ones)
Matlab Scripts

• Under Home ➔ New/Open you can create/open scripts (small programs) and/or functions
• Editor window also has debugging stuff
Hello World Program

disp('Hello World');

String argument
(Matlab strings are in single quotes)

disp function
Variables

• Variables in Matlab do not need to be declared as a type
  – It is inferred, and can change based on assignment
  – HOWEVER, often you can only do operations with certain data types, so you may need to do casting
    • `double(X)`
    • `uint8(X)`

• Variable names have the following rules:
  – Must start with a letter
  – All other characters must be letter, number, or the underscore `_` character
  – Variables are case-sensitive
Operators

- Matlab uses the standard arithmetic operators: + - * /
  - There is no increment/decrement operator 😞
- It also uses ^ for power
- Matlab uses % for comment lines, so to do modulus, we use a modulus function
  
  ```matlab
  x = mod(5, 3);
  ```
- Since it specializes in matrix operations, you can do standard matrix math (both matrix/vector and scalar)
- But can also do element-wise operations
  - Apply operator to each element of the two matrices/vectors
  - These start with a “.”
    ```matlab
    .* ./ .^  
    ```
Strings

- We define strings by putting characters between **single quotes**
  - Strings are though of as arrays/vectors of characters
    
    ```
    Myname = 'Hercules';
    ```

- We can **concatenate** string (or any vectors) by putting them next to each other within square brackets
  
  ```
  Myname = ['Iraklis' 'Tsekourakis'];
  ```
Example: Operators

```
x = 10.5;
y = 20.5;

// adding numbers
disp('Adding');
disp(['x + y = ' num2str(x + y)]);

disp function displays strings so to display numbers we must first convert them to strings. We can do that with the num2str function

// exponents
disp('Exponents');
disp(['x^y = ' num2str(x^y)]);

Concatenate the two strings to display one long string
```
Arrays

- In Matlab we refer to these as **vectors** (1D) and **matrices** (2D)
  - Can also be higher dimensional (we’ll see a lot of 3D matrices)
- Create a matrix/vector by putting elements in square brackets separated by commas or spaces
  
  ```matlab
  myVector = [4 3 8 0];
  ```
- To go to a new row, we use a semicolon
  
  ```matlab
  myMatrix = [4 3 5; 0 2 3];  % a 2x3 matrix
  ```
Arrays

- Can use built-in Matlab functions to make vectors/matrices
  - `myVector = zeros(1,5);`  % 1x5 matrix of zeros
  - `myMatrix = rand(2,4);`  % a 2x4 matrix of random numbers, each in the range of [0,1)

- We can create a vector using the colon : operator
  - `x = 1:1:10`  % from 1 to 10 in increments of 1
  - `x = 1:10`  % shortcut
  - `x = 1:3:50;`  % from 1 to 50 in increments of 3
Arrays

myMatrix = [4 3 5; 0 2 3];
myVector = [4 3 8 0];

• We can access locations in vectors/matrices by putting the location (index) in parenthesis after the variable name
  – NOTE: In Matlab locations start at 1 😞

  disp(myVector(3));
  disp(myVector(0));  %error!
  disp(myMatrix(2,1));

• We can use the function size to get the number of elements for a given dimension

  size(myMatrix,1);  %number of rows
  size(myMatrix,2);  %number of columns
More Matrix Indexing

myVector = [4 3 8 0];

myMatrix = [4 3 5; 0 2 3];

- We can specify several locations at once by specifying a vector for the index
  myMatrix([1 2], [1 3]);
  myVector(1:2);

- We can also use a special end keyword to say “the last”
  myMatrix(1, 2:end)

- Or use the colon operator by itself to mean “all”
  myMatrix(2,:);
Control Statements

• Matlab has the same standard control statements as most modern programming language
  – if ... else
  – switch
  – while
  – for

• NOTE: Control statements end with the **end** keyword
  – Doesn’t use brackets or indentation like other languages
num = -3;

if ( num < 0 )
   disp(['The number ' num2str(num) ' is negative']);
elseif (num == 0)
   disp(['The number ' num2str(num) ' is zero']);
else
   disp(['The number ' num2str(num) ' is positive or zero']);
end
For Loops

• These are actually more like for each loops in other languages
  – Does the body of the loop for each value in a vector

```matlab
for i=1:10
    disp( ['Current number: ' num2str(i)] );
end
```
Finding the Max

```matlab
lst = rand(1,10)*20;

mxm = lst(1);
// ?????????
disp(['The maximum was: ' num2str(mxm)]);

• Remember Matlab has lots of built-in functions, try using the search/help menu
  m = max(lst);
  [m, i] = max(lst);

myVector = [4 3 8 0];
```
Image Processing
Digital Images

- A digital image differs from a photograph in that the values are all discrete
  - A digital image can be considered as a large array of discrete dots. Each of the dots has a brightness associated with it. These dots are called *picture elements*, or more simply *pixels*.

- Similar to the concept of Compact Disks vs. Records:
  - Technically speaking, records are better because they contain infinite analog data.
  - CDs are digital versions that are *sampled* frequently enough that the difference is impossible for the human ear to detect
  - Sampling rate 48kHz = 2x limit of human hearing (20k)

- Likewise analog photography has more information but this becomes negligible as the sampling (pixels) becomes more frequent
Digital Images

- Most digital images contain either 1 (gray) or 3 color channels (RGB) each of whose values range according to the number of bits used (ex. 8bit, 16bit, etc.)

- Today, we will look at both color and grayscale images with 8 bits ($2^8=256$ values, 0-255) per channel
  - These values are unsigned integers and therefore the data type is uint8
### Image Data

All values are within 0 - 255

<table>
<thead>
<tr>
<th>48</th>
<th>219</th>
<th>168</th>
<th>145</th>
<th>244</th>
<th>188</th>
<th>120</th>
<th>58</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>218</td>
<td>87</td>
<td>94</td>
<td>133</td>
<td>35</td>
<td>17</td>
<td>148</td>
</tr>
<tr>
<td>174</td>
<td>151</td>
<td>74</td>
<td>179</td>
<td>224</td>
<td>3</td>
<td>252</td>
<td>194</td>
</tr>
<tr>
<td>77</td>
<td>127</td>
<td>87</td>
<td>139</td>
<td>44</td>
<td>228</td>
<td>149</td>
<td>135</td>
</tr>
<tr>
<td>138</td>
<td>229</td>
<td>136</td>
<td><strong>113</strong></td>
<td>250</td>
<td>51</td>
<td>108</td>
<td>163</td>
</tr>
<tr>
<td>38</td>
<td>210</td>
<td>185</td>
<td>177</td>
<td>69</td>
<td>76</td>
<td>131</td>
<td>53</td>
</tr>
<tr>
<td>178</td>
<td>164</td>
<td>79</td>
<td>158</td>
<td>64</td>
<td>169</td>
<td>85</td>
<td>97</td>
</tr>
<tr>
<td>96</td>
<td>209</td>
<td>214</td>
<td>203</td>
<td>223</td>
<td>73</td>
<td>110</td>
<td>200</td>
</tr>
</tbody>
</table>

3 × 5 neighbourhood

Current pixel
MATLAB Image Functions

• To load an image into a matrix we can use
  
  \[ X = \text{imread}(\text{filename}); \]

• To view an image (in the current figure) we can use
  
  \[ \text{imshow}(X); \]
  
  – To display properly, \( X \) must be of uint8 type, so you can do
    
  \[ \text{imshow}(	ext{uint8}(X)); \]

• To open (or make current) a figure we can type
  
  \[ \text{figure}(\text{figurenum}); \]
  
  – Most recent figure is use for next drawing

• We can also save an image with
  
  \[ \text{imwrite}(	ext{matrix}, \text{filename}, \text{format}) \]
More Useful MATLAB Commands

• Good to remove old variables first
  clear all;
• May also want to close old figures
  close all;
clear all;
close all;

X = imread(filename);

%Convert data to double type so we can do non-integer math
X = double(X);

%%%%%%%%%% DO IP STUFF%%%%

imshow(uint8(X));
Cropping in MATLAB

- `im = imread('lena.png');`
- `figure(1); imshow(im);`
- Let’s look at the first 100 rows (ordering is top to bottom) and the last 100 columns (ordering is left to right)
- `im2 = im(1:100, size(im, 2) - 100:size(im, 2), :)`
- `figure(2); imshow(im2);`
Grayscale

• To turn a color (RGB) image into a grayscale image we must compute a weighted average of the channels.

• The standard formula is

\[
\text{Gray} = 0.2989\times R + 0.5870\times G + 0.1140\times B
\]
Grayscale in MATLAB

• We could do this by processing each pixel

```matlab
im2 = zeros(size(im,1),size(im,2));
for r=1:size(im,1)
    for c = 1:size(im,2)
        im2(r,c) = 0.2989*im(r,c,1) + 0.5870*im(r,c,2) + 0.1140*im(r,c,3);
    end
end
```
Grayscale in MATLAB

• Or by some scalar-matrix multiplication

\[ \text{im3} = 0.2989 \times \text{im}(:,:,1) + 0.5870 \times \text{im}(:,:,2) + 0.1140 \times \text{im}(:,:,3); \]

• This is much faster!
  – Can take advantage of multiple processors, threads, GPU, etc..
Binary Images

- Aka as black-and-white images
- We can make one by setting pixels with intensity $< 128$ to black and the rest to white
thresh = 127;
im2 = 0.2989*im(:,:,1) + 0.5870*im(:,:,2) + 0.1140*im(:,:,3);
for r=1:size(im2,1)
    for c=1:size(im2,2)
        if im2(r,c) < thresh
            im3(r,c) = 0;
        else
            im3(r,c) = 255;
        end
    end
end
Again if we can do stuff in matrices, that tends to be faster

```matlab
im4 = zeros(size(im2));
im4(im2>thresh) = 255;
```
Additional Algorithms

1. Switch color channels
2. Brightness adjustment
3. Negative images
4. Noising images
5. Blending images
6. Smoothing images (de-noising)
7. Sharpening images
Switch Color Channels

Switching the red and blue channels
Brightness Adjustment

• How can we make the input image half as bright?
  – The brightness (intensity) is the value of the pixel
Negative Image

• Here we want to reverse the values:
  – 0→255
  – 255→0

• For each pixel $r→s$
  \[ s = 255 - r \]
Blending Images

• A lot of our algorithms involve combining data/images
  – Adding noise
  – Blending images
  – Sharpening images
• We call this blending and can control it with a blending factor

• Given images $X$ and $Y$, and a blending factor $0 \leq \alpha \leq 1$ we can generate a new image as

$$Z = \alpha X + (1 - \alpha) Y$$
Noising Images

- We can add *random noise* to our images
- One way:
  - Create a noise image and blend it with our original
Blending Two Images

- Read in two images and make a new image by blending them.
PART II

Area Processing
Area Processing

- Area processing takes into account **surrounding pixels** to determine a new pixel value
- This is commonly used for filtering, such as:
  - Smoothing
  - Sharpening
The Mean Filter

Original

\[
\begin{array}{ccc}
1 & 1 & 1 \\
\frac{1}{9} \times & 1 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

15x15
Smoothing Images

• To smooth an image choose the radius (neighborhood size) and replace each pixels value with the mean or median of the pixels in its neighborhood
The Median Filter

• The problem with averaging (mean) filters is the blurring of edges and other sharp details.
• A median filter replaces a pixel’s value with the median of the surrounding area
  – Original 3x3 Mean 3x3 Median
Sharpening Images

• Overly smoothed images can look a bit bland
• To add detail you can try to sharpen an image
• One way to sharpen an image is to create a smoothed image and then subtract it from the original
  – This is the “grain/grit” of the original image
• Then add some amount of this to the original image
  – Again, blend the grit with the original
Sharpening Images
Edge detection

- Edge detection is a fundamental tool in image processing and computer vision.
- The points at which image brightness changes sharply are organized in a set of curved lines, called edges.
- The Sobel filter is an operator used in edge detection that creates an image which emphasizes edges and transitions.
The Sobel Filter

- The Sobel operator performs a 2D spatial gradient (change) measurement on an image and so emphasizes regions of high spatial change that correspond to edges.

- The basic idea is:
  - For a given pixel, find out how much it changed in the x and y directions (independently)

\[
\begin{array}{ccc}
-1 & 0 & +1 \\
-2 & 0 & +2 \\
-1 & 0 & +1 \\
\end{array}
\quad
\begin{array}{ccc}
+1 & +2 & +1 \\
0 & 0 & 0 \\
-1 & -2 & -1 \\
\end{array}
\]

G_x \quad \text{Current Pixel} \quad G_y
The Sobel Filter

- Then Combine the $x$ and $y$ gradients ($G_x$ and $G_y$ respectively) to form the overall spatial gradient
- Gradient $G$

$$|G| = \sqrt{G_x^2 + G_y^2}$$

- Approximation of $G$ (faster)

$$|G| = |G_x| + |G_y|$$
The Sobel Filter
Custom Algorithms

• Form groups of 1-3 people and spend 30 minutes trying to come up with your own image processing ideas
• Here’s some I thought of quickly to get you started:
  – Switch one area of the image with the other
  – Flip an image
  – Turn all pixels less than some value to some other color
  – Smooth just some area of an image
  – Rotate an image (advanced!)
    • Hint: Apply some 2D rotation matrix
    • But be careful of some pixel locations being mapped “out of bounds”
  – “Embed” one image in another