L23. Working with Image Files

\texttt{imread, imwrite, imshow, uint8, rgb2gray}

Pictures as Arrays

A black and white picture can be encoded as a 2D Array

Typical:

\[ 0 \leq A(i,j) \leq 255 \]

(values in between correspond to different levels of grayness.

Just a Bunch of Numbers

318-by-250

\begin{array}{cccccccc}
49 & 51 & 54 & 59 & 57 & 53 \\
60 & 47 & 71 & 72 & 70 & 76 \\
100 & 108 & 111 & 112 & 112 & 113 \\
157 & 145 & 168 & 167 & 165 & 144 \\
154 & 190 & 193 & 196 & 194 & 192 \\
199 & 208 & 211 & 212 & 211 & 216 \\
190 & 192 & 193 & 195 & 195 & 197 \\
174 & 169 & 165 & 163 & 162 & 161 \\
\end{array}

A Color Picture is 3 Arrays

Stack them in a 3D array.

Typical:

\[ 0 \leq A(i,j,1) \leq 255 \quad \text{(red)} \]
\[ 0 \leq A(i,j,2) \leq 255 \quad \text{(green)} \]
\[ 0 \leq A(i,j,3) \leq 255 \quad \text{(blue)} \]

Note 3\textsuperscript{rd} Subscript

Encoding Images

There are a number of file formats for images. Some common ones:

JPEG  
(Joint Photographic Experts Group)

GIF  
(Graphics Interchange Format)

Behind the scenes: compressing data
A Compression Idea

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
<td>21</td>
<td>24</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>24</td>
<td>28</td>
<td>32</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>30</td>
<td>35</td>
<td>40</td>
<td>45</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>18</td>
<td>24</td>
<td>30</td>
<td>36</td>
<td>42</td>
<td>48</td>
<td>54</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>21</td>
<td>28</td>
<td>35</td>
<td>42</td>
<td>49</td>
<td>56</td>
<td>63</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>40</td>
<td>48</td>
<td>56</td>
<td>64</td>
<td>72</td>
</tr>
<tr>
<td>9</td>
<td>18</td>
<td>27</td>
<td>36</td>
<td>45</td>
<td>54</td>
<td>63</td>
<td>72</td>
<td>81</td>
</tr>
</tbody>
</table>

Store the array (81 num’s) or the purple vectors (18 num’s)?

More Dramatic

Suppose A is a 1000-by 2000 times table.

Do I store A (2,000,000 numbers)

or

Do I store the two 1-dimensional multiplier arrays (3000 numbers) and “reconstruct” A

Images can be written as a sum of a relatively small number of times tables

1000-by-2000 picture might be well approximated by the sum of 100 times tables.

2,000,000 vs (100 x 3000)

Operations on Images

They amount to operations on 2D Arrays.

A good place to practice “array” thinking.

Two Problems

We have:

LawSchool.jpg

Want:

LawSchoolMirror.jpg

Problem 1
Problem 2

Want:

Solution Framework

Read LawSchool.jpg from memory and convert it into an array.

Manipulate the Array.

Convert the array to a jpg file and write it to memory.

imread

% Read image and convert to a 3D array...

A = imread('LawSchool.jpg');

The 3D Array

>> [m,n,p] = size(A)

m = 1458  \text{rows}

n = 2084  \text{columns}

p = 3  \text{layers}

The Layers

Left-Right Mirror Image

A = imread('LawSchool.jpg')

[m,n,p] = size(A);

for \( j = 1:n \)

\[
B(:,j,1) = A(:,n+1-j,1) \\
B(:,j,2) = A(:,n+1-j,2) \\
B(:,j,3) = A(:,n+1-j,3)
\]

end

imwrite(B,'LawSchoolMirror.jpg')
Equivalent

\[
\text{for } j=1:n \\
\quad B(:,j,1) = A(:,n+1-j,1) \\
\quad B(:,j,2) = A(:,n+1-j,2) \\
\quad B(:,j,3) = A(:,n+1-j,3) \\
\text{end}
\]

\[
B = A(:,n:-1:1,:);
\]

The Upside Down Image

\[
A = \text{imread(‘LawSchool.jpg’)} \\
[m,n,p] = \text{size}(A); \\
\text{for } i=1:m \\
\quad C(i,:,1) = A(m+1-i,:,1) \\
\quad C(i,:,2) = A(m+1-i,:,2) \\
\quad C(i,:,3) = A(m+1-i,:,3) \\
\text{end} \\
\text{imwrite}(C,'LawSchoolUpDown.jpg')
\]

New Problem

Color → Black and White

Have:

\[
A = \text{imread(‘LawSchool.jpg’)}; \\
bwA = \text{rgb2gray}(A); \\
\text{imwrite}(bwA,'LawSchoolBW.jpg')
\]

Want:

\[
A = \text{imread(‘LawSchool.jpg’)}; \\
bwA = \text{rgb2gray}(A); \\
\text{imwrite}(bwA,'LawSchoolBW.jpg')
\]
How Does the Conversion Work?

<table>
<thead>
<tr>
<th>r</th>
<th>g</th>
<th>b</th>
<th>gray</th>
</tr>
</thead>
<tbody>
<tr>
<td>167</td>
<td>219</td>
<td>241</td>
<td>206</td>
</tr>
<tr>
<td>66</td>
<td>35</td>
<td>15</td>
<td>42</td>
</tr>
<tr>
<td>95</td>
<td>14</td>
<td>20</td>
<td>39</td>
</tr>
<tr>
<td>163</td>
<td>212</td>
<td>242</td>
<td>201</td>
</tr>
<tr>
<td>182</td>
<td>228</td>
<td>215</td>
<td>213</td>
</tr>
<tr>
<td>225</td>
<td>244</td>
<td>222</td>
<td>236</td>
</tr>
<tr>
<td>136</td>
<td>199</td>
<td>240</td>
<td>185</td>
</tr>
</tbody>
</table>

It's a complicated mapping.

Why not take Average?

```matlab
bwA = uint8(zeros(m,n))
for i=1:m
    for j = 1:n
        bwA(i,j) = (A(i,j,1) + A(i,j,2) + A(i,j,3))/3;
    end
end
imwrite(bwA,'LawSchoolBW.jpg')
```

uint8 : unsigned 8-bit integer

Why not take Max?

```matlab
bwA = uint8(zeros(m,n))
for i=1:m
    for j = 1:n
        bwA(i,j) = max([A(i,j,1) A(i,j,2) A(i,j,3)]);
    end
end
imwrite(bwA,'LawSchoolBW.jpg')
```

uint8 : unsigned 8-bit integer

Max:

Matlab:
Problem: Produce a Negative

Idea

If matrix $A$ represents the image and

$$B(i,j) = 255 - A(i,j)$$

for all $i$ and $j$, then $B$ will represent the negative.