Previous Lecture:
- Characters and strings
- Image processing, type \texttt{uint8}
- Add frame

Today’s Lecture:
- More image processing
- color $\rightarrow$ grayscale
- Mirror, “Noise” filtering
- Edge finding

Announcements:
- Project 4 due Thursday after break
- Be sure to review—re-do—prelim 1 now so that you have a firm foundation

Example: color $\rightarrow$ black and white

Can “average” the three color values to get one gray value.

Averaging the RGB values to get a gray value

```plaintext
\text{Scalar operation}
```

Here are 2 ways to calculate the average. Are gray value matrices \( g \) and \( h \) the same given image data \( A \)?

```plaintext
\text{A: yes \quad B: no}
```

```plaintext
\text{for } r=1:n_r \quad \text{for } c=1:\text{nc} \\
\quad g(r,c) = \frac{A(r,c,1)}{3} + \frac{A(r,c,2)}{3} + \ldots + \frac{A(r,c,3)}{3}; \\
\quad h(r,c) = \ldots \\
\quad \quad ( A(r,c,1)+A(r,c,2)+A(r,c,3) )/3; \\
\text{end end}
```

```plaintext
M = 0.3A(:,:,1) + 0.59A(:,:,2) + 0.11A(:,:,3)
```

Lecture slides
Matlab has a built-in function to convert from color to grayscale, resulting in a 2-d array:

\[ B = \text{rgb2gray}(A) \]

Example: Mirror Image

1. Read `LawSchool.jpg` from memory and convert it into an array.
2. Manipulate the array.
3. Convert the array to a jpg file and write it to memory.

```matlab
% Read jpg image and convert to a 3D array A
A = imread('LawSchool.jpg');

% Write 3D array B to memory as a jpg image
imwrite(B,'LawSchoolMirror.jpg')
```

```matlab
% Store mirror image of A in array B
[nr,nc,np] = size(A);
for r= 1:nr
    for c= 1:nc
        for p= 1:np
            B(r,c,p)= A(r,nc-c+1,p);
        end
    end
end
```

Both fragments create a mirror image of A.

```matlab
% Make mirror image of A -- the whole thing
A= imread('LawSchool.jpg');
[nr,nc,np]= size(A);
B= zeros(nr,nc,np);
B= uint8(B); % Type for image color values
for r= 1:nr
    for c= 1:nc
        for p= 1:np
            B(r,c,p)= A(r,nc-c+1,p);
        end
    end
end
imshow(B) % Show 3-d array data as an image
imwrite(B,'LawSchoolMirror.jpg')
```
Vectorized code simplifies things...
Work with a whole column at a time

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6</td>
<td>1 2 3 4 5 6</td>
</tr>
</tbody>
</table>

Column c in B
is column nc-c+1 in A

Vectorized code to create a mirror image

\[
A = \text{imread('LawSchool.jpg')} \\
[nr, nc, np] = \text{size(A)}; \\
\text{for c= 1:nc} \\
B(:,c,1) = A(:,nc-c+1,1) \\
B(:,c,2) = A(:,nc-c+1,2) \\
B(:,c,3) = A(:,nc-c+1,3) \\
\text{end} \\
\text{imwrite(B,'LawSchoolMirror.jpg')}
\]

Even more compact vectorized code to create a mirror image...

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

Clean up “noise” — median filtering

Dirt in the image!

Note how the “dirty pixels” look out of place

<table>
<thead>
<tr>
<th>150</th>
<th>149</th>
<th>152</th>
<th>153</th>
<th>152</th>
<th>155</th>
</tr>
</thead>
<tbody>
<tr>
<td>151</td>
<td>150</td>
<td>153</td>
<td>154</td>
<td>153</td>
<td>156</td>
</tr>
<tr>
<td>153</td>
<td>2</td>
<td>3</td>
<td>156</td>
<td>155</td>
<td>158</td>
</tr>
<tr>
<td>154</td>
<td>2</td>
<td>2</td>
<td>157</td>
<td>156</td>
<td>159</td>
</tr>
<tr>
<td>156</td>
<td>154</td>
<td>158</td>
<td>159</td>
<td>158</td>
<td>161</td>
</tr>
<tr>
<td>157</td>
<td>156</td>
<td>159</td>
<td>160</td>
<td>159</td>
<td>162</td>
</tr>
</tbody>
</table>

Assign “typical” neighborhood gray values to “dirty pixels”

<table>
<thead>
<tr>
<th>150</th>
<th>149</th>
<th>152</th>
<th>153</th>
<th>152</th>
<th>155</th>
</tr>
</thead>
<tbody>
<tr>
<td>151</td>
<td>150</td>
<td>153</td>
<td>154</td>
<td>153</td>
<td>156</td>
</tr>
<tr>
<td>153</td>
<td>7</td>
<td>7</td>
<td>156</td>
<td>155</td>
<td>158</td>
</tr>
<tr>
<td>154</td>
<td>7</td>
<td>7</td>
<td>157</td>
<td>156</td>
<td>159</td>
</tr>
<tr>
<td>156</td>
<td>154</td>
<td>158</td>
<td>159</td>
<td>158</td>
<td>161</td>
</tr>
<tr>
<td>157</td>
<td>156</td>
<td>159</td>
<td>160</td>
<td>159</td>
<td>162</td>
</tr>
</tbody>
</table>
What are “typical neighborhood gray values”?

**Median**

**Mean**

radius 1

radius 2

Median Filtering

- Visit each pixel
- Replace its gray value by the median of the gray values in the “neighborhood”

Using a radius 1 “neighborhood”

Before

After

Visit every pixel; compute its new value.

m = 9

n = 18

for i=1:m

for j=1:n

Compute new gray value for pixel (i,j).

end

end

What we need...

1. A function that computes the median value in a 2-dimensional array C:

   \[ m = \text{medVal}(C) \]

2. A function that builds the filtered image by using median values of radius r neighborhoods:

   \[ B = \text{medFilter}(A,r) \]

Computing the median

\[ x = \begin{bmatrix} 21 & 89 & 36 & 28 & 19 & 88 & 43 \end{bmatrix} \]

\[ x = \text{sort}(x) \]

\[ x = \begin{bmatrix} 19 & 21 & 28 & 36 & 43 & 88 & 89 \end{bmatrix} \]

\[ n = \text{length}(x); \quad \% \ n = 7 \]

\[ m = \text{ceil}(n/2); \quad \% \ m = 4 \]

\[ \text{med} = x(m); \quad \% \ \text{med} = 36 \]

If \( n \) is even, then use:

\[ \text{med} = x(m)/2 + x(m+1)/2 \]
Median of a 2D array

```matlab
function med = medVal(C)
    [nr,nc] = size(C);
    x = zeros(1,nr*nc);
    for r=1:nr
        x((r-1)*nc+1:r*nc) = C(r,:);
    end
    % Compute median of x and assign to med
    % ...
end
```

See `medVal.m`

Back to filtering...

```matlab
m = 9
n = 18

for i=1:m
    for j=1:n
        Compute new gray value for pixel (i,j)
    end
end
```

When window is inside...

```matlab
m = 9
n = 18

New gray value for pixel (7,4) =
medVal( A(6:8,3:5) )
```

When window is partly outside...

```matlab
m = 9
n = 18

New gray value for pixel (7,1) =
medVal( A(6:8,1:2) )
```

The Pixel (i,j) Neighborhood

```matlab
iMin = max(1,i-r)
iMax = min(m,i+r)
jMin = max(1,j-r)
jMax = min(n,j+r)
C = A(iMin:iMax,jMin:jMax)
```

```matlab
m
n
z = 1
z = 2
```
**Finding Edges**

**What is an edge?**

Near an edge, grayness values change abruptly

<table>
<thead>
<tr>
<th>200</th>
<th>200</th>
<th>200</th>
<th>200</th>
<th>200</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
<td>200</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

**General plan for showing the edges in an image**

- Identify the “edge pixels”
- Highlight the edge pixels
  - make edge pixels white; make everything else black

**Rate-of-Change-Array**

Suppose \(A\) is an image array with integer values between 0 and 255.
Let \(B(i,j)\) be the maximum difference between \(A\) and its eight neighbors.
So \(B(i,j)\) is the maximum value in

\[
\text{Neighborhood of } A(i,j) = \max(\max(1,i-1):\min(m,i+1),\ldots,\max(1,j-1):\min(n,j+1)) - A(i,j)
\]

**Rate-of-change example**

Be careful! In "uint8 arithmetic" 57 - 60 is 0

<table>
<thead>
<tr>
<th>90</th>
<th>81</th>
<th>65</th>
</tr>
</thead>
<tbody>
<tr>
<td>62</td>
<td>60</td>
<td>59</td>
</tr>
<tr>
<td>56</td>
<td>57</td>
<td>58</td>
</tr>
</tbody>
</table>
function Edges(jpgIn,jpgOut,tau)
% jpgOut is the "edge diagram" of image jpgIn.
% At each pixel, if rate-of-change > tau
% then the pixel is considered to be on an edge.
A = rgb2gray(imread(jpgIn)); % Built-in function to
[m,n] = size(A); % convert to grayscale.
B = uint8(zeros(m,n)); % Returns 2-d array.
for i = 1:m
    for j = 1:n
        Neighbors = A(max(1,i-1):min(i+1,m), ...
                        max(1,j-1):min(j+1,n)); % for an interior pixel
        Diff= abs(double(Neighbors)-double(A(i,j)));
        colMax = max(Diff); % Compute largest value in each column
        B(i,j) = max(colMax); % Compute the max of the column max's
    end
end
imwrite(B,jpgOut,'jpg')

Recipe for rate-of-change \( B(i,j) \)

\[
\begin{align*}
\text{The 3-by-3 subarray that includes } A(i,j) \\
\text{and its 8 neighbors (for an interior pixel)} \\
\text{Diff= abs(double(Neighbors) - double(A(i,j)))} \\
\text{colMax = max(Diff)} \\
\text{B(i,j) = max(colMax)}
\end{align*}
\]

“Edge pixels” are now identified; display them with maximum brightness (255)

\[
\begin{align*}
A & \\
\text{if } B(i,j) > \tau \\
\text{B(i,j) = 255; } & \\
\text{else } & \\
\text{B(i,j) = 0; }
\end{align*}
\]

Edge finding: Effect of edge threshold, \( \tau \)