- Previous Lecture:
  - Image processing
    - Add frame, mirror

- Today’s Lecture:
  - More image processing
    - Flipping an image
    - Color $\rightarrow$ grayscale
    - “Noise” filtering
    - (Watch online/read in book: Edge finding example)

- Announcements:
  - Discussion this week in the classrooms as listed on Student Center
  - Project 4 due Mon Oct 24th
  - Pick up your prelim paper during consulting hours
Grayness: a value in $[0..255]$

- $0 = \text{black}$
- $255 = \text{white}$

These are integer values
Type: `uint8`

<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>151</td>
<td>155</td>
<td>156</td>
<td>155</td>
<td>158</td>
</tr>
<tr>
<td>154</td>
<td>153</td>
<td>156</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>
A color picture is made up of RGB matrices → 3-d array

E.g., color image data is stored in a 3-d array $A$:

$$0 \leq A(i,j,1) \leq 255$$
$$0 \leq A(i,j,2) \leq 255$$
$$0 \leq A(i,j,3) \leq 255$$
A color picture is made up of RGB matrices → 3-d array

Operations on images amount to operations on matrices!
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.
Reading and writing jpg files

% 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')
A 3-d array as 3 matrices

\[[\text{nr}, \text{nc}, \text{np}] = \text{size}(A) \quad \% \text{ dimensions of 3-d array } A\]

- **#rows:**
- **#columns:**
- **#layers (pages):**

\[
\begin{align*}
\text{M1} &= A(:,:,1) \\
\text{M2} &= A(:,:,2) \\
\text{M3} &= A(:,:,3)
\end{align*}
\]

\[
\begin{align*}
\text{A}(1:\text{nr},1:\text{nc},1) &= 4\text{-by}-6 \\
\text{M1} &= A(:,:,1) \\
\text{M2} &= A(:,:,2) \\
\text{M3} &= A(:,:,3)
\end{align*}
\]
% Store mirror image of A in array B

\[ \text{[nr, nc, np]} = \text{size}(A); \]

\textit{for } r = 1:nr

\textit{for } c = 1:nc

\hspace{1cm} \text{B}(r, c) = \text{A}(r, \text{nc} - c + 1);

\textit{end}

\textit{end}
%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
end
Both fragments create a mirror image of $A$.

true
false

$A$  $B$
\[ \text{Both fragments create a mirror image of } A. \]

\[ A \quad \text{true} \]

\[ B \quad \text{false} \]
% 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

Column c in B
is column nc-c+1 in A
Consider a single matrix (just one layer)

\[
\begin{align*}
[\text{nr}, \text{nc}, \text{np}] &= \text{size}(A); \\
\text{for } c &= 1:\text{nc} \\
&B(1:\text{nr}, c) = A(1:\text{nr}, \text{nc}-c+1); \\
\end{align*}
\]

end
Consider a single matrix (just one layer)

\[
[nr,nc,np] = \text{size}(A);
\]

\[
\text{for } c = 1:nc
\]

\[
B( :,c ) = A( :,nc-c+1 );
\]

\[
\text{end}
\]

The colon says “all indices in this dimension.” In this case it says “all rows.”
Now repeat for all layers

```matlab
[nr,nc,np] = size(A);
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)
end
```
Vectorized code to create a mirror image

\[
\begin{align*}
A &= \text{imread('LawSchool.jpg')} \\
[nr,nc,np] &= \text{size}(A); \\
\text{for } c &= 1:nc \\
   B(:,c,1) &= A(:,nc-c+1,1) \\
   B(:,c,2) &= A(:,nc-c+1,2) \\
   B(:,c,3) &= A(:,nc-c+1,3) \\
\text{end} \\
\text{imwrite}(B,'LawSchoolMirror.jpg')
\end{align*}
\]
Even more compact vectorized code to create a mirror image...

```matlab
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)
end

B = A(:,nc:-1:1,:)```
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

\[ \frac{R}{3} + \frac{G}{3} + \frac{B}{3} \]

\[ .3R + .59G + .11B \]
Averaging the RGB values to get a gray value

\[
M(i,j) = 0.3R(i,j) + 0.59G(i,j) + 0.11B(i,j)
\]

for \( i = 1:m \)
for \( j = 1:n \)
end
end

scalar operation
Averaging the RGB values to get a gray value

\[ M = 0.3R + 0.59G + 0.11B \]
Here are 2 ways to calculate the average. Are gray value matrices g and h the same given image data A?

\[
\begin{align*}
\text{for } r &= 1 : nr \\
&\quad \text{for } c = 1 : nc \\
&\quad \quad g(r,c) = \frac{A(r,c,1)}{3} + \frac{A(r,c,2)}{3} + \ldots \\
&\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \Quad
Matlab has a built-in function to convert from color to grayscale, resulting in a 2-d array:

\[ B = \text{rgb2gray}(A) \]
Clean up “noise” — median filtering
Dirt in the image!

Note how the “dirty pixels” look out of place.

150 149 152 153 152 155
151 150 153 154 153 156
153 2 3 156 155 158
154 2 1 157 156 159
156 154 158 159 158 161
157 156 159 160 159 162
What to do with the dirty pixels?

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

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>149</td>
<td>152</td>
<td>153</td>
<td>152</td>
<td>155</td>
</tr>
<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>?</td>
<td>?</td>
<td>154</td>
<td>156</td>
<td>155</td>
</tr>
<tr>
<td>154</td>
<td>?</td>
<td>?</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.

\[
\begin{array}{c}
\text{for } i=1:m \\
\quad \text{for } j=1:n \\
\quad \quad \text{Compute new gray value for pixel \((i,j)\).} \\
\quad \text{end} \\
\text{end}
\end{array}
\]
Replace \( \square \) with the median of the values under the window.
Replace $\times$ with the median of the values under the window.
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

\[
x : \begin{array}{cccccc}
21 & 89 & 36 & 28 & 19 & 88 & 43
\end{array}
\]

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

\[
x : \begin{array}{cccccc}
19 & 21 & 28 & 36 & 43 & 88 & 89
\end{array}
\]

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

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

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

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

\[
\text{med} = x(m)/2 + x(m+1)/2
\]
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
    % ...
Back to filtering…

\[
\begin{array}{cccccccccc}
\rotatebox{90}{\( m = 9 \)} & & & & & & & & & \\
\rotatebox{90}{\( n = 18 \)} & & & & & & & & & \\
\end{array}
\]

\begin{verbatim}
for i=1:m
  for j=1:n
    Compute new gray value for pixel (i,j)
  end
end
\end{verbatim}
When window is inside...

New gray value for pixel (7,4) =

\[ \text{medVal}( A(6:8,3:5) ) \]
When window is partly outside...

New gray value for pixel (7,1) =

\[
\text{medVal}( A(6:8,1:2) )
\]
When window is partly outside...

New gray value for pixel \((9,18)\) = 

\[
\text{medVal}( A(8:9,17:18) )
\]
The Pixel (i,j) Neighborhood

\[ i_{\text{Min}} = i - r \]
\[ i_{\text{Max}} = i + r \]
\[ j_{\text{Min}} = j - r \]
\[ j_{\text{Max}} = j + r \]
\[ C = A(i_{\text{Min}}:i_{\text{Max}},j_{\text{Min}}:j_{\text{Max}}) \]
The Pixel (i,j) Neighborhood

\[ i_{\text{Min}} = \max(1, i-r) \]
\[ i_{\text{Max}} = \min(m, i+r) \]
\[ j_{\text{Min}} = \max(1, j-r) \]
\[ j_{\text{Max}} = \min(n, j+r) \]
\[ C = A(i_{\text{Min}}:i_{\text{Max}}, j_{\text{Min}}:j_{\text{Max}}) \]
function $B = \text{medFilter}(A, r)$
\text{\% B from A via median filtering}
\text{\% with radius r neighborhoods.}

$$[m,n] = \text{size}(A);$$

$B = \text{uint8}(\text{zeros}(m,n));$

\text{for i=1:m}
\text{\hspace{1em}for j=1:n}
\text{\hspace{2em}$C = \text{pixel (i,j) neighborhood}$}
\text{\hspace{2em}$B(i,j) = \text{medVal}(C);$}
\text{\hspace{1em}end}
\text{\hspace{1em}end}$

end
$B = \text{medianFilter}(A, 3)$