Previous Lecture:
- Characters and strings
- Image processing, type `uint8`
  - Add frame

Today’s Lecture:
- More image processing
  - `color`→`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
Grayness: a value in \([0..255]\)

0 = black
255 = white

These are \textit{integer} values
Type: \texttt{uint8}

\begin{verbatim}
150 149 152 153 152 155
151 150 153 154 153 156
153 151 155 156 155 158
154 153 156 157 156 159
156 154 158 159 158 161
157 156 159 160 159 162
\end{verbatim}
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

\[ 0.3R + 0.59G + 0.11B \]

\[ \frac{R}{3} + \frac{G}{3} + \frac{B}{3} \]
Averaging the RGB values to get a gray value

\[ M = 0.3R + 0.59G + 0.11B \]
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

for \( i = 1:m \)
    for \( j = 1:n \)
        \[ M(i,j) = 0.3A(i,j,1) + 0.59A(i,j,2) + 0.11A(i,j,3) \]
    end
end
Averaging the RGB values to get a gray value

\[ M(i,j) = 0.3A(i,j,1) + 0.59A(i,j,2) + 0.11A(i,j,3) \]

Non-vectorized

\[
\begin{align*}
\text{for } i &= 1:m \\
&\quad \text{for } j = 1:n \\
&\quad \quad M(i,j) = 0.3A(i,j,1) + 0.59A(i,j,2) + 0.11A(i,j,3) \\
&\quad \text{end} \\
&\text{end}
\end{align*}
\]

Vectorized

\[ M = 0.3A(:, :, 1) + 0.59A(:, :, 2) + 0.11A(:, :, 3) \]
Here are 2 ways to calculate the average. Are gray value matrices \( g \) and \( h \) the same given image data \( A \)?

\[
\text{for } r = 1:\text{nr}
\text{for } c = 1:\text{nc}
\begin{align*}
g(r,c) &= \frac{A(r,c,1)}{3} + \frac{A(r,c,2)}{3} + \ldots + \frac{A(r,c,3)}{3}; \\
h(r,c) &= \ldots \\
&\quad (\frac{A(r,c,1)+A(r,c,2)+A(r,c,3)}{3})/3;
\end{align*}
\]

A: yes  
B: no
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.
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')
% Store mirror image of A in array B

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

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

end

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
```

Both fragments create a mirror image of $A$. 

- $A$ : true
- $B$ : false
Both fragments create a mirror image of \( A \).

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

This is non-vectorized code.
% Make mirror image of A -- the whole thing

A = imread('LawSchool.jpg');
[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

imshow(B) % Show 3-d array data as an image
imwrite(B,'LawSchoolMirror.jpg')
% 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

A
Vectorized code simplifies things…
Work with a whole column at a time
Vectorized code simplifies things…
Work with a whole column at a time
Vectorized code simplifies things…
Work with a whole column at a time
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)

\[
[\text{nr,nc,np}] = \text{size}(A) ;
\]
\[
\text{for } c = 1 : \text{nc} \\
\quad B(1:\text{nr},c) = A(1:\text{nr},\text{nc-c+1}) ;
\]

end
Consider a single matrix (just one layer)

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

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

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

\[
\text{end}
\]
Now repeat for all layers

\[
\begin{align*}
[nr, nc, np] &= \text{size}(A); \\
\text{for } c &= 1:nc \\
B(:,c,1) &= A(:,nc-c+1,1) \\
B(:,c,2) &= A(:,nc-c+1,2) \\
B(:,c,3) &= A(:,nc-c+1,3)
\end{align*}
\] 
end
Vectorized code to create a mirror image

```
A = imread('LawSchool.jpg')
[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
imwrite(B,'LawSchoolMirror.jpg')
```
Even more compact vectorized code to create a mirror image…

```latex
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,:)```

Clean up “noise” — median filtering
Dirt in the image!

Note how the "dirty pixels" look out of place.
What to do with the dirty pixels?

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

\[
\begin{array}{c}
\text{m} = 9 \\
\text{n} = 18 \\
\end{array}
\]

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

\[ i = 1 \]
\[ j = 2 \]

Filtered:

Replace \( \bullet \) with the median of the values under the window.
Replace \( \Box \) with the median of the values under the window.
Original:

\[
i = 2
\]
\[
j = 2
\]

Filtered:

Replace \( \times \) with the median of the values under the window.
Original:

\[ i = m \]
\[ j = n \]

Filtered:

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

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

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

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

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

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

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

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 
    % ...

See medVal.m
Back to filtering...

\[ m = 9 \]
\[ n = 18 \]

\[
\begin{align*}
\text{for } i &= 1:m \\
& \quad \text{for } j = 1:n \\
& \quad \quad \text{Compute new gray value for pixel (i,j)} \\
& \quad \text{end} \\
\text{end}
\end{align*}
\]
When window is inside...

\[ m = 9 \]
\[ n = 18 \]

New gray value for pixel \((7,4)\) = medVal( \(A(6:8,3:5)\) )
When window is partly outside…

\[
\text{New gray value for pixel } (7,1) = \text{medVal}( \text{A}(6:8,1:2) )
\]
When window is partly outside...

New gray value for pixel (9,18) = 

\[
\text{medVal}( A(8:9,17:18) )
\]
function B = medFilter(A,r)
% B from A via median filtering
% with radius r neighborhoods.

[m,n] = size(A);
B = uint8(zeros(m,n));
for i=1:m
    for j=1:n
        C = pixel (i,j) neighborhood
        B(i,j) = medVal(C);
    end
end
The Pixel \((i,j)\) Neighborhood

\[
\begin{align*}
  \text{iMin} &= i-r \\
  \text{iMax} &= i+r \\
  \text{jMin} &= j-r \\
  \text{jMax} &= j+r
\end{align*}
\]

\[
C = A(\text{iMin} : \text{iMax}, \text{jMin} : \text{jMax})
\]
The Pixel \((i,j)\) Neighborhood

\[
\begin{align*}
    i_{\text{Min}} &= \max(1, i-r) \\
    i_{\text{Max}} &= \min(m, i+r) \\
    j_{\text{Min}} &= \max(1, j-r) \\
    j_{\text{Max}} &= \min(n, j+r) \\
    C &= A(i_{\text{Min}}:i_{\text{Max}}, j_{\text{Min}}:j_{\text{Max}})
\end{align*}
\]
B = medianFilter(A,3)