Previous Lecture:
- Image processing
  - Add frame, grayscale

Today’s Lecture:
- More image processing
  - Mirror, vectorized code
  - Color $\rightarrow$ grayscale, uint8
  - “Noise” filtering
  - (Read in book: Edge-finding example)

Announcements:
- Project 4A due Thurs; Project 4B delayed (due early next week)
- Lab exercise tomorrow: problem-solving, 2D vectorization
- Prelim 1 feedback on Canvas
  - Be sure to review—re-do—Prelim 1 now so that you have a firm foundation
Where did we leave off?

How to put a picture in a frame

Two approaches:

1. Ask every pixel whether it is covered by the frame
   - Easy to understand
2. Identify which subarrays are covered by the frame
   - More efficient; easy to vectorize
Pictures as matrices

Pixel: an element in a matrix (location corresponds to row, column index)

“Greyness”: a value in 0..255
A color picture is made up of RGB matrices → 3D array

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

$0 \leq A(i,j,1) \leq 255$

$0 \leq A(i,j,2) \leq 255$

$0 \leq A(i,j,3) \leq 255$
Visualize a 3D array as a stack of “layers” which are 2D arrays

Beware the two different “3”s:

- `dims = size(A)`  % `[720, 1280, 3]`
- `length(dims) == 3`  % A has 3 dimensions: rows, columns, layers
- `dims(3) == 3`  % A has 3 layers: red, green, blue
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, uncompressed to a
% a 3D array A of type uint8
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

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

for \( r = 1 : nr \)

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

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

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.
% 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); % zeros returns type double
B = uint8(B); % Convert B to type uint8

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 \( n_c - c + 1 \) in \( A \)
Consider a single matrix (just one layer)

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

for c = 1:nc

\[
\text{B}(\text{all rows}, c) = \text{A}(\text{all rows}, \text{nc}-c+1);
\]

end
Consider a single matrix (just one layer)

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

end
Consider a single matrix (just one layer)

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

for \( c = 1 : \text{nc} \)

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

end
Now repeat for all layers

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

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
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.
Converting from color (RGB) to grayscale
Averaging the RGB values to get a gray value

\[ R + G + \frac{3}{7}B \]

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

\[
M(i,j) = 0.21 \cdot R(i,j) + 0.72 \cdot G(i,j) + 0.07 \cdot B(i,j)
\]

for \( i = 1:m \)
for \( j = 1:n \)
\[
M(i,j) = 0.21 \cdot R(i,j) + 0.72 \cdot G(i,j) + 0.07 \cdot B(i,j)
\]
end
end

scalar operation
Averaging the RGB values to get a gray value

for i = 1:m
    for j = 1:n
        \( M(i,j) = 0.21 \times R(i,j) + 0.72 \times G(i,j) + 0.07 \times B(i,j) \)
    end
end
Averaging the RGB values to get a gray value

\[
\begin{align*}
\text{for } i &= 1:m \\
\text{for } j &= 1:n \\
M(i,j) &= 0.21A(i,j,1) + 0.72A(i,j,2) + 0.07A(i,j,3) \\
\text{end} \\
\text{end}
\end{align*}
\]
Averaging the RGB values to get a gray value

for i = 1:m
    for j = 1:n
        M(i,j) = 0.21*A(i,j,1) + 0.72*A(i,j,2) + 0.07*A(i,j,3)
    end
end

M = 0.21*A(:, :, 1) + 0.72*A(:, :, 2) + 0.07*A(:, :, 3)
Computing in type `uint8`

- Respect the range $[0..255]$
- Arithmetic on `uint8`'s results in `uint8`'s
- **Saturation** (also called “capped”)
  - $\text{uint8}(90) + \text{uint8}(200) \rightarrow 255 \text{ (type uint8)}$
  - $\text{uint8}(90) - \text{uint8}(200) \rightarrow 0 \text{ (type uint8)}$
- **Rounding** (not truncation)
  - $\text{uint8}(32)/\text{uint8}(3) \rightarrow 11 \text{ (type uint8)}$
- Arithmetic between a `uint8` and a `double` results in a `uint8`
  - $\text{uint8}(90) + 200 \rightarrow 255 \text{ (type uint8)}$
Here are 2 ways to calculate the average. Are gray value matrices $g$ and $h$ the same given uint8 image data $A$?

```plaintext
for r= 1:nr
    for c= 1:nc
        g(r,c)= A(r,c,1)/3 + A(r,c,2)/3 + ... + A(r,c,3)/3;
        h(r,c)= ...
            ( A(r,c,1)+A(r,c,2)+A(r,c,3) )/3;
    end
end
```

A: yes  B: not quite (rounding)  C: no (saturation)
Application: median filtering

How can we remove noise?
Dirty pixels look out-of-place
How to fix “bad” pixels?

• Visit each pixel
• Replace with typical values from its neighborhood
  • How to choose “typical” value?
  • How big is the neighborhood?
• “Typical”: mean vs. median
  • Median better for rejecting noise, preserving edges
• Neighborhood: moving window of radius $r$
Using a radius-1 neighborhood

Before

After
Top-down design

- Visit each pixel
- Choose a new gray value equal to the median of the old gray values in the “neighborhood”

```matlab
[nr,nc] = size(A);  % A is 2d array of image data
B = uint8(zeros(nr,nc));
for i = 1:nr
    for j = 1:nc
        C = neighborhood of pixel (i,j)
        B(i,j) = median of elements in C
    end
end
```
Replace \( \times \) with the median of the values under the window.
Replace \[ \Box \] with the median of the values under the window.
Replace with the median of the values under the window.
Original:

\[ i = 1 \]
\[ j = nc \]

Filtered:

Replace ✗ with the median of the values under the window.
Replace $\times$ with the median of the values under the window.
Replace \( \times \) with the median of the values under the window.
Replace the value at position \((i = nr, j = nc)\) with the median of the values under the window.
Details at a pixel \((i,j)\) with a radius 1 “neighborhood”

\[
\begin{array}{ccc}
7 & 7 & 6 \\
7 & 0 & 6 \\
7 & 6 & 6 \\
\end{array}
\quad \rightarrow \quad
\begin{array}{ccc}
7 & 7 & 6 \\
7 & 6 & 6 \\
7 & 6 & 6 \\
\end{array}
\]

% Get median value in a matrix \(xMat\)
\(xVec = xMat(:)\) % Convert matrix to vector
\(\text{medianVal} = \text{median}(xVec)\) % Use built-in function

Before

Replace pixel \((i,j)\) with median value

After
Deal with boundary issues – moving window

% Get C, the radius r
% neighborhood of pixel (i,j)

\[ i_{\text{Min}} = i - r \]
\[ i_{\text{Max}} = i + r \]
\[ j_{\text{Min}} = j - r \]
\[ j_{\text{Max}} = j + r \]

\[ C = A(i_{\text{Min}}:i_{\text{Max}},j_{\text{Min}}:j_{\text{Max}}) \]
Deal with boundary issues – moving window

% Get C, the radius r
% neighborhood of pixel (i,j)

\[
\begin{align*}
\text{iMin} &= \max(1, i-r) \\
\text{iMax} &= \min(nr, i+r) \\
\text{jMin} &= \max(1, j-r) \\
\text{jMax} &= \min(nc, j+r)
\end{align*}
\]

\[C = A(\text{iMin: iMax, jMin: jMax})\]

See Insight §12.4 for complete code: MedianFilter.m
\[ B = \text{medianFilter}(A, 3) \]
Mean Filter with radius 3
Mean Filter with radius 10
Mean filter fails because the mean does not capture representative values.

<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>1</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>

Mean-filtered values with radius 1 neighborhood:

85 86 87 88

Median-filtered values with radius 1 neighborhood:

150 150 153 154
Finding Edges: read example in Sec 12.4

Identify “sharp changes” in image data—a kind of outliers.

Subtracting uint8 values correctly to prevent “underflow”

“Thresholding”—use a parameter to control the amount of details extracted from image