Announcements

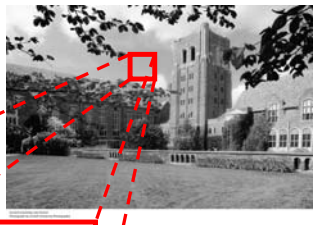
- Pick up Prelim 1 during consulting hrs (Carpenter Hall, ACCEL Green Rm)
- Dr. Fan will be away for the next three lectures. Lectures are being pre-recorded and will be put online W, M, and next W. Access them on the course website using your NetID, preferably before normal lecture time
- TAs will lead the lectures on 10/23, 10/28, 10/30. Attendance is optional
  - 10/23 Thurs: Go over Qs 1,2,5 of Prelim 1; answer your questions on the recorded lecture
  - 10/28 Tues: Go over Qs 3 & 4 of Prelim 1; answer your questions on the recorded lecture
  - 10/30 Thurs: Answer your questions on the recorded lecture

- Previous Lecture:
  - Image processing
    - Add frame, mirror
- Today's Lecture:
  - More image processing
    - color→grayscale
    - "Noise" filtering
    - Edge finding
- Announcements:
  - Discussion this week in the classrooms as listed on Student Center
  - Project 4 due Mon Oct 27th

Grayness: a value in [0..255]

0 = black
255 = white

These are *integer* values
Type: `uint8`

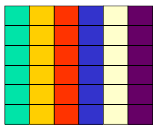


150	149	152	153	152	155
151	150	153	154	153	156
153	151	155	156	155	158
154	153	156	157	156	159
156	154	158	159	158	161
157	156	159	160	159	162


```
% 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')
```

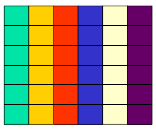

Vectorized code simplifies things…
Work with a whole column at a time

A



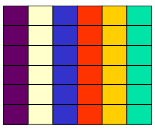
Vectorized code simplifies things…
Work with a whole column at a time

A



1 2 3 4 5 6

B



1 2 3 4 5 6

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

Consider a single matrix (just one layer)

```
[nr,nc,np] = size(A);
for c= 1:nc
    B(1:nr,c ) = A(1:nr,nc-c+1 );


end
```


Consider a single matrix (just one layer)

```
[nr,nc,np] = size(A);
for c= 1:nc
    B( : ,c  ) = A( : ,nc-c+1  );


end
```

The colon says "all indices in this dimension." In this case it says "all rows."

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…

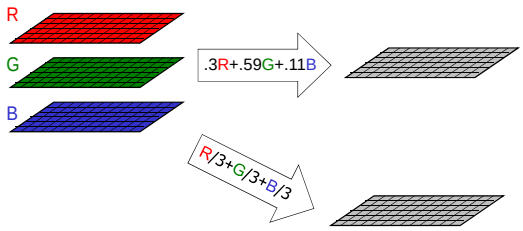
```
for c= 1:nc
    B(:,c,1) = A(:,nc-c+1,1)
    B(:,c,2) = A(:,nc-c+1,2)
    B(:,c,3) = A(:,nc-c+1,3)
end
```

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


Example: color → black and white

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

Averaging the RGB values to get a gray value



Averaging the RGB values to get a gray value


```
for  i= 1:m
   for j= 1:n
      M(i,j)= .3*R(i,j) + .59*G(i,j) + .11*B(i,j)
   end
end
```

scalar operation

Averaging the RGB values to get a gray value


```
M= .3*R + .59*G + .11*B
```

vectorized operation

showToGrayscale.m

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

B = rgb2gray(A)

Clean up "noise" — median filtering

Dirt in the image!

Note how the "dirty pixels" look out of place

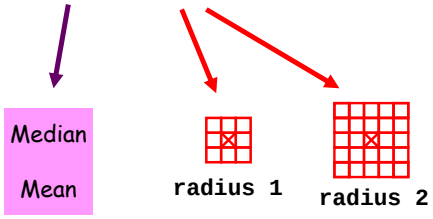
150	149	152	153	152	155
151	150	153	154	153	156
153	2	3	156	155	158
154	2	1	157	156	159
156	154	158	159	158	161
157	156	159	160	159	162

What to do with the dirty pixels?

Assign "typical" neighborhood gray values to "dirty pixels"

150	149	152	153	152	155
151	150	153	154	153	156
153	?	?	156	155	158
154	?	?	157	156	159
156	154	158	159	158	161
157	156	159	160	159	162

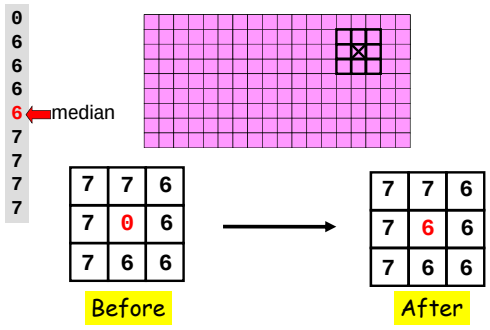
What are "typical neighborhood gray values"?



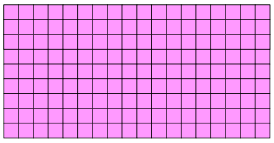
Median Filtering

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

Using a radius 1 "neighborhood"



Visit every pixel; compute its new value.




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


What we need…

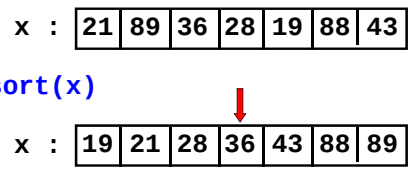
- (1) A function that computes the median value in a 2-dimensional array C:

```
m = medVal(C)
```

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

```
B = medFilter(A,r)
```


Computing the median




```
n = length(x);   % n = 7
m = ceil(n/2);   % m = 4
med = x(m);      % med = 36
```

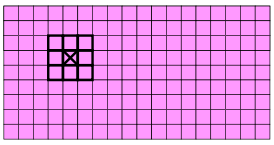
If n is even, then use : `med = x(m)/2 + x(m+1)/2`

Median of a 2D array


```
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

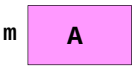
```
for i=1:m
   for j=1:n
      Compute new gray value for pixel (i,j)
   end
end
```



```
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

```
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)
```



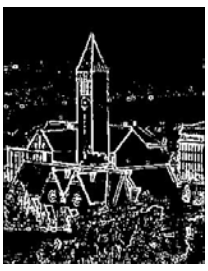
r = 1

r = 2

A

B = medianFilter(A,3)

Finding Edges



What is an edge?

Near an edge, grayness values change abruptly

```
200  200  200  200  200  200
200  200  200  200  200  100
200  200  200  200  100  100
200  200  200  100  100  100
200  200  100  100  100  100
200  100  100  100  100  100
```

General plan for showing the edges in in image

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

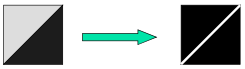
```
200  200  200  200  200  200
200  200  200  200  200  100
200  200  200  200  100  100
200  200  200  100  100  100
200  200  100  100  100  100
200  100  100  100  100  100
```

General plan for showing the edges in in image

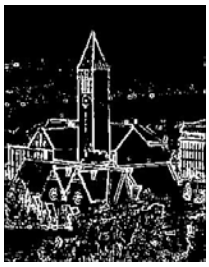
- Identify the "edge pixels"
- Highlight the edge pixels
  - make edge pixels white; make everything else black

```
200  200  200  200  200  200
200  200  200  200  200  100
200  200  200  200  100  100
200  200  200  100  100  100
200  200  100  100  100  100
200  100  100  100  100  100
```

BLACK WHITE BLACK



Finding Edges



The 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 and its eight neighbors.

So **B(i,j)** is the maximum value in

```
A(max(1,i-1):min(m,i+1),...
  max(1,j-1):min(n,j+1))   - A(i,j)
```

Neighborhood of A(i,j)

Rate-of-change example

90	81	65
62	60	59
56	57	58

Rate-of-change at middle pixel is 30

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

```
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));
[m,n] = size(A);
B = uint8(zeros(m,n));
for i = 1:m
    for j = 1:n

        B(i,j) =  ?????

    end
end
```

Built-in function to convert to grayscale. Returns 2-d array.

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

```
% The 3-by-3 subarray that includes A(i,j)
% and its 8 neighbors (for an interior pixel)
Neighbors = A(i-1:i+1,j-1:j+1);

% Subtract A(i,j) from each entry
Diff= abs( double(Neighbors) – double(A(i,j)) );

% Compute largest value in each column
colMax = max(Diff);
% Compute the max of the column max's
B(i,j) = max(colMax);
```



```
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));
[m,n] = size(A);
B = uint8(zeros(m,n));
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));
    B(i,j)=max(max(abs(double(Neighbors)– ...
                       double(A(i,j)))));
    if B(i,j) > tau
      B(i,j) = 255;
    end
  end
end
imwrite(B,jpgOut,'jpg')
```

Edge finding: Effect of edge threshold, τ

τ ⟶