

Lecture 19

Image processing



Lecture 19 Outline:

Image Processing (in MATLAB):

- Image Processing Toolbox
 - imread
 - imwrite
- Image Data
 - Vectorizing computations
- Example Transformations
 - Mirroring
 - Color to grayscale
 - Negative
 - Median noise filter
 - Edge detection

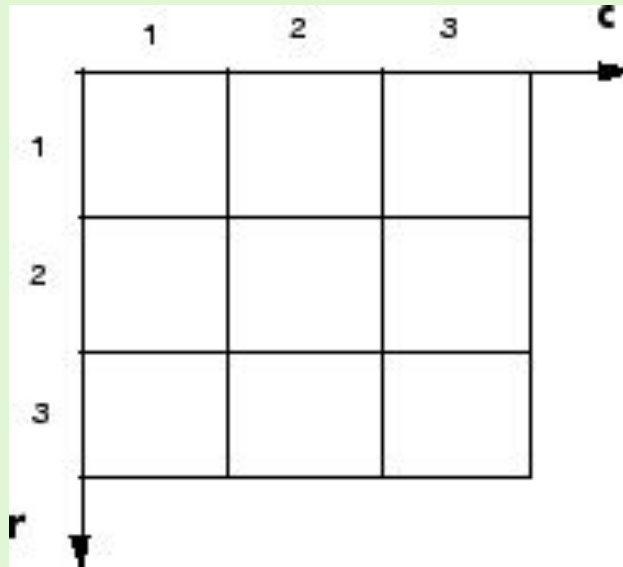
Images are stored as a bunch of numbers

318-by-25
0

49	55	58	59	57	53
60	67	71	72	72	70
102	108	111	111	112	112
157	167	169	167	165	164
196	205	208	207	205	205
199	208	212	214	213	216
190	192	193	195	195	197
174	169	165	163	162	161



Pixel Indexing in MATLAB



For an image A

- $A(1,1)$ is upper left corner pixel of image
- $A(\text{end},1)$ is lower left corner pixel
- $A(1,\text{end})$ is upper right corner
- $A(\text{end},\text{end})$ is lower right corner

Color Images

Each pixel has 3 color components, so can view a color image as 3 different matrices (aka *pages*), one for each color.

MATLAB stacks them in a 3D array, eg

$$0 \leq A(i,j,1) \leq 255 \quad (\text{red})$$

$$0 \leq A(i,j,2) \leq 255 \quad (\text{green})$$

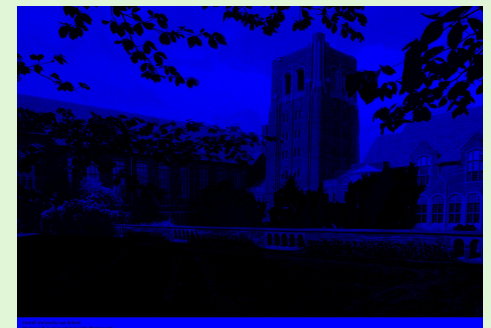
$$0 \leq A(i,j,3) \leq 255 \quad (\text{blue})$$



Note 3rd Subscript



Cornell University Law School
Photograph by Cornell University Photography



Storing Images

There are a lot of file formats for images.
Some common ones:

JPEG

(Joint Photographic Experts Group)

GIF

(Graphics Interchange Format)

PNG

(Portable Network Graphics)

Storing Images

Why are there so many formats?

- Different Compression Algorithms
 - Lossy (JPEG, GIF)
 - Lossless (PNG)
- Need for additional attributes
 - Patient Name, Radiologist Comments, etc. (DICOM)
- Intellectual Property Rights, Royalties

Compression Idea: Small Regions May be Similar

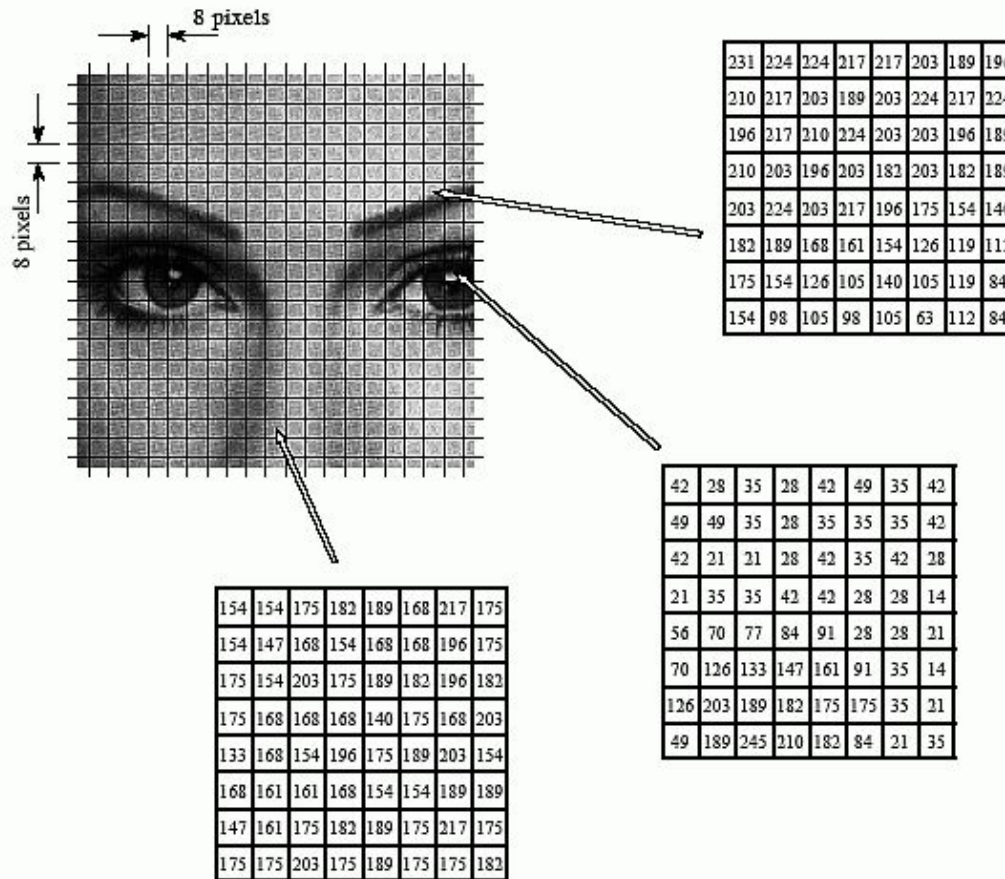


FIGURE 27-9
JPEG image division. JPEG transform compression starts by breaking the image into 8×8 groups, each containing 64 pixels. Three of these 8×8 groups are enlarged in this figure, showing the values of the individual pixels, a single byte value between 0 and 255.

Images can be written as a sum of a relatively small number of tables

1000-by-2000 picture might be well approximated by weighted sums of 100 tables:

2,000,000 vs 300,000 (100 x 3000)

Operations on Images

Amount to operations on 2D and 3D Arrays.

A good place to practice “array” thinking.

Two Problems

We have:



Cornell University Law School
Photograph by Cornell University Photography

LawSchool.jpg

Problem 1

Want:

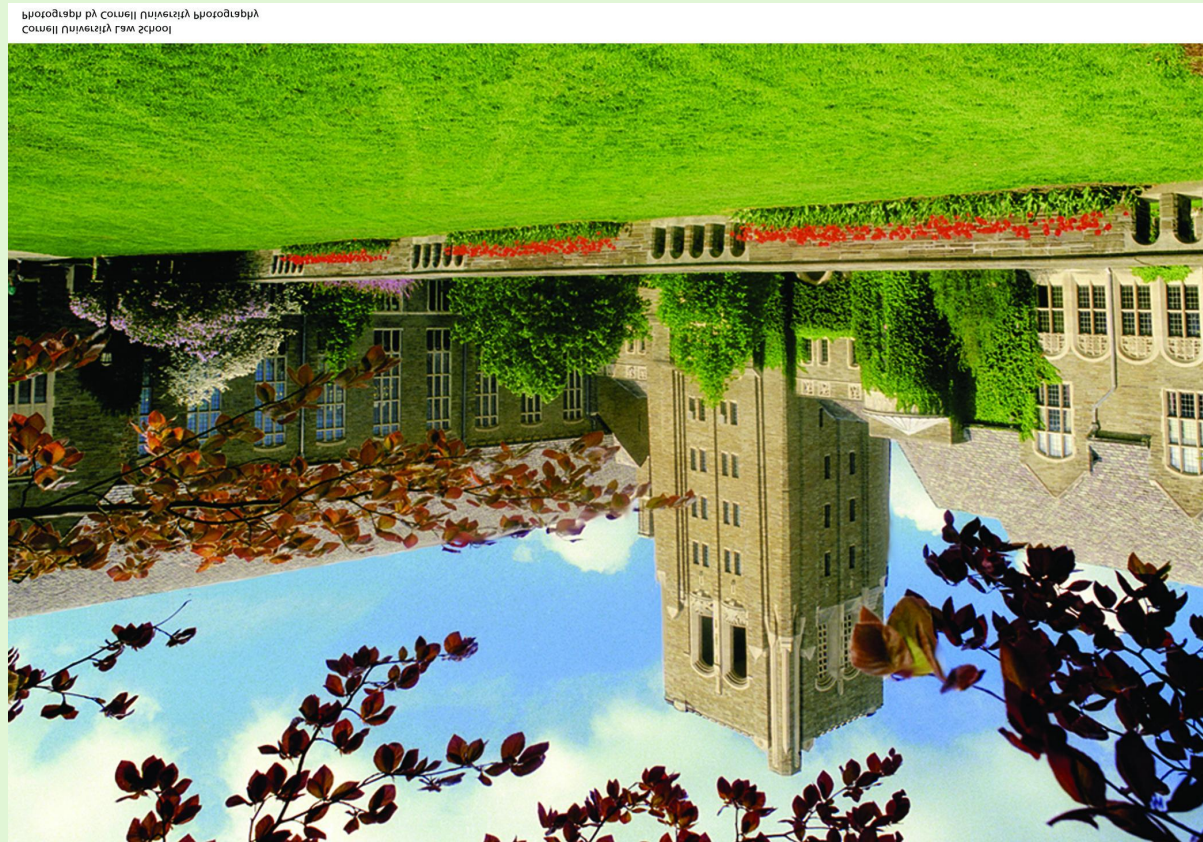


Photograph by Cornell University Photography
Cornell University Law School

LawSchoolMirror.jpg

Problem 2

Want:



LawSchoolUpDown . jpg

Solution Framework

Read `LawSchool1.jpg` from disk and convert it into an array.

Manipulate the Array.

Convert the array to a jpg file and write it to memory.

imread

```
% Read in image as 3D array...  
A = imread('LawSchool.jpg');
```

The color of the pixel at location (i,j) is given by

$A(i,j,1)$ = red value

$A(i,j,2)$ = green value

$A(i,j,3)$ = blue value

The 3D Array

```
>> [m,n,p] = size(A)
```

```
m =
```

```
1458 rows
```

```
n =
```

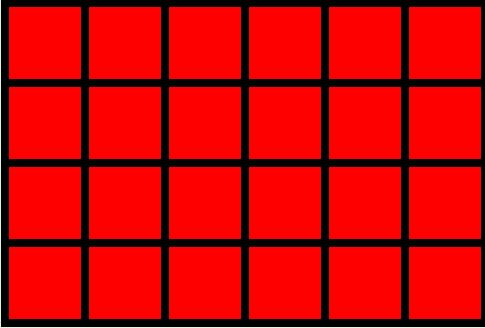
```
2084 column
```

```
p =
```

```
s
```

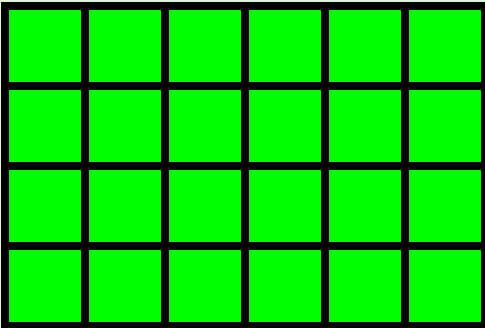
```
3 layers
```

The Layers



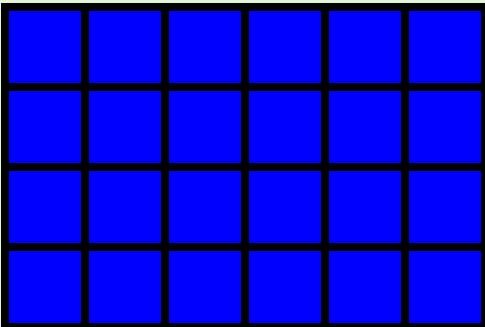
1458-by-2084

$A(:, :, 1)$



1458-by-2084

$A(:, :, 2)$



1458-by-2084

$A(:, :, 3)$

How can we find the Mirror image of A?

```
% Store left-right mirror of A  
% in array B
```

```
[nr,nc,np]= size(A);
```

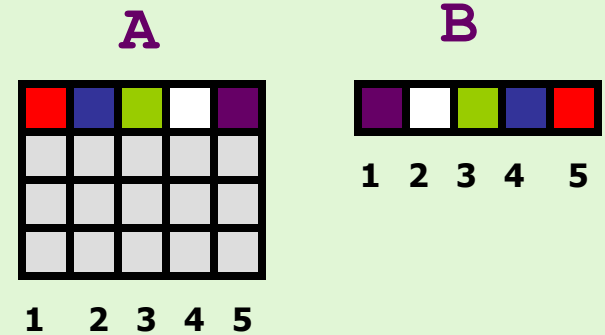
```
for r= 1:nr
```

```
    for c= 1:nc
```

```
        B(r,c, )= A(r,nc-c+1, );
```

```
    end
```

```
end
```



```
% Store left-right mirror 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
```

Left-Right Mirror Image (vectorized)

```
A = imread('LawSchool.jpg')
[m,n,p] = size(A);
for j=1:n
    B(:,j,1) = A(:,n+1-j,1)
    B(:,j,2) = A(:,n+1-j,2)
    B(:,j,3) = A(:,n+1-j,3)
end
imwrite(B, 'LawSchoolMirror.jpg')
```

What does each loop copy to B?

```
A = imread('LawSchool.jpg')
[m,n,p] = size(A);
for j=1:n
    B(:,j,1) = A(:,n+1-j,1)
    B(:,j,2) = A(:,n+1-j,2)
    B(:,j,3) = A(:,n+1-j,3)
end
imwrite(B, 'LawSchoolMirror.jpg')
```

- A) row
- B) column
- C) row+panel
- D) column+panel

What does each loop copy to B?

```
A = imread('LawSchool.jpg')
[m,n,p] = size(A);
for j=1:n
    B(:,j,1) = A(:,n+1-j,1)
    B(:,j,2) = A(:,n+1-j,2)
    B(:,j,3) = A(:,n+1-j,3)
end
imwrite(B, 'LawSchoolMirror.jpg')
```

- A) row
- B) column
- C) row+panel
- D) column+panel

MATLAB Loves to Vectorize!

```
for j=1:n
    B(:,j,1) = A(:,n+1-j,1)
    B(:,j,2) = A(:,n+1-j,2)
    B(:,j,3) = A(:,n+1-j,3)
end
```



```
B = A(:,end:-1:1,:);
```

The Mirror Image

```
A = imread('LawSchool.jpg');  
B = A(:,end:-1:1,:);  
Imwrite(A, 'LawSchoolMirror.jpg');
```

The Upside Down Image

```
A = imread('LawSchool.jpg');  
C = _____  
imwrite(C, 'LawSchoolUpDown.jpg');
```

- A) C(:, :, end:-1:1)
- B) C(:, end:-1:1, :)
- C) C(end:-1:1, :, :)
- D) I don't know

The Upside Down Image

```
A = imread('LawSchool.jpg');  
C = _____  
imwrite(C, 'LawSchoolUpDown.jpg');
```

- A) `C(:, :, end:-1:1)`
- B) `C(:, end:-1:1, :)`
- C) `C(end:-1:1, :, :)`
- D) I don't know

New Problem

Color → Black and White

Have:



Cornell University Law School
Photograph by Cornell University Photography

New Problem

Color → Black and White

Want:



Cornell University Law School
Photograph by Cornell University Photography

rgb2gray

```
A = imread('LawSchool.jpg');  
bwA = rgb2gray(A);  
imwrite(bwA, 'LawSchoolBW.jpg')
```


How Does the Conversion Work?

r	g	b	gray
167	219	241	206
66	35	15	42
95	14	20	39
163	212	242	201
182	228	215	213
225	244	222	236
136	199	240	185

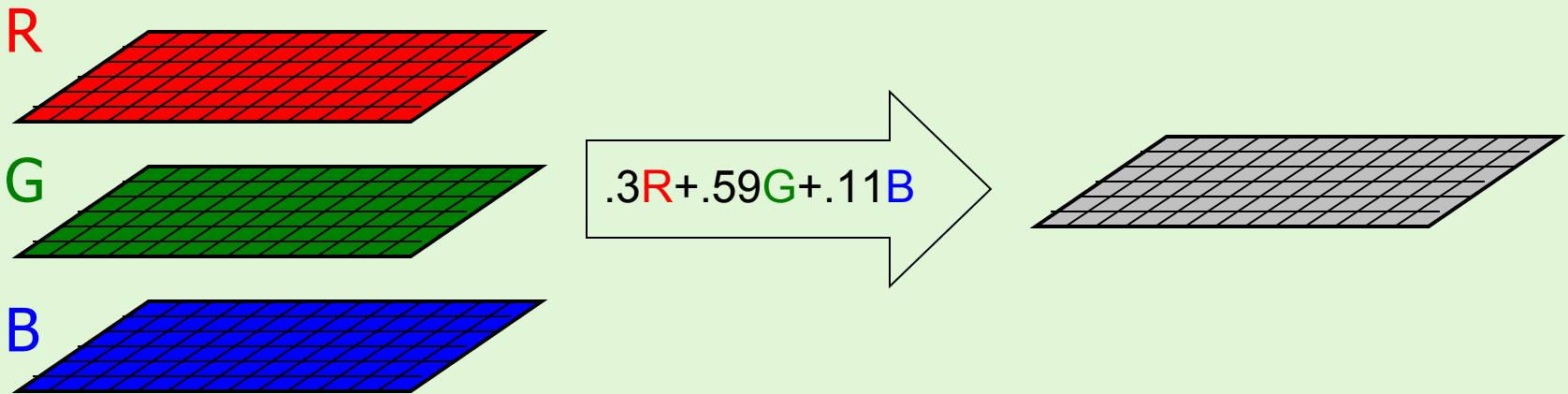
It's a
weighted
average

rgb2gray:



Cornell University Law School
Photograph by Cornell University Photography

Weighted average of the RGB values



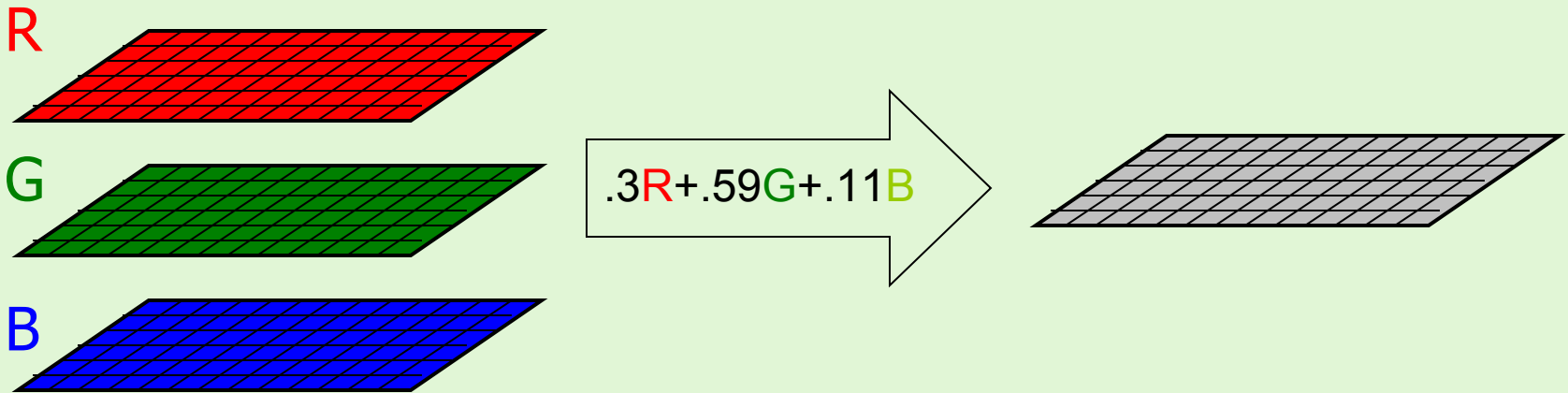
```
for i= 1:m
  for j= 1:n
    M(i,j)= .3*R(i,j) + .59*G(i,j) + .11*B(i,j)
  end
end
```

scalar operation

Why a Weighted Average?

- Due to the way our visual system works
- We are most sensitive to Green, then Red then Blue
- Other approaches are close
 - Equal Weights (i.e. $(r+g+b)/3$)
 - $\max(r, g, b)$

Weighted average of the RGB values



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

vectorized operation

Coding Average

```
bwA = uint8(zeros(m,n))
for i=1:m
    for j = 1:n
        bwA(i,j) = ( A(i,j,1) + ...
                    + A(i,j,2) + A(i,j,3) ) / 3;
    end
end
imwrite(bwA, 'LawSchool1BW.jpg')
```

Type **uint8**: unsigned 8-bit integers (0,1,2,...,255)

imshow(bwA)



Cornell University Law School
Photograph by Cornell University Photography

Whoa! Why is it so bad?

- uint8
 - Values always between 0 and 255
 - “uint8 arithmetic clips values”
 - Negative => Zero
 - 256 and larger => 255
 - E.g.,
 - `uint8(255)+100 == 255`

Convert image to double for average to work

Work with Doubles, then convert back

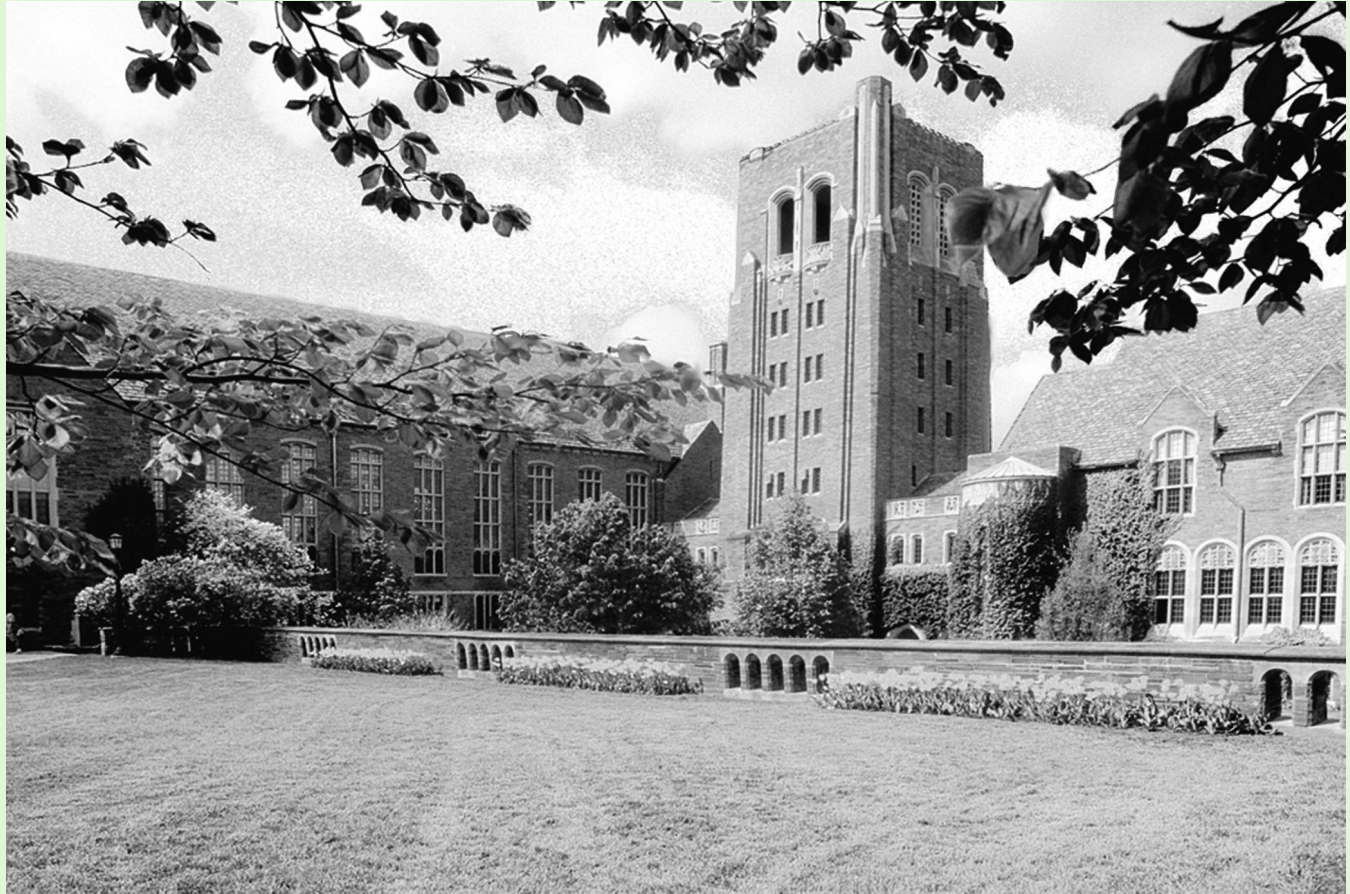
```
A = double(A);  
bwA = uint8(zeros(m,n))  
for i=1:m  
    for j = 1:n  
        bwA(i,j) = ( A(i,j,1) + ...  
                    + A(i,j,2) + A(i,j,3) ) / 3;  
    end  
end  
imwrite(bwA, 'LawSchoolBW.jpg')
```

Type **uint8**: unsigned 8-bit integers (0,1,2,...,255)

Max

```
bwA = uint8(zeros(m,n))
for i=1:m
    for j = 1:n
        bwA(i,j) = max([A(i,j,1) ...
            A(i,j,2)  A(i,j,3)]);
    end
end
imwrite(bwA,'LawSchoolBW.jpg')
```

Max:



Cornell University Law School
Photograph by Cornell University Photography

Vectorized Max?

- $\text{max}(A, [], n)$ finds the max along dimension n
 - For $n=1$, returns $r(j,k) = \text{max}(A(:,j,k))$
 - For $n=2$, returns $r(i,k) = \text{max}(A(i,:,k))$
 - For $n=3$, returns $r(i,j) = \text{max}(A(i,j,:))$

$$M = \text{max}(A, [], 3)$$

Problem: Produce a Negative



Idea

If matrix A represents the image and

$$B(i, j) = 255 - A(i, j)$$

for all i and j , then B will represent the negative.

```
function newIm = toNegative(im)
% newIm is the negative of image im
% im, newIm are 3-d arrays; each component is uint8

[nr,nc,np]= size(im);      % dimensions of im
newIm= zeros(nr,nc,np);   % initialize newIm
newIm= uint8(newIm);      % Type for image color values

for r= 1:nr
    for c= 1:nc
        for p= 1:np
            newIm(r,c,p)= 255 - im(r,c,p);
        end
    end
end
end
```

Vectorized toNegative

```
function newIm = toNegative(im)
% newIm is the negative of image im
% im, newIm are 3-d arrays
newIm = 255-im;
```

This is cleaner, clearer, and less error prone.
But, always keep complexity in mind.

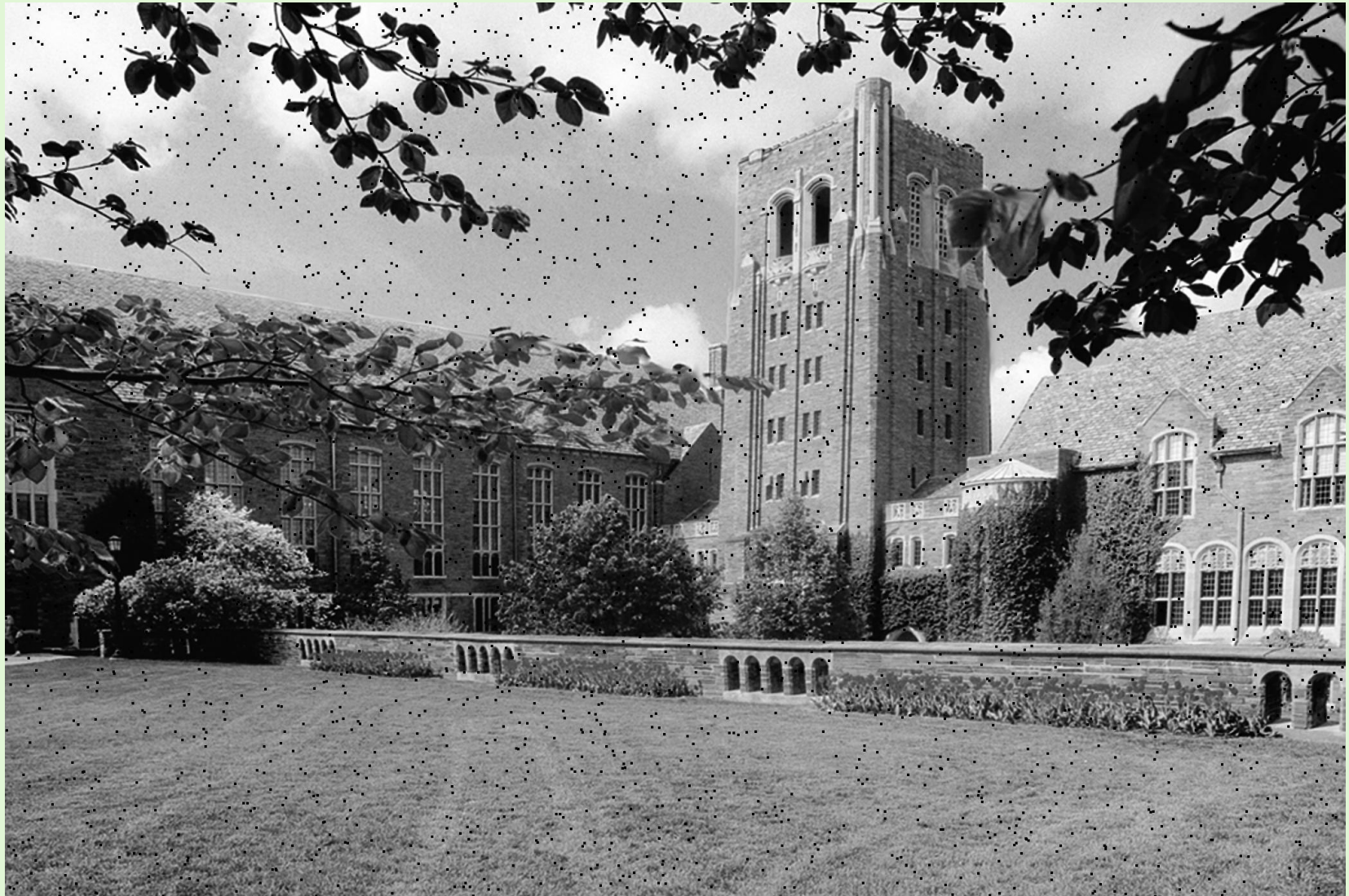
Working with Images

Previous operations were all element-wise, now we'll look at some region based operations

Filtering Noise

Edge Detection

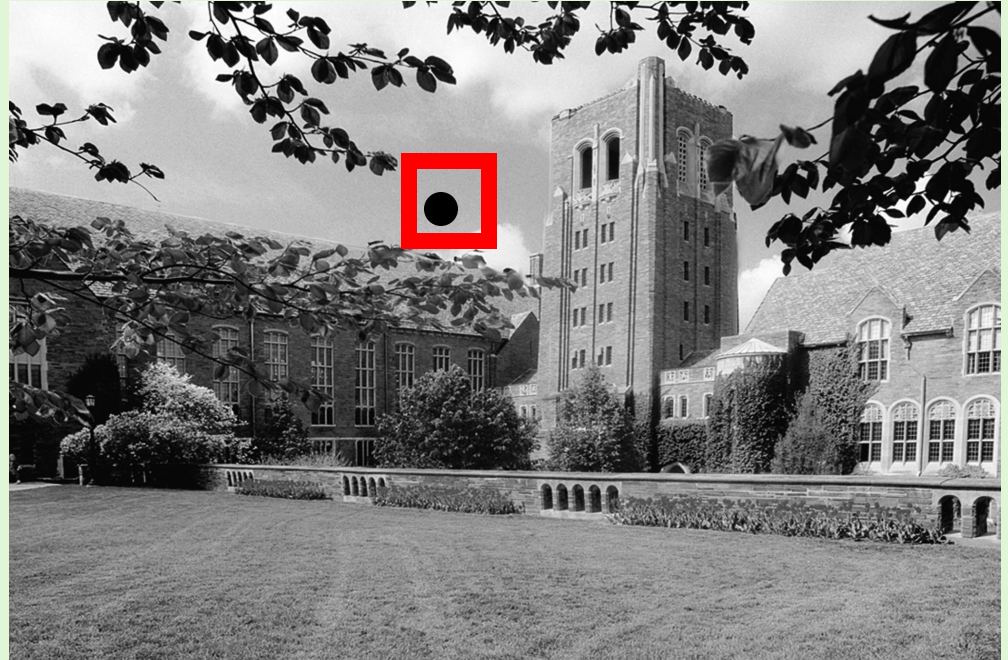
Can We Filter Out the “Noise”?



Cornell University Law School
Photograph by Cornell University Photography

Dirt!

1458-by-2084



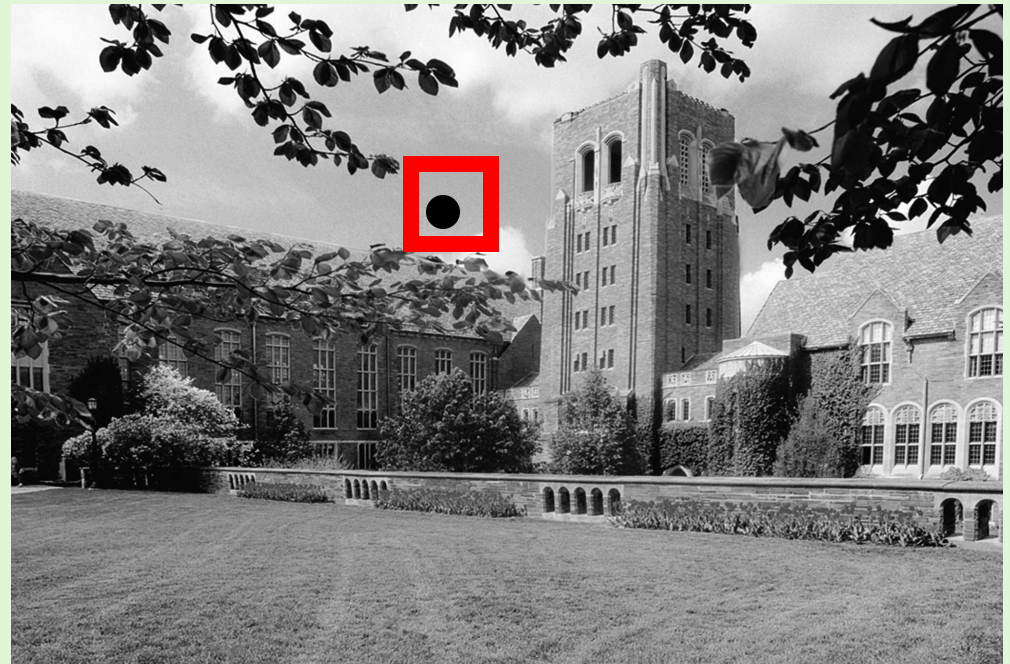
Cornell University Law School
Photograph by Cornell University Photography

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

Note how the
“dirty pixels”
look out of
place

Idea

1458-by-2084



Cornell University Law School
Photograph by Cornell University Photography

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

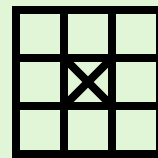
Assign “typical”
neighborhood
gray values to
“dirty pixels”

Getting Precise

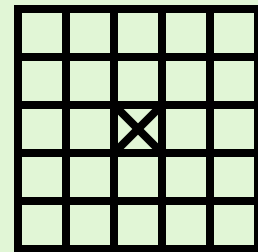
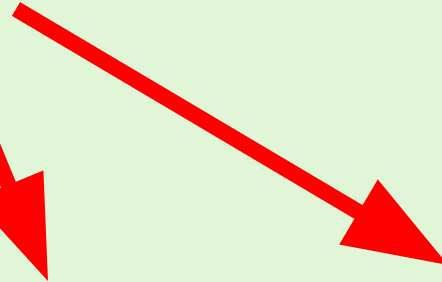
“Typical neighborhood gray values”



Could use
Median
Or
Mean



radius 1



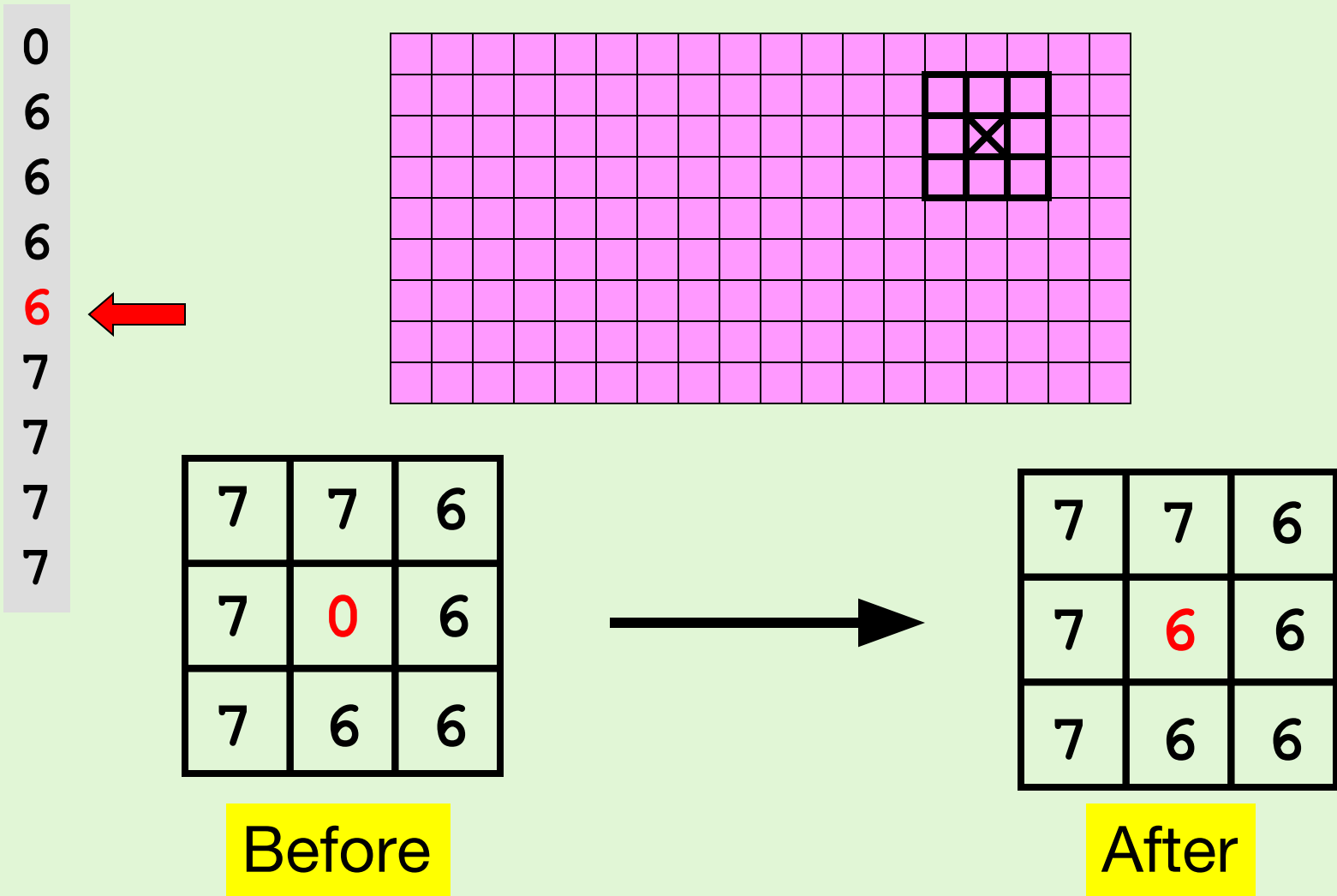
radius 3

We'll look at “Median Filtering” first...

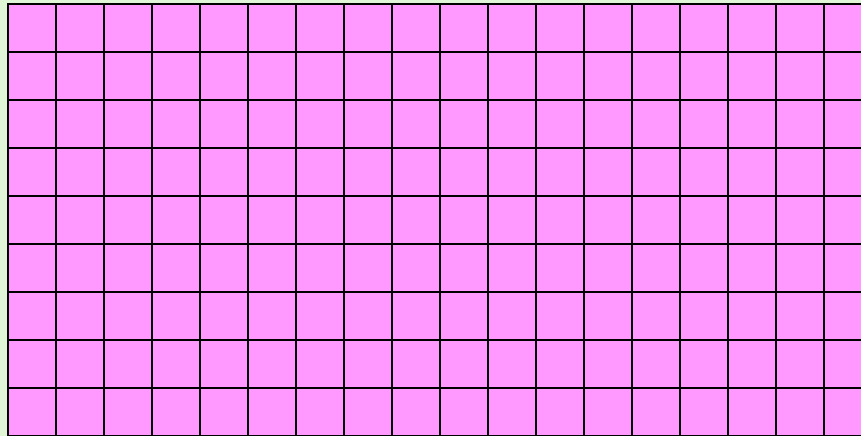
Median Filtering

At each pixel, replace its gray value by the median of the gray values in the “neighborhood” i.e. a small region surrounding the pixel.

Using a radius 1 “Neighborhood”



How to Visit Every Pixel

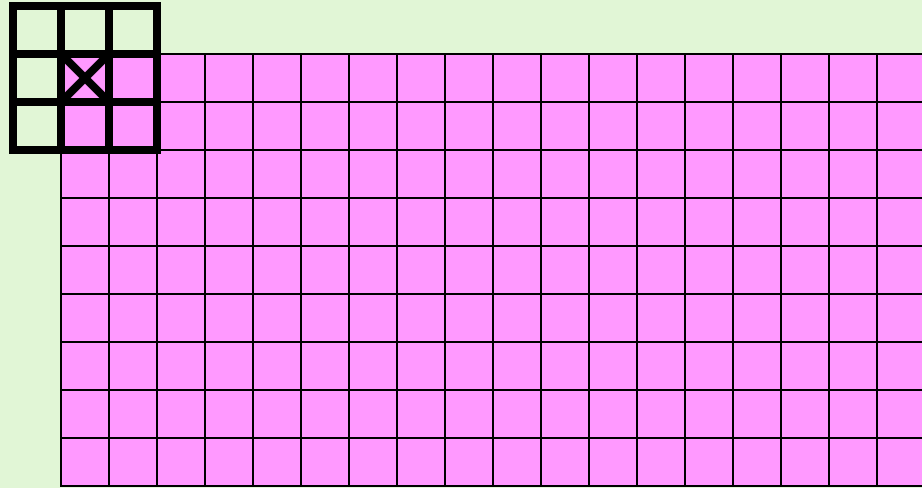


$m = 9$

$n = 18$

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


Original:

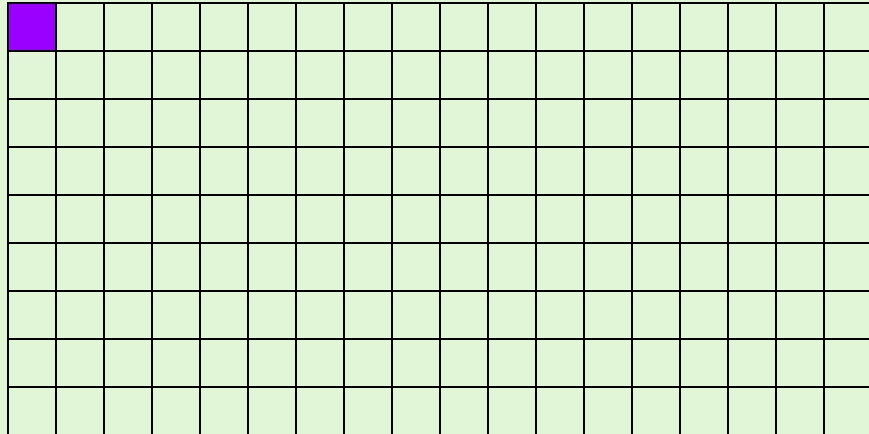


$$i = 1$$

$$j = 1$$

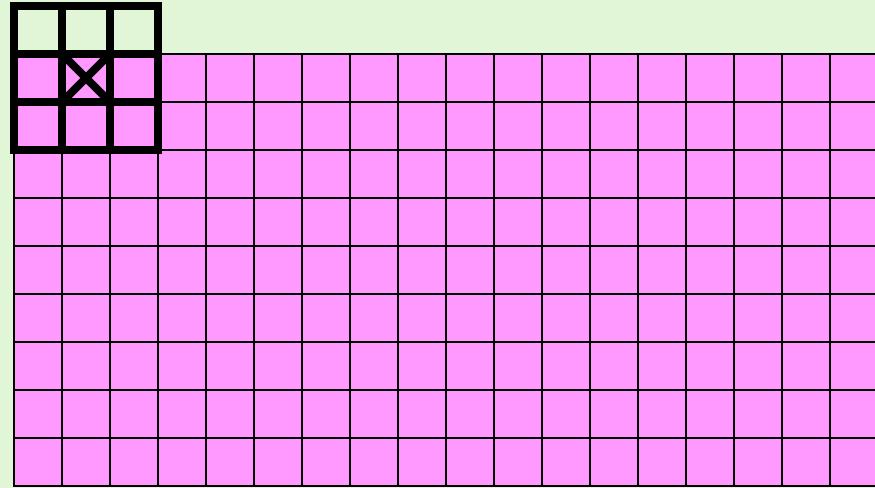


Filtered:



Replace  with the median of the values under the window.

Original:

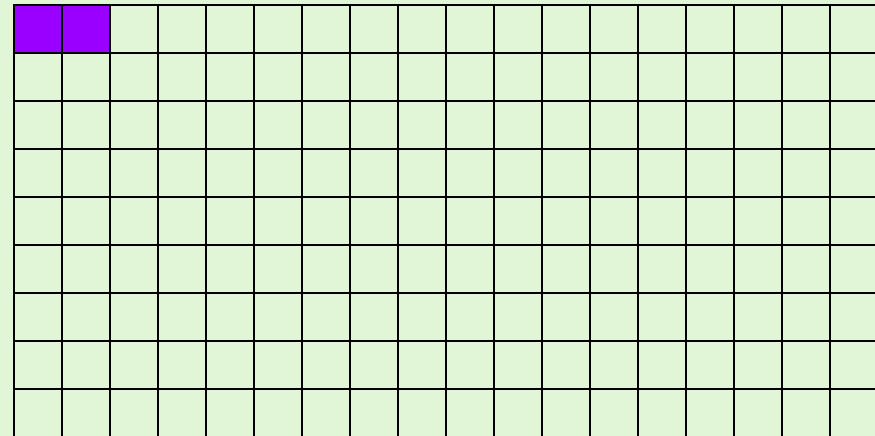



$$i = 1$$

$$j = 2$$

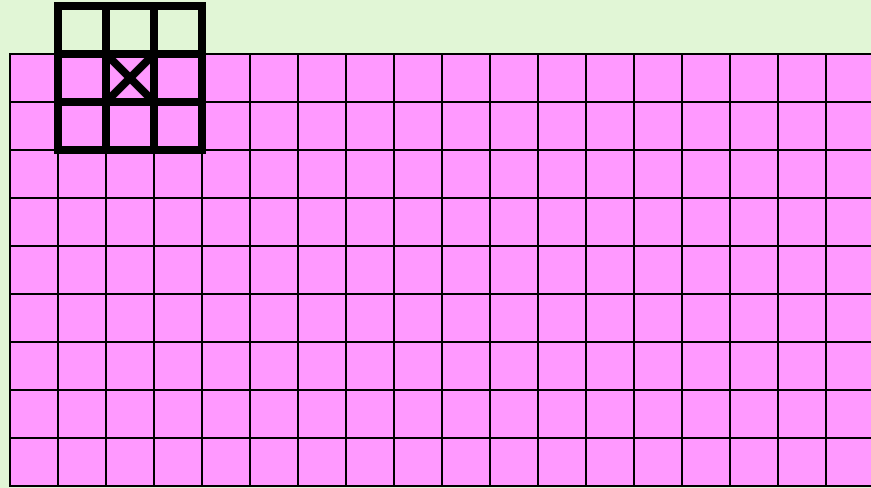


Filtered:



Replace  with the median of the values under the window.

Original:

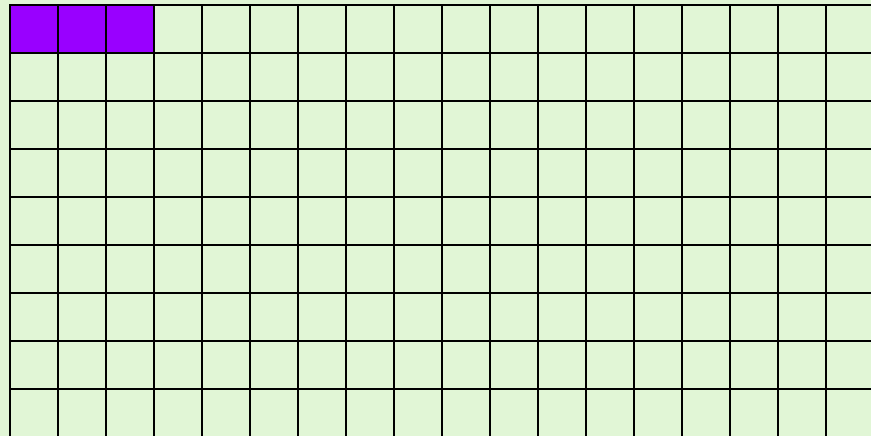



$$i = 1$$

$$j = 3$$

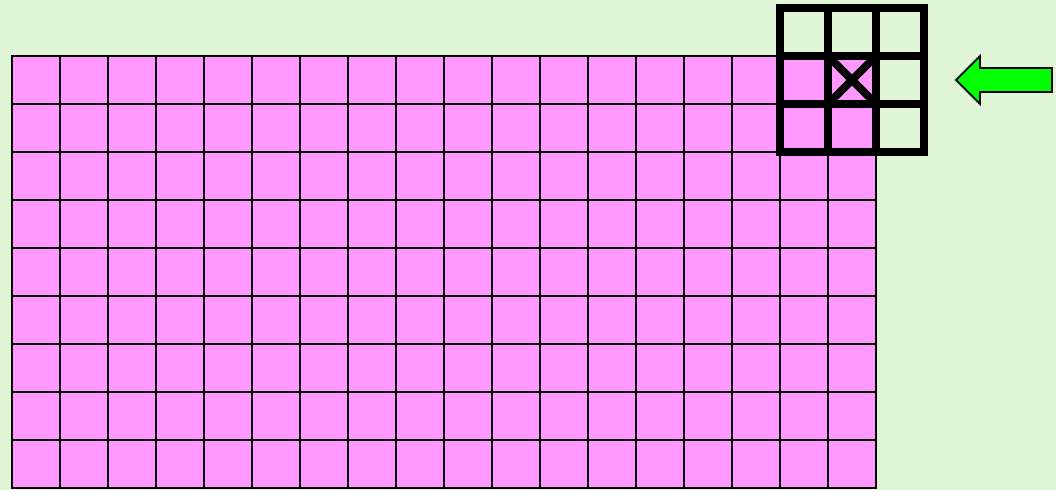


Filtered:



Replace  with the median of the values under the window.

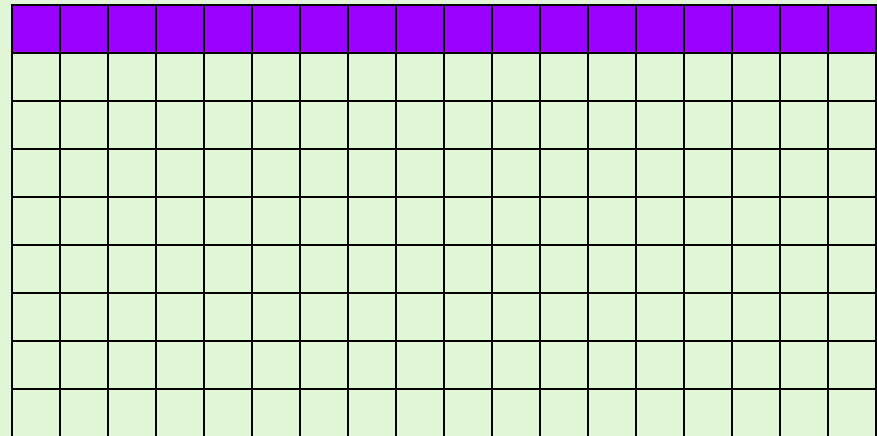
Original:



$$i = 1$$

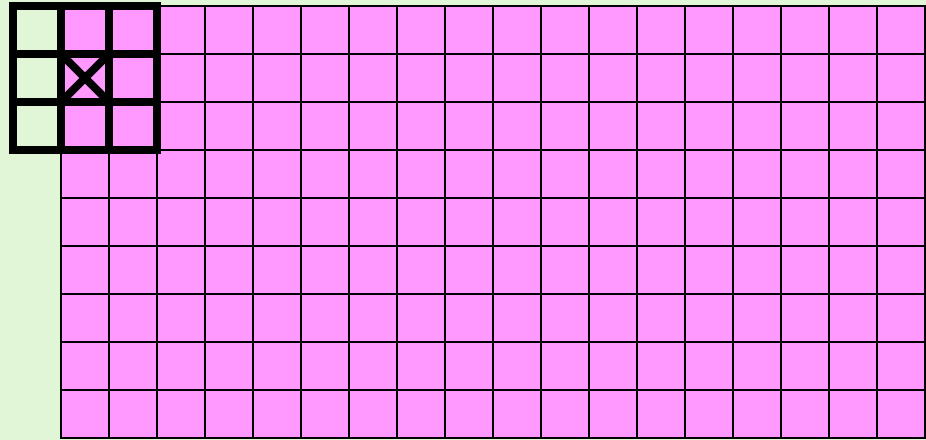
$$j = n$$

Filtered:



Replace  with the median of the values under the window.

Original:

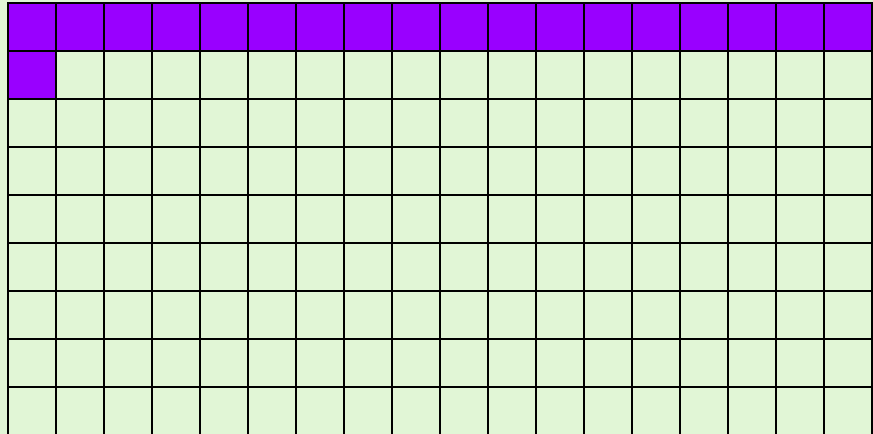


$$i = 2$$

$$j = 1$$

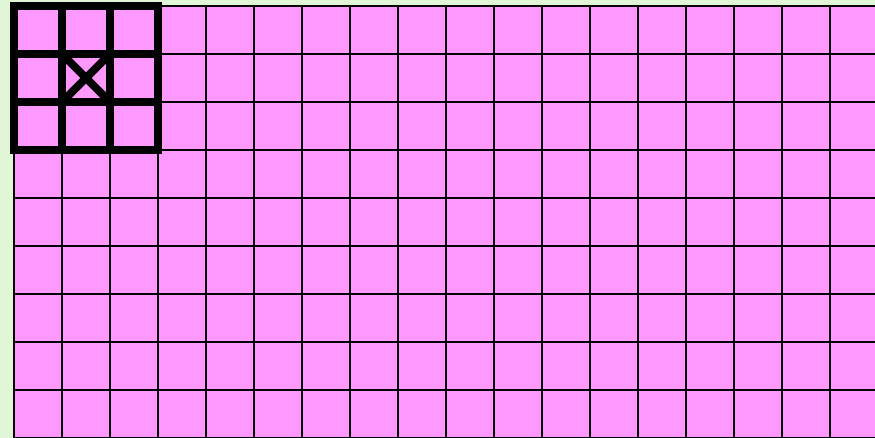


Filtered:



Replace  with the median of the values under the window.

Original:

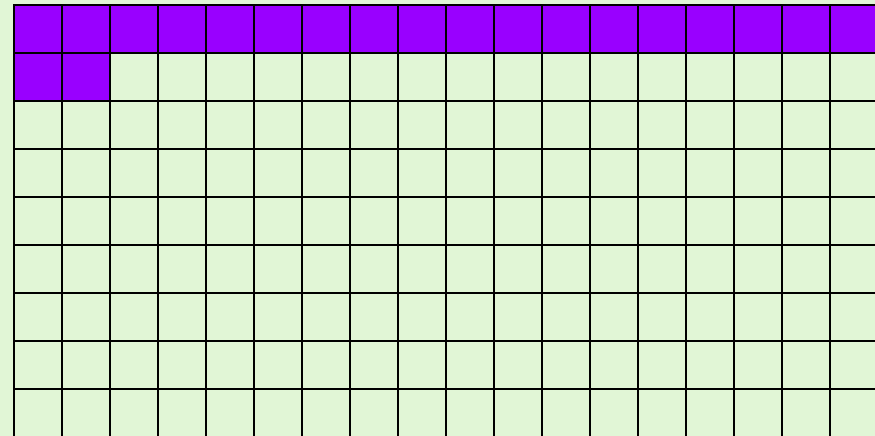



$$i = 2$$

$$j = 2$$

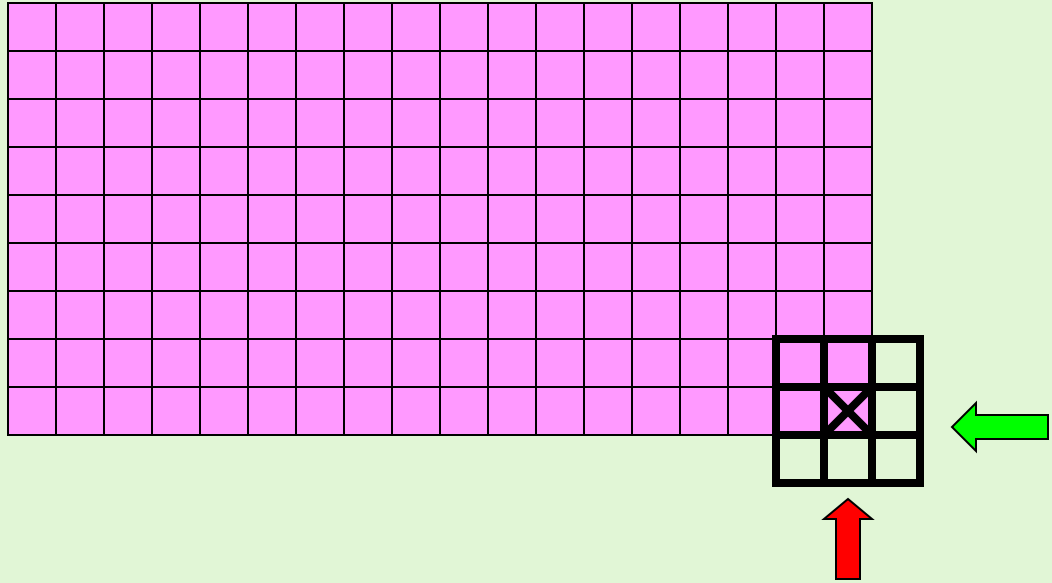


Filtered:



Replace  with the median of the values under the window.

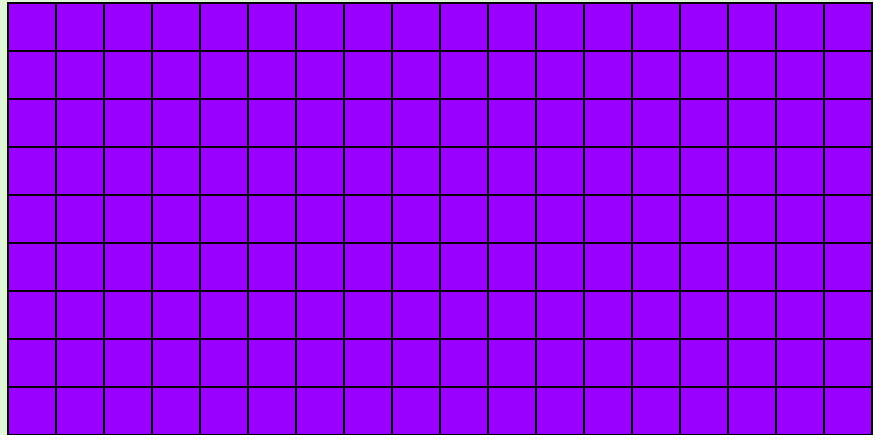
Original:



$$i = m$$

$$j = n$$

Filtered:



Replace  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 Medians


x :

21	89	36	28	19	88	43
----	----	----	----	----	----	----

x = sort(x)

x :

19	21	28	36	43	88	89
----	----	----	----	----	----	----



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) + x(m+1)) / 2**

Median of a 2D Array

```
function med = medVal(C)
[p,q] = size(C);
x = [];
for k=1:p
    x = [x C(k,:)];
end
```

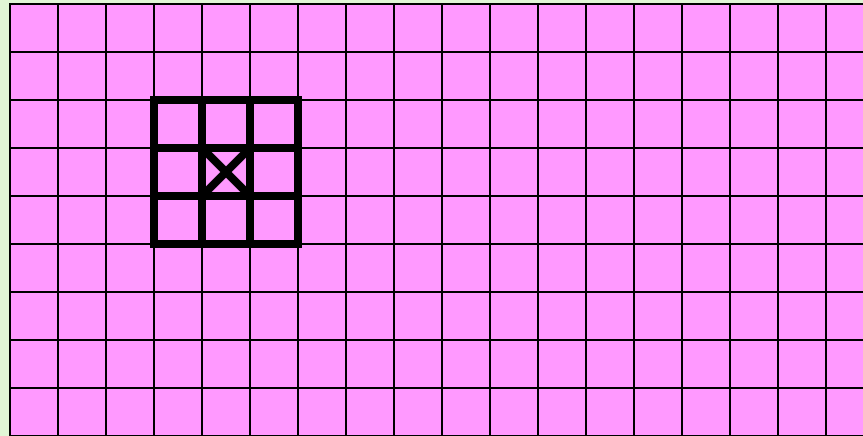
Compute median of x and assign to med.

Vectorized Median

```
function med = medVal(C)
```

```
med = median(C(:));
```

Back to Filtering...

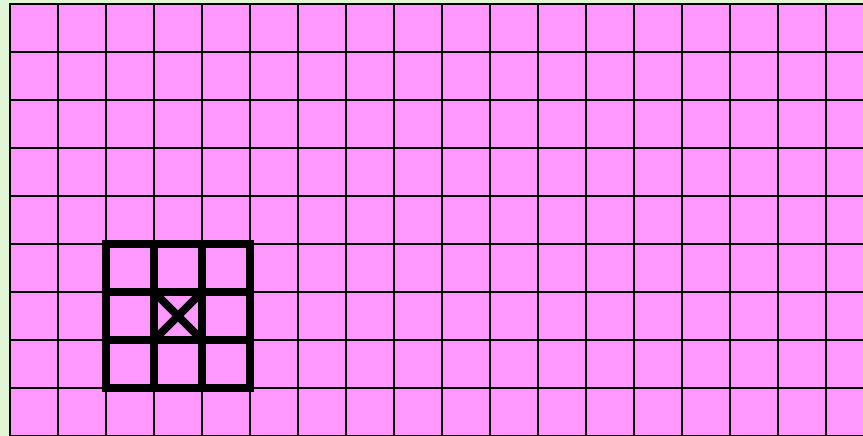


$m = 9$

$n = 18$

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

Window Inside...



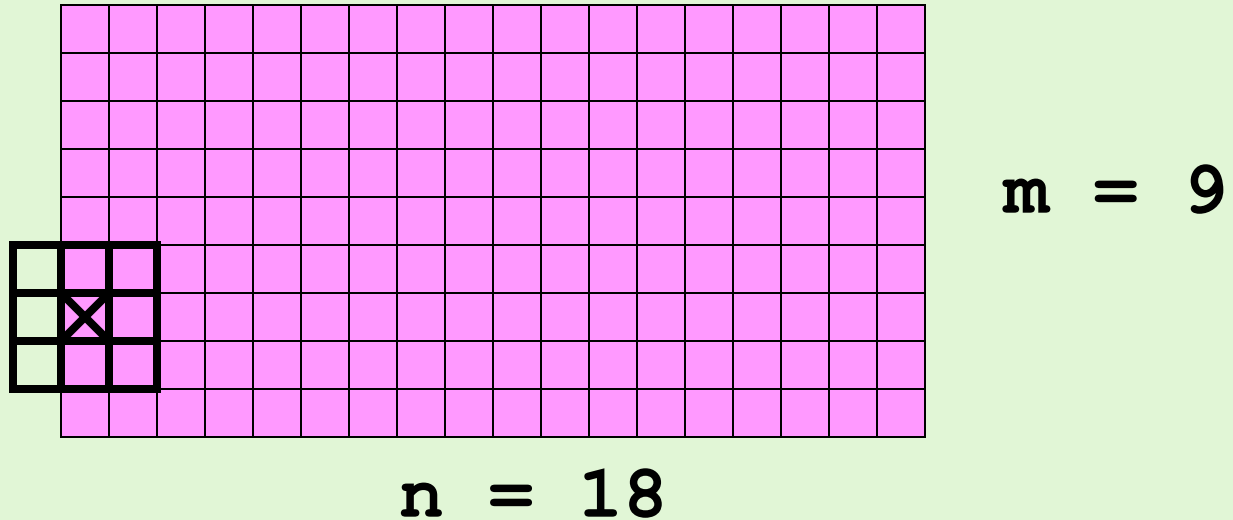
$$m = 9$$

$$n = 18$$

New gray value for pixel (7,4) =

`medVal (A (6 : 8 , 3 : 5))`

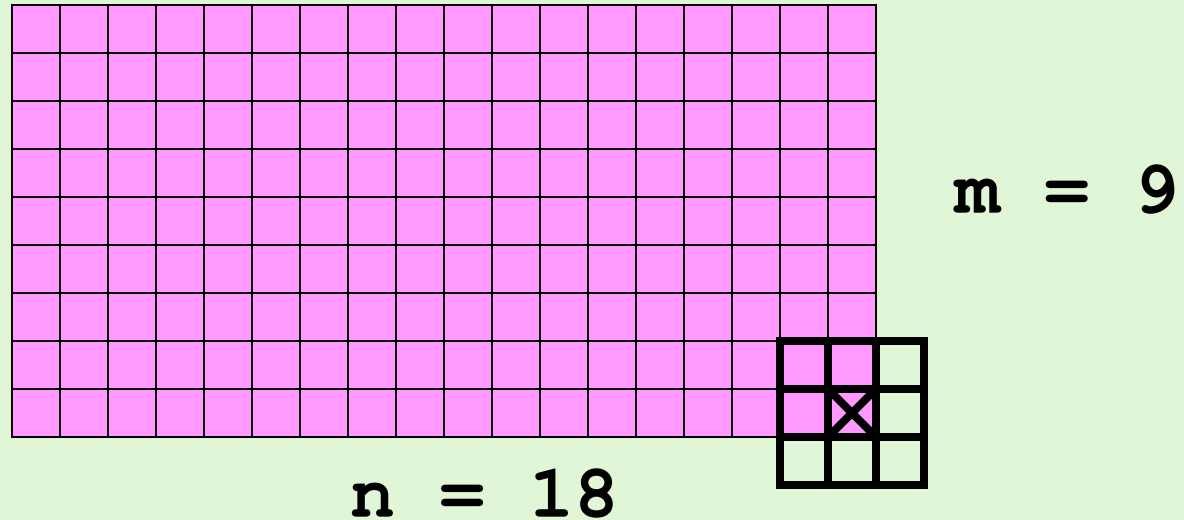
Window Partly Outside...



New gray value for pixel (7,1) =

`medVal (A (6 : 8 , 1 : 2))`

Window Partly Outside...



New gray value for pixel (9,18) =

`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
end
```


The Pixel (i,j) Neighborhood

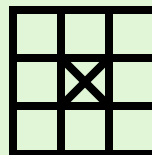
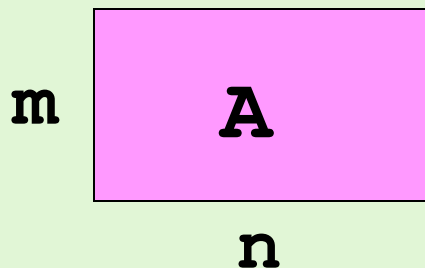
$$iMin = \max(1, i-r)$$

$$iMax = \min(m, i+r)$$

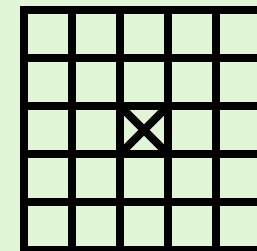
$$jMin = \max(1, j-r)$$

$$jMax = \min(n, j+r)$$

$$C = A(iMin:iMax, jMin:jMax)$$



$$r = 1$$



$$r = 2$$

B = medFilter(A)



Original

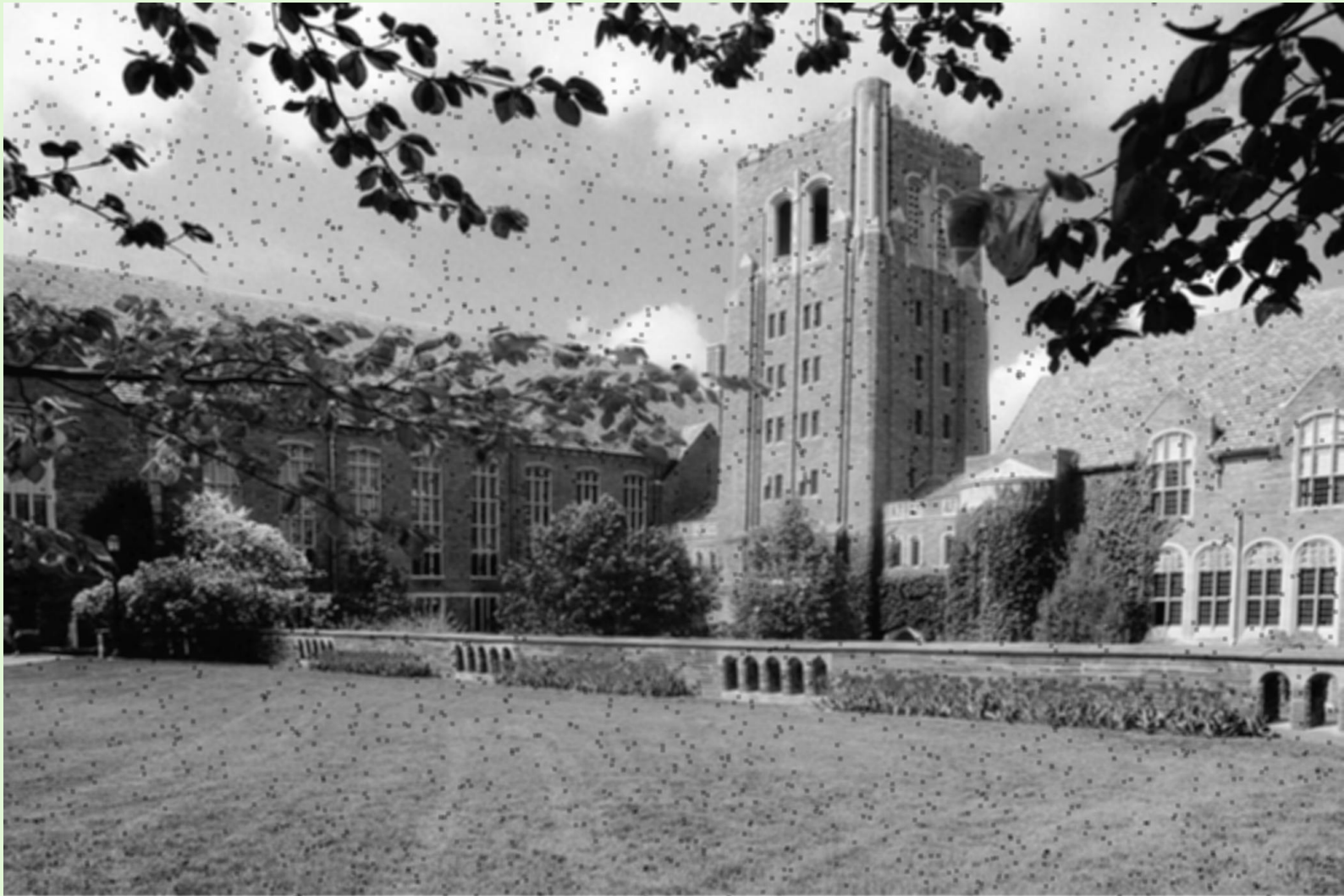


Cornell University Law School
Photograph by Cornell University Photography

What About Using the Mean instead of the Median?

Replace each gray value with the average gray value in the radius r neighborhood.

Mean Filter with $r = 3$



Cornell University Law School
Photography Cornell University Photography

Mean Filter with $r = 10$



Medians vs Means

A =

150	151	158	159	156
153	151	156	155	151
150	155	152	154	159
156	154	152	158	152
152	158	157	150	157

Median = 154 Mean = 154.2

Medians vs Means

A =

150	151	158	159	156
153	151	156	155	151
150	155	0	154	159
156	154	152	158	152
152	158	157	150	157

Median = 154 Mean = 148.2

Why it Fails

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

85 86
87 88

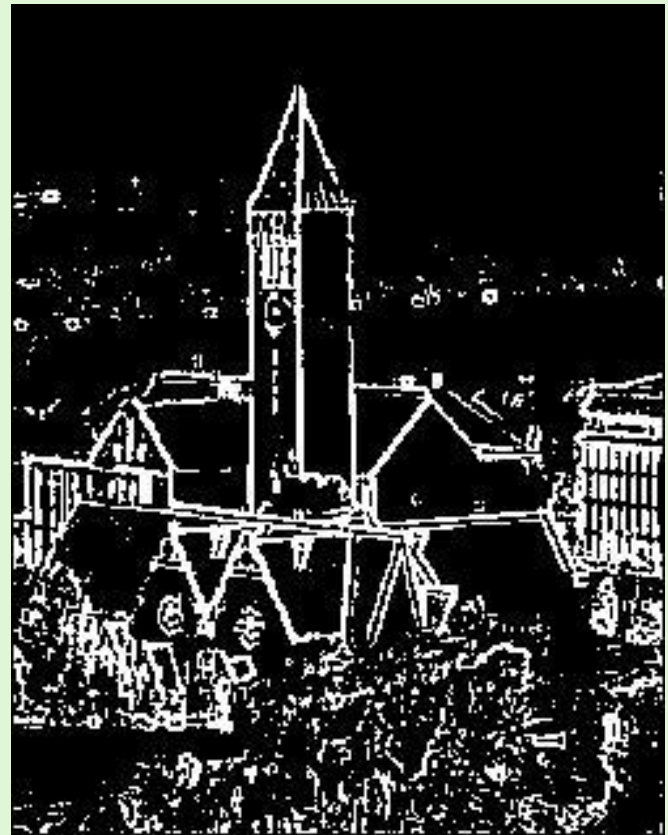
The mean does not capture representative values.

And Median Filters Leave Edges (Pretty Much) Alone

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

Inside the box, the 200's stay at 200 and the 100's stay at 100.

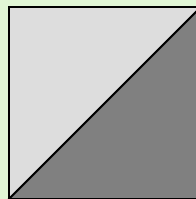
Finding Edges



What is an Edge?

Near an edge, grayness values change abruptly

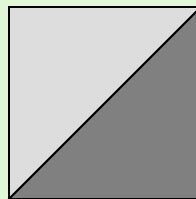
200	200	200	200	200	200
200	200	200	200	200	100
200	200	200	200	100	100
200	200	200	100	100	100
200	200	100	100	100	100
200	100	100	100	100	100



General plan for showing the edges in in image

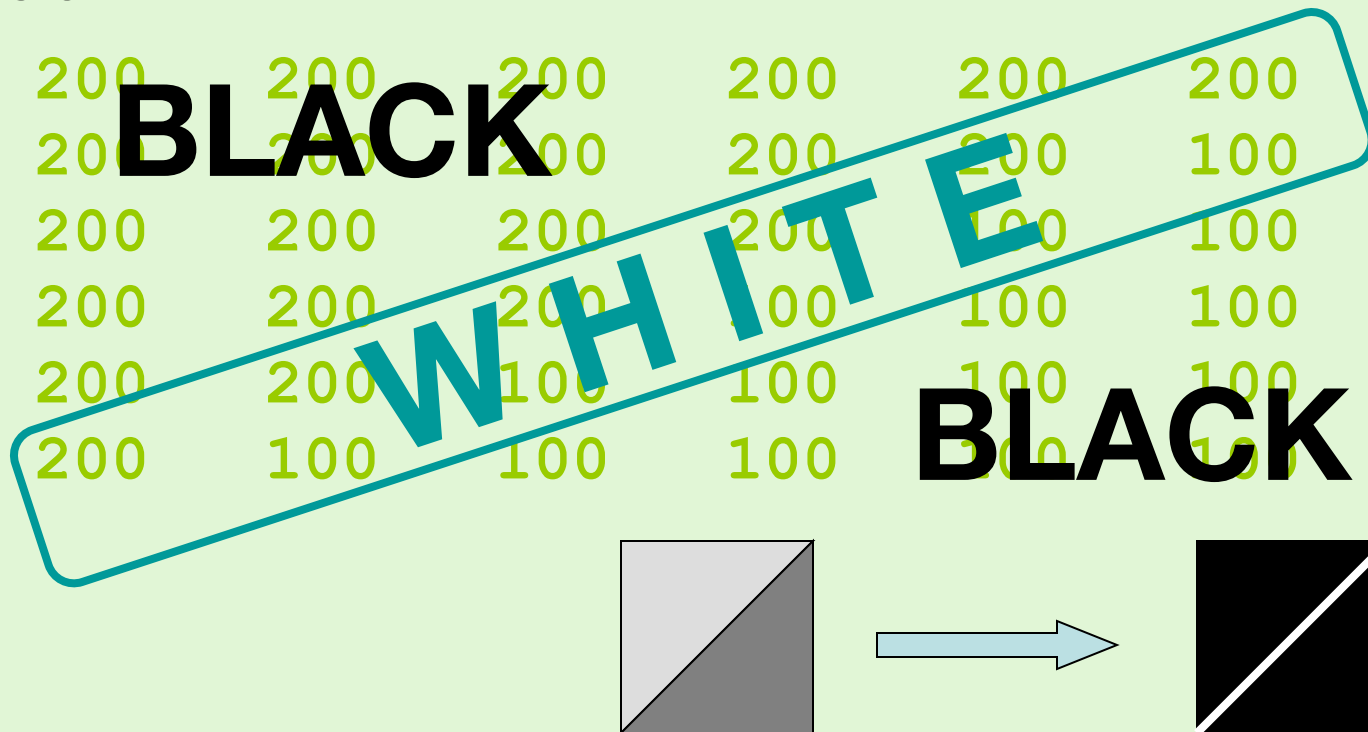
- Identify the “edge pixels”
- Highlight the edge pixels
 - make edge pixels white; make everything else black

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



The Rate-of-Change-Array

Suppose A is an image array with integer values between 0 and 255

$B(i,j)$ be the maximum difference between $A(i,j)$ and any of its eight neighbors.

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 value in

$$A(\max(1, i-1) : \min(m, i+1), \dots, \max(1, j-1) : \min(n, j+1)) - A(i, j)$$

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
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
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

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

    end
end
```

```

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)))));
    end
end
end

```

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

A

1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	90	90
1	1	1	90	90	90
1	1	90	90	90	90
1	1	90	90	90	90

threshold

```
if B(i,j) > tau
    B(i,j) = 255;
end
```

B(i,j)

0	0	0	0	0	0
0	0	0	89	89	89
0	0	89	89	0	0
0	89	89	0	0	0
0	89	0	0	0	0
0	89	0	0	0	0

0	0	0	0	0	0
0	0	0	255	255	255
0	0	255	255	0	0
0	255	255	0	0	0
0	255	0	0	0	0
0	255	0	0	0	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
        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
end

```

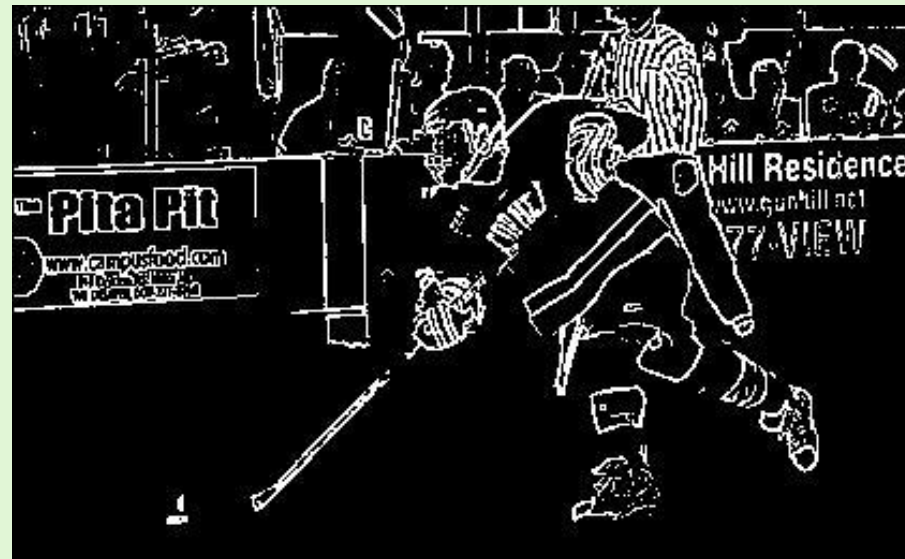
```

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
end
imwrite(B,jpgOut,'jpg')

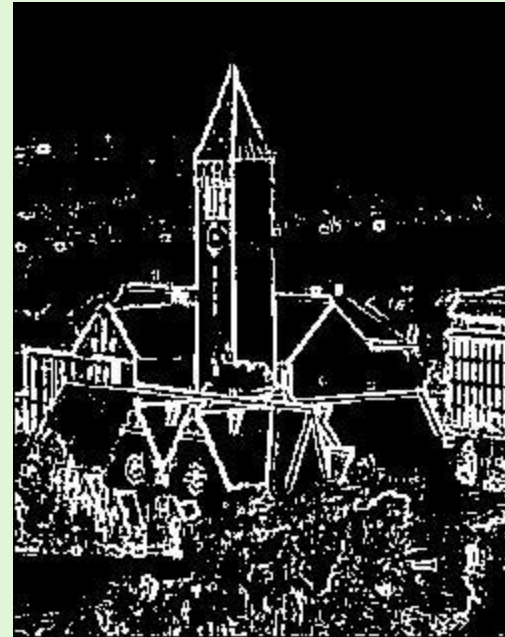
```


Threshold
= 40





Threshold = 20



Threshold = 30