

# Intro Math Problem Solving

## November 2

Lightening A Dark Image

Repairing A Damaged Image

Finding Edges in an Image

Homework #9

## Reference

Chapter 12, Section 4 of our textbook "Insight Through Computing" discusses the way black-and-white and color image data is stored and manipulated.

.

# Lighten a Dark Image



# Dark Image

Many photographs are taken in bad light, or have been badly developed.

Darkened images, in particular, are hard for the eye to interpret, whereas lighter shades are easier to "read".

For this photograph, we can display how the dark and light shades have been used by counting how many times each gray shade was used, between 0 and 255.

In other words, we want a bar plot or histogram.

# Convert Back and Forth

When working with image data, sometimes it will be easier to make a "double" copy of the uint8 array, and then use any numerical operations we want, and at the end, convert the whole array back to uint8 format:

```
A = double ( I ); % <- copy I
```

```
... % <- work on A
```

```
I = uint8 ( A ); % <- replace I with results
```

# Why So Dark?

A histogram of the gray levels will show how often each shade is used. However, MATLAB's `hist()` command will only accept "double" information, and it must be a vector:

```
I = imread ( 'snap.png' );
```

```
G = double ( I );           <- Convert to double
```

```
[ m, n ] = size ( G );
```

```
gvec = reshape ( G, m * n, 1 ); <- Column vector
```

```
hist ( gvec, 256 );         <- histogram
```

# RESHAPE: Matrix -> Vector

MATLAB's histogram plotting command `hist(g,n)` expects the input to be a 1D vector of type "double", but we want to histogram all the data in an image (a 2D matrix of type `uint8`).

We have seen before how to use the `double()` function to convert the `uint8` data to the right type.

We introduce MATLAB's `reshape()` command:

```
a_new = reshape ( a_old, m_new, n_new );
```

which copies data in the  $M \times N$  object `A_OLD` into the  $M\_NEW \times N\_NEW$  object `A_NEW`. We can easily copy the matrix data into a vector shape.

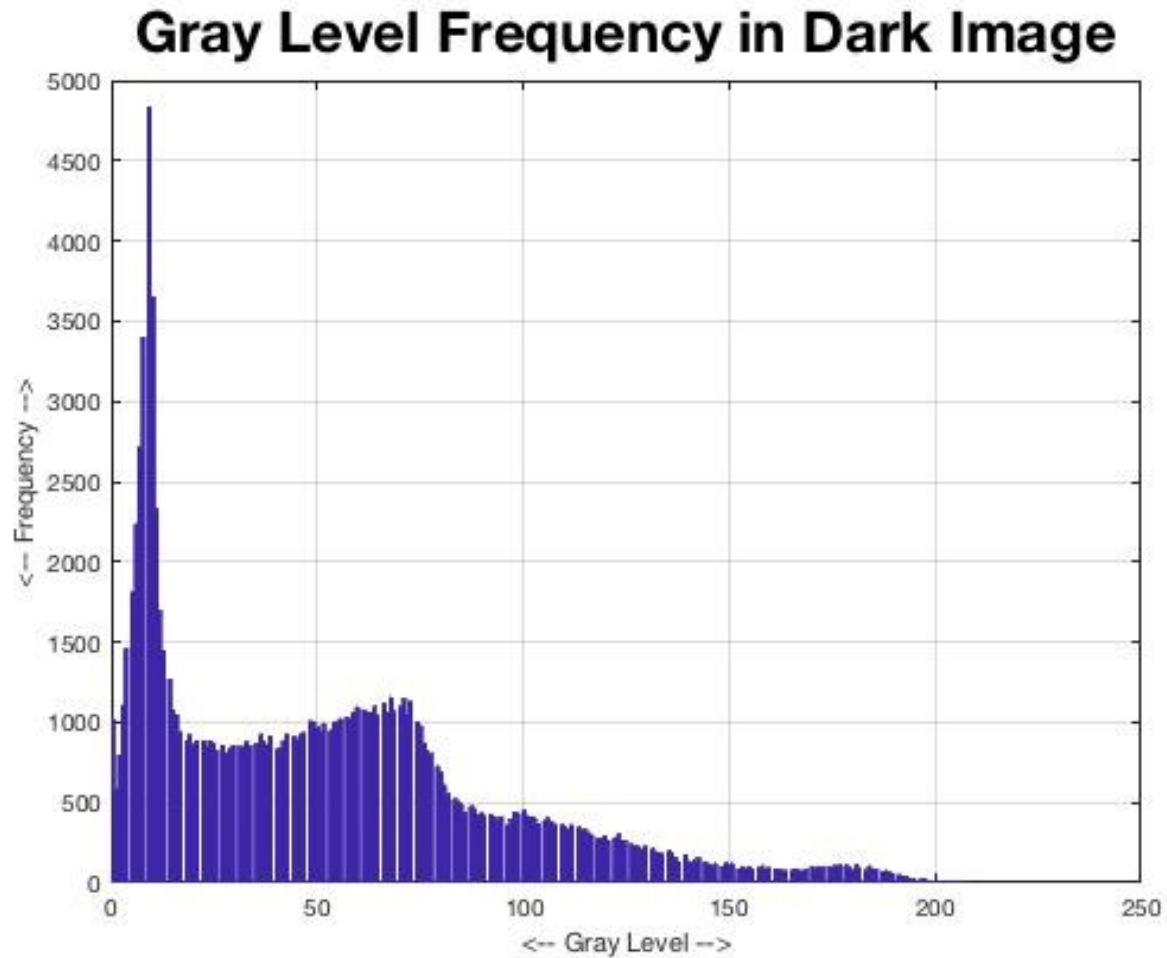
# RESHAPE Examples

```
A = [ 11, 12, 13;  
      21, 22, 23 ];      <- 2x3 matrix  
b = reshape ( A, 1, 6 ); <- 1x6 row vector  
b <- [ 11, 21, 12, 22, 13, 23 ];
```

```
C = reshape ( A, 3, 2 ); <- 3x2 matrix  
C <- [ 11, 22;  
      21, 13;  
      12, 23];
```



# Heavy use of Darkest Grays



# Lighten the Levels

We are hardly using the lighter gray levels in the picture. Most of the picture information is presented in dark grays. Our eye has trouble seeing the details. A better picture would spread the information out.

Here are two ways we can try:

1) Double all grays between 0 and 127;

Grays above 127 become 255;

2)  $\text{Gray} \rightarrow \sqrt{\text{Gray}/255} * \text{Gray}$ ;

## Doubling

```
I = imread ( 'snap.png' );  
i1 = ( I < 128 );  
i2 = ( 128 <= I );  
I2(i1) = 2 * I(i1);    % <- Double the darks  
I2(i2) = 255;         % <- Lights -> White  
imshow ( I2 );
```

# Lightening by Doubling



# Sqrt

```
I = imread ( 'snap.png' );  
G = double ( I );  
G = sqrt ( G / 255 ) * 255;  
I3 = uint8 ( G );  
imshow ( I3 );
```

# Lighten by Sqrt



# Compare Double vs Sqrt



# Dark, Medium, Light

We can separate the dark, medium and light sectors:

```
I = imread ( 'snap.png' );  
i1 = ( I < 25 );  
i2 = ( 25 <= I & I < 100 );  
i3 = ( 100 <= I );
```

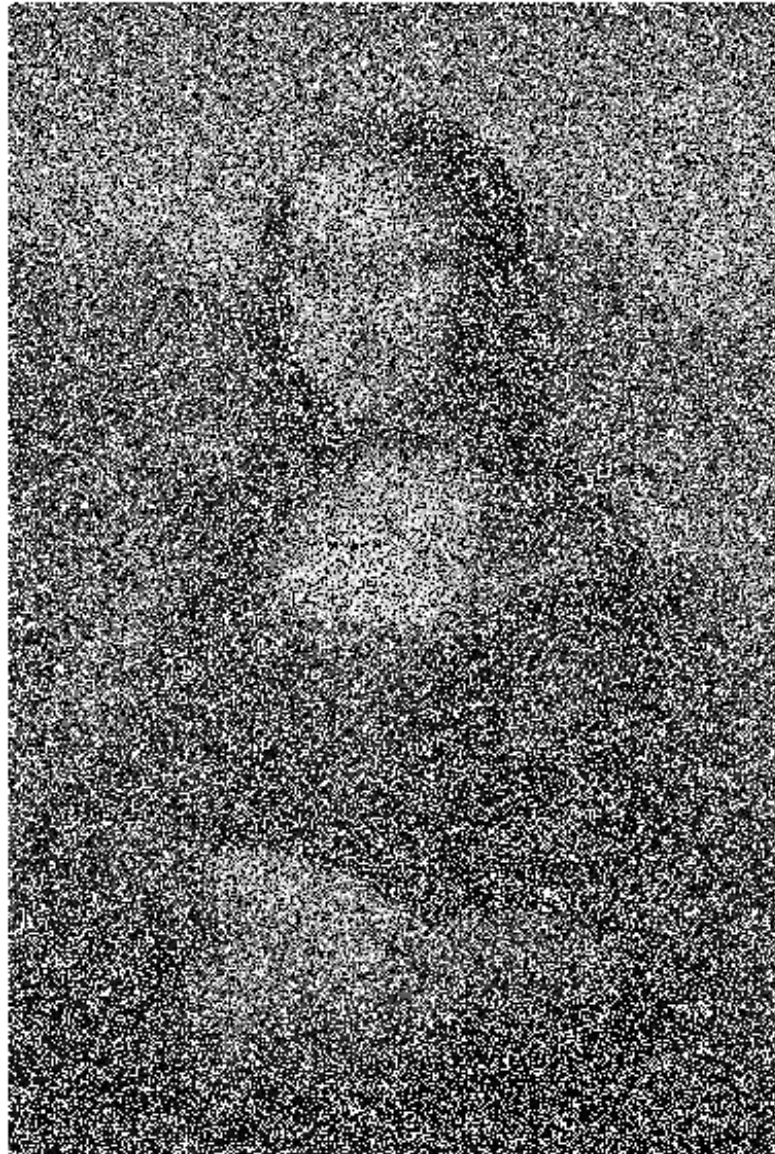
```
I2 = I;  
I2(i1) = 0;  
I2(i2) = 127;  
I2(i3) = 255;  
imshow ( I2 );
```



3 shades: Black, Medium Gray, White

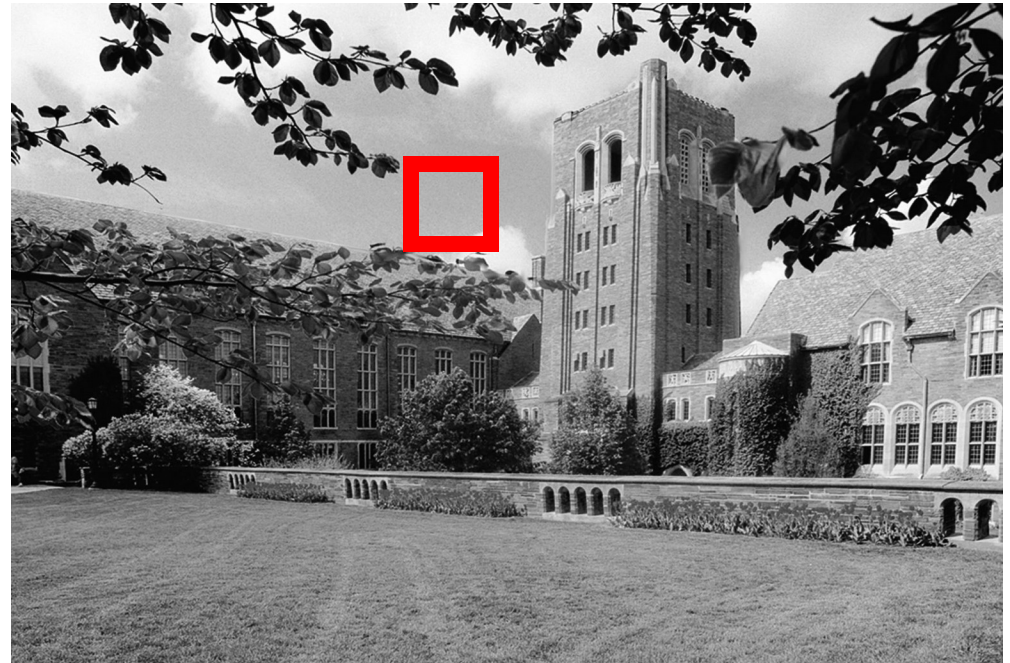


# Repairing Damaged Images



# Just a Bunch of Numbers

1458-by-2084

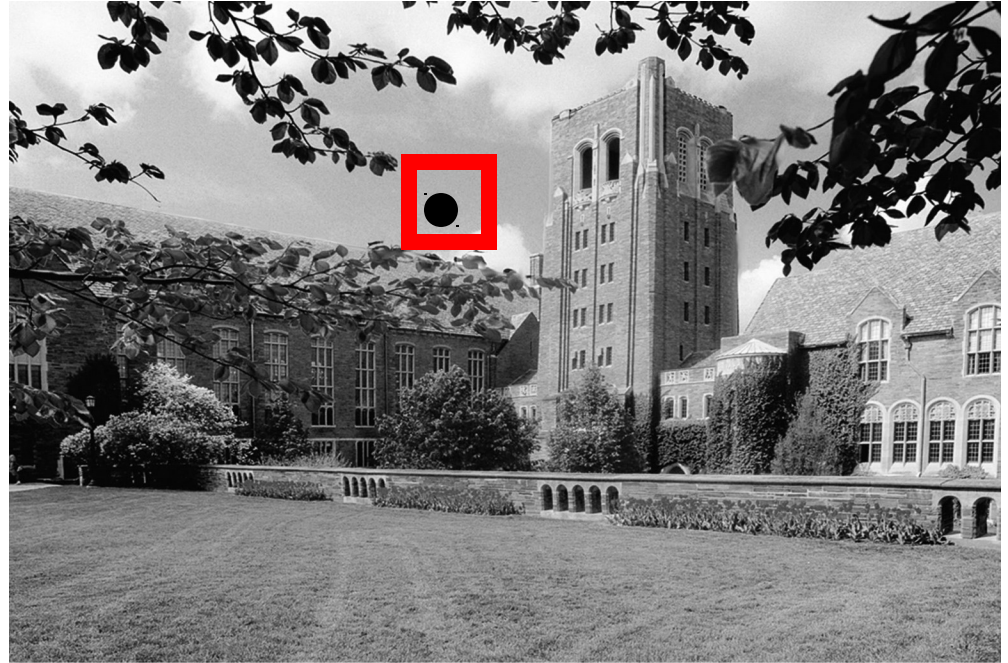


Cornell University Law School  
Photograph by Cornell University Photography

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

# Dirt!

1458-by-2084

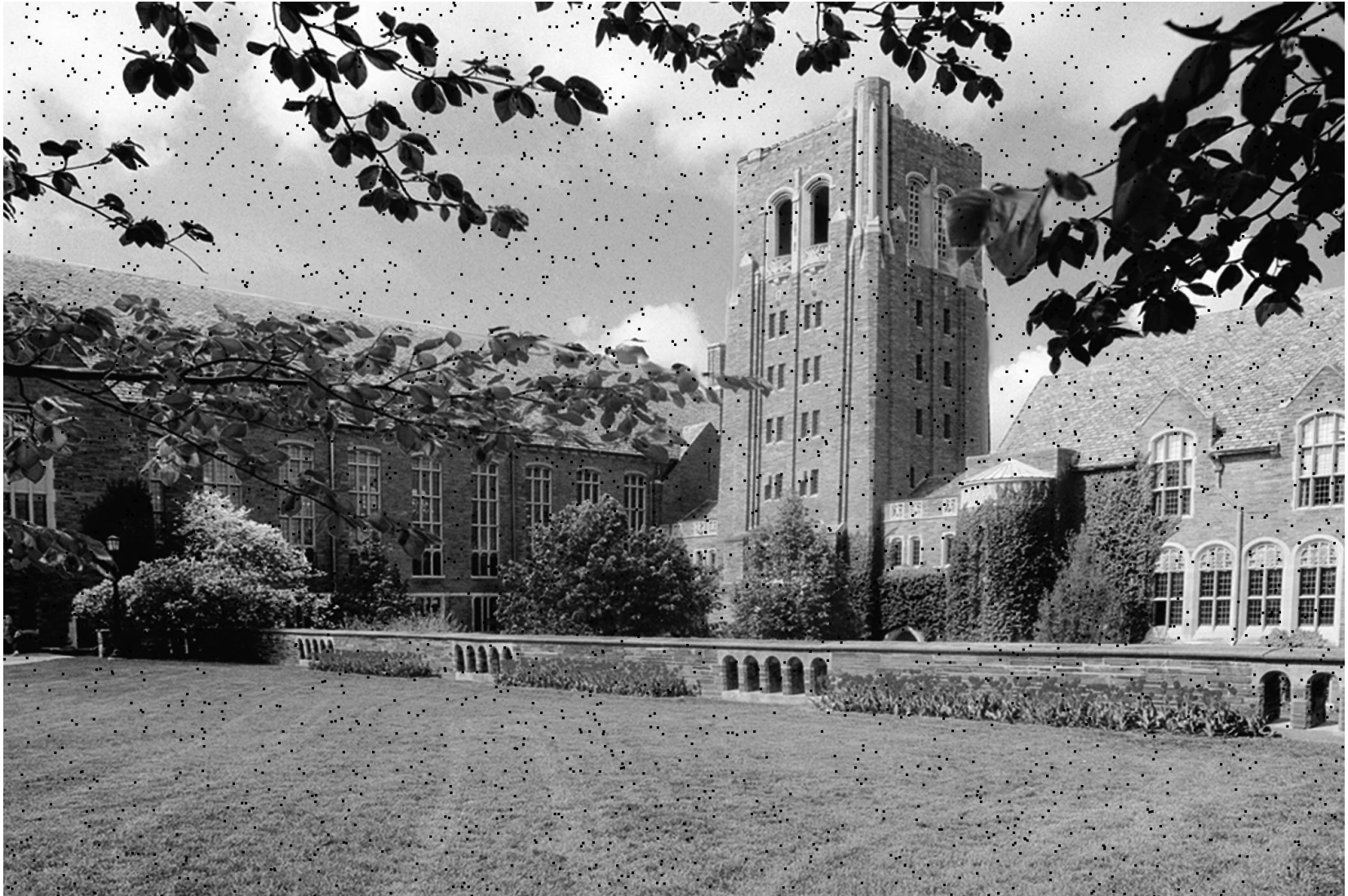


Cornell University Law School  
Photograph by Cornell University Photography

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

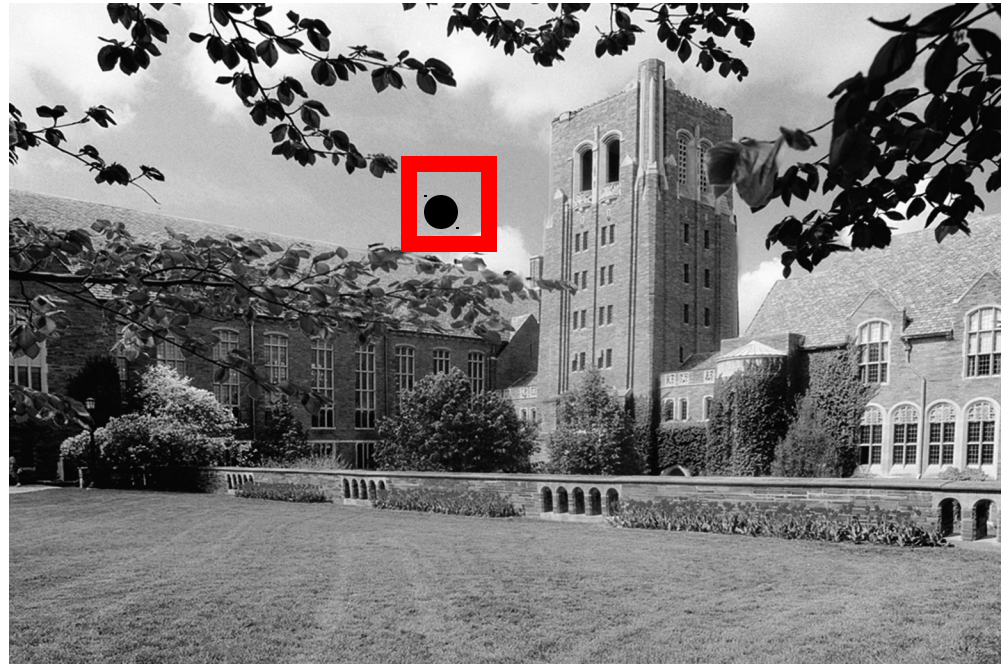
# Can We Filter Out the "Noise"?



Cornell University Law School  
Photograph by Cornell University Photography

# Idea

1458-by-2084



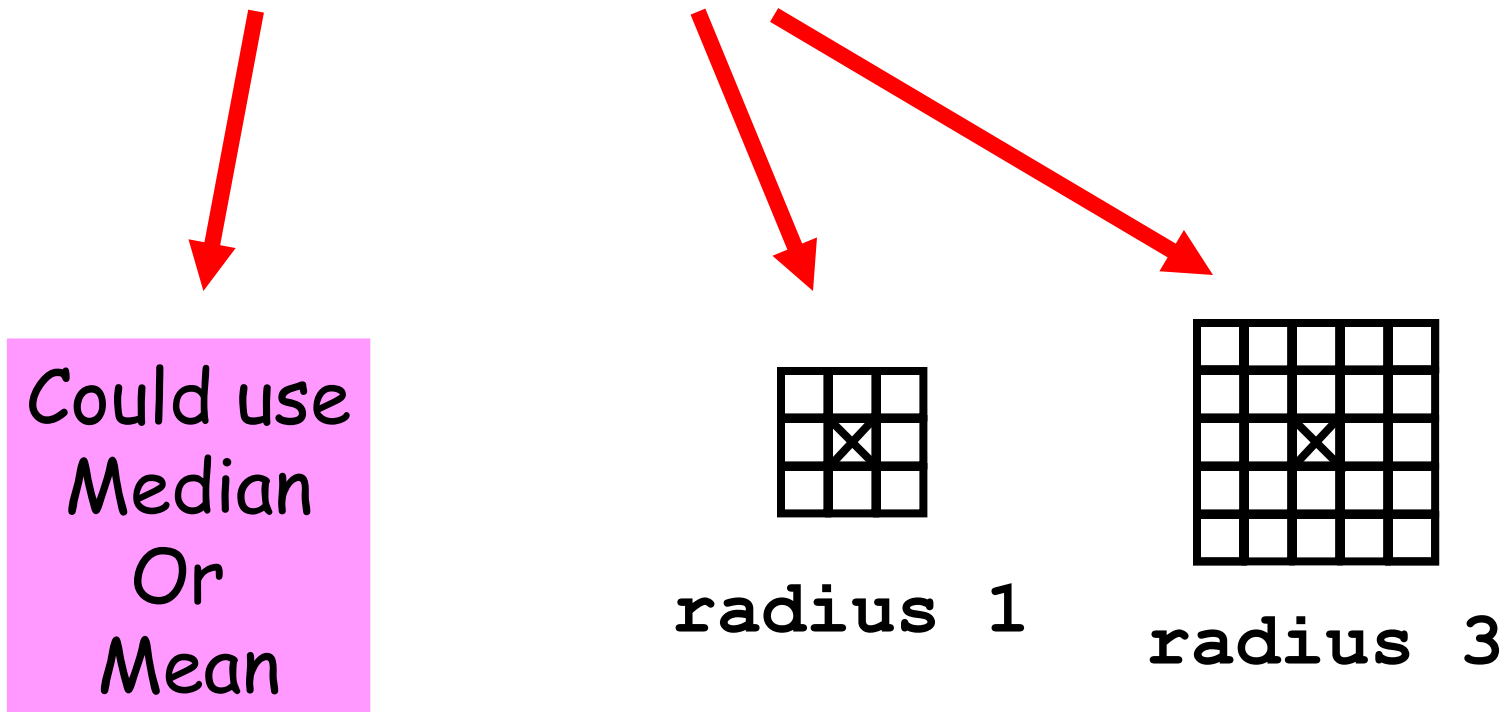
Cornell University Law School  
Photograph by Cornell University Photography

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"



We'll look at "Median Filtering" first...

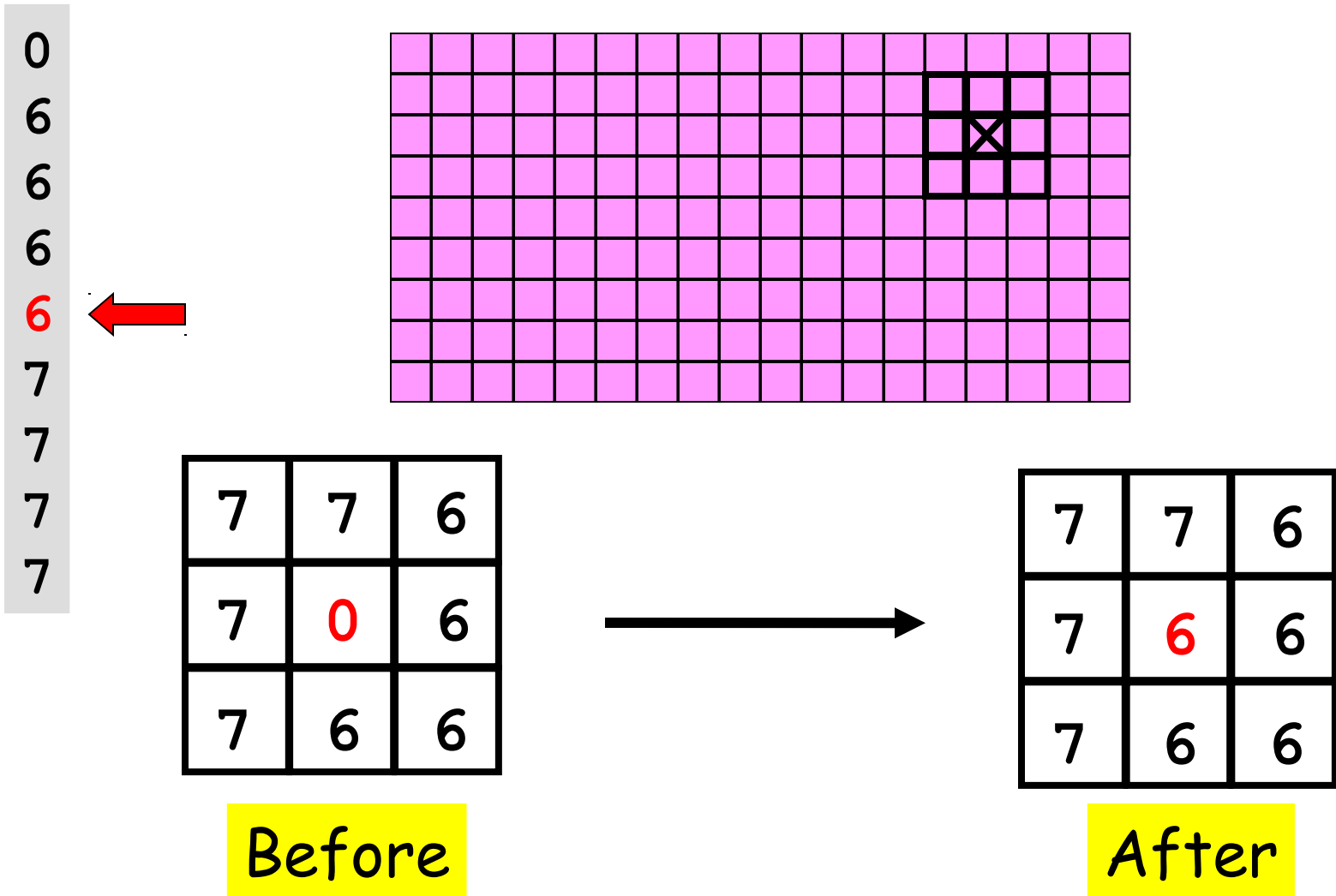
# Median Filtering

Visit each pixel.

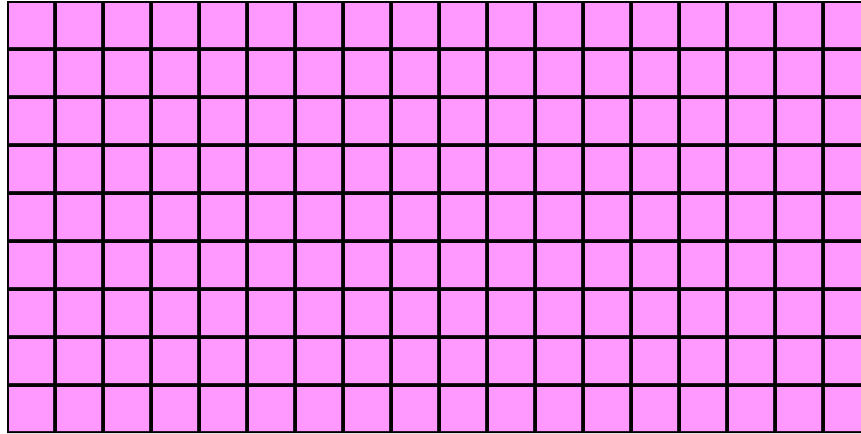
Replace its gray value by the median of the gray values in the "neighborhood".



# Using a radius 1 "Neighborhood"



# How to Visit Every Pixel



$m = 9$

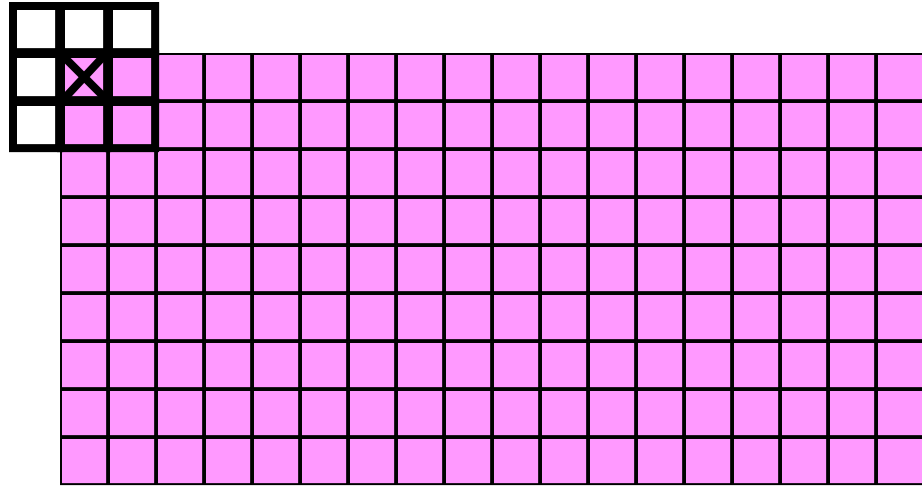
$n = 18$

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

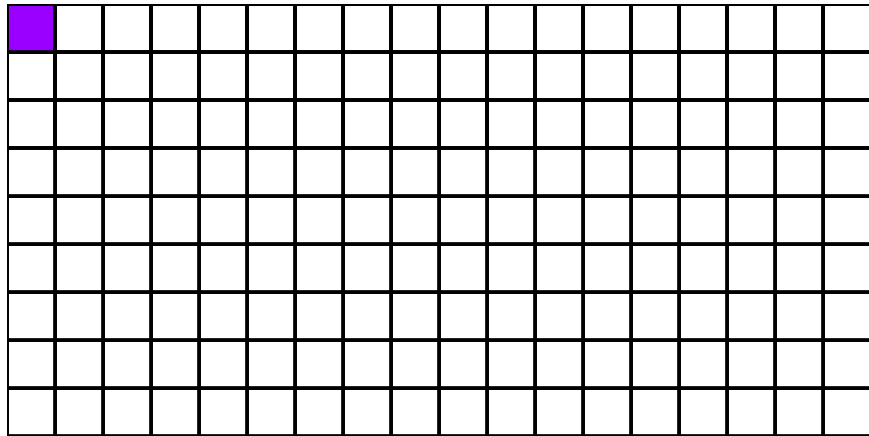
Original:


$$i = 1$$

$$j = 1$$



Filtered:

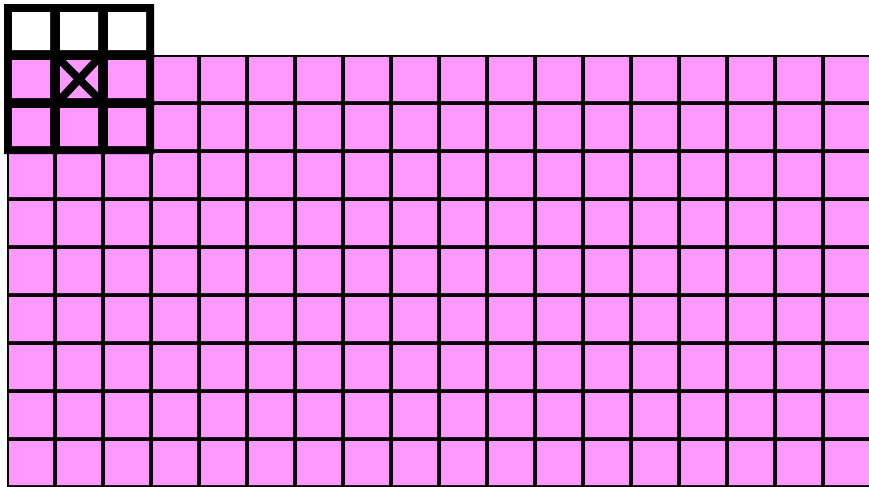


Replace  with the median of the values under the window.

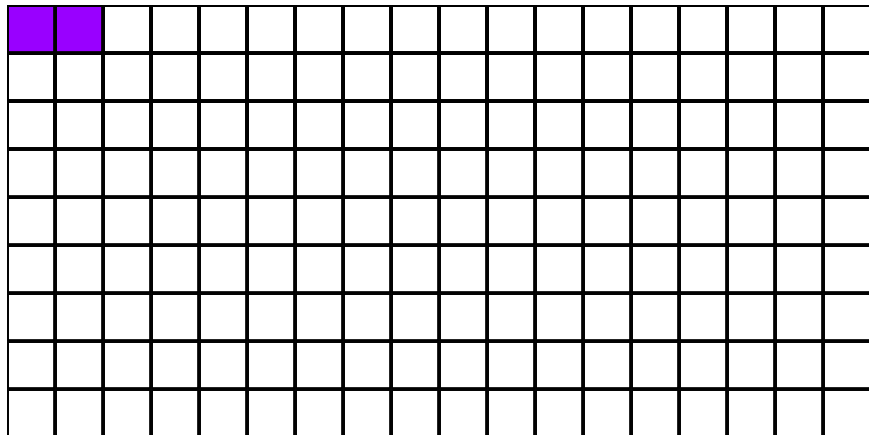
Original:


$$i = 1$$

$$j = 2$$



Filtered:

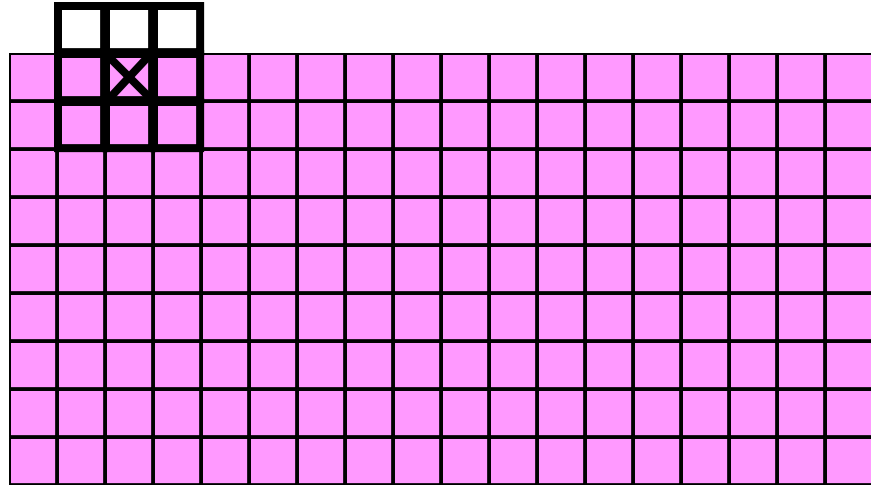


Replace  with the median of the values under the window.

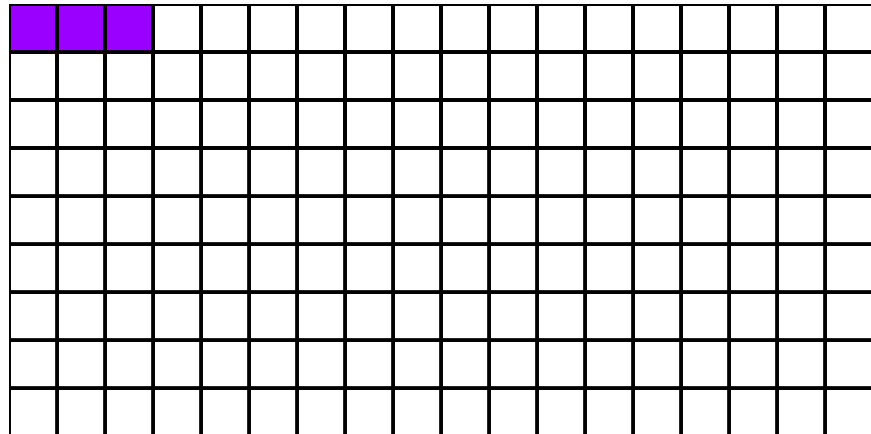
Original:

$$i = 1$$

$$j = 3$$



Filtered:

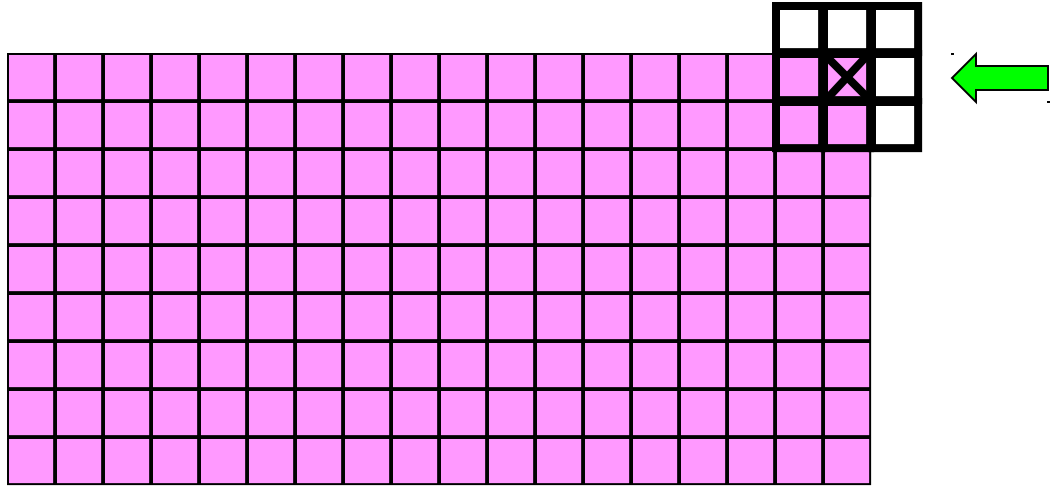


Replace  with the median of the values under the window.

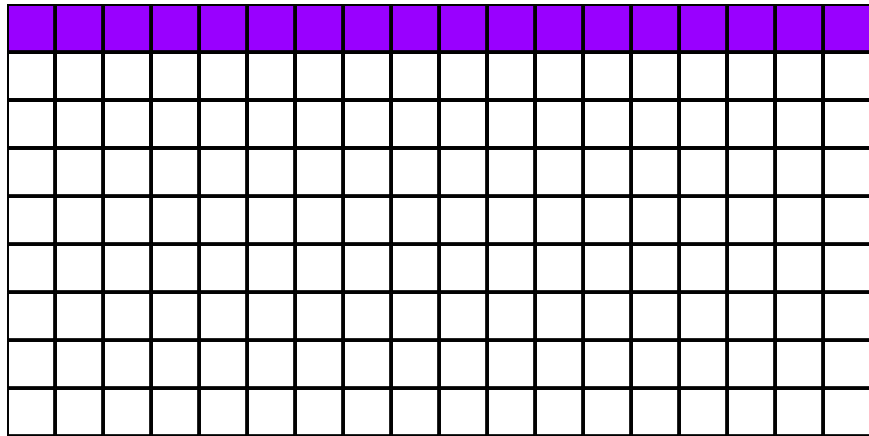
Original:


$$i = 1$$

$$j = n$$



Filtered:

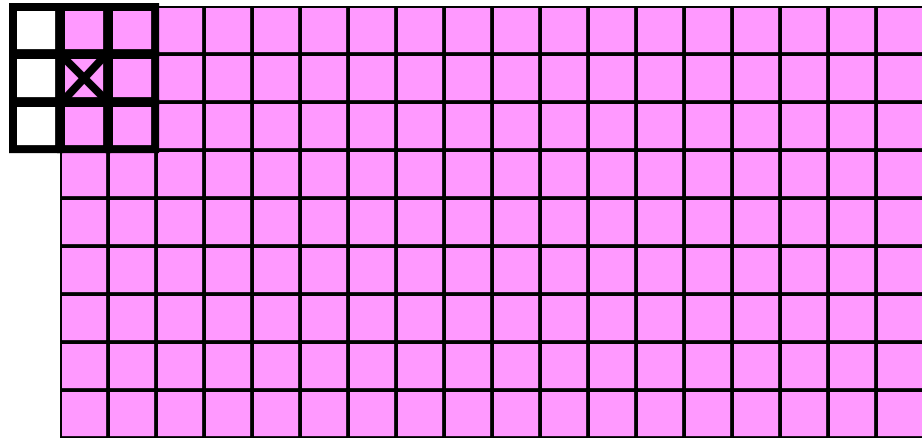


Replace  with the median of the values under the window.

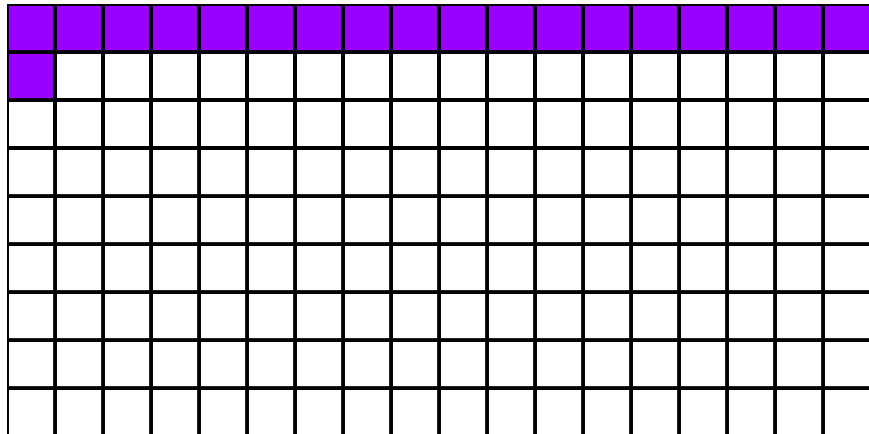
Original:


$$i = 2$$

$$j = 1$$



Filtered:

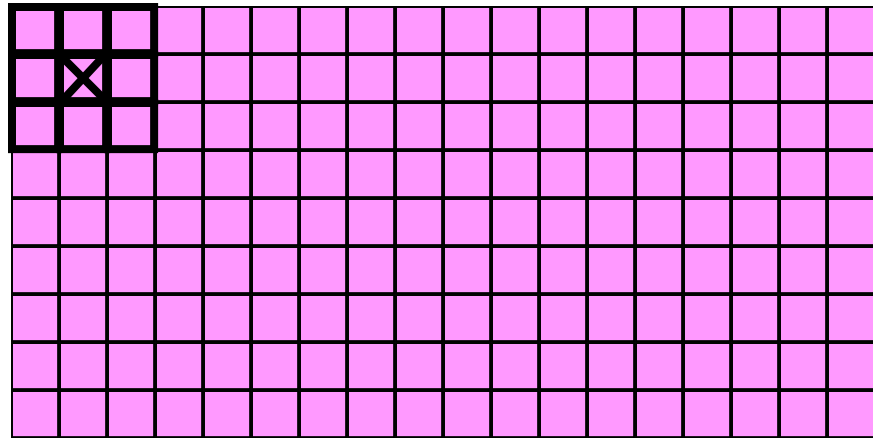


Replace  with the median of the values under the window.

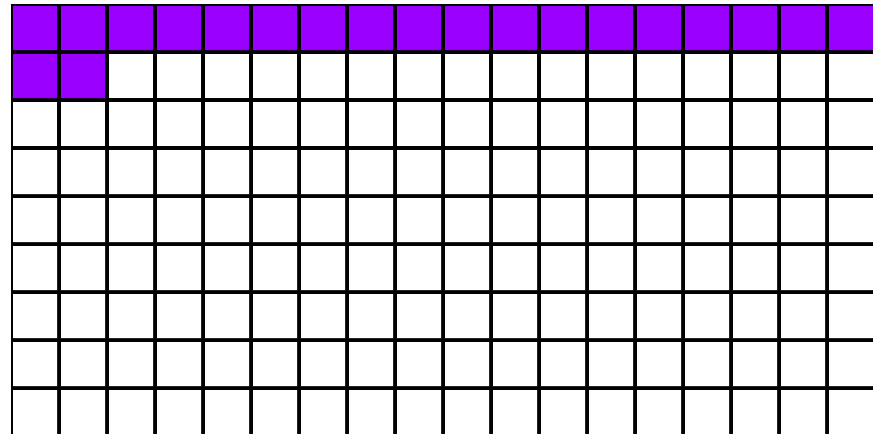
Original:

$$i = 2$$

$$j = 2$$



Filtered:



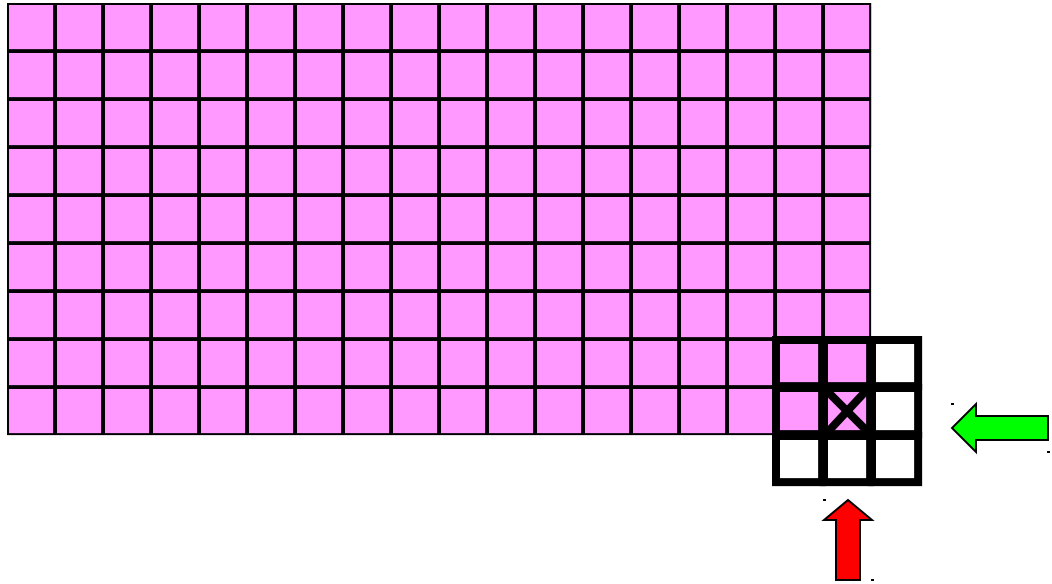
Replace  with the median of the values under the window.



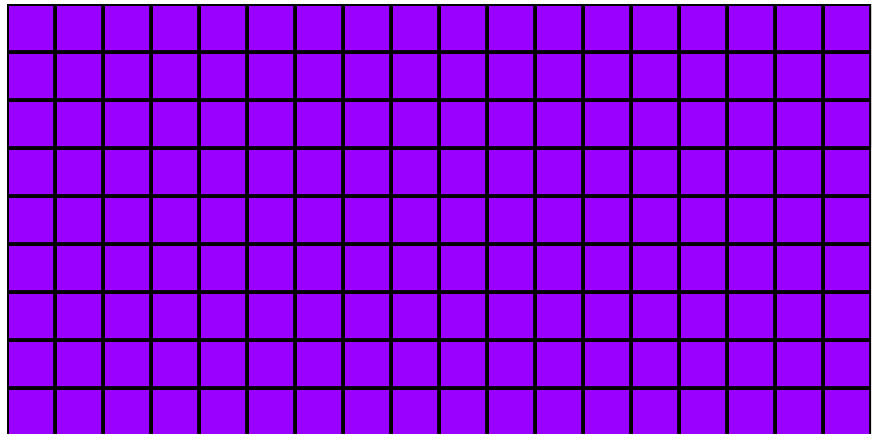
Original:


$$i = m$$

$$j = n$$



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 C

```
function med = medVal( C )
    [p,q] = size(C);
    x = reshape ( C, 1, p*q );

    % x = sort ( x );
    % middle = ceil ( ( p * q ) / 2 )
    % med = x(middle);

    med = median ( x );

    return
end
```

# 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**

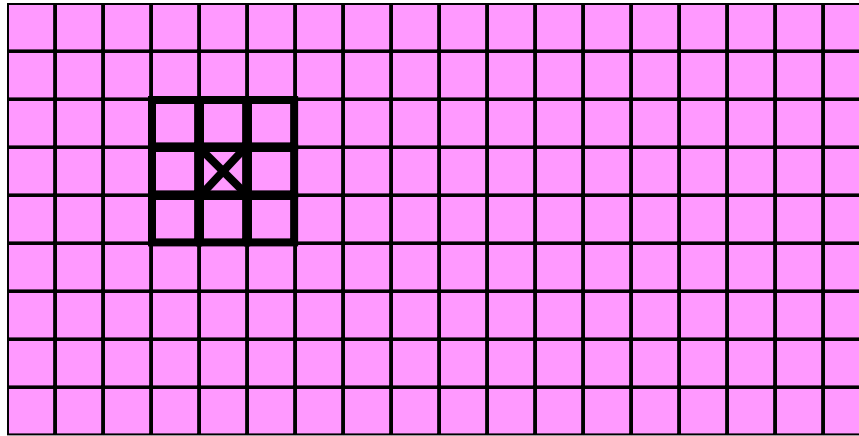
# Medians vs Means

**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**

# Back to Filtering...

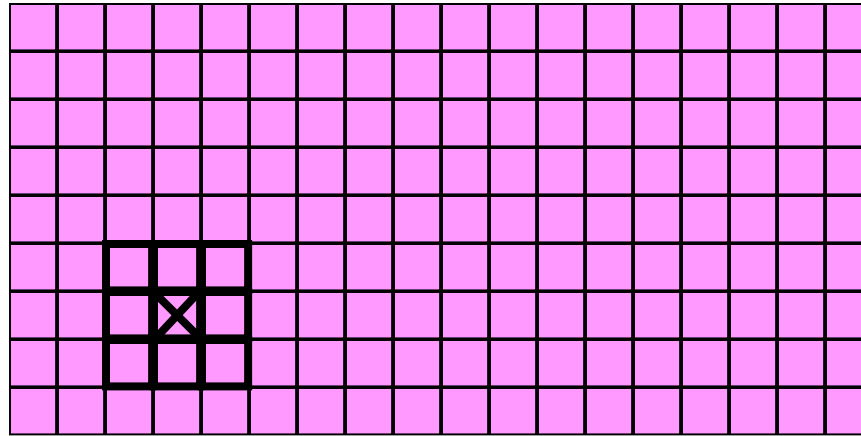


$m = 9$

$n = 18$

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

# Window Inside...



$$m = 9$$

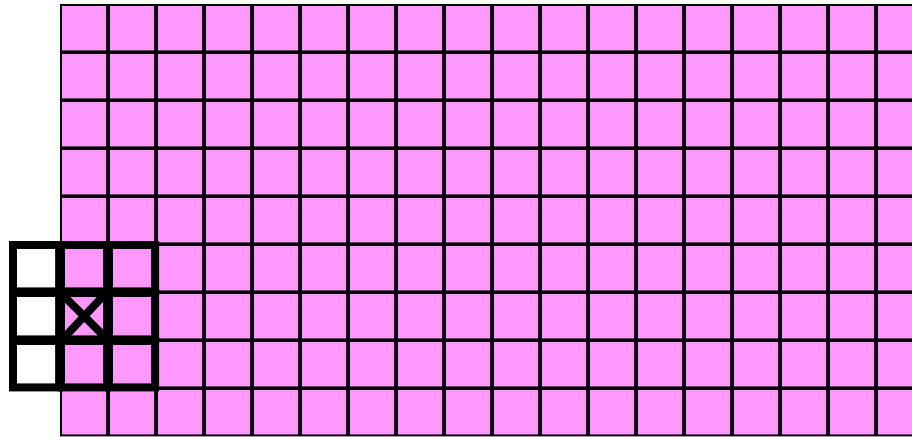
$$n = 18$$

New gray value for pixel (7,4) =

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



# Window Partly Outside...



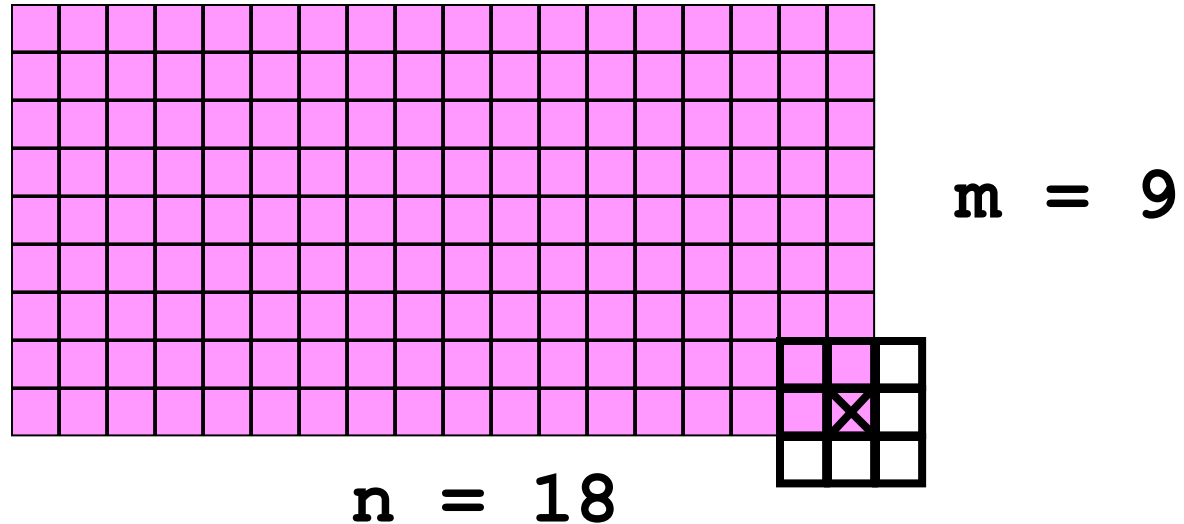
$$m = 9$$

$$n = 18$$

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 = A;
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

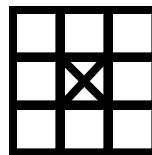
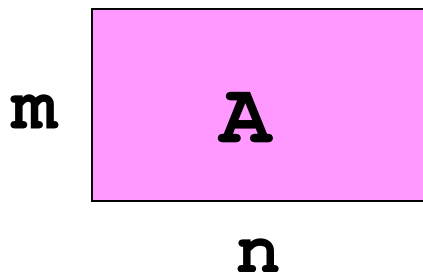
$$iMin = \max(1, i-r)$$

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

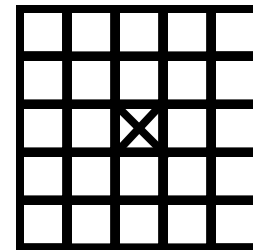
$$jMin = \max(1, j-r)$$

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

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



$$r = 1$$



$$r = 2$$

```
I = imread ( 'glassware_noisy.png' )
```



$I_2 = \text{MedianFilter}(I)$ ;



## What About Using the Mean instead of the Median?

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

Bits of noise don't get removed, just smeared around. They are still visible.

$I_3 = \text{MeanFilter}(I)$





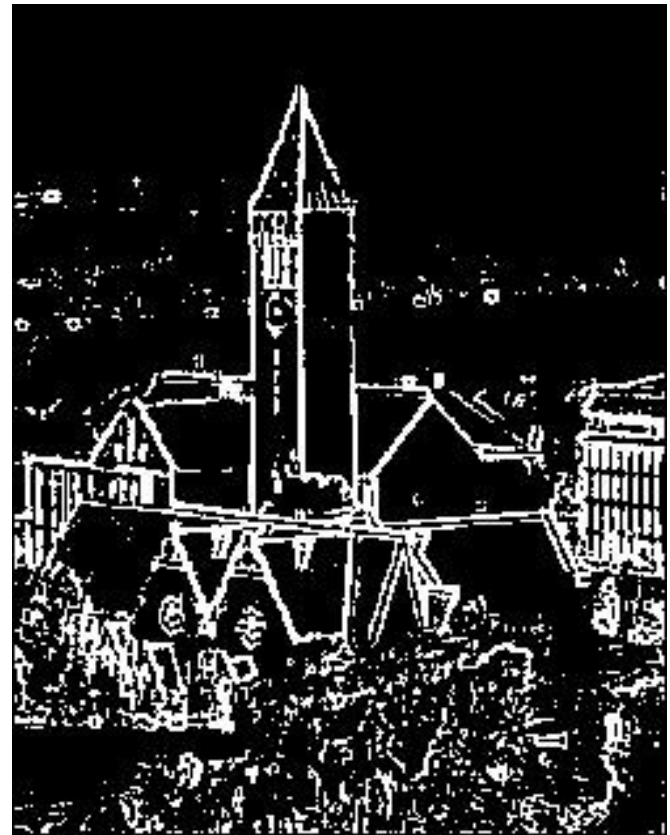
# Why it Fails

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.

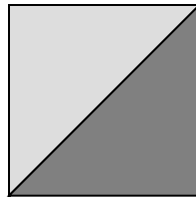
# Finding Edges



# What is an Edge?

Near an edge, grayness values change abruptly

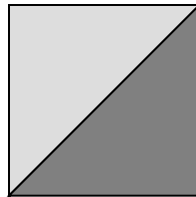
200	200	200	200	200	200
200	200	200	200	200	100
200	200	200	200	100	100
200	200	200	100	100	100
200	200	100	100	100	100
200	100	100	100	100	100



## General plan for showing the edges in in image

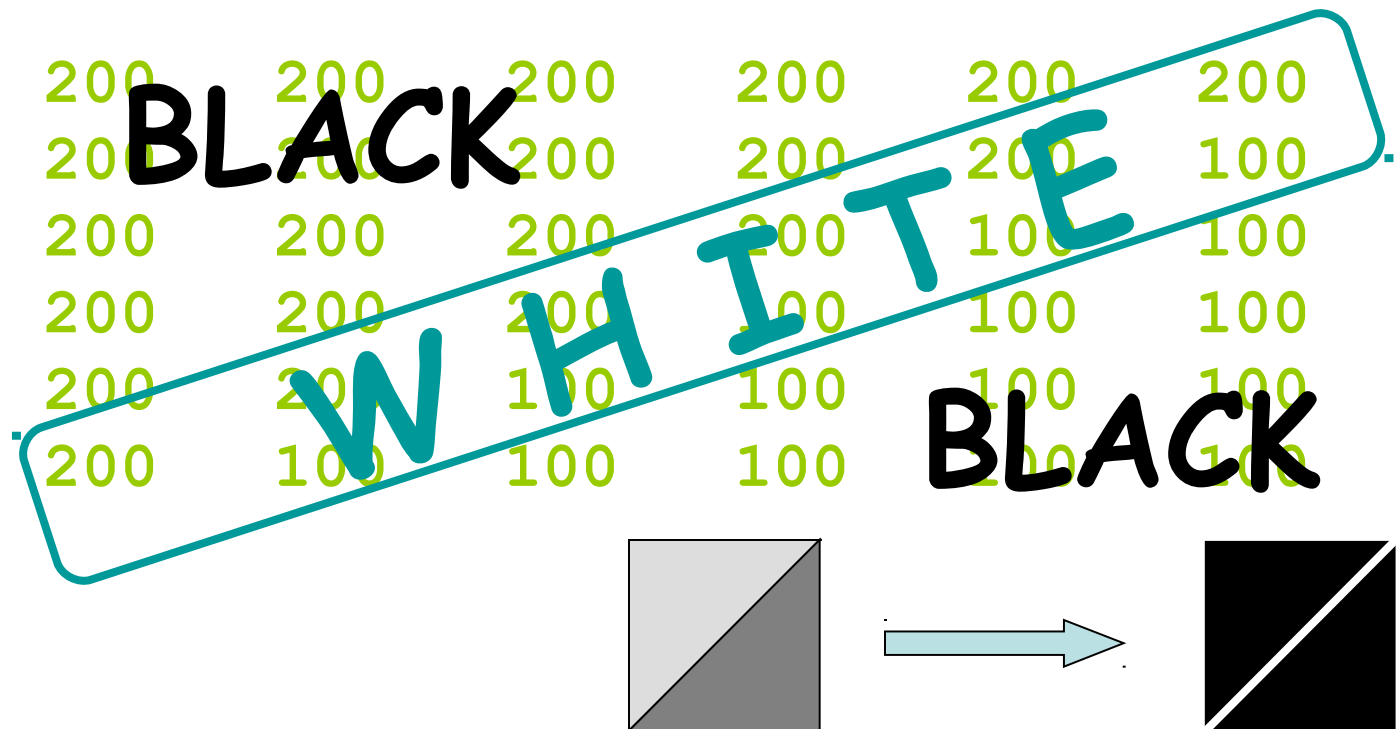
- Identify the "edge pixels"
- Highlight the edge pixels
  - make edge pixels white; make everything else black

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

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

*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
```

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 I2 = rgb_edges(I,tau)
% I2 is the "edge diagram" of image I.
% At each pixel, if rate-of-change > tau
% then the pixel is considered to be on an edge.

A = rgb2gray(I);
[m,n] = size(A);
I2 = A;
for i = 1:m
    for j = 1:n

        I2(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
```

*I2(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 I2 = rgb_edges(I,tau)
% I2 is the "edge diagram" of image I.
% At each pixel, if rate-of-change > tau
% then the pixel is considered to be on an edge.
A = rgb2gray(I);
[m,n] = size(A);
I2 = A;
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));
        I2(i,j) = max(max(abs(double(Neighbors) - ...
                           double(A(i,j)))));

        if I2(i,j) > tau
            I2(i,j) = 255;
        end
    end
end
end
end

```

```
I = imread ( 'charlie.jpg' )
```



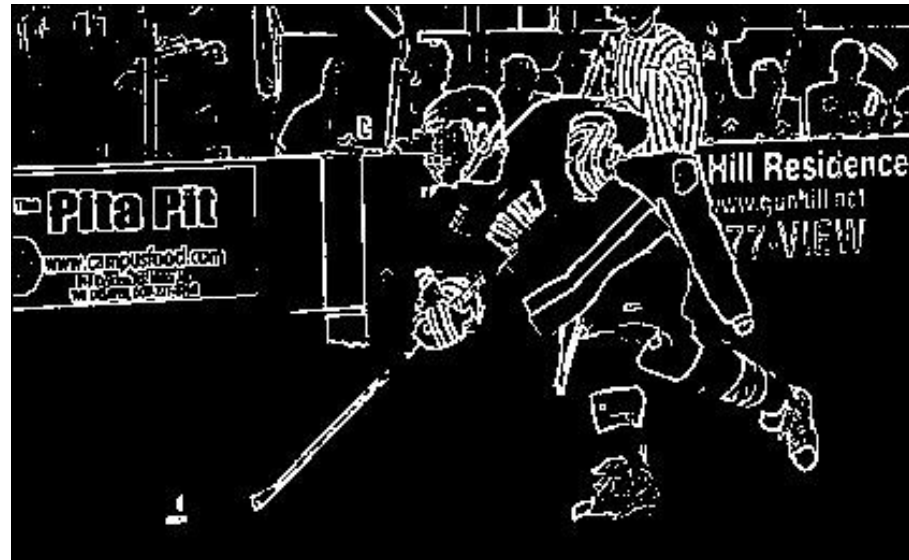
```
I2 = rgb_edges ( I, 15 )
```

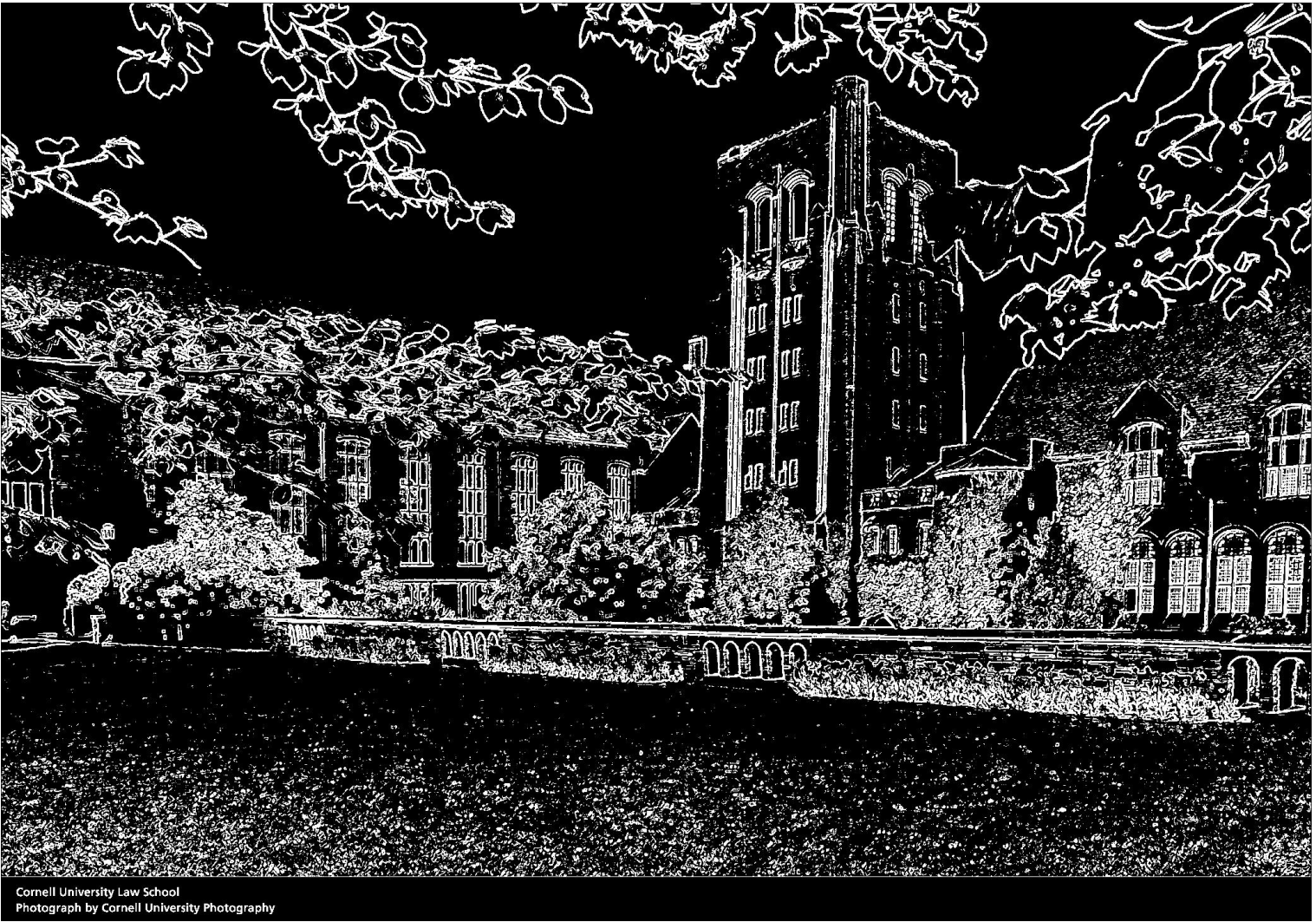




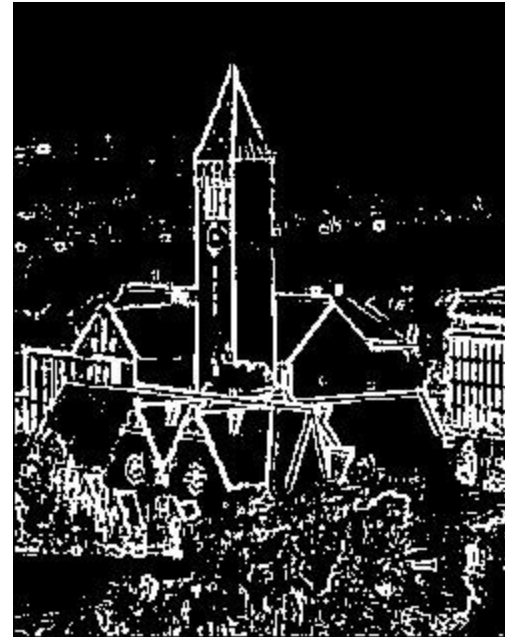


Threshold  
= 40





**Threshold = 20**



**Threshold = 30**

# Homework #9

hw047: Read a PNG file containing a grayscale image of Saturn, change to white all the pixels in a wide strip along the border of the image, and save the modified image to a JPG file.

hw048: Read a PNG file containing a grayscale image of a scene from the movie "Casablanca"; copy Ingrid Bergman's head and use it to replace the heads of Humphrey Bogart and Dooley Wilson; save the image as a JPG file.

hw049: Read a JPG file which is a photograph of a person. Make a new JPG file which only displays pixels that closely match a particular shade of the person's face.

Homework #9 is due by midnight, Friday November 10th.

Homework #8 is due by tomorrow night.