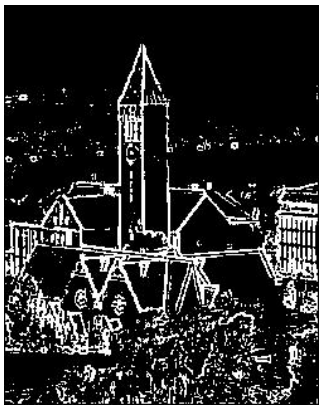
- Previous Lecture:
  - Working with images
- Today's Lecture:
  - More on manipulating images
    - "Noise" filtering
    - Edge finding

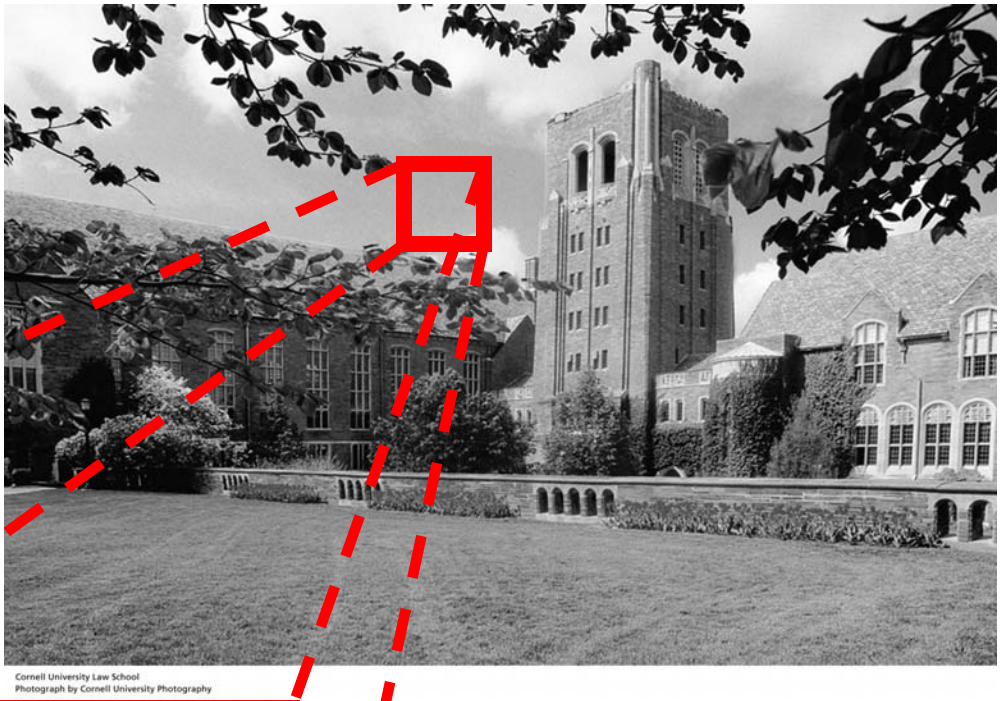


- Announcements:
  - Project 4 due tonight at 11pm

An image as an array: values 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

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+1-c,1)
    B(:,c,2) = A(:,nc+1-c,2)
    B(:,c,3) = A(:,nc+1-c,3)
end
imwrite(B,'LawSc                pg')
```

Can improve efficiency by initializing B to be a 3-d array of the appropriate size

Example: produce a negative

Problem: produce a negative

- "Negative" is what we say, but all color values are positive numbers!
- Think in terms of the extremes, 0 and 255. Then the "negative" just means the opposite side.
- So 0 is the opposite of 255;

  1 … 254;

  5 … 250;

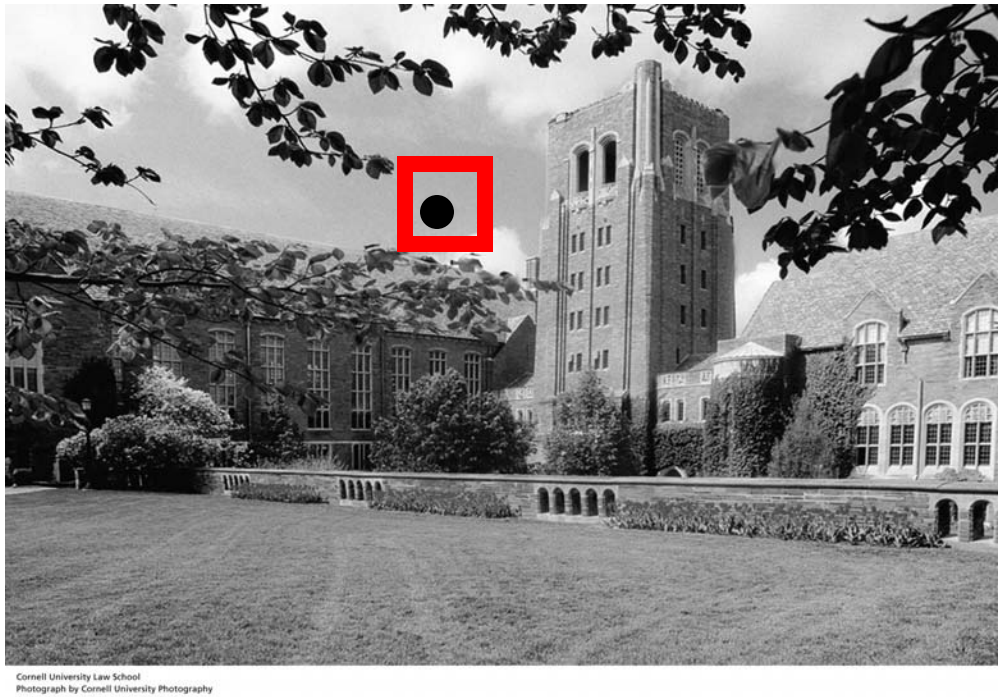
  30 … 225;

  x …

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

[nr,nc,np]= size(im);     % dimensions of im
newIm= zeros(nr,nc);      % 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)= ____________________;
        end
    end
end
```

```
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
```

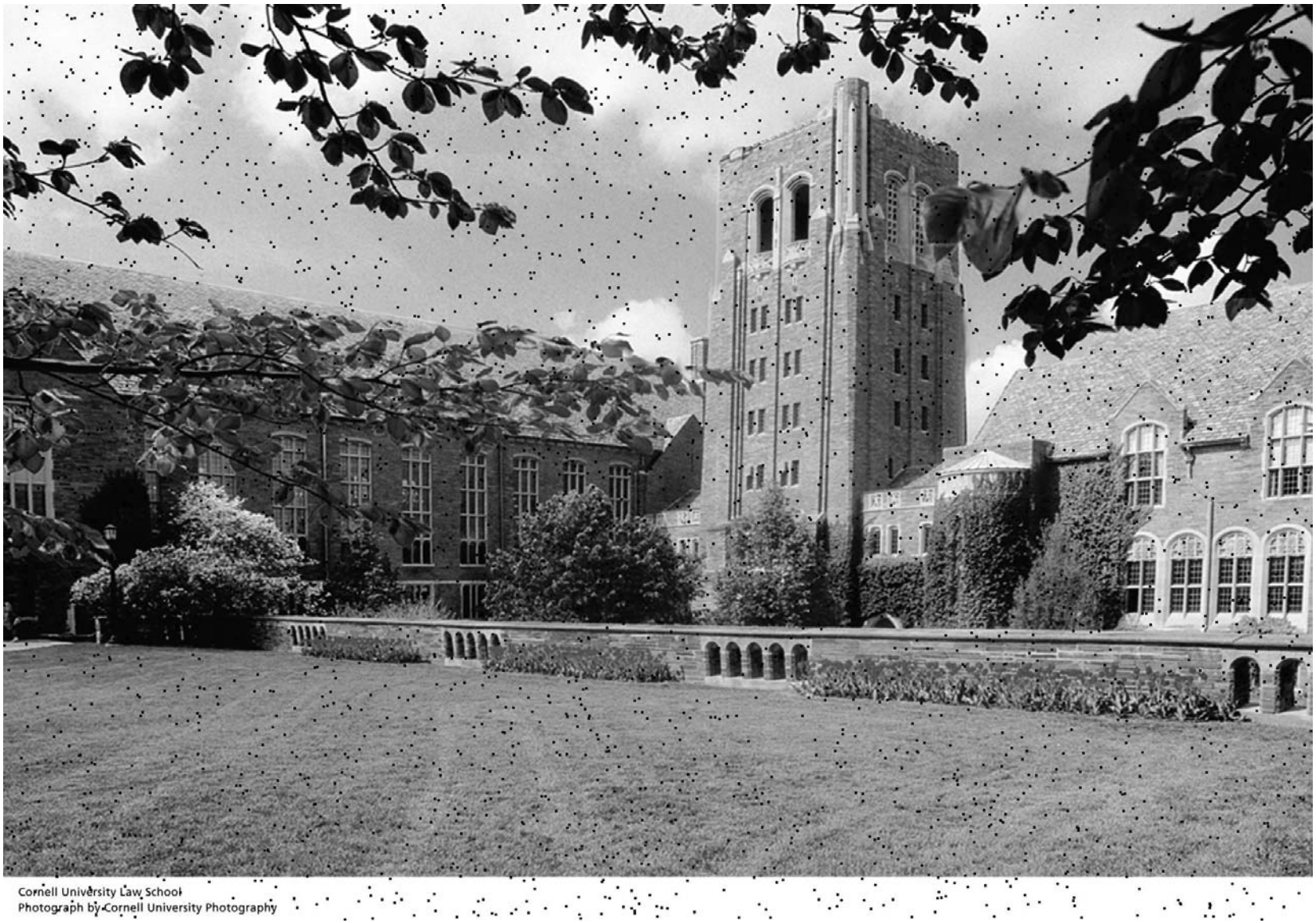
Dirt in the image!



Note how the "dirty pixels" look out of place

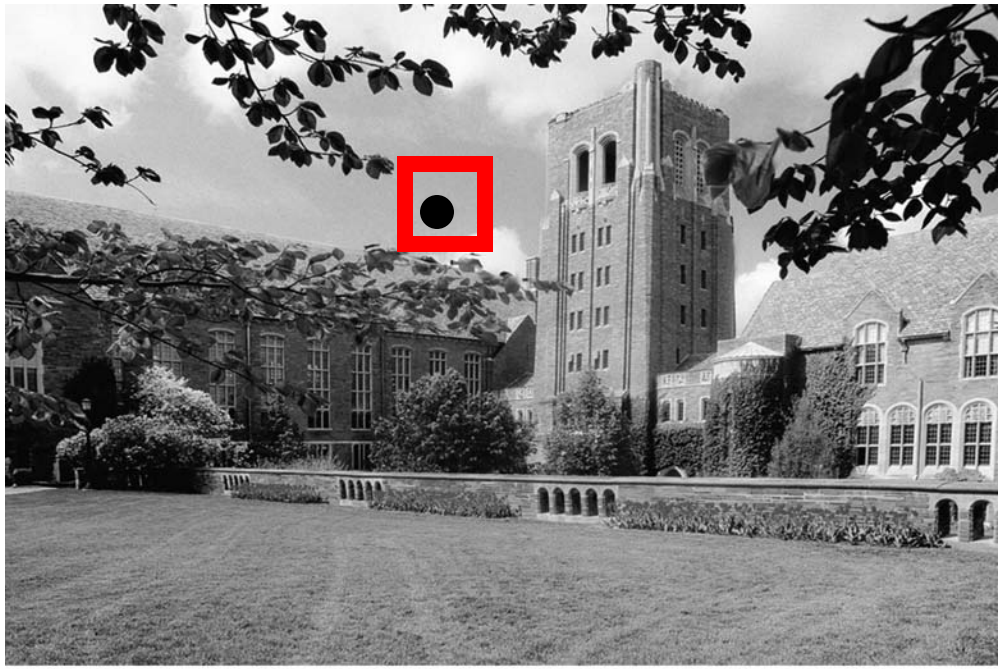
150	149	152	153	152	155
151	150	153	154	153	156
153	2	3	156	155	158
154	2	1	157	156	159
156	154	158	159	158	161
157	156	159	160	159	162

Clean up “noise” — median filtering



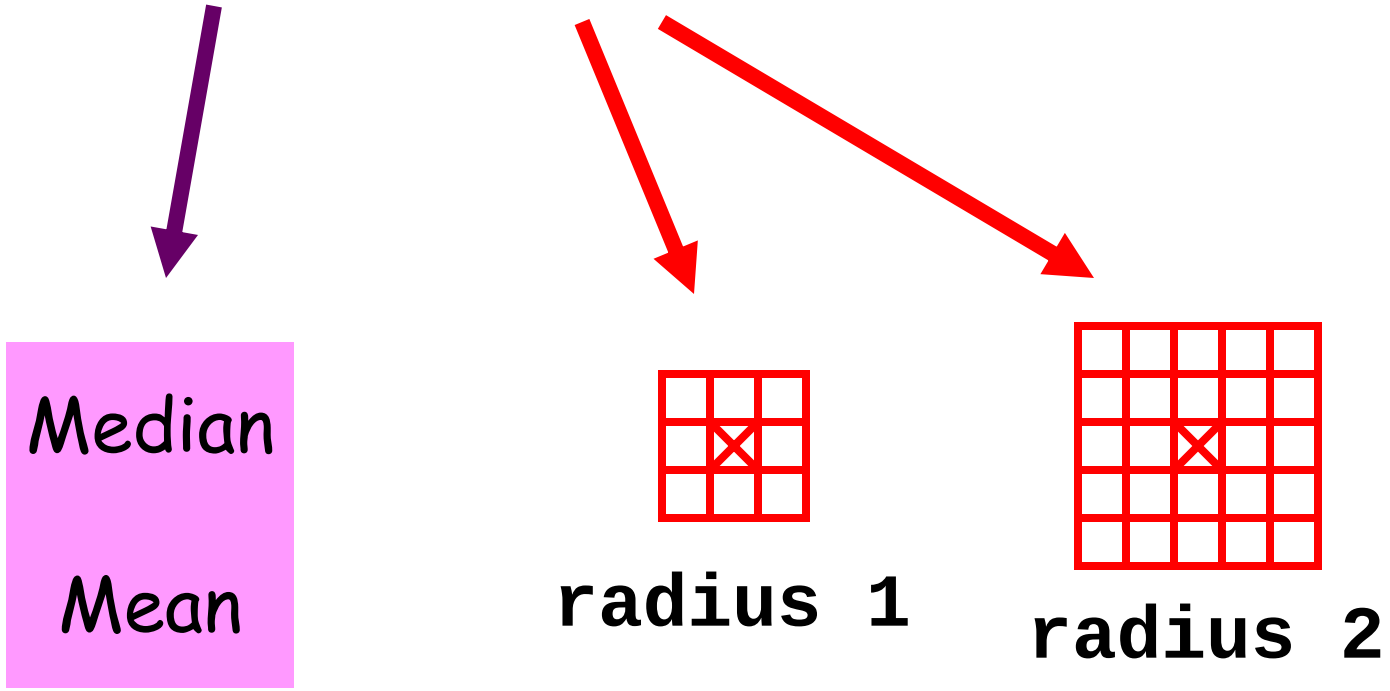
What to do with the dirty pixels?

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

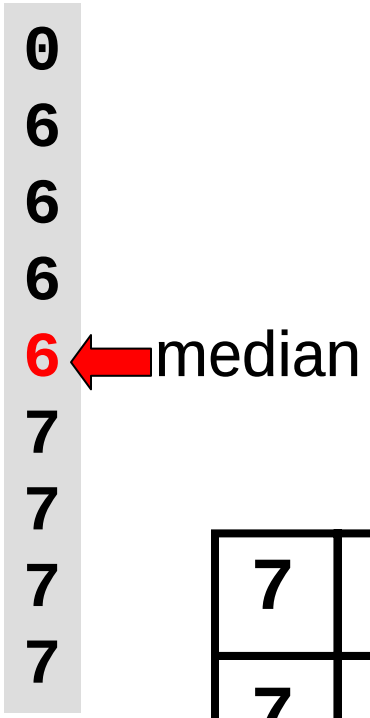
What are "typical neighborhood gray values"?

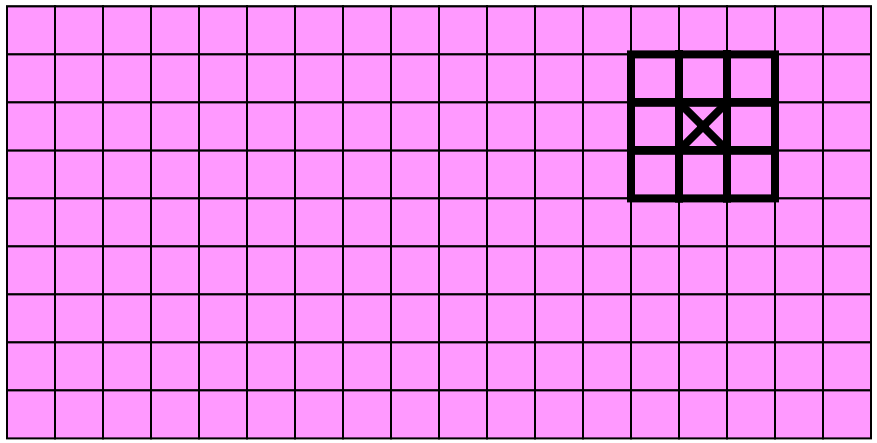


Median Filtering

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

Using a radius 1 "neighborhood"





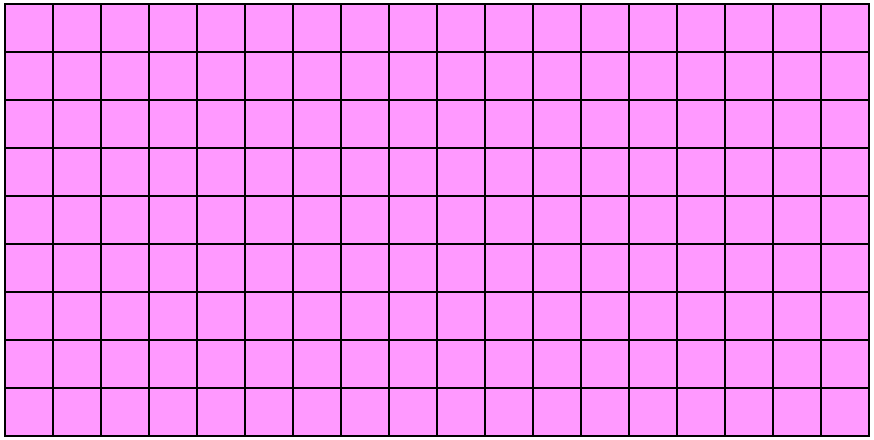
7	7	6
7	0	6
7	6	6

Before

7	7	6
7	6	6
7	6	6

After

Visit every pixel; compute its new value.

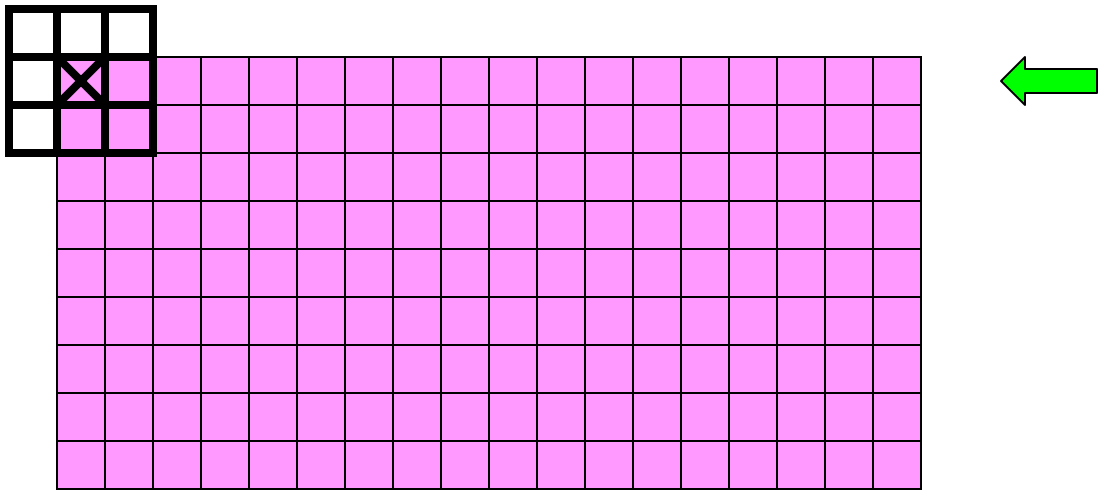


m = 9

n = 18

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

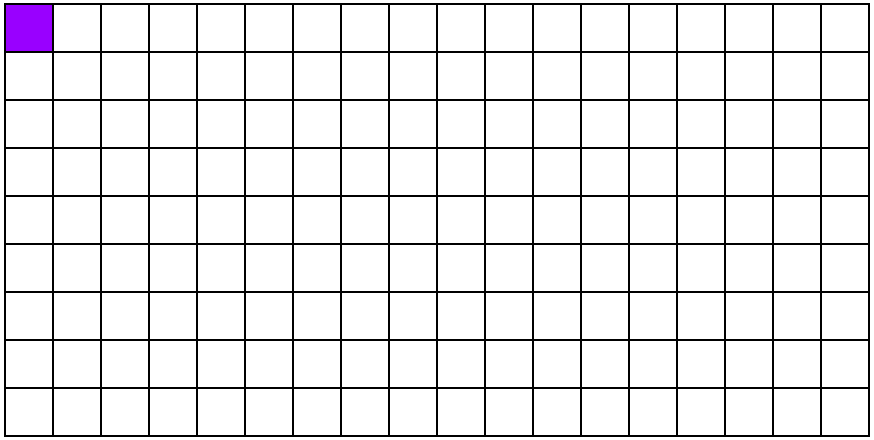
Original:



```
i = 1
```

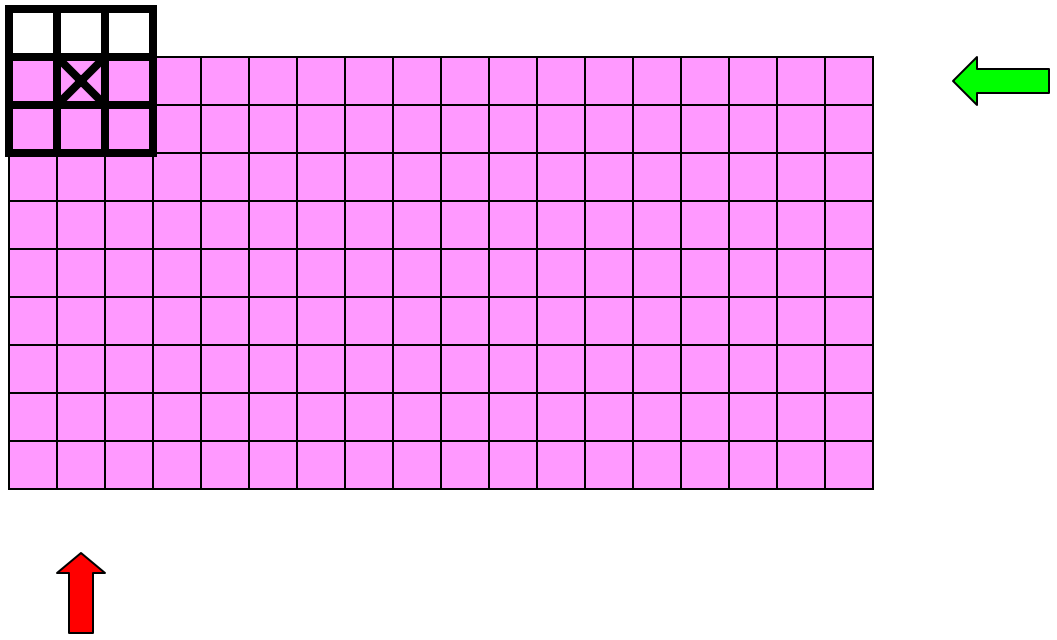
```
j = 1
```

Filtered:



Replace ⊠ with the median of the values under the window.

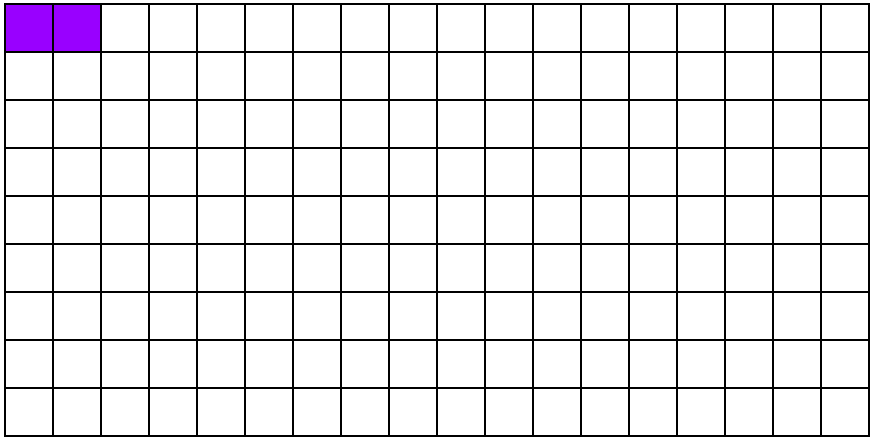
Original:



`i = 1`

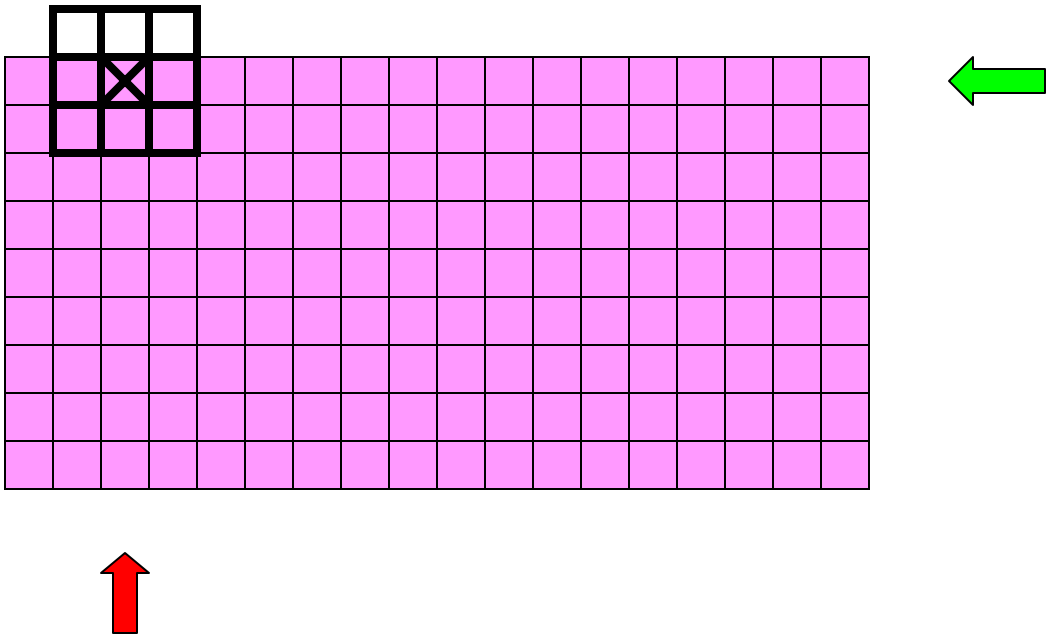
`j = 2`

Filtered:



Replace ⊠ with the median of the values under the window.

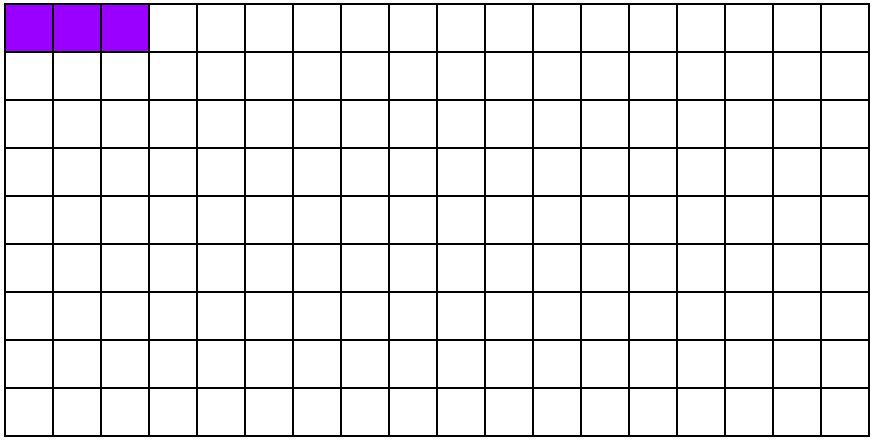
Original:



`i = 1`

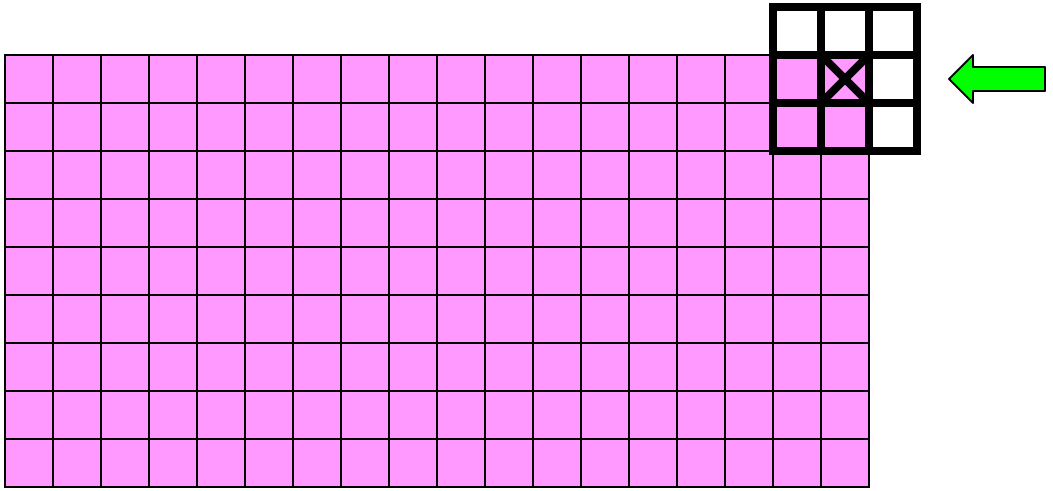
`j = 3`

Filtered:



Replace ⊠ with the median of the values under the window.

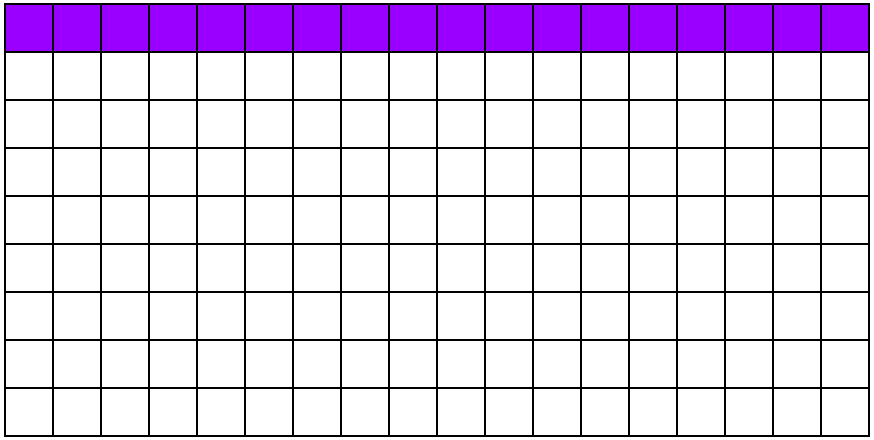
Original:



```
i = 1
```

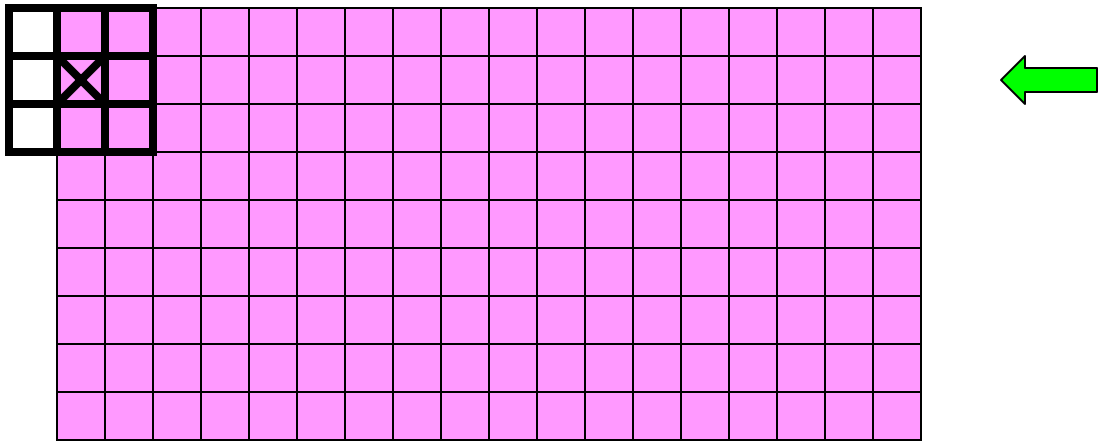
```
j = n
```

Filtered:



Replace ⊠ with the median of the values under the window.

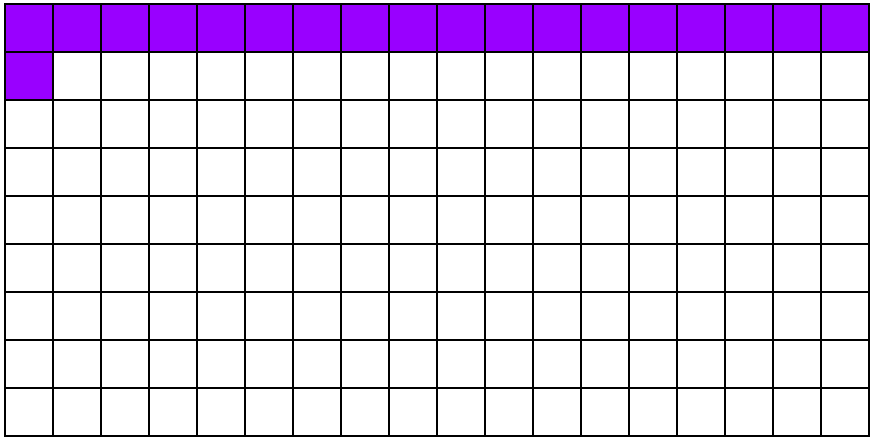
Original:



`i = 2`

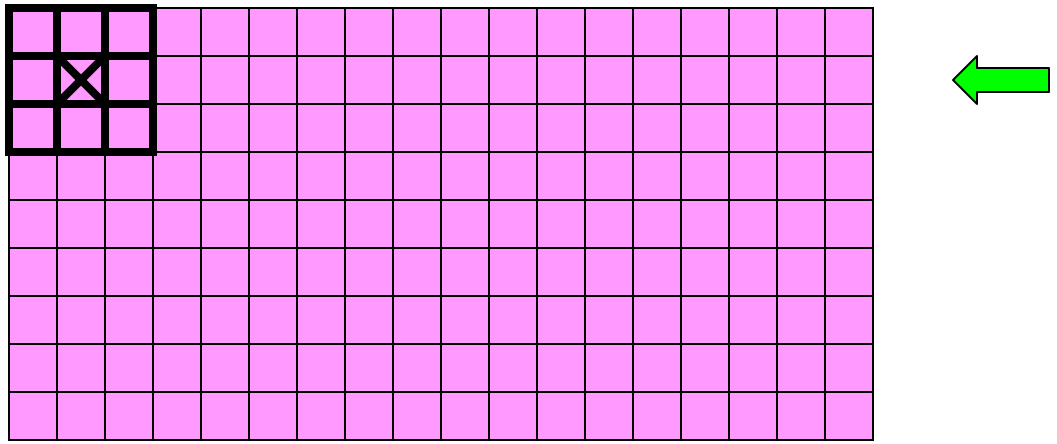
`j = 1`

Filtered:



Replace ⊠ with the median of the values under the window.

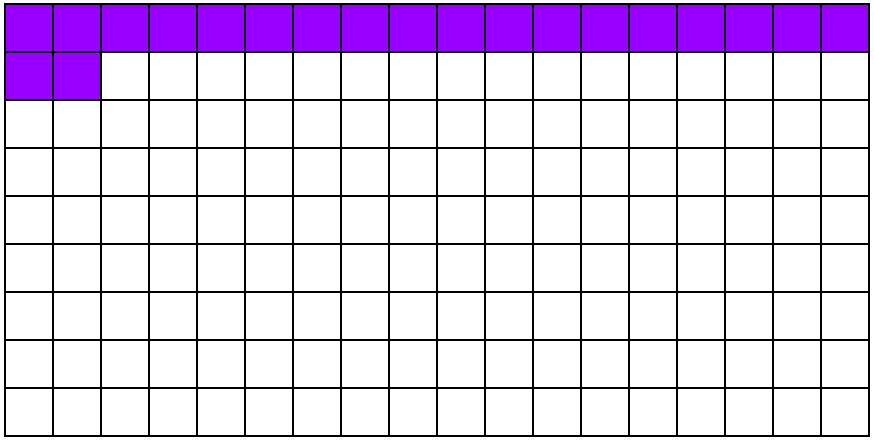
Original:



```
i = 2
```

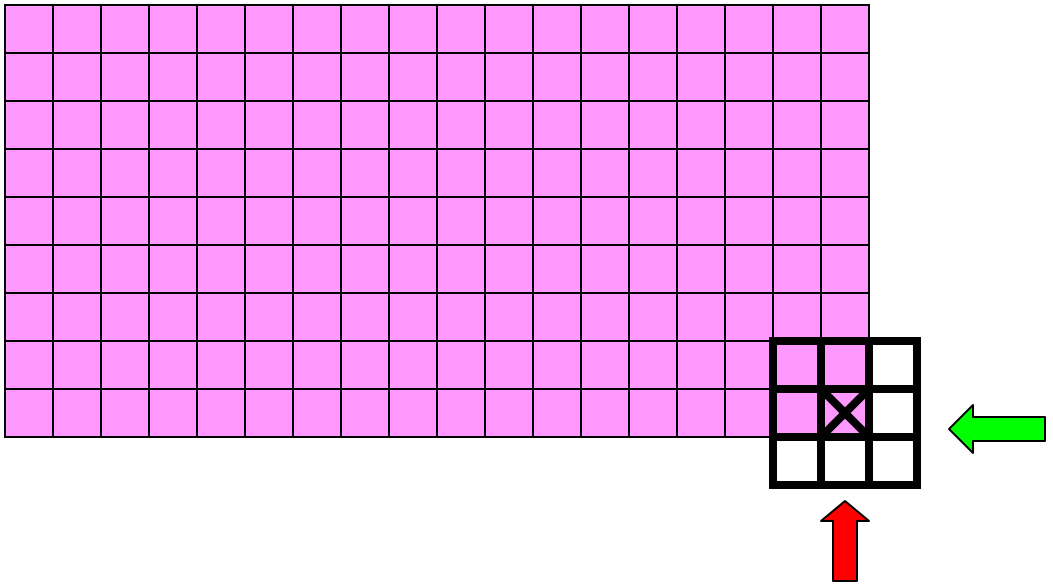
```
j = 2
```

Filtered:



Replace ⊠ with the median of the values under the window.

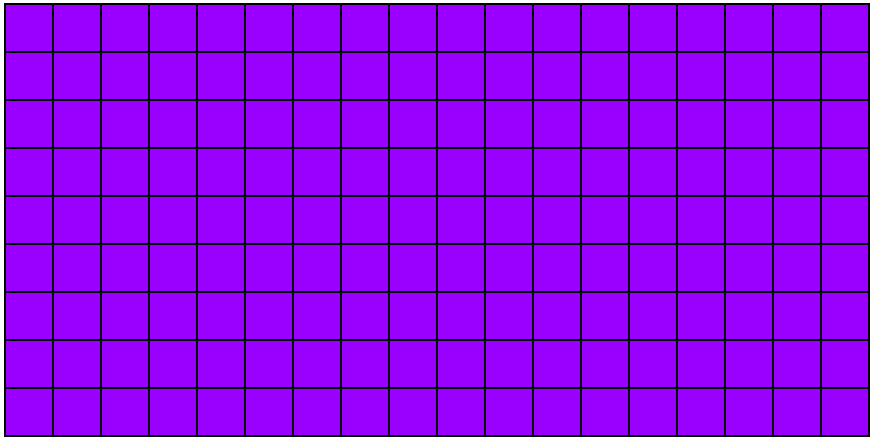
Original:



`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:

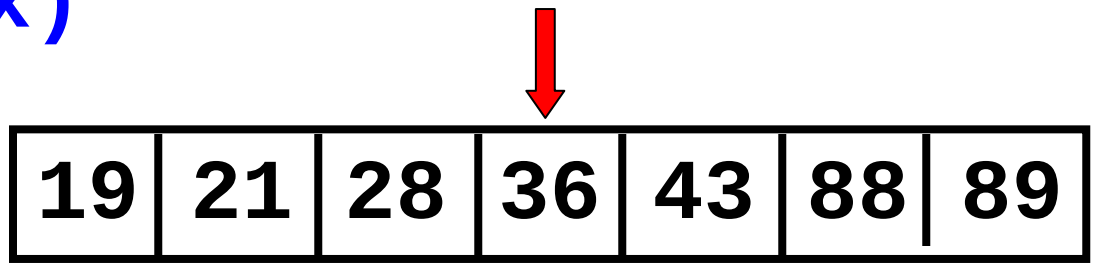
```
m = medVal(C)
```

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

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

Computing Medians


```
x = sort(x)
```




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

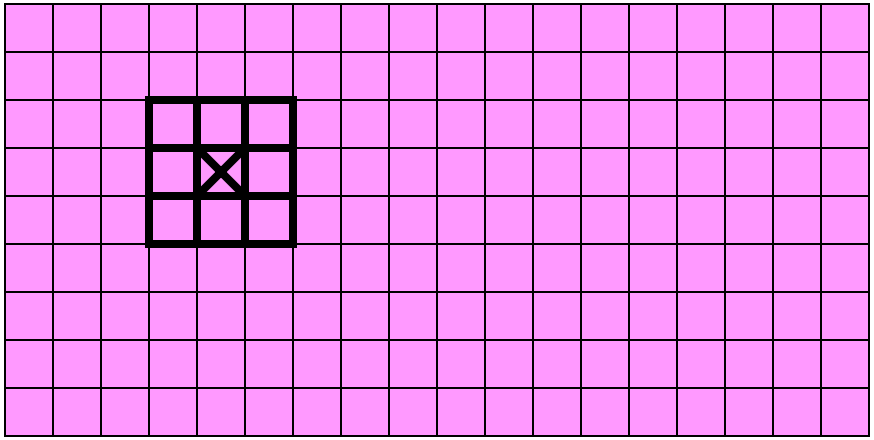
If n is even, then use : `med = x(m)/2 + x(m+1)/2`

Median of a 2D Array

```
function  med = medVal(C)
[p,q] = size(C);
x = [];
for k=1:p
    x = [x C(k,:)];
end
%Compute median of x and assign to med
```

See **medVal.m**

Back to Filtering…

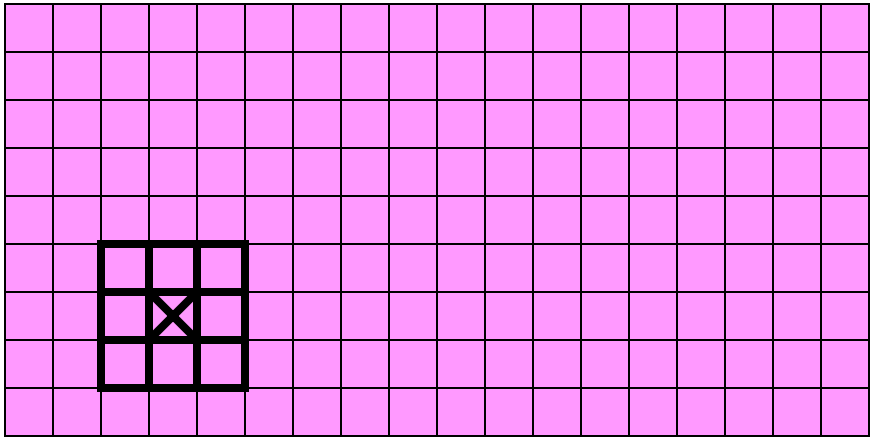


m = 9

n = 18

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

When window is inside…



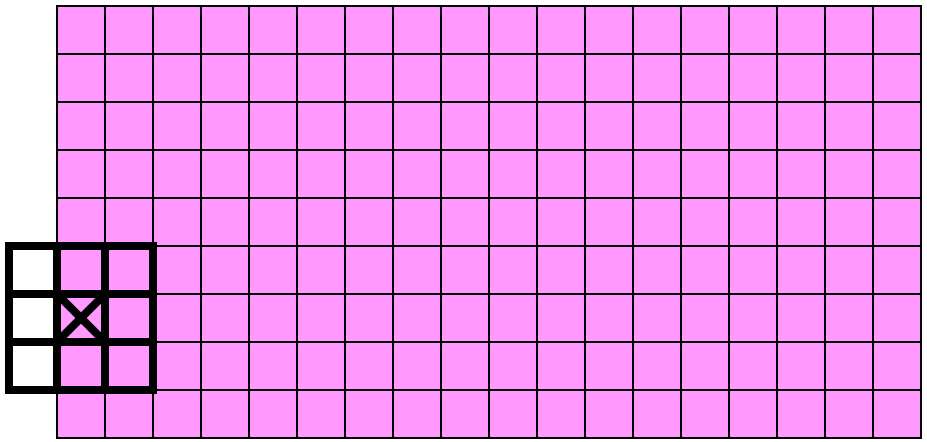
m = 9

n = 18

New gray value for pixel (7,4) =

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

When window is partly outside…



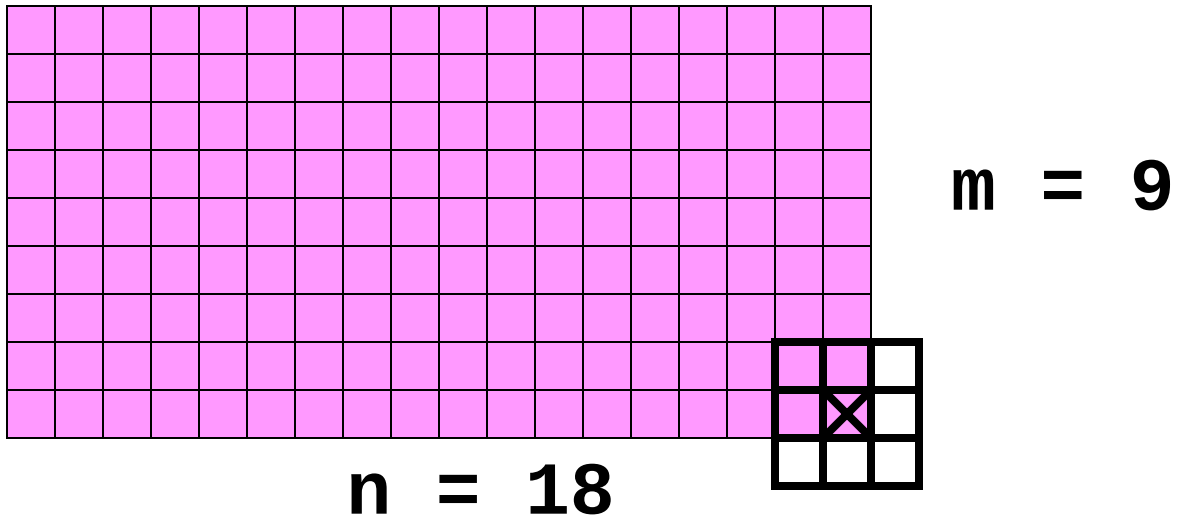
m = 9

n = 18

New gray value for pixel (7,1) =

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

When window is partly outside…



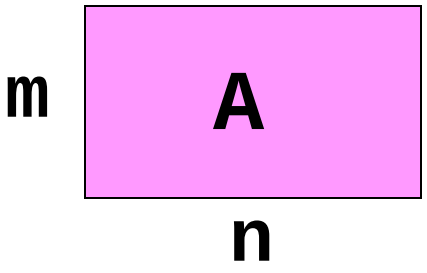
New gray value for pixel (9,18) =

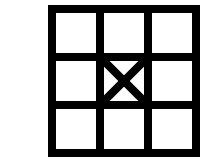
```
medVal( A(8:9,17:18) )
```

```
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
```

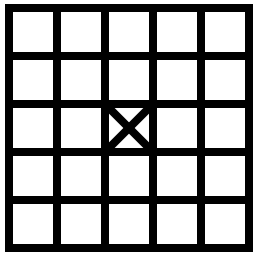
The Pixel (i,j) Neighborhood

```
iMin =          i-r
iMax =          i+r
jMin =          j-r
jMax =          j+r
C = A(iMin:iMax,jMin:jMax)
```





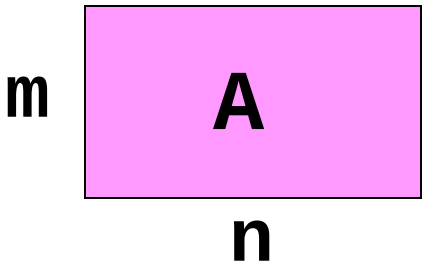
r = 1

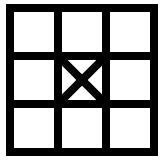


r = 2

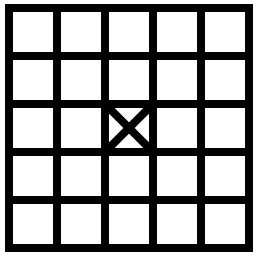
The Pixel (i,j) Neighborhood

```
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)
```

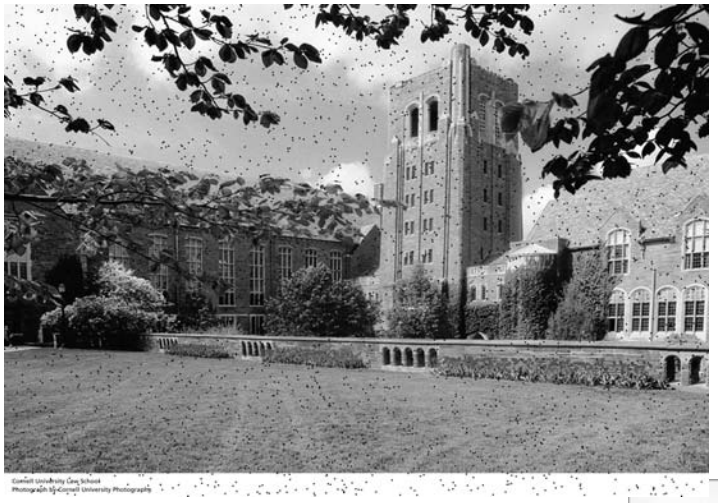




r = 1



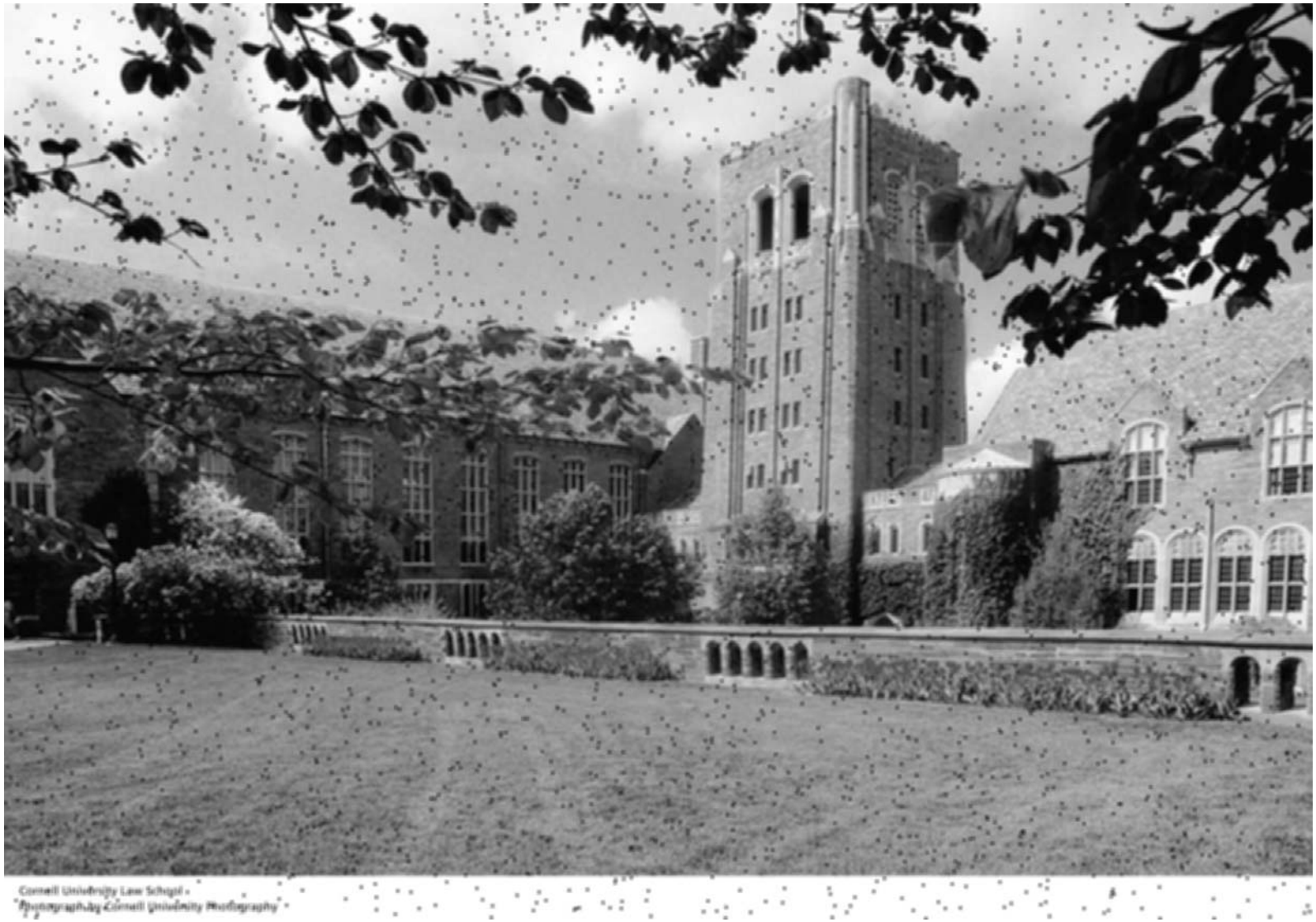
r = 2



A

B = medianFilter(A,3)

Mean Filter with radius 3



Mean Filter with radius 10

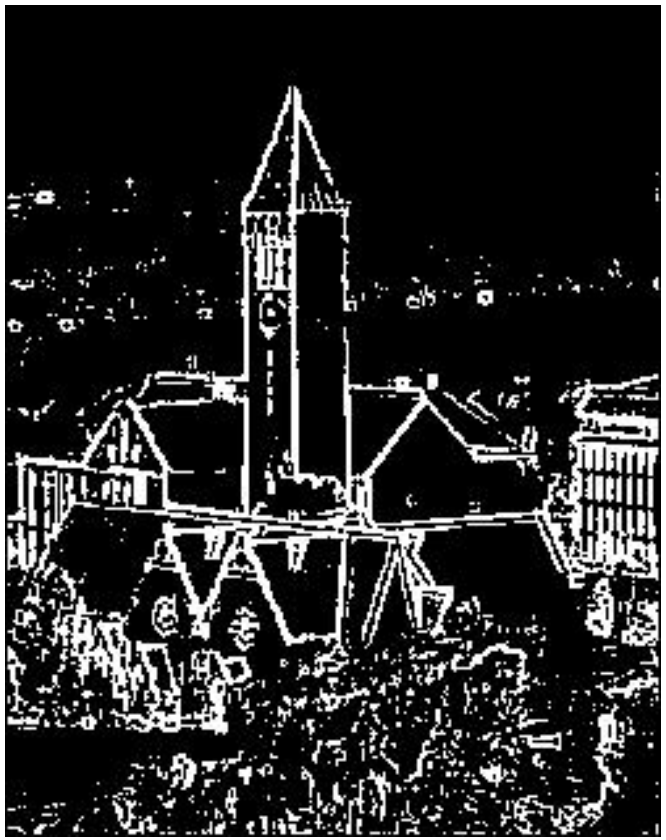
Mean filter fails because the mean does not capture representative values.

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

mean-filtered values

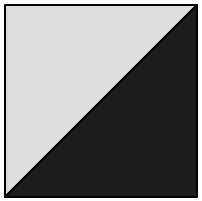
Finding Edges



What is an Edge?

Near an edge, grayness values change abruptly

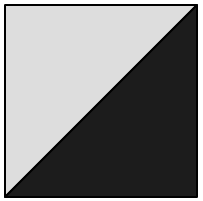
200	200	200	200	200	200
200	200	200	200	200	100
200	200	200	200	100	100
200	200	200	100	100	100
200	200	100	100	100	100
200	100	100	100	100	100



General plan for showing the edges in in image

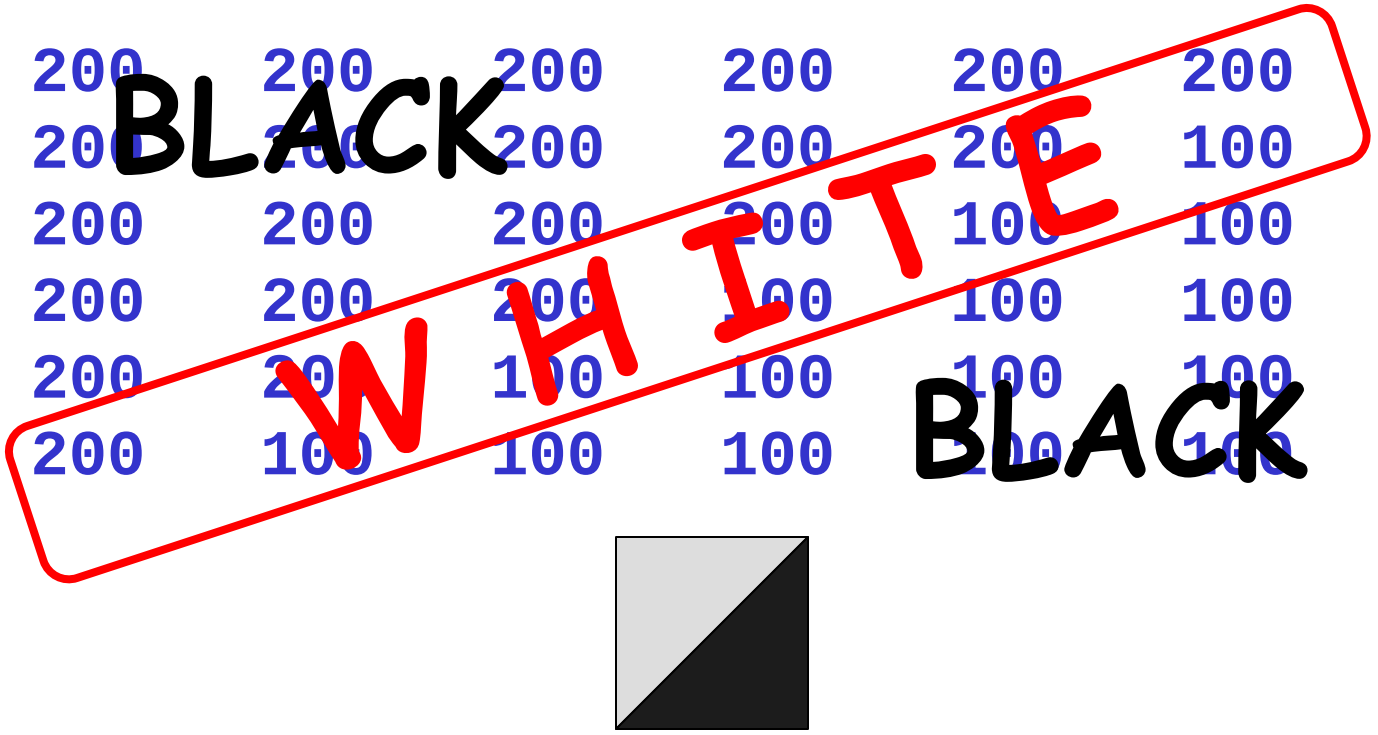
- Identify the "edge pixels"
- Highlight the edge pixels
  - make edge pixels white; make everything else black

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

Let **B(i,j)** be 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

Remember that we're doing "uint8 arithmetic"!
57 - 60 is 0 in uint8

There can be different definitions for the rate-of-change

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

| Neighborhood – A(i,j) |

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

Neighborhood – A(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

A(i,j) – Neighborhood

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

        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 = Neighbors – 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(Neighbors – A(i,j)));



  end
end
```

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

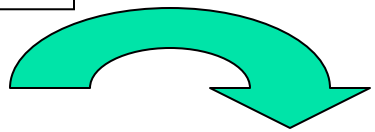
A

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

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

B(i,j)

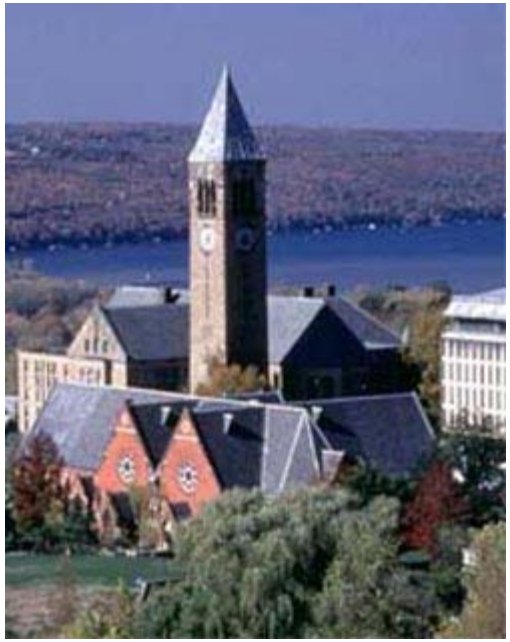
0	0	0	0	0	0
0	0	0	89	89	89
0	0	89	89	0	0
0	89	89	0	0	0
0	89	0	0	0	0
0	89	0	0	0	0

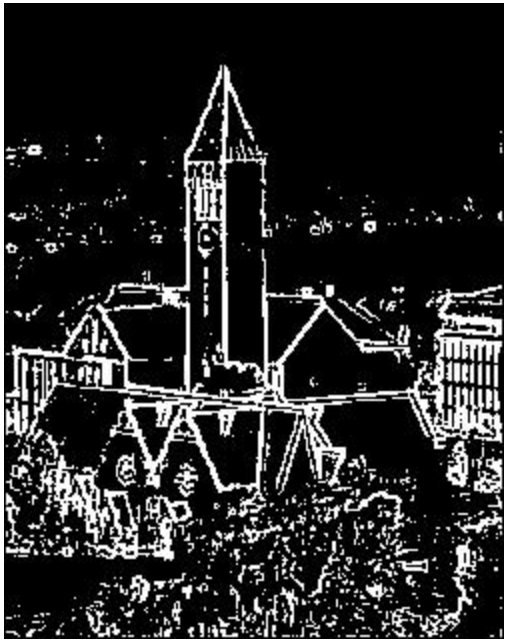


0	0	0	0	0	0
0	0	0	255	255	255
0	0	255	255	0	0
0	255	255	0	0	0
0	255	0	0	0	0
0	255	0	0	0	0

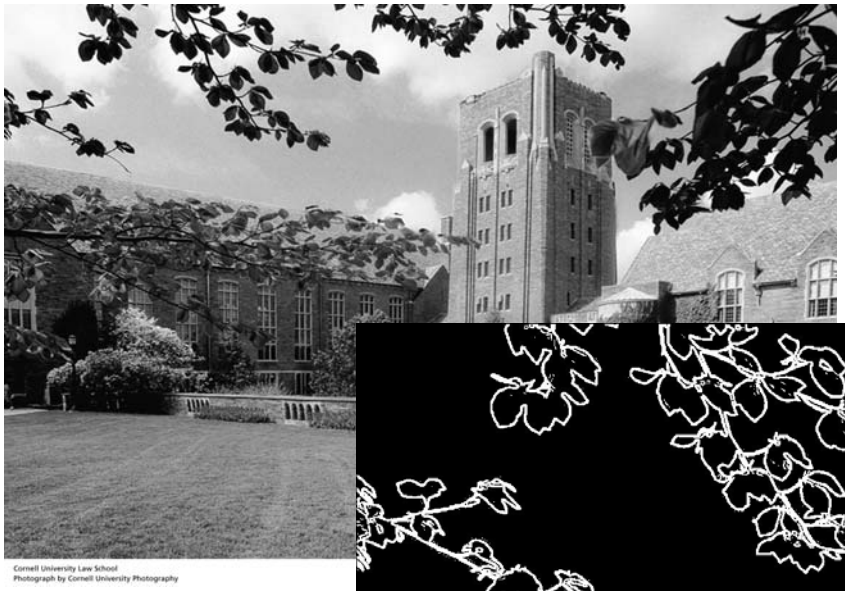
```
function Edges(jpgIn,jpgOut,tau)
% jpgOut is the “edge diagram” of image jpgIn.
% At each pixel, if   rate-of-change > tau
% then the pixel is considered to be on an edge.
A = rgb2gray(imread(jpgIn));
[m,n] = size(A);
B = uint8(zeros(m,n));
for i = 1:m
  for j = 1:n
    Neighbors = A(max(1,i-1):min(i+1,m), ...
                  max(1,j-1):min(j+1,n));
    B(i,j) = max( max(Neighbors – A(i,j)));
    if B(i,j) > tau
      B(i,j) = 255;
    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(Neighbors – A(i,j)));
    if B(i,j) > tau
      B(i,j) = 255;
    end
  end
end
imwrite(B,jpgOut,’jpg’)
```

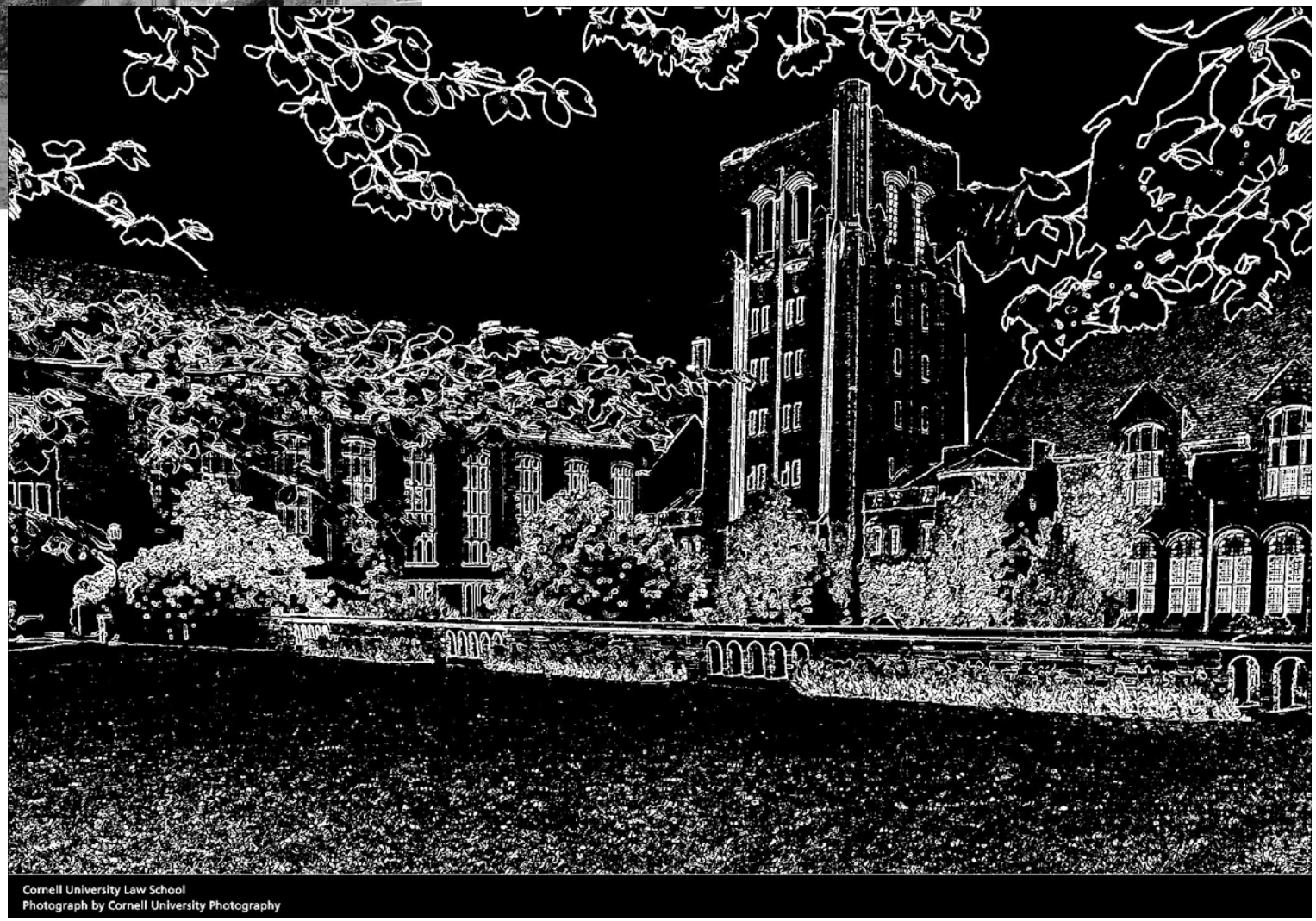




tau = 30


Cornell University Law School
Photograph by Cornell University Photography

tau = 20


Cornell University Law School
Photograph by Cornell University Photography