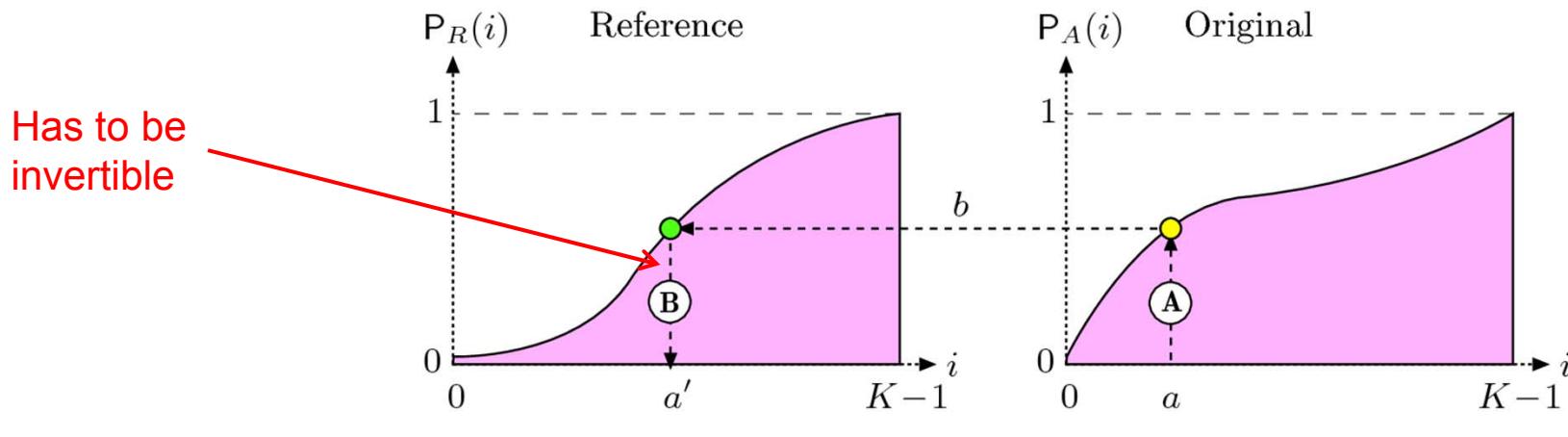




Histogram Matching

- Prior method needed reference distribution to be invertible



$$f_{hs}(a) = a' = P_R^{-1}(P_A(a))$$

- What if reference histogram is not invertible?
- For example not invertible if histogram has some intensities that occur with probability 0? i.e. $p(k) = 0$
- Use different method called **histogram matching**

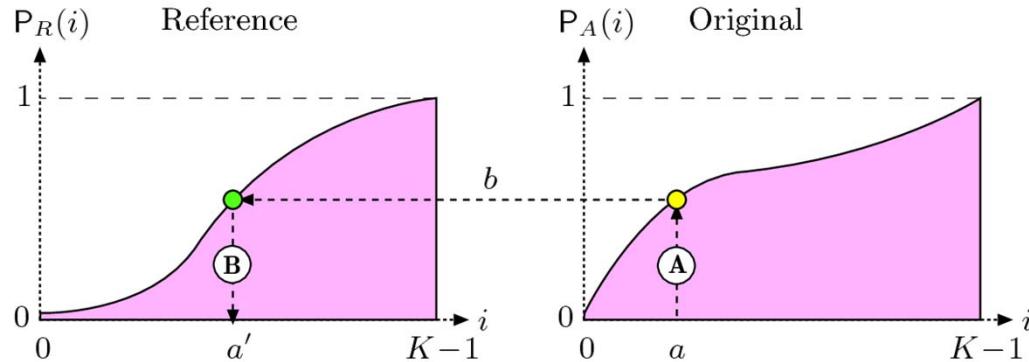


Histogram Matching

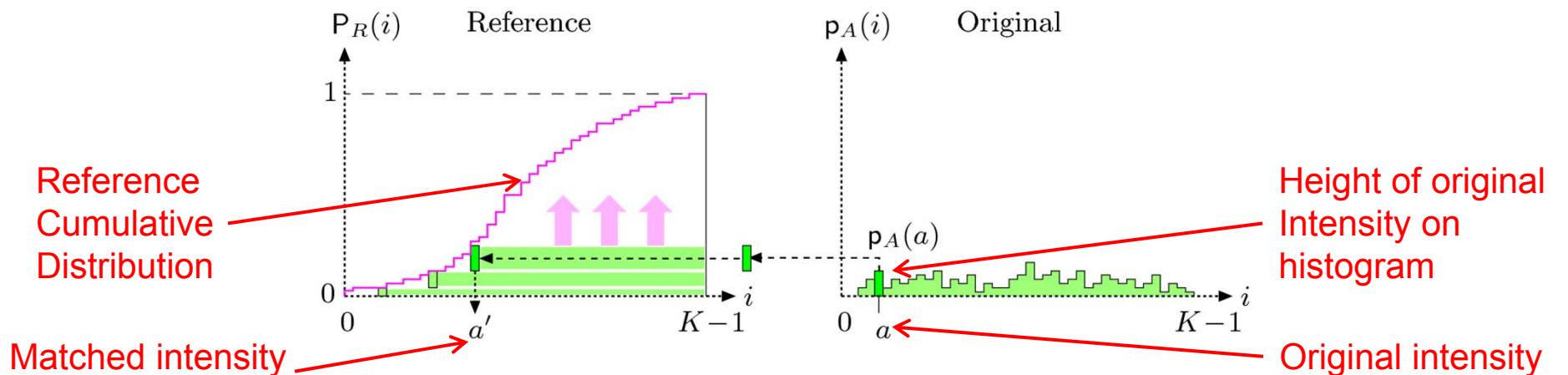
- Given two images I_A and I_B , we want to make their intensity profiles look as similar as possible.
- How?
 - “Match” their cumulative histograms H_A and H_B
- Works well for images with similar content.
- Looks bad for images with different content.



Adjusting to a Given Histogram

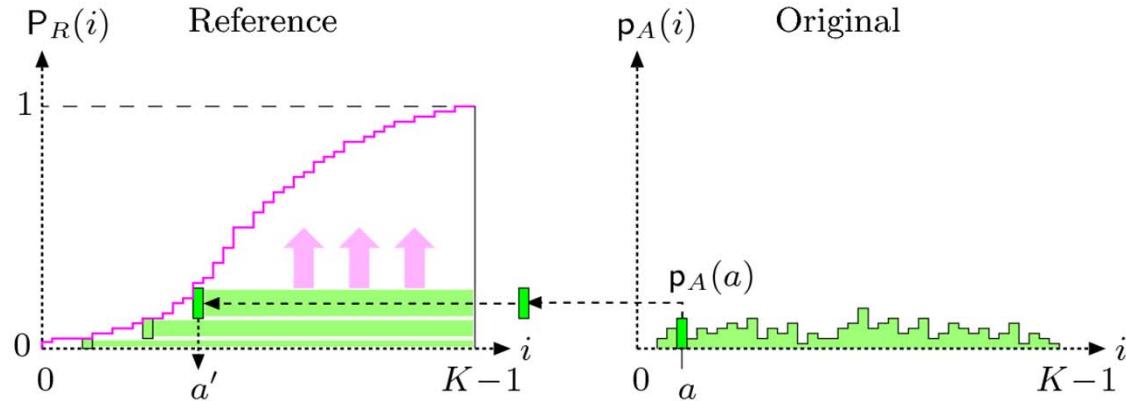


$$f_{hs}(a) = a' = \min \{ j \mid (0 \leq j < K) \wedge (P_A(j) \leq P_R(j)) \}$$





Adjusting to a Given Histogram



1: MATCHHISTOGRAMS($\mathbf{h}_A, \mathbf{h}_R$)

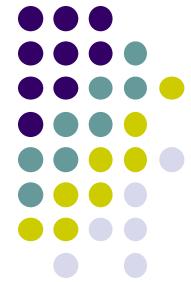
\mathbf{h}_A : histogram of the target image

\mathbf{h}_R : reference histogram (of same size as \mathbf{h}_A)

```

2: Let  $K \leftarrow \text{Size}(\mathbf{h}_A)$ 
3: Let  $\mathbf{P}_A \leftarrow \text{CDF}(\mathbf{h}_A)$                                  $\triangleright$  cdf for  $\mathbf{h}_A$  (Alg. 5.1)
4: Let  $\mathbf{P}_R \leftarrow \text{CDF}(\mathbf{h}_R)$                                  $\triangleright$  cdf for  $\mathbf{h}_R$  (Alg. 5.1)
5: Create a table  $f_{\text{hs}}[ ]$  of size  $K$                        $\triangleright$  pixel mapping function  $f_{\text{hs}}$ 
6: for  $a \leftarrow 0 \dots (K-1)$  do
7:    $j \leftarrow K-1$ 
8:   repeat
9:      $f_{\text{hs}}[a] \leftarrow j$ 
10:     $j \leftarrow j - 1$ 
11:   while ( $j \geq 0$ )  $\wedge (\mathbf{P}_A(a) \leq \mathbf{P}_R(j))$ 
12: return  $f_{\text{hs}}$ .
```

Adjusting to a Given Histogram

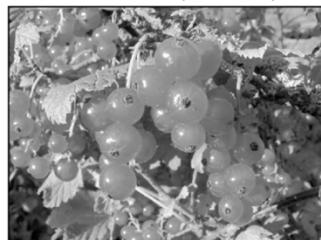


Original Image



(a) I_A

Gaussian ($\sigma = 50$)



(b) I_{G50}

Gaussian ($\sigma = 100$)



(c) I_{G100}

Reference Histogram



$$\mathbf{h}_R(i)$$

Cumulative Reference Histogram

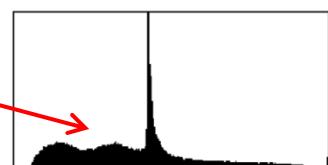


$$\mathsf{H}_R(i)$$

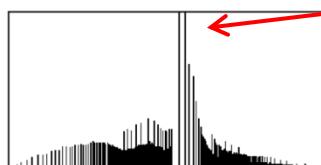
(d)

(e)

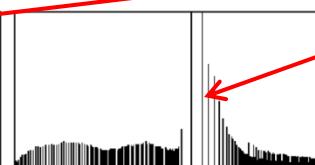
original histogram



(f) h_A

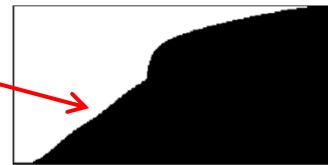


(g) h_{G50}



(h) h_{G10e}

CDF of original histogram



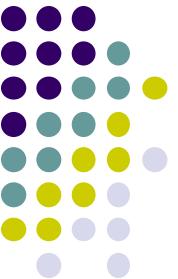
(i) H_A



(j) H_{G50}

Original histogram after matching

CDF of original
histogram after
matching



Adjusting to a Given Histogram

Target Image



(a) I_A

Reference Image

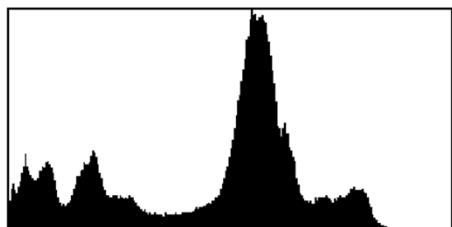


(b) I_R

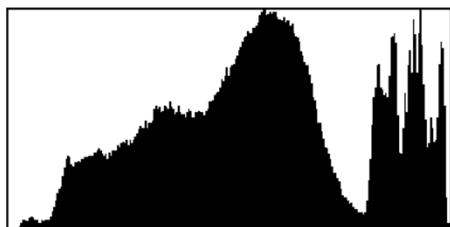
Modified Image



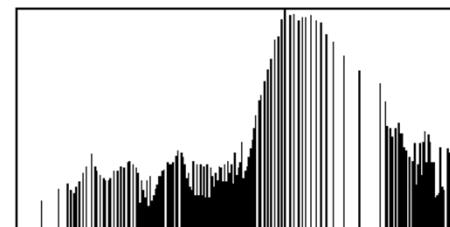
(c) $I_{A'}$



(d) h_A



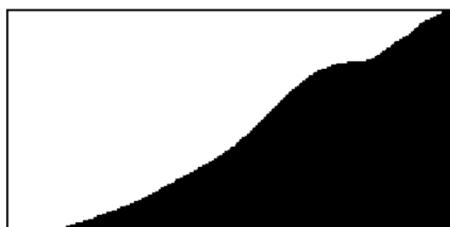
(e) h_R



(f) $h_{A'}$



(g) H_A

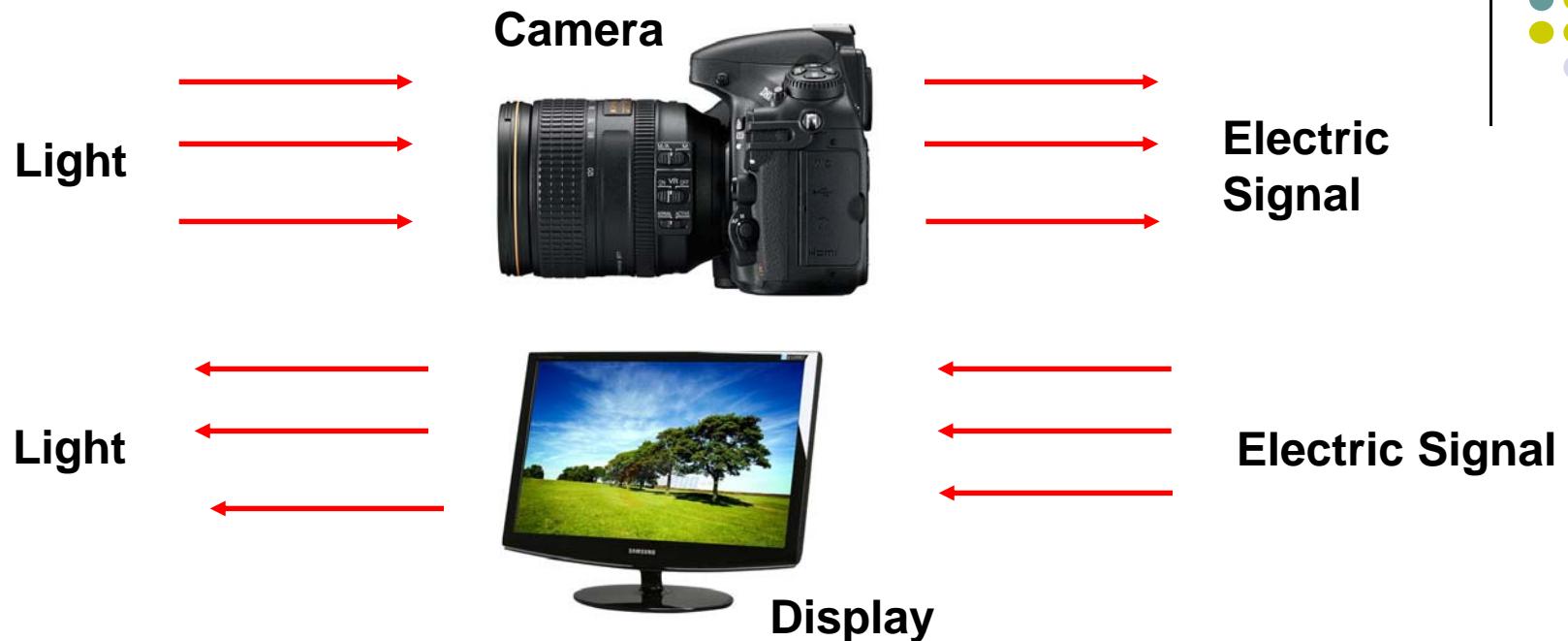
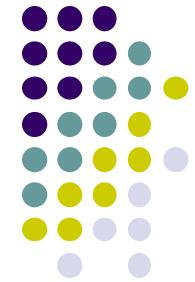


(h) H_R



(i) $H_{A'}$

Gamma Correction



- Different camera sensors
 - Have different responses to light intensity
 - Produce different electrical signals for same input
- How do we ensure there is consistency in:
 - a) Images recorded by different cameras for given light input
 - b) Light emitted by different display devices for same image?



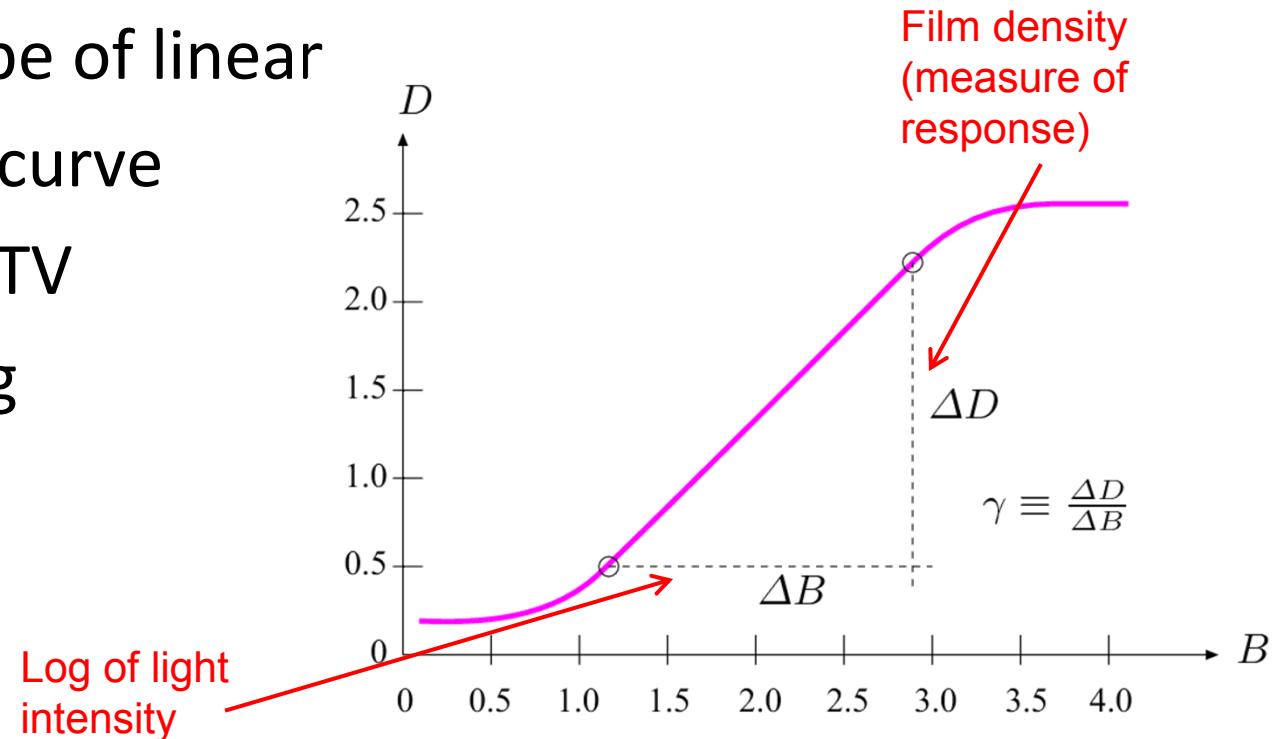
Gamma Correction

- What is the relation between:
 - **Camera:** Light on sensor vs. “**intensity**” of corresponding pixel
 - **Display:** Pixel intensity vs. light from that pixel
- Relation between pixel value and corresponding physical quantity is usually complex, nonlinear
- An approximation ?



What is Gamma?

- Originates from analog photography
- **Exposure function:** relationship between:
 - logarithmic light intensity vs. resulting film density.
- **Gamma:** slope of linear range of the curve
- The same in TV broadcasting



What is Gamma?

- **Gamma function:** a good approximation of exposure curve
- Inverse of a Gamma function is another gamma function with

$$\bar{\gamma} = 1/\gamma$$

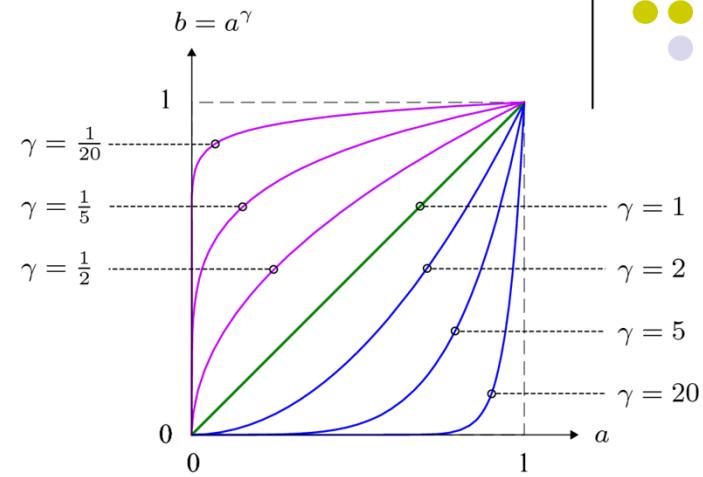
- Gamma of CRT and LCD monitors:
- 1.8-2.8 (typically 2.4)

$$s = B^{\gamma_c}$$

Output signal
Raised by gamma

$$b = s^{1/\gamma_c} = (B^{\gamma_c})^{1/\gamma_c} = B^{(\gamma_c \frac{1}{\gamma_c})} = B^1$$

Correct output signal
By dividing by 1/ gamma
(called Gamma correction)



$$b = f_\gamma(a) = a^\gamma \quad \text{for } a \in \mathbb{R}, \gamma > 0$$

$$a = f_\gamma^{-1}(b) = b^{1/\gamma}$$

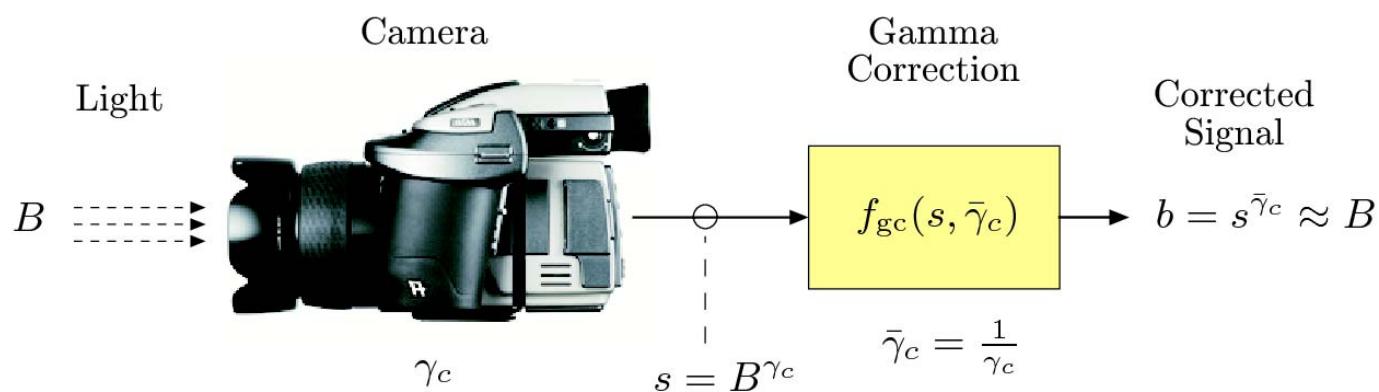
$$f_\gamma^{-1}(b) = f_{\bar{\gamma}}(b)$$

$$\bar{\gamma} = 1/\gamma$$



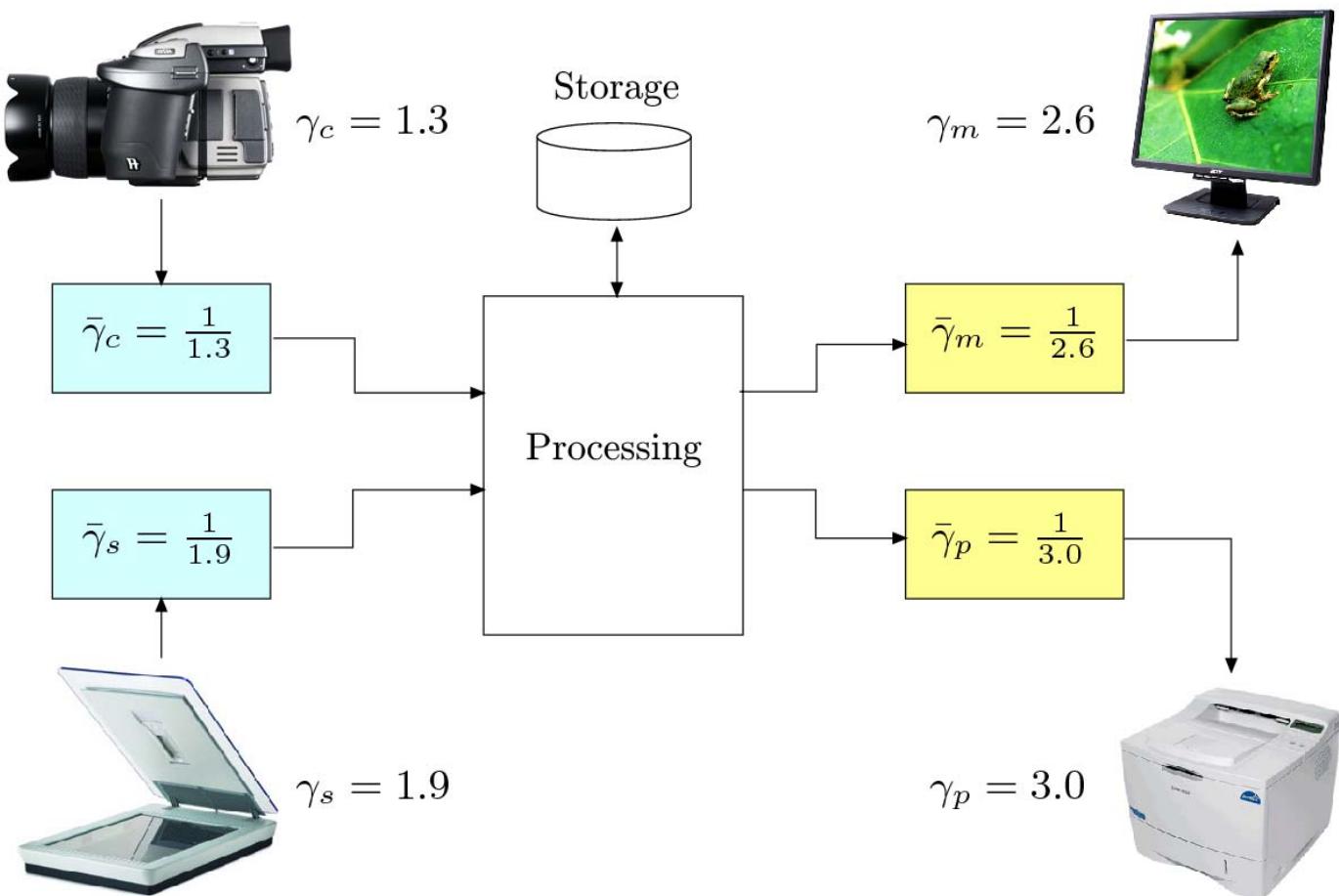
Gamma Correction

- Obtain a measurement **b** proportional to original light intensity **B** by applying inverse gamma function
- Gamma correction is important to achieve a device independent representation





Gamma Correction





Gamma Correction Code

```
1  public void run(ImageProcessor ip) {  
2      // works for 8-bit images only  
3      int K = 256;  
4      int aMax = K - 1;  
5      double GAMMA = 2.8;  
6  
7      // create a lookup table for the mapping function  
8      int[] Fgc = new int[K];  
9  
10     for (int a = 0; a < K; a++) {  
11         double aa = (double) a / aMax;    // scale to [0, 1]  
12         double bb = Math.pow(aa,GAMMA); // gamma function  
13         // scale back to [0, 255]:  
14         int b = (int) Math.round(bb * aMax);  
15         Fgc[a] = b;                  ←  
16     }  
17  
18     ip.applyTable(Fgc); // modify the image ip  
19 }
```

Compute corrected intensity and store in lookup table **fgc**



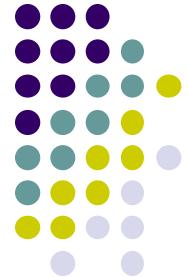
Point Operations in ImageJ

void abs()	$I'(u, v) \leftarrow I(u, v) $
void add(int <i>p</i>)	$I'(u, v) \leftarrow I(u, v) + p$
void gamma(double <i>g</i>)	$I'(u, v) \leftarrow (I(u, v)/255)^g \cdot 255$
void invert(int <i>p</i>)	$I'(u, v) \leftarrow 255 - I(u, v)$
void log()	$I'(u, v) \leftarrow \log_{10}(I(u, v))$
void max(double <i>s</i>)	$I'(u, v) \leftarrow \max(I(u, v), s)$
void min(double <i>s</i>)	$I'(u, v) \leftarrow \min(I(u, v), s)$
void multiply(double <i>s</i>)	$I'(u, v) \leftarrow \text{round}(I(u, v) \cdot s)$
void sqr()	$I'(u, v) \leftarrow I(u, v)^2$
void sqrt()	$I'(u, v) \leftarrow \sqrt{I(u, v)}$



ImageJ Operations involving 2 images

ADD	$ip1 \leftarrow ip1 + ip2$
AVERAGE	$ip1 \leftarrow (ip1 + ip2) / 2$
DIFFERENCE	$ip1 \leftarrow ip1 - ip2 $
DIVIDE	$ip1 \leftarrow ip1 / ip2$
MAX	$ip1 \leftarrow \max(ip1, ip2)$
MIN	$ip1 \leftarrow \min(ip1, ip2)$
MULTIPLY	$ip1 \leftarrow ip1 \cdot ip2$
SUBTRACT	$ip1 \leftarrow ip1 - ip2$



Example: Alpha Blending

$$I'(u, v) \leftarrow \alpha \cdot I_{\text{BG}}(u, v) + (1 - \alpha) \cdot I_{\text{FG}}(u, v)$$



I_{BG}



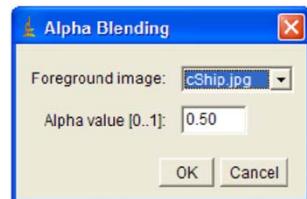
$\alpha = 0.25$



I_{FG}



$\alpha = 0.50$



GenericDialog

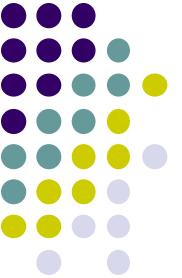


$\alpha = 0.75$



Alpha Blending PlugIn

```
1 import ij.IJ;
2 import ij.ImagePlus;
3 import ij.WindowManager;
4 import ij.gui.GenericDialog;
5 import ij.plugin.filter.PlugInFilter;
6 import ij.process.*;
7
8 public class Alpha_Blending implements PlugInFilter {
9
10    static double alpha = 0.5; // transparency of foreground image
11    ImagePlus fgIm = null;    // foreground image
12
13    public int setup(String arg, ImagePlus imp) {
14        return DOES_8G;
15    }
16
17    public void run(ImageProcessor bgIp) { // background image
18        if(runDialog()) {
19            ImageProcessor fgIp
20                = fgIm.getProcessor().convertToByte(false);
21            fgIp = fgIp.duplicate();
22            fgIp.multiply(1-alpha);
23            bgIp.multiply(alpha);
24            bgIp.copyBits(fgIp, 0, 0, Blitter.ADD);
25        }
26    }
27
28    // continued ...
```



What Is Image Enhancement?

Image enhancement makes images more useful by:

- Highlighting interesting detail in images
- Removing noise from images
- Making images more visually appealing

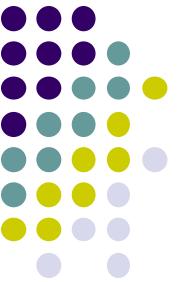


Image Enhancement Examples

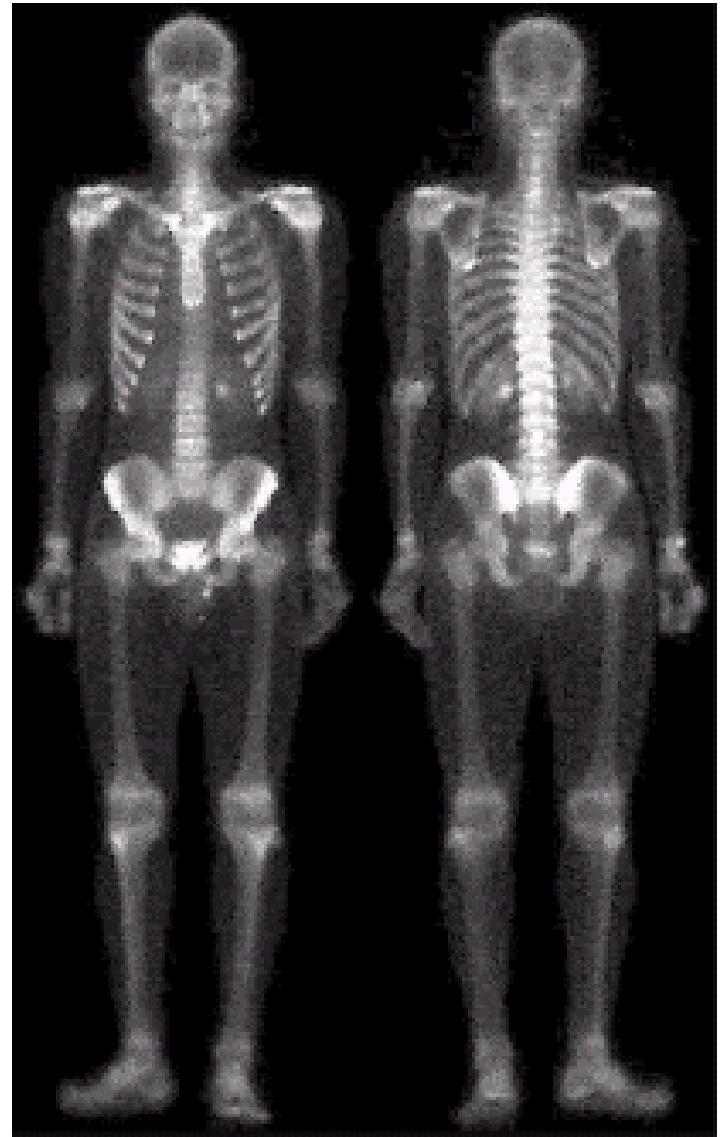
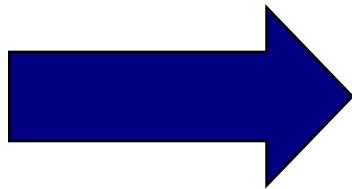
Images taken from Gonzalez & Woods, Digital Image Processing (2002)





Image Enhancement Examples (cont...)

Images taken from Gonzalez & Woods, Digital Image Processing (2002)



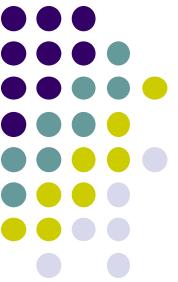


Image Enhancement Examples (cont...)

Images taken from Gonzalez & Woods, Digital Image Processing (2002)

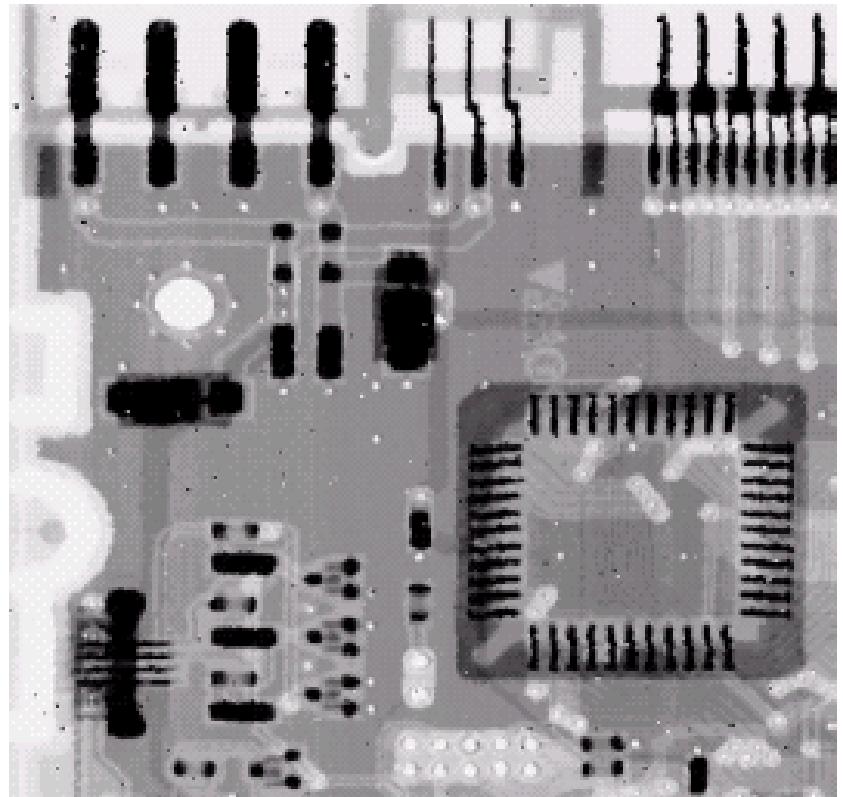
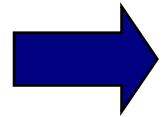
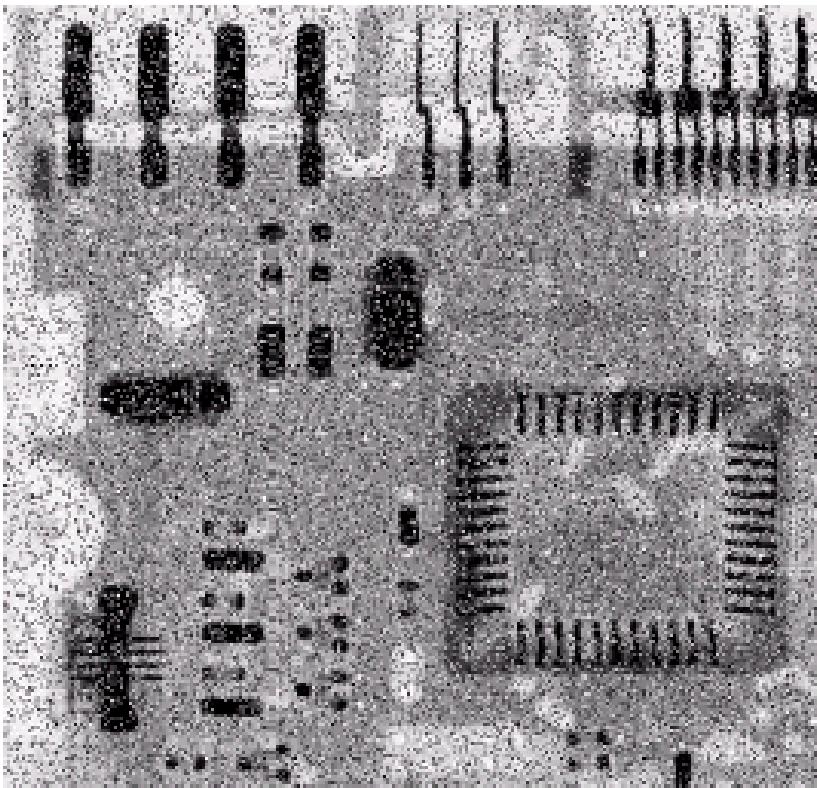




Image Enhancement Examples (cont...)

Images taken from Gonzalez & Woods, Digital Image Processing (2002)





Spatial & Frequency Domains

There are two broad categories of image enhancement techniques

- Spatial domain techniques
 - Direct manipulation of image pixels (intensity values)
- Frequency domain techniques
 - Manipulation of Fourier transform or wavelet transform of an image

First spatial domain techniques

Later: frequency domain techniques



What is a Filter?

- Capabilities of point operations are limited
- **Filters:** combine **pixel's value + values of neighbors**
- **E.g blurring:** Compute average intensity of block of pixels



- Combining multiple pixels needed for certain operations:
 - Blurring, Smoothing
 - Sharpening



What Point Operations Can't Do

- Example: sharpening





What Point Operations Can't Do

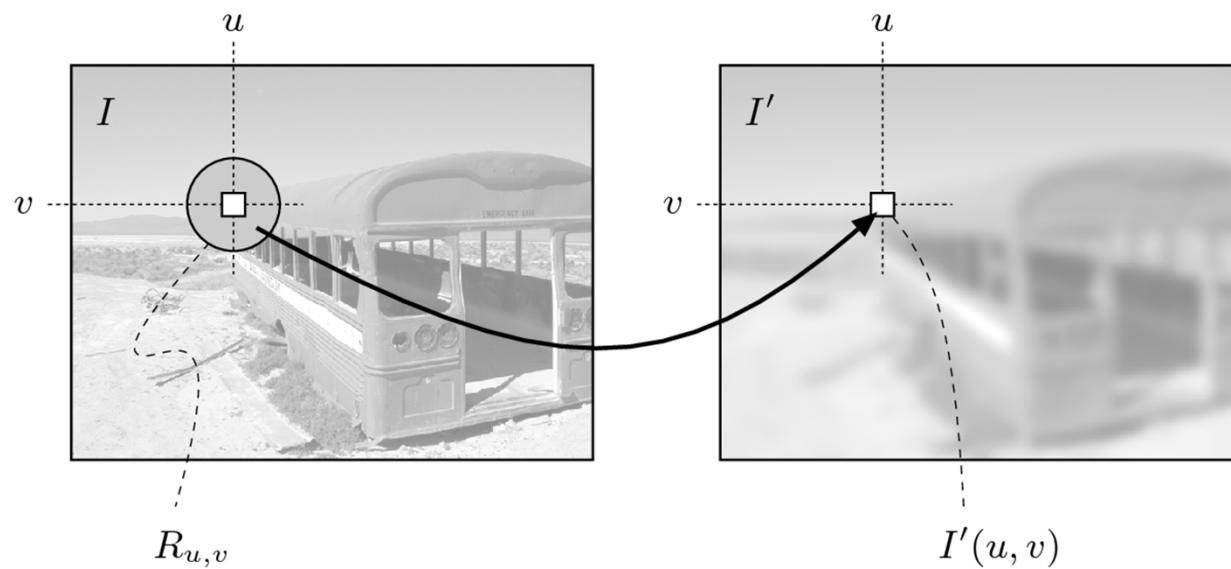
- Other cool artistic patterns by combining pixels



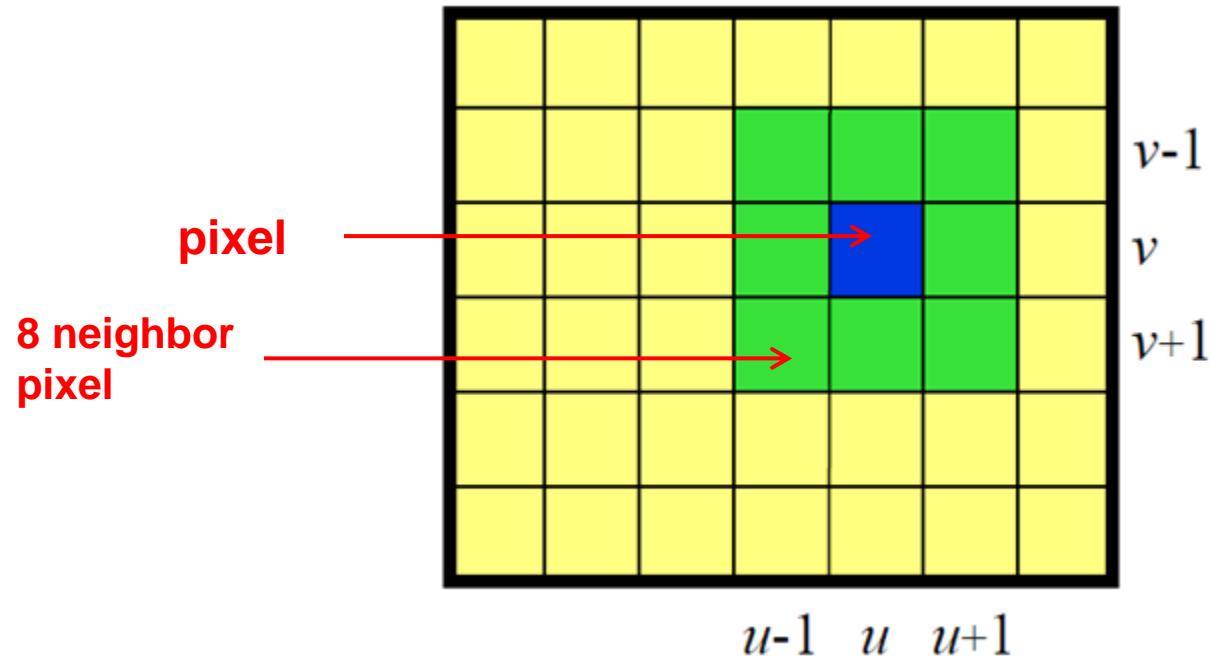
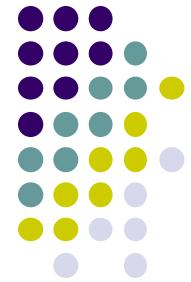


Definition: Spatial Filter

- An image operation that combines each pixel's intensity $I(u, v)$ with that of neighboring pixels
- E.g: average/weighted average of group of pixels



Example: Average (Mean) of 3x3 Neighborhood



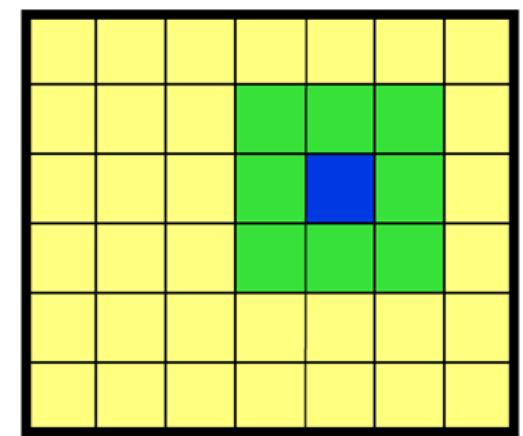
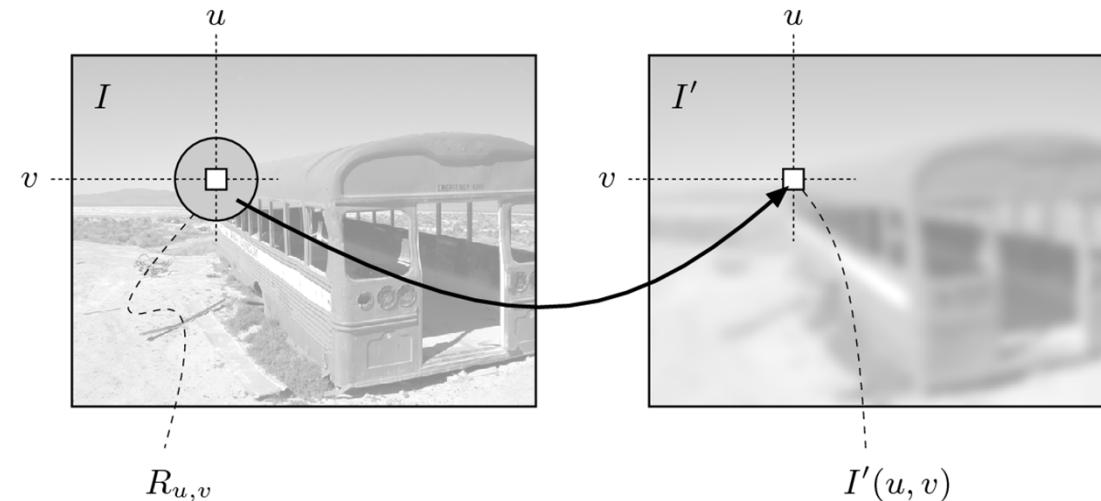
Blurring: Replace each pixel with AVERAGE Intensity of pixel + neighbors



Smoothing an Image by Averaging

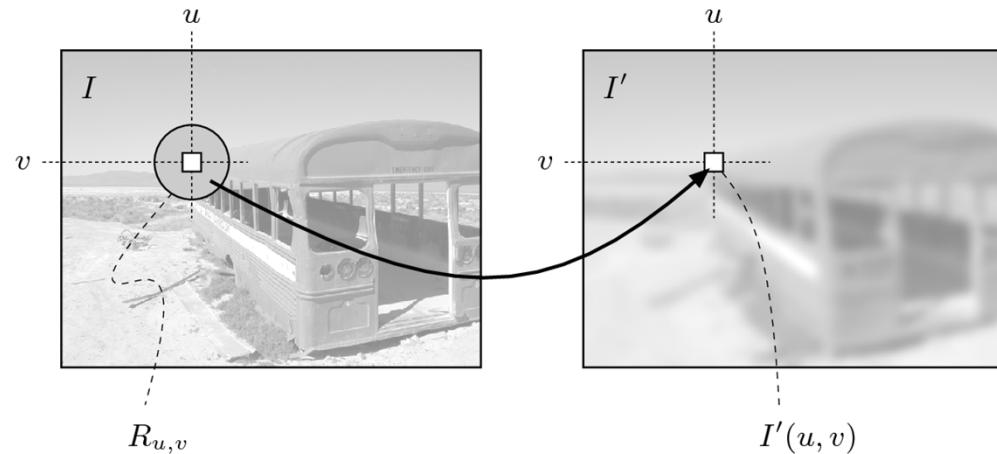
- Replace each pixel by average of pixel + neighbors
- For 3x3 neighborhood:

$$I'(u, v) \leftarrow \frac{p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8}{9}$$





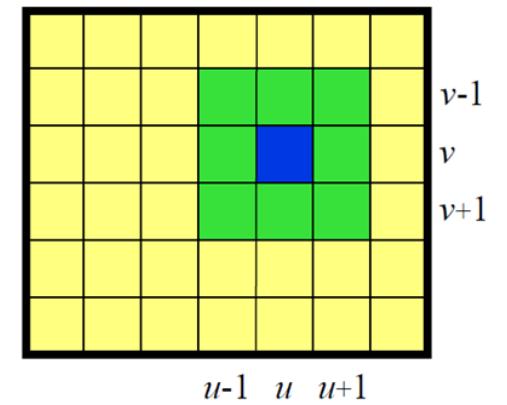
Smoothing an Image by Averaging



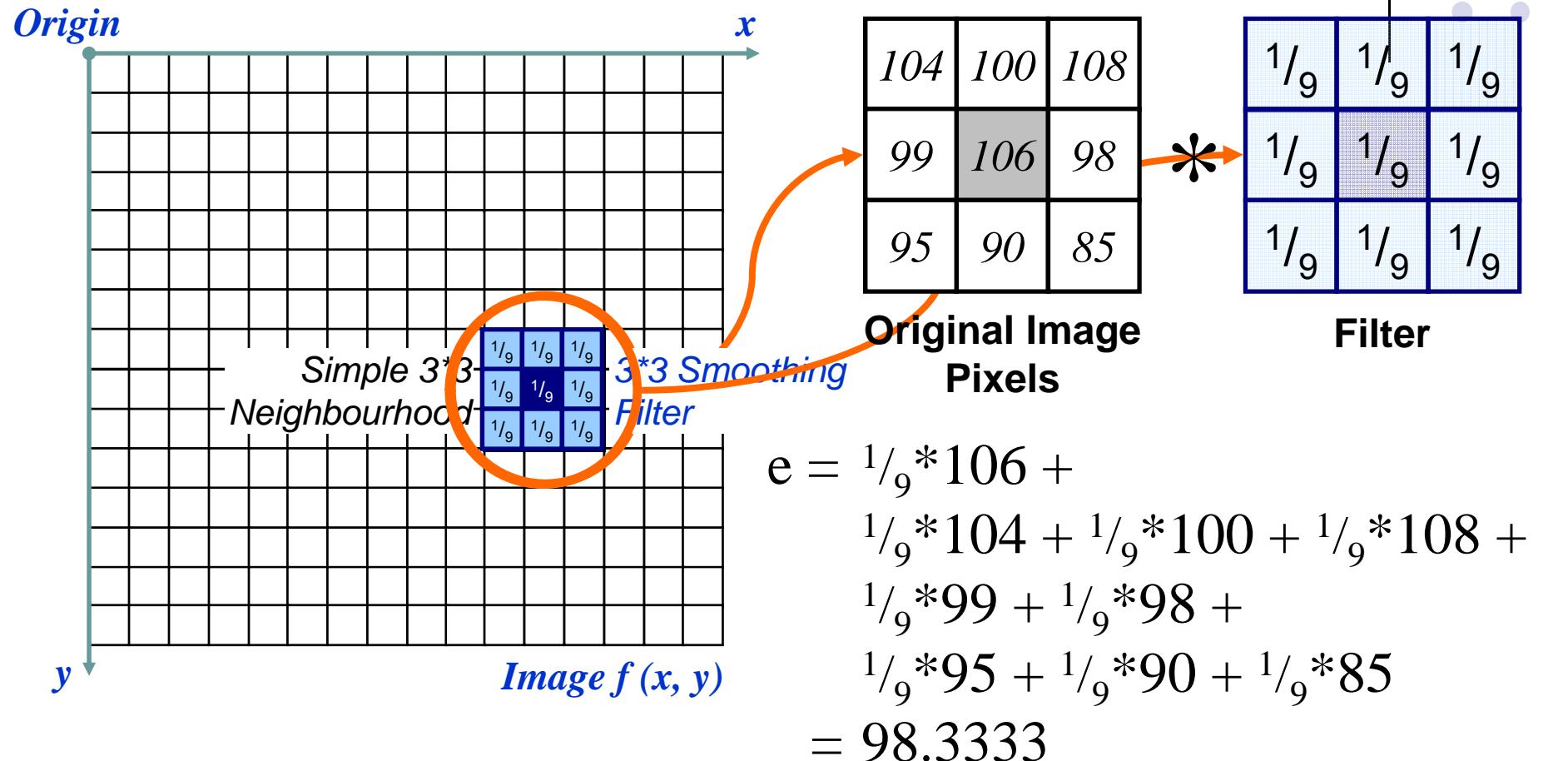
$$I'(u, v) \leftarrow \frac{p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8}{9}$$

$$I'(u, v) \leftarrow \frac{1}{9} \cdot [I(u-1, v-1) + I(u, v-1) + I(u+1, v-1) + I(u-1, v) + I(u, v) + I(u+1, v) + I(u-1, v+1) + I(u, v+1) + I(u+1, v+1)]$$

$$I'(u, v) \leftarrow \frac{1}{9} \cdot \sum_{j=-1}^1 \sum_{i=-1}^1 I(u+i, v+j)$$

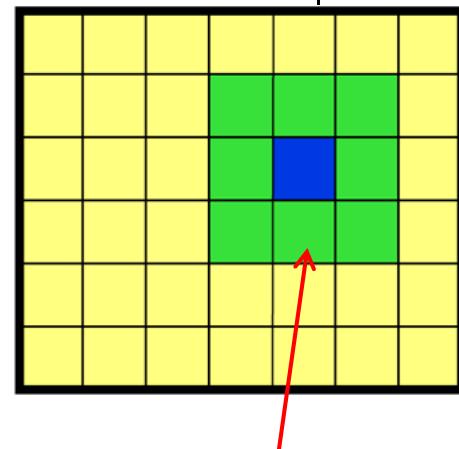
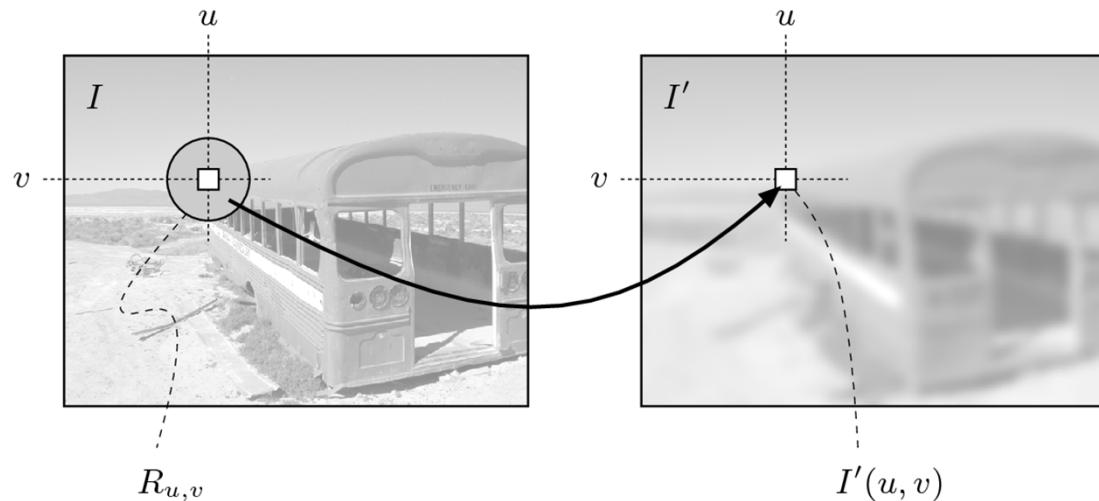
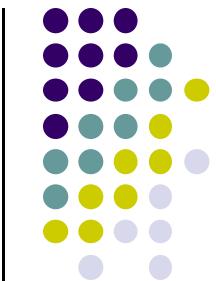


Example: Smoothing Spatial Filtering



The above is repeated for every pixel in the original image to generate the smoothed image

Smoothing an Image by Averaging



Previous example:
Filter size: 3x3

- Many possible filter parameters (size, weights, function, etc)
- **Filter size (size of neighborhood):** 3x3, 5x5, 7x7, ..., 21x21,..
- **Filter shape:** not necessarily square. Can be rectangle, circle, etc
- **Filter weights:** May apply unequal weighting to different pixels
- **Filters function:** can be linear (a weighted summation) or nonlinear



The Filter Matrix

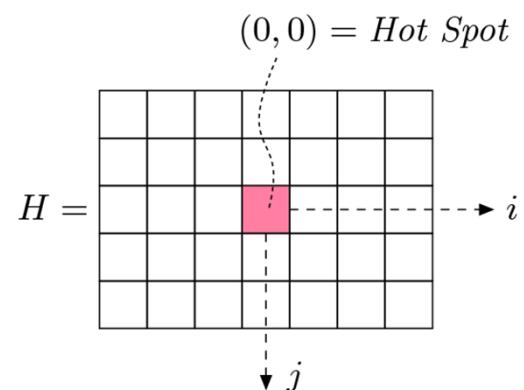
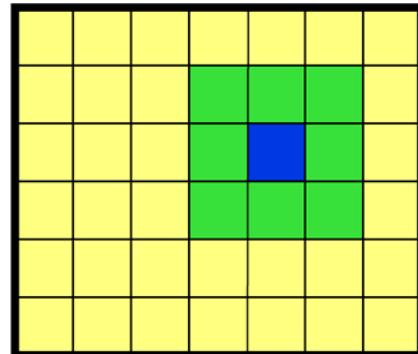
$$I'(u, v) \leftarrow \frac{p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8}{9}$$

$$I'(u, v) \leftarrow \frac{1}{9} \cdot [I(u-1, v-1) + I(u, v-1) + I(u+1, v-1) + \\ I(u-1, v) + I(u, v) + I(u+1, v) + \\ I(u-1, v+1) + I(u, v+1) + I(u+1, v+1)]$$

$$H(i, j) = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Filter operation can be expressed as a matrix
Example: averaging filter

Filter matrix also called filter mask $H(i,j)$





Example: What does this Filter Do?



0	0	0
0	1	0
0	0	0



Identity function (leaves image alone)



What Does this Filter Do?



$$\frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$



Mean (averages neighborhood)



Mean Filters: Effect of Filter Size



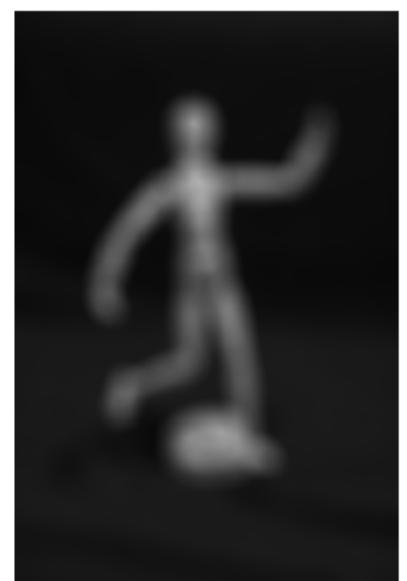
Original



7×7



15×15



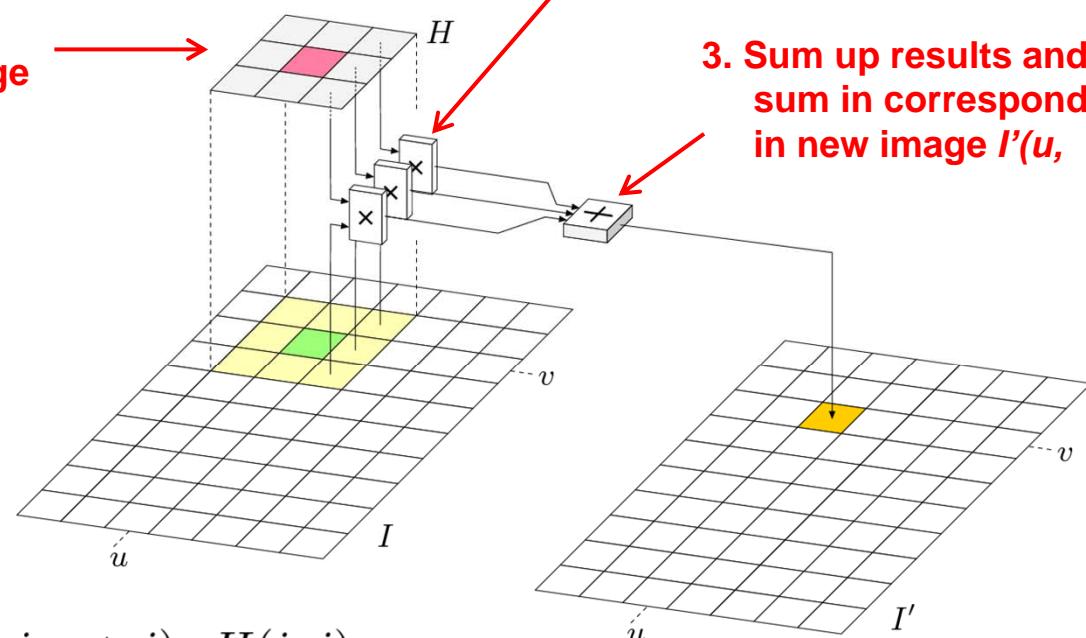
41×41



Applying Linear Filters: Convolution

For each image position $I(u, v)$:

1. Move filter matrix H over image such that $H(0,0)$ coincides with current image position (u, v)



Stated formally:

$$I'(u, v) \leftarrow \sum_{(i,j) \in R_H} I(u + i, v + j) \cdot H(i, j)$$

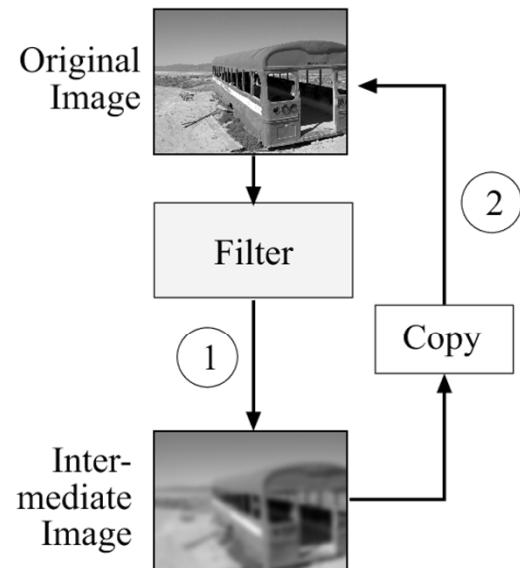
R_H is set of all pixels
Covered by filter.
For 3x3 filter, this is:

$$I'(u, v) \leftarrow \sum_{i=-1}^{i=1} \sum_{j=-1}^{j=1} I(u + i, v + j) \cdot H(i, j)$$



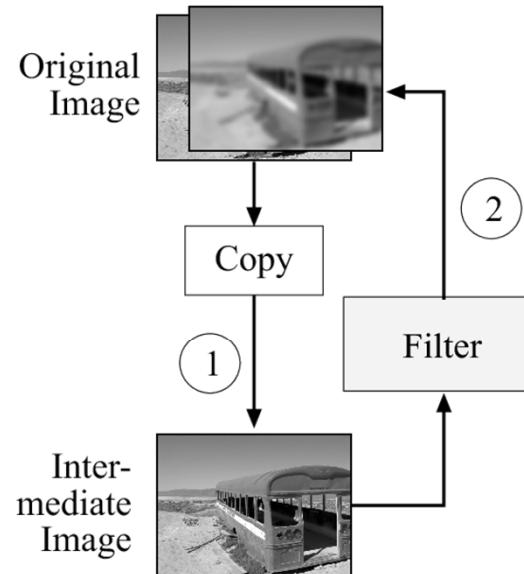
Computing Filter Operation

- Filter matrix H moves over each pixel in original image I to compute corresponding pixel in new image I'
- Cannot overwrite new pixel value in original image I Why?



Version A

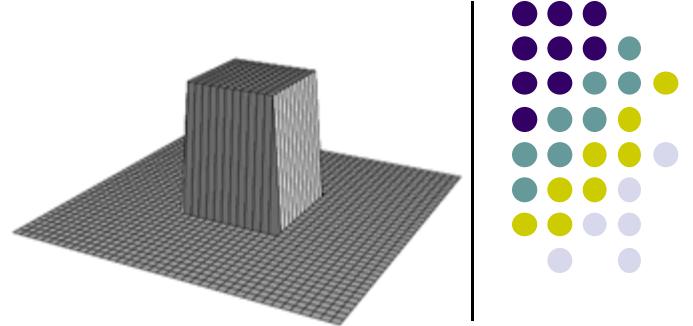
Store results I' in intermediate image, then copy back to replace I



Version B

Copy original image I to intermediate image, use it as source, then store results I' to replace original image

Simple 3x3 Averaging Filter (“Box” Filter)



```
1 import ij.*;
2 import ij.plugin.filter.PlugInFilter;
3 import ij.process.*;
4
5 public class Filter_Average3x3 implements PlugInFilter {
6     ...
7     public void run(ImageProcessor orig) {
8         int w = orig.getWidth();
9         int h = orig.getHeight();
10        ImageProcessor copy = orig.duplicate(); ←
11
12        for (int v = 1; v <= h-2; v++) {
13            for (int u = 1; u <= w-2; u++) { ←
14                //compute filter result for position (u,v)
15                int sum = 0;
16                for (int j = -1; j <= 1; j++) {
17                    for (int i = -1; i <= 1; i++) {
18                        int p = copy.getPixel(u+i, v+j);
19                        sum = sum + p;
20                    }
21                }
22                int q = (int) Math.round(sum/9.0);
23                orig.putPixel(u, v, q); ←
24            }
25        }
26    }
27 } // end of class Filter_Average3x3
```

No explicit filter matrix since all coefficients are the same (1/9)

No clamping required

Make copy of original image to use as source

Loop over all pixels in image

Filter computation by adding current pixel's neighbors

Store result back in original image



Weighted Smoothing Filters

- More effective smoothing filters can be generated by allowing different pixels in the neighbourhood different weights in the averaging function
 - Pixels closer to central pixel more important
 - Often referred to as a *weighted averaging*

$1/_{16}$	$2/_{16}$	$1/_{16}$
$2/_{16}$	$4/_{16}$	$2/_{16}$
$1/_{16}$	$2/_{16}$	$1/_{16}$

Weighted
averaging filter



Another Smoothing Filter

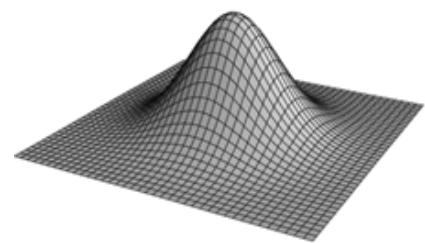
```

1  public void run(ImageProcessor orig) {
2      int w = orig.getWidth();
3      int h = orig.getHeight();
4      // 3 × 3 filter matrix
5      double[][] filter = {
6          {0.075, 0.125, 0.075},
7          {0.125, 0.200, 0.125},
8          {0.075, 0.125, 0.075}
9      };
10     ImageProcessor copy = orig.duplicate();
11
12     for (int v = 1; v <= h-2; v++) {
13         for (int u = 1; u <= w-2; u++) {
14             // compute filter result for position (u, v)
15             double sum = 0;
16             for (int j = -1; j <= 1; j++) {
17                 for (int i = -1; i <= 1; i++) {
18                     int p = copy.getPixel(u+i, v+j);
19                     // get the corresponding filter coefficient:
20                     double c = filter[j+1][i+1];
21                     sum = sum + c * p;
22                 }
23             }
24             int q = (int) Math.round(sum);
25             orig.putPixel(u, v, q);
26         }
27     }
28 }
```

Use real filter matrix with coefficients
Apply bell-shaped function $H(i,j)$

$$H(i, j) = \begin{bmatrix} 0.075 & 0.125 & 0.075 \\ 0.125 & \textbf{0.2} & 0.125 \\ 0.075 & 0.125 & 0.075 \end{bmatrix}$$

Bell-shaped function $H(i,j)$?
- More weight applied to center



Apply filter

Store result back in original



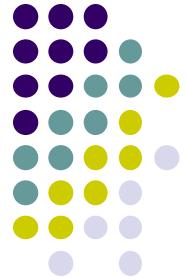
Integer Coefficients

- Instead of floating point coefficients, more efficient, simpler to use:

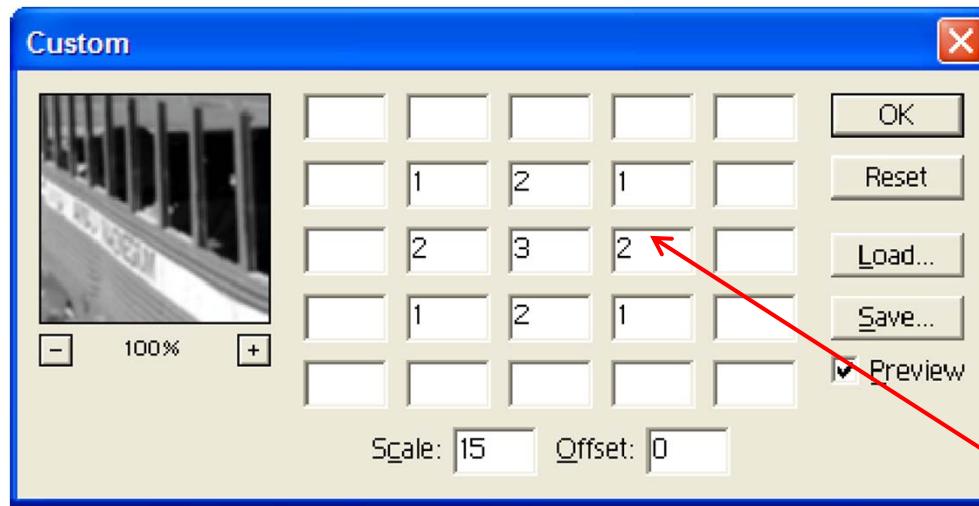
scaling factor + integer coefficients

$$H(i, j) = \begin{bmatrix} 0.075 & 0.125 & 0.075 \\ 0.125 & \underline{0.200} & 0.125 \\ 0.075 & 0.125 & 0.075 \end{bmatrix} = \frac{1}{40} \begin{bmatrix} 3 & 5 & 3 \\ 5 & \underline{8} & 5 \\ 3 & 5 & 3 \end{bmatrix}$$

$$H(i, j) = s \cdot H'(i, j)$$



Example: 5x5 Filter in Adobe Photoshop



Integer filter
coefficients

$$I'(u, v) \leftarrow \text{Offset} + \frac{1}{\text{Scale}} \sum_{j=-2}^{j=2} \sum_{i=-2}^{i=2} I(u+i, v+j) \cdot H(i, j)$$

If resulting pixel value is
negative, offset shifts it
into visible range

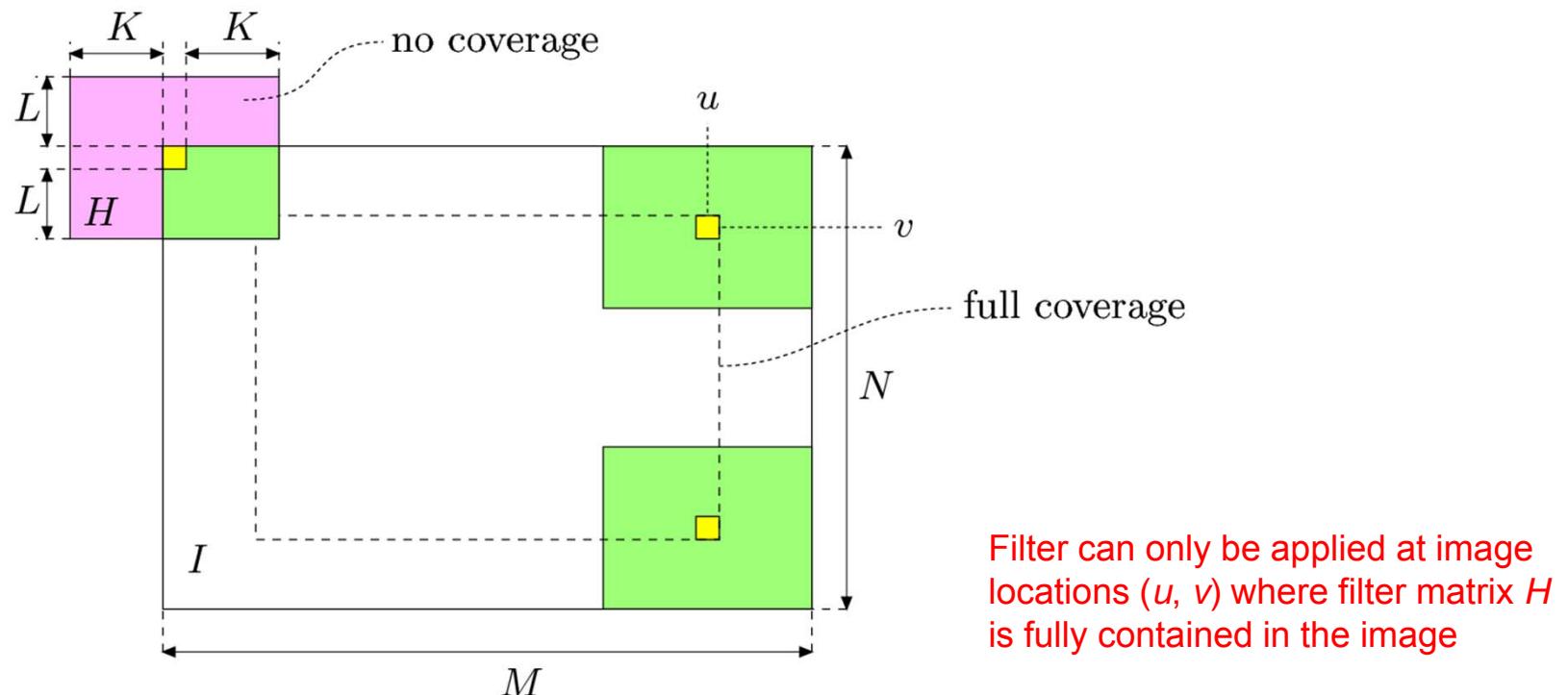
Scaling factor for
coefficients



Computation Range

- For a filter of size $(2K+1) \times (2L+1)$, if image size is $M \times N$, filter is computed over the range:

$$K \leq u' \leq (M-K-1) \quad \text{and} \quad L \leq v' \leq (N-L-1)$$





Filter of Arbitrary Size

```
1  public void run(ImageProcessor orig) {
2      int M = orig.getWidth();
3      int N = orig.getHeight();
4
5      // filter matrix of size  $(2K + 1) \times (2L + 1)$ 
6      int[][] filter = {
7          {0,0,1,1,1,0,0},
8          {0,1,1,1,1,1,0},
9          {1,1,1,1,1,1,1},
10         {0,1,1,1,1,1,0},
11         {0,0,1,1,1,0,0}
12     };
13     double s = 1.0/23; // sum of filter coefficients is 23
14
15     int K = filter[0].length/2;
16     int L = filter.length/2;
17
18     ImageProcessor copy = orig.duplicate();
19
20     for (int v = L; v <= N-L-1; v++) {
21         for (int u = K; u <= M-K-1; u++) {
22             // compute filter result for position (u, v)
23             int sum = 0;
24             for (int j = -L; j <= L; j++) {
25                 for (int i = -K; i <= K; i++) {
26                     int p = copy.getPixel(u+i, v+j);
27                     int c = filter[j+L][i+K];
28                     sum = sum + c * p;
29                 }
30             }
31             int q = (int) Math.round(s * sum);
32             if (q < 0) q = 0; ←
33             if (q > 255) q = 255;
34             orig.putPixel(u, v, q);
35         }
36     }
37 }
```

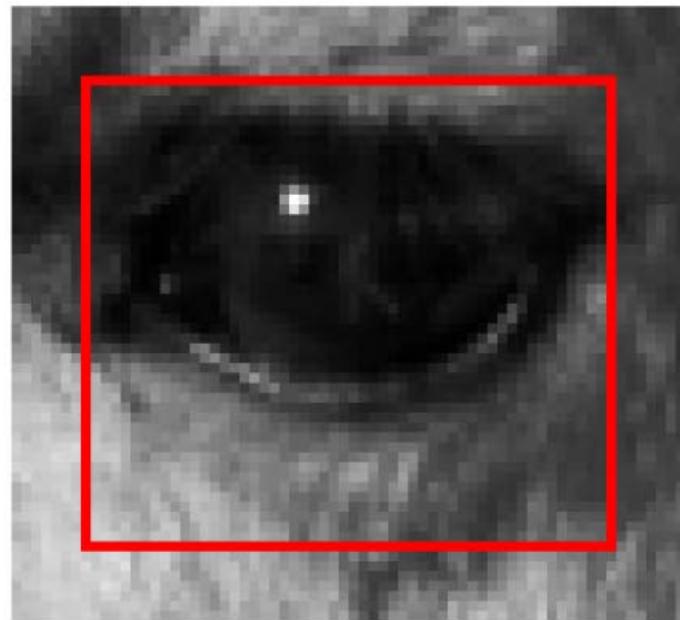
Declare filter

Clamp result as a
precaution



What to do at image boundaries?

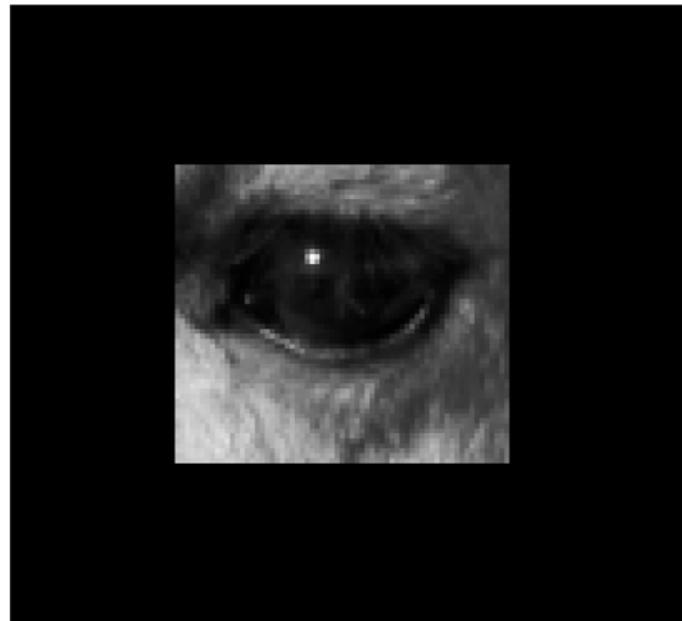
a) Crop





What to do at image boundaries?

- a) Crop
- b) Pad





What to do at image boundaries?

- a) Crop
- b) Pad
- c) Extend





What to do at image boundaries?

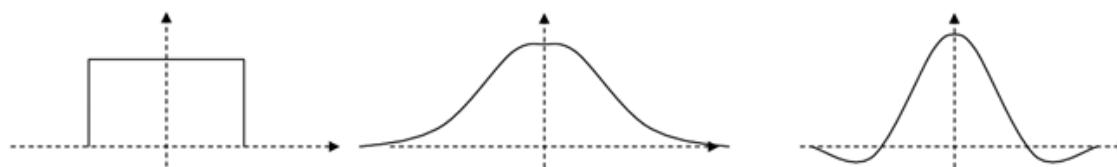
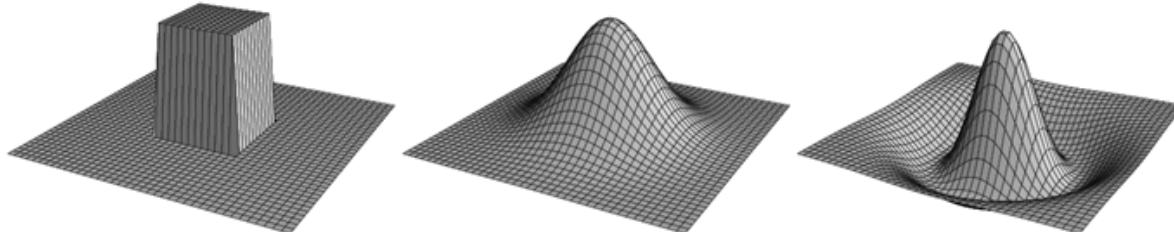
- a) Crop
- b) Pad
- c) Extend
- d) Wrap





Linear Filters: Smoothing Filters

- 2 main classes of linear filters:
 - **Smoothing:** +ve coefficients (weighted average). E.g box, gaussian
 - **Difference** filters: +ve and -ve weights. E.g. Laplacian



0	0	0	0	0	0
0	1	1	1	1	0
0	1	1	1	1	0
0	1	1	1	1	0
0	0	0	0	0	0

(a)

Box

0	1	2	1	0
1	3	5	3	1
2	5	9	5	2
1	3	5	3	1
0	1	2	1	0

(b)

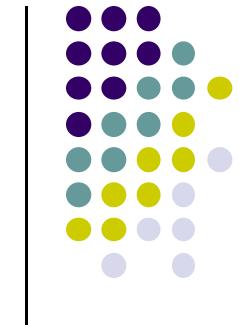
Gaussian

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

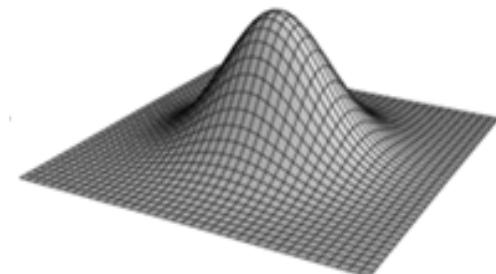
(c)

Laplacian

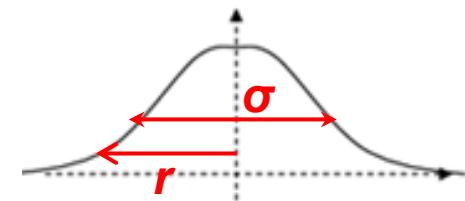
Gaussian Filter



$$G_\sigma(r) = e^{-\frac{r^2}{2\sigma^2}} \quad \text{or} \quad G_\sigma(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$



- where
 - σ is width (standard deviation)
 - r is distance from center



0	1	2	1	0
1	3	5	3	1
2	5	9	5	2
1	3	5	3	1
0	1	2	1	0

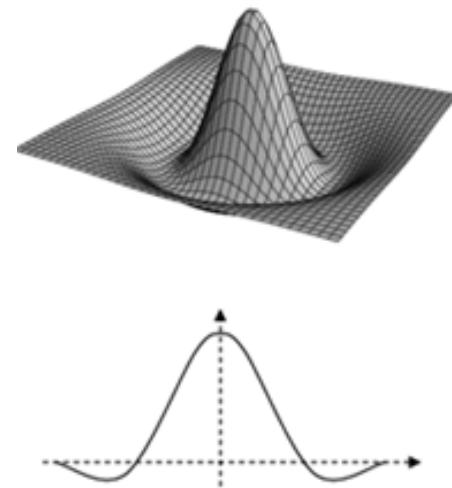
Gaussian
filter



Difference Filters

- **Coefficients:** some +ve, some negative
- Example: Laplacian filter
- Computation is difference

$$\sum (+ve \text{ coefficients}) - \sum (-ve \text{ coefficients})$$



$$I'(u, v) = \sum_{(i,j) \in R_H^+} I(u+i, v+j) \cdot |H(i, j)| - \sum_{(i,j) \in R_H^-} I(u+i, v+j) \cdot |H(i, j)|$$

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

Laplacian
filter



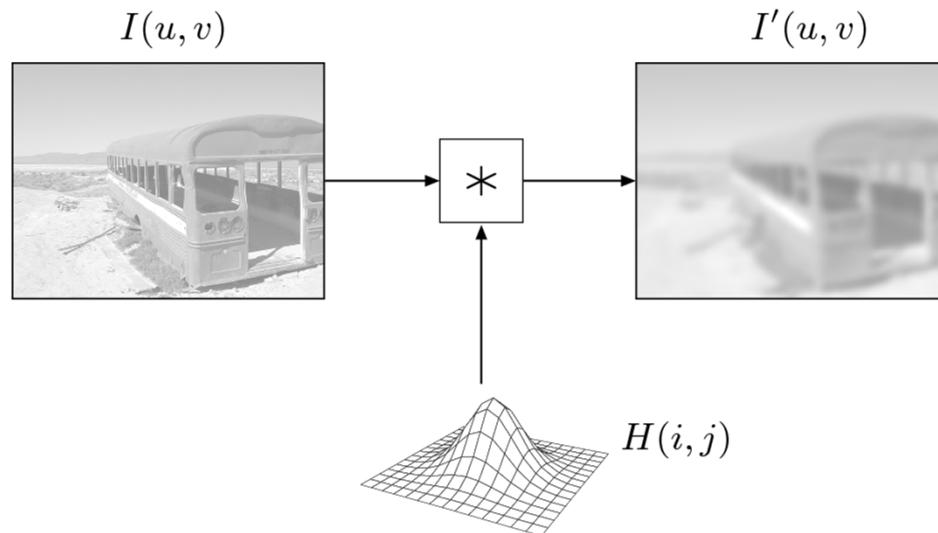
Mathematical Properties of Convolution

- Applying a filter as described called ***linear convolution***
- For discrete 2D signal, convolution defined as:

$$I'(u, v) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(u-i, v-j) \cdot H(i, j)$$

Formal definition:
Sum to $\pm \infty$

$$I' = I * H$$





Properties of Convolution

- Commutativity

$$I * H = H * I$$

Same result if we convolve
image with filter or vice versa

- Linearity

$$(s \cdot I) * H = I * (s \cdot H) = s \cdot (I * H)$$

If image multiplied by scalar
Result multiplied by same scalar

$$(I_1 + I_2) * H = (I_1 * H) + (I_2 * H)$$

(notice)

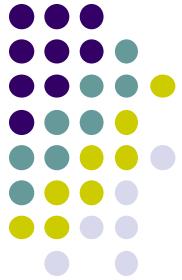
$$(b + I) * H \neq b + (I * H)$$

If 2 images added and convolve
result with a kernel H ,
Same result if each image
is convolved individually + added

- Associativity

$$A * (B * C) = (A * B) * C$$

Order of filter application irrelevant
Any order, same result



References

- Wilhelm Burger and Mark J. Burge, Digital Image Processing, Springer, 2008
 - Histograms (Ch 4)
 - Point operations (Ch 5)
 - Filters (Ch 6)
- University of Utah, CS 4640: Image Processing Basics, Spring 2012
- Rutgers University, CS 334, Introduction to Imaging and Multimedia, Fall 2012