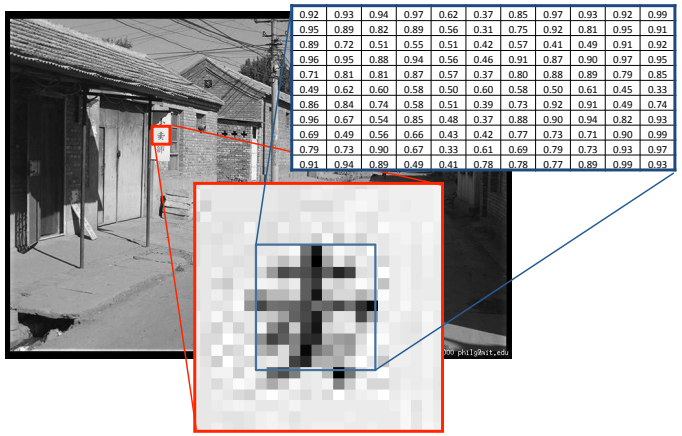
Image Filtering



Reading: Sections 3.1, 3.2 and 10.3.1 in Szeliski book

A Digital Image is a Matrix of Pixels



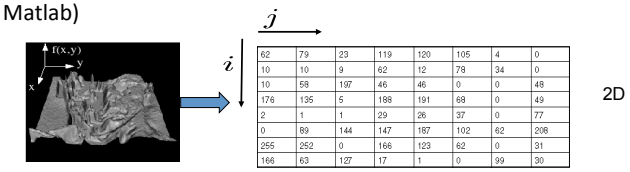
Source: D. Hoiem

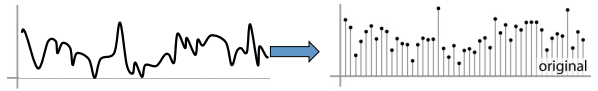
Pixels Measure Brightness over an Area



Digital Images

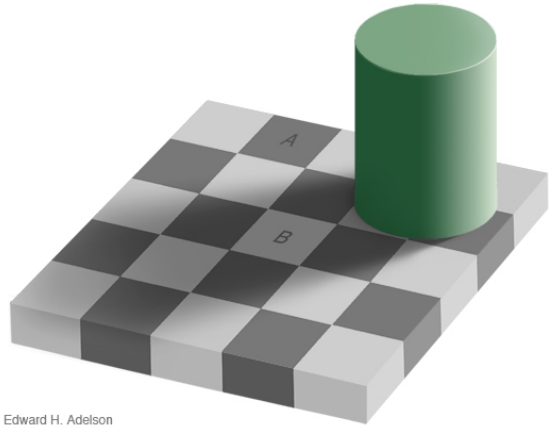
- **Sample** the 2D space on a regular grid (the "pixels")
- **Quantize** each sample (often, 8 or 24 bits per pixel) (the "brightness", "intensity" or "gray level")
- Image represented as a (row, column) matrix of integer values (in Matlab)





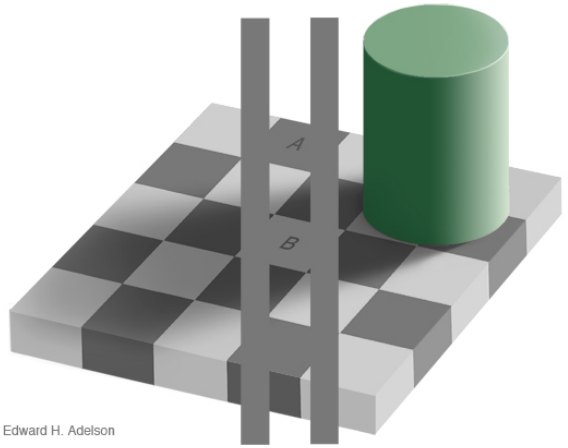
Source: S. Seitz

Perception of Intensity



Edward H. Adelson

Perception of Intensity

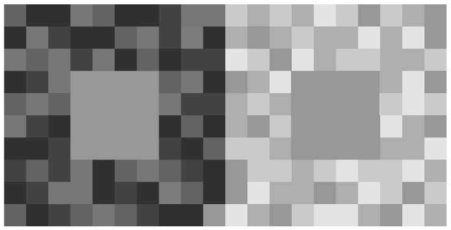


Edward H. Adelson

Simultaneous Contrast Effect

VMeisoft

An articulated background enhances the illusion.



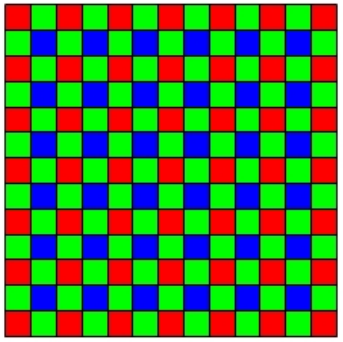
Source: E. Adelson

Color Image



Source: D. Hoiem

Digital Color Images

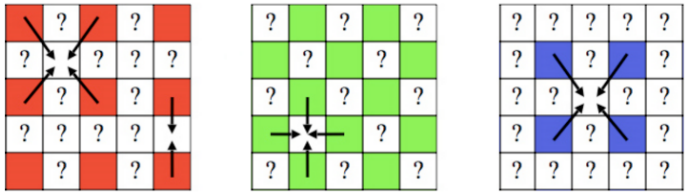


Bayer filter

© 2000 How Stuff Works

Source: D. Hoiem

Interpolate Values to Create the 3 RGB Channels



Images in Matlab

- Image represented as a matrix
- Suppose we have an *n* x *m* RGB image called "im"
  - im(1,1,1) = top-left pixel value in R-channel
  - im(y, x, b) = y pixels down, x pixels to right in the bth channel
  - im(*n*, *m*, 3) = bottom-right pixel in B-channel
- `imread(filename)` returns a `uint8` image (values 0 to 255 for each color)
  - Convert to double format (values 0 … 1) with `im2double`

Except returns type `logical` if 1 bpp

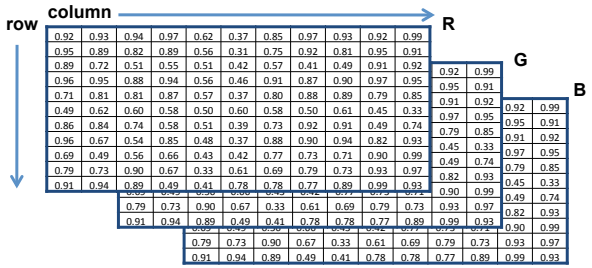


Image Filtering

- **Image filtering**: computes a function of a *local neighborhood* at each pixel position
- Called "**Local operator**," "**Neighborhood operator**," or "**Window operator**"
- *h*: image ➔ image

- Uses:
  - **Enhance images**
    - Noise reduction, smooth, resize, increase contrast, recolor, artistic effects, etc.
  - **Extract features from images**
    - Texture, edges, distinctive points, etc.
  - **Detect patterns**
    - Template matching, e.g., eye template

Source: D. Hoiem

Point Operations

- Map each pixel's value to a new value
- Neighborhood is 1 x 1
- $g(i,j) = h(f(i,j))$ where f is the input image, g is the output (i.e., transformed) image, and h is the point operator / transformation
- Examples
  - $g(i,j) = af(i,j) + b$ where $a > 0$ is a ***gain*** parameter and b controls the ***brightness***
  - Mapping one color space to another, e.g., RGB ⇒ HSV
  - Image rotation, translation, scale change, …

Instagram Filters

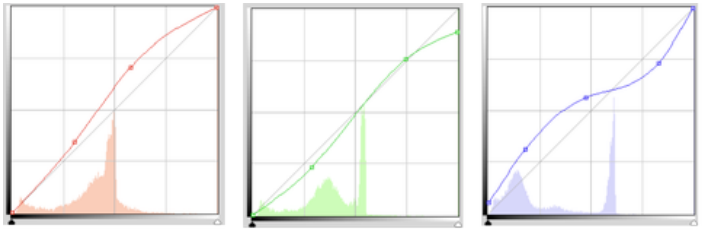
- How do they make those Instagram filters?



"It's really a combination of a bunch of different methods. In some cases we draw on top of images, in others we do pixel math. It really depends on the effect we're going for." --- Kevin Systrom, co-founder of Instagram

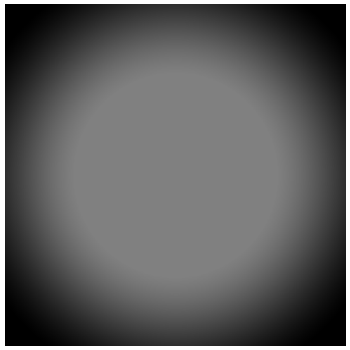
Example Instagram Steps

1. Perform an independent RGB color point transformation on the original image to increase contrast or make a color cast



Example Instagram Steps

2. Overlay a circle background image to create a vignette effect



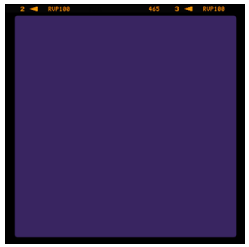
Example Instagram Steps

3. Overlay a background image as decorative grain



Example Instagram Steps

4. Add a border or frame



Result

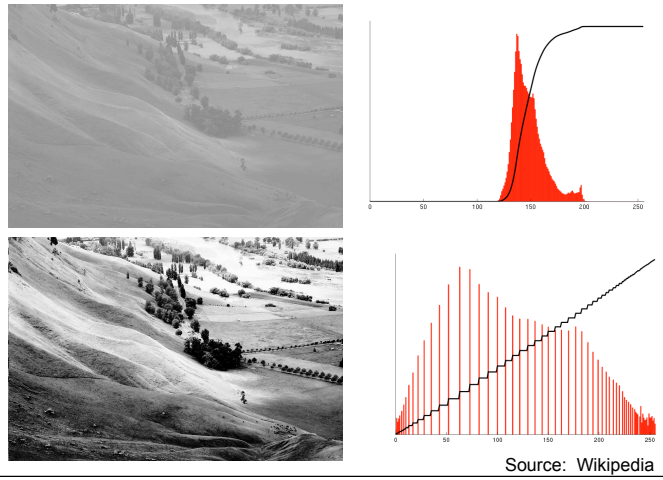


Javascript library for creating Instagram-like effects, see:
http://alexmic.net/filtrr/

Histogram Equalization / Flattening

- Transform image by modifying its histogram
  - create a lookup table defining *h*
- Why?
  - Image normalization
  - Comparing images
  - Contrast enhancement
- When an image's histogram is transformed so that all gray levels occur about equally often, the result tends to produce an image with higher contrast. Why?

Histogram Equalization



Source: Wikipedia

Histogram Equalization Algorithm

Goal: Given an m x n image, f, with 8 bpp (gray levels 0 .. 255), create a new image, g, that has about $mn/256$ pixels with each gray level value

1. Compute f's histogram: $h(i)$ = number of pixels with gray level i for i = 0..255
2. Compute f's cumulative histogram: $c(i) = \sum_{j=0..i} h(j)$
   Ideally, image g has $c(k) = (mn/256)*k$
3. Compute transformation $k = t(i) = (256/mn)*c(i)$ for i = 0..255
4. Create output image: $g(i, j) = t(f(i, j))$

Note: Max value in Step 3 above is 256, but legal values are 0..255, so subtract 1

Color Transfer

- Goal: Change the colors of a given "source" image to match the color "palette" of another image in order to transfer the "mood" or "style" of one image to another
- One simple method: Shift and scale the pixel values of the target image to match the mean and standard deviation of the source image
- Color space representation of images affects quality of results. One good one: "Lab" color space.

"Lab" Color Representation



A transformation of the colors into a color space that is more perceptually meaningful:
L: luminance,
a: red − green,
b: blue − yellow

Color Transfer

Let *I* = Source image, i.e., one being recolored

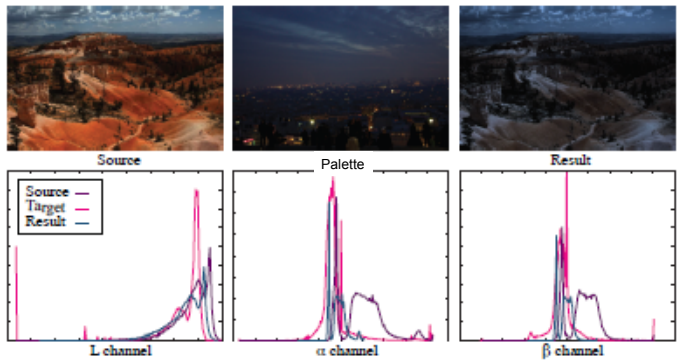
Let *J* = Palette image, i.e., one colors taken from

1. Convert *I* and *J* to *Lab* color space; call result images *S* and *P,* respectively
2. Compute *L* channel output image matrix:

$$L_{out} = \frac{\sigma_P}{\sigma_S}(L_S - \mu_S) + \mu_P$$

3. Similarly, compute *a* and *b* channels
4. Combine *L*, *a* and *b*, and convert back to *RGB*

Color Transfer

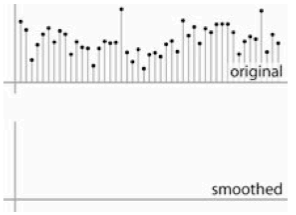


Local / Neighborhood Operations

- Value of pixel in output image is a function of the corresponding pixel in the input image plus other nearby pixels (usually defined by a *square or rectangular "window"* centered on the given pixel)

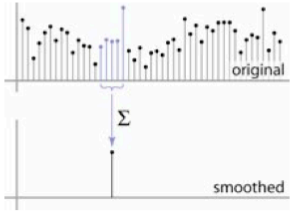
Linear Filtering

- basic idea: define a new function by averaging over a sliding window
- a simple example to start off: smoothing



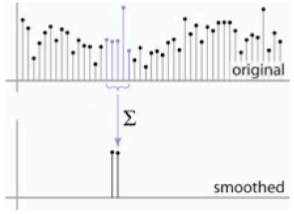
Linear Filtering

- basic idea: define a new function by averaging over a sliding window
- a simple example to start off: smoothing



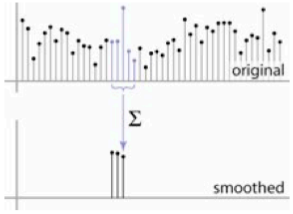
Linear Filtering

- basic idea: define a new function by averaging over a sliding window
- a simple example to start off: smoothing



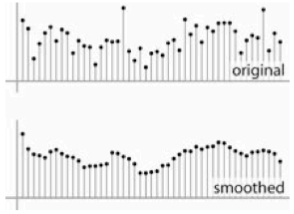
Linear Filtering

- basic idea: define a new function by averaging over a sliding window
- a simple example to start off: smoothing



Linear Filtering

- basic idea: define a new function by averaging over a sliding window
- a simple example to start off: smoothing



Linear Filtering

- Same moving average operation, expressed mathematically:

$$b_{\text{smooth}}[i] = \frac{1}{2r+1} \sum_{j=i-r}^{i+r} b[j]$$

Original image

Linear Filtering

- Simple averaging:

$$b_{\text{smooth}}[i] = \frac{1}{2r+1} \sum_{j=i-r}^{i+r} b[j]$$

every sample gets the same weight

- Convolution: same idea but with *weighted* average

$$(a \star b)[i] = \sum_j a[j]b[i-j]$$

each sample gets its own weight (normally zero far away)

- This is all convolution is: it is a **moving weighted average**

Image Filtering

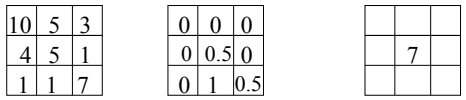
- Modify the pixels in an image based on some function of a local neighborhood of the pixels
- Simplest: **linear filtering**
  - *Replace each pixel by a linear combination of its neighbors*



Original, local image data — Output image data

Linear Functions

- Simplest: **linear filtering**
  - *Replace each pixel by a linear combination of its neighbors*



Local image data — kernel — Output image data

2D Example: Box Filter

$g[\cdot,\cdot]$

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Credit: David Lowe

Image Filtering

Filter

$g[\cdot,\cdot]\ \frac{1}{9}$

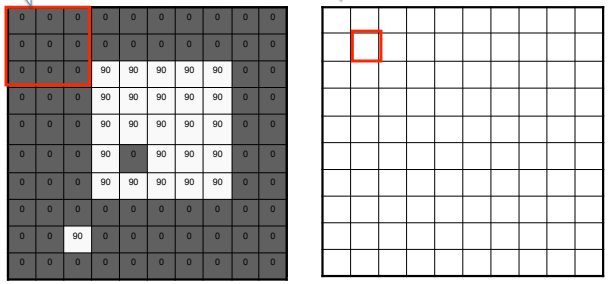
1	1	1
1	1	1
1	1	1

Input Image

$f[.,.]$

Output Image

$h[.,.]$



$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

Credit: S. Seitz

Image Correlation Filtering

- Center filter *g* at each pixel in image *f*
- Multiply weights by corresponding pixels
- Set resulting value in output image *h*
- *g* is called a ***filter***, ***mask***, ***kernel***, or ***template***
- Linear filtering is sum of dot product at each pixel position
- Filtering operation called ***cross-correlation***, and denoted $h = f \otimes g$

Image Filtering

f

10	11	10	0	0	1
9	10	11	1	0	1
10	9	10	0	2	1
11	10	9	10	9	11
9	10	11	9	99	11
10	9	9	11	10	10

g

1	1	1
1	1	1
1	1	1

1/9

h

X	X	X	X	X	X
X	10				X
X					X
X					X
X					X
X	X	X	X	X	X

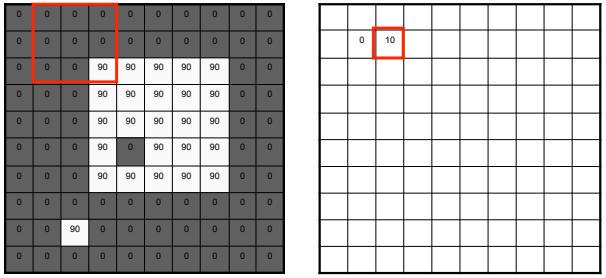
1/9(10x1 + 11x1 + 10x1 + 9x1 + 10x1 + 11x1 + 10x1 + 9x1 + 10x1) =
1/9(90) = 10

Credit: C. Rasmussen

Image Filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$ 1 1 1 / 1 1 1 / 1 1 1

$f[.,.]$ $h[.,.]$



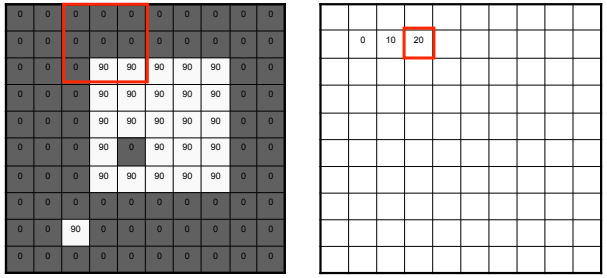
$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Credit: S. Seitz

Image Filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$ 1 1 1 / 1 1 1 / 1 1 1

$f[.,.]$ $h[.,.]$



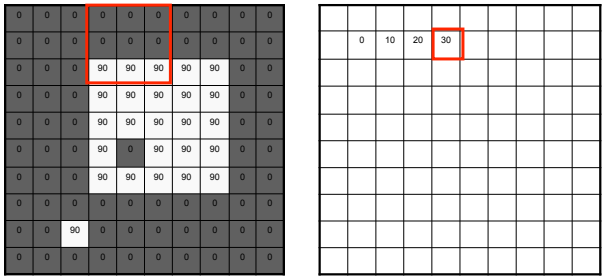
$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Credit: S. Seitz

Image Filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$ 1 1 1 / 1 1 1 / 1 1 1

$f[.,.]$ $h[.,.]$



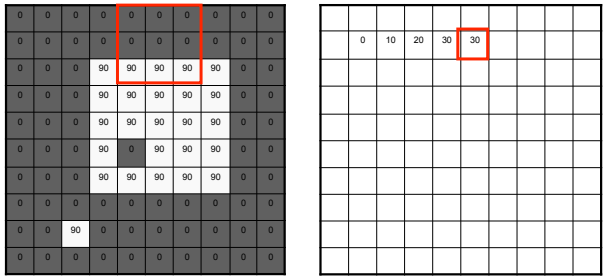
$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Credit: S. Seitz

Image Filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$ 1 1 1 / 1 1 1 / 1 1 1

$f[.,.]$ $h[.,.]$



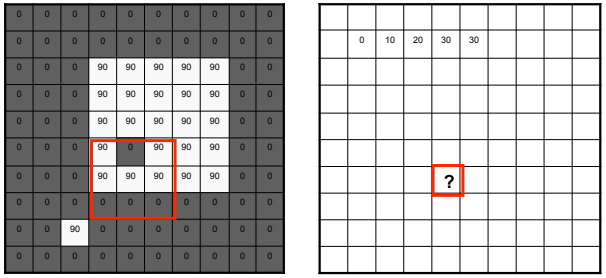
$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Credit: S. Seitz

Image Filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$ [1 1 1; 1 1 1; 1 1 1]

$f[.,.]$ $h[.,.]$

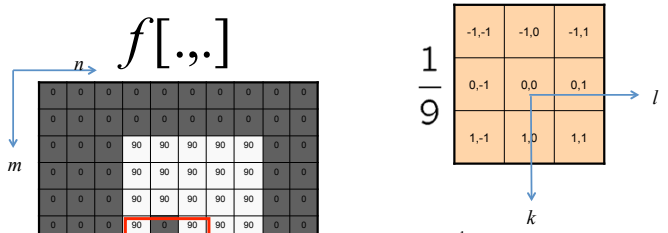


$$h[m,n]=\sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Credit: S. Seitz

Image Filtering

$g[\cdot,\cdot]$



$f[.,.]$

$$h[m,n]=\sum_{k,l=-1}^{+1} g[k,l]\, f[m+k,n+l]$$

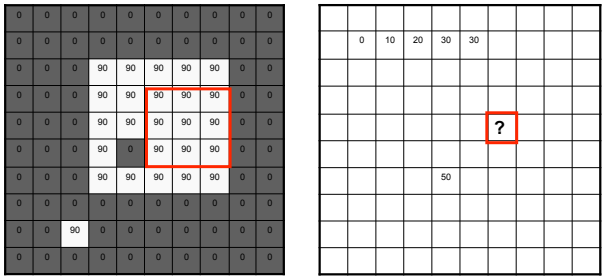
$h[7,5] = g[-1,-1]f[6,4] + g[-1,0]f[6,5] + g[-1,1]f[6,6] + g[0,-1]f[7,4] + g[0,0]f[7,5] + \ldots$

Credit: S. Seitz

Image Filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$ [1 1 1; 1 1 1; 1 1 1]

$f[.,.]$ $h[.,.]$



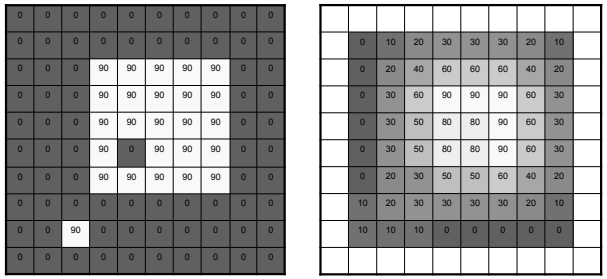
$$h[m,n]=\sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Credit: S. Seitz

Image Filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$ [1 1 1; 1 1 1; 1 1 1]

$f[.,.]$ $h[.,.]$



$$h[m,n]=\sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Credit: S. Seitz

Box Filter

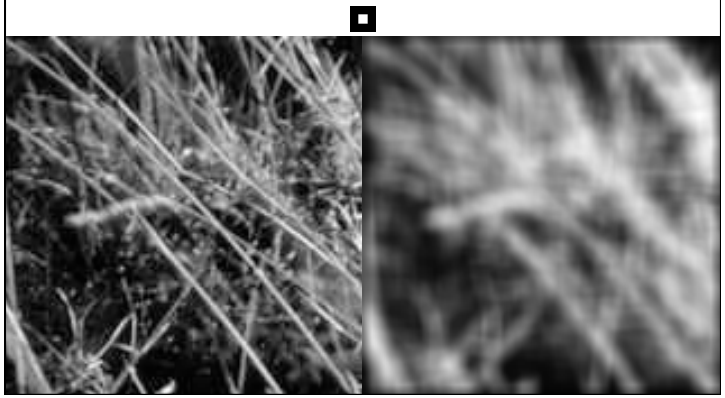
What does it do?

- Replaces each pixel with an *average* (arithmetic mean) of its neighborhood
- Achieves smoothing effect (i.e., removes sharp features)
- Weaknesses:
  - Blocky results
  - Axis-aligned streaks

$$g[\cdot,\cdot] = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Credit: D. Lowe

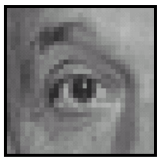
Smoothing with Box Filter



Properties of Smoothing Filters

- Values all positive
- Sum to 1 ⇒ constant regions same as input
- Amount of smoothing proportional to mask size
- Removes "high-frequency" components
- "low-pass" filter

Practice with Linear Filters



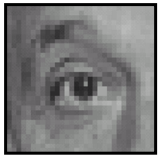
Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

?

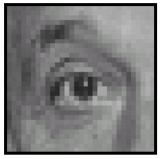
Credit: D. Lowe

Practice with Linear Filters



Original

0	0	0
0	1	0
0	0	0



Filtered
(no change)

Credit: D. Lowe

Practice with Linear Filters



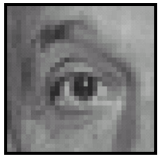
Original

0	0	0
0	0	1
0	0	0

?

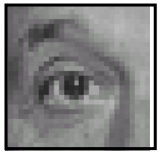
Credit: D. Lowe

Practice with Linear Filters



Original

0	0	0
0	0	1
0	0	0



Shifted ***left***
by 1 pixel

Credit: D. Lowe

Practice with Linear Filters



Original

0	0	0
0	2	0
0	0	0

$-\frac{1}{9}$

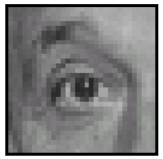
1	1	1
1	1	1
1	1	1

?

(Note that filter sums to 1)

Credit: D. Lowe

Practice with Linear Filters



Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

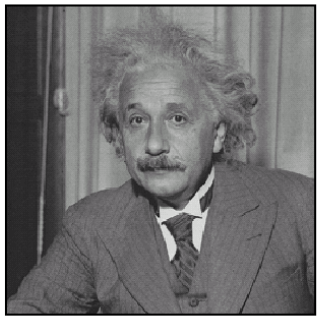


Image Sharpening filter

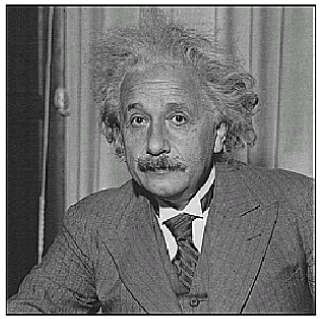
- Sharpen an out of focus image by subtracting a blurred version

Credit: D. Lowe

Image Sharpening



before



after

Credit: D. Lowe

Image Sharpening by Unsharp Masking

- $h = f - k(f * g)$ where k is a small positive constant and $g =$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

← called a Laplacian mask

- Why does it work?
- Say f is a blurred image produced from an ideal image p by convolving it with a box filter $s=$

$$\begin{bmatrix} 0 & 1/5 & 0 \\ 1/5 & 1/5 & 1/5 \\ 0 & 1/5 & 0 \end{bmatrix}$$

- $p * g \propto p - (p * s) = p - f$
- $h = f - k(f * g) \approx f - k(-1/5(p - f)) \approx p$
- Simulates ***Mach Band*** effect in human vision
- Called ***unsharp masking*** in photography

Sharpening using Unsharp Mask Filter



Original

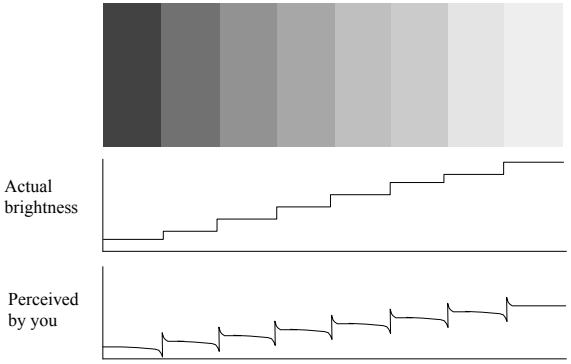


Filtered result

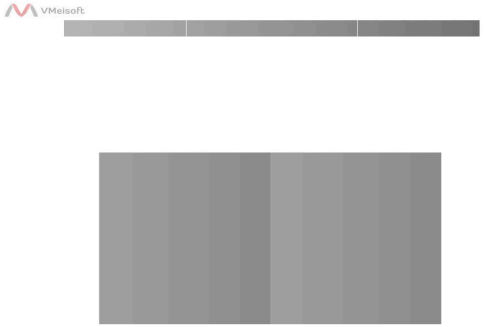
Unsharp Masking



Mach Bands



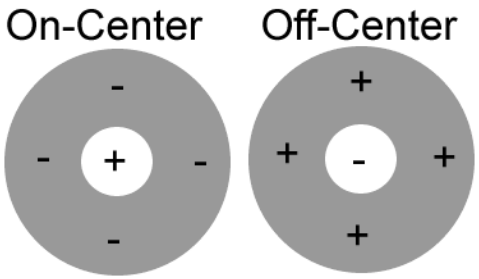
Mach Bands and the Cornsweet Illusion



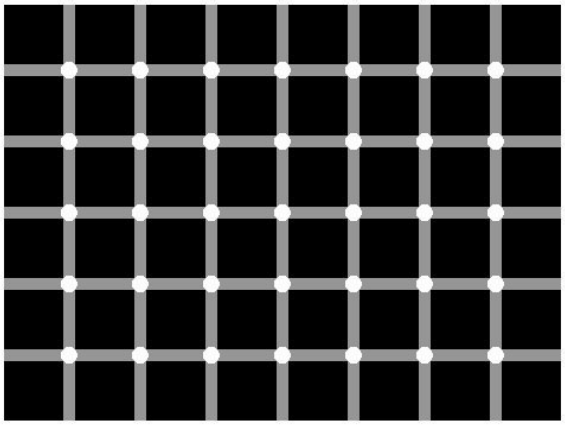
Credit: E. Adelson

Center/Surround Cells in Retina

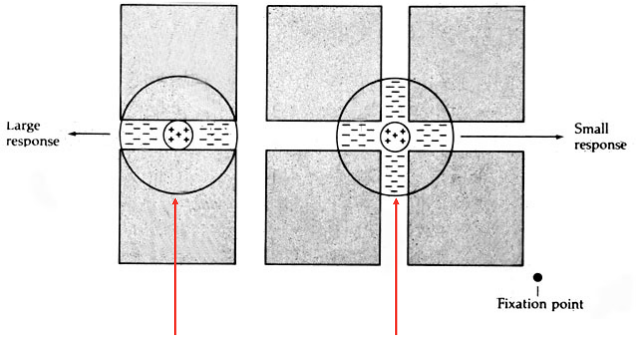
Organization of ganglion cells in retina respond to *discontinuities* of light (S. Kuffler, 1953)



E. Lingelbach's version of "Hermann Grid"



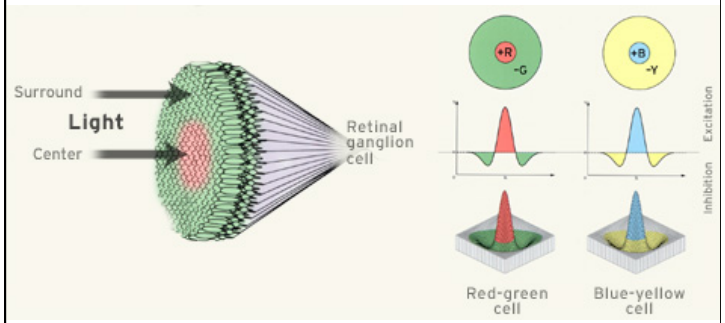
Lateral Inhibition Explanation



Inhibited less by white stripes

Inhibited more by white stripes

Lateral Inhibition in HVS

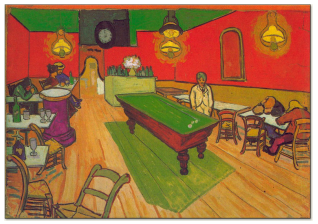


Cone photoreceptors

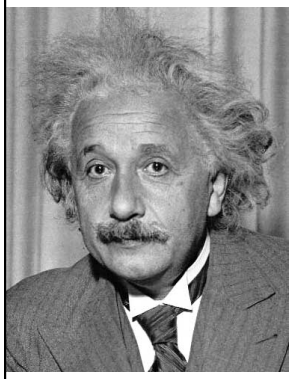
Simultaneous Contrast

In Van Gogh's cafes, the contrast between yellow & blue, and red & green, cause both colors to seem more vivid. Color perception always depends on the other colors nearby



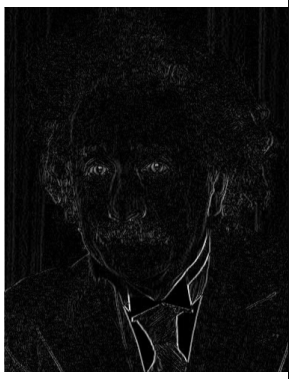


Other Filters: Edge Detection



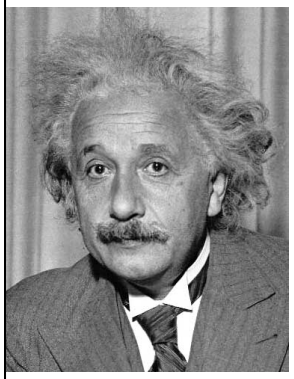
1	0	-1
2	0	-2
1	0	-1

Sobel filter



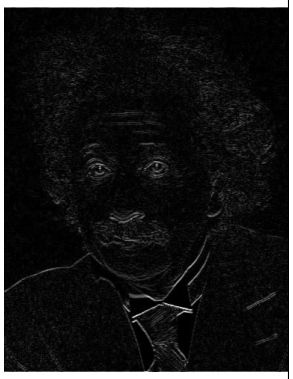
Vertical edges (absolute value)

Other Filters: Edge Detection



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

Sobel filter



Horizontal edges (absolute value)

Cross-Correlation vs. Convolution

g = filter f = image

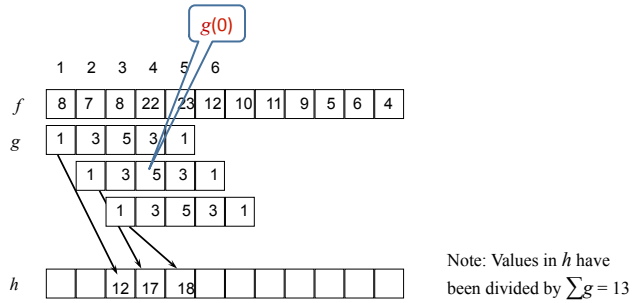
- 2D filtering / cross-correlation
  - `h=filter2(g,f);` or `h=imfilter(f,g);`

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

- 2D convolution
  - `h=conv2(g,f);`

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m-k, n-l]$$

Example of 1D Convolution



$$h(4) = \sum_{i=-2}^{2} g(i) f(4-i)$$
$$= g(-2)f(6) + g(-1)f(5) + g(0)f(4) + g(1)f(3) + g(2)f(2)$$
$$= (1)(12) + (3)(23) + (5)(22) + (3)(8) + (1)(7)$$
$$= 222$$

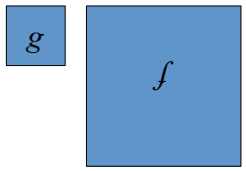
Convolution

Convolution is equivalent to: Flipping the image, f, in both dimensions (bottom to top, right to left), and then performing cross-correlation with g; (or, flipping filter, g, and then applying cross-correlation with f)

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m-k, n-l]$$

$h = g * f$

Notation for convolution operator



Credit: K. Grauman

Key Properties of Linear Filters

Linearity:

`filter(f1 + f2) = filter(f1) + filter(f2)`

Shift invariance: same behavior regardless of pixel location

`filter(shift(f)) = shift(filter(f))`

Any linear, shift-invariant (LSI) operator can be represented as a ***convolution*** operation

Credit: S. Lazebnik

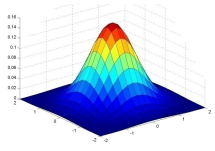
More Properties

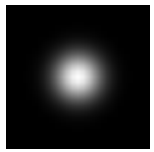
- Commutative: $a * b = b * a$
  - Conceptually no difference between filter and image
- Associative: $a * (b * c) = (a * b) * c$
  - Often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$
  - This is equivalent to applying one filter: $a * (b_1 * b_2 * b_3)$
- Distributes over addition: $a * (b + c) = (a * b) + (a * c)$
- Scalars factor out: $ka * b = a * kb = k\,(a * b)$
- Identity: unit impulse $e = [0, 0, 1, 0, 0] \Rightarrow a * e = a$

Credit: S. Lazebnik

Gaussian Filtering

- Weight contributions of neighboring pixels by *distance from center pixel*





0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

5 x 5, σ = 1

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

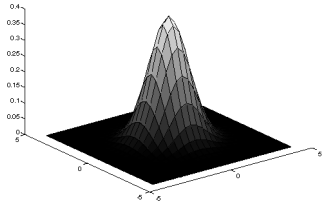
Constant factor in front makes it sum to 1

(x, y) are coordinates of pixel, where center pixel is at (0, 0)

Credit: C. Rasmussen

Smoothing with a Gaussian

- Smoothing with a box filter doesn't model well a defocused lens
- Most obvious difference is that a single point of light viewed in a defocused lens looks like a fuzzy blob; but the averaging process would give a little square
- Gaussian is ***isotropic*** (i.e., rotationally symmetric)



- A Gaussian gives a good model of a fuzzy blob
- It closely models many physical processes (the sum of many small effects)

Credit: D. Forsyth

What does Blurring take away?



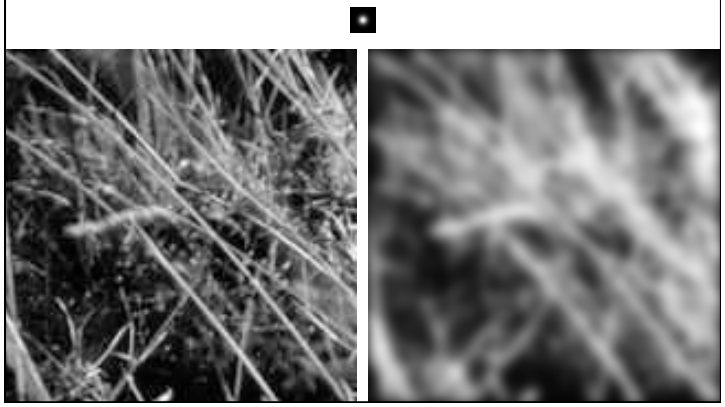
original

What does Blurring take away?



smoothed (5 x 5 Gaussian)

Smoothing with Gaussian Filter



Smoothing with Box Filter



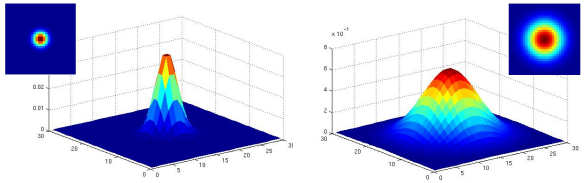






Gaussian Filters

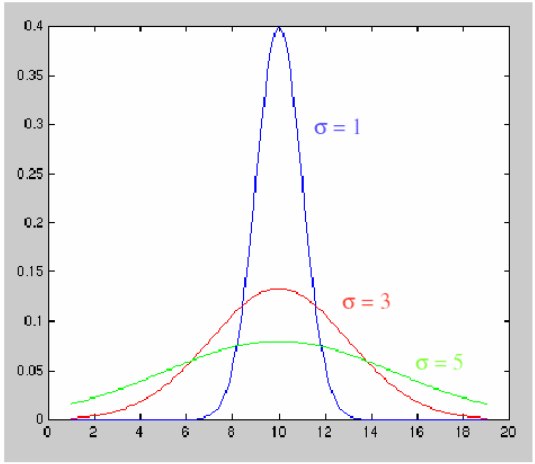
- What parameters matter here?
- **Standard deviation**, σ, of Gaussian: determines amount of smoothing



σ = 2 with 30 x 30 kernel

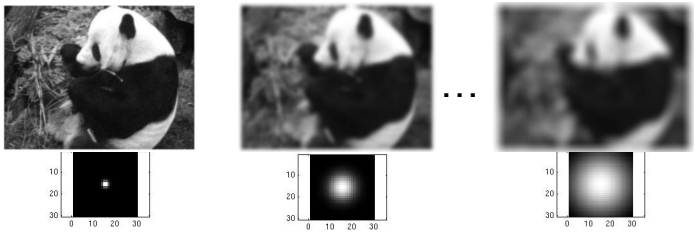
σ = 5 with 30 x 30 kernel

Credit: D. Hoiem



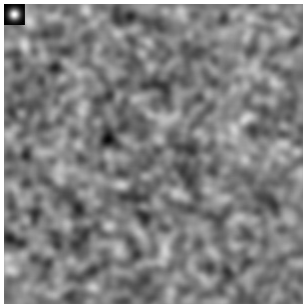
Smoothing with a Gaussian

Parameter σ is the "scale" / "width" / "spread" of the Gaussian kernel, and controls the amount of smoothing



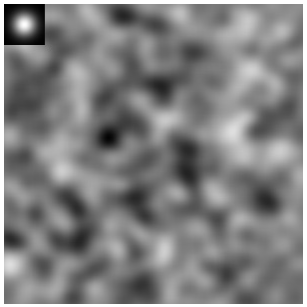
```
for sigma=1:3:10
   h = fspecial('gaussian', hsize, sigma);
   out = imfilter(im, h);
   imshow(out);
   pause;
end
```

Small σ



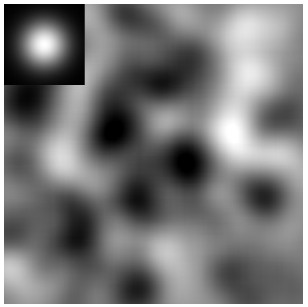
Credit: D. Forsyth

Medium σ



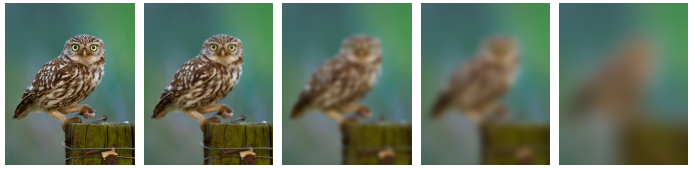
Credit: D. Forsyth

Large σ



Credit: D. Forsyth

Gaussian Filters



σ = 1 pixel σ = 5 pixels σ = 10 pixels σ = 30 pixels

Spatial Resolution and Color



original

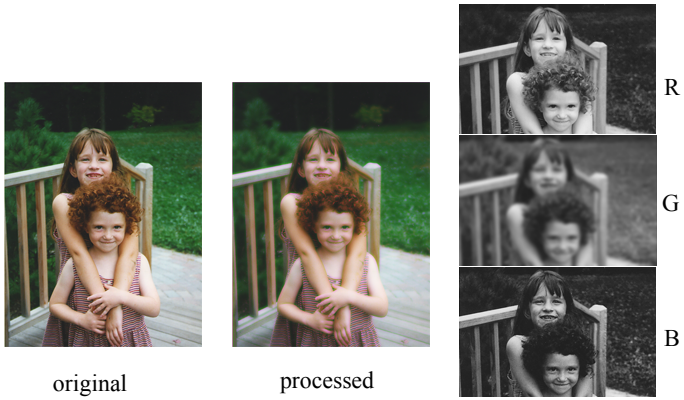


R

G

B

Blurring the G Component



original processed

Blurring the R Component



original processed

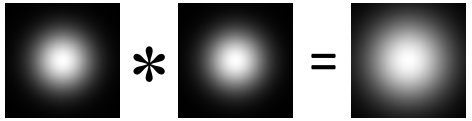
Blurring the B Component



original processed

Cascading Gaussian Filters

- Removes "high-frequency" information from the image ("low-pass" filter)
- Convolution of two Gaussians is another Gaussian



– Convolving *two times* with Gaussian of size σ is same as convolving *once* with Gaussian of size $\sigma\sqrt{2}$

Credit: K. Grauman

Separability

- A function $g(x, y)$ is **separable** if $g(x, y) = g_1(x)\, g_2(y)$
- The Gaussian function is separable:

$$e^{-[(x^2 + y^2)/2\sigma^2]} = e^{-[x^2/2\sigma^2]} * e^{-[y^2/2\sigma^2]}$$

- First convolve the image with a *1D horizontal filter*
- Then convolve the result of the first convolution with a *1D vertical filter*
- For a k x k Gaussian filter, 2D convolution requires k^2 multiplications and k^2-1 additions per pixel
- But using the separable filters, we reduce this to $2k$ multiplications and $2k$-1 additions per pixel
- Matlab: `h = conv2(g1,g2,f);`

Separable filtering

$$a_2[i,j] = a_1[i]a_1[j]$$

1	4	6	4	1
4	16	24	16	4
6	24	36	24	6
4	16	24	16	4
1	4	6	4	1

1	4	6	4	1

1
4
6
4
1

Separable filtering

$$a_2[i,j] = a_1[i]a_1[j]$$

1	4	6	4	1
4	16	24	16	4
6	24	36	24	6
4	16	24	16	4
1	4	6	4	1

0	0	0	0	0
0	0	0	0	0
1	4	6	4	1
0	0	0	0	0
0	0	0	0	0

0	0	1	0	0
0	0	4	0	0
0	0	6	0	0
0	0	4	0	0
0	0	1	0	0

Separable filtering

$$a_2[i,j] = a_1[i]a_1[j]$$

1	4	6	4	1
4	16	24	16	4
6	24	36	24	6
4	16	24	16	4
1	4	6	4	1

0	0	0	0	0
0	0	0	0	0
1	4	6	4	1
0	0	0	0	0
0	0	0	0	0

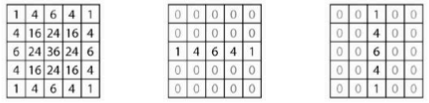
0	0	1	0	0
0	0	4	0	0
0	0	6	0	0
0	0	4	0	0
0	0	1	0	0

first, convolve with this

$$\sum_{i'} a_1[i'] \left(\sum_{j'} a_1[j'] b[i - i', j - j'] \right)$$

Separable filtering

$$a_2[i,j] = a_1[i]a_1[j]$$



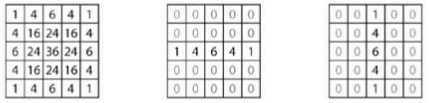
second, convolve with this

first, convolve with this

$$\sum_{i'} a_1[i'] \left(\sum_{j'} a_1[j']b[i-i', j-j'] \right)$$

Separable filtering

$$a_2[i,j] = a_1[i]a_1[j]$$



second, convolve with this

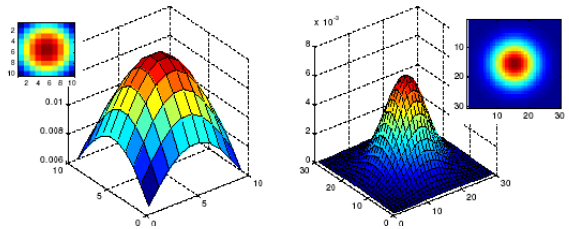
first, convolve with this

$$\sum_{i'} a_1[i'] \left(\sum_{j'} a_1[j']b[i-i', j-j'] \right)$$

Which is faster? $O(2(n+1))$ or $O((n+1)^2)$

Gaussian Filters

- What other parameter(s) must be set?
- **Size** of kernel or mask

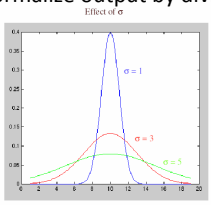


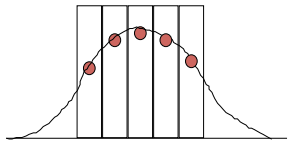
σ = 5 with 10 x 10 kernel

σ = 5 with 30 x 30 kernel

How Big should the Filter be?

- Gaussian function has infinite "support" but need a finite-size kernel
- Values at edges should be near 0
- ~98.8% of area under Gaussian in mask of size 5σ x 5σ
- In practice, use mask of size about 2*k*+1 x 2*k*+1 where $k \approx 3\sigma$
- Multiply real values of Gaussian by a scale factor (= min real value) to obtain integer weights
- Normalize output by dividing by sum of all weights





Gaussian Filter

3 x 3 approximation of a Gaussian:

1/16

1	2	1
2	4	2
1	2	1

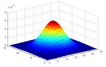
1D separable kernel for σ = 1: 1/38 [1, 9, 18, 9, 1]

Matlab Implementation

```
>> hsize = 10;
>> sigma = 5;
>> h = fspecial('gaussian' hsize, sigma);
```

hsize = width of mask

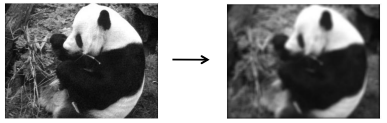
```
>> mesh(h);
```



```
>> imagesc(h);
```



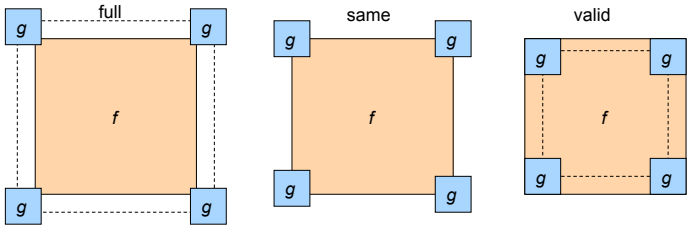
```
>> outim = imfilter(im, h); % correlation
>> imshow(outim);
```



original outim

What is the Size of the Output Image?

- MATLAB: filter2(g, f, *shape*)
  - *shape* = 'full': output size is sum of sizes of *f* and *g*
  - *shape* = 'same': output size is same as *f*
  - *shape* = 'valid': output size is difference of sizes of *f* and *g*



Credit: S. Lazebnik

What about Near the Image Border?

- the filter window falls off the border of the image
- need to extrapolate
- methods:
  - clip filter (black)
  - wrap around
  - copy border
  - reflect across border



Credit: S. Marschner

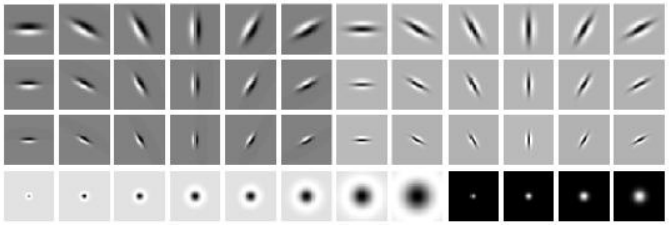
What about Near the Image Border?

- methods (MATLAB):
  - clip filter (black): imfilter(f, g, 0)
  - wrap around: imfilter(f, g, 'circular')
  - copy border: imfilter(f, g, 'replicate')
  - reflect across border: imfilter(f, g, 'symmetric')

Credit: S. Marschner

Application: Filter Banks for Feature Detection

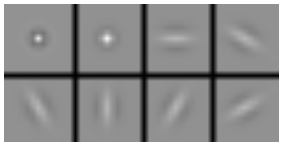
Filter Bank (48 filters)



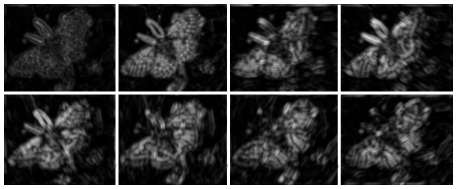
Code for filter banks: *www.robots.ox.ac.uk/~vgg/research/texclass/filters.html*

Filter Banks

Process image with each filter and keep responses (or squared or abs value of responses)







Application: Representing Texture

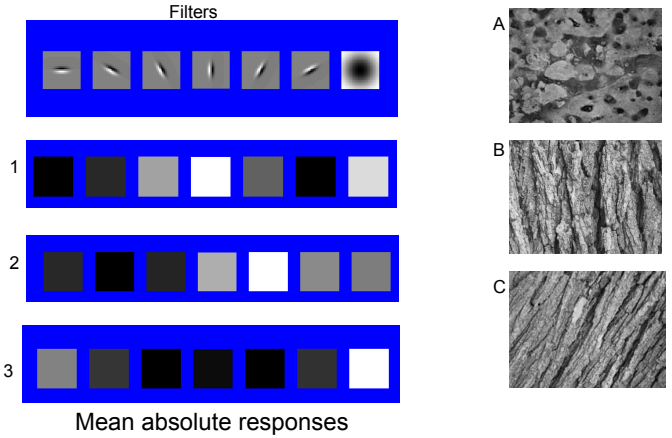


Source: Forsyth

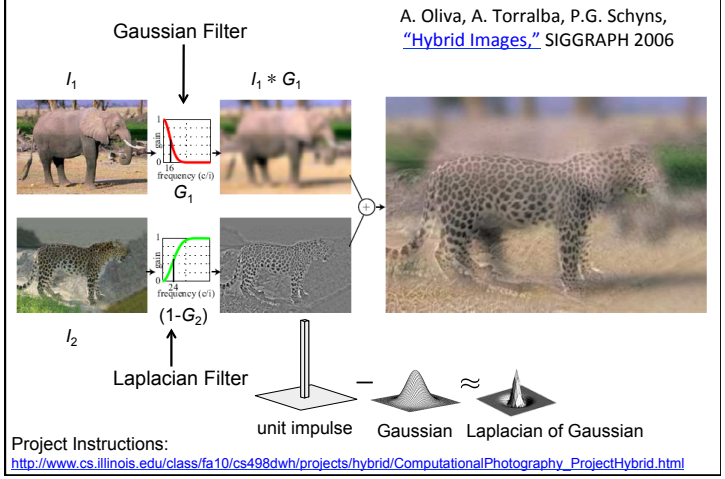
How can we Represent Texture?

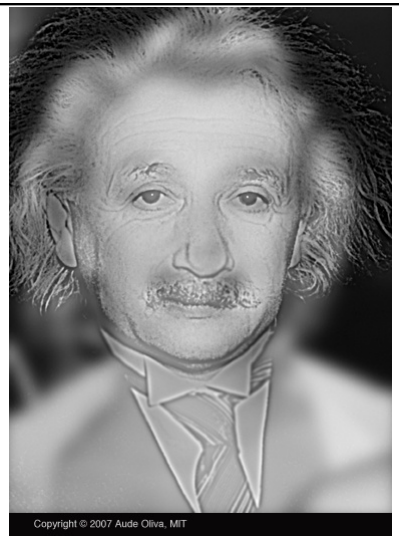
- Textures are characterized by the material, orientation, scale and other regular or statistical properties
- Describe a region's texture as a feature vector defined by measuring the responses of blobs and edges at various orientations and scales
- Record simple statistics (e.g., mean, std) of absolute filter responses

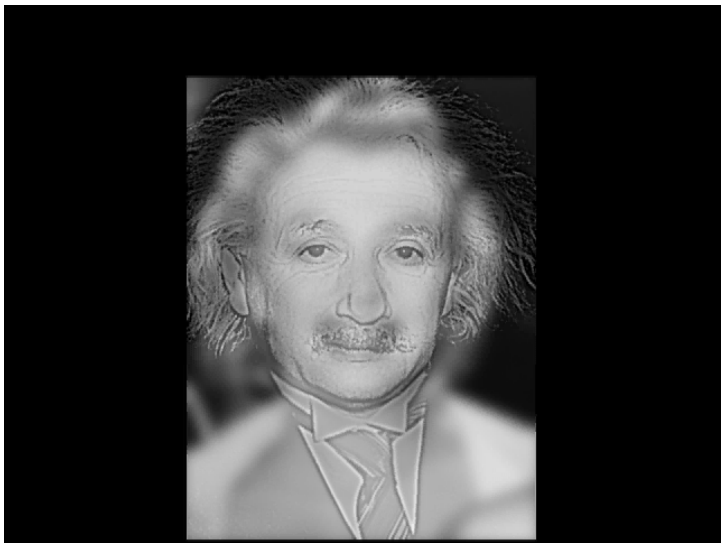
Can you Match the Texture to the Responses?

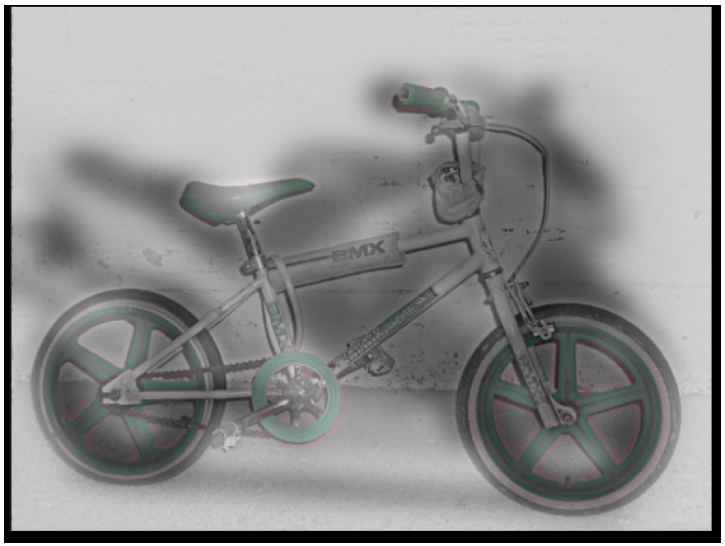


Application: Hybrid Images



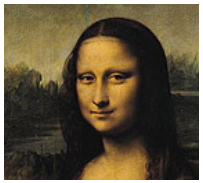






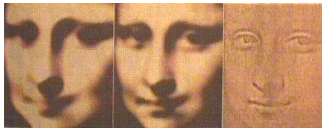
What Makes the Mona Lisa Smile?

The smile only becomes apparent if a viewer looks at her eyes or elsewhere on her face; the smile disappears when looking directly at her mouth



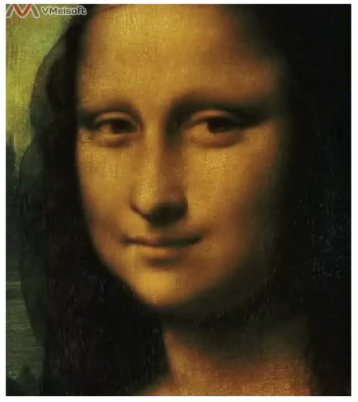
Peripheral vision is low resolution and blurs, picking up shadows from the Mona Lisa's cheekbones, which suggests the curvature of a smile

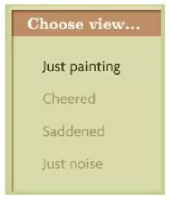
"*The elusive quality of the Mona Lisa's smile can be explained by the fact that her smile is almost entirely in low spatial frequencies, and so is seen best by your peripheral vision*"
-- Margaret Livingstone



coarse medium fine

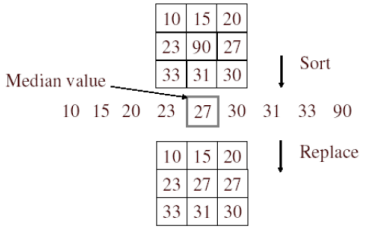
What Makes the Mona Lisa Smile?





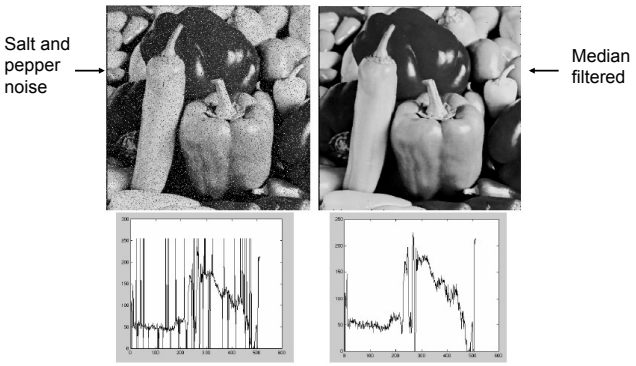
Non-Linear Filtering

Median Filter



- Replace pixel by the median value of its neighbors
- No new pixel values introduced
- Removes spikes: good for impulse, salt & pepper noise
- **Nonlinear operation**

Median Filter



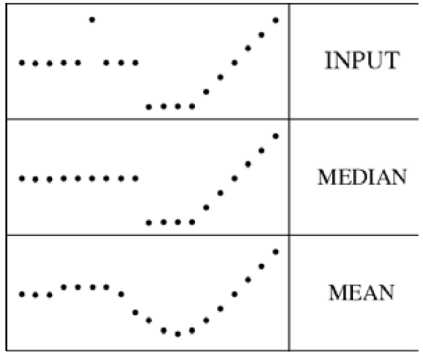
Plots of 1 row in the images

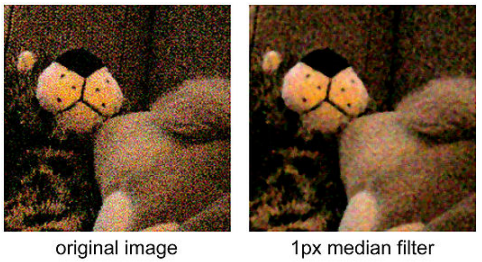
Matlab: `output im = medfilt2(im, [h w]);`

Credit: M. Hebert

Median Filter

- Median filter is **edge preserving**





original image | 1px median filter

3px median filter | 10px median filter

Median Filter (19 x 19 window)



input | output

Credit: J. Plush

Bilateral Filter for Image Smoothing and "Tooning"

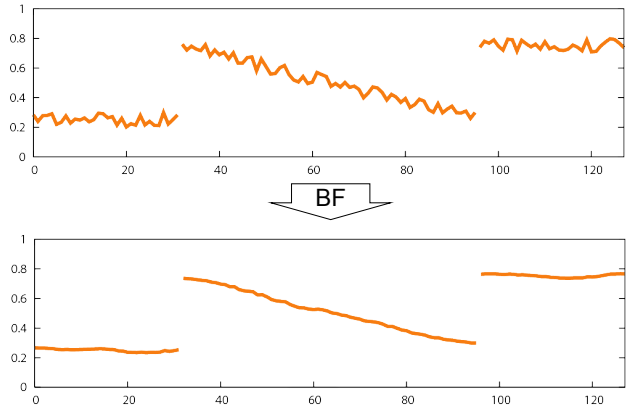


Basis for many apps such as ToonPaint

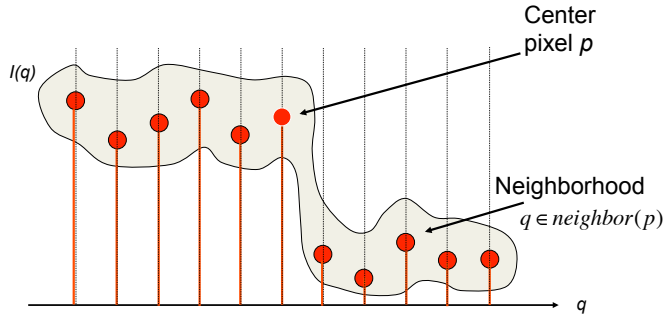
Bilateral Filter

- Idea: Edge-preserving smoothing filter, i.e., smooth image but preserve edges, using a **weighted average of pixels**
- Weight pixels by spatial distance *and* intensity difference

Bilateral Filter on 1D Image



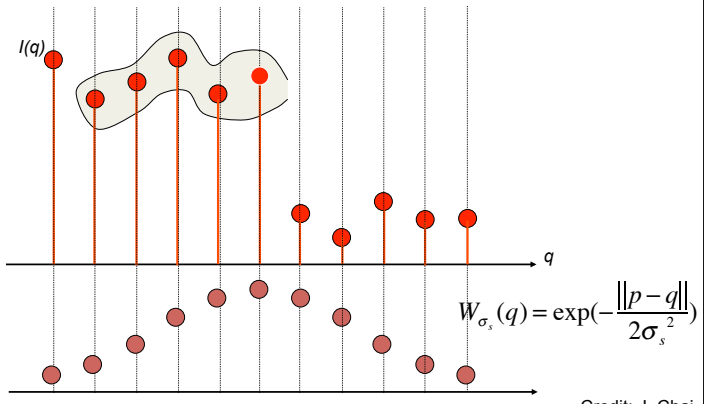
1D Example



In weighting the neighbors of *p*, we would like to preserve the "step" in brightness

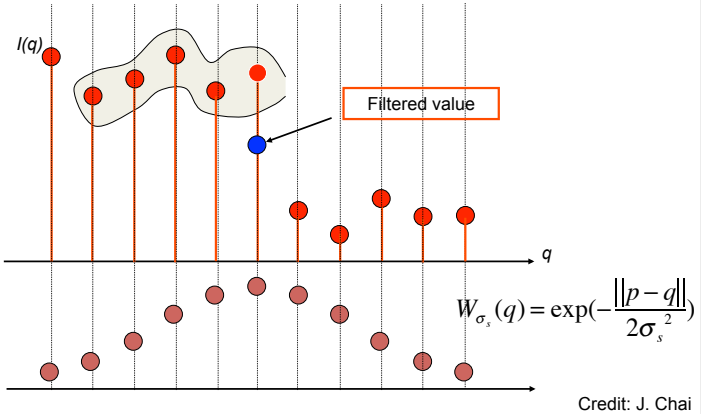
Credit: J. Chai

Smoothing using Gaussian Weights



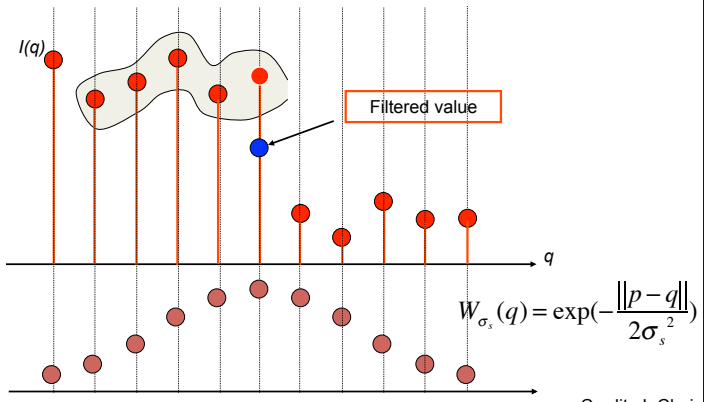
Credit: J. Chai

Gaussian Filtered Result



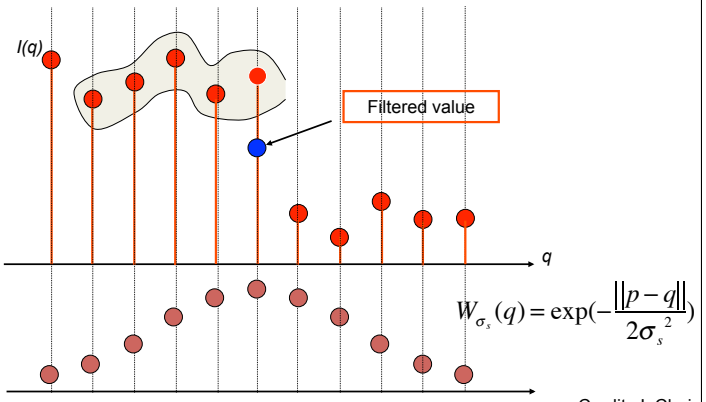
Credit: J. Chai

Edges Are Smoothed



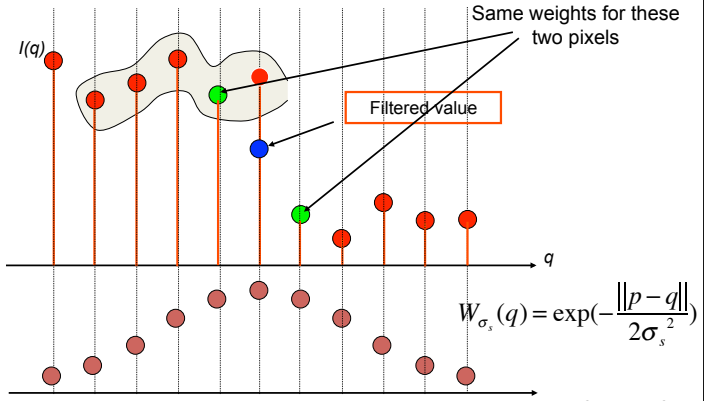
Credit: J. Chai

What Causes the Problem?



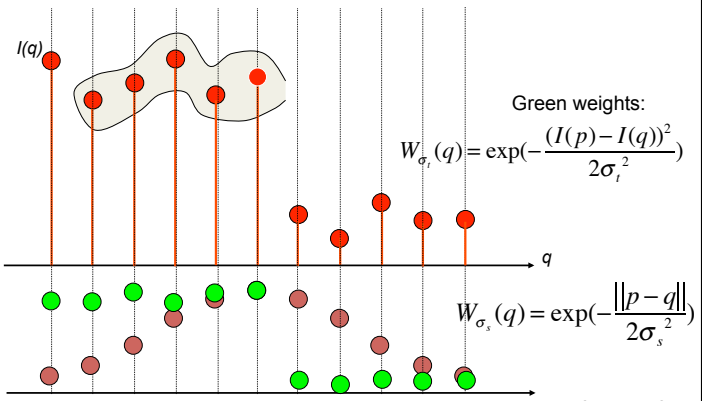
Credit: J. Chai

What Causes the Problem?



Credit: J. Chai

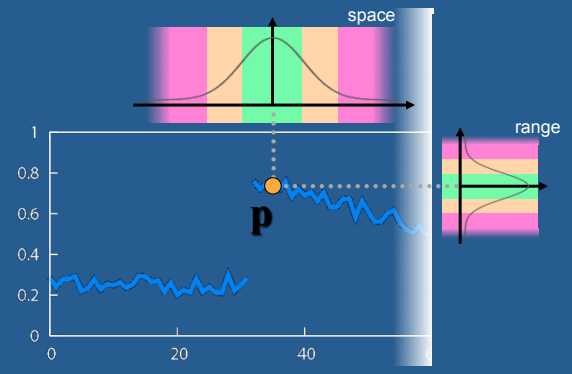
The Kernel Weights



Credit: J. Chai

Influence of Pixels

Only pixels close in space ***and*** in brightness are used



Bilateral Filter

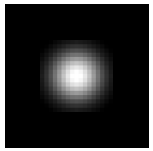
$$h(\mathbf{p}) = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in N(\mathbf{p})} G_{\sigma_s}\left(\|\mathbf{p}-\mathbf{q}\|\right) G_{\sigma_r}\left(|I(\mathbf{p})-I(\mathbf{q})|\right) I(\mathbf{q})$$

normalization factor

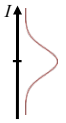
space weight

brightness weight

$\mathbf{p} = (x_1, y_1)$
$\mathbf{q} = (x_2, y_2)$

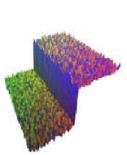


weight nearest neighbors more



weight neighbors with similar brightness more

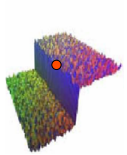
Bilateral Filter



input

Credit: J. Chai

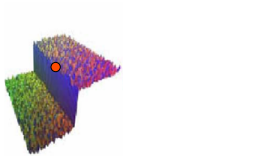
Bilateral Filter



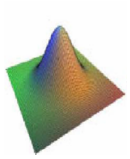
input

Credit: J. Chai

Bilateral Filter



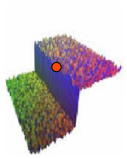
input



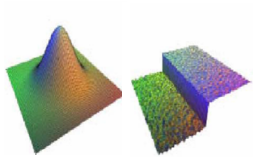
W_s

Credit: J. Chai

Bilateral Filter



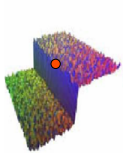
input



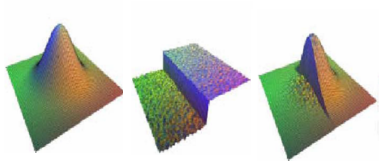
W_s W_t

Credit: J. Chai

Bilateral Filter



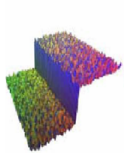
input



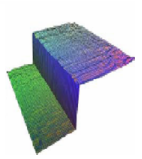
W_s W_t W_t*W_s

Credit: J. Chai

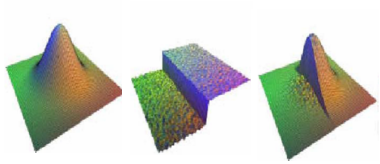
Bilateral Filter



input



Output



W_s W_t W_t*W_s

Credit: J. Chai

Bilateral Filter for Image Smoothing



Original | Bilateral filtered | Bilateral filter applied to log-intensity image

Bilateral Filter for Image Smoothing



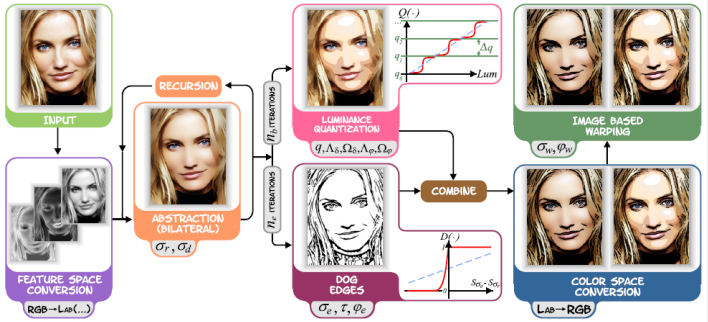
Bilateral filter implemented in Photoshop as “Surface Blur” tool



Top row: original; sharpened with Gaussian; sharpened with median (note fewer halo artifacts.) Middle row: Filtered at 20th; 50th [median]; and 80th percentiles. Bottom row: “High Pass” using median; bilateral smoothing filter; logarithmic bilateral filter.

Image Abstraction and “Tooning”

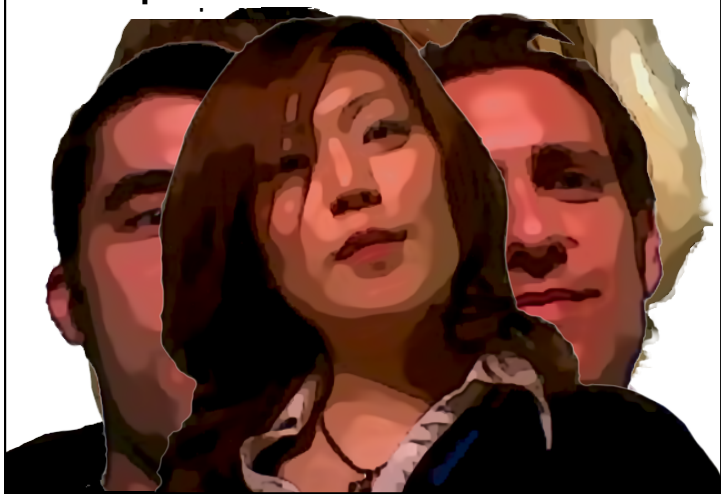
[Winnemöller, Olsen, Gooch, 2006]







Examples



Real-Time Video Abstraction

