

Lecture 16:

Image Compression and Basic Image Processing

Computer Graphics: Rendering, Geometry, and Image Manipulation
Stanford CS248A, Winter 2023

Color spaces demo

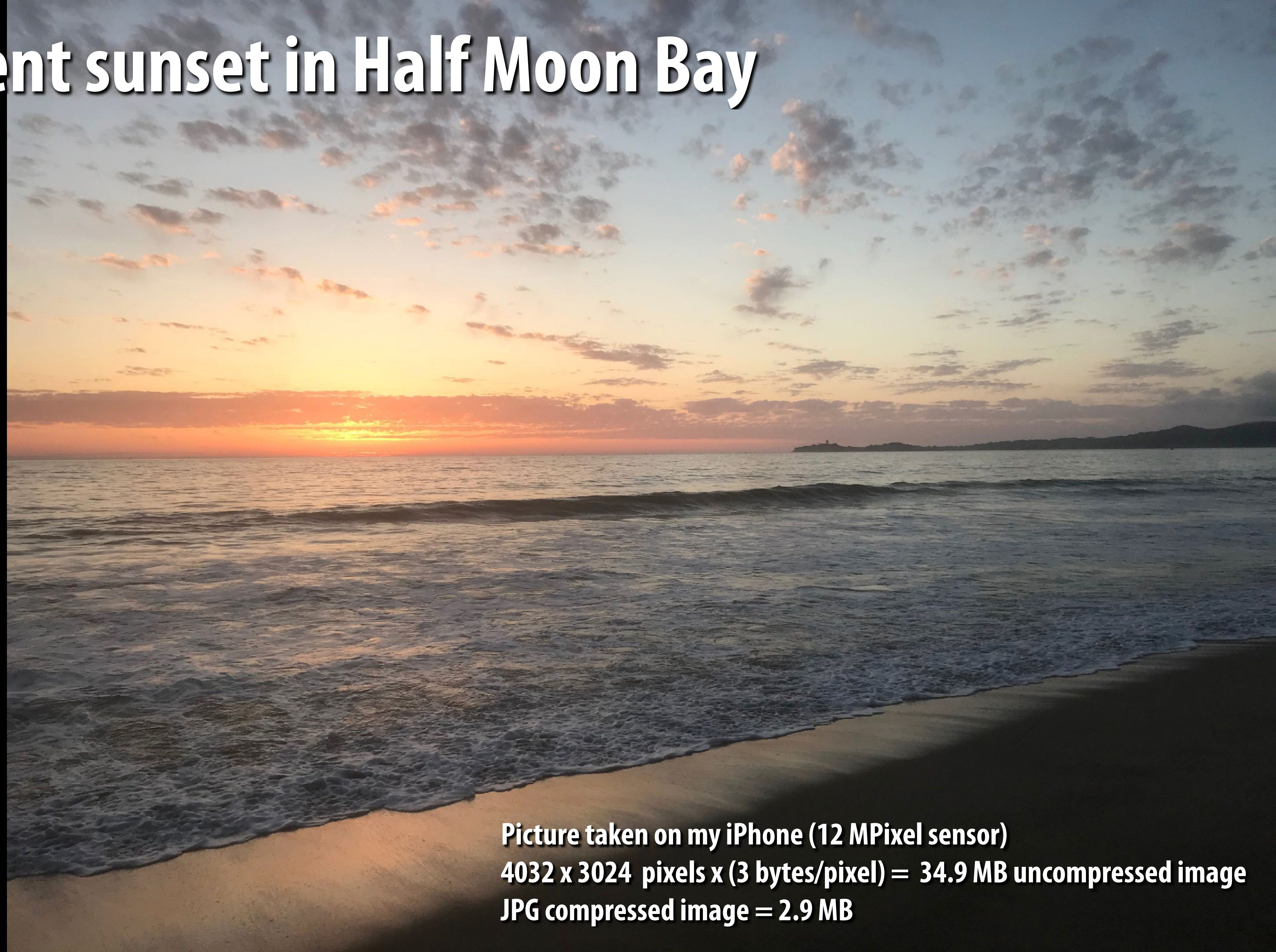
(Finishing up from last time)

Recurring themes in the course

- **Choosing the right representation for a task**
 - e.g., choosing the right basis
- **Exploiting human perception for computational efficiency**
 - Errors/approximations in algorithms can be tolerable if humans do not notice
- **Convolution as a useful operator**
 - To remove high frequency content from images
 - What else can we do with convolution?

Image Compression

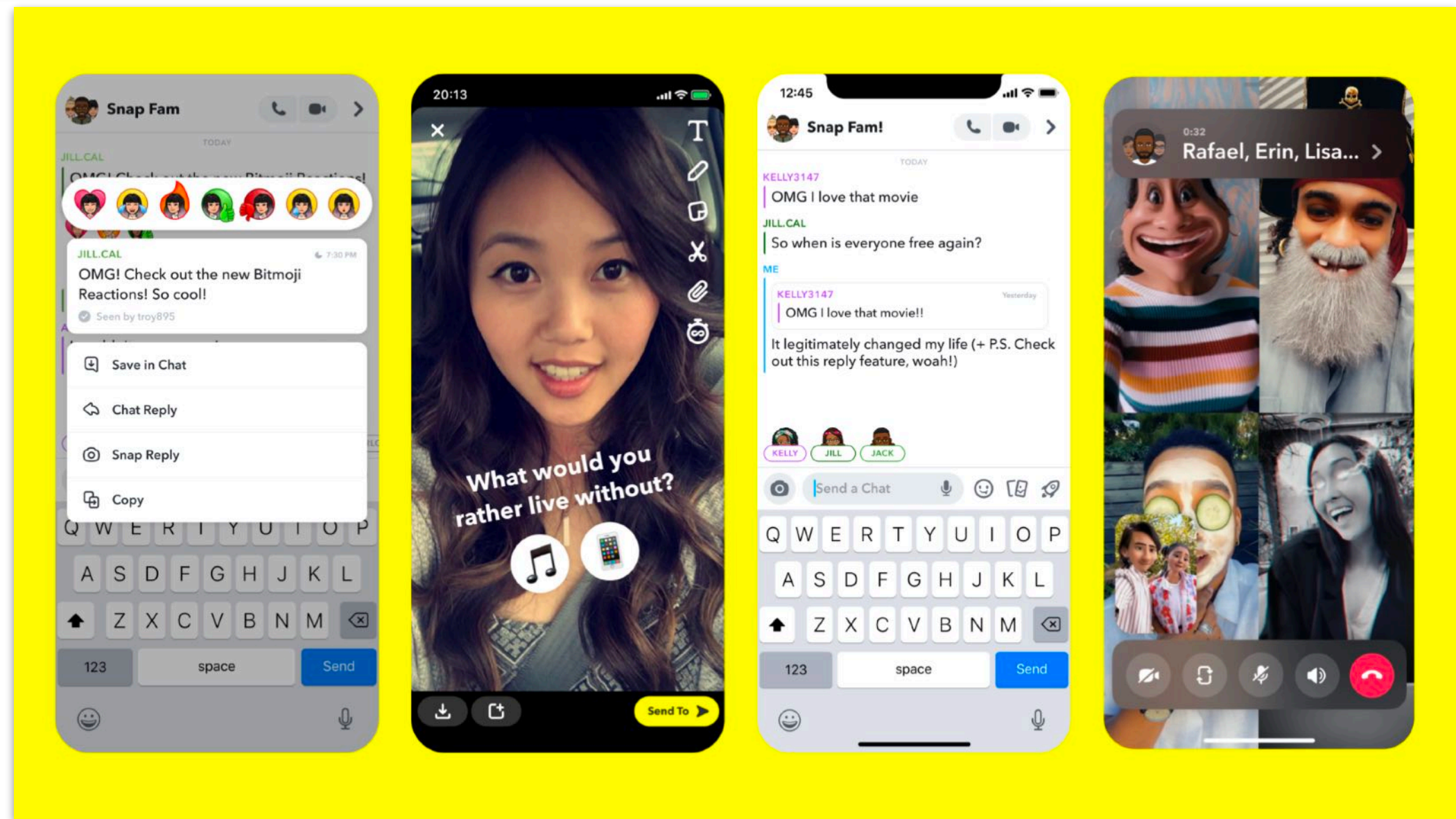
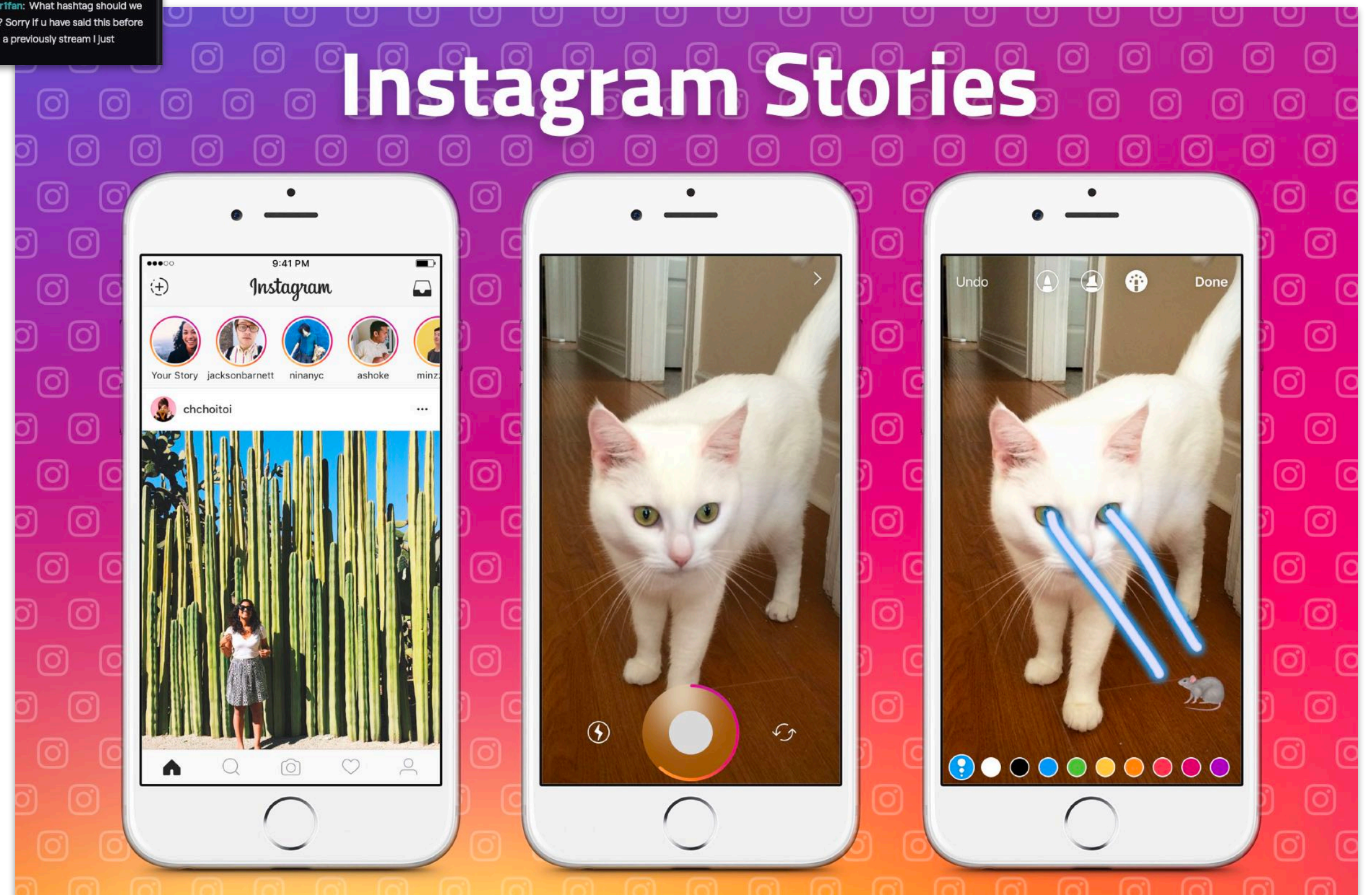
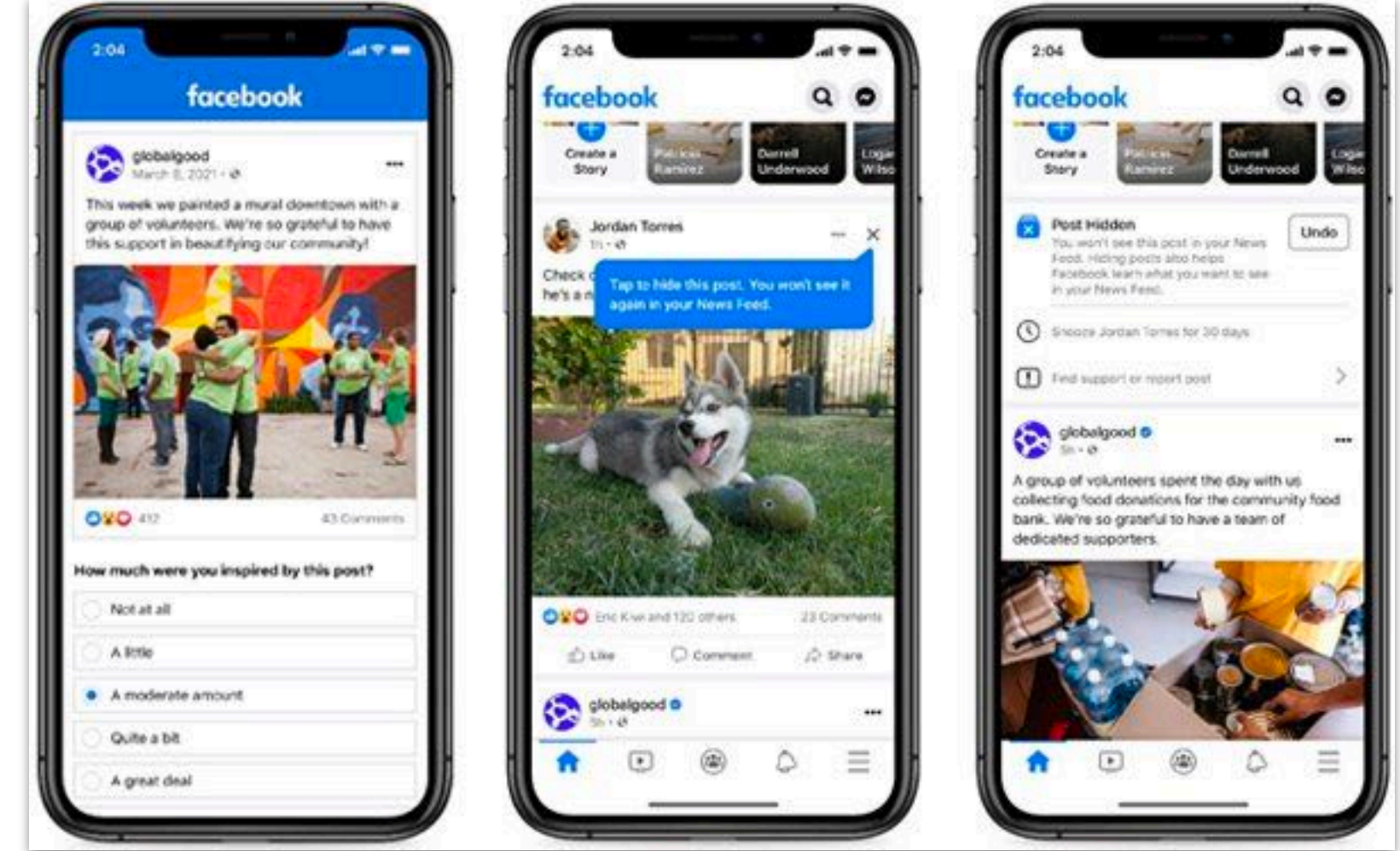
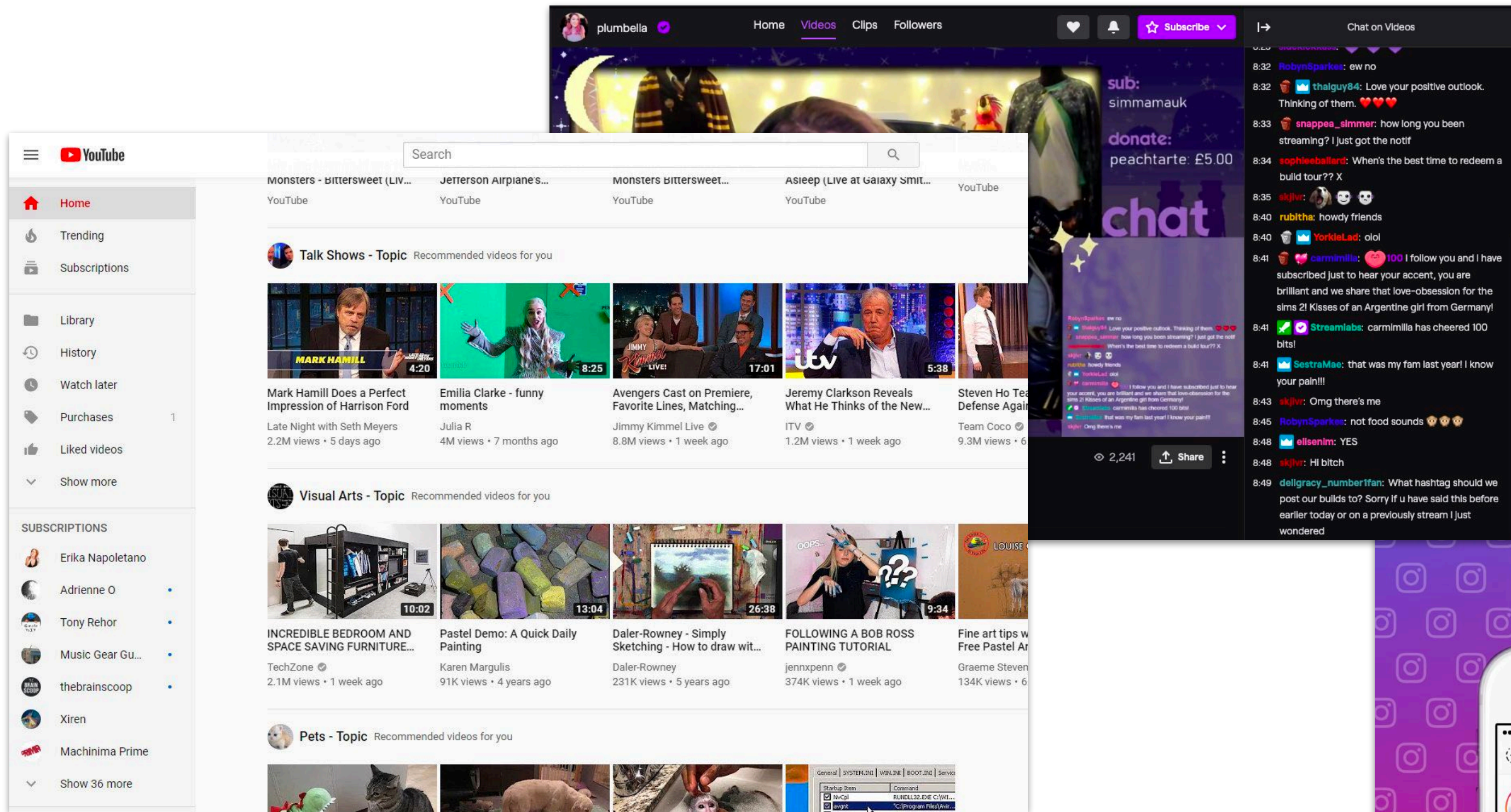
A recent sunset in Half Moon Bay



Picture taken on my iPhone (12 MPixel sensor)

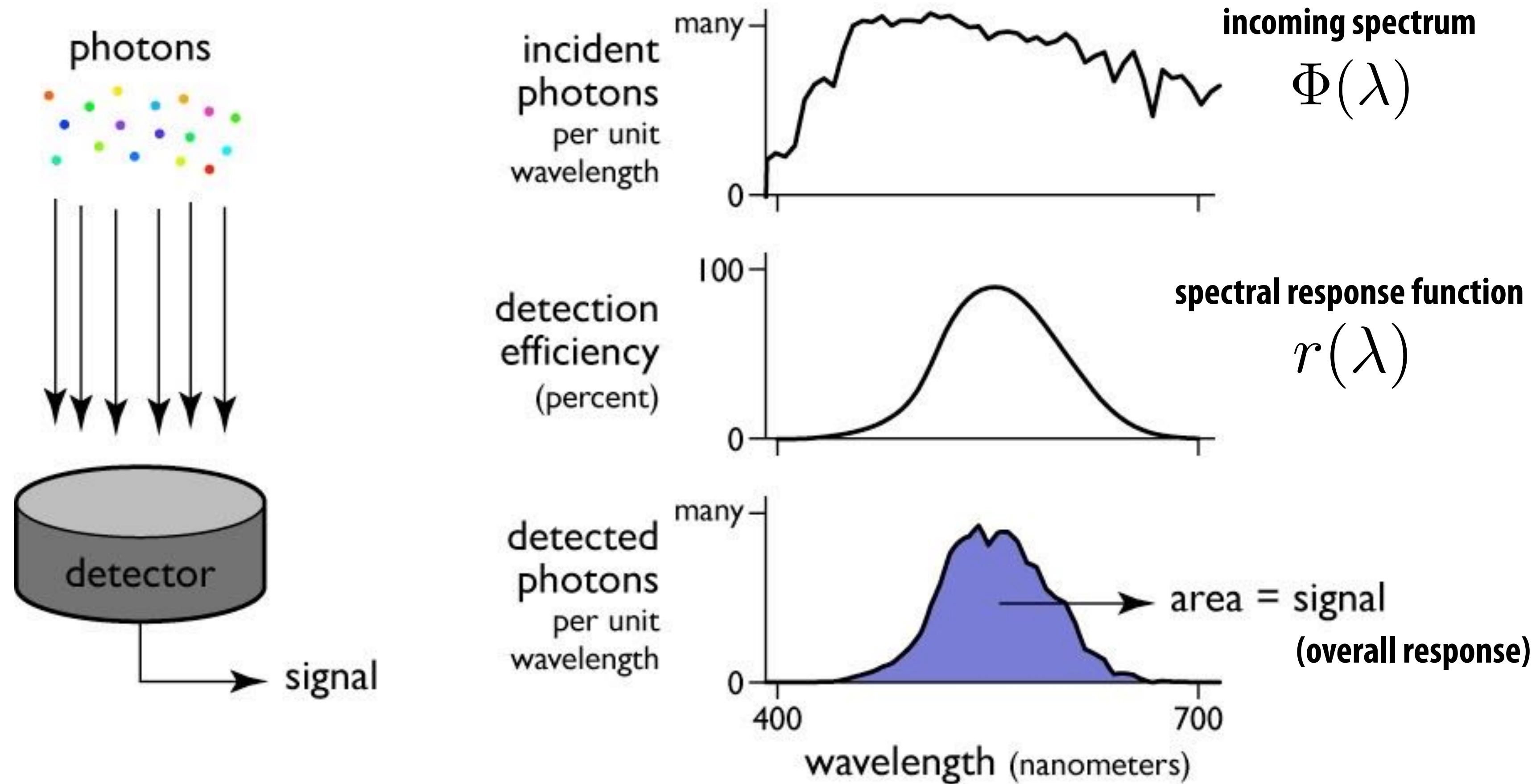
$4032 \times 3024 \text{ pixels} \times (3 \text{ bytes/pixel}) = 34.9 \text{ MB uncompressed image}$

JPG compressed image = 2.9 MB



Review from last time

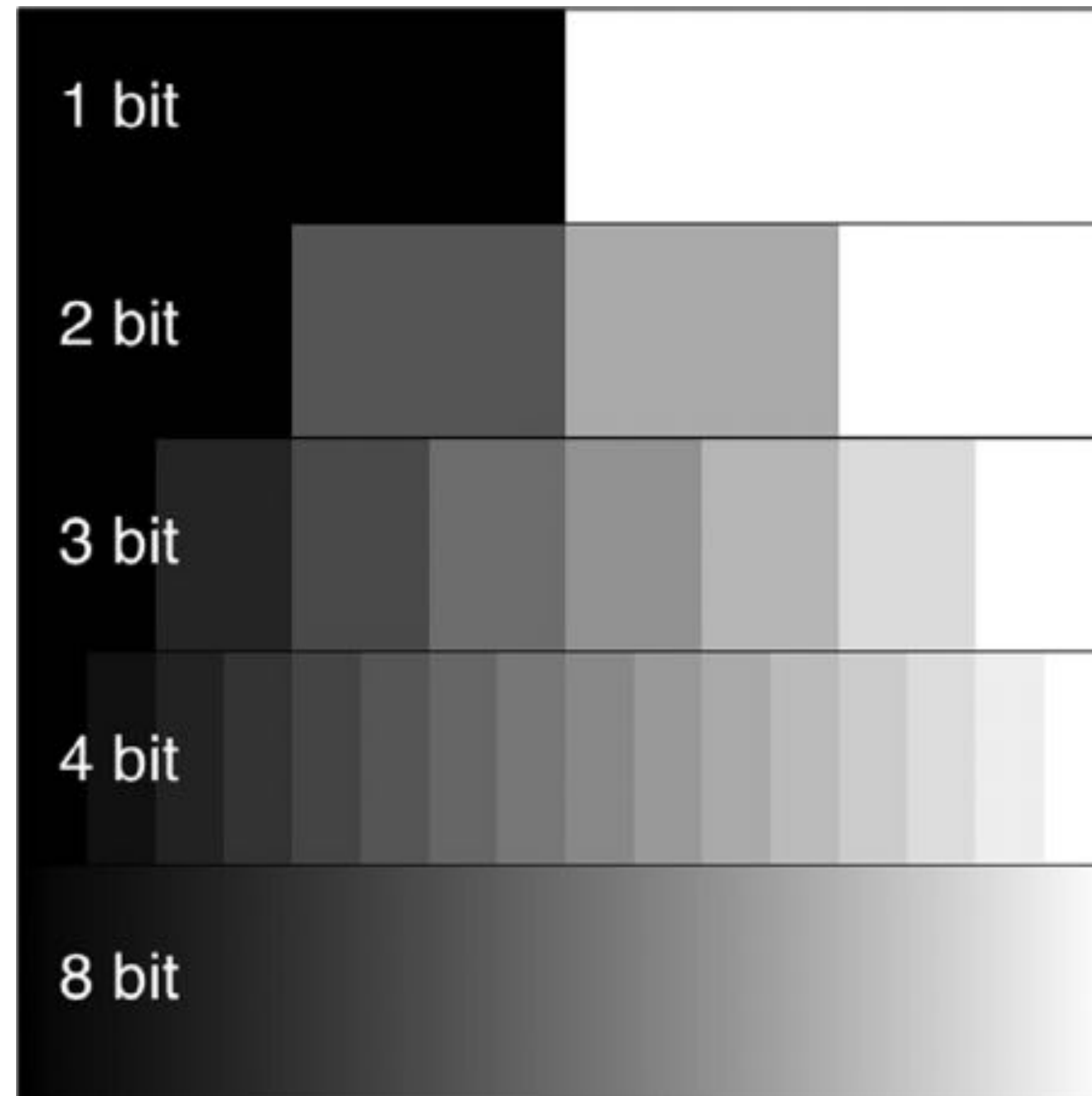
Sensor's response is proportional to amount of light arriving at sensor



$$R = \int_{\lambda} \Phi(\lambda) r(\lambda) d\lambda$$

Encoding numbers

- More bits → can represent more unique numbers
- 8 bits → 256 unique numbers (0-255)

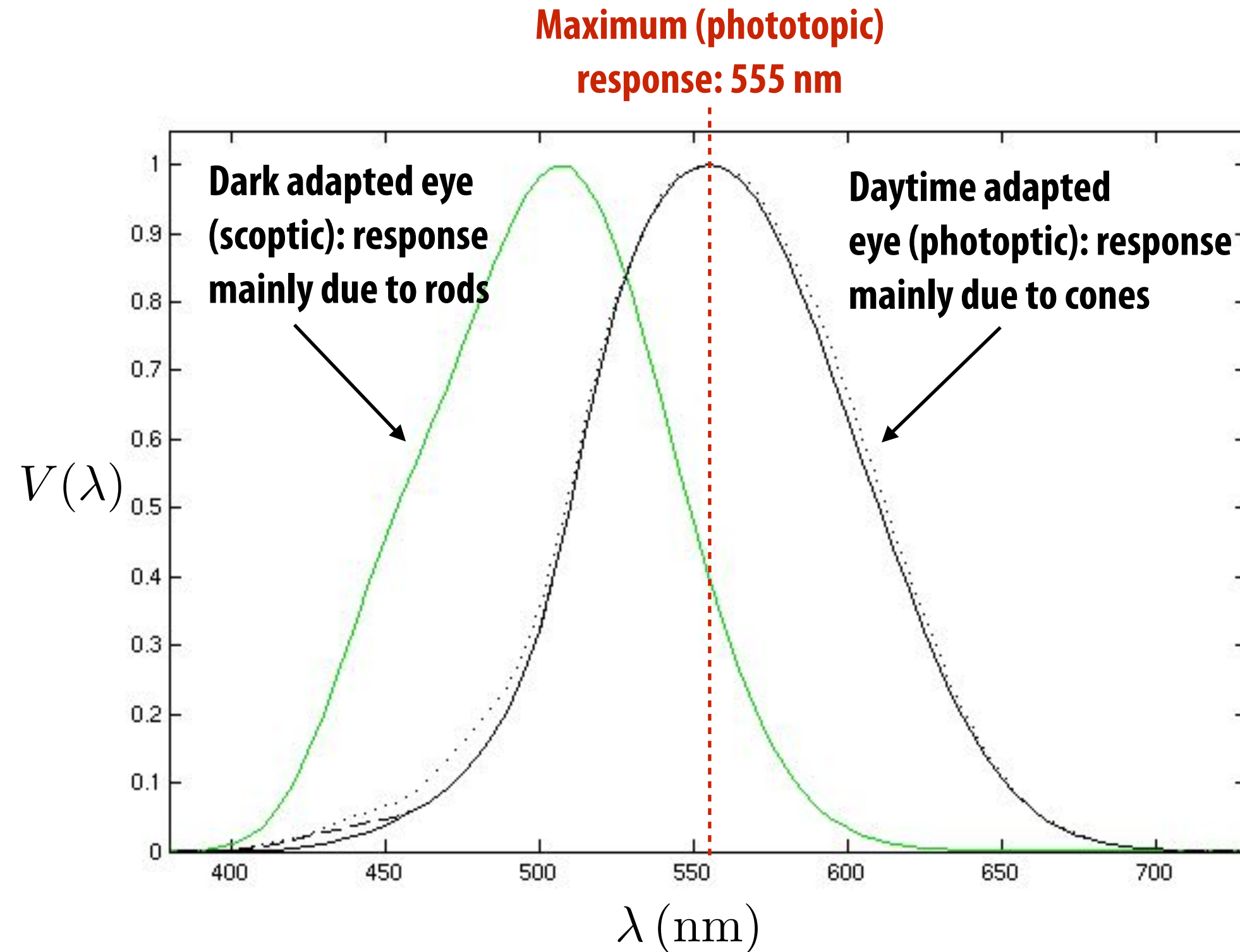


Luminance (brightness)

Product of radiance and the eye's luminous efficiency

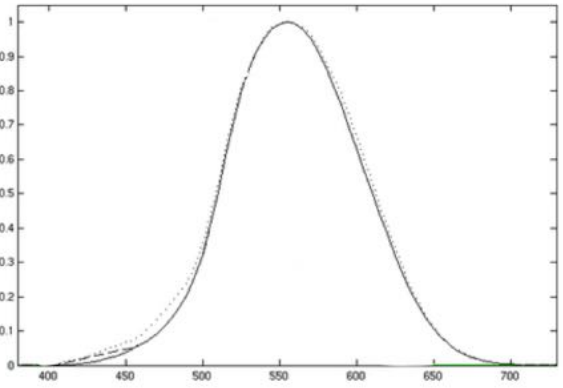
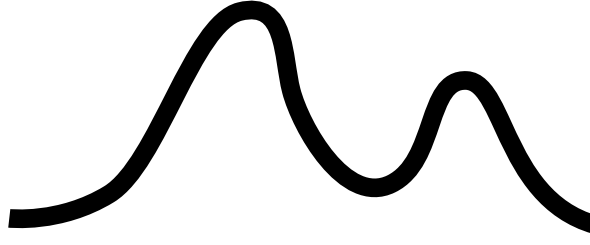
$$Y = \int \Phi(\lambda) V(\lambda) d\lambda$$

- **Luminous efficiency is measure of how bright light at a given wavelength is perceived by a human (due to the eye's response to light at that wavelength)**



- **How to measure the eye's response curve $V(\lambda)$?**
 - **Adjust power of monochromatic light source of wavelength λ until it matches the brightness of reference 555 nm source (photopic case)**
 - **Notice: the sensitivity of photopic eye is maximized at ~ 555 nm**

Lightness (perceived brightness) aka luma

Lightness (L^*) $\xleftarrow{?}$ Luminance (Y) = \int_{λ}  * 

(Perceived by brain) (Response of eye)

Spectral sensitivity of eye (eye's response curve)

Radiance (energy spectrum from scene)

Dark adapted eye: $L^* \propto Y^{0.4}$

Bright adapted eye: $L^* \propto Y^{0.5}$

In a dark room, you turn on a light with luminance: Y_1

You turn on a second light that is identical to the first. Total output is now: $Y_2 = 2Y_1$

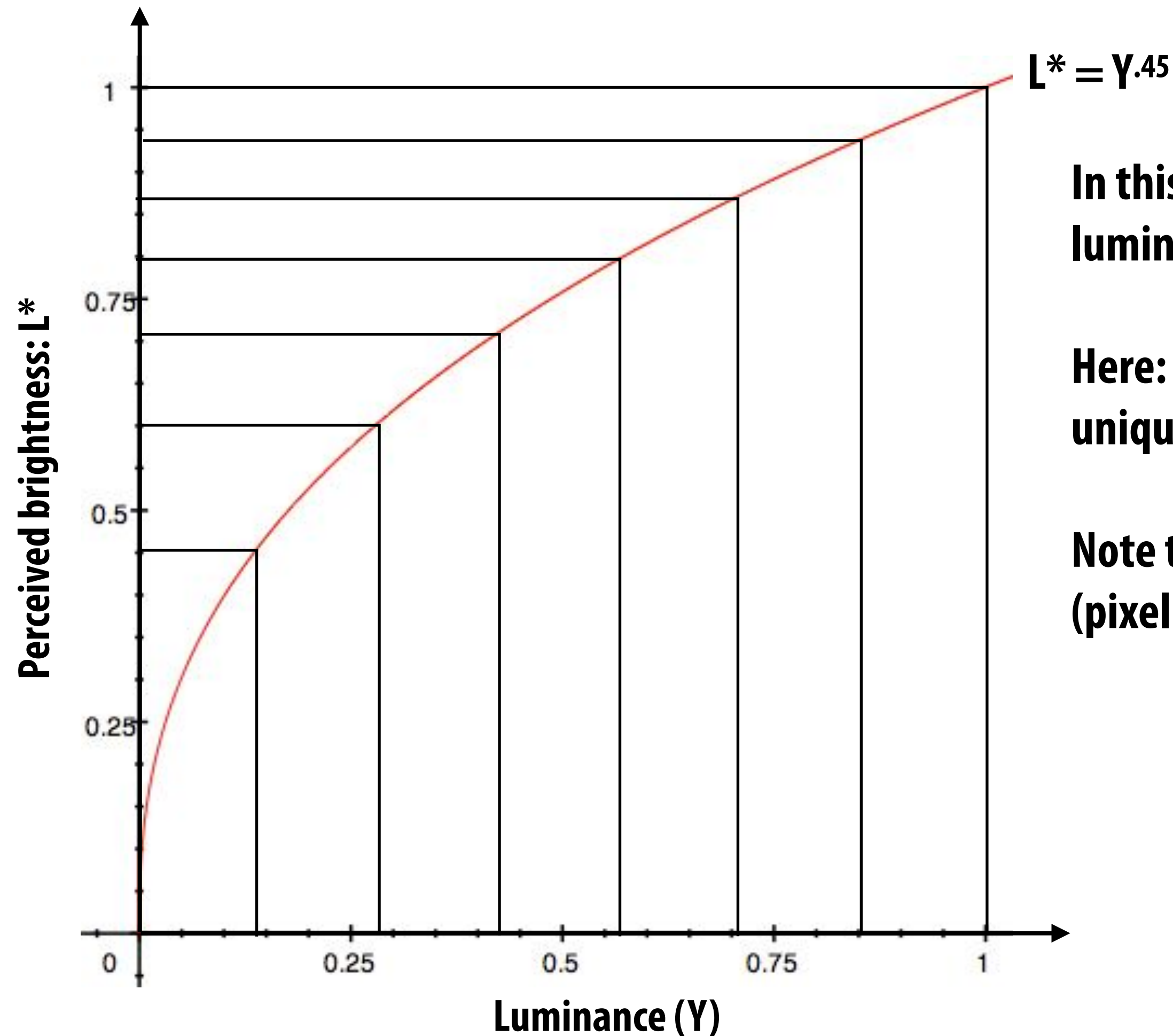
Total output appears $2^{0.4} = 1.319$ times brighter to dark-adapted human

Note: Lightness (L^*) is often referred to as luma (Y')

Idea 1:

- **What is the most efficient way to encode intensity values as a byte?**
- **Idea: encode based on how the brain *perceives brightness*, not based on the response of eye**

Consider an image with pixel values encoding luminance (linear in energy hitting sensor)



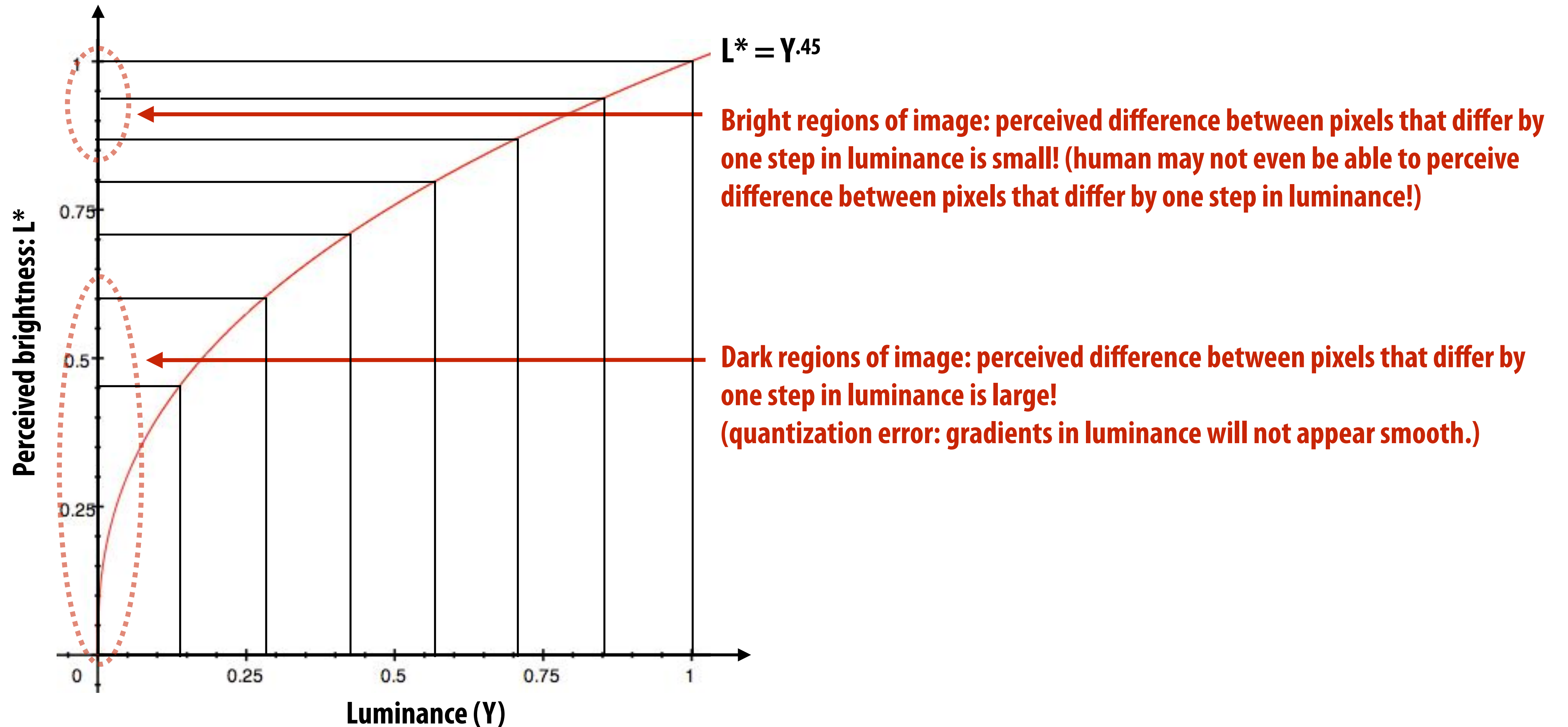
In this visualization: Pixel can represent 8 unique luminance values (3-bits/pixel)

Here: lines indicate luminance associated with each unique pixel value

Note that "spacing" of pixel values is linear in luminance (pixel value encode equally spaced sensor responses)

Problem: quantization error

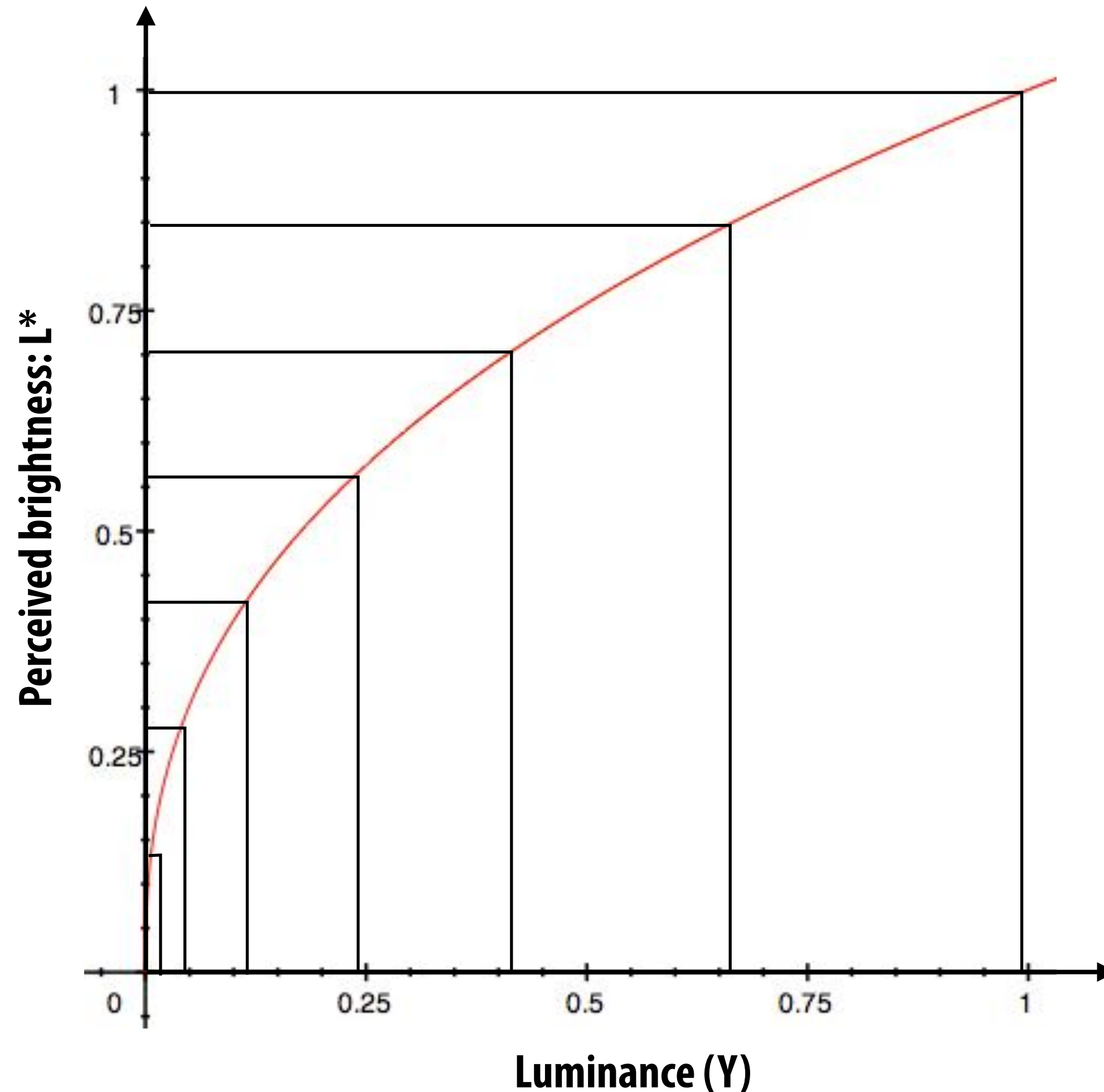
Many common image formats store 8 bits per channel (256 unique values)
Insufficient precision to represent brightness in darker regions of image



Rule of thumb: human eye cannot differentiate <1% differences in luminance

Store lightness, not luminance

Idea: distribute representable pixel values evenly with respect to lightness (perceived brightness), not evenly in luminance (**make more efficient use of available bits**)



Solution: pixel stores $Y^{0.45}$

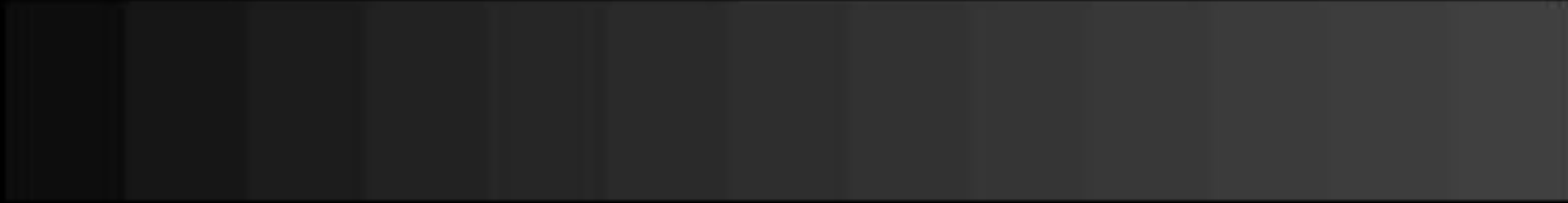
Must compute $(\text{pixel_value})^{2.2}$ prior to display on LCD

Warning: must take caution with subsequent pixel processing operations once pixels are encoded in a space that is not linear in luminance.

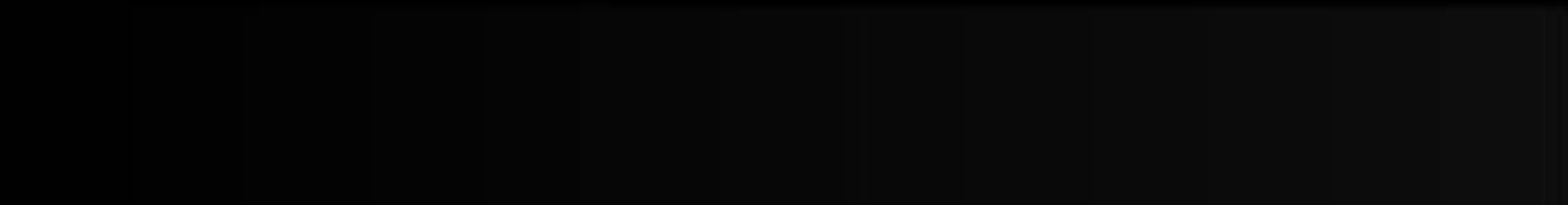
e.g., When adding images should you add pixel values that are encoded as lightness or as luminance?

[Everything on this slide is dark to make it easier to see the differences]

Equal steps (in luminance)



Equal steps (in brightness)



Idea 2:

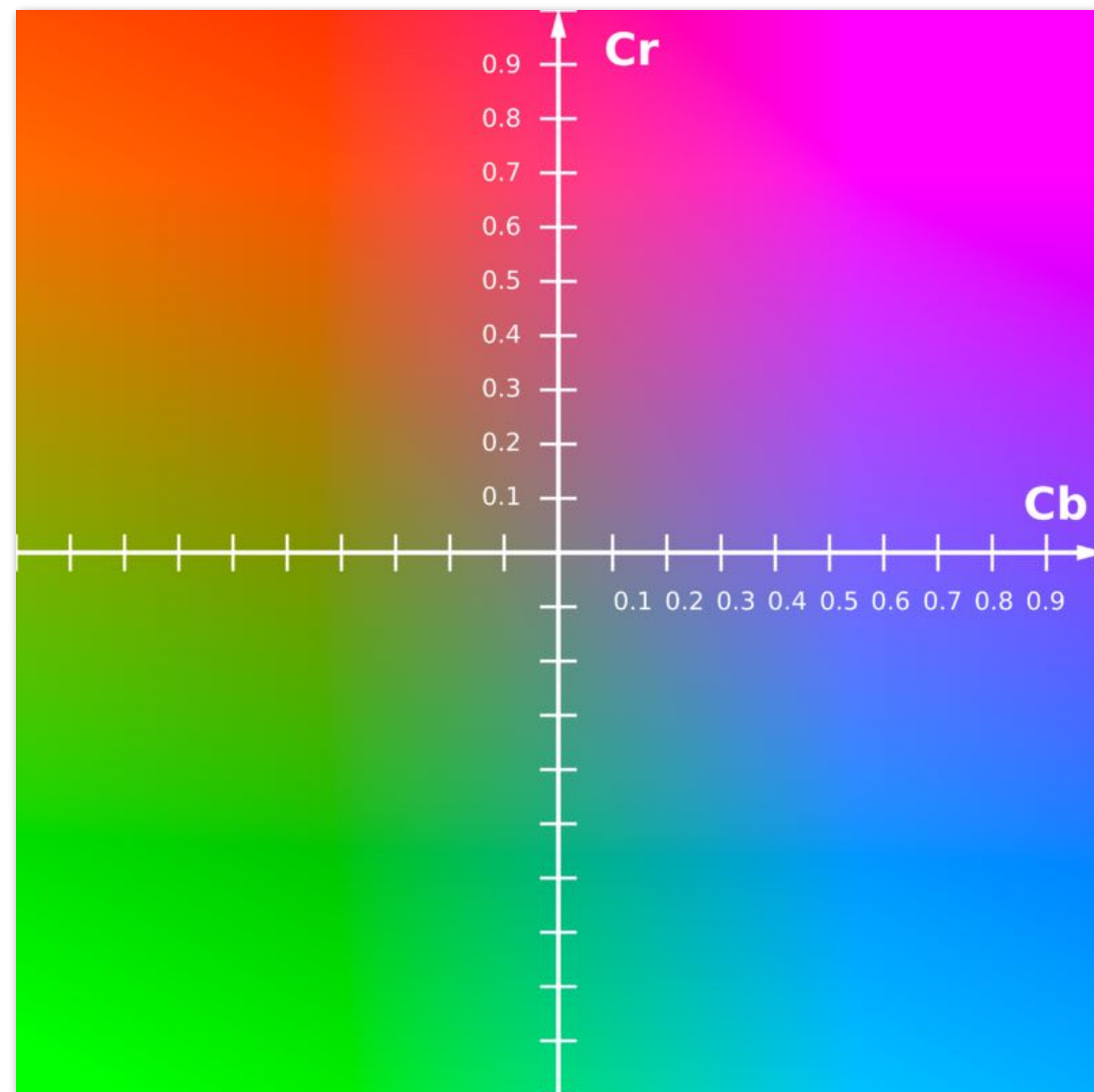
- **Chrominance (“chroma”) subsampling**
- **The human visual system is less sensitive to detail in chromaticity than in luminance**
 - **So it is sufficient to sample chroma more sparsely in space**

Y'CbCr color space

Y' = luma: perceived luminance (non-linear)

Cb = blue-yellow deviation from gray

Cr = red-cyan deviation from gray



CbCr plane at a fixed value of Y'

Non-linear RGB
(primed notation indicates
perceptual (non-linear) space)

Conversion from R'G'B' to Y'CbCr:

$$\begin{aligned}
 Y' &= 16 + \frac{65.738 \cdot R'_D}{256} + \frac{129.057 \cdot G'_D}{256} + \frac{25.064 \cdot B'_D}{256} \\
 C_B &= 128 + \frac{-37.945 \cdot R'_D}{256} - \frac{74.494 \cdot G'_D}{256} + \frac{112.439 \cdot B'_D}{256} \\
 C_R &= 128 + \frac{112.439 \cdot R'_D}{256} - \frac{94.154 \cdot G'_D}{256} - \frac{18.285 \cdot B'_D}{256}
 \end{aligned}$$



Y'



Cb



Cr

Example: compression in Y'CbCr



Original picture of Kayvon

Example: compression in Y'CbCr



**Contents of CbCr color channels downsampled by a factor of 20 in each dimension
(400x reduction in number of samples)**

Example: compression in Y'CbCr



Full resolution sampling of luma (Y')

Example: compression in Y'CbCr



**Reconstructed result
(looks pretty good)**

Chroma subsampling

Y'CbCr is an efficient representation for storage (and transmission) because Y' can be stored at higher resolution than CbCr without significant loss in perceived visual quality

Y'₀₀ Cb₀₀ Cr₀₀	Y'₁₀	Y'₂₀ Cb₂₀ Cr₂₀	Y'₃₀
Y'₀₁ Cb₀₁ Cr₀₁	Y'₁₁	Y'₂₁ Cb₂₁ Cr₂₁	Y'₃₁

4:2:2 representation:

Store Y' at full resolution

**Store Cb, Cr at full vertical resolution,
but only half horizontal resolution**

X:Y:Z notation:

X = width of block

Y = number of chroma samples in first row

Z = number of chroma samples in second row

Y'₀₀ Cb₀₀ Cr₀₀	Y'₁₀	Y'₂₀ Cb₂₀ Cr₂₀	Y'₃₀
Y'₀₁	Y'₁₁	Y'₂₁	Y'₃₁

4:2:0 representation:

Store Y' at full resolution

**Store Cb, Cr at half resolution in both
dimensions**

Real-world 4:2:0 examples:

most JPG images and H.264 video

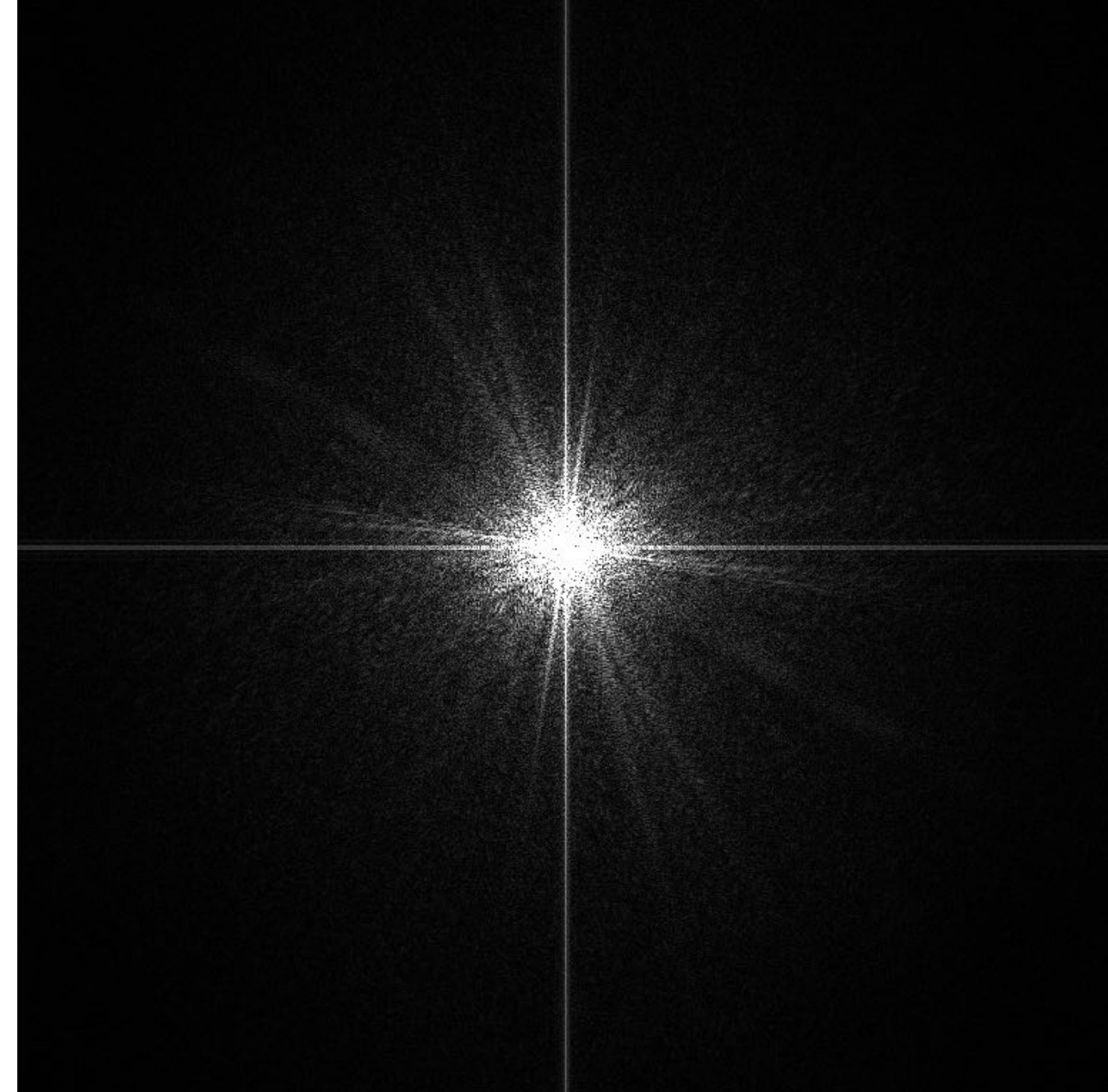
Idea 3:

- **Low frequency content is predominant in the real world**
- **The human visual system is less sensitive to high frequency sources of error in images**
- **So a good compression scheme needs to accurately represent lower frequencies, but it can be acceptable to sacrifice accuracy in representing higher frequencies**

Recall: frequency content of images



Spatial domain result

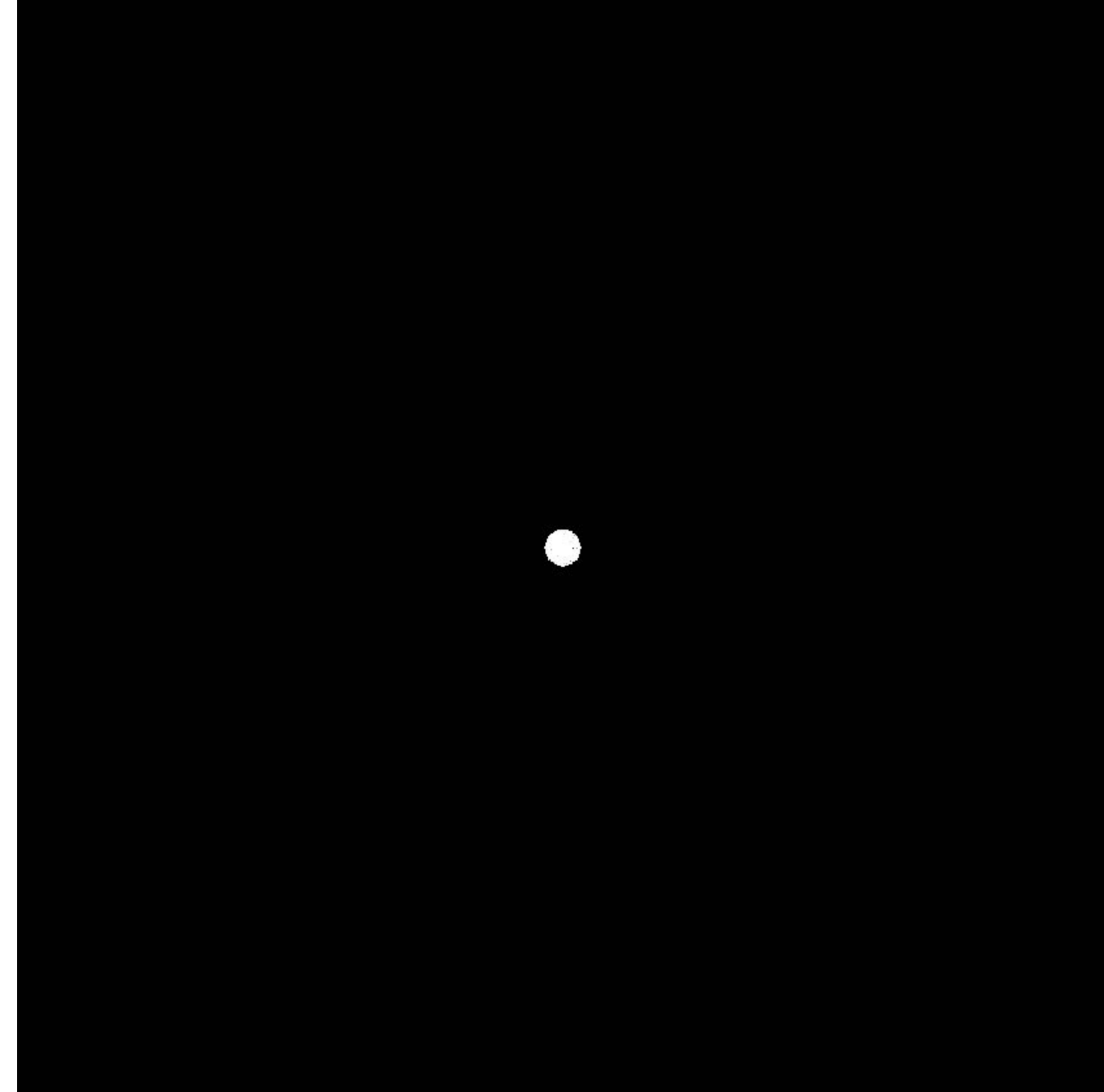


Spectrum of image

Recall: frequency content of images

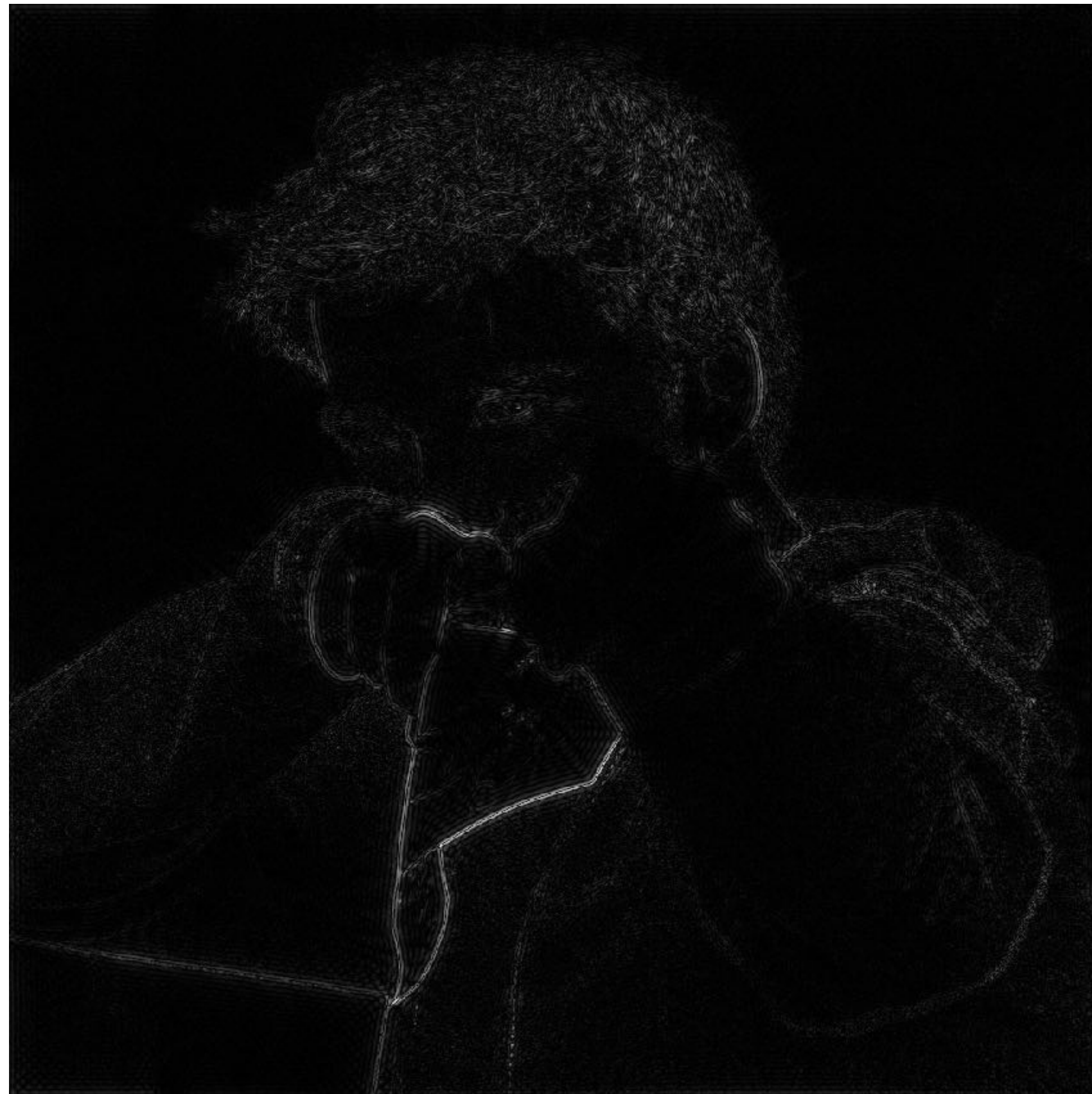


Spatial domain result

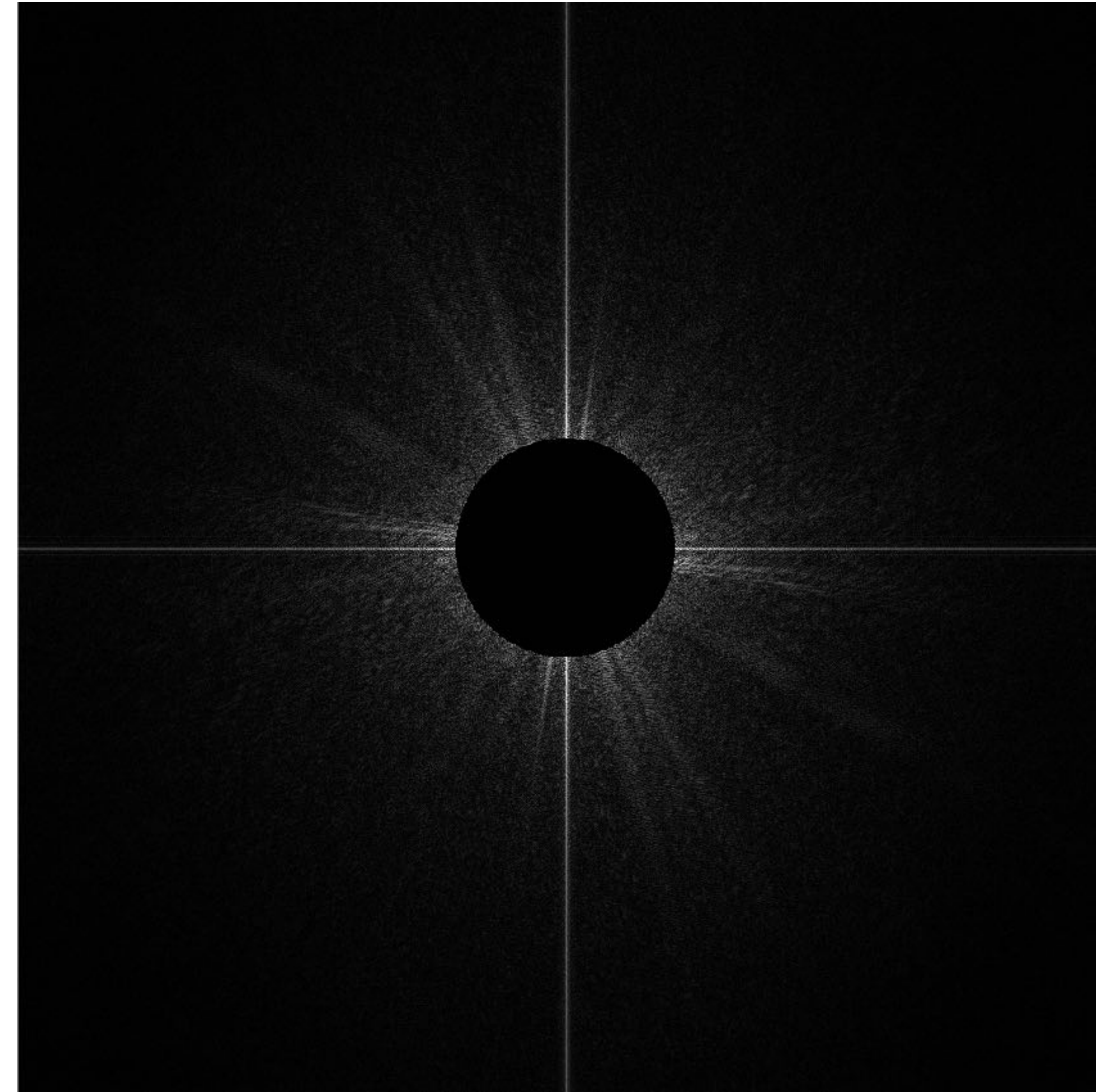


Spectrum (after low-pass filter)
All frequencies above cutoff have 0 magnitude

Recall: frequency content of images



**Spatial domain result
(strongest edges)**



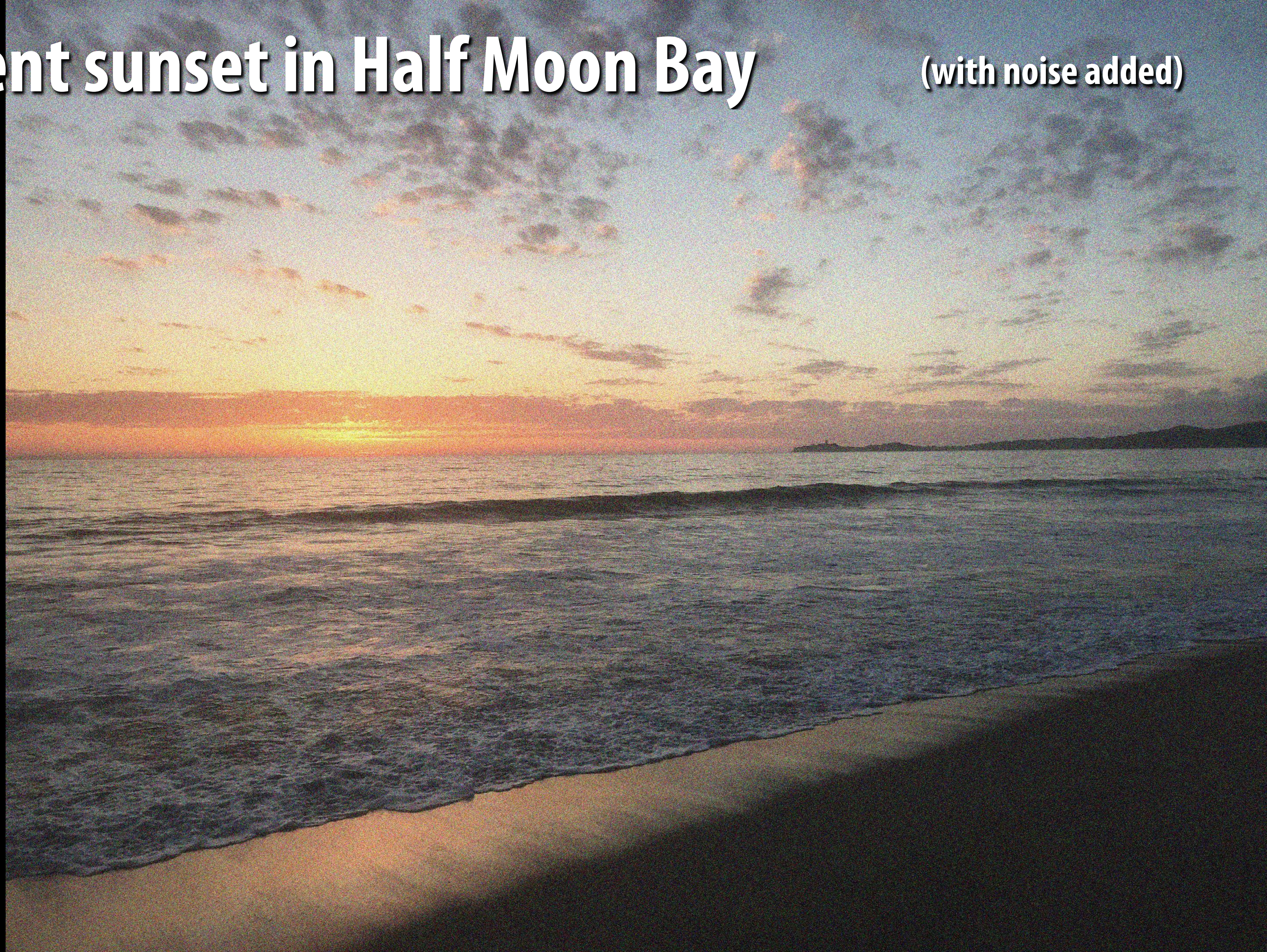
**Spectrum (after high-pass filter)
All frequencies below threshold
have 0 magnitude**

A recent sunset in Half Moon Bay



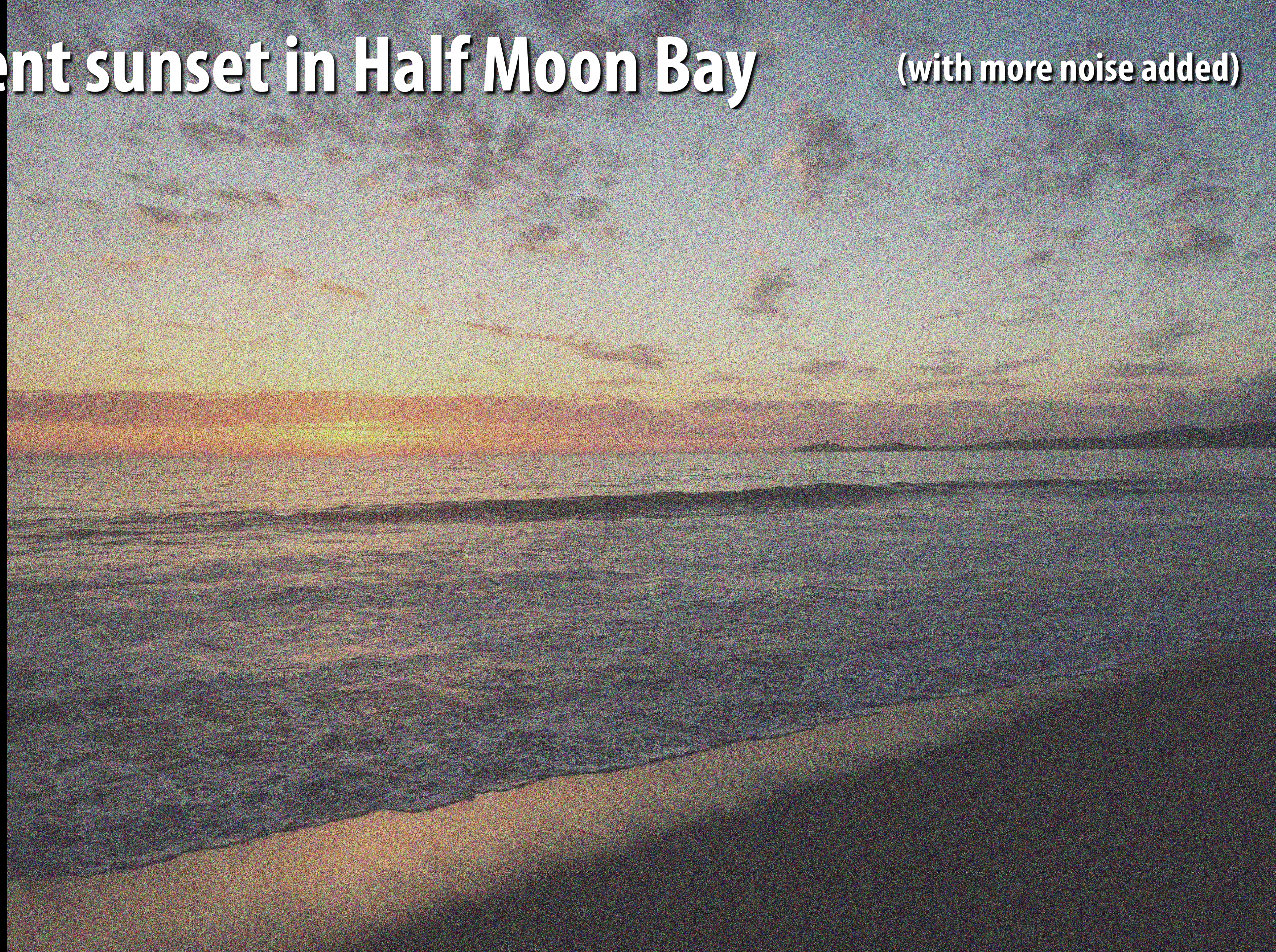
A recent sunset in Half Moon Bay

(with noise added)

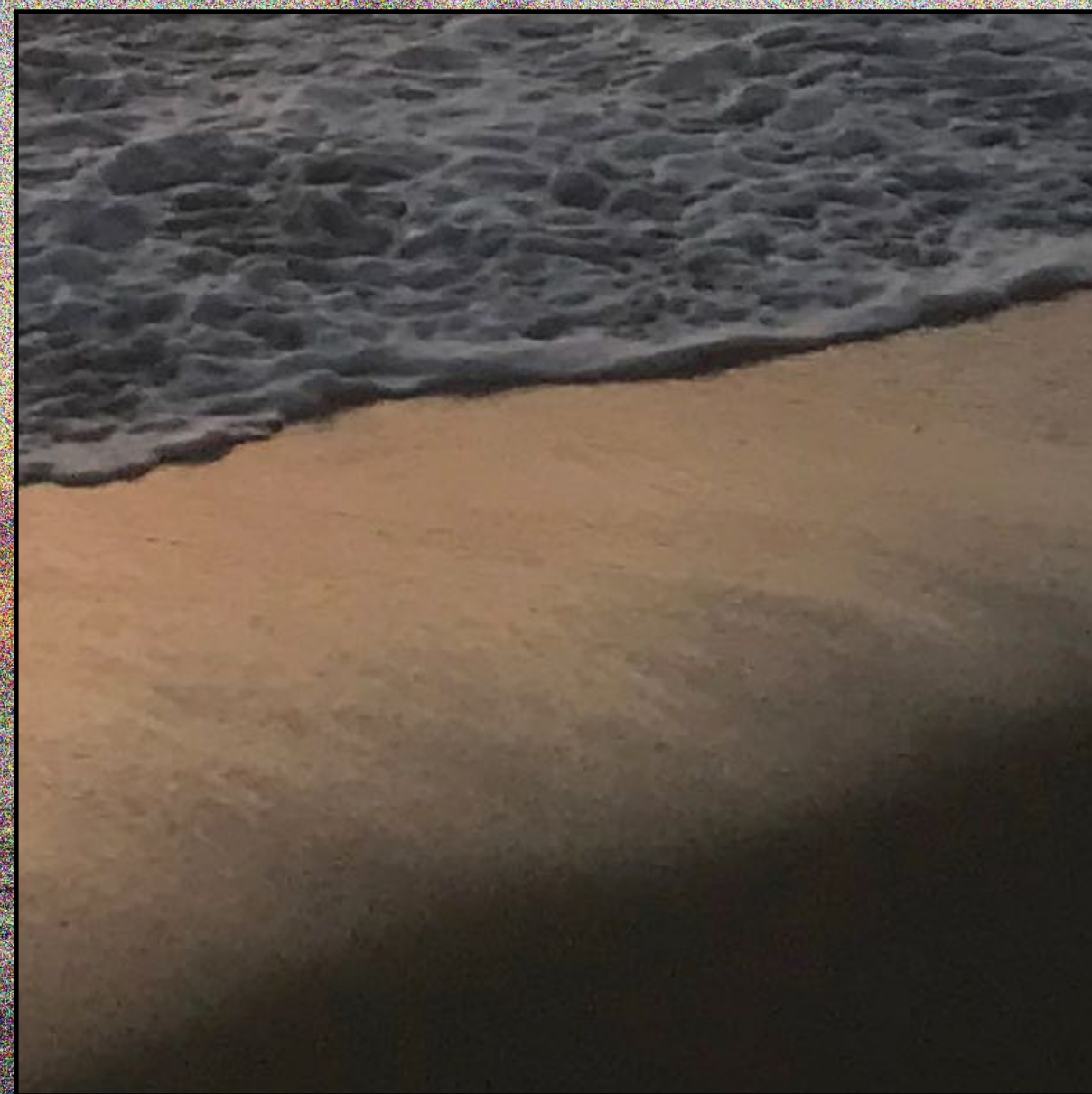


A recent sunset in Half Moon Bay

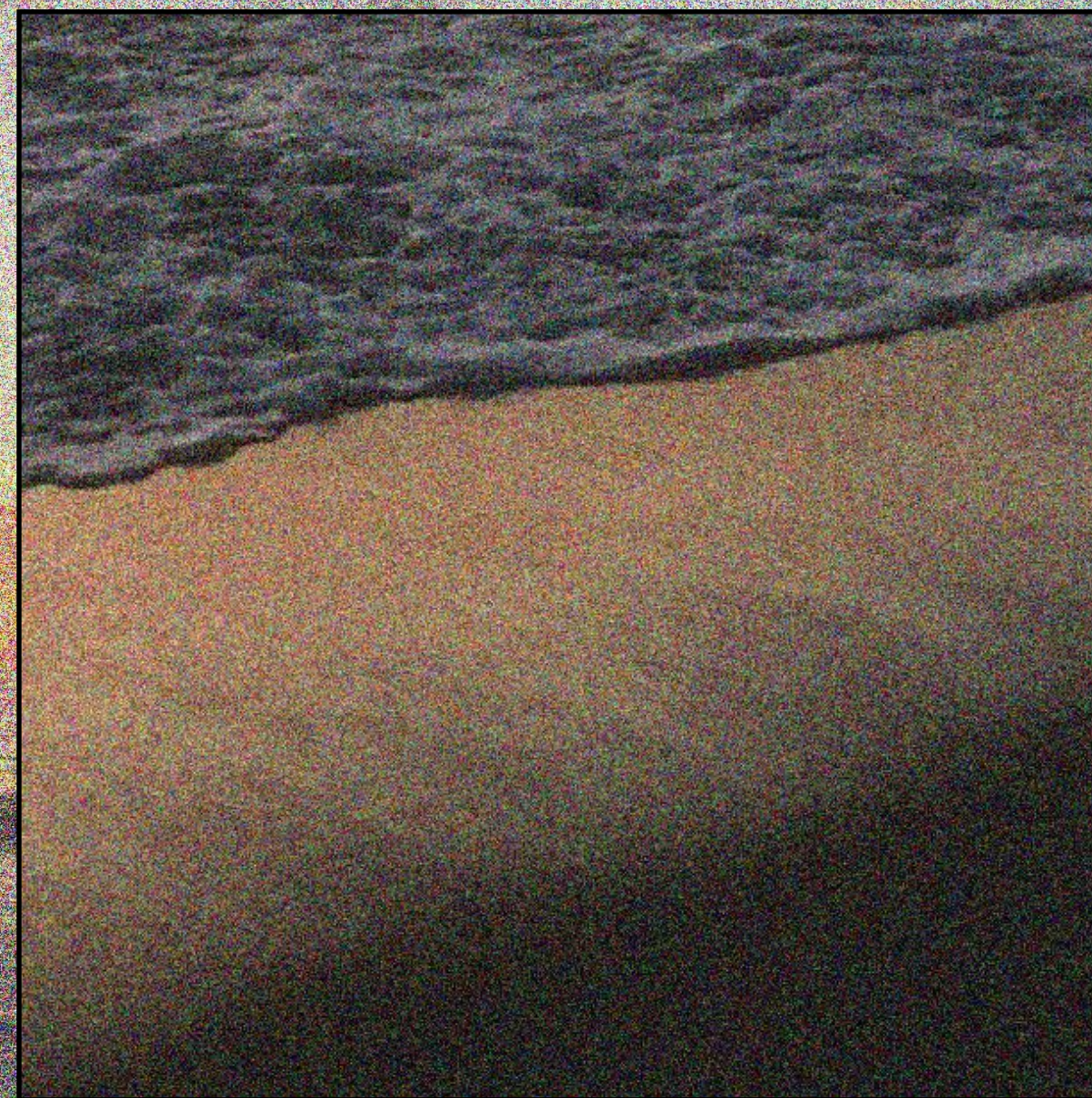
(with more noise added)



A recent sunset in Half Moon Bay



Original image



Noise added
(increases high frequency content)



More noise added



What is a good representation for manipulating frequency content of images?

Hint:

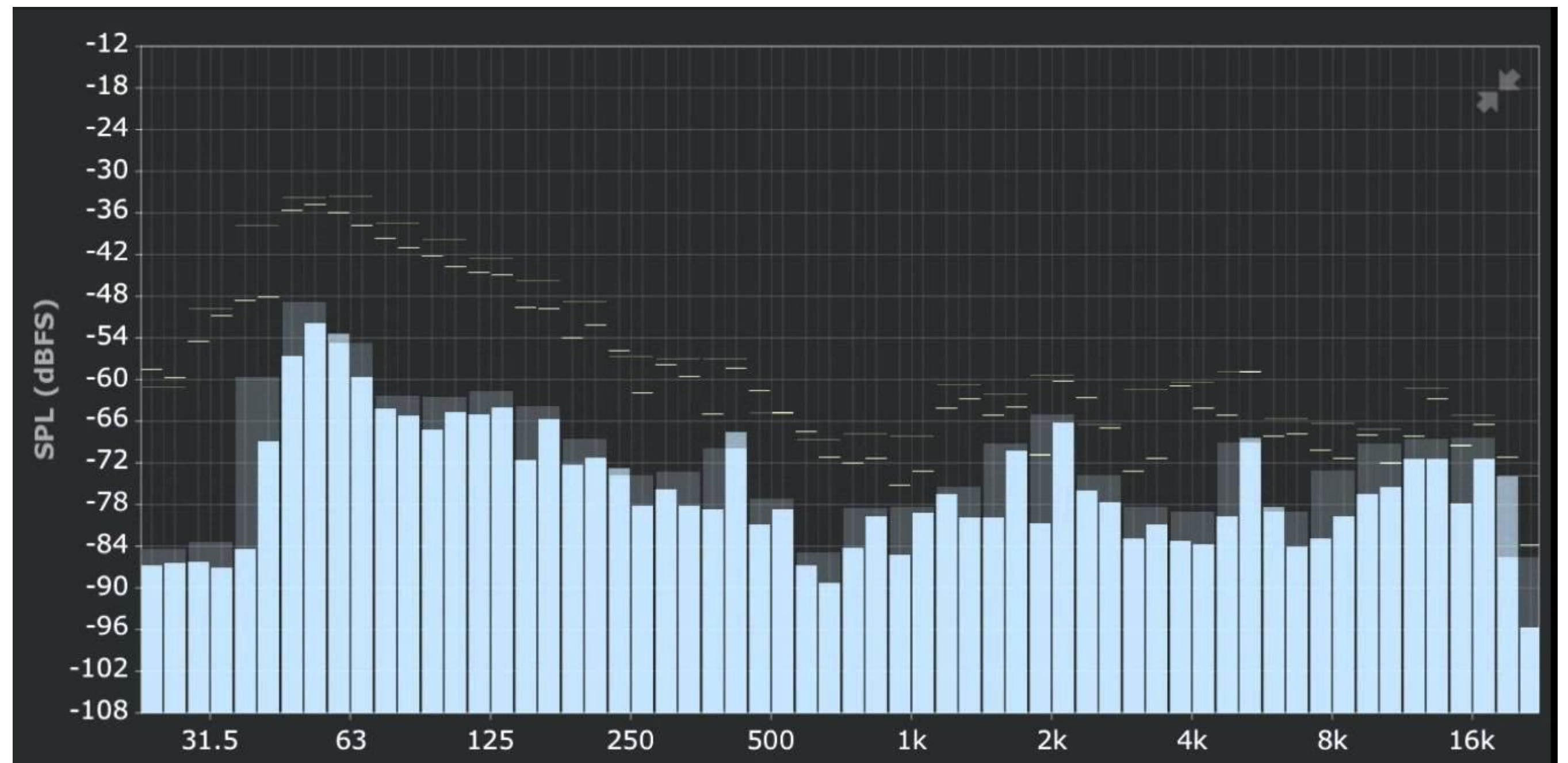
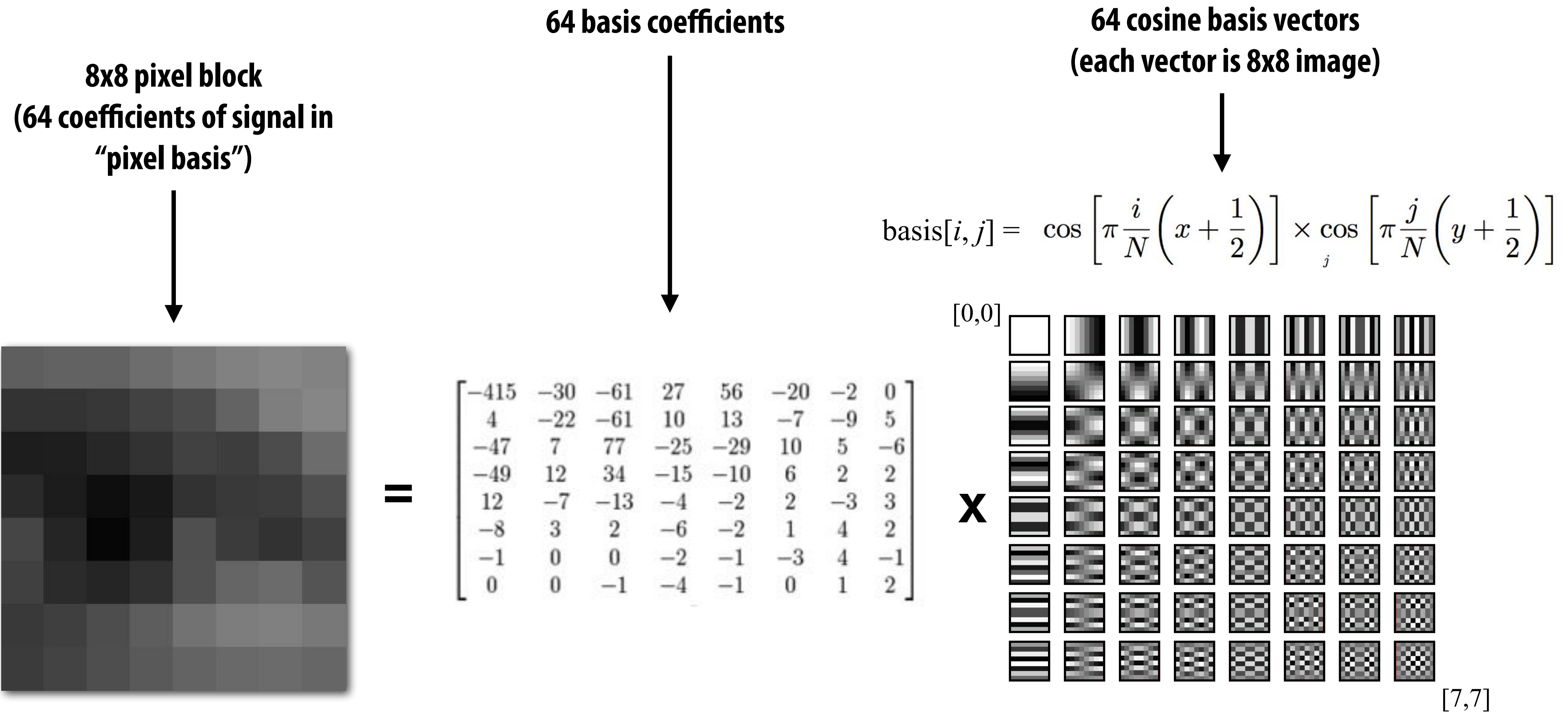


Image transform coding using the discrete cosign transform (DCT)



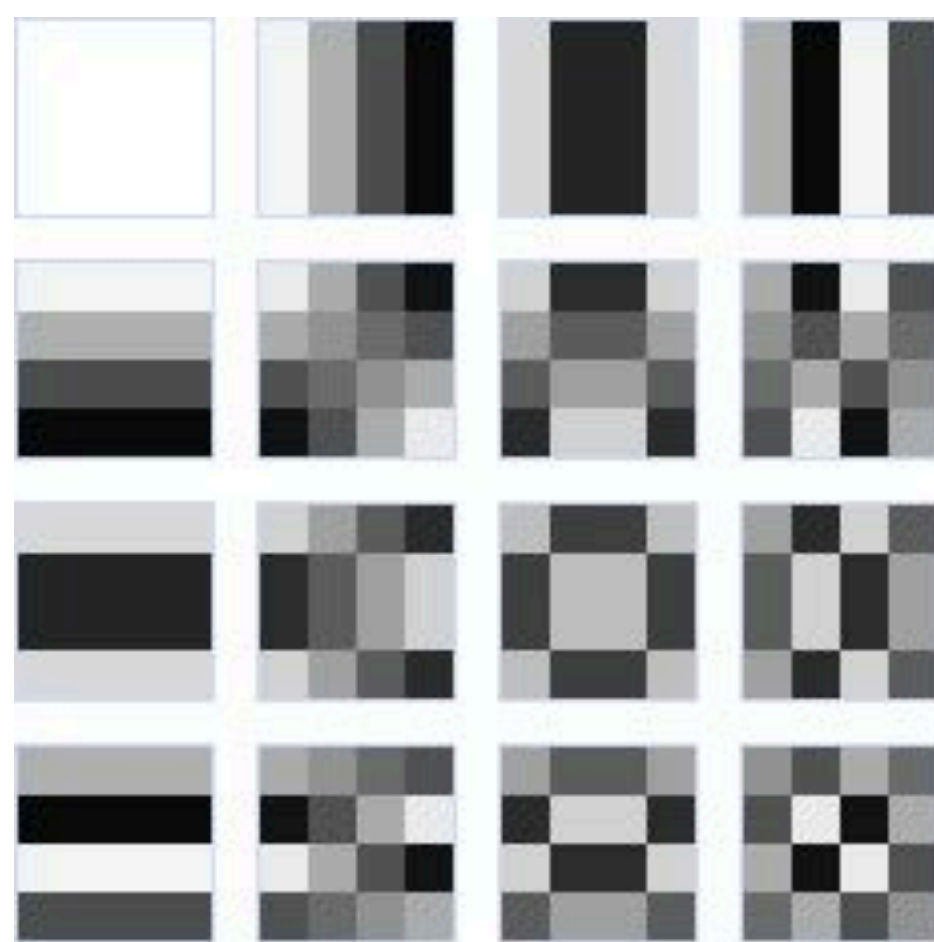
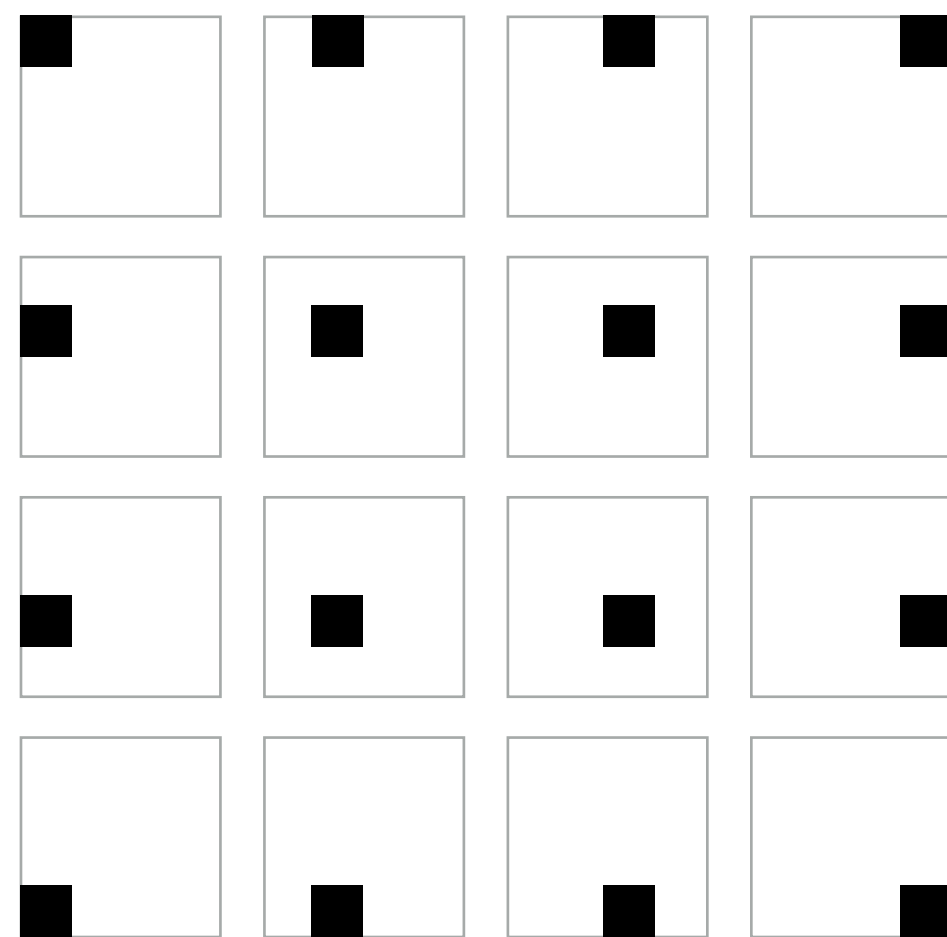
In practice: DCT is applied to 8x8 pixel blocks of Y' channel, 16x16 pixel blocks of Cb, Cr (assuming 4:2:0)

Examples of other bases

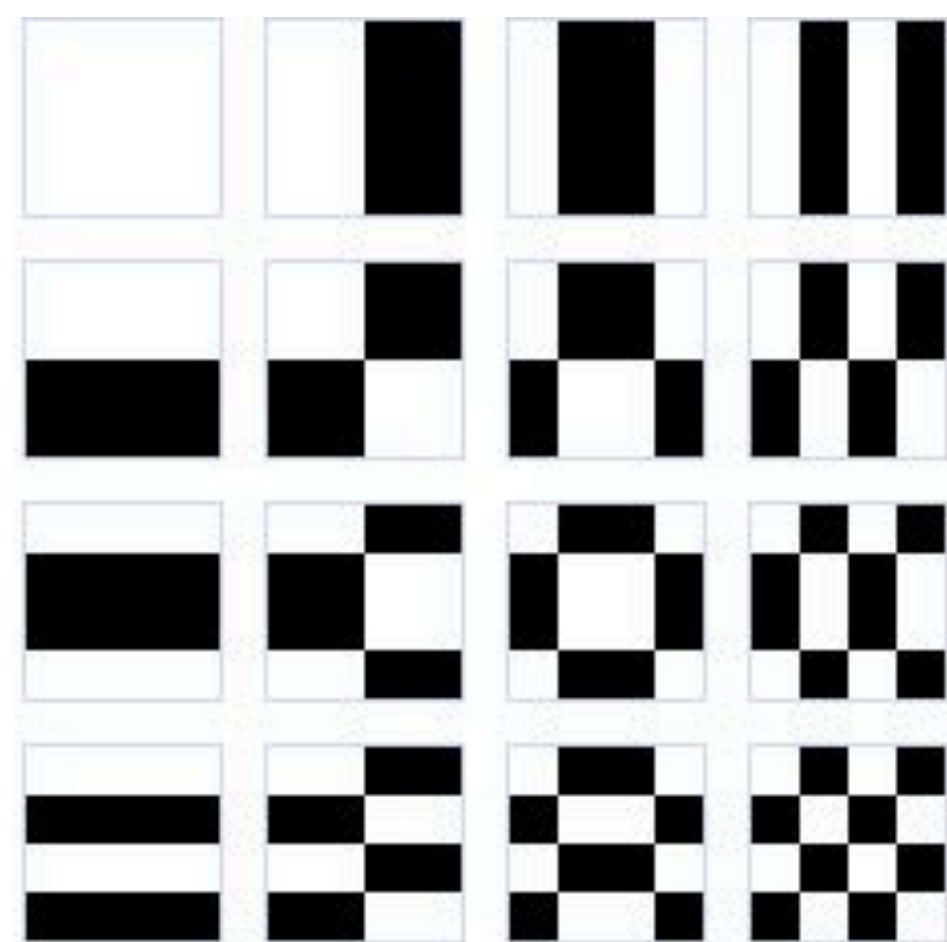
This slide illustrates basis images for 4x4 block of pixels (although JPEG works on 8x8 blocks)

Pixel Basis

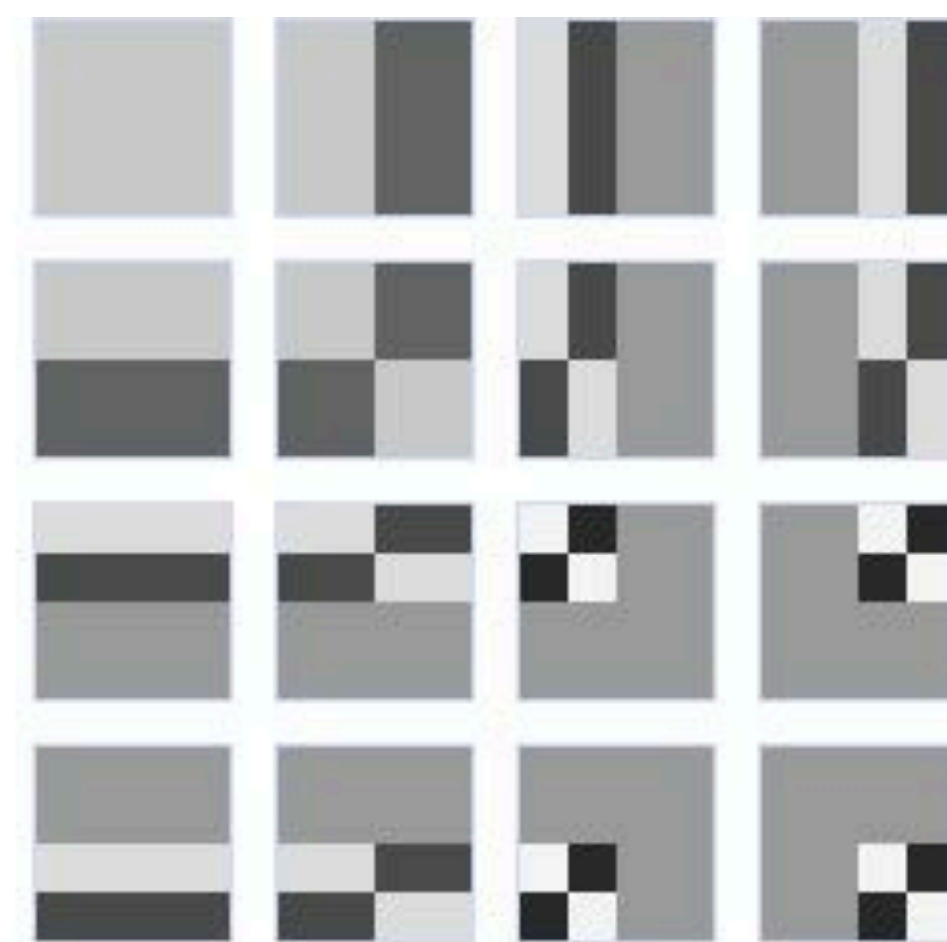
(Compact: each coefficient in representation only effects a single pixel of output)



DCT



Walsh-Hadamard



Haar Wavelet

Quantization

$$\begin{bmatrix} -415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\ 4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\ -47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\ -49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\ 12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\ -8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\ -1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\ 0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \end{bmatrix}$$

Result of DCT

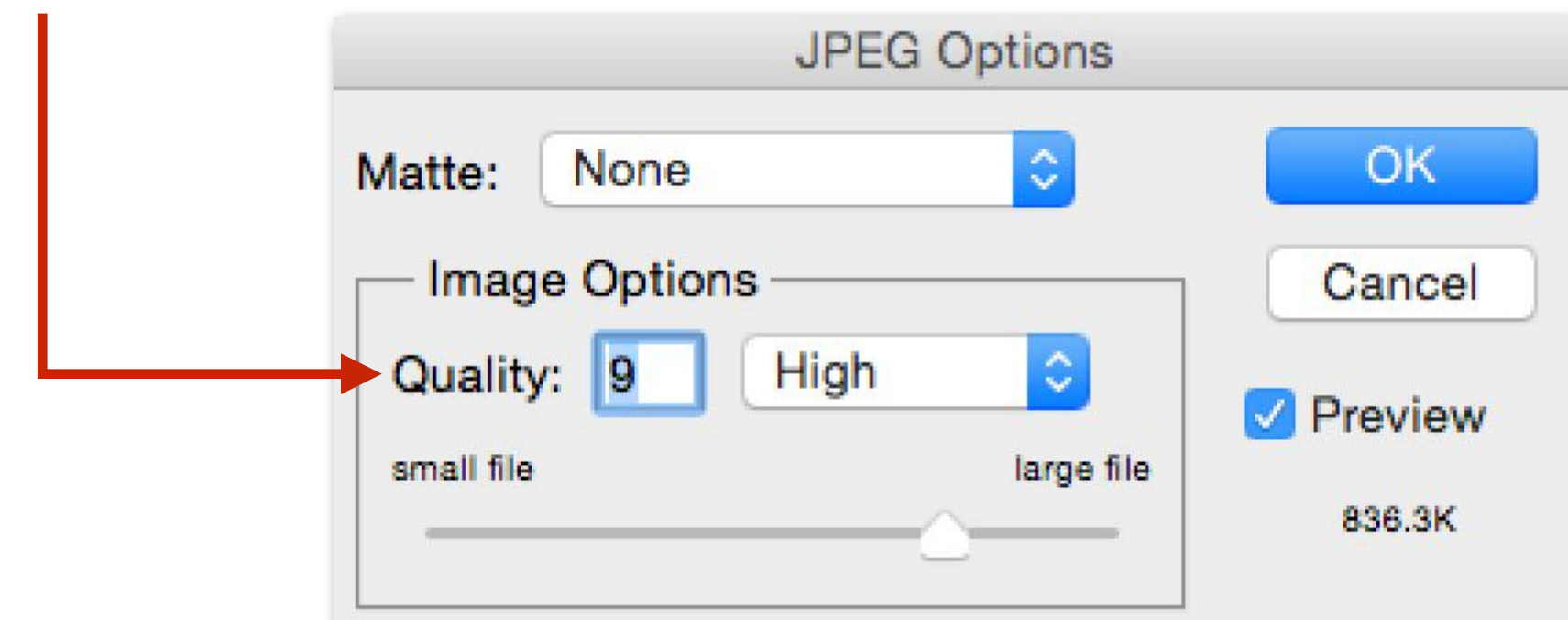
(representation of image in cosine basis)

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Quantization Matrix

Changing JPEG quality setting in your favorite photo app modifies this matrix ("lower quality" = higher values for elements in quantization matrix)

$$= \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$



Quantization produces small values for coefficients (only few bits needed per coefficient)

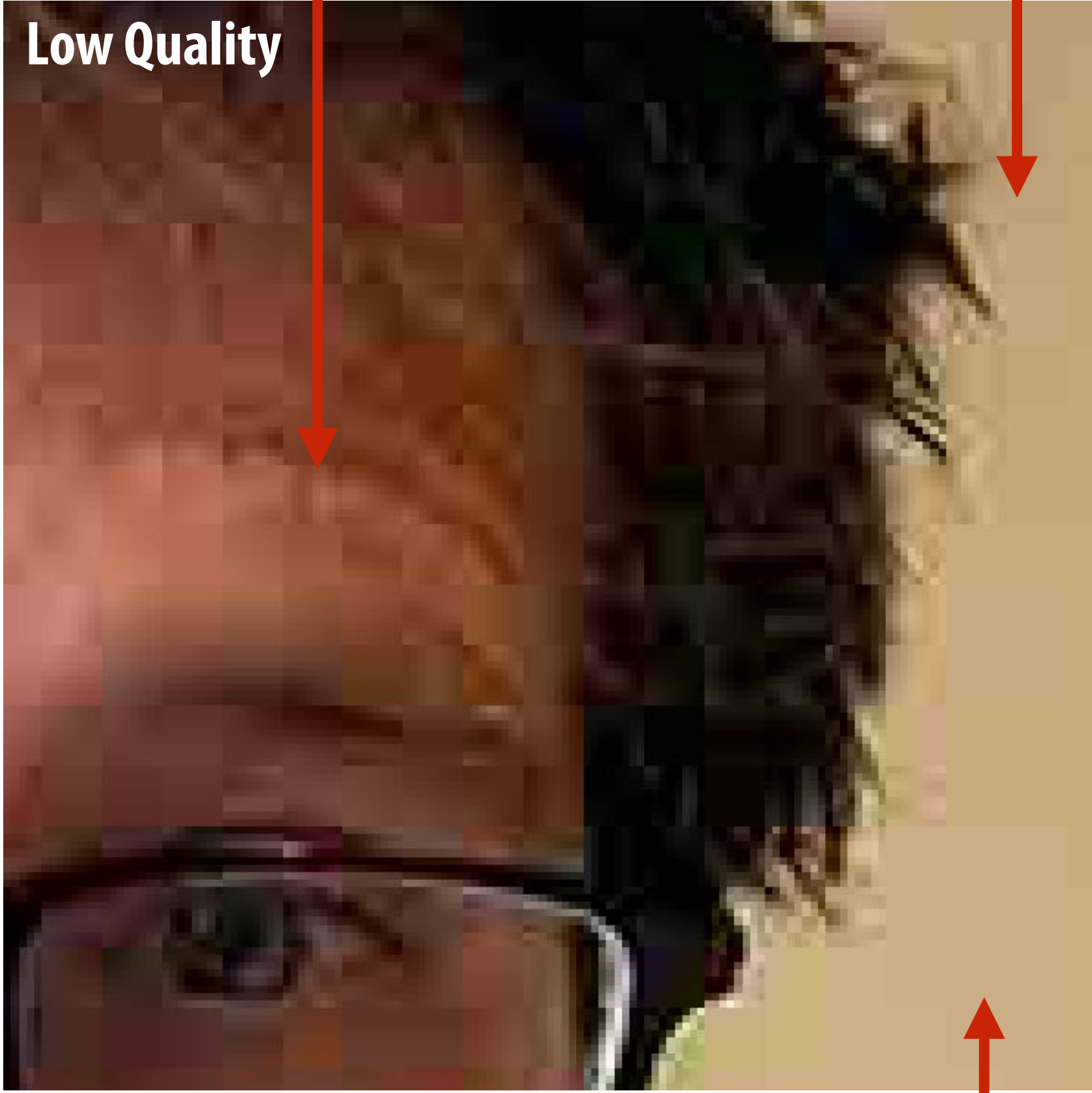
Quantization zeros out many coefficients

JPEG compression artifacts



Noticeable 8x8 pixel block boundaries

Noticeable error near high gradients

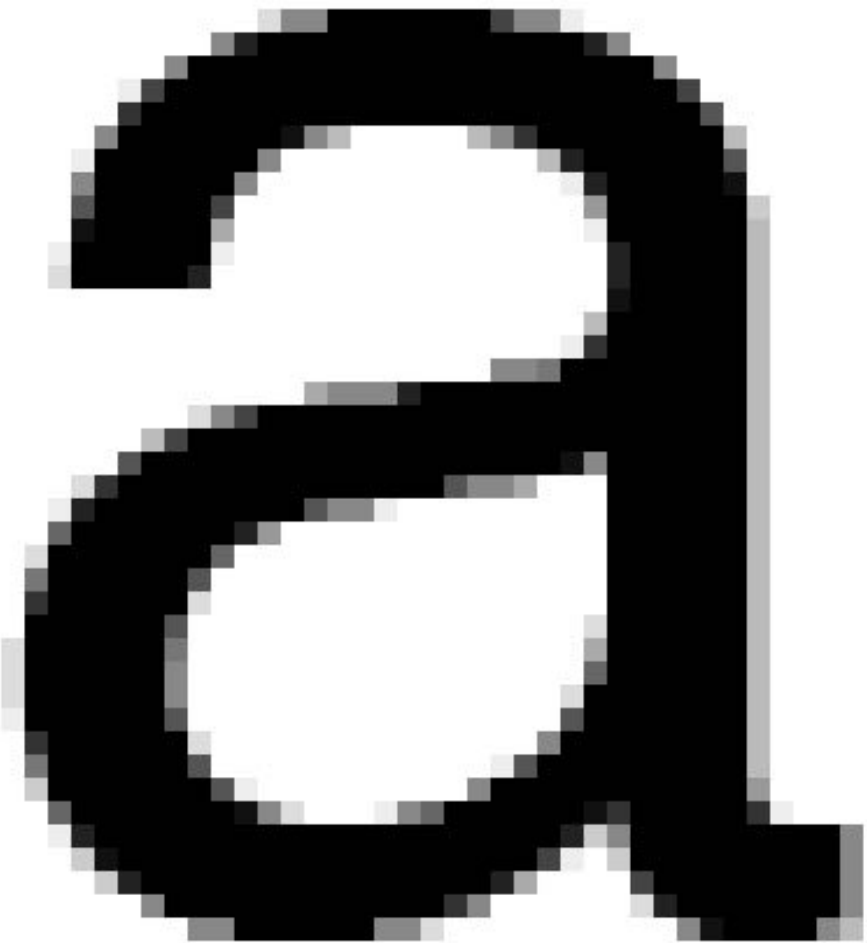


Low-frequency regions of image represented accurately even under high compression

JPEG compression artifacts



Original Image
(actual size)



Original Image



Quality Level 9



Quality Level 6



Quality Level 3



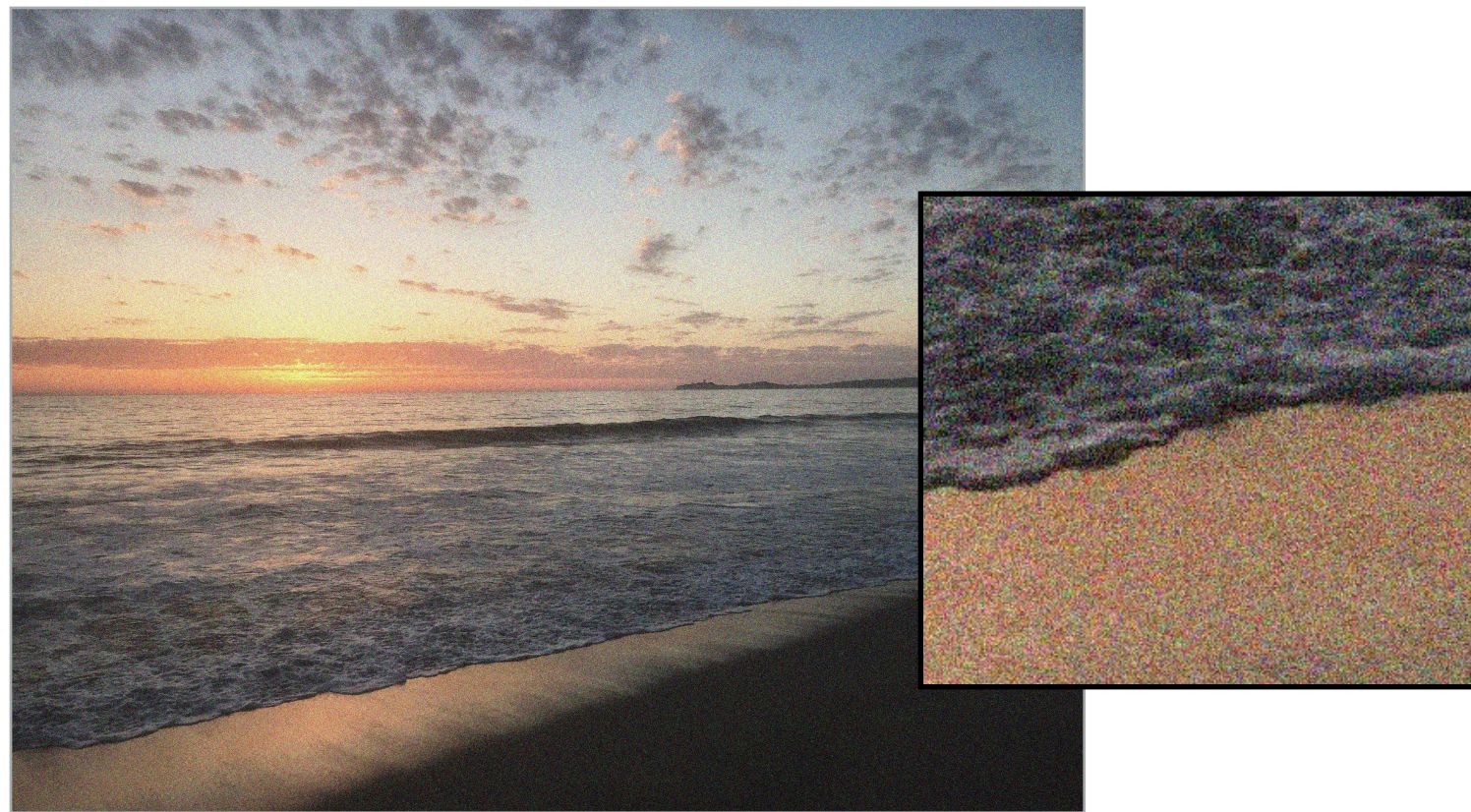
Quality Level 1

Why might JPEG compression not be a good compression scheme for illustrations and rasterized text?

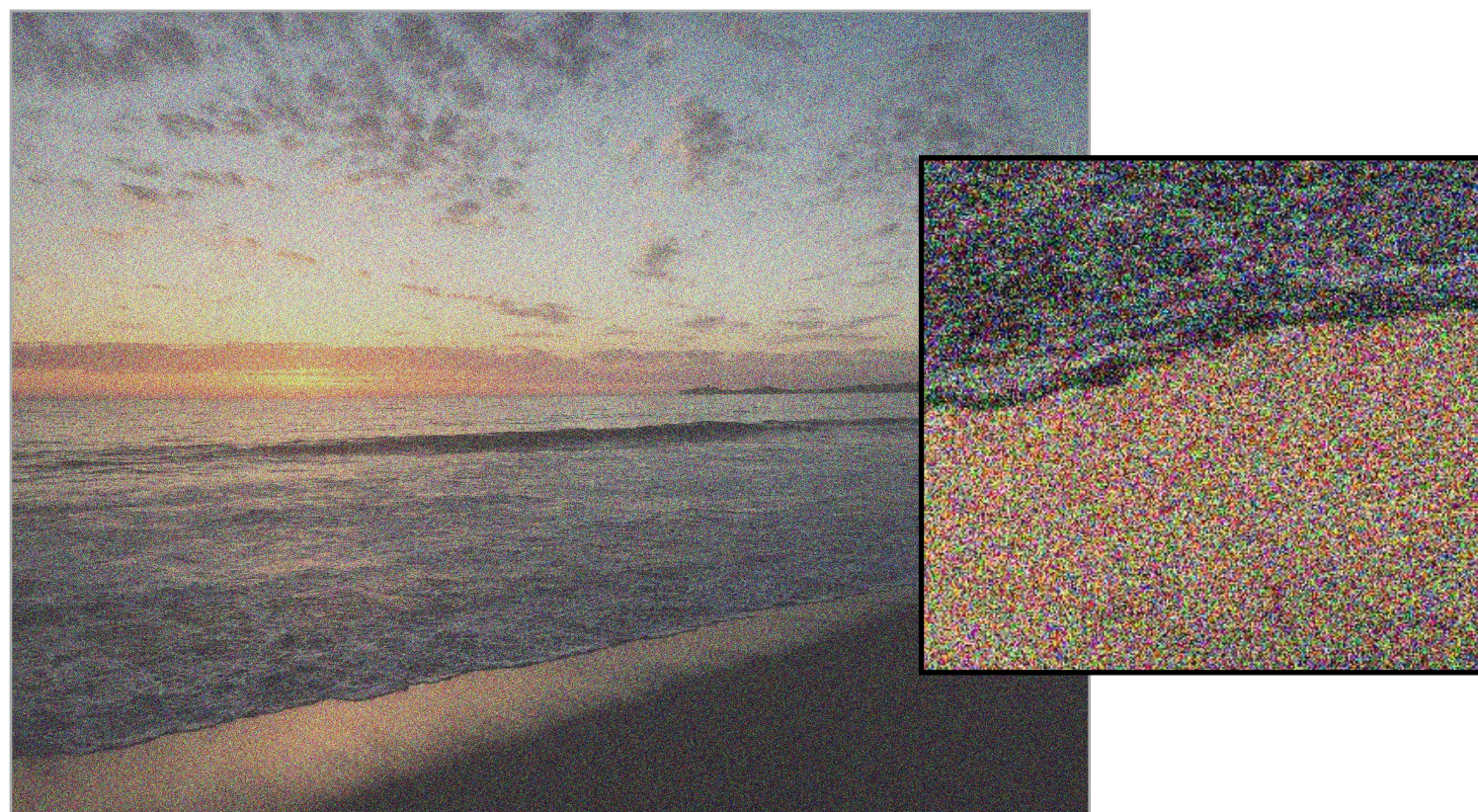
Images with high frequency content do not exhibit as high of compression ratios. Why?



Original image: 2.9MB JPG



Medium noise: 22.6 MB JPG



High noise: 28.9 MB JPG

**Photoshop JPG compression level = 10
used for all compressed images**

**Uncompressed image:
 $4032 \times 3024 \times 24 \text{ bytes/pixel} = 36.6 \text{ MB}$**

JPEG compression summary

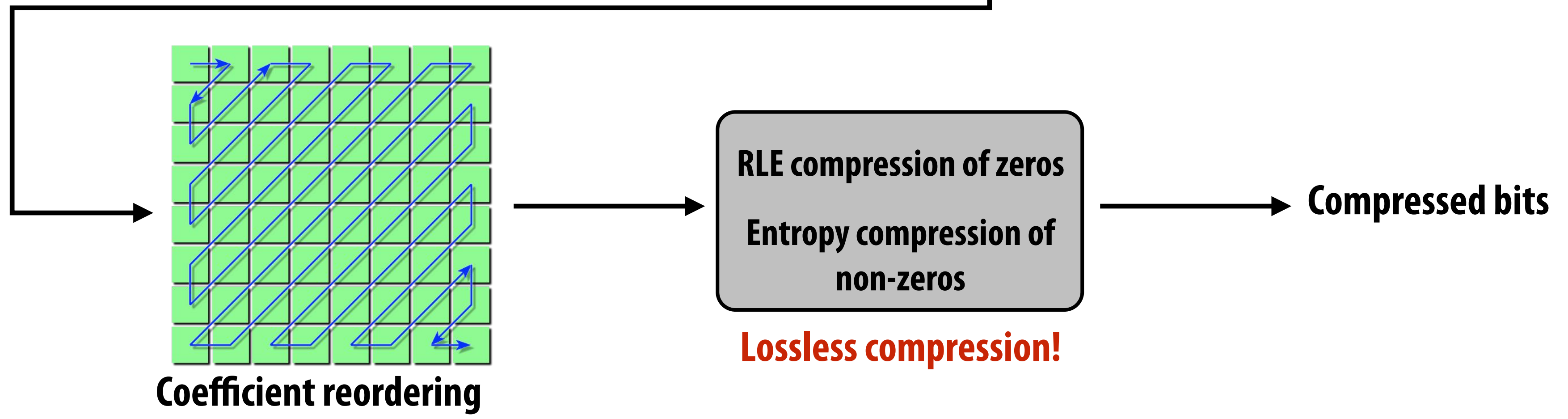
$$\begin{bmatrix} -415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\ 4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\ -47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\ -49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\ 12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\ -8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\ -1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\ 0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \end{bmatrix} \bigg/ \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

DCT
Quantization Matrix

$$= \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Quantized DCT

Quantization loses information (lossy compression!)



JPEG compression summary

Convert image to Y'CbCr

Downsample CbCr (to 4:2:2 or 4:2:0) (information loss occurs here)

For each color channel (Y', Cb, Cr):

For each 8x8 block of values

Compute DCT

Quantize results (information loss occurs here)

Reorder values

Run-length encode 0-spans

Huffman encode non-zero values

Key idea: exploit characteristics of human perception to build efficient image storage and image processing systems

- **Separation of luminance from chrominance in color representation (Y'CrCb) allows reduced resolution in chrominance channels (4:2:0)**
- **Encode pixel values linearly in lightness (perceived brightness), not in luminance (distribute representable values uniformly in perceptual space)**
- **JPEG compression significantly reduces file size at cost of quantization error in high spatial frequencies**
 - **Human brain is more tolerant of errors in high frequency image components than in low frequency ones**
 - **Images of the real world are dominated by low-frequency components**

Video compression: example

30 second video: 1920 x 1080, @ 30fps

Uncompressed: 8-bits per channel RGB → 24 bits/pixel → 6.2MB/frame
(6.2 MB * 30 sec * 30 fps = 5.2 GB)

Size of data when each frames stored as JPG: 531MB

Actual H.264 video file size: 65.4 MB (80-to-1 compression ratio, 8-to-1 compared to JPG)

Compression/encoding performed in real time on my iPhone



Go Swallows!

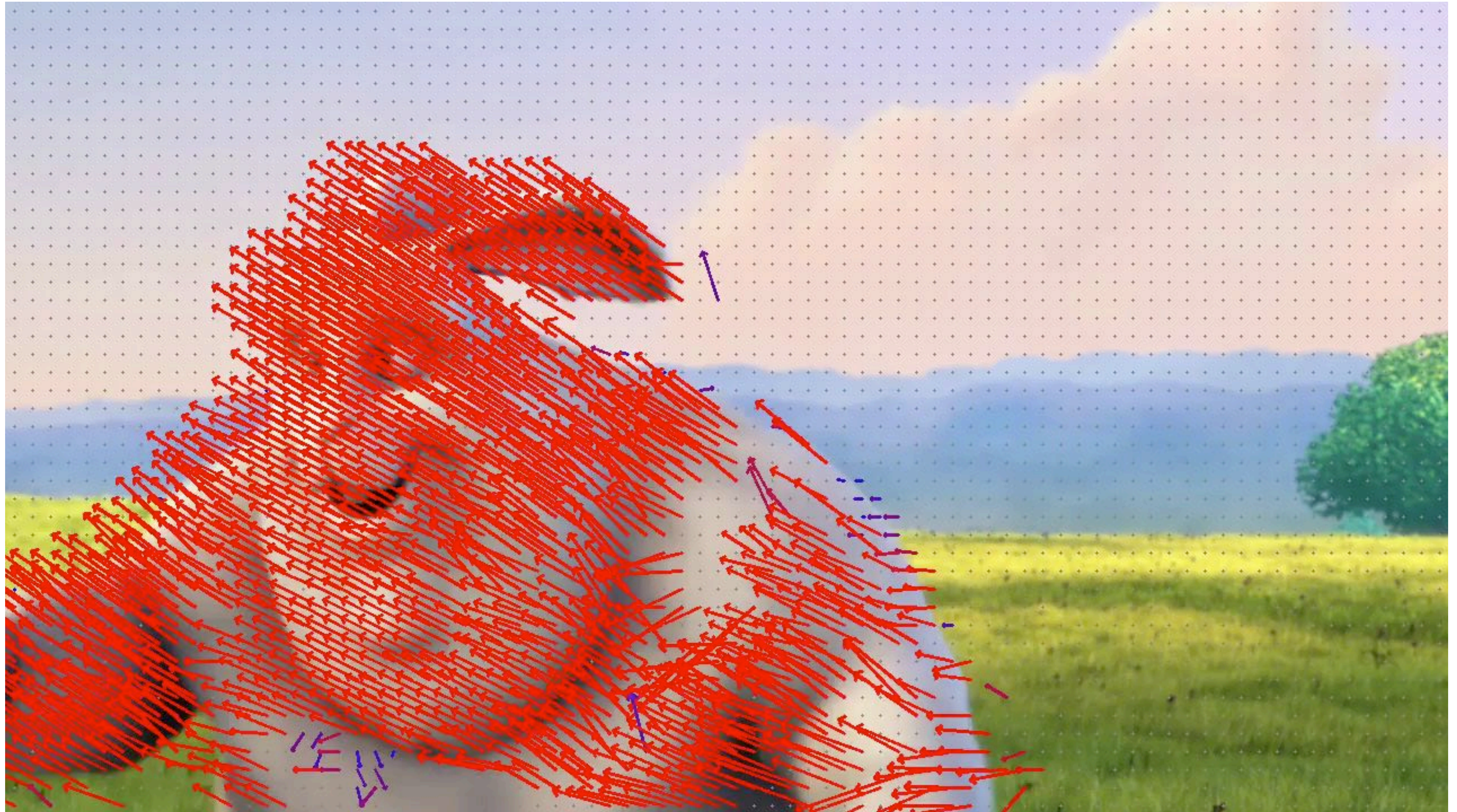


Video compression adds two main ideas

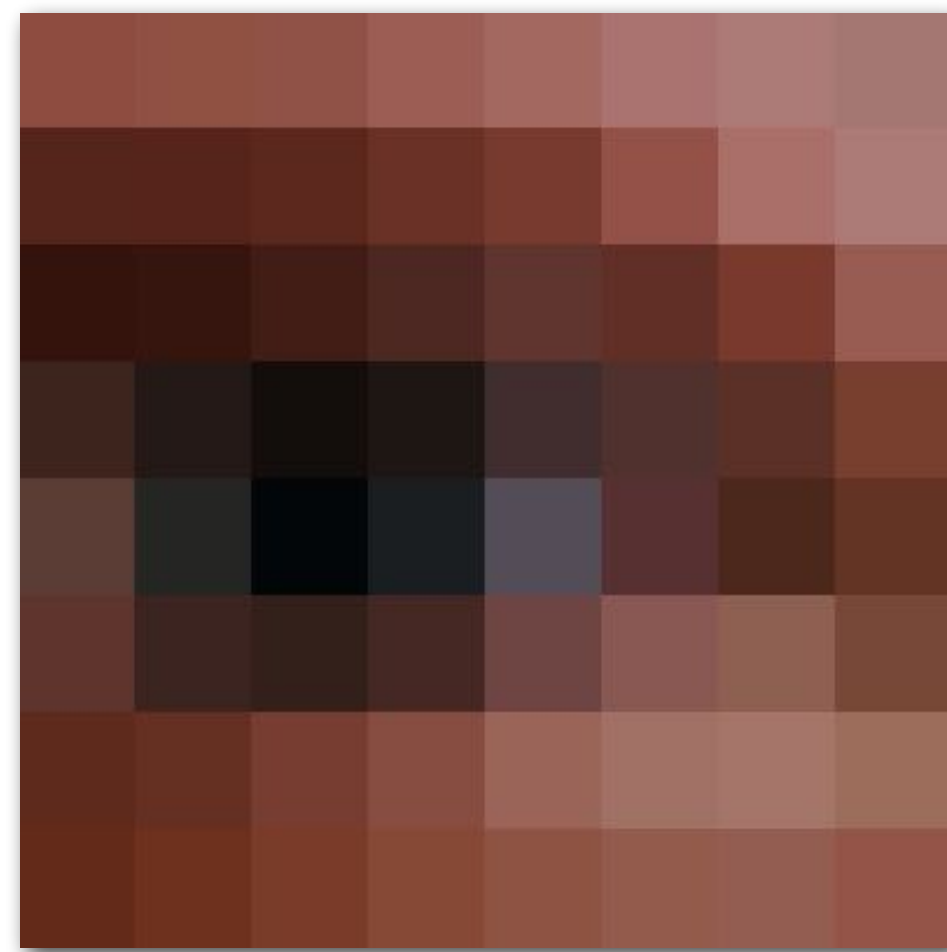
■ Exploiting redundancy:

- **Intra-frame redundancy: value of pixels in neighboring regions of a frame are good predictor of values for other pixels in the frame (spatial redundancy)**
- **Inter-frame redundancy: pixels from nearby frames in time are a good predictor for the current frame's pixels (temporal redundancy)**

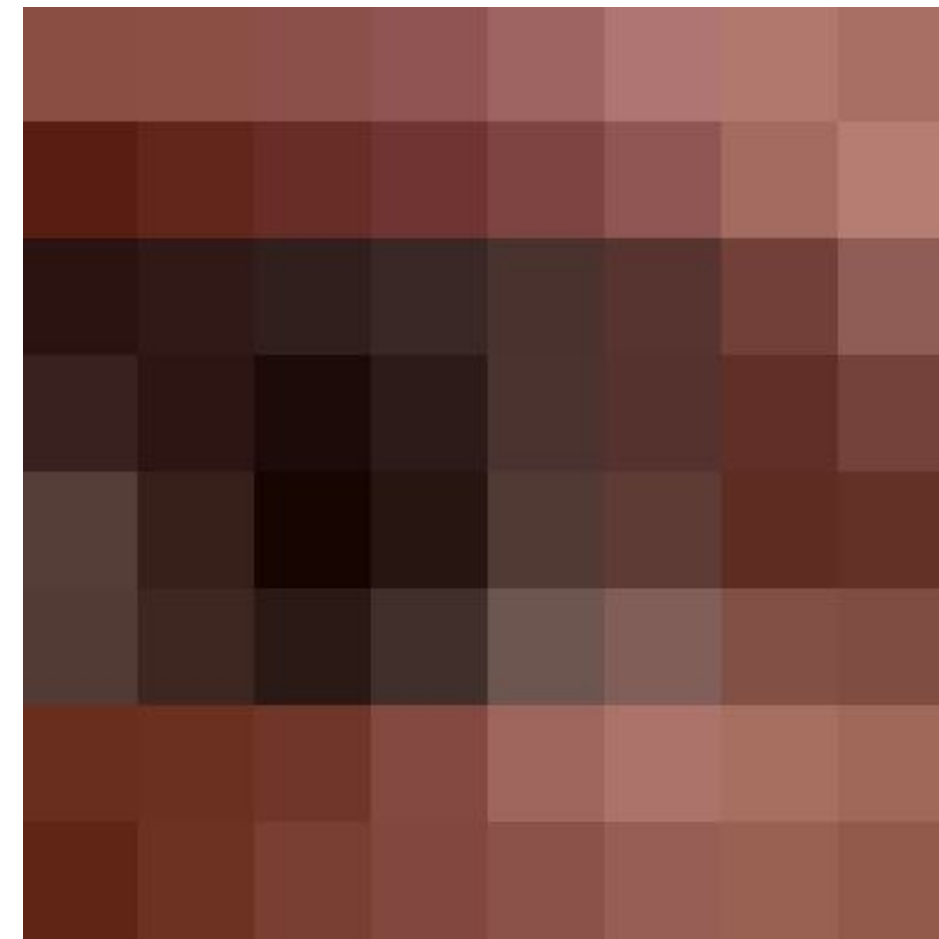
Motion vector visualization



Residual: difference between predicted image and original image



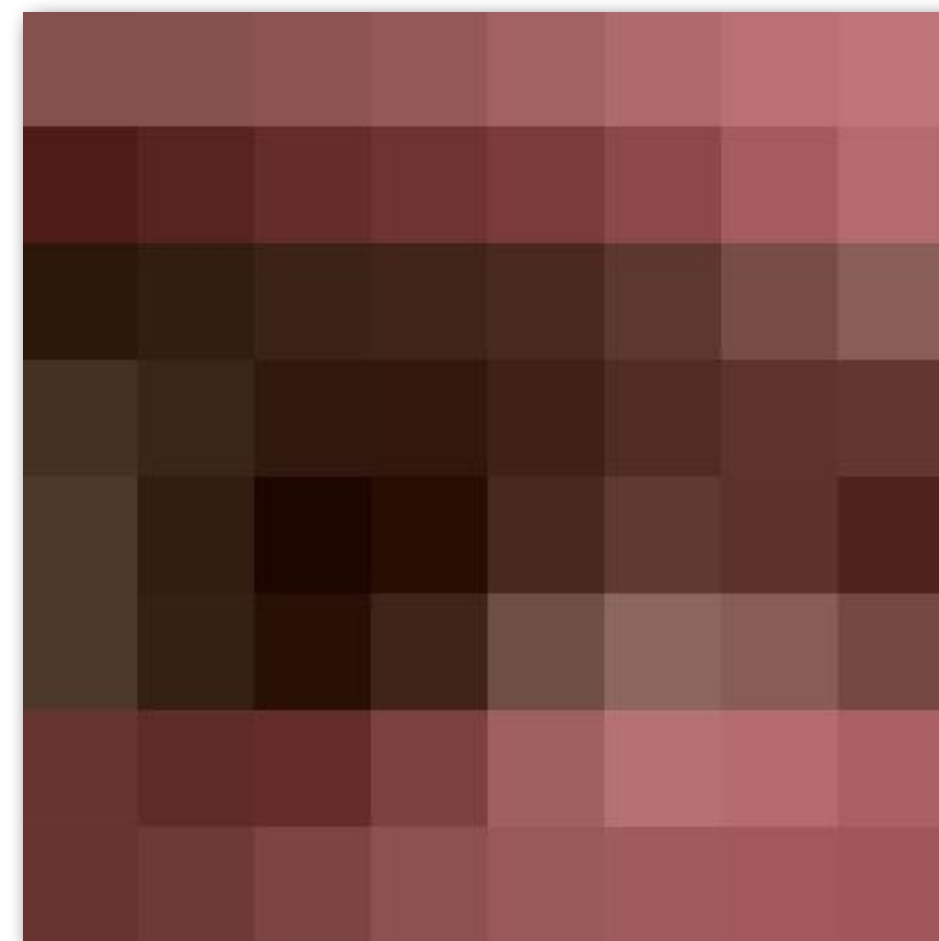
Original pixels



Predicted pixels
(Prediction A)



Residual
(amplified for visualization)



Predicted pixels
(Prediction B)



Residual
(amplified for visualization)

In video compression schemes, the residual image is compressed using lossy compression techniques like those described in the earlier part of this lecture. Better predictions lead to smaller and more compressible residuals!

Video compression overview

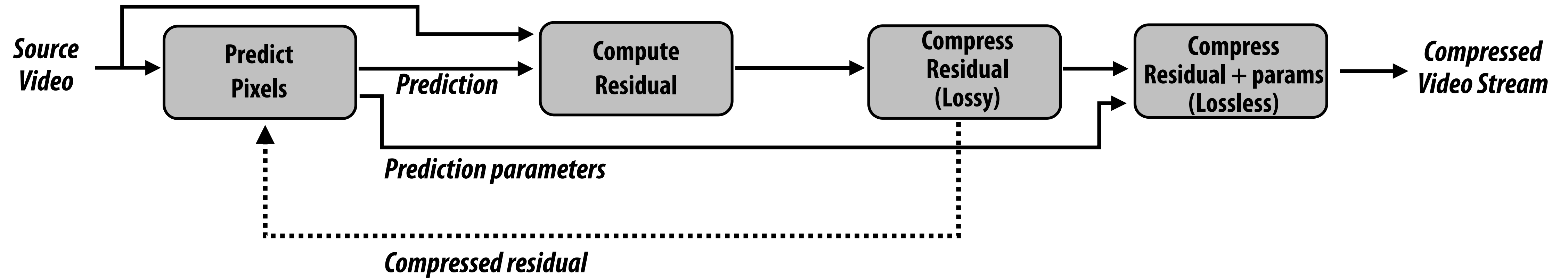


Image processing basics

Example image processing operations

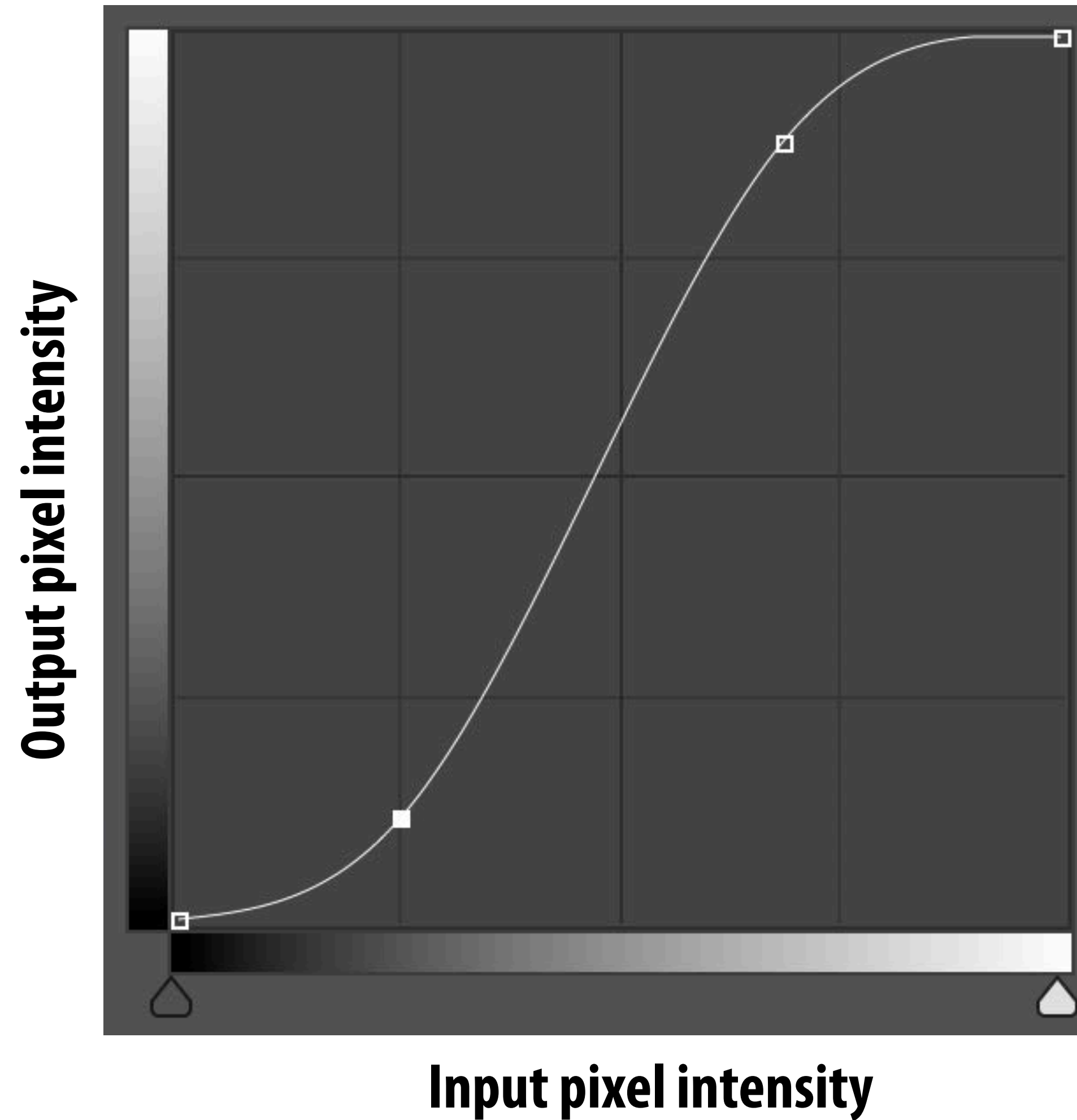


Increase contrast

Increasing contrast with “S curve”

Per-pixel operation:

$$\text{output}(x,y) = f(\text{input}(x,y))$$

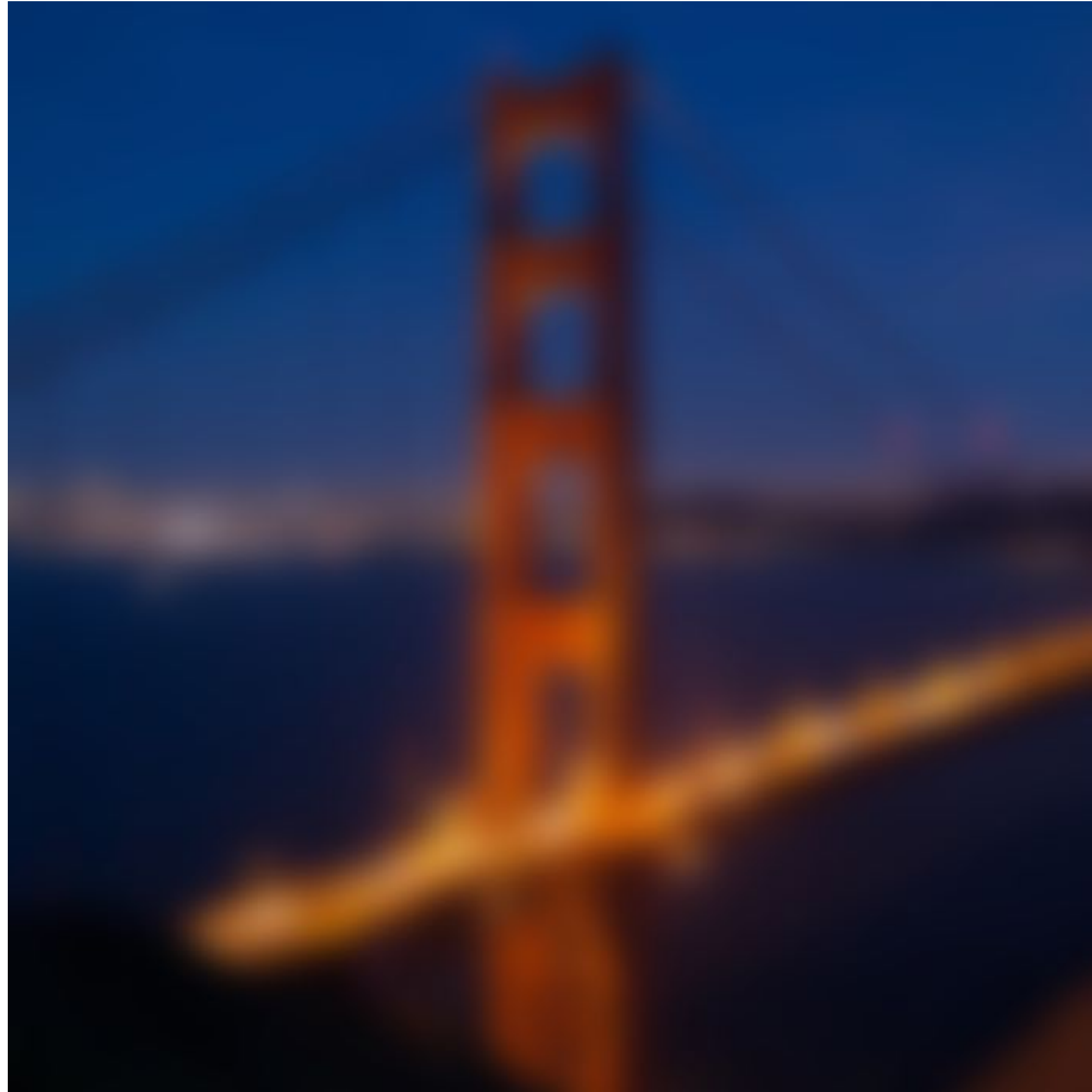


Example image processing operations



Image Invert:
 $out(x,y) = 1 - in(x,y)$

Example image processing operations



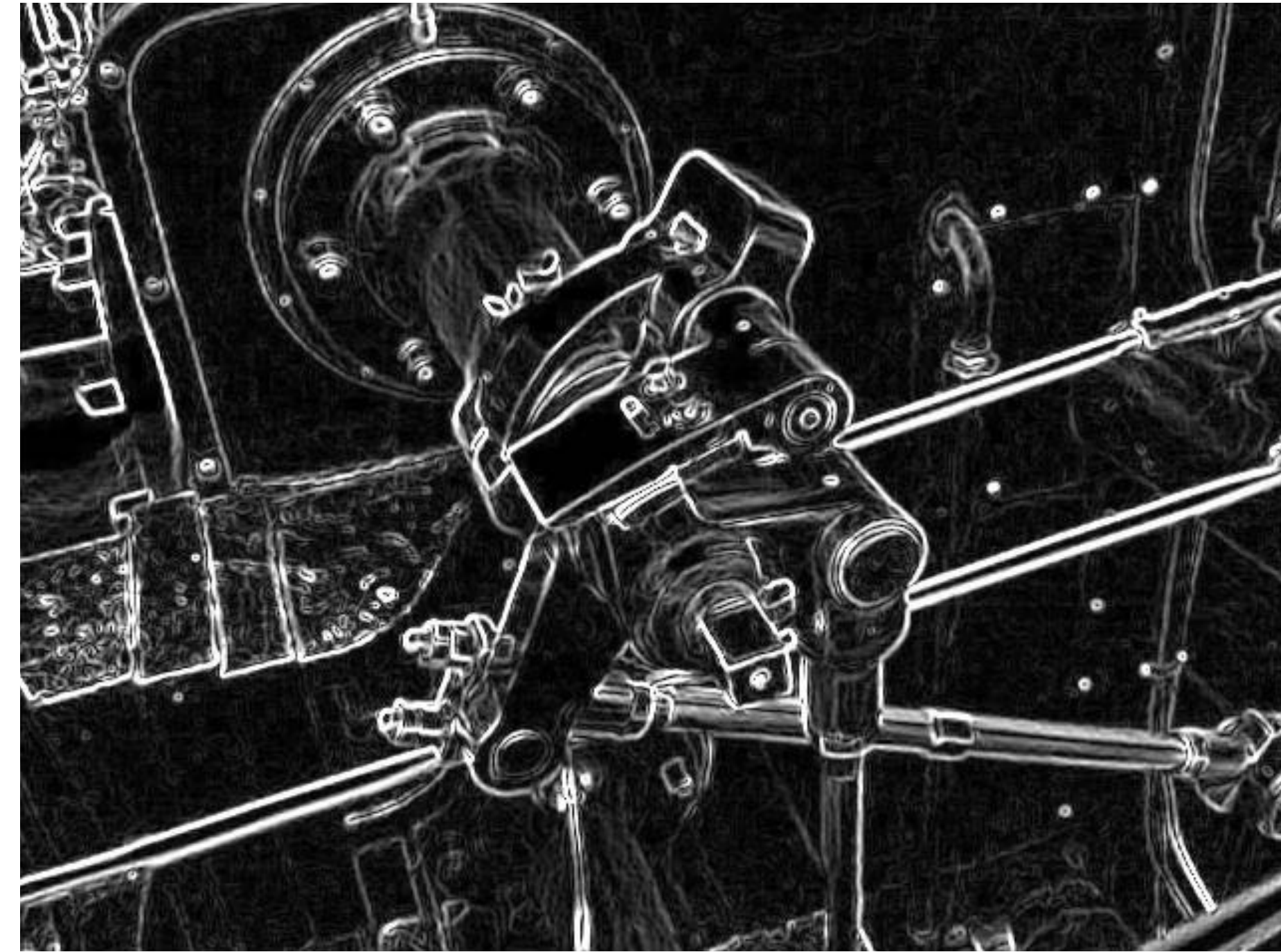
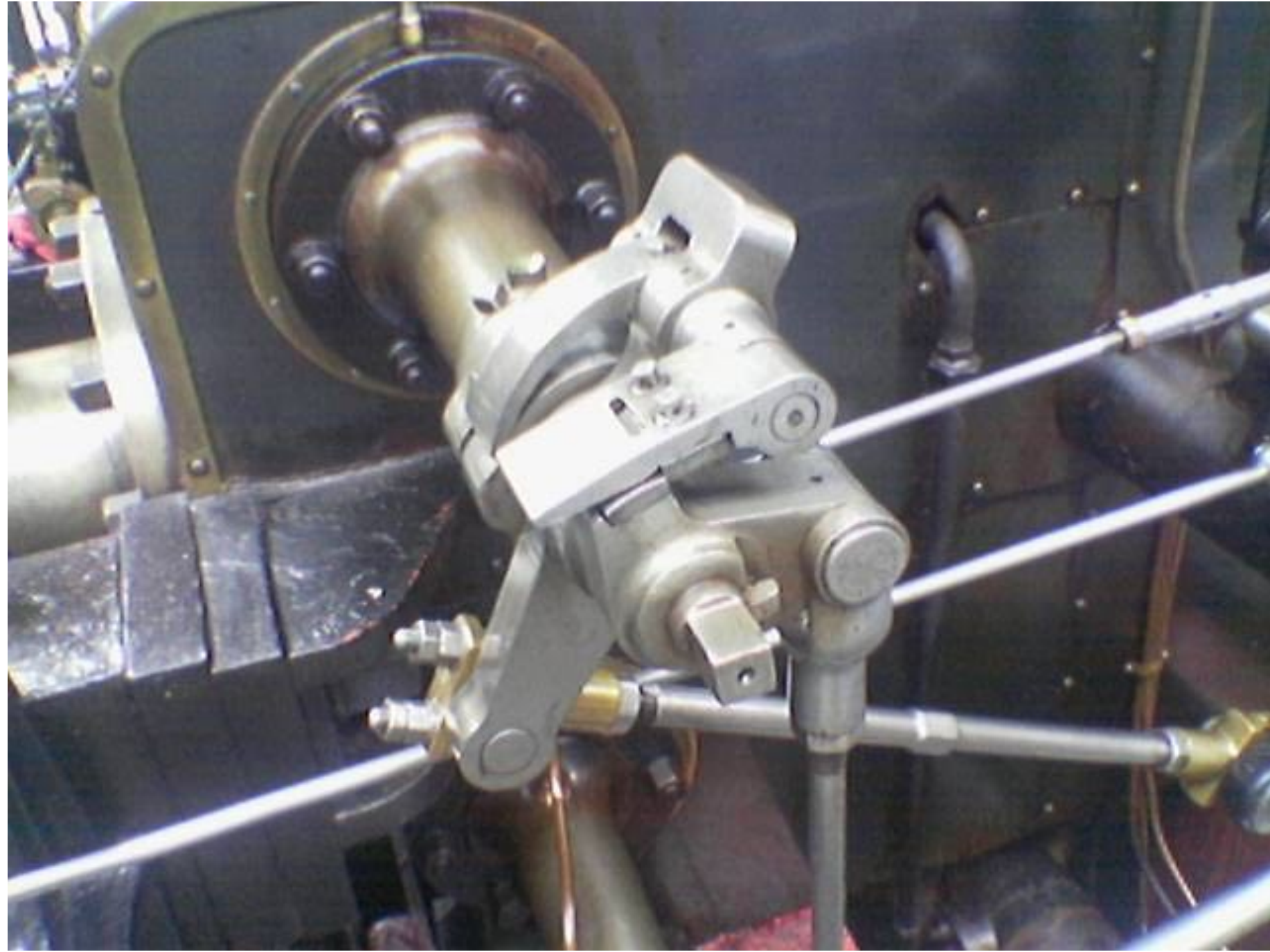
Blur

Example image processing operations



Sharpen

Edge detection



A “smarter” blur (doesn't blur over edges)



Review: convolution

$$(f * g)(x) = \int_{-\infty}^{\infty} f(y)g(x - y)dy$$

output signal

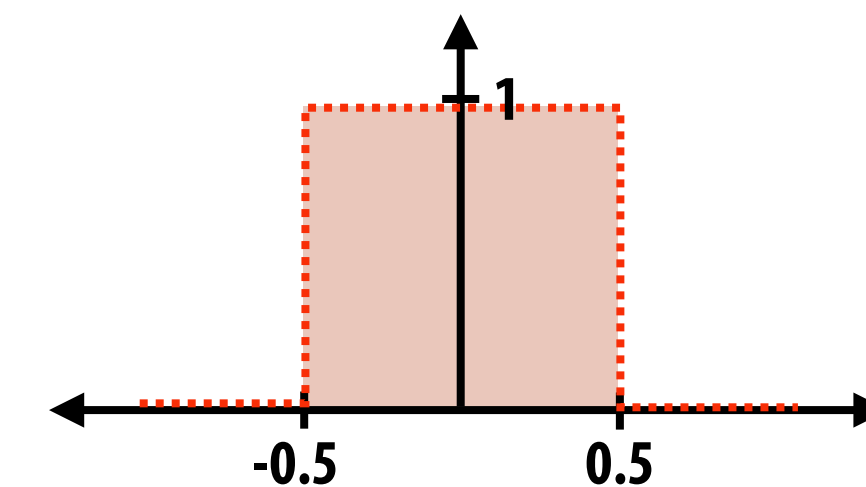
filter

input signal
(e.g. the input image)

It may be helpful to consider the effect of convolution with the simple unit-area “box” function:

$$f(x) = \begin{cases} 1 & |x| \leq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

$$(f * g)(x) = \int_{-0.5}^{0.5} g(x - y)dy$$



$f * g$ is a “blurred” version of g where the output at x is the average value of the input between $x-0.5$ to $x+0.5$

Discrete 2D convolution

$$(f * g)(x, y) = \sum_{i, j = -\infty}^{\infty} f(i, j) I(x - i, y - j)$$

output image filter input image

Consider $f(i, j)$ that is nonzero only when: $-1 \leq i, j \leq 1$

Then:

$$(f * I)(x, y) = \sum_{i, j = -1}^1 f(i, j) I(x - i, y - j)$$

And we can represent $f(i, j)$ as a 3x3 matrix of values where:

$$f(i, j) = \mathbf{F}_{i, j} \quad \text{(often called: "filter weights", "filter kernel")}$$

Simple 3x3 box blur

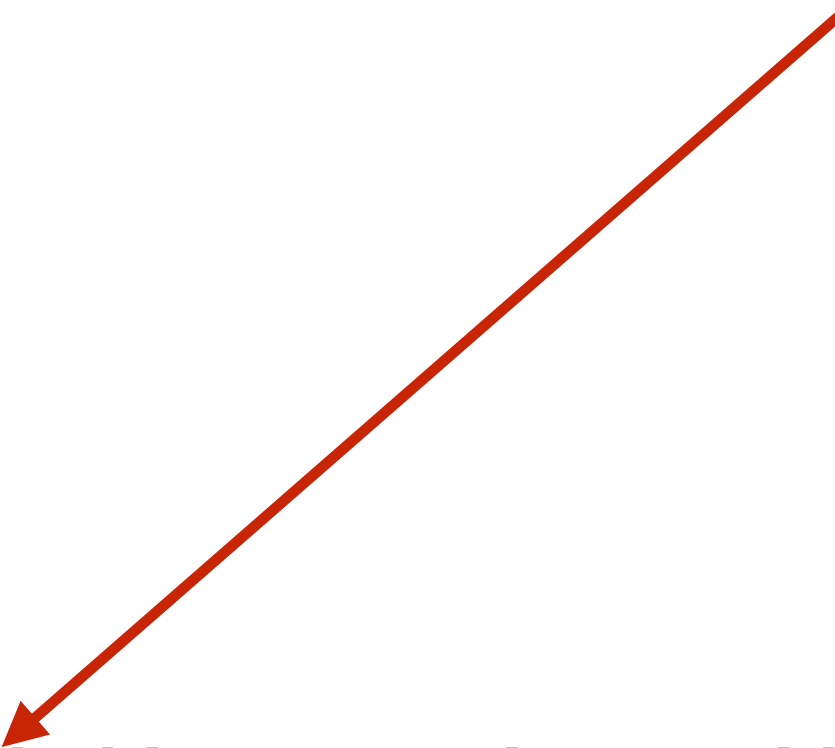
```
float input[(WIDTH+2) * (HEIGHT+2)];  
float output[WIDTH * HEIGHT];
```

```
float weights[] = {1./9, 1./9, 1./9,  
                  1./9, 1./9, 1./9,  
                  1./9, 1./9, 1./9};
```

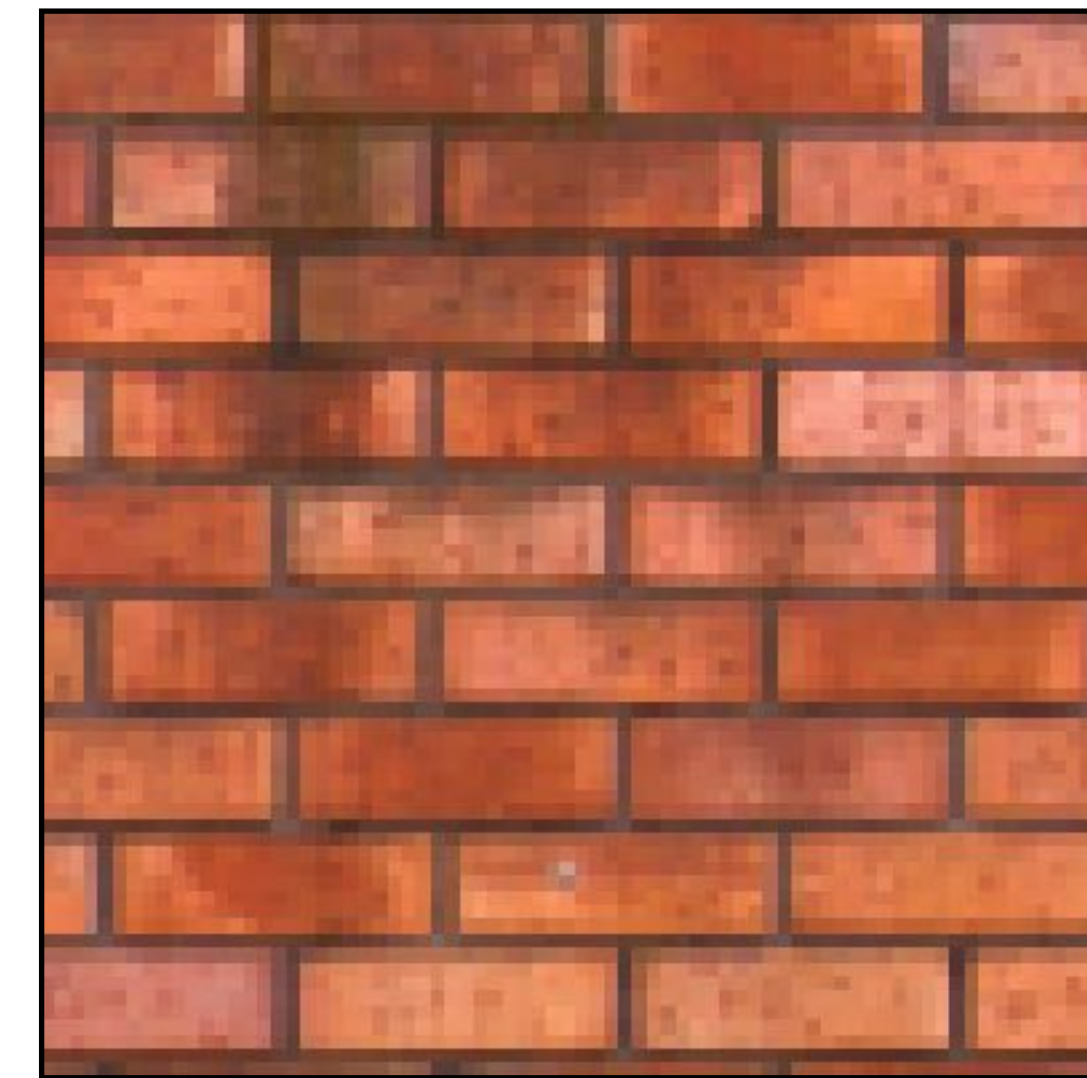
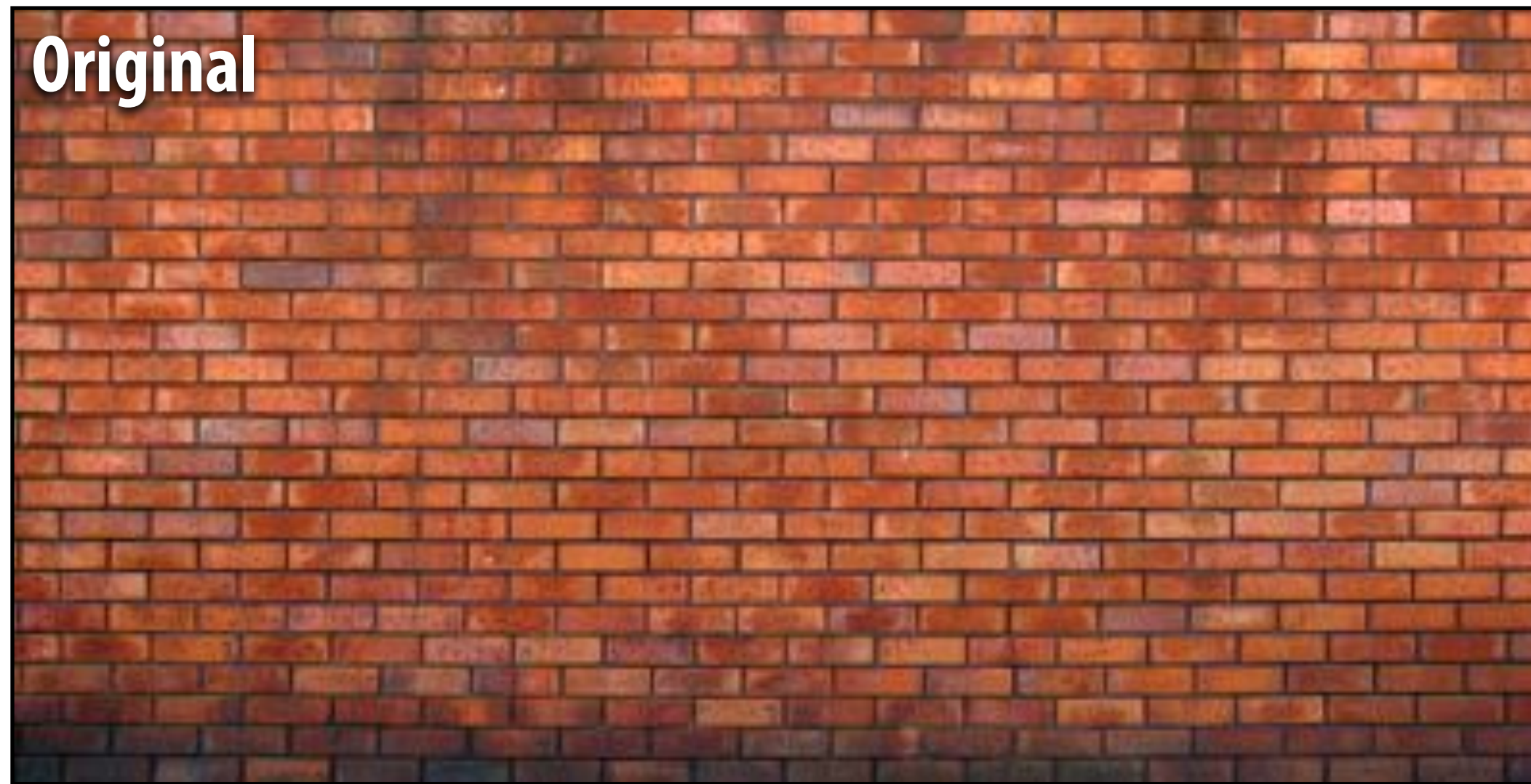
```
for (int j=0; j<HEIGHT; j++) {  
    for (int i=0; i<WIDTH; i++) {  
        float tmp = 0.f;  
        for (int jj=0; jj<3; jj++)  
            for (int ii=0; ii<3; ii++)  
                tmp += input[(j+jj)*(WIDTH+2) + (i+ii)] * weights[jj*3 + ii];  
        output[j*WIDTH + i] = tmp;  
    }  
}
```



**For now: ignore boundary pixels and
assume output image is smaller than
input (makes convolution loop bounds
much simpler to write)**



7x7 box blur



Gaussian blur

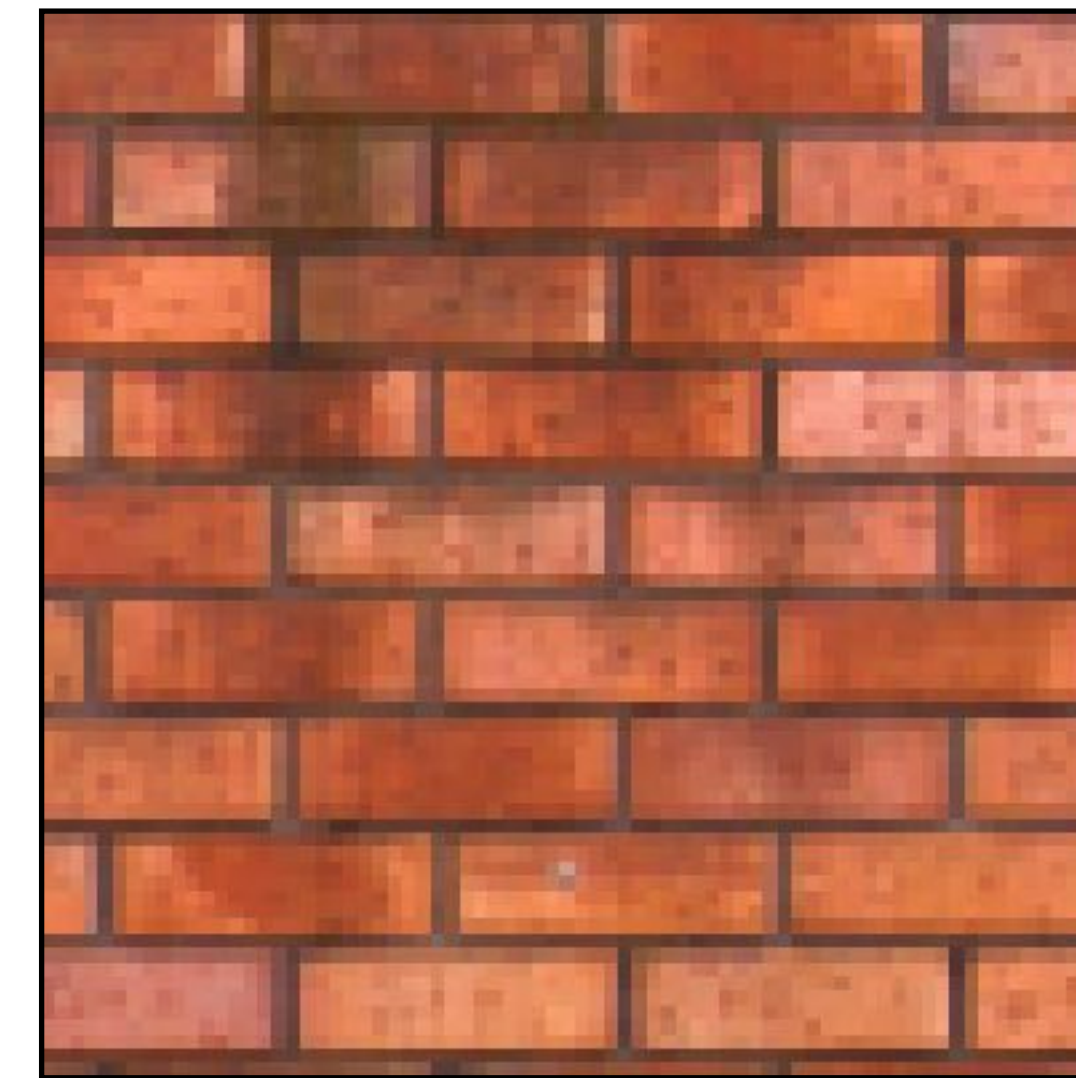
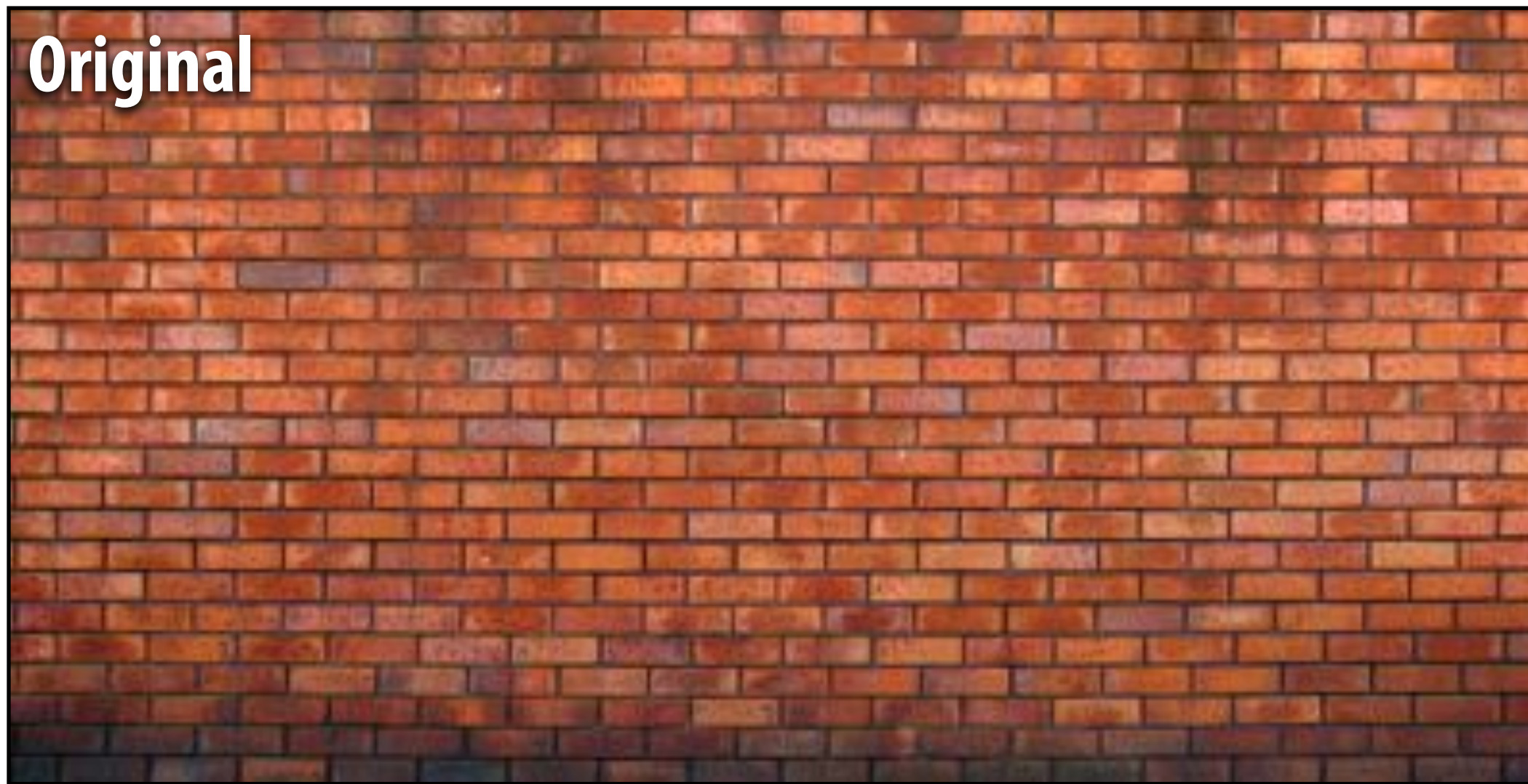
- Obtain filter coefficients by sampling 2D Gaussian function

$$f(i, j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2 + j^2}{2\sigma^2}}$$

- Produces weighted sum of neighboring pixels (contribution falls off with distance)
 - In practice: truncate filter beyond certain distance for efficiency

$$\begin{bmatrix} .075 & .124 & .075 \\ .124 & .204 & .124 \\ .075 & .124 & .075 \end{bmatrix}$$

7x7 gaussian blur

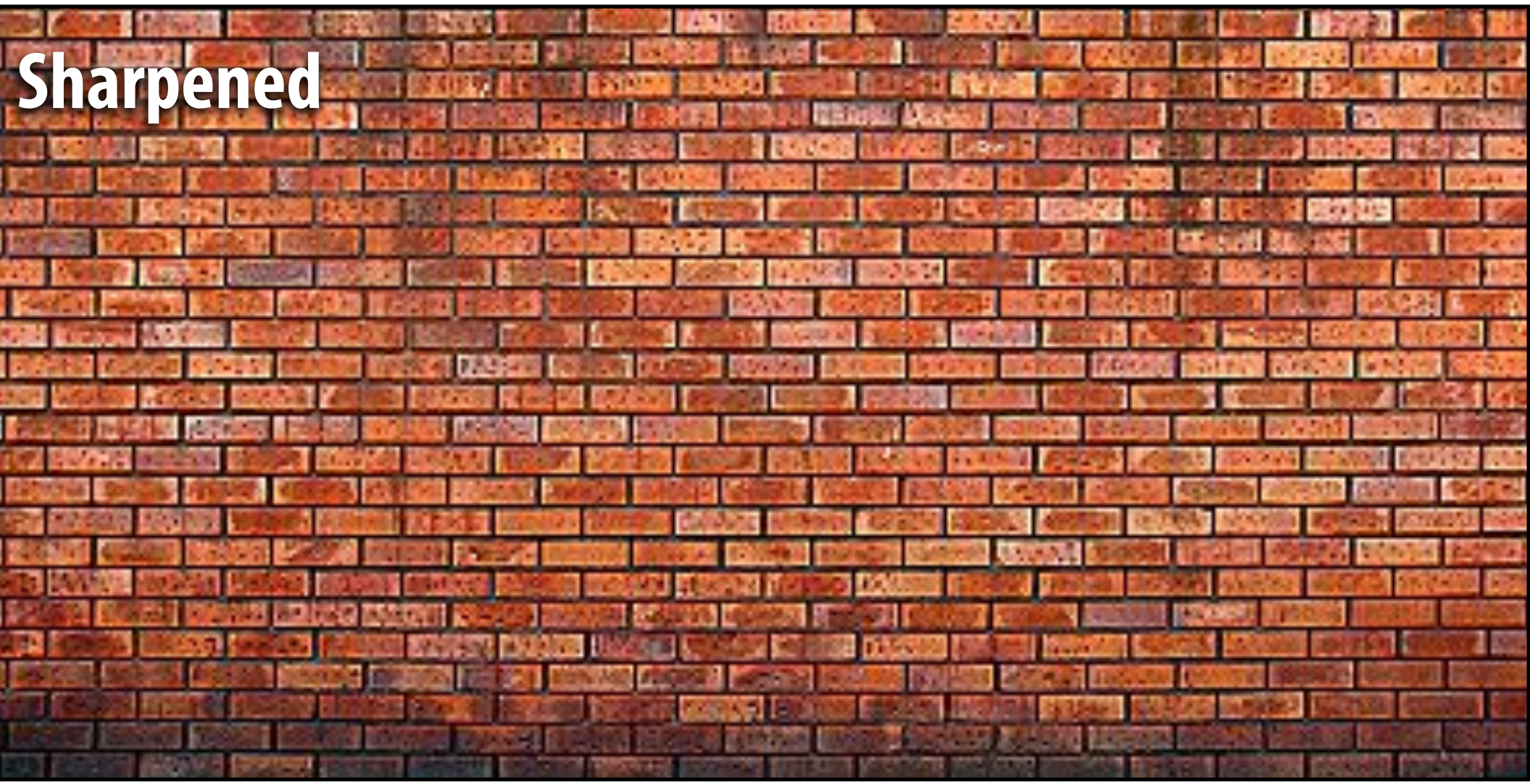
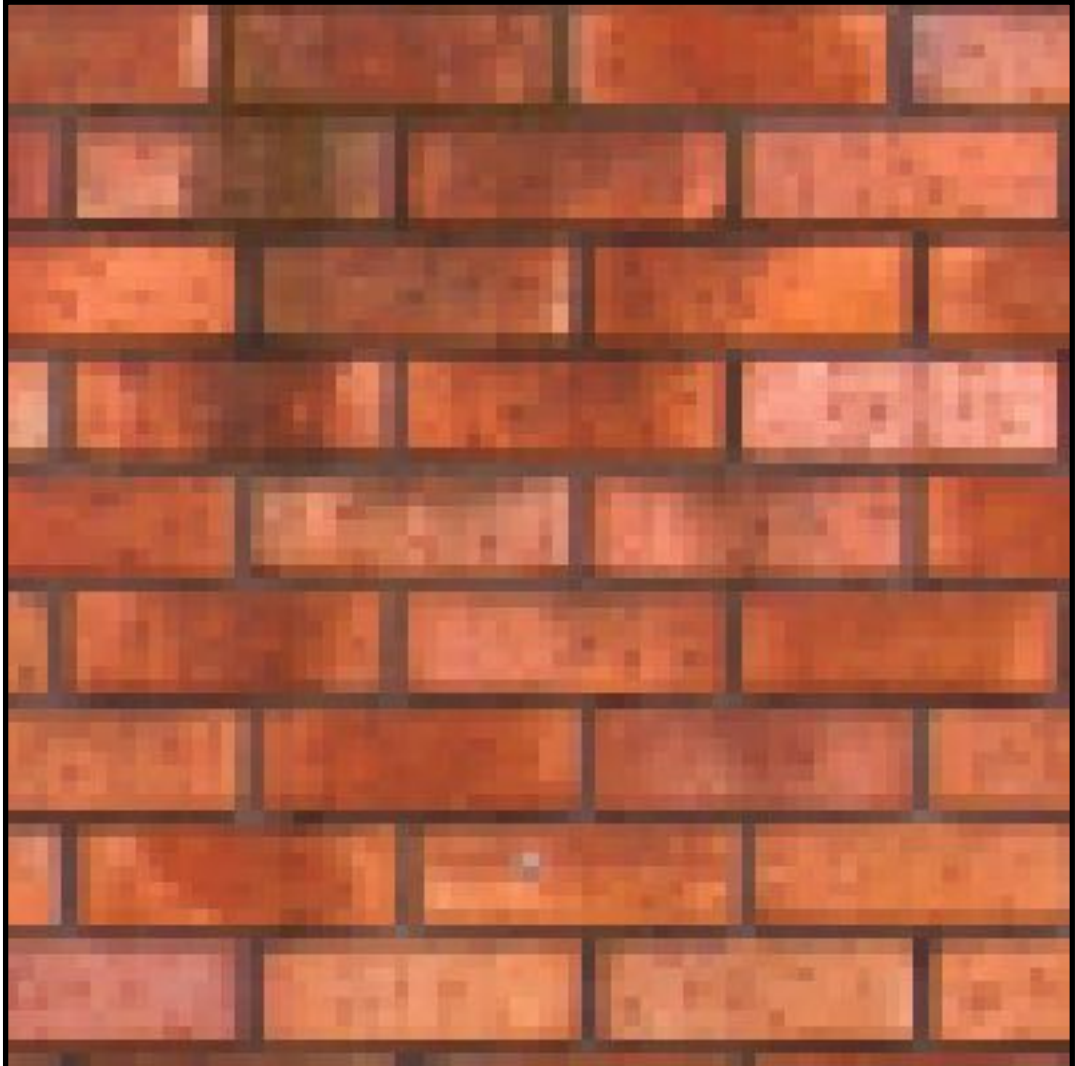
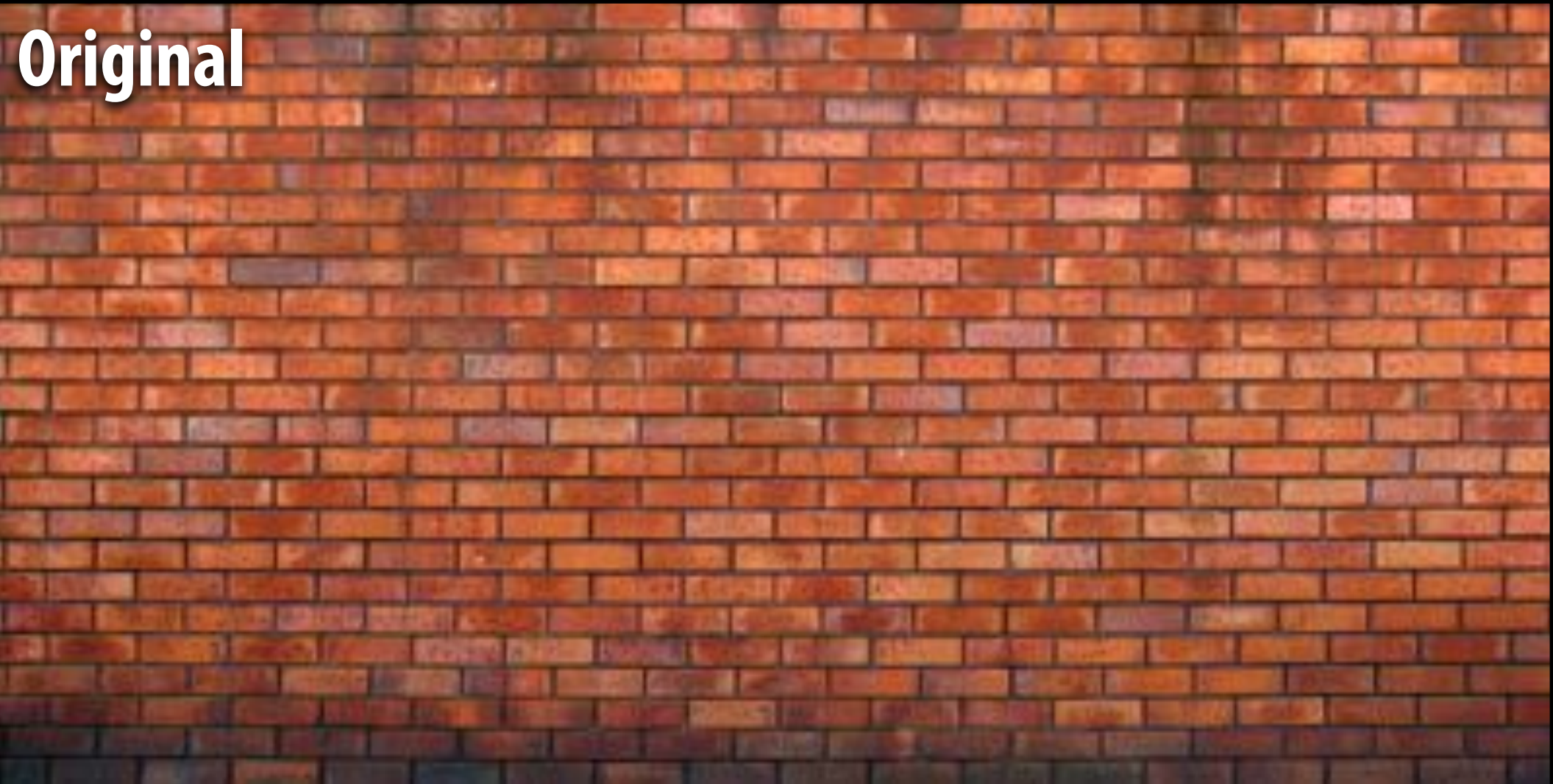


What does convolution with this filter do?

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

Sharpens image!

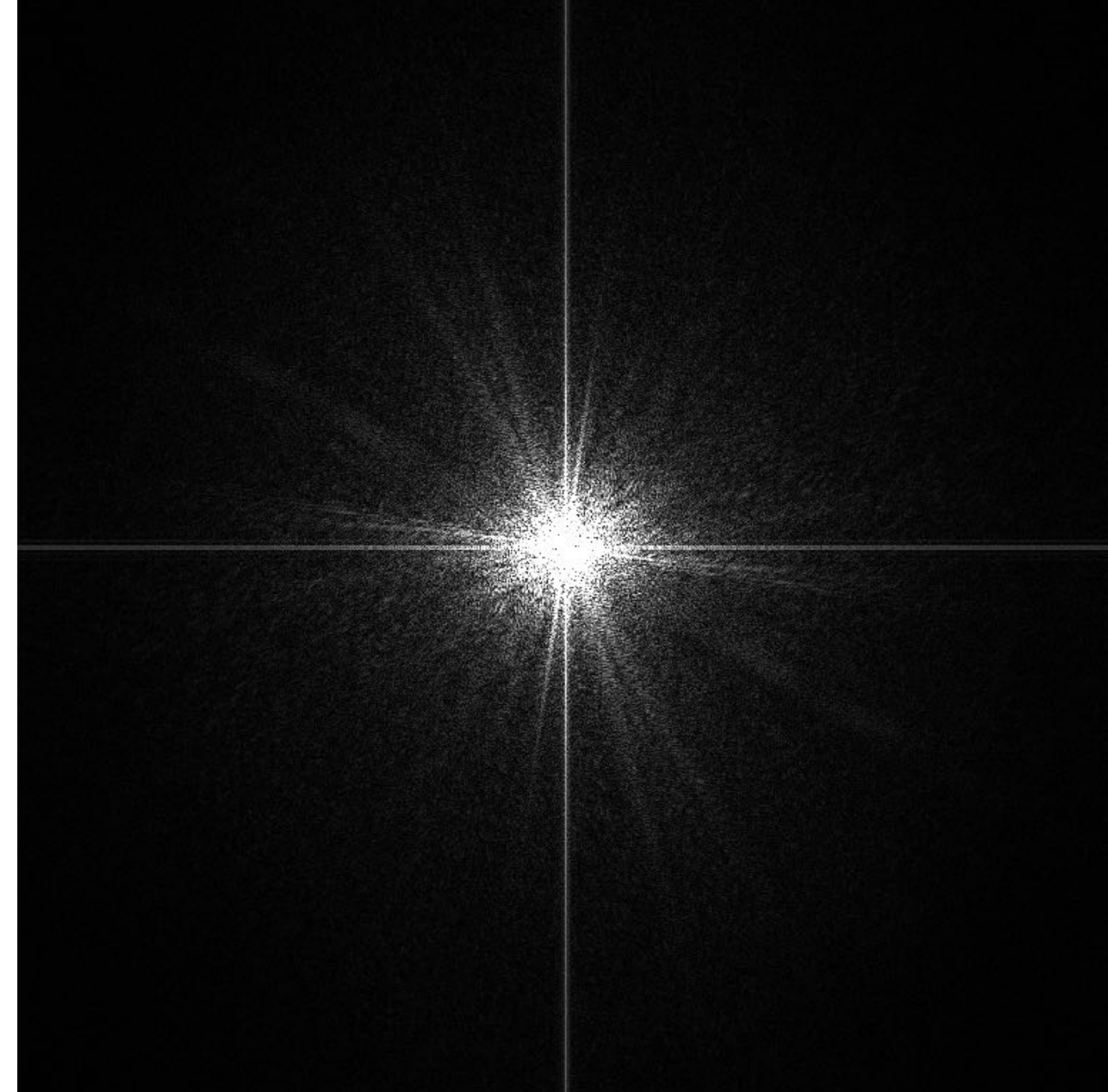
3x3 sharpen filter



Recall: blurring is removing high frequency content



Spatial domain result

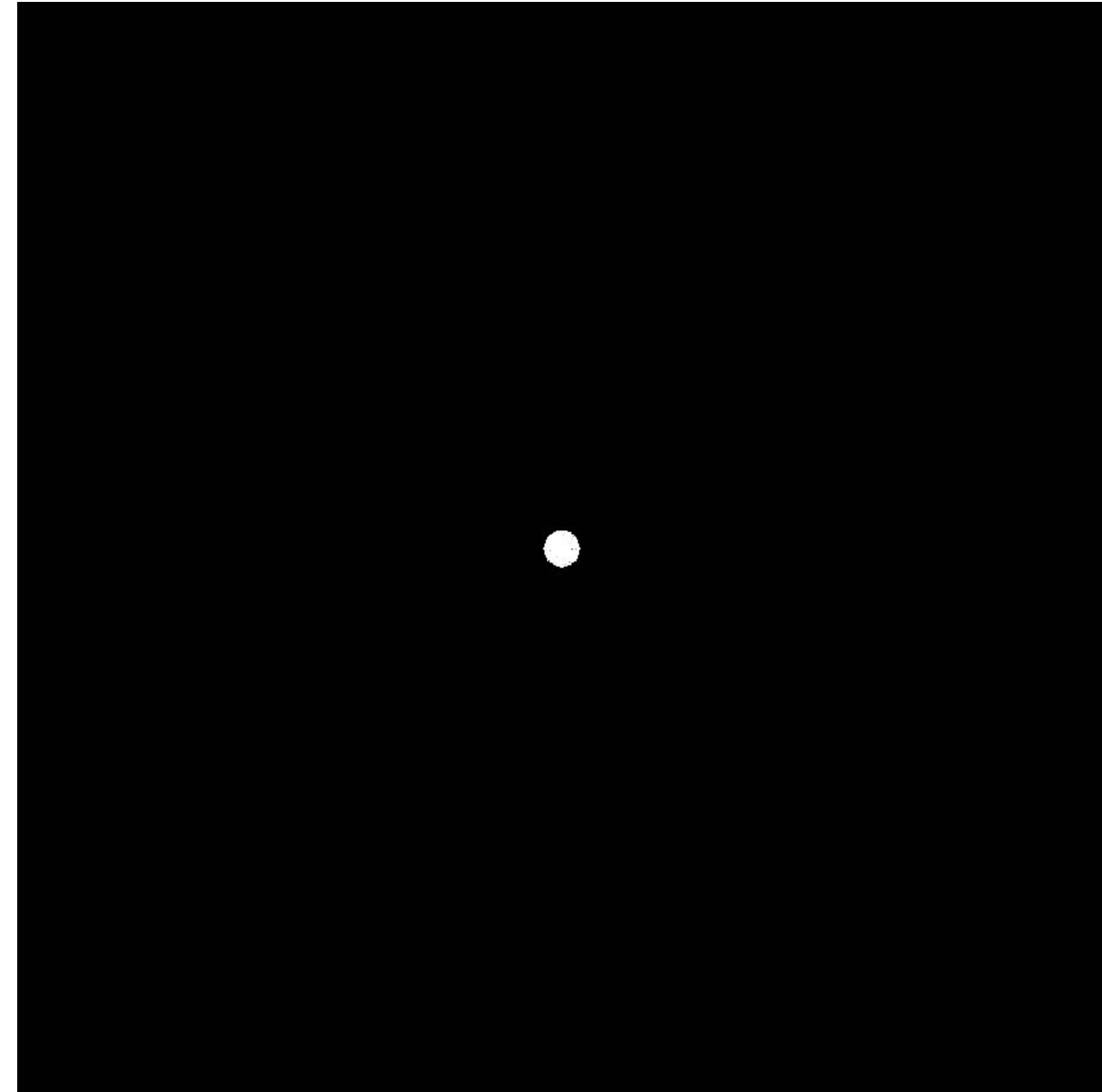


Spectrum

Recall: blurring is removing high frequency content



Spatial domain result



Spectrum (after low-pass filter)
All frequencies above cutoff have 0 magnitude

Sharpening is adding high frequencies

- Let I be the original image
- High frequencies in image $I = I - \text{blur}(I)$
- Sharpened image = $I + (I - \text{blur}(I))$



“Add high frequency content”

Original image (I)

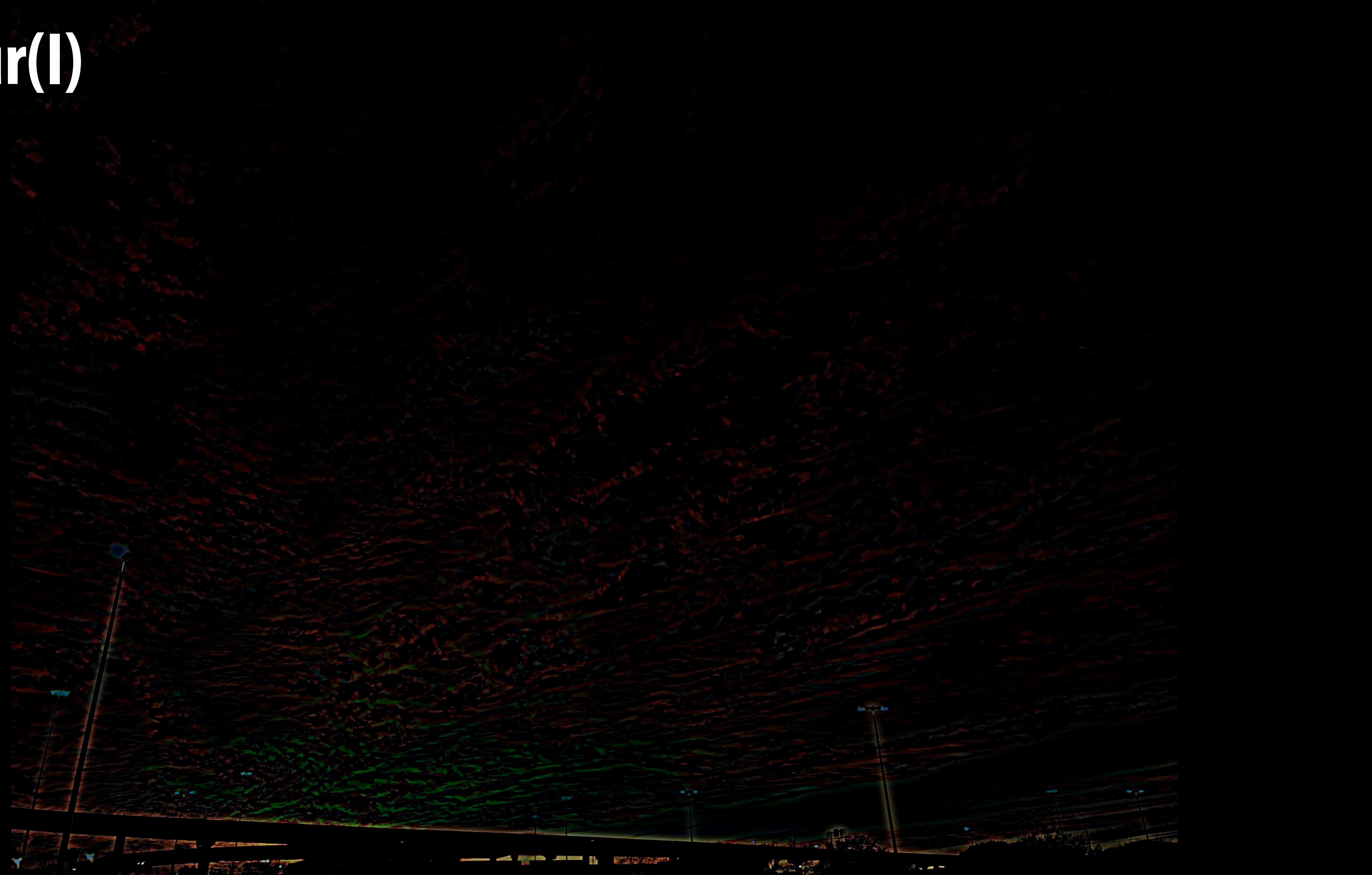


Image credit:
Kayvon's parents

Blur(I)



I - blur(I)



$I + (I - \text{blur}(I))$



What does convolution with these filters do?

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

**Extracts horizontal
gradients**

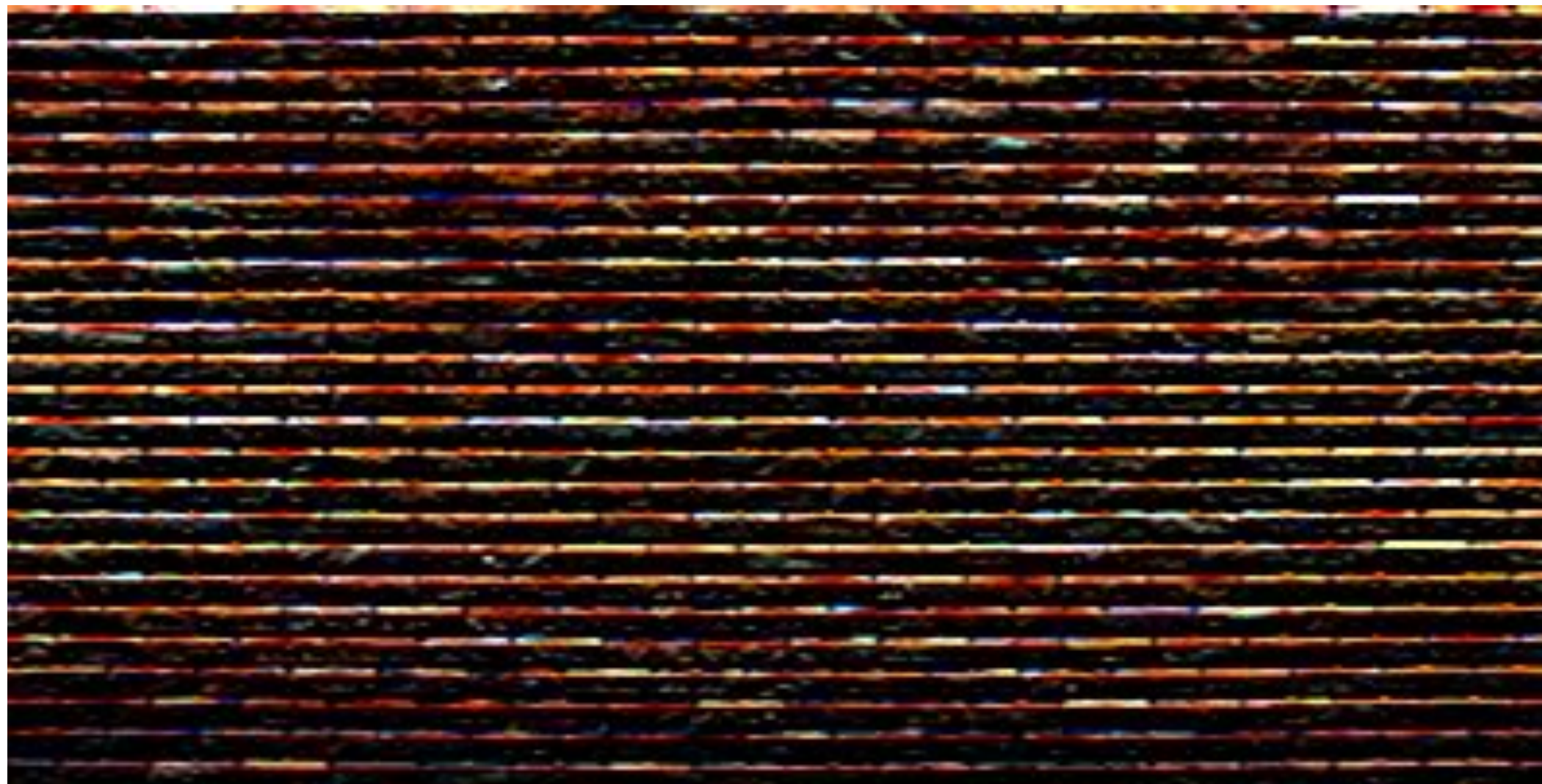
$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

**Extracts vertical
gradients**

Gradient detection filters



Horizontal gradients



Vertical gradients

Note: you can think of a filter as a “detector” of a pattern, and the magnitude of a pixel in the output image as the “response” of the filter to the region surrounding each pixel in the input image (this is a common interpretation in computer vision)

Sobel edge detection

- Compute gradient response images

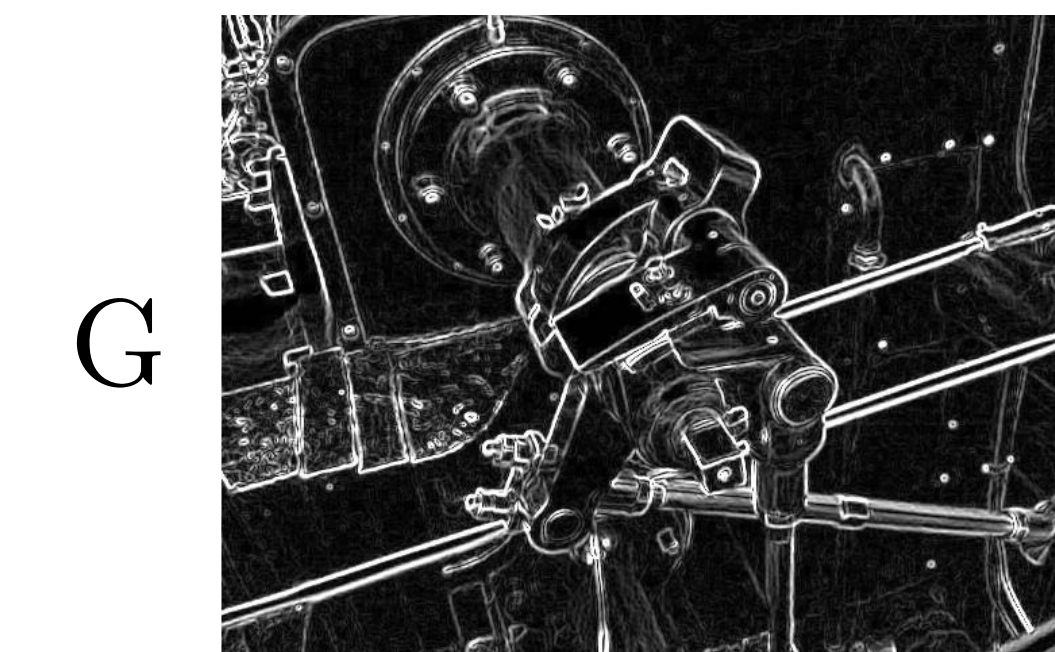
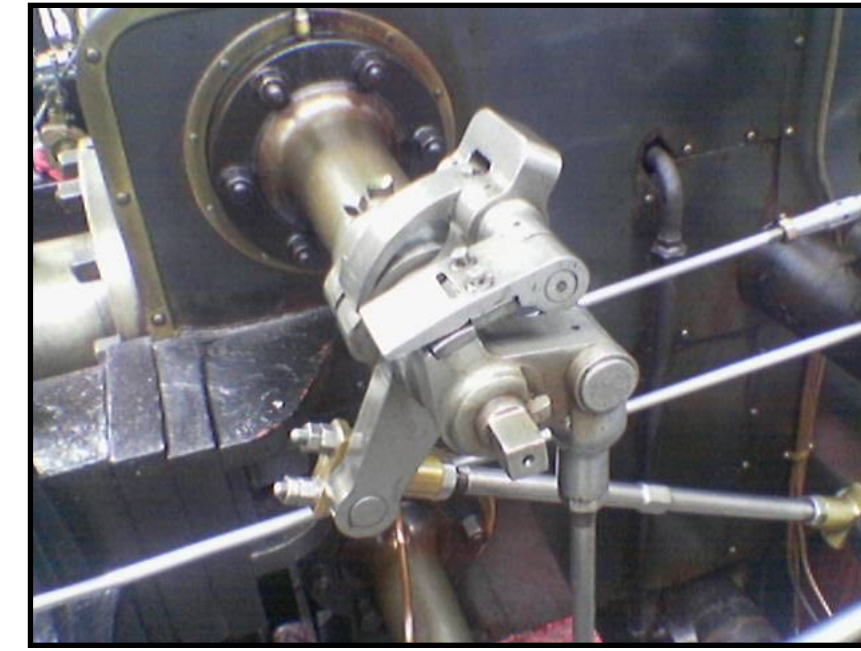
$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * I$$

- Find pixels with large gradients

$$G = \sqrt{G_x^2 + G_y^2}$$

Pixel-wise operation on images



Cost of convolution with N x N filter?

```
float input[(WIDTH+2) * (HEIGHT+2)];  
float output[WIDTH * HEIGHT];
```

```
float weights[] = {1./9, 1./9, 1./9,  
                  1./9, 1./9, 1./9,  
                  1./9, 1./9, 1./9};
```

```
for (int j=0; j<HEIGHT; j++) {  
    for (int i=0; i<WIDTH; i++) {  
        float tmp = 0.f;  
        for (int jj=0; jj<3; jj++)  
            for (int ii=0; ii<3; ii++)  
                tmp += input[(j+jj)*(WIDTH+2) + (i+ii)] * weights[jj*3 + ii];  
        output[j*WIDTH + i] = tmp;  
    }  
}
```

In this 3x3 box blur example:

Total work per image = 9 x WIDTH x HEIGHT

For N x N filter: N^2 x WIDTH x HEIGHT

Separable filter

- A filter is separable if can be written as the outer product of two other filters. Example: a 2D box blur

$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} * \frac{1}{3} [1 \quad 1 \quad 1]$$

- Exercise: write 2D gaussian and vertical/horizontal gradient detection filters as product of 1D filters (they are separable!)
- Key property: 2D convolution with separable filter can be written as two 1D convolutions!

Implementation of 2D box blur via two 1D convolutions

```
int WIDTH = 1024
int HEIGHT = 1024;
float input[(WIDTH+2) * (HEIGHT+2)];
float tmp_buf[WIDTH * (HEIGHT+2)];
float output[WIDTH * HEIGHT];

float weights[] = {1./3, 1./3, 1./3};

for (int j=0; j<(HEIGHT+2); j++)
  for (int i=0; i<WIDTH; i++) {
    float tmp = 0.f;
    for (int ii=0; ii<3; ii++)
      tmp += input[j*(WIDTH+2) + i+ii] * weights[ii];
    tmp_buf[j*WIDTH + i] = tmp;
  }

for (int j=0; j<HEIGHT; j++) {
  for (int i=0; i<WIDTH; i++) {
    float tmp = 0.f;
    for (int jj=0; jj<3; jj++)
      tmp += tmp_buf[(j+jj)*WIDTH + i] * weights[jj];
    output[j*WIDTH + i] = tmp;
  }
}
```

**Total work per image for NxN filter:
2N x WIDTH x HEIGHT**

Preview of next time...

Bilateral filter

Do not smooth over hard edges, but smooth when there are not hard edges.

Original



After bilateral filter



Example use of bilateral filter: removing noise while preserving image edges

Bilateral filter

Original

After bilateral filter



Example use of bilateral filter: removing noise while preserving image edges

Summary

■ Last two lectures: representing images

- Choice of color space (different representations of color)
- Store values in perceptual space (non-linear in energy)
- JPEG image compression (tolerate loss due to approximate representation of high frequency components)

■ Basic image processing operations

- Per-pixel operations $out(x,y) = f(in(x,y))$ (e.g., contrast enhancement)
- Image filtering via convolution (e.g., blur, sharpen, simple edge-detection)
- Non-linear, data-dependent filters (avoid blurring over strong edges, etc.)