Lecture 3:

The Camera Image Processing Pipeline

(part 2: tone mapping and autofocus)

Visual Computing Systems

Stanford CS348K, Spring 2021

Previous class and today…

The pixels you see on screen are quite different than the values recorded by the sensor in a modern digital camera.

Computation is now a fundamental aspect of producing high-quality pictures.

Sensor output ("RAW") → **Computation** →

Beautiful image that impresses your friends on Instagram

Summary: simplified image processing pipeline

- Correct pixel defects
- Align and merge (to create high signal to noise ration RAW image)
- Correct for sensor bias (using measurements of optically black pixels)
- Vignetting compensation
- White balance

(10-12 bits per pixel)
1 intensity value per pixel
Pixel values linear in energy

- Demosaic
- Denoise

3x10 bits per pixel
RGB intensity per pixel
Pixel values linear in energy

- Gamma Correction (non-linear mapping)

- Local tone mapping
- Final adjustments sharpen, fix chromatic aberrations, hue adjust, etc.

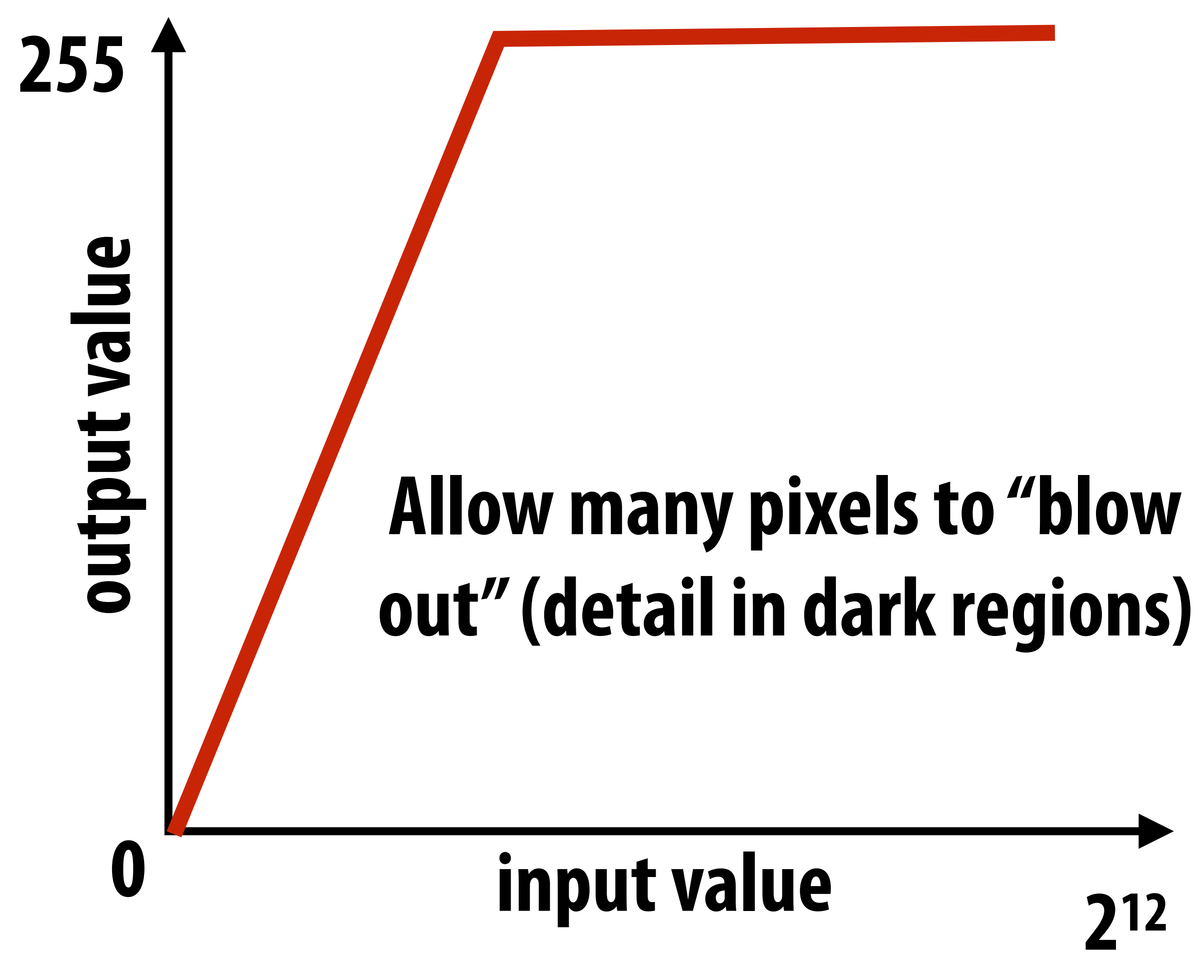
3x8-bits per pixel
Pixel values perceptually linear

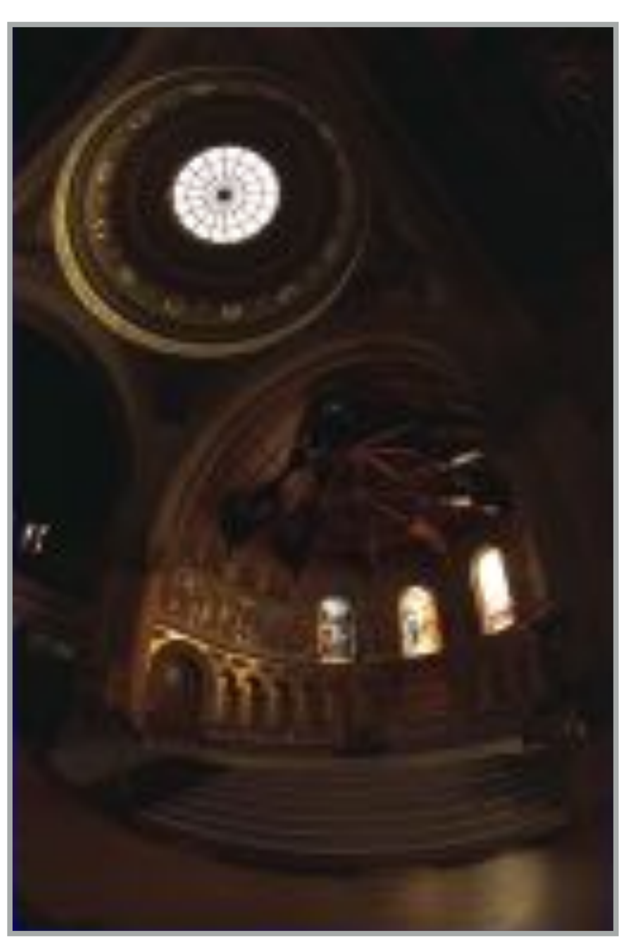
Today

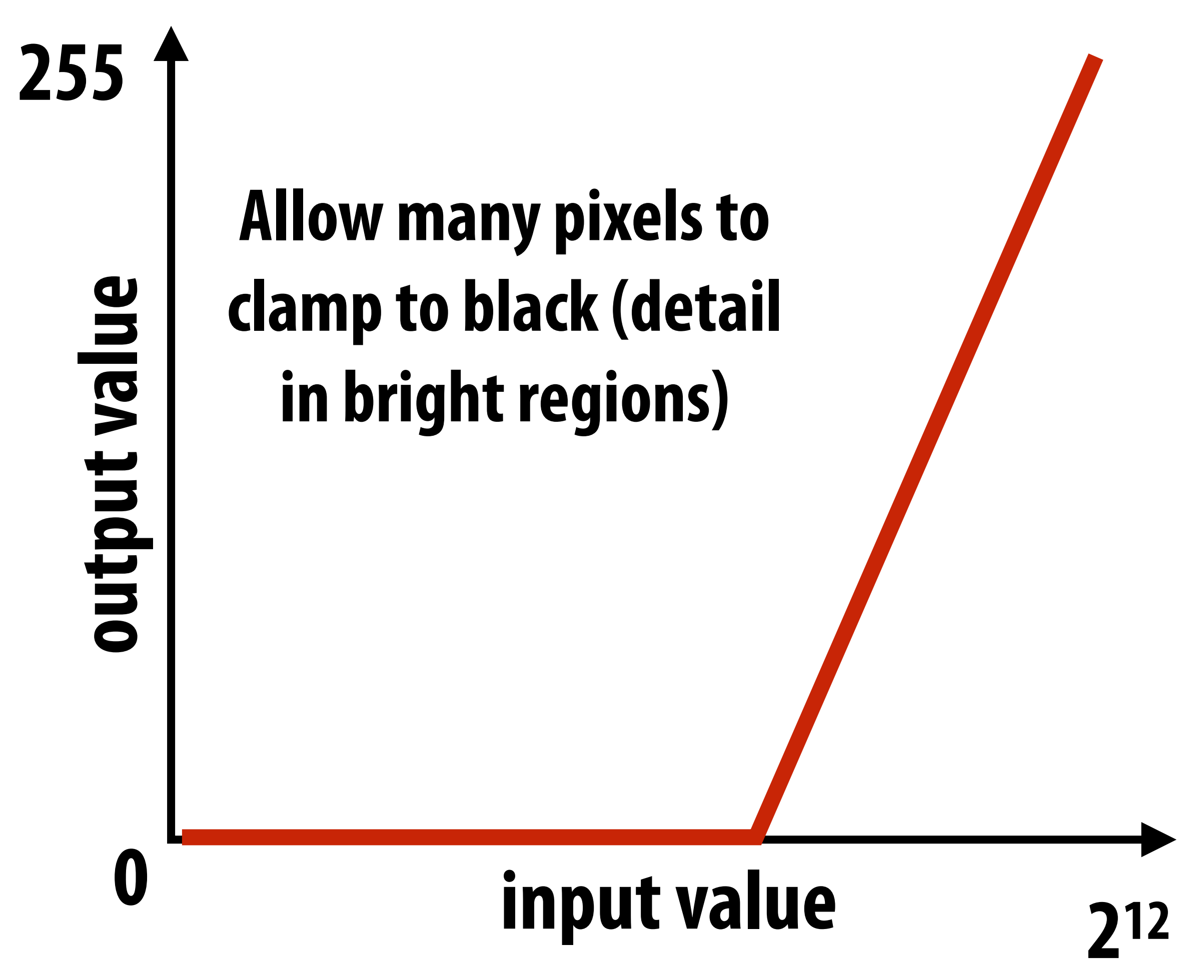
Auto Exposure and Tone Mapping

Global tone mapping

- Measured image values: 10-12 bits / pixel, but common image formats (8-bits/ pixel)
- How to convert 12 bit number to 8 bit number?



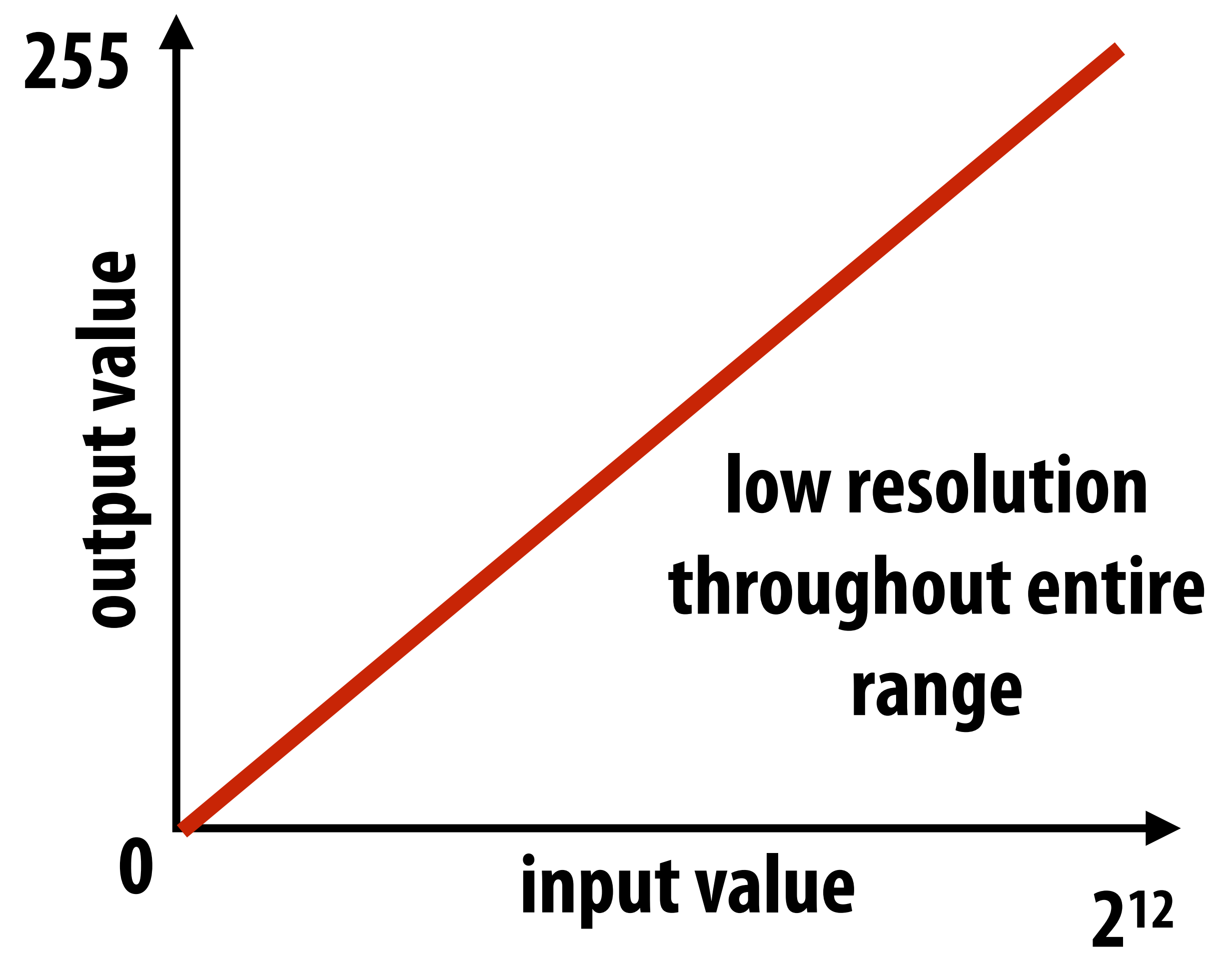


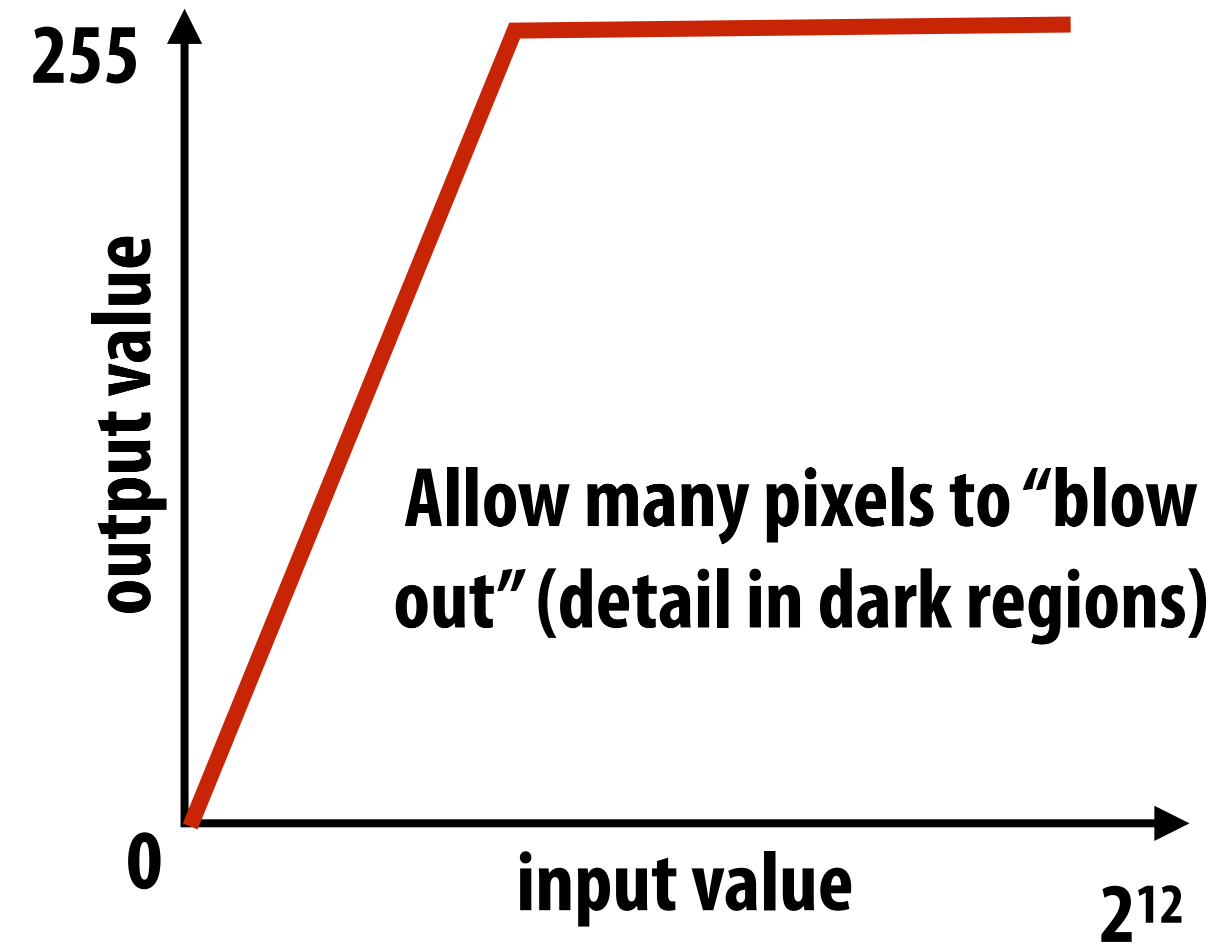


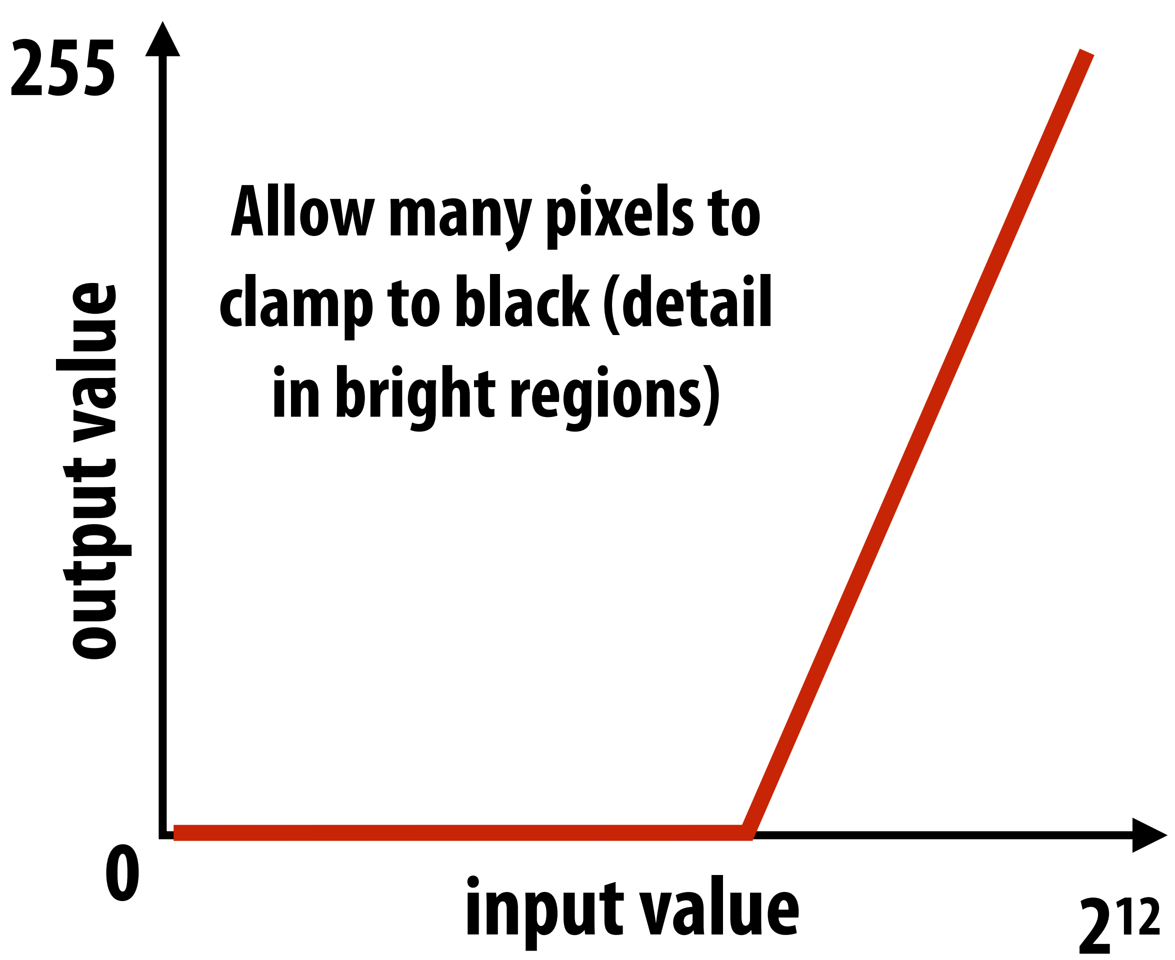
Global tone mapping

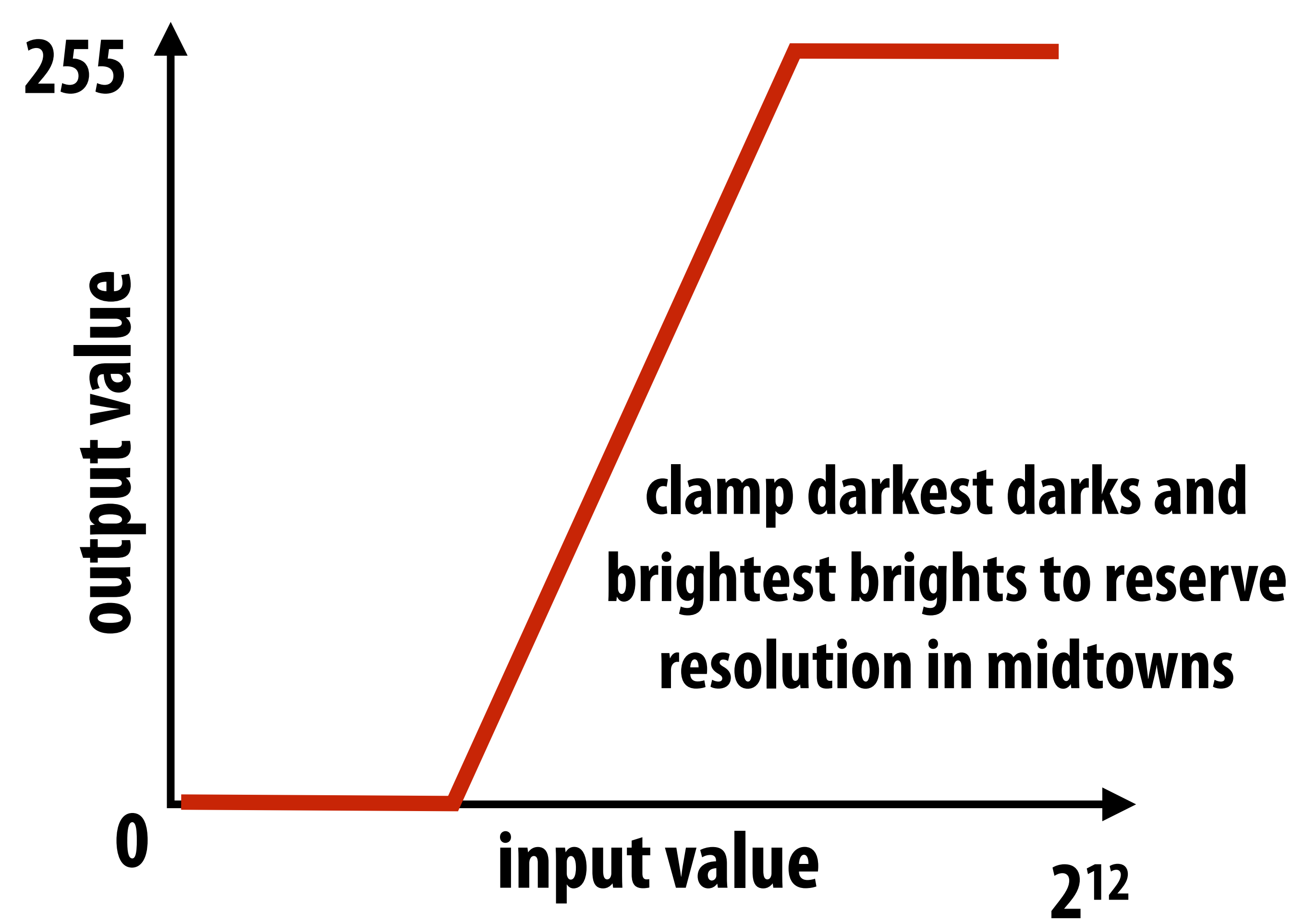
- Measured image values: 10-12 bits / pixel, but common image formats (8-bits/ pixel)
- How to convert 12 bit number to 8 bit number?

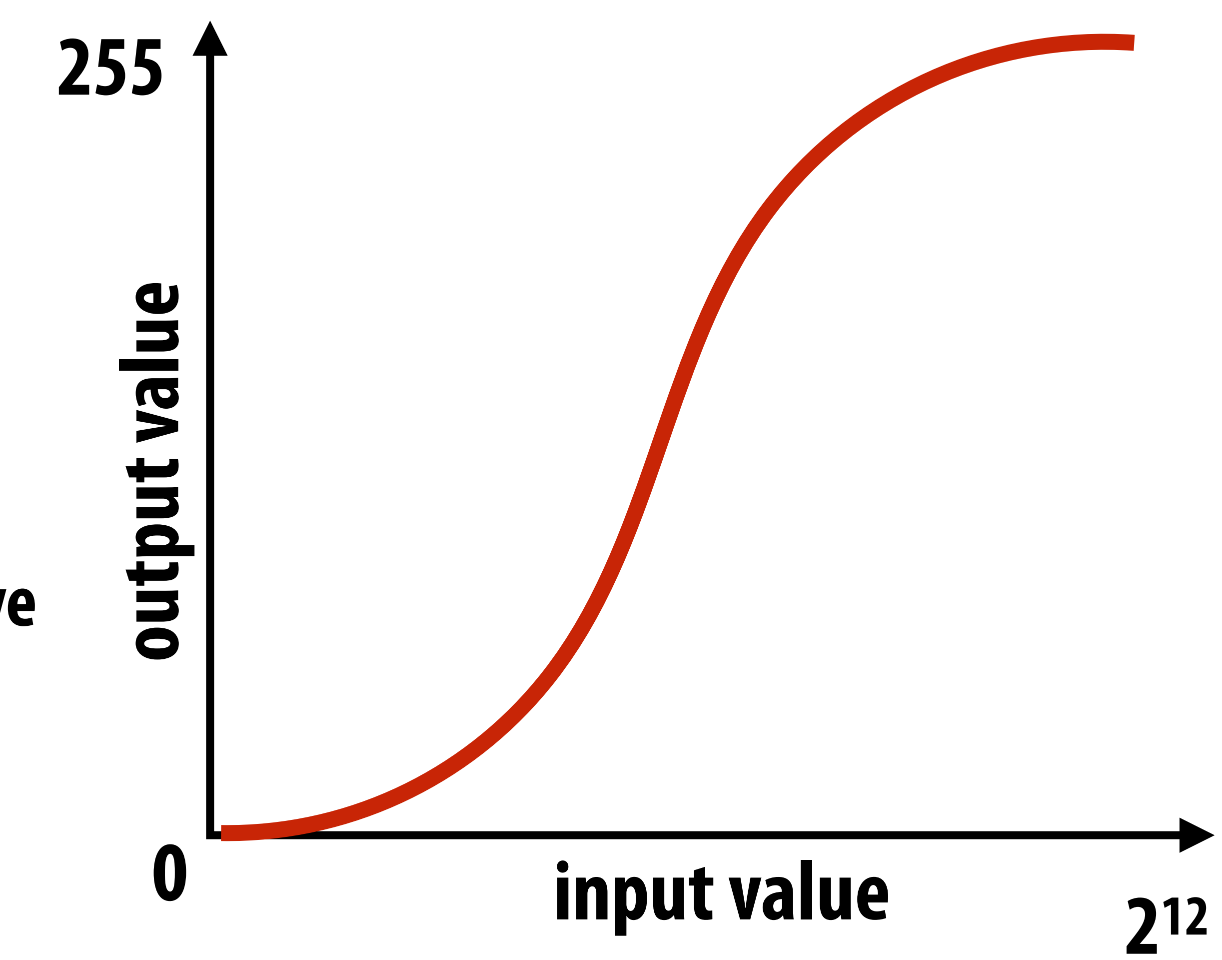
$$out(x,y) = f(in(x,y))$$





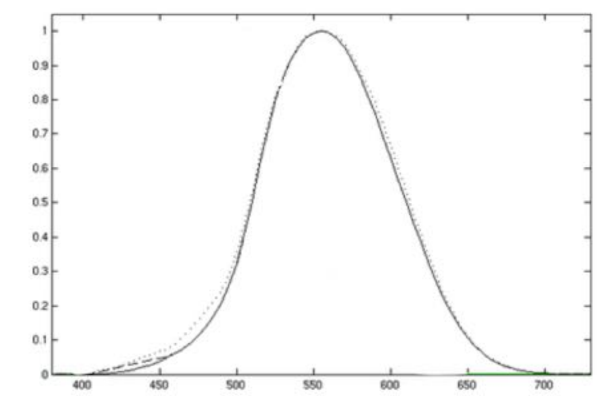




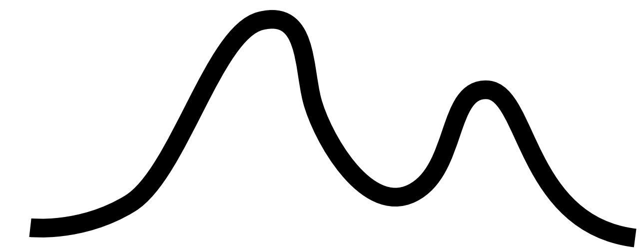


Lightness (perceived brightness) aka luma

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



$*$



(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: $L^* \propto Y^{0.5}$

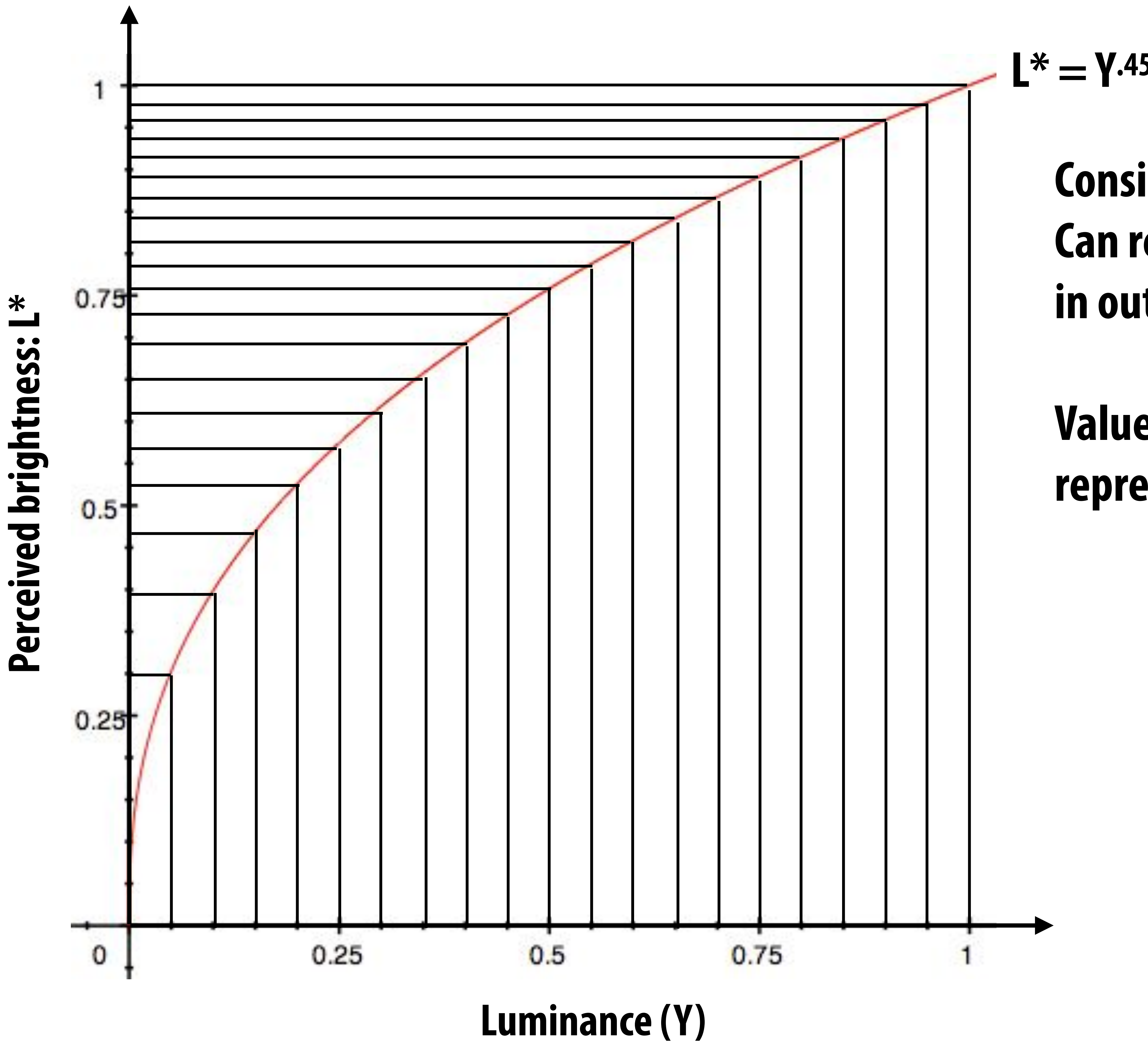
In a dark room, you turn on a light with luminance: Y_1

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')

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



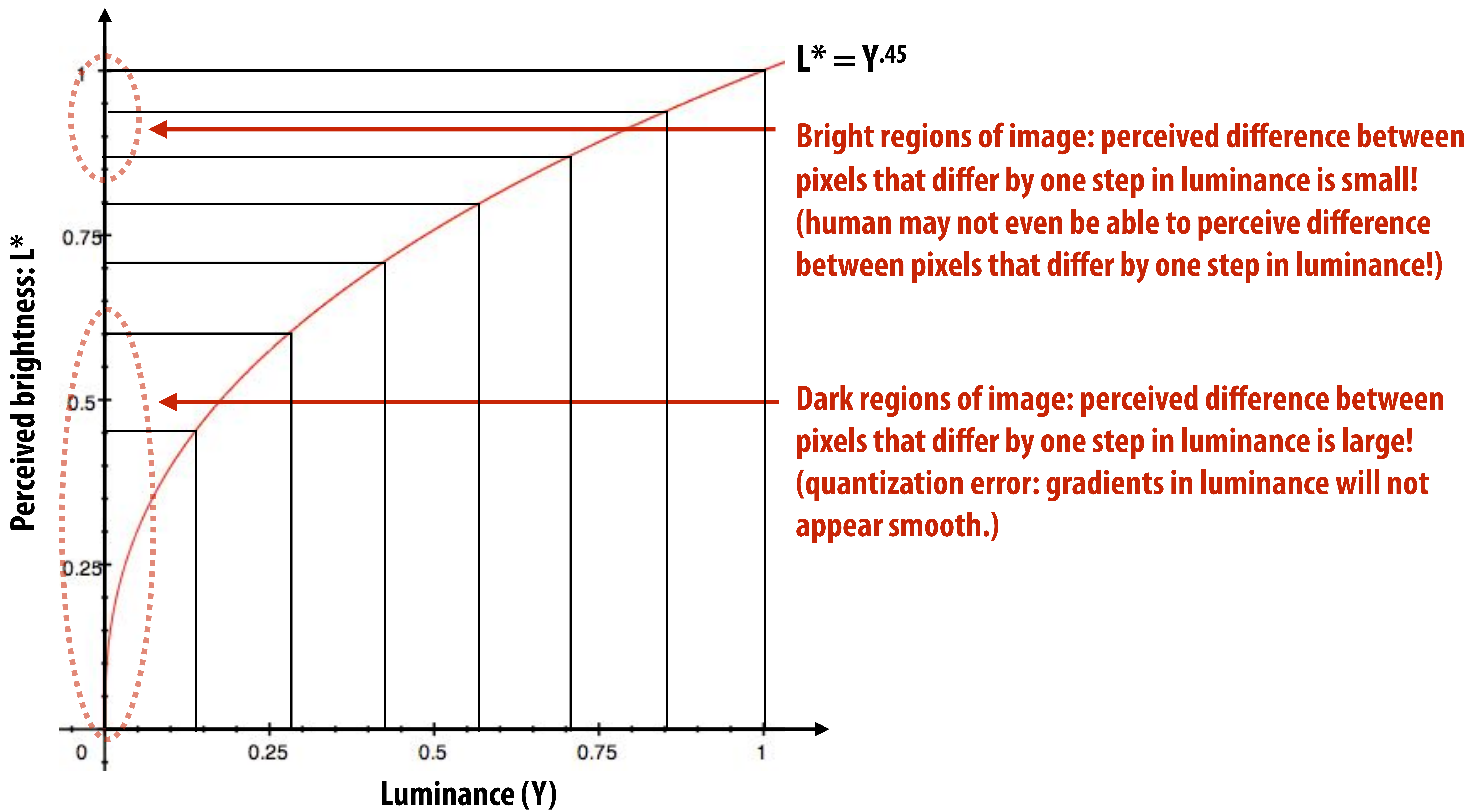
Consider 12-bit sensor pixel:
Can represent 4096 unique luminance values in output image

Values are ~ linear in luminance since they represent the sensor's response

Problem: quantization error

Many common image formats store 8 bits per channel (256 unique values)

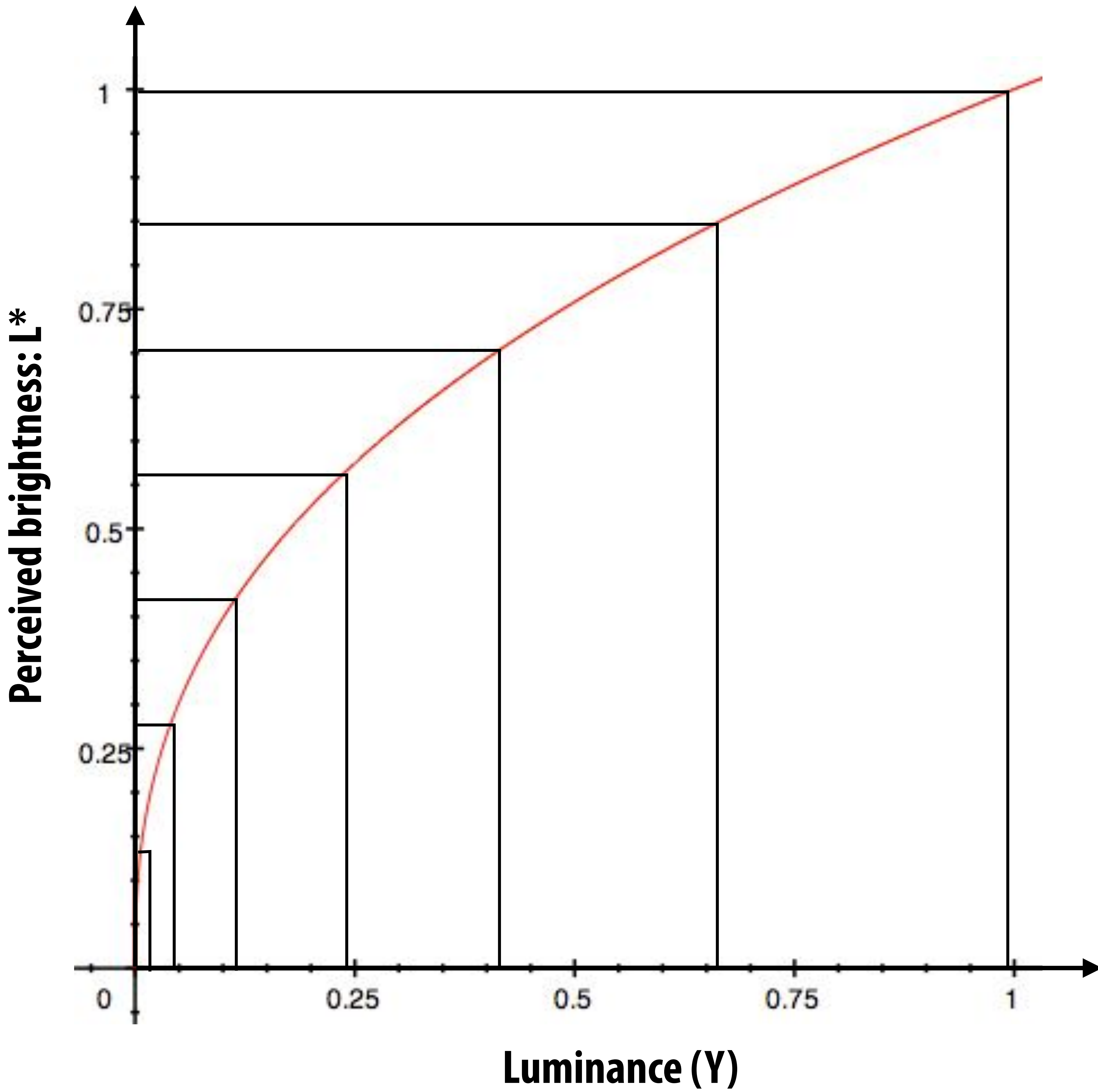
Insufficient precision to represent brightness in darker regions of image



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

Store lightness in 8-bit value, not luminance

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

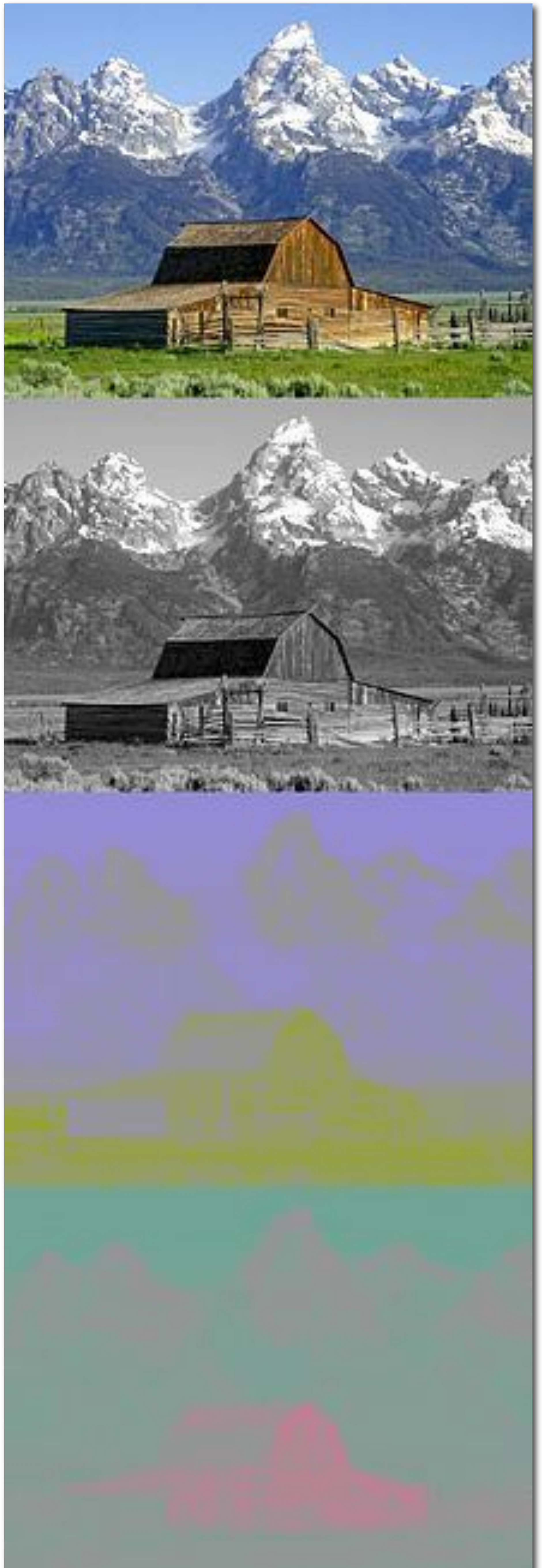


Solution: pixel stores $Y^{0.45}$
Must compute $(\text{pixel\_value})^{2.2}$ prior to display on LCD

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?

Y'CbCr color space



Y'

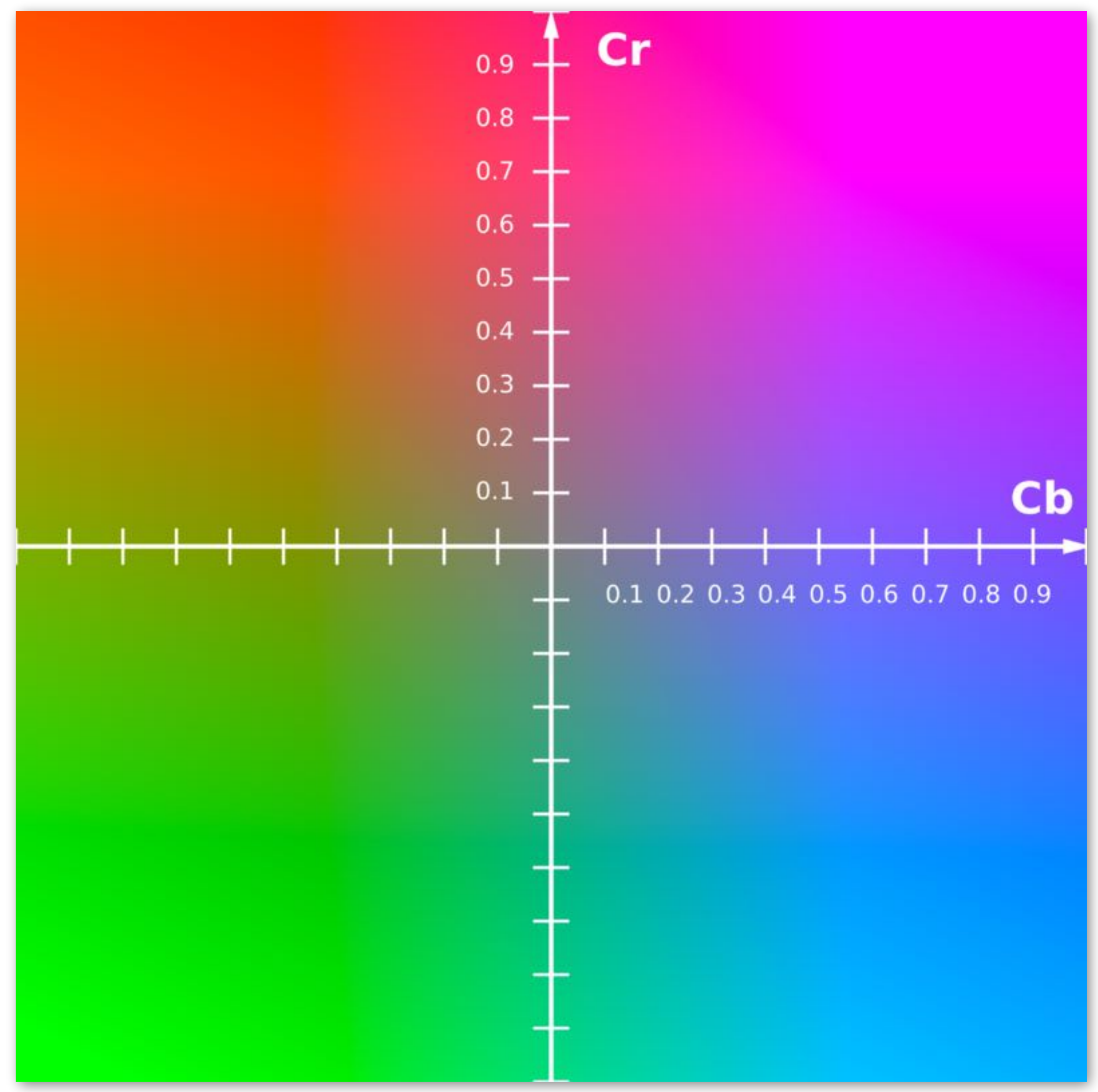
Cb

Cr

Recall: colors are represented as point in 3-space

RGB is just one possible basis for representing color

Y'CbCr separates luminance from hue in representation



Y' = luma: perceived luminance

Cb = blue-yellow deviation from gray

Cr = red-cyan deviation from gray

"Gamma corrected" RGB (primed notation indicates perceptual (non-linear) space)

We'll describe what this means this later in the lecture.

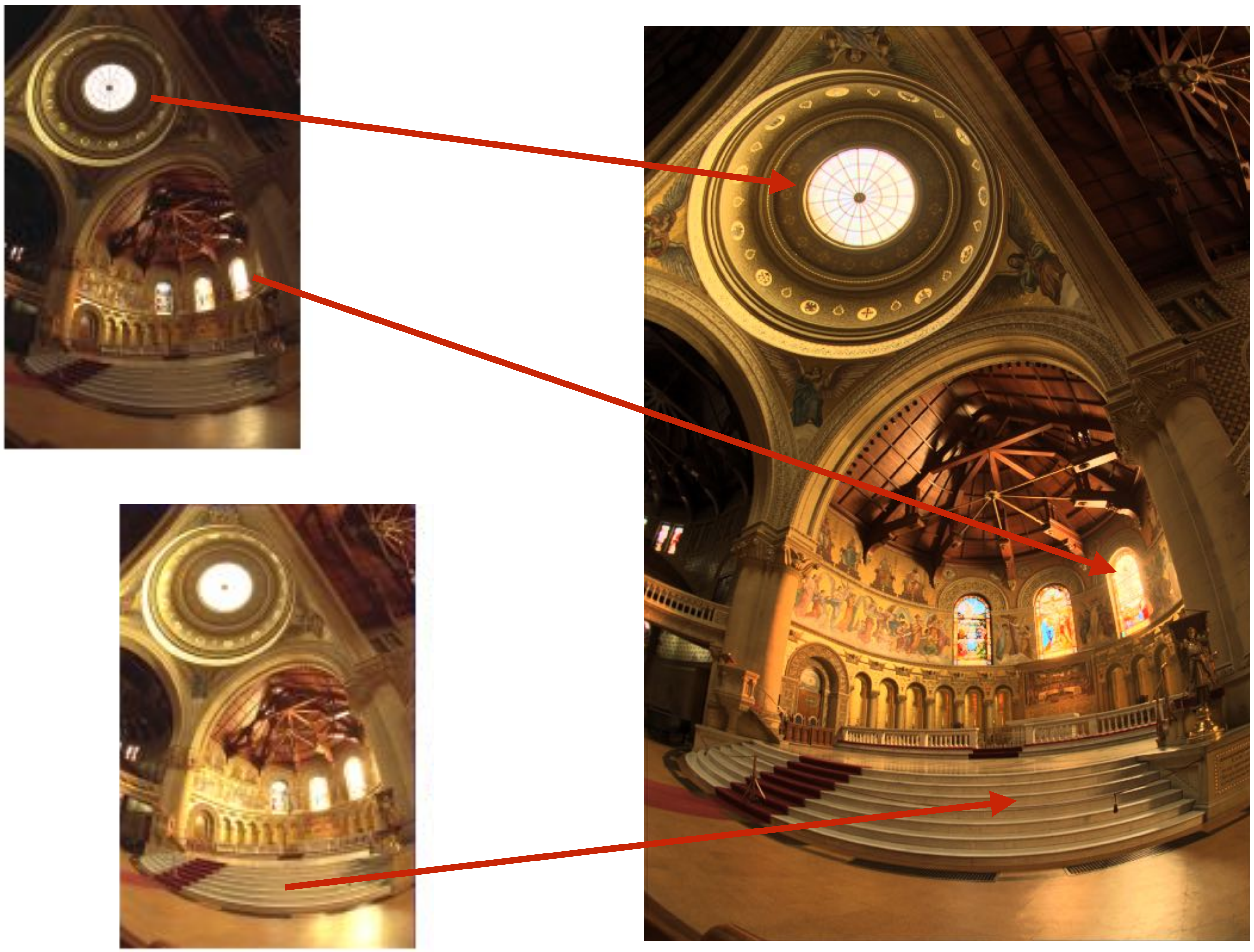
Conversion matrix from R'G'B' to Y'CbCr:

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

Image credit: Wikipedia

Local tone mapping

- **Different regions of the image undergo different tone mapping curves (preserve detail in both dark and bright regions)**



Local tone adjustment

Pixel values

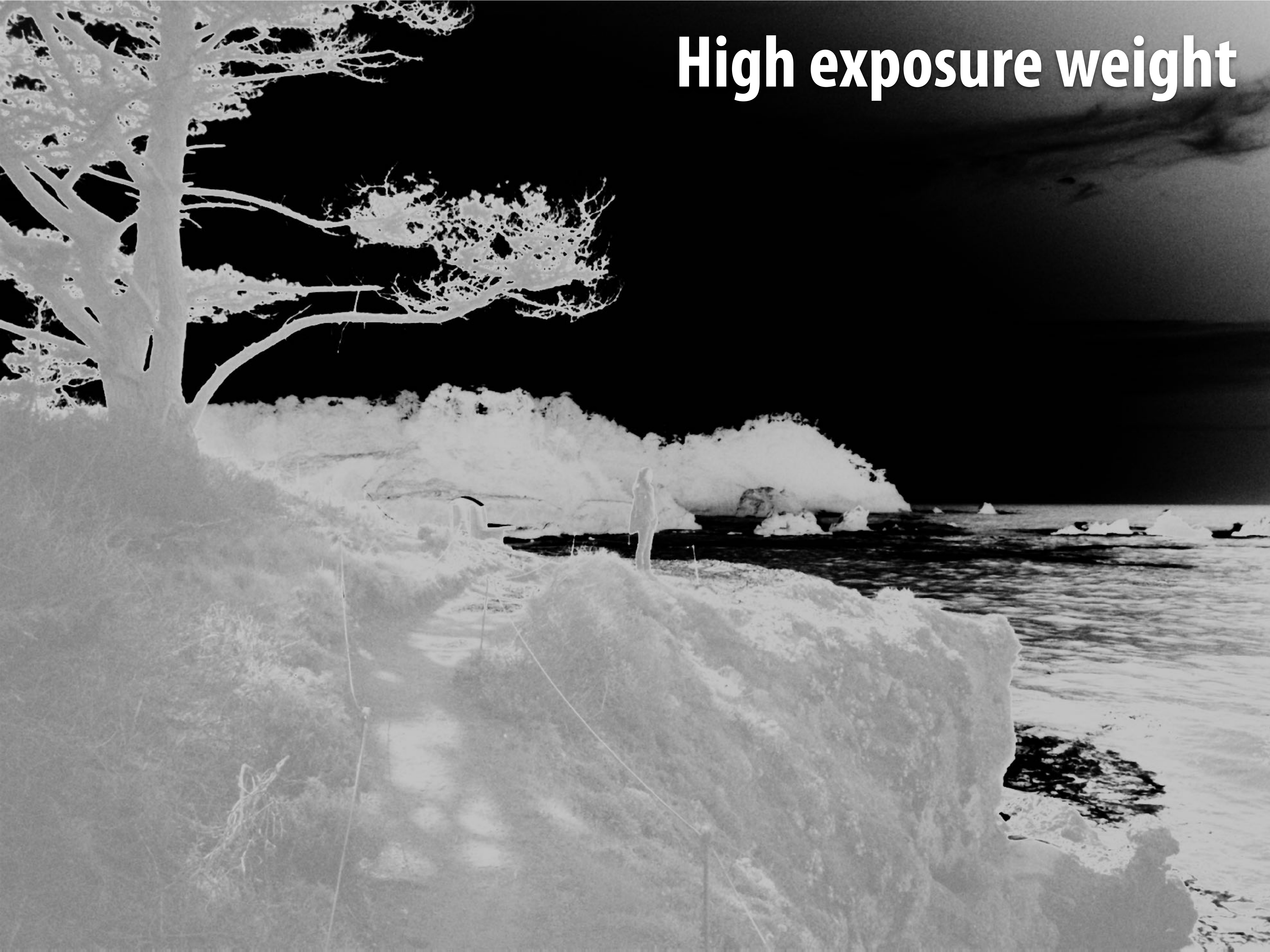
Weights

Improve picture's aesthetics by locally adjusting contrast, boosting dark regions, decreasing bright regions (no physical basis)

Combined image
(unique weights per pixel)

Image credit: Mertens 2007

High exposure image


High exposure weight


Low exposure image


Low exposure weight

Combined result

Combined result

Local tone mapping was performed on lightness (luma).
Now I added back in chrominance channels.

Challenge of merging images

Four exposures (weights not shown)

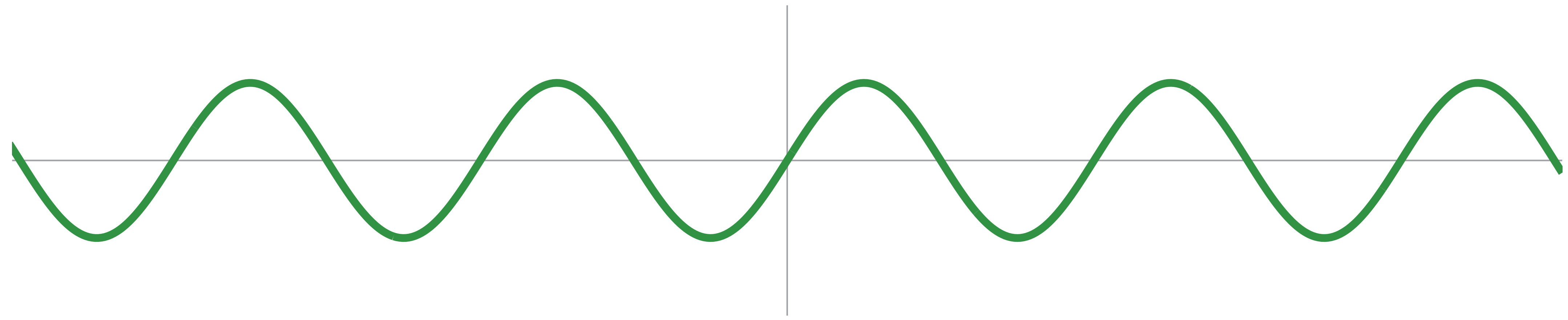
Merged result
(based on weight masks)
Notice heavy "banding" since absolute
intensity of different exposures is different

Merged result
(after blurring weight mask)
Notice "halos" near edges

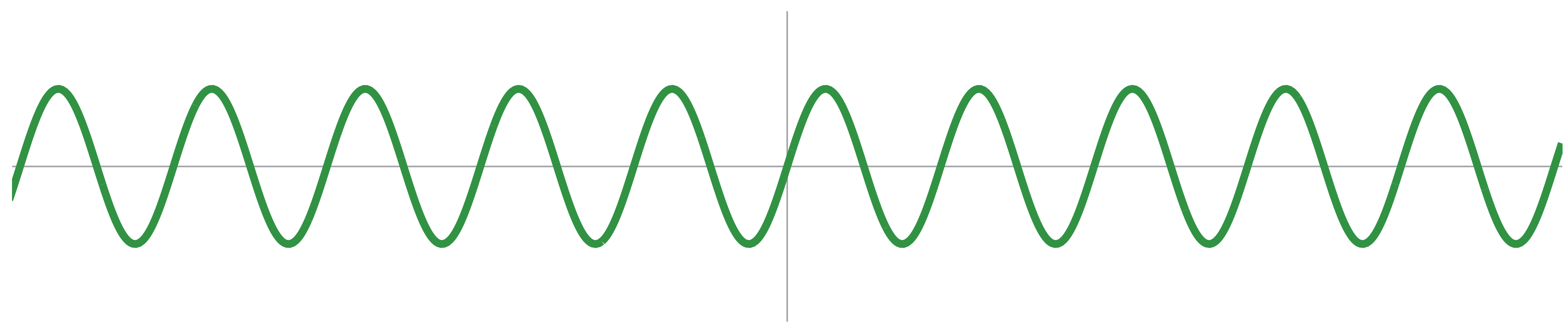
Review:
Frequency interpretation of images

Representing sound as a superposition of frequencies

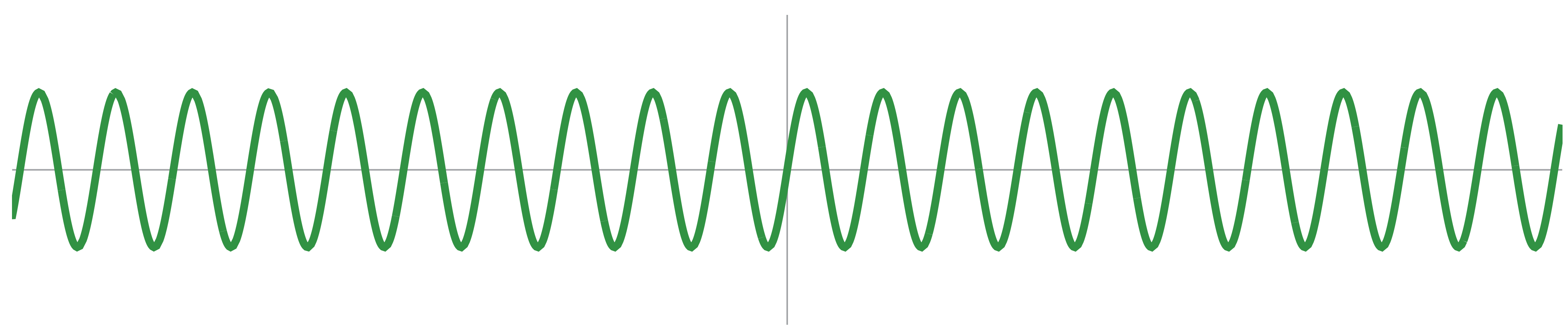
$f_1(x) = sin(\pi x)$



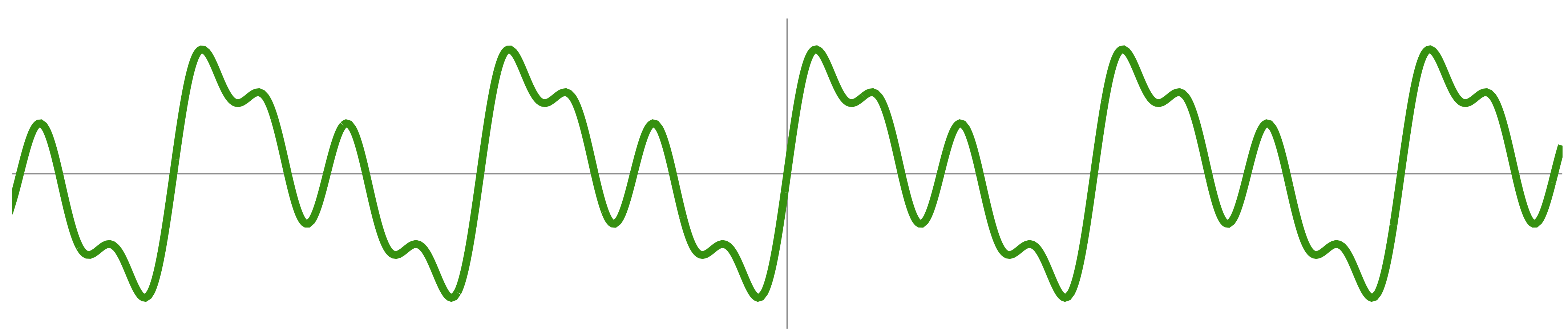
$f_2(x) = sin(2\pi x)$



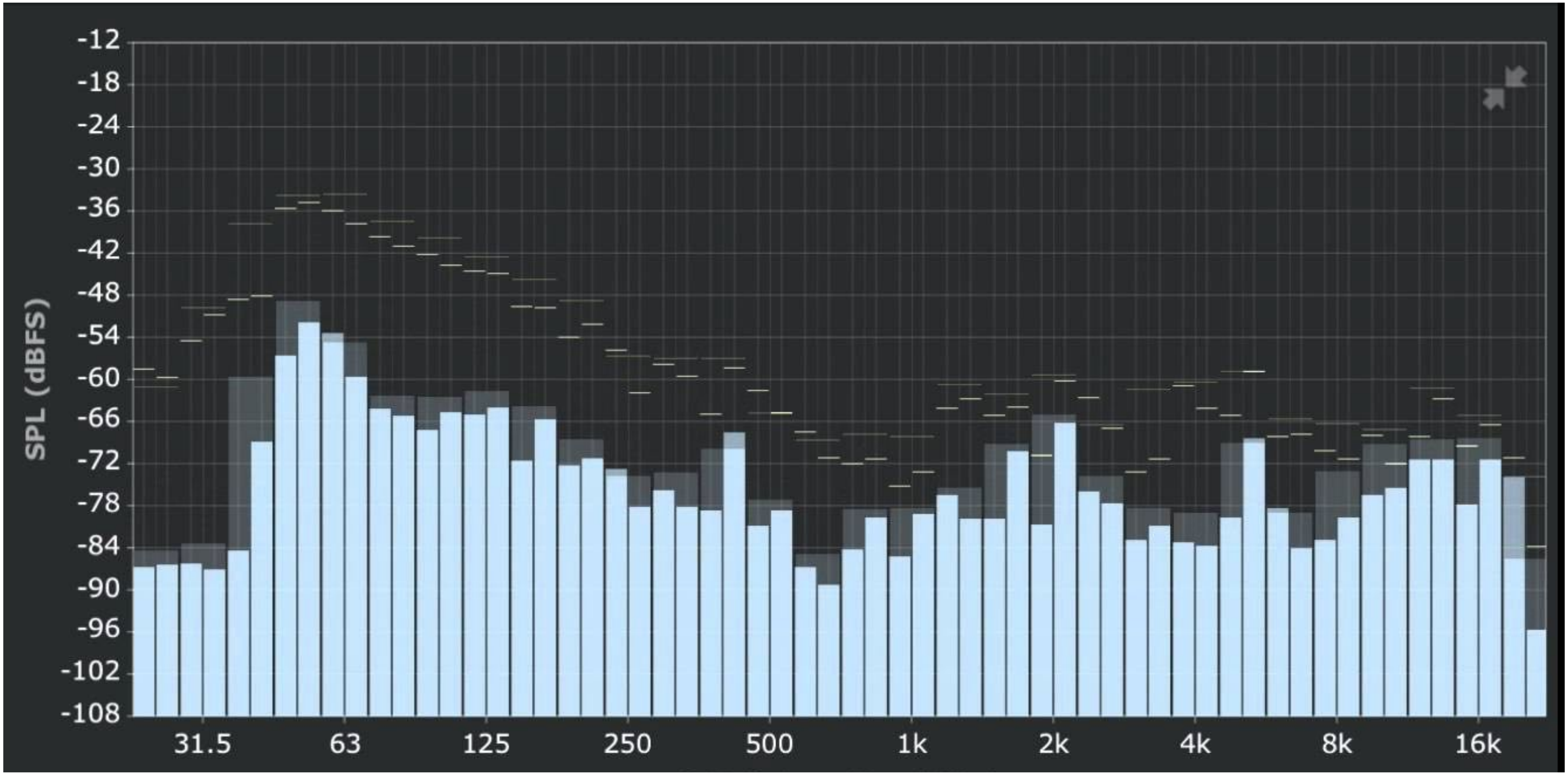
$f_4(x) = sin(4\pi x)$



$f(x) = f_1(x) + 0.75\, f_2(x) + 0.5\, f_4(x)$



Audio spectrum analyzer: representing sound as a sum of its constituent frequencies



Intensity of low-frequencies (bass)

Intensity of high frequencies

Image credit: ONYX Apps

Fourier transform

- **Convert representation of signal from spatial/temporal domain to frequency domain by projecting signal into its component frequencies**

$$f(\xi) = \int_{-\infty}^{\infty} f(x)e^{-2\pi i x \xi} dx$$

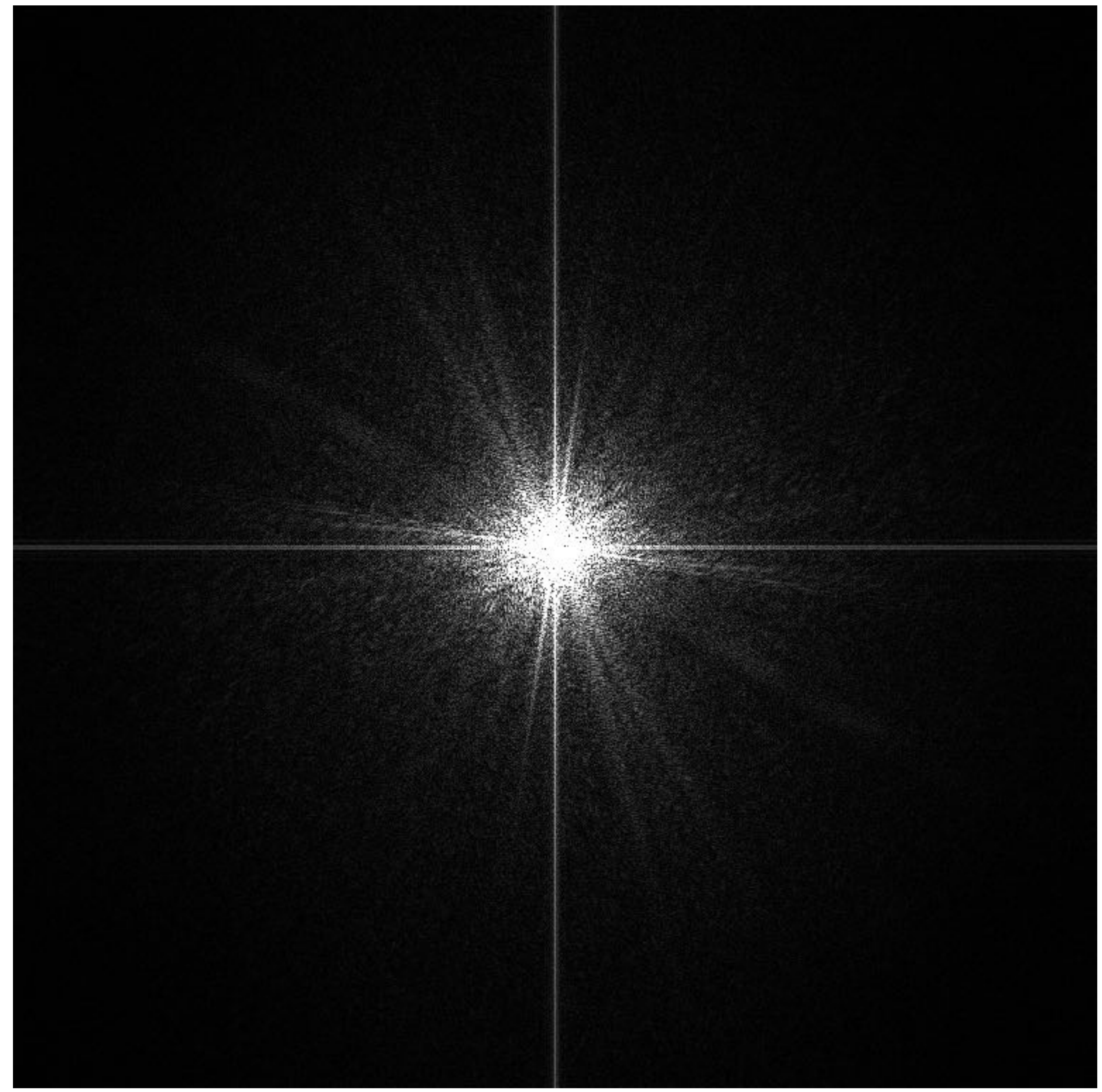
$$= \int_{-\infty}^{\infty} f(x)(\cos(2\pi\xi x) - i\sin(2\pi\xi x))dx$$

- **2D form:**

$$f(u,v) = \int\int f(x,y)e^{-2\pi i(ux+vy)} dxdy$$

Visualizing the frequency content of images

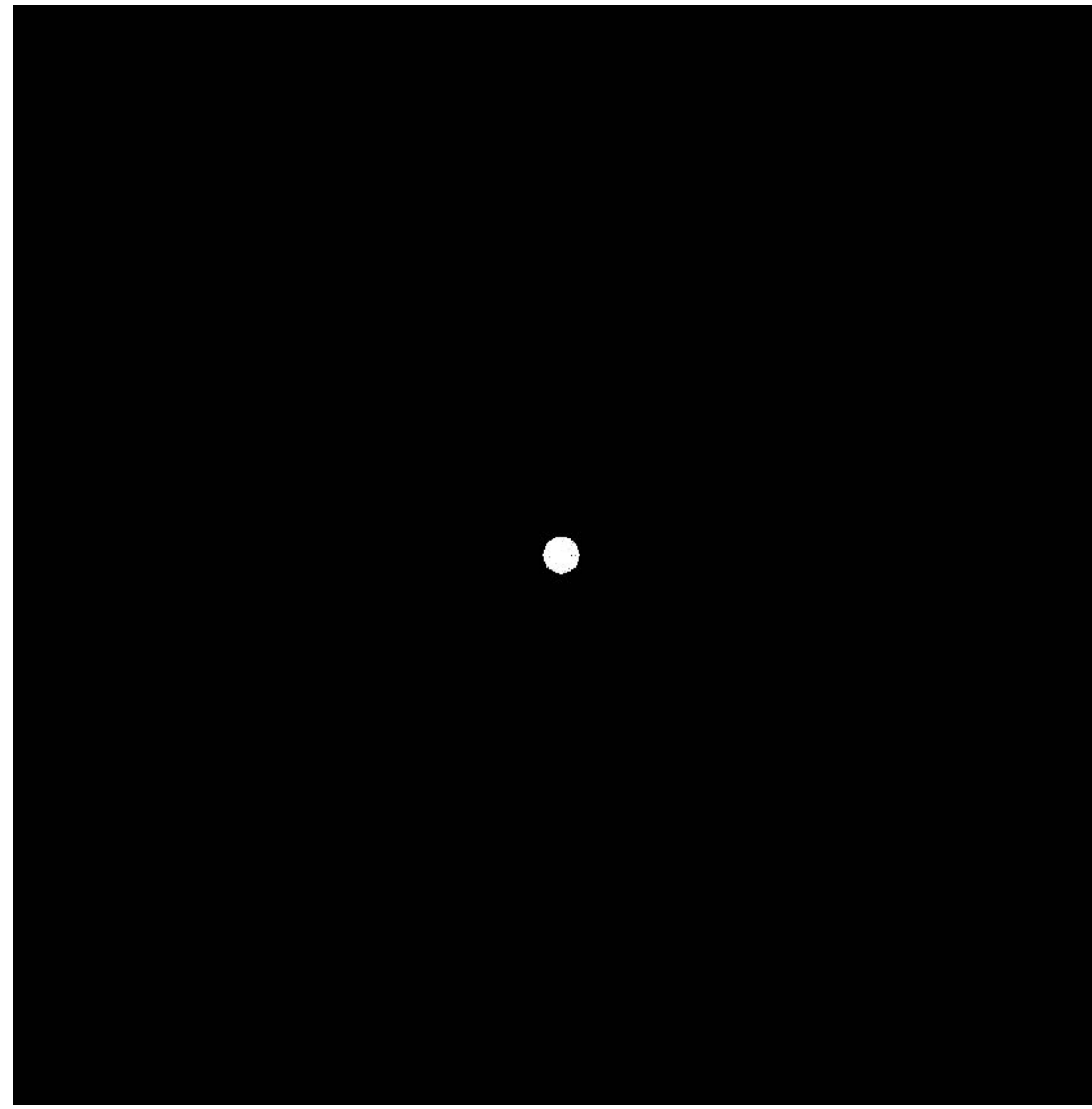
Spatial domain result



Spectrum

Low frequencies only (smooth gradients)

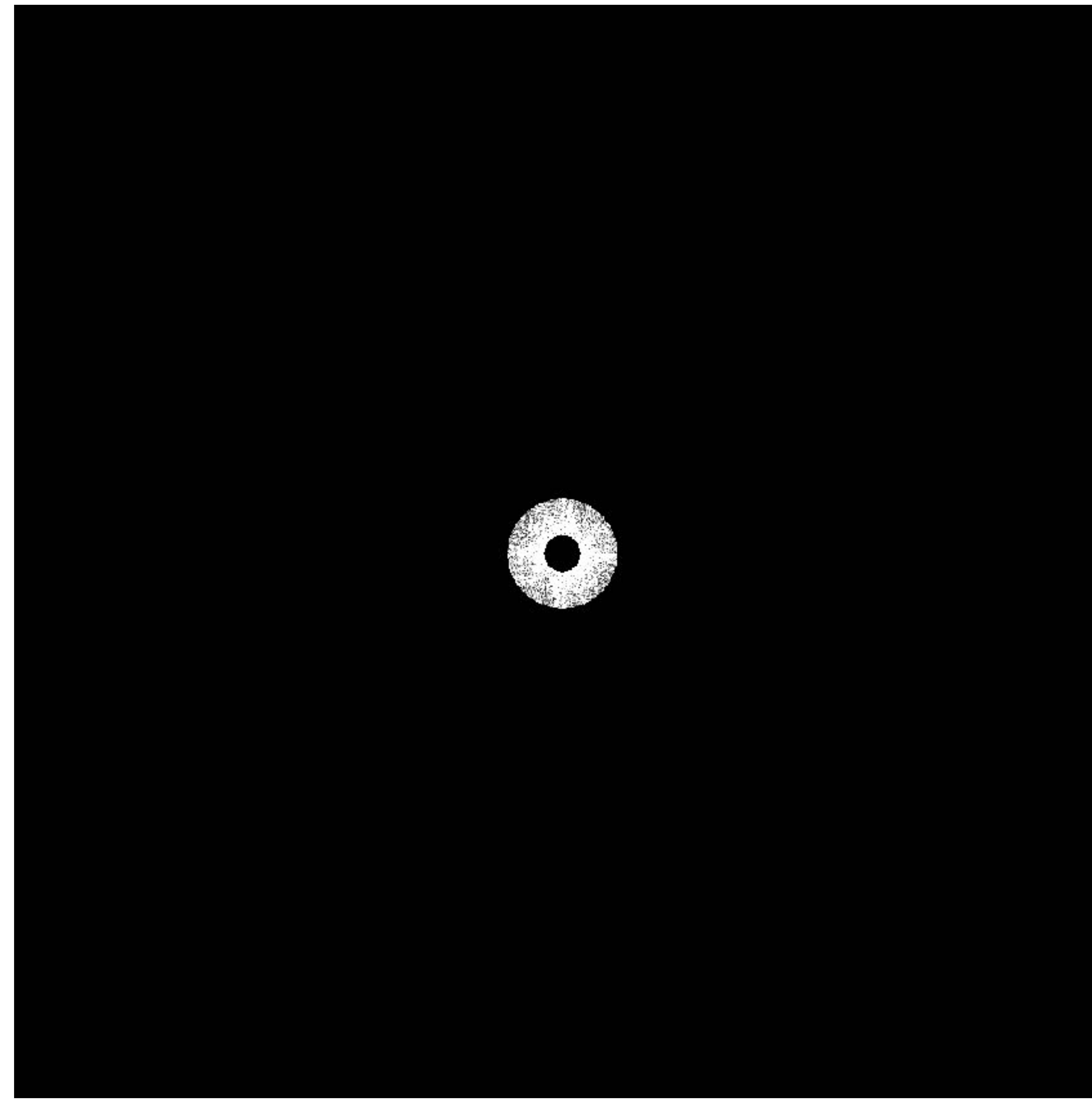
Spatial domain result



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

Mid-range frequencies

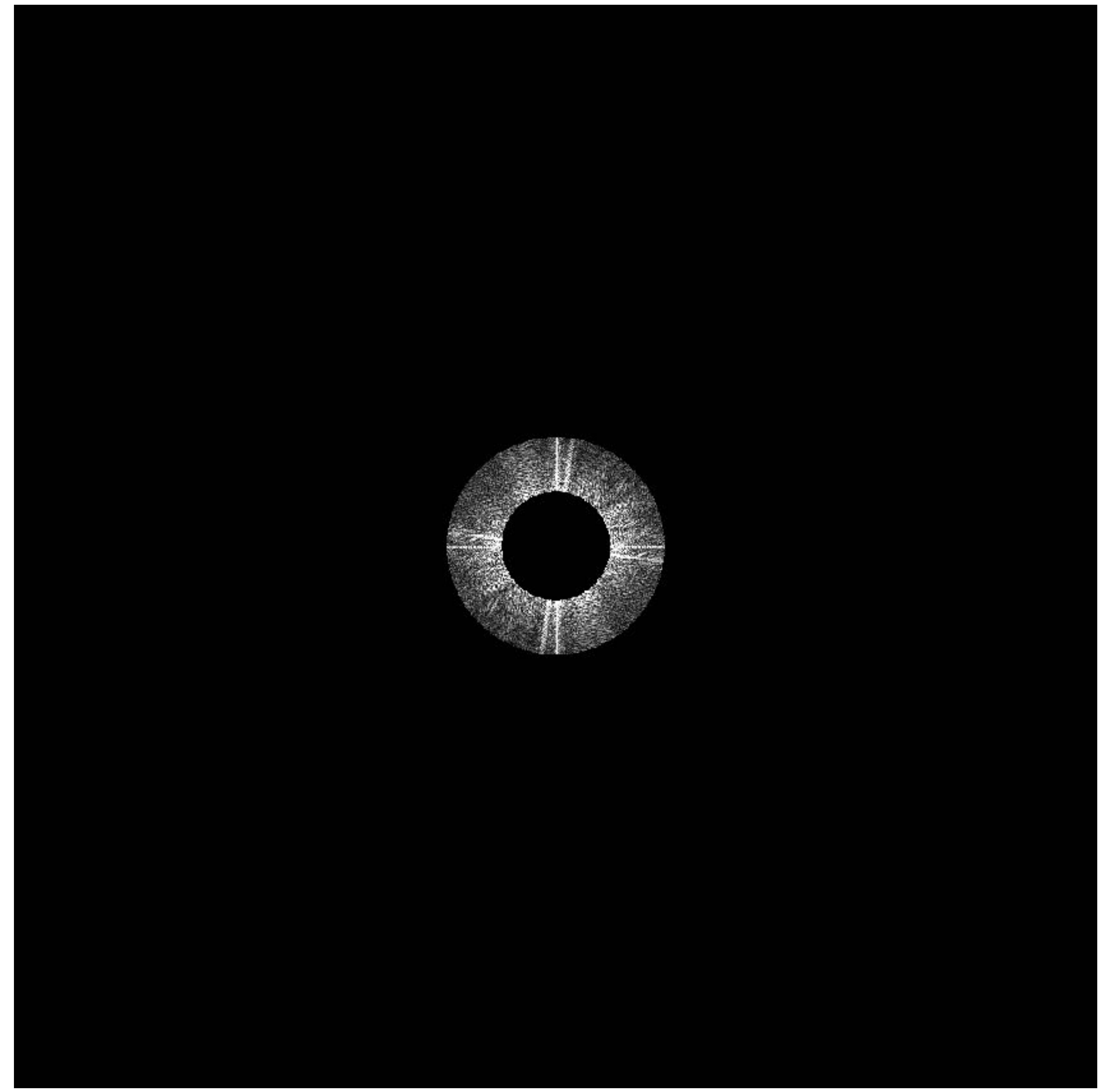
Spatial domain result



Spectrum (after band-pass filter)

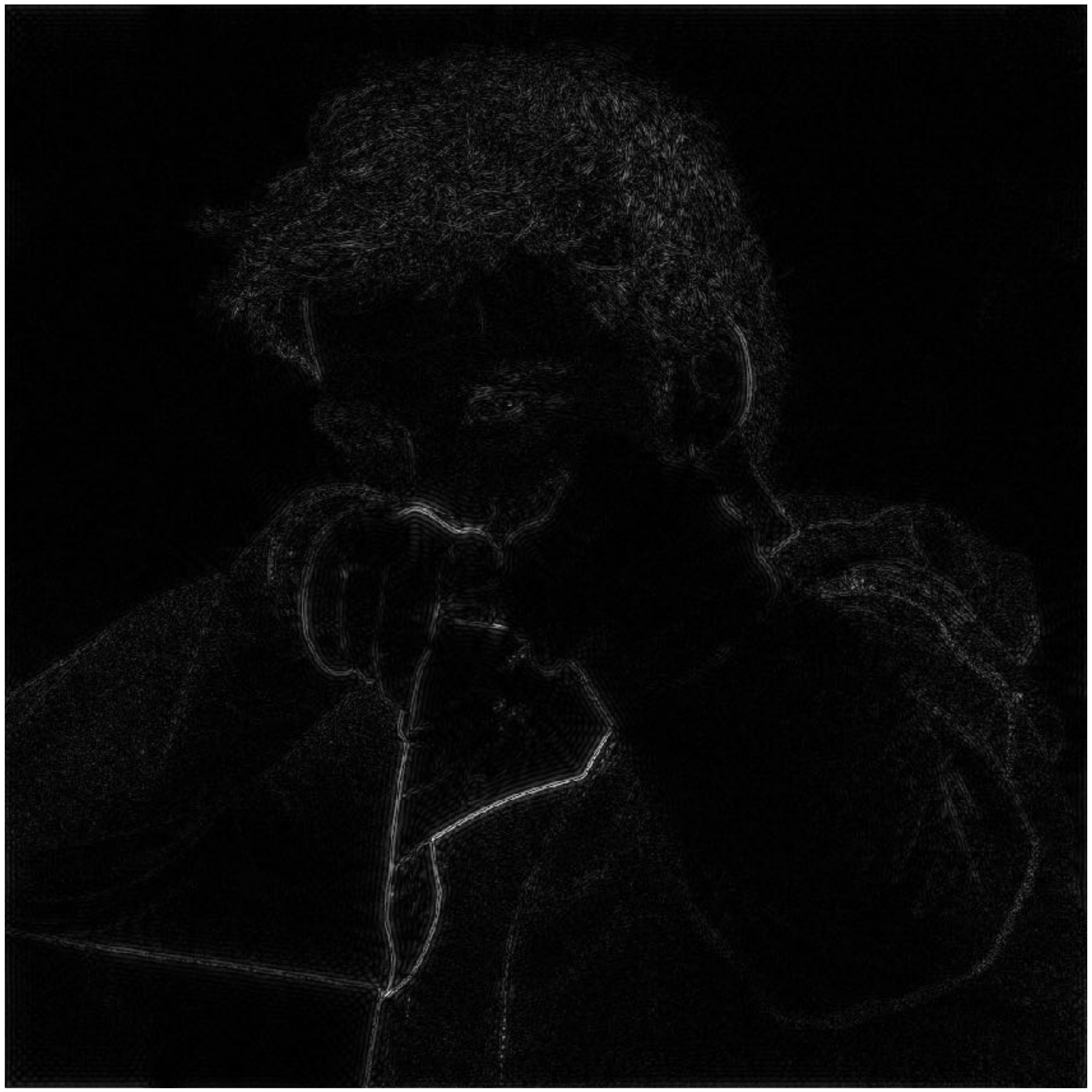
Mid-range frequencies

Spatial domain result

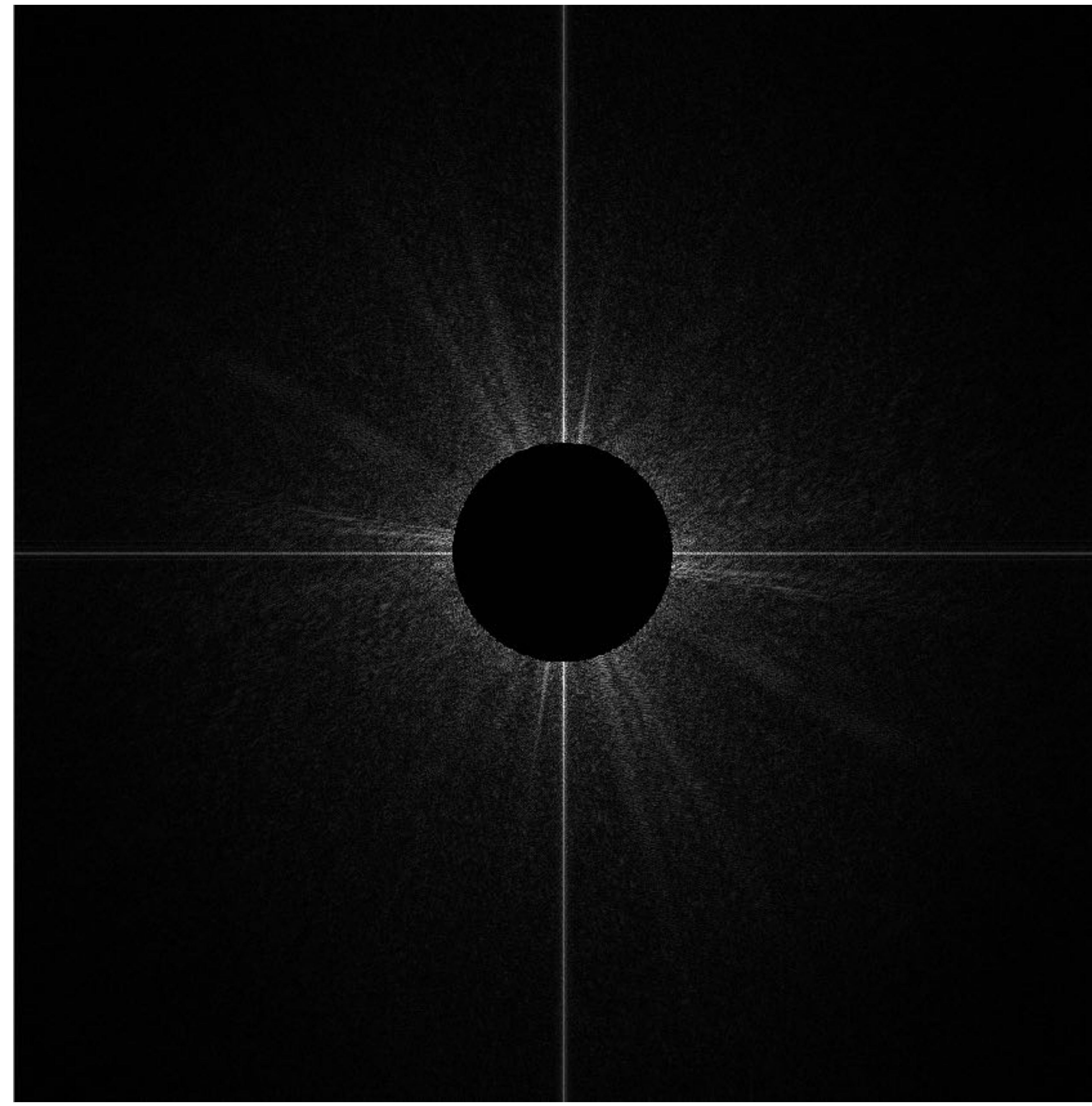


Spectrum (after band-pass filter)

High frequencies (edges)



Spatial domain result
(strongest edges)



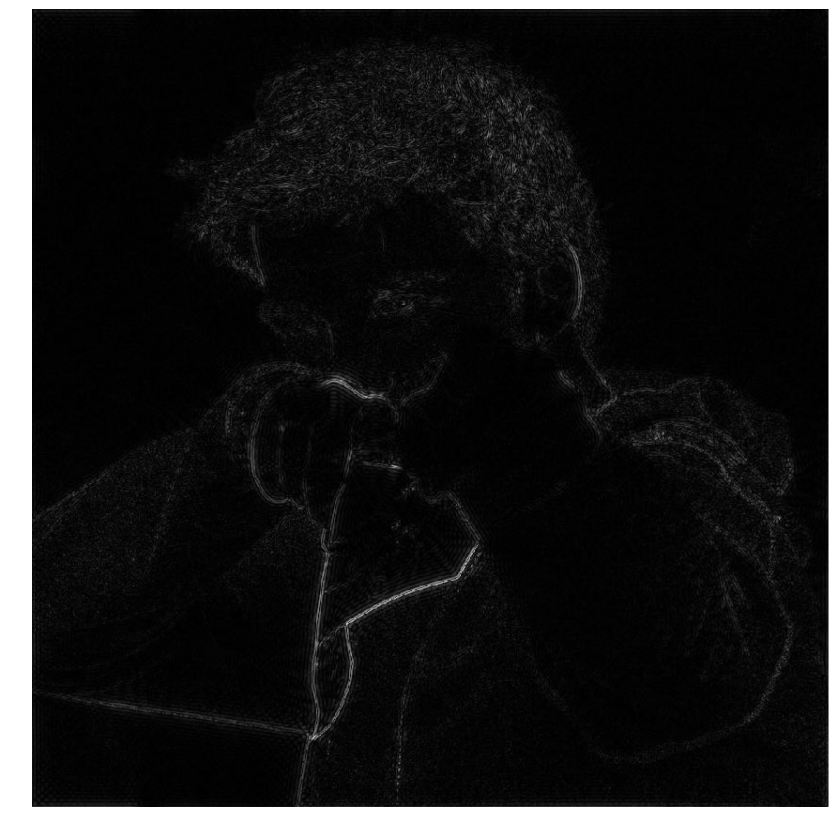
Spectrum (after high-pass filter)
All frequencies below threshold
have 0 magnitude

An image as a sum of its frequency components

+

+

+



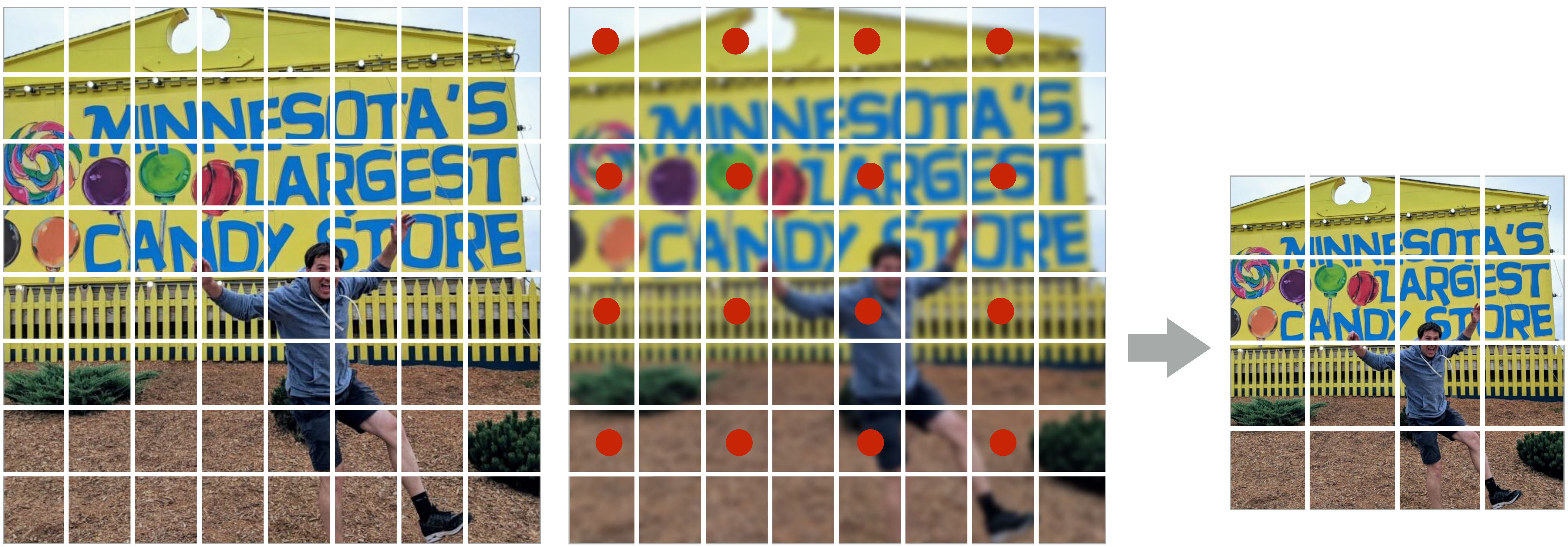
=

But what if we wish to localize image edits both in space and in frequency?

(Adjust certain frequency content of image,
in a particular region of the image)

Downsample

- Step 1: Remove high frequencies (aka blur)
- Step 2: Sparsely sample pixels (in this example: every other pixel)



Downsample

- **Step 1: Remove high frequencies**
- **Step 2: Sparsely sample pixels (in this example: every other pixel)**

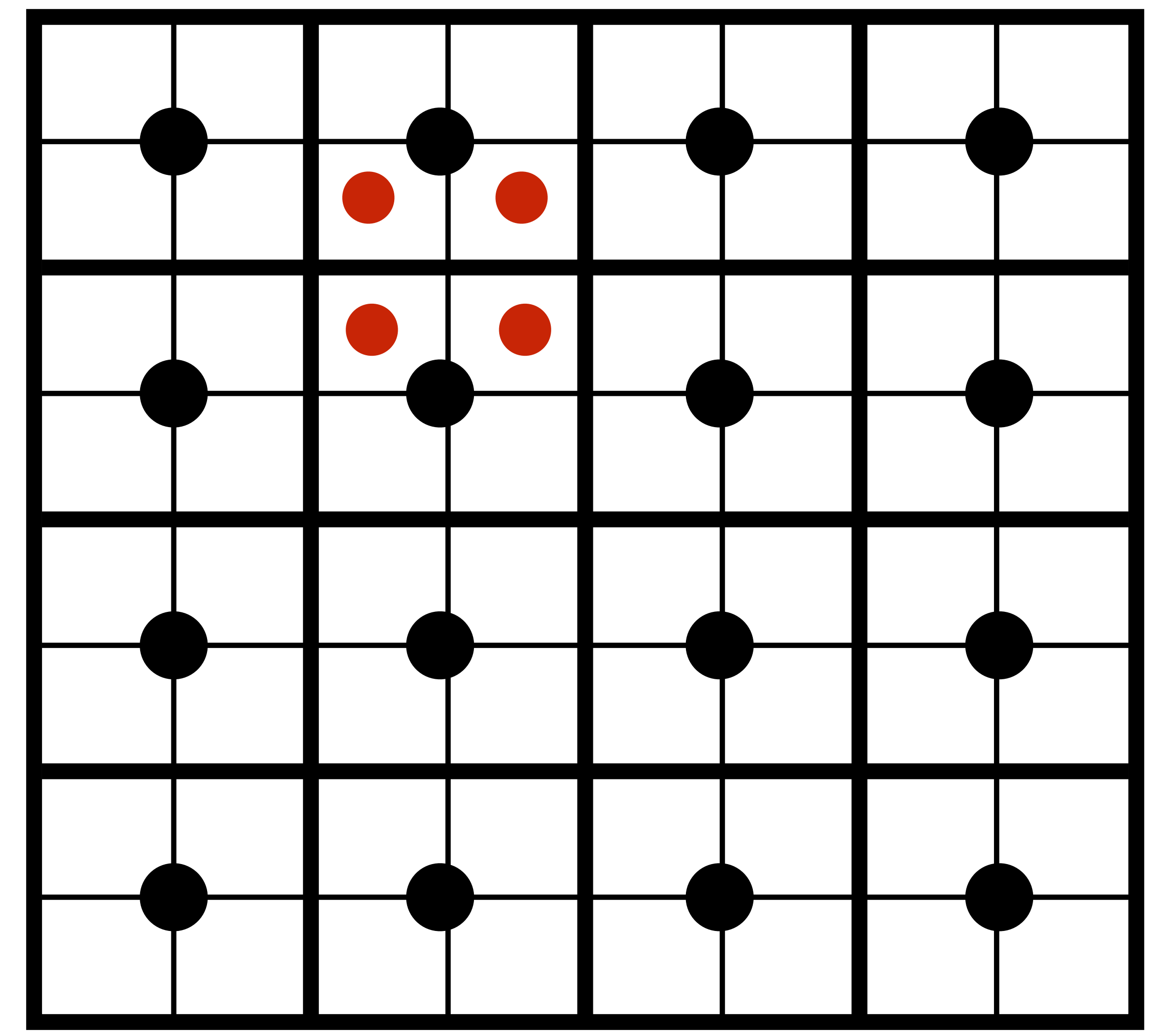
```
float input[(WIDTH+2) * (HEIGHT+2)];
float output[WIDTH/2 * HEIGHT/2];

float weights[] = {1/64, 3/64, 3/64, 1/64,    // 4x4 blur (approx Gaussian)
                   3/64, 9/64, 9/64, 3/64,
                   3/64, 9/64, 9/64, 3/64,
                   1/64, 3/64, 3/64, 1/64};

for (int j=0; j<HEIGHT/2; j++) {
   for (int i=0; i<WIDTH/2; i++) {
      float tmp = 0.f;
      for (int jj=0; jj<4; jj++)
         for (int ii=0; ii<4; ii++)
            tmp += input[(2*j+jj)*(WIDTH+2) + (2*i+ii)] * weights[jj*3 + ii];
      output[j*WIDTH/2 + i] = tmp;
  }
}
```

Upsample

Via bilinear interpolation of samples from low resolution image

Upsample

Via bilinear interpolation of samples from low resolution image



```
float input[WIDTH * HEIGHT];
float output[2*WIDTH * 2*HEIGHT];

for (int j=0; j<2*HEIGHT; j++) {
   for (int i=0; i<2*WIDTH; i++) {
      int row = j/2;
      int col = i/2;
      float w1 = (i%2) ? .75f : .25f;
      float w2 = (j%2) ? .75f : .25f;

      output[j*2*WIDTH + i] = w1 * w2 * input[row*WIDTH + col] +
                              (1.0-w1) * w2 * input[row*WIDTH + col+1] +
                              w1 * (1-w2) * input[(row+1)*WIDTH + col] +
                              (1.0-w1)*(1.0-w2) * input[(row+1)*WIDTH + col+1];
  }
}
```

Gaussian pyramid

$G_0 = \text{image}$

$G_1 = \text{down}(G_0)$

$G_2 = \text{down}(G_1)$

Each image in pyramid contains increasingly low-pass filtered signal

down() = downsample operation

Gaussian pyramid

G_0

Gaussian pyramid

G_1

Gaussian pyramid

G_2

Gaussian pyramid

G_3

Gaussian pyramid

G_4

Gaussian pyramid

G_5

Laplacian pyramid

$L_0 = G_0 - up(G_1)$

G_0

$G_1 = down(G_0)$

Each (increasingly numbered) level in Laplacian pyramid represents a band of (increasingly lower) frequency information in the image

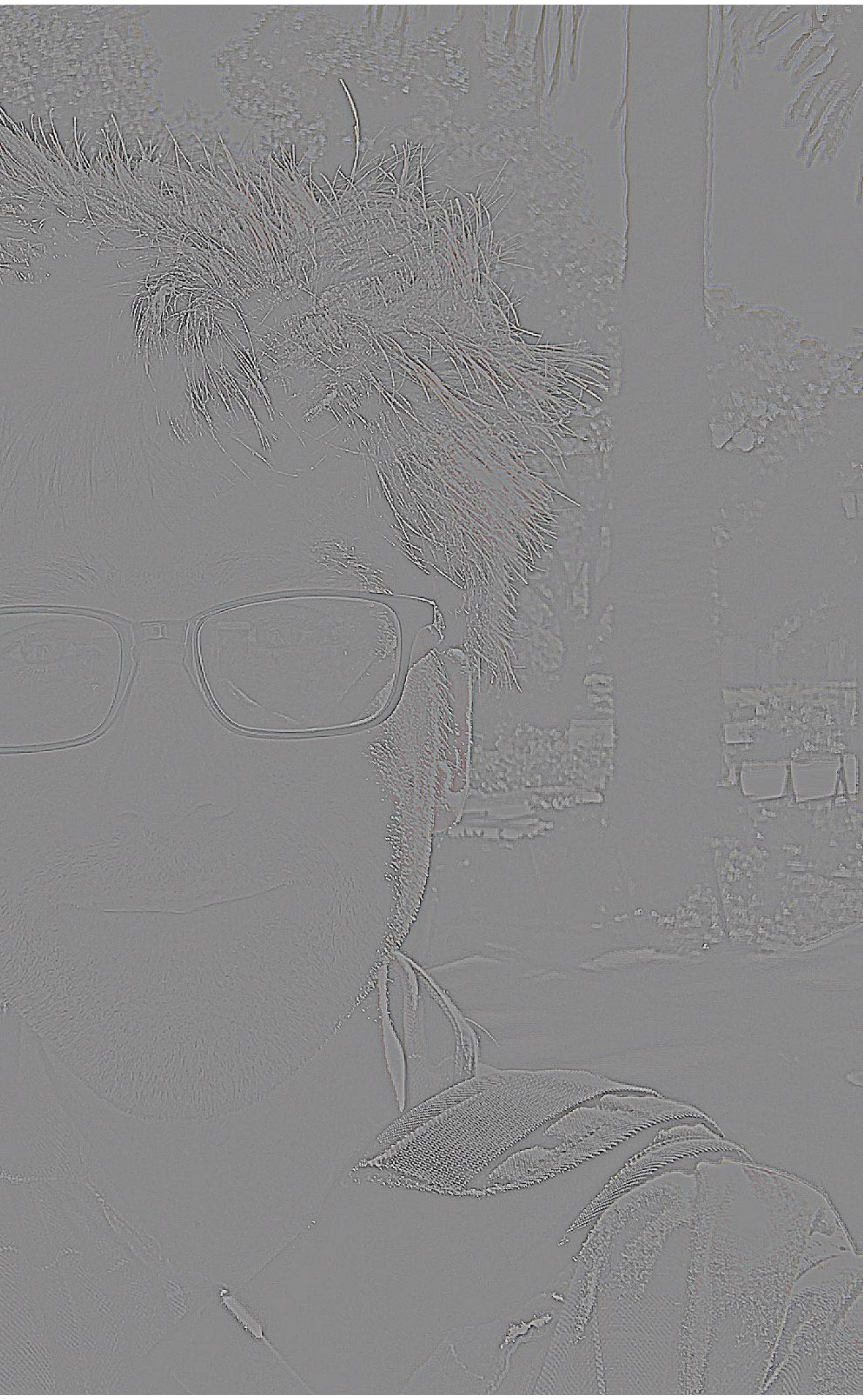
[Burt and Adelson 83]

Laplacian pyramid

$L_0 = G_0 - up(G_1)$

$L_1 = G_1 - up(G_2)$

Laplacian pyramid

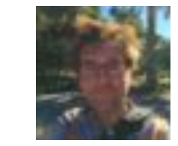


$L_0 = G_0 - up(G_1)$

$L_1 = G_1 - up(G_2)$

$L_2 = G_2 - up(G_3)$

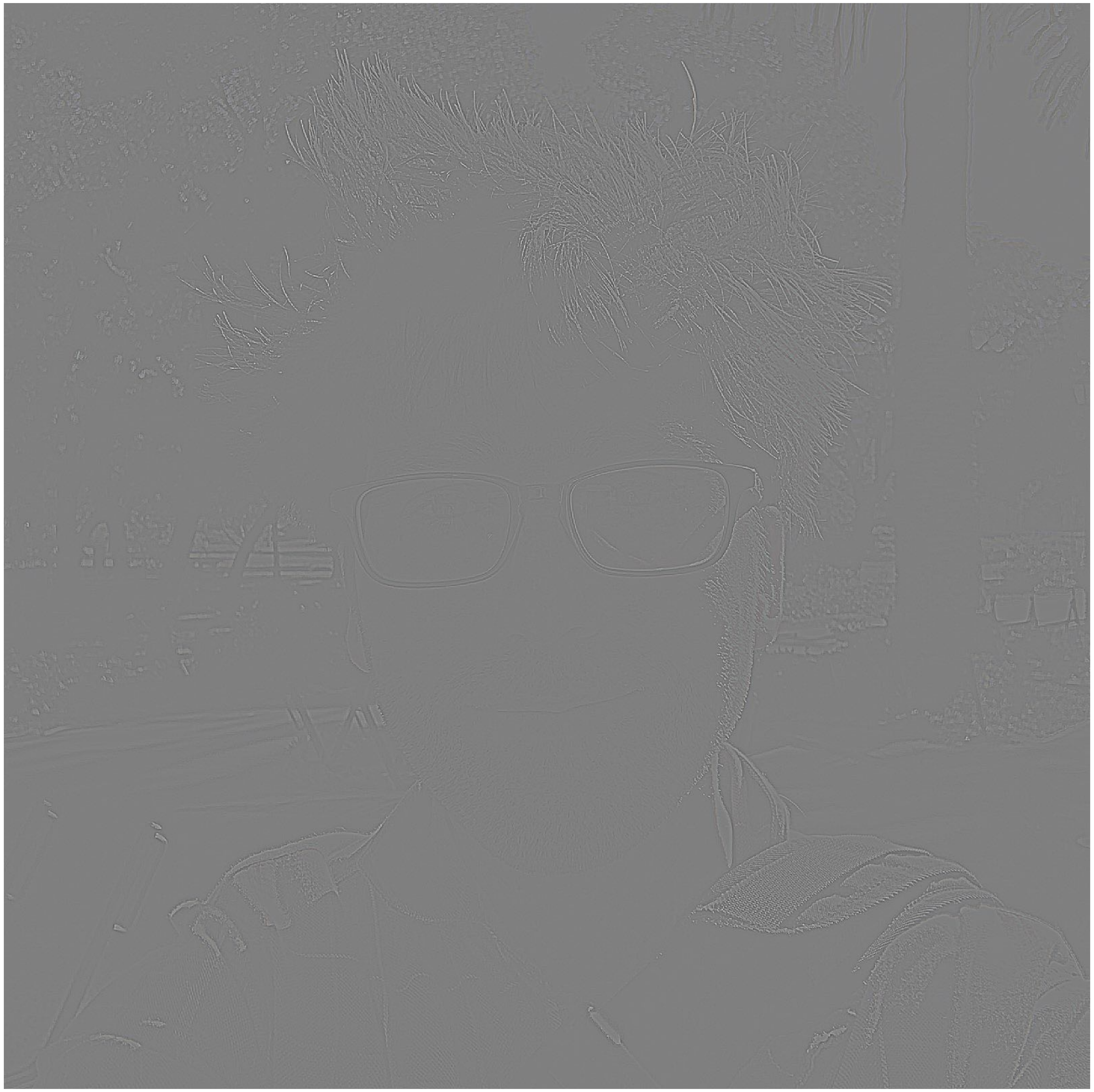
$L_3 = G_3 - up(G_4)$



$L_4 = G_4$

Question: how do you reconstruct original image from its Laplacian pyramid?

Laplacian pyramid



$L_0 = G_0 - up(G_1)$

Laplacian pyramid

$$L_1 = G_1 - \mathrm{up}(G_2)$$

Laplacian pyramid

$L_2 = G_2 - \text{up}(G_3)$

Laplacian pyramid

$L_3 = G_3 - up(G_4)$

Laplacian pyramid

$L_4 = G_4 - up(G_5)$

Laplacian pyramid

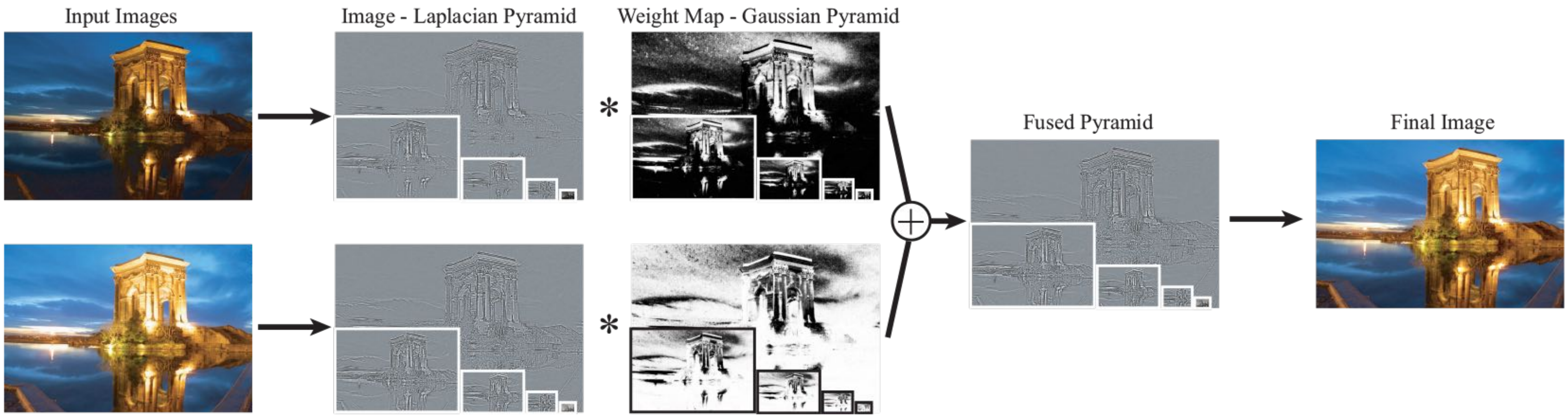
$L_5 = G_5$

Summary

- **Gaussian and Laplacian pyramids are image representations where each pixel maintains information about frequency content in a region of the image**
- **$G_i(x,y)$ — frequencies up to limit given by *i***
- **$L_i(x,y)$ — frequencies added to G_{i+1} to get G_i**
- **Notice: to boost the band of frequencies in image around pixel (x,y), increase coefficient $L_i(x,y)$ in Laplacian pyramid**

Use of Laplacian pyramid in tone mapping

- Compute weights for all Laplacian pyramid levels
- Merge pyramids (image features) not image pixels
- Then "flatten" merged pyramid to get final image



Challenges of merging images



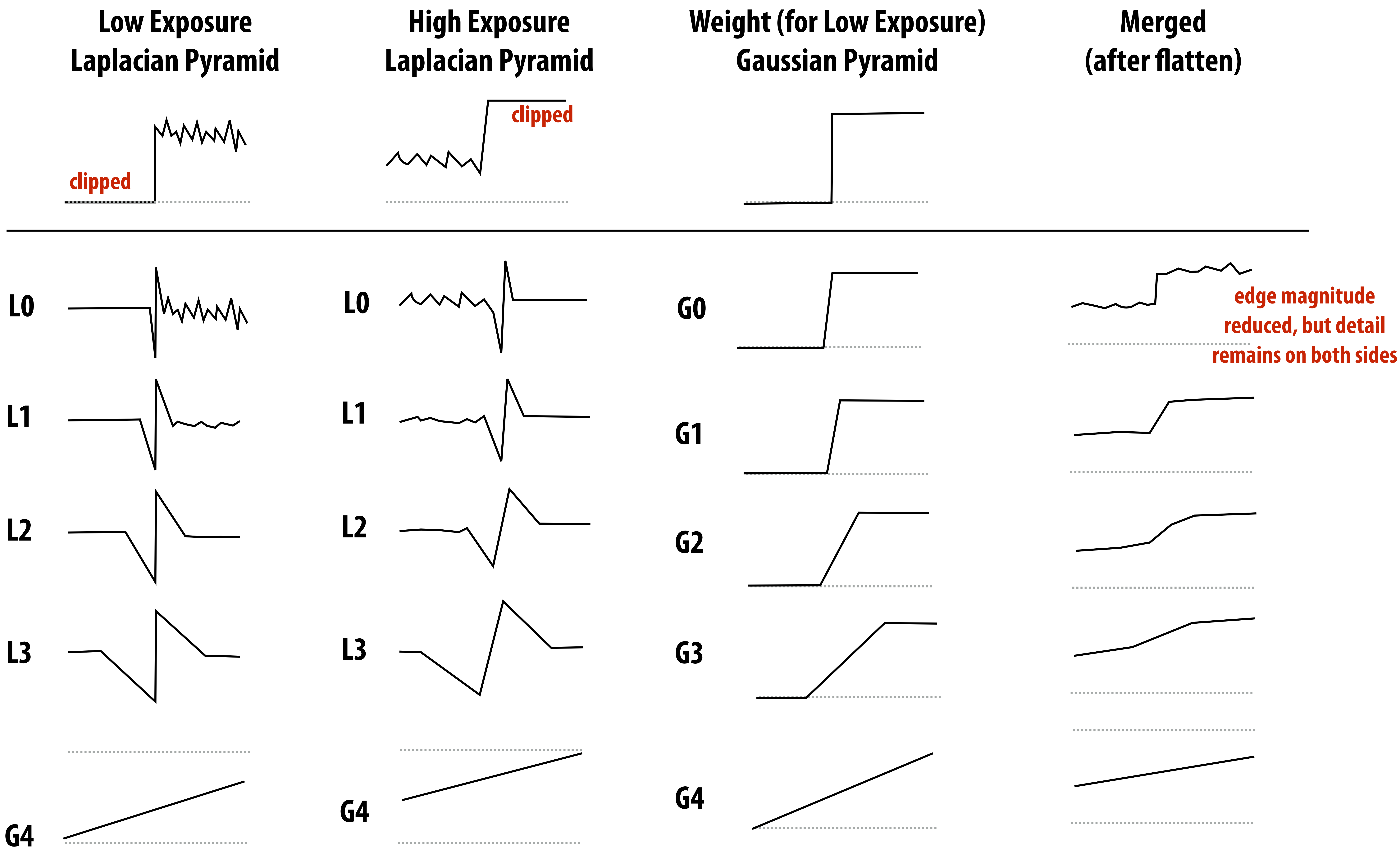

Four exposures (weights not shown)

Merged result
(after blurring weight mask)
Notice "halos" near edges

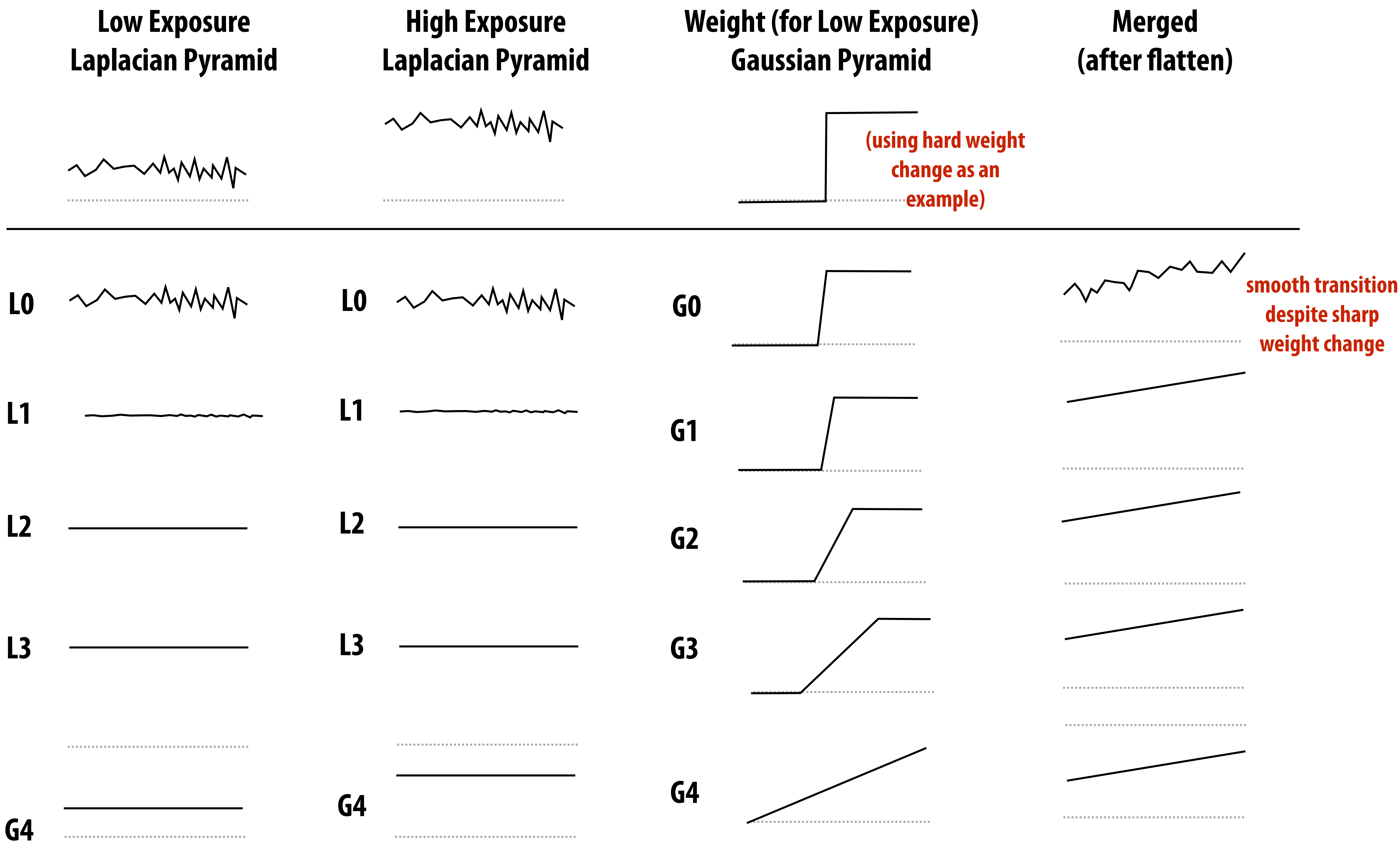
Merged result
(based on multi-resolution pyramid merge)

Why does merging Laplacian pyramids work better than merging image pixels?

Consider low and high exposures of an edge



Consider low and high exposures of flat image region



Summary: simplified image processing pipeline

- Correct pixel defects
- Align and merge (to create high signal to noise ration RAW image)
- Correct for sensor bias (using measurements of optically black pixels)
- Vignetting compensation
- White balance

(10-12 bits per pixel)
1 intensity value per pixel
Pixel values linear in energy

- Demosaic
- Denoise

3x10 bits per pixel
RGB intensity per pixel
Pixel values linear in energy

- Gamma Correction (non-linear mapping)

- Local tone mapping
- Final adjustments sharpen, fix chromatic aberrations, hue adjust, etc.

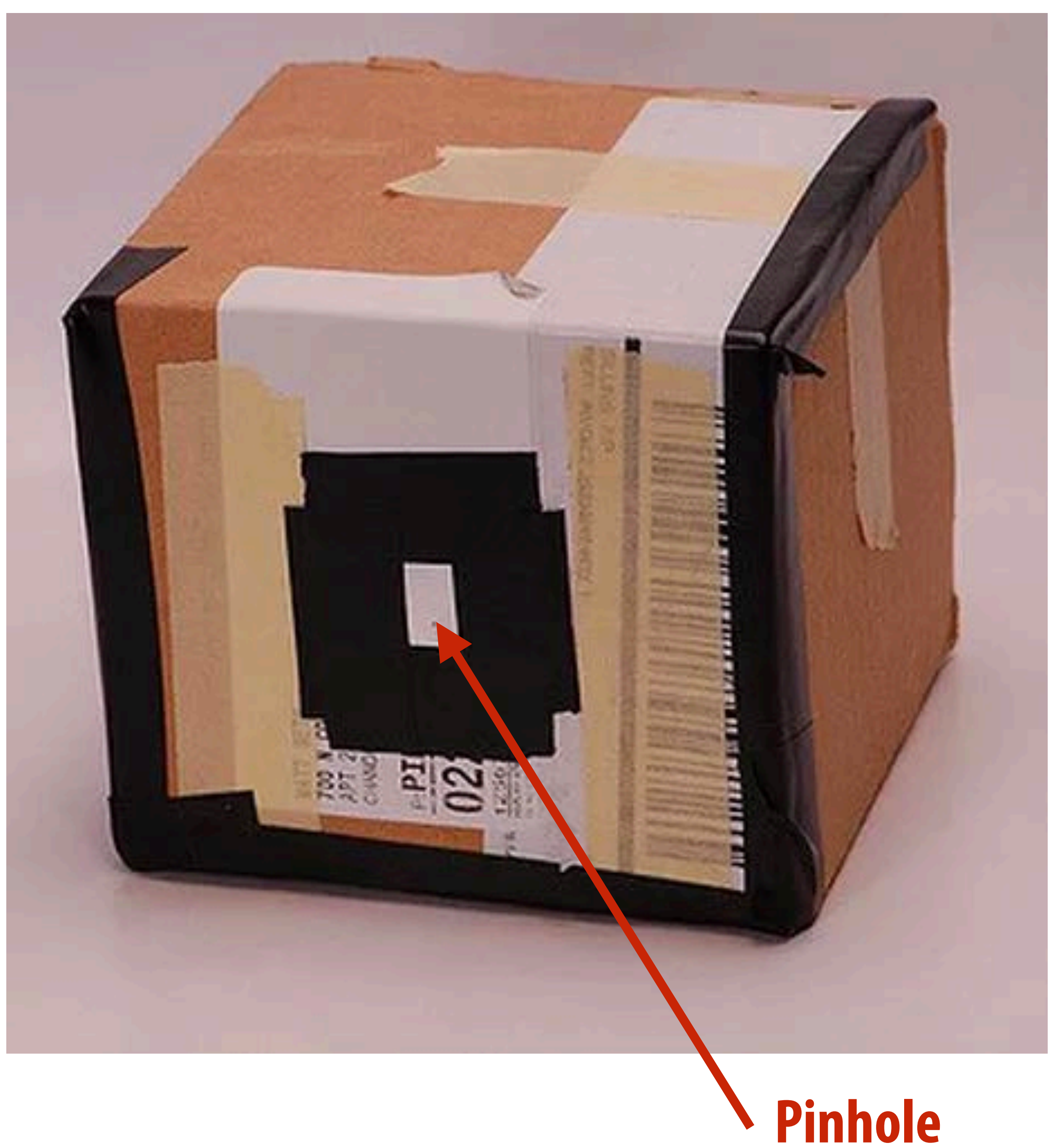
3x8-bits per pixel
Pixel values perceptually linear

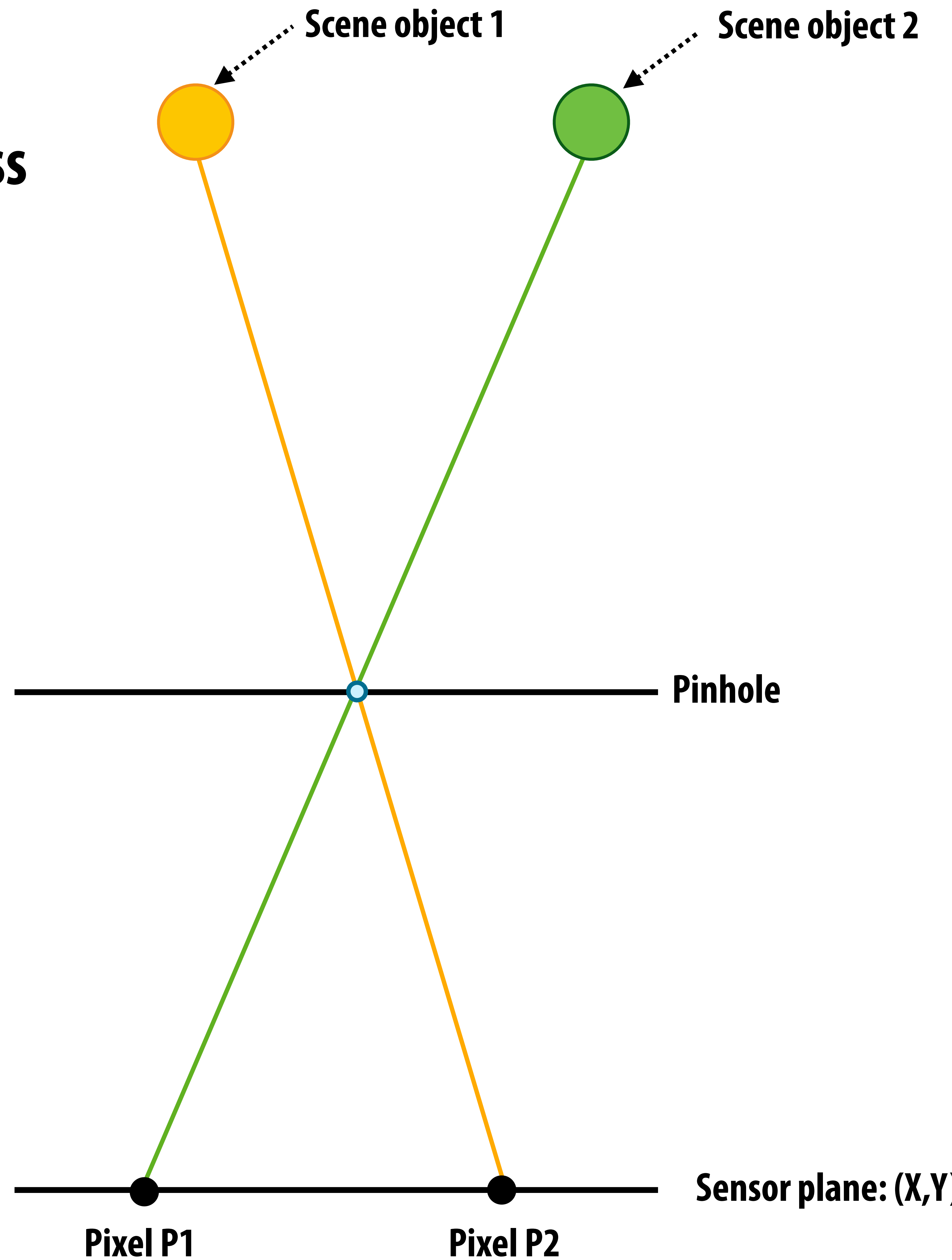
Today

Auto Focus

What does a lens do?

Recall: pinhole camera you may have made in science class (every pixel measures ray of light passing through pinhole and arriving at pixel)



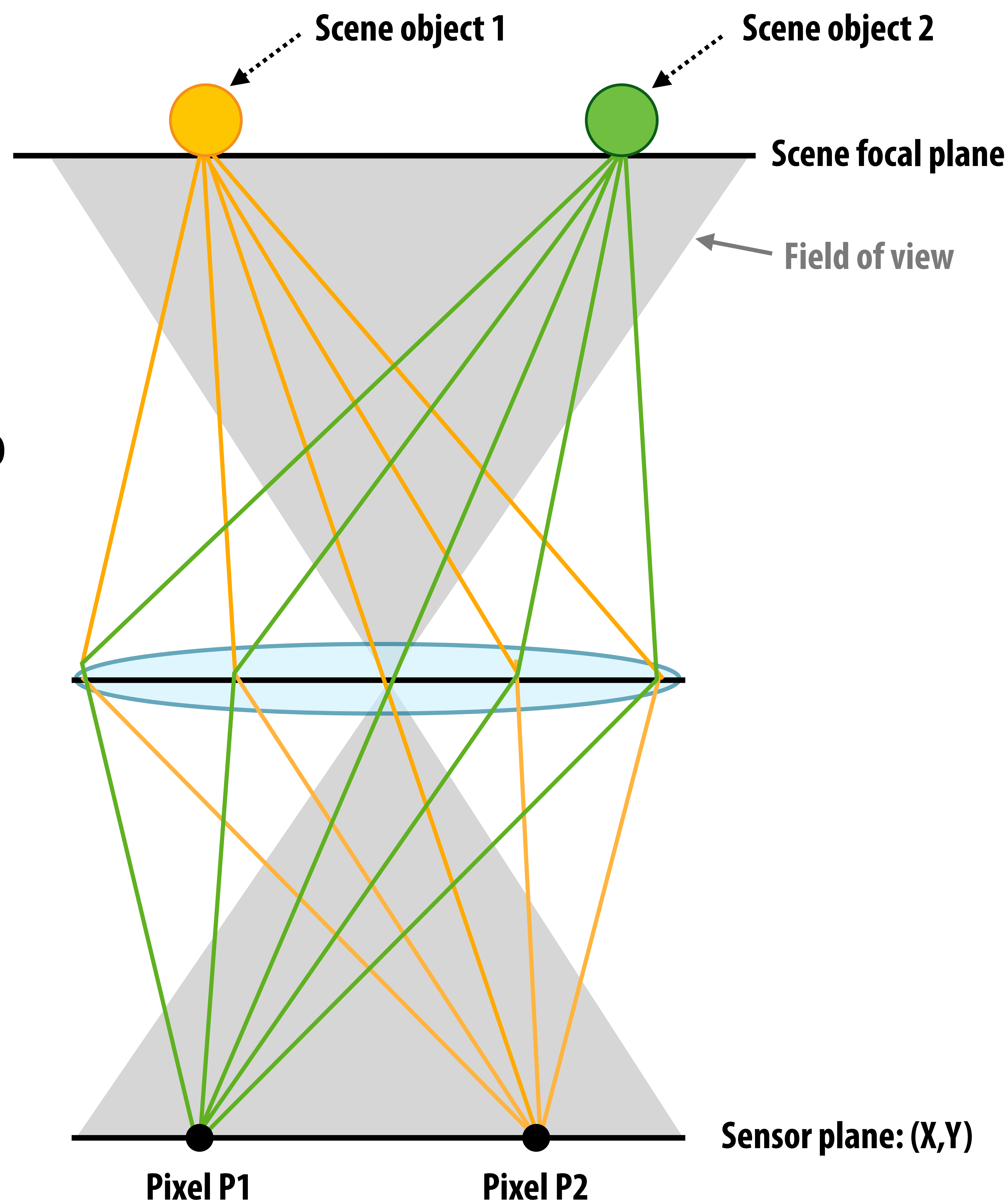


What does a lens do?

Camera with lens:

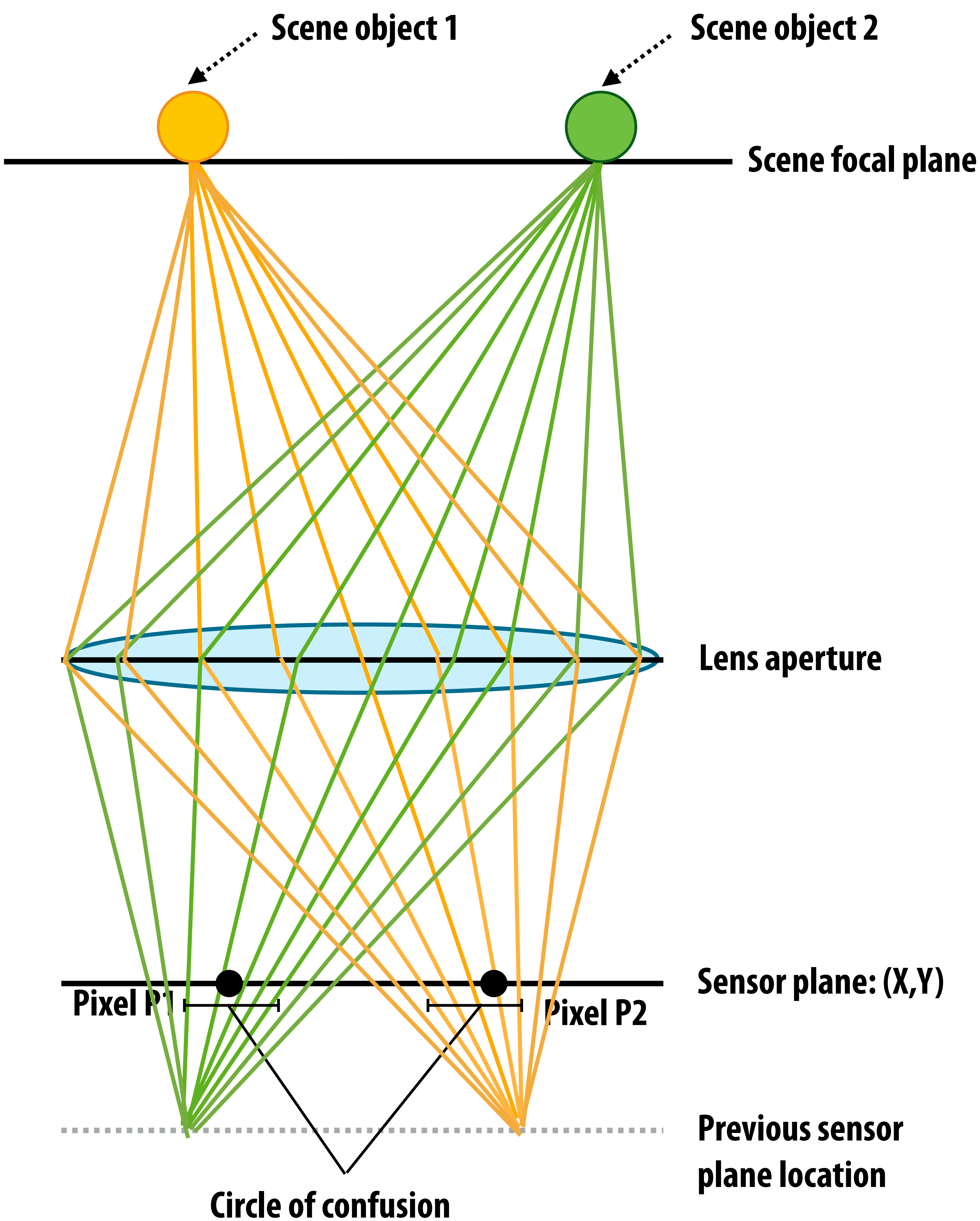
Every pixel accumulates all rays of light passing through lens aperture and refracted to location of pixel

In-focus camera: all rays of light from one point in scene arrive at one point on sensor plane



Out of focus camera

Out of focus camera: rays of light from one point in scene do not converge at point on sensor



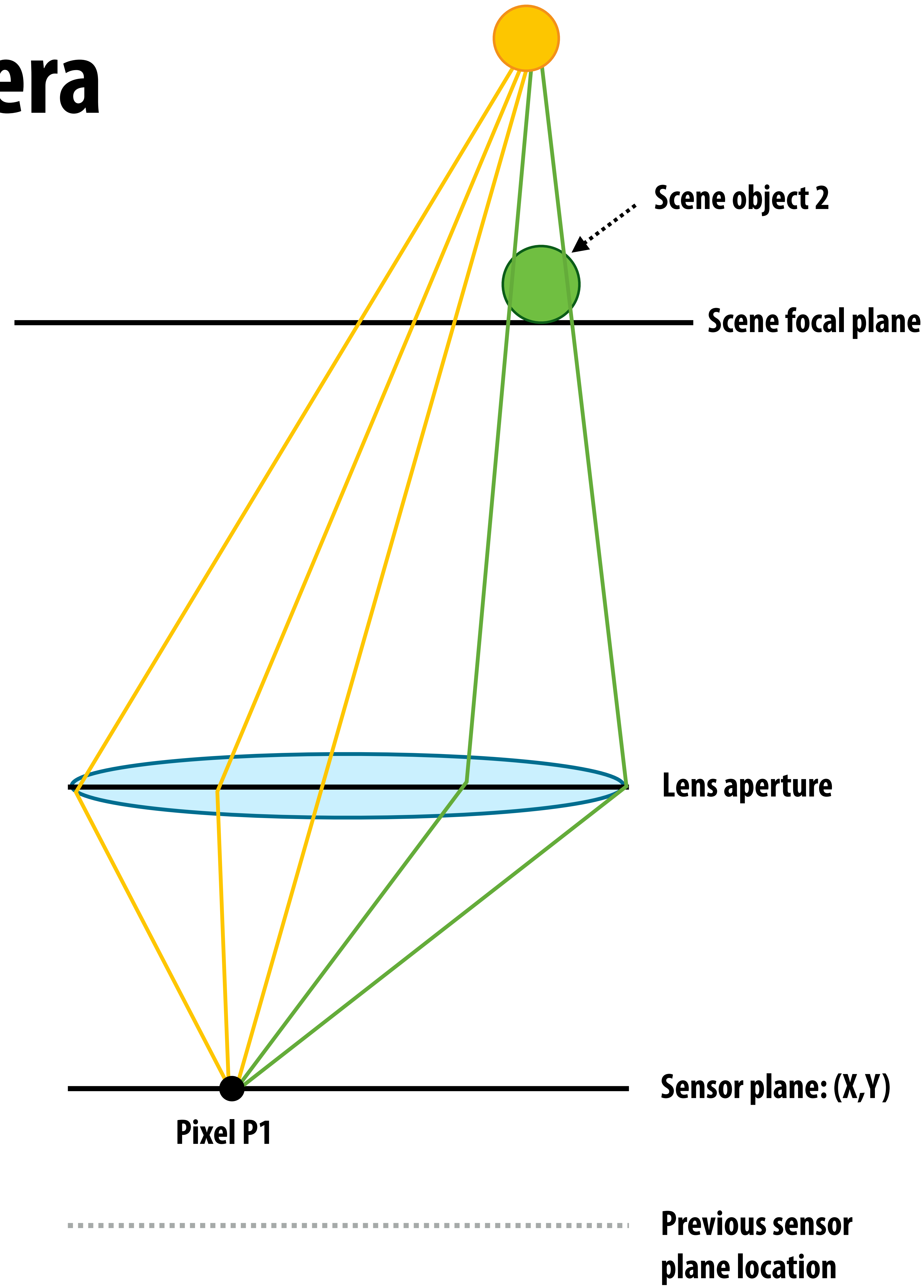
Bokeh

Out of focus camera

Out of focus camera: rays of light from one point in scene do not converge at point on sensor

=

Rays of light from different scene points converge at single point on sensor

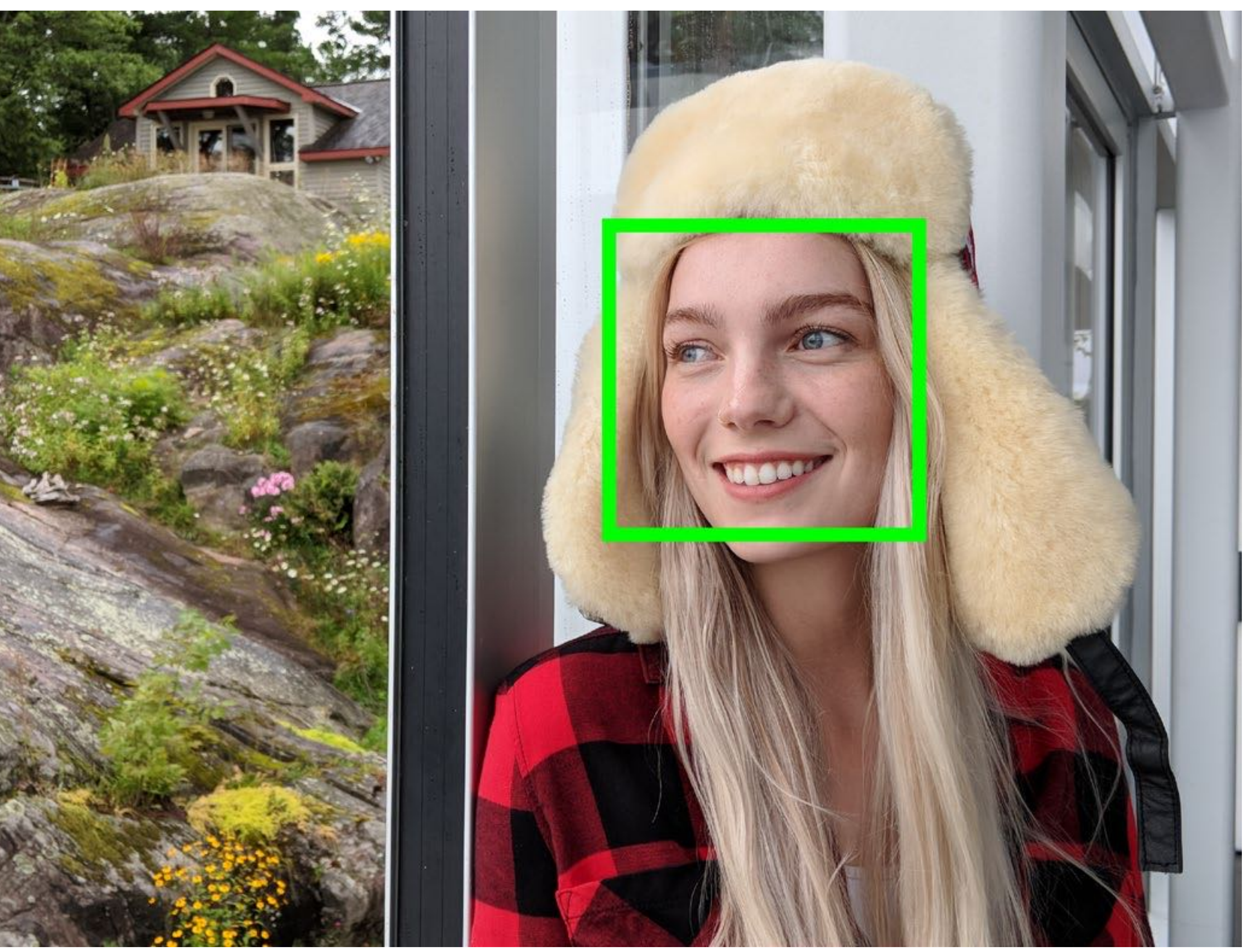


Sharp foreground / blurry background

Cell phone camera lens(es)

Portrait mode in modern smartphones

- Smart phone cameras have small apertures
  - Good: thin. lightweight lenses, often fast focus
  - Bad: cannot physically create aesthetically please photographs with nice bokeh, blurred background
- Answer: simulate behavior of large aperture lens (hallucinate image formed by large aperture lens)



Input image /w detected face

Segmentation

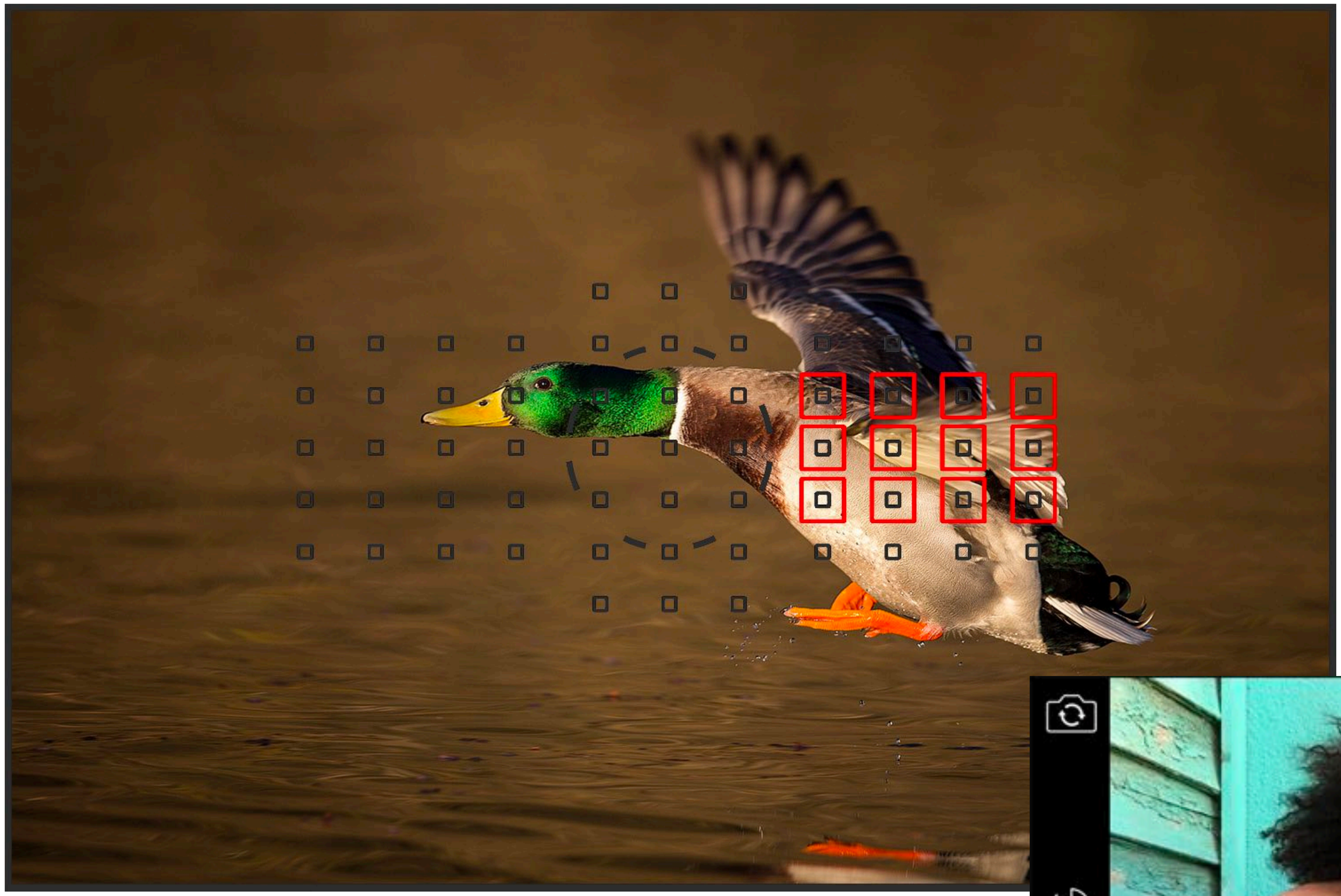
Scene Depth Estimate

Generated image (note blurred background. Blur increases with depth)

Image credit: [Wadha 2018]

Autofocus

What part of image should be in focus?



Heuristics:

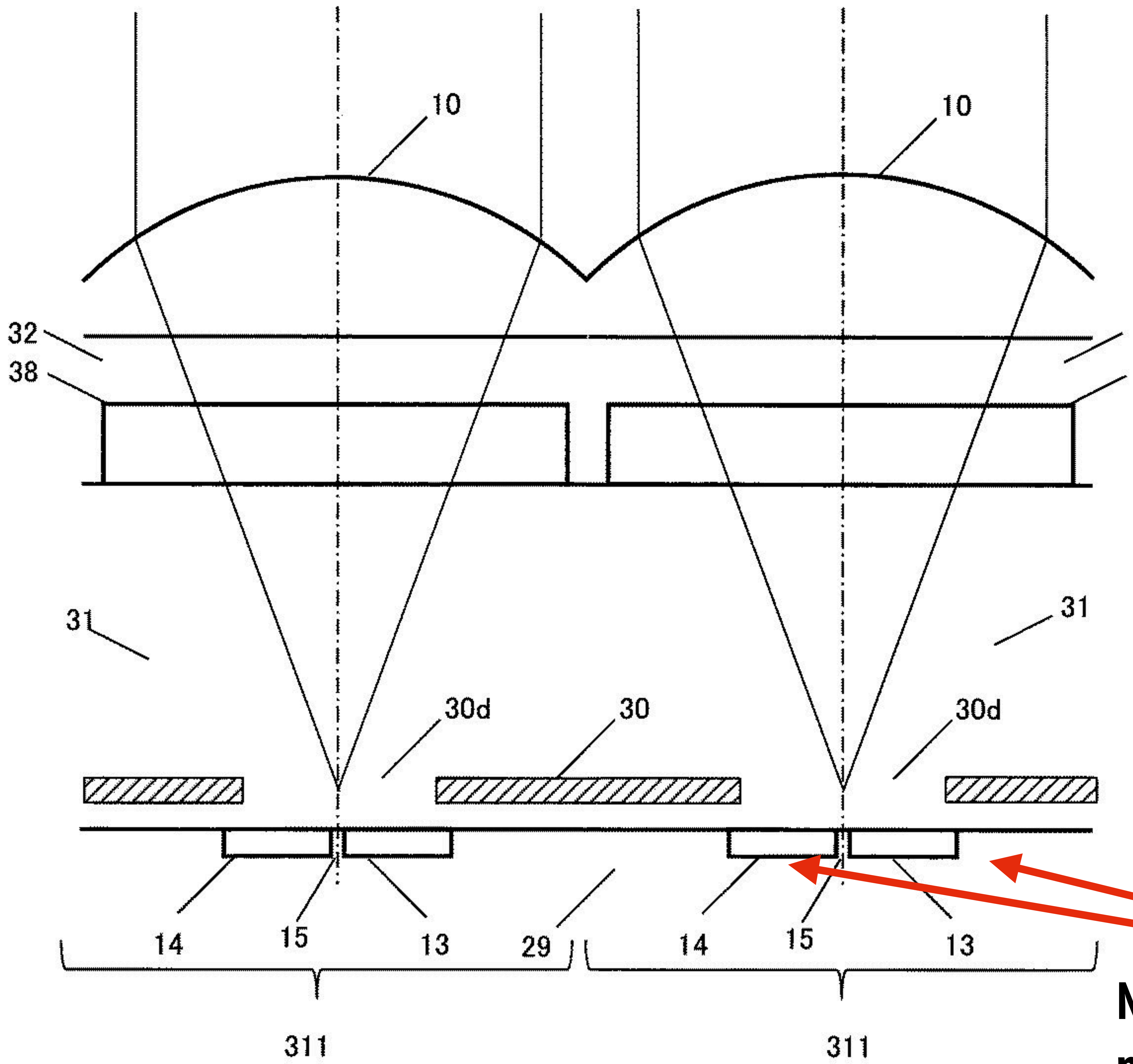
Focus on closest scene region

Put center of image in focus

Detect faces and focus on closest/largest face

Image credit: DPReview:
https://www.dpreview.com/articles/9174241280/configuring-your-5d-mark-iii-af-for-fast-action

Split pixel sensor



When both pixels have the same response, camera is in focus, why?

Now two pixels under each microlens (not one)

Image credit: Nikon

Additional sensing modalities

Apple's TrueDepth camera
(infrared dots projected by phone, captured by infrared camera)

Additional sensing modalities

Fuse information from all modalities to obtain best estimate of depth

iPhone Xr depth estimate with lights ON in room

iPhone Xr depth estimate with lights OFF in room

Image credit: https://blog.halide.cam/iphone-xr-a-deep-dive-into-depth-47d36ae69a81

Summary

Summary

- Computation now a fundamental part of producing a pleasing photograph
- Used to compensate for physical constraints (demosaic, denoise, lens corrections)
- Used to analyze image to guess system parameters (focus, exposure), or scene contents (white balance, portrait mode)
- Used to make non-physically plausible images that have aesthetic merit

Sensor output ("RAW") → **Computation** →

Beautiful image that impresses your friends on Instagram

Image processing workload characteristics

- "Pointwise" operations
  - output_pixel = f(input_pixel)
- "Stencil" computations (e.g., convolution, demosaic, etc.)
  - Output pixel (x,y) depends on <u>fixed-size</u> local region of input around (x,y)
- Lookup tables
  - e.g., contrast s-curve
- Multi-resolution operations (upsampling/downsampling)
- Fast-fourier transform
  - We didn't talk about Fourier domain techniques in class (but Hasinoff 16 reading has many examples)
- Long pipelines of these operations

Upcoming classes: efficiently mapping these workloads to modern processors