Lecture 3:

The Camera Image Processing Pipeline
(part 2: tone mapping and autofocus)

Visual Computing Systems
Stanford CS348K, Fall 2018

Acknowledgement to Marc Levoy (Stanford/Google) for various slides used in this lecture.

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
- 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
- Gamma Correction (non-linear mapping)

3x12 bits per pixel
RGB intensity per pixel
Pixel values linear in energy

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

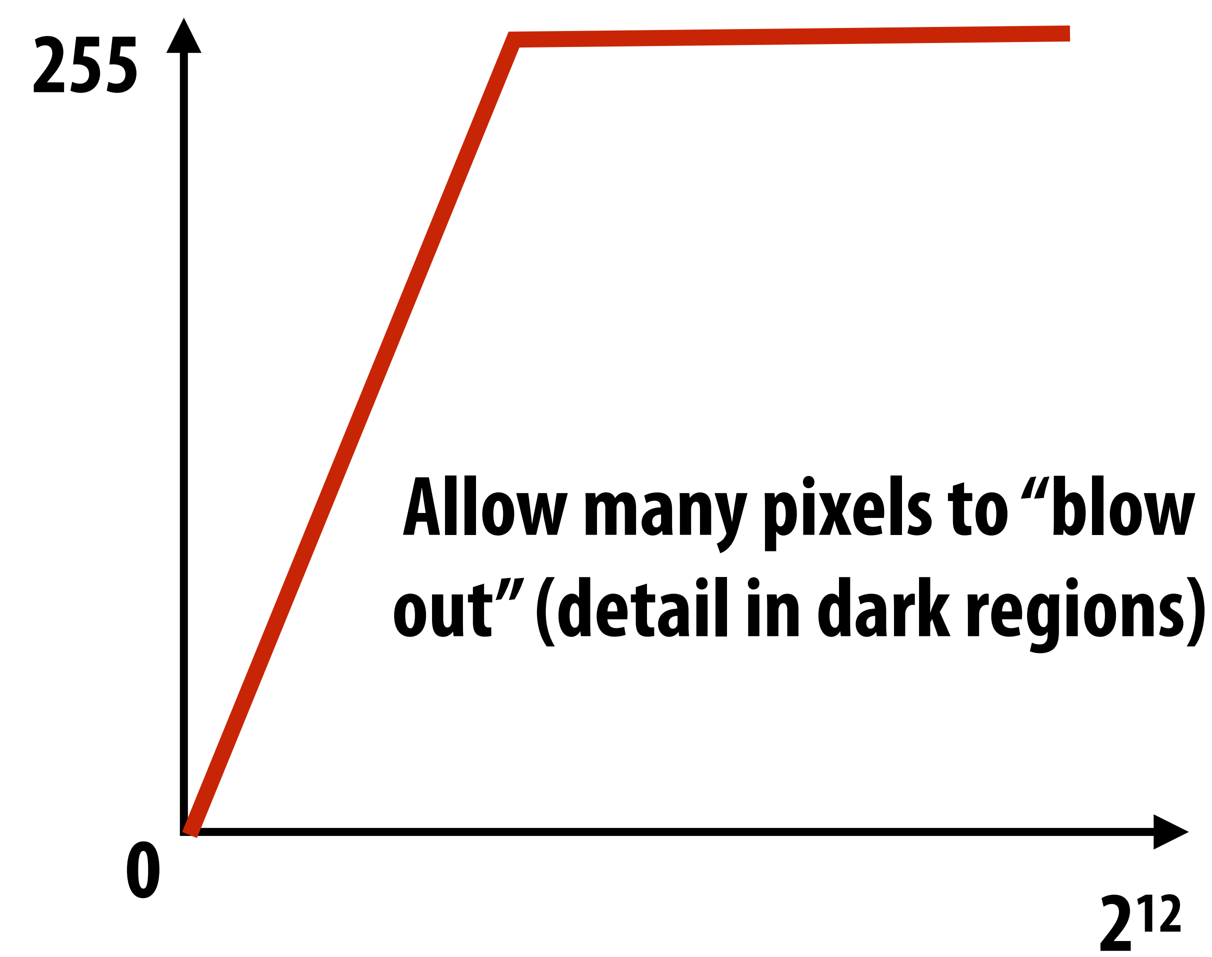
3x8-bits per pixel
Pixel values perceptually linear

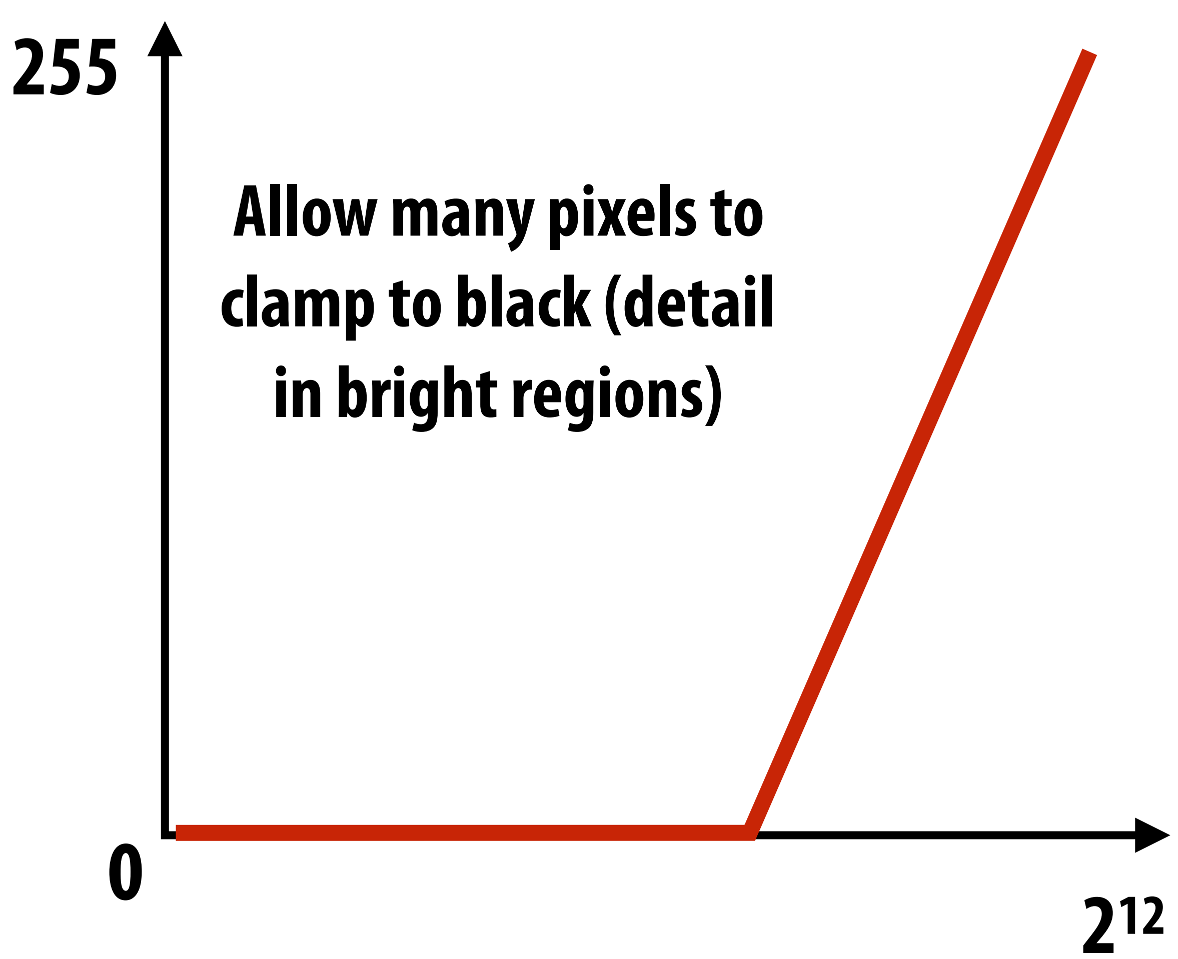
Auto Exposure and Tone Mapping

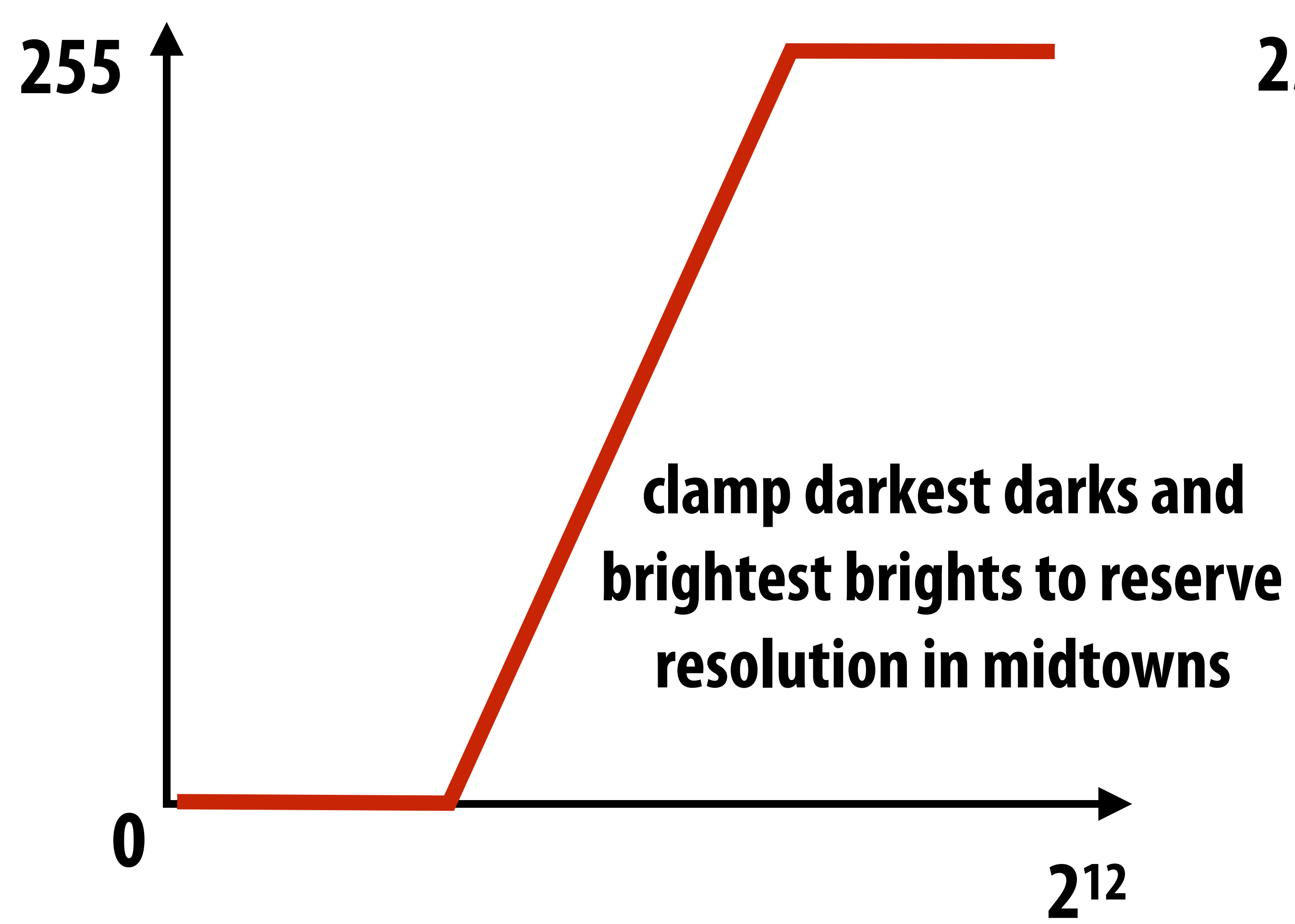
Global tone mapping

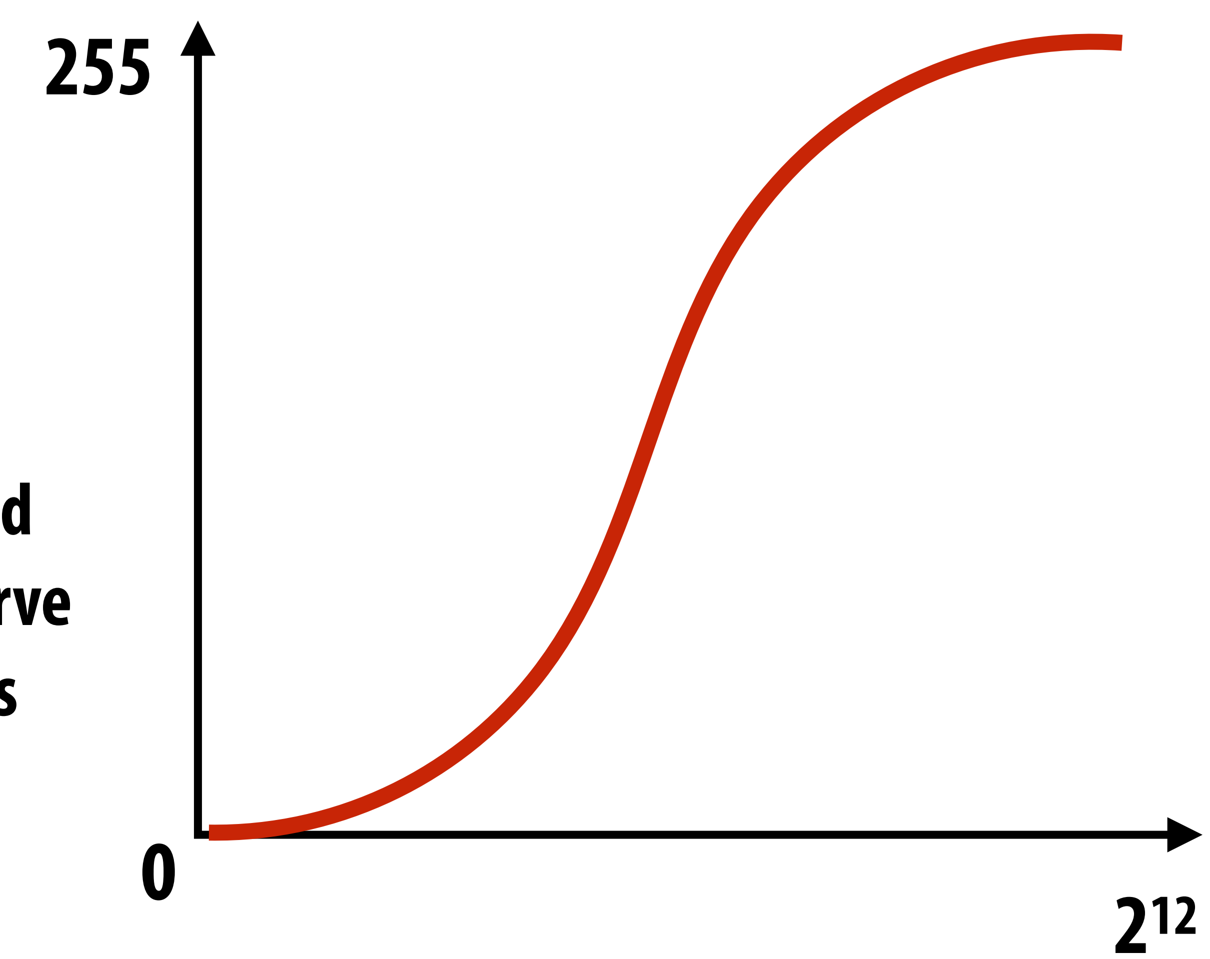
- Measured image values: 10-12 bits / pixel, but common image formats (8-bits/ pixel)
- How to convert 12 bit number to 8 bit number?

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

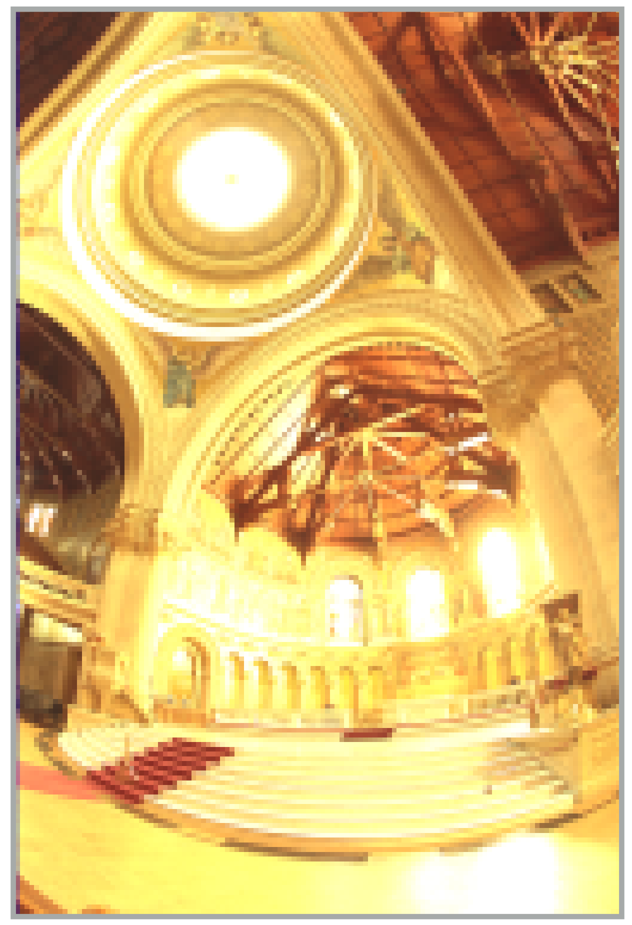


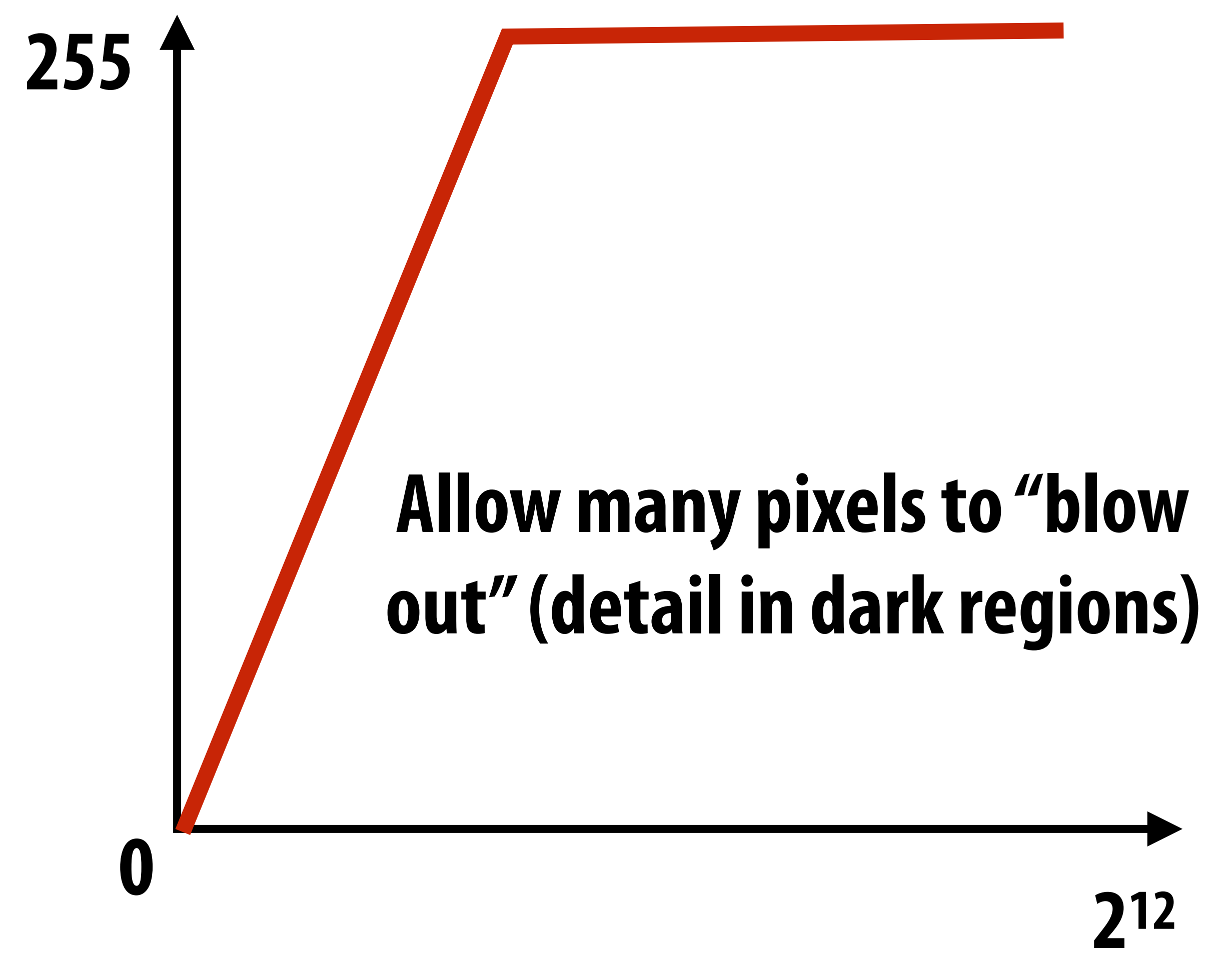


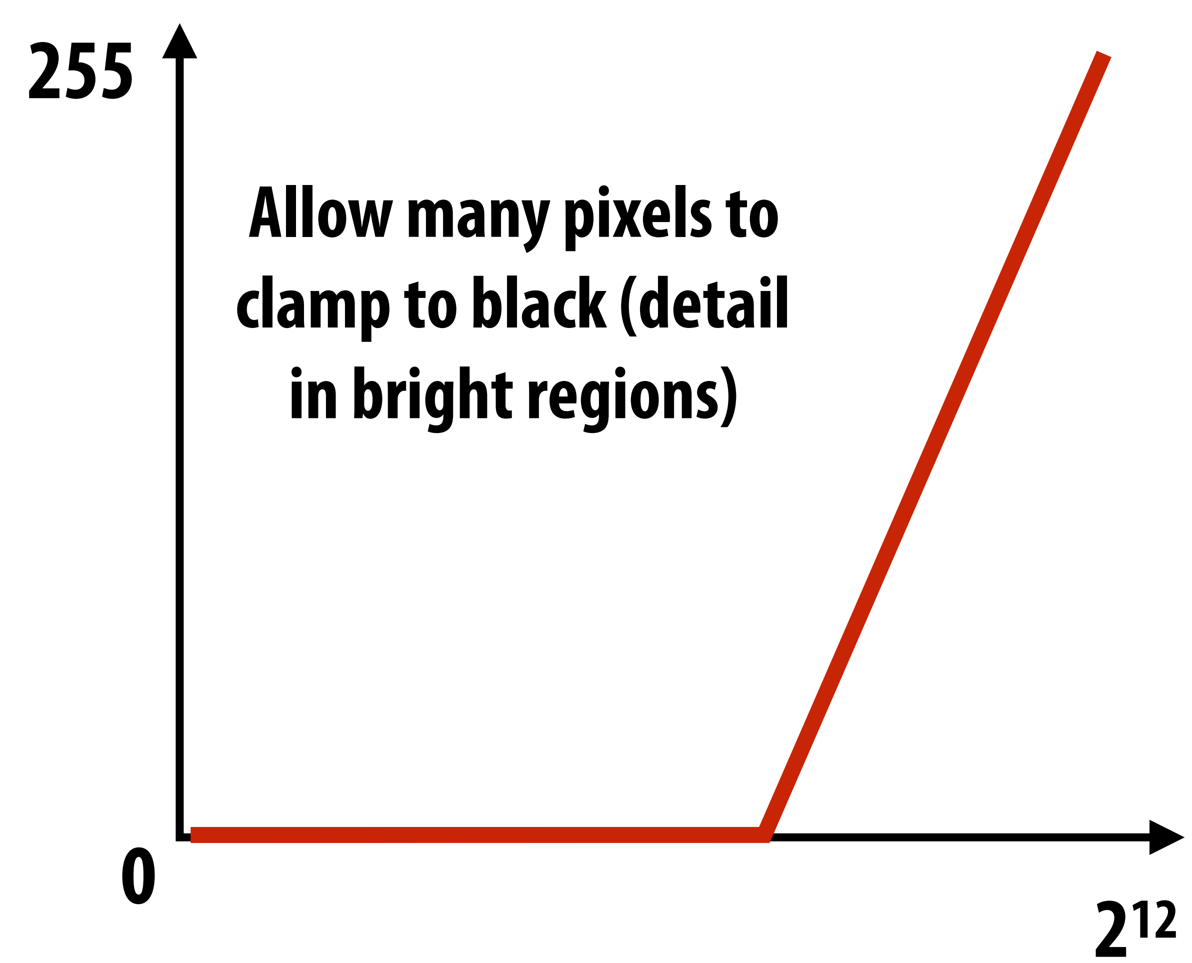




Global tone mapping

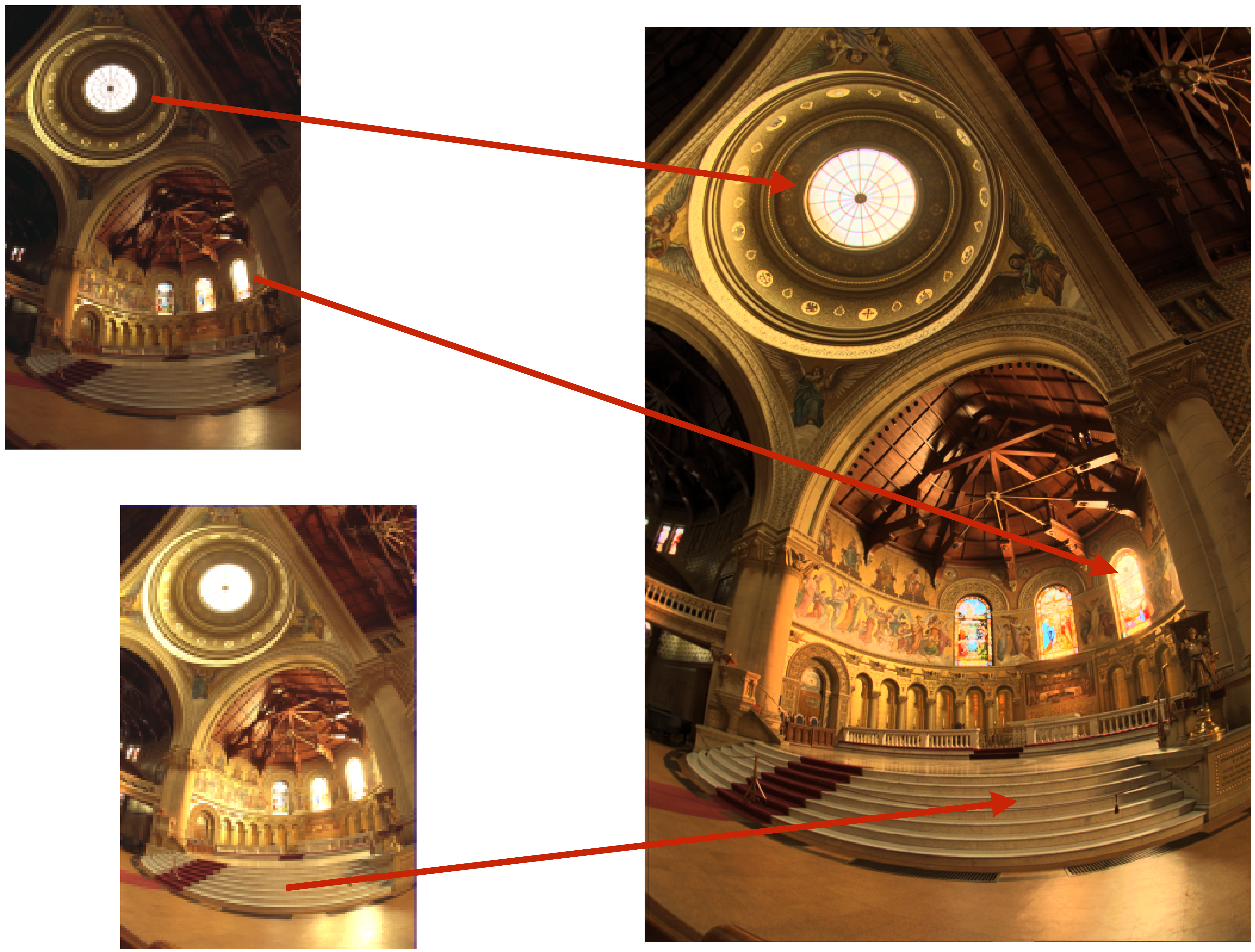






Local tone mapping

- 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

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

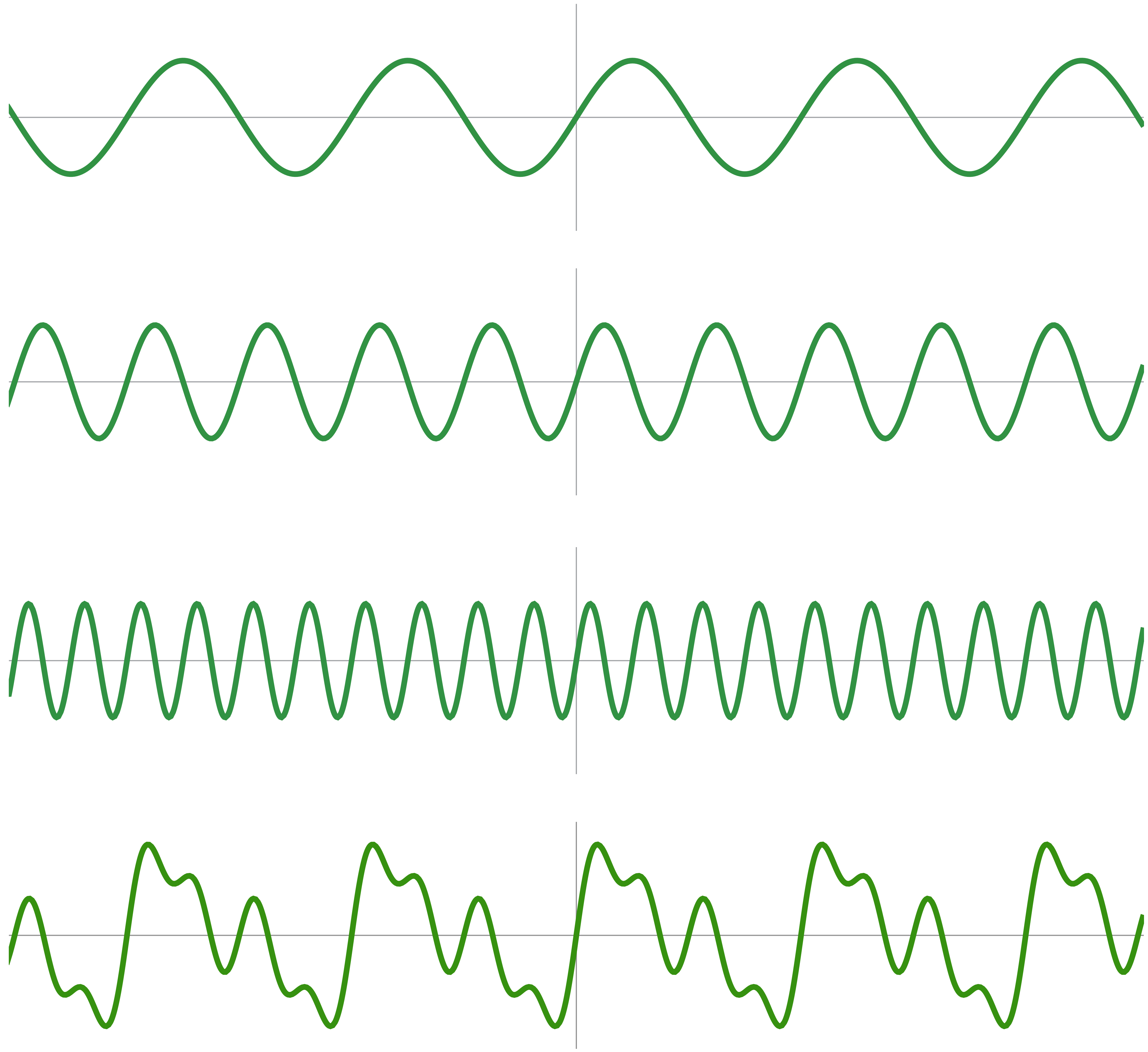
Representing sound as a superposition of frequencies

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

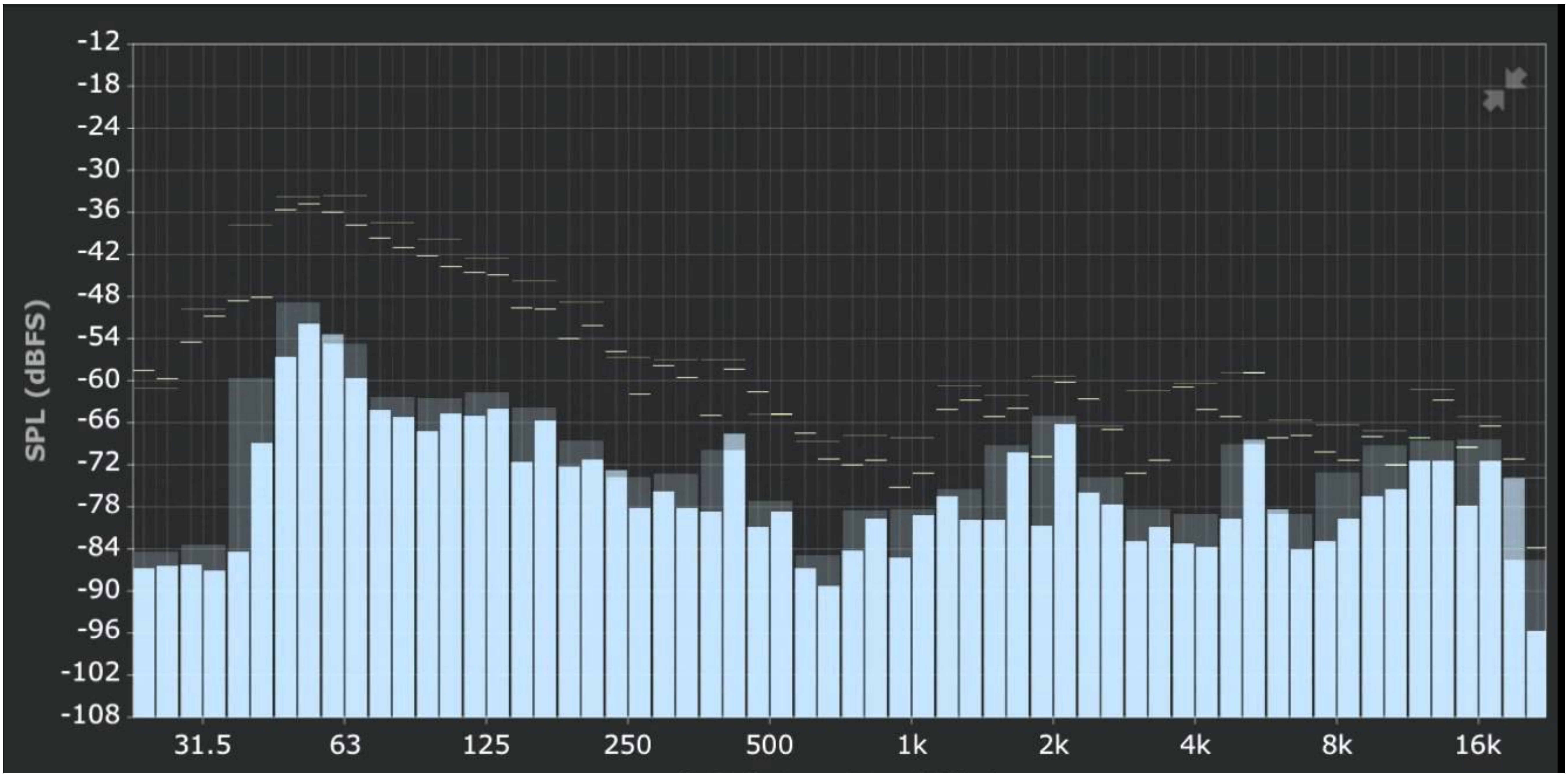
$f_2(x) = sin(2\pi x)$

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

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



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**

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

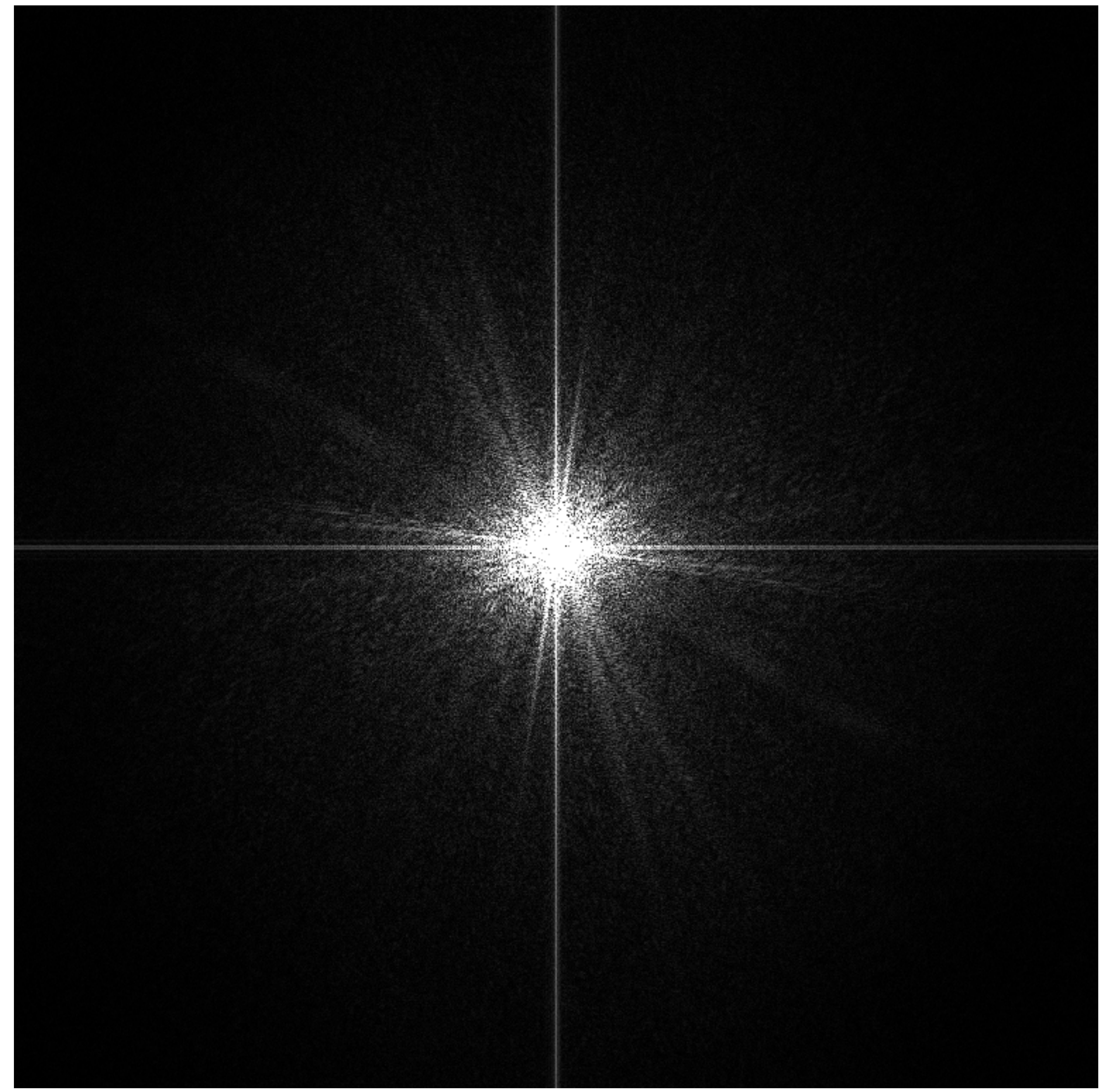
$$= \int_{-\infty}^{\infty} f(x)(\cos(2\pi\xi x) - i\sin(2\pi\xi x))dx$$

- **2D form:**

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

Visualizing the frequency content of images

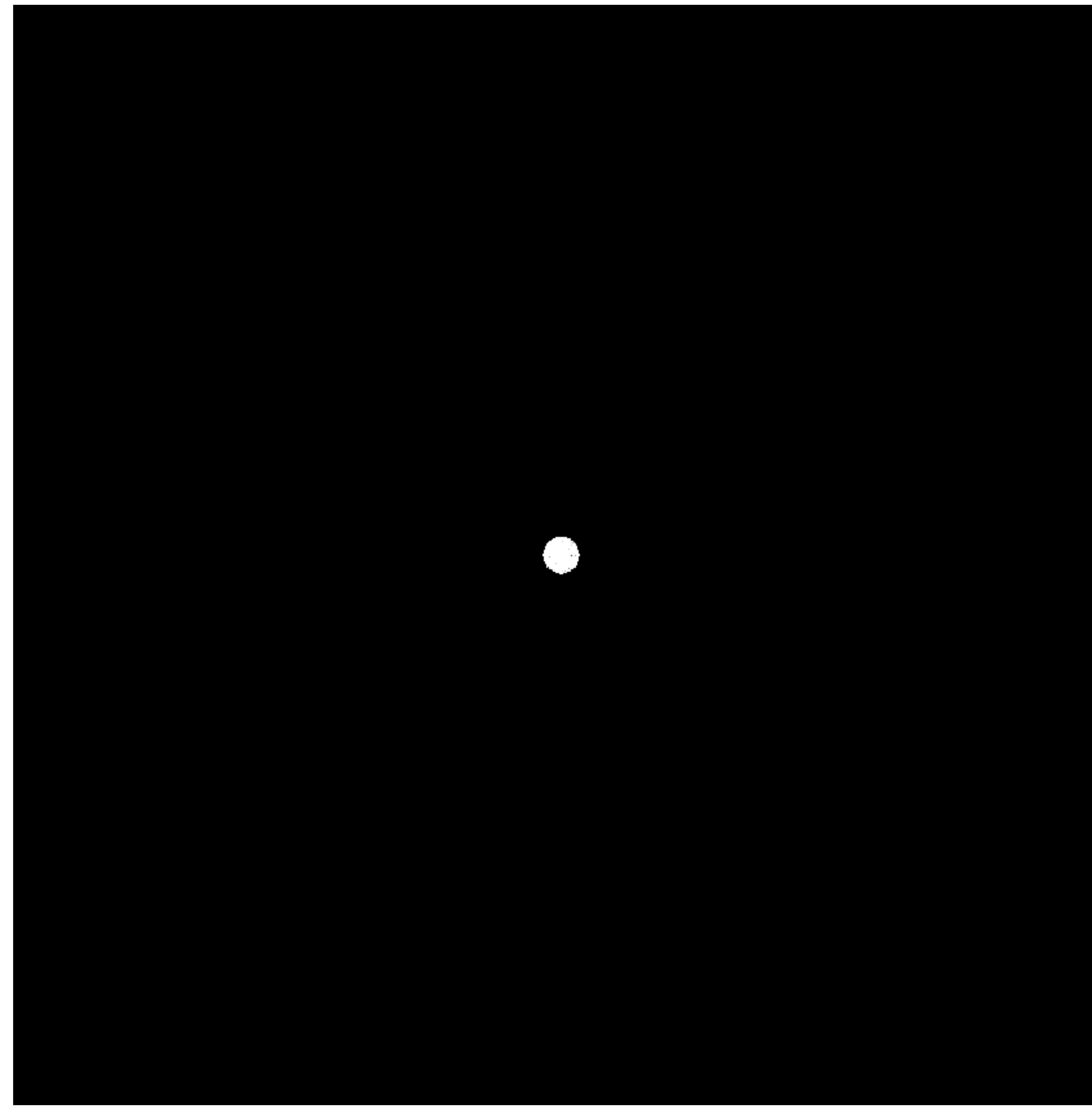
Spatial domain result



Spectrum

Low frequencies only (smooth gradients)

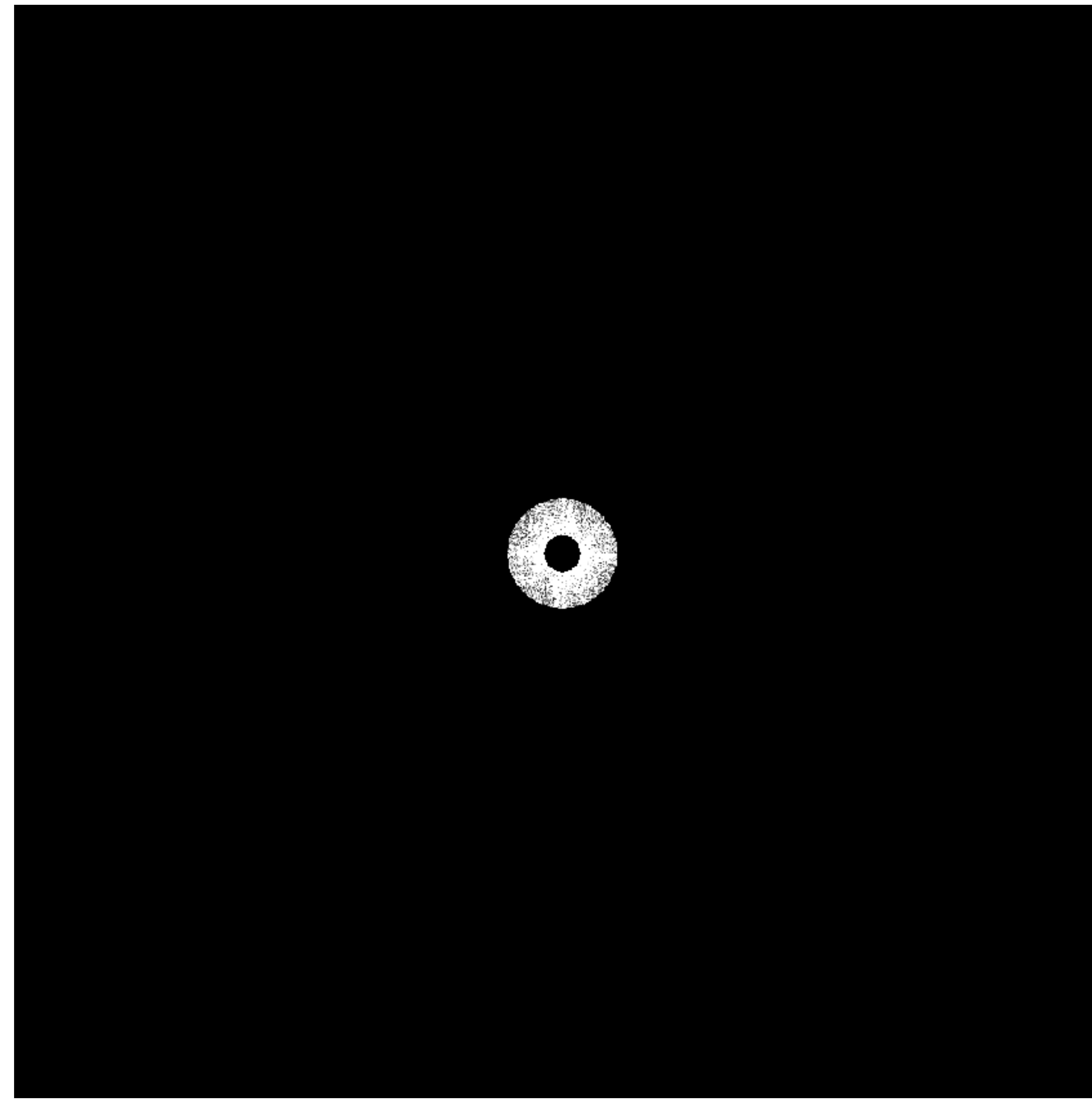
Spatial domain result



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

Mid-range frequencies

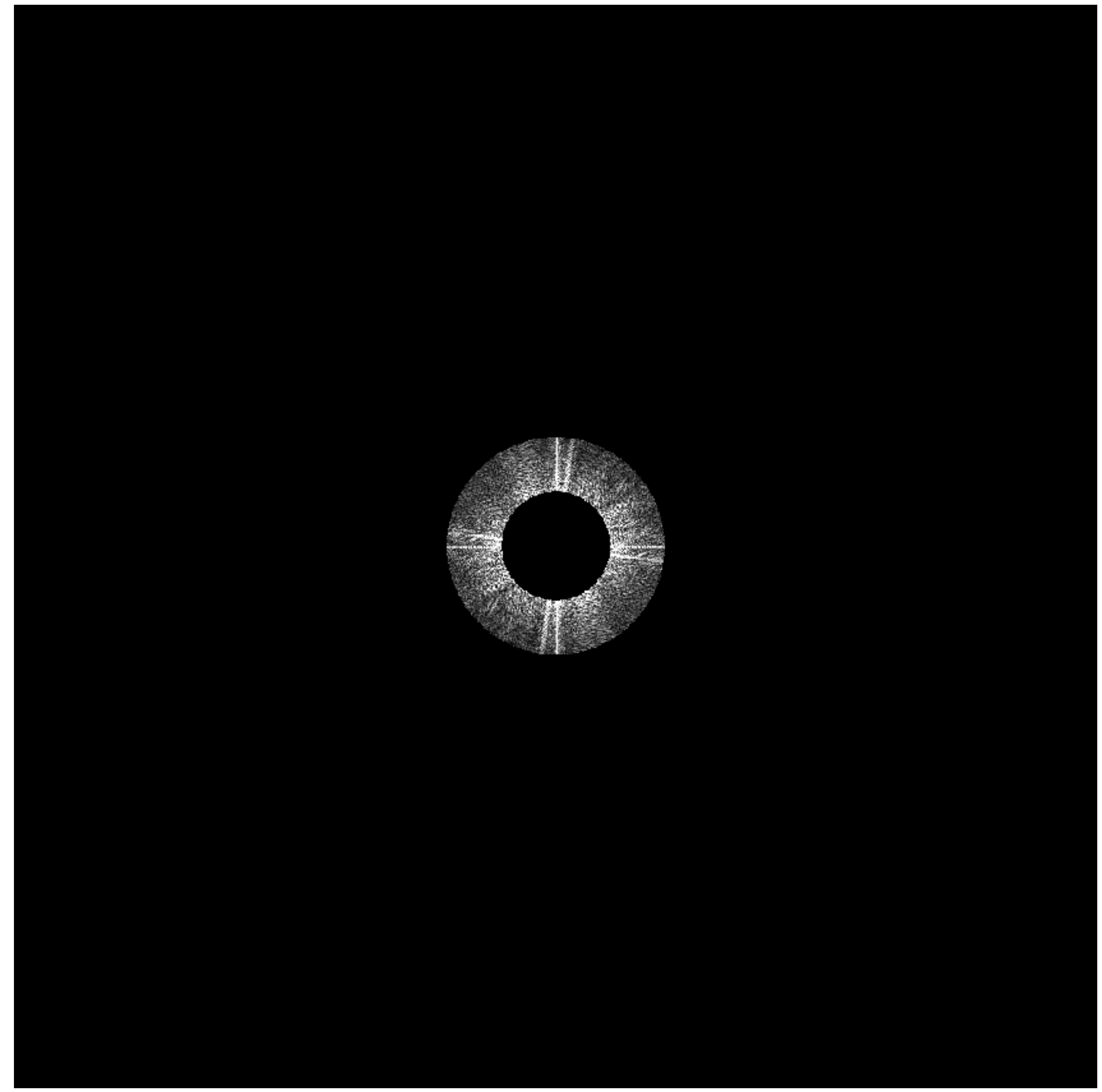
Spatial domain result



Spectrum (after band-pass filter)

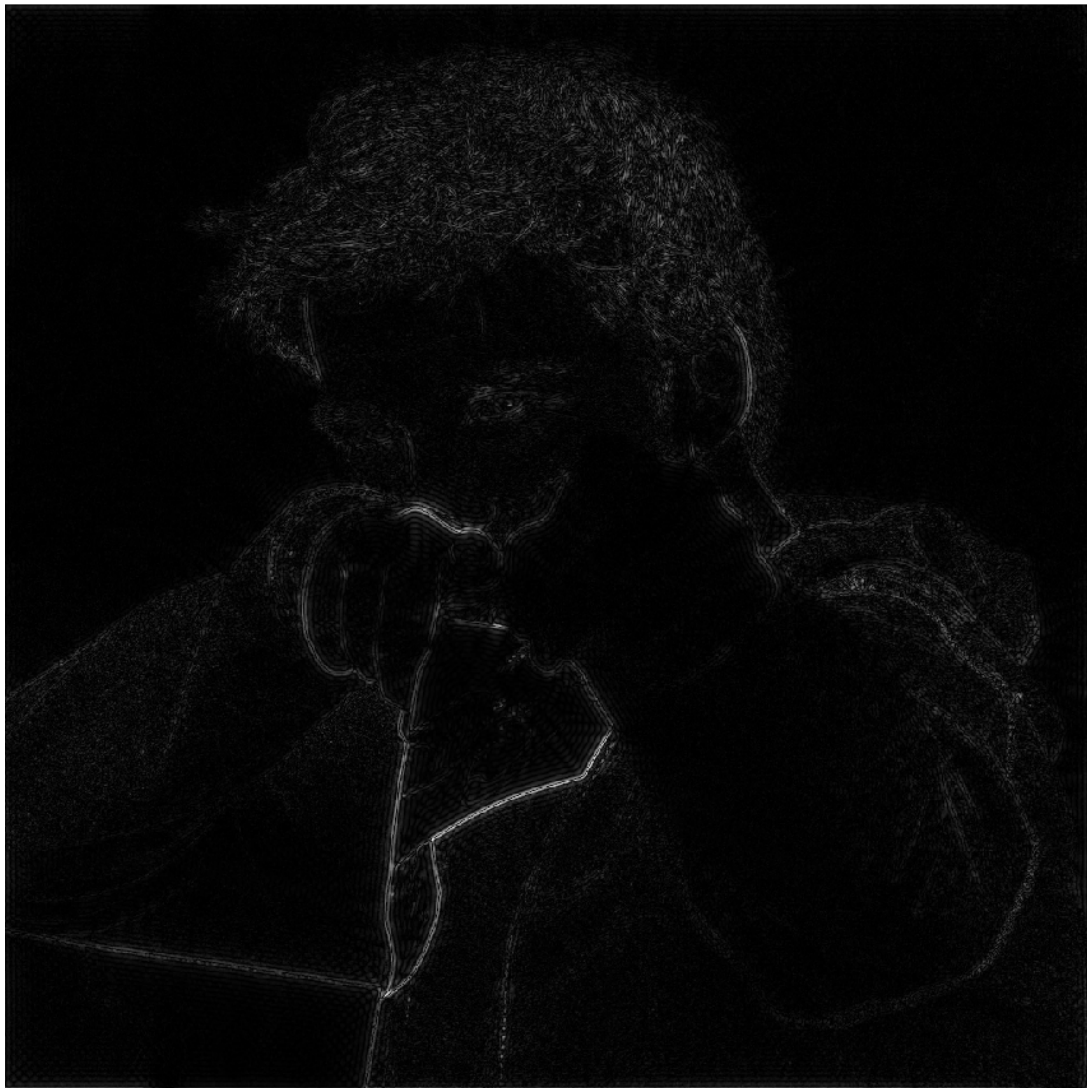
Mid-range frequencies

Spatial domain result

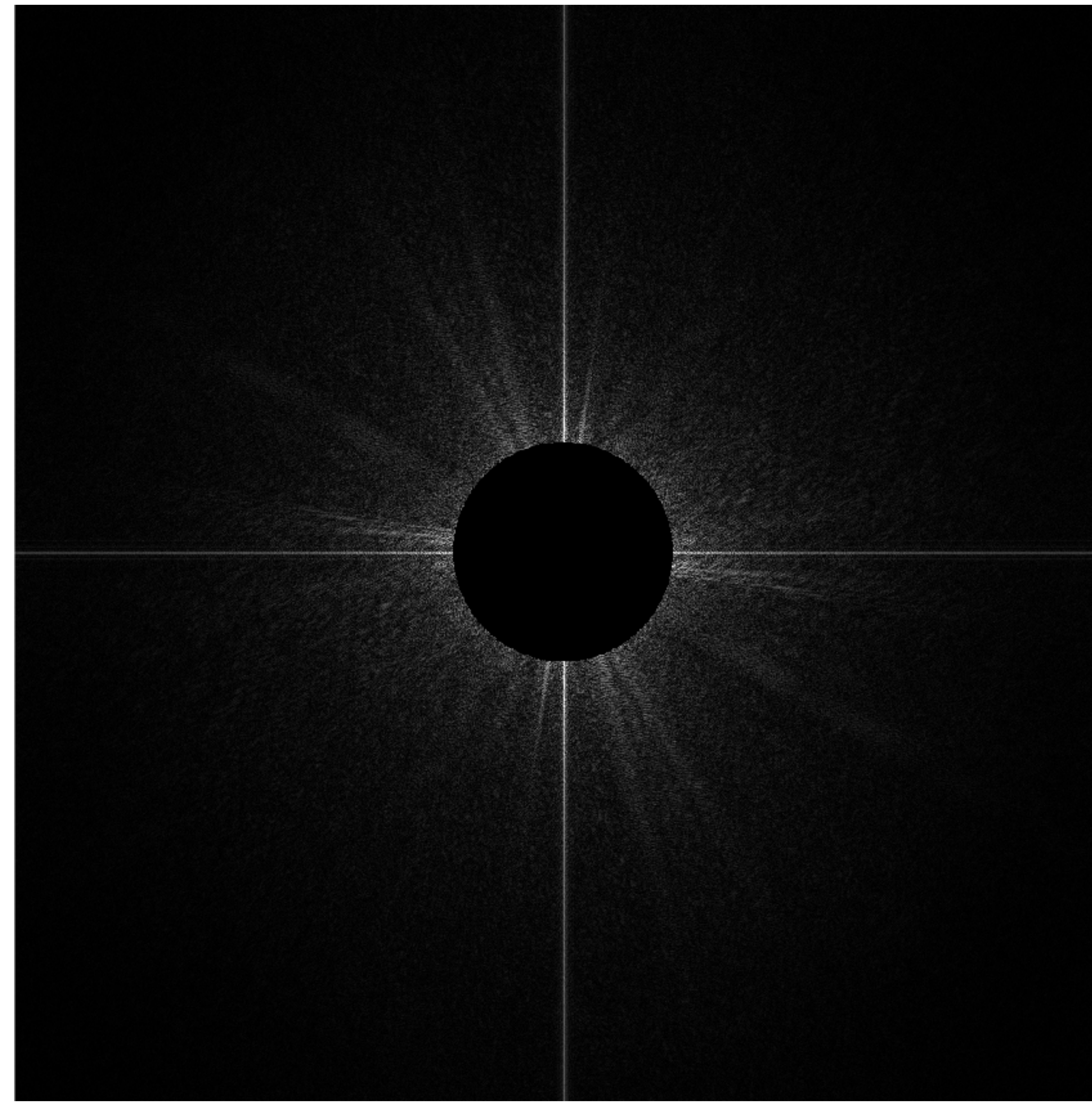


Spectrum (after band-pass filter)

High frequencies (edges)

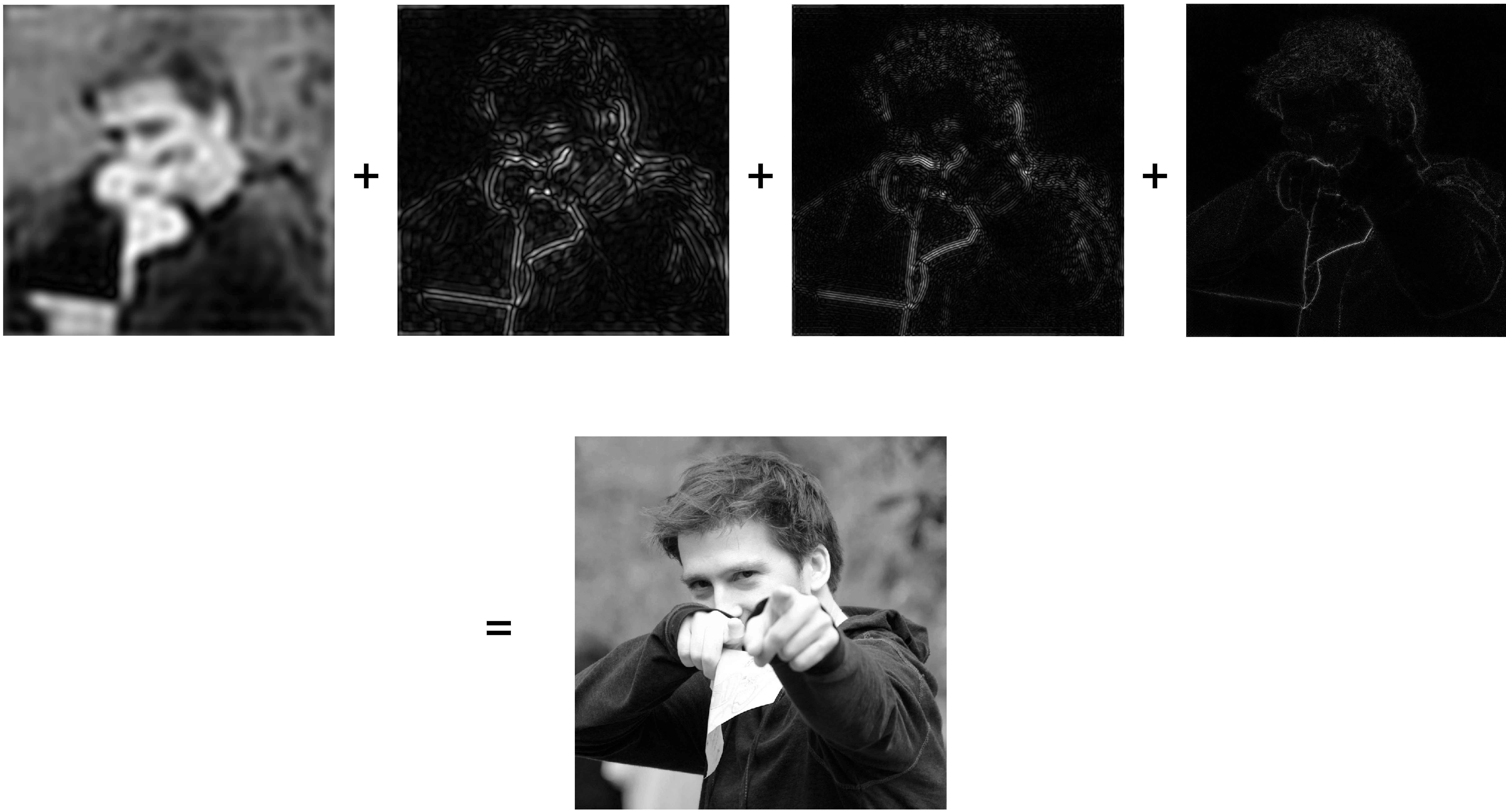


Spatial domain result
(strongest edges)



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

An image as a sum of its frequency components



Another (linear) sharpening filter

$$\text{blurred} = g * I$$

$$\text{fine} = I - \text{blurred}$$ ← **Extract high frequencies**

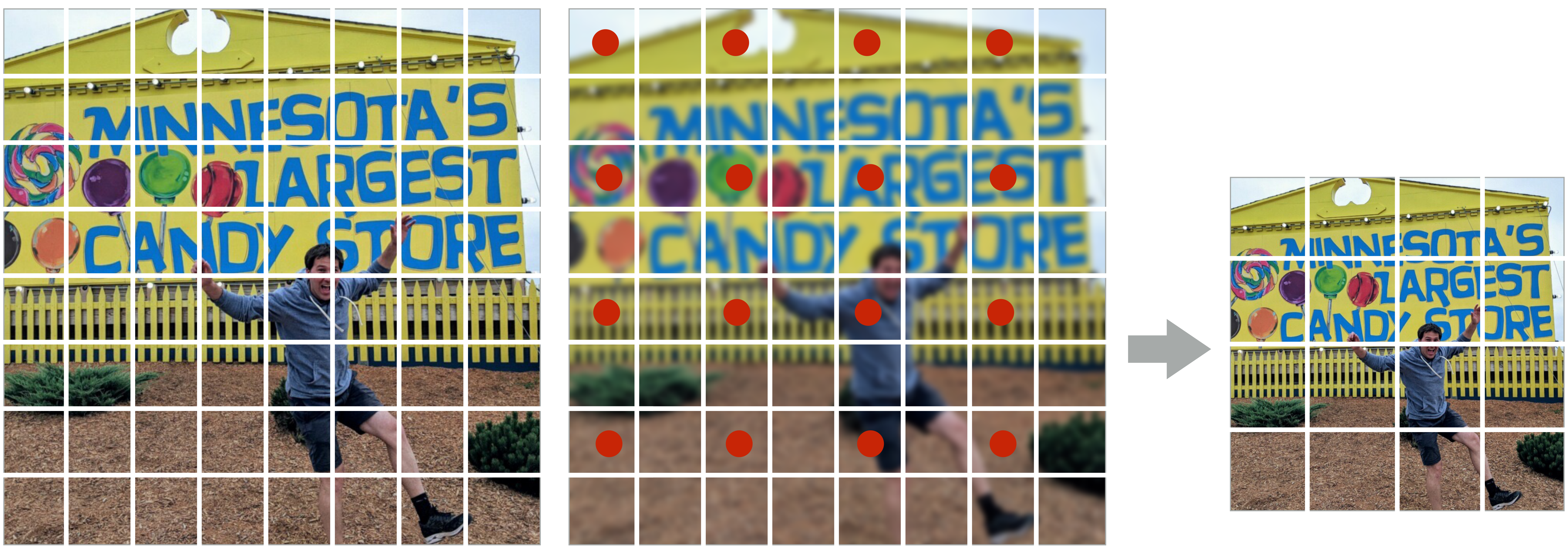
$$\text{sharpened} = I + 0.5 \times \text{fine}$$ ← **Boost high frequencies**

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
- 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)**

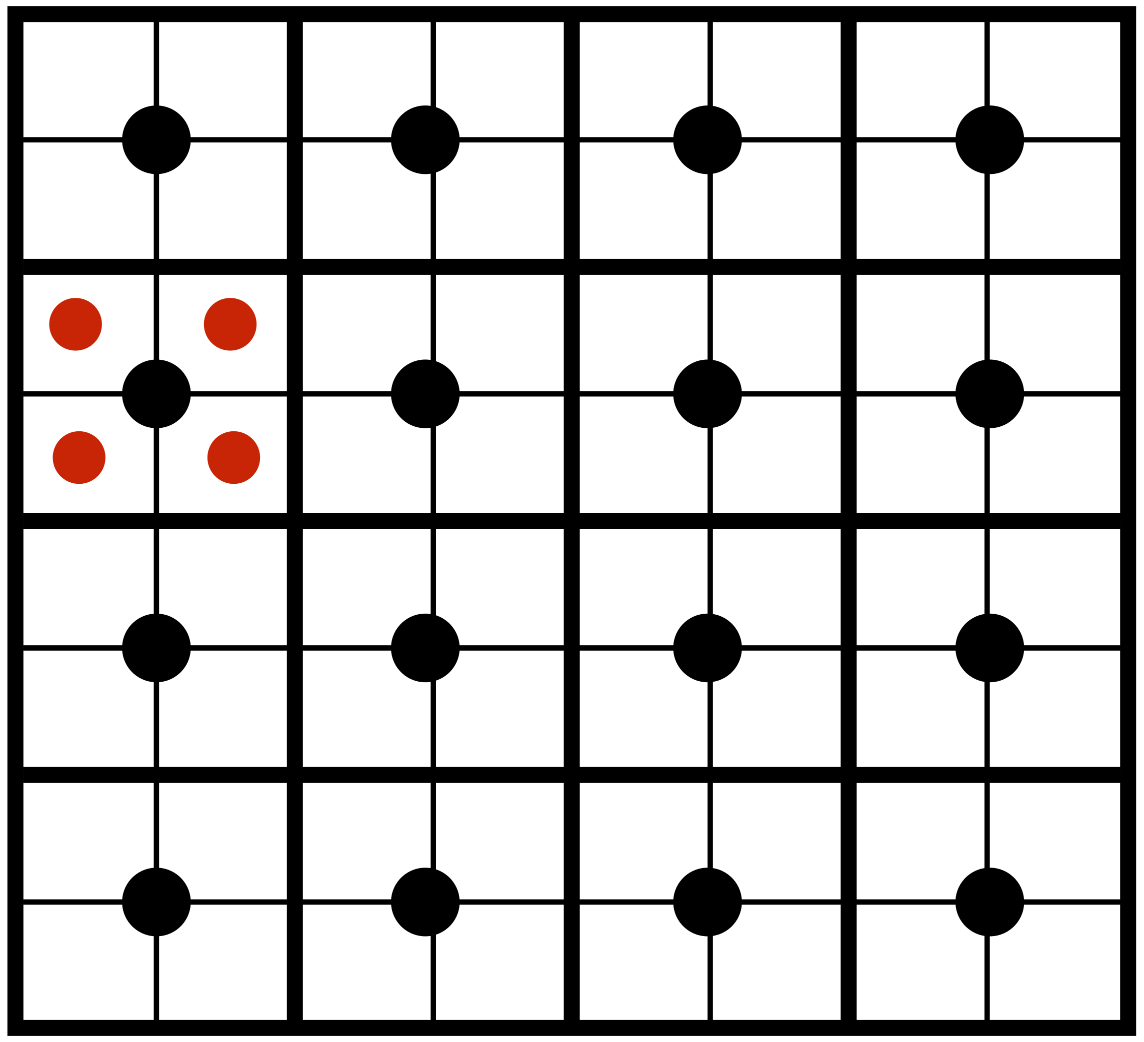
```
float input[(WIDTH+2) * (HEIGHT+2)];
float output[WIDTH/2 * HEIGHT/2];

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<3; jj++)
         for (int ii=0; ii<3; 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 = 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

G_0

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

$L_0 = G_0 - \text{up}(G_1)$

Each level in Laplacian pyramid represents increasingly high frequency content from image

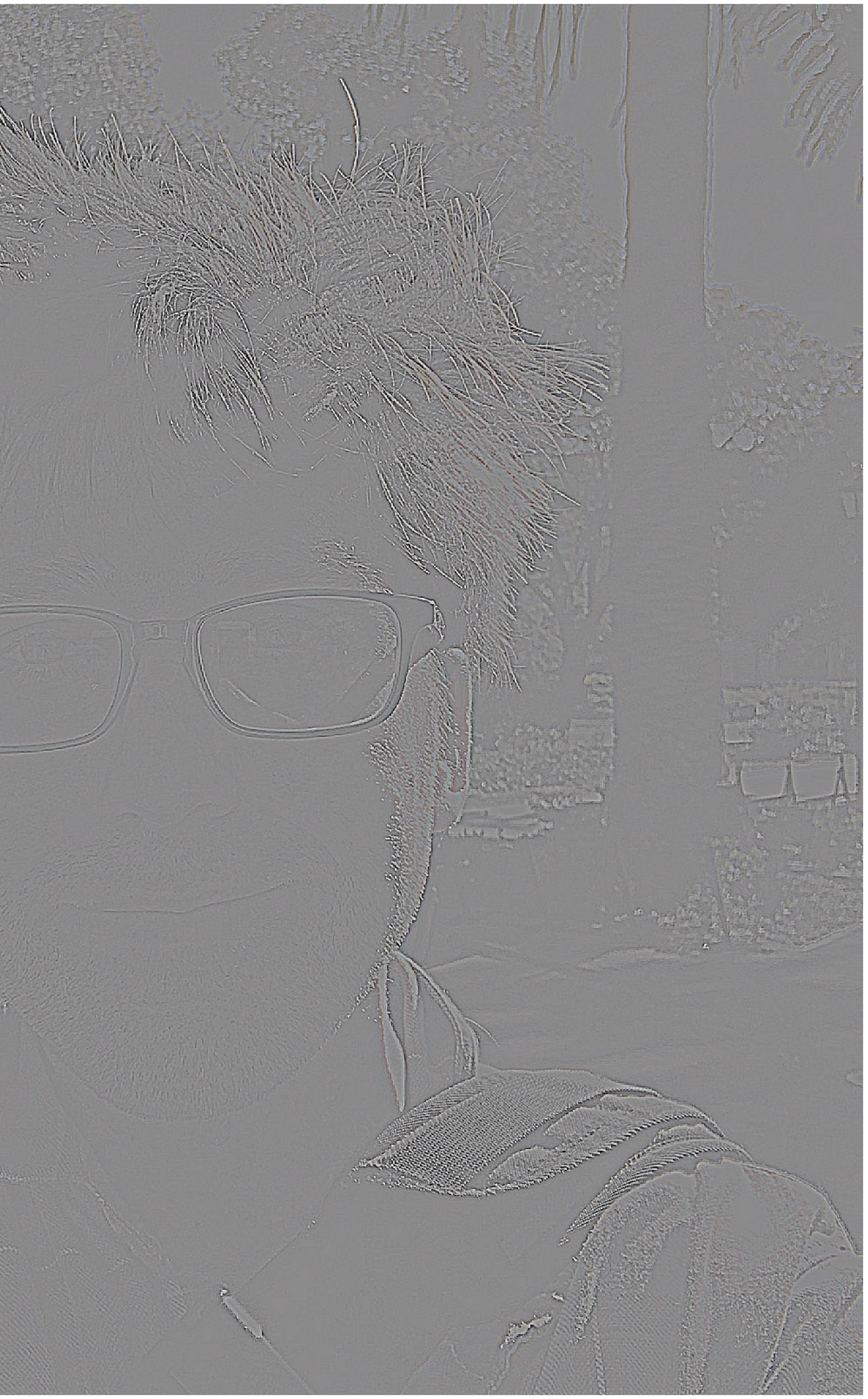
[Burt and Adelson 83]

Laplacian pyramid

$L_0 = G_0 - up(G_1)$

$L_1 = G_1 - up(G_2)$

Laplacian pyramid



$L_0 = G_0 - up(G_1)$

$L_1 = G_1 - up(G_2)$

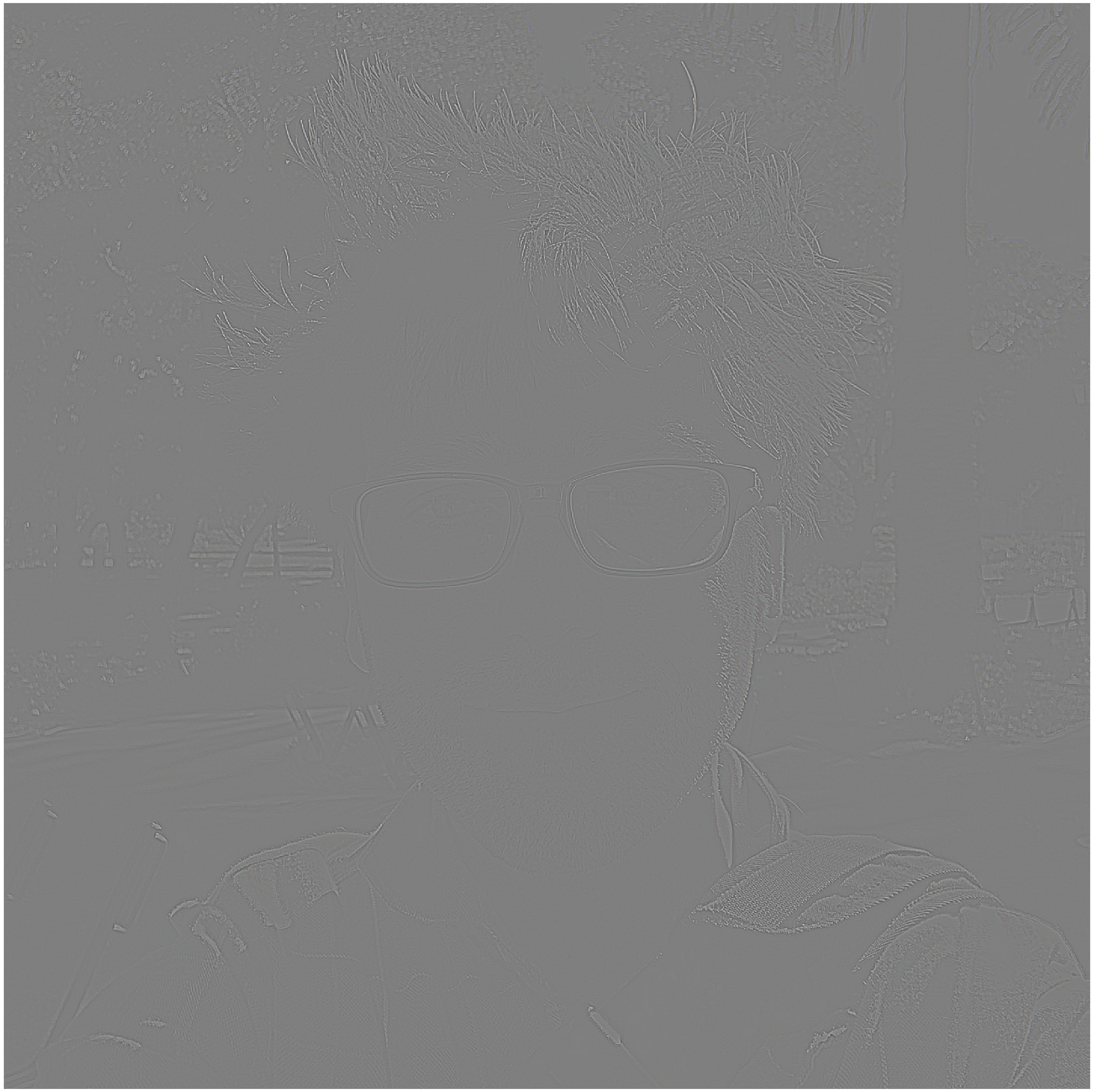
$L_2 = G_2 - up(G_3)$

$L_3 = G_3 - up(G_4)$

$L_4 = G_4$

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

Laplacian pyramid



$L_0 = G_0 - \text{up}(G_1)$

Laplacian pyramid

$L_1 = G_1 - up(G_2)$

Laplacian pyramid

$L_2 = G_2 - up(G_3)$

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

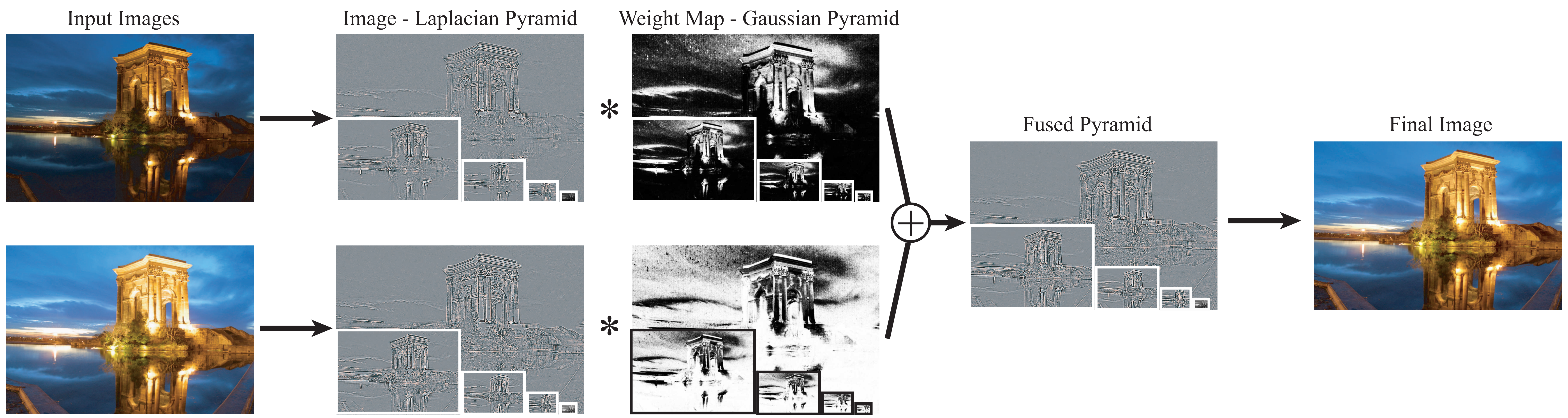
- $G_i(x,y)$ — frequencies up to limit given by *i*

- $L_i(x,y)$ — 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

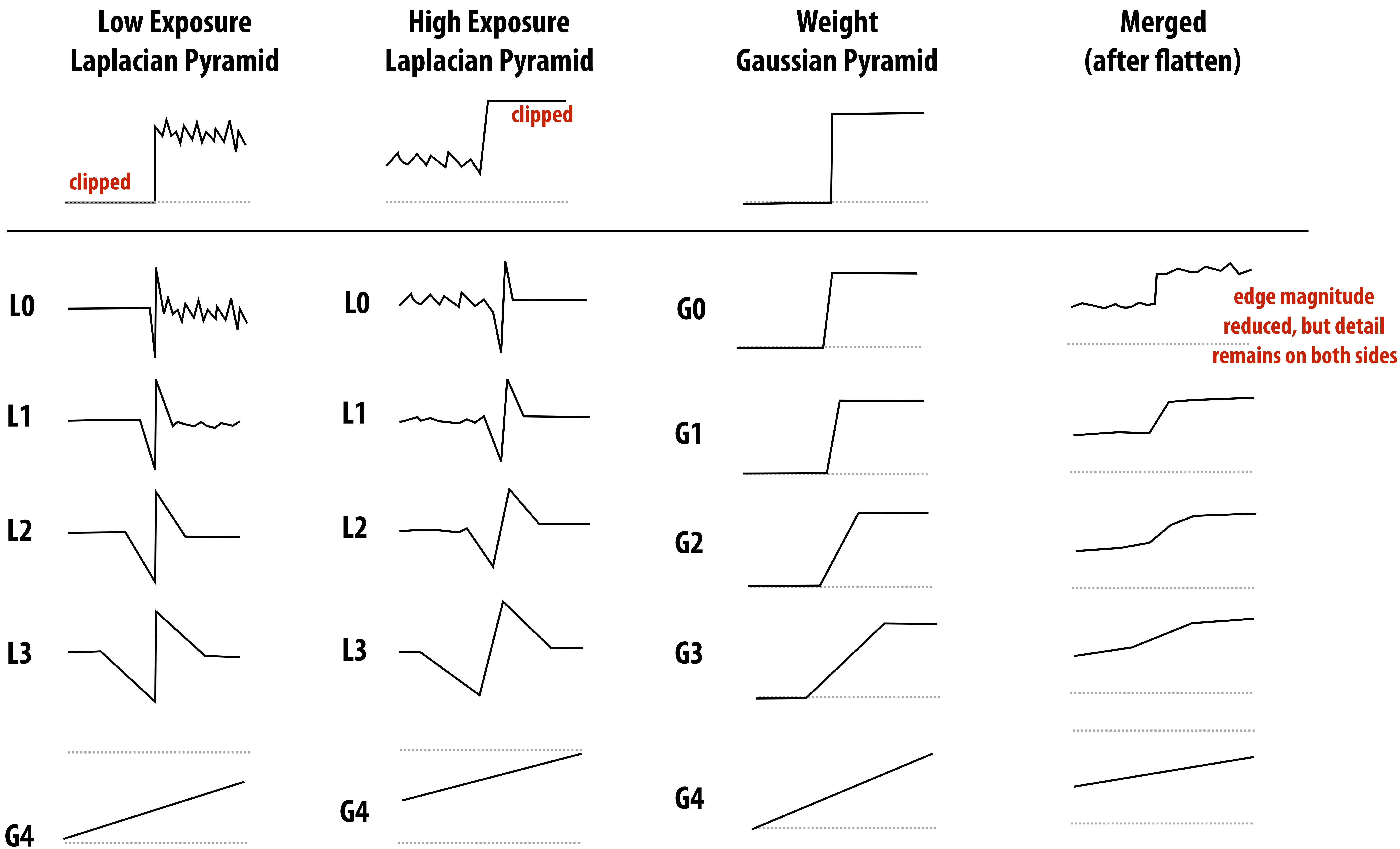
Four exposures (weights not shown)

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

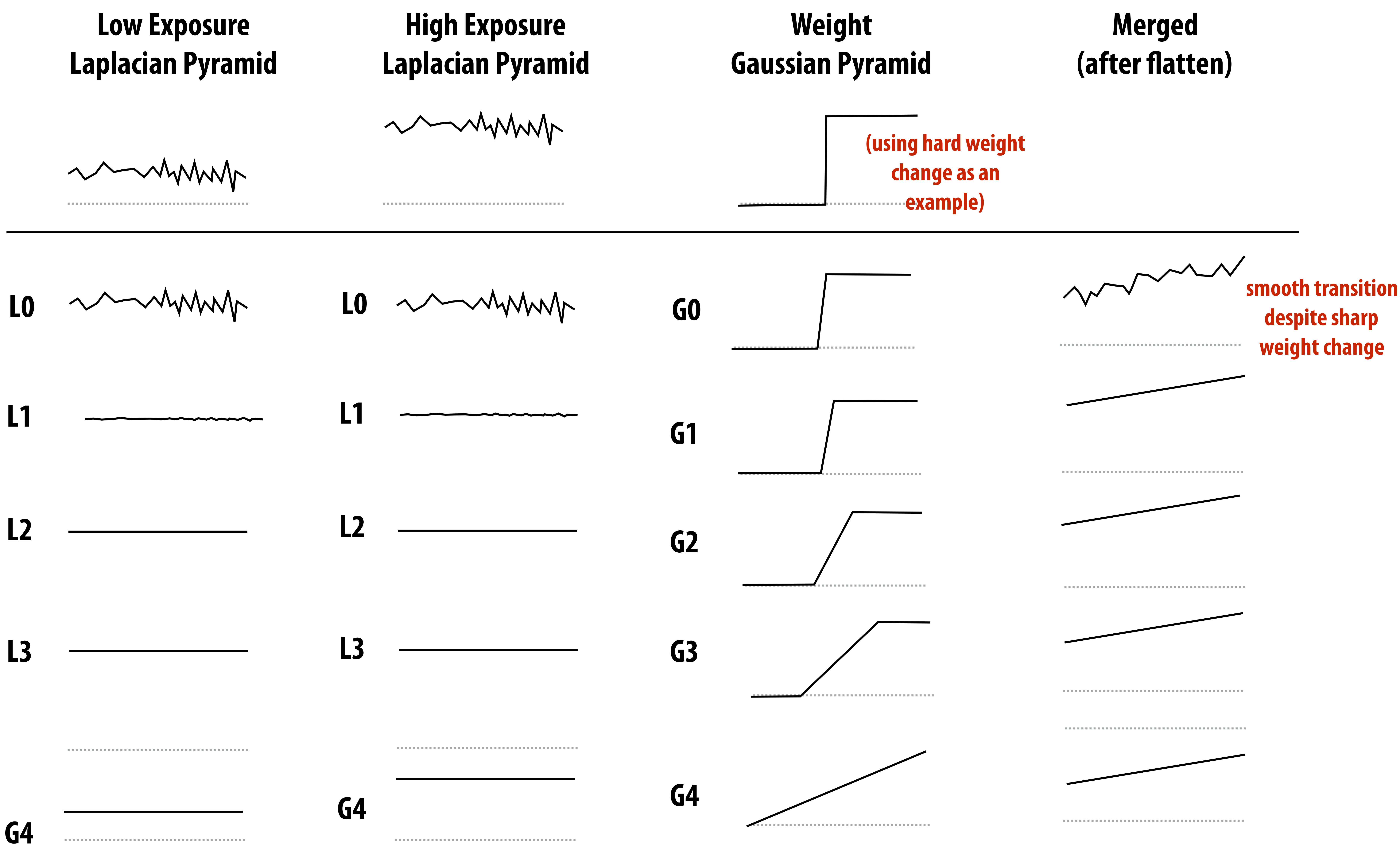
Merged result
(based on multi-resolution pyramid merge)

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
- 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
- Gamma Correction (non-linear mapping)

3x12 bits per pixel
RGB intensity per pixel
Pixel values linear in energy

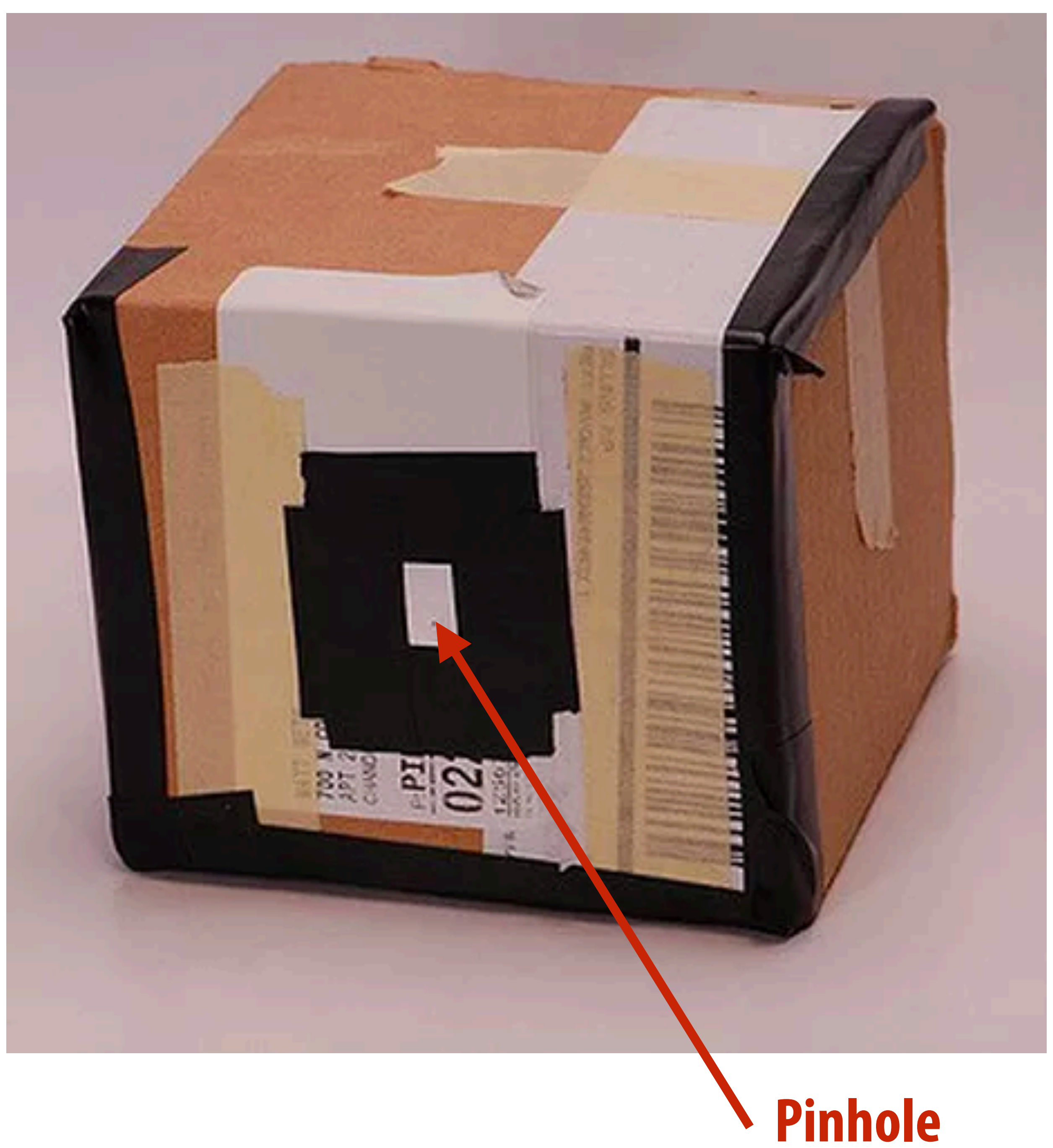
- Local tone mapping
- Final adjustments sharpen, fix chromatic aberrations, hue adjust, etc.

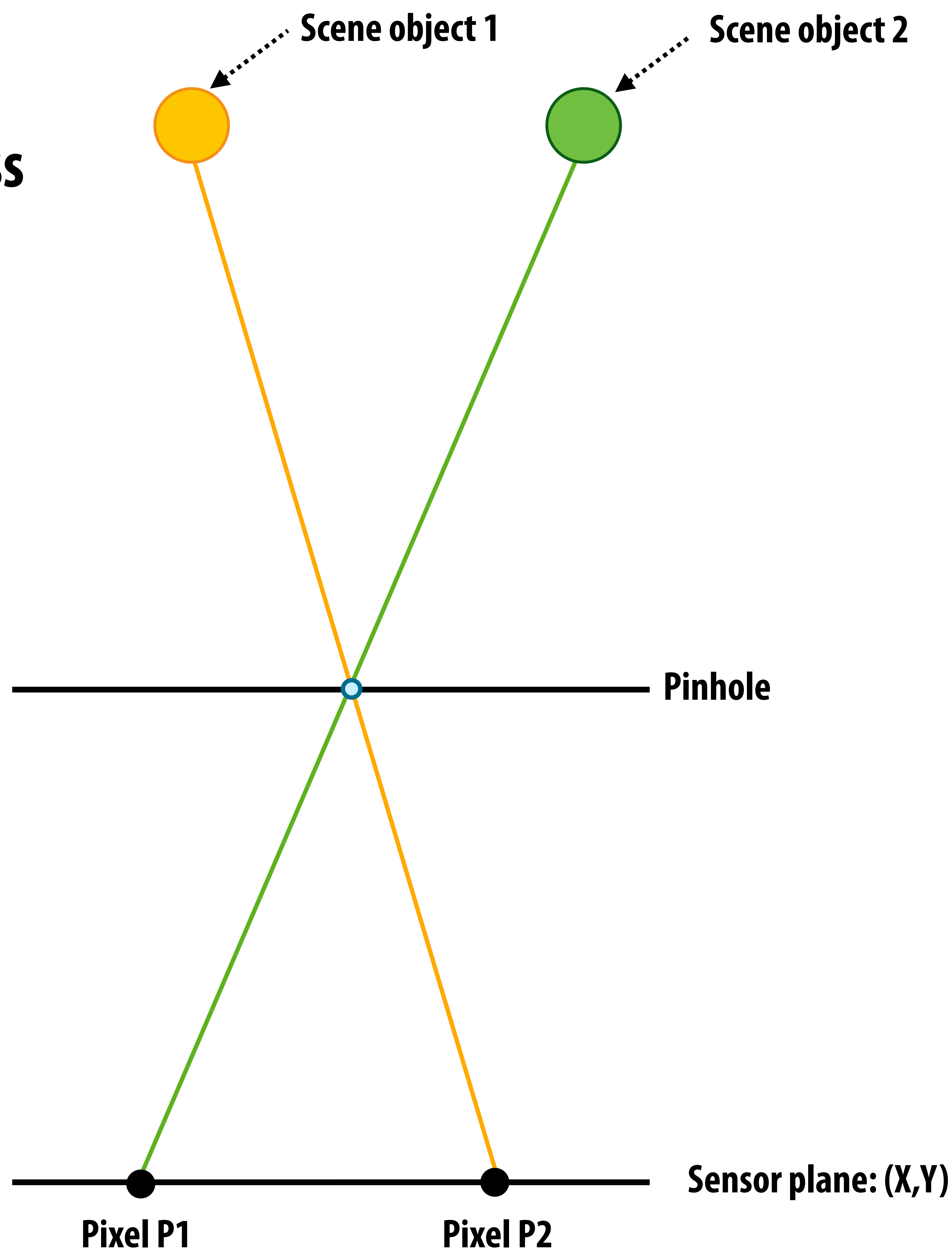
3x8-bits per pixel
Pixel values perceptually linear

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)



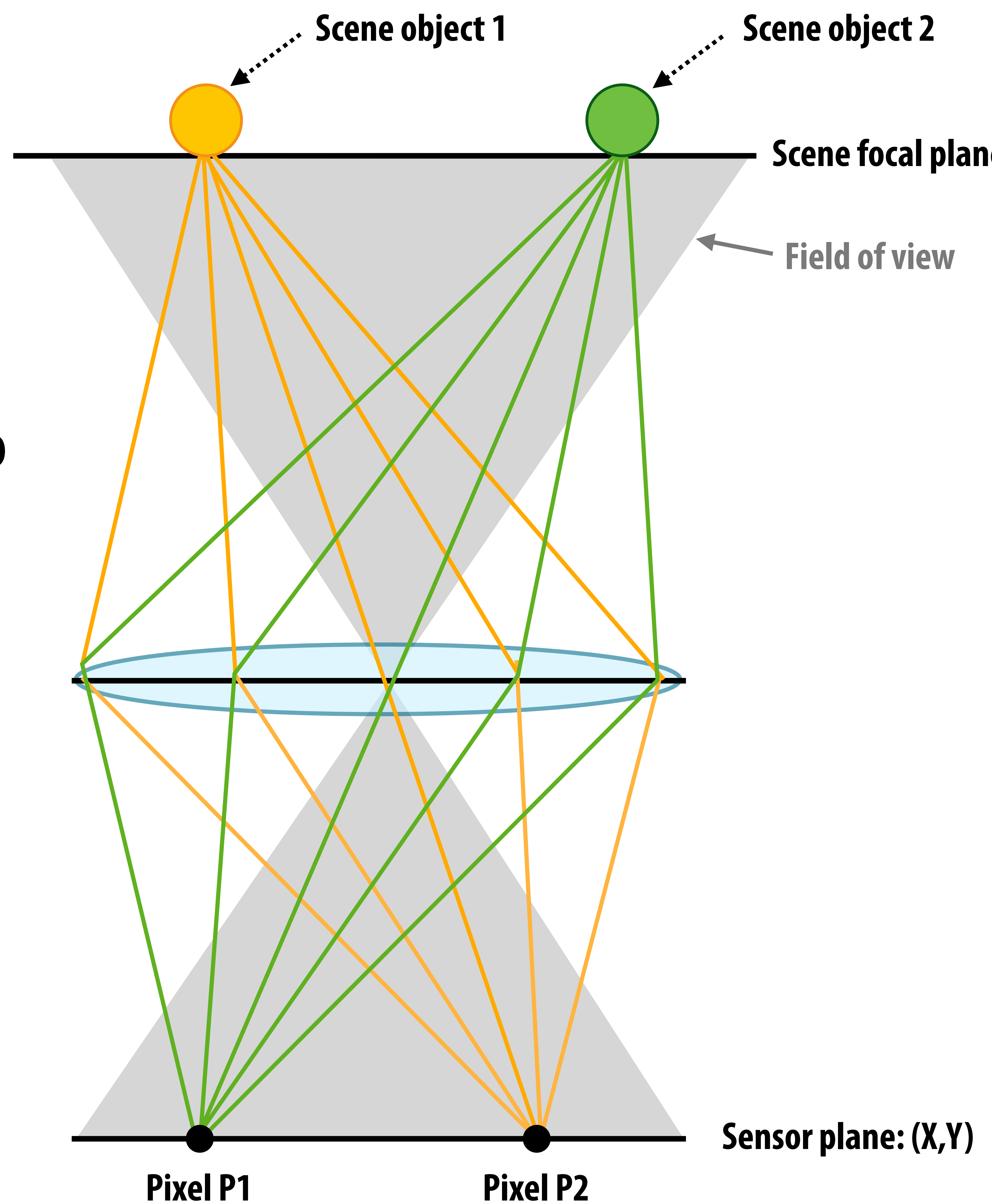


What does a lens do?

Camera with lens:

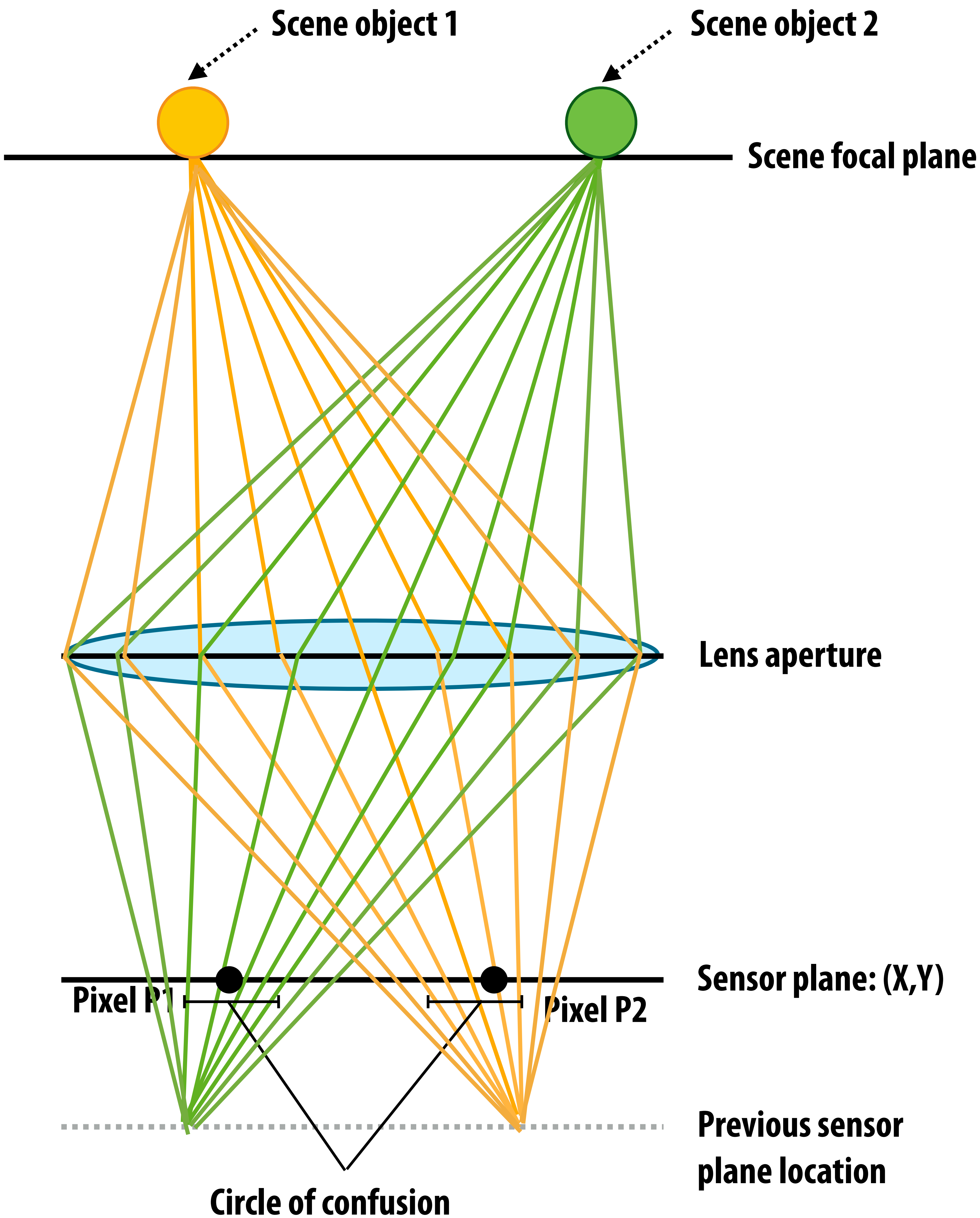
Every pixel accumulates all rays of light passing through lens aperture and refracted to location of pixel

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



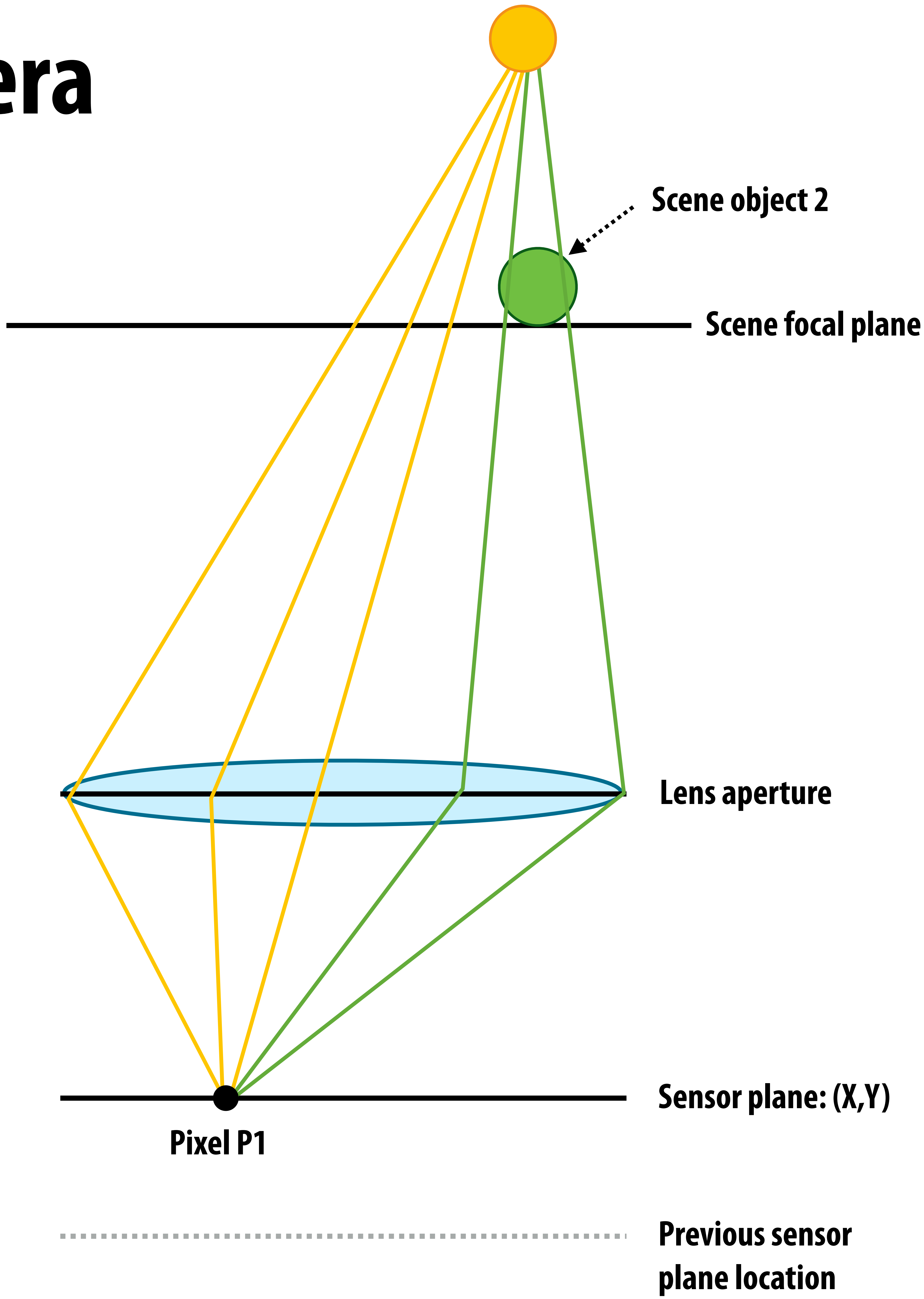
Bokeh

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

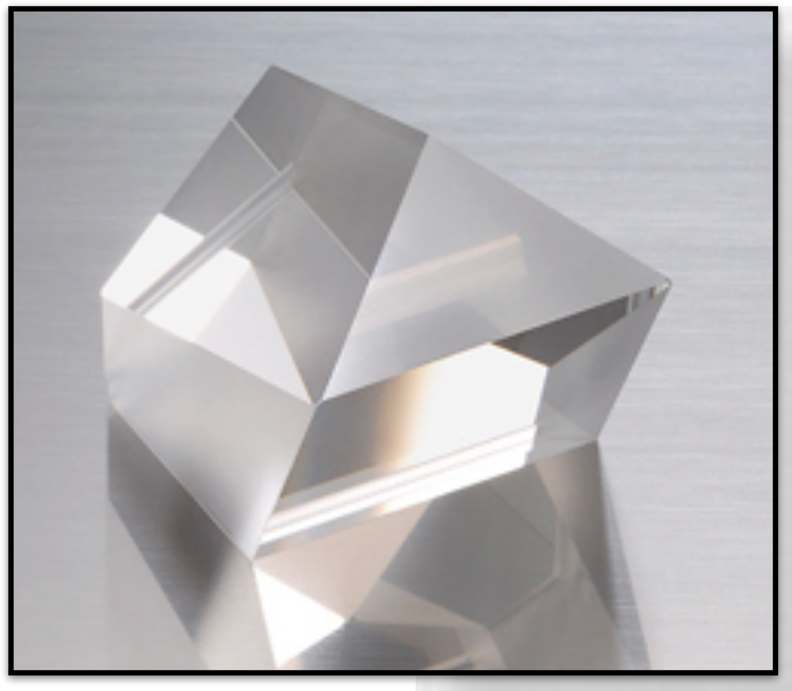
Autofocus demos

- **Phase-detection auto focus**
  - Common in SLRs

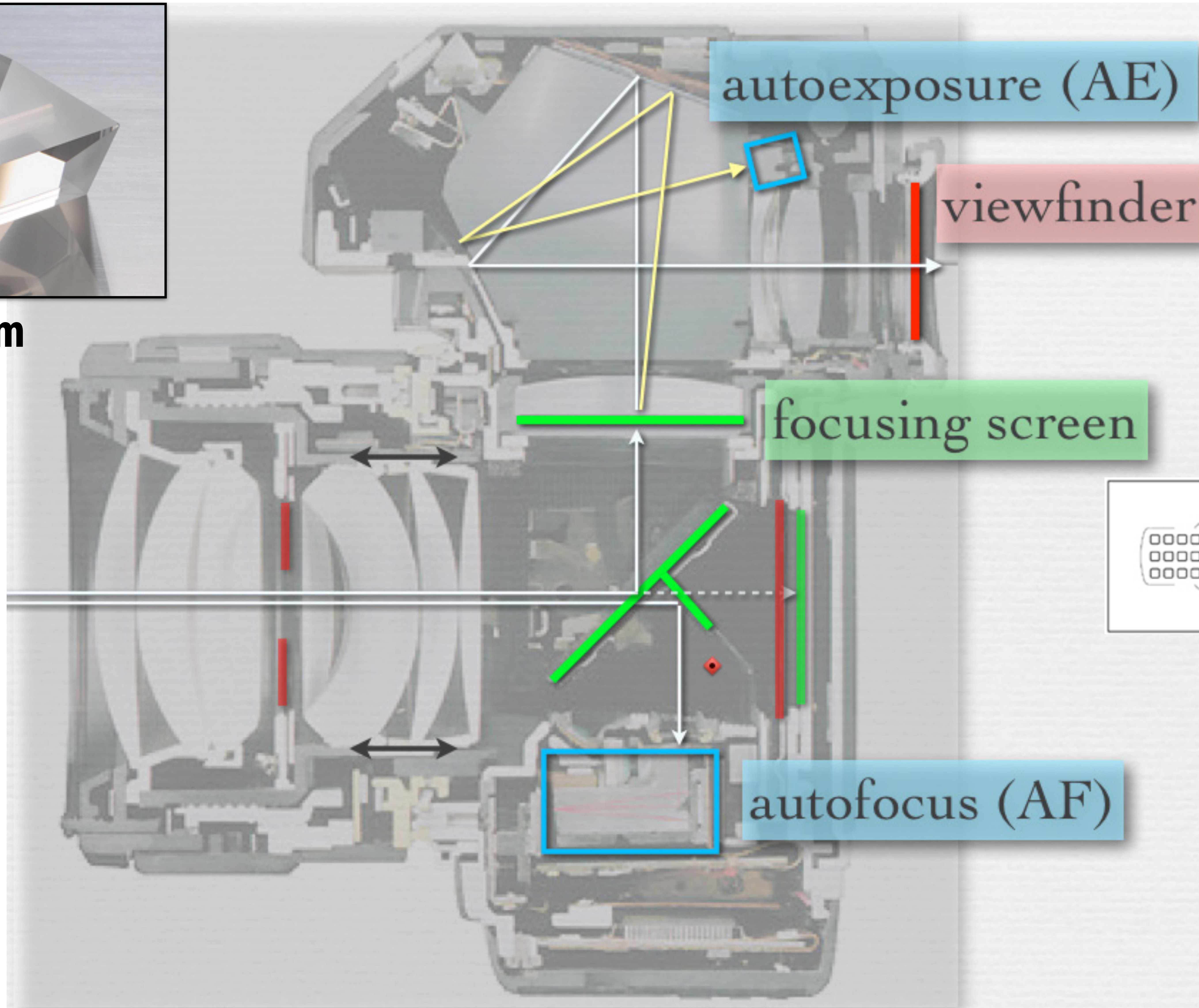
- **Contrast-detection auto focus**
  - Smartphone cameras

Demo credits: Marc Levoy and Stanford CS178 course staff

Single lens reflex (SLR) camera



Pentaprism



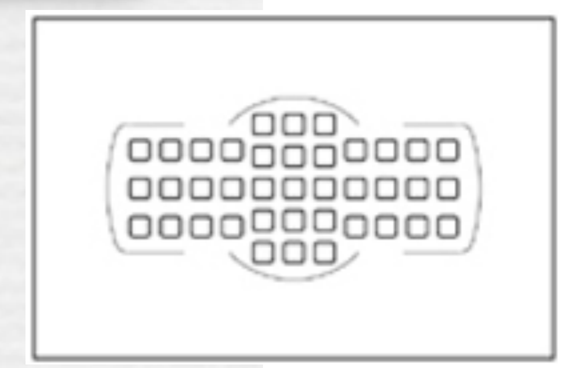
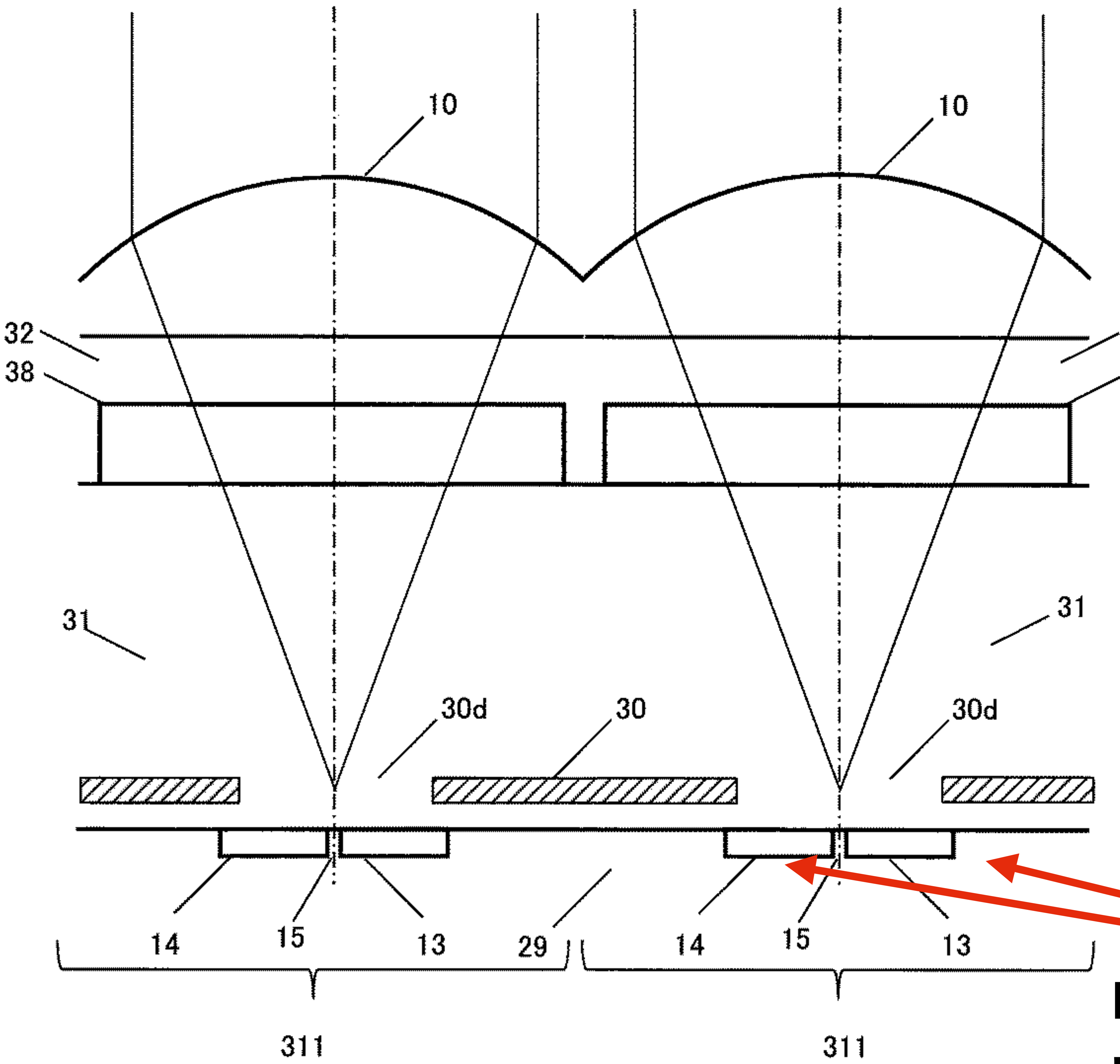


Image credits: Nikon, Marc Levoy

Split pixel sensor

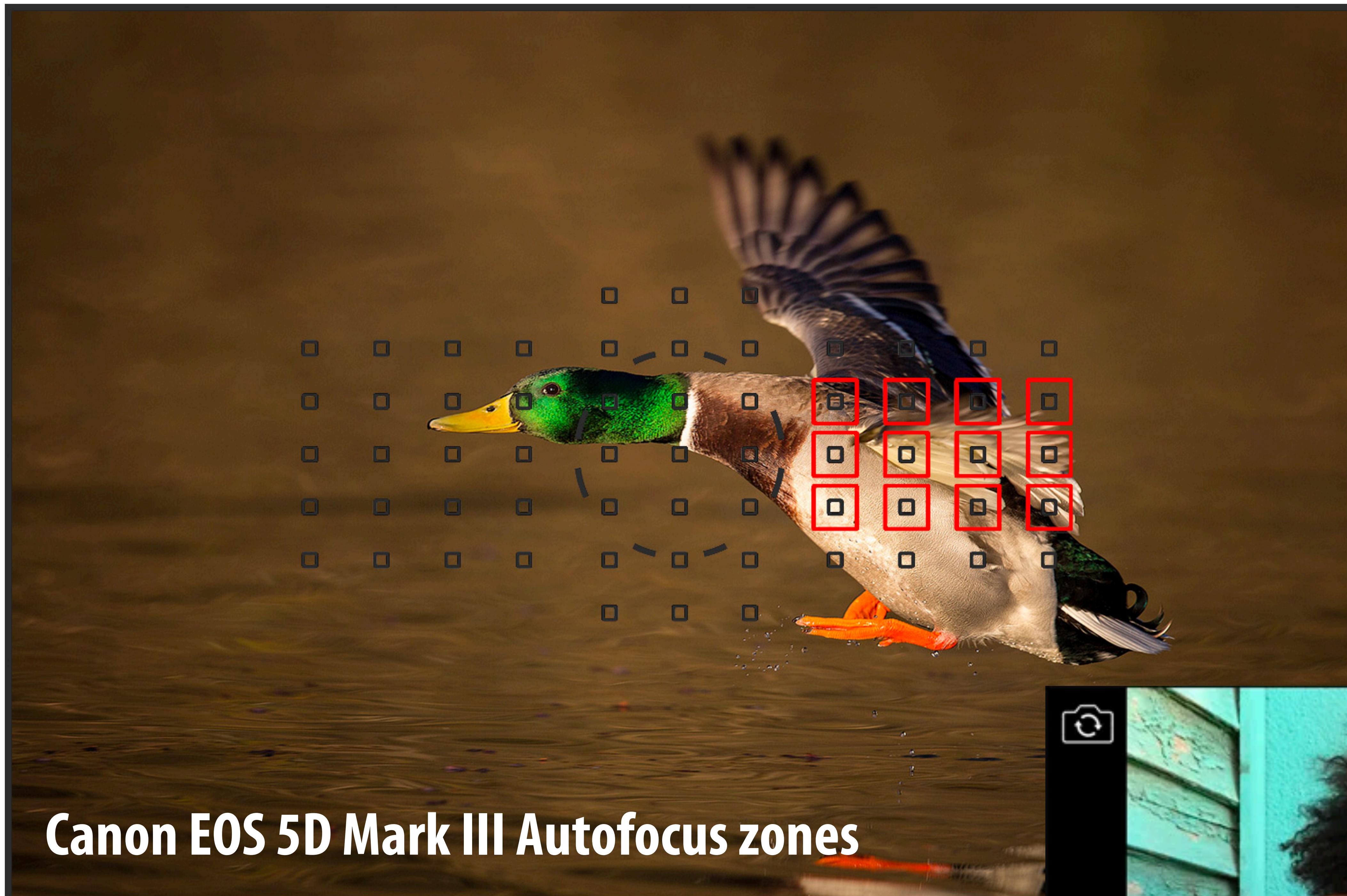


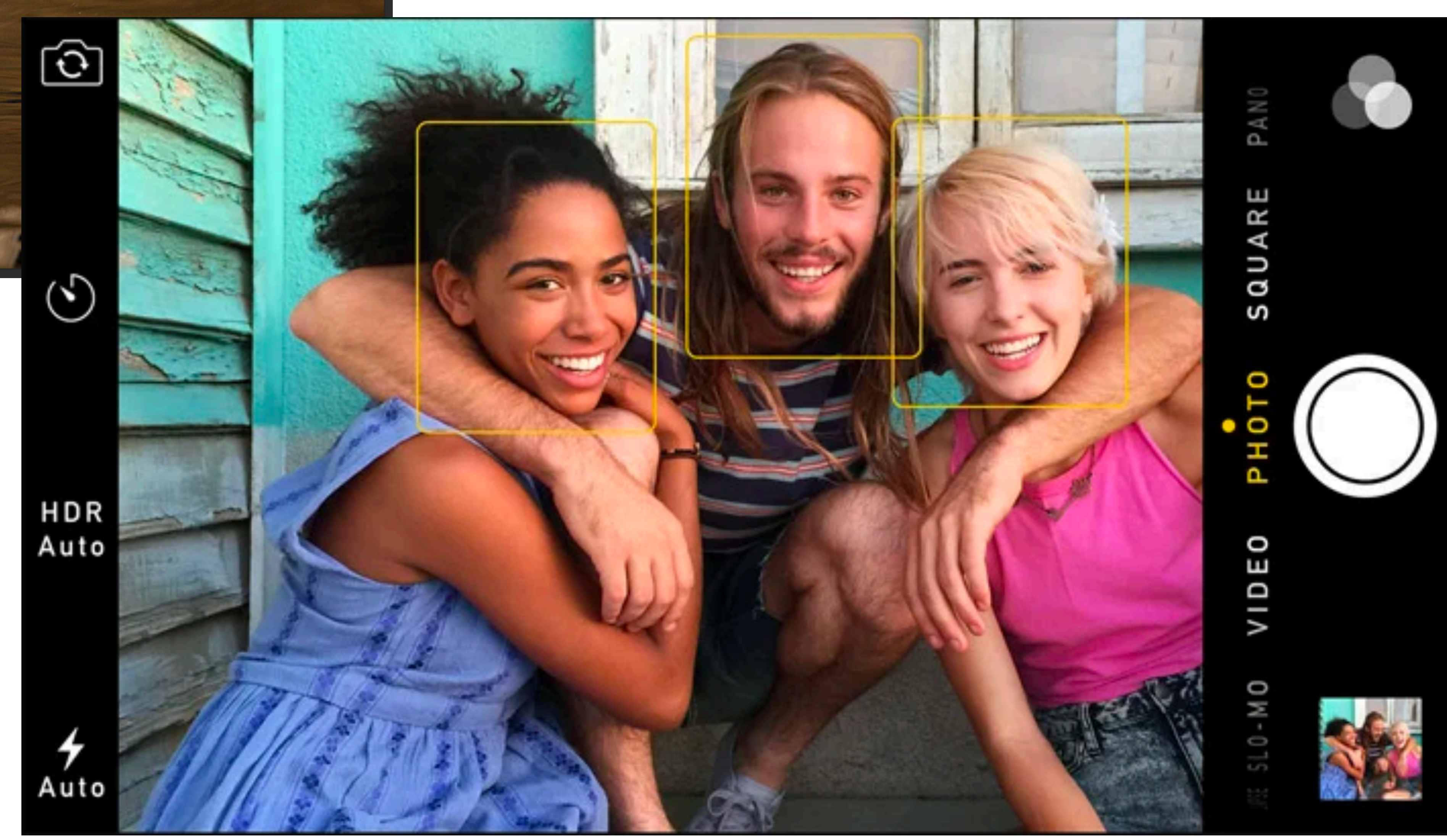
When both pixels have the same response, camera is in focus, why?

Now two pixels under each microlens (not one)

Image credit: Nikon

What part of image should be in focus?





Heuristics:

Focus on closest scene region

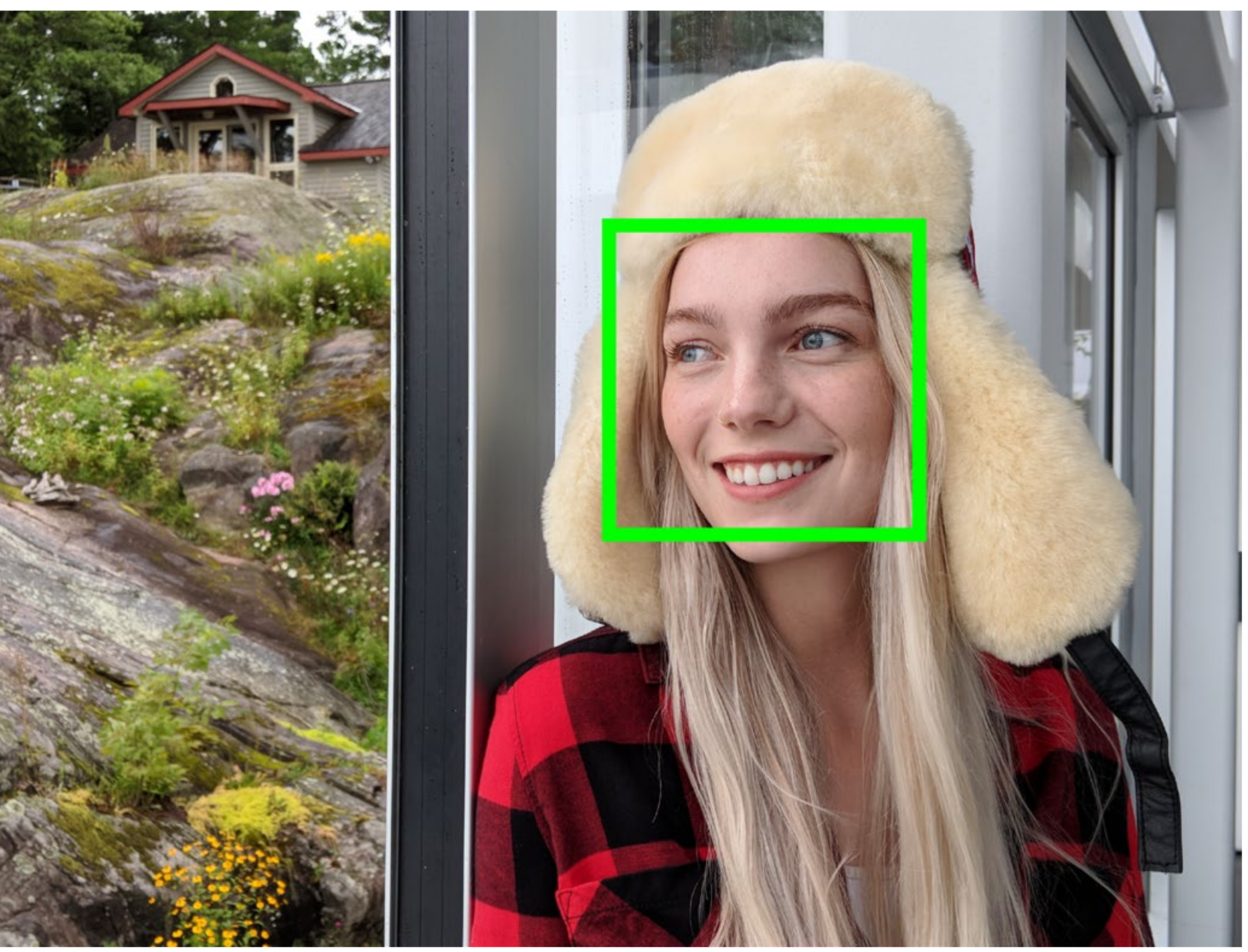
Put center of image in focus

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

Portrait mode (tonight's reading)

- 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]

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