Lecture 16:

Image Processing for Digital Photography

Interactive Computer Graphics
Stanford CS248, Winter 2020

Finishing up from last time

Review: discrete 2D convolution

$$\underbrace{(f * g)(x, y)}_{\text{output image}} = \sum_{i,j=-\infty}^{\infty} \underbrace{f(i, j)}_{\text{filter}} \underbrace{I(x - i, y - j)}_{\text{input image}}$$

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:

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

Bilateral filter

Original

After bilateral filter

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

https://www.thebest3d.com/howler/11/new-in-version-11-bilateral-noise-filter.html

Bilateral filter

Original | **After bilateral filter**

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

http://opencvpython.blogspot.com/2012/06/smoothing-techniques-in-opencv.html

Bilateral filter

$$\mathrm{BF}[I](p) = \frac{1}{W_p} \sum_{i,j} \underbrace{f(|I(x-i,y-j) - I(x,y)|)}_{\text{Re-weight based on difference in input image pixel values}} \overbrace{G_\sigma(i,j)}^{\text{Gaussian blur kernel}} \overbrace{I(x-i,y-j)}^{\text{Input image}}$$

Normalization
(weights should sum to 1)

For all pixels in support region of Gaussian kernel

$$\frac{1}{W_p} = \sum_{i,j} f(|I(x-i,y-j) - I(x,y)|)G_\sigma(i,j)$$

- The bilateral filter is an "edge preserving" filter: down-weight contribution of pixels on the "other side" of strong edges. $f(x)$ defines what "strong edge means"
- Spatial distance weight term $f(x)$ could itself be a gaussian
  - Or very simple: $f(x) = 0$ if $x > \mathit{threshold}$, 1 $\mathit{otherwise}$

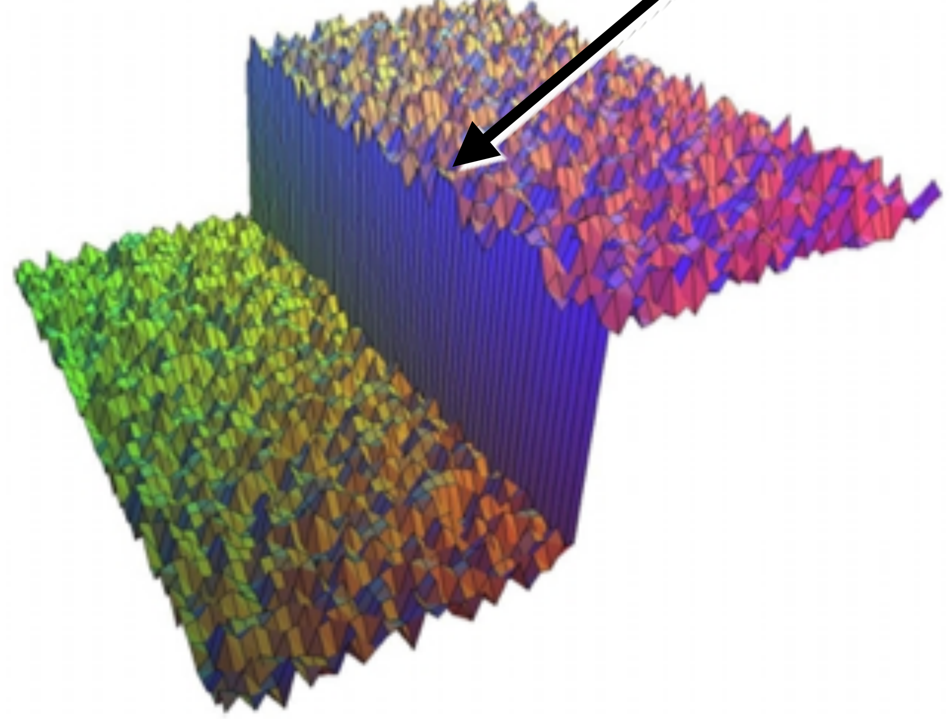
Value of output pixel (x,y) is the weighted sum of all pixels in the support region of a truncated gaussian kernel

But weight is combination of spatial distance and input image pixel intensity difference. (the filter's weights depend on input image content)

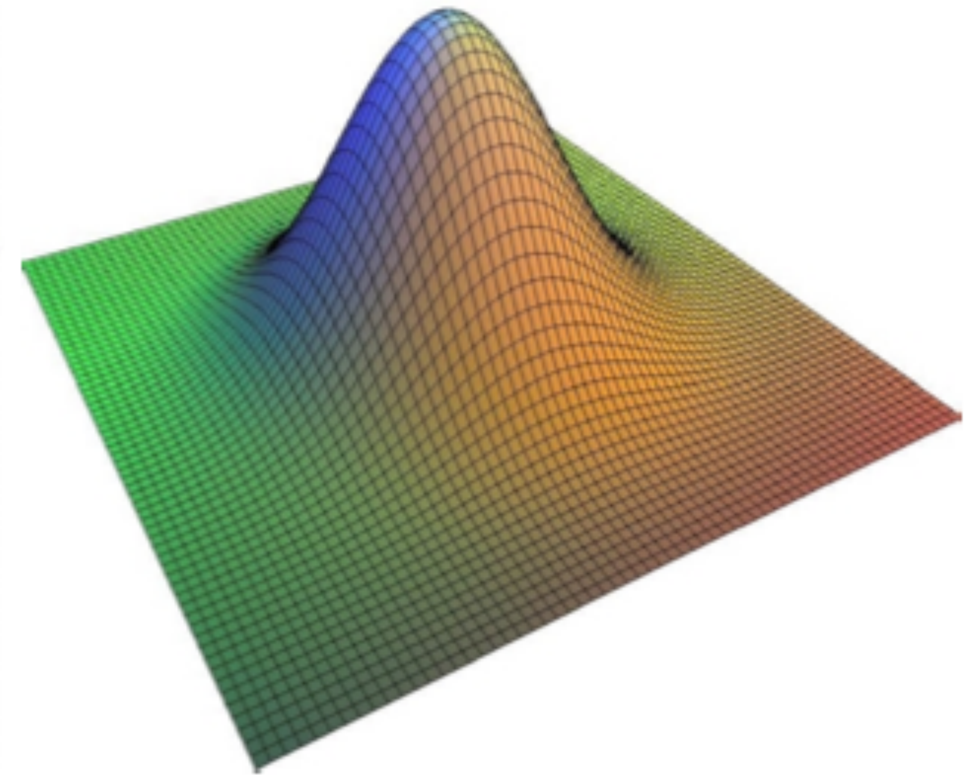
Bilateral filter

Input pixel *p*

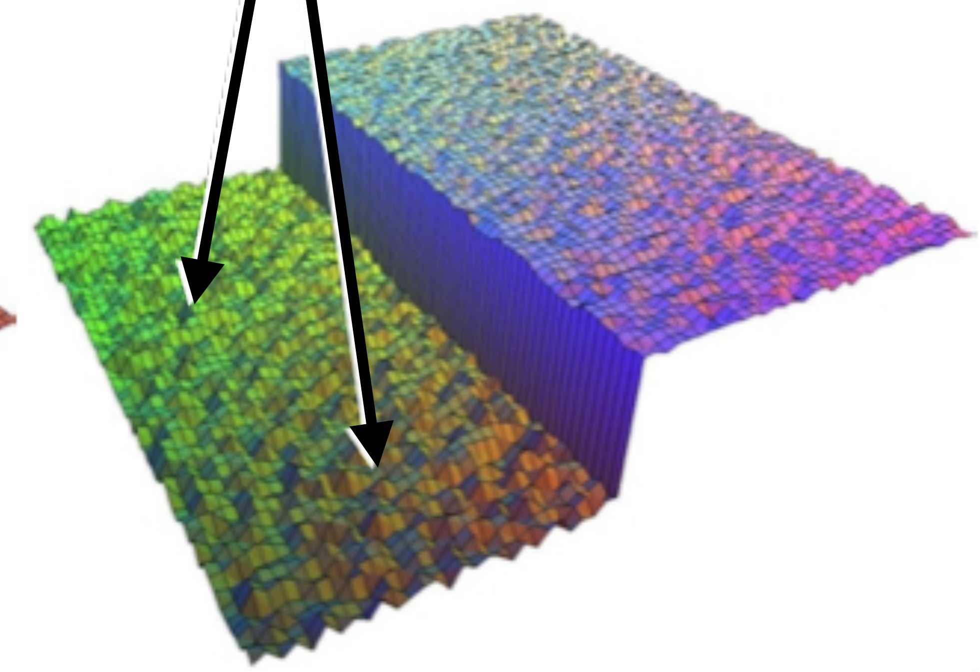
Pixels with significantly different intensity as *p* contribute little to filtered result (they are "on the "other side of the edge"



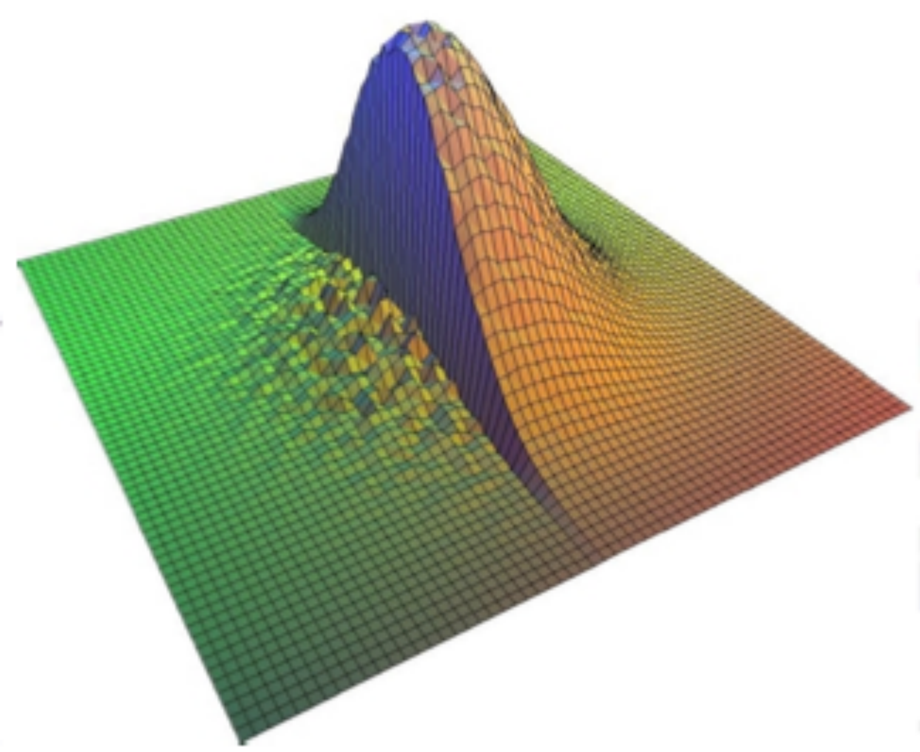
Input image



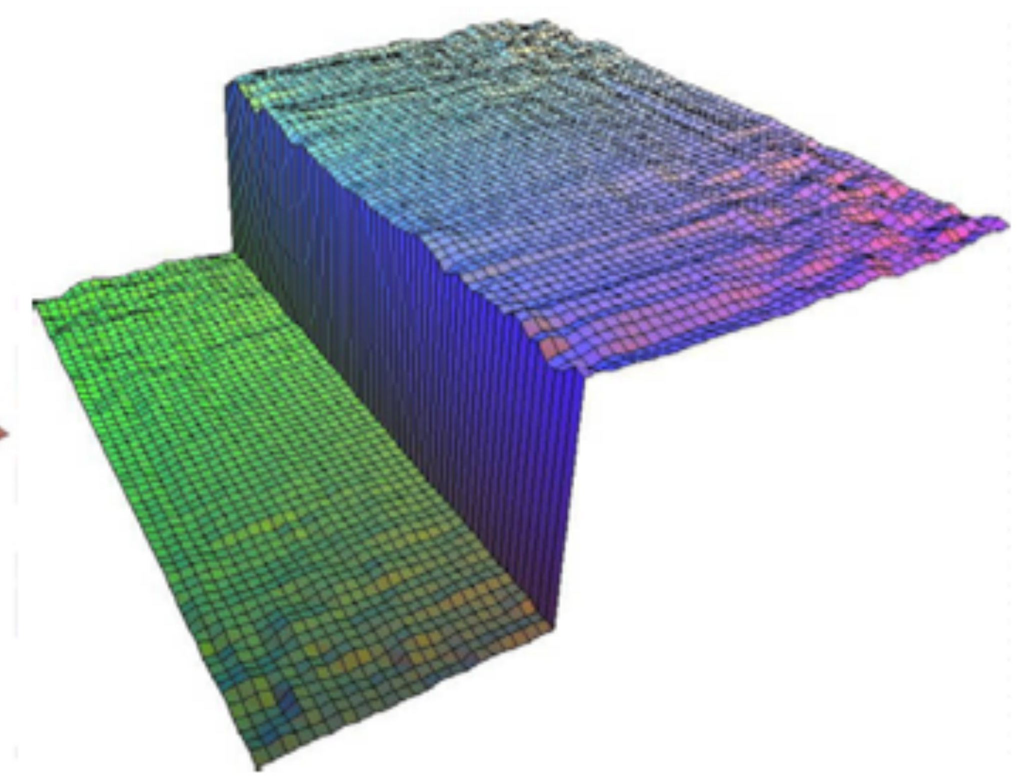
G(): gaussian about input pixel *p*



f(): Influence of support region



G x f: filter weights for pixel *p*



Filtered output image

Figure credit: Durand and Dorsey, "Fast Bilateral Filtering for the Display of High-Dynamic-Range Images", SIGGRAPH 2002

Bilateral filter: kernel depends on image content

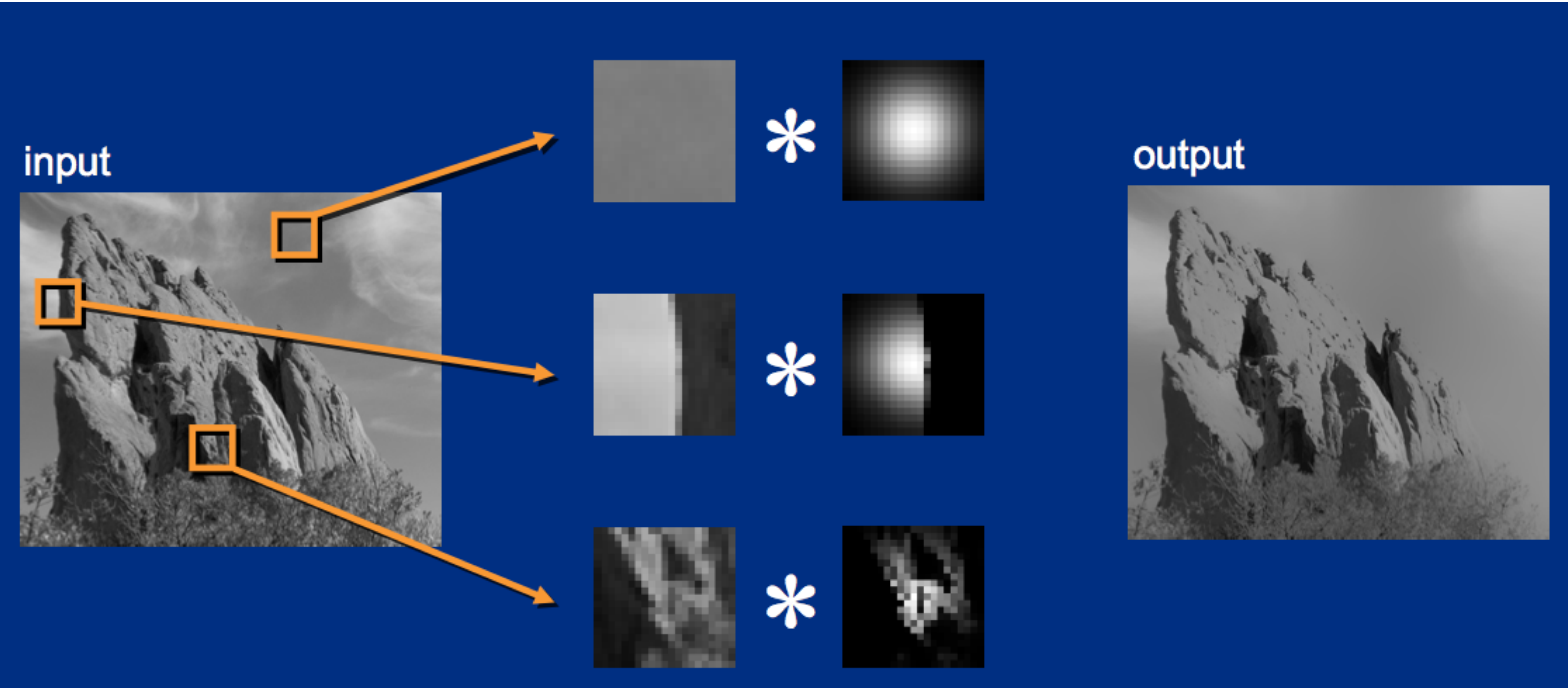


Figure credit: SIGGRAPH 2008 Course: "A Gentle Introduction to Bilateral Filtering and its Applications" Paris et al.

Review

- We've talked about how to manipulate images in terms of adjusting pixel values (localize edits in space to certain pixels)

- We've talked about how to manipulate images in terms of adjusting coefficients of frequencies (localize edits to certain frequencies)
    - Eliminate high frequencies (blur)
    - Increase high frequencies (sharpen)

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

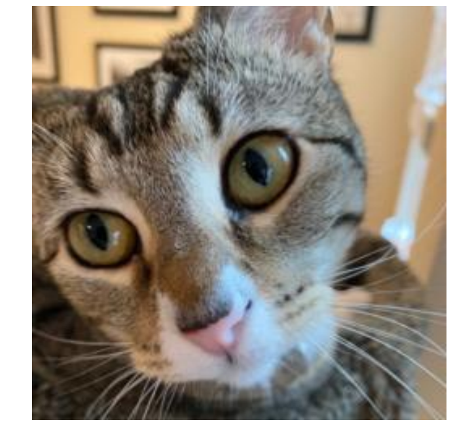
(Adjust certain frequency content of image, in a particular region of the image)

Josephine the Graphics Cat

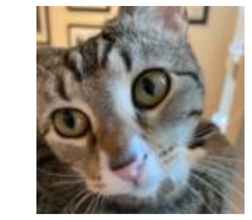
Gaussian pyramid

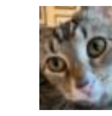
G_0 = original image

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



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





Each image in pyramid contains increasingly low-pass filtered signal

down() = Gaussian blur, then downsample by factor of 2 in both X and Y dimensions

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*4 + ii];
      output[j*WIDTH/2 + i] = tmp;
  }
}
```

Gaussian pyramid

G_0 (original image)

Gaussian pyramid

G_1

(upsampled back to full res for visualization)

Gaussian pyramid

G_2

(upsampled back to full res for visualization)

Gaussian pyramid

G_3

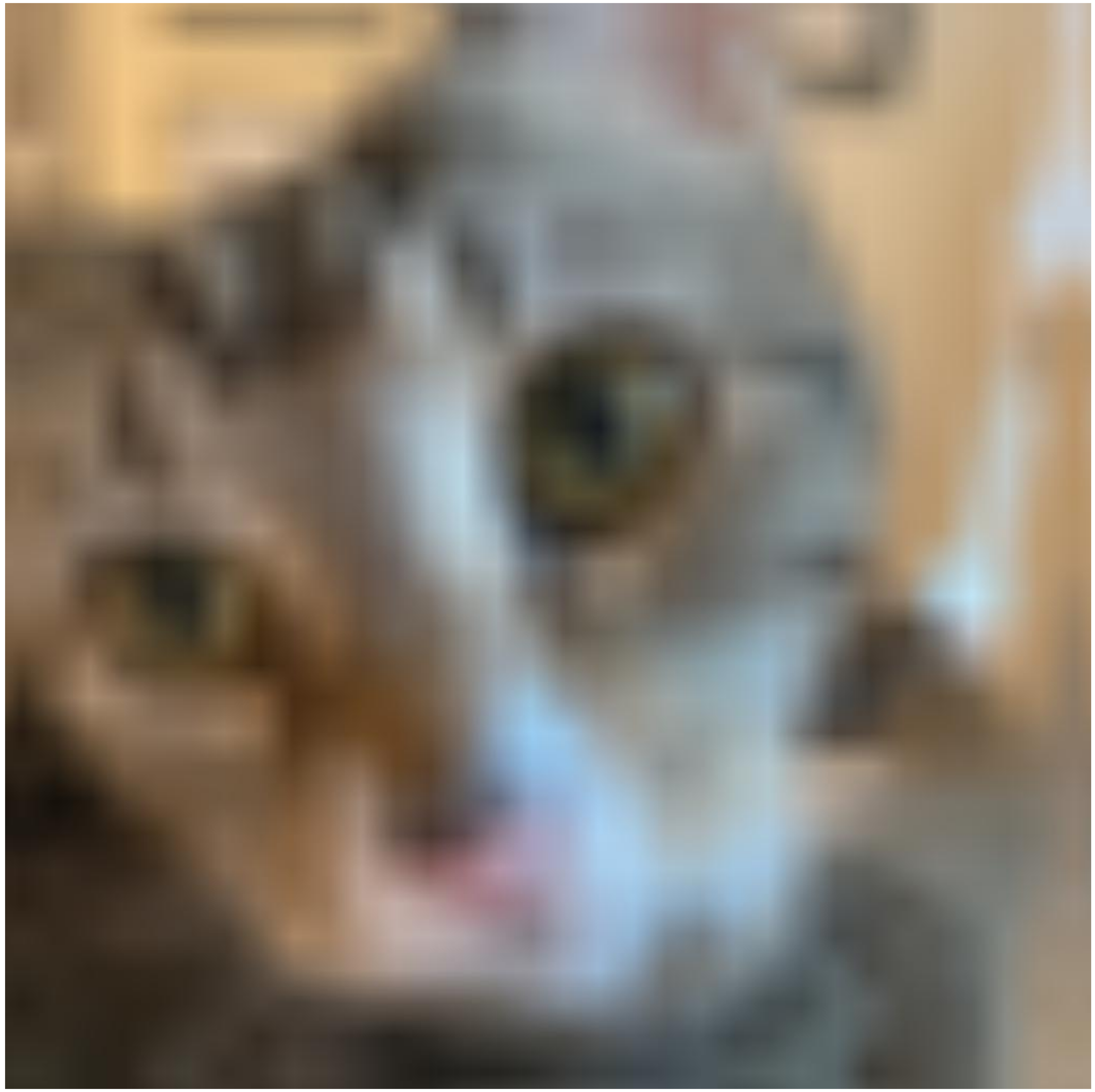
(upsampled back to full res for visualization)

Gaussian pyramid

G_4

(upsampled back to full res for visualization)

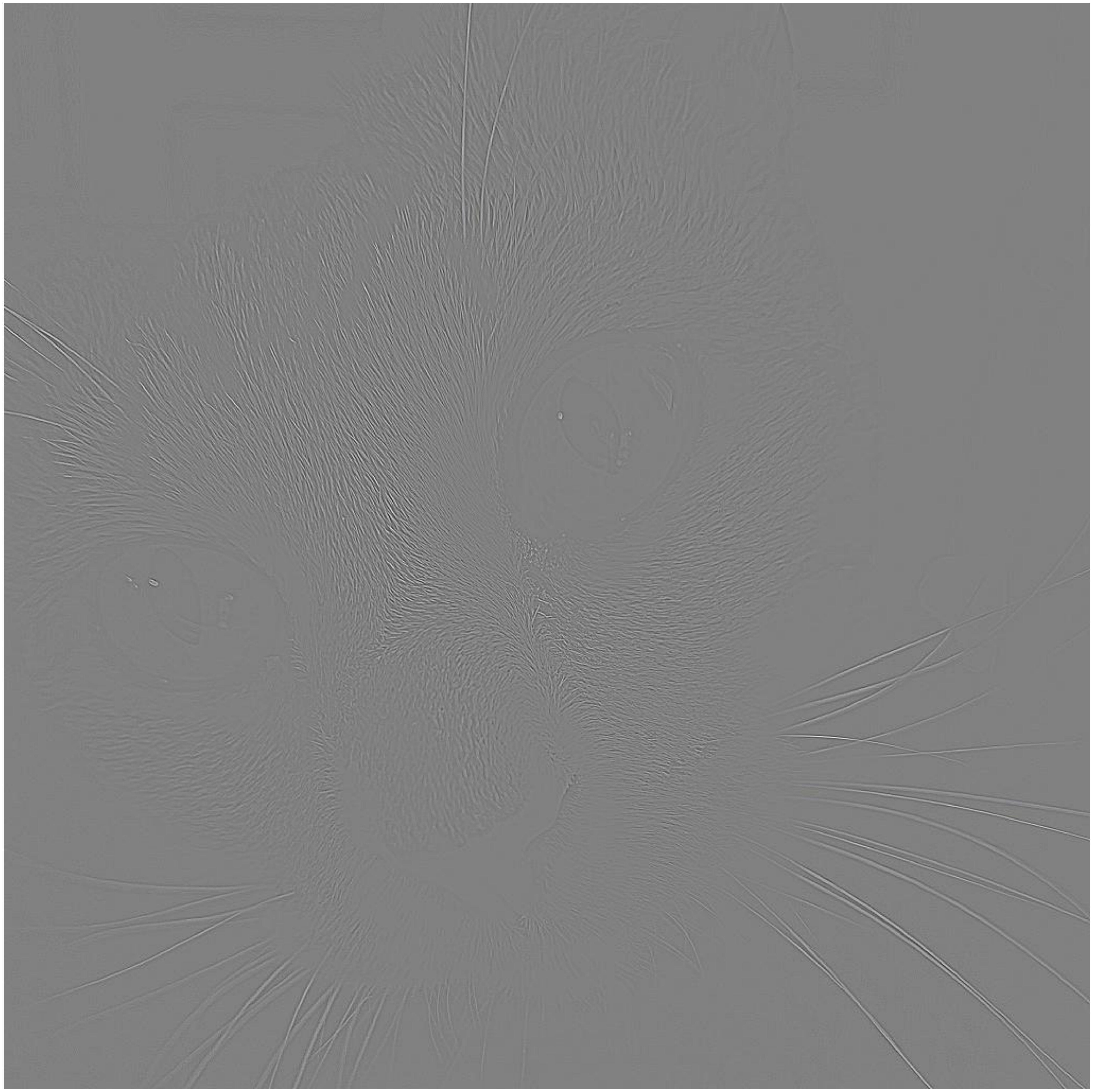
Gaussian pyramid



G_5

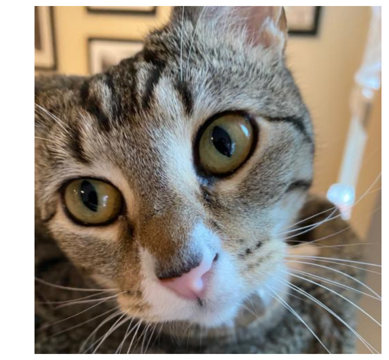
(upsampled back to full res for visualization)

Laplacian pyramid



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

G_0



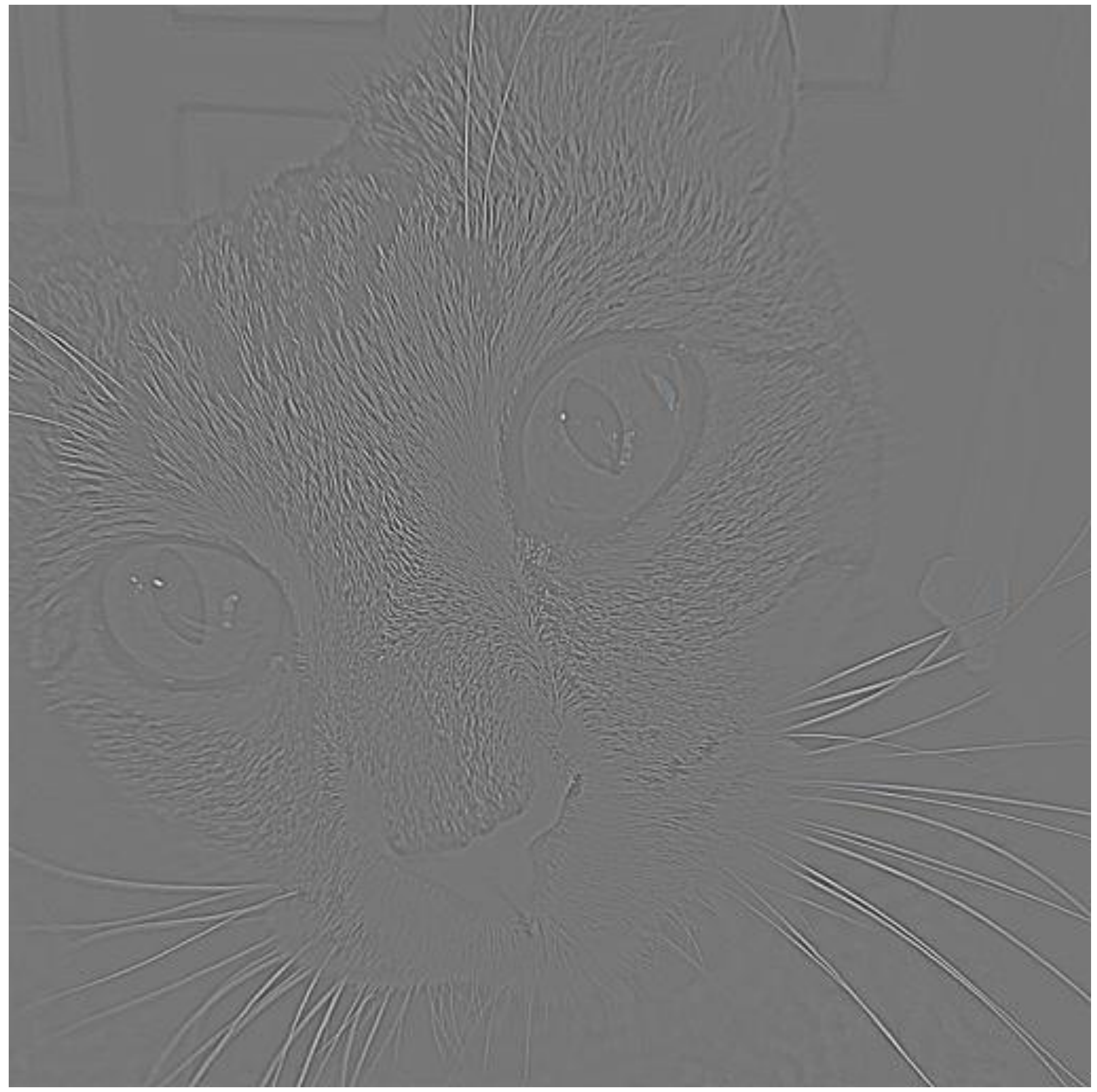
$G_1 = \mathrm{down}(G_0)$

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

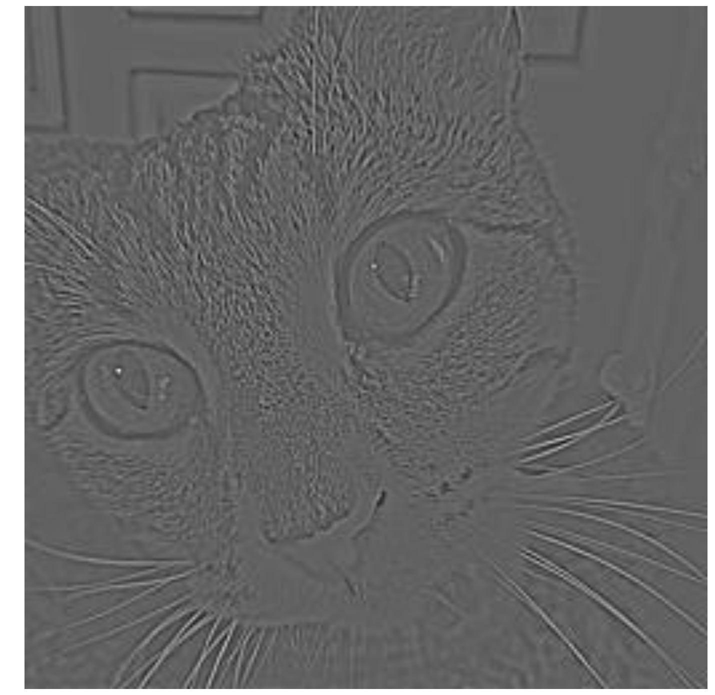
[Burt and Adelson 83]

Laplacian pyramid

$L_0 = G_0$ - up(G_1)



$L_1 = G_1$ - up(G_2)

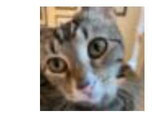


$L_2 = G_2$ - up(G_3)

$L_3 = G_3$ - up(G_4)

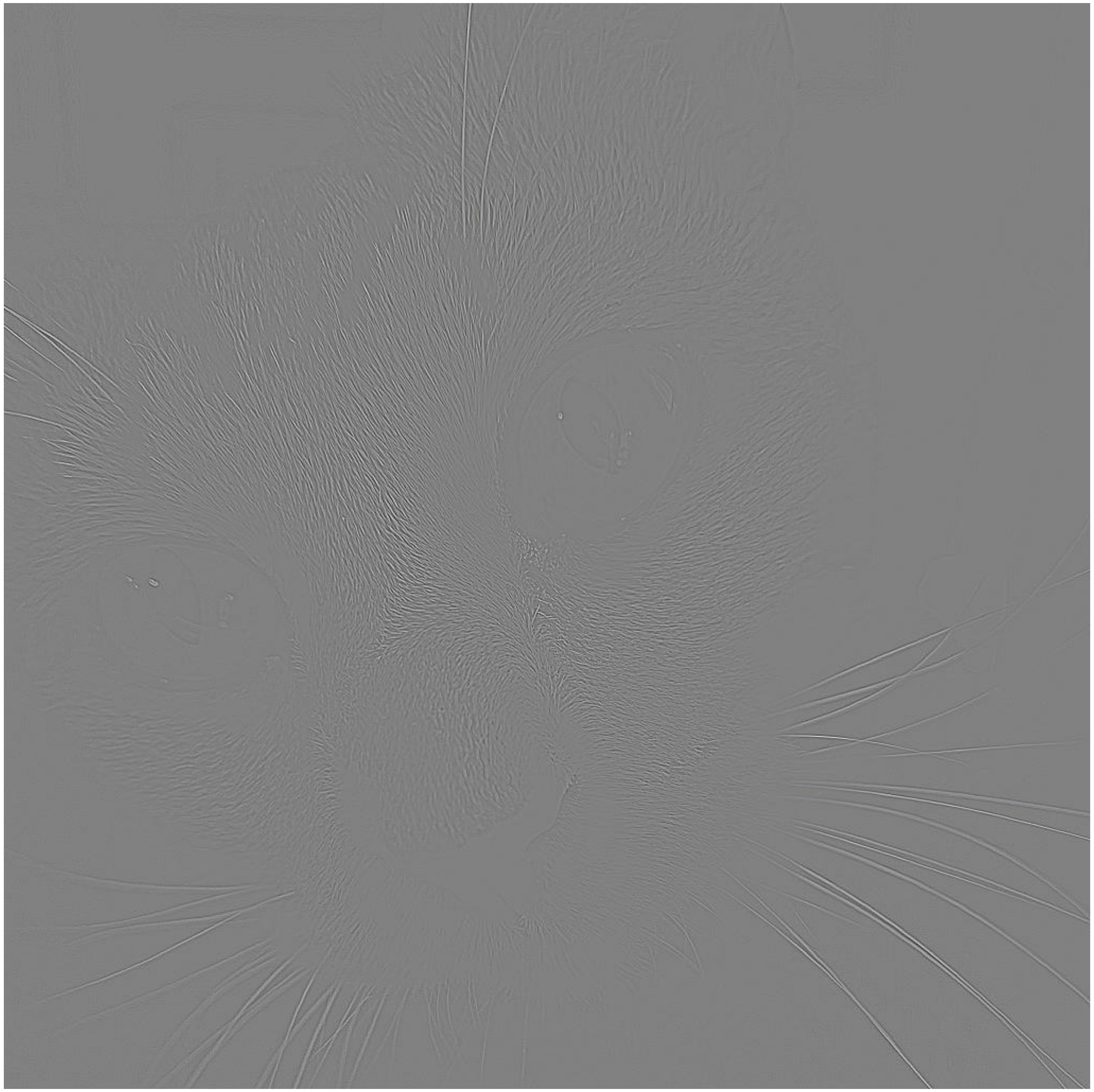
$L_4 = G_4$ - up(G_5)

$L_5 = G_5$



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

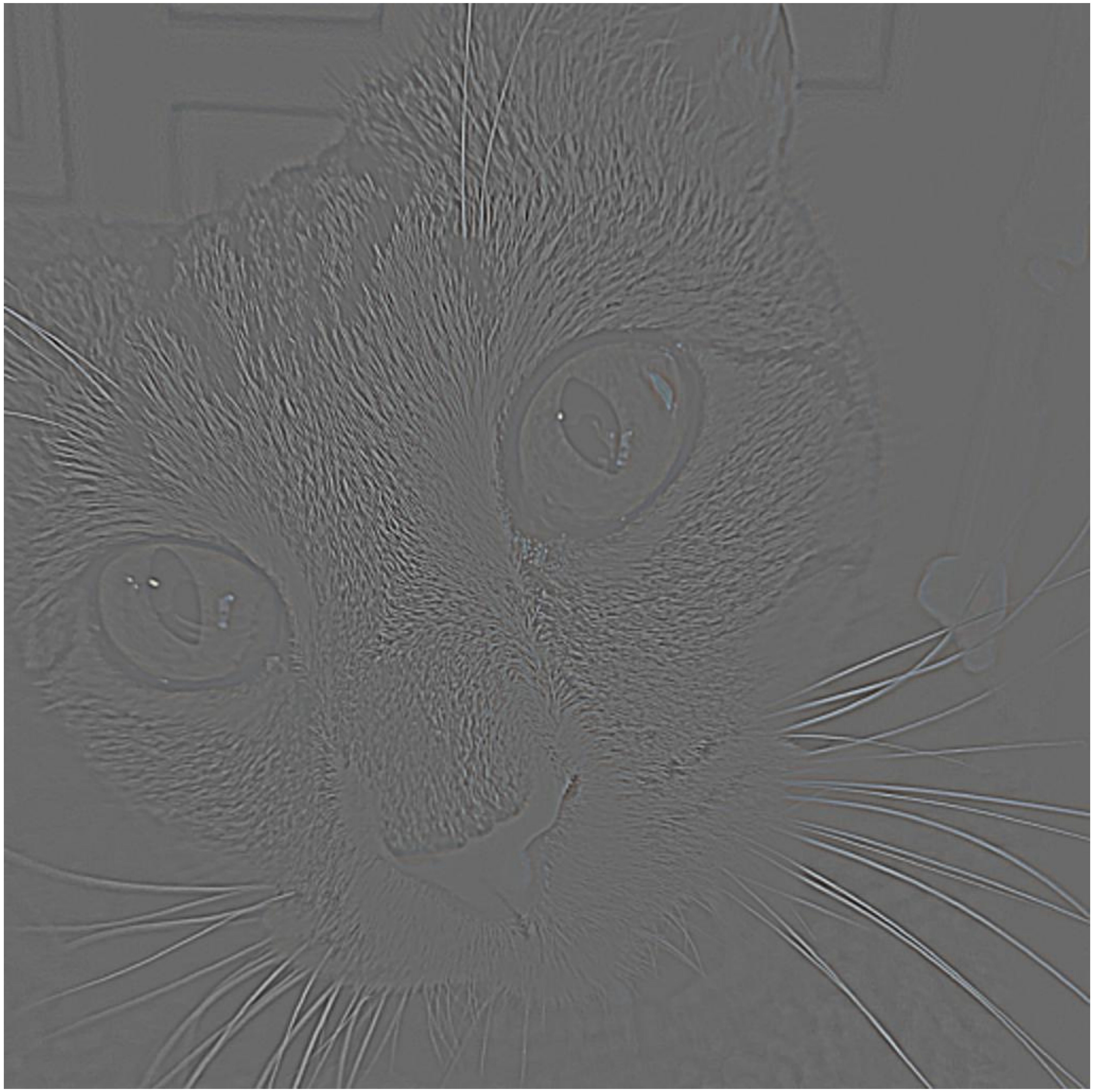
Laplacian pyramid



$L_0 = G_0$ - up(G_1)

(upsampled back to full res for visualization)

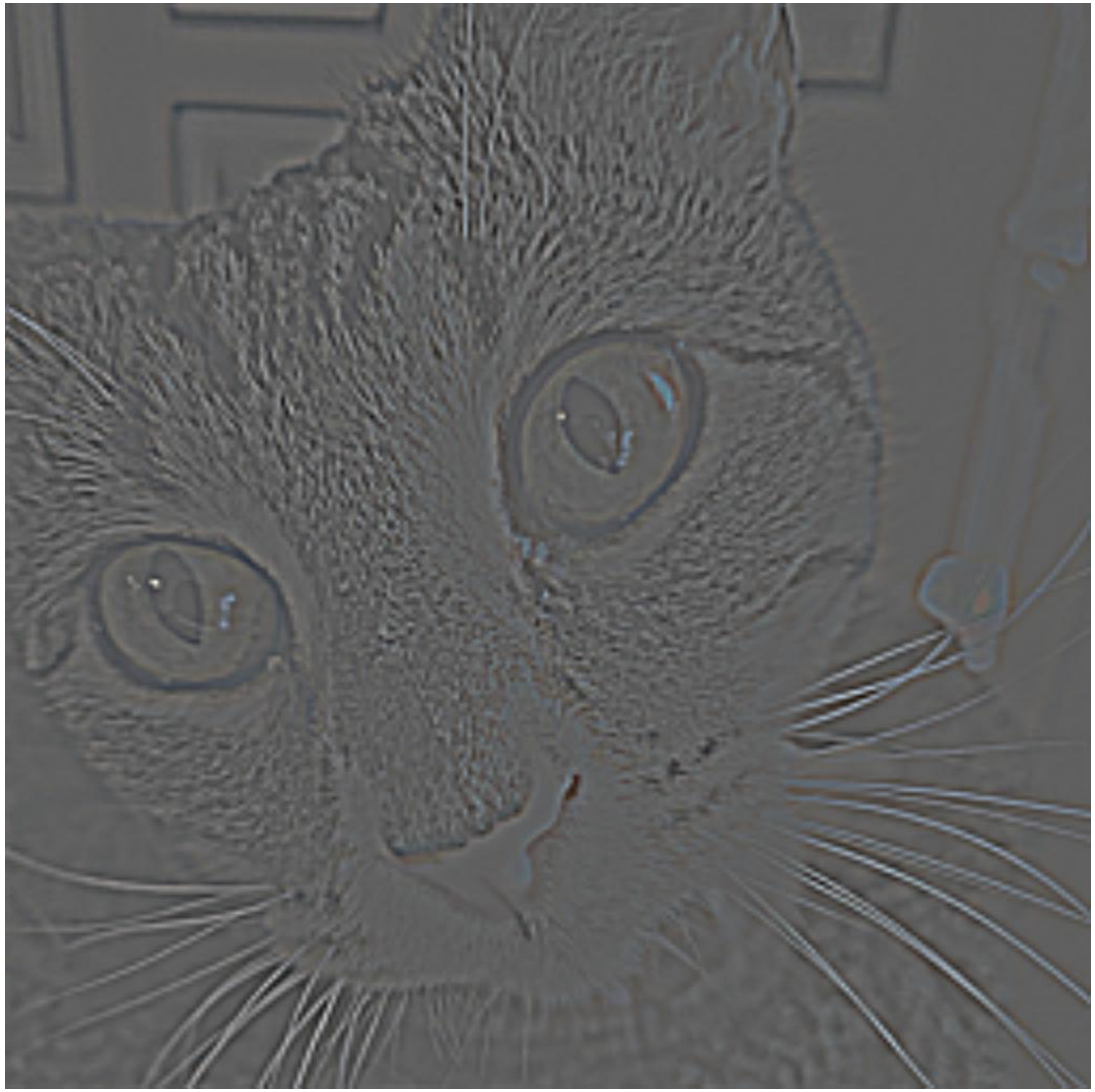
Laplacian pyramid



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

(upsampled back to full res for visualization)

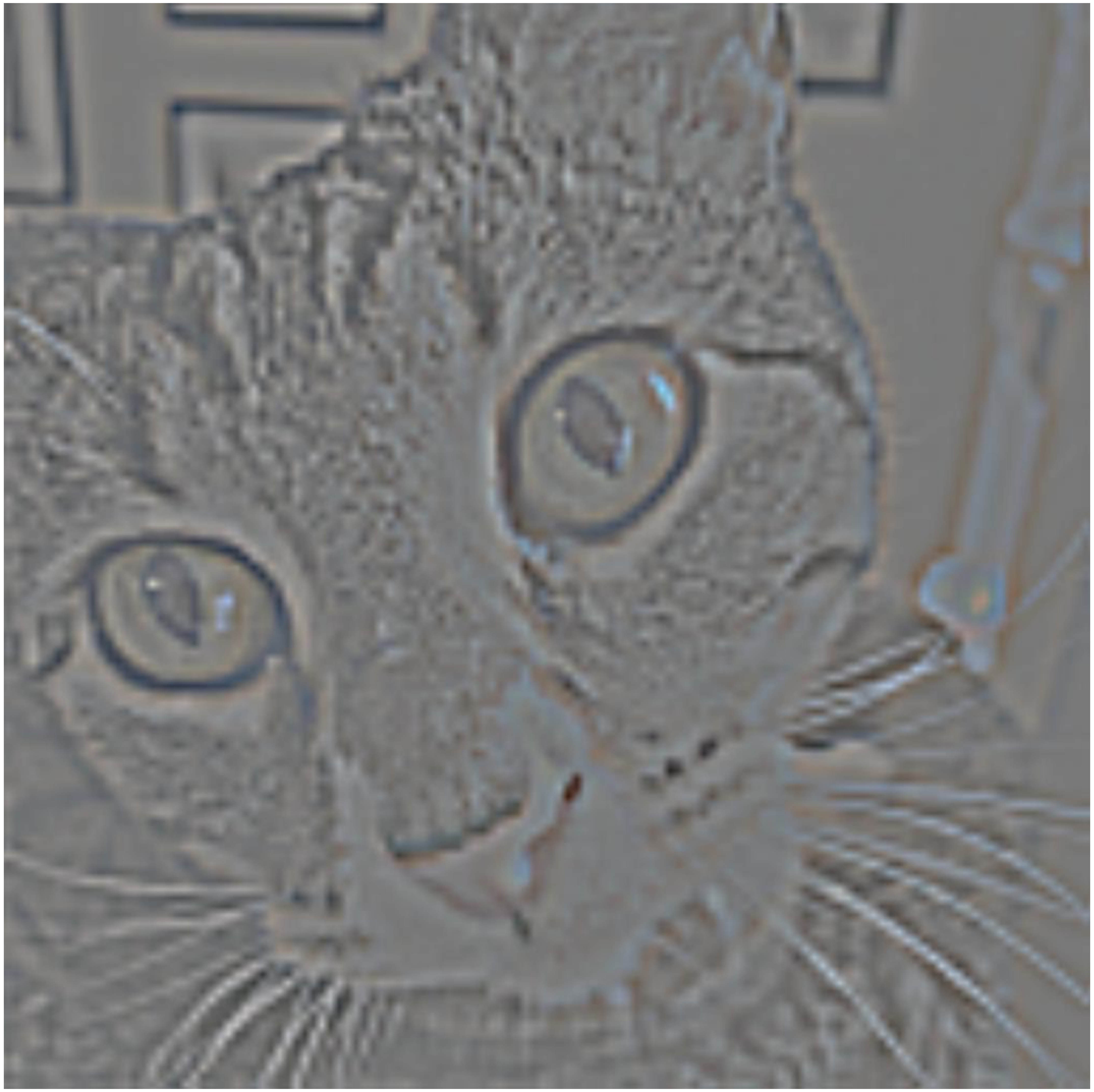
Laplacian pyramid



$L_2 = G_2 - up(G_3)$

(upsampled back to full res for visualization)

Laplacian pyramid



$L_3 = G_3$ - up(G_4)

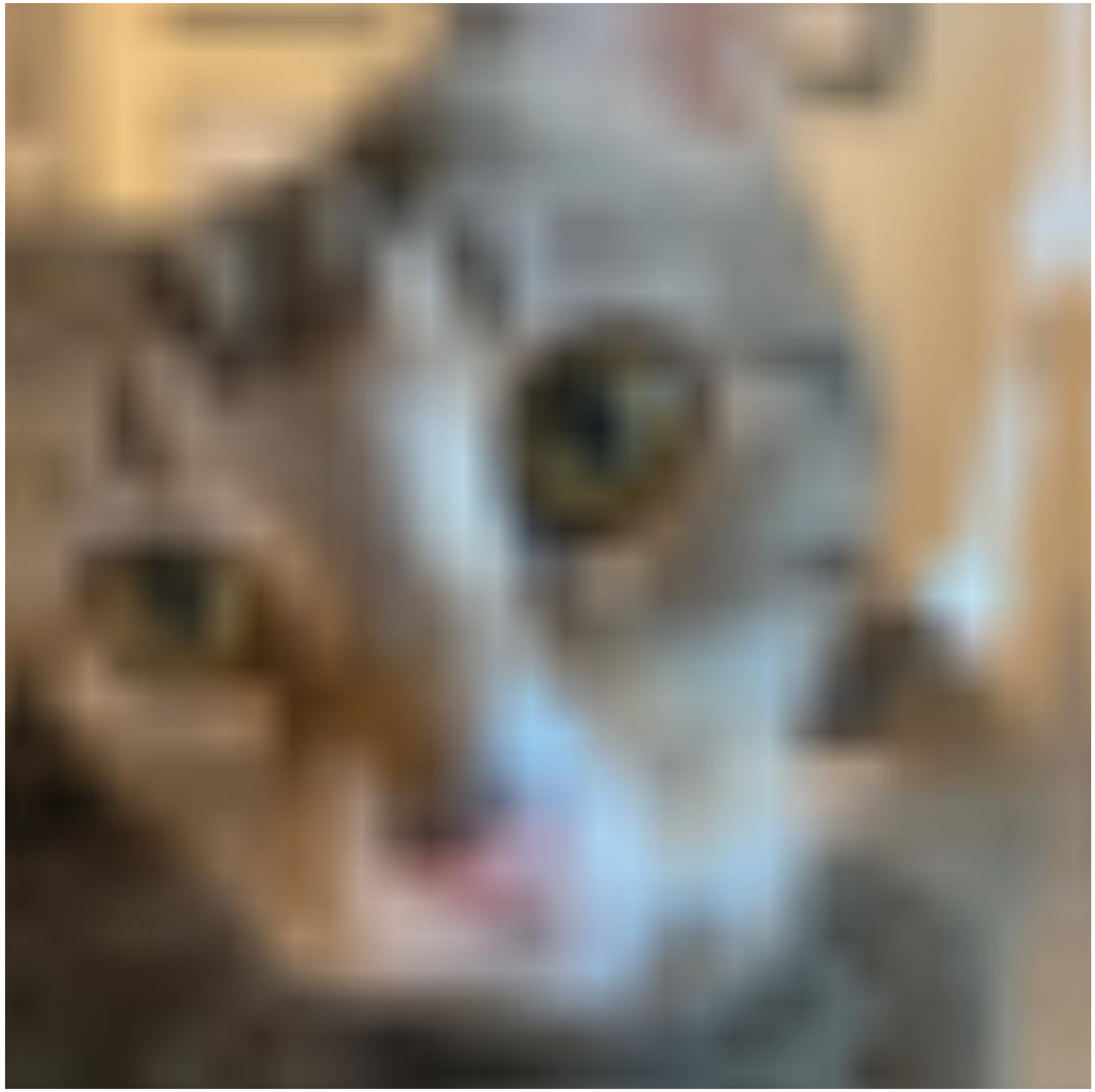
(upsampled back to full res for visualization)

Laplacian pyramid

$L_4 = G_4$ - up(G_5)

(upsampled back to full res for visualization)

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

A digital camera processing pipeline

Main theme…

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

Image processing computations are now a fundamental aspect of producing high-quality pictures from commodity cameras.

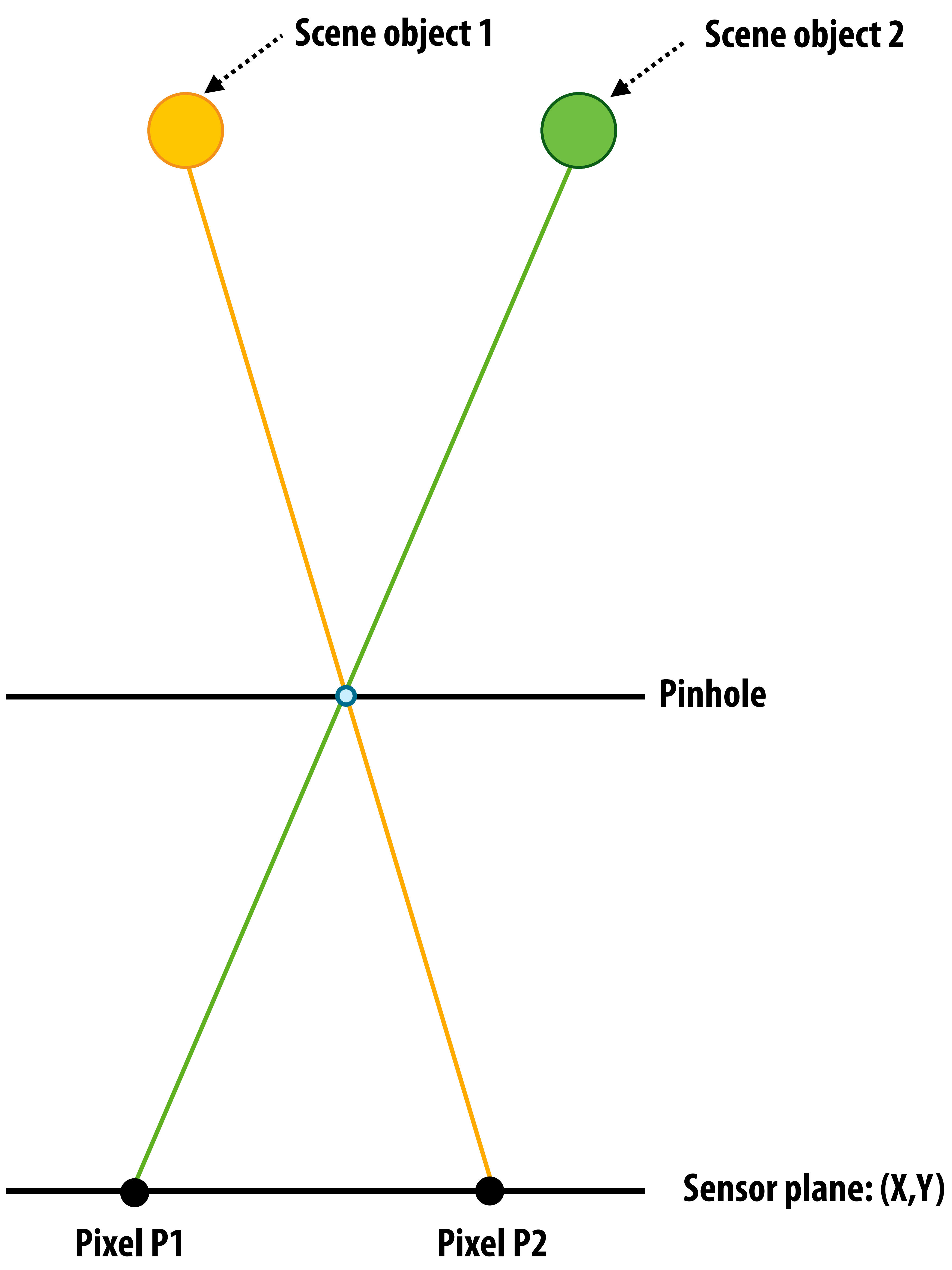
Sensor output ("RAW")

Computation

Beautiful image that impresses your friends on Instagram

Recall: pinhole camera (no lens)

(every pixel measures light intensity along ray of light passing through pinhole and arriving at pixel)



Camera with a lens

Camera with a large (zoom) lens

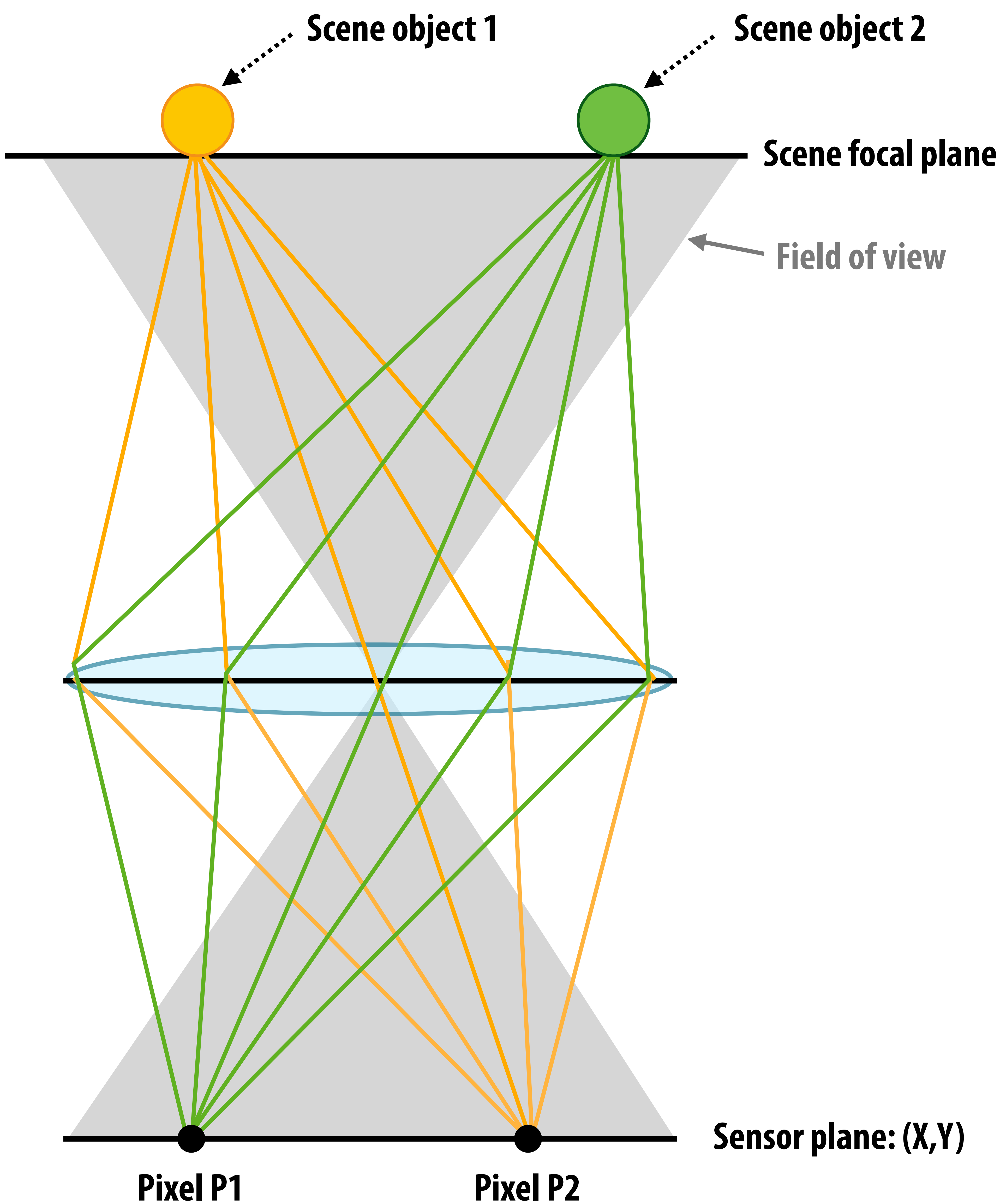
Cell phone camera lens(es)

What does a lens do?

Camera with lens:

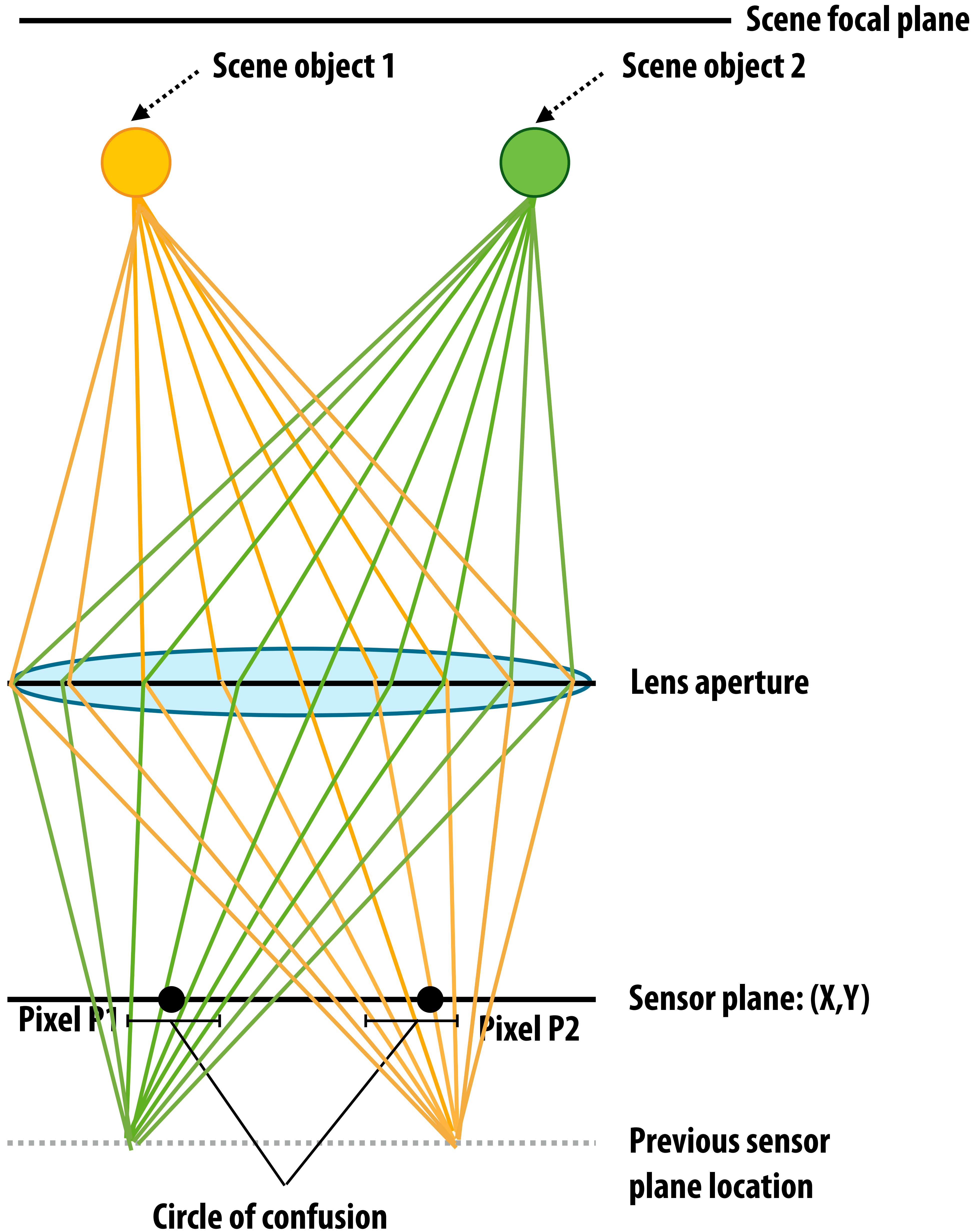
Every pixel accumulates all rays of light passing through lens aperture and refracted to location of pixel

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



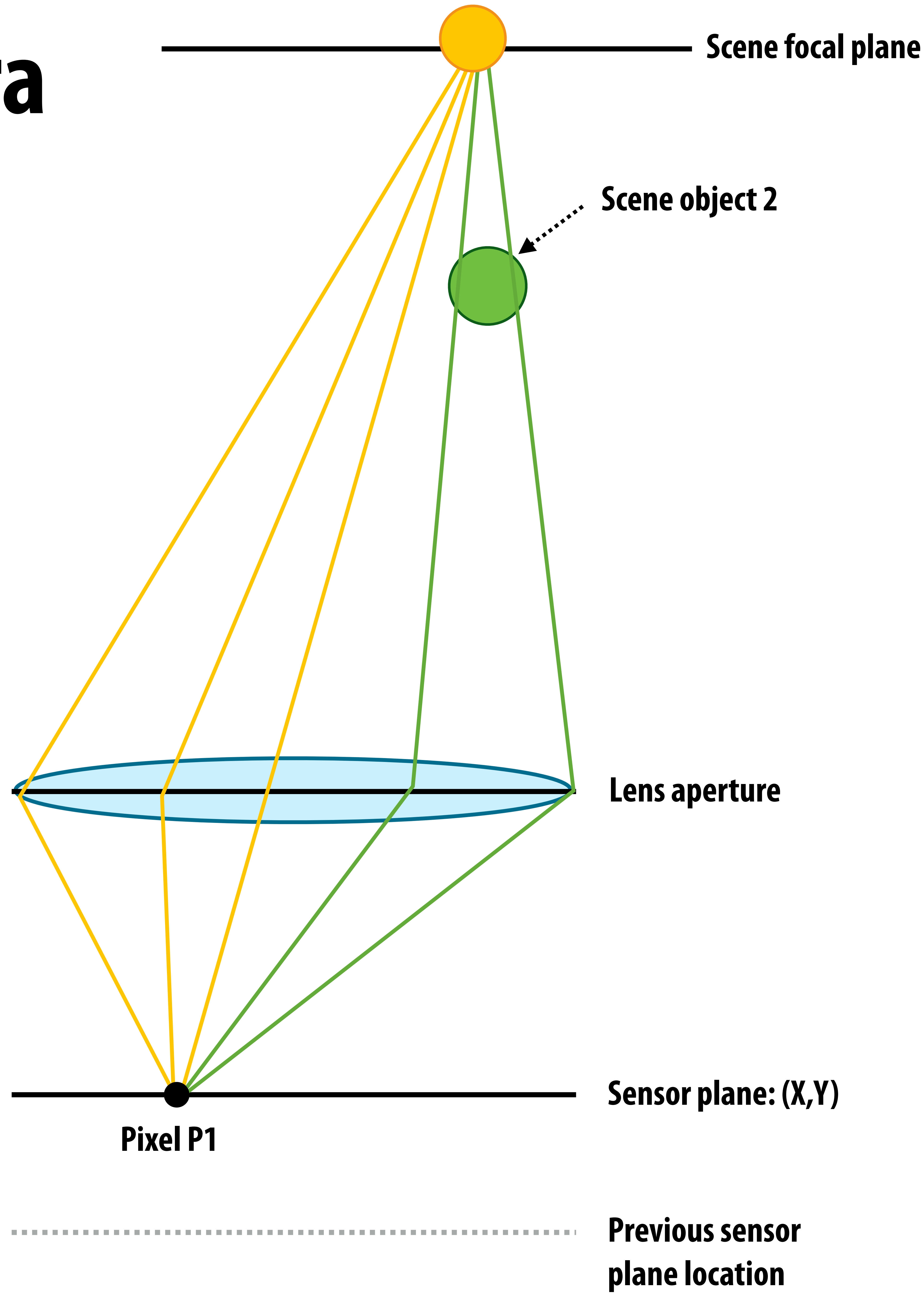
Bokeh

Out of focus camera

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

=

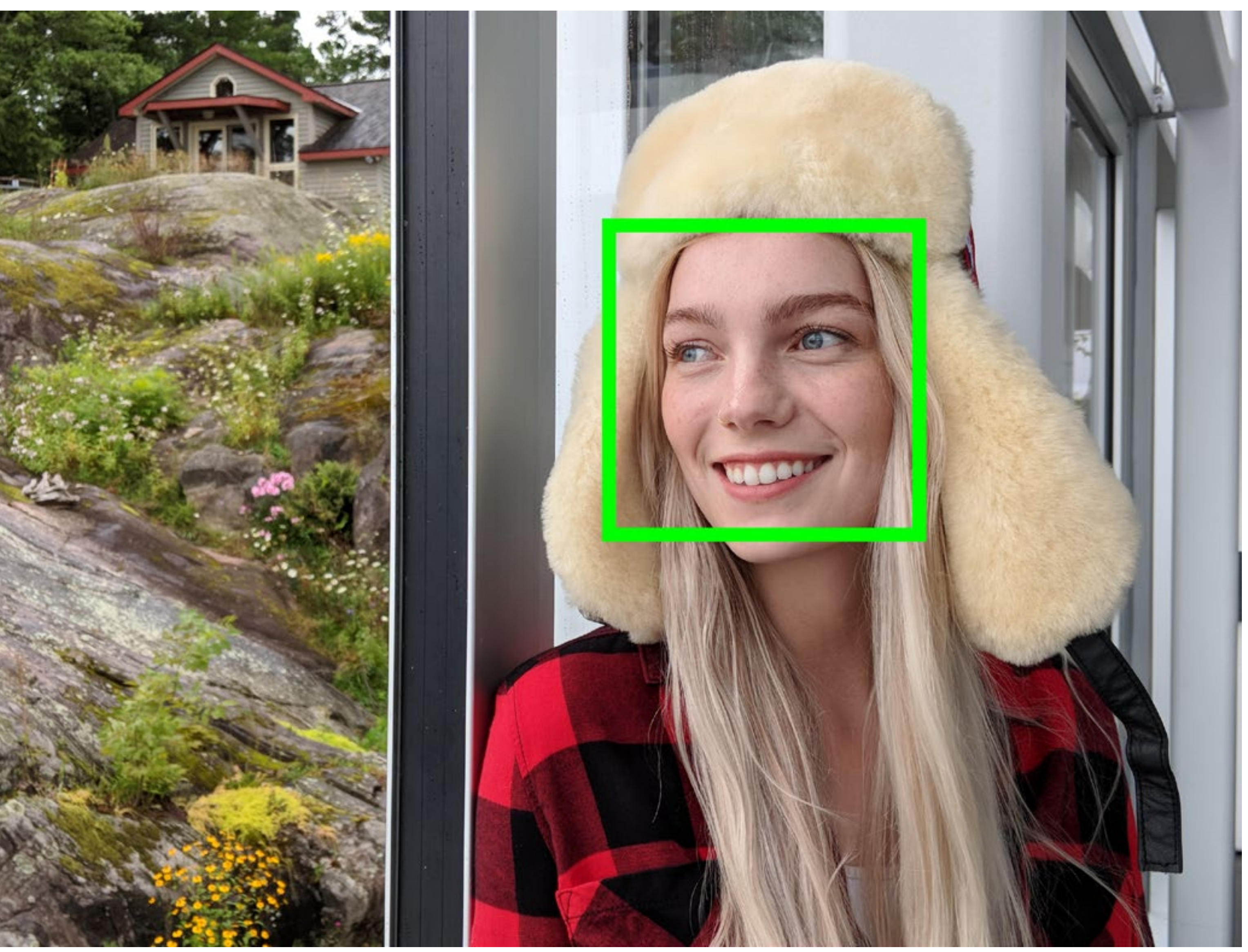
Rays of light from different scene points converge at single point on sensor



Sharp foreground / blurry background

“Portrait mode” (fake depth of field)

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



Input image /w detected face

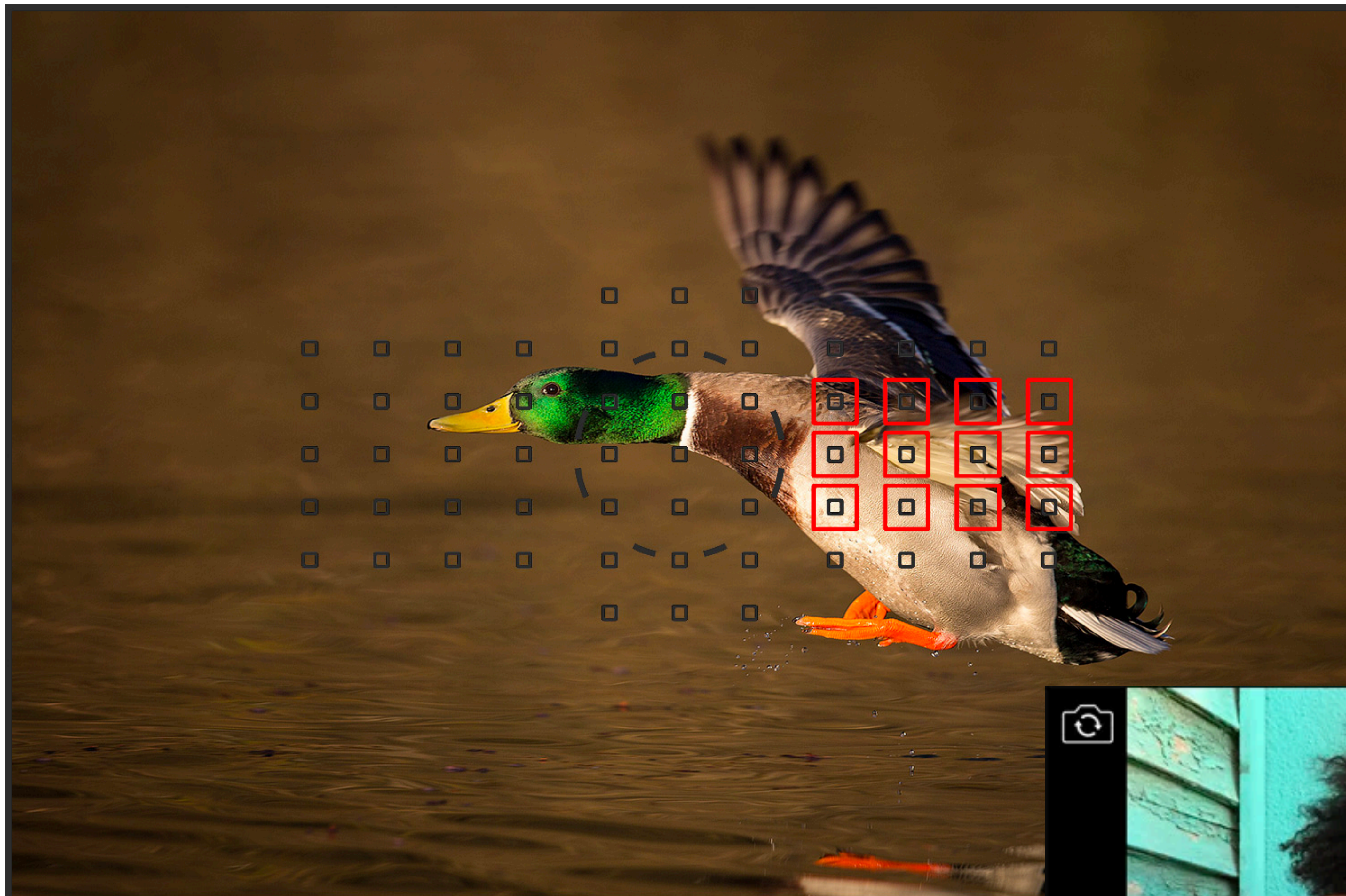
Segmentation

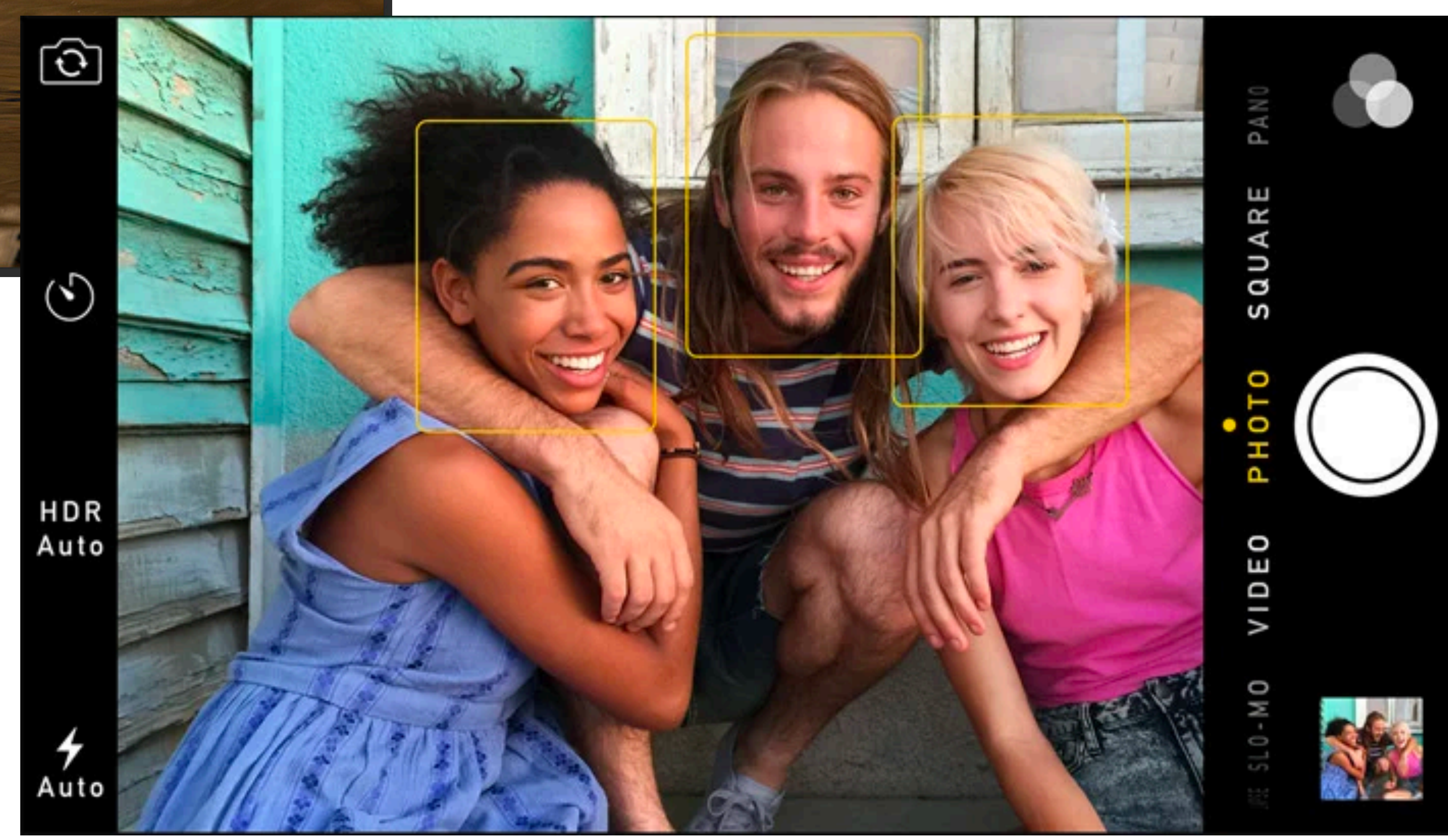
Scene Depth Estimate

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

Image credit: [Wadha 2018]

What part of image should be in focus?





Heuristics:

Focus on closest scene region

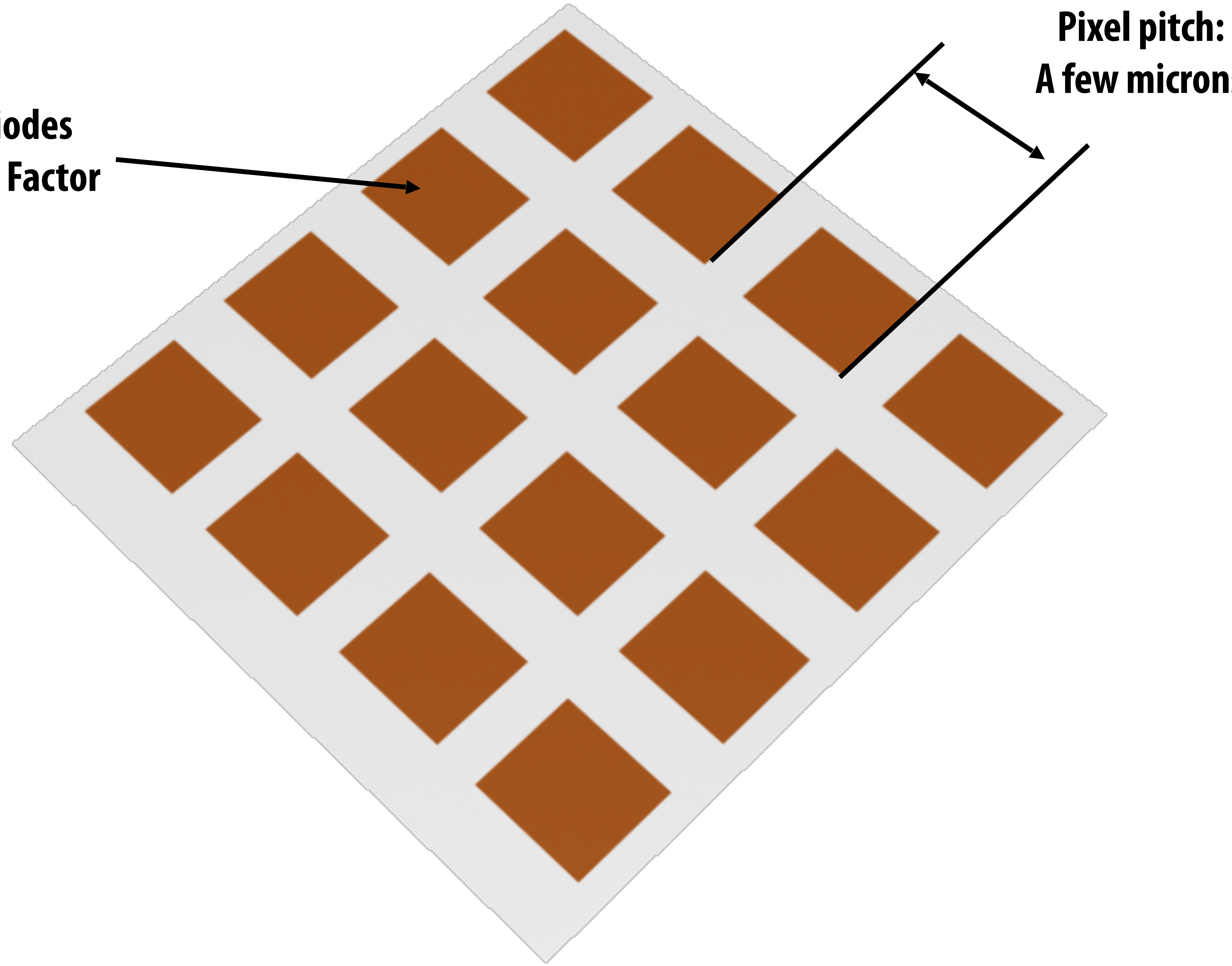
Put center of image in focus

Detect faces and focus on closest/largest face

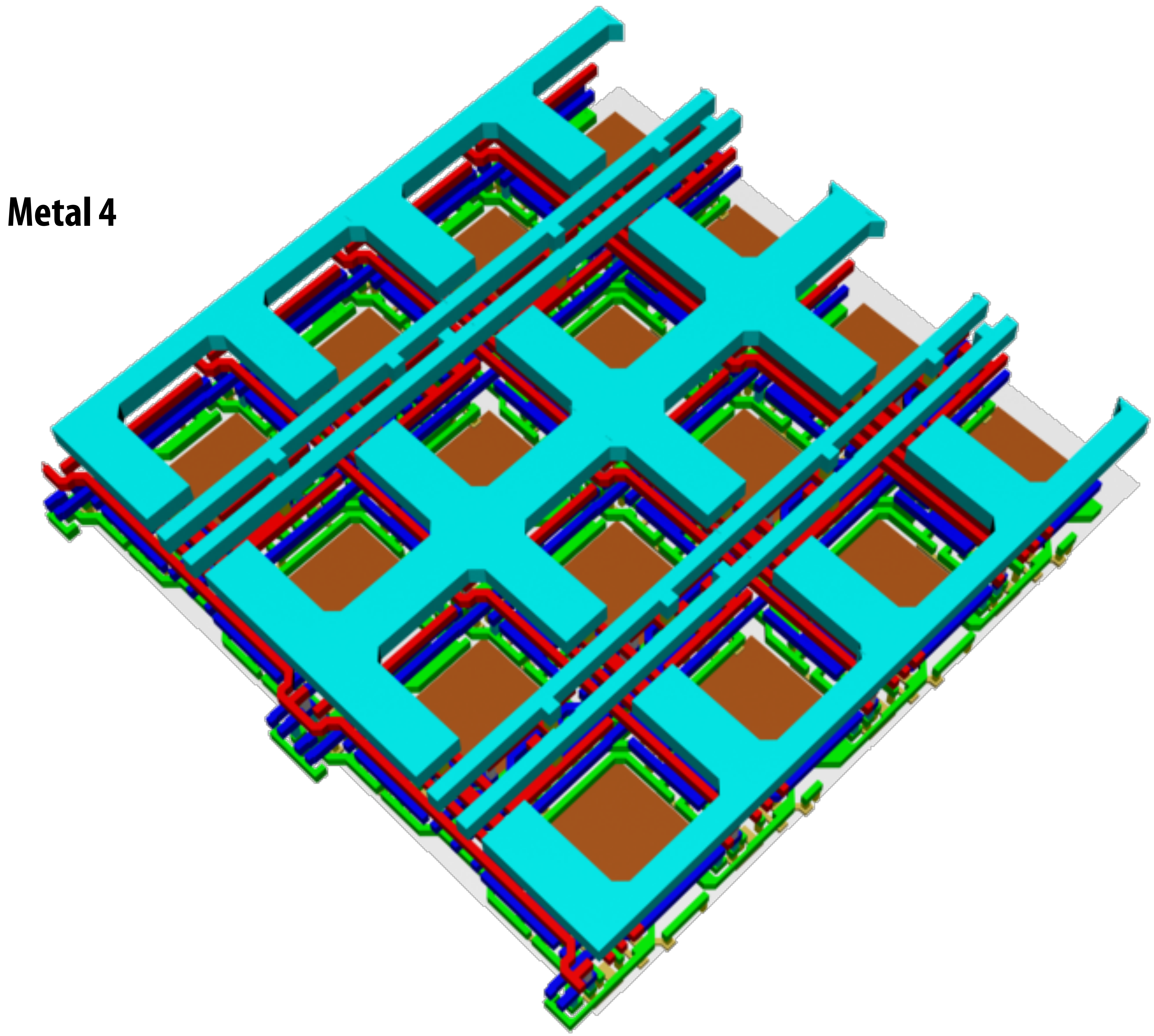
Image credit: DPReview:
https://www.dpreview.com/articles/9174241280/configuring-your-5d-mark-iii-af-for-fast-action

The Sensor

Front-side-illuminated (FSI) CMOS

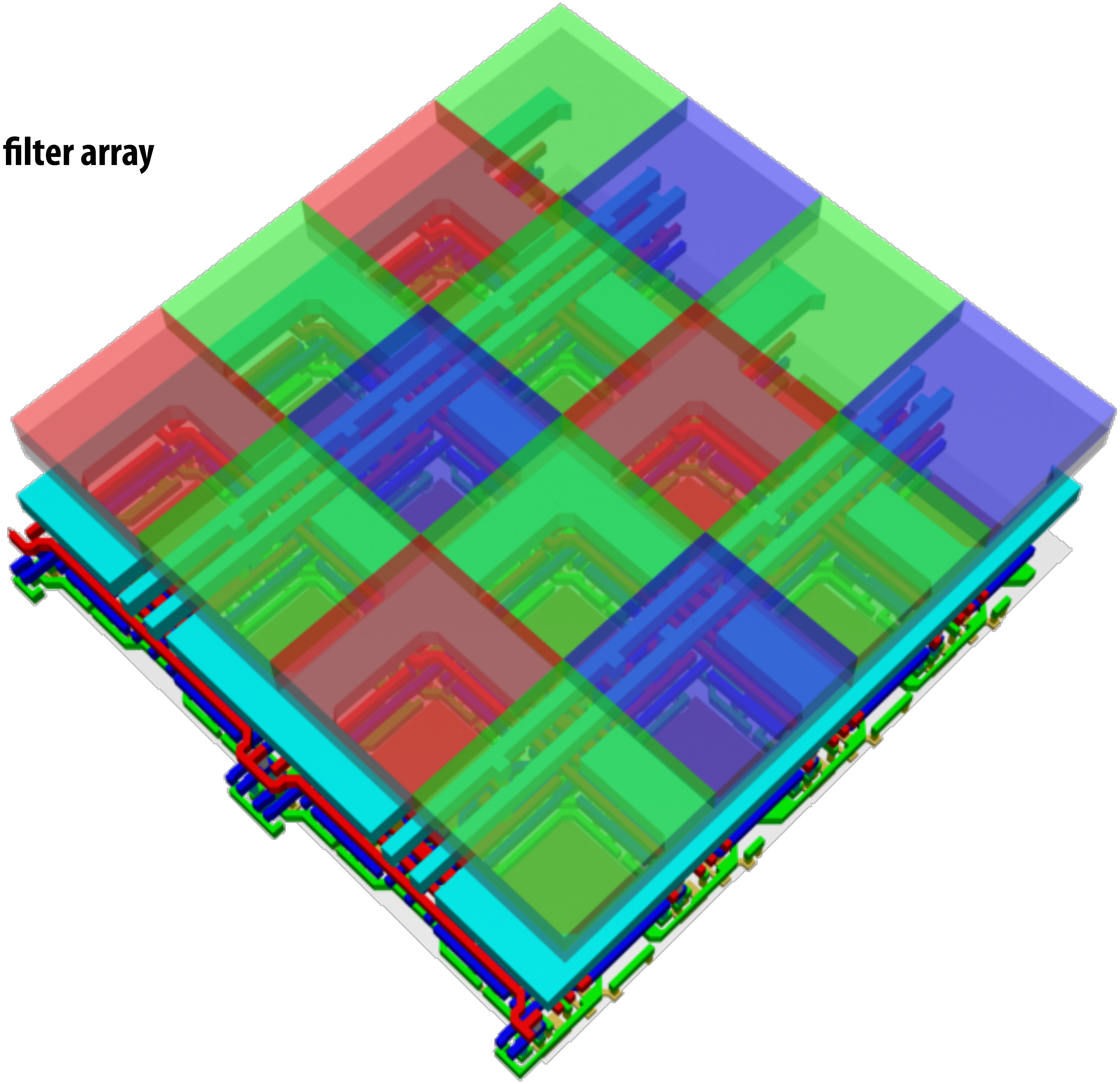


Courtesy R. Motta, Pixim



Courtesy R. Motta, Pixim

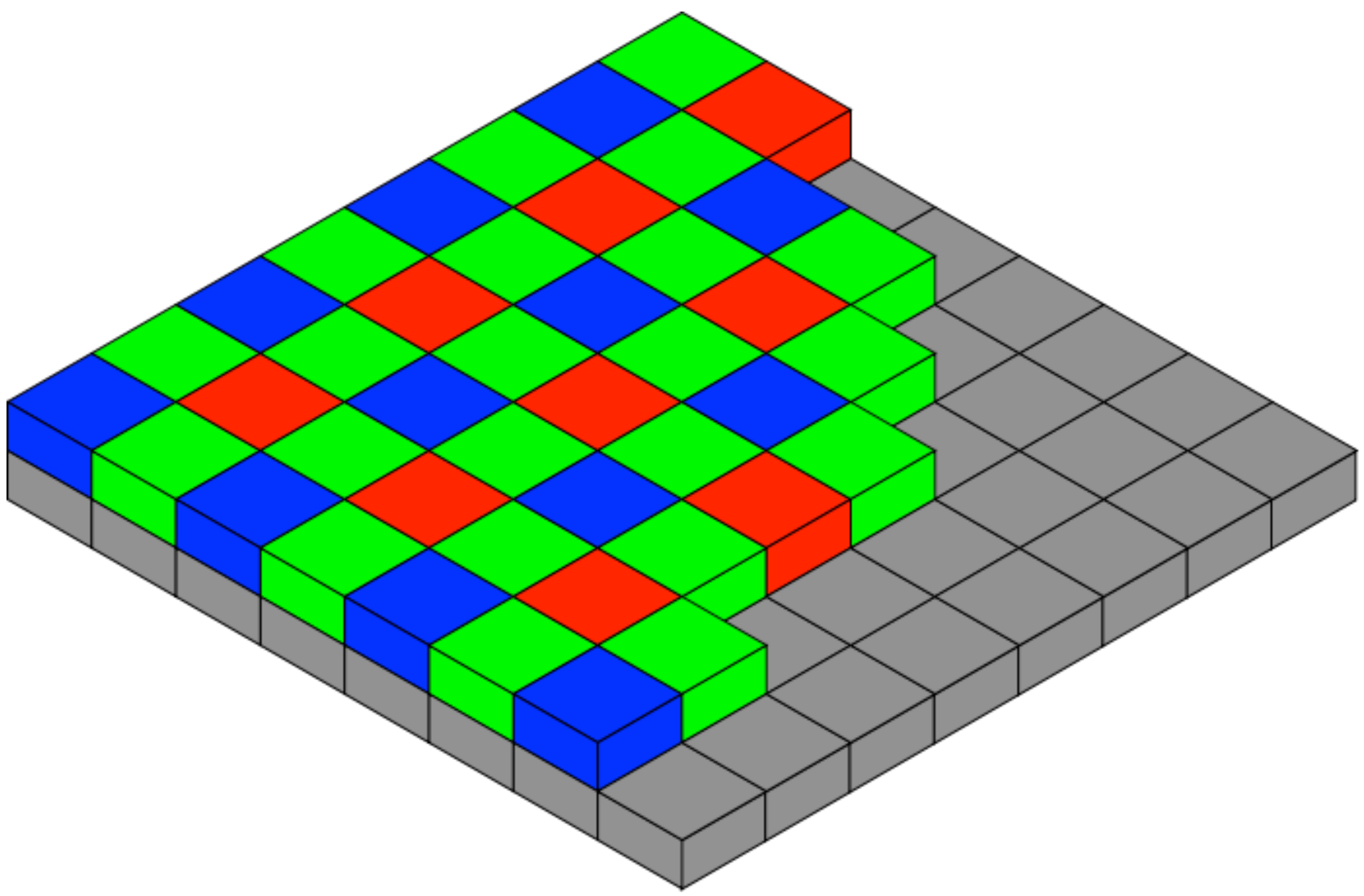
Color filter array



Courtesy R. Motta, Pixim

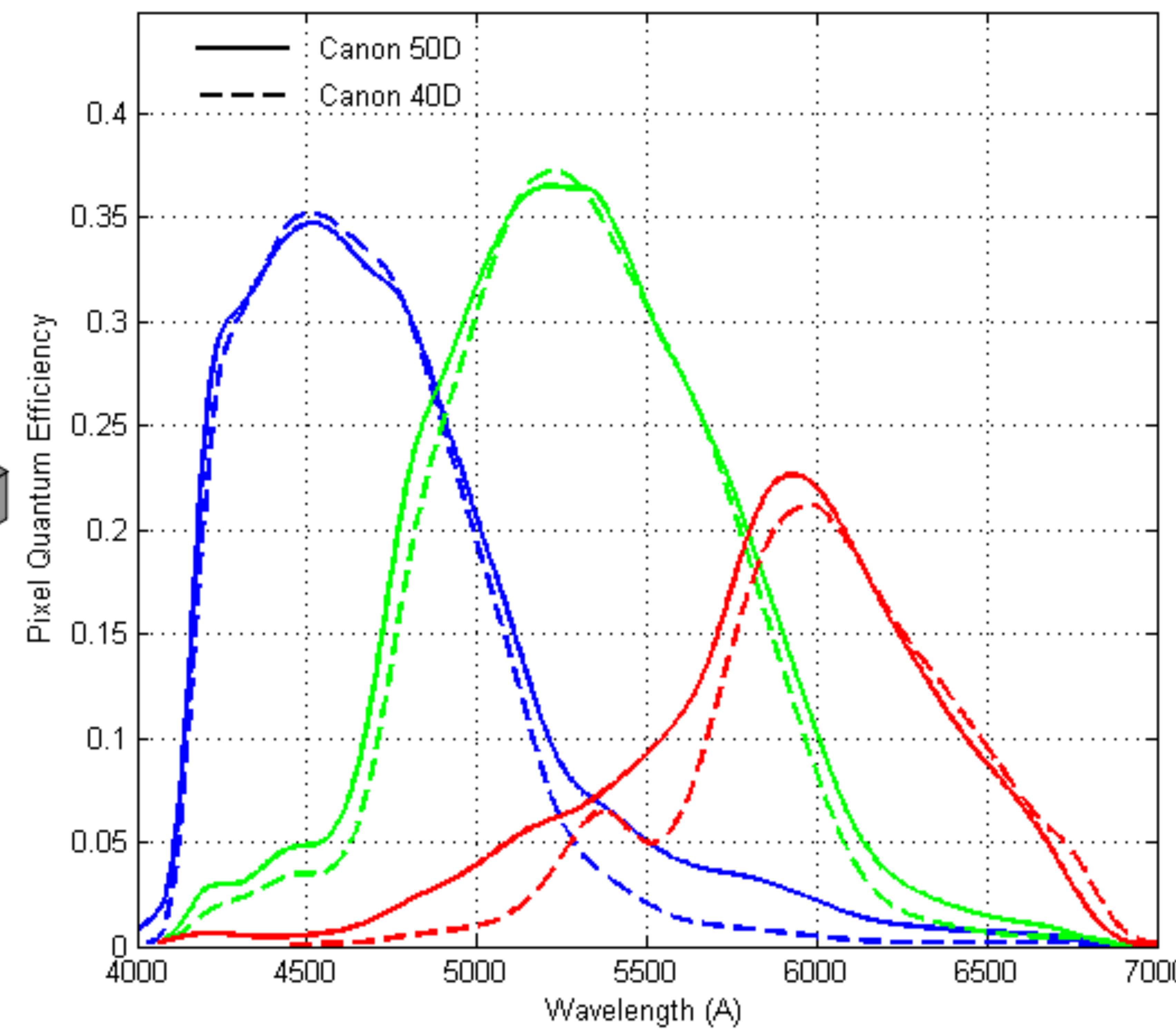
Digital image sensor: color filter array (Bayer mosaic)

- Color filter array placed over sensor
- Result: different pixels have different spectral response (each pixel measures red, green, or blue light)
- 50% of pixels are green pixels



Traditional Bayer mosaic
(other filter patterns exist: e.g., Sony's RGBE)

Pixel response curve: Canon 40D/50D



$$f(\lambda)$$

Image credit:
Wikipedia, Christian Buil (http://www.astrosurf.com/~buil/cameras.htm)

Demosiac

- Produce RGB image from mosaiced input image
- Basic algorithm: bilinear interpolation of mosaiced values (need 4 neighbors)
- More advanced algorithms:
  - Bicubic interpolation (wider filter support region… may overblur)
  - Good implementations attempt to find and preserve edges in photo

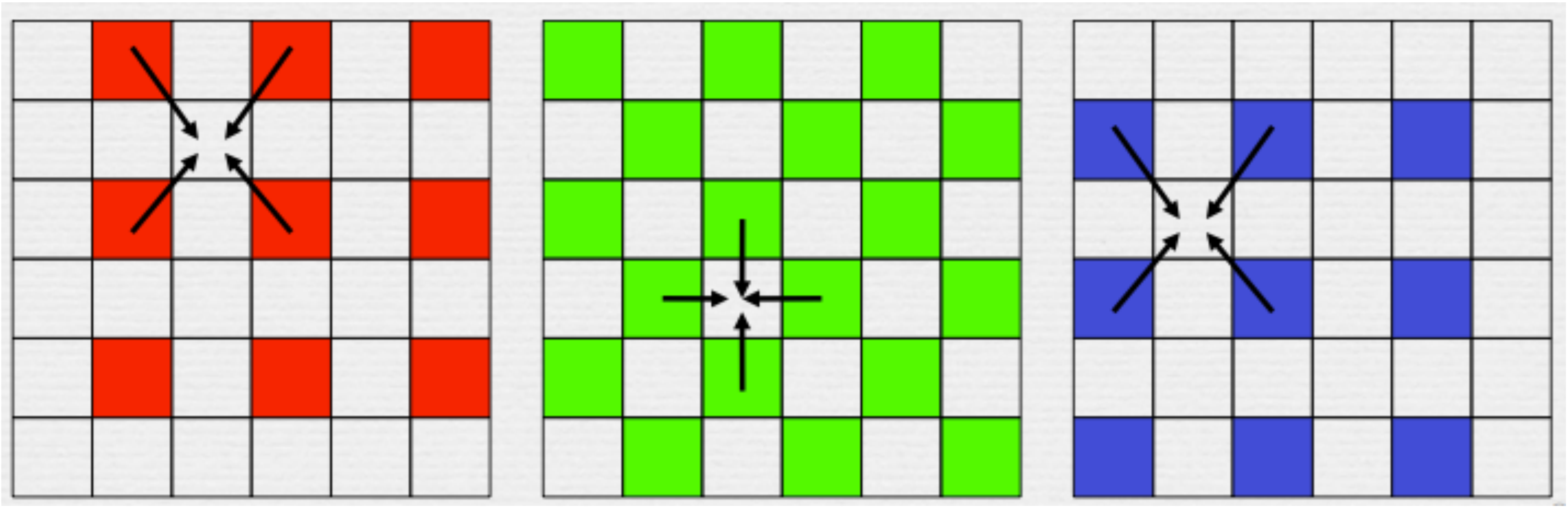
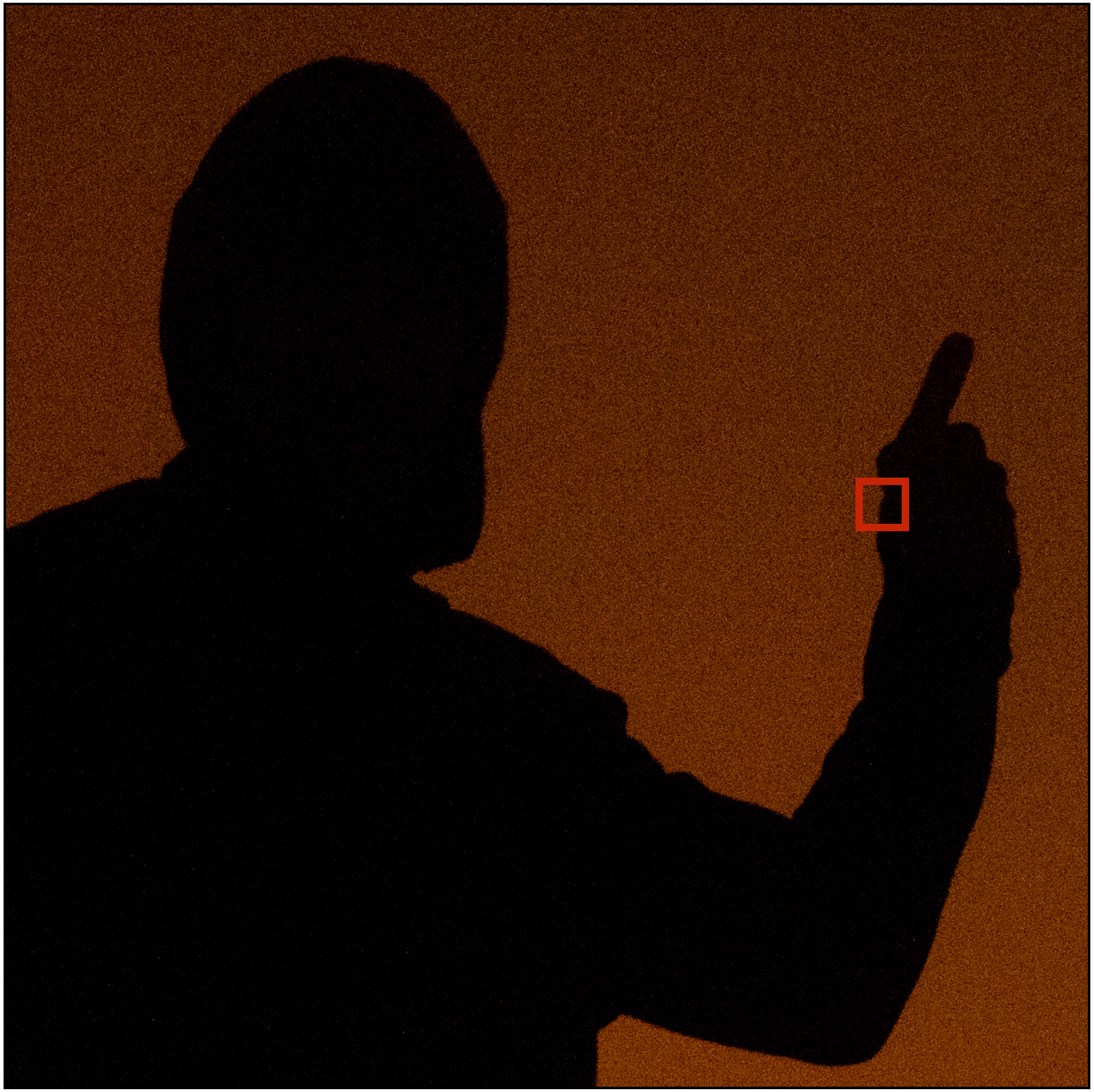
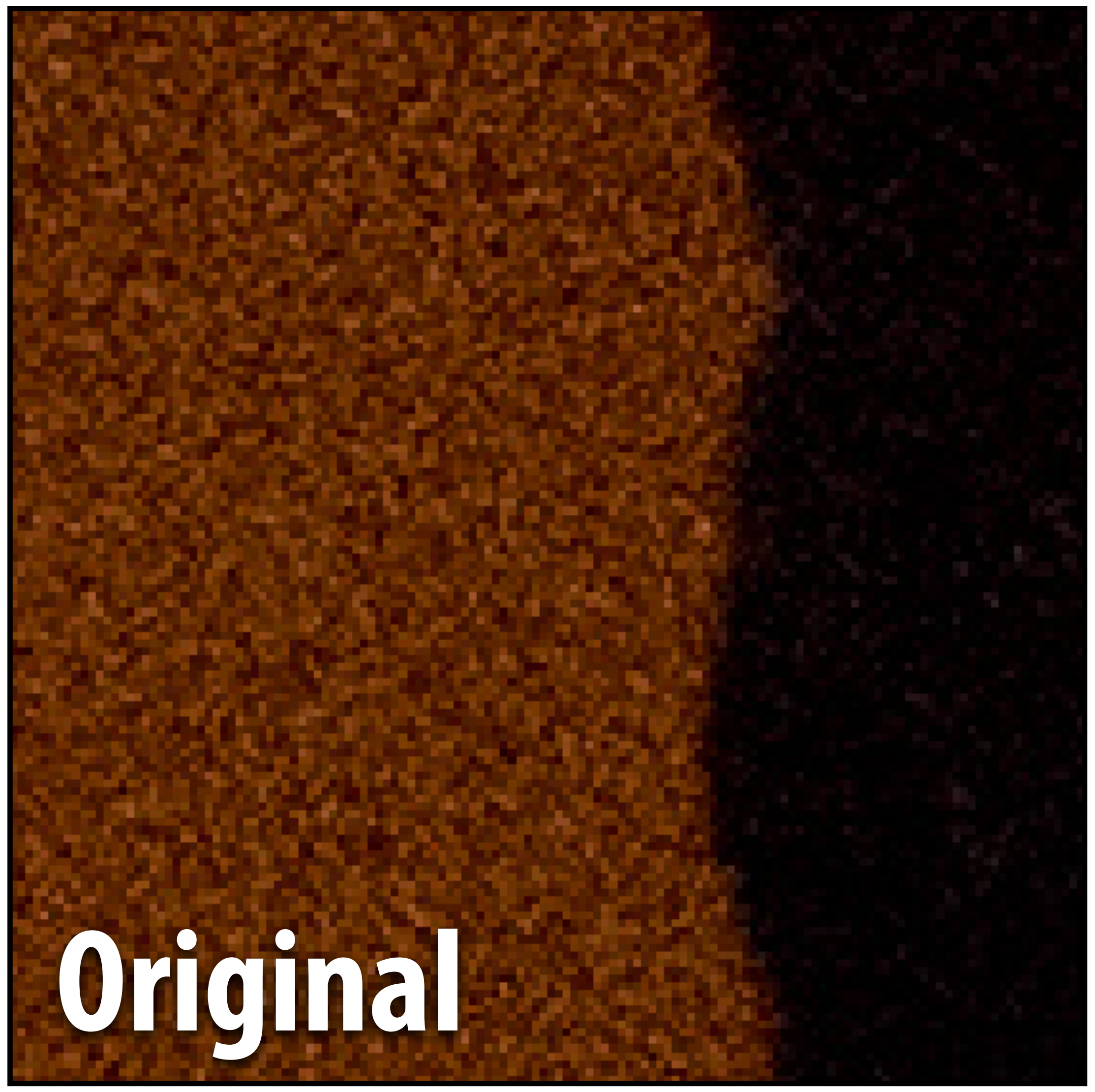


Image credit: Mark Levoy

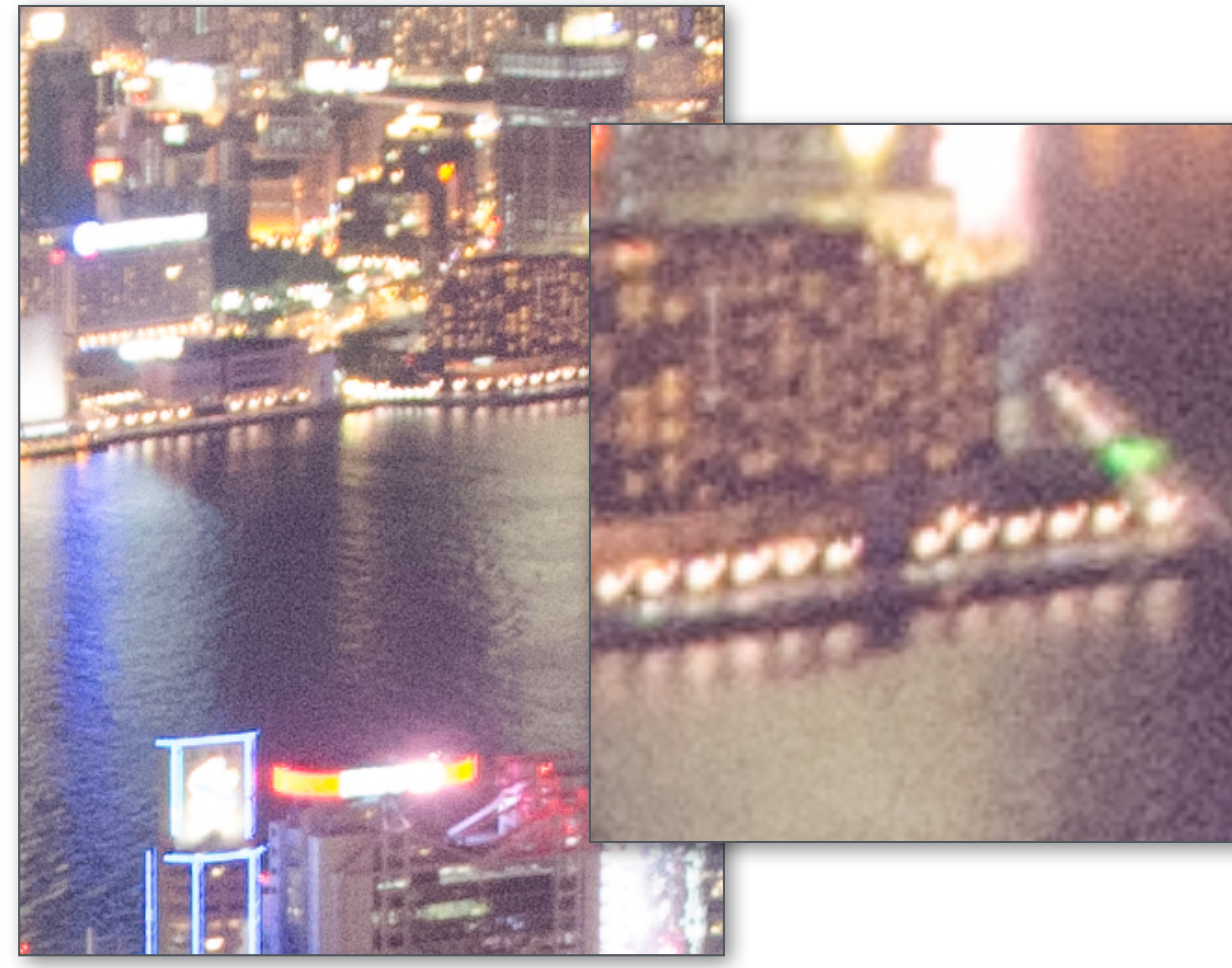
High dynamic range / exposure / noise

Denoising





Denoising via downsampling



Downsample via point sampling

(noise remains)

Downsample via averaging (bilinear resampling)

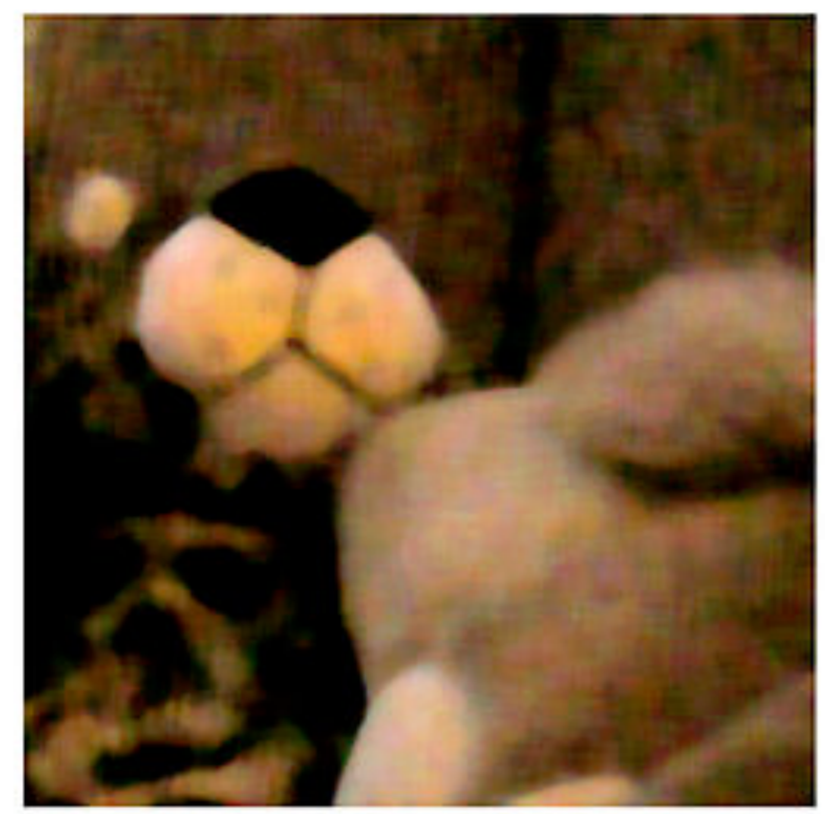
Noise reduced

Median filter

```
uint8 input[(WIDTH+2) * (HEIGHT+2)];
uint8 output[WIDTH * HEIGHT];
for (int j=0; j<HEIGHT; j++) {
   for (int i=0; i<WIDTH; i++) {
      output[j*WIDTH + i] =
           // compute median of pixels
           // in surrounding 5x5 pixel window
   }
}
```

original image

1px median filter



3px median filter

10px median filter

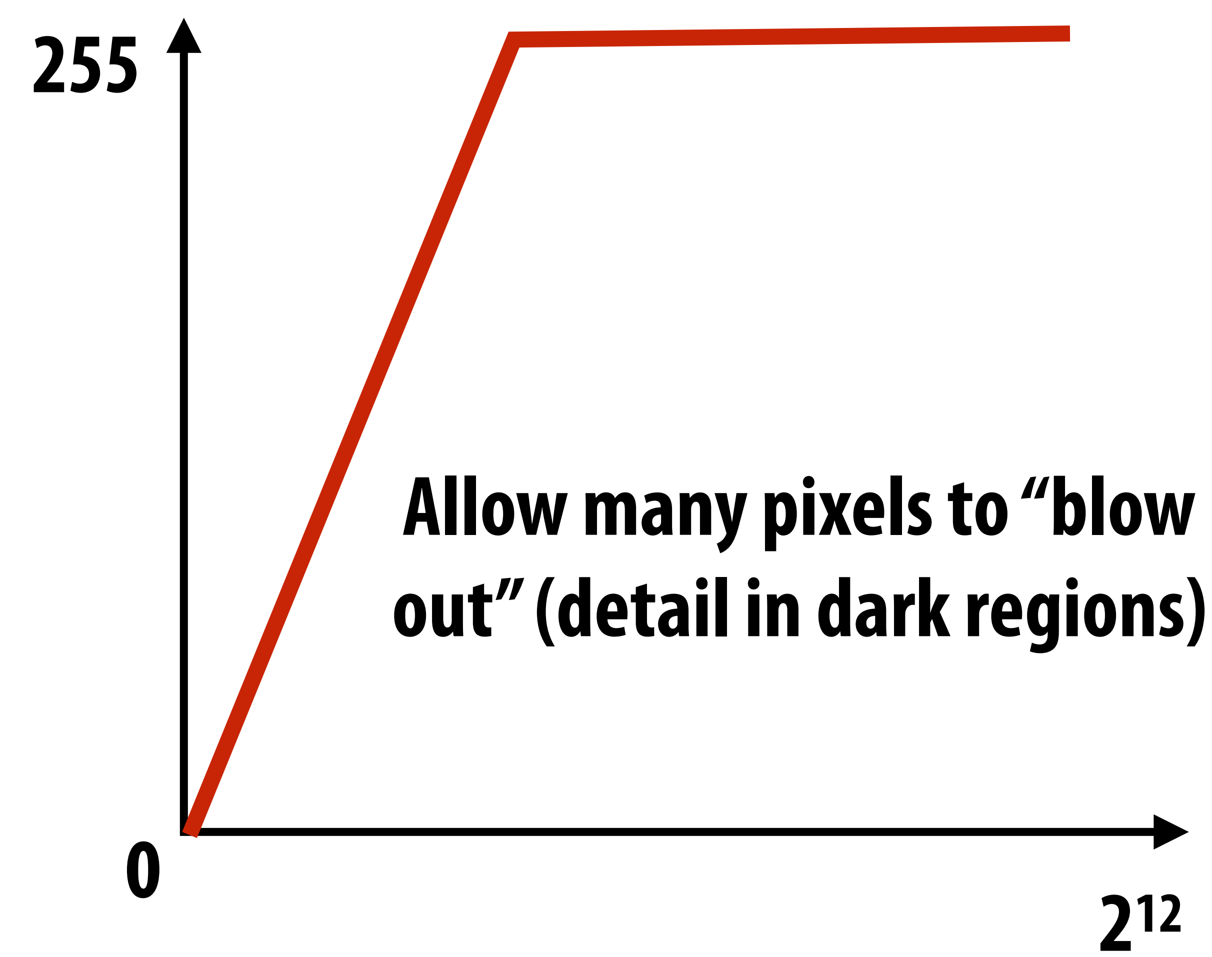
- **Replace pixel with median of its neighbors**
  - Useful noise reduction filter: unlike gaussian blur, one bright pixel doesn't drag up the average for entire region
- **Not linear, not separable**
  - Filter weights are 1 or 0 (depending on image content)
- **Basic algorithm for NxN support region:**
  - Sort N^2 elements in support region, then pick median: $O(N^2\log(N^2))$ work per pixel
  - Can you think of an $O(N^2)$ algorithm? What about $O(N)$?

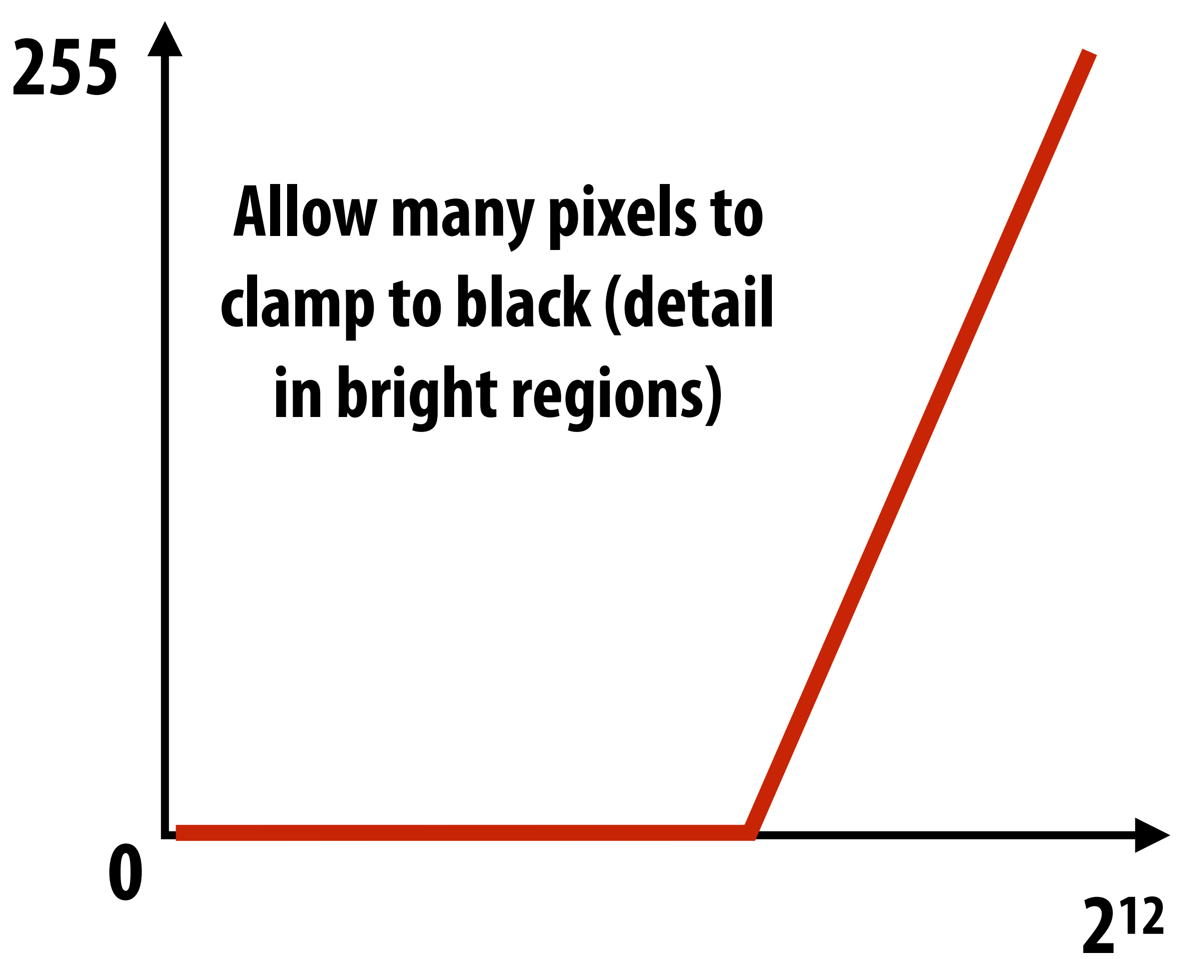
Saturated pixels

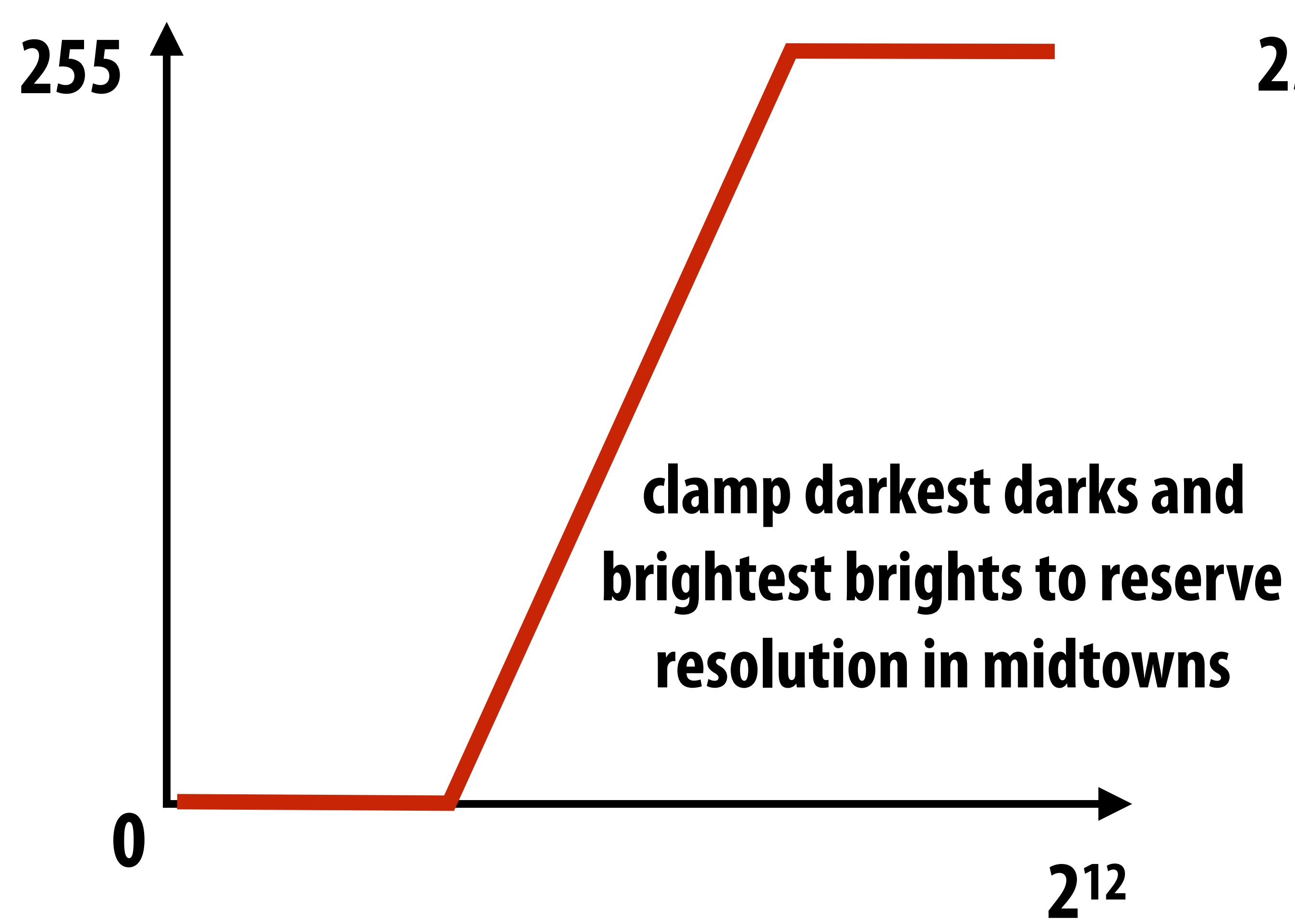
Global tone mapping

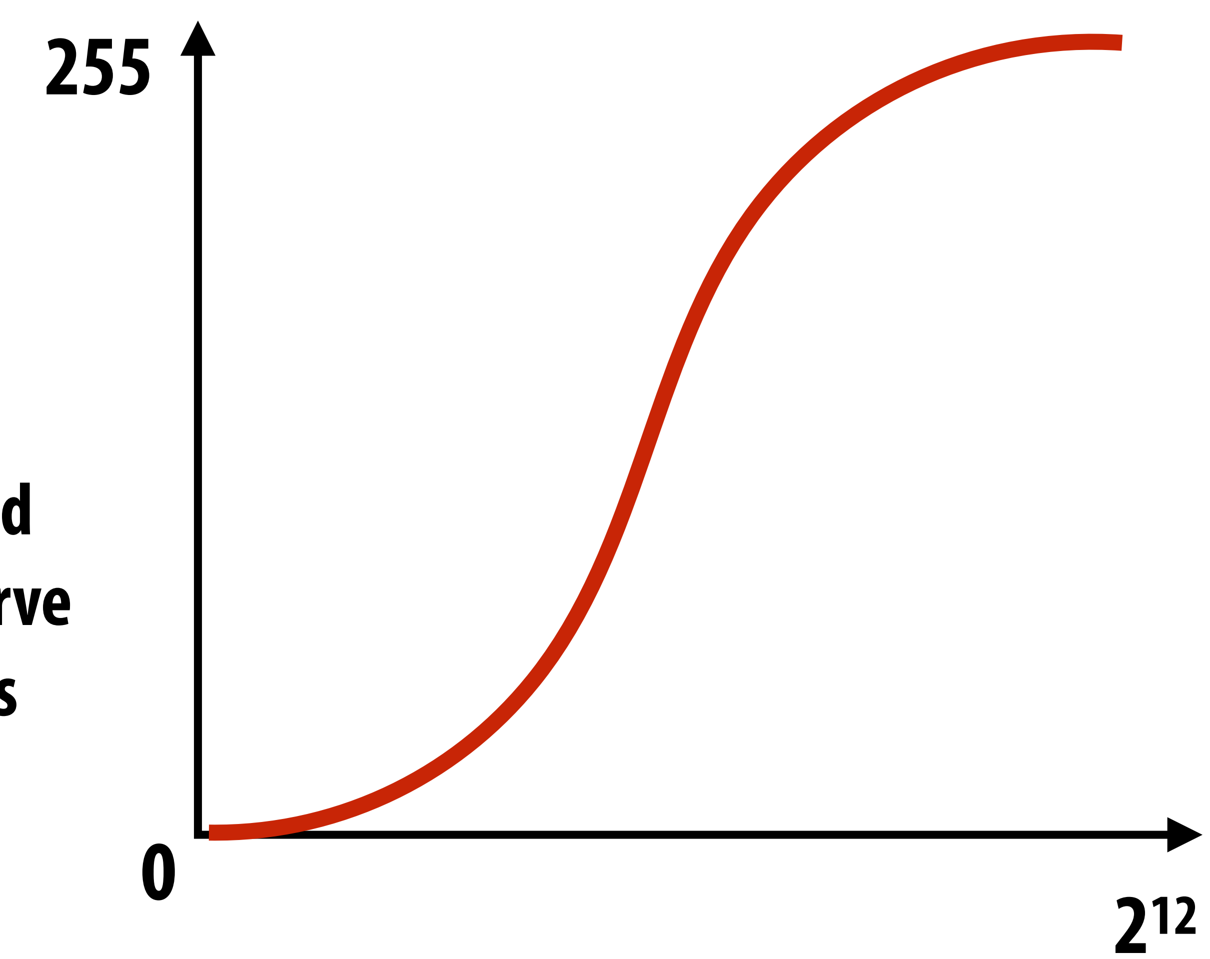
- Measured image values: 10-12 bits/pixel, but common image formats (8-bits/pixel)
- How to convert 12 bit number to 8 bit number?

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

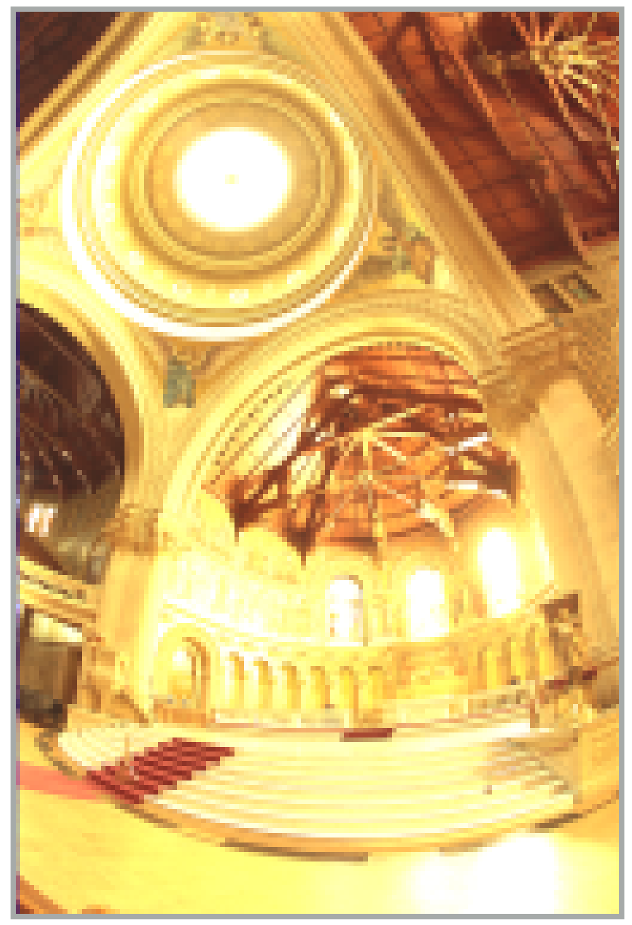


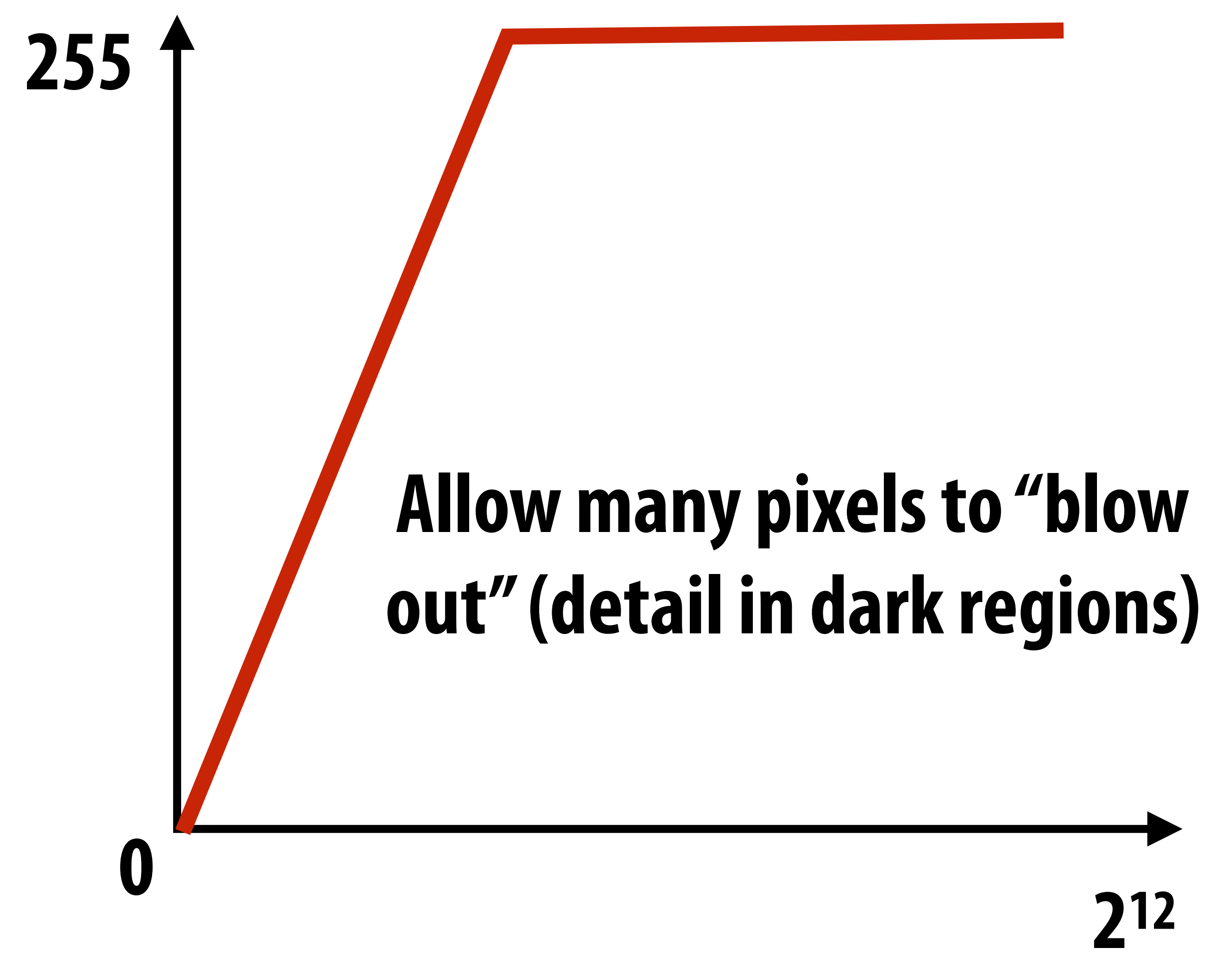


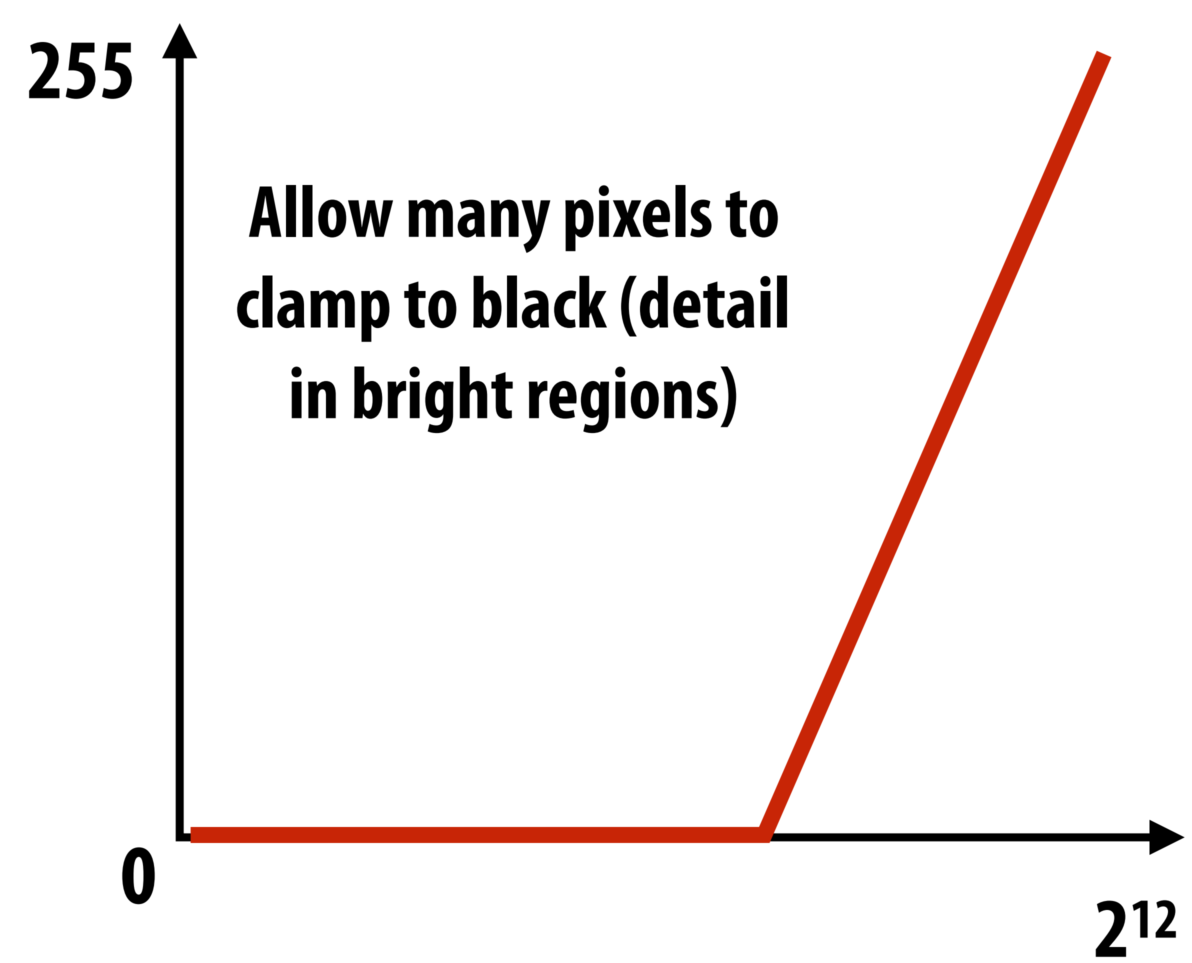




Global tone mapping

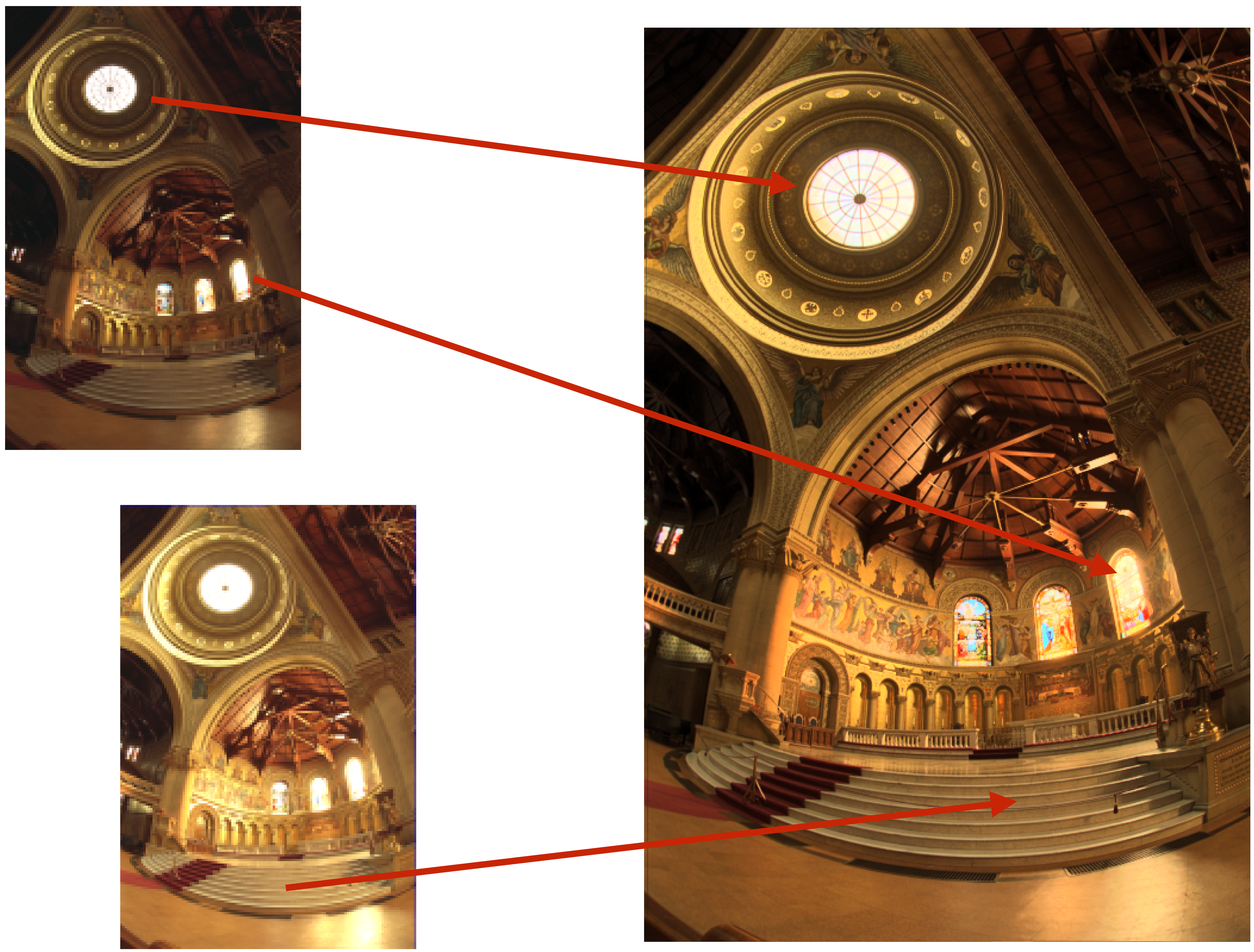






Local tone mapping

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



Local tone adjustment

Pixel values

Weight Masks

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

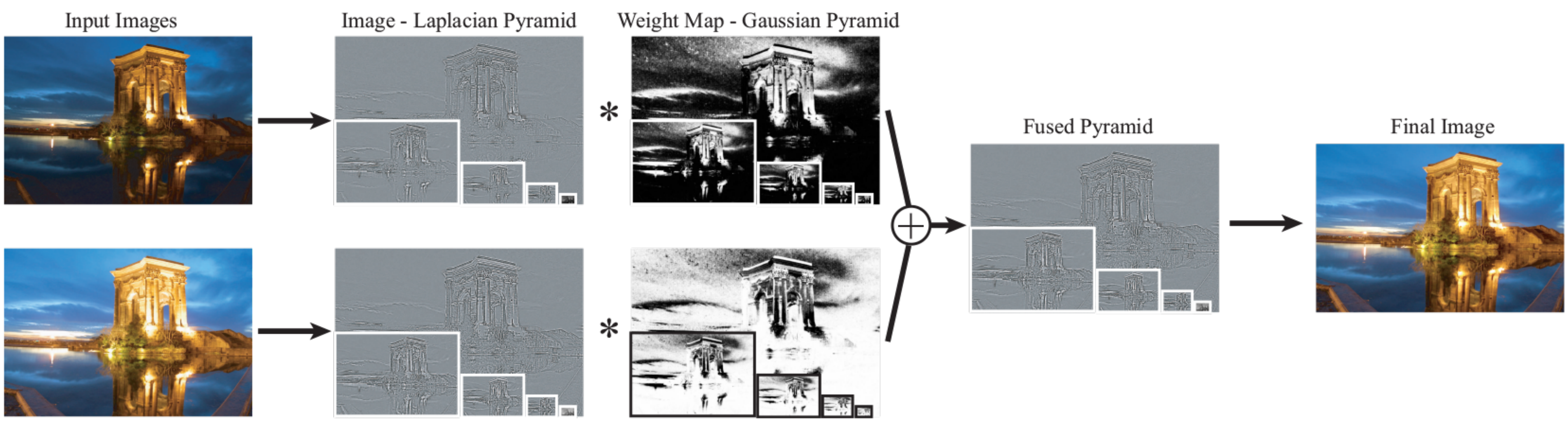
Four exposures (weight masks not shown)

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

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

Use of Laplacian pyramid in tone mapping

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



Challenges of merging images

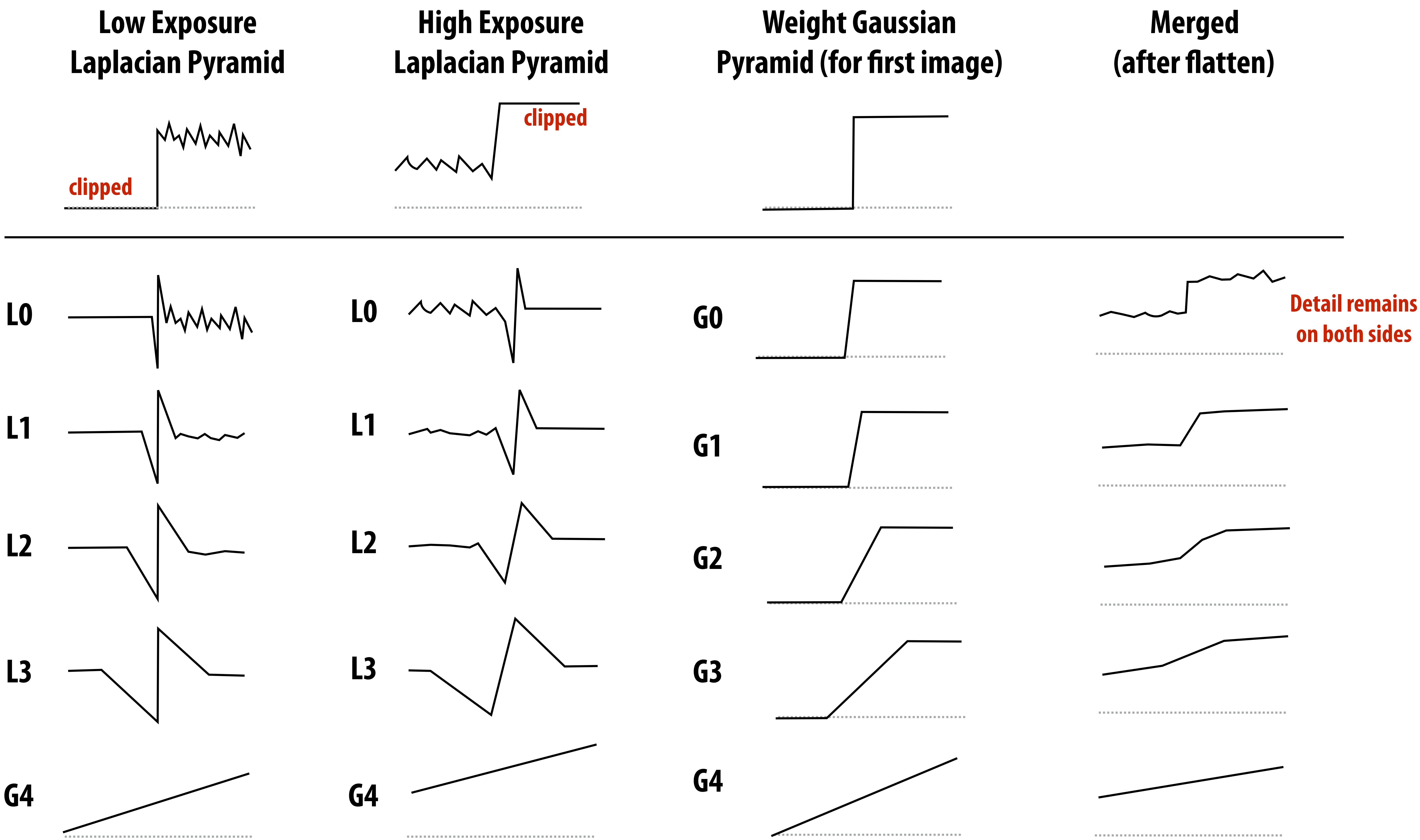
Four exposures (weights not shown)

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

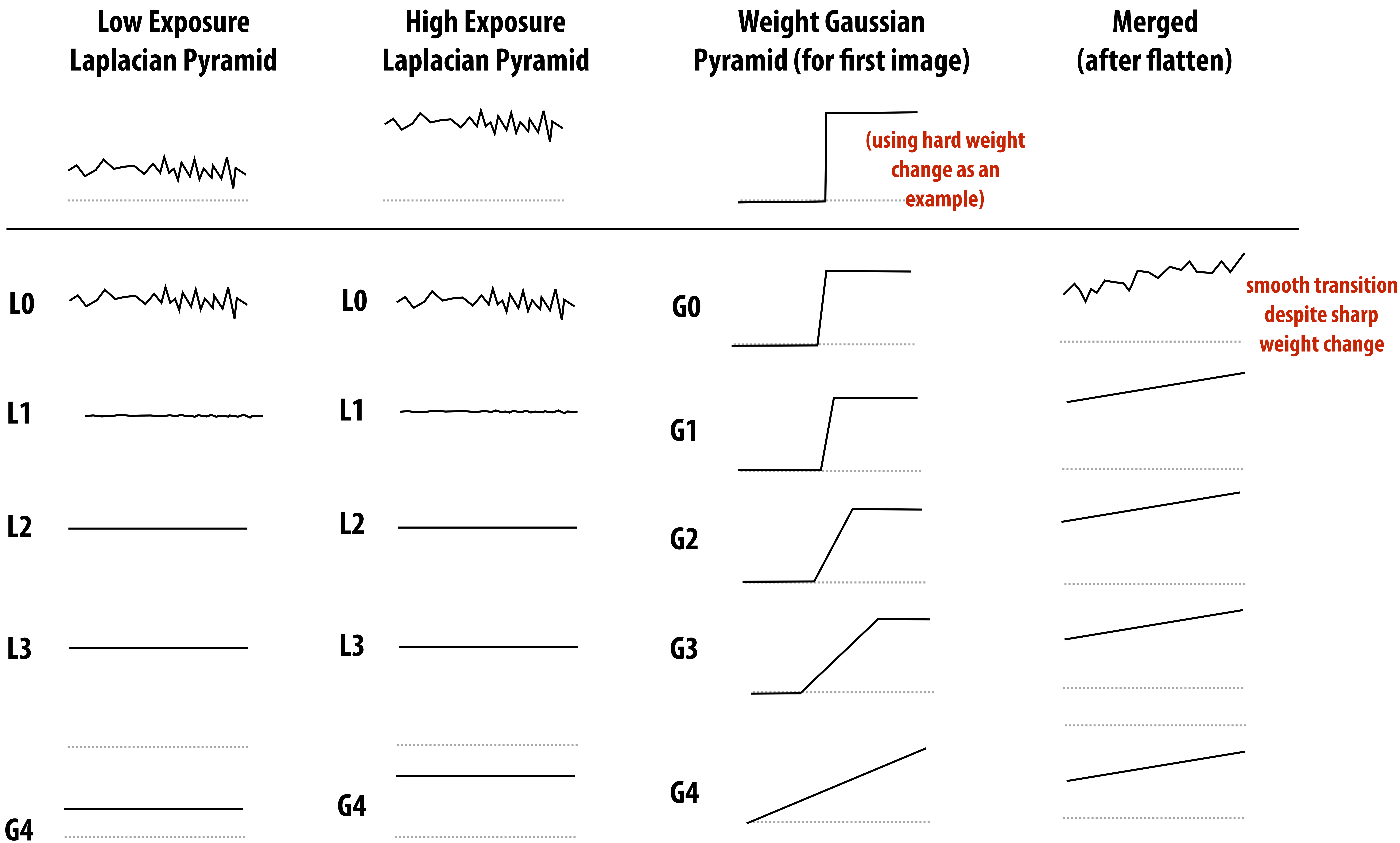
Merged result
(based on multi-resolution pyramid merge)

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

- Image processing is now a fundamental part of producing a pleasing photograph
- Used to compensate for physical constraints
  - Today: demosaic, tone mapping
  - Other examples not discussed today: denoise, lens distortion correction, etc.
- Used to determine how to configure camera (e.g., autofocus)
- Used to make non-physically plausible images that have aesthetic merit

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

Beautiful image that impresses your friends on Instagram