Lecture 15:

Image Processing for Digital Photography

Interactive Computer Graphics
Stanford CS248, Winter 2021

A review of image processing via convolution

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")

Gaussian blur

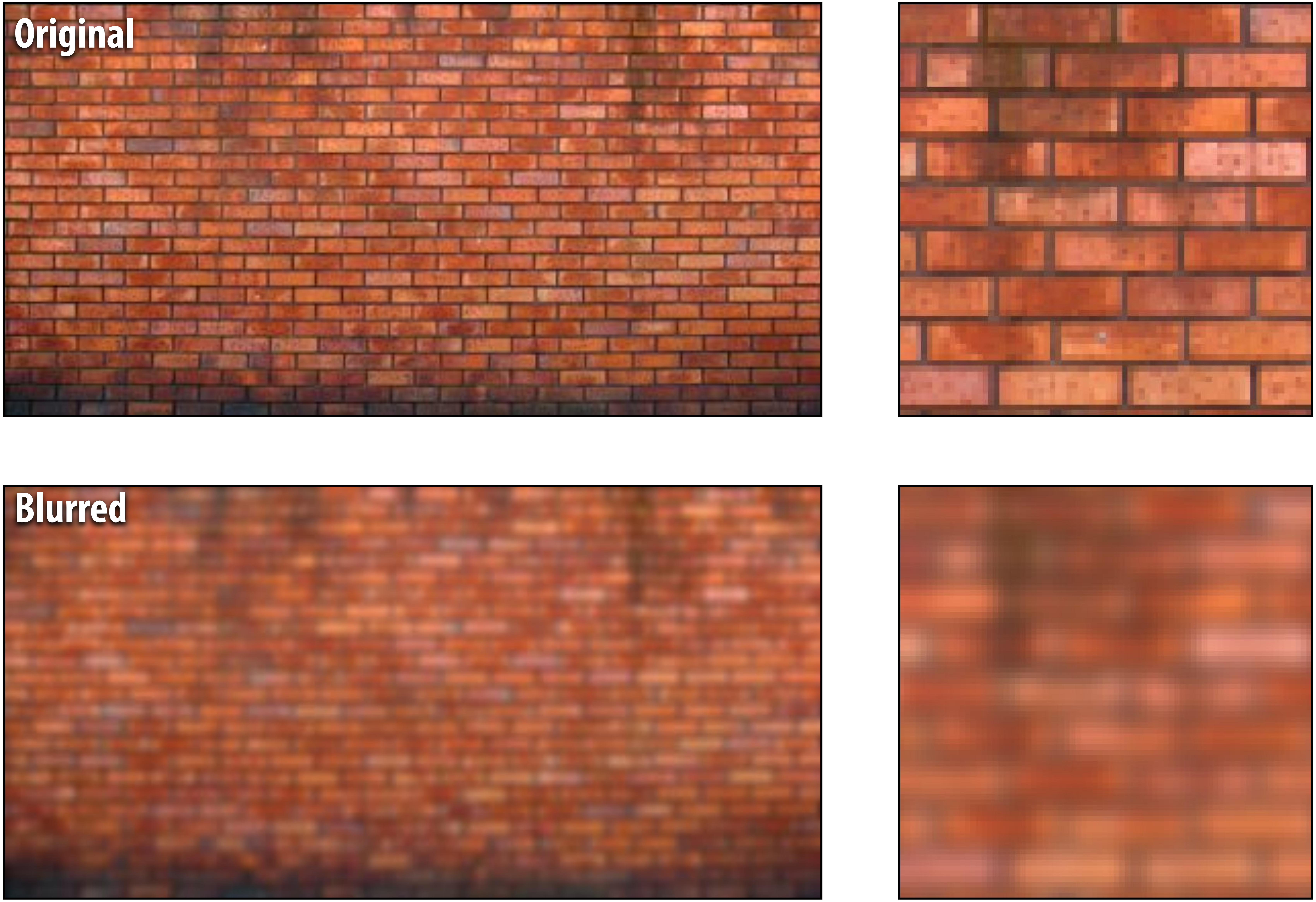
- **Obtain filter coefficients by sampling 2D Gaussian function**

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

- **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

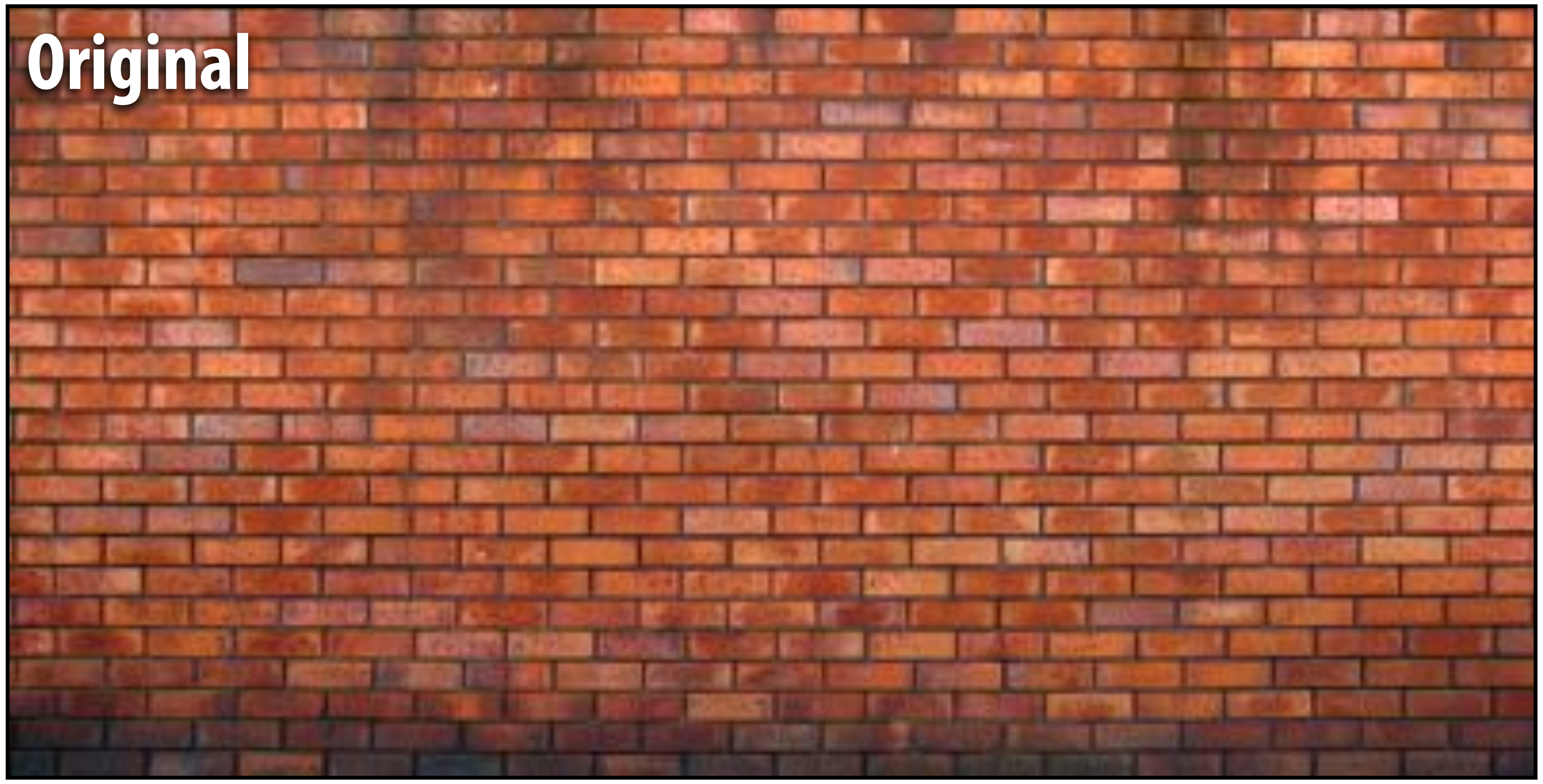


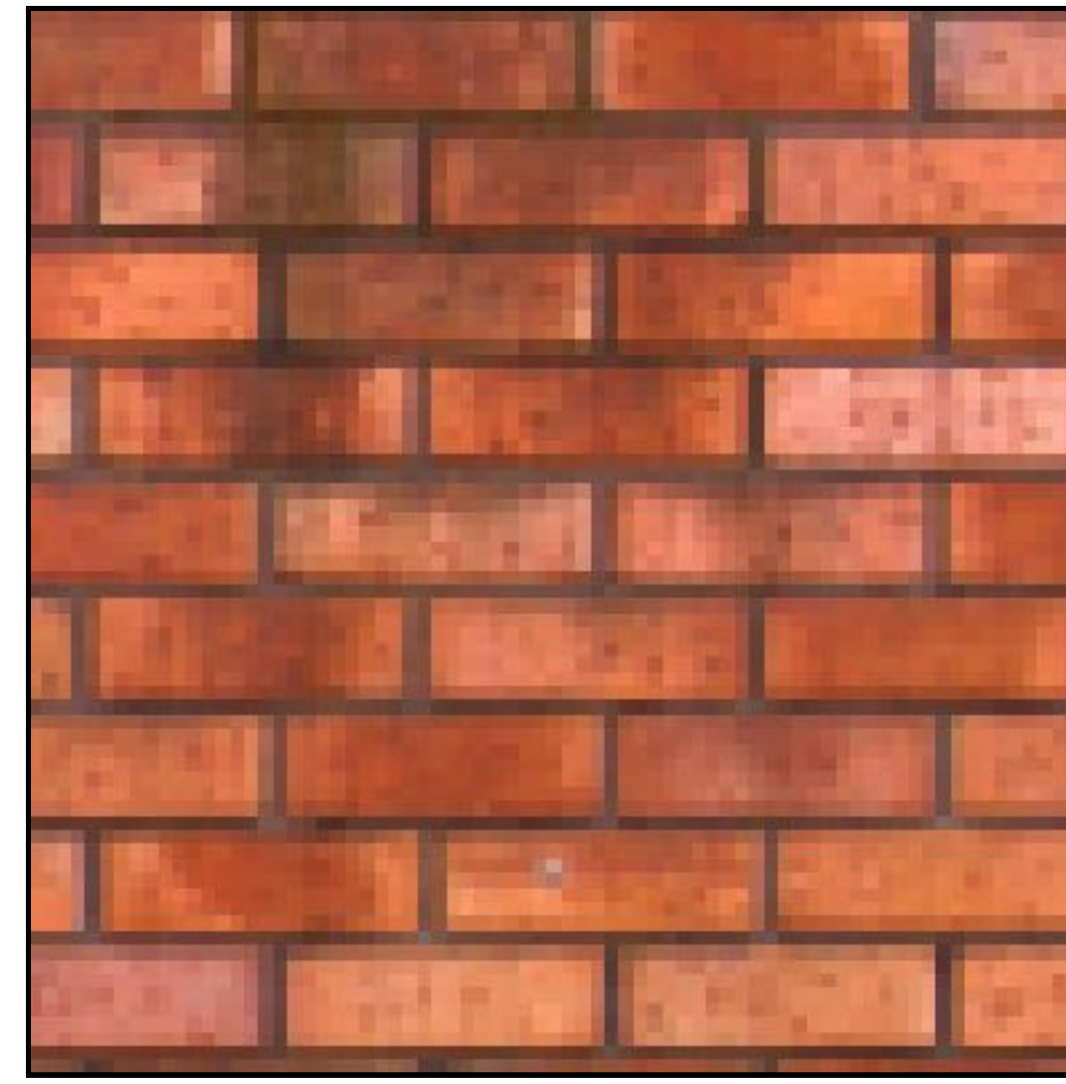
What does convolution with this filter do?

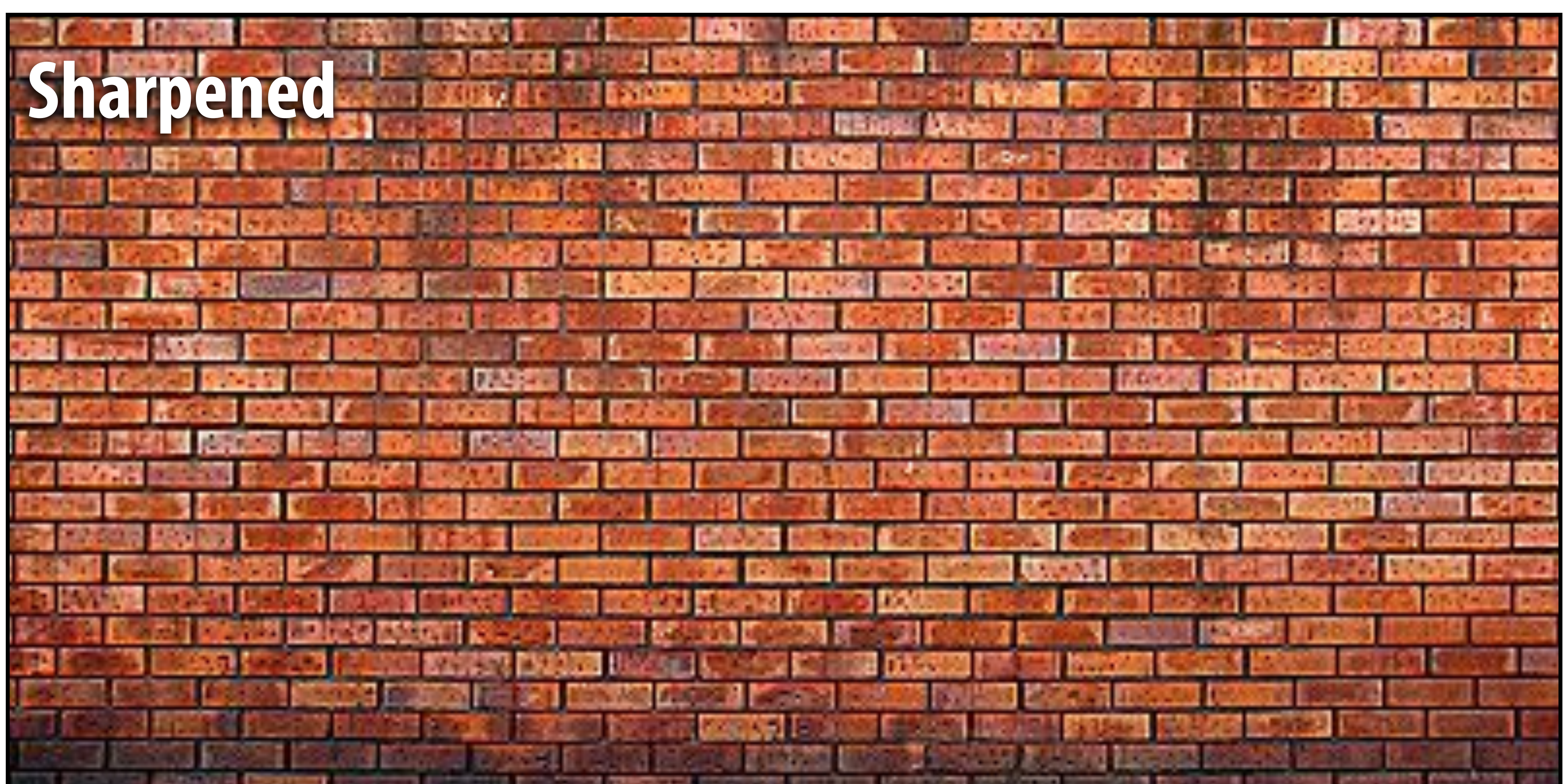
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

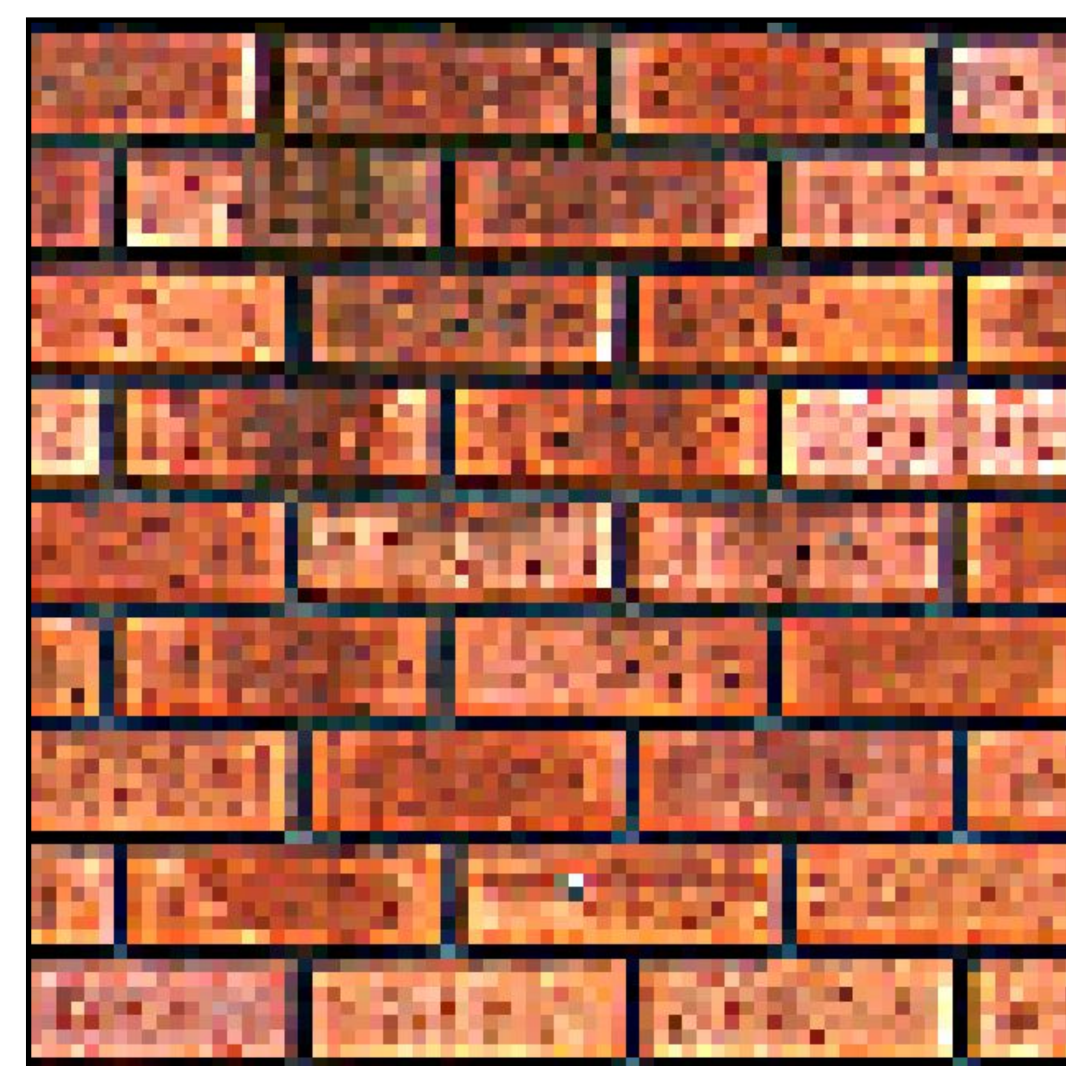
Sharpens image!

3x3 sharpen filter



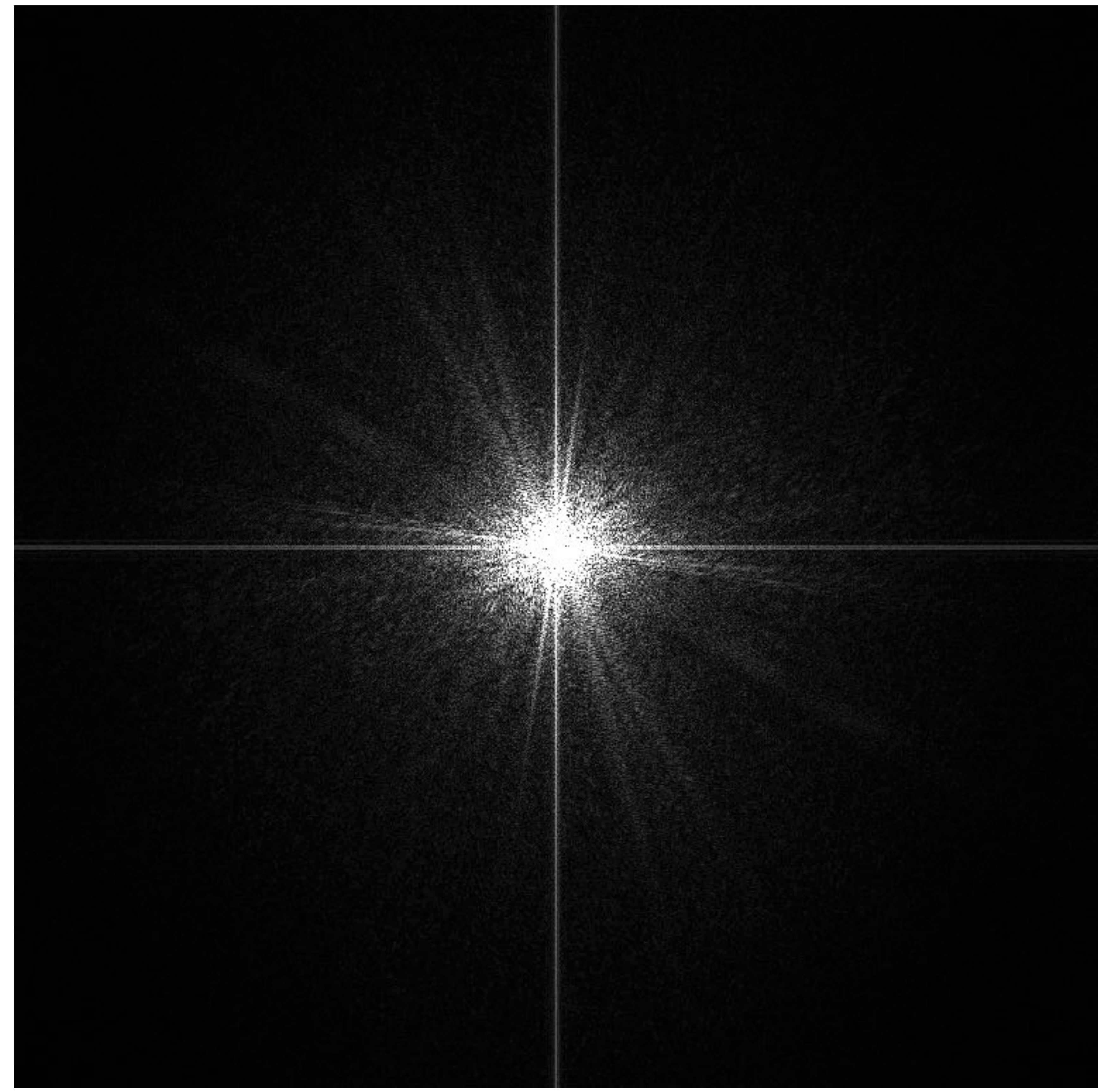






Recall: blurring is removing high frequency content

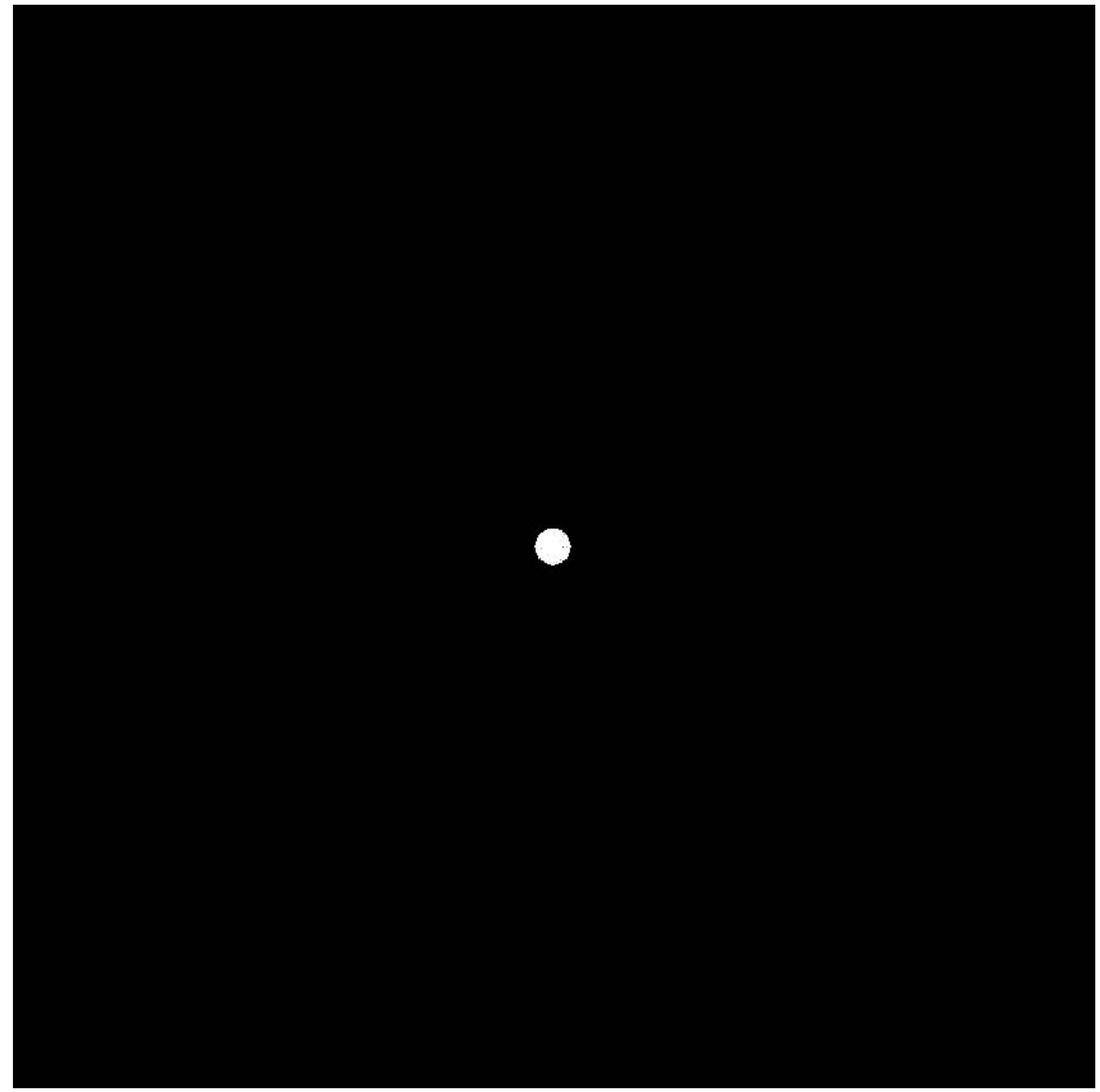
Spatial domain result



Spectrum

Recall: blurring is removing high frequency content

Spatial domain result



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

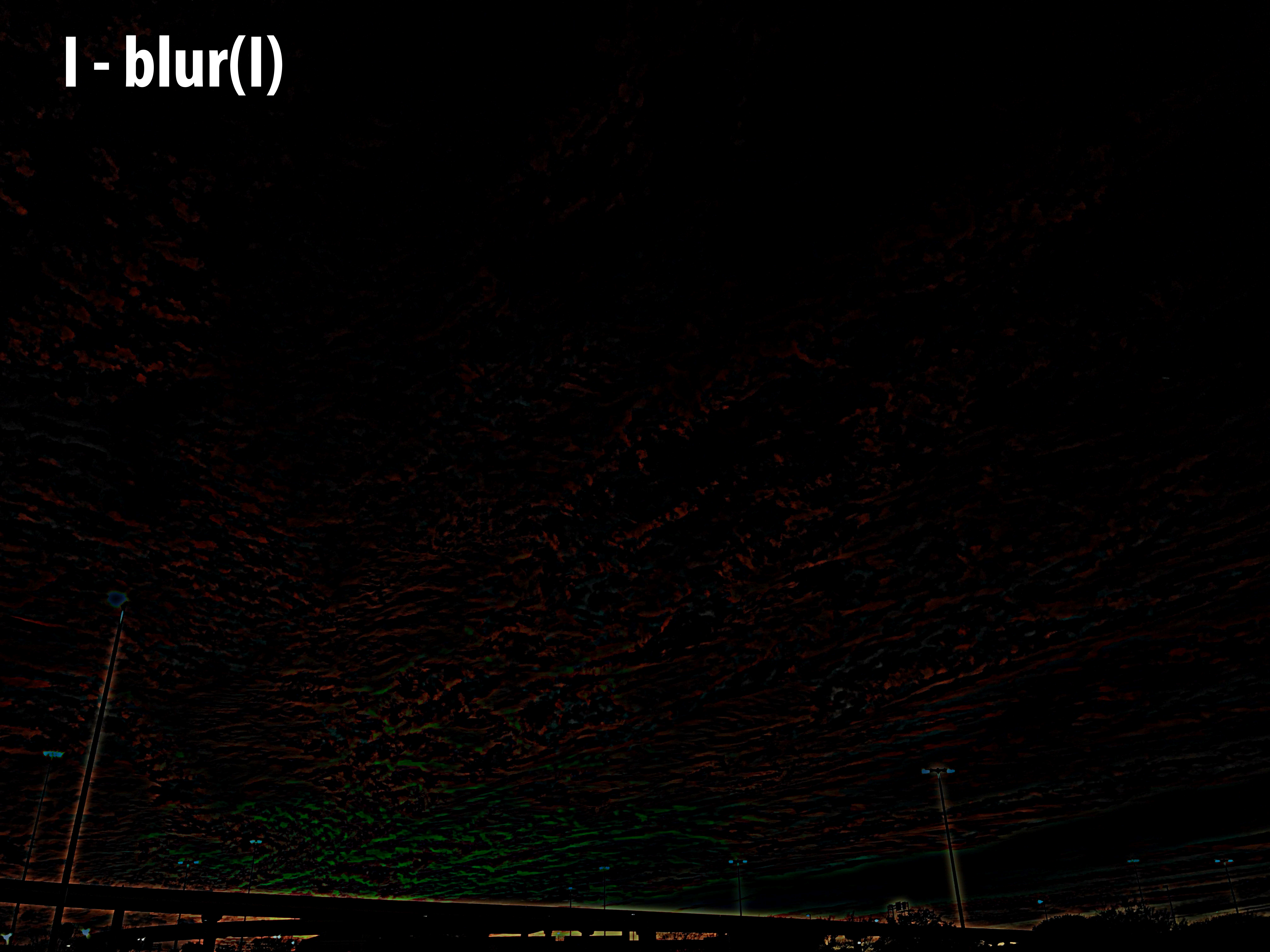
Sharpening is adding high frequencies

- Let I be the original image
- High frequencies in image $I = I$ - blur(I)
- Sharpened image $= I + \underbrace{(I\text{-blur}(I))}_{\text{“Add high frequency content”}}$


Original image (I)
Image credit: Kayvon's parents


Blur(I)

I - blur(I)




I + (I - blur(I))

What does convolution with these filters do?

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

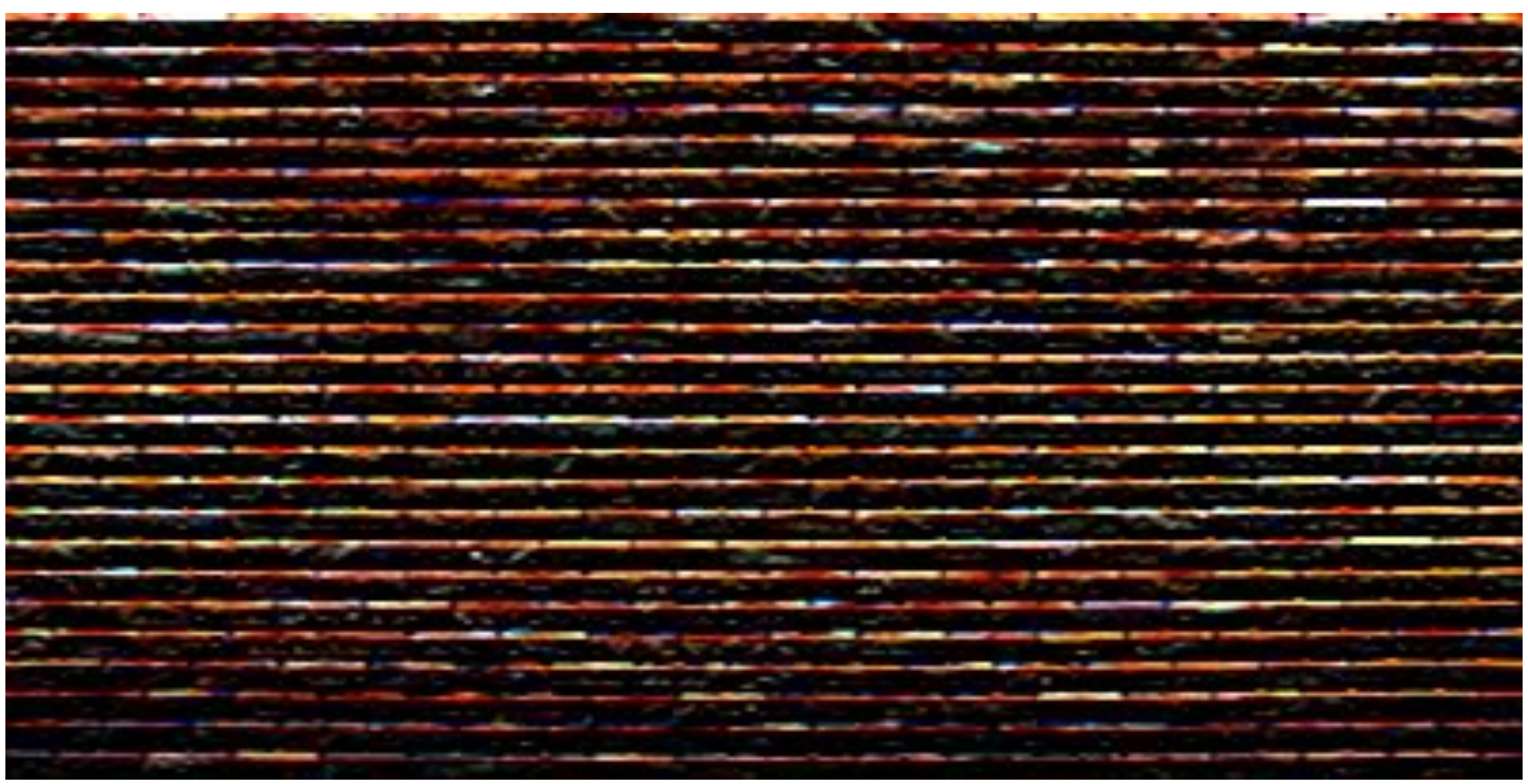
Extracts horizontal gradients

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

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**

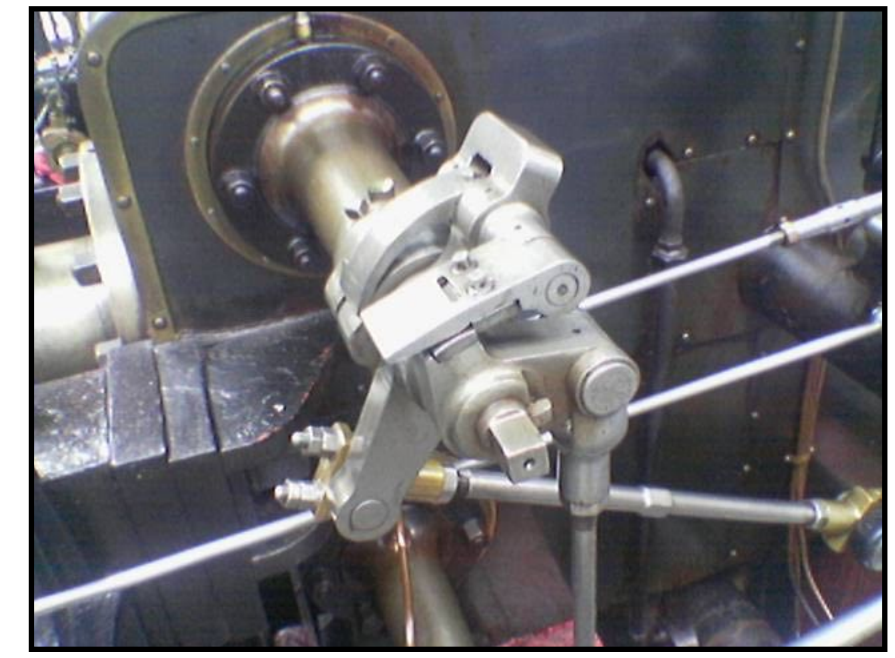
$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I$$

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

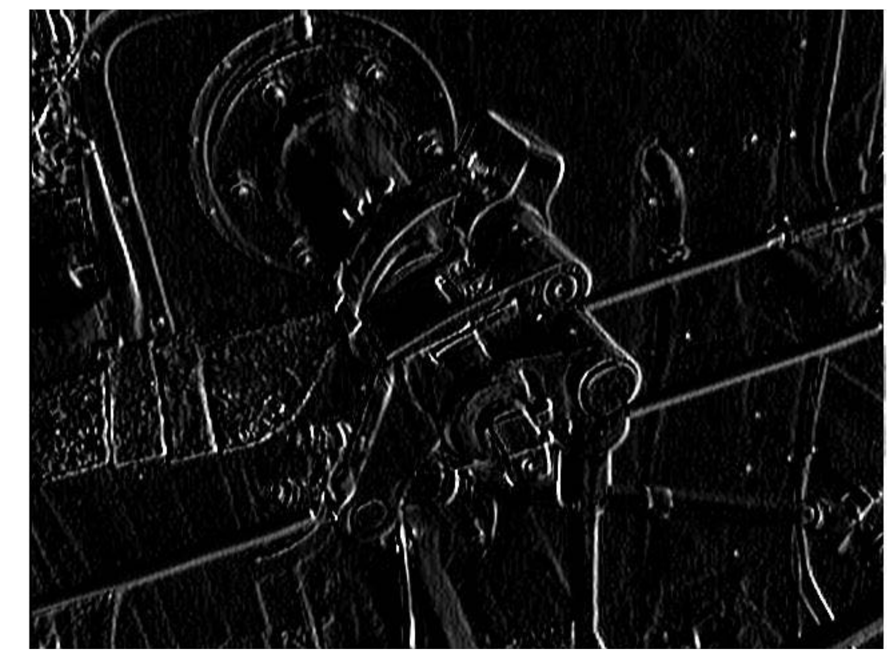
- **Find pixels with large gradients**

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

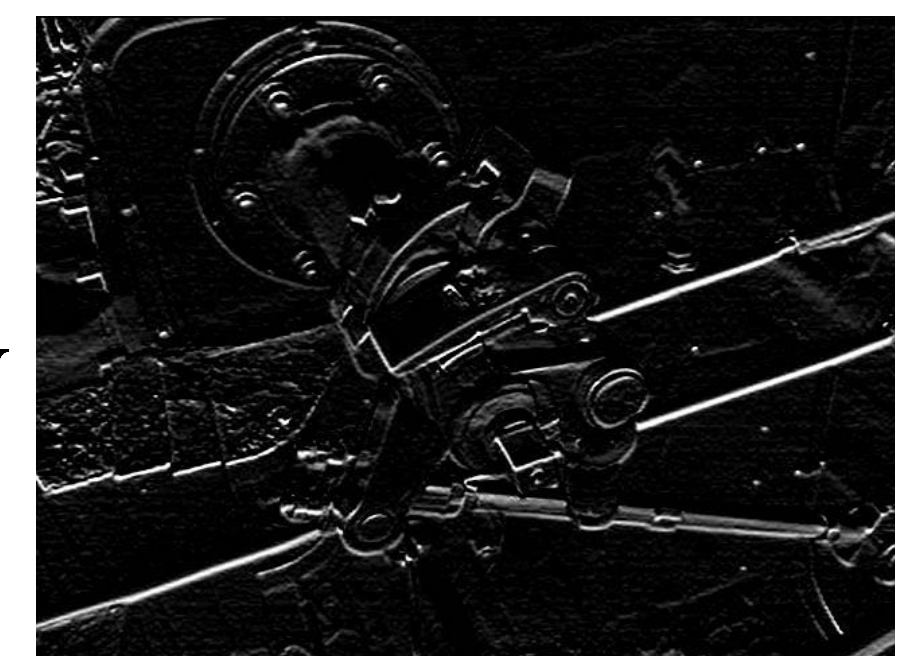
Pixel-wise operation on images



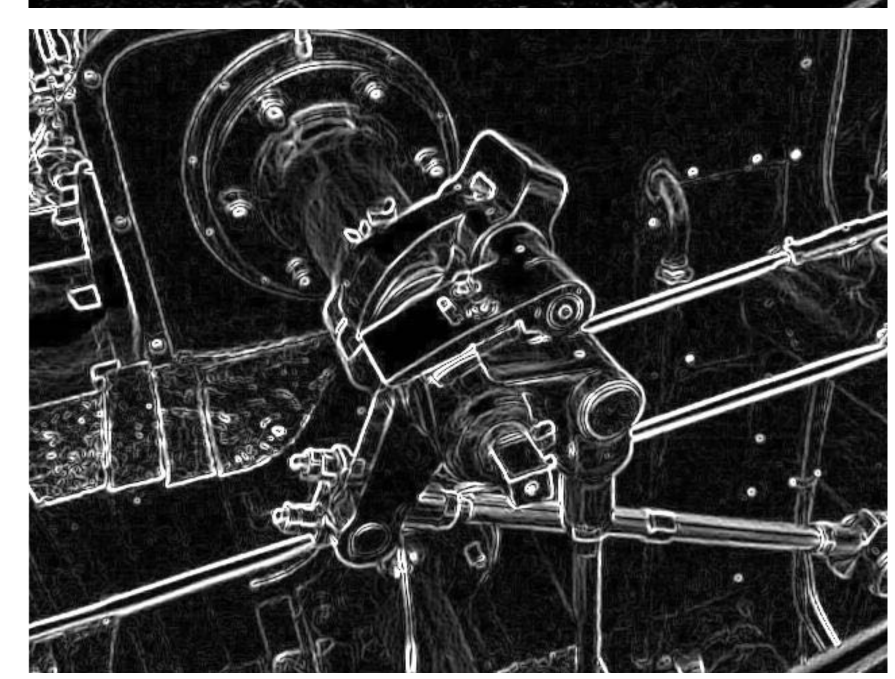
G_x



G_y



G



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**

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

  - **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

Bilateral filter

Original

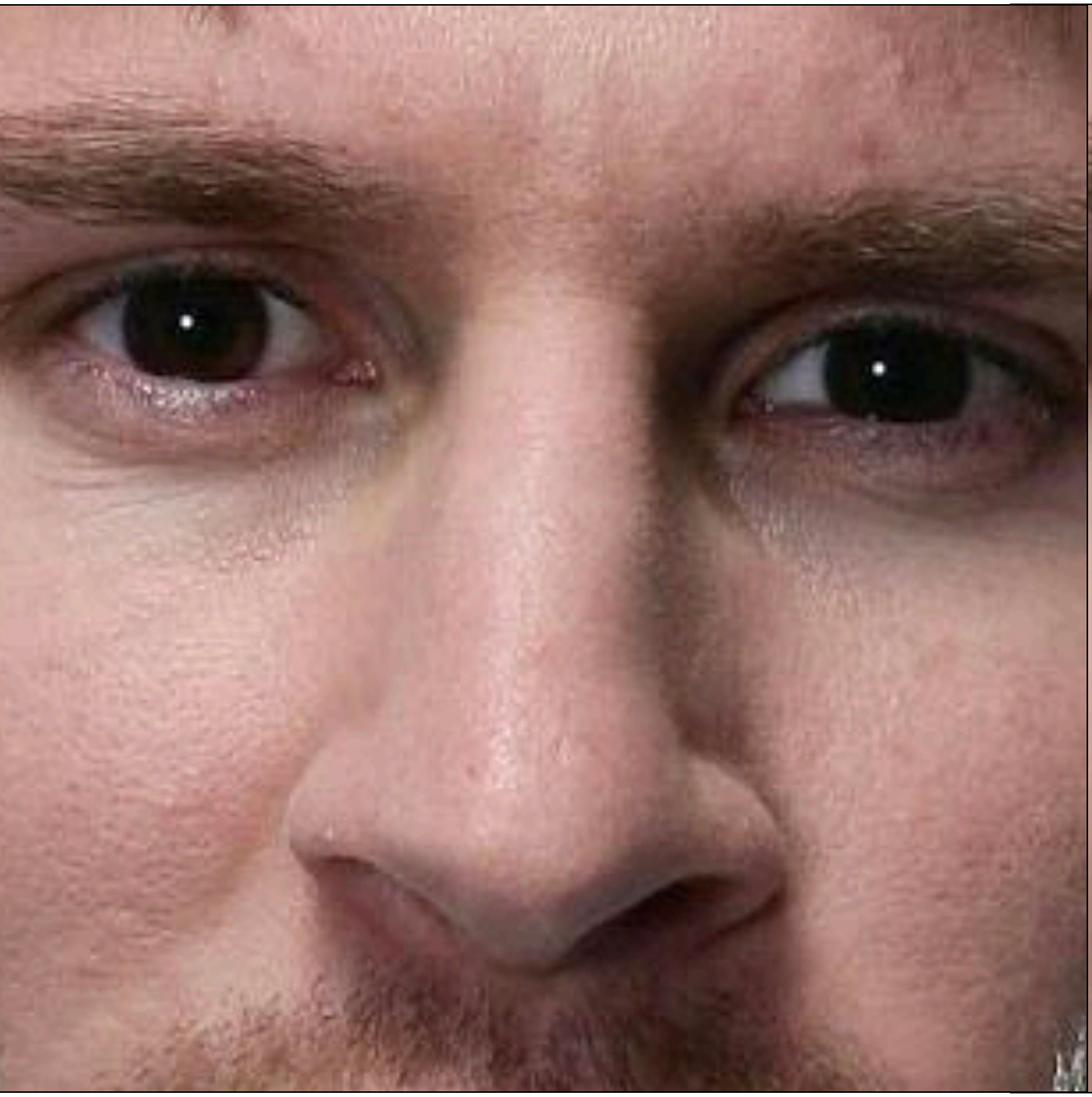
After bilateral filter

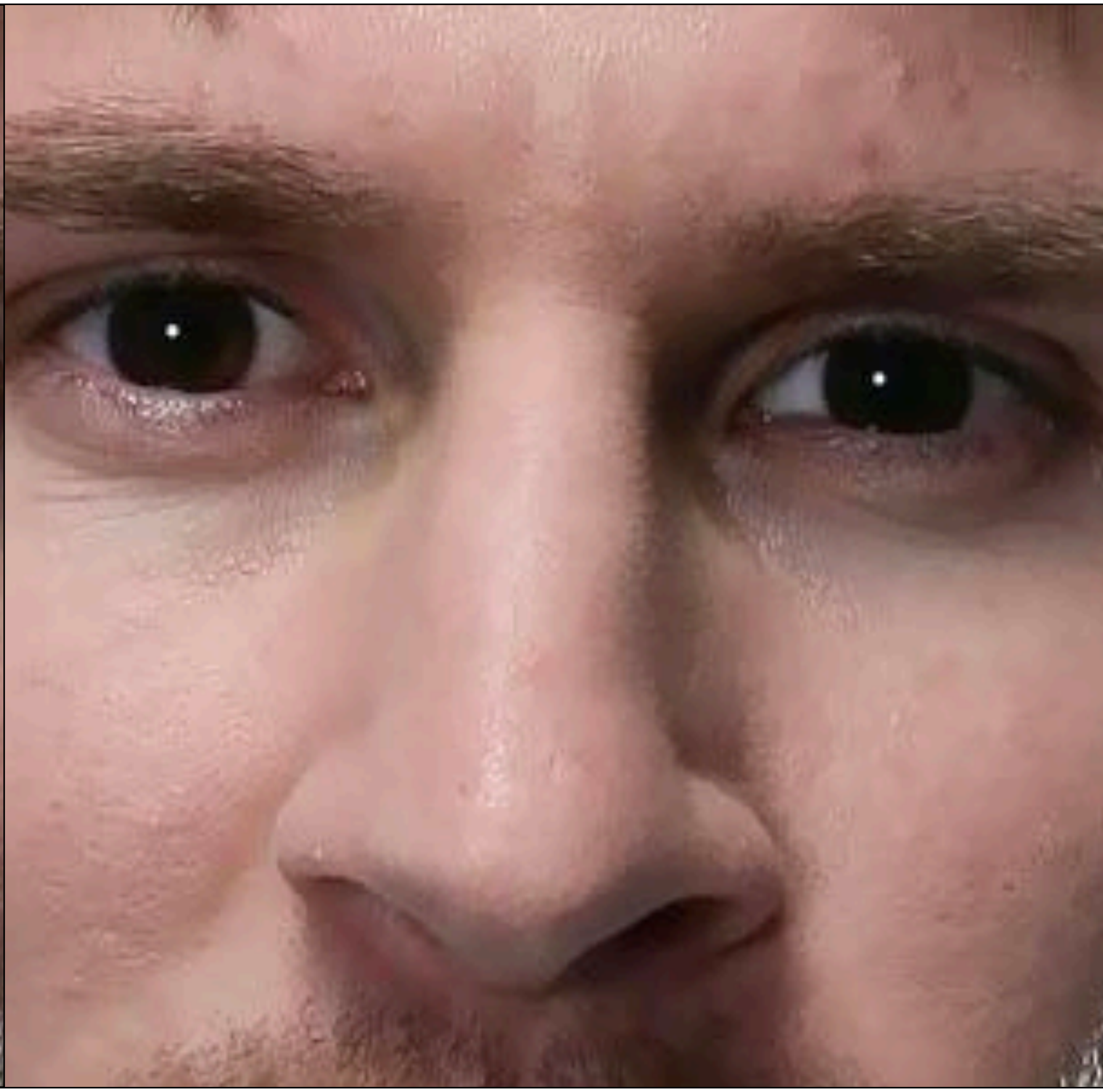
Example use of bilateral filter: removing noise while preserving image edges

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:** $f(x) = 0 \text{ if } x > \textit{threshold}, 1 \textit{ otherwise}$

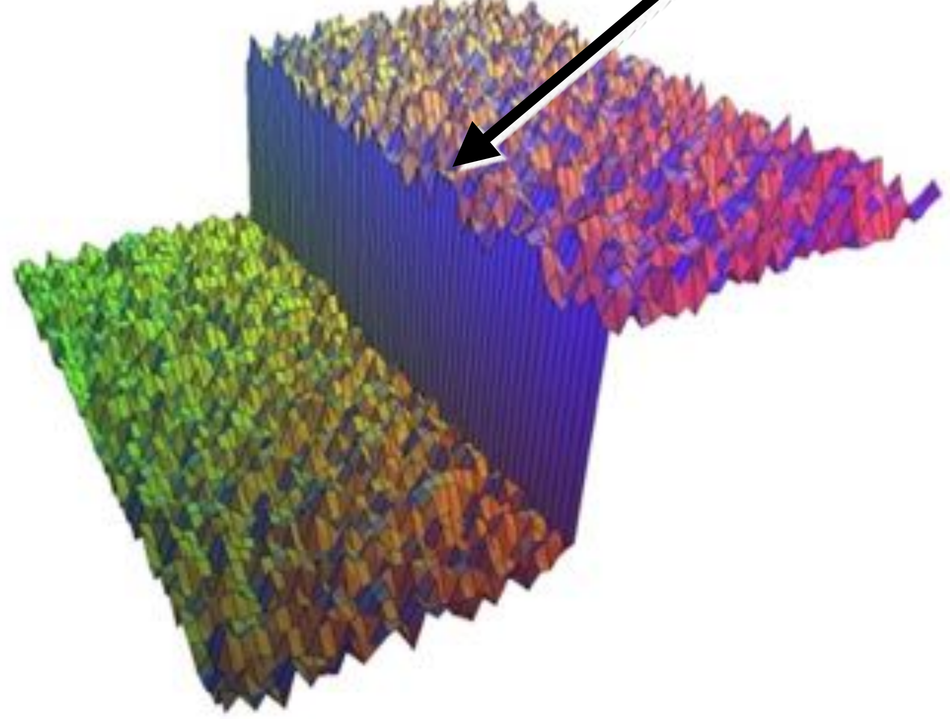
Value of output pixel (x,y) is the weighted sum of all pixels in the support region of a truncated gaussian kernel

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

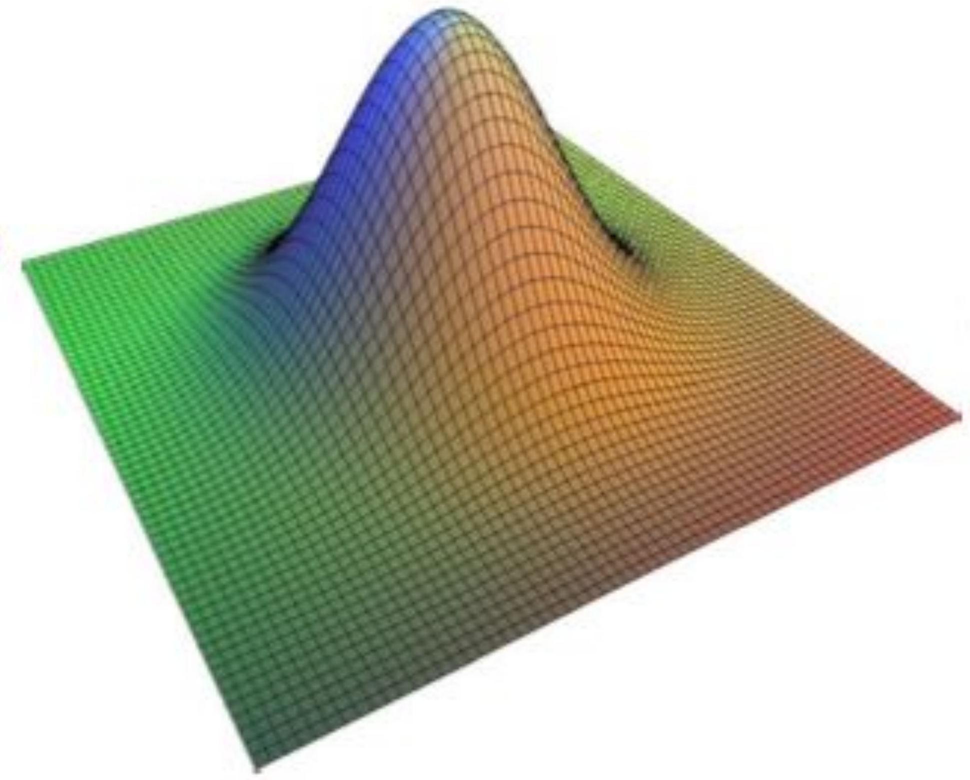
Bilateral filter

Input pixel p

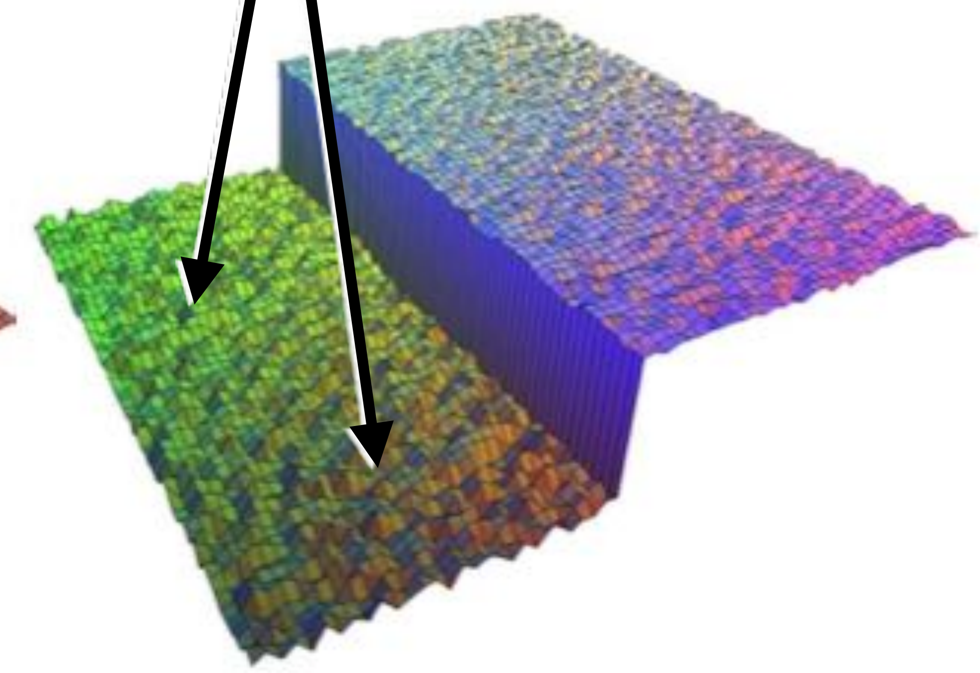
Pixels with significantly different intensity as p contribute little to filtered result (they are “on the “other side of the edge”



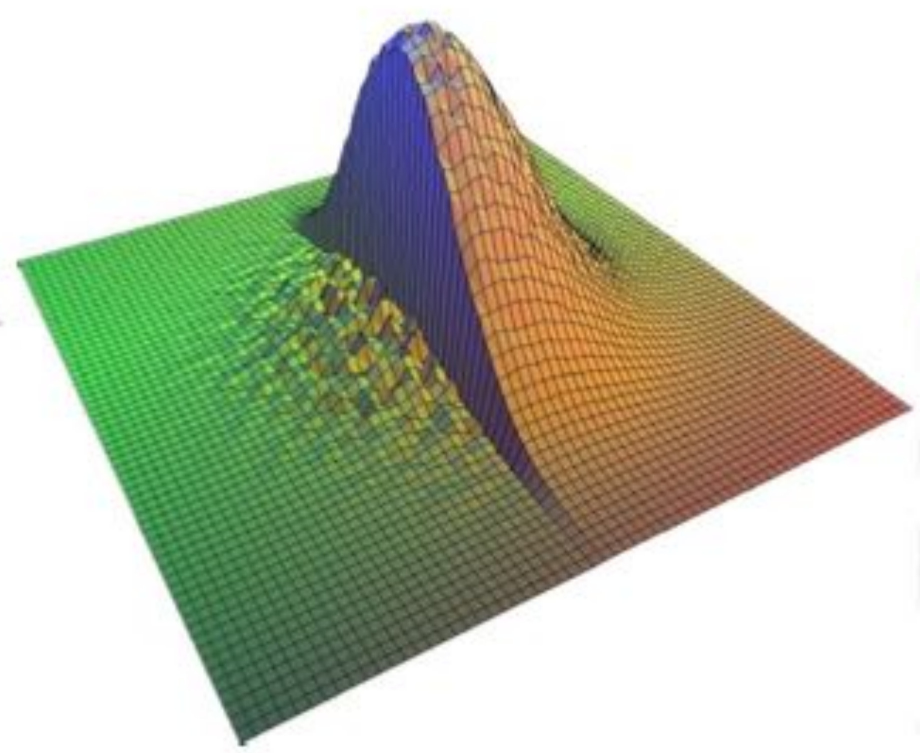
Input image



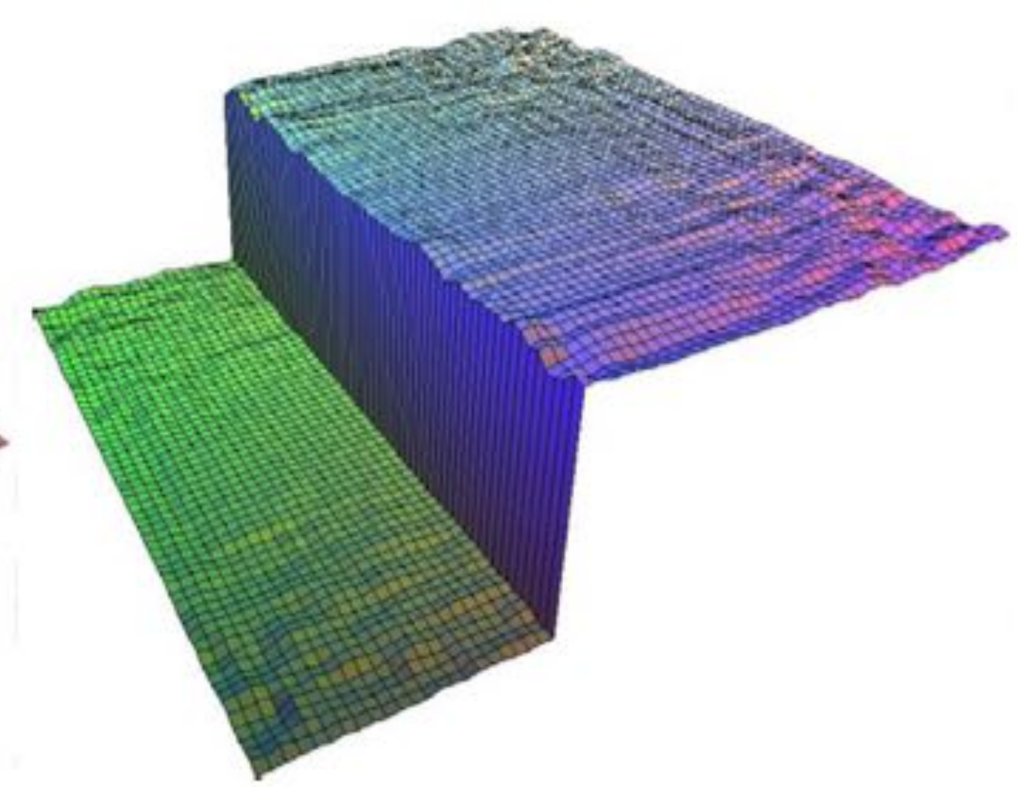
G(): gaussian about input pixel p



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

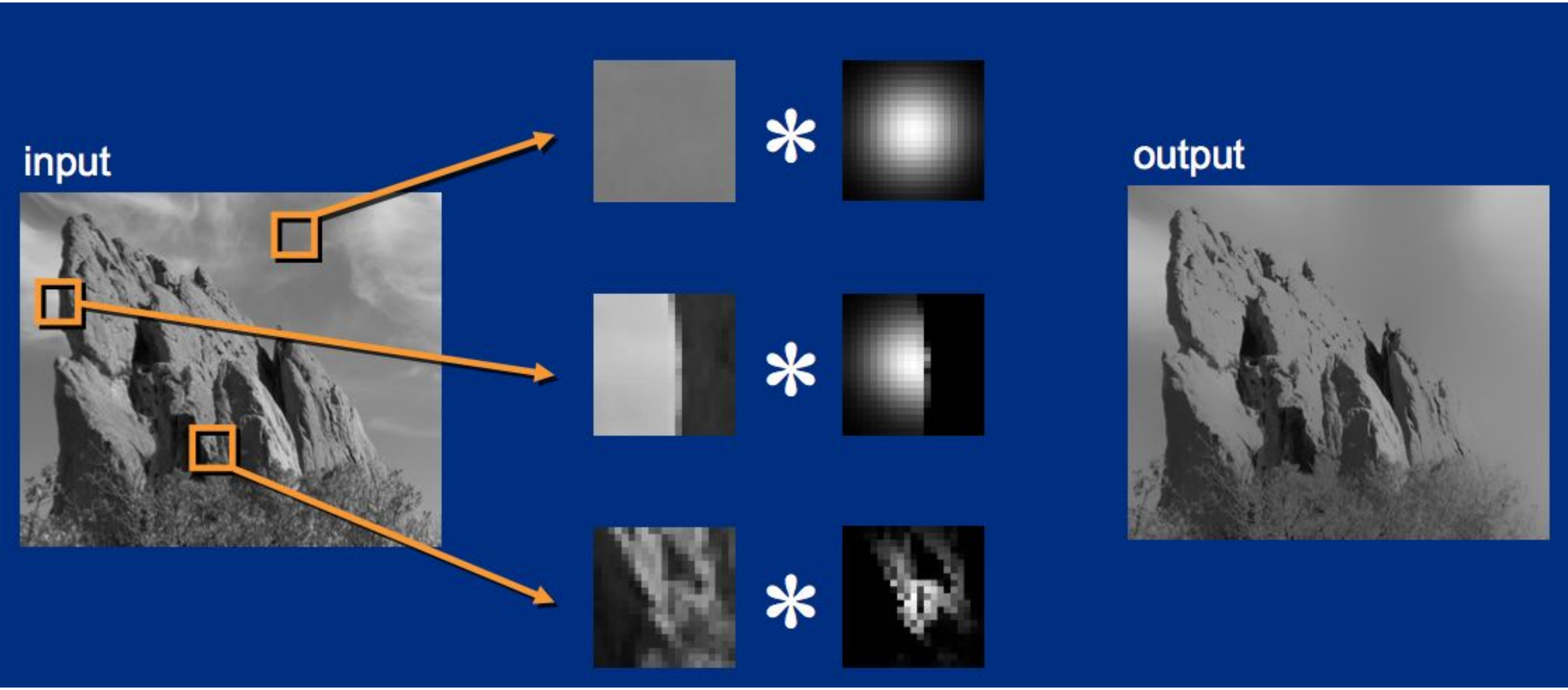


Figure credit: SIGGRAPH 2008 Course: "A Gentle Introduction to Bilateral Filtering and its Applications" Paris et al.

Spatially local vs. frequency local edits

- **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?

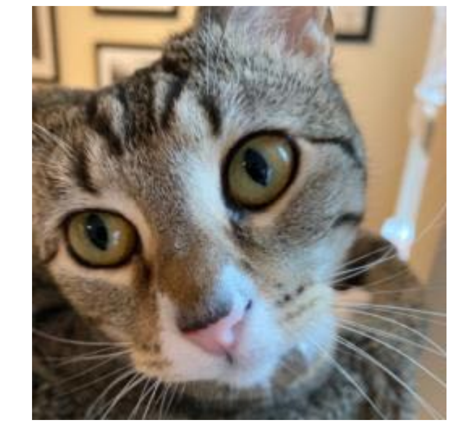
(Adjust certain frequency content of image, in a particular region of the image)

Josephine the Graphics Cat

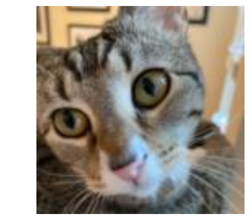
Gaussian pyramid

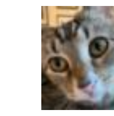
G_0 = original image

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



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





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

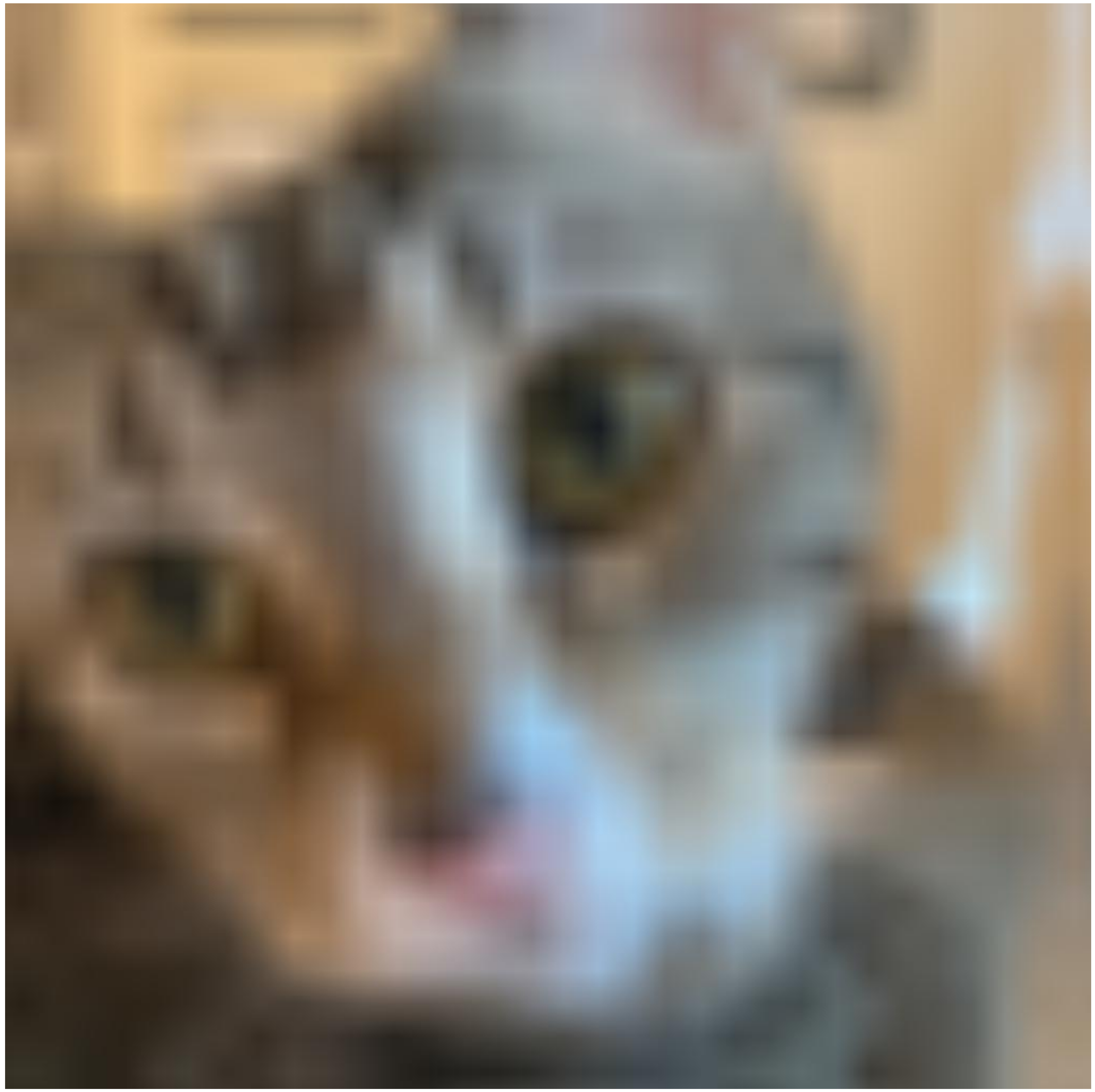
G_3
(upsampled back to full res for visualization)

Gaussian pyramid

G_4

(upsampled back to full res for visualization)

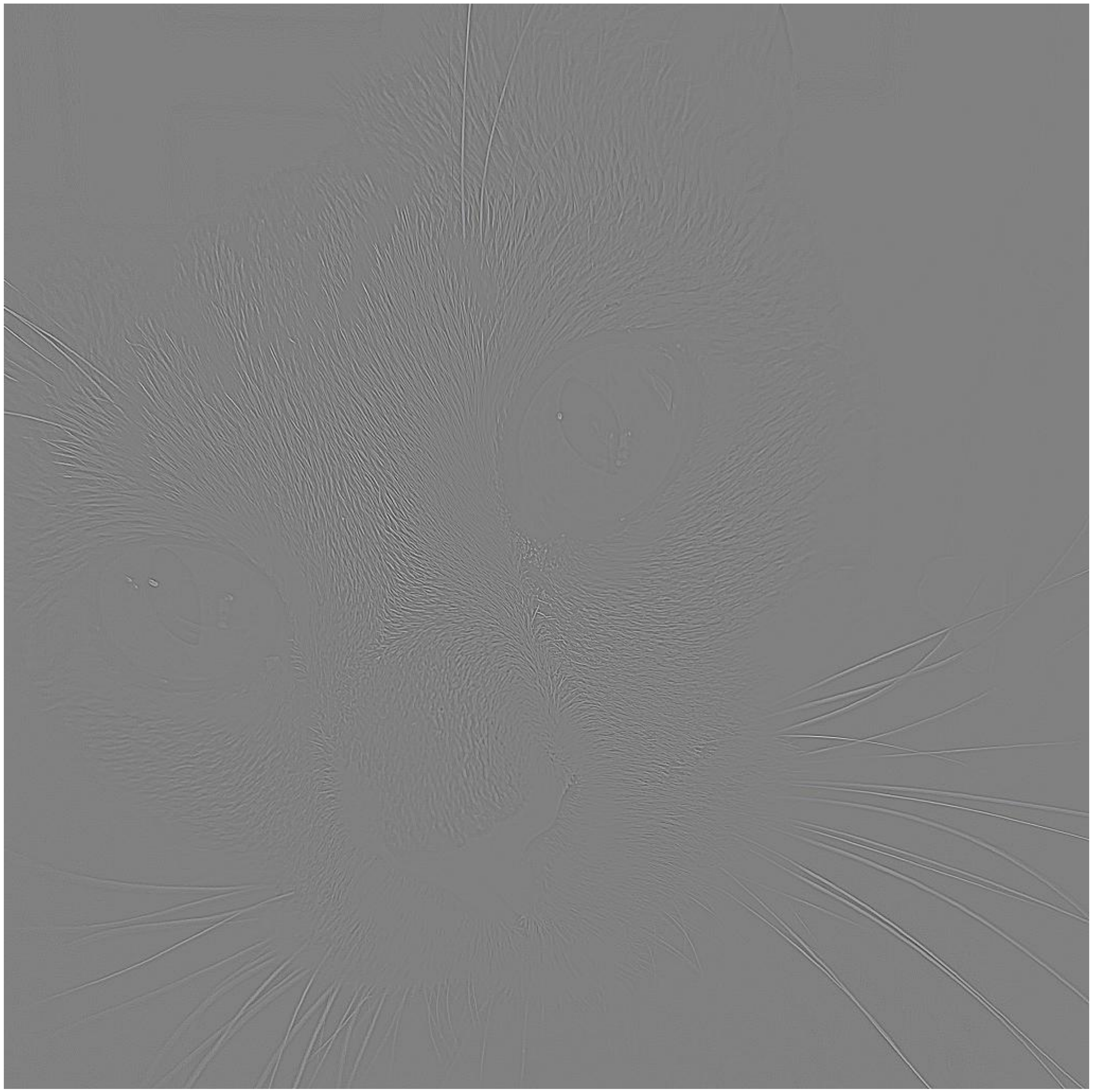
Gaussian pyramid



G_5

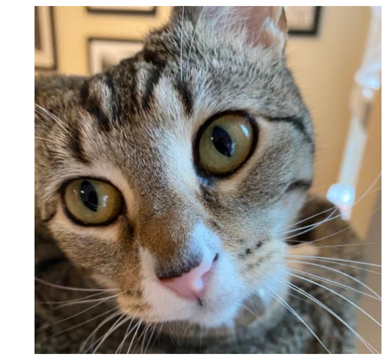
(upsampled back to full res for visualization)

Laplacian pyramid



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

G_0

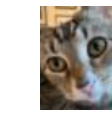


$G_1 = \text{down}(G_0)$

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

[Burt and Adelson 83]

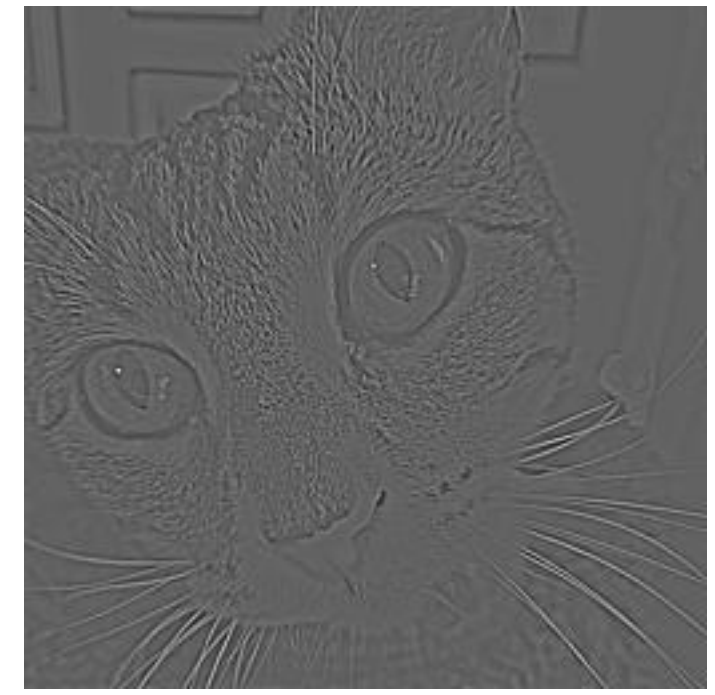
Laplacian pyramid



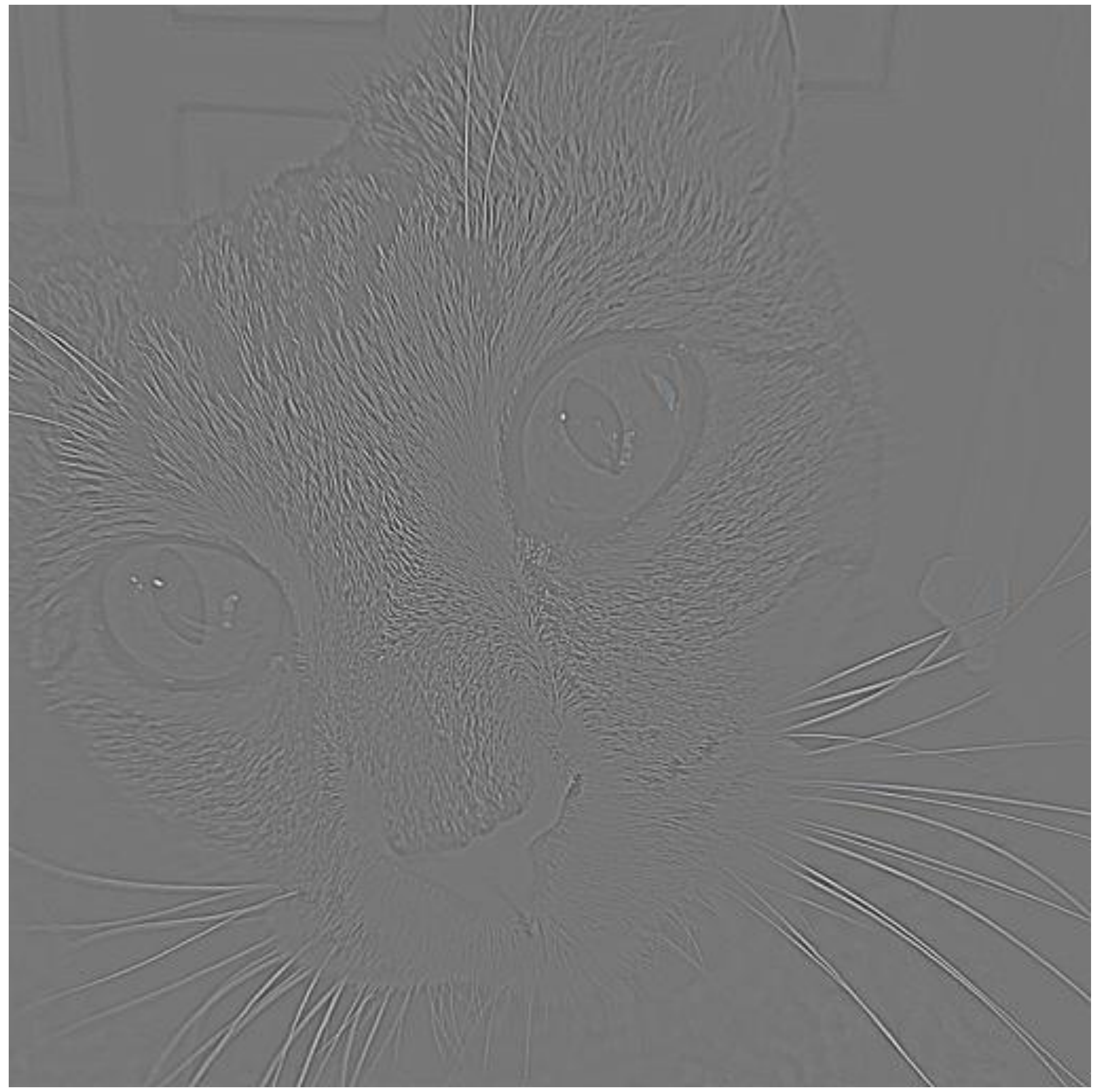
$L_5 = G_5$

$L_4 = G_4$ - up(G_5)

$L_3 = G_3$ - up(G_4)



$L_2 = G_2$ - up(G_3)

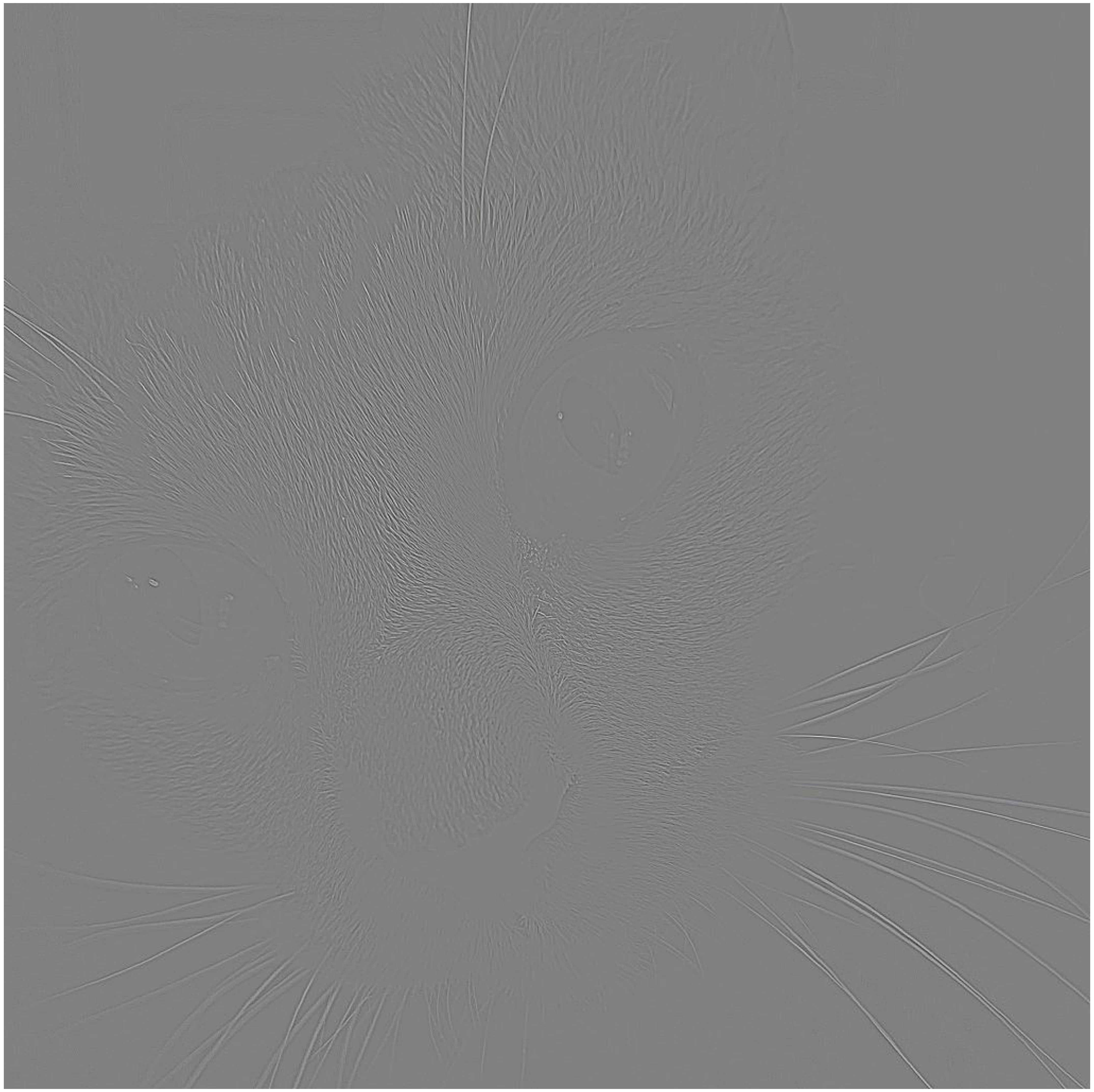


$L_1 = G_1$ - up(G_2)

$L_0 = G_0$ - up(G_1)

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

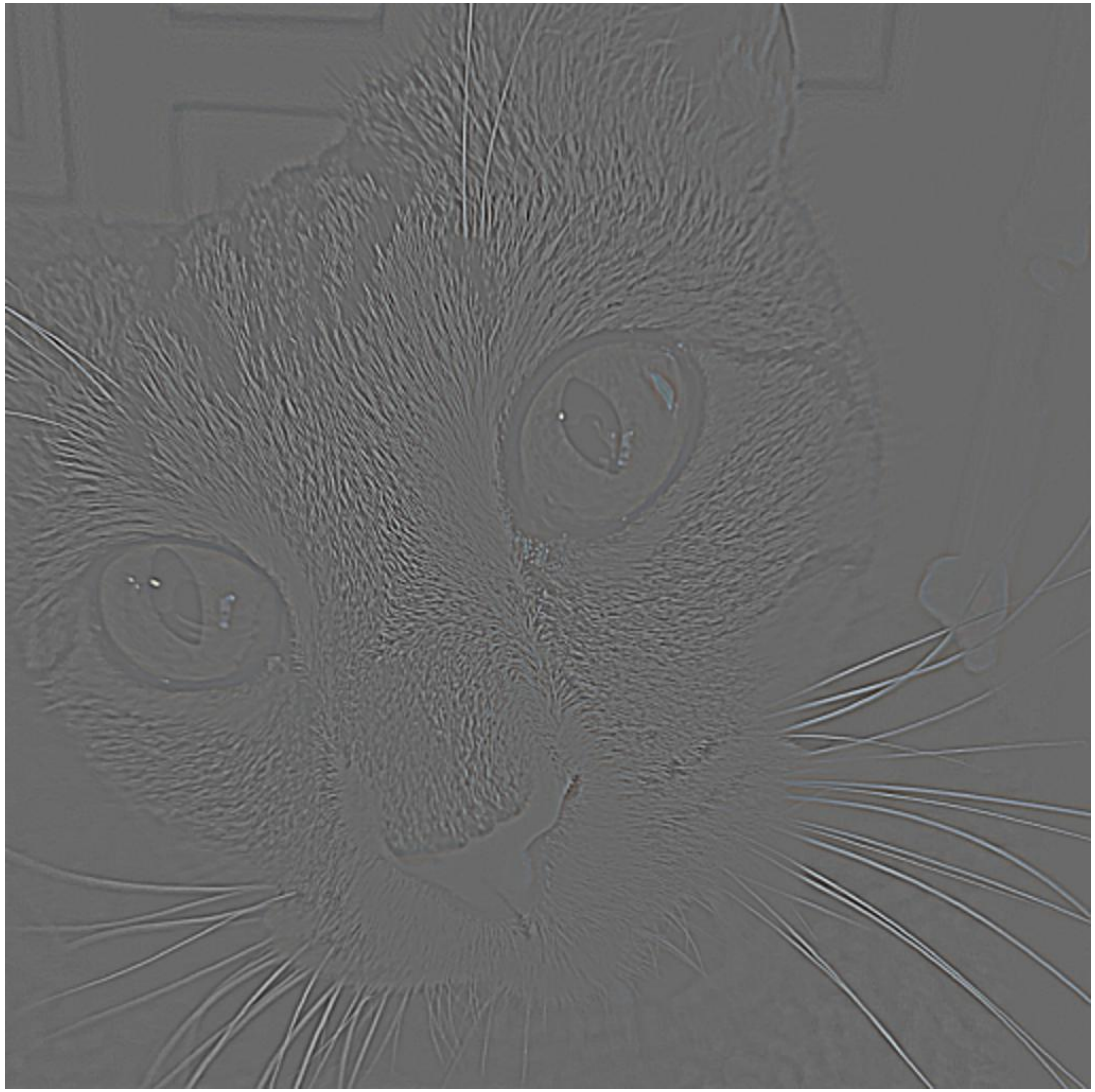
Laplacian pyramid



$L_0 = G_0 - up(G_1)$

(upsampled back to full res for visualization)

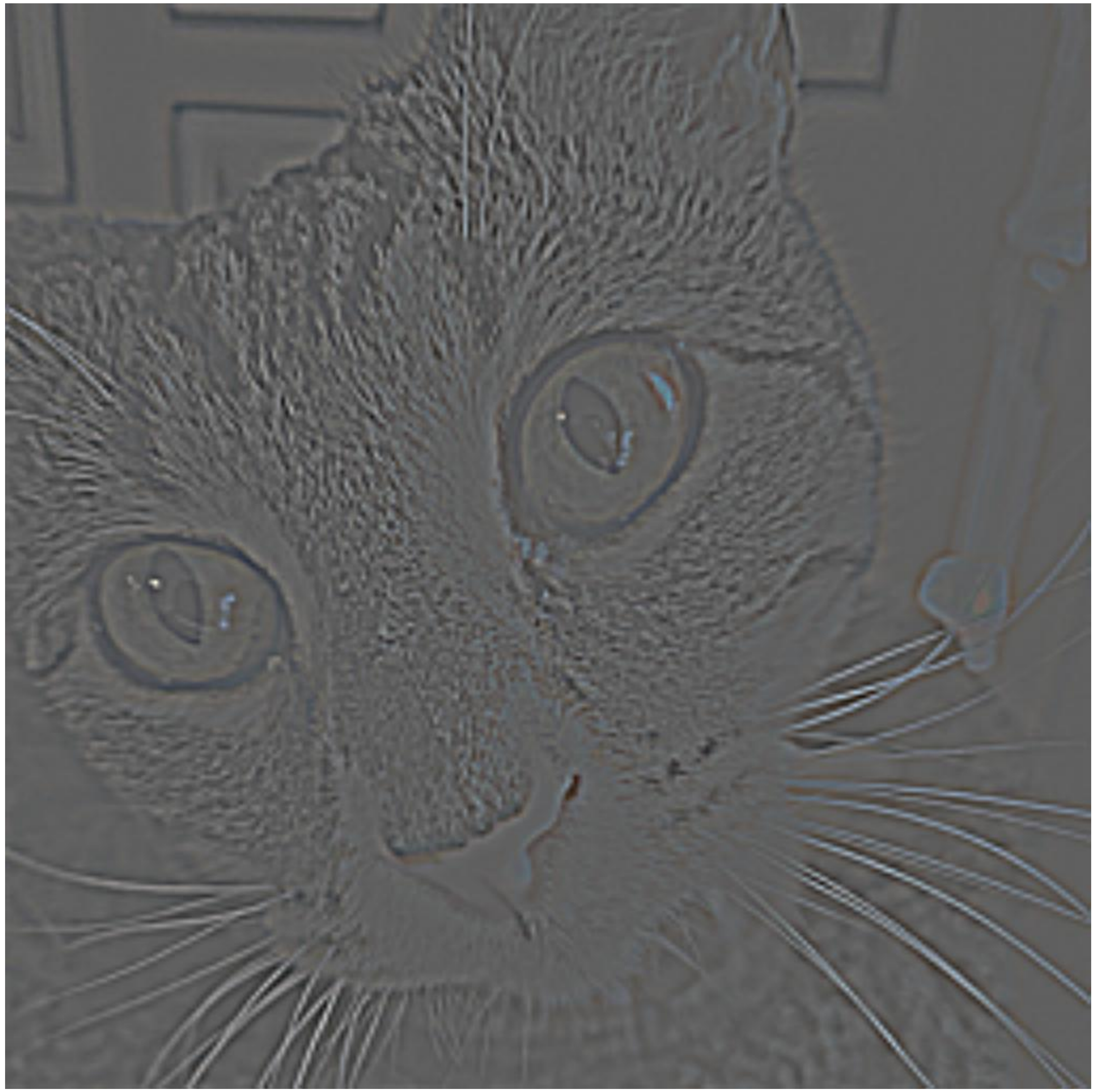
Laplacian pyramid



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

(upsampled back to full res for visualization)

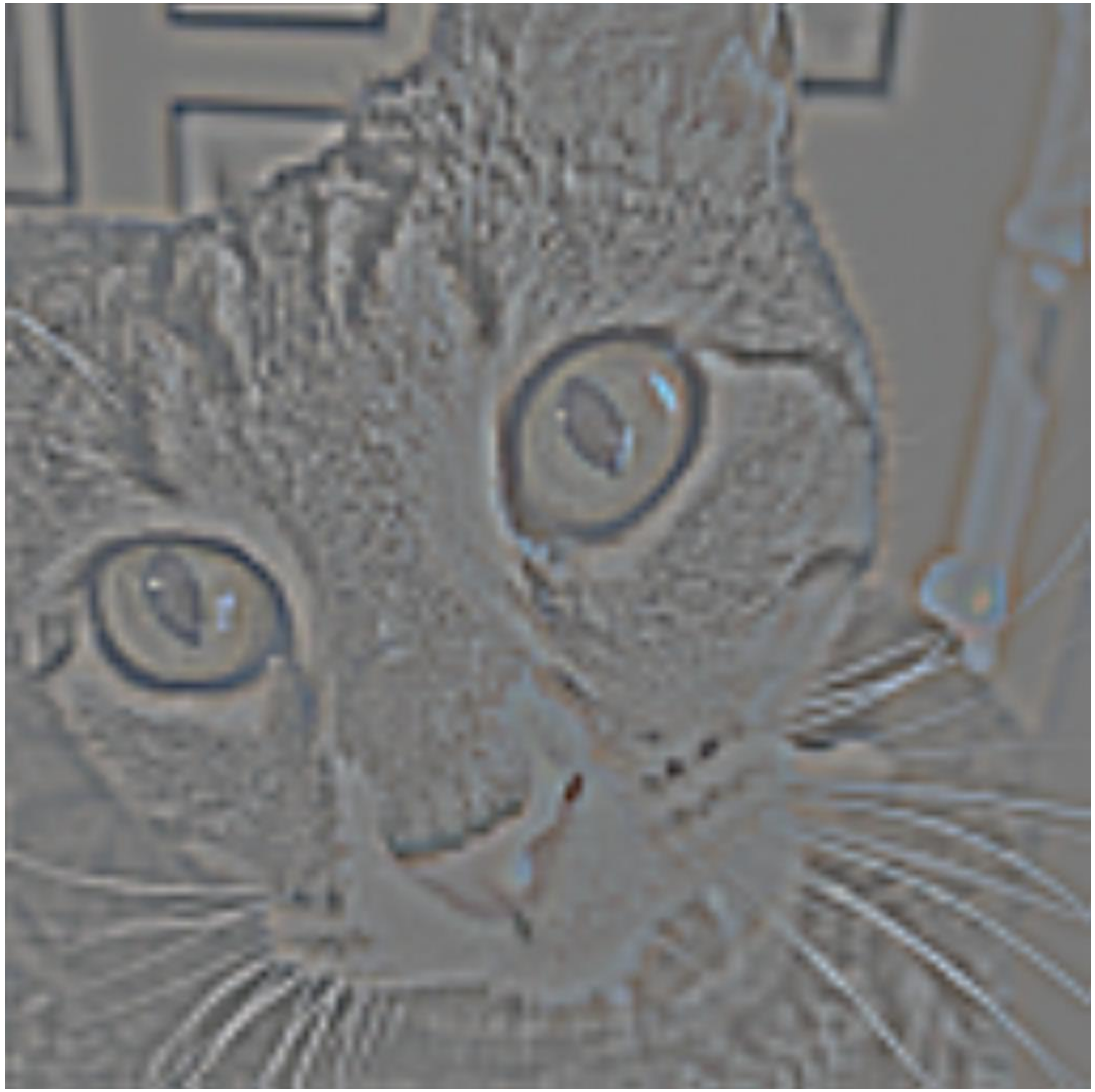
Laplacian pyramid



$L_2 = G_2 - up(G_3)$

(upsampled back to full res for visualization)

Laplacian pyramid



$L_3 = G_3$ - up(G_4)

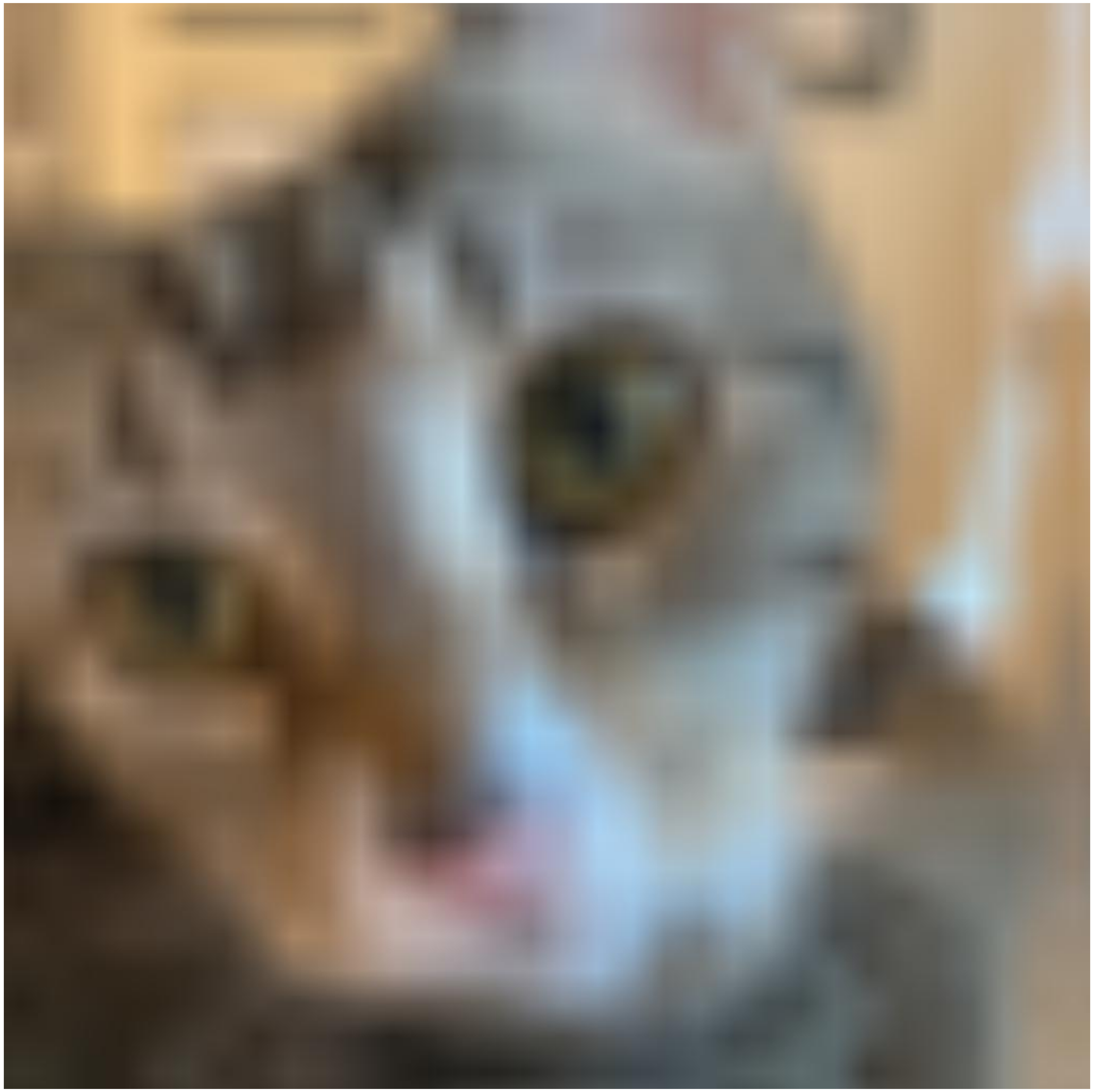
(upsampled back to full res for visualization)

Laplacian pyramid

$L_4 = G_4 - \text{up}(G_5)$

(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.

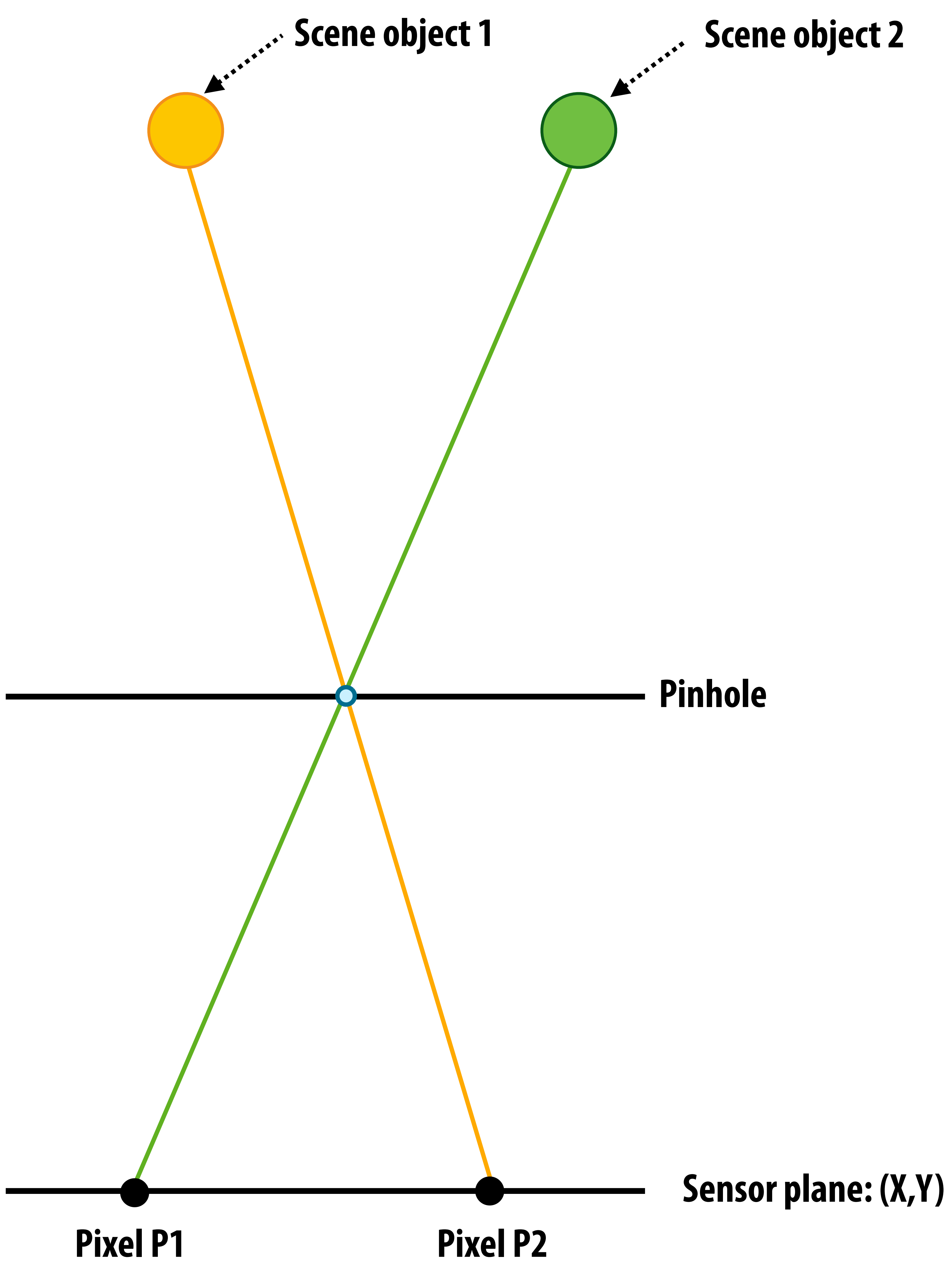
Sensor output ("RAW")

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)

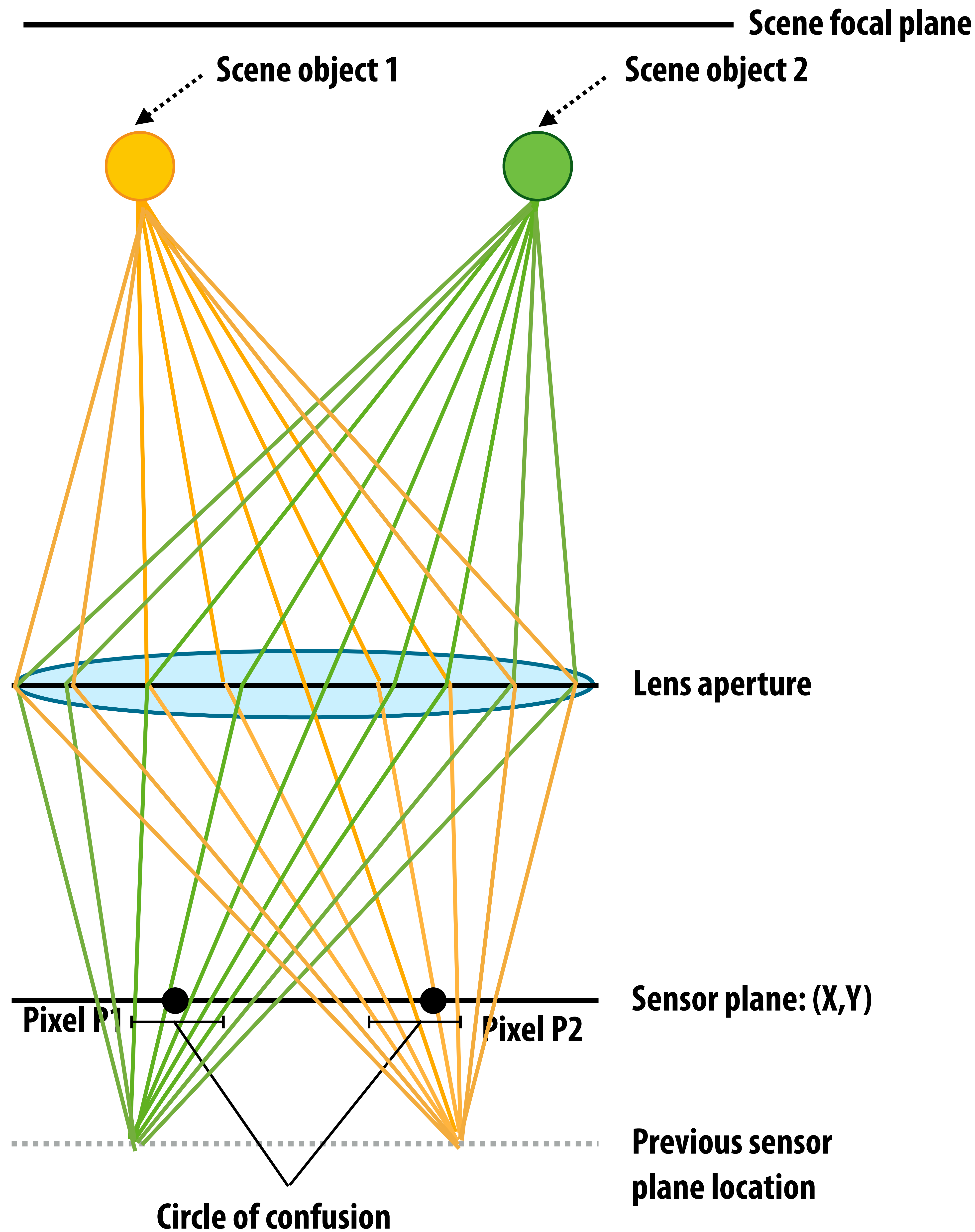


Camera with a lens

Camera with a large (zoom) lens

Review: out of focus camera

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



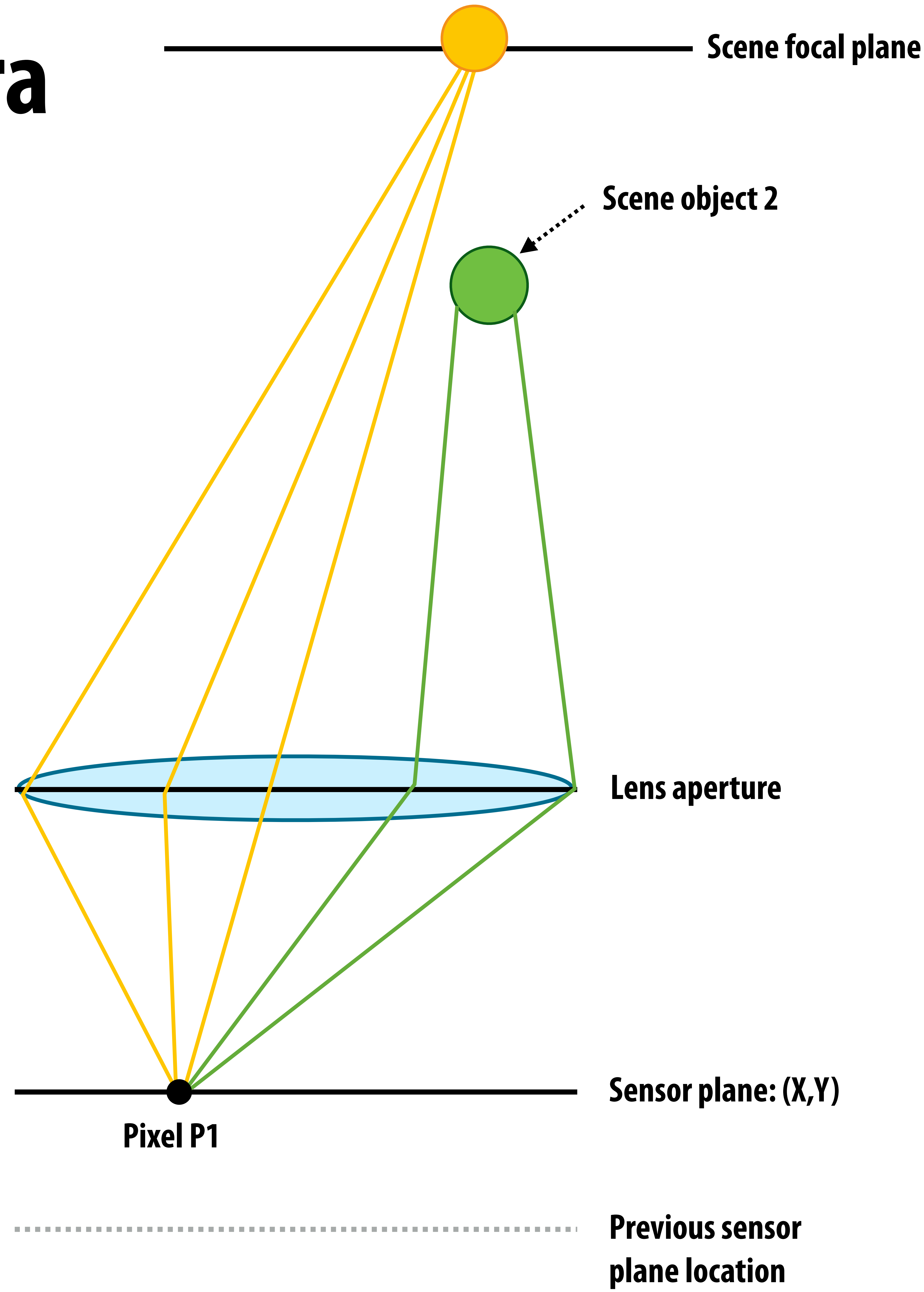
Bokeh

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

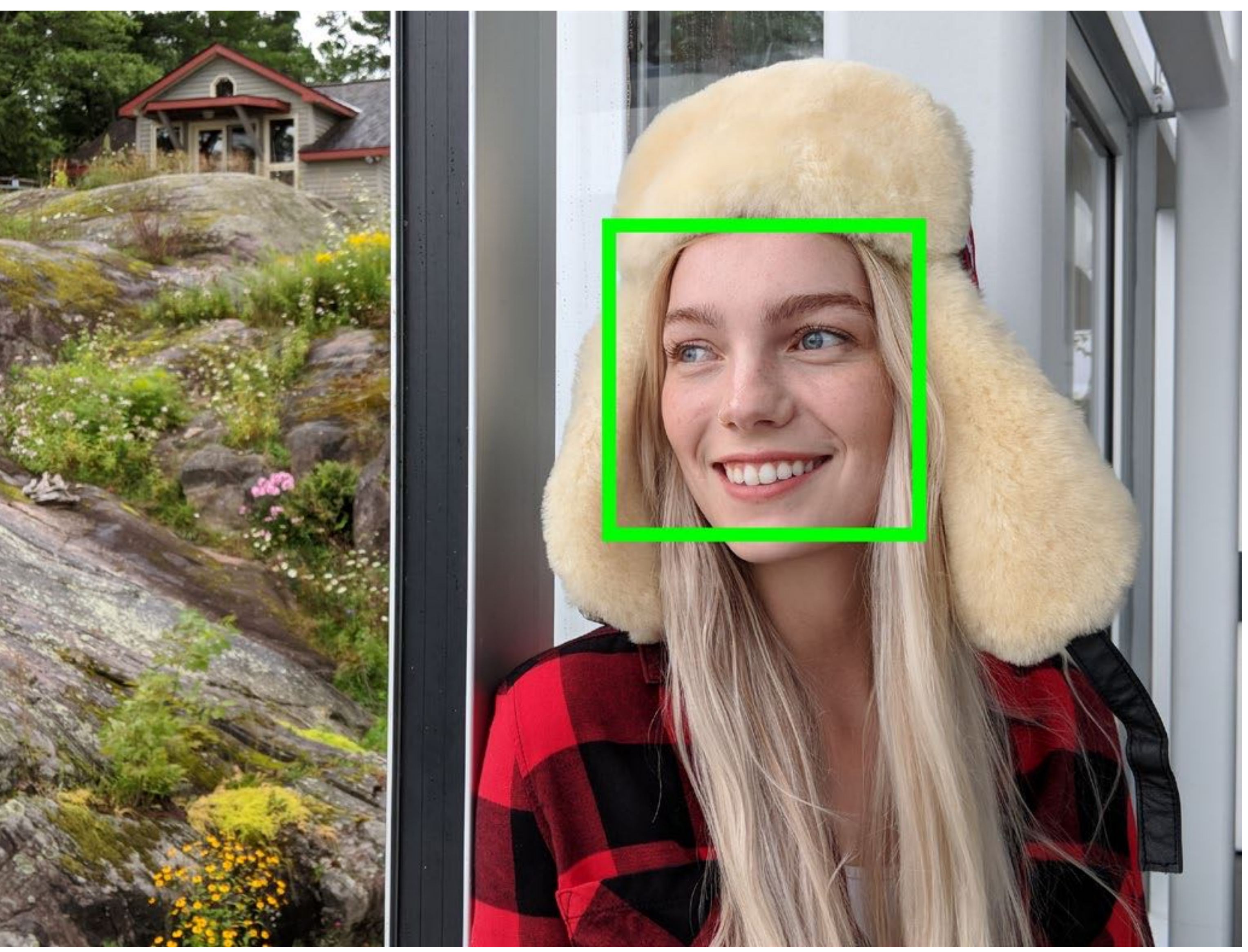


Sharp foreground / blurry background

Cell phone camera lens(es) (small aperture)

“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

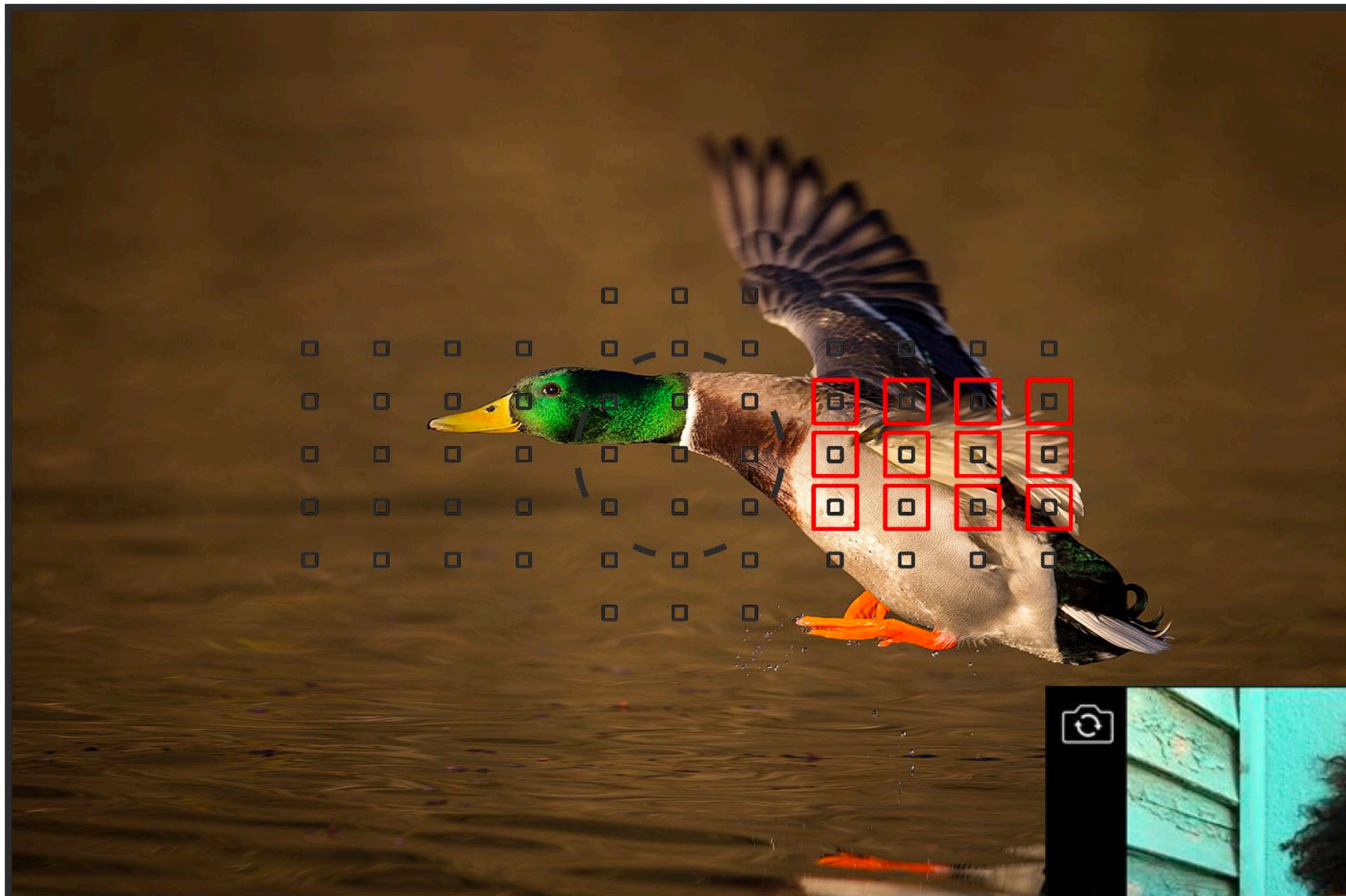
Segmentation

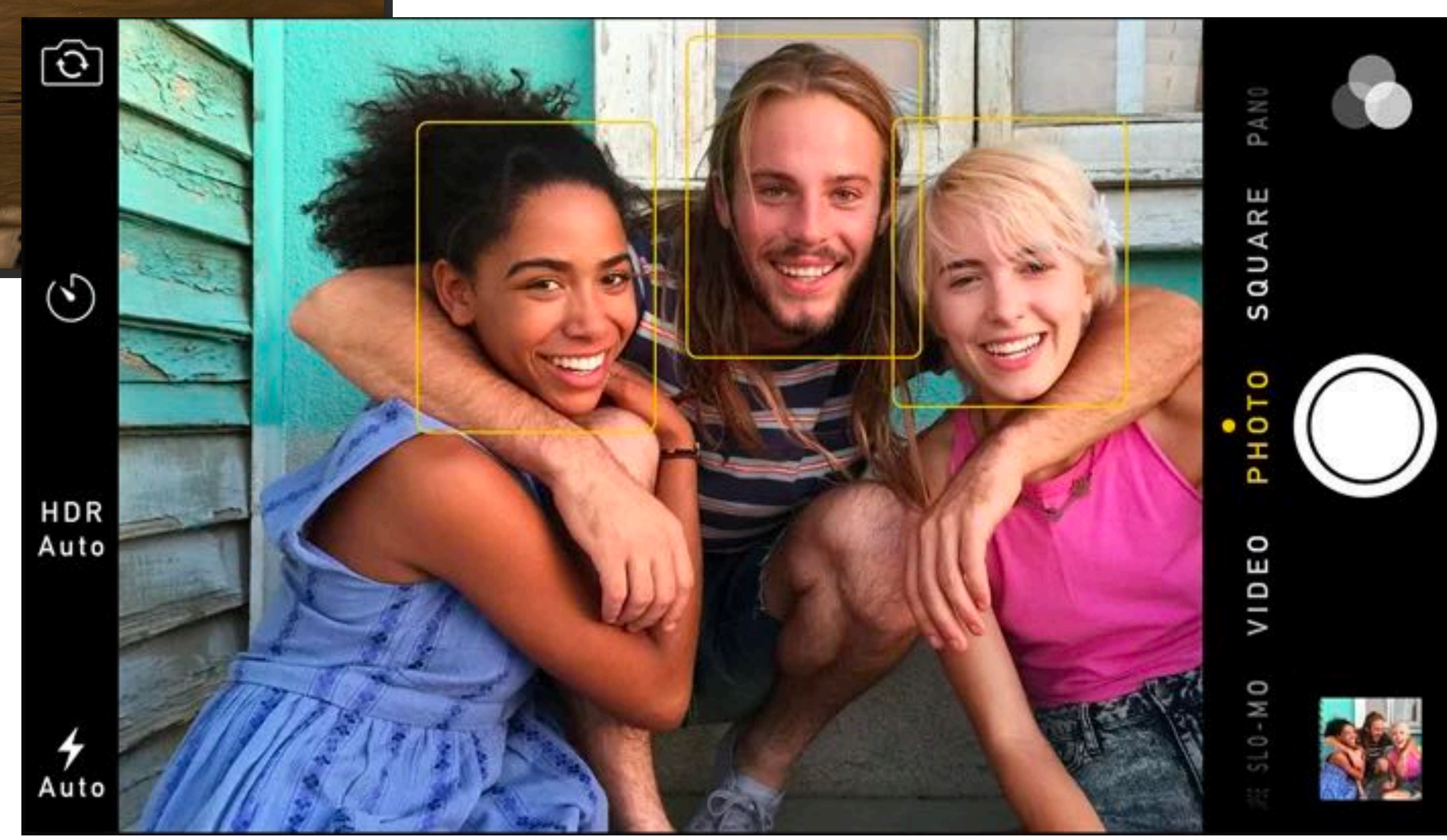
Scene Depth Estimate

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

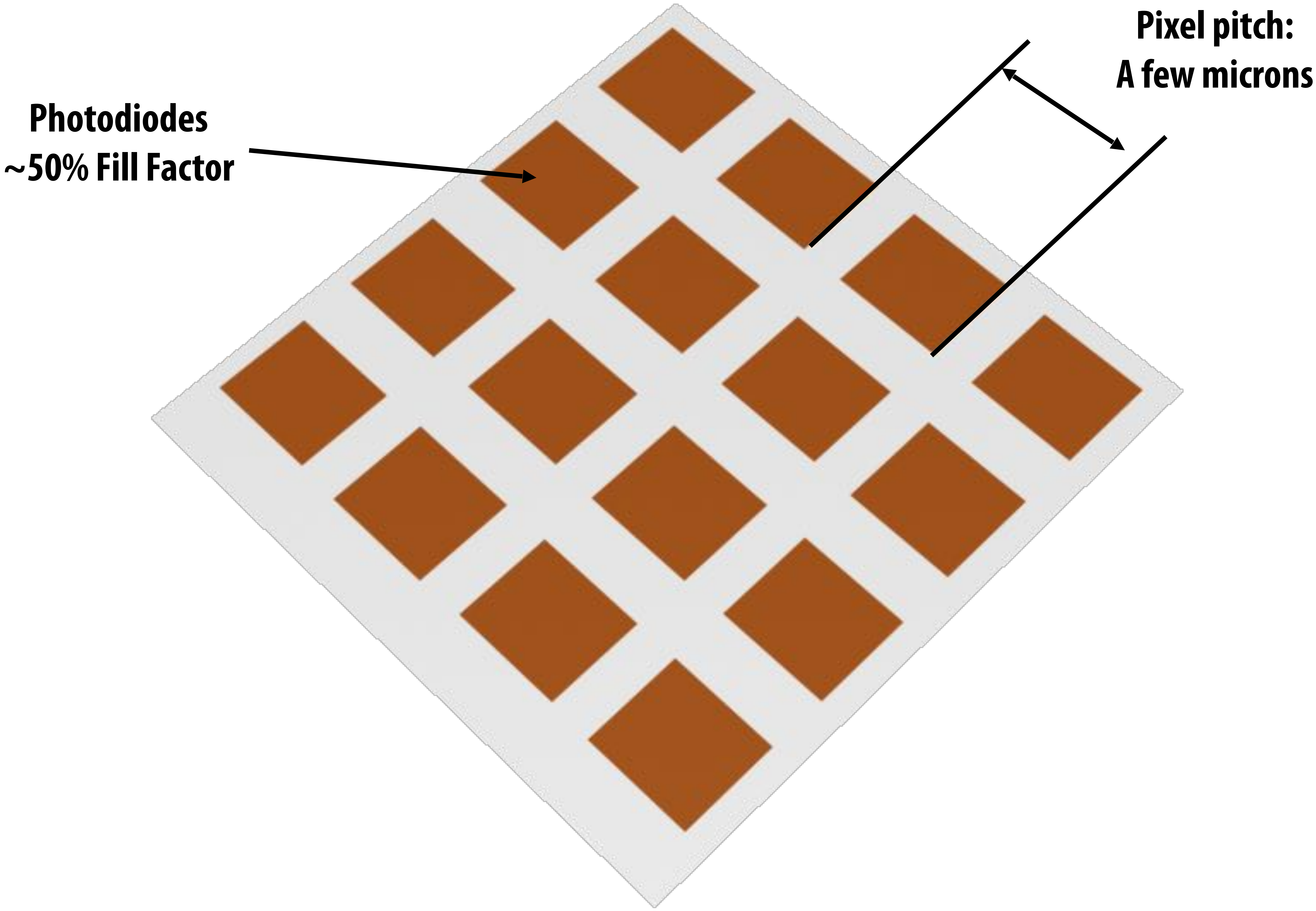
Put center of image in focus

Detect faces and focus on closest/largest face

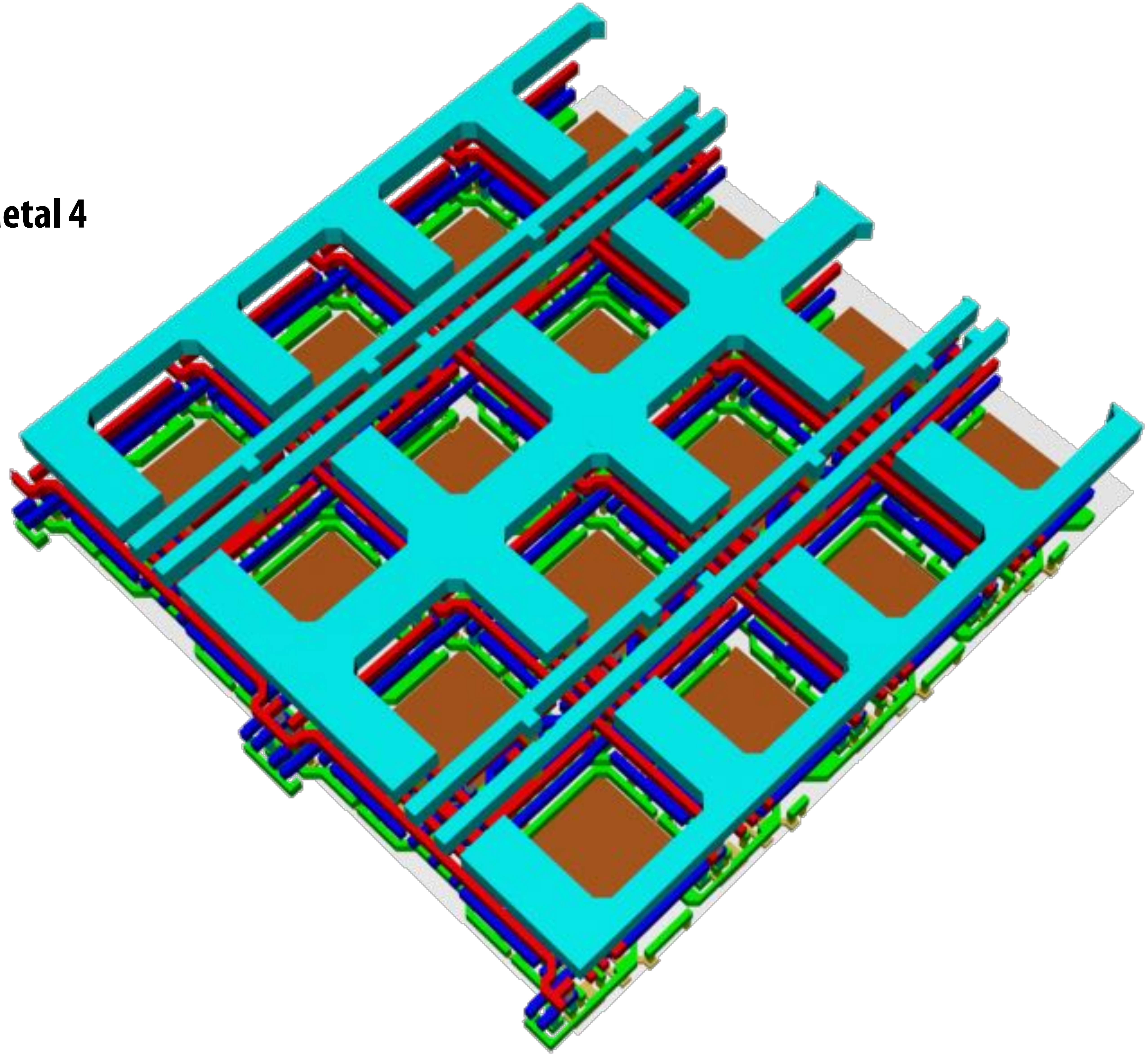
Image credit: DPReview:
https://www.dpreview.com/articles/9174241280/configuring-your-5d-mark-iii-af-for-fast-action

The Sensor

Front-side-illuminated (FSI) CMOS

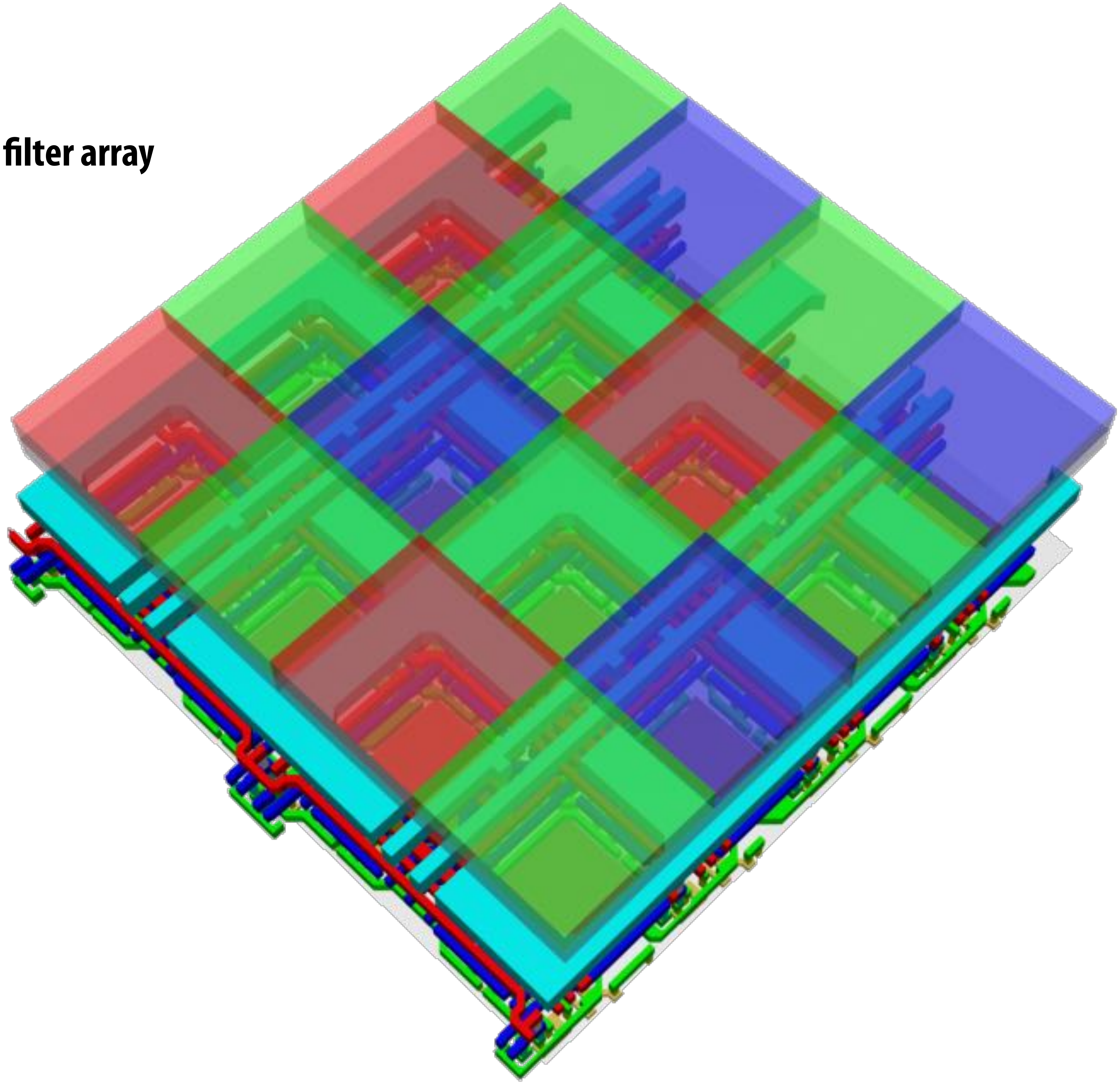


Courtesy R. Motta, Pixim



Courtesy R. Motta, Pixim

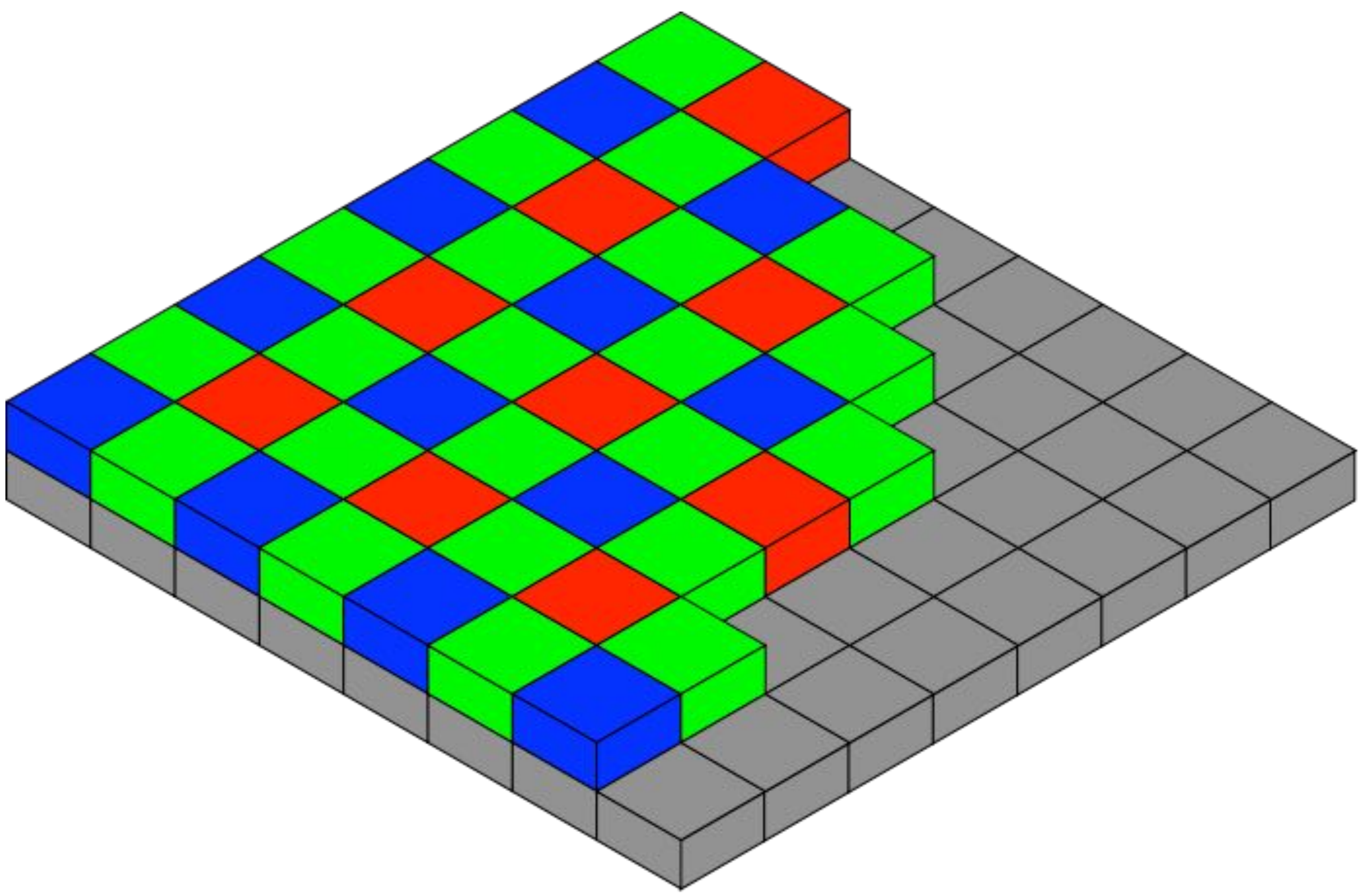
Color filter array



Courtesy R. Motta, Pixim

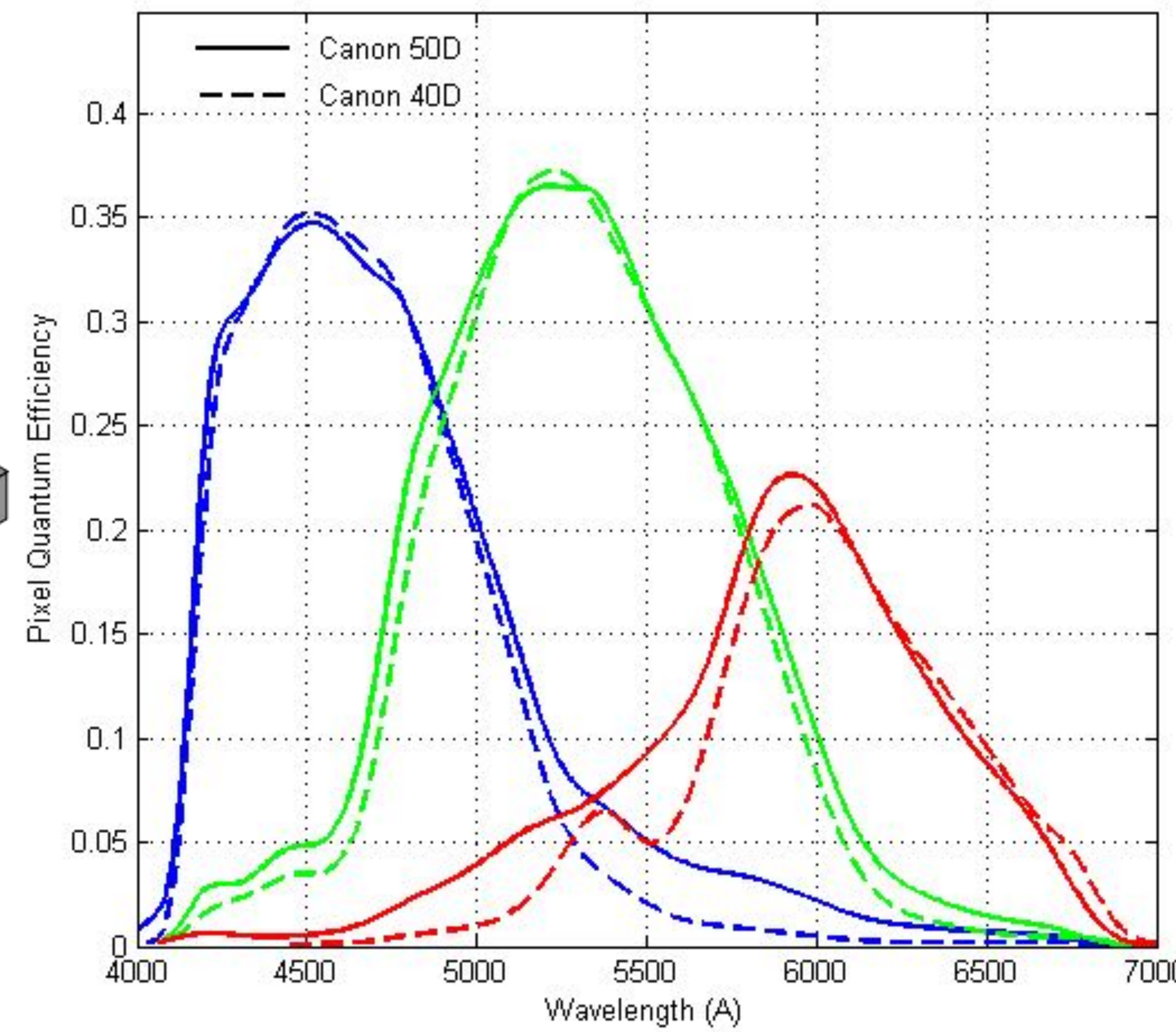
Digital image sensor: color filter array (Bayer mosaic)

- 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

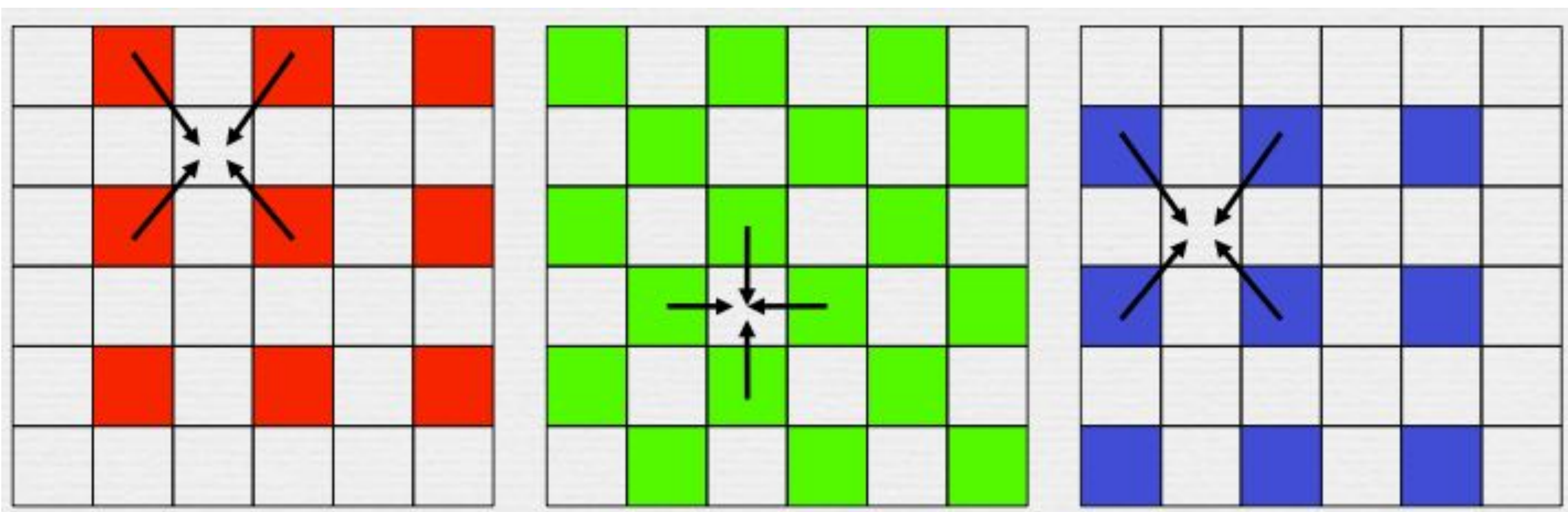
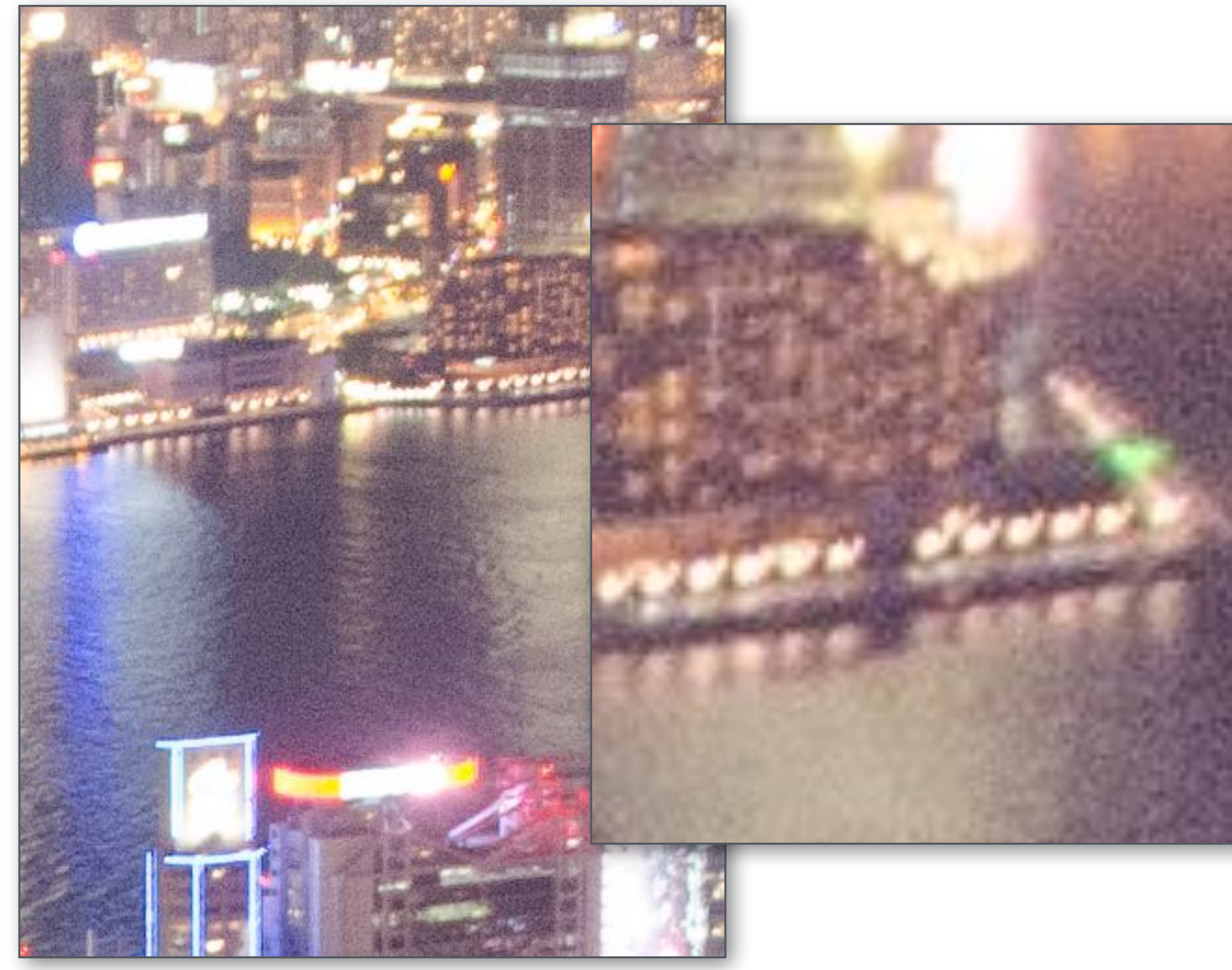


Image credit: Mark Levoy

High dynamic range / exposure / noise

Denoising

Denoising via downsampling



Downsample via point sampling

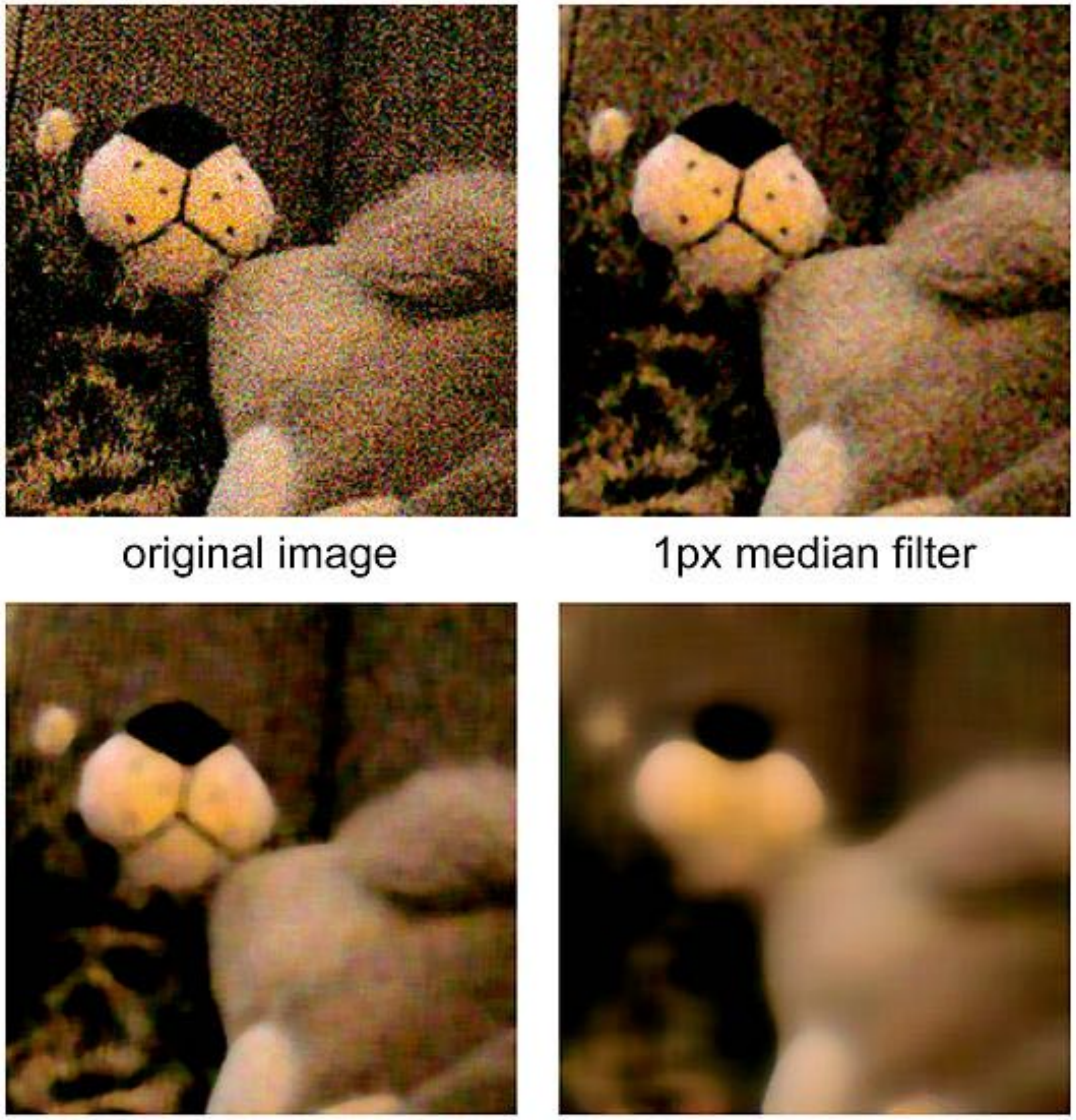
(noise remains)

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 $O(N)$?

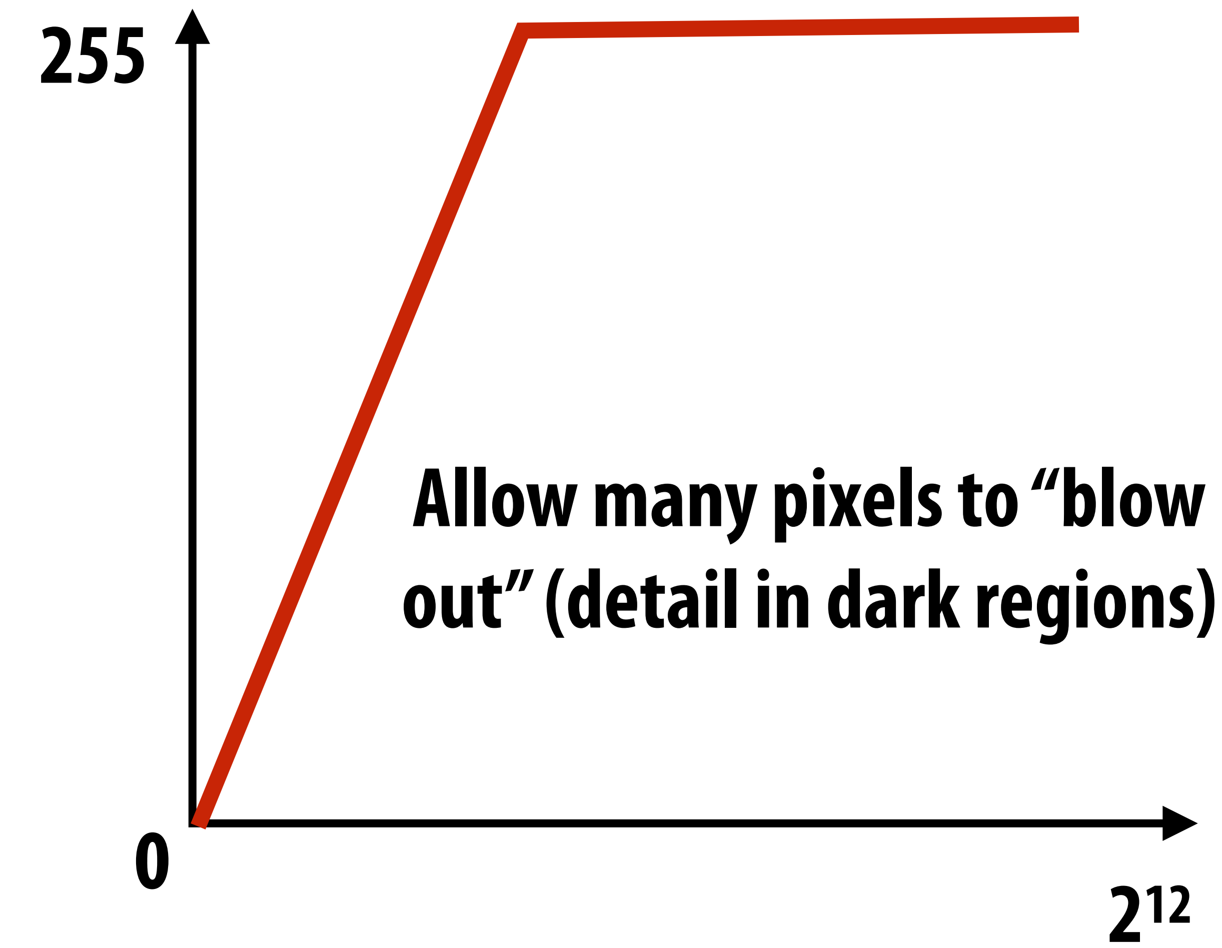
Saturated pixels

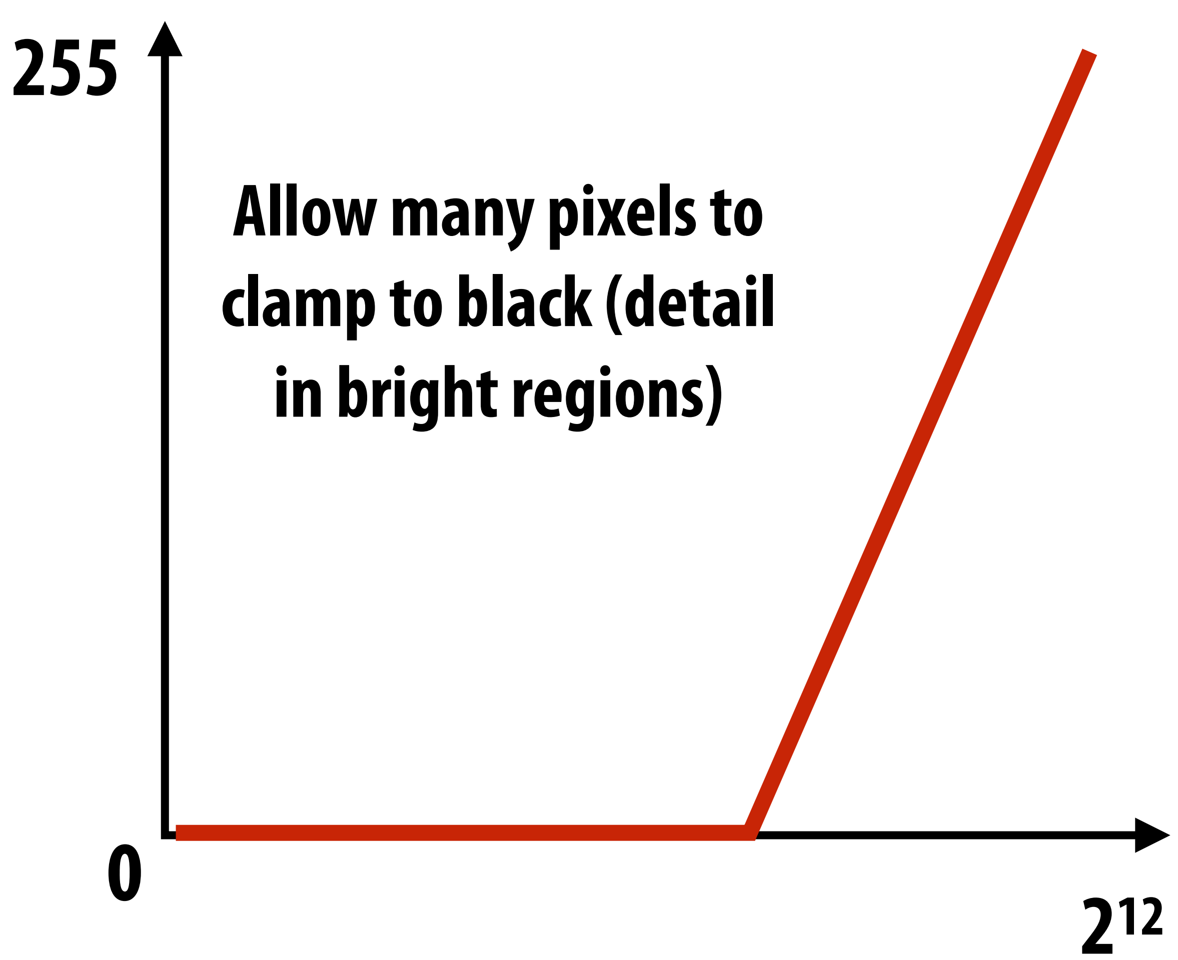
Pixels have saturated (no detail in image)

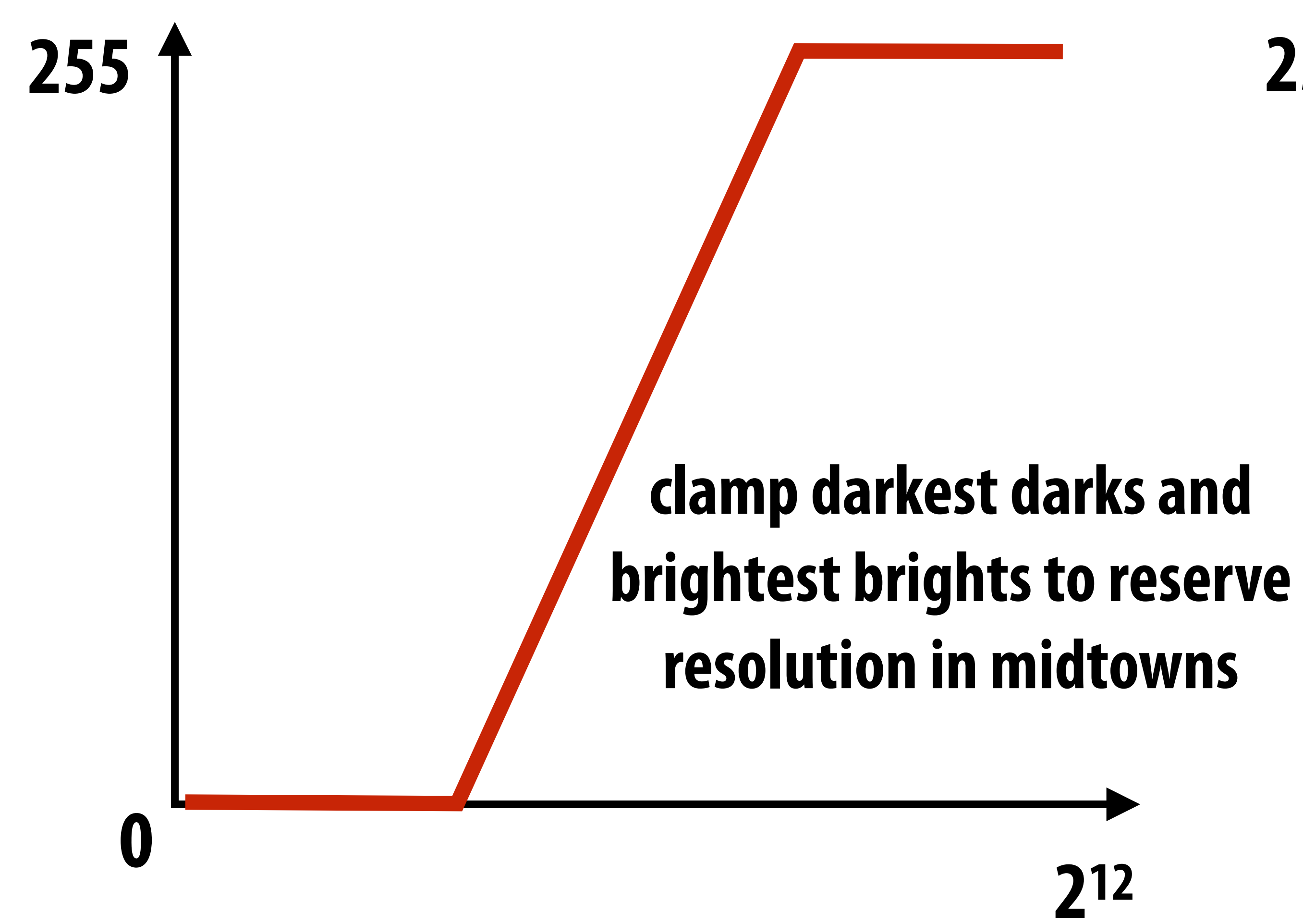
Global tone mapping

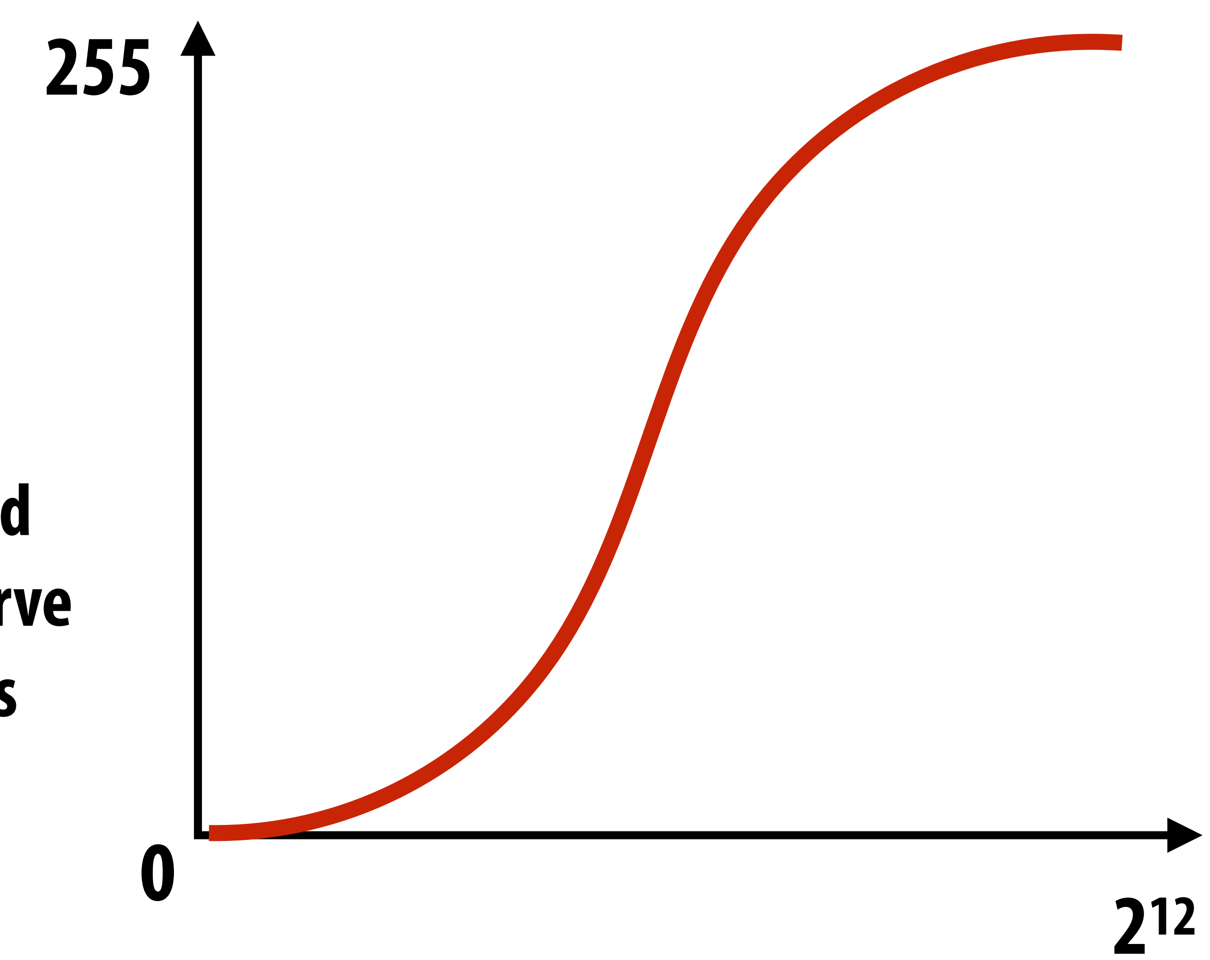
- Measured image values: 10-12 bits/pixel, but common image formats (8-bits/pixel)
- How to convert 12 bit number to 8 bit number?

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

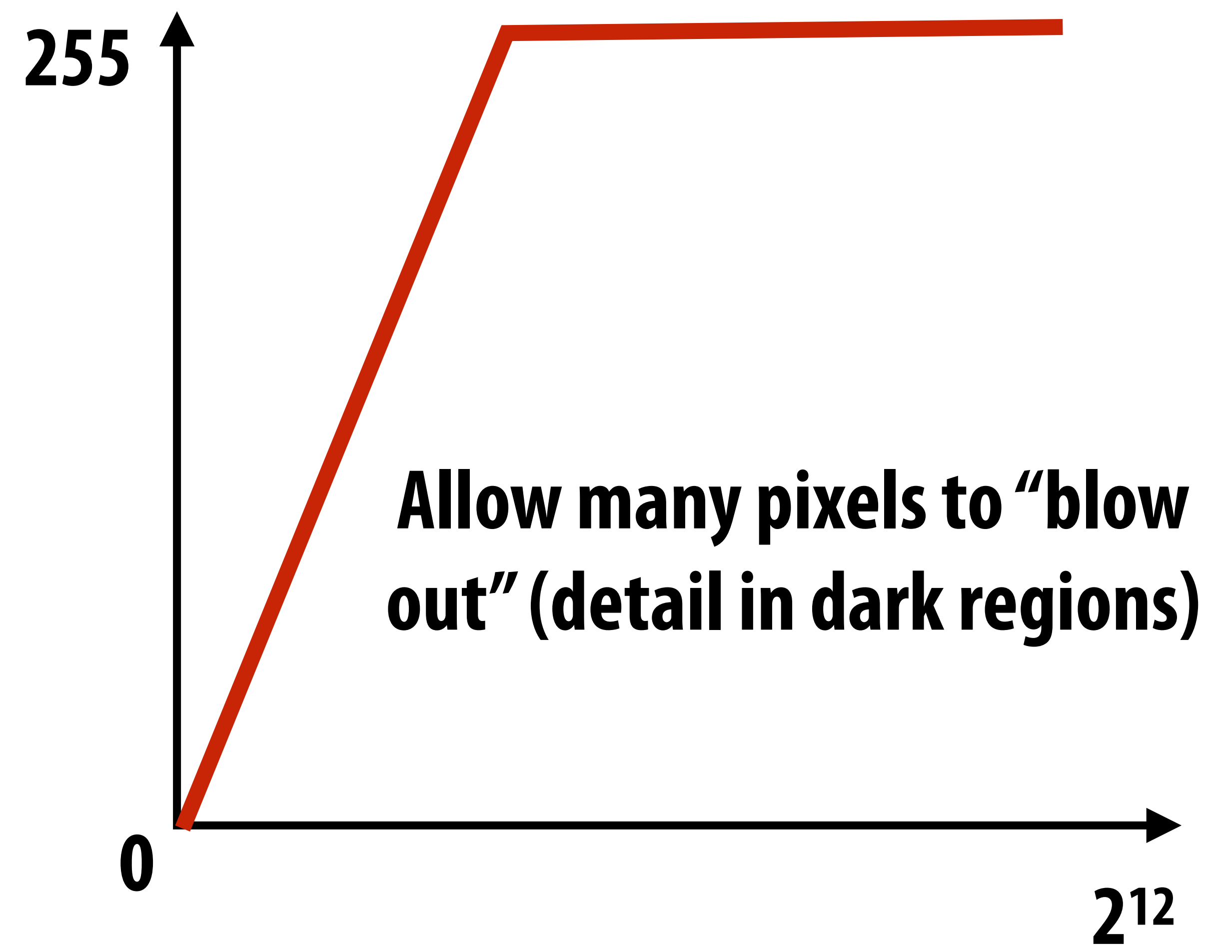


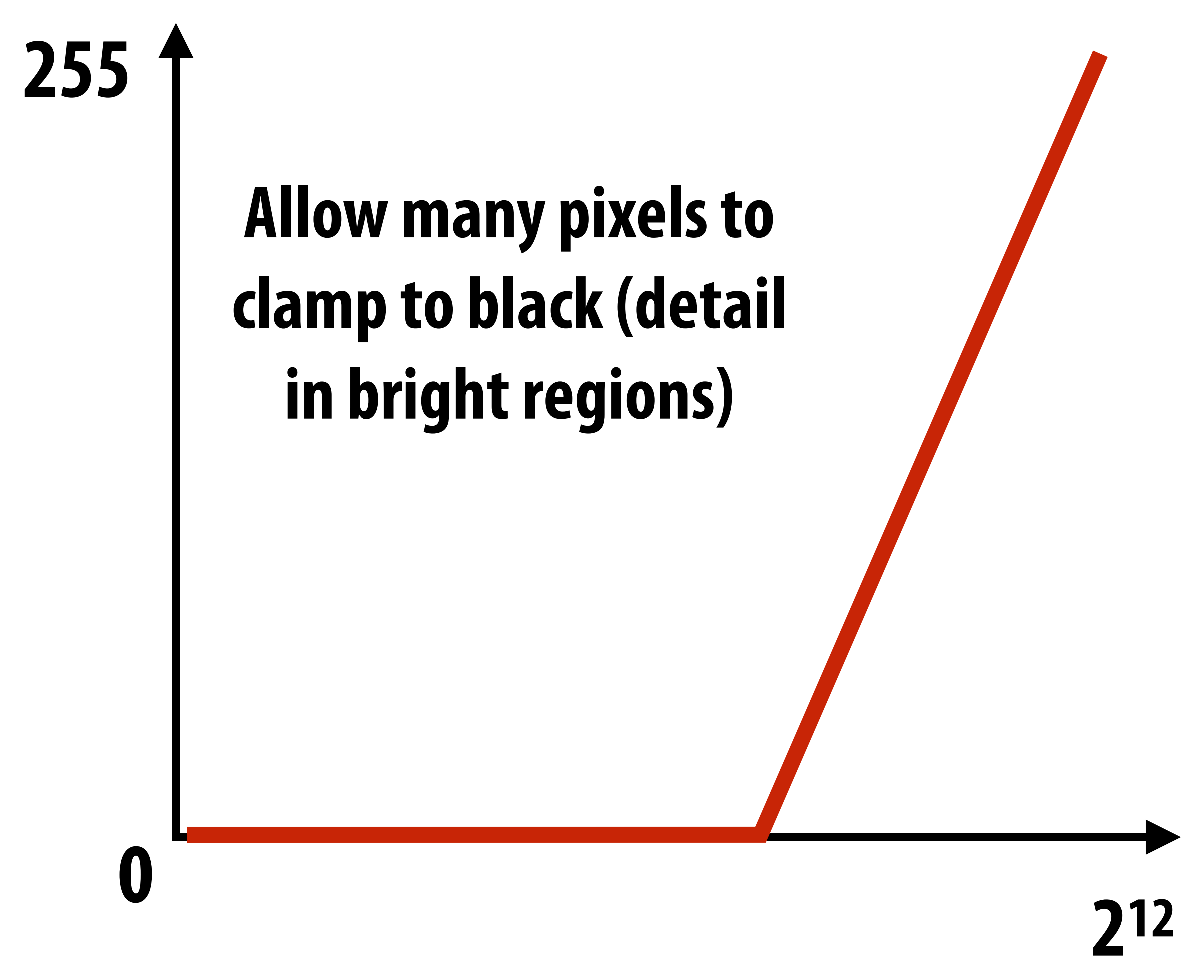






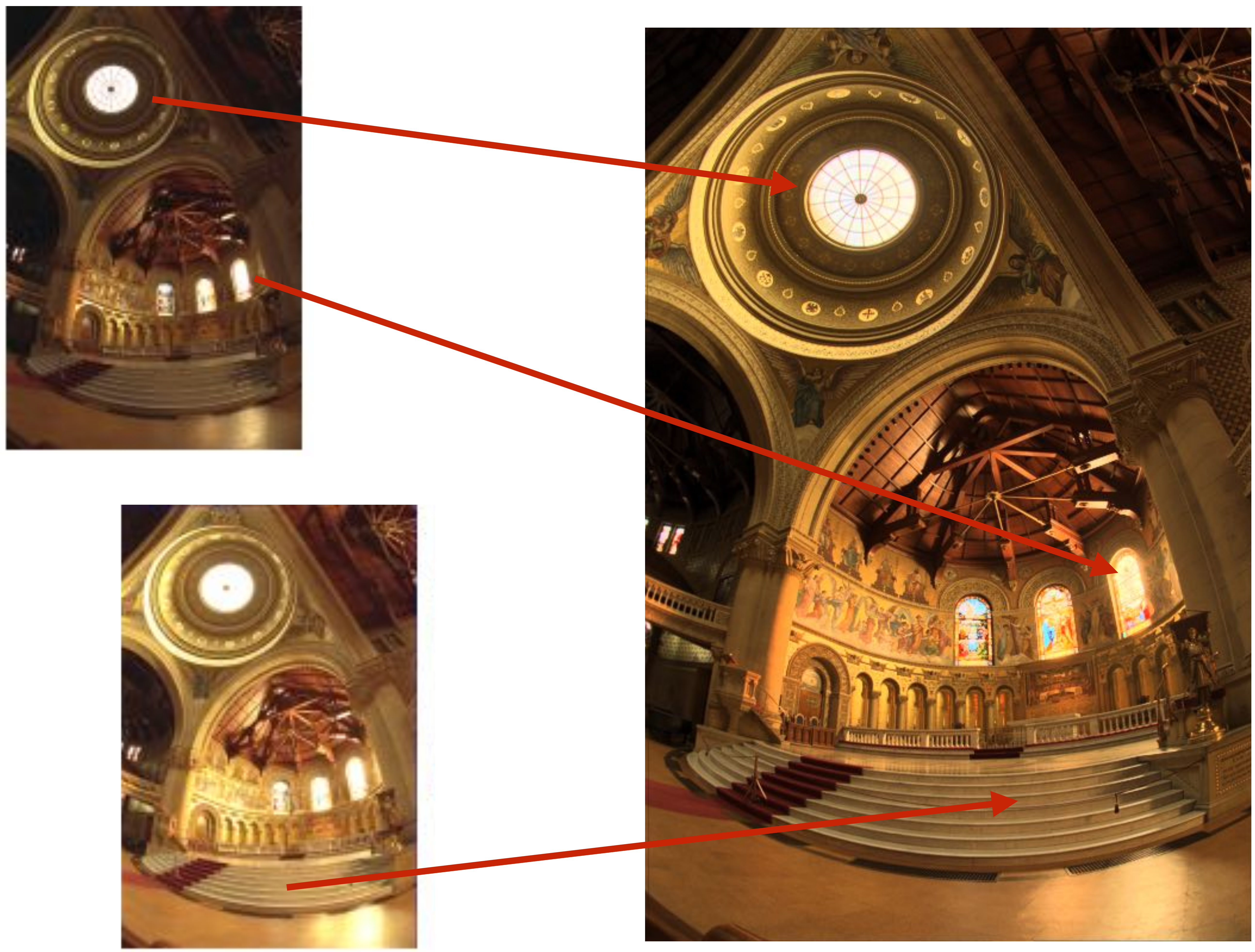
Global tone mapping





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 at all!)

Combined image
(unique weights per pixel)

[Image credit: Mertens 2007]

Challenge of merging images

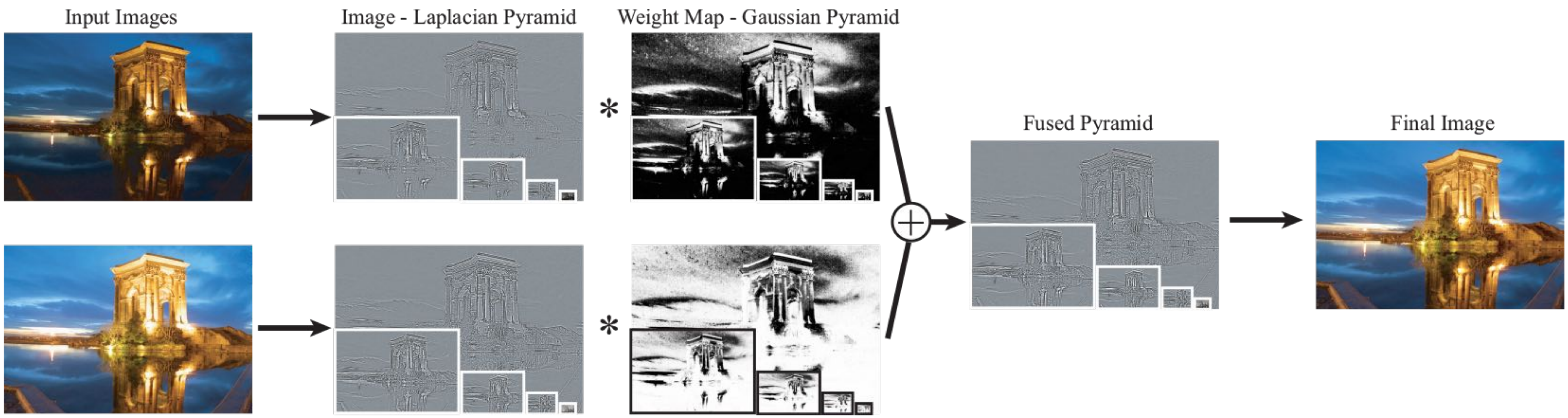
Four different exposures (corresponding weight masks not shown)

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?

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