Lecture 16:

The Digital Camera Pipeline

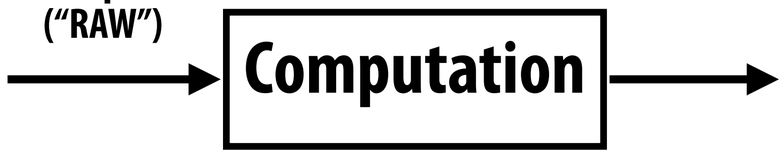
Computer Graphics: Rendering, Geometry, and Image Manipulation
Stanford CS248A, Winter 2026

Today's theme

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

Computation (computer graphics, image processing, and ML) is a fundamental aspect of producing high-quality photographs.

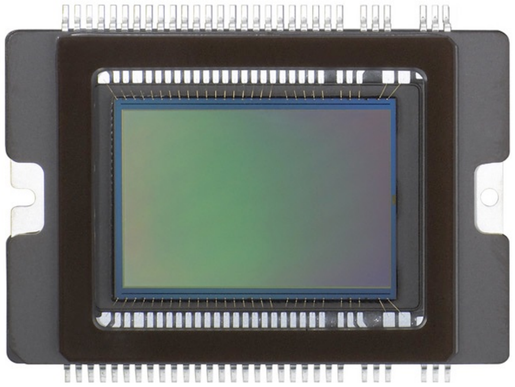
Sensor output ("RAW")



Beautiful image that impresses your Instagram friends

Camera cross section

Sensor



Canon 14 MP CMOS Sensor
(14 bits per pixel)

Image credit: Canon (EOS M)

Camera cross section

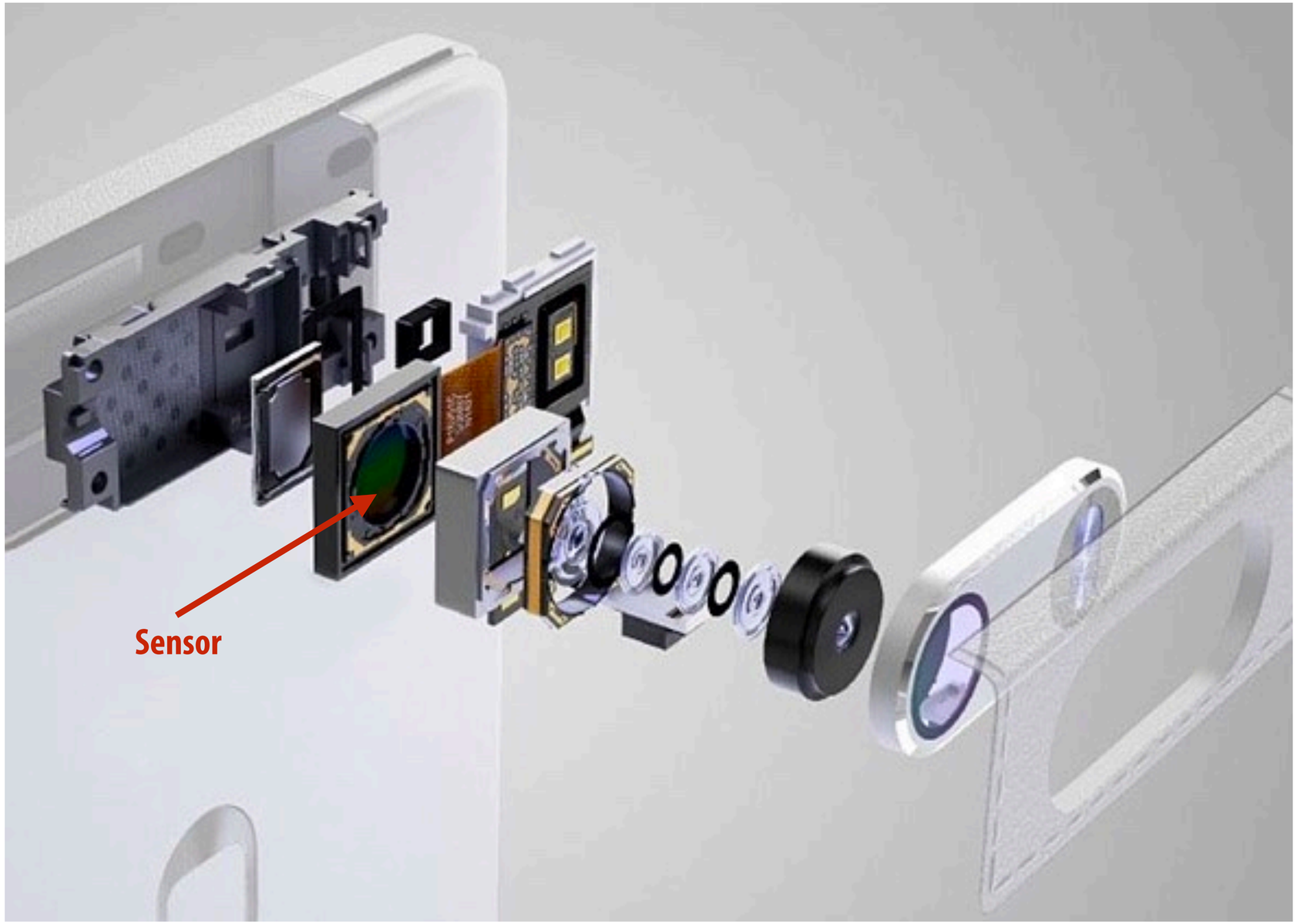
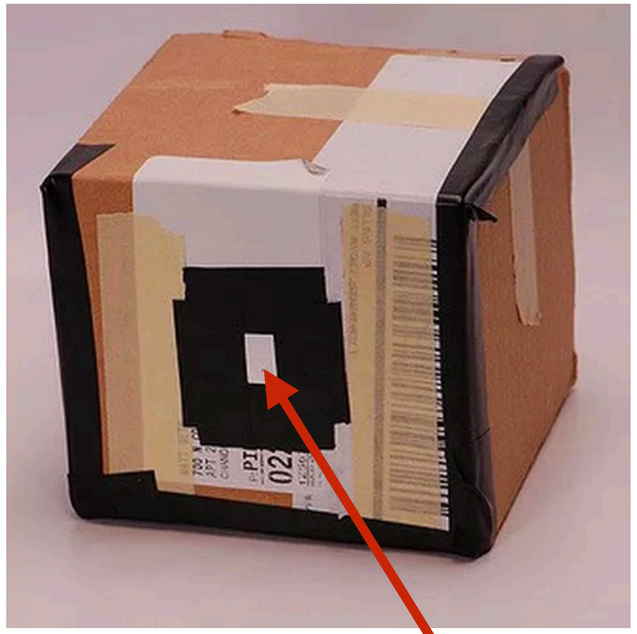
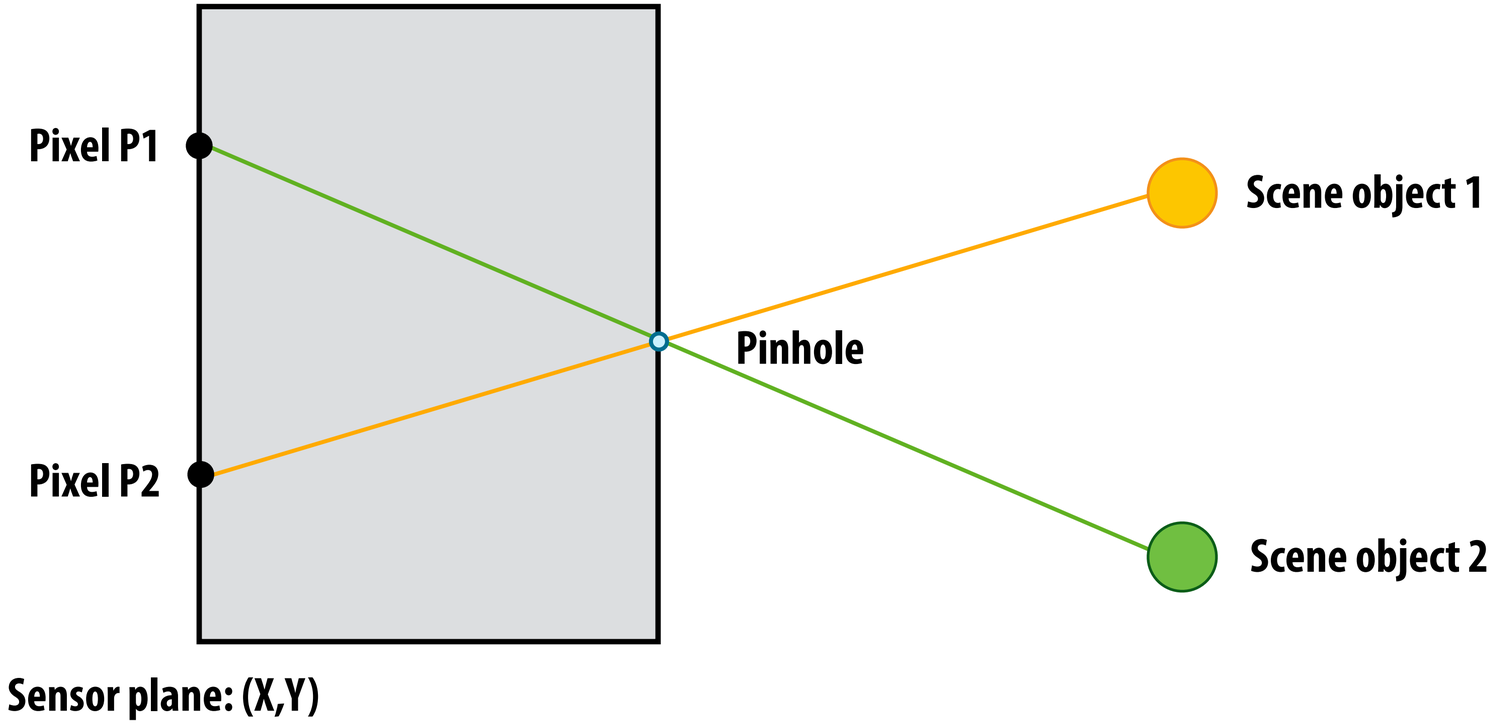


Image credit: https://www.dpreview.com/news/3717128828/the-future-is-bright-technology-trends-in-mobile-photography

Part 1: the lens

Pinhole camera (no lens)



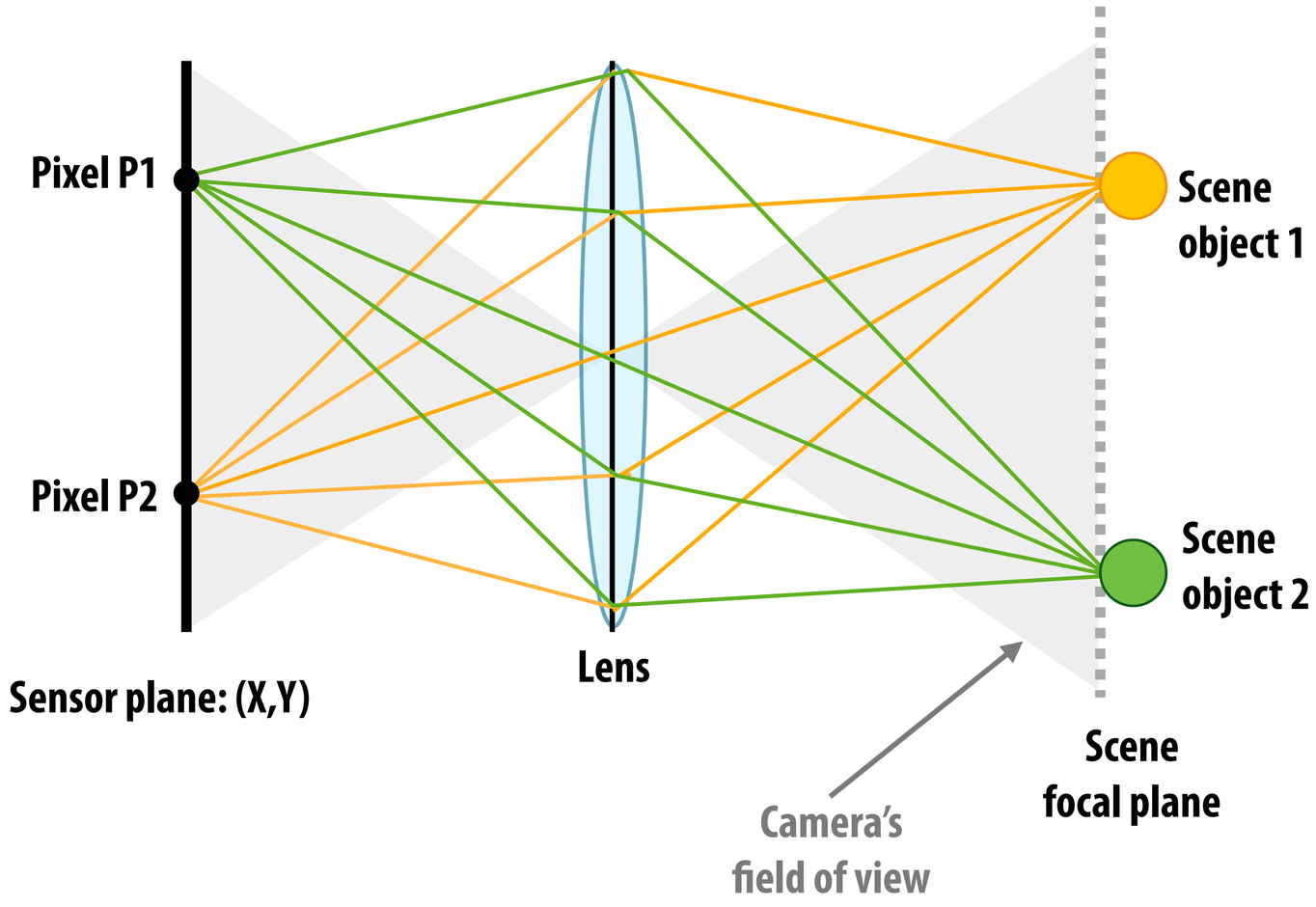


What does a lens do?

A lens refracts light.

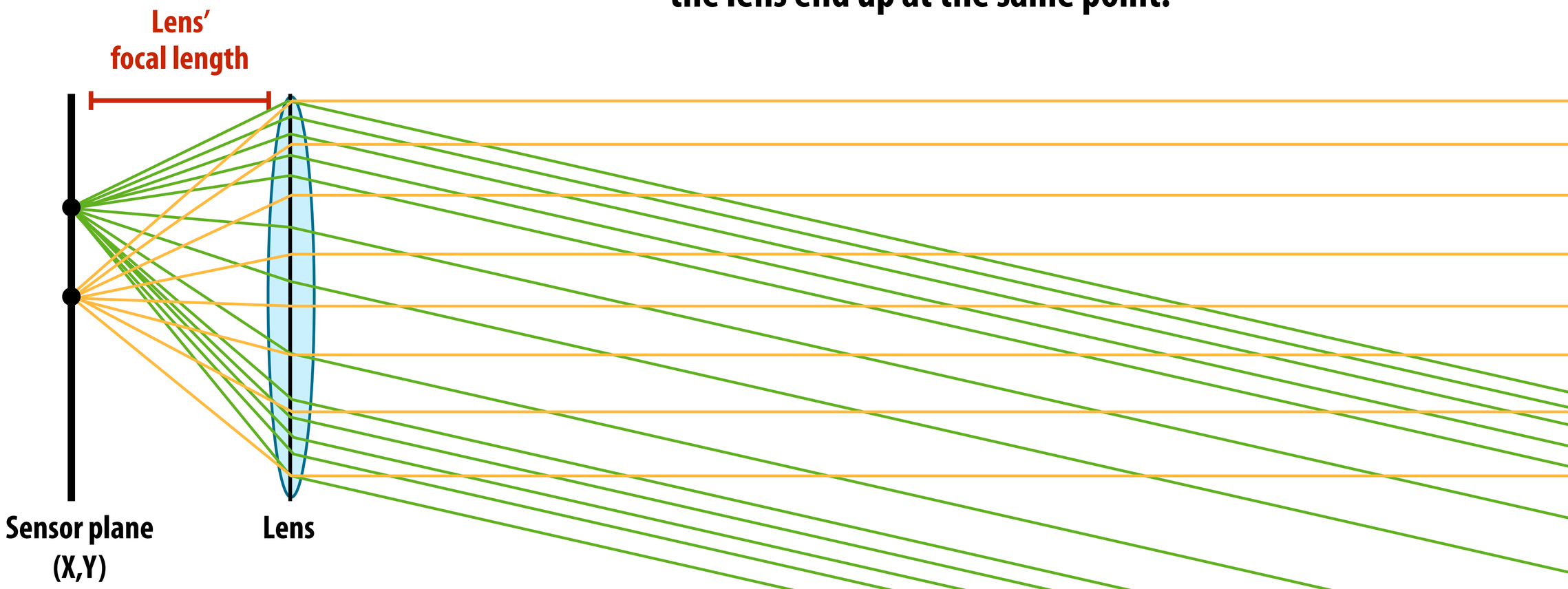
Camera with lens: every sensor pixel accumulates all rays of light that pass through lens aperture and refract toward that pixel

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



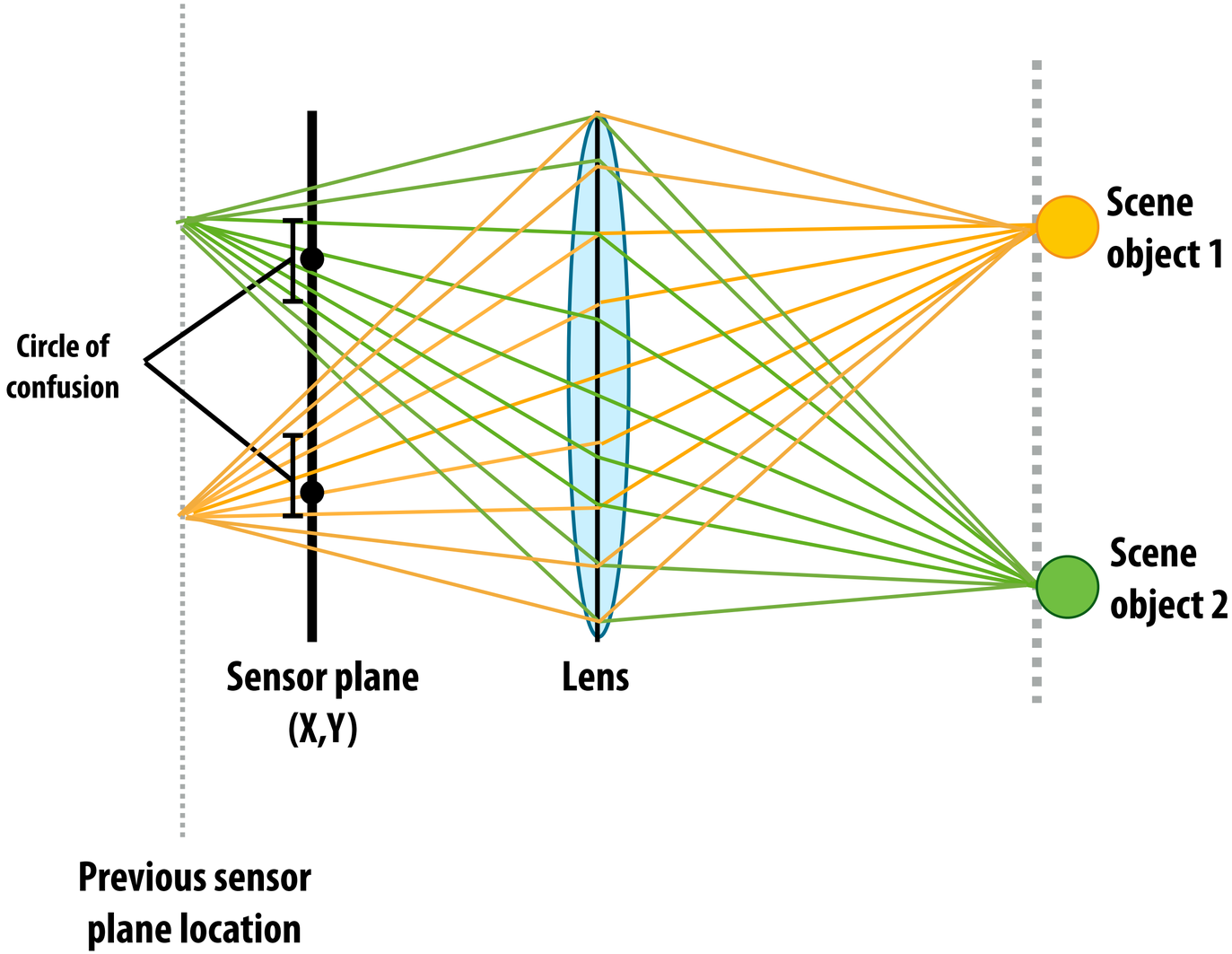
“Focus at infinity”

All rays of light from the same direction that hit the lens end up at the same point.



Out of focus camera

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



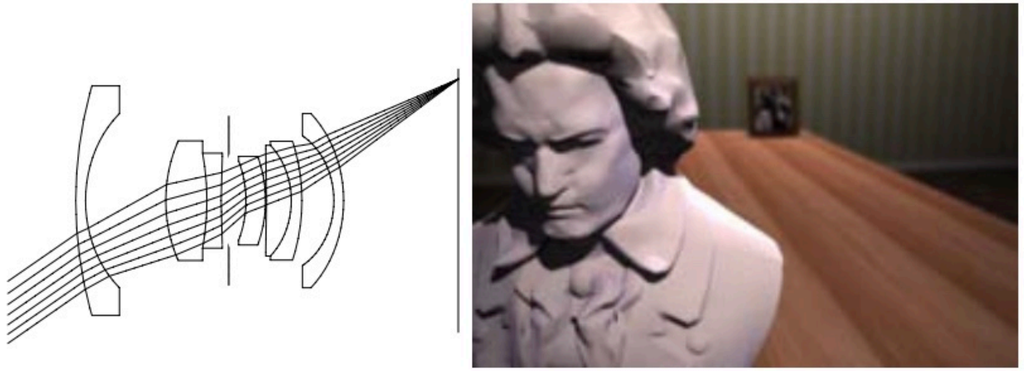
Bokeh

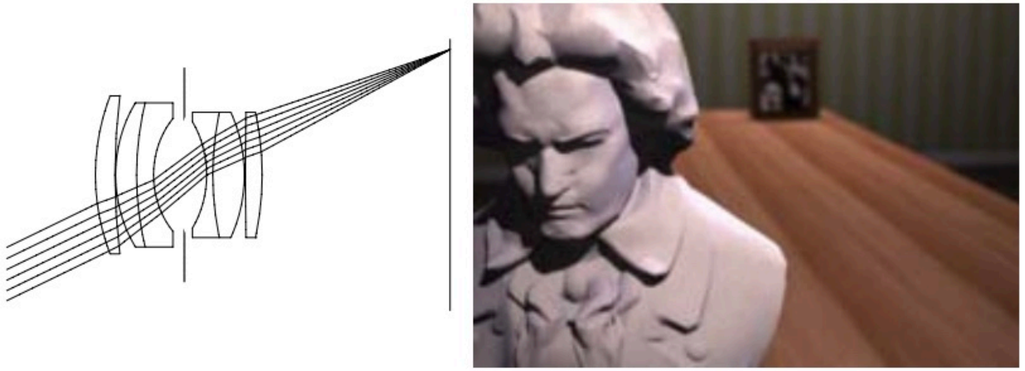
Sharp foreground, defocused background

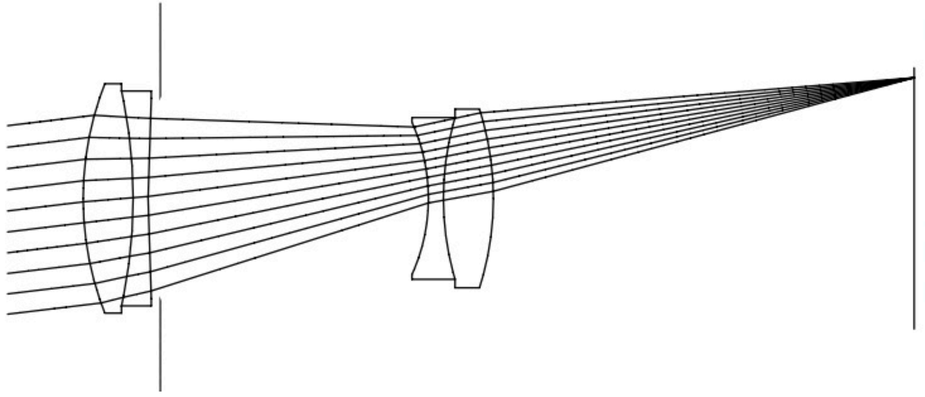
Common technique to emphasize the subject in a photo

Simulating a more realistic camera model in a raytracer

- Goal: integrate light incident on a pixel that arrives via passing through the last lens element
- The last lens element is just an area source! So sample it!
  - Trace ray out of camera by refracting it through the camera's lens system







[Kolb et al. 1995]

Problem: cell phone camera lens(es) (small aperture)

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)

(a) Input image with detected face

(c) Mask + disparity from DP

(d) Our output synthetic shallow depth-of-field image

Input image /w detected face

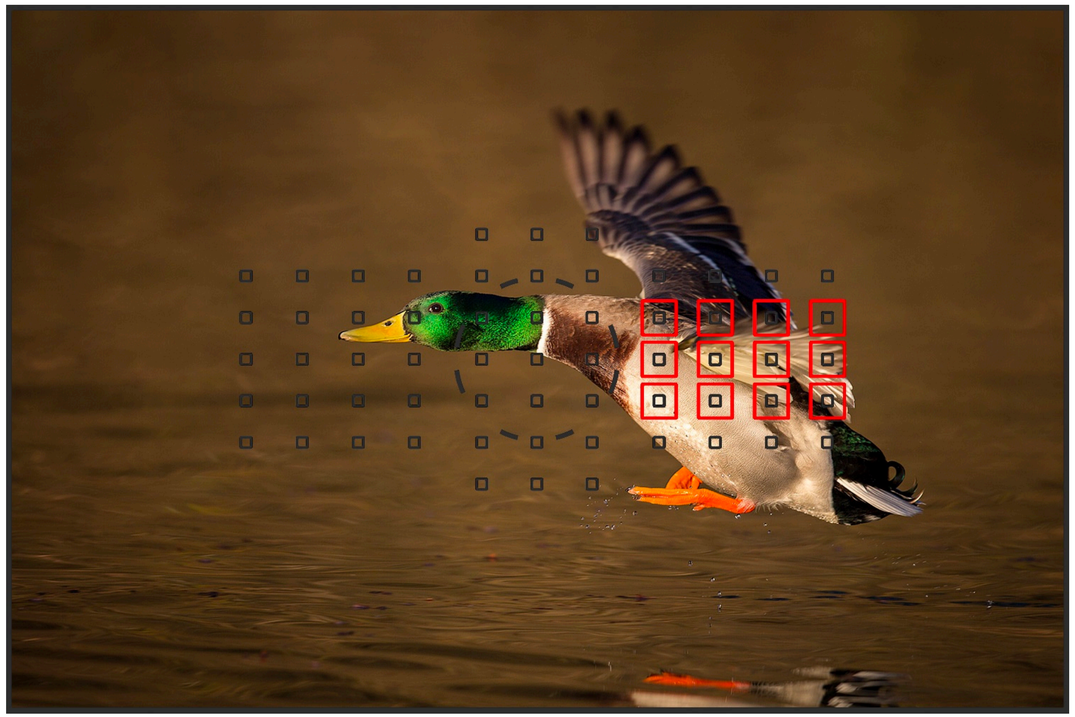
Segmentation

Scene Depth Estimate

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

Image credit: [Wadha 2018]

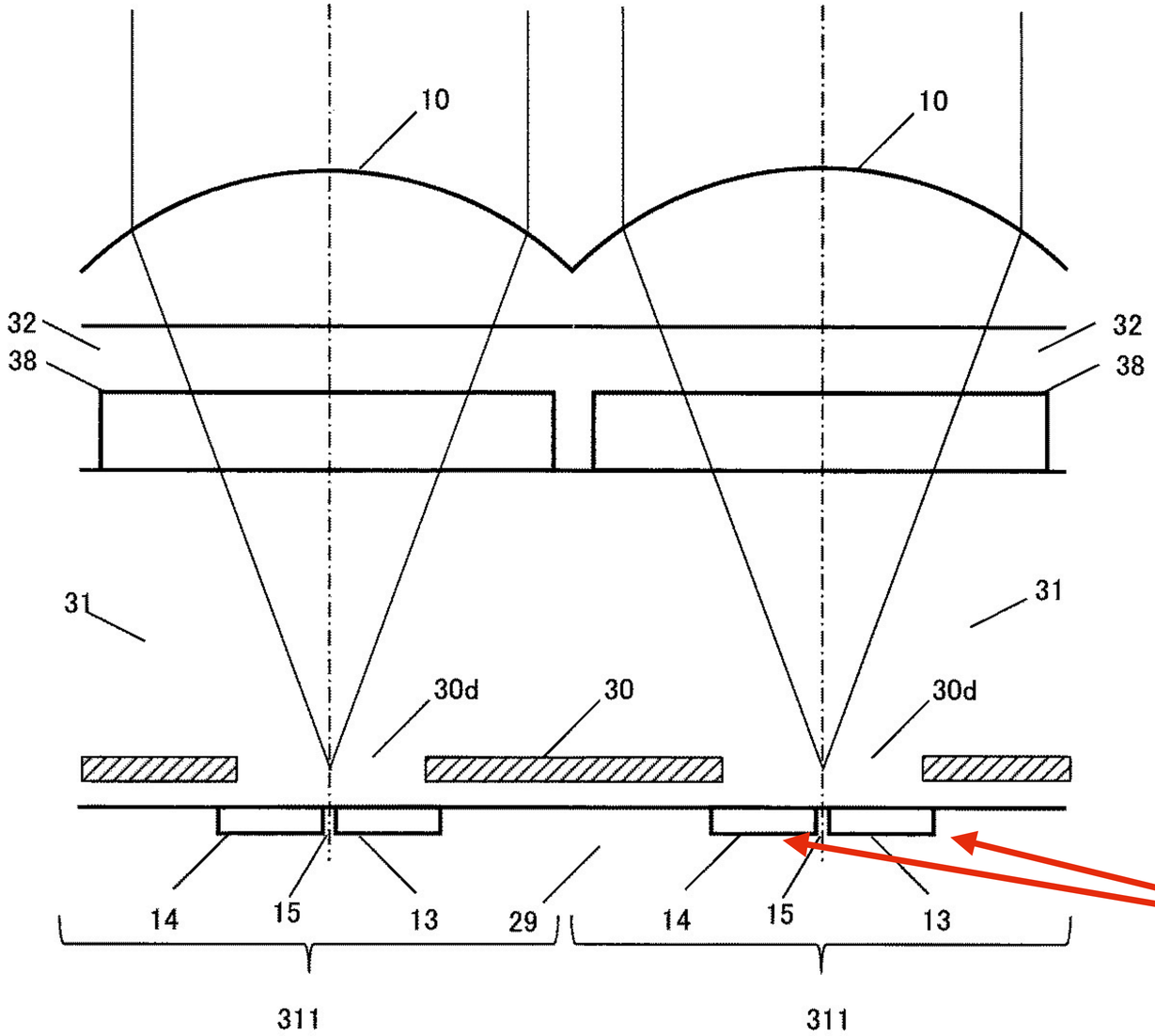
Autofocus: what part of image should be in focus?



Consider possible 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

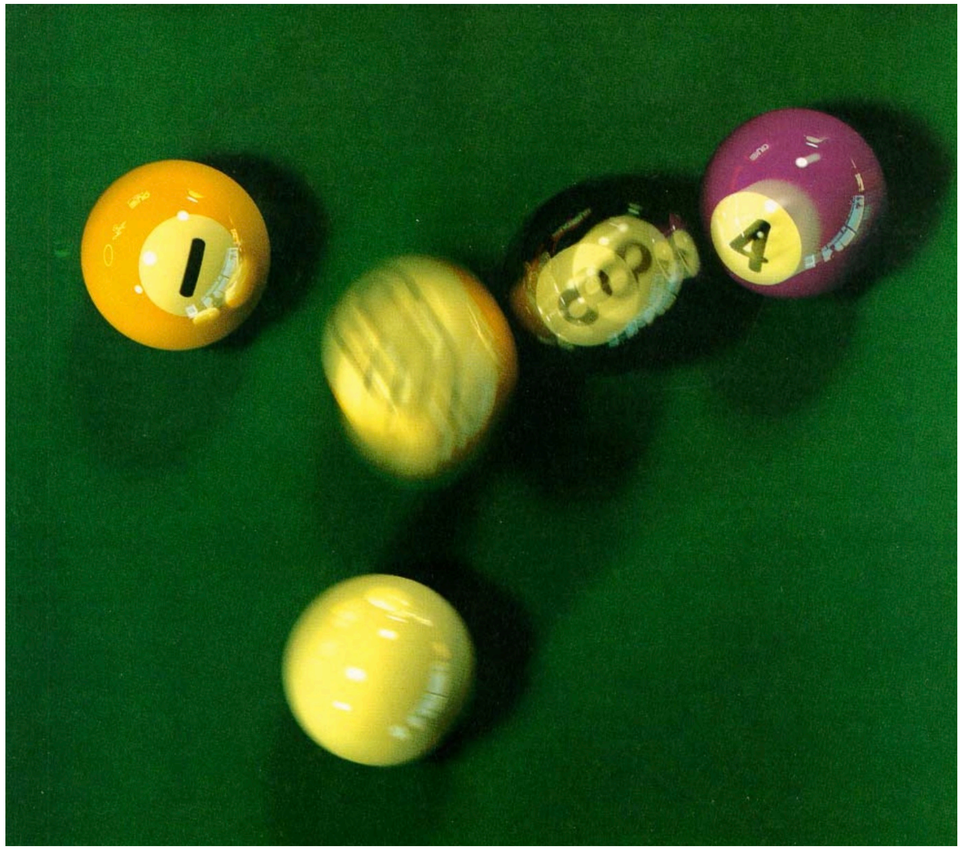
Motion blur

Image credit: https://www.adorama.com/alc/how-to-capture-motion-blur-in-photography/

Motion blur

- Goal: integrate light incident on a pixel that arrives via passing through the last lens element *over a period of time*

- Assign time *t* to ray. Trace ray out of camera by refracting it through the camera's lens system, and intersect with scene objects positioned at time *t*
  - Requires object positions to be a function of t



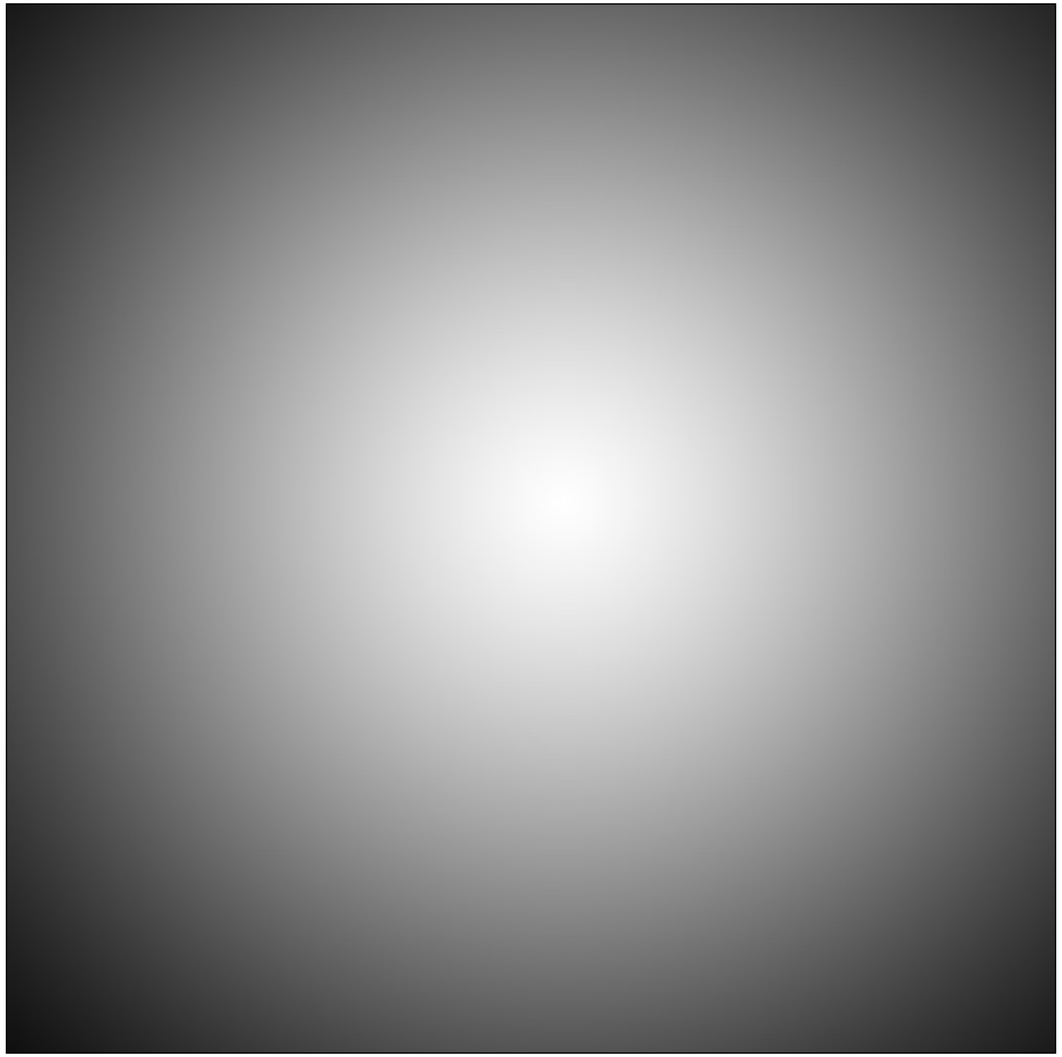
[Cook 84]

Artifacts arising from lenses

Vignetting

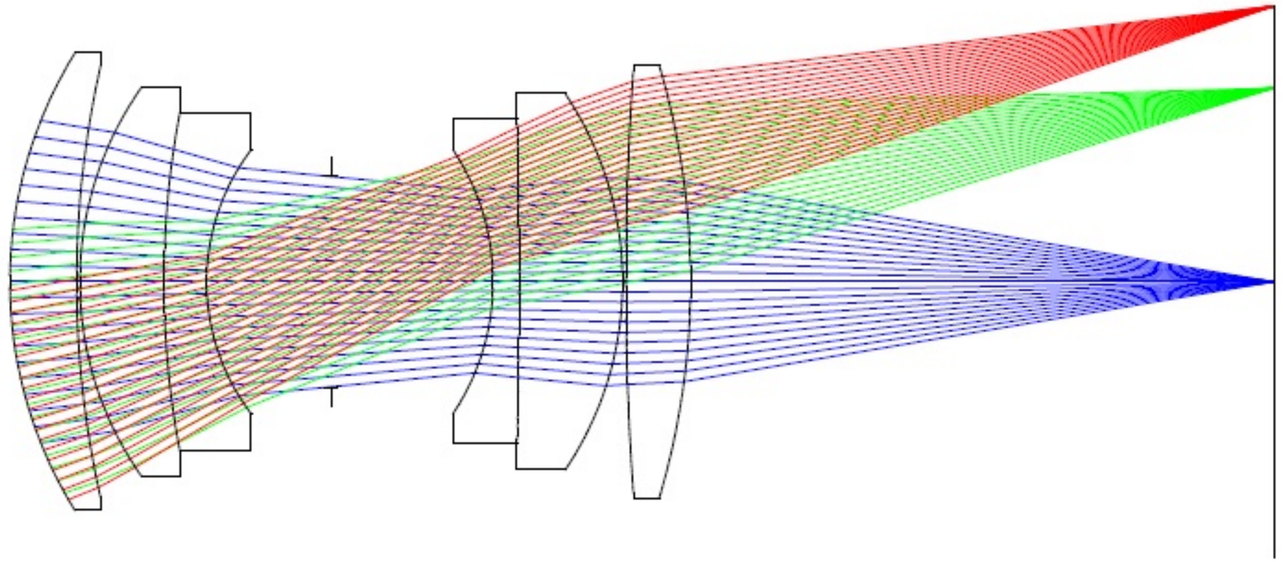
This is a photograph of a white wall

(Note: I contrast-enhanced the image to show effect more prominently)



Types of vignetting

Optical vignetting: less light reaches edges of sensor due to physical obstruction in lens



Pixel vignetting: light reaching pixel at an oblique angle is less likely to hit photosensitive region than light incident from straight above (e.g., obscured by electronics)

- Microlens reduces pixel vignetting

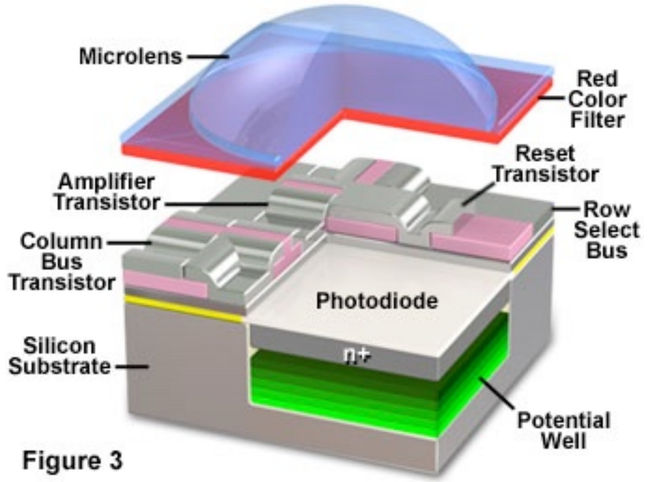
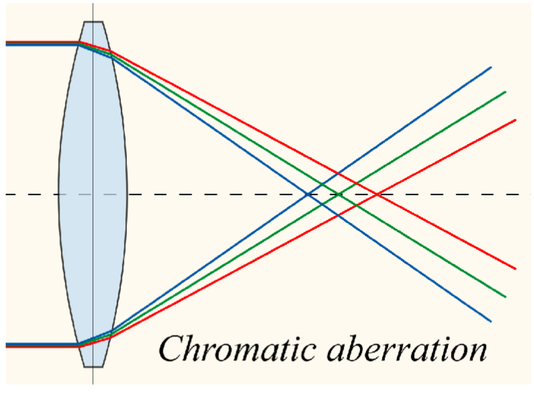


Image credit: Mark Butterworth

Chromatic aberration

Different wavelengths of light are refracted by different amounts



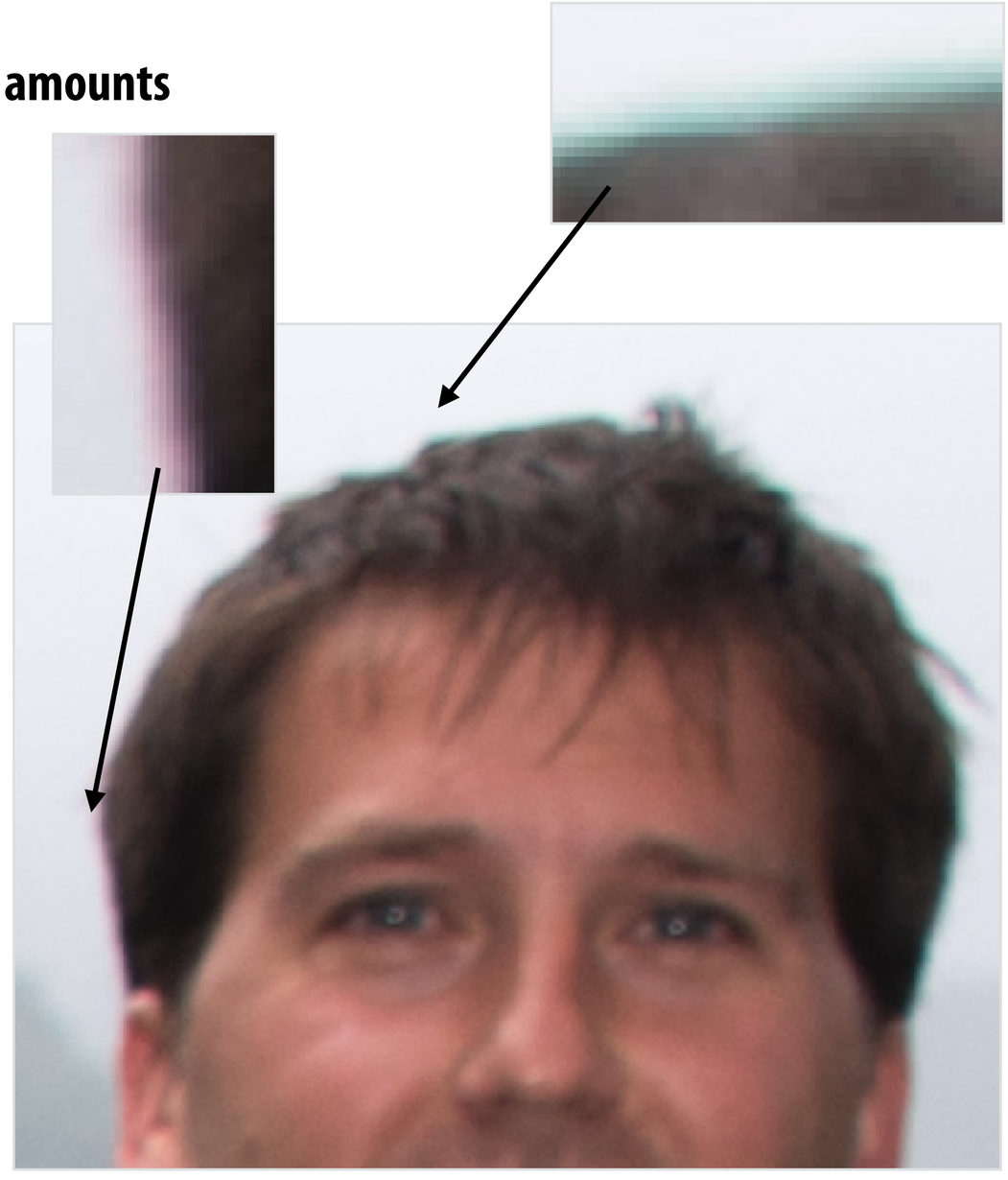
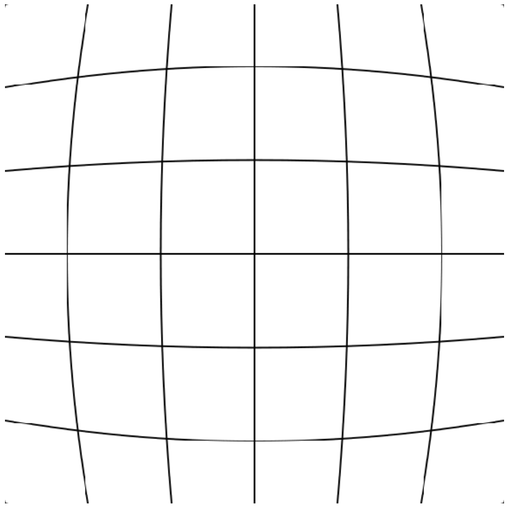


Image credit: Wikipedia

More challenges

- **Chromatic shifts over sensor**
  - Pixel light sensitivity changes over sensor due to interaction with microlens

    (Index of refraction depends on wavelength, so some wavelengths are more likely to suffer from cross-talk or reflection. Ug!)
- **Lens distortion**



Pincushion distortion

Captured Image

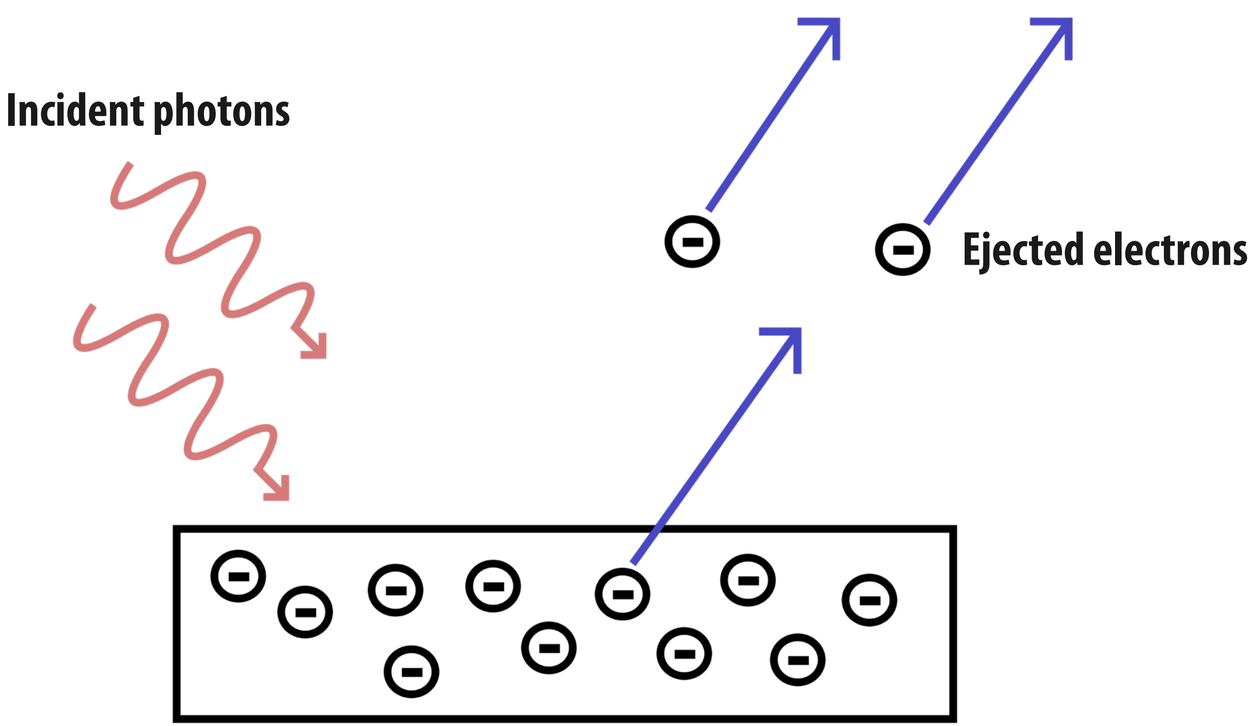
Corrected Image

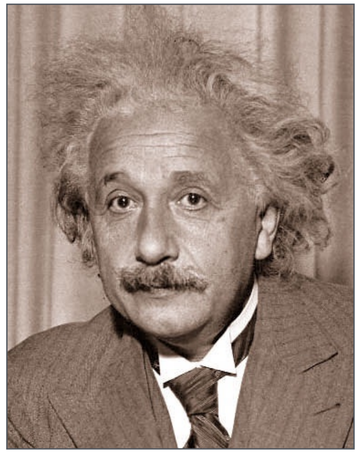
Image credit: PCWorld

Part 2: the sensor

Photoelectric effect

Einstein's Nobel Prize in 1921 "for his services to Theoretical Physics, and especially for his discovery of the law of the photoelectric effect"

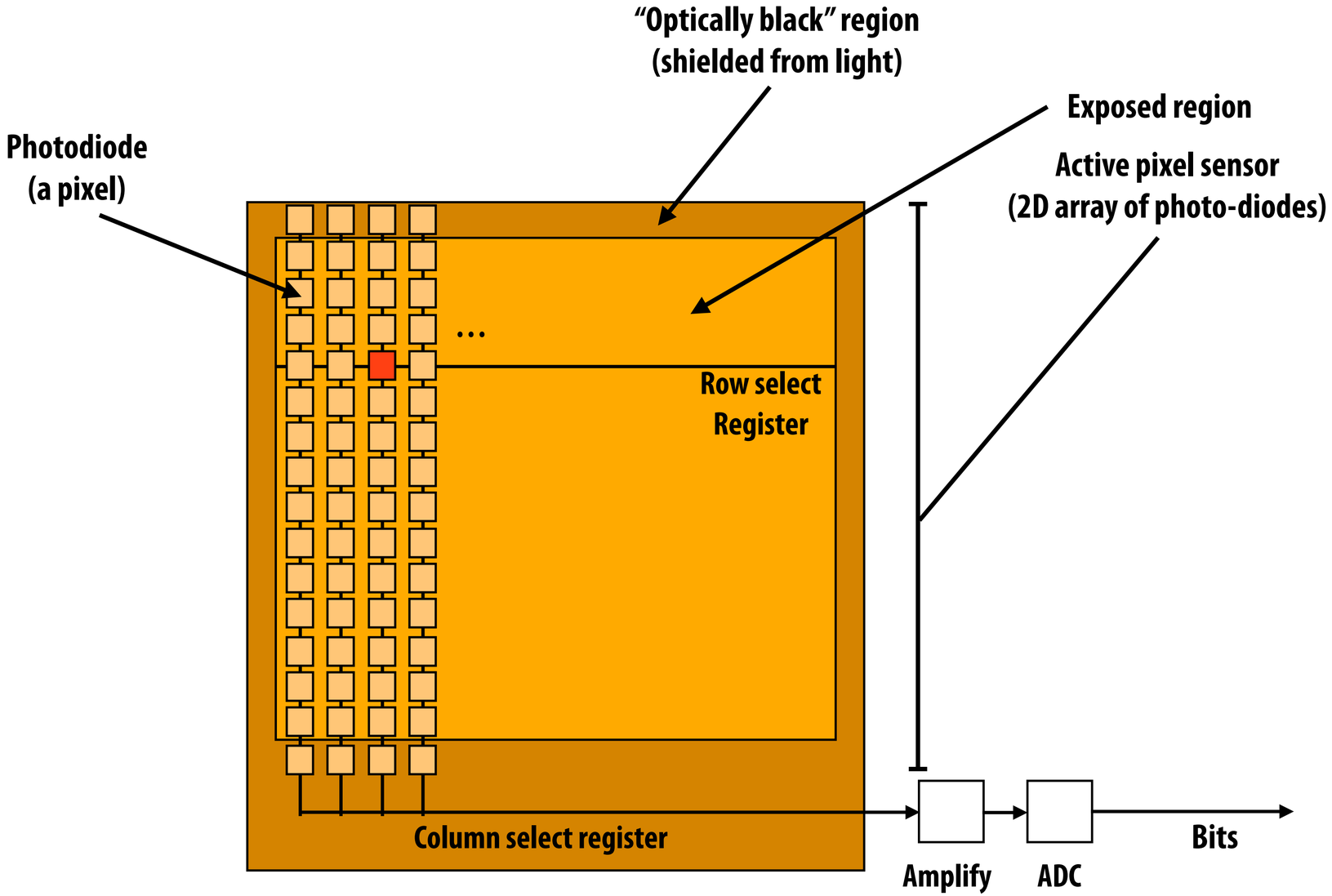




Albert Einstein

Slide credit: Ren Ng

CMOS sensor



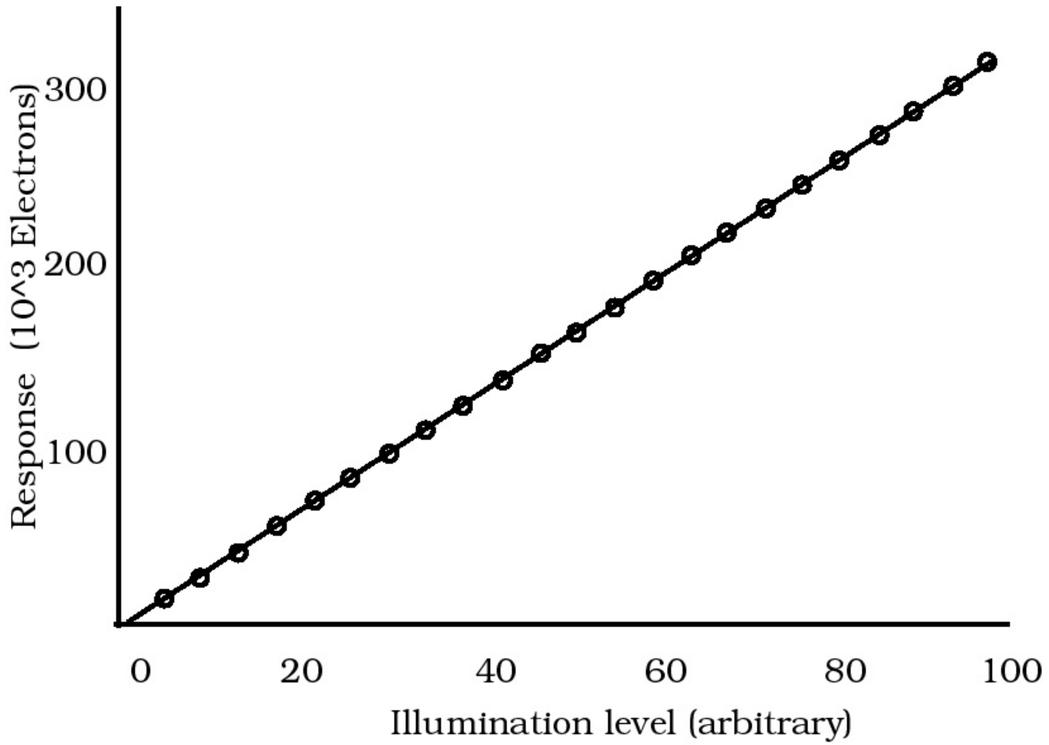
CMOS = complementary metal-oxide semiconductor

CMOS response functions are linear

Photoelectric effect in silicon:

- **Response function from photons to electrons is linear**

  (Some nonlinearity close to 0 due to noise and when close to pixel saturation)



(Epperson, P.M. et al. Electro-optical characterization of the Tektronix TK5 ..., Opt Eng., 25, 1987)

Slide credit: Ren Ng

Quantum efficiency

- Not all photons will produce an electron (depends on quantum efficiency of the device)

$$QE = \frac{\#\,electrons}{\#\,photons}$$

- Human vision: ~15%
- Typical digital camera: < 50%
- Best back-thinned CCD: > 90% (e.g., telescope)

Slide credit: Ren Ng

Sensing Color

Review: electromagnetic spectrum

Describes distribution of power (energy/time) by wavelength
Below: spectrum of various common light sources

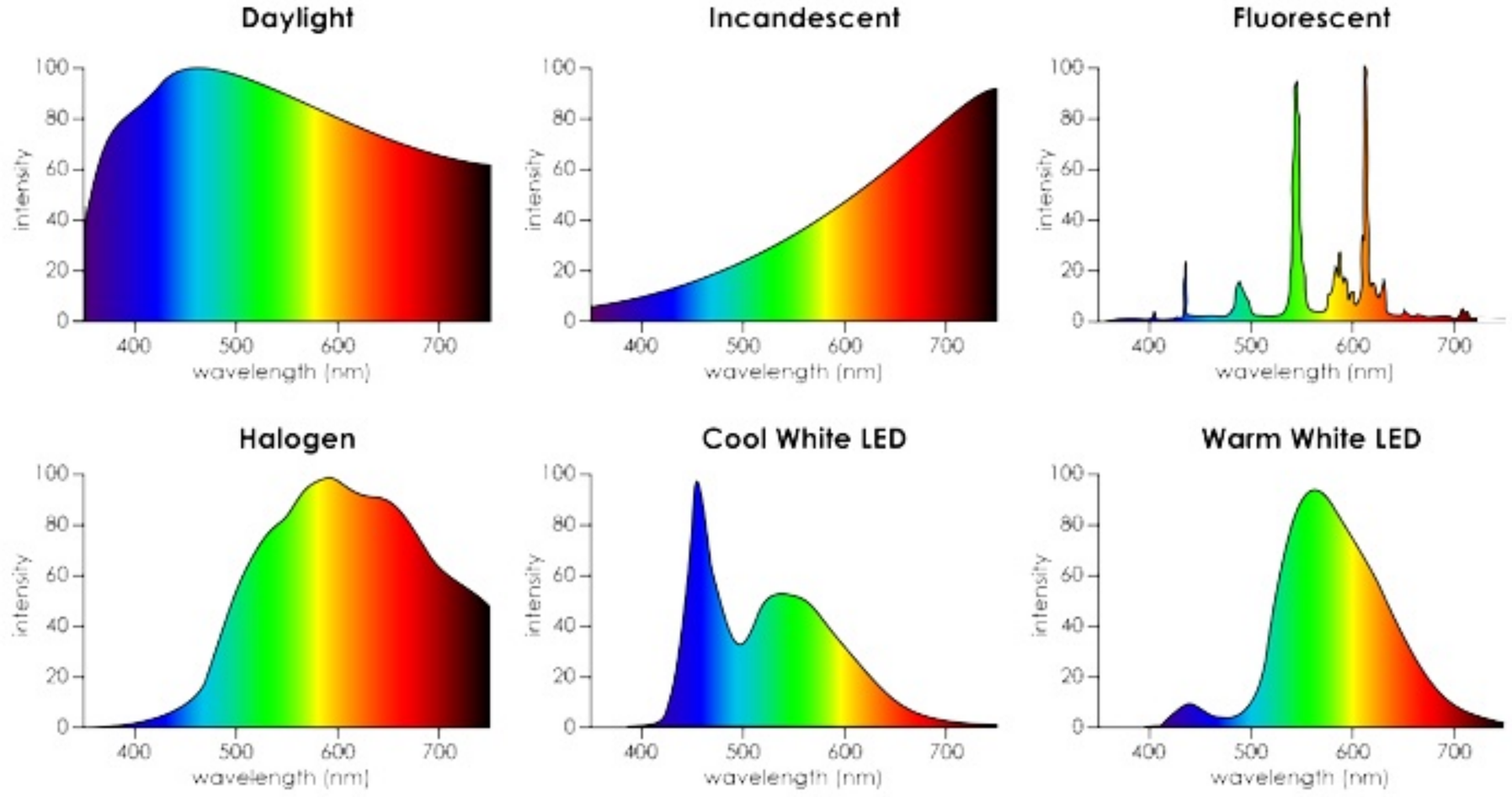


Figure credit:

admesy
ADVANCED MEASUREMENT SYSTEMS

Review: spectral response of cone cells in human eye

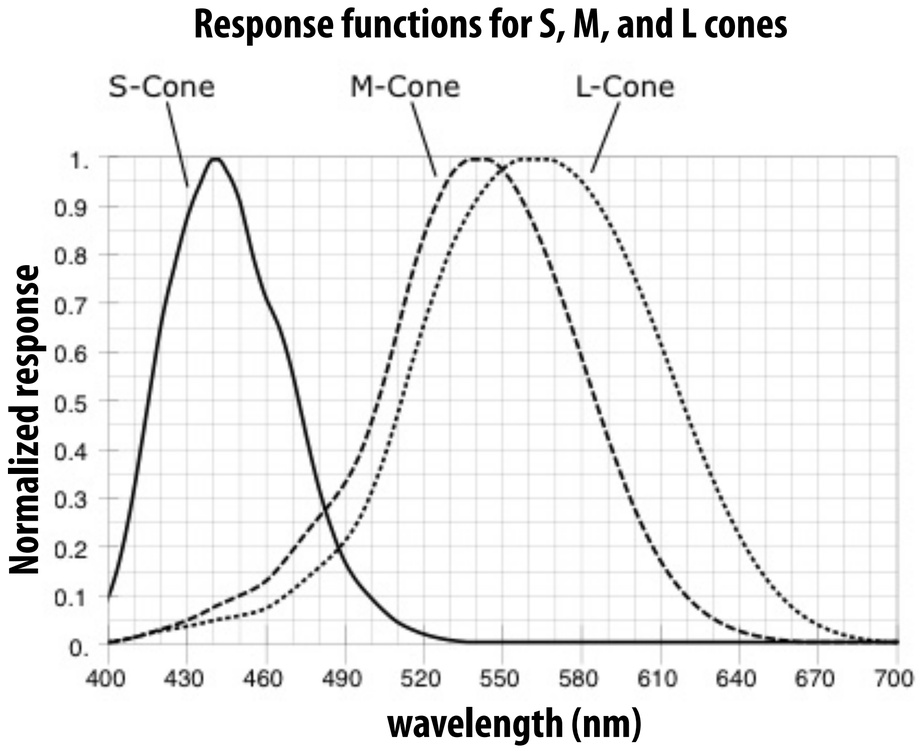
Three types of cells in eye responsible for color perception: S, M, and L cones (corresponding to peak response at short, medium, and long wavelengths)

Implication: the space of human-perceivable colors is three dimensional

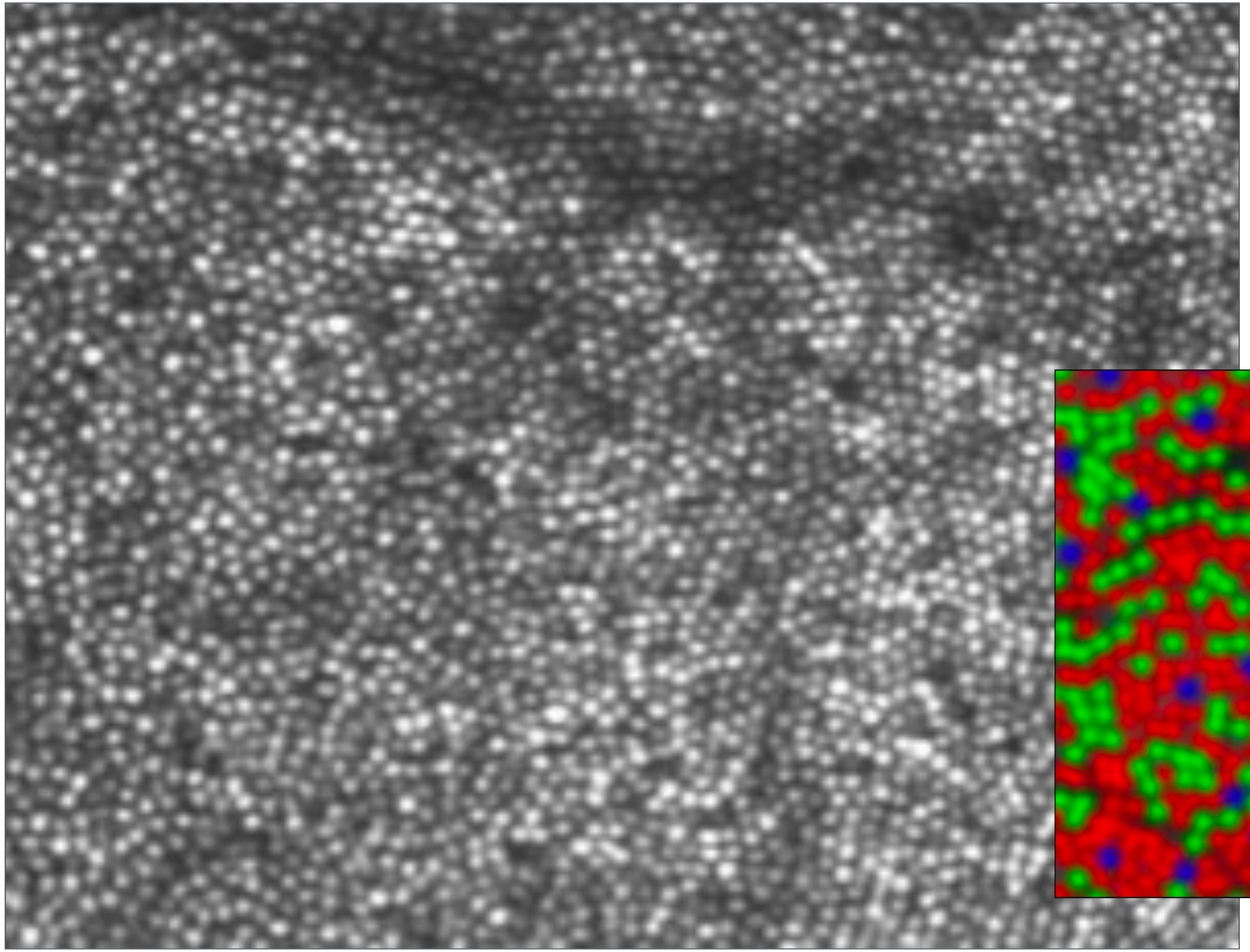
$$S = \int_\lambda \Phi(\lambda) S(\lambda) d\lambda$$

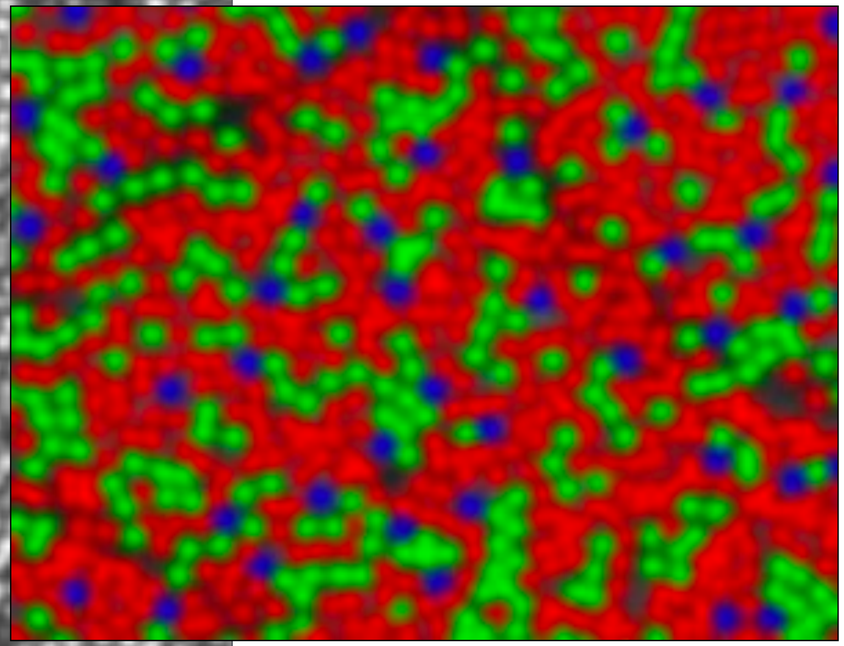
$$M = \int_\lambda \Phi(\lambda) M(\lambda) d\lambda$$

$$L = \int_\lambda \Phi(\lambda) L(\lambda) d\lambda$$



Human eye cone cell mosaic



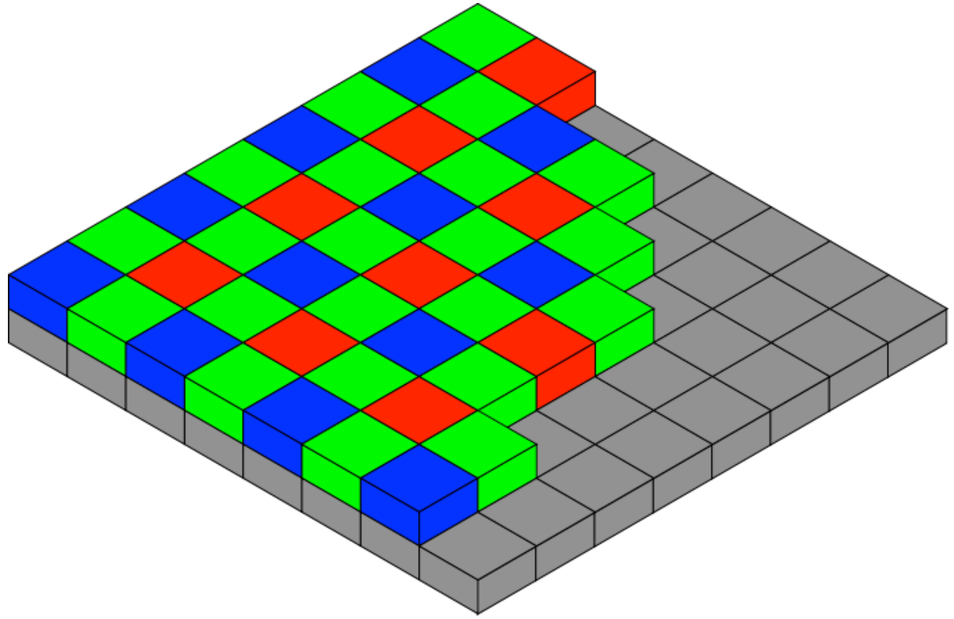


False color image:
red = L cones
green = M cones
blue = R cones

Image Credit: Ramkumar Sabesan Lab

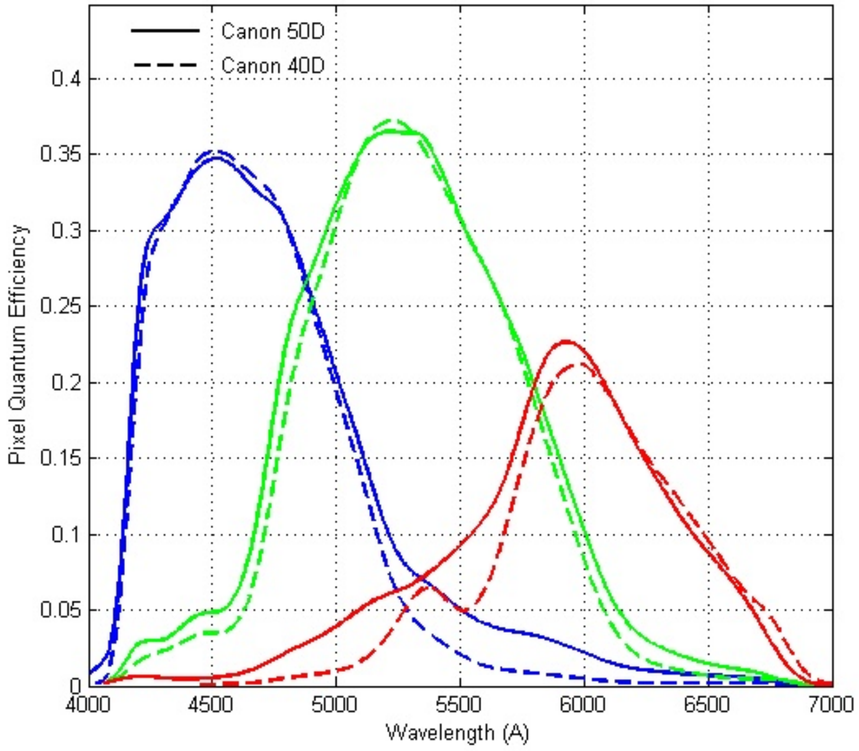
Color filter array (Bayer mosaic)

- 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)

Light incident on camera

What sensor measures

What sensor measures (zoomed view)

Demosiac'ing

- 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

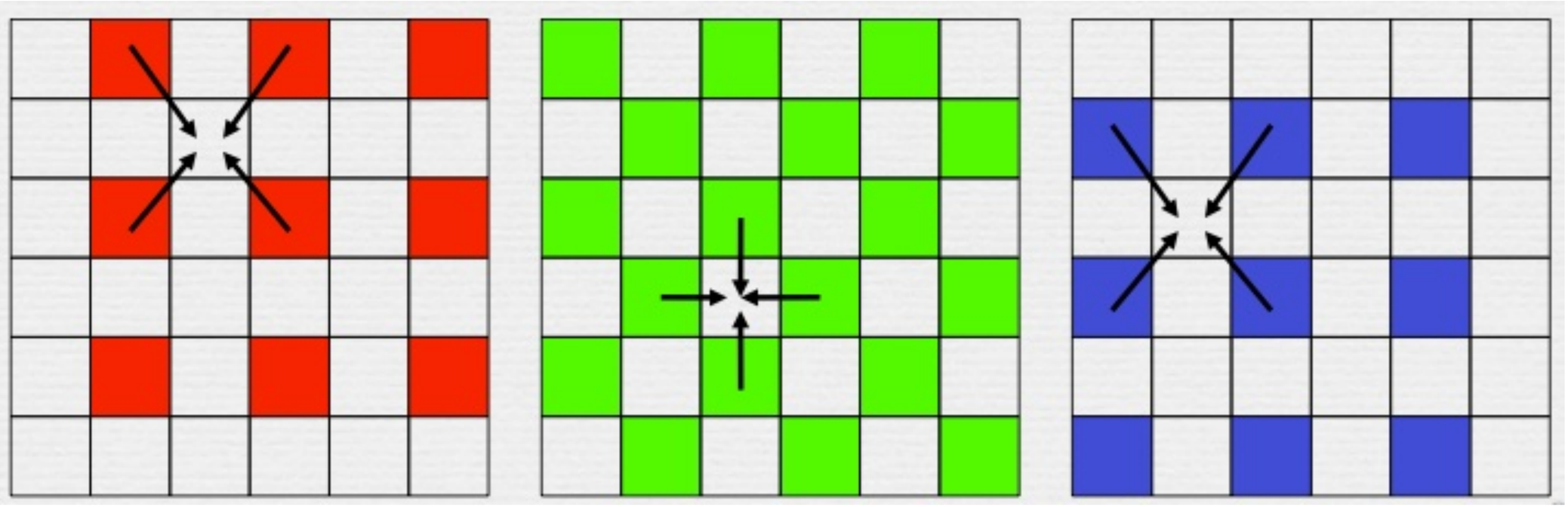
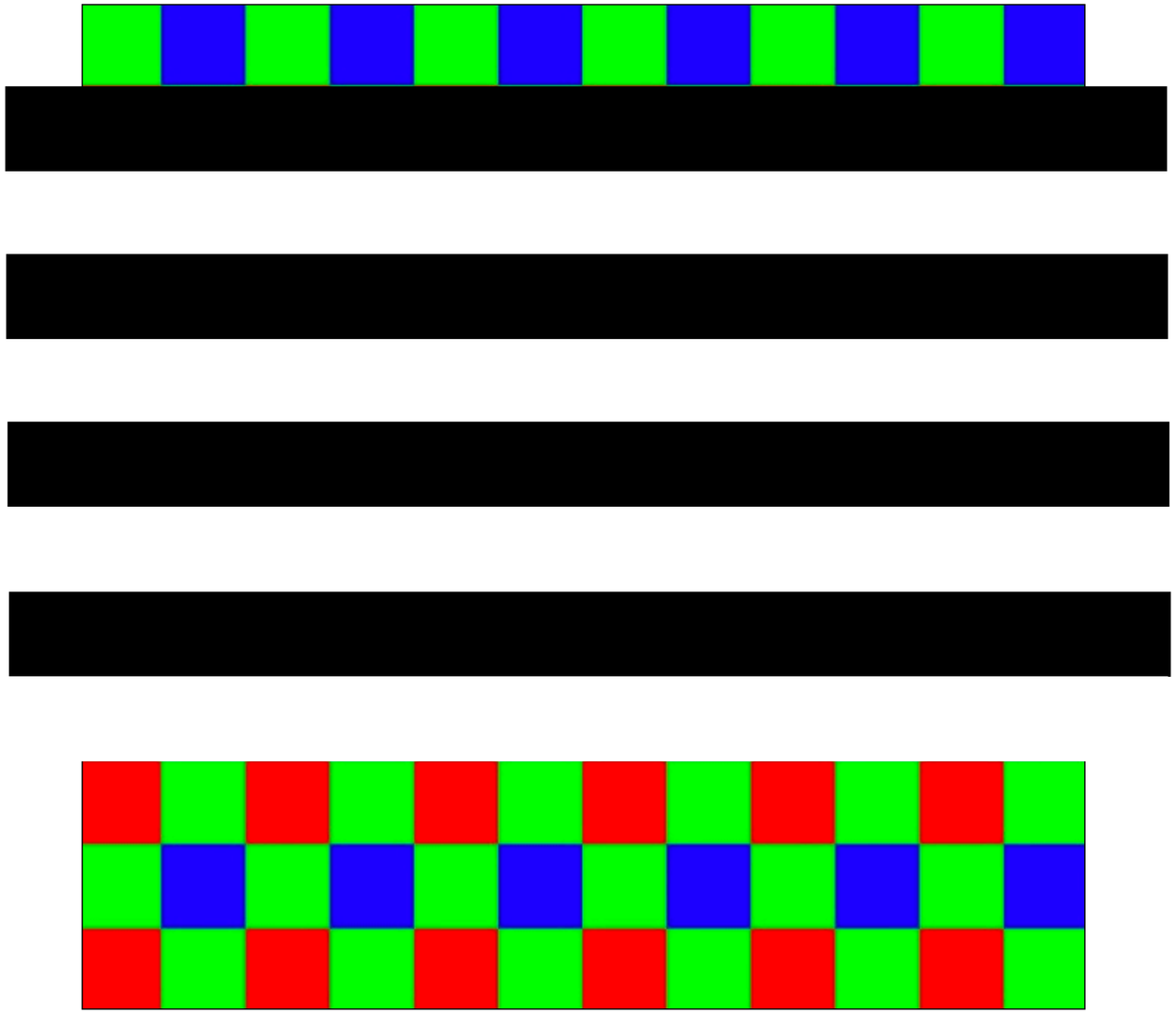


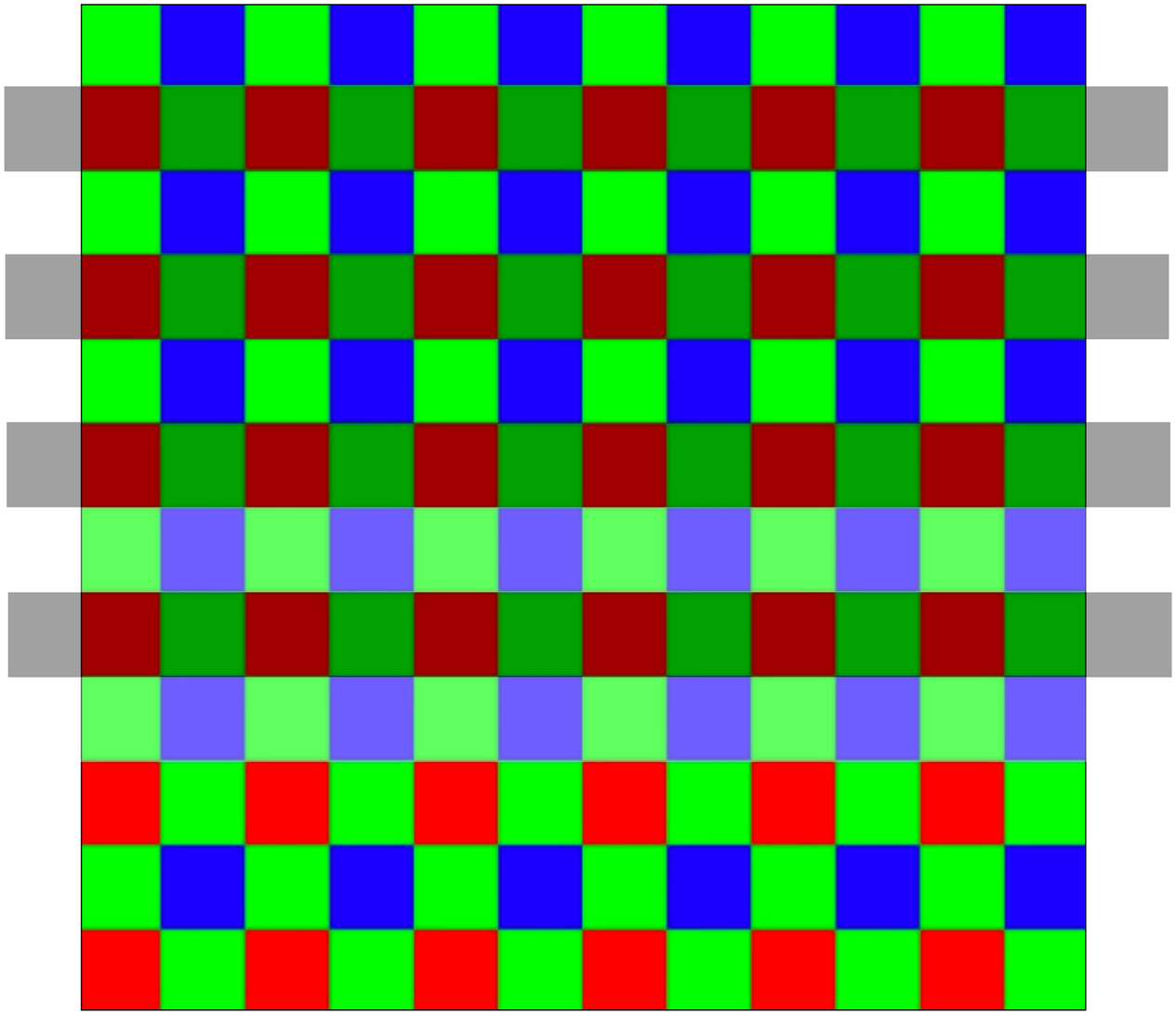
Image credit: Mark Levoy

Demosaicing errors



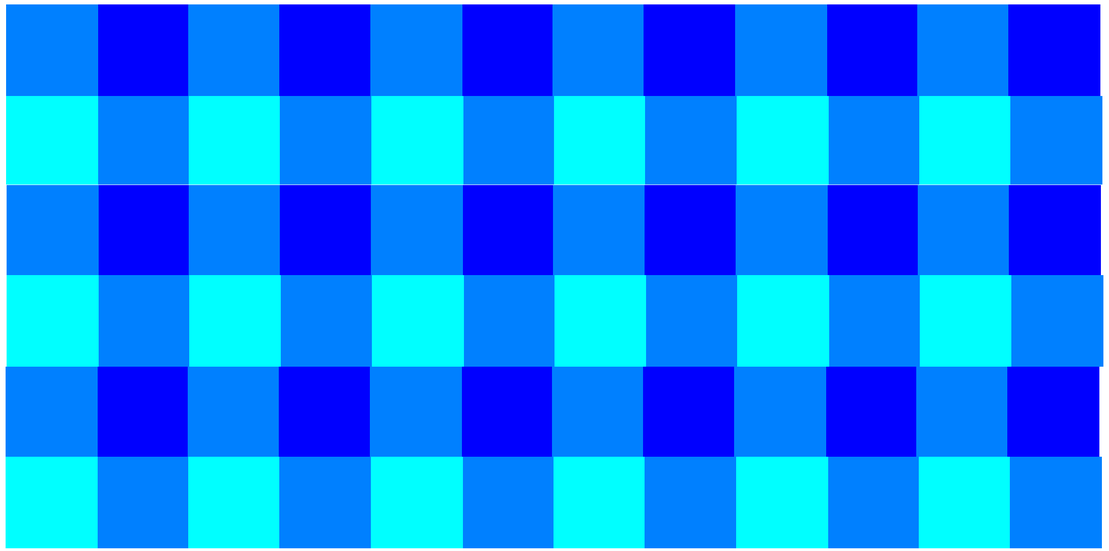
What will demosaiced result look like if this black and white signal was captured by the sensor?

Demosaicing errors



(Visualization of signal and Bayer pattern)

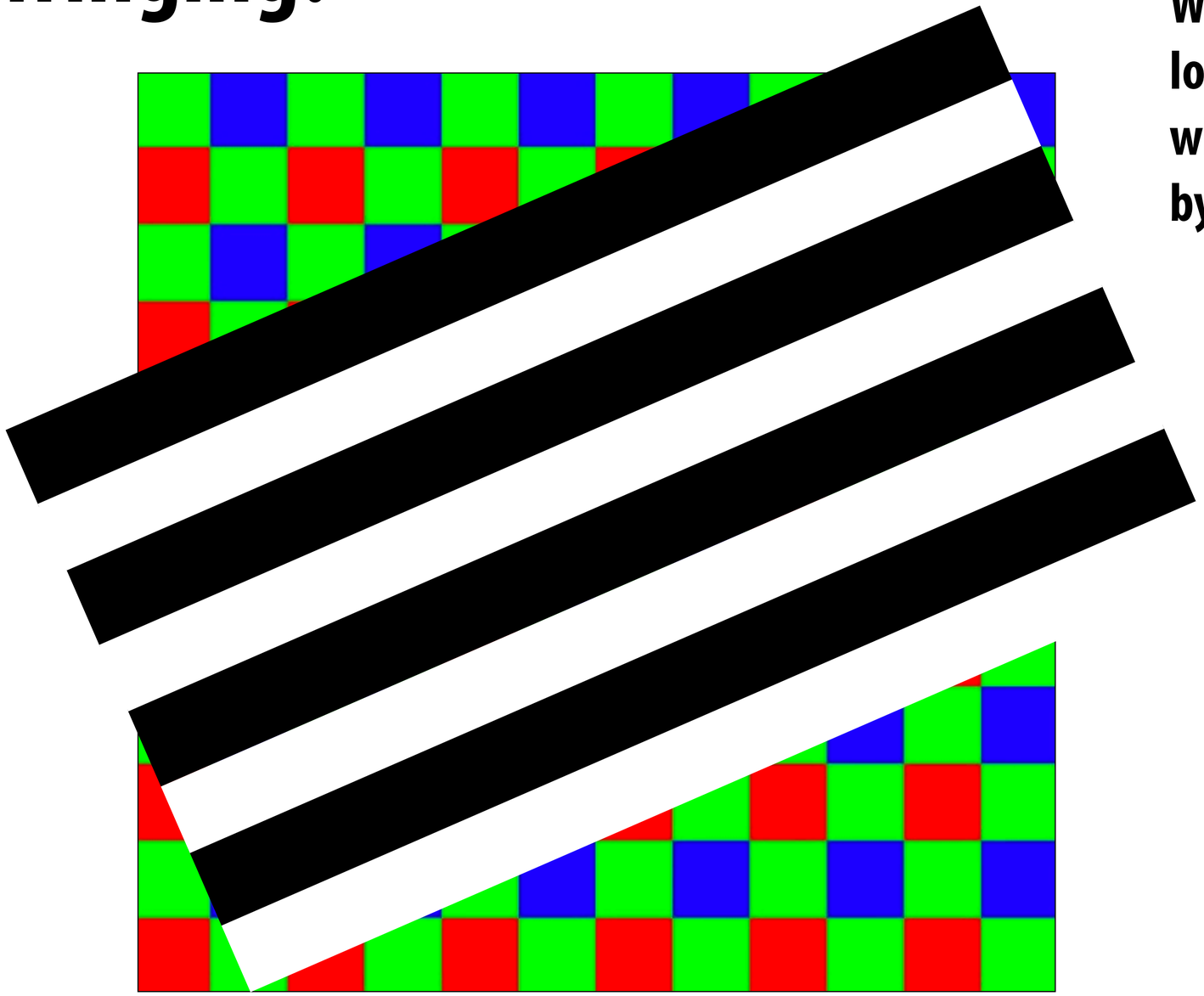
Demosaicing errors



No red measured.

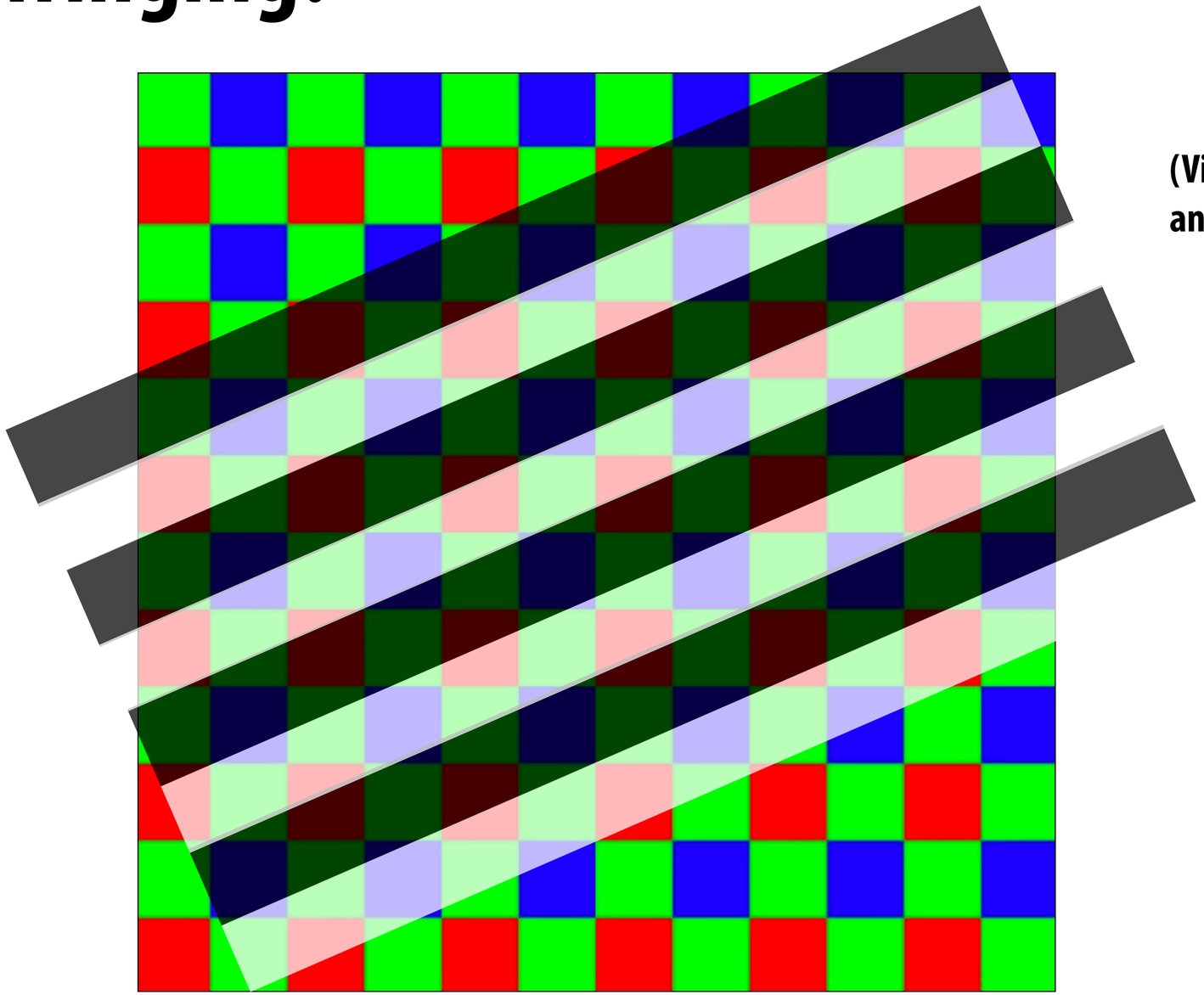
Interpolation of green yields dark/light pattern.

Why color fringing?



What will demosaiced result look like if this black and white signal was captured by the sensor?

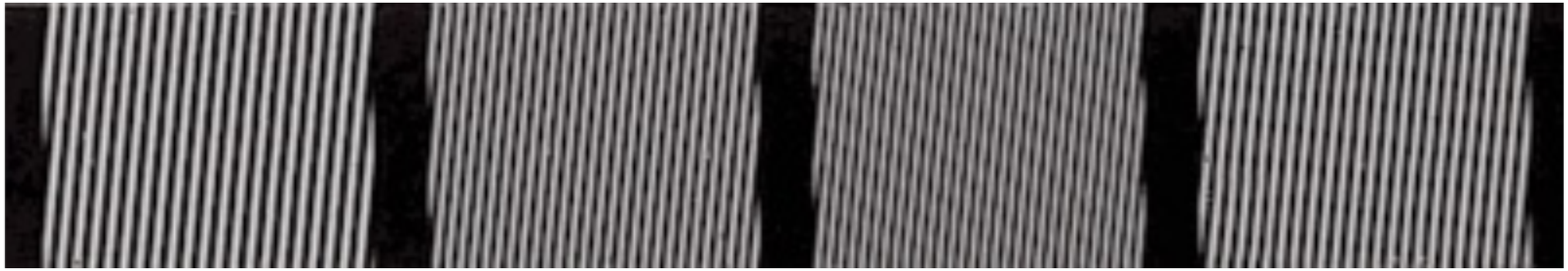
Why color fringing?



(Visualization of signal and Bayer pattern)

Demosaicing errors

- Common difficult case: fine diagonal black and white stripes
- Result: moire pattern color artifacts



RAW data
from sensor

RGB result after
demosaic

Image credit: http://almanazir.typepad.com/almanazir/2006/11/how_a_camera_ph_1.html

Better demosaic

- Convert demosaic'ed RGB value to YCbCr
- Low-pass filter (blur) or median filter CbCr channels
  - Remember from last class… human eye less sensitive to spatial variation in chromaticity
- Combine filtered CbCr with full resolution Y from sensor to get RGB

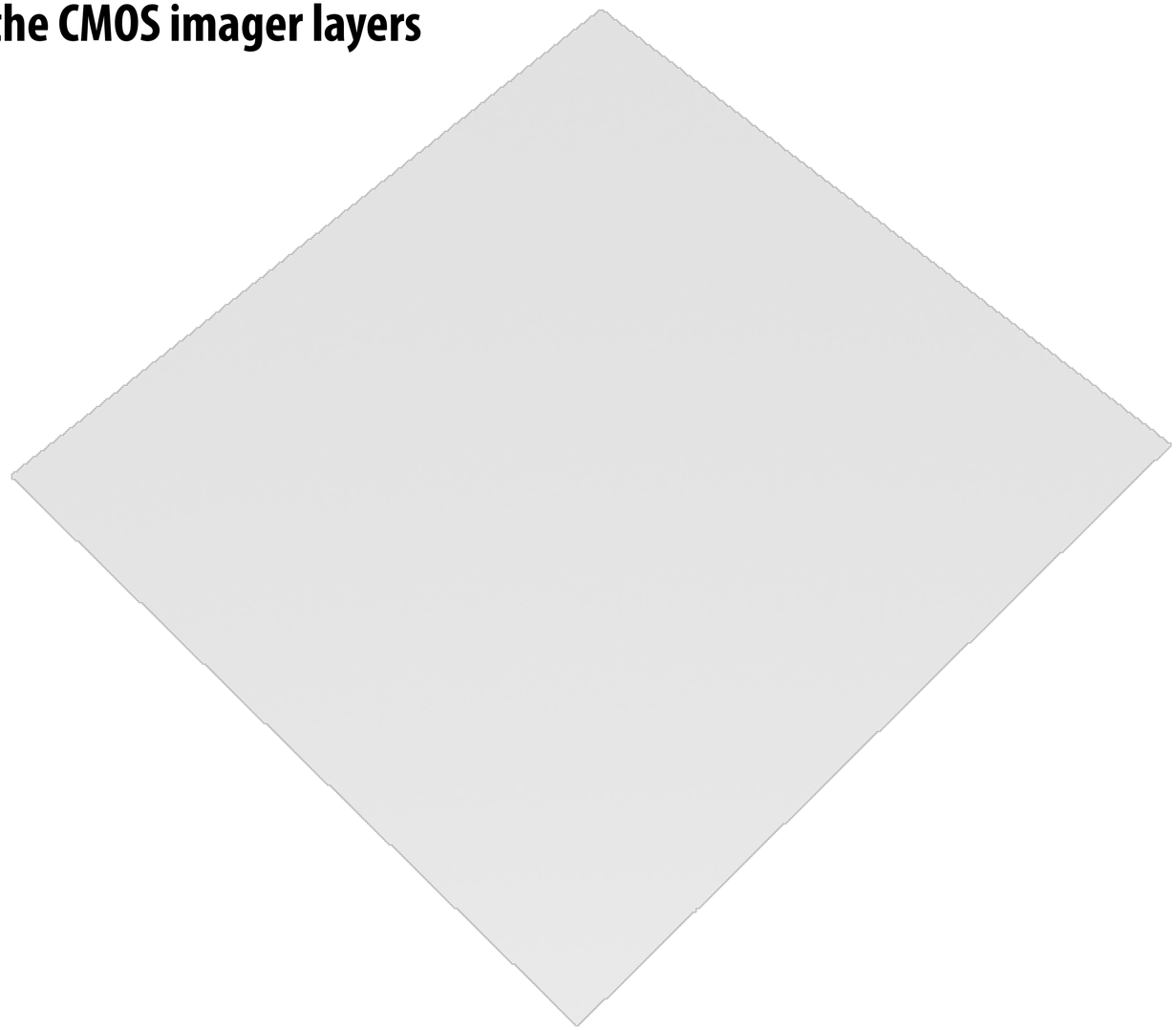
- Trades off spatial resolution of chroma information to avoid objectionable color fringing

- Modern solutions = learned demosaic'ing function (train DNN to take mosaic'ed image to demosaic'ed)

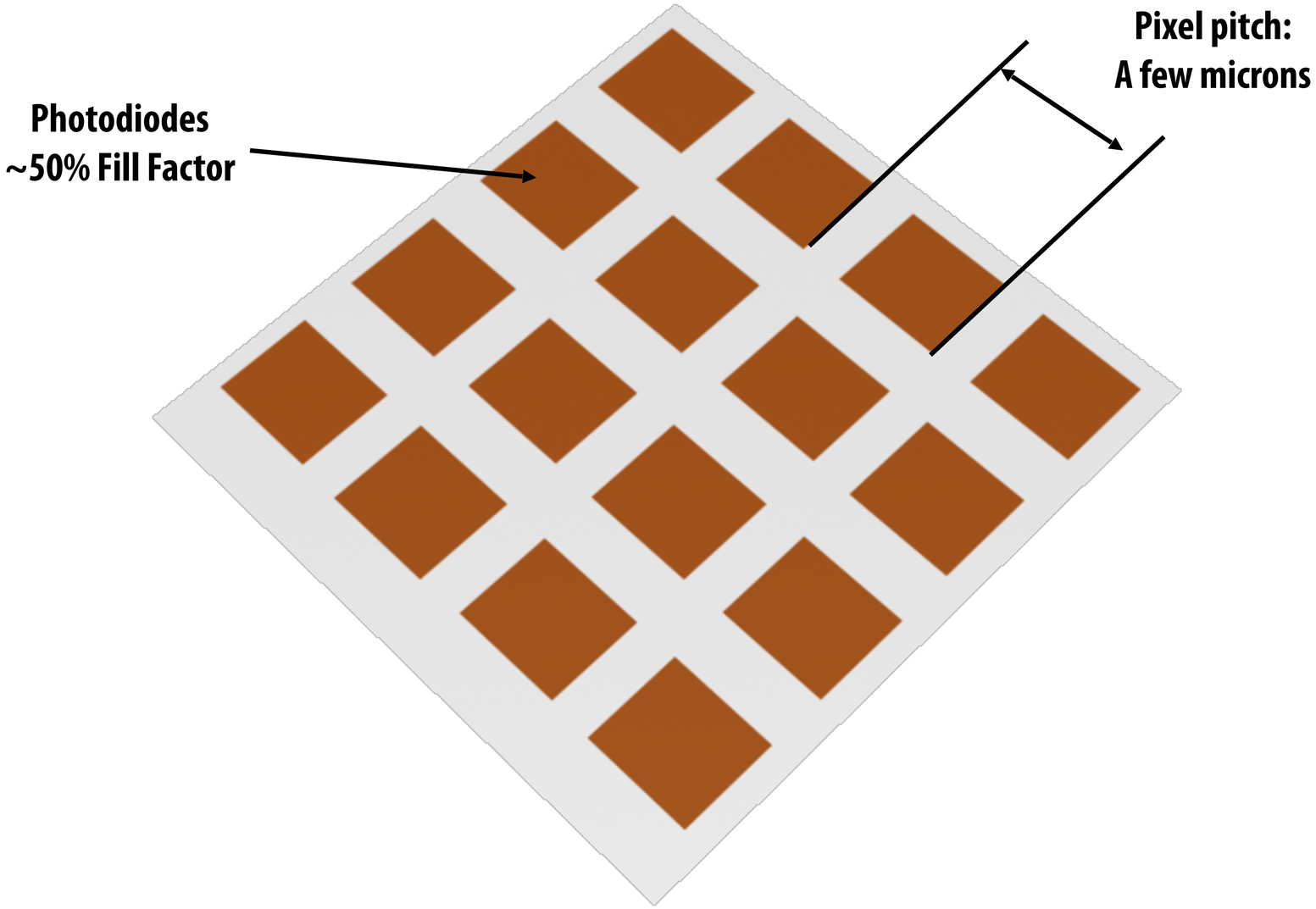
CMOS Pixel Structure

Front-side-illuminated (FSI) CMOS

Building up the CMOS imager layers

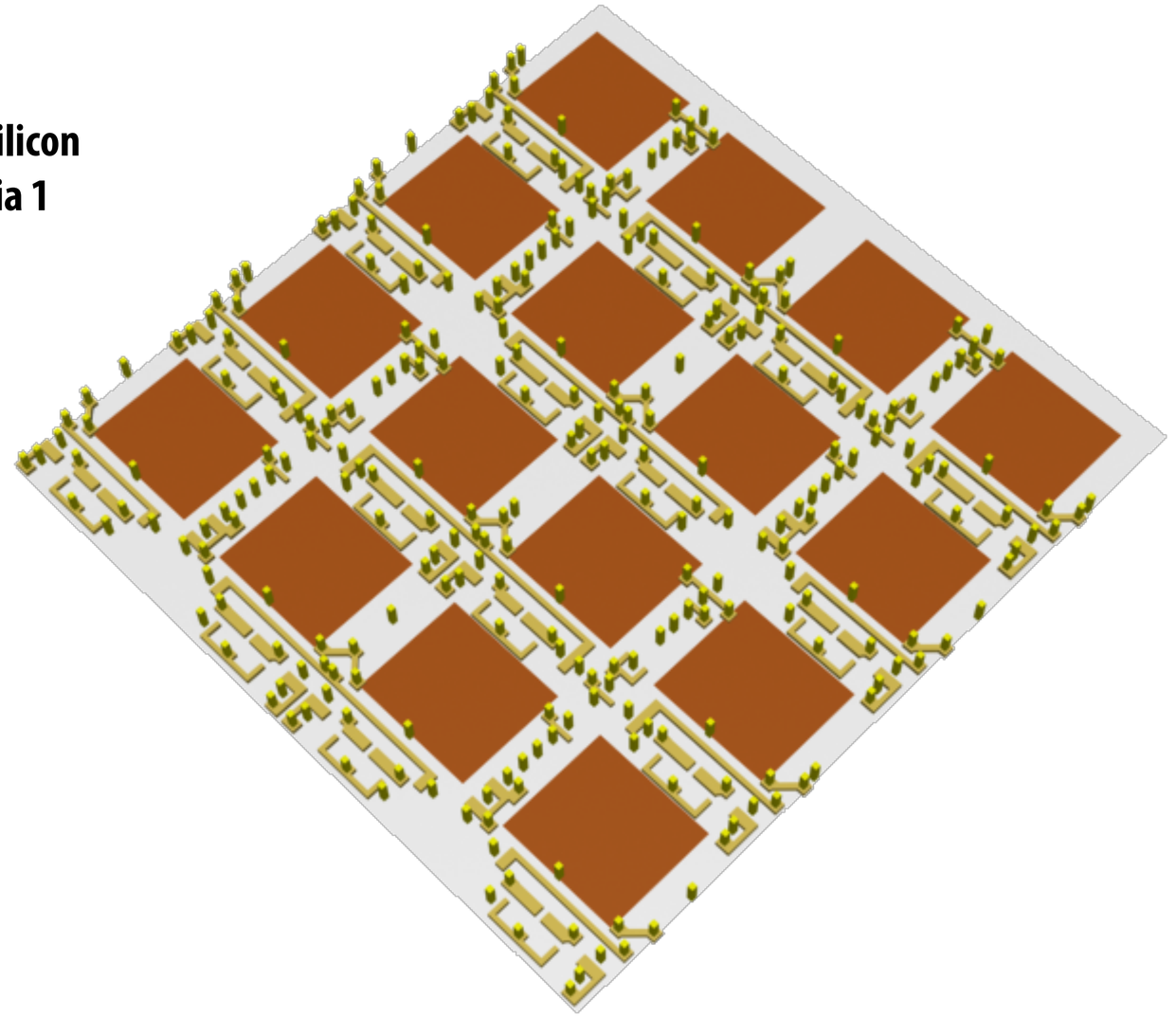


Courtesy R. Motta, Pixim



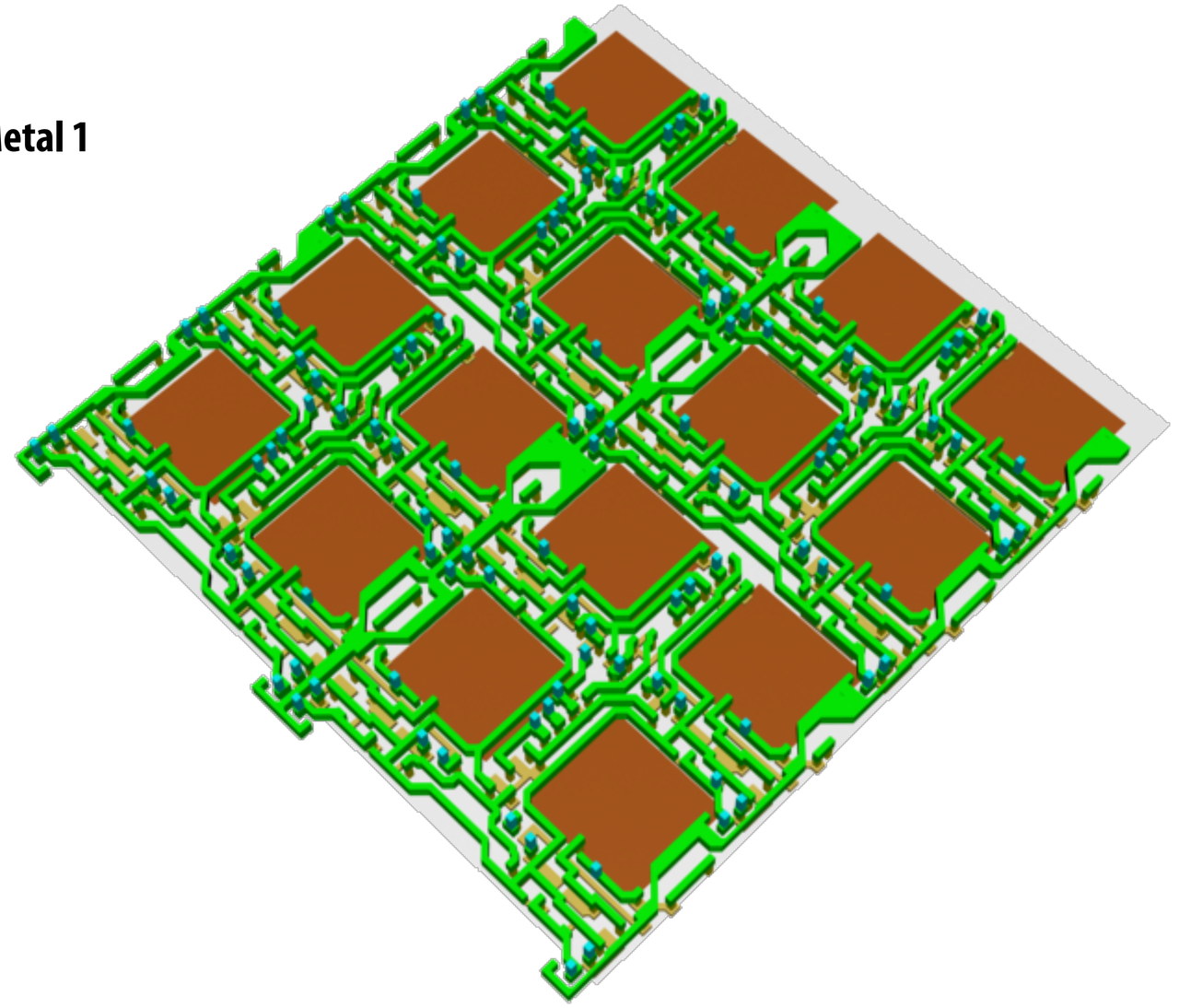
Courtesy R. Motta, Pixim

Polysilicon
& Via 1



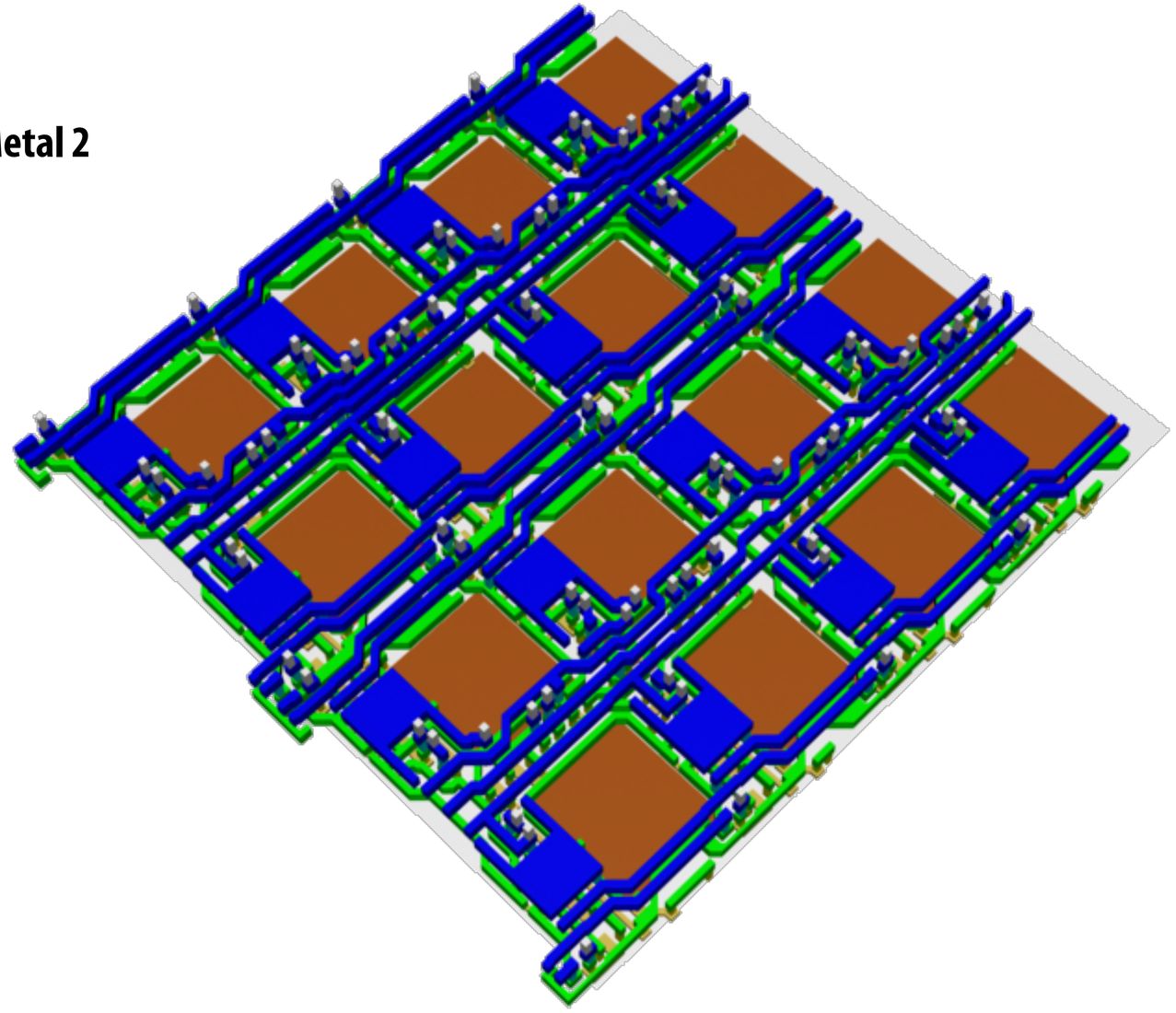
Courtesy R. Motta, Pixim

Metal 1



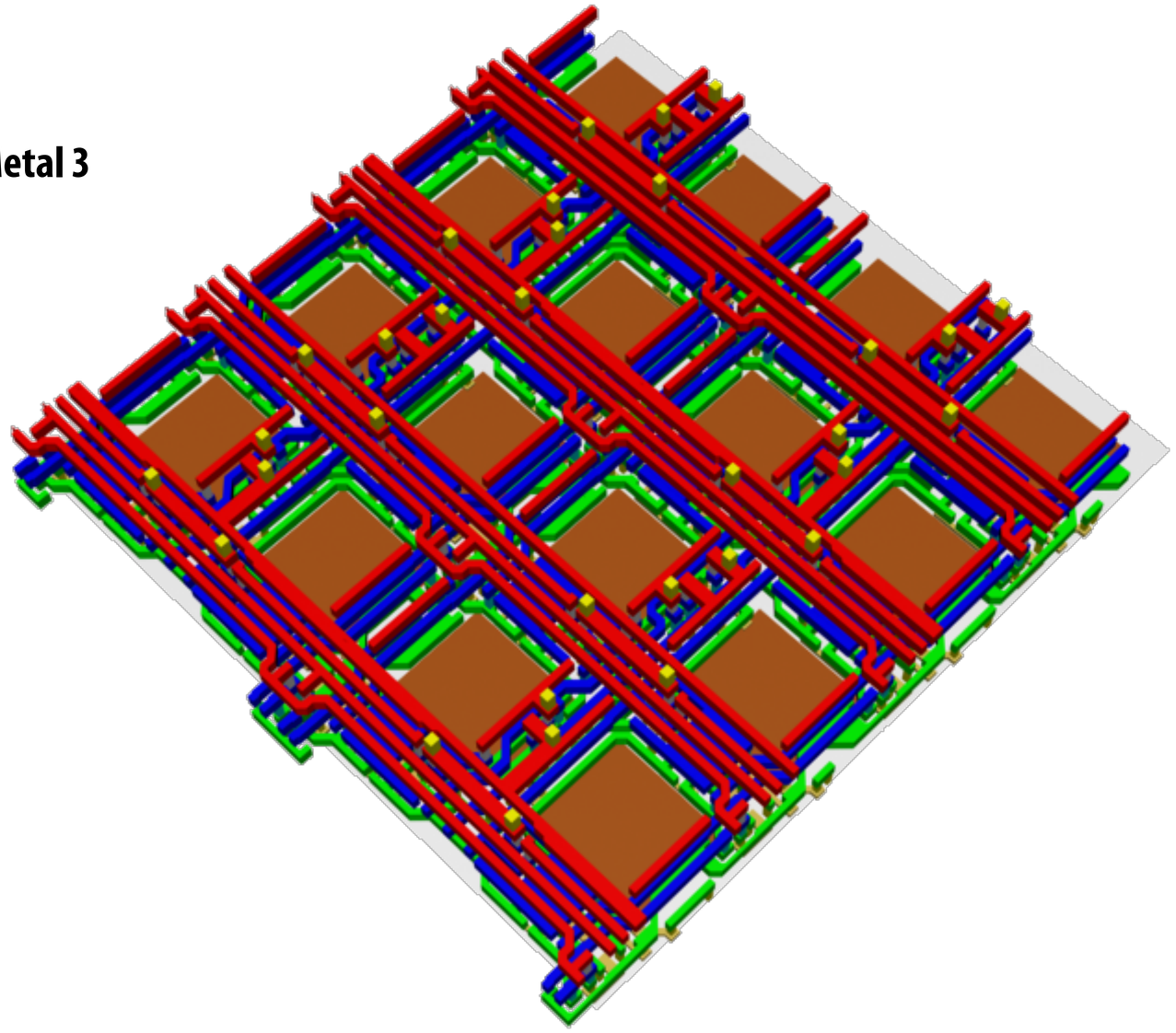
Courtesy R. Motta, Pixim

Metal 2



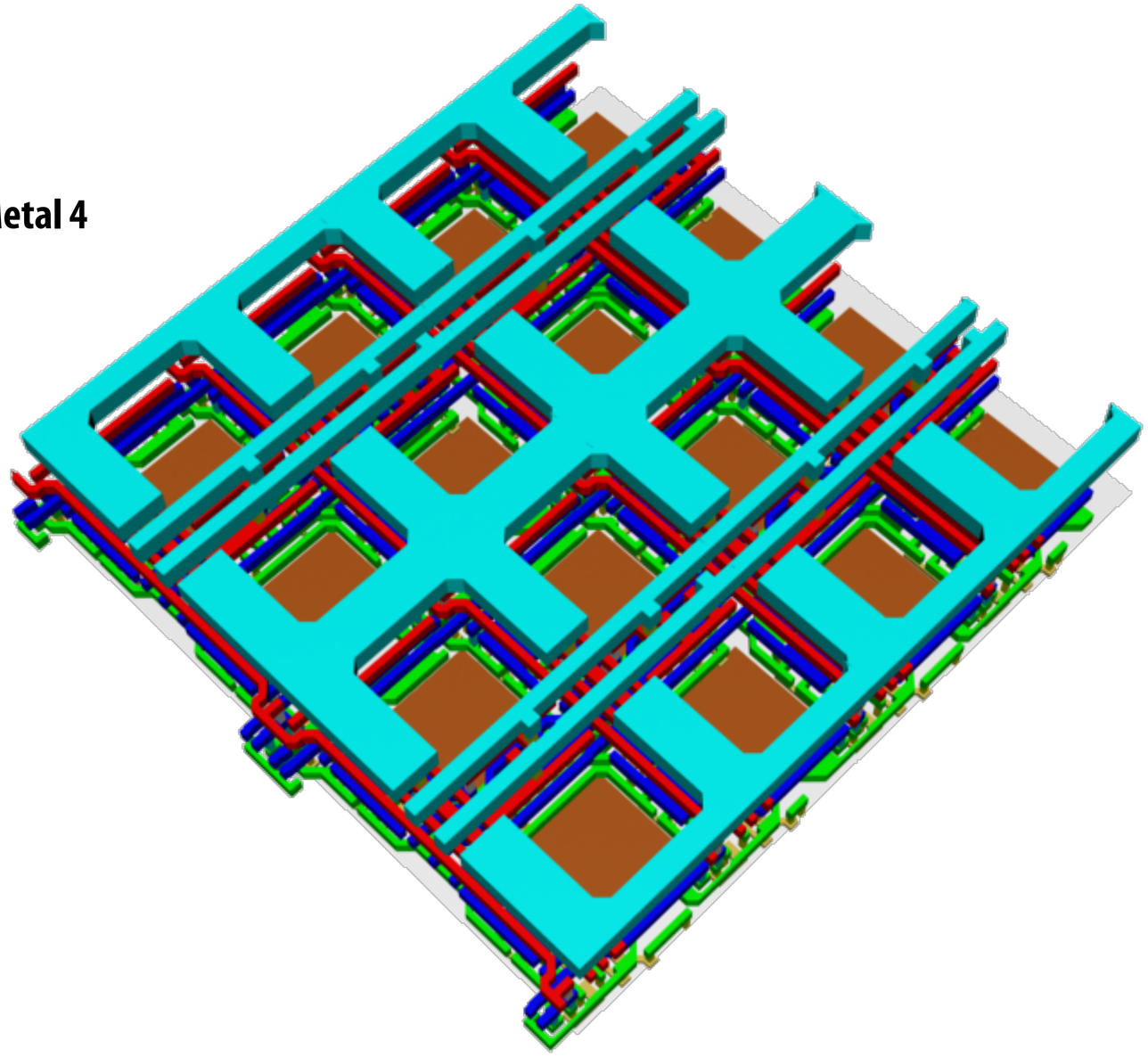
Courtesy R. Motta, Pixim

Metal 3



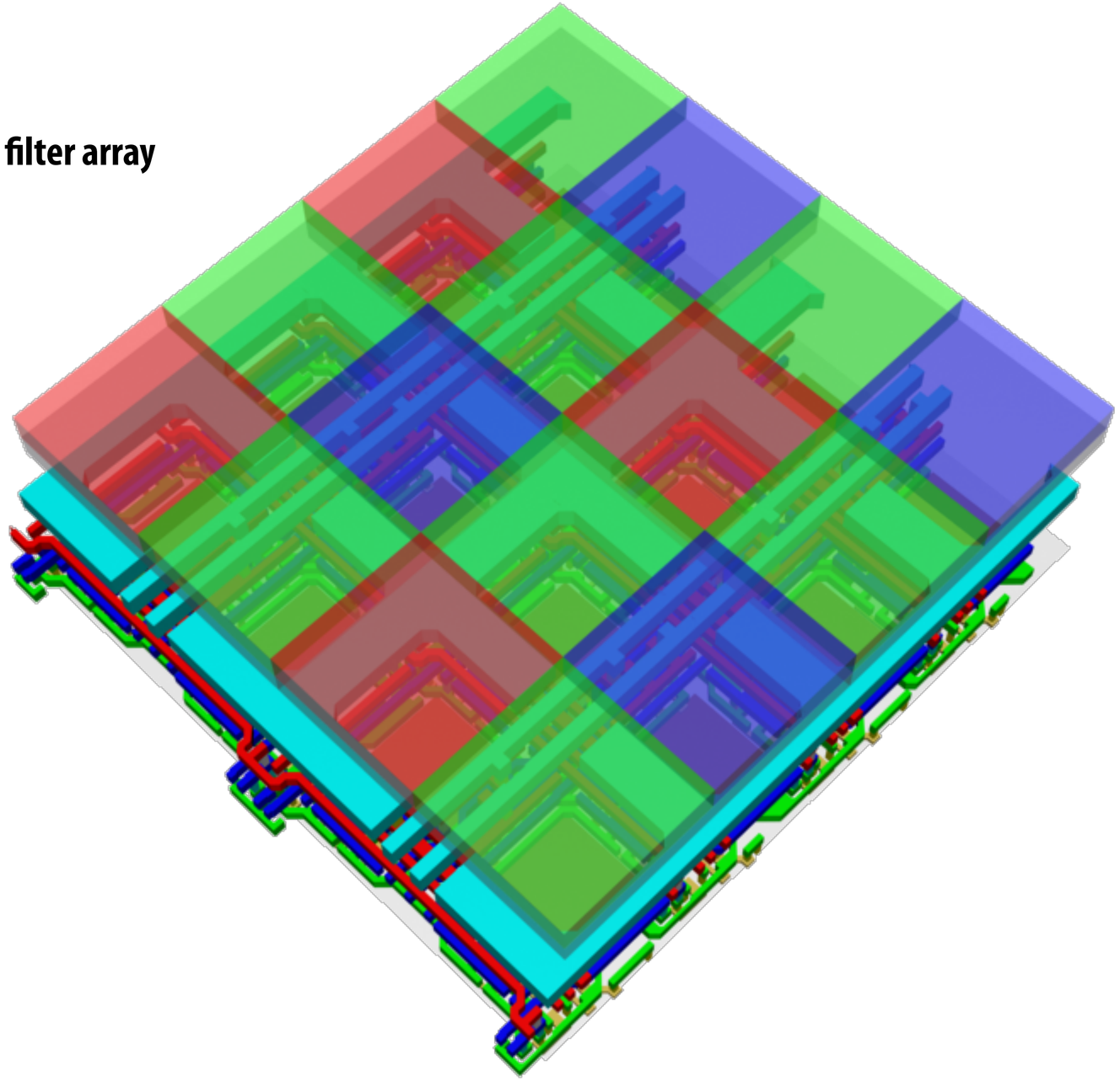
Courtesy R. Motta, Pixim

Metal 4



Courtesy R. Motta, Pixim

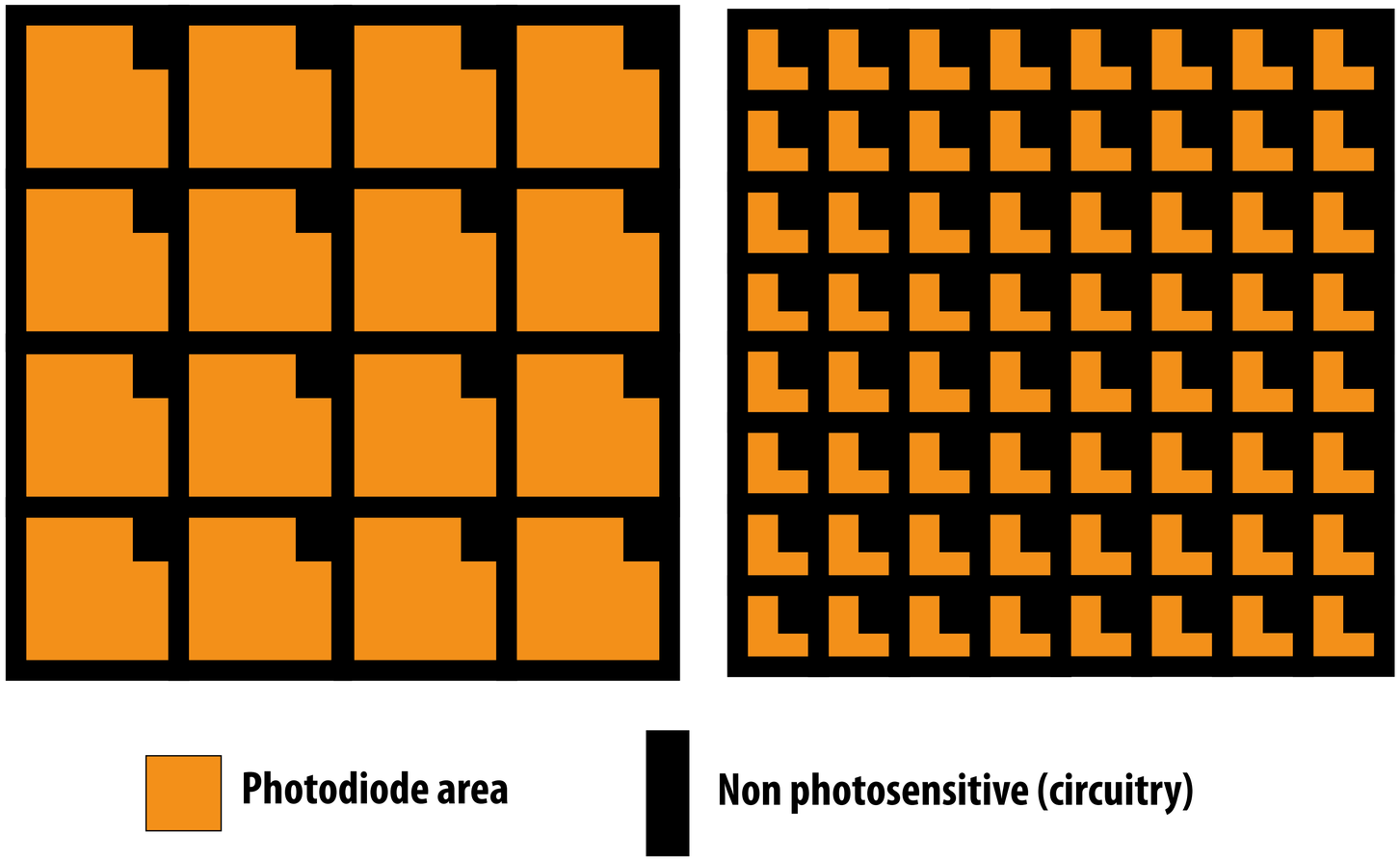
Color filter array



Courtesy R. Motta, Pixim

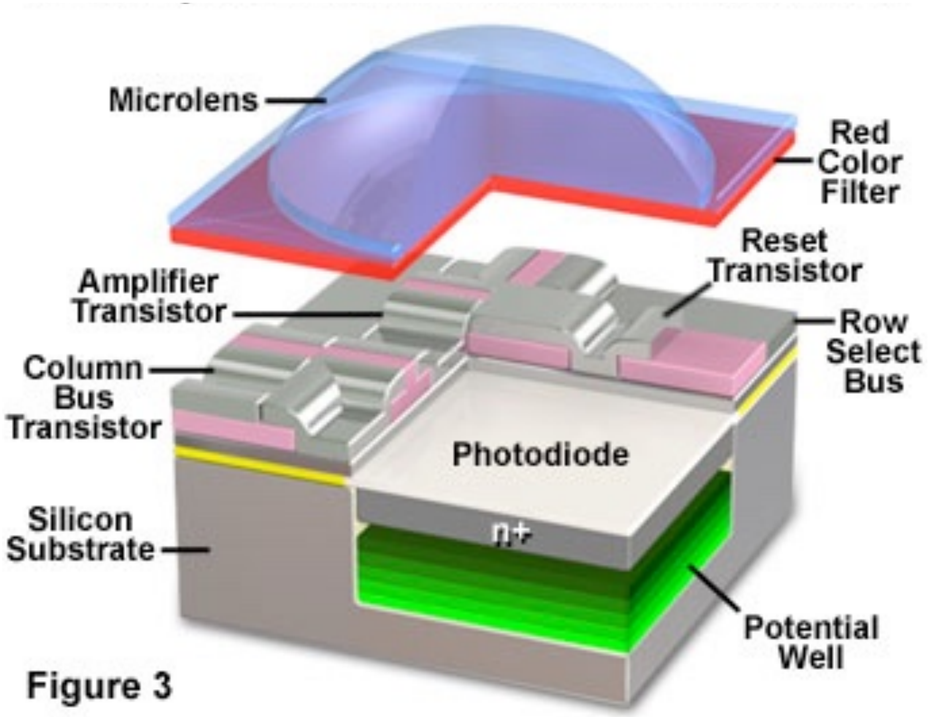
Pixel fill factor

Fraction of pixel area that integrates incoming light



Slide credit: Ren Ng

CMOS sensor pixel



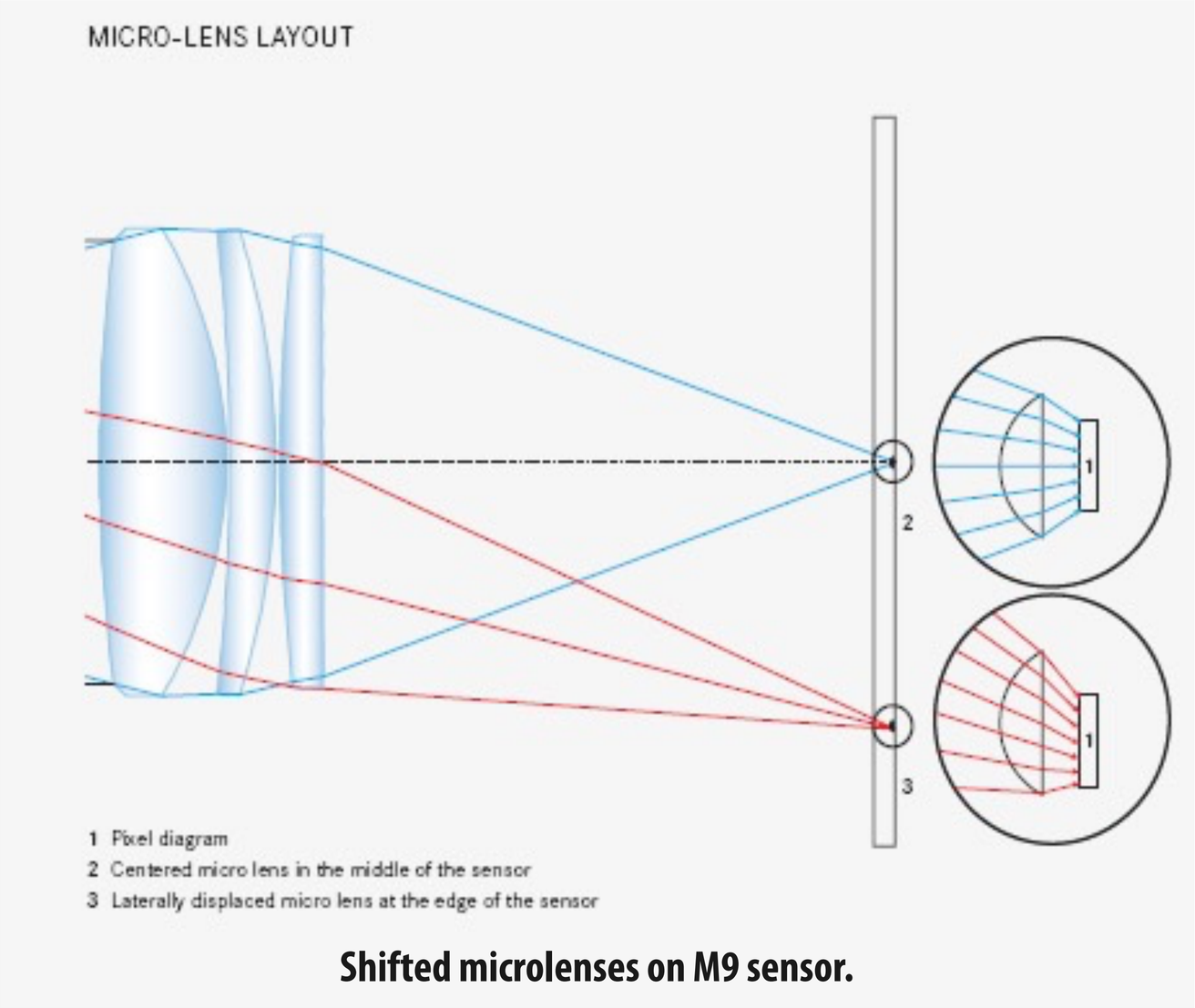
Color filter attenuates light

Microlens (a.k.a. lenslet) steers light toward photo-sensitive region (increases light-gathering capability)

Advanced question: Microlens also serves to reduce aliasing signal. Why?

Illustration credit: Molecular Expressions (http://micro.magnet.fsu.edu/primer/digitalimaging/cmosimagesensors.html)

Using micro lenses to improve fill factor



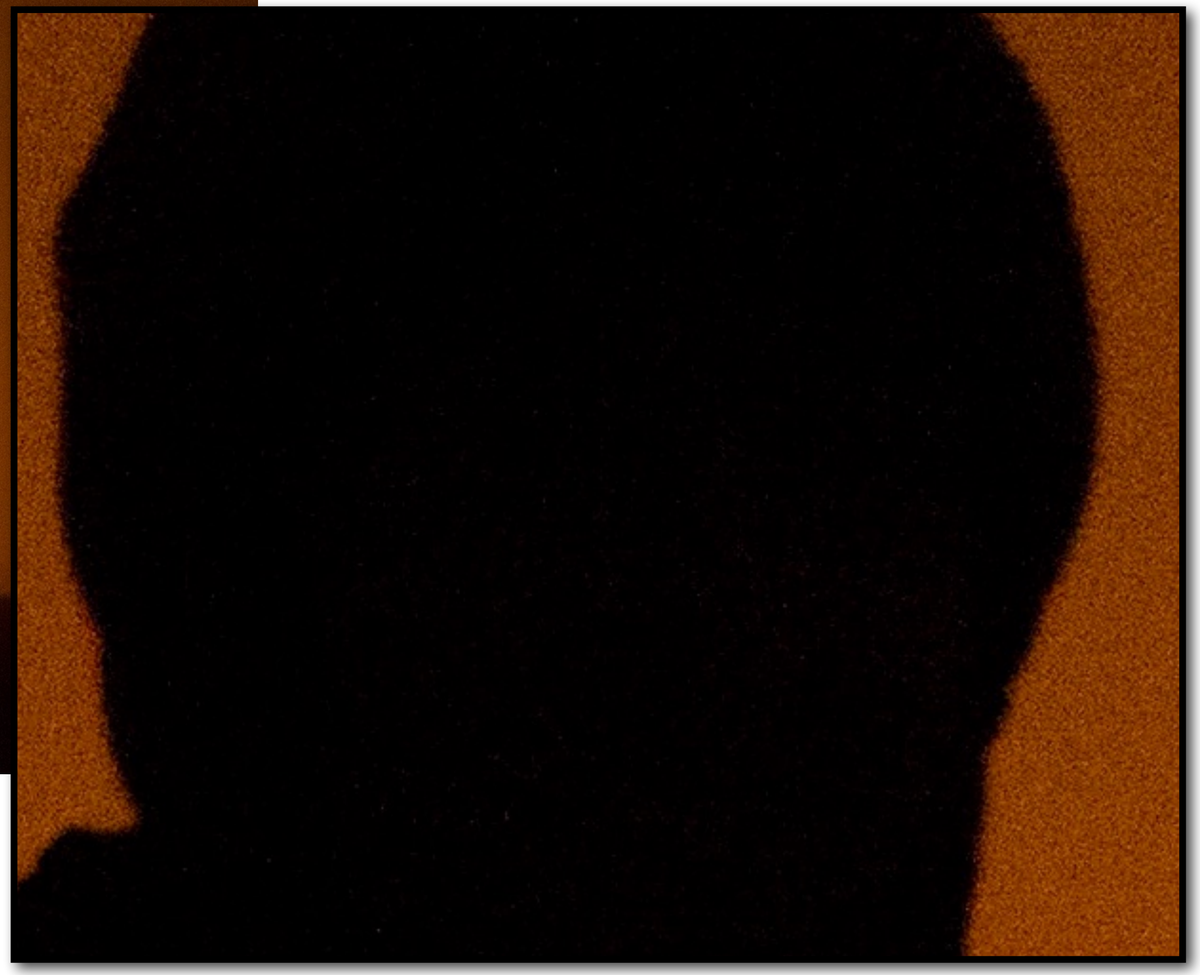
Shifted microlenses on M9 sensor.

Leica M9

Slide credit: Ren Ng

Noise

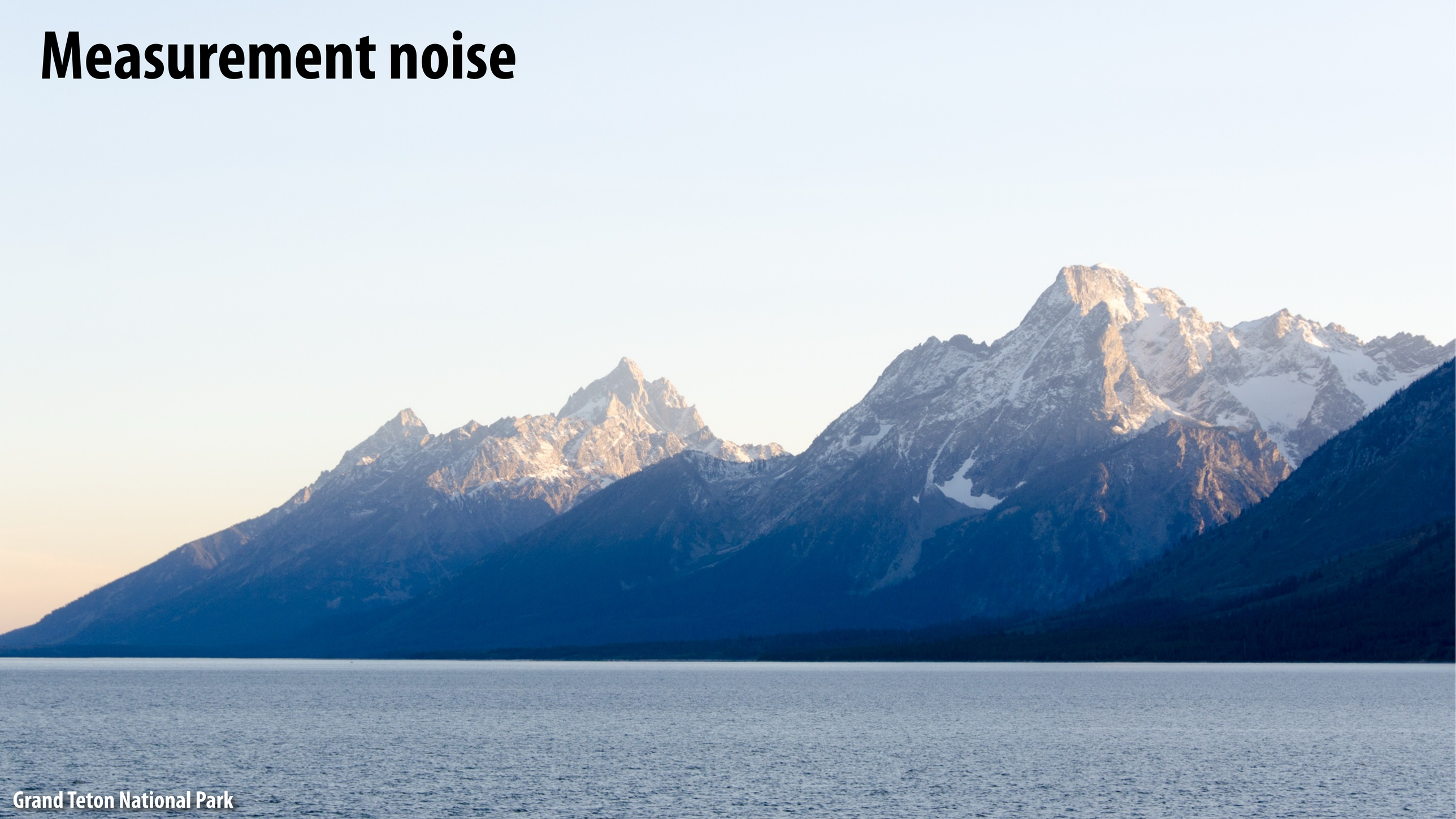
Measurement noise



We've all been frustrated by noise in low-light photographs

(or in shadows in day time images)

Measurement noise



Grand Teton National Park

Measurement noise

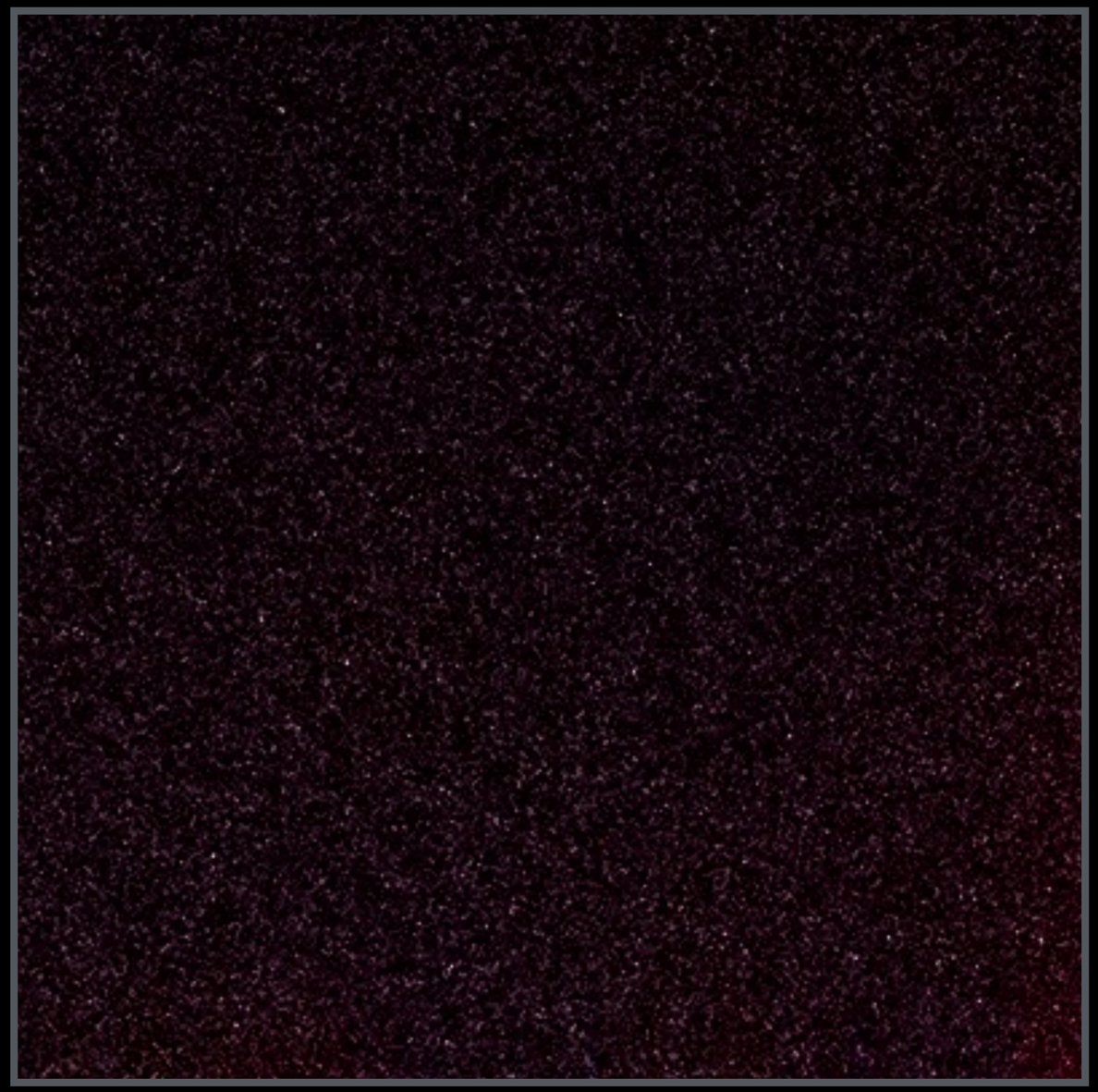
Grand Teton National Park

Sources of measurement noise

- Photon shot noise:
  - Photon arrival rate takes on Poisson distribution
  - Standard deviation = sqrt(N) (N = number of photon arrivals)
  - Signal-to-noise ratio (SNR) = N/sqrt(N)
  - Implication: brighter the signal, the higher the SNR
- Dark-shot noise
  - Due to leakage current in sensor
  - Electrons dislodged due to thermal activity (increases exponentially with sensor temperature)
- Non-uniformity of pixel sensitivity (due to manufacturing defects)
- Read noise
  - e.g., due to amplification / ADC

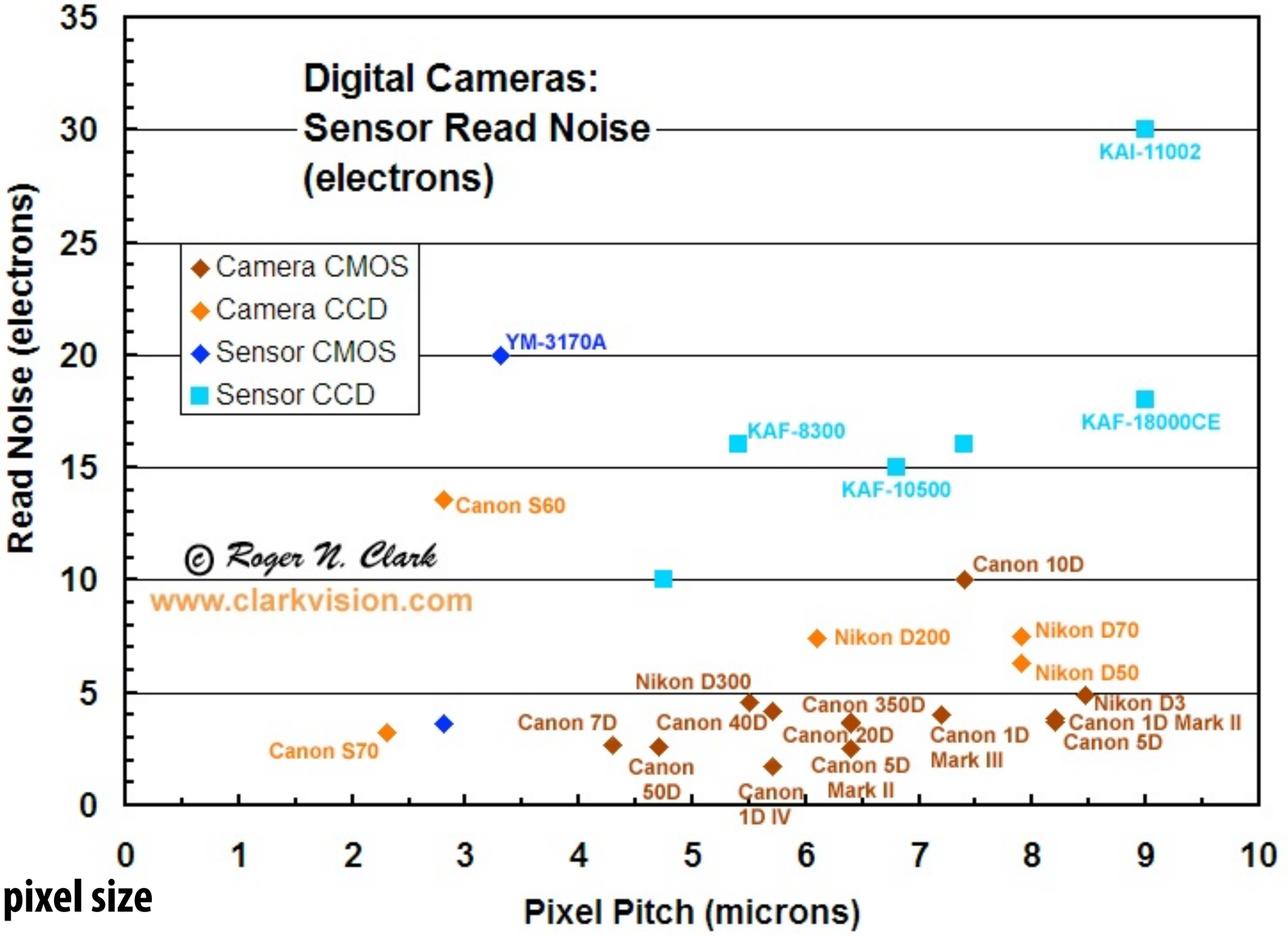
Dark shot noise / read noise

Black image examples: Nikon D7000, High ISO



1 sec exposure

Read noise



Read noise is largely independent of pixel size

Large pixels + bright scene = large N

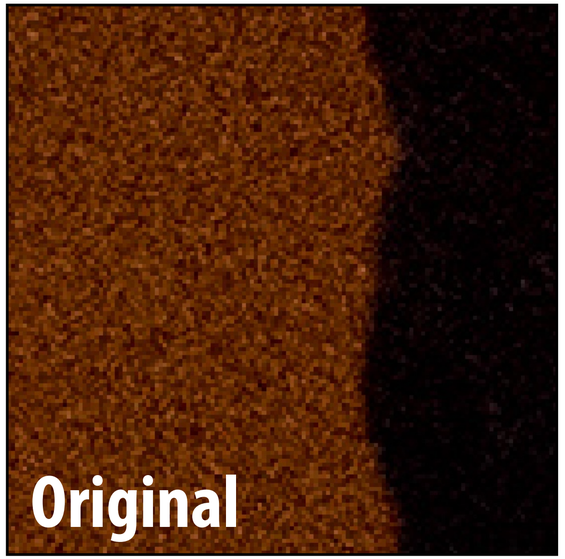
So, noise determined largely by photon shot noise

Image credit: clarkvision.com

Maximize light gathering capability

- **Goal: increase signal-to-noise ratio**
  - Dynamic range of a pixel (ratio of brightest light measurable to dimmest light measurable) is determined by the noise floor (minimum signal) and the pixel's full-well capacity (maximum signal)
- **Use big pixels**
  - Nikon D4: 7.3 um
  - iPhone X: 1.2 um
- **Manufacture sensitive pixels**
  - Good materials
  - High fill factor

Denoising



Low light conditions need long exposure…
blur due to camera shake

Image credit: https://www.colorexpertsbd.com/blog/how-to-fix-blurry-photos-induced-by-camera-shake-in-photoshop

Low light photo: many regions underexposed (short exposure) to avoid blur + some regions overexposed

Brightened image to see detail in dark regions, notice noise in dark regions

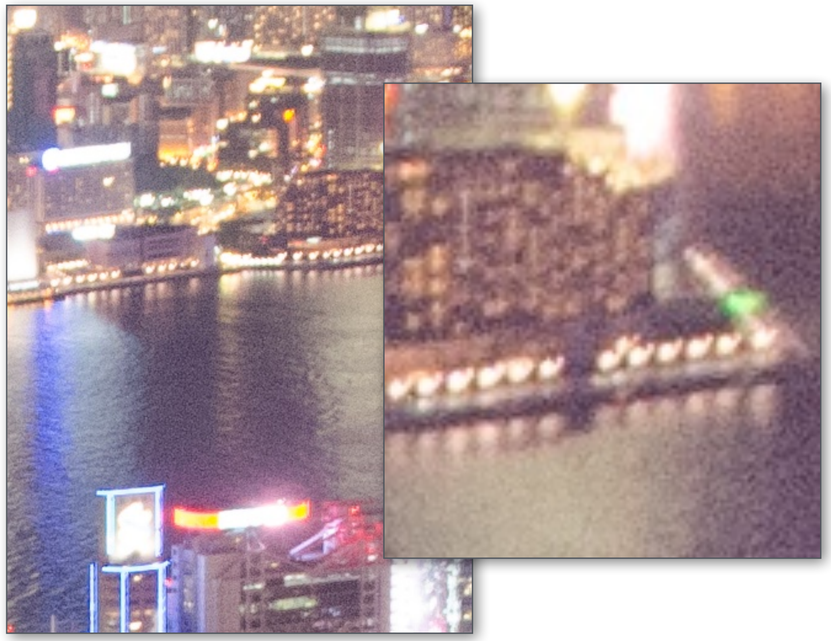
Attempt to denoise… splotchy effect remains

Long exposure: walking people are blurred…

Long exposure: walking people are blurred…

Also: still significant noise in dark regions

Reduce noise via image processing: denoising via downsampling



Downsample via point sampling

(noise remains)

Downsample via averaging

Noise reduced

Like a smaller number of bigger pixels!

Averaging = discrete 2D convolution

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

output image
(the result of convolving f with input image I)

filter

input image

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

Then:

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

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

$$f(i,j) = \mathbf{F}_{i,j}$$

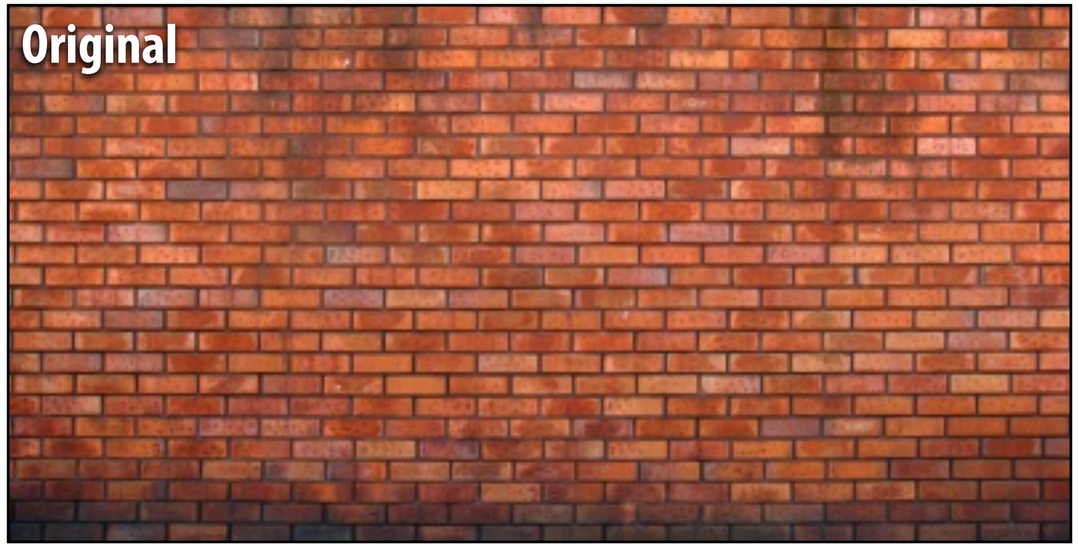
(often called: "filter weights", "filter kernel")

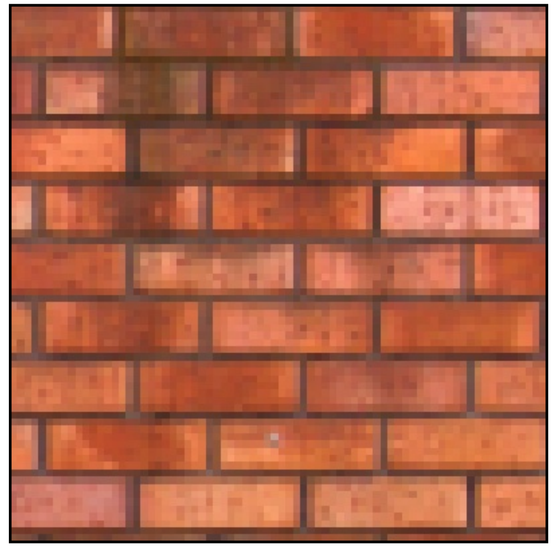
Simple 3x3 box blur in C code

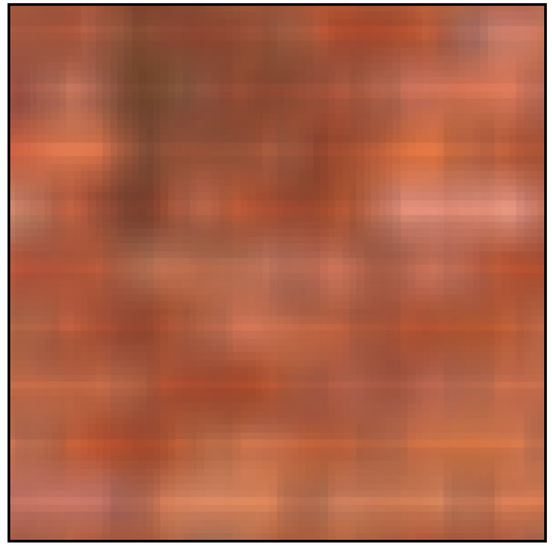
```
float input[(WIDTH+2) * (HEIGHT+2)];
float output[WIDTH * HEIGHT];

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 from sampling 2D Gaussian

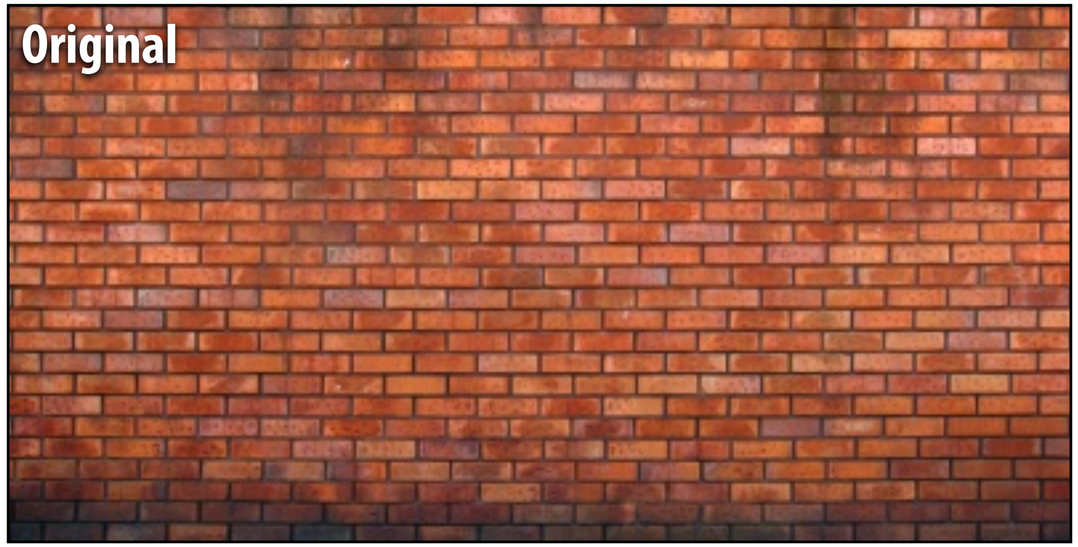
$$f(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$

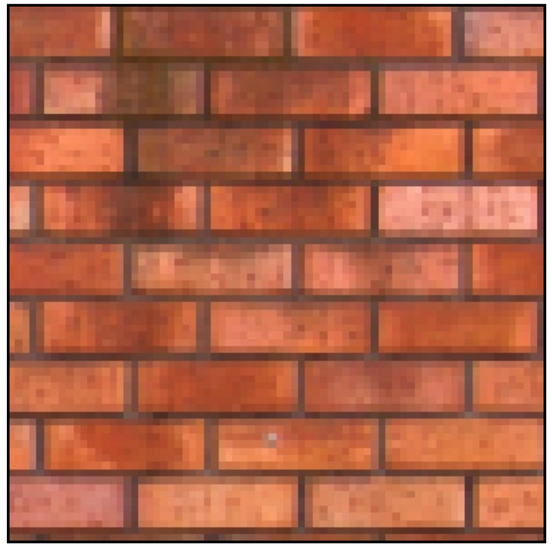
- Produces weighted sum of neighboring pixels (contribution falls off with distance)
  - In practice: truncate filter beyond certain distance for efficiency

$$\frac{1}{256} \cdot \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Note: this is a 5x5 truncated Gaussian filter

7x7 gaussian blur

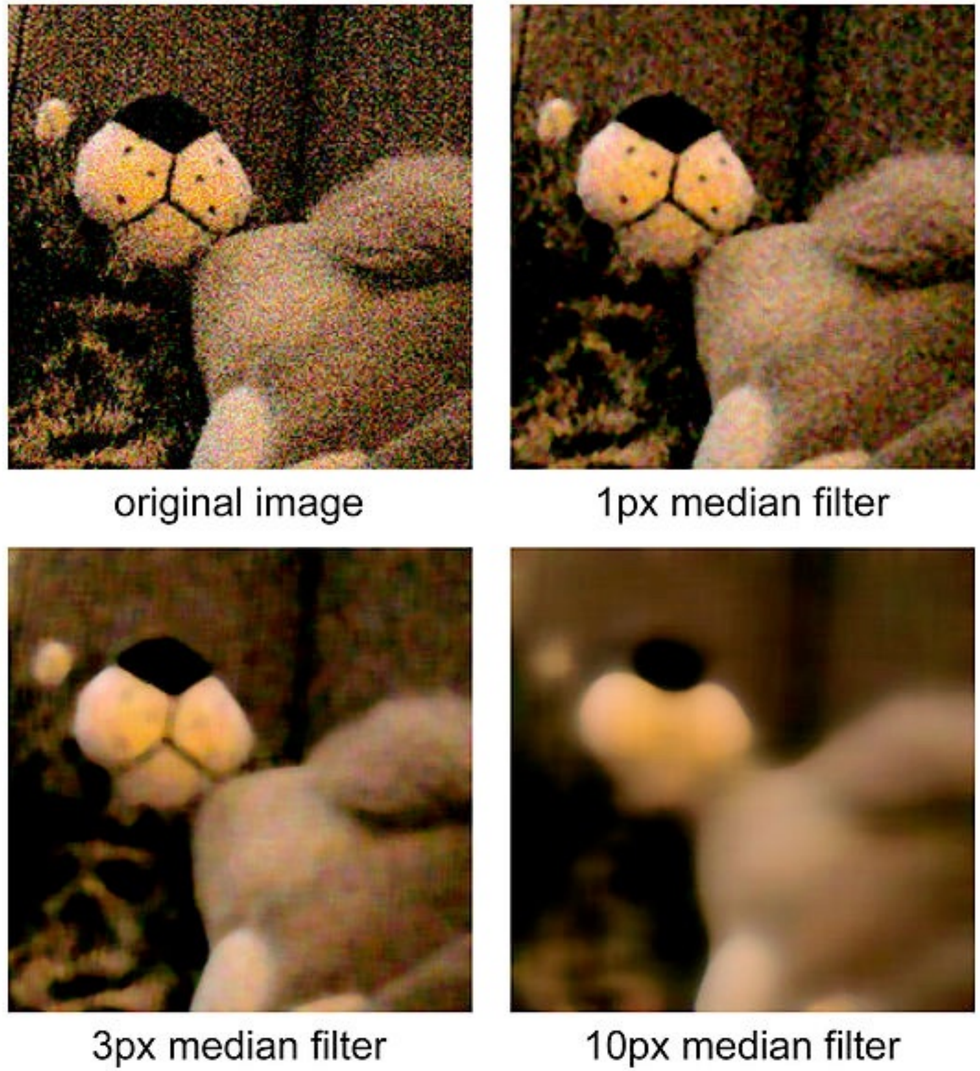




Median filter

- 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: filter weights are 1 or 0 (depending on image content)

```
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
   }
}
```



- 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 $O(N)$?

Bilateral filter

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

Bilateral filter

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

Gaussian blur kernel

Input image

Normalization

For all pixels in support region of Gaussian kernel

Re-weight based on difference in input image pixel values

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

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

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. (non-linear filter: like the median filter, the filter's weights depend on input image content)

Bilateral filter

- Visualization of bilateral filter

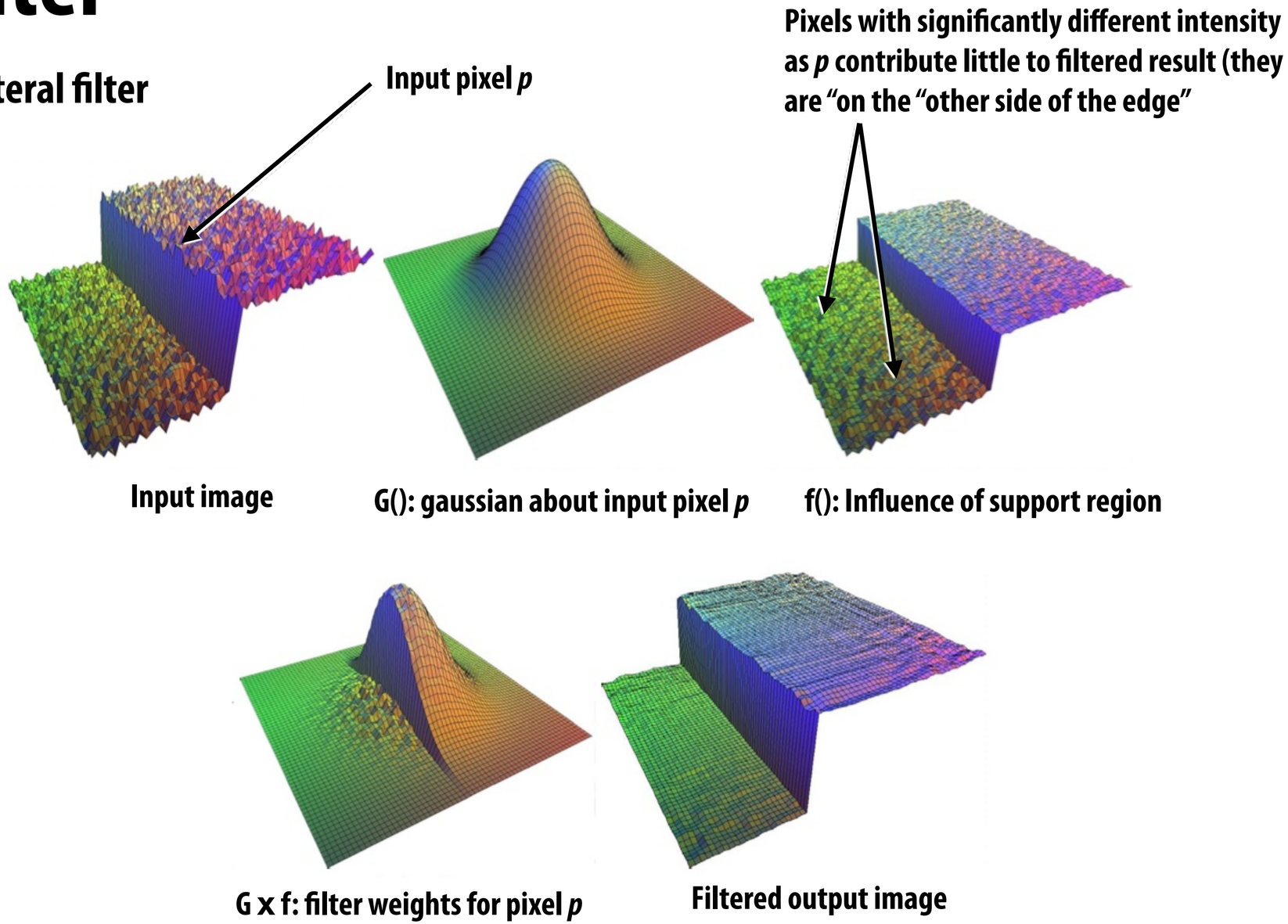
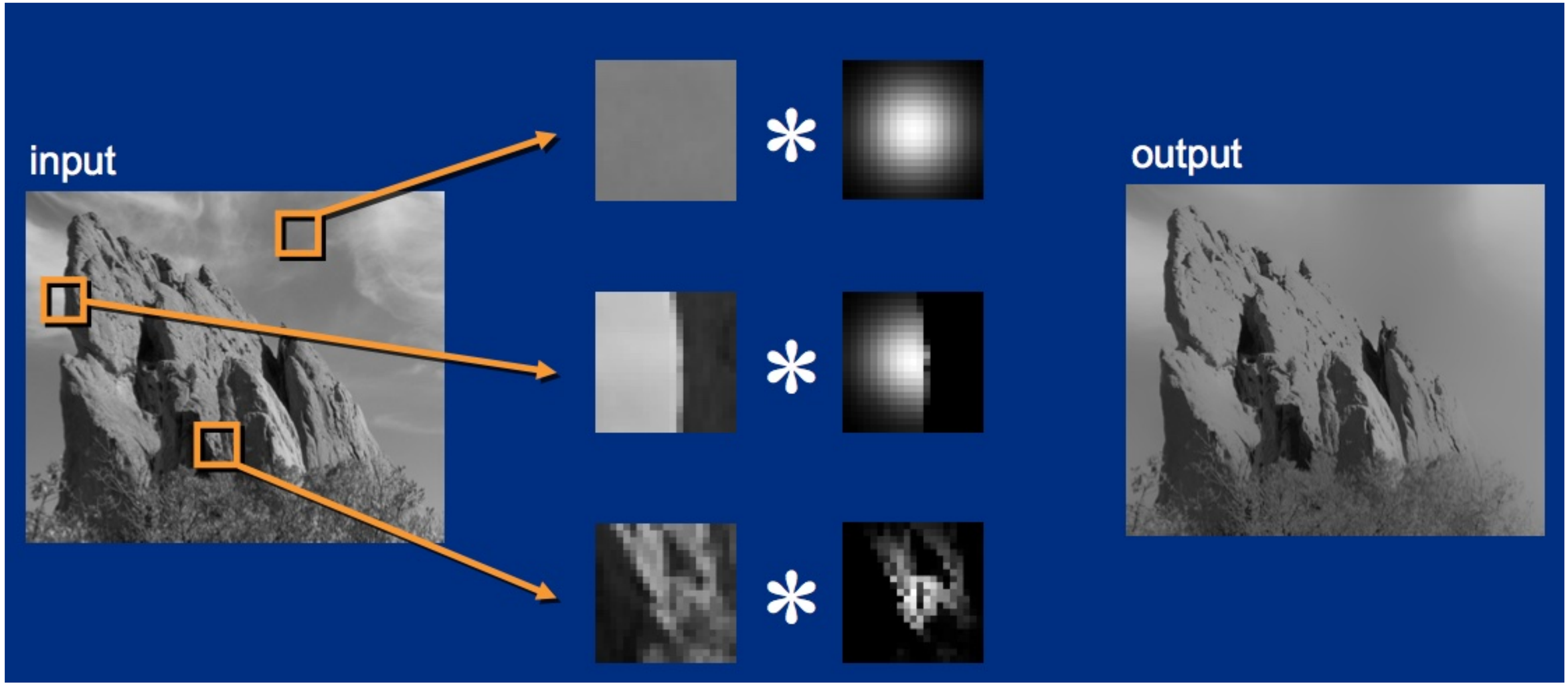


Figure credit: Durand and Dorsey, "Fast Bilateral Filtering for the Display of High-Dynamic-Range Images", SIGGRAPH 2002

Bilateral filter: kernel depends on image content



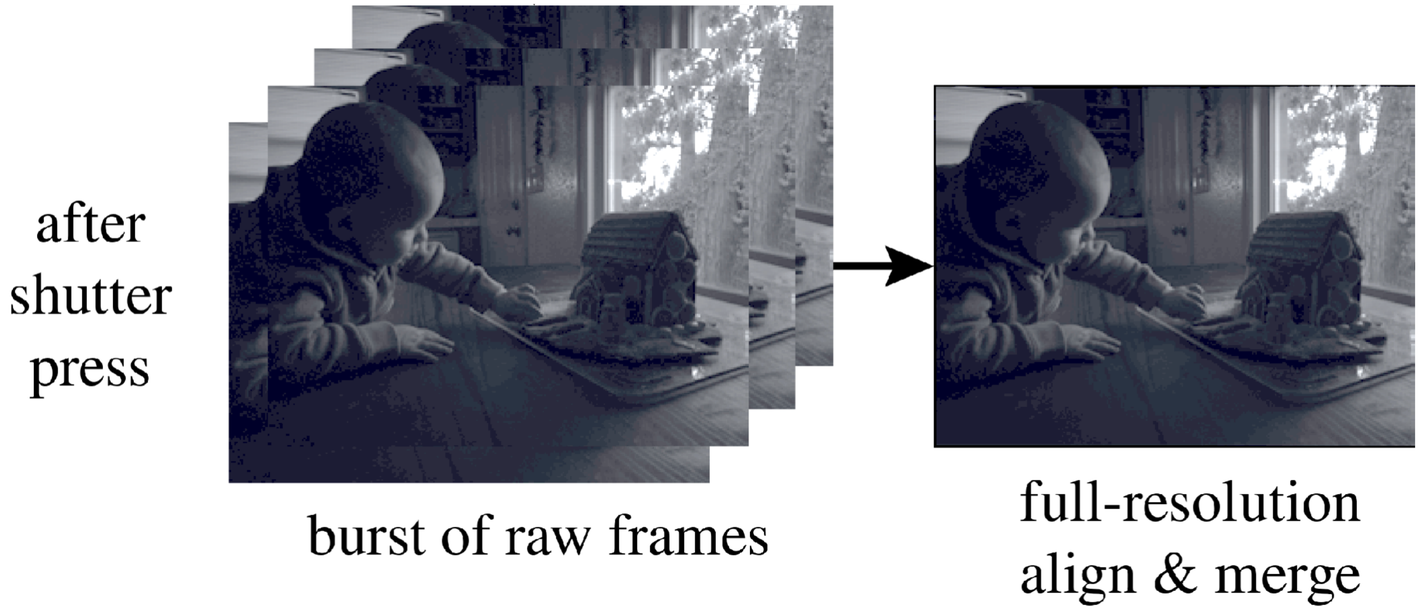
See Paris et al. [ECCV 2006] for a fast approximation to the bilateral filter

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

Better denoising idea: merge sequence of captures

Algorithm used in Google Pixel Phones [Hasinoff 16]

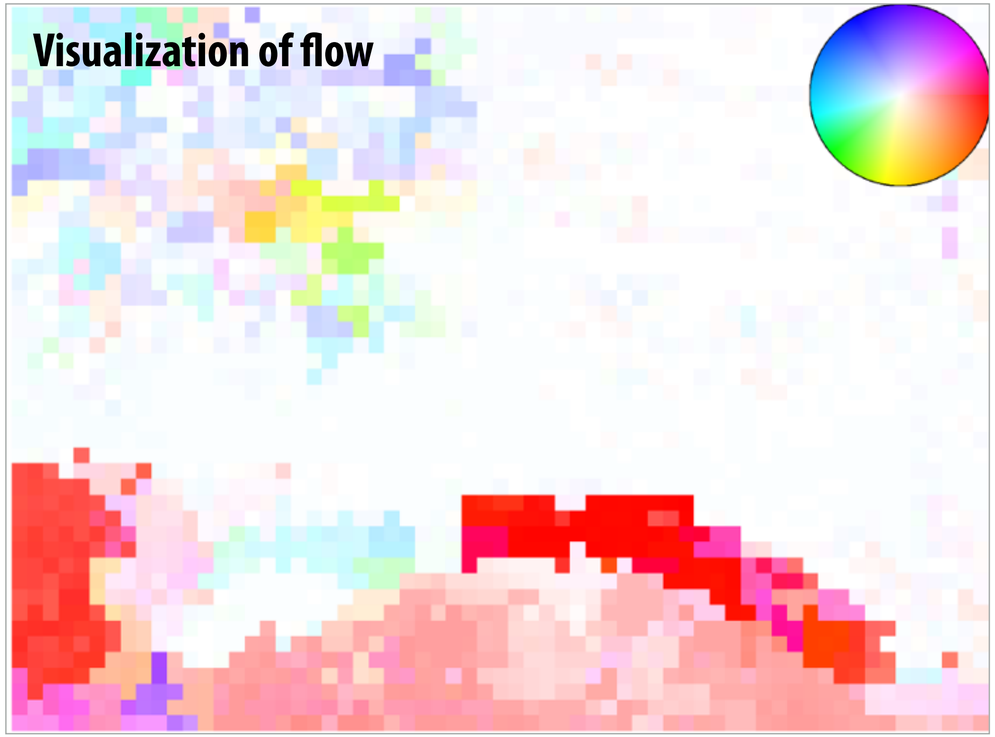
- **Long exposure: reduces noise (acquires more light), but introduces blur (camera shake or scene movement)**
- **Short exposure: sharper image, but lower signal/noise ratio**
- **Idea: take sequence of short full-resolution exposures, but align images in software, then merge them into a single sharp image with high signal to noise ratio**



Google's align-and-merge algorithm

Image pair

- For each image in burst, align to reference frame (use sharpest photo as reference frame)
  - Compute optical flow field aligning image pair
- Simple merge algorithm: warp images according to flow, and sum
- More sophisticated techniques only merge pixels where confidence in alignment is high (tolerate noisy reference pixels when alignment fails)



[Image credit: Hasinoff 16]

Results of align and merge

[Hasinoff 16]

[Image credit: Hasinoff 16]

Saturated Pixels

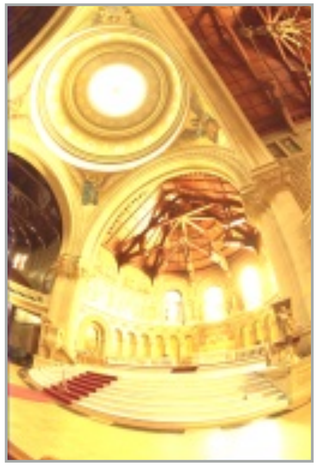
Saturated pixels

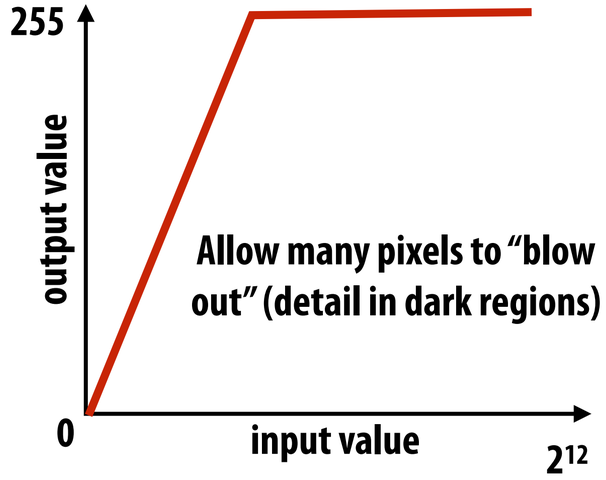
Saturated pixels

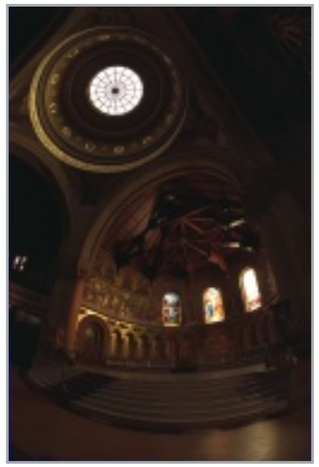
Credit: P. Debevec

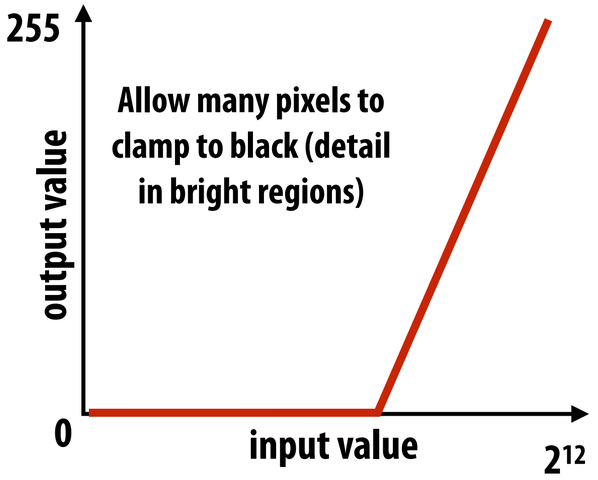
Global tone mapping

- Measured image values (by camera's sensor): 10-12 bits / pixel, but common image formats are 8-bits/pixel
- How to convert 12 bit number to 8 bit number?









High dynamic range image (HDR)
Detail in dark and light images

Image credit: Wikipedia

Local tone adjustment

Pixel values

Weights

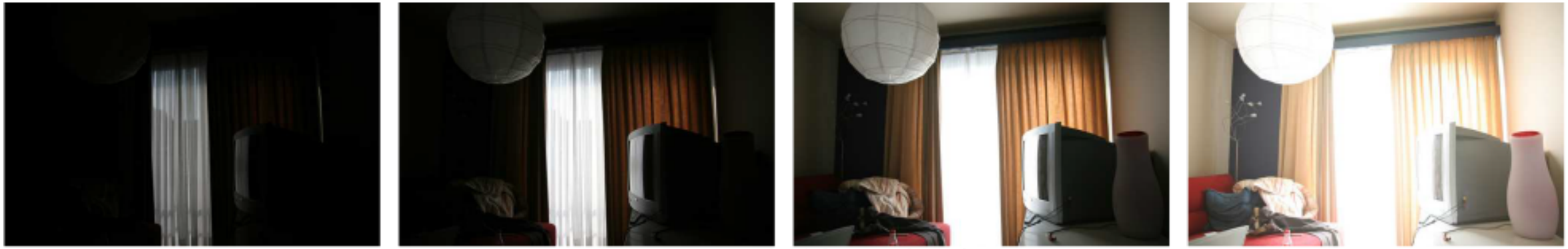
Improve picture's aesthetics by locally adjusting contrast, boosting dark regions, decreasing bright regions

(no physical basis for this)

Combined image
(unique weights per pixel)

Image credit: Mertens 2007

Challenge of merging images



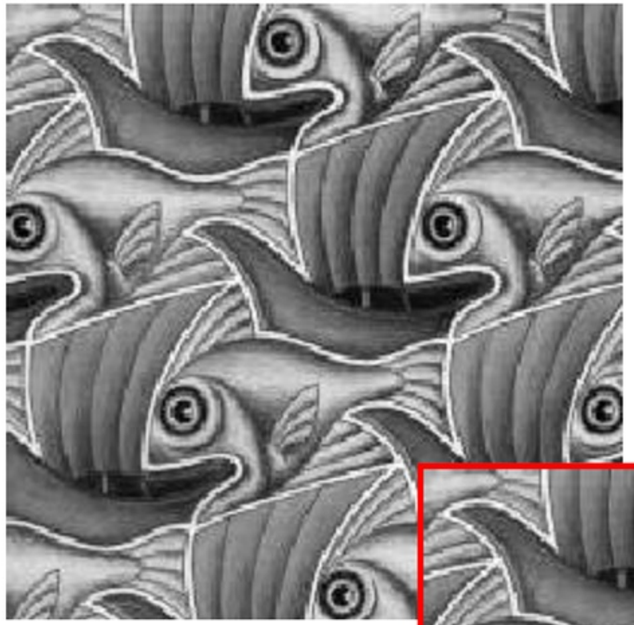
Four exposures (weights not shown)

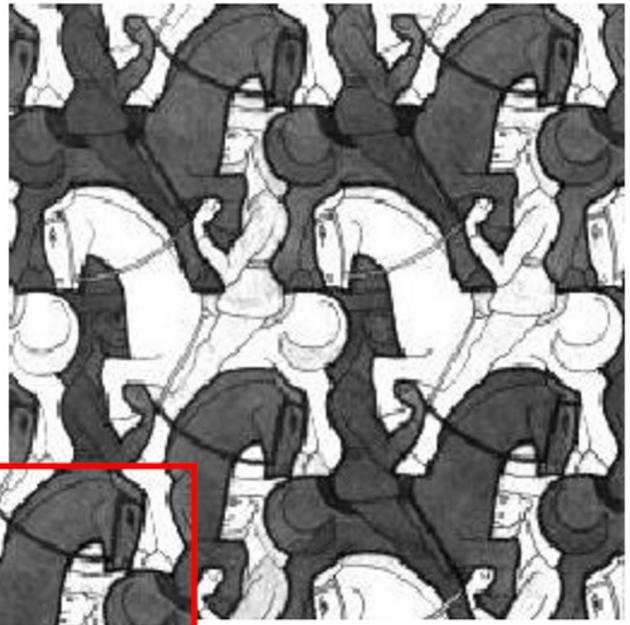
Merged result (based on weight masks)
Notice heavy "banding" since absolute intensity
of different exposures is different

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

Image blending

Consider a simple case where we want to blend two patterns:



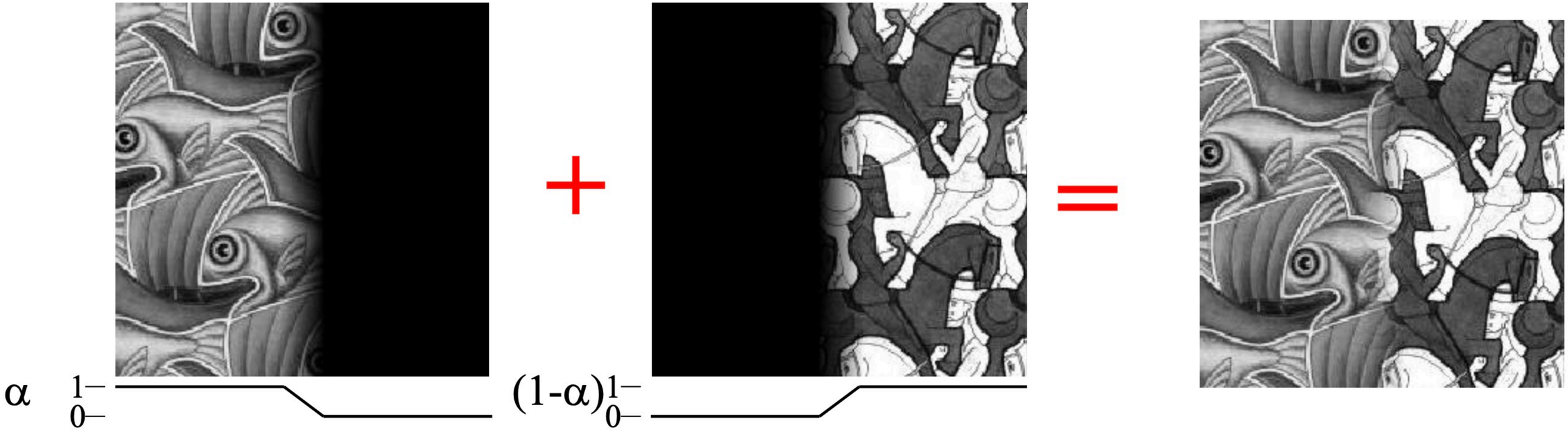


Problem: not "smooth"

Slide credit: Efros

“Feather” the alpha mask

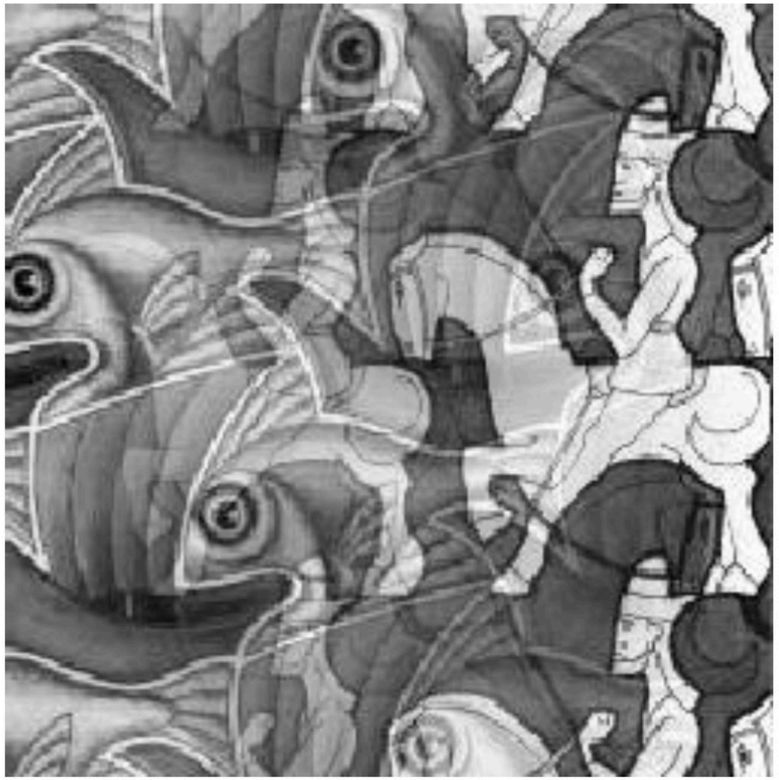
For a “smoother” look…

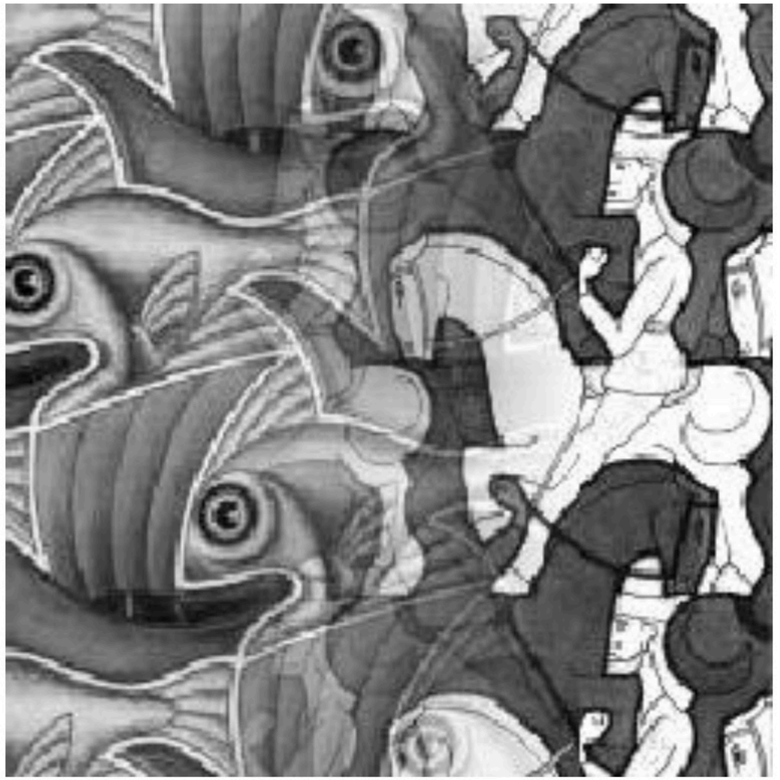


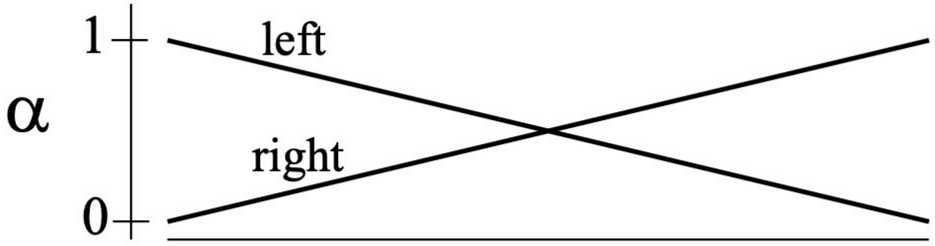
$$I_{blend} = \alpha I_{left} + (1 - \alpha) I_{right}$$

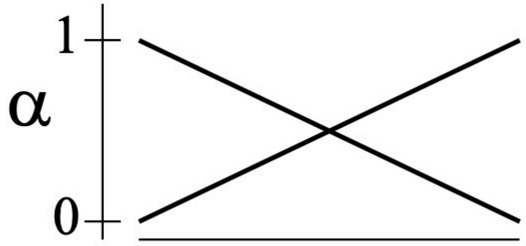
Slide credit: Efros

Effect of feather window size





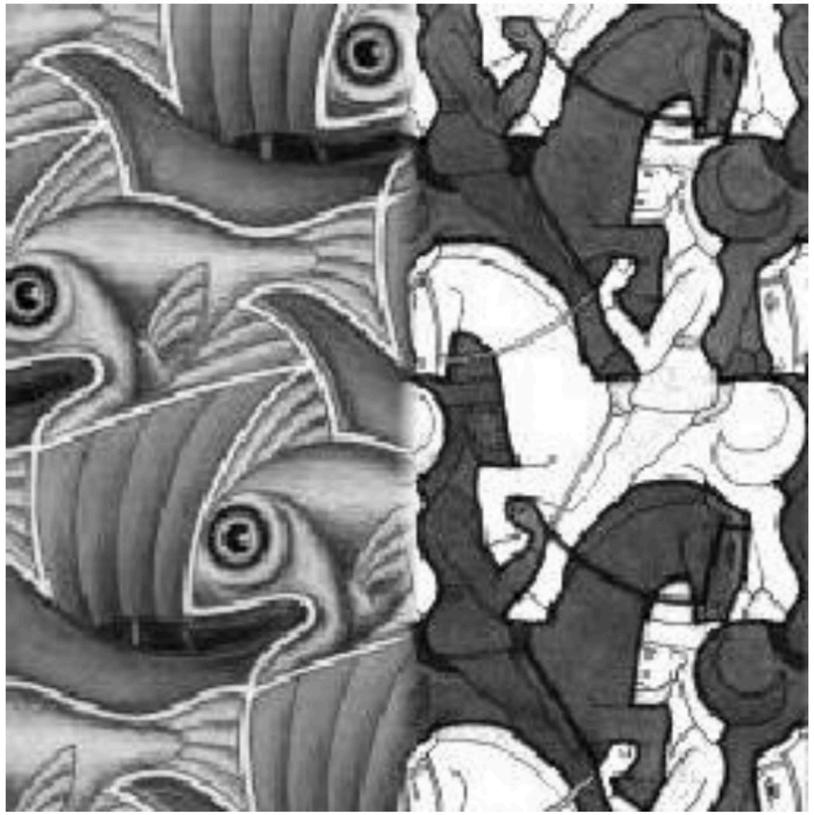


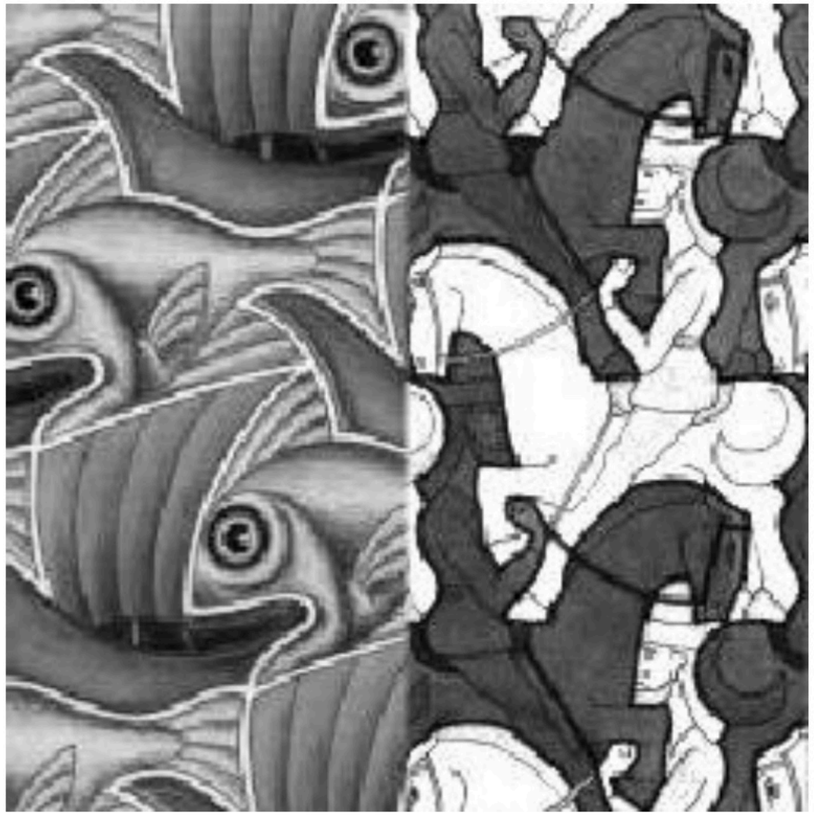


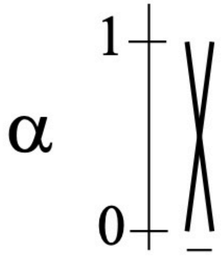
"Ghosting" visible if feather window (transition) is too large

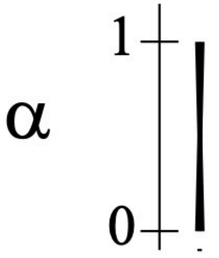
Slide credit: Efros

Effect of feather window size









Seams visible if feather window (transition) is too small

Slide credit: Efros

What do we want

- To avoid seams, transition window should be >= size of largest prominent feature
- To avoid ghosting, transition window should be smaller than ~2X smallest prominent feature
- In other words, the largest and smallest features need to be within a factor of two for feathering to generate good results
- Intuition:
  - Coarse structure of images (large features) should transition slowly between images
  - Fine structure should blend quickly!

Slide credit: Efros, Guerzhoy

Gaussian pyramid

$G_0 = \text{image}$

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

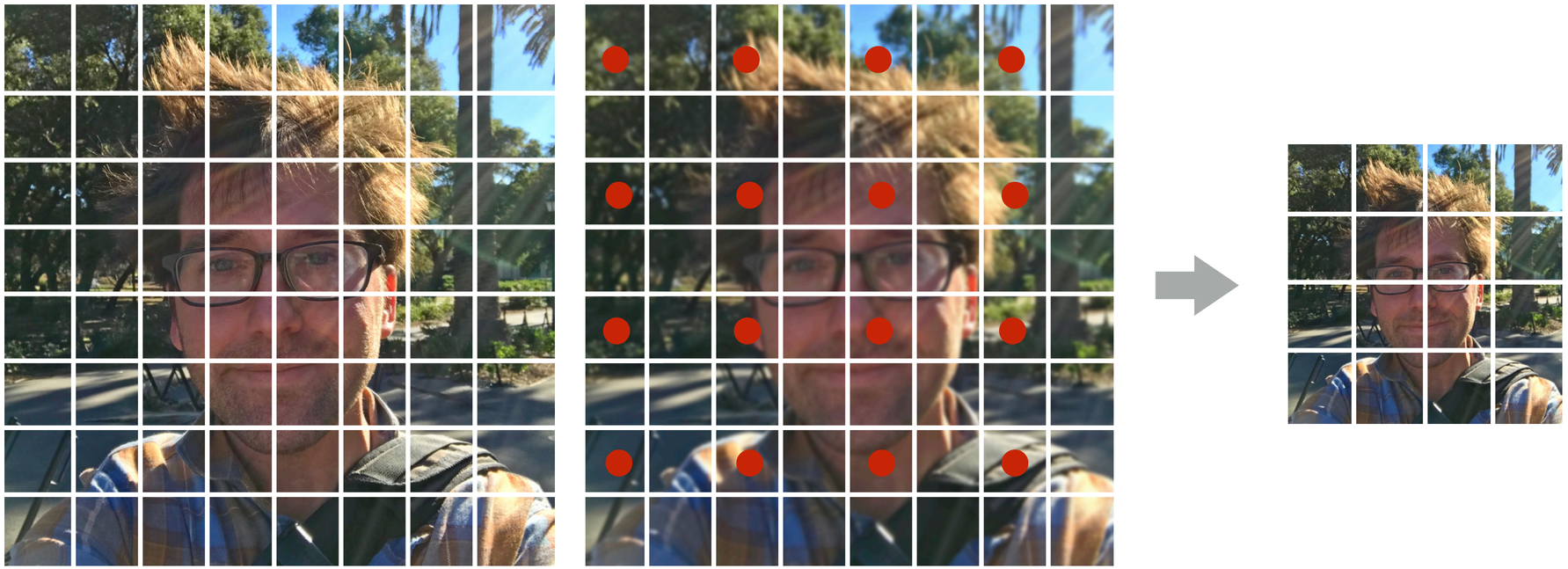
$G_2 = \text{down}(G_1)$

Each image in pyramid contains increasingly low-pass filtered signal

down() = image downsample operation

Downsample

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



Downsample

- Step 1: Remove high frequencies (convolution)
- 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;
  }
}
```

Gaussian pyramid

G_0

Gaussian pyramid

G_1

Gaussian pyramid

G_2

Gaussian pyramid

G_3

Gaussian pyramid

G_4

Gaussian pyramid

G_5

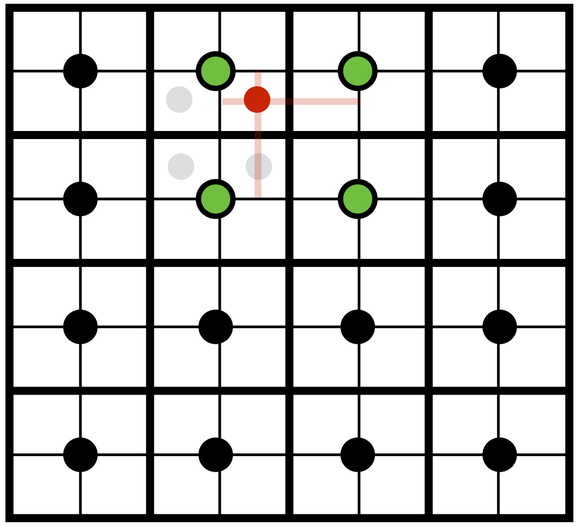
Upsample

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];
  }
}
```



Laplacian pyramid

$L_0 = G_0 - up(G_1)$

[Burt and Adelson 83]

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

Laplacian pyramid

$L_0 = G_0 - up(G_1)$

$L_1 = G_1 - up(G_2)$

Laplacian pyramid

$L_0 = G_0 - up(G_1)$

$L_1 = G_1 - up(G_2)$

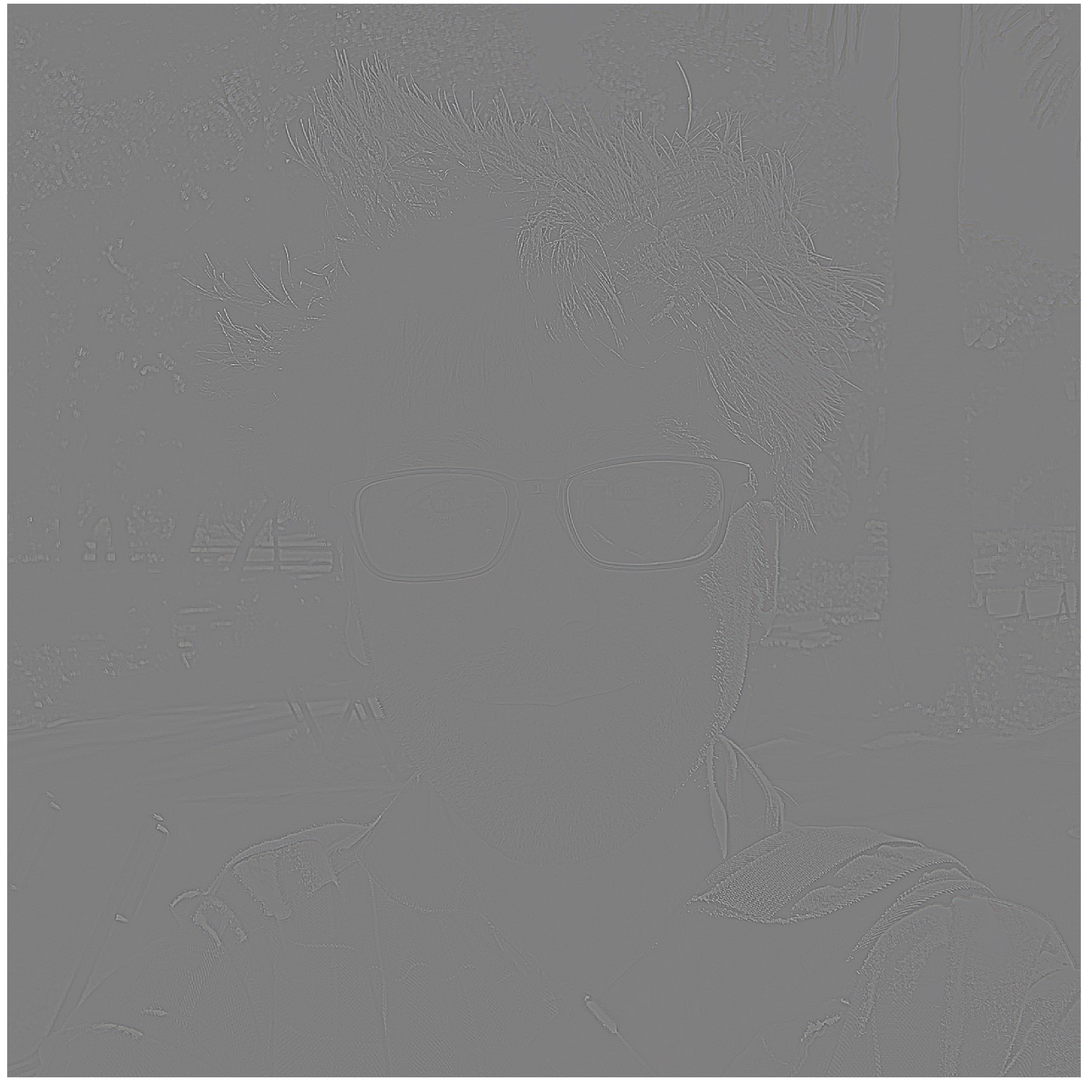
$L_2 = G_2 - up(G_3)$

$L_3 = G_3 - up(G_4)$

$L_4 = G_4$

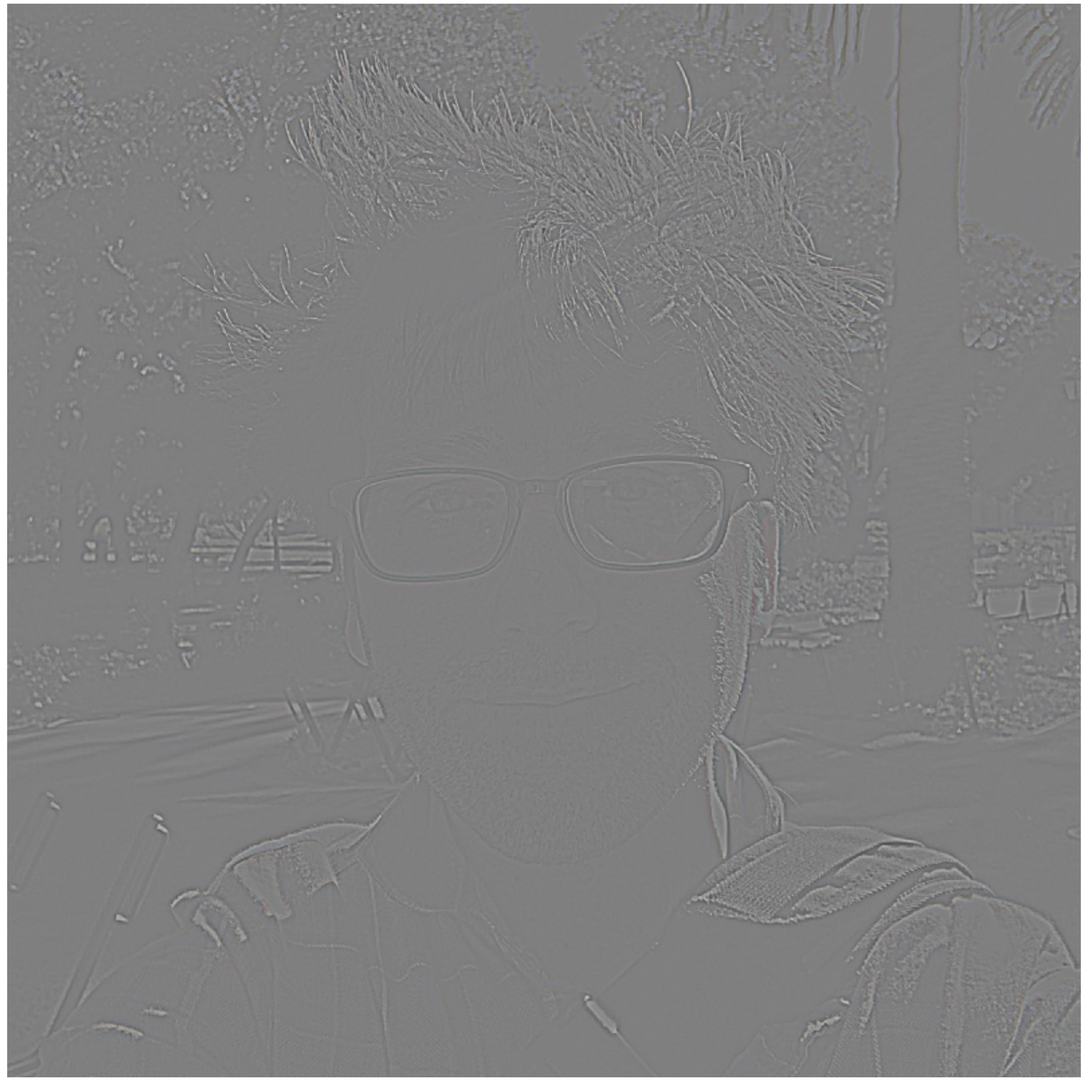
Question: how do you reconstruct original image from its Laplacian pyramid?

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

$L_5 = G_5$

Gaussian/Laplacian pyramid summary

- Gaussian and Laplacian pyramids are image representations where each pixel maintains information about frequency content in a region of the image

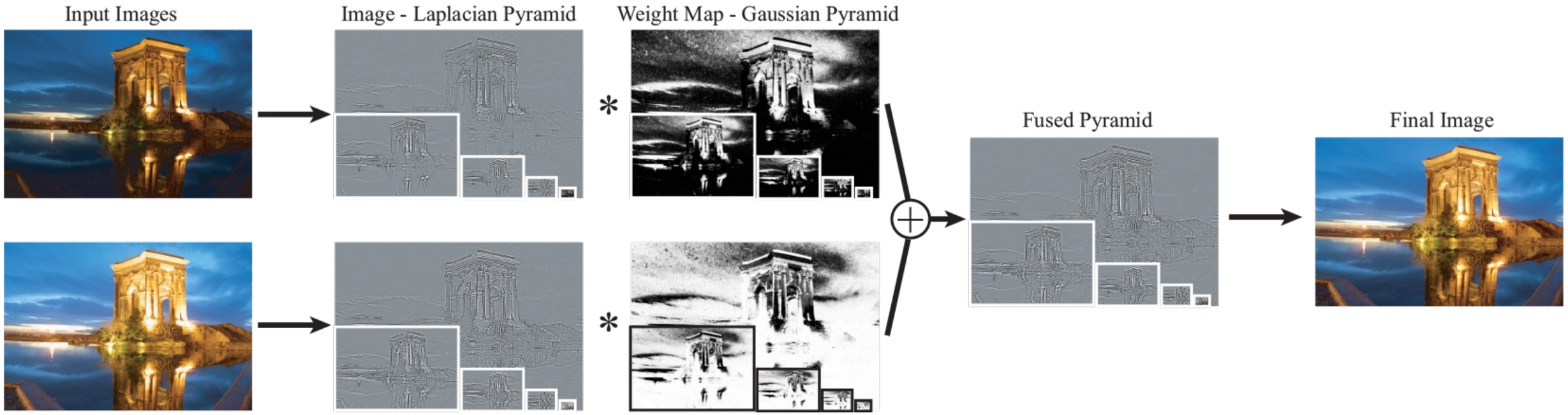
- $G_i(x,y)$ — frequencies up to limit given by *i*

- $L_i(x,y)$ — 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 local tone mapping

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



Merging Laplacian pyramids

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?

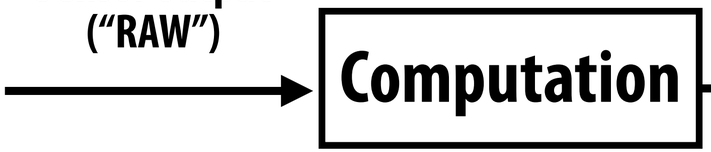
More sophisticated manipulations… magic eraser

(Feature in recent Google Pixel phones)

Summary

- Computation now a fundamental part of producing a pleasing photograph
- Used to compensate for physical constraints (demosaic, denoise, lens corrections, portrait mode)
- Used to analyze image to estimate system parameters (autofocus, autoexposure, white balance, depth estimation)
- Used to make non-physically plausible images that have aesthetic merit
- Principles from this class are central to many of these operations

Sensor output ("RAW")



Beautiful image that impresses your friends on Instagram