Lecture 2:

Digital Camera Image Processing

Visual Computing Systems
Stanford CS348K, Spring 2026

Discussion: The role of specifying goals and constraints

- Consider the task: make a system that does X?
  - It's easy to satisfy X if there are no constraints

- A system is the result of a set of decisions by a designer
  - For each decision there are likely multiple alternatives

- The goals and constraints are the basis for evaluating whether the design decisions were good ones.
  - By comparing the chosen decision against some of the alternatives

Digital Camera Basics
A review (if you've taken CS248A)
A quick primer (for everyone else)

Key point: substantial computation produces the output we see in a digital photograph

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 fundamental to producing high-quality photographs.

Sensor output ("RAW")

Beautiful image that impresses your Instagram friends

Camera cross section (in a smart phone)

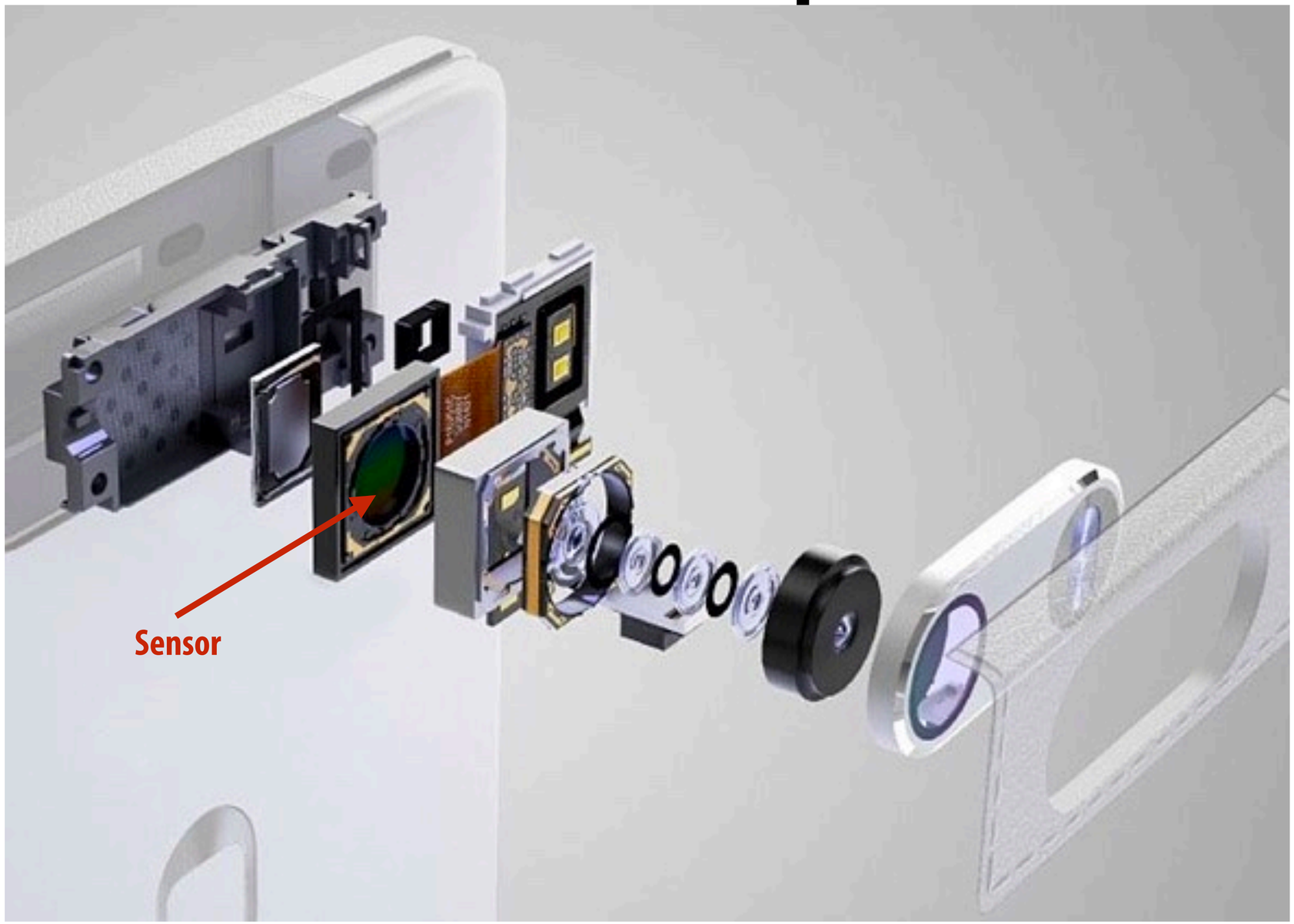
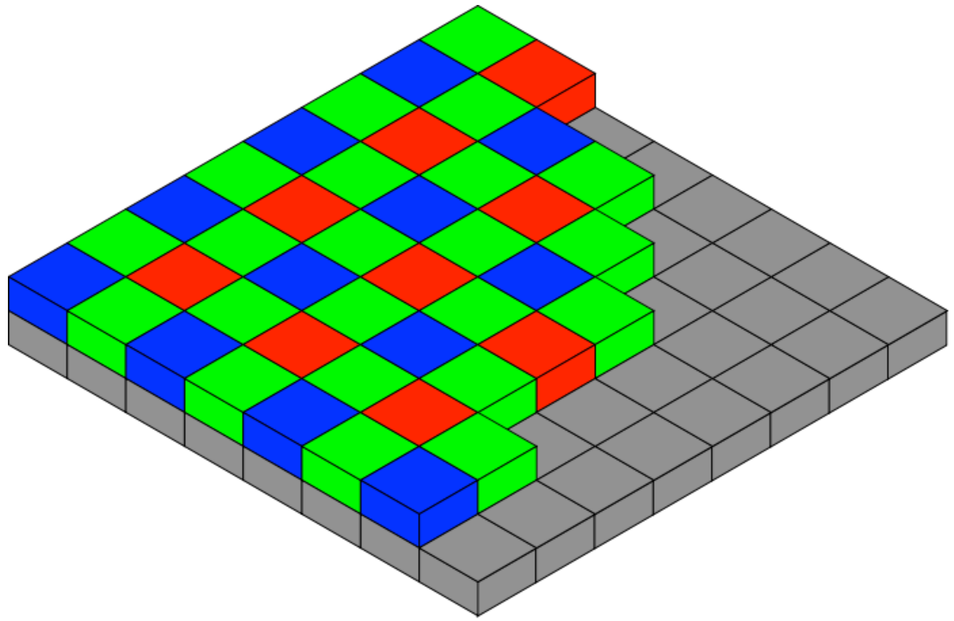


Image credit: https://www.dpreview.com/news/3717128828/the-future-is-bright-technology-trends-in-mobile-photography

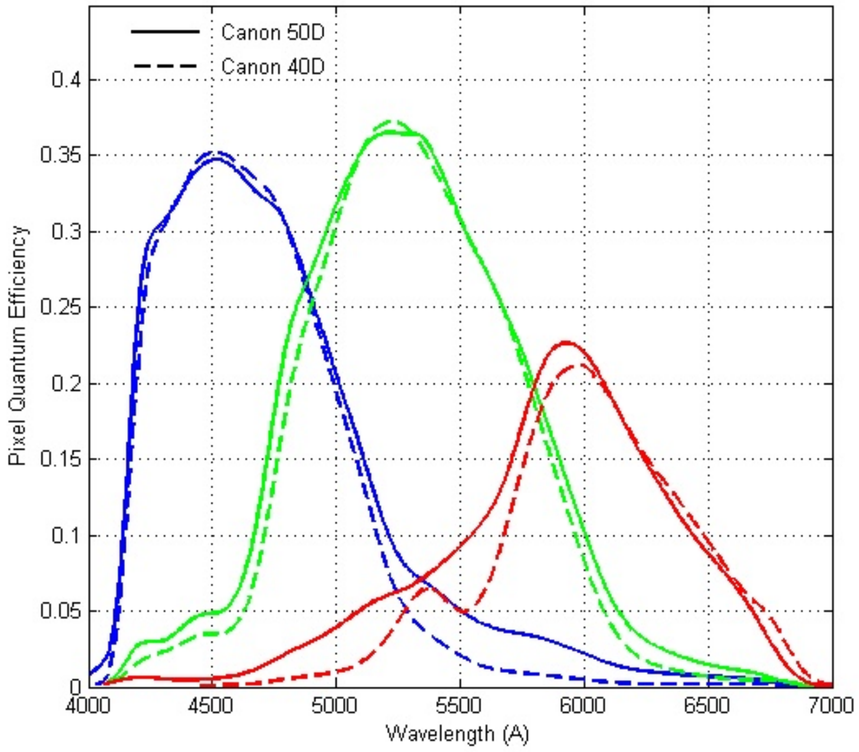
Color filter array (Bayer mosaic) on the sensor

- 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

- 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

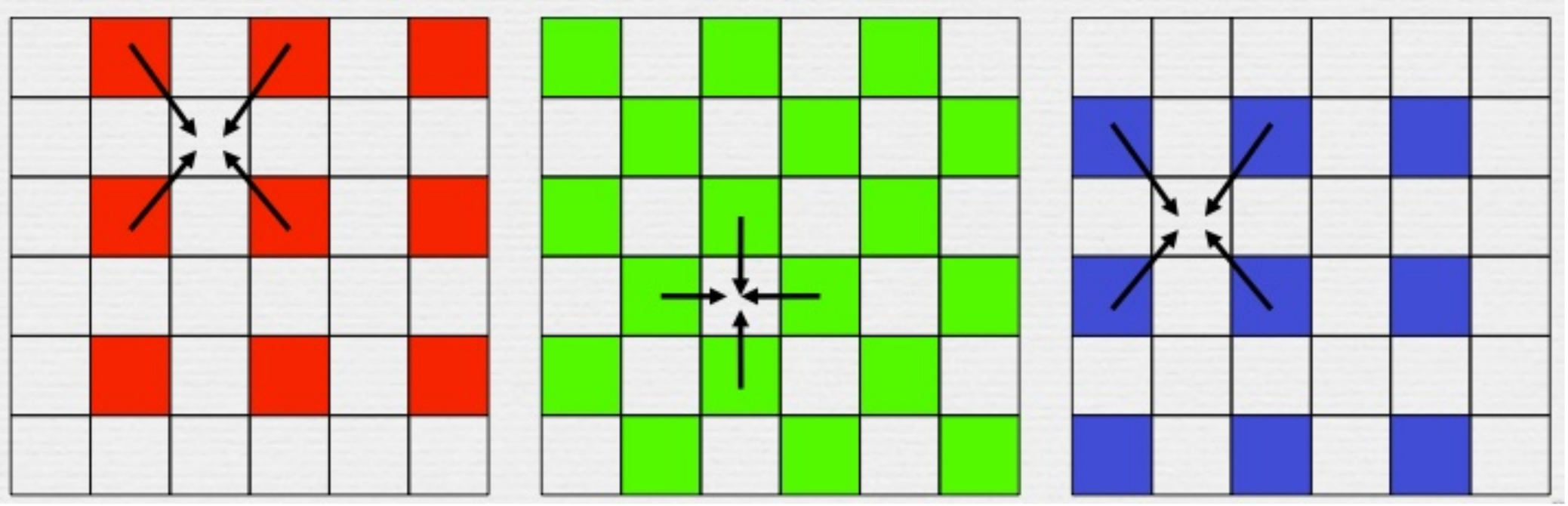


Image credit: Mark Levoy

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

Long exposure: walking people are blurred…


Also: still significant noise in
dark regions (not long enough!)

Sources of measurement noise

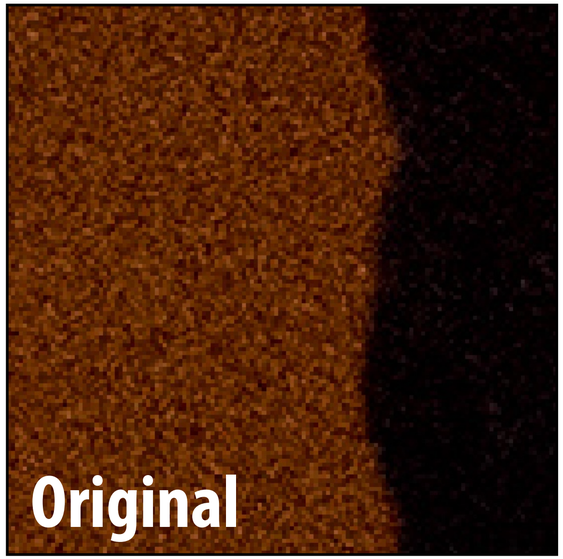
- Photon shot noise:
  - Photon arrival rate takes on Poisson distribution
  - Standard deviation = sqrt(N) (N = number of photon arrivals)
  - Signal-to-noise ratio (SNR) = N/sqrt(N)
  - Implication: brighter the signal, the higher the SNR
- Dark-shot noise
  - Due to leakage current in sensor
  - Electrons 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

Problem

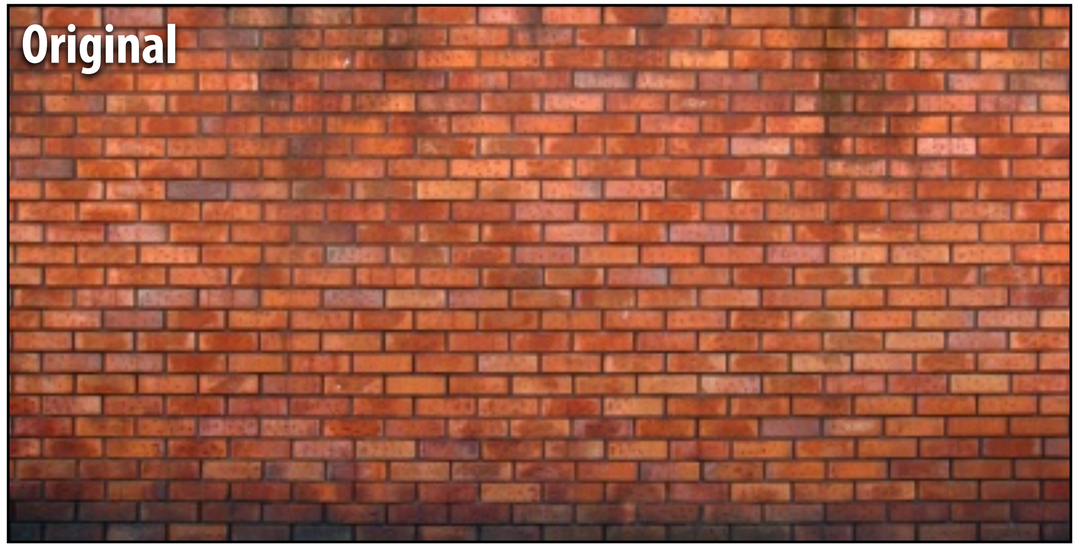
- Long exposure: reduces noise (acquires more light), but introduces blur (camera shake or scene movement)

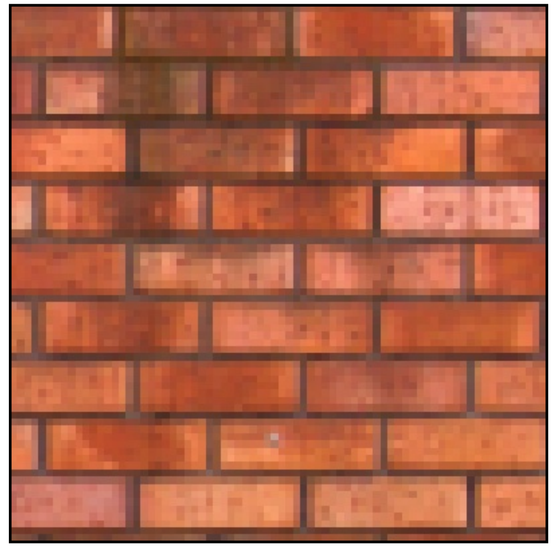
- Short exposure: sharper image, but lower signal/noise ratio

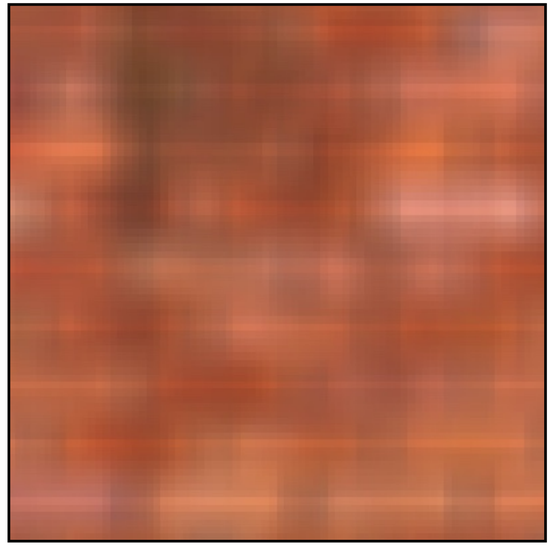
Denoising: attempting to genoise noise via image processing



7x7 box blur







Gaussian blur

- Obtain filter coefficients from sampling 2D Gaussian

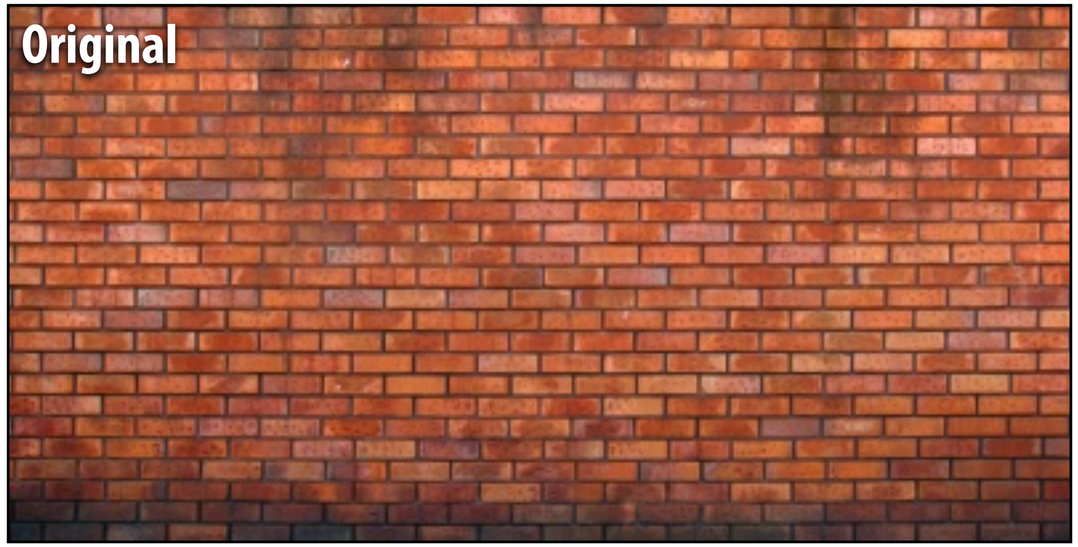
$$f(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$

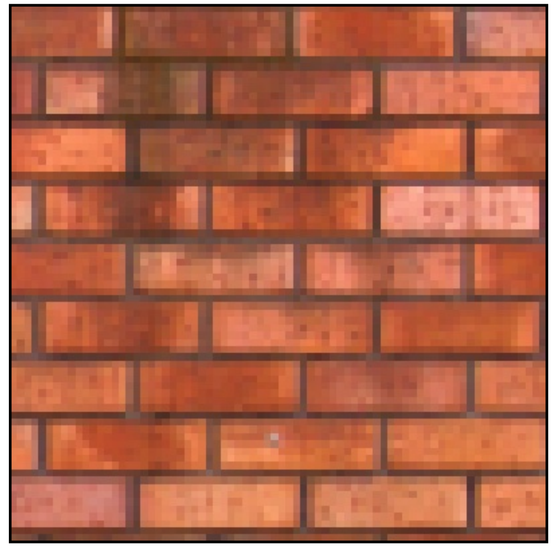
- Produces weighted sum of neighboring pixels (contribution falls off with distance)
  - In practice: truncate filter beyond certain distance for efficiency

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

Note: this is a 5x5 truncated Gaussian filter

7x7 gaussian blur





Bilateral filter: data-dependent filter weights

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: $f(x) = 0$ 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)

Addressing the problem by taking multiple shots

- Imagine I took a sequence of N photos with exposure time T…
- And then I directly added the values in the pixels up

- What would I get?

- What is the problem with this approach?

Addressing the problem by taking multiple shots

- Exposure bracketing
- Take a short-exposure photo (sharp, but higher noise) AND take a long-exposure video (blurry, but low noise)
  - And use computation to merge them into a single sharp, low noise photo

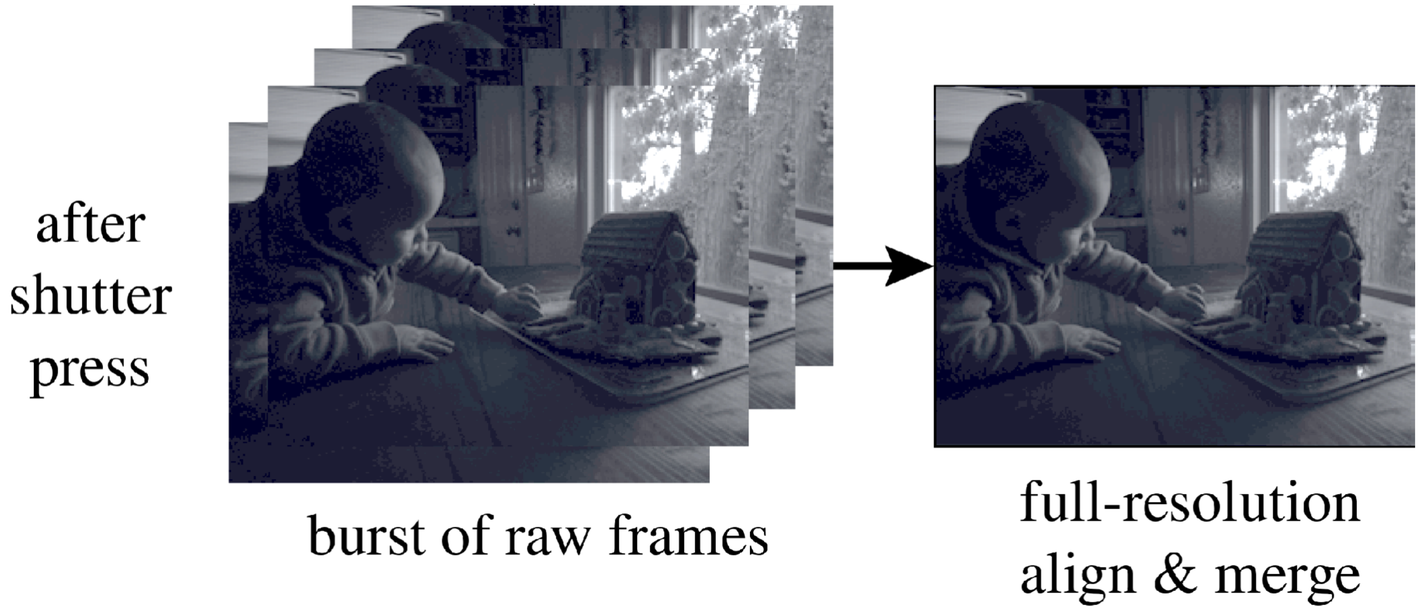
no-flash flash result

Another variant: flash-no-flash photography [Eisemann and Durand]
(use flash image for sharp, colored image, infer room lighting from no-flash image)

Google's denoising idea: merge sequence of captures

Algorithm used in Google Pixel Phones [Hasinoff 16]

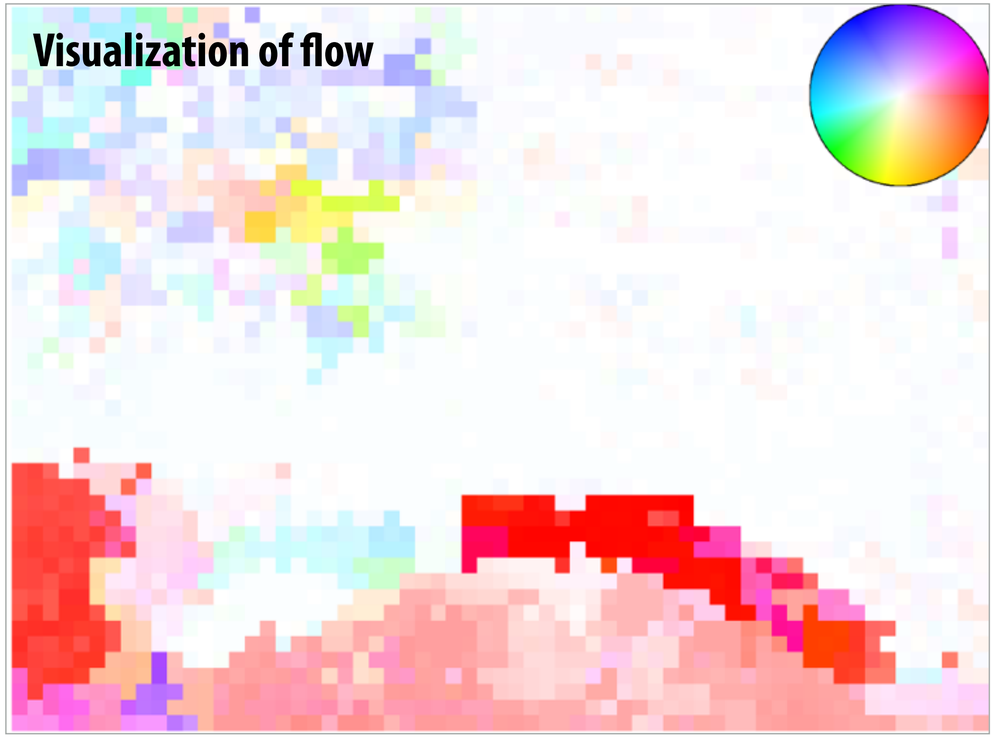
- Idea: take sequence of short exposures (all underexposed!), but align images in software, then merge them into a single sharp image with high signal to noise ratio



Google's align-and-merge algorithm (tonight's reading)

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]

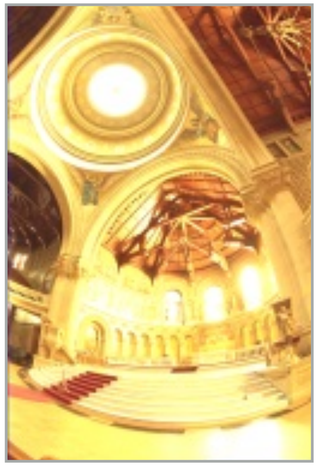
[Image credit: Hasinoff 16]

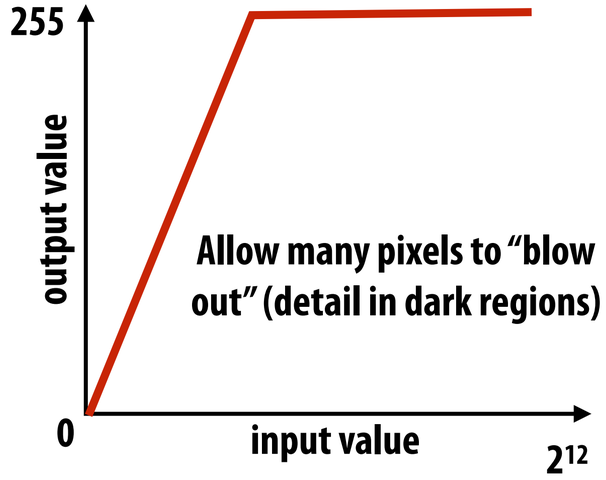
Another problem: saturated pixels

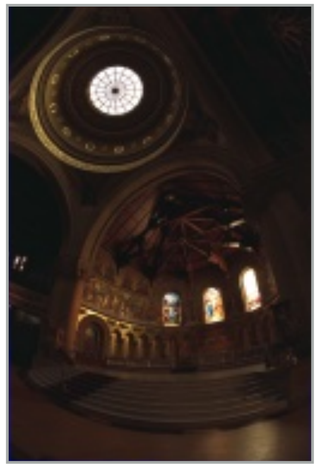
Credit: P. Debevec

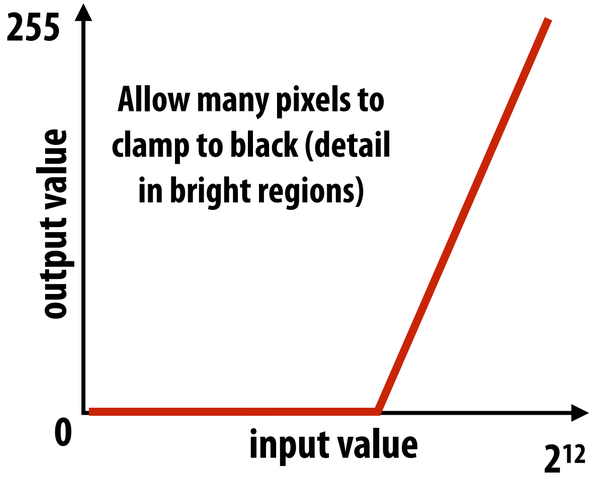
Global tone mapping

- 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?

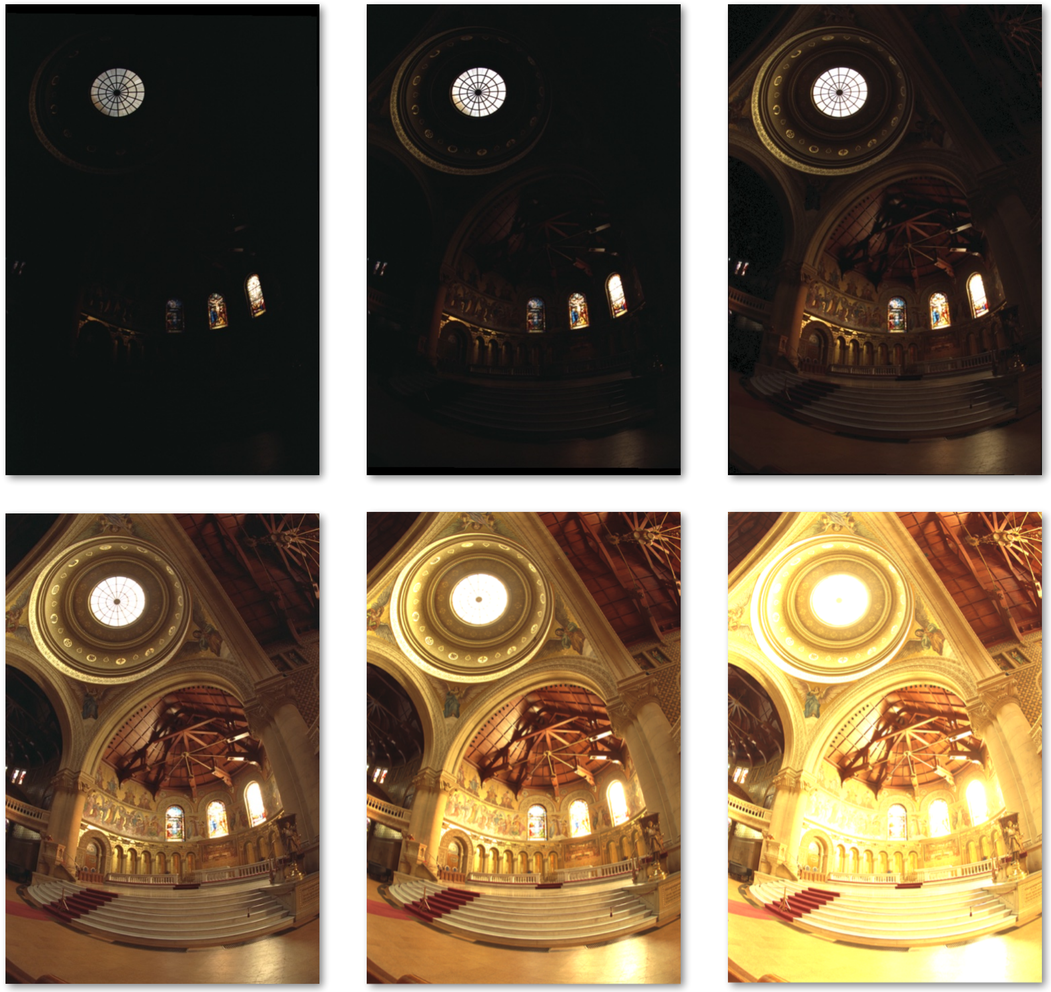




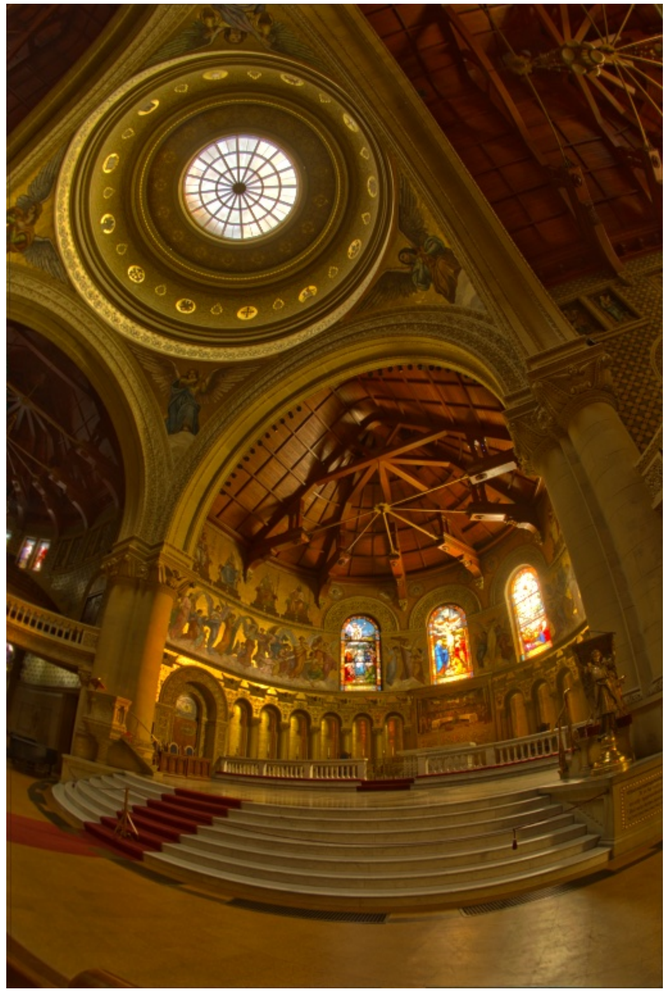




Multi-shot photography example: high dynamic range (HDR) images



Source photographs: each photograph has different exposure



Tone mapped HDR image

Credit: Debevec and Malik

Local tone adjustment

Pixel values

Weights

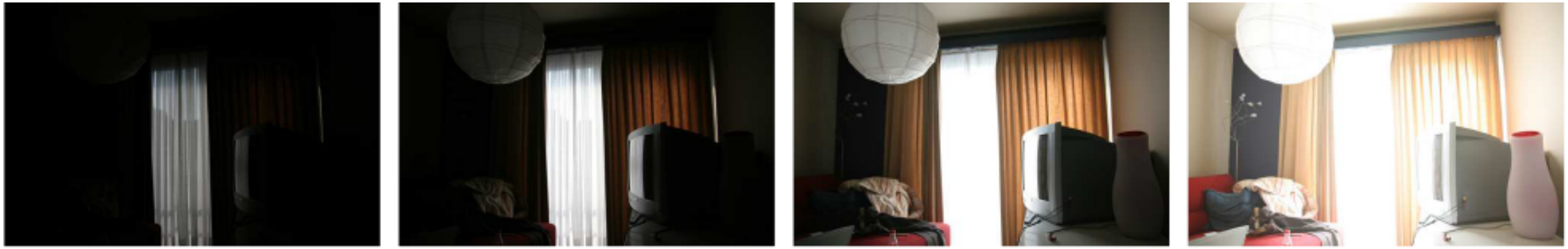
Improve picture's aesthetics by locally adjusting contrast, boosting dark regions, decreasing bright regions

(no physical basis for this)

Combined image (unique weights per pixel)

Image credit: Mertens 2007

Challenge of merging images



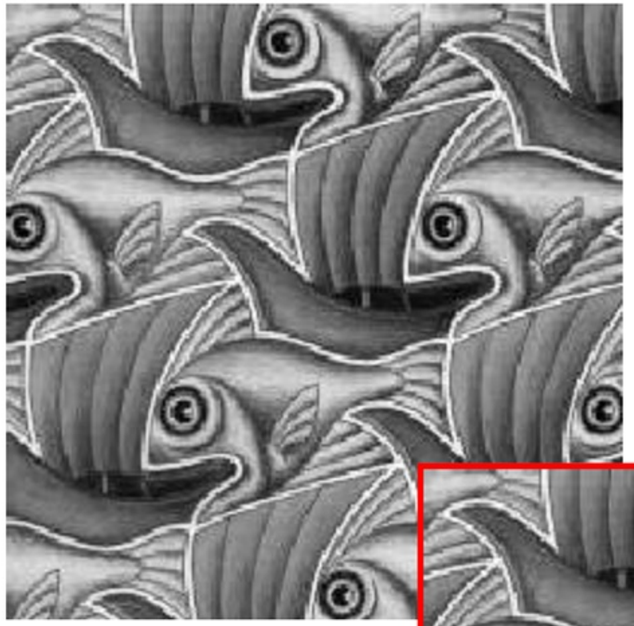
Four exposures (weights not shown)

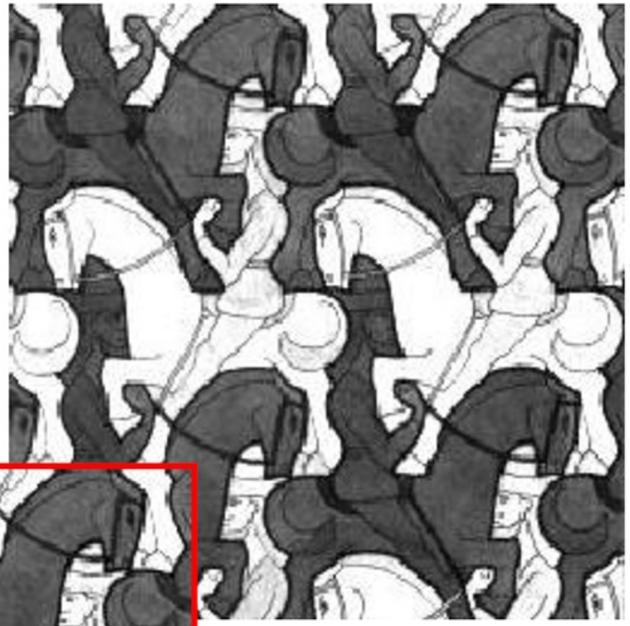
Merged result (based on weight masks)
Notice heavy "banding" since absolute intensity
of different exposures is different

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

Image blending

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



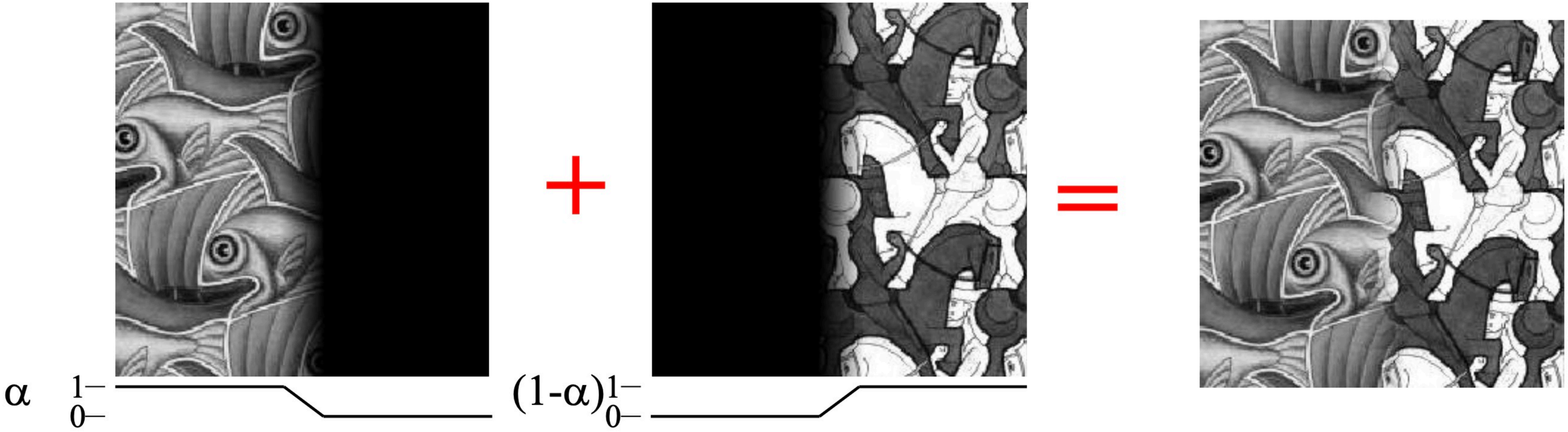


Problem: not "smooth"

Slide credit: Efros

“Feather” the alpha mask

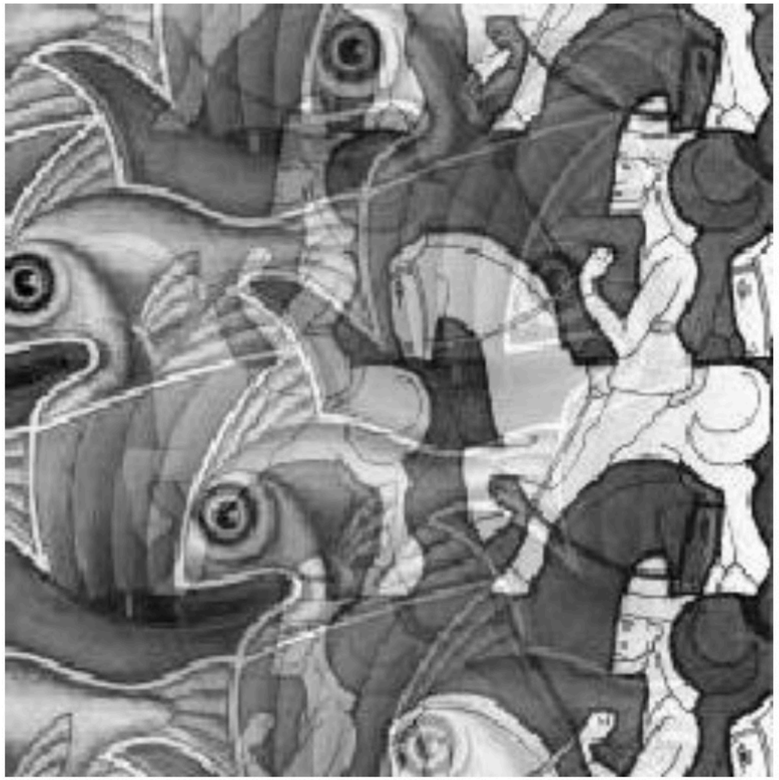
For a “smoother” look…

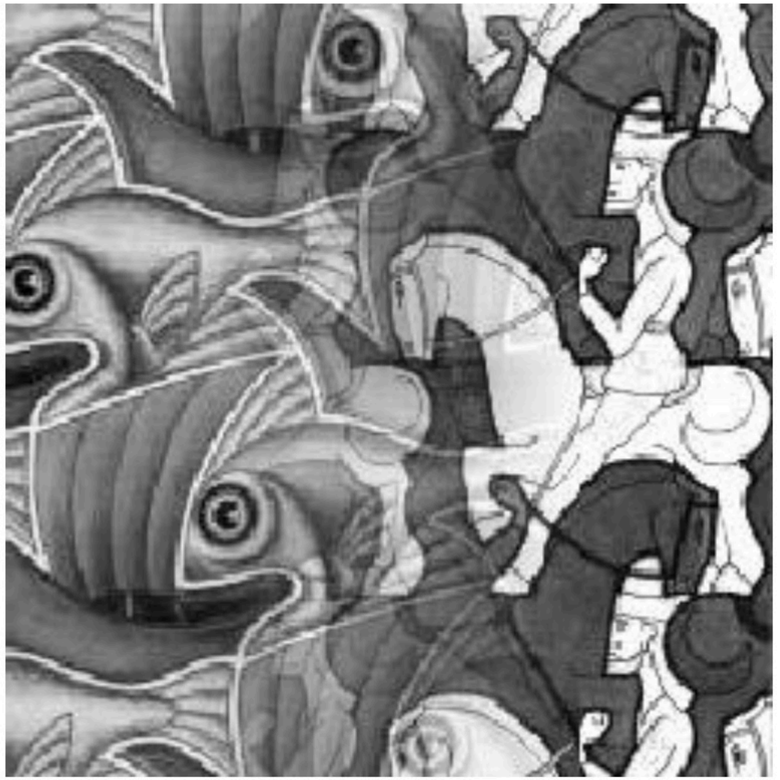


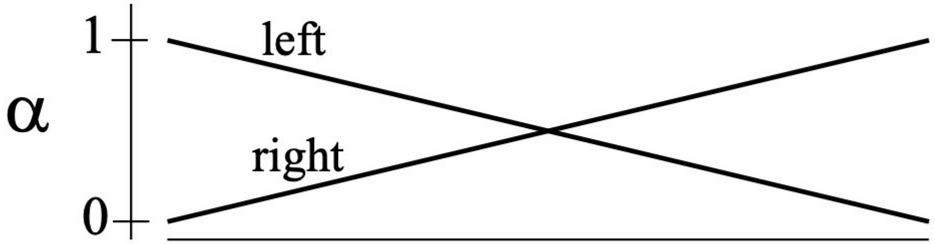
$$I_{blend} = \alpha I_{left} + (1 - \alpha) I_{right}$$

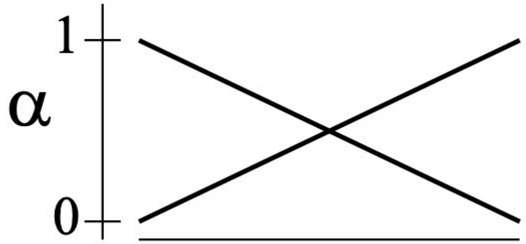
Slide credit: Efros

Effect of feather window size





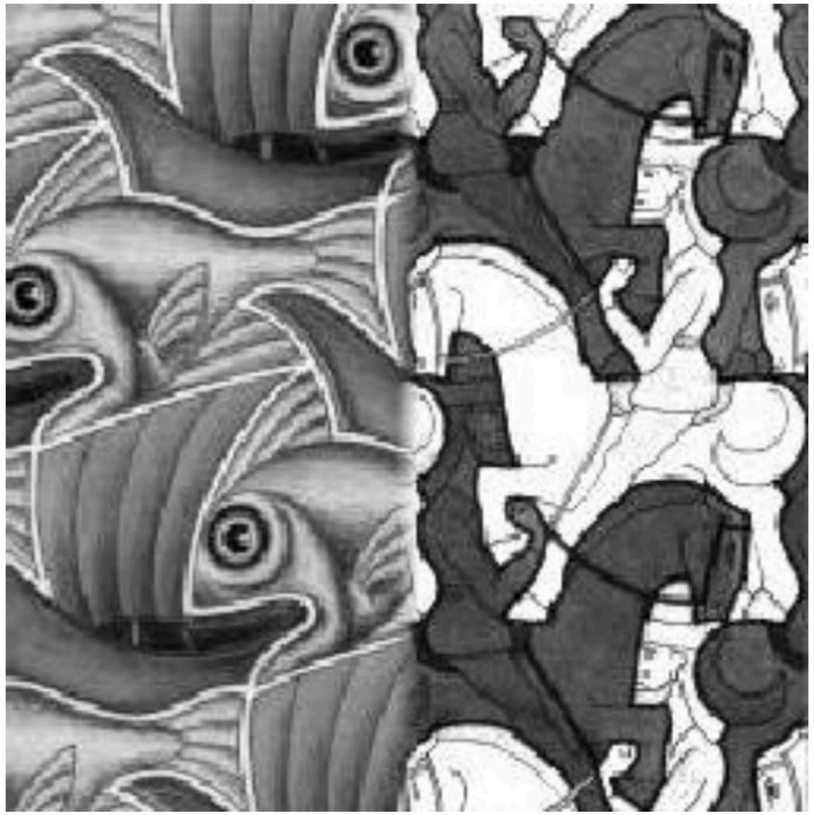


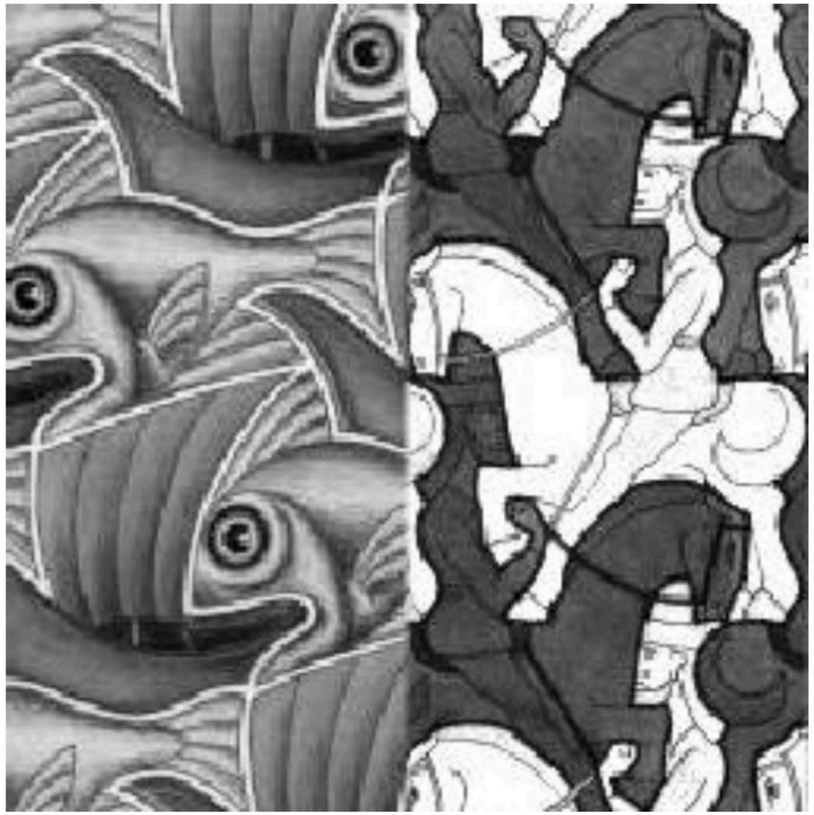


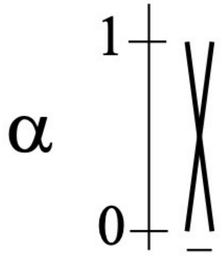
“Ghosting” visible if feather window (transition) is too large

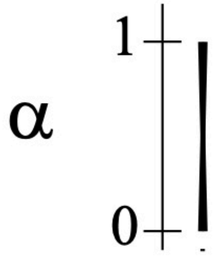
Slide credit: Efros

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

$G_0 = \text{image}$

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

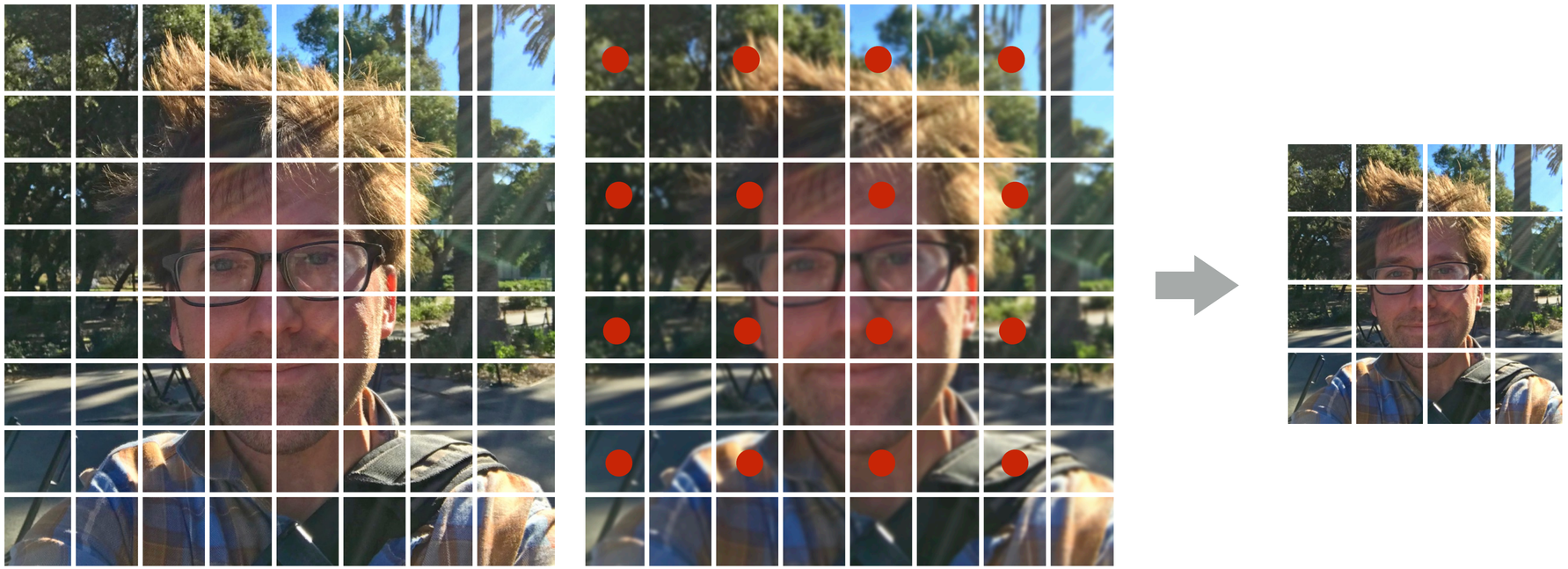
$G_2 = \text{down}(G_1)$

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

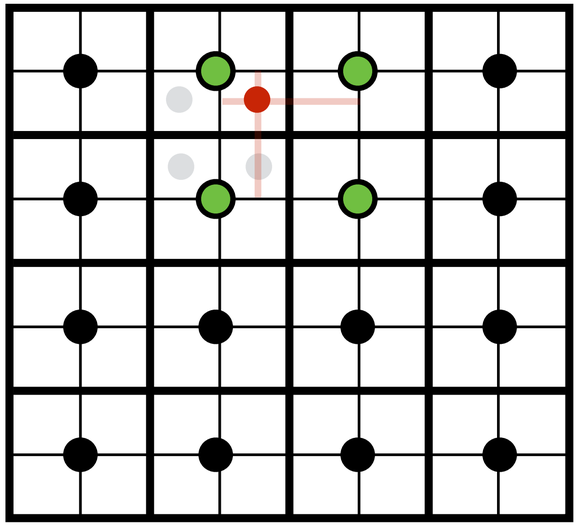
Upsample

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

$L_0 = G_0 - up(G_1)$

$L_1 = G_1 - up(G_2)$

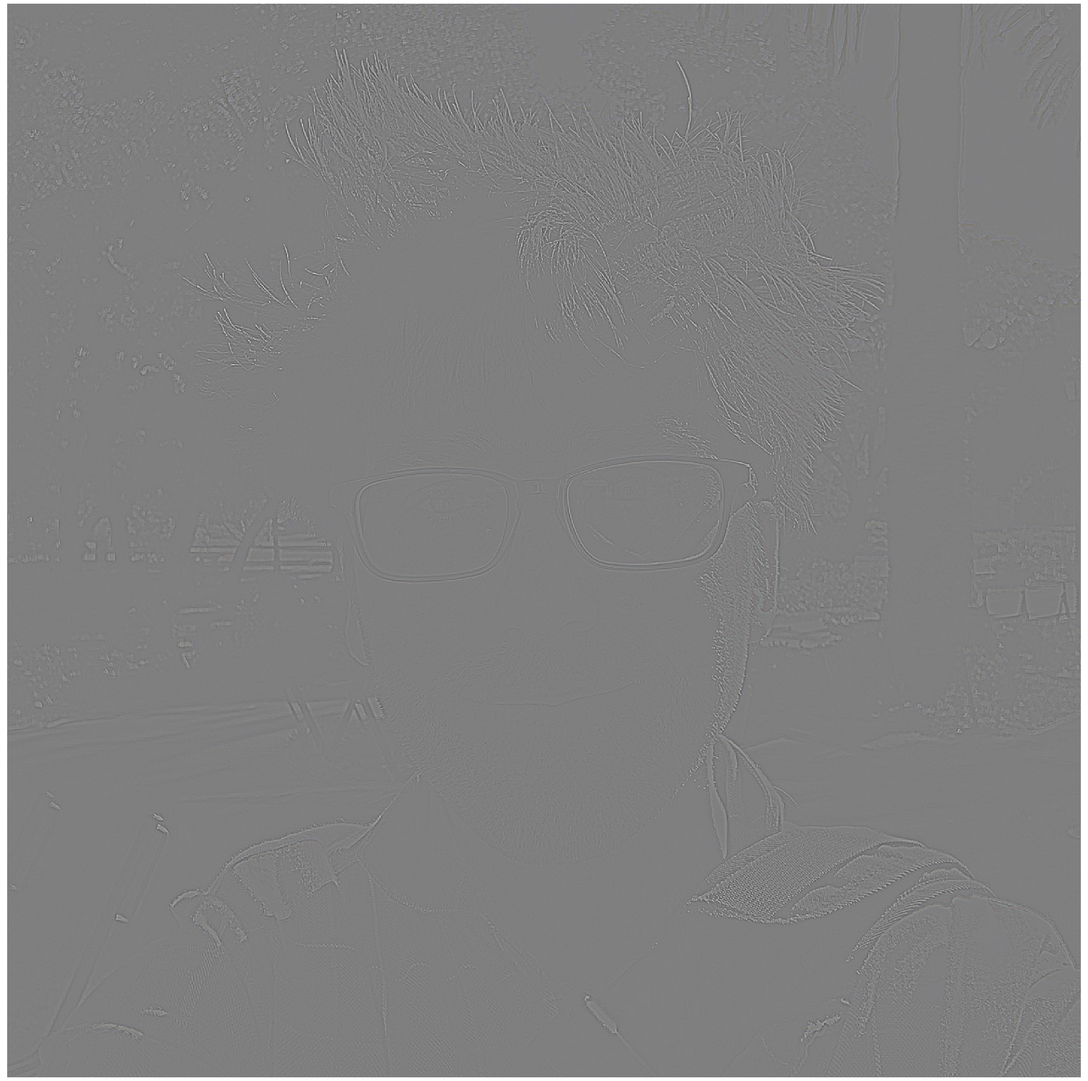
$L_2 = G_2 - up(G_3)$

$L_3 = G_3 - up(G_4)$

$L_4 = G_4$

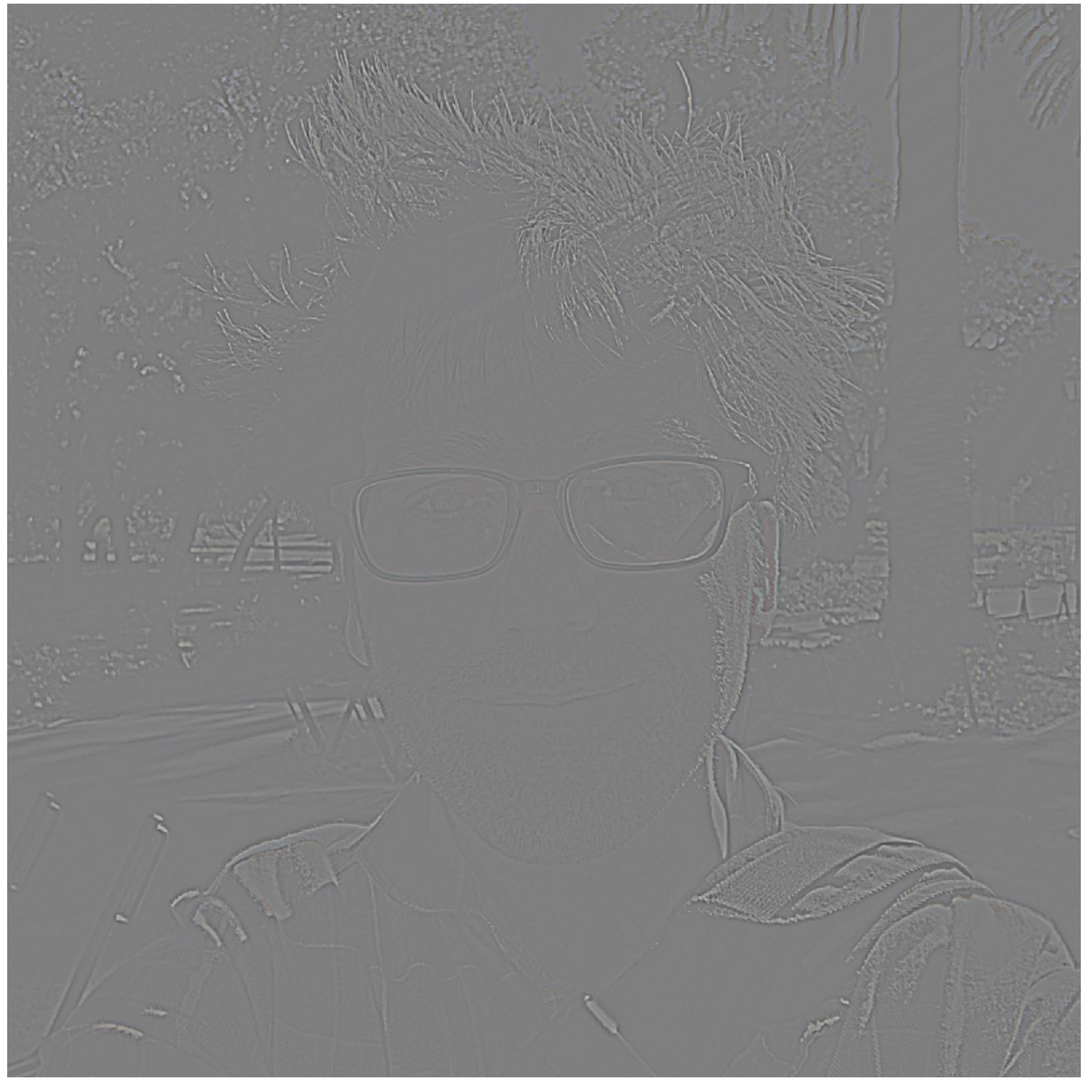
Question: how do you reconstruct original image from its Laplacian pyramid?

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

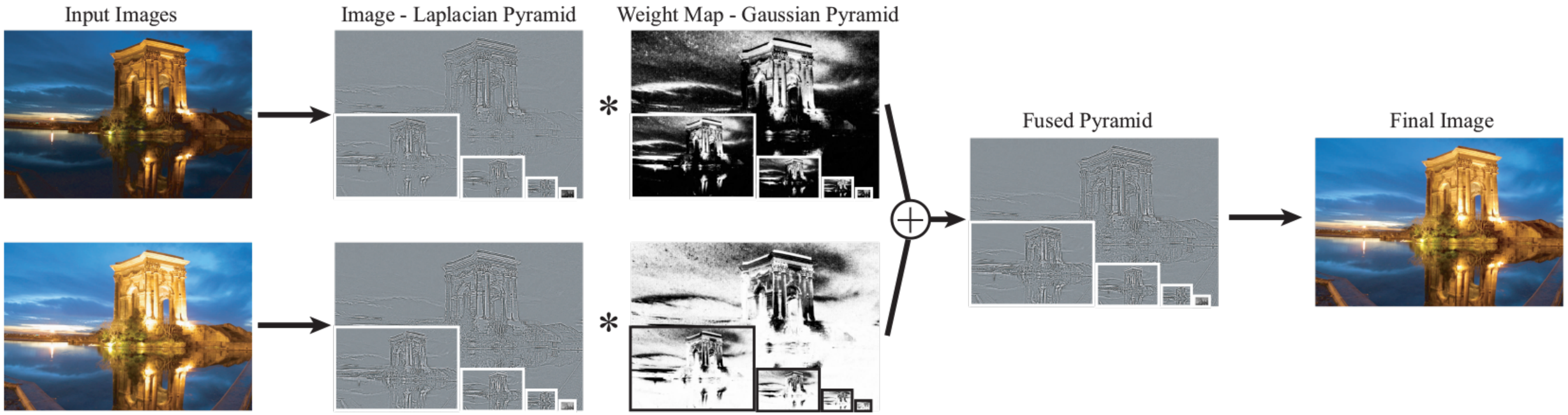
- $G_i(x,y)$ — frequencies up to limit given by *i*

- $L_i(x,y)$ — 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?

White balance

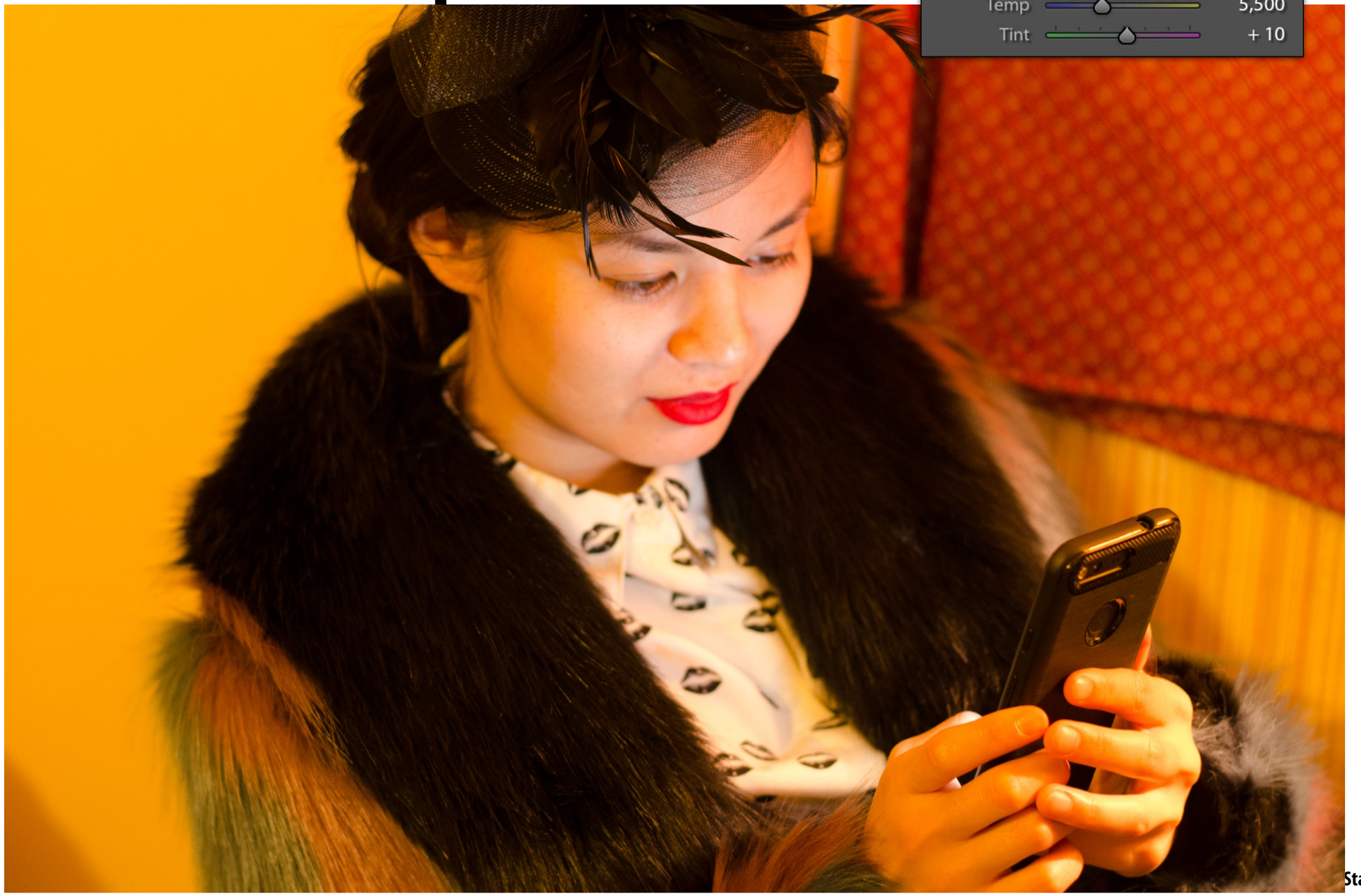
- Adjust relative intensity of rgb values (goal: make neutral tones in scene appear neutral in image)

```
output_pixel = white_balance_coeff * input_pixel
// note: in this example, white_balance_coeff is vec3
// (adjusts ratio of red-blue-green channels)
```

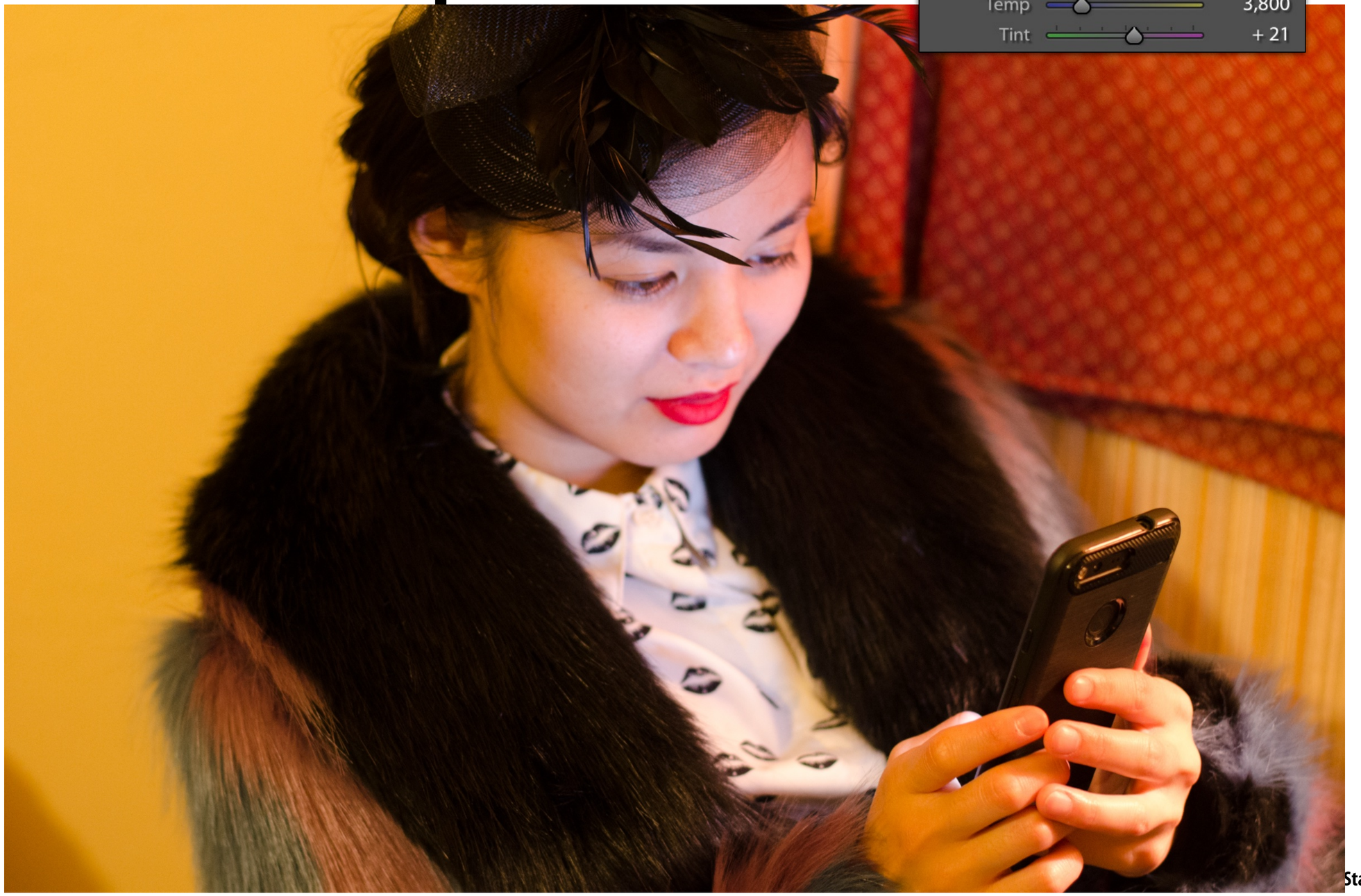
- The same "white" object will generate different sensor response when illuminated by different spectra. Camera needs to infer what the lighting in the scene was.

Image credit: basedigitalphotography.com

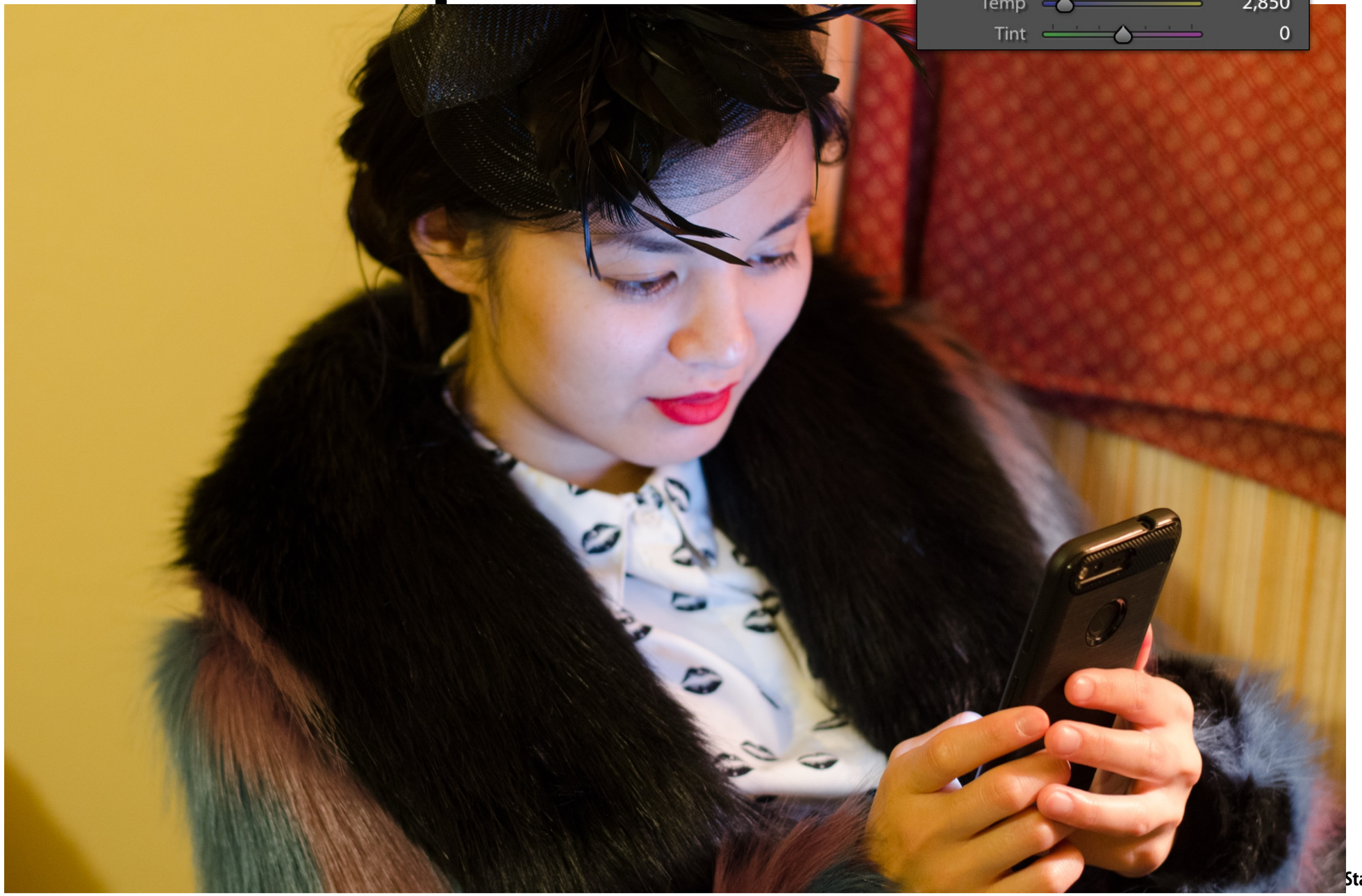
White balance example



White balance example



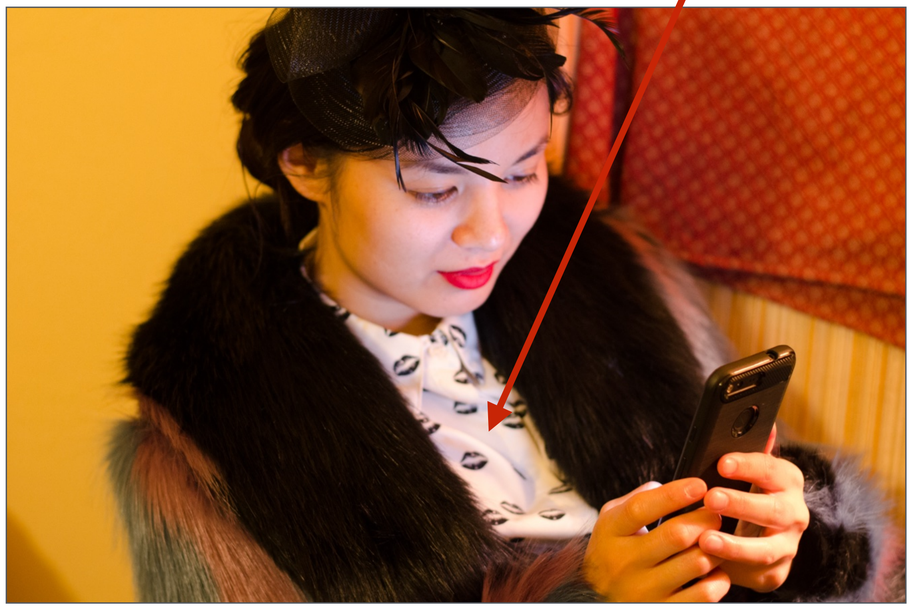
White balance example



White balance algorithms

- White balance coefficients depend on analysis of image contents
  - Calibration based: get value of pixel of "white" object: (r_w, g_w, b_w)
    - Scale all pixels by ($1/r_w$, $1/g_w$, $1/b_w$)
  - Heuristic based: camera must guesse which pixels correspond to white objects in scene
    - Gray world assumption: make average of all pixels in image gray
    - Brightest pixel assumption: find brightest region of image, make it white ([1,1,1])

- Modern white-balance algorithms are based on learning correct scaling from many "good photograph" examples
  - Create database of images for which good white balance settings are known (e.g., manually set by human)
  - Learn mapping from image features to white balance settings
  - When new photo is taken, use learned model to predict good white balance settings



Bokeh

Sharp foreground, defocused background

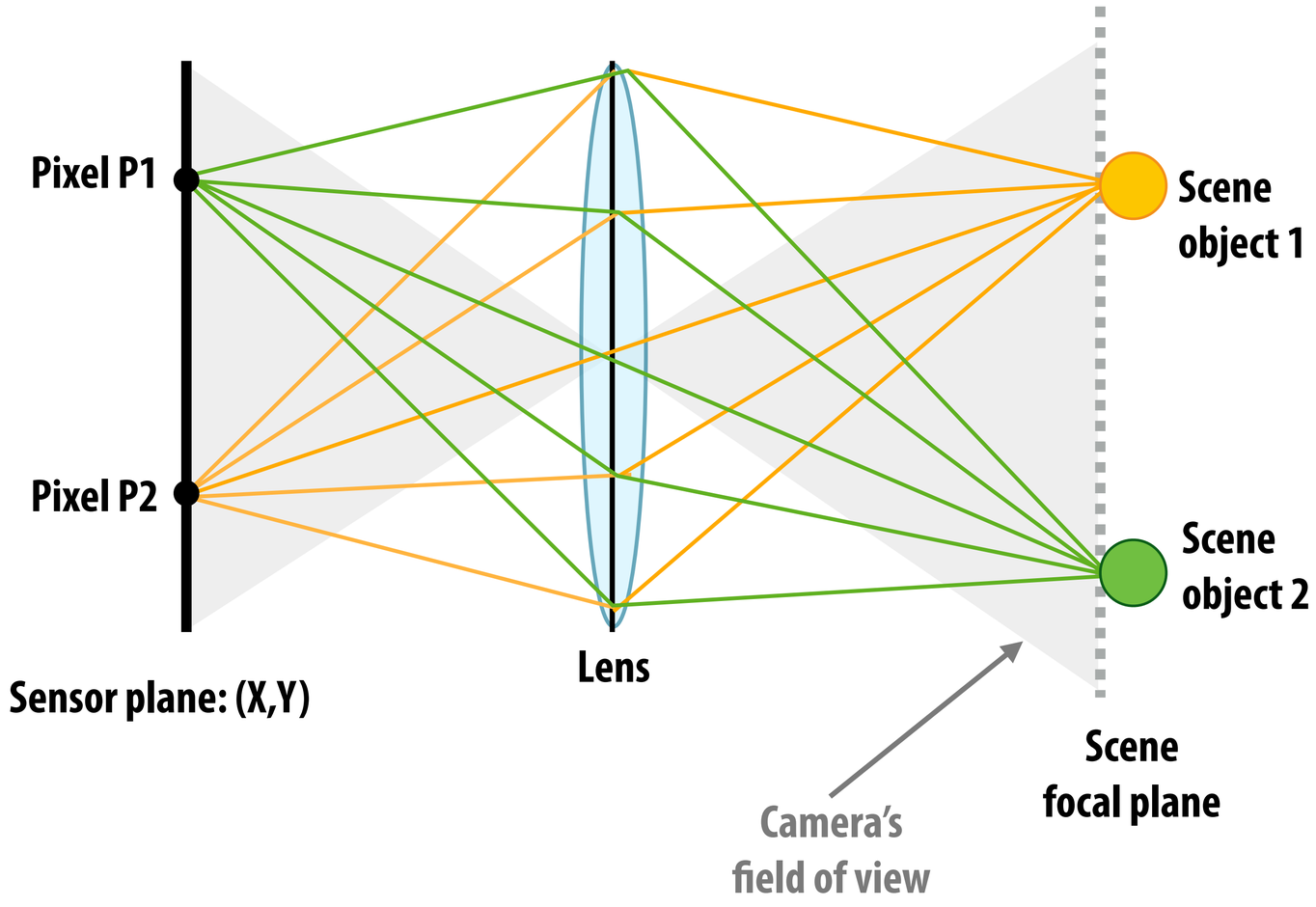
Common technique to emphasize subject in a photo

What does a lens do?

A lens refracts light.

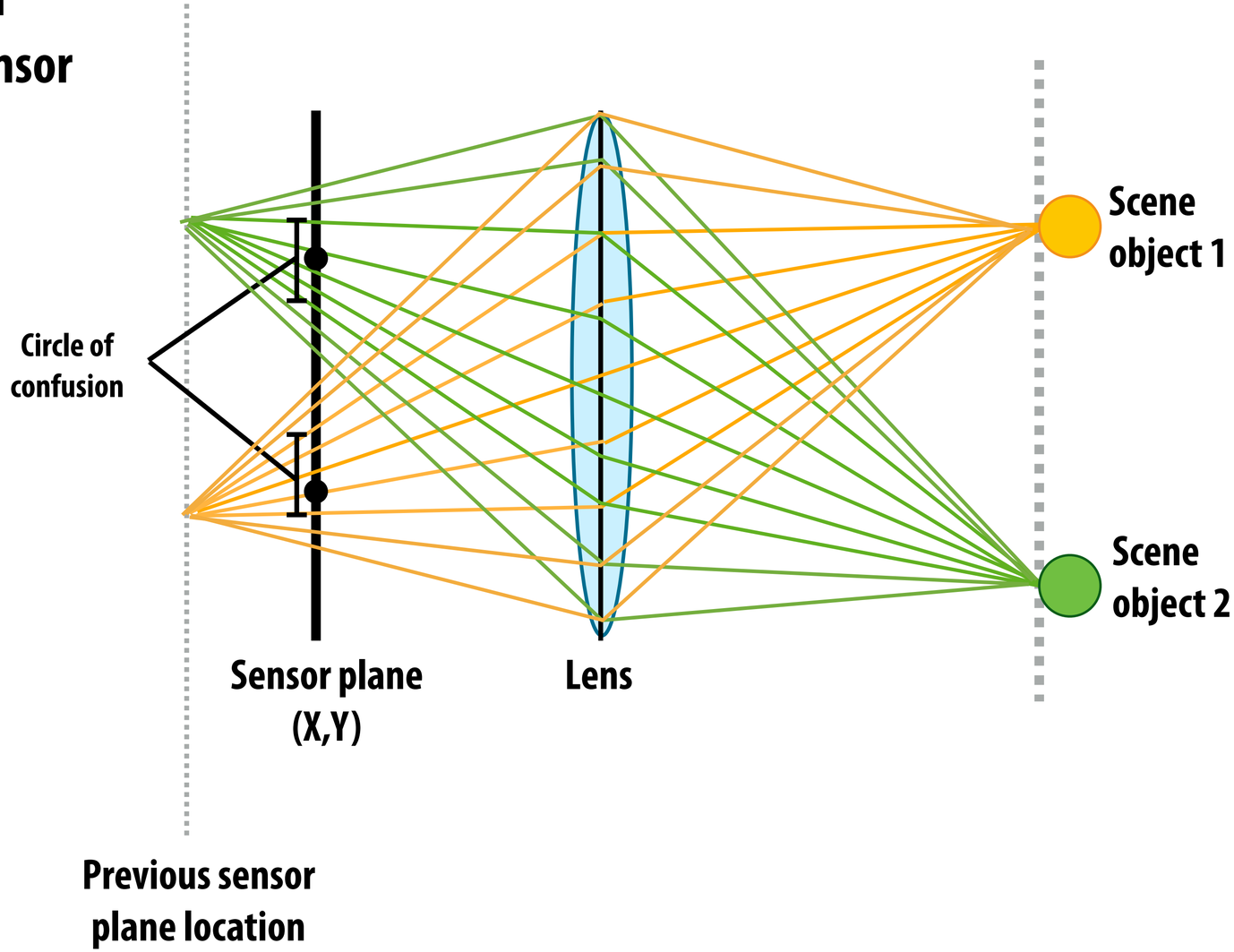
Camera with lens: every pixel accumulates all rays of light that pass through lens aperture and refract toward that pixel

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



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



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

Input image /w detected face

Segmentation

(c) Mask + disparity from DP

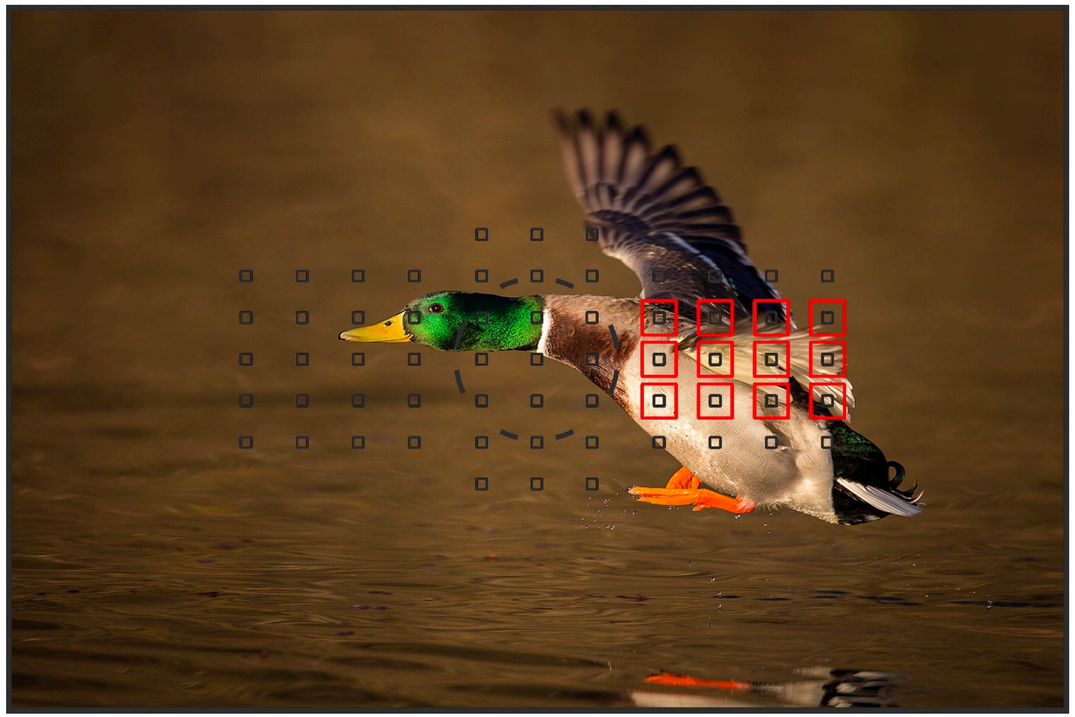
Scene Depth Estimate

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

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

Image credit: [Wadha 2018]

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

Estimating depth

Additional sensing modalities

Fuse information from all modalities to obtain best estimate of depth

iPhone Xr depth estimate
with lights ON in room

iPhone Xr depth estimate
with lights OFF in room
(No help from RGB)

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

Magic eraser

(Feature in recent Google Pixel phones)

Takeaways

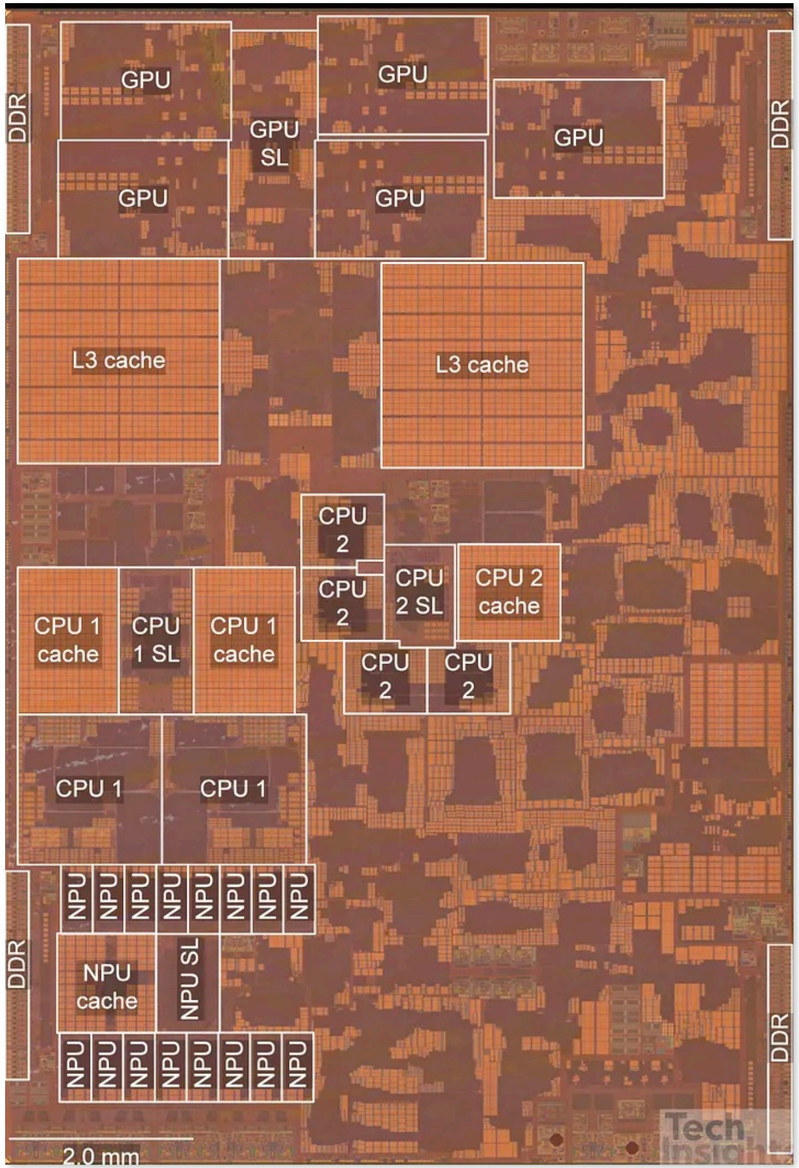
- The modern smartphone camera is a high performance computer graphics engine and a AI world interpreter

- Algorithms have to run at high frame rate on large resolutions
  - Viewfinder must run a real-time
  - Final photo generation must occur in a second or so. (Otherwise it's not ready to share)

- Processing must run on phone computing resources
  - Multi-core CPUs, GPUs, and increasing the neural engines of the smart phone

- Algorithms must work under an incredibly diverse range of real-world conditions

Specialized processing is ubiquitous in mobile systems



Apple A15 Bionic
(in iPhone 13, 14)

15 billion transistors

6-core GPU
- **2 "big" CPU cores**
- **4 "small" CPU cores**

Apple-designed multi-core GPU
Neural Engine (NPU) for DNN acceleration +
Image/video encode/decode processor +
Motion (sensor) processor

Image Credit: TechInsights Inc.

Acknowledgements

- Thanks and credit for slides to Ren Ng and Marc Levoy