Lecture 3:

Finishing up the Camera Pipeline + Frankencamera Discussion

Visual Computing Systems
Stanford CS348K, Spring 2024

Today

- Finish up description of algorithms for HDR+ pipeline (using slides from last lecture)
- Frankencamera discussion
- Modern AI-based camera pipeline features

Picking up from last time…
Finishing up the HDR+ pipeline

Frankencamera
(Discussion)

Choosing the “right” representation for the job

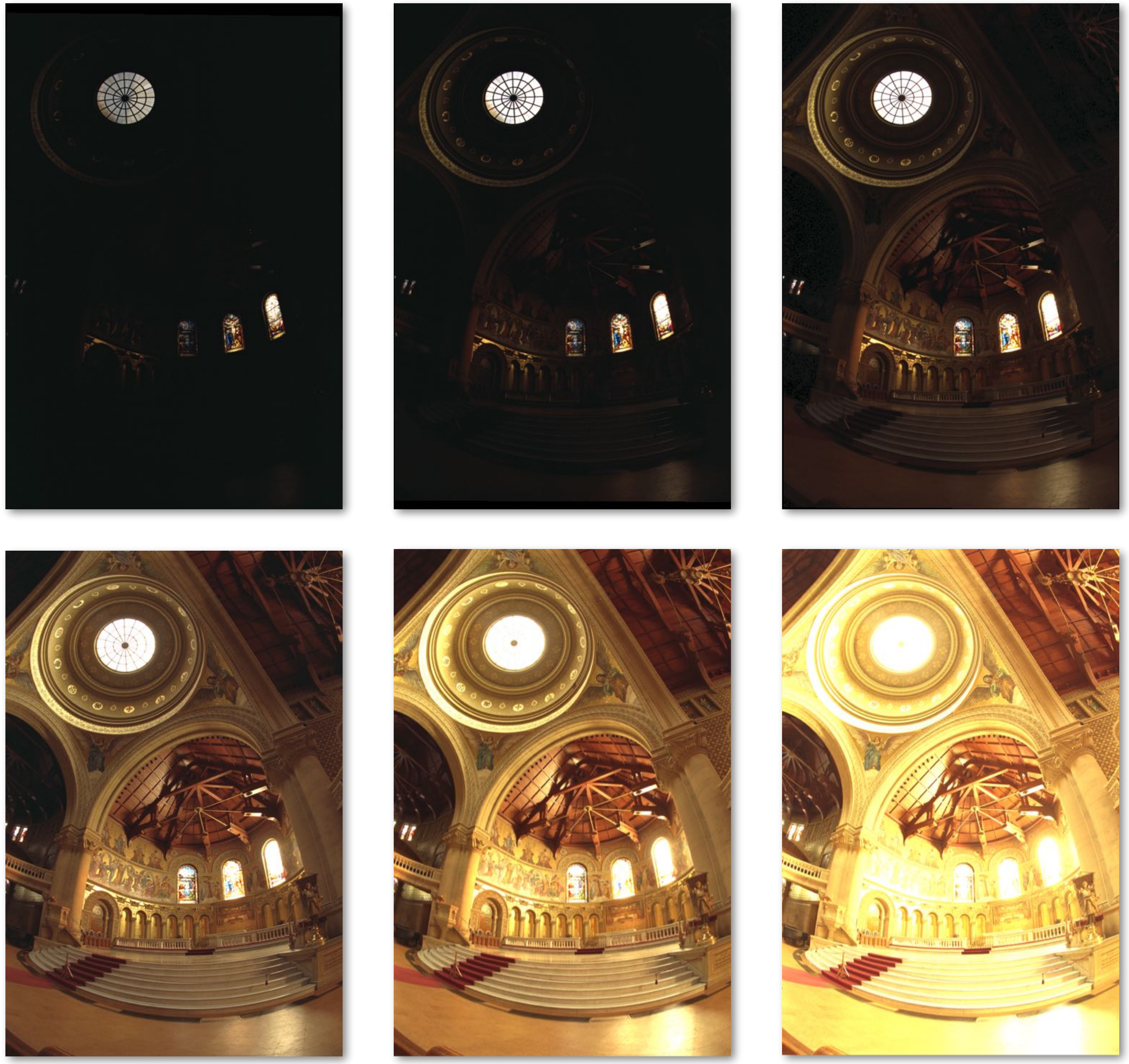
- **Good representations are productive to use:**
  - They embody the natural way of thinking about a problem

- **Good representations enable the system to provide the application developer useful services:**
  - Validating/providing certain guarantees (correctness, resource bounds, conversation of quantities, type checking)
  - Performance optimizations (parallelization, vectorization, use of specialized hardware)
  - Implementations of common, difficult-to-implement functionality (texture mapping and rasterization in 3D graphics, auto-differentiation in ML frameworks)

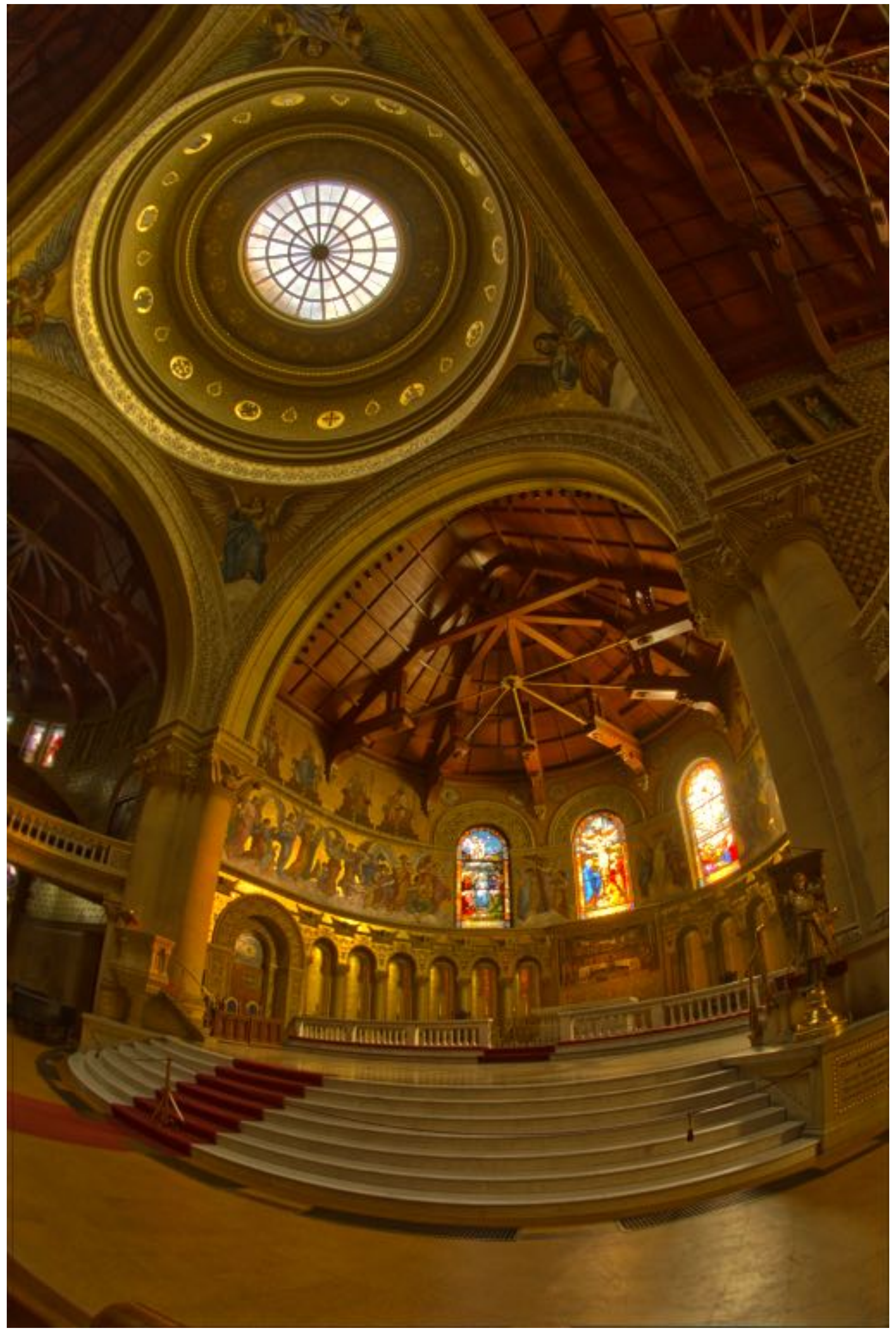
Frankencamera: some 2010 context

- Cameras were becoming increasingly cheap and ubiquitous
- Cameras featured increasing processing capability
- Significant graphics research focus on developing techniques for combining multiple photos to overcome deficiencies of traditional camera systems

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



Source photographs: each photograph has different exposure



Tone mapped HDR image

Credit: Debevec and Malik

More multi-shot photography examples

"Lucky" imaging

Take several photos in rapid succession: likely to find one without camera shake

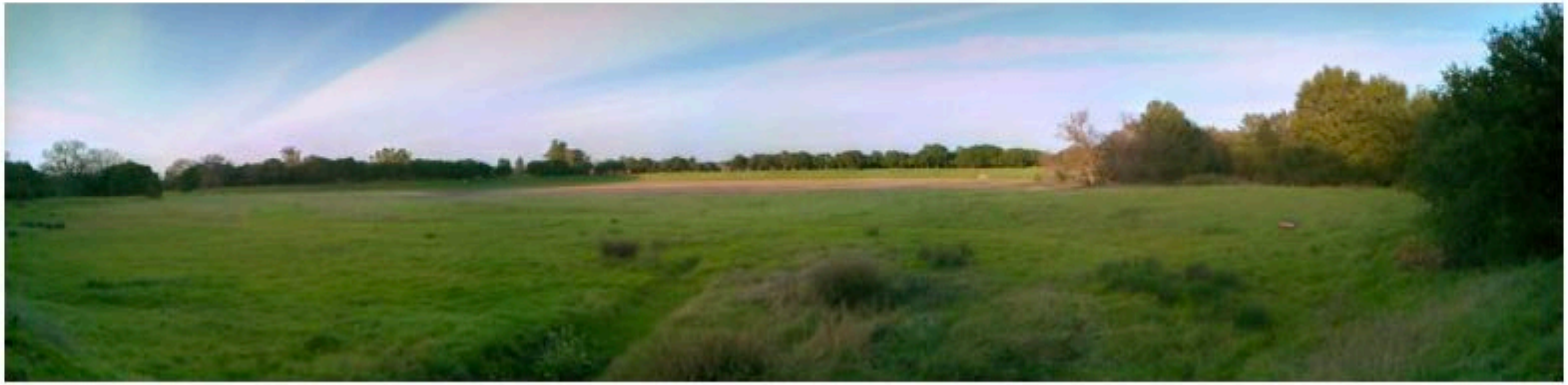
no-flash flash result

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

More multi-shot photography examples

Panorama capture

individual images



extended dynamic range panorama

Frankencamera: some 2010 context

- Cameras were becoming increasingly cheap and ubiquitous
- Cameras featured increasing processing capability
- Significant graphics research focus on developing techniques for combining multiple photos to overcome deficiencies of traditional camera systems

- Problem: the ability to implement multi-shot techniques on cameras was limited by camera system programming abstractions
  - Programmable interface to camera was very basic
  - Echoed physical button interface to a point-and-shoot camera:
    - `take_photograph(parameters, output_jpg_buffer)`
  - Result: on most camera implementations, latency between two photos was high, mitigating utility of multi-shot techniques (large scene movement or camera shake between shots)

Frankencamera (F-cam) goals

[Adams et al. 2010]

1. Create open, handheld computational camera platform for researchers

2. Define system architecture for computational photography applications
   - Motivated by impact of OpenGL on graphics application and graphics hardware development (portable apps despite highly optimized GPU implementations)
   - Motivated by proliferation of smart-phone apps

F2 Reference Implementation

Note: Apple was not involved in Frankencamera's industrial design. ;-)

Nokia N900 Smartphone Implementation

F-cam scope

- F-cam provides a set of abstractions that allow for manipulating configurable camera components
  - Timeline-based specification of actions
  - Feed-forward system: no feedback loops

- F-cam architecture performs image processing, but...
  - This functionality as presented by the architecture is <u>not programmable</u>
  - Hence, F-cam does not provide an image processing language (it's like fixed-function OpenGL)
  - Other than work performed by the image processing stage, F-cam applications perform their own image processing (e.g., on smartphone/camera's CPU or GPU resources)

Android Camera2 API

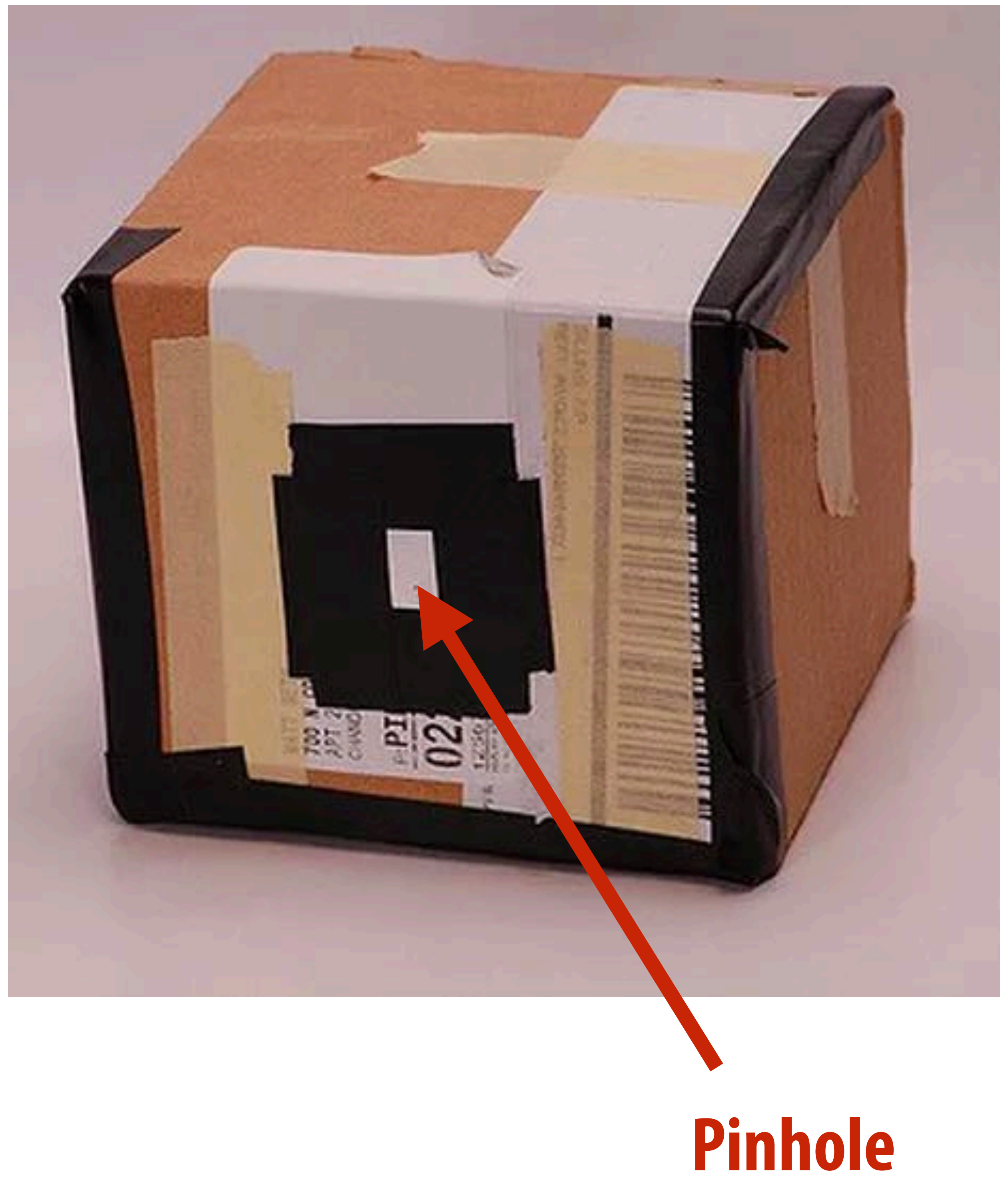
- **Take a look at the documentation of the Android Camera2 API, and you'll see influence of F-Cam.**

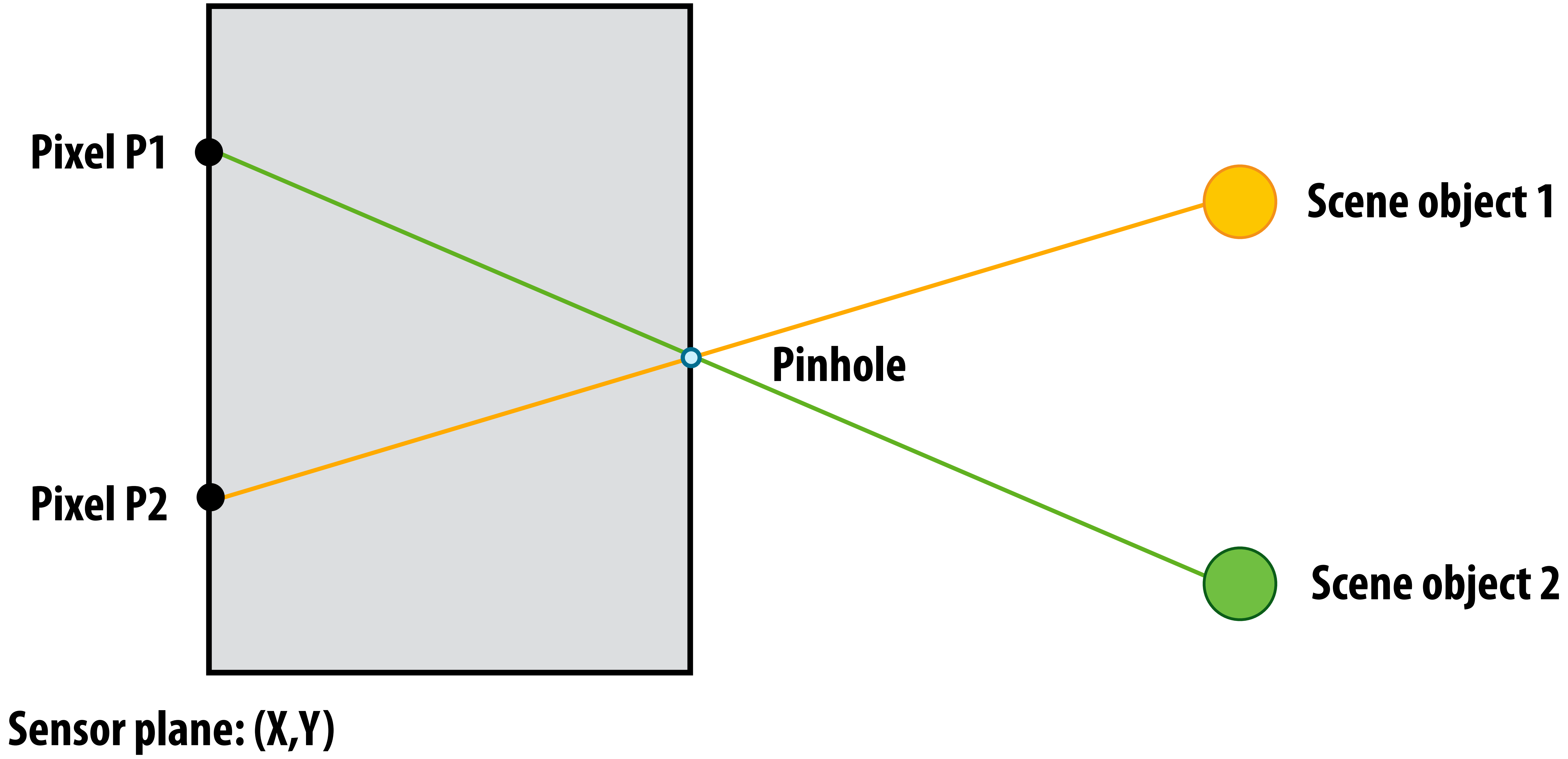
Modern smartphone cameras perform advanced image analysis functions

Image analysis examples from prior lectures:
auto white balance, auto exposure, image denoising

Auto Focus

Pinhole camera (no lens)



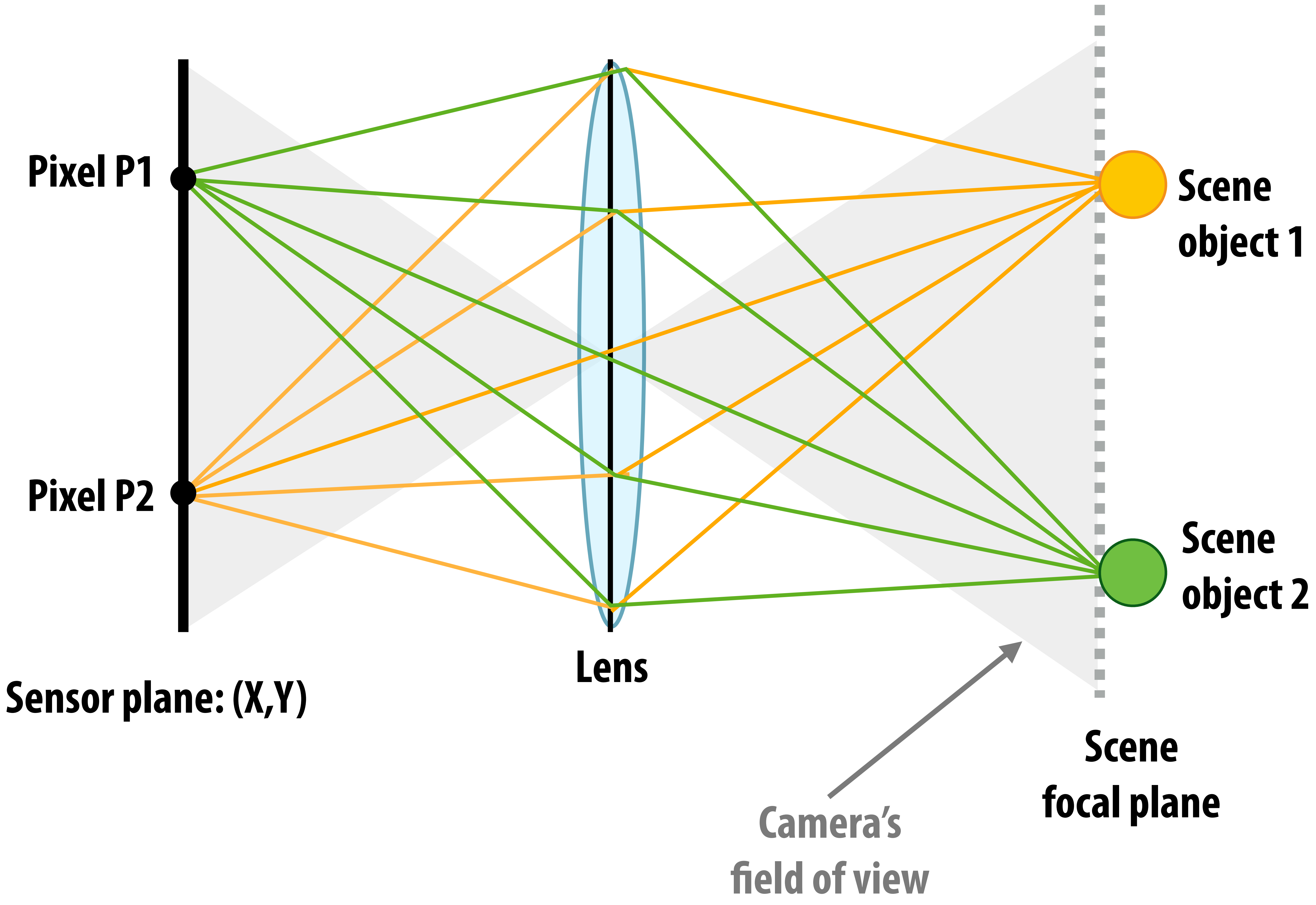


What does a lens do?

A lens refracts light.

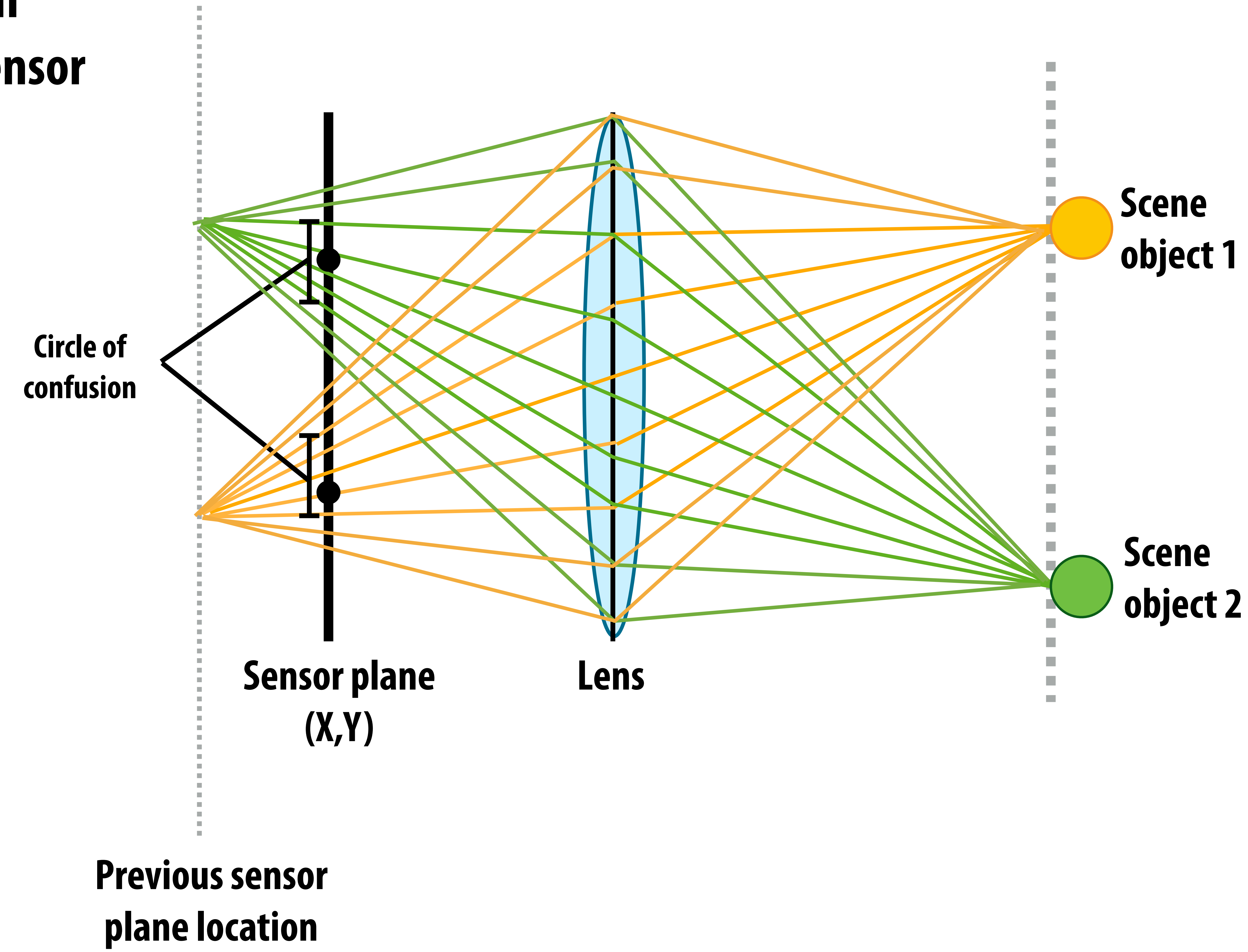
Camera with lens: every pixel accumulates all rays of light that pass through lens aperture and refract toward that pixel

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



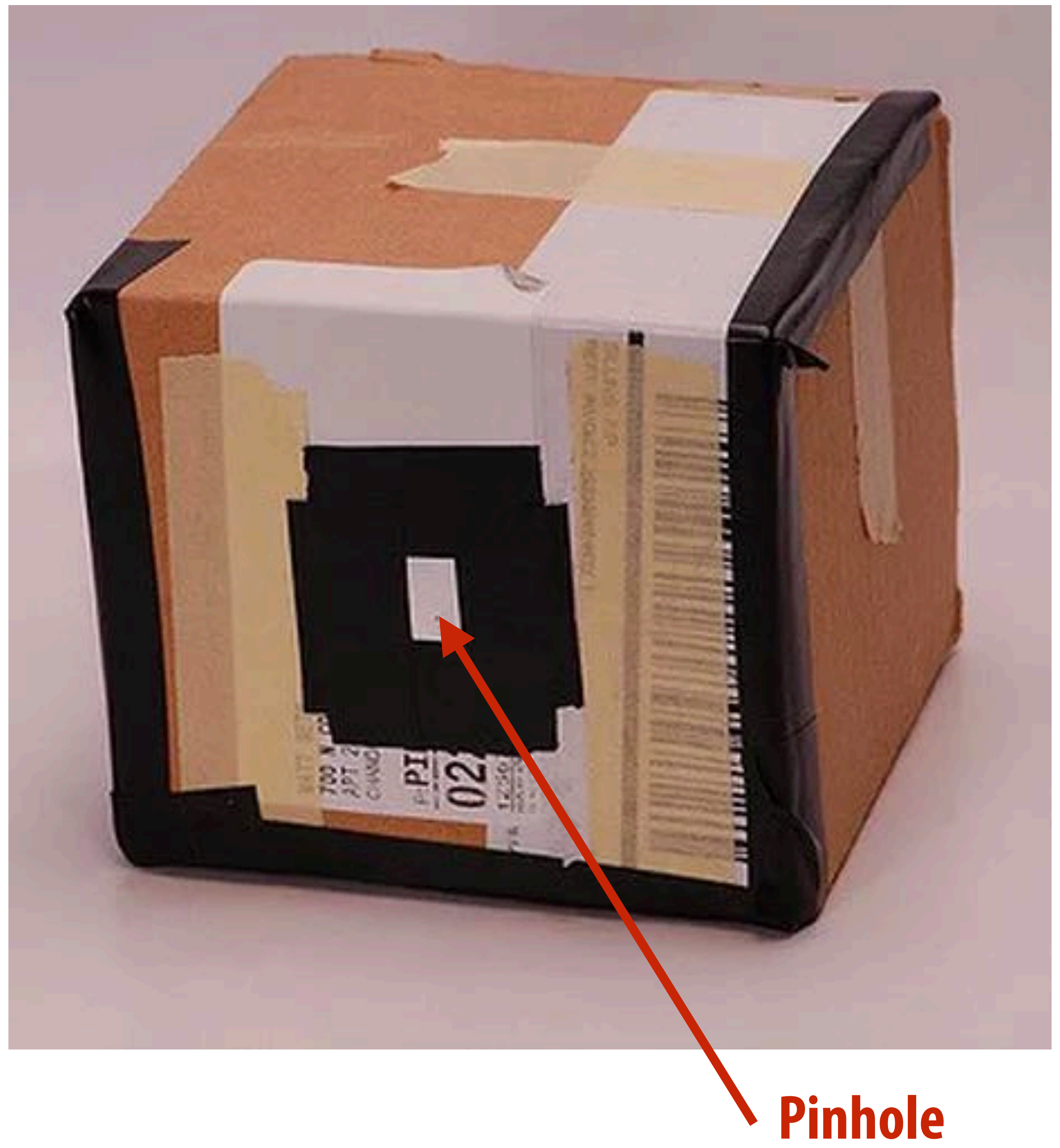
Out of focus camera

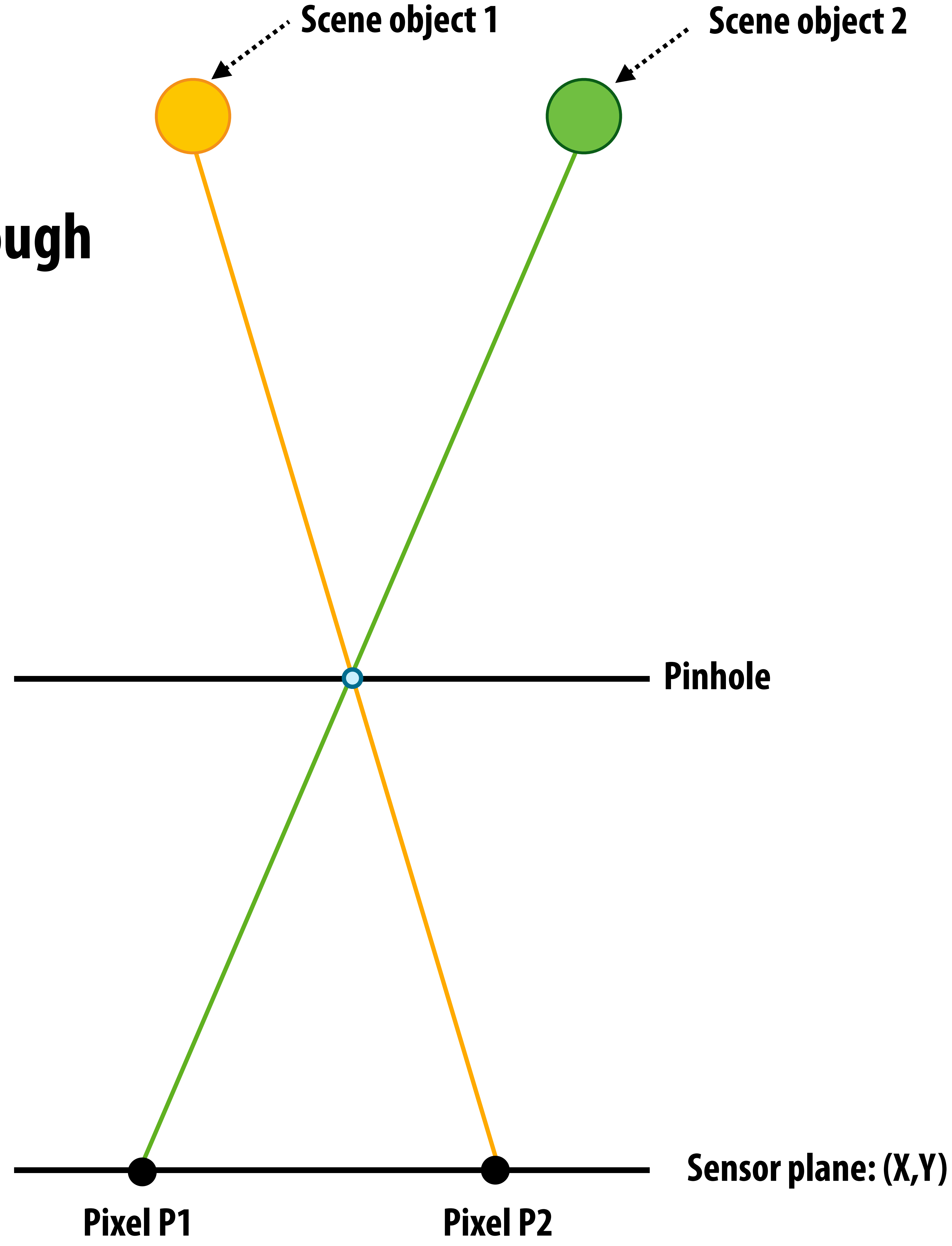
Out of focus camera: rays of light from one point in scene do not converge to the same point on the sensor



What does a lens do?

Recall: pinhole camera you may have made in science class
(every pixel measures ray of light passing through pinhole and arriving at pixel)





Bokeh

Sharp foreground, defocused background

Common technique to emphasize subject in a photo

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

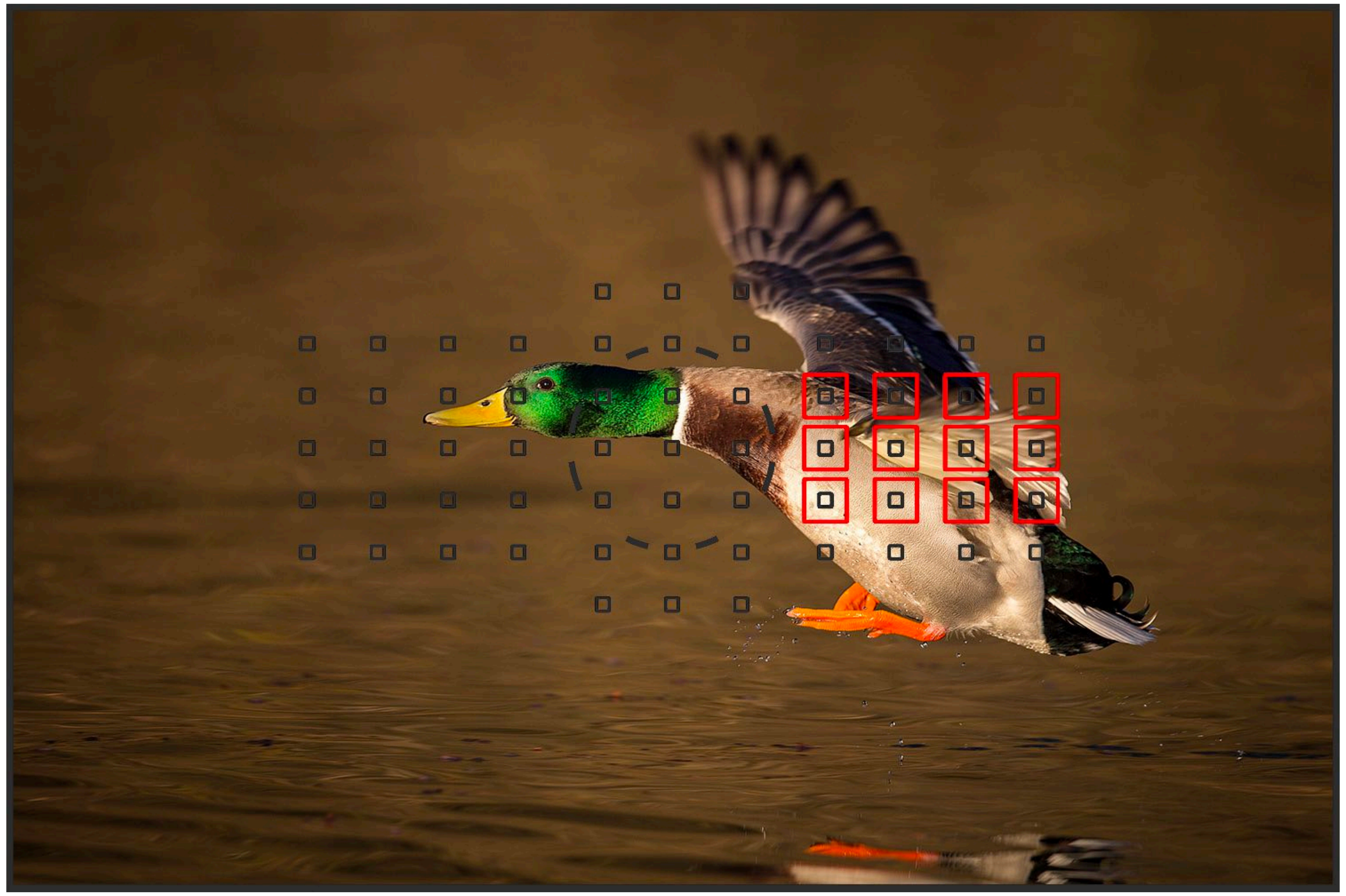
Input image /w detected face

Scene Depth Estimate

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

Image credit: [Wadha 2018]

What part of image should be in focus?



Consider possible heuristics:

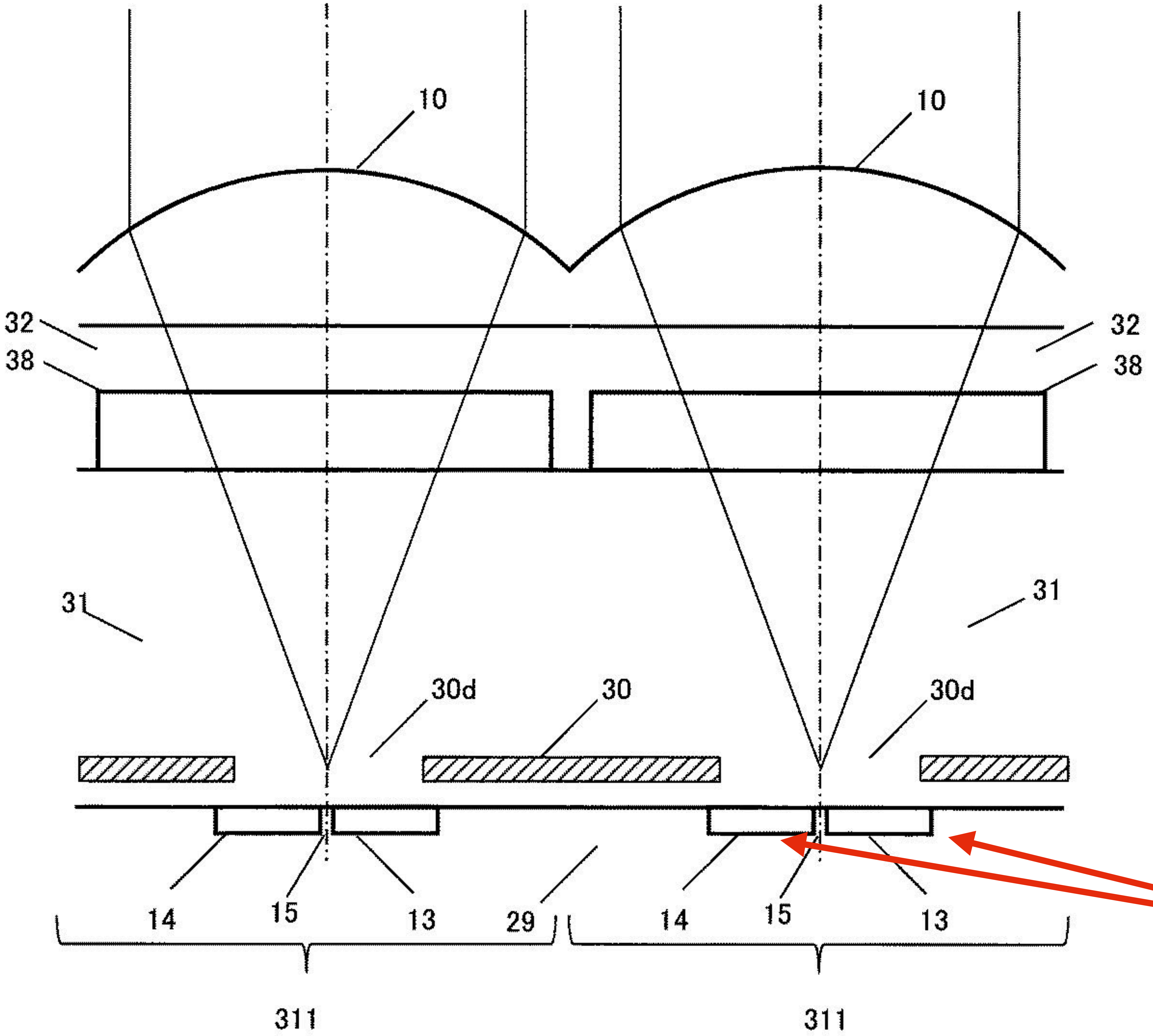
Focus on closest scene region

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

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

Estimating depth

Additional sensing modalities

Fuse information from all modalities to obtain best estimate of depth

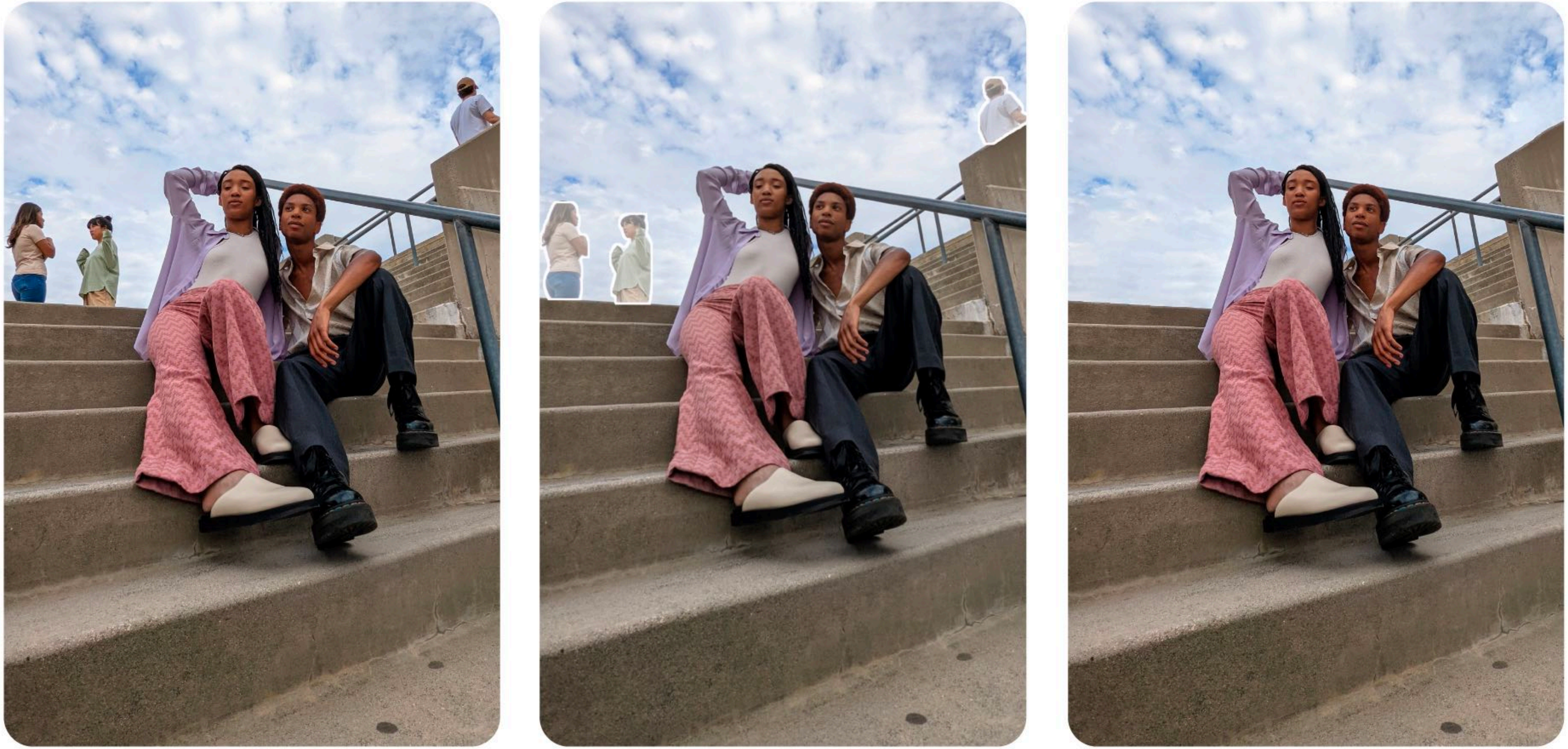
iPhone Xr depth estimate with lights ON in room

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)



Summary

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

Sensor output
("RAW")

Computation

Beautiful image that impresses your friends on Instagram

Image processing workload characteristics

- "Pointwise" operations
  - output_pixel = f(input_pixel)
- "Stencil" computations (e.g., convolution, demosaic, etc.)
  - Output pixel (x,y) depends on <u>fixed-size</u> local region of input around (x,y)
- Lookup tables
  - e.g., contrast s-curve
- Multi-resolution operations (upsampling/downsampling)
- Fast-Fourier transforms
  - We didn't talk about Fourier domain techniques in class (but Hasinoff 16 reading has many examples)
- Long pipelines of these operations

Next class: efficiently mapping these workloads to modern processors

Abstractions for authoring image processing pipelines

Choosing the “right” representation for the job (again)

- This was the theme of our Frankencamera discussion

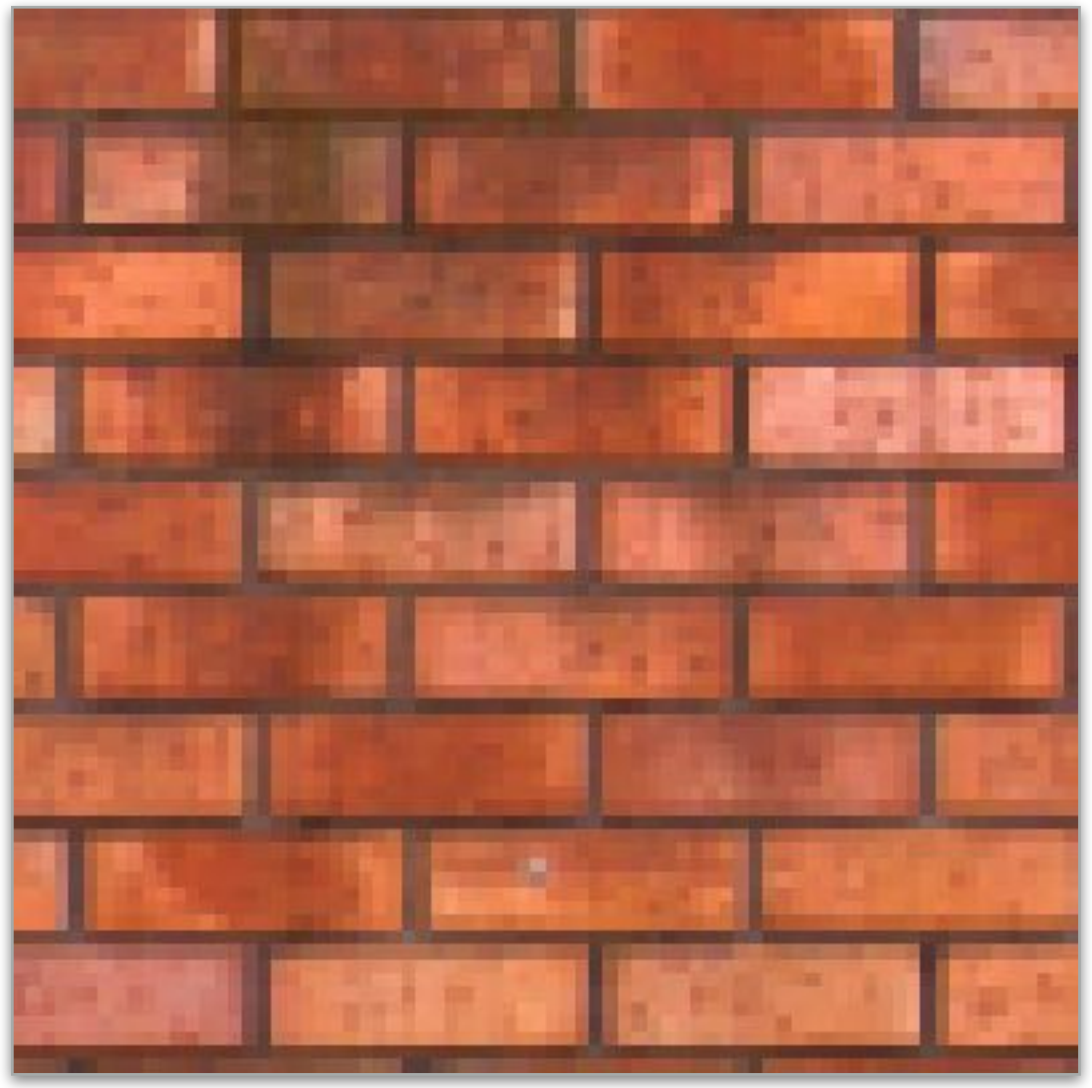
- Good representations are productive to use:
  - They embody the natural way of thinking about a problem

- Good representations enable the system to provide the application developer useful services:
  - Validating/providing certain guarantees (correctness, resource bounds, conversation of quantities, type checking)
  - Performance optimizations (parallelization, vectorization, use of specialized hardware)
  - Implementations of common, difficult-to-implement functionality (texture mapping and rasterization in 3D graphics, auto-differentiation in ML frameworks)

Consider a single task: sharpen an image

Example: sharpen an image

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



Input

Output

Four different representations of sharpen

1

```
Image input;
Image output = sharpen(input);
```

2

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

```
Image input;
Image output = convolve(input, F);
```

3

```
Image input;
Image output;
output[i][j]
    = F[0][0] * input[i-1][j-1] +
      F[0][1] * input[i-1][j]   +
      F[0][2] * input[i-1][j+1] +
      F[1][0] * input[i][j-1]   +
      F[1][1] * input[i][j]     +
      ...
```

4

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

float weights[] = {0., -1., 0.,
                   -1., 5, -1.,
                   0., -1., 0.};

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;
  }
}
```

Image processing tasks from previous lectures

Sobel Edge Detection

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

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

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

3x3 Gaussian blur

$$F = \begin{bmatrix} .075 & .124 & .075 \\ .124 & .204 & .124 \\ .075 & .124 & .075 \end{bmatrix}$$

2x2 downsample (via averaging)

```
output[x][y] = (input[2x][2y]   + input[2x+1][2y] +
                input[2x][2y+1] + input[2x+1][2y+1]) / 4.f;
```

Gamma Correction

```
output[x][y] = pow(input[x][y], 0.5f);
```

LUT-based correction

```
output[x][y] = lookup_table[input[x][y]];
```

Local Pixel Clamp

```
float f(image input) {
   float min_value = min( min(input[x-1][y], input[x+1][y]),
                          min(input[x][y-1], input[x][y+1]) );
   float max_value = max( max(input[x-1][y], input[x+1][y]),
                          max(input[x][y-1], input[x][y+1]) );
output[x][y] = clamp(min_value, max_value, input[x][y]);
output[x][y] = f(input);
```

Histogram

```
bin[input[x][y]]++;
```

New goals (setting up for next class)

- Be expressive: facilitate intuitive expression of a broad class of image processing applications
  - e.g., all the components of a modern camera RAW pipeline

- Be high performance: want to generate code that efficiently utilizes the multi-core and SIMD processing resources of modern CPUs and GPUs, and is memory bandwidth efficient