Reconstruction

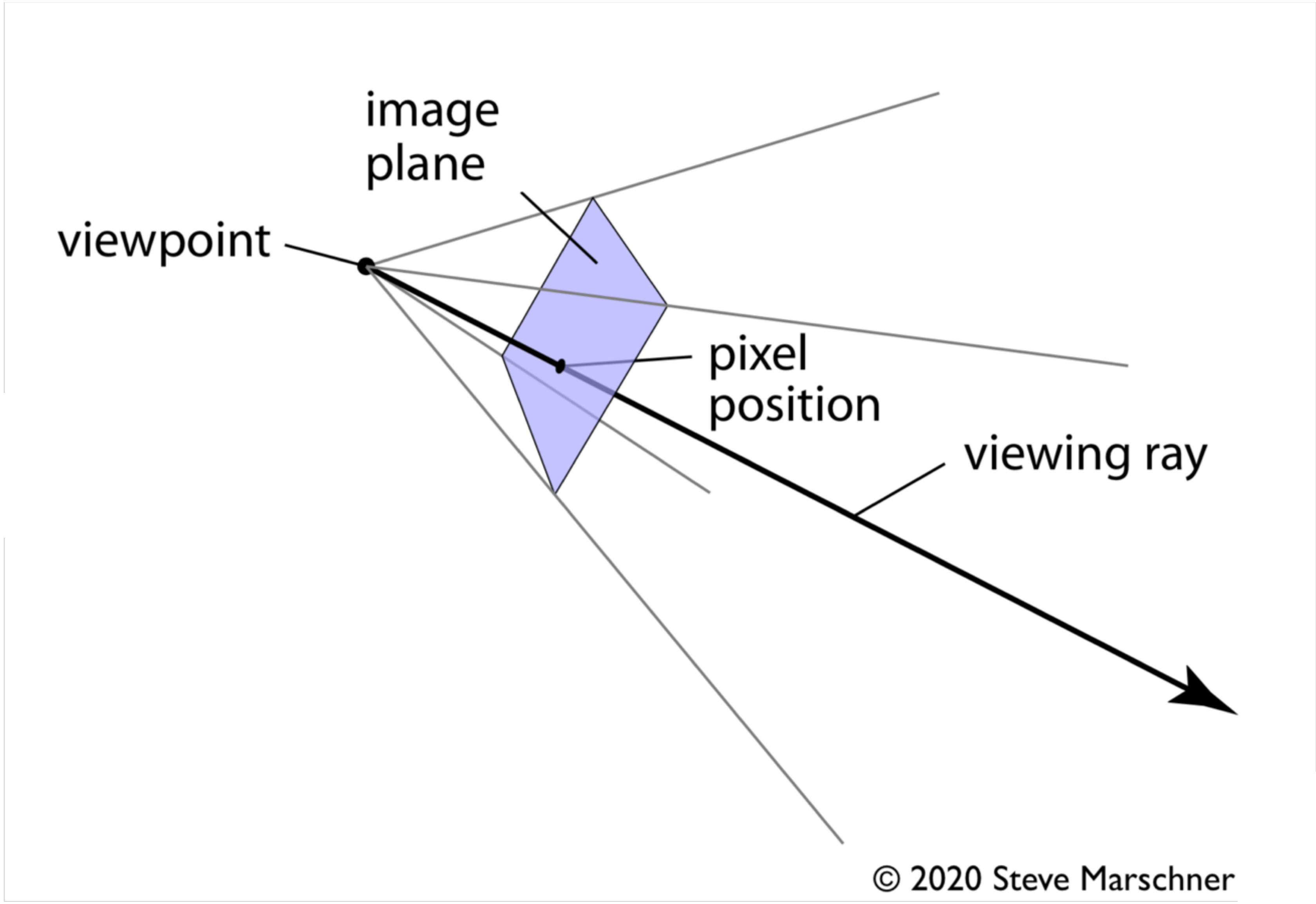
Reconstruction

- Given an image, can we reconstruct the 3D world that created the image?

Why is reconstruction hard?

- Perspective projection
  - $x = \frac{fX}{Z} + p_x, y = \frac{fY}{Z} + p_y$
- Simple case: $f = 1, p_x = p_y = 0$
  - $x = \frac{X}{Z}$
  - $y = \frac{Y}{Z}$
- (X, Y, Z) and $(\lambda X, \lambda Y, \lambda Z)$ project to the same point!
  - "Ill-posed problem"

Why is reconstruction hard?



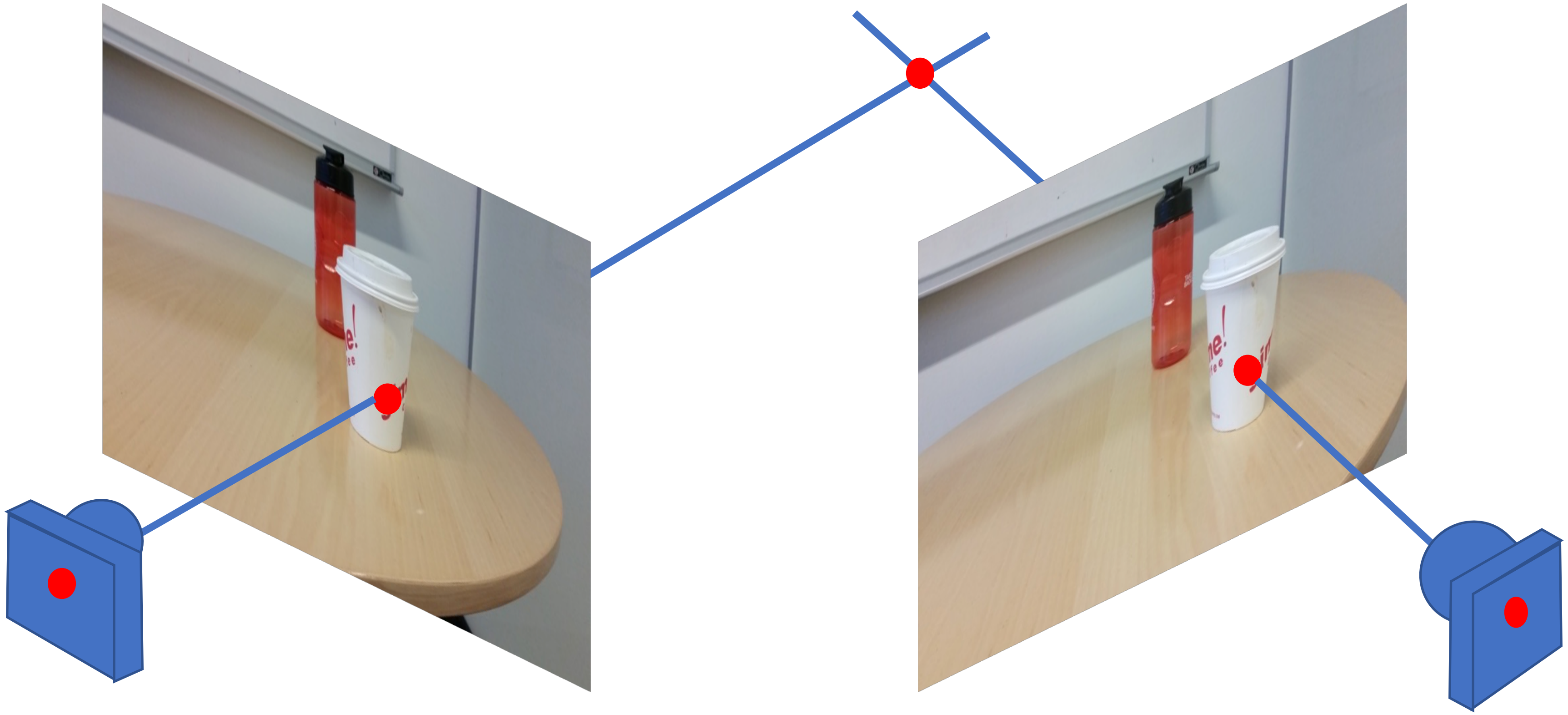
One way out: multiple images

- Multiple images can give a clue about 3D structure

One way out: multiple images

- Parallax: nearby objects move more than far away objects

One way out: multiple images

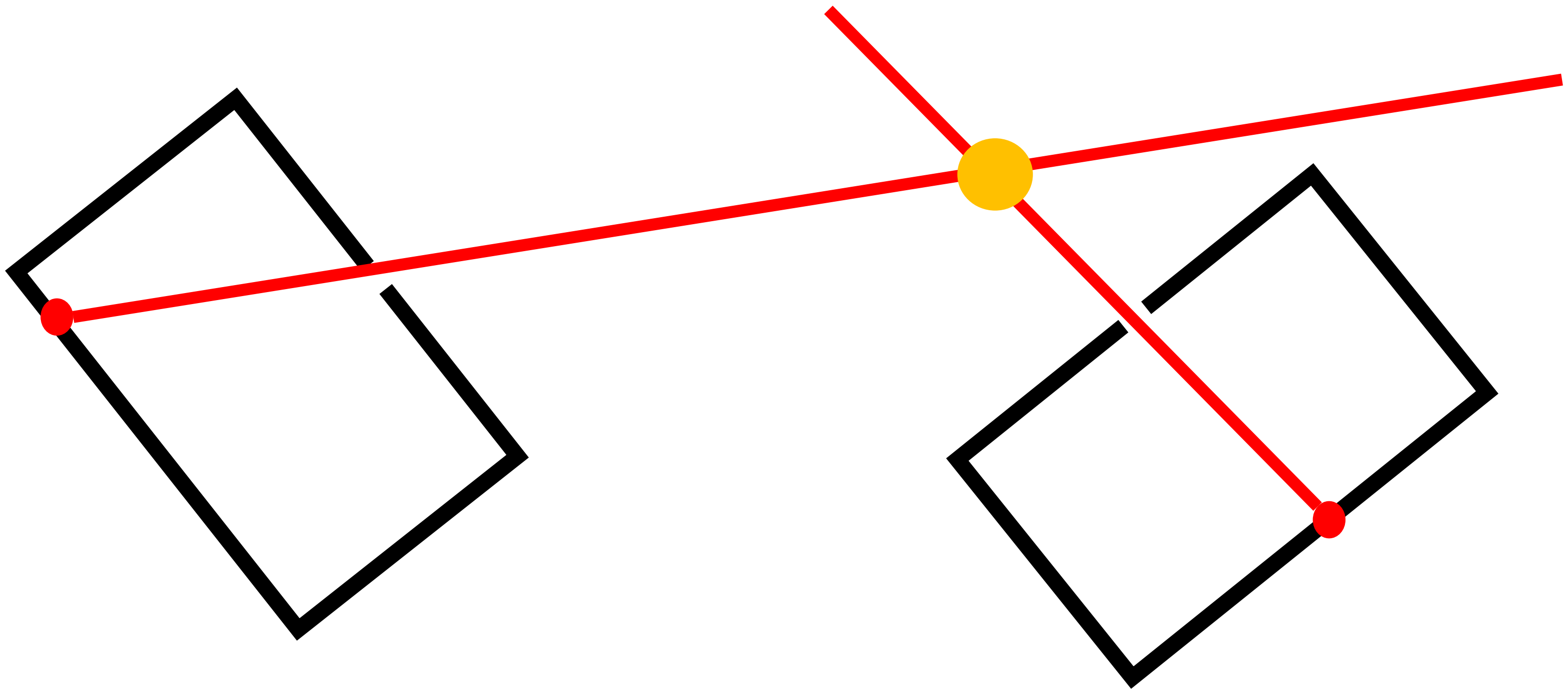


One way out: multiple images

- Step 1: Need to find *correspondences* between pixels in image 1 and image 2
- Step 2: Use correspondences to locate point in 3D

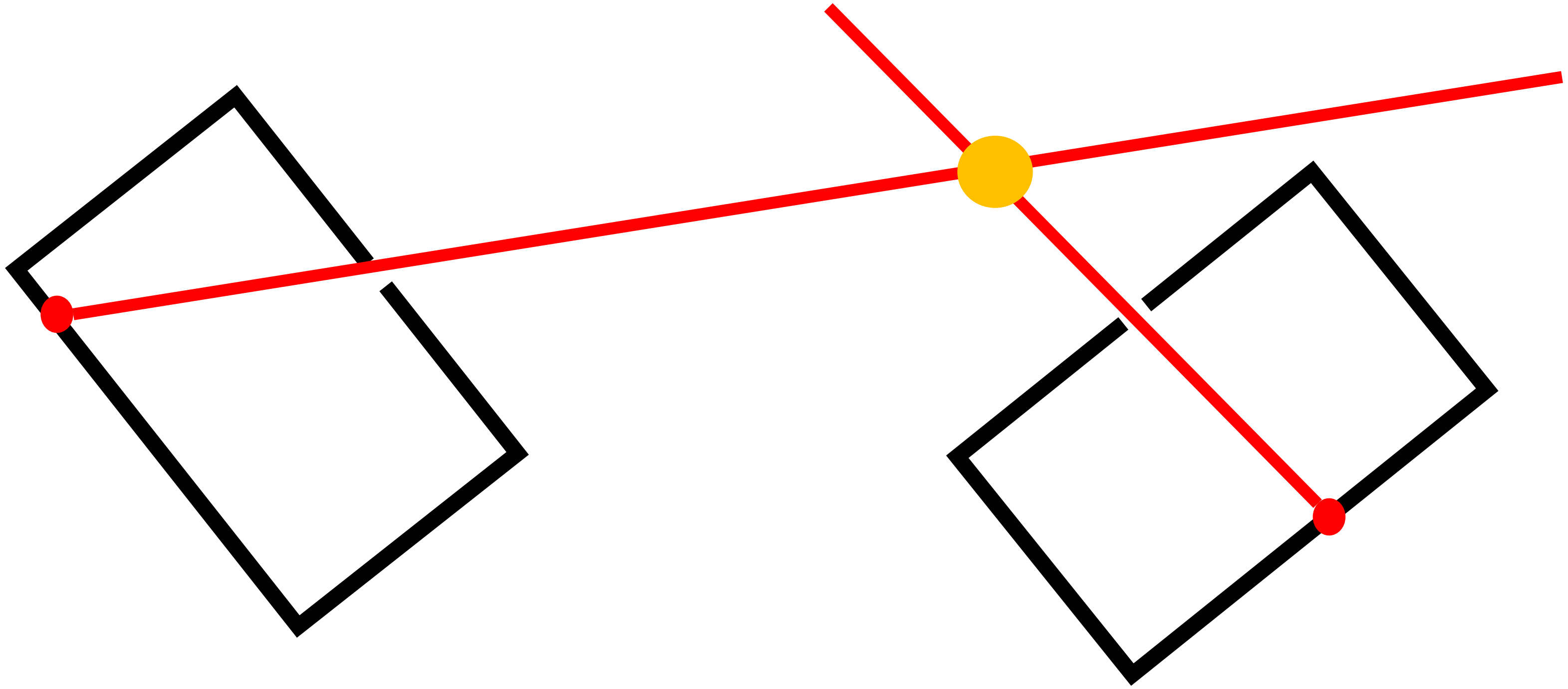
Reconstruction from correspondence

- Given known cameras, correspondence gives the location of 3D point (*Triangulation*)



Reconstruction from correspondence

- Given a 3D point, correspondence gives relationship between cameras (*Pose estimation / camera calibration*)



Next few classes

- How do we find correspondences?
- How do we use correspondences to reconstruct 3D?

Other applications of correspondence

- Image alignment
- Motion tracking
- Robot navigation

Easy correspondence

by [Diva Sian]

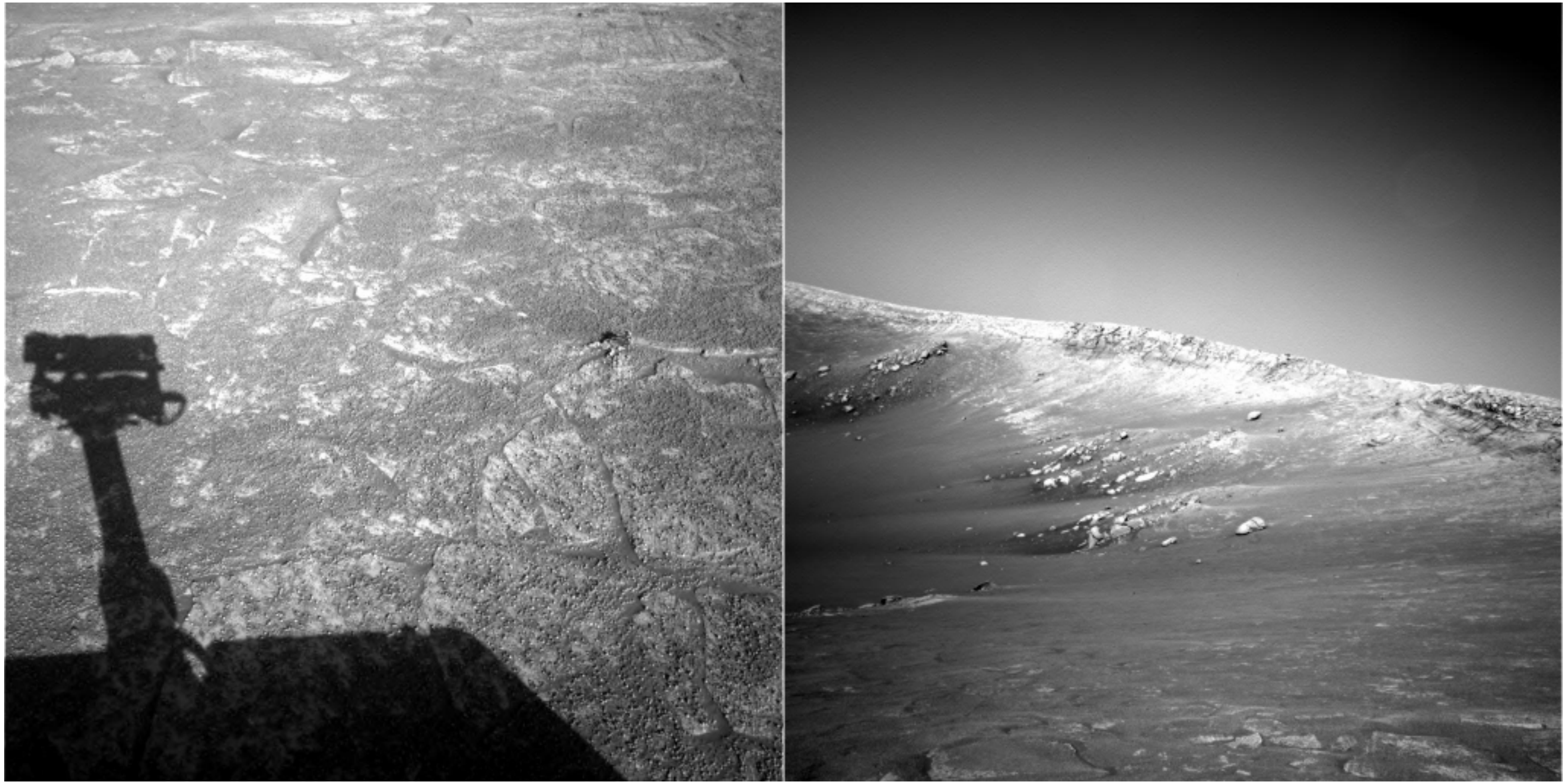
by [swashford]

Harder case

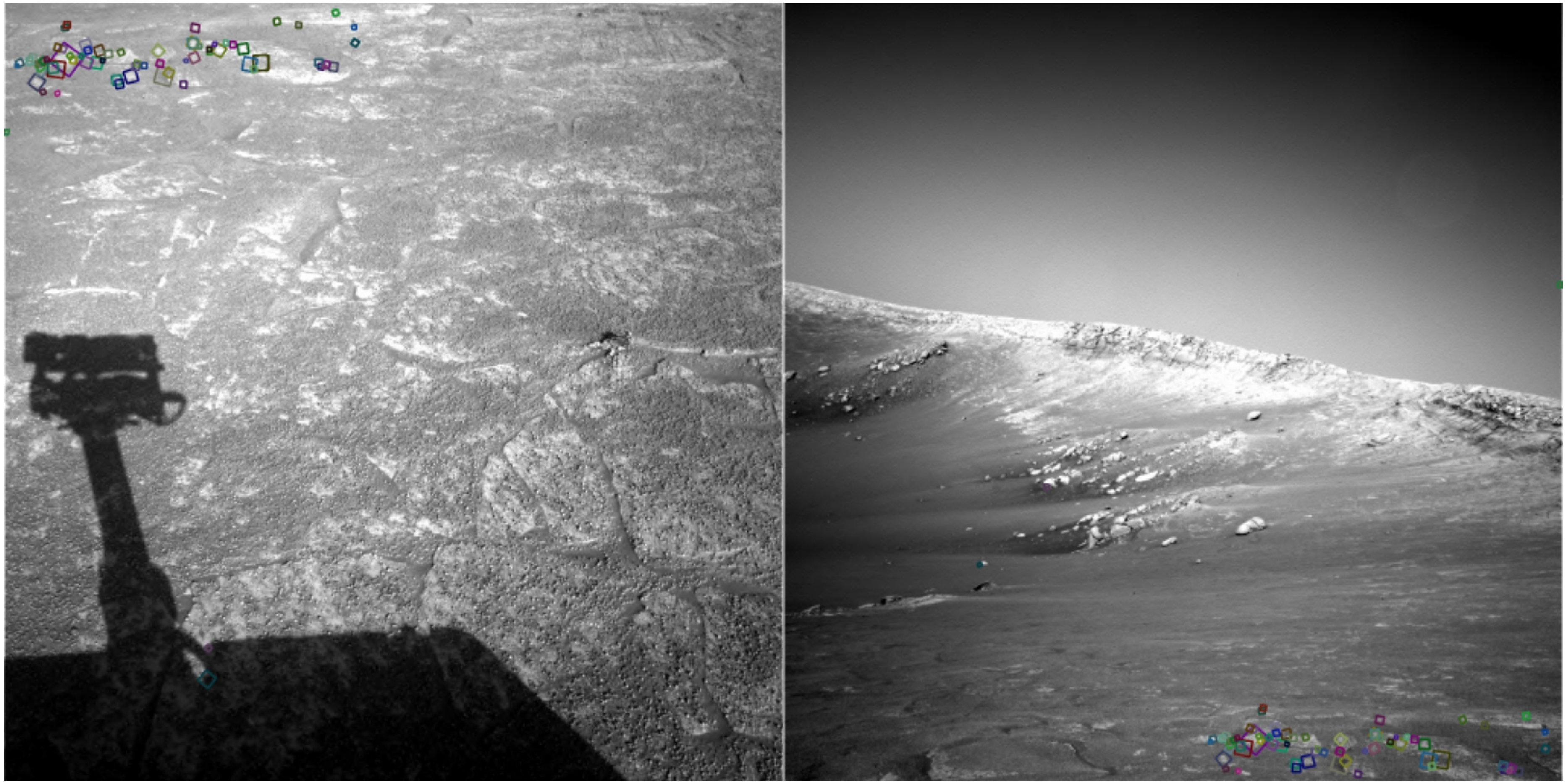
by Diva Sian

by scgbt

Harder still?



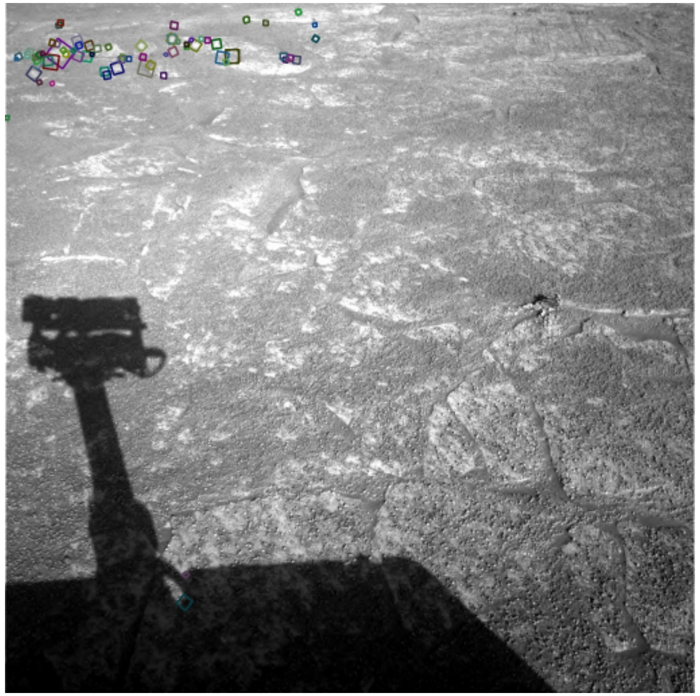
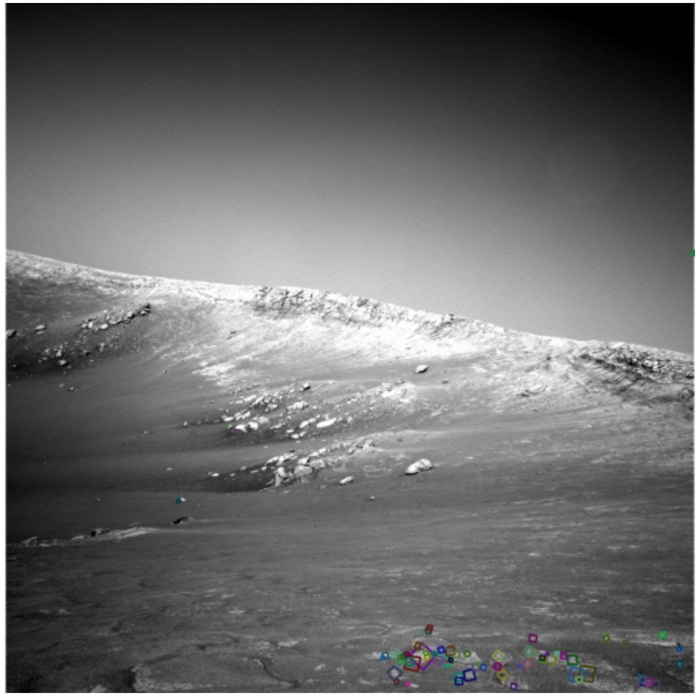
Answer below (look for tiny colored squares…)



NASA Mars Rover images
with SIFT feature matches

Sparse vs dense correspondence

- Sparse correspondence: produce a few, high confidence matches
  - Good enough for estimating pose or relationship between cameras
  - Easier
- Dense correspondence: try to match every pixel
  - Needed if we want 3D location of every pixel

A general pipeline for correspondence

1. **Feature detection:** If sparse correspondences are enough, *choose points for which we will search for correspondences (feature points)*
2. **Feature description:** For each point (or every pixel if dense correspondence), describe point using a *feature descriptor*
3. **Feature matching:** Find best matching descriptors across two images (*feature matching*)
4. Use feature matches to perform downstream task, e.g., pose estimation

What makes a good feature point?

Characteristics of good feature points

- Repeatability / invariance
  - The same feature point can be found in several images despite geometric and photometric transformations
- Saliency / distinctiveness
  - Each feature point is distinctive
  - Fewer ”false” matches

Goal: repeatability

- We want to detect (at least some of) the same points in both images.

No chance to find true matches!

- Yet we have to be able to run the detection procedure *independently* per image.

Kristen Grauman

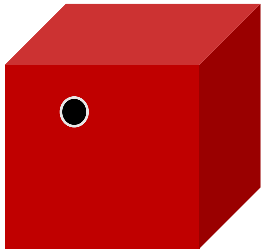
Goal: distinctiveness

- The feature point should be distinctive enough that it is easy to match
    - Should *at least* be distinctive from other patches nearby

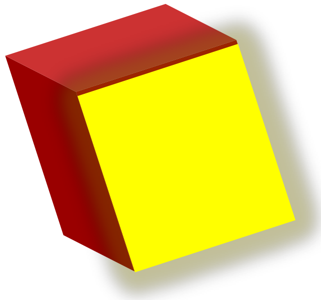
The aperture problem

- A single pixel by itself is not distinctive

Input pixel



Matching pixels



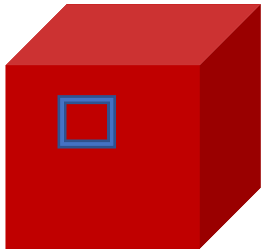
Pixel appearance

The aperture problem

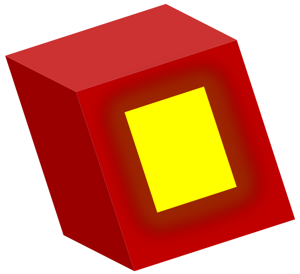
- Individual pixels are ambiguous
- Idea: Look at whole patches!

The aperture problem

- Individual pixels are ambiguous
- Idea: Look at whole patches!



Input patch



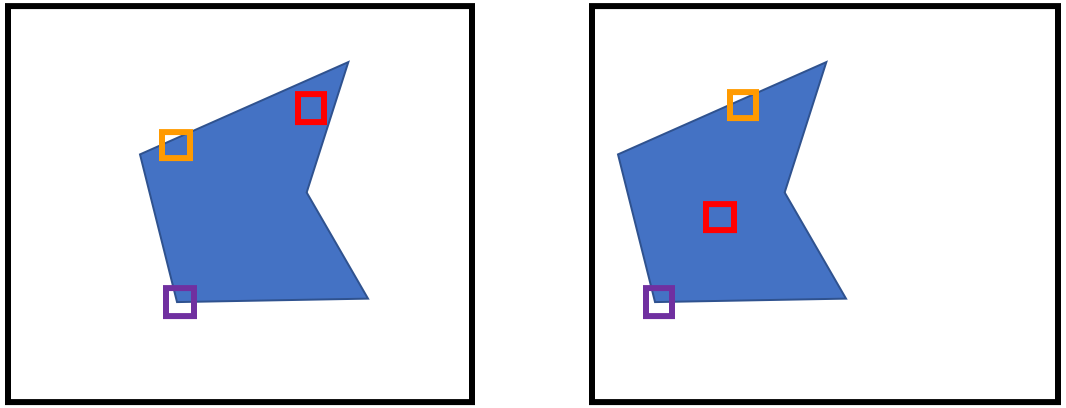
Matching
patch centers

The aperture problem

- *Patches can be ambiguous too!*
- What patches are distinctive?

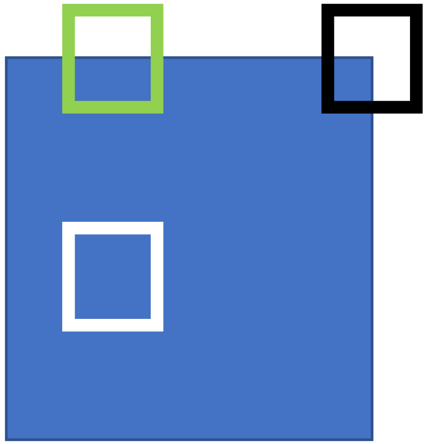
The aperture problem

- Corners are distinctive!
- How do we define/find corners?



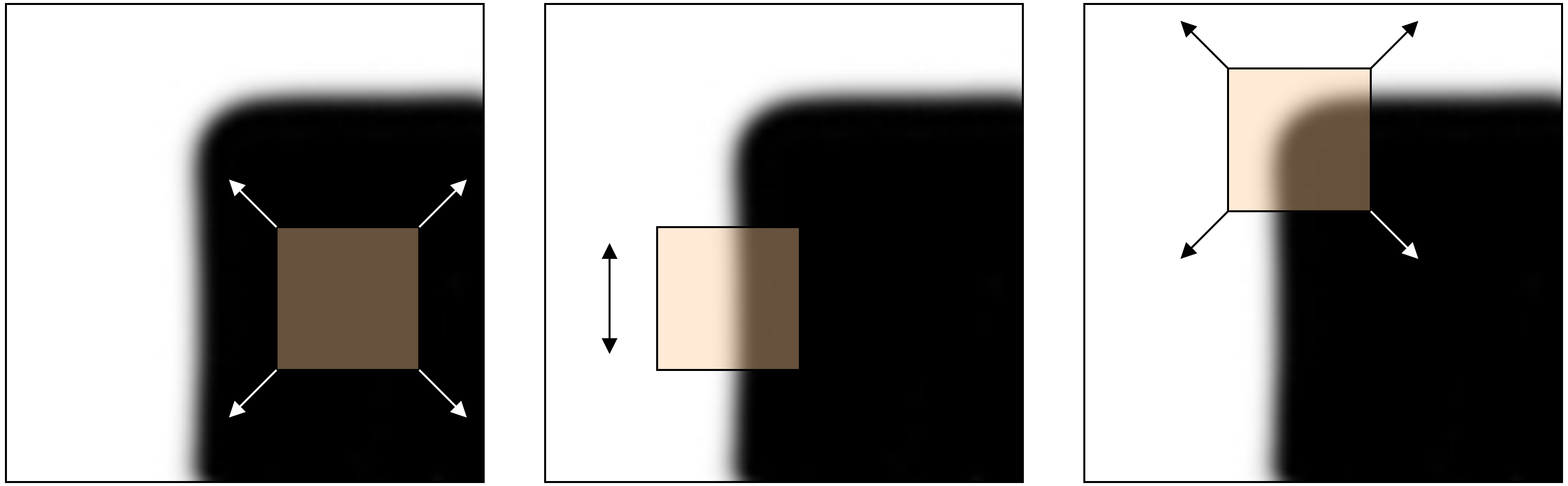
Corner detection

- Main idea: Translating window should cause large differences in patch appearance



Corner Detection: Basic Idea

- We should easily recognize the point by looking through a small window
- Shifting a window in *any direction* should give *a large change* in intensity



"flat" region: no change in all directions

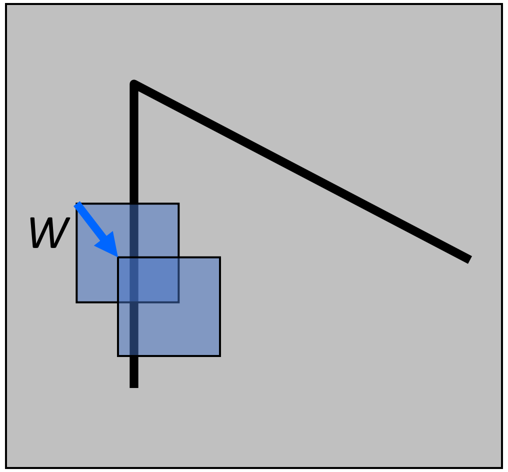
"edge": no change along the edge direction

"corner": significant change in all directions

Source: A. Efros

Corner detection the math

- Consider shifting the window *W* by (*u,v*)
  - how do the pixels in W change?
- Write pixels in window as a vector:



$$\phi_0 = [I(0,0), I(0,1), \ldots, I(n,n)]$$

$$\phi_1 = [I(0+u, 0+v), I(0+u, 1+v), \ldots, I(n+u, n+v)]$$

$$E(u,v) = \|\phi_0 - \phi_1\|_2^2$$

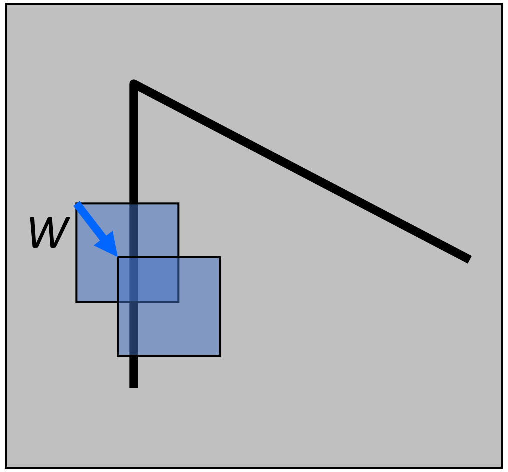
Corner detection: the math

Consider shifting the window *W* by (*u,v*)

- how do the pixels in W change?
- compare each pixel before and after by summing up the squared differences (SSD)
- this defines an SSD "error" *E*(*u,v*):

$$E(u,v) = \sum_{(x,y)\in W} [I(x+u, y+v) - I(x,y)]^2$$

- We want E(u,v) to be *as high as possible for all u, v!*



Small motion assumption

Taylor Series expansion of *I*:

$$I(x+u, y+v) = I(x, y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \text{higher order terms}$$

If the motion (u,v) is small, then first order approximation is good

$$\begin{aligned} I(x+u, y+v) &\approx I(x, y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v \\ &\approx I(x, y) + [I_x \ I_y] \begin{bmatrix} u \\ v \end{bmatrix} \end{aligned}$$

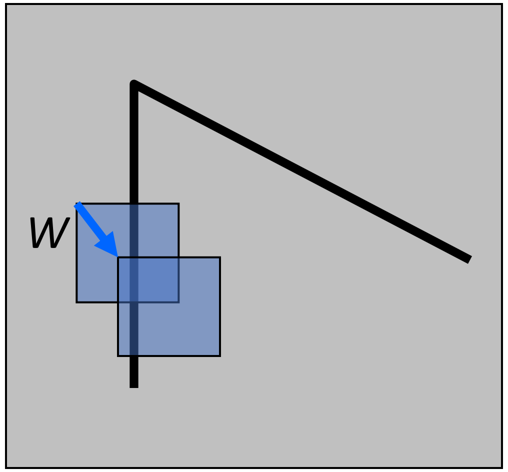
shorthand: $I_x = \frac{\partial I}{\partial x}$

Plugging this into the formula on the previous slide…

Corner detection: the math

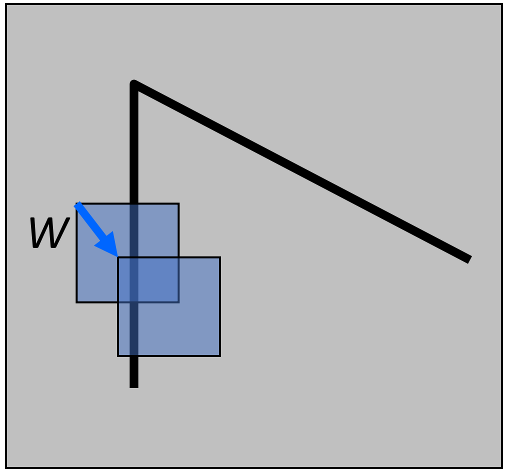
Consider shifting the window *W* by (*u,v*)

- define an SSD "error" *E(u,v)*:



$$
\begin{aligned}
E(u,v) &= \sum_{(x,y)\in W} \left[I(x+u,y+v) - I(x,y)\right]^2 \\
&\approx \sum_{(x,y)\in W} \left[I(x,y) + I_x u + I_y v - I(x,y)\right]^2 \\
&\approx \sum_{(x,y)\in W} \left[I_x u + I_y v\right]^2
\end{aligned}
$$

Corner detection: the math



Consider shifting the window *W* by (*u,v*)

- define an "error" *E(u,v)*:

$$E(u,v) \approx \sum_{(x,y)\in W} [I_x u + I_y v]^2$$

$$\approx Au^2 + 2Buv + Cv^2$$

$$A = \sum_{(x,y)\in W} I_x^2 \qquad B = \sum_{(x,y)\in W} I_x I_y \qquad C = \sum_{(x,y)\in W} I_y^2$$

- Thus, *E*(*u*,*v*) is locally approximated as a quadratic error function

A more general formulation

- Maybe all pixels in the patch are not equally important
- Consider a “window function” $w(x, y)$ that acts as weights
- $E(u, v) = \sum_{(x,y)\in W} w(x, y)[I(x + u, y + v) - I(x, y)]^2$
- Case till now:
    - w(x,y) = 1 inside the window, 0 otherwise

Using a window function

- Change in appearance of window w(x,y) for the shift [u,v]:

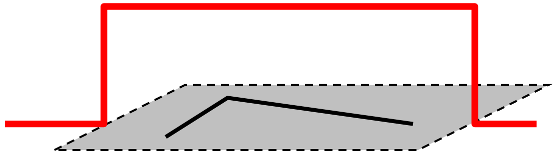
$$E(u,v)=\sum_{x,y} w(x,y)\left[I(x+u,y+v)-I(x,y)\right]^2$$

Window function

Shifted intensity

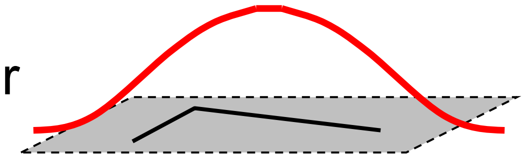
Intensity

Window function $w(x,y) =$



1 in window, 0 outside

or



Gaussian

Source: R. Szeliski

Redoing the derivation using a window function

$$E(u,v) = \sum_{x,y\in W} w(x,y)\,[I(x+u,y+v) - I(x,y)]^2$$

$$\approx \sum_{x,y\in W} w(x,y)\big[I(x,y) + uI_x(x,y) + vI_y(x,y) - I(x,y)\big]^2$$

$$= \sum_{x,y\in W} w(x,y)\big[uI_x(x,y) + vI_y(x,y)\big]^2$$

$$= \sum_{x,y\in W} w(x,y)[u^2 I_x(x,y)^2 + v^2 I_y(x,y)^2 + 2uvI_x(x,y)I_y(x,y)]$$

Redoing the derivation using a window function

-

$$E(u,v) \approx \sum_{x,y \in W} w(x,y)[u^2 I_x(x,y)^2 + v^2 I_y(x,y)^2 + 2uvI_x(x,y)I_y(x,y)]$$

$$= Au^2 + 2Buv + Cv^2$$

$$A = \sum_{x,y \in W} w(x,y) I_x(x,y)^2$$

$$B = \sum_{x,y \in W} w(x,y) I_x(x,y) I_y(x,y)$$

$$C = \sum_{x,y \in W} w(x,y) I_y(x,y)^2$$

The second moment matrix

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{M} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$\underbrace{M = \sum_{x,y \in W} w(x,y) \begin{bmatrix} I_x(x,y)^2 & I_x(x,y) I_y(x,y) \\ I_x(x,y) I_y(x,y) & I_y(x,y)^2 \end{bmatrix}}_{\text{Second moment matrix}}$$

The second moment matrix

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{M} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$\underbrace{M = \sum_{x,y \in W} w(x,y) \begin{bmatrix} I_x(x,y)^2 & I_x(x,y)I_y(x,y) \\ I_x(x,y)I_y(x,y) & I_y(x,y)^2 \end{bmatrix}}_{\text{Second moment matrix}}$$

Recall that we want E(u,v) to be as large as possible for all u,v

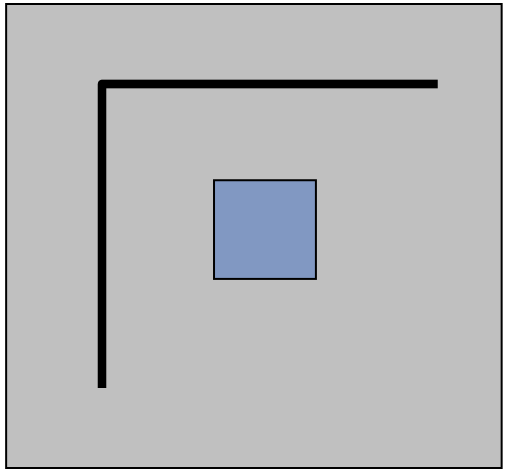
What does this mean in terms of M?

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{M} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$A = \sum_{(x,y)\in W} I_x^2$$

$$B = \sum_{(x,y)\in W} I_x I_y$$

$$C = \sum_{(x,y)\in W} I_y^2$$



Flat patch: $I_x = 0$

$I_y = 0$

$$M = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$M \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

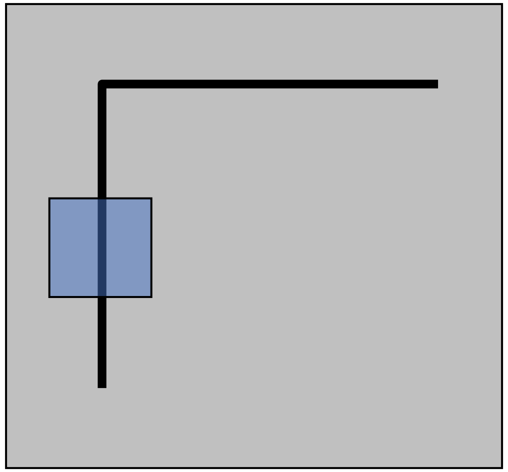
$$E(u,v) = 0 \quad \forall u, v$$

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{M} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$A = \sum_{(x,y)\in W} I_x^2$$

$$B = \sum_{(x,y)\in W} I_x I_y$$

$$C = \sum_{(x,y)\in W} I_y^2$$



Vertical edge: $I_y = 0$

$$M = \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix}$$

$$M \begin{bmatrix} 0 \\ v \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

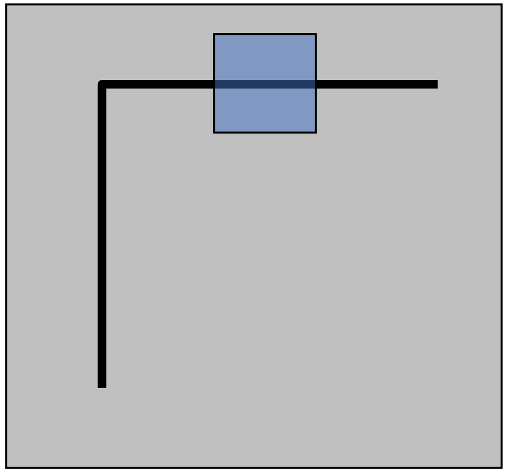
$$E(0,v) = 0 \ \ \forall v$$

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{M} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$A = \sum_{(x,y)\in W} I_x^2$$

$$B = \sum_{(x,y)\in W} I_x I_y$$

$$C = \sum_{(x,y)\in W} I_y^2$$



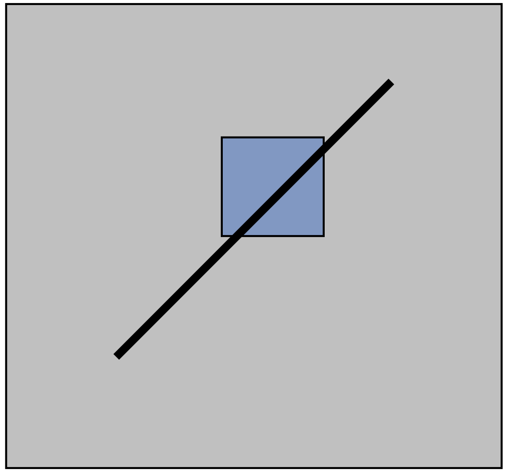
Horizontal edge: $I_x = 0$

$$M = \begin{bmatrix} 0 & 0 \\ 0 & C \end{bmatrix}$$

$$M \begin{bmatrix} u \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$E(u,0) = 0 \; \forall u$$

What about edges in arbitrary orientation?



$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

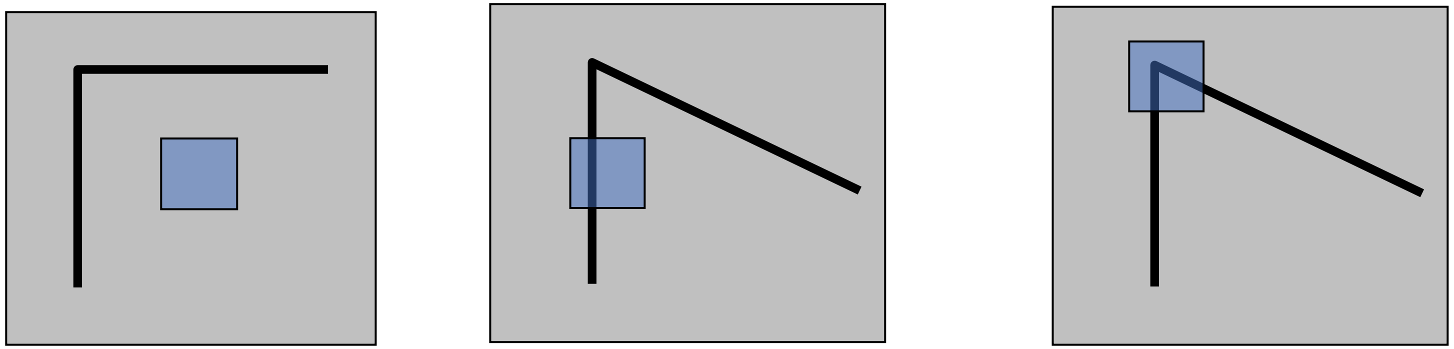
$$M \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Leftrightarrow E(u,v) = 0$$

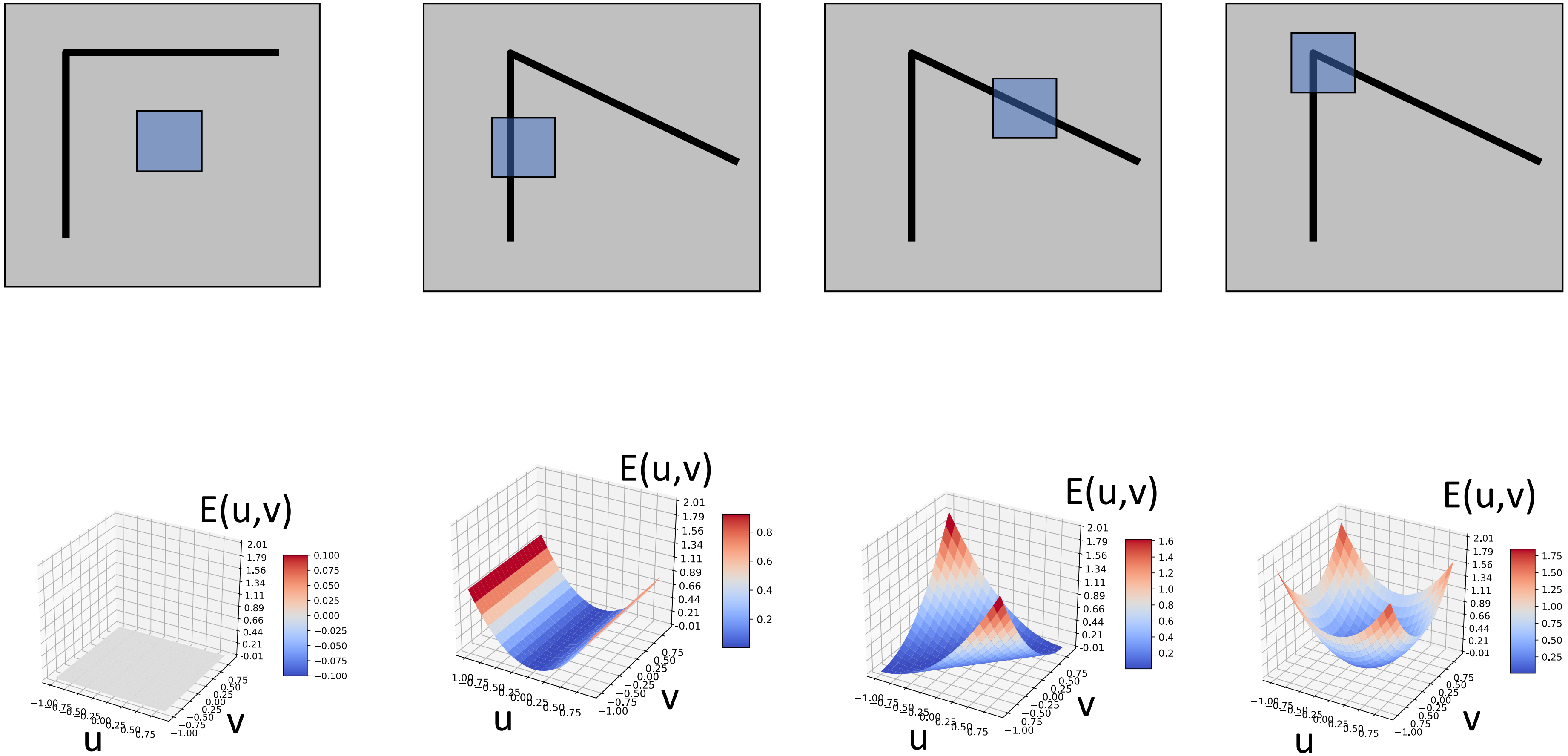
Solutions to Mx = 0 are directions for which E is 0: window can slide in this direction without changing appearance

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

Solutions to Mx = 0 are directions for which E is 0: window can slide in this direction without changing appearance

For corners, we want no such directions to exist



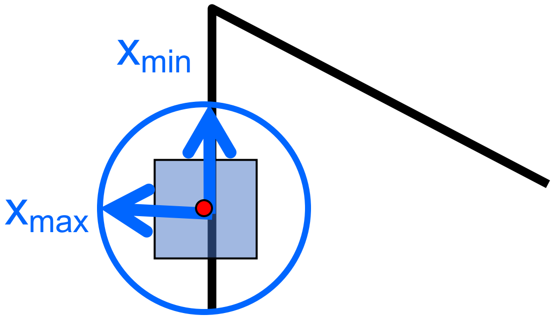

E(u,v)
u
v
E(u,v)
u
v
E(u,v)
u
v
E(u,v)
u
v

Eigenvalues and eigenvectors of M

- $Mx = 0 \Rightarrow Mx = \lambda x$: x is an eigenvector of M with eigenvalue 0
- M is 2 x 2, so it has 2 eigenvalues $(\lambda_{max}, \lambda_{min})$ with eigenvectors (x_{max}, x_{min})
- $E(x_{max}) = \;\; x_{max}^T M x_{max} = \lambda_{max}||x_{max}||^2 = \lambda_{max}$ (eigenvectors have unit norm)
- $E(x_{min}) = x_{min}^T M x_{min} = \lambda_{min}||x_{min}||^2 = \lambda_{min}$

Eigenvalues and eigenvectors of M

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$



$$M\, x_{\max} = \lambda_{\max} x_{\max}$$

$$M\, x_{\min} = \lambda_{\min} x_{\min}$$

Eigenvalues and eigenvectors of M

- Define shift directions with the smallest and largest change in error
- x_{max} = direction of largest increase in *E*
- λ_{max} = amount of increase in direction x_{max}
- x_{min} = direction of smallest increase in *E*
- λ_{min} = amount of increase in direction x_{min}

Interpreting the eigenvalues