CIS 5800

Machine Perception

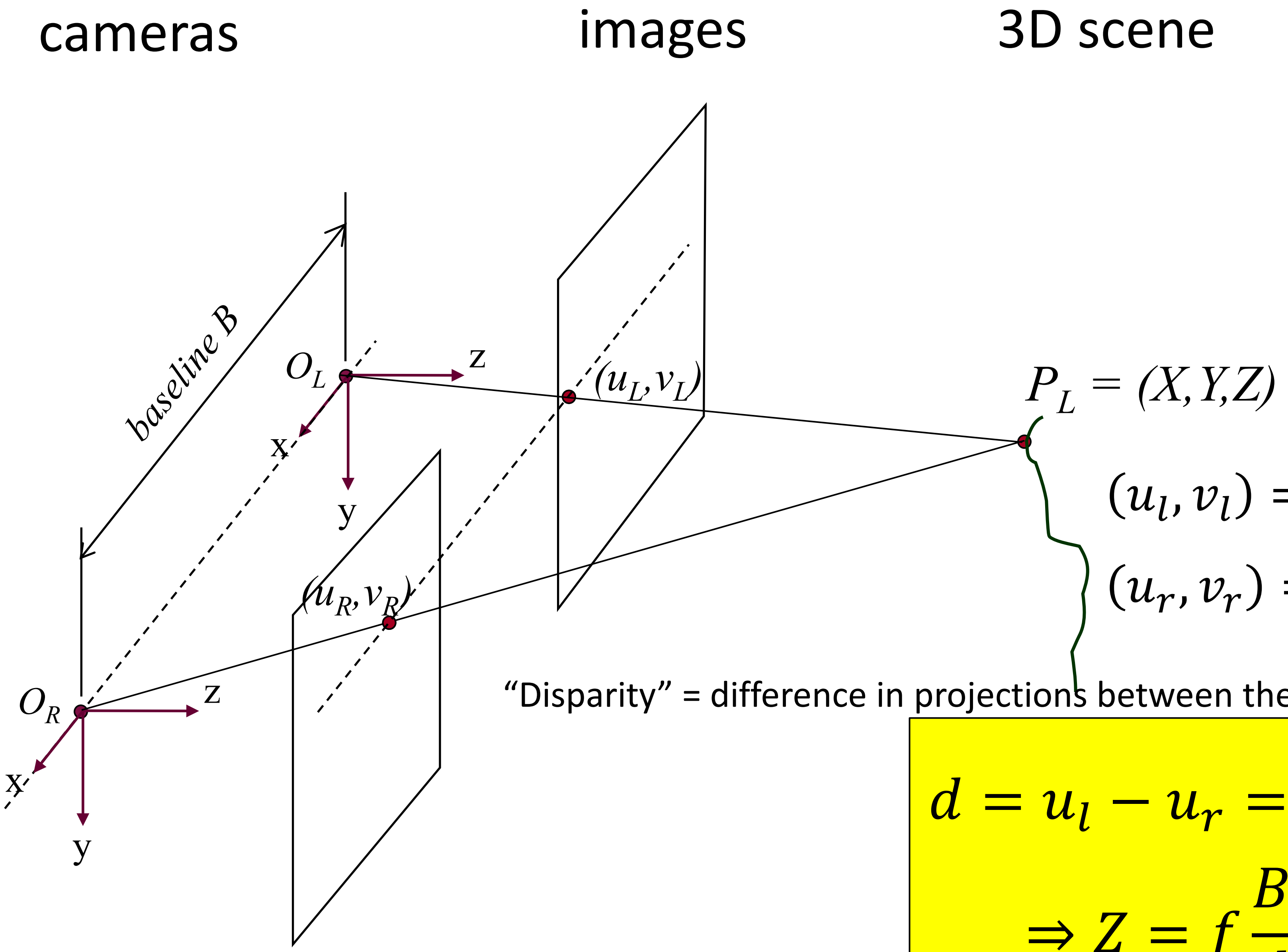
Instructor: Lingjie Liu

Lec 22: April 23, 2025

Administrivia

- HW5 (optional, 6pts) for grade compensation will release by Friday and the due is May 14.
- Final exam coming up
  - Date reminder: Wednesday May 7, 3-5pm in DRLB A1. (Info is on courses.upenn.edu)
  - Syllabus: Mainly the material covered in class after Wed March 19 (not covered by mid-term exam).
  - Review lecture in the last class on Wed April 30.
  - If you are unable to attend the midterm exam in person on May 7, please complete the form by April 30: https://forms.gle/JwaAxrKGfBzoo6z77
  - Also, you need to contact the Weingarten Office for academic accommodations and send me the paperwork or approval from the Weingarten Office.

Recap: Basic Parallel Stereo Derivations



$$(u_l, v_l) = \left(f\frac{X}{Z}, f\frac{Y}{Z}\right)$$

$$(u_r, v_r) = \left(f\frac{X-B}{Z}, f\frac{Y}{Z}\right)$$

"Disparity" = difference in projections between the two cameras

$$\boxed{\begin{gathered} d = u_l - u_r = f\frac{B}{Z} \\ \Rightarrow Z = f\frac{B}{d} \end{gathered}}$$

Recap: Putting this in context

	SfM / stitching	Motion from flow*	Triangulation	Optical flow	Stereo & correspondences
3D structure	unknown	unknown	unknown	unknown	unknown
Camera rotations	unknown	known	known	unknown	known
Camera translations	unknown	unknown	known	unknown	known
Image pixel correspondences	known	known	known	unknown	Unknown (and large motions and dense)

Note: red unknown = we want to find, black unknown = we don't care

Our strategy

- First deal with dense correspondence finding for the frontoparallel 2-camera case

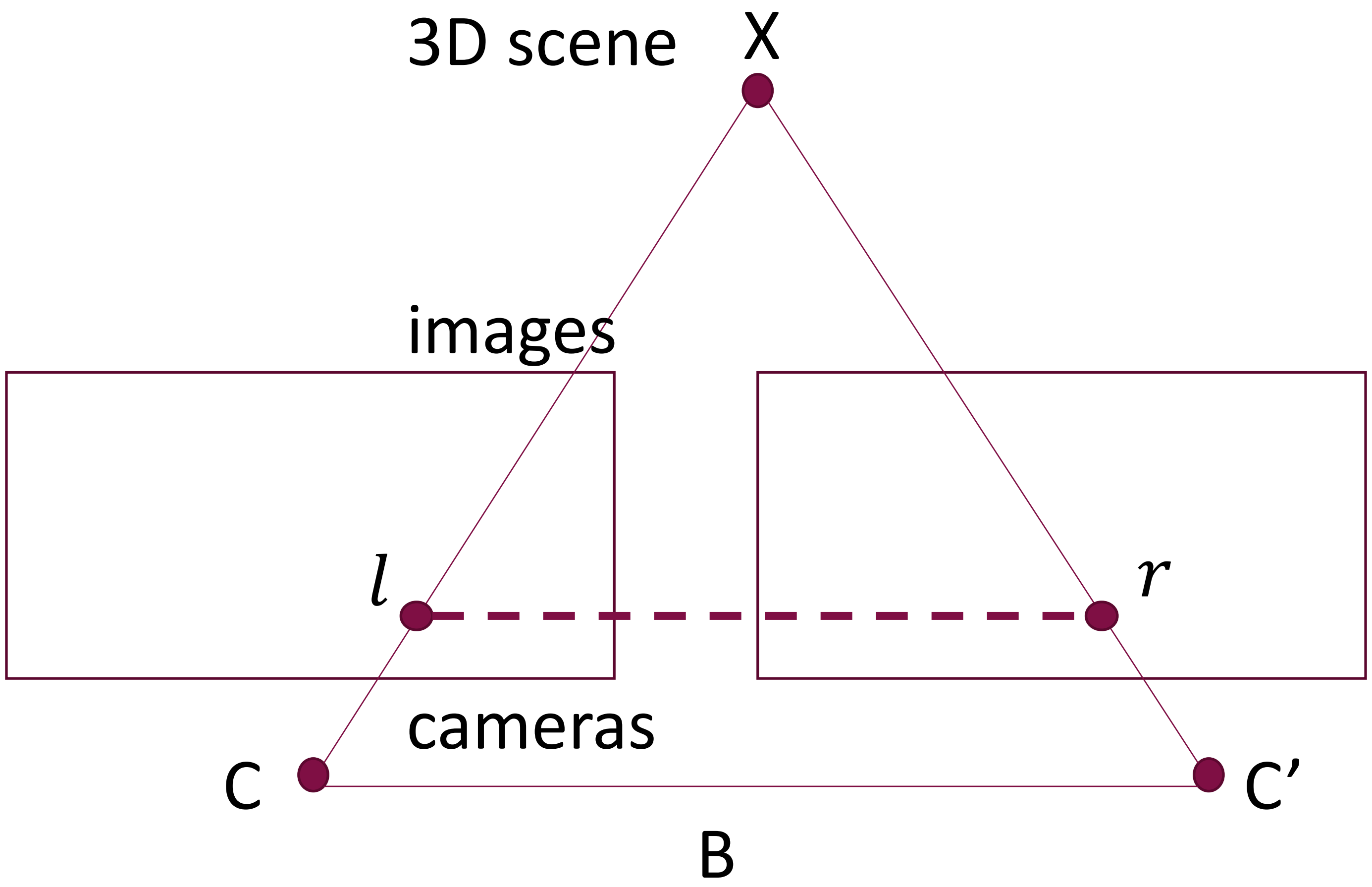
- Then see how to “rectify” non-frontoparallel cameras to be frontoparallel.

- Then, how to perform multi-view stereo (MVS)
  - Straightforward multi-baseline extension of 2-view stereo
  - The “plane sweep” technique for MVS

- Finally, improvement through dynamic programming.

Searching for dense correspondences
in the frontoparallel stereo setting

Correspondences for frontoparallel cameras



We have derived that

$$(u_l, v_l) = \left(f\,\frac{X}{Z}, f\,\frac{Y}{Z}\right) \text{ and}$$

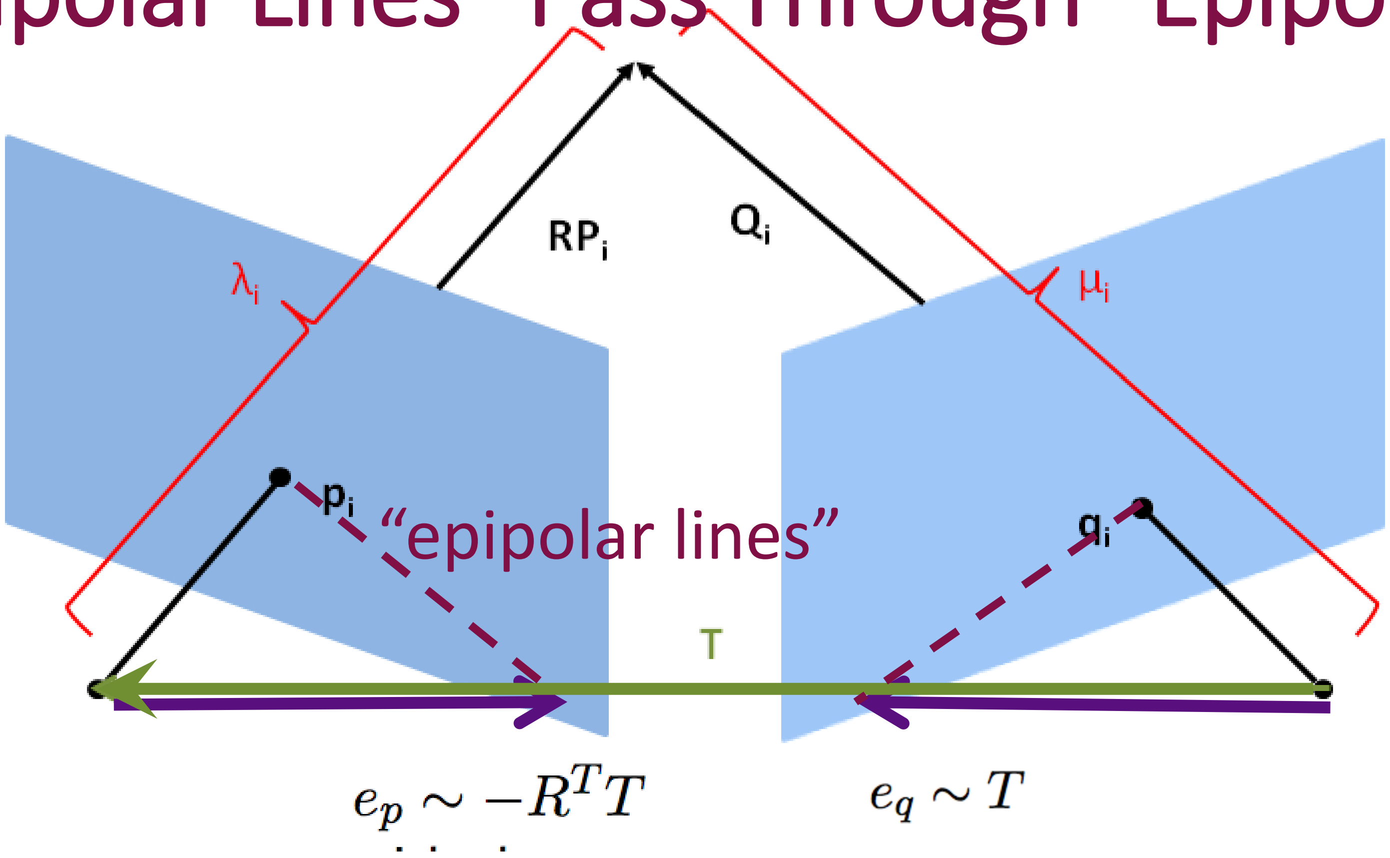
$$(u_r, v_r) = \left(f\,\frac{X-B}{Z}, f\,\frac{Y}{Z}\right)$$

So correspondences must lie on the same horizontal line!

Note: we have also seen earlier that epipolar lines for this case are horizontal!

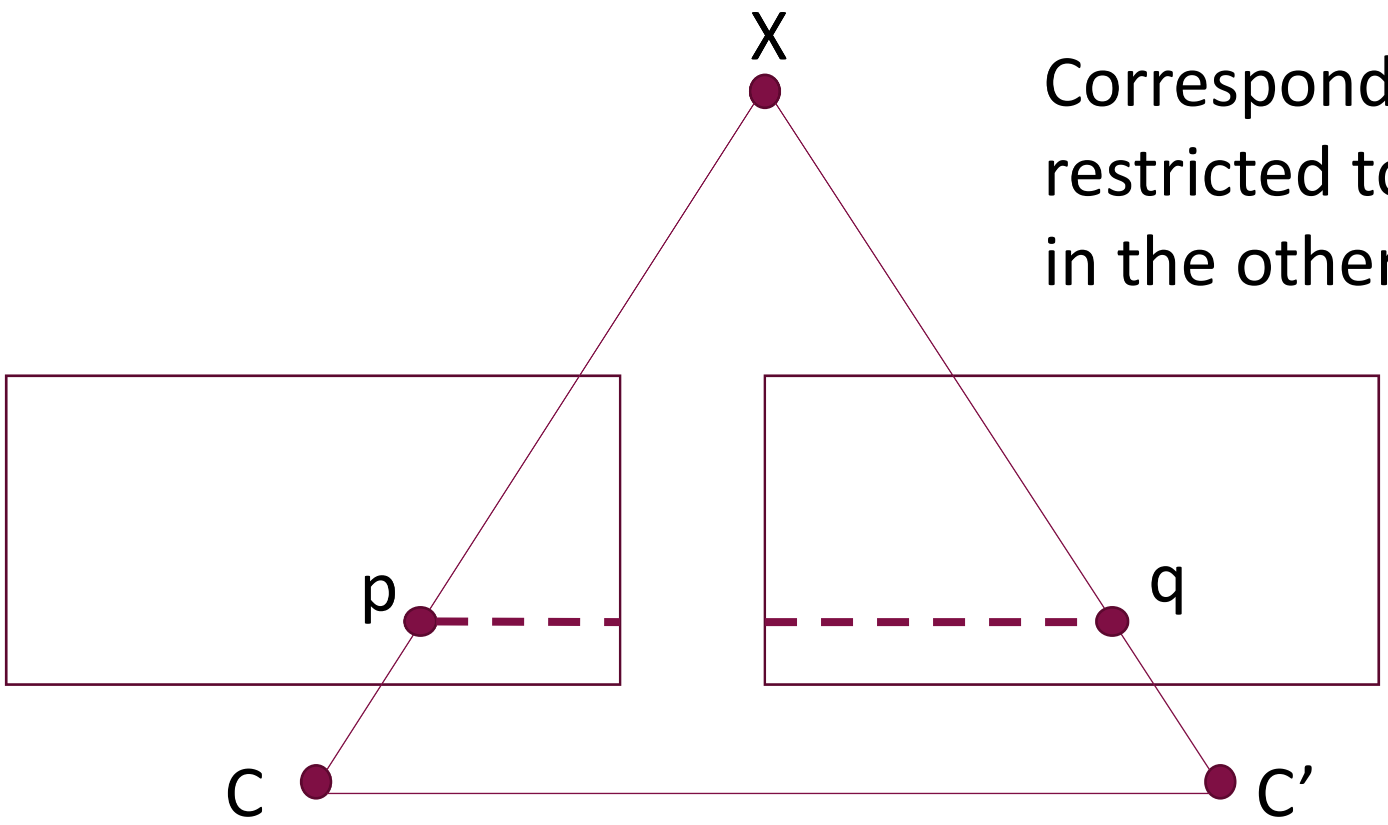
(Q: Recall why?)

Recap: “Epipolar Lines” Pass Through “Epipoles”



$e_p \sim -R^T\ T$ and $e_q \sim T$ are the “epipoles” = images of the other camera center on each plane = intersections of baseline T with the two planes = VP of the translation direction in each plane.

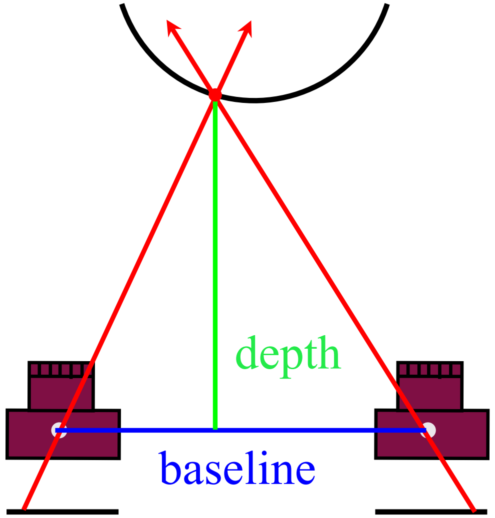
Recap: “frontoparallel” / “parallel stereo” cameras



Correspondences are restricted to the “same” row in the other image!

$$e_p \sim -R^T\, T = -T = [-B, 0, 0]^T$$
$$e_q \sim T = [B, 0, 0]^T$$

Correspondences for parallel stereo



$$Z(x, y) = \frac{f\,B}{d(x, y)}$$

$Z(x, y)$ is depth at pixel (x, y)
$d(x, y)$ is disparity

Matching correlation windows across scan lines

Finding correspondences is a search problem!

Search range of disparity

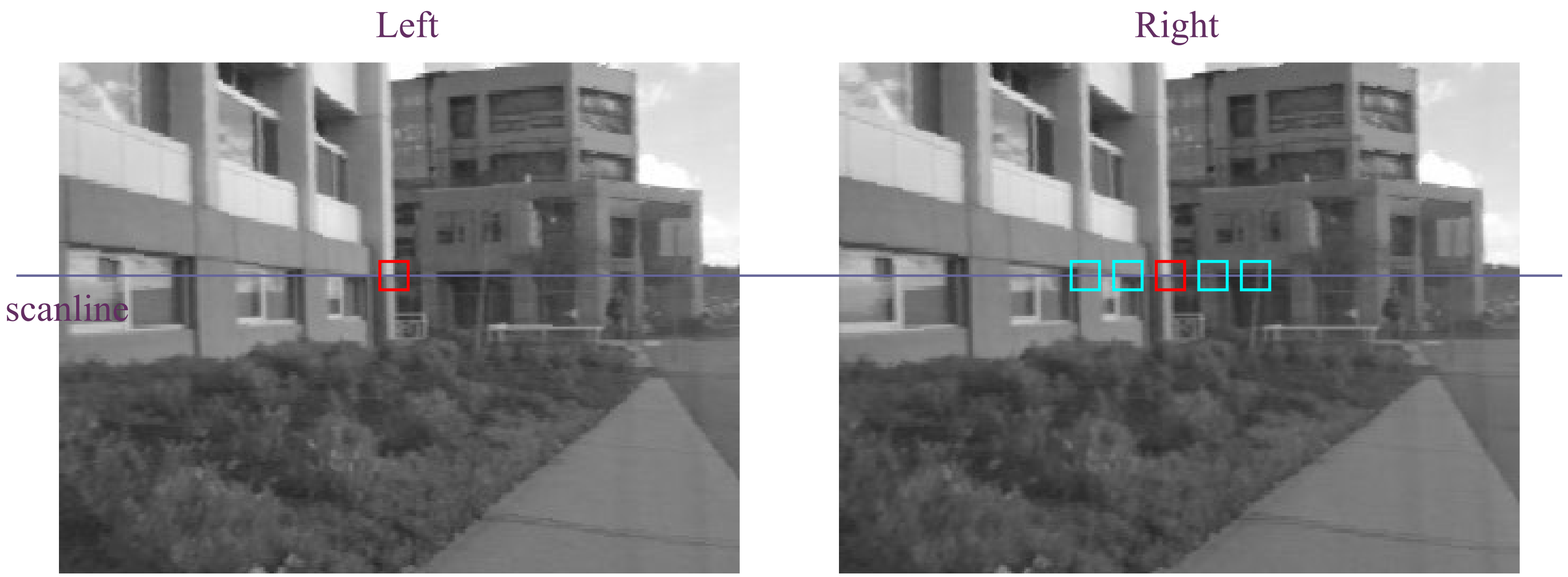
- Assume some minimum "depth": Z_{min}
  - Set $d_{\max} = \frac{fB}{Z_{\min}}$, e.g. $d_{\max} = 100$
- Quantize the interval [-d_{max}, d_{max}]!
  - e.g. [-100, 100]-> candidate disparities [-100,-95, ..., 0, 5, 10, ..., 100]
- Now, must select from these candidate disparities for each pixel.

Components of Stereo Correspondence Matching

- Matching criterion (error function)
  - Quantify similarity of a pixel pair (candidate correspondence)
  - Options: **direct RGB intensity difference**, **correlation** etc.
- Aggregation method
  - How the error function is accumulated
  - Options: Pixelwise, edgewise, **window-wise**, segment-wise …
- Optimization and winner selection
  - How the final correspondences are determined
  - Options: **Winner-take-all**, **dynamic programming**, graph cuts, belief propagation

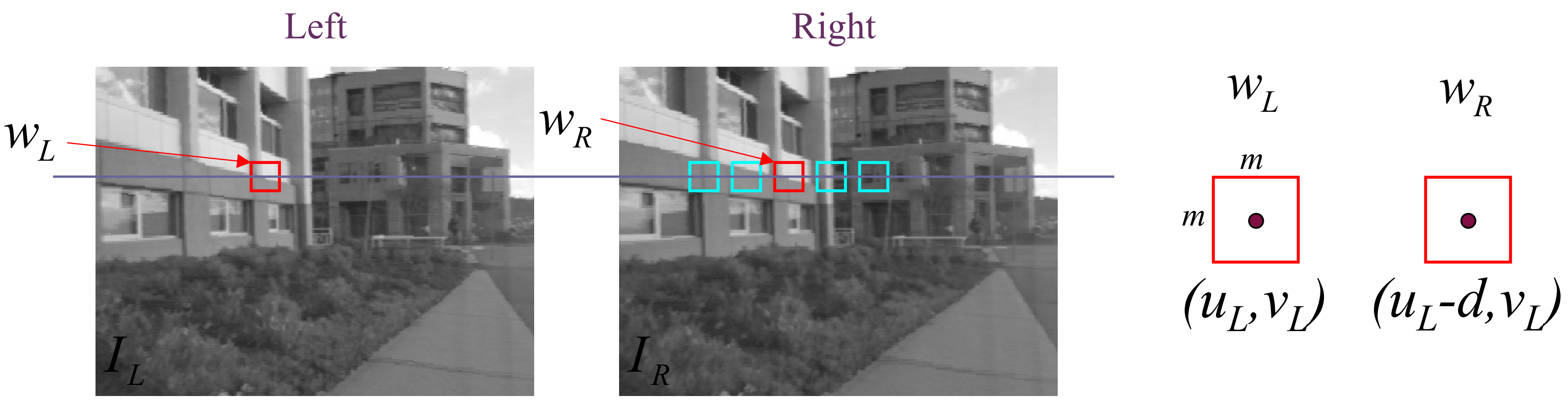
We will focus on the bolded choices

Matching Windows



For a given left window, we will select from various right windows along the scanline, as candidate correspondences.

Sum of Squared Differences (SSD) Over the Window



w_L and w_R are corresponding m by m windows of pixels.

We define the window function :

$$W_m(x,y) = \{u,v \mid x - \tfrac{m}{2} \le u \le x + \tfrac{m}{2}, y - \tfrac{m}{2} \le v \le y + \tfrac{m}{2}\}$$

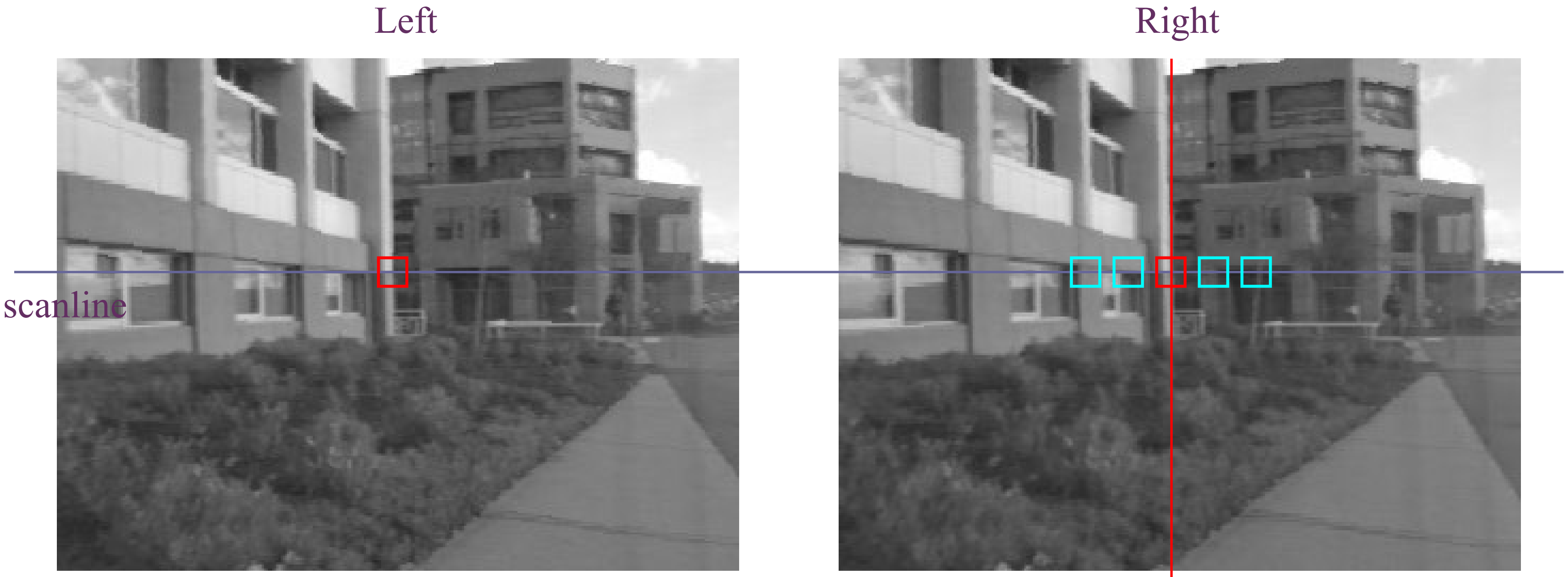
The SSD cost measures the intensity difference as a function of disparity :

$$C_r(x,y,d) = \sum_{(u,v)\in W_m(x,y)} [I_L(u,v) - I_R(u-d,v)]^2$$

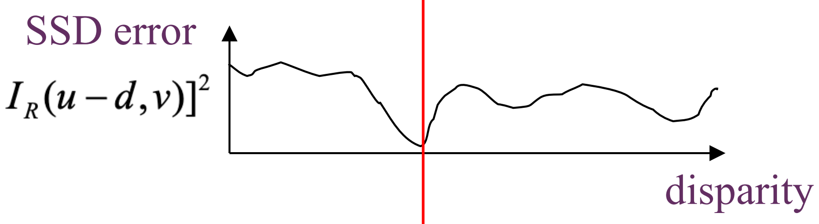
Note: SSD is also what we minimized in LK Optical Flow!

Matching Windows with SSD + Winner-Take-All

You will learn an alternative to the SSD, called the normalized cross-correlation



$$C_r(x,y,d) = \sum_{(u,v)\in W_m(x,y)} [I_L(u,v) - I_R(u-d,v)]^2$$



Simplest optimization scheme: simply assign lowest SSD as match for every window

“Winner-take-all”

What if the cameras are not frontoparallel?

Our strategy

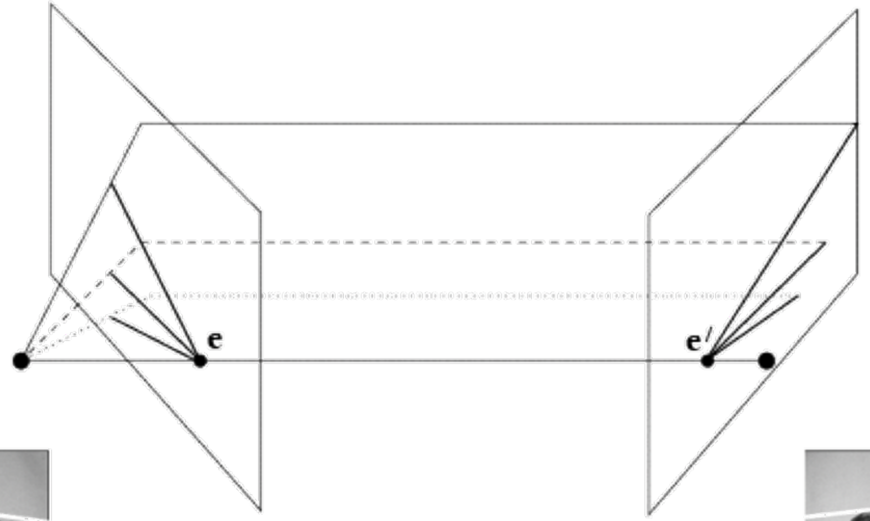
- First deal with dense correspondence finding for the frontoparallel 2-camera case

- Then see how to “rectify” non-frontoparallel cameras to be frontoparallel.

- Then, how to perform multi-view stereo (MVS)
  - Straightforward multi-baseline extension of 2-view stereo
  - The “plane sweep” technique for MVS

- Finally, improvement through dynamic programming.

Stereo Rectification



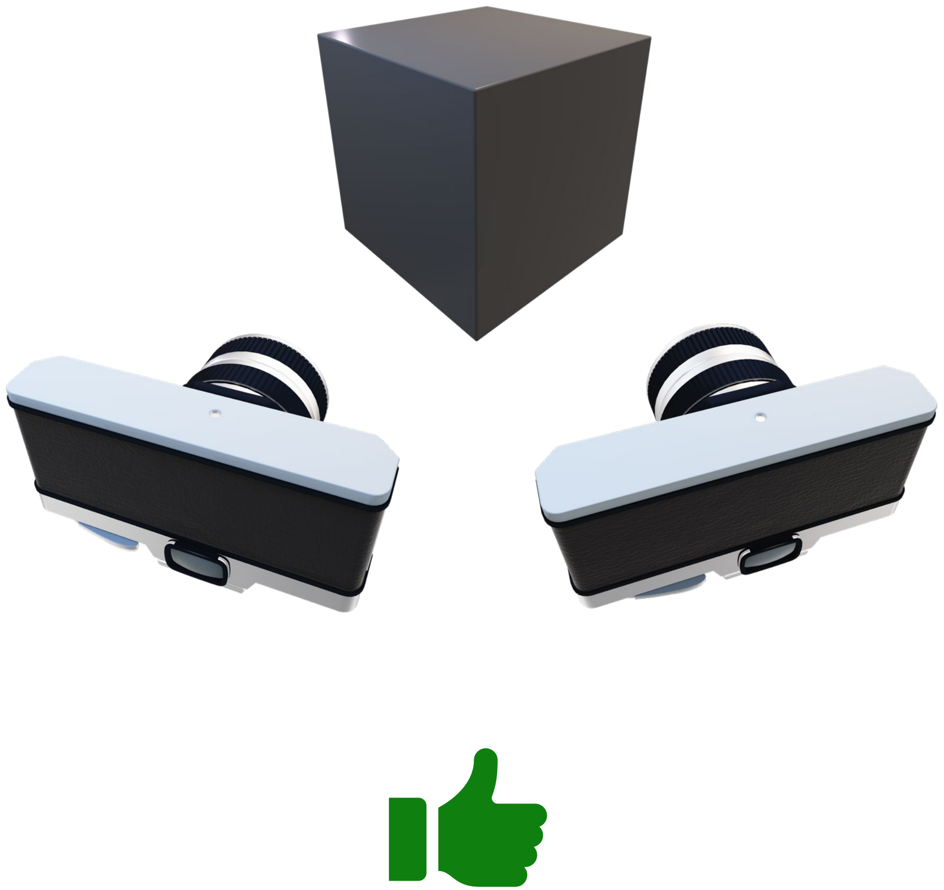
Q: What if the cameras weren’t frontoparallel to start with?

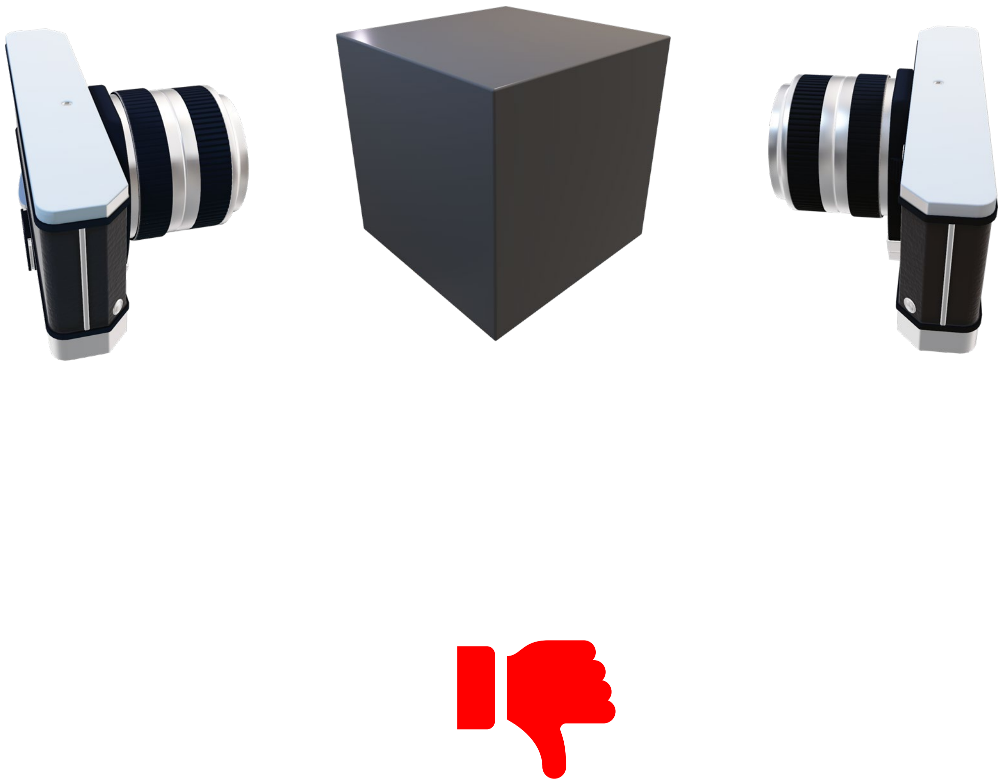
A: “Rectification”. *Make* them!

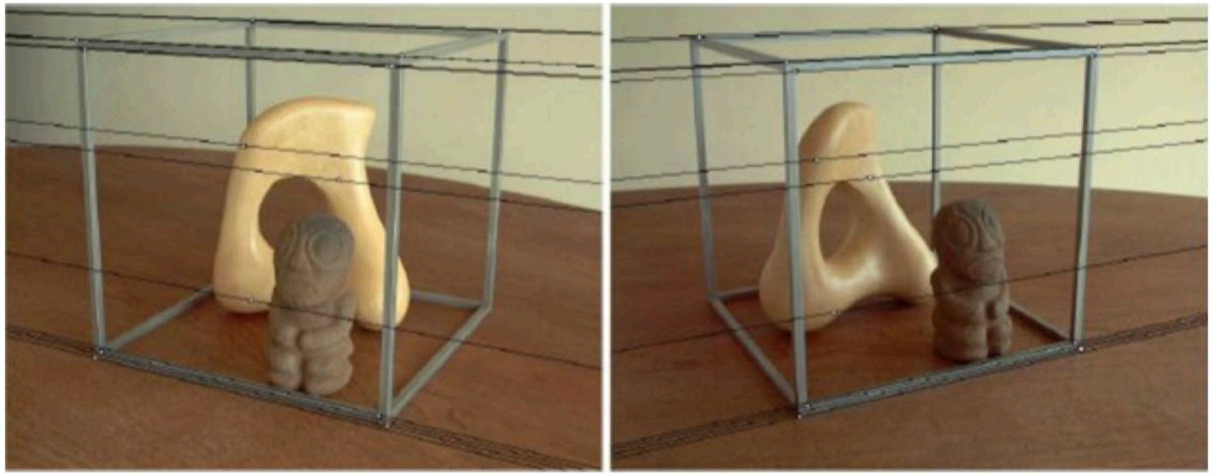
Key ideas: (1) cameras can be rotated in place through homographies
(2) frontoparallel => image plane parallel to the line connecting the cameras.

Stereo Rectification

Works best in settings where the cameras are roughly aligned and nearby

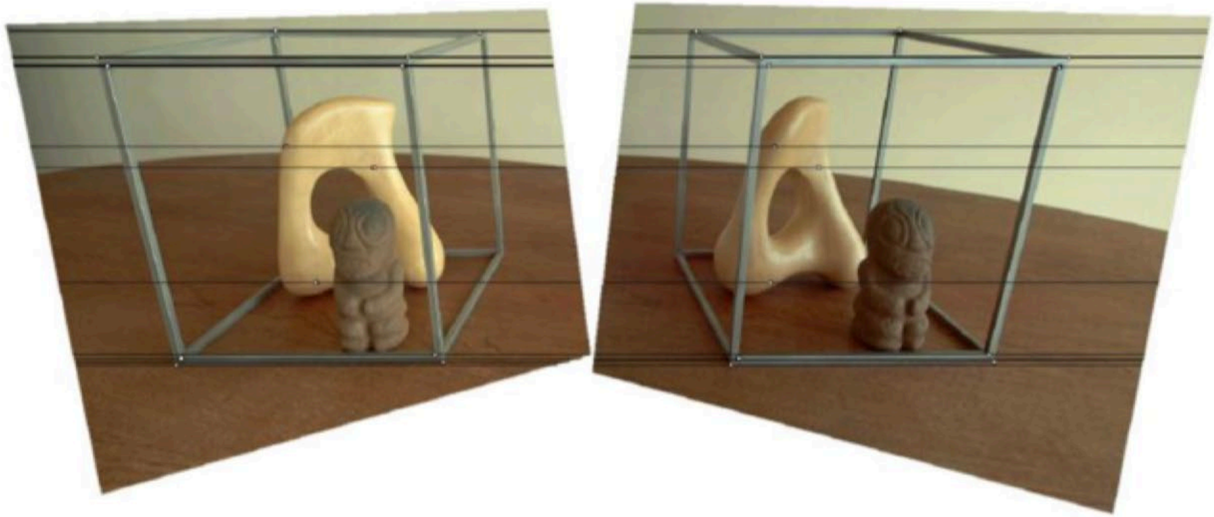






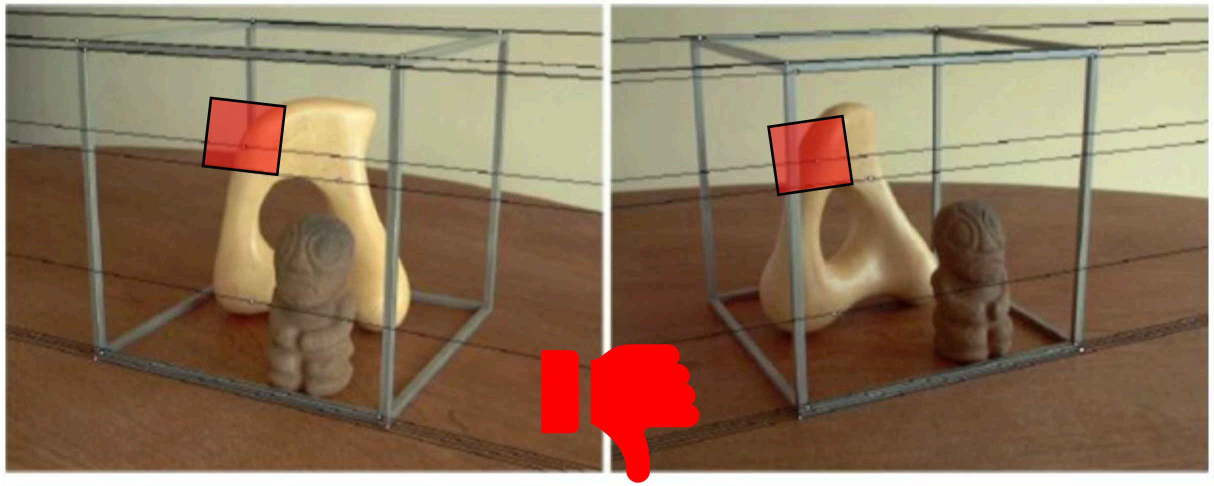
We know that a frontoparallel camera arrangement <=> horizontal epipolar lines

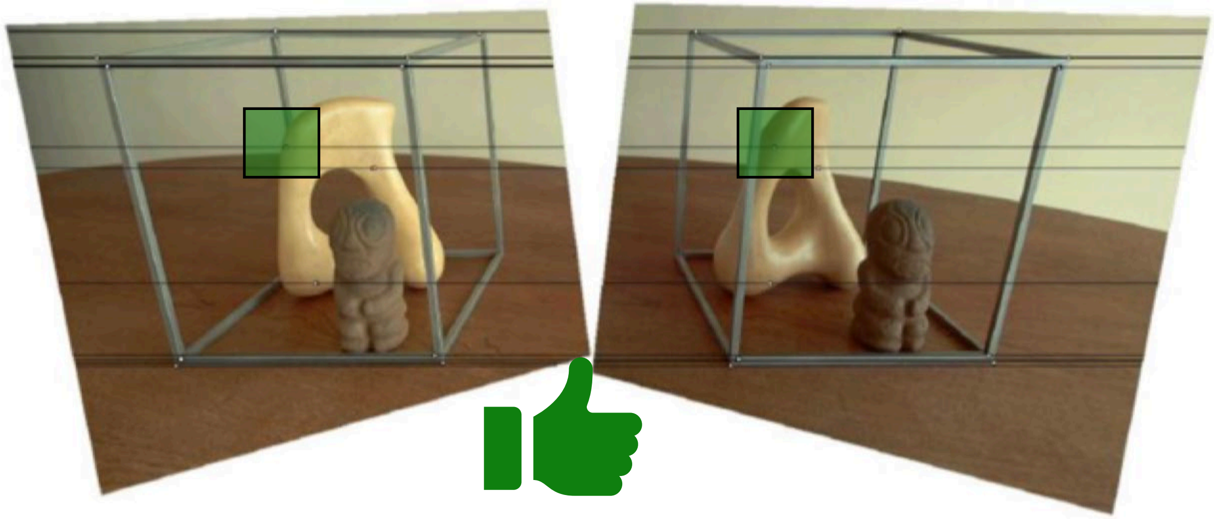
How can you make the epipolar lines horizontal?



Why Rectify?

- Rectification makes triangulation easy (depth $\propto$ 1/disparity)
- Also makes axis-aligned window search for correspondences easy, rather than searching along slanted epipolar lines

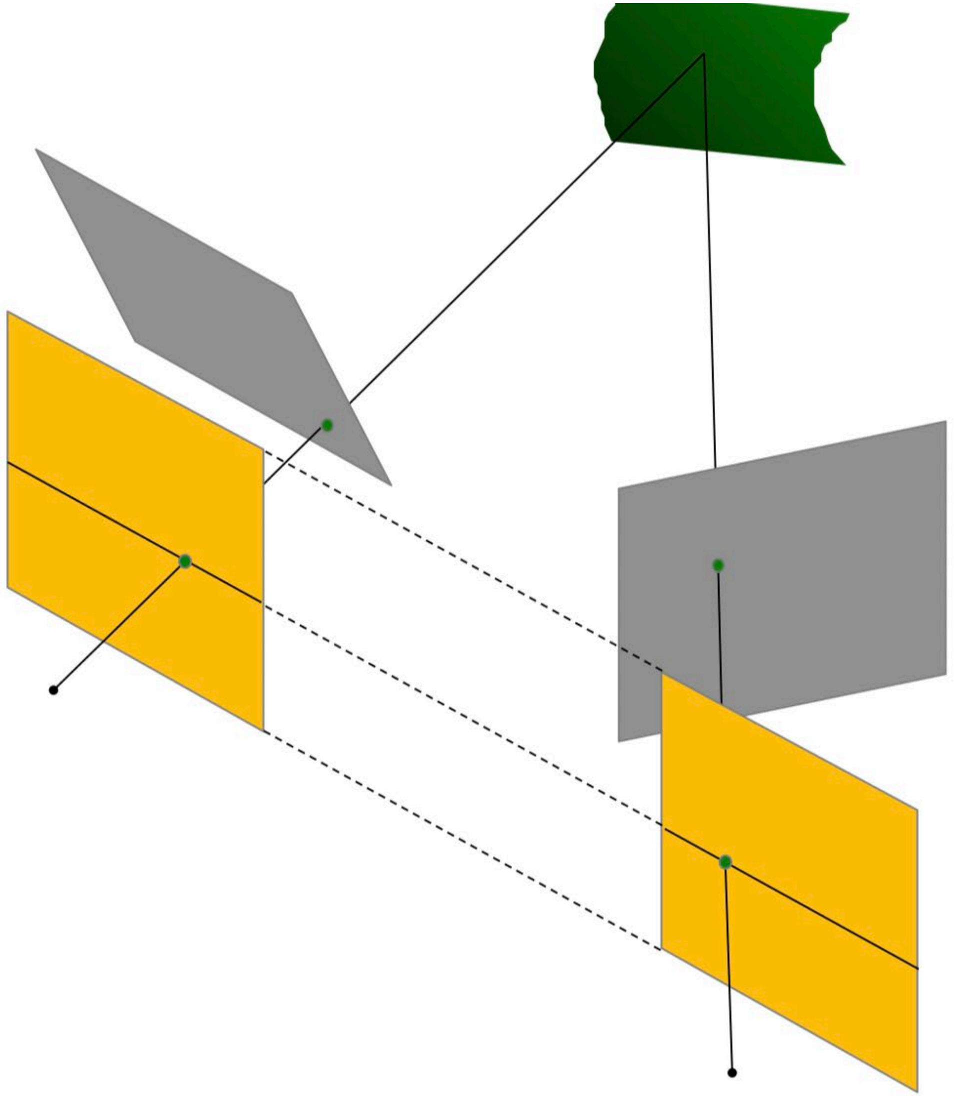




Key Idea

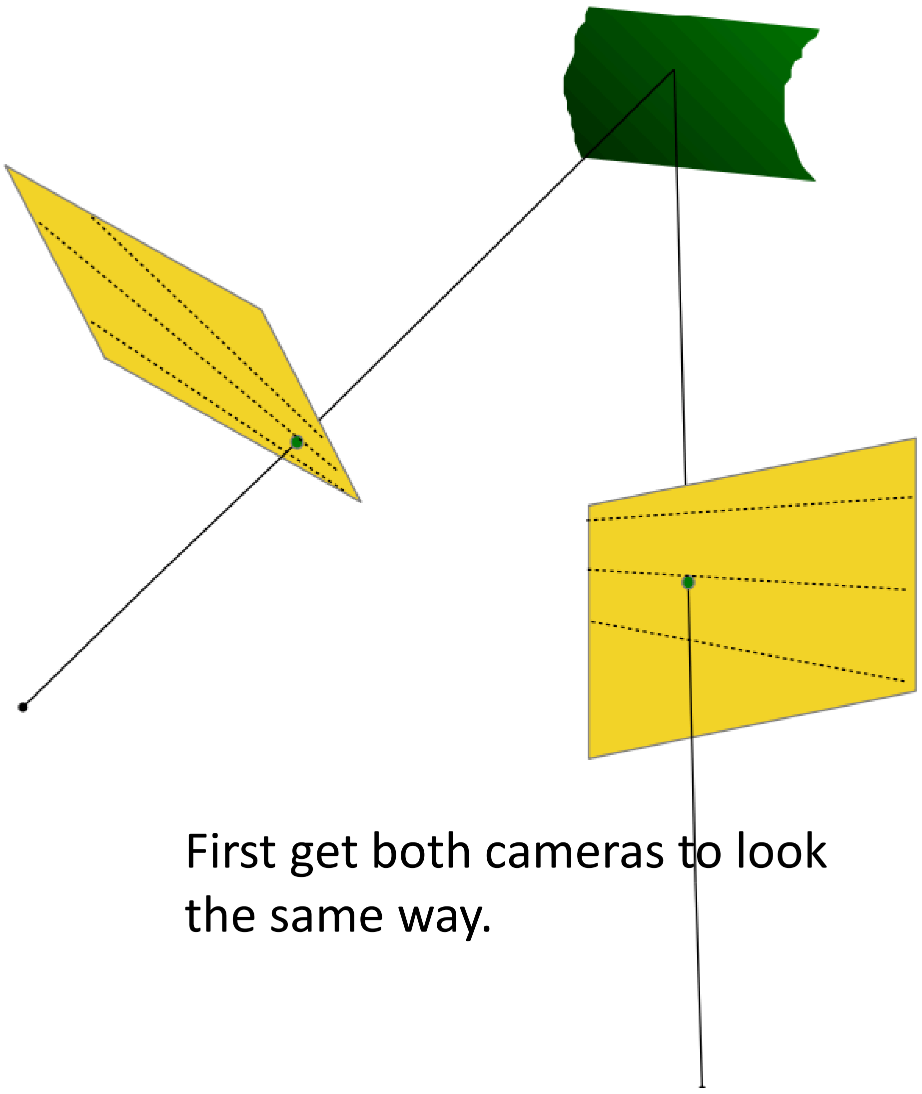
Reproject image planes onto a common plane parallel to the line between camera centers

Need two homographies (3x3 transform), one for each input image reprojection



C. Loop and Z. Zhang. Computing Rectifying Homographies for Stereo Vision.Computer Vision and Pattern Recognition, 1999.

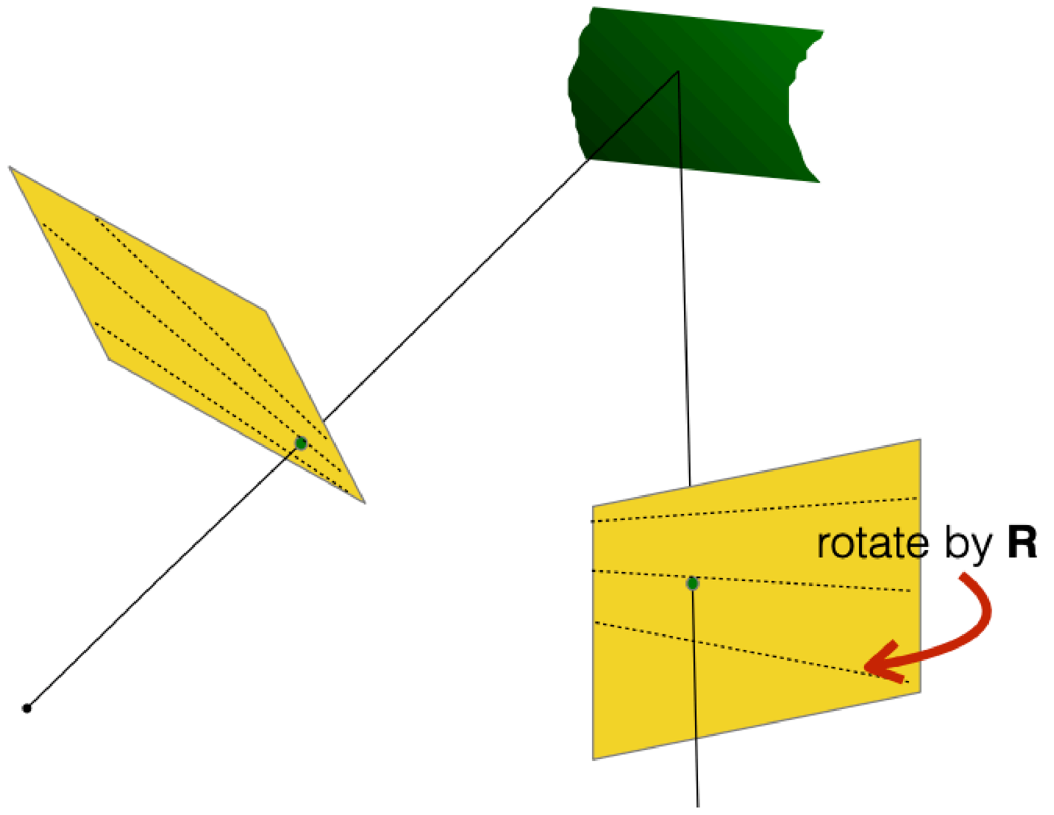
Stereo Rectification:



1. Compute **E** to get **R**
2. Rotate right image by **R**
3. Rotate both images by $\mathbf{R}_{\mathbf{rect}}$

First get both cameras to look the same way.

Stereo Rectification:

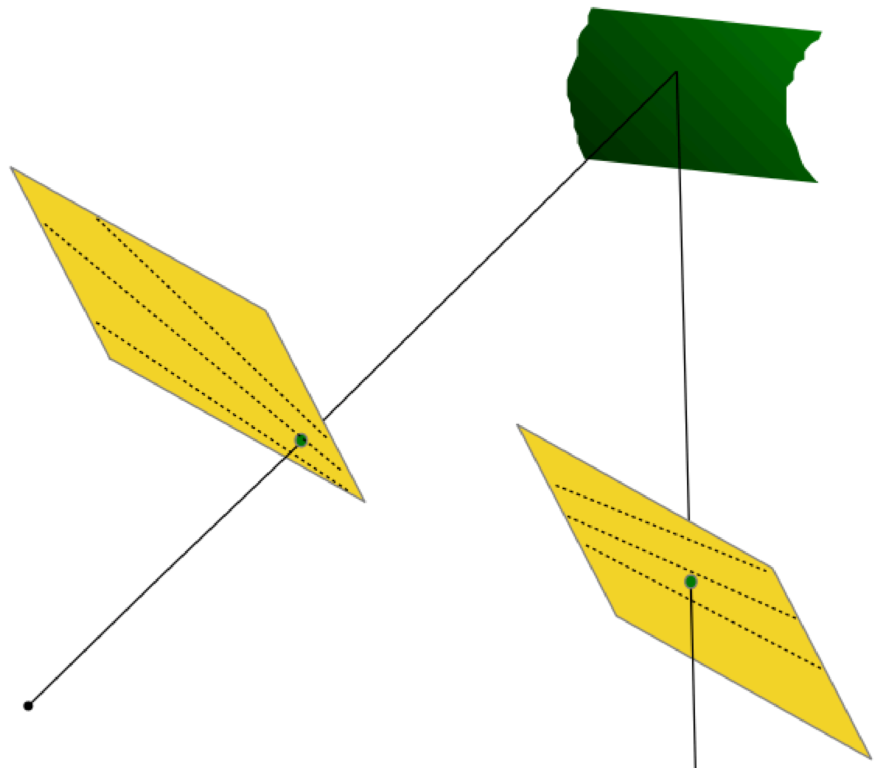


1. Compute **E** to get **R**
2. Rotate right image by **R**
3. Rotate both images by $\mathbf{R}_{\mathbf{rect}}$

Recall, when a camera is rotated by R, it corresponds to homography R:

$$\lambda q = \mu Rp + T(= 0) = \mu Rp, \text{ or } q \sim Rp$$

Stereo Rectification:



1. Compute **E** to get **R**
2. Rotate right image by **R**
3. Rotate both images by $\mathbf{R}_{\mathbf{rect}}$

At this stage, both cameras are facing in the same direction.
But this is not enough!

Next, we must rotate both cameras (by the same rotation), to face perpendicular to baseline, as in frontoparallel cameras.

Building R_{rect} by setting epipole to ∞

(unit vector of epipole, because epipole = image of the other camera center)

If $\boldsymbol{r}_1 = \boldsymbol{e}_1 = \dfrac{T}{||T||}$ and $\boldsymbol{r}_2$ $\boldsymbol{r}_3$ orthogonal

then where does the homography containing these row vectors move the epipole to?

$$\begin{bmatrix} r_1^T \\ r_2^T \\ r_3^T \end{bmatrix} e_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$, the point at infinity in the X direction!

In other words, such a homography would suffice to move the epipole off to infty (along image plane x-axis), which, we know ⇔ frontoparallel cameras (displaced along camera X-axis)!

Formula for "smallest rotation" $\boldsymbol{R_{rect}}$ (minimum distortion homography)

Let $$R_{\text{rect}} = \begin{bmatrix} \boldsymbol{r}_1^\top \\ \boldsymbol{r}_2^\top \\ \boldsymbol{r}_3^\top \end{bmatrix}$$

Given:

Epipole $\mathbf{e}_1 = T/||T||$,
where T from essential matrix E
By solving $ET = 0$ (smallest right singular vector of E)

Proof Sketch Part 1: Singular Values of A Valid Essential Matrix

$\boxed{\sigma_1(E) = \sigma_2(E) \neq 0 \text{ and } \sigma_3(E) = 0.}$

$$EE^T = \widehat{T}\widehat{T}^T = -\begin{bmatrix} t_x^2 & t_x t_y & t_x t_z \\ t_x t_y & t_y^2 & t_y t_z \\ t_x t_z & t_y t_z & t_z^2 \end{bmatrix} + \|T\|^2 I$$

If we solve the characteristic polynomial $\det(EE^T - \lambda I) = 0$ we will find two eigenvalues both equal to $\|T\|^2$. (Bonus exercise: try this out by hand)

So, $\sigma_1(E) = \sigma_2(E) = ||T||$ and $\sigma_3(E) = 0$

Side note: the 'third singular vector' of E (null vector because $\sigma_3 = 0$) is nothing but the translation vector T because:

$$EE^T T = \hat{T}\hat{T}^T T = T \times (-T \times T) = 0!$$

So we already know T in terms of E! (the 3rd left singular vector of E)

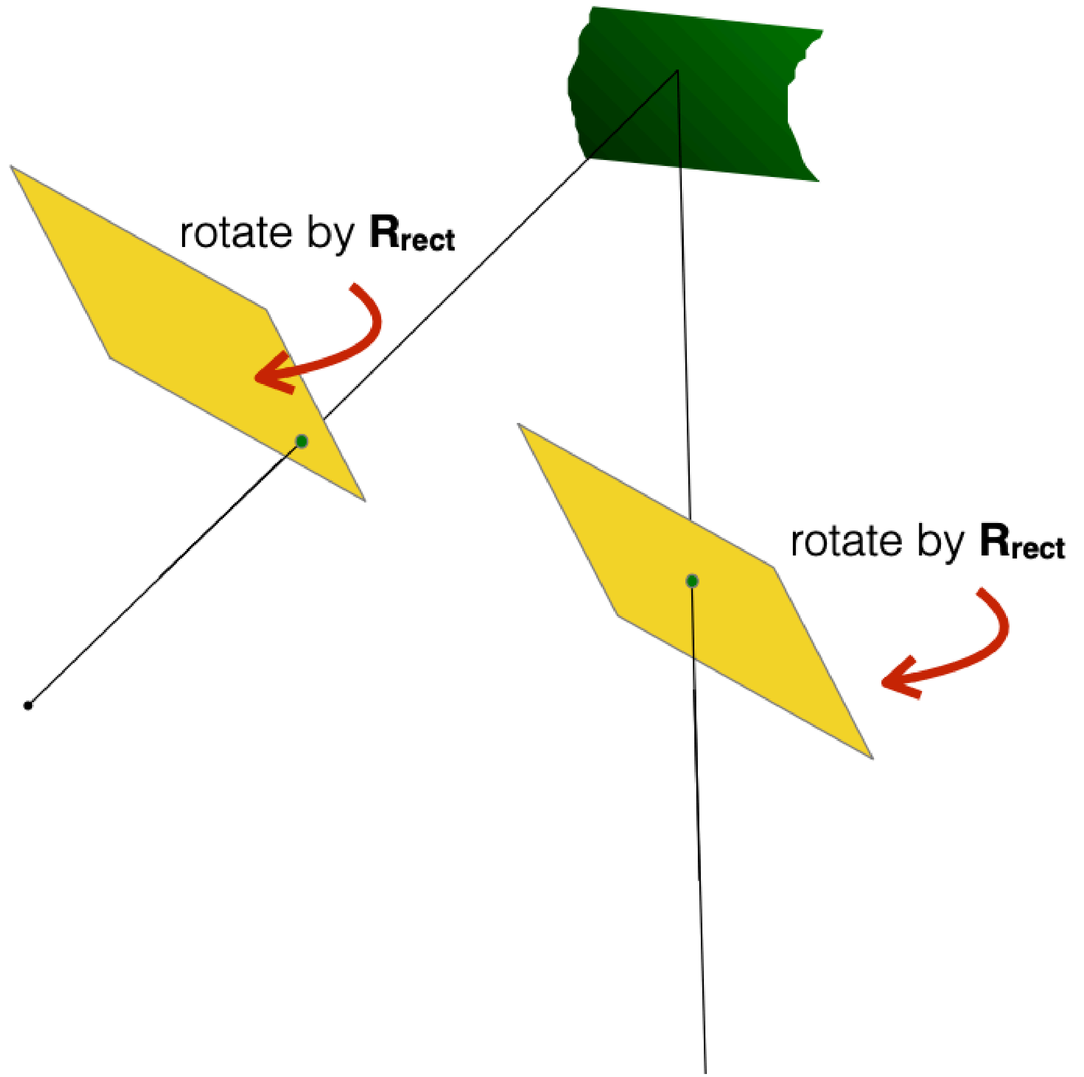
$$\boldsymbol{r}_1 = \boldsymbol{e}_1 = \frac{T}{||T||}$$

$$\boldsymbol{r}_2 = \frac{1}{\sqrt{T_x^2 + T_y^2}} \begin{bmatrix} -T_y & T_x & 0 \end{bmatrix}$$

$$\boldsymbol{r}_3 = \boldsymbol{r}_1 \times \boldsymbol{r}_2$$

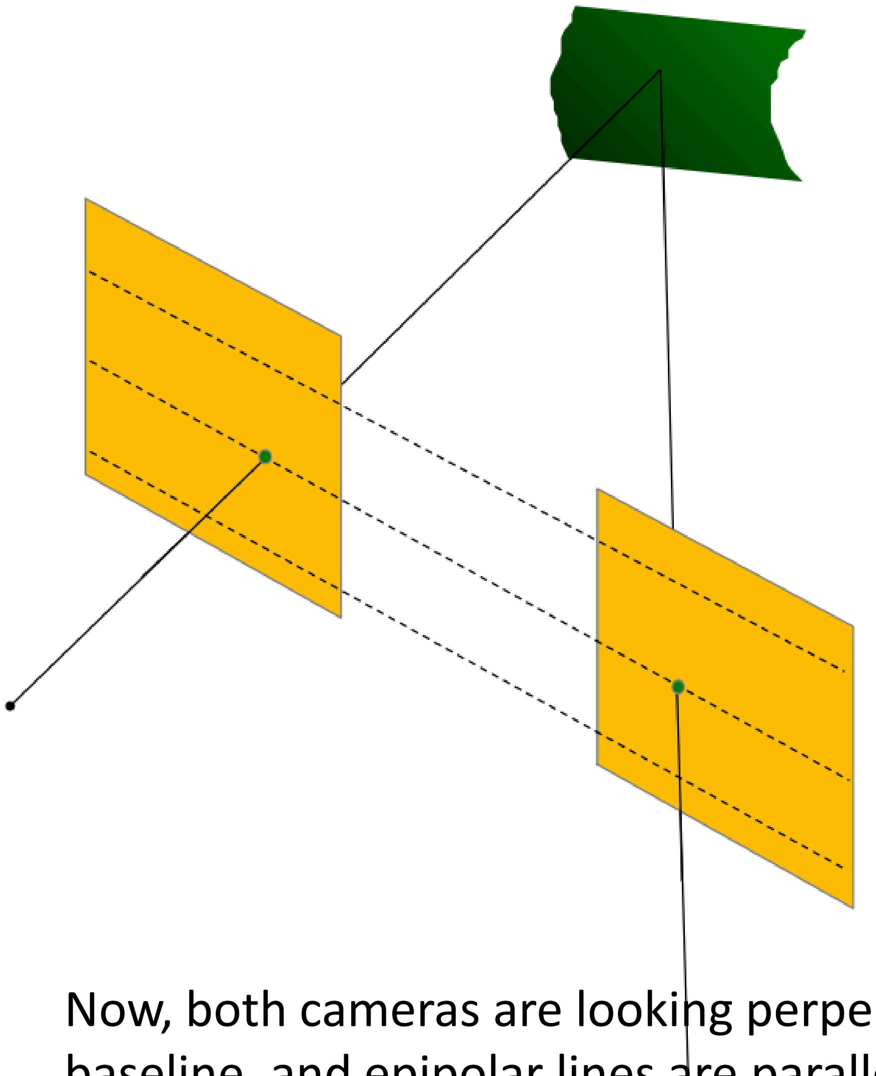
Q: Are r_2, r_3 actually perpendicular to $r_1 = e_1$ here (as required by previous slide)? Why?

Stereo Rectification:



1. Compute **E** to get **R**
2. Rotate right image by **R**
3. Rotate both images by $\mathbf{R}_{rect}$

Stereo Rectification:



1. Compute **E** to get **R**
2. Rotate right image by **R**
3. Rotate both images by $\mathbf{R}_{\mathbf{rect}}$

Now, both cameras are looking perpendicular to the baseline, and epipolar lines are parallel and horizontal!

Stereo Rectification Algorithm Pseudocode

1. Estimate E using 8-point algorithm
2. Decompose E into R and $T \sim \boldsymbol{e}$
3. Build R_{rect} from $\boldsymbol{e}$
4. Set $R_1 = R_{rect}$ and $R_2 = RR_{rect}$
5. Finally, on left and right camera pixel planes, apply the homographies: KR_1 and KR_2 respectively*

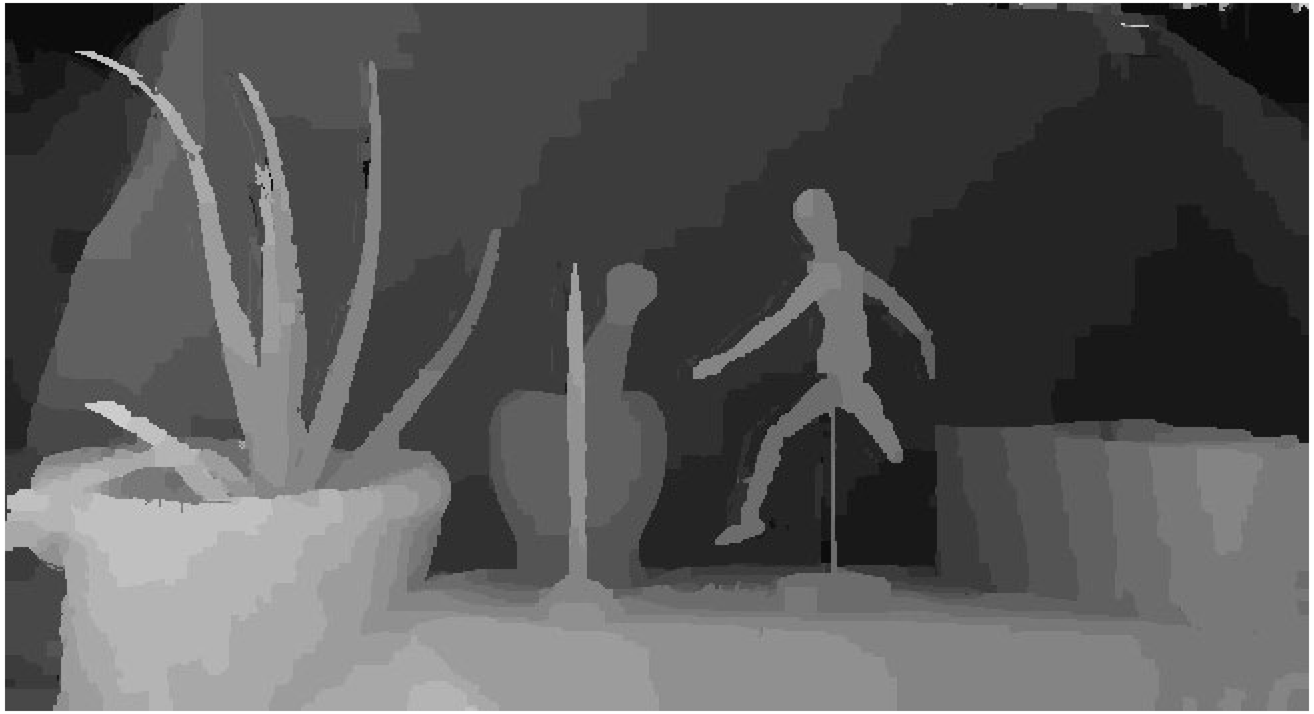
*You may need to alter K to keep points within the original image size

After Rectification

- And now after rectification, we are back in the frontoparallel setting.
- So, can find dense correspondences by searching along horizontal scanlines e.g. using SSD on windows and applying winner-take-all matching.
- And then, use the parallel camera triangulation equation to find depths $Z = f\frac{B}{d}$, at all the dense correspondence points

And thus, dense 3D!

Dense Disparity / Depth Maps



Left Image
Right Image

View Interpolation

Note: the interpolation here is from left image, black regions where pixels are disoccluded between left and right images.

Multi-View Stereo

Based on slides by Noah Snavely

Our strategy

- First deal with dense correspondence finding for the frontoparallel 2-camera case

- Then see how to “rectify” non-frontoparallel cameras to be frontoparallel.

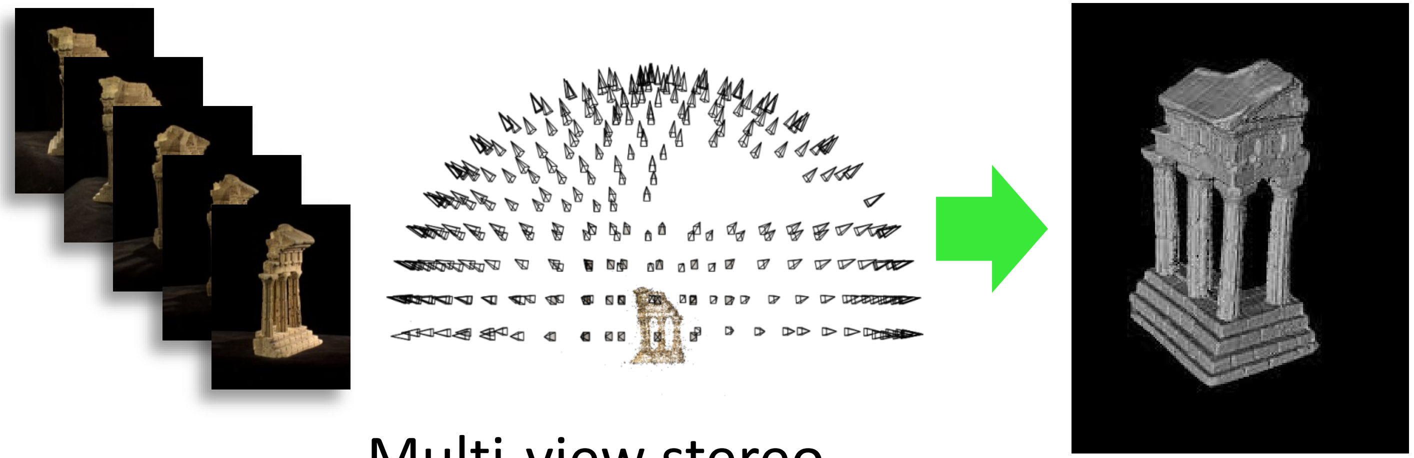
- Then, how to perform multi-view stereo (MVS)
  - Straightforward multi-baseline extension of 2-view stereo
  - The “plane sweep” technique for MVS

- Finally, improvement through dynamic programming.

Multi-view Stereo

Problem formulation: given several images of the same object or scene, compute a representation of its 3D shape

Binocular Stereo



Multi-view stereo

Multi-view Stereo Camera Systems

Point Grey's Bumblebee XB3

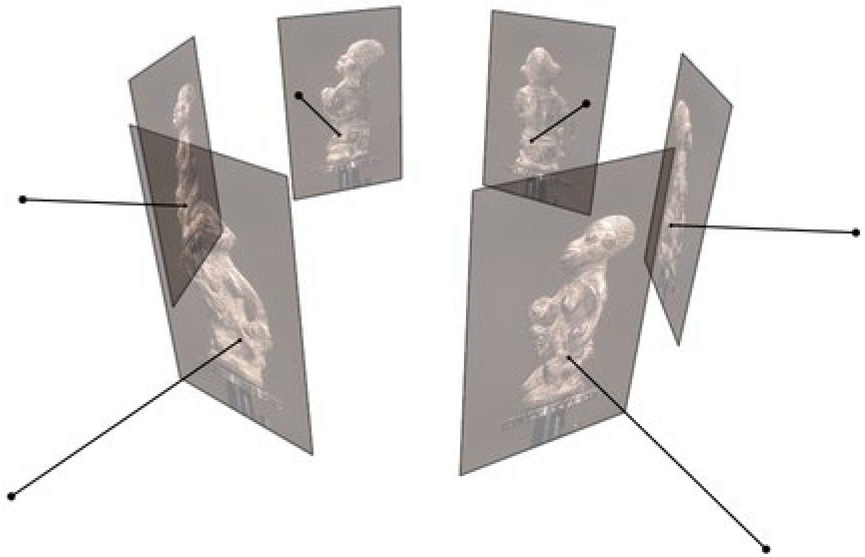
Point Grey's ProFusion 25

CMU's Panoptic Studio

Multi-view Stereo

Input: calibrated images from several viewpoints (known intrinsics and extrinsics / projection matrices)

Output: 3D object model



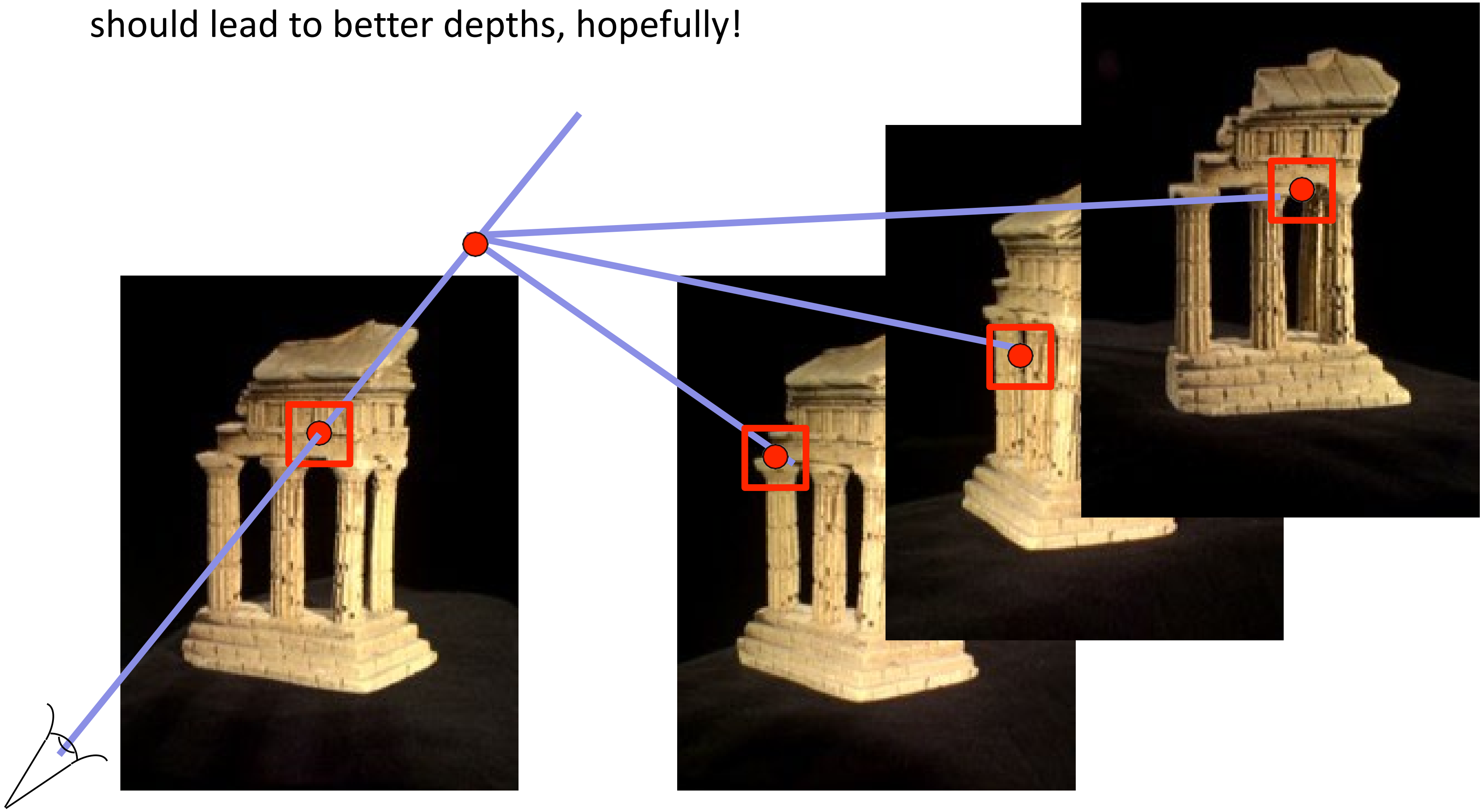
Figures by Carlos Hernandez

Dense maps of cities with MVS

Dense models of historical artifacts with MVS

Extending 2-view correspondences to multiple views

When trying to figure out the depth of a pixel in a reference view, we now have more than just 1 additional view. More information should lead to better depths, hopefully!



reference view

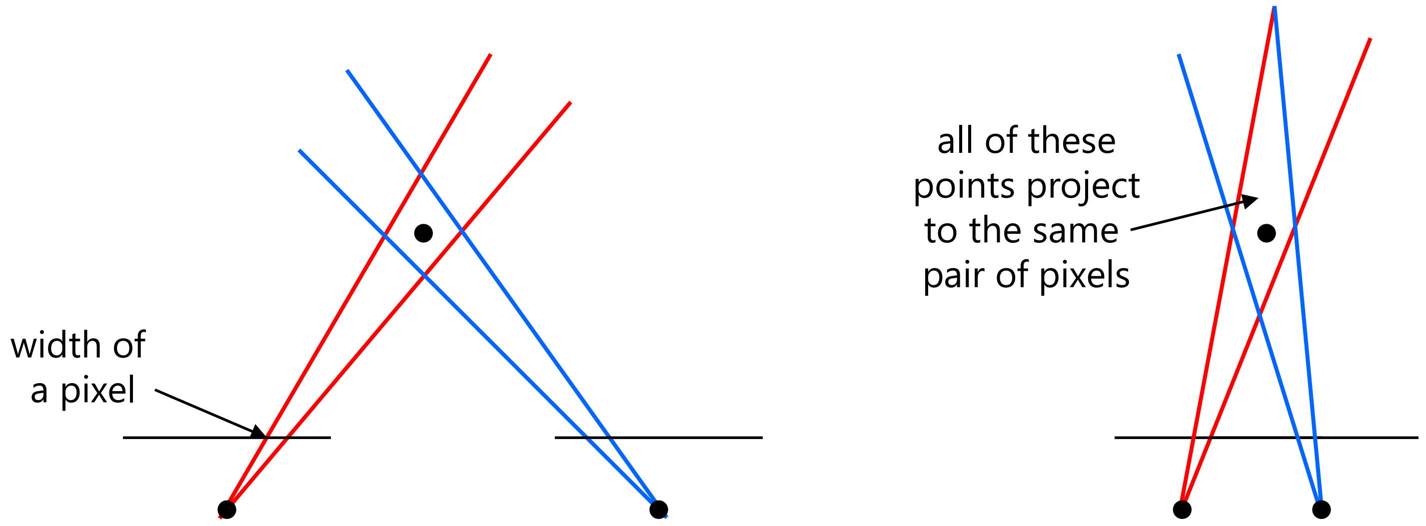
neighbor views

Source: Y. Furukawa

Exploiting multiple neighbors

- Can match windows using more than 1 neighboring view, giving a **stronger match signal**
- If you have lots of potential neighbors, can **choose the best subset** of neighbors to match per reference image
- Can reconstruct a depth map for each reference frame, and then merge into a **complete 3D model**

Pick Neighbors Based On Baseline?



Large Baseline **Small Baseline**

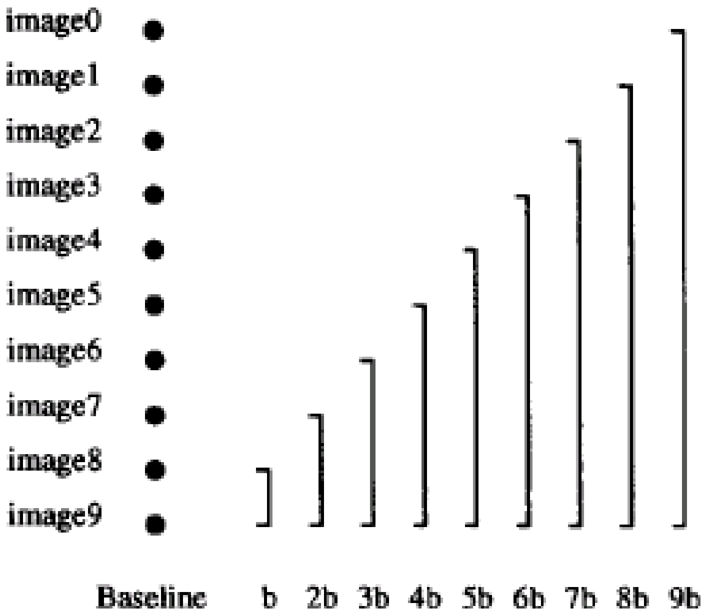
(Recall that we have already seen this when discussing ORB-SLAM)

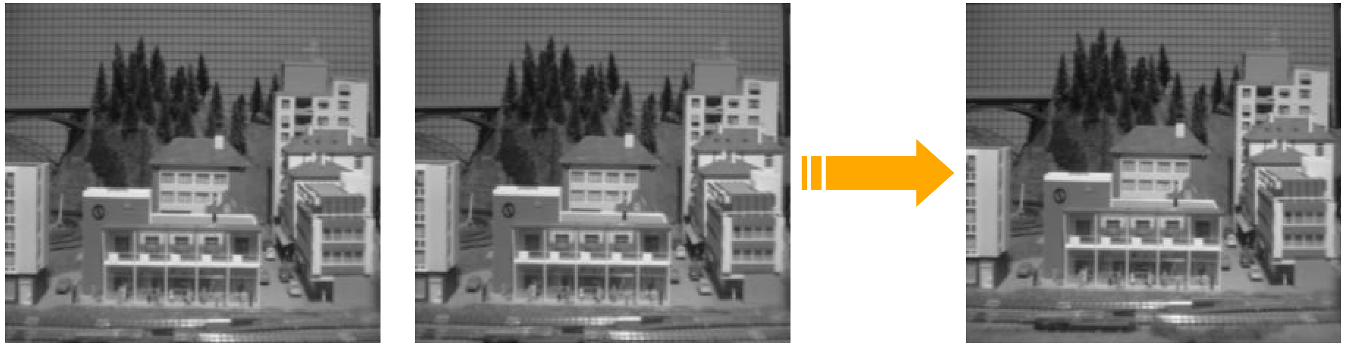
What's the optimal baseline?

- Too small: large depth error
- Too large: difficult search problem

The Effect of Baseline on Depth Estimation

Figure 2: An example scene. The grid pattern in the background has ambiguity of matching.

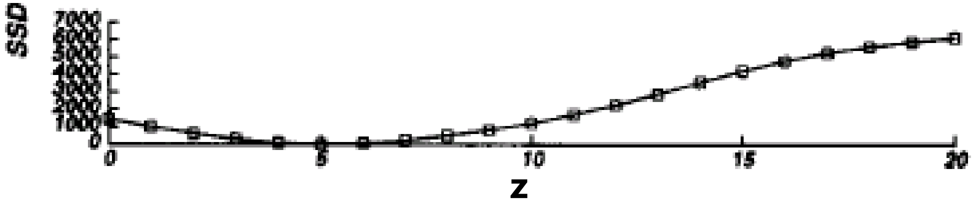




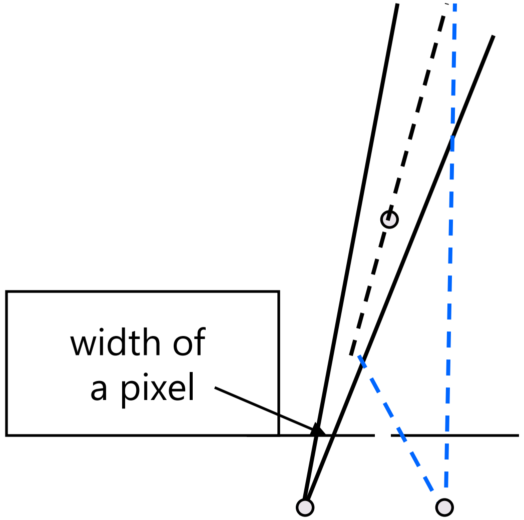
I_1 I_2 I_{10}

What if we could use all baselines together somehow?

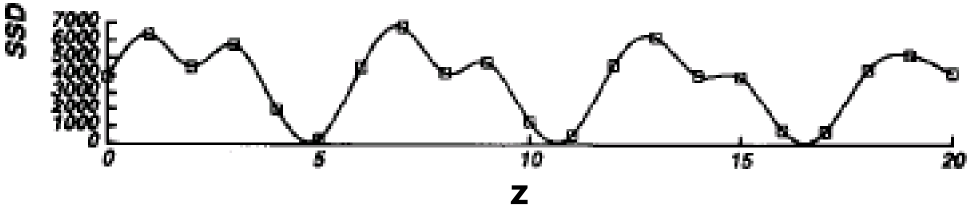
Different Baselines Have Different Problems

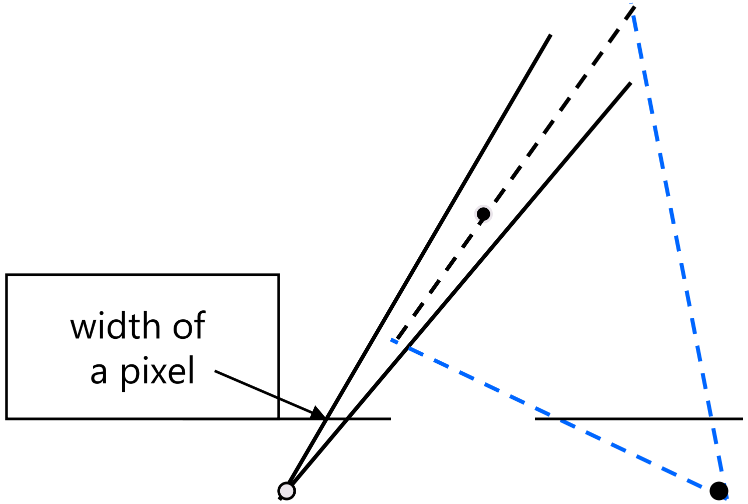


pixel matching score



- For short baselines, estimated depth will be less precise due to narrow triangulation





- For larger baselines, must search larger area in second image

M. Okutomi and T.Kanade, "A Multiple-Baseline Stereo System," IEEE Trans. on Pattern Analysis and Machine Intelligence, 15(4):353-363, 1993.

Simple Solution: Combine Them All!

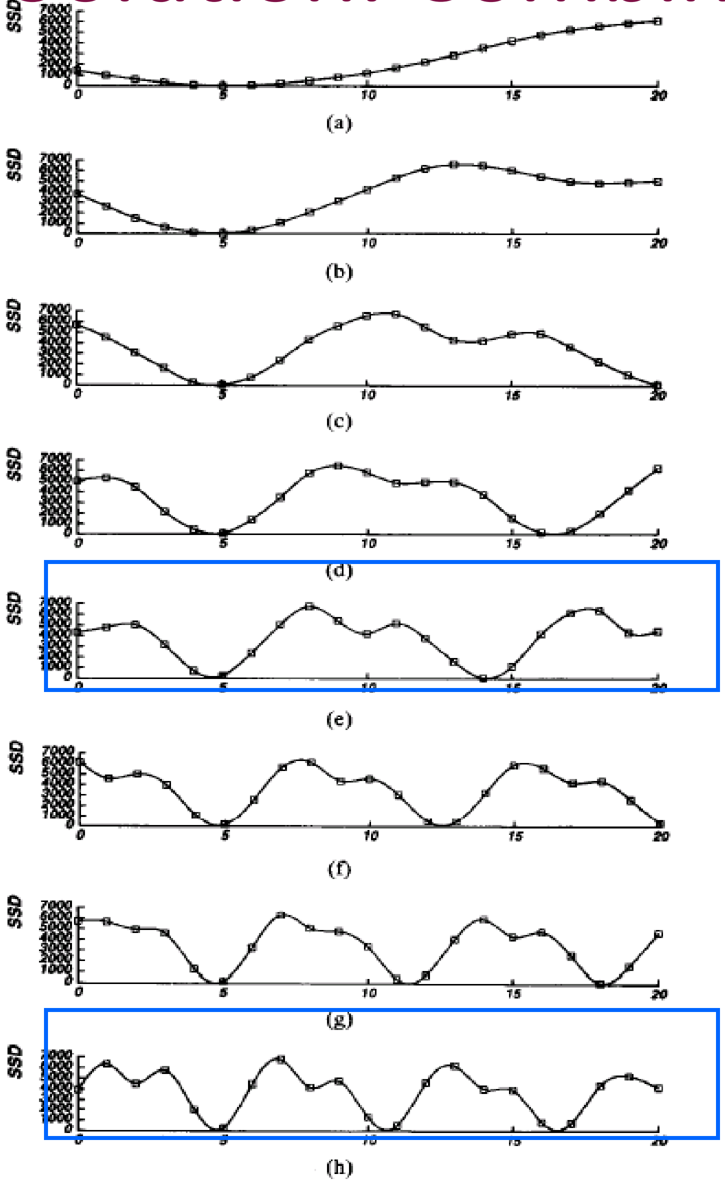


Fig. 5. SSD values versus inverse distance: (a) $B = b$; (b) $B = 2b$; (c) $B = 3b$; (d) $B = 4b$; (e) $B = 5b$; (f) $B = 6b$; (g) $B = 7b$; (h) $B = 8b$. The horizontal axis is normalized such that $8bF = 1$.

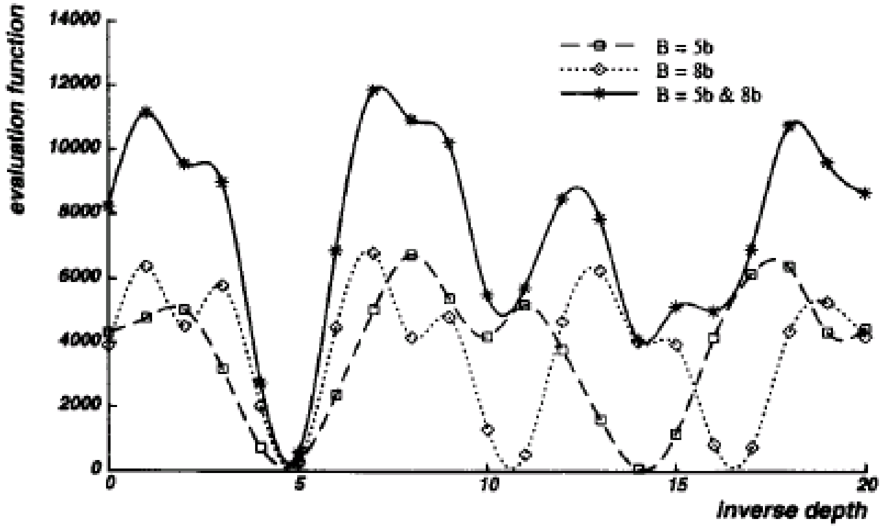


Fig. 6. Combining two stereo pairs with different baselines.

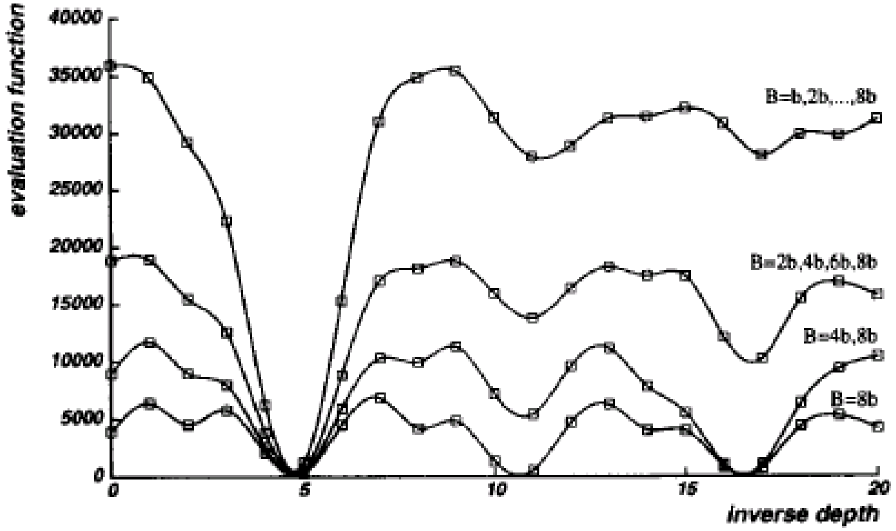
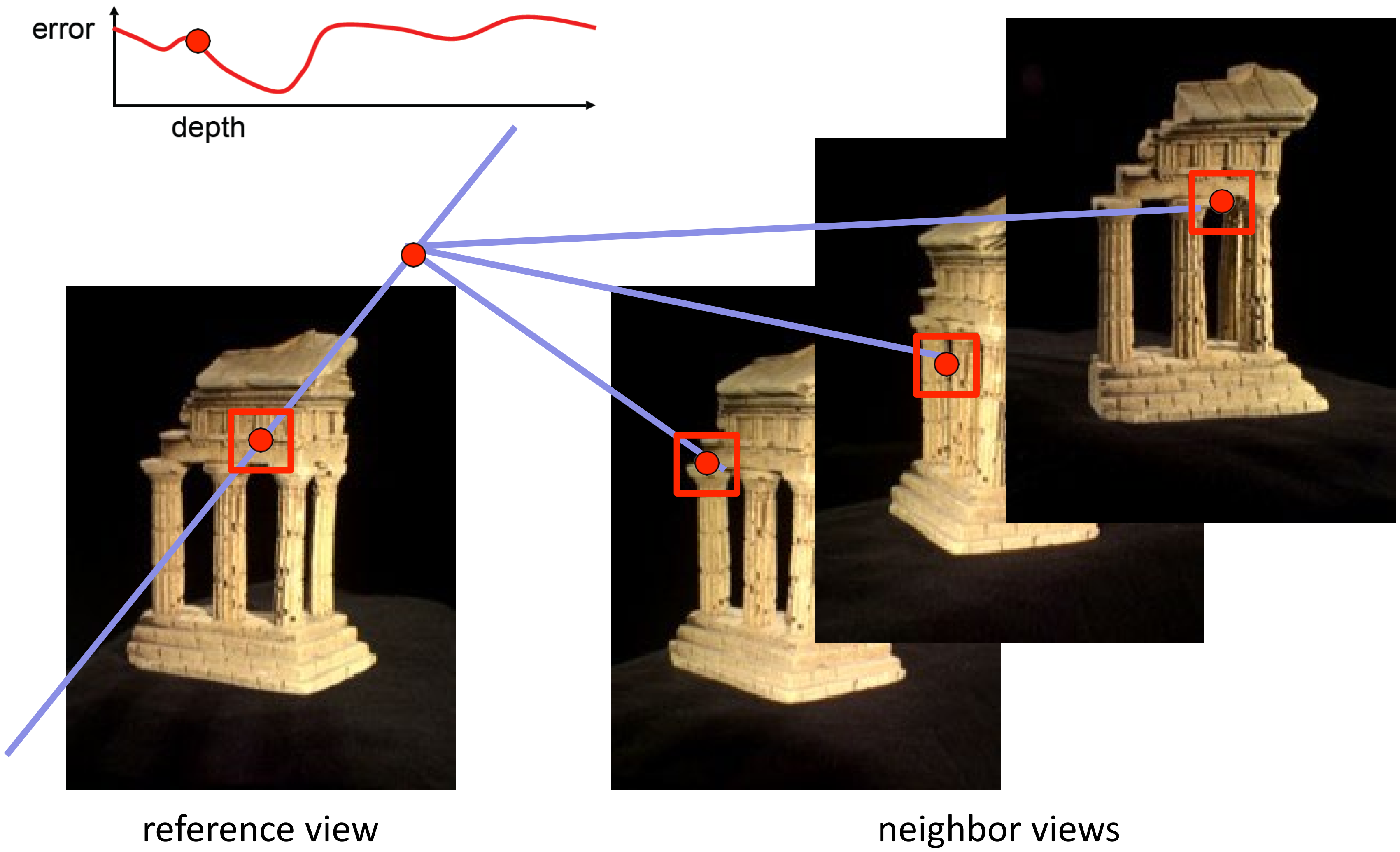


Fig. 7. Combining multiple baseline stereo pairs.

Multiple Baseline Sum of SSD errors

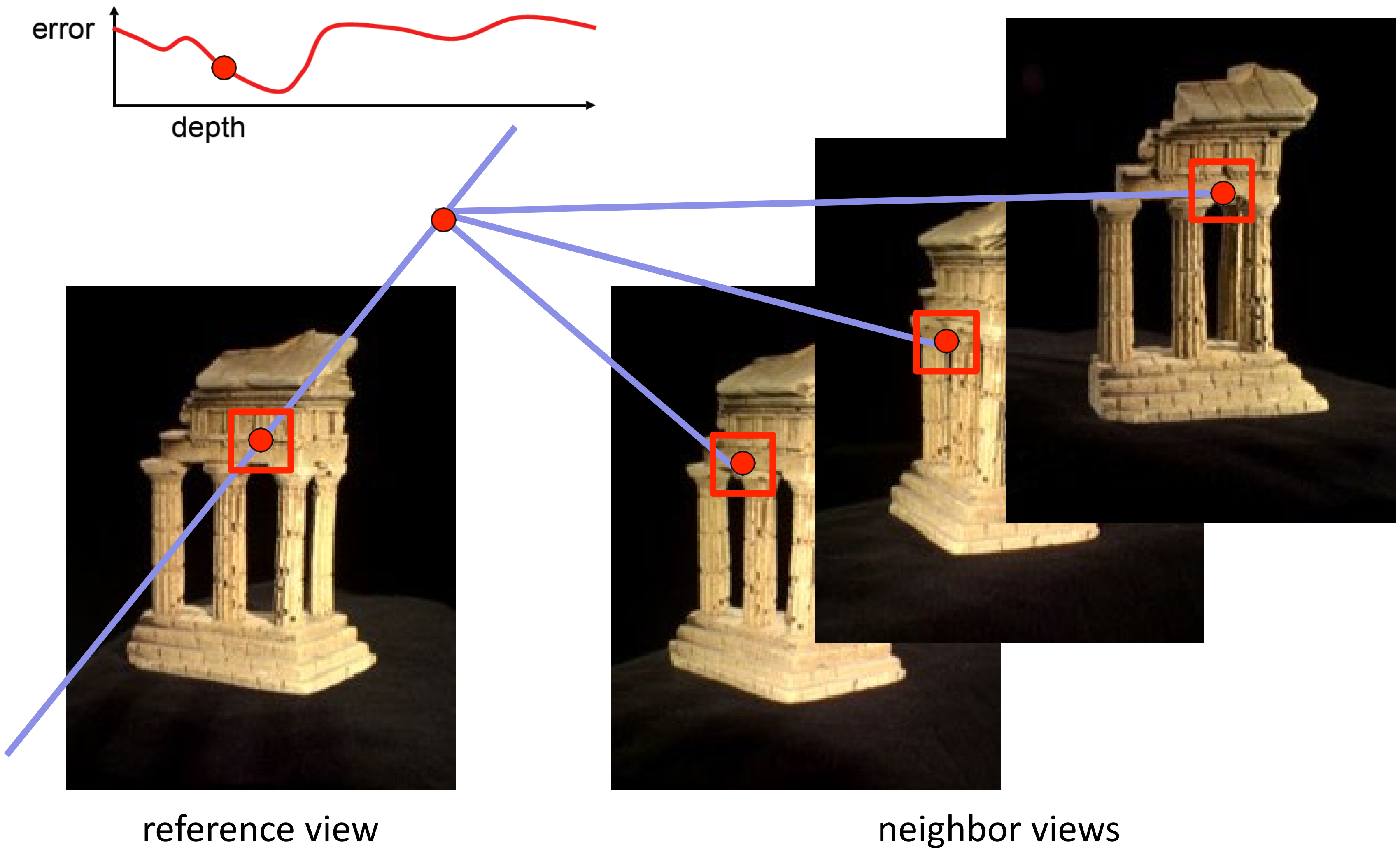
Error aggregated over all (reference 0, neighbor i) pairs



Source: Y. Furukawa

Multiple Baseline Sum of SSD errors

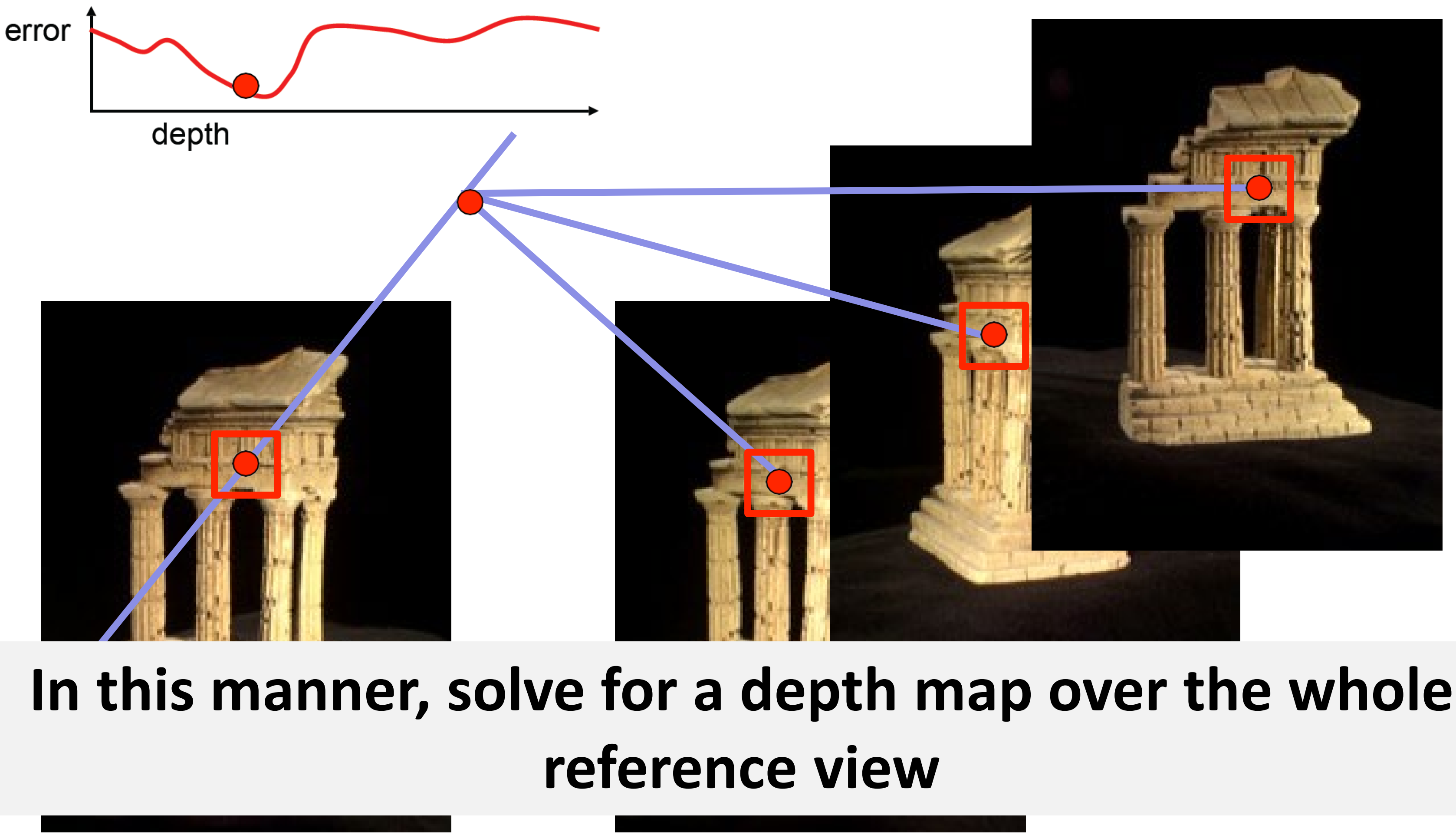
Error aggregated over all (reference 0, neighbor i) pairs



Source: Y. Furukawa

Multiple Baseline Sum of SSD errors

Error aggregated over all (reference 0, neighbor i) pairs



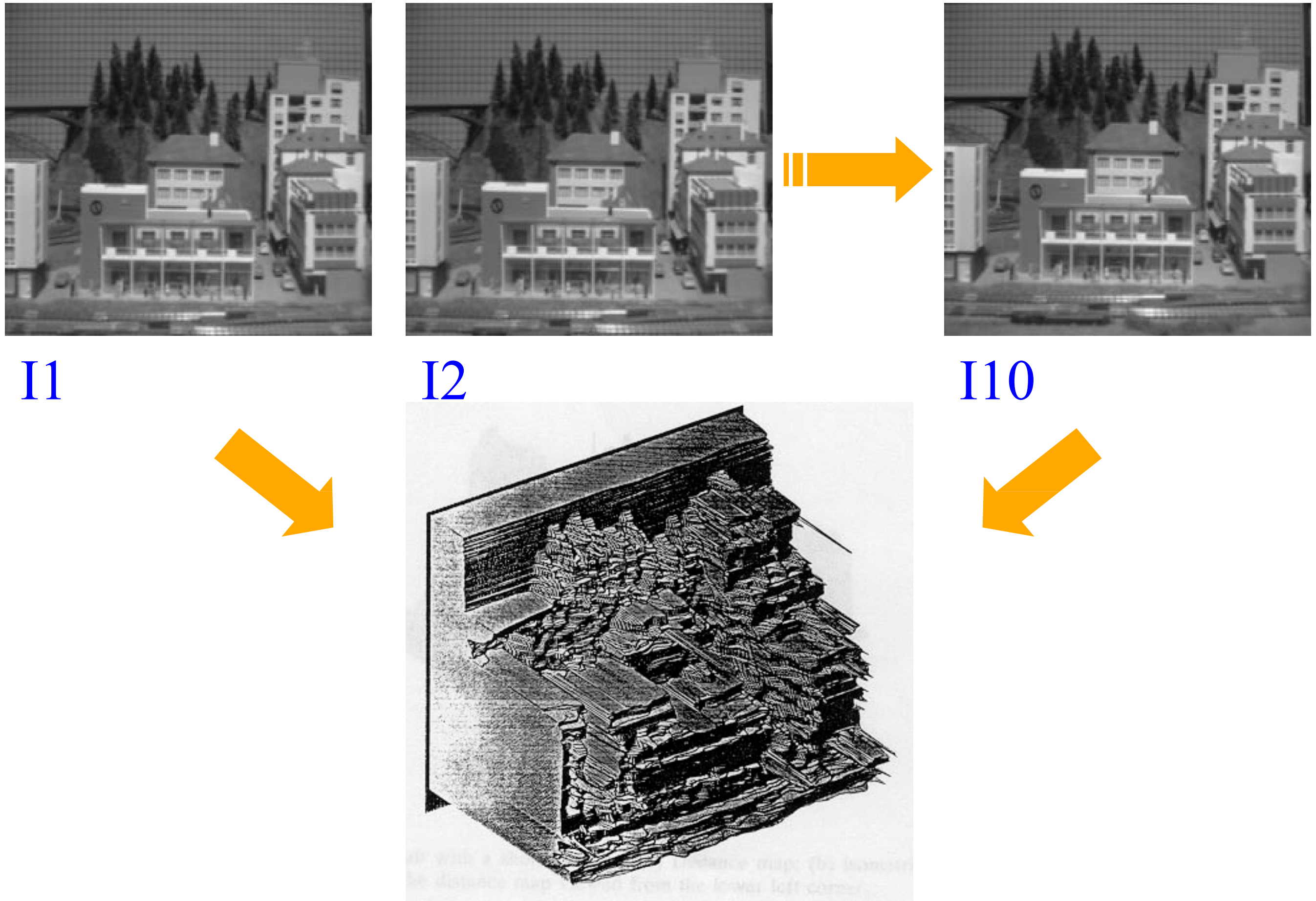
In this manner, solve for a depth map over the whole reference view

reference view

neighbor views

Source: Y. Furukawa

Multiple-Baseline Multi-View Stereo Results



M. Okutomi and T. Kanade, *A Multiple-Baseline Stereo System*, IEEE Trans. on Pattern Analysis and Machine Intelligence, 15(4):353-363 (1993).

Multiple-Baseline Multi-View Stereo Summary

Basic Approach

- Choose a reference view
- Use your favorite stereo algorithm BUT
  - replace two-view SSD with **SSSD** (sum of sums of squared distances) over ***all baselines***
  - **SSSD**: the SSD values are computed first for each pair of stereo images, and then add all together from multiple stereo pairs.

Limitations

- Won’t work for widely distributed views.

Our strategy

- First deal with dense correspondence finding for the frontoparallel 2-camera case

- Then see how to “rectify” non-frontoparallel cameras to be frontoparallel.

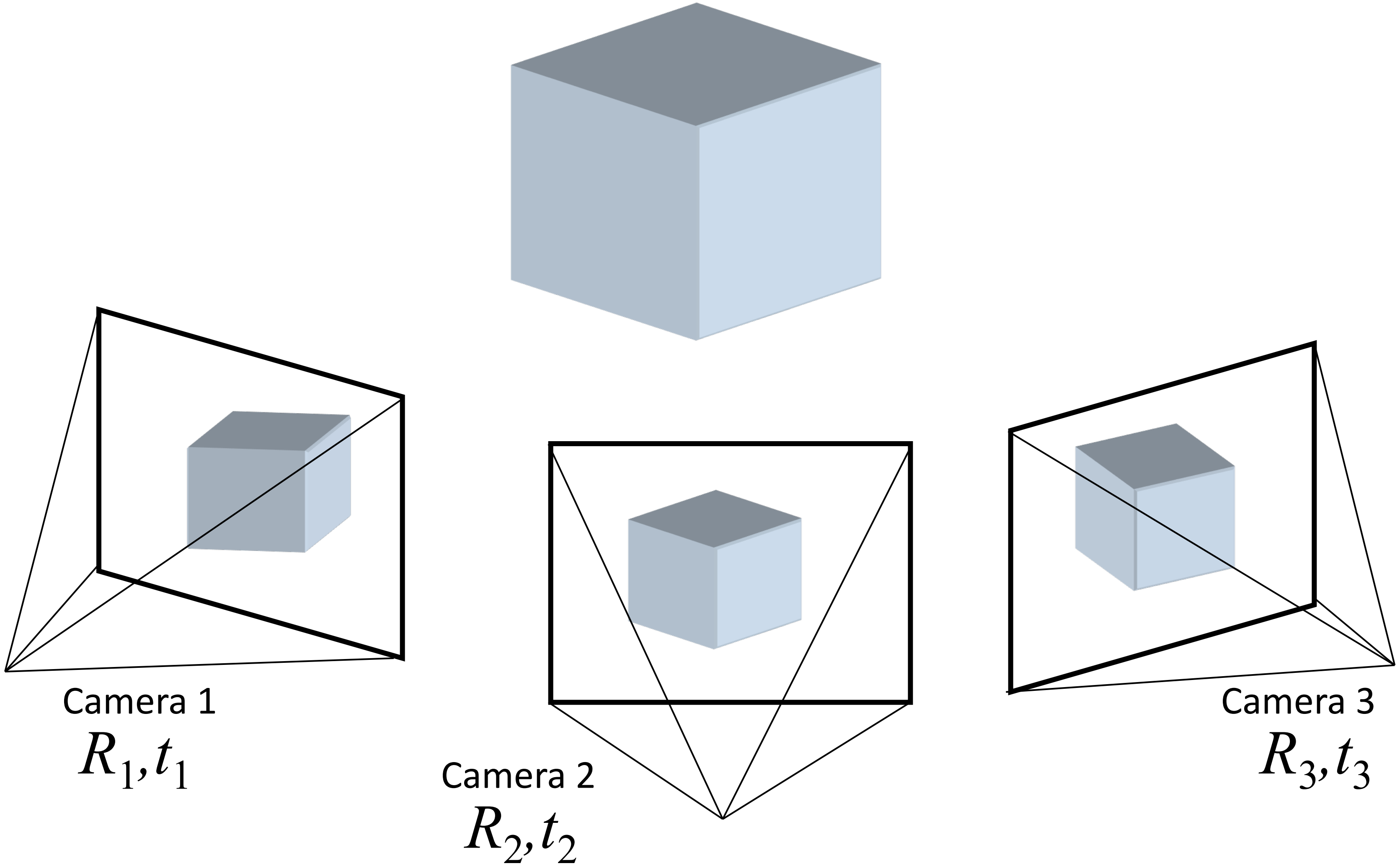
- Then, how to perform multi-view stereo (MVS)
  - Straightforward multi-baseline extension of 2-view stereo
  - The “plane sweep” technique for MVS

- Finally, improvement through dynamic programming.

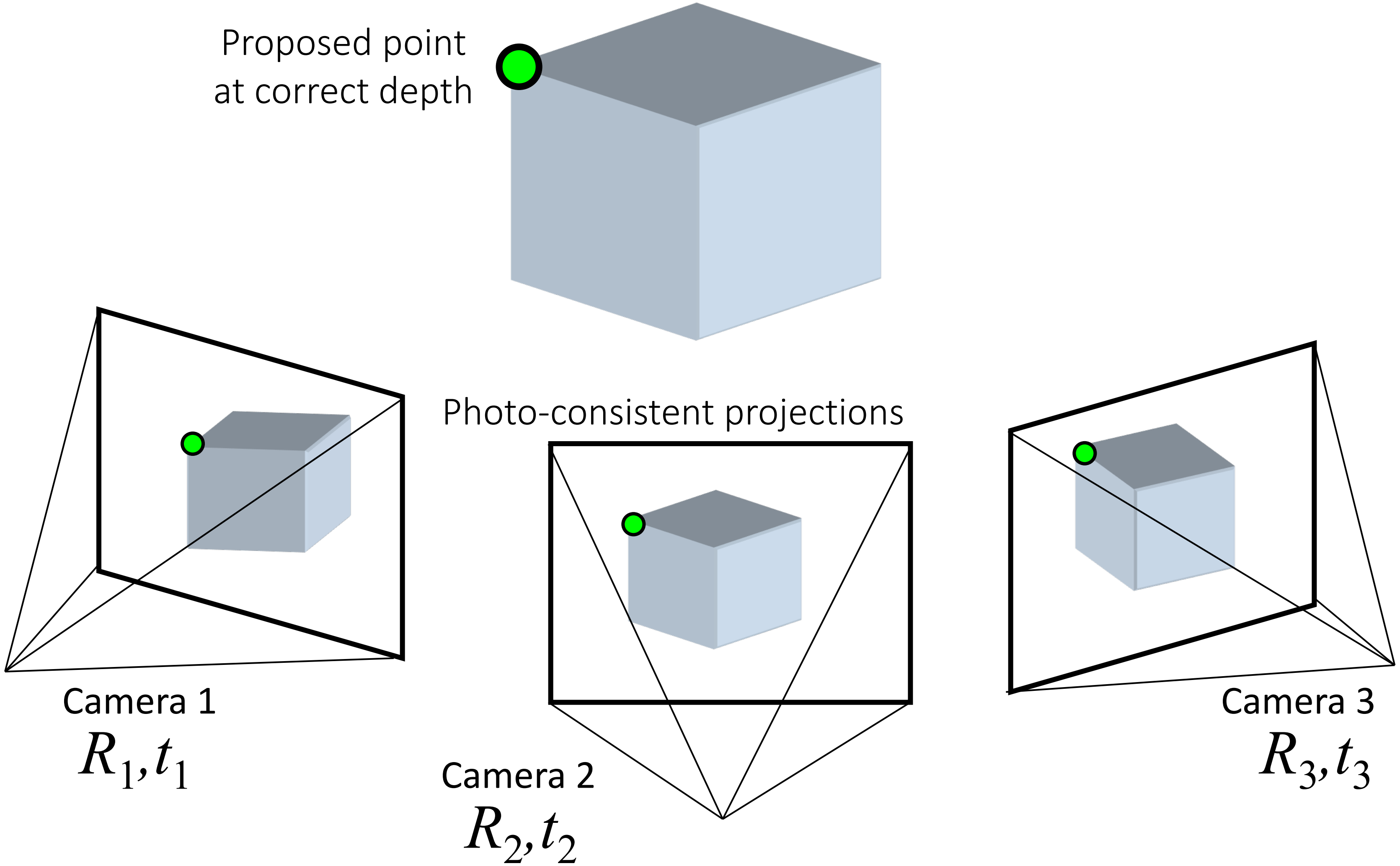
Plane Sweep

Plane-Sweep Stereo

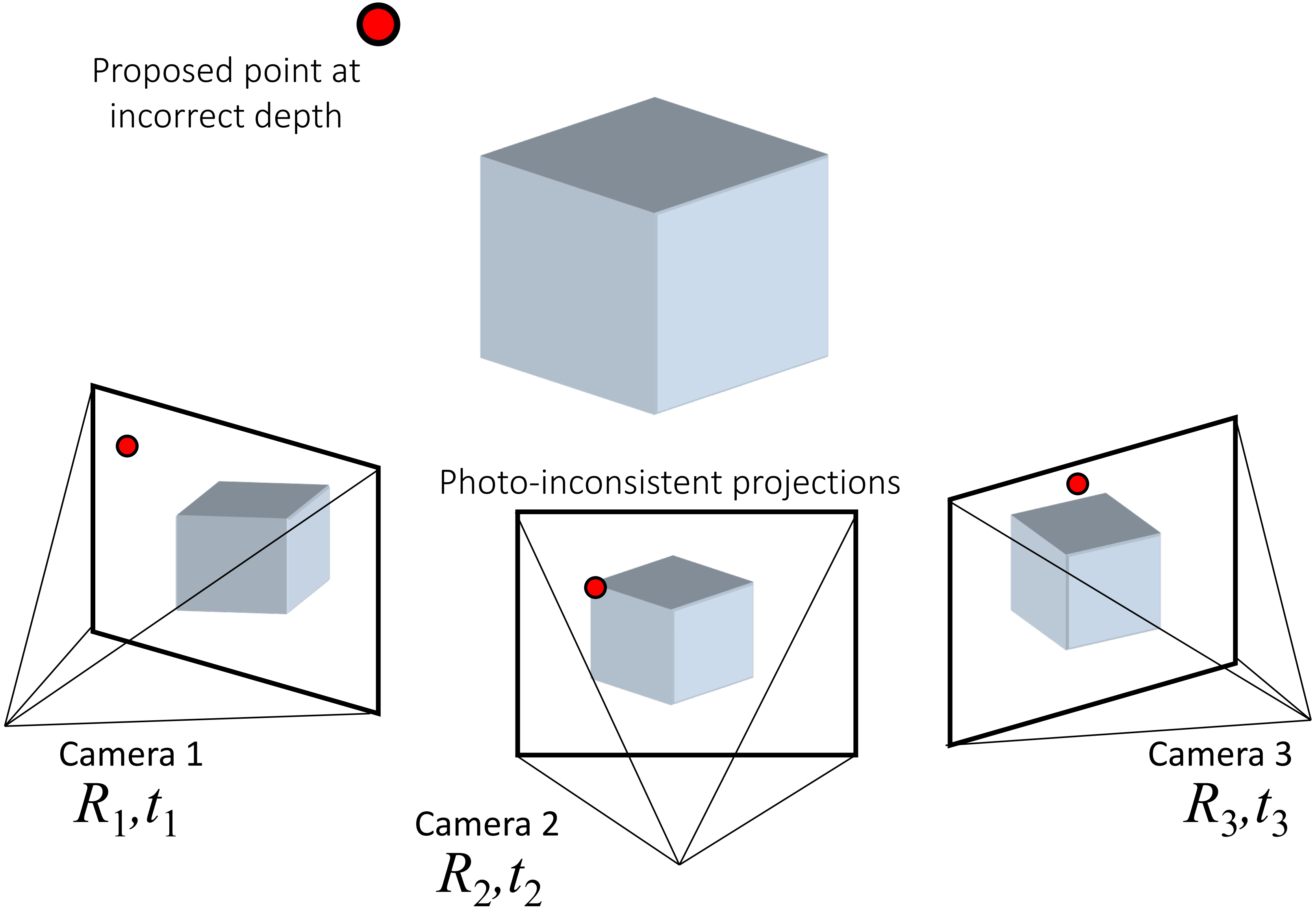
An efficient way to compute multi-view stereo



Plane-Sweep Stereo

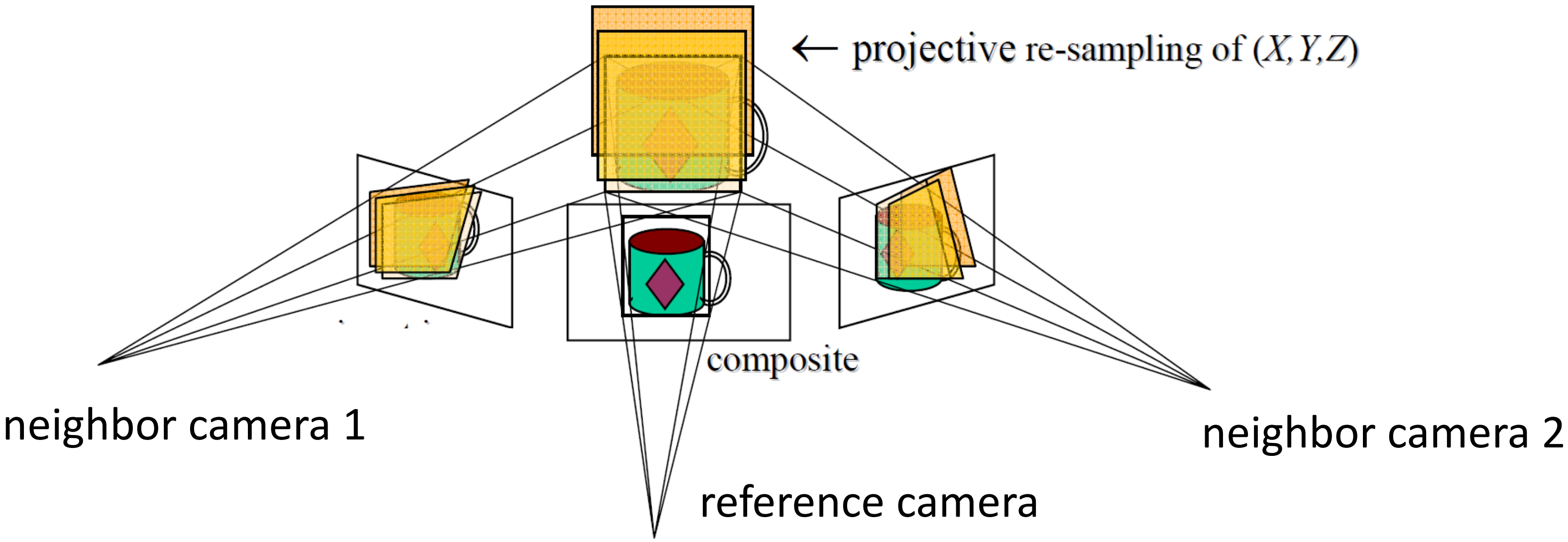


Plane-Sweep Stereo



Plane-Sweep Stereo

- At each iteration, we pretend that each camera's entire image was of a single plane at depth z from the reference camera, and backproject onto that plane from each camera, and see how much the neighbors agree, for each pixel.
- **When neighbors agree at a pixel, that pixel is likely to have depth z_0. The pixel's "cost" for depth z is the variance over neighbor backprojections.**
- Then z is incremented and the next iteration begins!
- At the end of the "sweep" over z, the min-cost z is selected for each pixel, to form the full dense depth map



Plane-Sweep Stereo

Blurriness in the average images => more disagreement.

Left neighbor

Reference image

Right neighbor

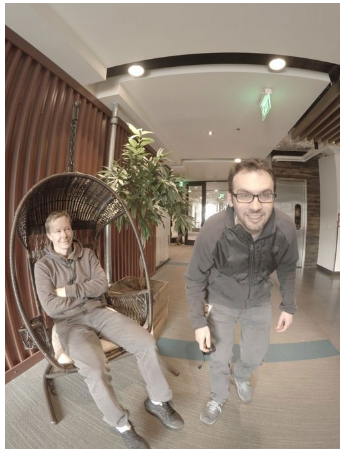
Gifs show increasing depth z

Left neighbor projected into reference camera's Z=z plane

Average image on the reference camera's Z=z plane

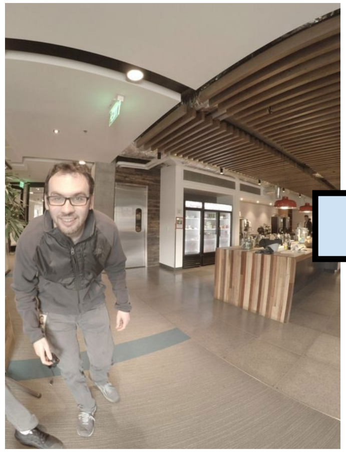
Right neighbor projected into reference camera's Z=z plane

Another example

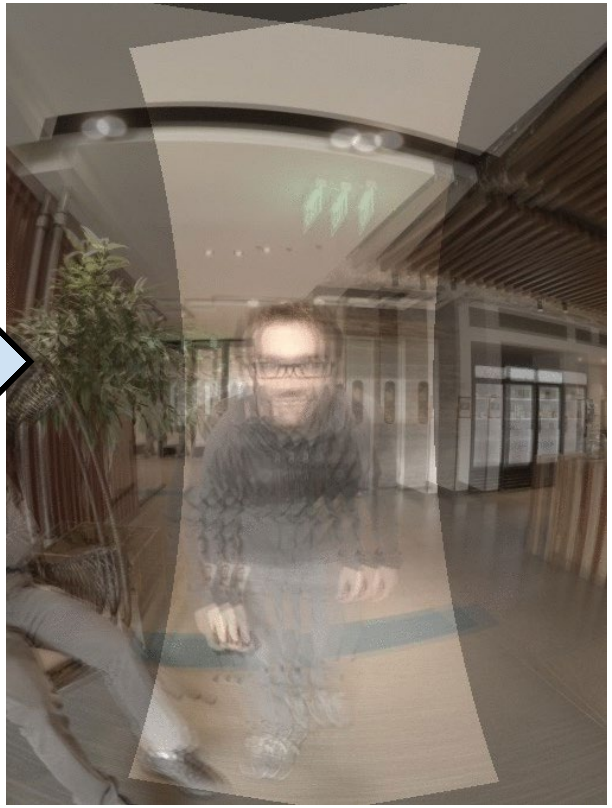


Left neighbor

Reference image

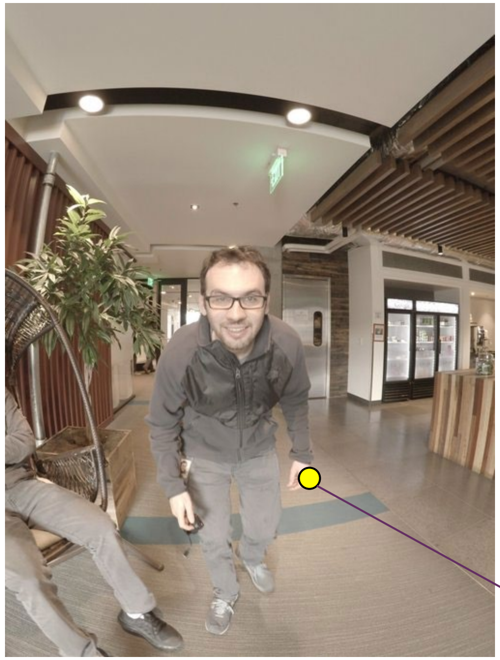


Right neighbor



Planar image reprojections swept over depth (averaged)

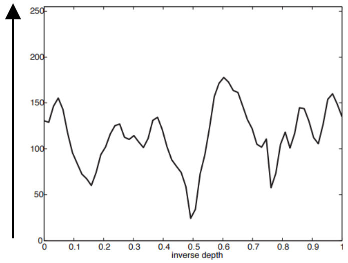
Cost Volumes -> Depth Maps



Reference image

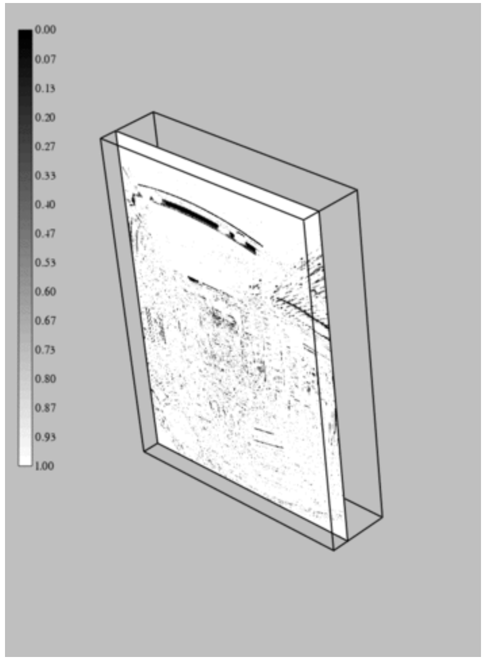
Plane sweep

cost



depth

Single pixel's cost profile



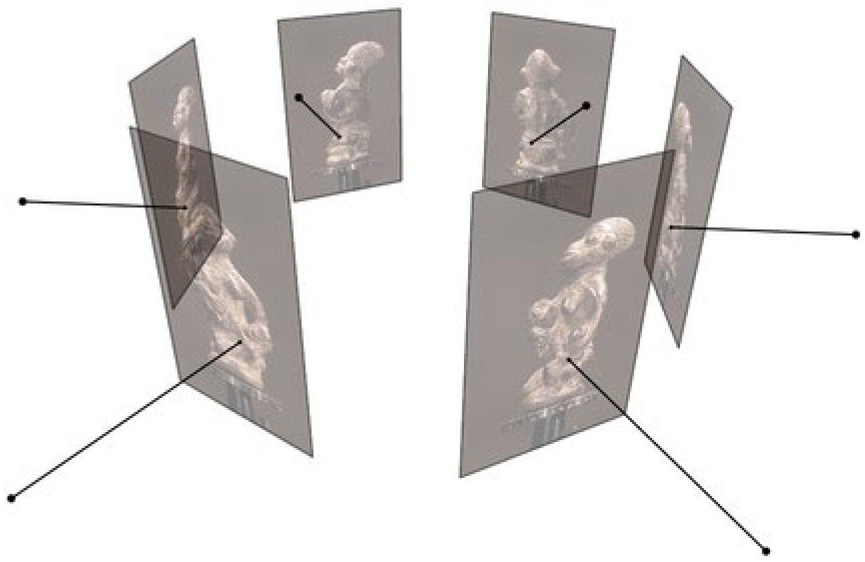
Full cost volume

Depth map solver

(Belief propagation, graph cuts, etc.)

Fusing multiple depth maps

- Compute depth map per image
- Fuse the depth maps into a 3D model



Figures by Carlos Hernandez

Note on Visibility in MVS

- When backprojecting in this fashion and measuring disagreement to check for whether a point is at the right depth plane, we are assuming that disagreement can only arise from projecting to the wrong depth.
- In reality, other possibilities:
  - specular/shiny objects that might look different from different angles
  - Occlusions! Not every point is even visible in every camera to start with, so often MVS requires jointly estimating visibility *and* dense correspondences.
    - For example, if there is large agreement among one subset of views, but large disagreement among others, this may indicate occlusion in the other views.