CIS 5800

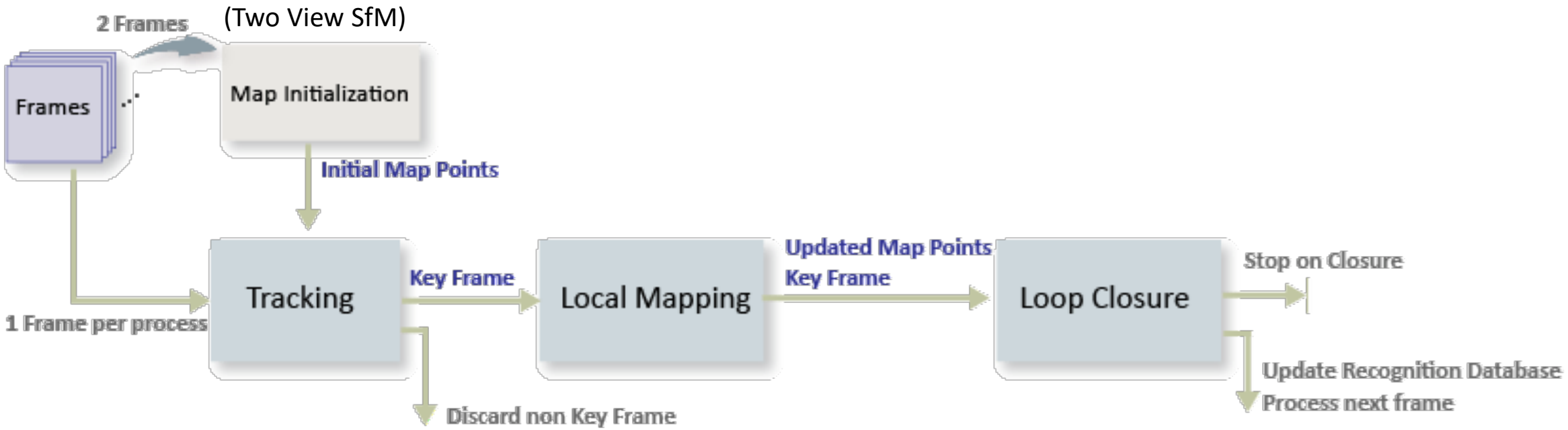
Machine Perception

Instructor: Lingjie Liu

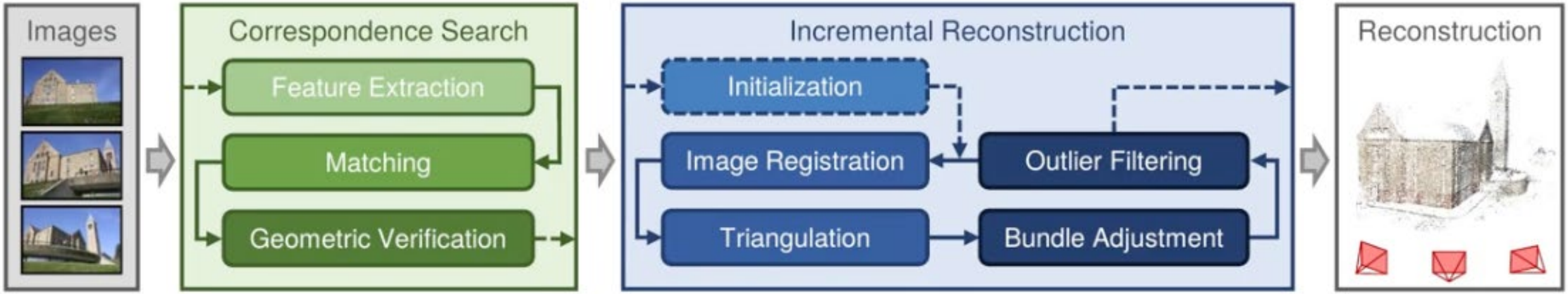
Lec 21: April 21, 2025

Robot Image Credit: Viktoriya Sukhanova © 123RF.com

Recap: ORB-SLAM Pipeline



Recap: Structure from Motion



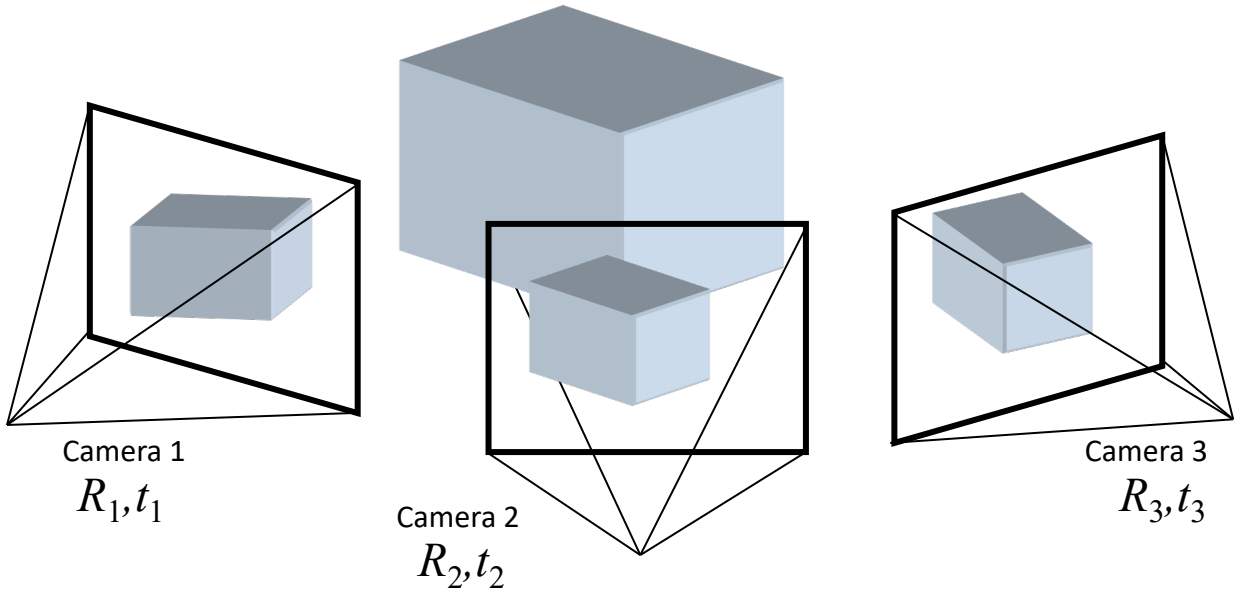
BA is often the final step in SfM, used to *refine* camera poses and 3D points starting from some (pretty good) initialization.

Recap: Bundle Adjustment Objective Function

- So what is the error that bundle adjustment tries to minimize?
- **“Reprojection error”**: The sum of errors between 2D observations and the “re-projected” 2D points.

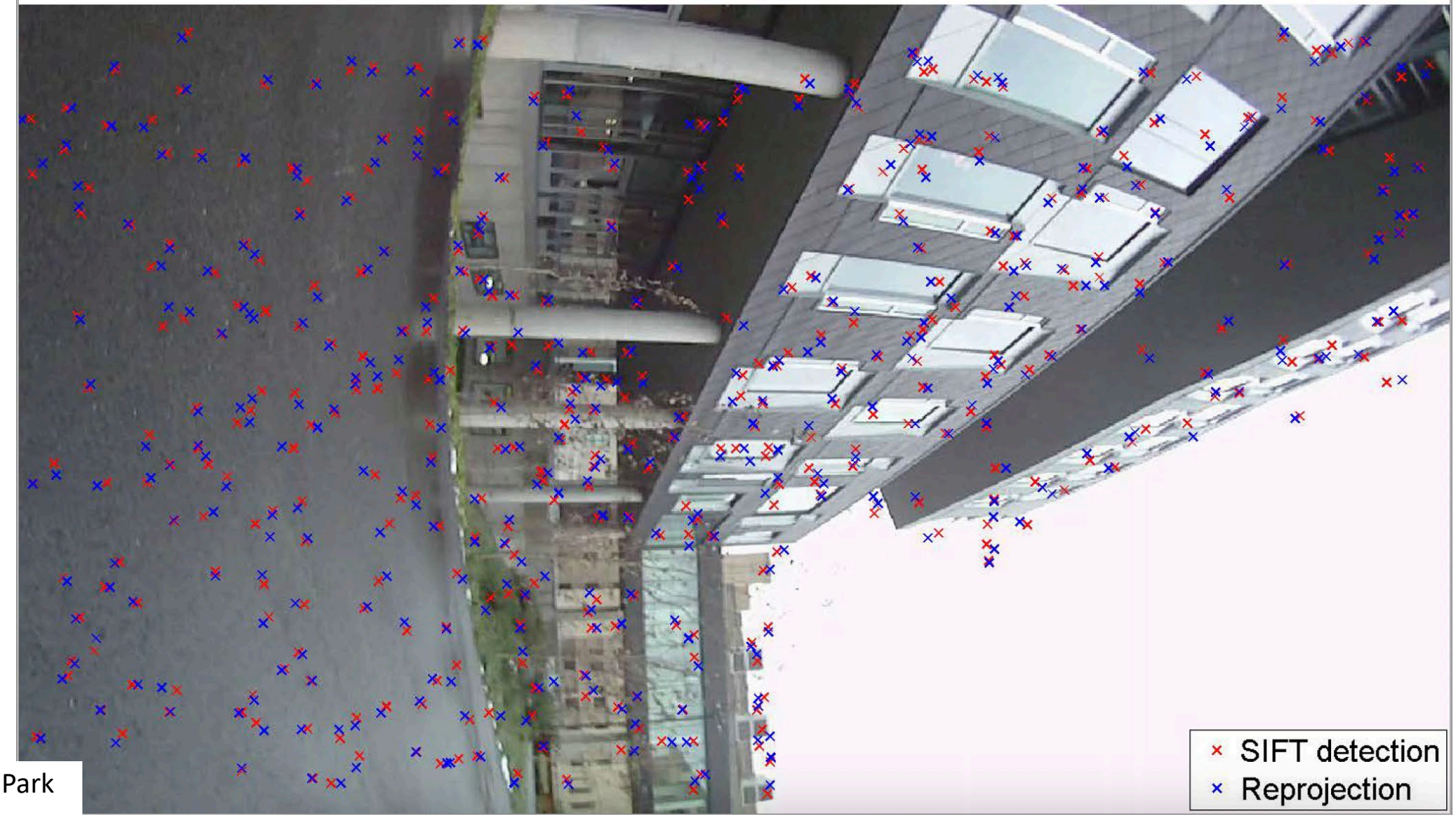
$$\underset{\{X_n\}_{n=1}^{N},\{R_f,T_f\}_{f=1}^{F}}{\operatorname{argmin}} \sum_{n,f} d(\boldsymbol{x}_{nf}, \quad K_f[R_f|T_f]\boldsymbol{X_n})$$

$d(.)$ is usually a simple 2D mean squared error after normalizing to homogeneous coordinate w=1.



Geometric Refinement

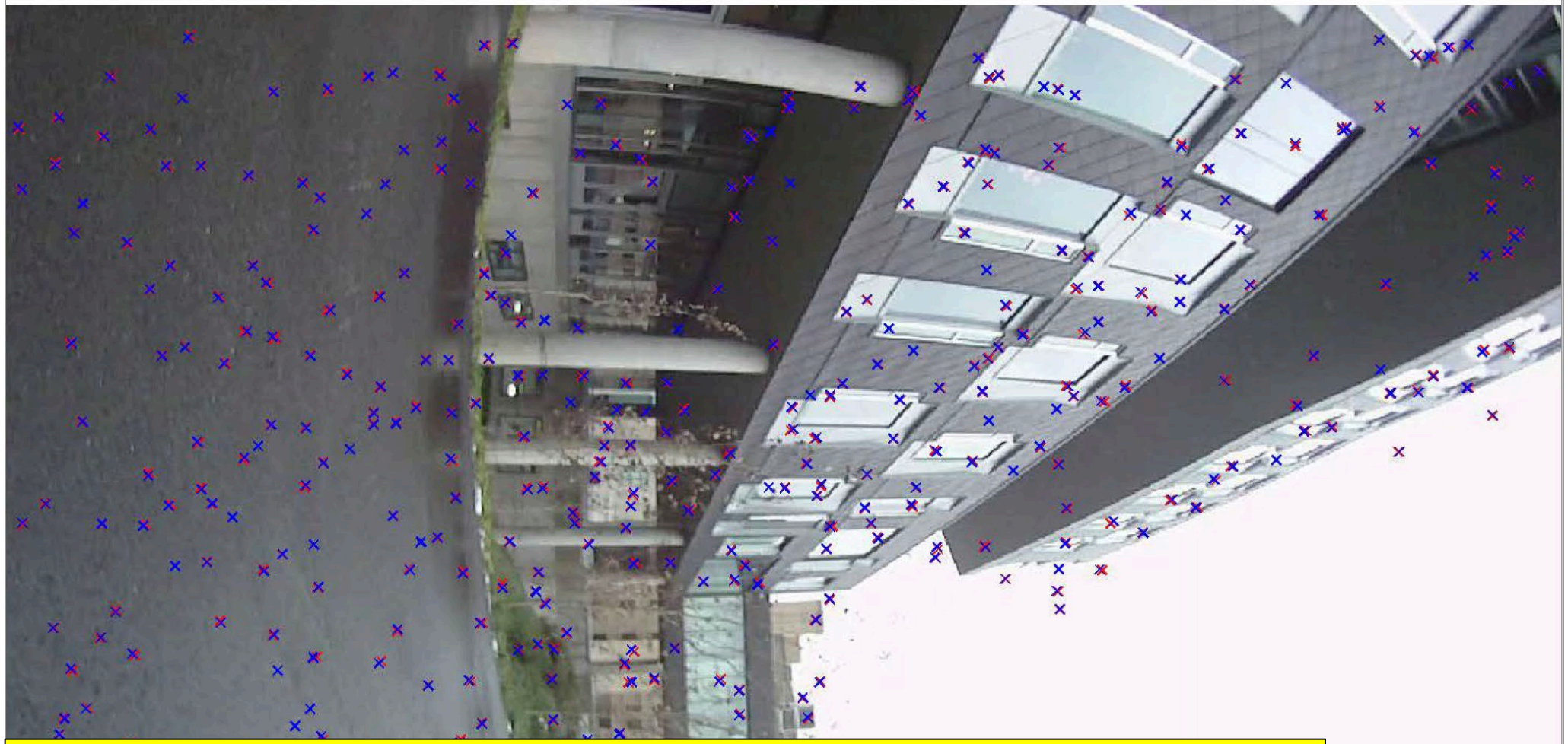
Before Bundle Adjustment



Hyun Soo Park

Geometric Refinement

After Bundle Adjustment



Near-zero reprojection errors
(perfect alignment between detection and reprojection)

3D model from F frames, N points

Reference frame ambiguity hence we fix the first frame to be the world frame:

$$R_1 = I \qquad \text{and} \qquad T_1 = 0$$

Even with fixing the first frame, a global scale factor is still present. If we multiply all 3D points and T with the same scale measurements do not change.

Hence we have $6(F-1)+3N-1$ independent unknowns

+6(F-1) camera poses
+3N 3D point
-1 scale unknown

and $2NF$ equations:

Homogeneous equations
$\boldsymbol{x}^f \sim [R|T]\boldsymbol{X}$

$$x_p^f = \frac{R_{11}^f X_p + R_{12}^f Y_p + R_{13}^f Z_p + T_x}{R_{31}^f X_p + R_{32}^f Y_p + R_{33}^f Z_p + T_z}$$

$$y_p^f = \frac{R_{21}^f X_p + R_{22}^f Y_p + R_{23}^f Z_p + T_y}{R_{31}^f X_p + R_{32}^f Y_p + R_{33}^f Z_p + T_z}$$

Best case, where every point is visible in every frame.

How many equations do we need?

If equations are independent (not always) then

$$2NF \geq 6F + 3N - 7$$

For two frames, it was already known that $N \geq 5$.

For three frames, $N \geq 4$.

N: point correspondences

F: frames

Example: how many unknowns and equations?

- E.g. F=1k images, and N=100k points in total
  - 6x1k + 3x100k -7 = ~306k unknowns!
- How many equations?
  - If all points are visible everywhere, then:
    - 2NF = 2*1k*100k = 200M equations!
    - If at least 306k of these equations are independent, we should be able to solve for the unknowns.

Note: These are all “back of the envelope” calculations that hide lots of assumptions. E.g. all points observed in all images etc. But still useful.

Expanding the reprojection error

$$\underset{\{X_n\}_{n=1}^{N},\{R_f,T_f\}_{f=1}^{F}}{\operatorname{argmin}} \quad d(\boldsymbol{x}_{nf}, \quad K_f[R_f|T_f]\boldsymbol{X_n})$$

The usual thing we do for overdetermined systems … least squares

$$\underset{u=\{\{X_n\}_{n=1}^{N},\{R_f,T_f\}_{f=1}^{F}\}}{\operatorname{argmin}} \quad ||\epsilon(u)||_2^2$$

Here, there is no consideration of uncertainty of each correspondences. We will introduce uncertainties later.

$$\epsilon^T = \left(\dots \quad x_p^f - \frac{R_{11}^f X_p + R_{12}^f Y_p + R_{13}^f Z_p + T_x}{R_{31}^f X_p + R_{32}^f Y_p + R_{33}^f Z_p + T_z} \quad y_p^f - \frac{R_{21}^f X_p + R_{22}^f Y_p + R_{23}^f Z_p + T_y}{R_{31}^f X_p + R_{32}^f Y_p + R_{33}^f Z_p + T_z} \quad \dots \right.$$

So, BA is a non-linear least squares problem. How to solve?

A Mini-Course on Optimization

First-order optimization methods: gradient descent

To minimize a differentiable function $f(x)$ over parameters x

1. Start from some initialization x
2. Compute the gradient $g = \nabla f(x)$, the direction of local descent
3. Descend along that direction by some step-length α i.e. $\delta x = -\alpha g$
4. Set $x \leftarrow x + \delta x$, then go back to step 2 and repeat

Q: How to set step-length α?

A: Often set to a small constant. Many other options, like Adaptive Gradients.

Gradient Descent Illustration

$\mathcal{L}(\boldsymbol{\theta}_0, \boldsymbol{\theta}_1)$

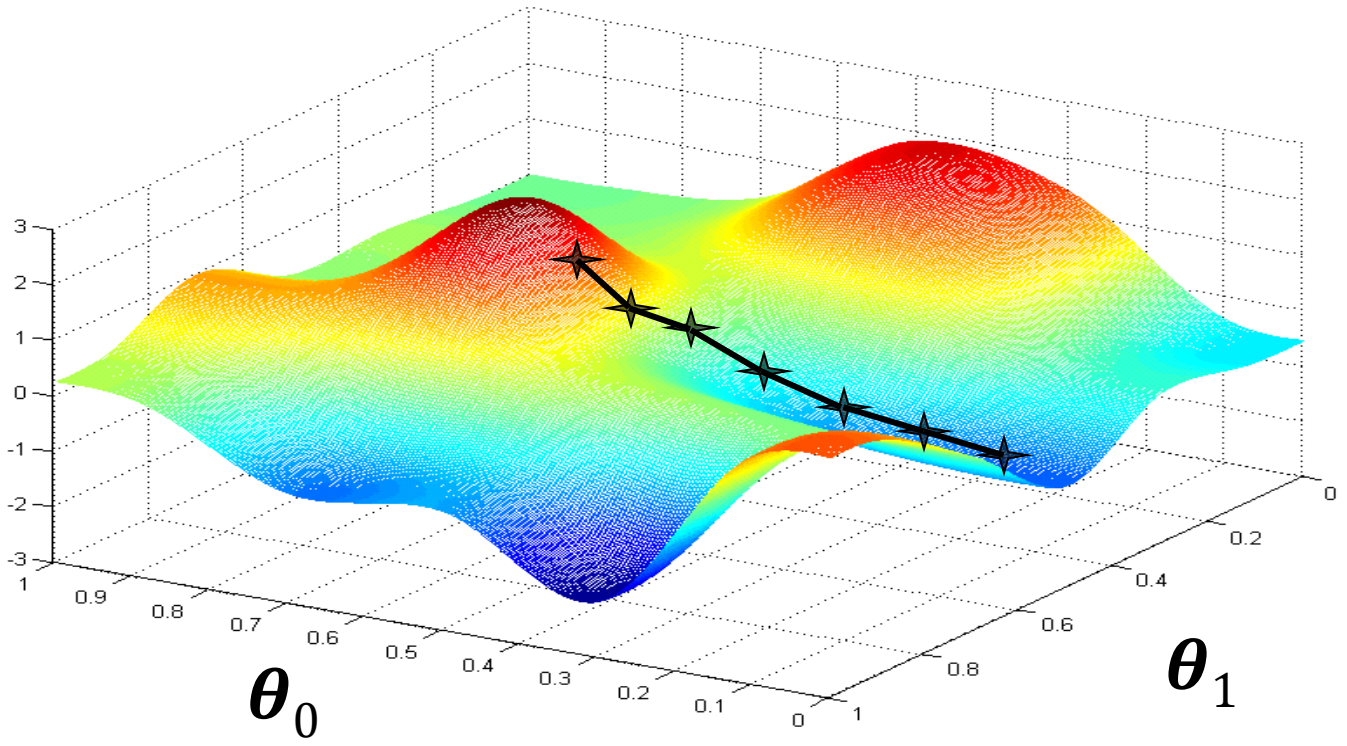
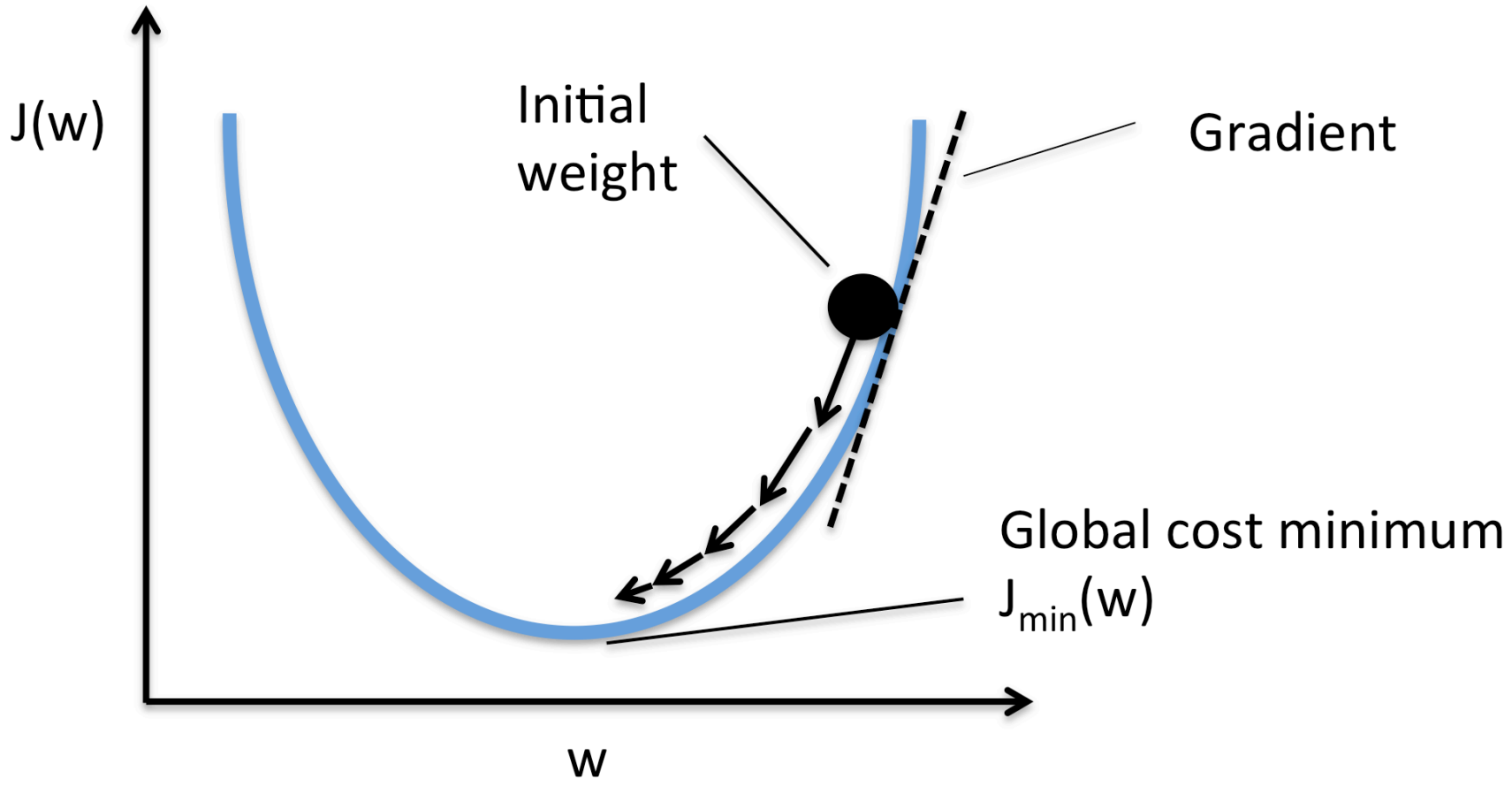


Figure by Andrew Ng

Gradient Descent Illustration



Slow convergence due to small gradients near the minimum.

https://sebastianraschka.com/faq/docs/gradient-optimization.html

Second-order optimization: a bird's eye view of the strategy

To minimize a (twice-) differentiable function $f(x)$ over parameters x

- Start with an initial estimate for x
- Locally approximate $f(x)$ using e.g. a second-order Taylor expansion.

$$f(x+\delta x) \approx f(x) + g^T \delta x + \frac{1}{2}\delta x^T H \delta x,$$

where: $g = \nabla f(x)$, and $H = \nabla^2 f(x)$

Hessian

- Try to find a displacement $x \to x + \delta x$ that locally minimizes the quadratic local approximation of $f(x)$
- This does not usually give the exact minimum of $f(x)$, but with luck it will improve over the initial estimate, and allow us to iterate to convergence.

Newton/Newton-Raphson's method (second-order)

- Locally approximate $f(x)$ at x_0 using e.g. a Taylor expansion.

$$f(x_0 + \delta x) \approx f(x_0) + g(x_0)^T \delta x + \frac{1}{2} \delta x^T H(x_0) \delta x,$$

where: $g(x_0) = \nabla f(x)|_{x=x_0}$ and $H(x_0) = \nabla^2 f(x)|_{x=x_0}$. Assume the "Hessian" H is positive semi-definite for now, so that you can find the minimum.

$$\nabla f(x_0 + \delta x) \approx H(x_0)\delta x + g(x_0) = 0 \Rightarrow$$

$$\delta x = -H(x_0)^{-1} g(x_0)$$

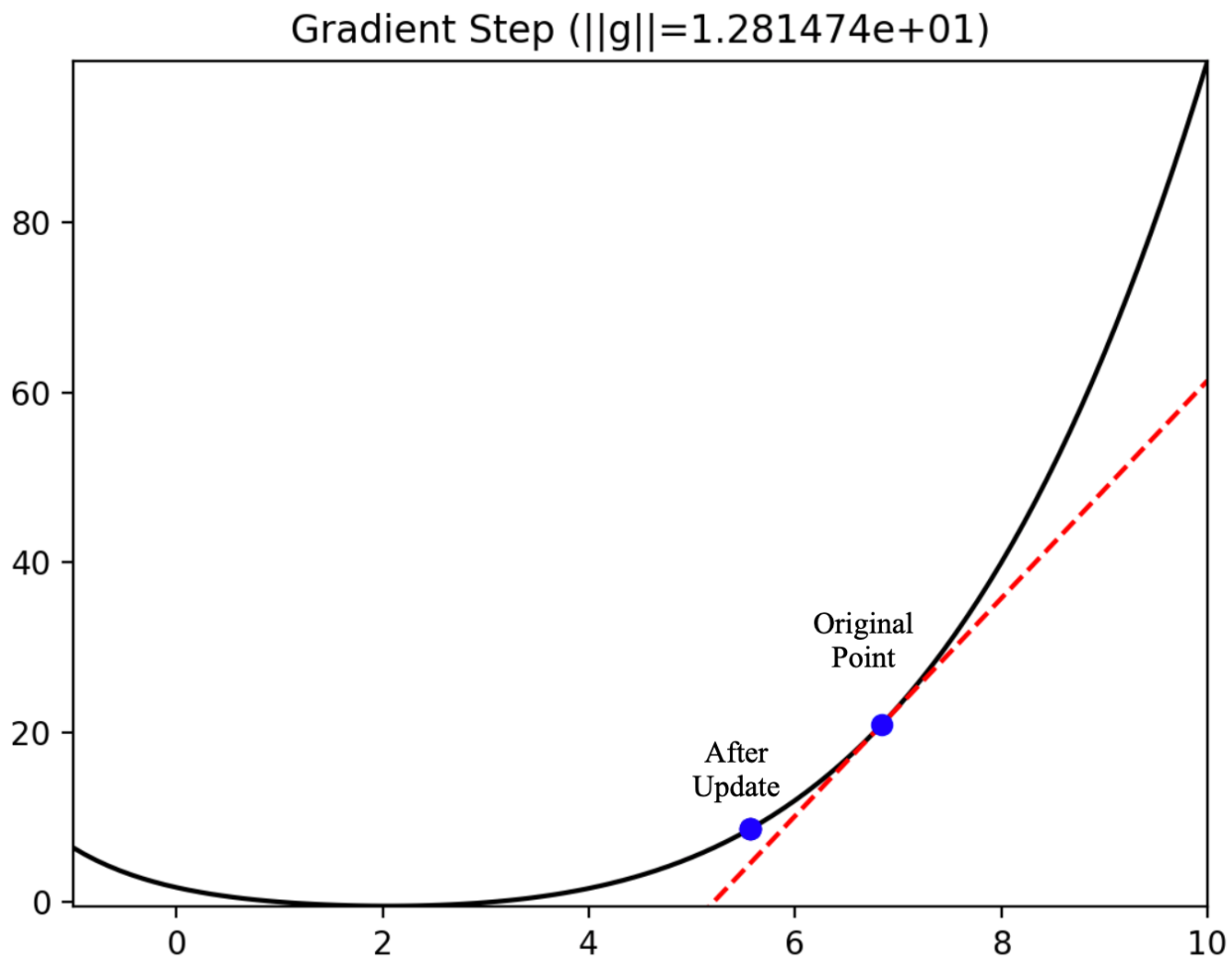
Goes directly to the minimum of the local quadratic approximation of f

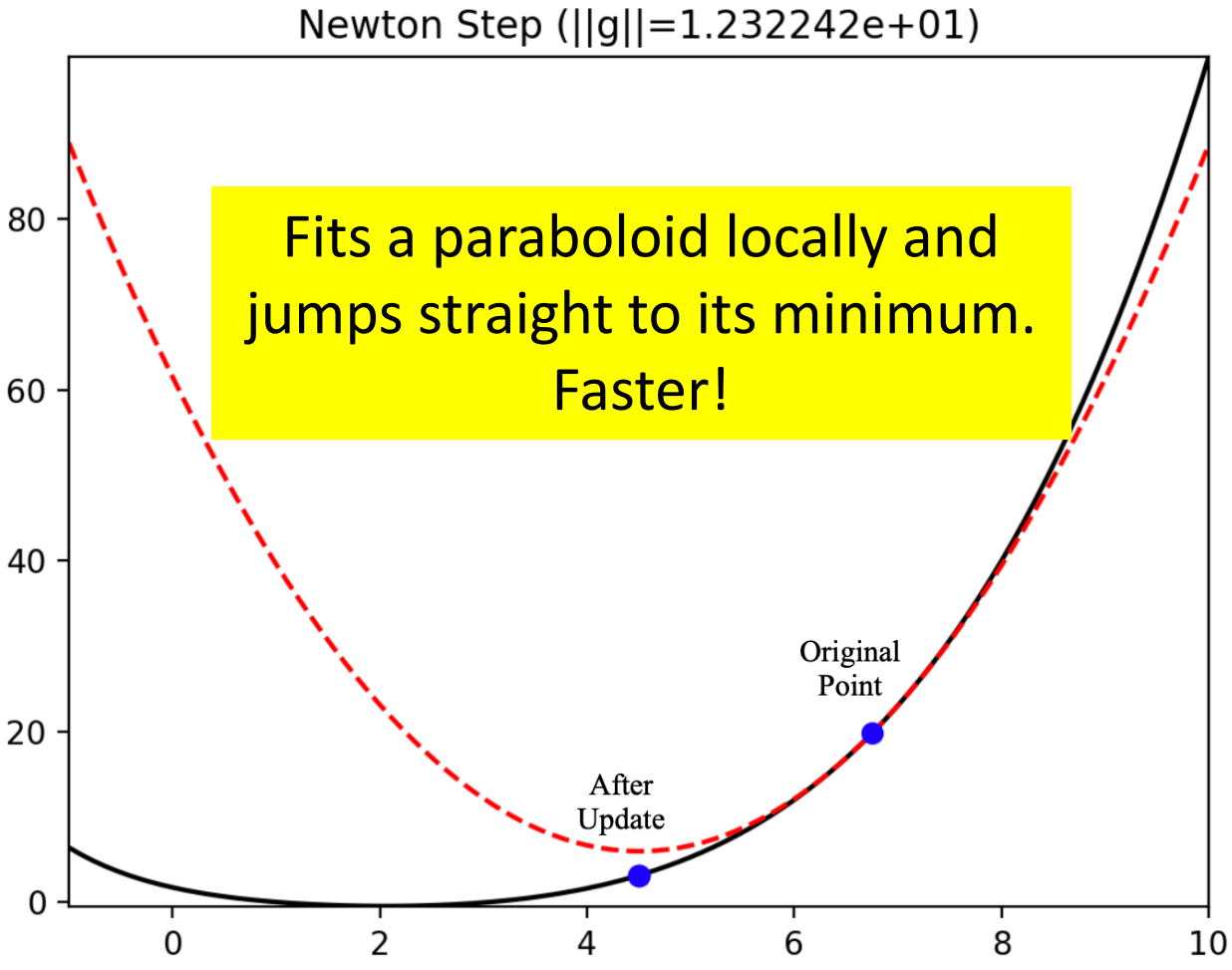
Iterating over this update = "Newton's method"

The most basic "second-order" non-linear optimization method

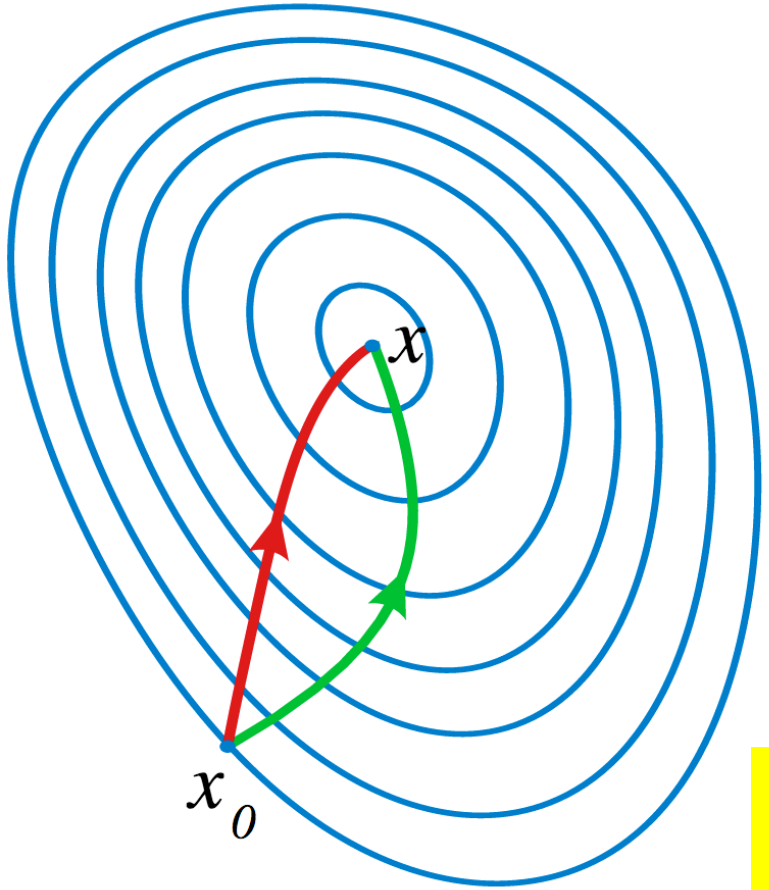
(Asymptotic quadratic convergence = error approx. squared at each iteration)

Gradient Descent Vs. Newton's Method





Gradient Descent Vs. Newton's Method



Red – Newton's Method
Green - gradient descent iteration.

When it works, Newton's method often finds faster, more direct routes!

Newton's method uses curvature information (i.e. the second derivative) to take a more direct route.

Problem: Computing the Hessian $H = \nabla^2 f$ is expensive!

Wikipedia

Gauss-Newton Method for Least Squares part 1/2

An approximate version of Newton's method for least squares:

$$\underset{u=\left\{\{X_n\}_{n=1}^{N},\{R_f,T_f\}_{f=1}^{F}\right\}}{\operatorname{argmin}} \quad f(u) = ||\epsilon(u)||_2^2$$

Gradient $g = \nabla_u f = 2\sum_i \epsilon_i(u)\nabla_u \epsilon_i(u) = 2\,J(u)^T \boldsymbol{\epsilon}(u),$ A

Where $J_{ij} = \frac{\partial \epsilon_i}{\partial u_j}$ Jacobian

Hessian reads (computing the gradient of g):

$$H(u) = 2\sum_i \nabla_u \epsilon_i(u)\nabla_u \epsilon_i(u) + \epsilon_i(u)\frac{\partial^2 \epsilon_i}{\partial^2 u^2}$$

$$= 2J(u)^T J(u) + 2\sum_i \epsilon_i(u)\frac{\partial^2 \epsilon_i}{\partial u^2}$$

Ignoring the hard-to-compute quadratic terms, $H(u) \approx 2\,J(u)^T J(u)$ B

Saves computation, and is approx. true when error ϵ_i is small, or the function is ~ linear.

Gauss-Newton Method for Least Squares part 2/2

We saw earlier, Newton's update for general non-linear optimization:

$$\delta u = -H^{-1} g$$

For least squares problems $\underset{u}{\operatorname{argmin}} f(u) = ||\epsilon(u)||_2^2$, we have seen:

- By A on the last slide, $g = 2\, J(u)^T \epsilon(u)$
- By B on the last slide, $H(u) \approx 2 J(u)^T J(u)$

Directly leads to Gauss-Newton update, often called "Normal Equation":

$$\delta u = -\left(J(u)^T J(u)\right)^{-1} J(u)^T \epsilon(u)$$

Q: Have you seen an expression like this before when solving linear equations?

Hint: If you were minimizing $\left|\left|Ax - b\right|\right|_2^2$, what would the solution be?

Gauss-Newton for the BA Least Squares Problem?

Recall that the BA problem looked like:

$$\underset{u=\left\{\{X_n\}_{n=1}^{N},\{R_f,T_f\}_{f=1}^{F}\right\}}{\operatorname{argmin}} \quad f(u) = ||\epsilon(u)||_2^2$$

So the new update at each iteration would look like:

$$\delta u = -(J^T J)^{-1} J^T \epsilon$$

(J and ϵ are both functions of the parameters u)

Computational Nightmares!

$$\underset{u=\{\{X_n\}_{n=1}^{N},\{R_f,T_f\}_{f=1}^{F}\}}{\operatorname{argmin}} \|\epsilon(u)\|_2^2$$

$$\epsilon^T = \left(\dots \quad x_p^f - \frac{R_{11}^f X_p + R_{12}^f Y_p + R_{13}^f Z_p + T_x}{R_{31}^f X_p + R_{32}^f Y_p + R_{33}^f Z_p + T_z} \quad y_p^f - \frac{R_{21}^f X_p +}{R_{31}^f X_p +} \right.$$

We've decided we want to do something like:

$$\delta x = -(J^T J)^{-1} J^T \epsilon$$

What is the size of $J = \left[\frac{\partial \epsilon_i}{\partial u_j}\right]_{ij}$?

- E.g. F=1k images, and N=100k points in total
  - 6x1k + 3x100k -7 = ~306k unknowns!
- How many equations?
  - If all points are visible everywhere, then:
    - 2NF = 2*1k*100k = 200M equations!

Recall:

- The dimension of the reprojection error vector $\boldsymbol{\epsilon}$ is 2NF (F frames, N points in each frame, 2 dimensions per point in the reprojection error vector)
- The number of unknown parameters $\boldsymbol{u}$ is $\mathrm{M} = 6F + 3N - 7$

The size of J is $2NF \times M$. (e.g. 200e6 x 306e3)

$J^T J$ is $M \times M$ e.g. (306e3 x 306e3).

For general matrices, inverse scales as $O(M^3) \approx 2.8e16$.

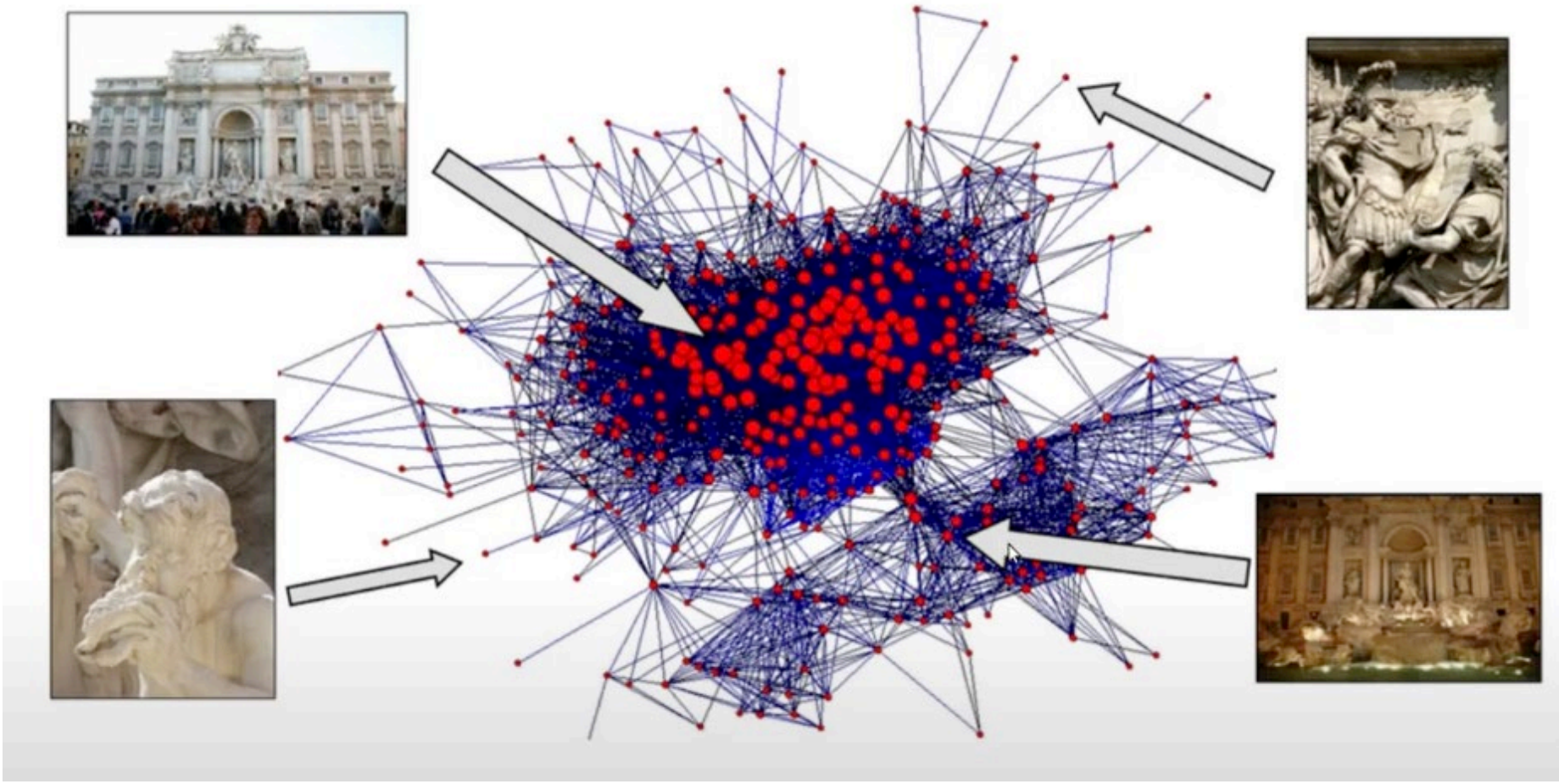
This is all bad news!

BA is all about linear algebra implementation tricks!

- The modern bundle adjustment literature is all about how to deal with this massive computational complexity cleverly.

- Some useful properties to simplify huge linear equation systems:
  - Try to avoid inverses of *large general matrices* to the extent possible, and instead reduce to "simpler" matrix inverses.
    - Block diagonal matrices can be inverted block-by-block.
  - Try to set up equations $A\boldsymbol{x} = \boldsymbol{b}$ with triangular matrices A, much easier to solve. ("forward/backward substitutions")
  - Avoid matrix multiplications of large general matrices:
    - Sparse matrices, including non-diagonal ones, are much easier to multiply

Sparsity in Cameras – 3D points connectivity



Schonberger and Frahm, the COLMAP paper, 2016

Separating the unknowns to see structure

Instead of the original monolithic Gauss-Newton update rule:

$$\delta \boldsymbol{u} = -(J^T J)^{-1} J^T \boldsymbol{\epsilon}$$

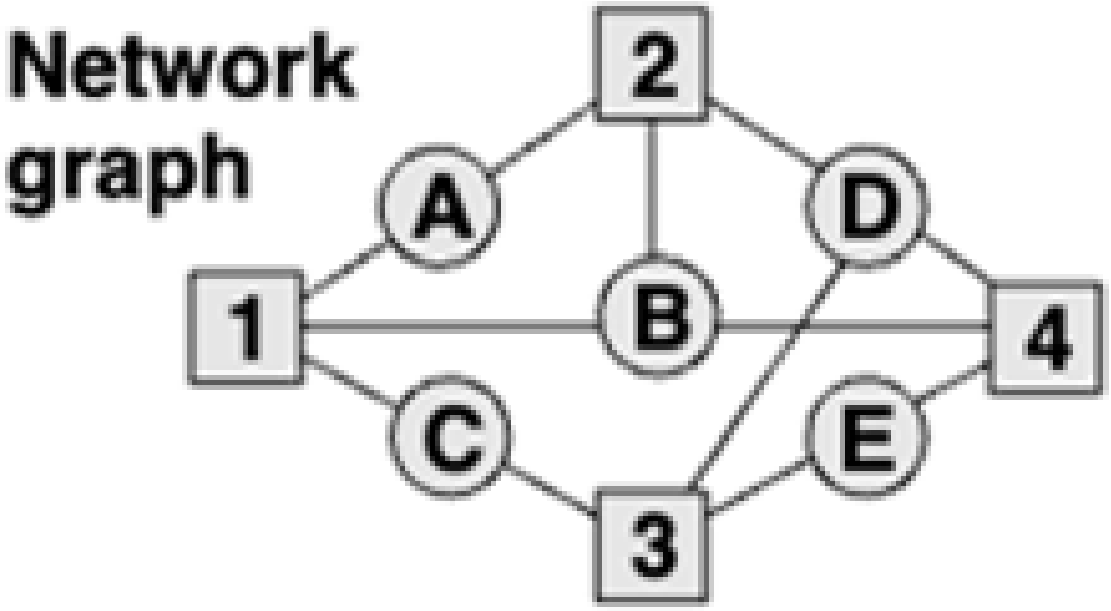
Let us separate out the parameters x into:

- structure parameters a (3P-1 in number), and
- camera parameters b (6F-6 in number)

$$\boldsymbol{u} = \begin{bmatrix} \boldsymbol{a} \\ \boldsymbol{b} \end{bmatrix} \begin{matrix} \rightarrow \text{Structure parameters} \\ \rightarrow \text{Camera parameters} \end{matrix}$$

Note: Often many more points $\boldsymbol{a}$ than cameras $\boldsymbol{b}$

Camera -> 3D point connectivity graph



Points A, B, C, D, E

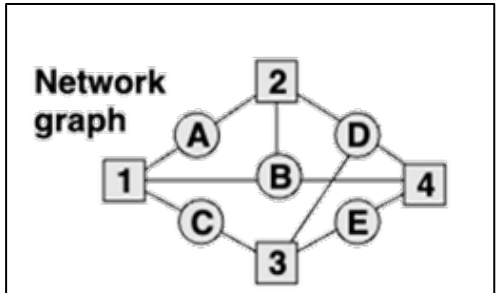
Cameras 1, 2, 3, 4

Note that each point is visible only in a small number of cameras.

How does this affect the Jacobian J with ij-th element $J_{ij} = \frac{\partial \epsilon_i}{\partial u_j}$?

The Structure of the Jacobian J

$$u = \begin{bmatrix} a \\ b \end{bmatrix} \begin{matrix} \rightarrow \text{Structure parameters} \\ \rightarrow \text{Camera parameters} \end{matrix}$$

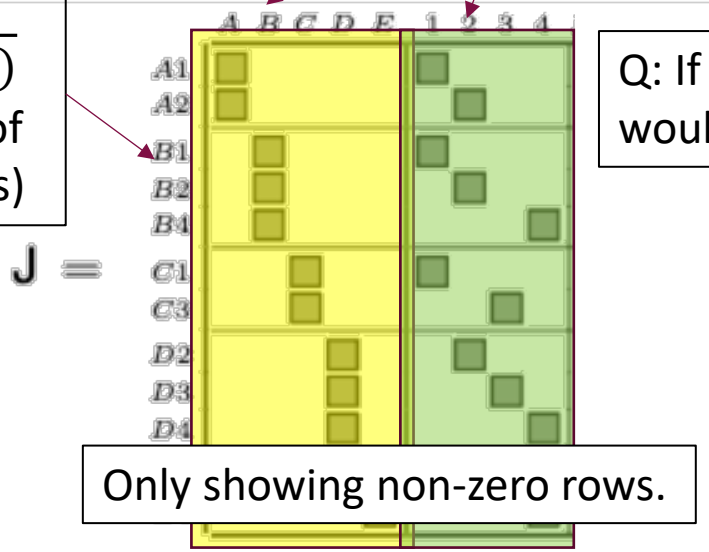


J_a: Jacobian w.r.t 3D point positions (3 cols per point)

J_b: Jacobian w.r.t camera poses (6 cols per camera)

$$J_{ij} = \frac{\partial \epsilon_i \text{ (errors)}}{\partial u_j \text{ (parameters)}}$$

Reprojection xy errors of pt B in camera 1 (2 rows)



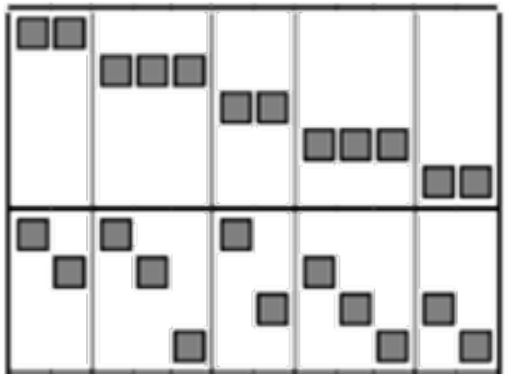
Q: If every point were visible in every camera, how would this Jacobian change?

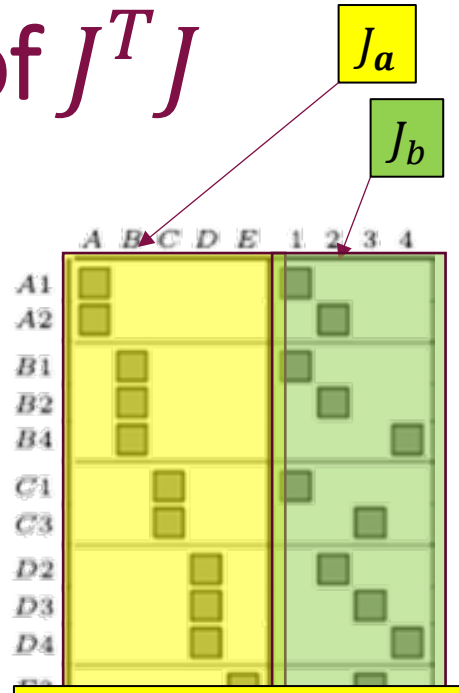
Only showing non-zero rows.

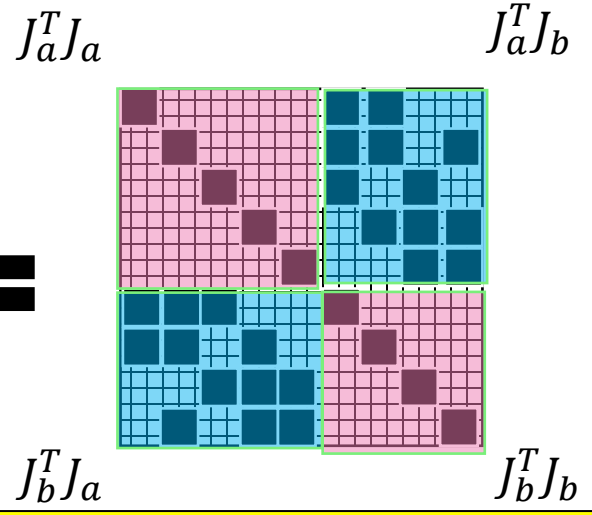
The Jacobian is sparse, and has distinct structure in its two parts J_a and J_b.

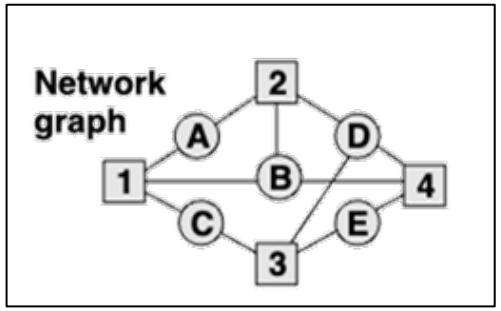
Perhaps this will be useful in computing $(J^T J)^{-1} J^T \epsilon$!

The Structure of $J^T J$









Q: If every point were visible in every camera, what would $J^T J$ look like?

$J_a^T J_a$ and $J_b^T J_b$ are block-diagonal, making them easy to invert.
$J_a^T J_b$ is also sparse, which is convenient in matrix operations.

So we rewrite $\delta \boldsymbol{u} = -(J^T J)^{-1} J^T \boldsymbol{\epsilon}$ as:

$$\begin{bmatrix} \delta \boldsymbol{a} \\ \delta \boldsymbol{b} \end{bmatrix} = - \begin{bmatrix} J_a^T J_a & J_a^T J_b \\ J_b^T J_a & J_b^T J_b \end{bmatrix}^{-1} J^T \boldsymbol{\epsilon},$$

or

$$\begin{bmatrix} U & W \\ W^T & V \end{bmatrix} \begin{bmatrix} \delta \boldsymbol{a} \\ \delta \boldsymbol{b} \end{bmatrix} = -J^T \boldsymbol{\epsilon}$$

Sidenote: Exploiting structure to make BA easier

$$\begin{bmatrix} U & W \\ W^T & V \end{bmatrix} \begin{bmatrix} \delta \boldsymbol{a} \\ \delta \boldsymbol{b} \end{bmatrix} = [-J^T \boldsymbol{\epsilon}]_{(|\boldsymbol{a}|+|\boldsymbol{b}|)\times 1}$$

Computing the matrix product on the right:

$$\begin{bmatrix} U & W \\ W^T & V \end{bmatrix} \begin{bmatrix} \delta \boldsymbol{a} \\ \delta \boldsymbol{b} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\epsilon}'_{\boldsymbol{a}} \\ \boldsymbol{\epsilon}'_{\boldsymbol{b}} \end{bmatrix}$$

Premultiplying with a carefully chosen matrix on both sides:

$$\begin{bmatrix} I & -WV^{-1} \\ 0 & I \end{bmatrix} \begin{bmatrix} U & W \\ W^T & V \end{bmatrix} \begin{bmatrix} \delta \boldsymbol{a} \\ \delta \boldsymbol{b} \end{bmatrix} = \begin{bmatrix} I & -WV^{-1} \\ 0 & I \end{bmatrix} \begin{bmatrix} \boldsymbol{\epsilon}'_{\boldsymbol{a}} \\ \boldsymbol{\epsilon}'_{\boldsymbol{b}} \end{bmatrix}$$

$$\boldsymbol{u} = \begin{bmatrix} \boldsymbol{a} \\ \boldsymbol{b} \end{bmatrix} \begin{matrix} \rightarrow \text{Structure parameters} \\ \rightarrow \text{Camera parameters} \end{matrix}$$

Sidenote: Exploiting structure to make BA easier

$\boldsymbol{u} = \begin{bmatrix} \boldsymbol{a} \\ \boldsymbol{b} \end{bmatrix}$ → Structure parameters
→ Camera parameters

$$\begin{bmatrix} I & -WV^{-1} \\ 0 & I \end{bmatrix} \begin{bmatrix} U & W \\ W^T & V \end{bmatrix} \begin{bmatrix} \delta \boldsymbol{a} \\ \delta \boldsymbol{b} \end{bmatrix} = \begin{bmatrix} I & -WV^{-1} \\ 0 & I \end{bmatrix} \begin{bmatrix} \boldsymbol{\epsilon}'_{\boldsymbol{a}} \\ \boldsymbol{\epsilon}'_{\boldsymbol{b}} \end{bmatrix}$$

$$\begin{bmatrix} U - WV^{-1}W^T & 0 \\ W^T & V \end{bmatrix} \begin{bmatrix} \delta \boldsymbol{a} \\ \delta \boldsymbol{b} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\epsilon}'_{\boldsymbol{a}} - WV^{-1}\boldsymbol{\epsilon}'_{\boldsymbol{b}} \\ \boldsymbol{\epsilon}'_{\boldsymbol{b}} \end{bmatrix}$$

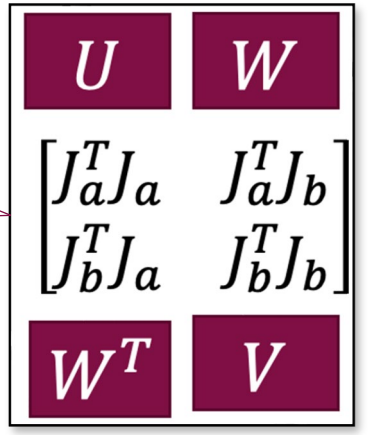
Now, life becomes much easier. First solve for camera parameter updates $\boldsymbol{\delta a}$:

$$(U - WV^{-1}W^T)\delta \boldsymbol{a} = \boldsymbol{\epsilon}'_{\boldsymbol{a}} - WV^{-1}\boldsymbol{\epsilon}'_{\boldsymbol{b}}$$

Next, plug this in to solve for structure updates $\boldsymbol{\delta b}$:

$$V\delta \boldsymbol{b} = \boldsymbol{\epsilon}'_{\boldsymbol{b}} - W^T \delta a$$

Plug in



Complexity reduced!

$$\left(J_a^T J_a - J_a^T J_b \left(J_b^T J_b\right)^{-1}_{|b|\times|b|} J_b^T J_a\right)_{|a|\times|a|} \delta \boldsymbol{a} = \boldsymbol{\epsilon}'_{\boldsymbol{a}} - \left(J_a^T J_b\right)\left(J_b^T J_b\right)^{-1}_{|b|\times|b|} \boldsymbol{\epsilon}'_{\boldsymbol{b}}$$

"Schur complement" form, positive definite,
makes the equation easy to solve using "Cholesky decomposition" LL^T into lower triangular matrix and its transpose

$$\left(J_b^T J_b\right)_{|b|\times|b|} \delta \boldsymbol{b} = \boldsymbol{\epsilon}'_{\boldsymbol{b}} - J_b^T J_a \delta a$$

Recall that $J_b^T J_b$ is block-diagonal, making it easy to invert block-by-block!

$$\boldsymbol{u} = \begin{bmatrix} \boldsymbol{a} \\ \boldsymbol{b} \end{bmatrix} \begin{matrix} \rightarrow \text{Structure parameters} \\ \rightarrow \text{Camera parameters} \end{matrix}$$

Note

- The following grayed out slides were not covered in detail and not tested, but may be of interest to you.

Afternotes: Accounting for uncertainties in BA?

$$\underset{u=\{\{X_n\}_{n=1}^{N},\{R_f,T_f\}_{f=1}^{F}\}}{\operatorname{argmin}} \quad ||\epsilon(u)||_2^2$$

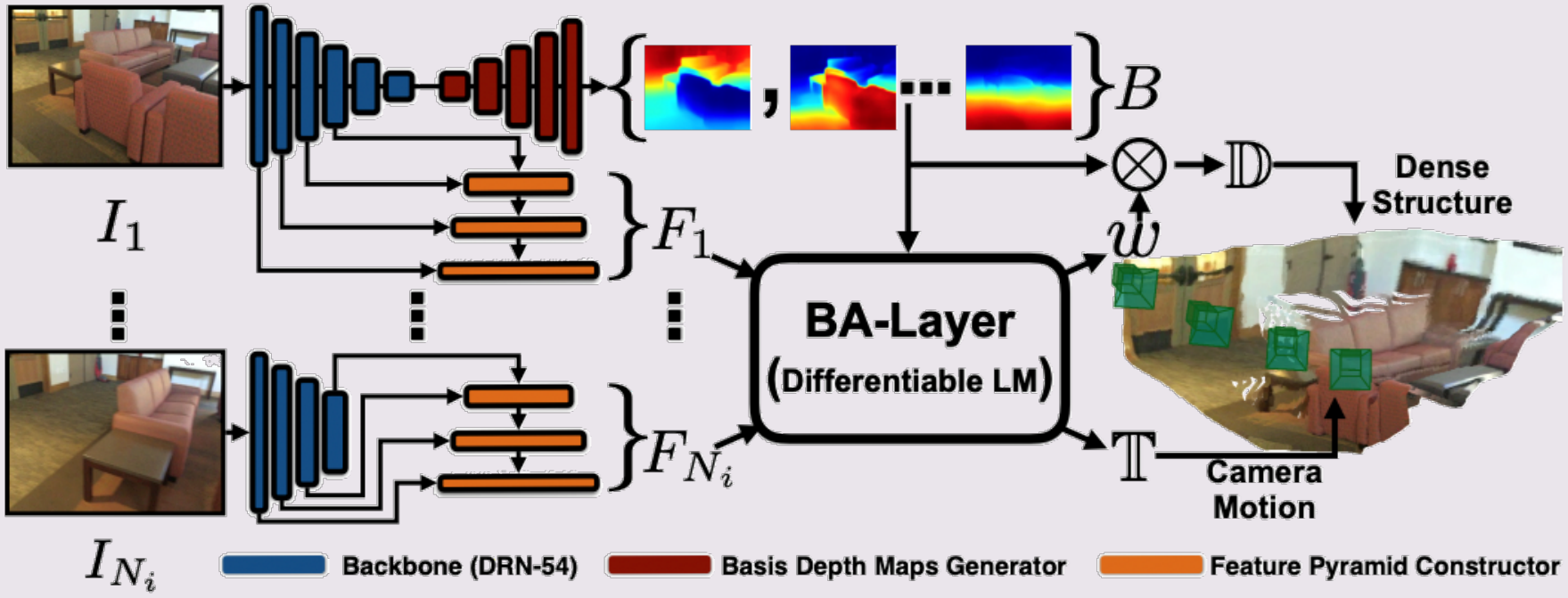
$$\underset{u=\{\{X_n\}_{n=1}^{N},\{R_f,T_f\}_{f=1}^{F}\}}{\operatorname{argmin}} \quad \epsilon^T W \epsilon$$

Prioritizes some errors over others, and accounts for possible correlations between errors etc.

Basically everything we've discussed applies to this case too, with minor adjustments.

Afternotes: "Can't we just deep-learn Bundle Adjustment?"

- After all, we constantly solve non-linear least squares problems in deep learning with neural networks. Could we just deep learn BA?
- Answer: Researchers are trying!



Tang et al, BA-Net, 2019

Afternotes: Speed-up using advances in hardware

- F=125 frames, N=2000 points.
- “Normal computation time” ~ 1.5 seconds on CPU.

Abstract

Graph processors such as Graphcore's Intelligence Processing Unit (IPU) are part of the major new wave of novel computer architecture for AI, and have a general design with massively parallel computation, distributed on-chip memory and very high inter-core communication bandwidth which allows breakthrough performance for message passing algorithms on arbitrary graphs.

We show for the first time that the classical computer vision problem of bundle adjustment (BA) can be solved extremely fast on a graph processor using Gaussian Belief Propagation. Our simple but fully parallel implementation uses the 1216 cores on a single IPU chip to, for instance, solve a real BA problem with 125 keyframes and 1919 points in under 40ms, compared to 1450ms for the Ceres CPU library. Further code optimisation will surely increase this difference on static problems, but we argue that the real promise of graph processing is for flexible in-place optimisation of general, dynamically changing factor graphs representing Spatial AI problems. We give indications of this with experiments showing the ability of GBP to efficiently solve incremental SLAM problems, and deal with robust cost functions and different types of factors.

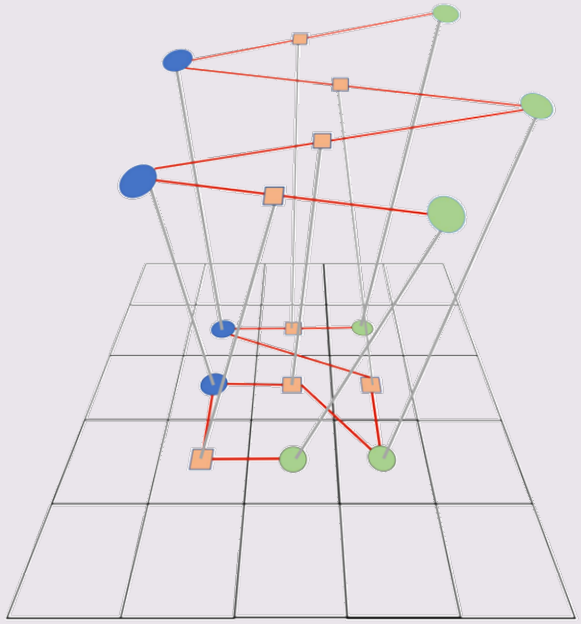


Figure 1: We map a bundle adjustment factor graph onto the tiles (cores) of Graphcore's IPU and show that Gaussian Belief Propagation can be used for rapid, distributed, in-place inference for large problems. Here we display the most simple mapping in which each node in the factor graph is mapped onto a single arbitrary tile. Keyframe nodes are blue, landmark nodes are green and measurement factor nodes are orange.

Ortiz et al, Bundle Adjustment on a Graph Processor, CVPR 2020

The original structure from motion paper from 1979!

The interpretation of structure from motion

By S. Ullman

Artificial Intelligence Laboratory, Massachusetts Institute of Technology,
545 Technology Square (Room 808), Cambridge, Massachusetts 02139 U.S.A.

(*Communicated by S. Brenner, F.R.S. – Received 20 April* 1978)

The interpretation of structure from motion is examined from a computional point of view. The question addressed is how the three dimensional structure and motion of objects can be inferred from the two dimensional transformations of their projected images when no three dimensional information is conveyed by the individual projections.

Ullman, 1979

Some widely used solvers

- Ceres: – http://ceres-solver.org/
- GTSAM: https://gtsam.org/

Onwards from BA: Can Further Improve Structure

- Output point cloud at the end of everything we have learned is a “sparse” point cloud --- in the 3D map, it only locates the *feature points* for which we had appearance-based correspondences. What about other points?
  - Freezing camera parameter estimates, we can further improve structure using “multi-view stereo”, which can produce **dense 3D reconstructions.**

Stereopsis For Dense Reconstruction!

material from Szeliski's Computer Vision

and Frisbee's book "Seeing" and slides by Kris Kitani, and Noah Snavely

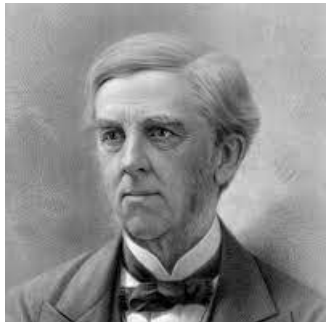
Stereogram

Two images from slightly moved cameras fed separately into the two eyes

"The first effect of looking at a good photograph through the stereoscope is a surprise such as no painting ever produced. The mind feels its way into the very depths of the picture. The scraggy branches of a tree in the foreground run out at us as if they would scratch our eyes out."

Charles Wheatstone
1838

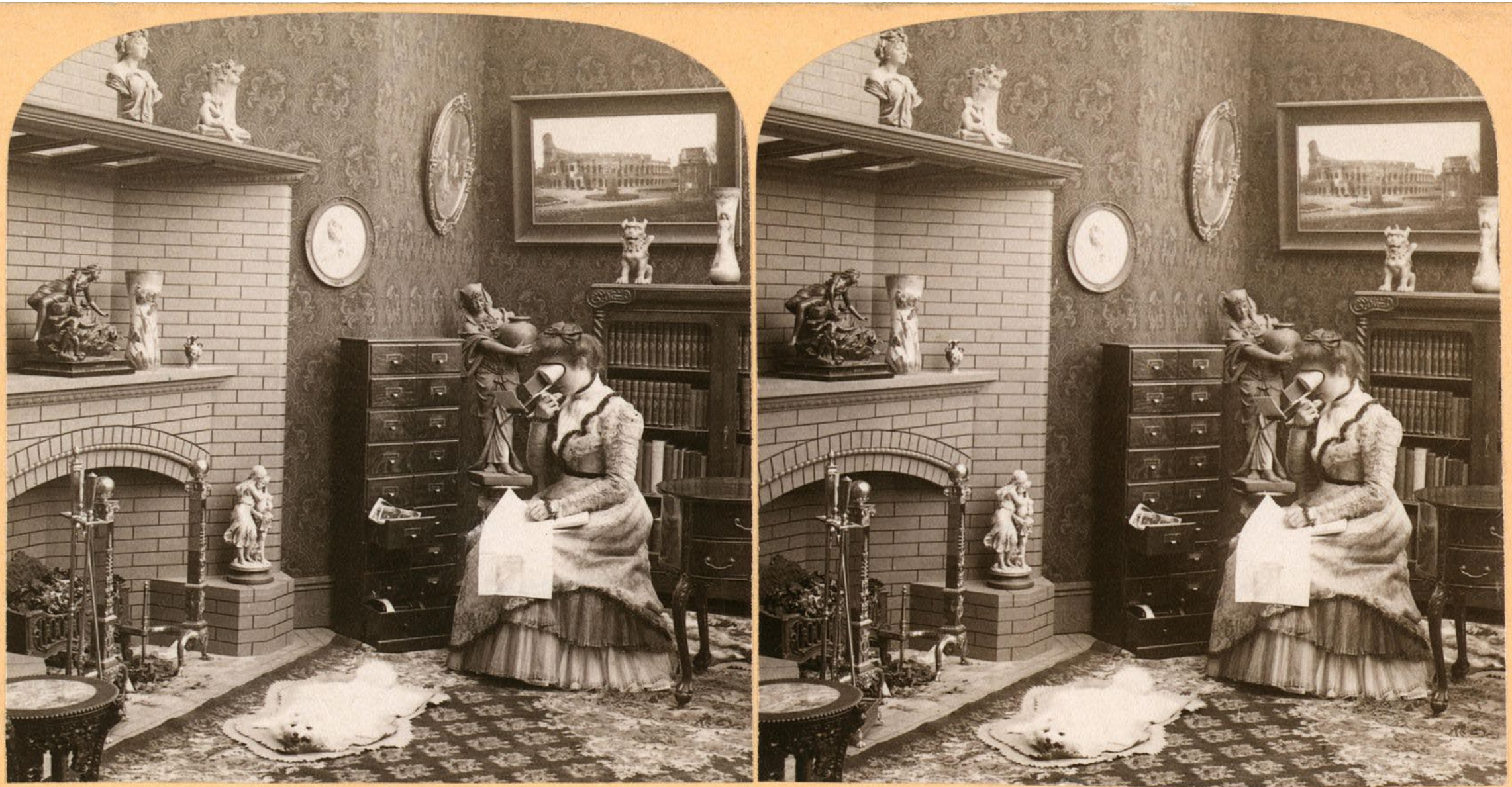
David Brewster
1849



Oliver Wendell Holmes
1859

https://scalar.chapman.edu/scalar/this-land-is-your-land/stereoscopes

Painting Stereographs: “Solid Writing”



Underwood & Underwood, Publishers.
New York, London, Toronto-Canada, Ottawa-Kansas.

Works and Studios
Arlington, N.J. Littleton, N.H. Washington, D.C.

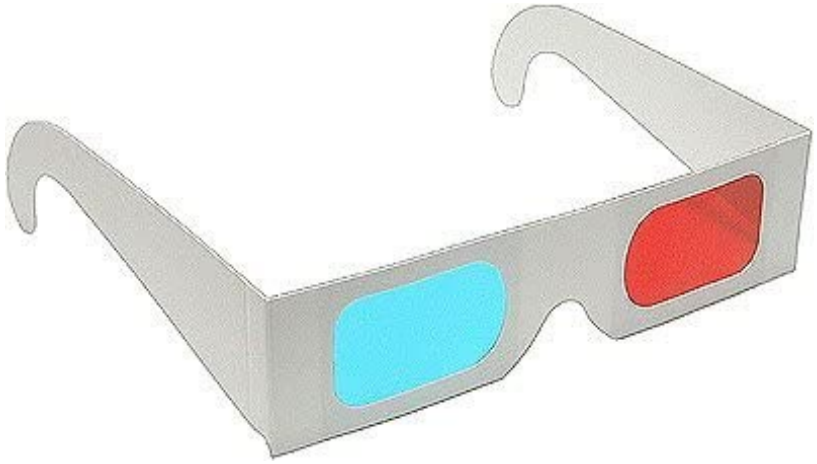
The Stereograph as an Educator—Underwood Patent Extension Cabinet in a home Library.

(2)

Anaglyphs

Wikipedia

Anaglyphs and 3D movies



The left eye image is filtered to remove blue & green. The right eye image is filtered to remove red.

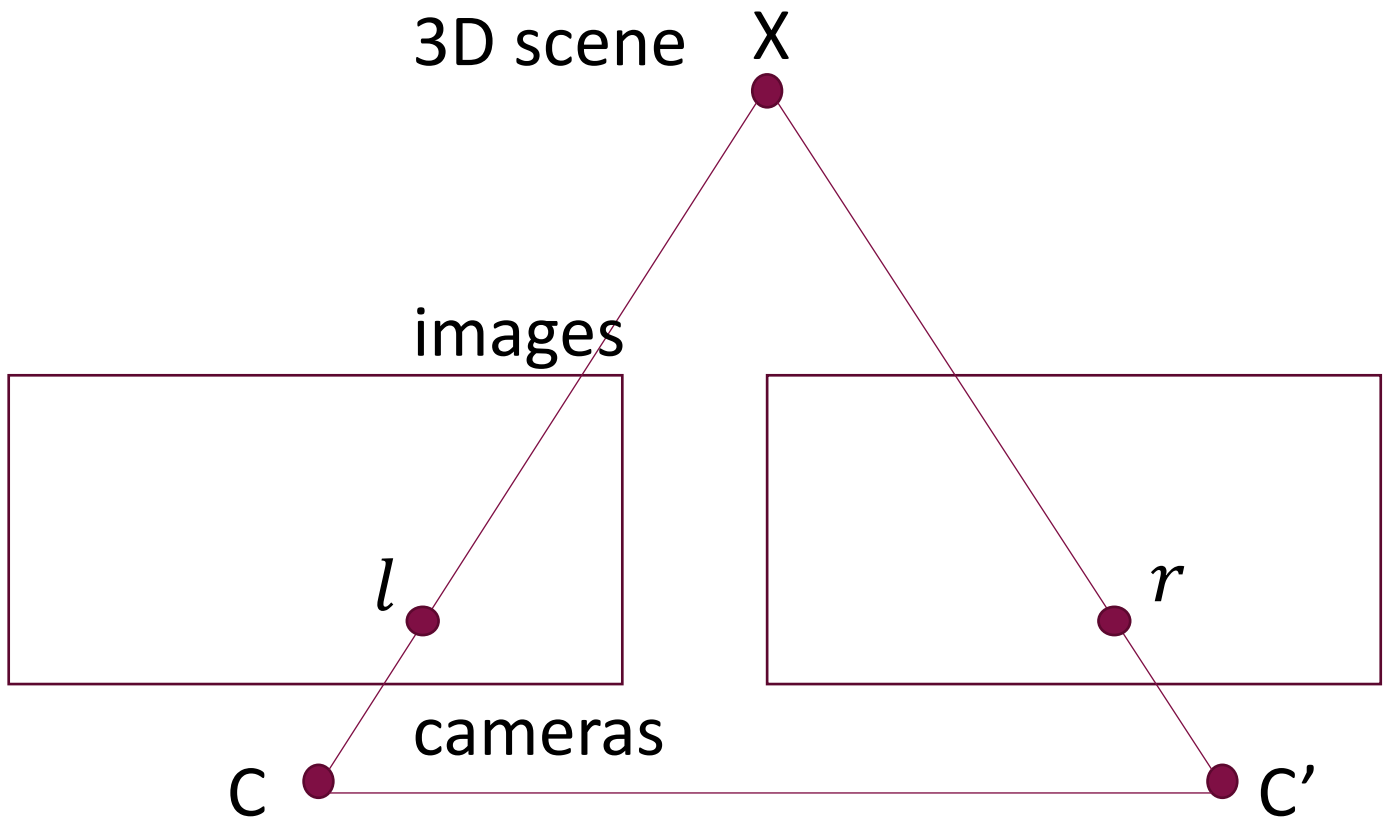
Dense Reconstruction from *Known* Cameras (e.g. after SfM)

- Desired: Find full 3D of the scene, not just a sparse set of points in it.
- Given:
  - Images of the same scene from many cameras.
  - K, R, T all known for all cameras.

- "Binocular" stereo = 2-cameras.
  - Plain "stereo" usually means binocular.

- "Multi-view" stereo = many cameras

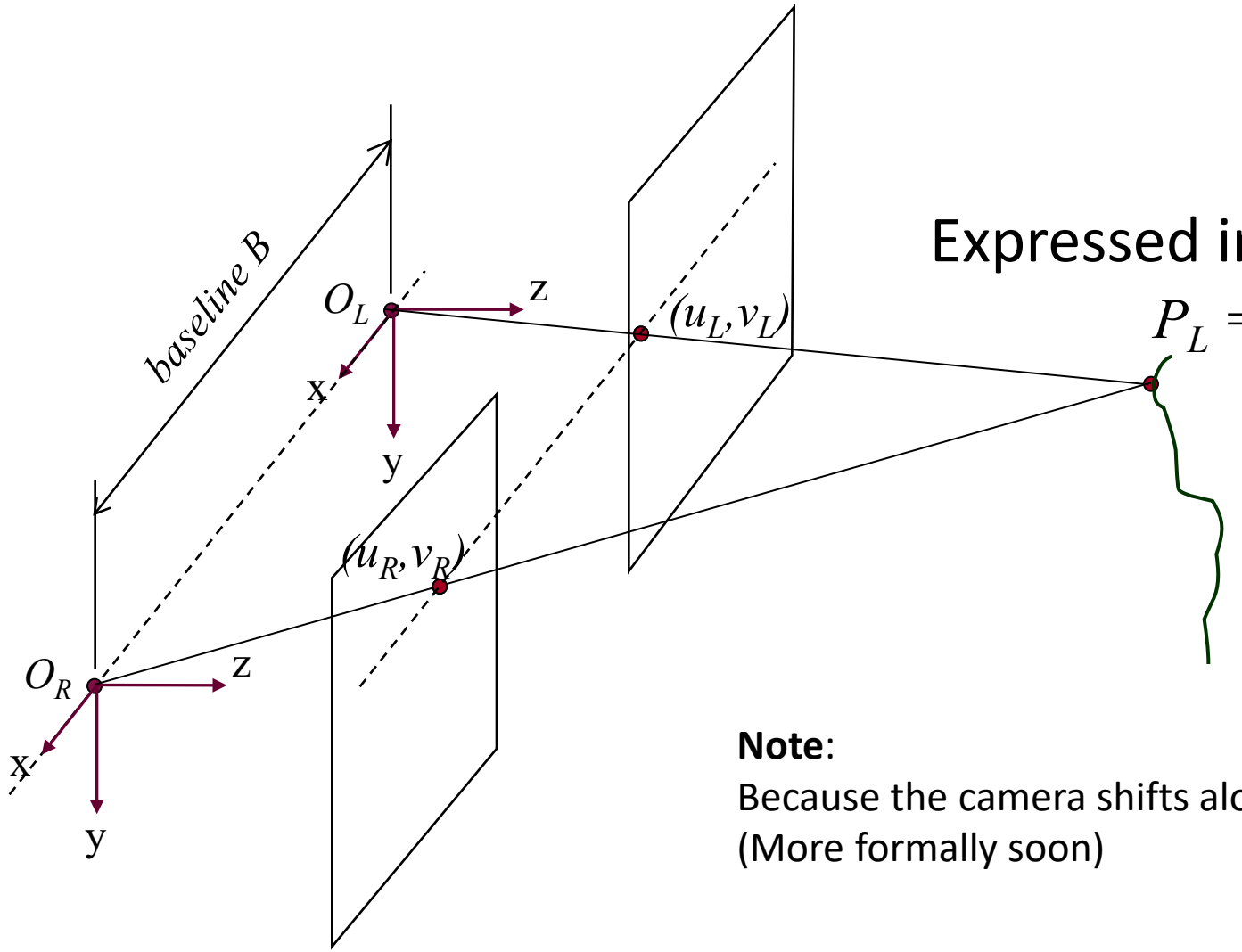
V0: Frontoparallel cameras + correspondences known



How to get 3D information from $l = (u_l, v_l)$ and $r = (u_r, v_r)$?

Basic Parallel Stereo Derivations

cameras images 3D scene



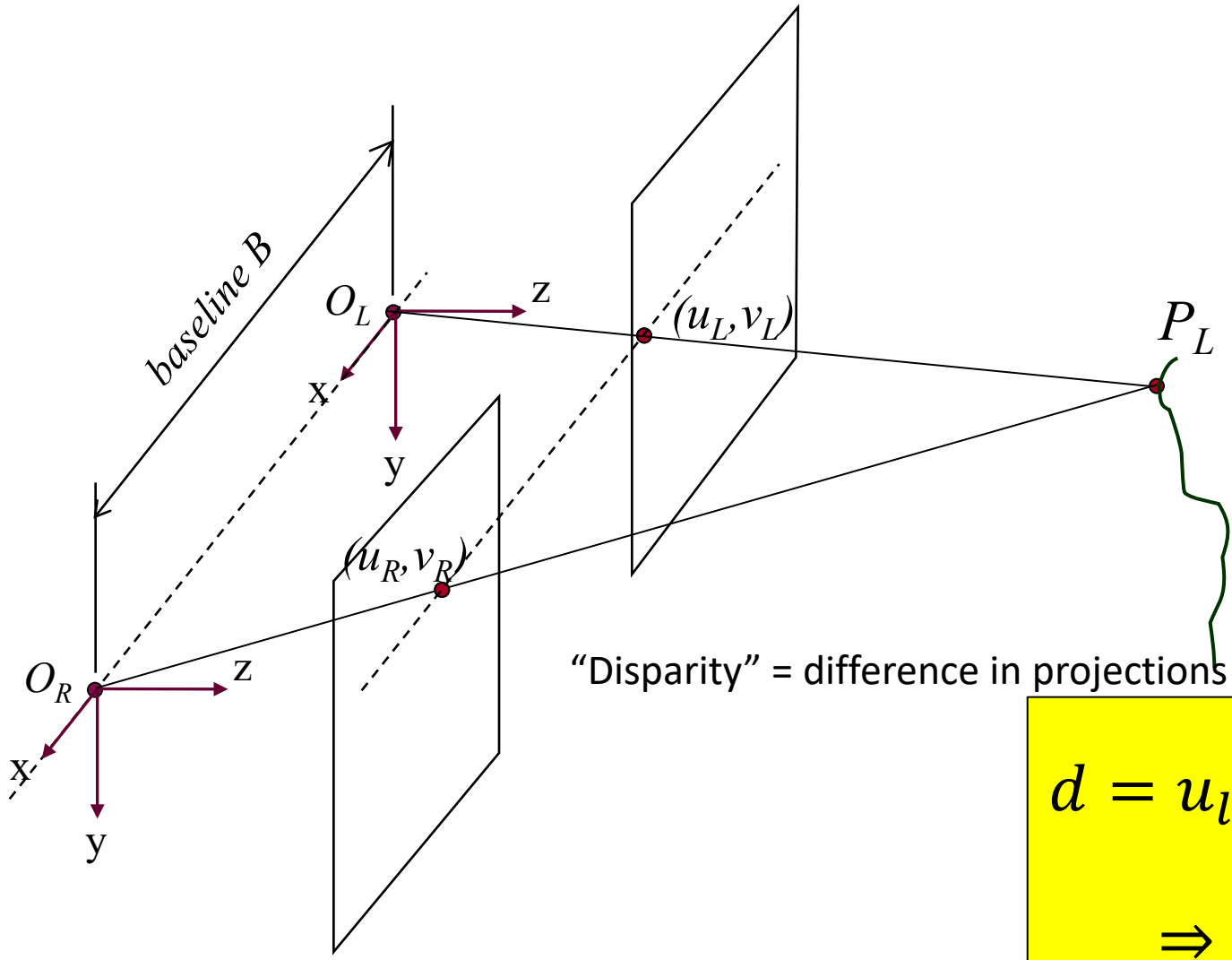
Expressed in left camera frame

$P_L = (X, Y, Z)$

Note:
Because the camera shifts along x, $v_L = v_R$
(More formally soon)

Basic Parallel Stereo Derivations

cameras images 3D scene



$P_L = (X, Y, Z)$

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

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

"Disparity" = difference in projections between the two cameras

$$d = u_l - u_r = f\frac{B}{Z}$$

$$\Rightarrow Z = f\frac{B}{d}$$

Examples

- Let baseline $B = 10\text{cm}$
- Let focal length f be 1000 px
  - Nominal for an HD image (1920 x 1080) image with wide 90 degree FOV.
- If object is 1m away,
  - Then disparity = $d = \frac{fB}{Z} = 1000\ \text{px}\frac{0.1}{1} = 100\ \text{px}$
- If object is 100m away,
  - Then disparity = $d = \frac{fB}{Z} = 1000\ \text{px}\frac{0.1}{100} = 1\ \text{px}$
  - If baseline had been larger?
    - E.g. with $B = 10\text{m}$, $d = \frac{fB}{Z} = 1000\ \text{px}\frac{10}{100} = 100\ \text{px}$

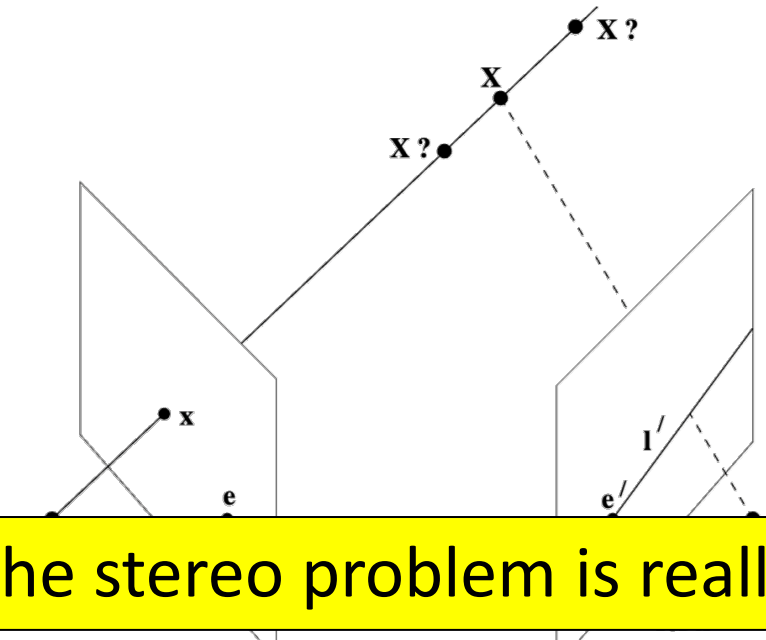
Larger baseline stereo systems can resolve larger depths!

Point motions between cameras can be large!

Correspondences For Stereo?

- We have seen: finding 3D is easy given correspondences and K, R, T
- We know to find correspondences using optical flow.
- But pixel movements here can be large! No longer sufficient to match locally alone! Plus, we would like *dense, per-pixel correspondences!*

Fortunately, correspondences are a bit easier once camera orientations are known, as in stereo! (Hint: epipolar lines constrain point correspondences!)



Note: We have seen many times before, correspondences help geometry.
Now: geometry helps find correspondences!

The stereo problem is really all about finding correspondences!

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

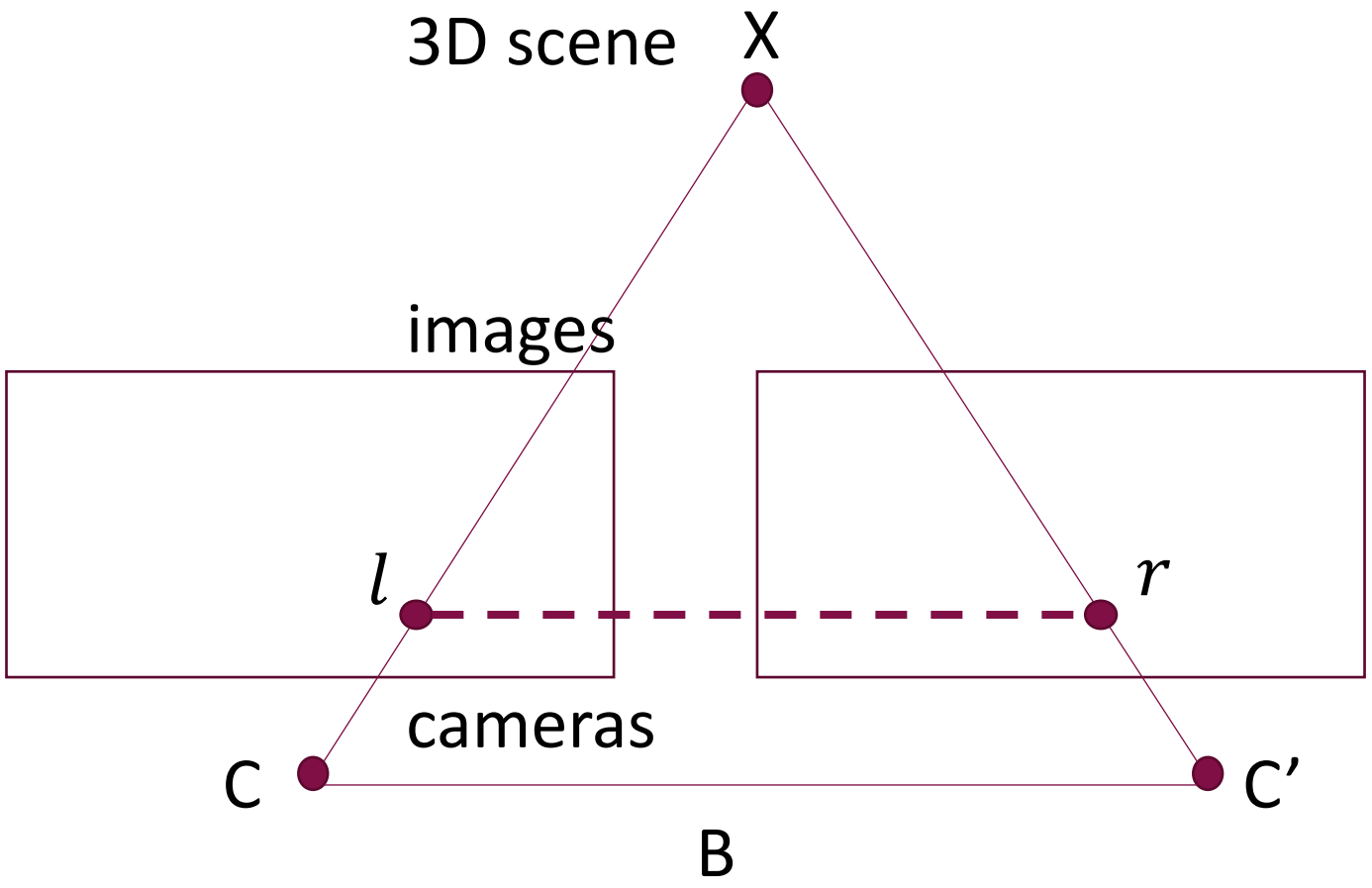
- Then see how to “rectify” non-frontoparallel cameras to be frontoparallel.

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

- Finally (likely next class), improvement through dynamic programming.

Searching for dense correspondences in the frontoparallel stereo setting

Correspondences for frontoparallel cameras



We have just derived that

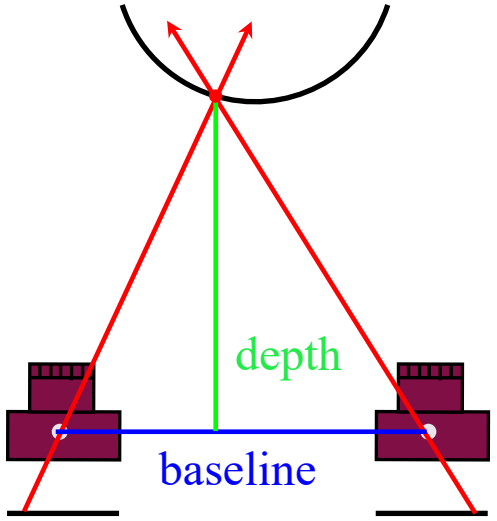
$(u_l, v_l) = \left(f\frac{X}{Z}, f\frac{Y}{Z}\right)$ and

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

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?)

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

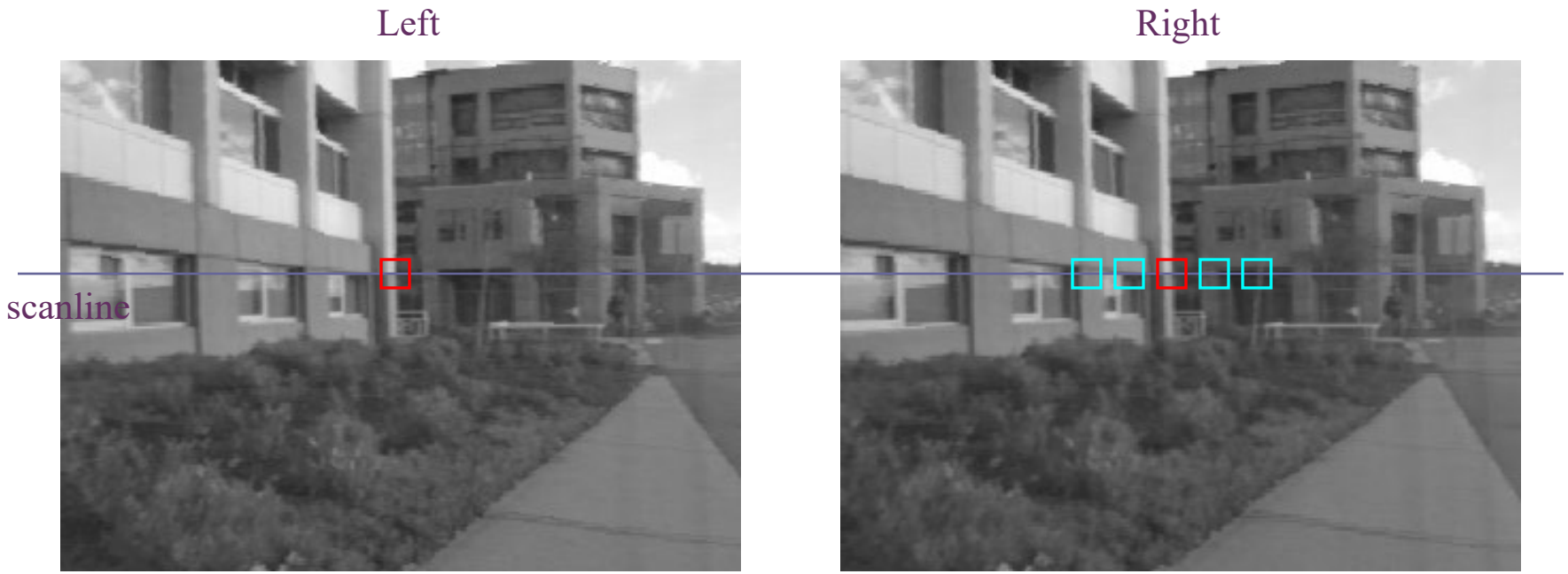
- Assume some minimum "depth": Z_{min}
  - Set $d_{\max} = \frac{1}{Z_{\min}}$, e.g. $d_{\max} = 100$
- Quantize the interval [-d_{max}, d_{max}]!
  - e.g. [-100, 100]-> candidate disparities [-100,-95, …, 0, 5, 10, …, 100]
- Now, must select from these candidate disparities for each pixel.

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

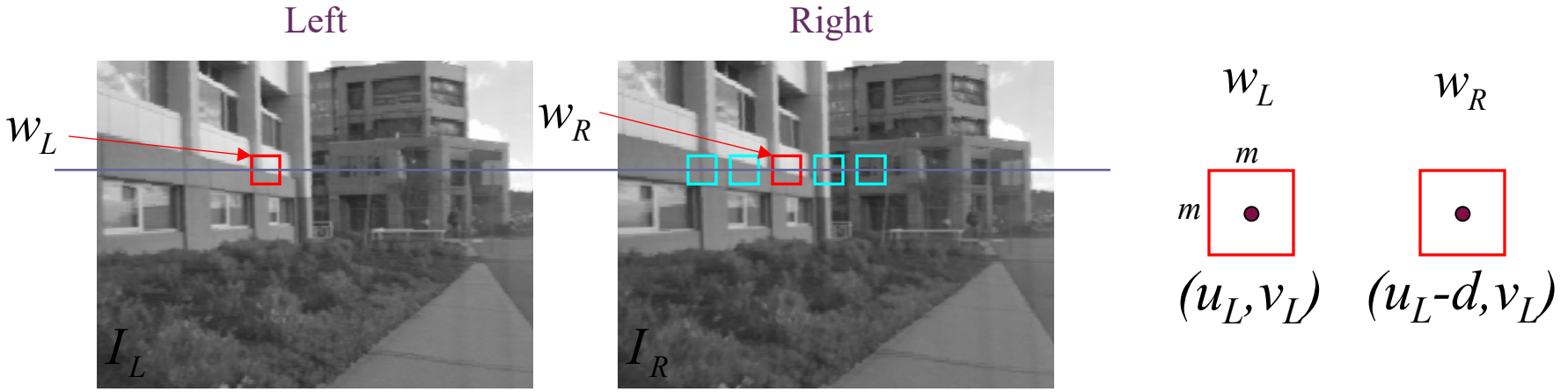
We will focus on the bolded choices

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 :

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

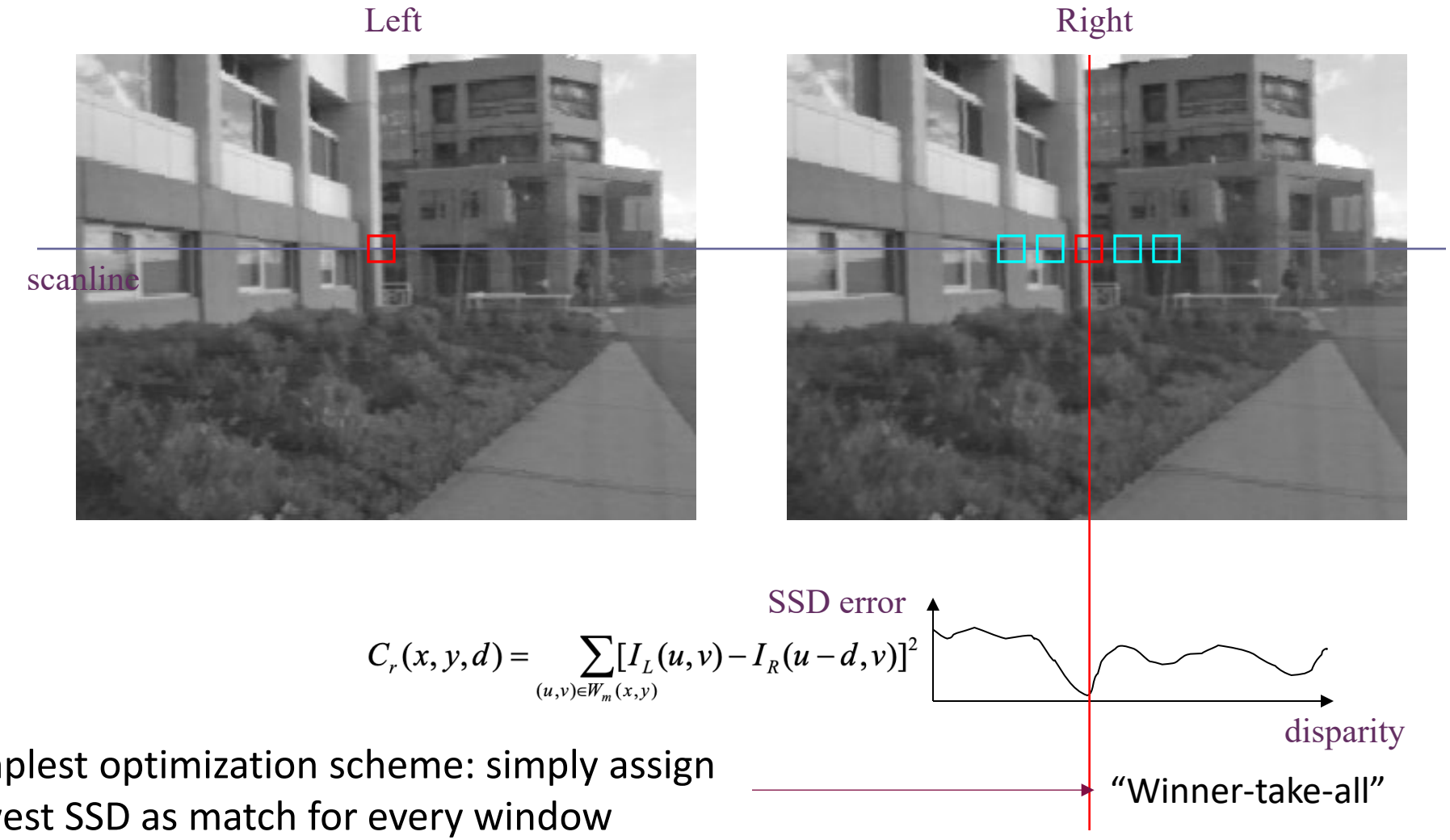
The SSD cost measures the intensity difference as a function of disparity :

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

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

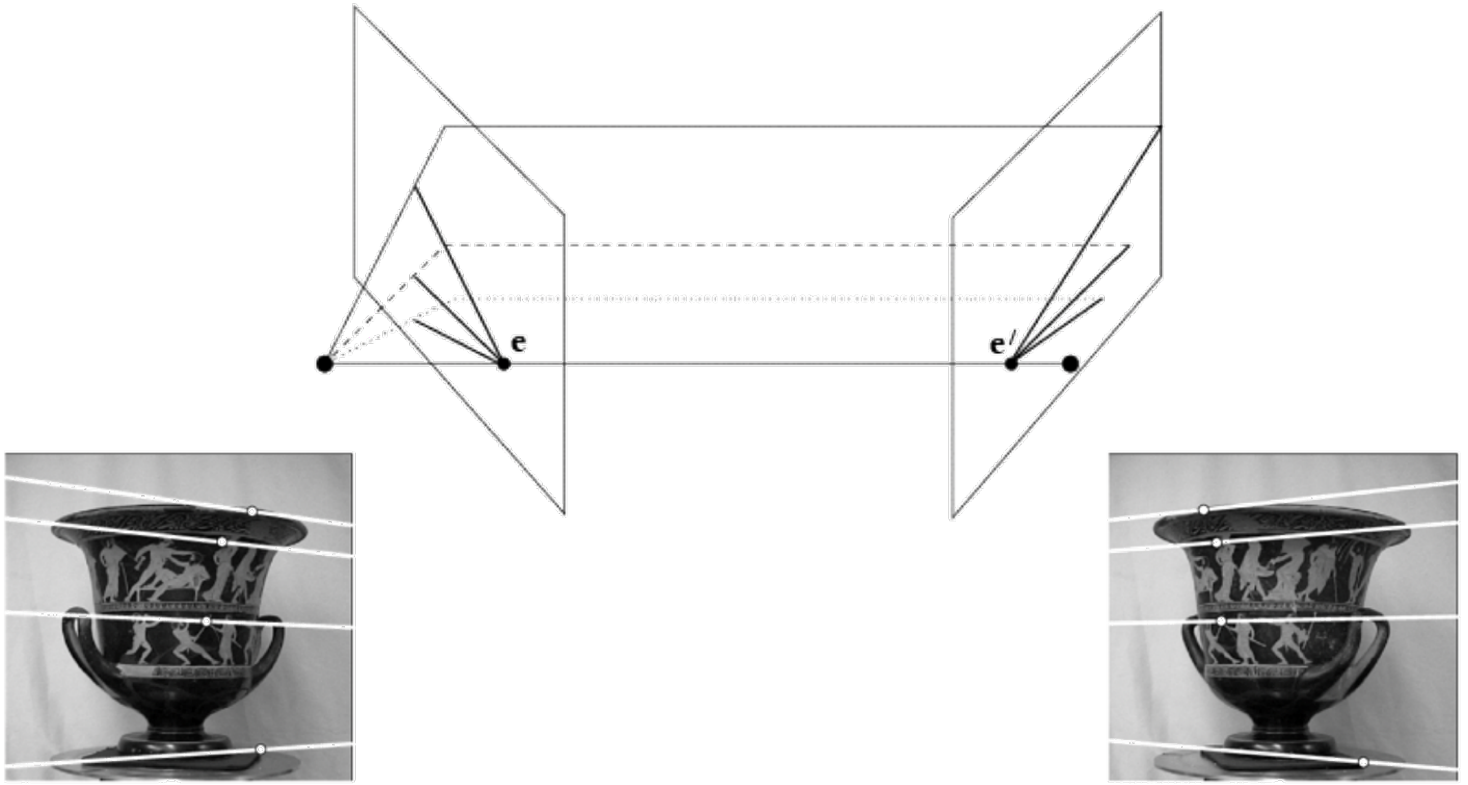
Matching Windows with SSD + Winner-Take-All

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



What if the cameras are not frontoparallel?

Stereo Rectification



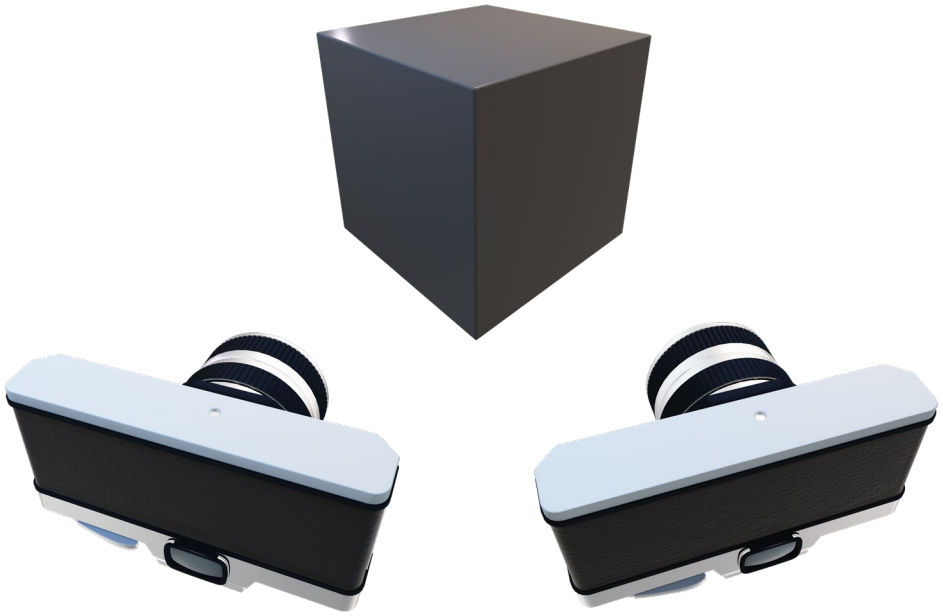
Q: What if the cameras weren't frontoparallel to start with?

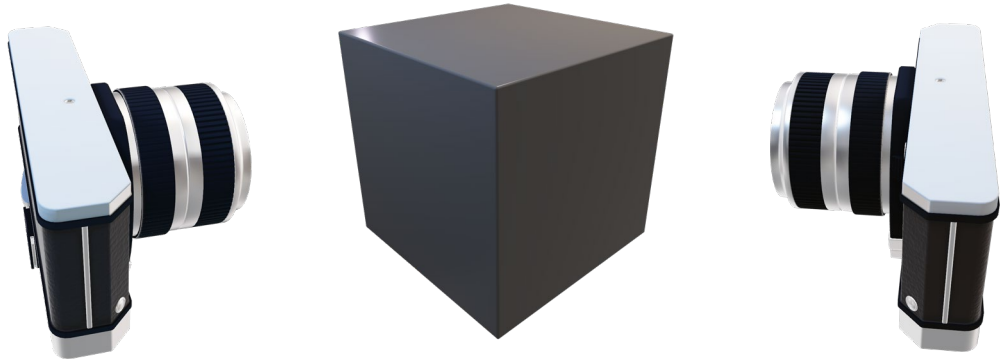
A: "Rectification". *Make* them!

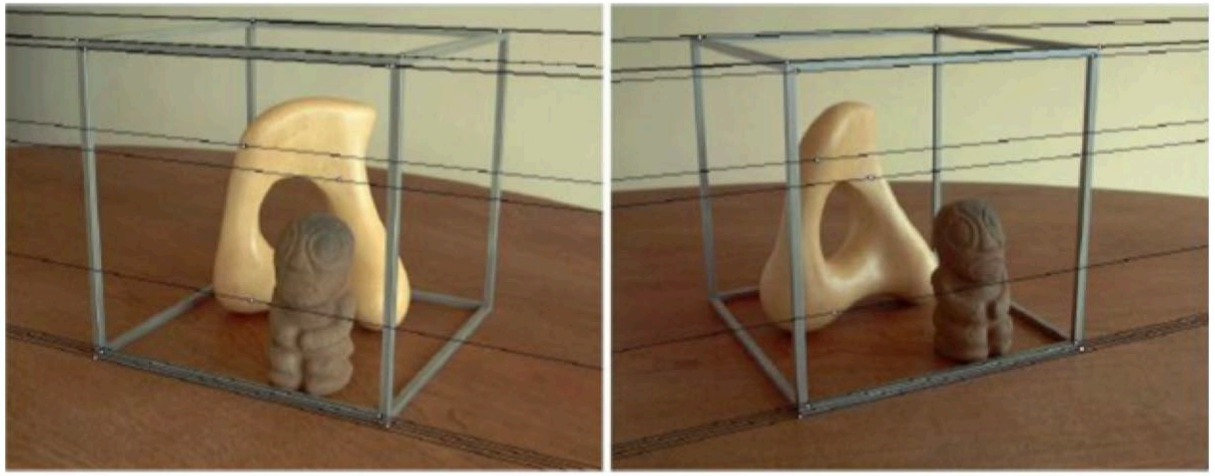
Key ideas: (1) cameras can be rotated in place through homographies
(2) frontoparallel => image plane parallel to the line connecting the cameras.

Stereo Rectification

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

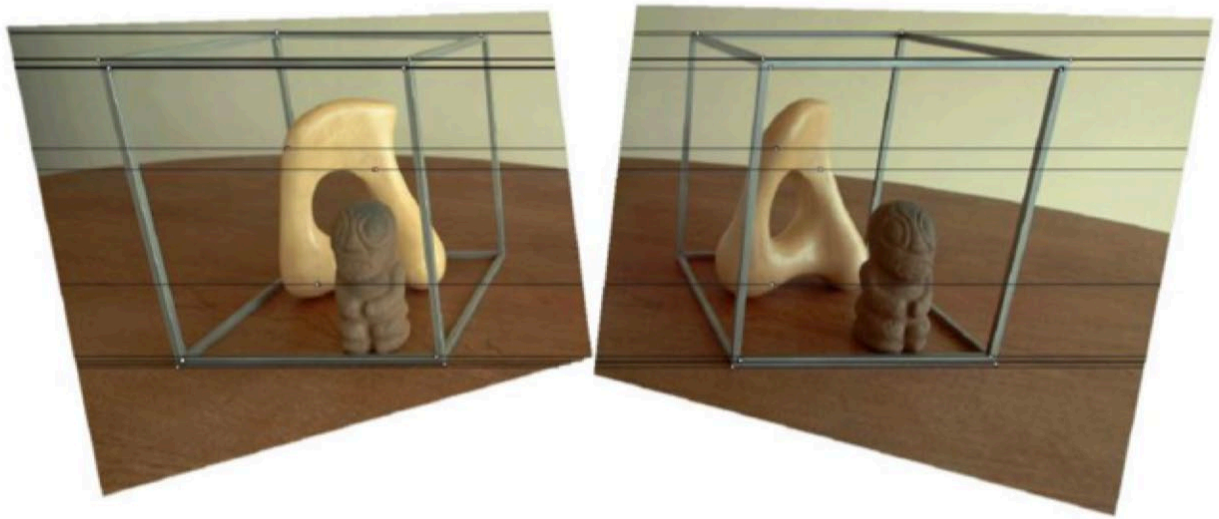






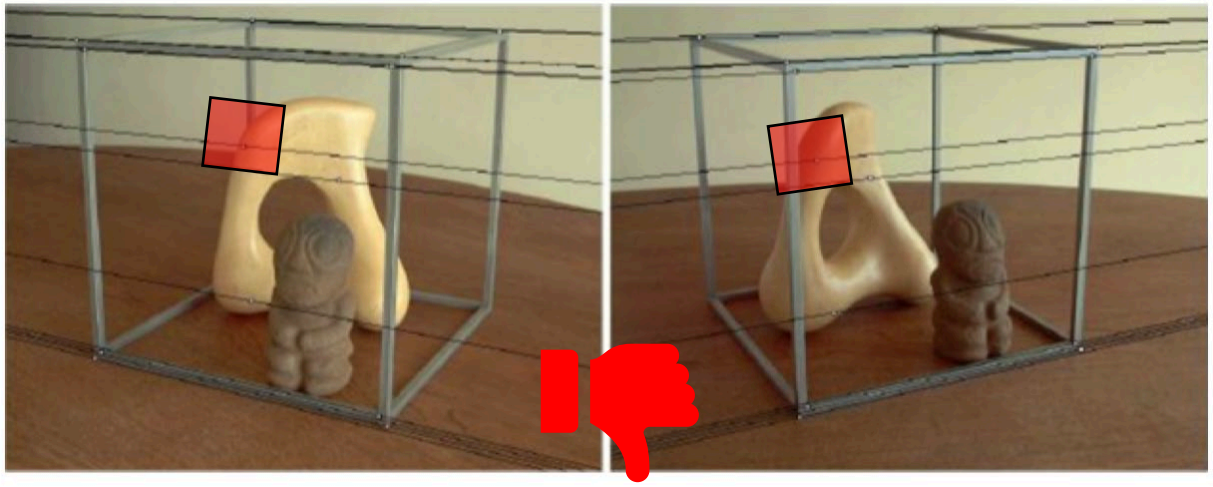
We know that a frontoparallel camera arrangement <=> horizontal epipolar lines

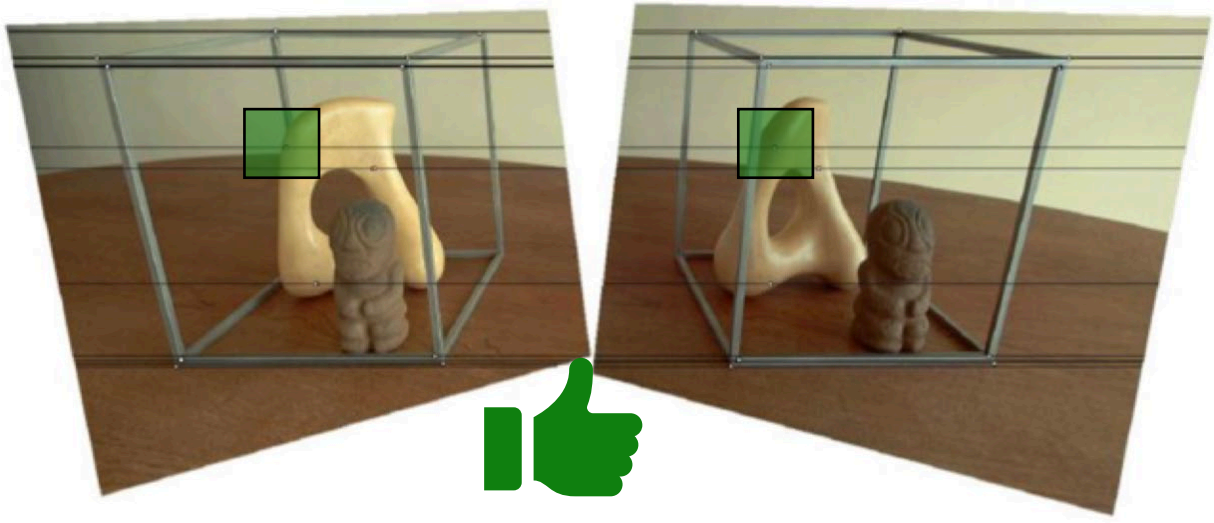
How can you make the epipolar lines horizontal?



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

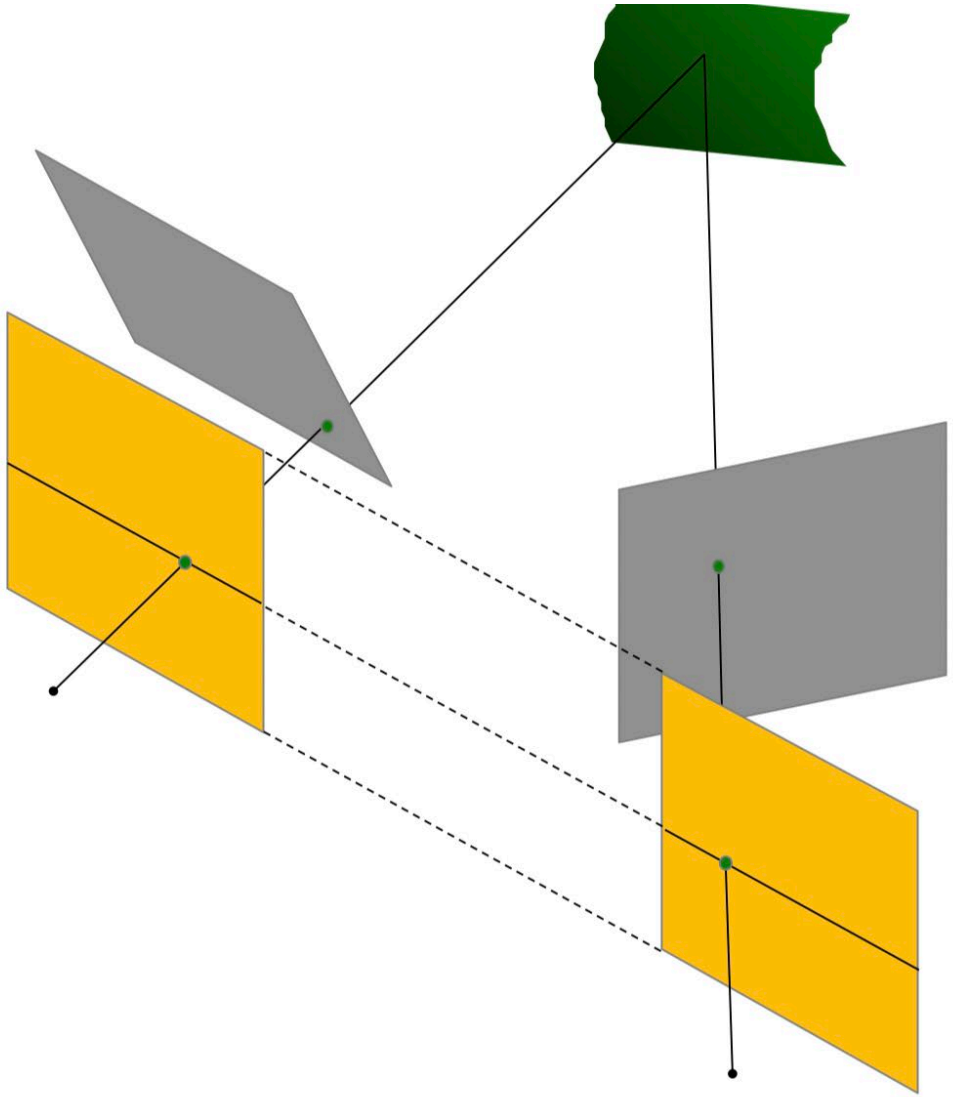




Key Idea

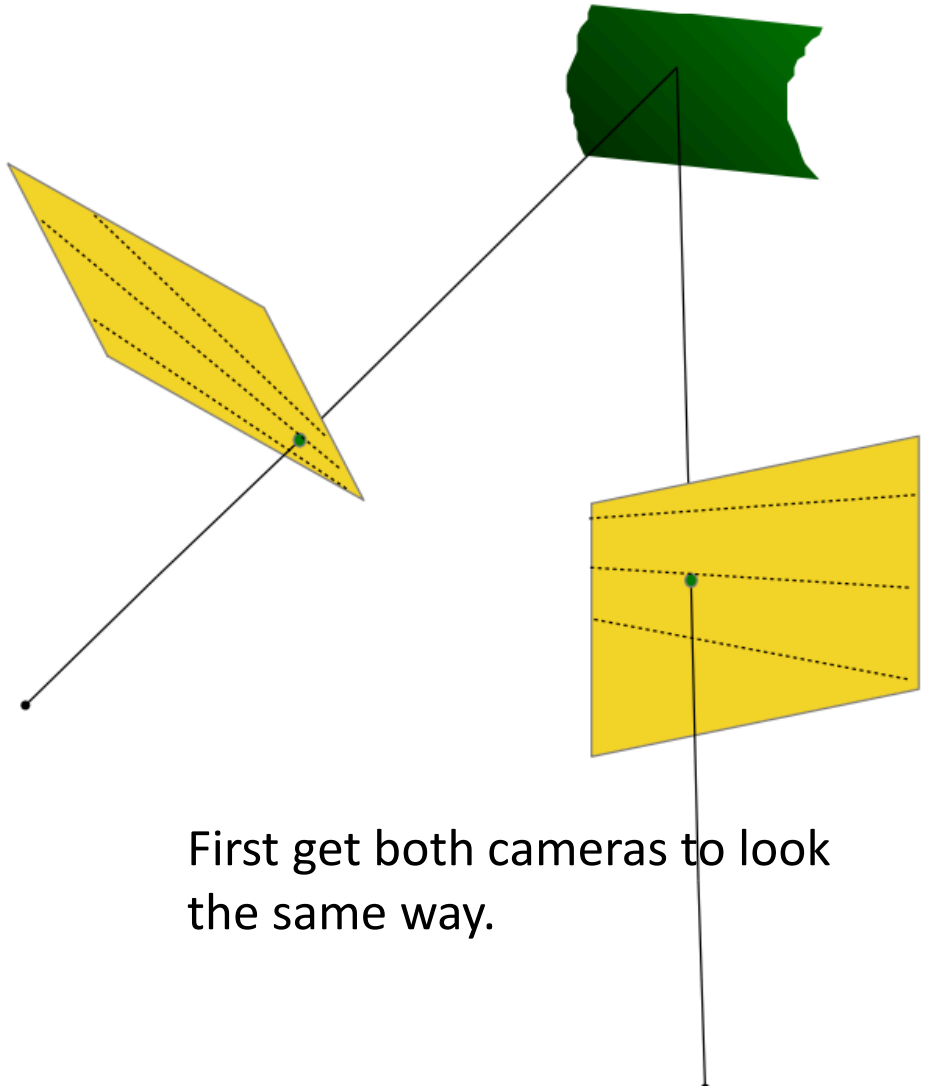
Reproject image planes onto a common plane parallel to the line between camera centers

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.

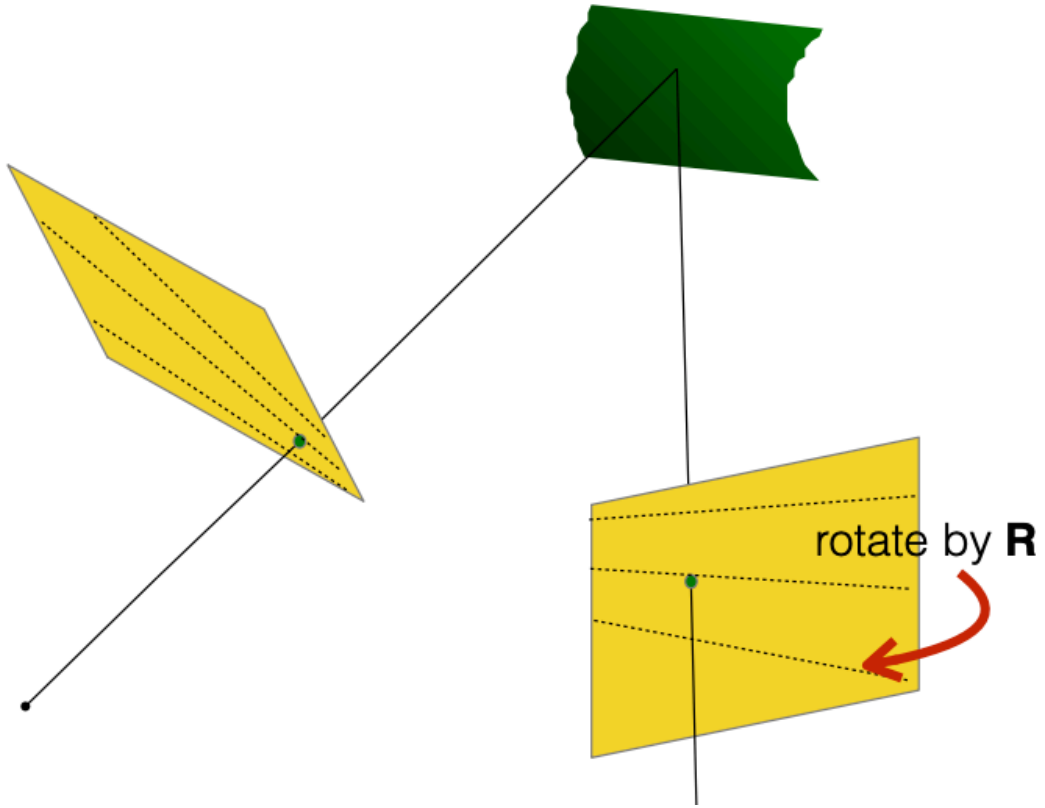
Stereo Rectification:



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

First get both cameras to look the same way.

Stereo Rectification:

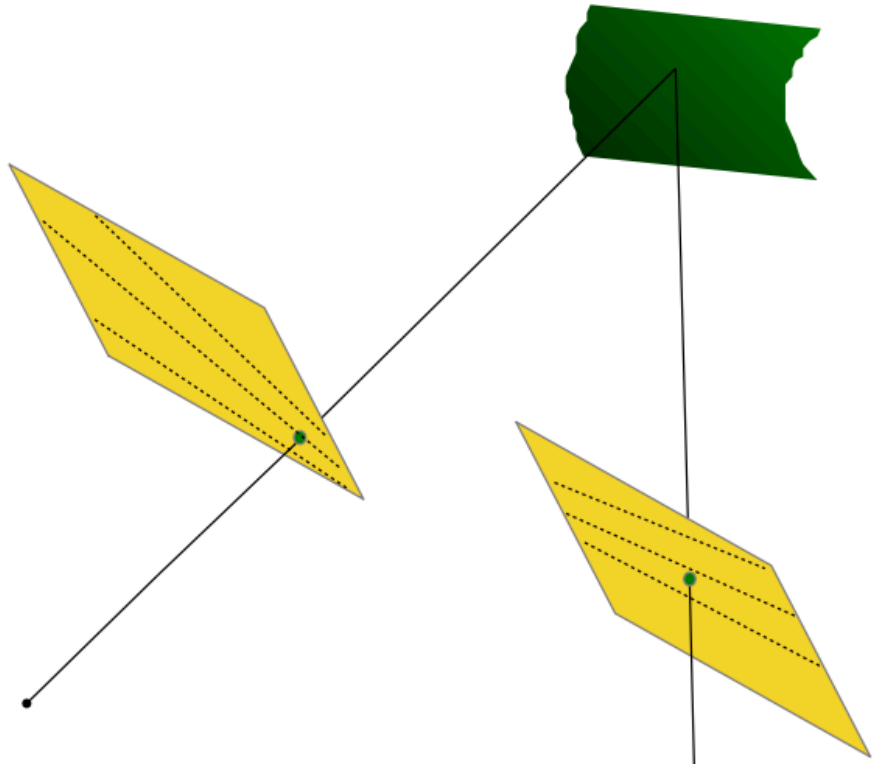


1. Compute **E** to get **R**
2. Rotate right image by **R**
3. Rotate both images by $\mathbf{R}_{rect}$

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

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

Stereo Rectification:



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

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.