CIS 5800

Machine Perception

Instructor: Lingjie Liu

Lec 24: April 28, 2025

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.

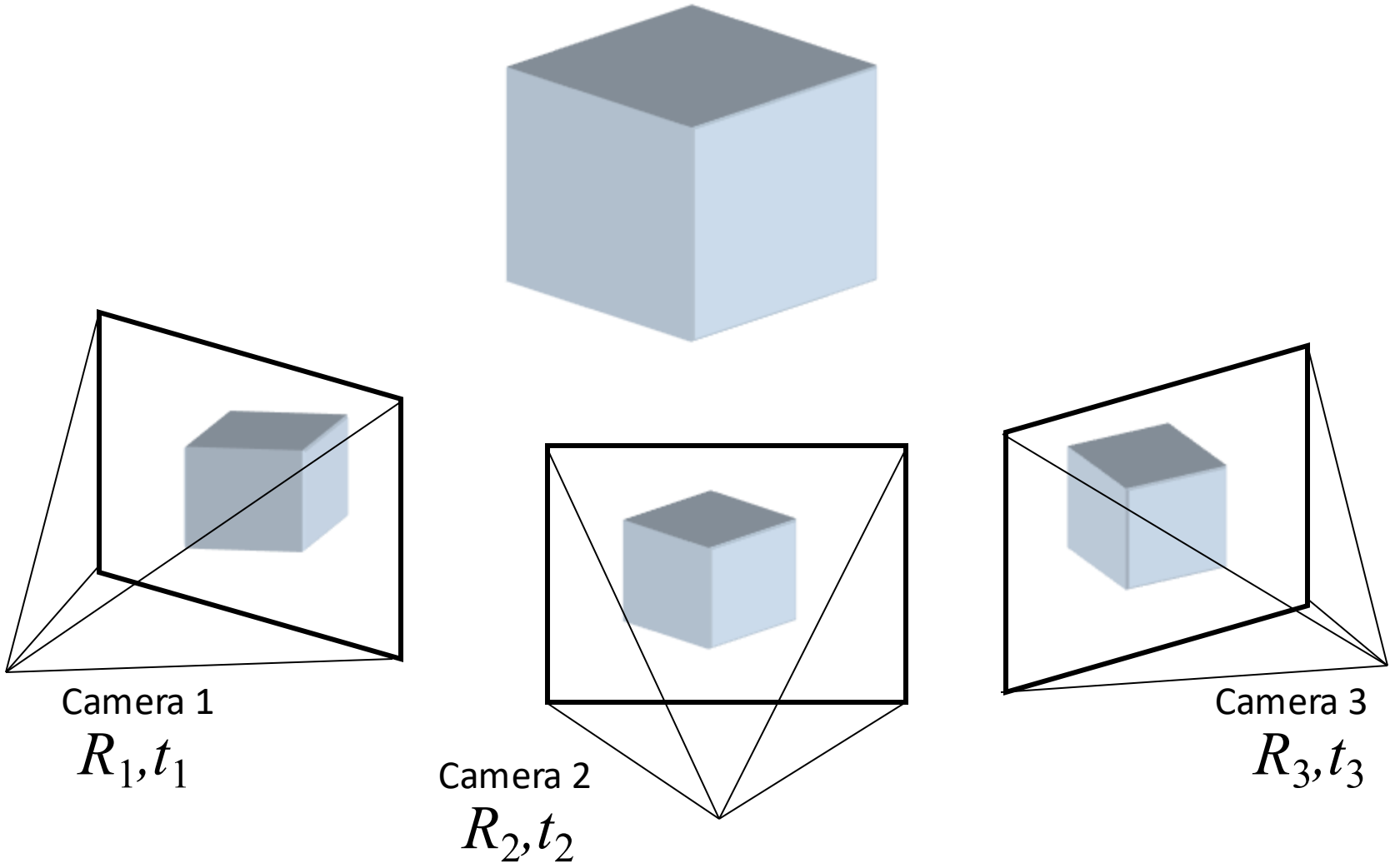
- Then, how to perform multi-view stereo (MVS)
  - Straightforward multi-baseline extension of 2-view stereo
  - The “plane sweep” technique for MVS

- Finally, improvement through dynamic programming.

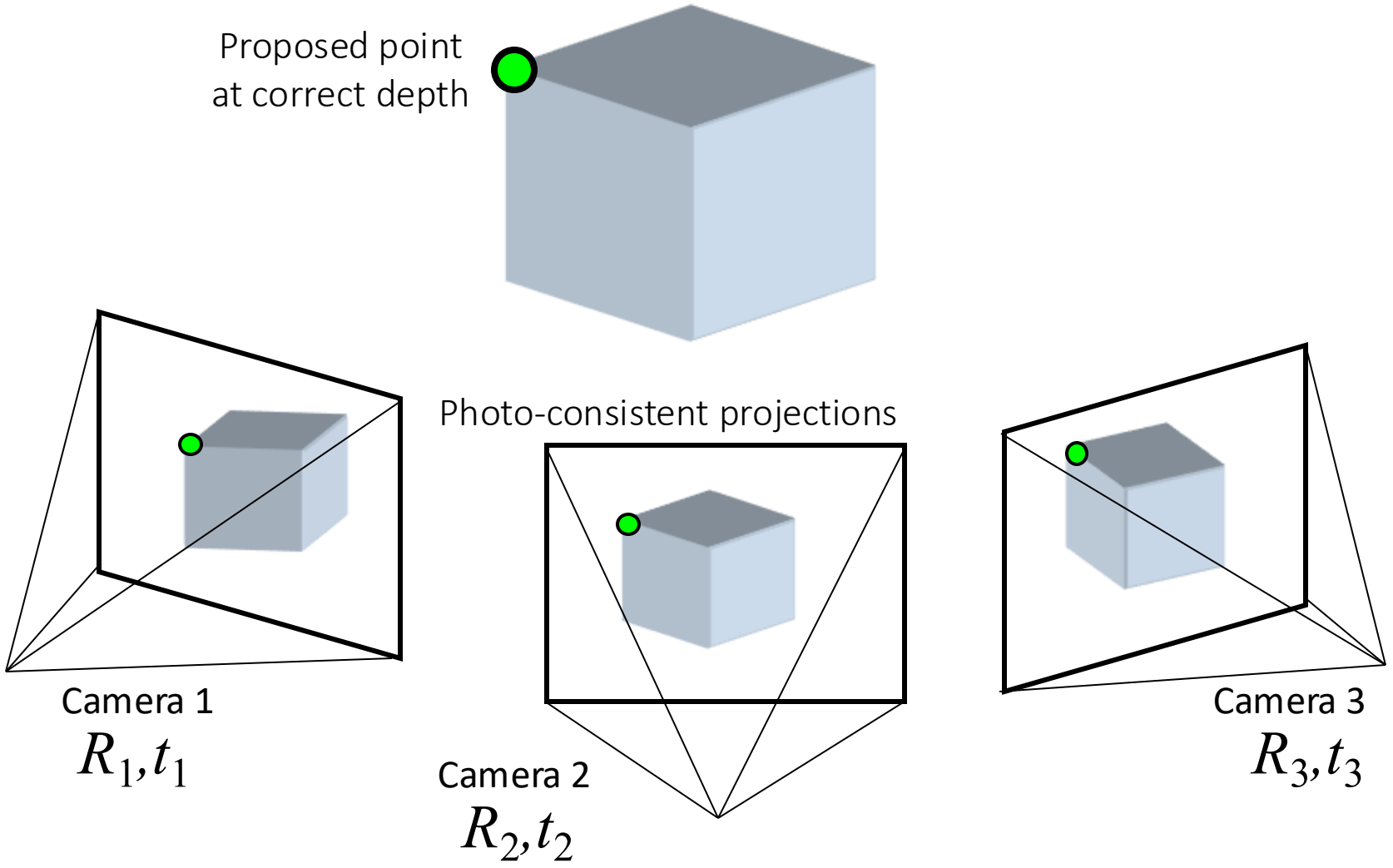
Plane Sweep

Plane-Sweep Stereo

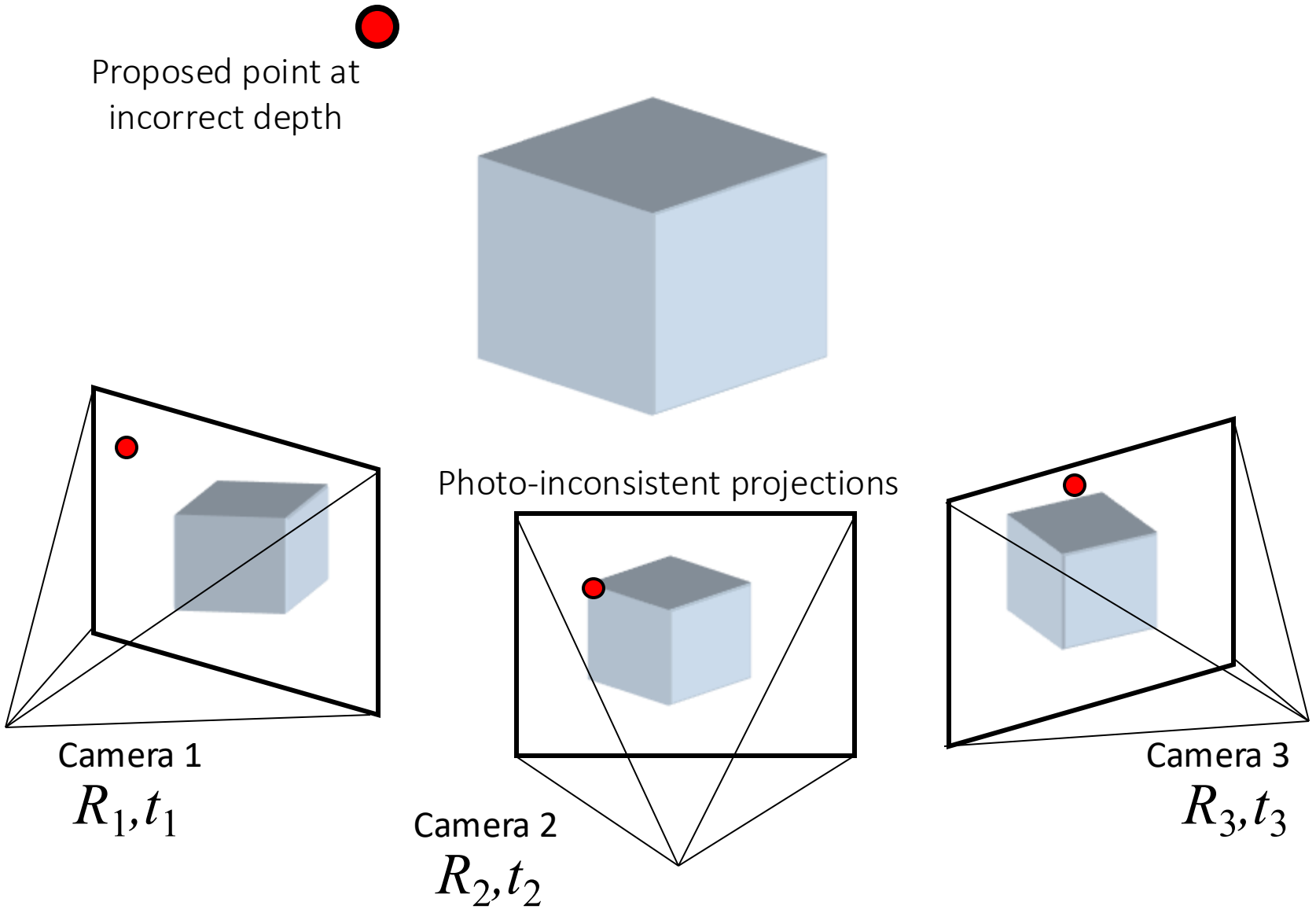
An efficient way to compute multi-view stereo



Plane-Sweep Stereo

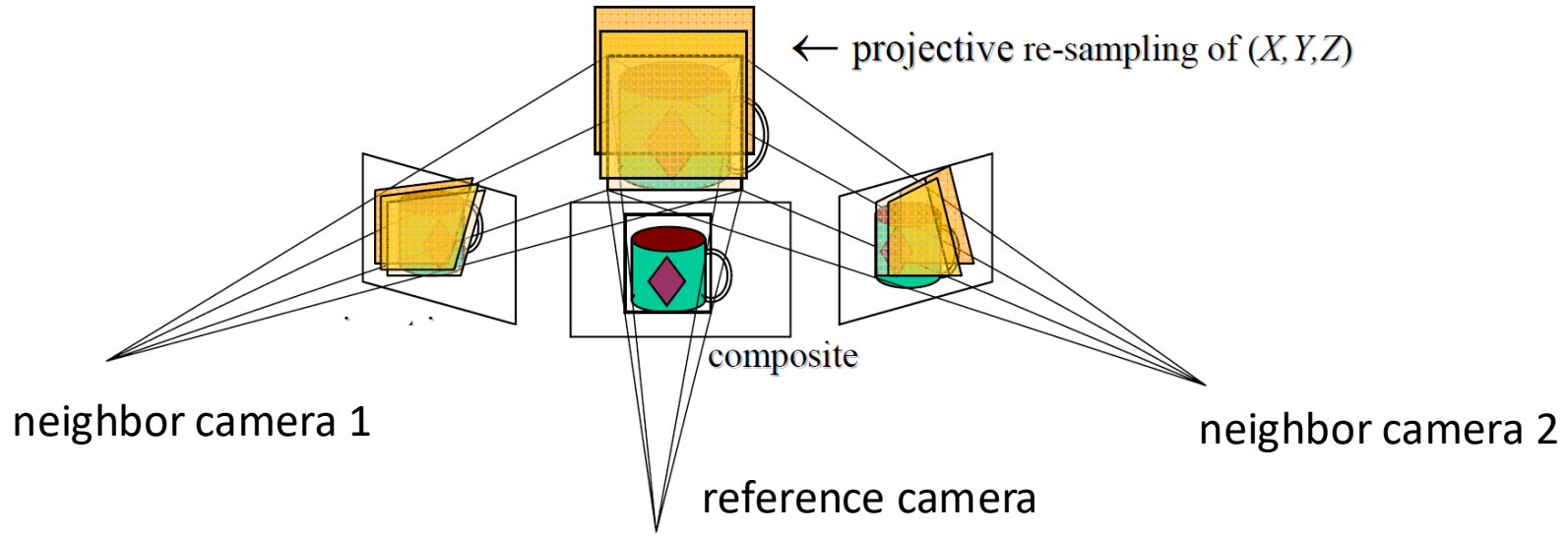


Plane-Sweep Stereo



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

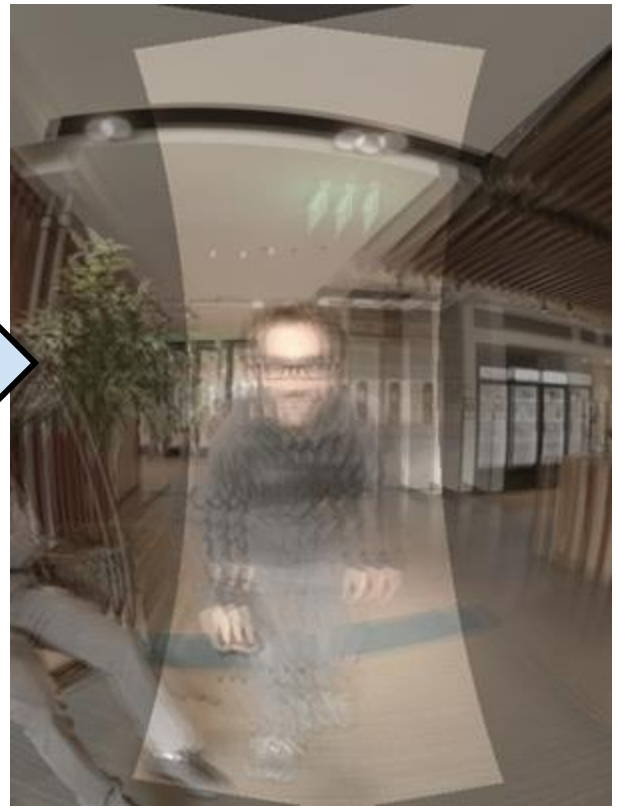
Right neighbor projected into reference camera's Z=z plane

Another example

Left neighbor

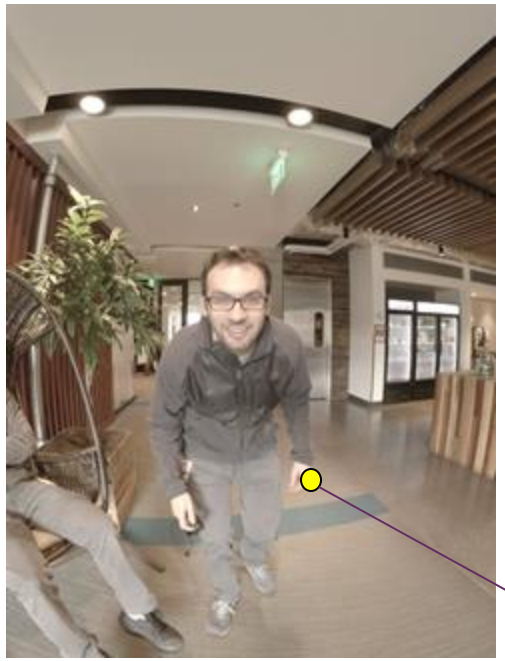
Reference image

Right neighbor



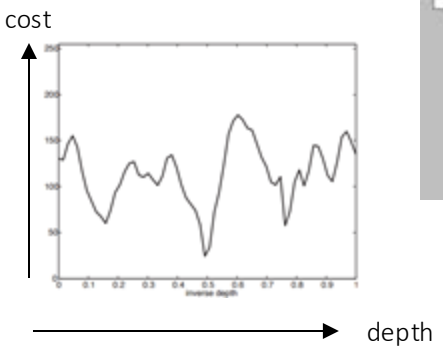
Planar image reprojections swept over depth (averaged)

Cost Volumes -> Depth Maps

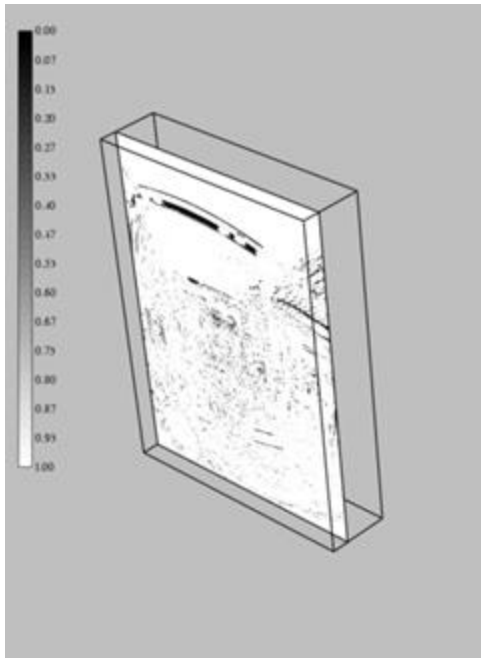


Reference image

Plane sweep



Single pixel's cost profile



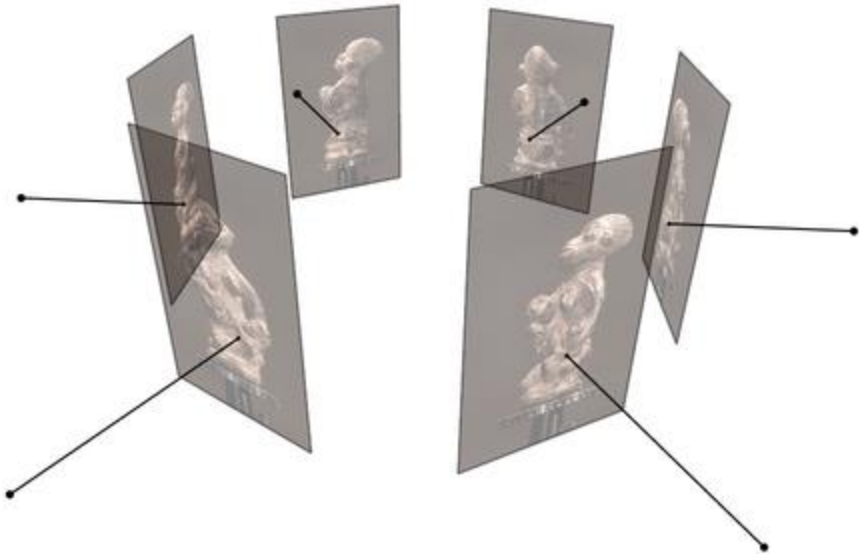
Full cost volume

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.

Improved Techniques for Multi-view Stereo

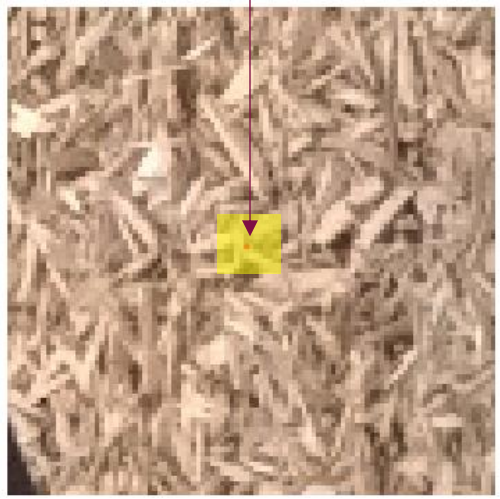
Components of Stereo Correspondence Matching

- Matching criterion (error function)
  - Quantify similarity of pixels
  - Options: **direct RGB intensity difference (SSD based)**, **NCC** etc.
- Aggregation method
  - How error function is accumulated
  - Options: Pixelwise, edgewise, **window-wise**, segment-wise …
- Optimization and winner selection
  - Examples: **Winner-take-all**, **dynamic programming**, graph cuts, belief propagation

Alternatives to SSD based matching

Window Matching for Stereo Correspondence Finding

W_i



Searching for matching W_j along epipolar lines

Note: these are unrectified images, so epipolar lines are not horizontal, but in practice, matching is done after rectification.

Popular window matching scores beyond SSD

- SSD (Sum of Squared Differences) $\sum_{x,y} |W_1(x,y) - W_2(x,y)|^2$
- SAD (Sum of Absolute Differences) $\sum_{x,y} |W_1(x,y) - W_2(x,y)|$
- ZNCC (Zero-mean Normalized Cross Correlation), sometimes just “NCC”

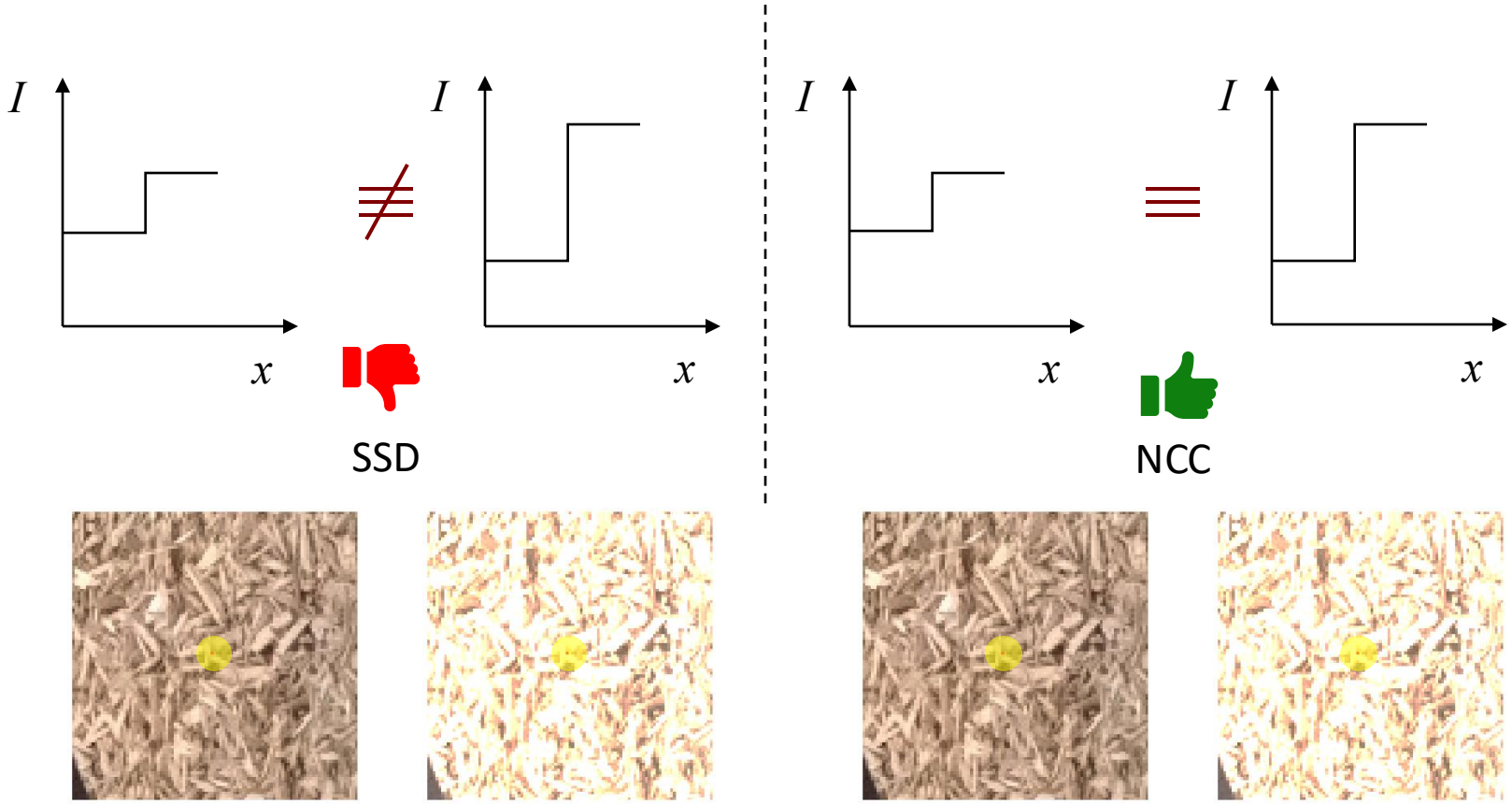
$$\frac{\sum_{x,y}(W_1(x,y) - \overline{W_1})(W_2(x,y) - \overline{W_2})}{\sigma_{W_1}\sigma_{W_2}}$$

  - where $\overline{W_i} = \frac{1}{n}\sum_{x,y} W_i$ $\quad \sigma_{W_i} = \sqrt{\frac{1}{n}\sum_{x,y}(W_i - \overline{W_i})^2}$

NCC Injects Some Invariance

$$\frac{\sum_{x,y}(W_1(x,y)-\overline{W_1})(W_2(x,y)-\overline{W_2})}{\sigma_{W_1}\sigma_{W_2}}$$

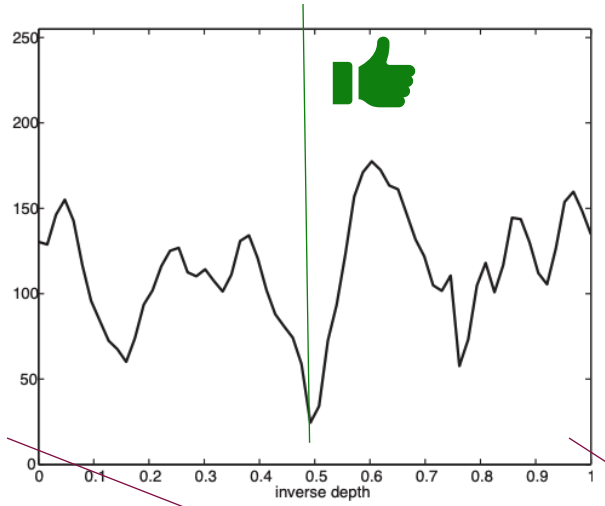
- NCC instead of SSD helps account for the fact that **pixels might not be exactly same, because images have been captured with different illumination / from different cameras.**



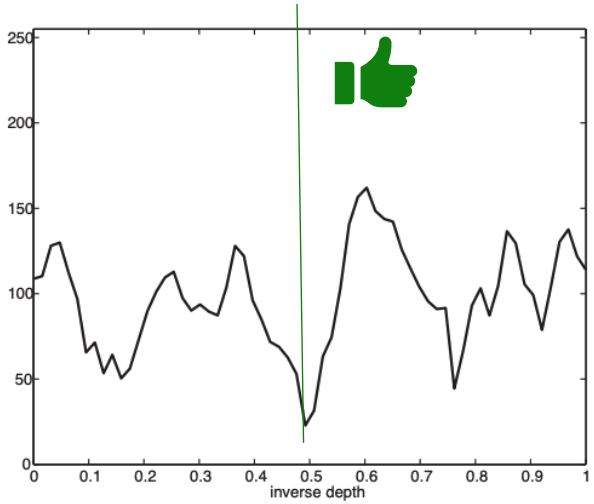
False Positives in NCC

But the invariance in NCC can also produce more “false positives”, because it can be too forgiving on mismatched patches.

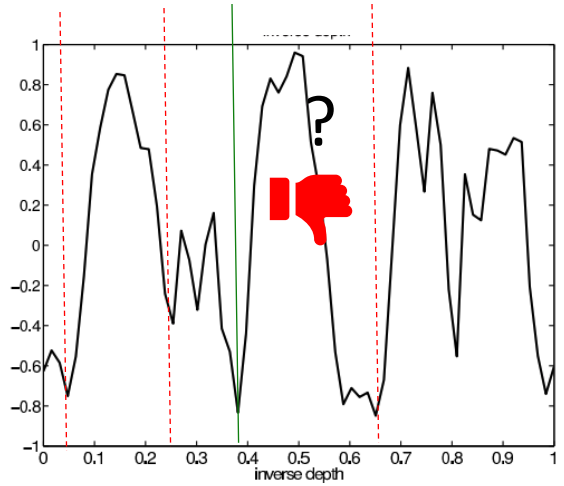
SSD

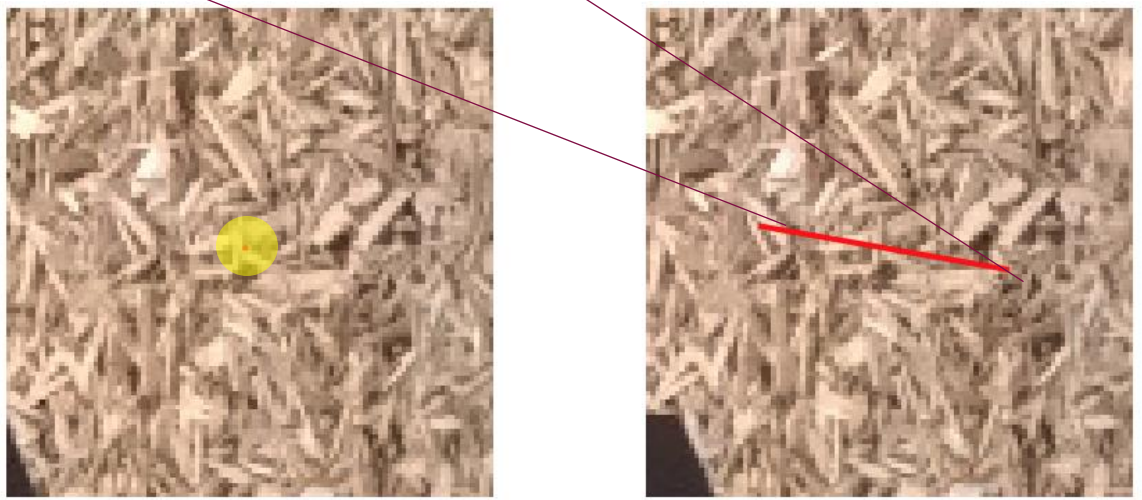


SAD



NCC





Components of Stereo Correspondence Matching

- Matching criterion (error function)
  - Quantify similarity of pixels
  - Options: **direct RGB intensity difference (SSD based)**, **NCC** etc.
- Aggregation method
  - How error function is accumulated
  - Options: Pixelwise, edgewise, **window-wise**, segment-wise …
- Optimization and winner selection
  - Examples: **Winner-take-all**, **dynamic programming**, graph cuts, belief propagation

Doing better than “Winner-Take-All” matching

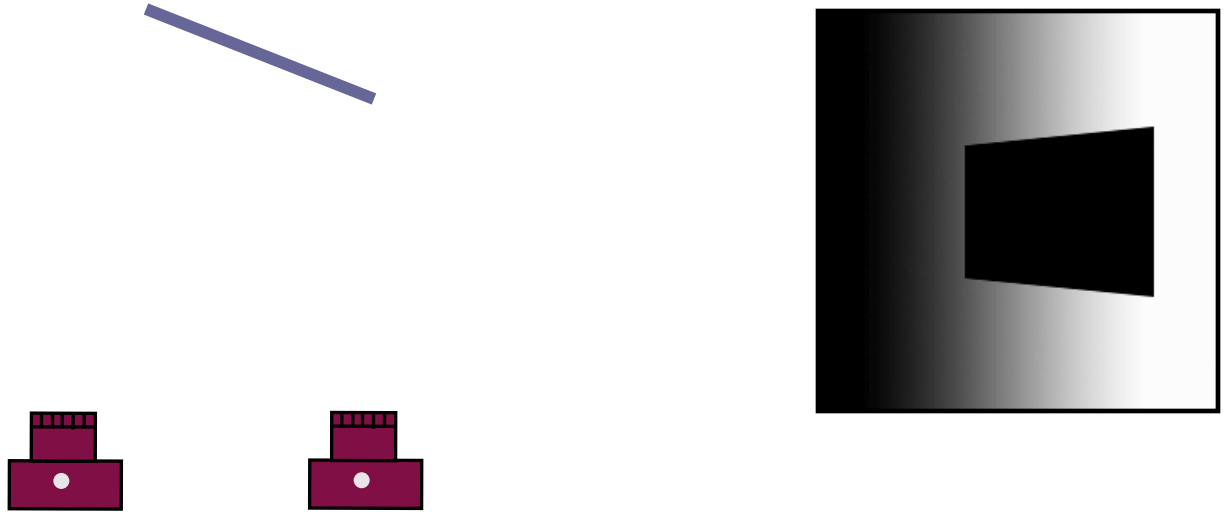
Winner-Take-All is Fragile

- Recall, until now, we have assigned independent correspondences for each window / pixel / region based purely on lowest SSD / NCC etc.
- But this can be fragile and ignore signs of erroneous correspondences:
  - 1. A correspond to B, but B does not correspond to A.

Left-Right Consistency Checking

- Can check for consistency between correspondence from left to right image, and correspondence computed in the opposite direction.
  - A way to discard outlier correspondences.

- Sometimes wrong! E.g. Slanted plane: Matching between M pixels and N pixels



Winner-Take-All is Fragile

- Recall, until now, we have assigned independent correspondences for each window / pixel / region based purely on lowest SSD / NCC etc.
- But this can be fragile and ignore signs of erroneous correspondences:
  - 1. A correspond to B, but B does not correspond to A.
  - 2. Correspondences along a line might be completely jumbled.
  - 3. Or, many points might all map to a single point.

- Technically, 2 and 3 are not implausible, and these might even be correct correspondences, but this is very rare. And knowing this can allow us to acquire better dense correspondences.

Solutions: Constrain the correspondences!

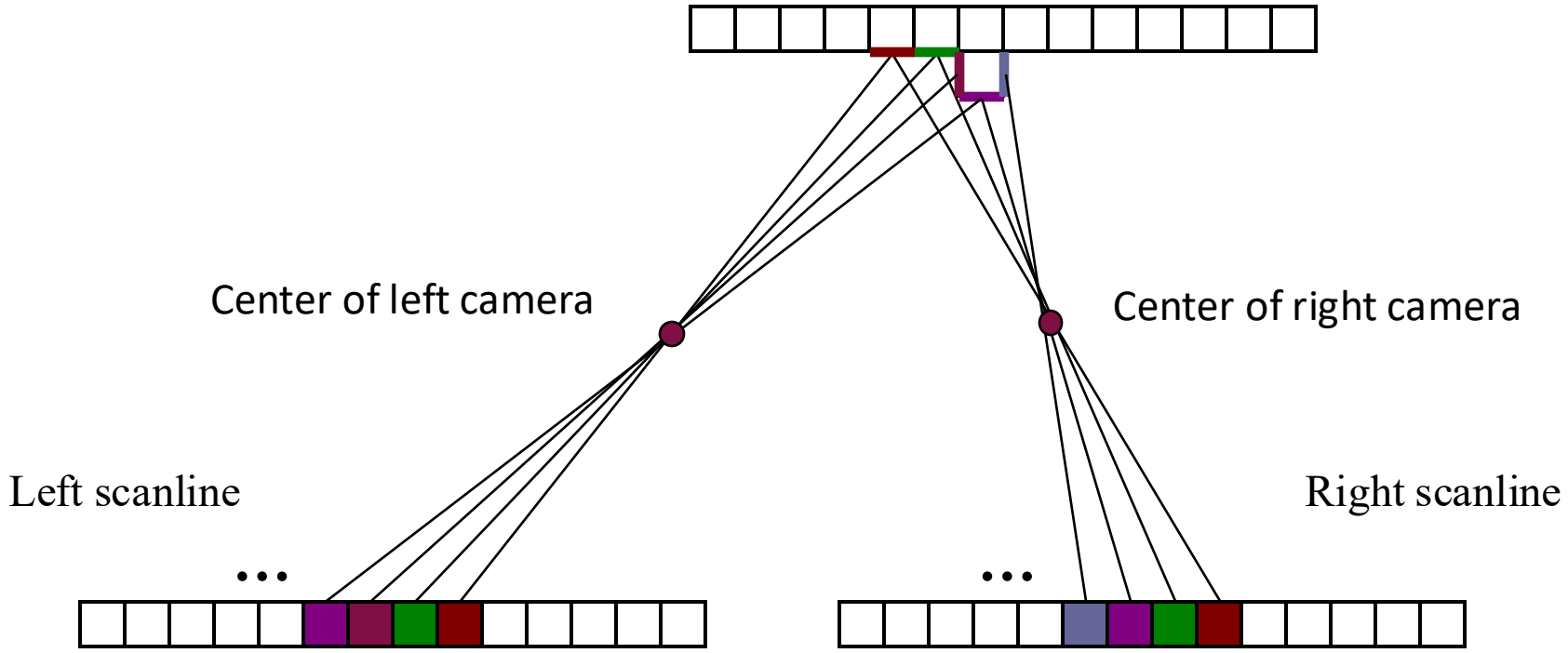
Rather than greedy pixelwise winner-take-all matching, we need to use more "global" methods, imposing constraints

- Ordering constraint:
  - Impose same matching order along scanlines
- Uniqueness constraint:
  - Each pixel in one image maps to unique pixel in other

Can encode these constraints easily in dynamic programming

Ordering Constraint Illustration

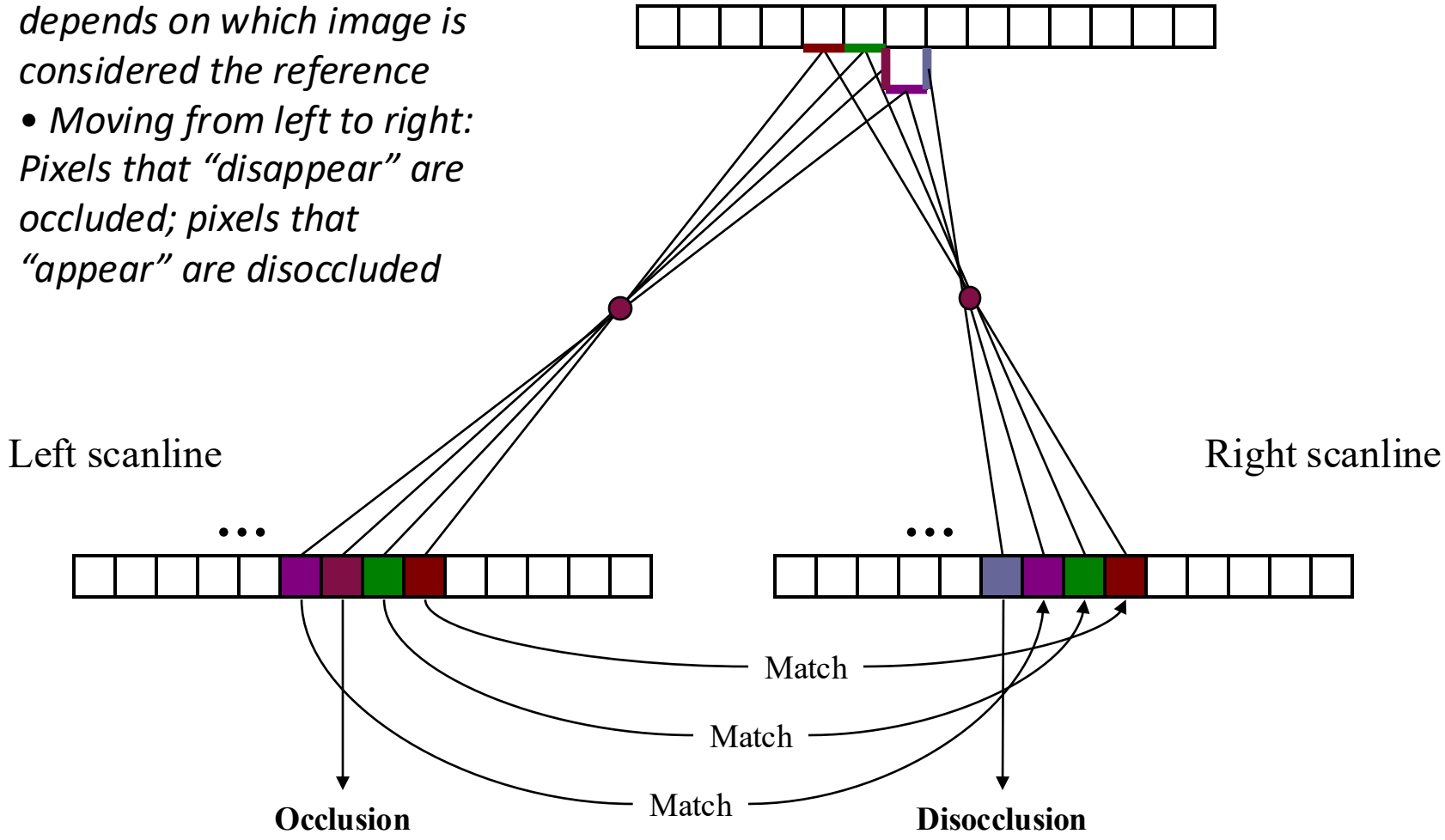
For illustration, we will assume no aggregation i.e. purely pixelwise



(NOTE: We're depicting the actual image behind the camera, rather than the virtual image in front)

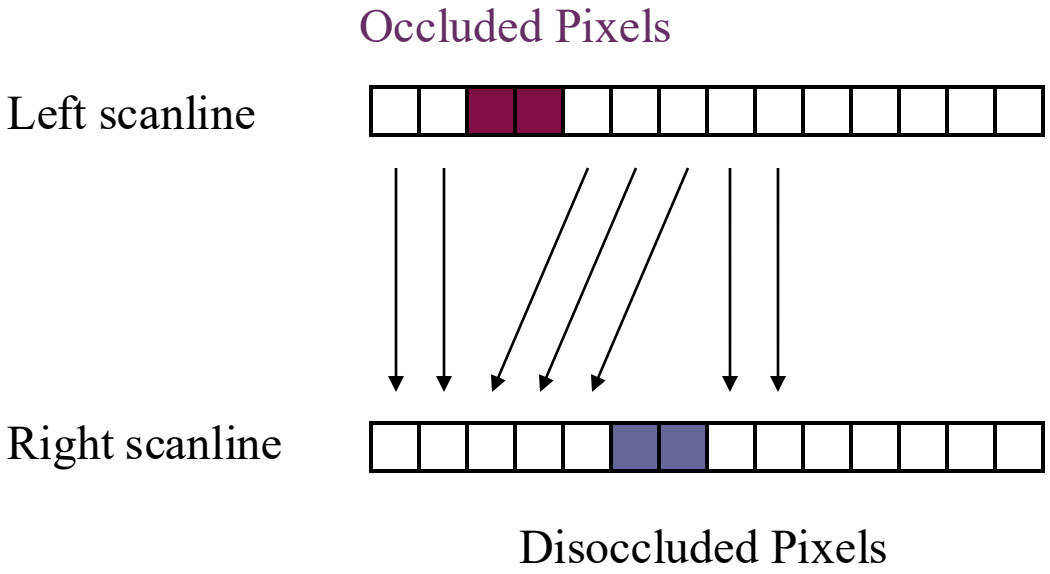
Ordering Constraint Illustration

- *Definition of occlusion/disocclusion depends on which image is considered the reference*
- *Moving from left to right: Pixels that "disappear" are occluded; pixels that "appear" are disoccluded*



Ordering Constraint Illustration

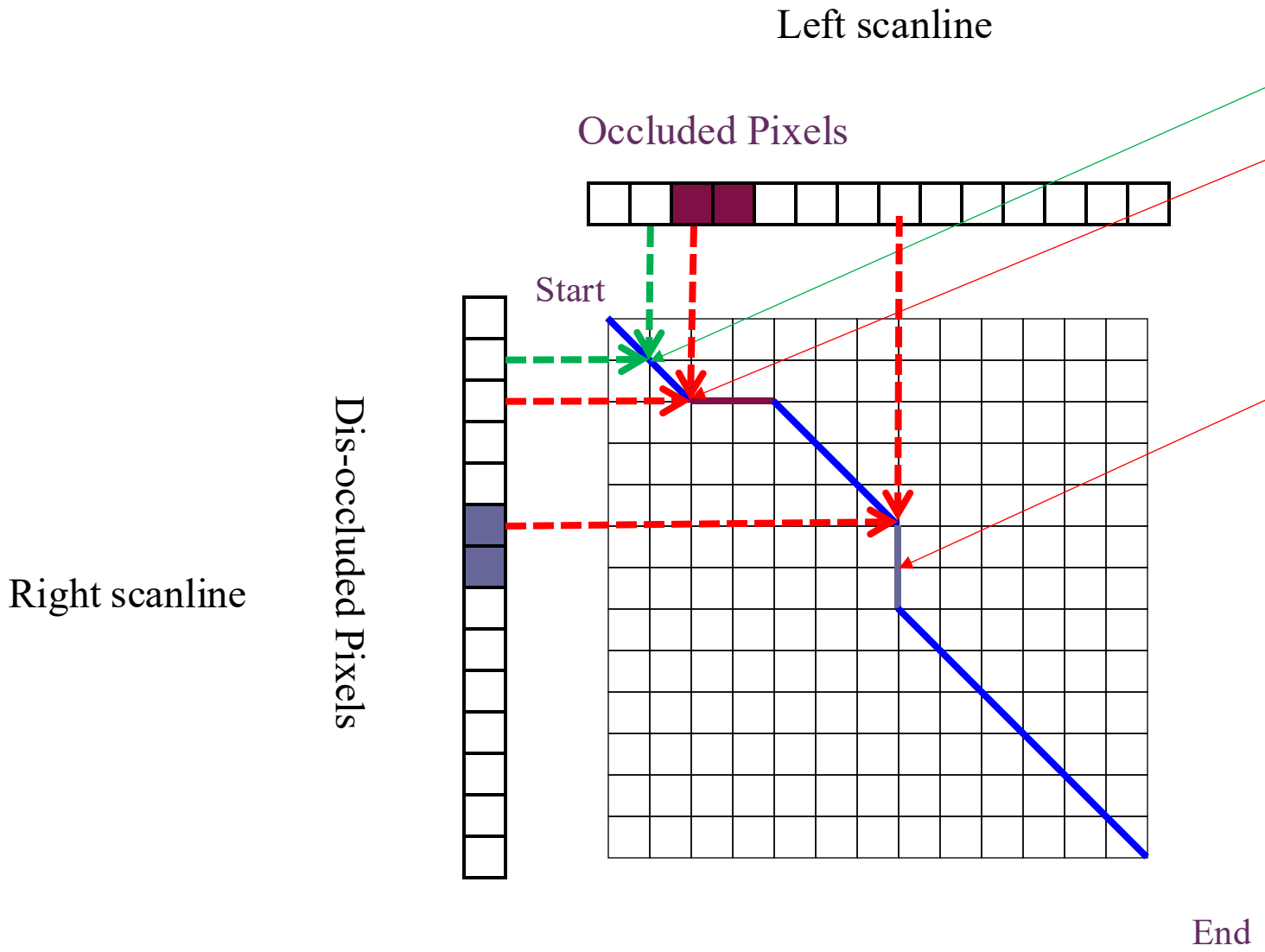
For matching the two scanlines in an orderly way, we can assume monotonicity*.
If pixel i matches to i', then $j > i$ can only match to some $j' > i'$.



Three cases:

- Sequential – cost of match
- Occluded – cost of no match
- Disoccluded – cost of no match

Overview: Stereo Matching with Dynamic Programming



Matched, move diagonally

no matching pixel found, move right

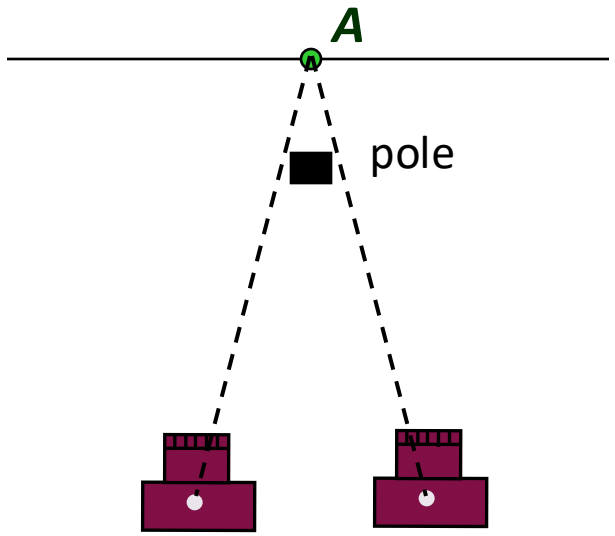
new unmatched pixels, move down

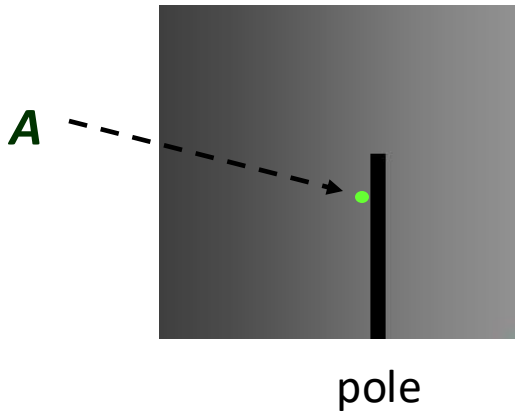
Given cost definitions for matches, skips, etc., dynamic programming searches for the optimal lowest-cost path through grid, producing improved correspondences!

This is no longer greedy matching; it optimizes “globally” within scanlines.

Ordering Constraint is Sometimes Wrong!

- Ordering constraint is often violated with thin structures in the foreground.

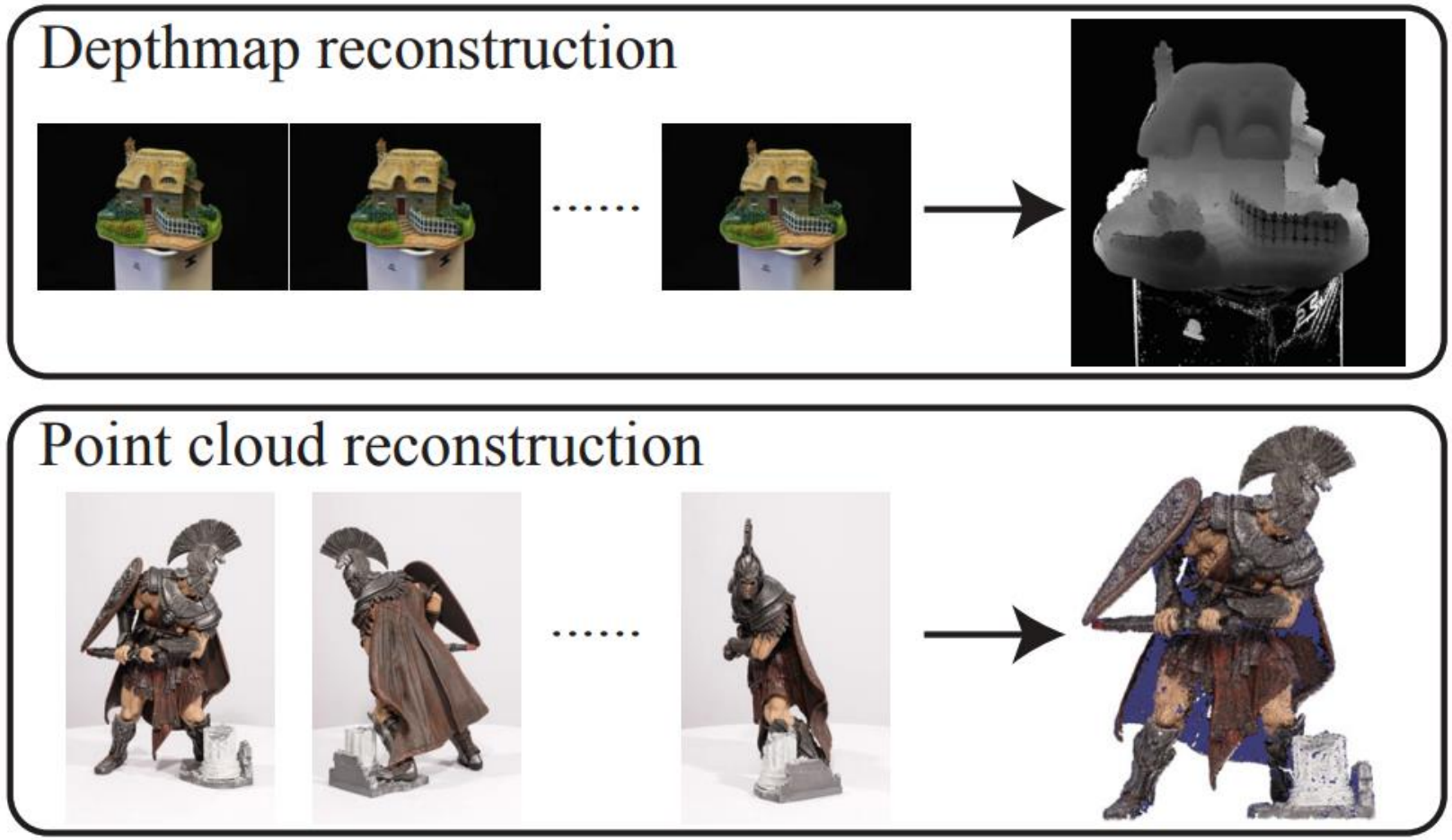




Alternative dense shape representations

Beyond depth maps and point clouds

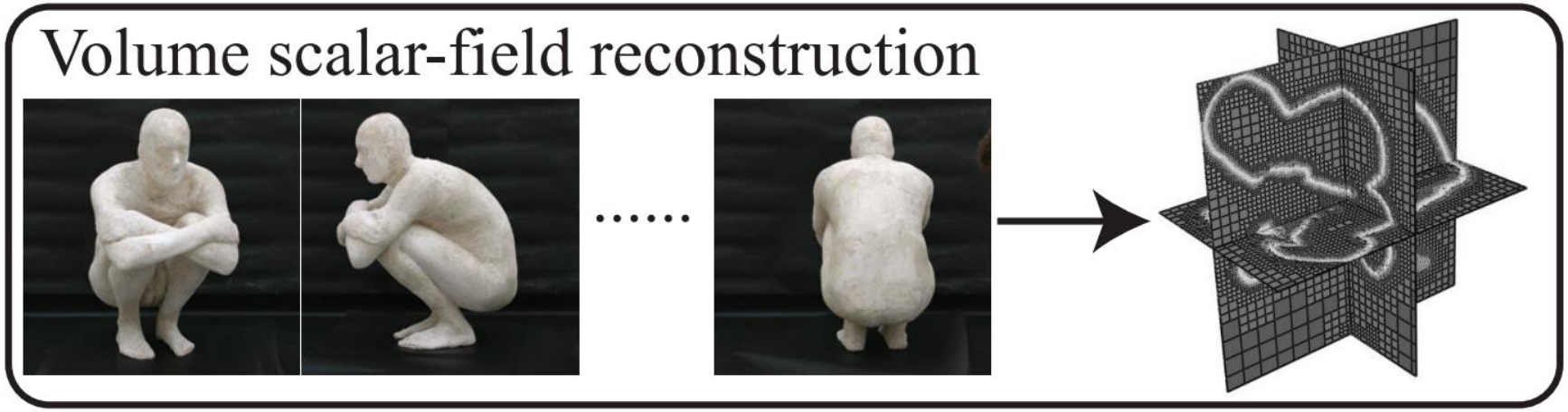
- We have seen so far:



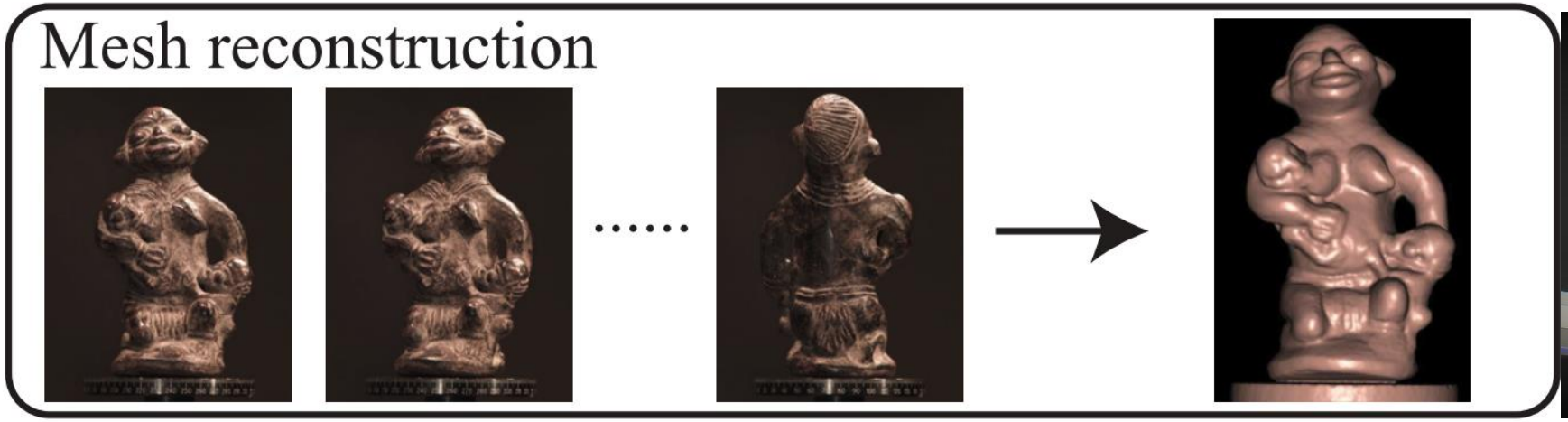
Hernandez and Furakawa, 2015

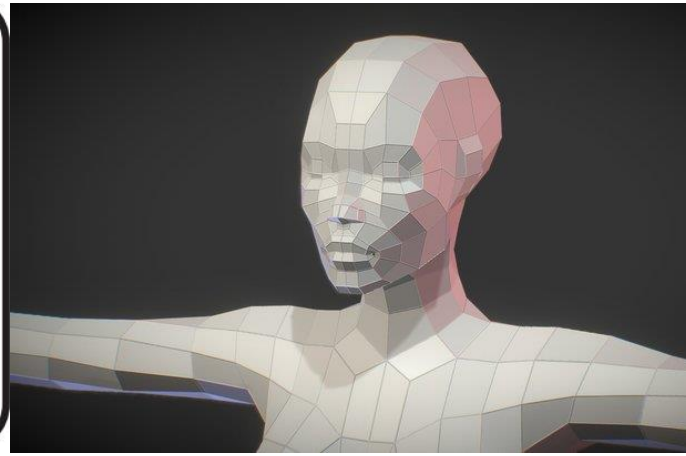
Beyond depth maps and point clouds

- But there are more sophisticated ways to represent 3D shape:



(e.g. signed distance function fields)

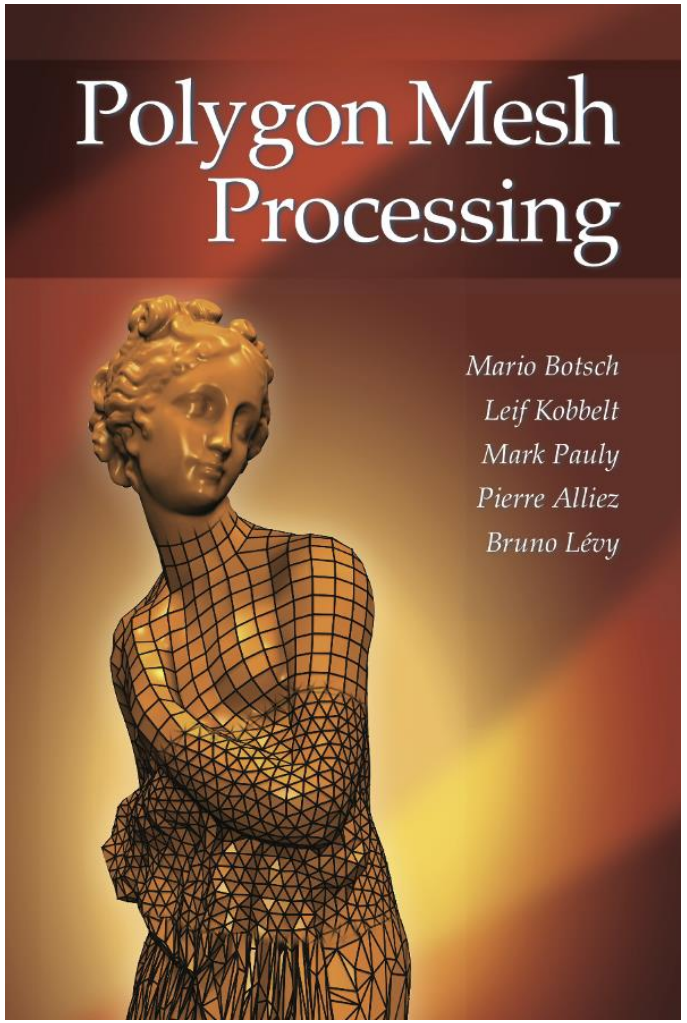


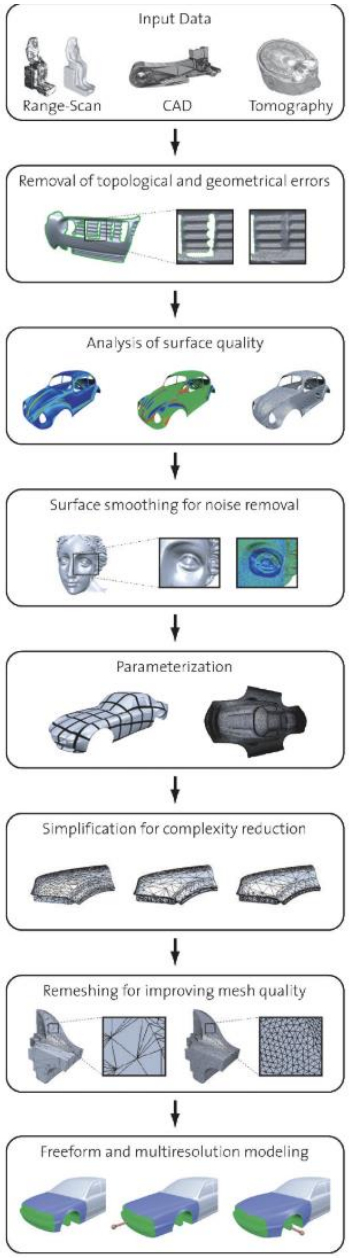


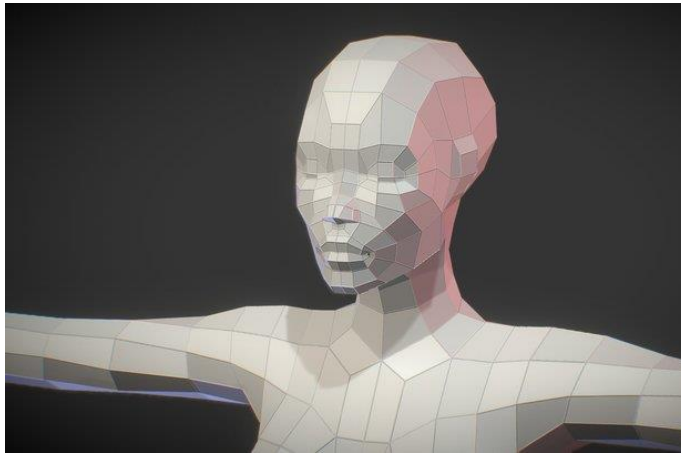
(lots of work from the graphics community on converting point clouds into compact mesh models)

Hernandez and Furakawa, 2015

Beyond depth maps and point clouds







Hernandez and Furakawa, 2015

The Rest of the Multi-View Stereo Pipeline

- Selecting the right images where some 3D point of interest is consistently visible: Visibility Estimation

Hernandez and Furakawa, 2015

Visibility Estimation

- In the techniques above, like plane-sweep MVS, we assumed we could see the same point in all images. Otherwise, a view with an occluded point could cause high variance even at the correct depth plane.
- Particularly critical when scaling to large numbers (e.g. millions) of images.

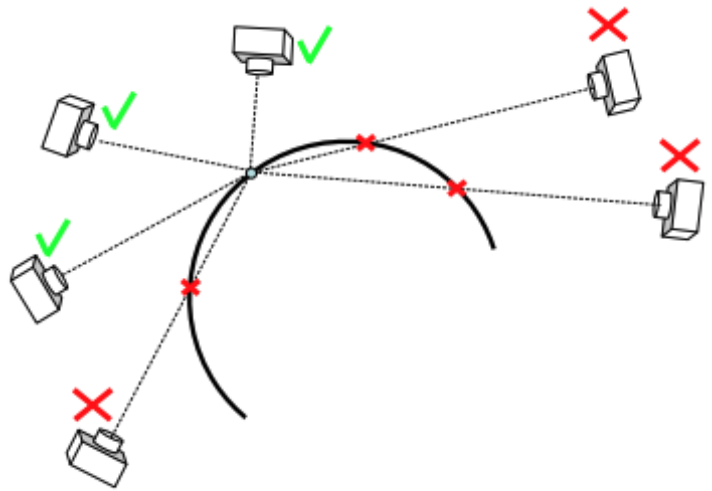


Figure 2.7: Occlusion problem. In order to compute geometry using photo-consistency, the camera visibility is required. At the same time, in order to compute the camera visibility, the geometry is required.

Hernandez and Furakawa, 2015

Finding Clusters of Images to Scale MVS

- For each image being considered as a “reference”, create a “cluster” of views of computationally tractable size that:
  - Overlap in terms of visible points, and
  - Provide informative viewpoints for precise triangulation
- To achieve these two criteria, we consider:
  - Number of matched points between view pairs (more => higher overlap),
  - How far away the cameras are in space (larger baselines => better triangulation)
- Then solve MVS within each cluster, and combine.
- Finally, to keep the number of clusters computationally tractable, carefully select diverse reference views that maximize coverage, and only form clusters for them. i.e. not every image gets to be a reference view.

View Clusters in Large-Scale MVS

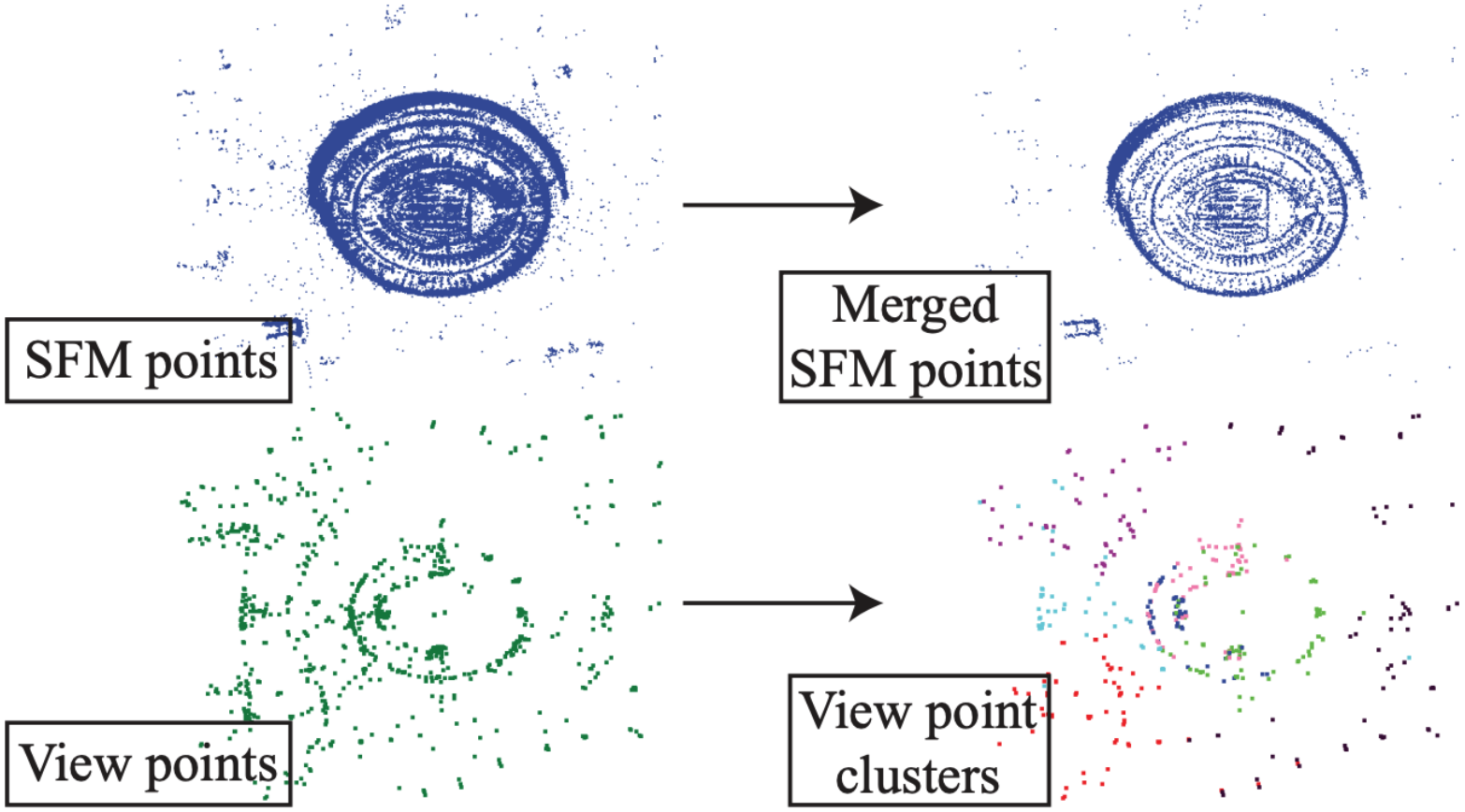


Figure 2.9: Large scale view clustering of internet images of the Colosseum [69]. Top: The first step of the algorithm is to merge SFM points to enrich visibility information (SFM filter). Bottom: Sample results of the view clustering algorithm. View points belonging to extracted clusters are illustrated in different colors.

Yasutaka Furukawa, Brian Curless, Steven M. Seitz, and Richard Szeliski. Towards Internet-scale multi-view stereo. CVPR 2010

Iterative Visibility Estimation

- There is a chicken-and-egg problem in visibility estimation and geometry reconstruction.
  - So, above, we discussed how the initial SfM scene geometry + cameras can be used to select view clusters for MVS
  - But this can repeat: more fine-grained visibility estimation, and in turn, improved MVS, can come from iterating between running MVS, and then using the currently reconstructed scene geometry to compute occlusions, and repeating.

Taking stock of what we've covered in this course

The Ground We've Covered: 1-View Geometry

In this case, inferring geometry relies on some knowledge

knowledge

- Parallel-ness of world lines + image correspondences of those lines → Vanishing Points
- 3D point locations + image point correspondences
- Distances of world collinear points + point correspondences → Length ratios
- Planarity of points + point correspondences → Homographies

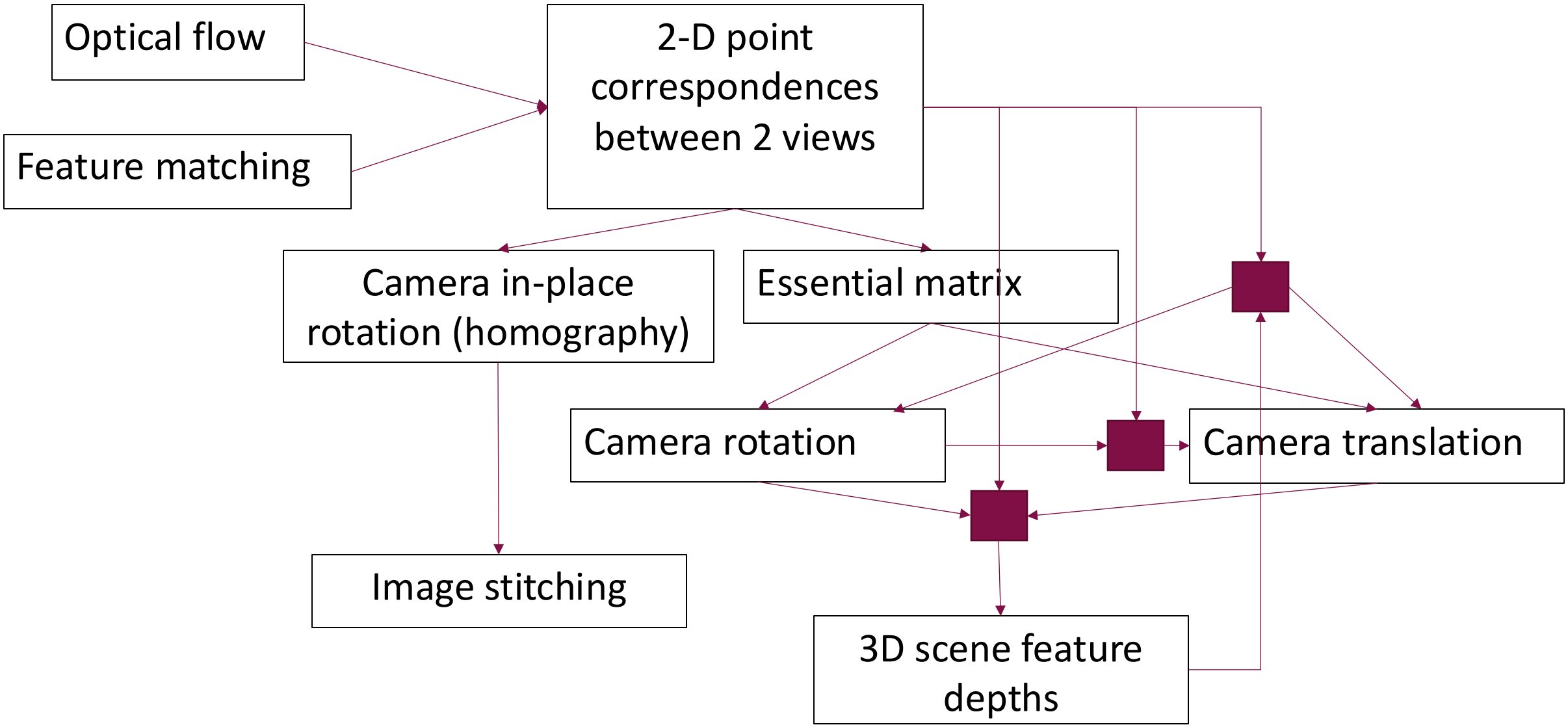
Vanishing Points, 3D point locations + image point correspondences → Camera intrinsics

Camera intrinsics, Length ratios, Homographies → Camera extrinsics

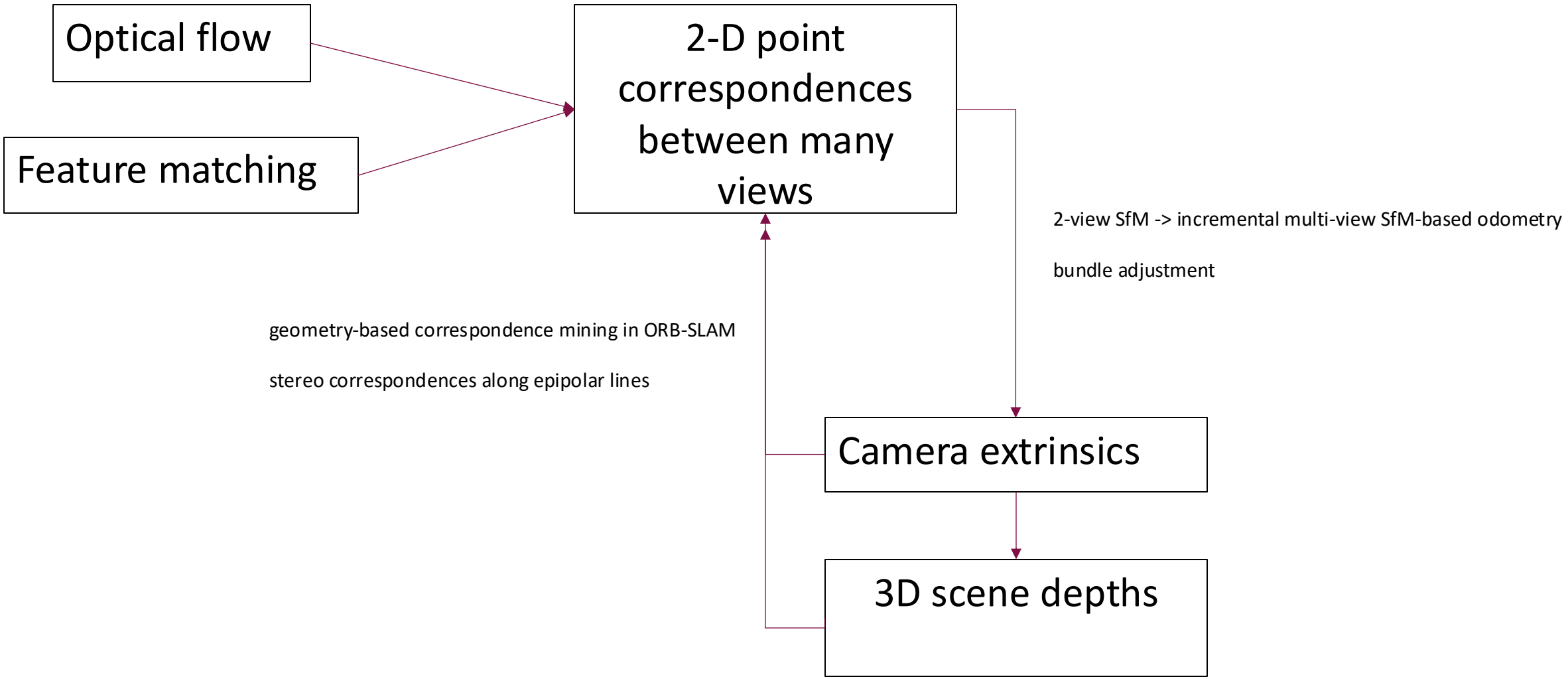
Camera intrinsics, Camera extrinsics → Projecting AR objects

Homographies → Virtual billboards

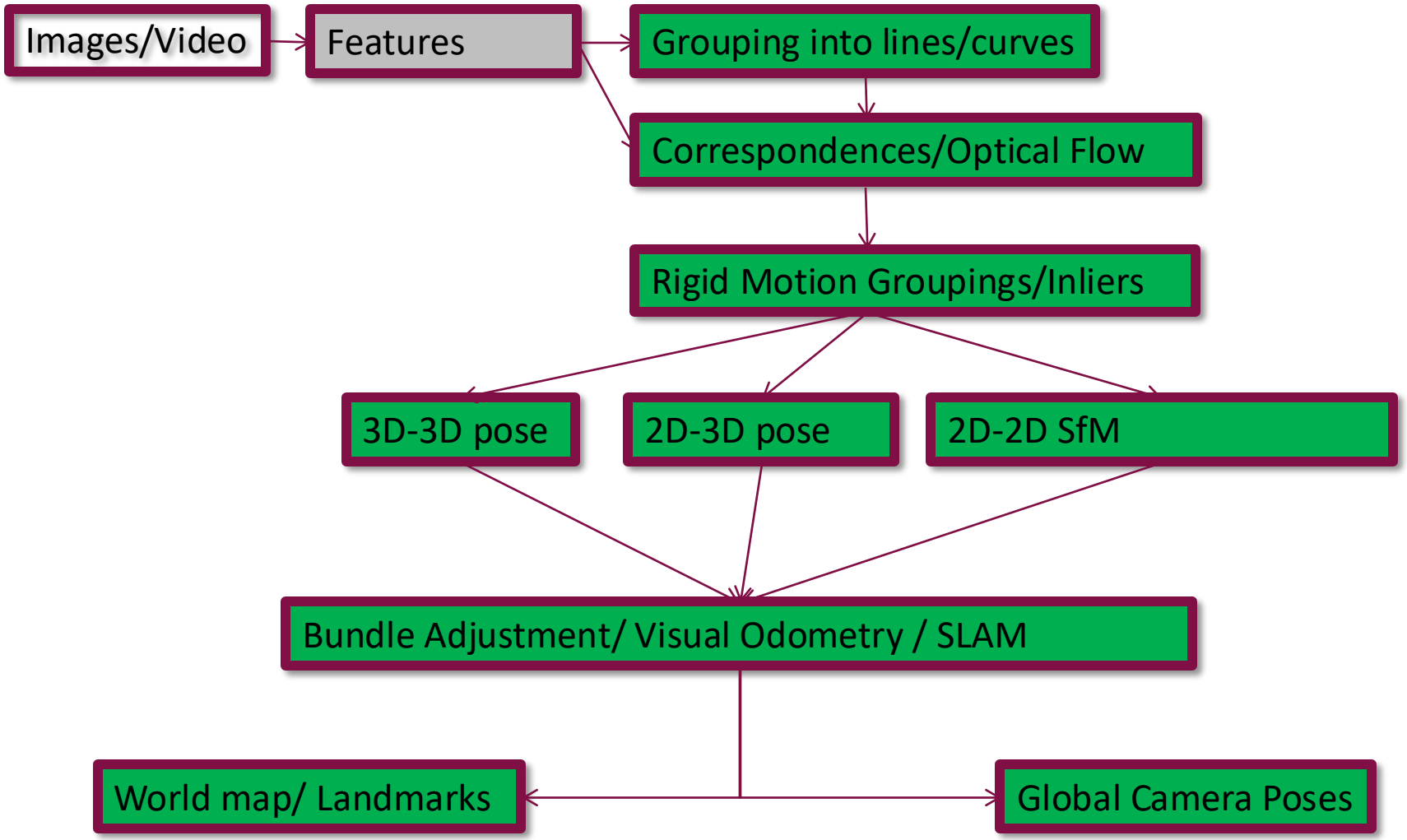
The Ground We've Covered: 2-View Geometry



The Ground We've Covered: Multi-View Geometry



Another view of what we've done



An example of a combined system (slide 1)

Input: a large collection of unordered images

1. First find SIFT feature correspondences among them (efficient methods)
2. Then, find overlapping image pairs (geometric verification using homographies / essential matrices)
3. Then run 2-view SfM using carefully selected initial views based on overlap. (lots of heuristics here)
4. Solve PnP to find extrinsics for new images, registering them to current 3D structure (good view selection may be critical here)
5. Using these newly registered images, triangulate to improve current 3D points, plus add new points into the 3D scene.
6. Run bundle adjustment, and potentially go back to step 4 to add other images..

Output: sparse 3D point cloud, and registered cameras

Based on ColMap