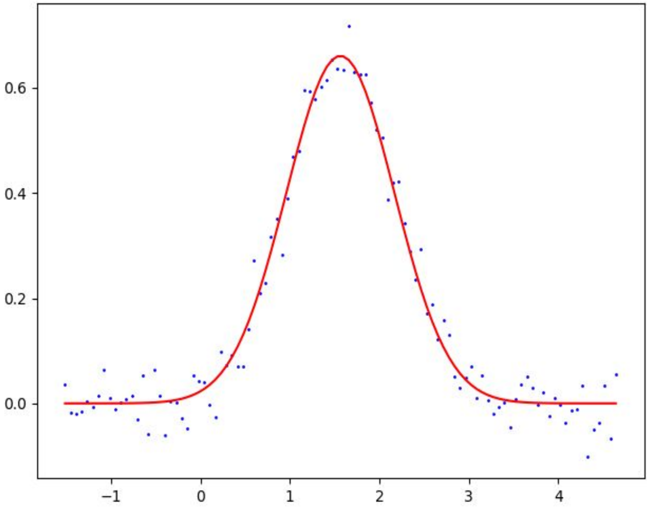
3D Scanning & Motion Capture

Exercise - 5

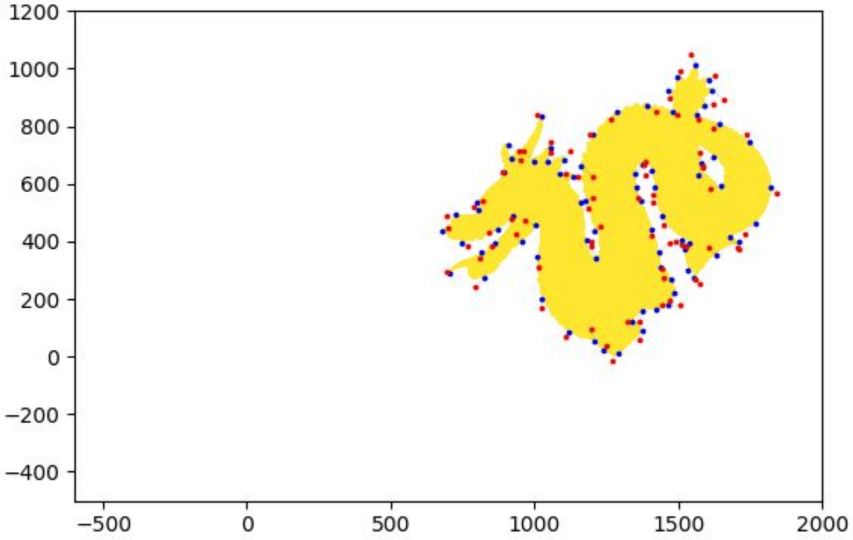
Felix Altenberger, Andrei Burov

Optimization - Solution



Gaussian

Hyperbolic Function



Registration

Exercises – Overview

1. Exercise $\longrightarrow$ Camera Intrinsics, Back-projection, Meshes

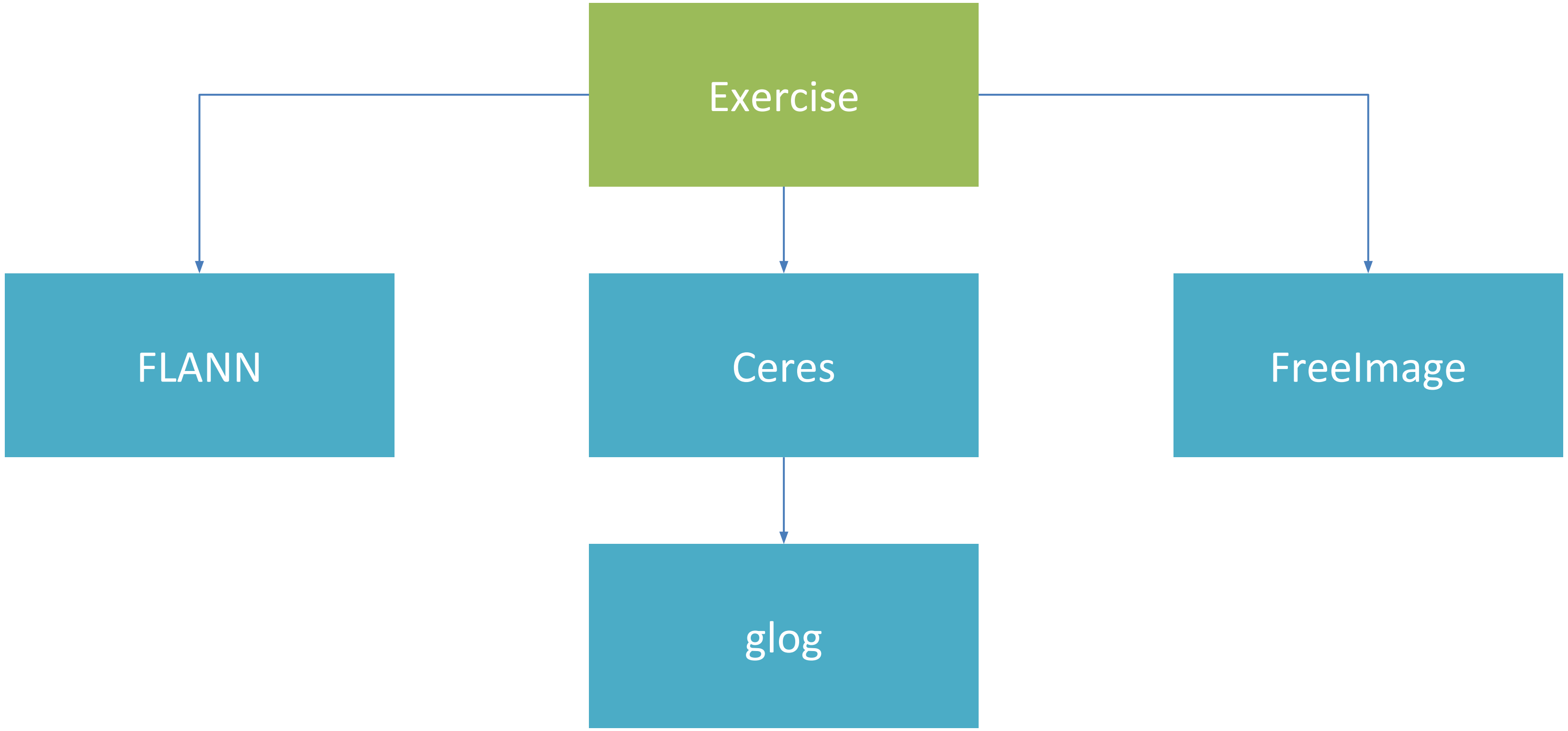
2. Exercise $\longrightarrow$ Surface Representations

3. Exercise $\longrightarrow$ Coarse Alignment (Procrustes)

4. Exercise $\longrightarrow$ Optimization

5. **Exercise $\longrightarrow$ Object Alignment, ICP**

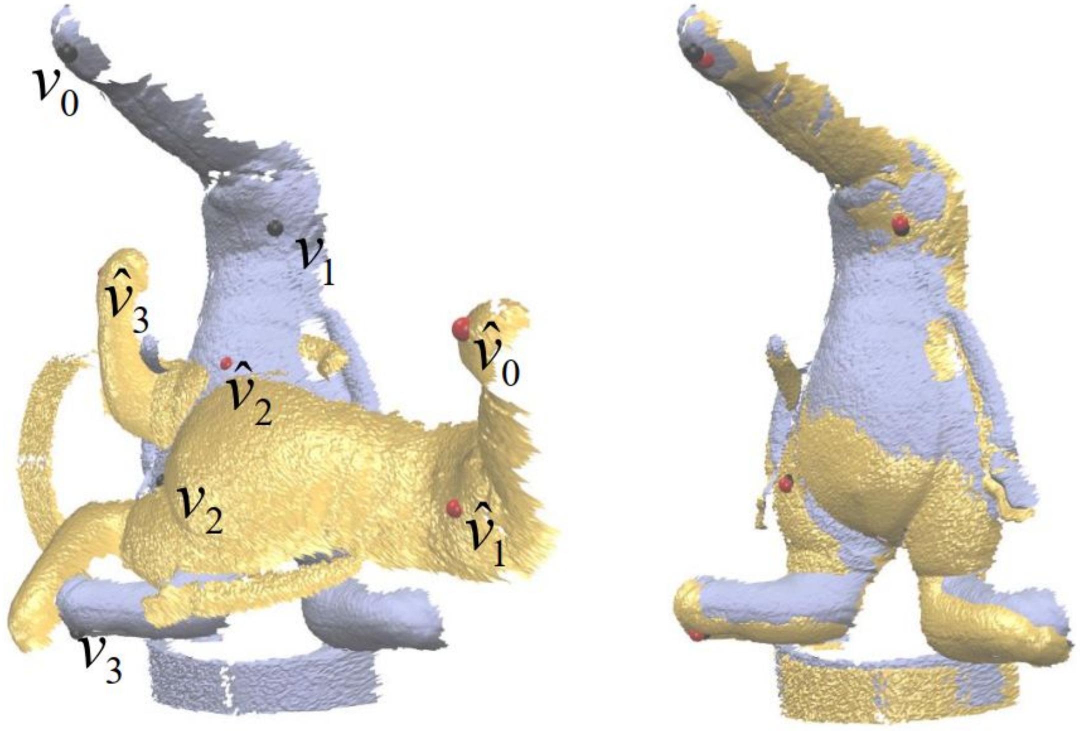
Project Dependencies



https://github.com/jlblancoc/nanoflann

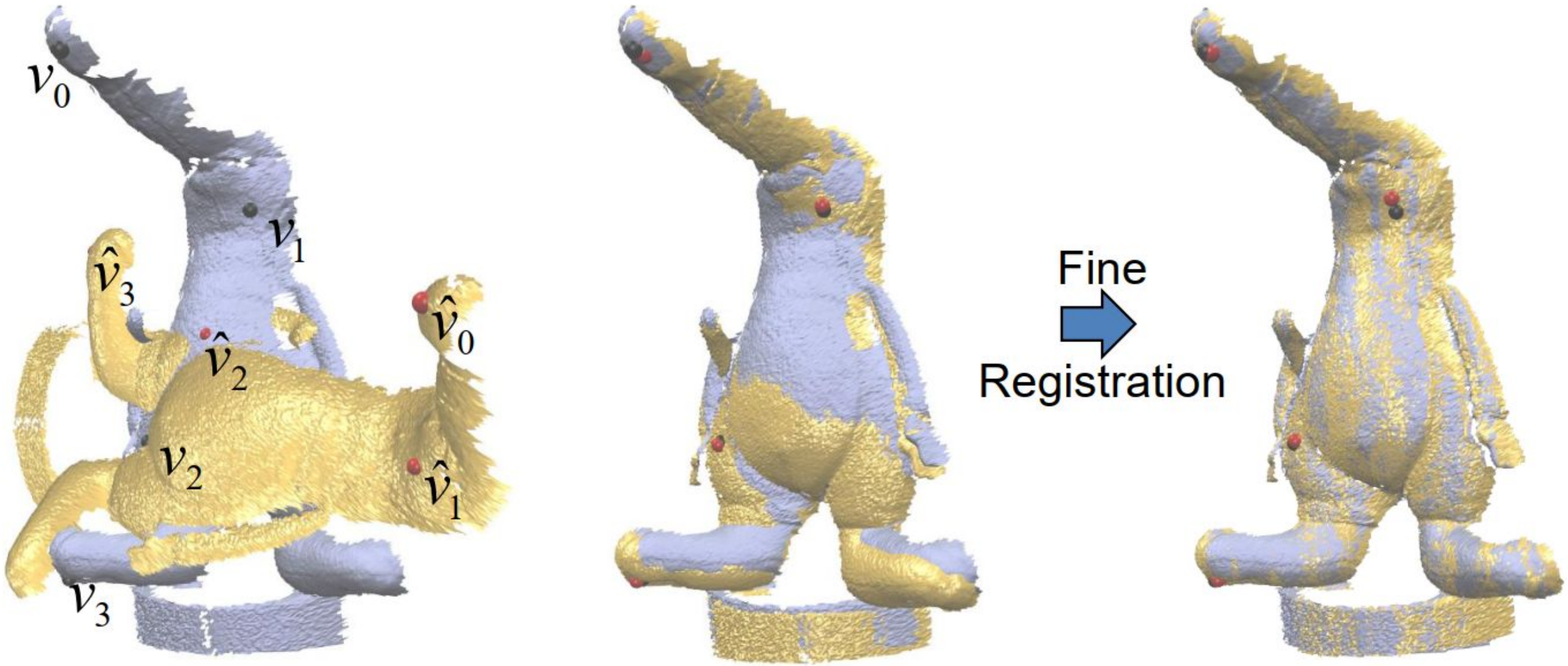
Previous Exercise – Coarse Alignment

- Given known correspondences, compute a transformation to the target shape
- Procrustes algorithm



Today – Fine Alignment

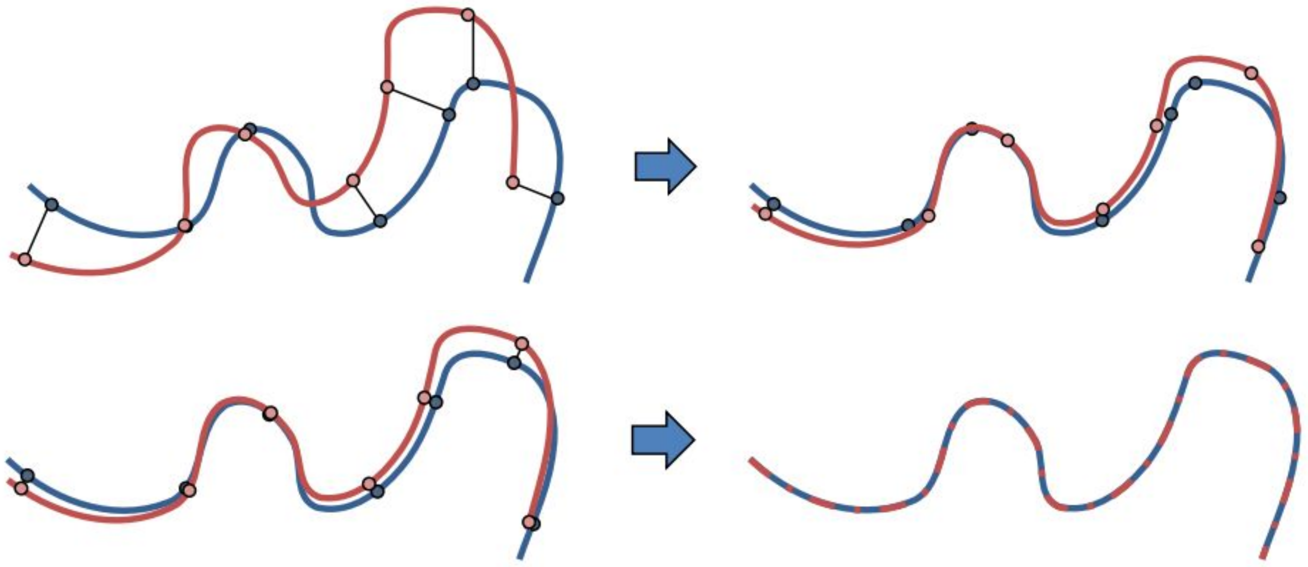
- Correspondences are not perfect → misalignment



ICP (Iterative Closest Point)

- Problem: Align two objects with unknown correspondences
  - Iterate:
    - Estimate correspondences using the current alignment and nearest neighbors
    - Use the correspondences to compute new alignment based on
      - Point-to-point distances
        - » Procrustes

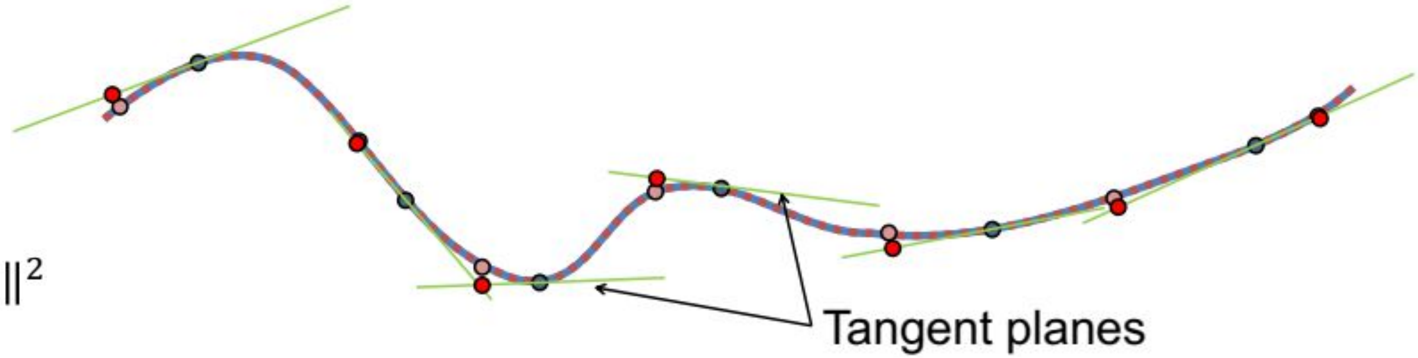
$$\min_{R,t} \sum_i \|p_i - (Rq_i + t)\|^2$$



ICP (Iterative Closest Point)

- Problem: Align two objects with unknown correspondences
  - Iterate:
    - Estimate correspondences using the current alignment and nearest neighbors
    - Use the correspondences to compute new alignment based on
      - Point-to-point distances
      - Point-to-plane distances
        - » Faster convergence
        - » Non-linear!

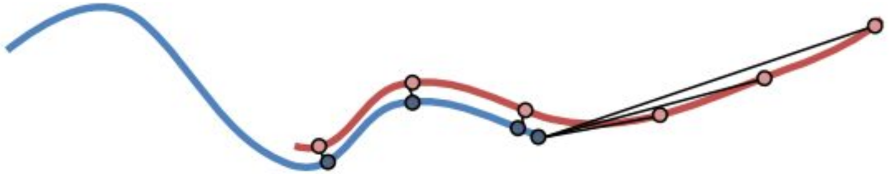
$$\min_{R,t} \sum_i \|(p_i - (Rq_i + t)) \cdot n_i\|^2$$



ICP (Iterative Closest Point)

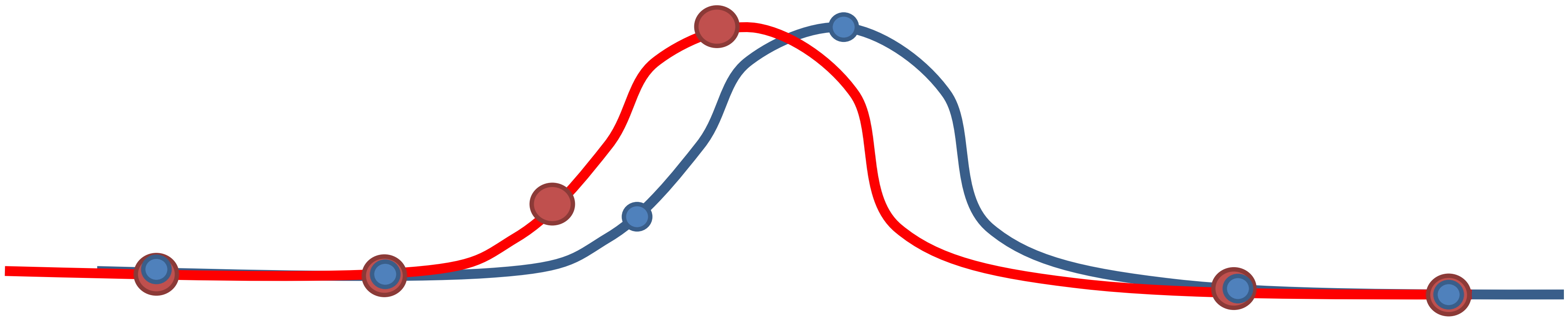
- Problem: Align two objects with unknown correspondences
  - Iterate:
    - Estimate correspondences using the current alignment and nearest neighbors
    - Use the correspondences to compute new alignment based on
      - Point-to-point distances
      - Point-to-plane distances
    - Use weighting of correspondences and pruning
      - Good correspondences are close, have similar normal, …
      - Prune correspondences to border

$$\min_{R,t} \sum_i w_i \|p_i - (Rq_i + t)\|^2$$



Point-to-point vs point-to-plane

- Point-to-point may converge slowly or not at all
- Point-to-plane allows one shape to “slide” over the other



Linearized ICP

- Point-to-plane distances make the problem non-linear. It is possible to linearize it:
  https://www.comp.nus.edu.sg/~lowkl/publications/lowk_point-to-plane_icp_techrep.pdf
- Both point-to-plane and point-to-point constraints can be included in a single system matrix. Point-to-point constraints must satisfy the following equation:

$$\begin{bmatrix} 1 & -\gamma & \beta & t_x \\ \gamma & 1 & -\alpha & t_y \\ -\beta & \alpha & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x \\ s_y \\ s_z \\ 1 \end{bmatrix} = \begin{bmatrix} d_x \\ d_y \\ d_z \\ 1 \end{bmatrix}$$

- where (alpha, beta, gamma) represents the rotation angles, *t* is the translation vector, *s* is the source point and *d* the destination point. Three constraints are to be added for every point (one for every coordinate of the point).

Recap – ICP

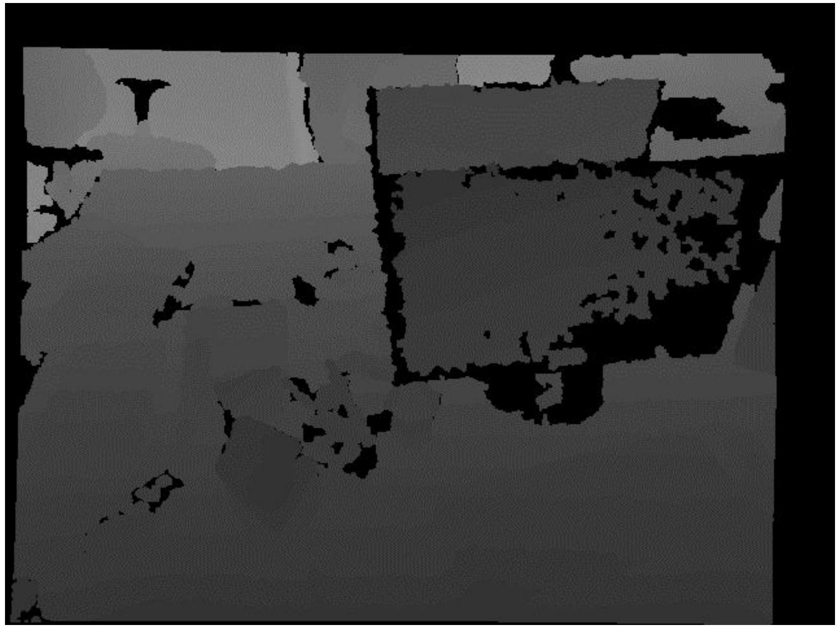
1. Select source points
2. Transform source points with current transformation estimate
3. Find matching target points (usually nearest neighbor)
4. Weigh the correspondences
5. Prune (reject) correspondences (outliers/border)
6. Compute optimal transformation with respect to chosen metric (point-to-point vs point-to-plane) and update the estimate
7. If not converged, repeat

Normal Computation

- Cross product between back-projected points
- Take object boundaries into account (e.g. via point distance)

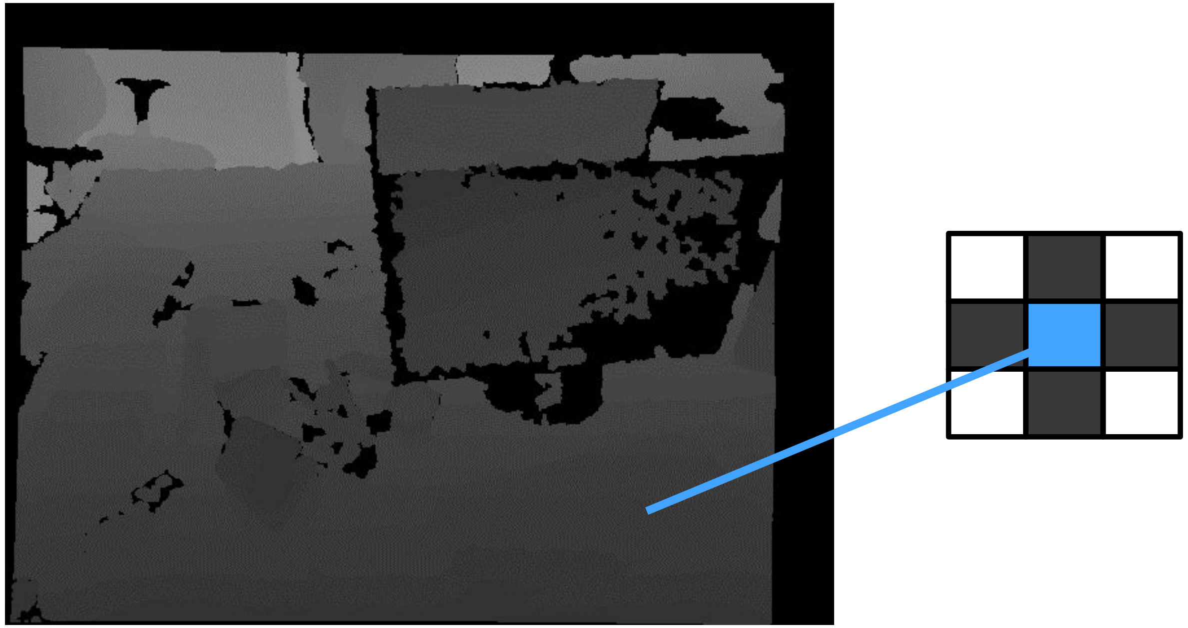
Normal Computation

- Cross product between back-projected points
- Take object boundaries into account (e.g. via point distance)



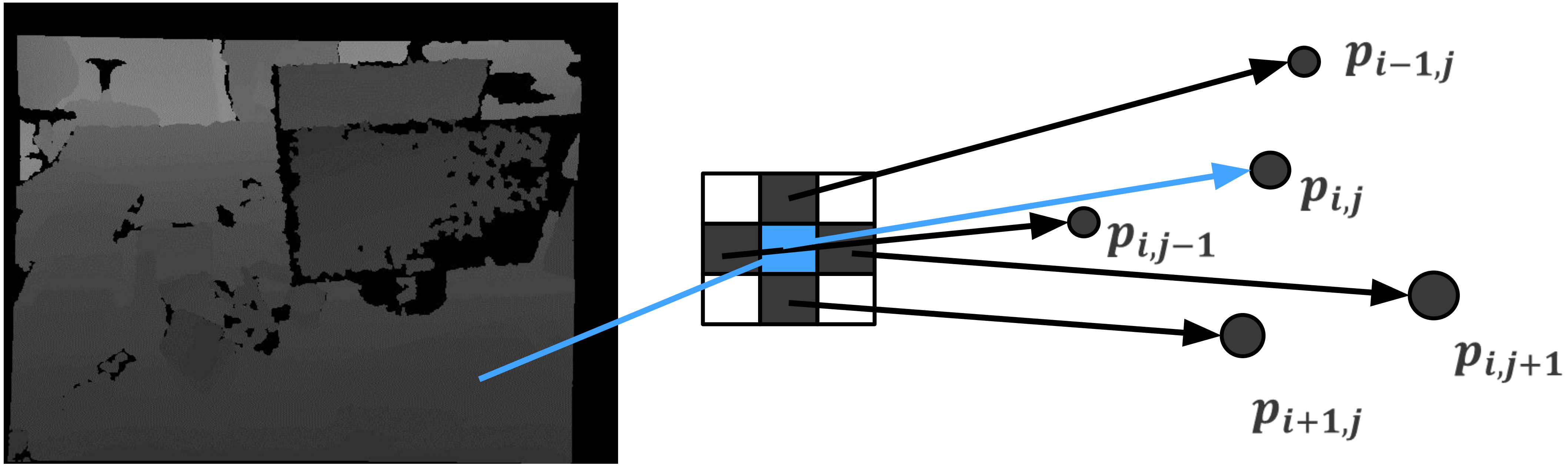
Normal Computation

- Cross product between back-projected points
- Take object boundaries into account (e.g. via point distance)



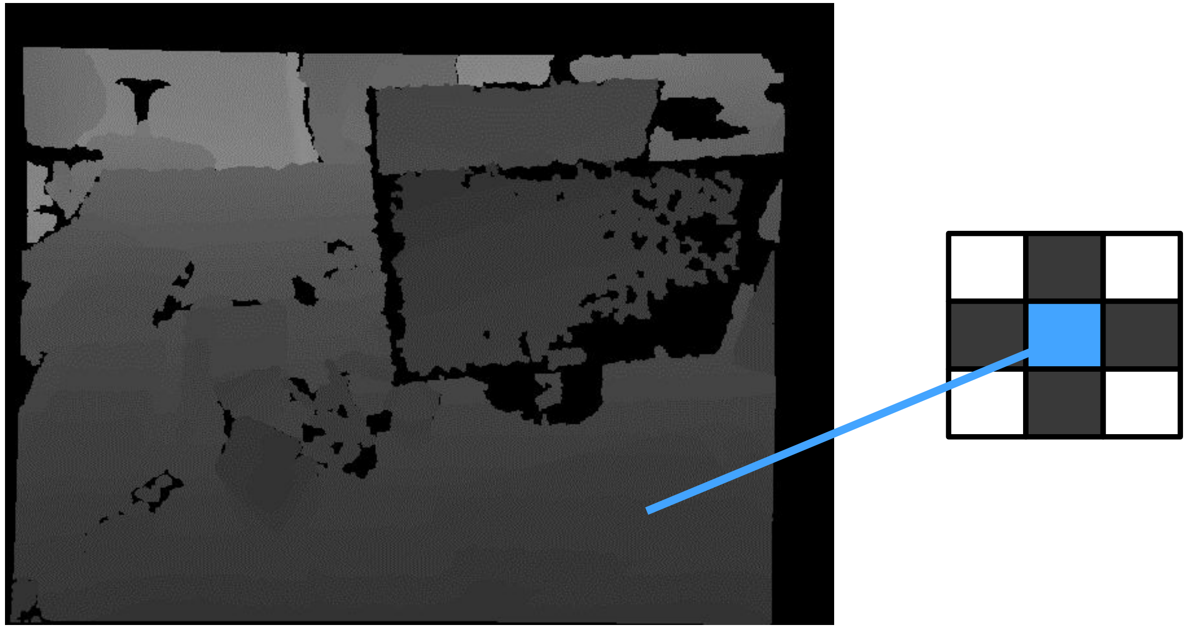
Normal Computation

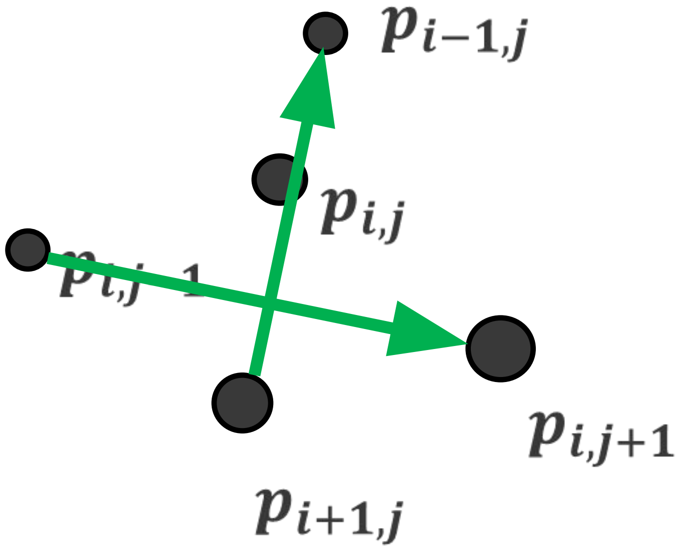
- Cross product between back-projected points
- Take object boundaries into account (e.g. via point distance)



Normal Computation

- Cross product between back-projected points
- Take object boundaries into account (e.g. via point distance)

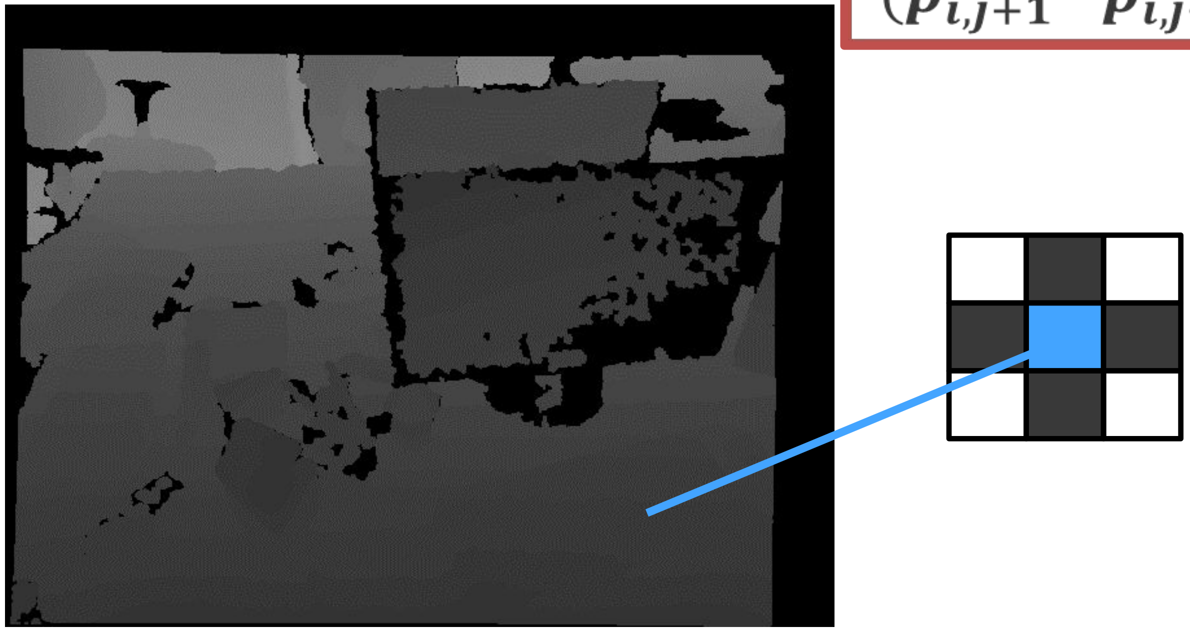


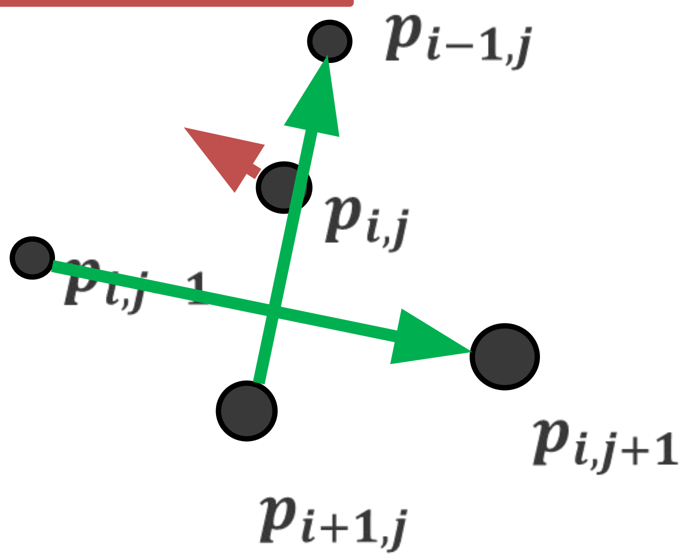


Normal Computation

- Cross product between back-projected points
- Take object boundaries into account (e.g. via point distance)

$$\boxed{(p_{i,j+1} - p_{i,j-1}) \times (p_{i-1,j} - p_{i+1,j})}$$





Other approaches for normal computation

- Based on differences in depth of neighboring pixels
  - Don’t forget to take field of view into account
- Principal Component Analysis (PCA):
  - Search for points in the neighbourhood
  - Compute principal components
  - The normal will be the smallest principal component

See you next time!