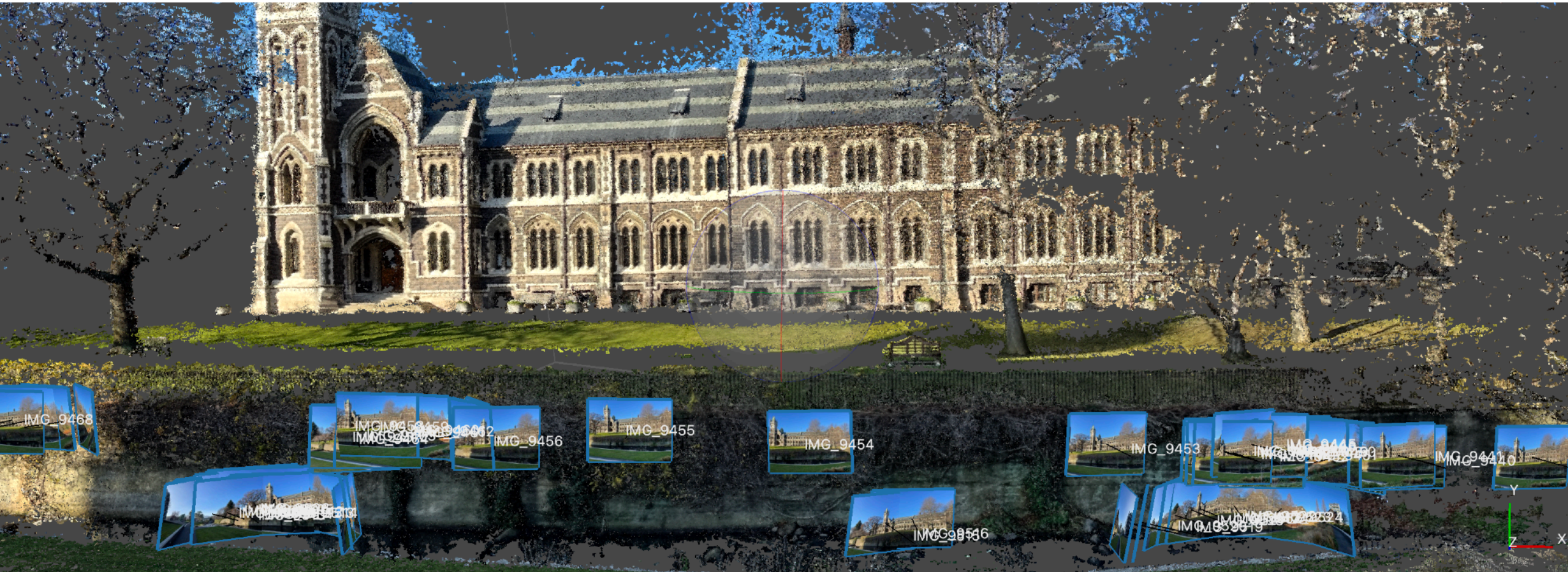
Visual Computing I:

Interactive Computer Graphics and Vision



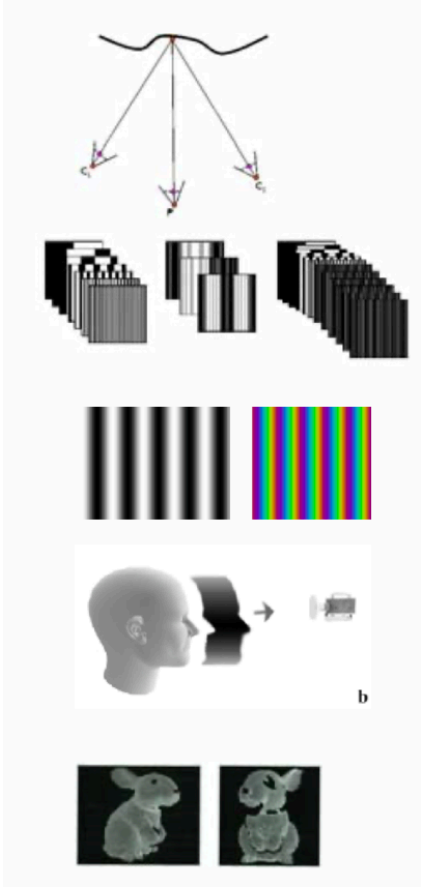
Depth Maps Processing, 3D Reconstruction

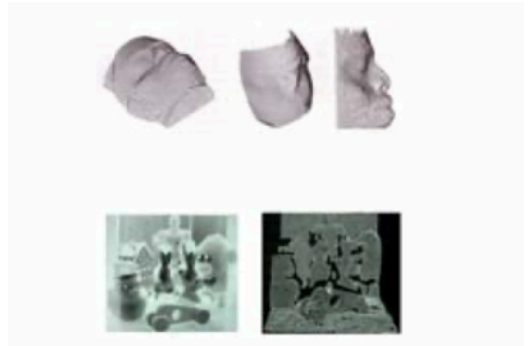
Stefanie Zollmann and Tobias Langlotz

Last time..

Overview

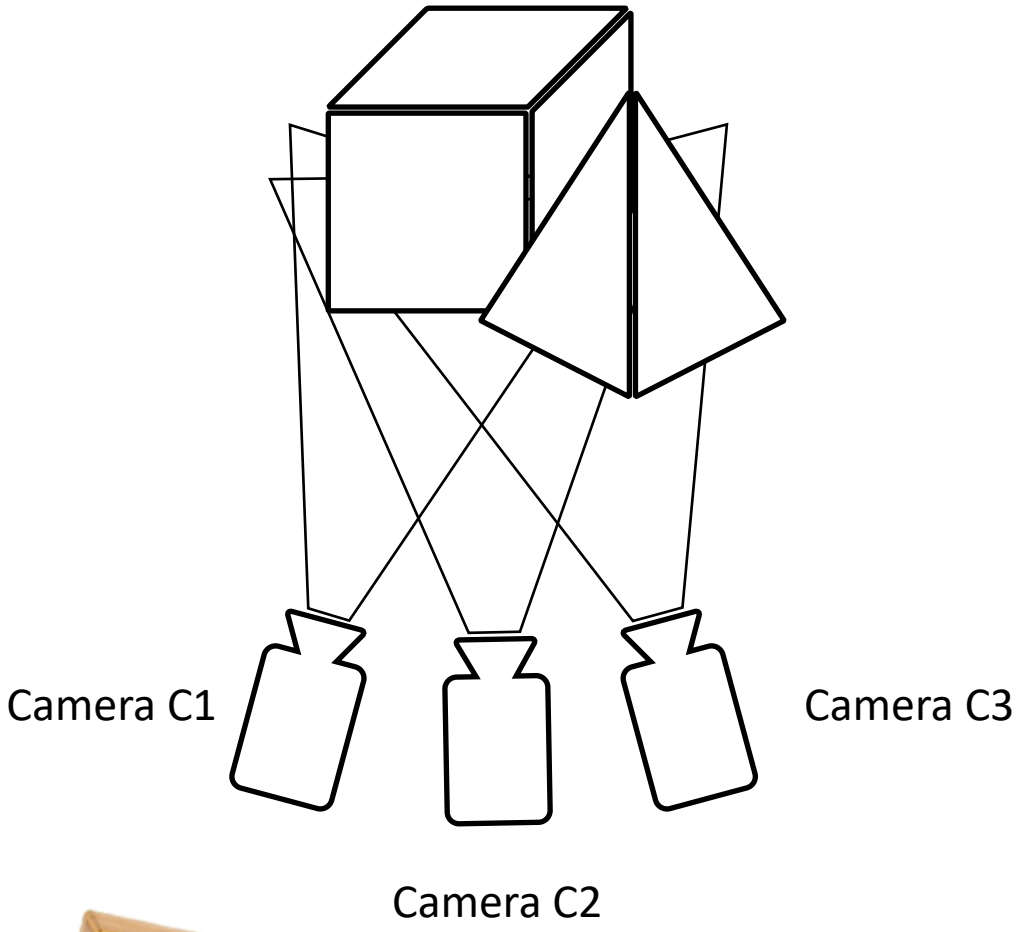
- Range Scanning
  - Passive:
    - Stereo or Multi-Camera
    - Active:
    - Structured Light Projection
    - Optical Time-of-Flight
      - Direct ToF
      - Indirect ToF
- Registration of Depth Maps
- Processing Depth Maps





Passive Range Scanning

- Two or more cameras
  - Stereo camera
  - Two cameras
  - One camera takes several photos from different perspectives
  - Multiple cameras
- Which one is the easiest and why?
- What is the main challenge?

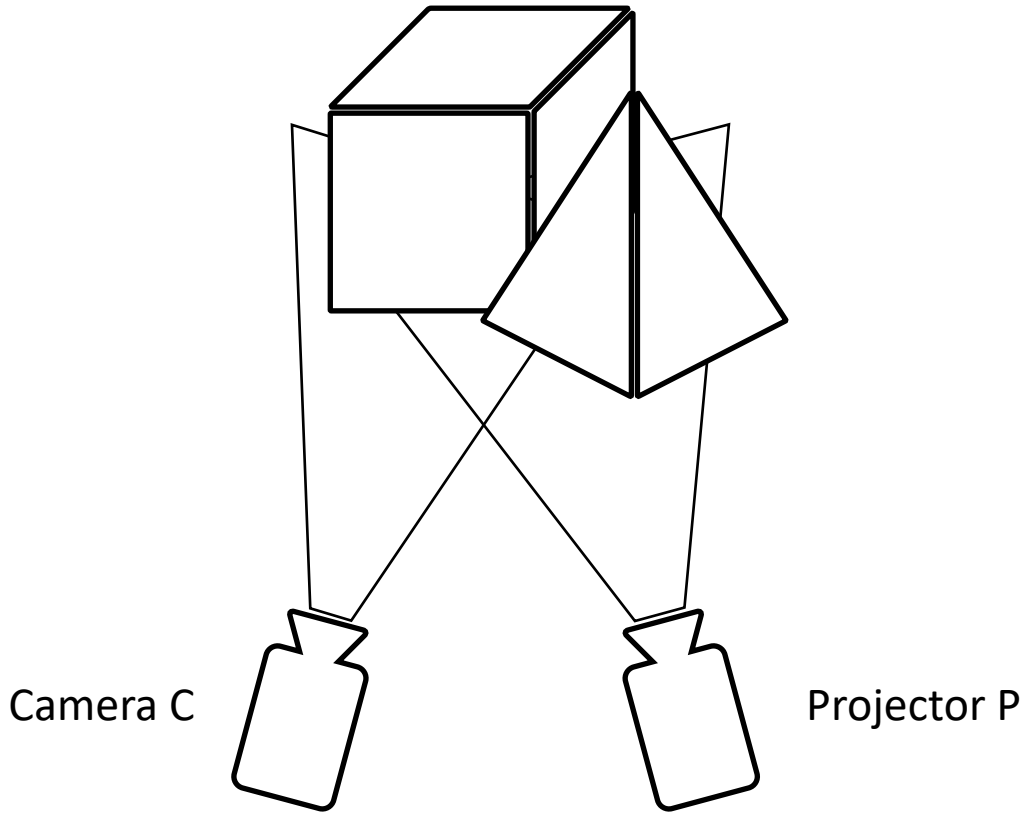


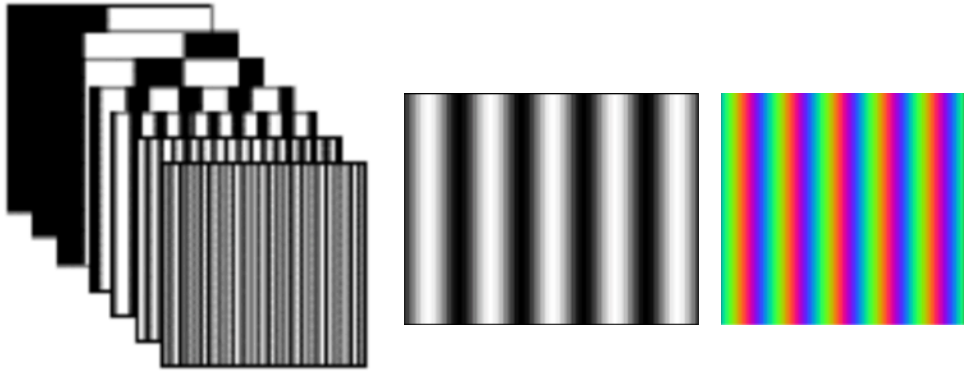
PointGrey Bumble Bee

Zed Mini

Structured Light Projection

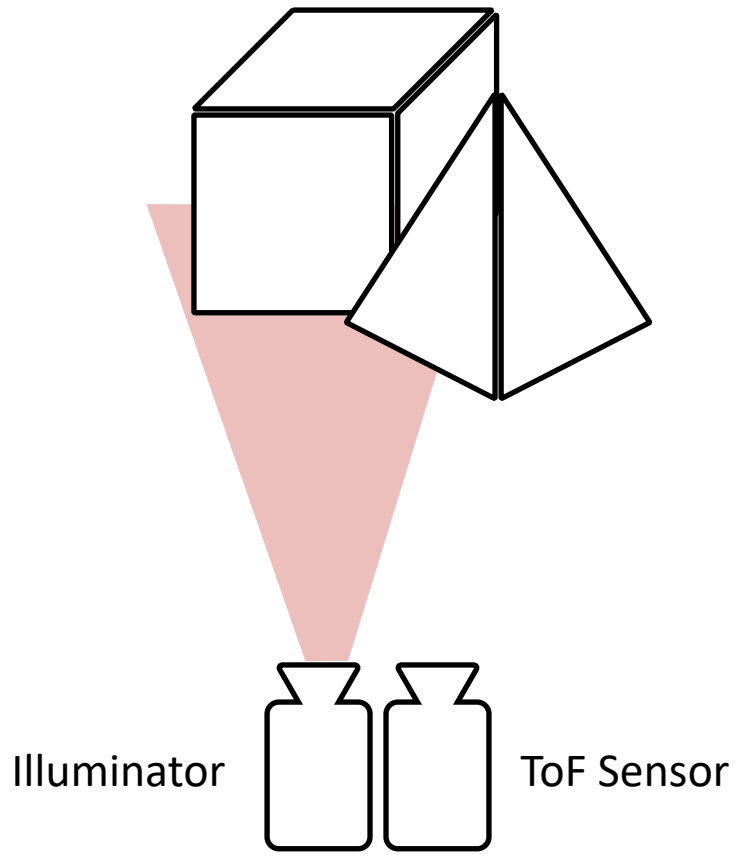
- Many structured light techniques exist that follow the two objectives:
  - Determine projector-camera correspondences with as few images as possible
  - Be robust against surface modulation and noise
- Codes include:
  - Binary codes, such as the Gray code (most common): <100 images, quite robust
  - Intensity codes that apply cosine patterns and phase shifts: 6 images, but are not robust against strong absorptions
  - Color coding: <3 images, but very fragile (does not work on coloured surfaces)





Optical Time-of-Flight (ToF)

- General idea:
  - Compute depth by round-trip estimation of a light wave emitted and its reflection back to the sensor
  - $D = C \cdot \Delta t/2$
- Use low-power IR illumination for pulsed illumination
  - Can be switched on/off with <1ns
  - Generates a “light wall” that is reflected back by objects



Depth from Mono / Depth Estimation using ML

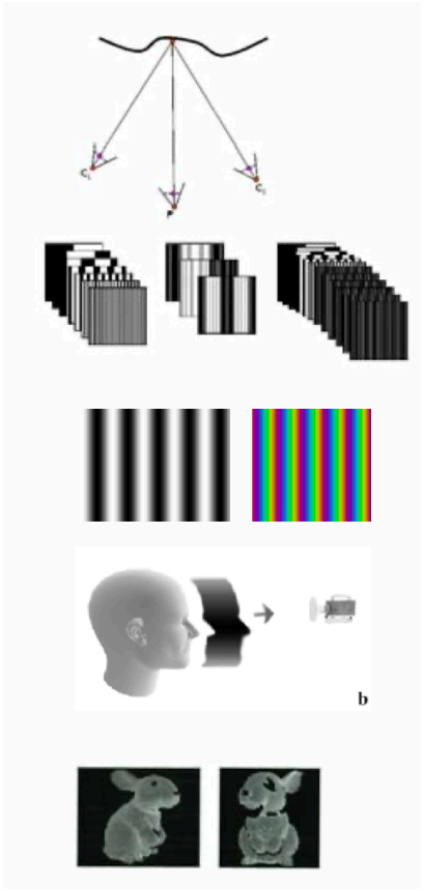
- Convolutional neural networks (CNNs) have been applied to monocular depth estimation
- Common architectures include U-Net, ResNet, and DenseNet
- During training:
  - Network is given pairs of images and their corresponding ground-truth depth maps
  - Minimize the difference between predicted depth and ground truth depth

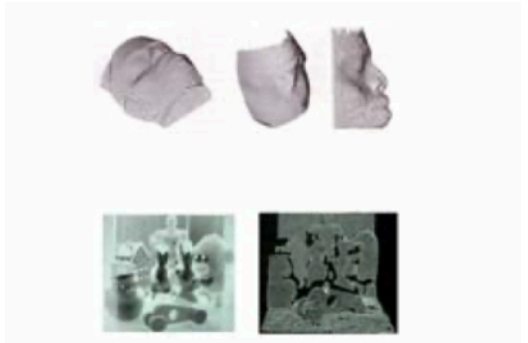
https://github.com/yuhsuanyeh/BiFuse

Today:
Processing of Depth Maps, Relative Pose and Multi-view Stereo

Overview

- Range Scanning
  - Passive:
    - Stereo or Multi-Camera
    - Active:
    - Structured Light Projection
    - Optical Time-of-Flight
      - Direct ToF
      - Indirect ToF
- Registration of Depth Maps
- Processing Depth Maps





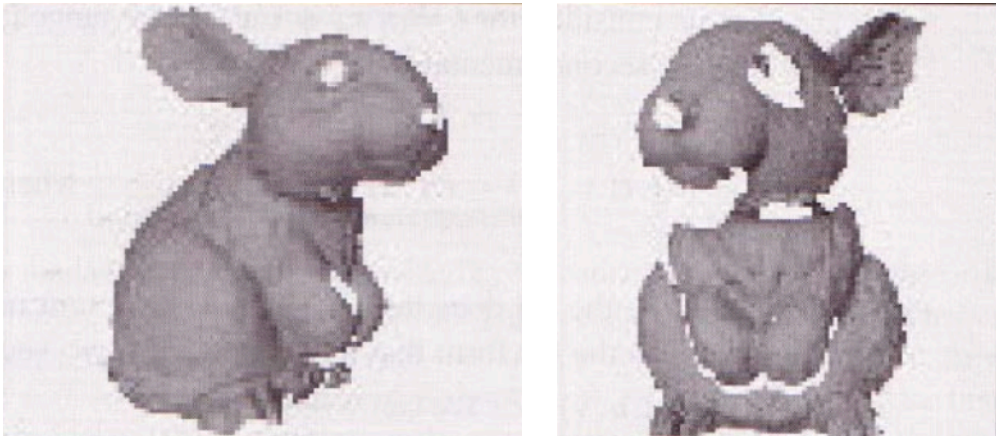
Registration of Depth Maps

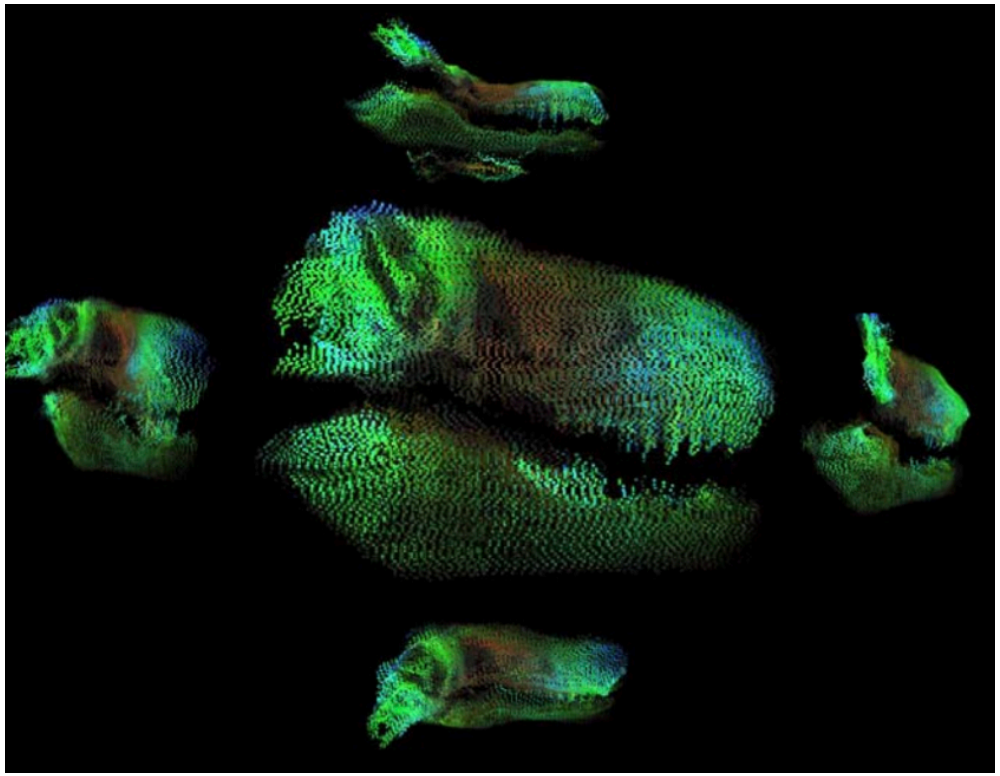
Registration of Depth Maps

- Depth map gives us distance for each pixel, can compute 3D points:

$$\mathbf{X}_{\text{camera}} = D(u, v) \cdot K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

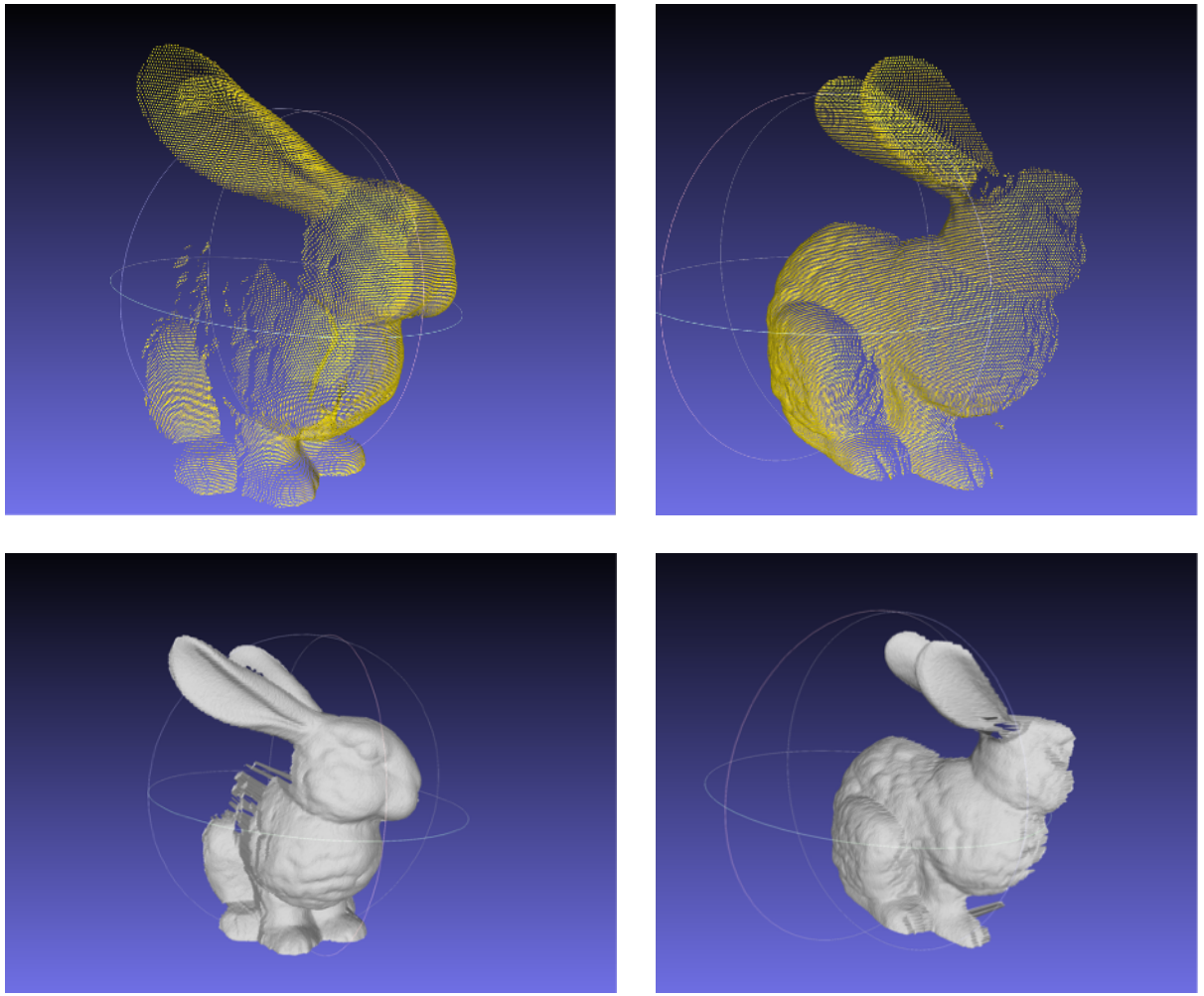
- Having multiple depth maps of the same scene, how can they be merged?
- This process is known as registration
- If camera poses are unknown, we need to align the 3D points purely based on geometry





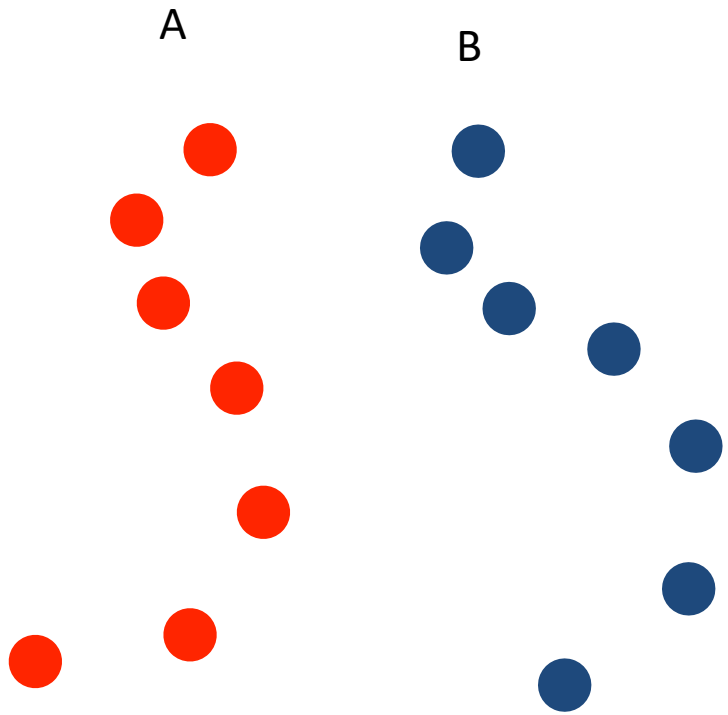
Iterative Closest-Point Method

- Registration of two sets of three-dimensional points requires the computation of a rigid transformation that maps the first point set (B) to the second one (A)
- Iteratively minimise the average distance between the two point sets:
  - Find correspondences between the sets by matching every point in B with the point closest to it in A
  - Compute rigid transformation that maps B to A and apply it to B (B’)
  - Compute average distance E between A and B’
  - Repeat until E is minimal



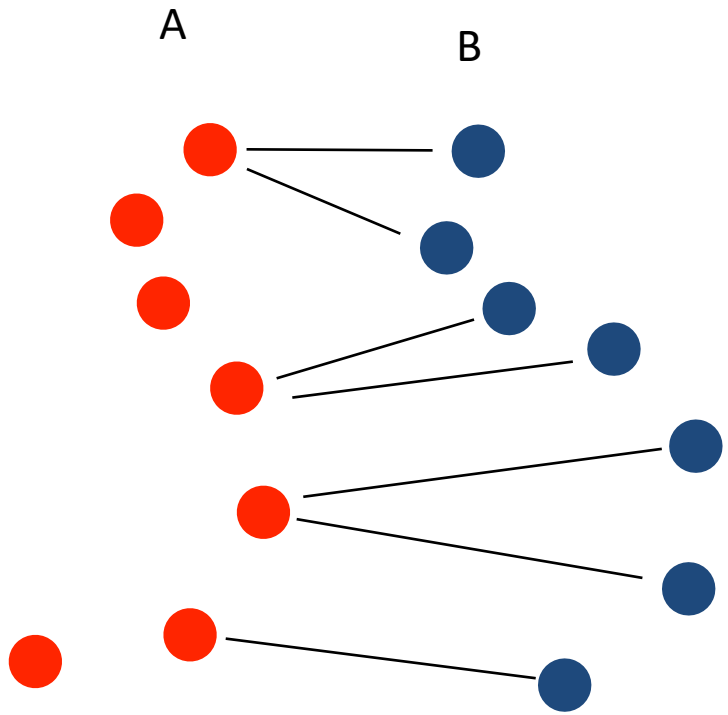
Iterative Closest-Point Method

- Registration of two sets of three-dimensional points requires the computation of a rigid transformation that maps the first point set (B) to the second one (A)
- Iteratively minimise the average distance between the two point sets:
  - Find correspondences between the sets by matching every point in B with the point closest to it in A
  - Compute rigid transformation that maps B to A and apply it to B (B’)
  - Compute average distance E between A and B’
  - Repeat until E is minimal



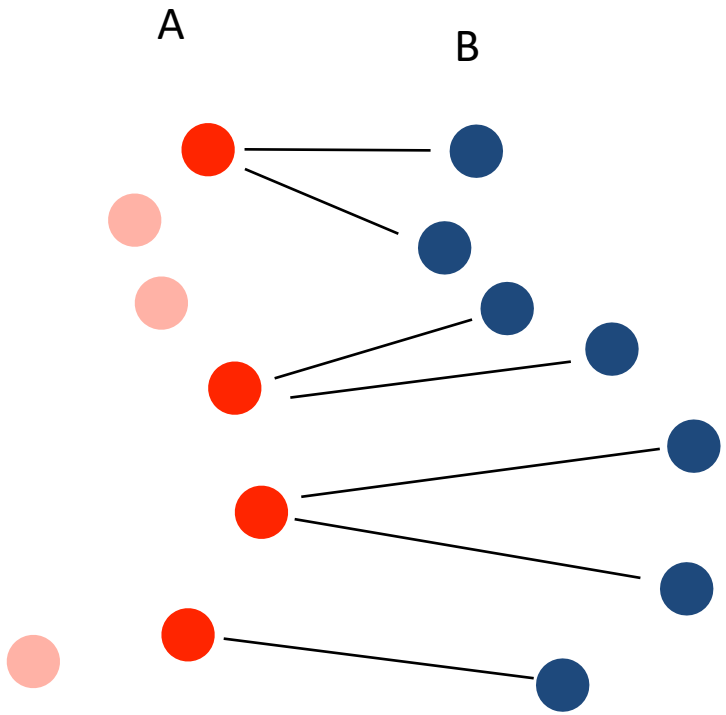
Iterative Closest-Point Method

- Registration of two sets of three-dimensional points requires the computation of a rigid transformation that maps the first point set (B) to the second one (A)
- Iteratively minimise the average distance between the two point sets:
  - Find correspondences between the sets by matching every point in B with the point closest to it in A
  - Compute rigid transformation that maps B to A and apply it to B (B’)
  - Compute average distance E between A and B’
  - Repeat until E is minimal



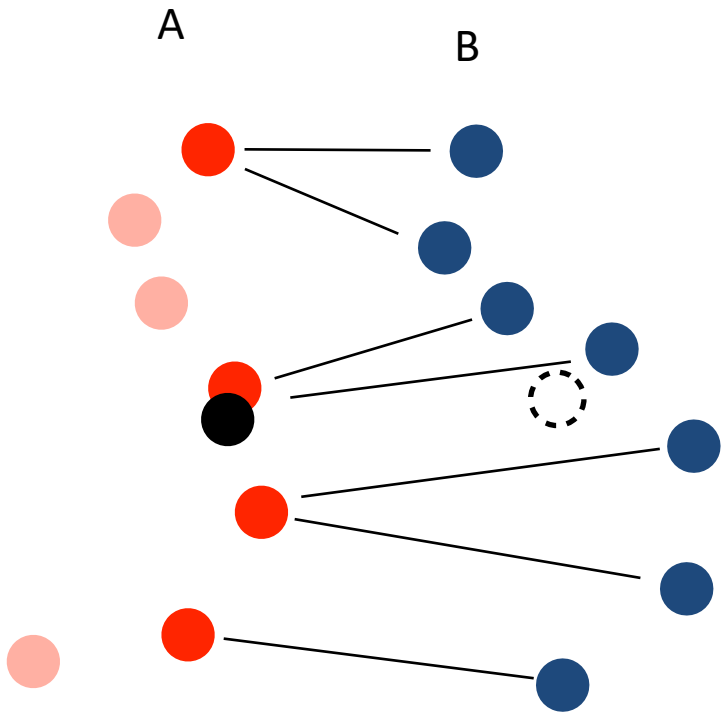
Iterative Closest-Point Method

- Registration of two sets of three-dimensional points requires the computation of a rigid transformation that maps the first point set (B) to the second one (A)
- Iteratively minimise the average distance between the two point sets:
  - Find correspondences between the sets by matching every point in B with the point closest to it in A
  - Compute rigid transformation that maps B to A and apply it to B (B’)
  - Compute average distance E between A and B’
  - Repeat until E is minimal



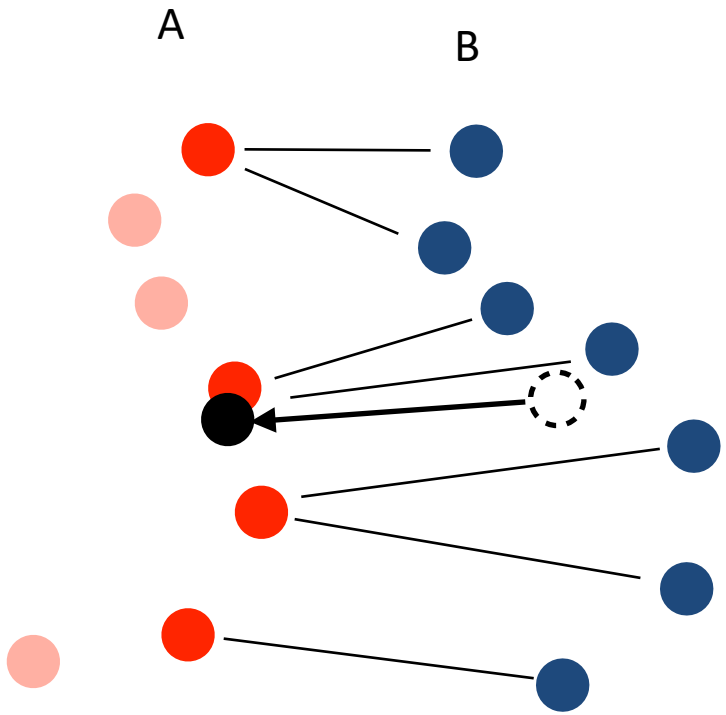
Iterative Closest-Point Method

- Registration of two sets of three-dimensional points requires the computation of a rigid transformation that maps the first point set (B) to the second one (A)
- Iteratively minimise the average distance between the two point sets:
  - Find correspondences between the sets by matching every point in B with the point closest to it in A
  - Compute rigid transformation that maps B to A and apply it to B (B’)
  - Compute average distance E between A and B’
  - Repeat until E is minimal



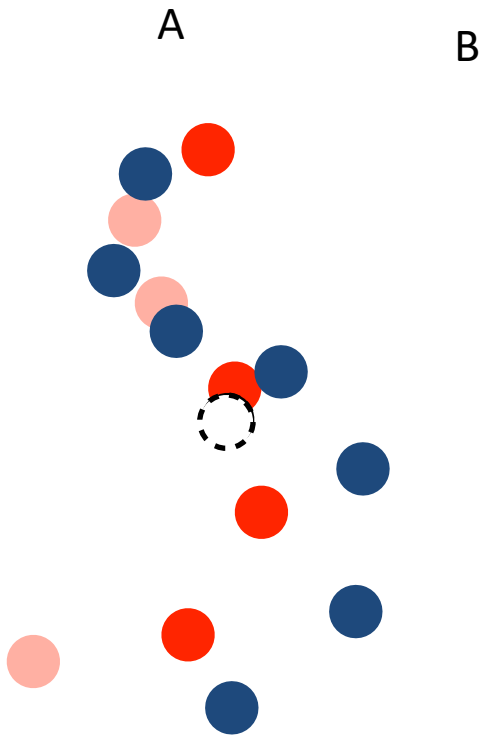
Iterative Closest-Point Method

- Registration of two sets of three-dimensional points requires the computation of a rigid transformation that maps the first point set (B) to the second one (A)
- Iteratively minimise the average distance between the two point sets:
  - Find correspondences between the sets by matching every point in B with the point closest to it in A
  - Compute rigid transformation that maps B to A and apply it to B (B’)
  - Compute average distance E between A and B’
  - Repeat until E is minimal



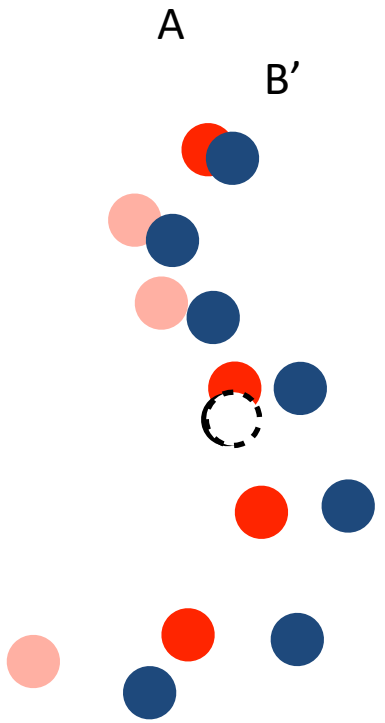
Iterative Closest-Point Method

- Registration of two sets of three-dimensional points requires the computation of a rigid transformation that maps the first point set (B) to the second one (A)
- Iteratively minimise the average distance between the two point sets:
  - Find correspondences between the sets by matching every point in B with the point closest to it in A
  - Compute rigid transformation that maps B to A and apply it to B (B’)
  - Compute average distance E between A and B’
  - Repeat until E is minimal



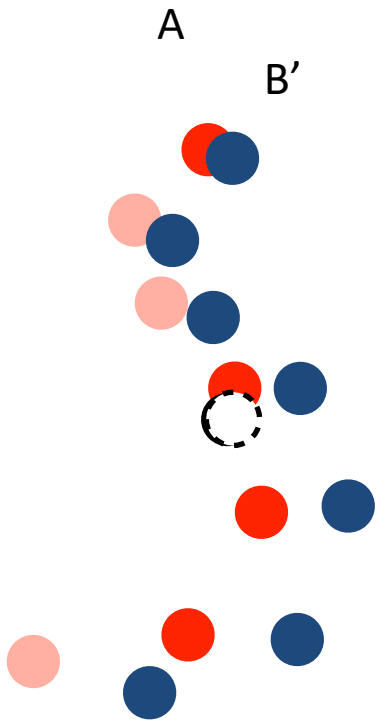
Iterative Closest-Point Method

- Registration of two sets of three-dimensional points requires the computation of a rigid transformation that maps the first point set (B) to the second one (A)
- Iteratively minimise the average distance between the two point sets:
  - Find correspondences between the sets by matching every point in B with the point closest to it in A
  - Compute rigid transformation that maps B to A and apply it to B (B’)
  - Compute average distance E between A and B’
  - Repeat until E is minimal



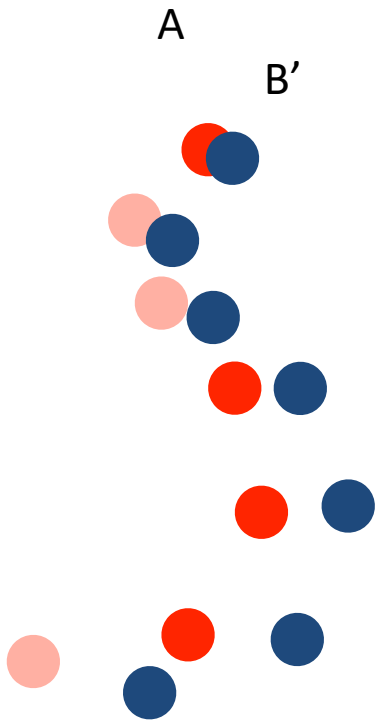
Iterative Closest-Point Method

- Registration of two sets of three-dimensional points requires the computation of a rigid transformation that maps the first point set (B) to the second one (A)
- Iteratively minimise the average distance between the two point sets:
  - Find correspondences between the sets by matching every point in B with the point closest to it in A
  - Compute rigid transformation that maps B to A and apply it to B (B’)
  - Compute average distance E between A and B’
  - Repeat until E is minimal



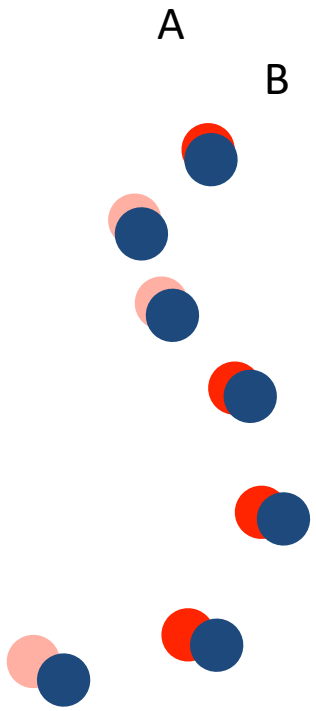
Iterative Closest-Point Method

- Registration of two sets of three-dimensional points requires the computation of a rigid transformation that maps the first point set (B) to the second one (A)
- Iteratively minimise the average distance between the two point sets:
  - Find correspondences between the sets by matching every point in B with the point closest to it in A
  - Compute rigid transformation that maps B to A and apply it to B (B’)
  - Compute average distance E between A and B’
  - Repeat until E is minimal



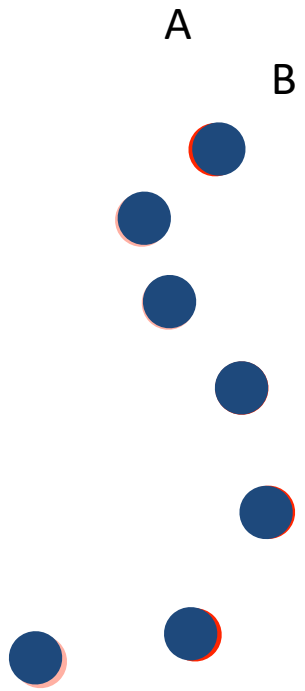
Iterative Closest-Point Method

- Registration of two sets of three-dimensional points requires the computation of a rigid transformation that maps the first point set (B) to the second one (A)
- Iteratively minimise the average distance between the two point sets:
  - Find correspondences between the sets by matching every point in B with the point closest to it in A
  - Compute rigid transformation that maps B to A and apply it to B (B’)
  - Compute average distance E between A and B’
  - Repeat until E is minimal



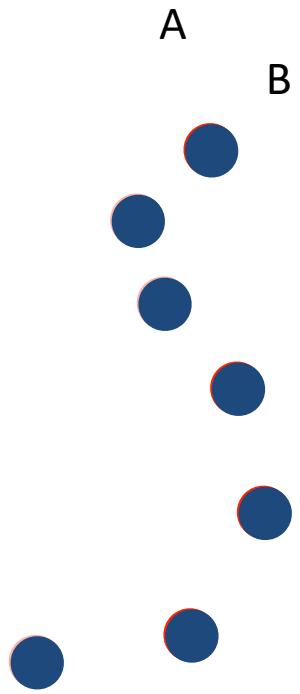
Iterative Closest-Point Method

- Registration of two sets of three-dimensional points requires the computation of a rigid transformation that maps the first point set (B) to the second one (A)
- Iteratively minimise the average distance between the two point sets:
  - Find correspondences between the sets by matching every point in B with the point closest to it in A
  - Compute rigid transformation that maps B to A and apply it to B (B’)
  - Compute average distance E between A and B’
  - Repeat until E is minimal



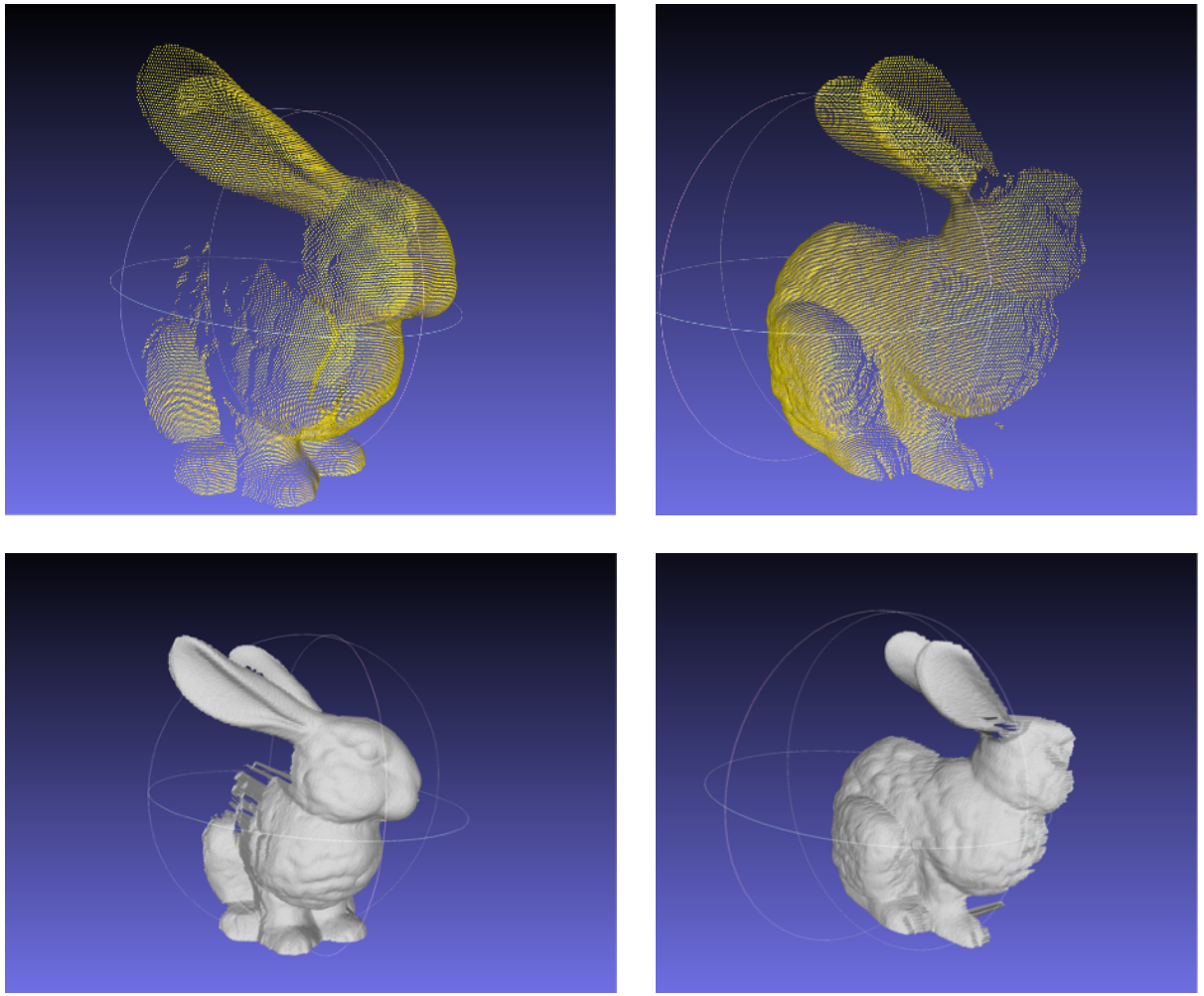
Iterative Closest-Point Method

- Registration of two sets of three-dimensional points requires the computation of a rigid transformation that maps the first point set (B) to the second one (A)
- Iteratively minimise the average distance between the two point sets:
  - Find correspondences between the sets by matching every point in B with the point closest to it in A
  - Compute rigid transformation that maps B to A and apply it to B (B’)
  - Compute average distance E between A and B’
  - Repeat until E is minimal



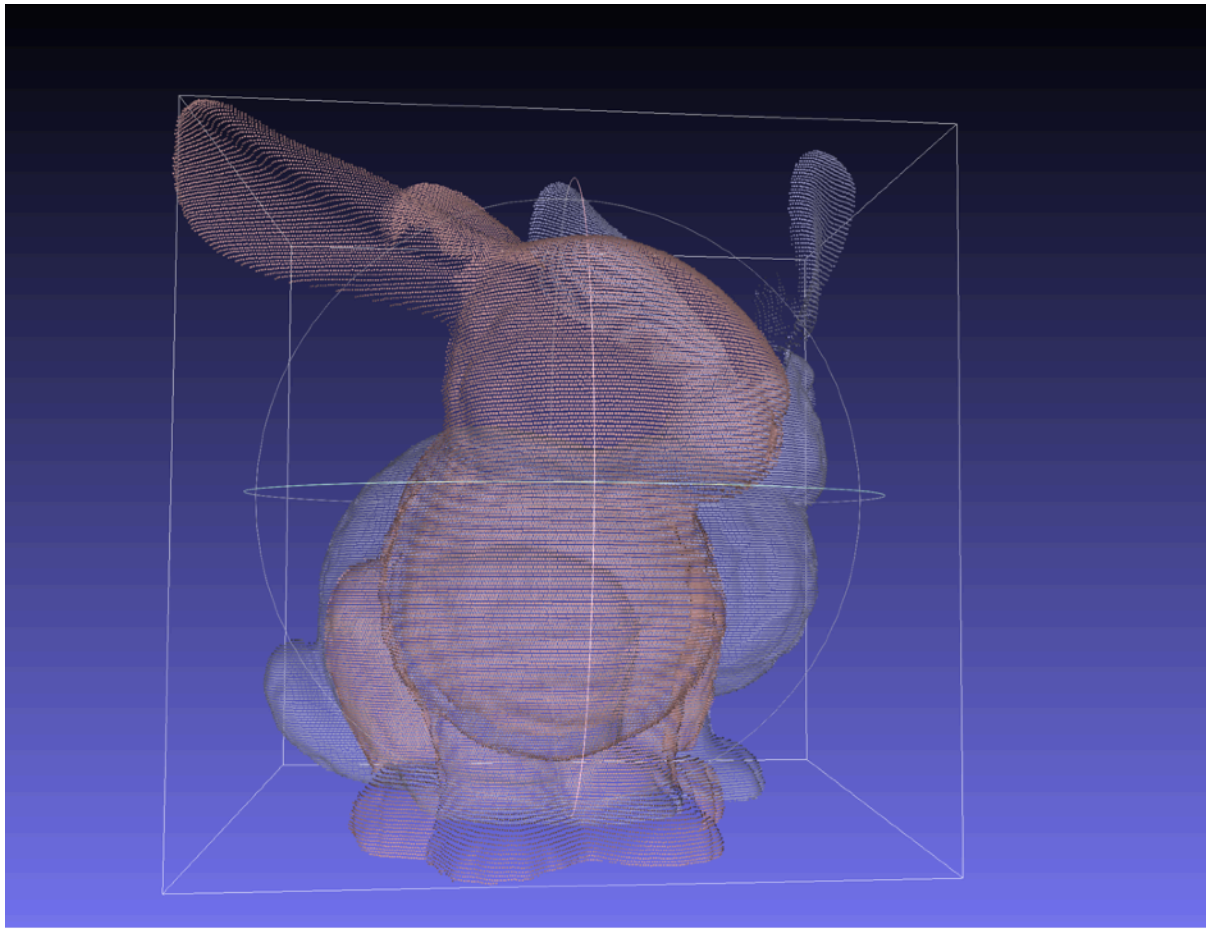
Iterative Closest-Point Method

- Registration of two sets of three-dimensional points requires the computation of a rigid transformation that maps the first point set (B) to the second one (A)
- Iteratively minimise the average distance between the two point sets:
  - Find correspondences between the sets by matching every point in B with the point closest to it in A
  - Compute rigid transformation that maps B to A and apply it to B (B’)
  - Compute average distance E between A and B’
  - Repeat until E is minimal



Iterative Closest-Point Method

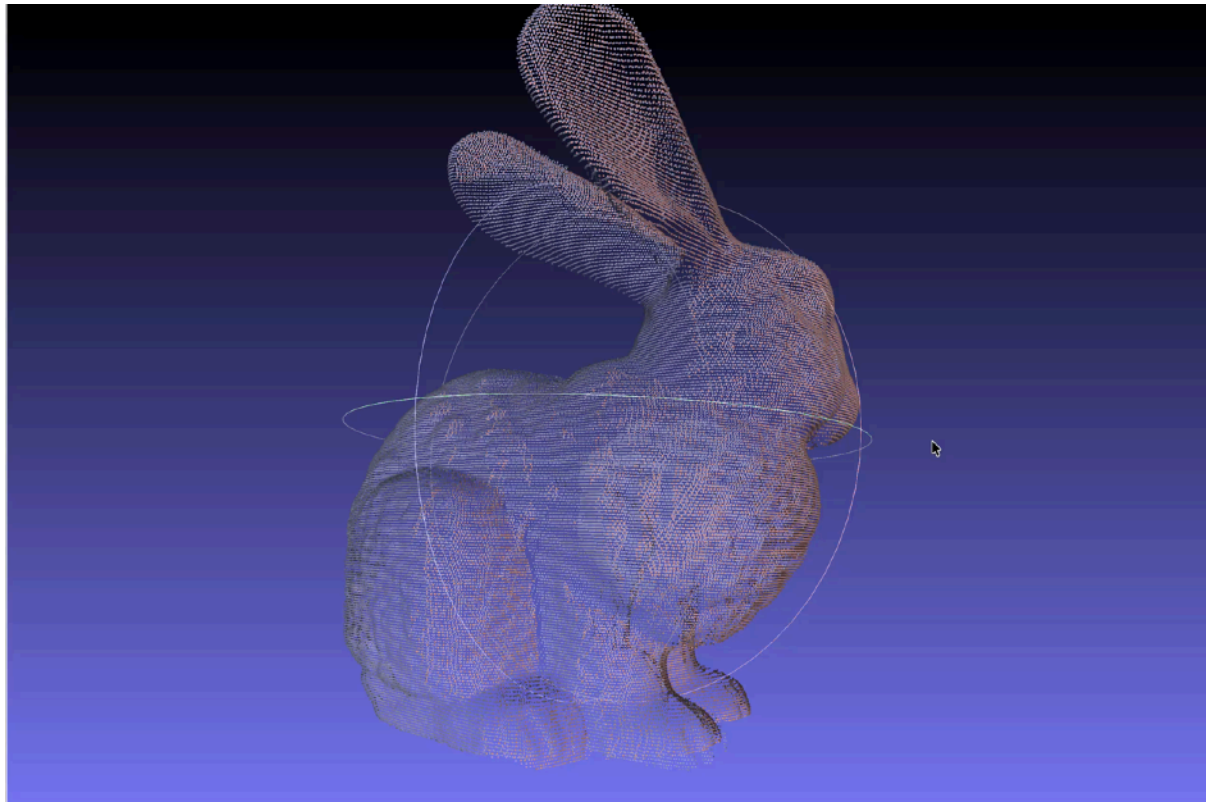
- Registration of two sets of three-dimensional points requires the computation of a rigid transformation that maps the first point set (B) to the second one (A)
- Iteratively minimise the average distance between the two point sets:
  - Find correspondences between the sets by matching every point in B with the point closest to it in A
  - Compute rigid transformation that maps B to A and apply it to B (B’)
  - Compute average distance E between A and B’
  - Repeat until E is minimal



ICP in Meshlab

Iterative Closest-Point Method

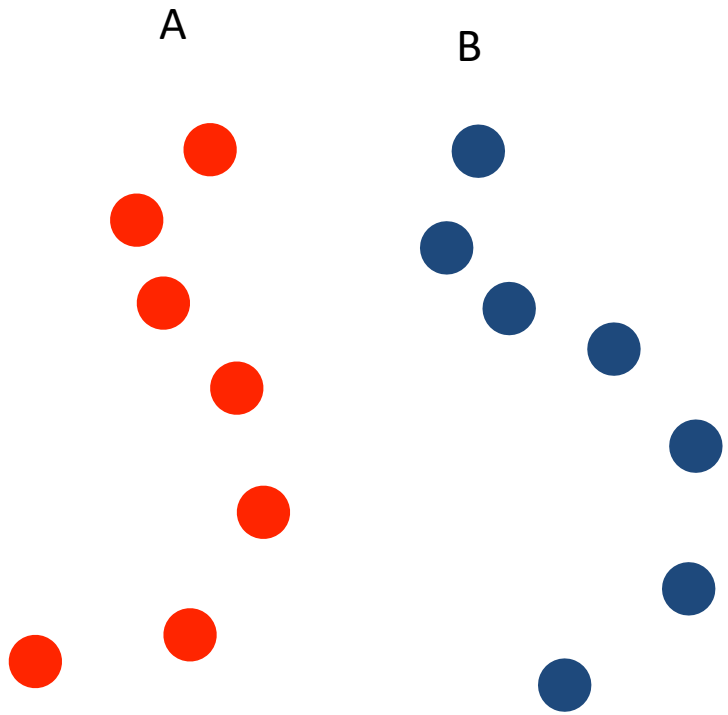
- Registration of two sets of three-dimensional points requires the computation of a rigid transformation that maps the first point set (B) to the second one (A)
- Iteratively minimise the average distance between the two point sets:
  - Find correspondences between the sets by matching every point in B with the point closest to it in A
  - Compute rigid transformation that maps B to A and apply it to B (B')
  - Compute average distance E between A and B'
  - Repeat until E is minimal



ICP in Meshlab

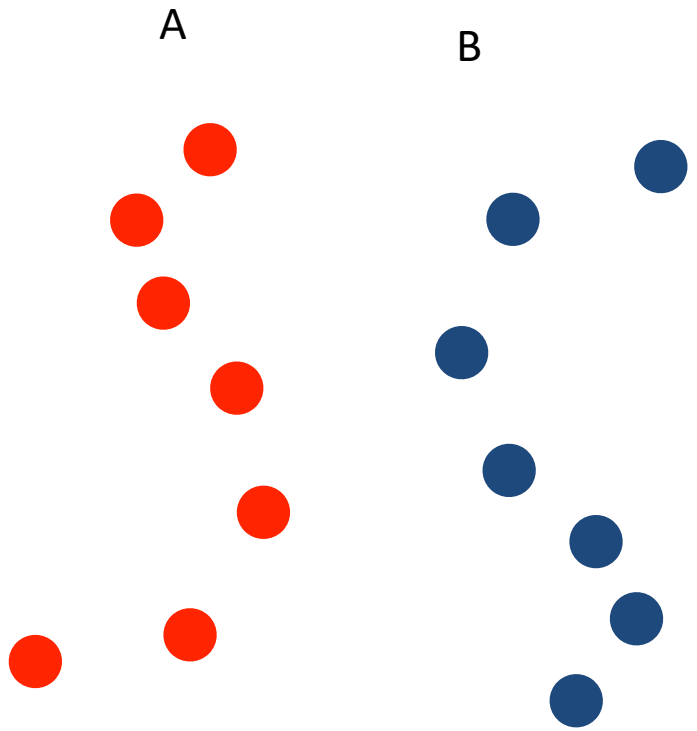
Iterative Closest-Point Method

- How to find a reasonable initial guess?



Iterative Closest-Point Method

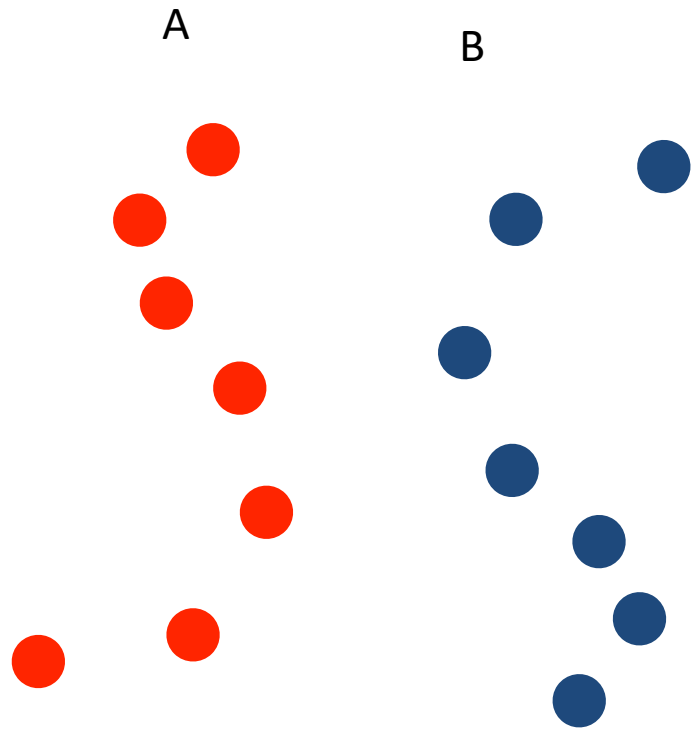
- How to find a reasonable initial guess?



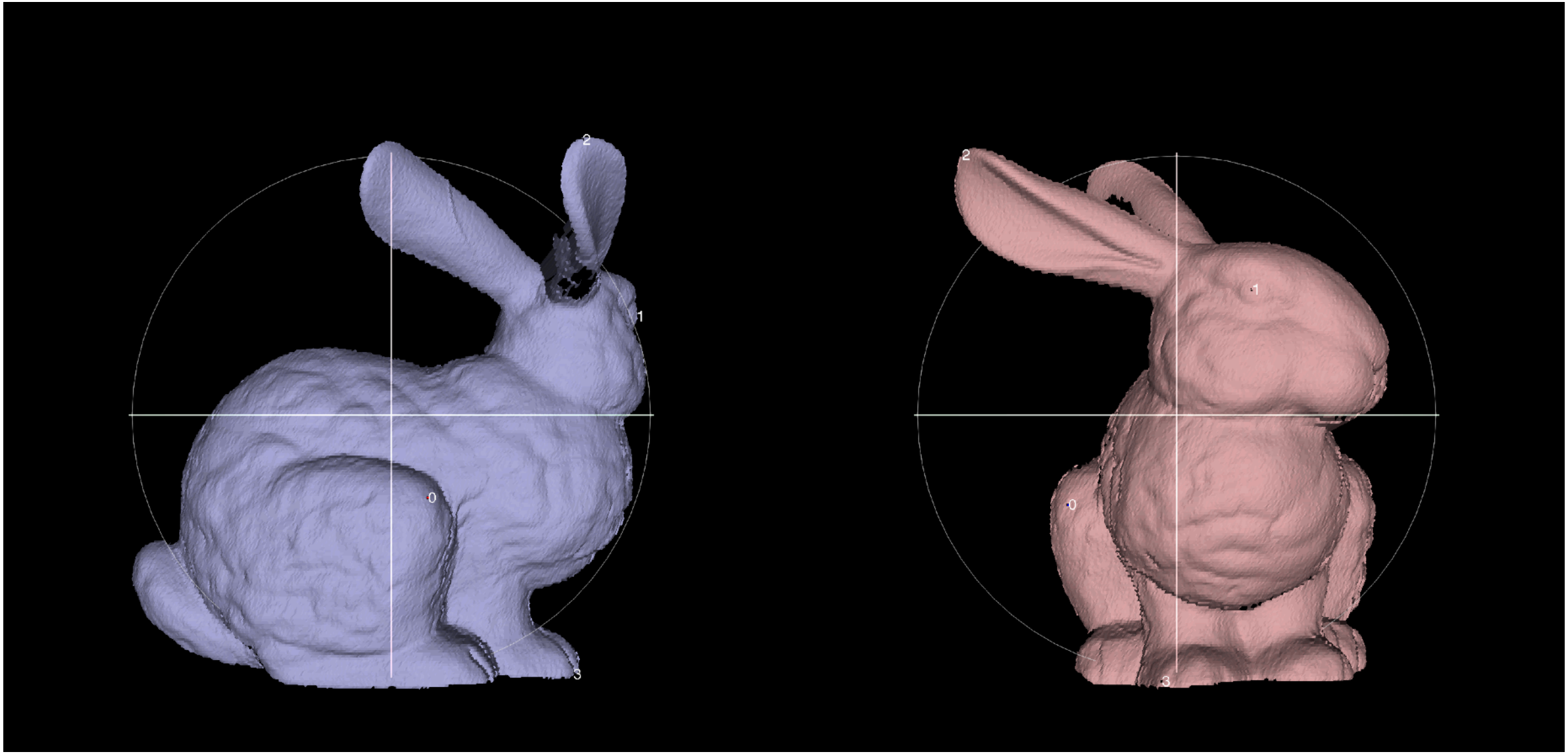
Iterative Closest-Point Method

- How to find a reasonable initial guess?
  - Manually
  - Matching distinctive features between clouds
  - Using sensor data
  - Global registration algorithms (algorithms not needing initial guess)

- At every iteration finding the closest point M in A to a given point S in B takes (for n points) how long?
  - O(n2) brute-force
- Can this be done faster?
  - O(log n) for nearest-neighbour matches with k-d trees



Iterative Closest-Point Method

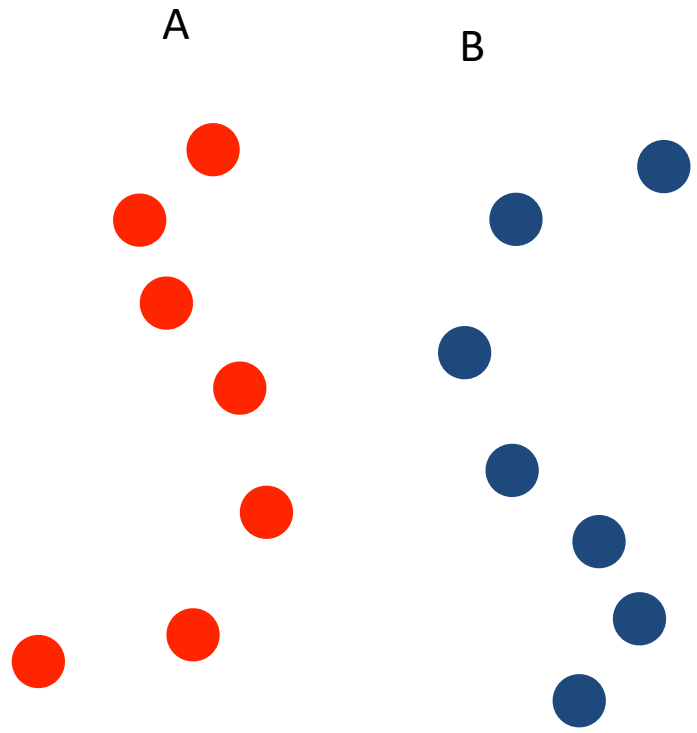


Manual alignment in Meshlab to initialise ICP

Iterative Closest-Point Method

- How to find a reasonable initial guess?
  - Manually
  - Matching distinctive features between clouds
  - Using sensor data
  - Global registration algorithms (algorithms not needing initial guess)

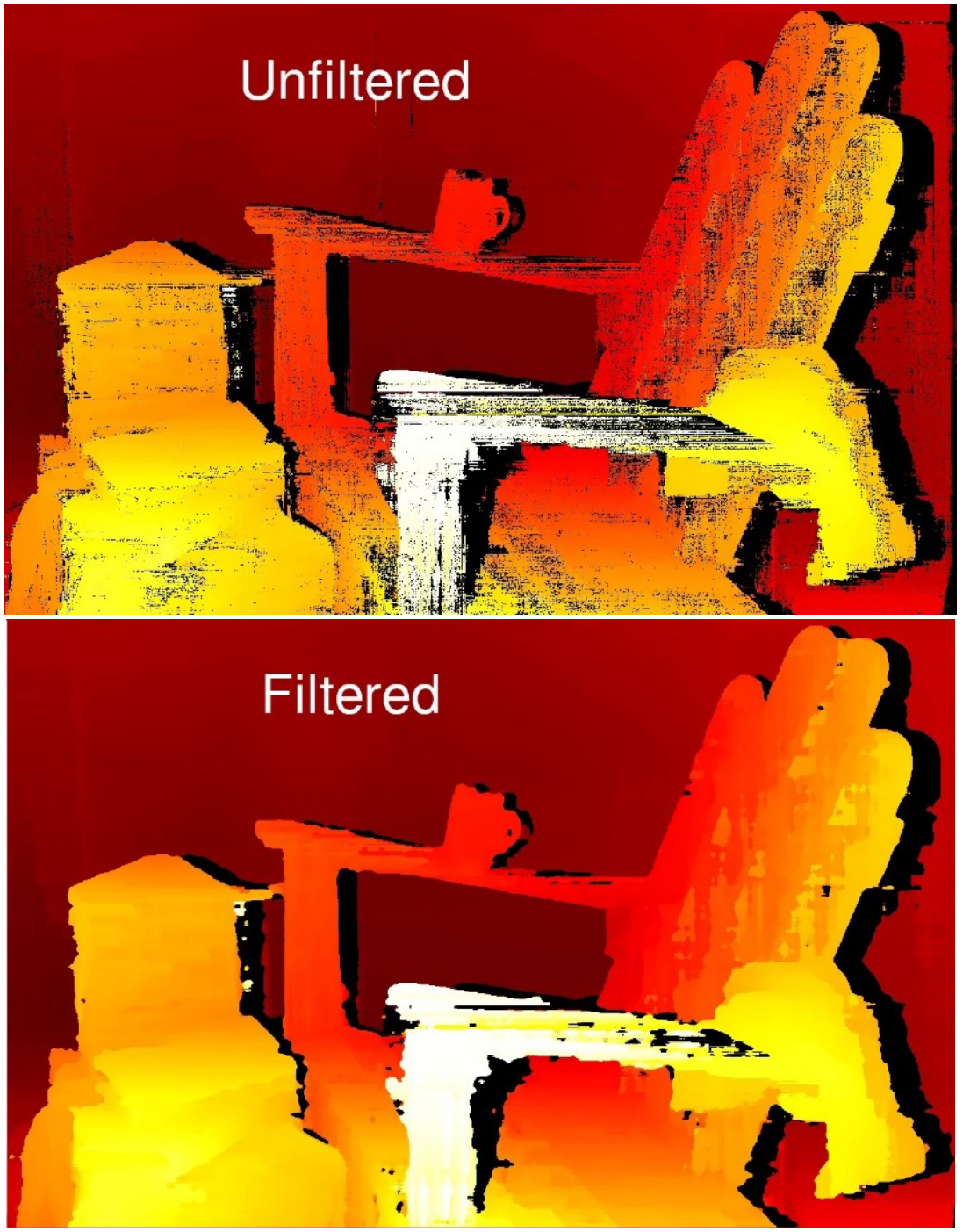
- At every iteration finding the closest point M in A to a given point S in B takes (for n points) how long?
  - O(n2) brute-force
- Can this be done faster?
  - O(log n) for nearest-neighbour matches with k-d trees



Processing Depth Maps

Processing Depth Maps

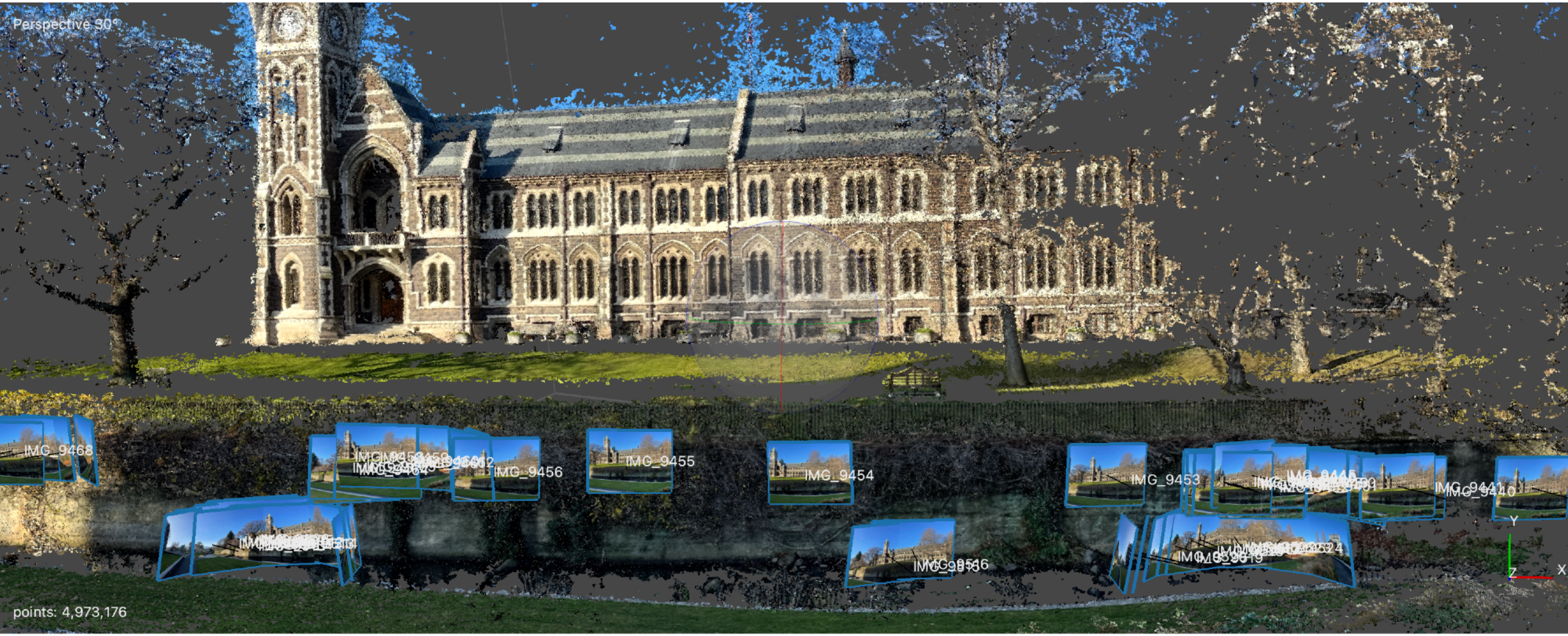
- Depth maps can be processed like images
- Image operators, such as filters, can be applied to:
  - Handle discontinuities and overlaps
  - Fill missing portions
  - Smoothen geometric noise
  - Analyse depth images
    - Find features such as edges
    - Segmentation of objects
    - Etc.



3-Min Discussion: Where do you see challenges with this approach for creating larger 3D models?

3D Reconstruction

Goal

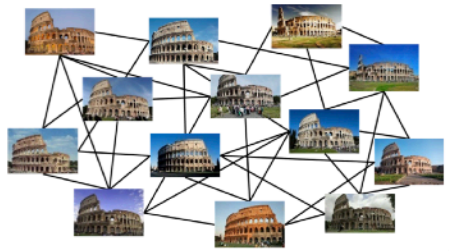

Perspective 30°
IMG_9468
IMG_9456
IMG_9455
IMG_9454
IMG_9453
points: 4,973,176

Workflow

Unstructured Images

Assoc.

Scene Graph



SFM

Sparse Model

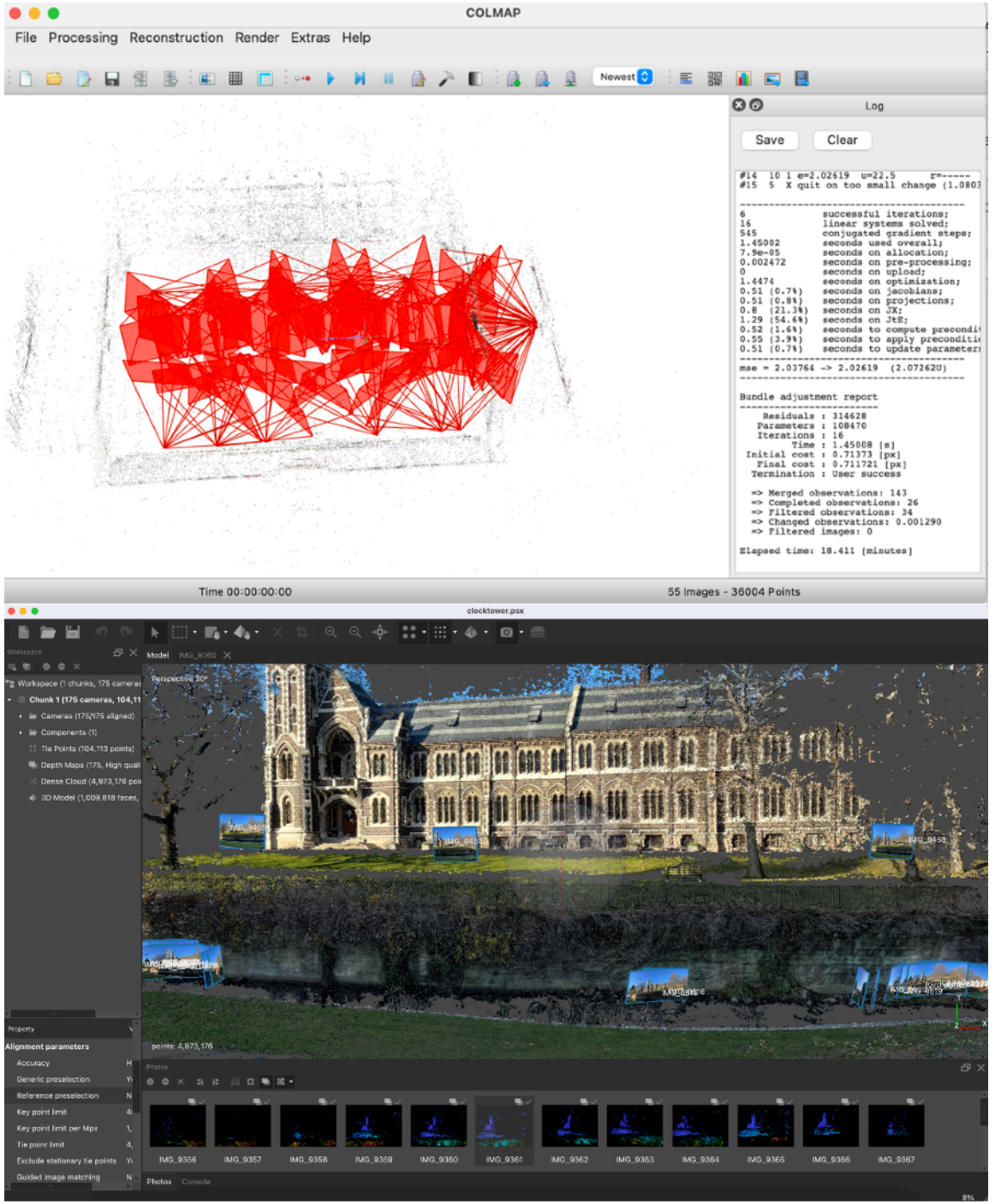
MVS

Dense Model

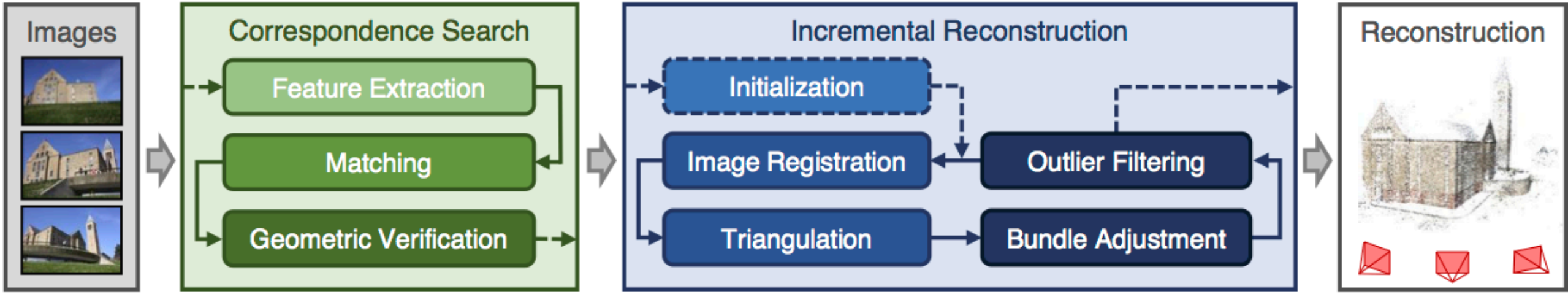
https://demuc.de/tutorials/cvpr2017/

Selection of Tools

- COLMAP https://colmap.github.io
- Metashape
- Meshroom
- VisualSFM
- Pix4D

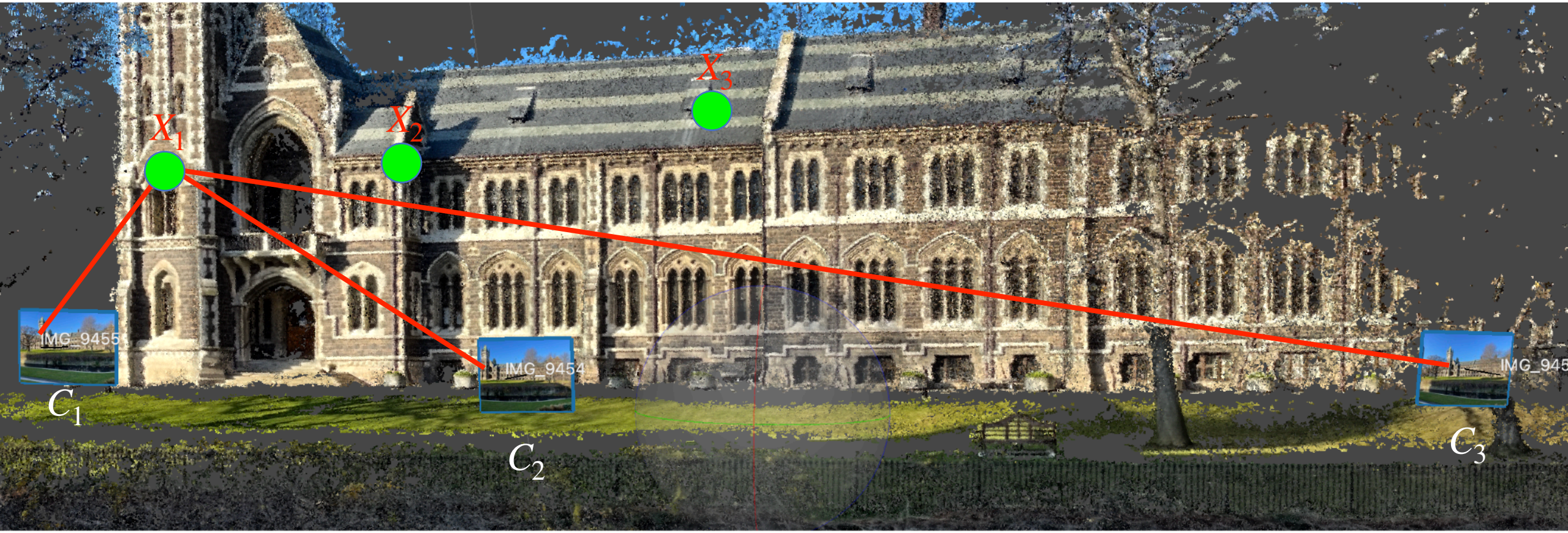


Colmap Workflow

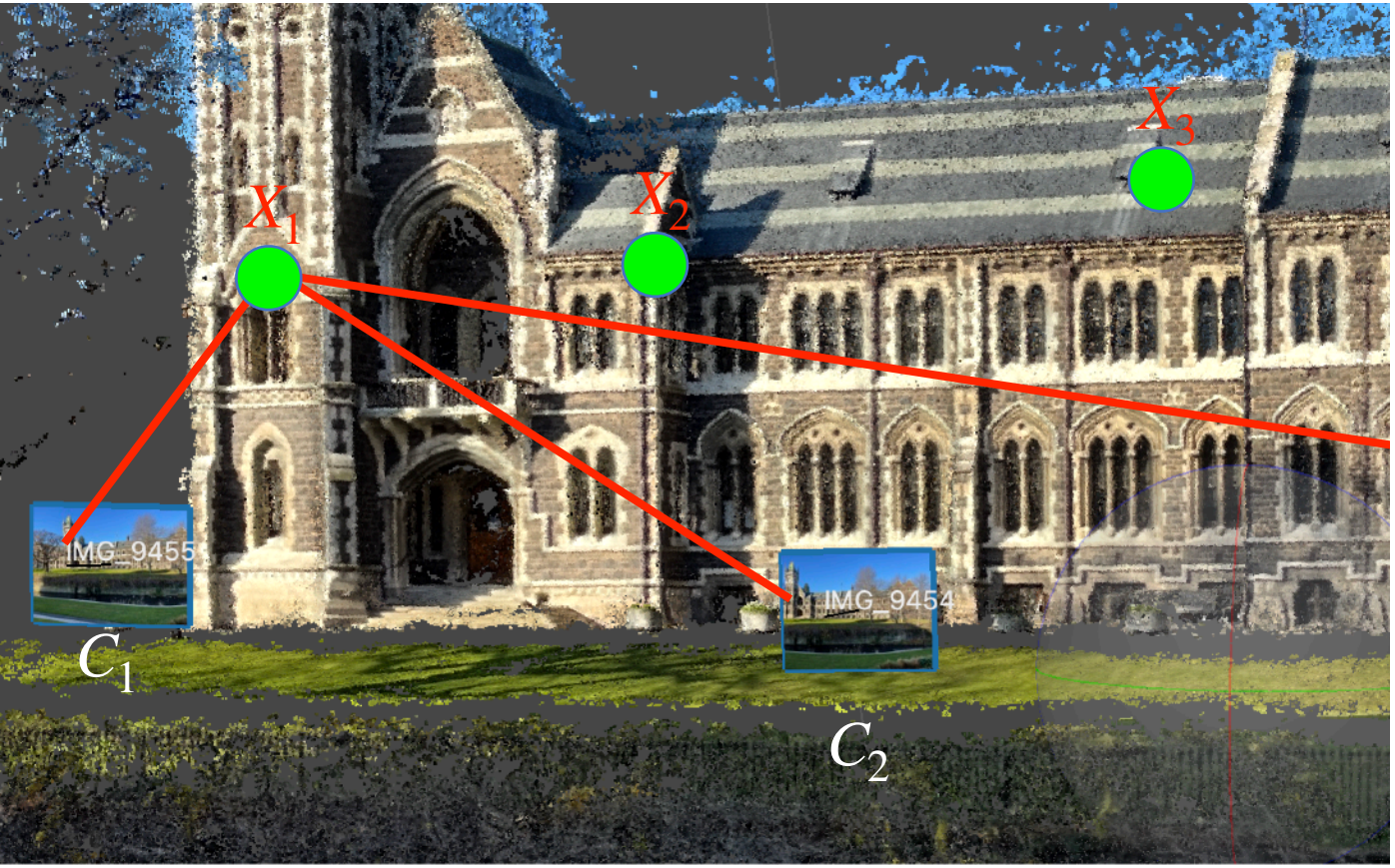


https://colmap.github.io/tutorial.html

Structure-from-Motion



Structure-from-Motion



- Estimation of 3D points X_i and camera poses C_j
- Based on motion (camera views from different viewpoints)

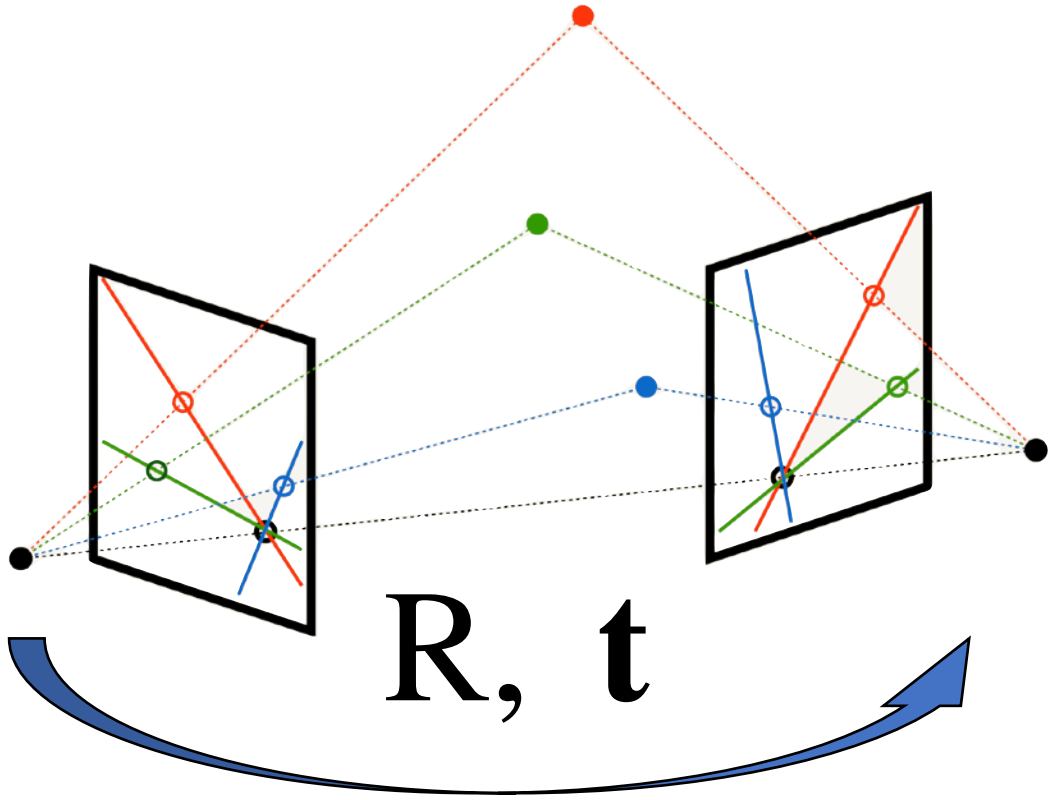
Relative Pose Estimation

The Essential Matrix

- Reminder: Fundamental matrix

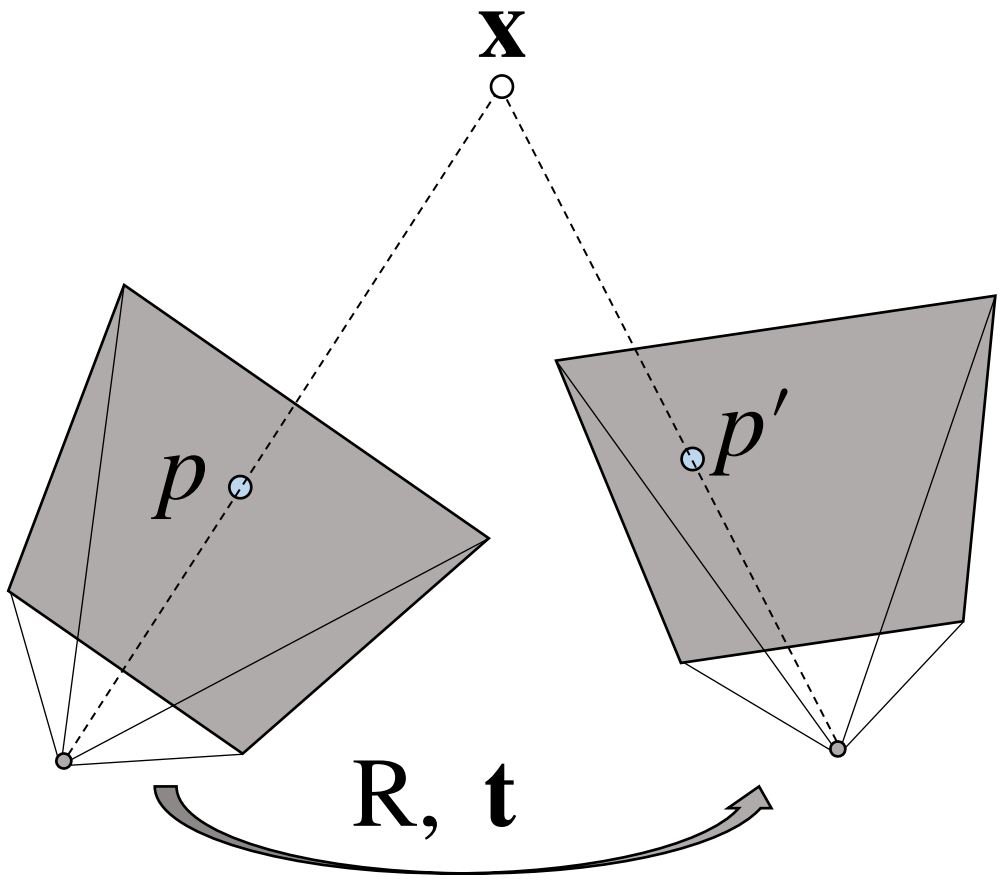
$$\mathbf{F} = \mathbf{K}_2^{-T}[\mathbf{t}]_\times \mathbf{R}\mathbf{K}_1^{-1}$$

- For calibrated cameras
  - We can factor out $\mathbf{K}_1$ and $\mathbf{K}_2$ in $\mathbf{F} = \mathbf{K}_2^{-T}[\mathbf{t}]_\times \mathbf{R}\mathbf{K}_1^{-1}$
    - This gives us the essential matrix $\mathbf{E} = \mathbf{K}_2^T \mathbf{F} \mathbf{K}_1 = [\mathbf{t}]_\times \mathbf{R}$
  - $\mathbf{E}$ captures the relative pose of the two cameras
    - A rotation, $\mathbf{R}$, and translation, $\mathbf{t}$



Relative Pose Estimation

- To recover 3D world point, $\mathbf{x}$
  - We need to know the rotation, $\mathbf{R}$, and translation, $\mathbf{t}$, between the cameras
  - This is known as their relative pose
- Conceptually, we intersect rays
  - From camera the centres through the matching image points $p \leftrightarrow p'$
  - Intersection (or nearest approach) is an estimate of $\mathbf{x}$

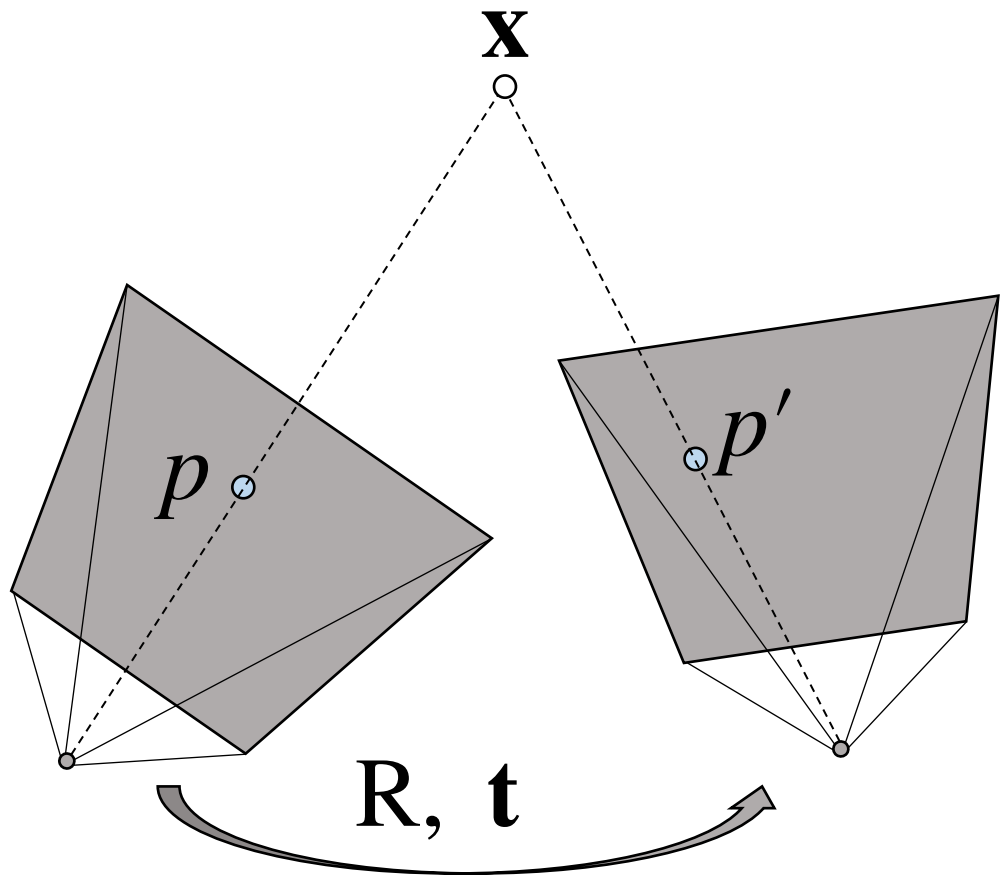


Relative Pose Estimation

- E encodes R and $\mathbf{t}$: $\mathrm{E} = [\mathbf{t}]_{\times}\mathrm{R}$
  - $[\mathbf{t}]_{\times}$ is the matrix that has the effect of giving the cross product with $\mathbf{t}$

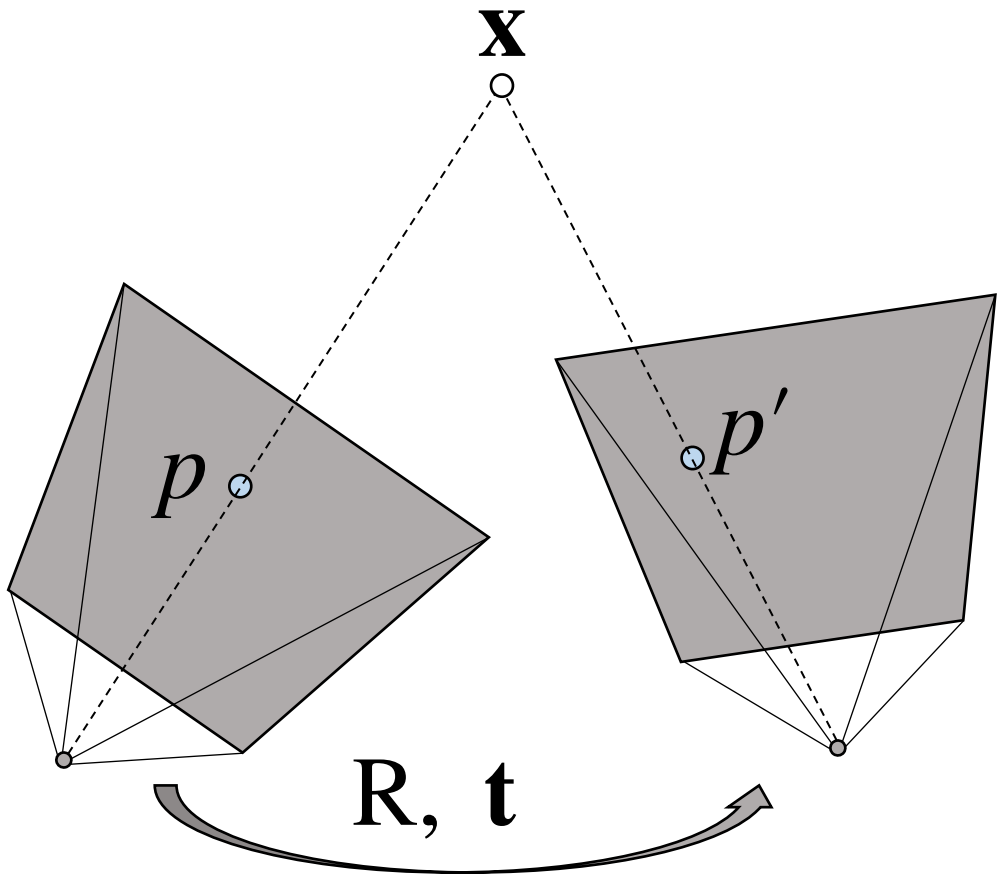
$$[\mathbf{t}]_{\times} = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$$

  - How do we recover R and $\mathbf{t}$ from E?

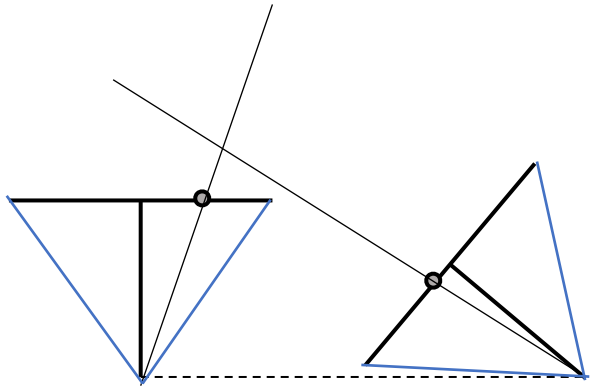


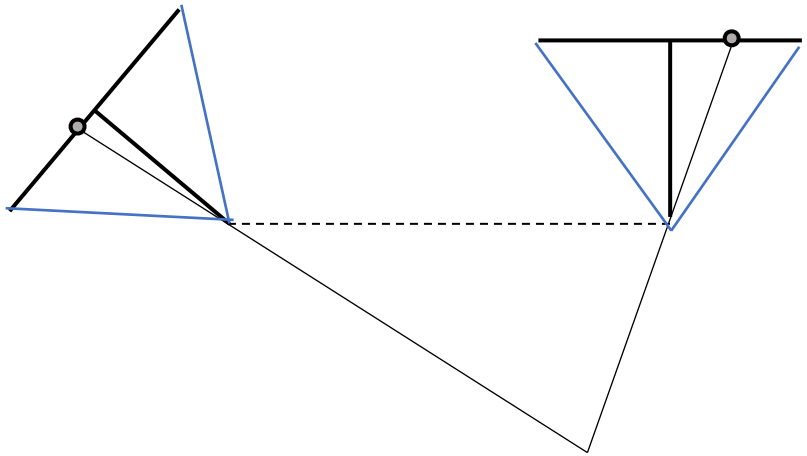
Decomposing the Essential Matrix

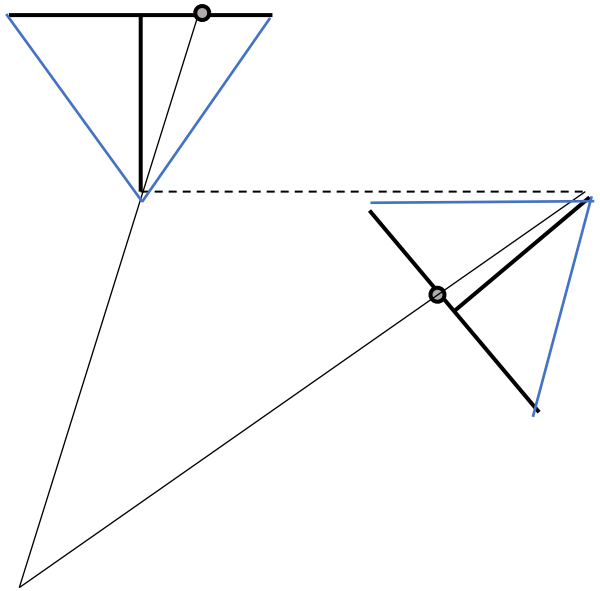
- To recover R and t
  - Compute SVD of $\mathbf{E}$
  - SVD: gives us three matrices U S and V transpose: $\mathbf{E} = \mathbf{U}\mathbf{S}\mathbf{V}^T$
  - From this decomposition, we get:
    - Two possible rotations (R)
    - Two possible translations (t)
  - That means there are four candidate solutions in total.
  - To find the correct one: check which solution places the 3D points in front of both cameras

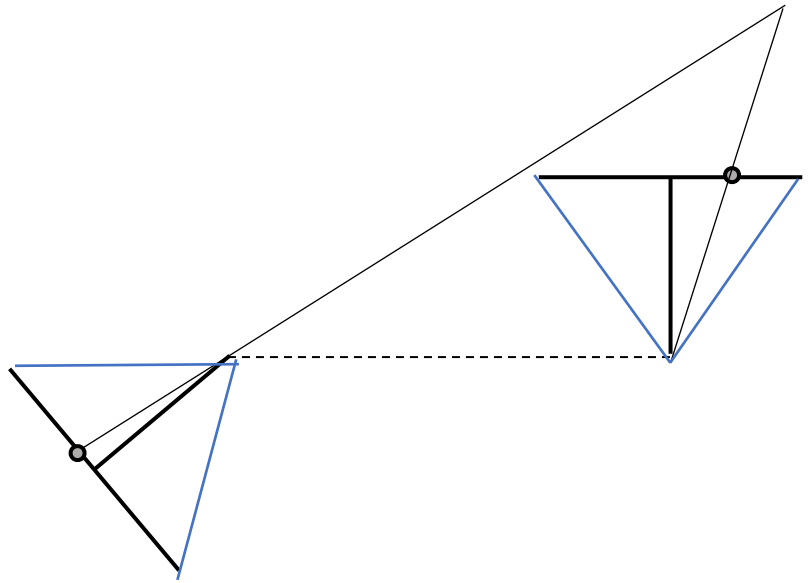


Which Solution is Correct?









3D Reconstruction

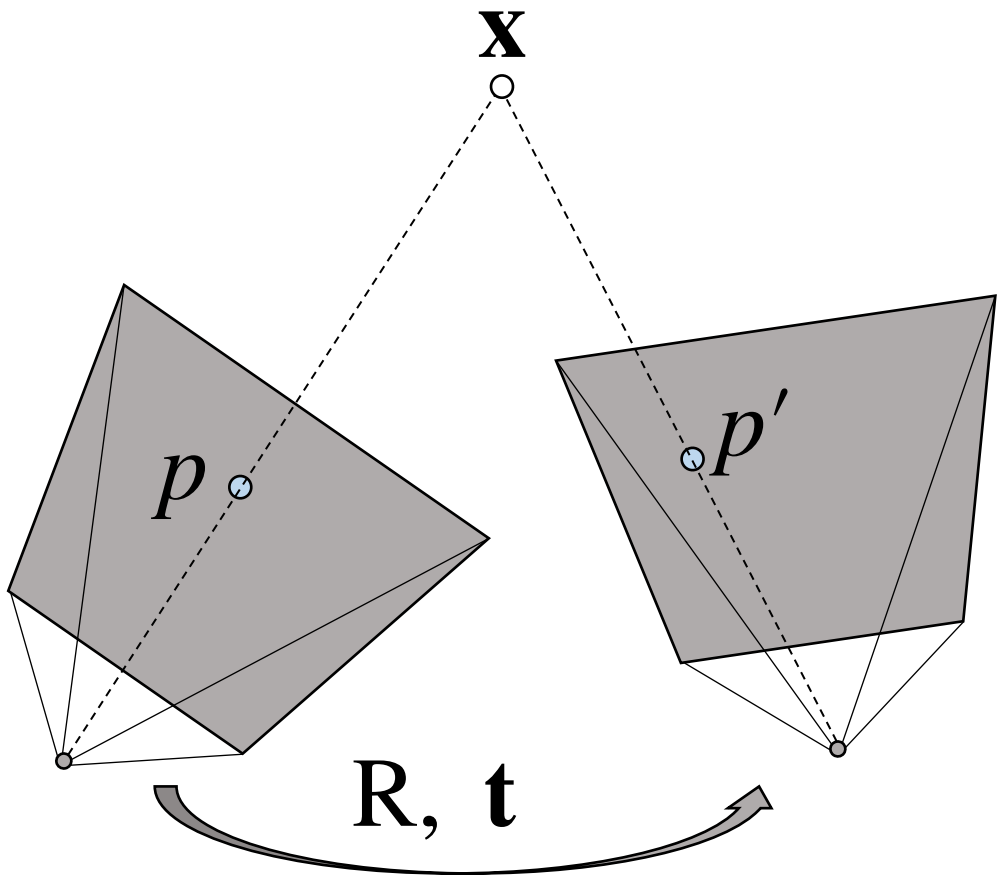
3D Structure Estimation

- We now know
  - Calibration matrices, $\mathbf{K}$ and $\mathbf{K}'$
  - Relative pose, $\mathbf{R}$ and $\mathbf{t}$
  - Matching points, $p \leftrightarrow p'$

- These give us a 3D world point, $\mathbf{x}$

$$p \equiv \mathbf{K}\begin{bmatrix}\mathbf{I} & \mathbf{0}\end{bmatrix}\mathbf{x} = \mathbf{P}\mathbf{x}$$

$$p' \equiv \mathbf{K}'\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\mathbf{x} = \mathbf{P}'\mathbf{x}$$



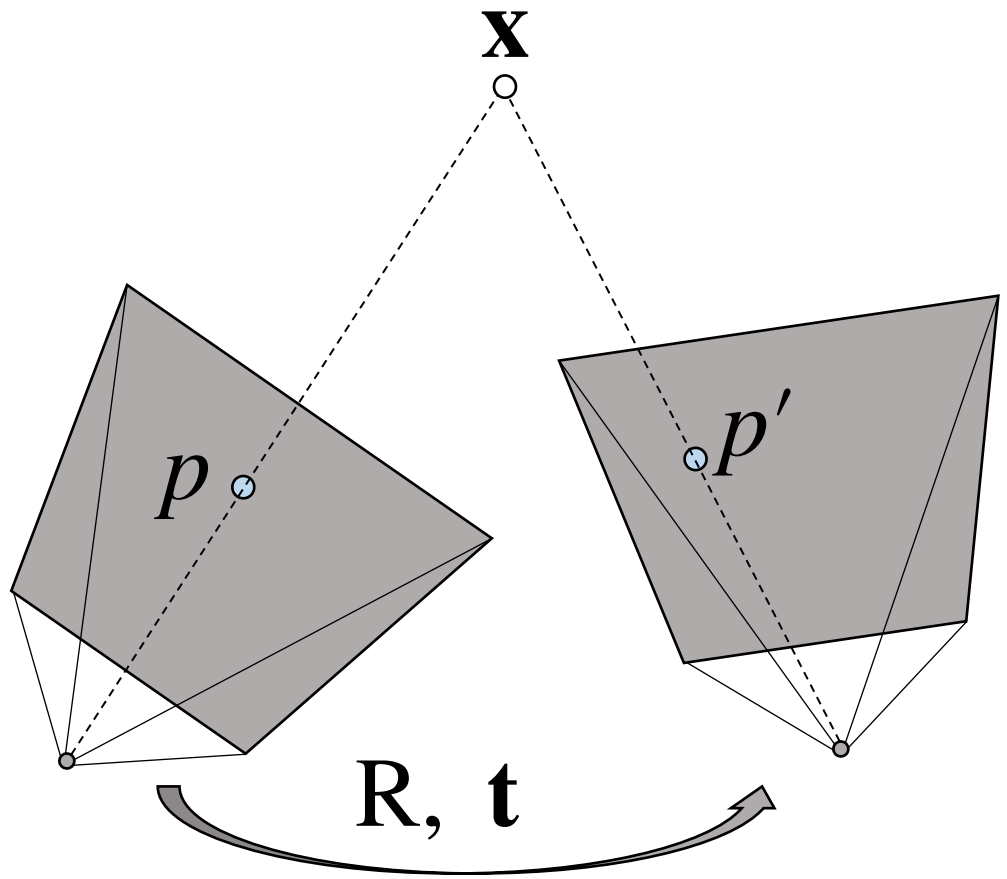
3D Structure Estimation

- If we let $\mathbf{r}_i^T$ be the ith row of P:

$$p \times \mathrm{P}\mathbf{x} = \mathbf{0}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \times \begin{bmatrix} \mathbf{r}_1^T\mathbf{x} \\ \mathbf{r}_2^T\mathbf{x} \\ \mathbf{r}_3^T\mathbf{x} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} \left(v\mathbf{r}_3^T - \mathbf{r}_2^T\right)\mathbf{x} \\ \left(\mathbf{r}_1^T - u\mathbf{r}_3^T\right)\mathbf{x} \\ \left(u\mathbf{r}_2^T - v\mathbf{r}_1^T\right)\mathbf{x} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$



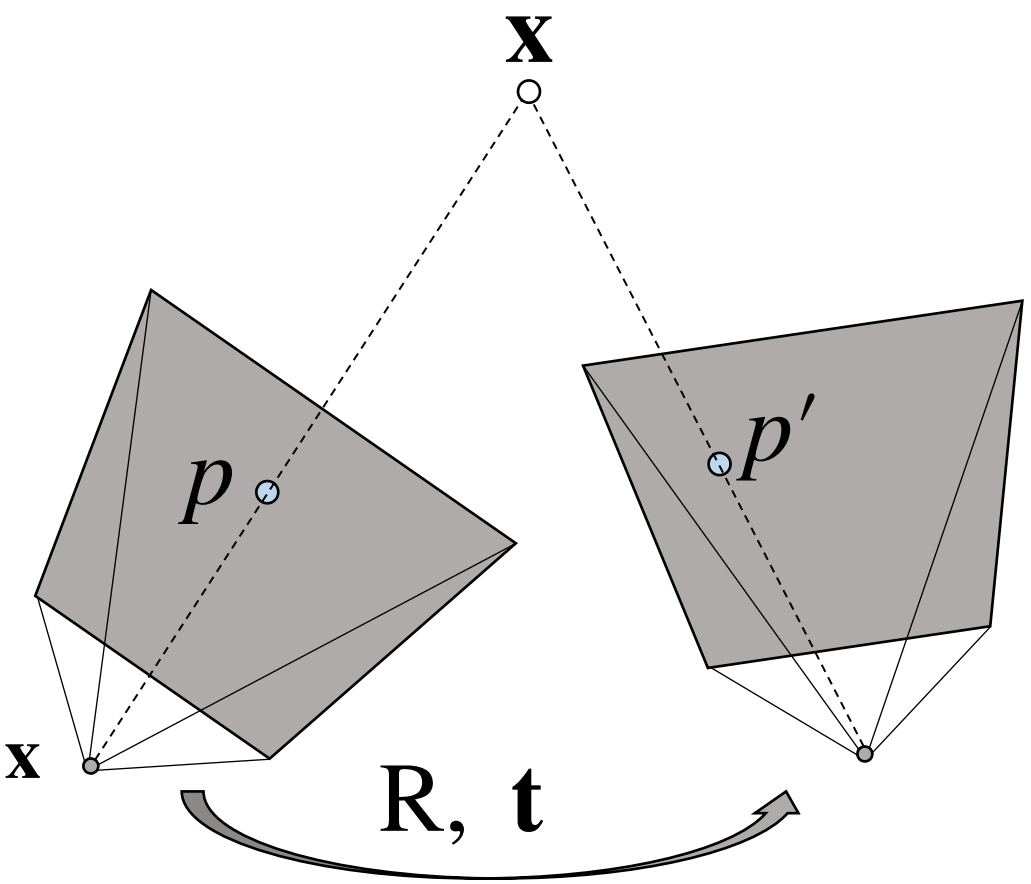
3D Structure Estimation

- This looks like 3 equations in $\mathbf{x}$
  - They are not linearly independent
  - Can only use two
  - Usually take the first two (symmetry)
  - Get two more from $p' \times \mathbf{P}'\mathbf{x} = \mathbf{0}$

$$\left(v\mathbf{r}_3^T - \mathbf{r}_2^T\right)\mathbf{x} = 0$$
$$\left(\mathbf{r}_1^T - u\mathbf{r}_3^T\right)\mathbf{x} = 0$$
$$\left(v'\mathbf{r}_3'^T - \mathbf{r}_2'^T\right)\mathbf{x} = 0$$
$$\left(\mathbf{r}_1'^T - u'\mathbf{r}_3'^T\right)\mathbf{x} = 0$$

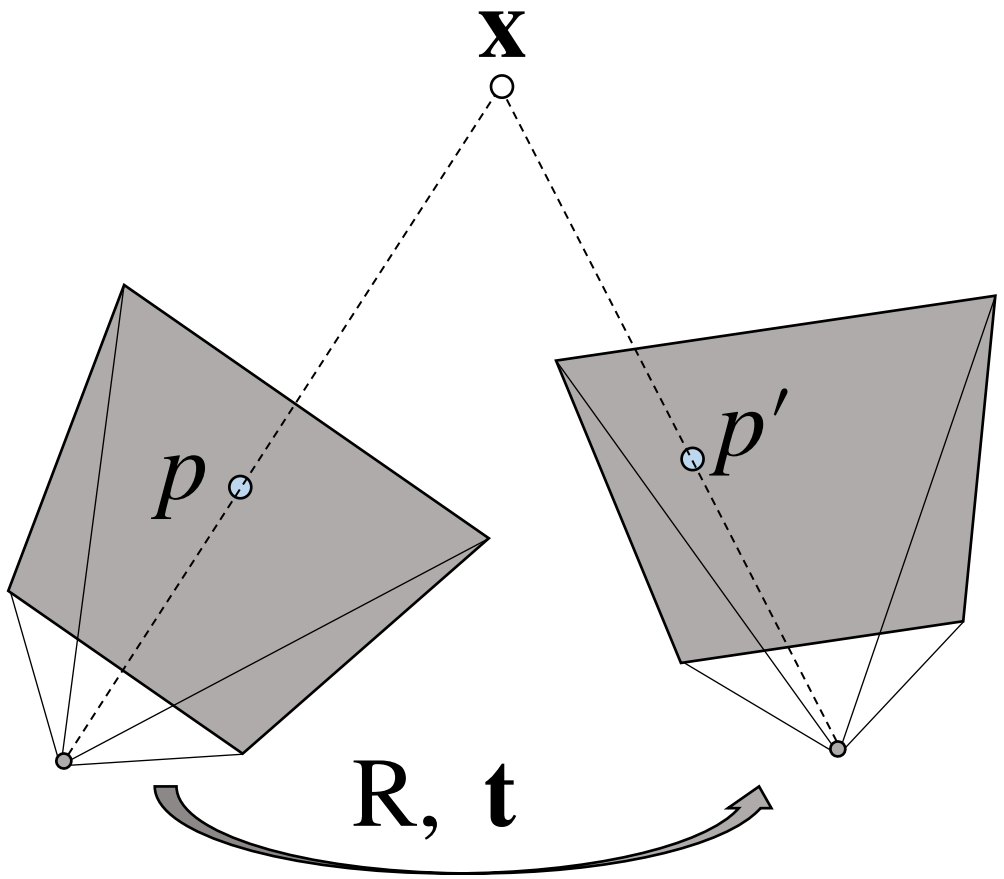
Four equations in $\mathbf{x}$

Solve $\mathrm{A}\mathbf{x} = \mathbf{0}$



Reprojection Error

- From our measurements
  - Point correspondences $p_i \leftrightarrow p_i'$
  - Camera calibration matrices $\mathbf{K},$ and $\mathbf{K}'$
- We have estimated
  - The fundamental/essential matrices
  - The camera's relative pose, $\mathbf{R},\ \mathbf{t}$
  - The 3D world points, $\mathbf{x}_i$
- Have minimised algebraic error
  - Convenient, but not meaningful

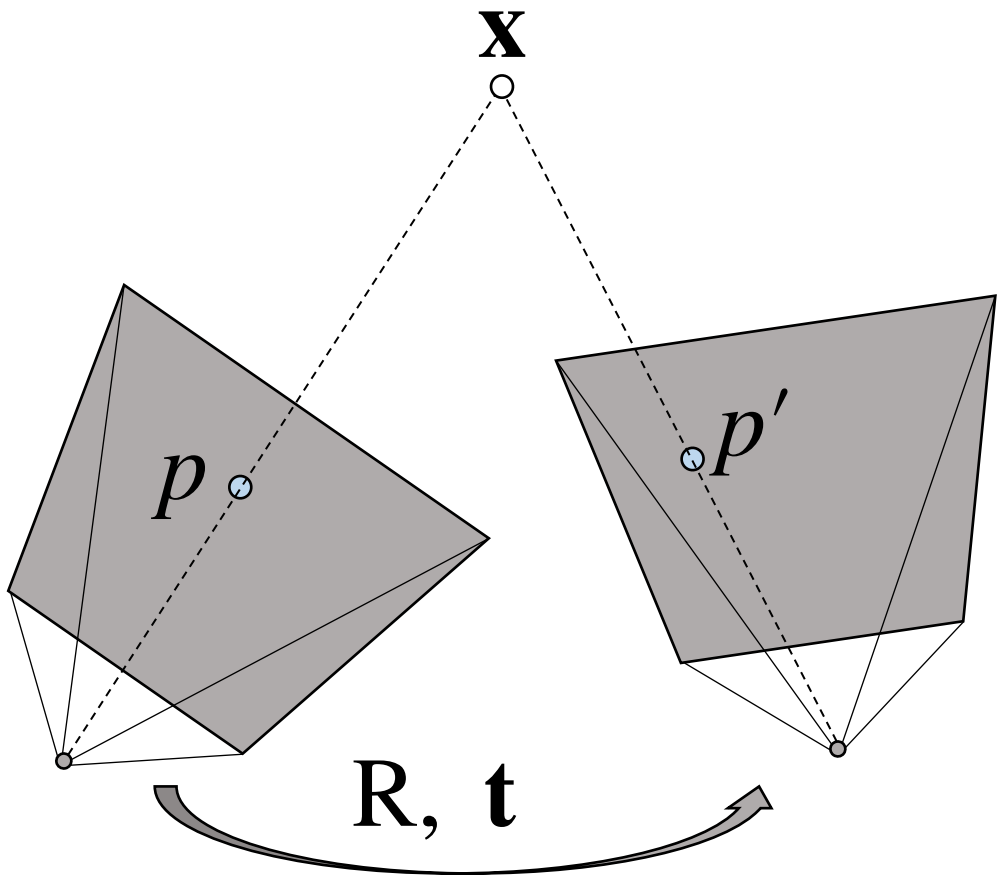


Reprojection Error

- There are true values for
  - The camera calibration matrices $\tilde{\mathrm{K}},\ \tilde{\mathrm{K}}'$
  - Their relative pose, $\tilde{\mathrm{R}},\ \tilde{\mathbf{t}}$
  - The 3D world points, $\boldsymbol{x}_i$
- These predict our measurements

$$\tilde{p}_i = \tilde{\mathrm{K}}\begin{bmatrix} \mathrm{I} & \mathbf{0} \end{bmatrix}\tilde{\mathbf{x}}_i$$

$$\tilde{p}'_i = \tilde{\mathrm{K}}'\begin{bmatrix} \tilde{\mathrm{R}} & \tilde{\mathbf{t}} \end{bmatrix}\tilde{\mathbf{x}}_i$$



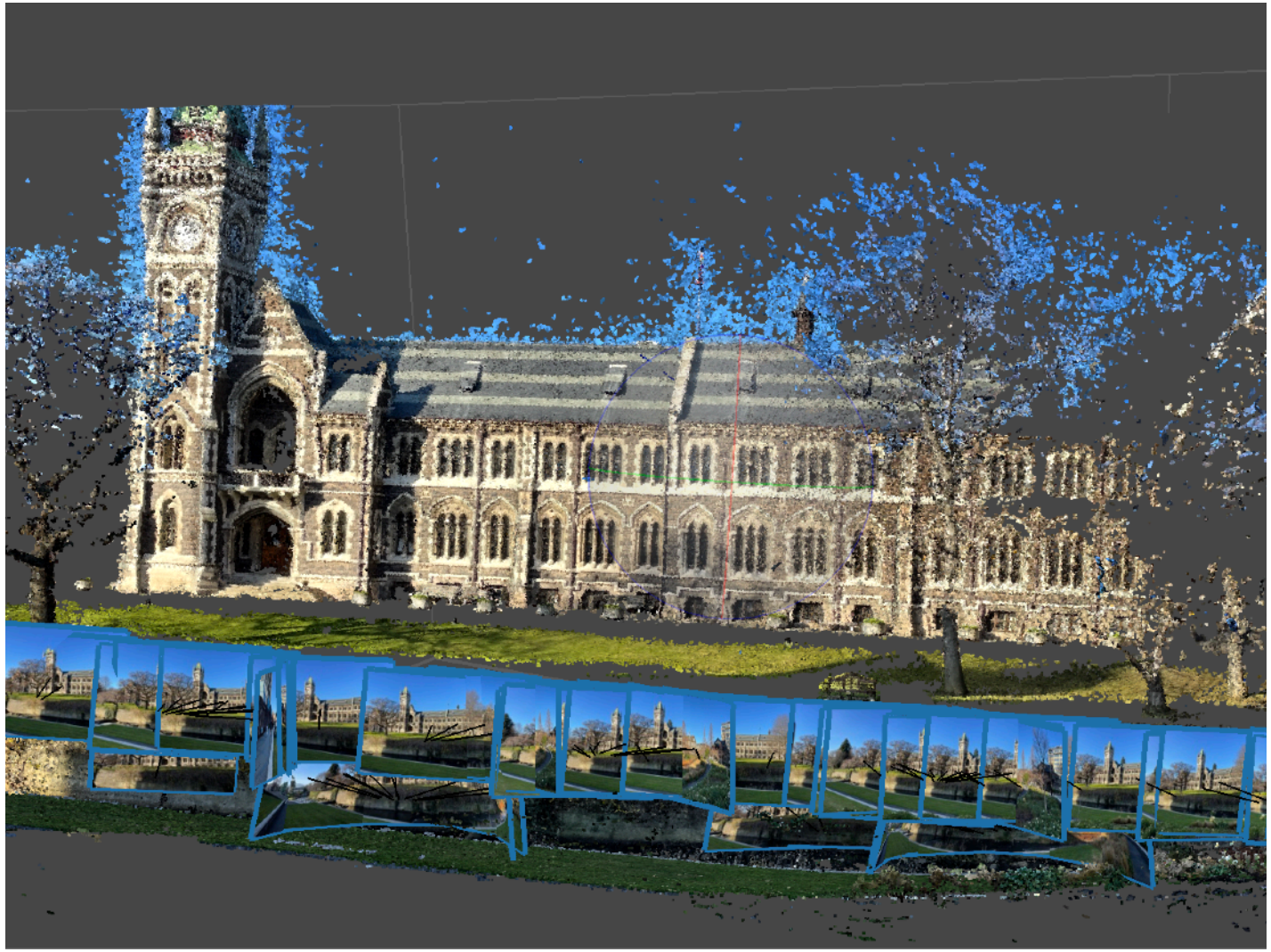
Reprojection Error

$$\sum \left(\left(p_i - \tilde{p}_i \right)^2 + \left(p_i' - \tilde{p}_i' \right)^2 \right) = \sum \left(\left(p_i - \tilde{\mathrm{K}} \left[\mathrm{I} \quad \mathbf{0} \right] \tilde{\mathbf{x}}_i \right)^2 + \left(p_i' - \tilde{\mathrm{K}}' \left[\tilde{\mathrm{R}} \quad \tilde{\mathbf{t}} \right] \tilde{\mathbf{x}}_i \right)^2 \right)$$

- This is a non-linear function with parameters $\tilde{\mathbf{K}},\ \tilde{\mathbf{K}}',\ \tilde{\mathbf{R}},\ \tilde{\mathbf{t}},\ \tilde{\mathbf{x}}_i$
  - Have initial estimates from minimising algebraic error
  - Minimise via Levenberg-Marquardt

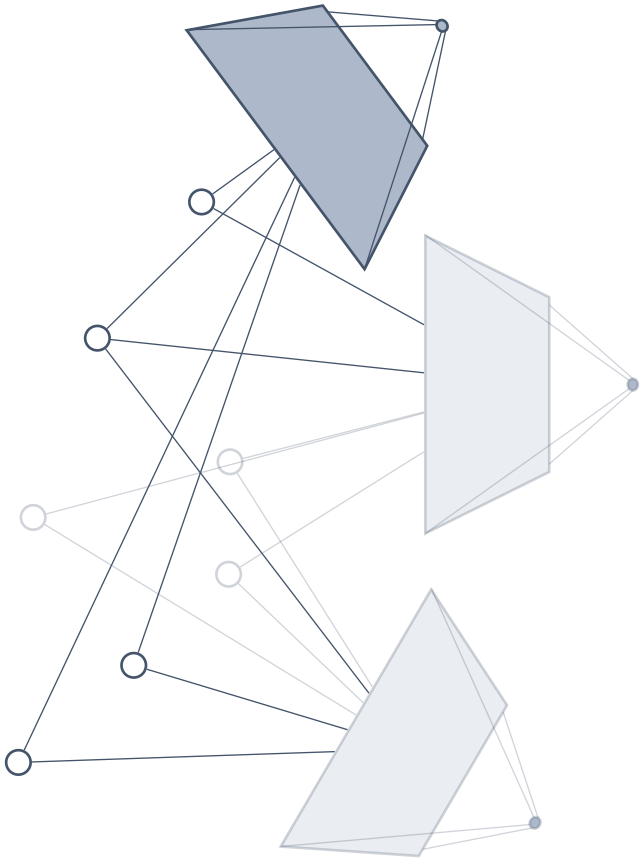
Multi-View Stereo

- More than two images
  - Multiple camera rigs
  - Single moving camera
  - Multiple moving cameras
- More information
  - Reconstruct larger areas
  - Resolve more details
  - More accurate models

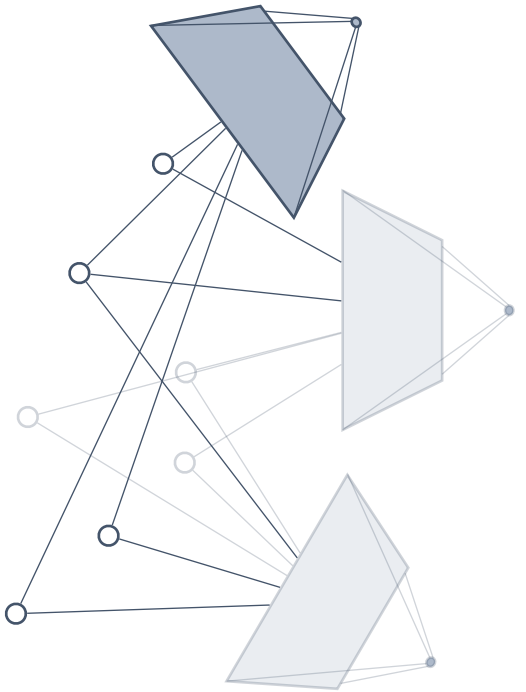


Discussion (1min):

Can we just find relative poses between each pair?

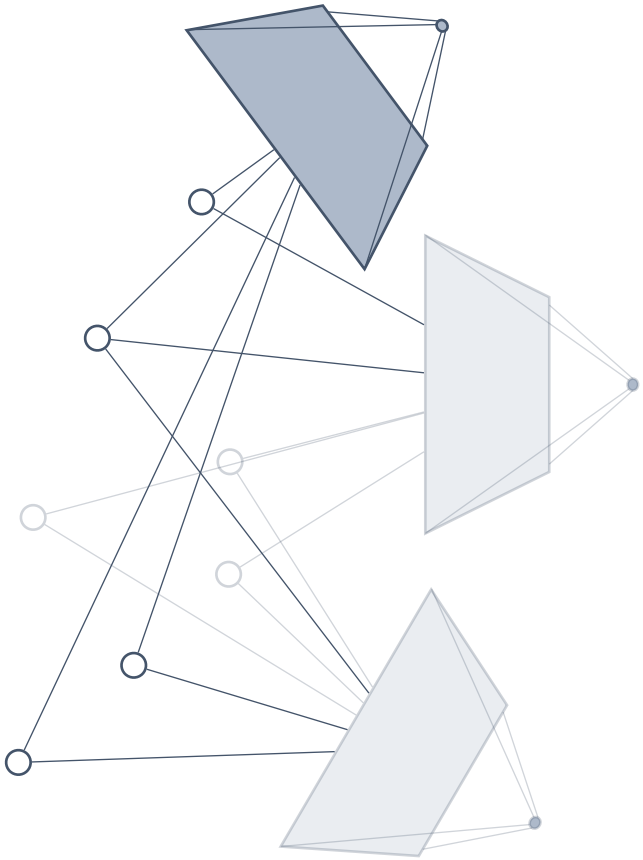


3-Min Discussion: Is finding relative poses between each pair enough? What are the challenges?



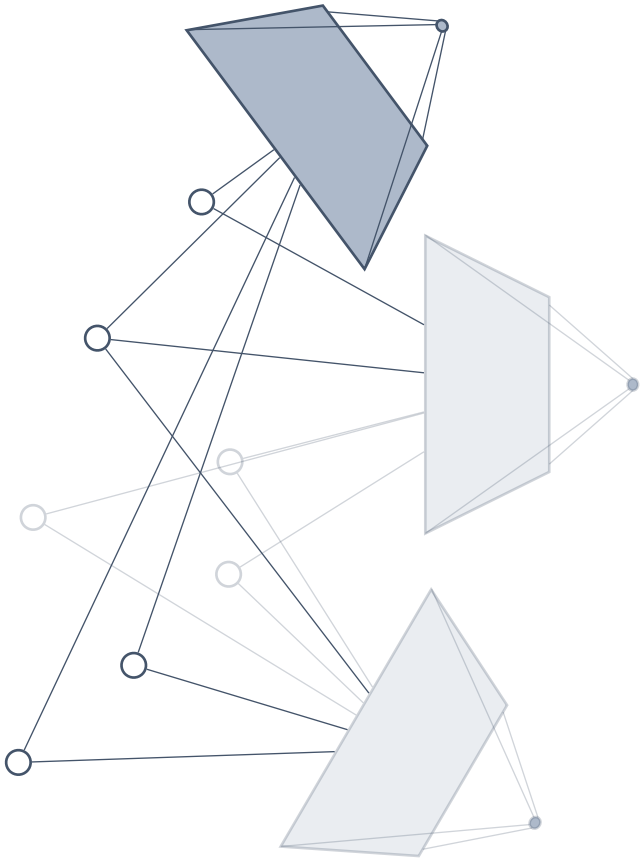
Multi-View Stereo

- Two-view methods
  - Find fundamental matrices between each pair
  - Scales are not independent
  - Non-overlapping views
- Incremental approach
  - Start with two views
  - Recover motion & structure
  - Find next camera's pose
  - Recover more structure
  - Tasks to solve
    - Determine order of images
    - Recover absolute pose



Multi-View Stereo Pipeline

- Identify best matching pair
  - Two-view stereo for relative pose
  - Reconstruct 3D points
  - Minimise reprojection error
- While still images to add
  - Select next best view
  - Determine absolute pose
  - Reconstruct more 3D points
  - Minimise reprojection error



Multi-View Stereo Pipeline

- Identify best matching pair
  - Two-view stereo for relative pose
  - Reconstruct 3D points
  - Minimise reprojection error
- While still images to add
  - Select next best view
  - Determine absolute pose
  - Reconstruct more 3D points
  - Minimise reprojection error

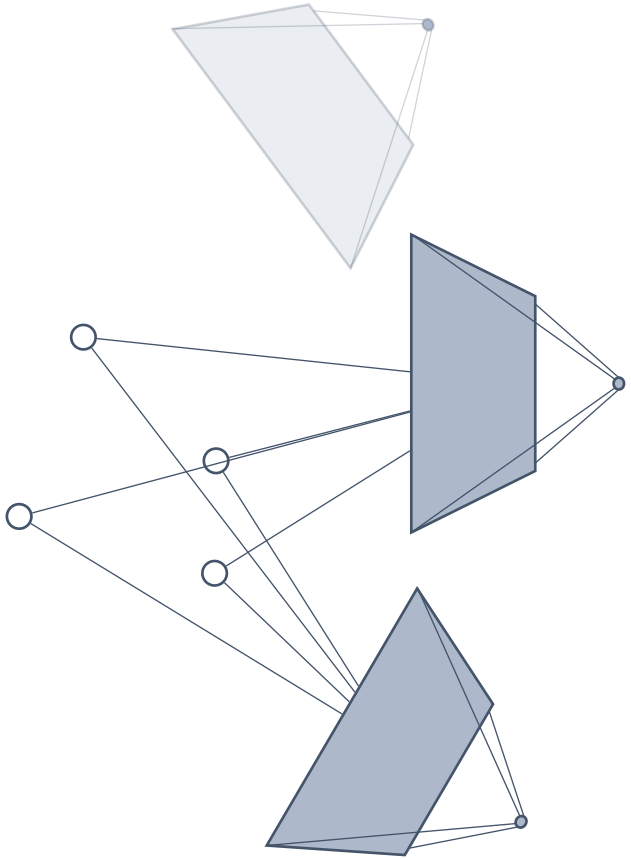


Image Bag-of-Words

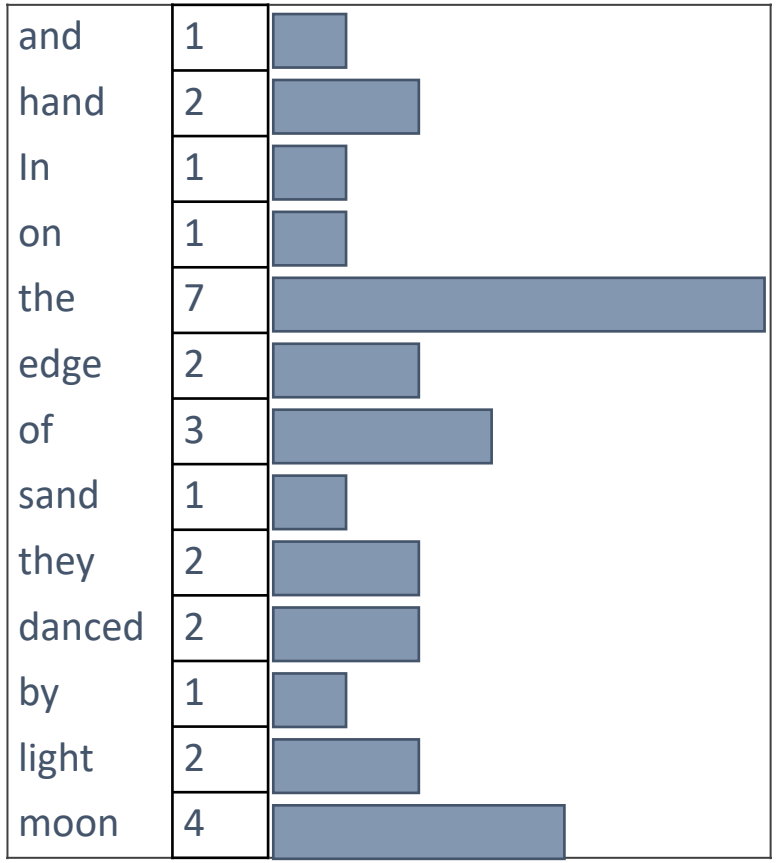
Multi-View Stereo Pipeline

- Start by finding image features
- Matching all pairs is expensive
  - For n images, $O(n)^2$ pairs
  - Feature matching between one pair is quite expensive (as we've seen)
- Need a quick check of pairs
  - How similar are two images?
  - How many likely matches are there?

Bags of Words (BoW)

And hand in hand on the edge of the sand
they danced by the light of the moon,
the moon, the moon
They danced by the light of the moon

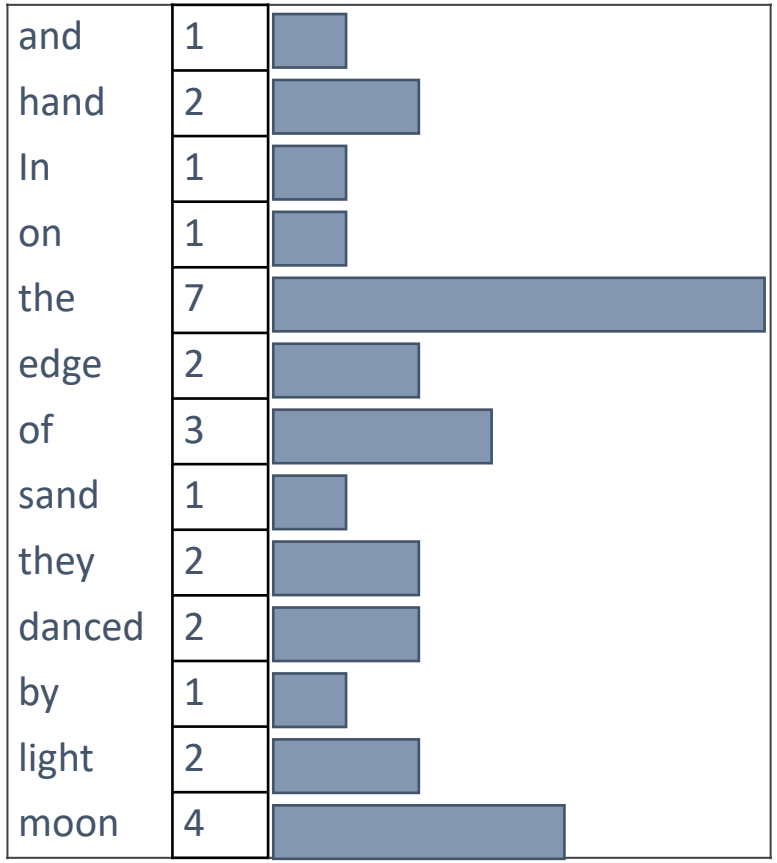
Edward Lear
from The Owl and the Pussycat



- From information retrieval
  - Group features into ‘words’
  - Count how many times each ‘word’ appears in each image -> histogram
  - Compare histograms

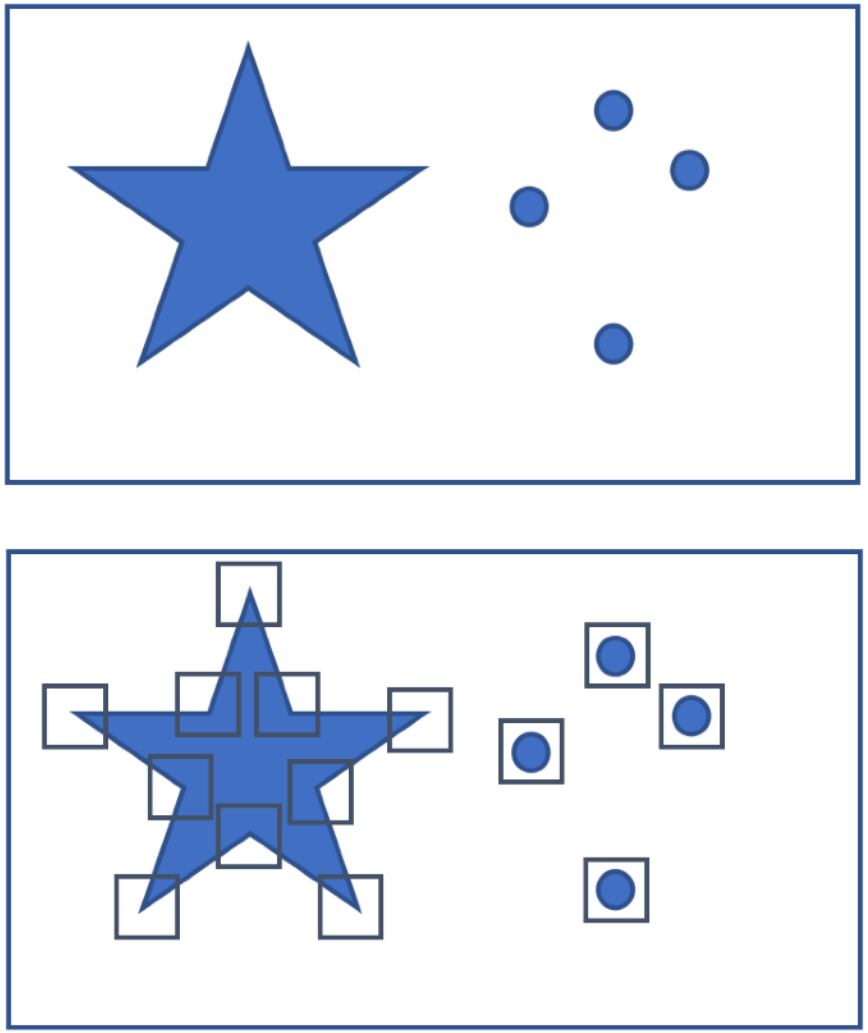
Bags of Words (BoW)

- Bag of Words (BoW) vectors:
  - Ignores word order
  - Can ignore counts – binary there/not
  - Content, not structure
- For text, group similar words
  - Computer, computing, compute…
- How to compare vectors?
- How to apply to images?



Bags of Words (BoW)

- Need equivalent of words
- Cluster features – can pre-train
  - Choose some number, k, of ‘words’
  - Make k clusters – e.g. k-means
  - Cluster centres become words
- Bag of words for images
  - Detect and describe feature
  - Count features closest to each word

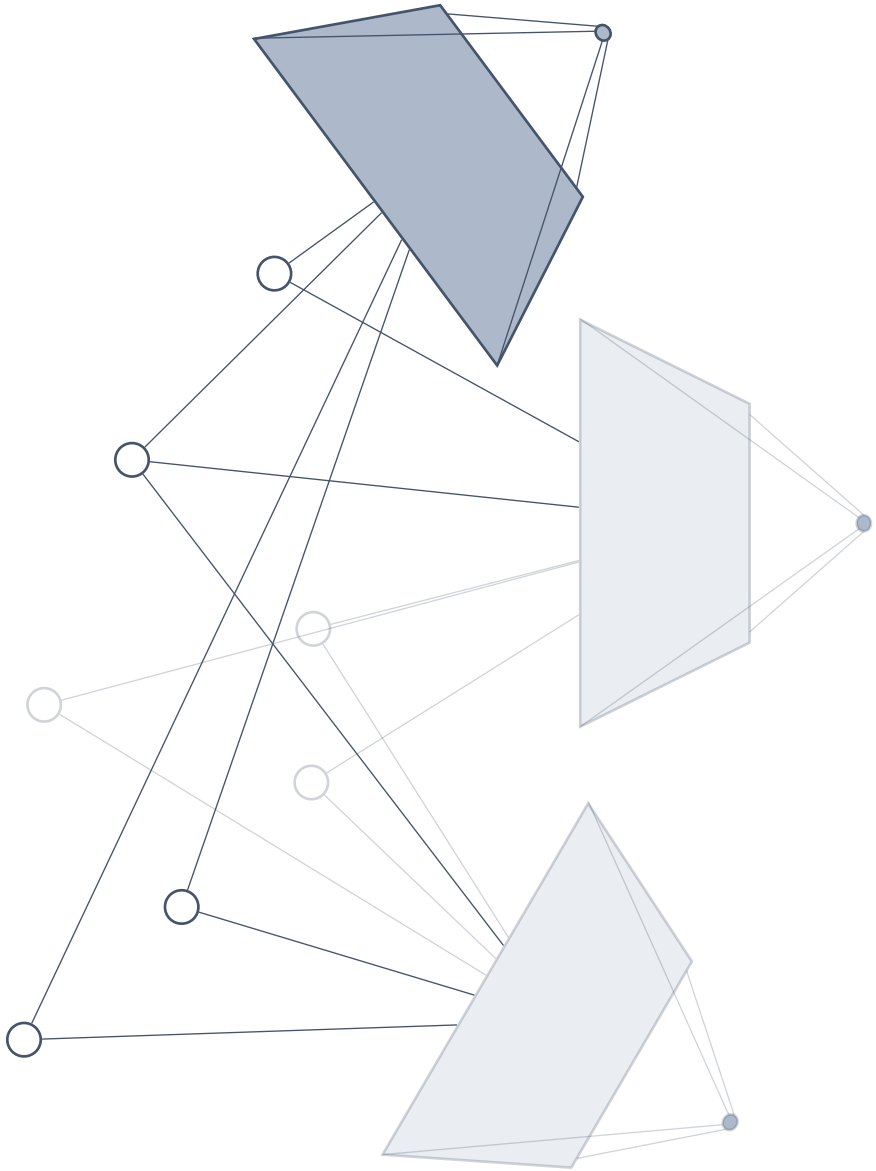


Absolute Pose

Multi-View Stereo Pipeline

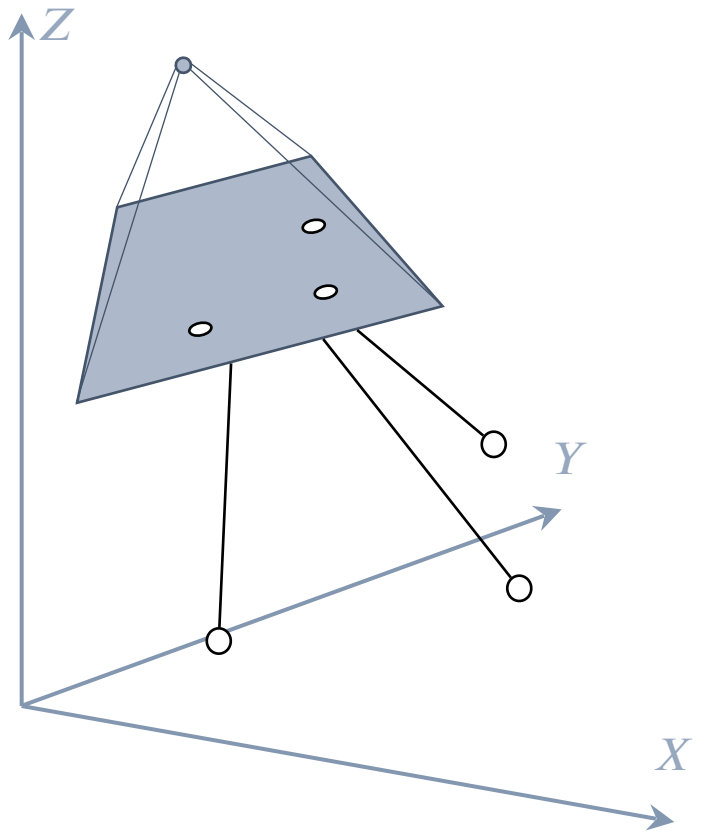
- Identify best matching pair
  - Two-view stereo for relative pose
  - Reconstruct 3D points
  - Minimise reprojection error

- While still images to add
  - Select next best view
  - *Determine absolute pose*
  - Reconstruct more 3D points
  - Minimise reprojection error



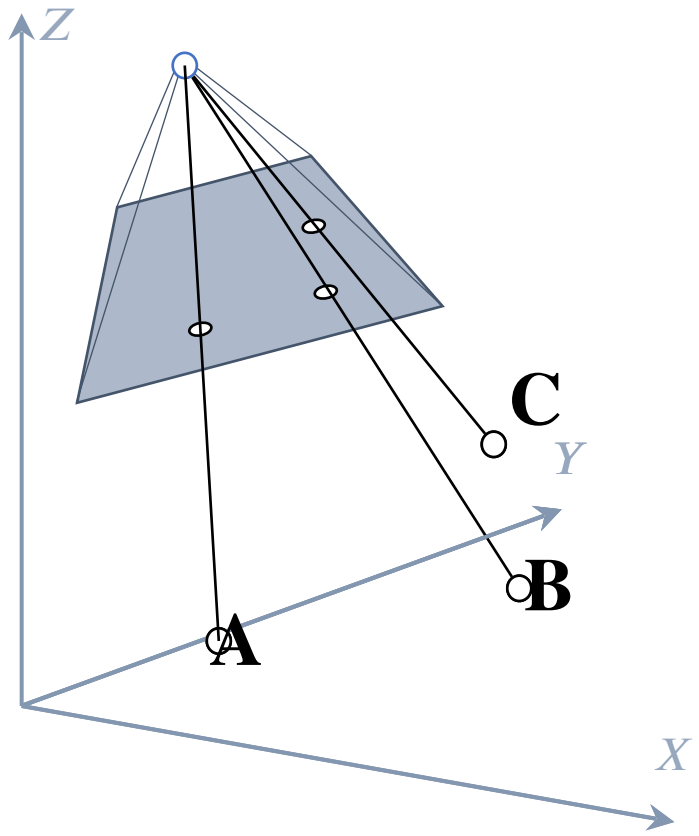
Absolute Pose Problem

- We have an image taken with a calibrated camera
- We've already established a coordinate frame
- Need to find pose (rotation and translation) of the camera when the image was taken



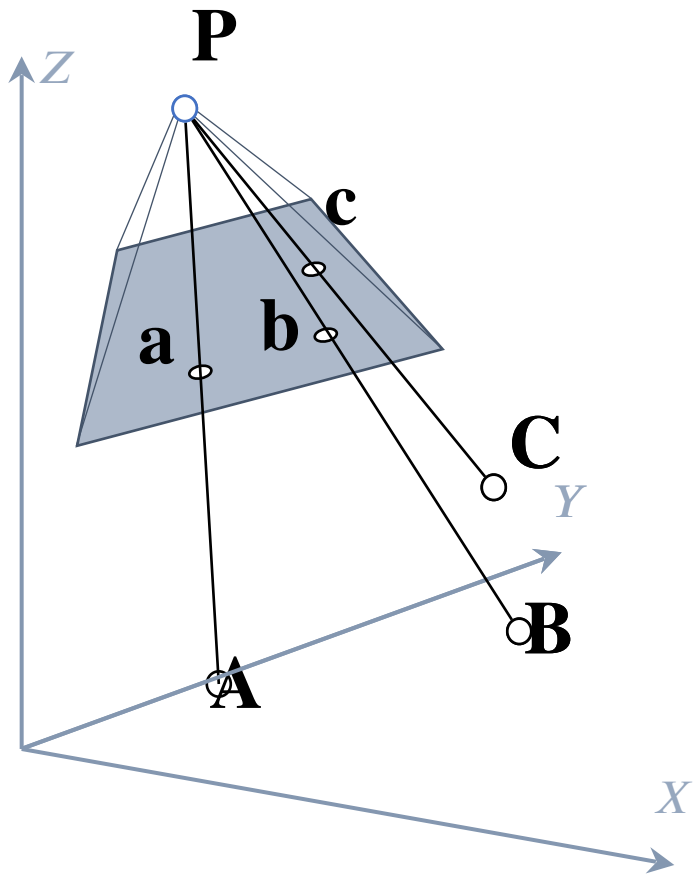
Perspective-n-Point Pose (PnP)

- Pose from 2D-3D matches
  - Have six unknowns ($\mathbf{R},\ \mathbf{t}$)
  - Each 2D-3D match $\mathbf{u}_i \leftrightarrow \mathbf{x}_i$ gives $k_i\mathbf{u}_i = \mathbf{K}[\mathbf{R} \quad \mathbf{t}]\mathbf{x}_i$
- How many matches?
  - Each point gives 3 equations
  - Adds 1 unknown (k_i)
  - $6 + n$ unknowns, $3n$ equations, $\mathbf{n} \geq 3$



Perspective-3-Point Pose (P3P)

- We have three 2D-3D matches
  - Call the 3D points $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$
  - The matching image points are $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$
  - The (unknown) camera location is $\mathbf{P}$
- Overview of solution
  - Use angles between $\vec{\mathbf{PA}}$, $\vec{\mathbf{PB}}$, and $\vec{\mathbf{PC}}$
  - Find the distances $\left|\vec{\mathbf{PA}}\right|$, $\left|\vec{\mathbf{PB}}\right|$, $\left|\vec{\mathbf{PC}}\right|$
  - This gives us equations for $\mathbf{P}$



Perspective-3-Point Pose (P3P)

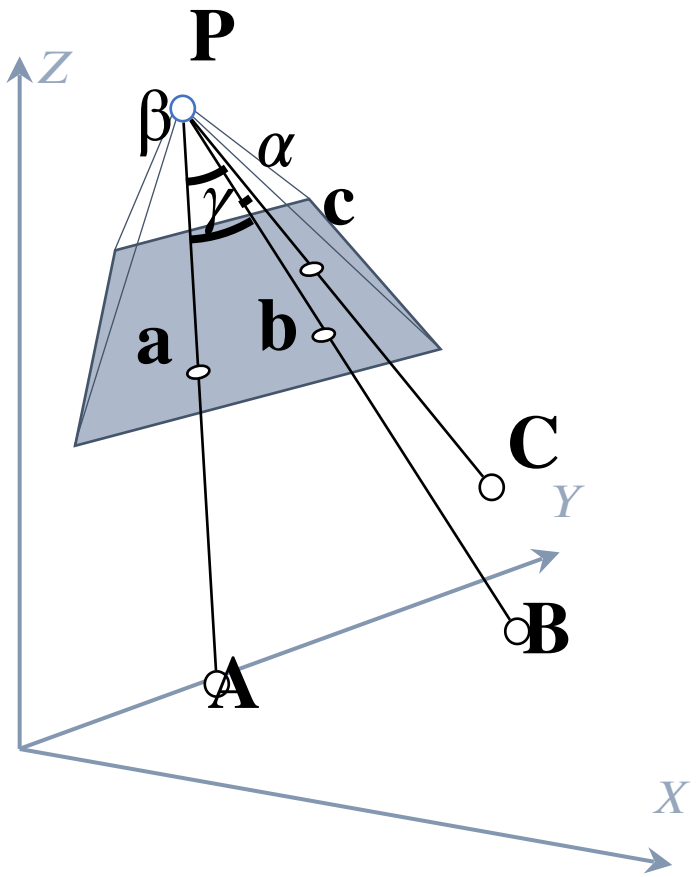
- Define the angles between rays

$$\alpha = \angle BPC \quad \beta = \angle APC \quad \gamma = \angle APB$$

- Using dot products:

$$\cos\alpha = \frac{\mathbf{b} \cdot \mathbf{c}}{|\mathbf{b}|\,|\mathbf{c}|}$$

$$\cos\beta = \frac{\mathbf{a} \cdot \mathbf{c}}{|\mathbf{a}|\,|\mathbf{c}|} \qquad \cos\gamma = \frac{\mathbf{a} \cdot \boldsymbol{b}}{|\mathbf{a}|\,|\mathbf{b}|}$$



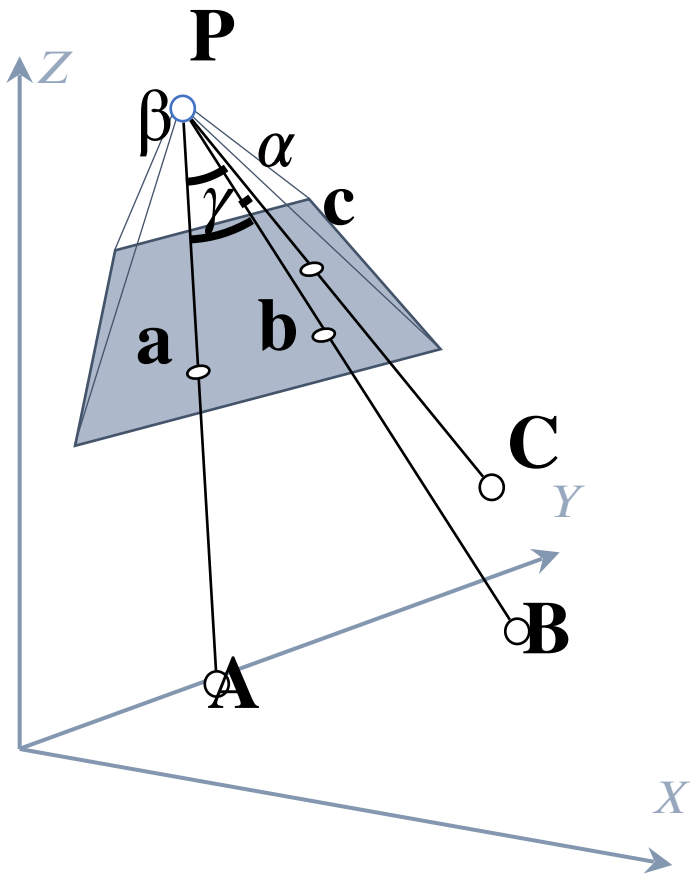
Perspective-3-Point Pose (P3P)

- The law of cosines tells us that

$$\left|\vec{AB}\right|^2 = \left|\vec{PA}\right|^2 + \left|\vec{PB}\right|^2 - 2\left|\vec{PA}\right|\left|\vec{PB}\right|\cos\gamma$$

$$\left|\vec{AC}\right|^2 = \left|\vec{PA}\right|^2 + \left|\vec{PC}\right|^2 - 2\left|\vec{PA}\right|\left|\vec{PC}\right|\cos\beta$$

$$\left|\vec{BC}\right|^2 = \left|\vec{PB}\right|^2 + \left|\vec{PC}\right|^2 - 2\left|\vec{PB}\right|\left|\vec{PC}\right|\cos\alpha$$



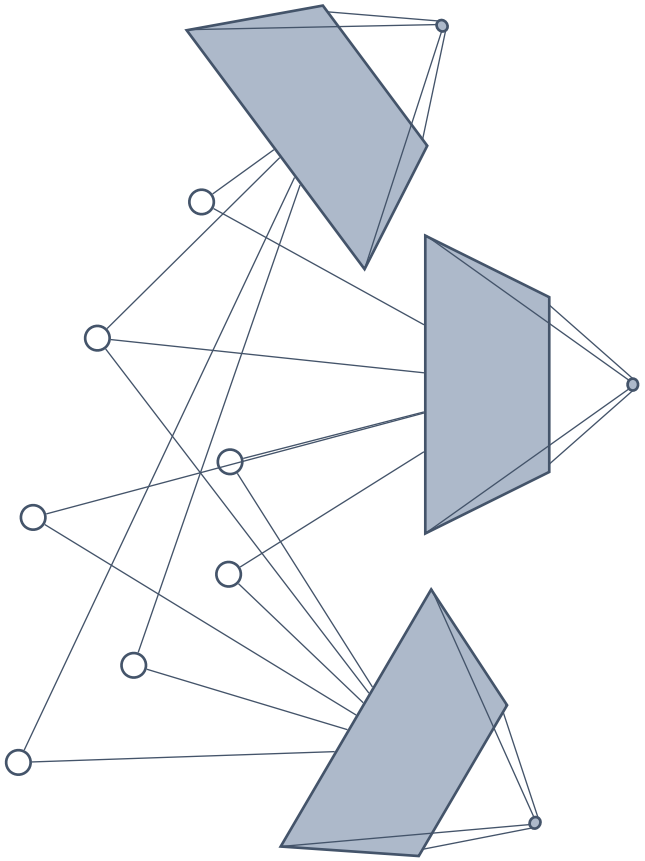
- Solve for

$$\left|\vec{PA}\right|, \left|\vec{PB}\right|, \text{ and } \left|\vec{PC}\right|$$

Bundle Adjustment

Multi-View Stereo Pipeline

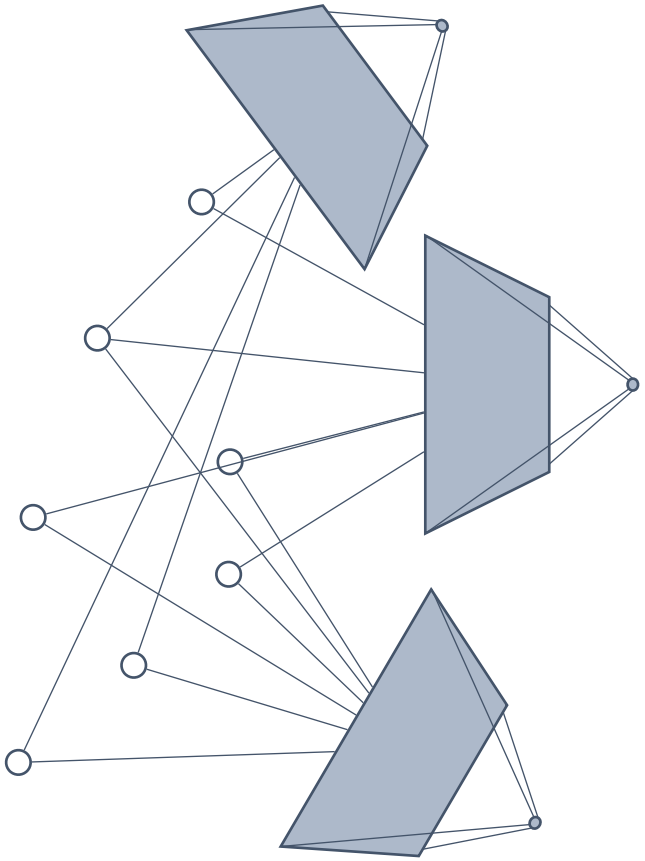
- Identify best matching pair
  - Two-view stereo for relative pose
  - Reconstruct 3D points
  - Minimise reprojection error
- While still images to add
  - Select next best view
  - Determine absolute pose
  - Reconstruct more 3D points
  - *Minimise reprojection error*



Bundle Adjustment

- Minimising reprojection error for multi-view stereo
- Jointly refines:
  - Camera parameters (poses + intrinsics)
  - 3D point positions (scene structure)
- Goal: minimize reprojection error across all images

- Geometry forms **bundles** of rays for each camera
- **Adjusting** these gives an optimal reconstruction solution
- Large sparse non-linear least squares
- Levenberg-Marquardt is commonly used for this

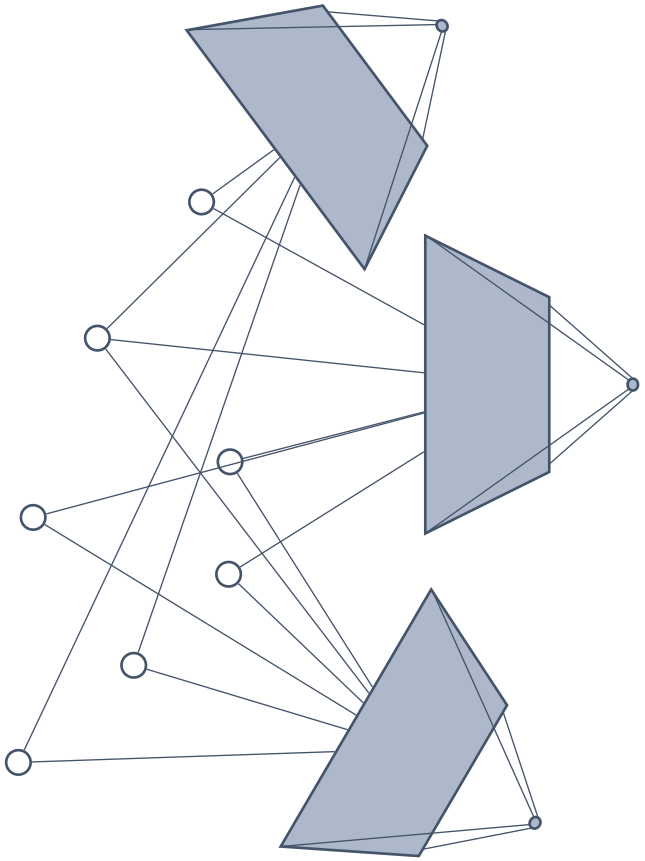


Reprojection Error

- If we have f features in n cameras, reprojection error is:

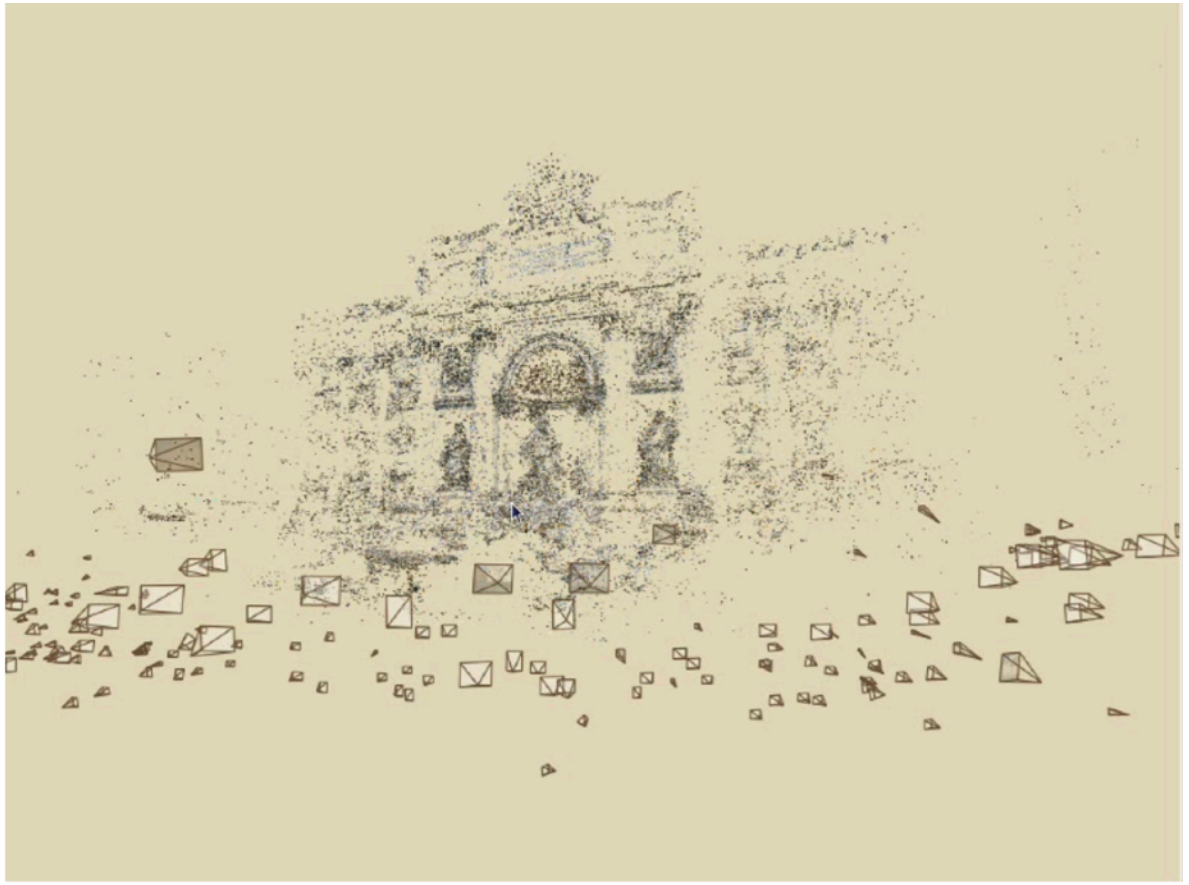
$$\sum_{i=1}^{n}\sum_{j=1}^{f}\left(\left(\mathbf{u}_{i,j}-\mathrm{K}_i\left[\begin{array}{cc}\mathrm{R_i} & \mathbf{t}_i\end{array}\right]\mathbf{x}_j\right)^2\right)$$

Measures Predictions



Bundle Adjustment

- Why is it Needed?
  - Individual pose or triangulation estimates are noisy
  - Errors accumulate across many views
- Bundle adjustment enforces global consistency



The end!