Structure from Motion and SLAM

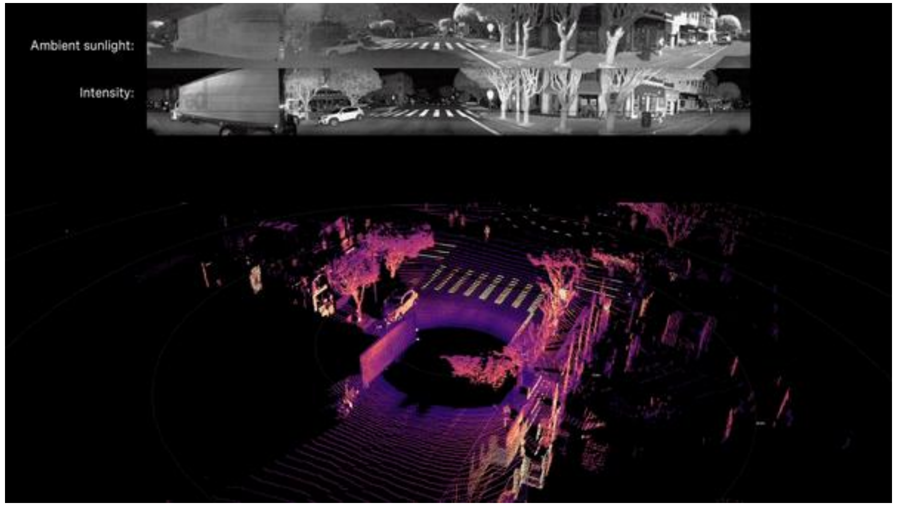
CS 6384 Computer Vision

Professor Yu Xiang

The University of Texas at Dallas

How to Recover the 3D World from Images?

- Structure from Motion (SfM)
  - Structure: the geometry of the 3D world
  - Motion: camera motion
  - Input: a set of images (no need to be videos)
  - From computer vision



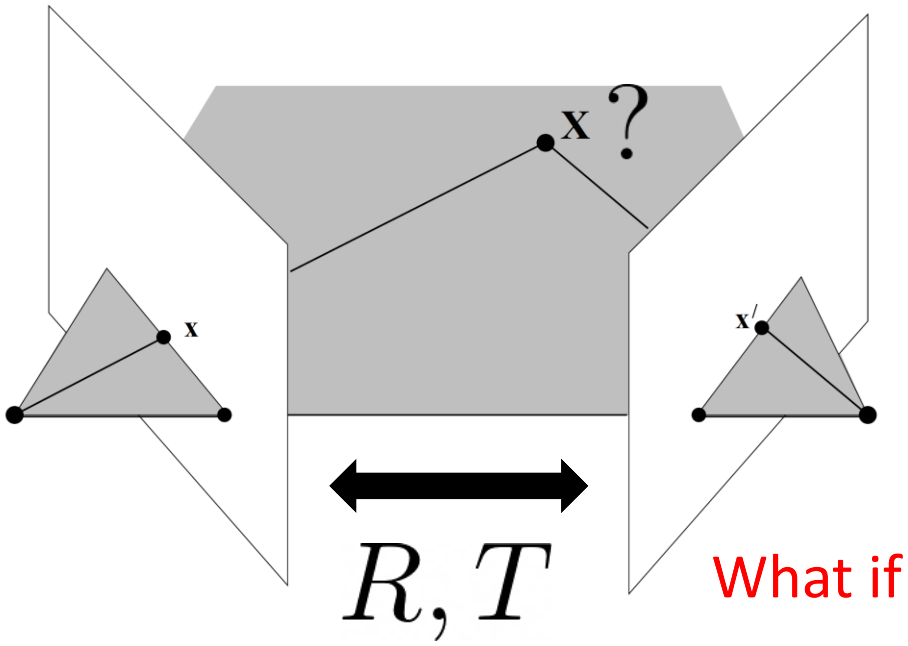
Point cloud captured on an Ouster OS1-128 digital lidar sensor

- Simultaneous Localization and Mapping (SLAM)
  - Localization: camera pose
  - Mapping: build the geometry of the 3D world
  - Input: video sequences
  - From robotics

Triangulation

- Idea: using images from different views and feature matching

- Triangulation from pixel correspondences to compute 3D location



Given $\mathbf{x} \longleftrightarrow \mathbf{x}'$

Intersection of two backprojected lines

$$\mathbf{X} = \mathbf{l} \times \mathbf{l}'$$

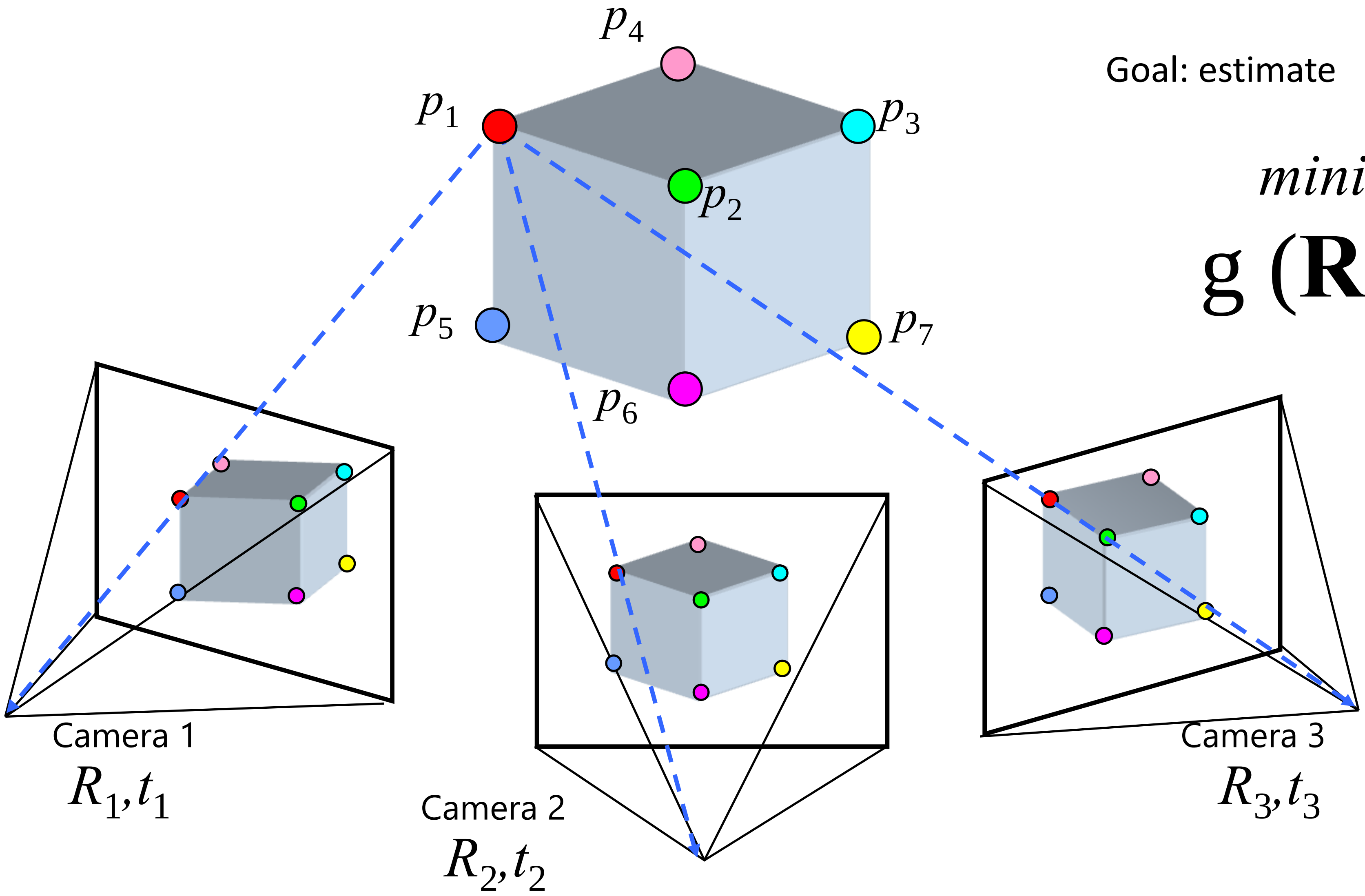
What if unknow camera pose?

Structure from Motion

- Input
  - A set of images from different views

- Output
  - 3D Locations of all feature points in a world frame
  - Camera poses of the images

Structure from motion



Goal: estimate $\mathbf{R}, \mathbf{T}, \mathbf{P}$

minimize

$$g\,(\mathbf{R}, \mathbf{T}, \mathbf{P})$$

Structure from Motion

- Minimize sum of squared reprojection errors

$$g(\mathbf{X}, \mathbf{R}, \mathbf{T}) = \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} \cdot \left\| \underbrace{\mathbf{P}(\mathbf{x}_i, \mathbf{R}_j, \mathbf{t}_j)}_{\substack{\textit{predicted} \\ \text{image location}}} - \underbrace{\begin{bmatrix} u_{i,j} \\ v_{i,j} \end{bmatrix}}_{\substack{\textit{observed} \\ \text{image location}}} \right\|^2$$

m points, n images

Indicator variable:
is point i visible in image j?

Projection

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \mathbf{R}\mathbf{x} + \mathbf{t}$$

$$u' = f_x \frac{x'}{z'} + p_x$$

$$v' = f_y \frac{y'}{z'} + p_y$$

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = \mathbf{P}(\mathbf{x}, \mathbf{R}, \mathbf{t})$$

Structure from Motion

- How to minimize

$$g(\mathbf{X}, \mathbf{R}, \mathbf{T}) = \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} \cdot \left\| \mathbf{P}(\mathbf{x}_i, \mathbf{R}_j, \mathbf{t}_j) - \begin{bmatrix} u_{i,j} \\ v_{i,j} \end{bmatrix} \right\|^2$$

- A non-linear least squares problem (why?)
  - E.g. Levenberg-Marquardt

The Levenberg-Marquardt Algorithm

- Nonlinear least squares $\hat{\boldsymbol{\beta}} \in \operatorname{argmin}_{\boldsymbol{\beta}} S(\boldsymbol{\beta}) \equiv \operatorname{argmin}_{\boldsymbol{\beta}} \sum_{i=1}^{m} \left[y_i - f(x_i, \boldsymbol{\beta})\right]^2$

  $n \times 1$

- An iterative algorithm
  - Start with an initial guess β_0
  - For each iteration $\beta \leftarrow \beta + \delta$

- How to get δ?
  - Linear approximation $f(x_i, \boldsymbol{\beta} + \boldsymbol{\delta}) \approx f(x_i, \boldsymbol{\beta}) + \mathbf{J}_i \boldsymbol{\delta}$ $\qquad \mathbf{J}_i = \frac{\partial f(x_i, \boldsymbol{\beta})}{\partial \boldsymbol{\beta}}$ $\quad 1 \times n$
  - Find δ to minimize the objective $S(\boldsymbol{\beta} + \boldsymbol{\delta}) \approx \sum_{i=1}^{m} \left[y_i - f(x_i, \boldsymbol{\beta}) - \mathbf{J}_i \boldsymbol{\delta}\right]^2$

Best to minimize the objective

Wikipedia

The Levenberg-Marquardt Algorithm

- Vector notation for $S(\boldsymbol\beta+\boldsymbol\delta) \approx \sum_{i=1}^{m}\left[y_i - f(x_i, \boldsymbol\beta) - \mathbf{J}_i\boldsymbol\delta\right]^2$

$$
\begin{aligned}
S(\boldsymbol\beta+\boldsymbol\delta) &\approx \|\mathbf{y} - \mathbf{f}(\boldsymbol\beta) - \mathbf{J}\boldsymbol\delta\|^2 \\
&= \left[\mathbf{y} - \mathbf{f}(\boldsymbol\beta) - \mathbf{J}\boldsymbol\delta\right]^{\mathrm{T}}\left[\mathbf{y} - \mathbf{f}(\boldsymbol\beta) - \mathbf{J}\boldsymbol\delta\right] \\
&= \left[\mathbf{y} - \mathbf{f}(\boldsymbol\beta)\right]^{\mathrm{T}}\left[\mathbf{y} - \mathbf{f}(\boldsymbol\beta)\right] - \left[\mathbf{y} - \mathbf{f}(\boldsymbol\beta)\right]^{\mathrm{T}}\mathbf{J}\boldsymbol\delta - (\mathbf{J}\boldsymbol\delta)^{\mathrm{T}}\left[\mathbf{y} - \mathbf{f}(\boldsymbol\beta)\right] + \boldsymbol\delta^{\mathrm{T}}\mathbf{J}^{\mathrm{T}}\mathbf{J}\boldsymbol\delta \\
&= \left[\mathbf{y} - \mathbf{f}(\boldsymbol\beta)\right]^{\mathrm{T}}\left[\mathbf{y} - \mathbf{f}(\boldsymbol\beta)\right] - 2\left[\mathbf{y} - \mathbf{f}(\boldsymbol\beta)\right]^{\mathrm{T}}\mathbf{J}\boldsymbol\delta + \boldsymbol\delta^{\mathrm{T}}\mathbf{J}^{\mathrm{T}}\mathbf{J}\boldsymbol\delta.
\end{aligned}
$$

Take derivation with respect to δ and set to zero $\left(\mathbf{J}^{\mathrm{T}}\mathbf{J}\right)\boldsymbol\delta = \mathbf{J}^{\mathrm{T}}\left[\mathbf{y} - \mathbf{f}(\boldsymbol\beta)\right]$

https://www.cs.ubc.ca/~schmidtm/Courses/340-F16/linearQuadraticGradients.pdf

Levenberg's contribution $\left(\mathbf{J}^{\mathrm{T}}\mathbf{J} + \lambda\mathbf{I}\right)\boldsymbol\delta = \mathbf{J}^{\mathrm{T}}\left[\mathbf{y} - \mathbf{f}(\boldsymbol\beta)\right]$ damped version

$$\beta \leftarrow \beta + \delta$$

Wikipedia

Structure from Motion

$$g(\mathbf{X}, \mathbf{R}, \mathbf{T}) = \sum_{i=1}^{m} \sum_{j=1}^{n} \underbrace{w_{ij}}_{\substack{\textit{indicator variable}: \\ \text{is point } i \text{ visible in image } j\text{ ?}}} \cdot \left\| \underbrace{\mathbf{P}(\mathbf{x}_i, \mathbf{R}_j, \mathbf{t}_j)}_{\substack{\textit{predicted} \\ \text{image location}}} - \underbrace{\begin{bmatrix} u_{i,j} \\ v_{i,j} \end{bmatrix}}_{\substack{\textit{observed} \\ \text{image location}}} \right\|^2$$

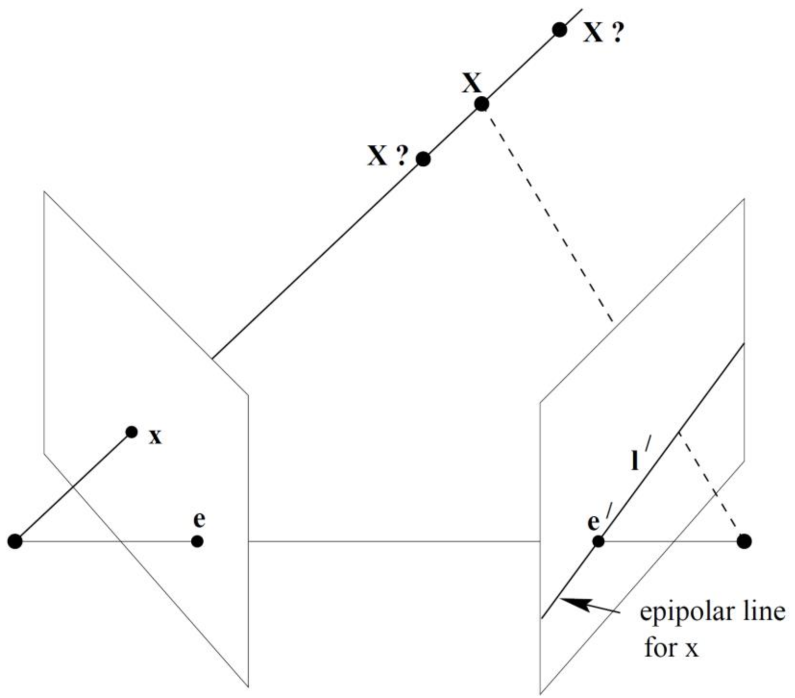
$$\beta = (\mathbf{X}, \mathbf{R}, \mathbf{T})$$

How to get the initial estimation β_0 ?

Random guess is not a good idea.

Matching Two Views

- Fundamental matrix



$\mathbf{x}'$ is on the epiploar line $\mathbf{l}' = F\mathbf{x}$

$$\mathbf{x}'^T F \mathbf{x} = 0$$

The 8-point algorithm

Matching Two Views

$$\mathbf{x}'^T F \mathbf{x} = 0$$

If we know camera intrinsics in SfM

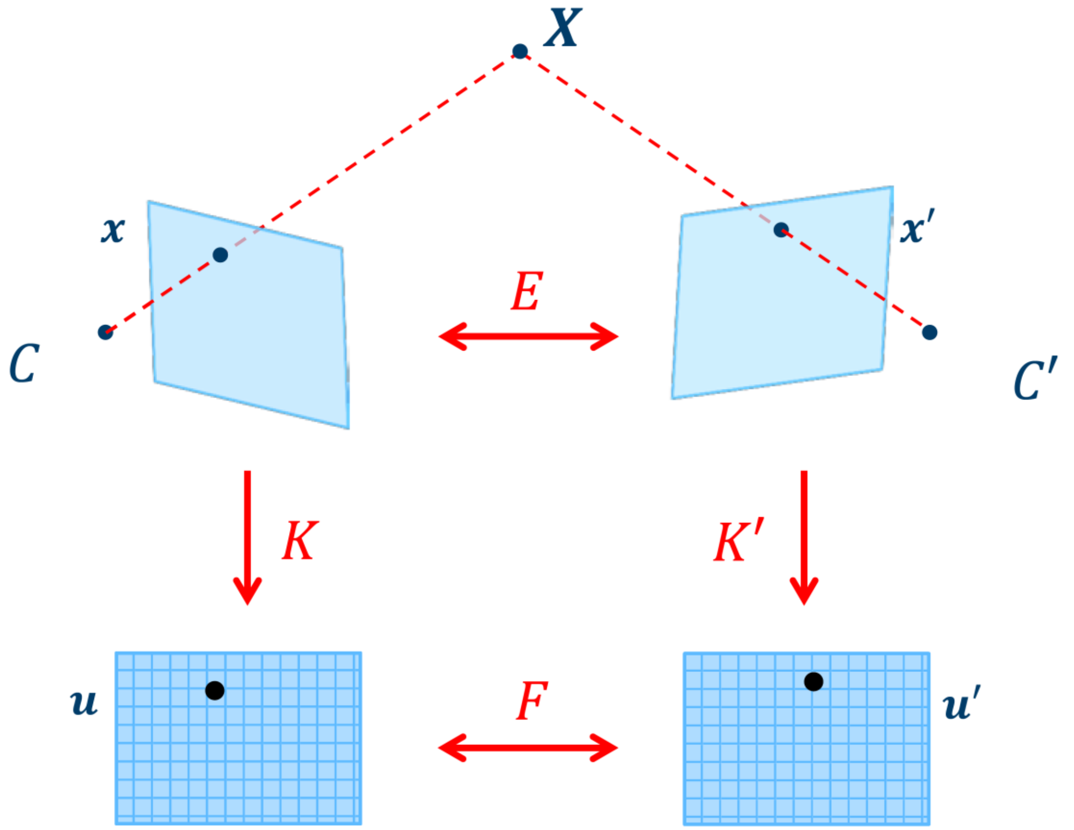
$$(K'^{-1}\mathbf{x}')^T E (K^{-1}\mathbf{x}) = 0$$

Normalized coordinates

$$F = K'^{-T} E K^{-1}$$

- Essential matrix E

$$E = K'^T F K$$



Credit: Thomas Opsahl

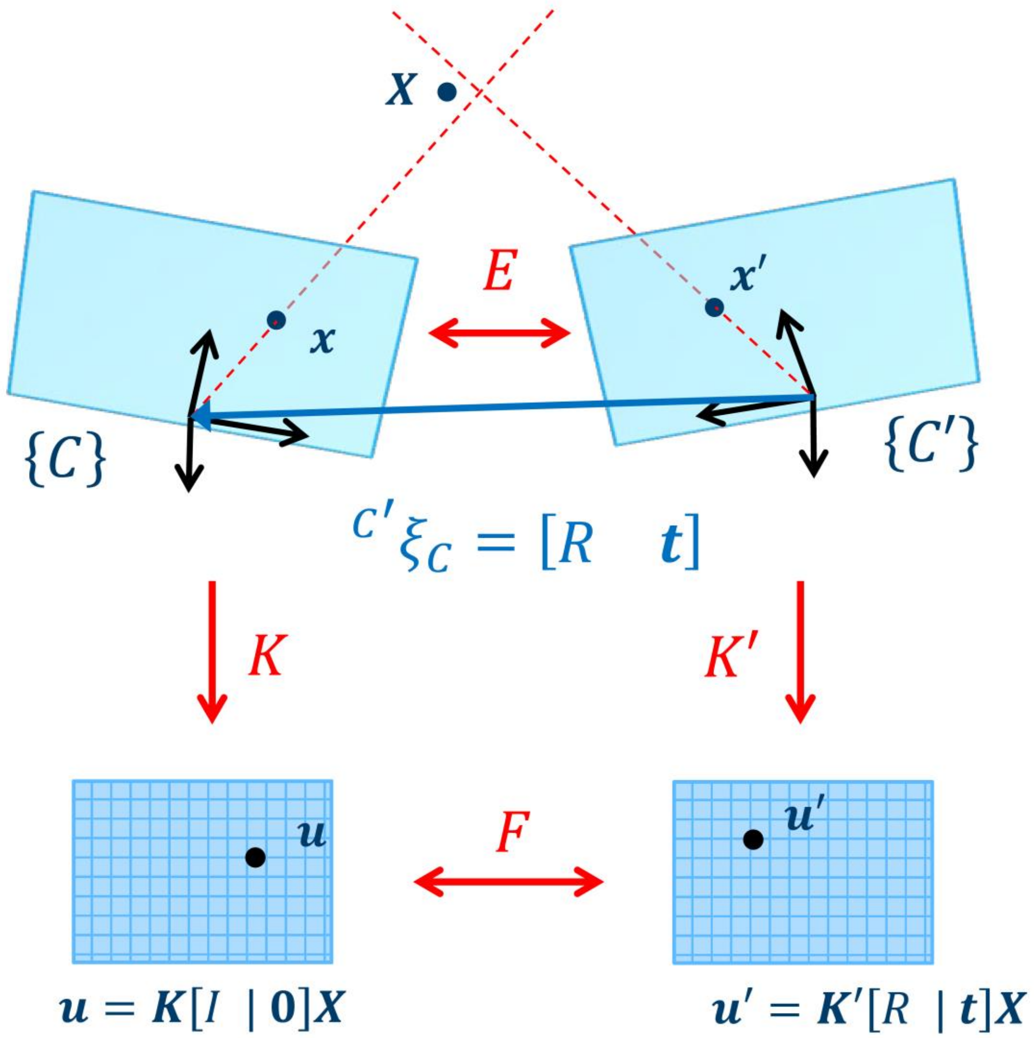
Matching Two Views

- Recover the relative pose R and $\boldsymbol{t}$ from the essential matrix E up to the scale of $\boldsymbol{t}$

$$\mathrm{F} = [\mathbf{e}']_\times \mathrm{K}'\mathrm{R}\mathrm{K}^{-1} = \mathrm{K}'^{-\mathsf{T}}[\mathbf{t}]_\times \mathrm{R}\mathrm{K}^{-1}$$

$$E = K'^T F K$$

$$\mathrm{E} = [\mathbf{t}]_\times \mathrm{R}$$



Credit: Thomas Opsahl

H. C Longuet-Higgins, *A computer algorithm for reconstructing a scene from two projections*, 1981

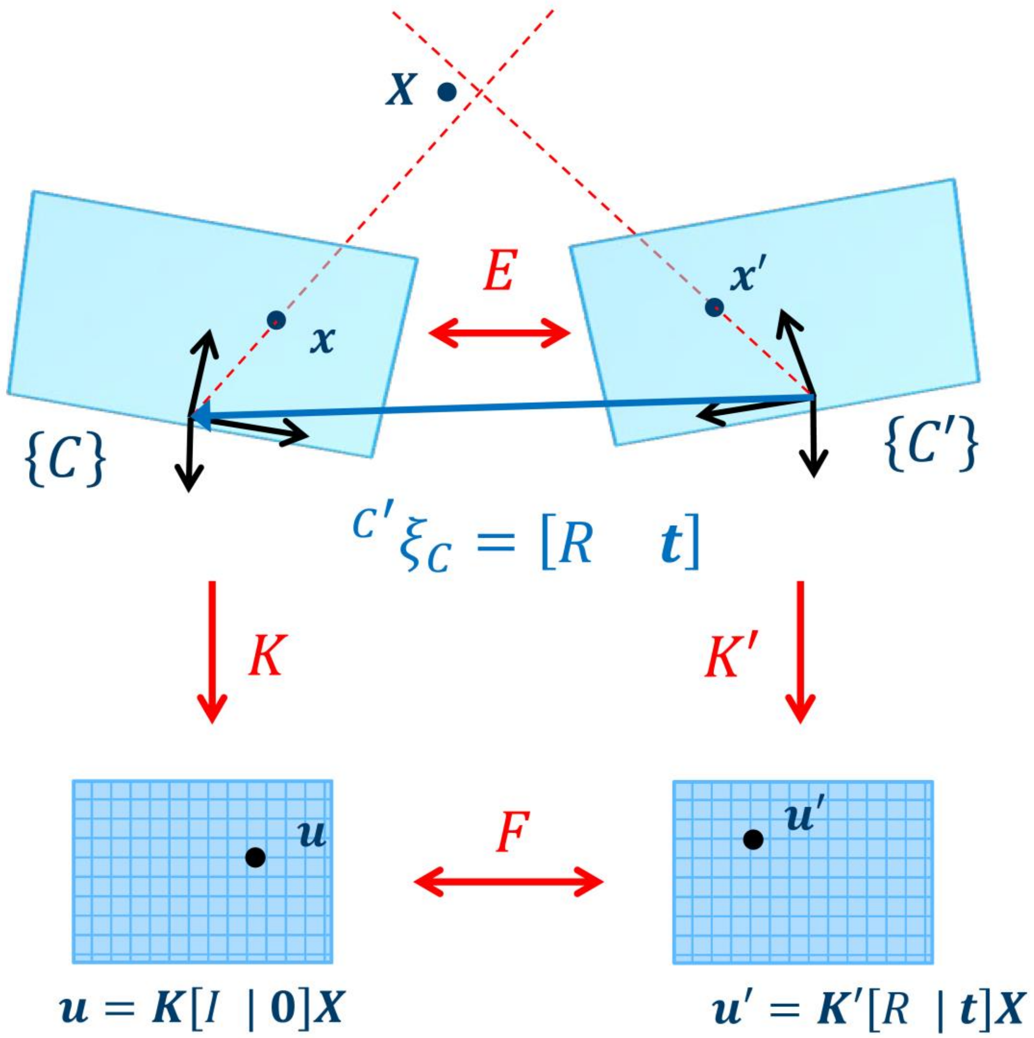
Matching Two Views

$$\mathbf{E} = [\mathbf{t}]_\times \mathbf{R}$$

$$E \cdot \mathbf{t} = [\mathbf{t}]_\times R \cdot \mathbf{t}$$
$$= (\mathbf{t} \times R) \cdot \mathbf{t} = 0$$

Use SVD to solve for **t**

$$R = -[\mathbf{t}]_\times E$$



Credit: Thomas Opsahl

H. C Longuet-Higgins, *A computer algorithm for reconstructing a scene from two projections*, 1981

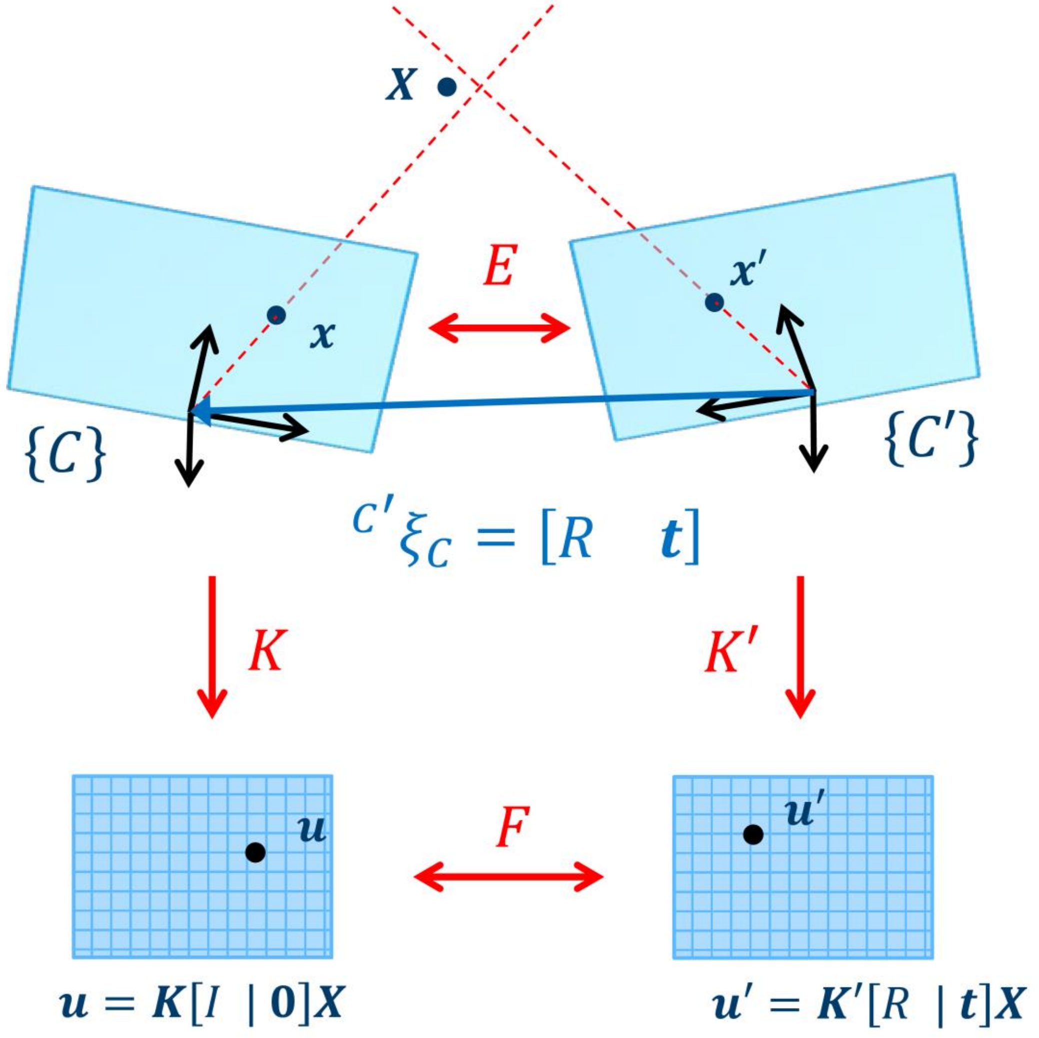
Matching Two Views

- If we do not know the camera intrinsics
- Work with projection matrix

$$P = [I|\mathbf{0}] \qquad P' = [A|\mathbf{b}]$$

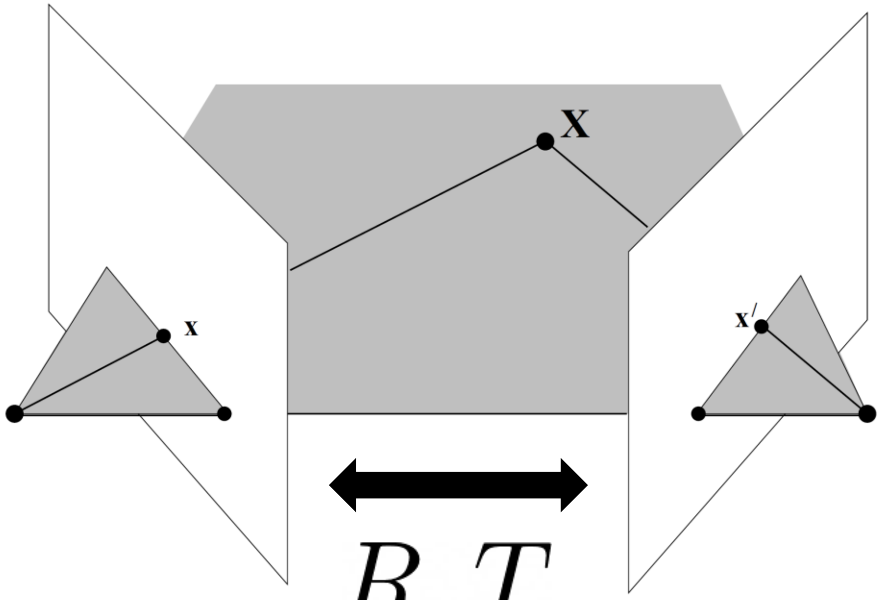
$$\mathbf{x}'^T F \mathbf{x} = 0$$

$$F = [\mathbf{b}]_\times A$$



Credit: Thomas Opsahl

Triangulation



$$R, T$$

Estimated from essential matrix E

Intersection of two backprojected lines

$$\mathbf{X} = \mathbf{l} \times \mathbf{l}'$$

How to get the initial estimation β_0 ?

$$\beta = (\mathbf{X}, \mathbf{R}, \mathbf{T})$$

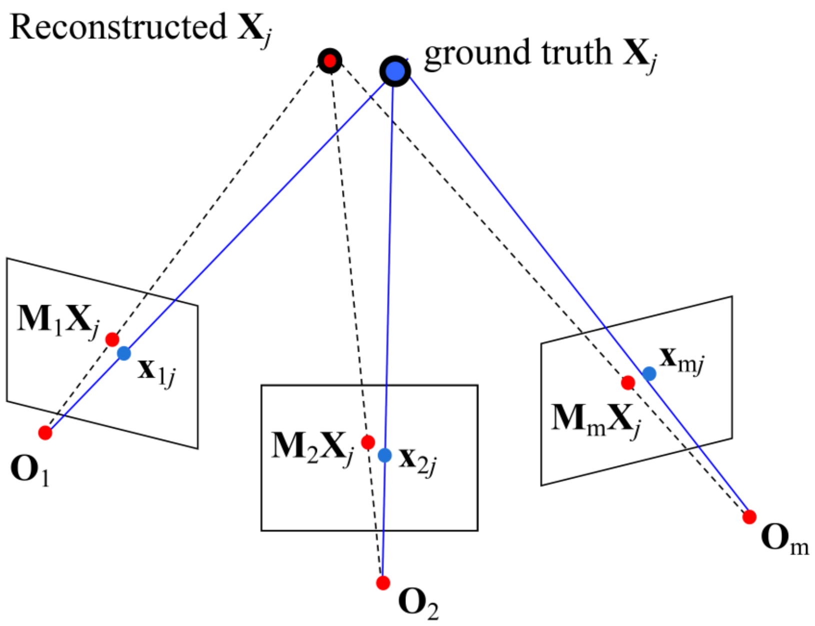
Structure from Motion

- Bundle adjustment
  - Iteratively refinement of structure (3D points) and motion (camera poses)

$$g(\mathbf{X},\mathbf{R},\mathbf{T}) = \sum_{i=1}^{m}\sum_{j=1}^{n} \underbrace{w_{ij}}_{\substack{\textit{indicator variable}: \\ \text{is point } i \text{ visible in image } j\,?}} \cdot \left\| \underbrace{\mathbf{P}(\mathbf{x}_i,\mathbf{R}_j,\mathbf{t}_j)}_{\substack{\textit{predicted} \\ \text{image location}}} - \underbrace{\begin{bmatrix} u_{i,j} \\ v_{i,j} \end{bmatrix}}_{\substack{\textit{observed} \\ \text{image location}}} \right\|^2$$

  - Levenberg-Marquardt algorithm

$$\beta \leftarrow \beta + \delta$$



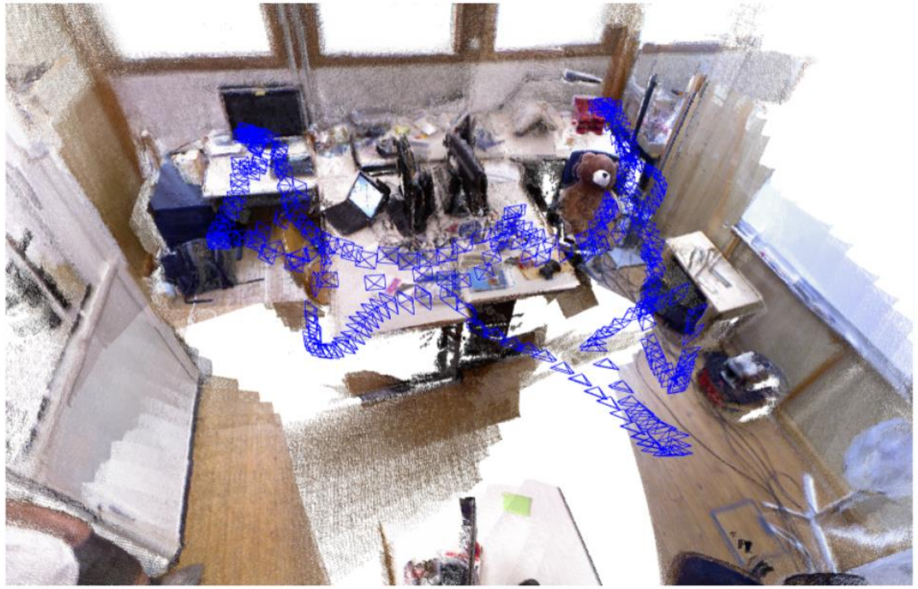
Examples: http://vision.soic.indiana.edu/projects/disco/

Build Rome in One Day

https://grail.cs.washington.edu/rome/

Simultaneous Localization and Mapping (SLAM)

- Localization: camera pose tracking
- Mapping: building a 2D or 3D representation of the environment
- The goal here is the same as structure from motion but with video input



ORB-SLAM2
- Point cloud and camera poses

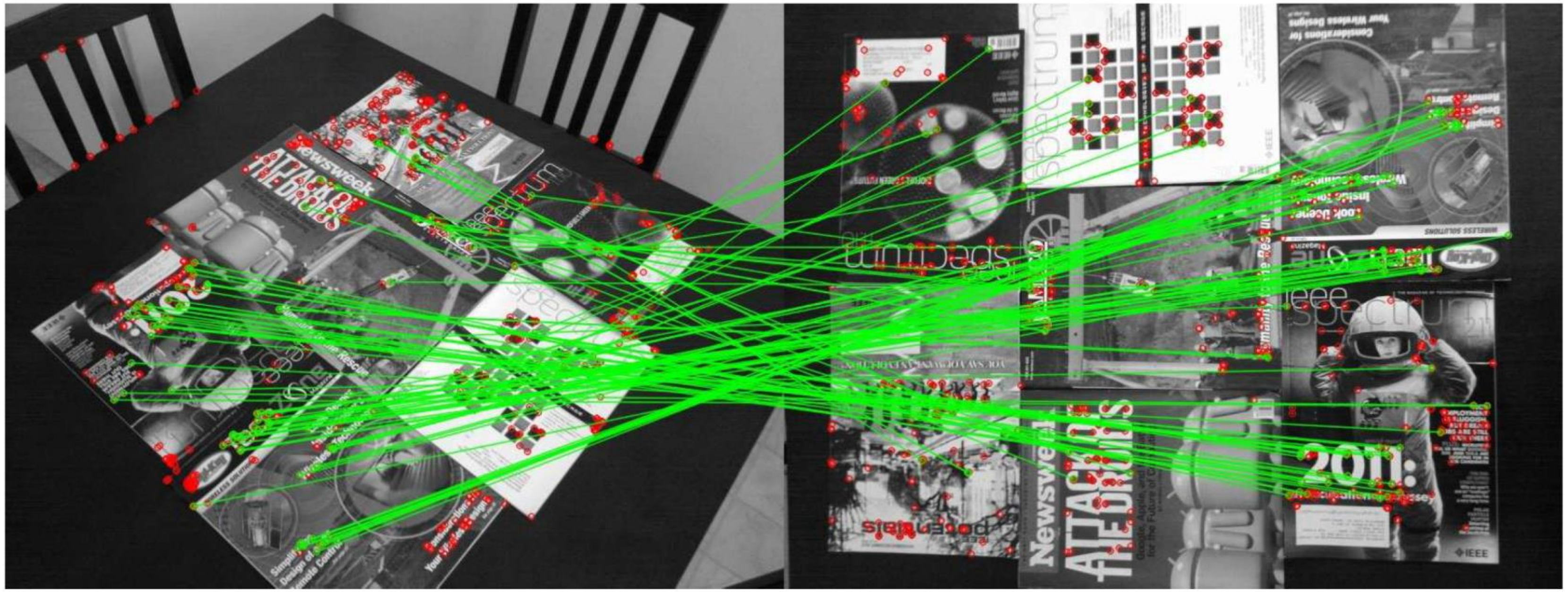
Case Study: ORB-SLAM

- Oriented FAST and Rotated BRIEF (ORB)
- Tracking camera poses
  - Motion only Bundle Adjustment (BA)
- Mapping
  - Local BA around camera pose
- Loop closing
  - Loop detection

https://webdiis.unizar.es/~raulmur/orbslam/

Case Study: ORB-SLAM

- Feature descriptors: Oriented FAST and Rotated BRIEF (ORB)
  - Similar matching performance as SIFT
  - Real-time computation without GPUs



ORB: an efficient alternative to SIFT or SURF. Rublee et al. ICCV’11.

Case Study: ORB-SLAM

- Tracking camera poses
  - Motion only Bundle Adjustment (BA)
    - Huber cost function and covariance matrix associated to the scale of the keypoint

$$\{\mathbf{R}, \mathbf{t}\} = \underset{\mathbf{R}, \mathbf{t}}{\operatorname{argmin}} \sum_{i \in \mathcal{X}} \rho \left(\left\| \mathbf{x}^i_{(\cdot)} - \pi_{(\cdot)} \left(\mathbf{R} \mathbf{X}^i + \mathbf{t} \right) \right\|^2_{\Sigma} \right)$$

Camera pose

Detected Keypoint

3D point in the map (world coordinates)

Levenberg–Marquardt method

$$L_\delta(a) = \begin{cases} \frac{1}{2} a^2 & \text{for } |a| \le \delta, \\ \delta(|a| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases}$$

Huber loss function

Case Study: ORB-SLAM

- Mapping
  - Local BA around the estimated camera pose
  - Refine 3D point locations

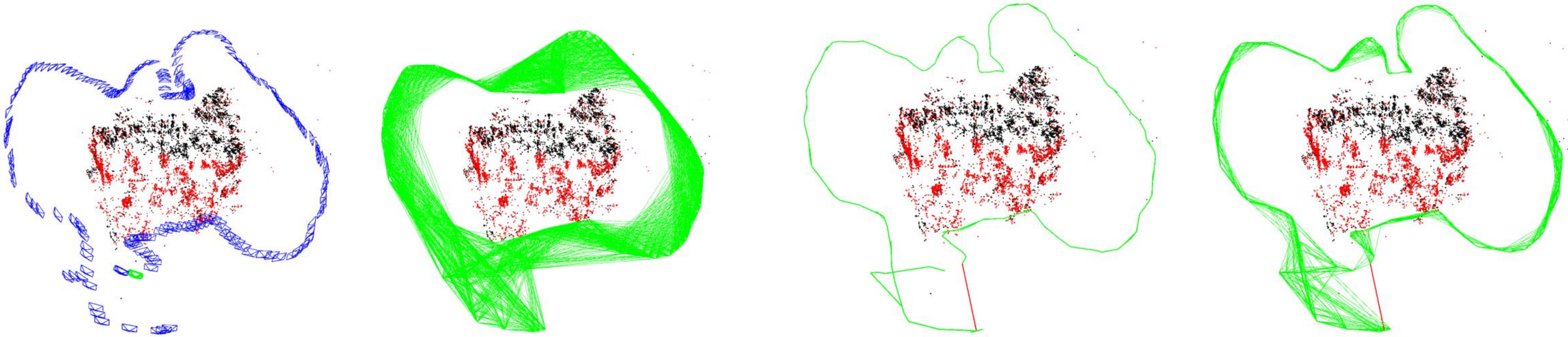
3D point Keyframe

$$\{\mathbf{X}^i, \mathbf{R}_l, \mathbf{t}_l | i \in \mathcal{P}_L, l \in \mathcal{K}_L\} = \underset{\mathbf{X}^i, \mathbf{R}_l, \mathbf{t}_l}{\operatorname{argmin}} \sum_{k \in \mathcal{K}_L \cup \mathcal{K}_F} \sum_{j \in \mathcal{X}_k} \rho\left(E_{kj}\right)$$

$$E_{kj} = \left\| \mathbf{x}^j_{(\cdot)} - \pi_{(\cdot)}\left(\mathbf{R}_k \mathbf{X}^j + \mathbf{t}_k\right) \right\|^2_{\Sigma}$$

Case Study: ORB-SLAM

- Loop closing and full BA



(a) KeyFrames (blue), Current Camera (green), MapPoints (black, red), Current Local MapPoints (red)

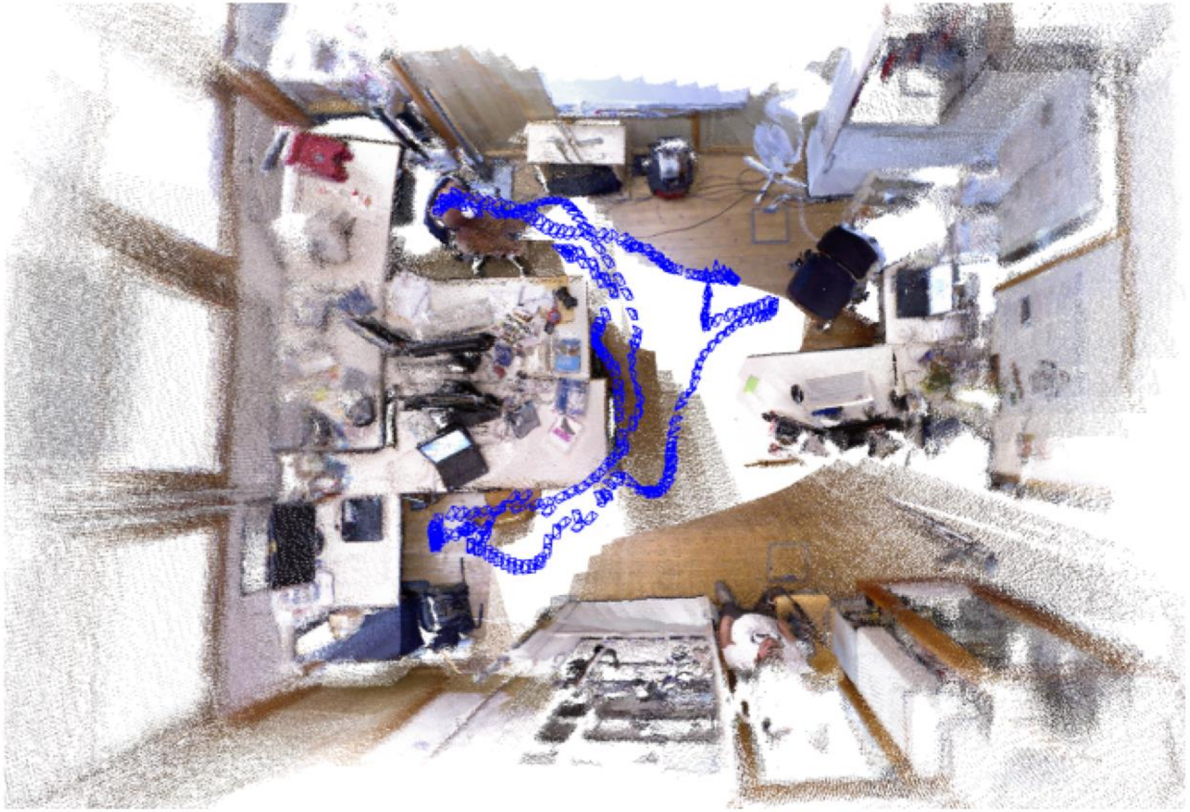
(b) Covisibility Graph

(c) Spanning Tree (green) and Loop Closure (red)

(d) Essential Graph

Edges from the covisibility graph with high covisibility

Case Study: ORB-SLAM



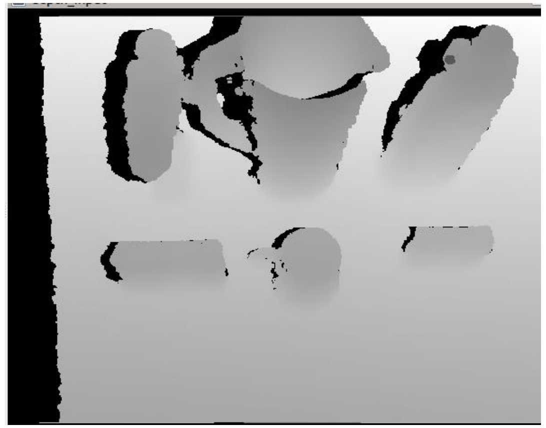
RGB-D SLAM

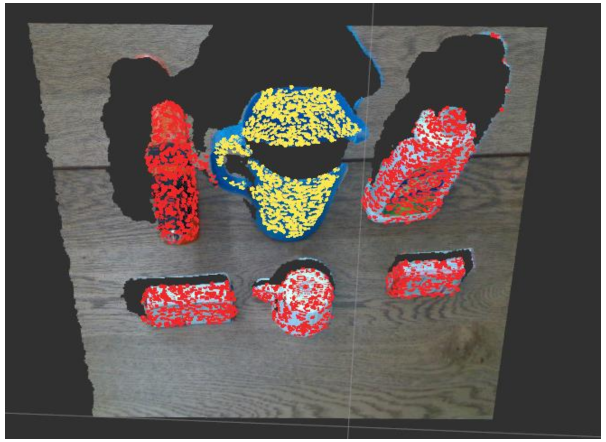
- RGB-D cameras

Microsoft Kinect

Intel RealSense

- Using depth images: 3D points in the camera frame





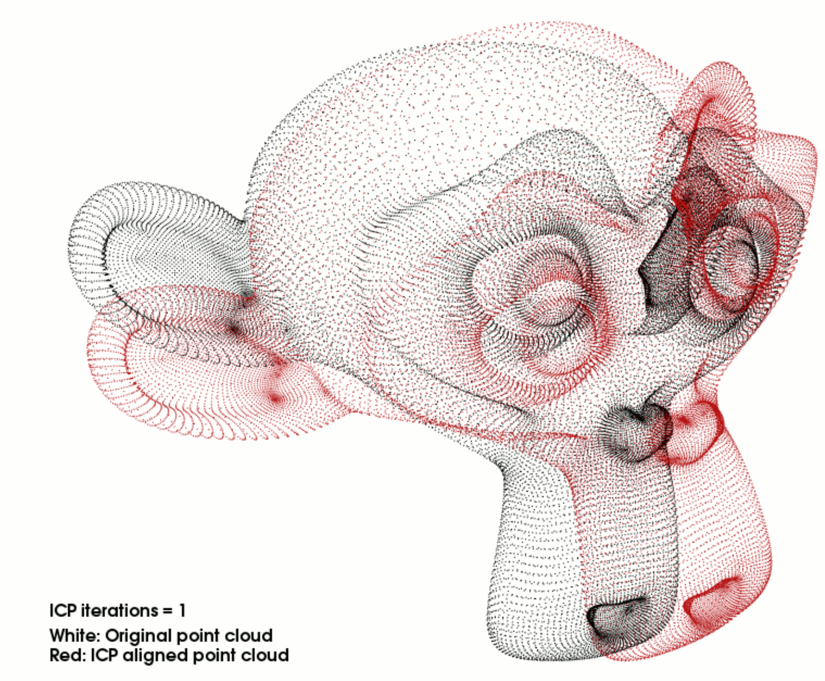
Point Cloud

RGB-D SLAM

- Camera pose tracking
  - Iterative closest point (ICP) algorithm

Input: source point cloud, target point cloud
Output: rigid transformation from source to target

- For i in range(N)
  - For each point in the source, find the closest point in the target (correspondences)
  - Estimation R and T using the correspondences
  - Transform the source points using R and T

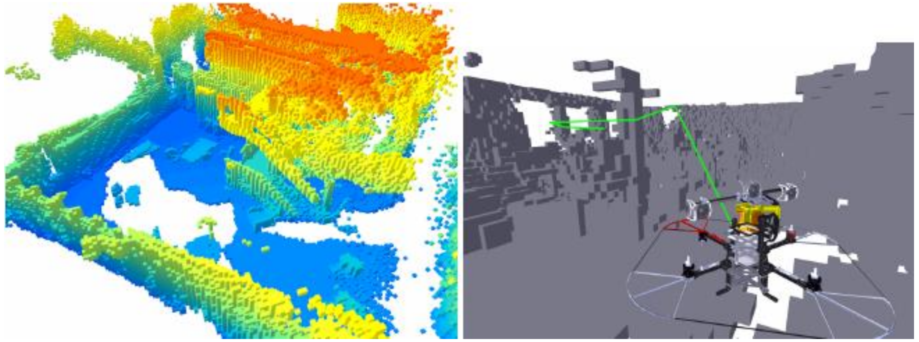


RGB-D SLAM

- Mapping: fuse point clouds into a global frame
- Map representation

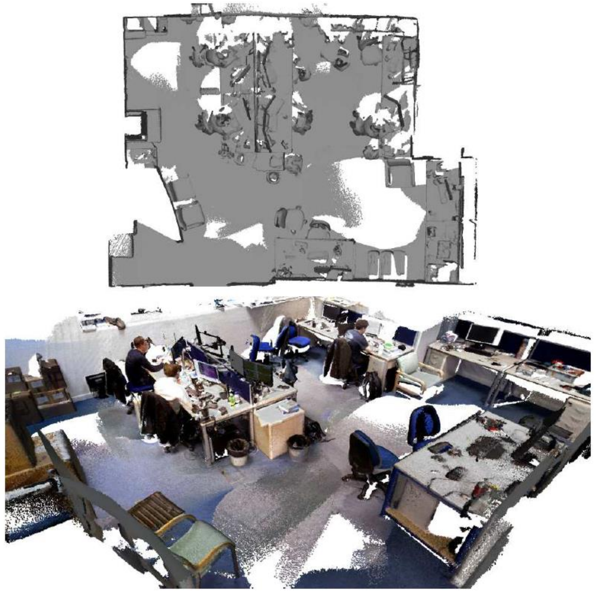
Point clouds

ORB-SLAM



Voxels

Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera. Huang, et al. 2011



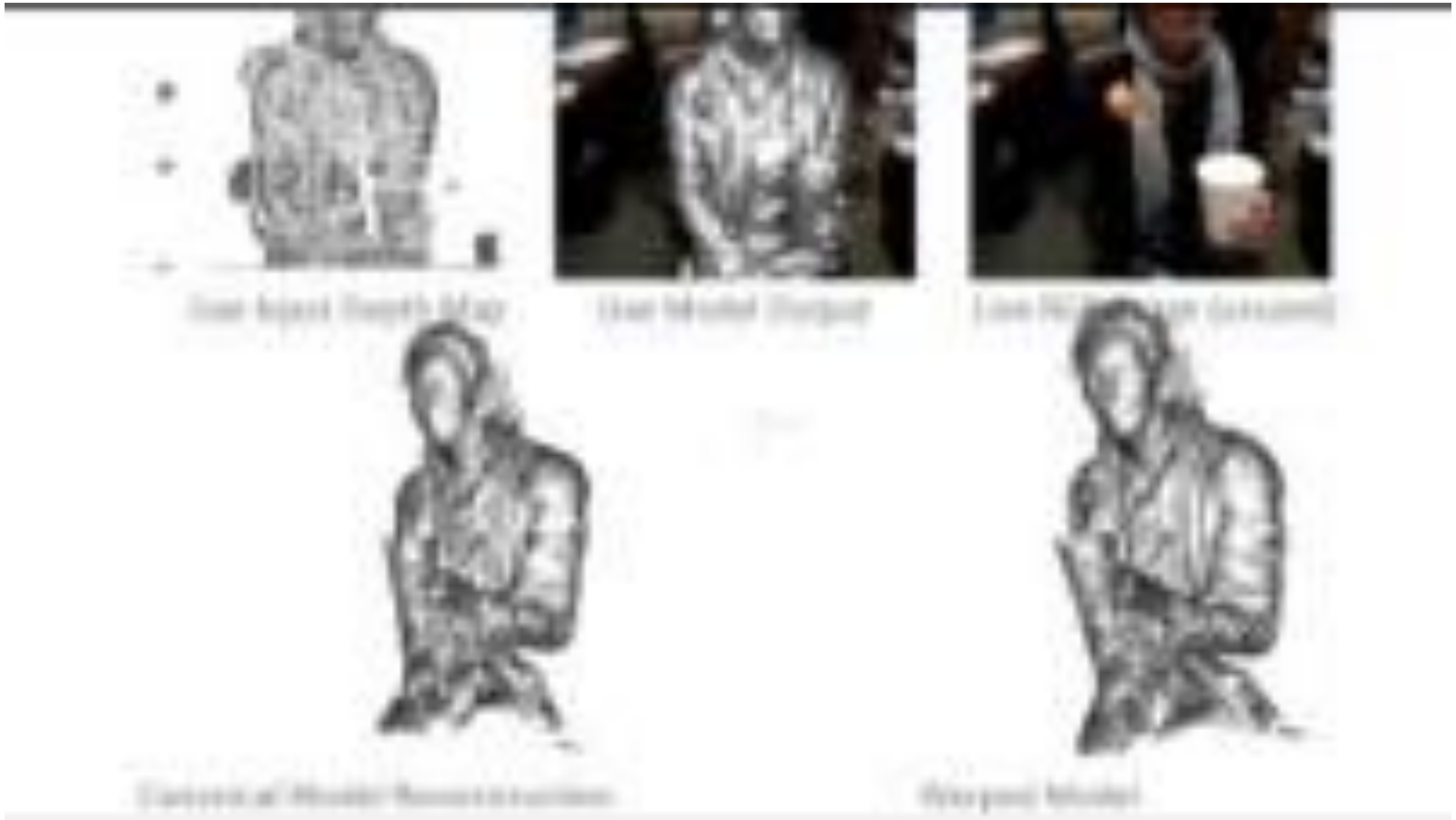
Surfels (small 3D surface)

ElasticFusion

KinectFusion

https://youtu.be/of6d7C_ZWwc

DynamicFusion



A volumetric flow field that transforms the state of the scene at each time instant into a fixed, canonical frame.

https://youtu.be/i1eZekcc_lM

DynamicFusion: Reconstruction and Tracking of Non-rigid Scenes in Real-Time. Newcombe, Fox, Seitz, CVPR'15.

Further Reading

- Chapter 11, Computer Vision, Richard Szeliski
- KinectFusion: Real-Time Dense Surface Mapping and Tracking. Newcombe et al., ISMAR'11
- ORB-SLAM https://webdiis.unizar.es/~raulmur/orbslam/