THE UNIVERSITY OF TEXAS AT DALLAS

Structure from Motion and SLAM

CS 4391 Introduction to Computer Vision

Professor Yapeng Tian

Department of Computer Science

Slides borrowed from Professor Yu Xiang

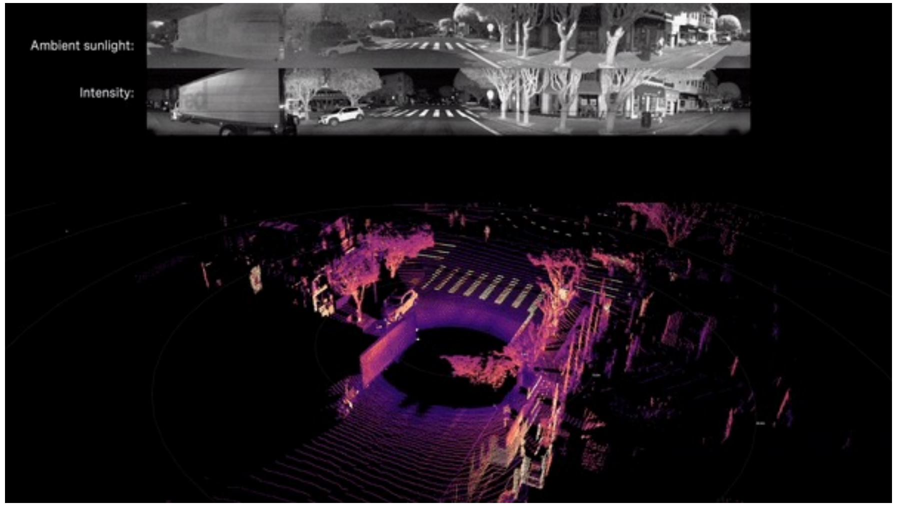
How to Recover the 3D World from Images?

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

Simultaneous Localization and Mapping (SLAM)

- Localization: camera pose
- Mapping: build the geometry of the 3D world
- Input: video sequences
- From robotics

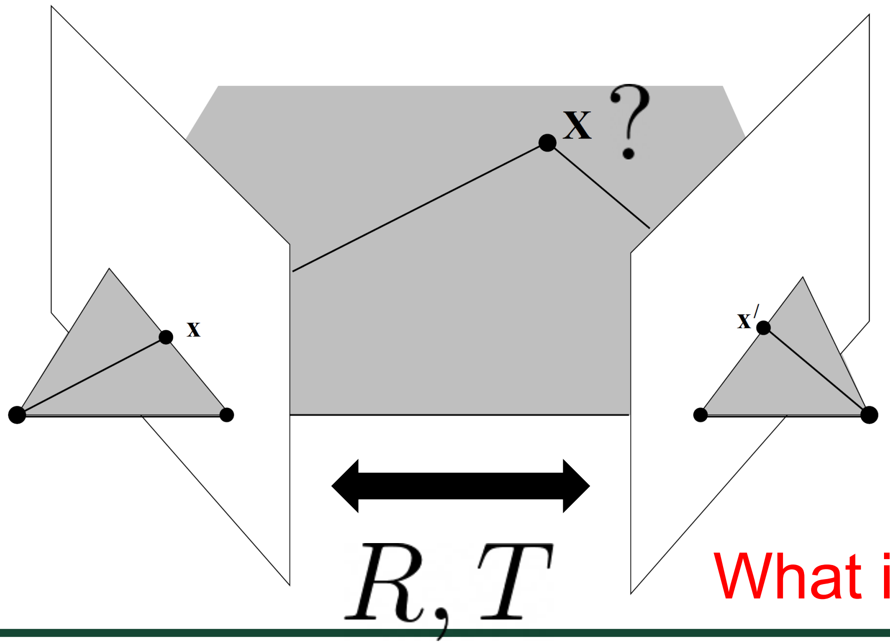


Point cloud captured on an Ouster OS1-128 digital lidar sensor

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?

Colosseum in Rome

Structure from Motion

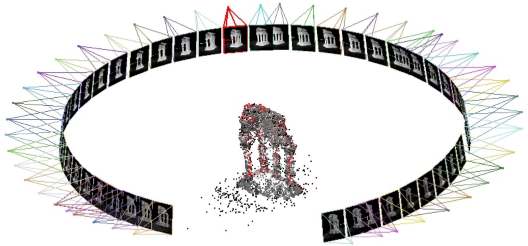
Input

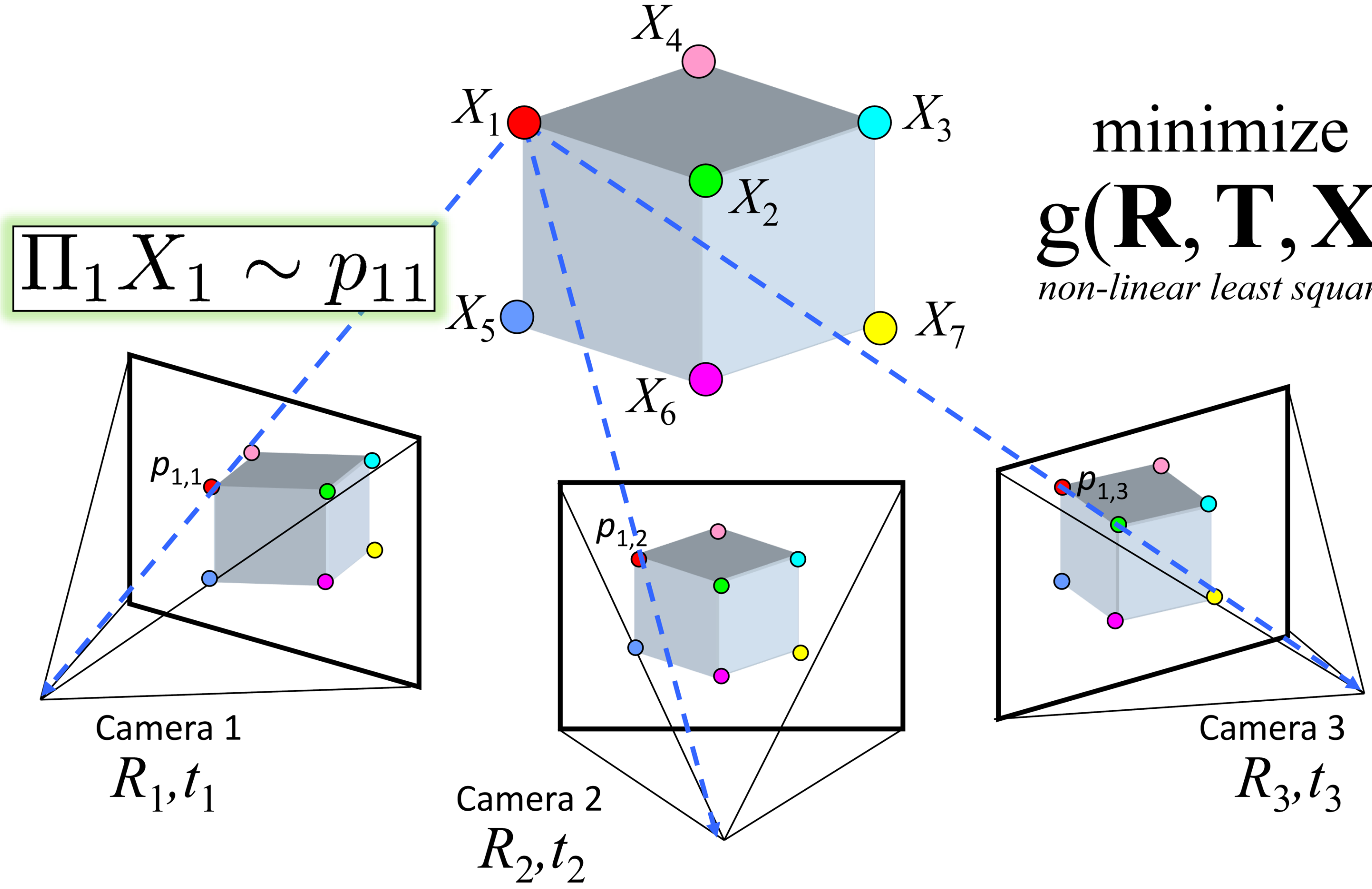
- A set of images from different views

Output

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

Structure from motion



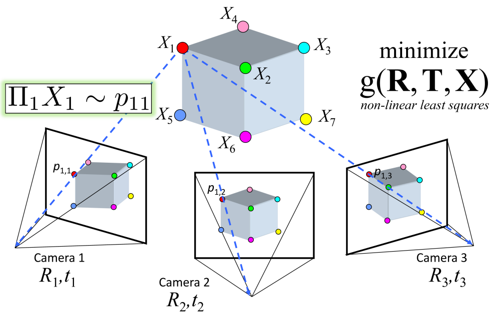


minimize

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

non-linear least squares

Structure from Motion



Minimize sum of squared reprojection errors

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

m points, n images

Indicator variable: is point i visible in image j?

Projection

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

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

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

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

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

An iterative algorithm

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

How to get δ ?

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

Wikipedia

The Levenberg-Marquardt Algorithm

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

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

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

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

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

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

Wikipedia

Structure from Motion

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

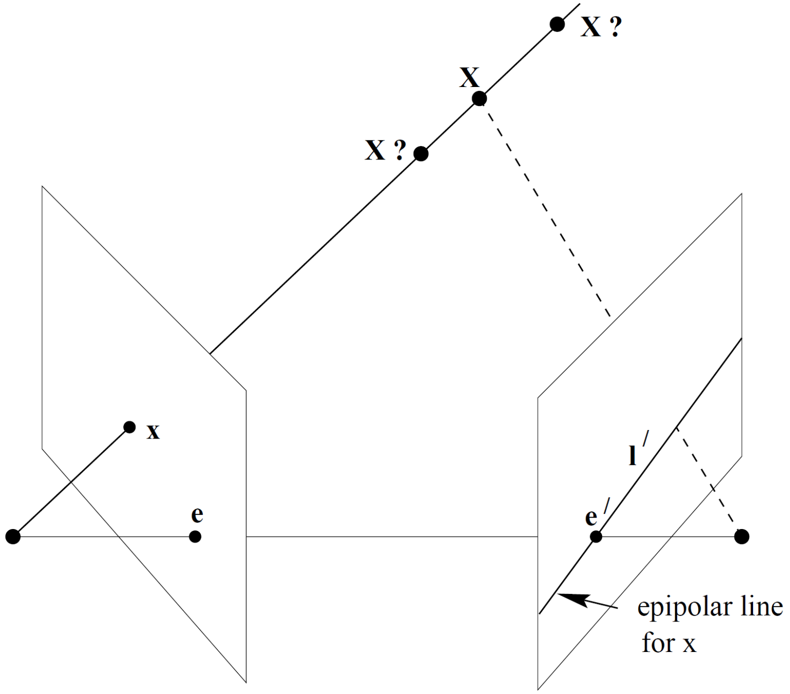
$$\beta = (\mathbf{X},\mathbf{R},\mathbf{T})$$

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 $\mathbf{l}' = F\mathbf{x}$

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

$$\begin{bmatrix} x_i' & y_i' & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = 0$$

$$x_i x_i' f_{11} + x_i y_i' f_{21} + x_i f_{31} + y_i x_i' f_{12} + y_i y_i' f_{22} + y_i f_{32} + x_i' f_{13} + y_i' f_{23} + f_{33} = 0$$

$$\begin{bmatrix} x_1 x_1' & x_1 y_1' & x_1 & y_1 x_1' & y_1 y_1' & y_1 & x_1' & y_1' & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_m x_m' & x_m y_m' & x_m & y_m x_m' & y_m y_m' & y_m & x_m' & y_m' & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{bmatrix} = 0$$

We need 8 points to solve this system.

Matching Two Views

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

If we know camera intrinsics in SfM

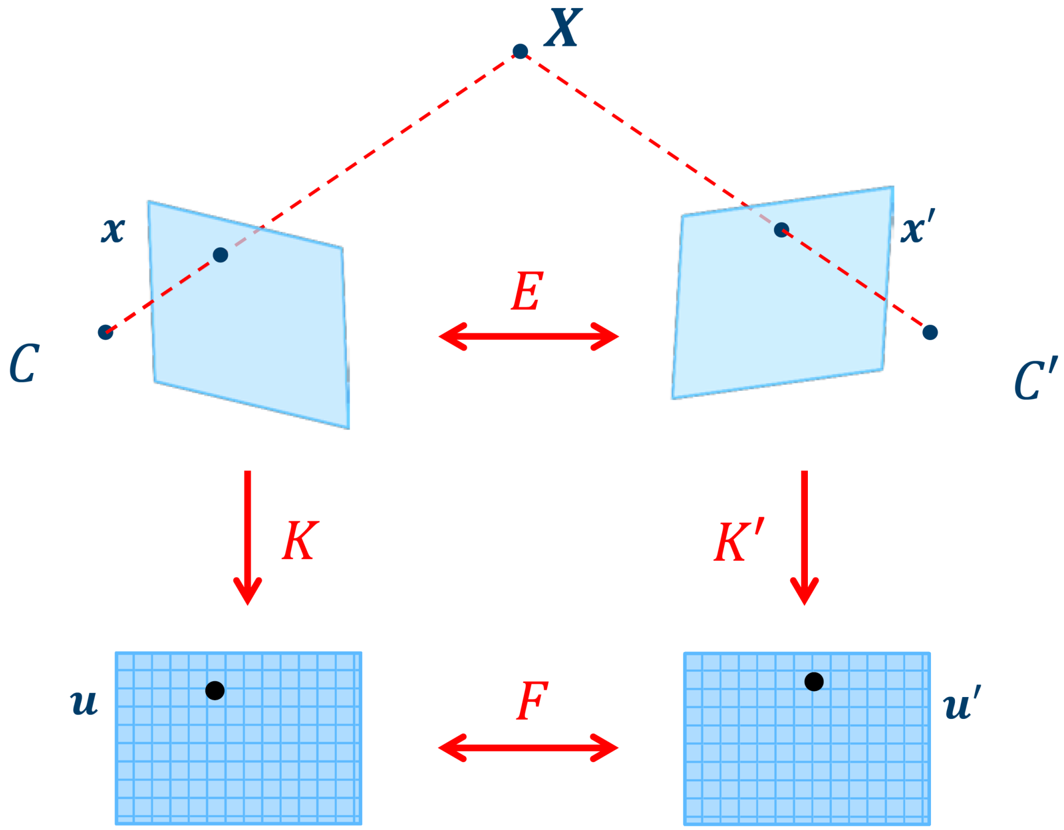
$$(K'^{-1}\mathbf{x}')^T E (K^{-1}\mathbf{x}) = 0$$

Normalized coordinates

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

Essential matrix E

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



Credit: Thomas Opsahl

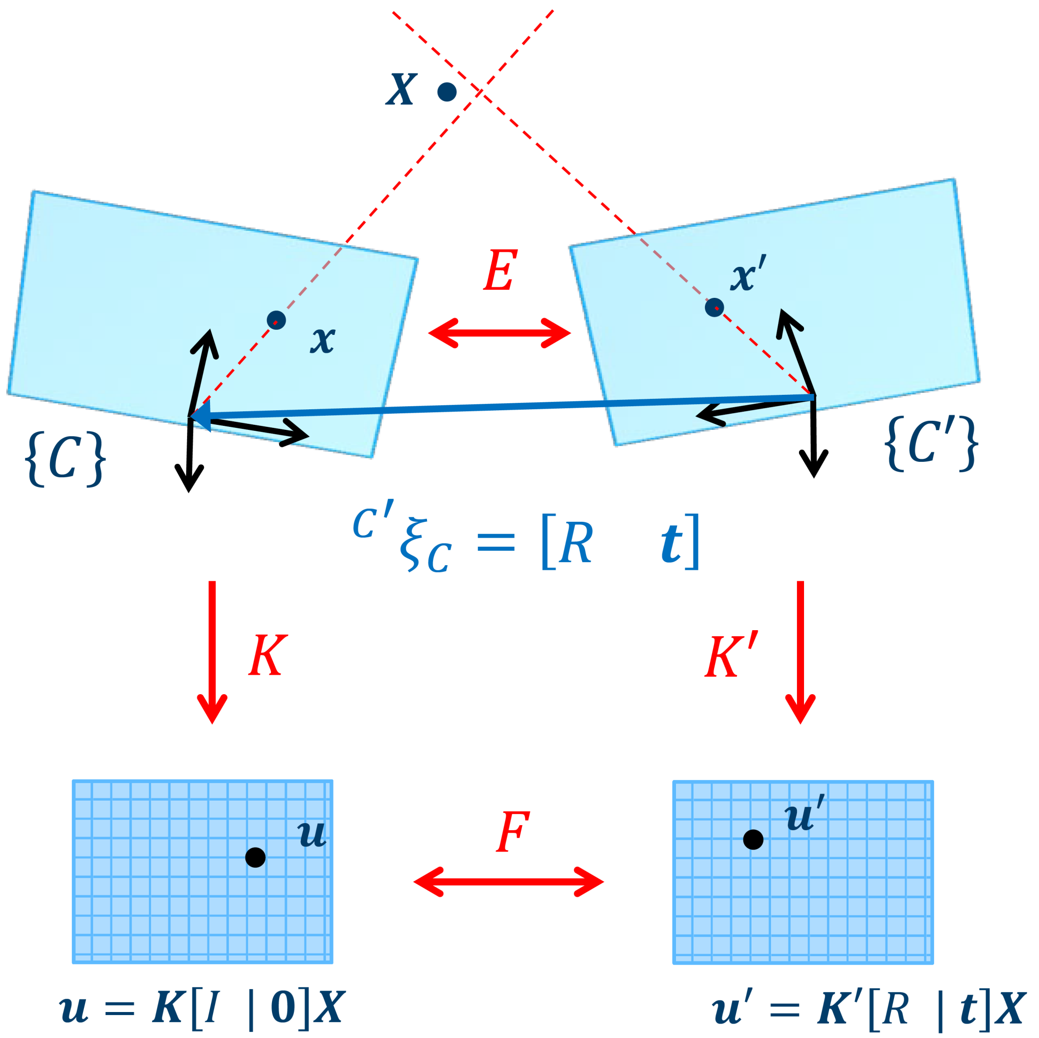
Matching Two Views

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

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

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

$$E = [t]_\times R$$



Credit: Thomas Opsahl

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

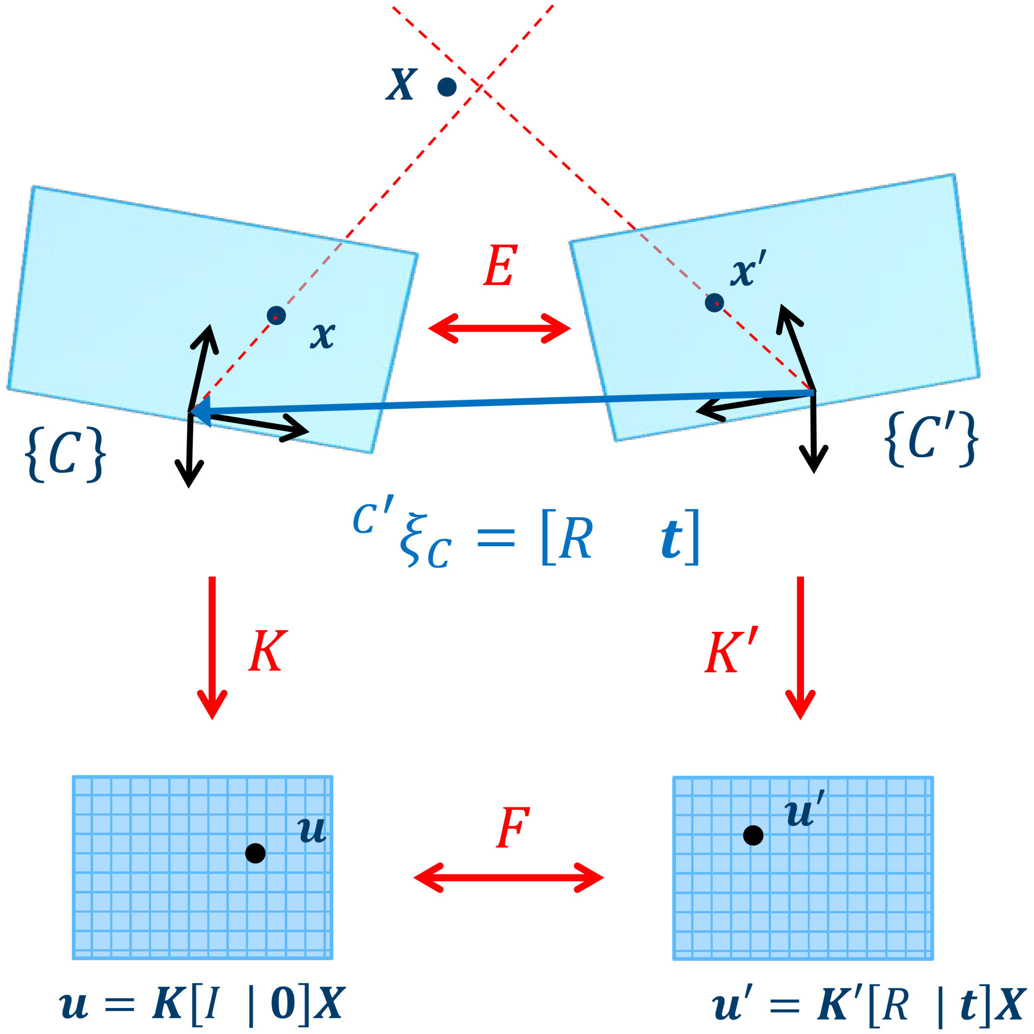
Matching Two Views

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

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

Use SVD to solve for **t**

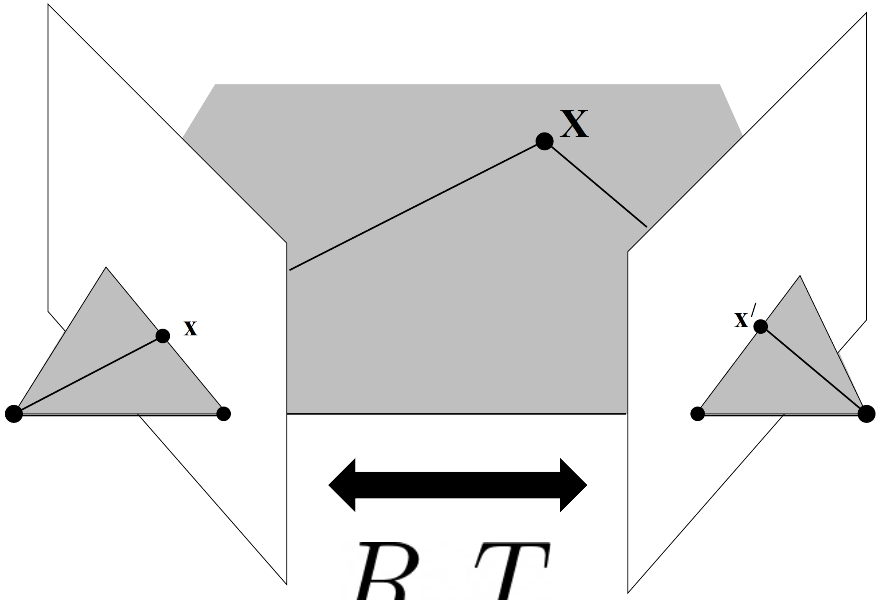
$$R = -[\mathbf{t}]_\times E$$



Credit: Thomas Opsahl

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

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 ?

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

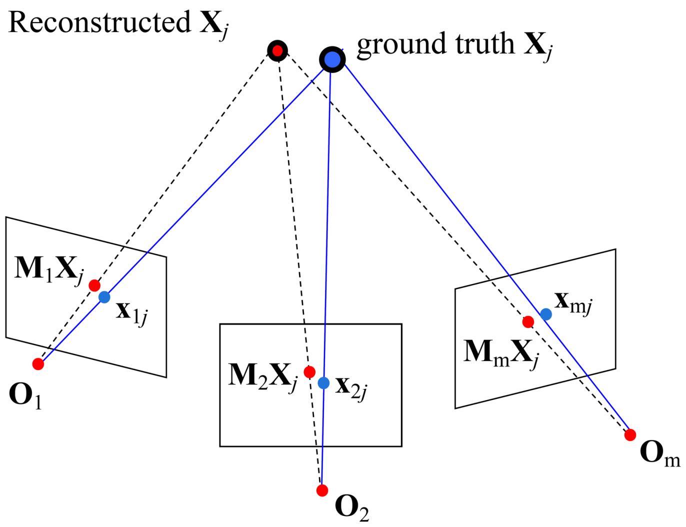
Structure from Motion

Bundle adjustment

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

- Levenberg-Marquardt algorithm

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



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

Build Rome in One Day

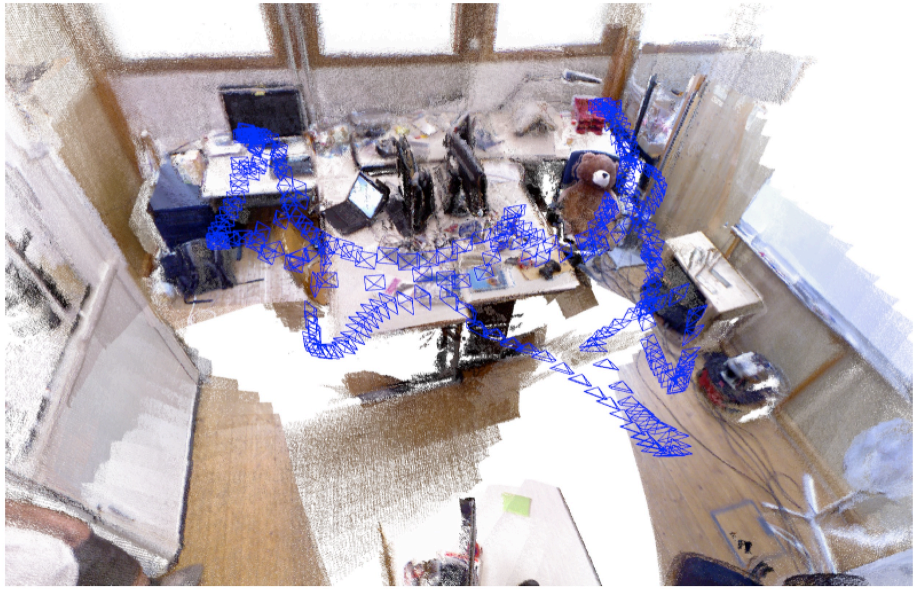
https://grail.cs.washington.edu/rome/

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

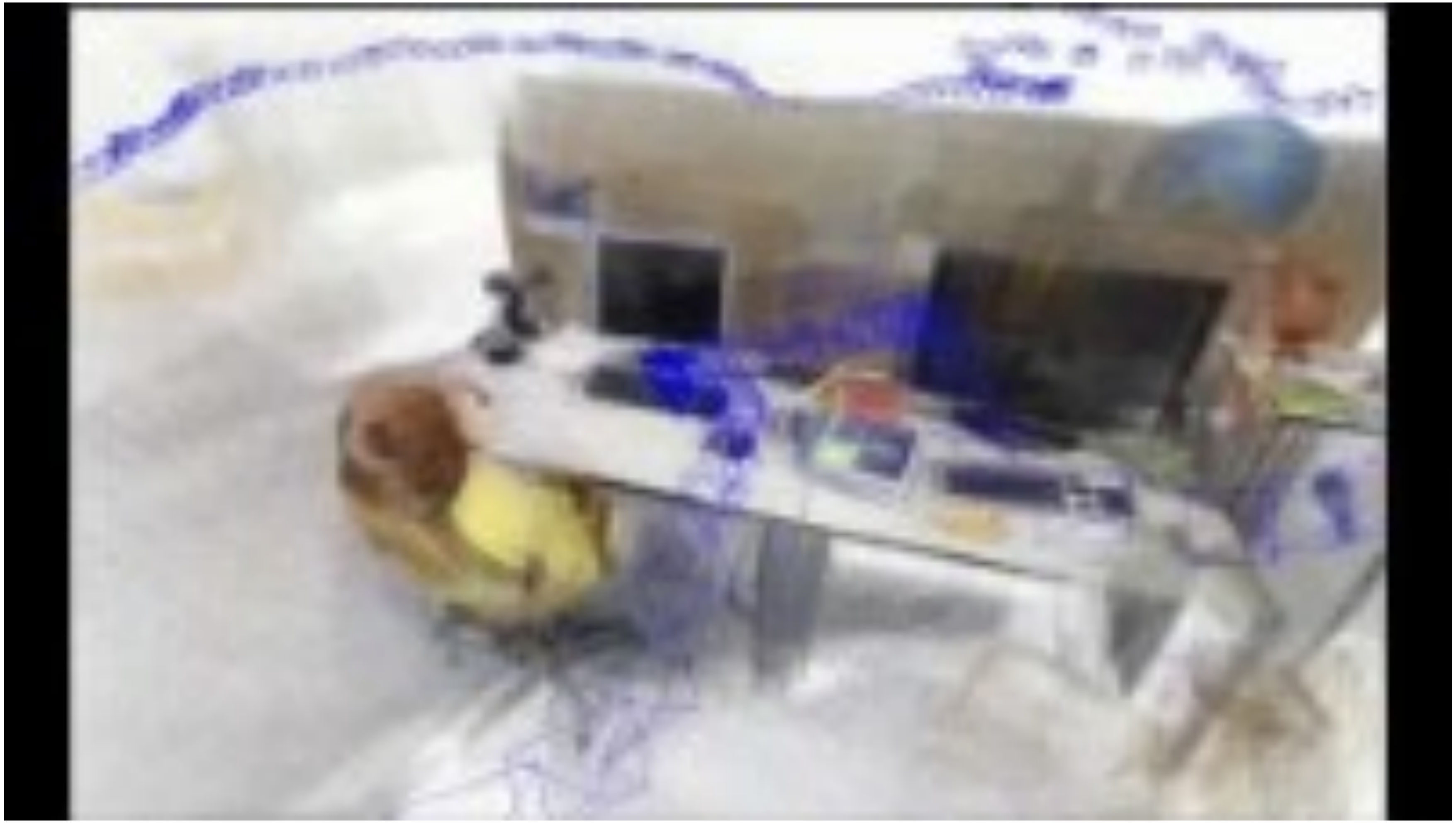


ORB-SLAM2

- 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 (3D location refinement)
- Loop closing
  - Loop detection



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

Case Study: ORB-SLAM

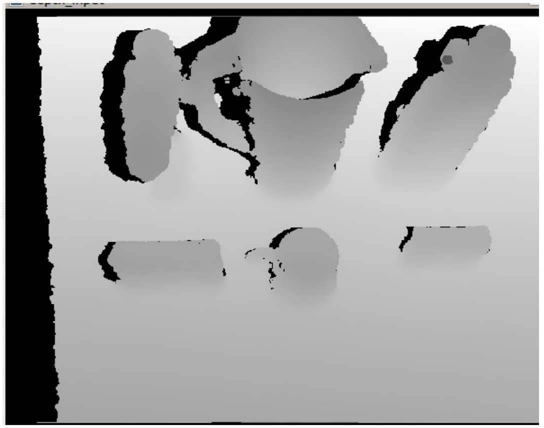
RGB-D SLAM

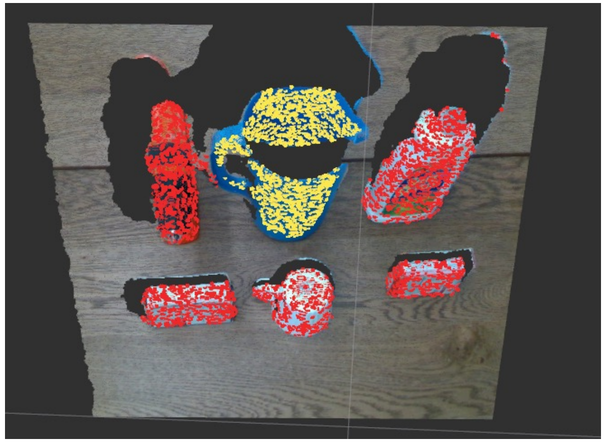
RGB-D cameras

Microsoft Kinect

Intel RealSense

Using depth images: 3D points in the camera frame





Point Cloud

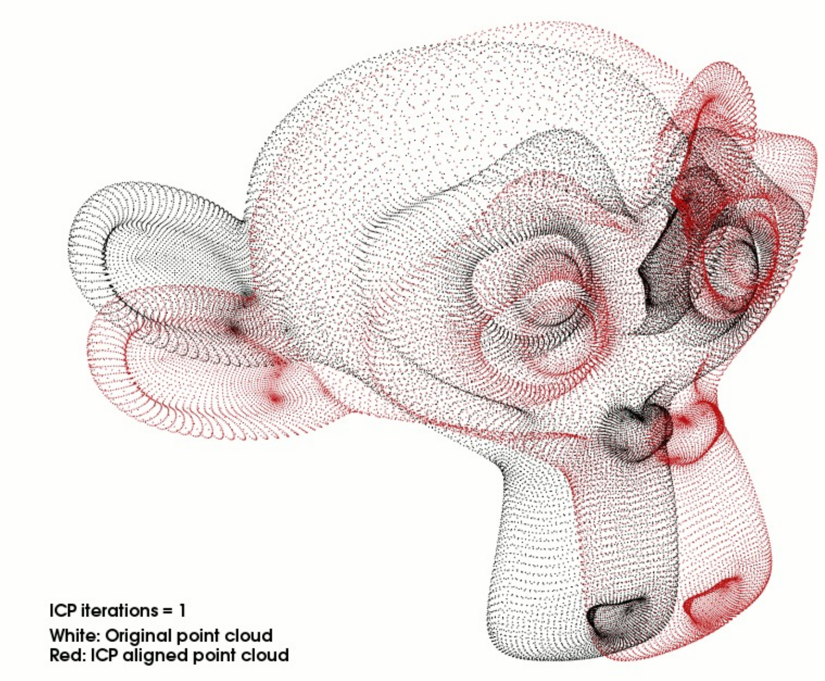
RGB-D SLAM

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



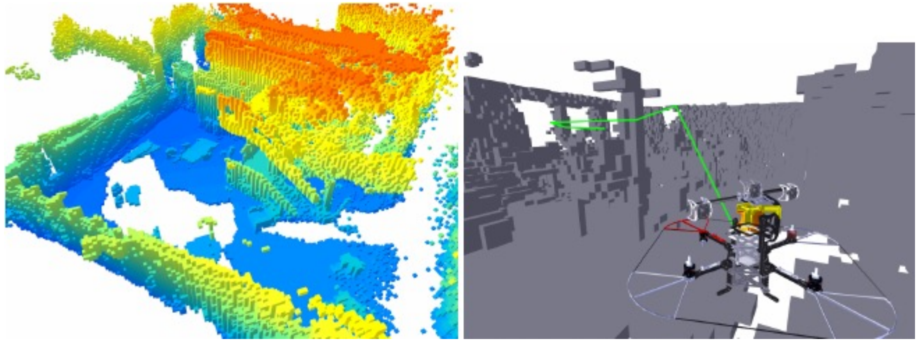
RGB-D SLAM

Mapping: fuse point clouds into a global frame

Map representation

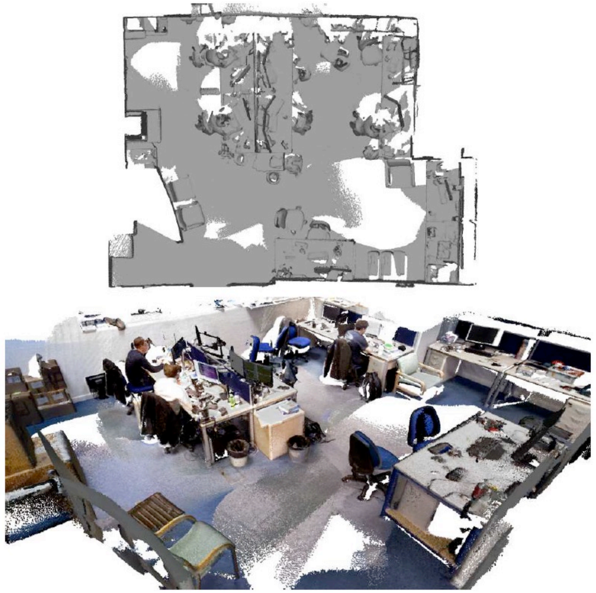
Point clouds

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

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/