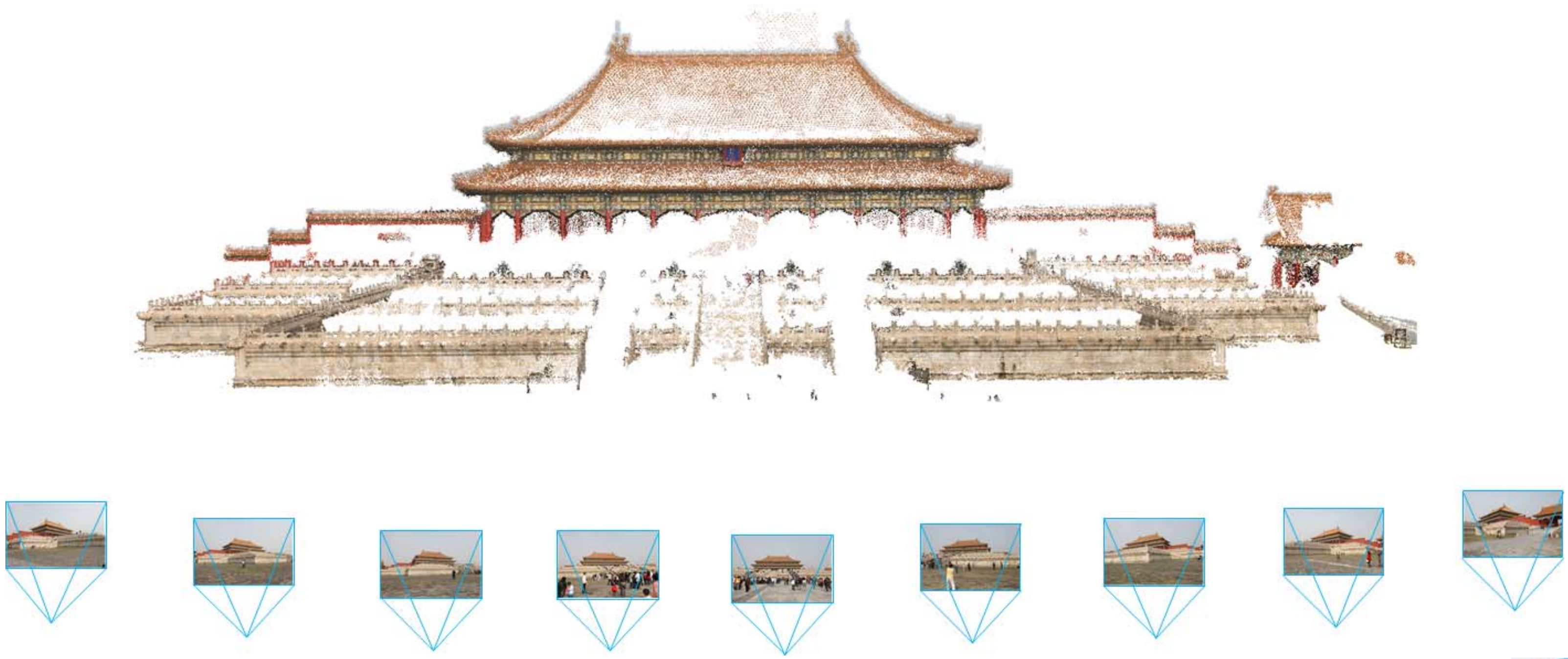
11. Structure-from-Motion

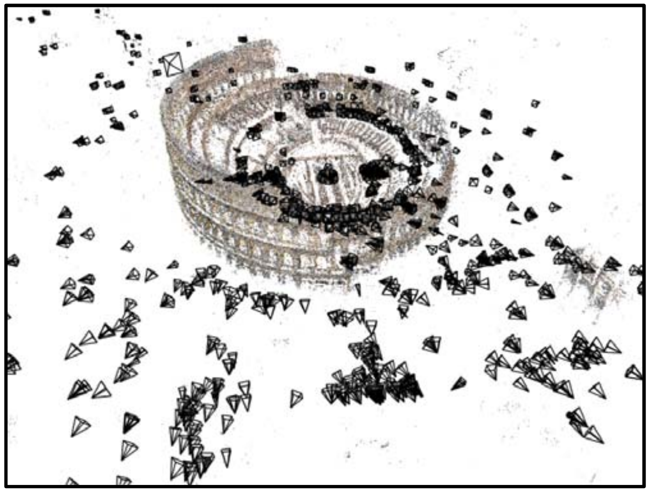


Outline

- Bundle Adjustment
- Rotation Parameterization
- Initializing BA

Structure-from-Motion

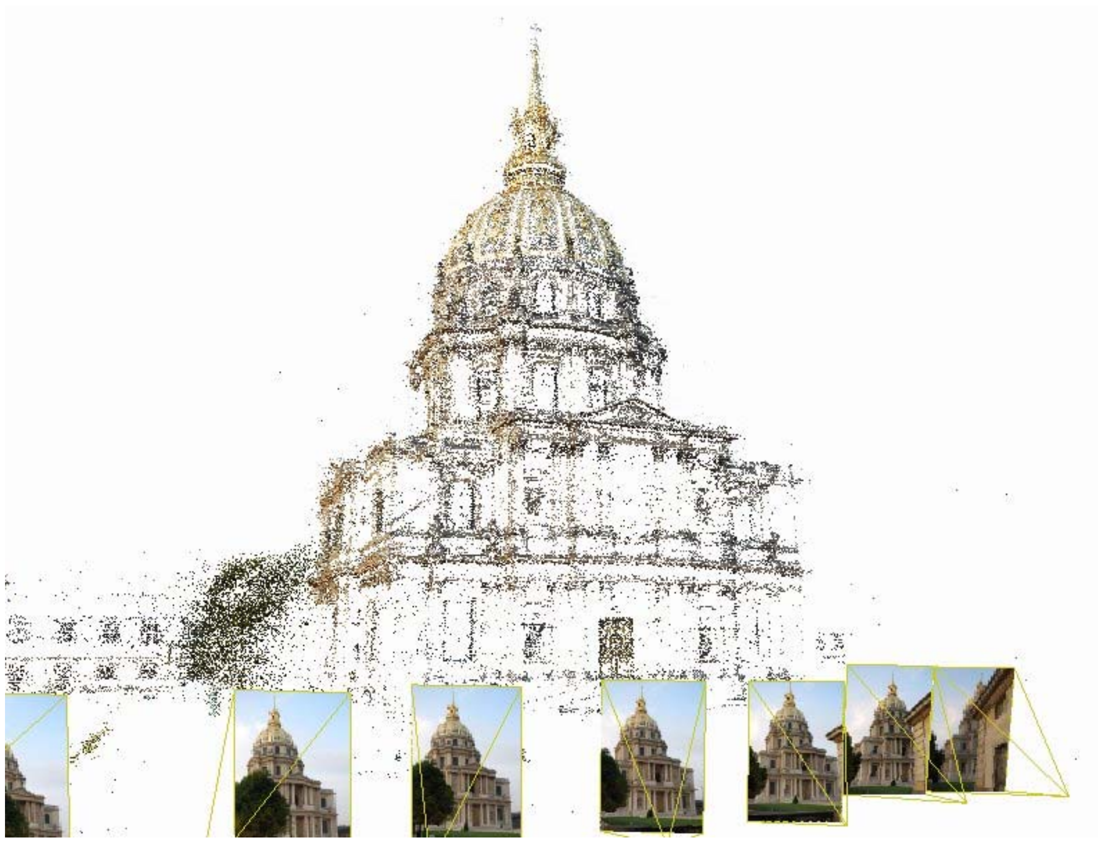
- Given many images, how can we
  - a) figure out where they were all taken from?
  - b) build a 3D model of the scene?



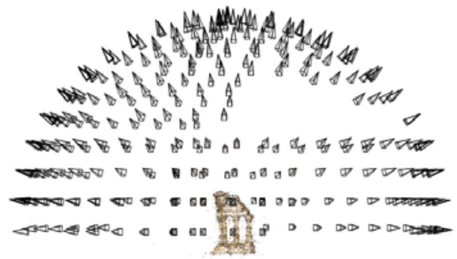
Structure-from-Motion

- Structure = 3D Point Cloud of the Scene
- Motion = Camera Location and Orientation
- SFM = Get the Point Cloud from Moving Cameras

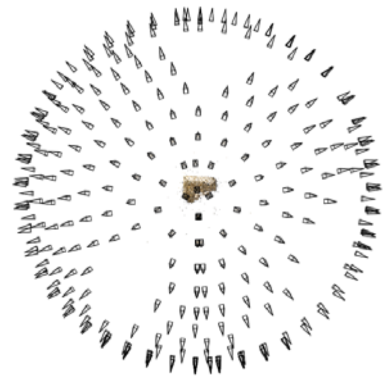
Also Doable from Videos



Formulation



Reconstruction (side)



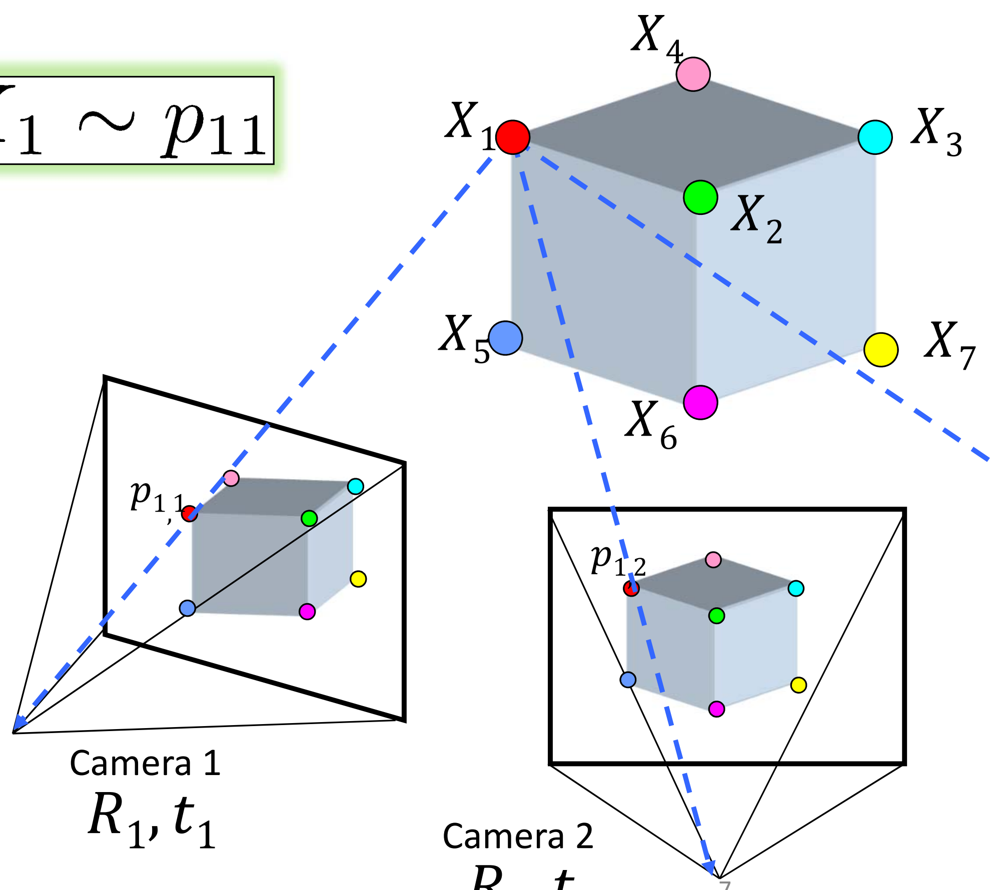
(top)

- Input: images with points in correspondence $p_{ij} = (u_{ij}, v_{ij})$
- Output
  - structure: 3D location X_i for each point p_i
  - motion: camera parameters R_j, t_j possibly K_j
- Objective function: minimize *reprojection error*

Formulation

$$\boxed{\Pi_1 X_1 \sim p_{11}}$$

minimize
$$g(R, T, X)$$
non-linear least squares



SFU

浙江大學 ZHEJIANG UNIVERSITY

Formulation

- Minimize sum of squared reprojection errors:

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

- Minimizing this function is called *bundle adjustment*
  - Optimized using non-linear least squares, e.g. Levenberg-Marquardt

Problem size

- What are the variables?
  - Cameras and points
- How many variables per camera?
- How many variables per point?

- An example with moderate size

  466 input photos

  + > 100,000 3D points

  = very large optimization problem

Questions?

Bundle Adjustment

- The objective function:

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

$$= \sum_{ij} e_{ij}^2 (X_i, R_i, t_i, K_i)$$

  - $e_{ij} = P\left(X_i, R_j, t_j\right) - p_{ij}$ is the 'reprojection error' of X_i in the jth image
- The parameters: $\boldsymbol{X} \in \mathbb{R}^{3m}$, $\boldsymbol{R} \in \mathbb{R}^{3n}$, $\boldsymbol{T} \in \mathbb{R}^{3n}$
  - Typically, $m \gg n$ (why?)
- The optimization method: Levenberg-Marquardt algorithm

SFU

浙江大學 ZHEJIANG UNIVERSITY

Gauss-Newton Method Revisit

- Steps:
- 1. linearize the objective function (nearby an initial solution P_0)

$$f(\mathrm{P}_0 + \Delta) \approx f(\mathrm{P}_0) + \mathrm{J}\Delta \qquad \mathrm{J} = \frac{\partial \mathrm{f}}{\partial \mathrm{P}}$$

- 2. minimize the linearized objective function

$$\Delta = \arg\min \|f(\mathrm{P}_0) + \mathrm{J}\Delta\|^2$$

$$\Rightarrow \mathrm{J}^{\mathrm{T}}\mathrm{J}\Delta = \mathrm{J}^{\mathrm{T}} f(P_0)$$

- 3. solve the linear system to update the initial solution

$$\mathrm{P}_{\mathrm{i}+1} = \mathrm{P}_{\mathrm{i}} + \Delta$$

- 4. iterate 1-3 until converge

SFU

ZHEJIANG UNIVERSITY

Linearize the re-projection error

- Error function: $f(P) = g(\boldsymbol{X}, \boldsymbol{R}, \boldsymbol{T}) = \sum_{ij} e_{ij}^2(\boldsymbol{X}, \boldsymbol{R}, \boldsymbol{T})$
  - $e_{ij} = P(X_i, R_j, t_j) - p_{ij}$

- Linearize it by Taylor expansion:

$$e_{ij}(P) = e_{ij}(P_0) + J_{ij}\Delta$$

$J_{ij} \in \mathbb{R}^{2\times(3m+6n)}$ is the Jacobian matrix, $\Delta \in \mathbb{R}^{3m+6n}$

- The sparse structure of J_{ij}:

$$J_{ij}(\boldsymbol{X}, \boldsymbol{R}, \boldsymbol{T}) = \left(0, \cdots, \frac{\partial e_{ij}}{\partial X_i}, \cdots, \frac{\partial e_{ij}}{\partial R_j}, \frac{\partial e_{ij}}{\partial T_i}, \cdots, 0\right)$$

SFU

浙江大学 ZHEJIANG UNIVERSITY

Linearize the re-projection error

- The linearized objective function:

$$f(P) = \sum_{ij} \left(e_{ij}(P_0) + J_{ij}\Delta\right)^2 \approx \boldsymbol{c} + 2\boldsymbol{b}^T\Delta + \Delta^T\boldsymbol{H}\Delta$$

with

$$\boldsymbol{b}^T = \sum_{ij} e_{ij}^T \mathrm{J_{ij}} \qquad \boldsymbol{H} = \sum_{ij} J_{ij}^T J_{ij} \in \mathbb{R}^{(3m+6n)\times(3m+6n)}$$

This is huge!

$\boldsymbol{H}$ is the Hessian matrix.

- Set the partial derivative to zero:

$$\boldsymbol{H}\Delta = -\boldsymbol{b}$$

- Solving this linear system for improved results:

$$P \leftarrow P + \Delta$$

SFU

浙江大学 ZHEJIANG UNIVERSITY

Gauss-Newton Algorithm

- Repeat until convergence:
- 1. Compute the terms of linear systems:

$$\boldsymbol{b}^T = \sum_{ij} e_{ij}^T \mathrm{J_{ij}} \qquad \boldsymbol{H} = \sum_{ij} J_{ij}^T J_{ij} \in \mathbb{R}^{(3m+6n)\times(3m+6n)}$$

- 2. Solve the linear systems by

$$\boldsymbol{H}\Delta = -\boldsymbol{b}$$

- 3. Update the previous results by:

$$P \leftarrow P + \Delta$$

SFU

浙江大學 ZHEJIANG UNIVERSITY

The Hessian

- The Hessian H is
  - Positive semi-definite
  - Symmetric
  - Sparse
- This allows efficient solution
  - Detailed late

Levenberg-Marquardt Algorithm

- Observations:
  - Gauss-Newton method typically converges very quickly
  - Sometimes diverges when initial solution is far off
  - Gradient descent (with line search) never diverges

- **How can we combine the advantages of both minimization methods?**

Levenberg-Marquardt Algorithm

- Idea: Add a damping factor

$$(H + \lambda I)\Delta = -\mathrm{b}$$

- The effect of this damping factor:
  - Small λ, the same as Gaussian-Newton
  - Large λ, the same as gradient descendant
- Algorithm:
  - If error decrease, accept Δ and reduce λ
  - If error increase, reject Δ and increase λ

- Update the previous results by:

$$P \leftarrow P + \Delta$$

SFU

ZHEJIANG UNIVERSITY

Various Open Source Solvers

- PBA [Wu et al. 2011]
- Ceres [Google, 2012]
- G2O [Kuemmerle et al., 2011]
- SBA [Lourakis and Argyros, 2009]
- iSAM [Kaess et al., 2008]

Questions?

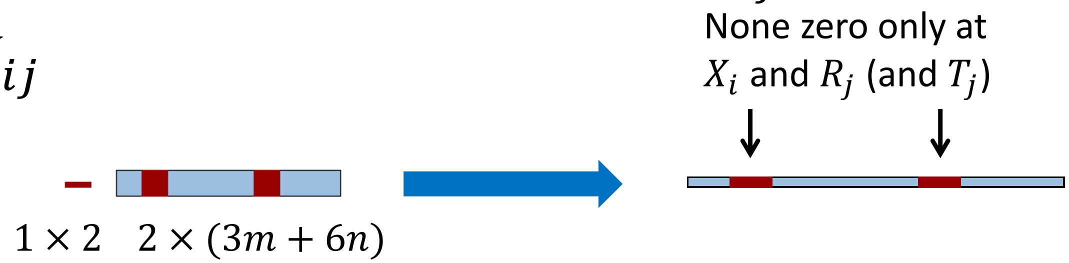
Structure of $\boldsymbol{b}$ and $\boldsymbol{H}$

$$\boldsymbol{b}^T = \sum_{ij} e_{ij}^T \mathrm{J}_{\mathrm{ij}} \qquad \boldsymbol{H} = \sum_{ij} J_{ij}^T J_{ij}$$

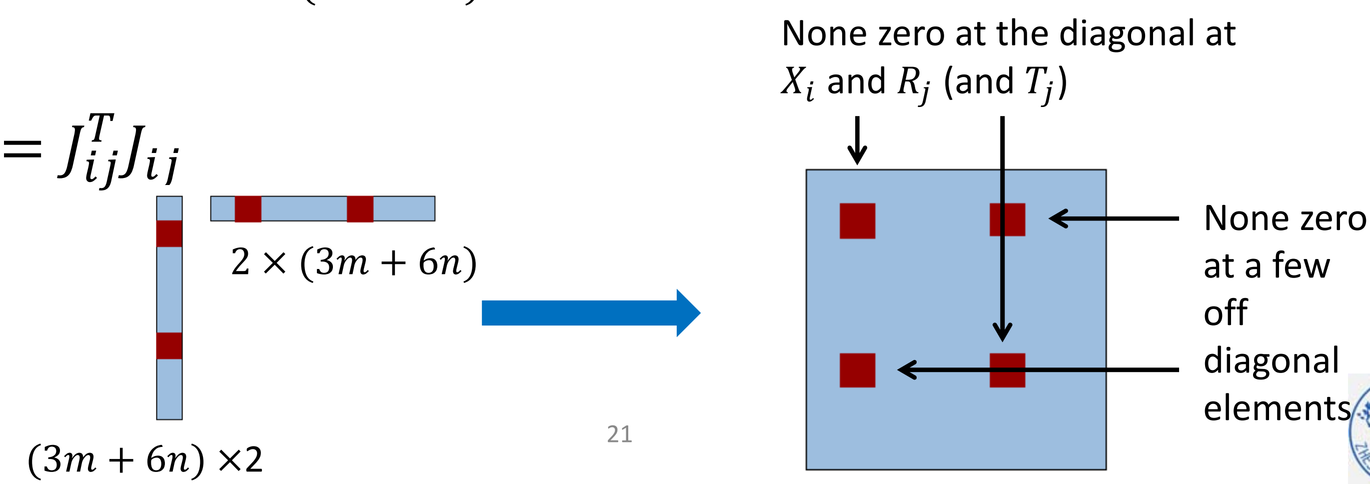
- Remember J_{ij}'s sparse structure

$$J_{ij}(\boldsymbol{X}, \boldsymbol{R}, \boldsymbol{T}) = \left(0, \cdots, \frac{\partial e_{ij}}{\partial X_i}, \cdots, \frac{\partial e_{ij}}{\partial R_j}, \frac{\partial e_{ij}}{\partial T_j}, \cdots, 0\right)$$

- So $b_{ij} = e_{ij}^T J_{ij}$



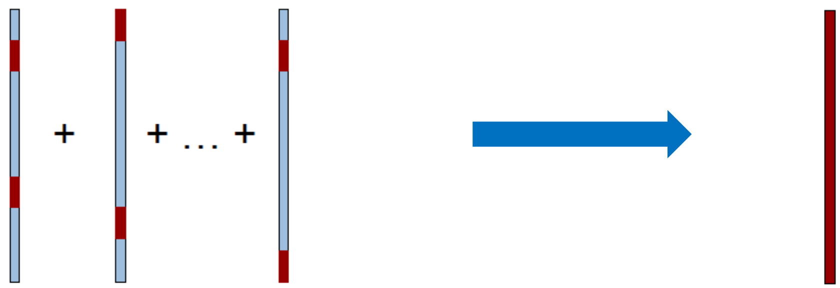
$H_{ij} = J_{ij}^T J_{ij}$



Structure of $\boldsymbol{b}$ and $\boldsymbol{H}$

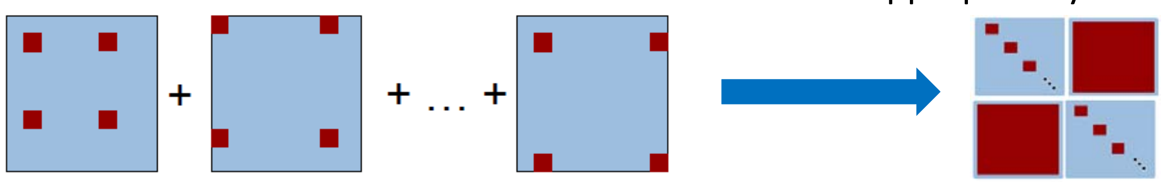
$$\boldsymbol{b}^T = \sum_{ij} b_{ij}$$

$\boldsymbol{b}$ is a dense vector



$$\boldsymbol{H} = \sum_{ij} J_{ij}^T J_{ij}$$

$\boldsymbol{H}$ has a special structure, if we order the parameters appropriately



Structure of $\boldsymbol{H}$

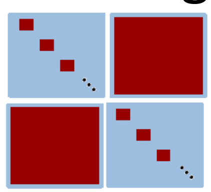
- Characteristic structure

$$\begin{pmatrix} J_C^T J_C & J_C^T J_P \\ J_P^T J_C & J_P^T J_P \end{pmatrix} \begin{pmatrix} \Delta_C \\ \Delta_P \end{pmatrix} = \begin{pmatrix} -b_C \\ -b_P \end{pmatrix}$$

Or

$$\begin{pmatrix} H_{CC} & H_{CP} \\ H_{PC} & H_{PP} \end{pmatrix} \begin{pmatrix} \Delta_C \\ \Delta_P \end{pmatrix} = \begin{pmatrix} -b_C \\ -b_P \end{pmatrix}$$

- Both H_{CC} and H_{PP} are block diagonal



$$\begin{pmatrix} \Delta_C \\ \Delta_P \end{pmatrix} = \begin{pmatrix} -b_C \\ -b_P \end{pmatrix}$$

- This can be solved using the Schur Complement

Schur Complement

- Given linear system

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix}$$

- If D is invertible, then by Gauss elimination,

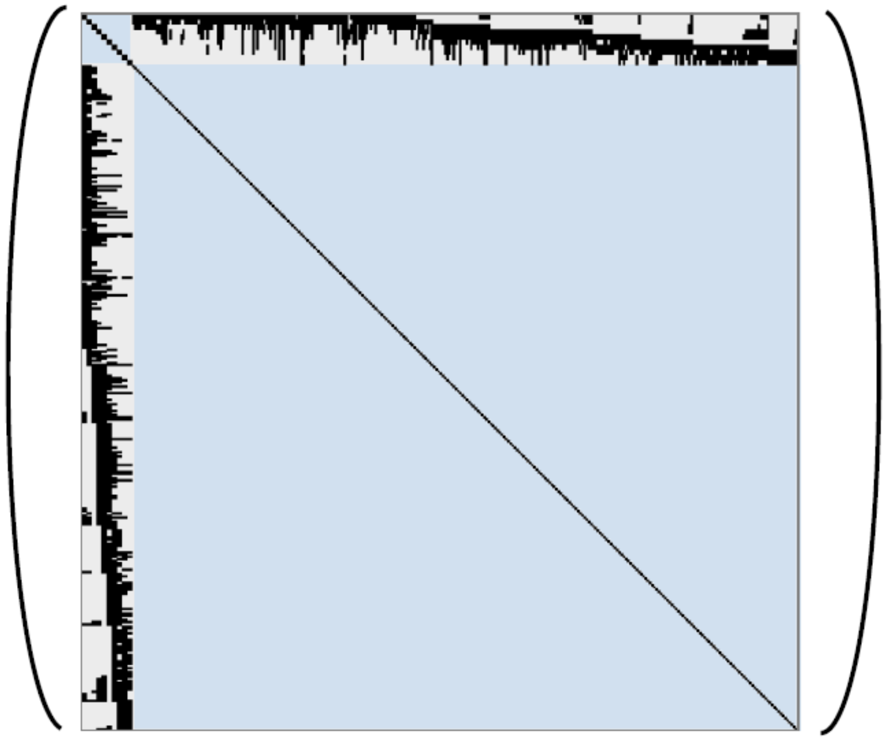
$$(A - BD^{-1}C)x = a - BD^{-1}b$$
$$y = D^{-1}(b - Cx)$$

- This reduces computation complexity,
  i.e. from inverting a $(3m + 6n) \times (3m + 6n)$ matrix to inverting a $3m \times 3m$ and a $6n \times 6n$ matrix, each is block-diagonal

SFU

浙江大学 ZHEJIANG UNIVERSITY

Example Hessian

$H =$ 

Questions?

Outline

- Bundle Adjustment
- Rotation Parameterization
- Initializing BA

Parameterizing Rotation Matrix

- One last problem
  - Recall $J_{ij}(\boldsymbol{X}, \boldsymbol{R}, \boldsymbol{T}) = (0, \cdots, \frac{\partial e_{ij}}{\partial X_i}, \cdots, \frac{\partial e_{ij}}{\partial R_j}, \frac{\partial e_{ij}}{\partial T_i}, \cdots, 0)$
  - How do we parameterize $\boldsymbol{R}$?

- A rotation matrix is a 3x3 orthogonal matrix

$$\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$$

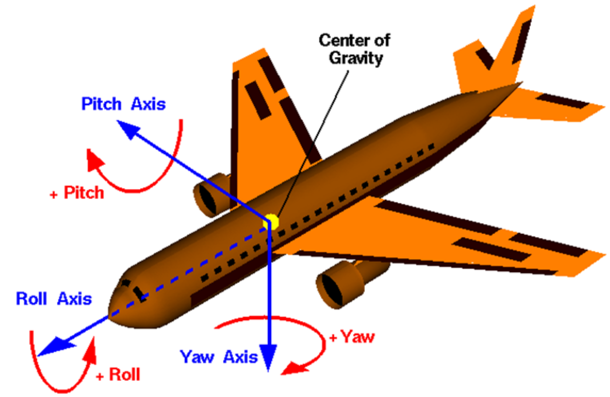
  - Also called the special orientation group SO(3)
- 9 parameters with 3 DoF!
  - The computed result might not be a rotation matrix, i.e. $R^T R \neq 1$

SFU

浙江大學 ZHEJIANG UNIVERSITY

Representing $\boldsymbol{R}$ by 3 Angles

- Roll ϕ , Pitch θ, Yaw ψ
  - is very common in aerial navigation
- Conversion to 3x3 rotation matrix:



$$
\begin{aligned}
\mathbf{R} &= \mathbf{R}_Z(\psi)\mathbf{R}_Y(\theta)\mathbf{R}_X(\phi) \\
&= \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix} \\
&= \begin{pmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{pmatrix}
\end{aligned}
$$

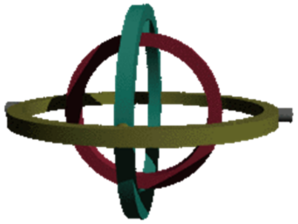
SFU

ZHEJIANG UNIVERSITY

Representing $\boldsymbol{R}$ by 3 Angles

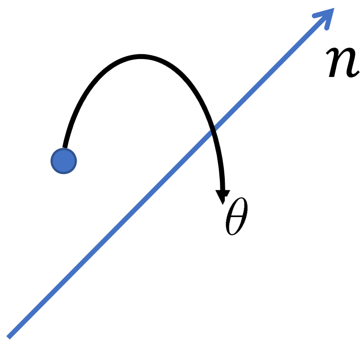
- Advantage:
  - Minimal representation (3 parameters)
  - Easy interpretation
- Disadvantages:
  - Many "alternative" Euler representations exist (XYZ, ZXZ, ZYX, …)
  - Difficult to concatenate
  - Singularities (gimbal lock)
    - E.g. when $\theta = 90^o$, ϕ, ψ cannot be differentiated (2 DoFs combine to 1)

$$\boldsymbol{R} = \begin{bmatrix} 0 & 0 & -1 \\ \sin(\psi - \phi) & \cos(\psi - \phi) & 0 \\ \cos(\psi - \phi) & -\sin(\psi - \phi) & 0 \end{bmatrix}$$



Axis-Angle Representation

- Represent rotation by
  - rotation axis n and angle θ
- 4 parameters (θ, n)
- 3 parameters $\theta \cdot n$
  - length is rotation angle
- Disadvantage:
  - Not a unique representation
  - Difficult to concatenate
  - Slow conversion



Axis-Angle Representation

- Rodriguez' formula

$$\mathbf{R}(\hat{\mathbf{n}}, \theta) = \mathbf{I} + \sin\theta[\hat{\mathbf{n}}]_\times + (1 - \cos\theta)[\hat{\mathbf{n}}]_\times^2$$

- Inverse

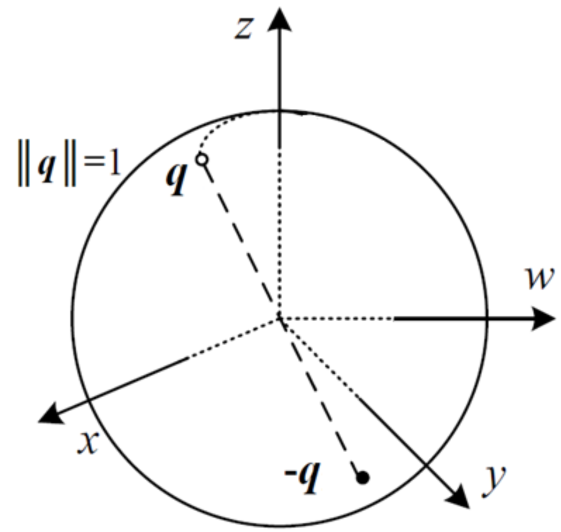
$$\theta = \cos^{-1}\left(\frac{\mathrm{trace}(\mathbf{R}) - 1}{2}\right), \hat{\mathbf{n}} = \frac{1}{2\sin\theta}\begin{pmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{pmatrix}$$

SFU

ZHEJIANG UNIVERSITY

Quaternions

- Quaternion $\mathbf{q} = (q_1, q_2, q_3, q_4)$
- It is an extension of complex numbers
  - $q_1 + q_2\boldsymbol{i} + q_3\boldsymbol{j} + q_4\boldsymbol{k}$
  - $\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = -1$
- Unit quaternions have $\left|\left|\mathbf{q}\right|\right| = 1$
- Relation to angle-axis representation
  - $\mathbf{q} = (r, \boldsymbol{v}) = \left(\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\,\boldsymbol{n}\right)$
- $\boldsymbol{q}$ and $-\boldsymbol{q}$ represent the same rotation



Quaternions

- Advantage:
  multiplication, inversion and rotations are very efficient
- Concatenation

$$(r_1, \mathbf{v}_1)(r_2, \mathbf{v}_2) = (r_1 r_2 - \mathbf{v}_1 \cdot \mathbf{v}_2, r_1 \mathbf{v}_2 + r_2 \mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2)$$

- Inverse (=flip signs of real or imaginary part)

$$(r, \mathbf{v})^{-1} = (r, \mathbf{v})^\star \equiv (-r, \mathbf{v}) \equiv (r, -\mathbf{v})$$

- Rotate 3D vector $\mathbf{p} \in \mathbb{R}^3$ using a quaternion:

$$(r, \mathbf{v})(0, \mathbf{p})(r, \mathbf{v})^\star$$

SFU

ZHEJIANG UNIVERSITY

Desired Rotation Parameterization

- No over-parameterization (avoid using 3×3 matrix)
  - Using 3 parameters to represent a rotation matrix
- No degeneracy (avoid using Euler angles)
  - Degeneracy: a subspace of the parameter space corresponds to a single rotation matrix
- The optimization algorithm can change parameters freely
  - The result is always a valid rotation matrix
- Still a somewhat unsolved problem

Rotation Parameterization in BA

- During the LM optimization:
  - Compute the terms $\boldsymbol{H}, \boldsymbol{b}$, wrt the parameters to be optimized (e.g. quaternions)
  - Solve the linear systems by
$$(\boldsymbol{H} + \lambda \boldsymbol{I})\Delta = -\boldsymbol{b}$$
  - Update the previous results by:
$$P \leftarrow P + \Delta$$
- A quaternion has 4 parameters $\mathbf{q} = (q_1, q_2, q_3, q_4)$, we can:
  - Use 4 independent parameters and enforce $||\mathbf{q}|| = 1$ at each step;
  - Enforce the constraint $||\mathbf{q}|| = 1$, e.g. by Lagrange multiplier;
  - Focus on Δ (a small update), and parameterize it by 3 parameters (e.g. the last three elements of a quaternion, or a axis-angle representation).

SFU

浙江大學 ZHEJIANG UNIVERSITY

Questions?

Outline

- Bundle Adjustment
- Rotation Parameterization
- Initializing BA

Initializing the Bundle Adjustment

- Levenberg-Marquardt algorithm requires good initial guess for:
  - 3D points X_i
  - camera parameters R_j, t_j, K_j

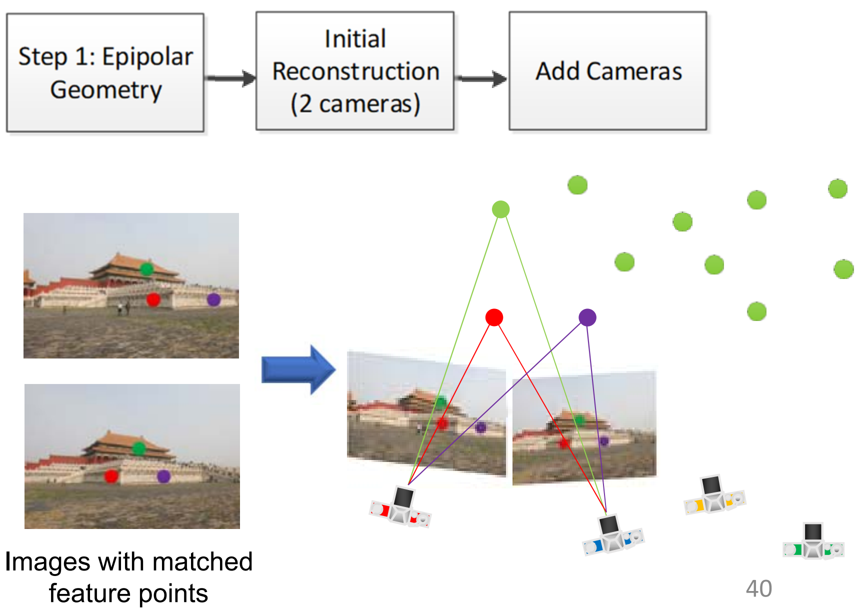
- How do we initialize?
- Two typical solutions:
  - Incremental Structure-from-Motion
  - Global Structure-from-Motion

SFU

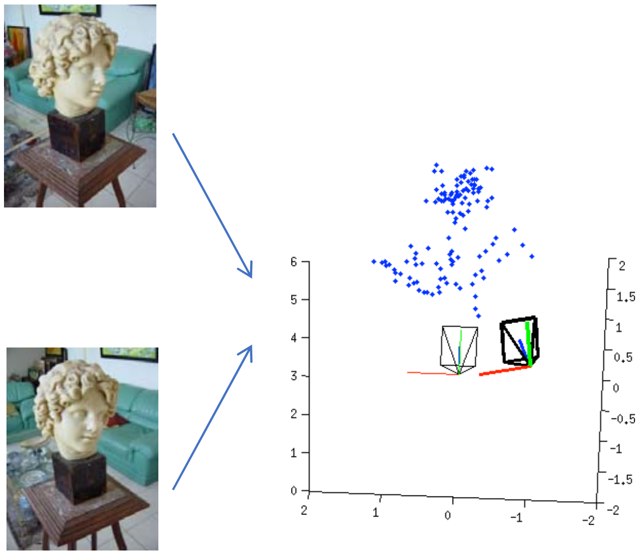
浙江大學 ZHEJIANG UNIVERSITY

Incremental Structure-from-Motion

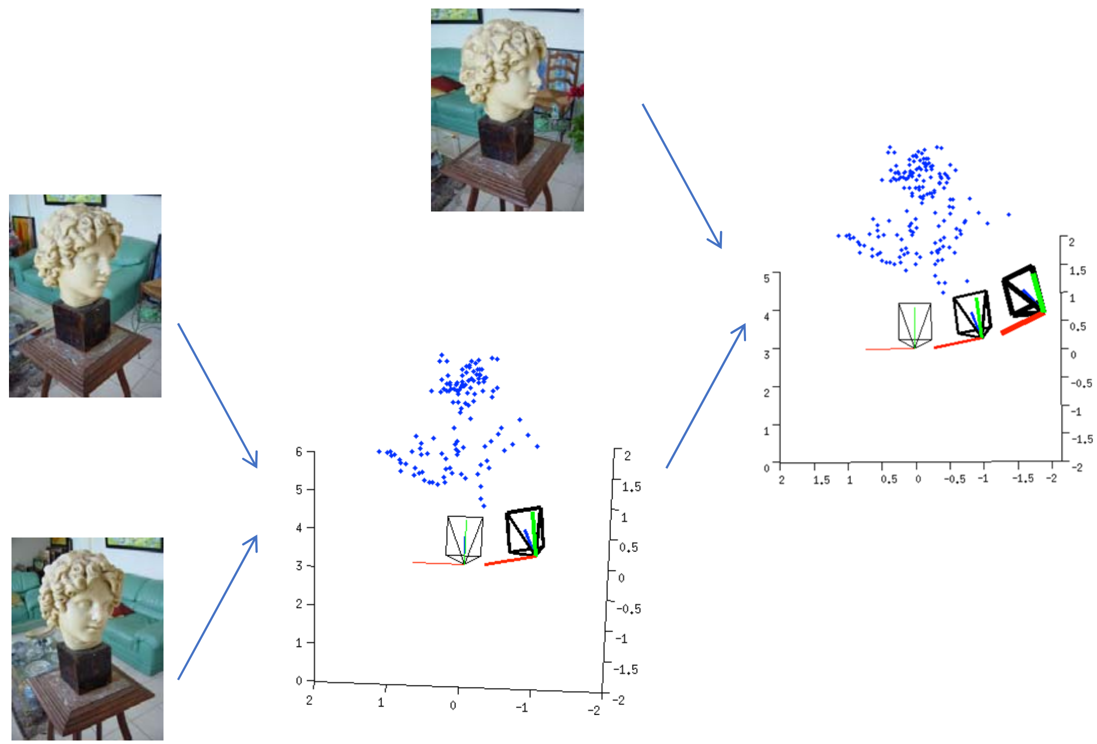
1. Solve a two-view reconstruction (essential matrix, decomposition, triangulation)
2. Add cameras by resection with 3D-2D correspondences (resection, PnP)
   Might triangulate more points from the newly added cameras (resection)



two-view reconstruction

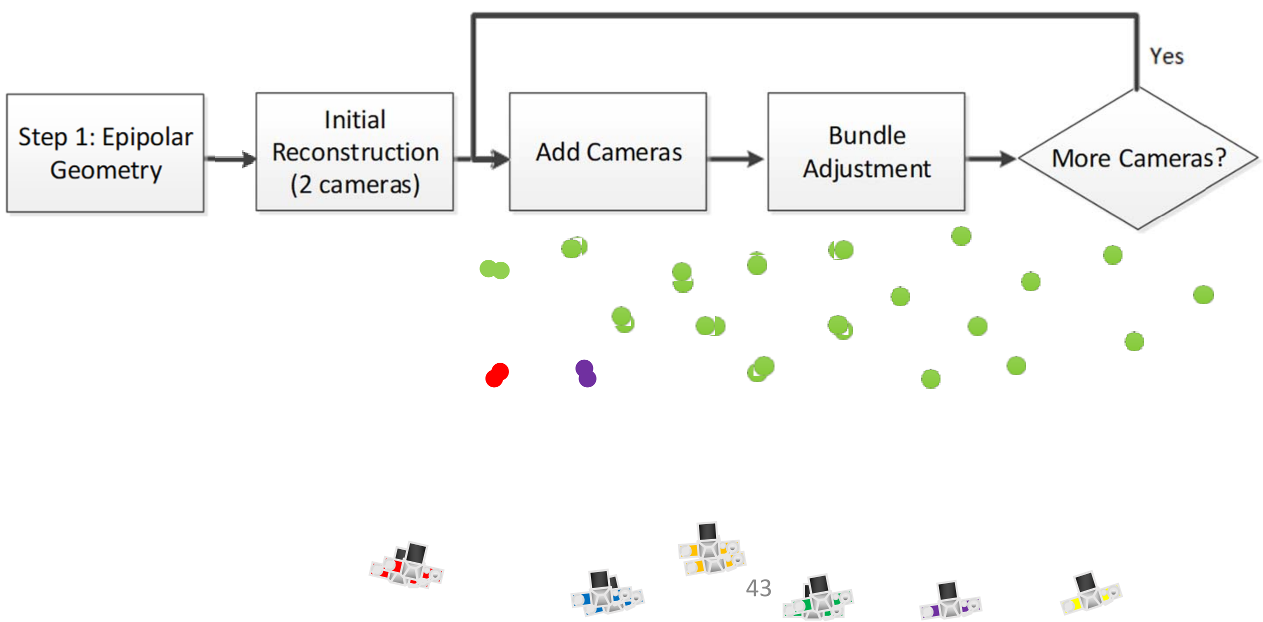


incrementally add the third view



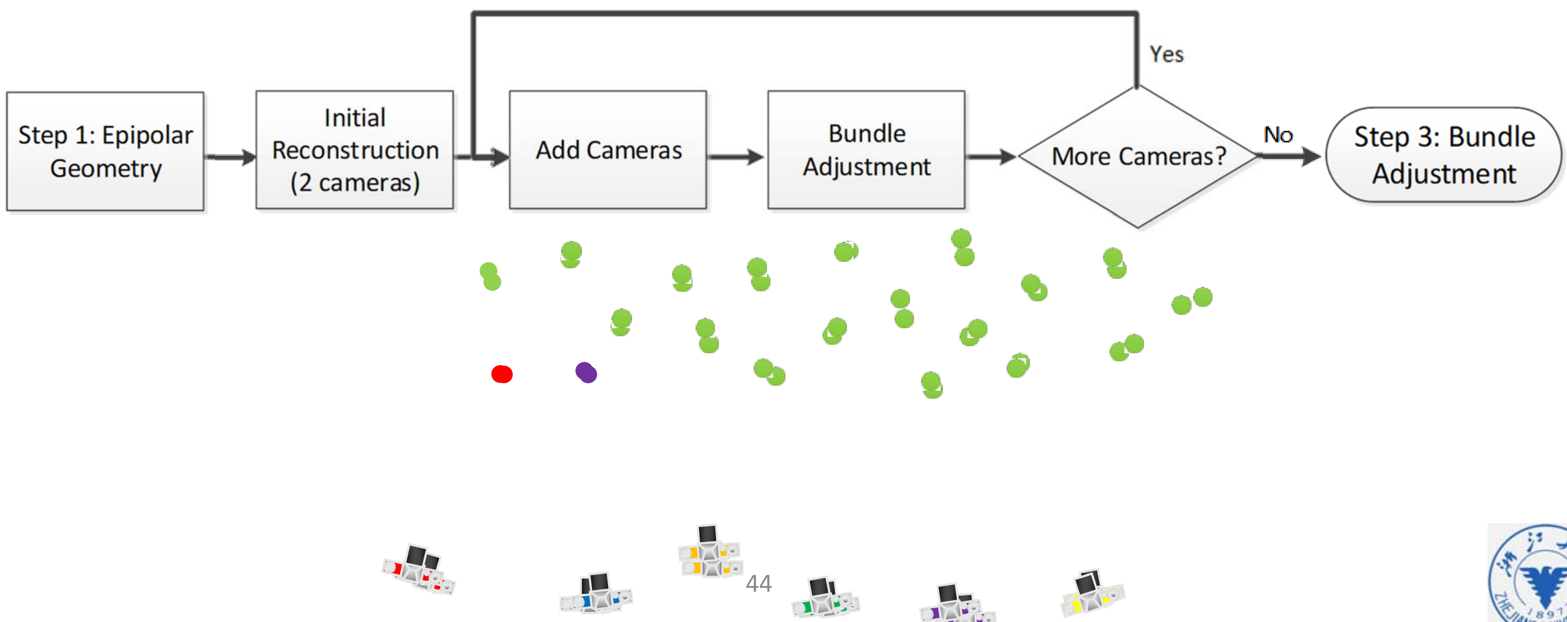
Incremental Structure-from-Motion

1. Solve a two-view reconstruction (essential matrix, decomposition, triangulation)
2. Add cameras by resection with 3D-2D correspondences (resection, PnP)
   Might triangulate more points from the newly added cameras (resection)
3. Repeat step-2 (with intermediate BA to reduce error accumulation)



Incremental Structure-from-Motion

1. Solve a two-view reconstruction (essential matrix, decomposition,triangulation)
2. Add cameras by resection with 3D-2D correspondences (resection, PnP)
   Might triangulate more points from the newly added cameras (resection)
3. Repeat step-2 (with intermediate BA to reduce error accumulation)



SFU

ZHEJIANG UNIVERSITY

Other Issues

- Which two images to begin with?
  - Maybe two images with high quality essential matrix
- Which is the next image to add (next-best-view)?
  - Maybe the one with most correspondences to existing 3D map

- Different answers to these questions lead to different result.

SFU

浙江大學 ZHEJIANG UNIVERSITY

Drawbacks of Incremental SfM

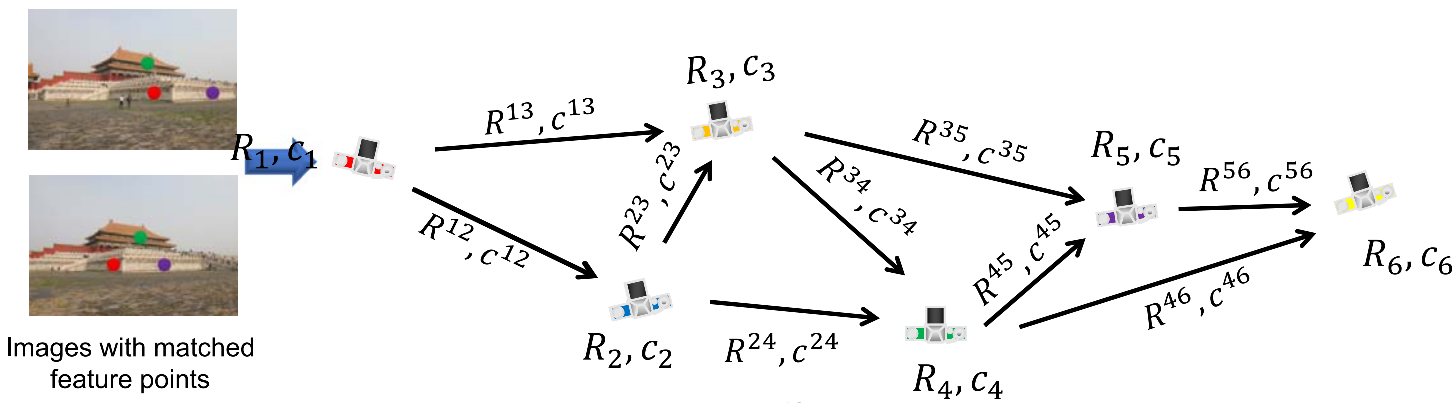
- Poor run-time efficiency
  - Repetitively solving the nonlinear bundle adjustment (though locally)
  - Most of the computation time is spent on bundle adjustment

- Inferior results
  - Some cameras are fixed when solving the others
  - It is desirable to solve all cameras simultaneously

Questions?

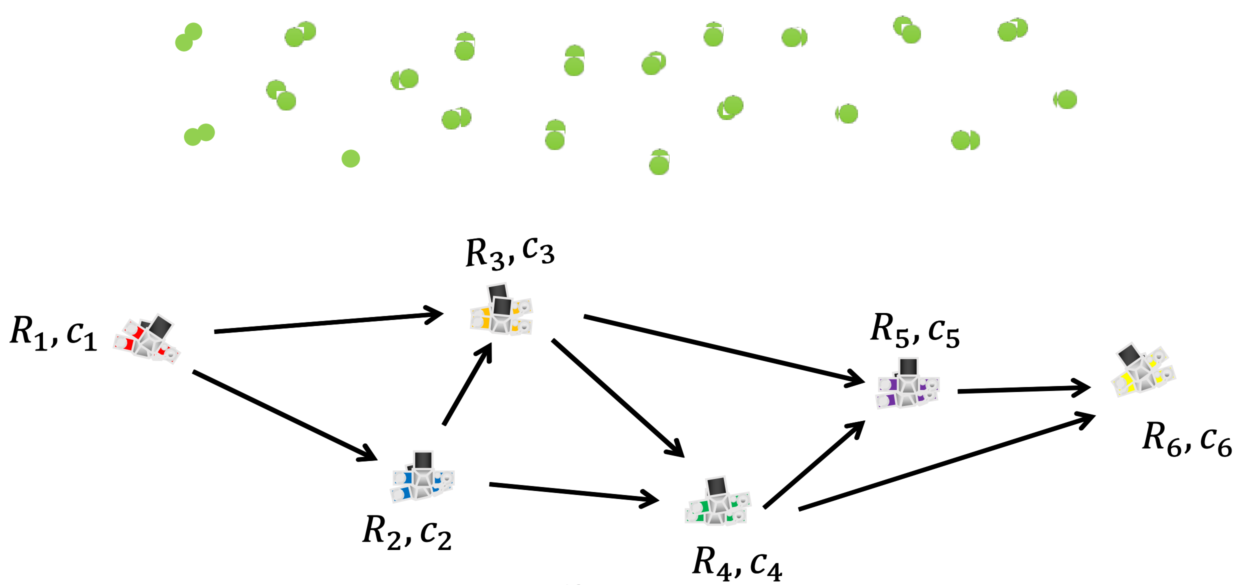
Global Structure-from-Motion

- Solve all pairwise camera motion (essential matrices, decomposition)
- Register all cameras simultaneously from input pairwise motions



Global Structure-from-Motion

- Solve all pairwise camera motion (essential matrices, decomposition)
- Register all cameras simultaneously from input pairwise motions
- Bundle adjustment only once



SFU

浙江大學 ZHEJIANG UNIVERSITY

Rotation Averaging

- Known relative rotation between two cameras

$$R_j = R^{ij} R_i$$

- Solving for R_i, R_j from all pairwise constraints
- In quaternion representation, $R_i = (r_i^1, r_i^2, r_i^3, r_i^4)$, therefore

$$\begin{pmatrix} r_j^1 \\ r_j^2 \\ r_j^3 \\ r_j^4 \end{pmatrix} = \begin{pmatrix} r_{ij}^1 & -r_{ij}^2 & -r_{ij}^3 & -r_{ij}^4 \\ r_{ij}^2 & r_{ij}^1 & -r_{ij}^4 & r_{ij}^3 \\ r_{ij}^3 & r_{ij}^4 & r_{ij}^1 & -r_{ij}^2 \\ r_{ij}^4 & -r_{ij}^3 & r_{ij}^2 & r_{ij}^1 \end{pmatrix} \begin{pmatrix} r_i^1 \\ r_i^2 \\ r_i^3 \\ r_i^4 \end{pmatrix}$$

$$\boldsymbol{r}_j = \mathcal{R}^{ij} \boldsymbol{r}_i$$

SFU

浙江大学 ZHEJIANG UNIVERSITY

Rotation Averaging

- Obtain a linear equations of $\boldsymbol{r}_i, \boldsymbol{r}_j$ for a pair (i, j)
$$\begin{bmatrix}\mathcal{R}^{ij} & -I\end{bmatrix}\begin{pmatrix}\boldsymbol{r}_i \\ \boldsymbol{r}_j\end{pmatrix} = 0$$
- Stack all equations, solve all $\boldsymbol{r}_i$ linearly
  - Ignore the unit quaternion constraint, i.e. $\left||\boldsymbol{r}_i|\right| = 1$
  - Normalize the result quaternions afterwards

Rotation Averaging

- Similar linear solution from matrix representation
- From $R_j = R^{ij} R_i$, $R_i = [r_i^1, r_i^2, r_i^3]$, $R_j = [r_j^1, r_j^2, r_j^3]$, obtain 3 equations

$$r_j^k = R^{ij} r_i^k \qquad k = 1, 2, 3$$

- Similarly,

$$\begin{bmatrix} R^{ij} & -I \end{bmatrix} \begin{pmatrix} r_i^k \\ r_j^k \end{pmatrix} = 0 \qquad k = 1, 2, 3$$

- Stack all equations, solve all r_i^k linearly
  - Ignore the orthogonal matrix constraint, i.e. $R_i^T R_i = I$
  - Normalize the result matrix afterwards

SFU

ZHEJIANG UNIVERSITY

Rotation Averaging

- Rotation averaging is still an open problem
- Most recent methods apply nonlinear optimization after the linear initialization

Robust Relative Rotation Averaging

Avishek Chatterjee and Venu Madhav Govindu

[PAMI 2017]

Translation Averaging

- Known relative rotation between two cameras

$$c_i - c_j = R_j^T t^{ij}$$

- Solving for c_i, c_j from all pairwise constraints
- Direct Linear Transform:

$$R_j^T t^{ij} \times (c_i - c_j) = 0$$

  - Problem 1: minimizing an algebraic error, faraway pairs are weighted more
  - Problem 2: cannot work on linear camera motion (i.e. all $(c_i - c_j)$ are colinear)

Essential matrices can only determine camera centers in a 'parallel rigid graph'

Robust Camera Location Estimation by Convex Programming

Onur Özyeşil[1] and Amit Singer[1,2]
[1]Program in Applied and Computational Mathematics, Princeton University
[2]Department of Mathematics, Princeton University
Princeton, NJ 08544-1000, USA
{oozyesil,amits}@math.princeton.edu

[CVPR 2015]

SFU

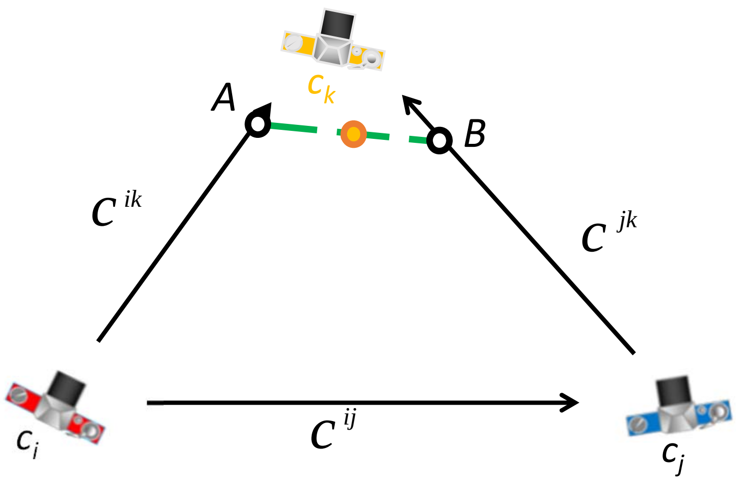
ZHEJIANG UNIVERSITY

Translation Averaging

A novel linear equation for three cameras from the 'mid-point' algorithm

$$c_k = \frac{1}{2}\left[\boxed{c_i + M_1(c_j - c_i)} + \boxed{c_j + M_2(c_i - c_j)}\right]$$

Similar linear equations for c_i and c_j



M_1, M_2 are both known matrices, computed from scene points.

$$\boxed{A = c_i + M_1(c_j - c_i)}$$

$$\boxed{B = c_j + M_2(c_i - c_j)}$$

AB: the mutual perpendicular line

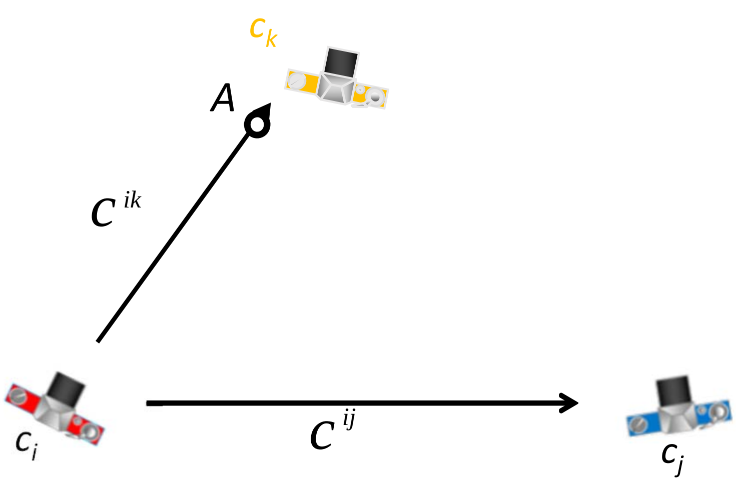
c_k: the middle point of AB

Translation Averaging

Geometric meaning of M_1

$$c_k = \frac{1}{2}\left[\boxed{c_i + M_1(c_j - c_i)} + \boxed{c_j + M_2(c_i - c_j)}\right]$$

1. rotate to match the orientation
2. shrink/grow to match the length



$$\boxed{A = c_i + M_1(c_j - c_i)}$$

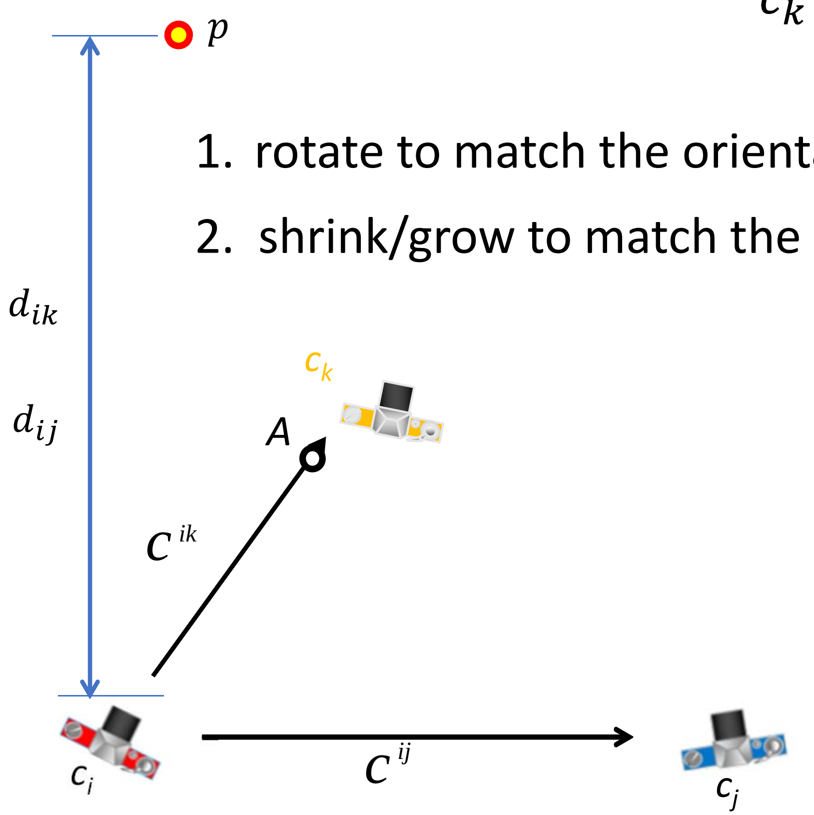
SFU

浙江大學 ZHEJIANG UNIVERSITY

Translation Averaging

Geometric meaning of M_1

$$c_k = \frac{1}{2}\left[\boxed{c_i + M_1(c_j - c_i)} + \boxed{c_j + M_2(c_i - c_j)}\right]$$



1. rotate to match the orientation ➔ Known from essential matices
2. shrink/grow to match the length ➔ Known from a scene point

$$\frac{|c_i - c_j|}{|c_i - c_k|} = \frac{d_{ik}}{d_{ij}}$$

The ratio of a scene point's depths

d_{ik} is p's depth when reconstructed by the pair (i, k).

d_{ij} is p's depth when reconstructed by the pair (i, j).

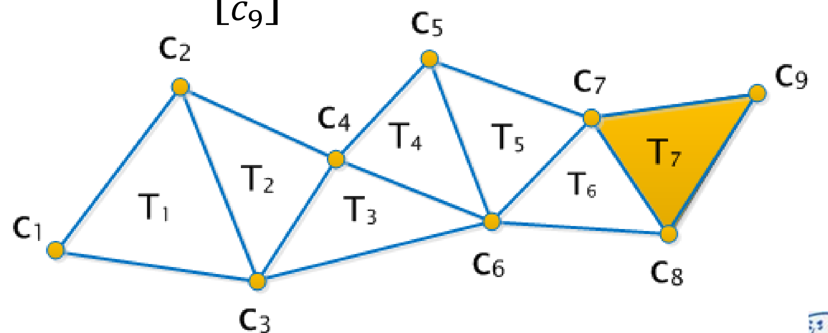
Translation Averaging

1. Collect equations from all triangles in the pose graph.

$$\boxed{AX = 0} \qquad X = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \\ c_9 \end{bmatrix}$$

The pose graph:
each camera is a vertex,
connect two cameras if their
relative motion is known.

2. Solve all equations



$A_1(c_1, c_2, c_3)^T = 0$

cameras can be non-coplanar.

Translation Averaging

- More details in the papers

A Global Linear Method for Camera Pose Registration

Nianjuan Jiang[1,*] Zhaopeng Cui[2,*] Ping Tan[2]
[1]Advanced Digital Sciences Center, Singapore [2]National University of Singapore

[ICCV 2013]

Questions?