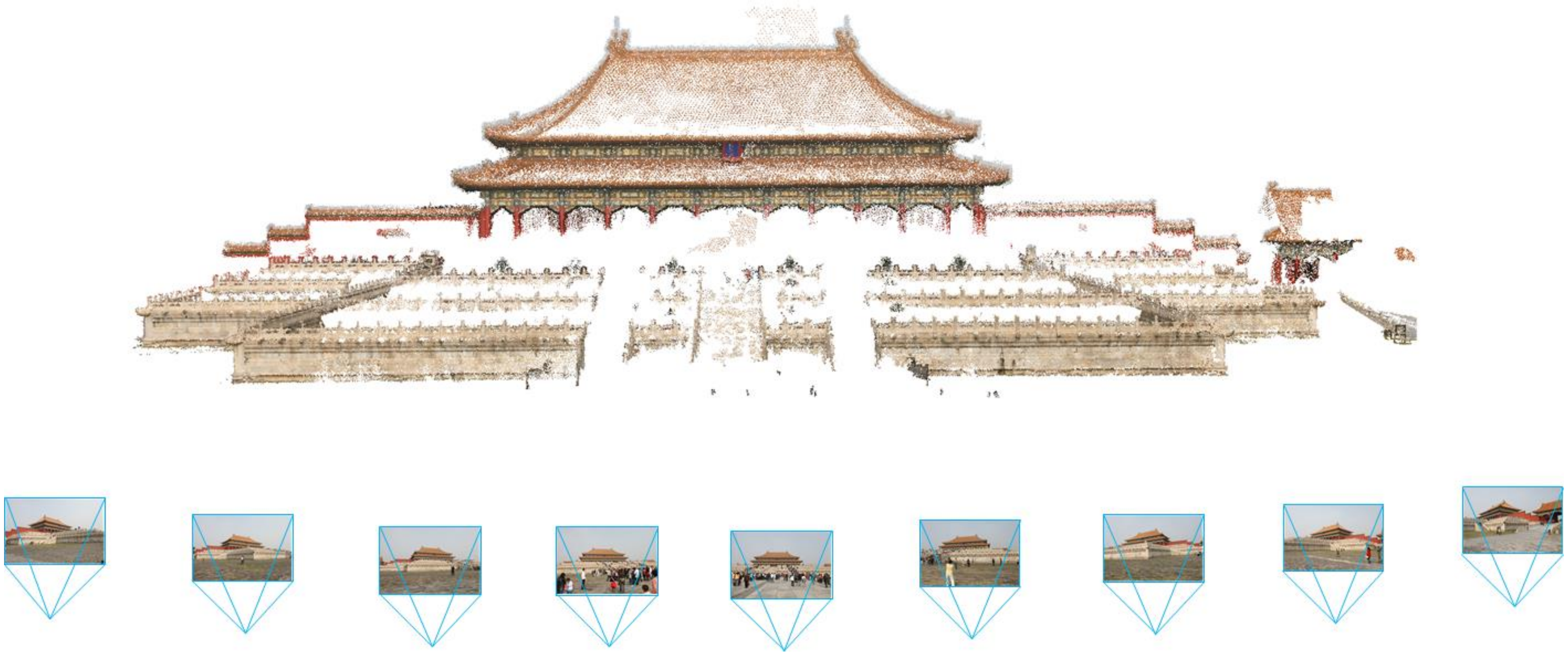
11. Structure-from-Motion

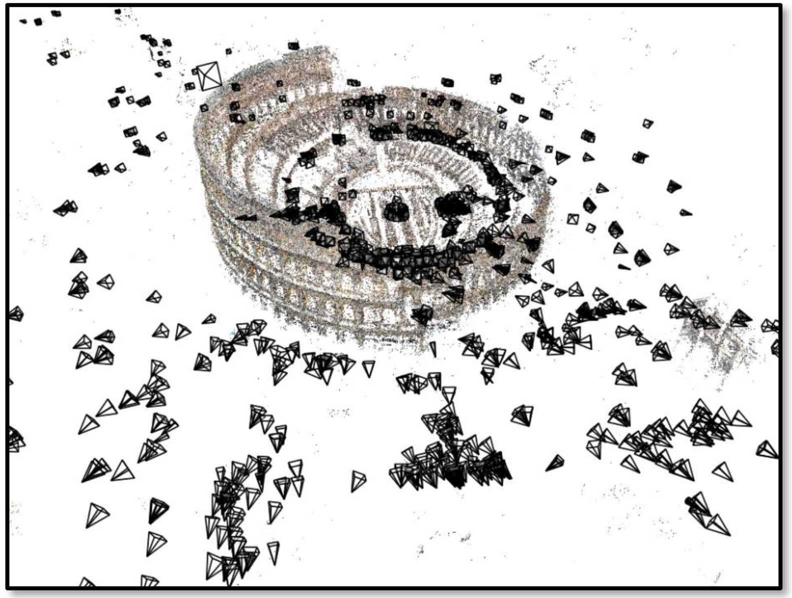


Outline

- Bundle Adjustment
- Initializing BA

Structure-from-Motion

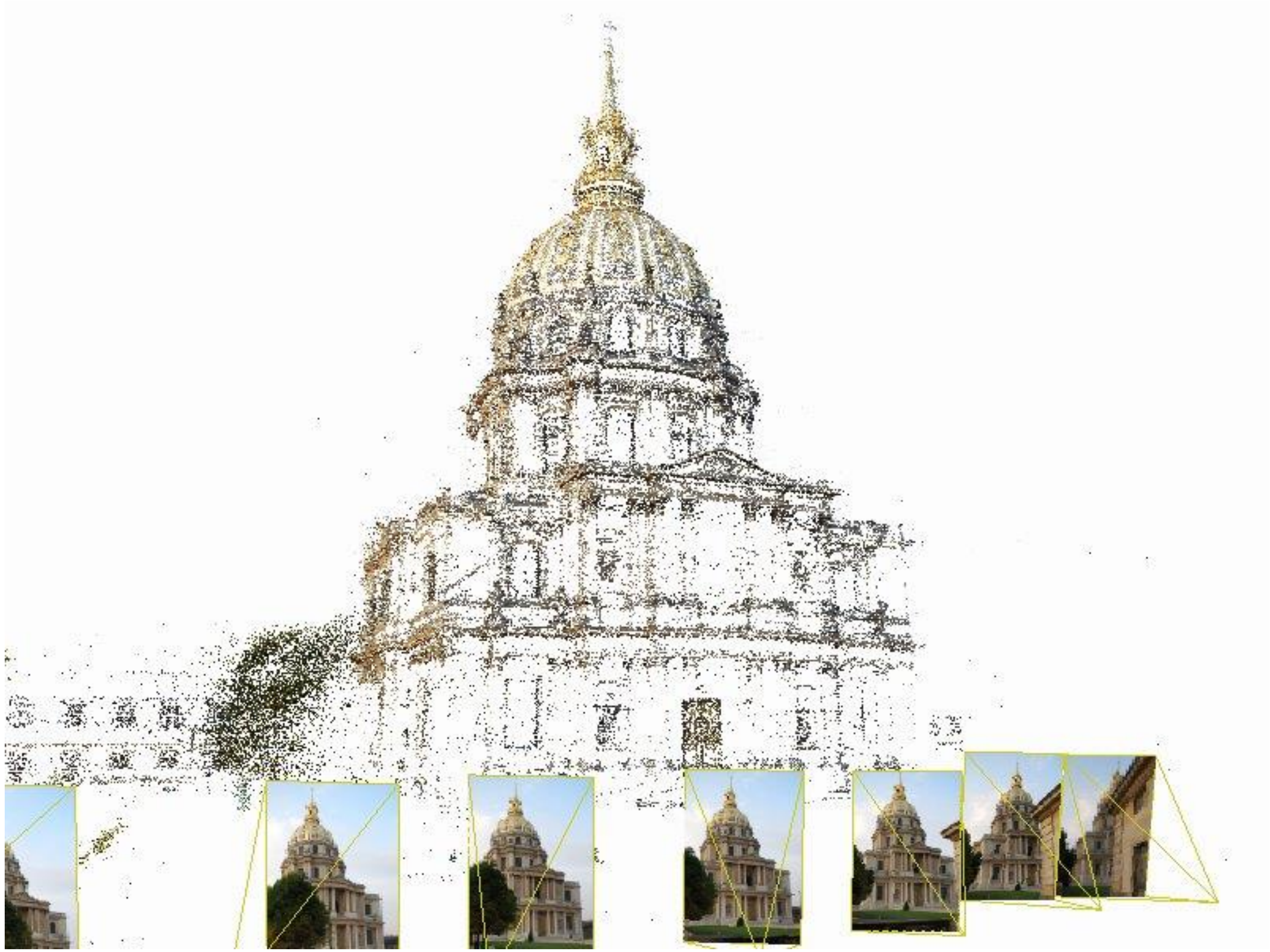
- Given many images, how can we
  a) figure out where they were all taken from?
  b) build a 3D model of the scene?



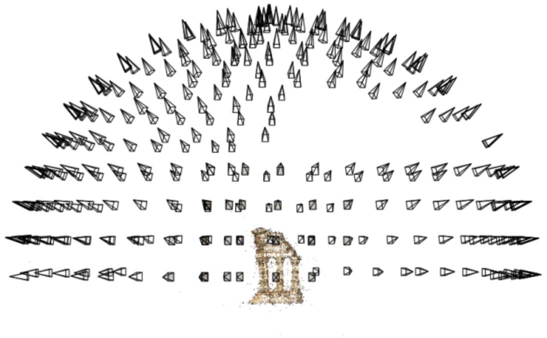
Structure-from-Motion

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

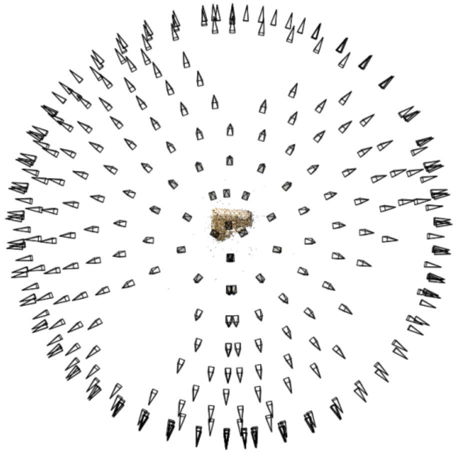
Also Doable from Videos



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 $\mathbf{R}_j, \mathbf{t}_j$ possibly $\mathbf{K}_j$
- Objective function: minimize *reprojection error*

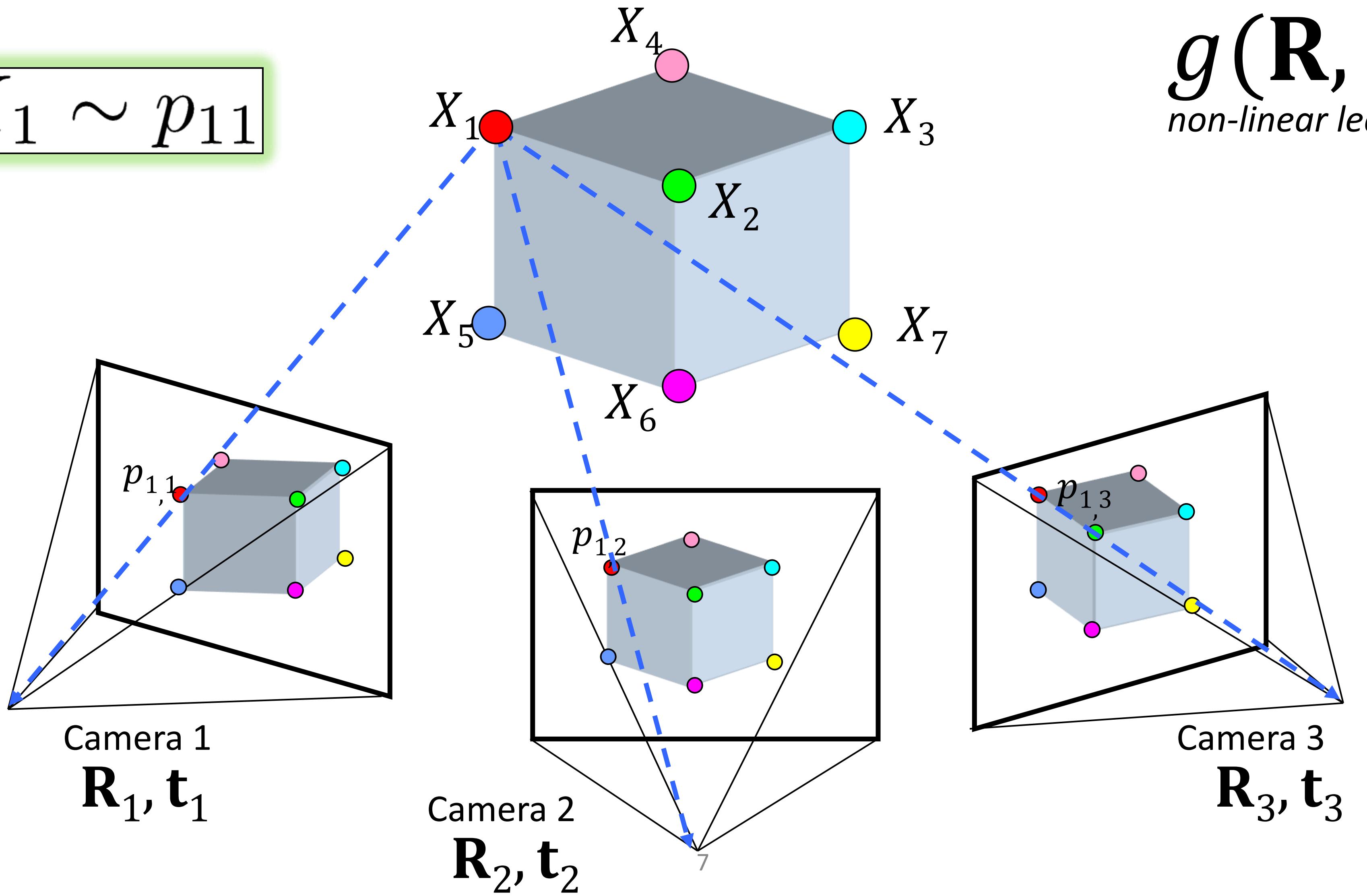
Formulation

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

minimize

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

non-linear least squares



Formulation

- Minimize sum of squared reprojection errors:

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

- 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:

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

$$= \sum_{ij} e_{ij}^2(\mathbf{x}_i, \mathbf{R}_j, \mathbf{t}_j, \mathbf{K}_j)$$

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

Gauss-Newton Method Revisit

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

$$f(P_0 + \Delta) \approx f(P_0) + \mathrm{J}\Delta, \quad \mathrm{J} = \frac{\partial f}{\partial P}$$

- 2. minimize the linearized objective function

$$\Delta = \arg\min \|f(P_0) + \mathrm{J}\Delta\|^2$$
$$\Rightarrow \mathrm{J}^{\mathrm{T}}\mathrm{J}\Delta = \mathrm{J}^{\mathrm{T}} f(P_0)$$

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

$$P_{\mathrm{i}+1} = P_{\mathrm{i}} + \Delta$$

- 4. iterate 1-3 until converge

Linearize the re-projection error

- Error function: $f(P) = g(\mathbf{X},\mathbf{R},\mathbf{T}) = \sum_{ij} e_{ij}^2(\mathbf{X},\mathbf{R},\mathbf{T})$
  - $e_{ij} = \mathbf{P}(\mathbf{x}_i, \mathbf{R}_j, \mathbf{t}_j) - p_{ij}$

- 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}:

$$J_{ij}(\mathbf{X},\mathbf{R},\mathbf{T}) = \left(0,\cdots,\frac{\partial e_{ij}}{\partial \mathbf{x}_i},\cdots,\frac{\partial e_{ij}}{\partial \mathbf{R}_j},\frac{\partial e_{ij}}{\partial \mathbf{t}_j},\cdots,0\right)$$

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:

$$P \leftarrow P + \Delta$$

Gauss-Newton Algorithm

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

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

- 2. Solve the linear systems by

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

- 3. Update the previous results by:

$$P \leftarrow P + \Delta$$

The Hessian

- The Hessian $\boldsymbol{H}$ is
  - Positive semi-definite
  - Symmetric
  - Sparse
- This allows efficient solution
  - Schur Complement

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

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

- 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:

$$P \leftarrow P + \Delta$$

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?

Outline

- Bundle Adjustment
- Initializing BA

Initializing the Bundle Adjustment

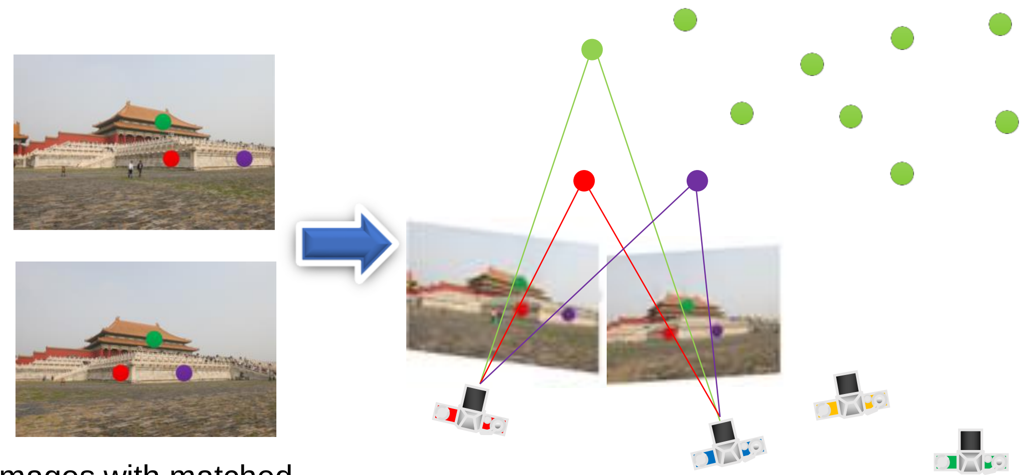
- Levenberg-Marquardt algorithm requires good initial guess for:
  - 3D points X_i
  - camera parameters $\mathbf{R}_j, \mathbf{t}_j, \mathbf{K}_j$

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

Incremental Structure-from-Motion

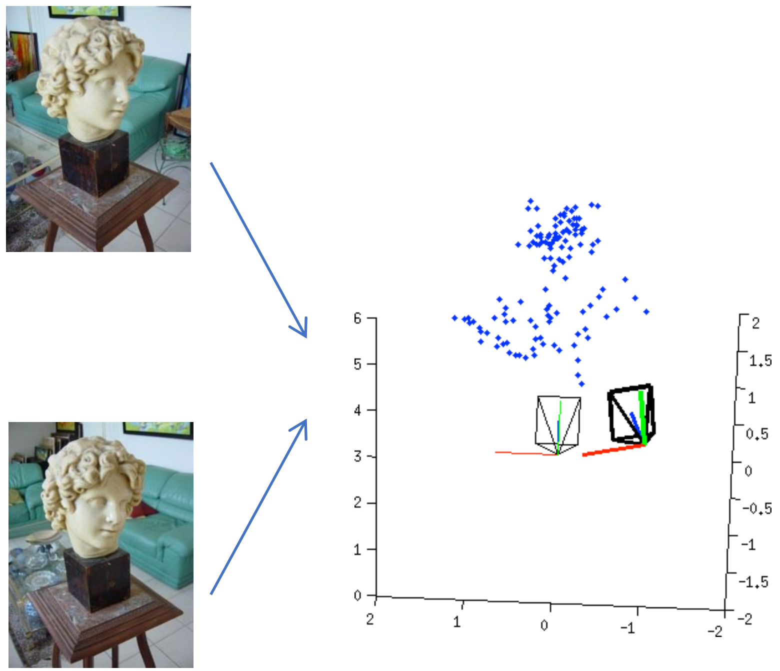
1. Solve a two-view reconstruction (essential matrix, decomposition, triangulation)
2. Add cameras by resection with 3D-2D correspondences (resection, PnP)
   Might triangulate more points from the newly added cameras (triangulation)

Step 1: Epipolar Geometry → Initial Reconstruction (2 cameras) → Add Cameras

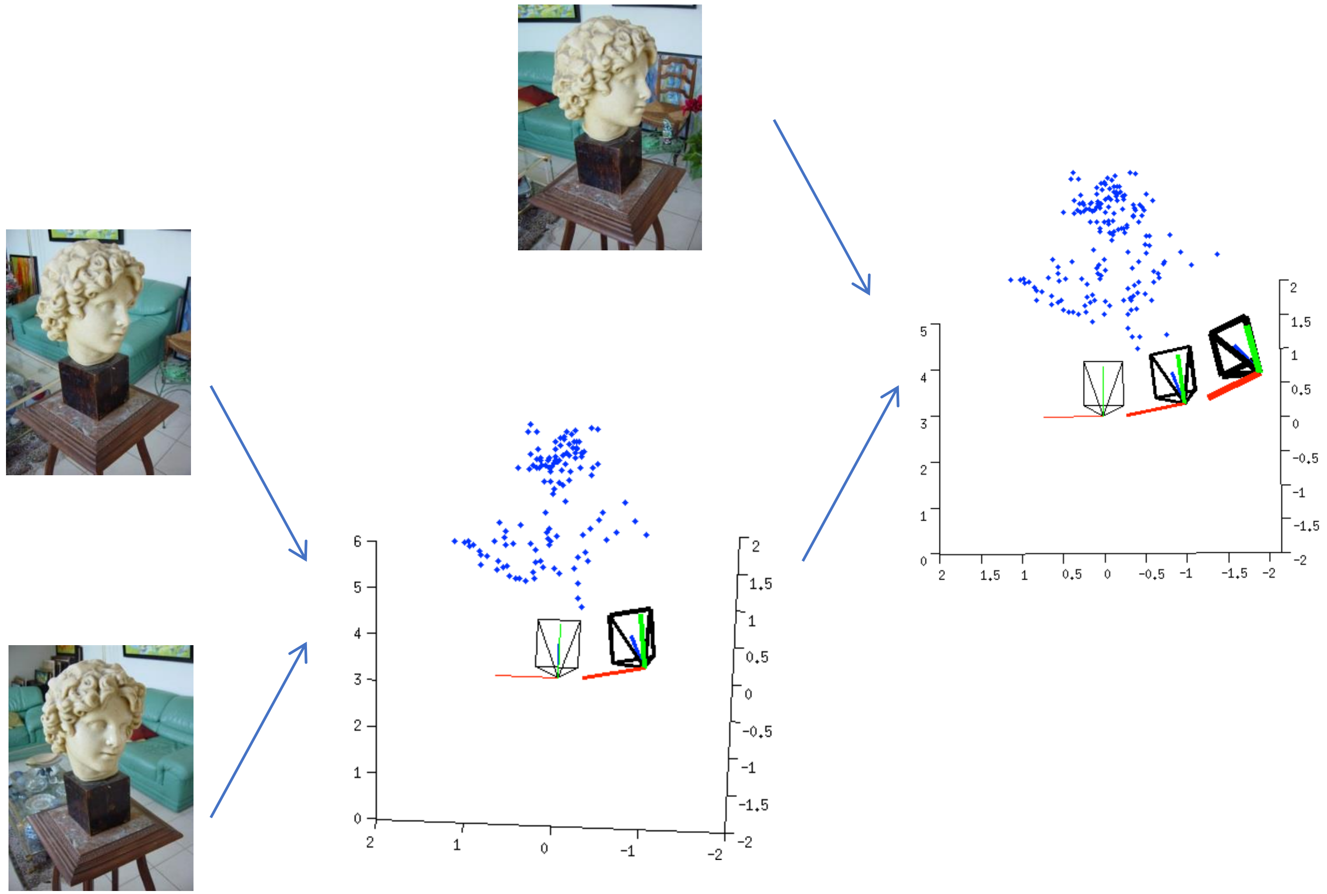


Images with matched feature points

two-view reconstruction

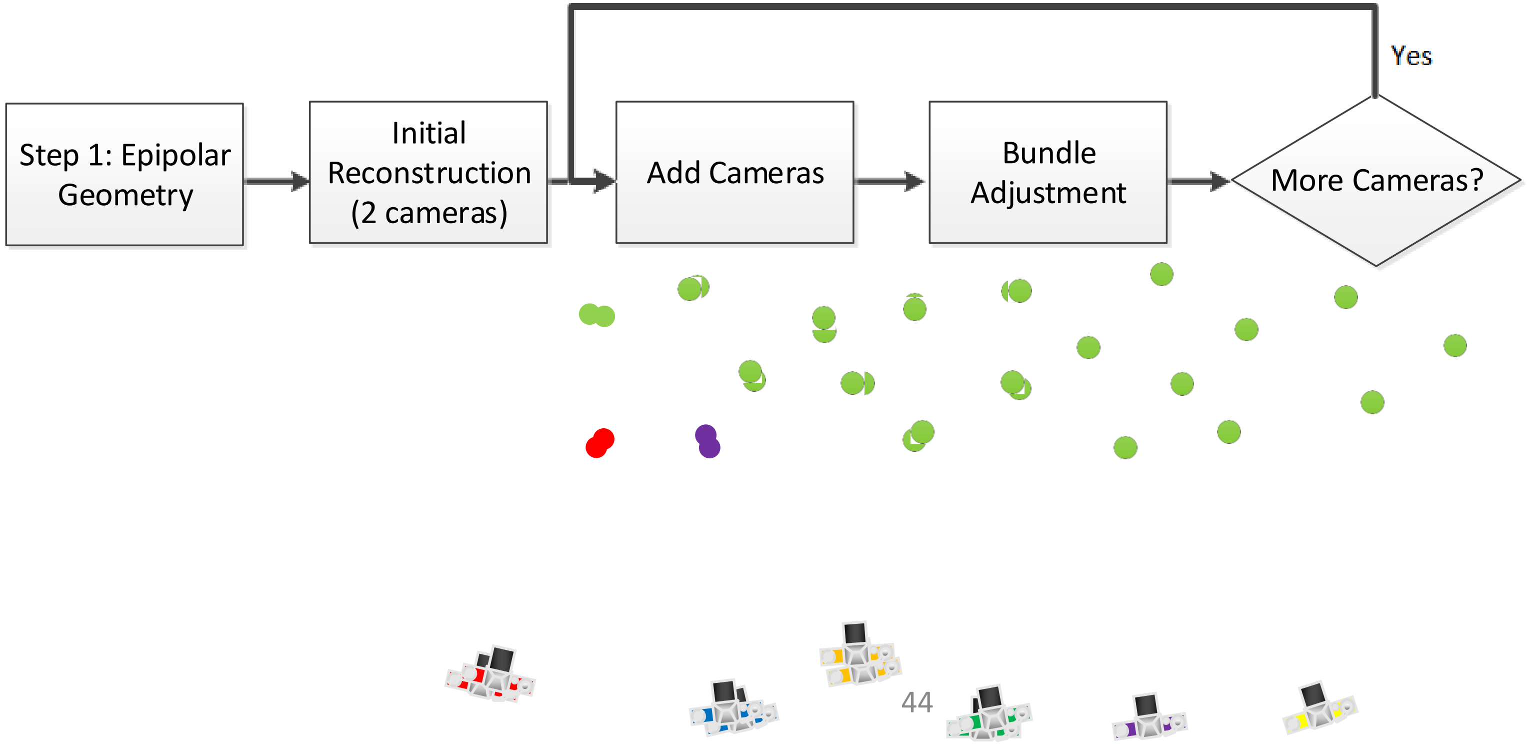


incrementally add the third view



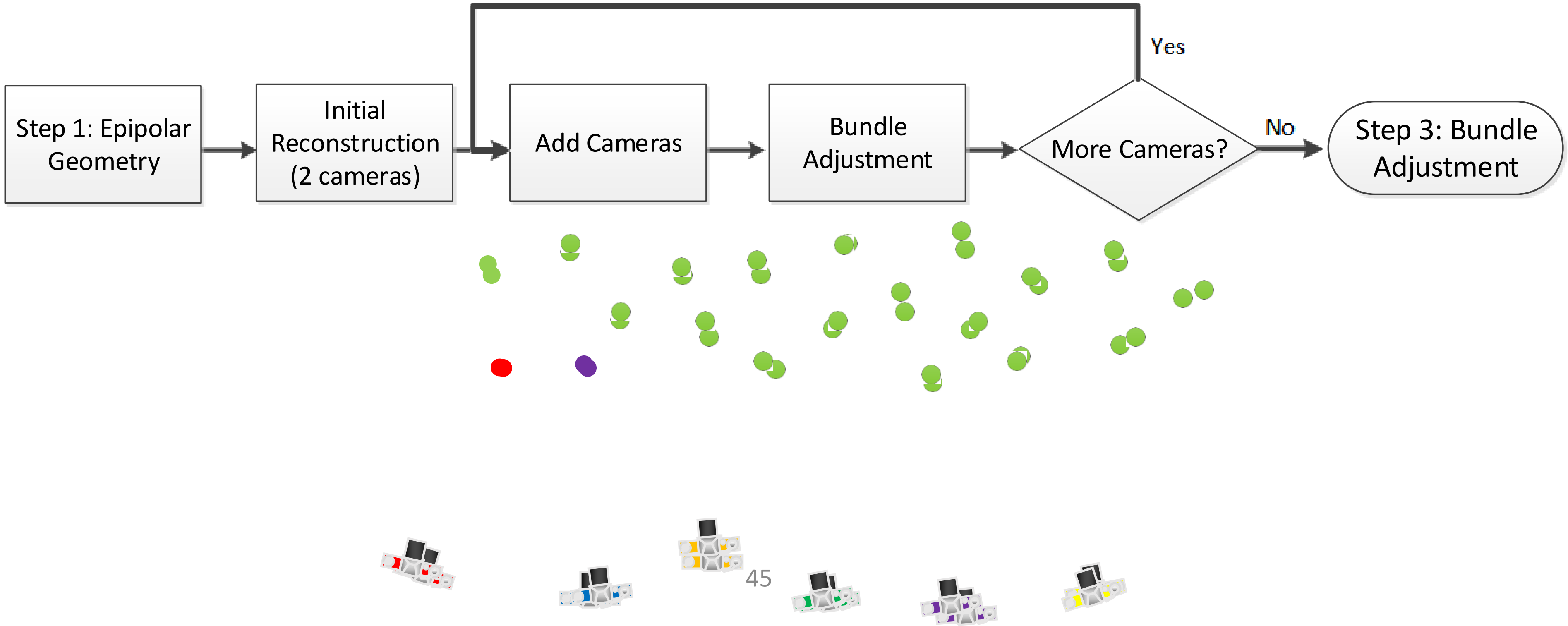
Incremental Structure-from-Motion

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 (triangulation)
3. Repeat step-2 (with intermediate BA to reduce error accumulation)

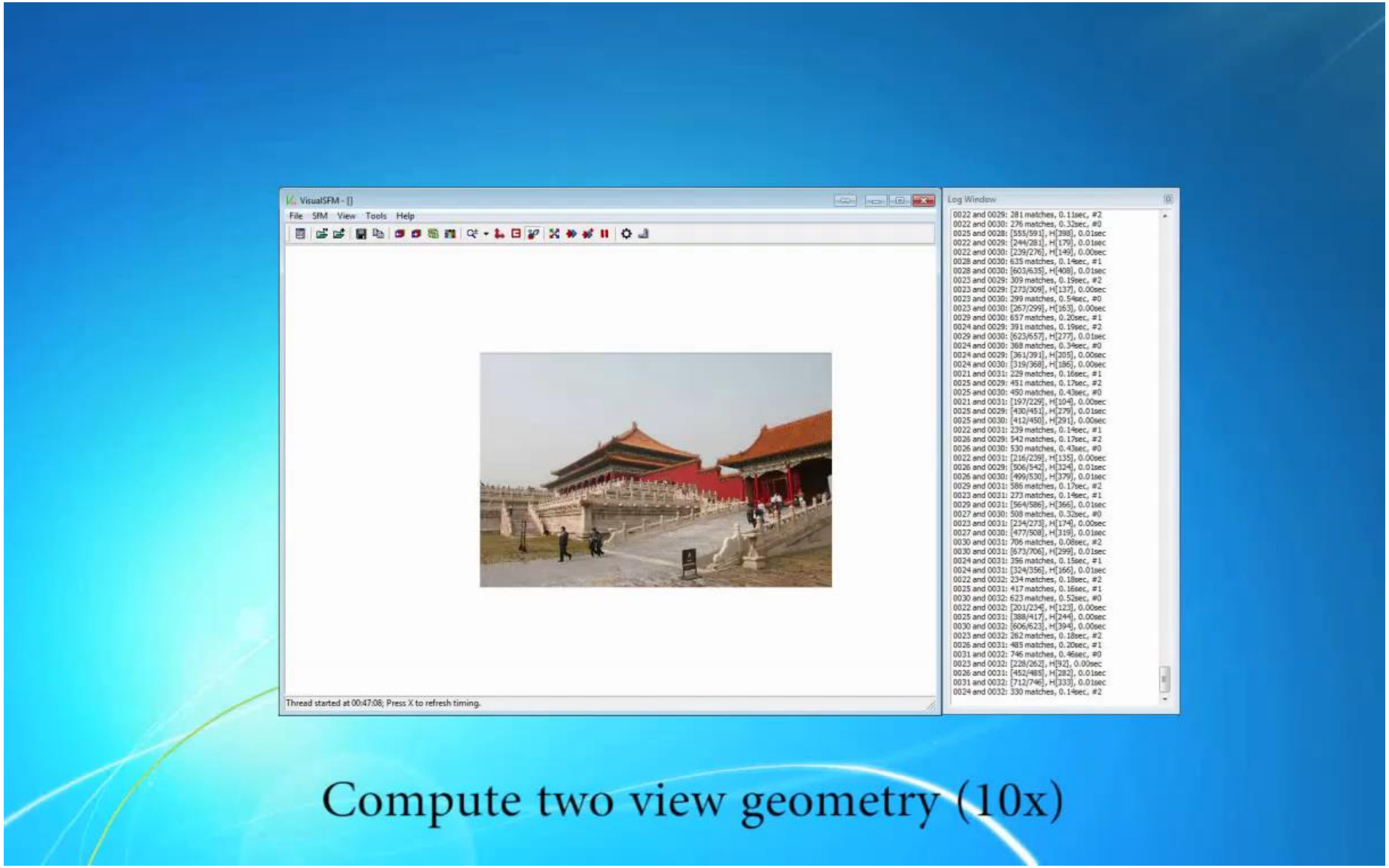


Incremental Structure-from-Motion

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 (triangulation)
3. Repeat step-2 (with intermediate BA to reduce error accumulation)



Incremental Structure-from-Motion



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.

Drawbacks of Incremental SfM

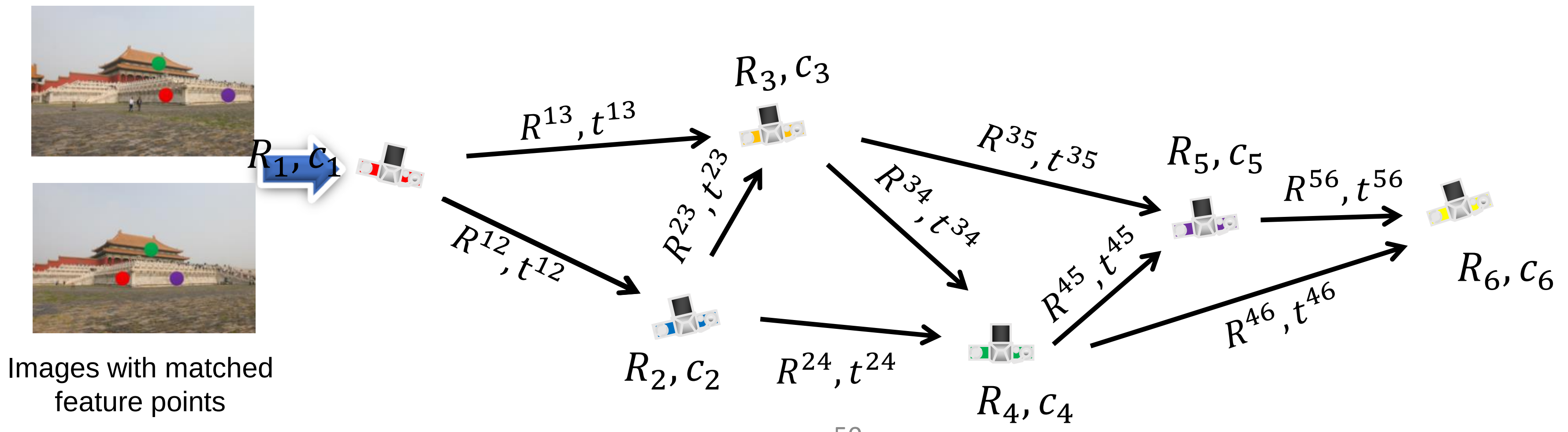
- Poor run-time efficiency
  - Repetitively solving the nonlinear bundle adjustment (though locally)
  - Most of the computation time is spent on bundle adjustment

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

Questions?

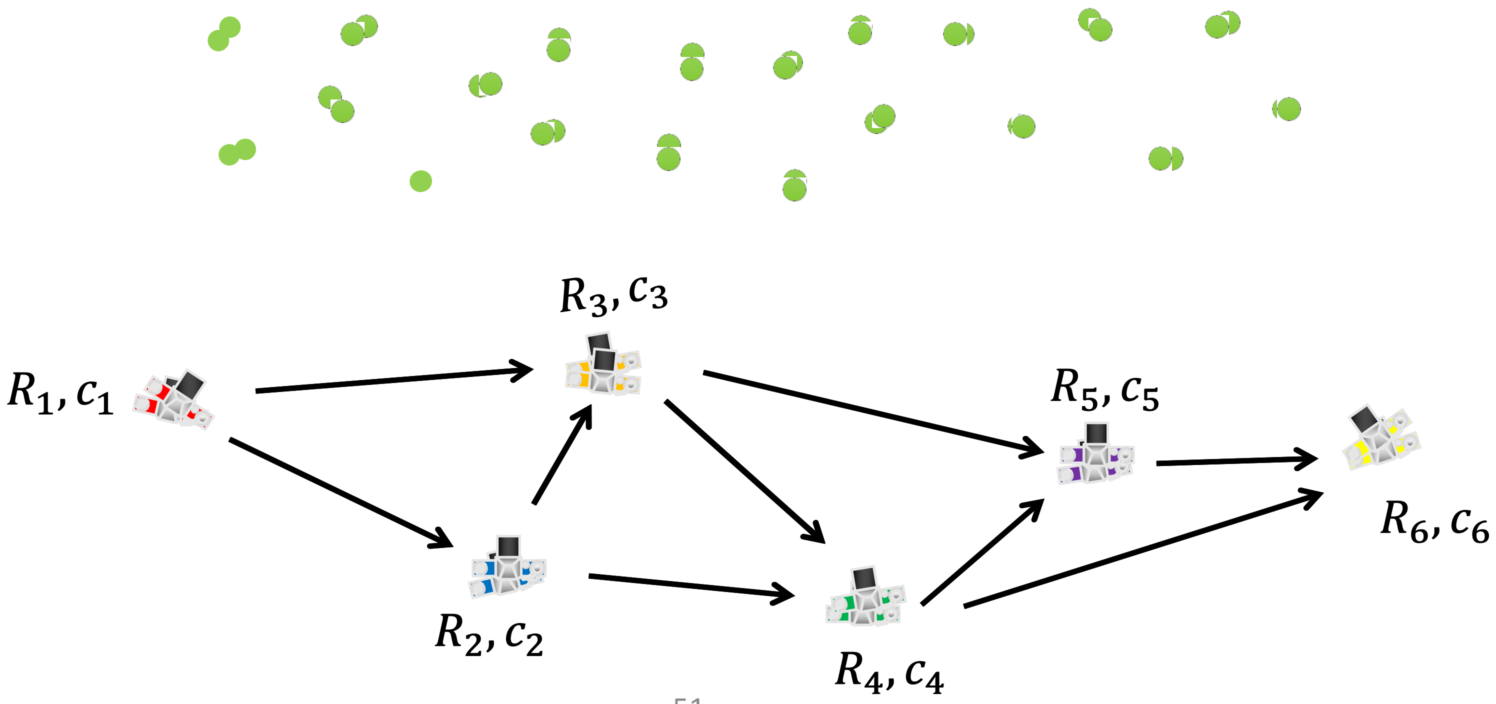
Global Structure-from-Motion

- 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



Rotation Averaging

- Known relative rotation between two cameras
$$\mathbf{R}_j = \mathbf{R}^{ij}\mathbf{R}_i$$
- Solving for $\mathbf{R}_i, \mathbf{R}_j$ from all pairwise constraints
- In quaternion representation, $\mathbf{R}_i = (r_i^1, r_i^2, r_i^3, r_i^4)$, therefore

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

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

Rotation Averaging

- Obtain a linear equations of $\boldsymbol{r}_i, \boldsymbol{r}_j$ for a pair (i, j)

$$\left[\mathcal{R}^{ij} \quad -I\right]\begin{pmatrix}\boldsymbol{r}_i \\ \boldsymbol{r}_j\end{pmatrix} = 0$$

- Stack all equations, solve all $\boldsymbol{r}_i$ linearly
  - Ignore the unit quaternion constraint, i.e. $||\boldsymbol{r}_i|| = 1$
  - Normalize the result quaternions afterwards

Rotation Averaging

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

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

- Similarly,

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

- Stack all equations, solve all r_i^k linearly
  - Ignore the orthogonal matrix constraint, i.e. $\mathbf{R}_i^{\mathrm{T}}\mathbf{R}_i = \mathrm{I}$
  - Normalize the result matrix afterwards

Rotation Averaging

- Rotation averaging is still an open problem
- Most recent methods apply nonlinear optimization

Robust Relative Rotation Averaging

Avishek Chatterjee and Venu Madhav Govindu

[PAMI 2017]

Rotation Averaging

Richard Hartley · Jochen Trumpf · Yuchao Dai · Hongdong Li

[IJCV 2013]

Translation Averaging

- Known relative rotation between two cameras

$$\mathbf{c}_i - \mathbf{c}_j = \mathbf{R}_j^T \mathbf{t}^{ij}$$

- Solving for $\mathbf{c}_i, \mathbf{c}_j$ from all pairwise constraints
- Direct Linear Transform:

$$\mathbf{R}_j^T \mathbf{t}^{ij} \times \left(\mathbf{c}_i - \mathbf{c}_j\right) = 0$$

  - Problem 1: minimizing an algebraic error, faraway pairs are weighted more
  - Problem 2: cannot work on linear camera motion (i.e. all $(\mathbf{c}_i - \mathbf{c}_j)$ 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

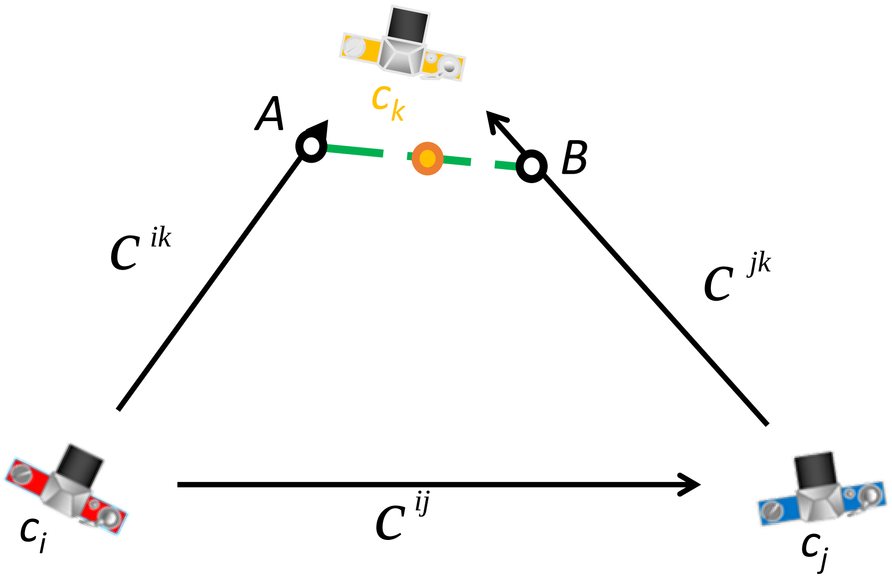
[CVPR 2015]

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

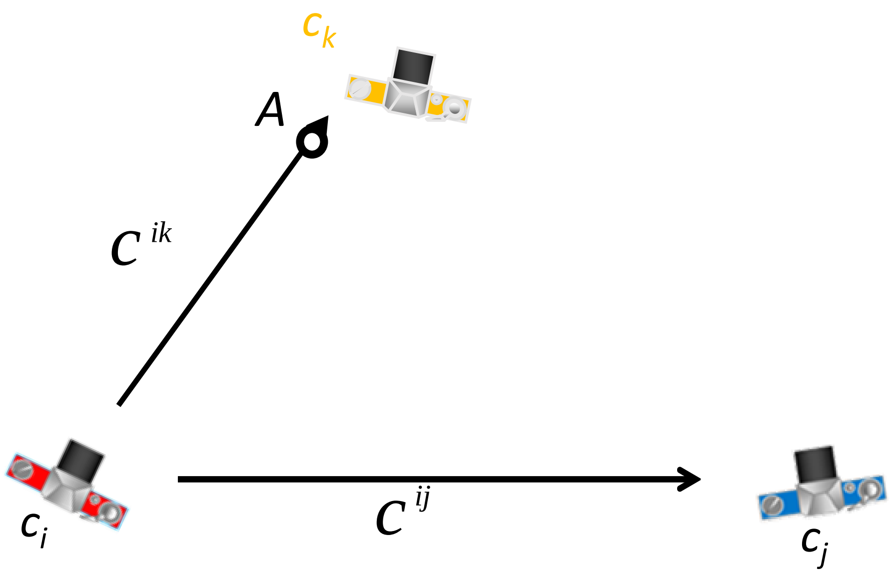
c_k: the middle point of AB

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

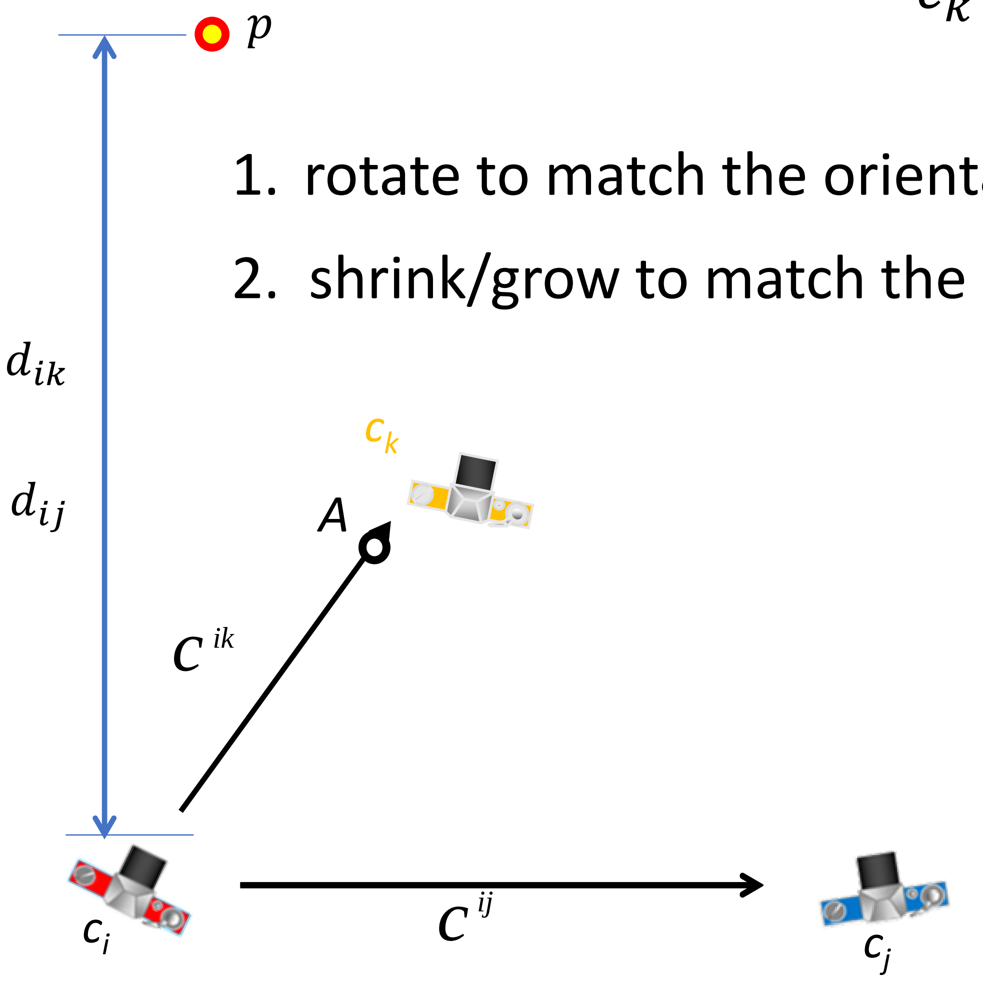


$$\boxed{A = c_i + M_1(c_j - c_i)}$$

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).

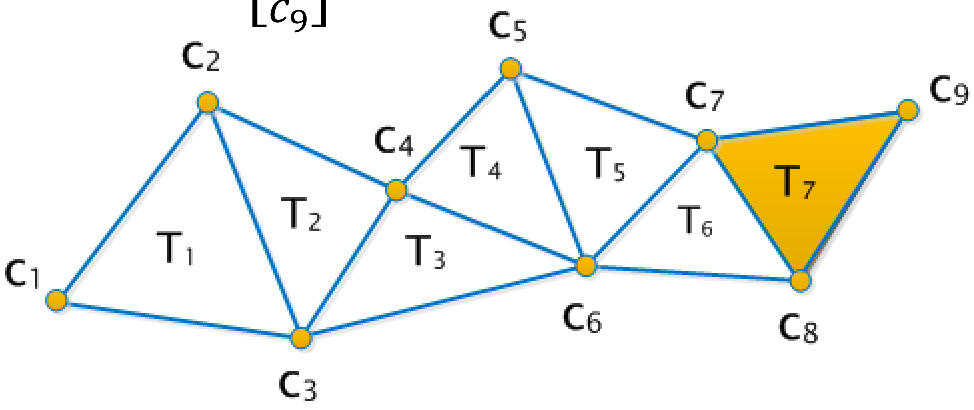
Translation Averaging

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?