NUS National University of Singapore | School of Computing

CS4277 / CS5477
3D Computer Vision

Lecture 10:
Structure-from-Motion (SfM) and Bundle Adjustment

Assoc. Prof. Lee Gim Hee

AY 2022/23

Semester 2

Course Schedule

Week	Date	Topic	Assignments
1	11 Jan	2D and 1D projective geometry	**Assignment 0:** Getting started with Python (Ungraded)
2	18 Jan	3D projective geometry, Circular points and Absolute conic	
3	25 Jan	Rigid body motion and Robust homography estimation	
4	01 Feb	Camera models and calibration	**Assignment 1:** Metric rectification and robust homography (10%) **Due:** 2359hrs, 07 Feb
5	08 Feb	Single view metrology	**Assignment 2**: Affine 3D measurement from vanishing line and point (10%) **Due:** 2359hrs, 14 Feb
6	15 Feb	The Fundamental and Essential matrices	
-	22 Feb	Semester Break	No lecture
7	**01 Mar**	**Mid-term Quiz (20%)** Lecture: Generalized cameras	**In-person Quiz (LT 15, 1900hrs – 2000hrs)** Lecture: 2000hrs – 2130hrs
8	08 Mar	Absolute pose estimation from points or lines	
9	15 Mar	Three-view geometry from points and/or lines	
10	22 Mar	Structure-from-Motion (SfM) and bundle adjustment	**Assignment 3:** SfM and Bundle adjustment (10%) **Due:** 2359hrs, 28 Mar
11	29 Mar	Two-view and multi-view stereo	**Assignment 4:** Dense 3D model from multi-view stereo (10%) **Due:** 2359hrs, 04 Apr
12	05 Apr	3D Point Cloud Processing	
13	12 Apr	Neural Field Representations	

Final Exam: 03 MAY 2023

Learning Outcomes

- Students should be able to:

1. Describe the pipeline of large-scale 3D reconstruction: data association, structure-from-motion and dense stereo.
2. Explain the use of robust two-view geometry and the bag-of-words algorithm for data association.
3. Use two-view geometry, PnP and triangulation to initialize the 3D reconstruction.
4. Apply the iterative methods: Newton, Gauss-Newton Gradient descent or Levenberg-Marquardt for bundle adjustment.

Acknowledgements

- A lot of slides and content of this lecture are adopted from:

1. Tutorial on large-scale 3D modeling, CVPR 2017 https://demuc.de/tutorials/cvpr2017/
2. R. Hartley, and A. Zisserman: "Multiple view geometry in computer vision", Appendix 6.
3. Bill Triggs et al, "Bundle Adjustment – A Modern Synthesis", 1999
4. Y. Ma, S. Soatto, J. Kosecka, S. S. Sastry, " An invitation to 3-D vision", Chapter 11.

Large-Scale 3D Reconstruction

- **Given:** a set of images $\{\mathrm{I}_1, \ldots \mathrm{I}_N\}$ taken from different viewpoints.

- **Find:**

1. The motions of the cameras w.r.t. a world coordinate frame F_W, i.e. the camera projection matrices $\{\mathrm{P}_1, \ldots, \mathrm{P}_N\}$.

2. And then recover the 3D structures $\{\mathbf{X}_1, \ldots, \mathbf{X}_M\}$ of the scene.

Large-Scale 3D Reconstruction: Pipeline

- Three key parts of the pipeline:

1. **Data association:** check whether a pair of images are related using image correspondences and robust two-view geometry.

2. **Structure-from-Motion (SfM):** initial reconstruction from relative/absolute pose estimation and triangulation, and then refine with bundle adjustment.

3. **Multi-view stereo (MVS):** get the dense 3D model from the plane sweeping algorithm (next lecture).

Large Scale 3D Reconstruction: Pipeline

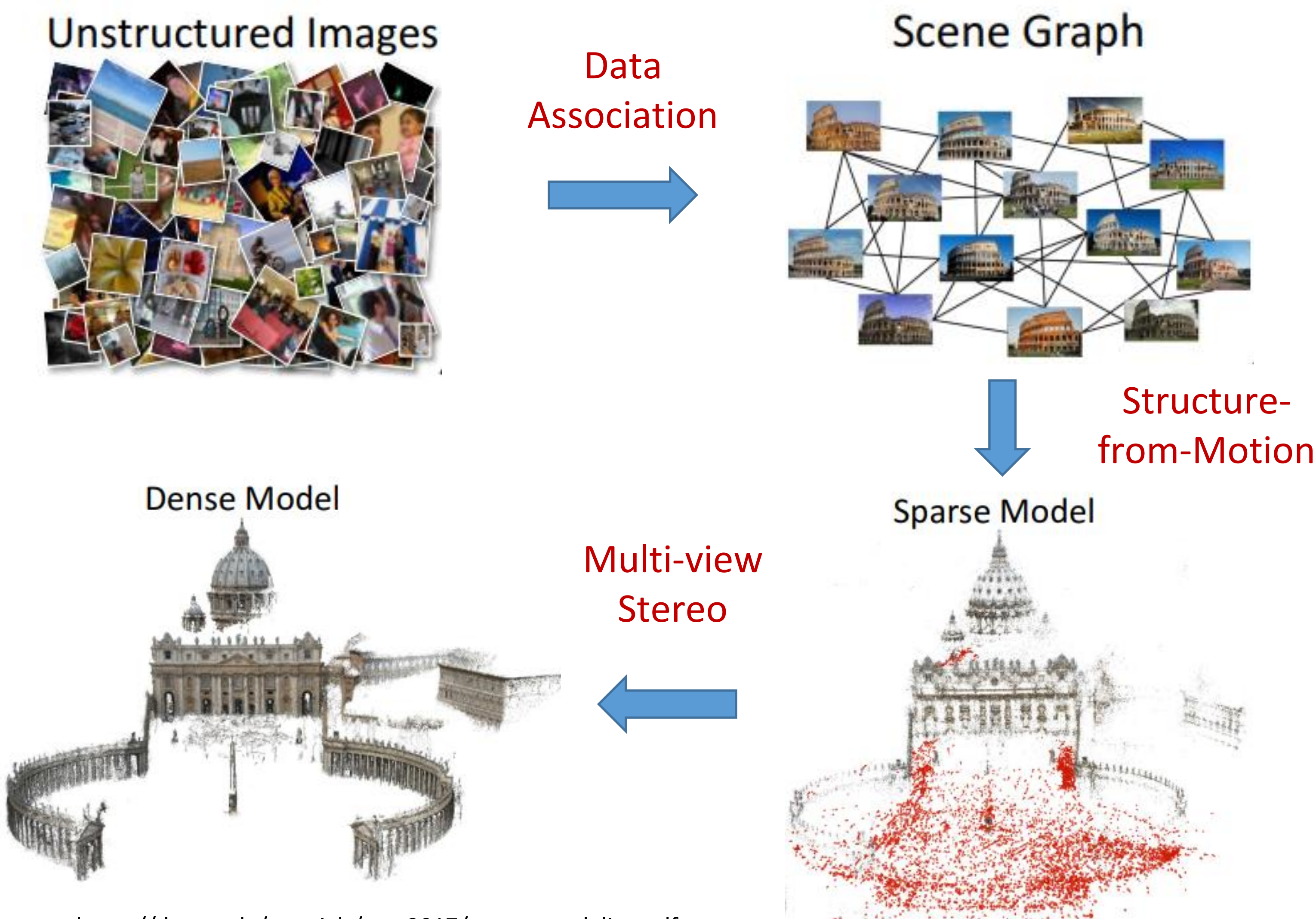


Image source: https://demuc.de/tutorials/cvpr2017/sparse-modeling.pdf

Structure-from-Motion

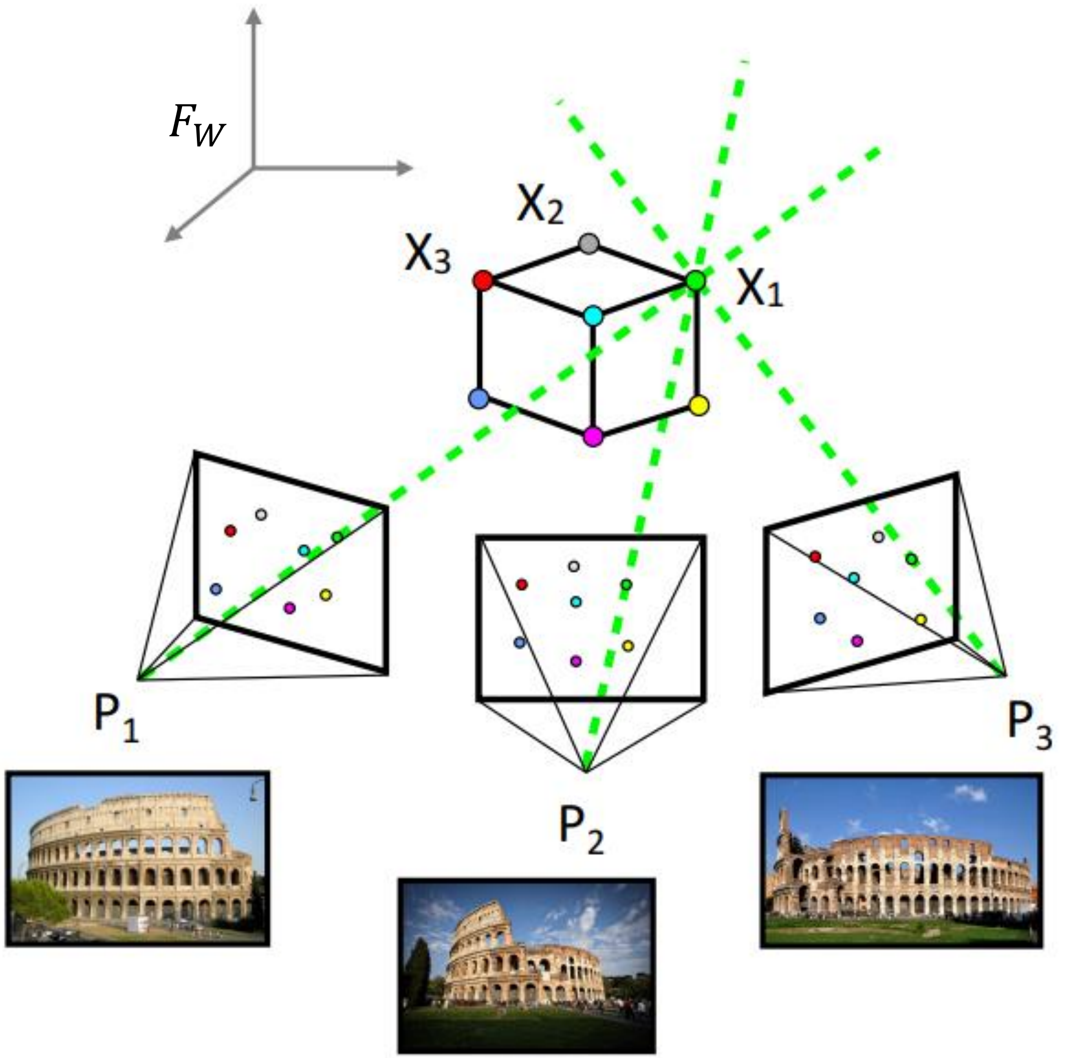
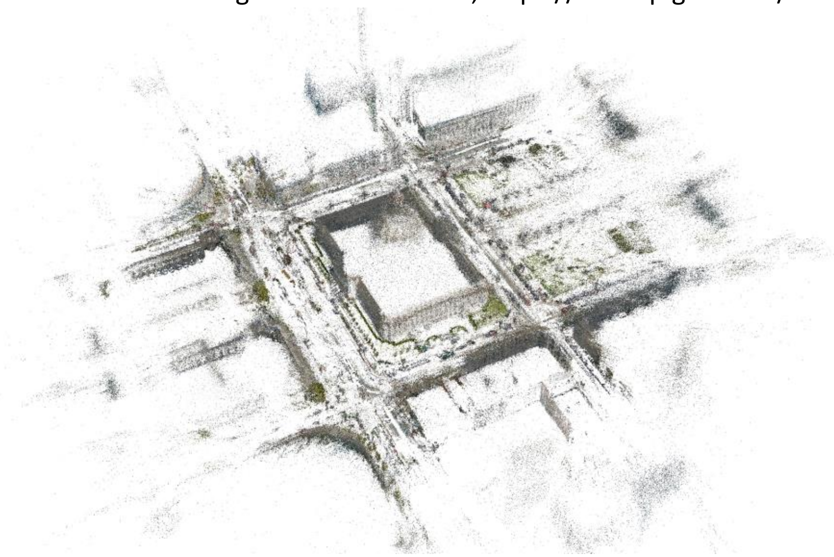


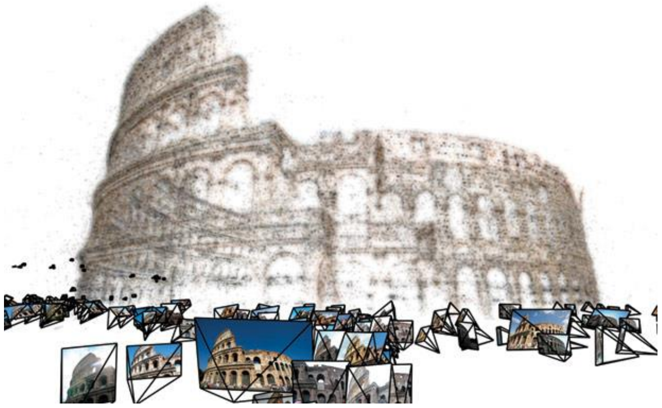
Image source: https://demuc.de/tutorials/cvpr2017/sparse-modeling.pdf

Structure-from-Motion

Image Source: COLMAP, https://colmap.github.io/



O. Saurer, M. Pollefeys, G. H. Lee, CVPR 2016



S. Agarwal, N. Snavely, I. Simon, S. Seitz and R. Szeliski, ICCV 2009

Data Association

- We get the scene graph by finding images with overlapping views, i.e. connected components.

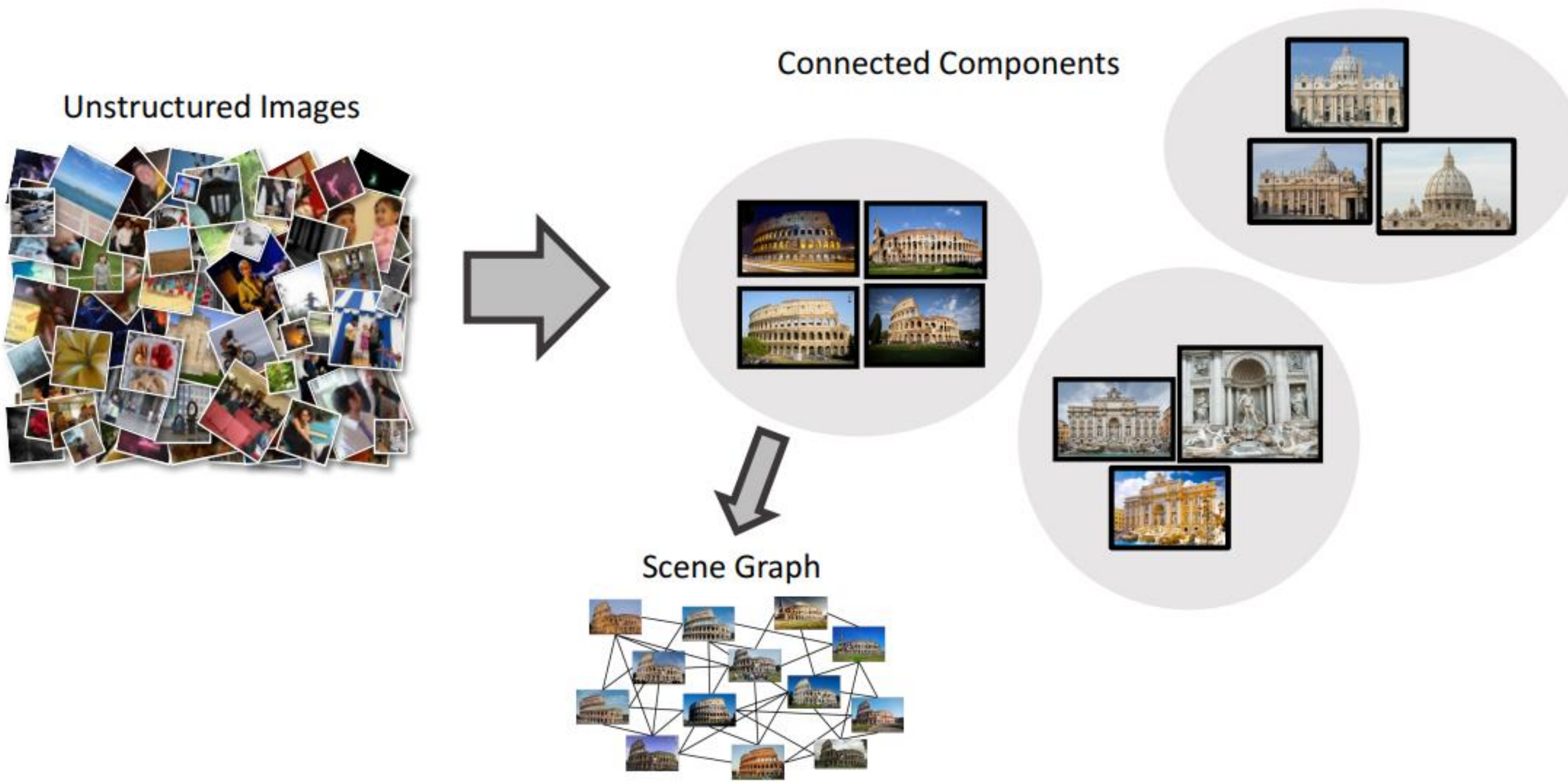


Image source: https://demuc.de/tutorials/cvpr2017/sparse-modeling.pdf

Data Association

- Use two-view geometry to establish the connected components.

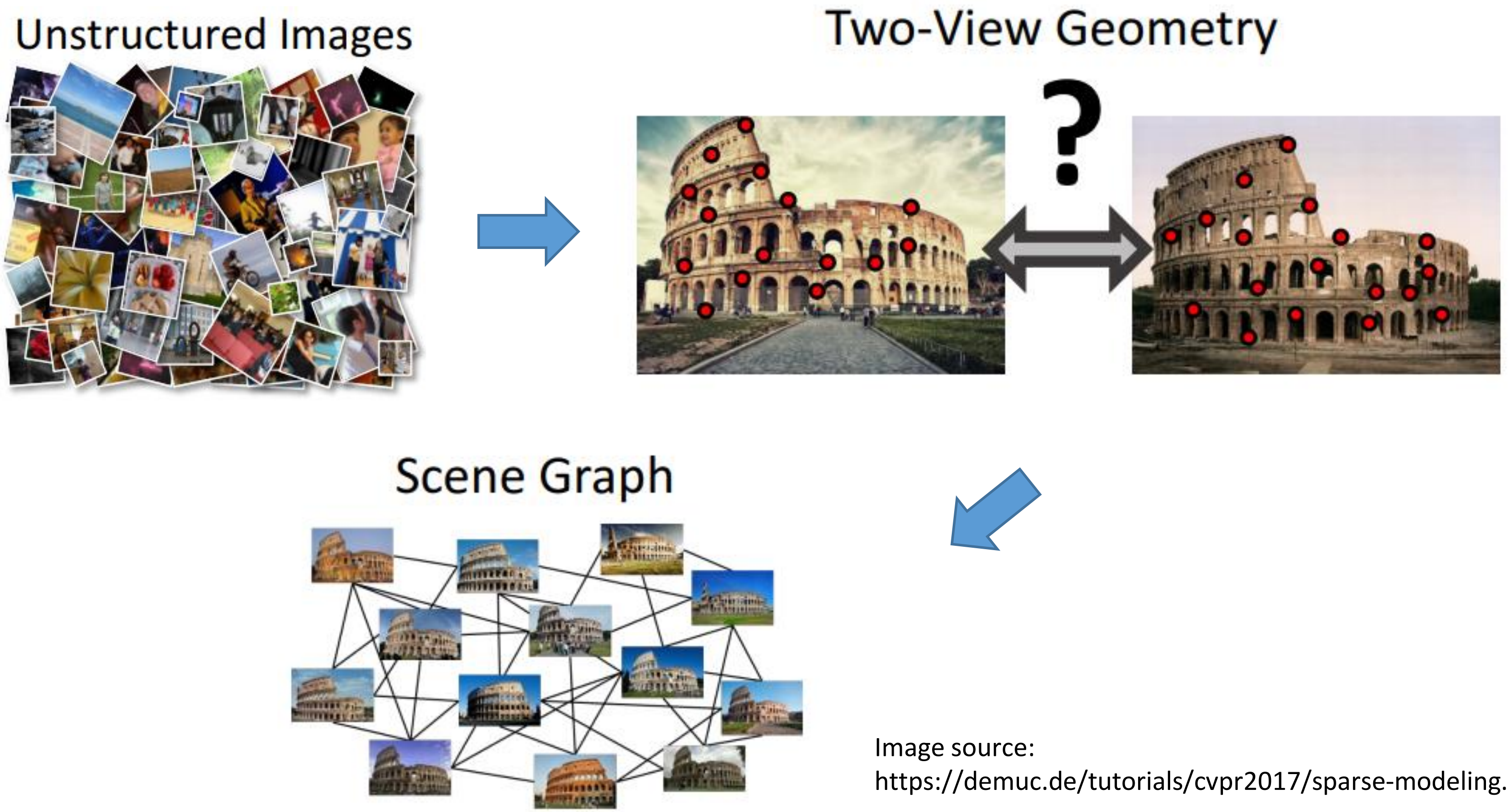


Image source:
https://demuc.de/tutorials/cvpr2017/sparse-modeling.pdf

Data Association

Step 1: extract image keypoints e.g. SIFT [1] or ORB [2].

Step 2: establish the putative correspondences.

Step 3: geometric verification to get correct image pairs.

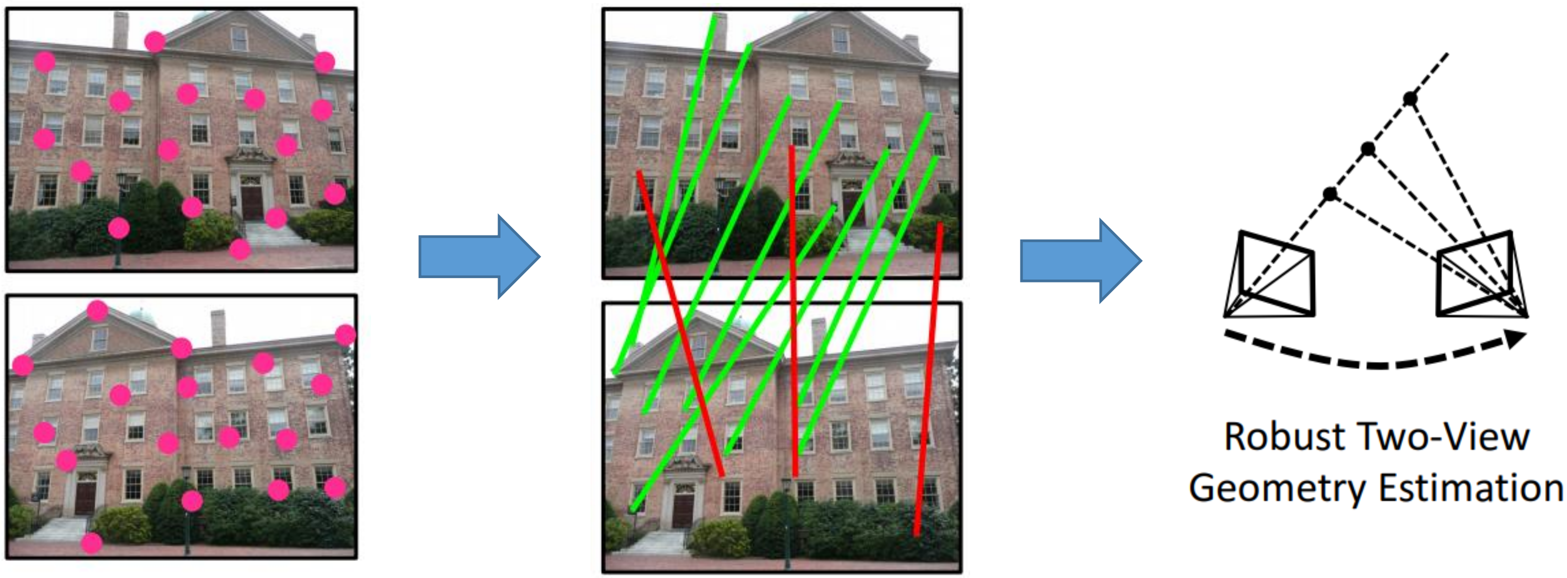


Image source: https://demuc.de/tutorials/cvpr2017/sparse-modeling.pdf

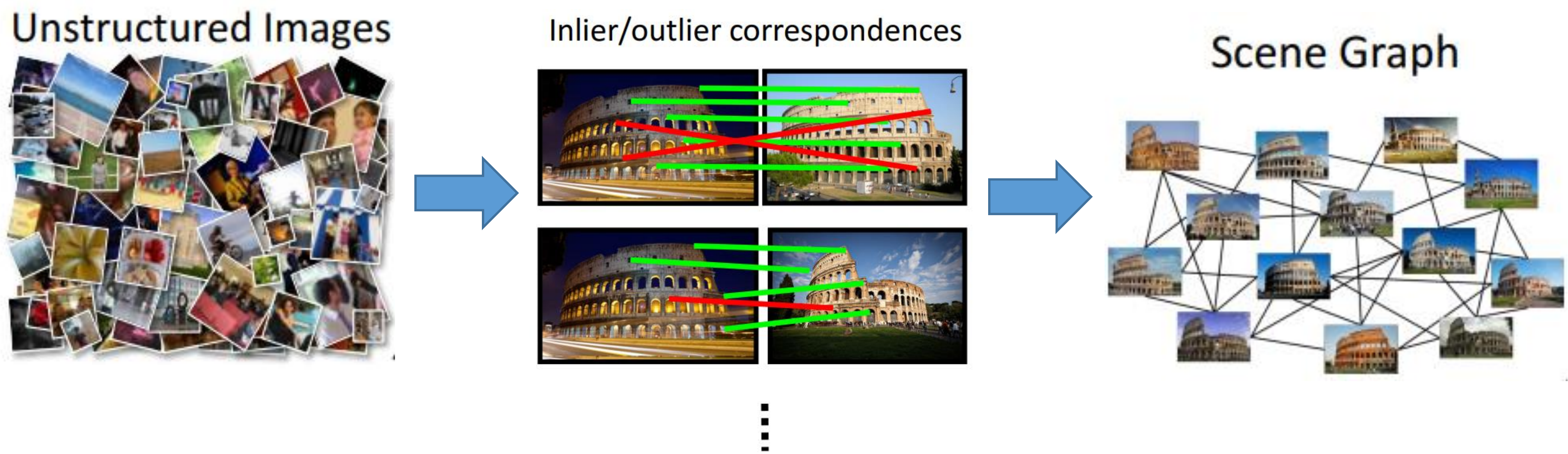
[1] David G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints“, IJCV 2004.
[2] Ethan et. al., “ORB: An efficient alternative to SIFT or SURF”, ICCV 2011.

Data Association

Geometric Verification:

- The image keypoint matchings is only based on appearance and might be wrong due to outliers.
- We do a RANSAC-based two-view geometry, e.g. homography, fundamental or essential matrix to detect outlier matches with geometry.
- Image pairs with inlier counts > threshold are added to the scene graph.

Data Association



Problem: Exhaustively searching through pairs of images in the set of N images is intractable when N is large!

Complexity of querying one image is $\mathcal{O}(N \, . \, K^2)$, where K is the number of keypoints in each image.

Image source: https://demuc.de/tutorials/cvpr2017/sparse-modeling.pdf

Data Association

Example:

- Assume 1,000 SIFT features per image $\Rightarrow K = 1{,}000$.
- Assume $N = 100{,}000{,}000$ images.
- $\Rightarrow N.K^2 = 100{,}000{,}000{,}000{,}000$ feature comparisons!
- If we assume 0.1 ms per feature comparison ⇒ 1 image query would take 317 years!

Slide credit: http://rpg.ifi.uzh.ch/docs/teaching/2019/12b_recognition.pdf

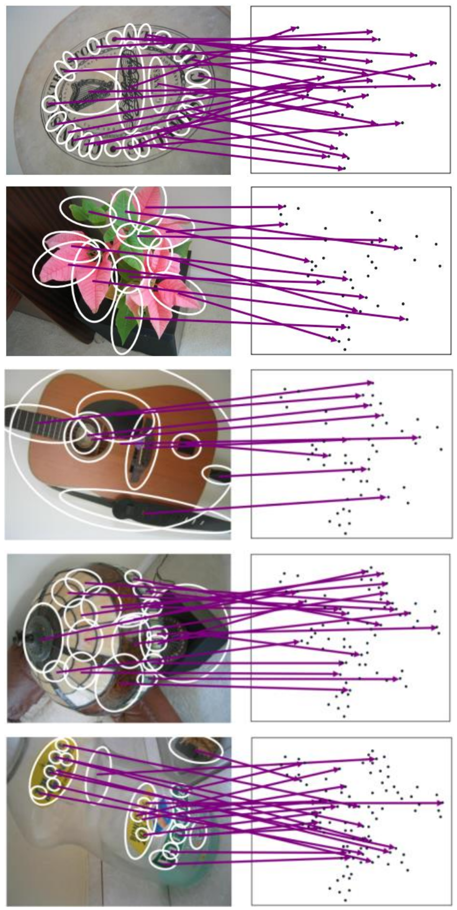
Place Recognition / Image Retrieval

Solution: Use Bag-of-Words image retrieval!

- The goal is to build an efficient tree-based search algorithm for matching the query image features with the image features from the database.

Place Recognition / Image Retrieval

Step 1: Extract image keypoints and descriptors from the training data.



For visualization we assume that the descriptor space has 2 dimensions. In the case of SIFT, the descriptor space would have 128 dimensional!

Image source: http://rpg.ifi.uzh.ch/docs/teaching/2019/12b_recognition.pdf

Place Recognition / Image Retrieval

Step 2: Perform hierarchical clustering on the descriptors. Each leave node is a "visual word" and the set forms a "bag-of-words".

Root node, 1st level node, Leaf node

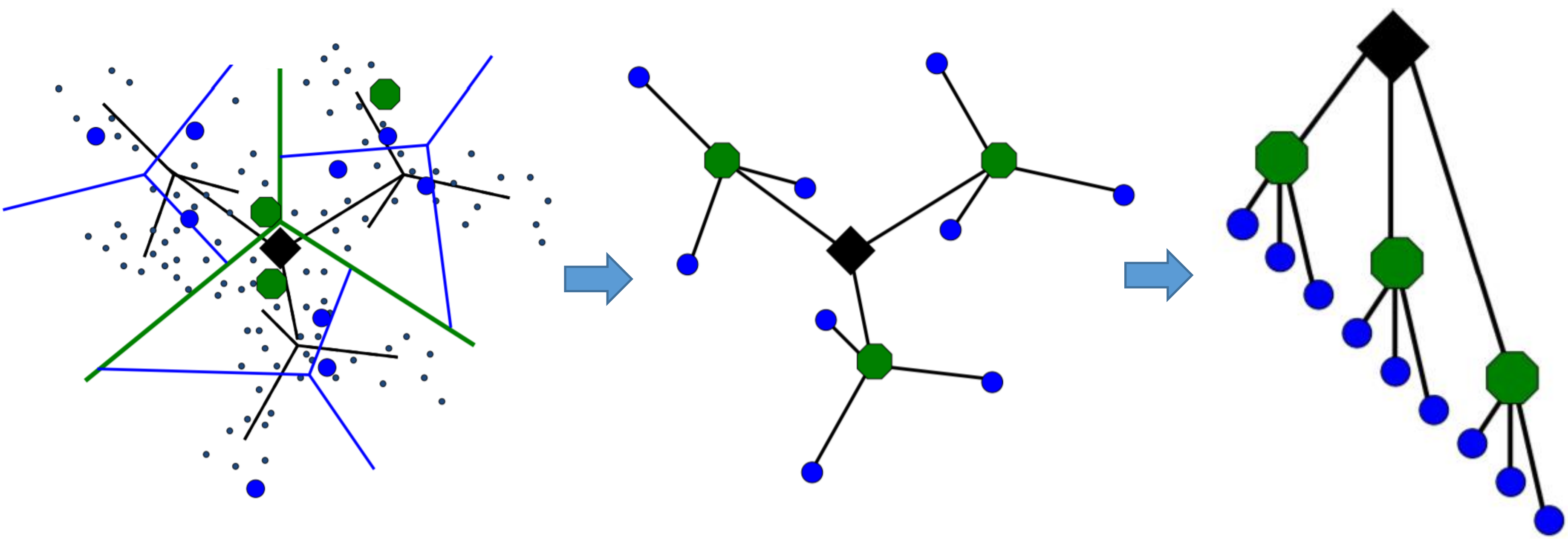


Image source: Nister et al, CVPR 2006

Place Recognition / Image Retrieval

Step 3: Build the inverted file index.

i. Extract the image features for each image in the database.

ii. Do a tree search on each image feature for the closest leaf nodes.

iii. Store the frequency count of the image features and the image ID on each leaf node as a global descriptor.

Place Recognition / Image Retrieval

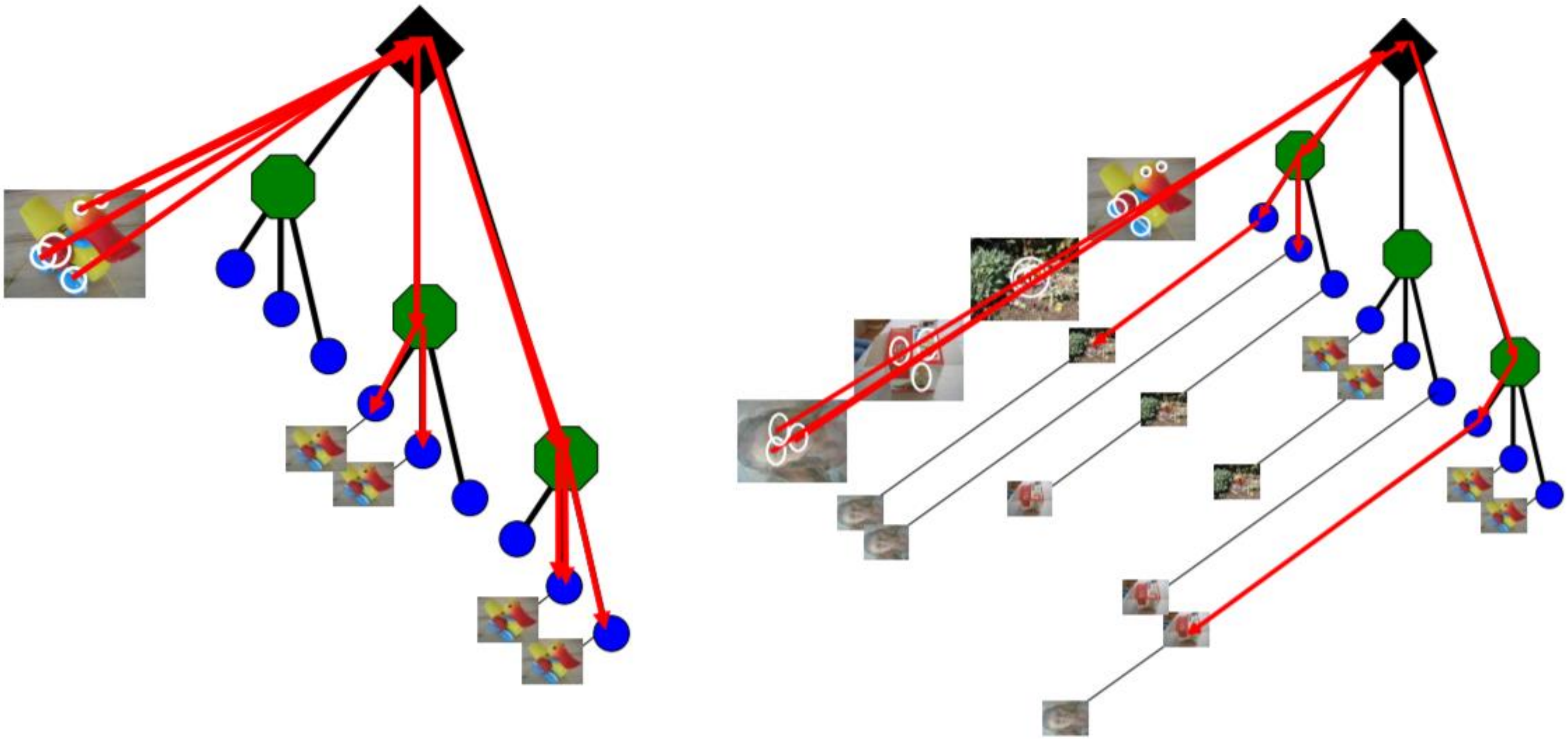


Image source: Nister et al, CVPR 2006

Place Recognition / Image Retrieval

Step 4: Given a query image, retrieve the visually most similar image from the database.

i. Extract image features from the query image.

ii. For each image feature, find the closest leaf node in the hierarchical k-mean tree.

iii. Increase the counter to the database image ID(s) with a feature in that leaf node.

iv. Choose the database image with the highest count (normalized by the frequency count) as the visually most similar image.

Place Recognition / Image Retrieval

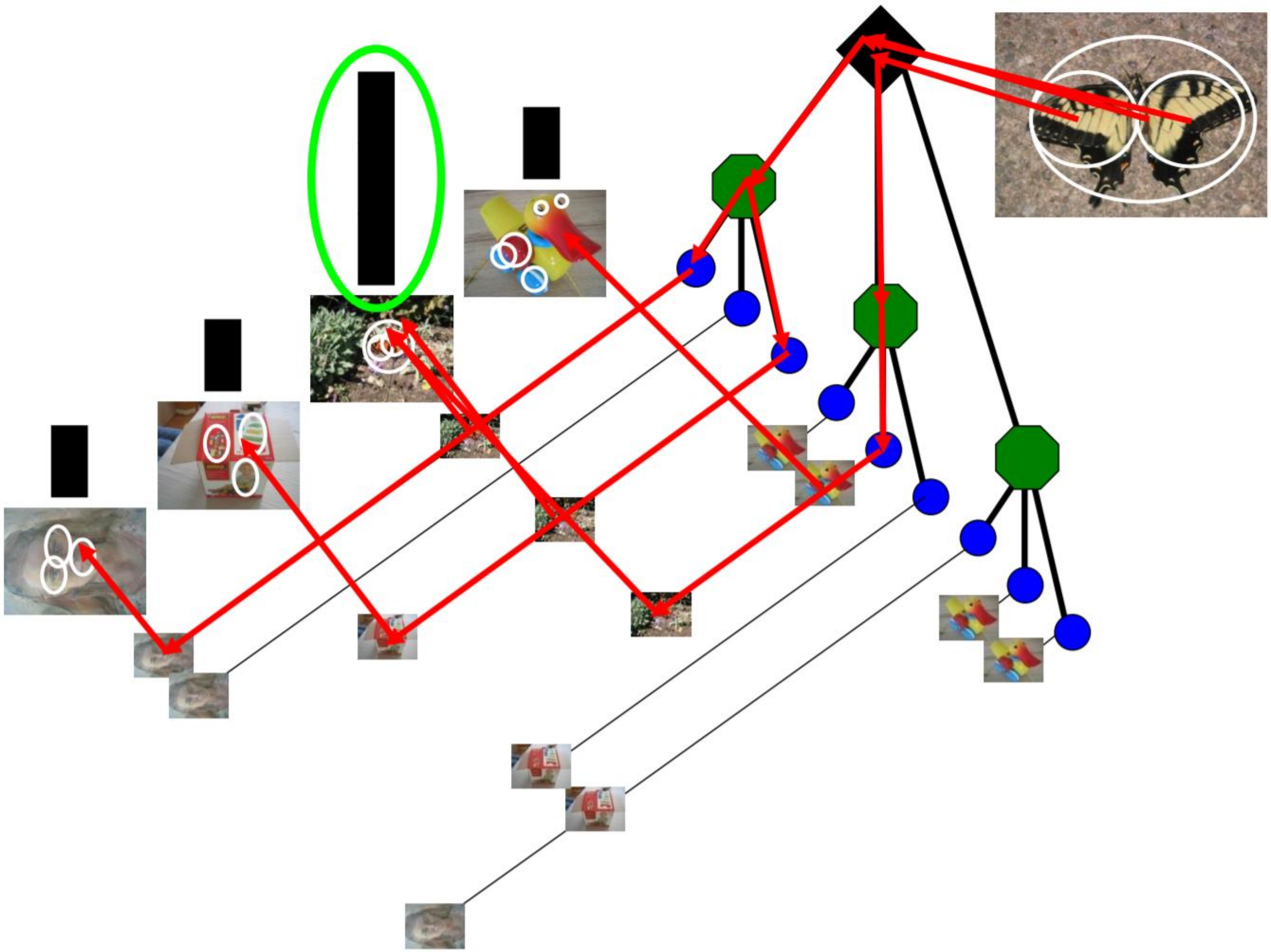


Image source: Nister et al, CVPR 2006

Place Recognition / Image Retrieval

Example: Querying an image in a database of 100 million images.

- Assume a query image with $1{,}000$ SIFT features $\Rightarrow K = 1{,}000$.
- Assume 10 branches and 6 depth levels (i.e., $b^L = 1{,}000{,}000$ visual words)
- $\Rightarrow$ Number of feature comparisons $= K \cdot b \cdot L = 1{,}000 \cdot 10 \cdot 6 = 60{,}000$
- If we assume 0.1 ms per feature comparison → 1 image query would take 6 seconds!

Place Recognition / Image Retrieval

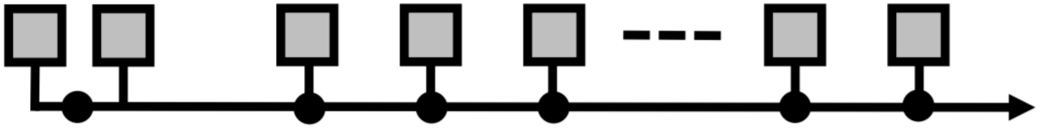
- **Reduced complexity:** For K features in the Query image, only $\mathcal{O}(K \cdot b \cdot L)$ comparisons need to be made instead of $\mathcal{O}(N \cdot K^2)$.

- We first apply image retrieval to eliminate the infeasible edges in the scene graph.

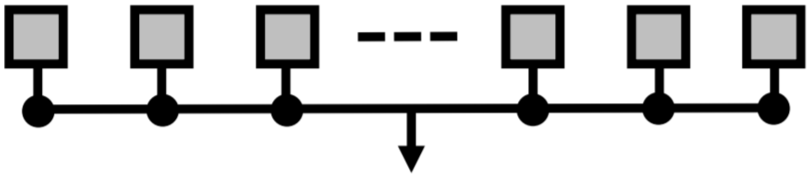
- Since image retrieval is appearance-based, we then apply geometric verification on the remaining edges in the scene graph.

Three Paradigms of SfM

1. Incremental:



2. Global:



3. Hierarchical:

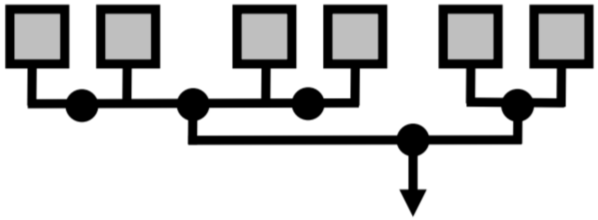


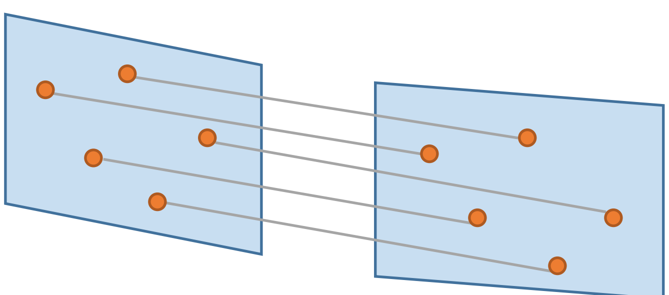
Image source: https://demuc.de/tutorials/cvpr2017/sparse-modeling.pdf

Incremental SfM

Step 1: Initialization

i. Choose two non-panoramic views ($||\mathbf{t}|| \neq 0$). A good choice is the image pair with the highest inlier count.

ii. Use 8-point algorithm (F or E matrix) for non-planar scene or 4-point algorithm (homography) for planar scene.



$\mathrm{P} = \mathrm{K}[\mathrm{I} \mid 0]$

$\mathrm{P}' = \mathrm{K}'[\mathrm{R}' \mid \mathbf{t}']$

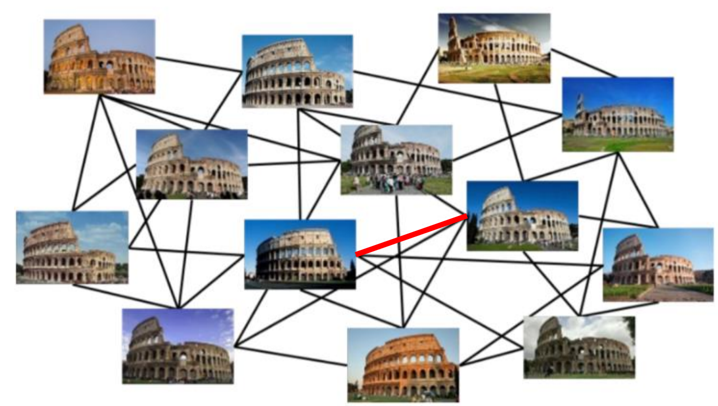


Image source: https://demuc.de/tutorials/cvpr2017/sparse-modeling.pdf

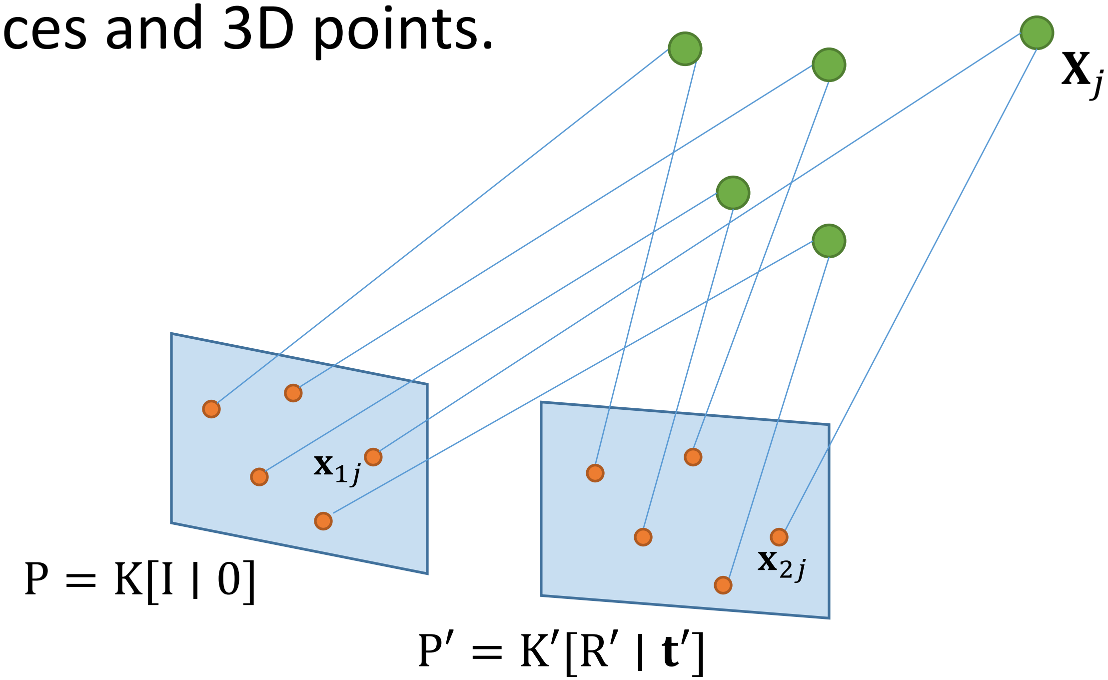
Incremental SfM

Step 1: Initialization

iii. Set the scale to 1, i.e. $||\mathbf{t}|| = 1$.

iv. Triangulate inlier correspondences to get the 3D points.

v. Apply bundle adjustment (detail later) to refine camera matrices and 3D points.

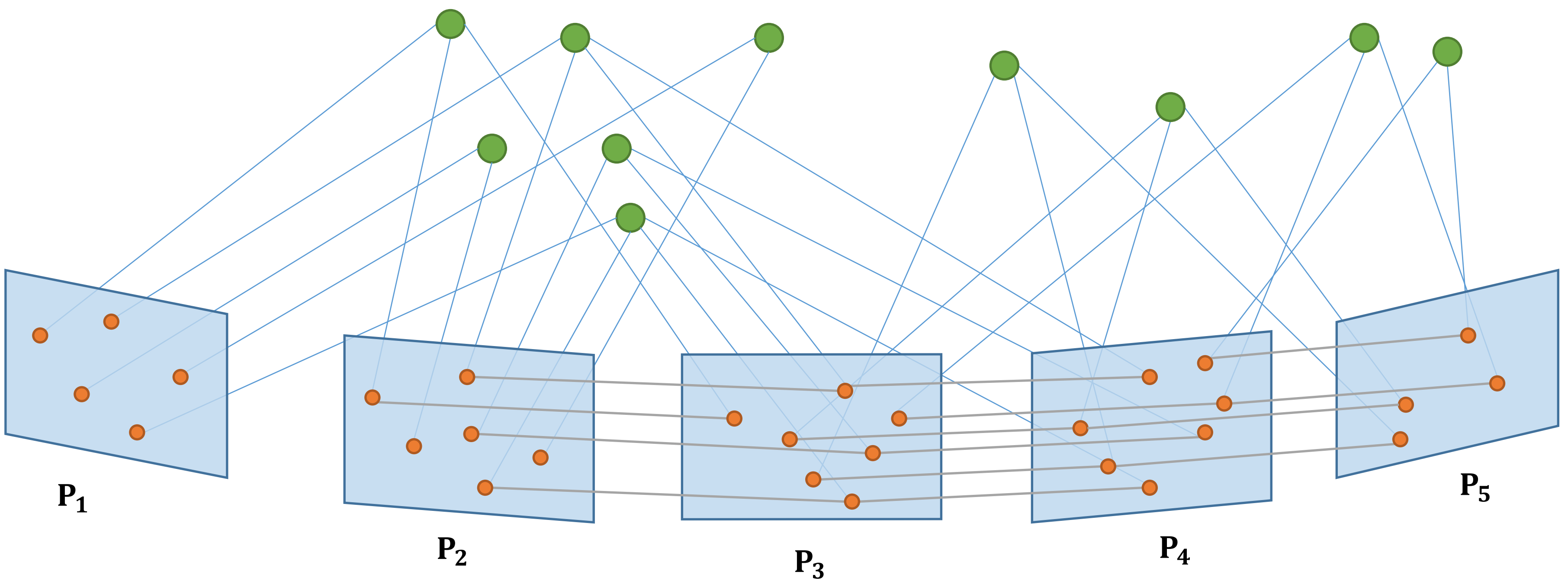


Incremental SfM

Step 2: Subsequent Views

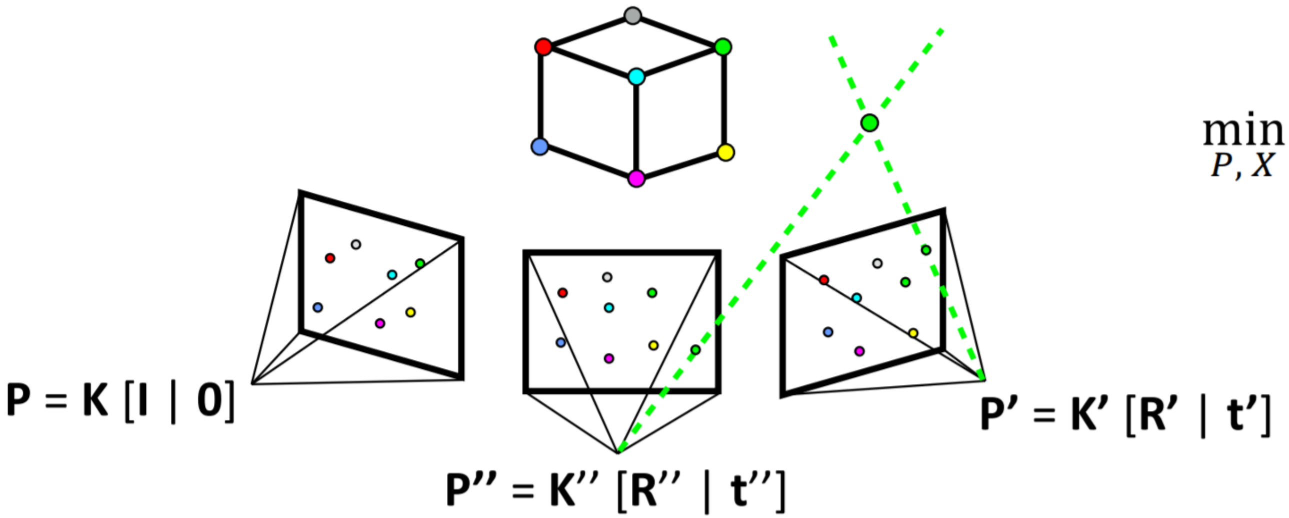
i. Find 2D-3D correspondences.

ii. Solve the Perspective-n-Point (PnP) problem.



Incremental SfM

Step 3: Refine the multi-view camera poses and structures with bundle adjustment (detail later).



$$\min_{P,X} \|x - \pi(P, X)\|$$

Image source: https://demuc.de/tutorials/cvpr2017/sparse-modeling.pdf

Global SfM

1. Compute all relative poses $(\mathrm{R}, \mathbf{t})$ in the edges of the scene graph.

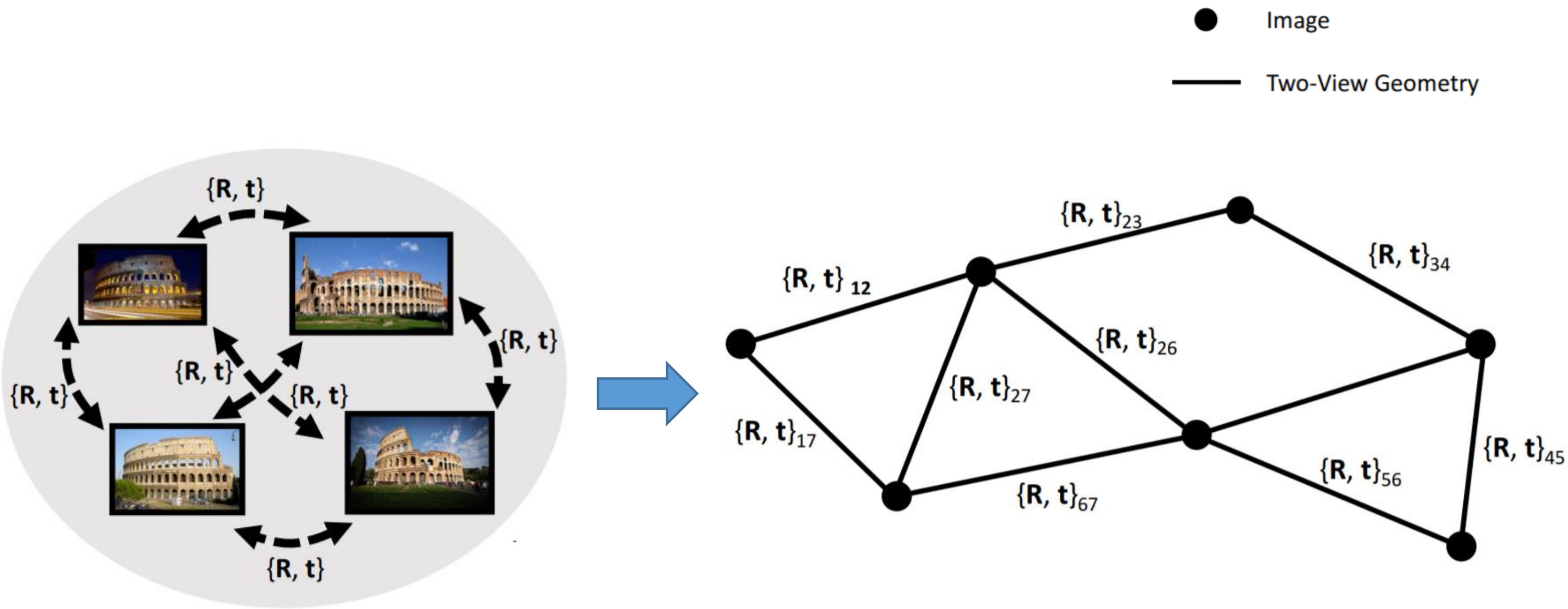


Image source: https://demuc.de/tutorials/cvpr2017/sparse-modeling.pdf

Reference: Chatterje and Govindu, "Efficient and Robust Large-Scale Rotation Averaging", 2013

Global SfM

2. Estimate the global rotations: $\min_{\mathrm{R}} \left\| \mathrm{R}_{ij} - \mathrm{R}_i \mathrm{R}_j^{\mathsf{T}} \right\|$,

   where R_{ij} are the relative rotations and R_i are the global rotations.

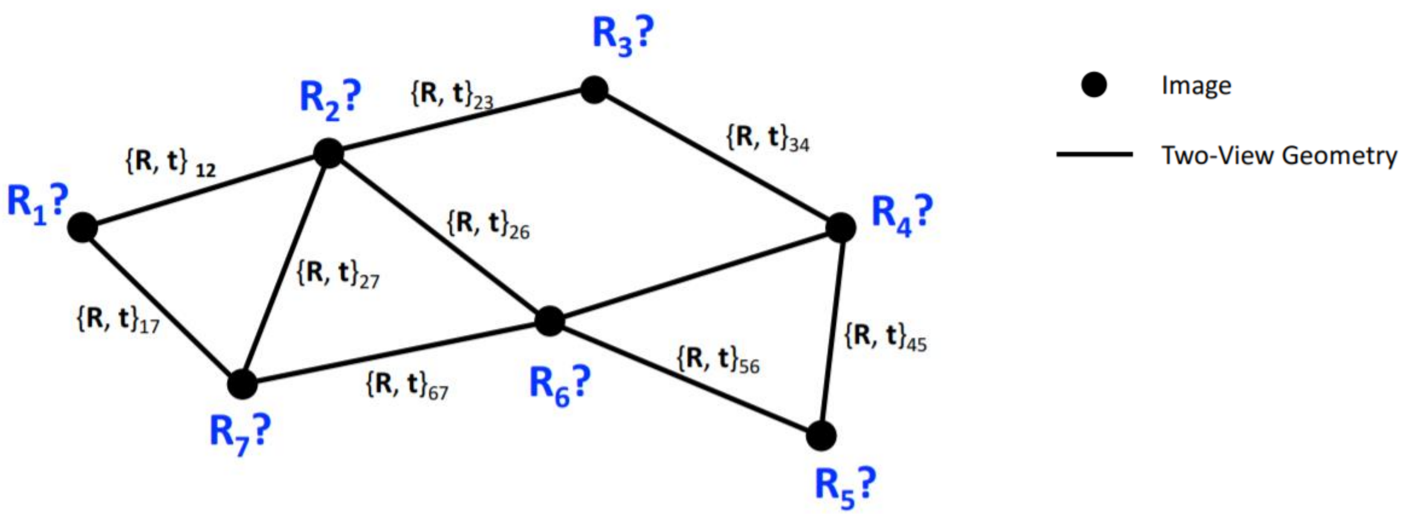


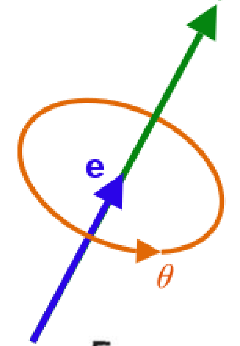
Image source: https://demuc.de/tutorials/cvpr2017/sparse-modeling.pdf

Reference: Chatterje and Govindu, "Efficient and Robust Large-Scale Rotation Averaging", 2013

Global SfM

- We define the logarithmic map as: $\boldsymbol{\theta} = \widehat{\omega} = \log(\mathrm{R}) \in so(3)$.

Angle-axis representation of rotation: $\boldsymbol{\theta} = \theta\mathbf{e}$



- Specifically, for $R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$,

$$\theta = \|\omega\| = \cos^{-1}\left(\frac{\mathrm{trace}(R) - 1}{2}\right), \qquad \mathbf{e} = \frac{\widehat{\omega}}{\|\omega\|} = \frac{1}{2\sin(\|\omega\|)} \begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix}.$$

- If $\mathrm{R} = \mathrm{I}$, then $\|\omega\| = 0$, and $\frac{\widehat{\omega}}{\|\omega\|}$ is not determined and therefore can set to $\frac{\widehat{\omega}}{\|\omega\|} = [0, 0, 0]^{\mathsf{T}}$.

Image source: https://en.wikipedia.org/wiki/Axis%E2%80%93angle_representation

Global SfM

- The inverse is given by the exponential map, i.e. $\mathrm{R} = \exp(\widehat{\omega}) \in SO(3)$.

- This is also known as the Rodrigues' formula for a rotation matrix:

$$\mathrm{R} = e^{\widehat{\omega}} = I + \mathrm{K}\sin(\|\omega\|) + \mathrm{K}^2(1-\cos(\|\omega\|)) ,$$

where

$$\mathrm{K} = [\mathbf{e}]_\times,$$

$$\mathrm{K}^2 = \mathbf{e}\mathbf{e}^\top - \mathrm{I}.$$

Global SfM

- $R_{ij} = R_i R_j^T$ can be written as:

$$\hat{\omega}_{ij} = \hat{\omega}_j - \hat{\omega}_i = \underbrace{\left[\begin{array}{c} \cdots -\mathbf{I} \cdots \mathbf{I} \cdots \end{array} \right]}_{\mathbf{A}_{ij}} \hat{\omega}_{global} \,.$$

- Then, $\min_{R} \| R_{ij} - R_i R_j^T \|$ can be solved as a least squares problem: $\mathbf{A}\hat{\omega}_{global} = \hat{\omega}_{rel}\,.$

- $\hat{\boldsymbol{\omega}}_{rel}$ is the vector made by stacking all relative rotation observations $\hat{\boldsymbol{\omega}}_{ij}$ and A is made by stacking the corresponding matrices A_{ij}

Global SfM

2. Estimate global translations:

$$\min_{\mathbf{t}} \left\| \mathbf{t}_{ij} - \frac{\mathbf{t}_i - \mathbf{t}_j}{\|\mathbf{t}_i - \mathbf{t}_j\|} \right\| \quad \xrightarrow{\text{solve as}} \quad \mathrm{A}\mathbf{x} = \mathbf{b}$$

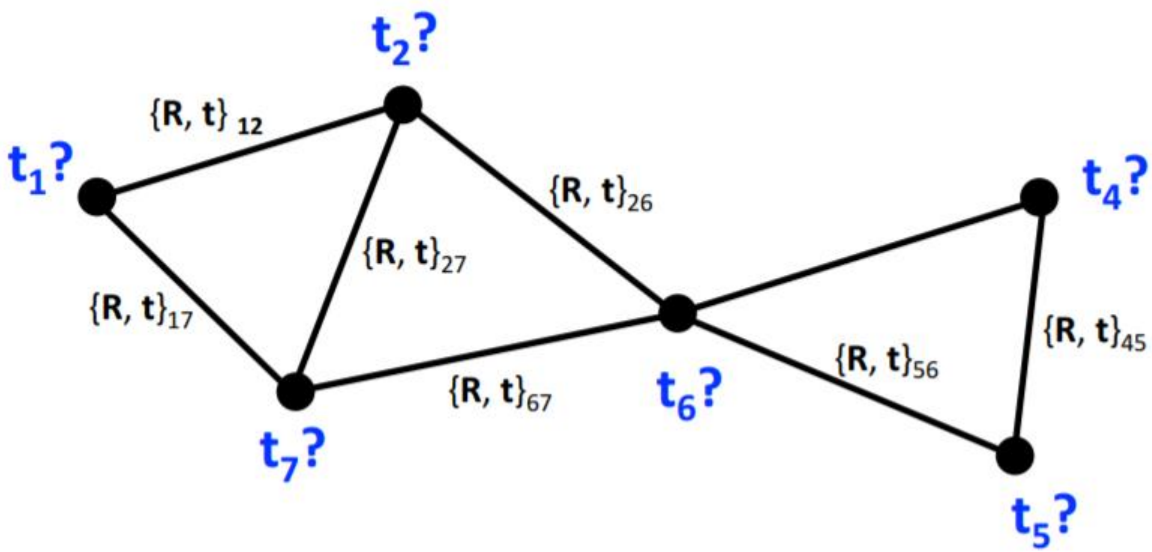
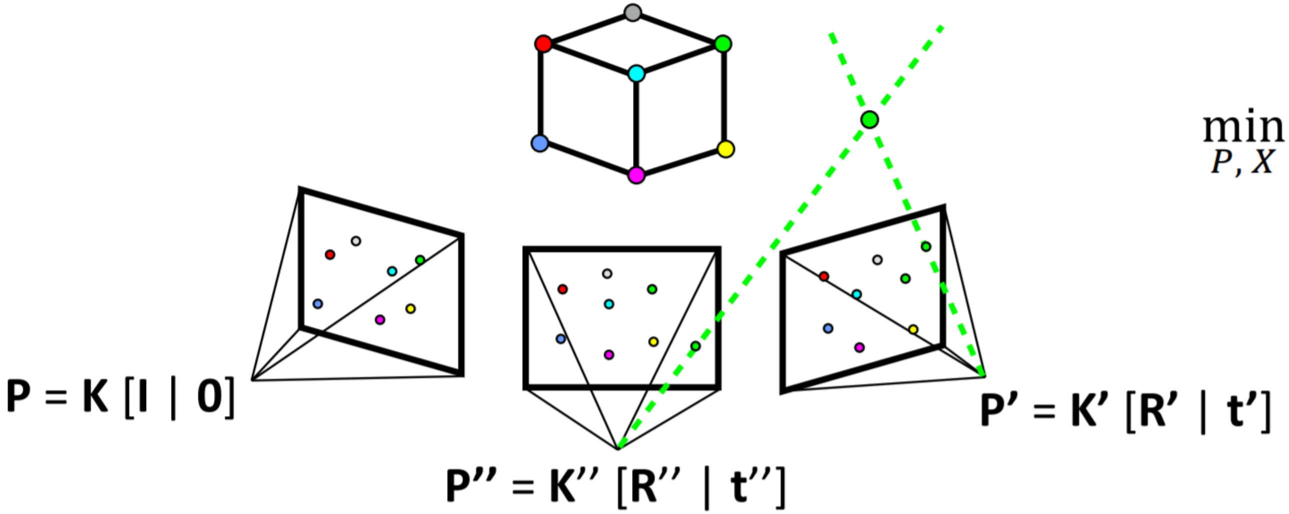


Image source: https://demuc.de/tutorials/cvpr2017/sparse-modeling.pdf

Reference: Chatterje and Govindu, “Efficient and Robust Large-Scale Rotation Averaging”, 2013

Global SfM

3. Triangulate and refine with bundle adjustment (detail later).



$$\min_{P,X} \|x - \pi(P, X)\|$$

Image source: https://demuc.de/tutorials/cvpr2017/sparse-modeling.pdf

Hierarchical SfM

1. Hierarchical clustering of scene graph
2. Reconstruct clusters independently (using incremental or global SfM)
3. Merge clusters using similarity transformations

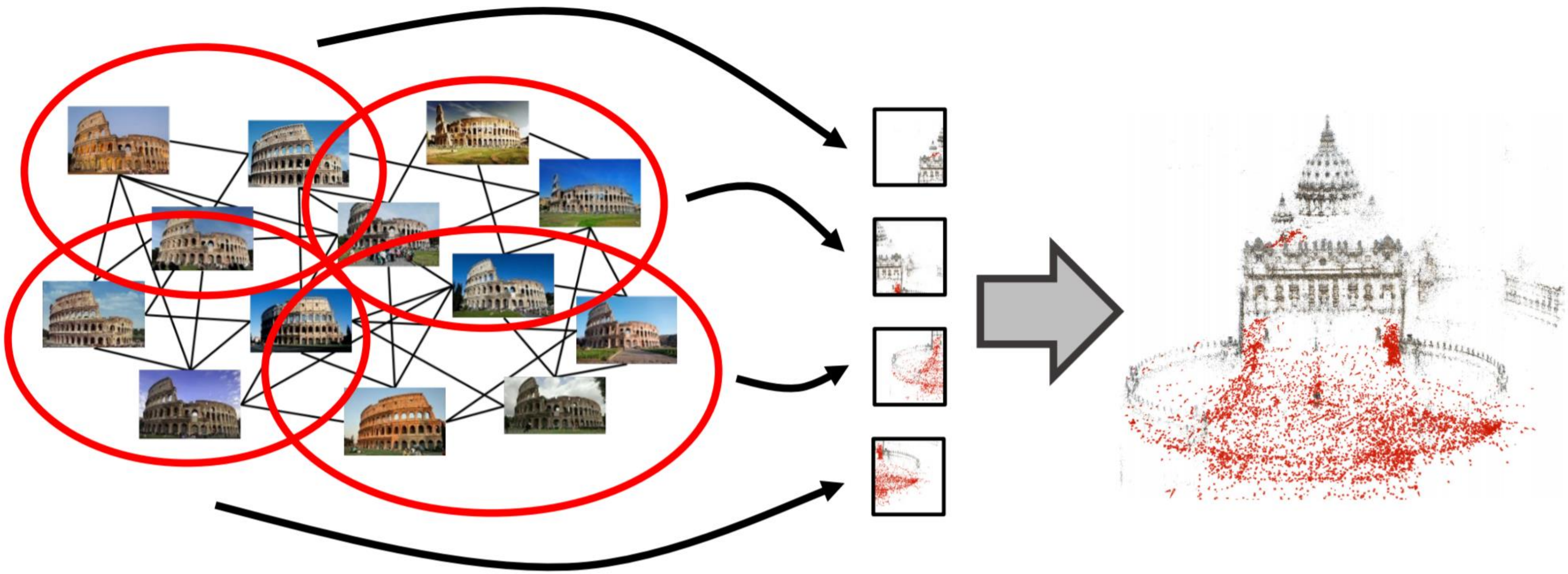
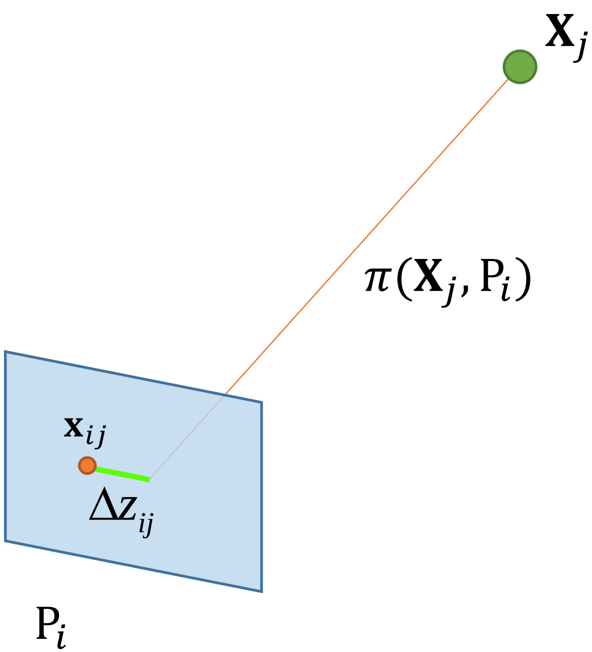


Image source: https://demuc.de/tutorials/cvpr2017/sparse-modeling.pdf

Bundle Adjustment

- Refine a visual reconstruction to produce jointly optimal 3D structures X_j and camera poses P_i.
- Minimize the total reprojection errors Δz_{ij}.



Cost Function:

$$\underset{\mathrm{P},\mathbf{X}}{\operatorname{argmin}} \sum_i \sum_j \| \underbrace{\mathbf{x}_{ij} - \pi(\mathrm{P}_i, \mathbf{X}_j)}_{\Delta z_{ij}} \|^2_{W_{ij}}$$

W_{ij}: Measurement error covariance

$\pi(\mathbf{X}_j, \mathrm{P}_i) = \frac{\mathrm{P}_i \mathbf{X}_j}{\hat{x}_3}$, where $\hat{x}_3$ is the 3rd element of $\mathrm{P}_i \mathbf{X}_j$

Bundle Adjustment

- The cost function can be rewritten as:

$$\underset{\boldsymbol{\mathcal{P}}}{\operatorname{argmin}} \underbrace{\sum_i \sum_j \Delta z_{ij}^\top W_{ij} \Delta z_{ij}}_{g(\boldsymbol{\mathcal{P}})}$$

  - W_{ij}: Measurement error covariance
  - $\boldsymbol{\mathcal{P}}$ is a $(12M \times 3N)$ vector with $12M$ camera matrix parameters and $3N$ point coordinates

Iterative Estimation Methods

- Suppose we are given a hypothesized **nonlinear** functional relation $\mathbf{X} = f(\mathbf{P})$.
- $\mathbf{X} \in \mathbb{R}^N$ is a measurement vector and $\mathbf{P} \in \mathbb{R}^M$ is a parameter vector in Euclidean spaces.
- A measured value of $\mathbf{X}$ approximating the true value $\bar{\mathbf{X}}$ is provided, and we wish to find the vector $\hat{\mathbf{P}}$ that most nearly satisfies this functional relation.
- More precisely, we seek the vector $\hat{\mathbf{P}}$ satisfying $\mathbf{X} = f(\hat{\mathbf{P}}) - \epsilon$ for which $g(\hat{\mathbf{P}}) = \frac{1}{2}\|\epsilon\|^2 = \frac{1}{2}\epsilon^\top\epsilon$ is minimized.

Iterative Estimation Methods

- We will look at four iterative estimation methods to minimize $g(\mathbf{P})$.

1. Newton's Method
2. Gauss-Newton
3. Gradient Descent
4. Levenberg-Marquardt

Newton's Method and the Hessian

- Let $\mathbf{P}_0$ be an initial estimated value, i.e. $g(\mathbf{P}_0)$.
- We may expand $g(\mathbf{P})$ about $\mathbf{P}_0$ in a Taylor series to get:

$$g(\mathbf{P}_0 + \mathbf{\Delta}) = g + g_{\mathbf{P}}\mathbf{\Delta} + \mathbf{\Delta}^{\mathsf{T}} g_{\mathbf{PP}}\mathbf{\Delta}/2 + \ldots$$

- where subscript $\mathbf{P}$ denotes differentiation, and the right-hand side is evaluated at $\mathbf{P}_0$, i.e.

$$g_{\mathbf{P}} = \left.\frac{\partial g}{\partial \mathbf{P}}\right|_{\mathbf{p}=\mathbf{P_0}} \quad \text{and} \quad g_{\mathbf{PP}} = \left.\frac{\partial^2 g}{\partial \mathbf{P}^2}\right|_{\mathbf{p}=\mathbf{P_0}}.$$

Jacobian and Hessian Matrices

- Suppose $\mathbf{f}(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is a function such that each of its 1st and 2nd order partial derivatives exist on $\mathbb{R}^n$, then the Jacobian matrix is given by an $m \times n$ matrix:

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

or simply

$$\mathbf{J}_{ij} = \frac{\partial f_i}{\partial x_j},$$

- And the Hessian matrix is an $n \times n$ matrix:

$$\mathbf{H} = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1\,\partial x_n} \\[2.2ex] \dfrac{\partial^2 f}{\partial x_2\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_2^2} & \cdots & \dfrac{\partial^2 f}{\partial x_2\,\partial x_n} \\[2.2ex] \vdots & \vdots & \ddots & \vdots \\[2.2ex] \dfrac{\partial^2 f}{\partial x_n\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

or simply

$$\mathbf{H}_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}.$$

https://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant
https://en.wikipedia.org/wiki/Hessian_matrix

Newton's Method and the Hessian

- We seek a point $g(\mathbf{P}_1)$, with $\mathbf{P}_1 = \mathbf{P}_0 + \boldsymbol{\Delta}$, which minimizes:

$$g(\mathbf{P}_1) = g(\mathbf{P}_0) + g_{\mathbf{p}}\boldsymbol{\Delta} + \boldsymbol{\Delta}^{\mathrm{T}} g_{\mathbf{PP}}\boldsymbol{\Delta}/2$$
$$= g_0 + g_{\mathbf{p}}\boldsymbol{\Delta} + \boldsymbol{\Delta}^{\mathrm{T}} g_{\mathbf{PP}}\boldsymbol{\Delta}/2$$

with respect to $\boldsymbol{\Delta}$.

- To this end, we differentiate with respect to $\boldsymbol{\Delta}$ and set the derivative to zero to get:

$$g_{\mathbf{P}} + g_{\mathbf{PP}}\boldsymbol{\Delta} = 0 \quad \text{or} \quad g_{\mathbf{PP}}\boldsymbol{\Delta} = -g_{\mathbf{P}}.$$

Newton's Method and the Hessian

- The solution vector $\widehat{\mathbf{P}}$ is obtained by starting with an estimate $\mathbf{P}_0$ and computing successive approximations according to the formula:

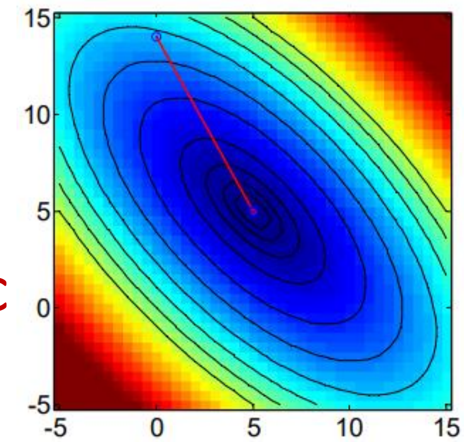
$$\mathbf{P}_{i+1} = \mathbf{P}_i + \boldsymbol{\Delta}_i$$

- where $\boldsymbol{\Delta}_i$ is the solution to the linear least-squares problem

$$g_{\mathbf{PP}}\boldsymbol{\Delta} = -g_{\mathbf{P}}.$$

Newton's Method and the Hessian

Remarks:



- Newton iteration is based on the assumption of an approximately quadratic cost function near the minimum.
- And will show rapid convergence if this condition is met.
- The disadvantage of this approach is that the computation of the Hessian may be difficult.
- In addition, the assumption of quadratic behaviour is probably invalid when far from the minimum.

Image source: http://www.robots.ox.ac.uk/~az/lectures/opt/lect1.pdf

Gauss-Newton Method

- Since $g(\mathbf{P}) = \frac{1}{2}\|\boldsymbol{\epsilon}(\mathbf{P})\|^2 = \boldsymbol{\epsilon}(\mathbf{P})^\mathsf{T}\boldsymbol{\epsilon}(\mathbf{P})/2$, we can write:

$$g_{\mathbf{P}} = \boldsymbol{\epsilon}_{\mathbf{P}}^\mathsf{T}\boldsymbol{\epsilon}, \quad \text{where} \quad \boldsymbol{\epsilon}_{\mathbf{P}} = \mathbf{f}_{\mathbf{P}} = \mathsf{J}, \text{ and}$$

$$g_{\mathbf{PP}} = \boldsymbol{\epsilon}_{\mathbf{P}}^\mathsf{T}\boldsymbol{\epsilon}_{\mathbf{P}} + \boldsymbol{\epsilon}_{\mathbf{PP}}^\mathsf{T}\boldsymbol{\epsilon}.$$

- In Gauss-Newton method, we ignore the $\epsilon_{\mathbf{PP}}^\mathsf{T}\epsilon$ term to get:

$$g_{\mathbf{PP}} = \boldsymbol{\epsilon}_{\mathbf{P}}^\mathsf{T}\boldsymbol{\epsilon}_{\mathbf{P}} = \mathsf{J}^\mathsf{T}\mathsf{J}.$$

Gauss-Newton Method

- The update equation $g_{\mathbf{PP}}\boldsymbol{\Delta} = -g_{\mathbf{P}}$ becomes:

$$\mathtt{J}^{\mathsf{T}}\mathtt{J}\boldsymbol{\Delta} = -\mathtt{J}^{\mathsf{T}}\boldsymbol{\epsilon}\,.$$

- This is also known as the normal equation.

- **Weighted iteration:** assume that the measurement $\mathbf{X}$ satisfies a Gaussian distribution with covariance matrix $\Sigma_{\mathbf{X}}$, the normal equation becomes

$$\mathtt{J}^{\mathsf{T}}\Sigma^{-1}\mathtt{J}\boldsymbol{\Delta}_i = -\mathtt{J}^{\mathsf{T}}\Sigma^{-1}\boldsymbol{\epsilon}_i.$$

- $\Sigma_{\mathbf{X}}$ is a symmetric and positive definite matrix.

Gradient Descent

- The negative (or down-hill) gradient vector $-g_{\mathbf{P}} = -\epsilon_{\mathbf{P}}^{\mathrm{T}}\epsilon$ defines the direction of most rapid decrease of the cost function.

- A strategy for minimization of g is to move iteratively in the gradient direction; this is known as gradient descent.

- The parameter increment $\mathbf{\Delta}$ is computed from the equation $\lambda\mathbf{\Delta} = -g_{\mathbf{P}}$, where λ controls the length of the step.

Gradient Descent

Remarks:

- We may consider this as related to Newton iteration where the Hessian is approximated (somewhat arbitrarily) by the scalar matrix $\lambda\mathrm{I}$.

- Gradient descent by itself is not a very good minimization strategy, typically characterized by slow convergence due to zig-zagging.

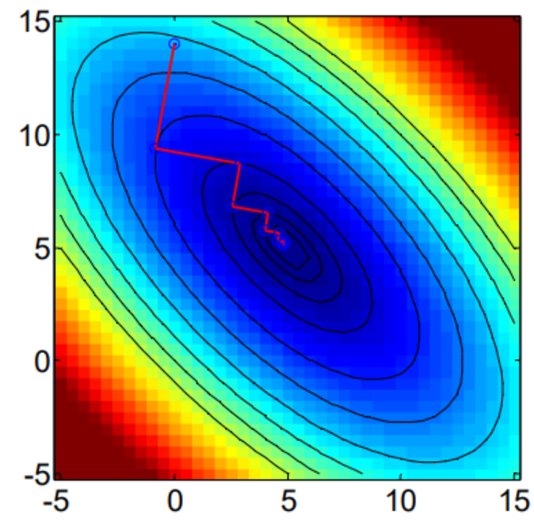


Image source: http://www.robots.ox.ac.uk/~az/lectures/opt/lect1.pdf

Levenberg–Marquardt Method

- The Levenberg–Marquardt (abbreviated LM) iteration method is a slight variation on the Gauss-Newton iteration method.

- The normal equations $\mathtt{J}^\mathsf{T}\mathtt{J}\boldsymbol{\Delta} = -\mathtt{J}^\mathsf{T}\boldsymbol{\epsilon}$ are replaced by the augmented normal equations

$$(\mathtt{J}^\mathsf{T}\mathtt{J} + \lambda\mathtt{I})\boldsymbol{\Delta} = -\mathtt{J}^\mathsf{T}\boldsymbol{\epsilon},$$

- For some value of λ that varies from iteration to iteration, and I is the identity matrix.

Levenberg–Marquardt Method

- A typical initial value of λ is 10^{-3} times the average of the diagonal elements of $\mathrm{N} = \mathrm{J}^{\top}\mathrm{J}$.

- If the value of $\mathbf{\Delta}$ obtained by solving the augmented normal equations leads to a:

i. Reduction in the error, then the increment is accepted and λ is divided by a factor (typically 10) before the next iteration.

ii. Increased error, then the increment is rejected; λ is multiplied by the same factor and the augmented normal equations are solved again.

Justification of LM

- When λ is very small, the method is essentially the same as Gauss-Newton iteration, i.e.

$$(\mathrm{J}^{\mathsf{T}}\mathrm{J} + \cancel{\lambda}\mathrm{I})\boldsymbol{\Delta} = -\mathrm{J}^{\mathsf{T}}\boldsymbol{\epsilon}$$

$$\Downarrow$$

$$\mathrm{J}^{\mathsf{T}}\mathrm{J}\boldsymbol{\Delta} = -\mathrm{J}^{\mathsf{T}}\boldsymbol{\epsilon}$$

- **Remarks:** The error function $||\epsilon||^2 = f(\mathbf{P}) - \mathbf{X}^2$ is quadratic in $\mathbf{P}$ when near the minimum, hence will converge fast to the minimum value.

Justification of LM

- When λ is large, the normal equation matrix is approximated by $\lambda \mathrm{I}$, and the normal equations become $\lambda \boldsymbol{\Delta} = -\mathrm{J}^{\top}\epsilon$.

- Recalling that $\mathrm{J}^{\top}\epsilon$ is simply the gradient vector of $||\epsilon||^2$, we see that the direction of the parameter increment $\boldsymbol{\Delta}$ approaches that given by gradient descent.

- **Remarks:** Gauss-Newton which does not work well due to the violation of the quadratic assumption when far from the minimum is replaced by Gradient descent.

Justification of LM

- The LM algorithm moves seamlessly between:

1. Gauss-Newton iteration, which will cause rapid convergence in the neighbourhood of the solution, and
2. A gradient descent approach, which will guarantee a decrease in the cost function when the going is difficult.

- Indeed, as λ becomes increasingly larger, the length of the increment step $\boldsymbol{\Delta}$ decreases and eventually it will lead to a decrease of the cost function $||\epsilon||^2$.

A Sparse Levenberg–Marquardt Algorithm

- The bare LM algorithm is not very suitable for minimizing cost functions w.r.t. large numbers of parameters, e.g. bundle adjustment.
- This is because the solution of the normal equations has complexity $\mathcal{O}(N^3)$ in the number of parameters, and this step is repeated many times.
- However, the normal equation matrix has a certain sparse block structure that we may take advantage of to realize very great time savings.

A Sparse Levenberg–Marquardt Algorithm

- More specifically, the set of parameters $\mathbf{P} \in \mathbb{R}^{\mathrm{M}}$ may be partitioned into parameter vectors $\mathbf{a}$ and $\mathbf{b}$ so that $\mathbf{P} = (\mathbf{a}^\top, \mathbf{b}^\top)^\top$.

- Now the Jacobian matrix $\mathrm{J} = [\partial \widehat{\mathbf{X}} / \partial \mathbf{P}]$ has a block structure of the form $\mathrm{J} = [\mathrm{A} \mid \mathrm{B}]$, where Jacobian submatrices are defined by

$$\mathtt{A} = \left[\partial \widehat{\mathbf{X}} / \partial \mathbf{a}\right] \quad \text{and} \quad \mathtt{B} = \left[\partial \widehat{\mathbf{X}} / \partial \mathbf{b}\right].$$

A Sparse Levenberg–Marquardt Algorithm

- The normal equation to be solved at each step of the LM algorithm now becomes:

$$\left[\begin{array}{c|c} \mathtt{A}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\mathtt{A} & \mathtt{A}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\mathtt{B} \\ \hline \mathtt{B}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\mathtt{A} & \mathtt{B}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\mathtt{B} \end{array}\right] \left(\begin{array}{c} \boldsymbol{\delta}_{\mathbf{a}} \\ \hline \boldsymbol{\delta}_{\mathbf{b}} \end{array}\right) = \left(\begin{array}{c} \mathtt{A}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\boldsymbol{\epsilon} \\ \hline \mathtt{B}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\boldsymbol{\epsilon} \end{array}\right).$$

- And may now be written in block form as:

$$\left[\begin{array}{cc} \mathtt{U}^* & \mathtt{W} \\ \mathtt{W}^{\mathsf{T}} & \mathtt{V}^* \end{array}\right] \left(\begin{array}{c} \boldsymbol{\delta}_{\mathbf{a}} \\ \boldsymbol{\delta}_{\mathbf{b}} \end{array}\right) = \left(\begin{array}{c} \boldsymbol{\epsilon}_{\mathtt{A}} \\ \boldsymbol{\epsilon}_{\mathtt{B}} \end{array}\right).$$

A Sparse Levenberg–Marquardt Algorithm

- As a first step to solving these equations, both sides are now multiplied on the left by

$$\begin{bmatrix} \mathtt{I} & -\mathtt{W}\mathtt{V}^{*-1} \\ 0 & \mathtt{I} \end{bmatrix}$$

resulting in

$$\begin{bmatrix} \mathtt{U}^* - \mathtt{W}\mathtt{V}^{*-1}\mathtt{W}^\mathsf{T} & 0 \\ \mathtt{W}^\mathsf{T} & \mathtt{V}^* \end{bmatrix} \begin{pmatrix} \boldsymbol{\delta}_\mathbf{a} \\ \boldsymbol{\delta}_\mathbf{b} \end{pmatrix} = \begin{pmatrix} \epsilon_\mathtt{A} - \mathtt{W}\mathtt{V}^{*-1}\epsilon_\mathtt{B} \\ \epsilon_\mathtt{B} \end{pmatrix}.$$

- This results in the elimination of the top right hand block.

A Sparse Levenberg–Marquardt Algorithm

- The top half of this set of equations is

$$(\mathtt{U}^* - \mathtt{W}\mathtt{V}^{*-1}\mathtt{W}^\top)\boldsymbol{\delta}_{\mathbf{a}} = \boldsymbol{\epsilon}_{\mathtt{A}} - \mathtt{W}\mathtt{V}^{*-1}\boldsymbol{\epsilon}_{\mathtt{B}}.$$

- This is also known as the Schur's complement; and these equations may be solved to find $\delta_{\mathbf{a}}$.

- Subsequently, the value of $\delta_{\mathbf{b}}$ may be found by back-substitution, giving

$$\mathrm{V}^*\delta_\mathrm{b} = \mathrm{W}^\top\delta_\mathrm{a} - \epsilon_B$$

A Sparse Levenberg–Marquardt Algorithm

- If the newly computed value of the parameter vector $\mathbf{P} = \left((\mathbf{a} + \delta\mathbf{a})^\top, (\mathbf{b} + \delta\mathbf{b})^\top\right)^\top$ results in a diminished value of the error function.
- Then accept the new parameter vector **P**, diminishes the value of λ by a factor of 10, and proceeds to the next iteration.
- On the other hand, if the error value is increased, then reject the new **P** and tries again with a new value of λ, increased by a factor of 10.

A Sparse Levenberg–Marquardt Algorithm

Given A vector of measurements $\mathbf{X}$ with covariance matrix $\Sigma_{\mathbf{X}}$, an initial estimate of a set of parameters $\mathbf{P} = (\mathbf{a}^{\mathsf{T}}, \mathbf{b}^{\mathsf{T}})^{\mathsf{T}}$ and a function $f : \mathbf{P} \mapsto \widehat{\mathbf{X}}$ taking the parameter vector to an estimate of the measurement vector.

Objective Find the set of parameters $\mathbf{P}$ that minimizes $\epsilon^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\epsilon$ where $\epsilon = \mathbf{X} - \widehat{\mathbf{X}}$.

Algorithm

(i) Initialize a constant $\lambda = 0.001$ (typical value).

(ii) Compute the derivative matrices $\mathtt{A} = [\partial\widehat{\mathbf{X}}/\partial\mathbf{a}]$ and $\mathtt{B} = [\partial\widehat{\mathbf{X}}/\partial\mathbf{b}]$ and the error vector ϵ.

(iii) Compute intermediate expressions

$$\mathtt{U} = \mathtt{A}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\mathtt{A} \qquad \mathtt{V} = \mathtt{B}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\mathtt{B} \qquad \mathtt{W} = \mathtt{A}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\mathtt{B}$$

$$\epsilon_{\mathtt{A}} = \mathtt{A}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\epsilon \qquad \epsilon_{\mathtt{B}} = \mathtt{B}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\epsilon$$

(iv) Augment $\mathtt{U}$ and $\mathtt{V}$ by multiplying their diagonal elements by $1 + \lambda$.

(v) Compute the inverse $\mathtt{V}^{*-1}$, and define $\mathtt{Y} = \mathtt{W}\mathtt{V}^{*-1}$. The inverse may overwrite the value of $\mathtt{V}^*$ which will not be needed again.

(vi) Find $\boldsymbol{\delta}_{\mathbf{a}}$ by solving $(\mathtt{U}^* - \mathtt{Y}\mathtt{W}^{\mathsf{T}})\boldsymbol{\delta}_{\mathbf{a}} = \epsilon_{\mathtt{A}} - \mathtt{Y}\epsilon_{\mathtt{B}}$.

(vii) Find $\boldsymbol{\delta}_{\mathbf{b}}$ by back-substitution: $\boldsymbol{\delta}_{\mathbf{b}} = \mathtt{V}^{*-1}(\epsilon_{\mathtt{B}} - \mathtt{W}^{\mathsf{T}}\boldsymbol{\delta}_{\mathbf{a}})$.

(viii) Update the parameter vector by adding the incremental vector $(\boldsymbol{\delta}_{\mathbf{a}}^{\mathsf{T}}, \boldsymbol{\delta}_{\mathbf{b}}^{\mathsf{T}})^{\mathsf{T}}$ and compute the new error vector.

(ix) If the new error is less than the old error, then accept the new values of the parameters, diminish the value of λ by a factor of 10, and start again at step (ii), or else terminate.

(x) If the new error is greater than the old error, then revert to the old parameter values, increase the value of λ by a factor of 10, and try again from step (iv).

Table Source: Page 605, R. Hartley and A. Zisserman, “Multiple View Geometry in Computer Vision”

Sparse LM on Multiple Image Bundle Adjustment

- We take advantage of the lack of interaction between parameters of the different cameras.

- **Measurement Data**:

  $\mathbf{X} = \{\mathbf{X}_1, \ldots, \mathbf{X}_i\}$, where $\mathbf{X}_i = \left(\mathbf{x}_{i1}^\top, \mathbf{x}_{i2}^\top, \ldots, \mathbf{x}_{ij}^\top\right)^\top$ and

  $\mathbf{x}_{ij}$ is the image of the i-th 3D point in the j-th image .

- The **parameter vector:**

  $\mathbf{a} = \left(\mathbf{a}_1^\top, \mathbf{a}_2^\top, \ldots, \mathbf{a}_j^\top\right)^\top$, where $\boldsymbol{a}_j$ are the parameters of the j-th camera.

  $\mathbf{b} = \left(\mathbf{b}_1^\top, \mathbf{b}_2^\top, \ldots, \mathbf{b}_i^\top\right)^\top$, where $\boldsymbol{b}_i$ are the parameters of the i-th 3D point.

Sparse LM on Multiple Image Bundle Adjustment

- Since the image point $\mathbf{x}_{ij}$ does not depend on the parameters of any but the j-th camera,

$$\partial\hat{\mathbf{x}}_{ij}/\partial\mathbf{a}_k = 0 \qquad \text{unless} \qquad j = k.$$

- In a similar way for derivatives with respect to the parameters $\mathbf{b}_k$ of the k-th 3D point,

$$\partial\hat{\mathbf{x}}_{ij}/\partial\mathbf{b}_k = 0 \qquad \text{unless} \qquad i = k.$$

Sparse LM on Multiple Image Bundle Adjustment

- The form of the Jacobian matrix J for this problem and the resulting normal equations $J^\top J\boldsymbol{\delta} = -J^\top \epsilon$ are shown in the figures.

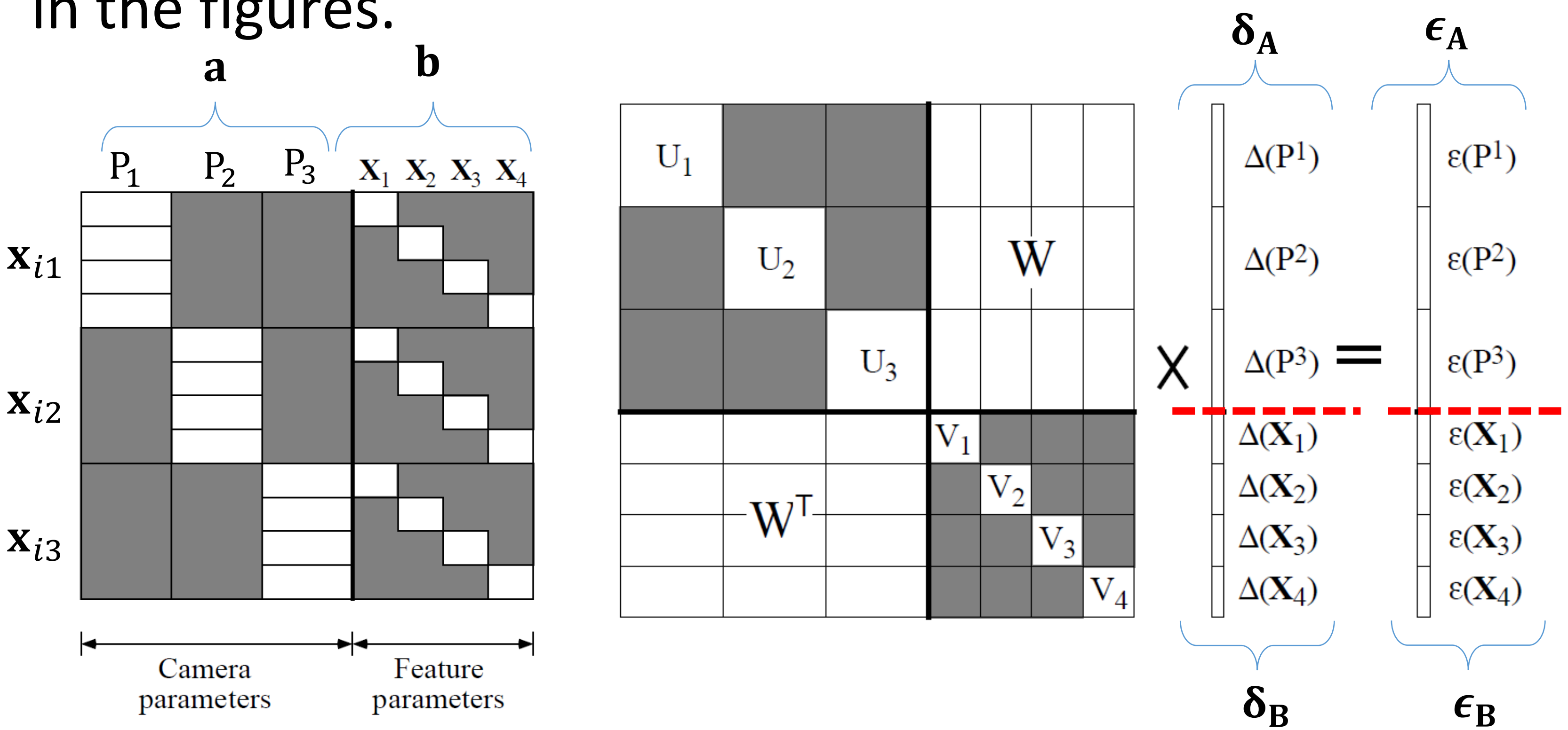


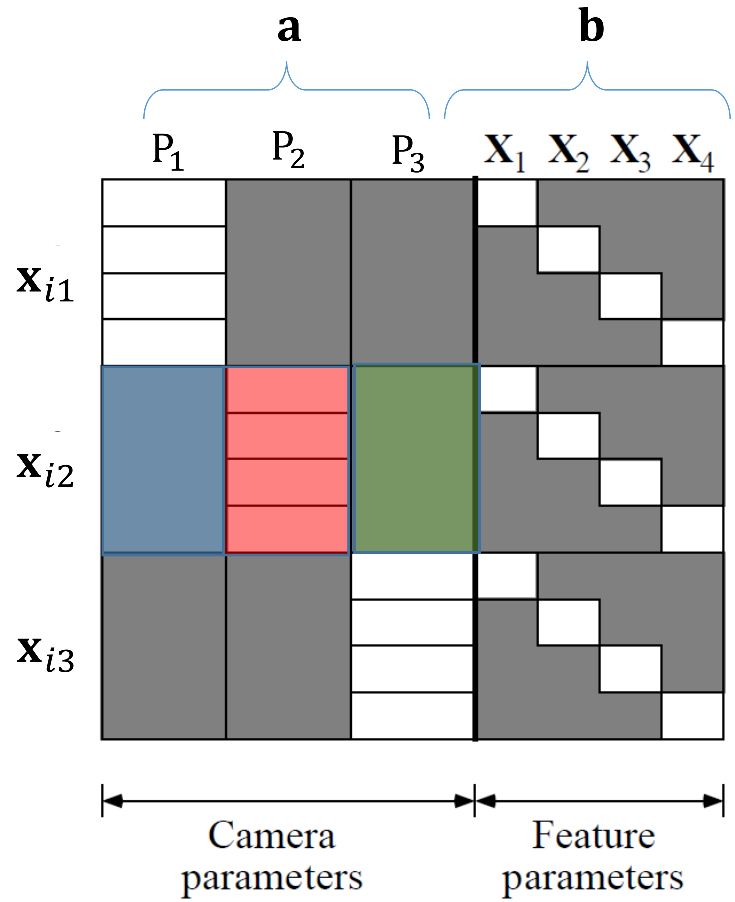
Image Source: R. Hartley and A. Zisserman, “Multiple View Geometry in Computer Vision”

Sparse LM on Multiple Image Bundle Adjustment

- We can see that $\mathtt{A}_i = [\partial \hat{\mathbf{X}}_i / \partial \mathbf{a}]$ is a block diagonal matrix

$$\mathtt{A}_i = \mathrm{diag}(\mathtt{A}_{i1}, \ldots, \mathtt{A}_{im}),$$

where $\mathtt{A}_{ij} = \partial \hat{\mathbf{x}}_{ij} / \partial \mathbf{a}_j$.



Example: i^{th} 3D Point in the 2nd camera , i.e. $\mathbf{x}_{i2}$

$$\frac{\partial \mathbf{x}_{i2}}{\partial P_1} = 0, \quad \frac{\partial \mathbf{x}_{i2}}{\partial P_2} \neq 0, \quad \frac{\partial \mathbf{x}_{i2}}{\partial P_3} = 0$$

Image Source: R. Hartley and A. Zisserman, “Multiple View Geometry in Computer Vision”

Sparse LM on Multiple Image Bundle Adjustment

- Similarly, matrix $\mathrm{B}_i = [\partial \hat{\mathbf{X}}_i / \partial \mathbf{b}_i]$ decomposes as:

$$\mathrm{B}_i = [\mathrm{B}_{i1}^\mathsf{T}, \ldots, \mathrm{B}_{im}^\mathsf{T}]^\mathsf{T},$$

where $\mathrm{B}_{ij} = \partial \hat{\mathbf{x}}_{ij} / \partial \mathbf{b}_i$.

Example: i^{th} 3D Point in the 2nd camera , i.e. $\mathbf{x}_{i2}$

$$\frac{\partial \mathbf{x}_{12}}{\partial \mathbf{X}_1} \neq 0, \quad \frac{\partial \mathbf{x}_{22}}{\partial \mathbf{X}_2} \neq 0, \quad \frac{\partial \mathbf{x}_{32}}{\partial \mathbf{X}_3} \neq 0$$

$$\frac{\partial \mathbf{x}_{42}}{\partial \mathbf{X}_4} \neq 0, \quad \frac{\partial \mathbf{x}_{i2}}{\partial \mathbf{X}_k} = 0, k \neq i$$

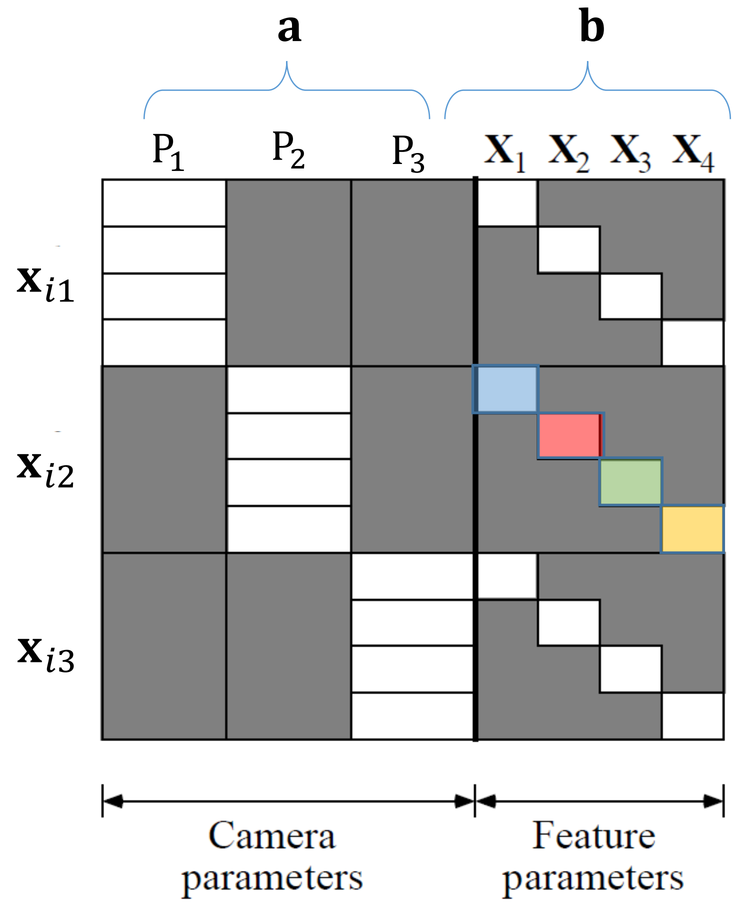


Image Source: R. Hartley and A. Zisserman, "Multiple View Geometry in Computer Vision"

Hessian and Adjacency Relation

- The Hessian matrix $\mathrm{H} = \mathrm{J}^{\mathrm{T}}\mathrm{J}$ is also the adjacency matrix of the graph that relates the parameters.

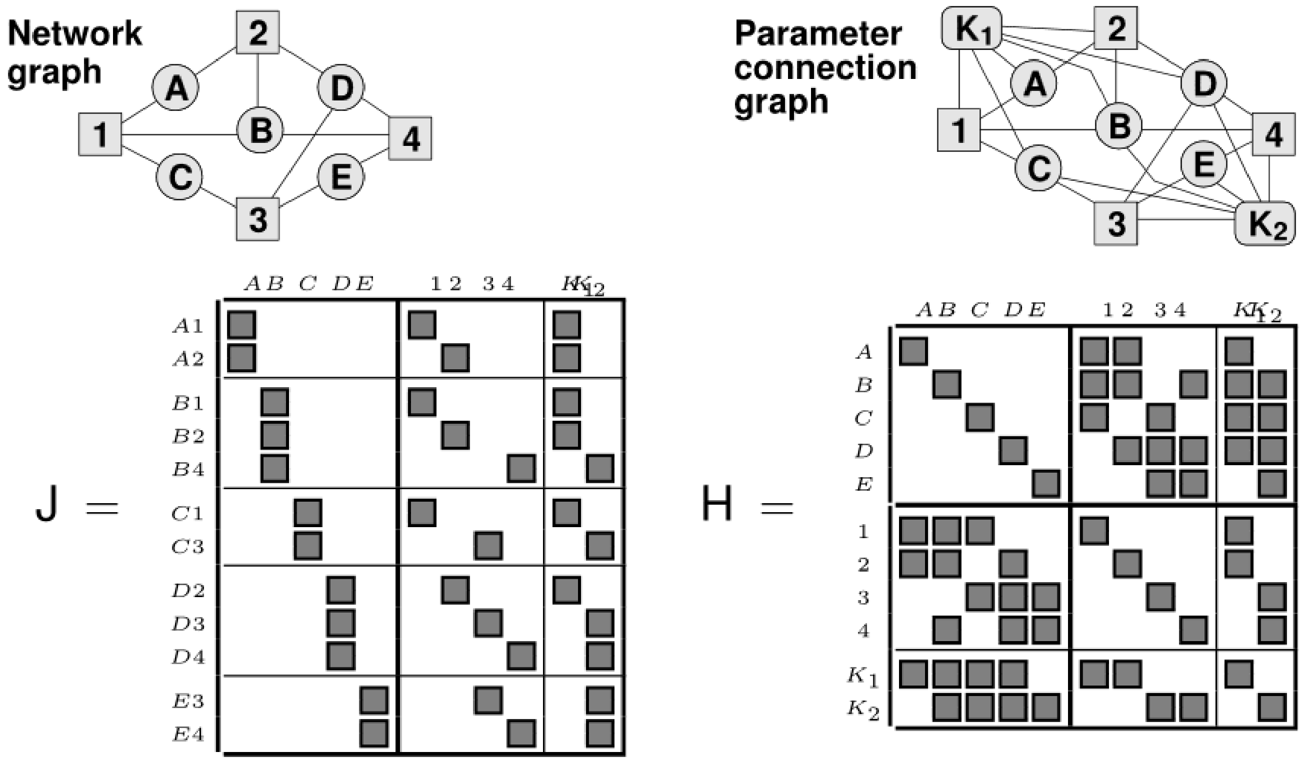
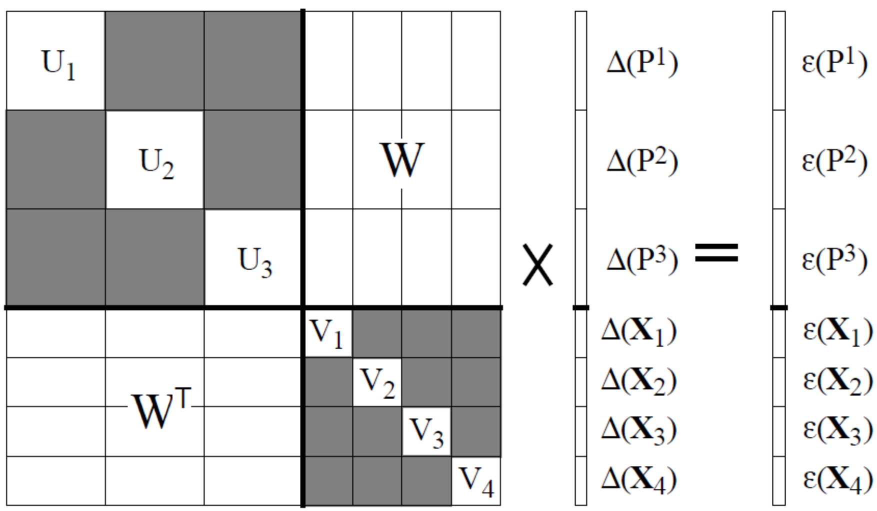


Image source: Bill Triggs et al, "Bundle Adjustment – A Modern Synthesis", 1999

Solving the Normal Equation

- As mentioned earlier, we use the Schur's complement to solve the normal equations.

$$\mathrm{H}_{LM}\delta = -\mathrm{J}^{\mathsf{T}}\Sigma\epsilon \quad \Rightarrow \quad \left[\begin{array}{c|c} \mathtt{A}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\mathtt{A} & \mathtt{A}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\mathtt{B} \\ \hline \mathtt{B}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\mathtt{A} & \mathtt{B}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\mathtt{B} \end{array}\right] \left(\begin{array}{c} \boldsymbol{\delta}_{\mathrm{a}} \\ \hline \boldsymbol{\delta}_{\mathrm{b}} \end{array}\right) = \left(\begin{array}{c} \mathtt{A}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\epsilon \\ \hline \mathtt{B}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\epsilon \end{array}\right)$$



$$\Downarrow$$

$$\left[\begin{array}{cc} \mathtt{U}^* & \mathtt{W} \\ \mathtt{W}^{\mathsf{T}} & \mathtt{V}^* \end{array}\right] \left(\begin{array}{c} \boldsymbol{\delta}_{\mathrm{a}} \\ \boldsymbol{\delta}_{\mathrm{b}} \end{array}\right) = \left(\begin{array}{c} \epsilon_{\mathtt{A}} \\ \epsilon_{\mathtt{B}} \end{array}\right).$$

Image Source: R. Hartley and A. Zisserman, "Multiple View Geometry in Computer Vision"

Solving the Normal Equation

- As mentioned earlier, we use the Schur's complement to solve the normal equations.

$$\begin{bmatrix} U^* & W \\ W^T & V^* \end{bmatrix} \begin{pmatrix} \delta_a \\ \delta_b \end{pmatrix} = \begin{pmatrix} \epsilon_A \\ \epsilon_B \end{pmatrix}$$

$\leftarrow$ Camera Parameters

$\leftarrow$ 3D Structures

$$\Downarrow \quad \times \begin{bmatrix} I & -WV^{*-1} \\ 0 & I \end{bmatrix}$$

$$\begin{bmatrix} U^* - WV^{*-1}W^T & 0 \\ W^T & V^* \end{bmatrix} \begin{pmatrix} \delta_a \\ \delta_b \end{pmatrix} = \begin{pmatrix} \epsilon_A - WV^{*-1}\epsilon_B \\ \epsilon_B \end{pmatrix}.$$

Solving the Normal Equation

- As mentioned earlier, we use the Schur's complement to solve the normal equations.

$$\begin{bmatrix} U^* - WV^{*-1}W^\top & 0 \\ W^\top & V^* \end{bmatrix} \begin{pmatrix} \delta_a \\ \delta_b \end{pmatrix} = \begin{pmatrix} \epsilon_A - WV^{*-1}\epsilon_B \\ \epsilon_B \end{pmatrix}$$

- First solve for $\boldsymbol{\delta}_\mathbf{a}$ from:

Easy to invert a block diagonal matrix

$$\underbrace{(U^* - WV^{*-1}W^\top)}_{\substack{\text{Schur Complement} \\ \text{(Sparse and Symmetric Positive Definite Matrix)}}}\boldsymbol{\delta}_\mathbf{a} = \epsilon_A - WV^{*-1}\epsilon_B$$

- Then Solve for $\boldsymbol{\delta}_\mathbf{b}$ by backward substitution.

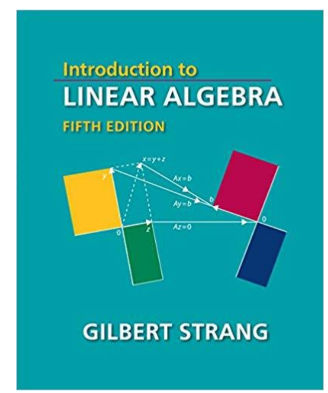
Solving the Normal Equation

- The non-homogeneous linear system of equations can be solved without inverting A since it is a sparse matrix!

$$(\mathrm{U}^* - \mathrm{W}\mathrm{V}^{*-1}\mathrm{W}^\top)\boldsymbol{\delta}_{\mathbf{a}} = \epsilon_{\mathrm{A}} - \mathrm{W}\mathrm{V}^{*-1}\epsilon_{\mathrm{B}} \Rightarrow \mathrm{A}\mathbf{x} = \mathrm{b}$$

- Sparse matrix factorization
  1. LU Factorization $\longrightarrow$ $A = LU$
  2. QR factorization $\longrightarrow$ $A = QR$
  3. Cholesky Factorization $\longrightarrow$ $A = LL^\top$

Solve for x by forward backward substitutions.



- Iterative methods
  1. Conjugate gradient
  2. Gauss-Seidel

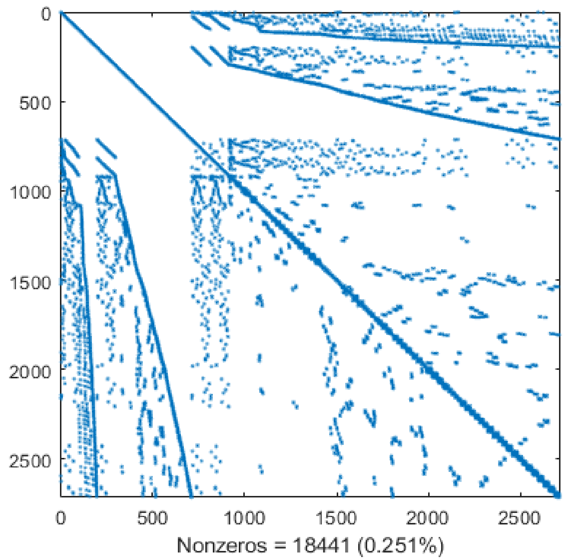
Linear Algebra reference:
Gilbert Strang, "Introduction to Linear Algebra", Fifth Edition, 2016

Problem of Fill-In

- Sparse matrix factorization can lead to the problem of fill-in, where the factorized matrix becomes dense.

Example:

Sparse matrix A



$A = LL^{\mathrm{T}}$

After Cholesky Decomposition L

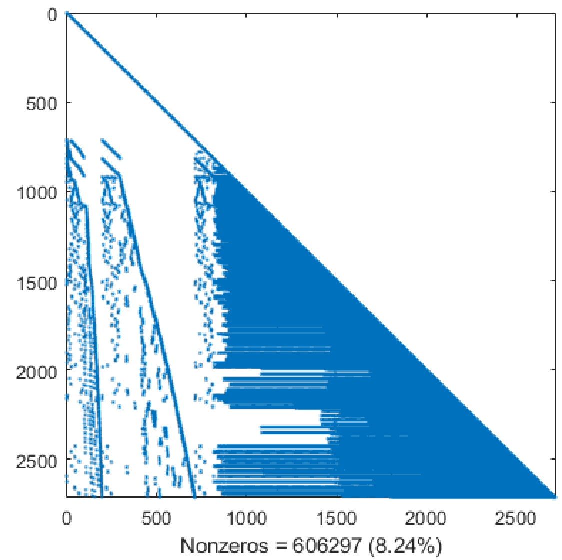


Image source: https://www.mathworks.com/help/matlab/math/sparse-matrix-reordering.html

Problem of Fill-In

- Reorder sparse matrix to minimize fill-in.

$$(\mathrm{P}^\top \mathrm{AP})(\mathrm{P}^\top \mathbf{x}) = \mathrm{P}^\top \mathbf{b}$$

Permutation matrix to reorder A

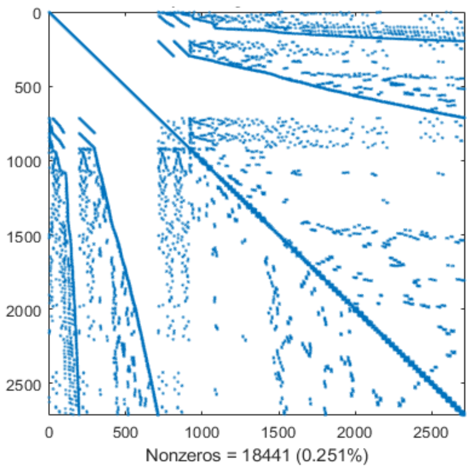
- NP-Complete problem.

- Approximate solutions:
    1. Minimum degree
    2. Column approximate minimum degree permutation
    3. Reverse Cuthill-Mckee
    4. Nested Dissection …

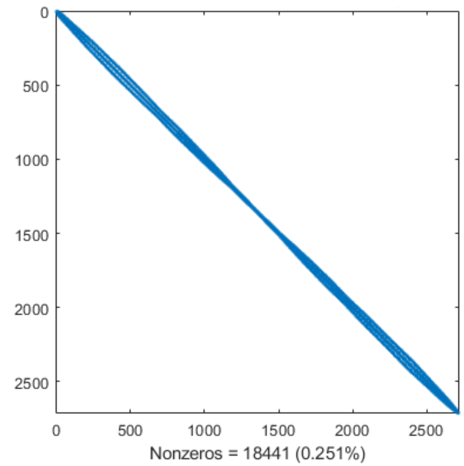
Reference: https://www.mathworks.com/help/matlab/math/sparse-matrix-reordering.html

Problem of Fill-In: Examples

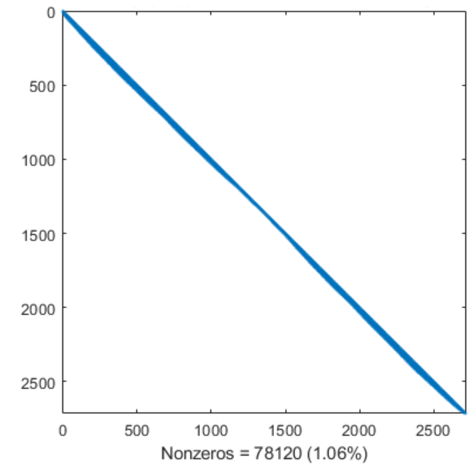
Sparse matrix A



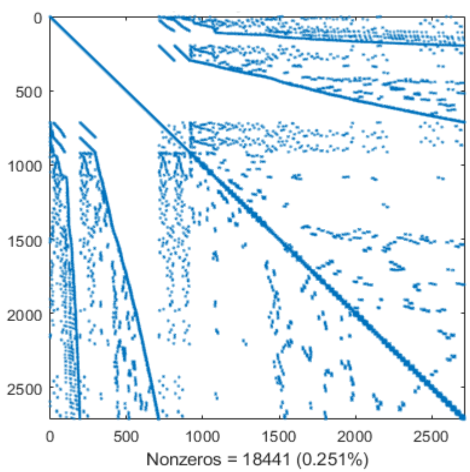
After Cuthill-McKee Ordering $P^{T}AP$



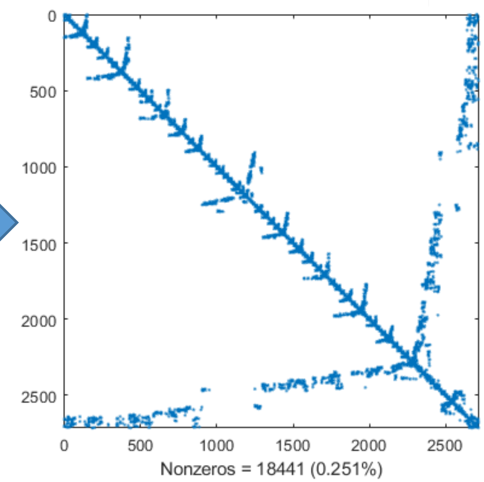
After Cholesky Decomposition L



Sparse matrix A



After Minimum Degree Ordering $P^{T}AP$



After Cholesky Decomposition L

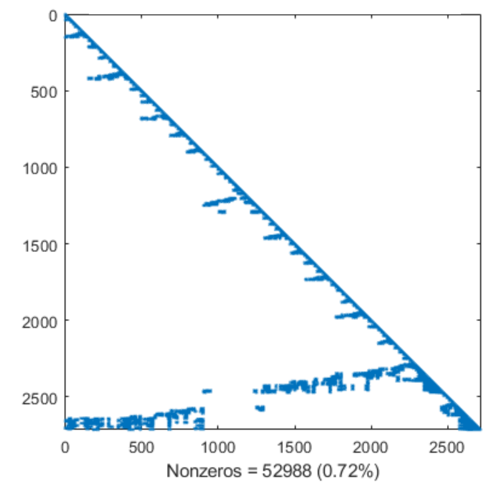
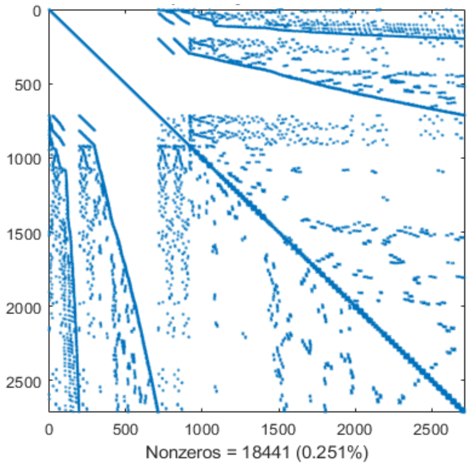


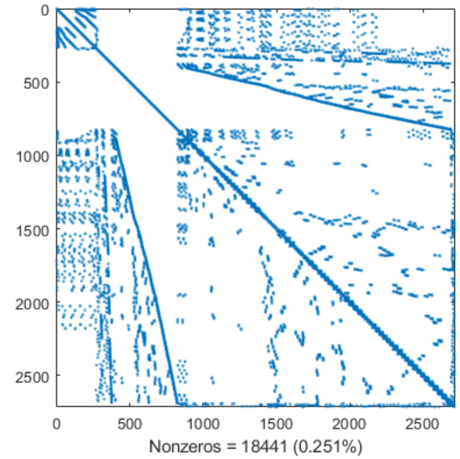
Image source: https://www.mathworks.com/help/matlab/math/sparse-matrix-reordering.html

Problem of Fill-In: Examples

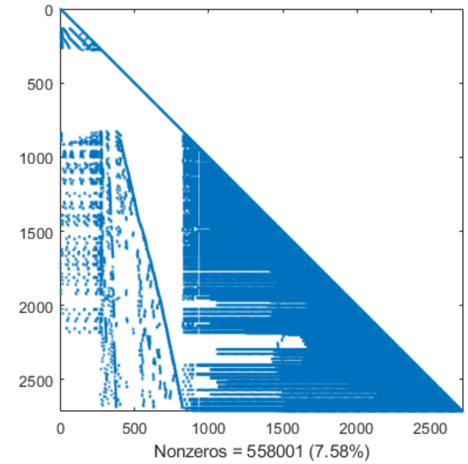
Sparse matrix A



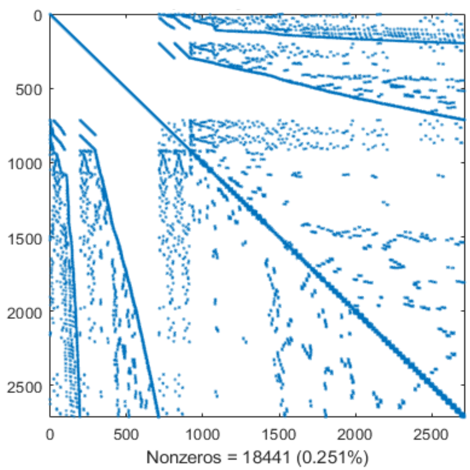
After Column Count Ordering $P^T AP$



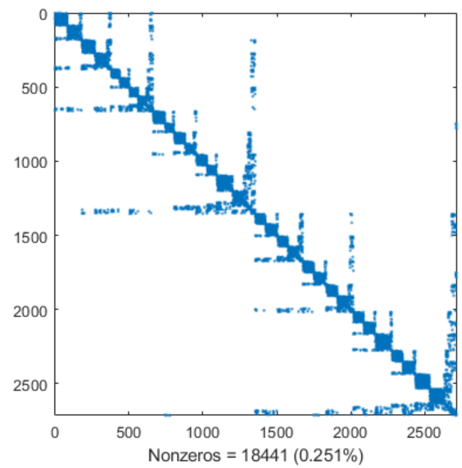
After Cholesky Decomposition L



Sparse matrix A



After Nested Dissection Ordering $P^T AP$



After Cholesky Decomposition L

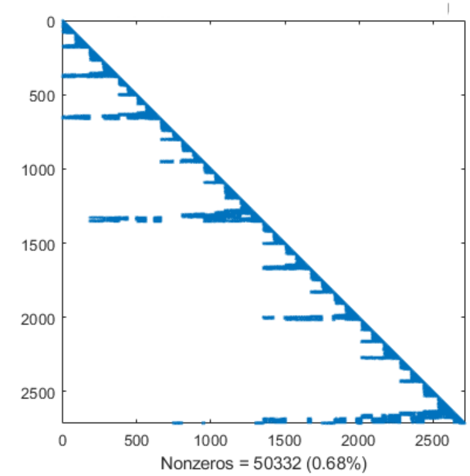


Image source: https://www.mathworks.com/help/matlab/math/sparse-matrix-reordering.html

Problem of Fill-In: Examples

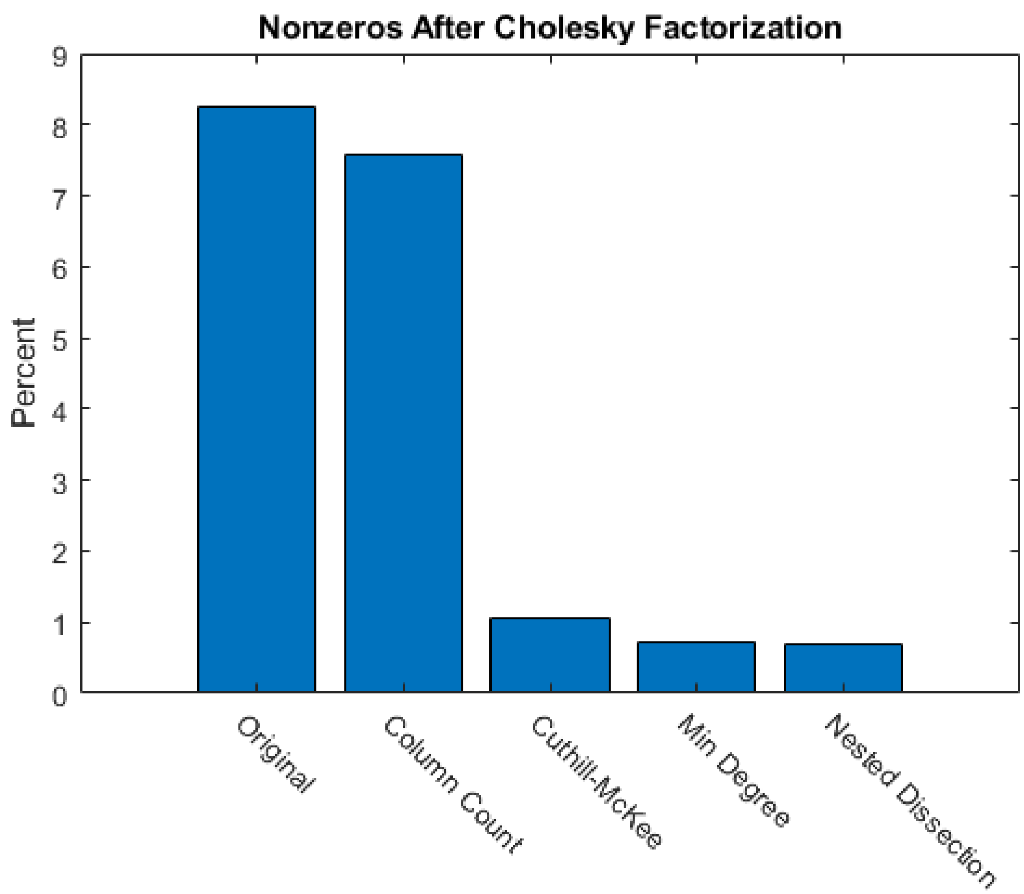


Image source: https://www.mathworks.com/help/matlab/math/sparse-matrix-reordering.html

Open-Source Software

- Google Ceres: http://ceres-solver.org/
- g2o: https://github.com/RainerKuemmerle/g2o
- gtsam: https://bitbucket.org/gtborg/gtsam/
- Colmap: https://colmap.github.io/

Image source: https://colmap.github.io/

Summary

- We have looked at how to:

1. Describe the pipeline of large-scale 3D reconstruction: data association, structure-from-motion and dense stereo.

2. Explain the use of robust two-view geometry and the bag-of-words algorithm for data association.

3. Use two-view geometry, PnP and triangulation to initialize the 3D reconstruction.

4. Apply the iterative methods: Newton, Gauss-Newton Gradient descent or Levenberg-Marquardt for bundle adjustment.