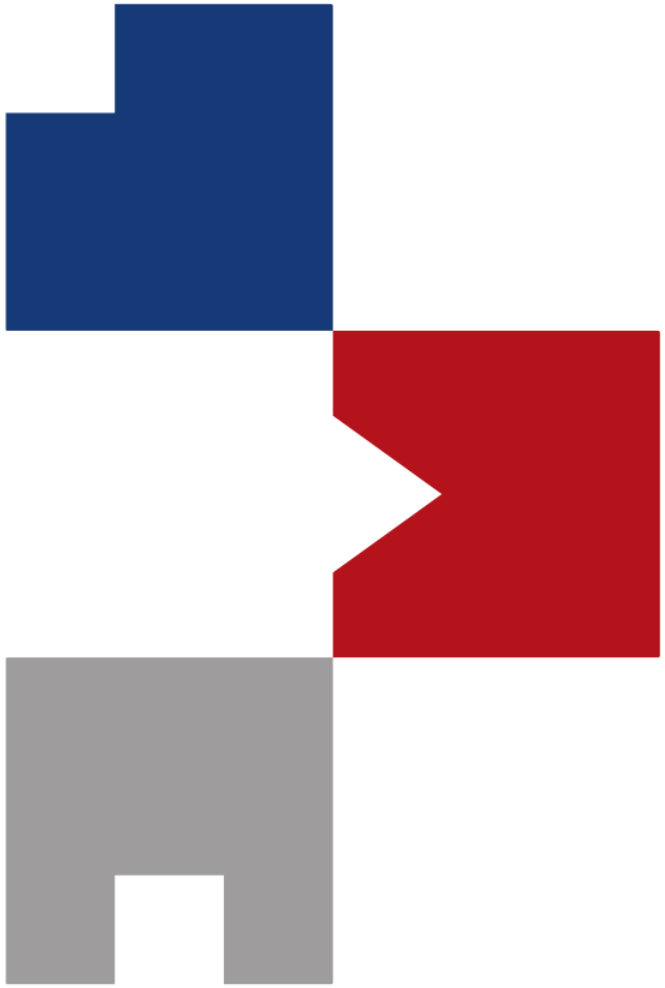
Note) Slides and example codes are available:
https://github.com/mint-lab/3dv_tutorial



An Invitation to 3D Vision: Multi-View Geometry

Sunglok Choi, Assistant Professor, Ph.D.
Dept. of Computer Science and Engineering, SEOULTECH
sunglok@seoultech.ac.kr | https://mint-lab.github.io/

Multi-view Geometry

Table of Contents: Multi-view Geometry

- **Bundle Adjustment**
- **Structure-from-Motion**
  - COLMAP (2016)

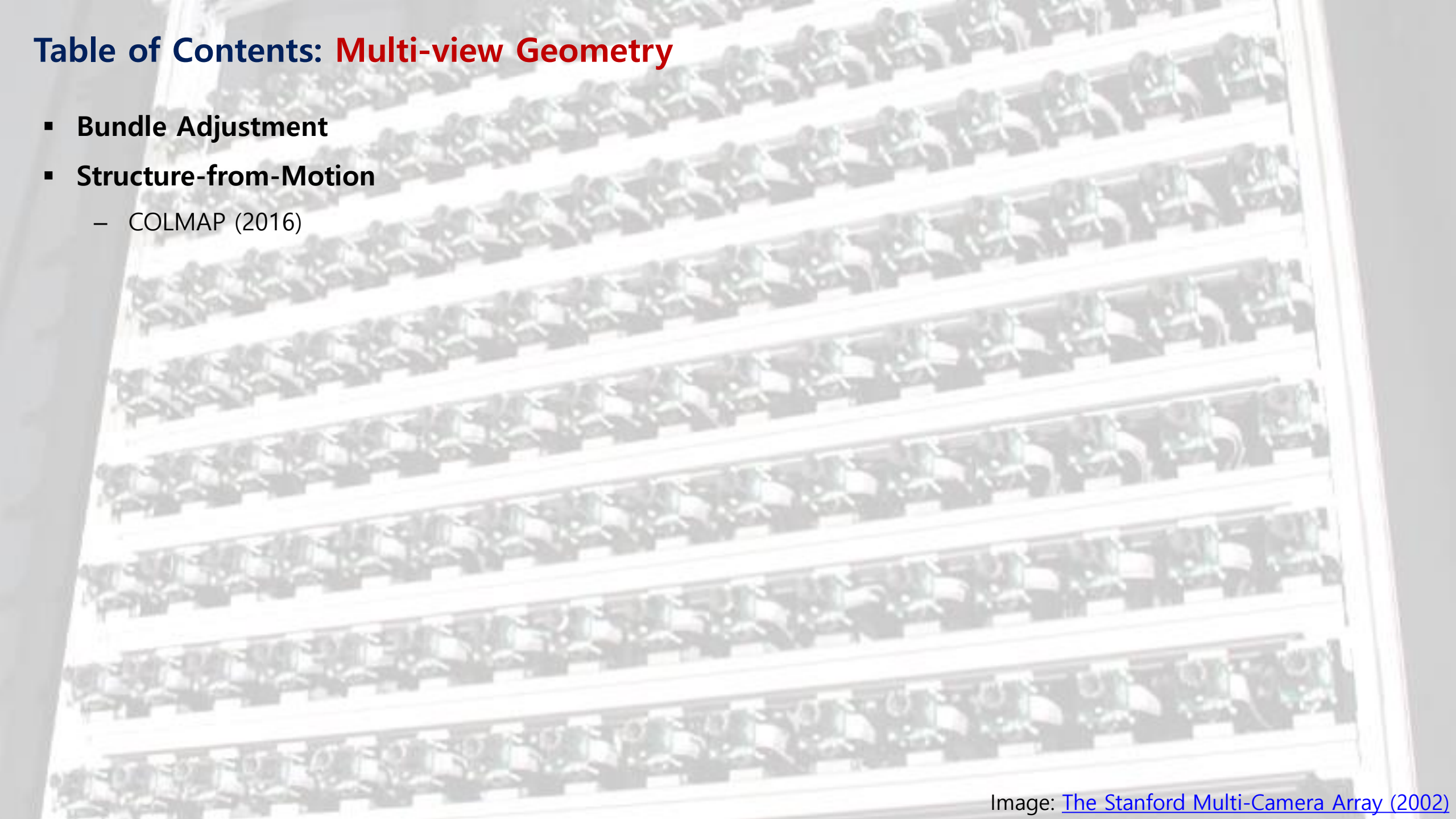


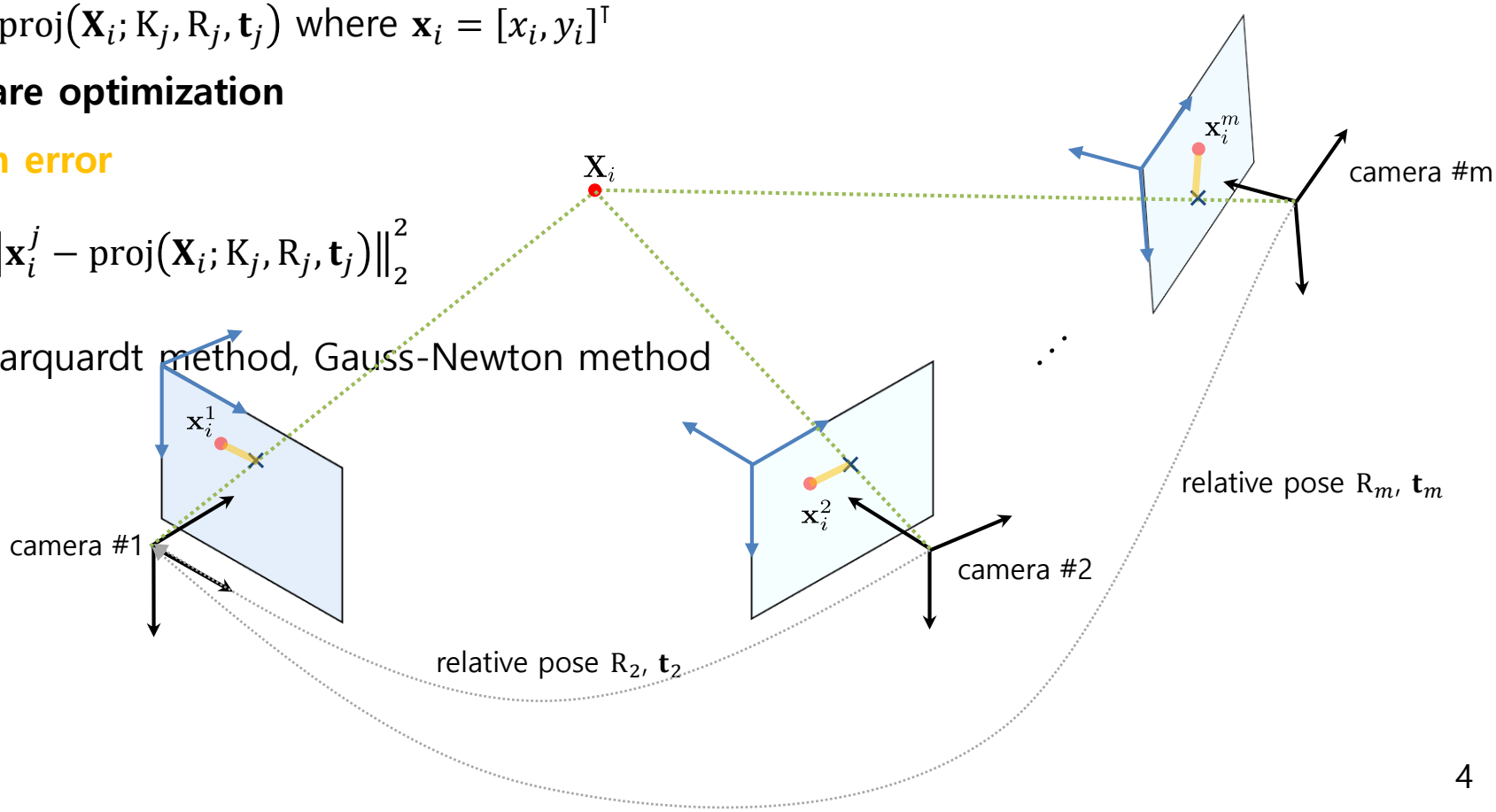
Image: The Stanford Multi-Camera Array (2002)

Bundle Adjustment

- **Bundle adjustment** (shortly BA)
  - Unknown: 3D points ($\mathbf{X}_1$, $\mathbf{X}_2$, …, $\mathbf{X}_i$, …, $\mathbf{X}_n$) and each camera's relative pose (R_j, $\mathbf{t}_j$) ($3n + 6m$ DOF)
  - Given: Point correspondence $\mathbf{x}_i^j$, camera matrices K_j, 3D points $\mathbf{X}_i$, and each camera's relative pose (R_j, $\mathbf{t}_j$)
  - ~~Constraints: m x n x projection $\mathbf{x}_i^j = \mathrm{K}_j \left[\mathrm{R}_j | \mathbf{t}_j \right] \mathbf{X}_i$ where $\mathbf{x}_i = [x_i, y_i, w_i]^\intercal$~~
  - More general constraints: $\mathbf{x}_i^j = \mathrm{proj}\left(\mathbf{X}_i; \mathrm{K}_j, \mathrm{R}_j, \mathbf{t}_j\right)$ where $\mathbf{x}_i = [x_i, y_i]^\intercal$
  - Solution: **Non-linear least-square optimization**
    - Cost Function: **Reprojection error**

$$\hat{\mathbf{X}}_{\dots}, \hat{\mathrm{R}}_{\dots}, \hat{\mathbf{t}}_{\dots} = \underset{\mathbf{X}_{\dots}, \mathrm{R}_{\dots}, \mathbf{t}_{\dots}}{\operatorname{argmin}} \sum_{j=1}^{m} \sum_{i=1}^{n} \left\| \mathbf{x}_i^j - \mathrm{proj}\left(\mathbf{X}_i; \mathrm{K}_j, \mathrm{R}_j, \mathbf{t}_j\right) \right\|_2^2$$

    - Optimization: Levenberg–Marquardt method, Gauss-Newton method



Bundle Adjustment

- **Bundle adjustment** (shortly BA)
  - Unknown: 3D points ($\mathbf{X}_1$, $\mathbf{X}_2$, …, $\mathbf{X}_i$, …, $\mathbf{X}_n$) and each camera's relative pose (R_j, $\mathbf{t}_j$) ($3n + 6m$ DOF)
  - Given: Point correspondence $\mathbf{x}_i^j$, camera matrices K_j, 3D points $\mathbf{X}_i$, and each camera's relative pose (R_j, $\mathbf{t}_j$)
  - ~~Constraints: m x n x projection $\mathbf{x}_i^j = \mathrm{K}_j \left[\mathrm{R}_j | \mathbf{t}_j \right] \mathbf{X}_i$ where $\mathbf{x}_i = [x_i, y_i, w_i]^\intercal$~~
  - More general constraints: $\mathbf{x}_i^j = \mathrm{proj}(\mathbf{X}_i; \mathrm{K}_j, \mathrm{R}_j, \mathbf{t}_j)$ where $\mathbf{x}_i = [x_i, y_i]^\intercal$
  - Solution: **Non-linear least-square optimization**
    - Cost Function: **Reprojection error**

$$\hat{\mathbf{X}}_{\dots}, \hat{\mathrm{R}}_{\dots}, \hat{\mathbf{t}}_{\dots} = \underset{\mathbf{X}_{\dots}, \mathrm{R}_{\dots}, \mathbf{t}_{\dots}}{\operatorname{argmin}} \sum_{j=1}^{m} \sum_{i=1}^{n} \left\| \mathbf{x}_i^j - \mathrm{proj}(\mathbf{X}_i; \mathrm{K}_j, \mathrm{R}_j, \mathbf{t}_j) \right\|_2^2$$

    - Optimization: Levenberg–Marquardt method, Gauss-Newton method

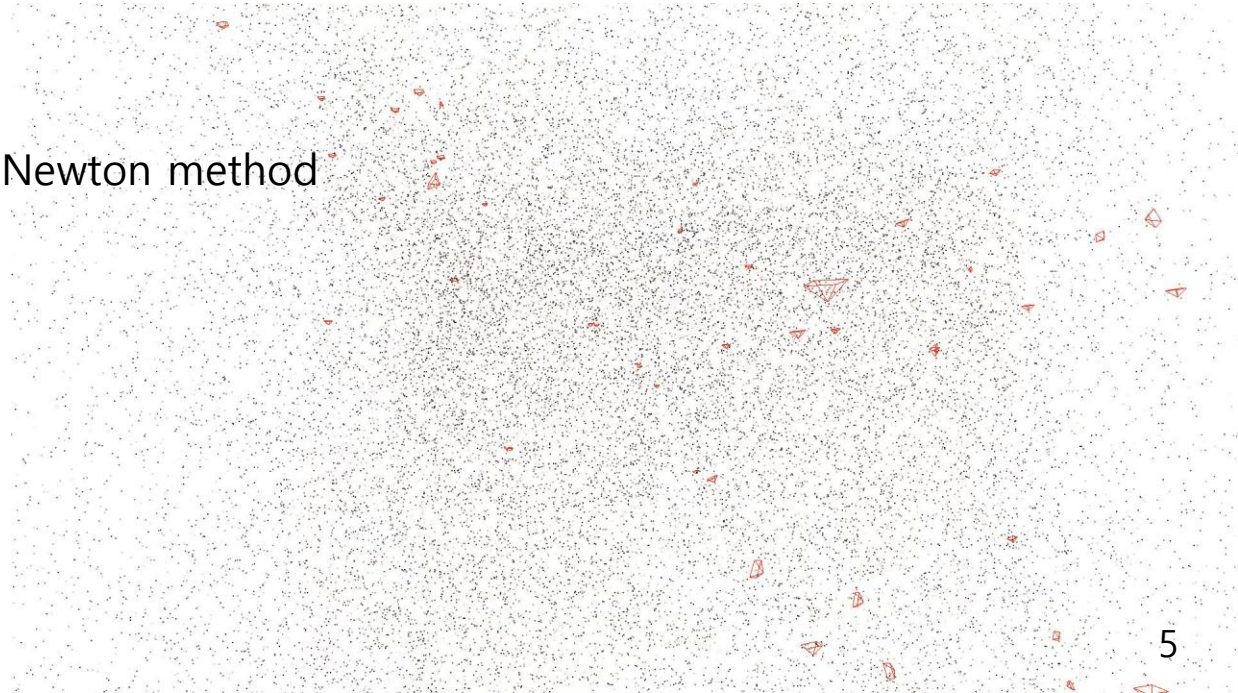


Image: GLOMAP

Bundle Adjustment

- **Bundle adjustment** (shortly BA)
  - Unknown: 3D points ($\mathbf{X}_1$, $\mathbf{X}_2$, …, $\mathbf{X}_i$, …, $\mathbf{X}_n$) and each camera's relative pose (R_j, $\mathbf{t}_j$) ($3n + 6m$ DOF)
  - Given: Point correspondence $\mathbf{x}_i^j$, camera matrices K_j, 3D points $\mathbf{X}_i$, and each camera's relative pose (R_j, $\mathbf{t}_j$)
  - More general constraints: $\mathbf{x}_i^j = \mathrm{proj}(\mathbf{X}_i; \mathrm{K}_j, \mathrm{R}_j, \mathbf{t}_j)$ where $\mathbf{x}_i = [x_i, y_i]^\mathsf{T}$
  - Solution: **Non-linear least-square optimization** $\sum_{i}^{n}\sum_{j}^{m} v_i^j \left\| \mathbf{x}_i^j - \mathrm{P}_j \mathbf{X}_i \right\|_\Sigma^2$ where $v_i^j = \begin{cases} 1 & \text{if } \mathbf{x}_i^j \text{ is visible} \\ 0 & \text{otherwise} \end{cases}$

    cf. Initial values
    - Cost Function: **Reprojection error**

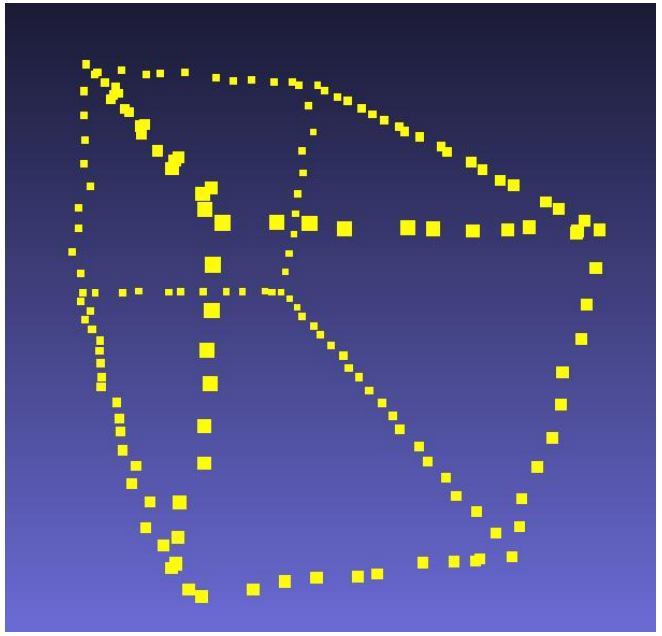
      $$\hat{\mathbf{X}}_{\dots}, \hat{\mathrm{R}}_{\dots}, \hat{\mathbf{t}}_{\dots} = \underset{\mathbf{X}_{\dots}, \mathrm{R}_{\dots}, \mathbf{t}_{\dots}}{\operatorname{argmin}} \sum_{j=1}^{m}\sum_{i=1}^{n} \left\| \mathbf{x}_i^j - \mathrm{proj}(\mathbf{X}_i; \mathrm{K}_j, \mathrm{R}_j, \mathbf{t}_j) \right\|_2^2$$

    - Optimization: Levenberg–Marquardt method, Gauss-Newton method
- **Bundle adjustment tools**
  - Bundle adjustment: sba (Sparse BA), SSBA (Simple Sparse BA), pba (Multicore BA), DABA (Decentralized and Accelerated Large-scale BA)
  - (Graph) optimization: g2o (General Graph Optimization), iSAM (Incremental Smoothing and Mapping), GTSAM, Ceres Solver in C++ / SciPy, PyTorch, … in Python

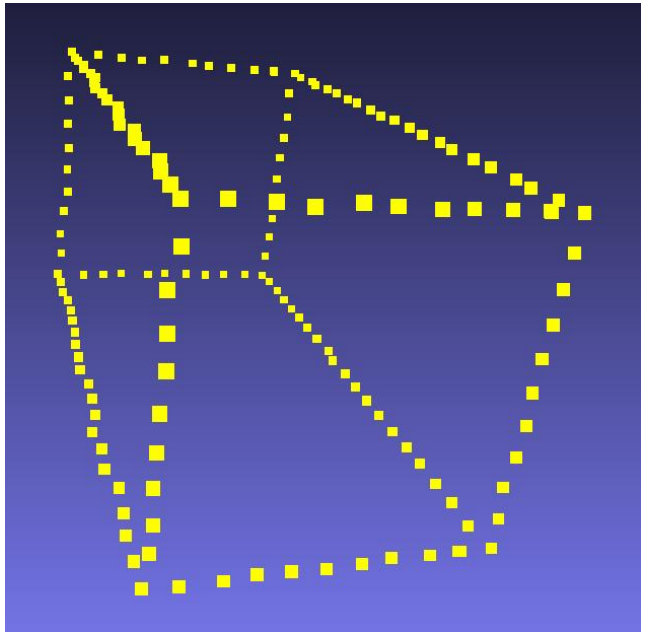
Bundle Adjustment

- Example) **Bundle adjustment** `[bundle_adjustment_global.cpp, bundle_adjustment.hpp]`

Result: "triangulation.xyz"

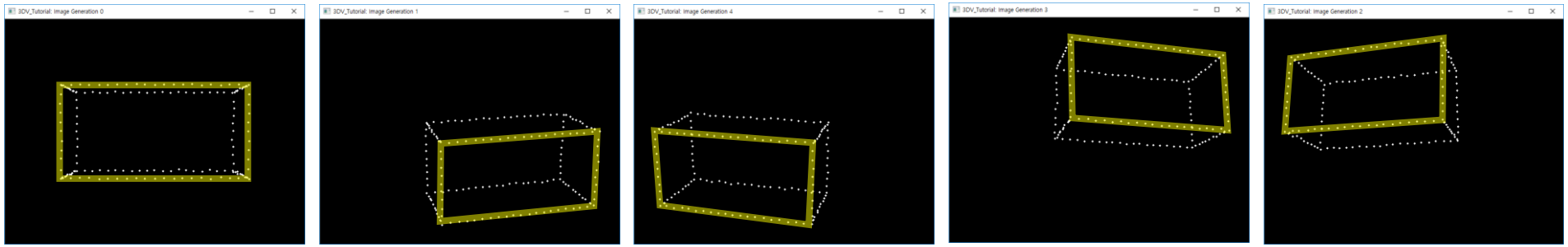


Result: "bundle_adjustment_global.xyz"



[Given]

- **Intrinsic parameters**: camera matrices K_j (all cameras has same and fixed camera matrix.)
- **2D point correspondence** $\mathbf{x}_i^j$ (all points are visible on all camera views.)



[Unknown]

- **Extrinsic parameters**: camera poses

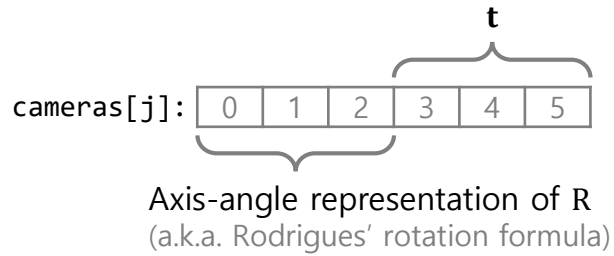
$$\mathrm{R}_0 = \mathrm{I}_{3x3} \quad \mathbf{t}_0 = [0,0,0]^\top$$
$$\mathrm{R}_1 = \mathrm{I}_{3x3} \quad \mathbf{t}_1 = [0,0,0]^\top$$
$$\mathrm{R}_2 = \mathrm{I}_{3x3} \quad \mathbf{t}_2 = [0,0,0]^\top$$
$$\mathrm{R}_3 = \mathrm{I}_{3x3} \quad \mathbf{t}_3 = [0,0,0]^\top$$
$$\mathrm{R}_4 = \mathrm{I}_{3x3} \quad \mathbf{t}_4 = [0,0,0]^\top$$

- **3D points** $\mathbf{X}_i = [0,0,5.5]^\mathsf{T}$ initial values

Ceres Solver (non-linear least-square optimization)

$$\sum_i^n \sum_j^m v_i^j \left\| \mathbf{x}_i^j - \mathrm{P}_j \mathbf{X}_i \right\|_\Sigma^2 \quad \text{where} \quad v_i^j = 1 \ (\because \text{all points are visible})$$

```cpp
#include "bundle_adjustment.hpp"

int main()
{
    // cf. You need to run 'image_formation.cpp' to generate point observation.
    const char* input = "image_formation%d.xyz";
    int input_num = 5;
    double f = 1000, cx = 320, cy= 240;

    // Load 2D points observed from multiple views
    std::vector<std::vector<cv::Point2d>> xs;
    ...

    // Initialize cameras and 3D points
    std::vector<cv::Vec6d> cameras(xs.size());
    std::vector<cv::Point3d> Xs(xs.front().size(), cv::Point3d(0, 0, 5.5));

    // Run bundle adjustment
    ceres::Problem ba;
    for (size_t j = 0; j < xs.size(); j++)
    {
        for (size_t i = 0; i < xs[j].size(); i++)
        {
            ceres::CostFunction* cost_func = ReprojectionError::create(xs[j][i], f, cv::Point2d(cx, cy));
            double* view = (double*)(&(cameras[j]));
            double* X = (double*)(&(Xs[i]));
            ba.AddResidualBlock(cost_func, NULL, view, X);
        }
    }
    ceres::Solver::Options options;
    options.linear_solver_type = ceres::ITERATIVE_SCHUR;
    options.num_threads = 8;
    options.minimizer_progress_to_stdout = true;
    ceres::Solver::Summary summary;
    ceres::Solve(options, &ba, &summary);

    // Store the 3D points and camera pose to an XYZ file
    ...
        cv::Vec3d rvec(cameras[j][0], cameras[j][1], cameras[j][2]), t(cameras[j][3], cameras[j][4], cameras[j][5]);
        cv::Matx33d R;
        cv::Rodrigues(rvec, R);
    ...
    return 0;
}
```

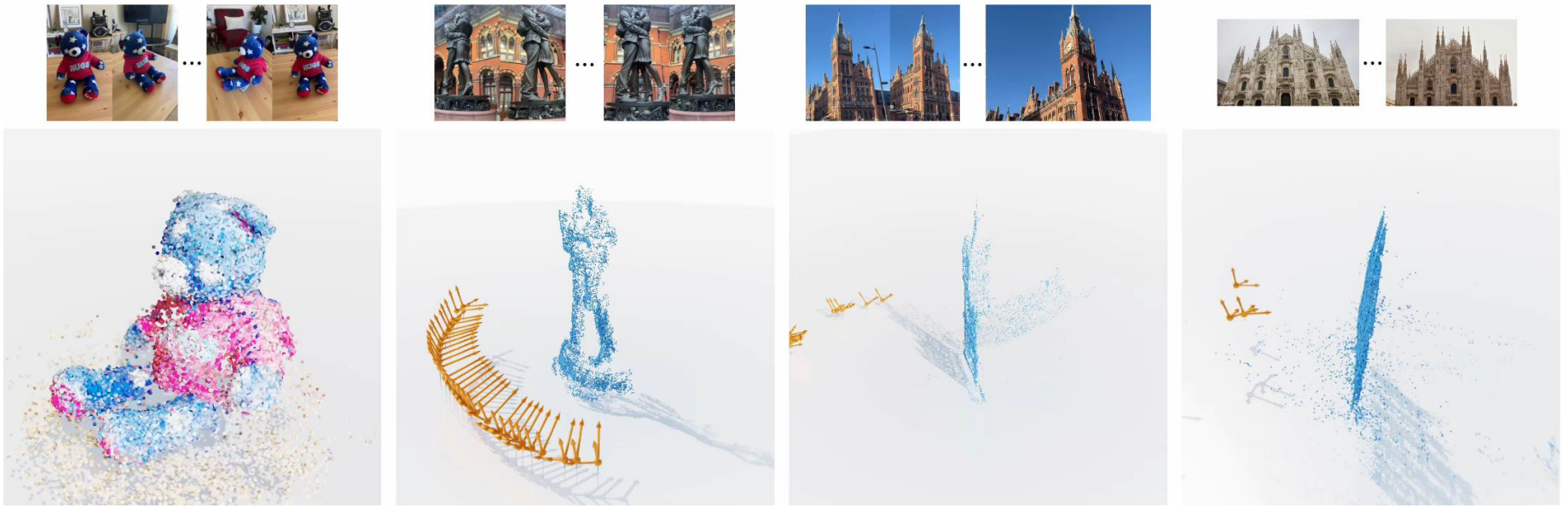


```cpp
#include "opencv2/opencv.hpp"
#include "ceres/ceres.h"
#include "ceres/rotation.h"

// Reprojection error for bundle adjustment
struct ReprojectionError
{
    ReprojectionError(const cv::Point2d& _x, double _f, cv::Point2d& _c) : x(_x), f(_f), c(_c) { }

    template <typename T>
    bool operator()(const T* const camera, const T* const point, T* residuals) const
    {
        // X' = R*X + t
        T X[3];
        ceres::AngleAxisRotatePoint(camera, point, X);
        X[0] += camera[3];
        X[1] += camera[4];
        X[2] += camera[5];

        // x' = K*X'
        T x_p = f * X[0] / X[2] + cx;
        T y_p = f * X[1] / X[2] + cy;

        // residual = x - x'
        residuals[0] = T(x.x) - x_p;
        residuals[1] = T(x.y) - y_p;
        return true;
    }

    static ceres::CostFunction* create(const cv::Point2d& _x, double _f, cv::Point2d& _c)
    {
        return (new ceres::AutoDiffCostFunction<ReprojectionError, 2, 6, 3>(new ReprojectionError(_x, _f, _c)));
    }

private:
    const cv::Point2d x;
    const double f;
    const cv::Point2d c;
};
```

$$\mathbf{x} - \mathrm{P}\mathbf{X}$$

Structure-from-Motion

- **Structure-from-Motion** (shortly SfM)
  - Goal: 3D structures + each camera's *relative pose*
  - Given: 2D images from different viewpoints (a.k.a. *relative pose* or *motion*)



- **Structure-from-Motion tools**
  - Classical SfM: Bundler, COLMAP, MVE, Theia, openMVG, OpenSfM, Toy SfM / VisualSFM (GUI, binary only)
    - DL-based Refinement: Pixel-Perfect SfM
  - DL-based SfM: DeepSFM, VGGSfM, GASFM

Image: VGGSfM

Structure-from-Motion

- Example) **Structure-from-Motion (global version)**
  0) Load images and extract features
  0) Match features and find good matches (which has enough inliers)

Note) **# of inliers** / # of matches

Structure-from-Motion

- Example) **Structure-from-Motion (global version)**

  0) Load images and extract features

  0) Match features and find good matches (which has enough inliers)

  1) Initialize camera parameters

$$\mathrm{K} = \begin{bmatrix} f & 0 & w/2 \\ 0 & f & h/2 \\ 0 & 0 & 1 \end{bmatrix}, \mathrm{R}_i = \mathrm{I}_{3\times3}, \text{ and } \mathbf{t}_i = [0,0,0]^\intercal$$

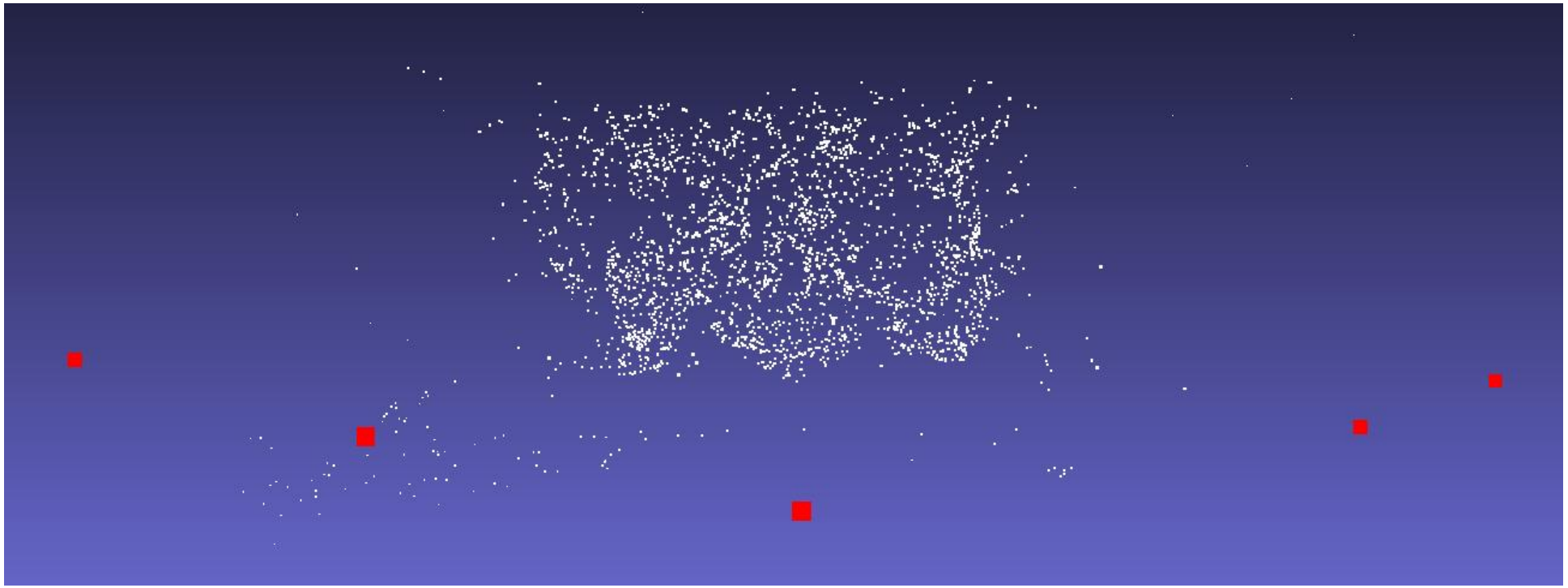
  2) Initialize 3D points and build a visibility graph

$$\mathrm{X}_i = [0,0,2]^\intercal$$

  3) Optimize camera poses and 3D points together **(BA)**

Structure-from-Motion

- Example) **Structure-from-Motion (batch version)** `[sfm_global.cpp]`



Structure-from-Motion

- Example) **Structure-from-Motion (incremental version)**

  0) Load images and extract features

  0) Build a viewing graph (well-matched pairs) while finding inliers

  1) Select the best pair

  2) Estimate relative pose from the best two views **(epipolar geometry)**

  3) Reconstruct 3D points of the best two views **(triangulation)**

  4) Select the next best image to add

     Separate points into known and unknown for PnP (known) and triangulation (unknown)

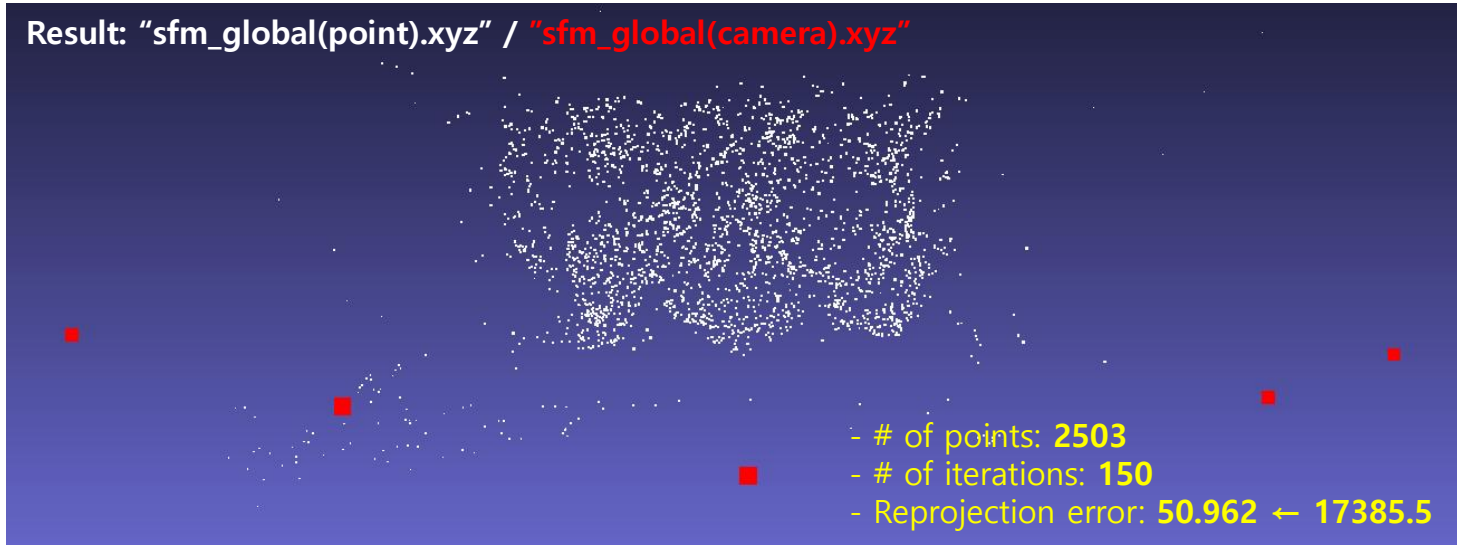
  5) Estimate relative pose of the next best view using (known) 3D points **(PnP)**

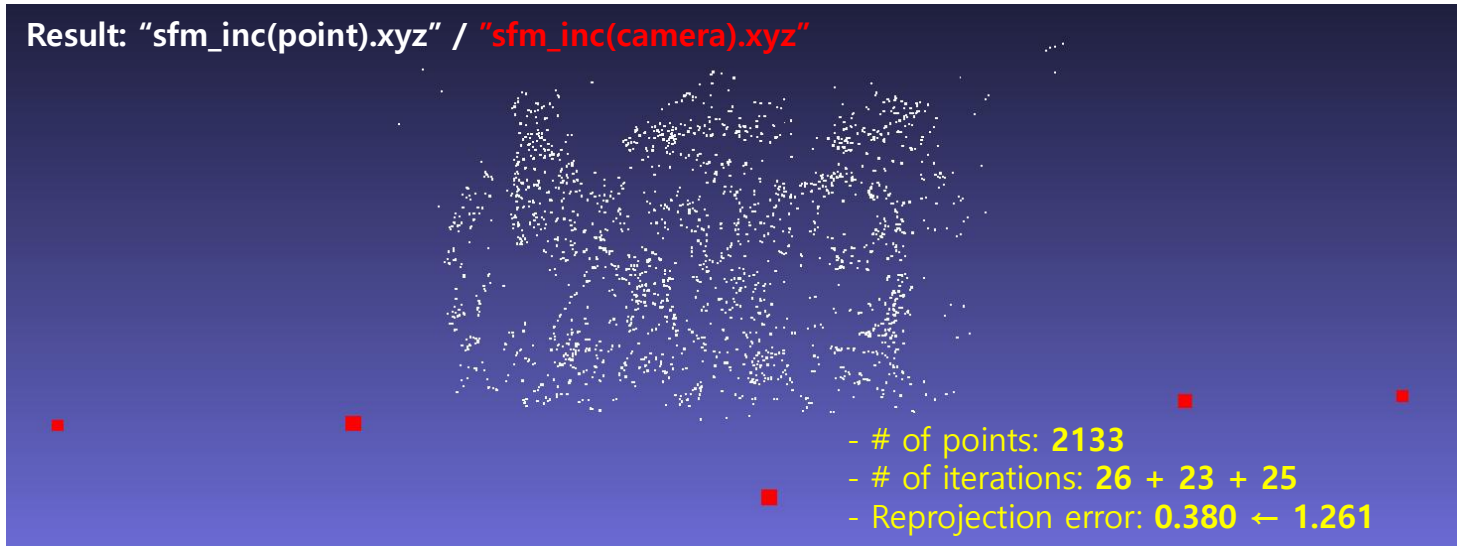
  6) Reconstruct newly observed (unknown) 3D points **(triangulation)**

  7) Optimize camera poses and 3D points together **(BA)**

Until there is no image left

Structure-from-Motion





Summary

- **Bundle Adjustment**
  - Problem)
  - Optimization of reprojection error
- **Structure-from-Motion**
  - Problem)
  - Two approaches: Global vs. Incremental
  - COLMAP (2016): The state-of-the-art incremental SfM and more

COLMAP (2016)

- **The state-of-the-art incremental SfM and more**
  - The author won the PAMI Mark Everingham Prize (2020) for COLMAP SfM and MVS.

- **References**
  - Code repositories
    - COLMAP (BSD license): Homepage, Documentation, Github
    - Note) COLMAP binary files are also available for rapid deployment.
  - Papers
    - Schonberger et al., *Structure-from-Motion Revisited*, CVPR, 2016 PDF DOI
    - Schonberger et al., *Pixelwise View Selection for Unstructured Multi-view Stereo*, ECCV, 2016 PDF DOI
    - Schonberger et al., *A Vote-and-Verify Strategy for Fast Spatial Verification in Image Retrieval*, ACCV, 2016 PDF DOI

COLMAP (2016)

- **The state-of-the-art incremental SfM and more**
  - The best performance in the view of **accuracy** and **time/space complexity** [Bianco et al., 2018]

Table 3. SfM camera pose evaluation results. N_c is the percentage of used cameras. $\bar{x}$ is the mean distance from ground truth of reconstructed camera positions and s_x its standard deviation. $\bar{r}$ is the mean rotation difference from ground truth of reconstructed camera orientations and s_r its standard deviation. n.a. means measure not available.

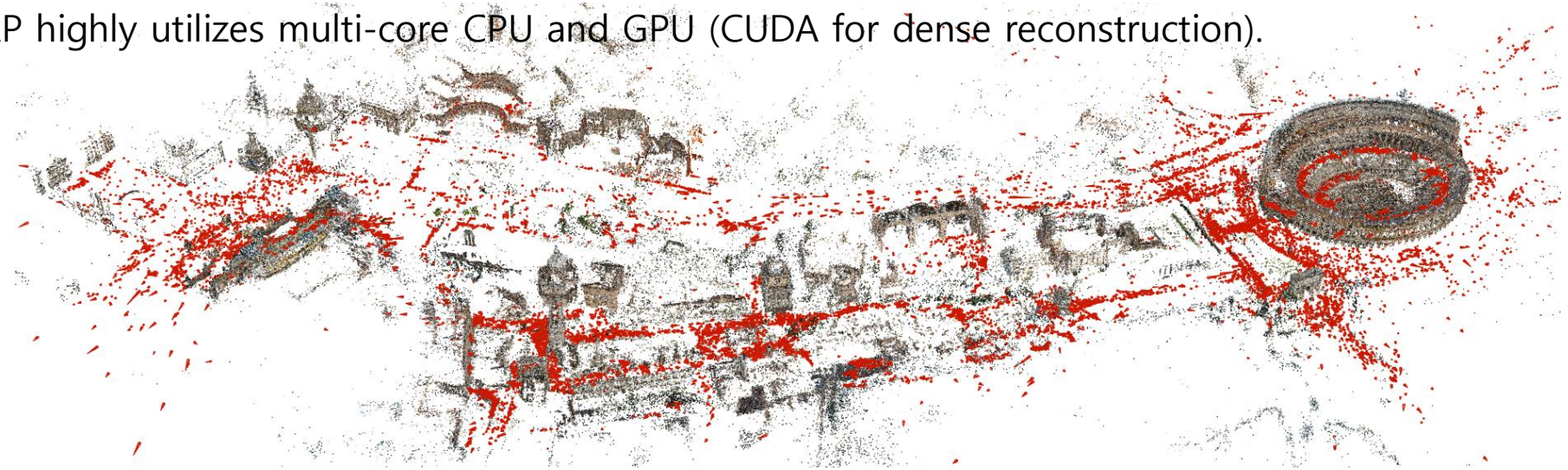
Model	COLMAP					OpenMVG					Theia					VisualSFM				
	N_c	$\bar{x}$ [m]	s_x [m]	$\bar{r}$ [°]	s_r [m]	N_c	$\bar{x}$ [m]	s_x [m]	$\bar{r}$ [°]	s_r [m]	N_c	$\bar{x}$ [m]	s_x [m]	$\bar{r}$ [°]	s_r [m]	N_c	$\bar{x}$ [m]	s_x [m]	$\bar{r}$ [°]	s_r [m]
Statue	100	0.08	0.01	0.04	0.05	100	0.27	0.03	0.47	0.22	100	1.86	0.09	0.45	0.22	100	1.45	0.91	3.55	2.88
E. Vase	100	0.01	0.01	0.51	0.05	83	0.78	1.62	32.19	64.89	100	0.13	0.07	0.91	0.35	94	0.15	0.14	4.91	5.07
Bicycle	88	0.04	0.02	0.25	0.19	94	0.60	1.09	7.00	12.53	37	0.60	0.03	1.10	0.31	47	0.27	0.14	1.32	1.03
Hydrant	82	2.63	2.09	72.28	64.98	3	–	–	–	–	6	3.49	0.29	174.27	0.51	80	2.43	1.76	66.29	58.90
Jeep	63	0.04	0.02	0.26	0.11	92	0.24	1.32	4.80	26.33	95	0.43	0.42	1.33	5.67	83	1.02	2.79	9.68	22.84
Ignatius	100	n.a.	n.a.	n.a.	n.a.	100	n.a.	n.a.	n.a.	n.a.	100	n.a.	n.a.	n.a.	n.a.	100	n.a.	n.a.	n.a.	n.a.

Table 5. Pipelines execution times in seconds and memory usage in MB.

Model	COLMAP				OpenMVG				Theia				VisualSFM			
	SfM		MVS		SfM		MVS		SfM		MVS		SfM		MVS	
	t [s]	*RAM*	t [s]	*RAM*	t [s]	*RAM*	t [s]	*RAM*	t [s]	*RAM*	t [s]	*RAM*	t [s]	*RAM*	t [s]	*RAM*
Statue	59	897	86	1062	43	1359	115	1300	196	1984	98	1249	86	1406	144	1452
Empire Vase	53	897	154	2101	28	628	130	1734	129	1988	134	1926	62	1226	159	2095
Bicycle	98	896	117	1356	57	1467	146	1720	63	1722	58	548	64	1226	68	641
Hydrant	19	894	55	793	16	1547	–	–	17	2048	3	1249	36	997	56	1452
Jeep	38	897	121	1812	69	1550	275	3209	213	2083	280	3293	109	1406	254	3078
Ignatius	1225	1825	430	5082	401	1555	494	5926	992	2588	484	5626	1639	2381	345	4742

COLMAP (2016)

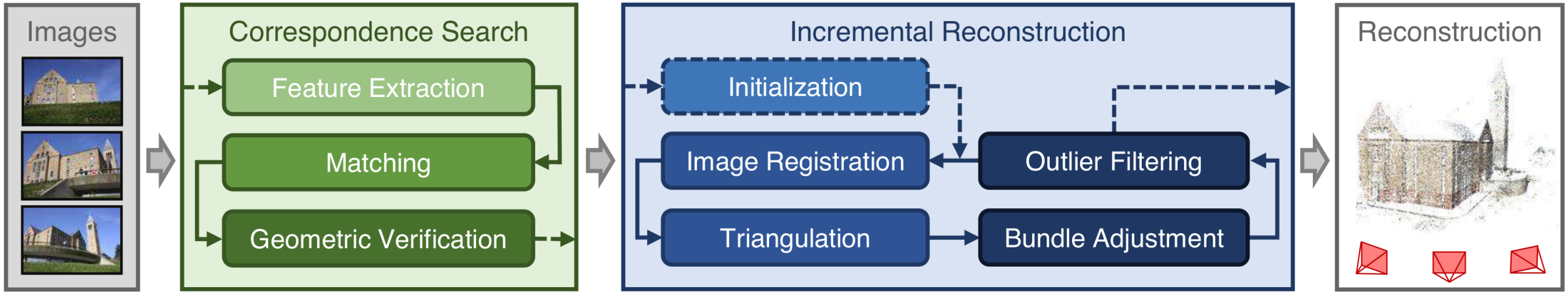
- **The state-of-the-art incremental SfM and <u>more</u>**
  - COLMAP includes not only **sparse reconstruction**, but also **dense reconstruction** and **mesh reconstruction**.
  - COLMAP highly utilizes multi-core CPU and GPU (CUDA for dense reconstruction).



A sparse model of *central Rome* using 21K photos produced by COLMAP's SfM pipeline

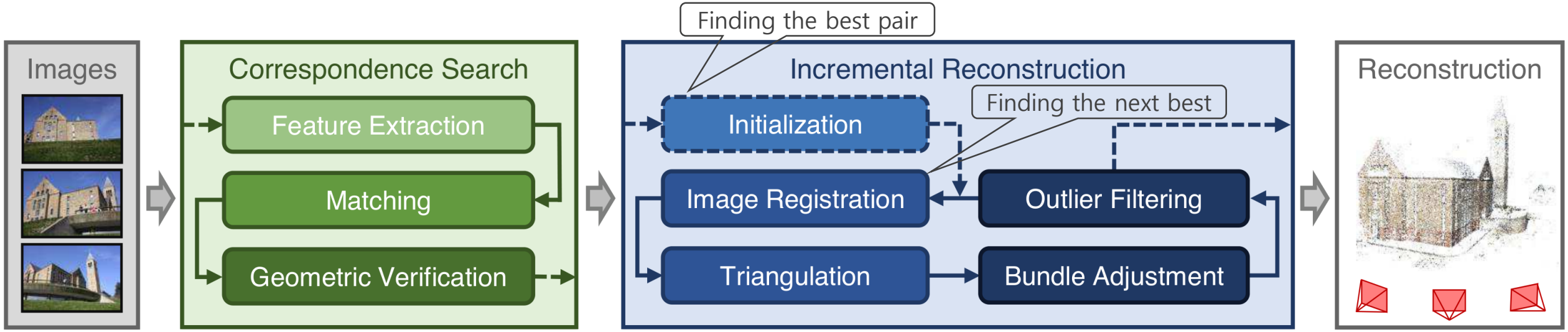
Dense models of several landmarks produced by COLMAP's MVS pipeline

Review) Incremental Structure-from-Motion



- **Correspondence Search**
  - **Feature extraction**
    - The features should be invariant under radiometric and geometric changes.
    - **SIFT** and its derivatives, learned features vs. binary features (better efficiency with reduced robustness)
  - **Matching** → Feature correspondence
    - Exhaustive search (time complexity: $O(N_I^2\, N_F^2)$) vs. more efficient search (for scalability)
  - **Geometric verification** → Scene graph (node: images, edge: feature/geometric relationship)
    - Geometric relations: H (for pure rotation and a planar scene) vs. F and E (for moving camera in a general scene)
    - Issues) Outliers: **RANSAC** / Model selection: **GRIC**, QDEGSAC

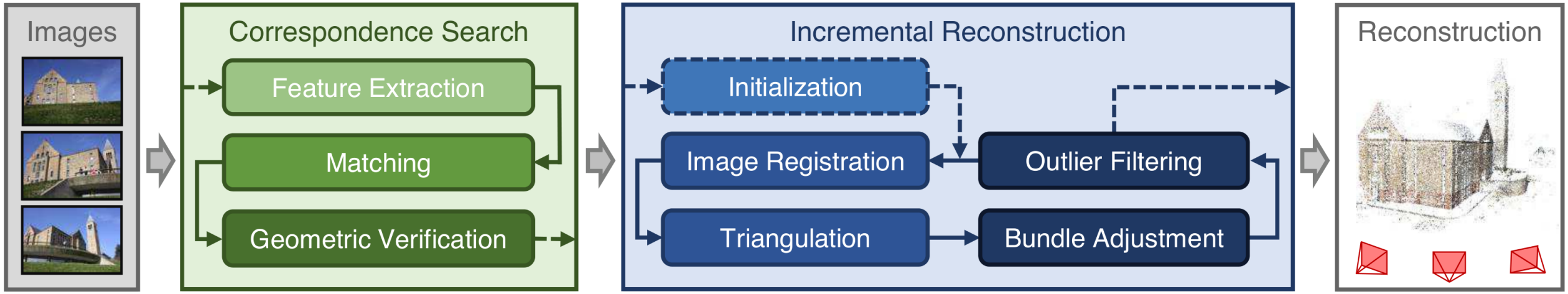
Review) Incremental Structure-from-Motion



- **Incremental Reconstruction**
  - **Initialization**: Two-view reconstruction from the best pair of images [critical]
  - **Image Registration**: PnP (+ intrinsic parameters) with 2D-3D correspondences and RANSAC
  - **Triangulation**: Adding more 2D-3D correspondences
  - **Bundle Adjustment (BA)**
    - Motivation) SfM usually drifts quickly without additional refinement to image registration and triangulation.
    - The joint non-linear ~~minimization~~ refinement of camera parameters $\mathbf{P}_c$ and 3D points $\mathbf{X}_k$

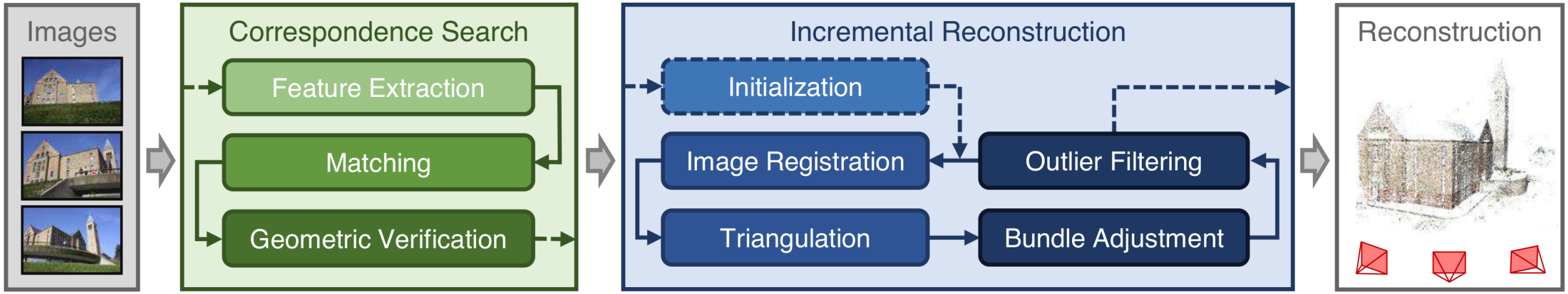
      $$E = \sum_j \rho_j \left(\left\| \pi(\mathbf{X}_k; \mathbf{P}_c) - \mathbf{x}_j \right\|_2^2 \right)$$ with a loss function ρ_j (against outliers)
    - Choices) Solvers: LM, …, exact vs. inexact methods / Error functions: Direct vs. sparse direct vs. **indirect**

Review) Incremental Structure-from-Motion



- **Problems**
  - SfM systems fail to register a large fraction of images.
  - SfM systems produce broken models due to mis-registrations or drift.
- **Why?**
  - Incomplete scene graph @ Correspondence Search
  - Missing or inaccurate scene structure @ Image Registration and Triangulation
- **Challenges**
  - Improving completeness, robustness, and accuracy while boosting efficiency

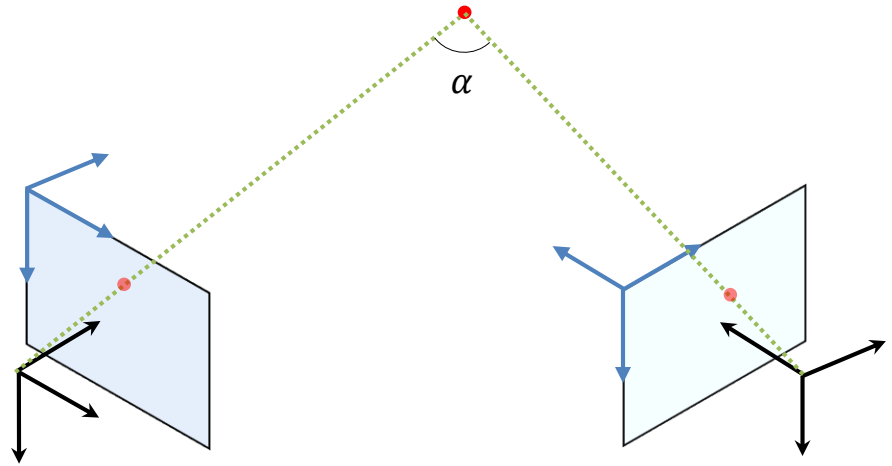
COLMAP (2016): Contributions



- Multi-model scene graph generation
- Next best view selection with multi-resolution scoring
- Multi-view triangulation using recursive RANSAC
- Iterative BA, re-triangulation, and outlier filtering
- Redundant view grouping

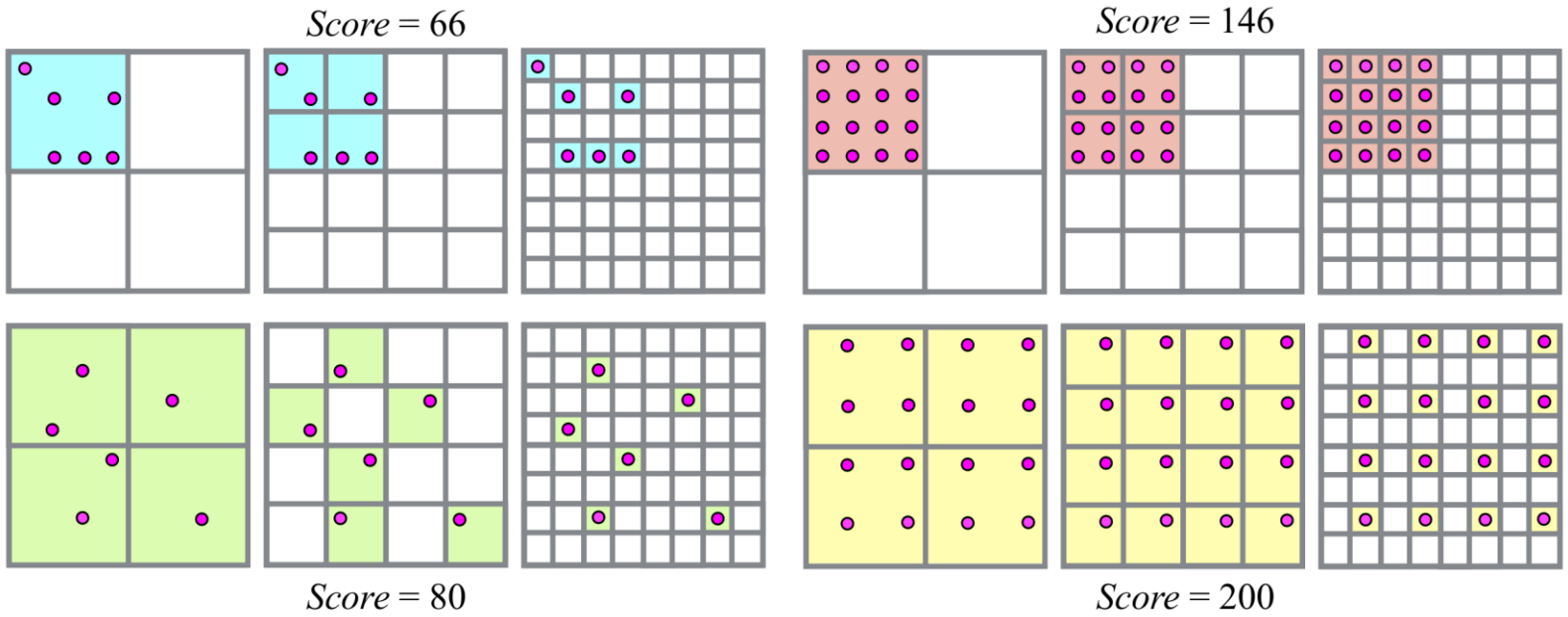
COLMAP (2016): Contributions

- **Multi-model scene graph generation**
  - Motivation) Each model has its degenerate (undefined) situations.
    - Models: F, E, and H
  - **Multi-model geometric verification**
    - Note) N_M: the number of inliers for model M, α: triangulation angle
    - $N_H/N_F < \epsilon_{HF}$ → Moving camera in a general scene
    - $N_E/N_F > \epsilon_{EF}$ → Correct calibration
    - $\text{median}(\alpha_0, \alpha_1, \dots)$ → Pure rotation (panoramic) or planar scenes (planar)
    - Model types (for edges): General, panoramic, planar | uncalibrated, calibrated
      - The first best pair is selected among *non-panoramic* and (preferably) *calibrated* image pairs.
      - Triangulation is not applied to *panoramic* image pairs.
  - **Watermarks, timestamps, and frames (WTFs) detection**
    - Note) S: A similarity transformation at the image border
    - $N_S/N_F > \epsilon_{SF}$ and $N_S/N_E > \epsilon_{SE}$ → WTFs → The pair is not used



COLMAP (2016): Contributions

- **Next best view selection with multi-resolution scoring**
  - Motivation) More visible and more uniformly distributed points get higher score.
    - Q) Why are uniformed distributed points preferred? (a.k.a. bucketing)
  - **Multi-resolution scoring**
    - Cell size: K_l bins for both dimension (Note: $K_l = 2^l$)
    - Cell states: *Empty* and *full*
    - Score: $S = \sum_{l=1}^{L} K_l N_l$
      - Note) N_l: The number of full cells at K_l resolution



COLMAP (2016): Contributions

- **Multi-view triangulation using recursive RANSAC**
  - Motivation
    - Two-view (small baseline) → Multi-view (larger baseline)
    - Outliers (from other feature tracks) → (recursive) RANSAC

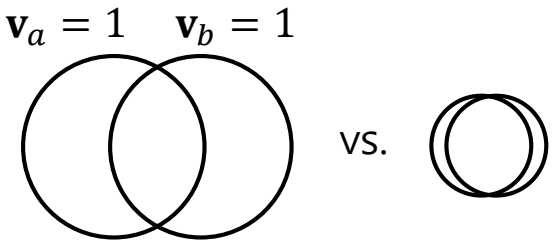
  - **Multi-view triangulation using recursive RANSAC**
    - Given data: Normalized 2D points for a single feature track
      - Note) $\bar{\mathbf{x}}_i$ on the i-th normalized image (or camera or view)
    - Model: Their 3D point $\mathbf{X}$
    - Model estimation: Two-view triangulation using DLT
    - Error measure: Reprojection error between $\bar{\mathbf{x}}_i$ and $\mathbf{X}$
    - RANSAC objective: Maximizing the number of supporting 2D points (~ camera or view)
      - Constraints #1) A sufficient triangulation angle α
      - Constraints #2) The positive depth constraint (a.k.a. the cheirality constraint)
    - Note) RANSAC is performed recursively after separating the consensus set from the remaining data.
    - Note) RANSAC is terminated adaptively using $\frac{\log(1-\eta)}{\log(1-\epsilon^2)}$ where η is success rate and ε is the inlier ratio.

COLMAP (2016): Contributions

- **Iterative BA, re-triangulation, and outlier filtering**
  - **Bundle adjustment (BA)**
    - A simple camera model: 1 x focal length and 1 x radial distortion parameter
      - The fixed principal point at the image center (← Its estimation is ill-posed problem.)
    - Optimization: PCG (preconditioned conjugate gradient) in Ceres Solver
      - Loss function: Cauchy function
    - Note) Local BA after every image registration
    - Note) Global BA after growing the model by a certain percentage (for an amortized linear runtime of SfM)
  - **Re-triangulation (RT)**
    - Pre-BA RT step (for reducing drift effect similar to *VisualSFM*)
    - Post-BA RT step (for recovering failed triangulation due to inaccurate pose)
  - **Filtering**
    - Filtering observations with large reprojection error
    - Filtering cameras with an abnormal field-of-view or a large distortion coefficient (← panoramas, artificial images)

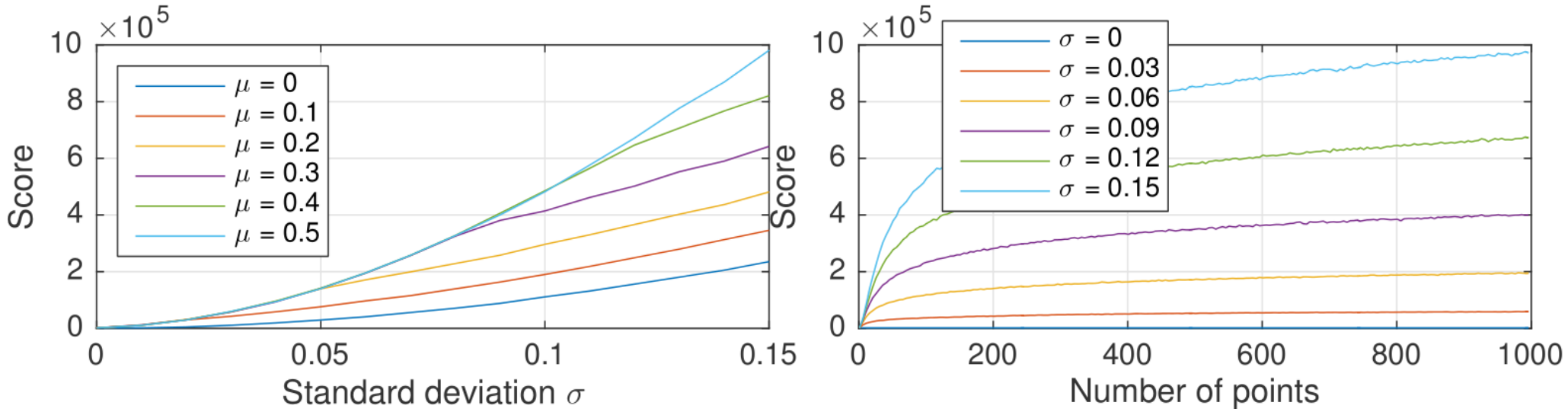
COLMAP (2016): Contributions

- Motivation
  - BA is a major ~~performance~~ computing time bottleneck in SfM.
  - Internet photo collections usually have a highly non-uniform visibility pattern due to POI (points of interest).
- **Redundant view grouping**
  - View grouping using visibility overlap
    - Visibility overlap $V_{ab} = \|\mathbf{v}_a \wedge \mathbf{v}_b\| / \|\mathbf{v}_a \vee \mathbf{v}_b\|$
    - Many small and highly overlapping groups (~ submaps)
    - Note) More efficient than graph-cut (non-overlapping submaps)
  - BA for grouped images (with group-local coordinate frame)
    - $E_g = \sum_j \rho_j \left(\left\| \pi_g(\mathbf{X}_k; \mathbf{G}_r, \mathbf{P_c}) - \mathbf{x}_j \right\|_2^2 \right)$ with fixed $\mathbf{P}_c$
      - $\mathbf{G}_r$: The extrinsic group parameters ($\in \mathbf{SE}(3)$)
      - $\mathbf{P}_{cr} = \mathbf{P}_c \, \mathbf{G}_r$: The projection matrix in the group r
    - Note) For standard BA, $E = \sum_j \rho_j \left(\left\| \pi(\mathbf{X}_k; \mathbf{P}_c) - \mathbf{x}_j \right\|_2^2 \right)$



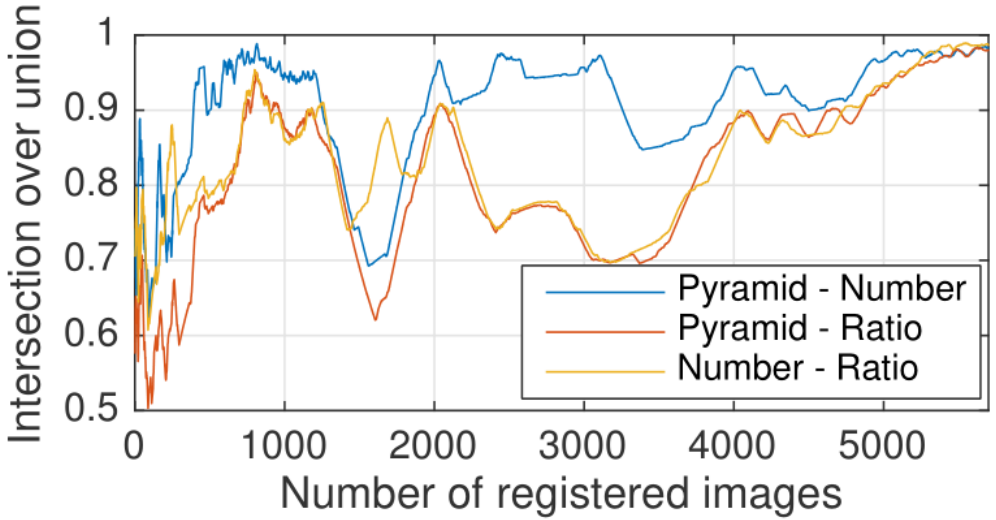
COLMAP (2016): Experiments

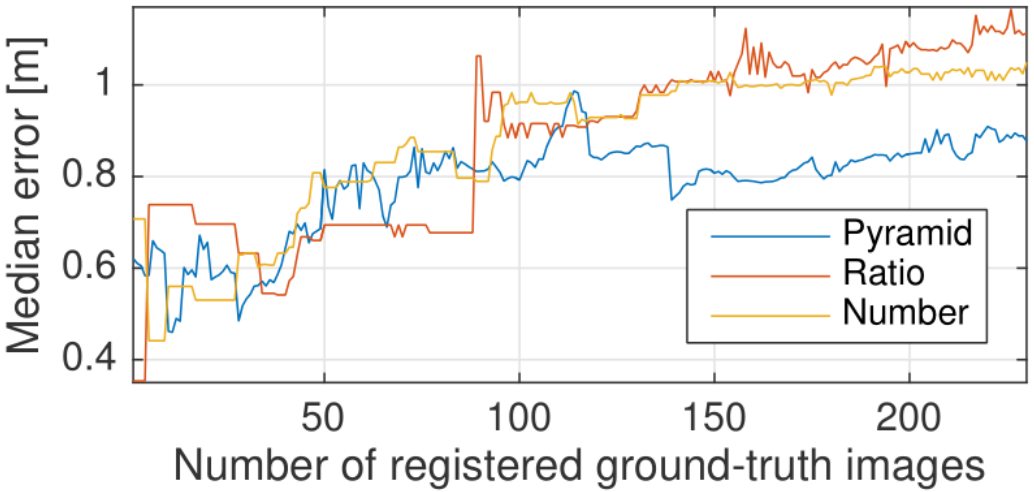
- **Next best view selection with multi-resolution scoring**
  - Data: Synthetic data
    - Pyramid level $L = 6$
    - Gaussian-distributed image points with spread σ and location μ
  - Results
    - A larger σ and a more central μ → A more uniform distribution → A higher score
    - The score was regardless of the number of points.
      - However, the score was affected when the number of points is small.



COLMAP (2016): Experiments

- **Next best view selection with multi-resolution scoring**
  - Data: *Quad* dataset with ground-truth camera positions
  - Comparison
    - Pyramid: The proposed method
    - Ratio: Maximizes the ratio of visible over potentially visible points
    - Number: Maximizes the number of triangulated points (used in *Bundler*)
  - Results
    - Pyramid produced the most accurate reconstruction by choosing a better registration order for the images.
    - All strategies converged to the same set of registered images.





COLMAP (2016): Experiments

- **Multi-view triangulation using recursive RANSAC**
  - Data: *Dubrovnik* dataset
    - 2.9M feature tracks from 47M matches
  - Comparison: *Bundler*, exhaustive strategy (limit: 10K iterations ~ η: ~~0.999~~ 0.99, ϵ: 0.02), recursive RANSAC
  - Results
    - The recursive approaches recovered more feature tracks and longer feature tracks.
    - The RANSAC-based approaches ($\eta_1 = 0.99$, $\eta_2 = 0.5$) were faster 10-40x than the exhaustive strategy.

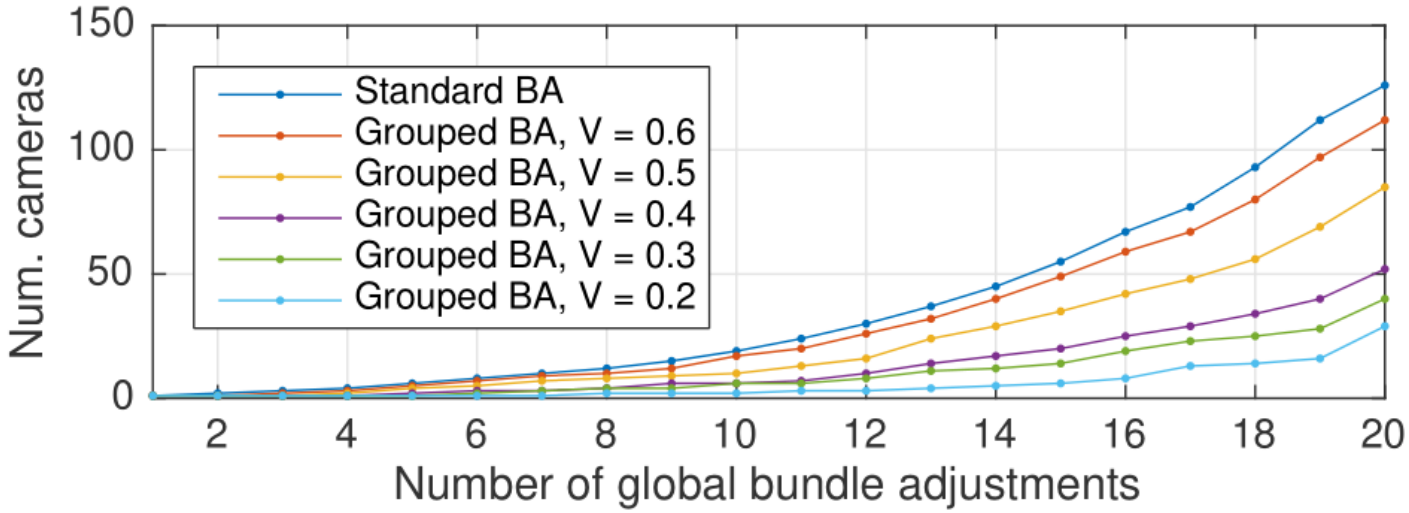
	#Points	#Elements	Avg. Track Length	#Samples
Bundler	713,824	5.58 M	7.824	136.34 M
Non-recursive				
Exhaustive	861,591	7.64 M	8.877	120.44 M
RANSAC, η_1	861,591	7.54 M	8.760	3.89 M
RANSAC, η_2	860,422	7.46 M	8.682	3.03 M
Recursive				
Exhaustive	894,294	8.05 M	9.003	145.22 M
RANSAC, η_1	902,844	8.03 M	8.888	12.69 M
RANSAC, η_2	906,501	7.98 M	8.795	7.82 M

COLMAP (2016): Experiments

- **Redundant view grouping**
  - Data: An unordered collection of densely distributed images
  - Comparison: Standard BA, grouped BA (V: Scene overlap)
  - Results

	S. BA	G. BA ($V = 0.6$)	G. BA ($V = 0.3$)	G. BA ($V = 0.1$)
The effective speed-up	-	5%	14%	32%
The average reprojection error	0.26px	0.27px	0.28px	0.29px

- The reconstruction quality is comparable for all choices of $V > 0.3$ and increasingly degrades for a smaller V.
- The total runtime for *Colosseum* with $V = 0.4$ reduces by 36% with an equivalent reconstruction.



COLMAP (2016): Experiments

- **Overall**
  - Comparison: Global SfM - *Theia* (a.k.a. DISCO) / Incremental SfM - Bundler, VisualSFM, COLMAP
  - Results
    - Computing time: Theia > VisualSFM > COLMAP >> Bundler
    - Completeness (# of registered images, # of points, and avg. track length): COLMAP > …
    - Accuracy (@ *Quad* dataset): COLMAP 0.85m > VisualSFM 0.89m > Bundler 1.01m > Theia 1.16m

	# Images	# Registered				# Points (Avg. Track Length)				Time [s]				Avg. Reproj. Error [px]			
		Theia	*Bundler*	*VSFM*	*Ours*	*Theia*	*Bundler*	*VSFM*	*Ours*	*Theia*	*Bundler*	*VSFM*	*Ours*	*Theia*	*Bundler*	*VSFM*	*Ours*
Rome [14]	74,394	–	13,455	14,797	20,918	–	5.4M	12.9M	5.3M	–	295,200	6,012	10,912	–	–	–	–
Quad [14]	6,514	–	5,028	5,624	5,860	–	10.5M	0.8M	1.2M	–	223,200	2,124	3,791	–	–	–	–
Dubrovnik [36]	6,044	–	–	–	5,913	–	–	–	1.35M	–	–	–	3,821	–	–	–	–
Alamo [61]	2,915	582	647	609	666	146K (6.0)	127K (4.5)	124K (8.9)	94K (11.6)	874	22,025	495	882	1.47	2.29	0.70	0.68
Ellis Island [61]	2,587	231	286	297	315	29K (4.9)	39K (4.1)	61K (5.5)	64K (6.8)	94	12,798	240	332	2.41	2.24	0.71	0.70
Gendarmenmarkt [61]	1,463	703	302	807	861	87K (3.8)	93K (3.7)	138K (4.9)	123K (6.1)	202	465,213	412	627	2.19	1.59	0.71	0.68
Madrid Metropolis [61]	1,344	351	330	309	368	47K (5.0)	27K (3.2)	48K (5.2)	43K (6.6)	95	21,633	203	251	1.48	1.62	0.59	0.60
Montreal Notre Dame [61]	2,298	464	501	491	506	154K (5.4)	135K (4.6)	110K (7.1)	98K (8.7)	207	112,171	418	723	2.01	1.92	0.88	0.81
NYC Library [61]	2,550	339	400	411	453	66K (4.1)	71K (3.7)	95K (5.5)	77K (7.1)	194	36,462	327	420	1.89	1.84	0.67	0.69
Piazza del Popolo [61]	2,251	335	376	403	437	36K (5.2)	34K (3.7)	50K (7.2)	47K (8.8)	89	33,805	275	380	2.11	1.76	0.76	0.72
Piccadilly [61]	7,351	2,270	1,087	2,161	2,336	197K (4.9)	197K (3.9)	245K (6.9)	260K (7.9)	1,427	478,956	1,236	1,961	2.33	1.79	0.79	0.75
Roman Forum [61]	2,364	1,074	885	1,320	1,409	261K (4.9)	281K (4.4)	278K (5.7)	222K (7.8)	1,302	587,451	748	1,041	2.07	1.66	0.69	0.70
Tower of London [61]	1,576	468	569	547	578	140K (5.2)	151K (4.8)	143K (5.7)	109K (7.4)	201	184,905	497	678	1.86	1.54	0.59	0.61
Trafalgar [61]	15,685	5,067	1,257	5,087	5,211	381K (4.8)	196K (3.7)	497K (8.7)	450K (10.1)	1,494	612,452	3,921	5,122	2.09	2.07	0.79	0.74
Union Square [61]	5,961	720	649	658	763	35K (5.3)	48K (3.7)	43K (7.1)	53K (8.2)	131	56,317	556	693	2.36	3.22	0.68	0.67
Vienna Cathedral [61]	6,288	858	853	890	933	259K (4.9)	276K (4.6)	231K (7.6)	190K (9.8)	764	567,213	899	1,244	2.45	1.69	0.80	0.74
Yorkminster [61]	3,368	429	379	427	456	143K (4.5)	71K (3.9)	130K (5.2)	105K (6.8)	164	34,641	661	997	2.38	2.61	0.72	0.70

COLMAP (2016): Conclusion

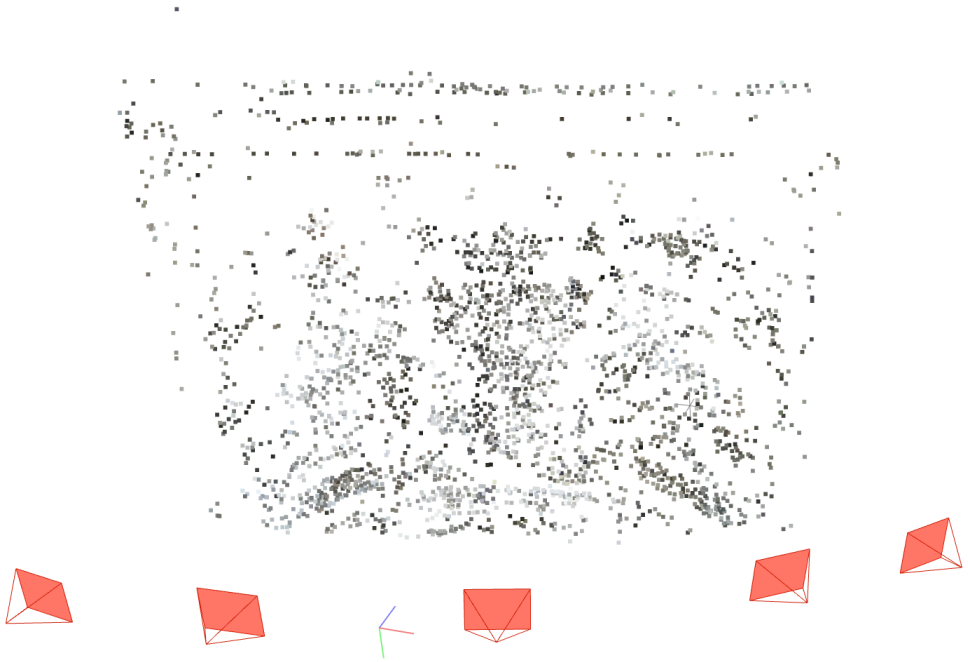
- **The state-of-the-art ~~incremental~~ SfM**
  - In terms of completeness, robustness, accuracy, and efficiency (~ computing time)
  - Comprehensive experiments and validation with large-scale datasets (individual components and overall system)
  - An open-source with BSD license

- **My opinions**
  - COLMAP works really well.
  - SLAM and SfM is not a (small) module, but a kind of (large) systems.
  - This is a kind of implementation papers (including many small tips).

COLMAP (2016): Getting Started with COLMAP GUI

- Example) COLMAP with the relief dataset

Sparse Reconstruction (SfM)



# of points: 2,889

Dense Reconstruction (MVS)

# of points: 336,223

Mesh Reconstruction

Viewer: CloudCompare

# of vertices: 102,694
# of faces: 205,633

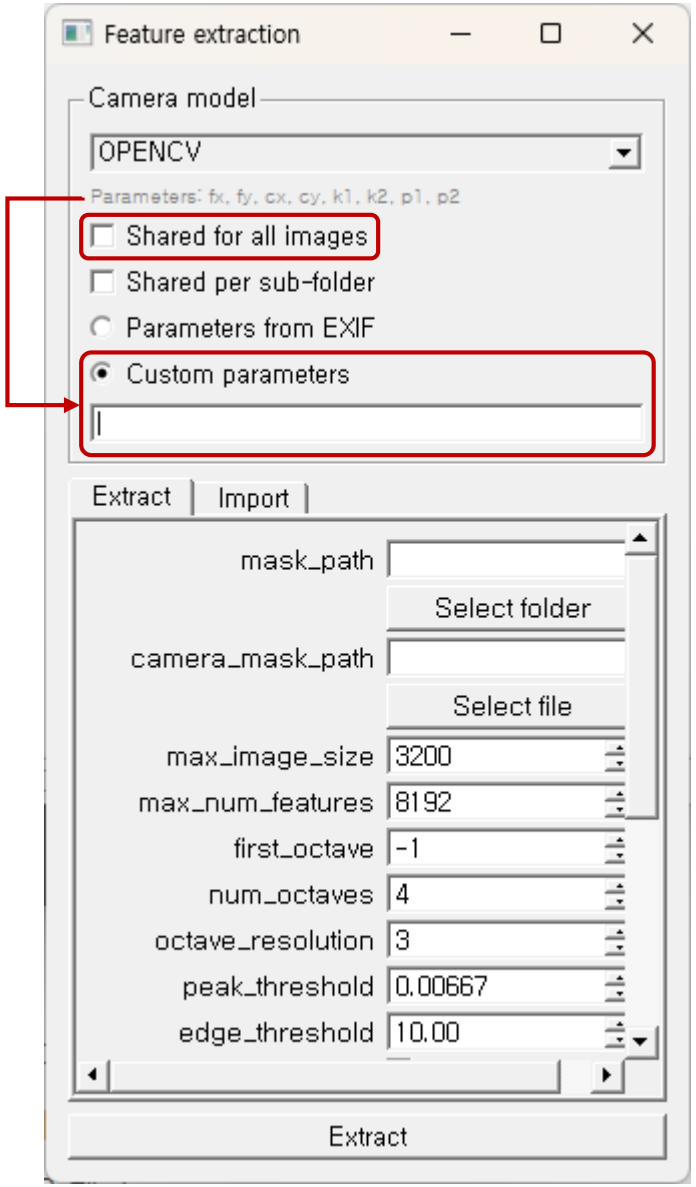
COLMAP (2016): Getting Started with COLMAP GUI

- **Download COLMAP Binaries**
  - Download the pre-release version available on the official website.
    - Note) Please download the version labeled with `-cuda` if you want to use an NVIDIA GPU for dense reconstruction.
  - For more efficient feature matching, download the vocabulary trees available on the COLMAP website.
- **Running COLMAP GUI @ Windows**
  - Execute `COLMAP.bat` file
  - Note) Please refer the official document for detailed mouse/keyboard usages.
- **COLMAP SfM/MVS Step-by-Step**
  1. Create a Project and Load Input Images
  2. Feature Extraction
  3. Feature Matching
  4. Sparse Reconstruction (SfM)
  5. Dense Reconstruction (MVS)
  6. Mesh Reconstruction

COLMAP (2016): Getting Started with COLMAP GUI

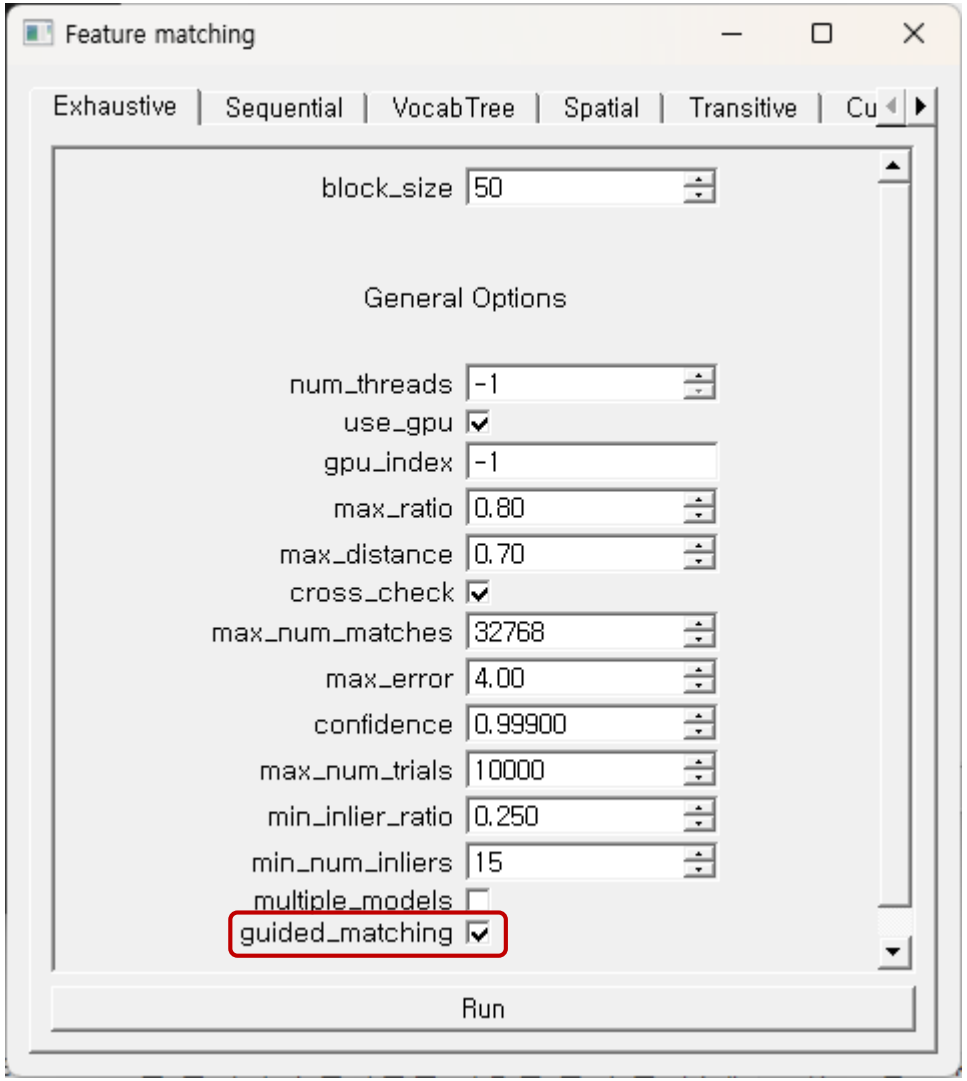
- **COLMAP SfM/MVS Step-by-Step (Cont'd)**

1. **Create a Project and Load Input Images**: Menu > `File` > `New project`
   - **For Database**: Click `New` to set the name of the database where the SfM results will be stored.
     - e.g. `C:\your_workspace\database.db`
   - **For Image**: Click `Select` to set the directory containing the images for SfM.
     - e.g. `C:\your_workspace\images`
   - Click the `Save` button to save the settings.
2. **Feature Extraction**: Menu > `Processing` > `Feature extraction`
   - Tip) If all images have the same camera parameters, check the `Shared for all images` option.
   - Tip) If you already know the internal parameters of the camera, select the `Custom parameters` option and fill the values (the order of values is shown in gray under the selected `Camera model`).



COLMAP (2016): Getting Started with COLMAP GUI

- **COLMAP SfM/MVS Step-by-Step (Cont'd)**
  1. Create a Project and Load Input Images
  2. Feature Extraction
  3. **Feature Matching**: Menu > `Processing` > `Feature matching`
     - Various matching methods are available.
       - By default, use `Exhaustive` to match all combinations.
       - For ordered images like SLAM data, use `Sequential` for faster matching (vocabulary tree setup needed for loop closing).
     - Tip) If your matching is not successful or SfM does not converge, try to check the `guided_matching` option.

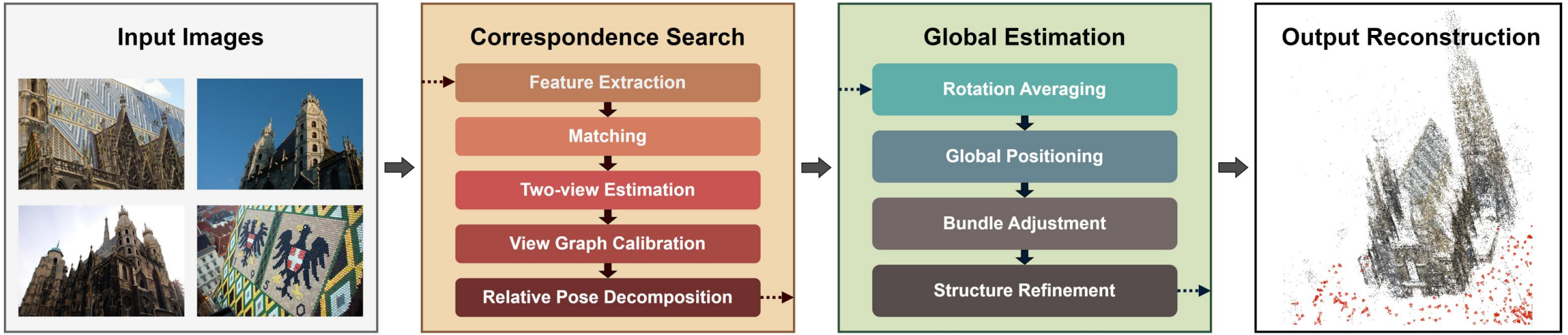


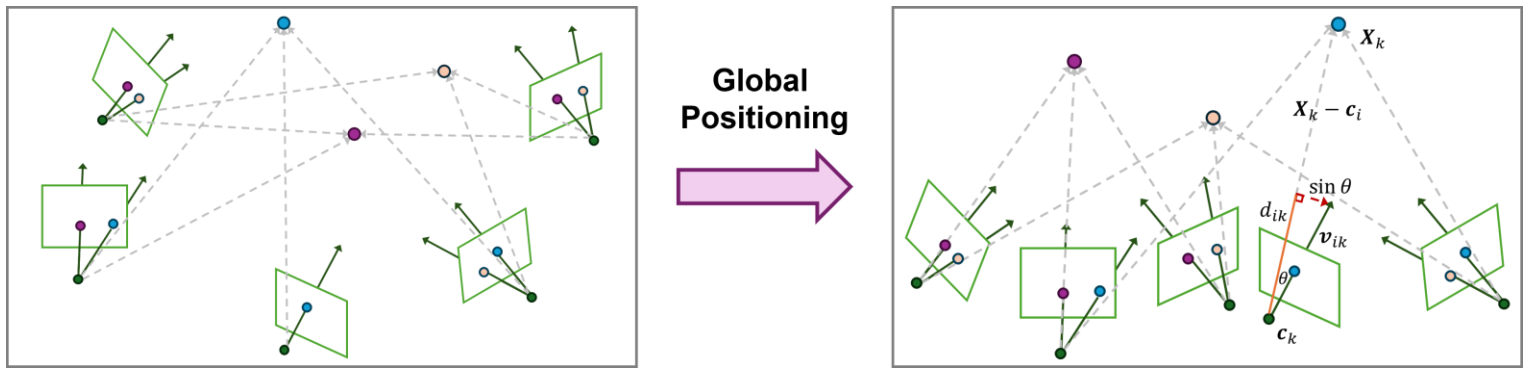
COLMAP (2016): Getting Started with COLMAP GUI

- **COLMAP SfM/MVS Step-by-Step (Cont'd)**
  1. Create a Project and Load Input Images
  2. Feature Extraction
  3. Feature Matching
  4. **Sparse Reconstruction (SfM)**: Menu > `Reconstruction` > `Start reconstruction`.
  5. **Dense Reconstruction (MVS)**: Menu > `Reconstruction` > `Dense reconstruction`.
     - Set the directory to save the results by clicking `Select`.
       - e.g. `C:\your_workspace\dense`
     - Follow the next steps, `Undistort` → `Stereo` → `Fusion`.
  6. **Mesh Reconstruction**
     - Follow the more steps, `Poisson` → `Delaunay`, after running the above dense reconstruction.

GLOMAP (2024)

- A COLMAP's alternative of **global SfM** with *global positioning*





Reference: Pan et al, "Global Structure-from-Motion Revisited", ECCV, 2024 Homepage Github