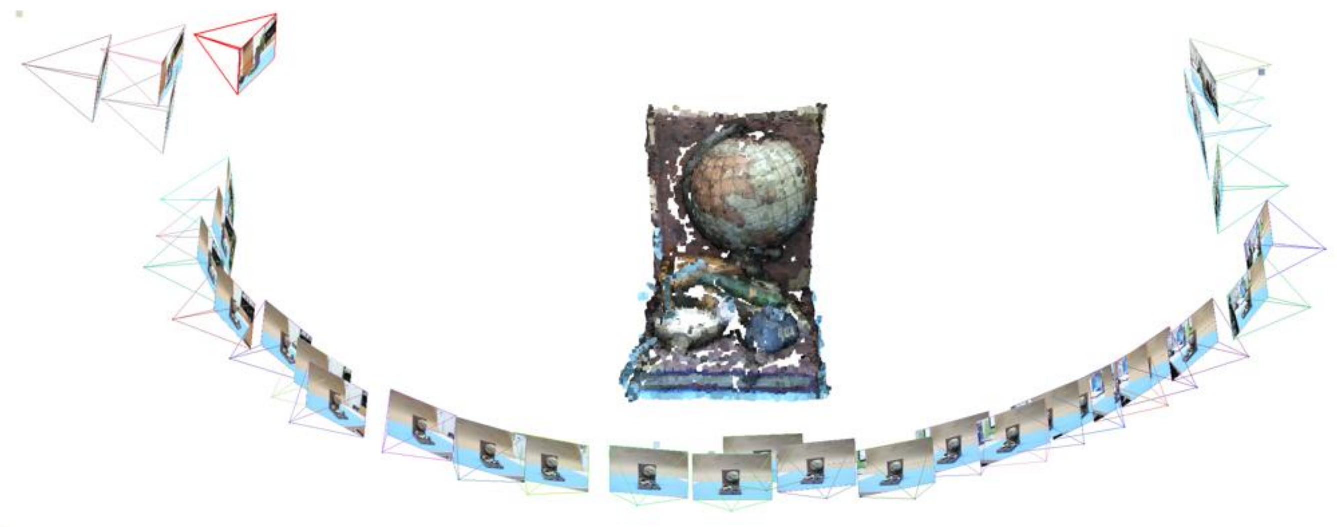
Programming Assignment #2: Structure from Motion

Computer Vision

What is SfM?

Structure from Motion

- Build a 3D and estimate camera poses, given the set of images.
- Construct **3D point cloud** from **multi-view images**



Applications

Gaussian splatting

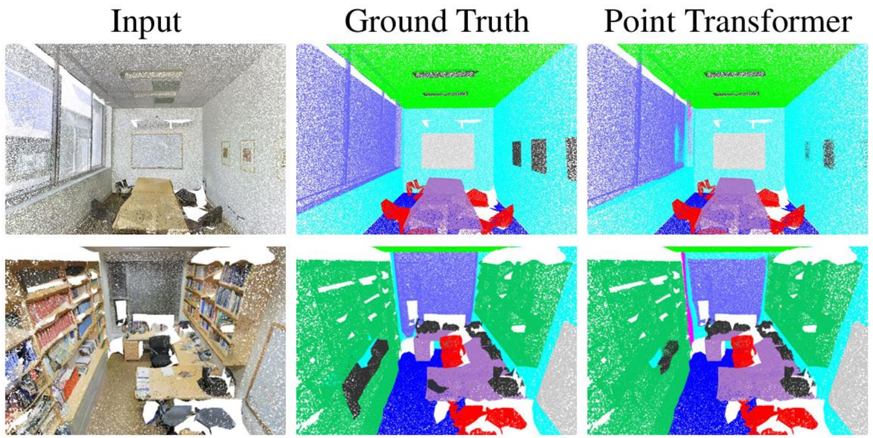
- Novel view synthesis from multi view images and 3D point cloud

- Bernhard et al. “3D Gaussian Splatting for Real-Time Radiance Field Rendering.” SIGGRAPH. (2023).

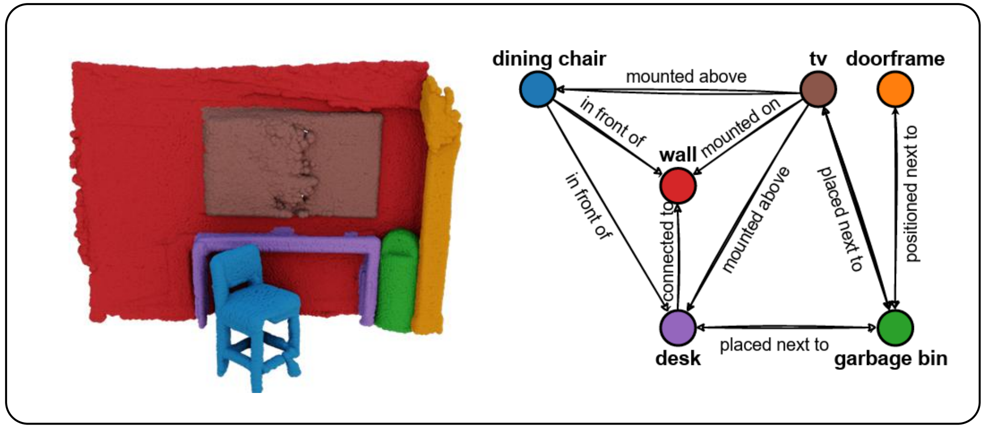
Applications

3D Scene Understanding

- Representations of semantic feature or objects from 3D representations as 3D point cloud.



3D Semantic Segmentation

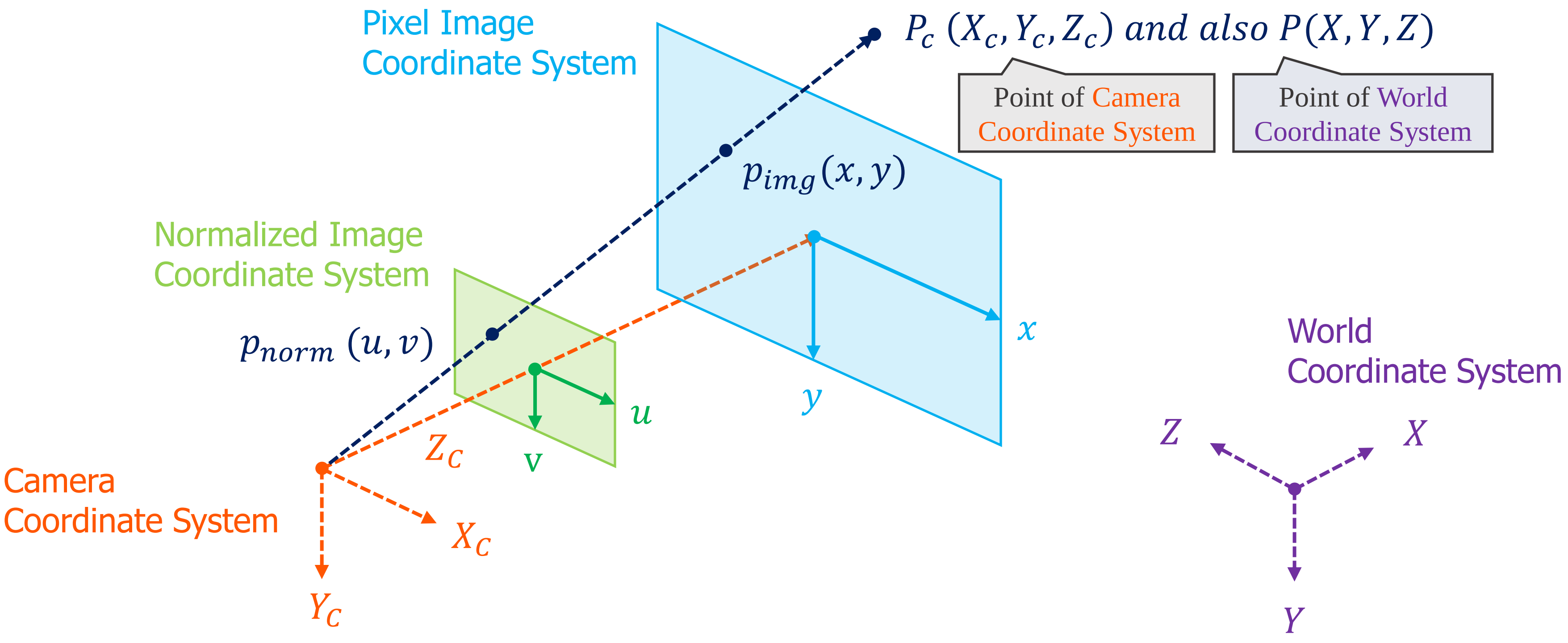


3D Scene Graph Generation

- Zhao, Hengshuang, et al. "Point transformer." ICCV. (2021).
- Koch, Sebastian, et al. "Open3dsg: Open-vocabulary 3d scene graphs from point clouds with queryable objects and open-set relationships." CVPR. (2024).

Preliminaries

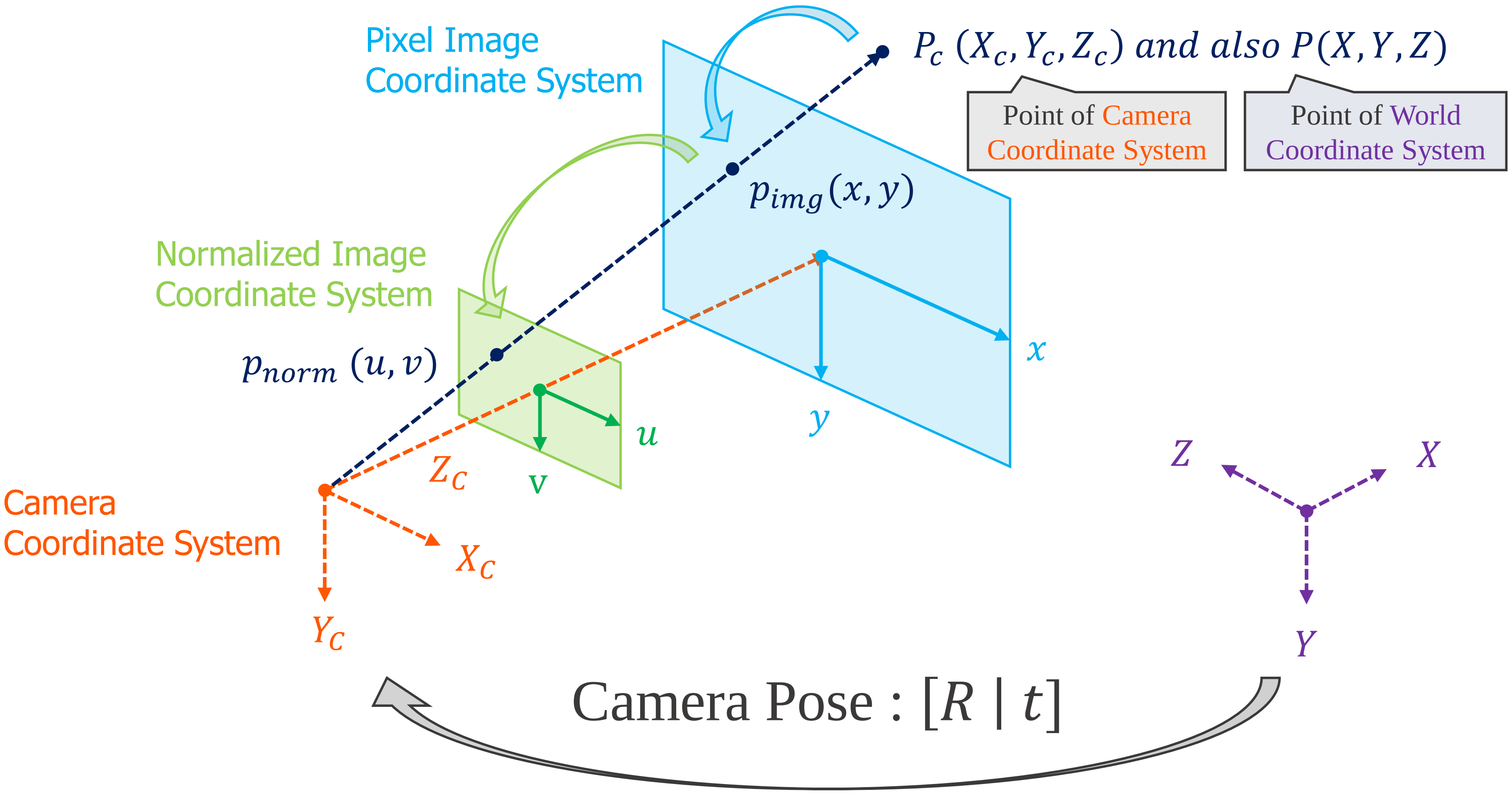
Camera coordinate system



Preliminaries

Camera coordinate system

$$p_{norm}(u,v) \leftarrow p_{img}(x,y) \leftarrow P_c(X_c,Y_c,Z_c) \leftarrow P(X,Y,Z)$$



Preliminaries

Camera coordinate system

$$p_{norm}(u,v) \leftarrow p_{img}(x,y) \leftarrow P_c(X_c,Y_c,Z_c) \leftarrow P(X,Y,Z)$$

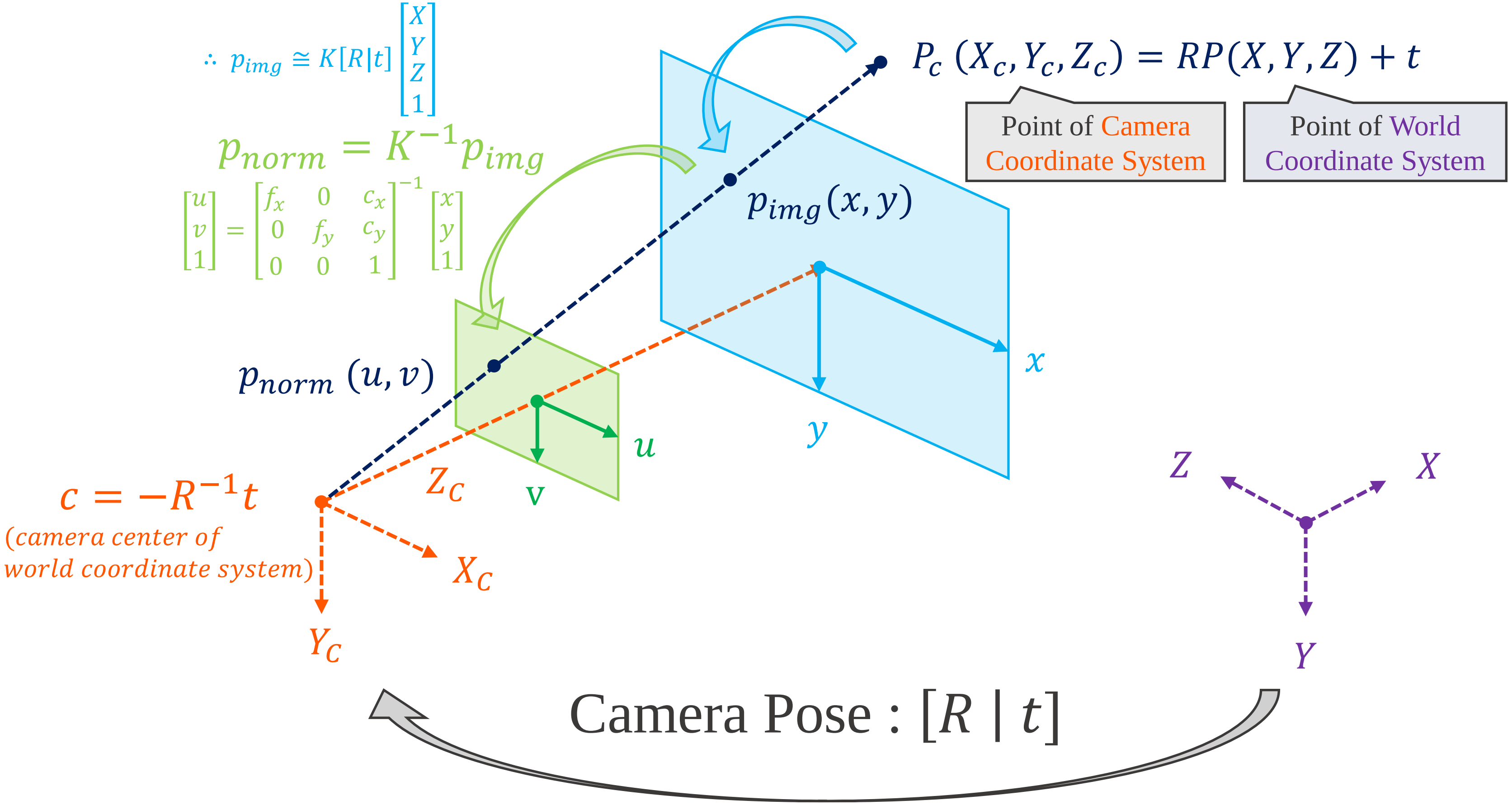
$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} (f_x X_c + c_x Z_c)/Z_c \\ (f_y Y_c + c_y Z_c)/Z_c \\ 1 \end{bmatrix} \sim \begin{bmatrix} f_x X_c + c_x Z_c \\ f_y Y_c + c_y Z_c \\ Z_c \end{bmatrix} = K \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \quad w.r.t\ K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$\therefore\ p_{img} \cong K[R|t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$p_{norm} = K^{-1} p_{img}$$

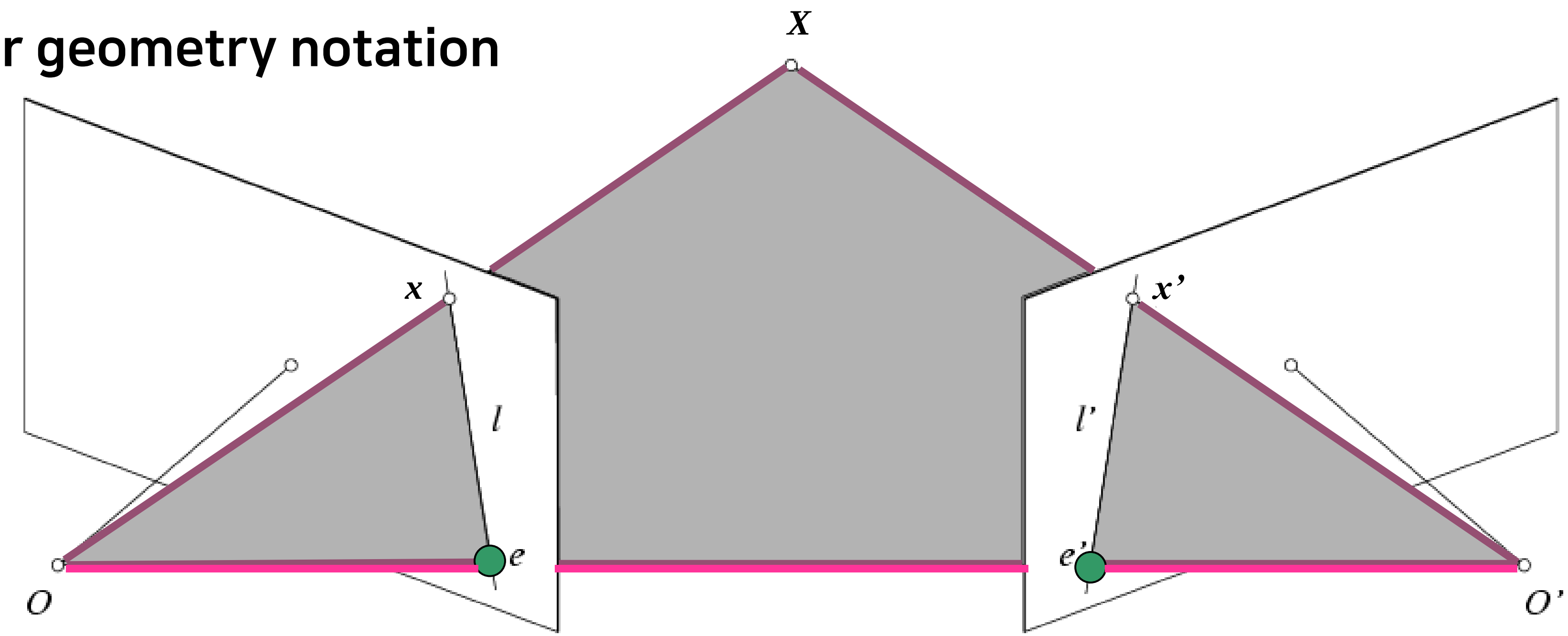
$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$P_c\ (X_c,Y_c,Z_c) = RP(X,Y,Z) + t$$



Preliminaries

Epipolar geometry notation



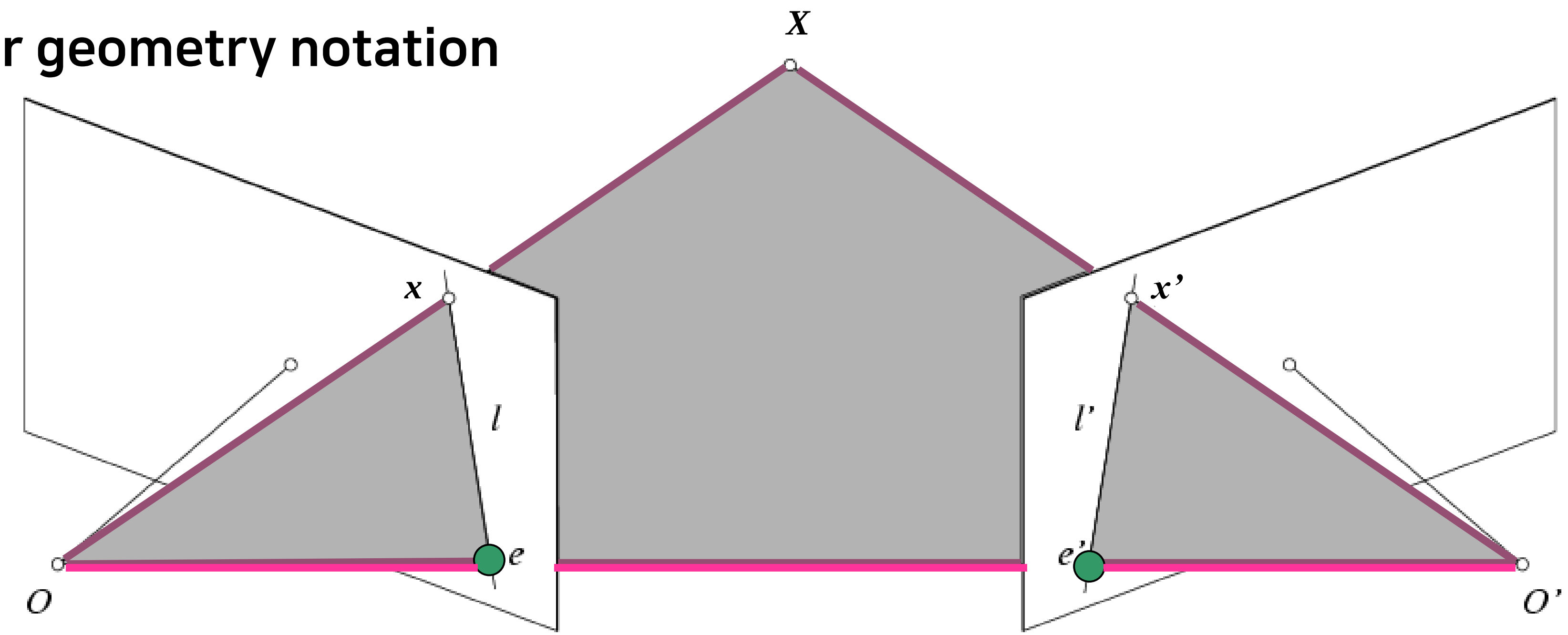
Baseline – line connecting the two camera centers

Epipoles
= intersections of baseline with image planes
= projections of the other camera center

Epipolar Plane – plane containing baseline (1D family)

Preliminaries

Epipolar geometry notation



Baseline – line connecting the two camera centers

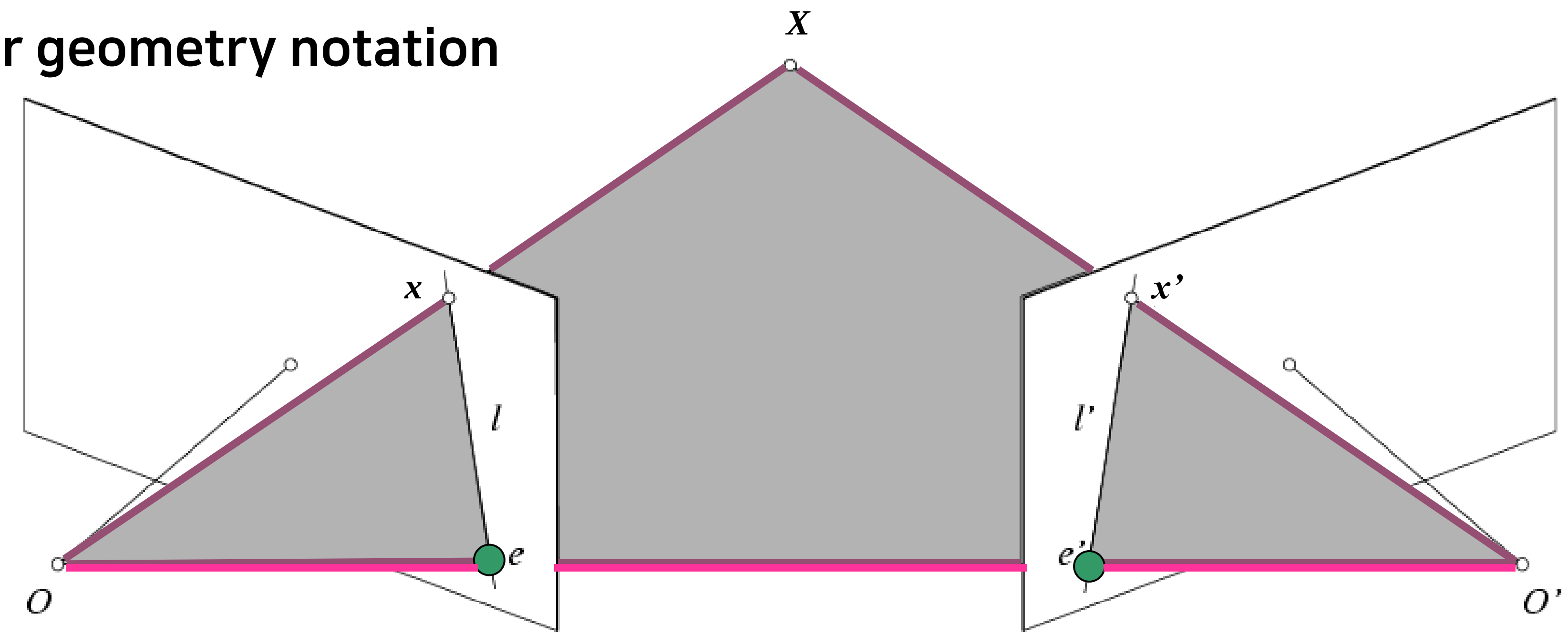
Epipoles
= intersections of baseline with image planes
= projections of the other camera center

Epipolar Plane – plane containing baseline (1D family)

Epipolar Lines - intersections of epipolar plane with image planes
(always come in corresponding pairs)

Preliminaries

Epipolar geometry notation



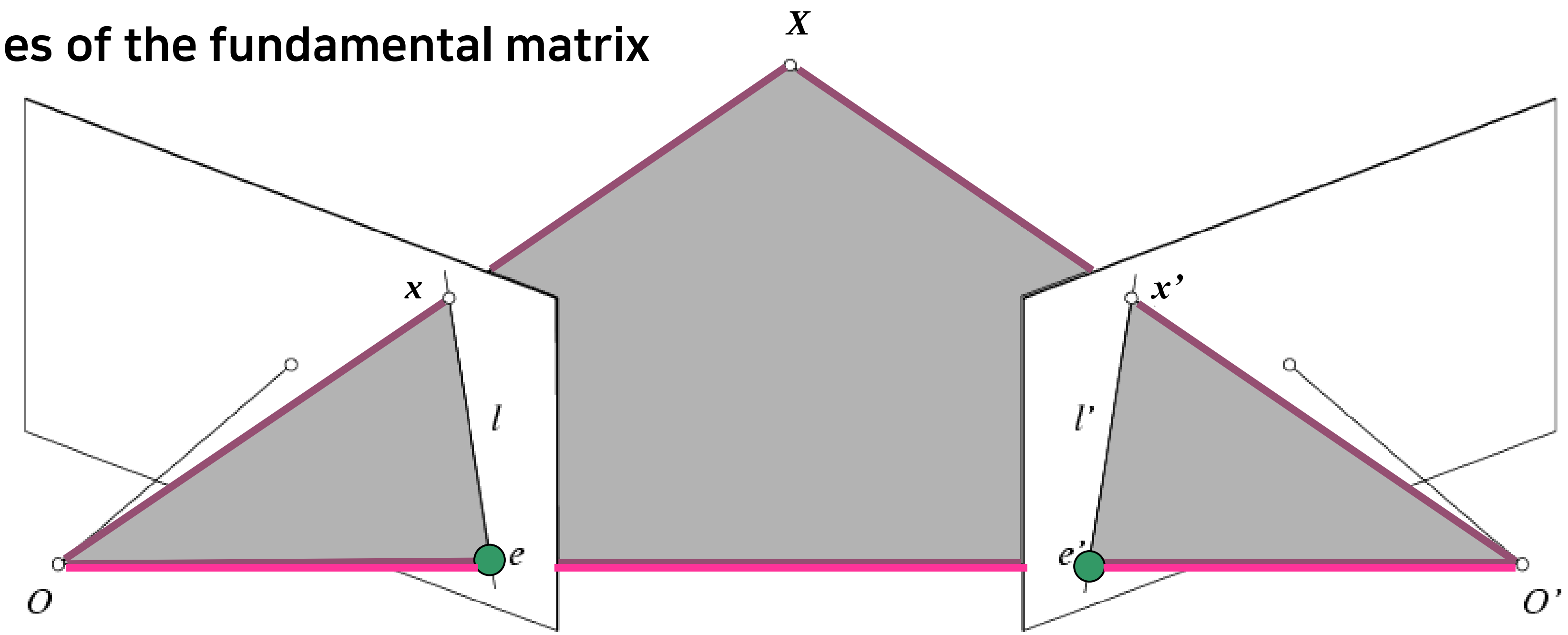
$$\hat{x} \cdot [t \times (R\hat{x}')] = 0 \quad \Rightarrow \quad \hat{x}^T E \hat{x}' = 0 \quad \text{with} \quad E = [t]_\times R$$

Skew
-symmetric
matrix

- Ex' is the epipolar line associated with x' ($l = Ex'$)
- $E^T x$ is the epipolar line associated with x ($l' = E^T x$)
- $Ee' = 0$ and $E^T e = 0$
- E is singular (rank two)
- E has five degrees of freedom (3 for R, 2 for t because it's up to a scale)

Preliminaries

Properties of the fundamental matrix



$x'^T F x = 0,$ $E = K'^T F K$

- Fx' is the epipolar line associated with x' ($l = Fx'$)
- $F^T x$ is the epipolar line associated with x ($l' = F^T x$)
- $Fe' = 0$ and $F^T e = 0$
- F is singular (rank two): $\det(F) = 0$
- E has seven degrees of freedom: 9 entries but defined up to scale, $\det(F) = 0$

Preliminaries

Estimating the Fundamental Matrix

- 8-point algorithm
  - Least squares solution using SVD on equations from 8 pairs of correspondences
  - Enforce $\det(F) = 0$ constraint using SVD on F

- 7-point algorithm
  - Use least squares to solve for null space (two vectors) using SVD and 7 pairs of correspondences
  - Solve for linear combination of null space vectors that satisfies $\det(F) = 0$

- Minimize reprojection error
  - Non-linear least squares

Note: estimation of F (or E) is degenerate for a planar scene.

Preliminaries

Solve a system of homogeneous linear equations

1. Write down the system of equations

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

$$uu'f_{11} + uv'f_{12} + uf_{13} + vu'f_{21} + vv'f_{22} + vf_{23} + u'f_{31} + v'f_{32} + f_{33} = 0$$

$$\mathrm{A}\boldsymbol{f} = \begin{bmatrix} u_1u_1{}' & u_1v_1{}' & u_1 & v_1u_1{}' & v_1v_1{}' & v_1 & u_1{}' & v_1{}' & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_nu_v' & u_nv_n{}' & u_n & v_nu_n{}' & v_nv_n{}' & v_n & u_n{}' & v_n{}' & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ \vdots \\ f_{33} \end{bmatrix} = \mathbf{0}$$

Preliminaries

Solve a system of homogeneous linear equations

1. Write down the system of equations
2. Solve $\boldsymbol{f}$ from $A\boldsymbol{f} = \boldsymbol{0}$ using SVD

Matlab:
```
[U, S, V] = svd(A);
f = V(:, end);
F = reshape(f, [3 3])';
```

Numpy:
```
U, S, V = np.linalg.svd(A)
f = V[:, end]
F = f.reshape(3, 3)
```

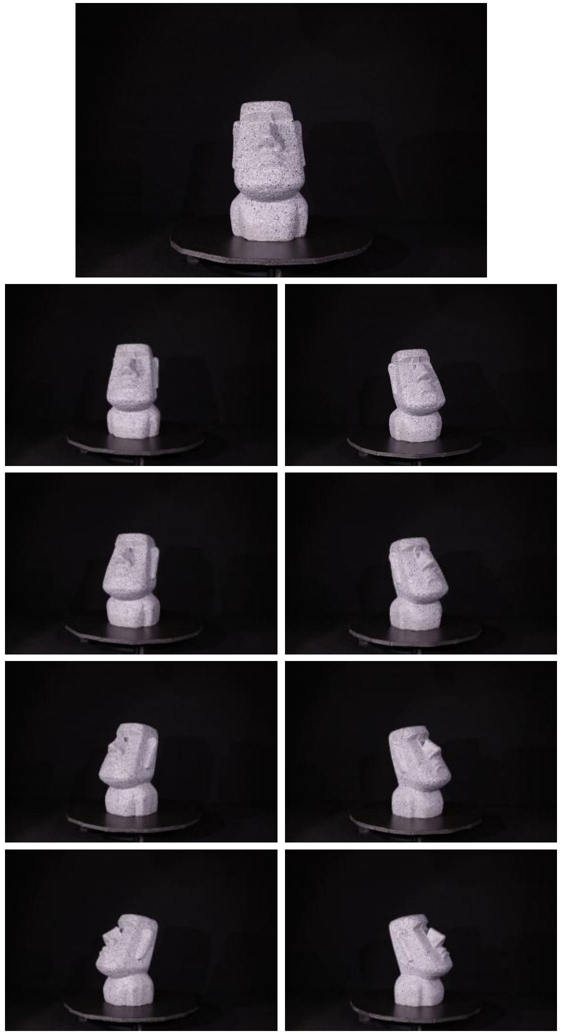
3. Resolve $\det(F) = 0$ constraint using SVD

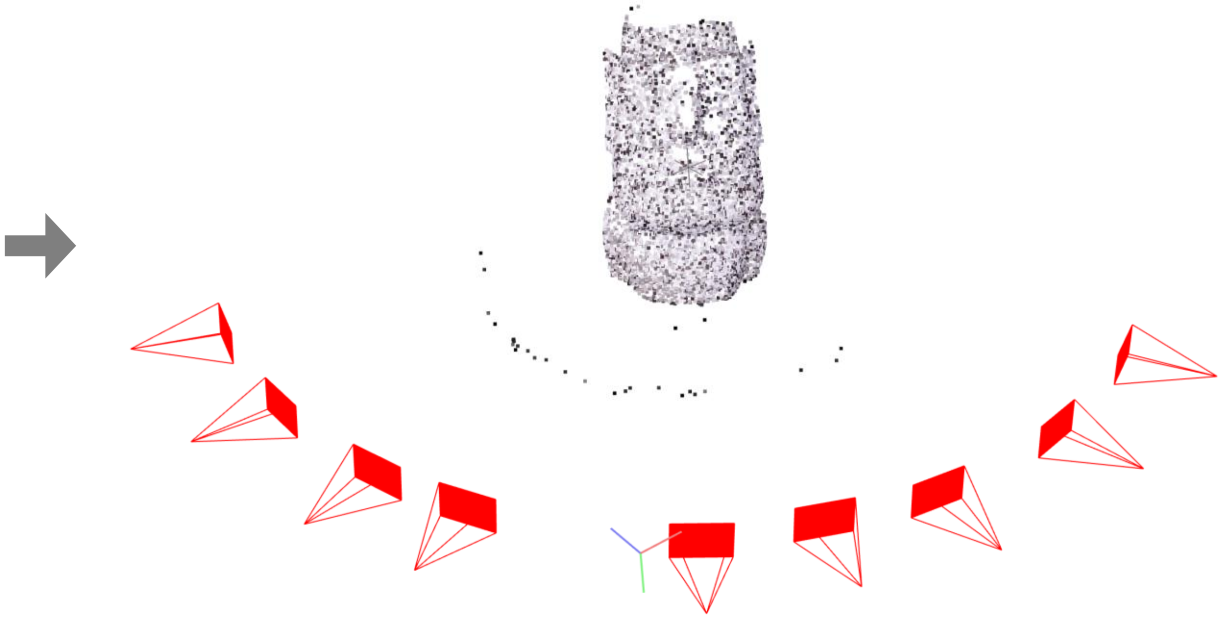
Matlab:
```
[U, S, V] = svd(F);
S(3,3) = 0;
F = U*S*V';
```

Numpy:
```
U, S, V = np.linalg.svd(A)
S[3,3] = 0;
F = U@S@V';
```

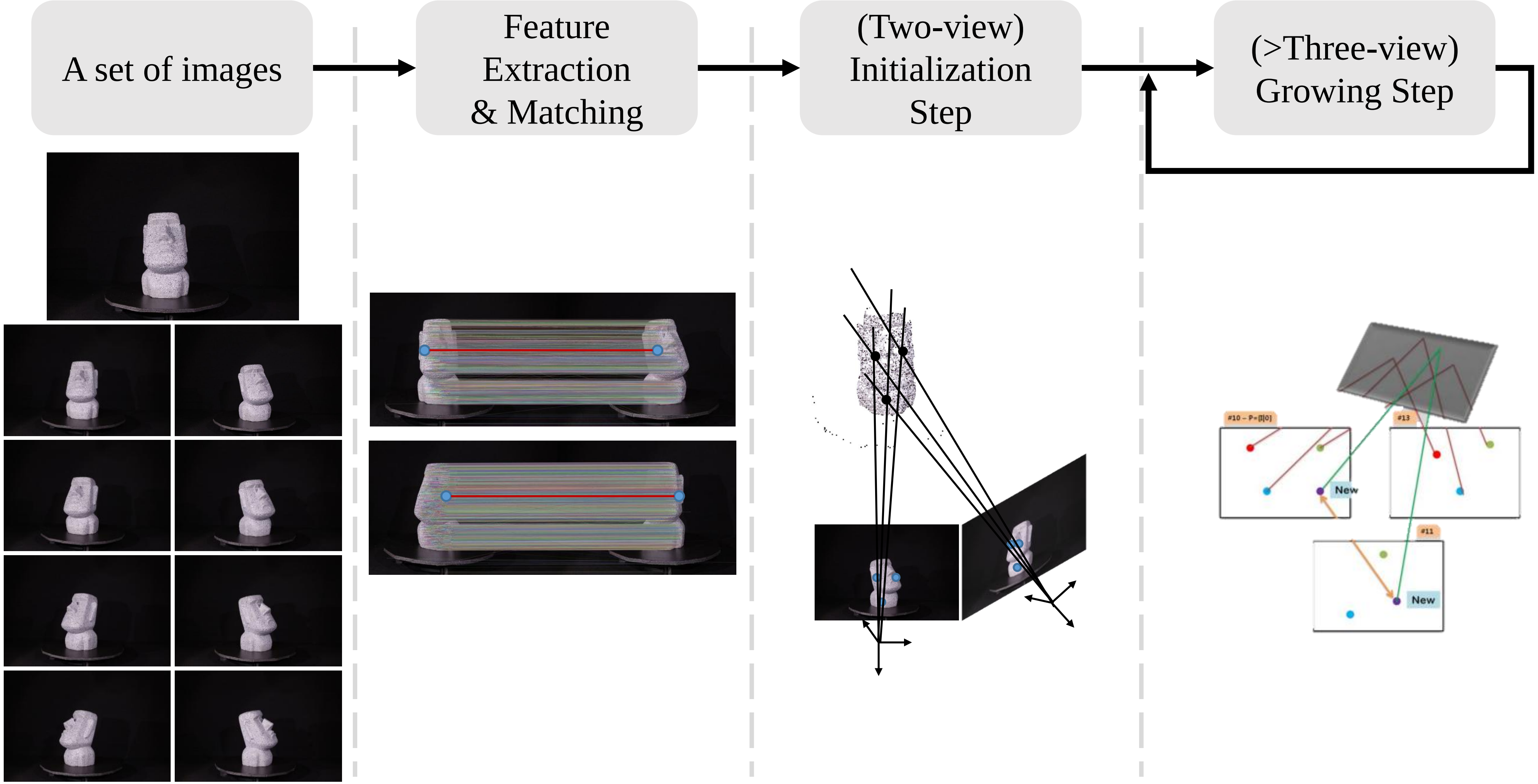
Goal

Build a 3D & Estimate camera poses, given the set of images





Overall



- Hartley et al. "Multiple view geometry in computer vision." Cambridge university press, (2003).
- Szeliski et al. "Computer vision: algorithms and applications." Springer Science & Business Media, (2010).

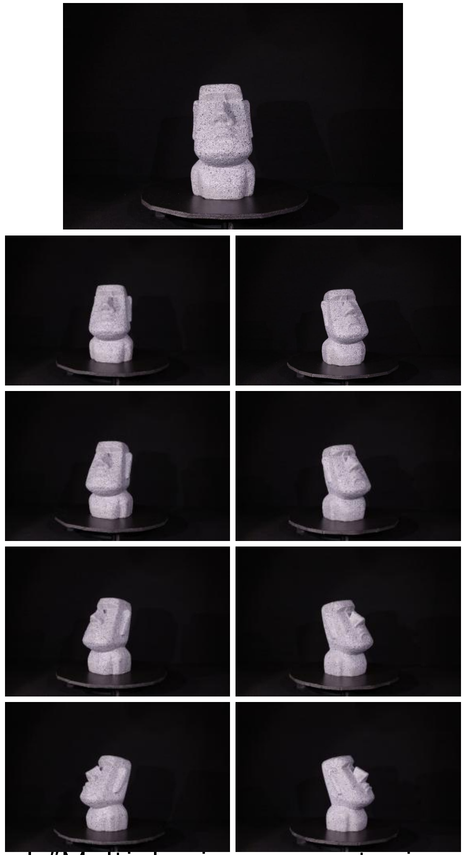
Overall

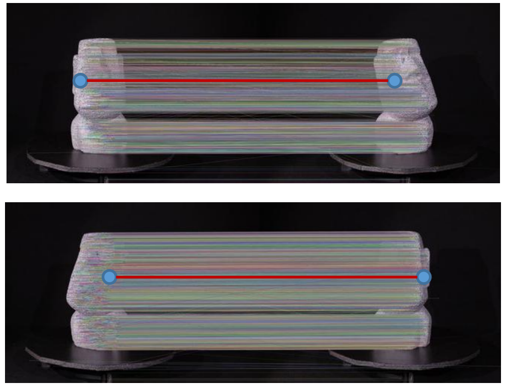
- **Correspondence** search, Relating images
  1. Extract SIFT from every image and find putative matches
  2. Outliers should be rejected by applying RANSAC
- **Initialization** Step
  3. Find the best image pair(simply, has the maximum matches or take the base-line into account)
  4. Estimate motion(R and t) and Reconstruct 3D points for the selected image pair. The camera coordinate of one camera is used for the world coordinate.
- **Growing** Step
  5. Search images which have enough points seeing the reconstructed 3D point
  6. Compute pose(R and t) for those images and reconstruct more 3D points seen from more than two images
  7. Bundle Optimization
- **Repeat** the Growing step until every camera is included.

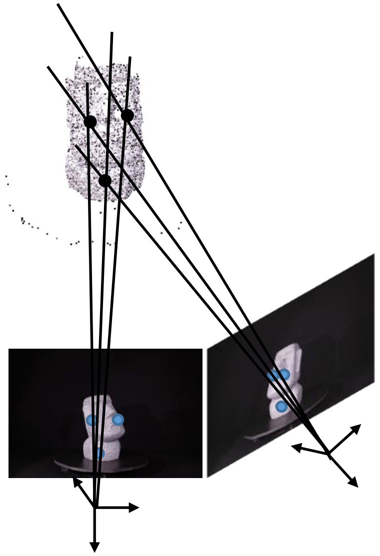
Overall

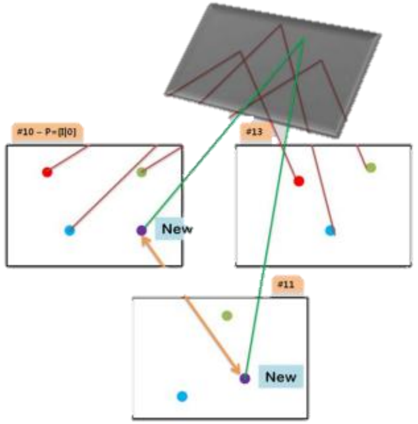
We will only do two-view in this PA

A set of images → Feature Extraction & Matching → (Two-view) Initialization Step → (>Three-view) Growing Step







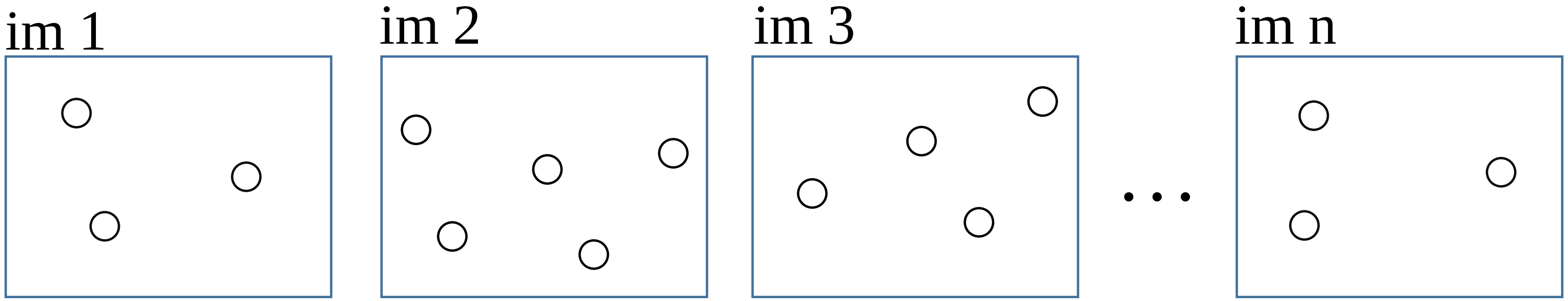


- Hartley et al. "Multiple view geometry in computer vision." Cambridge university press, (2003).
- Szeliski et al. "Computer vision: algorithms and applications." Springer Science & Business Media, (2010).

Step1. Feature extraction & matching in general

Feature types: SIFT, ORB, Hessian-Laplacian, ⋯

Feature Extraction

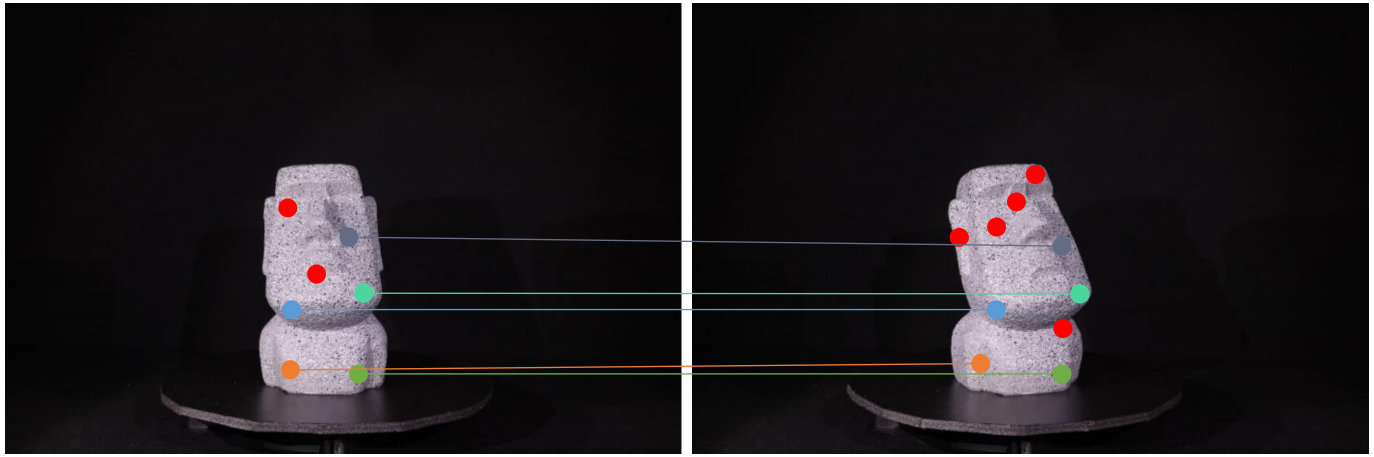


Each circle represents a set of detected features

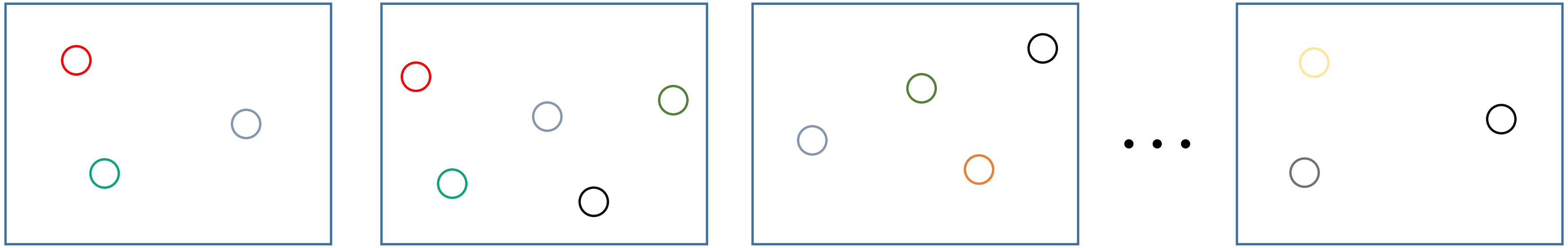
Step1. Feature extraction & matching in general

For each pair of images:

1. Match feature descriptors via approximate nearest neighbor
2. Solve for E and find inlier feature correspondences

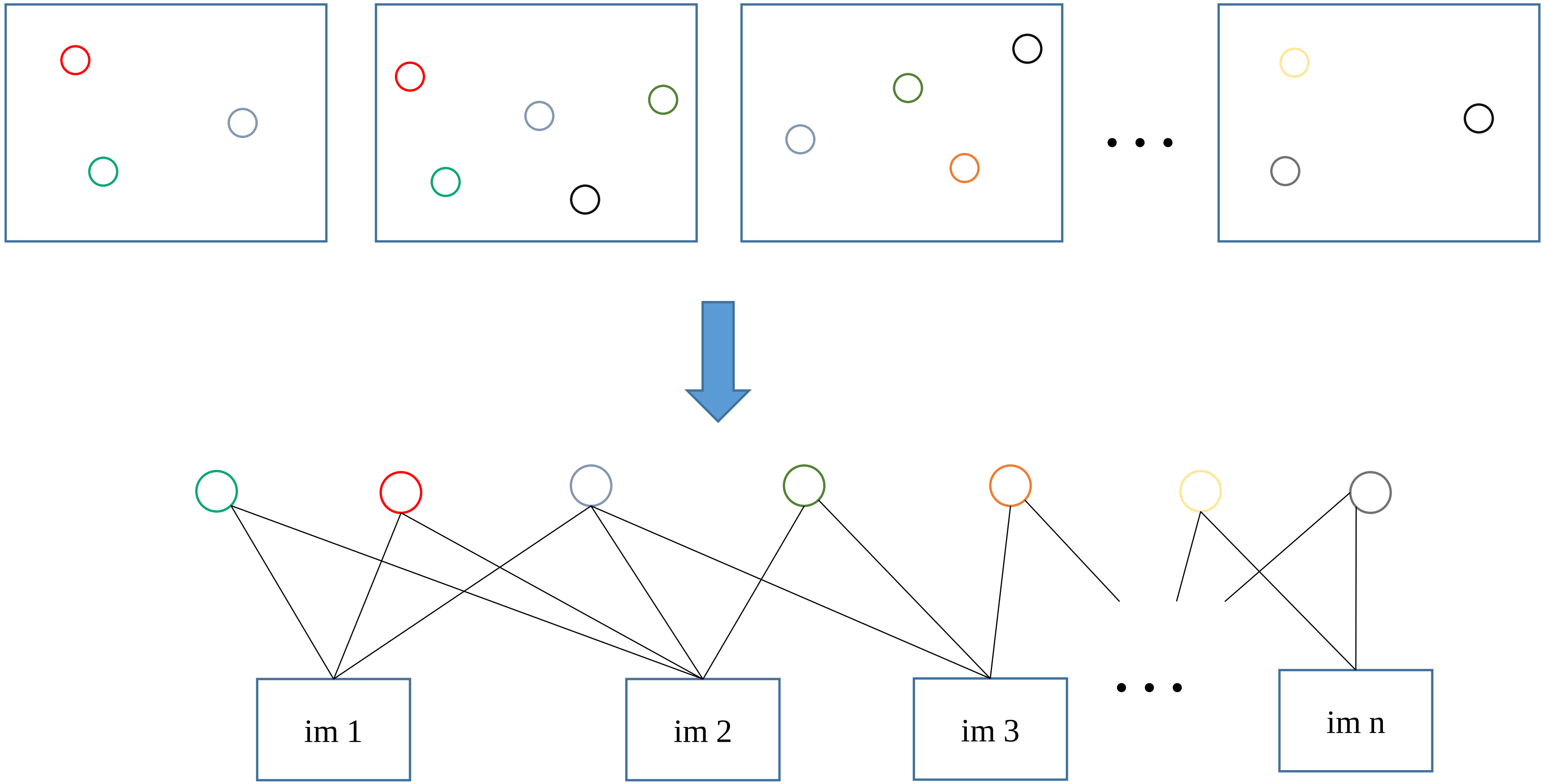


Feature Extraction



Points of same color have been matched to each other

Step1. Feature extraction & matching in general

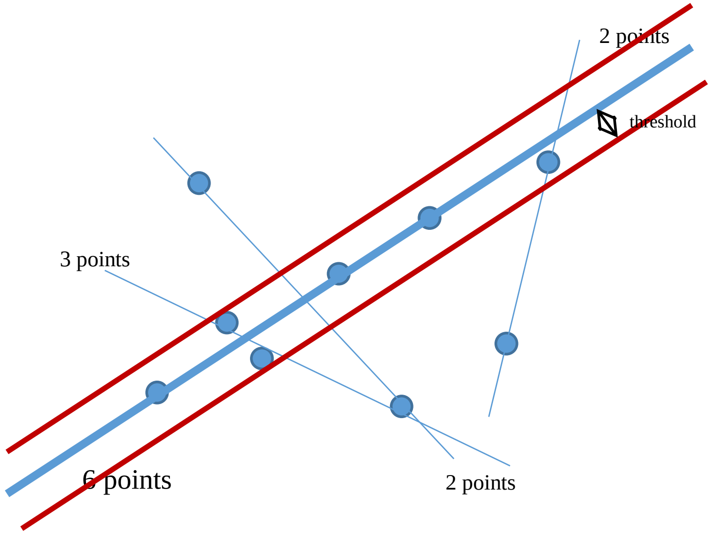


tracks graph: bipartite graph between observed 3D points and images

Step2. Essential matrix estimation

5-point algorithms with RANSAC

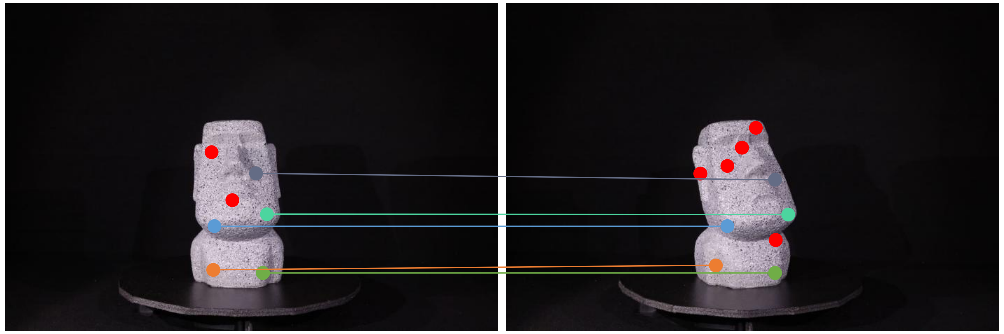
1. Randomly select sets of 5 points.
2. Generate E(hypothesis) and evaluate using other points with pre-defined threshold - epipolar distance
3. Do this for many times and choose the most supportive hypothesis having the most inliers.



Randomly Selected correspondence

$$\{x_{c1}, \ldots, x_{c5}\}$$
$$\{x'_{c1}, \ldots, x'_{c5}\}$$

E using calibrated 5-pt algorithm



Definition 9.16. The defining equation for the essential matrix is

$$\hat{\mathrm{x}}'^{\mathsf{T}} \mathrm{E} \hat{\mathrm{x}} = 0 \qquad (9.11)$$

in terms of the normalized image coordinates for corresponding points $\mathrm{x} \leftrightarrow \mathrm{x}'$.

Step3. Essential matrix decomposition

Essential Matrix Decomposition to [R|T]

- Camera matrix to essential metrix:

$$E = [t]_{\times}R = R[R^T t]$$

R, t: Rotation, Translation

- Essential matrix to camera matrix

$P' = [UWV^T | +u_3]$

$P' = [UWV^T | -u_3]$

$P' = [UW^TV^T | +u_3]$

$P' = [UW^TV^T | -u_3]$

- $SVD(E) = Udiag(1,1,0)V^T$
- $W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
- $u_3 = U(0,0,1)^T$: The last column vector of U

Proof. This is easily deduced from the decomposition of $\mathtt{E}$ as $[\mathbf{t}]_\times \mathtt{R} = \mathtt{SR}$, where $\mathtt{S}$ is skew-symmetric. We will use the matrices

$$\mathtt{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathtt{Z} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{9.13}$$

It may be verified that $\mathtt{W}$ is orthogonal and $\mathtt{Z}$ is skew-symmetric. From Result A4.1-(*p*581), which gives a block decomposition of a general skew-symmetric matrix, the 3×3 skew-symmetric matrix $\mathtt{S}$ may be written as $\mathtt{S} = k\mathtt{UZU}^\mathsf{T}$ where $\mathtt{U}$ is orthogonal. Noting that, up to sign, $\mathtt{Z} = \mathrm{diag}(1,1,0)\mathtt{W}$, then up to scale, $\mathtt{S} = \mathtt{U}\,\mathrm{diag}(1,1,0)\mathtt{WU}^\mathsf{T}$, and $\mathtt{E} = \mathtt{SR} = \mathtt{U}\,\mathrm{diag}(1,1,0)(\mathtt{WU}^\mathsf{T}\mathtt{R})$. This is a singular value decomposition of $\mathtt{E}$ with two equal singular values, as required. Conversely, a matrix with two equal singular values may be factored as $\mathtt{SR}$ in this way. □

Since $\mathtt{E} = \mathtt{U}\,\mathrm{diag}(1,1,0)\mathtt{V}^\mathsf{T}$, it may seem that $\mathtt{E}$ has six degrees of freedom and not five, since both $\mathtt{U}$ and $\mathtt{V}$ have three degrees of freedom. However, because the two singular values are equal, the SVD is not unique – in fact there is a one-parameter family of SVDs for $\mathtt{E}$. Indeed, an alternative SVD is given by $\mathtt{E} = (\mathtt{U}\,\mathrm{diag}(\mathtt{R}_{2\times 2}, 1))\,\mathrm{diag}(1,1,0)(\mathrm{diag}(\mathtt{R}^\mathsf{T}_{2\times 2}, 1))\mathtt{V}^\mathsf{T}$ for any 2×2 rotation matrix $\mathtt{R}$.

9.6.2 Extraction of cameras from the essential matrix

The essential matrix may be computed directly from (9.11) using normalized image coordinates, or else computed from the fundamental matrix using (9.12). (Methods of computing the fundamental matrix are deferred to chapter 11). Once the essential matrix is known, the camera matrices may be retrieved from $\mathtt{E}$ as will be described next. In contrast with the fundamental matrix case, where there is a projective ambiguity, the camera matrices may be retrieved from the essential matrix up to scale and a four-fold ambiguity. That is there are four possible solutions, except for overall scale, which cannot be determined.

We may assume that the first camera matrix is $\mathtt{P} = [\mathtt{I} \mid \mathbf{0}]$. In order to compute the second camera matrix, $\mathtt{P}'$, it is necessary to factor $\mathtt{E}$ into the product $\mathtt{SR}$ of a skew-symmetric matrix and a rotation matrix.

Result 9.18. *Suppose that the SVD of* $\mathtt{E}$ *is* $\mathtt{U}\,\mathrm{diag}(1,1,0)\mathtt{V}^\mathsf{T}$. *Using the notation of (9.13), there are (ignoring signs) two possible factorizations* $\mathtt{E} = \mathtt{SR}$ *as follows:*

$$\mathtt{S} = \mathtt{UZU}^\mathsf{T} \quad \mathtt{R} = \mathtt{UWV}^\mathsf{T} \quad \textit{or} \quad \mathtt{UW}^\mathsf{T}\mathtt{V}^\mathsf{T}\ . \tag{9.14}$$

Proof. That the given factorization is valid is true by inspection. That there are no other factorizations is shown as follows. Suppose $\mathtt{E} = \mathtt{SR}$. The form of $\mathtt{S}$ is determined by the fact that its left null-space is the same as that of $\mathtt{E}$. Hence $\mathtt{S} = \mathtt{UZU}^\mathsf{T}$. The rotation $\mathtt{R}$ may be written as $\mathtt{UXV}^\mathsf{T}$, where $\mathtt{X}$ is some rotation matrix. Then

$$\mathtt{U}\,\mathrm{diag}(1,1,0)\mathtt{V}^\mathsf{T} = \mathtt{E} = \mathtt{SR} = (\mathtt{UZU}^\mathsf{T})(\mathtt{UXV}^\mathsf{T}) = \mathtt{U}(\mathtt{ZX})\mathtt{V}^\mathsf{T}$$

from which one deduces that $\mathtt{ZX} = \mathrm{diag}(1,1,0)$. Since $\mathtt{X}$ is a rotation matrix, it follows that $\mathtt{X} = \mathtt{W}$ or $\mathtt{X} = \mathtt{W}^\mathsf{T}$, as required. □

The factorization (9.14) determines the $\mathbf{t}$ part of the camera matrix $\mathtt{P}'$, up to scale, from $\mathtt{S} = [\mathbf{t}]_\times$. However, the Frobenius norm of $\mathtt{S} = \mathtt{UZU}^\mathsf{T}$ is $\sqrt{2}$, which means that if $\mathtt{S} = [\mathbf{t}]_\times$ *including scale* then $\|\mathbf{t}\| = 1$, which is a convenient normalization for the baseline of the two camera matrices. Since $\mathtt{S}\mathbf{t} = \mathbf{0}$, it follows that $\mathbf{t} = \mathtt{U}(0,0,1)^\mathsf{T} = \mathbf{u}_3$, the last column of $\mathtt{U}$. However, the sign of $\mathtt{E}$, and consequently $\mathbf{t}$, cannot be determined. Thus, corresponding to a given essential matrix, there are four possible choices of the camera matrix $\mathtt{P}'$, based on the two possible choices of $\mathtt{R}$ and two possible signs of $\mathbf{t}$. To summarize:

Result 9.19. *For a given essential matrix* $\mathtt{E} = \mathtt{U}\,\mathrm{diag}(1,1,0)\mathtt{V}^\mathsf{T}$, *and first camera matrix* $\mathtt{P} = [\mathtt{I} \mid \mathbf{0}]$, *there are four possible choices for the second camera matrix* $\mathtt{P}'$, *namely*

$$\mathtt{P}' = [\mathtt{UWV}^\mathsf{T} \mid +\mathbf{u}_3] \ \text{ or } \ [\mathtt{UWV}^\mathsf{T} \mid -\mathbf{u}_3] \ \text{ or } \ [\mathtt{UW}^\mathsf{T}\mathtt{V}^\mathsf{T} \mid +\mathbf{u}_3] \ \text{ or } \ [\mathtt{UW}^\mathsf{T}\mathtt{V}^\mathsf{T} \mid -\mathbf{u}_3].$$

9.6.3 Geometrical interpretation of the four solutions

It is clear that the difference between the first two solutions is simply that the direction of the translation vector from the first to the second camera is reversed.

The relationship of the first and third solutions in result 9.19 is a little more complicated. However, it may be verified that

$$[\mathtt{UW}^\mathsf{T}\mathtt{V}^\mathsf{T} \mid \mathbf{u}_3] = [\mathtt{UWV}^\mathsf{T} \mid \mathbf{u}_3] \begin{bmatrix} \mathtt{VW}^\mathsf{T}\mathtt{W}^\mathsf{T}\mathtt{V}^\mathsf{T} & \\ & 1 \end{bmatrix}$$

and $\mathtt{VW}^\mathsf{T}\mathtt{W}^\mathsf{T}\mathtt{V}^\mathsf{T} = \mathtt{V}\,\mathrm{diag}(-1,-1,1)\mathtt{V}^\mathsf{T}$ is a rotation through 180° about the line joining the two camera centres. Two solutions related in this way are known as a "twisted pair".

The four solutions are illustrated in figure 9.12, where it is shown that a reconstructed point $\mathbf{X}$ will be in front of both cameras in one of these four solutions only. Thus, testing with a single point to determine if it is in front of both cameras is sufficient to decide between the four different solutions for the camera matrix $\mathtt{P}'$.

Note. The point of view has been taken here that the essential matrix is a homogeneous quantity. An alternative point of view is that the essential matrix is defined exactly by the equation $\mathtt{E} = [\mathbf{t}]_\times \mathtt{R}$, (i.e. including scale), and is determined only up to indeterminate scale by the equation $\mathbf{x}'^\mathsf{T}\mathtt{E}\mathbf{x} = 0$. The choice of point of view depends on which of these two equations one regards as the defining property of the essential matrix.

9.7 Closure

9.7.1 The literature

The essential matrix was introduced to the computer vision community by Longuet-Higgins [LonguetHiggins-81], with a matrix analogous to $\mathtt{E}$ appearing in the photogrammetry literature, e.g. [VonSanden-08]. Many properties of the essential matrix have been elucidated particularly by Huang and Faugeras [Huang-89], [Maybank-93], and [Horn-90].

The realization that the essential matrix could also be applied in uncalibrated situations, as it represented a projective relation, developed in the early part of the 1990s,

Step3. Essential matrix decomposition

Essential Matrix Decomposition to [R|T]

- (Result 9.18) Suppose that the SVD of E is Udiag(1,1,0)V^T. Using the notation of W and Z, there are (ignoring signs) two possible factorizations E=SR as follows:

$$\mathrm{S} = \mathrm{UZU}^T, \mathrm{R} = \mathrm{UWV}^T, \mathrm{R} = \mathrm{UW}^T\mathrm{V}^T$$

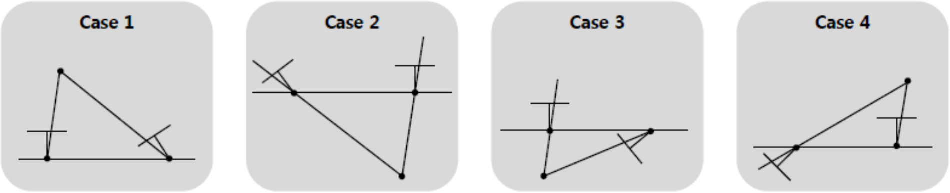
$$\mathrm{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } \mathrm{Z} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

- (Property 1) A 3x3 matrix is an essential matrix if and only if two of its singular values are equal, and the third is zero
- (Property 2: Block decomposition) 3x3 skew-symmetric matrix S may be written as S=kUZU^T w here U is orthogonal.

Step3. Essential matrix decomposition

Essential Matrix Decomposition to [R|T]

- There are four possible reconstruction
- Depth for both camera should be positive value (case 1)
- Not negative value (case 2, 3, 4)
- You should figure out the optimal camera pose

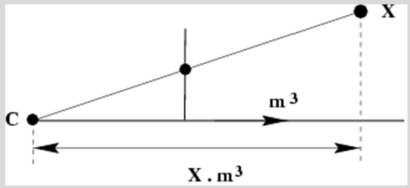


$P' = [UWV^T | +u_3]$

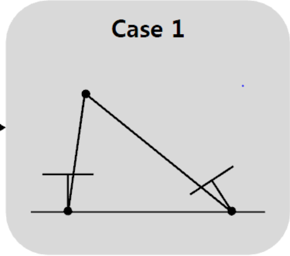
$P' = [UWV^T | -u_3]$

$P' = [UW^TV^T | +u_3]$

$P' = [UW^TV^T | -u_3]$



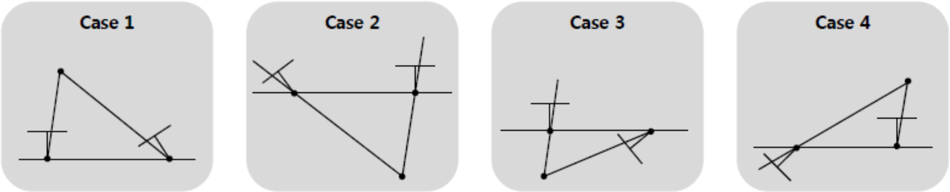
$$\mathbf{X} = (X, Y, Z, 1)^T, P = [M|\mathbf{p}_4]$$
$$P\mathbf{X} = w(x, y, 1)^T$$
$$\text{depth}(\mathbf{X}; P) = \frac{\text{sign}(\det M)w}{T\|\mathbf{m}_3\|}$$



Step3. Essential matrix decomposition

Essential Matrix Decomposition to [R|T]

- There are four possible reconstruction
- Depth for both camera should be positive value (case 1)
- Not negative value (case 2, 3, 4)
- You should figure out the optimal camera pose



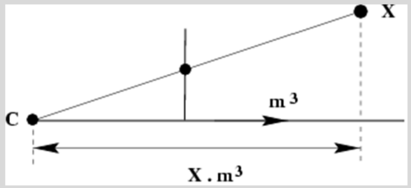
$P' = [UWV^T | +u_3]$

$P' = [UWV^T | -u_3]$

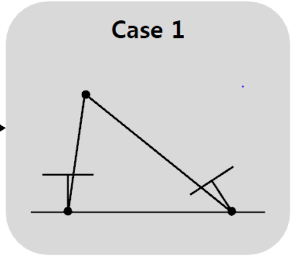
$P' = [UW^TV^T | +u_3]$

$P' = [UW^TV^T | -u_3]$

Triangulation



$$\mathbf{X} = (X, Y, Z, 1)^T,\ P = [M|\mathbf{p}_4]$$
$$P\mathbf{X} = w(x, y, 1)^T$$
$$\mathrm{depth}(\mathbf{X}; P) = \frac{\mathrm{sign}(\det M)w}{T\|\mathbf{m}_3\|}$$



Step4. Triangulation

Triangualation

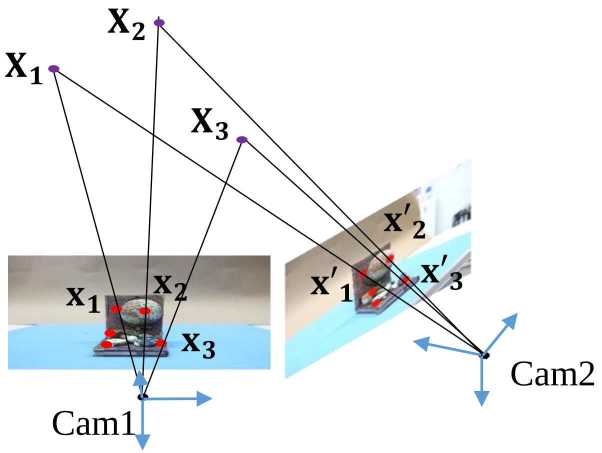
- Get 3D points from Camera pose & correspondences

$$x_{ci} = P X_i$$
$$[x_{ci}]_\times P X_i = 0$$

$$x(\boldsymbol{p}^{3T}\boldsymbol{X}) - (\boldsymbol{p}^{1T}\boldsymbol{X}) = 0$$
$$y(\boldsymbol{p}^{3T}\boldsymbol{X}) - (\boldsymbol{p}^{2T}\boldsymbol{X}) = 0$$
$$x(\boldsymbol{p}^{3T}\boldsymbol{X}) - y(\boldsymbol{p}^{1T}\boldsymbol{X}) = 0$$

$$A\boldsymbol{X} = 0$$
$$A = \begin{bmatrix} x\boldsymbol{p}^{3T} - \boldsymbol{p}^{1T} \\ y\boldsymbol{p}^{3T} - \boldsymbol{p}^{2T} \\ x'\boldsymbol{p}'^{3T} - \boldsymbol{p}'^{1T} \\ y'\boldsymbol{p}'^{3T} - \boldsymbol{p}'^{2T} \end{bmatrix}$$

X : 3D point
x : Point on image coordinate
K : Intrinsic matrix
P $(\mathbf{K}[\boldsymbol{R}|\boldsymbol{t}])$ **:** Extrinsic matrix

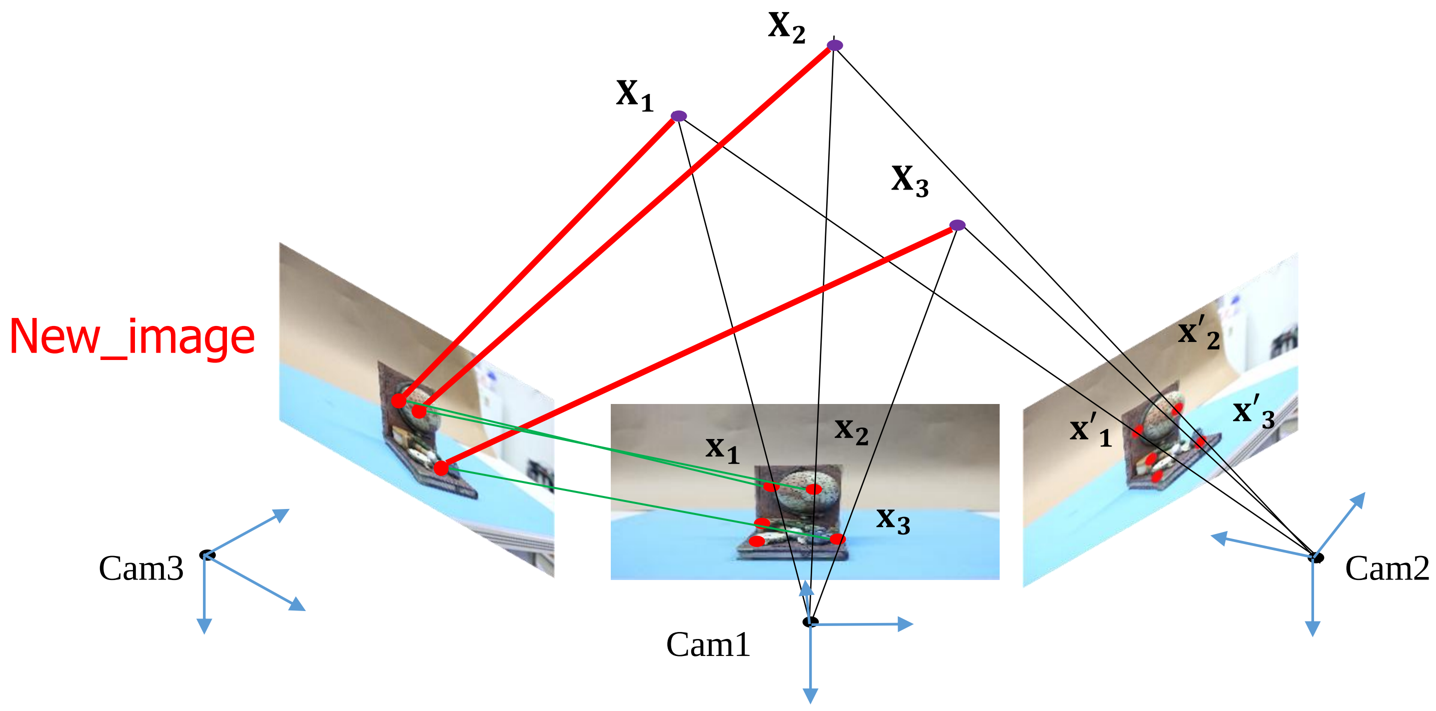


⟶ Solving linear equation by SVD

Step5. Growing Step (optional)

3-point PnP with RANSAC

- Estimate a camera pose of a selected image using 3-point PnP RANSAC

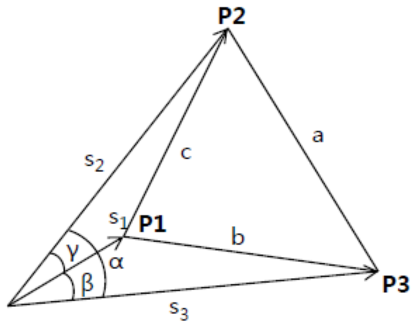


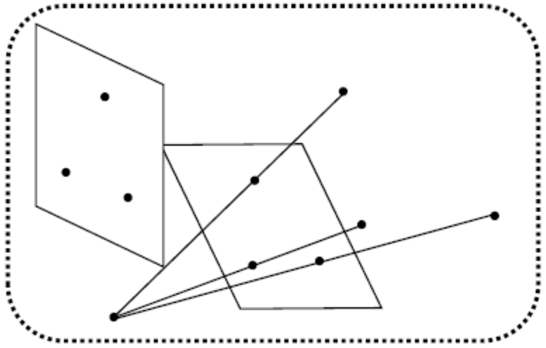
Step5. Growing Step (optional)

3-point PnP with RANSAC

- Estimate a camera pose of a selected image using 3-point PnP RANSAC
- 3-point algorithm

$$s_2{}^2 + s_3{}^2 - 2s_2s_3\cos\alpha = a^2$$
$$s_1{}^2 + s_3{}^2 - 2s_1s_3\cos\beta = b^2$$
$$s_1{}^2 + s_2{}^2 - 2s_1s_2\cos\gamma = c^2$$





Known : a, b, c, and unit vectors j_1, j_2, j_3

The **problem** is to determine the lengths s_1, s_2, s_3 from which the 3D vertex point positions P_1, P_2, and P_3 can be determined.

↓

Six solutions

Grunert(1841), Finsterwalder(1937), Merritt(1949), Fischler & Bolles(1981), Linnainmaa et al(1988), Grafarend et al(1989).

↓

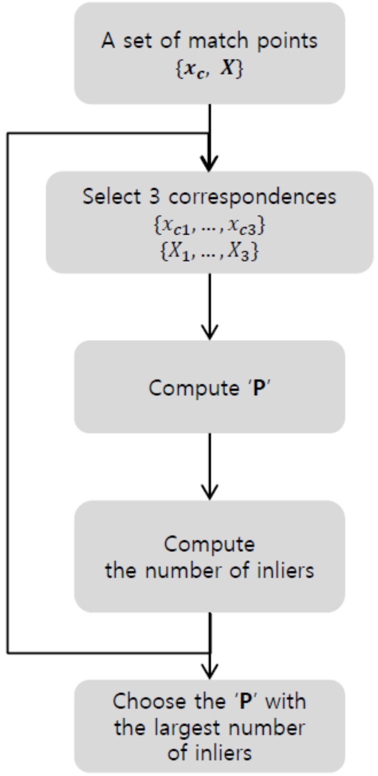
Estimate Pose(P)

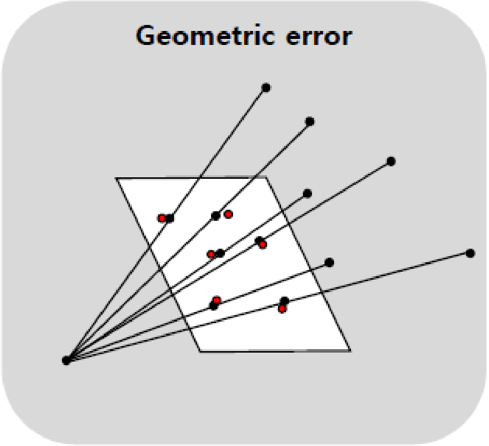
- Haralick, Bert M., et al. "Review and analysis of solutions of the three point perspective pose estimation problem." IJCV. (1994)

Step5. Growing Step (optional)

3-point PnP with RANSAC

- Compute the number of inliers
- Choose the best P with the largest number of inliers





$$d^2 = d(\boldsymbol{x}_1, KP_1\boldsymbol{X})^2 + d(\boldsymbol{x}_2, KP_2\boldsymbol{X})^2$$

$d < t$ pixels

- Haralick, Bert M., et al. "Review and analysis of solutions of the three point perspective pose estimation problem." IJCV. (1994)

Step5. Growing Step (optional)

Triangulation

- Get 3D points from Camera pose & correspondences

$$\boldsymbol{x}_{ci} = P\boldsymbol{X}_i$$

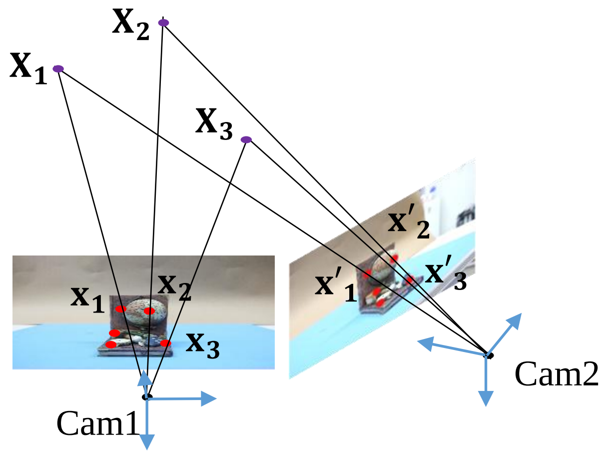
$$[\boldsymbol{x}_{ci}]_\times P\boldsymbol{X}_i = 0$$

$$x(\boldsymbol{p}^{3T}\boldsymbol{X}) - (\boldsymbol{p}^{1T}\boldsymbol{X}) = 0$$
$$y(\boldsymbol{p}^{3T}\boldsymbol{X}) - (\boldsymbol{p}^{2T}\boldsymbol{X}) = 0$$
$$x(\boldsymbol{p}^{3T}\boldsymbol{X}) - y(\boldsymbol{p}^{1T}\boldsymbol{X}) = 0$$

$$A\boldsymbol{X} = 0$$

$$A = \begin{bmatrix} x\boldsymbol{p}^{3T} - \boldsymbol{p}^{1T} \\ y\boldsymbol{p}^{3T} - \boldsymbol{p}^{2T} \\ x'\boldsymbol{p}'^{3T} - \boldsymbol{p}'^{1T} \\ y'\boldsymbol{p}'^{3T} - \boldsymbol{p}'^{2T} \end{bmatrix}$$

X : 3D point
x : Point on image coordinate
K : Intrinsic matrix
P (K[$\boldsymbol{R}|\boldsymbol{t}$]) : Extrinsic matrix

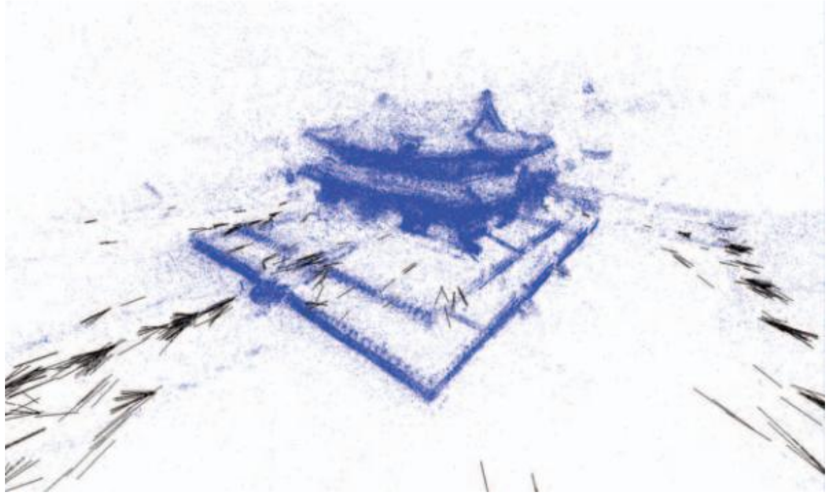


Step6. Optimization (optional)

Bundle Adjustment

- Refines a visual reconstruction to produce jointly optimal 3D structure and viewing parameters
- ‘Bundle’ refers to the bundle of light rays leaving each 3D feature and converging on each camer a center.

Before Bundle adjustment



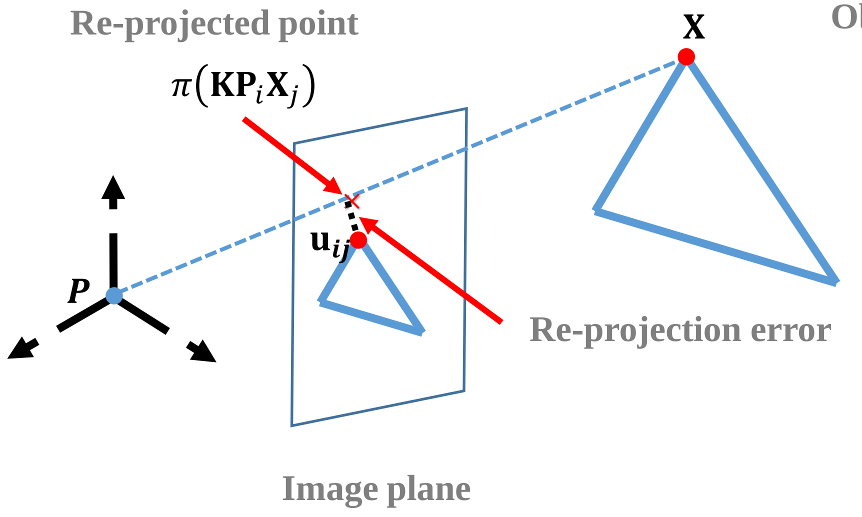
After Bundle adjustment

- Triggs, Bill, et al. "Bundle adjustment—a modern synthesis." International workshop on vision algorithms. (1999).
- Jeong, Yekeun, et al. "Pushing the envelope of modern methods for bundle adjustment." IEEE transactions on pattern analysis and machine intelligence. (2012).

Step6. Optimization (optional)

Bundle Adjustment’s Mathematical Problem

- Minimize re-projection error
- Non-linear Least Square approach
- Good approximate values are needed



Objective function

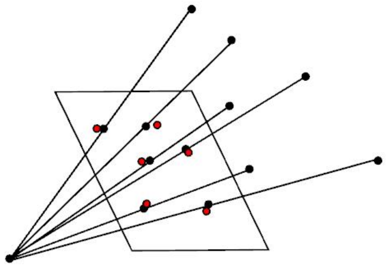
$$C(\mathbf{P},\mathbf{X}) = \sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij} \parallel \mathbf{u}_{ij} - \langle \mathbf{KP}_i\mathbf{X}_j \rangle \parallel^2$$

n: The number of cameras
m: The number of features
$\pi(\cdot)$: The projection function
$(\mathcal{R}^3 \rightarrow \mathcal{R}^2)$
w_{ij}: indicator variable
1 if visible, 0 other wise

Step6. Optimization (optional)

LM Optimization

- Optimize the function with ‘lsqnonlin’ function
- See the error(residual) is decreasing
- 3D points and camera poses might be barely moved



$$\textbf{CostFuction} = min \sum_i d(x_i, \hat{x}_i)^2$$
$$= min \sum_i d(x_i, KP\boldsymbol{X}_i)^2$$

lsqnonlin

Solve nonlinear least-squares (nonlinear data-fitting) problems

Equation

Solves nonlinear least-squares curve fitting problems of the form

$$\min_x \|f(x)\|_2^2 = \min_x \left(f_1(x)^2 + f_2(x)^2 + ... + f_n(x)^2\right)$$

Syntax

```
x = lsqnonlin(fun,x0)
x = lsqnonlin(fun,x0,lb,ub)
x = lsqnonlin(fun,x0,lb,ub,options)
x = lsqnonlin(problem)
[x,resnorm] = lsqnonlin(...)
[x,resnorm,residual] = lsqnonlin(...)
[x,resnorm,residual,exitflag] = lsqnonlin(...)
[x,resnorm,residual,exitflag,output] = lsqnonlin(...)
[x,resnorm,residual,exitflag,output,lambda] = lsqnonlin(...)
[x,resnorm,residual,exitflag,output,lambda,jacobian] = lsqnonlin(...)
```

```
options=optimset('Algorithm',{'levenberg-marquardt' 0.001},'Display','off');
options=optimset('Algorithm',{'levenberg-marquardt' 0.001},'TolFun',1e-8,'TolX',1e-8,'Display','off');
```

Step7. Camera calibration (optional)

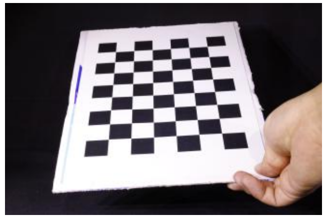
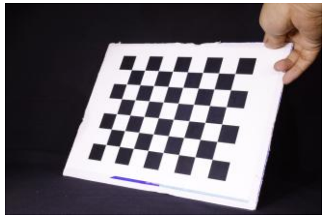
Camera Calibration with Checker Board

- Camera intrinsic matrix

$$\mathbf{P} = \overbrace{\mathbf{K}}^{\text{Intrinsic Matrix}} \times \overbrace{[\mathbf{R} \mid \mathbf{t}]}^{\text{Extrinsic Matrix}}$$

$$\mathbf{K} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

- Checker board example

Camera Calibration Flowchart

Define real world coordinates of 3D points using checkerboard pattern of known size.

↓

Capture the images of the checkerboard from different viewpoints.

↓

Use **findChessboardCorners** method in OpenCV to find the pixel coordinates (u, v) for each 3D point in different images

↓

Find camera parameters using **calibrateCamera** method in OpenCV, the 3D points, and the pixel coordinates.

https://foss4g.tistory.com/1665

Code Implementations

Step 0. Setting for Python and Matlab

- To use MATLAB within Python, please install the MATLAB Engine API.
- MATLAB R2022b or later is required.
- For more information, refer to the README file.

https://kr.mathworks.com/help/matlab/matlab_external/install-the-matlab-engine-for-python.html

Note: Due to GIST policy, we are unable to provide a license for MATLAB R2022b. Please use the 30-day trial version instead.

Code Implementations

Step 1. Feature extraction and matching

Extract features from two images using the SIFT algorithm and perform feature matching between them.

- We extract features from each image using the SIFT algorithm.
- The extracted features are then matched using the KNN algorithm.

Your Task) Complete the **matching_two_image()** in **feature_matching.py**

Code Implementations

Step 1. Feature extraction and matching

Extract features from two images using the SIFT algorithm and perform feature matching between them.

- Complete the **matching_two_image()** in **feature_matching.py**

- Allow functions:
  - cv2.imread
  - cv2.cvtColor()
  - cv2.SIFT_create()
  - cv2.SIFT_create().*
  - cv2.BFMatcher()
  - cv2.BFMatcher().*
  - cv2.drawMatchesKnn()
  - Numpy
  - Other functions are not allowed

Code Implementations

Step 2. Essential Matrix Estimation

Essential matrix estimation using 5 point algorithm and RANSAC.

- Convert cv2.KeyPoint and cv2.DMatch object into readable NumPy matrices.
- **Normalize** matched keypoints using intrinsic camera matrix
- In RANSAC step,
  - randomly **select 5 points** correspondences from each images and calculate essential matrix using eng.calibrated_fivepoint (refer to Step2/calibrated_fivepoint.m)
  - compute the error using the formula:

$$\text{error} = \text{diag}(\hat{x}^T E x),$$

and identify inlier points whose error falls within a predefined threshold.

Your Task) Complete the **essential_matrix_estimation()** in **E_estimation.py**

Code Implementations

Step 2. Essential Matrix Estimation

Essential matrix estimation using 5 point algorithm and RANSAC.

- Complete the **essential_matrix_estimation()** in **E_estimation.py**

- Allow functions:
  - numpy
  - tqdm
  - eng.calibrated_fivepoint

- Not allow functions:
  - cv2

Code Implementations

Step 3. Essential Matrix Decomposition

Calculate camera pose using essential matrix.

- Calculate U and V using SVD.

$$\mathrm{SVD}(E) = U\mathrm{diag}(1,1,0)V^T$$

- Make candidates of 4 camera matrix.

$$P_1 = [UWV^T | +u_3]$$
$$P_2 = [UWV^T | -u_3]$$
$$P_3 = [UW^TV^T | +u_3]$$
$$P_4 = [UW^TV^T | -u_3]$$

- Evaluate each candidate pose by triangulate inlier points. (refer to step 4)

Your Task) Complete the **essential_matrix_decomposition()** in **E_decomposition.py**

Code Implementations

Step 3. Essential Matrix Decomposition

Calculate camera pose using essential matrix.

- Complete the **essential_matrix_decomposition()** in **E_decomposition.py**

- Allow functions:
  - numpy
  - Tqdm

- Disallow functions:
  - cv2

Code Implementations

Step 4. Triangulation

Calculate 3D points cloud using triangulation and camera pose

- Calculate A following:

$$A = \begin{bmatrix} x\boldsymbol{p}^{3T} - \boldsymbol{p}^{1T} \\ y\boldsymbol{p}^{3T} - \boldsymbol{p}^{2T} \\ x'\boldsymbol{p}'^{3T} - \boldsymbol{p}'^{1T} \\ y'\boldsymbol{p}'^{3T} - \boldsymbol{p}'^{2T} \end{bmatrix}$$

- Solve linear system using SVD satisfying:

$$A\boldsymbol{X} = 0$$

Your Task) Complete the **triangulate_points()** in **triangulation.py**

Code Implementations

Step 4. Triangulation

Calculate 3D points cloud using triangulation and camera pose

- Complete the **triangulate_points()** in **triangulation.py**

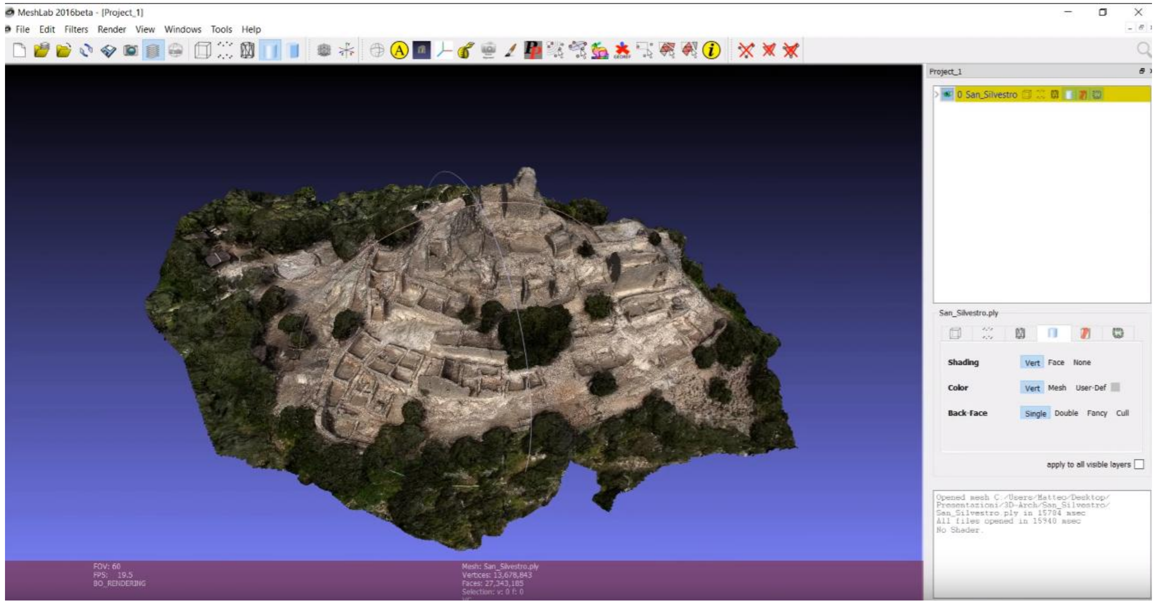
- Allow functions:
  - numpy
  - tqdm

- Deny functions:
  - cv2

Code Implementations

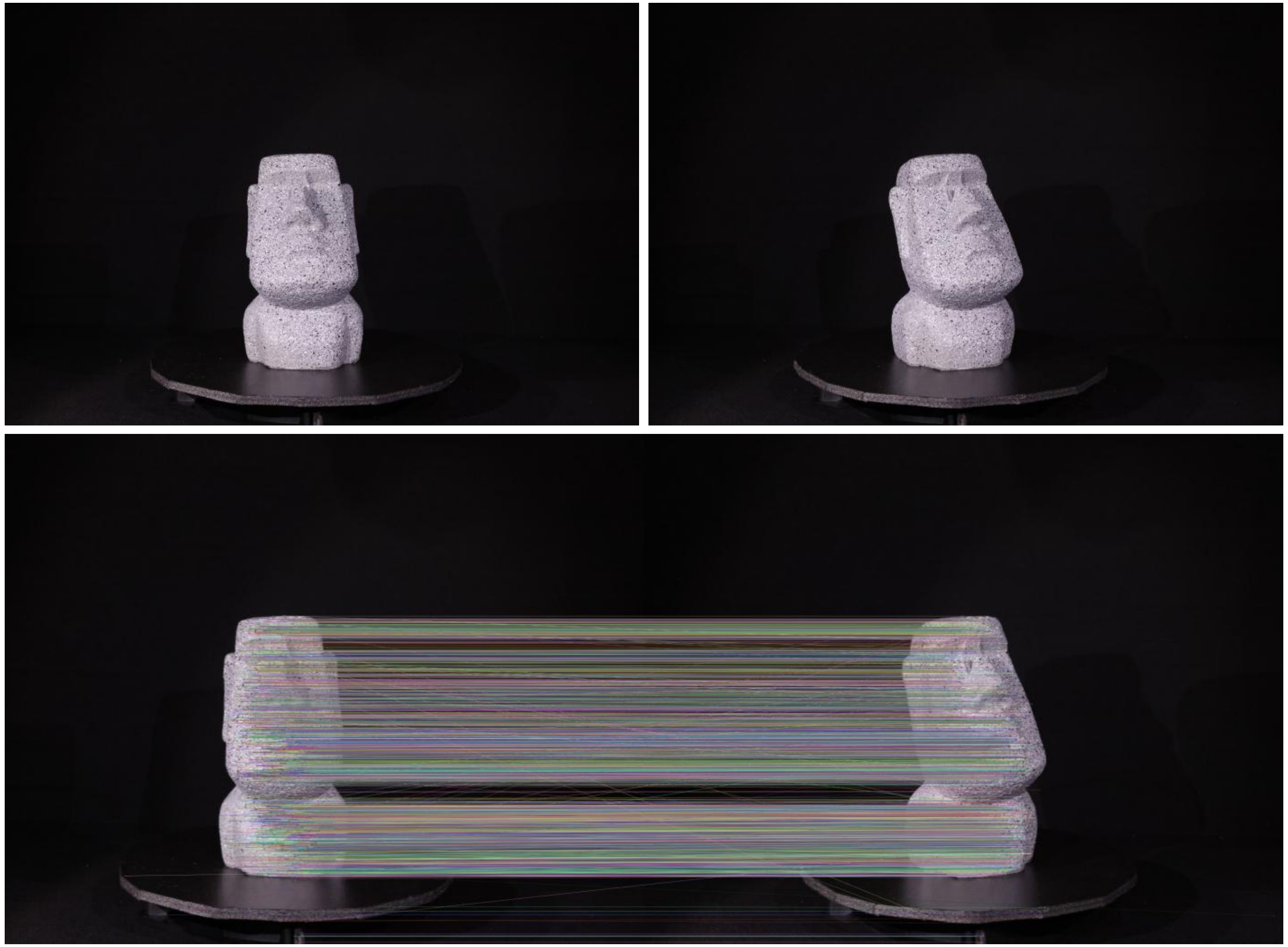
Visualization and Hyperparameter modification

- All visualization codes are already implemented in main_two_view.py
- Open the file two_view_result.ply using [MeshLab]
- Modify the Python arguments to adjust the hyperparameters, and write the final parameters in the report. (more information in README file)



Example Output

Moai feature matching output



Example Output

Moai point cloud output (two-view)

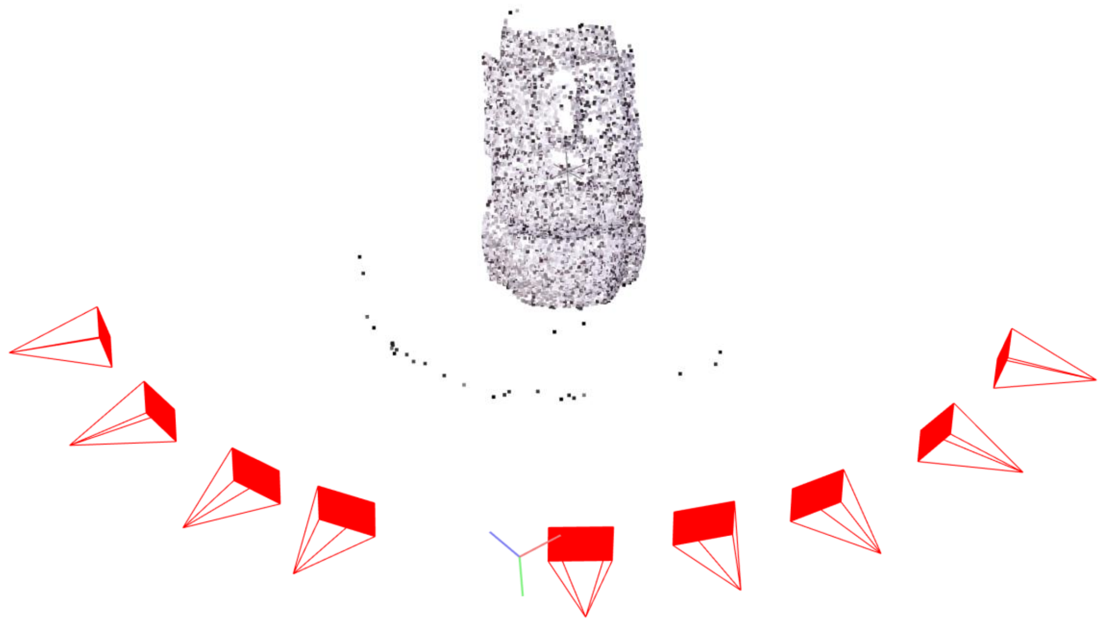
Left view

Front view

Right view

Example Output

Moai point cloud output (multi-view)



Guidance

Complete your code

- Following the steps below to complete your code based on a given skeleton.

Step 0. Install Matlab Engine API for python	
Step 1. Feature extraction and matching	**[feature_matching.py]**
Step 2. Essential matrix estimation with RANSAC	**[E_estimation.py]**
Step 3. Essential matrix decomposition	**[E_decomposition.py]**
Step 4. Triangulation	**[triangulation.py]**

- Before you start, carefully review the **README file** and **main_two_view.py.**
- Fill in the **#todo** sections in the given skeleton.
- Your implementation will be evaluated by running **main_two_view.py**.

data and code here!

Guidance

Write your report

- After completing your code, you need to write a report on your implementation.
- Your report should include:

Understanding the Steps	Explain the algorithms and key concepts used.
Visualizing the Results	Present feature matching, 3D point cloud.
Analyzing the Results	Discuss any issues and possible solutions.

- You have to write more than 3 pages.

Additional Credit

Multi-view SfM

- If you reconstruct 3D models from multiple view images (more than 3 views), I will give a huge extra credit (up to 5pts, see. Step 5, 6)
- Using main_multi_view.py

SfM with your own dataset

- Complete the function **camera_calibration()** in **camera_calibration.py**
- Make your dataset and run SfM with camera calibration
- Tips for making your own dataset (using your mobile phone)
    1. Use a fixed-focus camera.
    2. Do calibration for camera parameter estimation using camera_calibration() (to get intrinsic).
    3. Take pictures of your target scene with the camera of which the intrinsic parameter is known now.
    4. Run your SfM program with your dataset by replacing the images and the camera intrinsic matrix K to yours.

Instructions

Multi-view SfM

You should implement:

- Step 0. Settings for using Matlab in python (2 points)
- Step 1. Feature extraction (2 points) and matching (3 points)
- Step 2. Essential matrix estimation with RANSAC (5 points)
- Step 3. Essential matrix decomposition (5 points)
- Step 4. Triangulation (5 points)

You should write:

- A report (3 points)

Additional credit with multi-view SfM(up to 5 point)
and custom dataset (maximum 3 point)

Remember!

- 0. No Plagiarism
- 0. No delay
- 0. No use of any open libraries/functions and **AI assistance(like chatGPT).**

TA session : 2025.5.1 & 2025.5.8

Due Date: 2025.5.10

Any Questions: newdm2000@gm.gist.ac.kr (TA)

Good Luck!

Instructions

References

- Computer Vision: Algorithms and Applications (http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf)
  - Structure from Motion (ch. 7)

- Phillp Torr’s Structure from Motion toolkit
  - Includes F-matrix estimation, RANSAC, Triangulation and etc.
  - https://kr.mathworks.com/matlabcentral/fileexchange/4576-structure-and-motion-toolkit-in-matlab