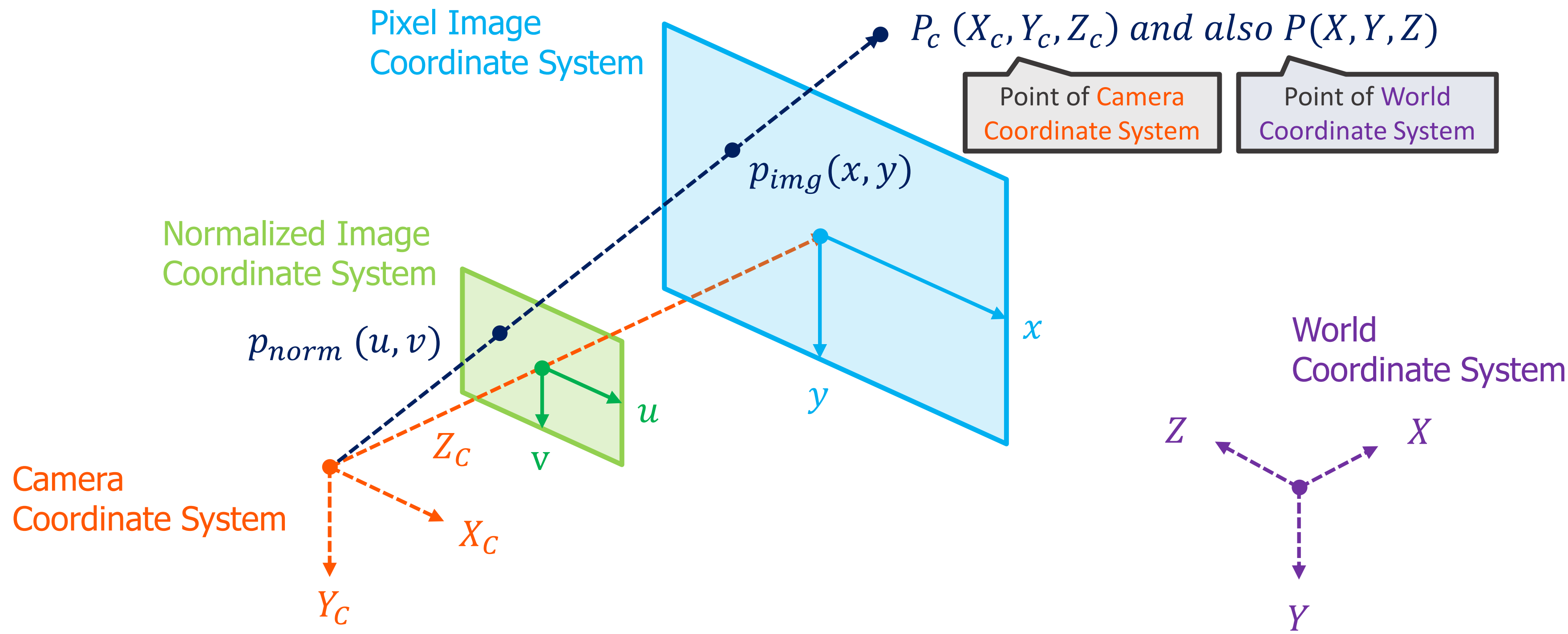
Programming Assignment2

Structure-from-Motion

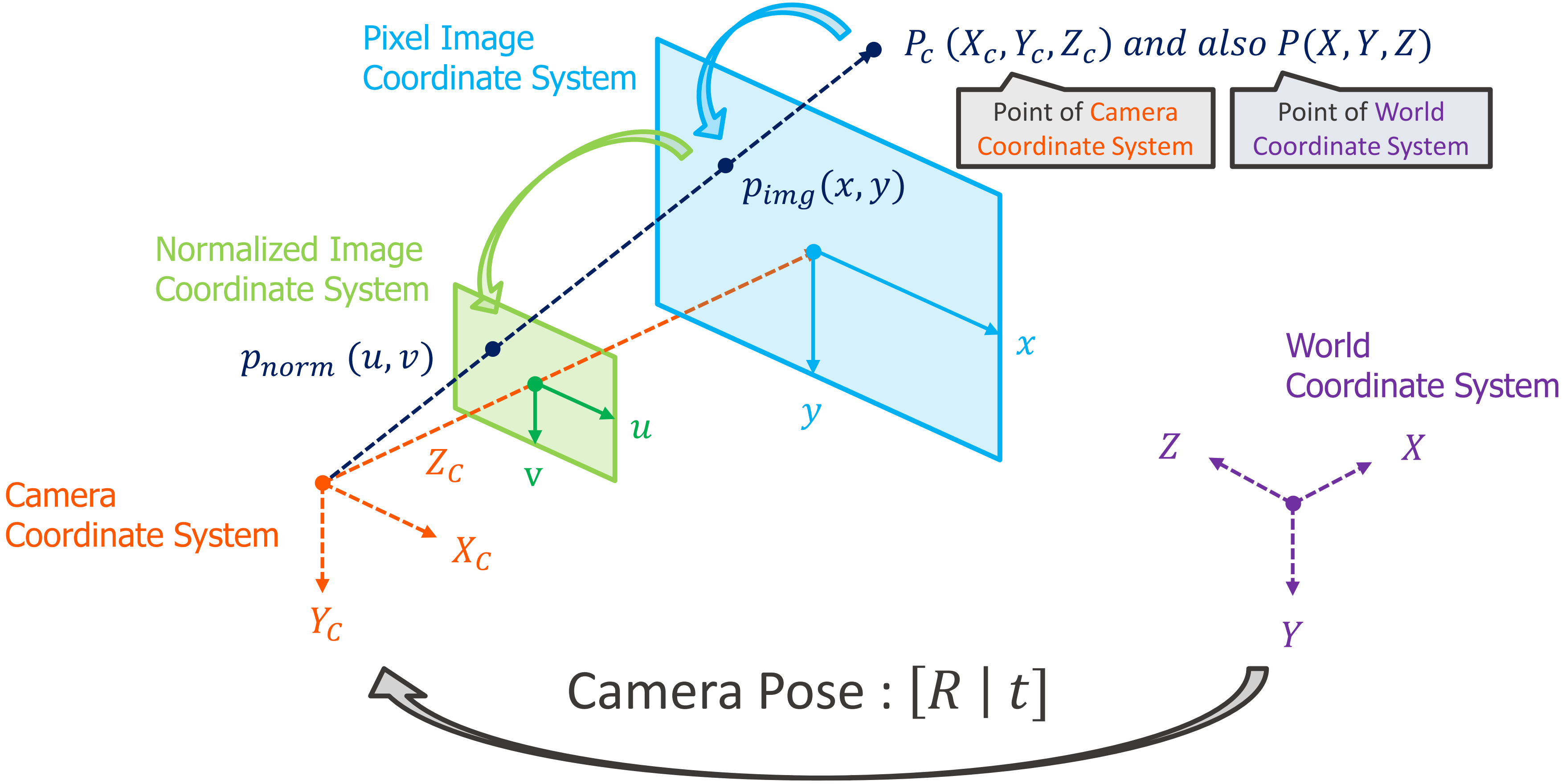
Prof. Hae-Gon Jeon

Recall: Coordinate System



Recall: Coordinate System

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



Recall: Coordinate System

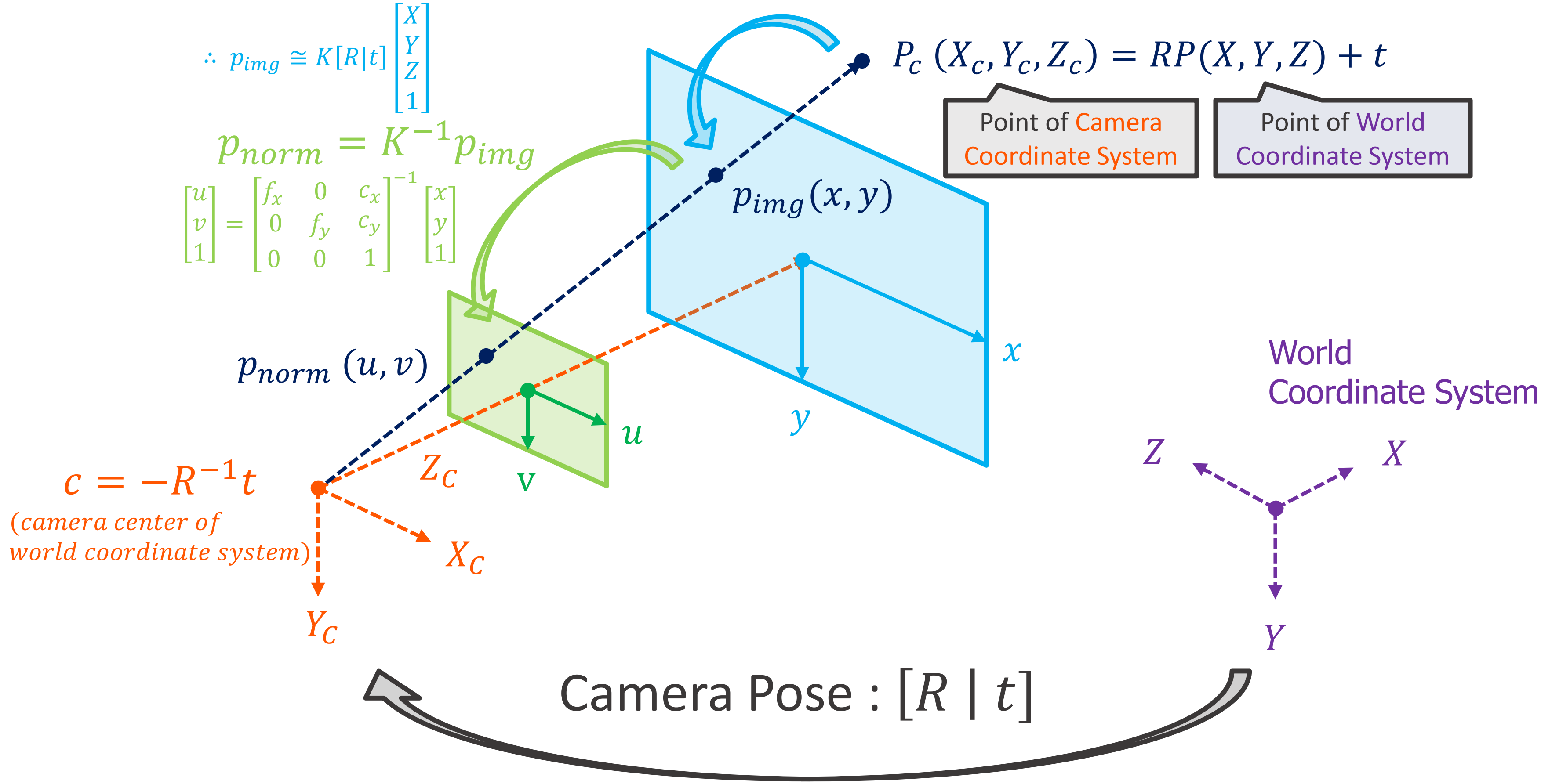
$$p_{norm}(u,v) \leftrightarrow p_{img}(x,y) \leftarrow P_c(X_c,Y_c,Z_c) \leftrightarrow P(X,Y,Z)$$

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

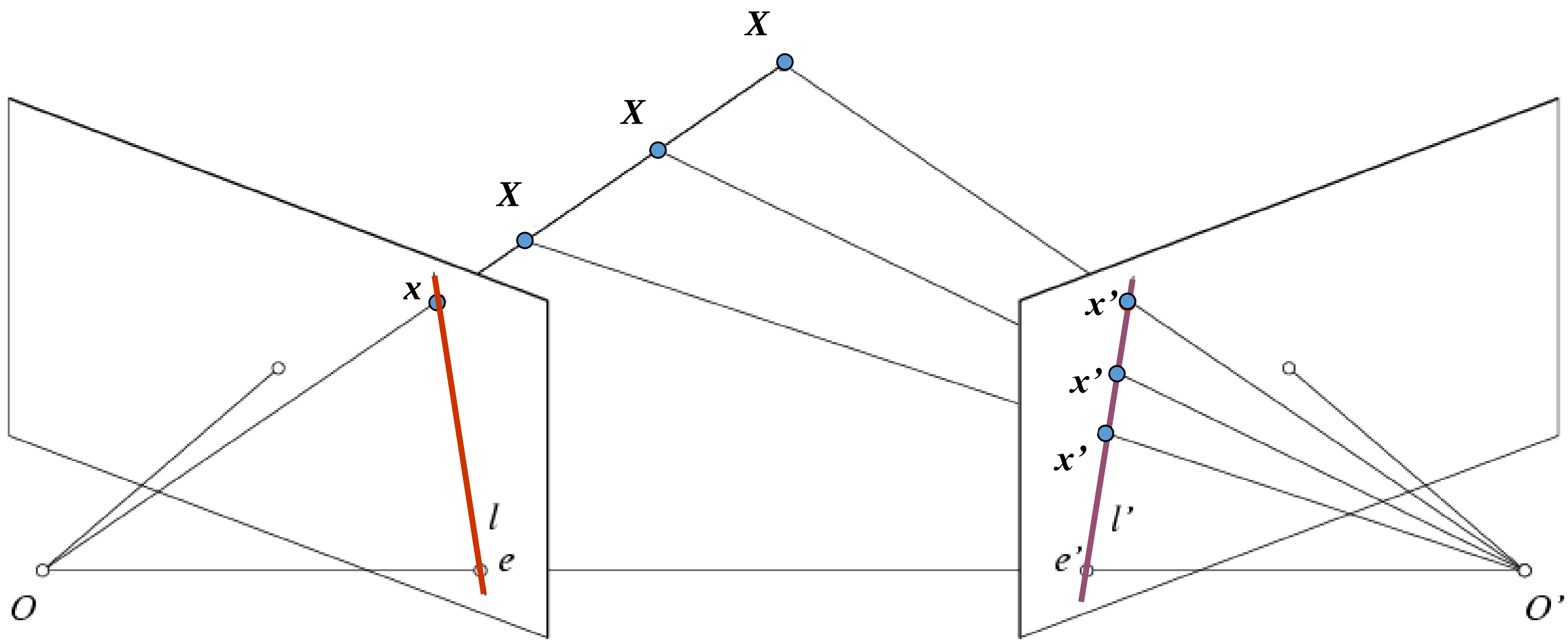
$$\therefore \; p_{img} \cong K[R|t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

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

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



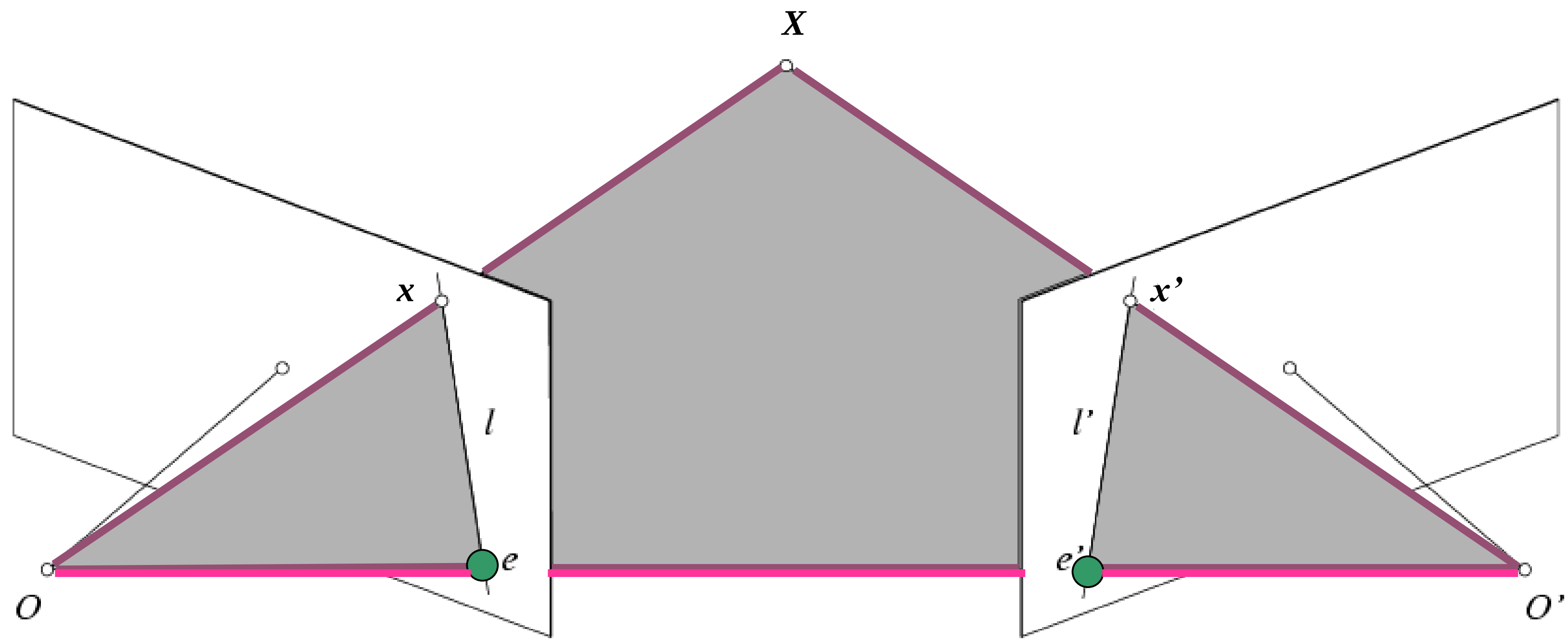
Recall: Epipolar constraint



Potential matches for *x* have to lie on the corresponding line *l'*.

Potential matches for *x'* have to lie on the corresponding line *l*.

Recall: Epipolar geometry notation



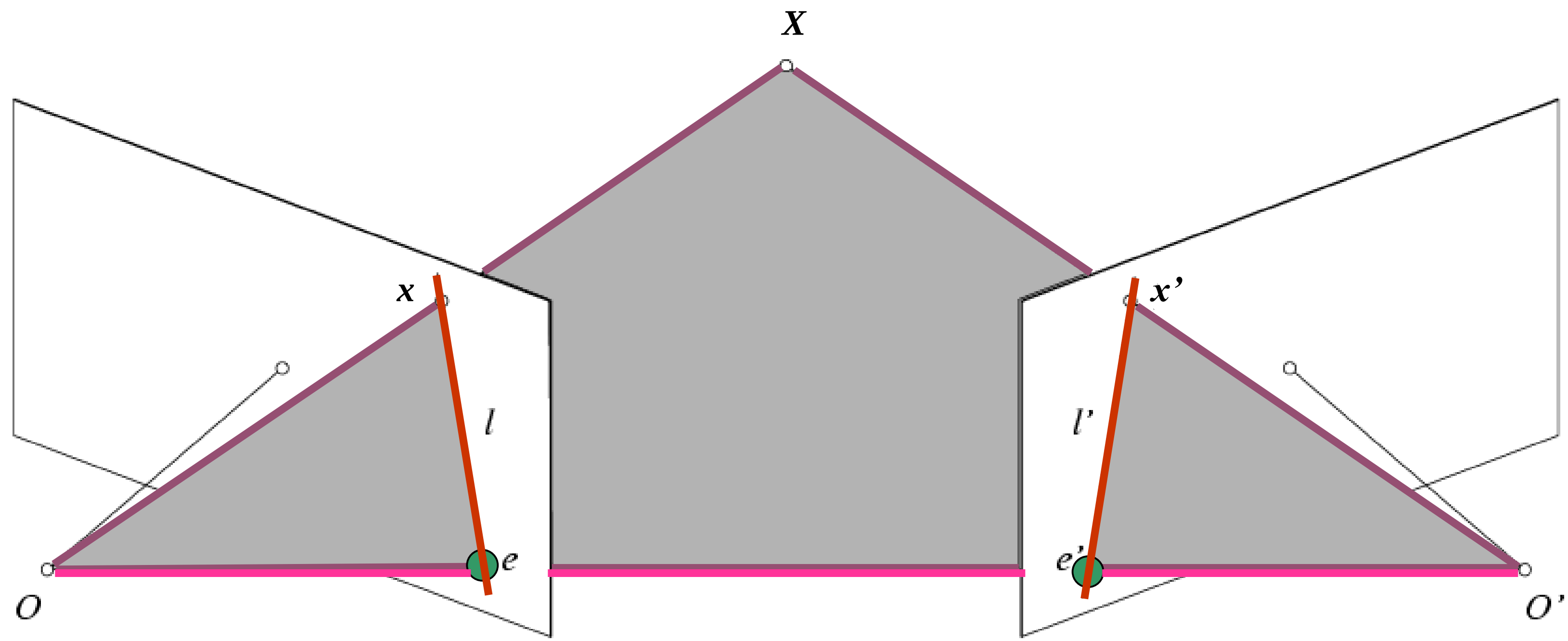
- **Baseline** – line connecting the two camera centers

- **Epipoles**

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

- **Epipolar Plane** – plane containing baseline (1D family)

Recall: Epipolar geometry notation



- **Baseline** – line connecting the two camera centers

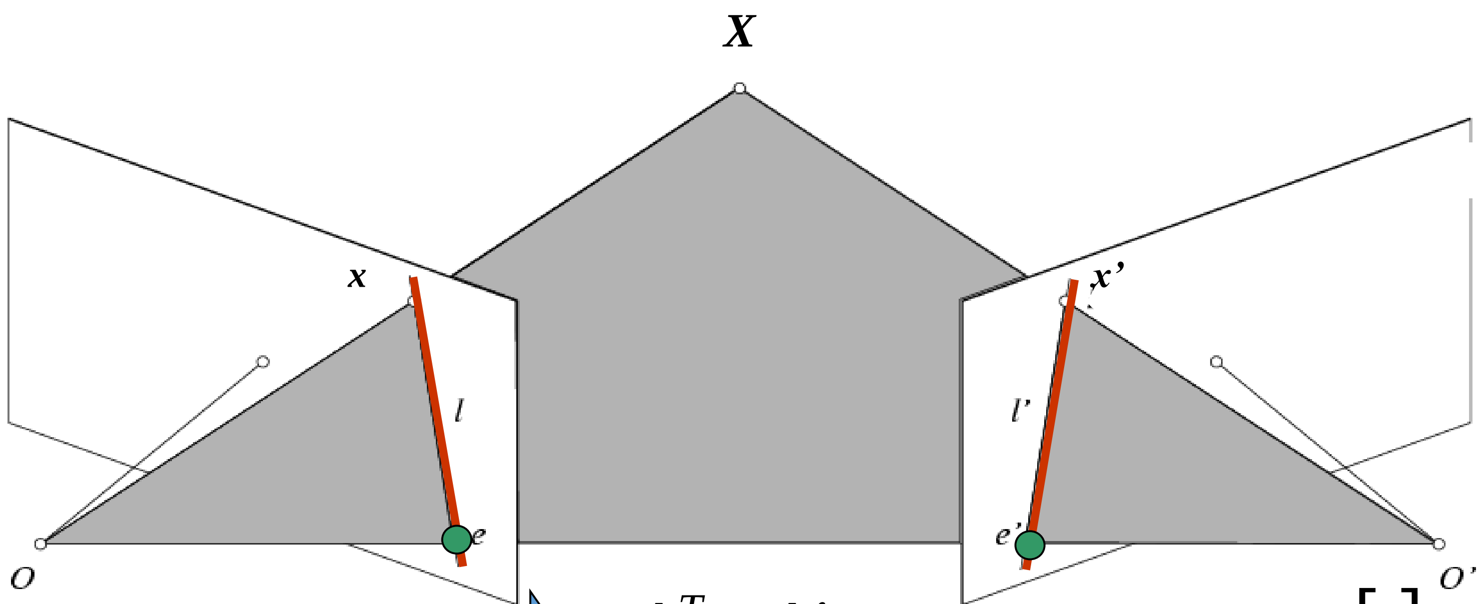
- **Epipoles**

= 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)

Recall: Properties of the Essential matrix



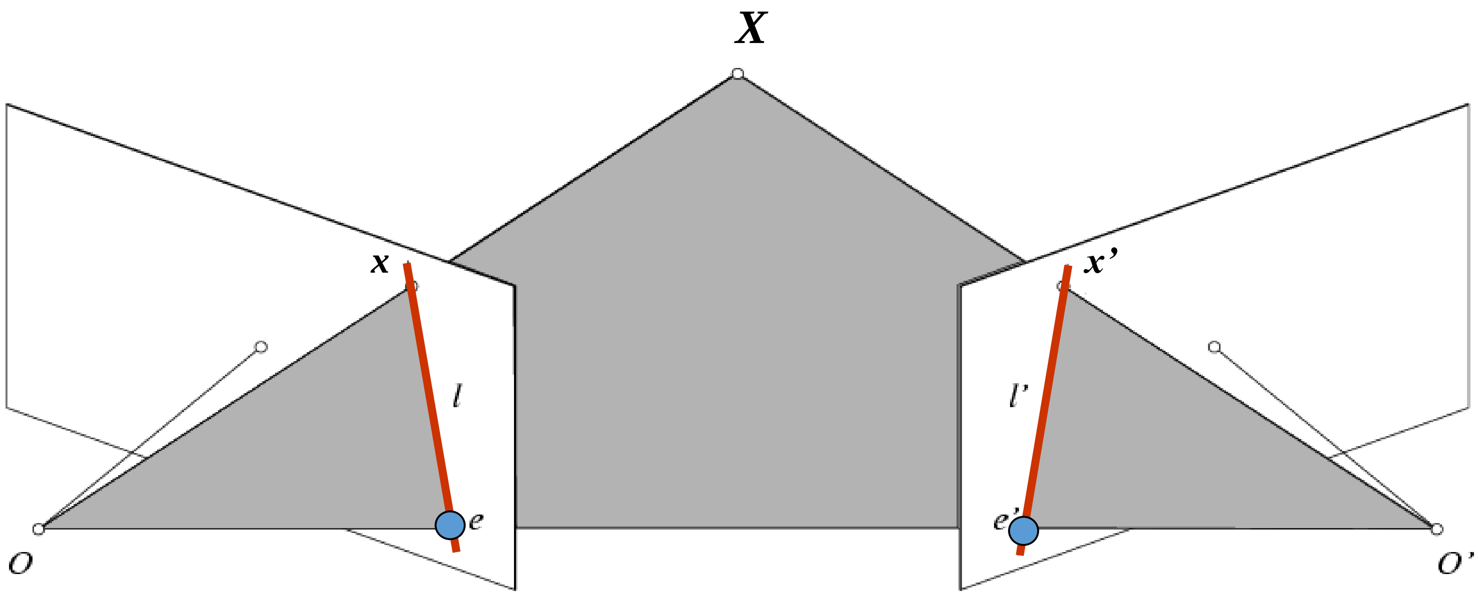
$$\hat{x}\cdot[t\times(R\hat{x}')]=0 \quad \Rightarrow \quad \hat{x}^T E\,\hat{x}'=0 \quad \text{with} \quad E=[t]_\times R$$

Drop ^ below to simplify notation

Skew -symmetric matrix

- $E\,x'$ is the epipolar line associated with x' ($l = E\,x'$)
- $E^T x$ is the epipolar line associated with x ($l' = E^T x$)
- $E\,e' = 0$ 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)

Recall: Properties of the Fundamental matrix



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

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

Recall: 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 corres pondences
  - 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.

Recall: 8-point algorithm

- Solve a system of homogeneous linear equations
  1. Write down the system of equations

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

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

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

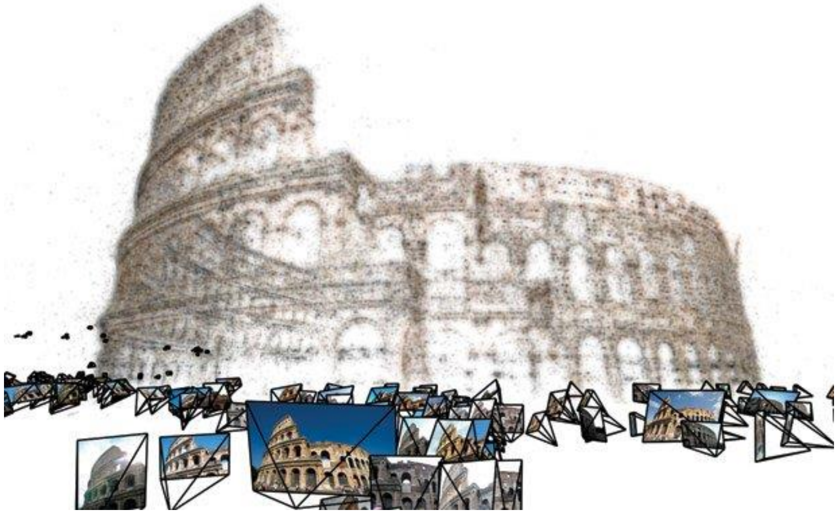
Recall: 8-point algorithm

- Solve a system of homogeneous linear equations
  1. Write down the system of equations
  2. Solve **f** from A**f**=**0** using SVD

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

- Resolve det(F) = 0 constraint using SVD

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

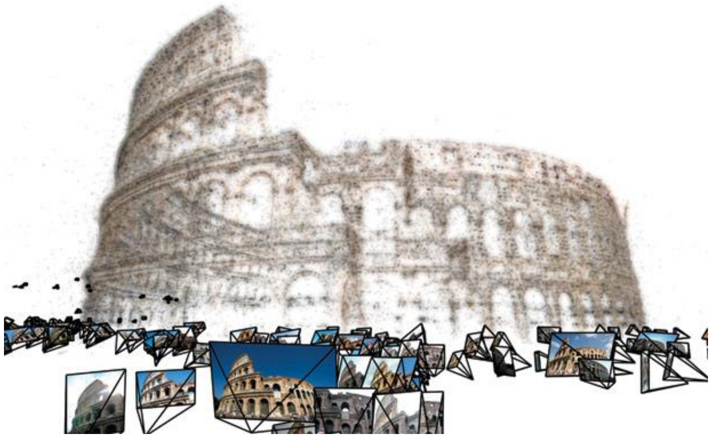


Structure from Motion

Chapter 7 in Szeliski

What is SfM?

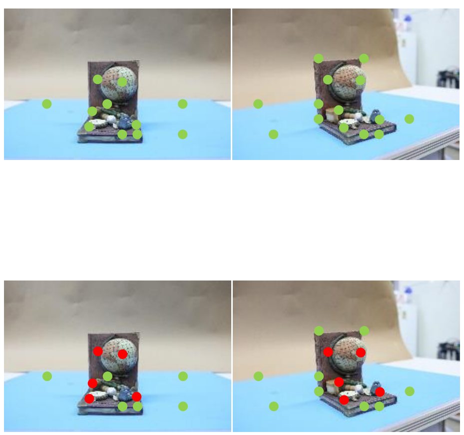
- **Structure from Motion (SfM)**
  - The process of **estimating three-dimensional structures from two-dimensional image sequences** which may be coupled with local motion signals.
  - Input: Freely taken images with overlapped scenery
  - Output: camera pose and 3D structure of the scene
  - Reference
    - http://photosynth.net
    - N.Snavely et al., “Photo Tourism: Exploring photo collections in 3D”, SIGGRAPH 2006

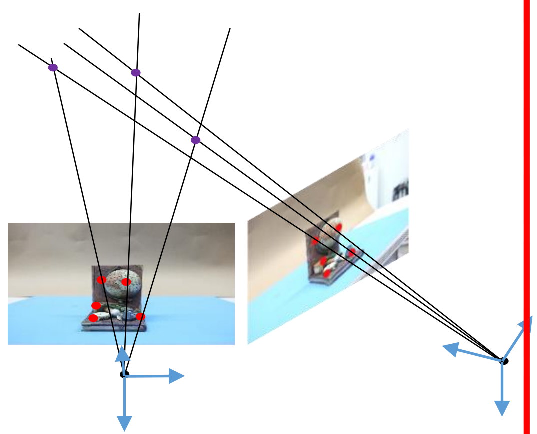


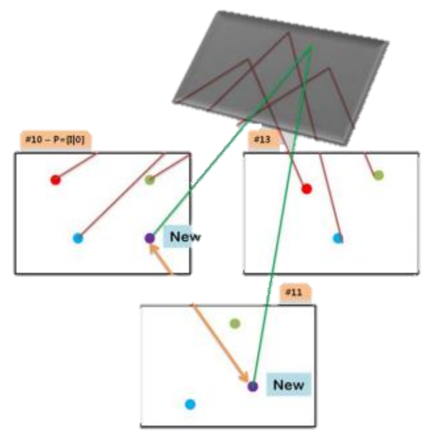
Overall Strategy

ToDo

Two images from dataset → **(Two-view) Initialization Step** → **(> Three-view) Growing Step** → **Bundle Adjustment** → **(> Three-view) Growing Step**



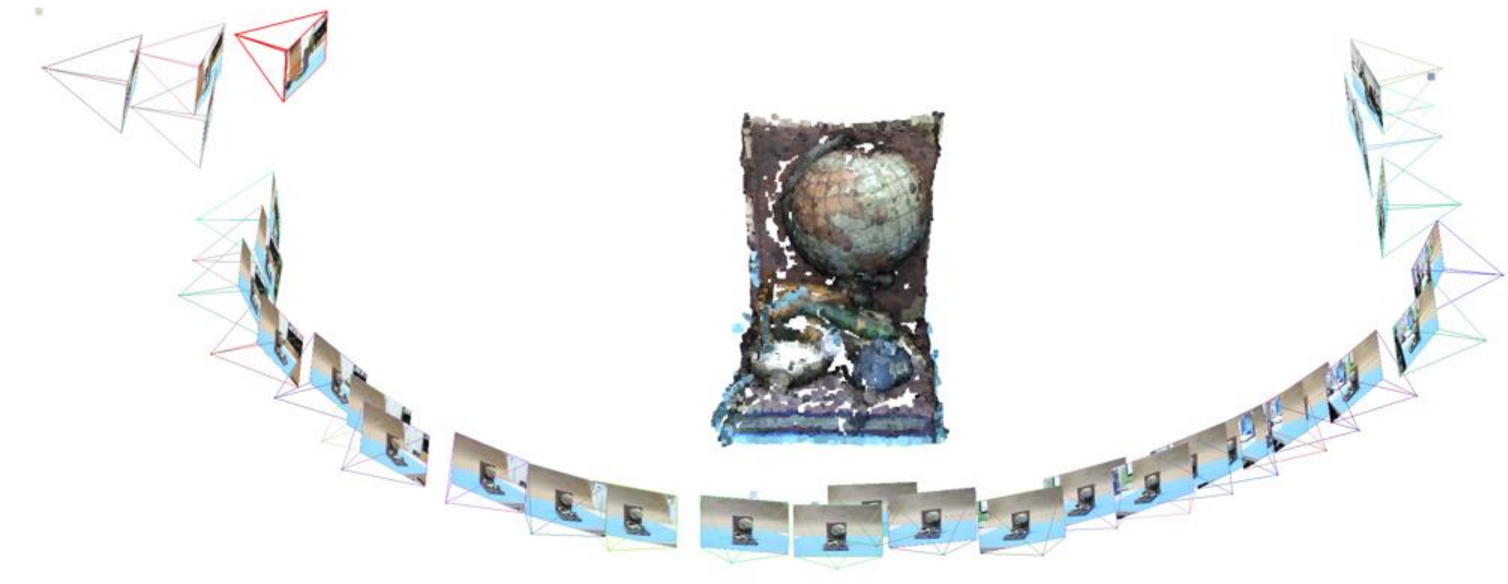




[2] Hartley, Richard, and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.
[3] Szeliski, Richard. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

Output

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



Overall Strategy

- **What are given?**
  - 1. **3D points** reconstructed from two images
  - 2. **2D keypoints** from each image corresponding to the reconstructed 3D points
  - 3. **Camera poses of two images**
  - Intrinsic matrix, remaining images(about 10 images)…etc

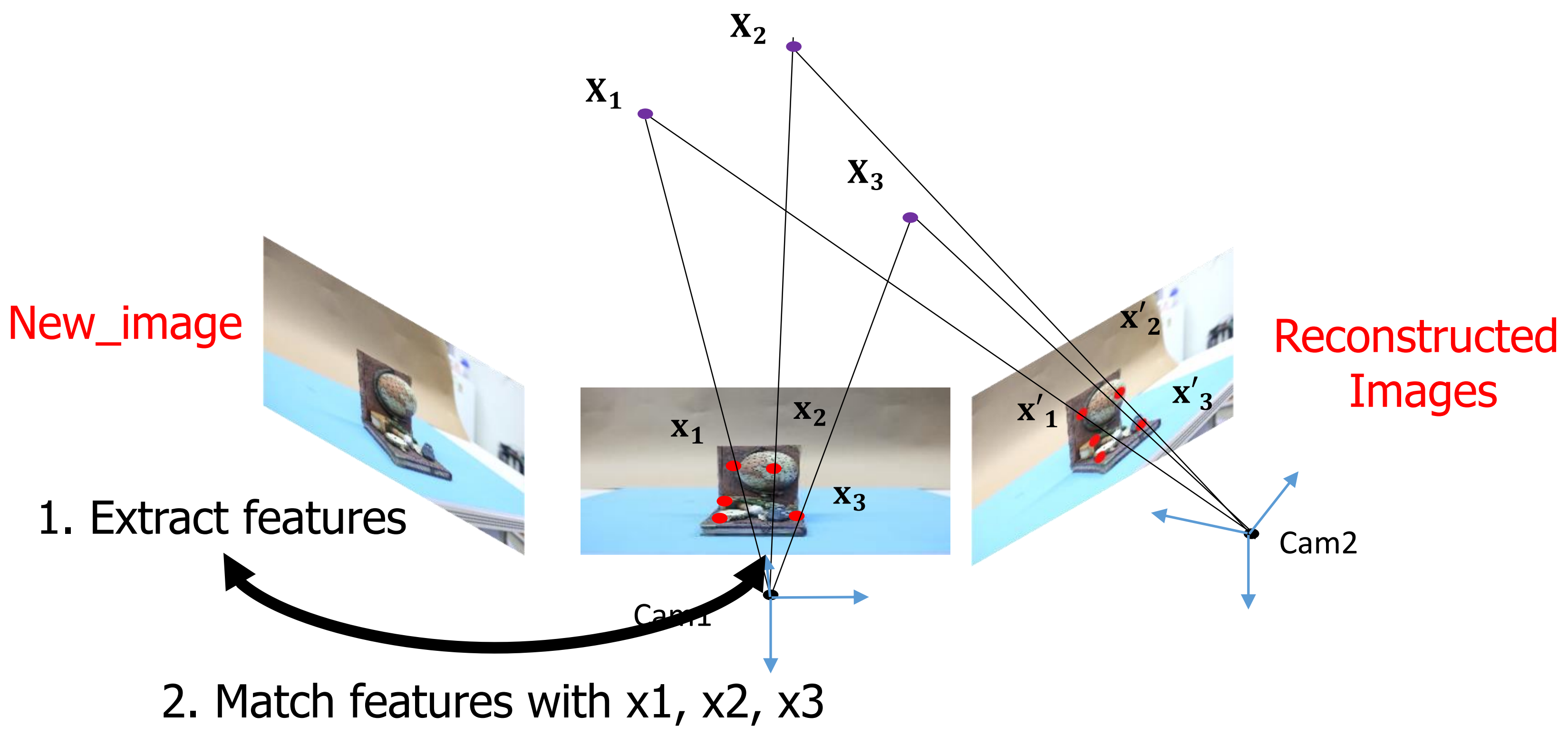
- **What have to be done?**
  - 1. **Repeatedly reconstruct remaining images** to the initialized model
  - 2. Do **Bundle Adjustment** to refine 3D reconstruction result.

Overall Strategy

- **1. Repeatedly reconstruct remaining image to the initialized model**
  - So called **Growing Step**
  - Algorithms are as follows:
    - 1. Select a image from remaining images whose the number of matched keypoints with reconstructed keypoints are largest.
    - 2. estimate a camera pose of a selected image using **3-point PnP RANSAC**
    - 3. reconstruct 3D points from keypoints from pose of the selected image and a pose of reconstructed image using **triangulation**
    - 4. repeat 1~3 until every images is included

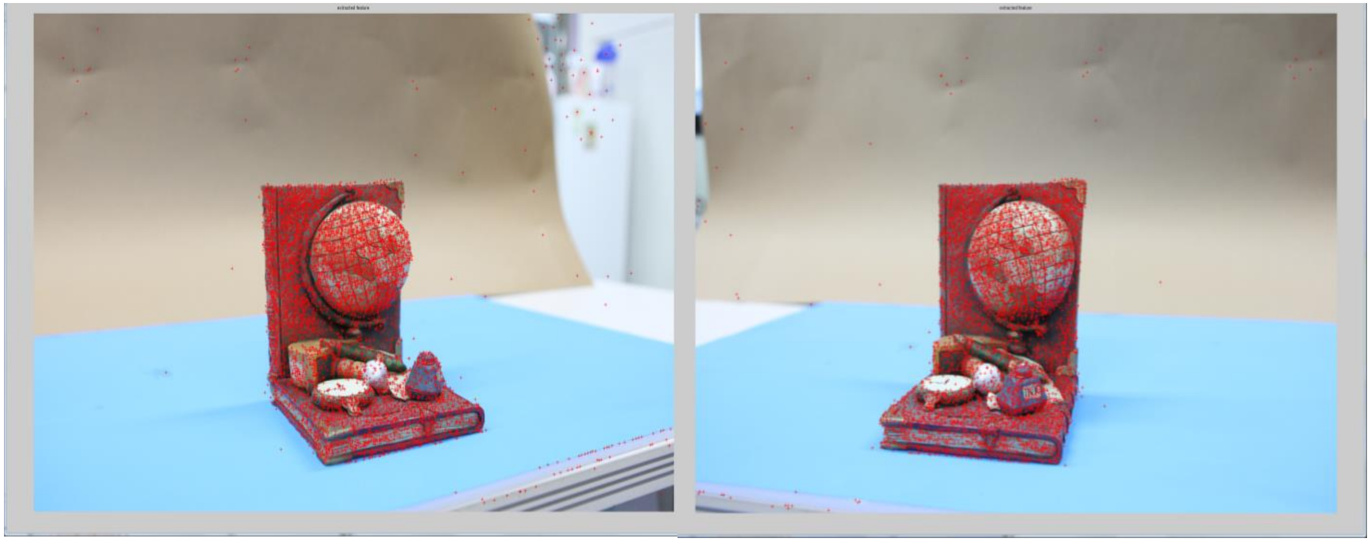
Overall Strategy

- **1-1. Select a image from remaining images whose the number of matched keypoints with reconstructed keypoints are largest.**



Overall Strategy

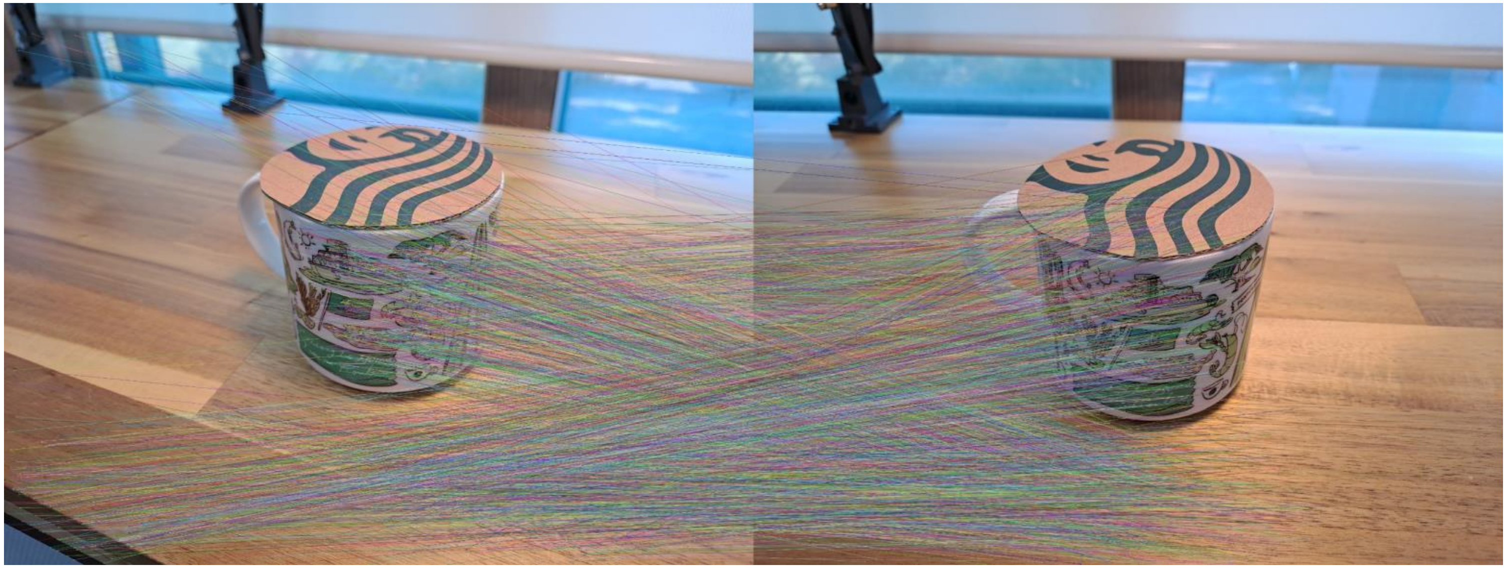
- **Feature extraction**
  - **cv2.SIFT_create().detectAndCompute**
  - Extract keypoints and descriptors from a image (red points on images)



Feature Extraction

Overall Strategy

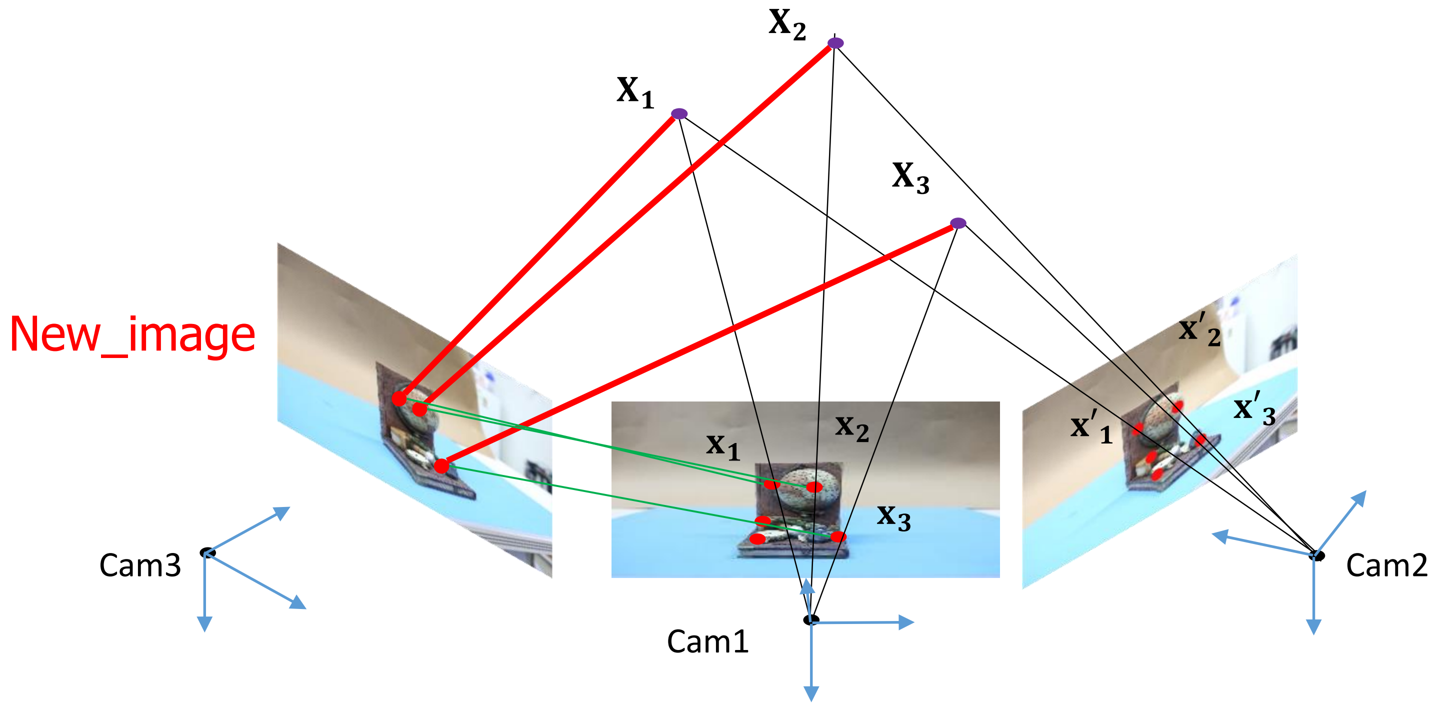
- **Feature matching**
  - **cv2.BFMatcher().knnMatch**
    - Match keypoints according to descriptors between two images



Feature matching

Overall Strategy

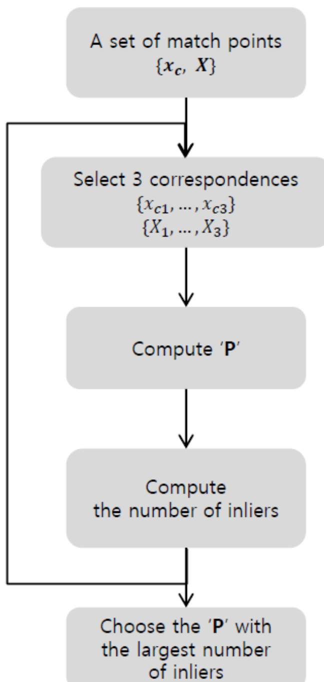
- **1-2. estimate a camera pose of a selected image using 3-point PnP RANSAC**
  - Estimate Camera matrix 'P'(Cam3) given a set of match points(red lines) and 3D {x, X}
  - Use the functions '**PerspectiveThreePoint**' or '**cv2.solveP3P**'

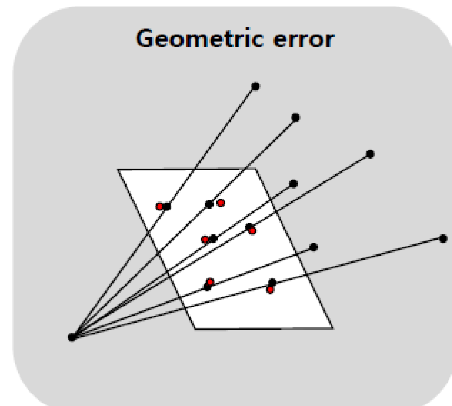


Overall Strategy

- **1-2. estimate a camera pose of a selected image using 3-point PnP RANSAC**
  - The output rotation vector is not 3x3 matrix, it is in Rodrigues form.
  - You should translate it into 3x3 matrix using **cv2.Rodrigues**

- **1-2. estimate a camera pose of a selected image using 3-point PnP RANSAC**
  - Compute the number of inliers
  - Choose the best **P** with the largest number of inliers





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

Pixel distance

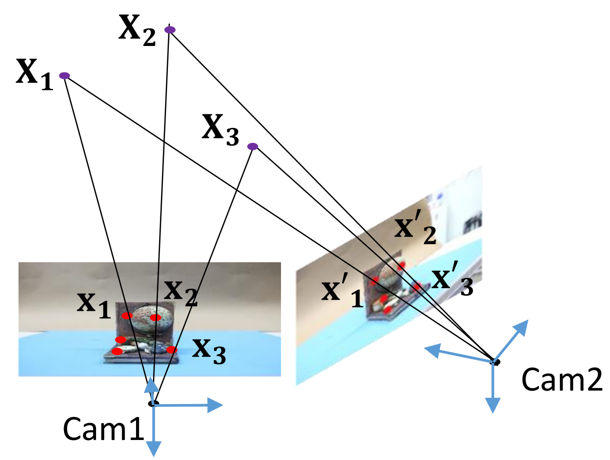
$d < t$ pixels

Set proper threshold

Overall Strategy

- **1-3. reconstruct 3D points from keypoints from the selected image using triangulation**
  - Reconstruct 3d points of matched keypoints using two camera poses

$$A \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \mathbf{0} \quad 0 = \begin{bmatrix} 0 & -1 & y \\ 1 & 0 & -x \\ -y & x & 0 \end{bmatrix} \begin{bmatrix} - & \mathbf{p}^{1T} & - \\ - & \mathbf{p}^{2T} & - \\ - & \mathbf{p}^{3T} & - \end{bmatrix} \mathbf{X}$$



$$A\mathbf{X} = \begin{bmatrix} x(\mathbf{p}_3^T) - \mathbf{p}_1^T \\ y(\mathbf{p}_3^T) - (\mathbf{p}_2^T) \\ x'(\mathbf{p}_3'^T) - \mathbf{p}_1'^T \\ y'(\mathbf{p}_3'^T) - (\mathbf{p}_2' T) \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \mathbf{0}$$

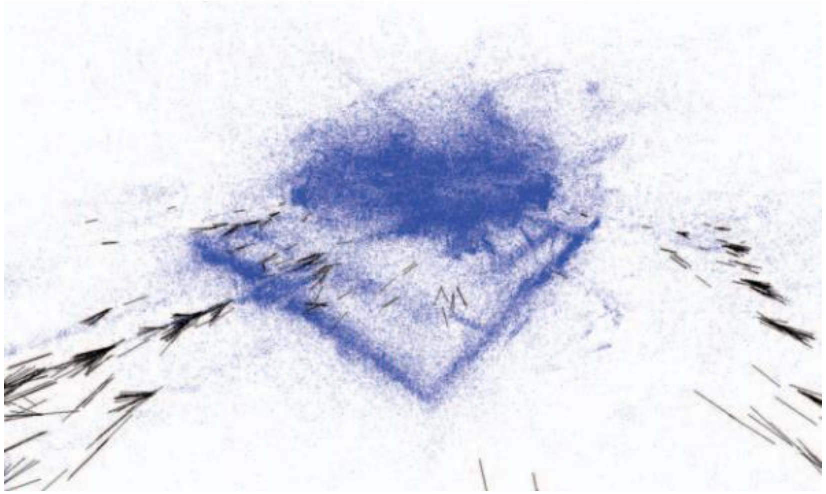
See this link for more explanation

Overall Strategy

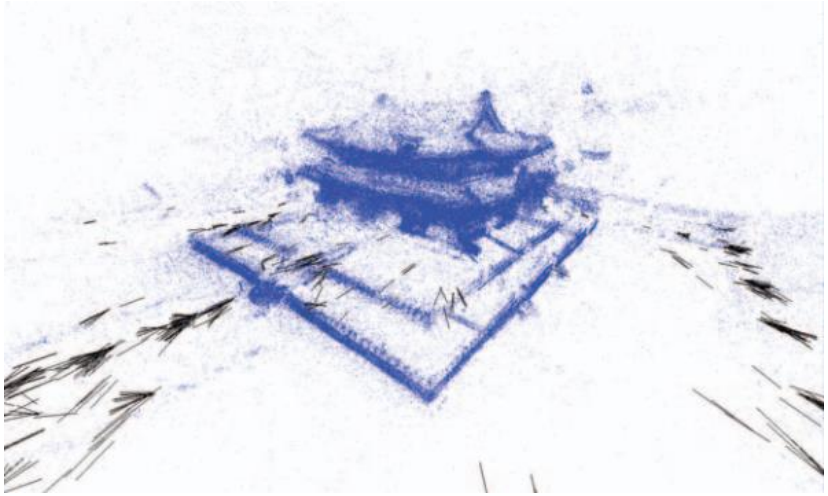
- **1-4. repeat 1-1~1-3 until every images is included**

Overall Strategy

- **2. Do Bundle Adjustment to refine 3D reconstruction**
  - 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 camera center.



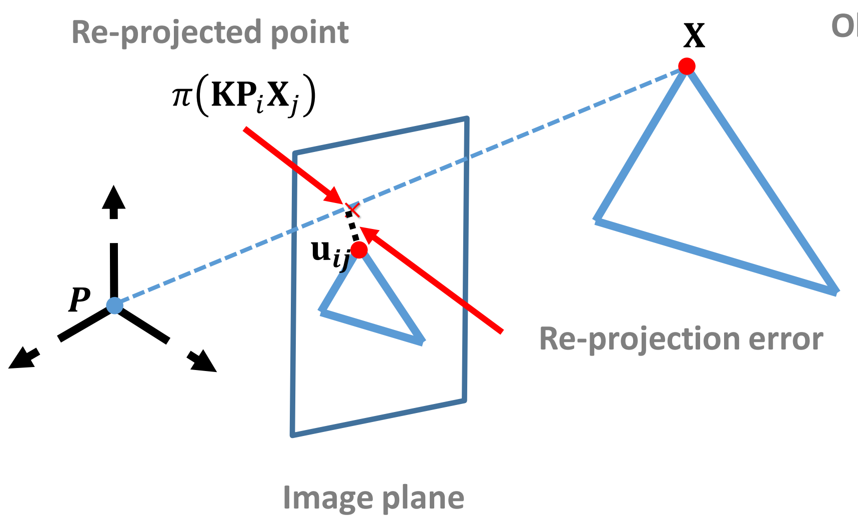
Before Bundle adjustment



After Bundle adjustment

[6] Triggs, Bill, et al. "Bundle adjustment—a modern synthesis." *International workshop on vision algorithms*. Springer Berlin Heidelberg, 1999.
[7] Jeong, Yekeun, et al. "Pushing the envelope of modern methods for bundle adjustment." *IEEE transactions on pattern analysis and machine intelligence* (2012): 1605-1617.

- Bundle Adjustment's Mathematical Problem
  - **Minimize re-projection error** manipulating 3D points and Camera poses
  - Non-linear Least Square approach



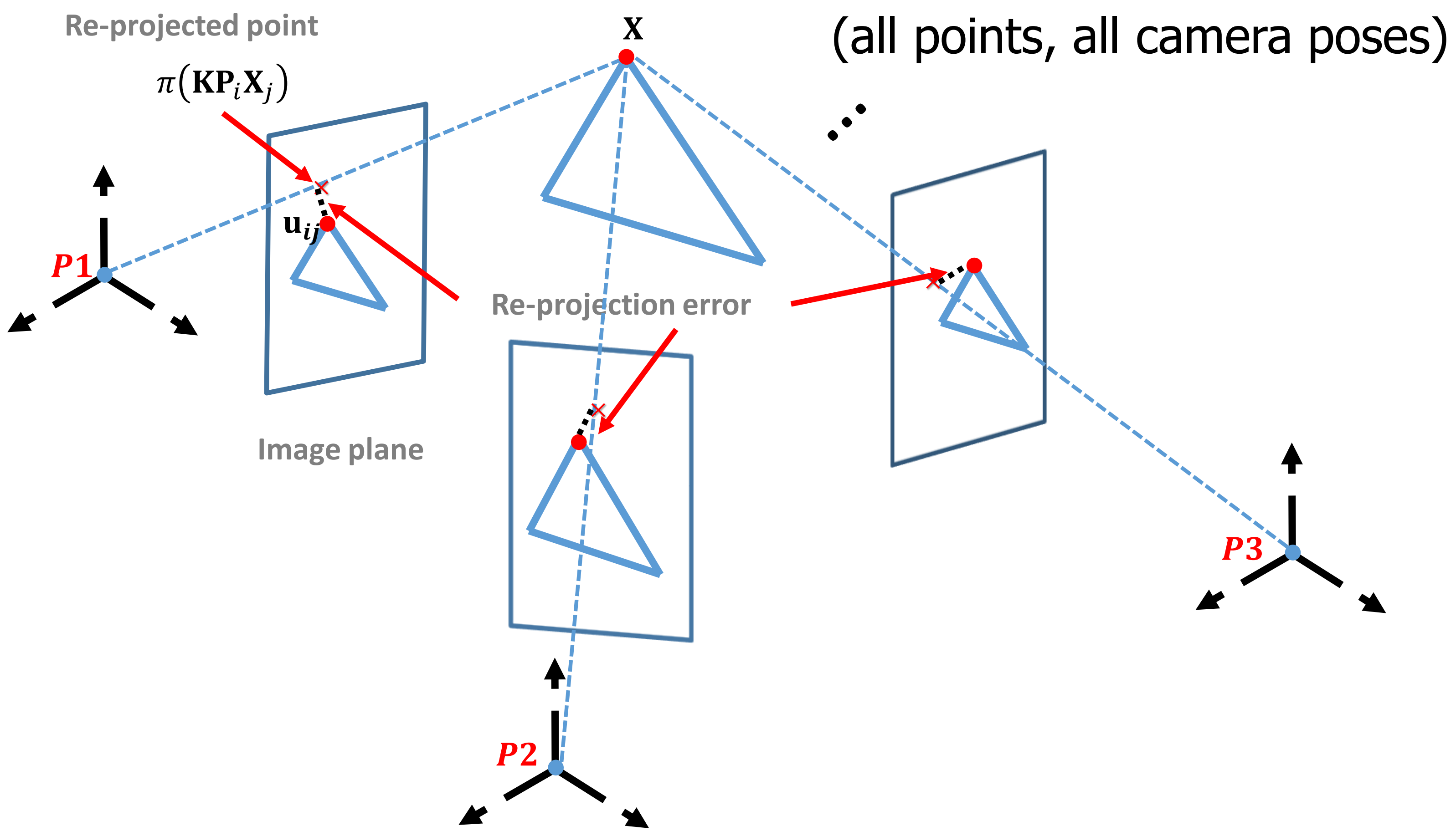
Objective function

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

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

Step VI. Optimization

- Project all 3D points to its visible image plane



Step VI. Optimization

- **2. Do Bundle Adjustment to refine 3D reconstruction**
  - JacobiancostE (Partially Provided, Matlab)
    - Calculates reprojection error and jacobian matrix
  - Jacobian Matrix
    - 2 X 6 matrix whose (i,j) element is partial differentiation between 2d point and certain variable

$$J = \begin{bmatrix} \frac{\partial f(R(q), C, X)}{\partial q} & \frac{\partial f(R(q), C, X)}{\partial C} & \frac{\partial f(R(q), C, X)}{\partial X} \end{bmatrix}$$

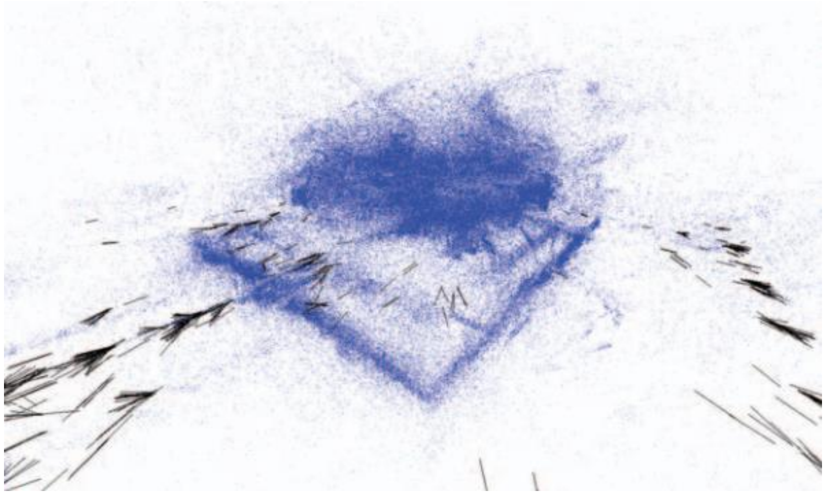
    - f is a function to project X using camera pose(R and C) to a image plane

Step VI. Optimization

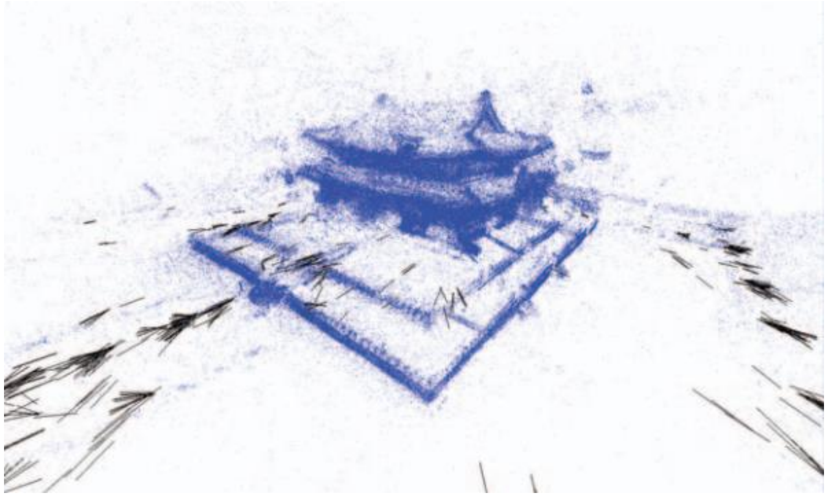
- **2. Do Bundle Adjustment to refine 3D reconstruction**
  - JacobiancostE (Partially Provided, Matlab)
    - A section that calculate reprojection error will be blank
    - **fill the blank area** to calculate reprojection error
    - A section that calculate jacobian matrix and optimization(LM2_iter_dof) **will be fully provided**
    - You have to include **understanding about jacobian matrix and optimization** in the report.

Step VI. Optimization

- Tips for Bundle Adjustment
  - How to check your BA module is working or not
    - Add some noise to 3D points and camera poses and do BA



Before Bundle adjustment

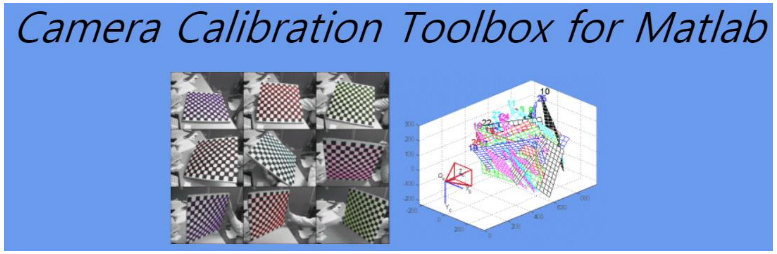


After Bundle adjustment

[6] Triggs, Bill, et al. "Bundle adjustment—a modern synthesis." *International workshop on vision algorithms*. Springer Berlin Heidelberg, 1999.
[7] Jeong, Yekeun, et al. "Pushing the envelope of modern methods for bundle adjustment." *IEEE transactions on pattern analysis and machine intelligence* (2012): 1605-1617.

Step VII. Camera calibration (Optional, no points)

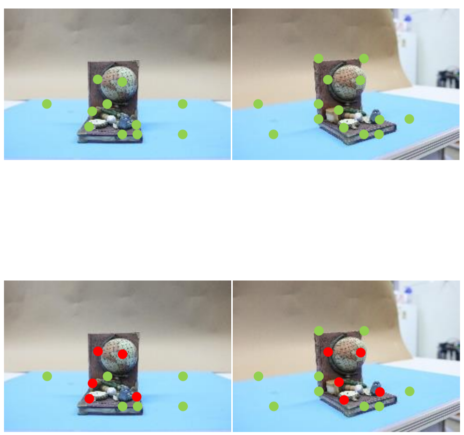
- Procedure
  - Download camera calibration toolbox
  - Print checkerboard
  - Capture multiple images of checkerboard
  - Run the camera calibration toolbox

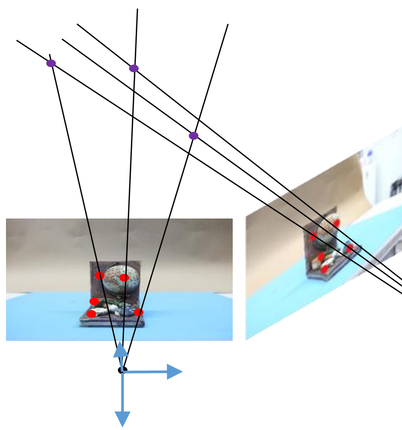


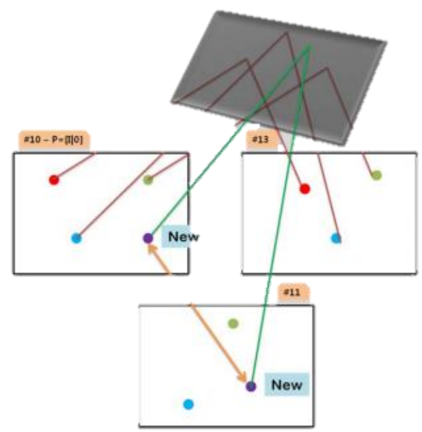
- Camera Calibration using Toolbox
  - Matlab instruction
    - http://www.vision.caltech.edu/bouguetj/calib_doc/index.html
  - C Library
    - http://docs.opencv.org/doc/tutorials/calib3d/camera_calibration/camera_calibration.html
  - Python
    - https://learnopencv.com/camera-calibration-using-opencv/

To Do List

Two images from dataset → **(Two-view) Initialization Step** → **(> Three-view) Growing Step** → **Bundle Adjustment** → **(> Three-view) Growing Step**







[2] Hartley, Richard, and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.
[3] Szeliski, Richard. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

To Do List

1. Select a view with the most matched features over the reference view
2. Implement 3 point **RANSAC** algorithm (4 pts)
3. Generate 3D point by implementing **Triangulation** (3 pts)
4. Bundle Adjustment (5 Pts)
   - Fill the blank of reprojection error part
   - Explain jacobian matrix and its optimization in your report
5. Make your own dataset and run your code (optional, 5 pts)
   - If you want to take your own image, please send me first two image input among them with corresponding calibrate parameters. Then I will give you the initial 3D model.

For Python Student

- **Not** allow to use OpenCV and Scipy functions except below :
  - cv2.SIFT_create (Step I)
  - cv2.xfeatures2d.SIFT_create (Step I)
  - cv2.findChessboardCorners (Step VII)
  - cv2.cornerSubPix (Step VII)
  - cv2.calibrateCamera (Step VII)
  - cv2.solveP3P (Step V)
  - Image I/O function
  - Image visualization function
  - If you think other functions are mandatory, email me

Provided files

- Below functions and files are provided:
  - Data Folder: contains a image dataset and a camera intrinsic file
  - Functions folder: contains functions about jacobian and optimization
    - costE: calculate re-projection error
    - JacobiancostE: calculate re-projection error and jacobian matrix
    - Deserialize: slicing input list into rotation, translation and 3D points
    - JacobianPoint: calculate jacobian matrix with respect to 3D points
    - jacobianPose2: calculate jacobian matrix with respect to camera poses
    - RotationVector_to_RotationMatrix
    - PerspectiveThreePoint: three-points algorithm function

Provided files

- Below functions and files are provided:
  - two_view_recon_info: contains files about initial two view reconstruction
    - Initial two images are **sfm03.jpg and sfm04.jpg**
    - sfm03_keypoints.npy and sfm04_keypoints.npy: keypoints of sfm03 and sfm04 images
    - sfm03_descriptors.npy and sfm04_descriptors.npy: descriptors of sfm03 and sfm04 images
    - sfm03_matched_idx.npy and sfm04_matched_idx.npy: indices of matched keypoints
    - sfm03_camera_pose.npy and sfm04_camera_pose.npy: camera poses of two images
    - inlinear.npy: indices of matched points which are reconstructed in 3D
    - 3D_points.npy: reconstructed 3D points

SfM with your own dataset

- Tips for making your own dataset (using your mobile phone)

  1) Use a fixed-focus camera.

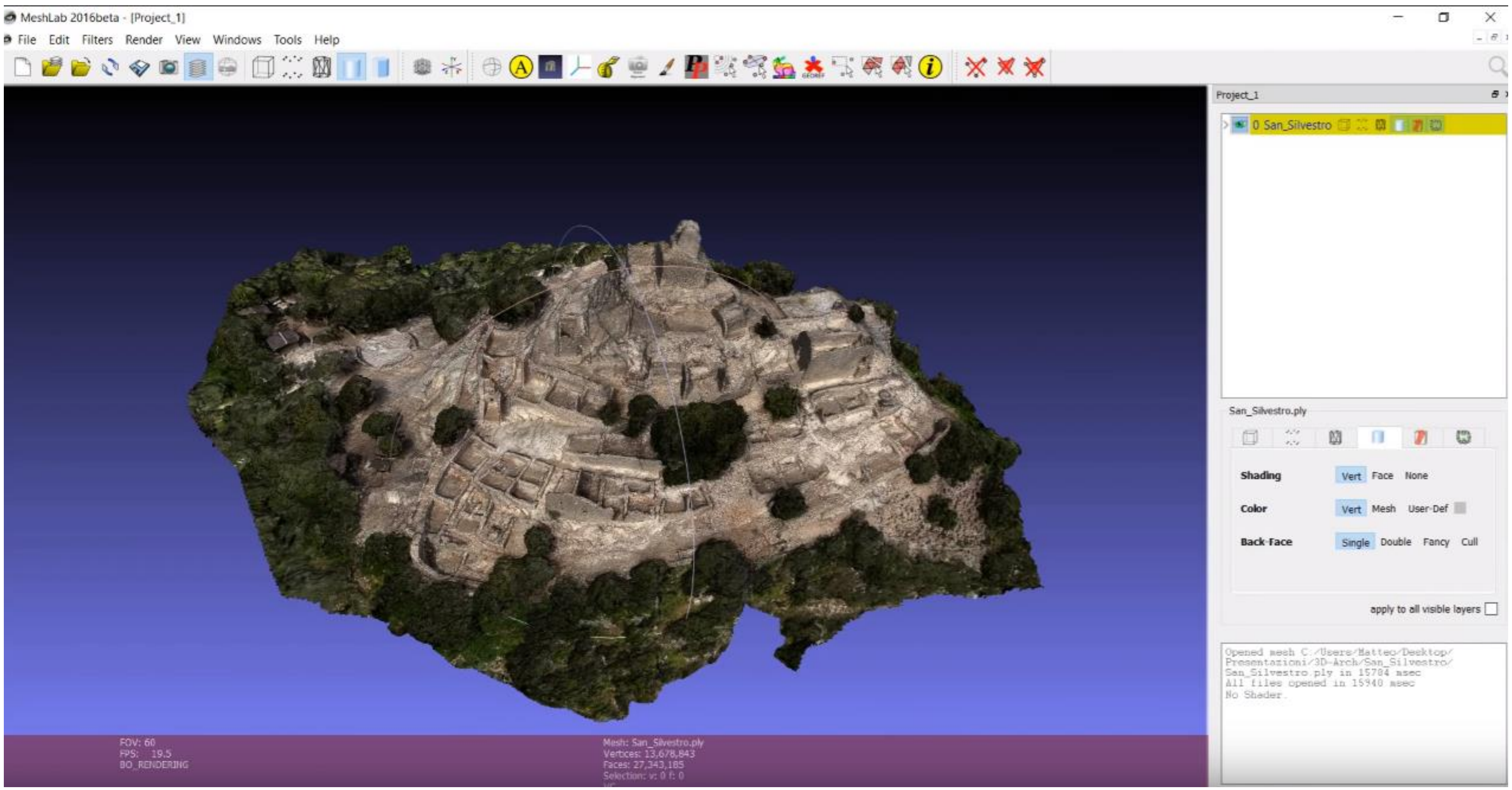
  2) Do calibration for camera parameter estimation using toolbox (to get intrinsic).

  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.

Display your 3D results (ply file)

Use Meshlab (download: http://www.meshlab.net)



Submission

- Submission should include…
  - Source code
  - Results (3D point cloud in ply file) of your code
  - Readme file explaining how to execute the program
  - **Report** (3pts)
- Report should include…
  - Your understanding of each steps of algorithms
  - Figures of results from your implementation
  - Understanding of jacobian matrix and optimization
- Notice
  - [Delayed submission] Not allowed
  - [Plagiarism] Definitely F grade for copied codes (from friends or internet)
  - [Implementation] No use any open function such as findFundamentalMat() other than the mentioned function

Due : Nov. 4, 11:55PM
To : LMS System

TA session: 10/30 and 11/1
TA: Seongmin Lee
Office Hour : Thu, Fri PM 2:30~3:30
(e-mail: sungmin9939@gm.gist.ac.kr)

Auxiliary References

- Multiple-View Geometry in Computer Vision (https://github.com/liulinbo/slam/blob/master/Multiple%20View%20Geometry%20in%20Computer%20Vision.pdf)
  - Basic Projective Geometry (ch. 2,3)
  - Camera Models and Calibration (ch. 6,8)
  - Epipolar Geometry and Implementation (ch. 9, 11)
  - Triangulation (ch. 12)

- 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