

Introduction to 3D Vision

: A Tutorial for Everyone

Sunglok Choi, Senior Researcher

sunglok@etri.re.kr | <http://sites.google.com/site/sunglok>

Electronics and Telecommunication Research Institute (ETRI)

The ETRI logo is located on the left side of the slide. It consists of a stylized orange 'R' and blue 'T' characters, with a blue vertical bar below them.

An Invitation ~~Introduction~~ to 3D Vision : A Tutorial for Everyone

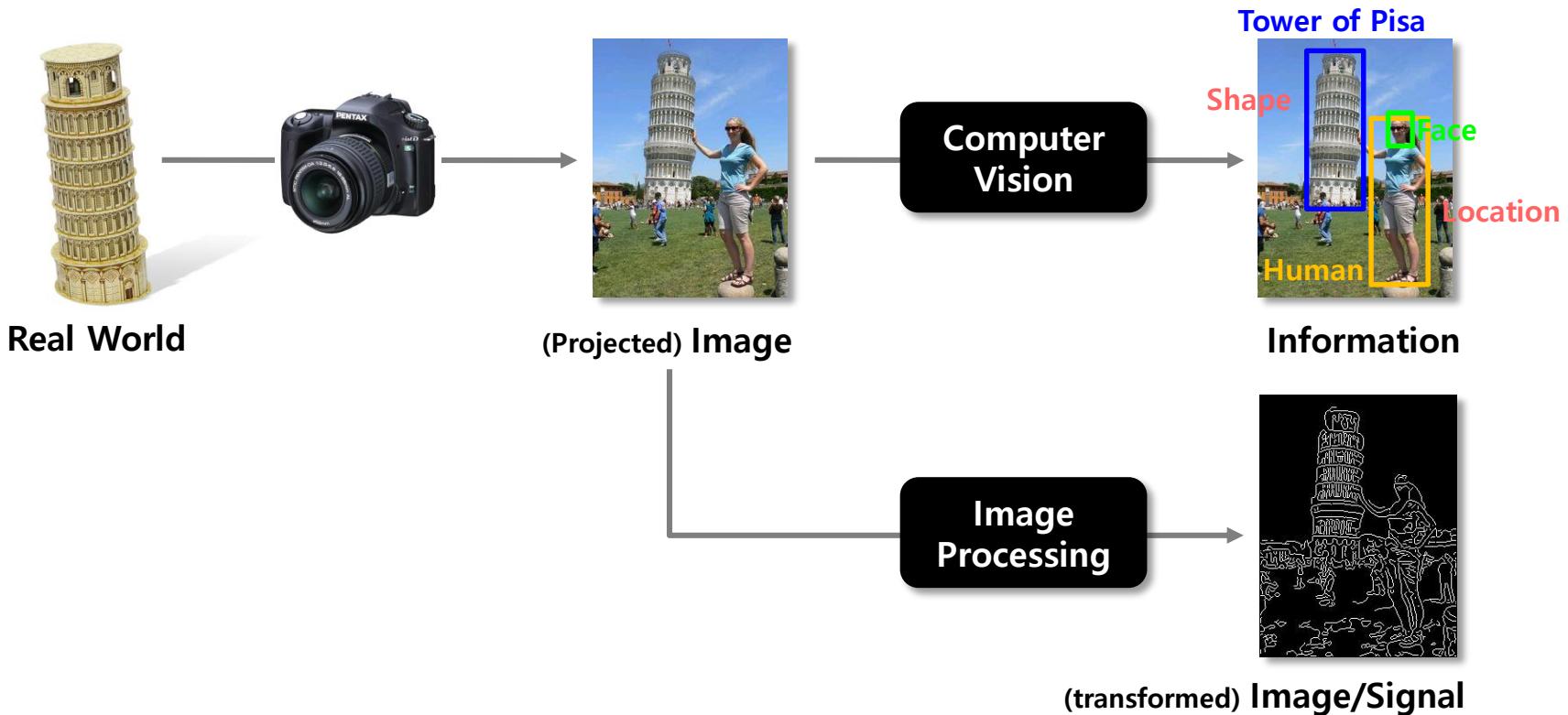
Sunglok Choi, Senior Researcher

sunglok@etri.re.kr | <http://sites.google.com/site/sunglok>

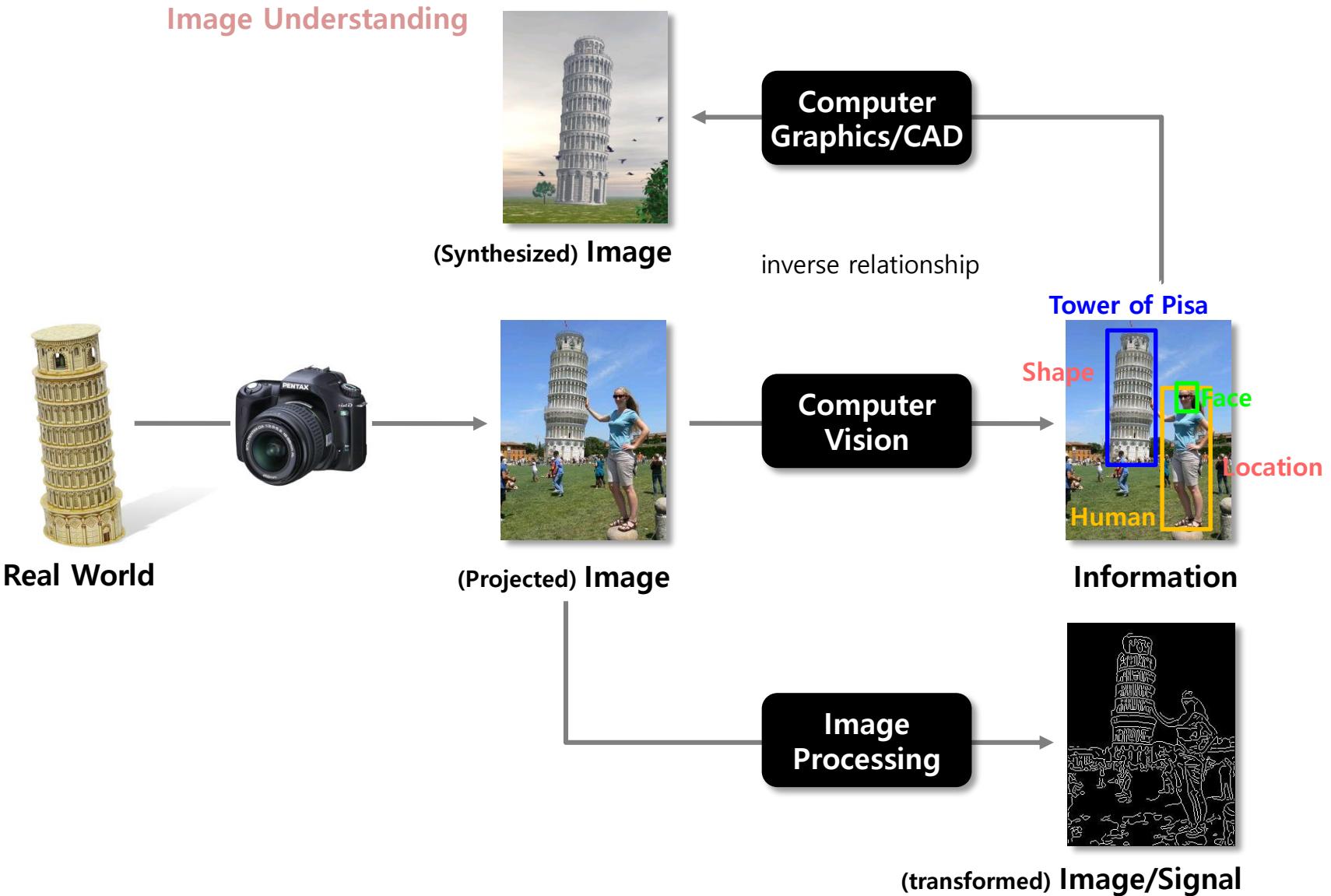
Electronics and Telecommunication Research Institute (ETRI)

What is Computer Vision?

Image Understanding



What is Computer Vision?



What is 3D Vision?

Computer Vision

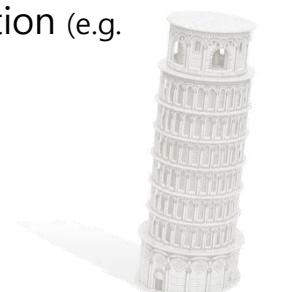
What is it?

- Label (e.g. Tower of Pisa)
- Shape (e.g.)



Where is it?

- Place (e.g. Piazza del Duomo, Pisa, Italy)
- Location (e.g.)



(84, 10, 18) [m]



What is 3D Vision?

Computer Vision

What is it?

- **Label** (e.g. Tower of Pisa)
- **Shape** (e.g.)



Where is it?

- **Place** (e.g. Piazza del Duomo, Pisa, Italy)
- **Location** (e.g.)



(84, 10, 18) [m]



Recognition Problems v.s. Reconstruction Problems

What is 3D Vision?

Visual Geometry (Multiple View Geometry)

Geometric Vision

Computer Vision

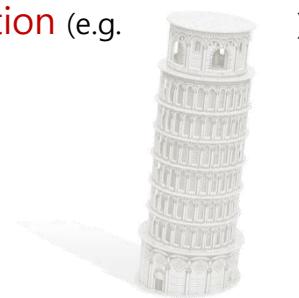
What is it?

- **Label** (e.g. Tower of Pisa)
- **Shape** (e.g.)



Where is it?

- **Place** (e.g. Piazza del Duomo, Pisa, Italy)
- **Location** (e.g.)



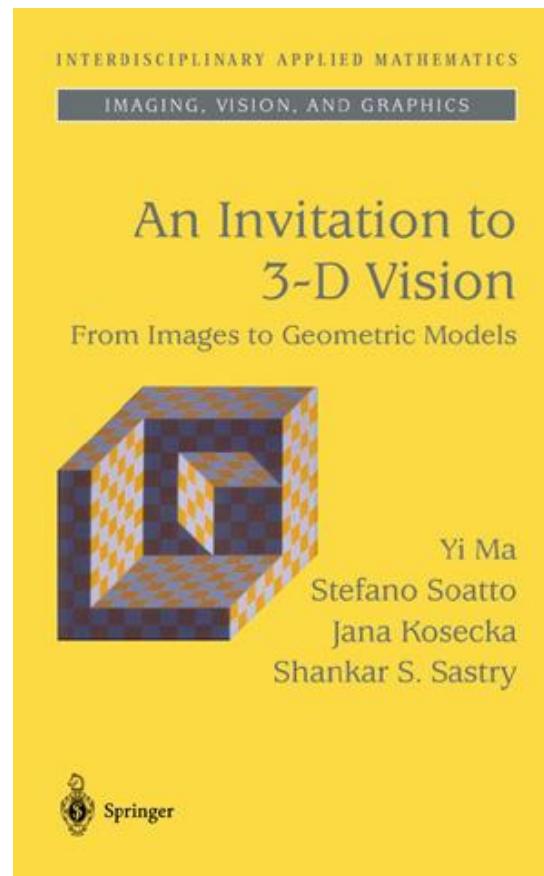
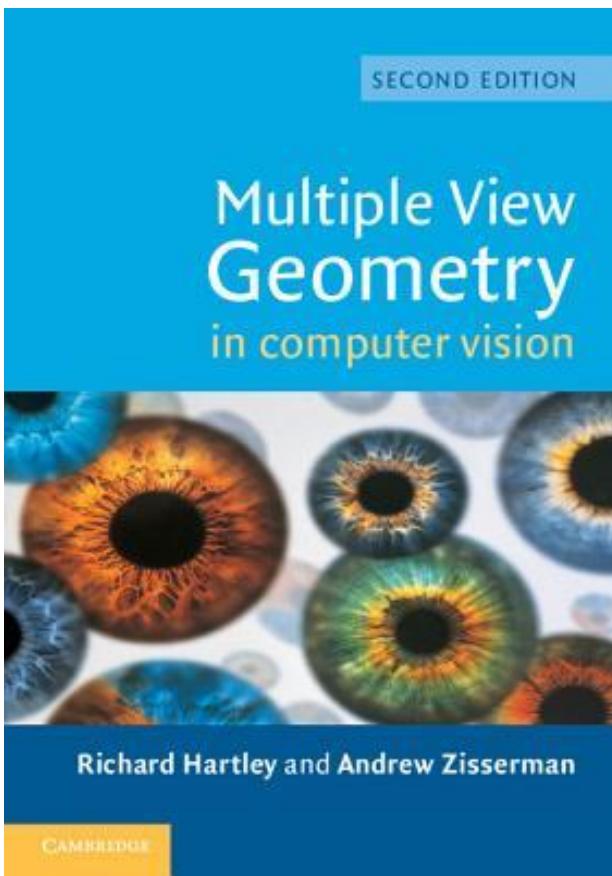
(84, 10, 18) [m]



Recognition Problems v.s. Reconstruction Problems

What is 3D Vision?

- Reference Books

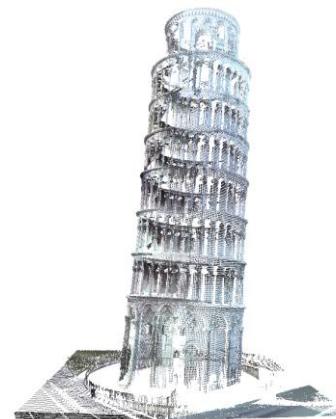




What is 3D Vision?

- OpenCV (Open Source Computer Vision)
 - calib3d Module: Camera Calibration and 3D Reconstruction
 - OpenCV API Reference
 - Introduction
 - core. The Core Functionality
 - imgproc. Image Processing
 - imgcodecs. Image file reading and writing
 - videoio. Media I/O
 - highgui. High-level GUI and Media I/O
 - video. Video Analysis
 - calib3d. Camera Calibration and 3D Reconstruction
 - features2d. 2D Features Framework
 - objdetect. Object Detection
 - ml. Machine Learning
 - flann. Clustering and Search in Multi-Dimensional Spaces
 - photo. Computational Photography
 - stitching. Images stitching
 - cuda. CUDA-accelerated Computer Vision
 - cudaarithm. CUDA-accelerated Operations on Matrices
 - cudabgsegm. CUDA-accelerated Background Segmentation
 - cudacodec. CUDA-accelerated Video Encoding/Decoding
 - cudafeatures2d. CUDA-accelerated Feature Detection and Description
 - cudafilters. CUDA-accelerated Image Filtering
 - cudaimproc. CUDA-accelerated Image Processing
 - cudaoptflow. CUDA-accelerated Optical Flow
 - cudastereo. CUDA-accelerated Stereo Correspondence
 - cudawarping. CUDA-accelerated Image Warping
 - shape. Shape Distance and Matching
 - superres. Super Resolution
 - videotab. Video Stabilization
 - viz. 3D Visualizer

What is 3D Vision?



point cloud, depth/range image, polygon mesh, ...



Perspective Camera

3D Vision

v.s.

3D Data Processing



RGB-D Camera
(Stereo, Structured Light, ToF, Light Field)



Omni-directional Camera

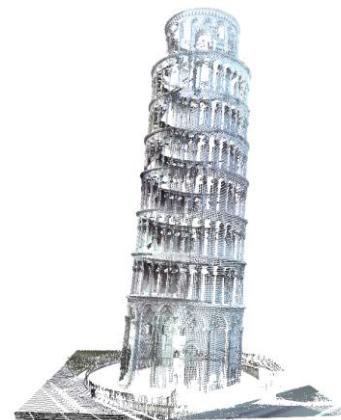


Range Sensor
(LiDAR, RADAR)

What is 3D Vision?



image



point cloud, depth/range image, polygon mesh, ...



Perspective Camera

3D Vision

v.s.

3D Data Processing



RGB-D Camera
(Stereo, Structured Light, ToF, Light Field)



Omni-directional Camera



Range Sensor
(LiDAR, RADAR)

Applications: Photo Browsing

- [Photo Tourism](#) (2006)



Applications: Large-scale 3D Reconstruction

- [Building Rome in a Day](#) (2009)



photo by [Ceci And Brandon](#)



2106 images, 819242 points
[click here to see more views](#)



photo by [angle_in_soul](#)



1381 images, 499044 points
[click here to see more views](#)



photo by [hon12gui1](#)



1294 images, 530076 points
[click here to see more views](#)



photo by [JBradshaw](#)



283 images, 230182 points
[click here to see more views](#)



Applications: Real-time Visual SLAM

- [ORB-SLAM](#) (2014)



Instituto Universitario de Investigación
en Ingeniería de Aragón
Universidad Zaragoza

ORB-SLAM2: an Open-Source SLAM System
for Monocular, Stereo and RGB-D Cameras

Raúl Mur-Artal and Juan D. Tardós

raulmur@unizar.es

tardos@unizar.es

Applications: Augmented Reality

- [PTAM: Parallel Tracking and Mapping](#) (2007)

Parallel Tracking and Mapping
for Small AR Workspaces

ISMAR 2007 video results

Georg Klein and David Murray
Active Vision Laboratory
University of Oxford

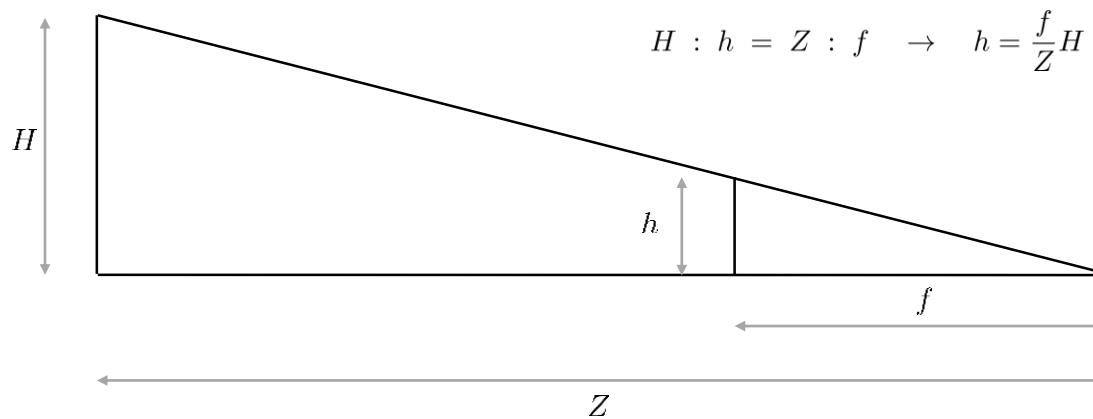


Table of Contents

- **What is 3D Vision?**
- **Single-view Geometry**
 - Camera Projection Model
 - General 2D-3D Geometry
- **Two-view Geometry**
 - Planar 2D-2D Geometry (**Projective Geometry**)
 - General 2D-2D Geometry (**Epipolar Geometry**)
- **Multi-view Geometry**
- **Correspondence Problem**
- **Summary**

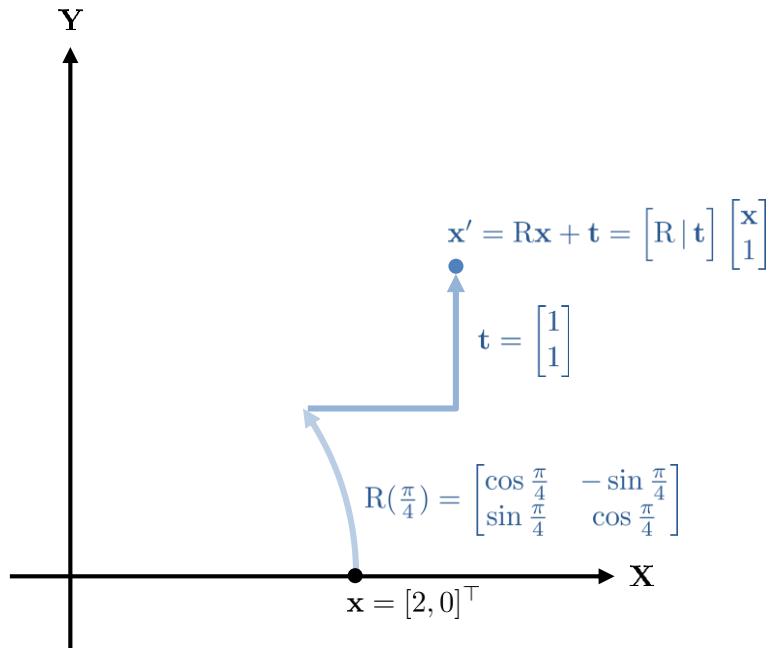
Getting Started with 2D

- Similarity



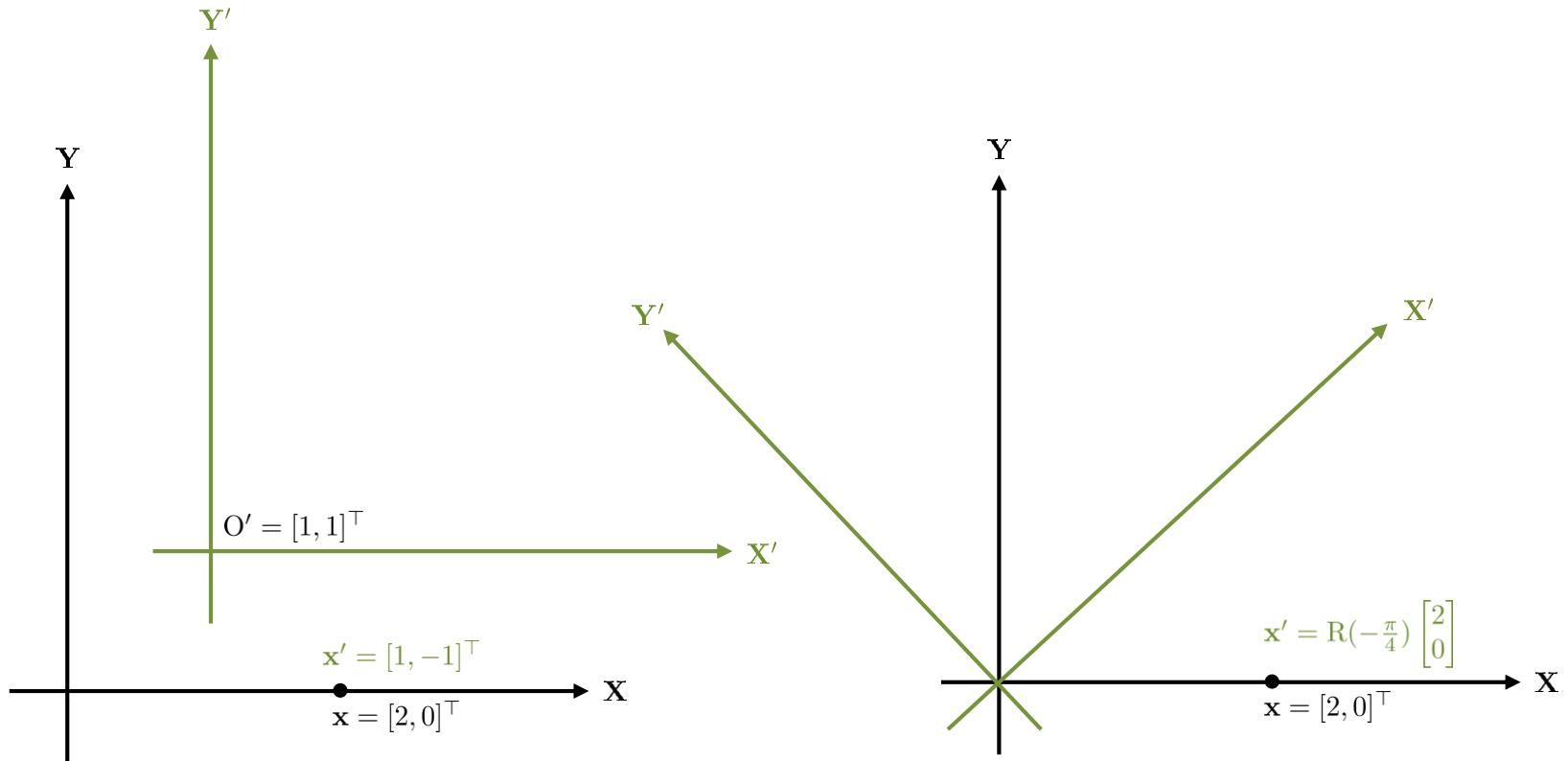
Getting Started with 2D

- Point transformation



Getting Started with 2D

- Coordinate transformation



Single-view Geometry



The Cyclops, gouache and oil by Odilon Redon

Single-view Geometry



Camera Projection Model
General 2D-3D Geometry

The Cyclops, gouache and oil by Odilon Redon

Camera Projection Model

- The Pinhole Camera Model



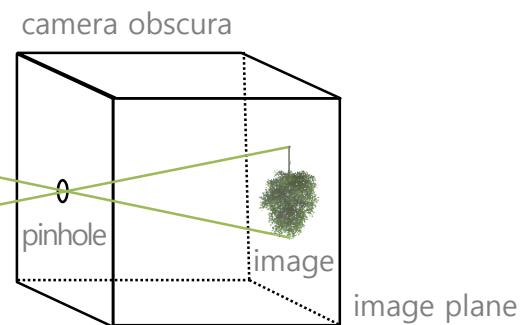
A large-scale camera obscura
at San Francisco, California



A modern-day camera obscura



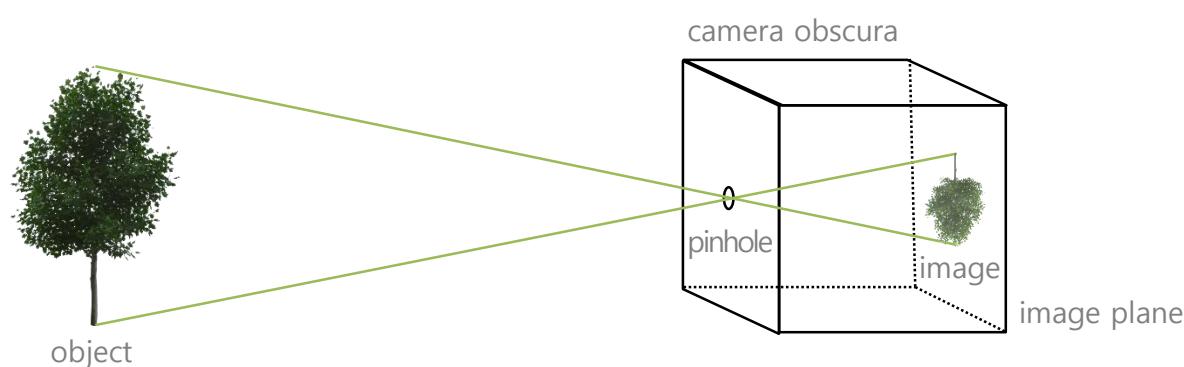
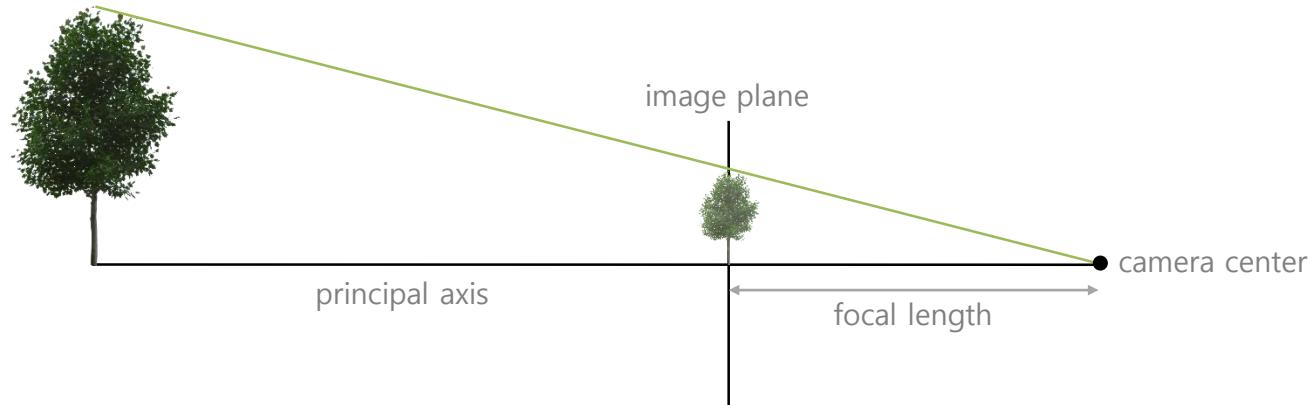
An Image in camera obscura
at Portslade, England



Camera Projection Model

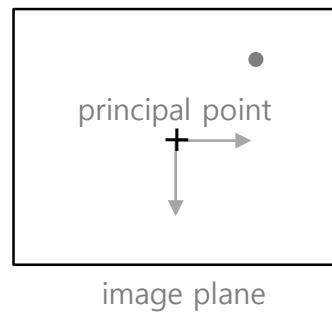
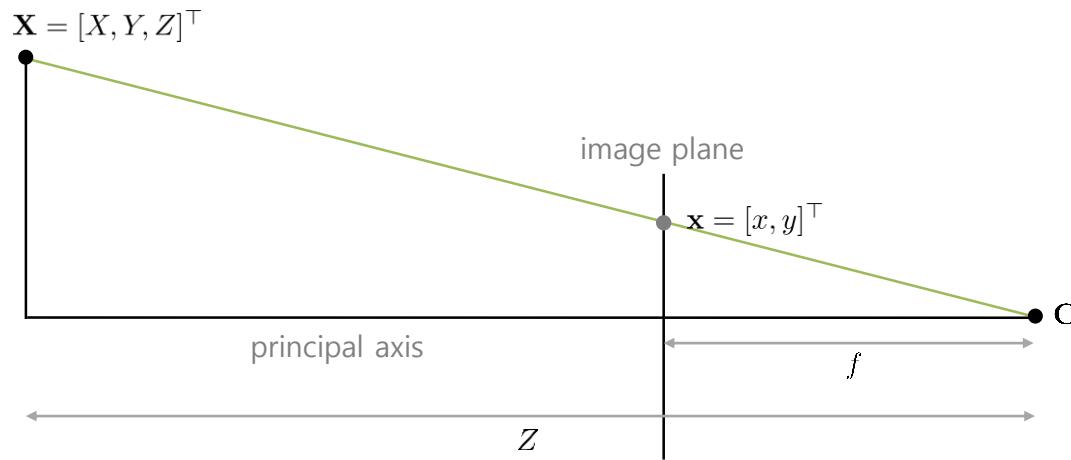
- The Pinhole Camera Model

c.f. conclusion in advance
 $\mathbf{x} = \mathbf{P}\mathbf{X}$ $(\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}])$



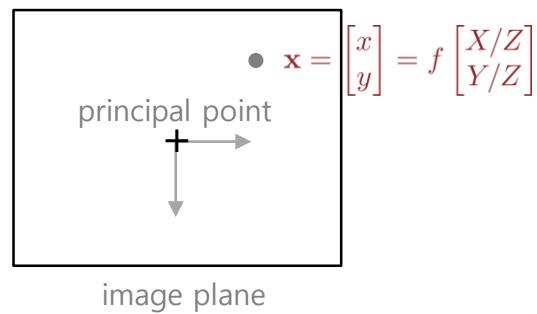
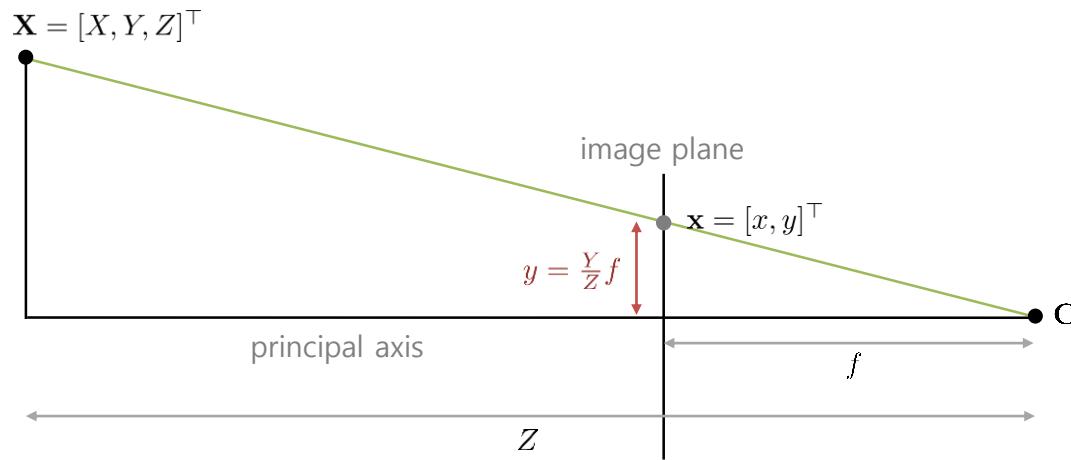
Camera Projection Model

- The Pinhole Camera Model



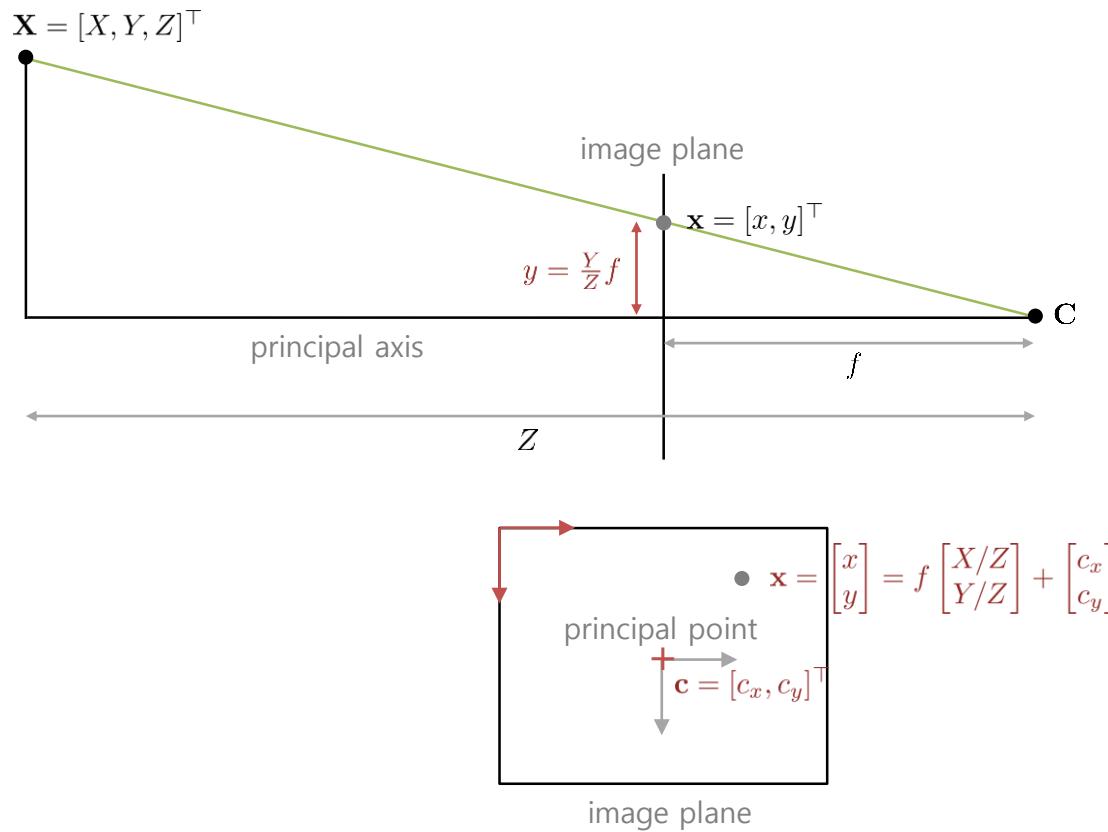
Camera Projection Model

- The Pinhole Camera Model



Camera Projection Model

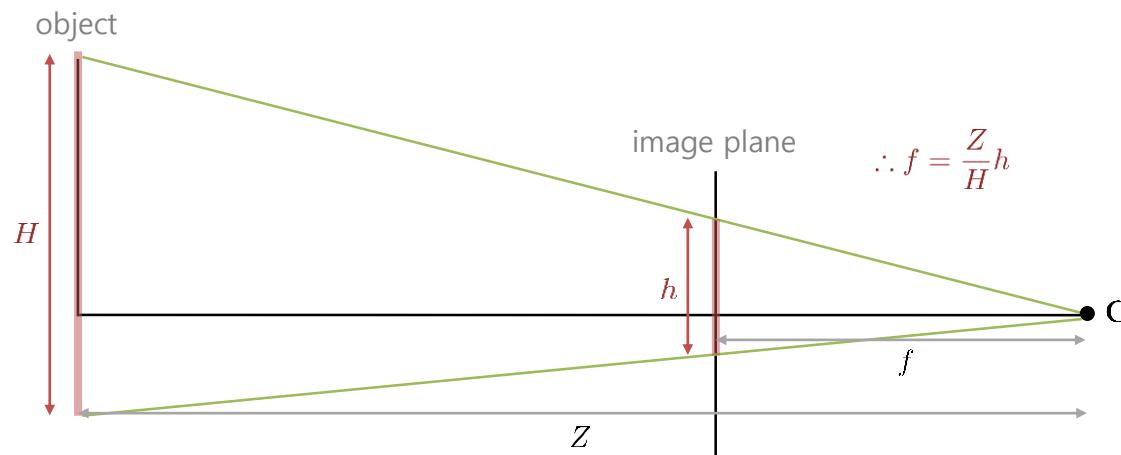
- The Pinhole Camera Model



Camera Projection Model

- **The Pinhole Camera Model**

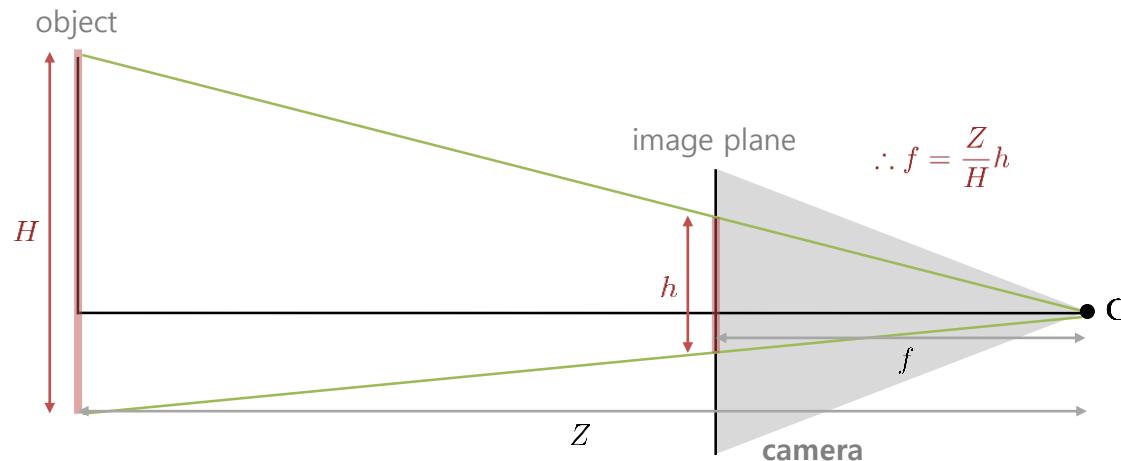
- Example: **Simple Camera Calibration #1**
 - Unknown: **Focal length** (unit: [pixel])
 - Assumptions
 - The object length and distance from the camera are known.
 - The object is aligned with the image plane.



Camera Projection Model

- The Pinhole Camera Model

- Example: Simple Camera Calibration #1
 - Unknown: **Focal length** (unit: [pixel])
 - Assumptions
 - The object length and distance from the camera are known.
 - The object is aligned with the image plane.



Camera Projection Model

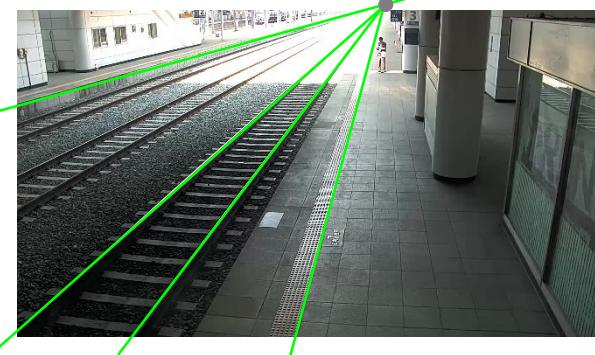
- **The Pinhole Camera Model**

- Example: **Simple Camera Calibration #2**

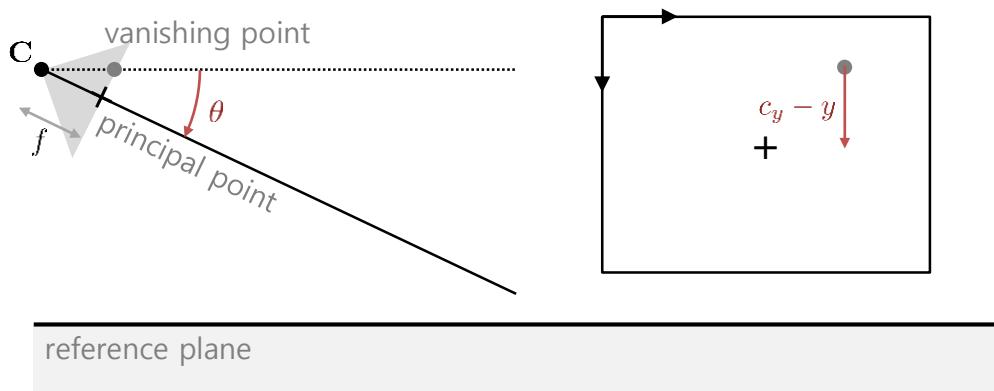
- Unknown: **Tilt angle** (unit: [rad])

- Assumptions

- The focal length is known.
 - The principal point is known or selected as the center of images.
 - A vanishing point from the reference plane is known.



$$\therefore \theta = \tan^{-1} \frac{c_y - y}{f}$$

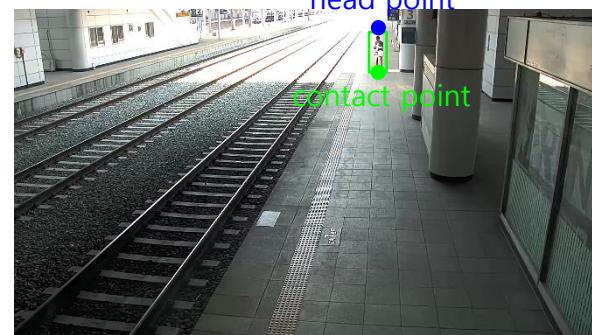


Camera Projection Model

- The Pinhole Camera Model

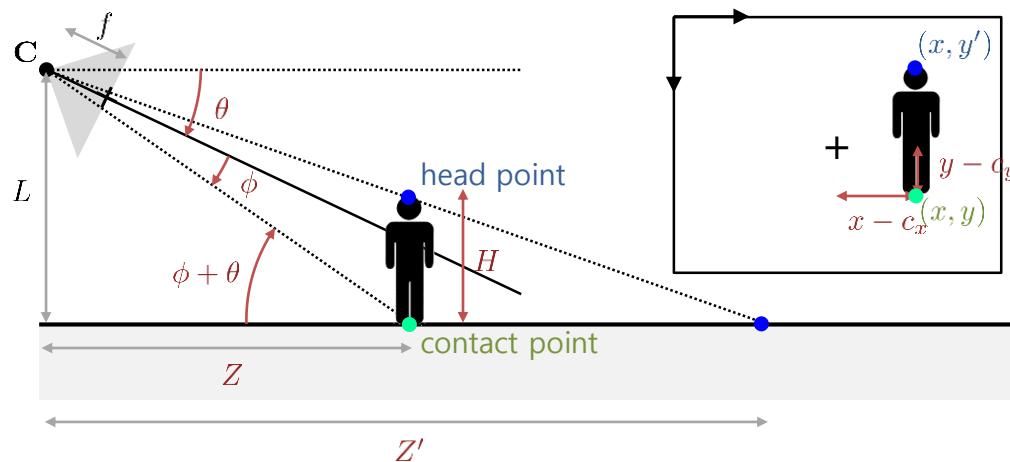
- Example: Simple Object Localization and Measurement

- Unknown: **Position** and also **height** (unit: [m])
- Assumptions
 - The focal length, principal points, camera height, and tilt angle are known.
 - The object is on the reference plane.



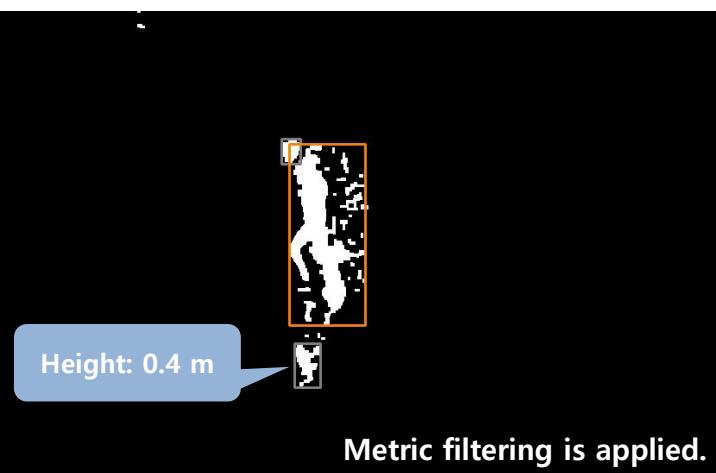
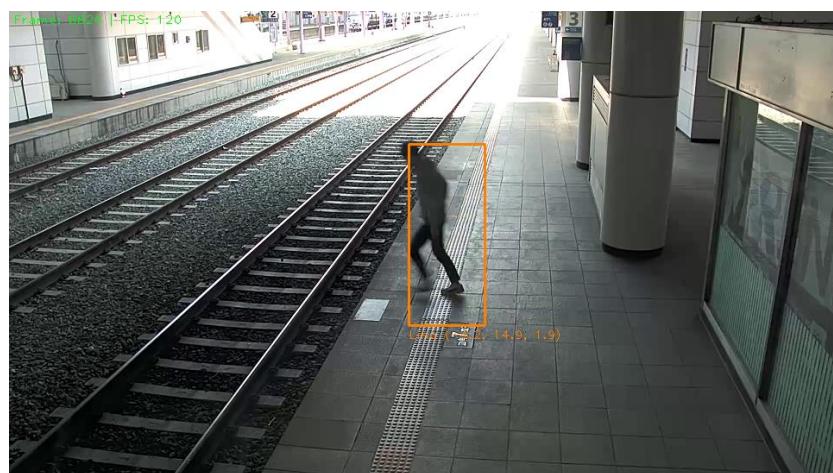
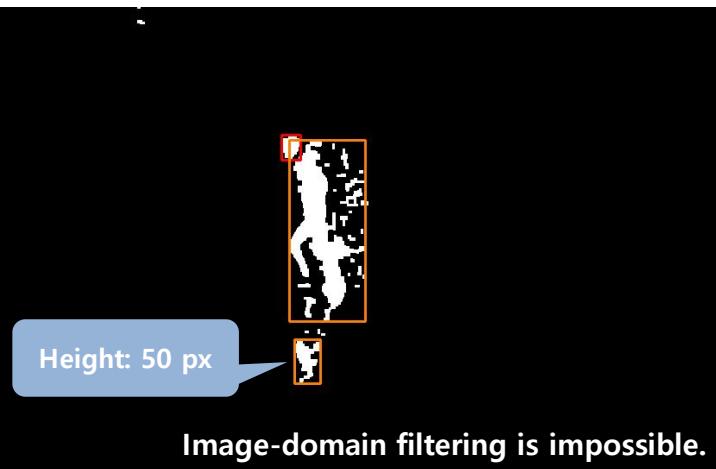
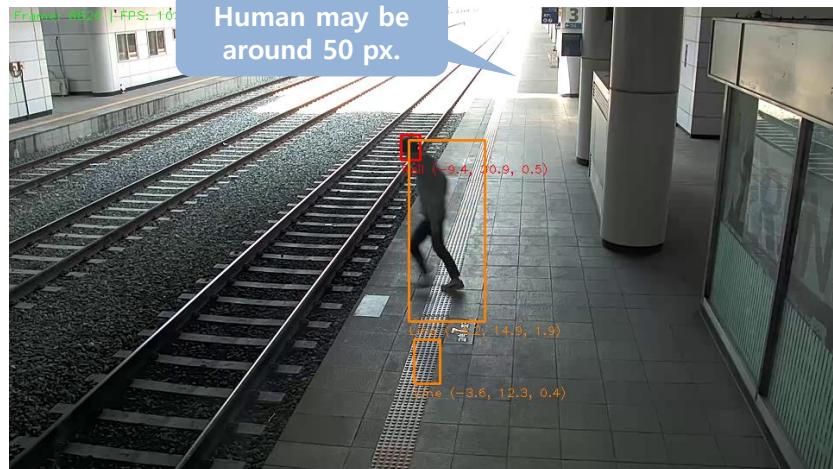
$$\therefore Z = \frac{L}{\tan(\phi + \theta)} \quad (\phi = \tan^{-1} \frac{y - c_y}{f})$$

$$X = \frac{x - c_x}{f} Z \quad H = \frac{Z' - Z}{Z'} L$$



Camera Projection Model

- The Pinhole Camera Model
 - Example: Simple Object Filtering

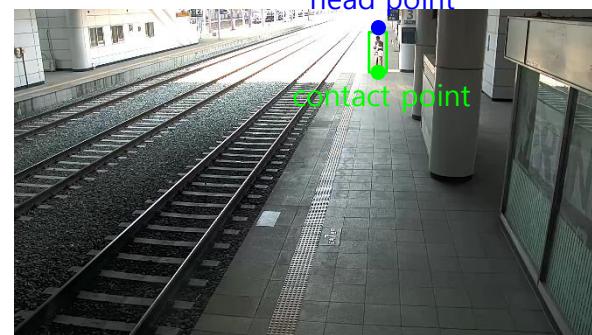


Camera Projection Model

- The Pinhole Camera Model

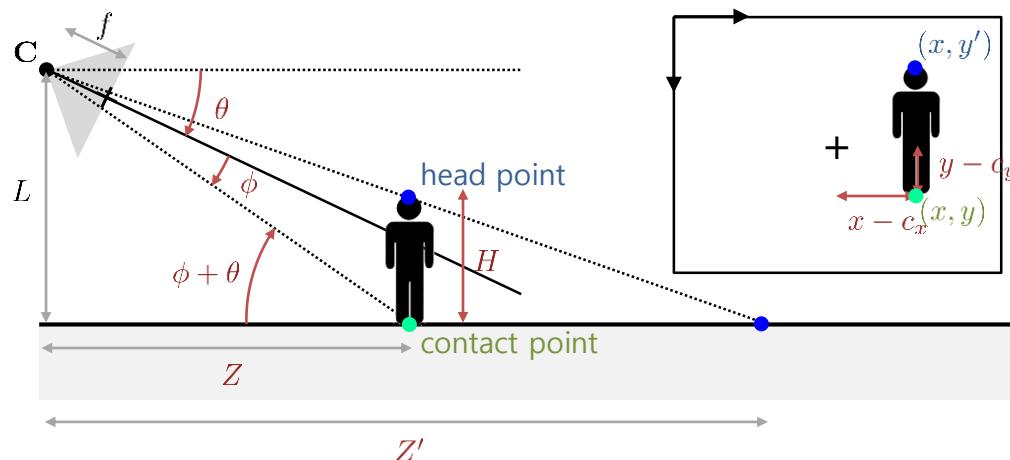
 - Example: Simple Object Proposal

 - Unknown: **Height** on the given image (unit: [pixel])
 - Assumptions
 - The focal length, principal points, camera height, and tilt angle are known.
 - The object is on the reference plane and its metric height is known.



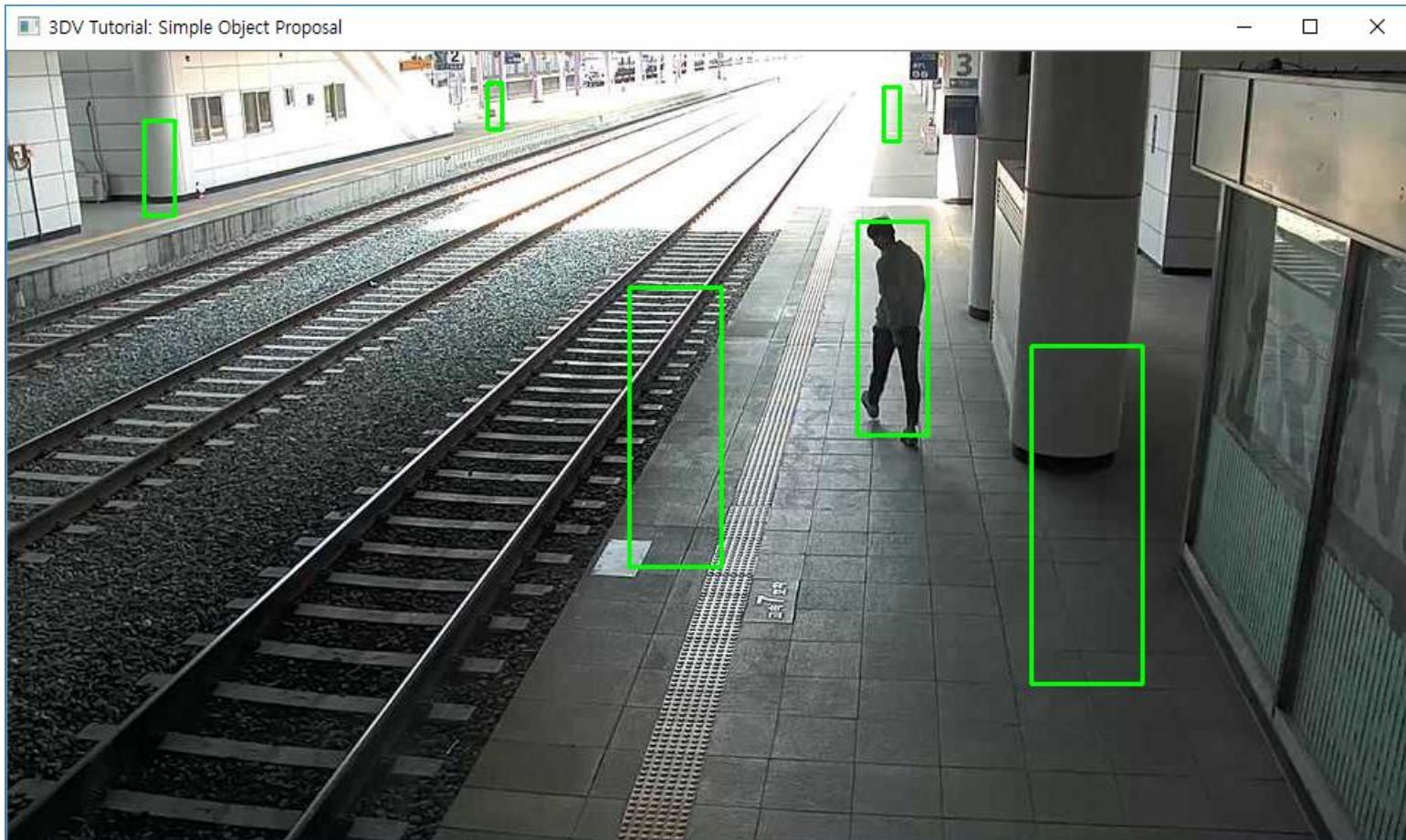
$$\therefore Z = \frac{L}{\tan(\phi + \theta)} \quad (\phi = \tan^{-1} \frac{y - c_y}{f})$$

$$y' = c_y + f \cdot \tan \left(\tan^{-1} \frac{L - H}{Z} - \theta \right)$$



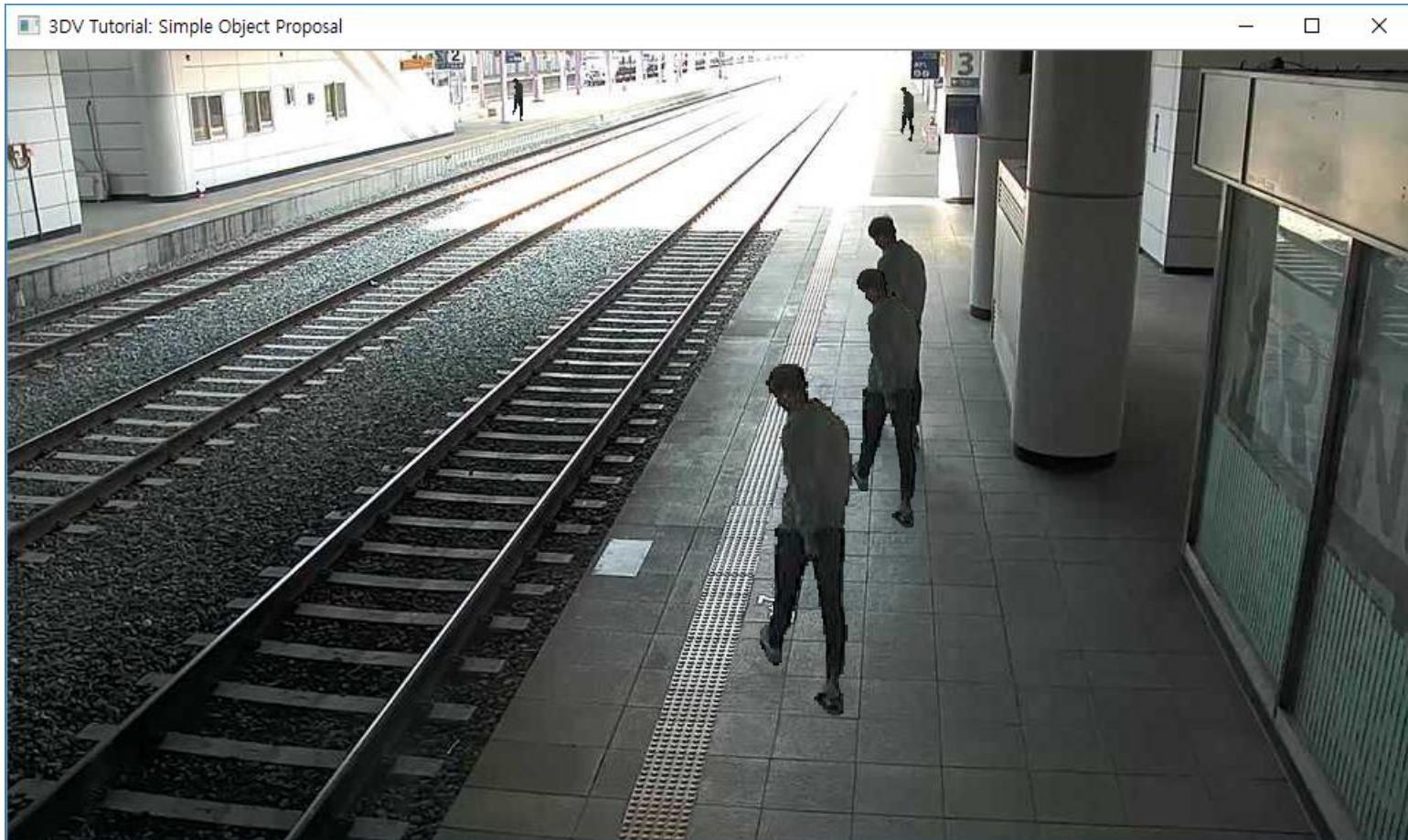
Camera Projection Model

- The Pinhole Camera Model
 - Example: Simple Object Proposal



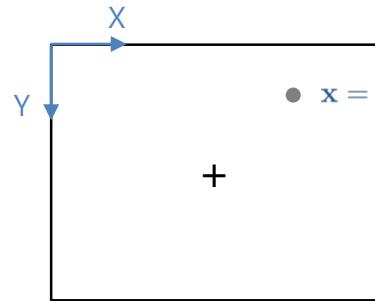
Camera Projection Model

- The Pinhole Camera Model
 - Example: Simple Object Proposal



Camera Projection Model

- Camera Matrix (Intrinsic Parameters)



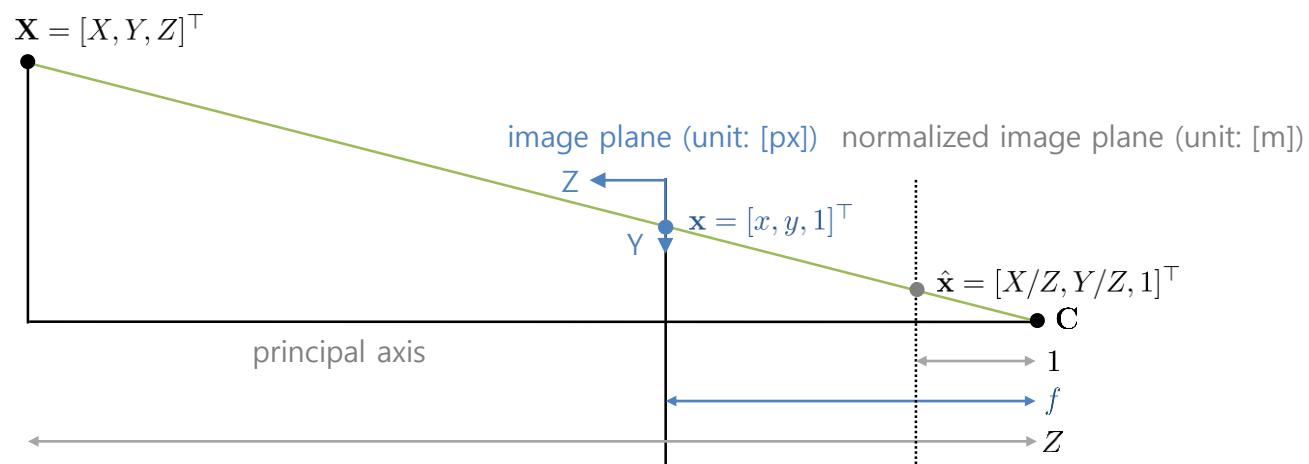
A diagram showing a 2D image plane with a coordinate system having X and Y axes. A point \mathbf{x} is shown on the plane, represented by a dot and a bracketed vector $[x, y]$. A plus sign (+) is placed below the image plane.

$$\bullet \quad \mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix} = f \begin{bmatrix} X/Z \\ Y/Z \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \end{bmatrix} \longrightarrow \mathbf{x} = \mathbf{K}\hat{\mathbf{x}}$$

where $\mathbf{K} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}$, $\mathbf{x} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$, and $\hat{\mathbf{x}} = \begin{bmatrix} X/Z \\ Y/Z \\ 1 \end{bmatrix}$

generalized

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



Camera Projection Model

- **Projection Matrix** (Intrinsic/Extrinsic Parameters)
 - If a 3D point is not based on the camera coordinate, the point should be transformed.

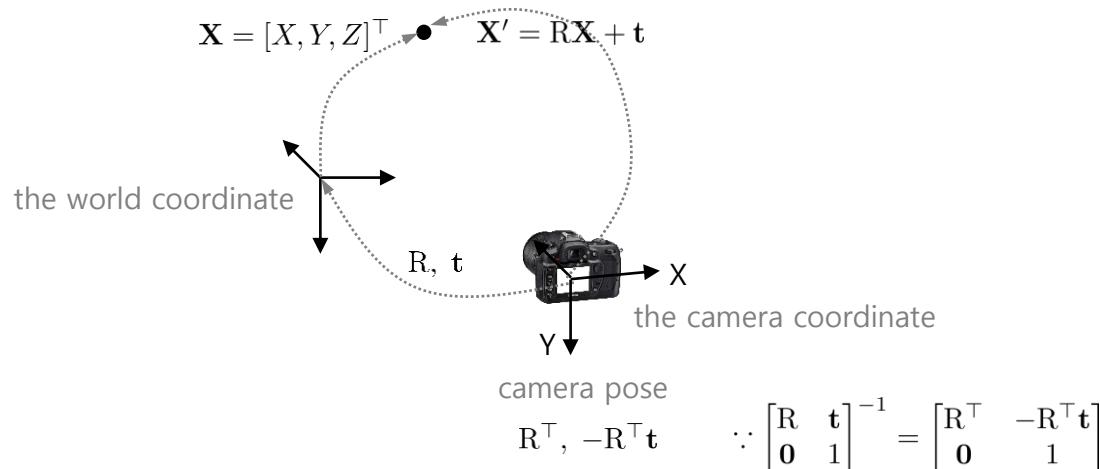
$$\mathbf{X}' = \mathbf{R}\mathbf{X} + \mathbf{t} \longrightarrow \mathbf{X}' = \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \mathbf{X} \text{ where } \mathbf{X} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$\mathbf{x} = \mathbf{P}\mathbf{X} \text{ where } \mathbf{P} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \text{ and } \mathbf{x} = \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

3x4 matrix

∴ \mathbf{x} is not normalized.

- No problem! To get its projected point on the image plane, we can normalize it later, $\begin{bmatrix} x/w \\ y/w \\ 1 \end{bmatrix}$.

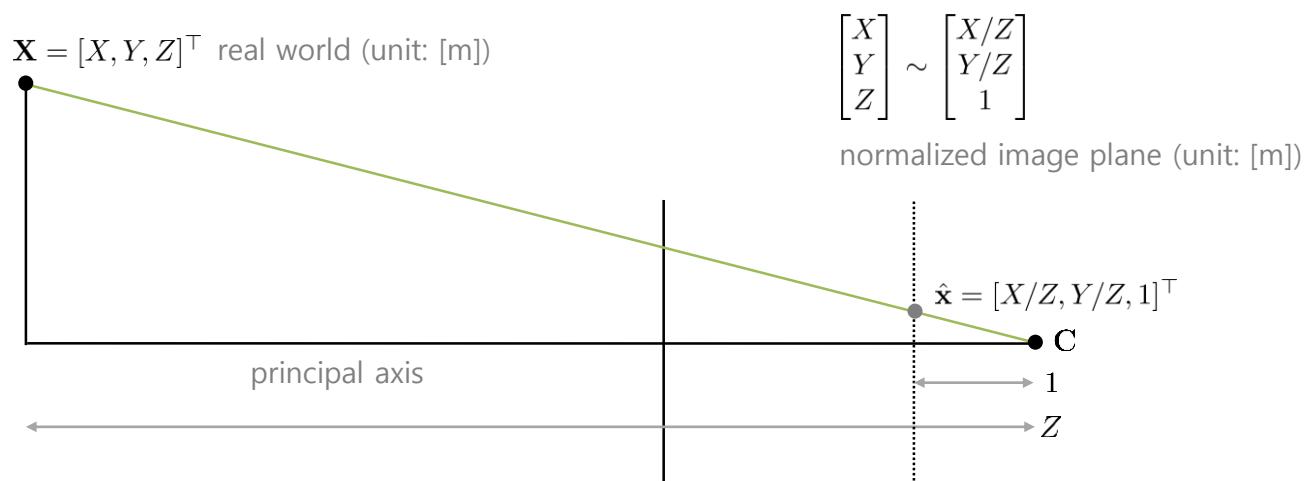


Camera Projection Model

$$\therefore \mathbf{x} = \mathbf{P}\mathbf{X} \quad \text{where} \quad \mathbf{P} = \mathbf{K}[\mathbf{R} | \mathbf{t}], \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ w \end{bmatrix}, \quad \text{and} \quad \mathbf{X} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- **Why Homogeneous Coordinate?**

- A linear transformation (rotation and translation) is applied by a single matrix multiplication.
- A light ray can be represented as a point on the image plane.
- An infinite point (a.k.a. ideal point) is numerically represented by $w = 0$.
- A point and line are represented beautifully as like $\mathbf{l}^T \mathbf{x} = 0$ or $\mathbf{x}^T \mathbf{l} = 0$ ($\mathbf{l} = [a, b, c]^T$).
- ...

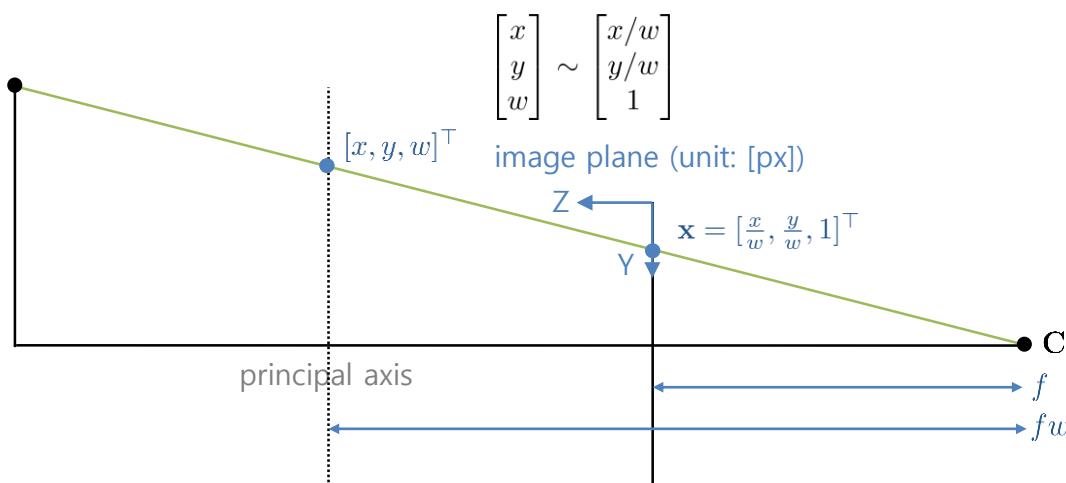


Camera Projection Model

$$\therefore \mathbf{x} = \mathbf{P}\mathbf{X} \quad \text{where} \quad \mathbf{P} = \mathbf{K}[\mathbf{R} | \mathbf{t}], \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ w \end{bmatrix}, \quad \text{and} \quad \mathbf{X} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- **Why Homogeneous Coordinate?**

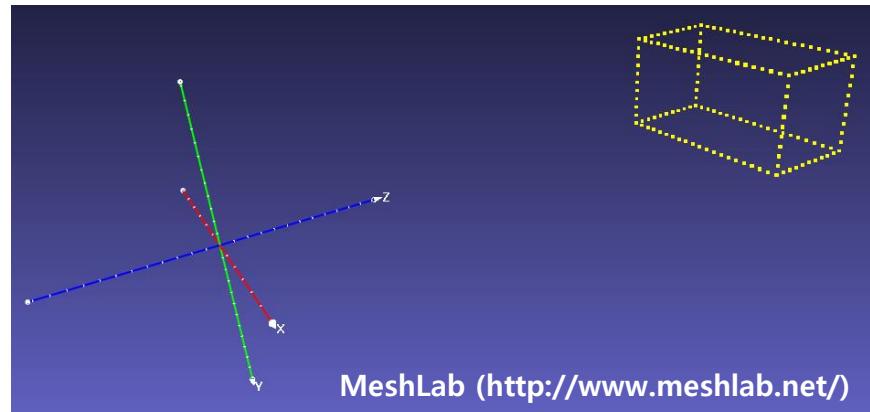
- A linear transformation (rotation and translation) is applied by a single matrix multiplication.
- A light ray can be represented as a point on the image plane.
- An infinite point (a.k.a. ideal point) is numerically represented by $w = 0$.
- A point and line are represented beautifully as like $\mathbf{l}^\top \mathbf{x} = 0$ or $\mathbf{x}^\top \mathbf{l} = 0$.
- ...



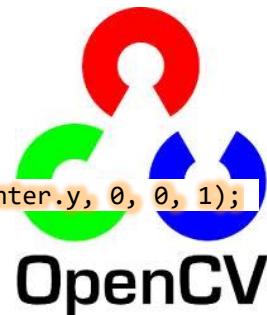
Example: Image Formation



```
1. #include "opencv_all.hpp"  
2.  
3. #define Rx(rx) (cv::Mat_<double>(3, 3) << 1, 0, 0, 0, cos(rx), -sin(rx), 0, sin(rx), cos(rx))  
4. #define Ry(ry) (cv::Mat_<double>(3, 3) << cos(ry), 0, sin(ry), 0, 1, 0, -sin(ry), 0, cos(ry))  
5. #define Rz(rz) (cv::Mat_<double>(3, 3) << cos(rz), -sin(rz), 0, sin(rz), cos(rz), 0, 0, 0, 1)  
6.  
7. int main(void)  
8. {  
9.     // The given camera configuration: focal length, principal point, image resolution, position, and orientation  
10.    double camera_focal = 1000;  
11.    cv::Point2d camera_center(320, 240);  
12.    cv::Size camera_res(640, 480);  
13.    cv::Point3d camera_pos[] = { cv::Point3d(0, 0, 0), cv::Point3d(-2, -2, 0), ... };  
14.    cv::Point3d camera_ori[] = { cv::Point3d(0, 0, 0), cv::Point3d(-CV_PI / 12, CV_PI / 12, 0), ... };  
15.    double camera_noise = 1;  
16.  
17.    // Load a point cloud in the homogeneous coordinate  
18.    FILE* fin = fopen("data/box.xyz", "rt");  
19.    if (fin == NULL) return -1;  
20.    cv::Mat X;  
21.    while (!feof(fin))  
22.    {  
23.        double x, y, z;  
24.        if (fscanf(fin, "%lf %lf %lf", &x, &y, &z) == 3) X.push_back(cv::Vec4d(x, y, z, 1));  
25.    }  
26.    fclose(fin);  
27.    X = X.reshape(1).t(); // Convert to a 4 x N matrix
```



Example: Image Formation



```

31. // Generate images for each camera pose
32. cv::Mat K = (cv::Mat_<double>(3, 3) << camera_focal, 0, camera_center.x, 0, camera_focal, camera_center.y, 0, 0, 1);
33. for (int i = 0; i < sizeof(camera_pos) / sizeof(cv::Point3d); i++)
34. {
35.     // Derive a projection matrix
36.     cv::Mat Rc = Rz(camera_ori[i].z) * Ry(camera_ori[i].y) * Rx(camera_ori[i].x);
37.     cv::Mat tc = (cv::Mat_<double>(3, 1) << camera_pos[i].x, camera_pos[i].y, camera_pos[i].z);
38.     cv::Mat Rt;
39.     cv::hconcat(Rc.t(), -Rc.t() * tc, Rt);           ∴  $\begin{bmatrix} R & t \end{bmatrix}^{-1} = \begin{bmatrix} R^T & -R^T t \\ 0 & 1 \end{bmatrix}$ 
40.     cv::Mat P = K * Rt;

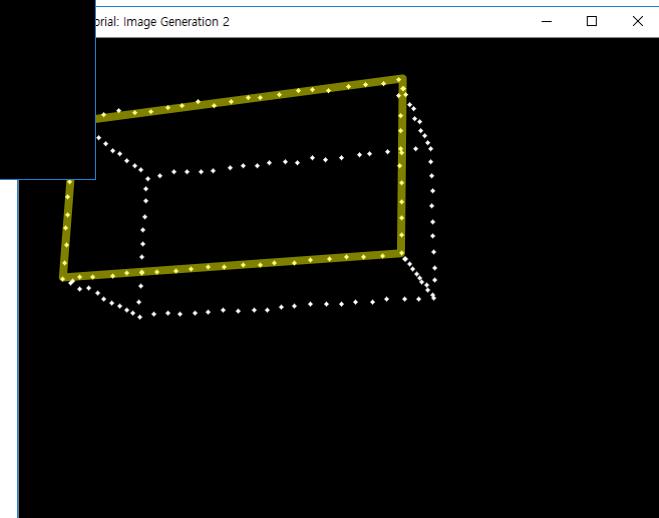
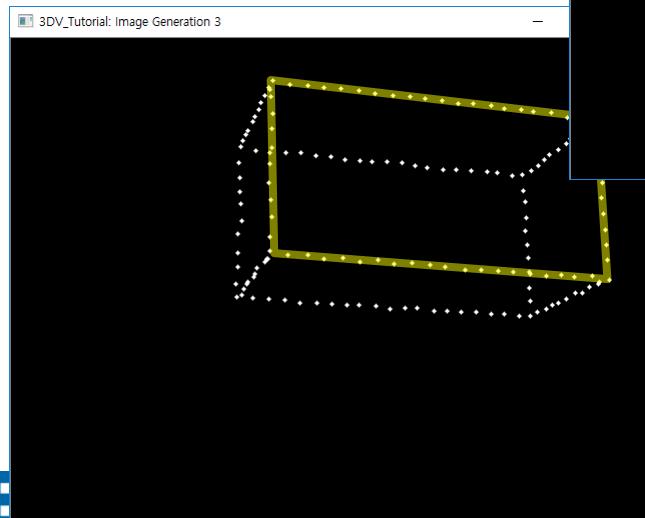
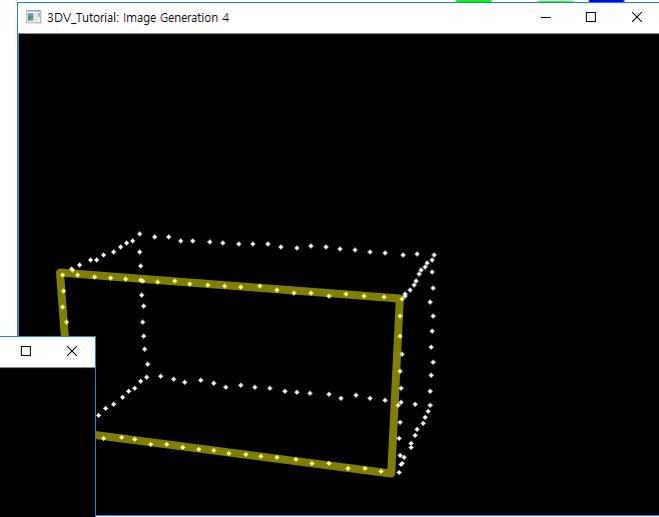
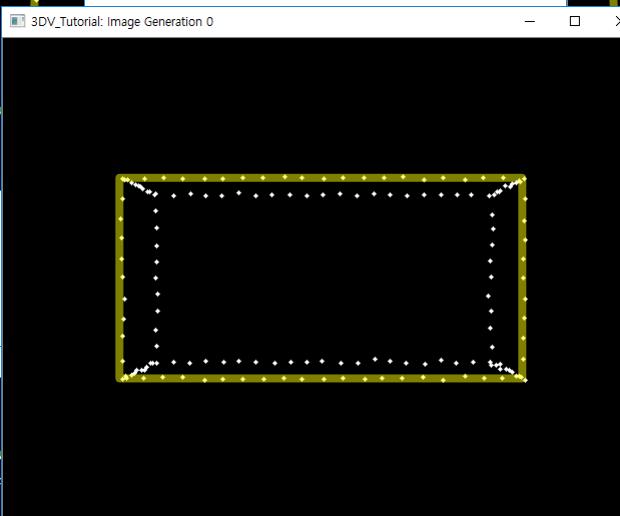
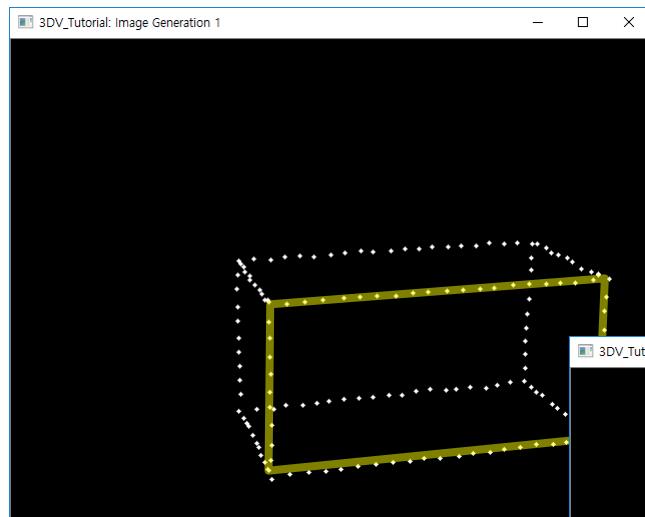
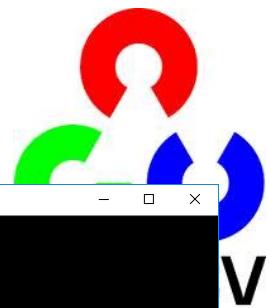
41.     // Project the points (c.f. OpenCV provide 'cv::projectPoints()' with consideration of distortion.)
42.     cv::Mat x = P * X;
43.     x.row(0) = x.row(0) / x.row(2);
44.     x.row(1) = x.row(1) / x.row(2);
45.     x.row(2) = 1;           ∴  $\begin{bmatrix} x \\ y \\ w \end{bmatrix} \sim \begin{bmatrix} x/w \\ y/w \\ 1 \end{bmatrix}$ 

46.     cv::Mat noise(2, x.cols, x.type());
47.     cv::randn(noise, cv::Scalar(0), cv::Scalar(camera_noise));
48.     x.rowRange(0, 2) = x.rowRange(0, 2) + noise; // Add noise

49.     // Show and store the points
50.     cv::Mat image = cv::Mat::zeros(camera_res, CV_8UC1);
51.     for (int c = 0; c < x.cols; c++)
52.     {
53.         cv::Point p(x.at<double>(0, c), x.at<double>(1, c));
54.         if (p.x >= 0 && p.x < camera_res.width && p.y >= 0 && p.y < camera_res.height)
55.             cv::circle(image, p, 2, 255, -1);
56.     }
57.     cv::imshow(cv::format("3DV_Tutorial: Image Formation %d", i), image);

58.     FILE* fout = fopen(cv::format("image_formation%d.xyz", i).c_str(), "wt");
59.     if (fout == NULL) return -1;
60.     for (int c = 0; c < x.cols; c++)
61.         fprintf(fout, "%f %f 1\n", x.at<double>(0, c), x.at<double>(1, c));
62.     fclose(fout);
63. }
64. ...
65. }
```

Example: Image Formation



Camera Projection Model

- **Geometric Distortion**

- Radial/Tangential Distortion Model

$$\begin{bmatrix} \hat{x}_d \\ \hat{y}_d \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4 + \dots) \begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} + \begin{bmatrix} 2p_1 \hat{x} \hat{y} + p_2 (r^2 + 2\hat{x}^2) \\ 2p_2 \hat{x} \hat{y} + p_1 (r^2 + 2\hat{y}^2) \end{bmatrix}$$

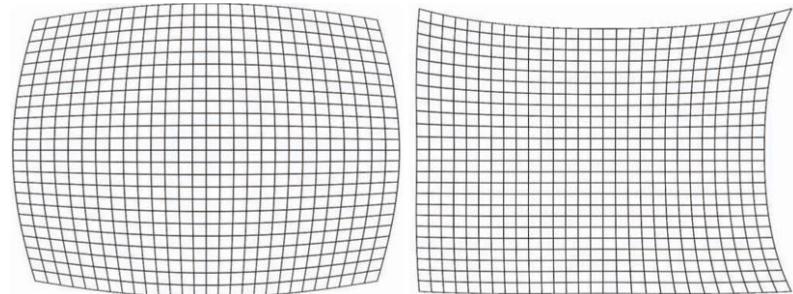
where $r^2 = \hat{x}^2 + \hat{y}^2$

defined on the normalized image plane

- Radial distortion: k_1, k_2, \dots
- Tangential distortion: p_1, p_2

- Image Rectification

- OpenCV `cv::undistort()` and `cv::undistortPoints()` (c.f. included in `imgproc` module)
↔ `cv::projectPoints()` (c.f. included in `calib3d` module)
 - Camera Distortion Correction: Theory and Practice, <http://darkpgmr.tistory.com/31>

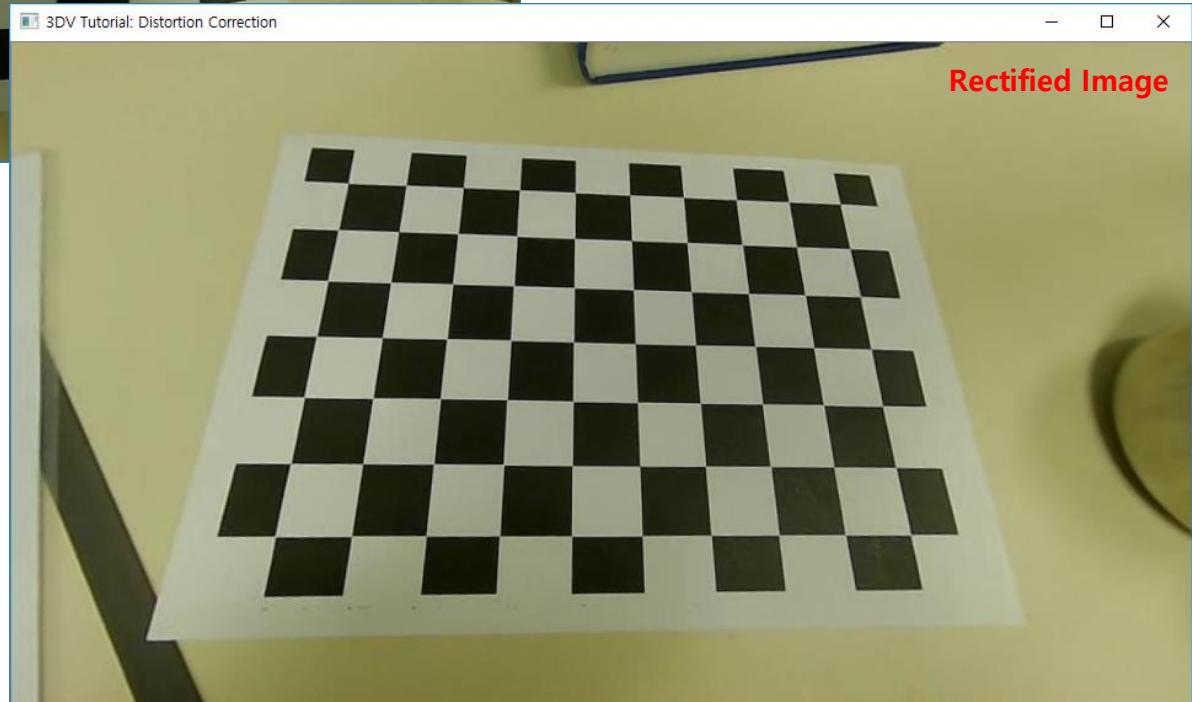
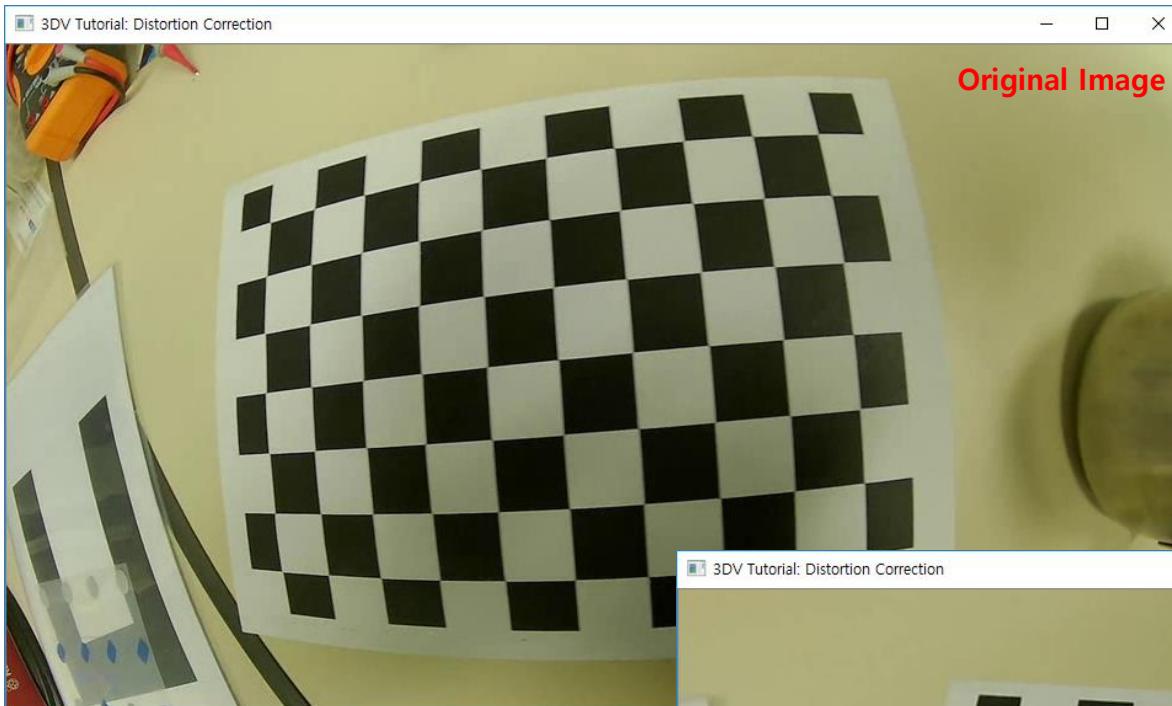


correction

K1: 1.105763E-01
K2: 1.886214E-02
K3: 1.473832E-02
P1:-8.448460E-03
P2:-7.356744E-03



Example: Geometric Distortion Correction



Example: Geometric Distortion Correction

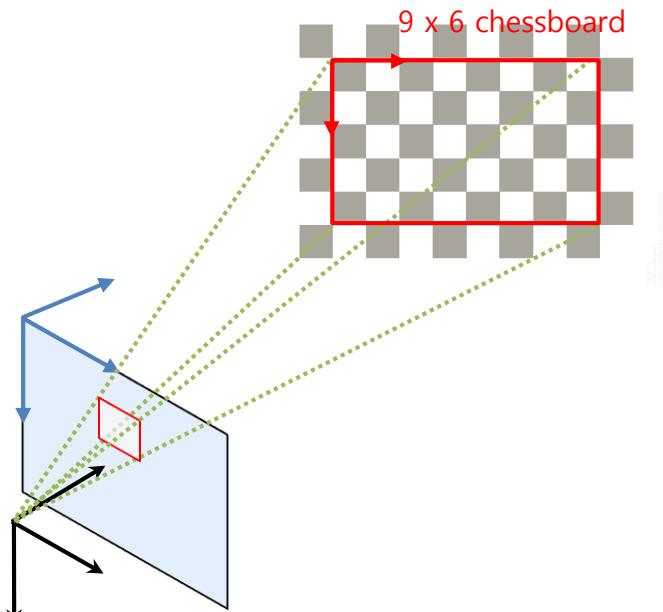


```
1. #include "opencv_all.hpp"
2.
3. int main(void)
4. {
5.     cv::Mat K = (cv::Mat<double>(3, 3) << 432.7390364738057, 0, 476.0614994349778, ..., 0, 0, 1);
6.     cv::Mat dist_coeff = (cv::Mat<double>(5, 1) << -0.2852754904152874, 0.1016466459919075, ...);
7.
8.     // Open a video
9.     cv::VideoCapture video;
10.    if (!video.open("data/chessboard.avi")) return -1;
11.
12.    // Run distortion correction
13.    bool show_rectify = true;
14.    cv::Mat map1, map2;
15.    while (true)
16.    {
17.        // Grab an image from the video
18.        cv::Mat image;
19.        video >> image;
20.        if (image.empty()) break;
21.
22.        // Rectify geometric distortion (c.f. 'cv::undistort()' can be applied for one-time remapping.)
23.        if (show_rectify)
24.        {
25.            if (map1.empty() || map2.empty())
26.                cv::initUndistortRectifyMap(K, dist_coeff, cv::Mat(), cv::Mat(), image.size(), CV_32FC1, map1, map2);
27.            cv::remap(image, image, map1, map2, cv::InterpolationFlags::INTER_LINEAR);
28.        }
29.
30.        // Show the image
31.        cv::imshow("3DV Tutorial: Distortion Correction", image);
32.        int key = cv::waitKey(1);
33.        if (key == 27) break;                                // 'ESC' key: Exit
34.        else if (key == 9) show_rectify = !show_rectify;    // 'Tab' key: Toggle rectification
35.        ...
36.    }
37.
38.    video.release();
39.    return 0;
40. }
```

General 2D-3D Geometry

- **Camera Calibration**

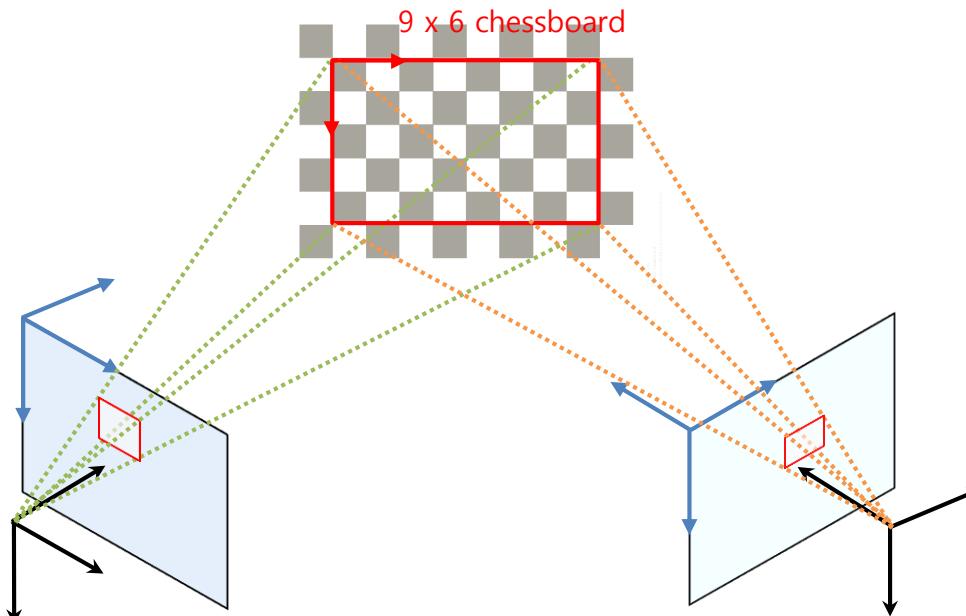
- Known: 3D points $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n$ and their projected points $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$
- Unknown: 5 (intrinsic w/o distortion) + 6 (extrinsic) DoF
- Constraints: $n \times$ projection $\mathbf{x}_i = \mathbf{P}\mathbf{X}_i$



General 2D-3D Geometry

- Camera Calibration

- Known: 3D points $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n$ and their projected points from each camera \mathbf{x}_i^j
- Unknown: 5 (intrinsic w/o distortion) + $m \times 6$ (extrinsic) DoF
- Constraints: $n \times m \times$ projection $\mathbf{x}_i^j = \mathbf{K}[\mathbf{R}_j | \mathbf{t}_j] \mathbf{X}_i$
- Solutions
 - OpenCV `cv::calibrateCamera()` and `cv::initCameraMatrix2D()`
 - Camera Calibration Toolbox for MATLAB, http://www.vision.caltech.edu/bouguetj/calib_doc/
 - GML C++ Camera Calibration Toolbox, <http://graphics.cs.msu.ru/en/node/909>
 - DarkCamCalibrator, <http://darkpgmr.tistory.com/139>

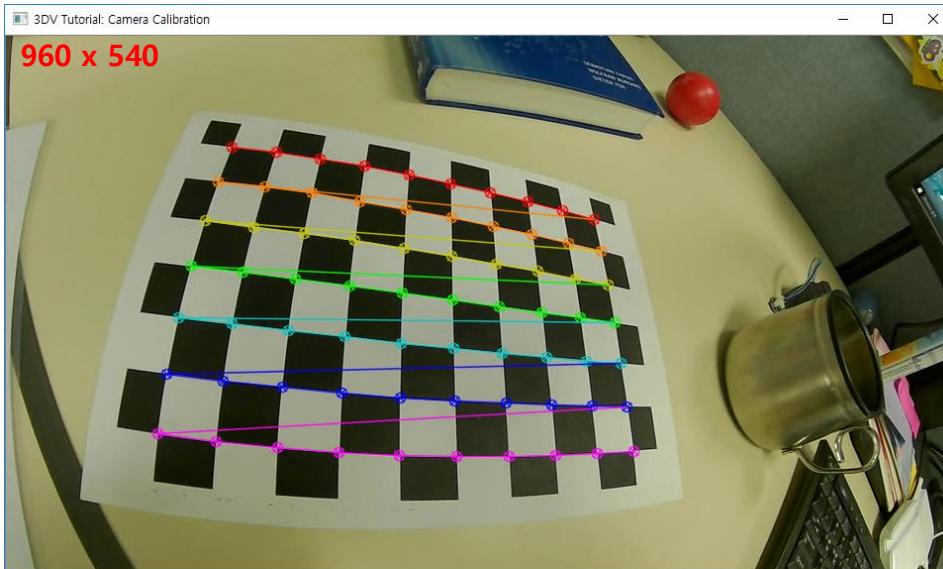




Example: Camera Calibration

```
1. #include "opencv_all.hpp"
2.
3. int main(void)
4. {
5.     bool select_images = true;
6.     cv::Size board_pattern(10, 7);
7.     float board_cellsize = 0.025f;
8.
9.     ...
10.
11.    // Select images
12.    std::vector<cv::Mat> images;
13.    ...
14.
15.    // Find 2D corner points from the given images
16.    std::vector<std::vector<cv::Point2f> > img_points;
17.    for (size_t i = 0; i < images.size(); i++)
18.    {
19.        std::vector<cv::Point2f> pts;
20.        if (cv::findChessboardCorners(images[i], board_pattern, pts))
21.            img_points.push_back(pts);
22.    }
23.    if (img_points.empty()) return -1;
24.
25.    // Prepare 3D points of the chess board
26.    std::vector<std::vector<cv::Point3f> > obj_points(1);
27.    for (int r = 0; r < board_pattern.height; r++)
28.        for (int c = 0; c < board_pattern.width; c++)
29.            obj_points[0].push_back(cv::Point3f(board_cellsize * c, board_cellsize * r, 0));
30.    obj_points.resize(img_points.size(), obj_points[0]); // Copy
31.
32.    // Calibrate the camera
33.    cv::Mat K = cv::Mat::eye(3, 3, CV_64F);
34.    cv::Mat dist_coeff = cv::Mat::zeros(4, 1, CV_64F);
35.    std::vector<cv::Mat> rvecs, tvecs;
36.    double rms = cv::calibrateCamera(obj_points, img_points, images[0].size(), K, dist_coeff, rvecs, tvecs);
37.
38.    // Report calibration results
39.    ...
40.    return 0;
41. }
```

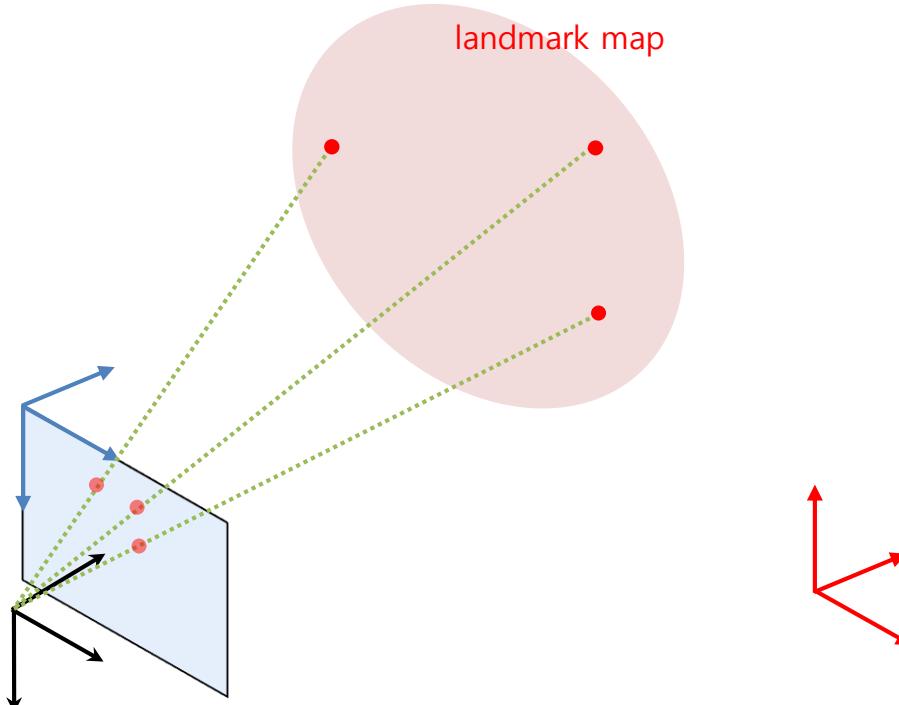
Example: Camera Calibration



```
## Camera Calibration Results
* The number of applied images = 22
* RMS error = 0.473353
* Camera matrix (K) =
[432.7390364738057, 0, 476.0614994349778]
[0, 431.2395555913084, 288.7602152621297]
[0, 0, 1]
* Distortion coefficient (k1, k2, p1, p2, k3, ...) =
[-0.2852754904152874, 0.1016466459919075,
 -0.0004420196146339175, 0.0001149909868437517,
 -0.01803978785585194]
```

General 2D-3D Geometry

- **Absolute Camera Pose Estimation** (Perspective-n-Point; PnP)
 - Known: 3D points $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n$, their projected points $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$, and camera matrix \mathbf{K}
 - Unknown: **Camera pose (6 DoF)**
 - Constraints: $n \times$ projection $\mathbf{x}_i = \mathbf{K}[\mathbf{R} | \mathbf{t}] \mathbf{X}_i$
 - Solutions ($n \geq 3$) → 3-point algorithm
 - OpenCV `cv::solvePnP()` and `cv::solvePnPRansac()`
 - Efficient PnP (EPnP), <http://cvlab.epfl.ch/EPnP/>



Example: Camera Pose Estimation (Chessboard)

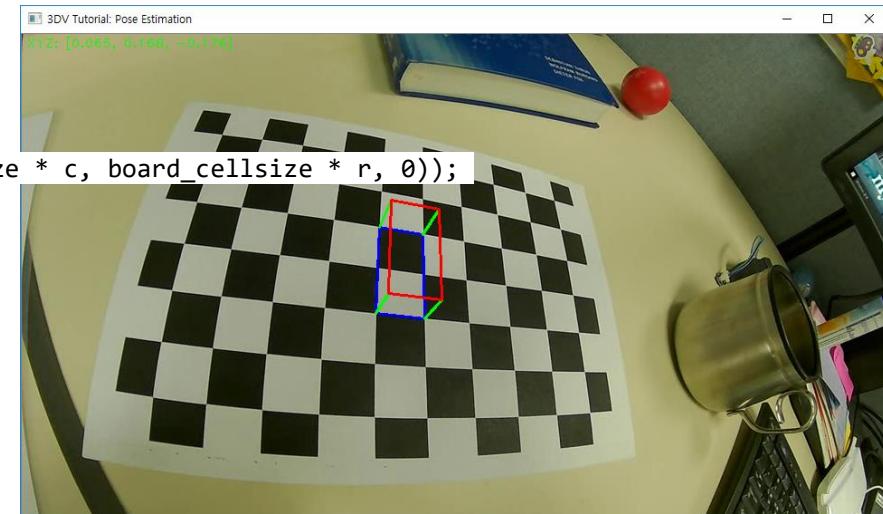
```
1. #include "opencv_all.hpp"

2. int main(void)
3. {
4.     cv::Mat K = (cv::Mat<double>(3, 3) << 432.7390364738057, 0, 476.0614994349778, ...);
5.     cv::Mat dist_coeff = (cv::Mat<double>(5, 1) << -0.2852754904152874, 0.1016466459919075, ...);
6.     cv::Size board_pattern(10, 7);
7.     double board_cellsize = 0.025;

8.     // Open a video
9.     cv::VideoCapture video;
10.    if (!video.open("data/chessboard.avi")) return -1;

11.    // Prepare a box for simple AR
12.    std::vector<cv::Point3d> box_lower, box_upper;
13.    box_lower.push_back(cv::Point3d(4 * board_cellsize, 2 * board_cellsize, 0));
14.    box_lower.push_back(cv::Point3d(5 * board_cellsize, 2 * board_cellsize, 0));
15.    box_lower.push_back(cv::Point3d(5 * board_cellsize, 4 * board_cellsize, 0));
16.    box_lower.push_back(cv::Point3d(4 * board_cellsize, 4 * board_cellsize, 0));
17.    box_upper.push_back(cv::Point3d(4 * board_cellsize, 2 * board_cellsize, -board_cellsize));
18.    box_upper.push_back(cv::Point3d(5 * board_cellsize, 2 * board_cellsize, -board_cellsize));
19.    box_upper.push_back(cv::Point3d(5 * board_cellsize, 4 * board_cellsize, -board_cellsize));
20.    box_upper.push_back(cv::Point3d(4 * board_cellsize, 4 * board_cellsize, -board_cellsize));

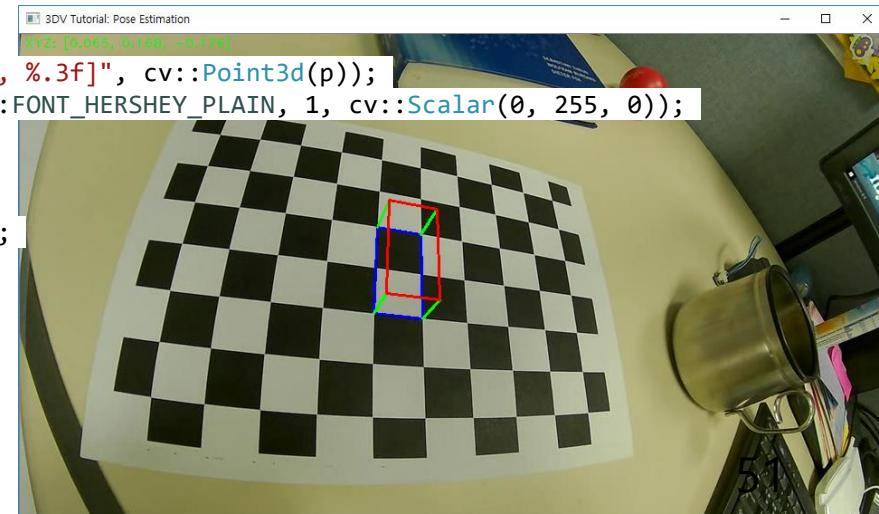
21.    // Run pose estimation
22.    std::vector<cv::Point3d> obj_points;
23.    for (int r = 0; r < board_pattern.height; r++)
24.        for (int c = 0; c < board_pattern.width; c++)
25.            obj_points.push_back(cv::Point3d(board_cellsize * c, board_cellsize * r, 0));
```



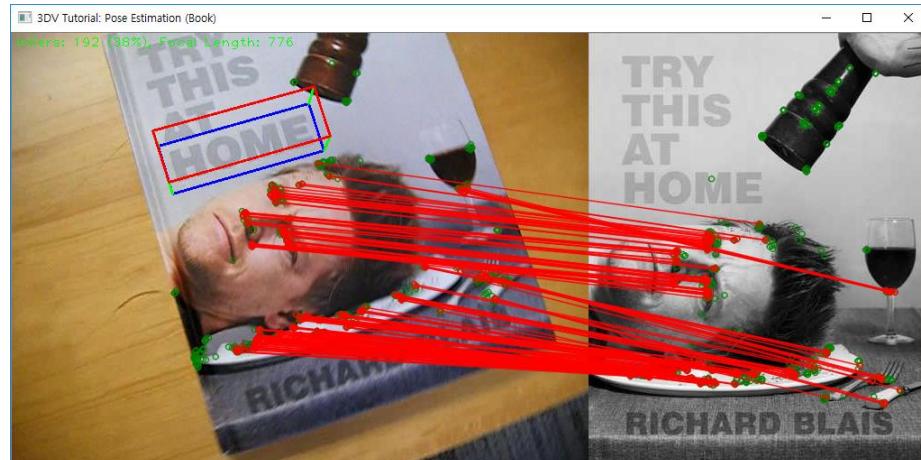
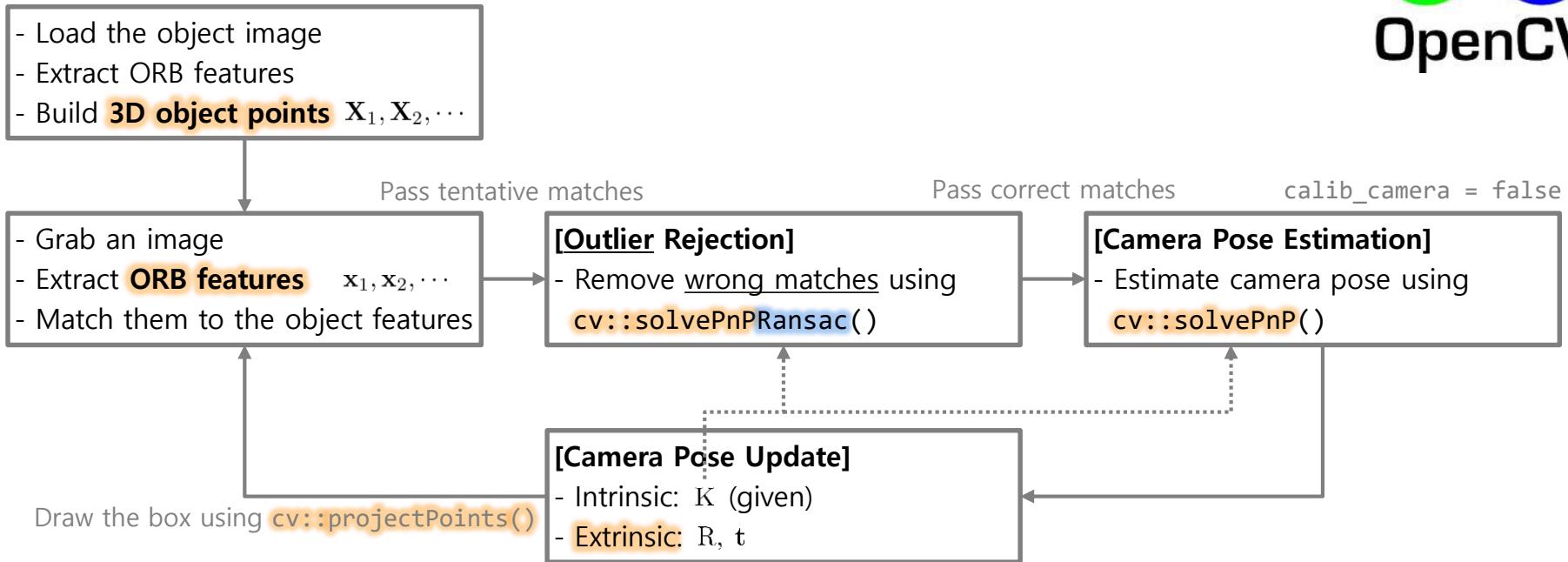
Example: Camera Pose Estimation (Chessboard)



```
1. while (true)
2. {
3.     // Grab an image from the video
4.     cv::Mat image;
5.     video >> image;
6.     if (image.empty()) break;
7.
8.     // Estimate camera pose
9.     std::vector<cv::Point2d> img_points;
10.    bool success = cv::findChessboardCorners(image, board_pattern, img_points, ...);
11.    if (success)
12.    {
13.        cv::Mat rvec, tvec;
14.        cv::solvePnP(obj_points, img_points, K, dist_coeff, rvec, tvec);
15.
16.        // Draw the box on the image
17.        cv::Mat line_lower, line_upper;
18.        cv::projectPoints(box_lower, rvec, tvec, K, dist_coeff, line_lower);
19.        cv::projectPoints(box_upper, rvec, tvec, K, dist_coeff, line_upper);
20.        ...
21.
22.        // Print camera position
23.        cv::Mat R;
24.        cv::Rodrigues(rvec, R);
25.        cv::Mat p = -R.t() * tvec;
26.        cv::String info = cv::format("XYZ: [% .3f, % .3f, % .3f]", cv::Point3d(p));
27.        cv::putText(image, info, cv::Point(5, 15), cv::FONT_HERSHEY_PLAIN, 1, cv::Scalar(0, 255, 0));
28.
29.        ...
30.    }
31.
32.    // Show the image
33.    cv::imshow("3DV Tutorial: Pose Estimation", image);
34.    int key = cv::waitKey(1);
35. }
```

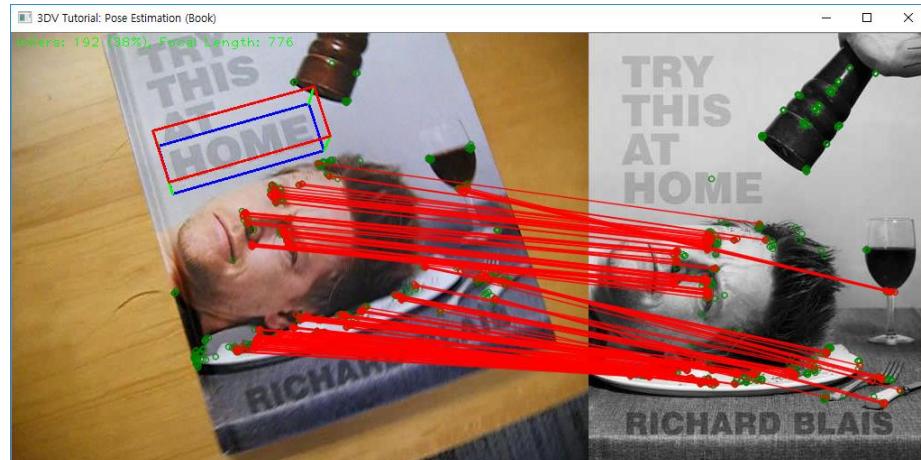
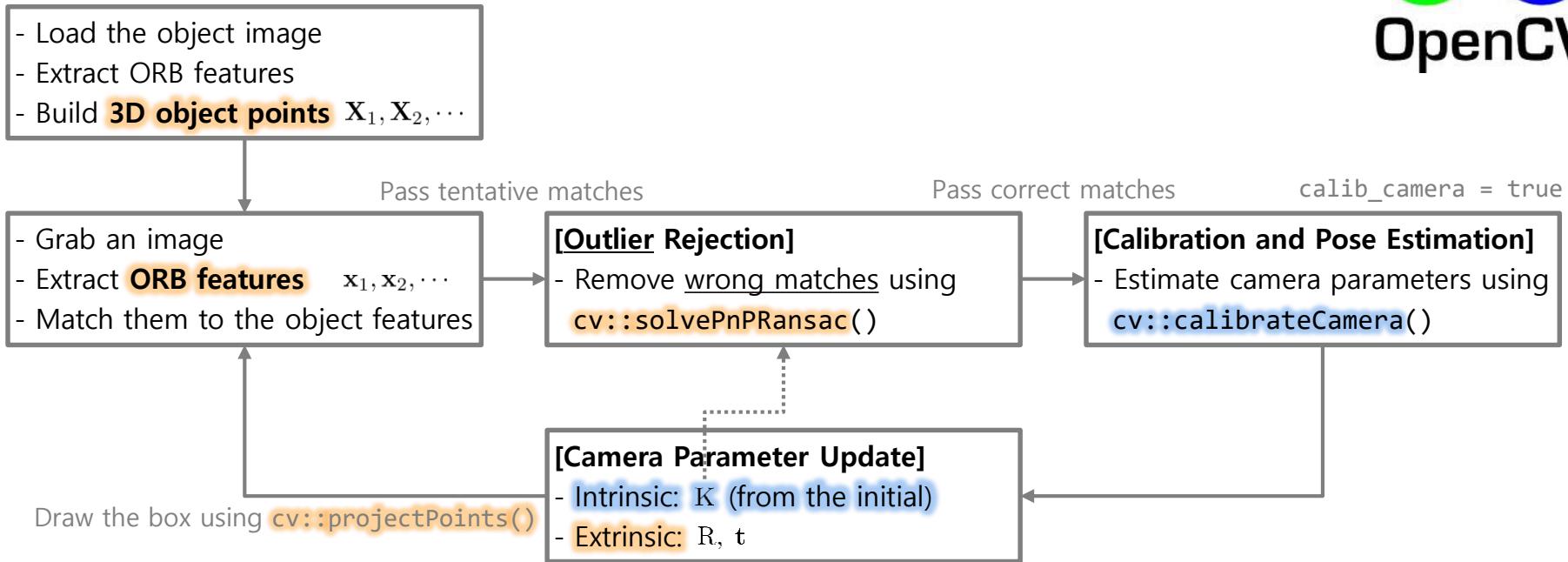
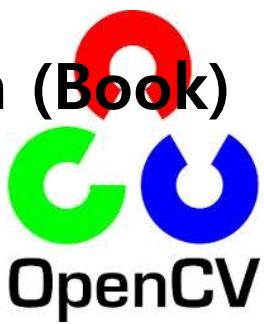


Example: Camera Pose Estimation (Book)



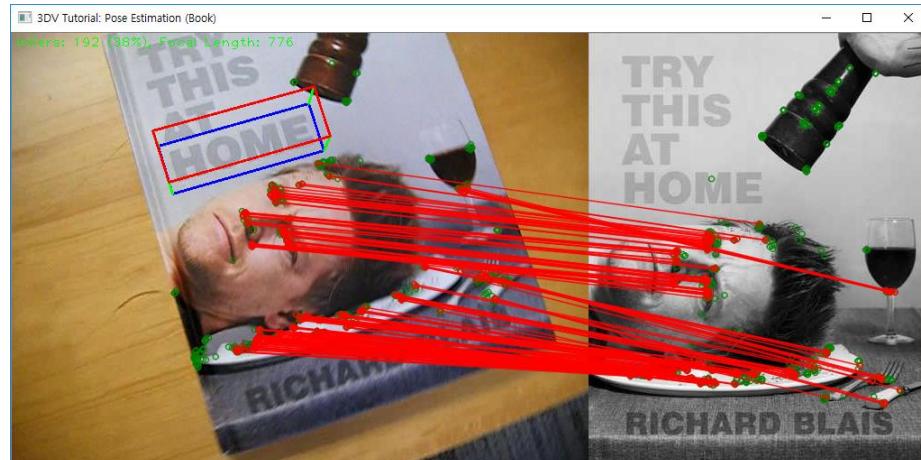
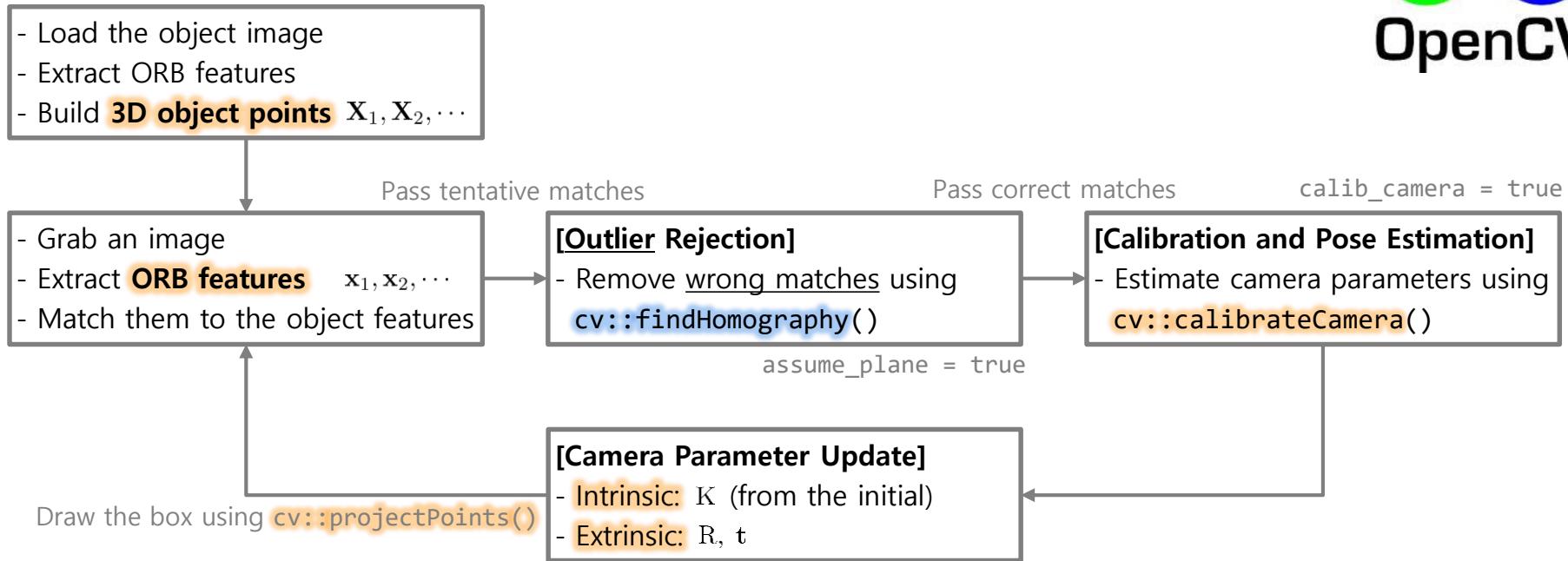
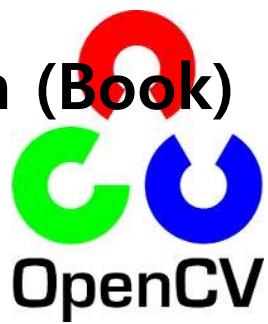
Example: Camera Calibration and Pose Estimation (Book)

(:: initially known camera parameters + autofocus)



Example: Camera Calibration and Pose Estimation (Book)

(\because unknown camera parameters + autofocus)



Two-view Geometry

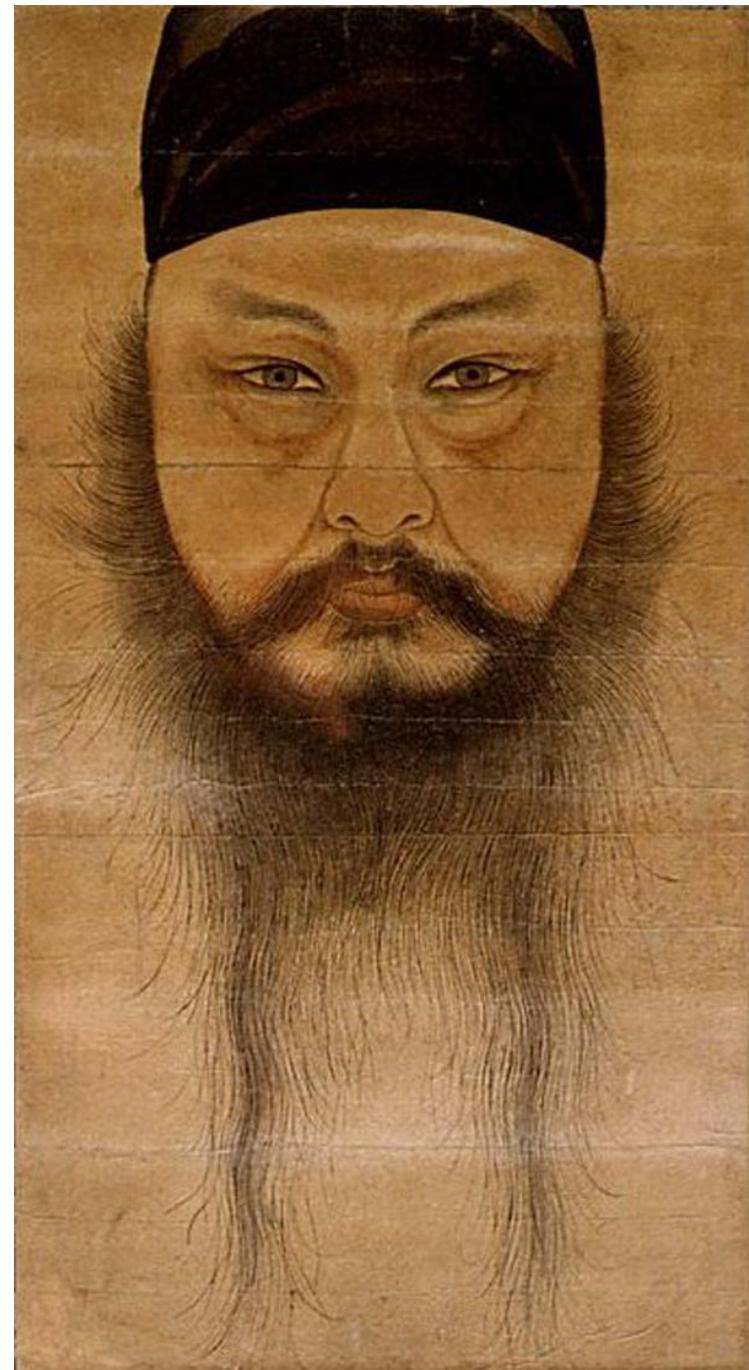


윤두서(1668-1715) 자화상, 국보 제240호
Korean National Treasure No. 240

Two-view Geometry

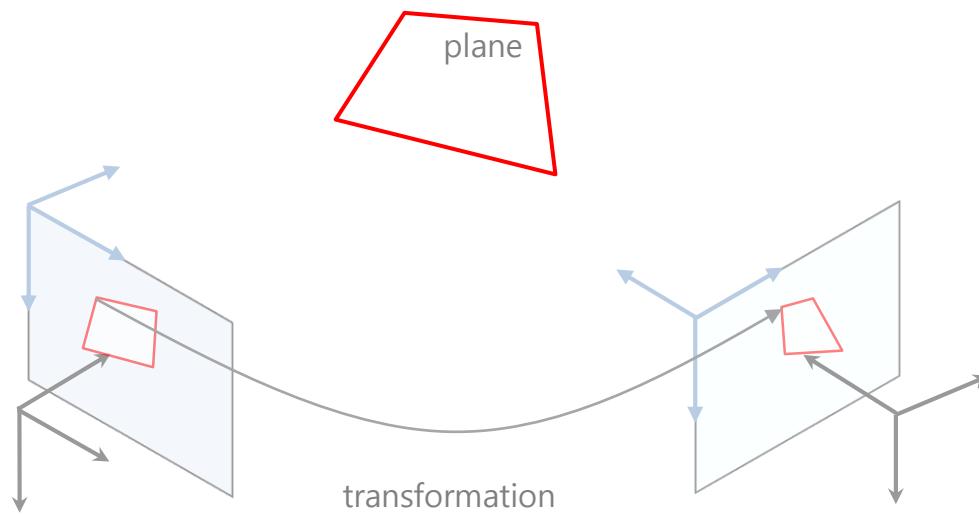
Planar 2D-2D Geometry (Projective Geometry)

General 2D-2D Geometry (Epipolar Geometry)

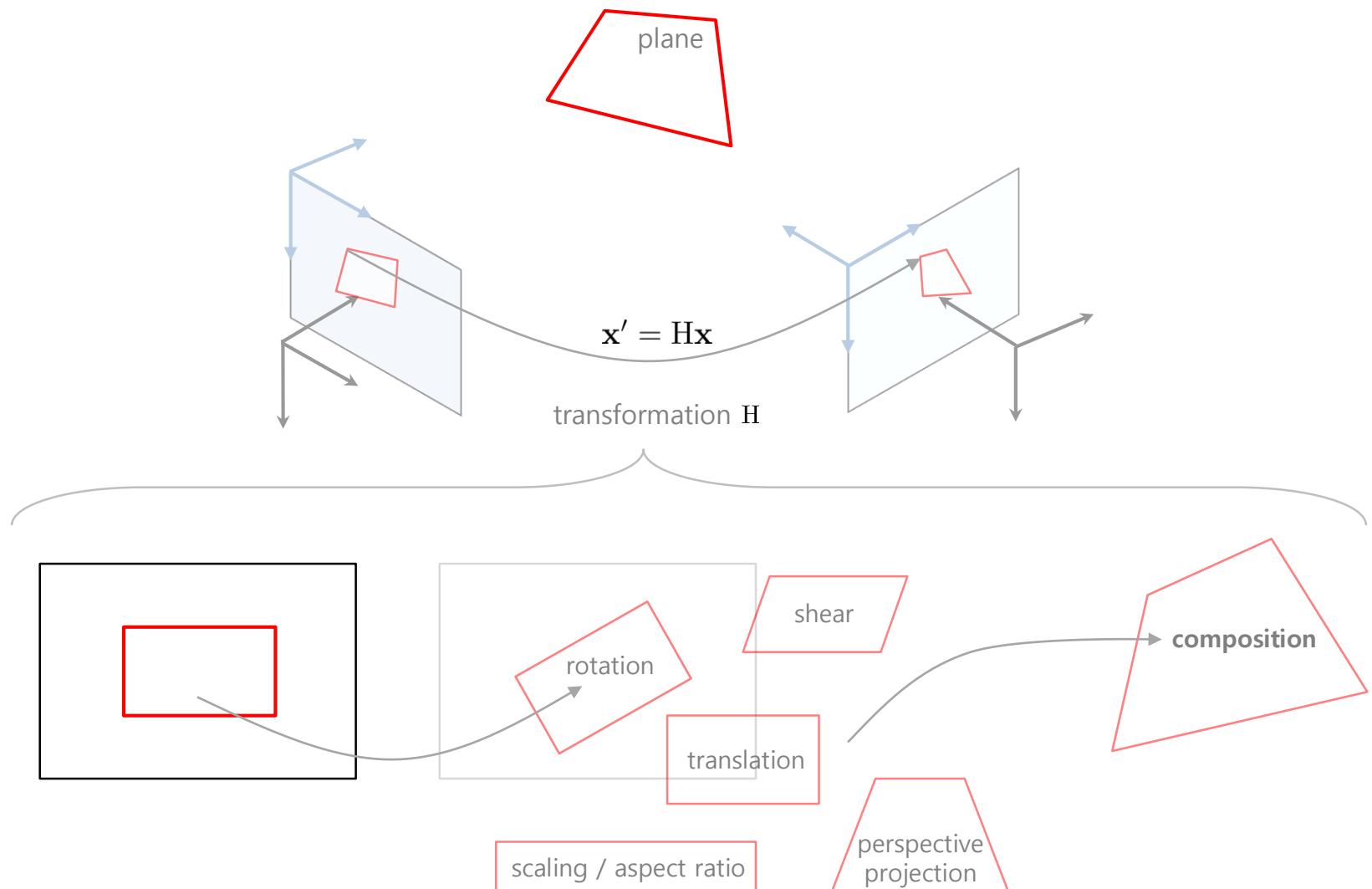


윤두서(1668-1715) 자화상, 국보 제240호
Korean National Treasure No. 240

Planar 2D-2D Geometry (Projective Geometry)



Planar 2D-2D Geometry (Projective Geometry)



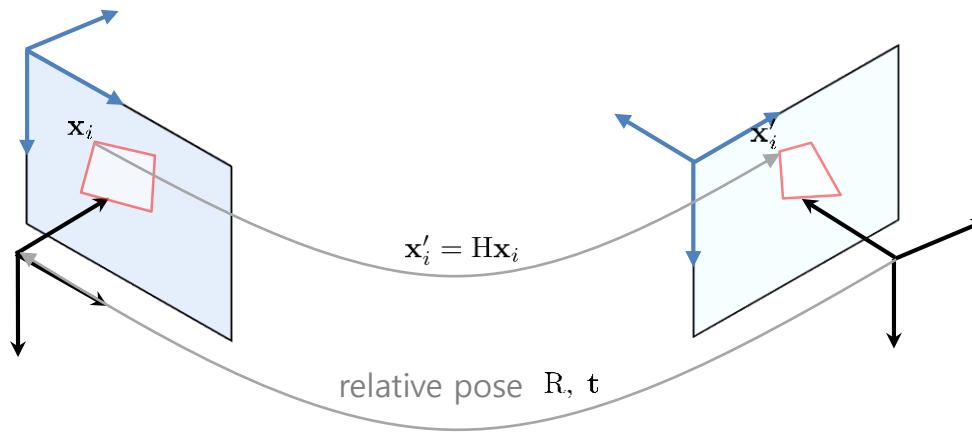
Overview of Projective Geometry

| | Euclidean Transform (a.k.a. Rigid Transform) | Similarity Transform | Affine Transform | Projective Transform (a.k.a. Planar Homography) |
|---|--|--|---|--|
| Matrix Forms H | $\begin{bmatrix} \cos \theta & -\sin \theta & t_x \\ \sin \theta & \cos \theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} s \cos \theta & -s \sin \theta & t_x \\ s \sin \theta & s \cos \theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ v_1 & v_2 & 1 \end{bmatrix}$ |
| DoF | 3 | 4 | 6 | 8 |
| Transformations - rotation - translation - scaling - aspect ratio - shear - perspective projection | O O | O O O | O O O O O O | O O O O O O O |
| Invariants - length - angle - ratio of lengths - parallelism - incidence - cross ratio | O O O O O O | O O O O O O | O O O O | O O |
| OpenCV Functions | | | <code>cv::getAffineTransform()</code> <code>cv::estimateRigidTransform()</code> - | <code>cv:: getPerspectiveTransform()</code> - <code>cv::findHomography()</code> <code>cv::warpAffine()</code> <code>cv::warpPerspective()</code> |

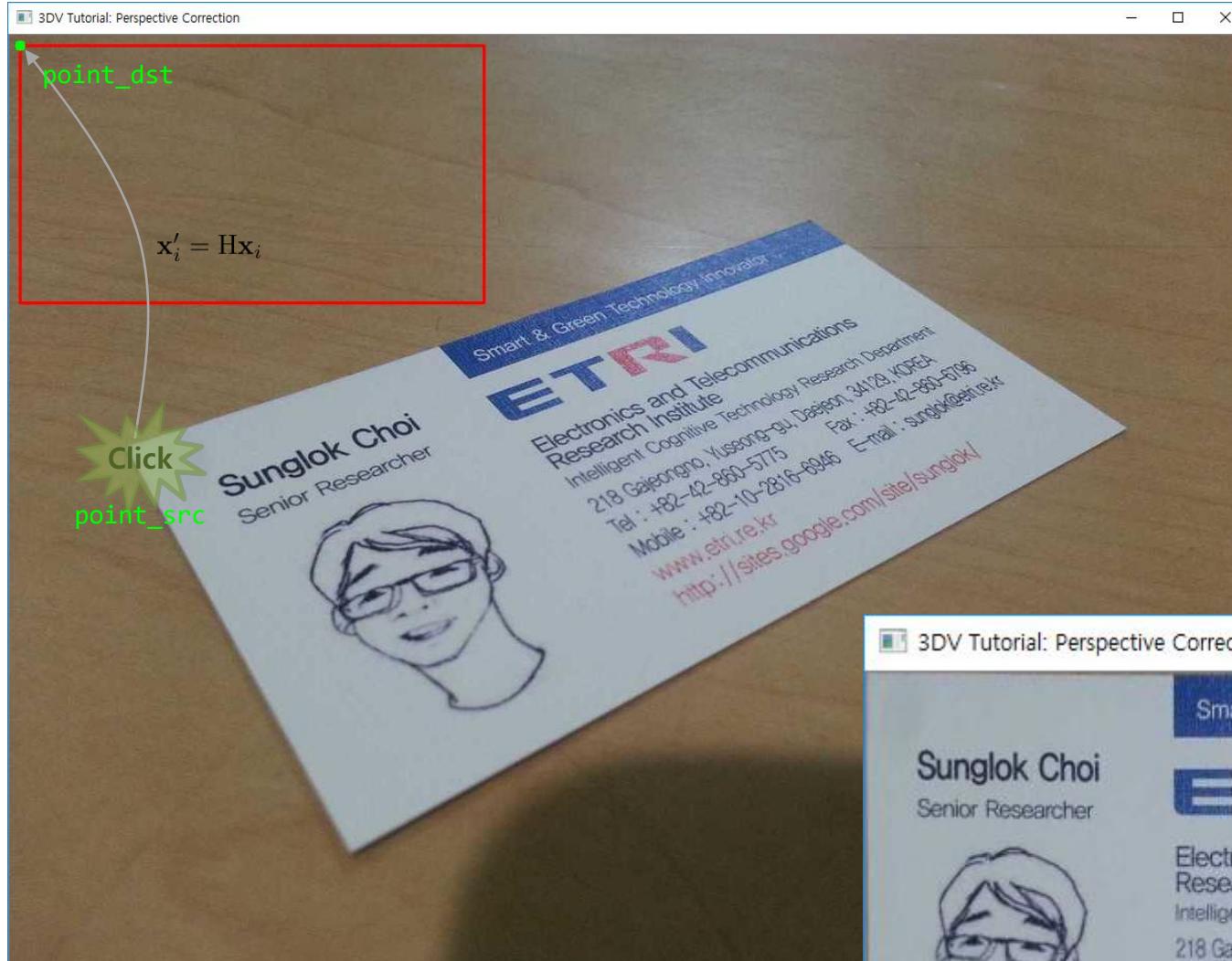
Planar 2D-2D Geometry (Projective Geometry)

- **Planar Homography Estimation**

- Known: Point correspondence $(\mathbf{x}_1, \mathbf{x}'_1), \dots, (\mathbf{x}_n, \mathbf{x}'_n)$
- Unknown: **Planar Homography (8 DoF)**
- Constraints: $n \times$ projective transformation $\mathbf{x}'_i = \mathbf{H}\mathbf{x}_i$
- Solutions ($n \geq 4$)
 - OpenCV `cv::getPerspectiveTransform()` and `cv::findHomography()`
 - c.f. More simplified transformations need less number of minimal correspondence.
 - Affine ($n \geq 3$), similarity ($n \geq 2$), Euclidean ($n \geq 2$)
- [Note] Planar homography can be decomposed as relative camera pose \mathbf{R}, \mathbf{t} .
 - OpenCV `cv::decomposeHomographyMat()`
 - c.f. However, the decomposition needs to know camera matrices.



Example: Perspective Distortion Correction





Example: Perspective Distortion Correction

```
1. #include "opencv_all.hpp"

2. void MouseEventHandler(int event, int x, int y, int flags, void* param)
3. {
4.     ...
5.     std::vector<cv::Point> *points_src = (std::vector<cv::Point> *)param;
6.     points_src->push_back(cv::Point(x, y));
7.     ...
8. }

9. int main(void)
10.{ 
11.     cv::Size card_size(450, 250);

12.    // Prepare the rectified points
13.    std::vector<cv::Point> points_dst;
14.    points_dst.push_back(cv::Point(0, 0));
15.    points_dst.push_back(cv::Point(card_size.width, 0));
16.    points_dst.push_back(cv::Point(0, card_size.height));
17.    points_dst.push_back(cv::Point(card_size.width, card_size.height));

18.    // Load an image
19.    cv::Mat original = cv::imread("data/sunglok.jpg");
20.    if (original.empty()) return -1;

21.    // Get the matched points from a user's mouse
22.    std::vector<cv::Point> points_src;
23.    cv::namedWindow("3DV Tutorial: Perspective Correction");
24.    cv::setMouseCallback("3DV Tutorial: Perspective Correction", MouseEventHandler, &points_src);
25.    while (points_src.size() < 4)
26.    {
27.        ...
28.    }
29.    if (points_src.size() < 4) return -1;

30.    // Calculate planar homography and rectify perspective distortion
31.    cv::Mat H = cv::findHomography(points_src, points_dst);
32.    cv::Mat rectify;
33.    cv::warpPerspective(original, rectify, H, card_size);
```

Example: Planar Image Stitching



```
1. #include "opencv_all.hpp"  
2. int main(void)  
3. {  
4.     // Load two images (c.f. We assume that two images have the same size and type)  
5.     cv::Mat image1 = cv::imread("data/hill01.jpg");  
6.     cv::Mat image2 = cv::imread("data/hill02.jpg");  
7.     if (image1.empty() || image2.empty()) return -1;  
8.  
9.     // Retrieve matching points  
10.    cv::Ptr<cv::FeatureDetector> fdetector = cv::ORB::create();  
11.    std::vector<cv::KeyPoint> keypoint1, keypoint2;  
12.    cv::Mat descriptor1, descriptor2;  
13.    fdetector->detectAndCompute(image1, cv::Mat(), keypoint1, descriptor1);  
14.    fdetector->detectAndCompute(image2, cv::Mat(), keypoint2, descriptor2);  
15.    cv::Ptr<cv::DescriptorMatcher> fmatcher = cv::DescriptorMatcher::create("BruteForce-Hamming");  
16.    std::vector<cv::DMatch> match;  
17.    fmatcher->match(descriptor1, descriptor2, match);  
18.  
19.    // Calculate planar homography and merge them  
20.    std::vector<cv::Point2f> points1, points2;  
21.    for (size_t i = 0; i < match.size(); i++)  
22.    {  
23.        points1.push_back(keypoint1.at(match.at(i).queryIdx).pt);  
24.        points2.push_back(keypoint2.at(match.at(i).trainIdx).pt);  
25.    }  
26.    cv::Mat H = cv::findHomography(points2, points1, cv::RANSAC);  
27.    cv::Mat merge;  
28.    cv::warpPerspective(image2, merge, H, cv::Size(image1.cols * 2, image1.rows));  
29.    merge.colRange(0, image1.cols) = image1 * 1; // Copy  
30. }
```



Example: 2D Video Stabilization



```
1. #include "opencv_all.hpp"  
2. int main(void)  
3. {  
4.     // Open a video and get the reference image and feature points  
5.     cv::VideoCapture video;  
6.     if (!video.open("data/traffic.avi")) return -1;  
7.  
8.     cv::Mat gray_ref;  
9.     video >> gray_ref;  
10.    if (gray_ref.empty())  
11.    {  
12.        video.release();  
13.        return -1;  
14.    }  
15.    if (gray_ref.channels() > 1) cv::cvtColor(gray_ref, gray_ref, cv::COLOR_RGB2GRAY);  
16.    std::vector<cv::Point2f> point_ref;  
17.    cv::goodFeaturesToTrack(gray_ref, point_ref, 2000, 0.01, 10);  
18.    if (point_ref.size() < 4)  
19.    {  
20.        video.release();  
21.        return -1;  
22.    }
```

A shaking CCTV video: data/traffic.avi





Example: 2D Video Stabilization

```
22.     // Run and show video stabilization
23.     while (true)
24.     {
25.         // Grab an image from the video
26.         cv::Mat image, gray;
27.         video >> image;
28.         if (image.empty()) break;
29.         if (image.channels() > 1) cv::cvtColor(image, gray, cv::COLOR_RGB2GRAY);
30.         else                         gray = image.clone();

31.         // Extract optical flow and calculate planar homography
32.         std::vector<cv::Point2f> point;
33.         std::vector<uchar> m_status;
34.         cv::Mat err, inlier_mask;
35.         cv::calcOpticalFlowPyrLK(gray_ref, gray, point_ref, point, m_status, err);
36.         cv::Mat H = cv::findHomography(point, point_ref, inlier_mask, cv::RANSAC);

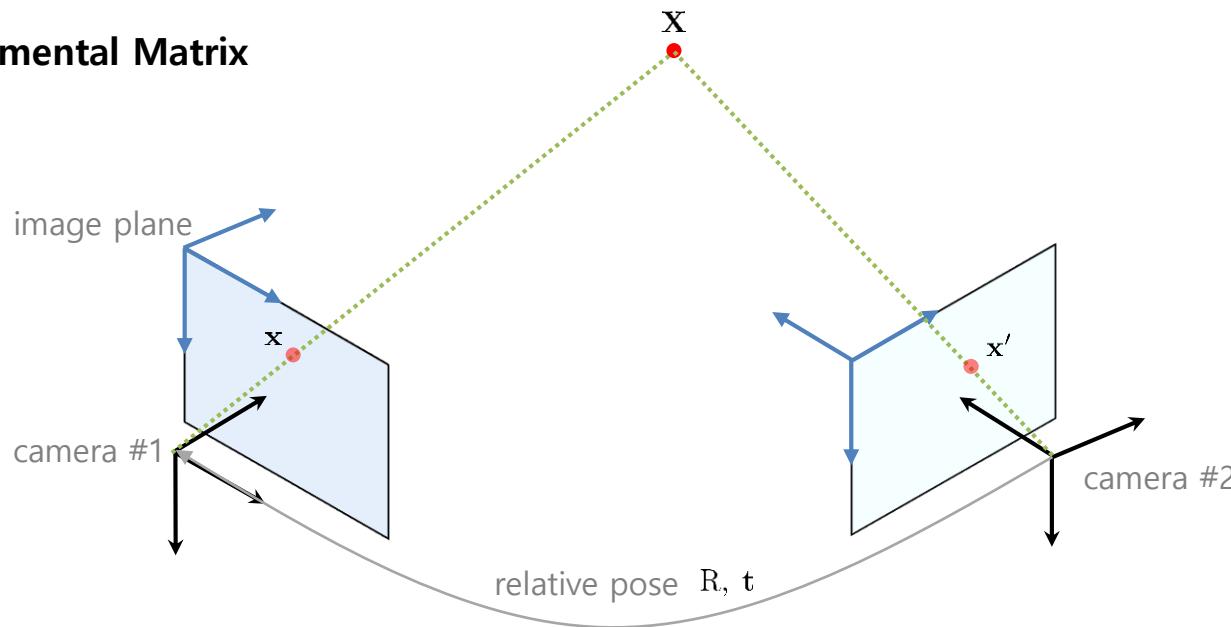
37.         // Synthesize a stabilized image
38.         cv::Mat warp;
39.         cv::warpPerspective(image, warp, H, cv::Size(image.cols, image.rows));

40.         // Show the original and rectified images together
41.         for (size_t i = 0; i < point_ref.size(); i++)
42.         {
43.             if (inlier_mask.at<uchar>(i) > 0) cv::line(image, point_ref[i], point[i], cv::Scalar(0, 0, 255));
44.             else cv::line(image, point_ref[i], point[i], cv::Scalar(0, 127, 0));
45.         }
46.         cv::hconcat(image, warp, image);
47.         cv::imshow("3DVT Tutorial: Video Stabilization", image);
48.         if (cv::waitKey(1) == 27) break; // 'ESC' key
49.     }

50.     video.release();
51.     return 0;
52. }
```

General 2D-2D Geometry (Epipolar Geometry)

- Fundamental Matrix



Epipolar Constraint

$$\mathbf{x}'^\top \mathbf{F} \mathbf{x} = 0$$

on the image plane

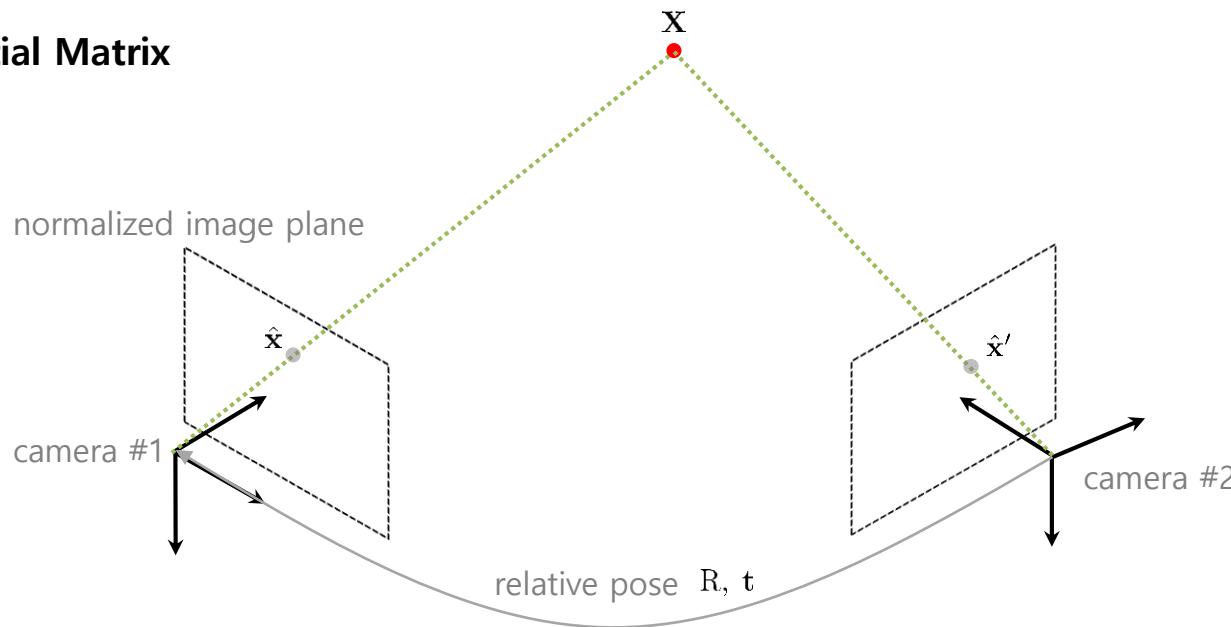
$$\mathbf{x} = \mathbf{K}\hat{\mathbf{x}}$$

$$\hat{\mathbf{x}}'^\top \mathbf{K}'^\top \mathbf{F} \mathbf{K} \hat{\mathbf{x}} = 0$$

$$\hat{\mathbf{x}}'^\top \mathbf{E} \hat{\mathbf{x}} = 0 \quad (\mathbf{E} = \mathbf{K}'^\top \mathbf{F} \mathbf{K}) \quad \text{on the normalized image plane}$$

General 2D-2D Geometry (Epipolar Geometry)

- Essential Matrix



Epipolar Constraint

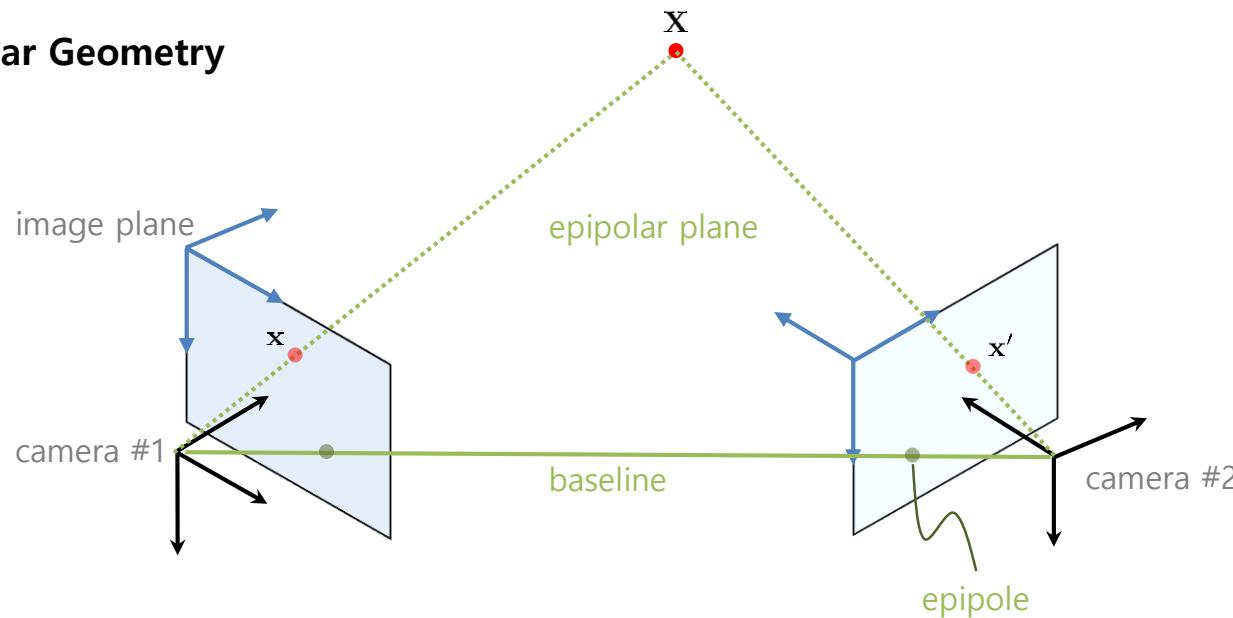
$$\hat{x}'^\top E \hat{x} = 0 \quad (E = [\mathbf{t}]_\times R)$$

c.f. $[\mathbf{t}]_\times = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$

$$\begin{aligned} \hat{x}' \cdot (\mathbf{t} \times \hat{x}') &= 0 \\ \hat{x}' \cdot (\mathbf{t} \times \hat{x}') &= \hat{x}' \cdot \mathbf{t} \times R \hat{x} \\ \hat{x}' \cdot & \\ \mathbf{t} \times \hat{x}' &= \mathbf{t} \times R \hat{x} \\ \mathbf{t} \times & \\ \hat{x}' &= R \hat{x} + \mathbf{t} \end{aligned}$$

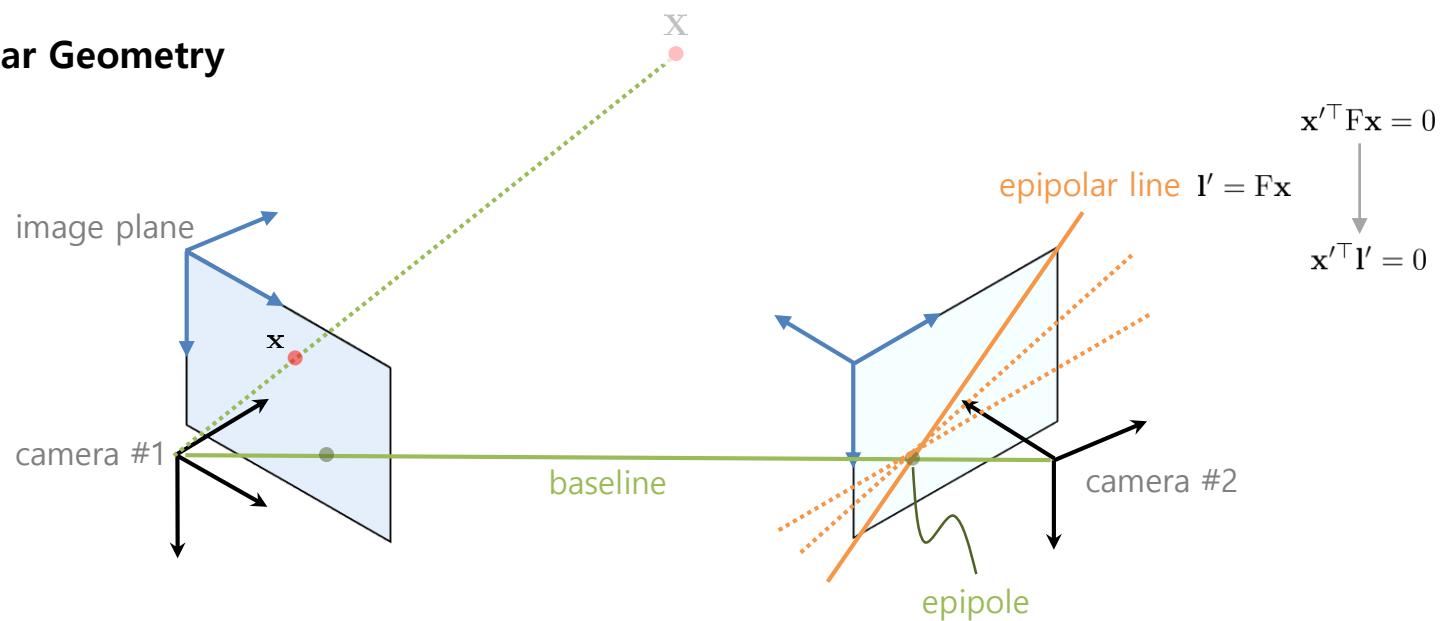
General 2D-2D Geometry (Epipolar Geometry)

- Epipolar Geometry



General 2D-2D Geometry (Epipolar Geometry)

- Epipolar Geometry

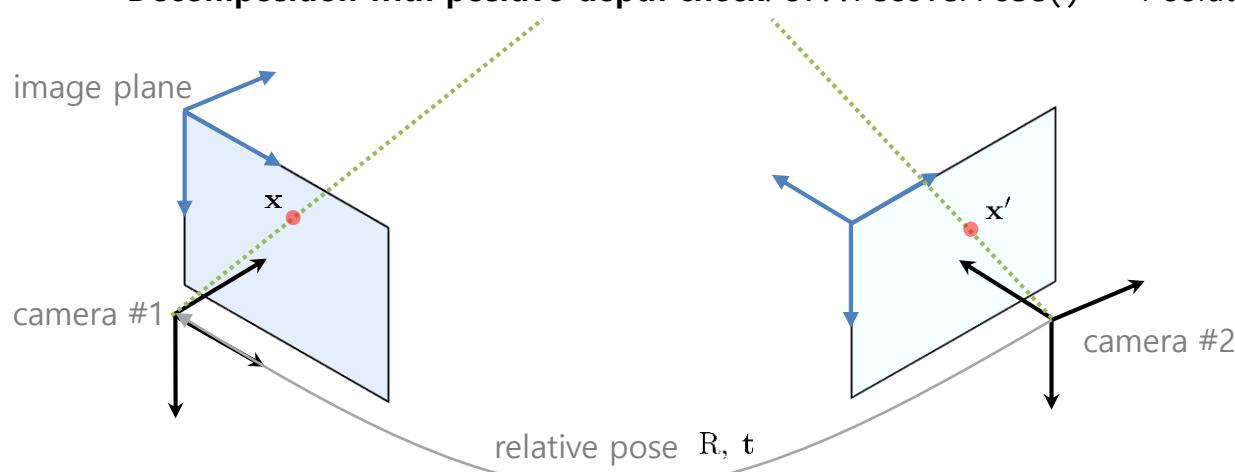
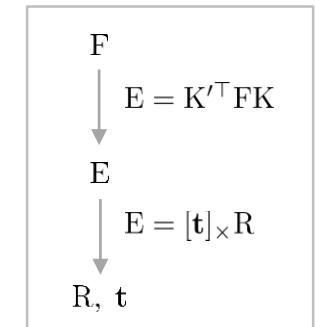


General 2D-2D Geometry (Epipolar Geometry)

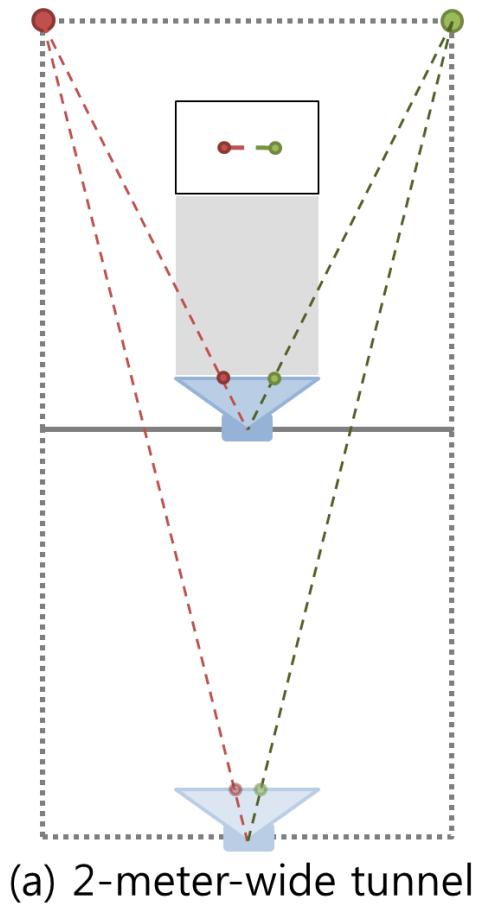
▪ Relative Camera Pose Estimation

- Known: Point correspondence $(x_1, x'_1), \dots, (x_n, x'_n)$ and camera matrices K, K'
- Unknown: **Rotation and translation** (5 DoF)
- Constraints: $n \times$ epipolar constraint $x'^\top F x = 0$
- Solutions (OpenCV)
 - **Fundamental matrix:** 7/8-point algorithm
 - **Estimation:** `cv::findFundamentalMat()` $\rightarrow 1$ solution
 - **Conversion to E:** $E = K'^\top F K$
 - **Essential matrix:** 5-point algorithm
 - **Estimation:** `cv::findEssentialMat()` $\rightarrow k$ solutions
 - **Decomposition:** `cv::decomposeEssentialMat()` $\rightarrow 4$ solutions
 - **Decomposition with positive-depth check:** `cv::recoverPose()` $\rightarrow 1$ solution

up-to scale \rightarrow scale ambiguity

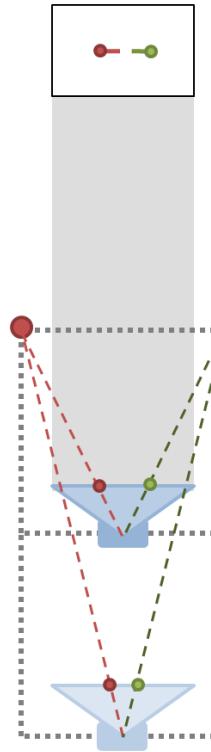


Epipolar Geometry: Scale Ambiguity



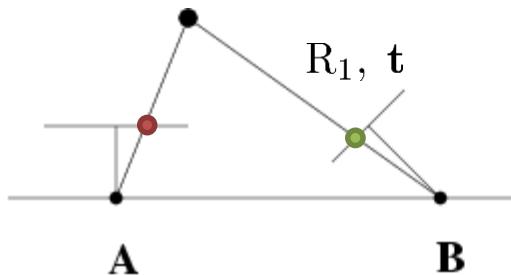
(a) 2-meter-wide tunnel

optical flow on an image

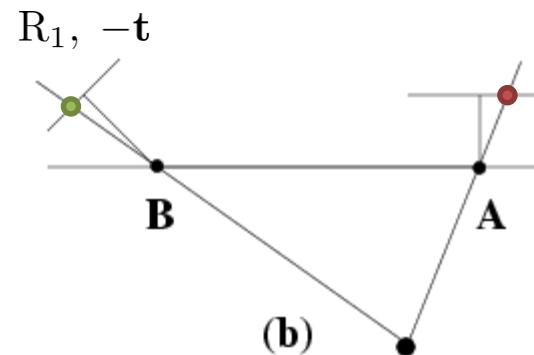


(b) 1-meter-wide tunnel

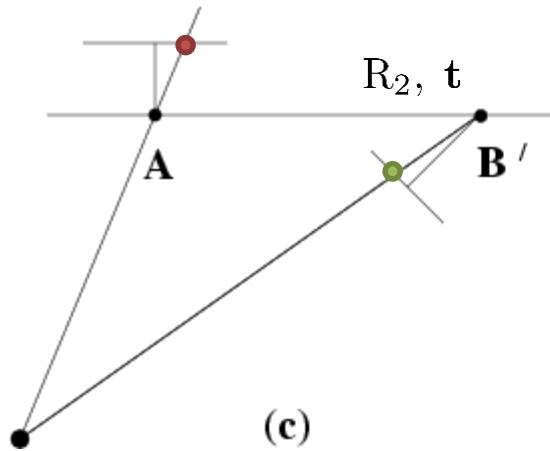
Epipolar Geometry: Relative Pose Ambiguity



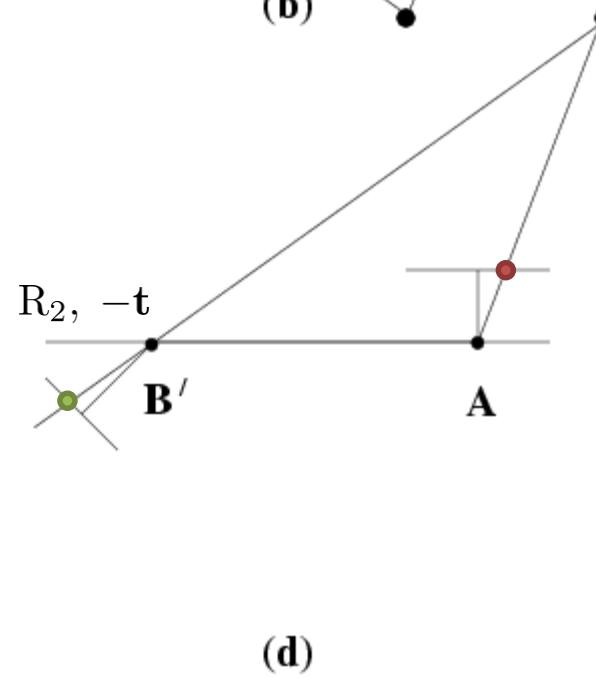
(a)



(b)



(c)



(d)

General 2D-2D Geometry (Epipolar Geometry)

▪ Relative Camera Pose Estimation

- Known: Point correspondence $(\mathbf{x}_1, \mathbf{x}'_1), \dots, (\mathbf{x}_n, \mathbf{x}'_n)$ and camera matrices \mathbf{K}, \mathbf{K}'
- Unknown: Rotation and translation (5 DoF)
- Constraints: $n \times$ epipolar constraint $\mathbf{x}'^\top \mathbf{F} \mathbf{x} = 0$
- Solutions (OpenCV)

intrinsic & extrinsic
camera parameters

- **Fundamental matrix:** 7/8-point algorithm (7 DoF)

$$\mathbf{F} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \quad \& \quad \text{rank}(\mathbf{F}) = 2 \quad \text{det}(\mathbf{F}) = 0$$

- **Estimation:** `cv::findFundamentalMat()` \rightarrow 1 solution
- **Conversion to E:** $\mathbf{E} = \mathbf{K}'^\top \mathbf{F} \mathbf{K}$
- **Degenerate cases:** No translation, correspondence from a single plane

extrinsic
camera parameters

- **Essential matrix:** 5-point algorithm (5 DoF)

$$2\mathbf{E}\mathbf{E}^\top - \text{tr}(\mathbf{E}\mathbf{E}^\top)\mathbf{E} = 0$$

- **Estimation:** `cv::findEssentialMat()` $\rightarrow k$ solutions
- **Decomposition:** `cv::decomposeEssentialMat()` \rightarrow 4 solutions
- **Decomposition with positive-depth check:** `cv::recoverPose()` \rightarrow 1 solution
- **Degenerate cases:** No translation ($\because \mathbf{E} = [\mathbf{t}]_\times \mathbf{R}$)

General 2D-2D Geometry (Epipolar Geometry)

▪ Relative Camera Pose Estimation

- Known: Point correspondence $(\mathbf{x}_1, \mathbf{x}'_1), \dots, (\mathbf{x}_n, \mathbf{x}'_n)$ and camera matrices \mathbf{K}, \mathbf{K}'
- Unknown: Rotation and translation (5 DoF)
- Constraints: $n \times$ epipolar constraint $\mathbf{x}'^\top \mathbf{F} \mathbf{x} = 0$
- Solutions (OpenCV)

intrinsic & extrinsic
camera parameters

- **Fundamental matrix:** 7/8-point algorithm (7 DoF)

$$\mathbf{F} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \quad \& \quad \text{rank}(\mathbf{F}) = 2 \quad \text{det}(\mathbf{F}) = 0$$

- **Estimation:** `cv::findFundamentalMat()` → 1 solution
- **Conversion to E:** $\mathbf{E} = \mathbf{K}'^\top \mathbf{F} \mathbf{K}$
- **Degenerate cases:** No translation, correspondence from a single plane

extrinsic
camera parameters

- **Essential matrix:** 5-point algorithm (5 DoF)

$$2\mathbf{E}\mathbf{E}^\top - \text{tr}(\mathbf{E}\mathbf{E}^\top)\mathbf{E} = 0$$

- **Estimation:** `cv::findEssentialMat()` → k solutions
- **Decomposition:** `cv::decomposeEssentialMat()` → 4 solutions
- **Decomposition with positive-depth check:** `cv::recoverPose()` → 1 solution
- **Degenerate cases:** No translation ($\because \mathbf{E} = [\mathbf{t}]_\times \mathbf{R}$)

intrinsic & extrinsic
camera parameters
+ plane normal

- **Planar homography:** 4-point algorithm (8 DoF)

- **Estimation:** `cv::findHomography()` → 1 solutions
- **Conversion to calibrated H:** $\hat{\mathbf{H}} = \mathbf{K}'^{-1} \mathbf{H} \mathbf{K}$
- **Decomposition:** `cv::decomposeHomographyMat()` → 4 solutions
- **Degenerate cases:** Correspondence not from a single plane

$$\begin{aligned} \mathbf{H} &= \mathbf{K}' \hat{\mathbf{H}} \mathbf{K}^{-1} \\ \hat{\mathbf{H}} &= \mathbf{R} + \frac{1}{d} \mathbf{t} \mathbf{n}^\top \end{aligned}$$

$$\hat{\mathbf{x}}' = \left(\mathbf{R} + \frac{1}{d} \mathbf{t} \mathbf{n}^\top \right) \hat{\mathbf{x}}$$

$$\frac{1}{d} \mathbf{n}^\top \hat{\mathbf{x}} = 1 \quad (\because n_x \hat{x} + n_y \hat{y} + n_z \hat{z} - d = 0)$$

Overview of Relative Pose Estimation

| Items | General 2D-2D Geometry | Planar 2D-2D Geometry |
|----------------------------|---|---|
| On Image Planes | Model Fundamental Matrix (7 DoF) | Planar Homography (8 DoF) |
| | Formulation $F = K'^{-\top} E K^{-1}$ | $E = K'^\top F K$ |
| | Estimation <ul style="list-style-type: none"> - 7-point algorithm ($n \geq 7$) $\rightarrow k$ solution - (normalized) 8-point algorithm $\rightarrow 1$ solution - <code>cv::findFundamentalMat()</code> | $H = K' \hat{H} K^{-1}$ |
| | Input - point correspondence [px] (x_i, x'_i) | - point correspondence [px] on a single plane (x_i, x'_i) |
| | Degenerate Cases - no translational motion - correspondence from a single plane | - correspondence not from a single plane |
| | Decomposition to R, t - convert to an essential matrix and decompose it | - <code>cv::decomposeHomographyMat()</code> |
| On Normalized Image Planes | Model Essentials Matrix (5 DoF) | (Calibrated) Planar Homography (8 DoF) |
| | Formulation $E = [\mathbf{t}]_{\times} R$ | $\hat{H} = R + \frac{1}{d} t n^\top$ |
| | Estimation <ul style="list-style-type: none"> - 5-point algorithm ($n \geq 5$) $\rightarrow k$ solution - <code>cv::findEssentialMat()</code> | 4-point algorithm ($n \geq 4$) $\rightarrow 1$ solution |
| | Input - calibrated point correspondence [m] (\hat{x}_i, \hat{x}'_i) | - calibrated point correspondence [m] on a single plane (\hat{x}_i, \hat{x}'_i) |
| | Degenerate Cases - no translational motion | - correspondence not from a single plane |
| | Decomposition to R, t - <code>cv::decomposeEssentialMat()</code> - <code>cv::recoverPose()</code> | - <code>cv::decomposeHomographyMat()</code> with $K = I_{3 \times 3}$ |

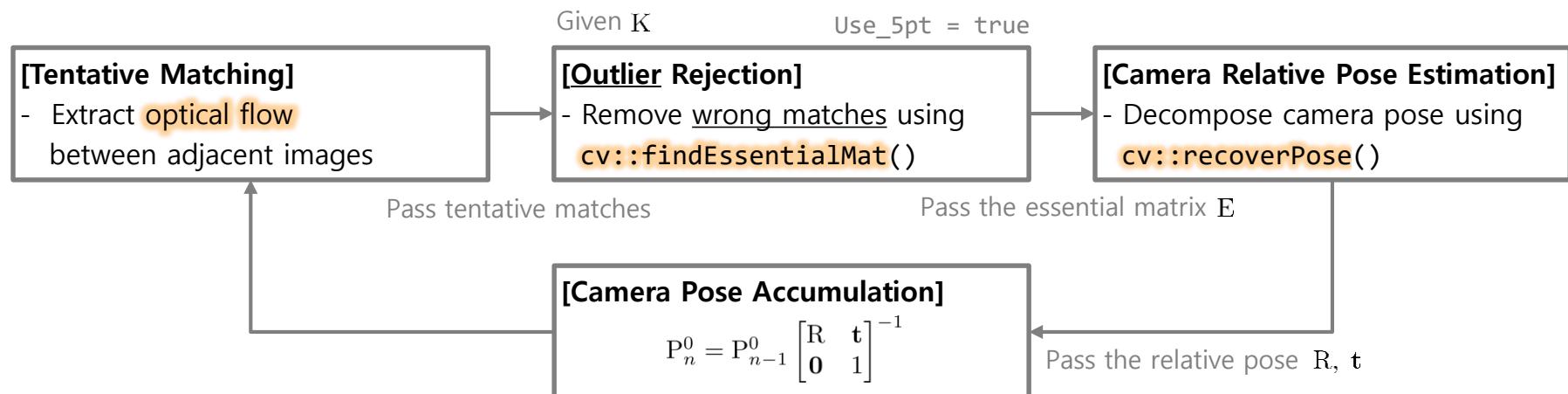
Example: Monocular Visual Odometry (Epipolar)



KITTI Odometry Dataset (#: 00, Left): data/KITTI_07_L/%06d.png



relative pose R, t





Example: Monocular Visual Odometry (Epipolar)

```
1. #include "opencv_all.hpp"

2. int main(void)
3. {
4.     bool use_5pt = false;
5.     double camera_focal = 718.8560;
6.     cv::Point2d camera_center(607.1928, 185.2157);

7.     // Open a file to write camera trajectory
8.     FILE* camera_traj = fopen("visual_odometry_epipolar.xyz", "wt");
9.     if (camera_traj == NULL) return -1;

10.    // Open a video and get the initial image
11.    cv::VideoCapture video;
12.    if (!video.open("data/KITTI_00_L/%06d.png")) return -1;

13.    cv::Mat gray_prev;
14.    video >> gray_prev;
15.    ...

16.    // Run and store monocular visual odometry
17.    cv::Mat camera_pose = cv::Mat::eye(4, 4, CV_64F);
18.    while (true)
19.    {
20.        // Grab an image from the video
21.        cv::Mat image, gray;
22.        video >> image;
23.        . .

24.        // Extract optical flow
25.        std::vector<cv::Point2f> point_prev, point;
26.        cv::goodFeaturesToTrack(gray_prev, point_prev, 2000, 0.01, 10);
27.        std::vector<uchar> m_status;
28.        cv::Mat err;
29.        cv::calcOpticalFlowPyrLK(gray_prev, gray, point_prev, point, m_status, err);
30.        gray_prev = gray;
```

Example: Monocular Visual Odometry (Epipolar)

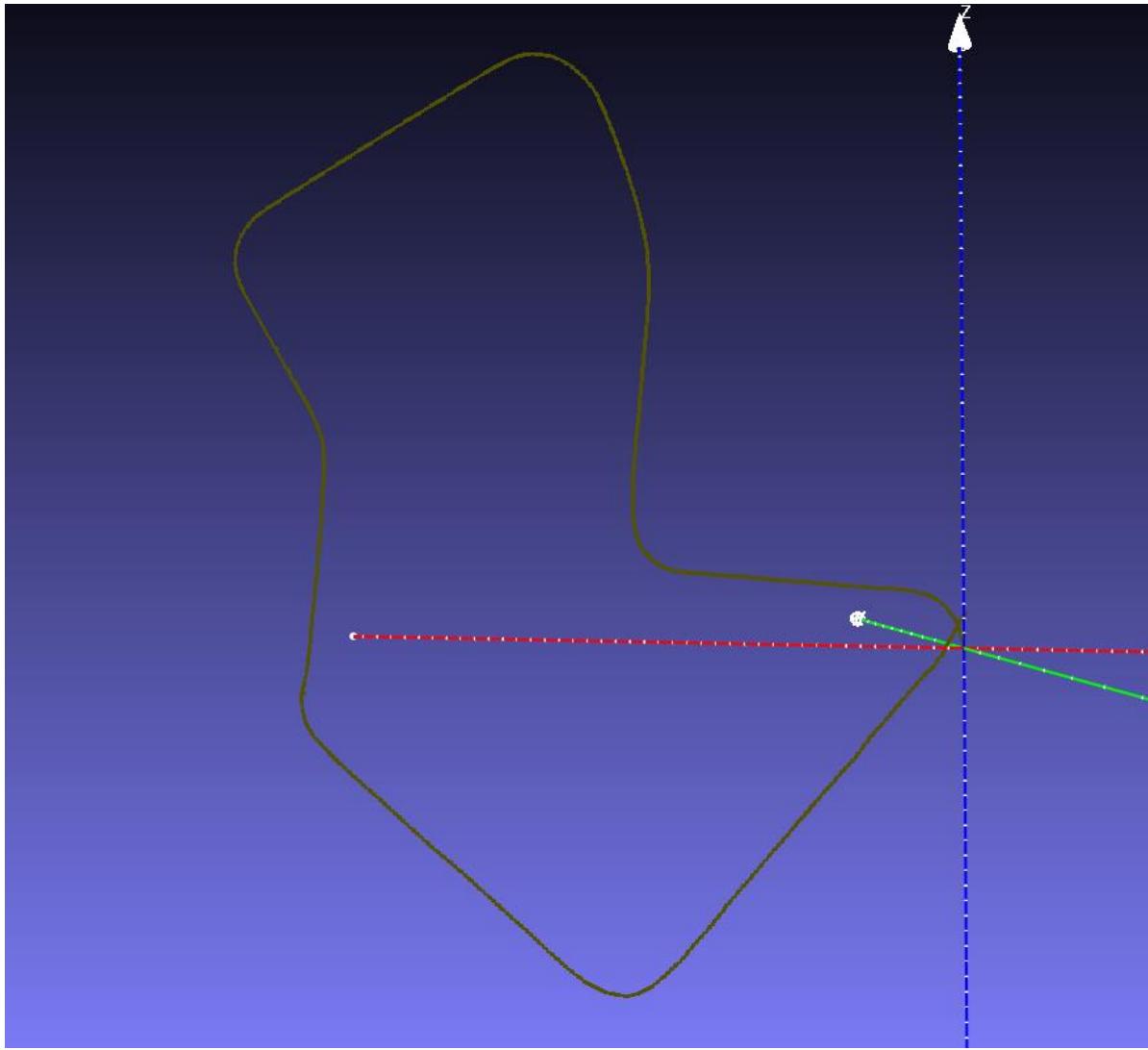


```
31.     // Calculate relative pose
32.     cv::Mat E, inlier_mask;
33.     if (use_5pt)
34.     {
35.         E = cv::findEssentialMat(point_prev, point, camera_focal, camera_center, cv::RANSAC, 0.99, 1,
36.                                   inlier_mask);
37.     }
38.     else
39.     {
40.         cv::Mat F = cv::findFundamentalMat(point_prev, point, cv::FM_RANSAC, 1, 0.99, inlier_mask);
41.         cv::Mat K = (cv::Mat<double>(3, 3) << camera_focal, 0, camera_center.x, 0, camera_focal,
42.                               camera_center.y, 0, 0, 1);
43.         E = K.t() * F * K;
44.     }
45.     cv::Mat R, t;
46.     int inlier_num = cv::recoverPose(E, point_prev, point, R, t, camera_focal, camera_center, inlier_mask);

47.     // Accumulate pose
48.     cv::Mat T = cv::Mat::eye(4, 4, R.type());
49.     T(cv::Rect(0, 0, 3, 3)) = R * 1.0;
50.     T.col(3).rowRange(0, 3) = t * 1.0;
51.     camera_pose = camera_pose * T.inv();    ::  $P_n^0 = P_{n-1}^0 \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}^{-1}$ 
52.     // Show the image and write camera pose
53.     if (image.channels() < 3) cv::cvtColor(image, image, CV_GRAY2RGB);
54.     for (size_t i = 0; i < point_prev.size(); i++)
55.     {
56.         if (inlier_mask.at<uchar>(i) > 0) cv::line(image, point_prev[i], point[i], cv::Scalar(0, 0, 255));
57.         else cv::line(image, point_prev[i], point[i], cv::Scalar(0, 127, 0));
58.     }
59.     cv::imshow("3DVT Tutorial: Monocular Visual Odometry", image);
60.     fprintf(camera_traj, "%f %f %f\n", camera_pose.at<double>(0, 3), camera_pose.at<double>(1, 3),
61.             camera_pose.at<double>(2, 3));
62.     if (cv::waitKey(1) == 27) break; // 'ESC' key
63. }
64. }
```



Example: Monocular Visual Odometry (Epipolar)



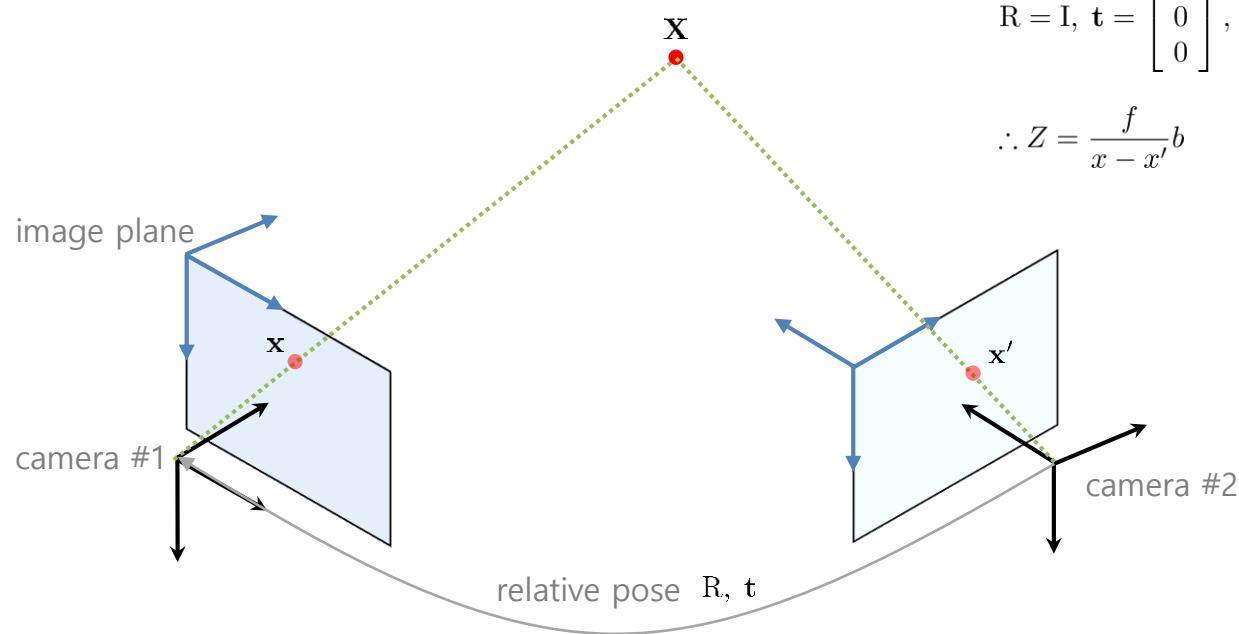
Result: visual_odometry_epipolar.xyz

General 2D-2D Geometry (Epipolar Geometry)

- **Relative Point Localization (Triangulation)**

- Known: Point correspondence, camera matrices, and relative pose
- Unknown: **Position of a 3D point** (3 DoF)
- Constraints: $2 \times$ projection $\mathbf{x} = \mathbf{K}[\mathbf{I} | \mathbf{0}] \mathbf{X}$, $\mathbf{x}' = \mathbf{K}'[\mathbf{R} | \mathbf{t}] \mathbf{X}$
- Solution (OpenCV): `cv::triangulatePoints()`

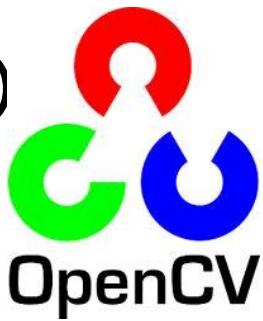
Special case) Stereo cameras



$$\mathbf{R} = \mathbf{I}, \mathbf{t} = \begin{bmatrix} -b \\ 0 \\ 0 \end{bmatrix}, \mathbf{K} = \mathbf{K}' = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

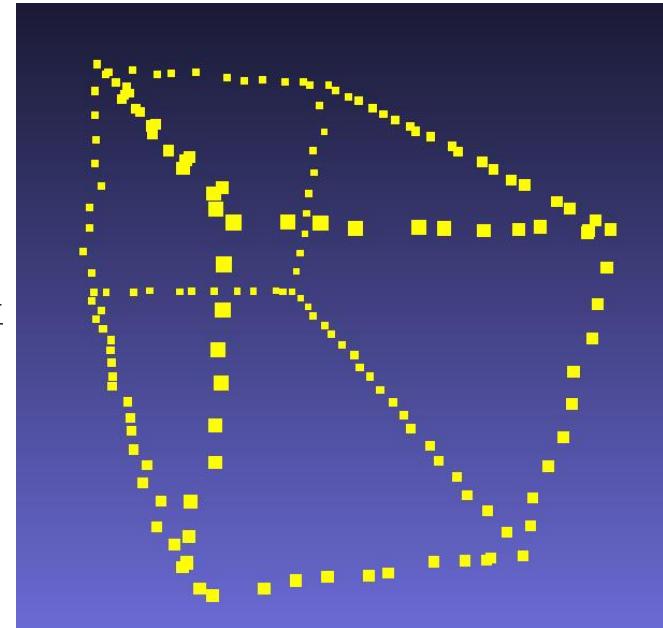
$$\therefore Z = \frac{f}{x - x'} b$$

Example: Triangulation (Two-view Reconstruction)

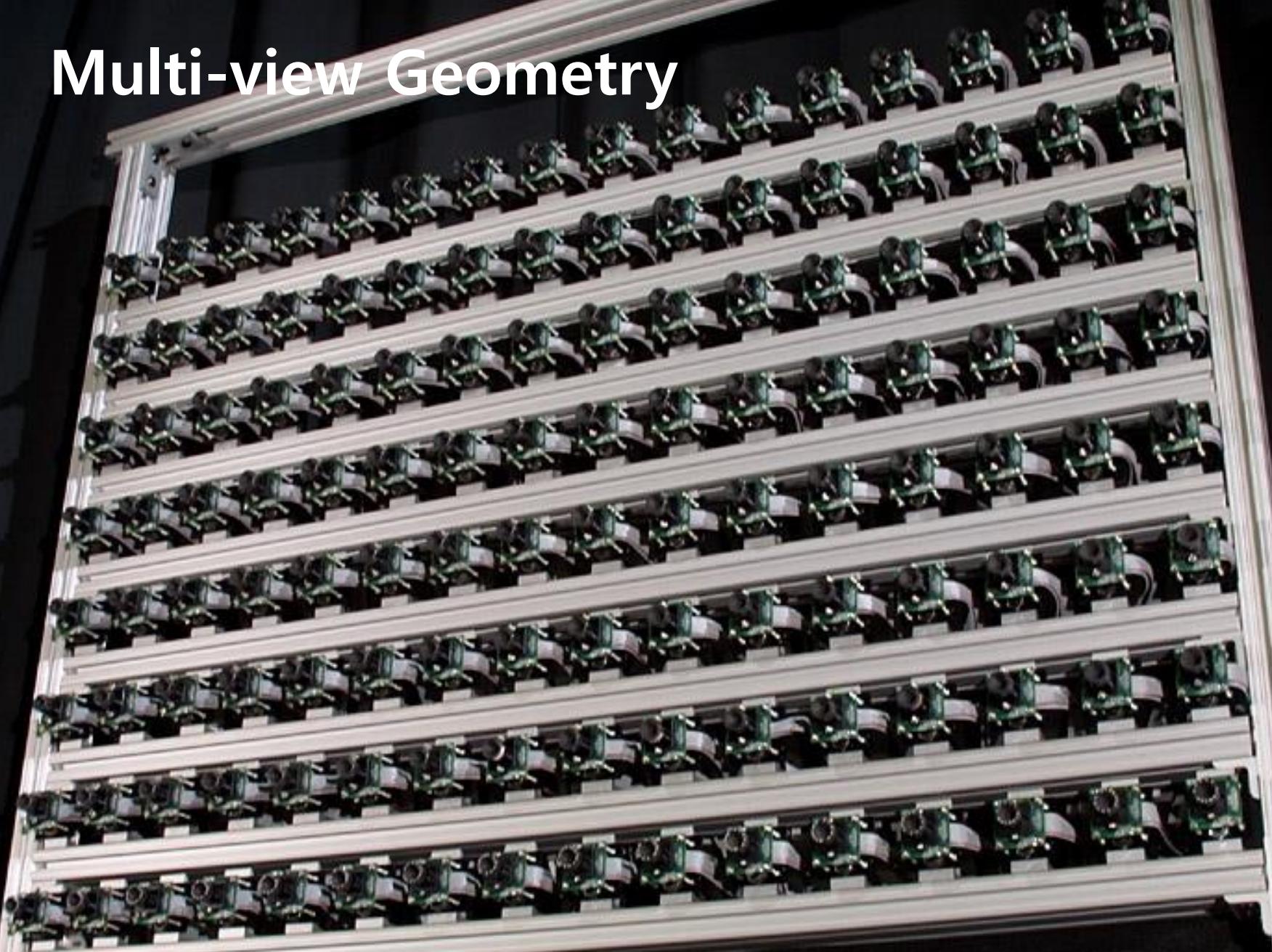


```
1. #include "opencv_all.hpp"
2.
3. int main(void)
4. {
5.     double camera_focal = 1000;
6.     cv::Point2d camera_center(320, 240);
7.
8.     // Load two views of 'box.xyz'
9.     std::vector<cv::Point2d> points0, points1;
10.    FILE* fin0 = fopen("image_formation0.xyz", "rt");
11.    FILE* fin1 = fopen("image_formation1.xyz", "rt");
12.    ...
13.
14.    // Estimate relative pose of two views
15.    cv::Mat F = cv::findFundamentalMat(points0, points1, cv::FM_8POINT);
16.    cv::Mat K = (cv::Mat<double>(3, 3) << camera_focal, ...);
17.    cv::Mat E = K.t() * F * K;
18.    cv::Mat R, t;
19.    cv::recoverPose(E, points0, points1, K, R, t);
20.
21.    // Reconstruct 3D points of 'box.xyz' (triangulation)
22.    cv::Mat P0 = K * cv::Mat::eye(3, 4, CV_64F);           :: x = K[I|0]X
23.    cv::Mat Rt, X;
24.    cv::hconcat(R, t, Rt);                                :: x' = K'[R|t]X
25.    cv::Mat P1 = K * Rt;
26.    cv::triangulatePoints(P0, P1, points0, points1, X);
27.    X.row(0) = X.row(0) / X.row(3);
28.    X.row(1) = X.row(1) / X.row(3);
29.    X.row(2) = X.row(2) / X.row(3);
30.    X.row(3) = 1;
31.
32.    // Store the 3D points
33.    FILE* fout = fopen("triangulation.xyz", "wt");
34.    if (fout == NULL) return -1;
35.    for (int c = 0; c < X.cols; c++)
36.        fprintf(fout, "%f %f %f\n", X.at<double>(0, c), X.at<double>(1, c), X.at<double>(2, c));
37.    fclose(fout);
38.    return 0;
39. }
```

Result: triangulation.xyz



Multi-view Geometry



The Stanford Multi-Camera Array,
Stanford Computer Graphics Lab.

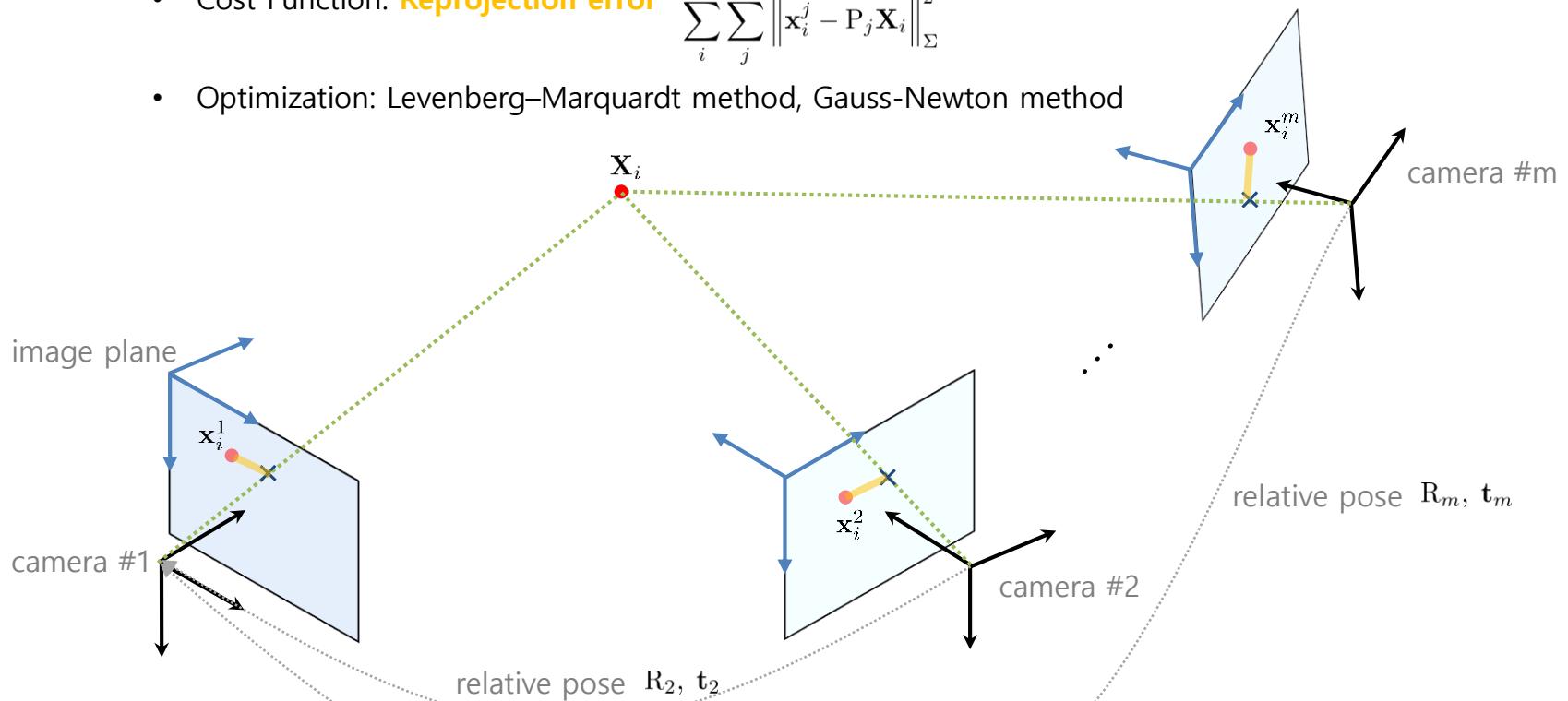
Multi-view Geometry



The Stanford Multi-Camera Array,
Stanford Computer Graphics Lab.

Bundle Adjustment

- **Bundle Adjustment** c.f. initial values
 - Known: Point correspondence, camera matrices, position of 3D points, and each camera's relative pose
 - Unknown: **Position of 3D points and each camera's relative pose (6n + 3m DoF)**
 - Constraints: $n \times m \times$ projection $\mathbf{x}_i^j = \mathbf{P}_j \mathbf{X}_i = \mathbf{K}_j [\mathbf{R}_j \mid \mathbf{t}_j] \mathbf{X}_i$
 - Solution: **Non-linear least-square optimization**
 - Cost Function: **Reprojection error** $\sum_i^n \sum_j^m \left\| \mathbf{x}_i^j - \mathbf{P}_j \mathbf{X}_i \right\|_{\Sigma}^2$
 - Optimization: Levenberg–Marquardt method, Gauss–Newton method



Bundle Adjustment

- **Bundle Adjustment** c.f. initial values
 - Known: Point correspondence, camera matrices, position of 3D points, and each camera's relative pose
 - Unknown: **Position of 3D points and each camera's relative pose** ($6n + 3m$ DoF)
 - Constraints: $n \times m \times$ projection $\mathbf{x}_i^j = \mathbf{P}_j \mathbf{X}_i = \mathbf{K}_j [\mathbf{R}_j \mid \mathbf{t}_j] \mathbf{X}_i$
 - Solution: **Non-linear least-square optimization**
 - Cost Function: **Reprojection error** $\sum_i^n \sum_j^m v_i^j \left\| \mathbf{x}_i^j - \mathbf{P}_j \mathbf{X}_i \right\|_{\Sigma}^2$ where $v_i^j = \begin{cases} 1 & \text{if } \mathbf{x}_i^j \text{ is visible} \\ 0 & \text{otherwise} \end{cases}$
 - Optimization: Levenberg–Marquardt method, Gauss–Newton method

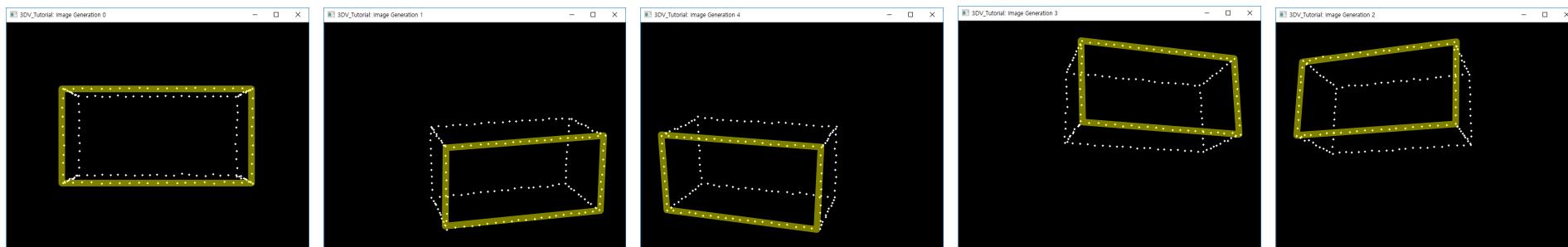
- **Bundle Adjustment and Optimization Tools**
 - OpenCV: No function (but [cvSBA](#) is available as a [SBA](#) wrapper.)
 - Bundle adjustment: [SBA](#) (Sparse Bundle Adjustment), [SSBA](#) (Simple Sparse Bundle Adjustment), [pba](#) (Multicore Bundle Adjustment)
 - Optimization: [g2o](#) (General Graph Optimization), [iSAM](#) (Incremental Smoothing and Mapping), [GTSAM](#), [Ceres Solver](#) (Google)

Example: Bundle Adjustment (Global)



[Known]

- **Intrinsic parameters:** camera matrices K_j (and also distortion coefficients)
- **2D point correspondence** \mathbf{x}_i^j



[Unknown]

- **Extrinsic parameters:** camera poses

$$\mathbf{R}_0 = \mathbf{I}_{3 \times 3}$$

$$\mathbf{t}_0 = [0, 0, 0]^\top$$

$$\mathbf{R}_1 = \mathbf{I}_{3 \times 3}$$

$$\mathbf{t}_1 = [0, 0, 0]^\top$$

$$\mathbf{R}_2 = \mathbf{I}_{3 \times 3}$$

$$\mathbf{t}_2 = [0, 0, 0]^\top$$

$$\mathbf{R}_3 = \mathbf{I}_{3 \times 3}$$

$$\mathbf{t}_3 = [0, 0, 0]^\top$$

$$\mathbf{R}_4 = \mathbf{I}_{3 \times 3}$$

$$\mathbf{t}_4 = [0, 0, 0]^\top$$

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

cvsba (non-linear least-square optimization)

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

Example: Bundle Adjustment (Global)



```
1. #include "opencv_all.hpp"
2. #include "cvbsa.h"

3. int main(void)
4. {
5.     // Load multiple views of 'box.xyz'
6.     std::vector<std::vector<cv::Point2d> > xs;
7.     ...

8.     // Assume that all cameras have the same and known camera matrix
9.     // Assume that all feature points are visible on all views
10.    cv::Mat K = (cv::Mat<double>(3, 3) << camera_focal, 0, camera_center.x, 0, camera_focal, camera_center.y, 0, 0, 1);
11.    cv::Mat dist_coeff = cv::Mat::zeros(5, 1, CV_64F);
12.    std::vector<int> visible_all(xs.front().size(), 1);

13.    // Initialize each camera projection matrix
14.    std::vector<cv::Mat> Ks, dist_coeffs, Rs, ts;
15.    std::vector<std::vector<int> > visibility;
16.    for (int i = 0; i < n_views; i++)
17.    {
18.        visibility.push_back(visible_all);
19.        Ks.push_back(K.clone());                                // K for all cameras
20.        dist_coeffs.push_back(cv::Mat::zeros(5, 1, CV_64F)); // dist_coeff for all cameras
21.        Rs.push_back(cv::Mat::eye(3, 3, CV_64F));           // R for all cameras
22.        ts.push_back(cv::Mat::zeros(3, 1, CV_64F));         // t for all cameras
23.    }

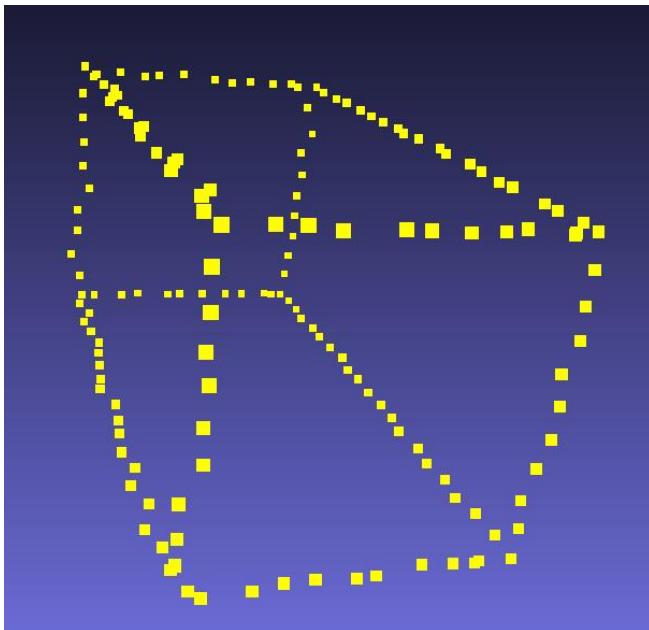
24.    // Initialize 3D points
25.    std::vector<cv::Point3d> Xs;
26.    Xs.resize(xs.front().size(), cv::Point3d(0, 0, 5.5));

27.    // Optimize camera pose and 3D points
28.    try
29.    {
30.        cvbsa::Sba sba;
31.        cvbsa::Sba::Params param;
32.        param.type = cvbsa::Sba::MOTIONSTRUCTURE;
33.        ...
34.        double error = sba.run(Xs, xs, visibility, Ks, Rs, ts, dist_coeffs);
35.    }
```

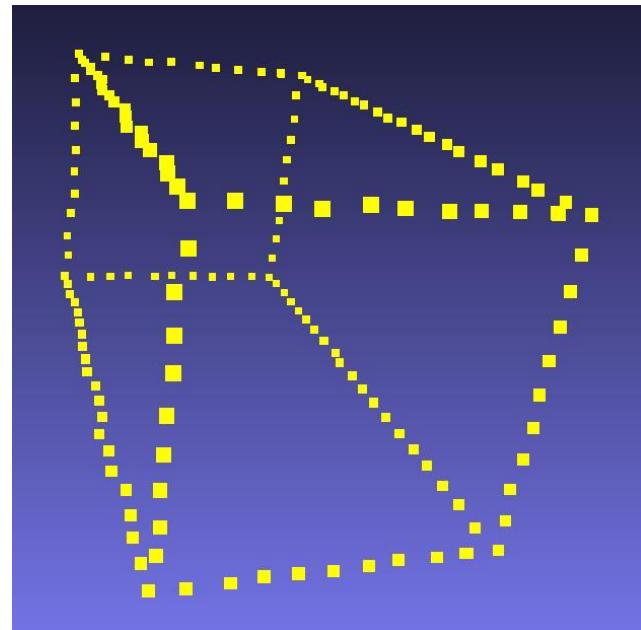
Example: Bundle Adjustment (Global)



Result: triangulation.xyz



Result: bundle_adjustment_global.xyz

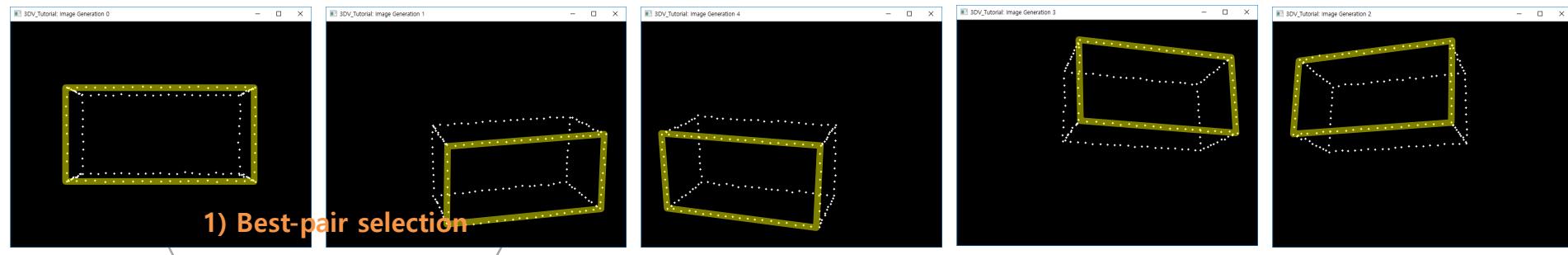


Example: Bundle Adjustment (Incremental)



[Known]

- Intrinsic parameters: camera matrices K_j (and also distortion coefficients)
- 2D point correspondence x_i^j



1) Best-pair selection

[Unknown]

- Extrinsic parameters: camera poses

$$R_0 = I_{3 \times 3}$$

$$t_0 = [0, 0, 0]^\top$$

$$R_1, t_1$$

2) Epipolar geometry

$$R_2, t_2$$

$$R_3, t_3$$

$$R_4, t_4$$

- 3D points X_i

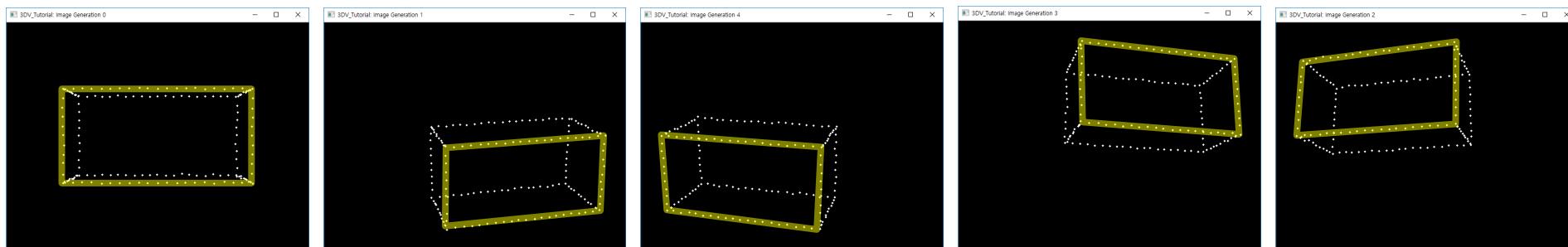
3) Triangulation

Example: Bundle Adjustment (Incremental)



[Known]

- **Intrinsic parameters:** camera matrices K_j (and also distortion coefficients)
- **2D point correspondence** x_i^j



[Unknown]

- **Extrinsic parameters:** camera poses

$$\begin{aligned} R_0 &= I_{3 \times 3} \\ t_0 &= [0, 0, 0]^\top \end{aligned}$$

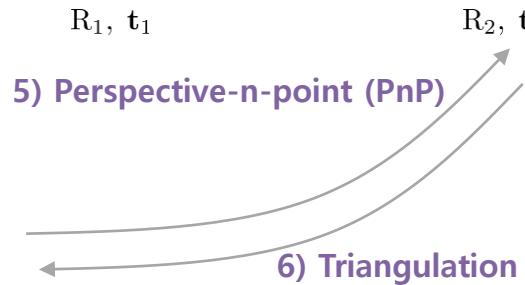
$$R_1, t_1$$

$$R_2, t_2$$

$$R_3, t_3$$

$$R_4, t_4$$

- **3D points** X_i

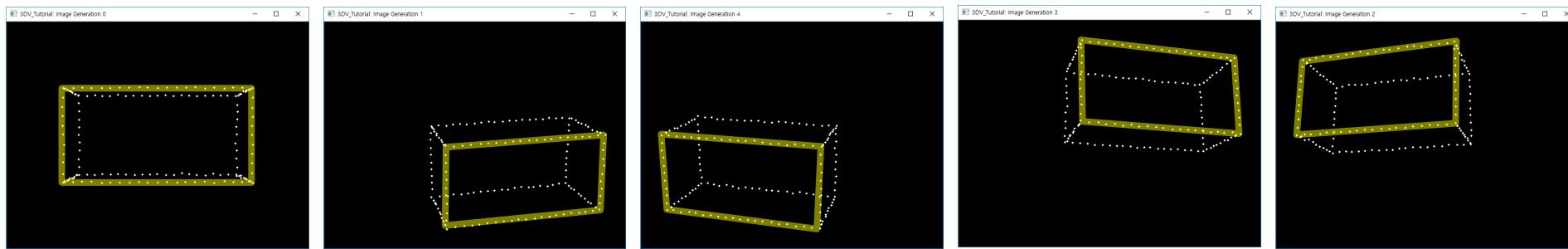


Example: Bundle Adjustment (Incremental)



[Known]

- **Intrinsic parameters:** camera matrices K_j (and also distortion coefficients)
- **2D point correspondence** x_i^j



[Unknown]

- **Extrinsic parameters:** camera poses

4) Next-view selection

Repeat 4 – 7 until the last image

$$R_0 = I_{3 \times 3}$$
$$t_0 = [0, 0, 0]^\top$$

- 3D points X_i

$$R_1, t_1$$

$$R_2, t_2$$

$$R_3, t_3$$

$$R_4, t_4$$

5) Perspective-n-point (PnP)

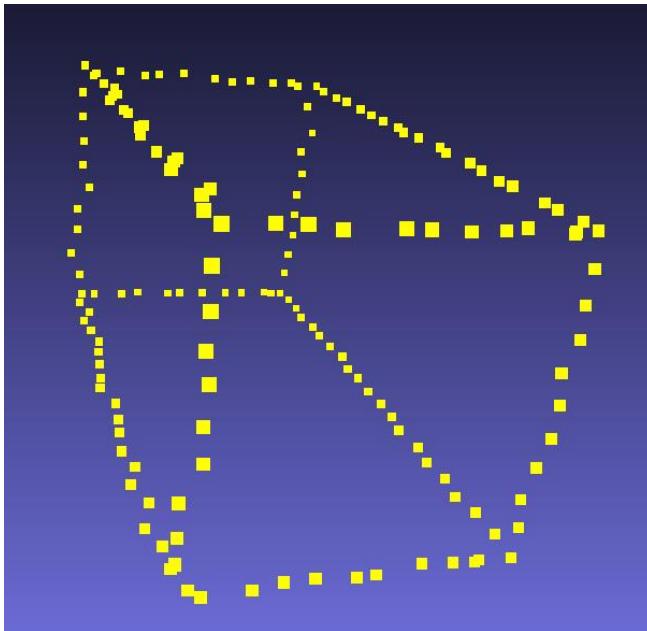
6) Triangulation

7) Bundle Adjustment

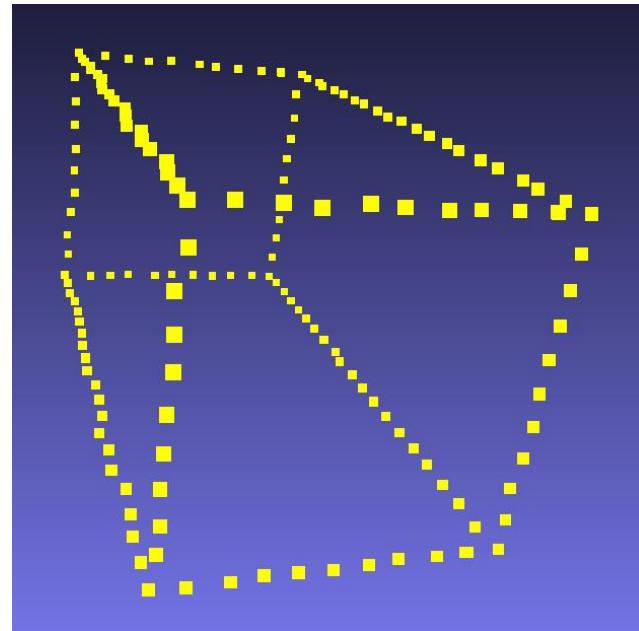
Example: Bundle Adjustment (Incremental)



Result: triangulation.xyz



Result: bundle_adjustment_global/_inc.xyz



[bundle_adjustment_global.xyz]

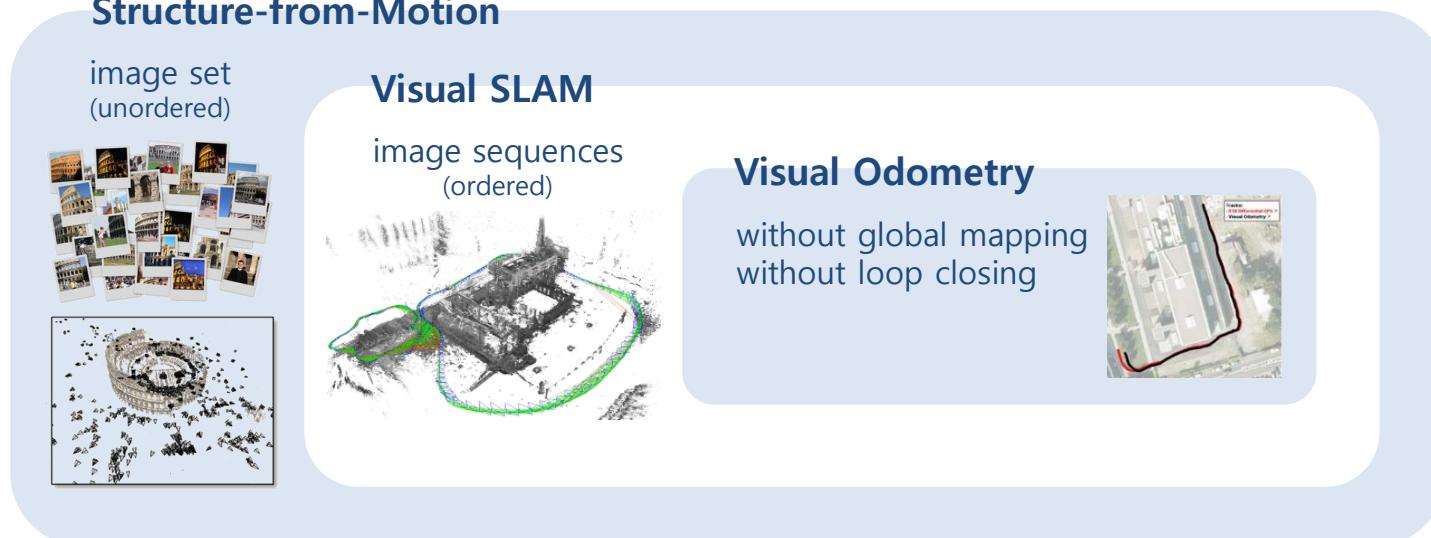
- # of iterations: 12
- Reprojection error: 1.409 ← 32970.3

[bundle_adjustment_inc.xyz]

- # of iterations: 10 + 8 + 8
- Reprojection error: 1.409 ← 3.604

Applications

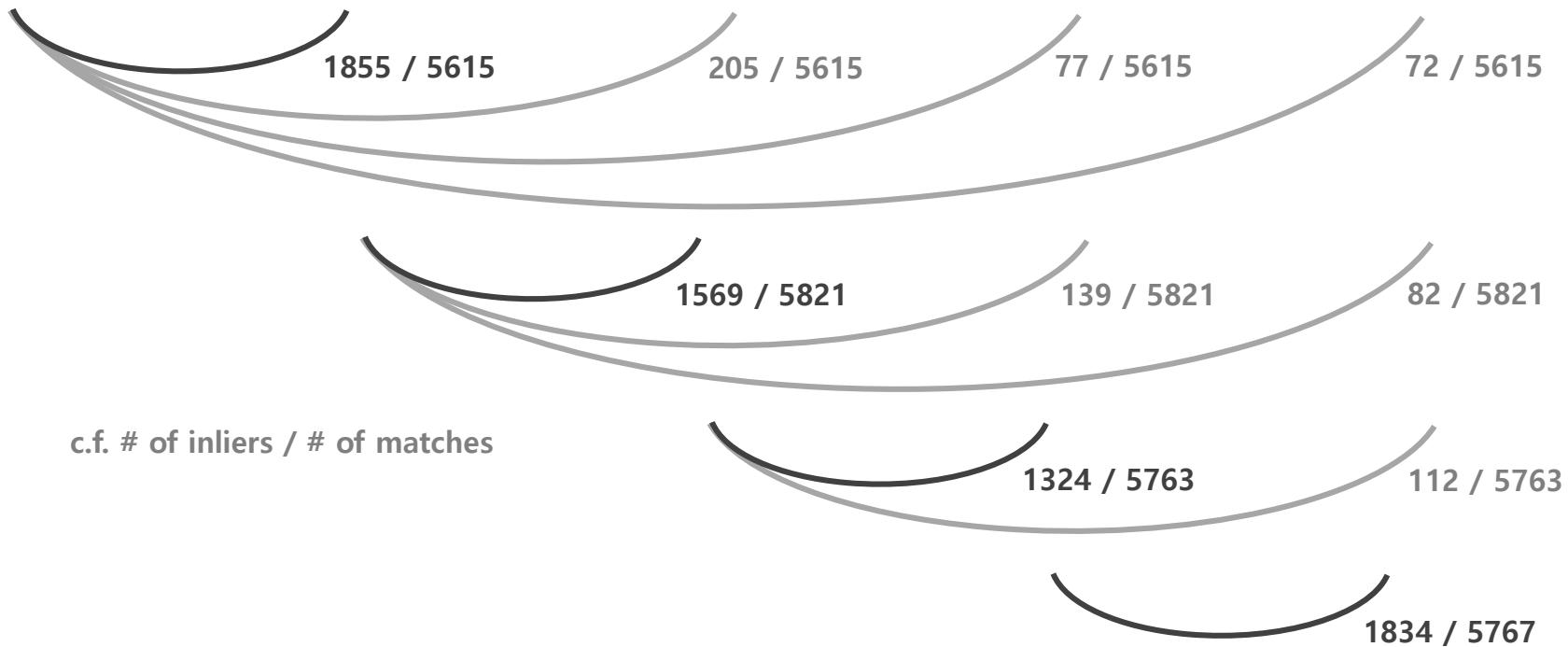
- **Structure-from-Motion (SfM)** → 3D Reconstruction, Photo Browsing
 - [Bundler](#), [COLMAP](#), [MVE](#), [Theia](#), [CMP-MVS](#), [OpenSfM](#), [Toy SfM](#) / [VisualSfM](#) (GUI, binary only)
 - [Photo Toursim](#), [PhotoSynth](#), [PhotoCity](#), [Building Rome in a Day](#), [Building Rome on a Cloudless Day](#)
- **Visual SLAM** → Augmented Reality, Navigation (Mapping and Localization)
 - [PTAM](#) (Parallel Tracking and Mapping), [ORB-SLAM](#) / [DTAM](#) (Dense Tracking and Mapping), [LSD-SLAM](#)
 - [ORB-SLAM for Windows](#)
- **Visual Odometry** → Navigation (Localization)
 - [LIBVISO2](#) (C++ Library for Visual Odometry 2) / [SVO](#) (Semi-direct Monocular Visual Odometry)



Example: Global Structure-from-Motion



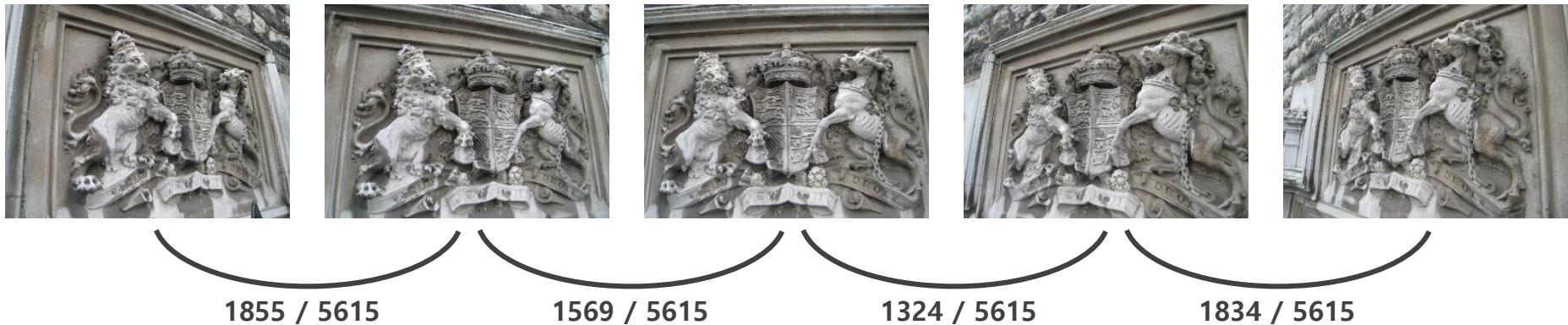
0) Build a viewing graph (well-matched pairs) while finding inliers



Example: Global Structure-from-Motion



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



1) Arrange 3D points, their observation and visibility $\mathbf{X}_i = [0, 0, 5]^\top$

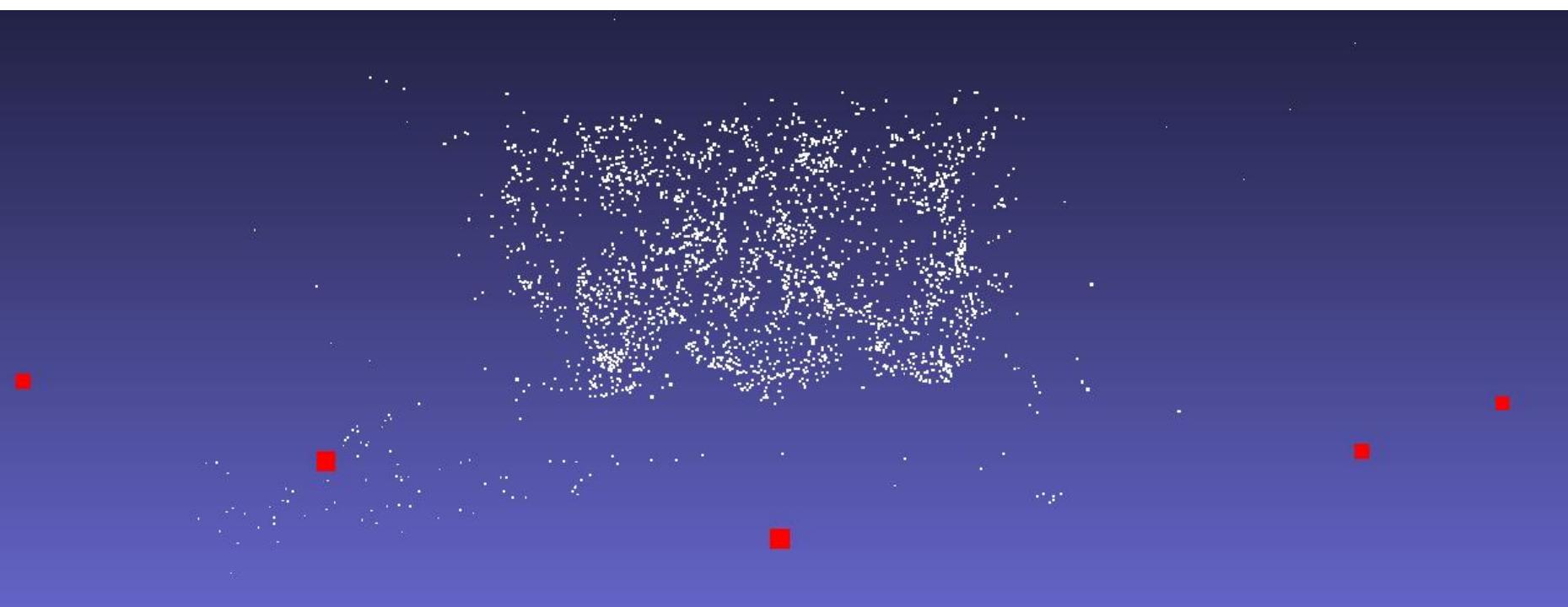
2) Initialize each camera projection matrix

3) Optimize camera pose and 3D points

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

cvsba (non-linear least-square optimization)

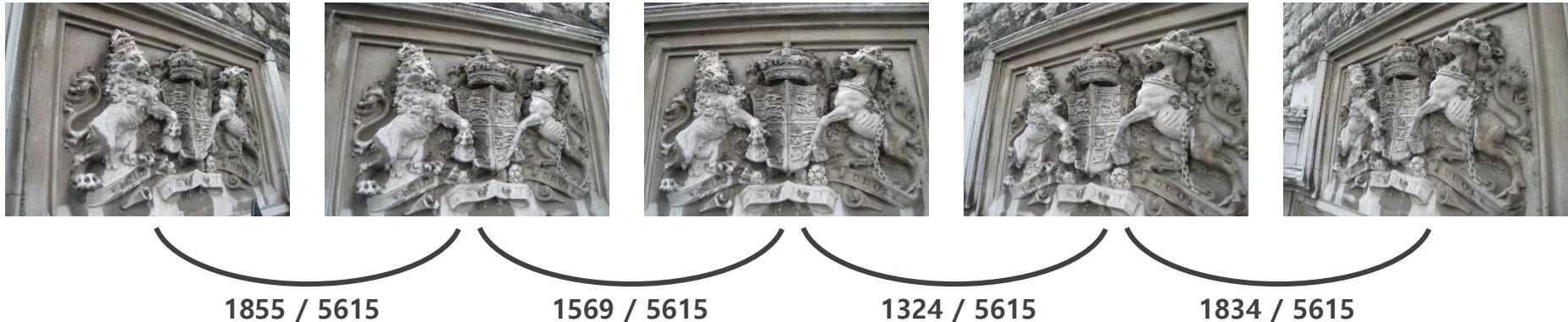
Example: Global Structure-from-Motion



Example: Incremental Structure-from-Motion



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

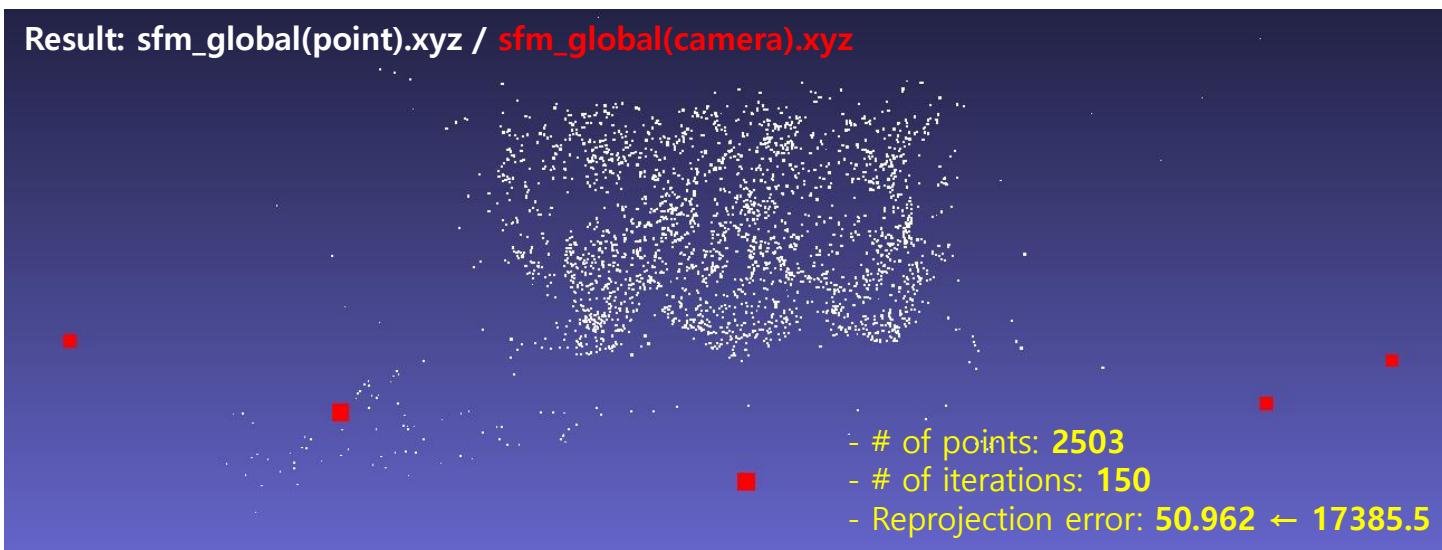


- 1) Select the best pair
- 2) Estimate relative pose from the best two views (epipolar geometry)
- 3) Reconstruct 3D points of the best two views (triangulation)
- 4) Select the next image to add
Separate points into known and unknown for PnP (known) and triangulation (unknown)
- 5) Estimate relative pose of the next view (PnP)
- 6) Reconstruct newly observed 3D points (triangulation)
- 7) Optimize camera pose and 3D points (bundle adjustment)

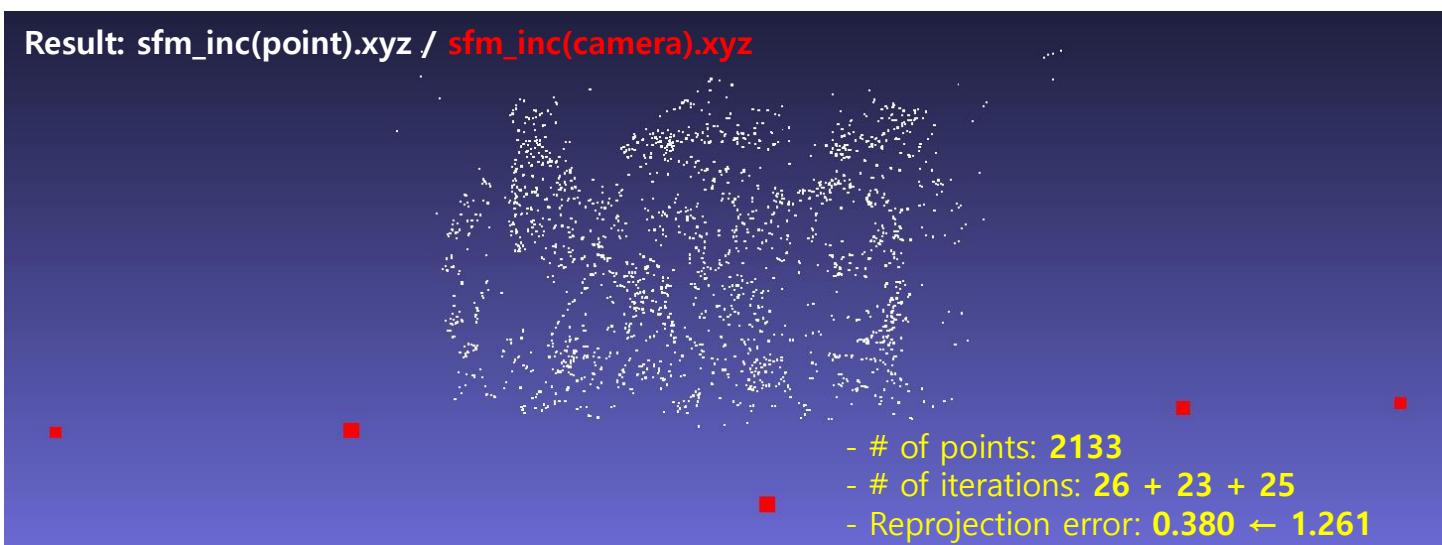
Example: Incremental Structure-from-Motion



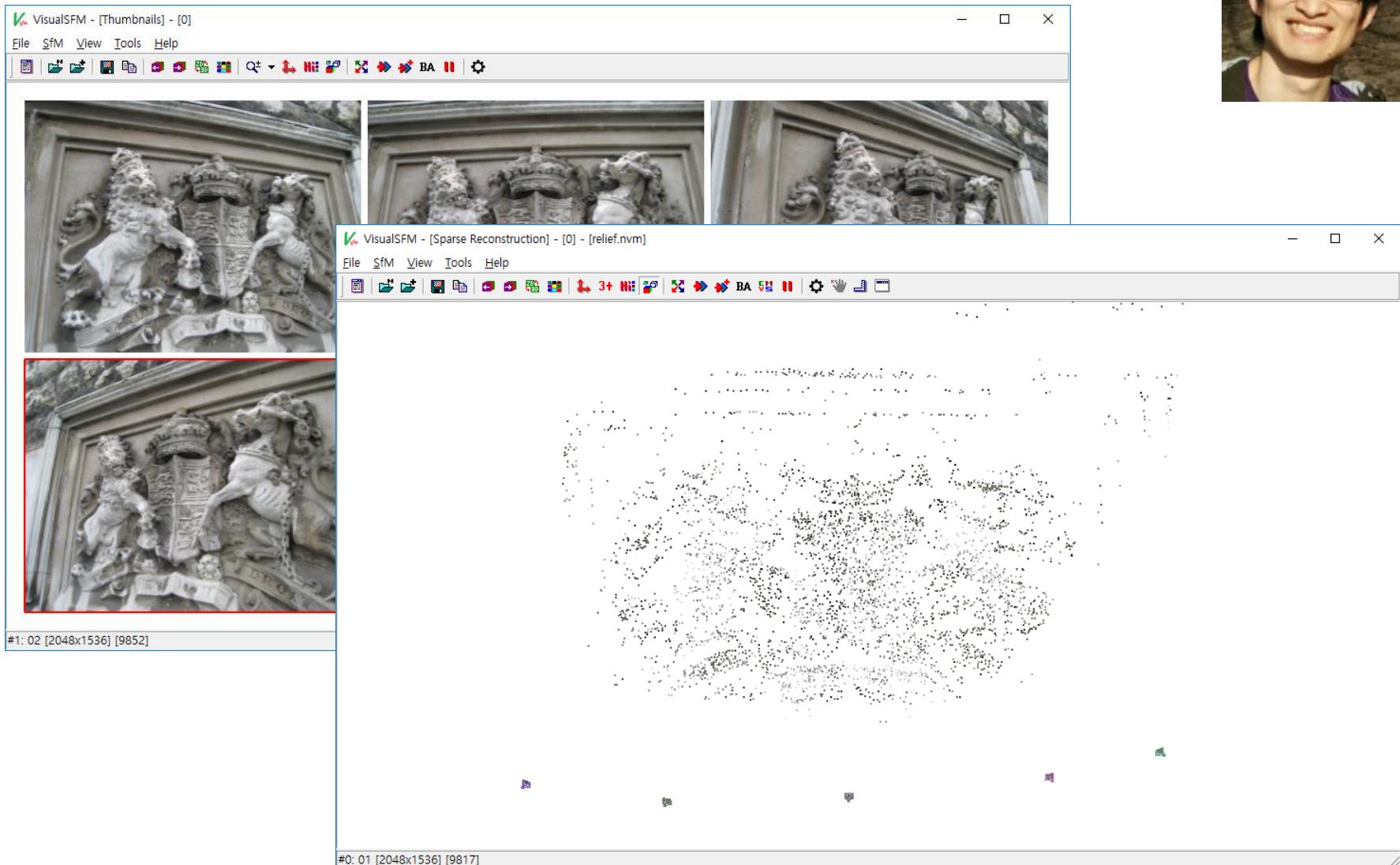
Result: `sfm_global(point).xyz` / `sfm_global(camera).xyz`



Result: `sfm_inc(point).xyz` / `sfm_inc(camera).xyz`

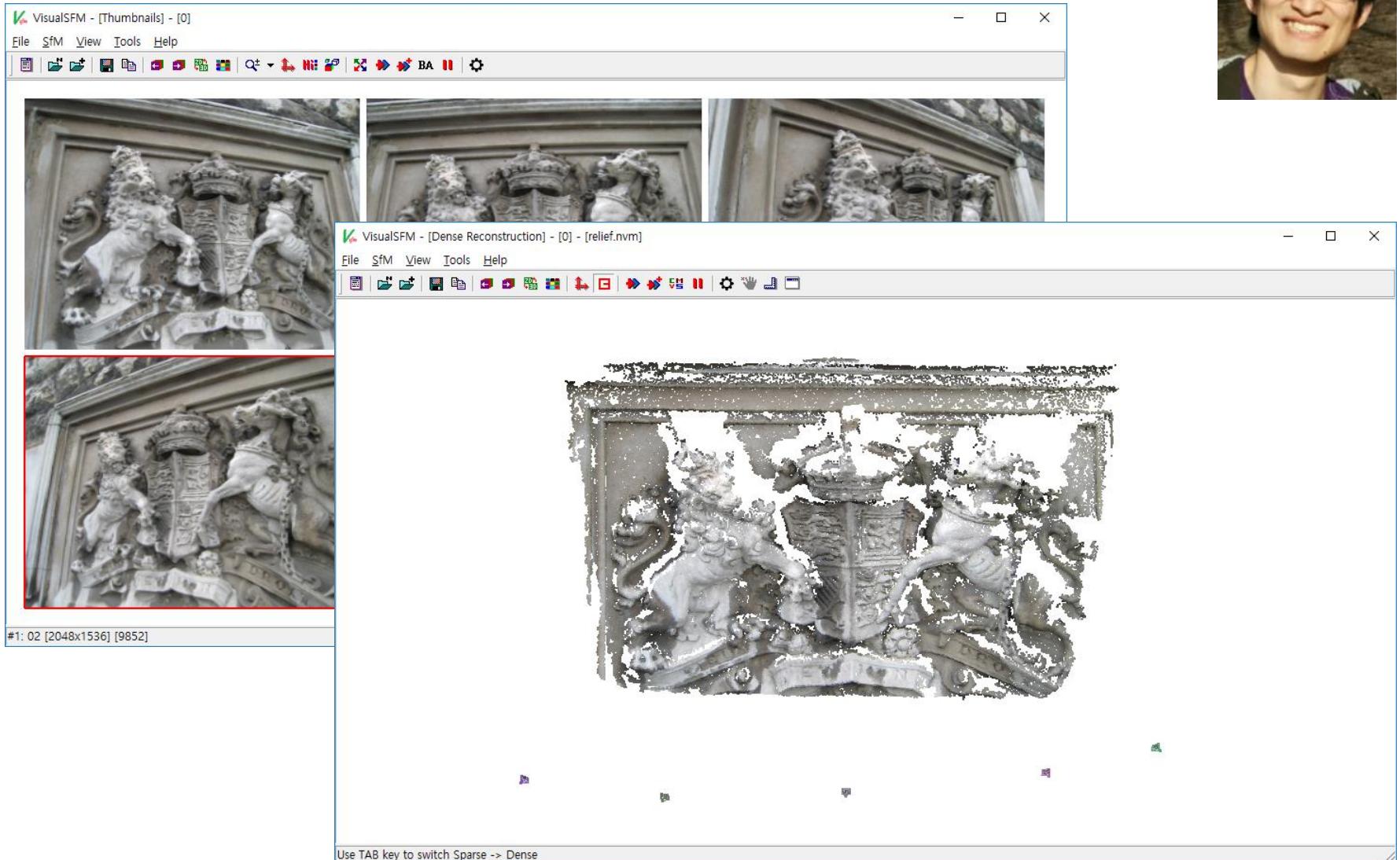


Incremental Structure-from-Motion using VisualSfM



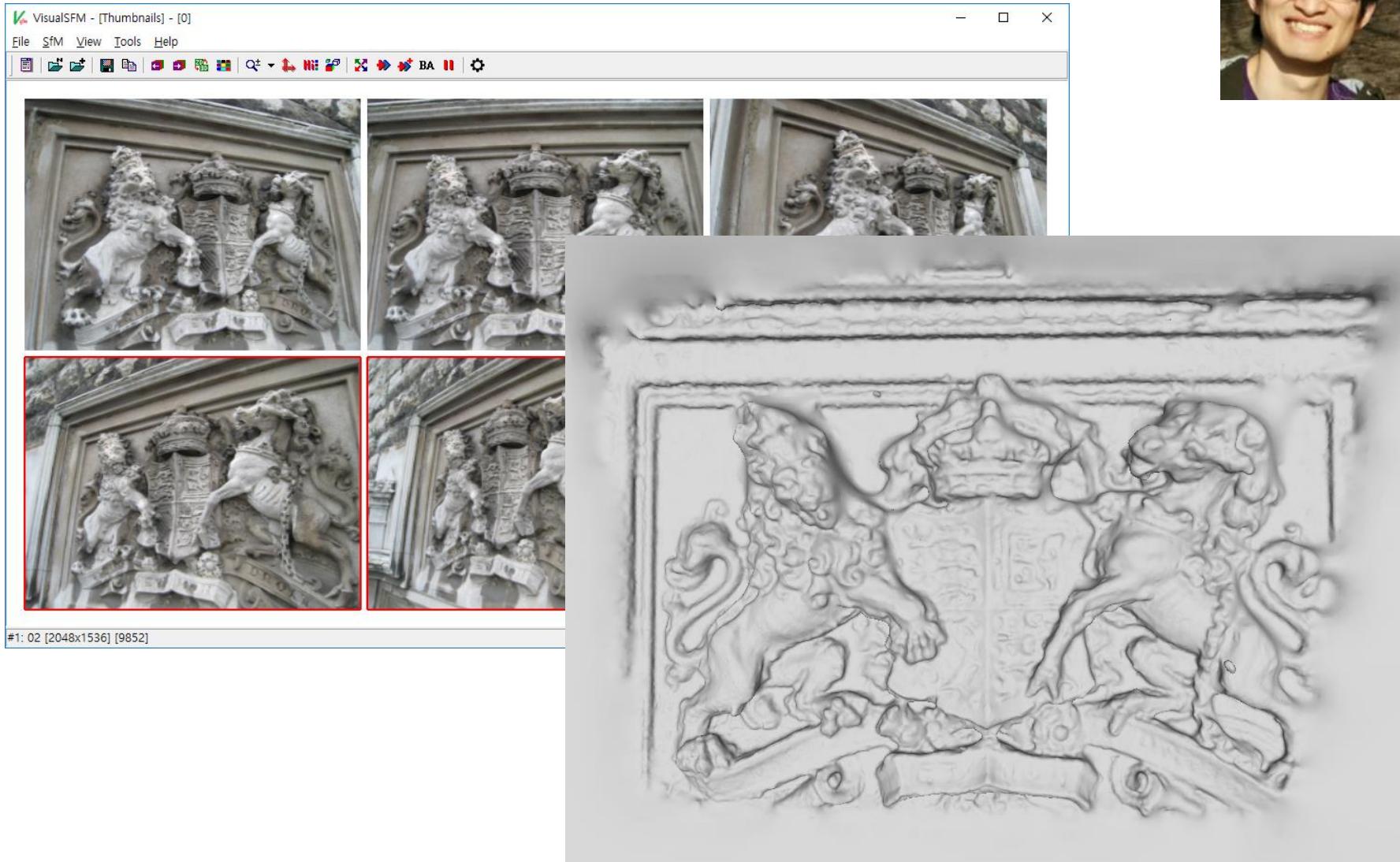
Sparse 3D Reconstruction by VisualSfM (SiftGPU and pba)

Incremental Structure-from-Motion using VisualSfM



Dense 3D Reconstruction by VisualSfM (CMVS and PMVS2)

Incremental Structure-from-Motion using VisualSfM



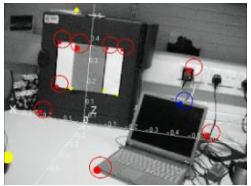
Surface Reconstruction by VisualSfM (Poisson Surface Reconstruction) [\[URL\]](#)

Visual SLAM: History

▪ 15 Years of Visual SLAM

Monocular Camera

MonoSLAM (2003): features, filtering

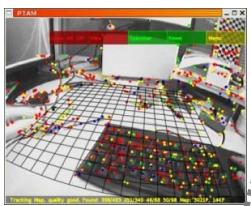


SLAM in Robotics

V.S.

SfM in Computer Vision

PTAM (2007): more features, optimization
(keyframes)



RGB-D Camera



DTAM (2011): dense volumetric



KinectFusion (2011): dense volumetric

ORB-SLAM (2014) v.s.

more features

LSD-SLAM (2014)

direct, semi-dense



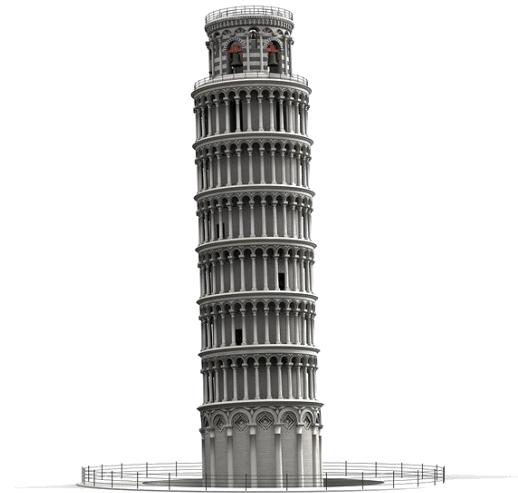
ElasticFusion (2015) / DynamicFusion (2015)



dense surfels

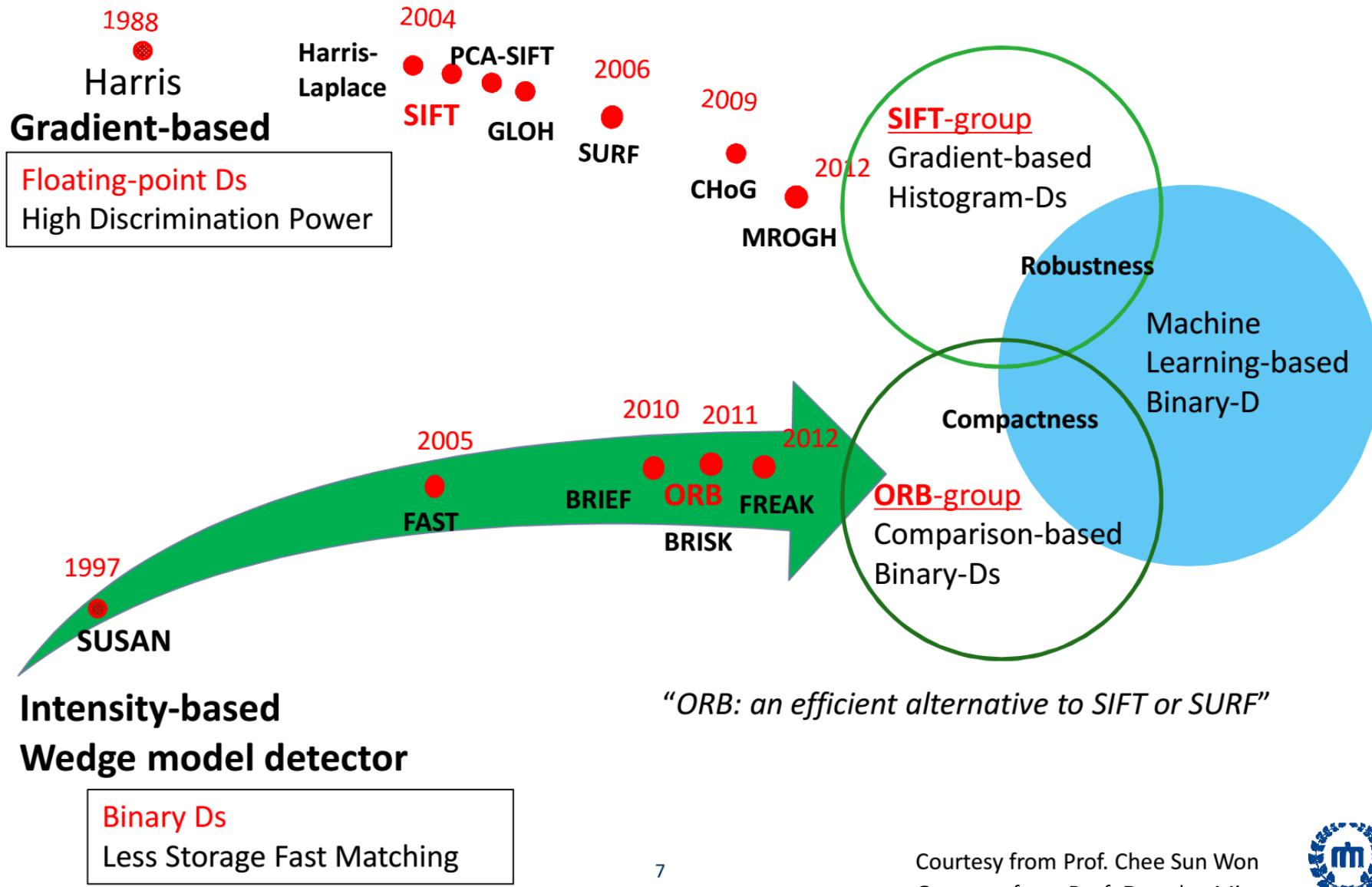
dynamic dense

Correspondence Problem

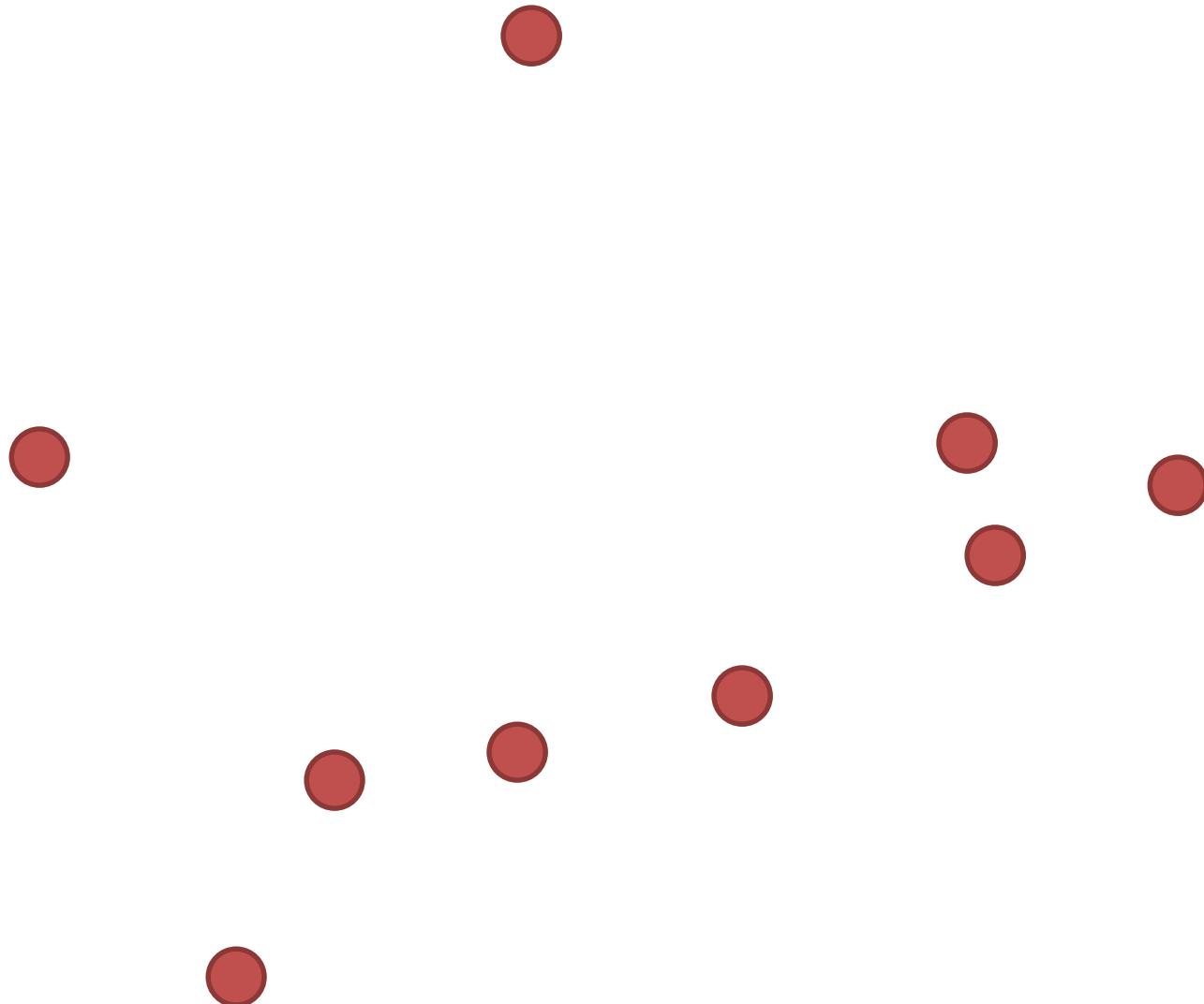


Images are from [pixabay](#).

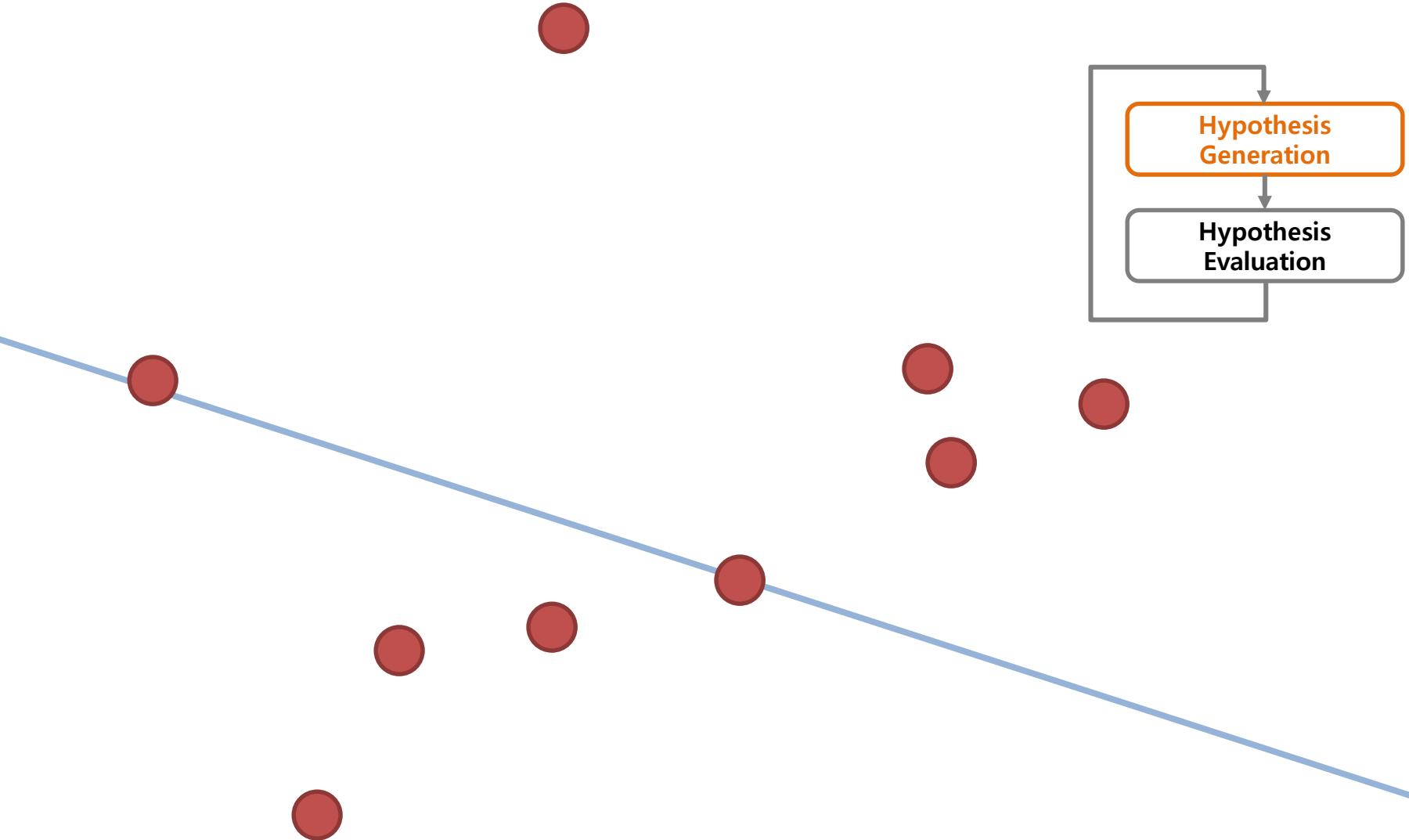
Feature Points and Descriptors



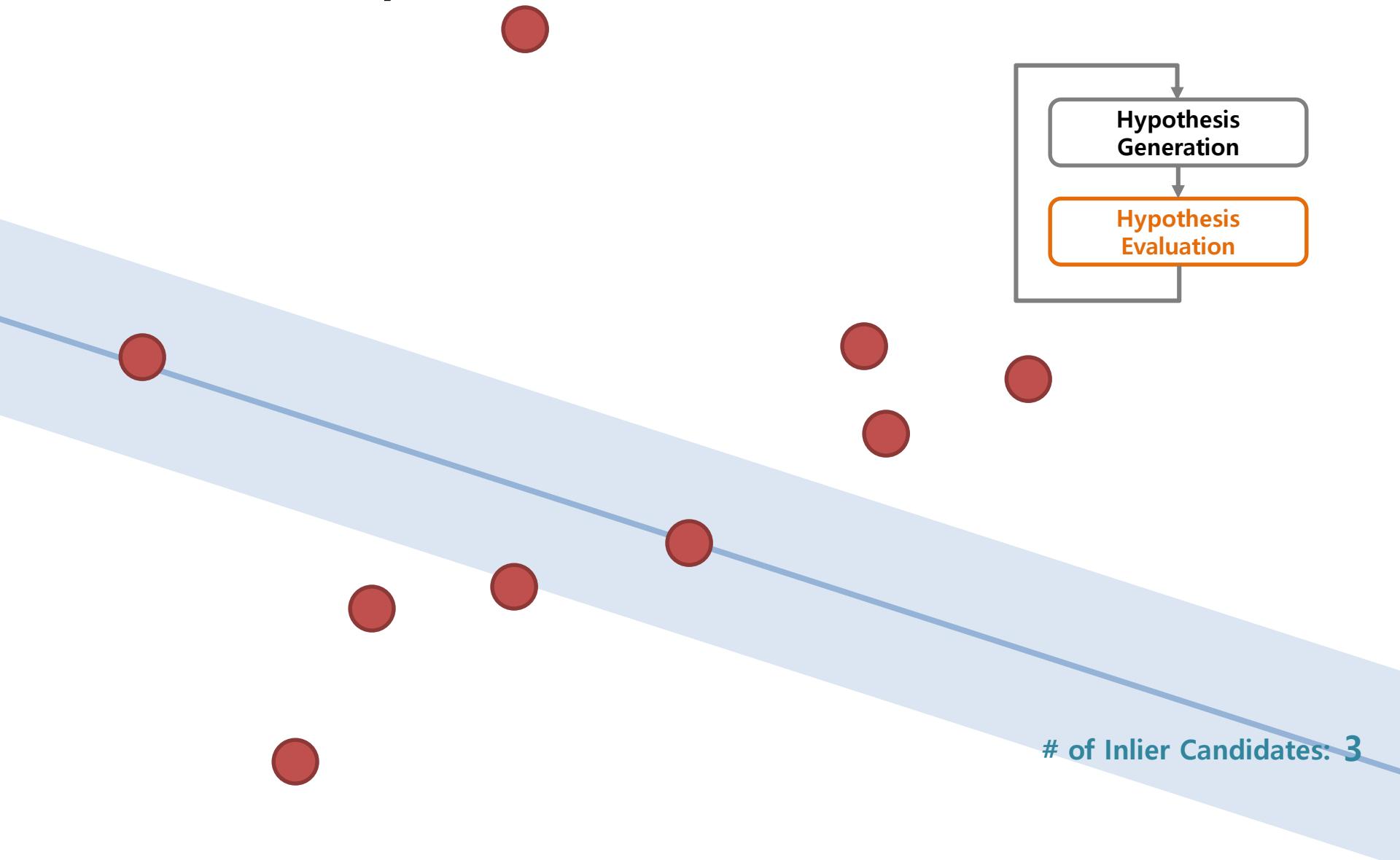
Random Sample Consensus (RANSAC)



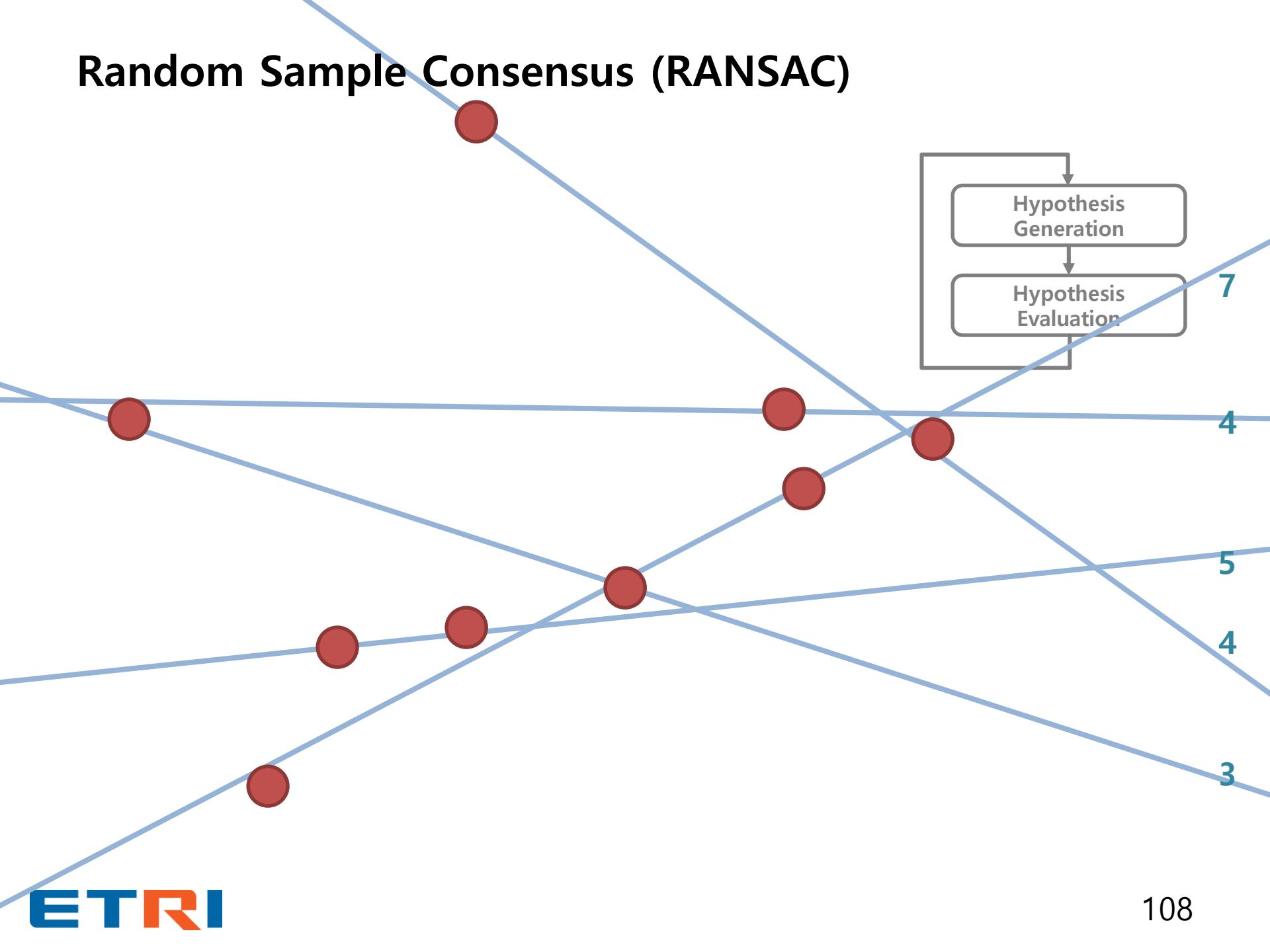
Random Sample Consensus (RANSAC)



Random Sample Consensus (RANSAC)



Random Sample Consensus (RANSAC)



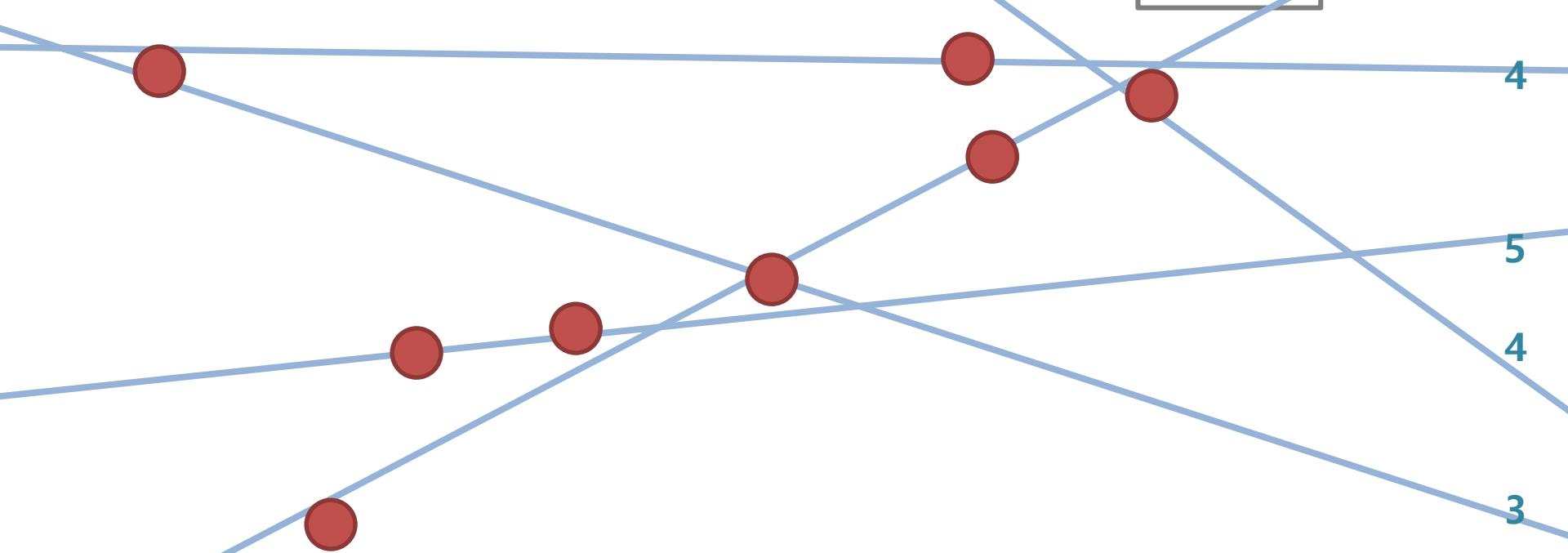
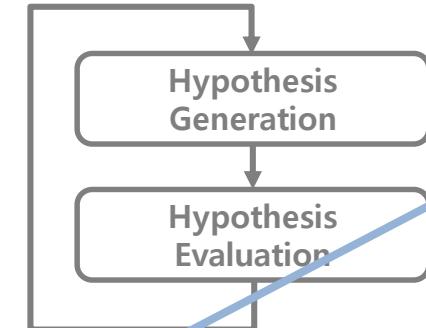
Random Sample Consensus (RANSAC)

Parameters:

- The inlier threshold
- The number of iterations

$$k \geq \frac{\log(1 - \eta_0)}{\log(1 - \epsilon^m)} \quad \eta_0: \text{The confidence level}$$

$\epsilon: \text{The inlier ratio}$



Example: Line Fitting with RANSAC



```
1. #include "opencv_all.hpp"
2. // Convert a line format, [n_x, n_y, x, y] to [a, b, c]
3. #define CONVERT_LINE(line) (cv::Vec3d(line(0), -line(1), -line(0) * line(2) + line(1) * line(3)))
4. int main(void)
5. {
6.     int ransac_trial = 50;
7.     double ransac_thresh = 3.0;
8.     int ransac_n_sample = 2;
9.     int sim_n_data = 1000;
10.    double sim_inlier_ratio = 0.5, sim_inlier_noise = 1.0;
11.    cv::Vec3d truth(1.0 / sqrt(2.0), 1.0 / sqrt(2.0), -240.0); // The line model: a*x + b*y + c = 0
12.    // Generate data
13.    std::vector<cv::Point2d> data;
14.    cv::RNG rng;
15.    for (int i = 0; i < sim_n_data; i++)
16.    {
17.        if (rng.uniform(0.0, 1.0) < sim_inlier_ratio)
18.        {
19.            double x = rng.uniform(0.0, 480.0);
20.            double y = (truth(0) * x + truth(2)) / -truth(1);
21.            x += rng.gaussian(sim_inlier_noise);
22.            y += rng.gaussian(sim_inlier_noise);
23.            data.push_back(cv::Point2d(x, y)); // Inlier
24.        }
25.        else data.push_back(cv::Point2d(rng.uniform(0.0, 640.0), rng.uniform(0.0, 480.0))); // Outlier
26.    }
27.    // Estimate a line using RANSAC ...
28.    // Estimate a line using least-squares method (for reference) ...
29.    // Display estimates
30.    printf("* The Truth: %.3f, %.3f, %.3f\n", truth(0), truth(1), truth(2));
31.    printf("* Estimate (RANSAC): %.3f, %.3f, %.3f (Score: %d)\n", best_line(0), best_line(1), ..., best_score);
32.    printf("* Estimate (LSM): %.3f, %.3f, %.3f\n", lsm_line(0), lsm_line(1), lsm_line(2));
33.    return 0;
34.}
```



Example: Line Fitting with RANSAC

```
27. // Estimate a line using RANSAC
28. int best_score = -1;
29. cv::Vec3d best_line;
30. for (int i = 0; i < ransac_trial; i++)
31. {
32.     // Step 1: Hypothesis generation
33.     std::vector<cv::Point2d> sample;
34.     for (int j = 1; j < ransac_n_sample; j++)
35.     {
36.         int index = rng.uniform(0, data.size());
37.         sample.push_back(data[index]);
38.     }
39.     cv::Vec4d vvxy;
40.     cv::fitLine(sample, vvxy, CV_DIST_L2, 0, 0.01, 0.01);
41.     cv::Vec3d line = CONVERT_LINE(vvxy);

42.     // Step 2: Hypothesis evaluation
43.     int score = 0;
44.     for (size_t j = 0; j < data.size(); j++)
45.     {
46.         double error = fabs(line(0) * data[j].x + line(1) * data[j].y + line(2));
47.         if (error < ransac_thresh) score++;
48.     }

49.     if (score > best_score)
50.     {
51.         best_score = score;
52.         best_line = line;
53.     }
54. }

55. // Estimate a line using least-squares method (for reference)
56. cv::Vec4d vvxy;
57. cv::fitLine(data, vvxy, CV_DIST_L2, 0, 0.01, 0.01);
58. cv::Vec3d lsm_line = CONVERT_LINE(vvxy);
```

Line Fitting Result

```
* The Truth: 0.707, 0.707, -240.000
* Estimate (RANSAC): 0.712, 0.702, -242.170 (Score: 434)
* Estimate (LSM): 0.748, 0.664, -314.997
```

Summary: Overview

- **What is 3D Vision?**
- **Single-view Geometry**
 - **Camera Projection Model**
 - A Pinhole Camera Model (**Simple 2D-3D Geometry**)
 - Homogenous Coordinates
 - Geometric Distortion and Rectification
 - **General 2D-3D Geometry**
 - Camera Calibration
 - Absolute Camera Pose Estimation (PnP Problem)
- **Two-view Geometry**
 - **Planar 2D-2D Geometry (Projective Geometry)**
 - Planar Homography
 - **General 2D-2D Geometry (Epipolar Geometry)**
 - Fundamental/Essential Matrix
 - Relative Camera Pose Estimation
 - Relative Point Localization (Triangulation)
- **Multi-view Geometry**
 - Bundle Adjustment (Non-linear Optimization)
 - Applications: Structure-from-motion, Visual SLAM, and Visual Odometry
- **Correspondence Problem**
 - Random Sample Consensus (RANSAC)

$$\therefore \mathbf{x} = \mathbf{P}\mathbf{X} \quad (\mathbf{P} = \mathbf{K}[\mathbf{R} \mid \mathbf{t}])$$

$$\mathbf{x} = \mathbf{K}\hat{\mathbf{x}}$$

$$\therefore \mathbf{x}' = \mathbf{H}\mathbf{x}$$

$$\hat{\mathbf{x}}' = \hat{\mathbf{H}}\hat{\mathbf{x}} \quad (\hat{\mathbf{H}} = \mathbf{R} + \frac{1}{d}\mathbf{t}\mathbf{n}^\top)$$

$$\therefore \mathbf{x}'^\top \mathbf{F} \mathbf{x} = 0$$

$$\hat{\mathbf{x}}'^\top \mathbf{E} \hat{\mathbf{x}} = 0 \quad (\mathbf{E} = [\mathbf{t}]_\times \mathbf{R})$$

$$\sum_i^n \sum_j^m \left\| \mathbf{x}_i^j - \mathbf{P}_j \mathbf{X}_i \right\|_{\Sigma}^2$$

Further Information

- **Beyond Point Features**
 - Other features: [Line-based SfM](#) (3DPVT 2006), [Pointless SfM](#) (ICCV 2015)
 - Direct methods (w/o feature extraction)
 - Deep learning: [PoseNet](#), [CNN-SLAM](#) (CVPR 2017)
 - Photometric error minimization: [DTAM](#), [LSD-SLAM](#), [DSO](#), [SVO](#)
- ~~Real-time / Large-scale SfM~~
- **(Spatially / Temporally) Non-static SfM**
 - Non-rigid (or moving) objects: [Non-rigid SfM](#)
- **Sensor Fusion and New Sensors**
 - Depth (RGB-D, stereo, LiDAR): [ElasticFusion](#), [DVO-SLAM](#), [RGB-D SLAM](#)
 - Omni-directional camera: [Multi-FoV datasets](#)
 - Light-field camera
 - Event camera: [ETAM](#) (ECCV 2016)
 - IMU: [OKVIS](#)
 - GPS
- **Minimal Solvers / Self-calibration**
 - [OpenGV](#)

Appendix: How to Solve Problems

- Linear Equations

- Inhomogeneous cases: Multiplying a pseudo-inverse

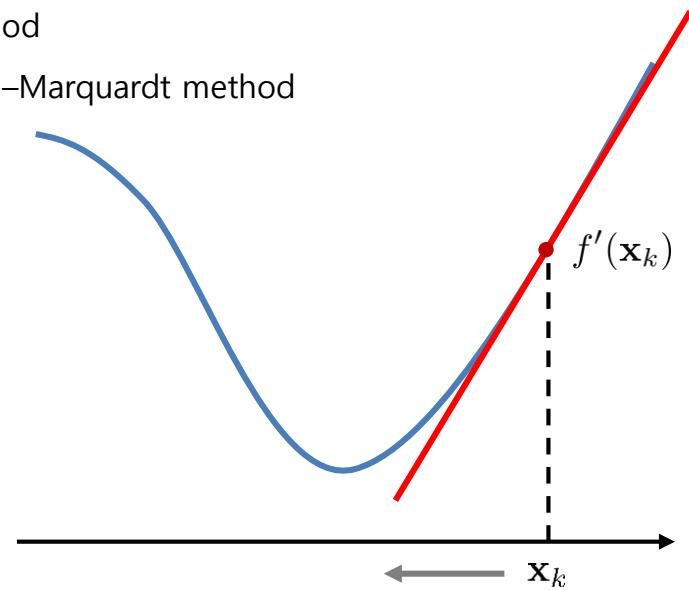
$$Ax = b \longrightarrow x = A^\dagger b$$

- Homogenous cases: Finding a null vector (or least-singular-valued vector) using SVD

$$Ax = 0 \longrightarrow x = V [0, 0, \dots, 1]^\top \text{ where } A = UDV^\top$$

- Non-linear Equations

- Non-linear optimization $\hat{x} = \underset{\mathbf{x}}{\operatorname{argmin}} f(\mathbf{x})$
 - General cases: Gradient-descent method, Newton method
 - Least-squares cases: Gauss-Newton method, Levenberg–Marquardt method



Appendix: Running Examples in Windows

1. Install Microsoft Visual Studio (<https://www.visualstudio.com/>)
 - *Visual Studio Community* is free for students, open-source, and individual developers.
2. Download examples and dependencies (https://github.com/sunglok/3dv_tutorial)
 - Click *Clone or download > Download ZIP* shown as a green button.
 - Click *OpenCV binary* and *CLAPACK binary* on README.md.
3. Unzip examples and dependencies at *your_folder*.
 - OpenCV and CLAPACK will be located at *your_folder\EXTERNAL* folder.
4. Run Microsoft Visual Studio and open the solution.
 - Or simply double-click *your_folder\msvs\3dv_tutorial.sln* file.
5. Build examples.
 - Click *MENU > BUILD > Build Solution*.
 - Executable will be located at *your_folder\bin* folder.
6. Run examples.