3D from Photographs: Structure From Motion

Francesco Banterle, Ph.D.
francesco.banterle@isti.cnr.it

Note: in these slides the optical center is placed back to simplify drawing and understanding.

3D from Photographs

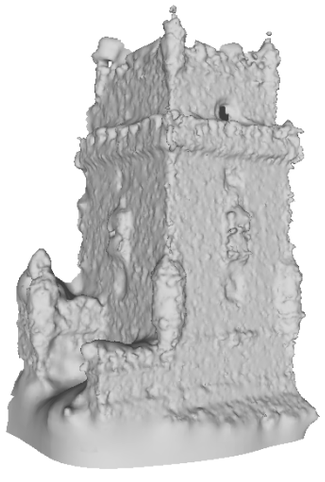
Photographs

Automatic Matching of Images

Camera Calibration

Dense Matching

Surface Reconstruction



3D model

3D from Photographs

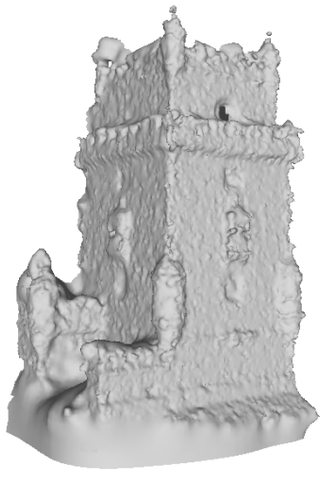
Photographs

Automatic Matching of Images

Camera Calibration

Dense Matching

Surface Reconstruction



3D model

Camera Pose Calibration

Camera Pose Calibration

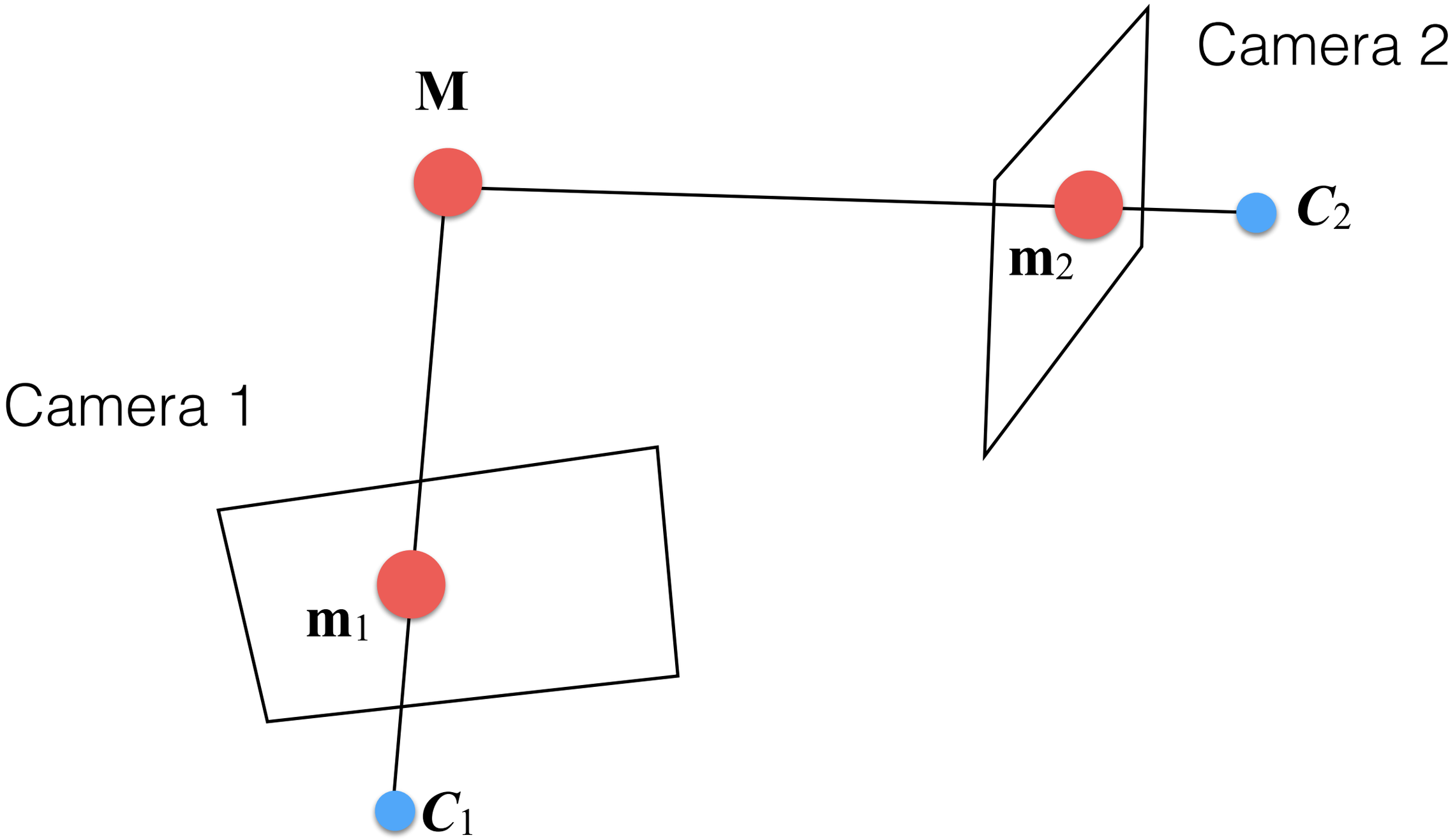
- We have seen methods for estimating the intrinsic matrix K, and the extrinsic matrix $G = [R \mid \boldsymbol{t}]$ using a calibration pattern:
  - DLT
  - Zhang's algorithm
- How do we get the camera's pose, G, without patterns?

Camera Pose Calibration

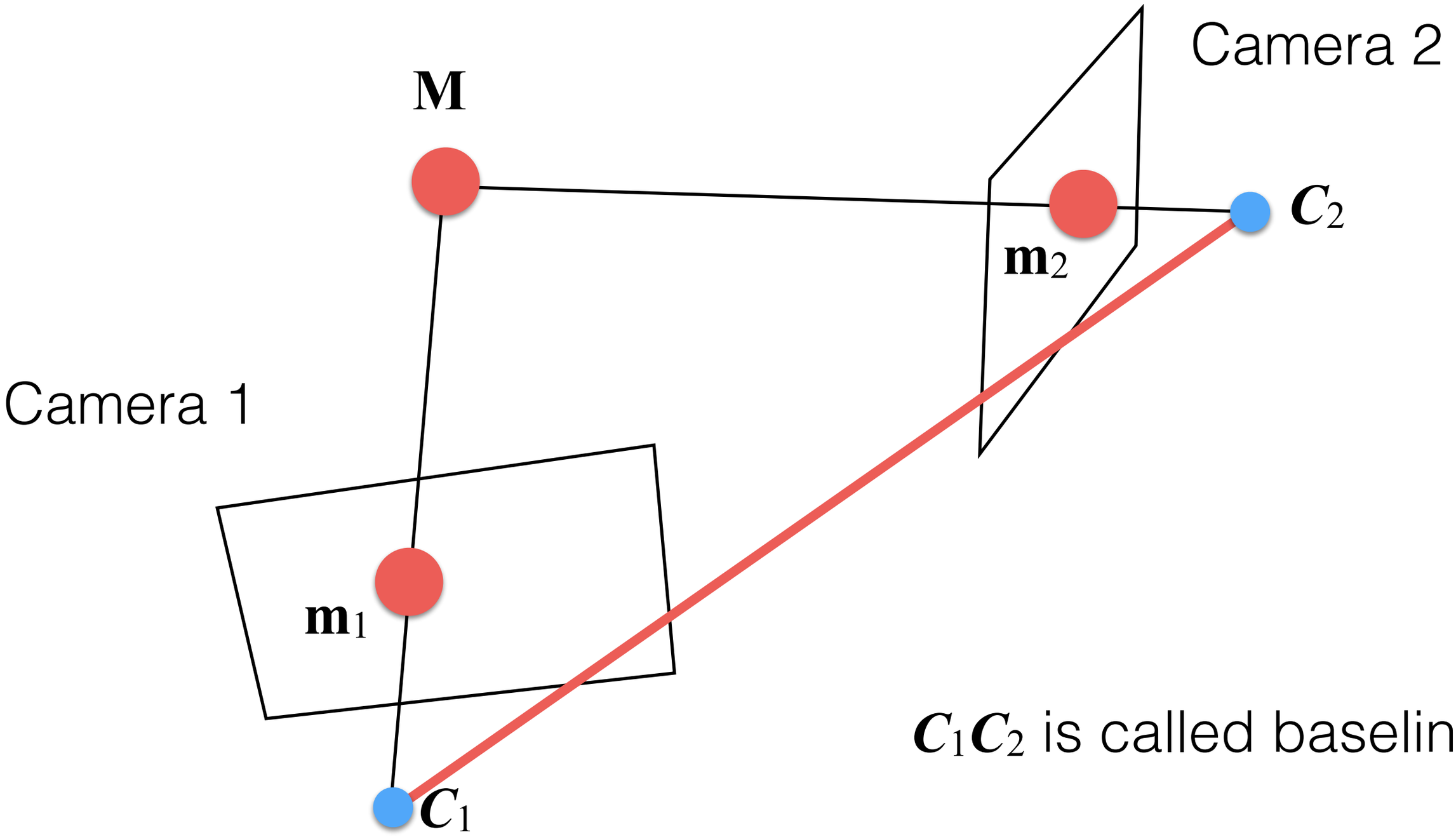
- Let's assume that:
  - We have K for each photograph.
  - We have matches between images.

A Two-Camera Example

A Two-Camera Example



A Two-Camera Example: Epipolar Geometry



C_1C_2 is called baseline

A Two-Camera Example

- Let's assume that we have K_1 and K_2.
- Let's assume that G_1 is set in the origin and aligned with the reference frame. This means:

$$G_1 = [I|\mathbf{0}] \rightarrow P_1 = K_1 \cdot G_1$$
$$P_2 = K_2 \cdot [R|\mathbf{t}]$$

Note that we only need to estimate R and $\boldsymbol{t}$!

A Two-Camera Example

- To simplify, let's remove K matrices:

$$P_1' = K_1^{-1} \cdot P_1 = [I|\mathbf{0}]$$
$$P_2' = K_2^{-1} \cdot P_2 = [R|\mathbf{t}]$$

- To points as well:

$$\hat{\mathbf{m}}_1 = K_1^{-1} \cdot \mathbf{m}_1$$
$$\hat{\mathbf{m}}_2 = K_2^{-1} \cdot \mathbf{m}_2$$

A Two-Camera Example

- To simplify, let's remove K matrices:

$$P_1' = K_1^{-1} \cdot P_1 = [I|\mathbf{0}]$$
$$P_2' = K_2^{-1} \cdot P_2 = [R|\mathbf{t}]$$

- To points as well:

$$\hat{\mathbf{m}}_1 = K_1^{-1} \cdot \mathbf{m}_1$$
$$\hat{\mathbf{m}}_2 = K_2^{-1} \cdot \mathbf{m}_2$$

Normalized coordinates

A Two-Camera Example

- Given the Longuet-Higgins equation, we know that:

$$\hat{\mathbf{m}}_2^{\mathsf{T}} \cdot E \cdot \hat{\mathbf{m}}_1 = 0$$

- E (a 3×3 matrix) is called the ***essential matrix*** and it is defined as:

$$E = [\mathbf{t}]_\times \cdot R \qquad\qquad [\mathbf{t}]_\times = \begin{bmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{bmatrix}$$

The Fundamental Matrix

- If we do not have K_1 and K_2 and apply the Longuet-Higgins equation, we obtain:

$$\mathbf{m}_2^\top \cdot F \cdot \mathbf{m}_1 = 0$$

- where F (a 3×3 matrix) is called the ***fundamental matrix***. This matrix is linked to E as:

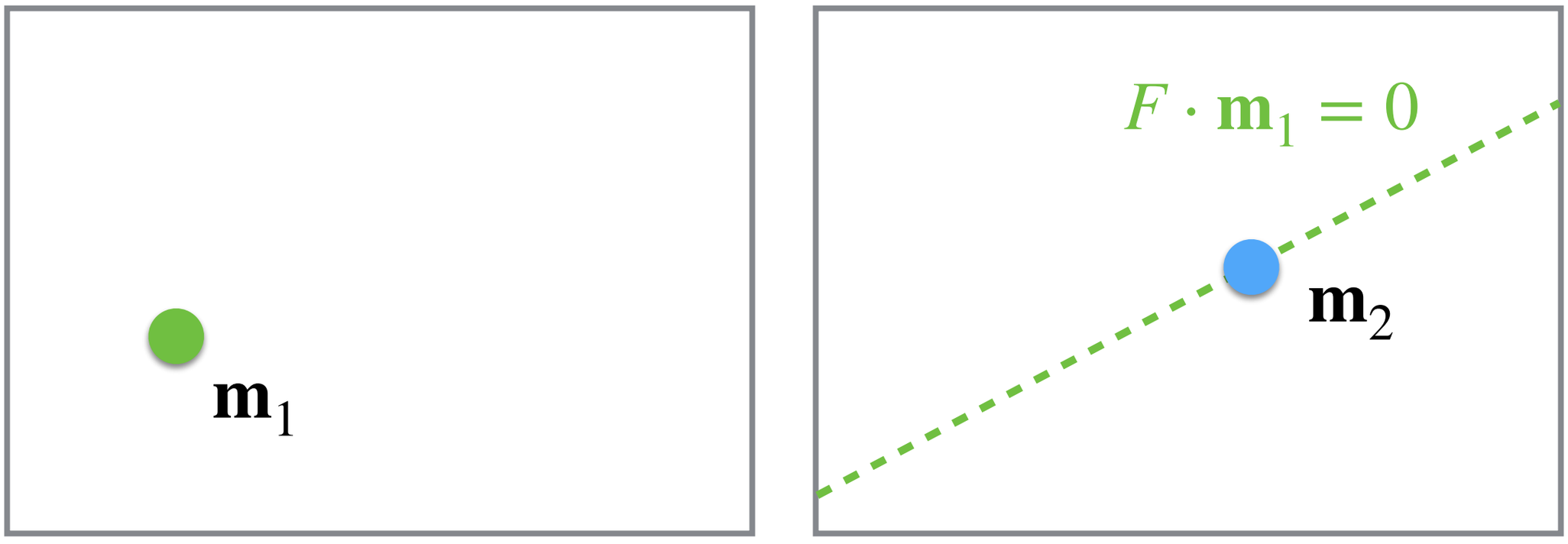
$$F = K_2^{-\top} \cdot E \cdot K_1^{-1} \leftrightarrow E = K_2^\top \cdot F \cdot K_1$$

The Fundamental Matrix

- F connects the two cameras "*geometry*" together.
- If we have a point $\mathbf{m}_1$ of I_1 and we multiply it by F, we obtain a straight line equation where its match, $\mathbf{m}_2$, lies in I_2:

$$\mathbf{l} = F \cdot \mathbf{m}_1 \leftrightarrow \left(l1x + l_2y + l_3\right) = 0$$

The Fundamental Matrix



I_1 I_2

The Fundamental Matrix

- Note that:

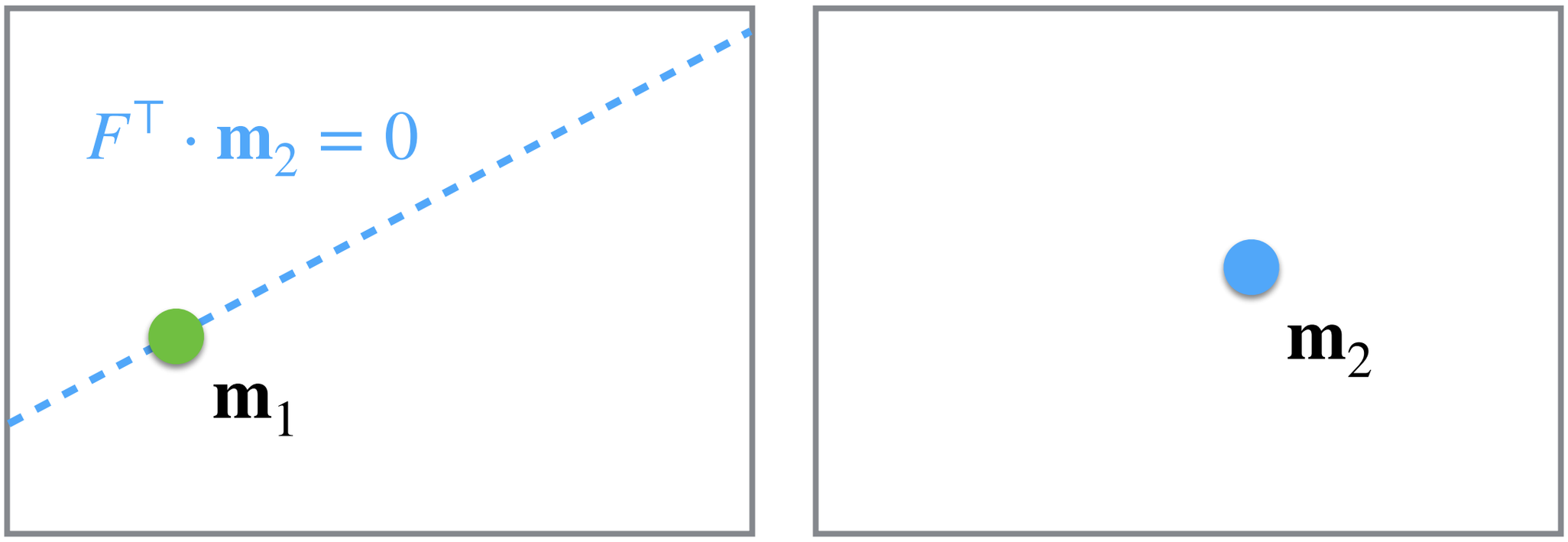
$$\mathbf{m}_2^\top \cdot F \cdot \mathbf{m}_1 = 0$$

$$\mathbf{m}_1^\top \cdot F^\top \cdot \mathbf{m}_2 = 0$$

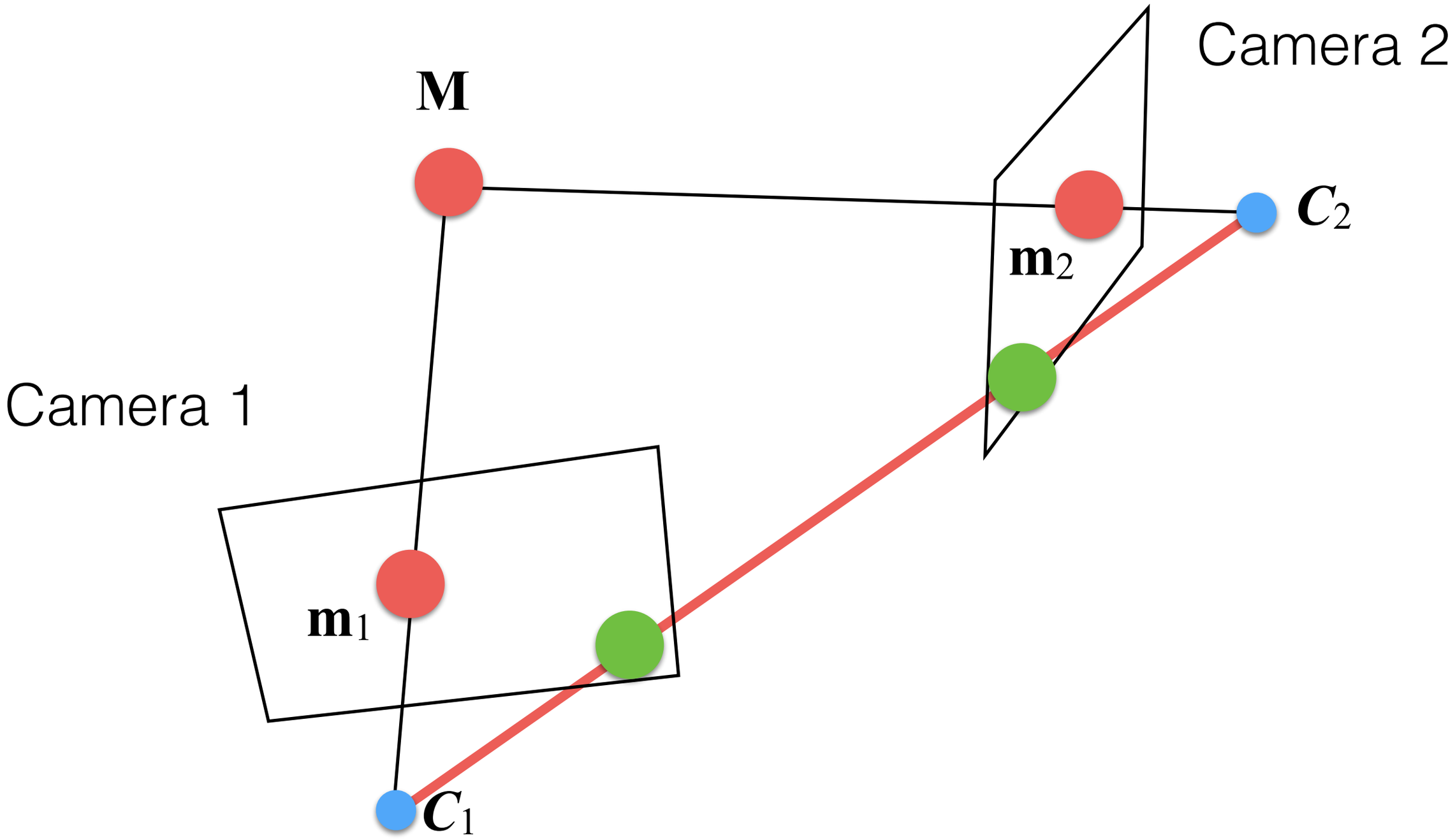
- Therefore, If we have a point $\boldsymbol{m}_2$ of I_2 and we multiply it by $F^\top$, we obtain a line equation in where its match, $\boldsymbol{m}_1$, lies in I_1:

$$\mathbf{l} = F^\top \cdot \mathbf{m}_2 \leftrightarrow (l_1 x + l_2 y + l_3) = 0$$

The Fundamental Matrix

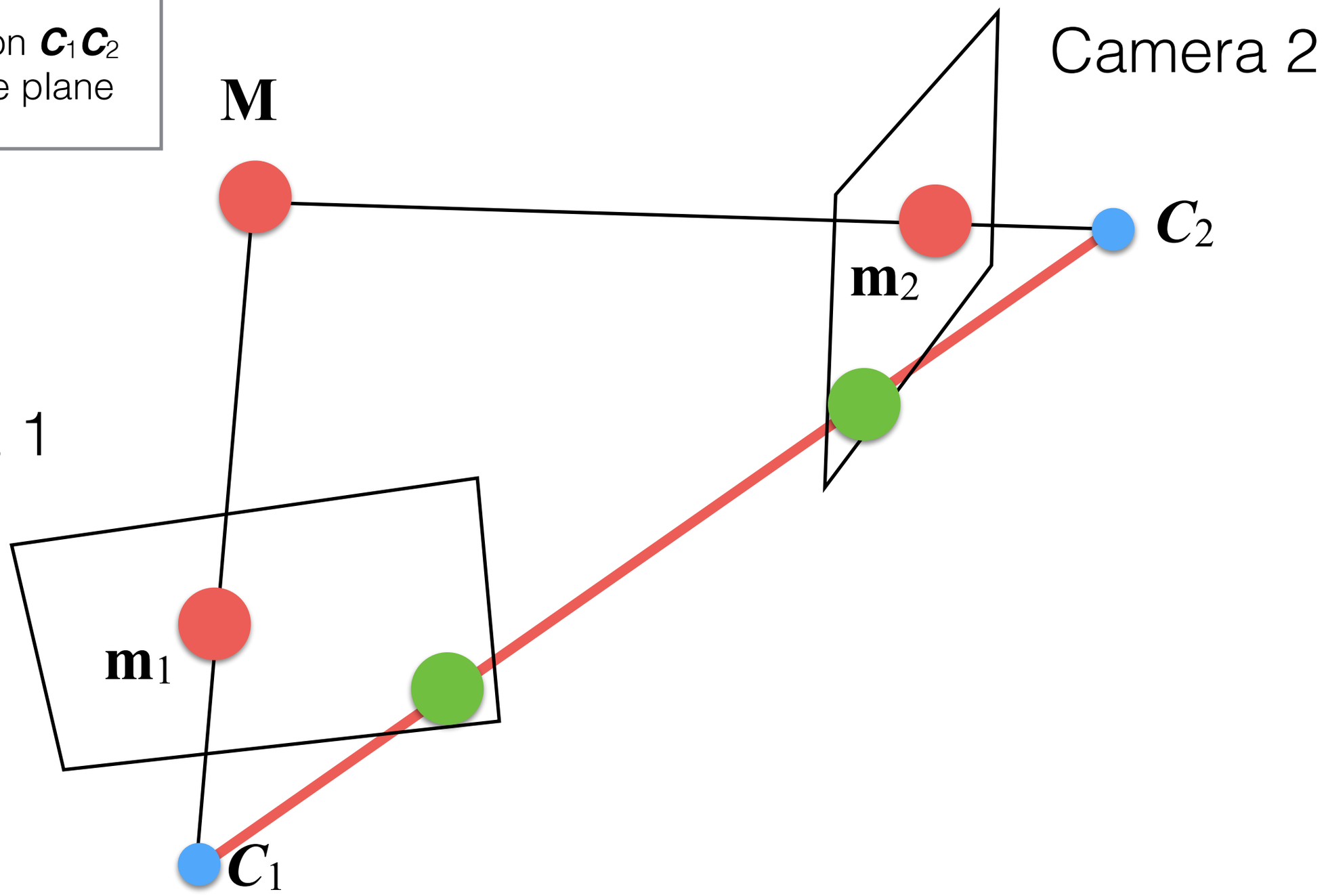


The Fundamental Matrix: A 3D Example

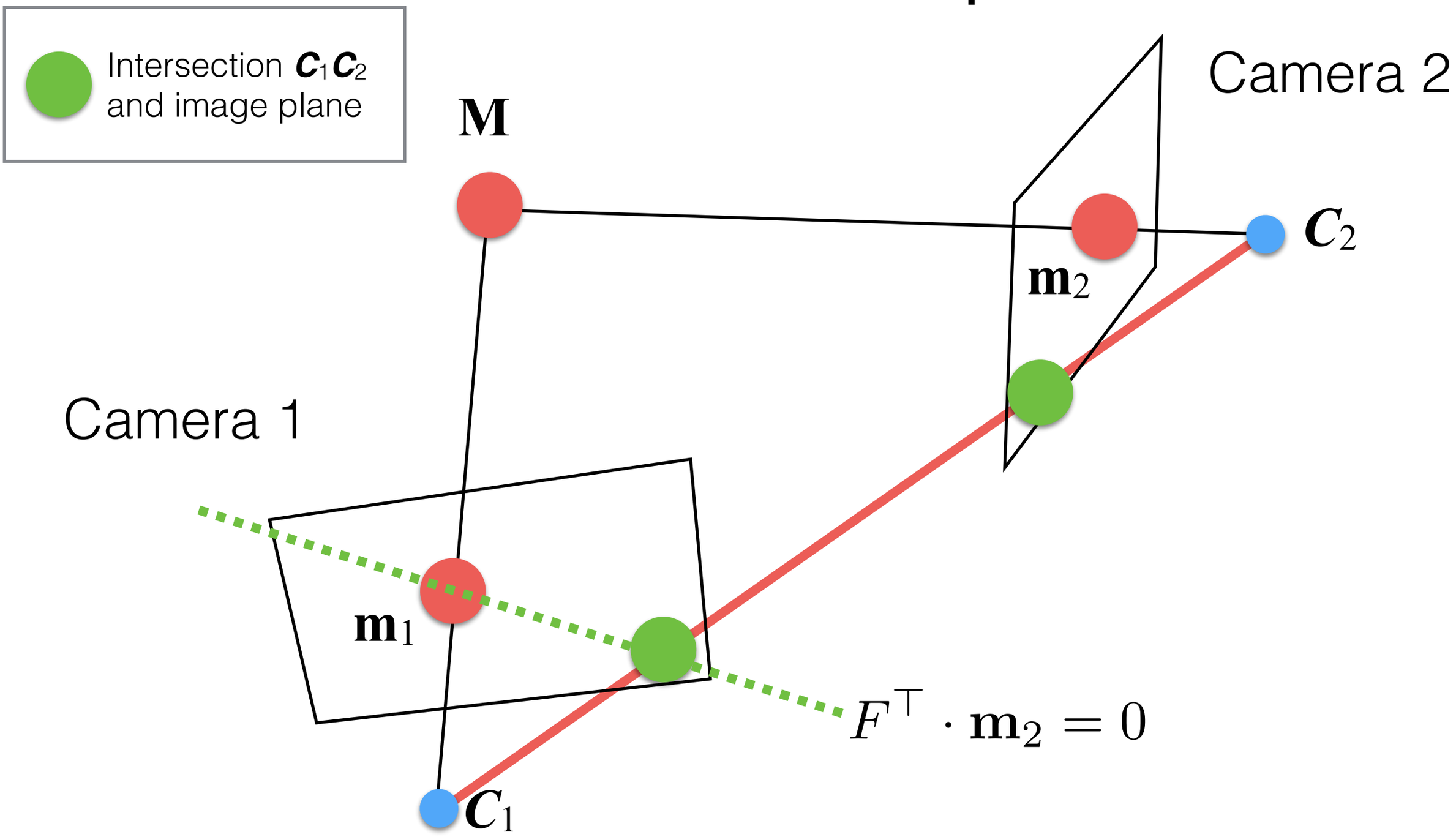


The Fundamental Matrix: A 3D Example

Intersection $\boldsymbol{c}_1\boldsymbol{c}_2$ and image plane



The Fundamental Matrix: A 3D Example



The Fundamental Matrix: 8-points algorithm

- From $\mathbf{m}_2^\top \cdot F \cdot \mathbf{m}_1 = 0$, we can define a linear system $A \cdot \mathbf{b} = \mathbf{0}$ as:

$$A = \begin{bmatrix} (\mathbf{m}_1^1)^\top \otimes (\mathbf{m}_2^1)^\top \\ \cdots \\ (\mathbf{m}_1^n)^\top \otimes (\mathbf{m}_2^n)^\top \end{bmatrix} \qquad \mathbf{b} = \mathbf{vec}(F)$$

- We can solve the system using the SVD. How many do we need?
  - **8 is the minimum**, as usual, the more the better!
- This method is called *8-points algorithm*, and it can be applied for computing E as well (we have to use normalized coordinates).

The Fundamental Matrix: Kronecker Product

- The Kronecker product is defined as

$$A \otimes B = \begin{bmatrix} a_{1,1} \cdot B & \dots & a_{1,n} \cdot B \\ a_{2,1} \cdot B & \dots & a_{2,n} \cdot B \\ \vdots & \dots & \vdots \\ a_{m,1} \cdot B & \dots & a_{m,n} \cdot B \end{bmatrix}$$

where A is $m \times n$ matrix, and B is a $r \times s$ matrix.

The Essential Matrix: Practice

- Typically, we compute E indirectly by computing F and then converting it into E:

$$E = K_2^\top \cdot F \cdot K_1$$

- Why?
  - The main reason is that we avoid to multiply K times all coordinates of points:
    - computational efficiency.
    - reducing accumulation of numerical errors.

The Essential Matrix: Practice

- If we solve the linear system using pixel coordinates, we may have ***numerical instabilities*** because pixel coordinates, (u, v), have large values:
  - $u \in [1, w]$
  - $v \in [1, h]$
  - where h and w are, respectively, the height and width of the image.

The Essential Matrix: Practice

- For reducing instabilities, we need to bound values in the range $[-\sqrt{2}, \sqrt{2}]$.

- Given the input n points $\mathbf{m}_i$, we compute:

$$\mathbf{m}_i = \begin{bmatrix} u_i & v_i & 1 \end{bmatrix}^\top \quad \hat{u} = \frac{1}{n}\sum_{i=1}^{n} u_i \quad \hat{v} = \frac{1}{n}\sum_{i=1}^{n} v_i$$

$$s = \frac{1}{n\sqrt{2}}\sum_{i=1}^{n}\sqrt{(u_i - \hat{u})^2 + (v_i - \hat{v})^2}$$

The Essential Matrix: Practice

- Finally, we shift and scale all n points using the following:

$$\tilde{u}_i = \frac{u_i - \hat{u}}{s} \qquad \tilde{v}_i = \frac{v_i - \hat{v}}{s}$$

- We can now solve the linear system without numerical instabilities (or reducing them).

- Note that this operation, shift and scale, needs to be done for estimating a homography as well.

Non-Linear Optimization

- As seen before, we need to refine E using a geometric error, note that we compute E indirectly so we minimize F:

$$\arg\min_{F} \sum_{i=1}^{n} d_\pi(F \cdot \mathbf{m}_1^i, \mathbf{m}_2^i)^2 + d_\pi(F^\top \cdot \mathbf{m}_2^i, \mathbf{m}_1^i)^2$$

- where d_π is the distance point-line ($F \cdot \mathbf{m}_1^i$ is a line), and n is the number of matched points.

- Again we can solve it with the Nelder-Mead method (**fminsearch** in MATLAB).

Non-Linear Optimization

- As seen before, we need to refine E using a geometric error, note that we compute E indirectly so we minimize F:

$$\arg\min_F \sum_{i=1}^{n} d_\pi(F \cdot \mathbf{m}_1^i, \mathbf{m}_2^i)^2 + d_\pi(F^\top \cdot \mathbf{m}_2^i, \mathbf{m}_1^i)^2$$

This is a line

- where d_π is the distance point-line ($F \cdot \mathbf{m}_1^i$ is a line), and n is the number of matched points.

- Again we can solve it with the Nelder-Mead method (**fminsearch** in MATLAB).

Non-Linear Optimization

- As seen before, we need to refine E using a geometric error, note that we compute E indirectly so we minimize F:

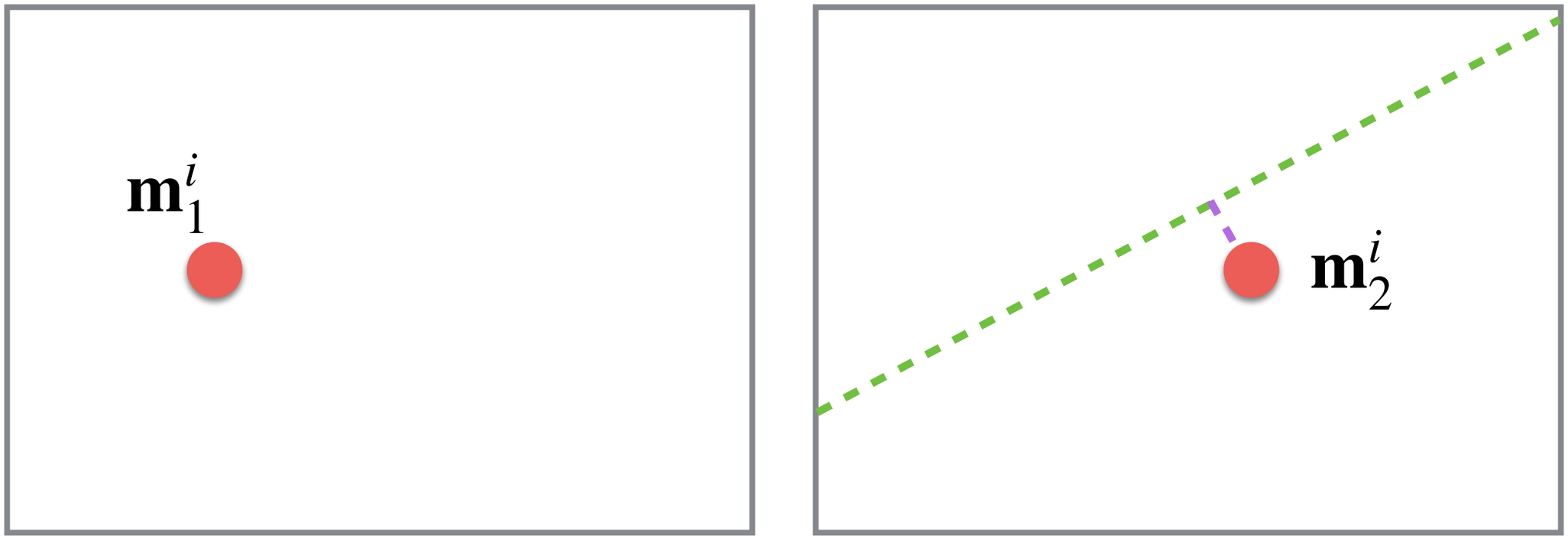
$$\arg\min_F \sum_{i=1}^{n} d_\pi(F \cdot \mathbf{m}_1^i, \mathbf{m}_2^i)^2 + d_\pi(F^\top \cdot \mathbf{m}_2^i, \mathbf{m}_1^i)^2$$

This is a line This is a line

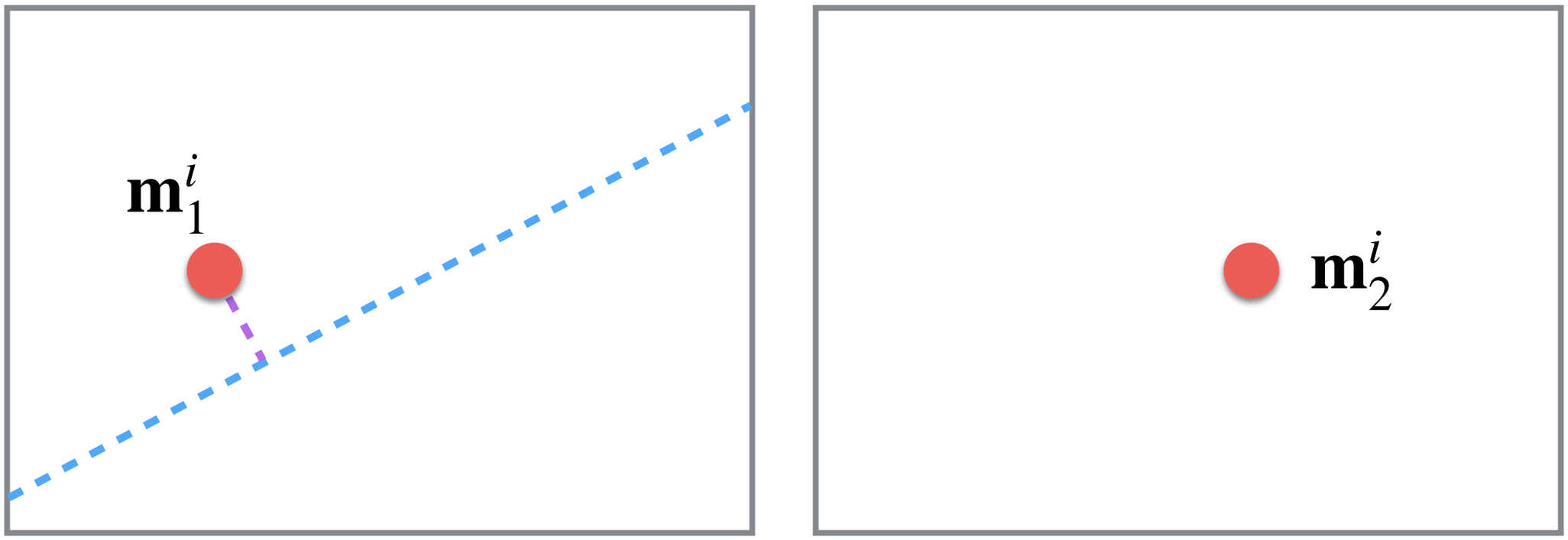
- where d_π is the distance point-line ($F \cdot \mathbf{m}_1^i$ is a line), and n is the number of matched points.

- Again we can solve it with the Nelder-Mead method (**fminsearch** in MATLAB).

Non-Linear Optimization: First Term of the Energy



Non-Linear Optimization: Second Term of the Energy



Now we have E, and so what?

E Factorization

- Once we have estimated E, we would like estimate R and $\mathbf{t}$ to get the pose of the camera:

$$E = [\mathbf{t}]_\times \cdot R$$

- As you may notice we have:
  - $[\mathbf{t}]_\times = S$ is an anti-symmetric matrix.
  - R is orthogonal matrix.

E Factorization

- Given a $m \times n$ matrix A, its SVD decomposition is:

$$\mathrm{SVD}(A) = U \cdot \Sigma \cdot V^*$$

where:

- U is an $m \times m$ orthogonal matrix.
- Σ is a diagonal $m \times n$ matrix.
- V^* is the conjugate transpose of an orthogonal matrix.

E Factorization

- **Theorem**: A 3×3 matrix is an essential matrix if and only if two singular values are equal and the third is zero.

- This means that:

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

- Note that $\mathrm{diag}(1,1,0) = W \cdot Z$, where W and Z are:

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad Z = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

E Factorization

- **Lemma**: Given R, a rotation matrix, and U and V, two orthogonal matrices, we have that:

$$R' = \det(U \cdot V^\top) \cdot U \cdot R \cdot R^\top$$

- **NOTE**: R' is still a rotation matrix!

E Factorization

- Given that:

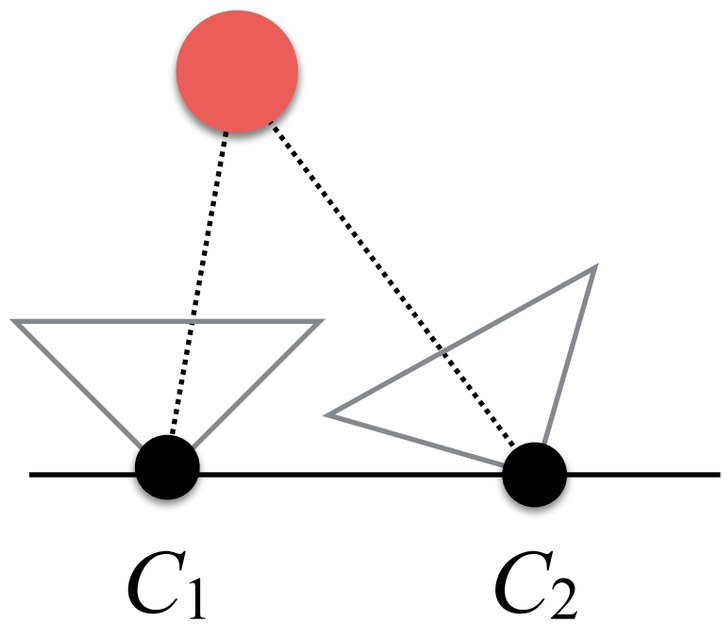
$$\mathrm{SVD}(E) = U \cdot \mathrm{diag}(1, 1, 0) \cdot V^\top$$

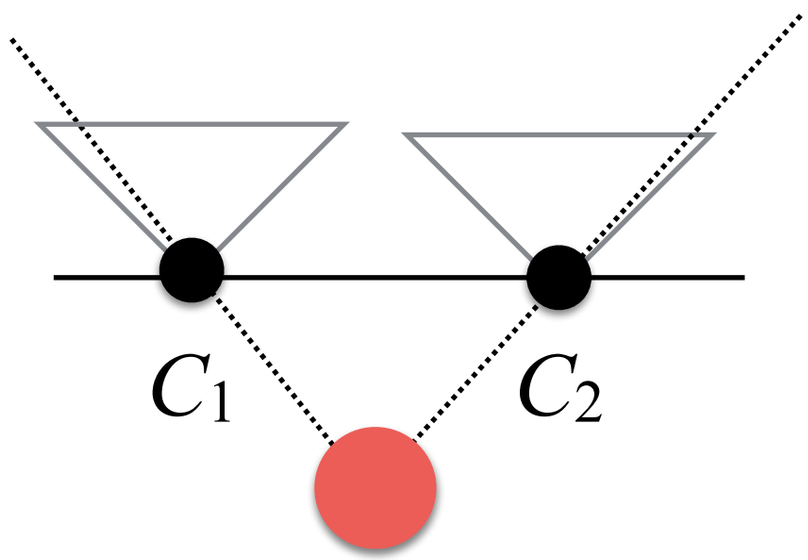
- We can have four possible factorizations of E such that $E = S \cdot R$:

$$S = U \cdot (\pm Z) \cdot U^\top$$

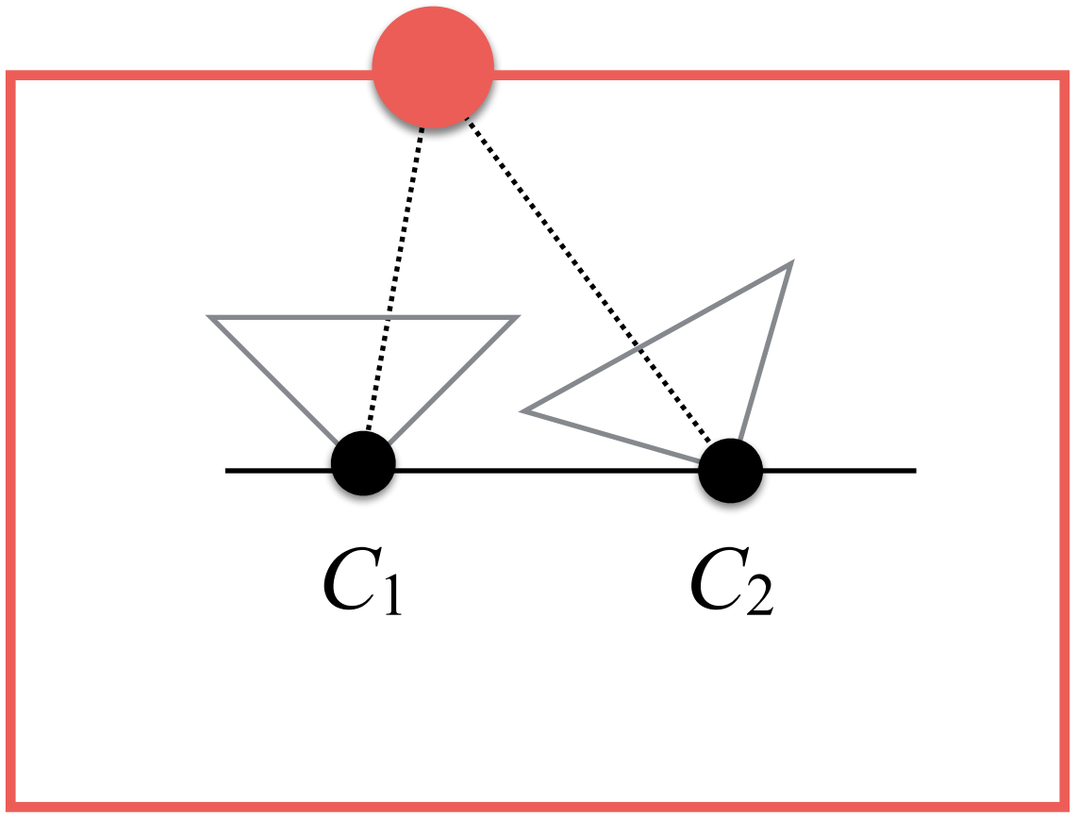
$$R = U \cdot W \cdot V^\top \text{ or } R = U \cdot W^\top \cdot V^\top$$

E Factorization: The Four Cases

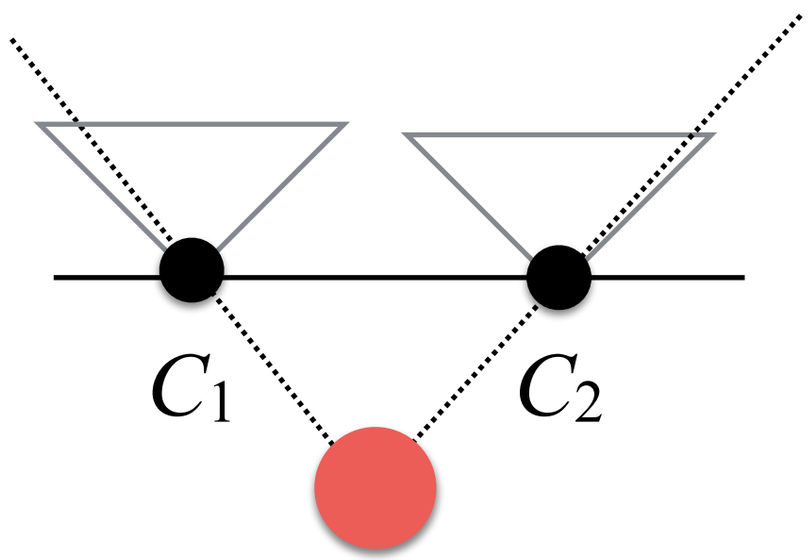




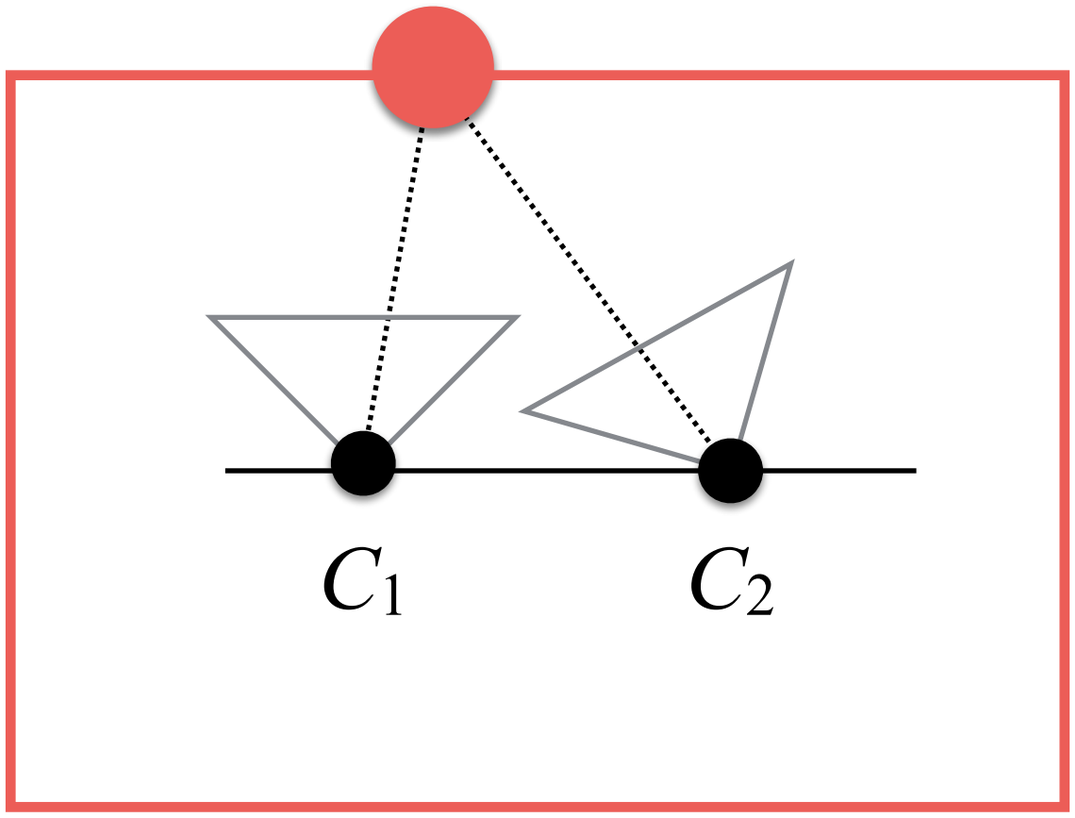
E Factorization: The Four Cases



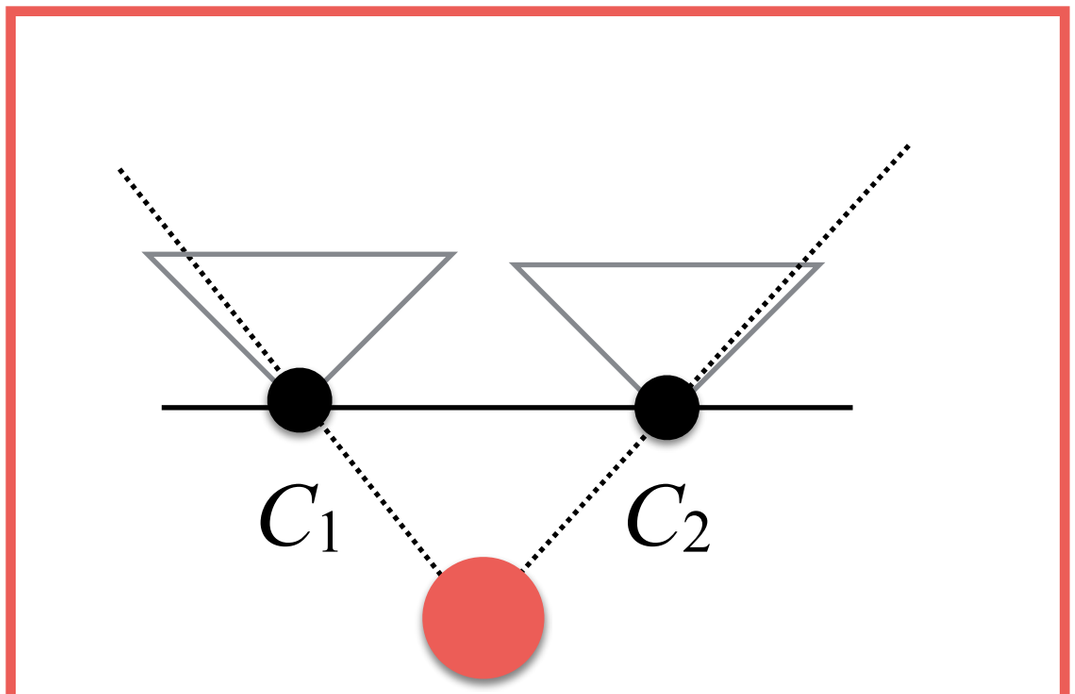
Both cameras see the point. It is in front of them



E Factorization: The Four Cases

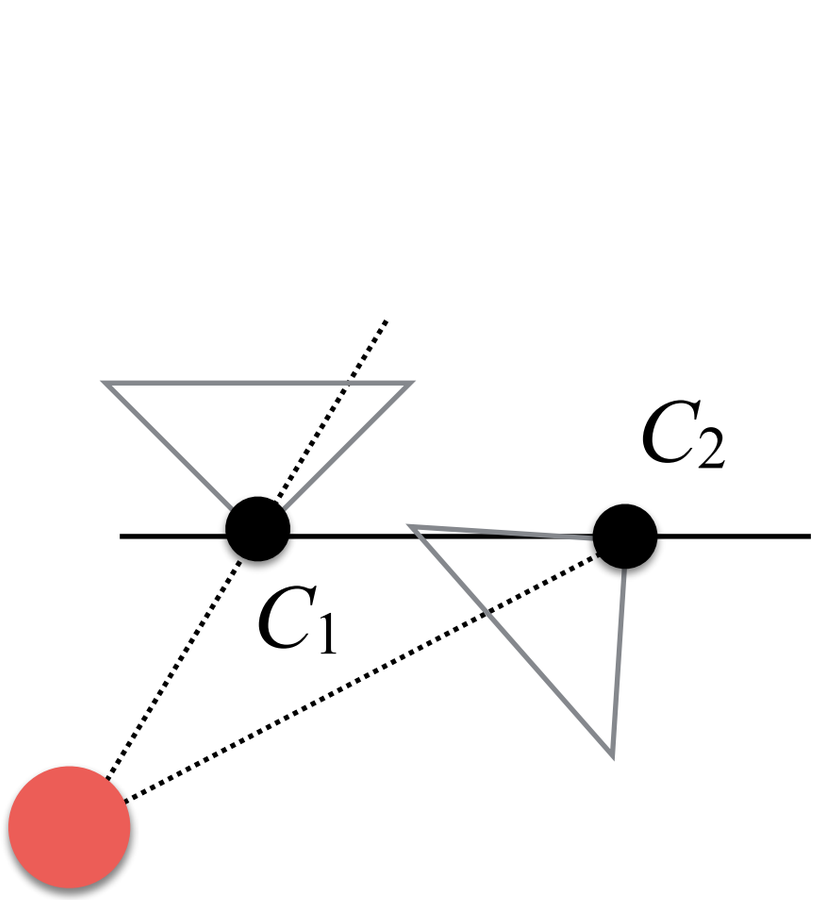


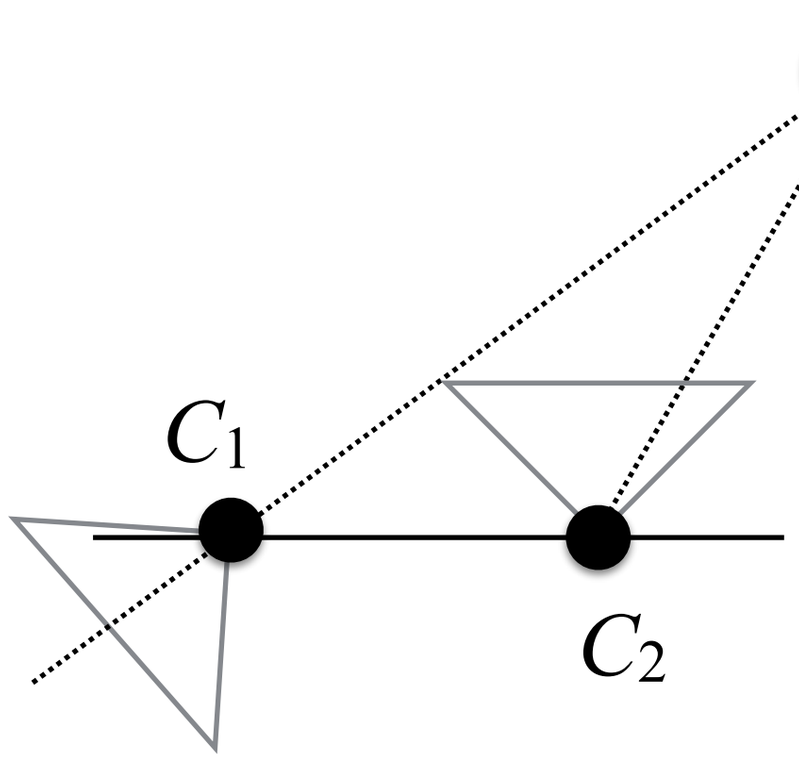
Both cameras see the point. It is in front of them



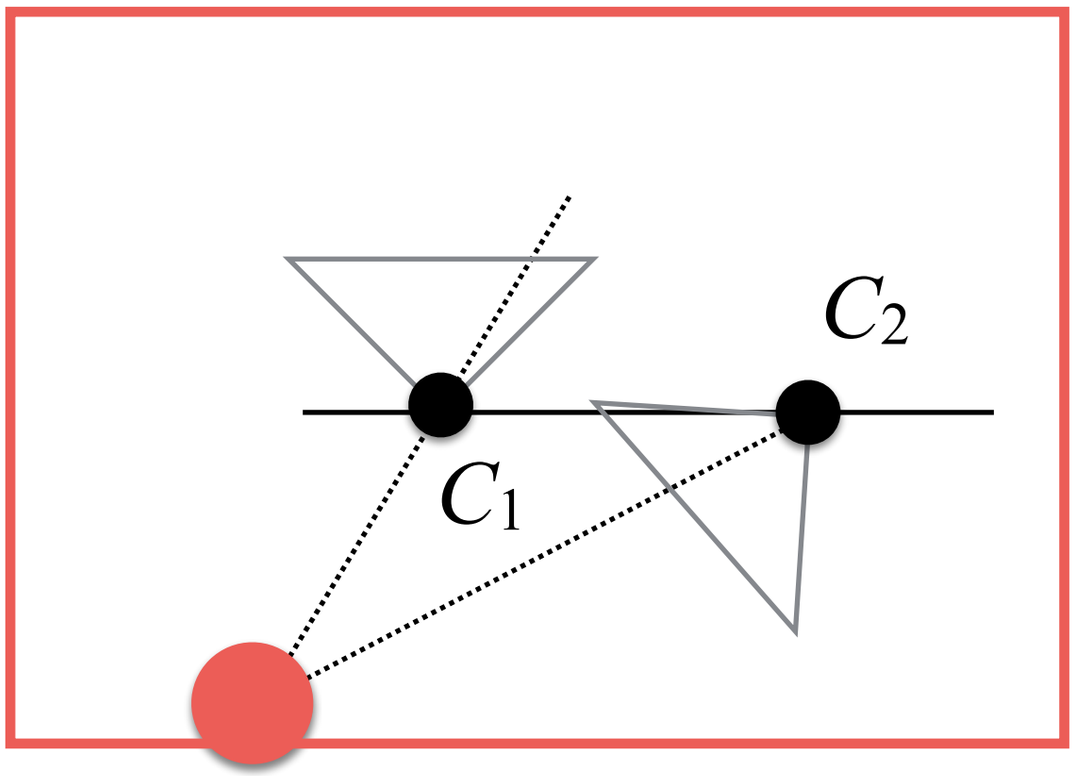
Both cameras don't see the point. It is behind them

E Factorization: The Four Cases

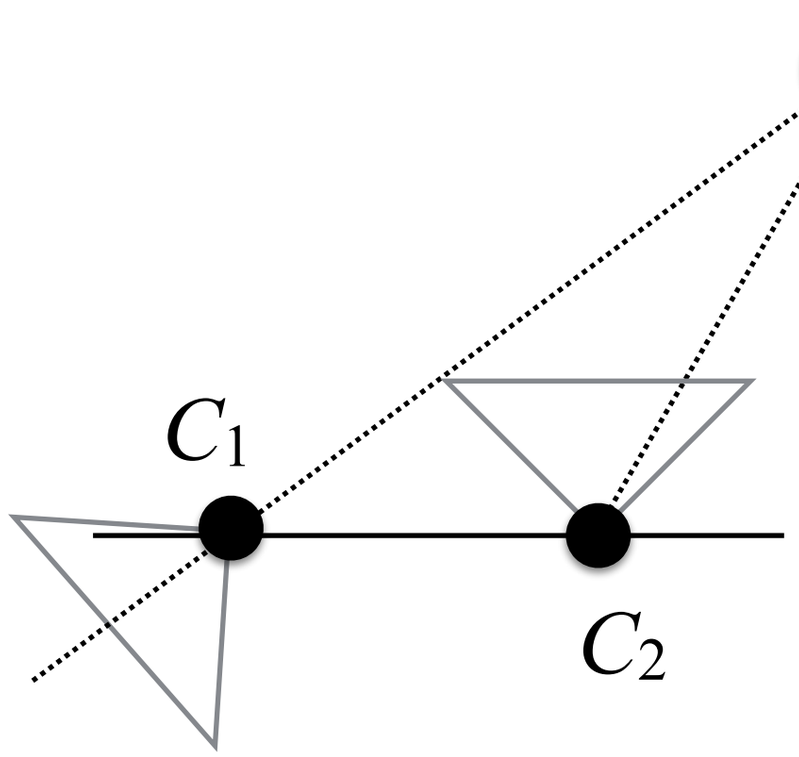




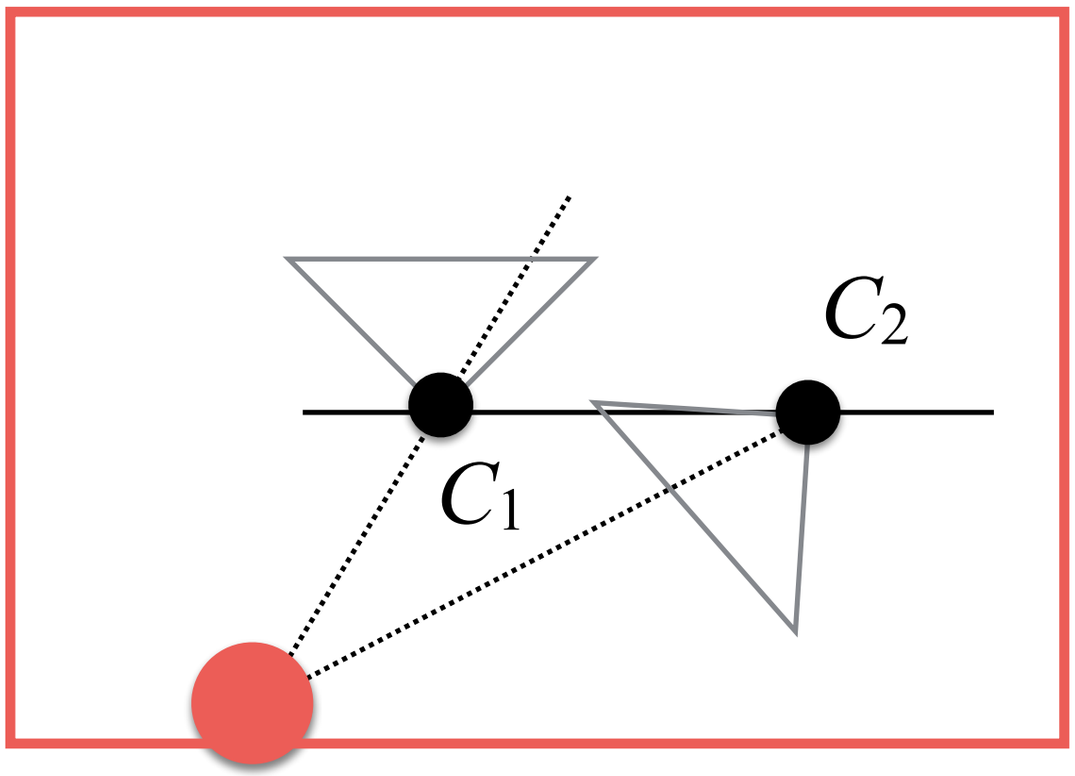
E Factorization: The Four Cases



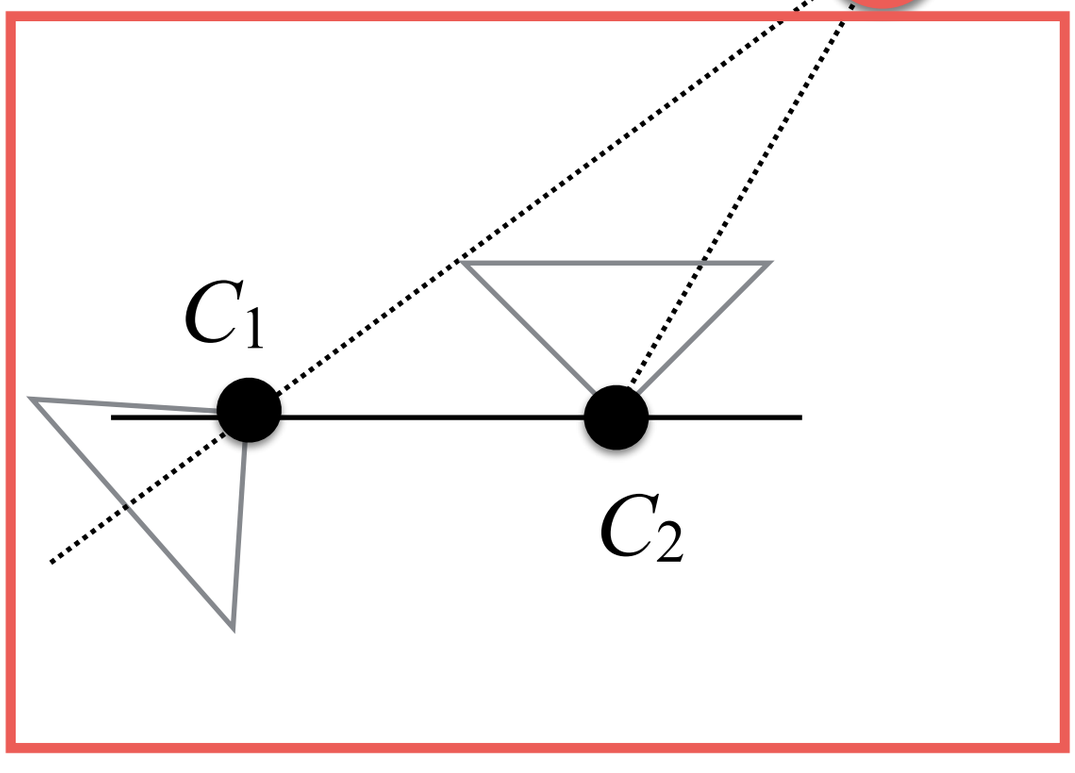
Left camera doesn't see the point. It is behind them



E Factorization: The Four Cases

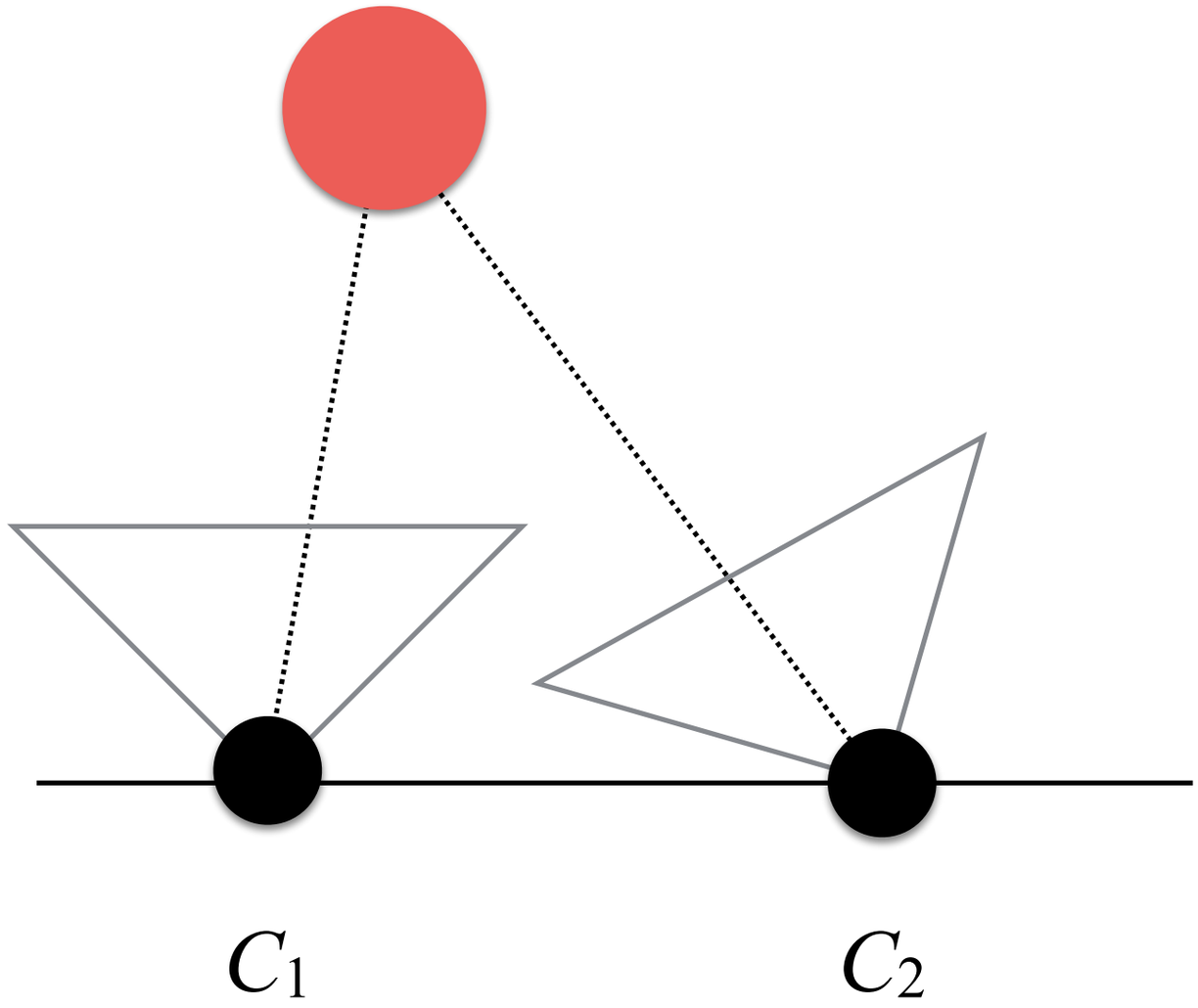


Left camera doesn't see the point. It is behind them



Left camera doesn't see the point. It is behind them

Which is the correct configuration?


C_1
C_2

Why?

The ***red point*** is seen by both cameras!

How do we find it?

We need to compute all cases.
Then, we determine the case in which the majority of 3D points are in front of both cameras!

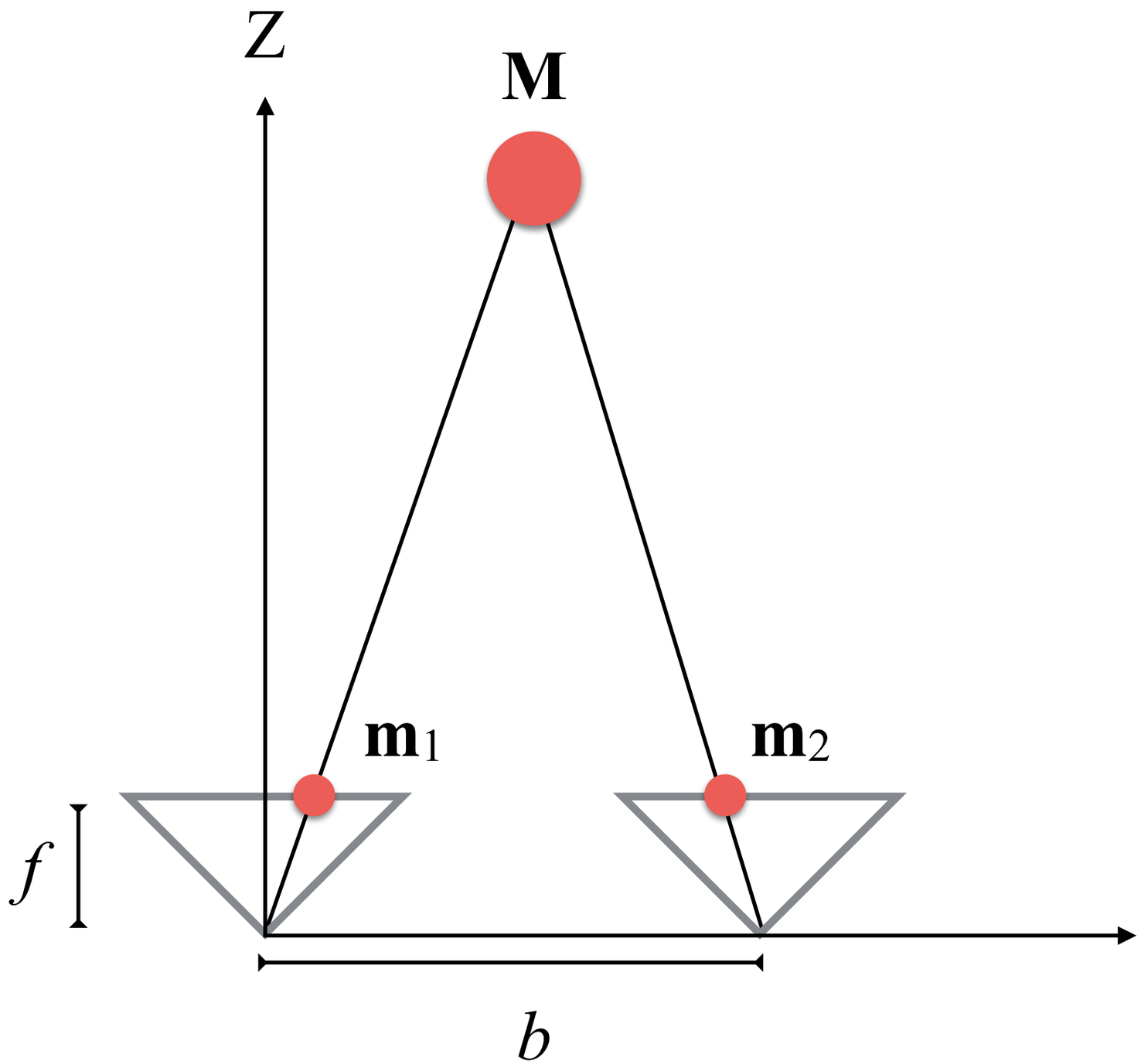
Triangulation

Triangulation

- **Input**: n matched 2D feature points in the two images and their P matrices (P_1, and P_2).

- **Output**: n 3D points.

Triangulation: Pure Translational Motion Case



$$\mathbf{M} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\mathbf{m}_i = \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix}$$

Triangulation

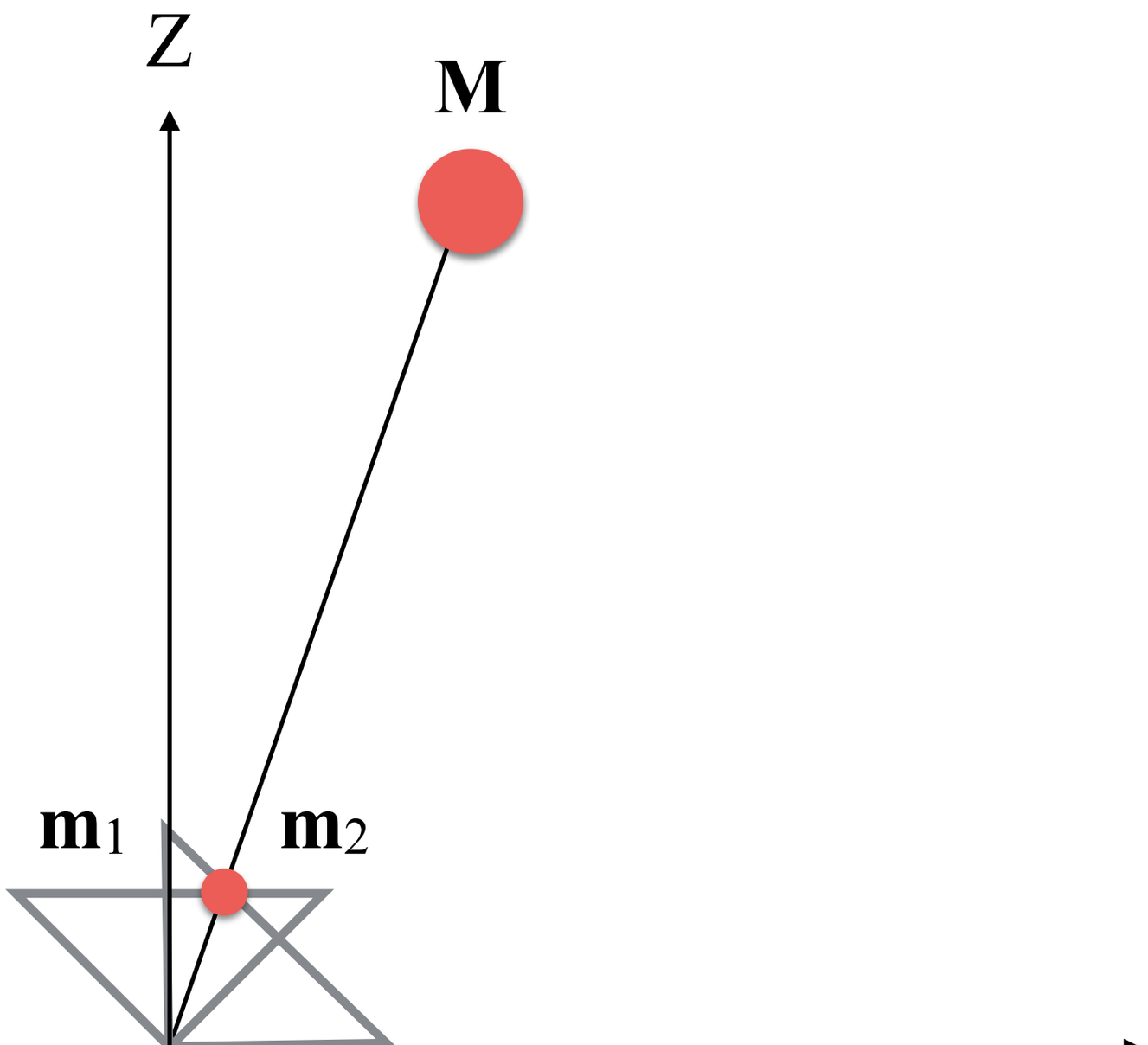
- We first fix the frame of reference to one of the two cameras. Then, we know that:

$$\begin{cases} \frac{f}{z} = -\frac{u_1}{x} \\ \frac{f}{z} = -\frac{u_2}{x-b} \end{cases}$$

- So, we can obtain:

$$z = \frac{b \cdot f}{u_2 - u_1}$$

Triangulation: Pure Rotational Motion Case

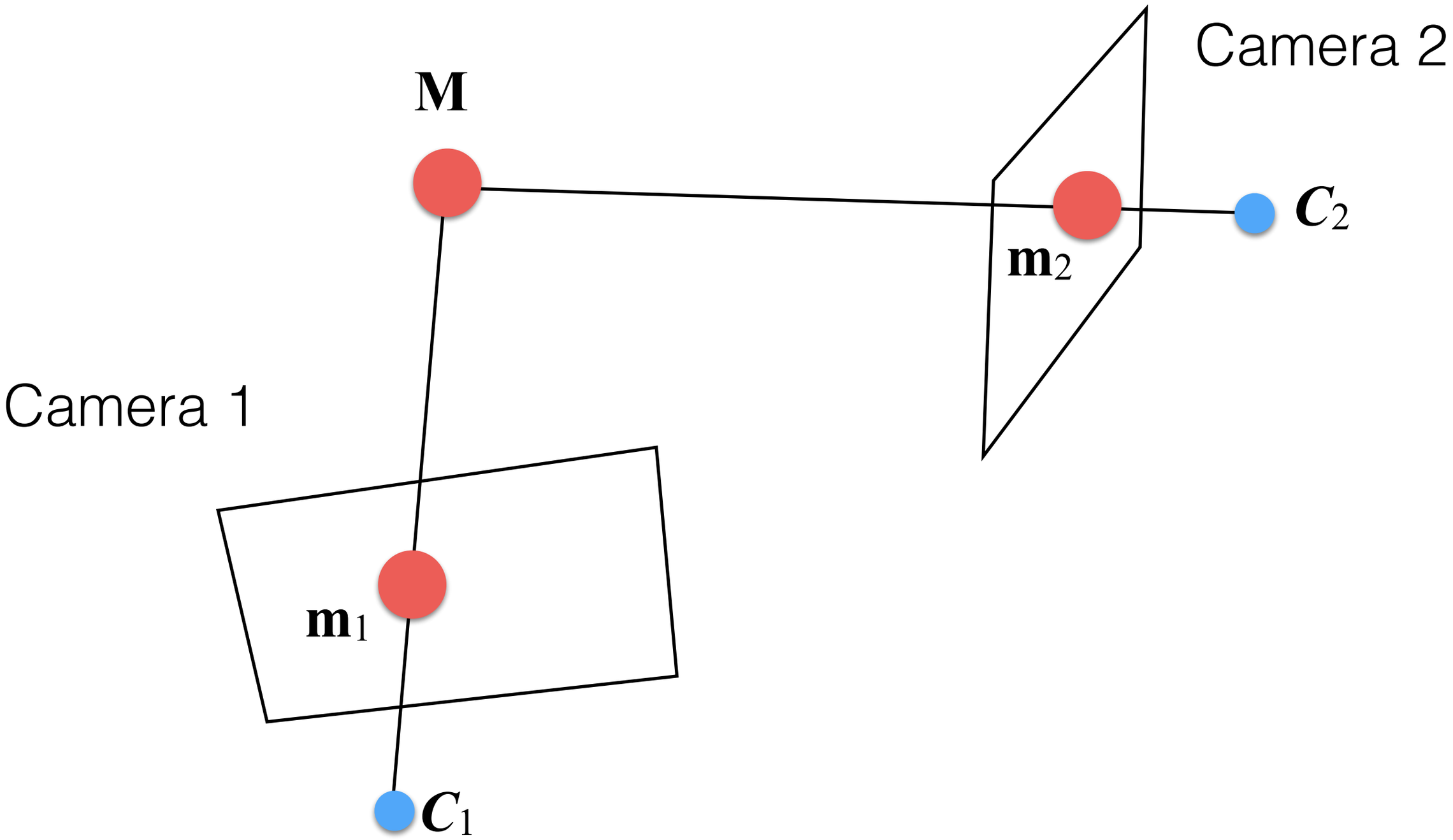


$$\mathbf{M} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\mathbf{m}_i = \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix}$$

There is no displacement —> The same lines for intersection —> no 3D

Triangulation:
The General Case



Similar to DLT

but different!

Triangulation: Eigen Method

$$P = \begin{bmatrix} \mathbf{p}_1^\top \\ \mathbf{p}_2^\top \\ \mathbf{p}_3^\top \end{bmatrix} \qquad \begin{cases} u = \dfrac{\mathbf{p}_1^\top \cdot \mathbf{M}}{\mathbf{p}_3^\top \cdot \mathbf{M}} \\ v = \dfrac{\mathbf{p}_2^\top \cdot \mathbf{M}}{\mathbf{p}_3^\top \cdot \mathbf{M}} \end{cases}$$

Triangulation: Eigen Method

$$P = \begin{bmatrix} \mathbf{p}_1^\top \\ \mathbf{p}_2^\top \\ \mathbf{p}_3^\top \end{bmatrix}$$

$$\begin{cases} u = \dfrac{\mathbf{p}_1^\top \cdot \mathbf{M}}{\mathbf{p}_3^\top \cdot \mathbf{M}} \\ v = \dfrac{\mathbf{p}_2^\top \cdot \mathbf{M}}{\mathbf{p}_3^\top \cdot \mathbf{M}} \end{cases}$$

known!

Triangulation: Eigen Method

$$P = \begin{bmatrix} \mathbf{p}_1^\top \\ \mathbf{p}_2^\top \\ \mathbf{p}_3^\top \end{bmatrix}$$

$$\begin{cases} u = \dfrac{\mathbf{p}_1^\top \cdot \mathbf{M}}{\mathbf{p}_3^\top \cdot \mathbf{M}} \\ v = \dfrac{\mathbf{p}_2^\top \cdot \mathbf{M}}{\mathbf{p}_3^\top \cdot \mathbf{M}} \end{cases}$$

known!

unknown!

Triangulation: Eigen Method

- This leads to:

$$\begin{cases} (\mathbf{p}_1 - u \cdot \mathbf{p}_1)^\top \cdot \mathbf{M} = 0 \\ (\mathbf{p}_2 - v \cdot \mathbf{p}_1)^\top \cdot \mathbf{M} = 0 \end{cases}$$

- Given that:

$$P_i = \begin{bmatrix} (\mathbf{p}_1^i)^\top \\ (\mathbf{p}_2^i)^\top \\ (\mathbf{p}_3^i)^\top \end{bmatrix} \qquad \mathbf{m}_i = \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix}$$

Triangulation: Eigen Method

- We obtain:

$$\begin{bmatrix} (\mathbf{p}_1^1 - u_1 \cdot \mathbf{p}_3^1)^\top \\ (\mathbf{p}_2^1 - v_1 \cdot \mathbf{p}_3^1)^\top \\ (\mathbf{p}_1^2 - u_2 \cdot \mathbf{p}_3^2)^\top \\ (\mathbf{p}_2^2 - v_2 \cdot \mathbf{p}_3^2)^\top \end{bmatrix} \cdot \mathbf{M} = \mathbf{0}$$

- For l cameras, this leads to:

$$\begin{bmatrix} (\mathbf{p}_1^1 - u_1 \cdot \mathbf{p}_3^1)^\top \\ (\mathbf{p}_2^1 - u_1 \cdot \mathbf{p}_3^1)^\top \\ \vdots \\ (\mathbf{p}_1^l - u_1 \cdot \mathbf{p}_3^l)^\top \\ (\mathbf{p}_2^l - u_1 \cdot \mathbf{p}_3^l)^\top \end{bmatrix} \cdot \mathbf{M} = \mathbf{0}$$

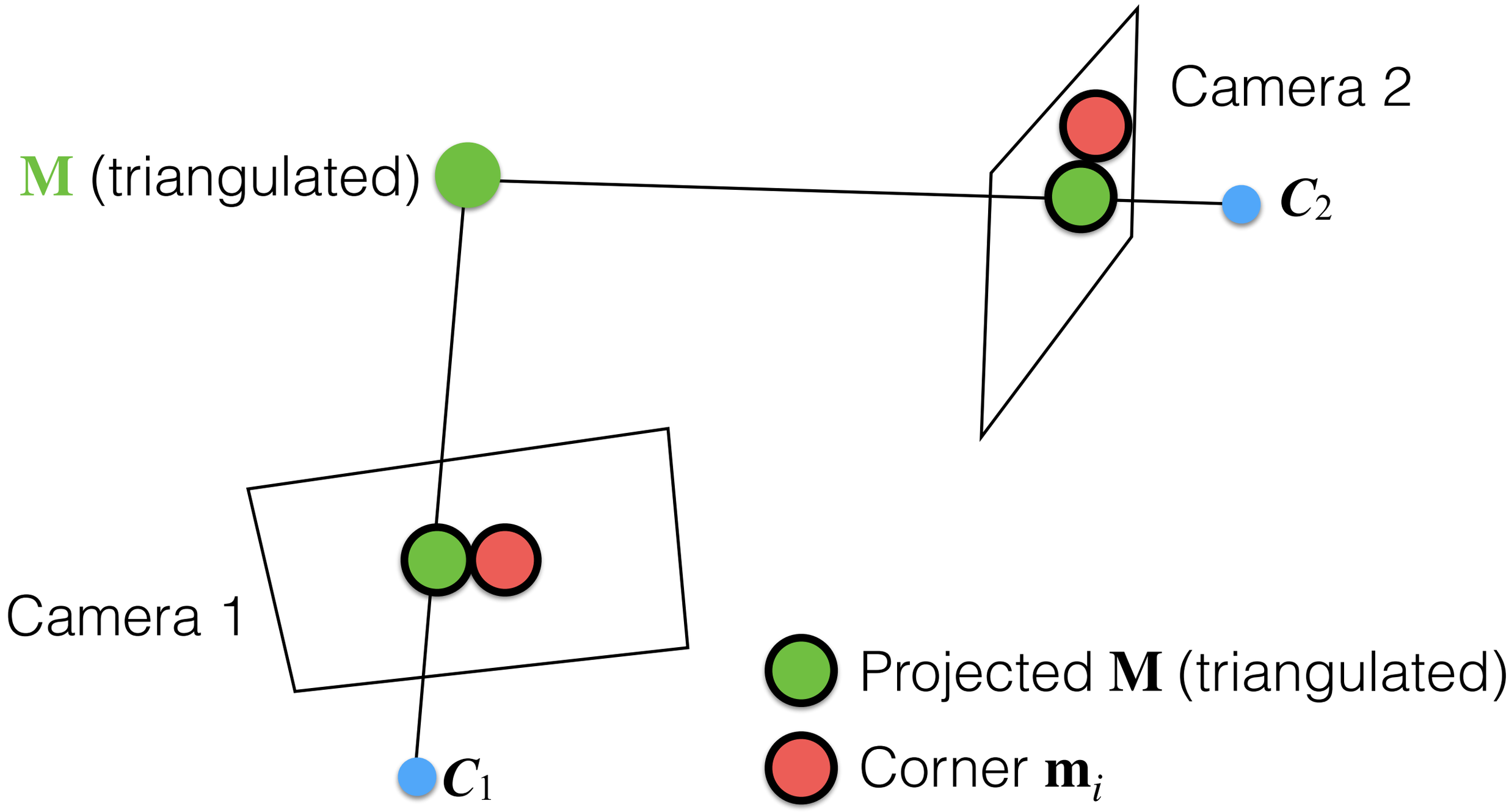
Triangulation: Eigen Method

- Again, we solve this linear system using SVD; i.e., the kernel of V:
  - Again, we minimized an algebraic error without a geometric meaning!
- Again, we use this initial solution for a non-linear method that minimizes a geometric error. Here for each 3D points $\mathbf{M}$, we minimize:

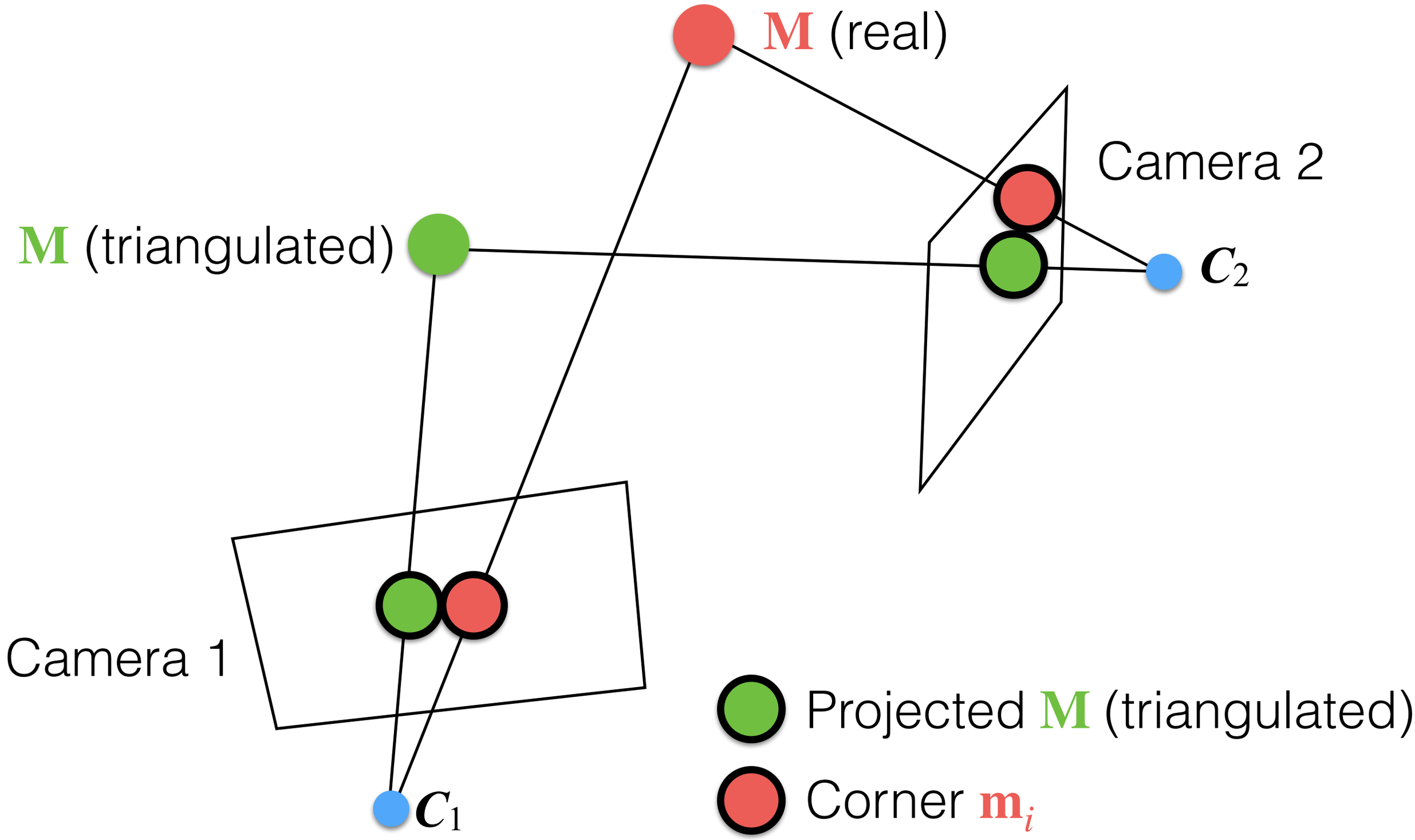
$$\arg\min_{\mathbf{M}} \sum_{j=1}^{l} \left(u_j - \frac{(\mathbf{p}_1^j)^\top \cdot \mathbf{M}}{(\mathbf{p}_3^j)^\top \cdot \mathbf{M}} \right)^2 + \left(v_j - \frac{(\mathbf{p}_2^j)^\top \cdot \mathbf{M}}{(\mathbf{p}_3^j)^\top \cdot \mathbf{M}} \right)^2$$

- where l is the number of cameras. $\begin{bmatrix} u_j & v_j & 1 \end{bmatrix}^\top$ is corner in the l-th camera (i.e., it is “the photograph” of $\mathbf{M}$).

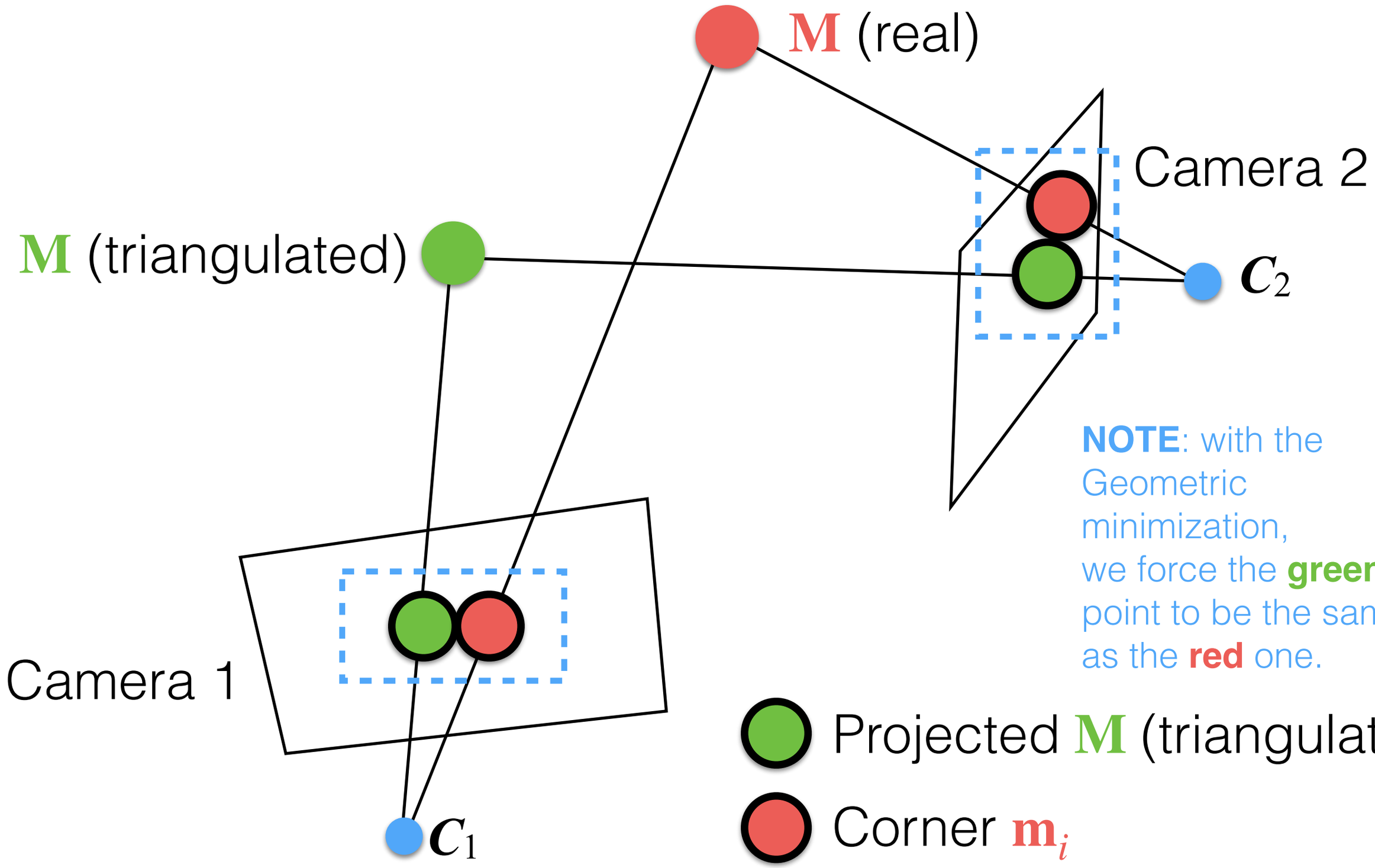
Triangulation: Eigen Method Example



Triangulation: Eigen Method Example

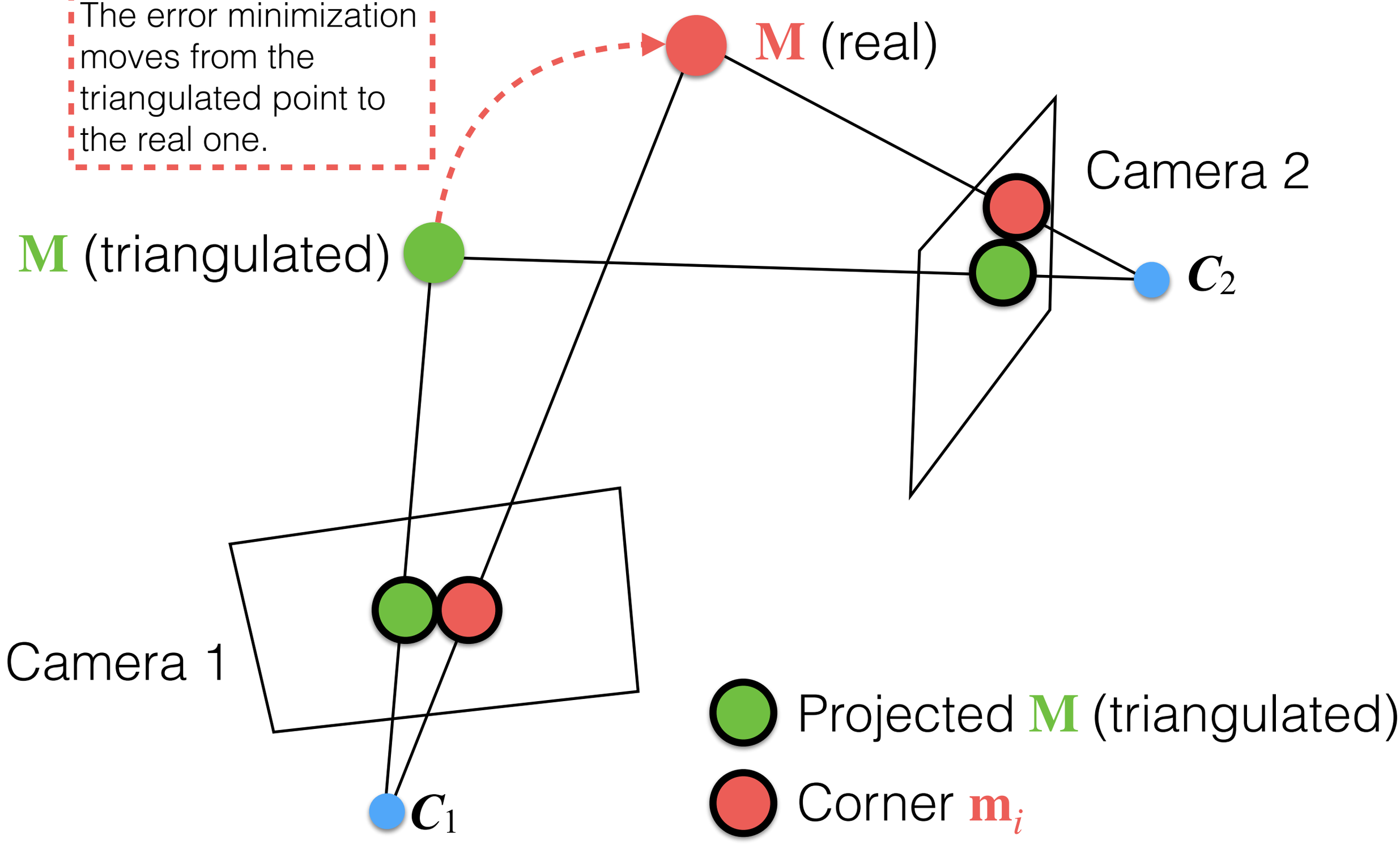


Triangulation: Eigen Method Example



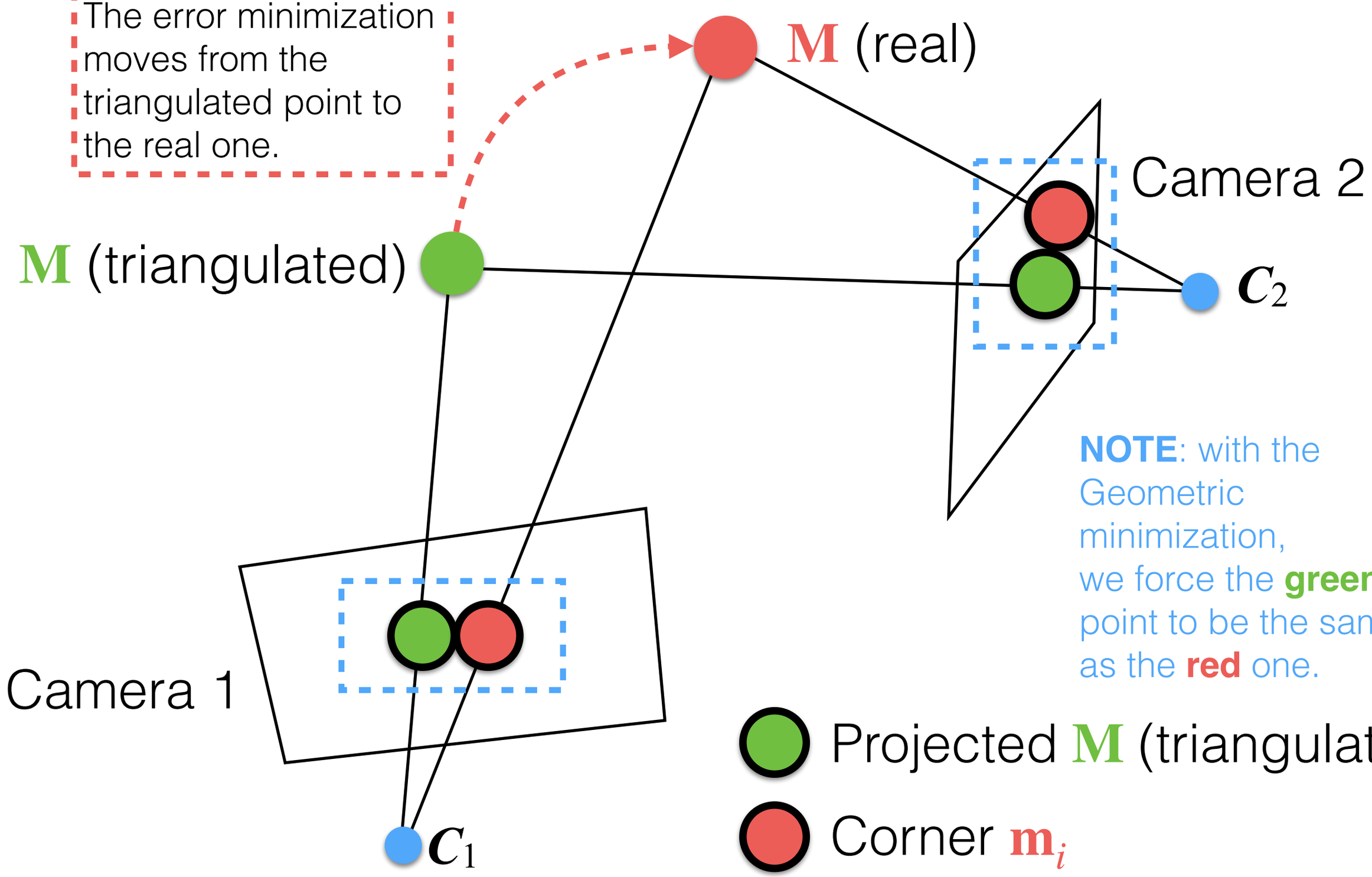
Triangulation: Eigen Method Example

The error minimization moves from the triangulated point to the real one.



Triangulation: Eigen Method Example

The error minimization moves from the triangulated point to the real one.



NOTE: with the Geometric minimization, we force the **green** point to be the same as the **red** one.

Projected $\mathbf{M}$ (triangulated)

Corner $\mathbf{m}_i$

Structure from Motion

Structure from Motion

- **Input**: n matched points (corners computed with the Harris algorithm) between two images, and K_1 and K_2 for the two cameras.

- **Output**: n 3D points, and G_1 and G_2 (i.e., for each camera).

Structure from Motion

- The algorithm is:
  - Estimation of E.
  - Factorization of E to obtain G.
  - Triangulation of the n matched points using P_1 and P_2.

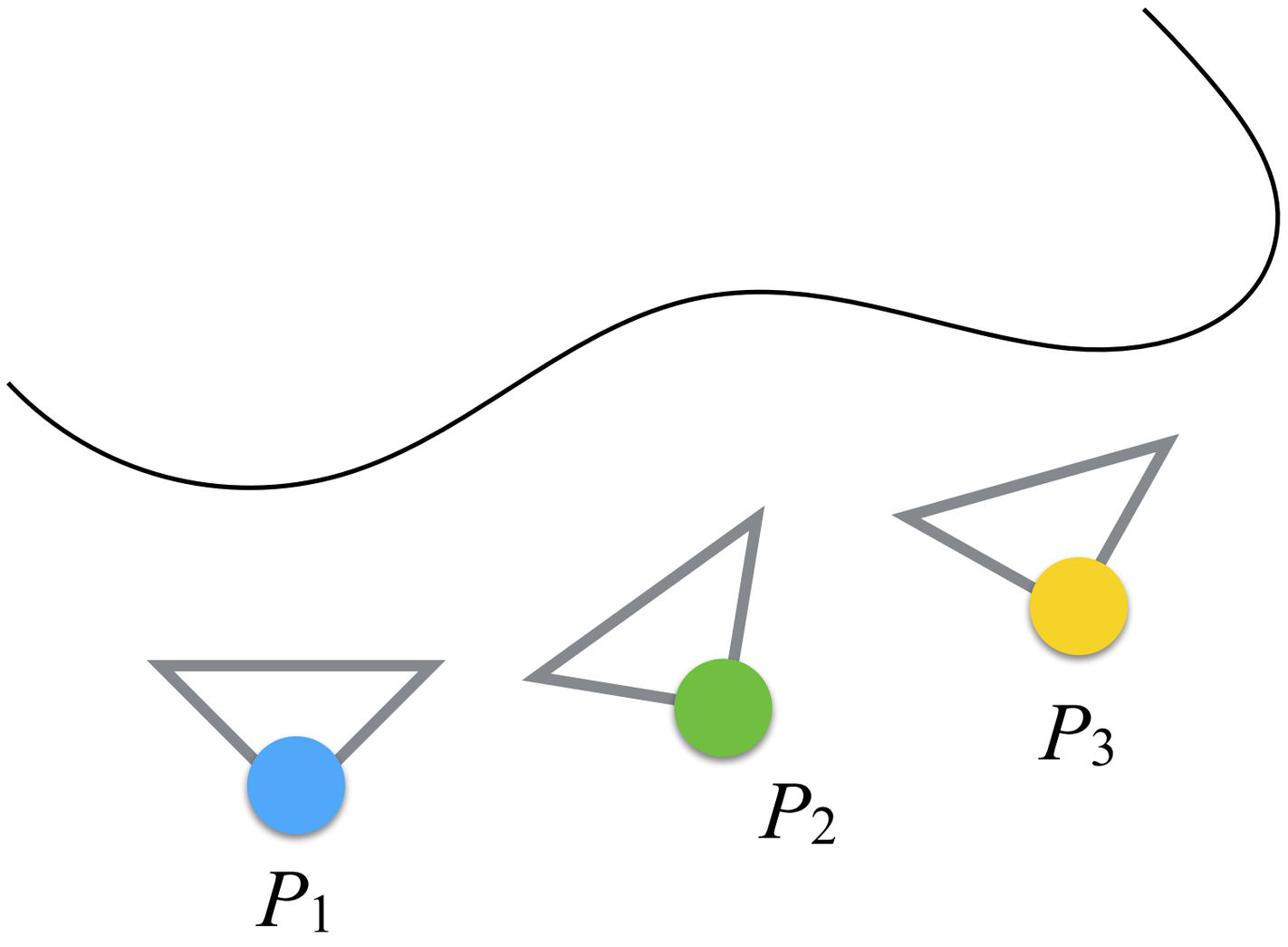
So far we have only used only a two cameras!

Structure from Motion: Multi-View

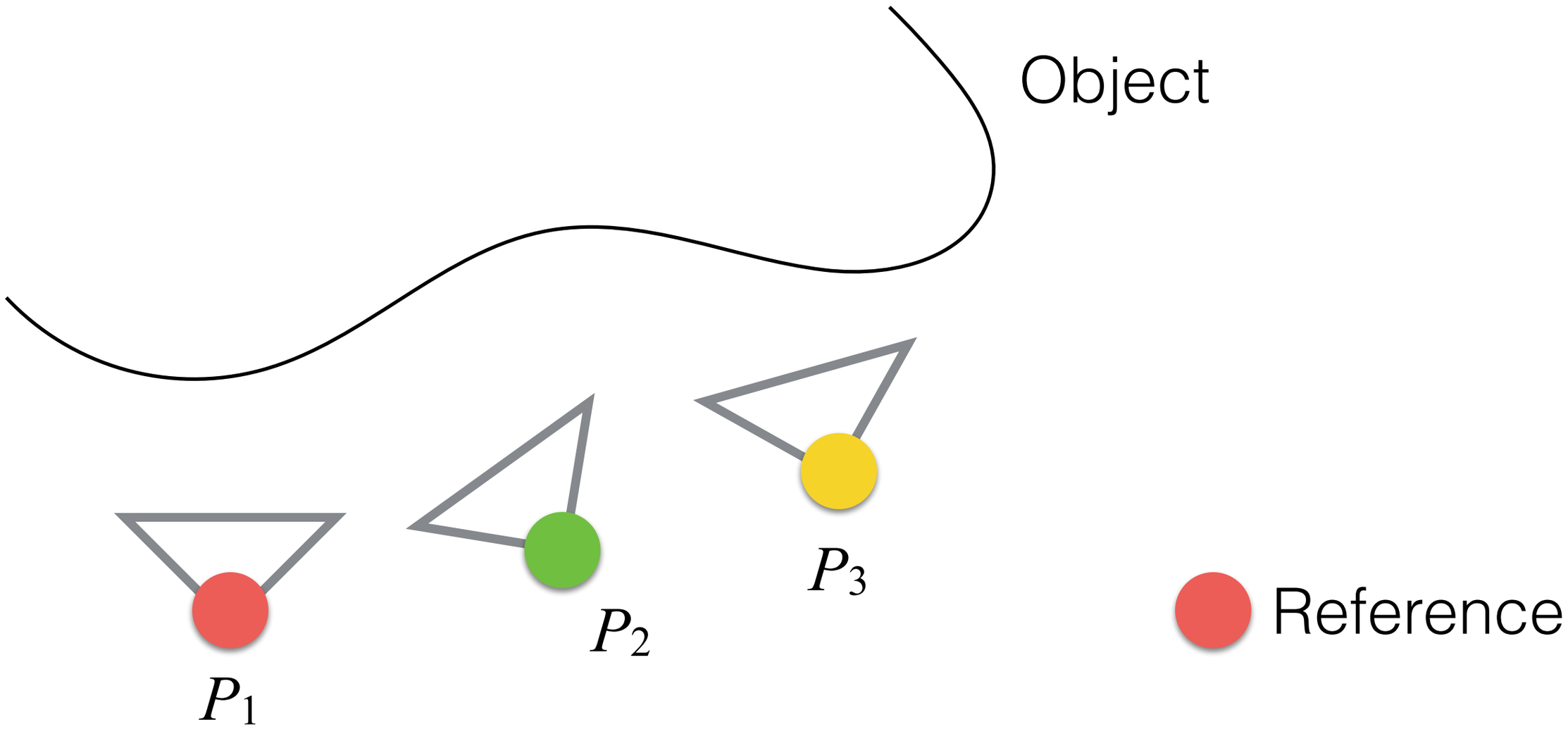
- We compute G for different views using the previous algorithm.

- We use a **reference** view for computing the different G matrices. For example, we can use the first image or another criteria.

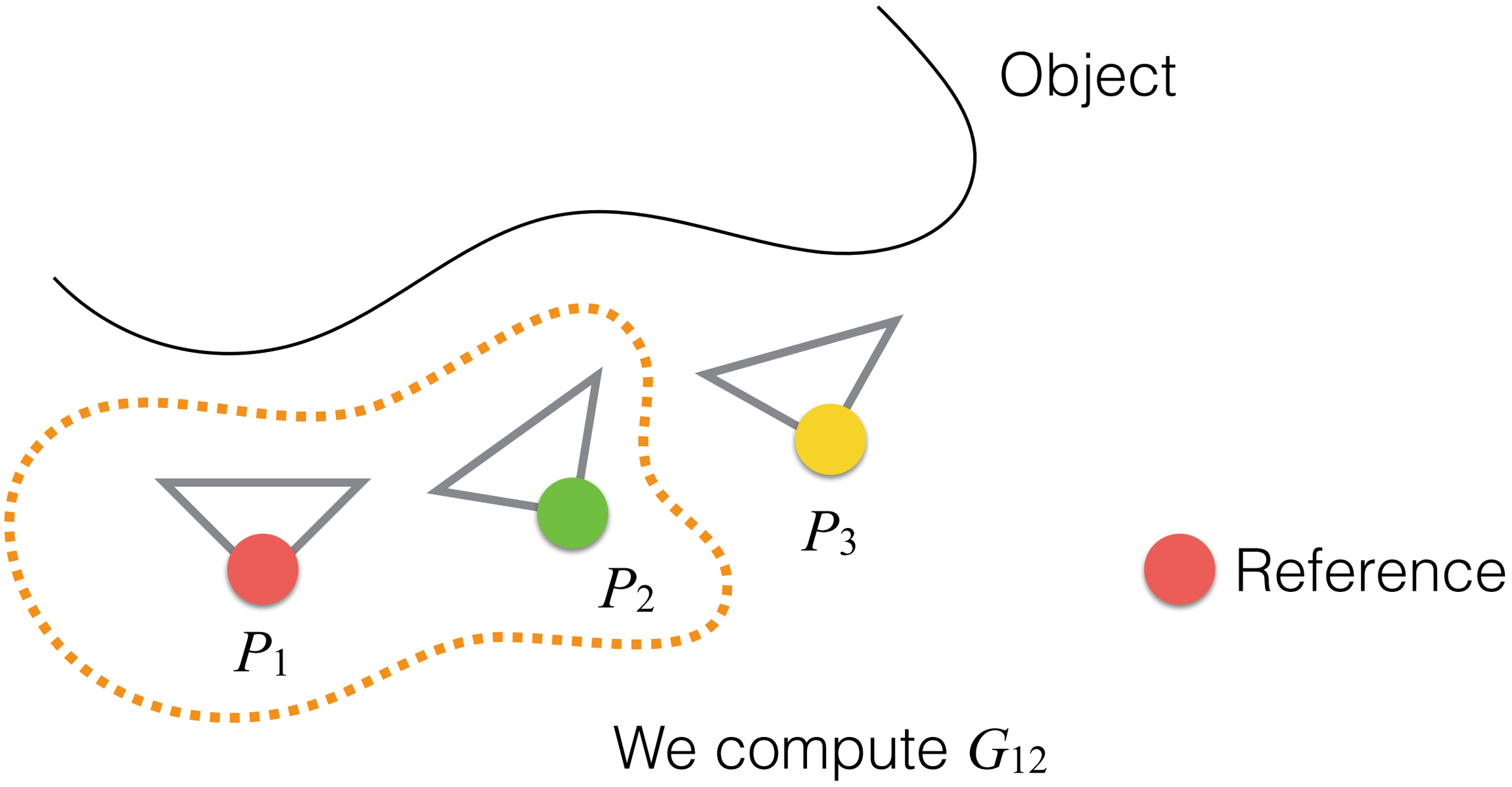
Structure from Motion: Multi-View Example



Structure from Motion: Multi-View Example

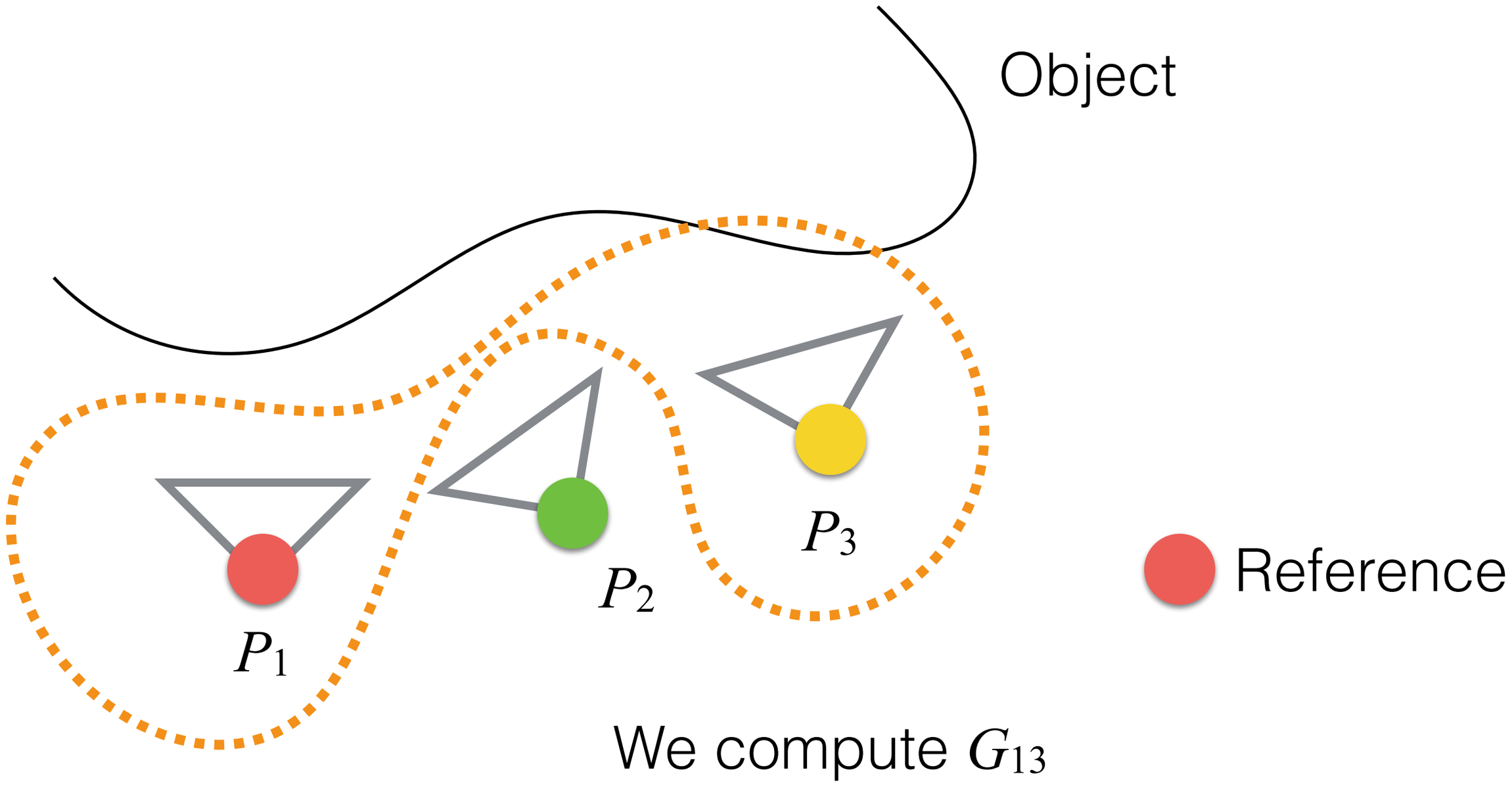


Structure from Motion: Multi-View Example



We compute G_{12}

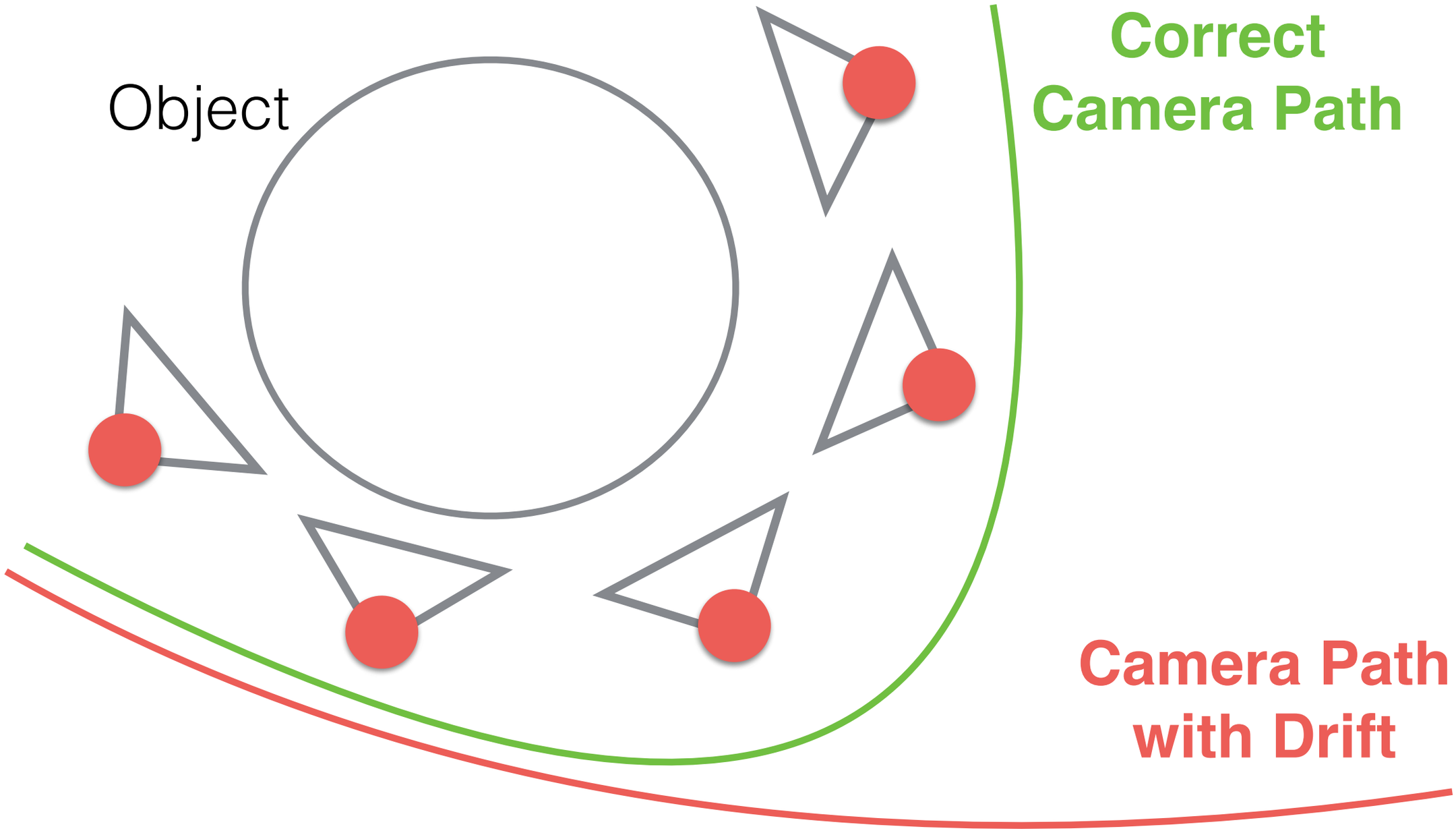
Structure from Motion: Multi-View Example



Structure from Motion: Multi-View

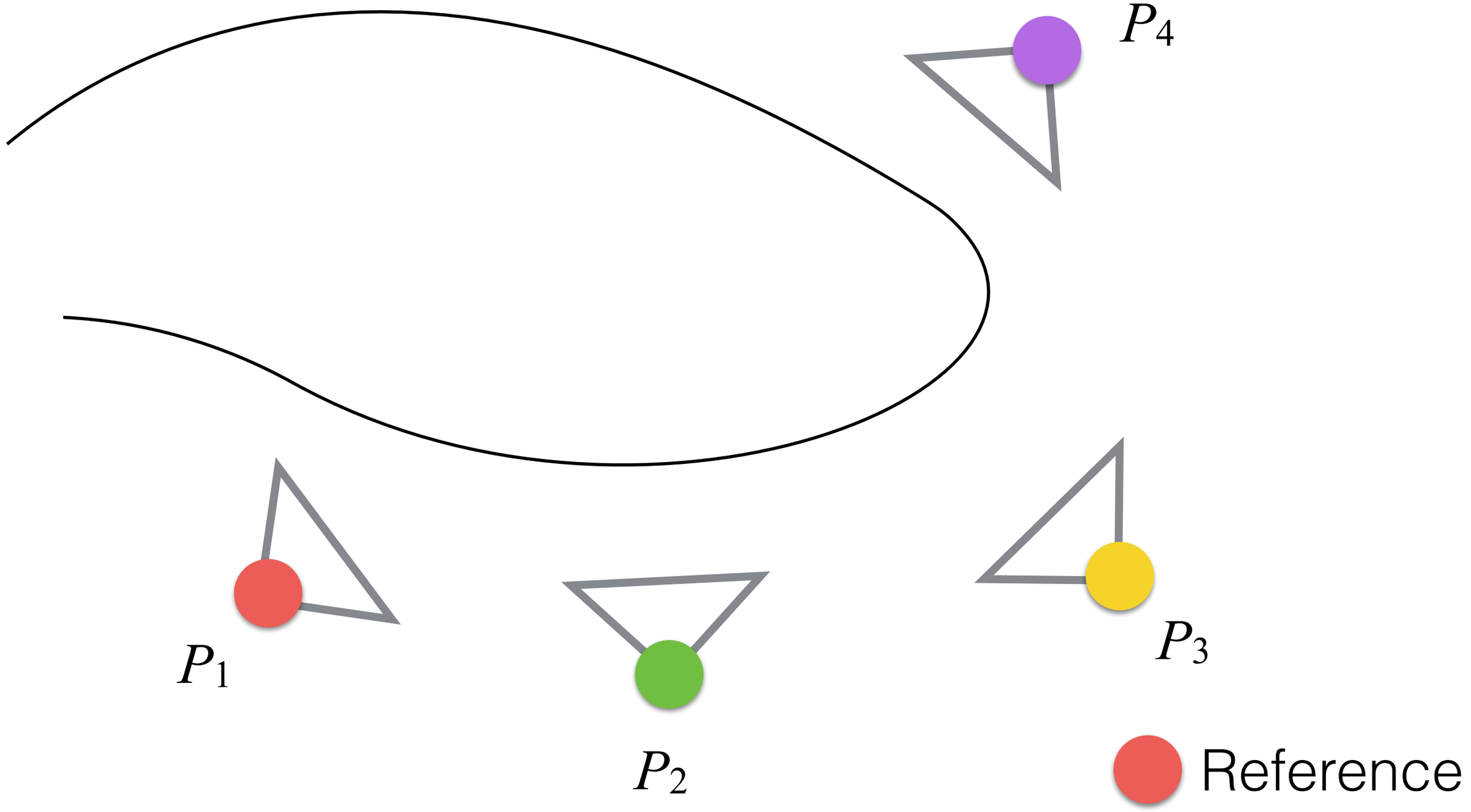
- Note that we could compute G matrices incrementally. For the previous example that means:
  - First, we compute G_{12} and G_{23}.
  - Second, we compute absolute G matrices (where the reference point is the first view). In this simple case, we need to compute G_{13} as:
    - $G_{13} = G_{12} \cdot G_{23}$
- Note that in doing, especially with many cameras, we may incur in accumulation error that leads to drift.

Structure from Motion: Drift Example

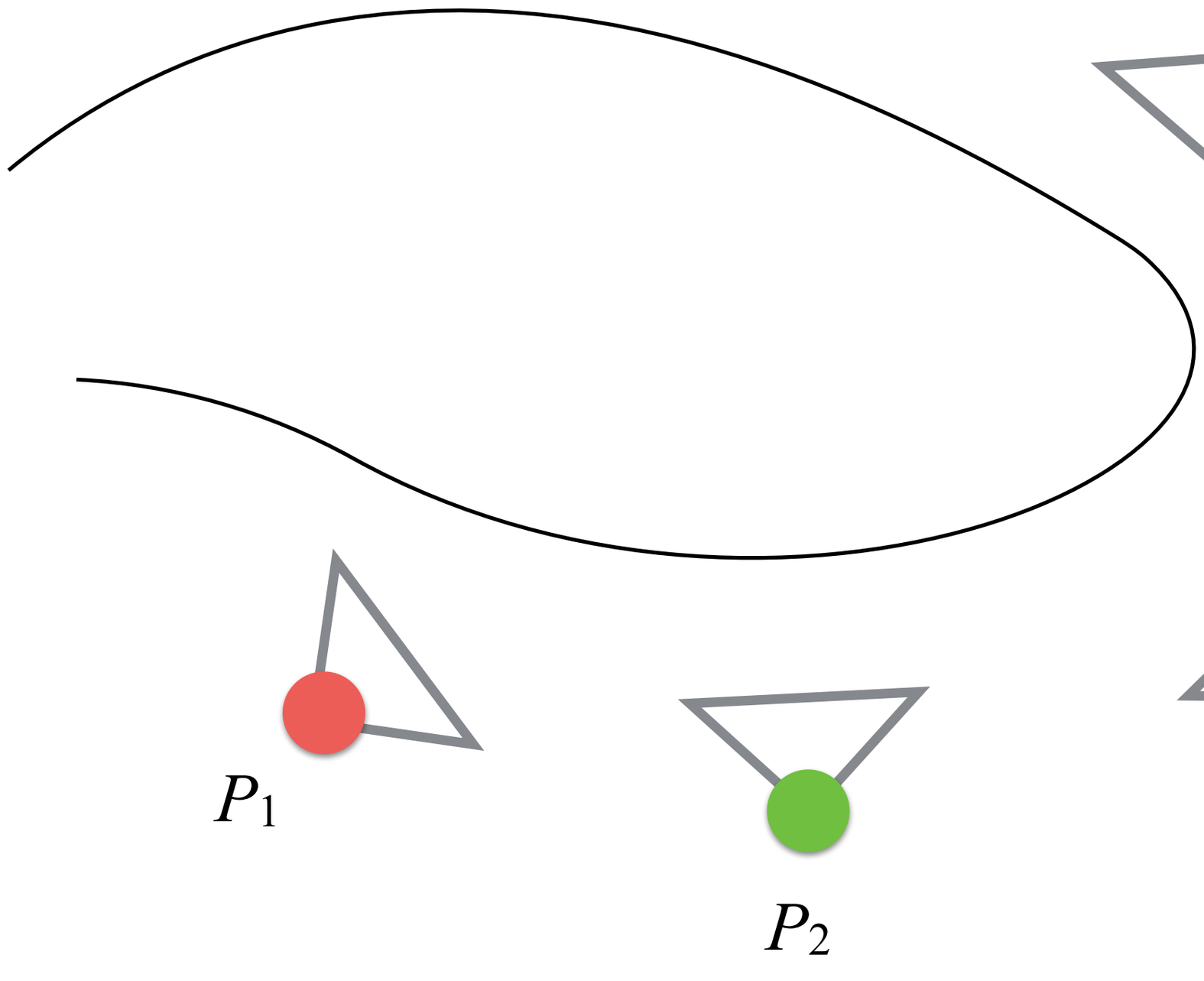


Note that, we need to change the reference after a while because it will be not be anymore valid

Structure from Motion: Invalid Reference example



Structure from Motion: Invalid Reference example



P_1 is not a valid reference For P_4 because the do not share features!
At this point we need to move the reference to have some shared features; e.g., P_3.

Reference

Hang on, was it a good reference the one before?

Hang on, what can possibly go wrong?

We are still accumulating error (we need to change the reference after a while), and we will drift from the solution!

Structure from Motion: Bundle Adjustment

- To avoid error accumulation, we minimize in a non-linear way at the same time both poses estimation and 3D points generation:

$$\arg \min_{R^i, \mathbf{t}^i, \mathbf{M}^i} \sum_{j=1}^{l} \sum_{i=1}^{n} \left(u_j^i - \frac{(\mathbf{p}_1^j)^\top \cdot \mathbf{M}^i}{(\mathbf{p}_3^j)^\top \cdot \mathbf{M}^i} \right)^2 + \left(v_j^i - \frac{(\mathbf{p}_2^j)^\top \cdot \mathbf{M}^i}{(\mathbf{p}_3^j)^\top \cdot \mathbf{M}^i} \right)^2$$

$$P = K \cdot [R|\mathbf{t}]$$

- where l is the number of cameras, and n is the number of points.

Structure from Motion: Bundle Adjustment

- Typically, the method is difficult to minimize as a whole thing. This is because there are many parameters to minimize.

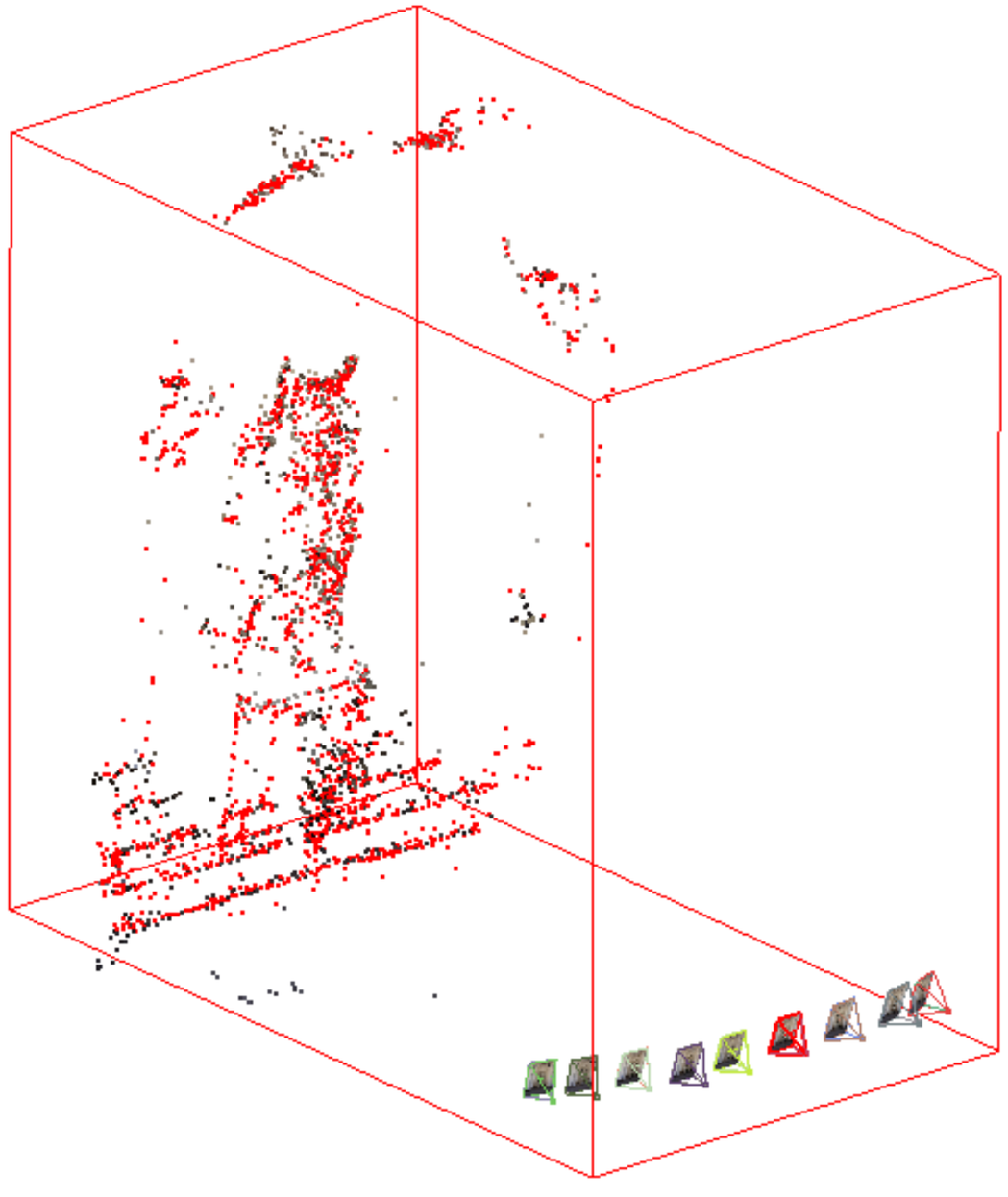
- A two-step approach:

  - First, minimize (or viceversa) all extrinsic parameters (G) without modifying the 3D points.

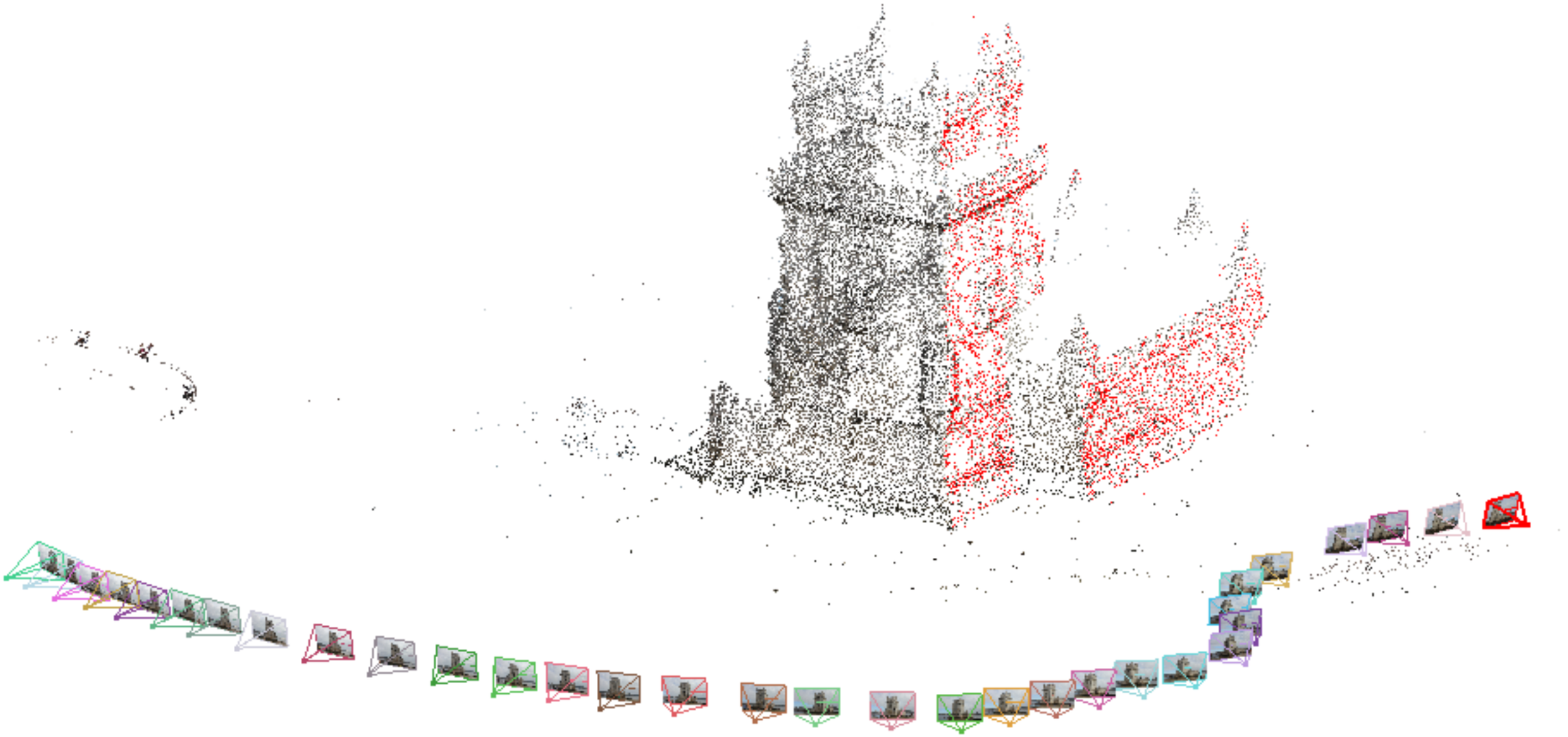
  - Then, minimize (or viceversa) 3D points coordinates without modifying G.

Structure from Motion: Example

Structure from Motion: Example



Structure from Motion: Example

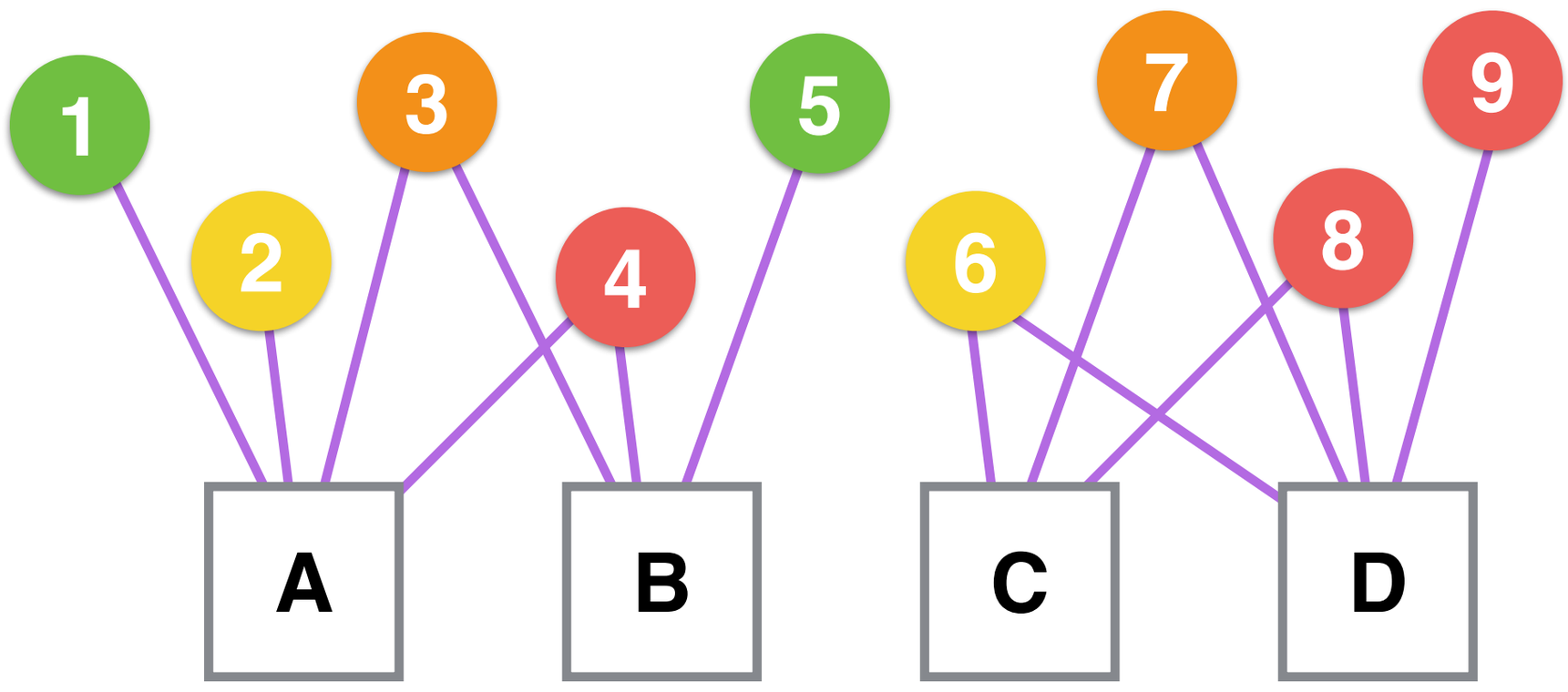


Structure From Motion: Multi-View

- To obtain something of interesting:
  - we need to feed into the system hundreds of images.
  - we need to manage thousands of features (corners)!
- Even the two-step approach would struggle a bit.

Structure from Motion: Multi-View

- To make the problem computational tractable, we can notice this:



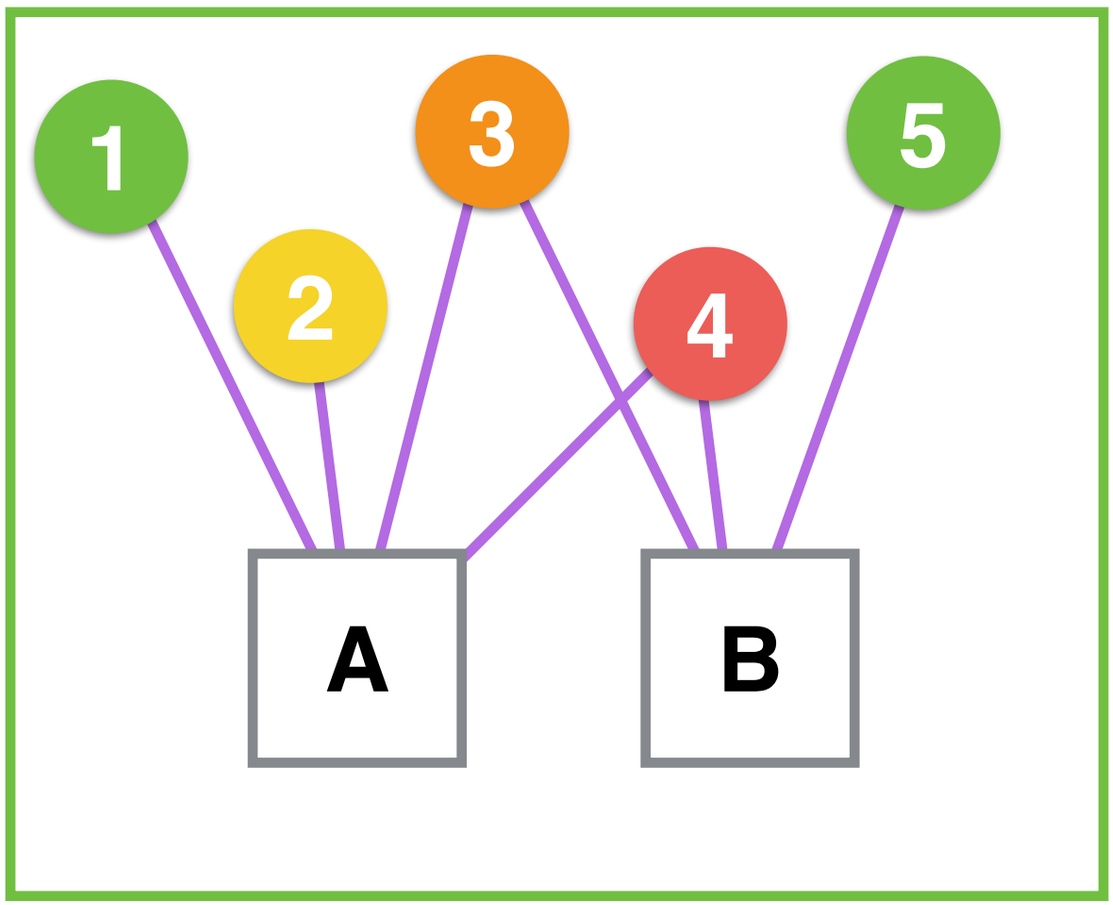
1,2,3,4,5,6,7,8, and 9 are points
A,B,C, and D are cameras

the point is seen in a camera

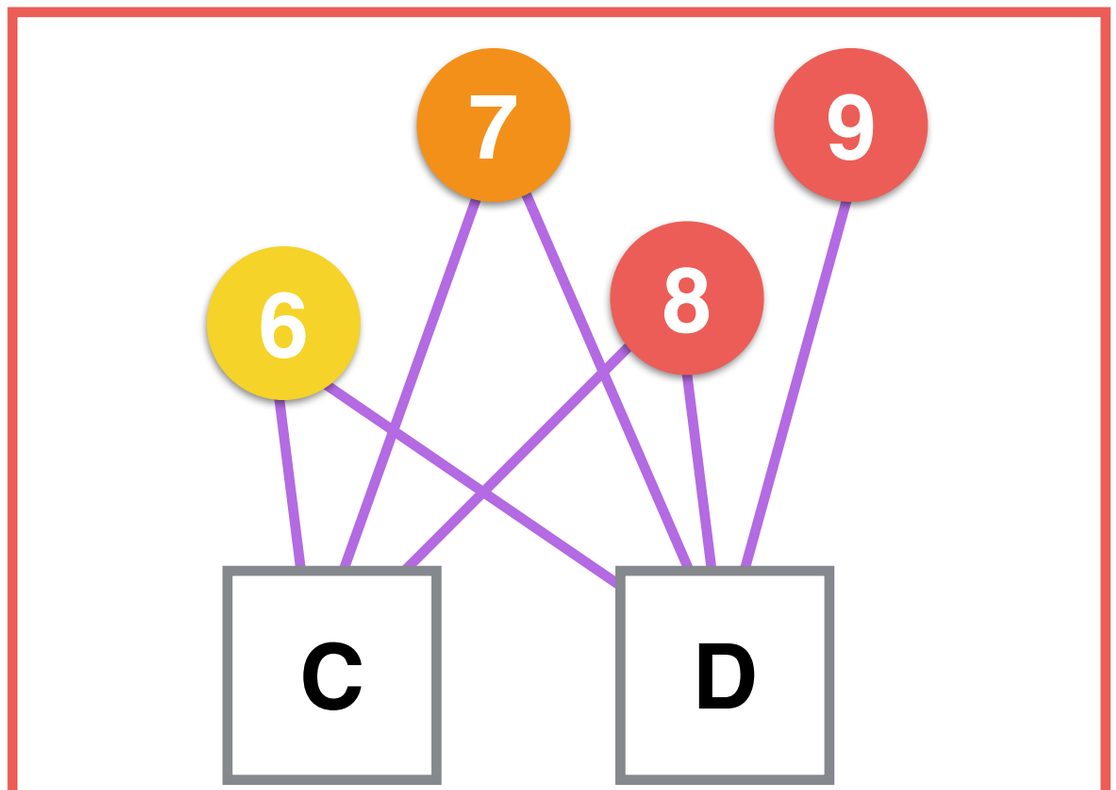
Structure from Motion: Multi-View

- The idea is to divide the scene into clusters.
- For each cluster we compute SfM.
- We combine all 3D reconstructions and camera poses together.

Structure from Motion: Multi-View



Cluster A



Cluster B

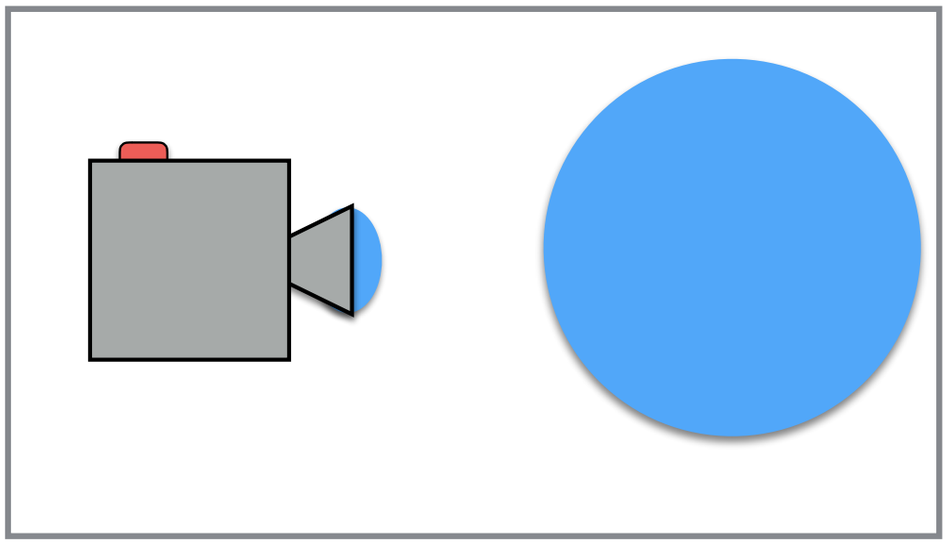
Structure from Motion: Conclusions

- Advantages:
  - It requires only photographs/videos: cheap and fast.
  - We can recover color information from photographs!
- Disadvantages:
  - The output model may be skewed; it is hard to keep two things going at the same time (3D points and cameras' poses).
  - We do not have the scale of the 3D scene; we need some reference measures.

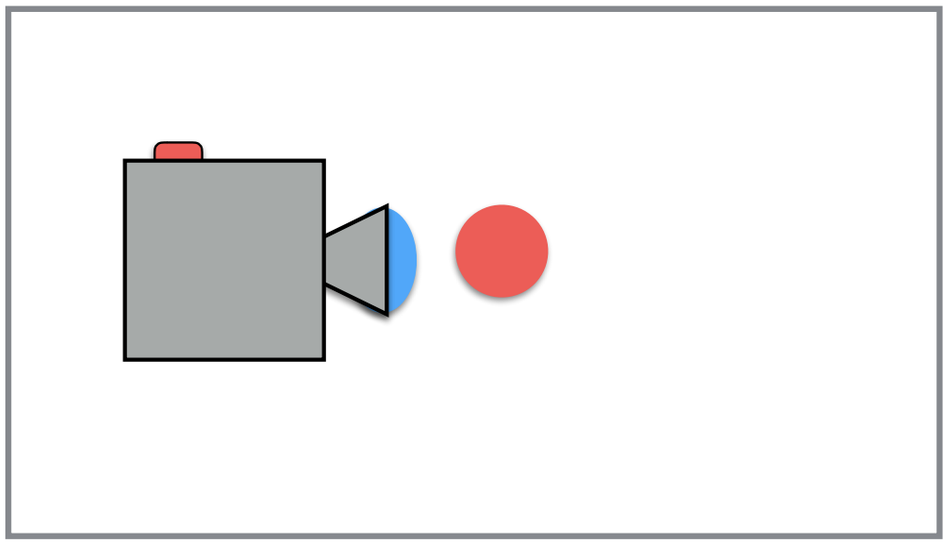
Structure from Motion: The Scale of a Scene

- The scale factor is due to the fact we do not know the real scale of a scene.
- If we take a photograph of a scene, we have two possible cases:
  - A **large** object **far away** from the camera.
  - A **small** object **near** the camera.

Structure from Motion: Scale Example

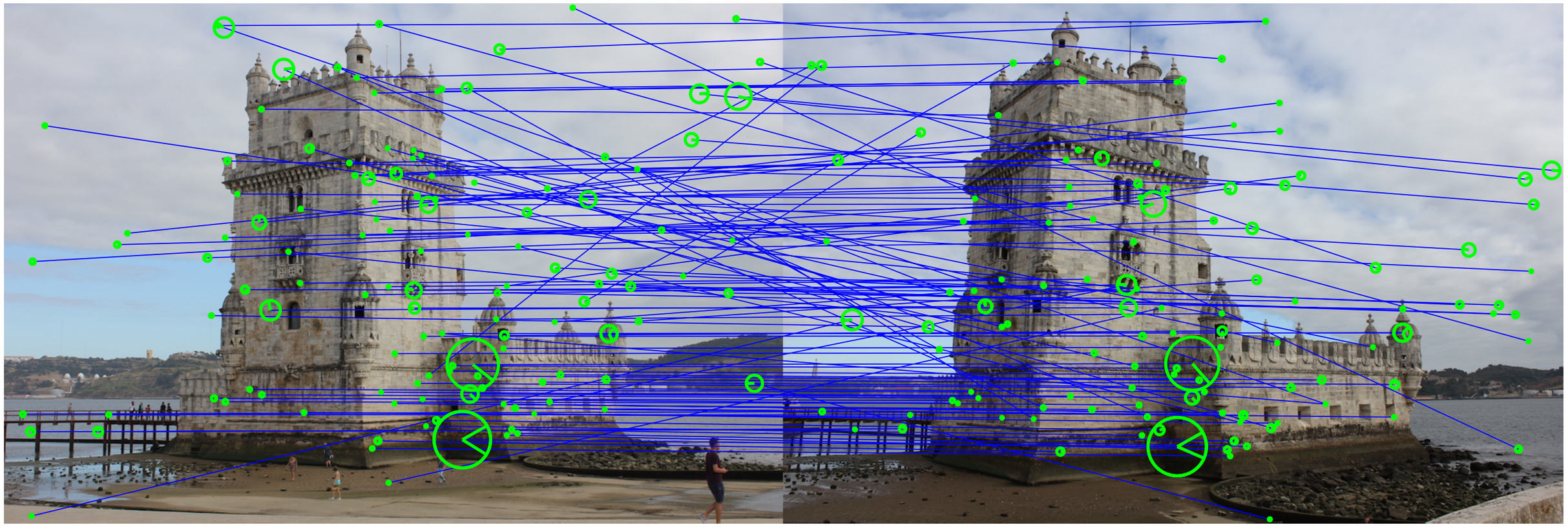


Case 1



Case 2

One more thing…



We have some wrong matches!

RANSAC

- Random sample consensus (RANSAC) is an iterative method for estimating the parameters of a model in a robust way.
- The main idea is to get a subset of the set of samples and to estimate the model with this subset:
    - We estimate the model using the best subset of samples!

RANSAC

- **Input**: a set of n samples S, and a model π.
- **Output**: parameters, P, for the model π., and the subset of S that was used to compute P.

RANSAC

- $e = +\infty$ and $S_b = \emptyset$
- For each iteration:
  - $S_i \subset S$ where S_i is random.
  - Estimate P_i for π using S_i
  - Compute the error e_i for P_i
  - if $e_i < e$ then
    - $e = e_i$ and $S_b = S_i$

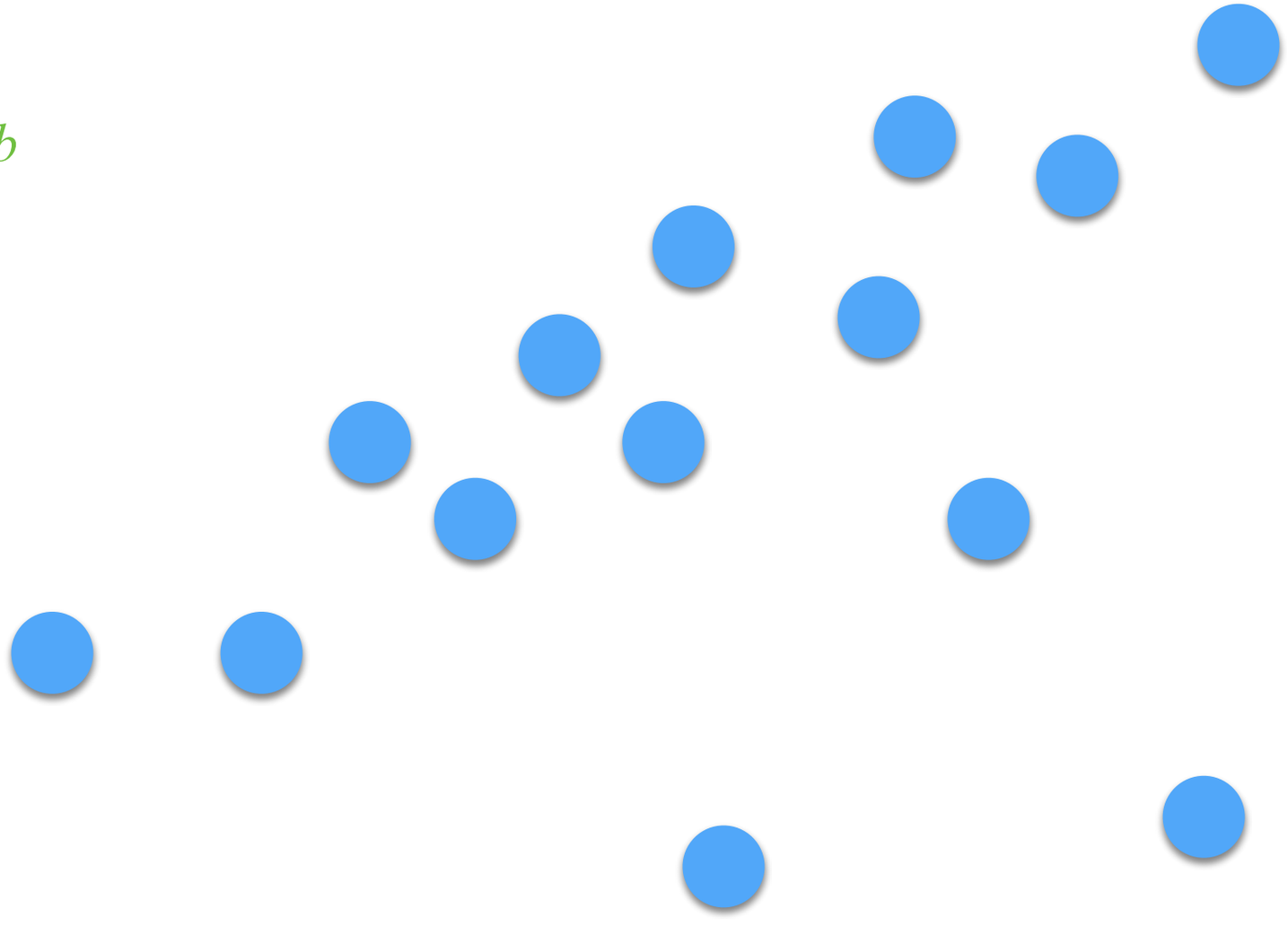
RANSAC

- $e = +\infty$ and $S_b = \emptyset$
- For each iteration:
  - $S_i \subset S$ where S_i is random. **Note that S never changes!**
  - Estimate P_i for π using S_i
  - Compute the error e_i for P_i
  - if $e_i < e$ then
    - $e = e_i$ and $S_b = S_i$

RANSAC: Example

π: a straight line

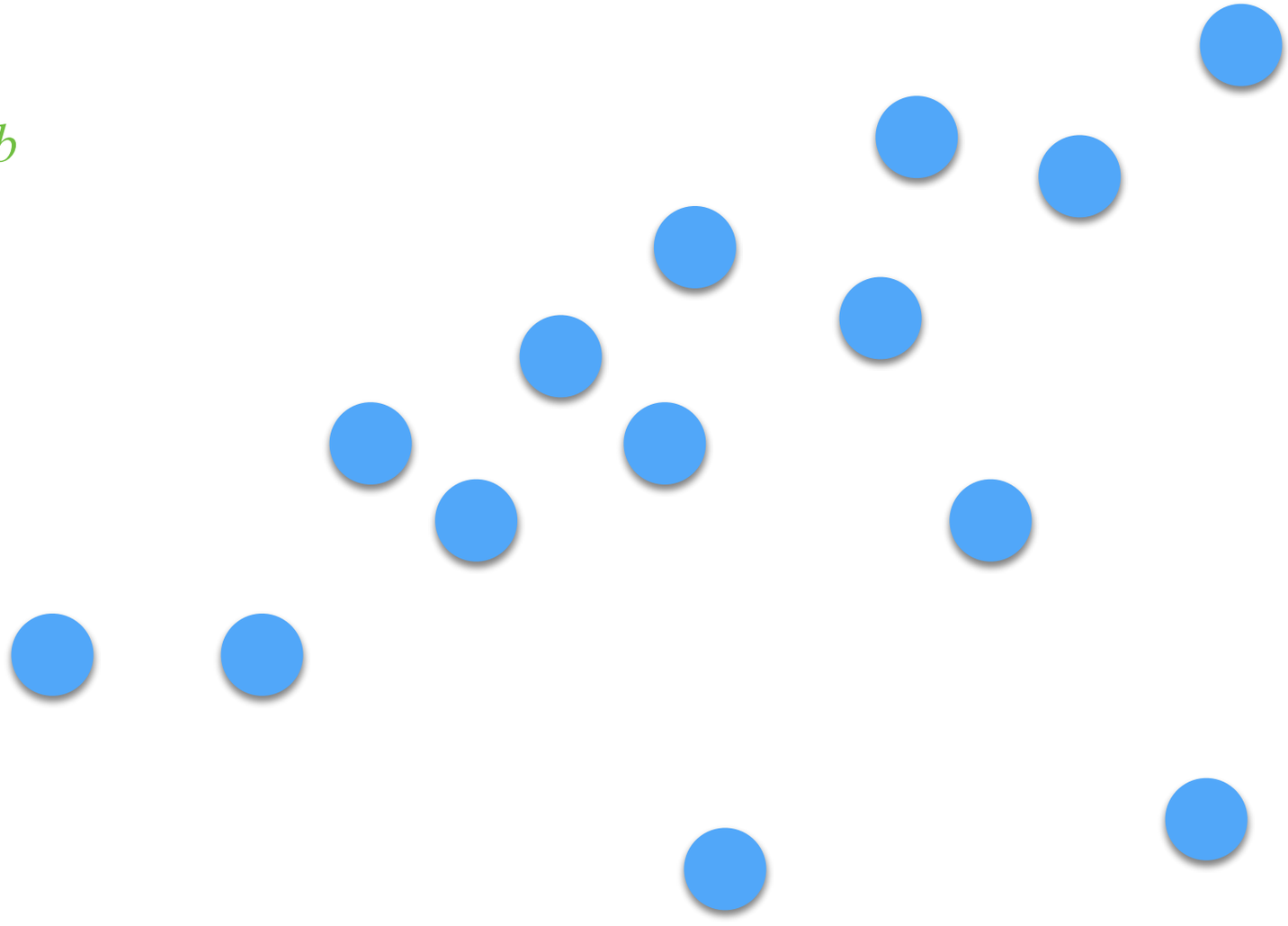
S_i and S_b



RANSAC: Example

π: a straight line

S_i and S_b



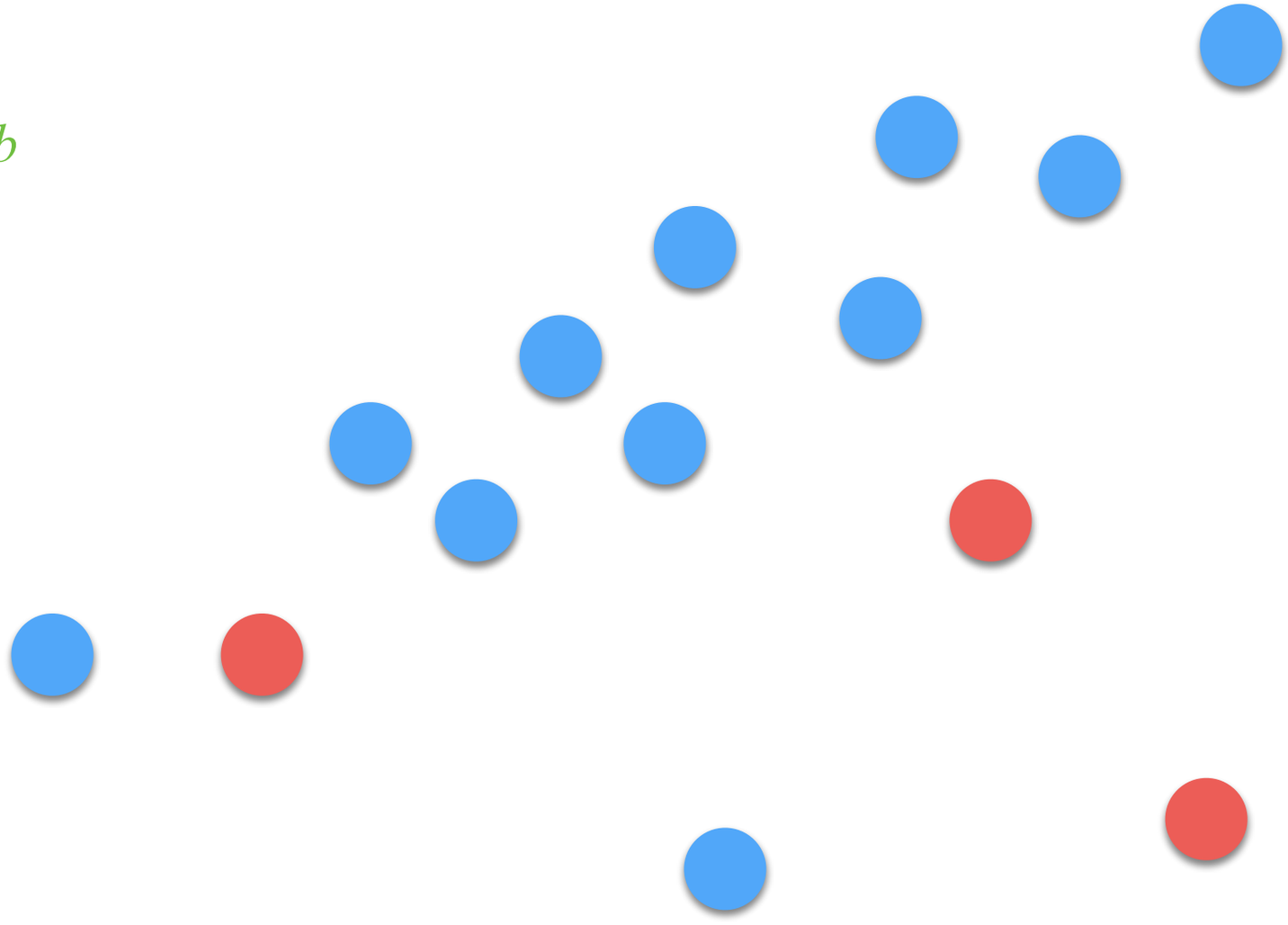
Iteration 0

$e = +\infty$

RANSAC: Example

π: a straight line

S_i and S_b



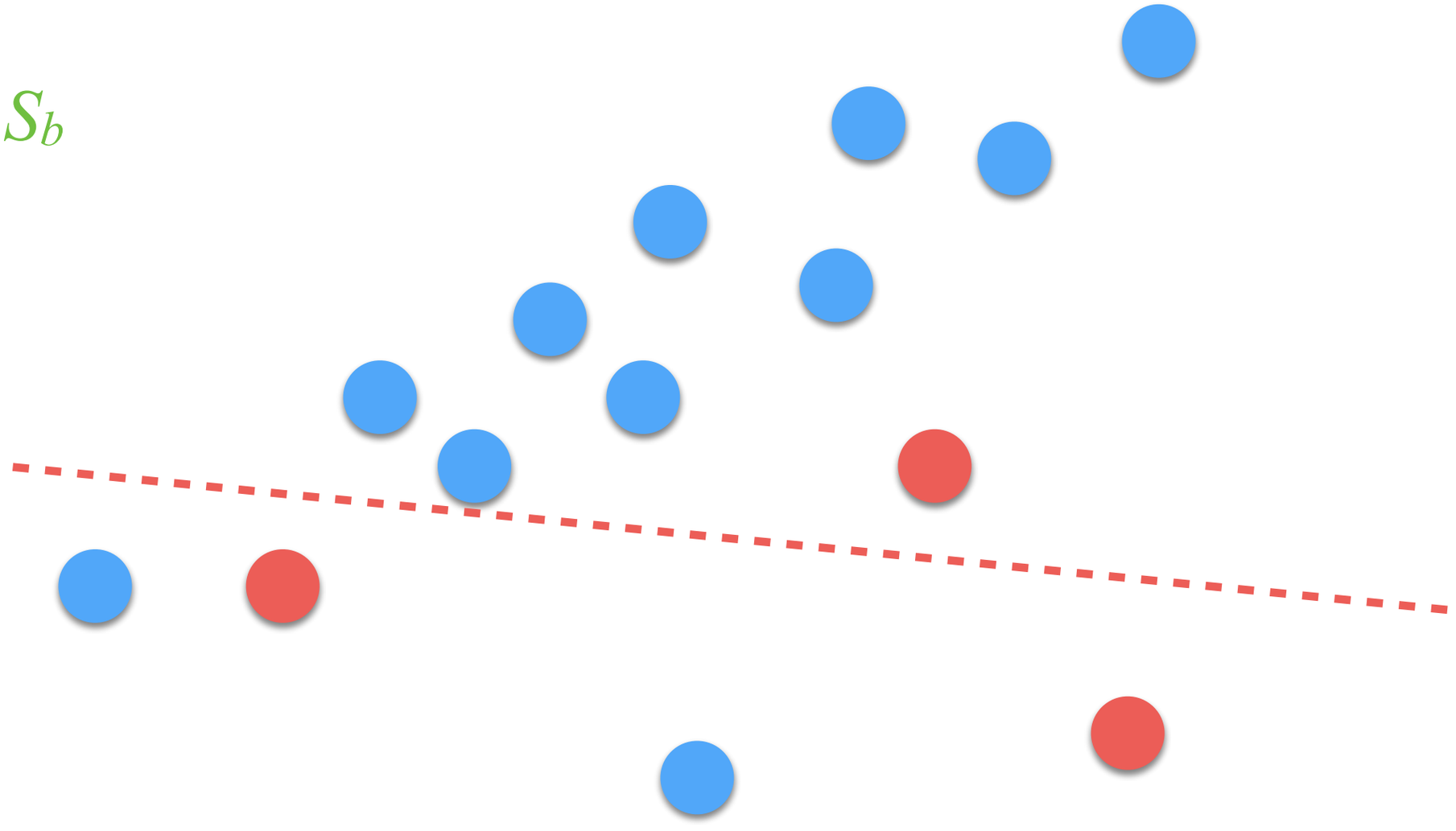
Iteration 1

$e = +\infty$

RANSAC: Example

π: a straight line

S_i and S_b



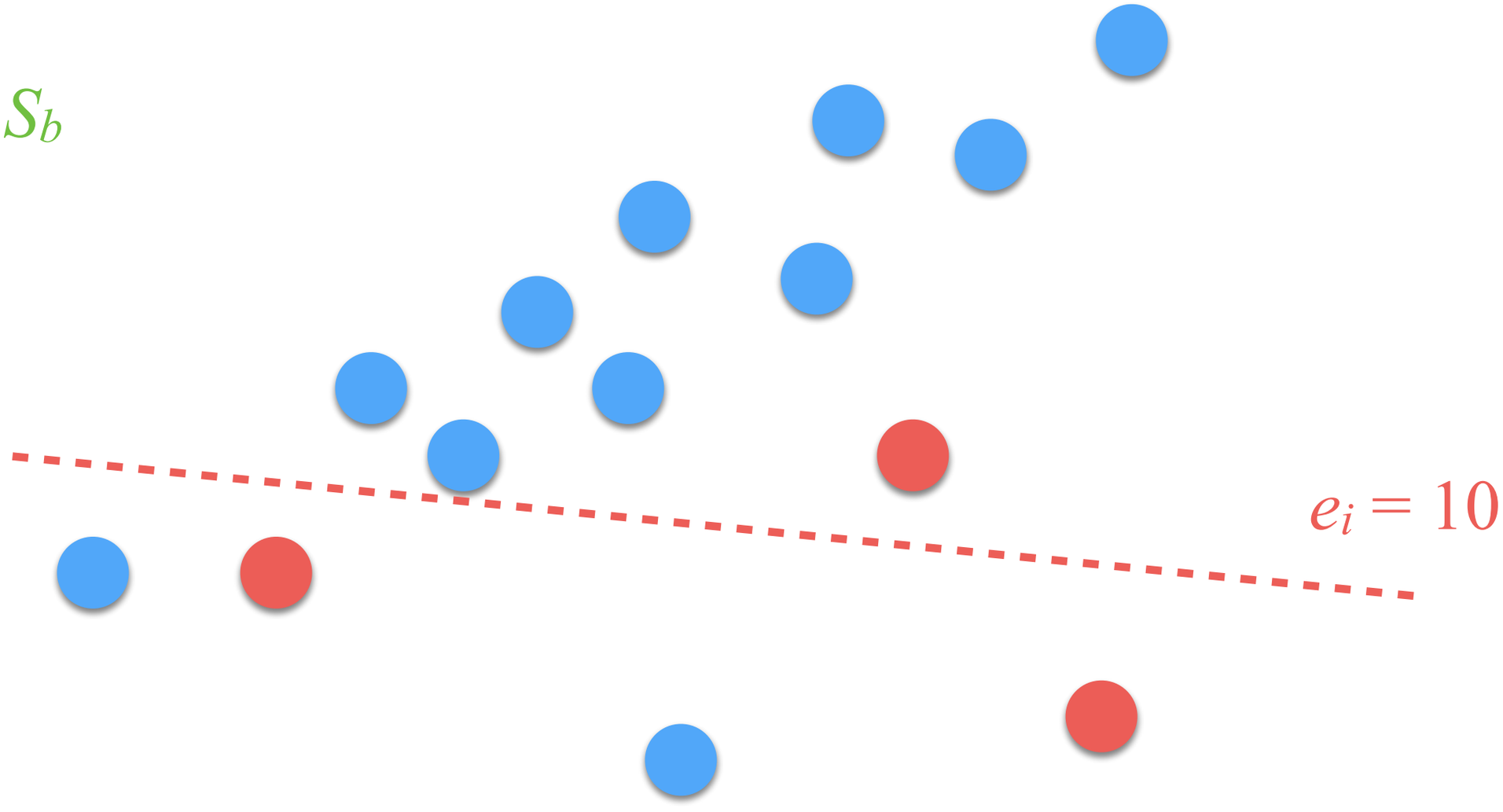
Iteration 1

$e = +\infty$

RANSAC: Example

π: a straight line

S_i and S_b



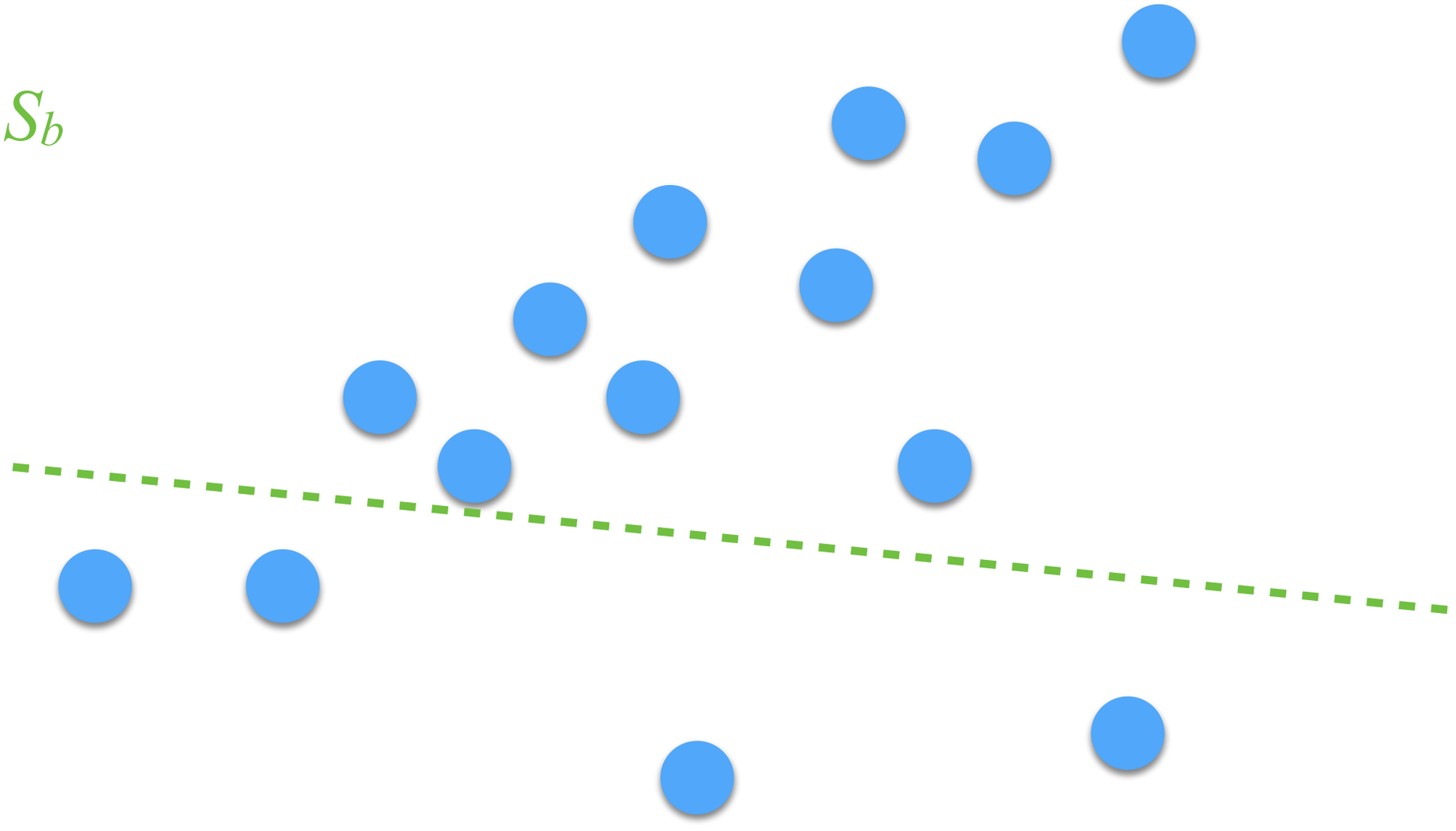
Iteration 1

$e = +\infty$

RANSAC: Example

π: a straight line

S_i and S_b



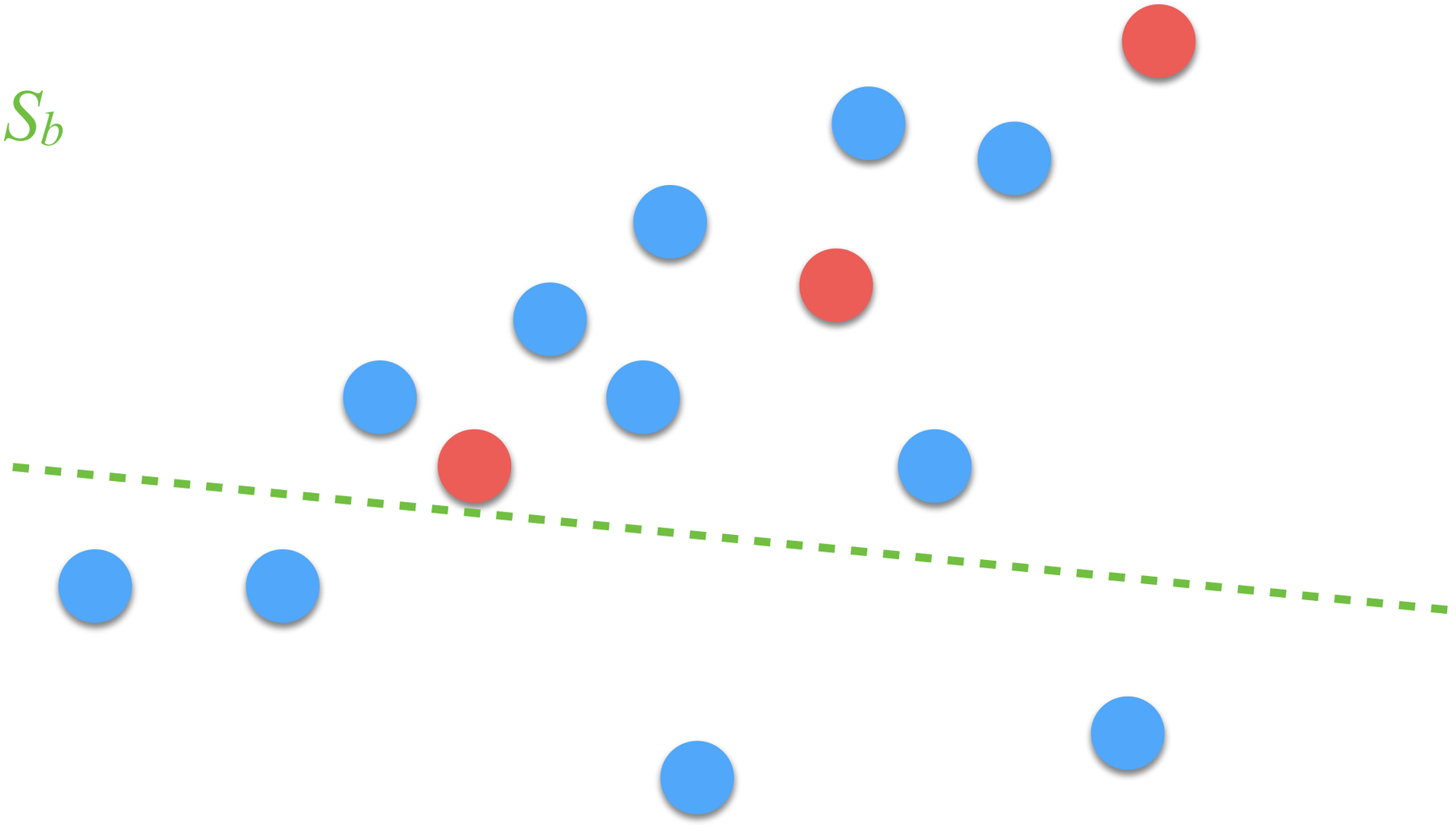
Iteration 2

$e = 10$

RANSAC: Example

π: a straight line

S_i and S_b



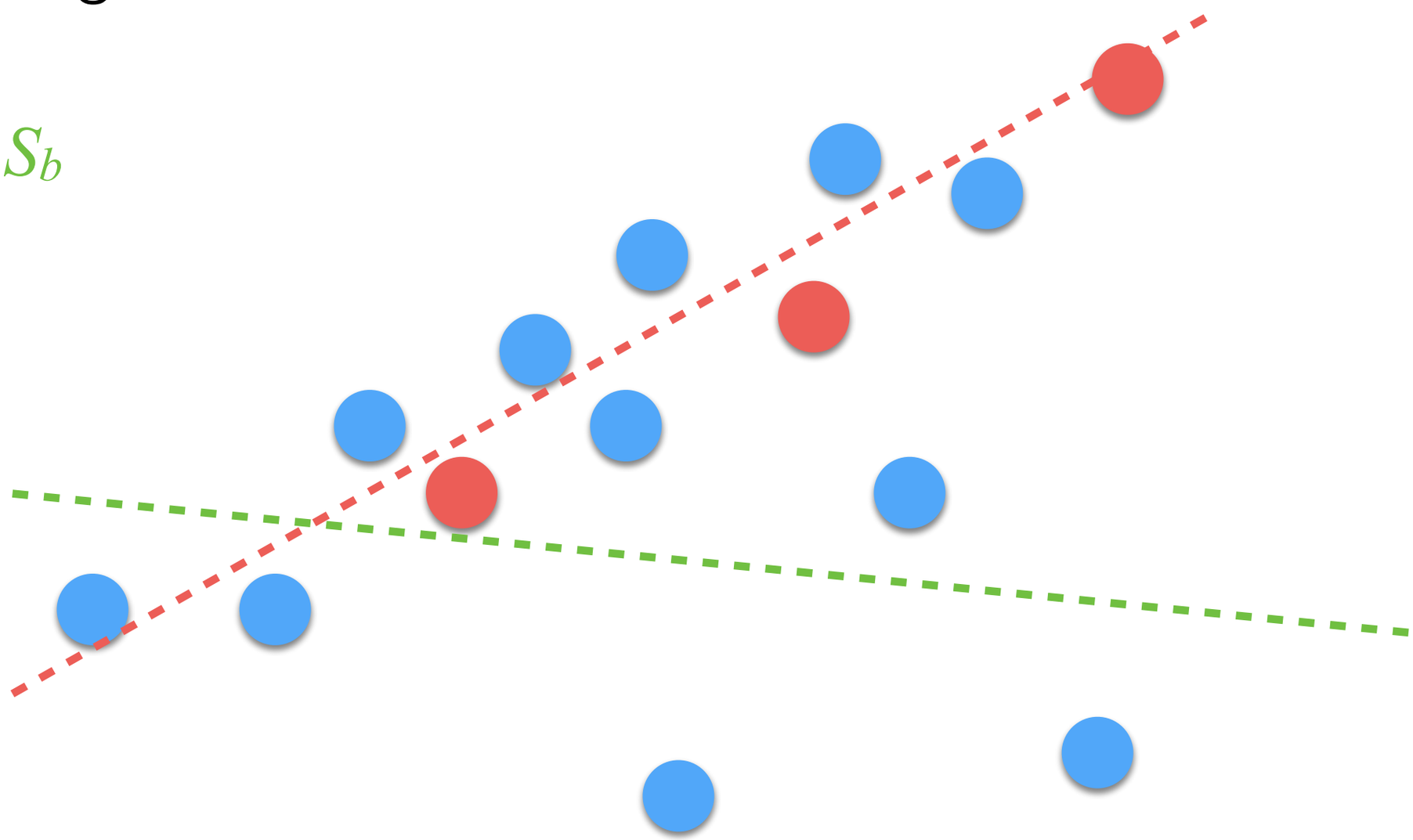
Iteration 2

$e = 10$

RANSAC: Example

π: a straight line

S_i and S_b



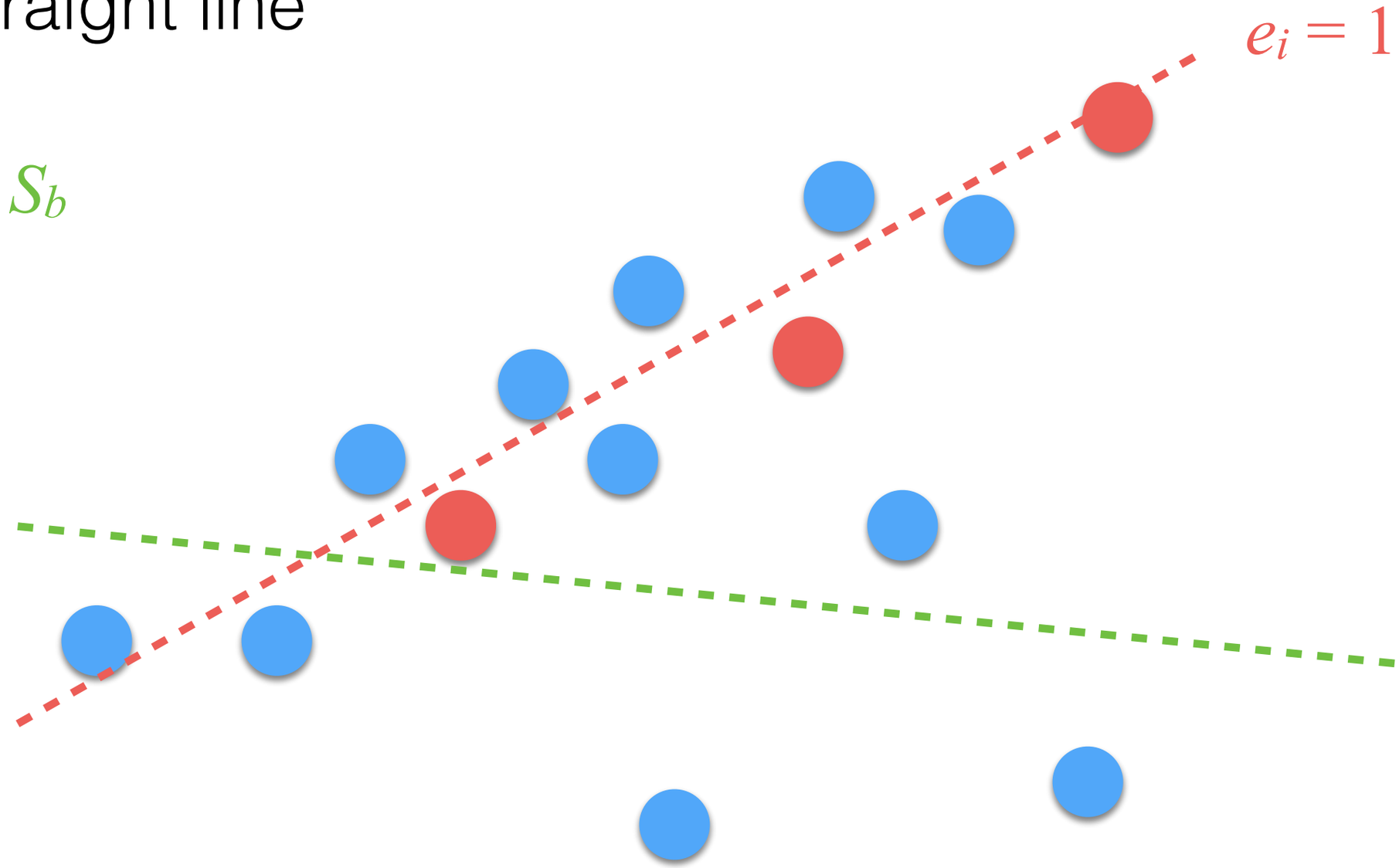
Iteration 2

$e = 10$

RANSAC: Example

π: a straight line

S_i and S_b



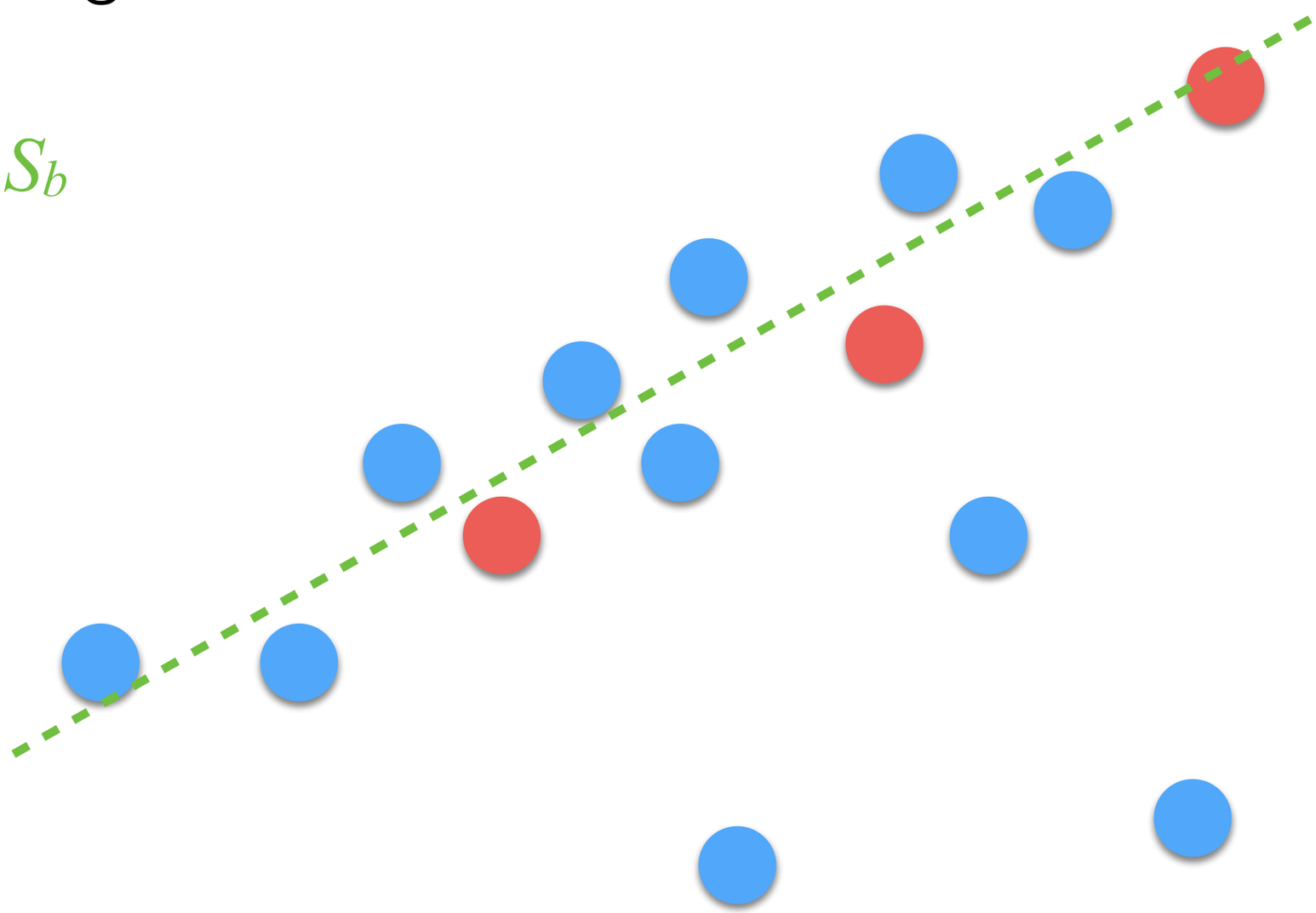
Iteration 2

$e = 10$

RANSAC: Example

π: a straight line

S_i and S_b



Iteration 2

$e = 1$

and we continue for n iterations…

how many?

RANSAC: Iterations

- n has to be large; i.e., we need to have at least one subset containing only inliers S_{inliers}:

$$P(|S_i| = c) = 1 - \left(1 - \left(1 - \frac{|S_{\text{outliers}}|}{|S|}\right)^c\right)^n$$

$$S_i \subseteq S_{\text{inliers}}$$

- We are interested for $P = 1$.

RANSAC

- When do we need to use it?
  - Estimation of the fundamental/essential matrix.
  - Estimation of a homography in the general case.
- When we do not need it for:
  - DLT and Zhang's algorithm:
    - corners are extracted in an accurate way using a calibration pattern!

RANSAC: Fundamental Matrix Estimation

- The algorithm is modified a bit:
    - We count the inliers of each set given a threshold:
        - t_{err} takes into account this constraint:

$$\mathbf{m}_2^\top \cdot F \cdot \mathbf{m}_1 = 0$$

$$\mathbf{m}_1^\top \cdot F^\top \cdot \mathbf{m}_2 = 0$$

- If we have a set with more inliers of the previous one it is accepted.

- We compute the F using only the inliers!

that’s all folks!