3D from Photographs: Structure From Motion

Dr Francesco Banterle
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

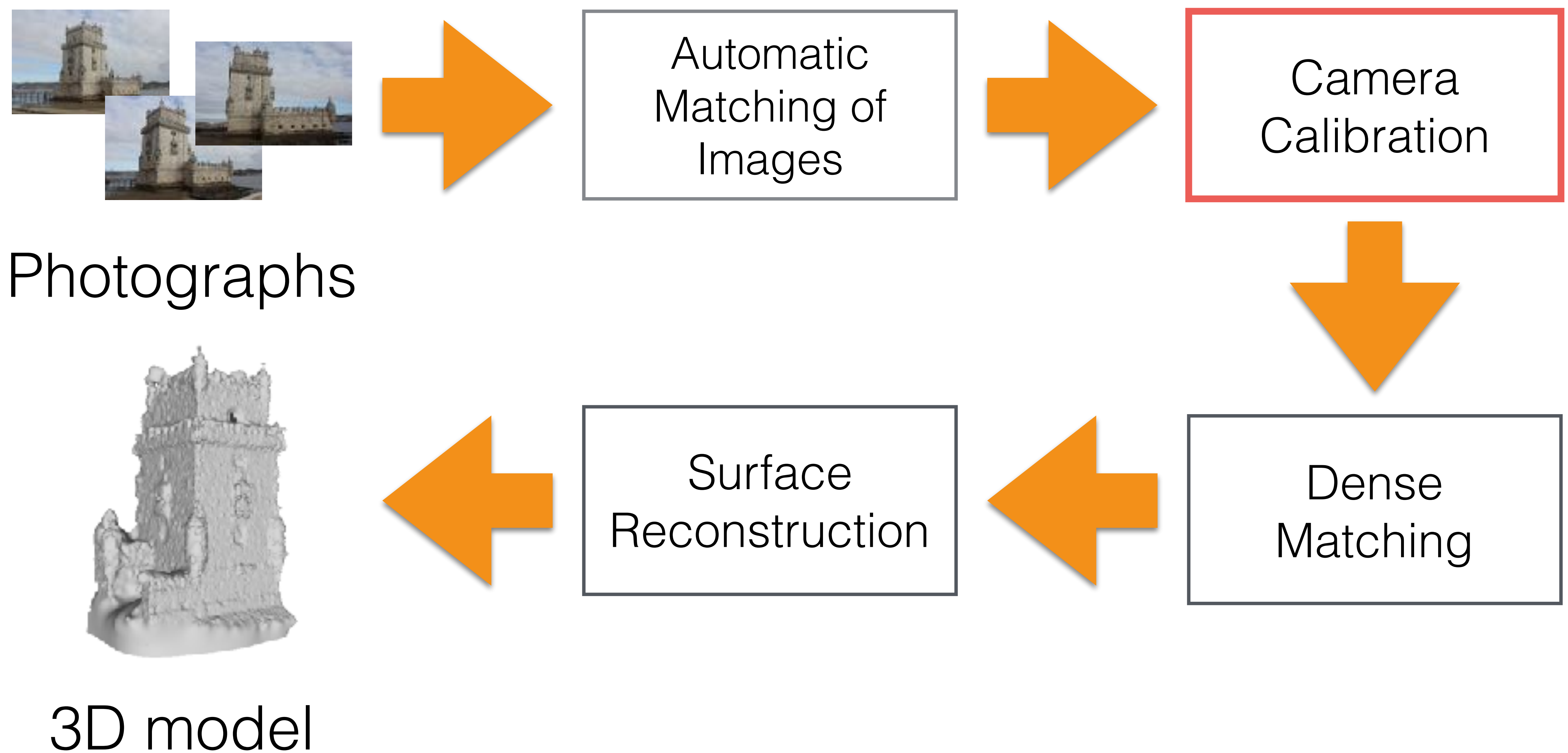
Camera Calibration

Dense Matching

Surface Reconstruction

3D model

3D from Photographs



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

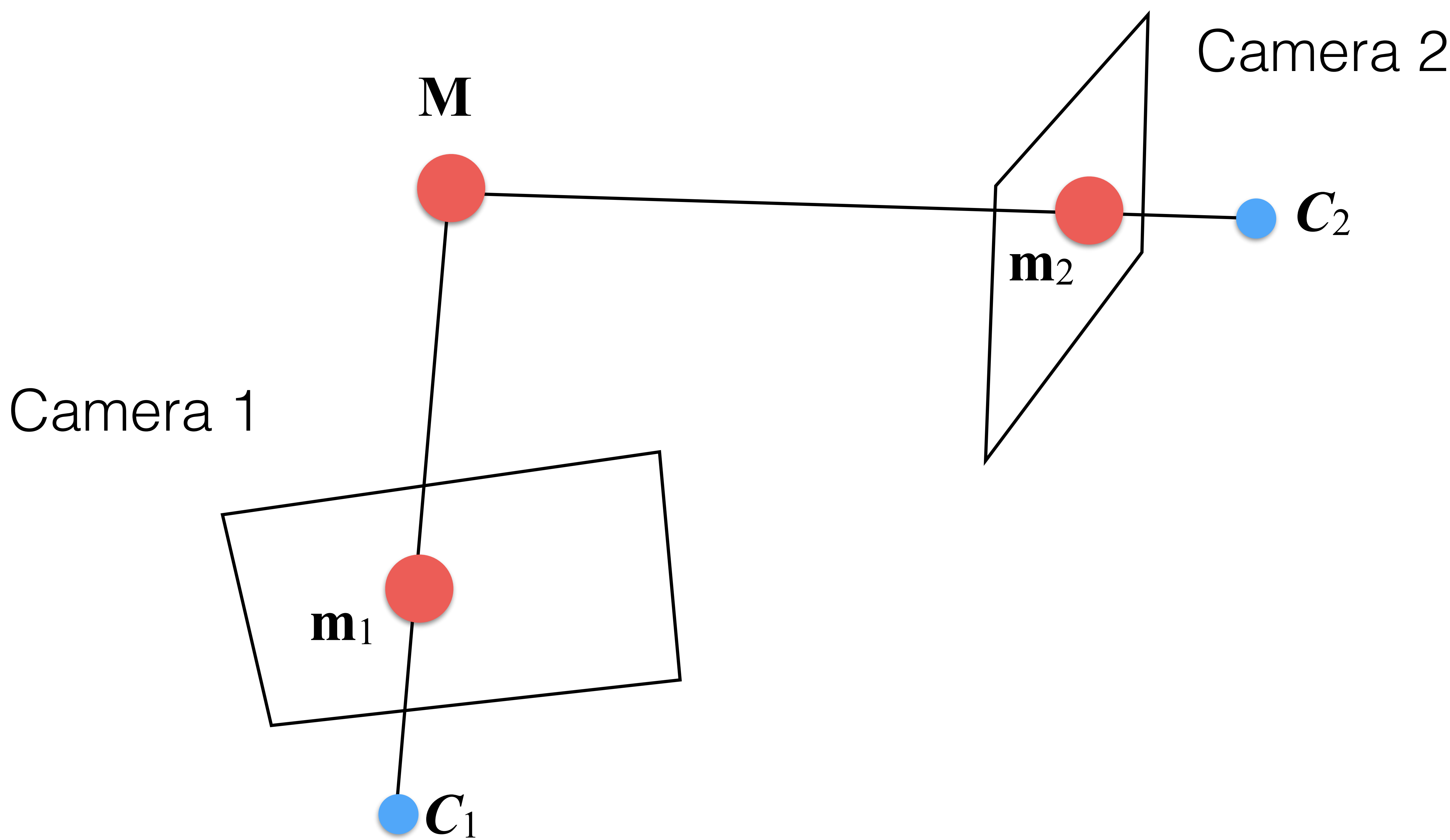
How do we get the camera's pose without the pattern?

Camera Pose Calibration

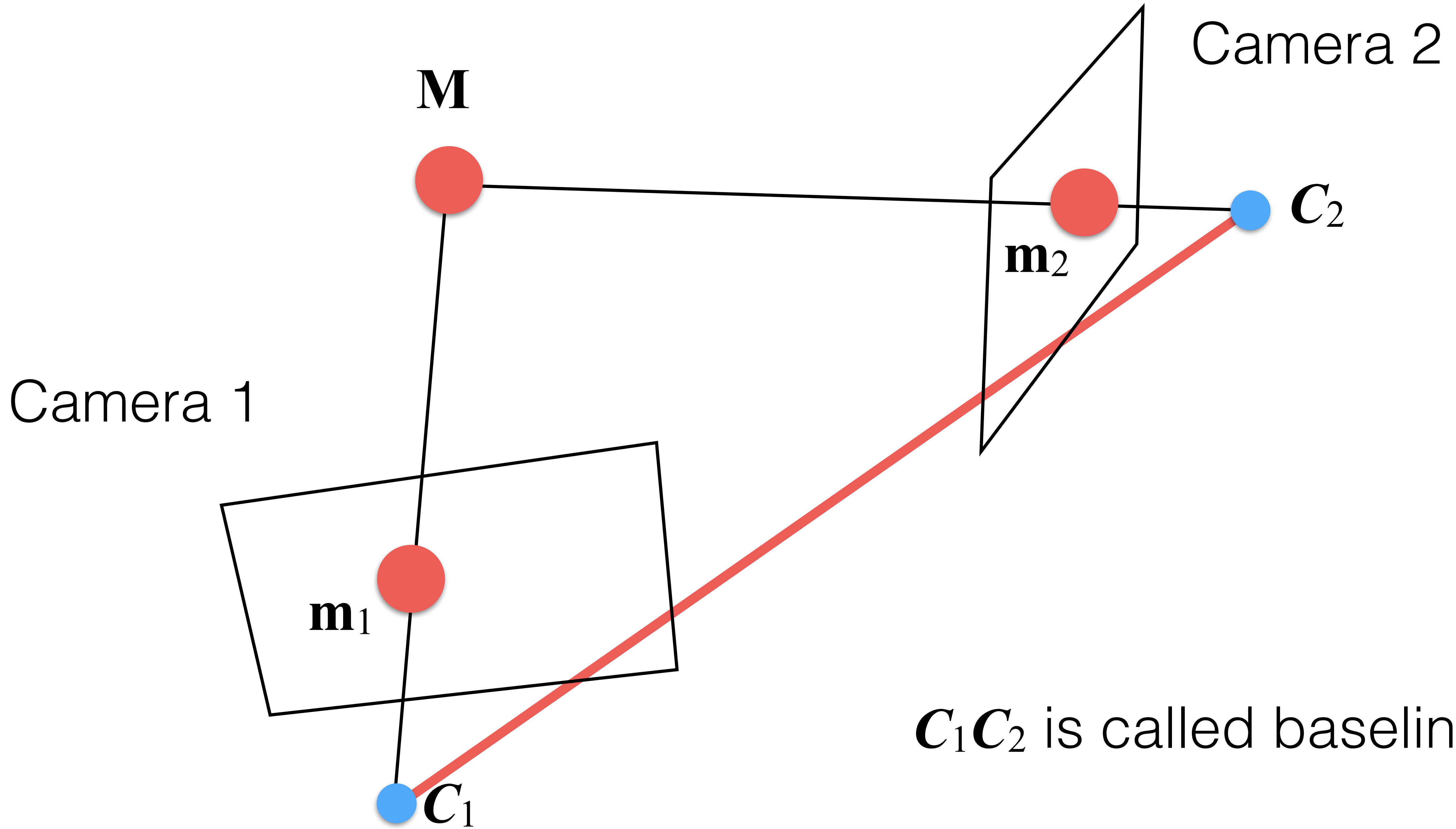
- Let’s assume that:
  - We have K for each photograph.
  - We have matches between images.

A Two-Camera Example

A Two-Camera Example

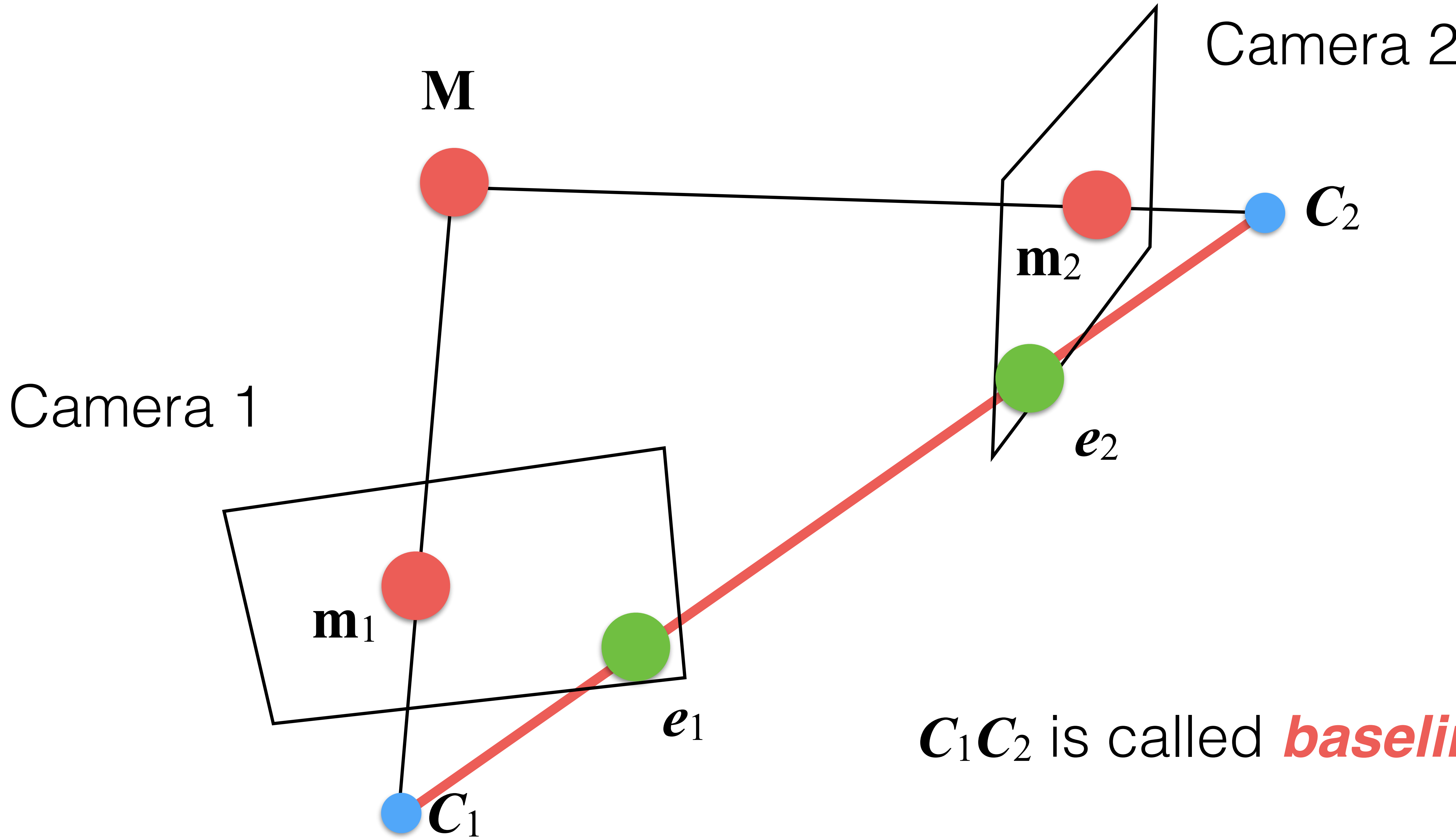


A Two-Camera Example: Epipolar Geometry



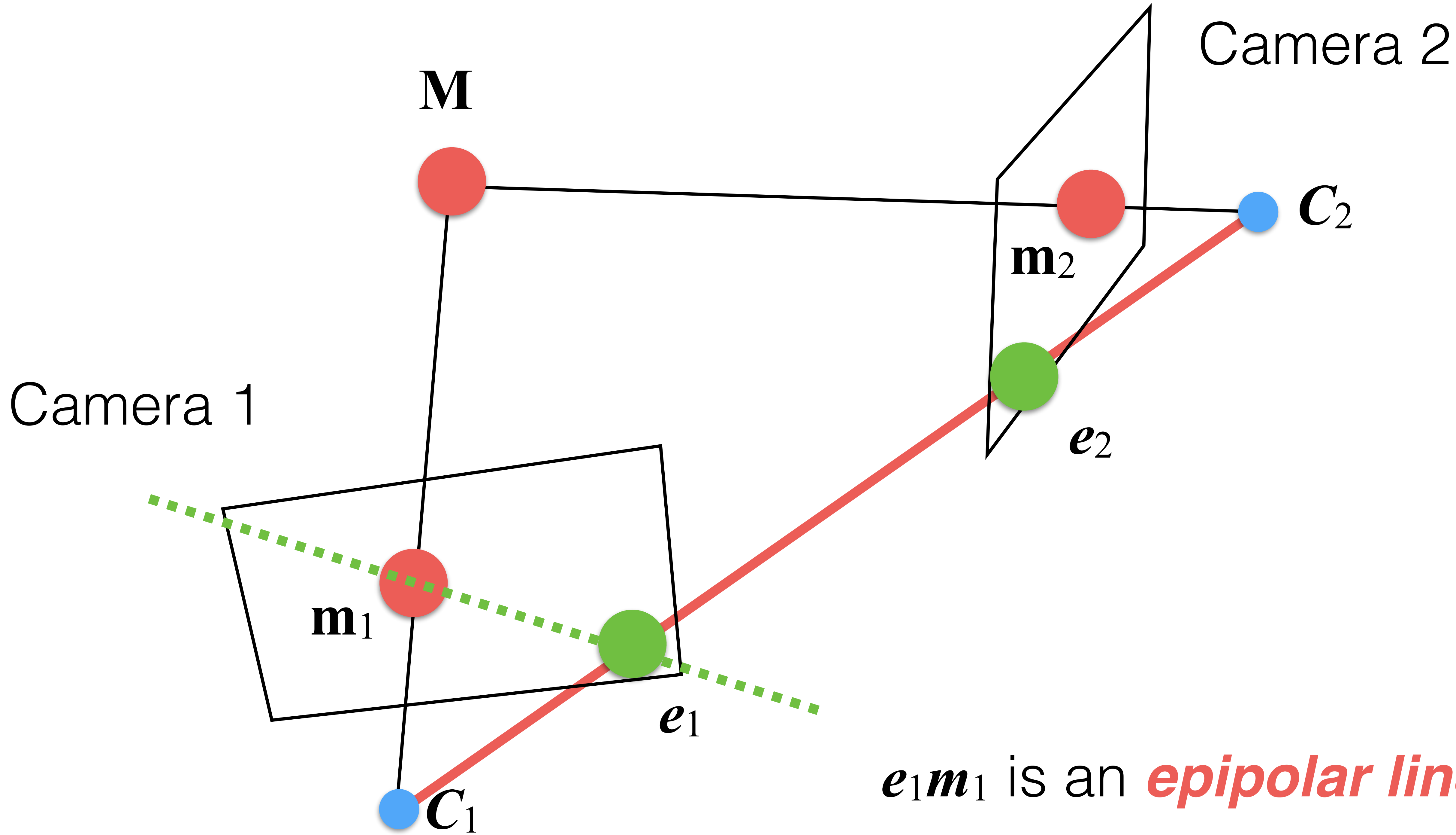
C_1C_2 is called baseline

A Two-Camera Example: Epipolar Geometry



C_1C_2 is called ***baseline***

A Two-Camera Example: Epipolar Geometry



$e_1 m_1$ is an ***epipolar line***

A Two-Camera Example: Epipolar Geometry

- The epipolar line is defined as

$$\mathbf{m}_1(t) \simeq (Q_1 \cdot Q_2^{-1}) \cdot t + \mathbf{e}_1$$

$$P_1[Q_1|\mathbf{q}_1] \quad P_2[Q_2|\mathbf{q}_2]$$

- where an epipole $\boldsymbol{e}_i$ is defined as

$$\mathbf{e}_1 \simeq P_1 \cdot \mathbf{C}_2$$

$$\mathbf{e}_2 \simeq P_2 \cdot \mathbf{C}_1$$

A Two-Camera Example

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

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

Note that we need to estimate both 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:

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

- where:

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

- and:

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

The Essential Matrix

- E is called the ***essential matrix***, and it is a 3×3 matrix.

- If we have the K matrices and apply the Longuet-Higgins equation we obtain:

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

- F is called the ***fundamental matrix***:

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

The Essential Matrix: 8-points algorithm

- From:

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

- We can define a linear system as $A \cdot \mathbf{b} = \mathbf{0}$

$$A = \begin{bmatrix} (\hat{\mathbf{m}}_1^1)^\top \otimes (\hat{\mathbf{m}}_2^1)^\top \\ \vdots \\ (\hat{\mathbf{m}}_1^n)^\top \otimes (\hat{\mathbf{m}}_2^n)^\top \end{bmatrix} \qquad \mathbf{b} = \mathrm{vec}(E)$$

- Given enough matches 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.

The Essential Matrix

- 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 do not estimate E directly, but F. Then, we compute E from F, K_1, and K_2.

- When estimation F, we use homogenous coordinates for $\boldsymbol{m}_i$, such that $u_i \in [0, w]$ and $v_i \in [0, h]$.

- However, solving the linear system with such values we can get numerical instabilities!

The Essential Matrix: Practice

- For removing numerical instabilities, it would be nice to have values with average distance $\sqrt{2}$ from the origin.

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

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

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

The Essential Matrix: Practice

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

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

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

- We can now solve the linear system!

- 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, and n is the number of matched points.

- Again we can solve it with 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, and n is the number of matched points.

- Again we can solve it with 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, and n is the number of matched points.

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

Now we have E, and so what?

E Factorization

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

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

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

E Factorization

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

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

- 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

$$\mathrm{diag}(1,1,0) = W \cdot Z$$

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

E Factorization

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

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

- R' is still a rotation matrix!

E Factorization

- Given that:

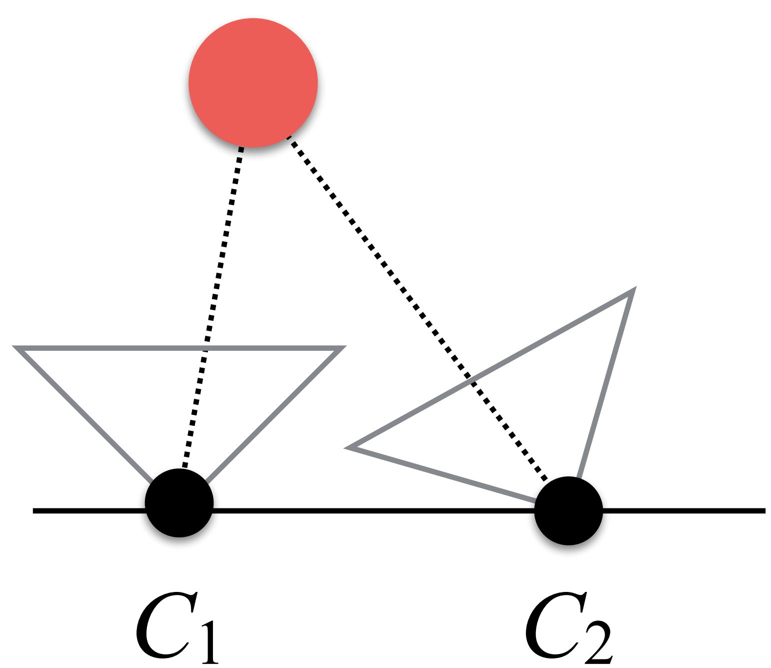
$$\mathrm{SVD}(E) = U \cdot \mathrm{diag}(1,1,0) \cdot V^\top$$

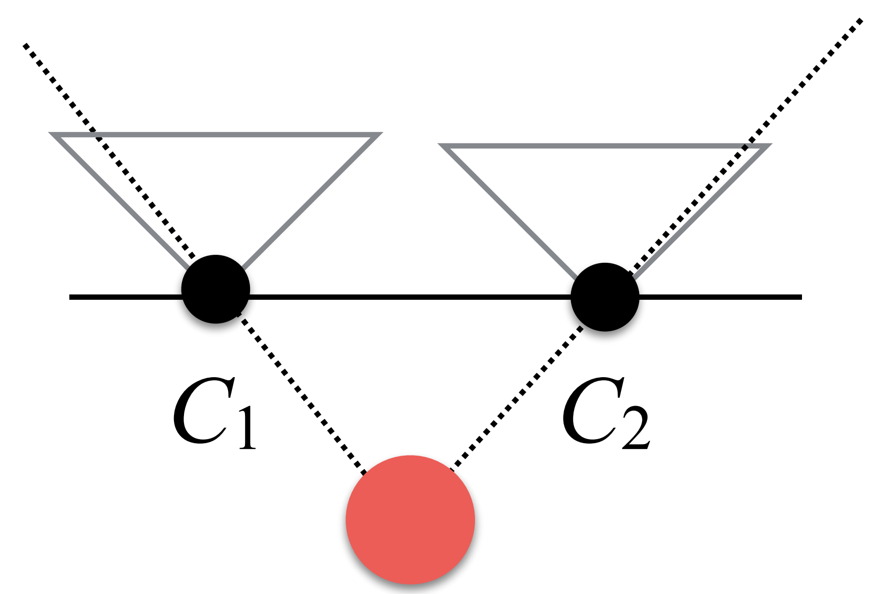
- We can have four possible factorizations of E such that $E = S \cdot R$:

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

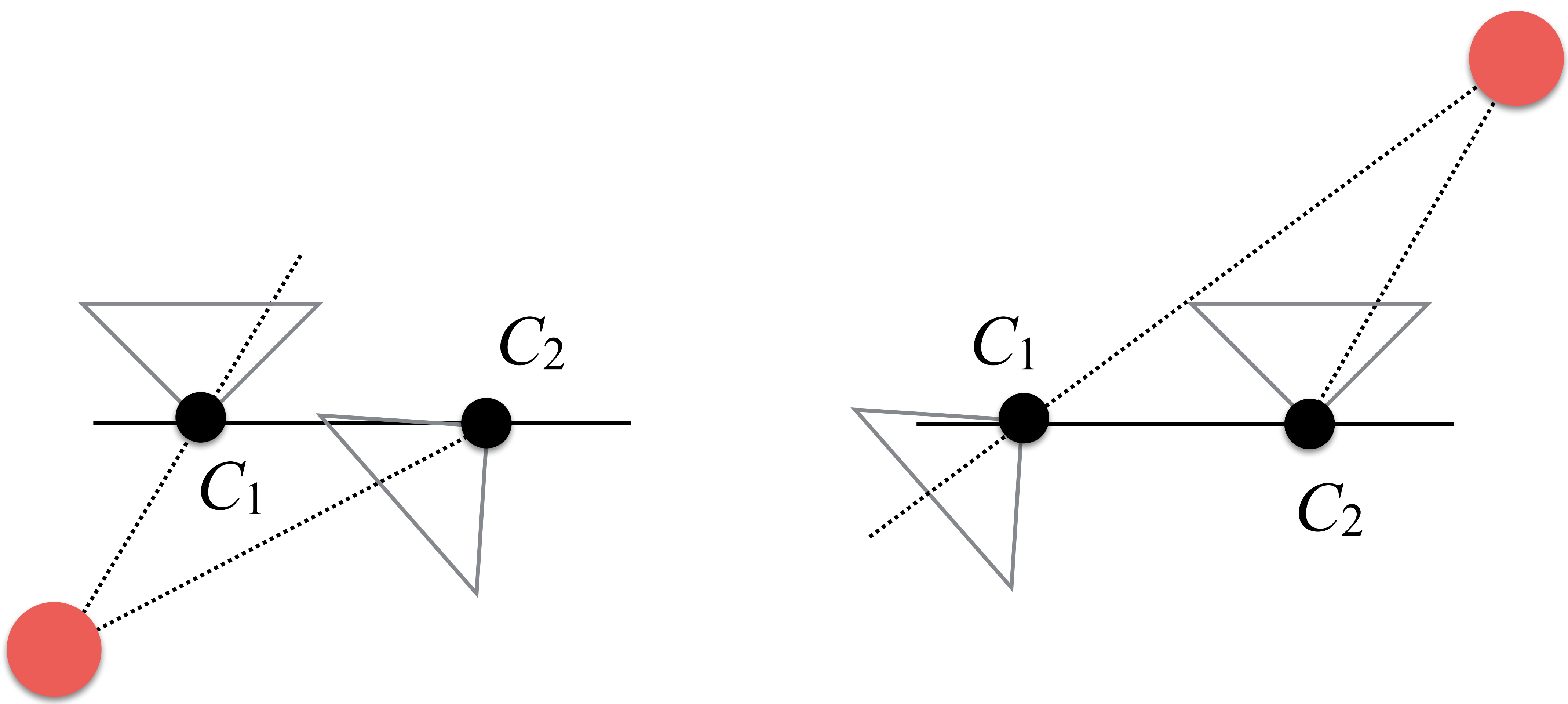
$$R = U \cdot W \cdot V^\top \text{ or } R = U \cdot W^\top \cdot V^\top$$

E Factorization: The Four Cases

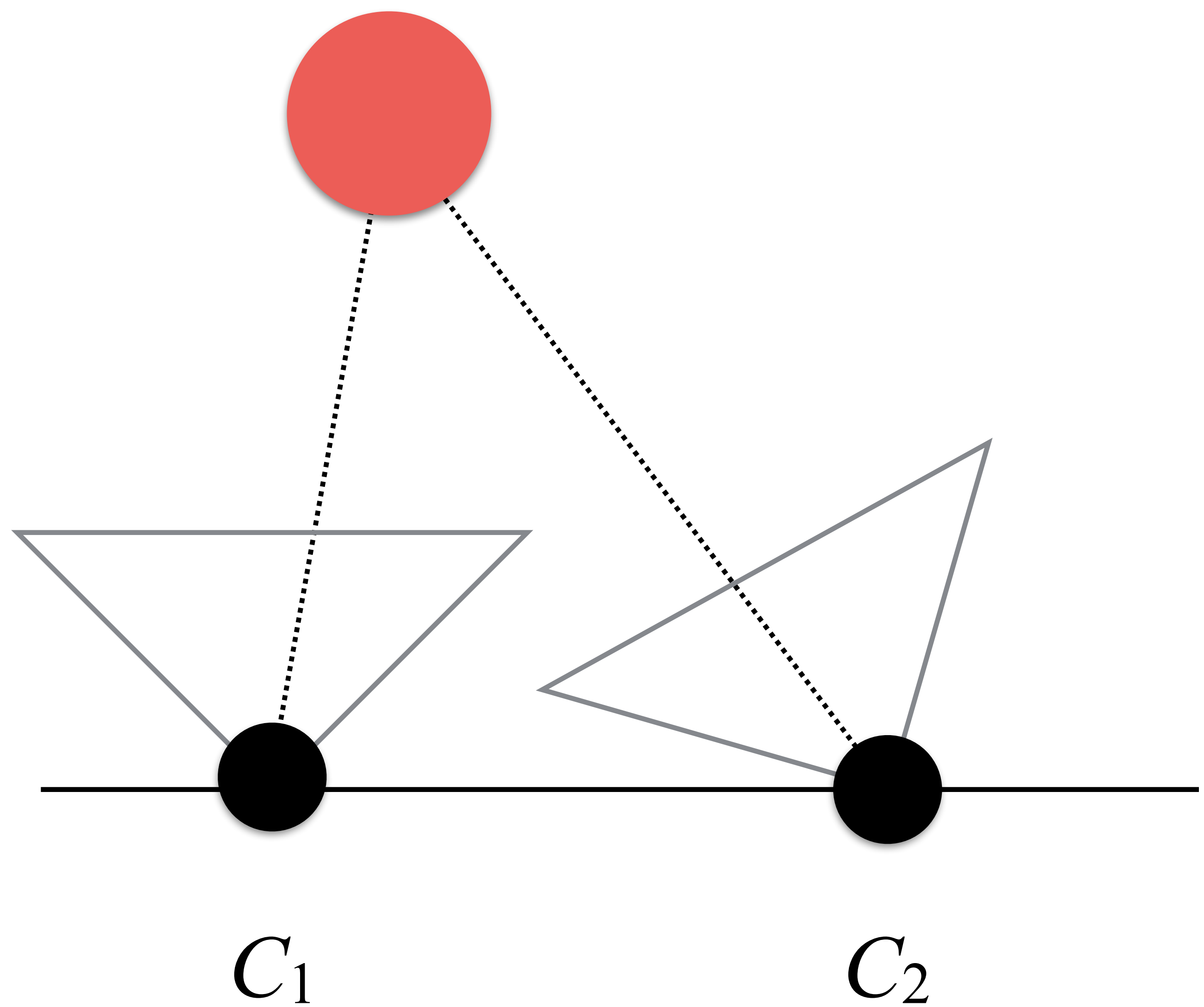




E Factorization: The Four Cases



Which is the correct configuration?


C_1
C_2

Why?

Both points are seen by the cameras!

How do we find it?

We need to find a case in which all 3D points are in the positive frustum of both cameras!

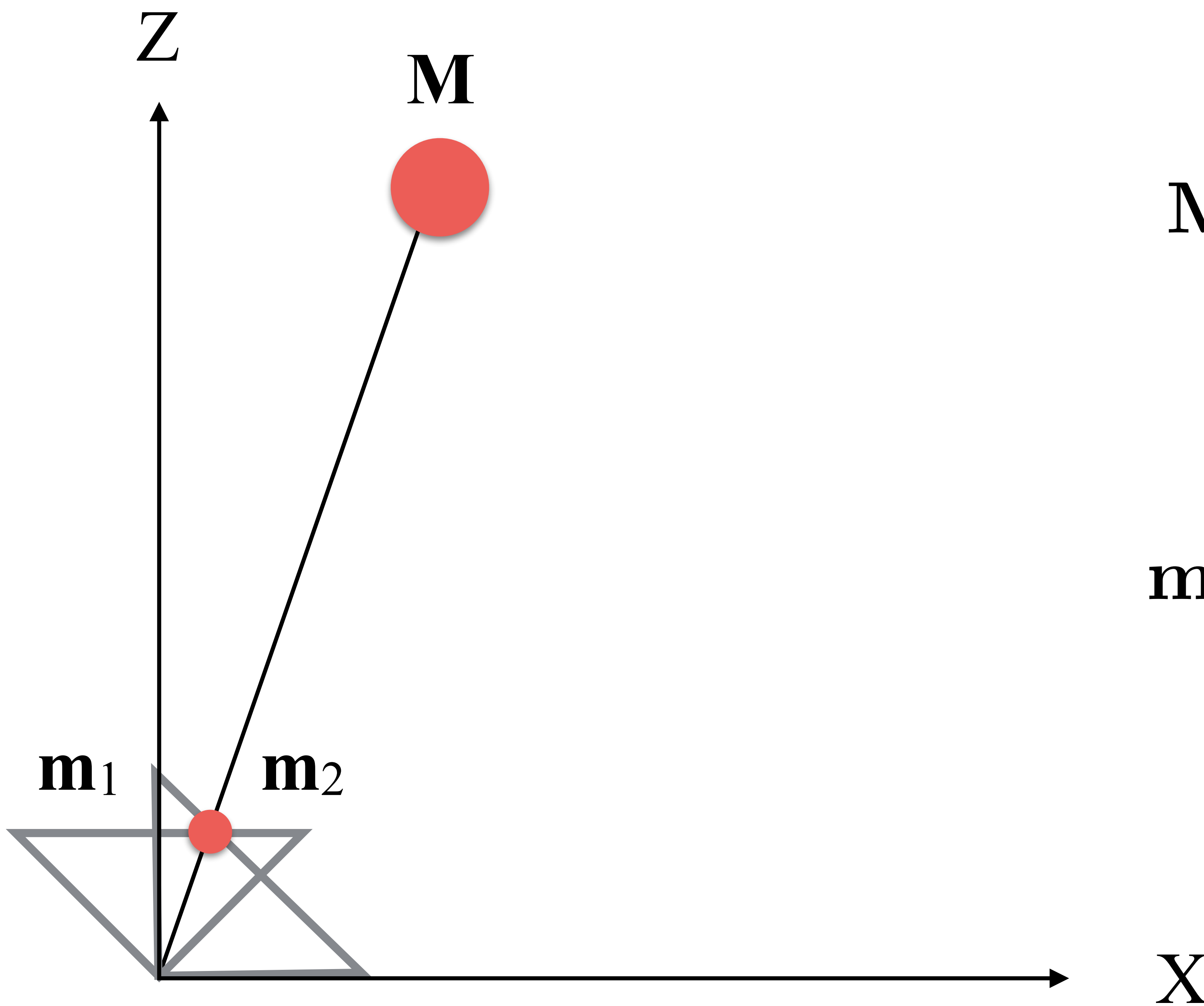
Triangulation

Triangulation

- **Input**: n matched 2D feature points in two images and their P matrices (i.e., we know K, G, and $\boldsymbol{t}$).

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

Triangulation: Pure Rotational Motion Case

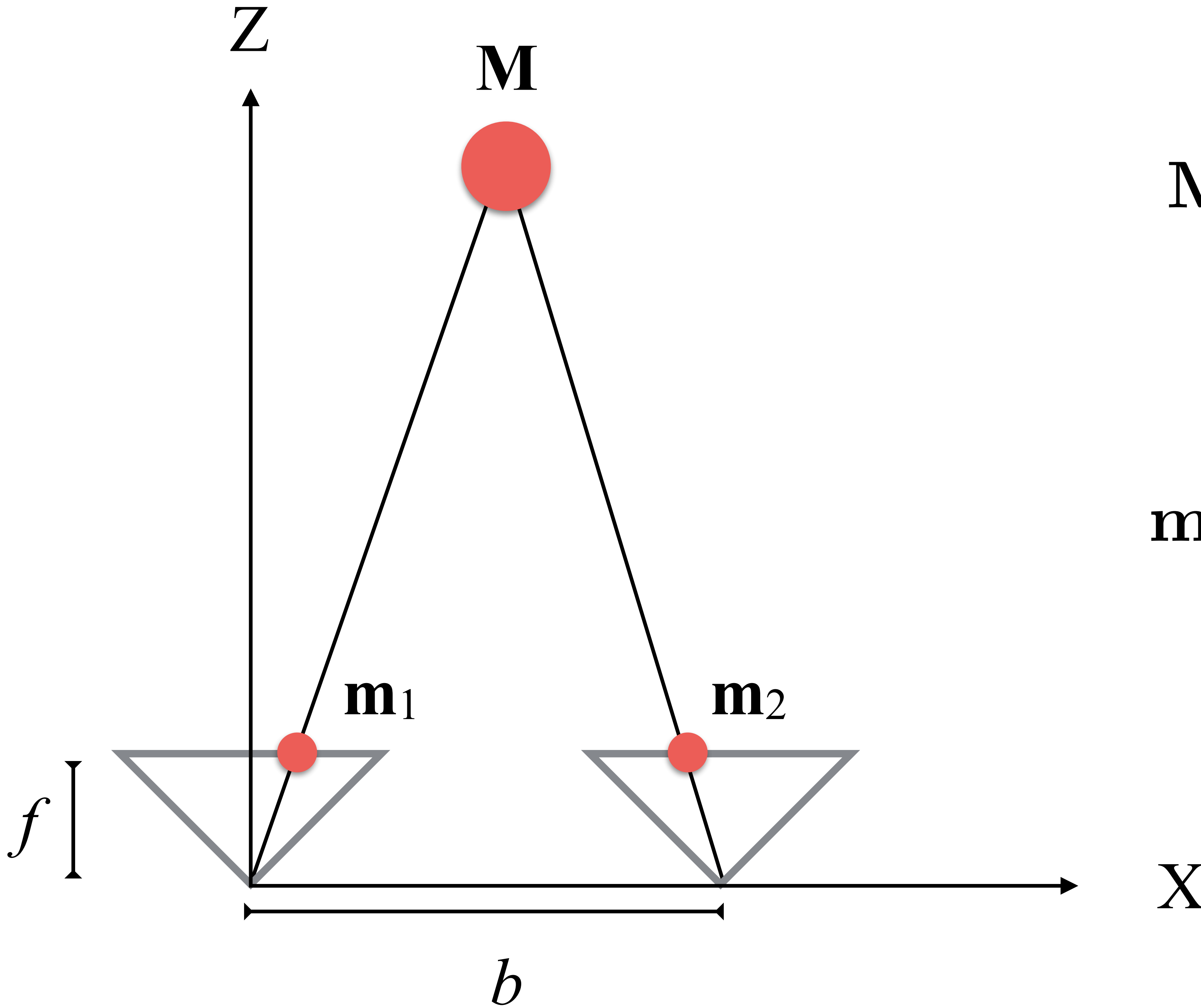


$$\mathbf{M} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

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

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

Triangulation: 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

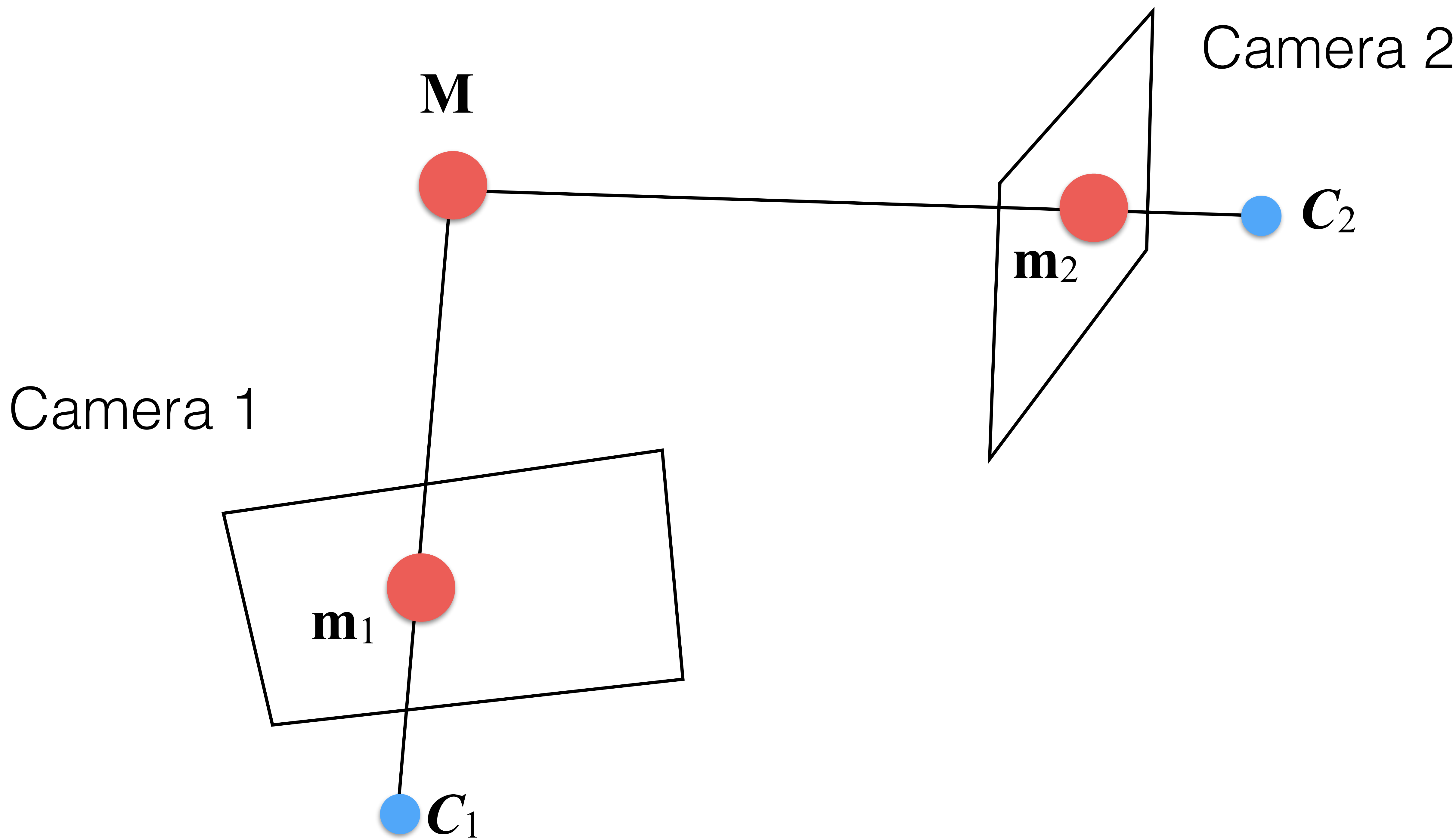
- We first fix the frame of reference to one of the two cameras. Then, we know that:

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

- So, we can obtain:

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

Triangulation:
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:

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

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:

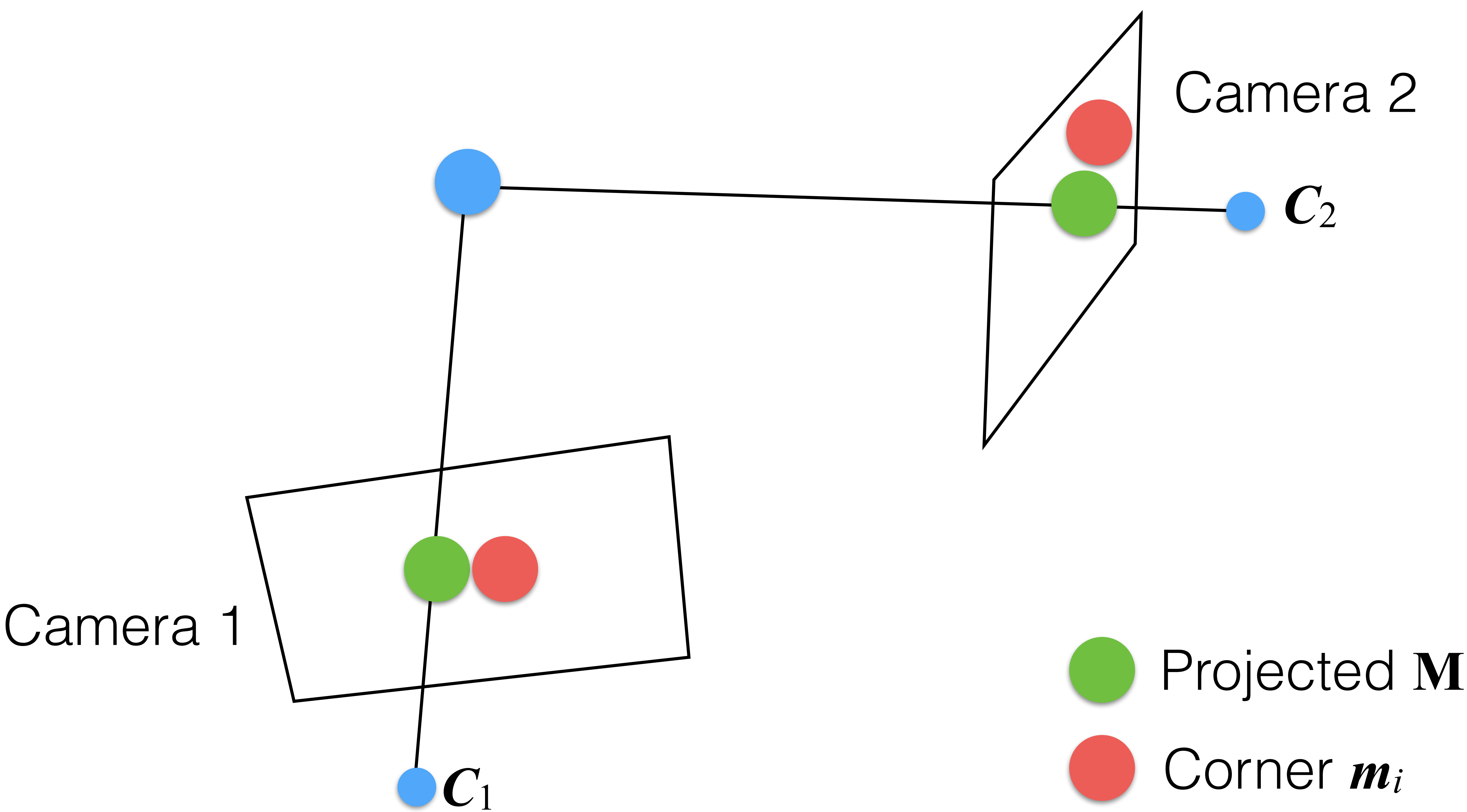
$$\begin{bmatrix} (\mathbf{p}_1^1 - u_1 \cdot \mathbf{p}_3^1)^\top \\ (\mathbf{p}_2^1 - u_1 \cdot \mathbf{p}_3^1)^\top \\ \vdots \\ (\mathbf{p}_1^l - u_1 \cdot \mathbf{p}_3^l)^\top \\ (\mathbf{p}_2^l - u_1 \cdot \mathbf{p}_3^l)^\top \end{bmatrix} \cdot \mathbf{M} = \mathbf{0}$$

Triangulation: Eigen Method

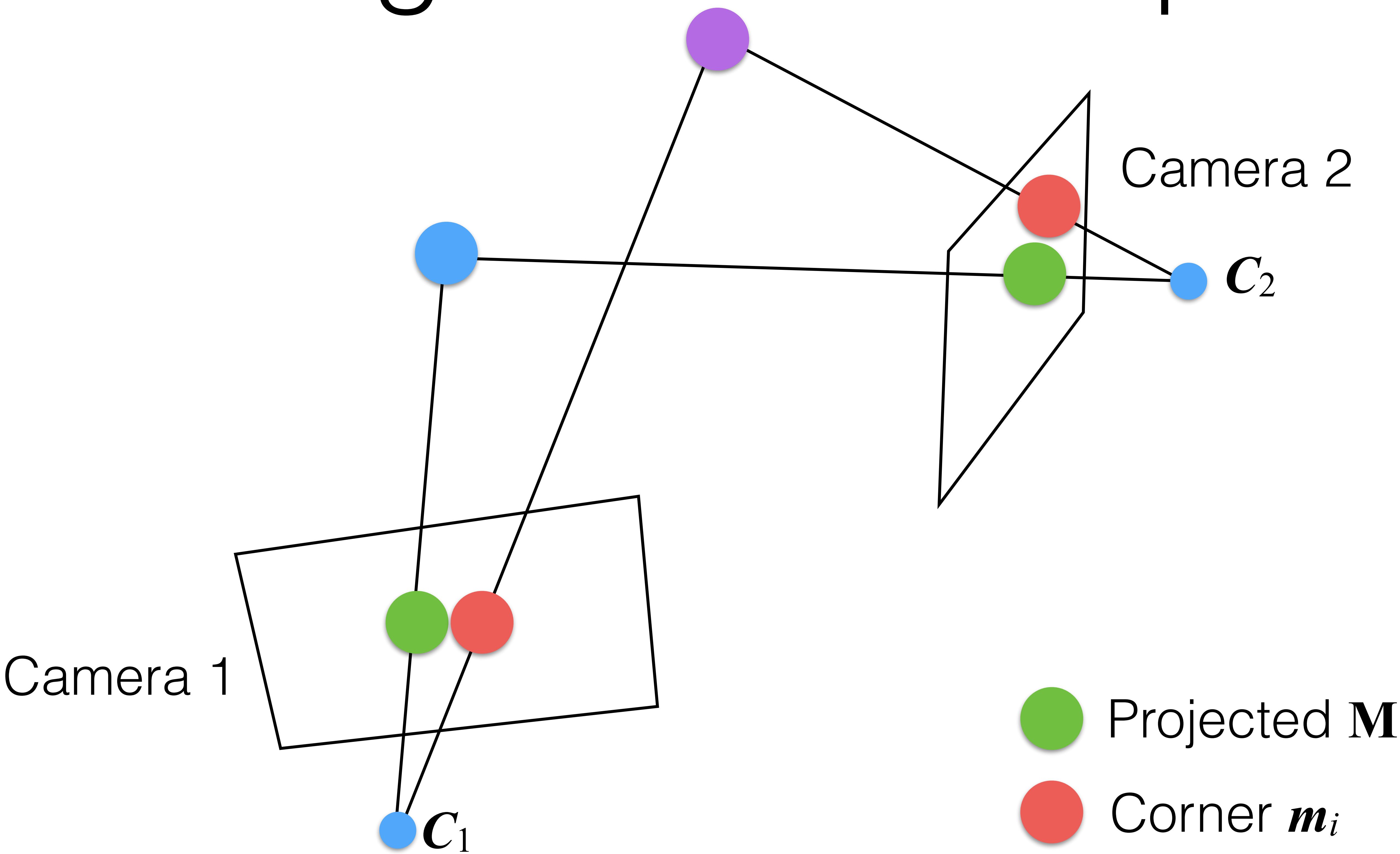
- Again, we solve this linear system using SVD; i.e., the kernel of V.
- Again, we minimized an algebraic error without a geometric meaning!
- Again, we use this initial solution for a non-linear method that minimizes a geometric error:

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

Triangulation: Example



Triangulation: Example



Structure From Motion

Structure From Motion

- **Input**: n matched points (corners computed with Harris algorithm) between two images, and K for all cameras.
- **Output**: n 3D points, and G for the two cameras.

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.

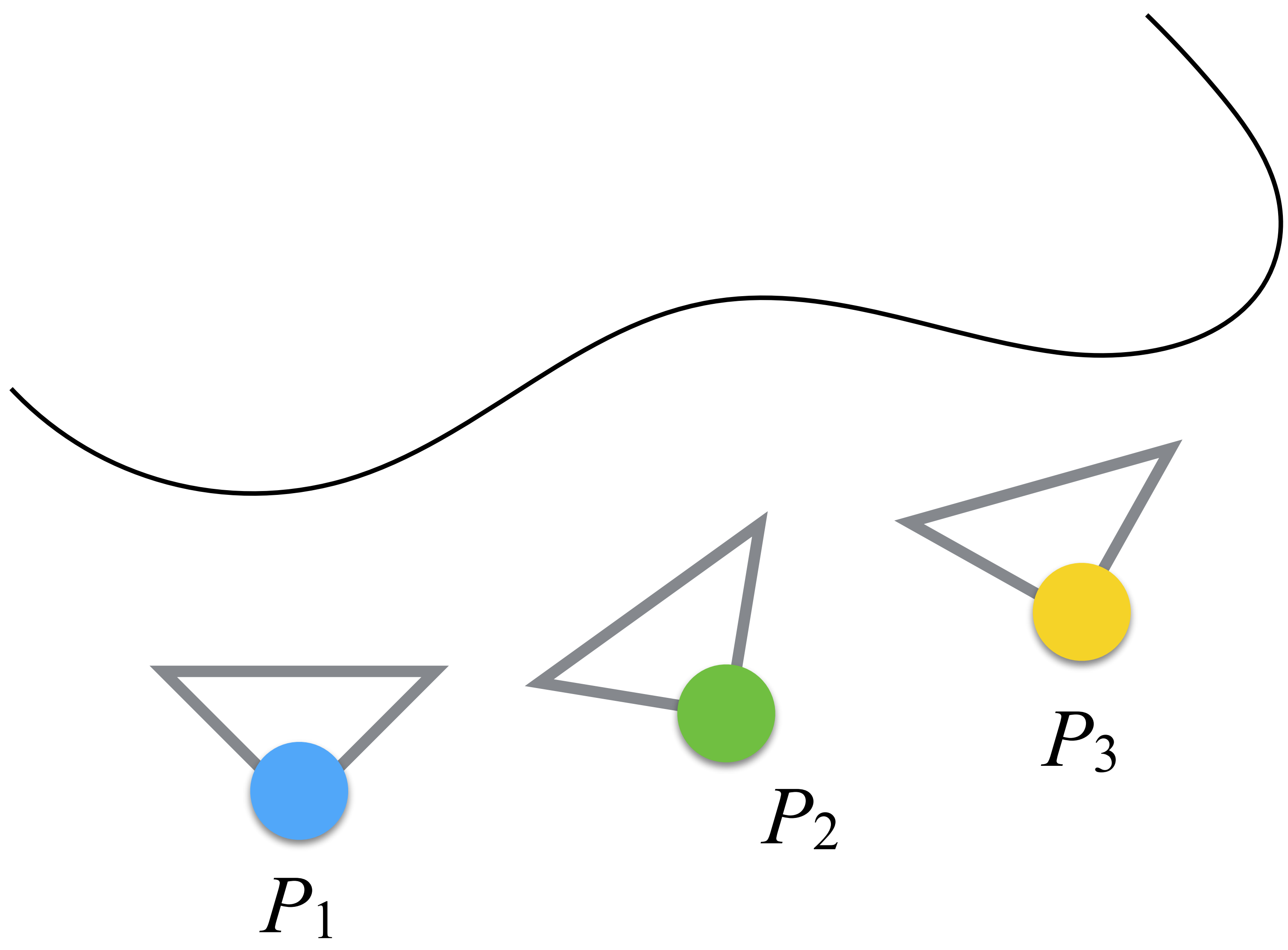
So far we have only used only a two cameras!

Structure From Motion: Multi-View

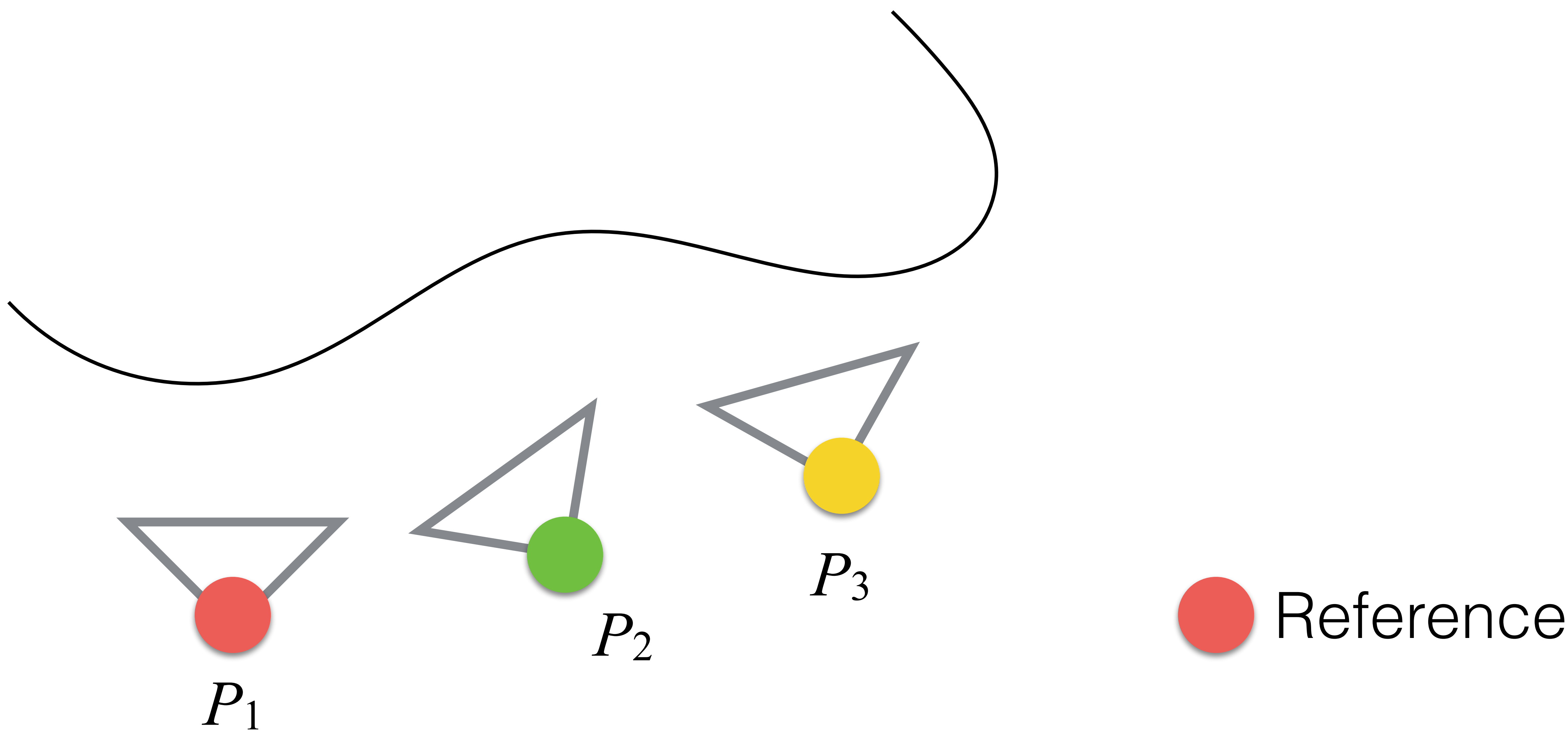
- We compute G for different views using the previous algorithm.

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

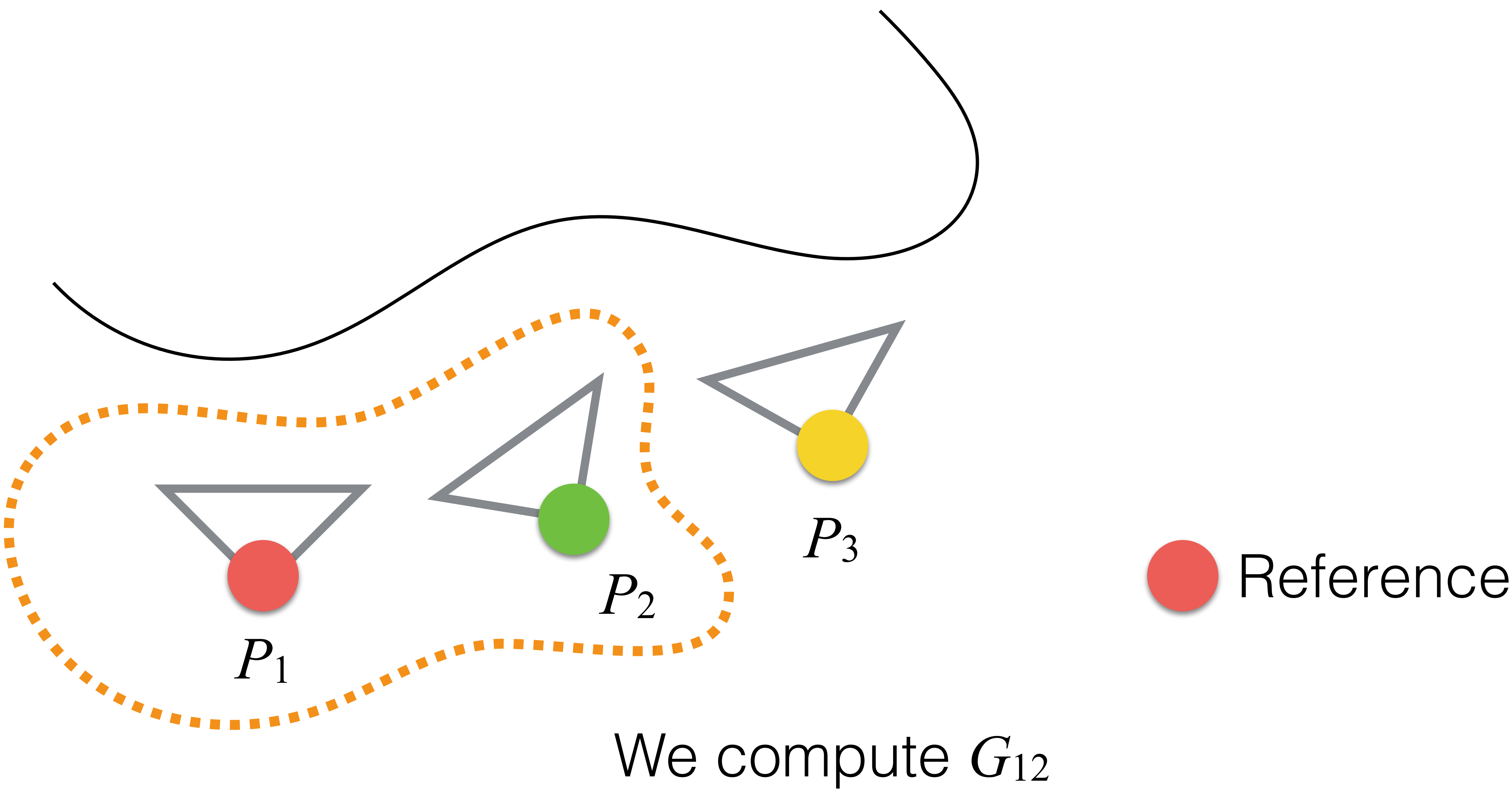
Structure From Motion: Multi-View Example



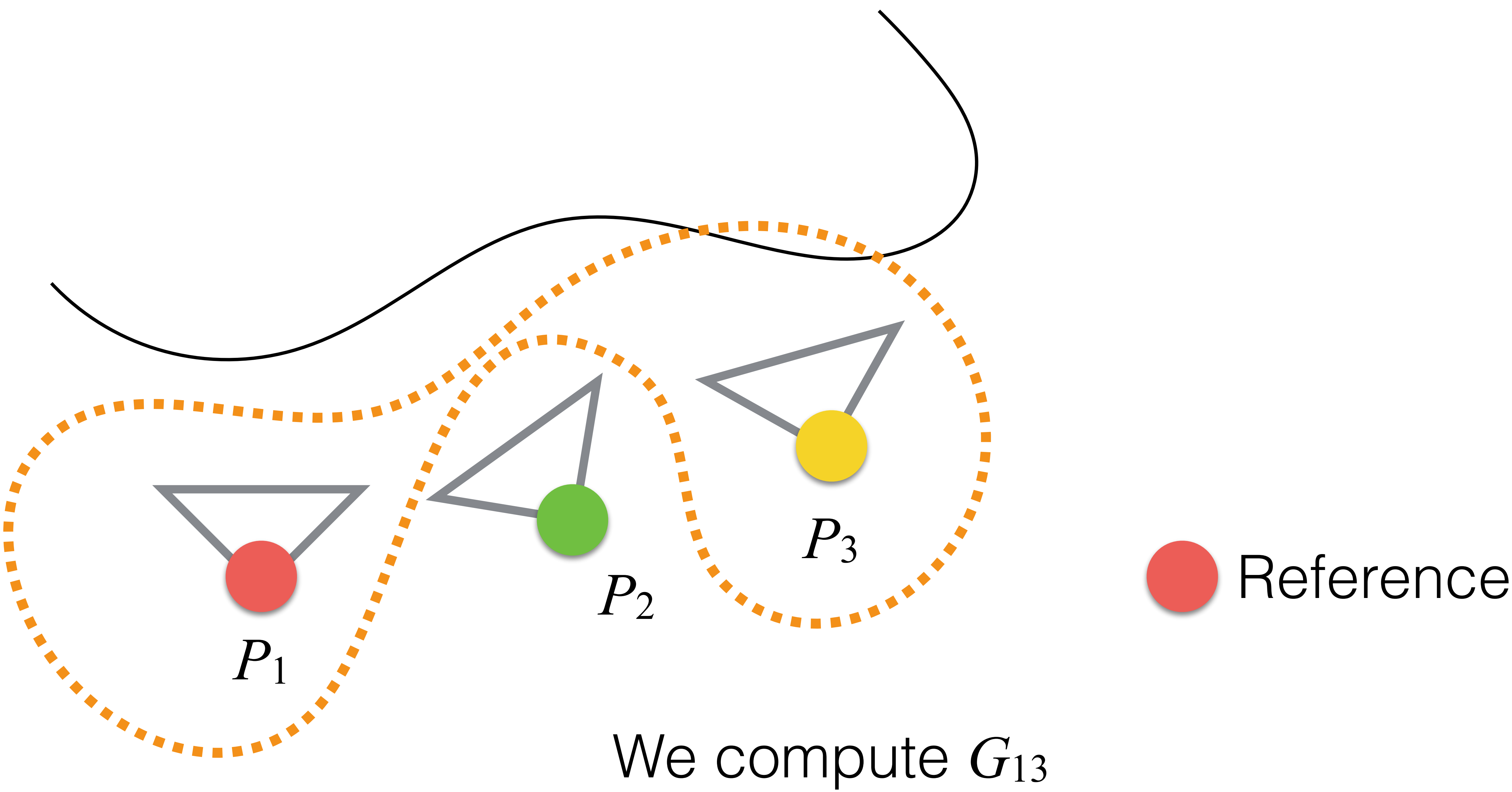
Structure From Motion: Multi-View Example



Structure From Motion: Multi-View Example



Structure From Motion: Multi-View Example



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

Hang on, what can possibly go wrong?

We are accumulating error, and we will drift from the solution!

Structure From Motion: Multi-View

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

$$\arg \min_{R_i, \mathbf{t}_i, \mathbf{M}^j} \sum_{i=1}^{l} \sum_{j=1}^{n} d\left(K_i \cdot [R_i | \mathbf{t}_i] \cdot \mathbf{M}^j, \mathbf{m}_i^j \right)^2$$

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

Structure From Motion: Multi-View

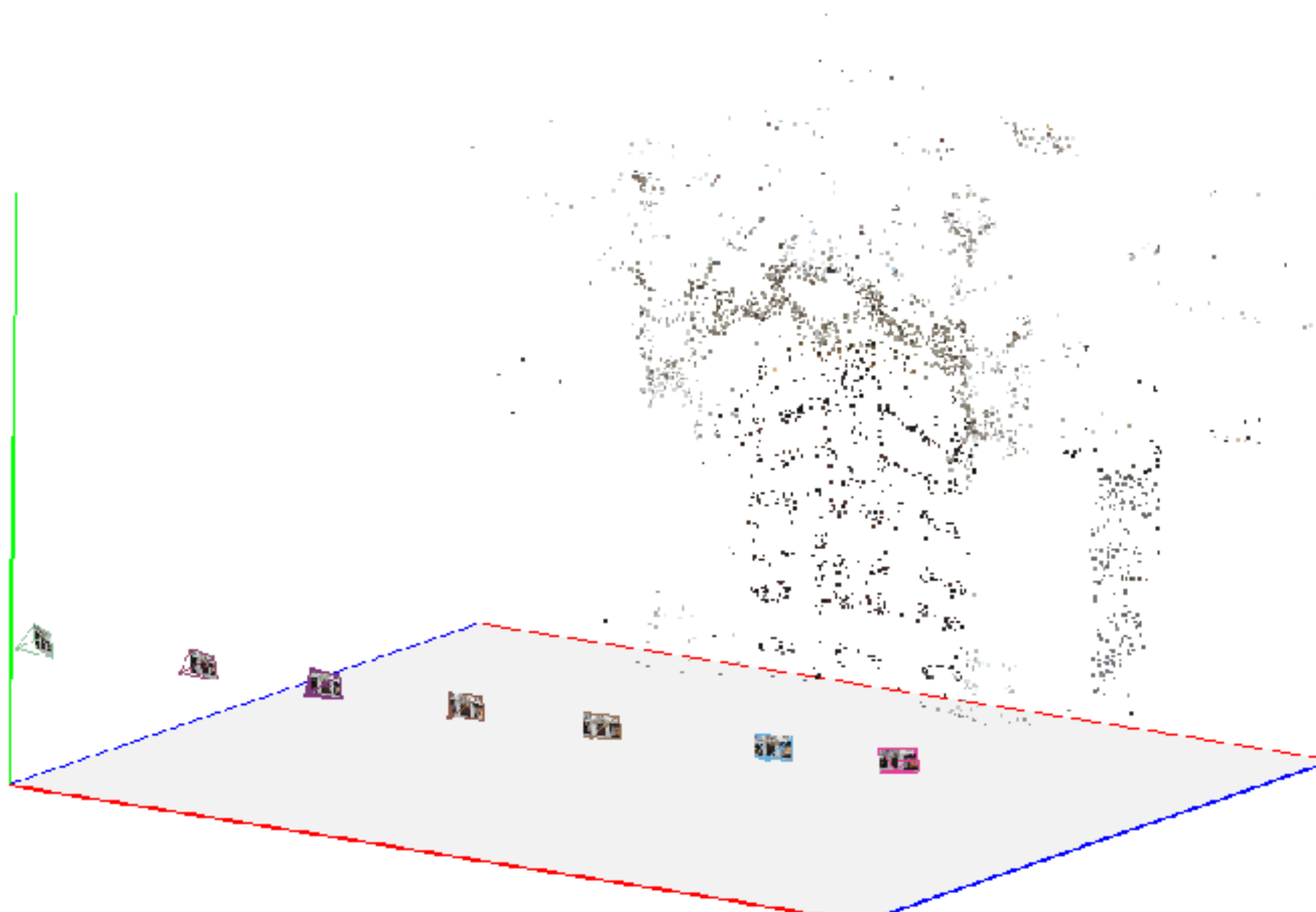
- Typically, the method is difficult to minimize as a whole thing. This is because there are many parameters to minimize.

- A two-step approach:

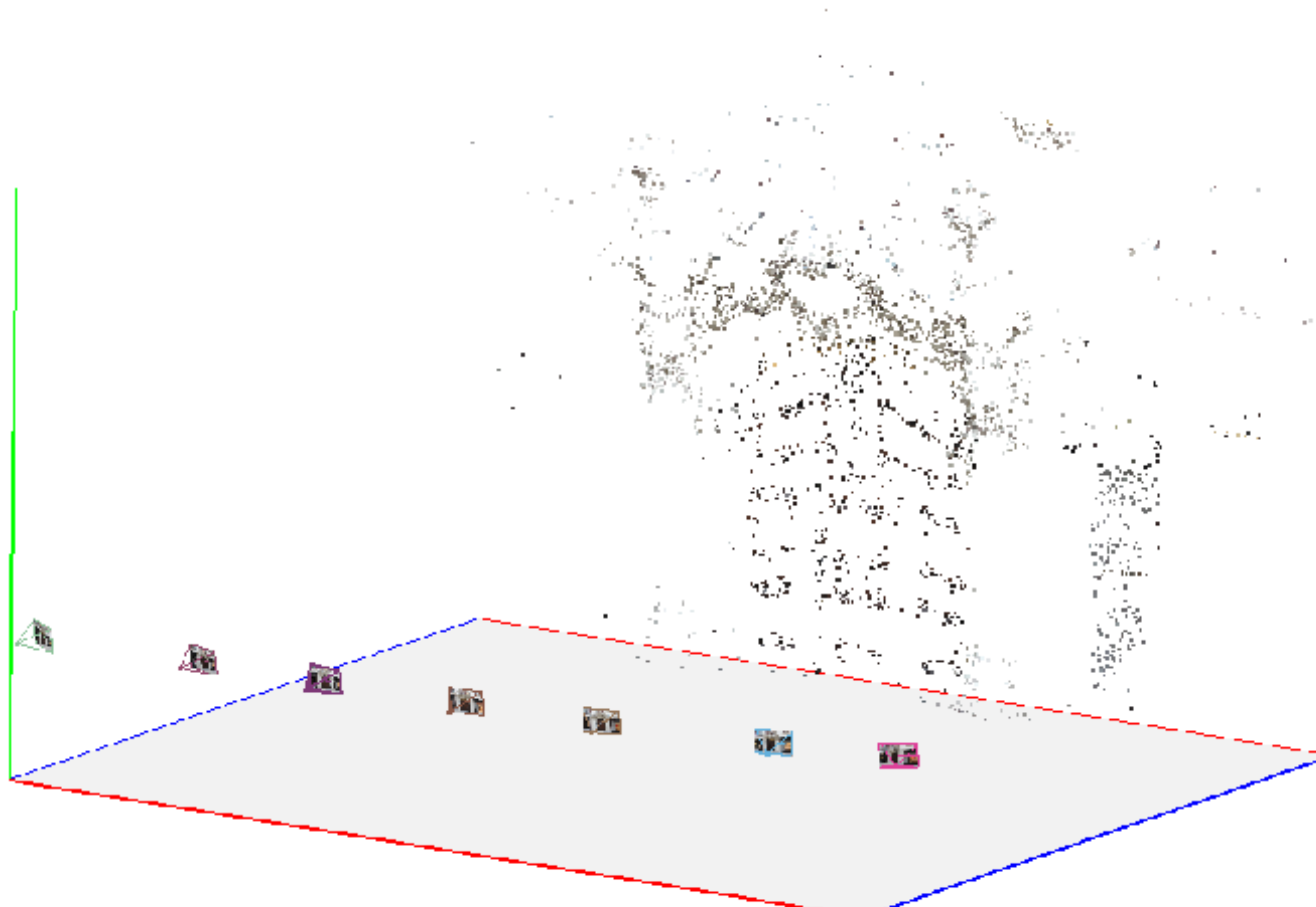
  - First, minimize (or viceversa) all extrinsic parameters (G) without modifying the 3D points.

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

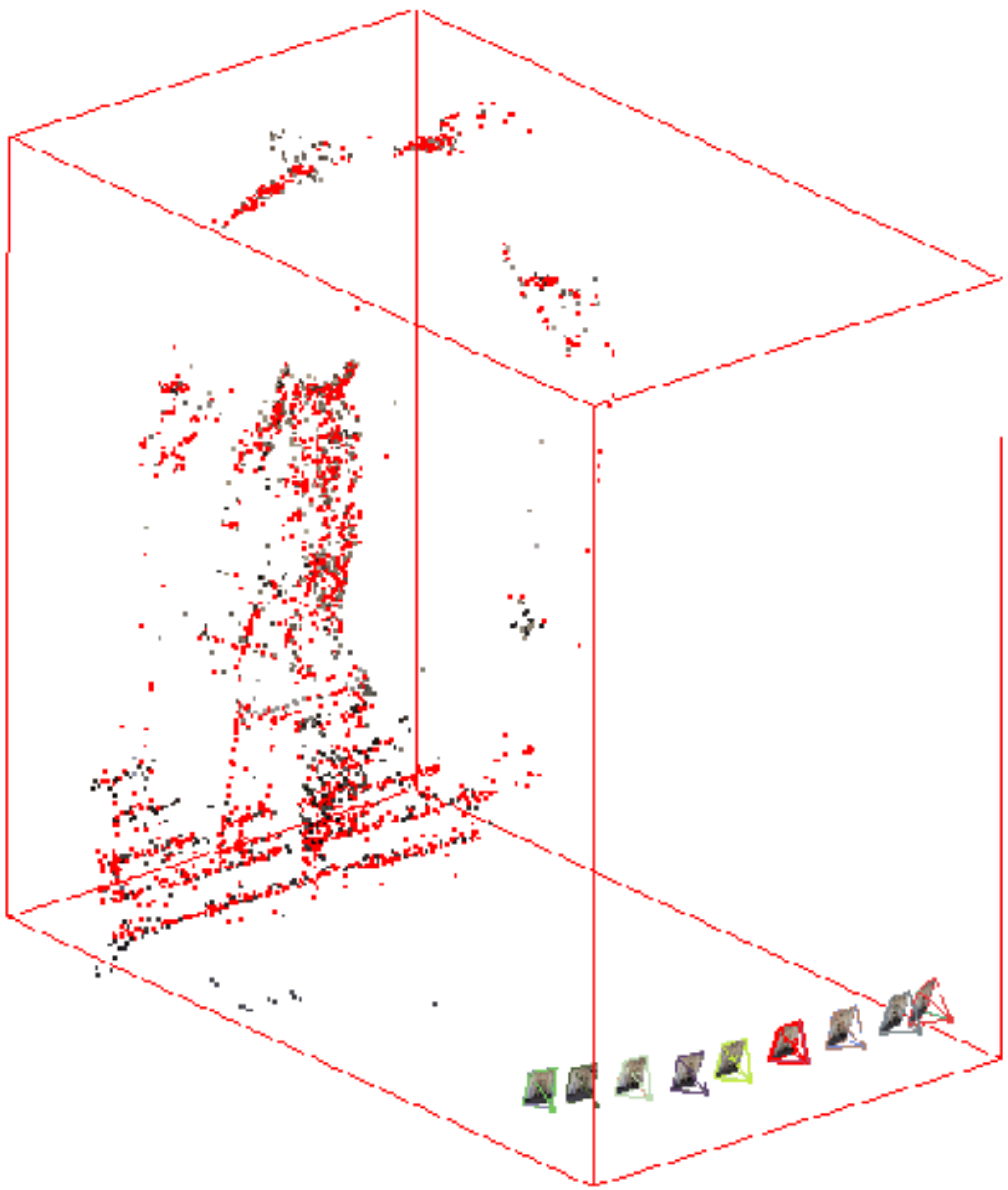
Structure From Motion: Example



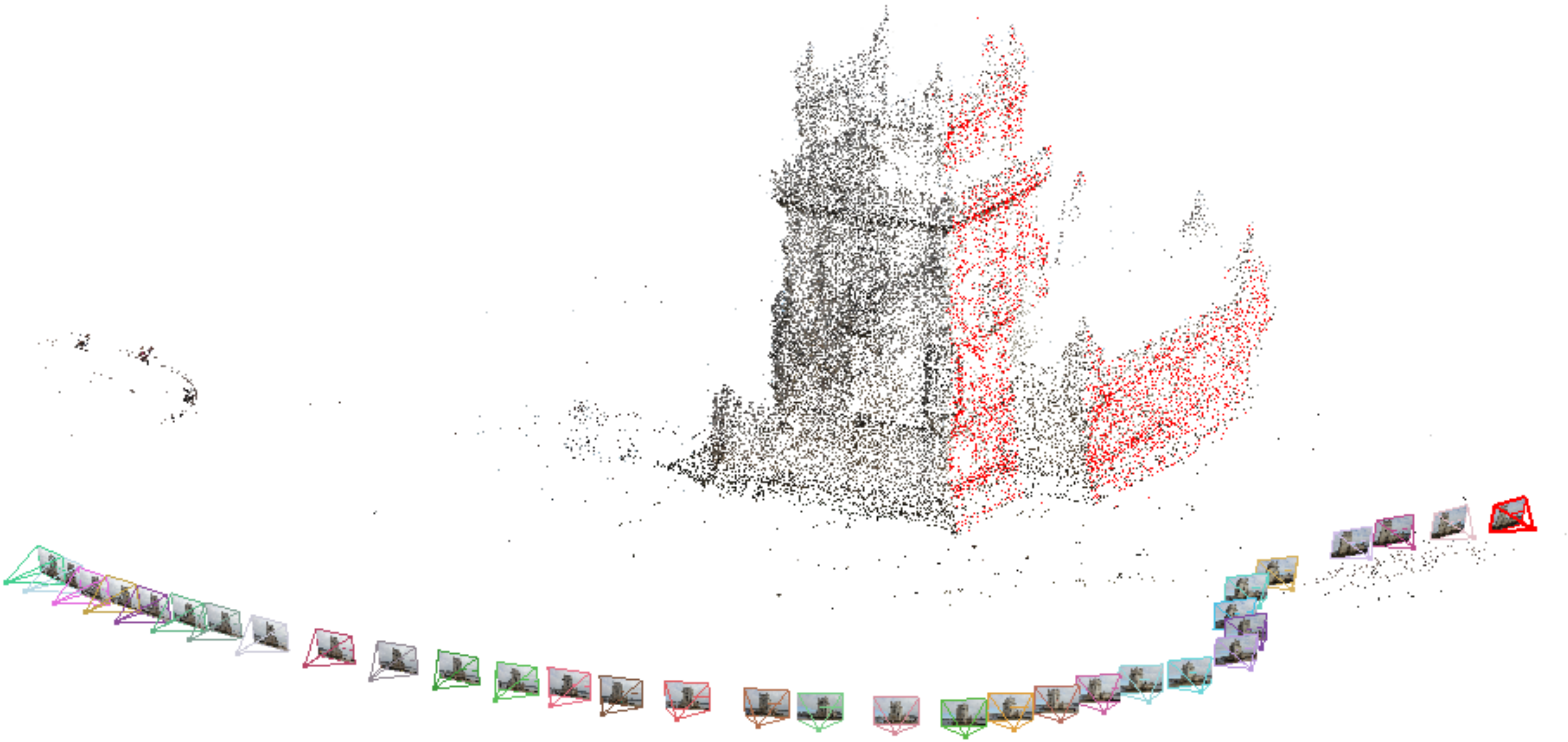
Structure From Motion: Example



Structure From Motion: Example



Structure From Motion: Example

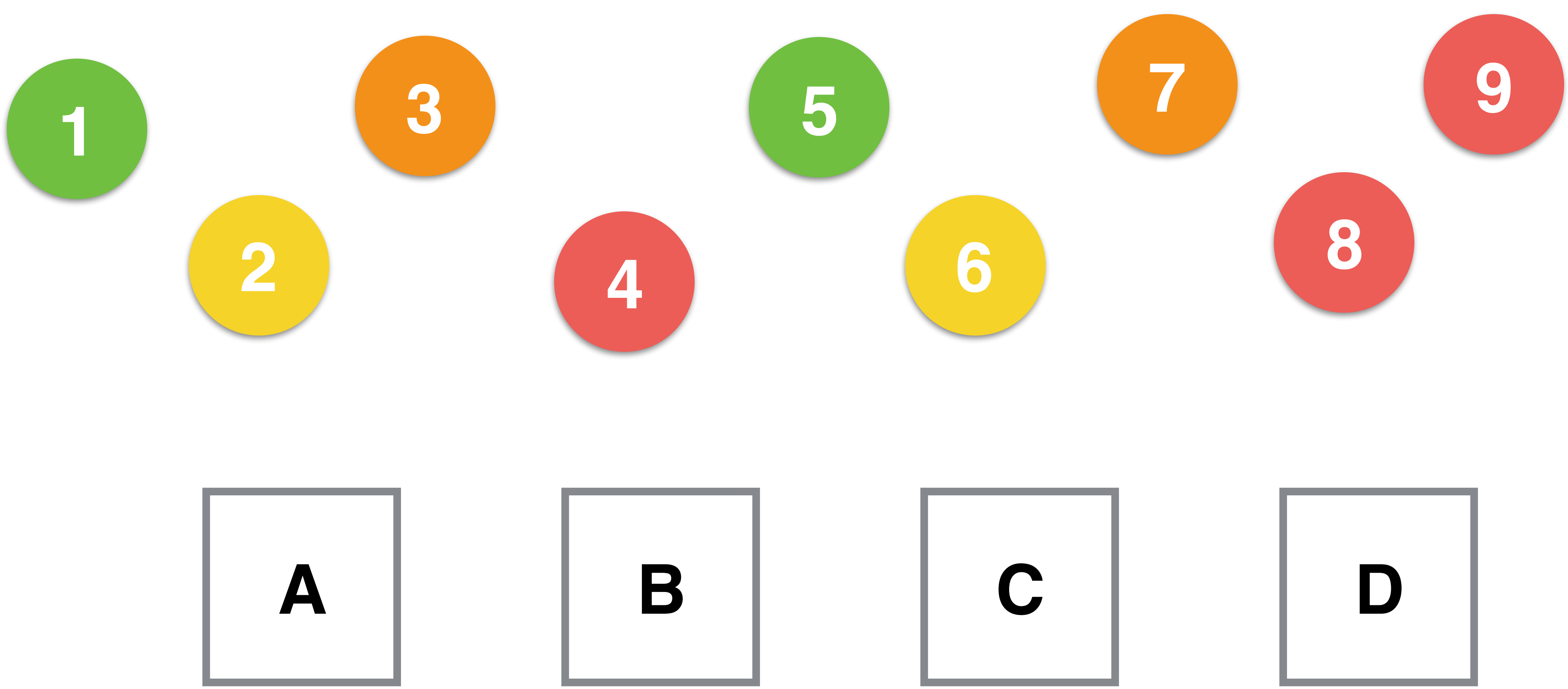


Structure From Motion: Multi-View

- 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.

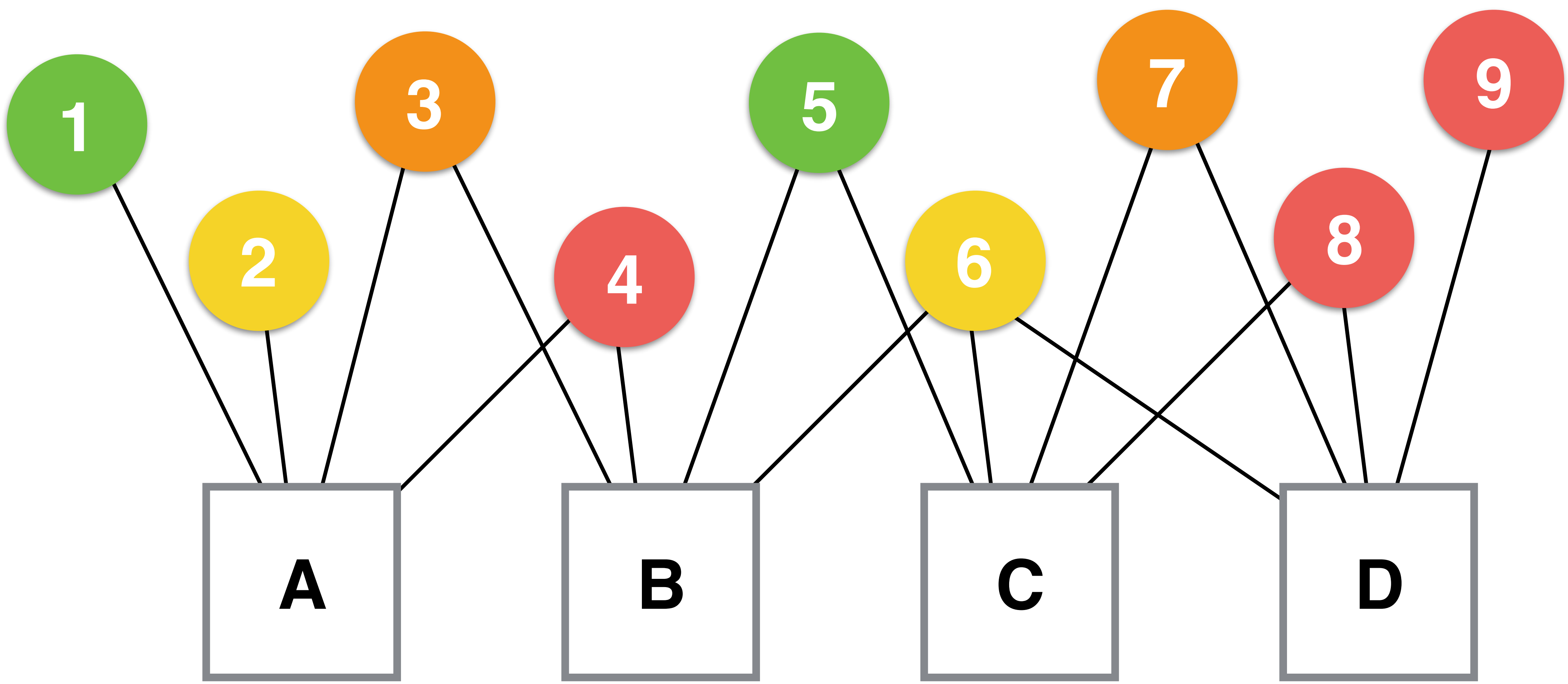
Structure From Motion: Multi-View

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



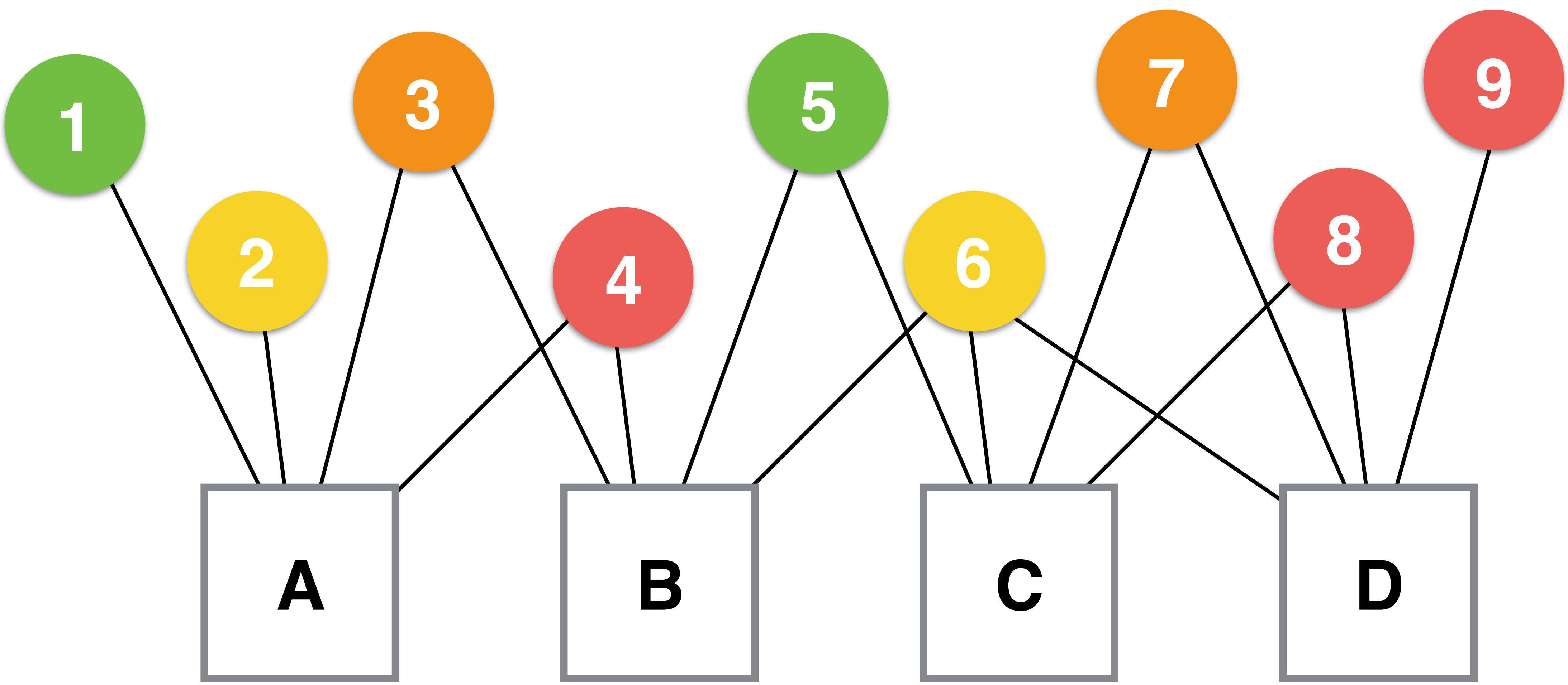
Structure From Motion: Multi-View

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



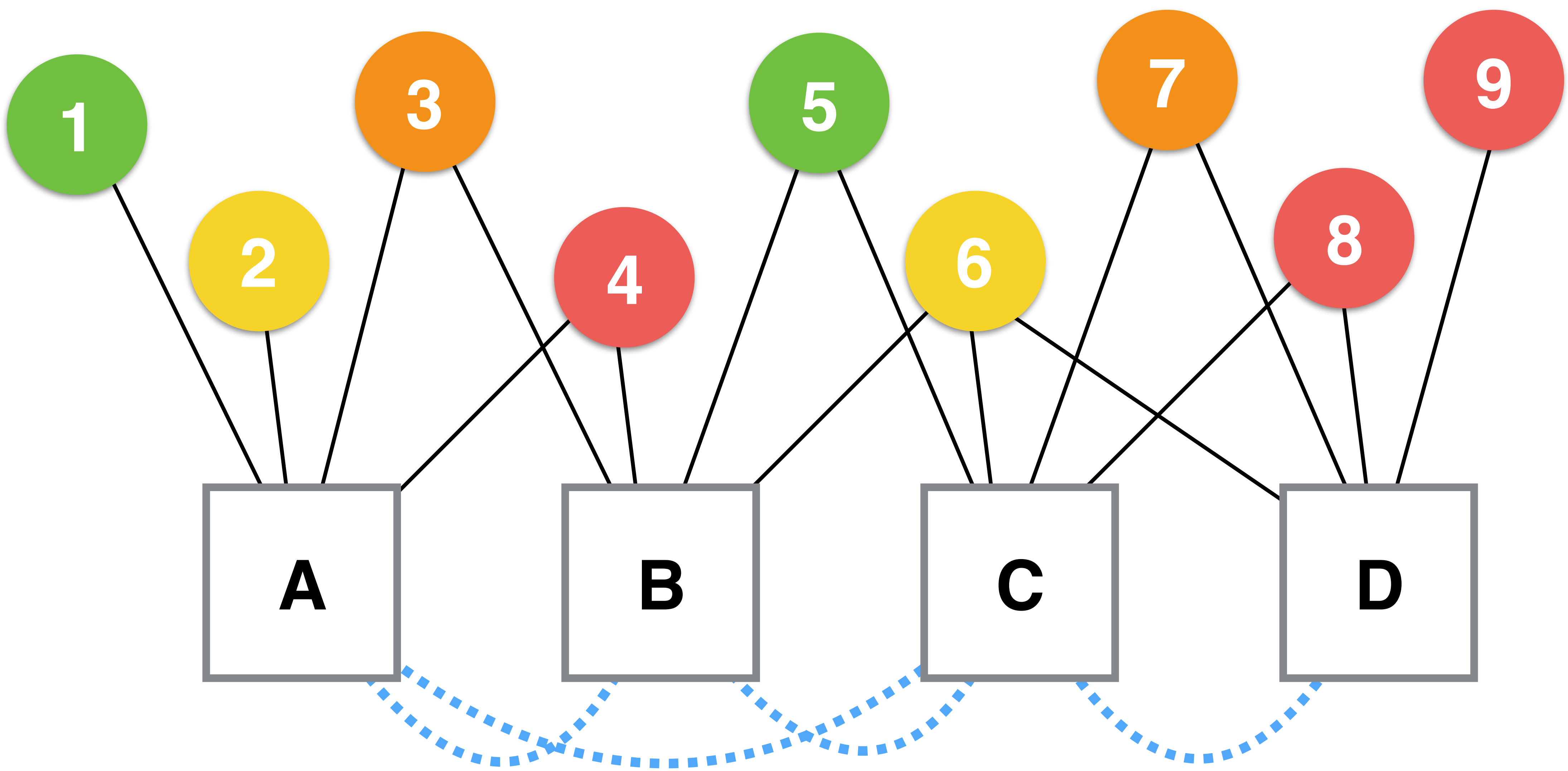
Structure From Motion: Multi-View

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



Structure From Motion: Multi-View

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



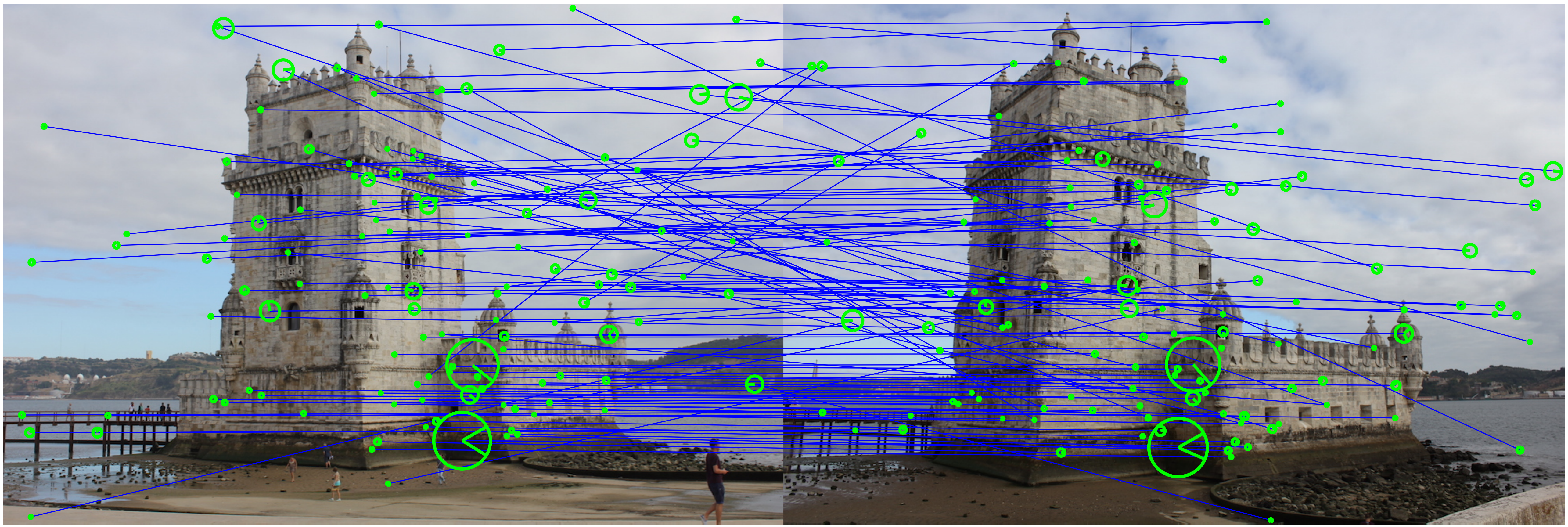
Structure From Motion: Multi-View

- 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: 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 a scale!

One thing…



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.

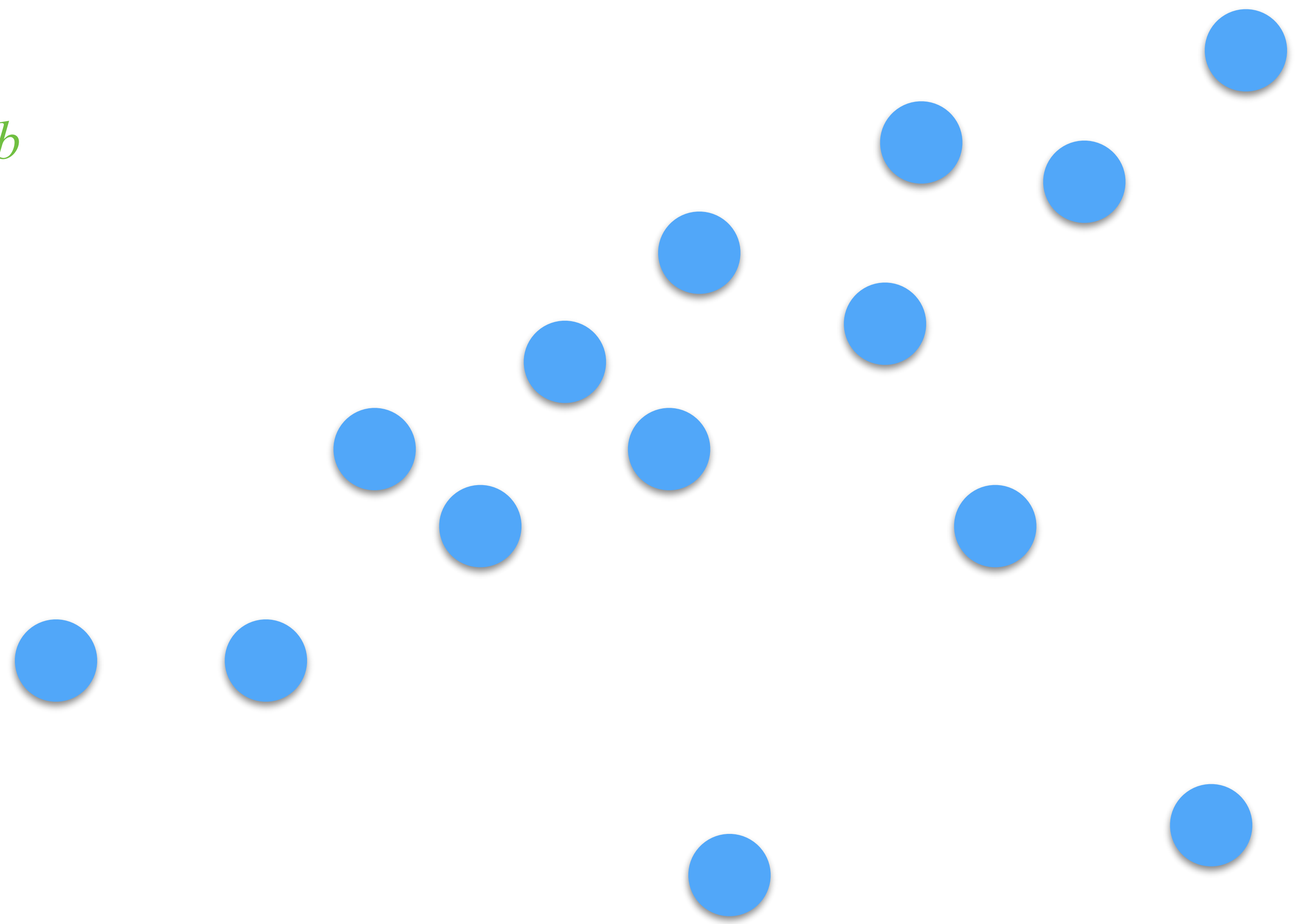
RANSAC

- $e = +\infty$ and $S_b = \varnothing$
- 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: Example

π: a straight line

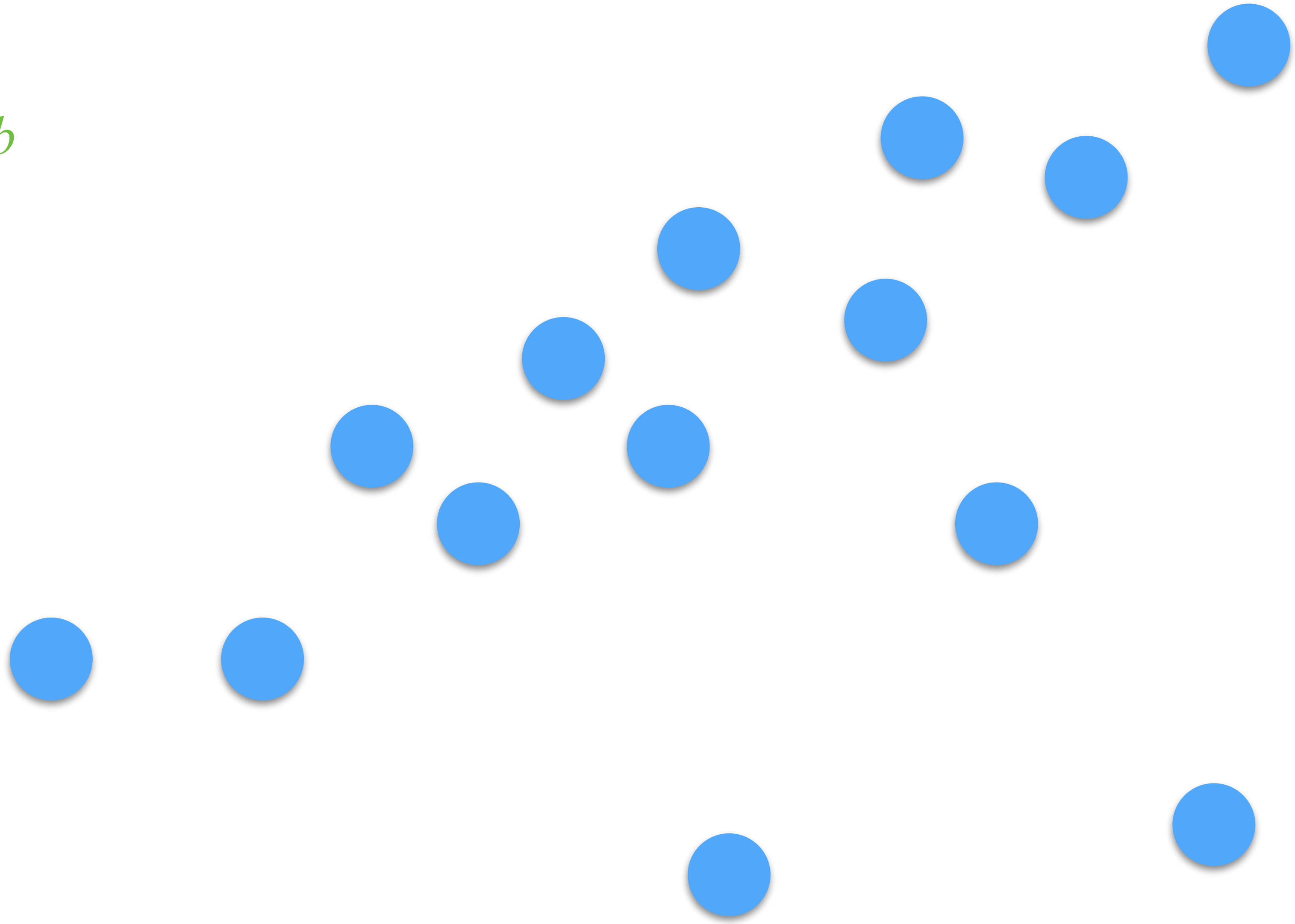
S_i and S_b



RANSAC: Example

π: a straight line

S_i and S_b



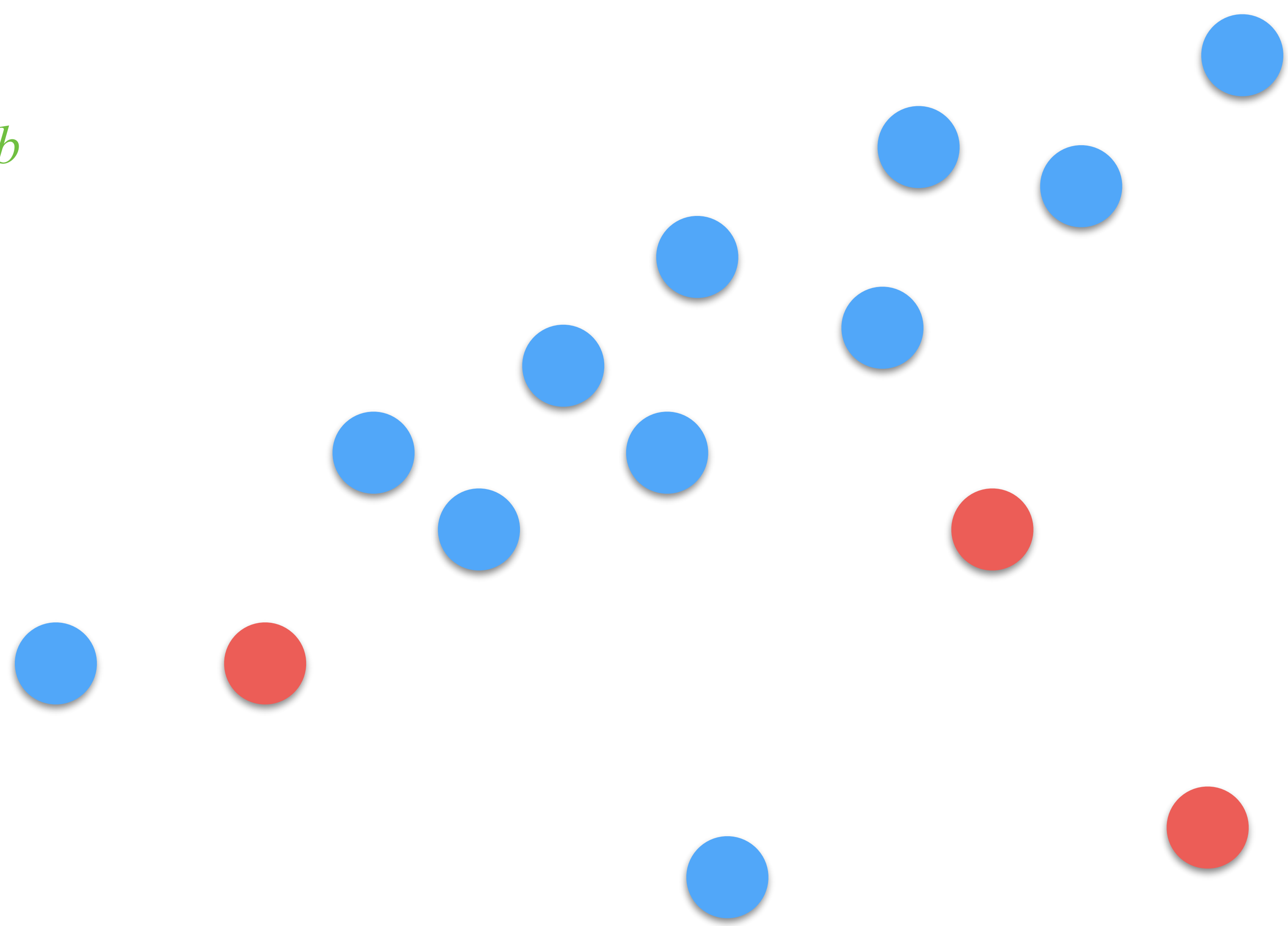
Iteration 0

$e = +\infty$

RANSAC: Example

π: a straight line

S_i and S_b



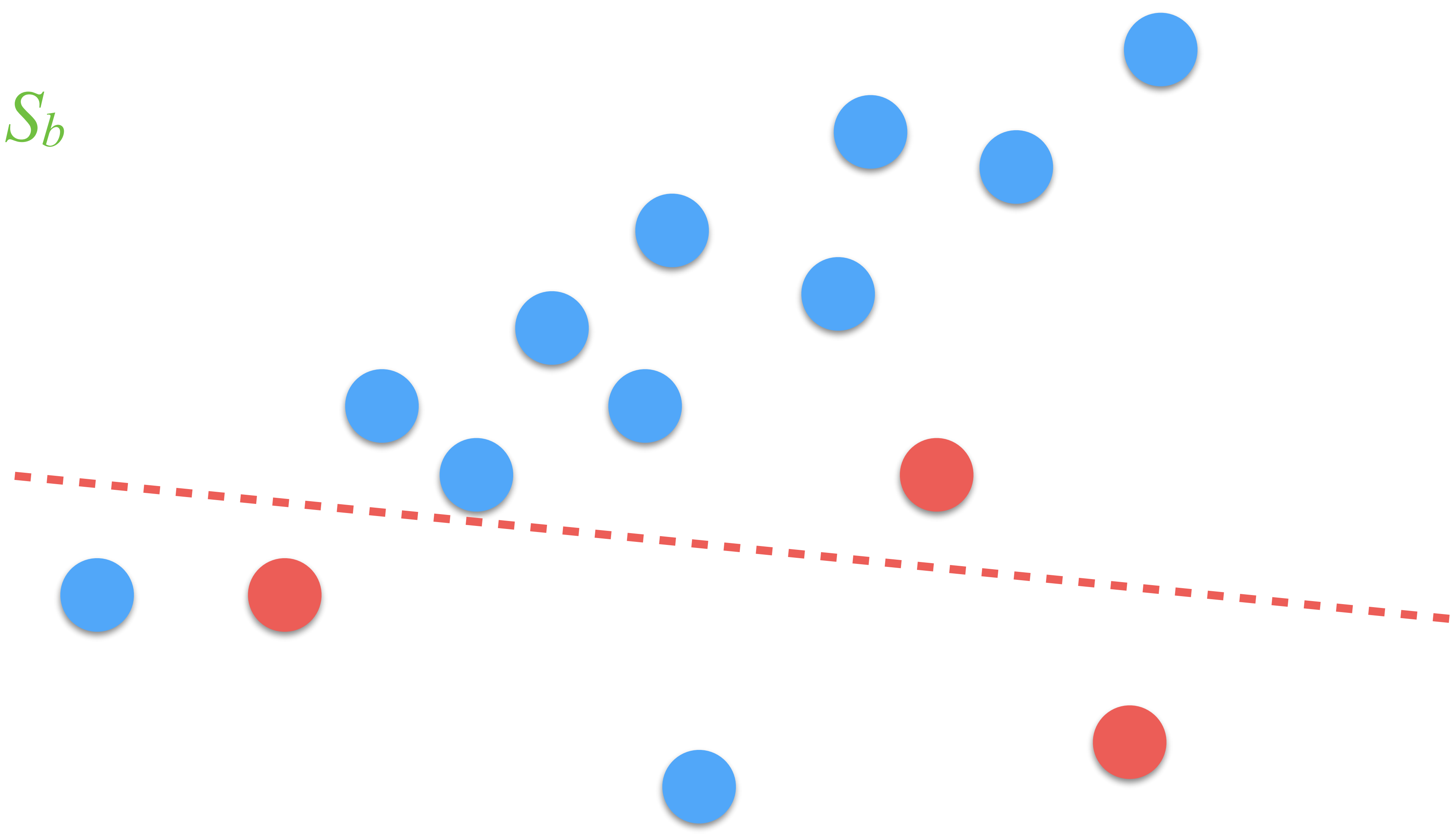
Iteration 1

$e = +\infty$

RANSAC: Example

π: a straight line

S_i and S_b



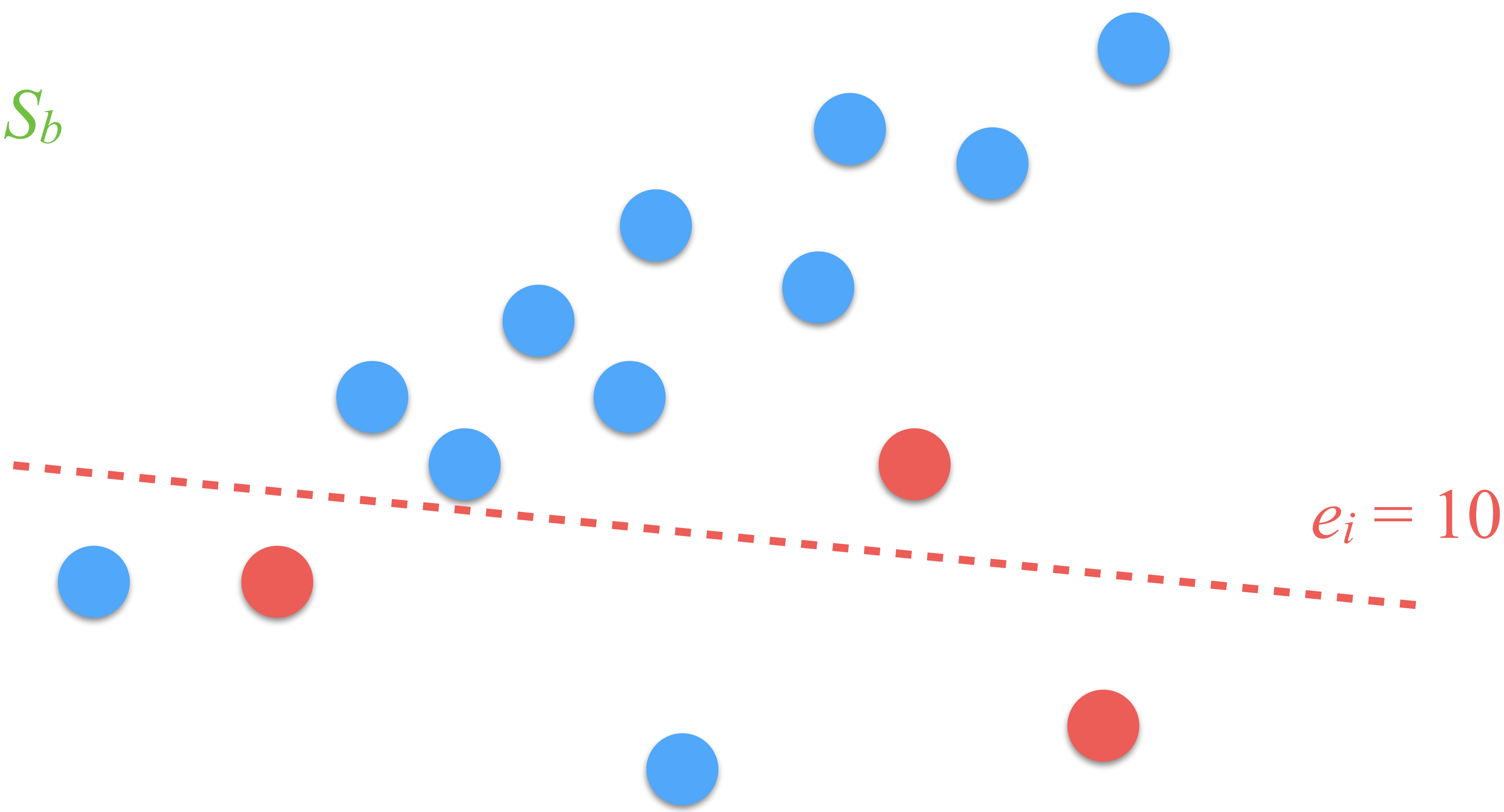
Iteration 1

$e = +\infty$

RANSAC: Example

π: a straight line

S_i and S_b



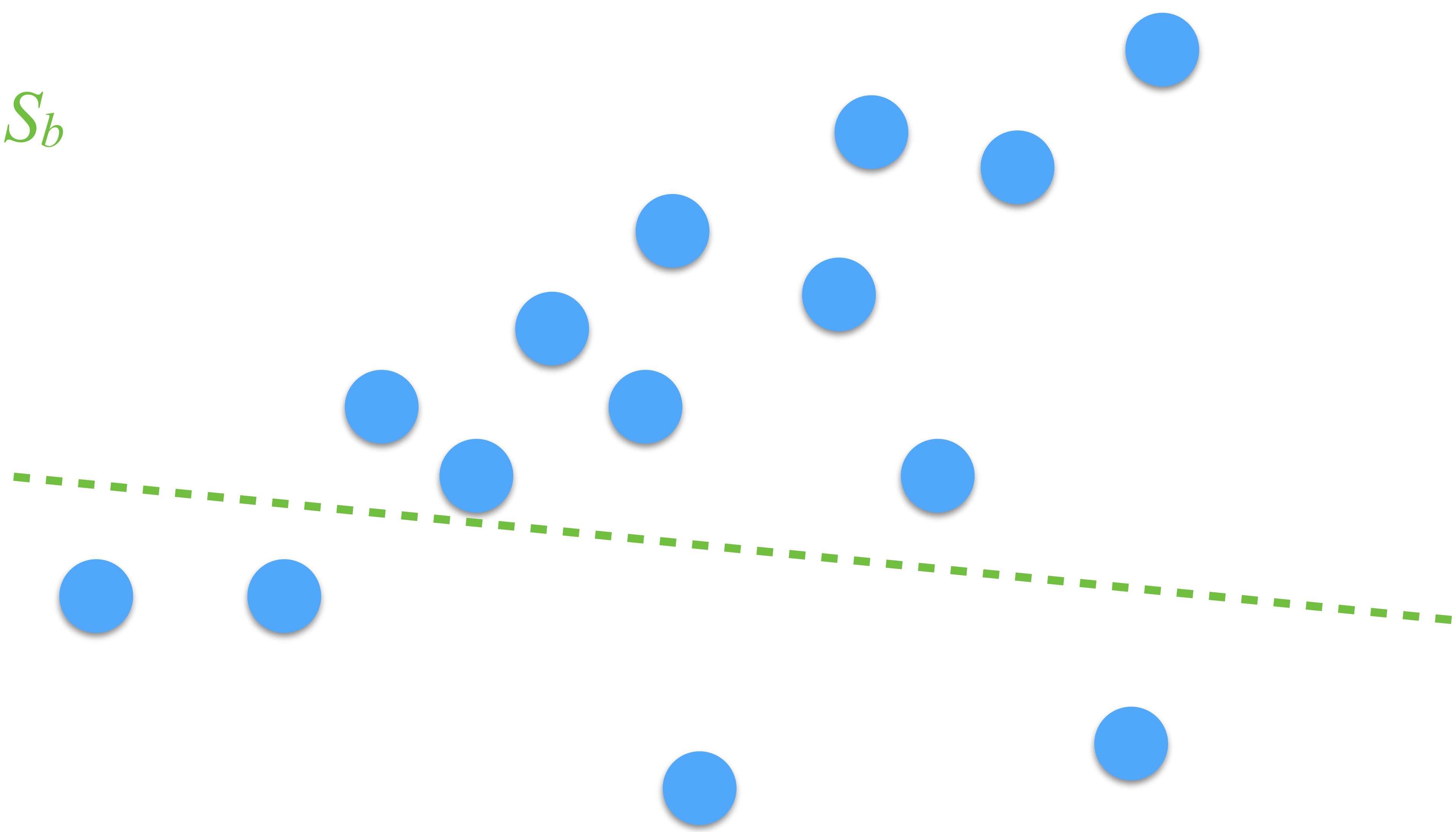
Iteration 1

$e = +\infty$

RANSAC: Example

π: a straight line

S_i and S_b

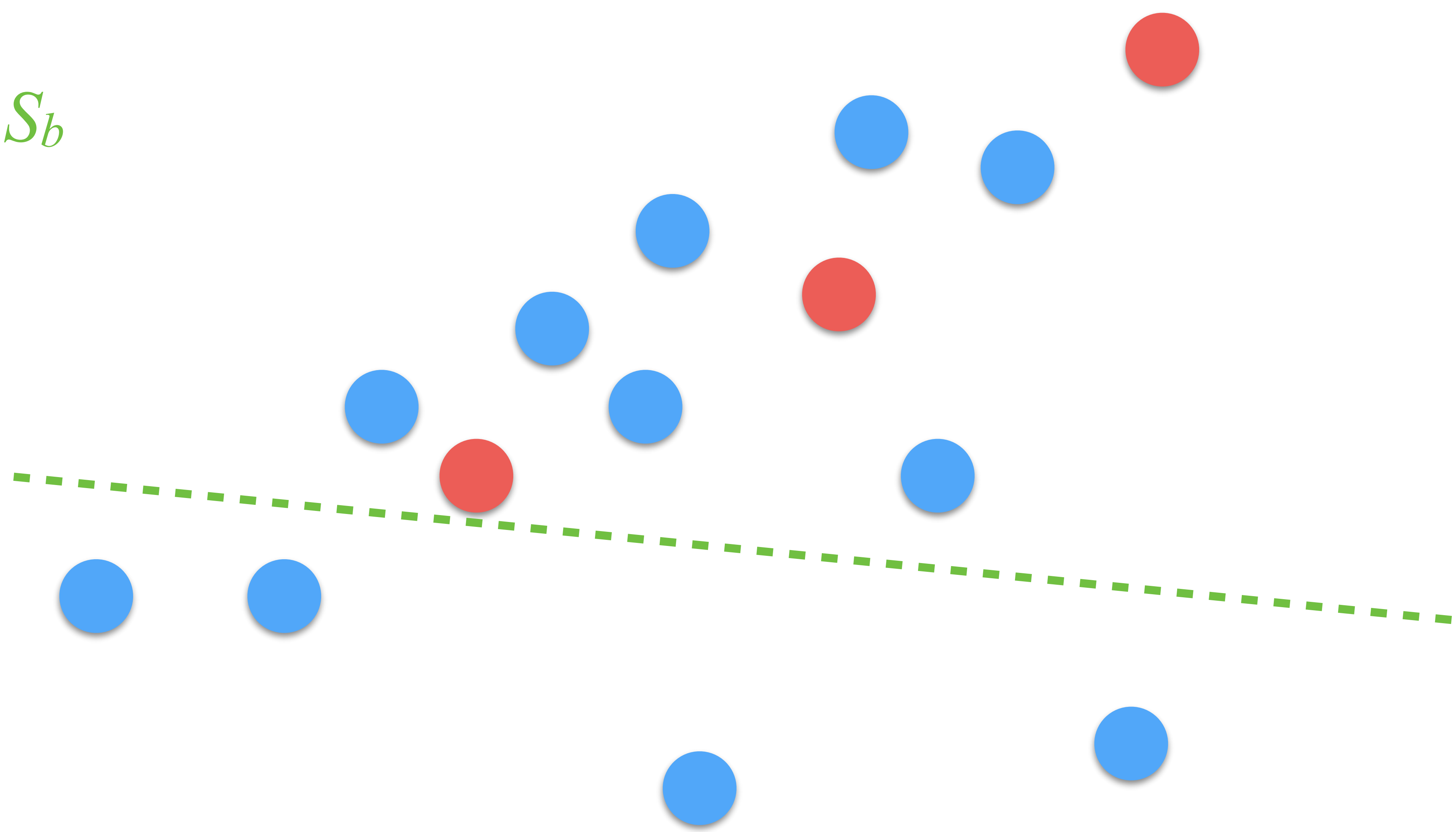


Iteration 2
$e = 10$

RANSAC: Example

π: a straight line

S_i and S_b



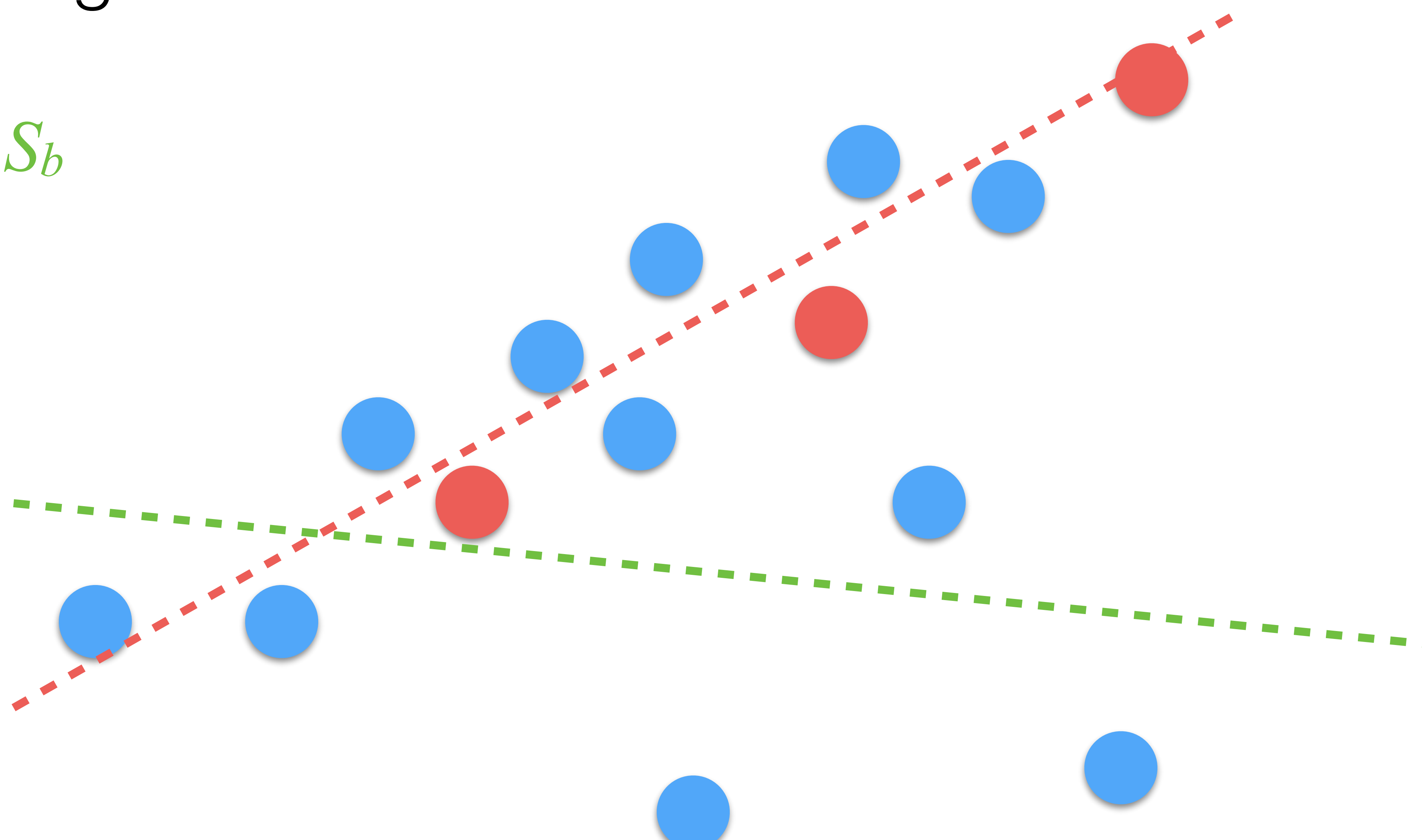
Iteration 2

$e = 10$

RANSAC: Example

π: a straight line

S_i and S_b



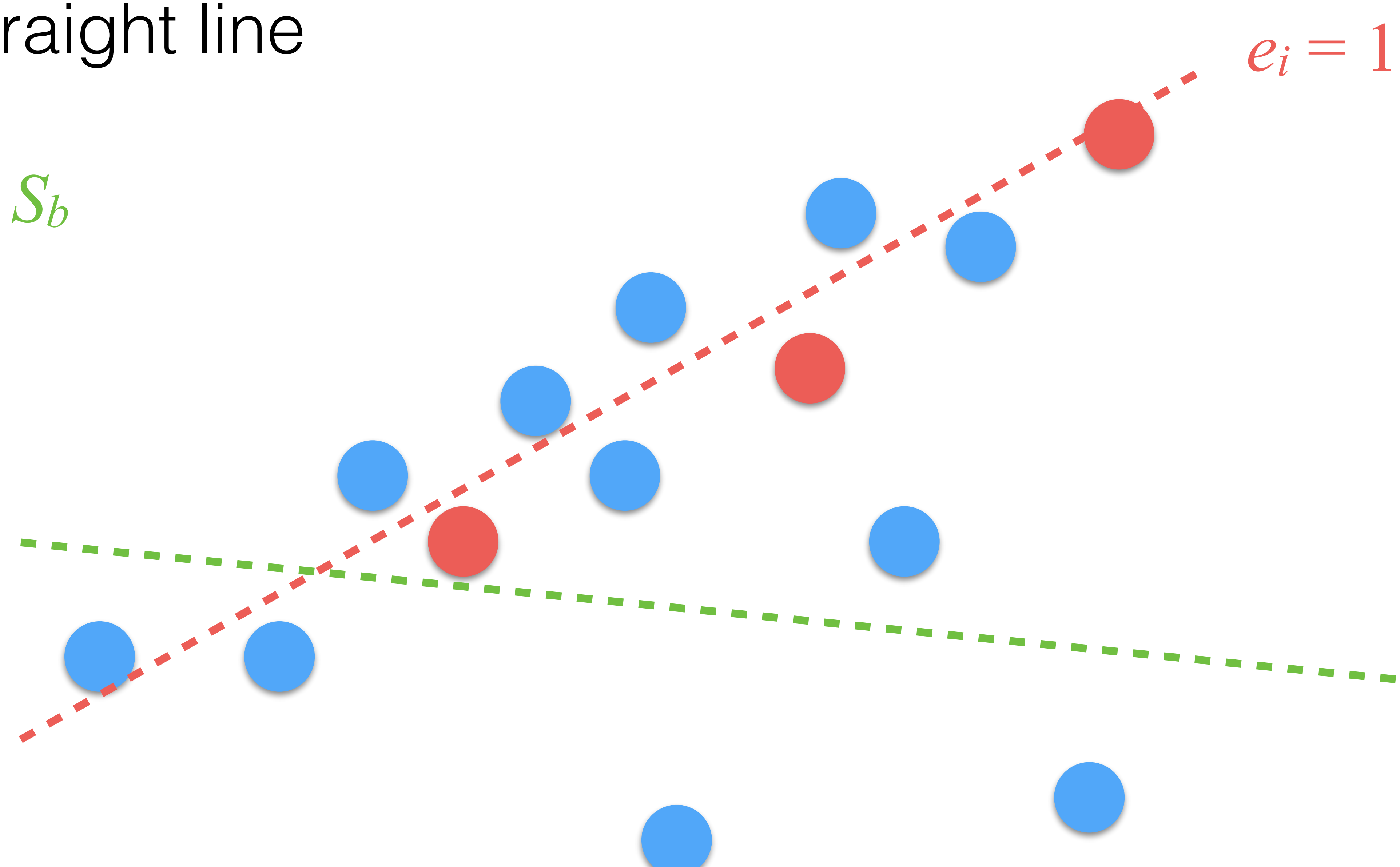
Iteration 2

$e = 10$

RANSAC: Example

π: a straight line

S_i and S_b



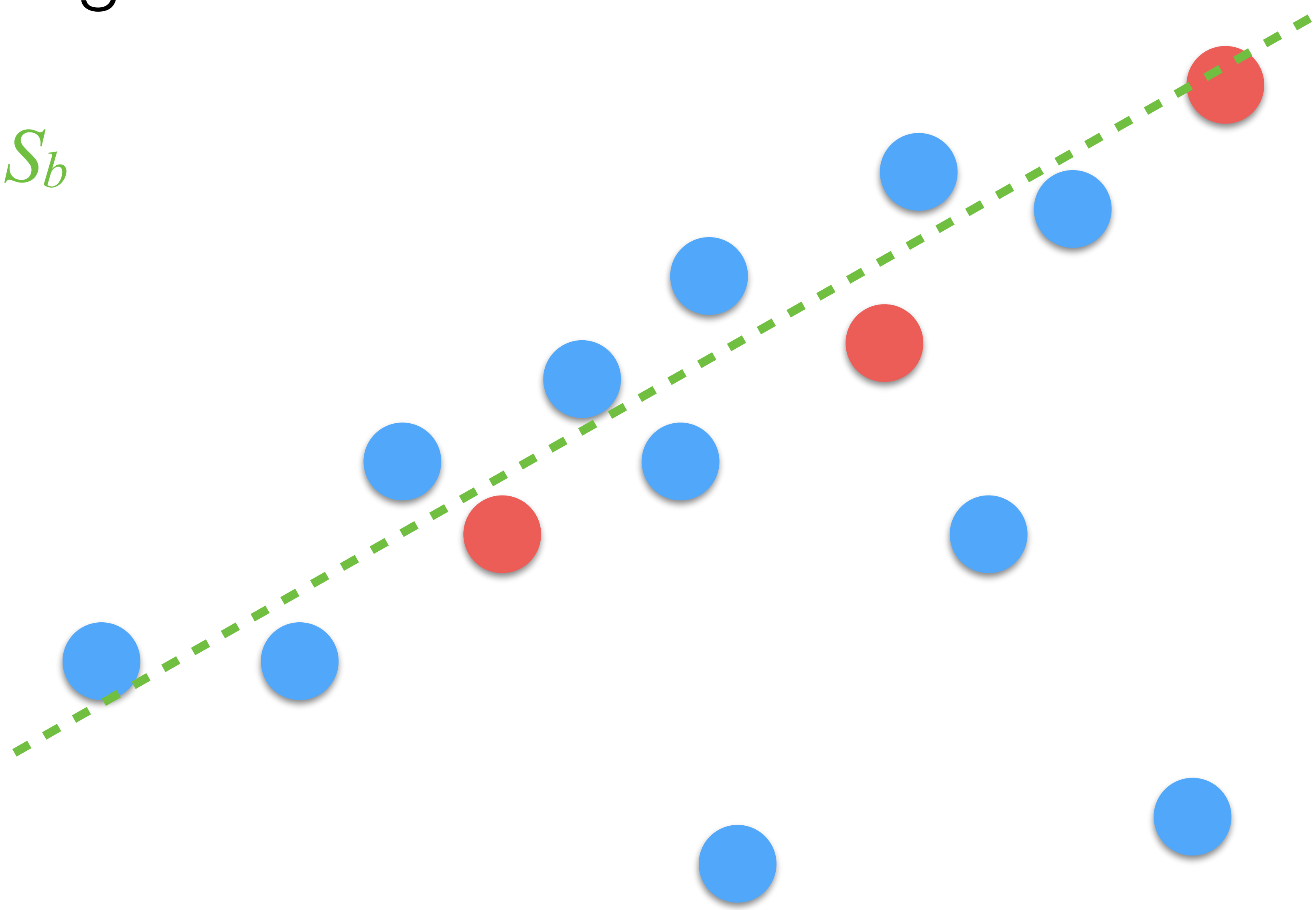
Iteration 2

$e = 10$

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:
  - 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:

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

  - 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!