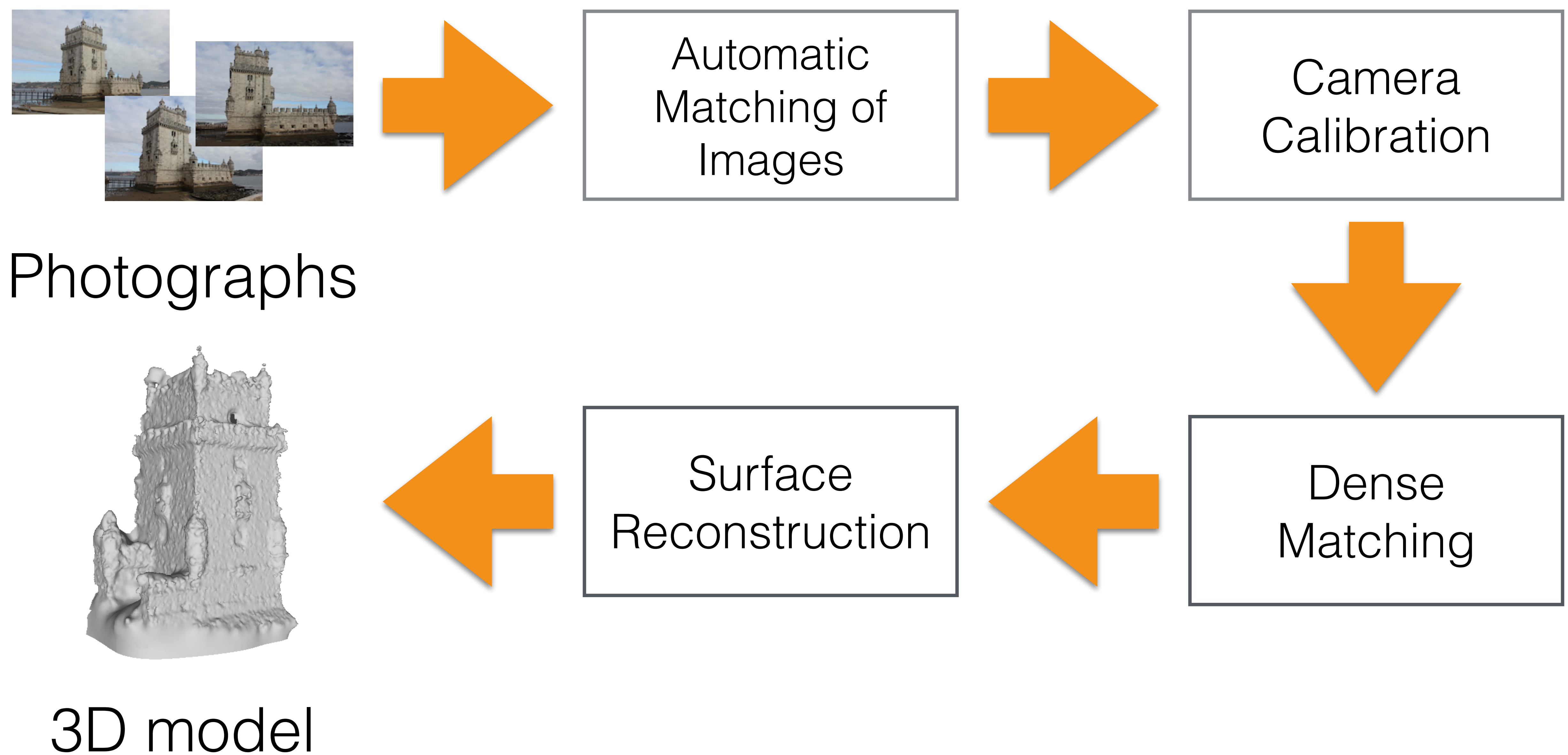
3D from Photographs: Dense Matching

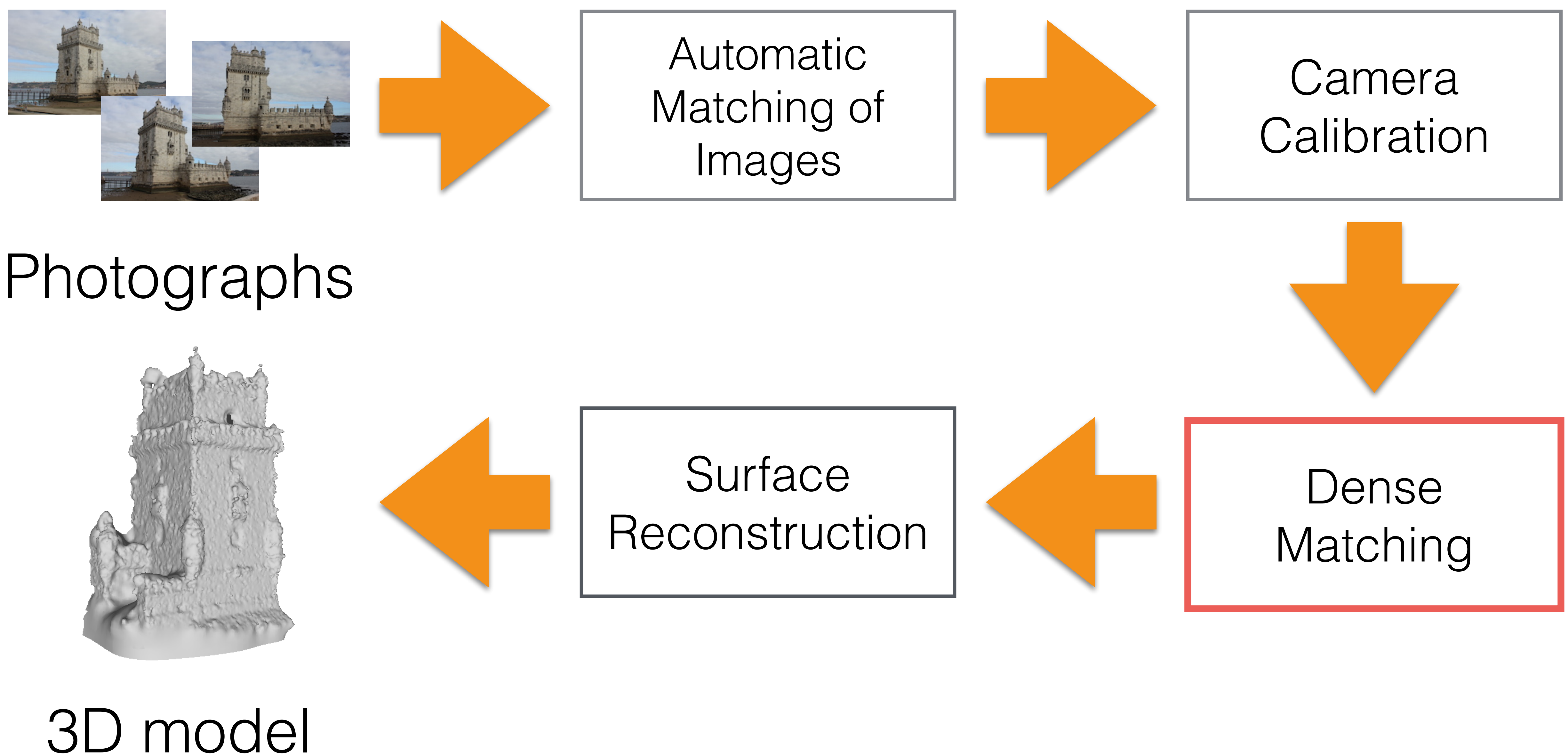
Francesco Banterle, Ph.D.
francesco.banterle@isti.cnr.it

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

3D from Photographs



3D from Photographs



Dense Matching

- Once we have cameras and sparse matching we can proceed in several ways:
  - Stereo.
  - Multi-View Stereo.

Stereo

Stereo

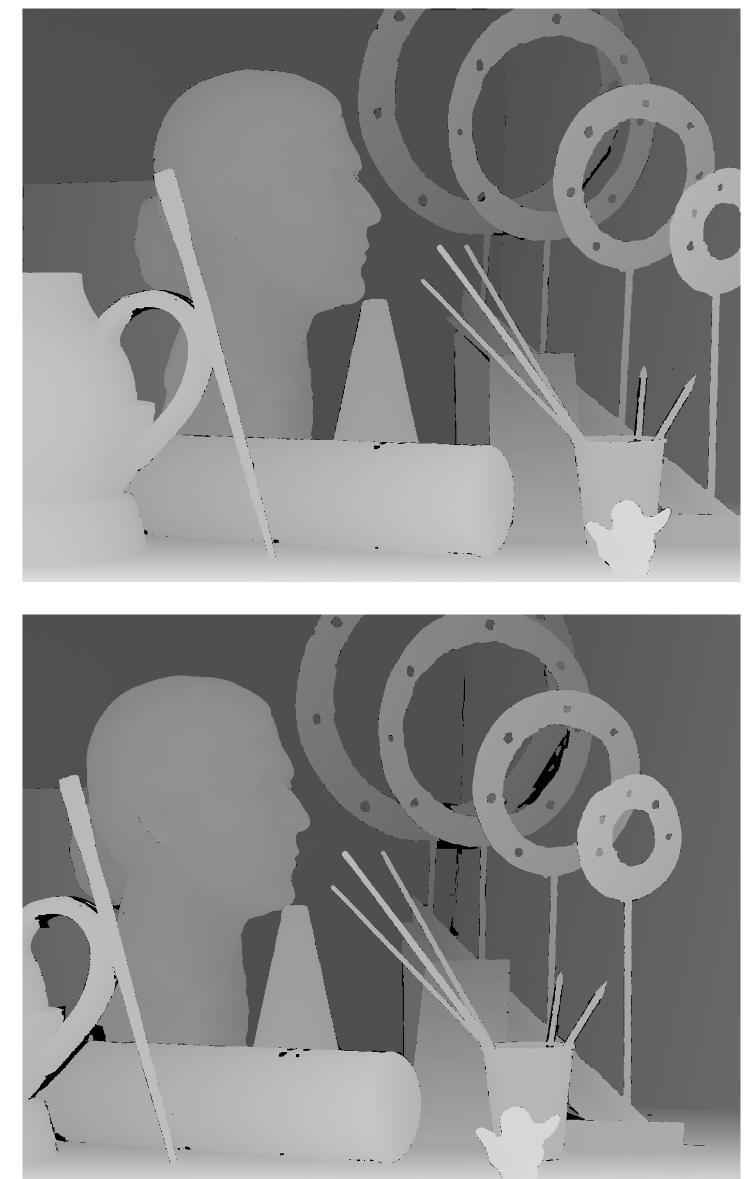
- **Input**: two images, I_1 and I_2, of the same scene taken from different positions (no pure rotational!) and their camera matrices (P_1 and P_2):
    - Optionally, we can have sparse 3D points or matched points if this is a SfM input.
- **Output**: a depth map for each image; i.e., two depth maps.

Stereo: Example

$+ \ P_1$

$+ \ P_2$

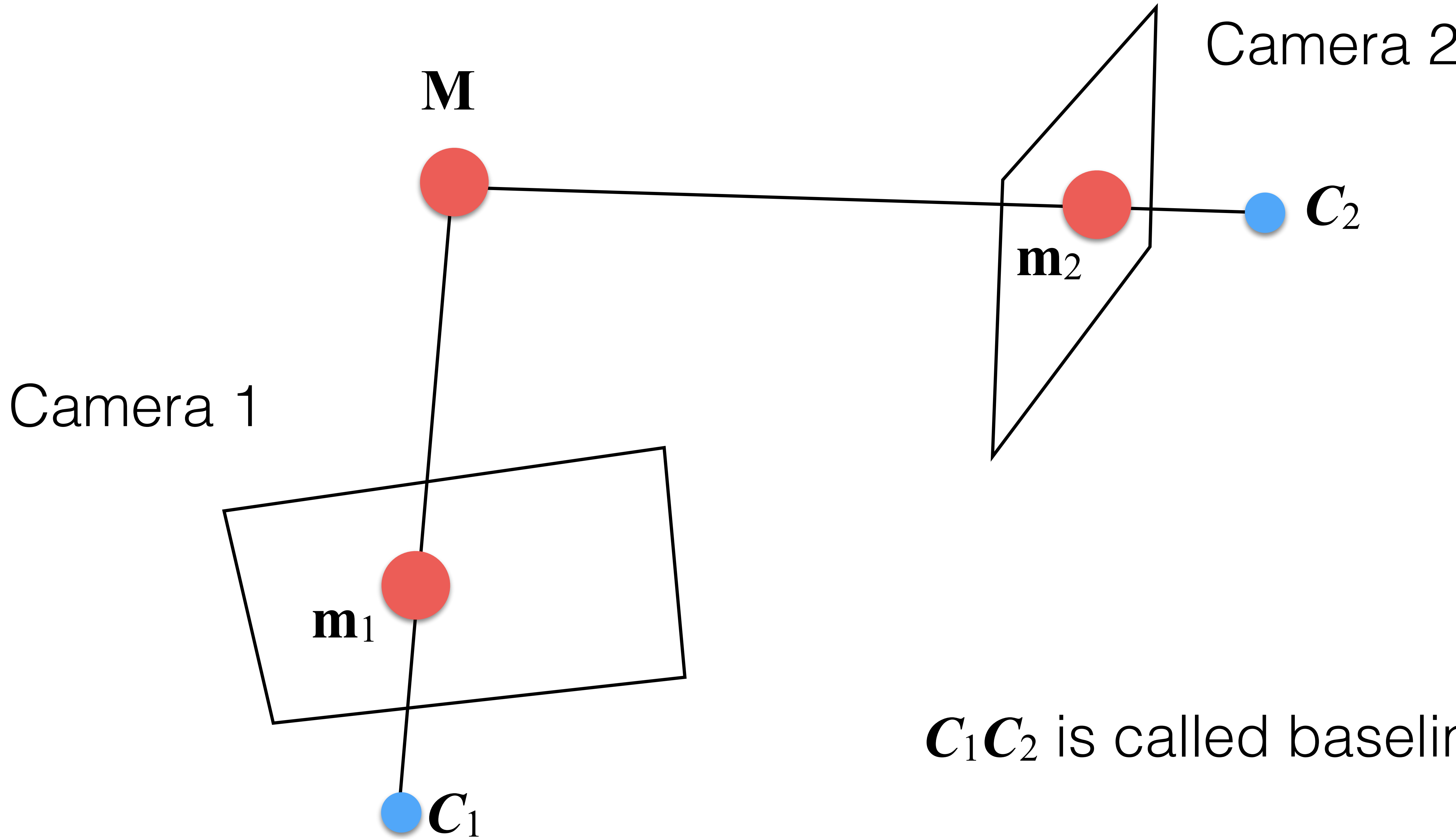
Input



Output

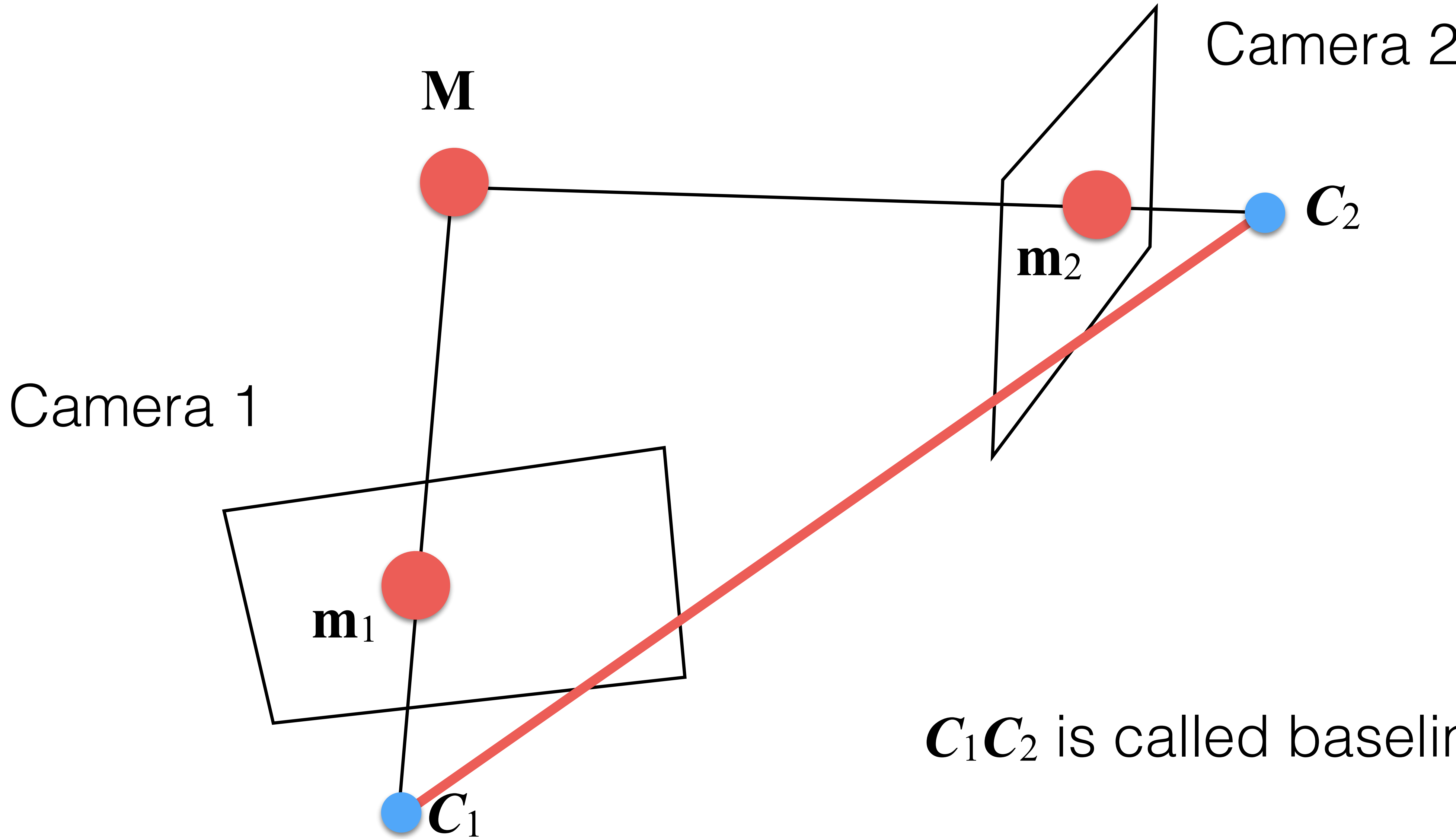
Images from the Middlebury Dataset:
http://vision.middlebury.edu/stereo/data/scenes2005/

Epipolar Geometry



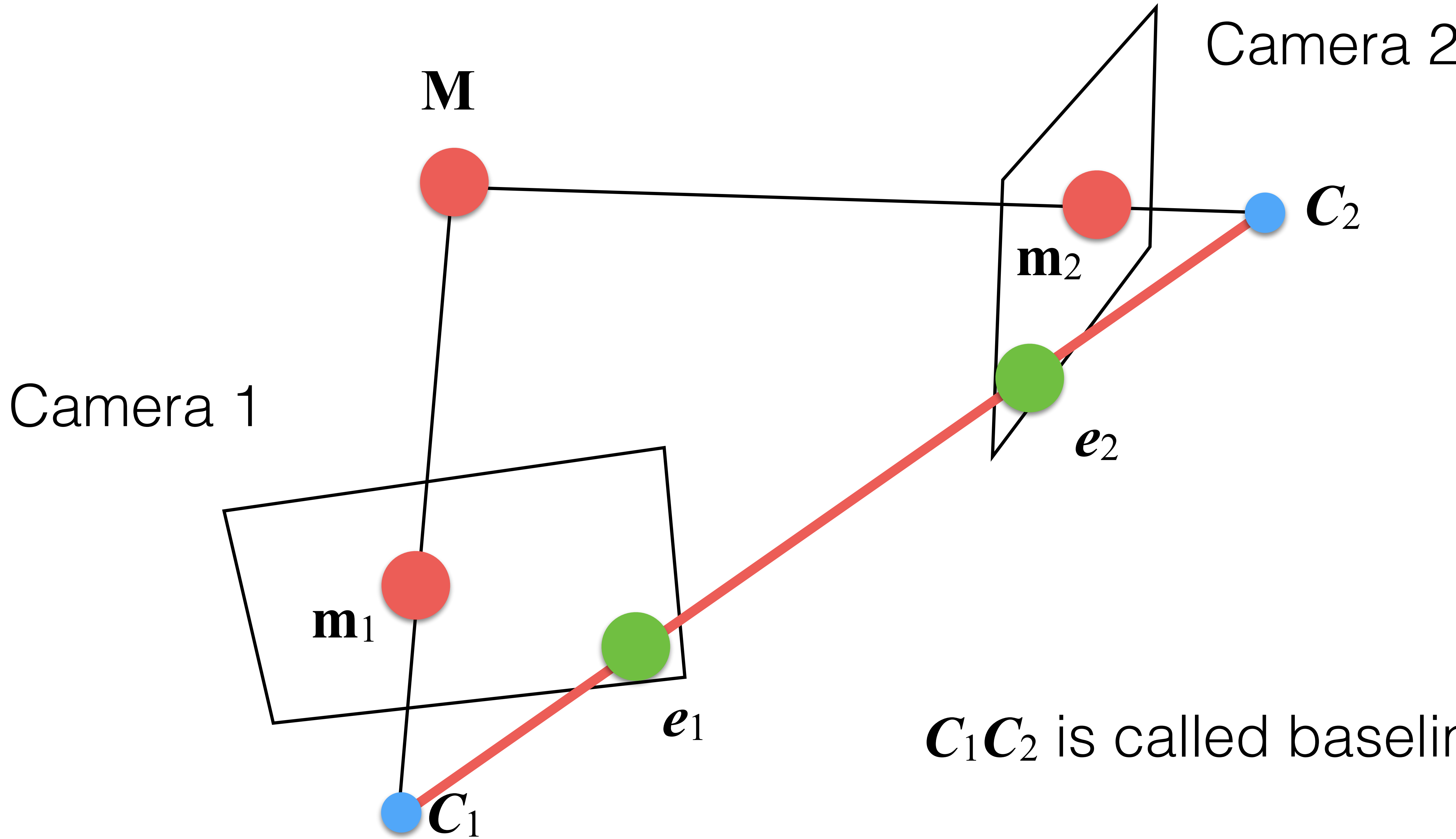
C_1C_2 is called baseline

Epipolar Geometry



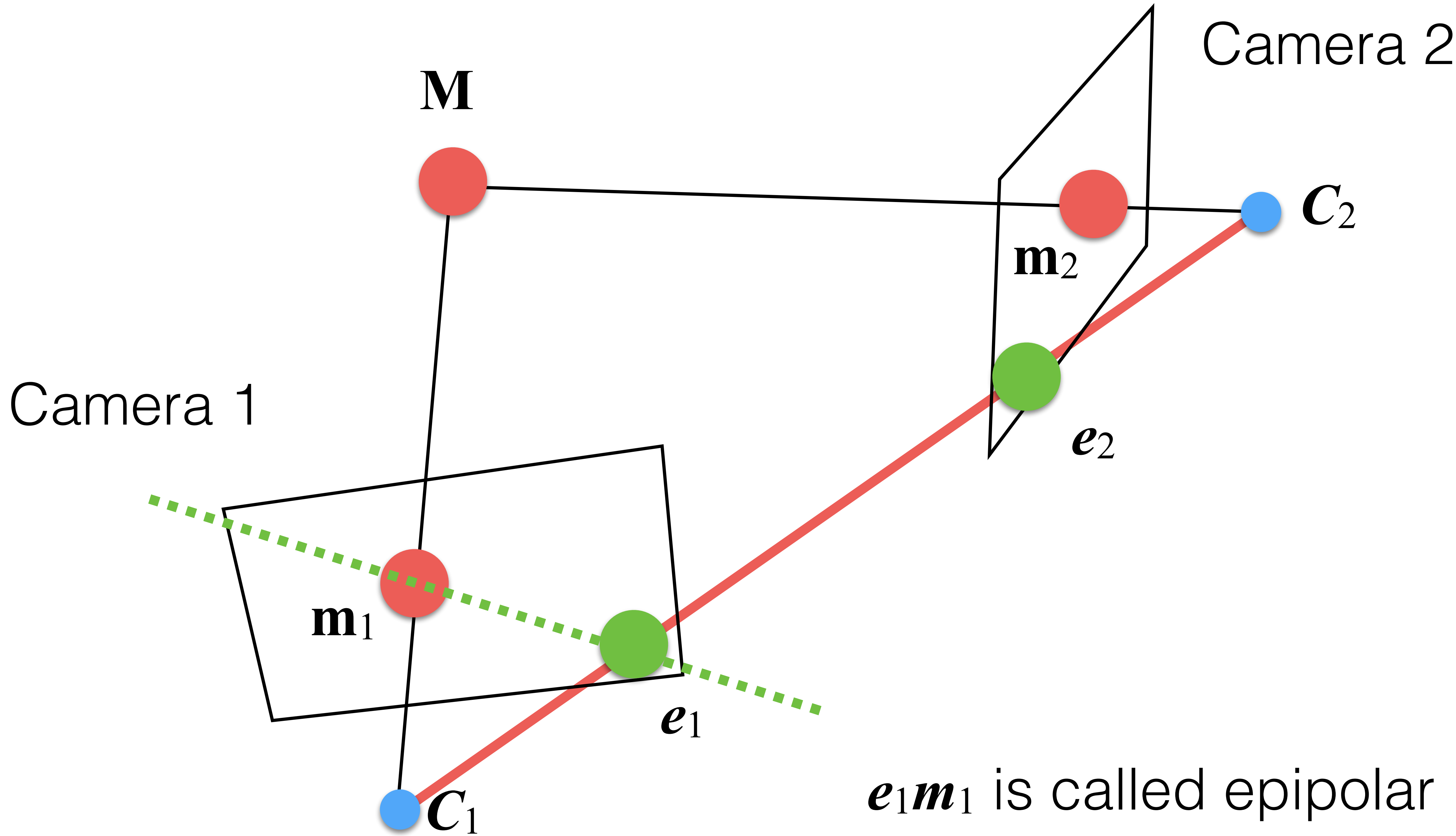
C_1C_2 is called baseline

Epipolar Geometry



C_1C_2 is called baseline

Epipolar Geometry



$e_1 m_1$ is called epipolar line

Epipolar Geometry: Epipoles

- An epipole is the intersection of the baseline with the two image planes.

- Therefore, it is defined as

$$\mathbf{e}_1 \sim P_1 \cdot C_1$$
$$\mathbf{e}_2 \sim P_2 \cdot C_2$$

- Note that, the projection matrices can be viewed as:

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

Epipolar Geometry: Epipolar Lines

- The fundamental matrix can be also defined as

$$F = [e_1]_\times \cdot Q_1 \cdot Q_2^{-1}$$

- and recalling

$$\mathbf{l} = F \cdot \mathbf{m}_1 \leftrightarrow (l_1 x + l_2 y + l_3) = 0$$

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

- we have:

$$\mathbf{m}_1 \sim (Q_1 \cdot Q_2^{-1} \cdot \mathbf{m}_2) \cdot t + \mathbf{e}_1$$

$$\mathbf{m}_2 \sim (Q_2 \cdot Q_1^{-1} \cdot \mathbf{m}_1) \cdot t + \mathbf{e}_2$$

Epipolar Geometry: Epipolar Lines

- This equation is very important because it implies that:
  - If we have a point $\mathbf{m}_2$ in image I_2, its match, $\mathbf{m}_1$, is located in image I_1 along that line!
    - This means that we need to find the match along a line! 1D search instead of a 2D search around the whole image!

Epipolar Geometry: Example

Left (I_1)

Right (I_2)

Epipolar Geometry: Example 1

Left (I_1)

Epipolar Geometry: Example 1

Right (I_2)

Epipolar Geometry: Example 2

Left (I_1)

Epipolar Geometry: Example 2

Right (I_2)

Epipolar Geometry: Example 2

Left (I_1)

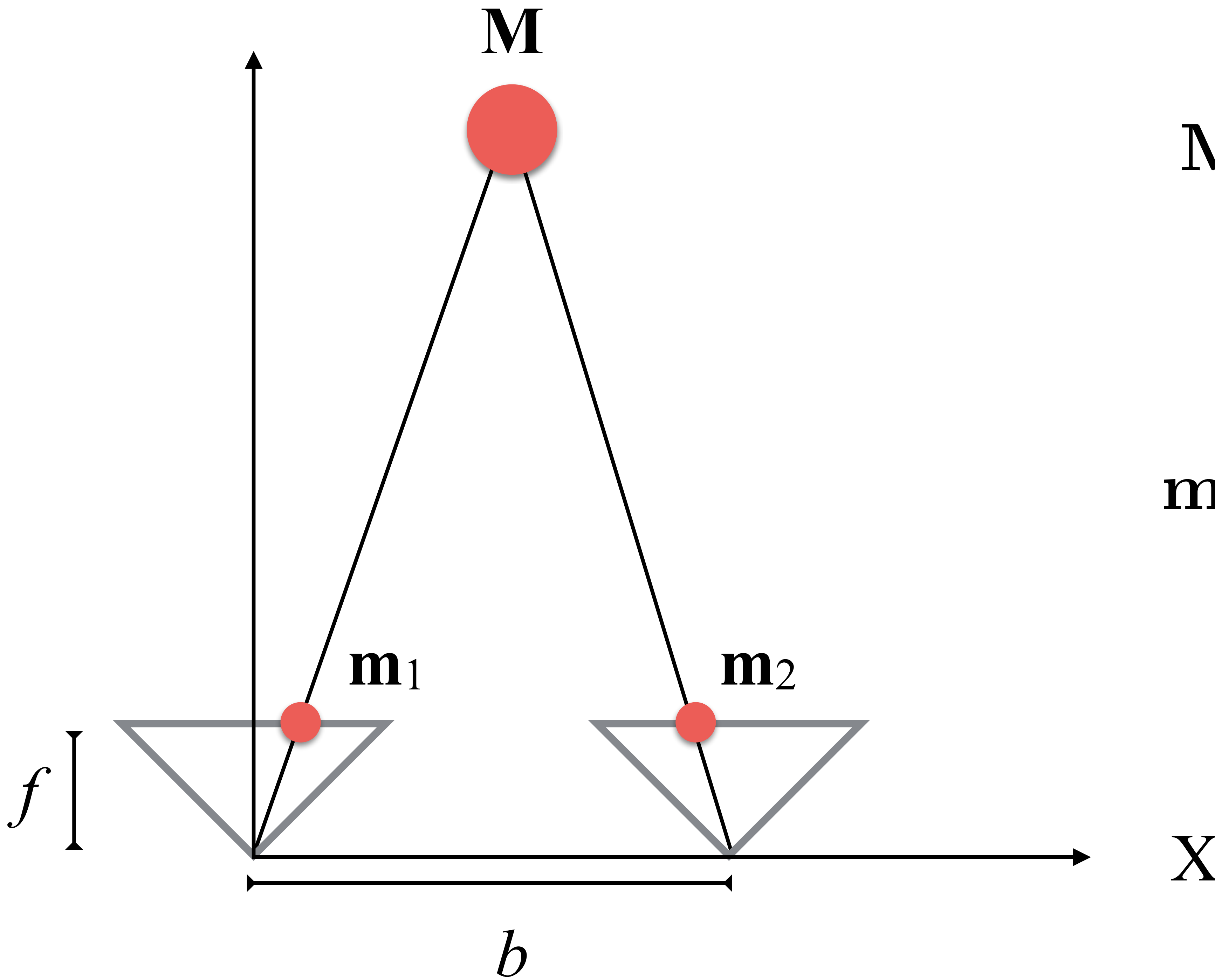
Epipolar Geometry Example

Right (I_2)

Epipolar Geometry: Epipolar Lines

- So do we search along that line?
- Not really, it is not very computationally efficient:
    - At each check we need to apply bilinear interpolation and to compute pixel coordinates.

Epipolar Geometry: Ideal Case



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

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

Epipolar Geometry: Ideal Case

Left

Right

Images from the Middlebury Dataset:
http://vision.middlebury.edu/stereo/data/scenes2005/

Epipolar Geometry: Ideal Case

Left

Right

Images from the Middlebury Dataset:
http://vision.middlebury.edu/stereo/data/scenes2005/

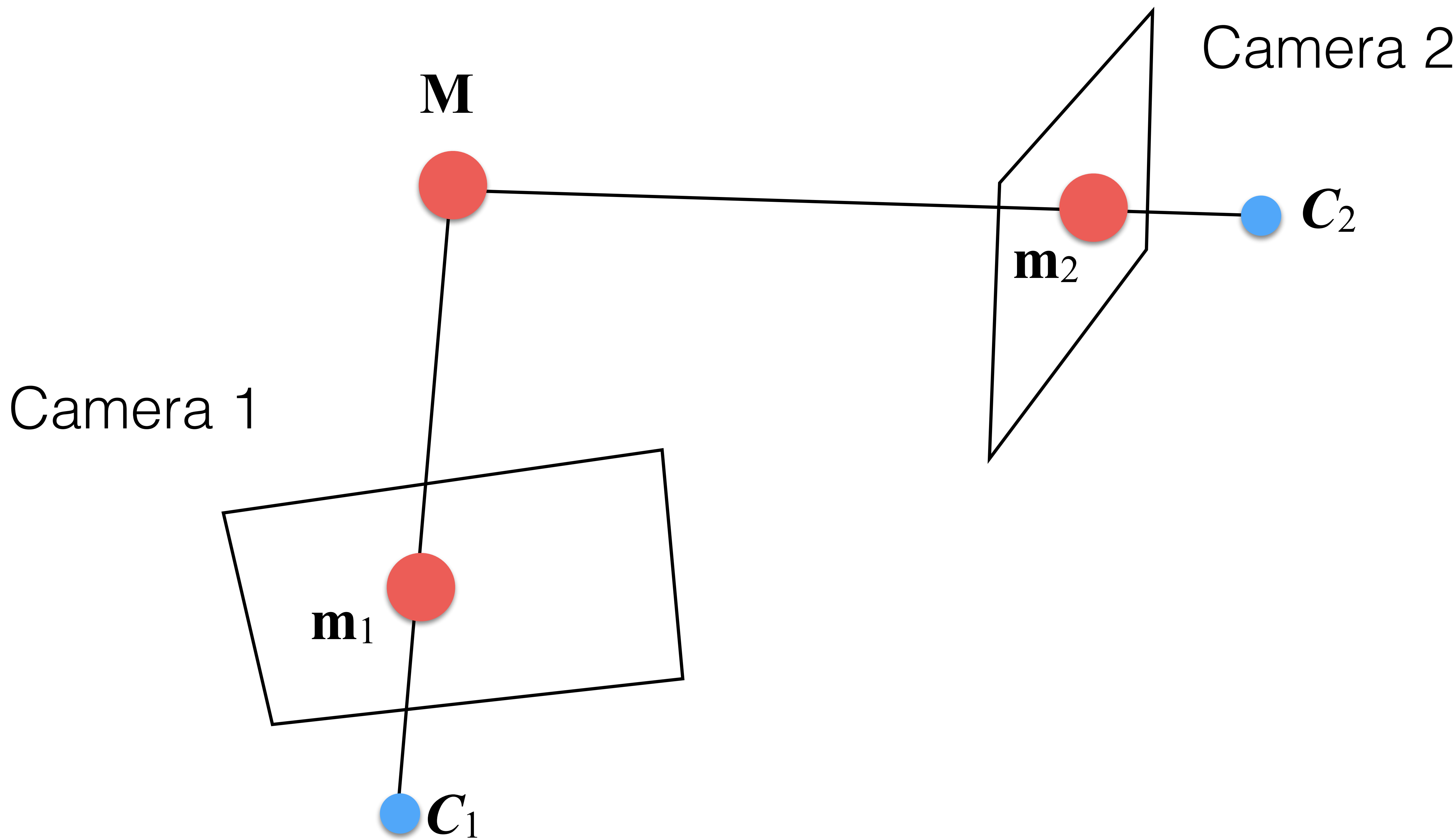
Epipolar Geometry: Ideal Case

Left

Right

Images from the Middlebury Dataset:
http://vision.middlebury.edu/stereo/data/scenes2005/

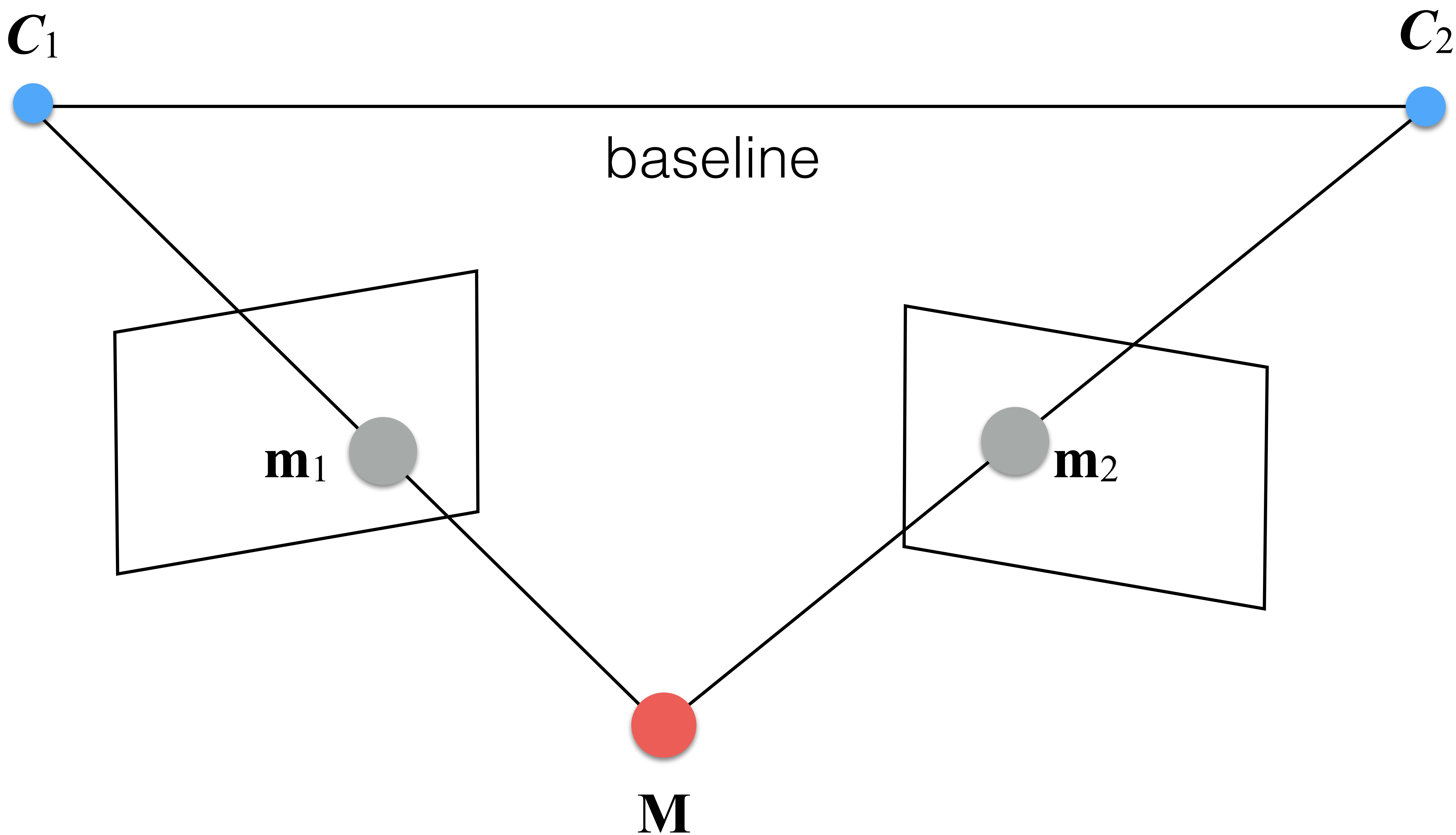
Epipolar Geometry: The General Case



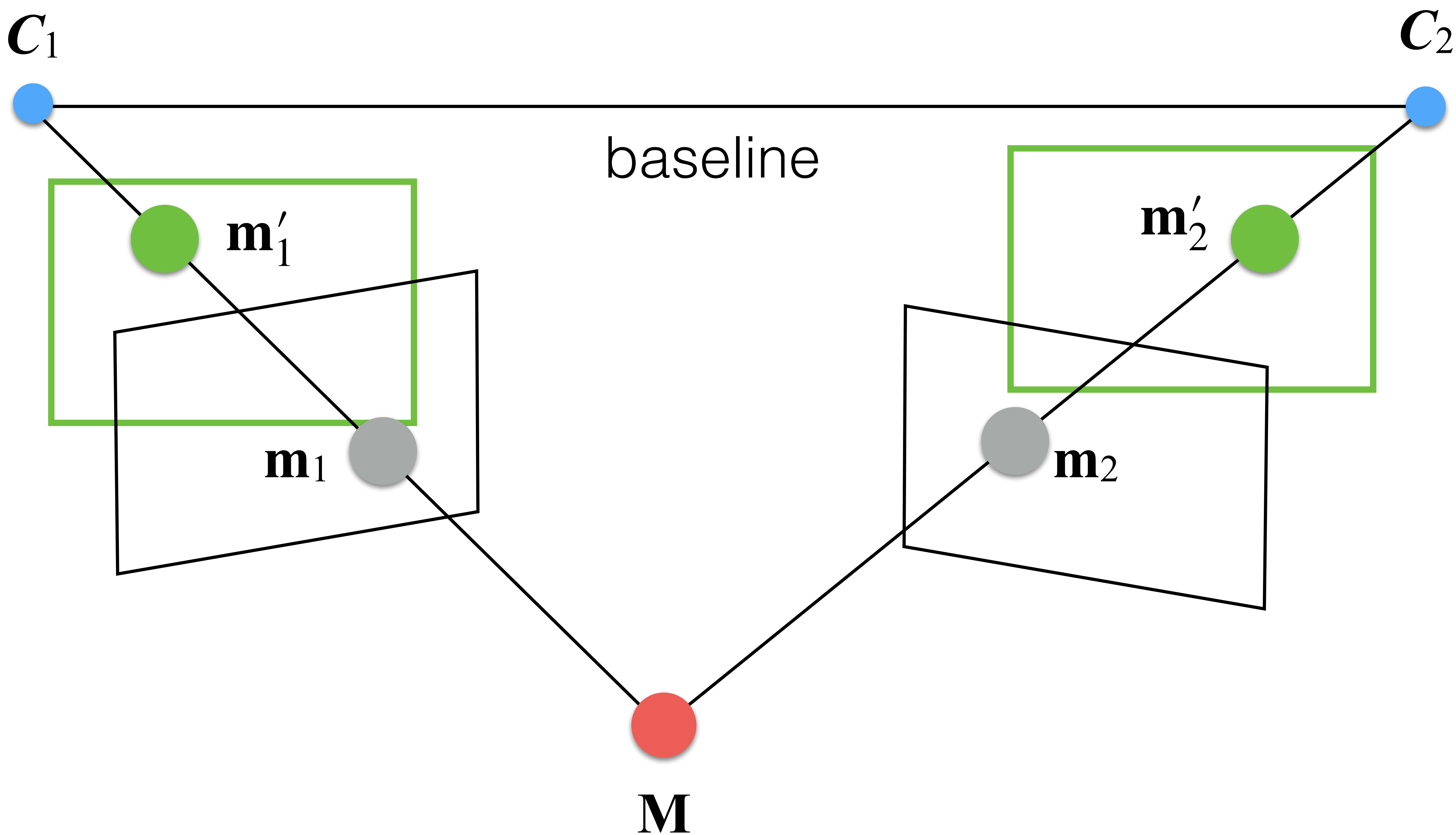
Epipolar Geometry: Rectification

- What do we need to do to transform the general case into the ideal one?
  - We keep the optical centers where they are.
  - We rotate both image planes to go back to the ideal case.

Epipolar Geometry: Rectification



Epipolar Geometry: Rectification



Epipolar Geometry: Rectification

- So we need to modify P_1 and P_2. Both matrices can be defined as

$$P_1 = K_1 \cdot [R_1 | - R_1 \cdot \mathbf{C}_1]$$
$$P_2 = K_2 \cdot [R_2 | - R_2 \cdot \mathbf{C}_2]$$

- where

$$\mathbf{C}_i = -Q_i^{-1} \cdot \mathbf{q}_i \qquad P_i = [Q_i | \mathbf{q}_i]$$

Epipolar Geometry: Rectification

- So we need to compute a new R' matrix for both cameras.
- Let's see the process for a generic camera with a starting rotation matrix R:

$$R = [\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3].$$

- Given that the optical centers do not move (they define the X -axis), the X-axis or $\mathbf{r}'_1$ is:

$$\mathbf{r}'_1 = \frac{\mathbf{C}_1 - \mathbf{C}_2}{\|\mathbf{C}_1 - \mathbf{C}_2\|}$$

Epipolar Geometry: Rectification

- The new Y-axis or $\mathbf{r}_2'$ is defined as:

$$\mathbf{r}_2' = \mathbf{r}_3 \times \mathbf{r}_1'.$$

- $\mathbf{r}_3$ is the old Z-axis vector. Why? The camera is still looking towards the same direction.

- The new Z-axis is obviously computed as the cross product of the twos:

$$\mathbf{r}_3' = \mathbf{r}_1' \times \mathbf{r}_2'.$$

Epipolar Geometry: Rectification

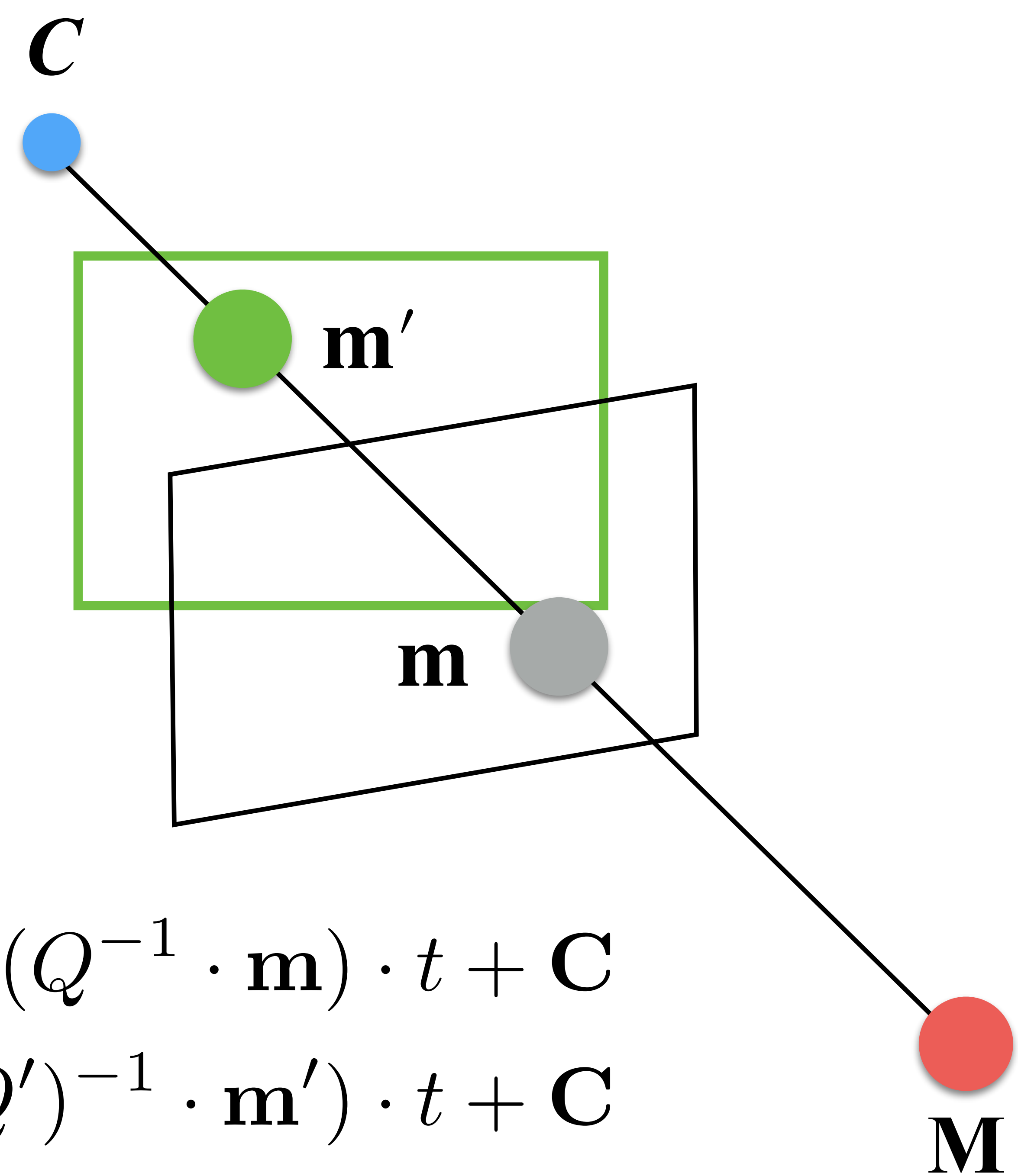
- Once we computed the new P' for a view (i.e., a new R), we need to compute the transform from P to P'. We know that:

$$\mathbf{m} \sim P \cdot \mathbf{M}$$
$$\mathbf{m}' \sim P' \cdot \mathbf{M}$$

- and that:

$$\mathbf{M} = (Q^{-1} \cdot \mathbf{m}) \cdot t + \mathbf{C}$$
$$\mathbf{M} = ((Q')^{-1} \cdot \mathbf{m}') \cdot t + \mathbf{C}$$

Epipolar Geometry: Rectification



$$\mathbf{M} = (Q^{-1} \cdot \mathbf{m}) \cdot t + \mathbf{C}$$

$$\mathbf{M} = ((Q')^{-1} \cdot \mathbf{m}') \cdot t + \mathbf{C}$$

Epipolar Geometry: Rectification

$$\mathbf{m} \sim P \cdot \mathbf{M} \qquad \mathbf{M} = (Q^{-1} \cdot \mathbf{m}) \cdot t + \mathbf{C}$$

$$\mathbf{m}' \sim P' \cdot \mathbf{M} \qquad \mathbf{M} = ((Q')^{-1} \cdot \mathbf{m}') \cdot t + \mathbf{C}$$

Epipolar Geometry: Rectification

$$\mathbf{m} \sim P \cdot \mathbf{M}$$

$$\mathbf{m}' \sim P' \cdot \mathbf{M}$$

$$\mathbf{M} = (Q^{-1} \cdot \mathbf{m}) \cdot t + \mathbf{C}$$

$$\mathbf{M} = ((Q')^{-1} \cdot \mathbf{m}') \cdot t + \mathbf{C}$$

Epipolar Geometry: Rectification

$$\mathbf{m} \sim P \cdot \mathbf{M}$$

$$\mathbf{m}' \sim P' \cdot \mathbf{M}$$

$$\mathbf{M} = (Q^{-1} \cdot \mathbf{m}) \cdot t + \mathbf{C}$$

$$\mathbf{M} = ((Q')^{-1} \cdot \mathbf{m}') \cdot t + \mathbf{C}$$

$$\mathbf{m}' \sim Q' \cdot Q^{-1} \cdot \mathbf{m}$$

Epipolar Geometry: Rectification

$$\mathbf{m} \sim P \cdot \mathbf{M}$$

$$\mathbf{m}' \sim P' \cdot \mathbf{M}$$

$$\mathbf{M} = (Q^{-1} \cdot \mathbf{m}) \cdot t + \mathbf{C}$$

$$\mathbf{M} = ((Q')^{-1} \cdot \mathbf{m}') \cdot t + \mathbf{C}$$

$$\mathbf{m}' \sim Q' \cdot Q^{-1} \cdot \mathbf{m}$$

Epipolar Geometry: Rectification

$$\mathbf{m} \sim P \cdot \mathbf{M} \qquad \mathbf{M} = (Q^{-1} \cdot \mathbf{m}) \cdot t + \mathbf{C}$$

$$\mathbf{m}' \sim P' \cdot \mathbf{M} \qquad \mathbf{M} = ((Q')^{-1} \cdot \mathbf{m}') \cdot t + \mathbf{C}$$

$$\mathbf{m}' \sim Q' \cdot Q^{-1} \cdot \mathbf{m}$$

$$\mathbf{m}' \sim T \cdot \mathbf{m} \qquad T = Q' \cdot Q^{-1}$$

Epipolar Geometry: Rectification Example

Left

Right

Epipolar Geometry: Rectification Example

Left

Right

Dense Matching

- For each pixel at coordinate $[x, y]$ in the left/right image, we extract a patch p_1 of size $n \times n$.
- Then, we look along the horizontal line at height y in the other image the patch p_2 that is closest to p_1.

Dense Matching

Left

Dense Matching

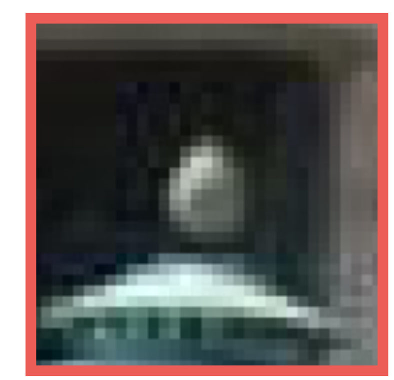
Left

Dense Matching

Left

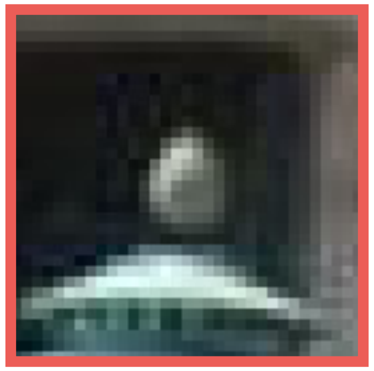
Dense Matching

Left



Extracted Patch (Left Image)

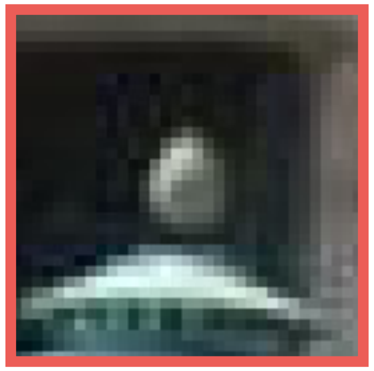
Dense Matching



Extracted
Patch
(Left Image)

Right

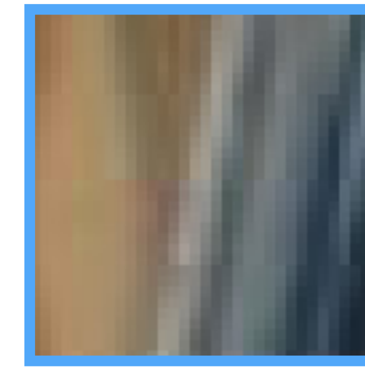
Dense Matching

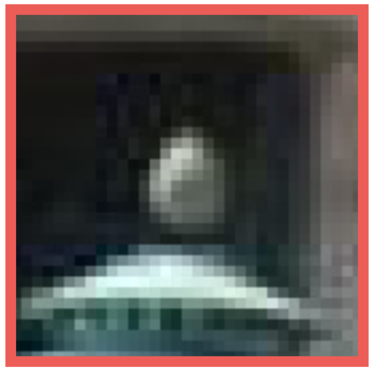


Extracted
Patch
(Left Image)

Right

Dense Matching

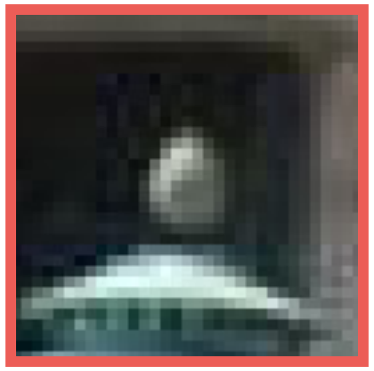




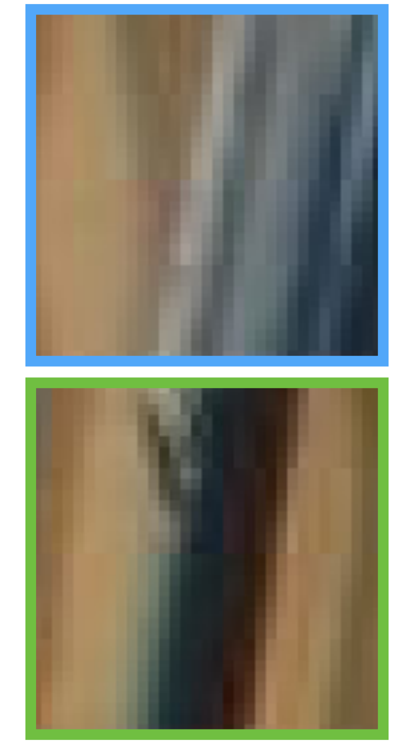
Extracted Patch (Left Image)

Right

Dense Matching

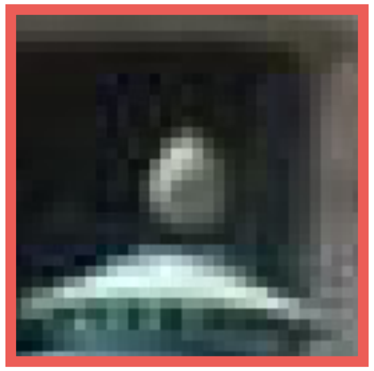


Extracted
Patch
(Left Image)



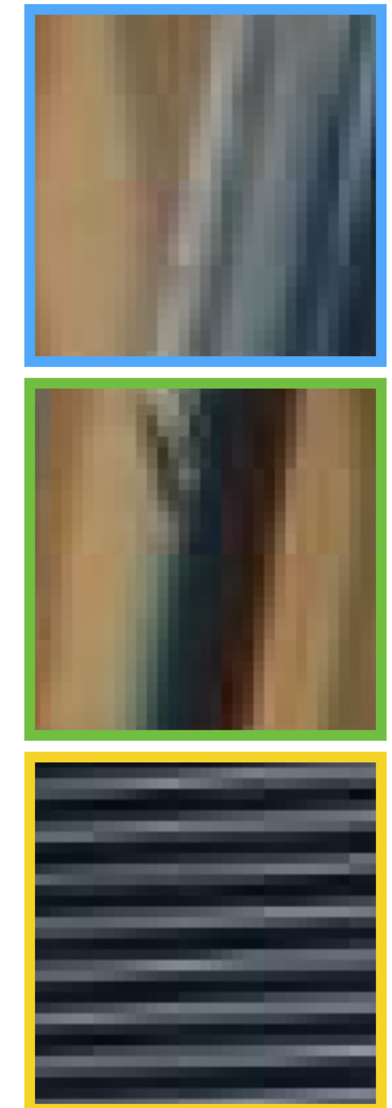
Right

Dense Matching

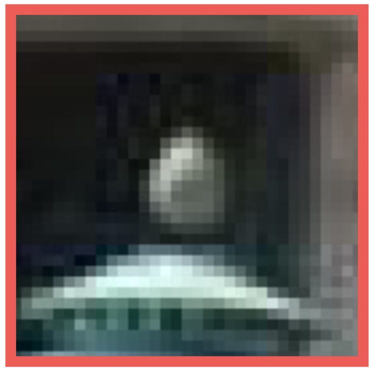


Extracted Patch (Left Image)

Right

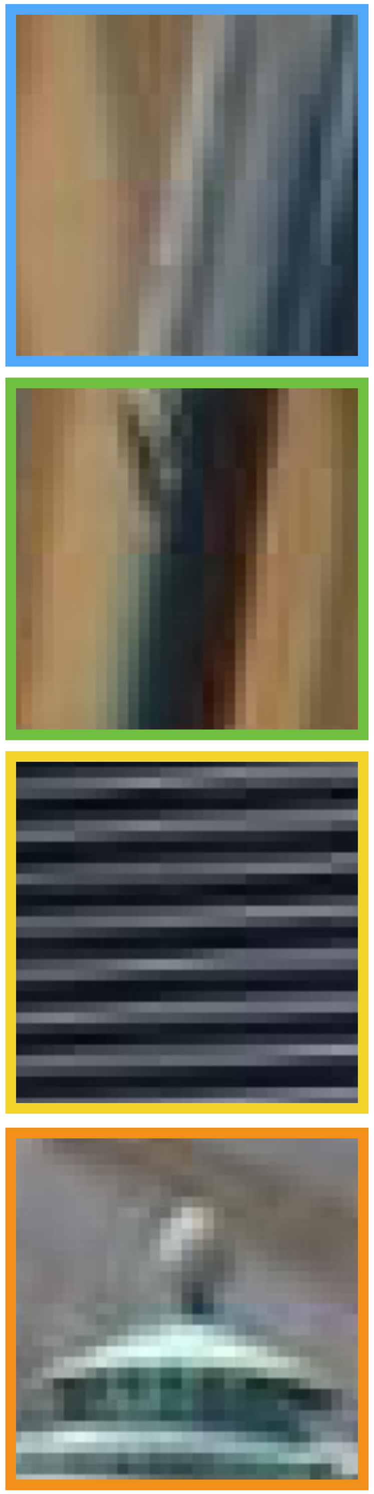


Dense Matching

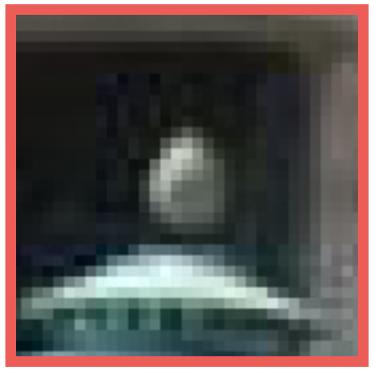


Extracted Patch (Left Image)

Right

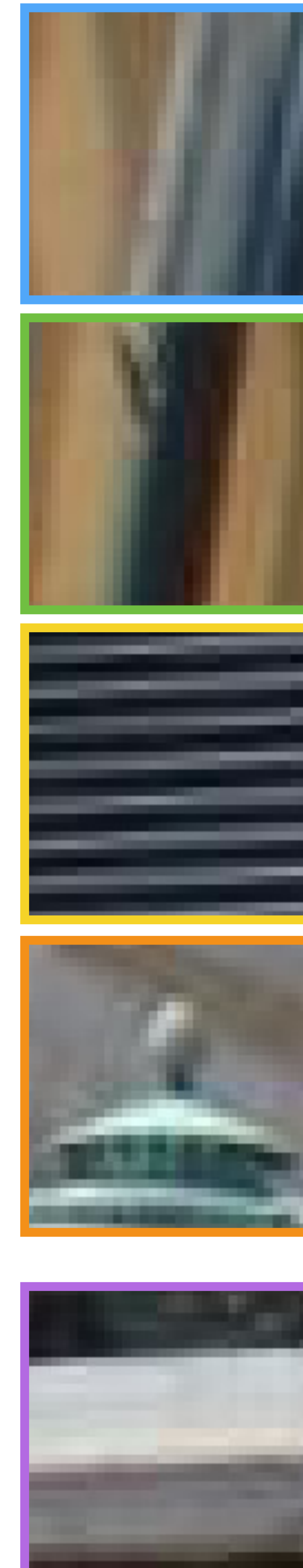


Dense Matching



Extracted Patch (Left Image)

Right



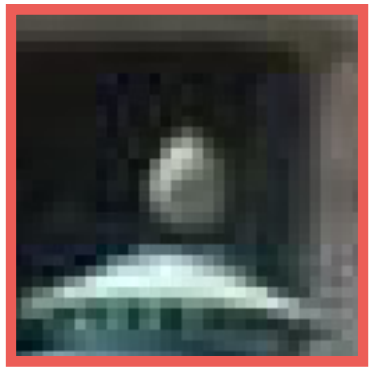
Dense Matching

Right

Dense Matching

Right

Dense Matching



Extracted Patch (Left Image)

Right

Dense Matching

- How do we compute if a patch is closer than another?

$$SSD(p_1, p_2) = \sum_{i=1}^{n} \sum_{j=1}^{n} \|p_1(i,j) - p_2(i,j)\|^2$$

$$SAD(p_1, p_2) = \sum_{i=1}^{n} \sum_{j=1}^{n} |p_1(i,j) - p_2(i,j)|$$

- We are looking for the closest, so for SSD and SAD the lower the closer.

Dense Matching

- There are many other metrics such as normalized cross correlation, zero mean normalized cross correlation, etc.

- To improve the matching quality, we can compute descriptors for each pixel:

    - Computationally expensive, typically not done!

Dense Matching

- In practice, for dense matching, we do not extract patches in an explicit way.

- We formalize the problem as an energy minimization problem:

$$\arg\min_{d} E(x, y, I_1, I_2, d)$$

- In the case of SAD, E is defined as:

$$E(x, y, d) = \sum_{i=1}^{n} \sum_{j=1}^{n} \big| I_1(x + i, y + j) - I_2(x + i + d, y + j) \big|$$

Dense Matching

- Note that $SAD(p_1, p_2) = \sum_{i=1}^{n} \sum_{j=1}^{n} |p_1(i,j) - p_2(i,j)|$ and

$$E(x, y, d) = \sum_{i=1}^{n} \sum_{j=1}^{n} \left| I_1(x+i, y+j) - I_2(x+i+d, y+j) \right|$$

are the same formulation when:

  - p_1 is extracted at (x, y) in I_1
  - p_2 is extracted at $(x+d, y)$ in I_2

Dense Matching

- When we minimize:

$$\arg\min_{d} E(x, y, I_1, I_2, d)$$

- We compute d, which is the ***disparity***. To compute the ***depth***, we need to apply:

$$z = \frac{b \cdot f}{d}$$

- where b is the baseline and f is the focal length.

- This is done for each pixel in the disparity map in order to obtain a depth map!

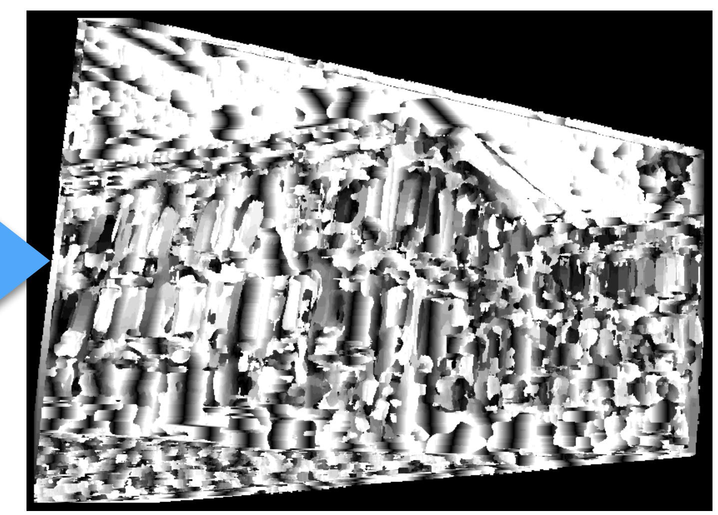
Dense Matching Example

Input

Left

Right

Output

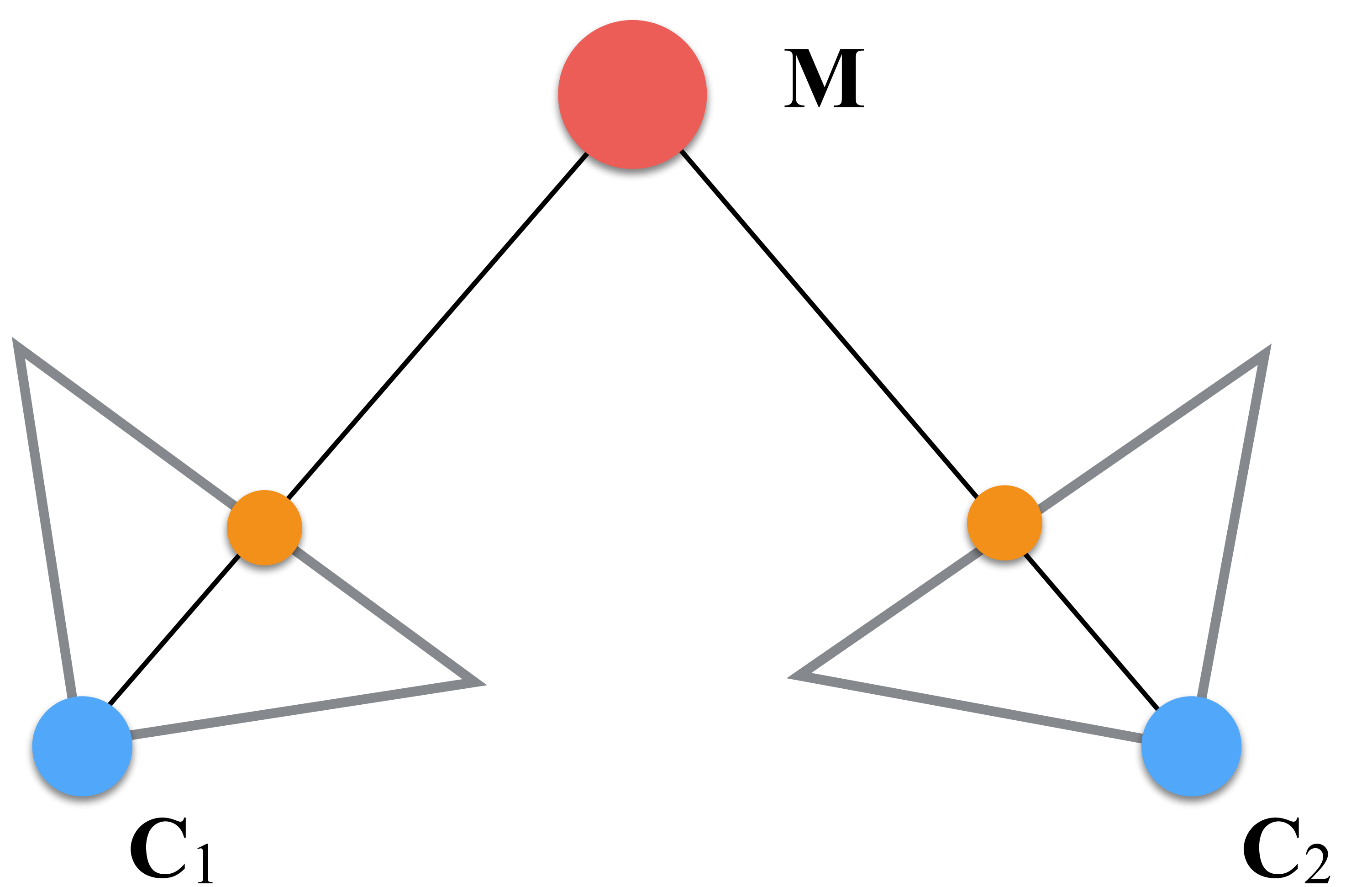


Disparity Map
for the Left Image

From previous example the disparity map is very noisy! How can we improve?

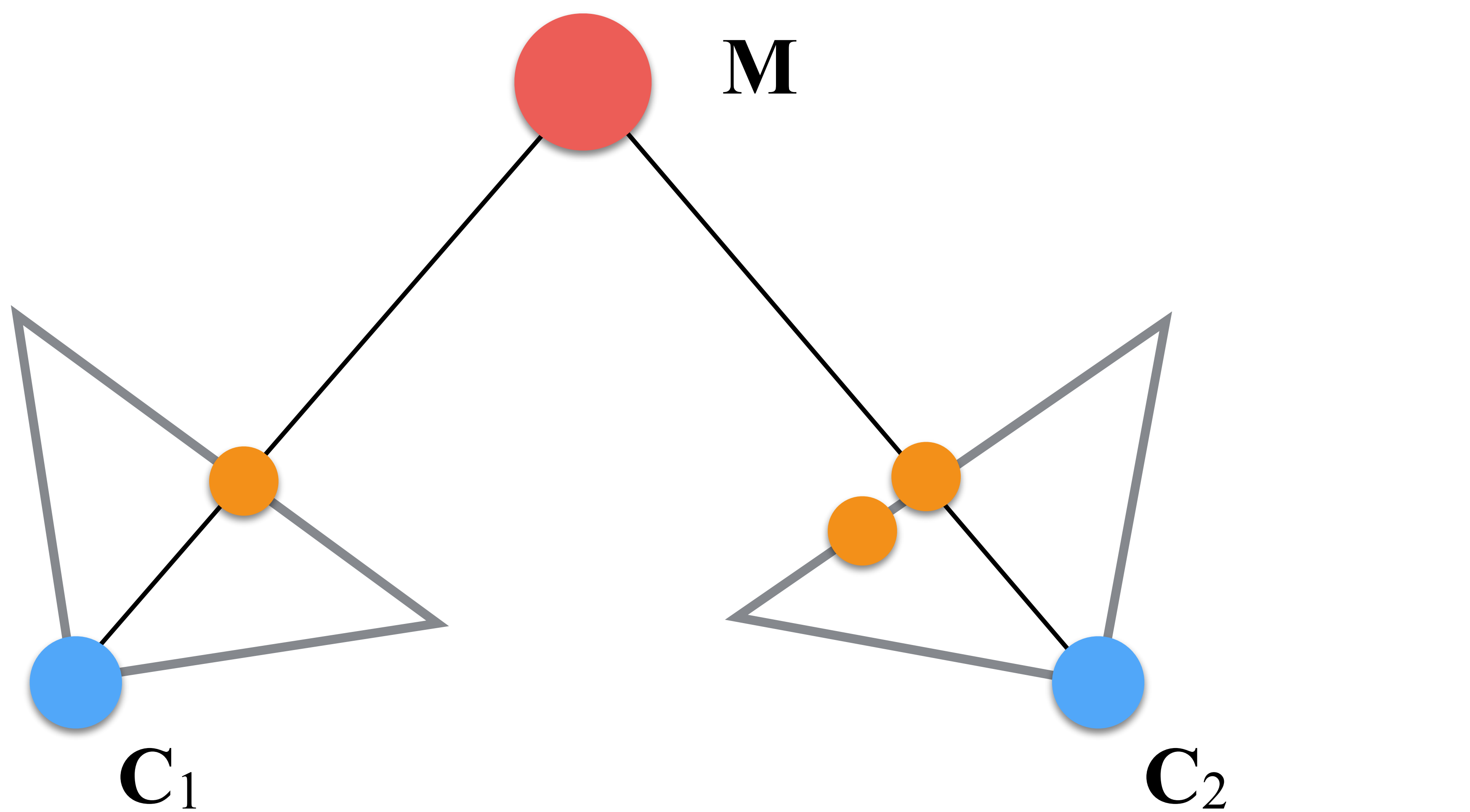
Non-Local Constraints: Uniqueness

- For each point in one image, there should be at maximum one matching point in the other:



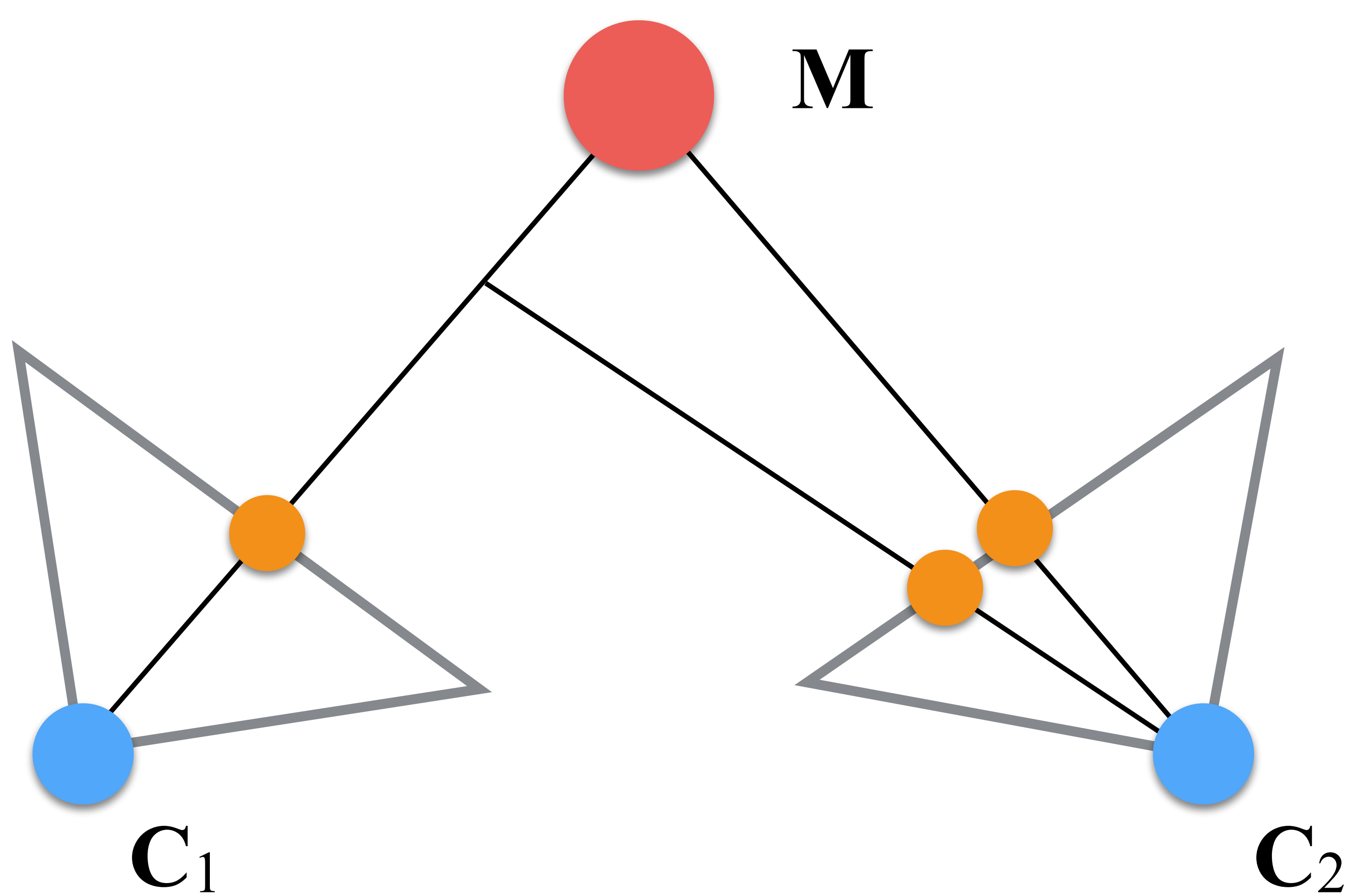
Non-Local Constraints: Uniqueness

- For each point in one image, there should be at maximum one matching point in the other:



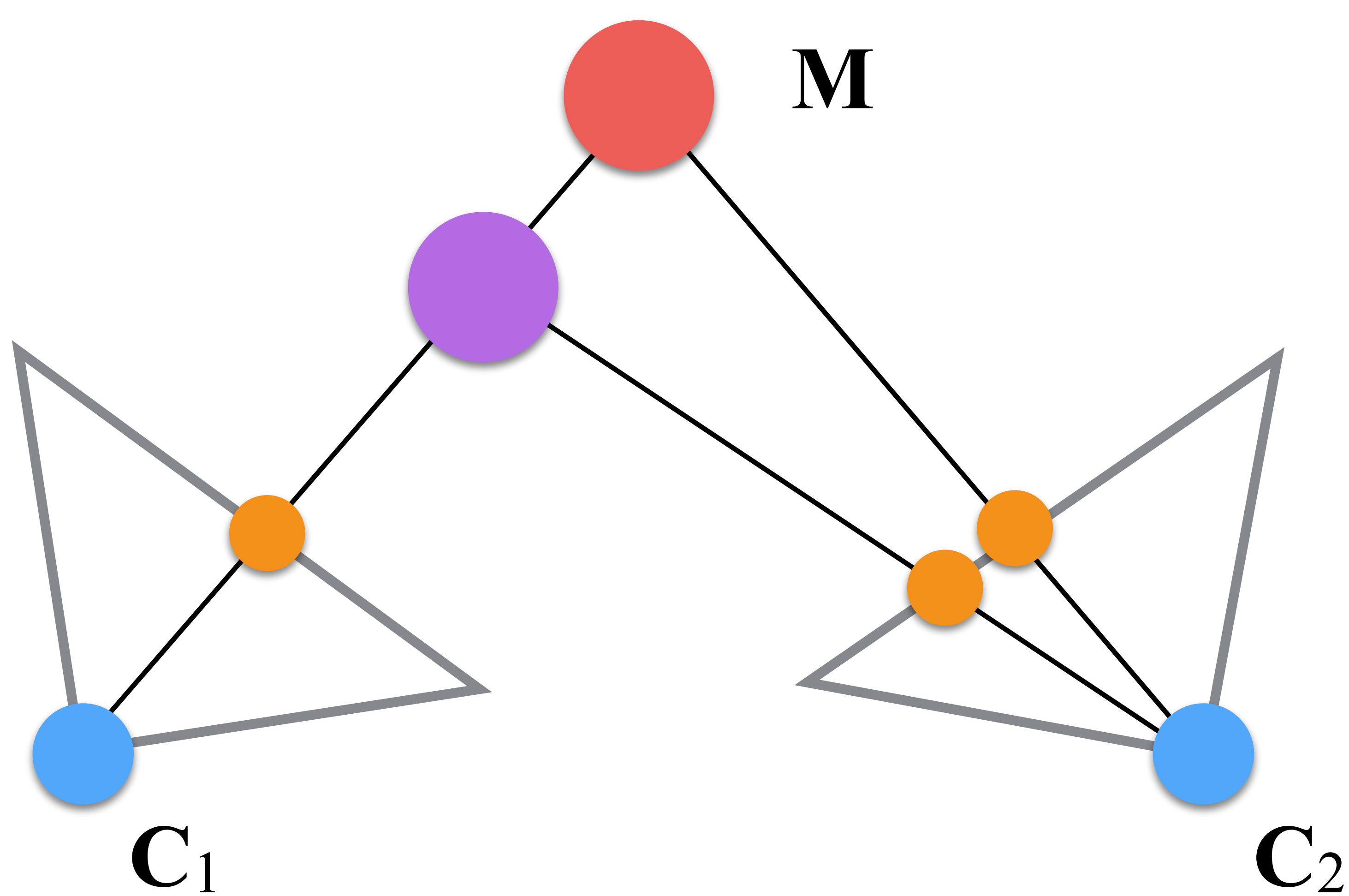
Non-Local Constraints: Uniqueness

- For each point in one image, there should be at maximum one matching point in the other:



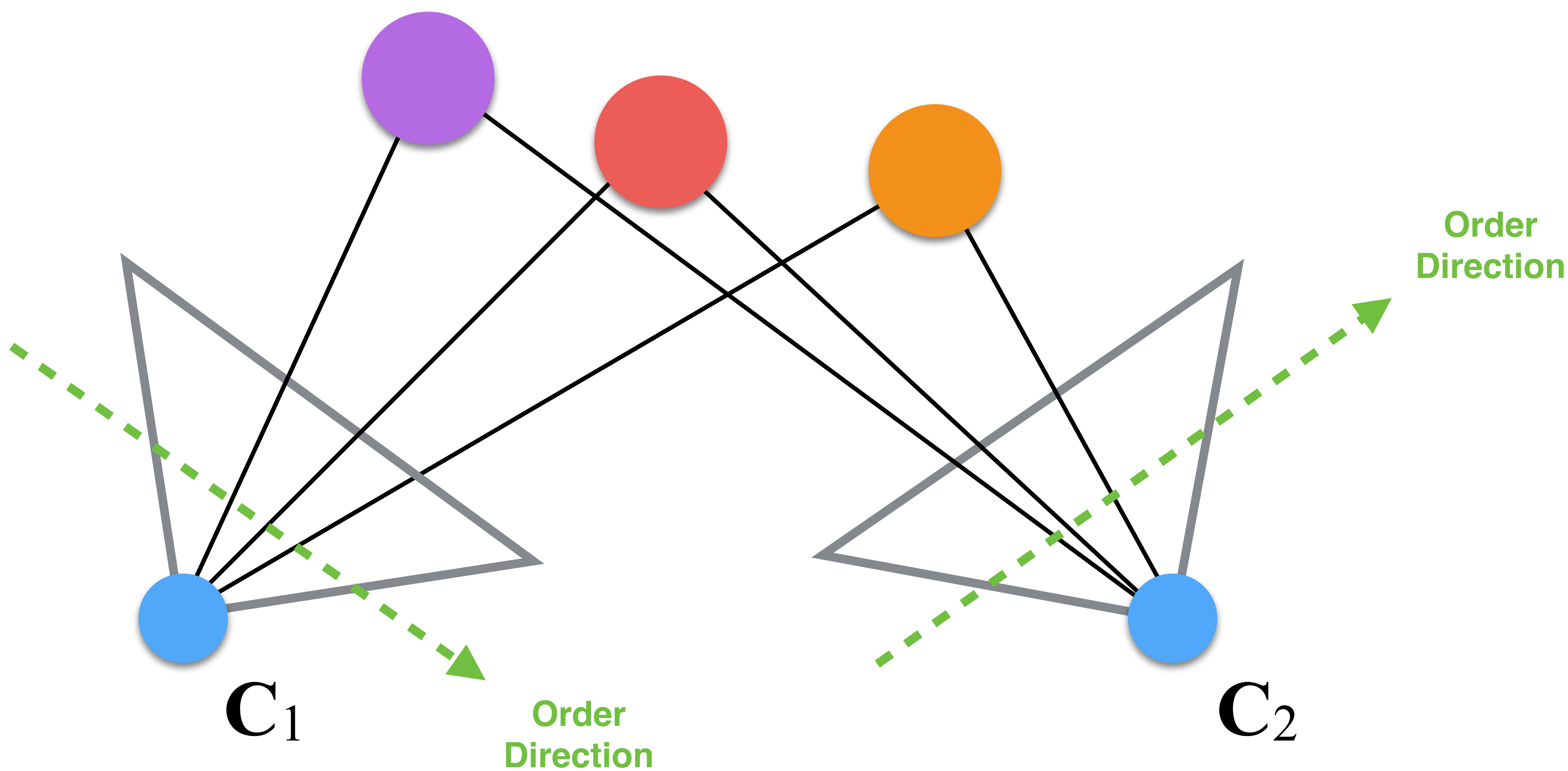
Non-Local Constraints: Uniqueness

- For each point in one image, there should be at maximum one matching point in the other:



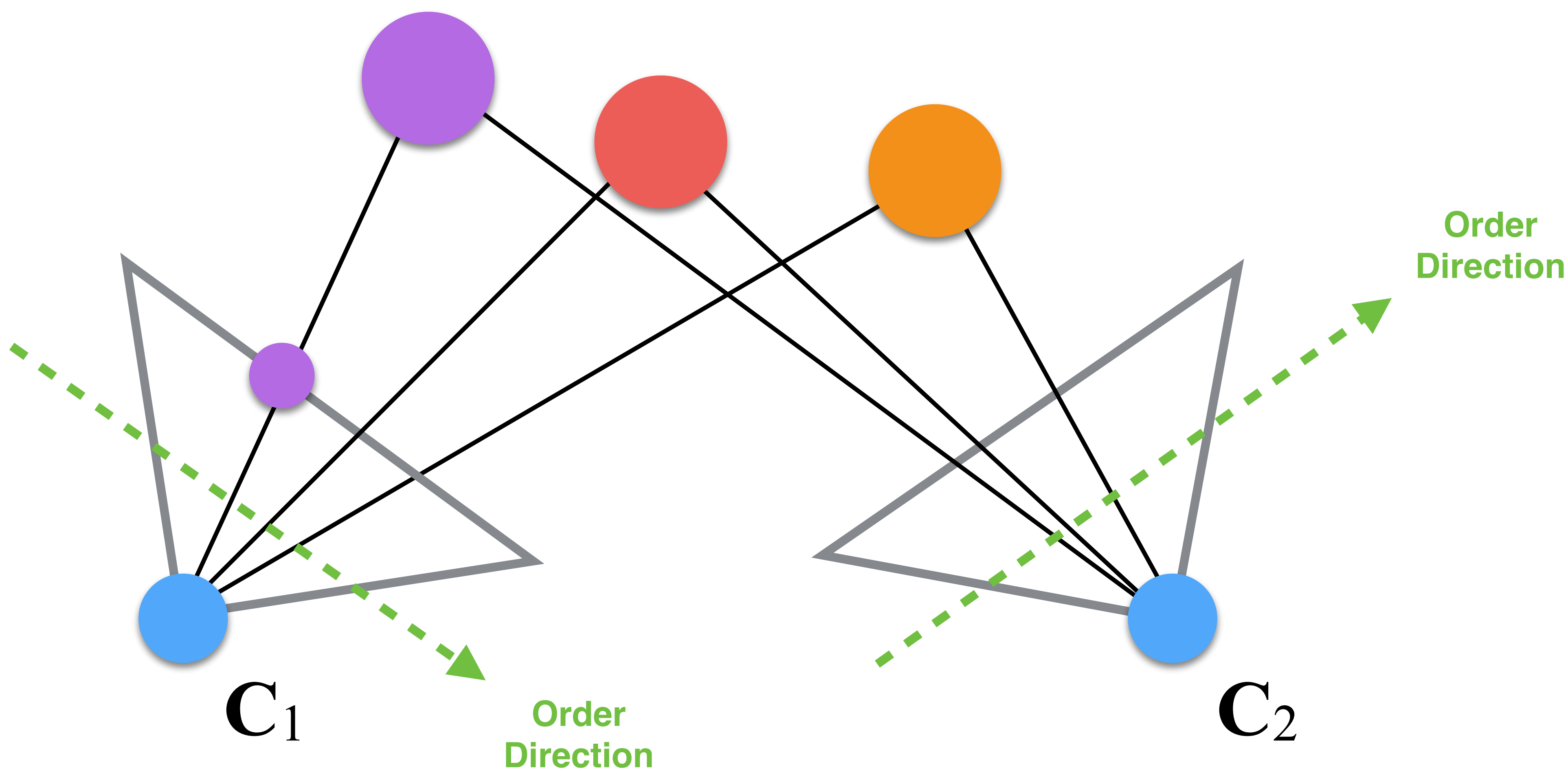
Non-Local Constraints: Correct Ordering

- Corresponding points should be in the same order in both views.



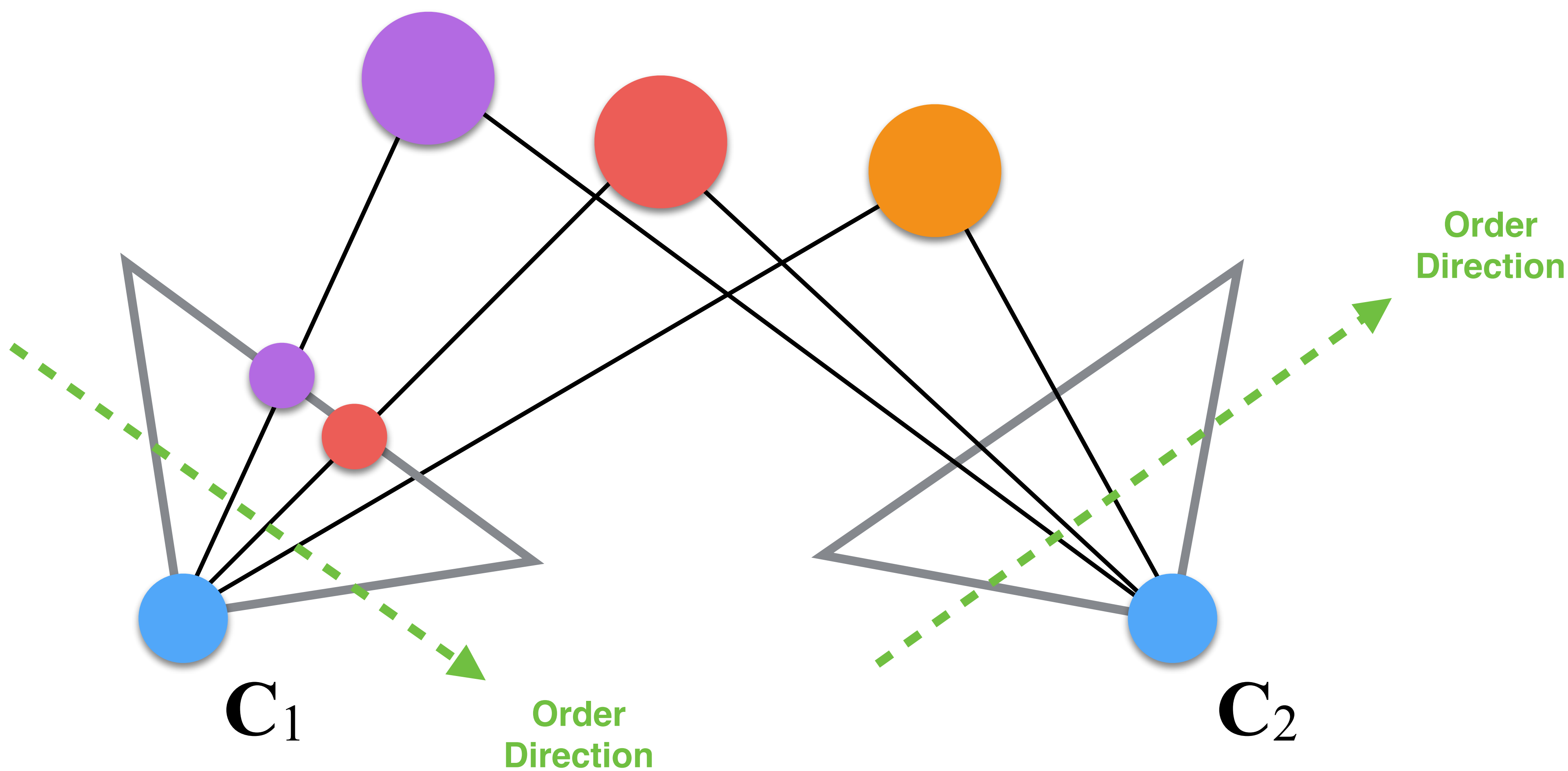
Non-Local Constraints: Correct Ordering

- Corresponding points should be in the same order in both views.



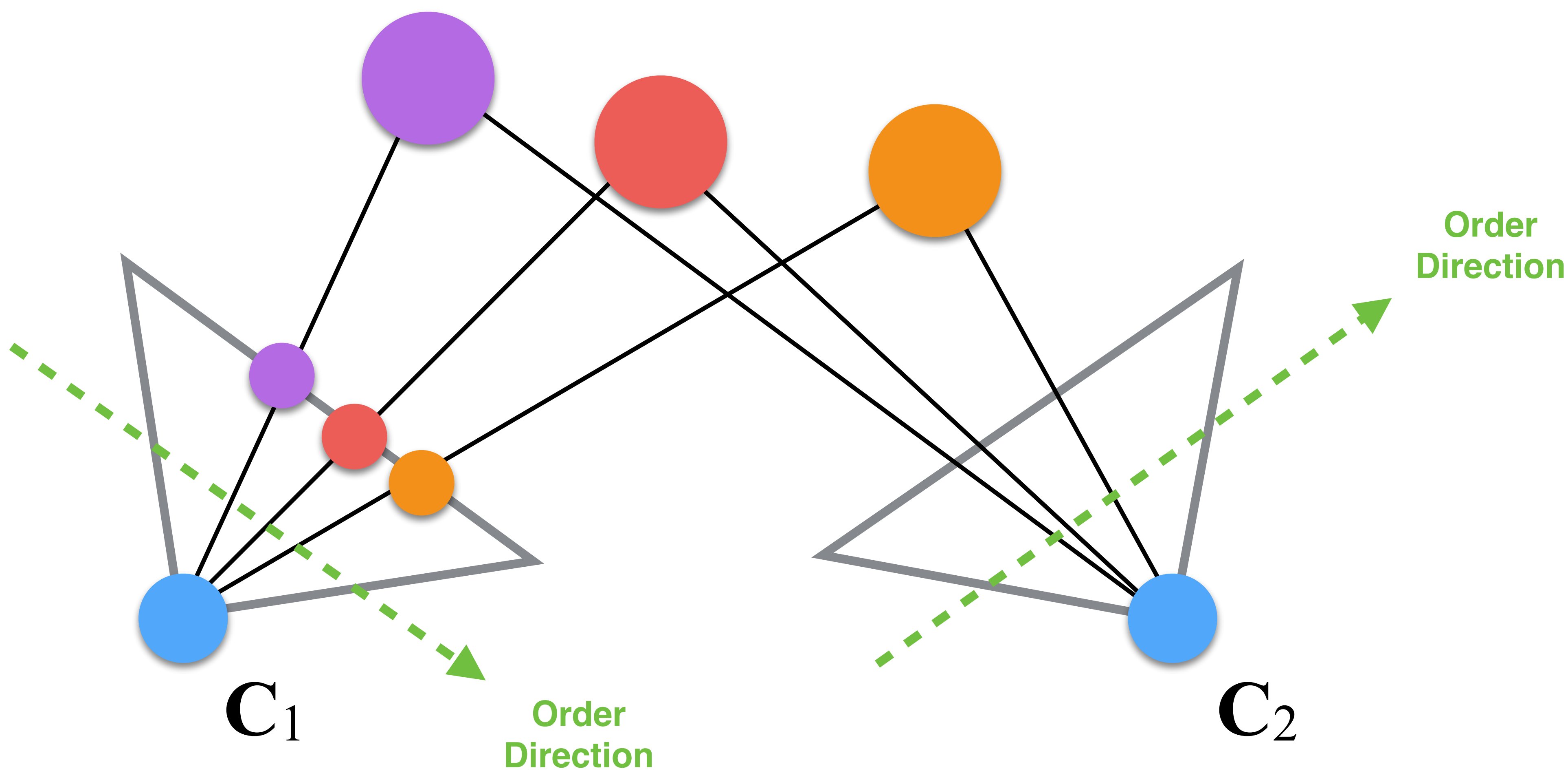
Non-Local Constraints: Correct Ordering

- Corresponding points should be in the same order in both views.



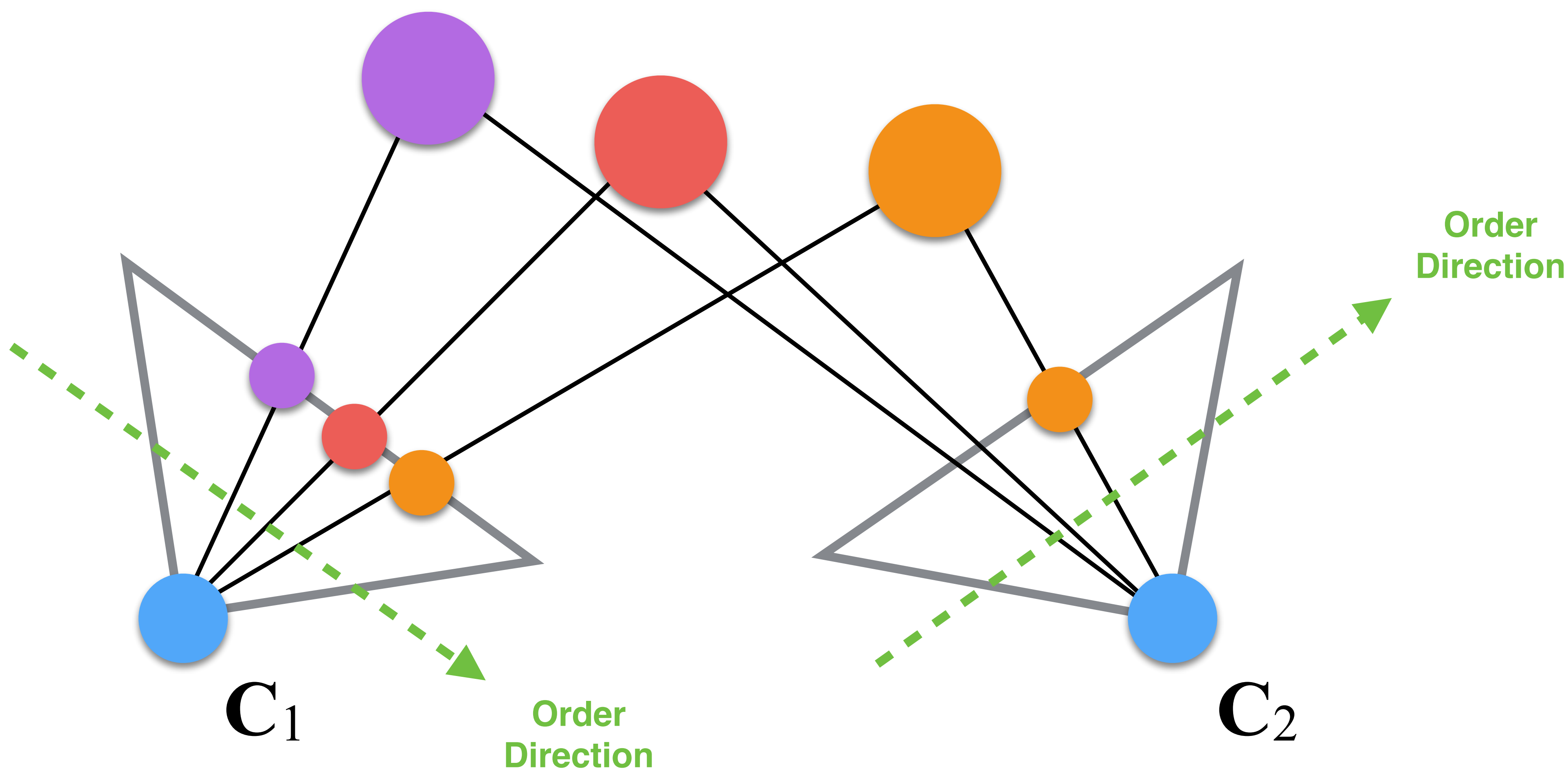
Non-Local Constraints: Correct Ordering

- Corresponding points should be in the same order in both views.



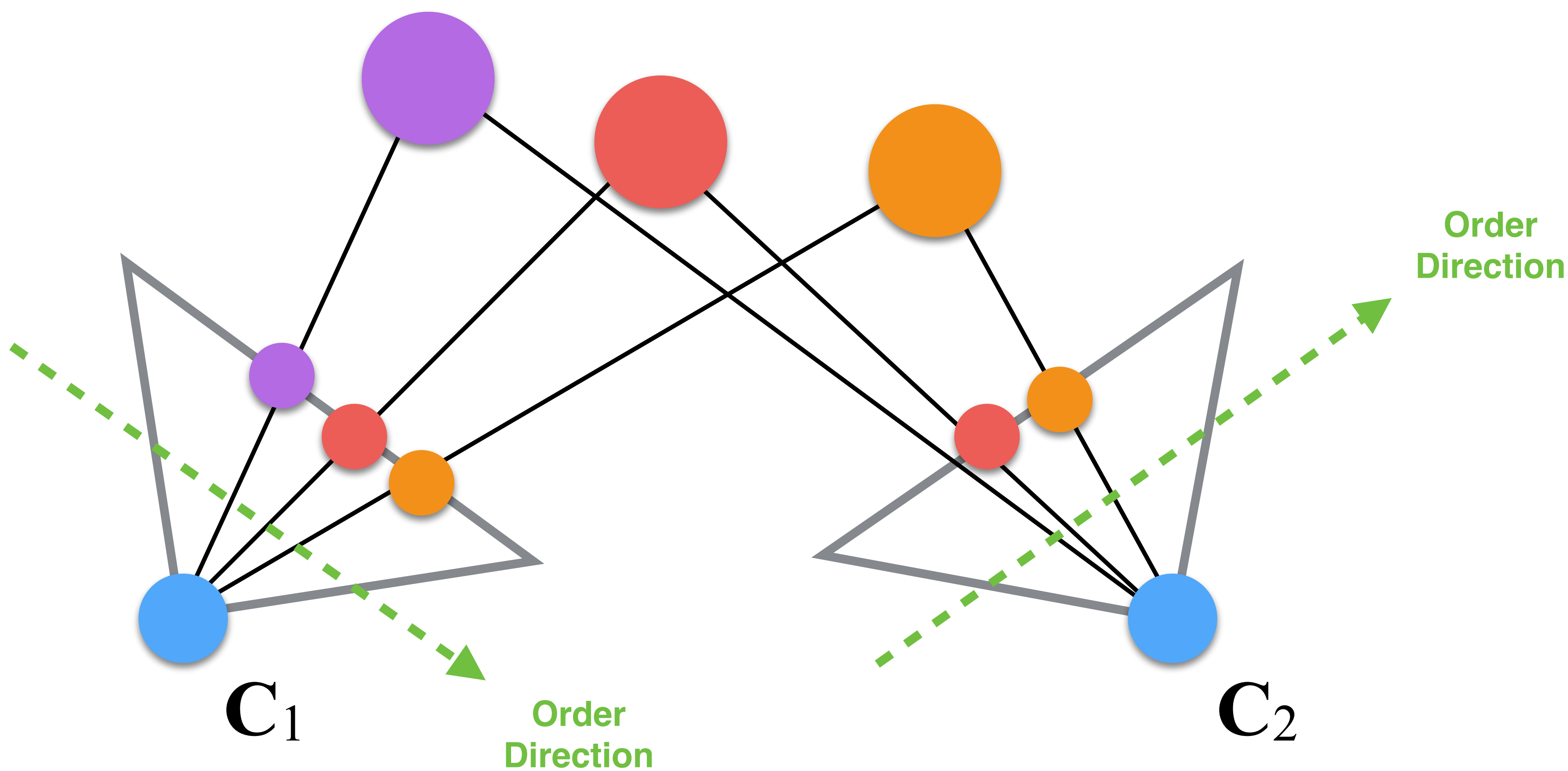
Non-Local Constraints: Correct Ordering

- Corresponding points should be in the same order in both views.



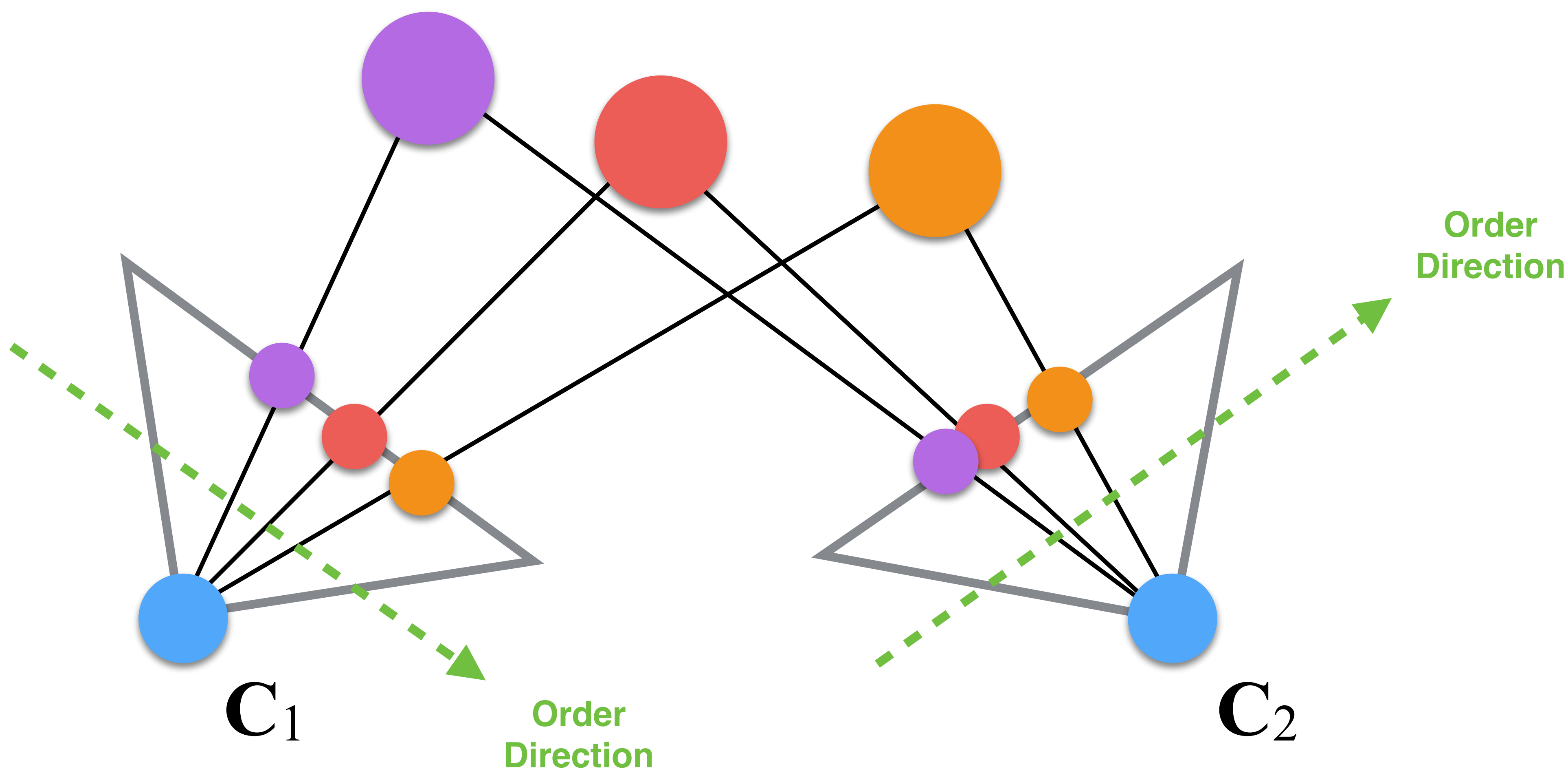
Non-Local Constraints: Correct Ordering

- Corresponding points should be in the same order in both views.

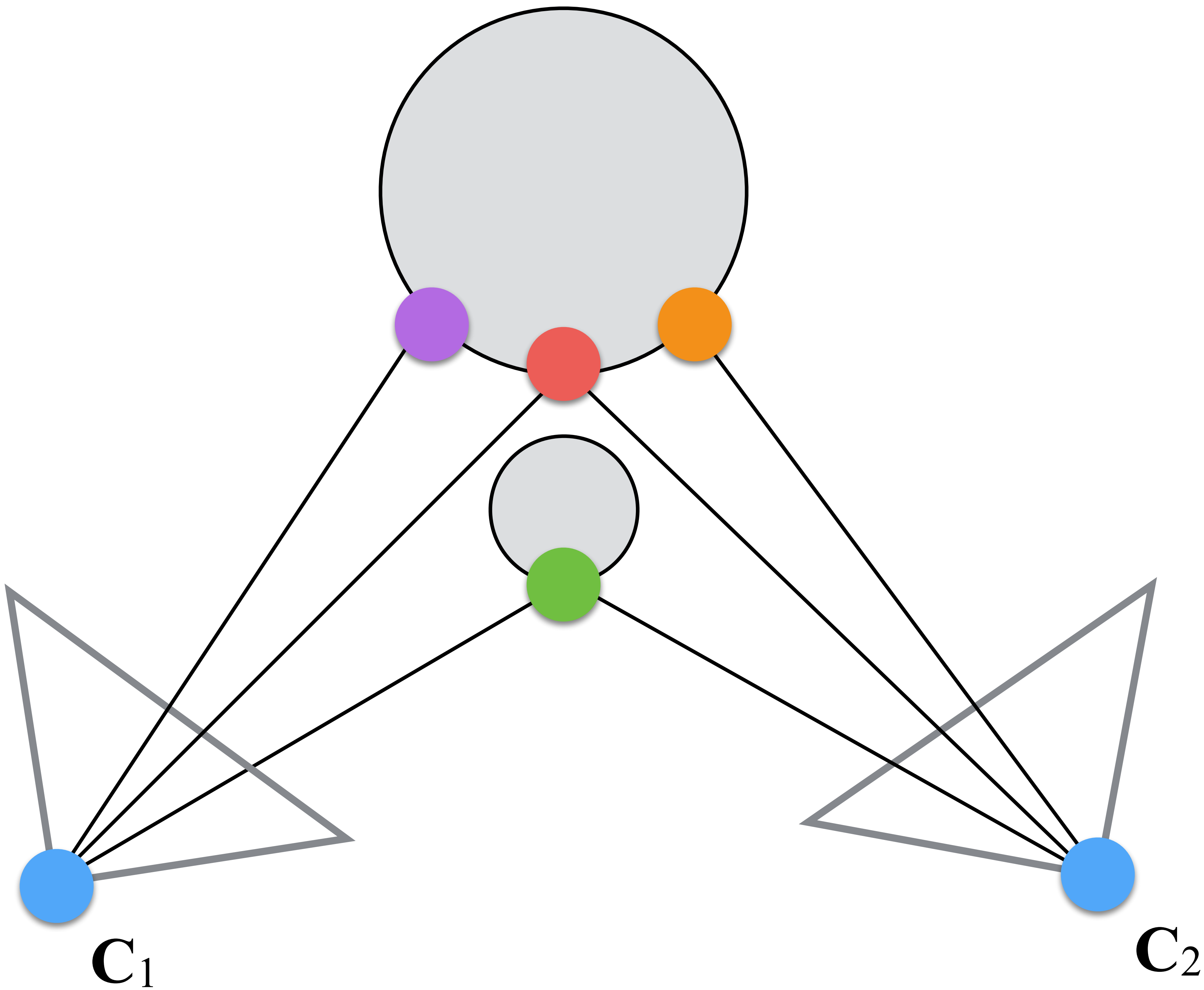


Non-Local Constraints: Correct Ordering

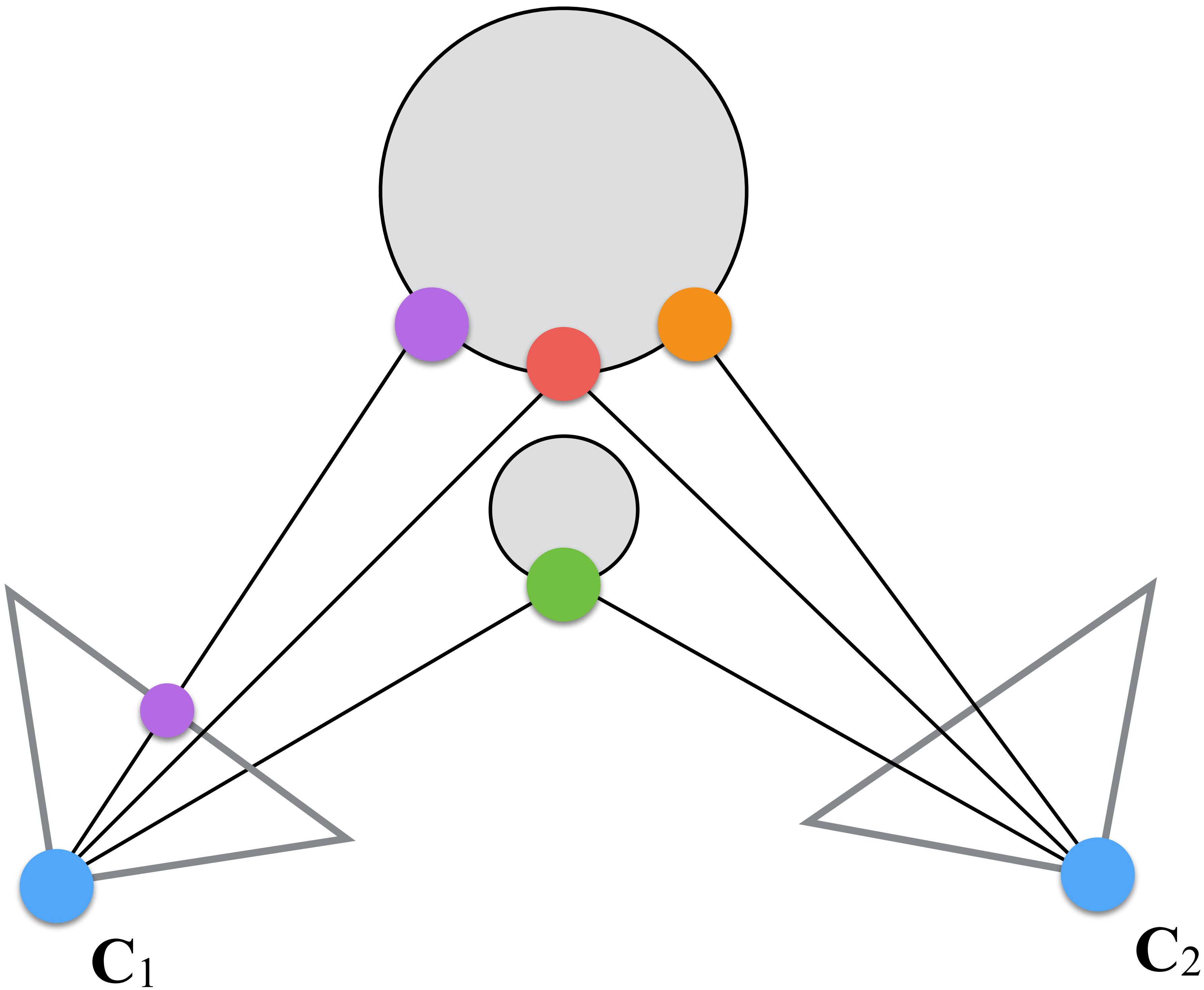
- Corresponding points should be in the same order in both views.



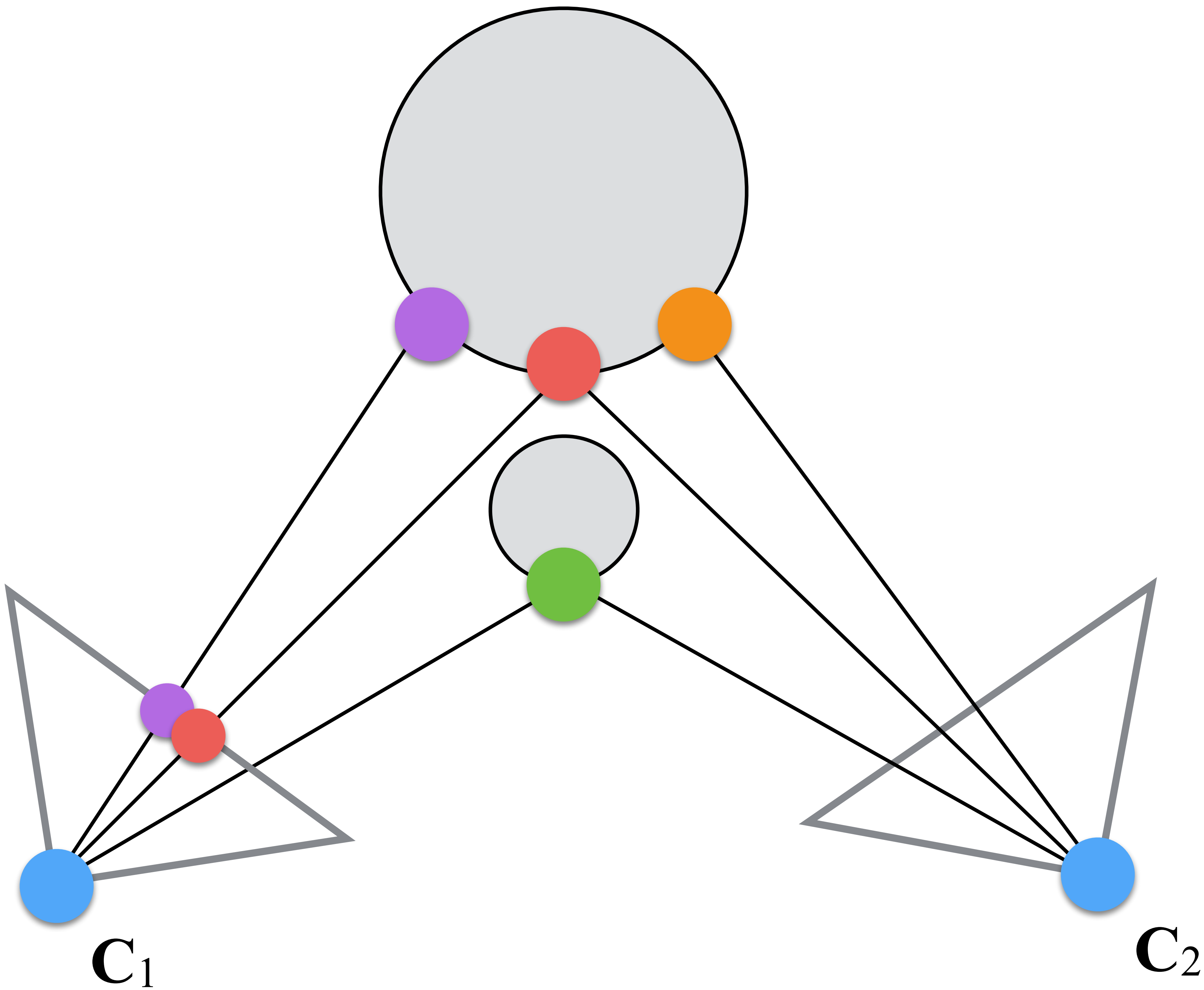
Non-Local Constraints: Ordering Fail



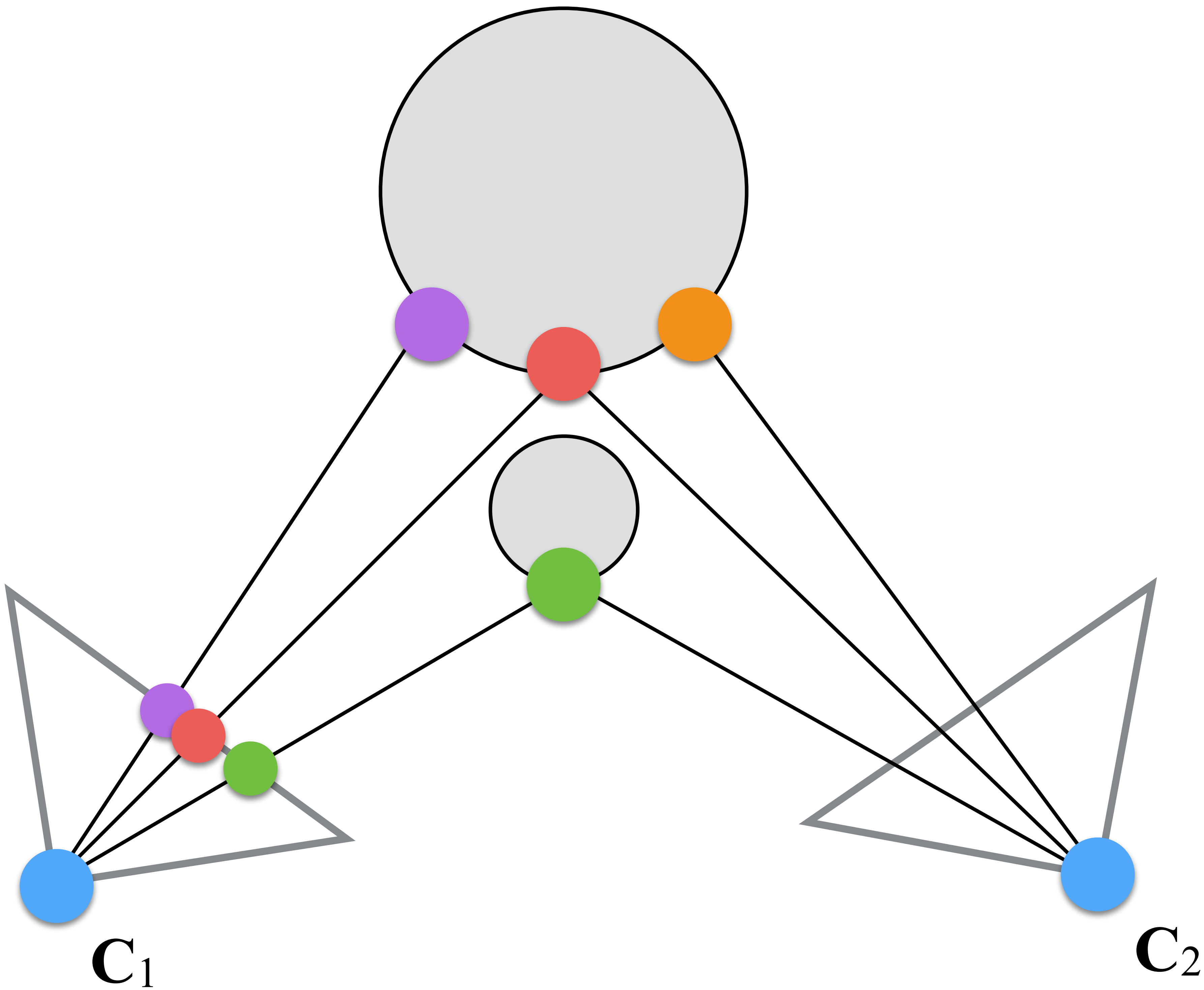
Non-Local Constraints: Ordering Fail



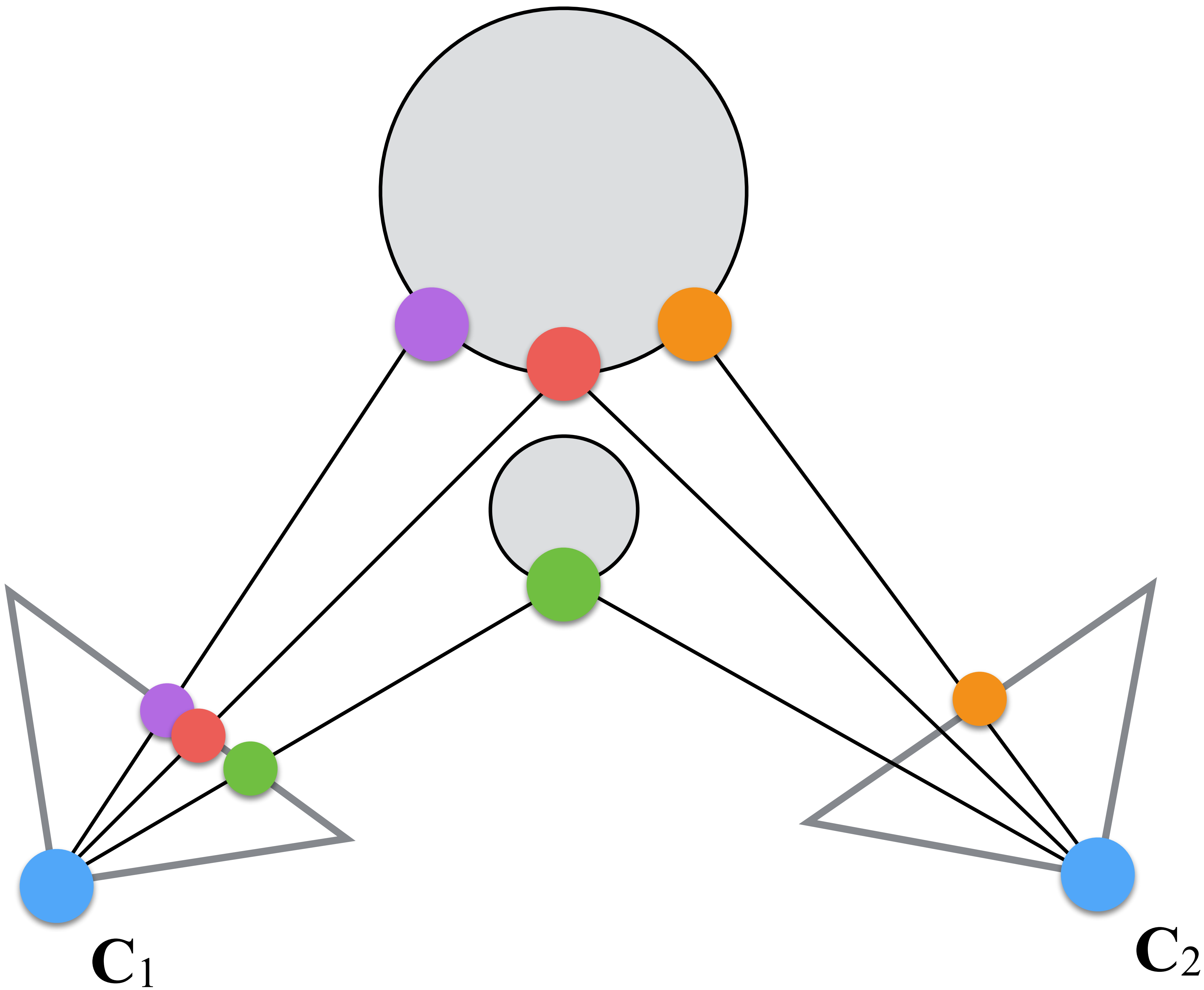
Non-Local Constraints: Ordering Fail



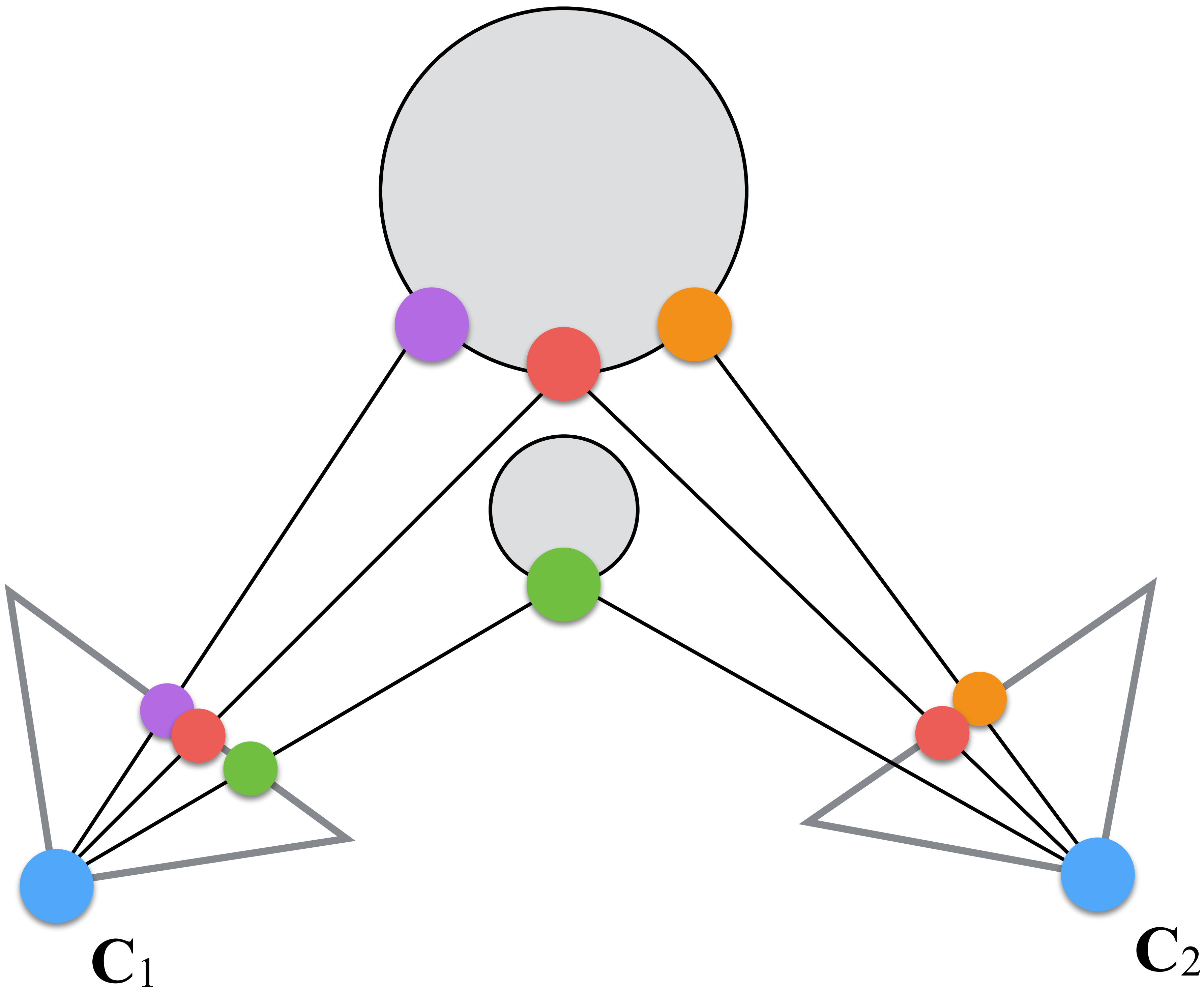
Non-Local Constraints: Ordering Fail



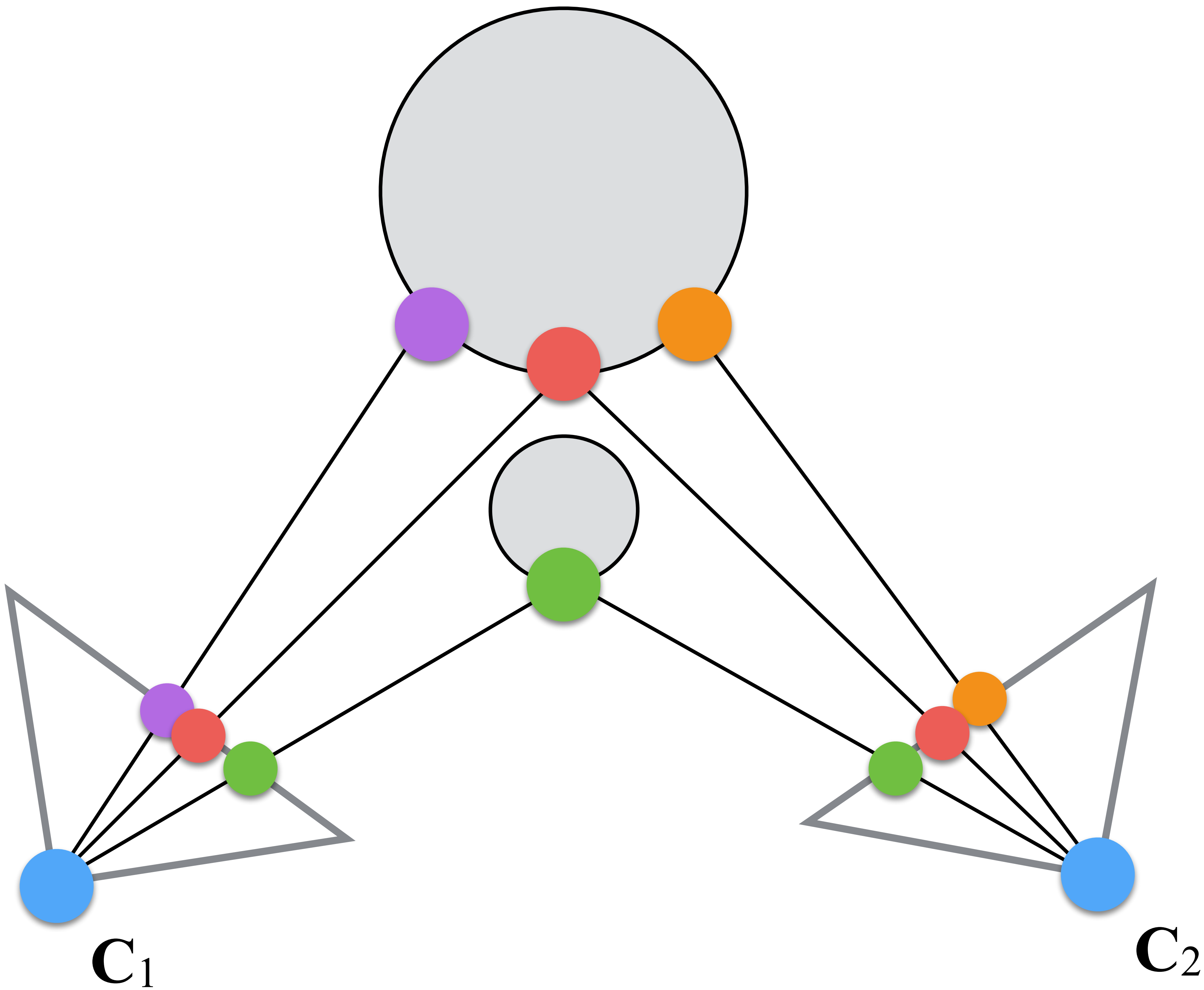
Non-Local Constraints: Ordering Fail



Non-Local Constraints: Ordering Fail

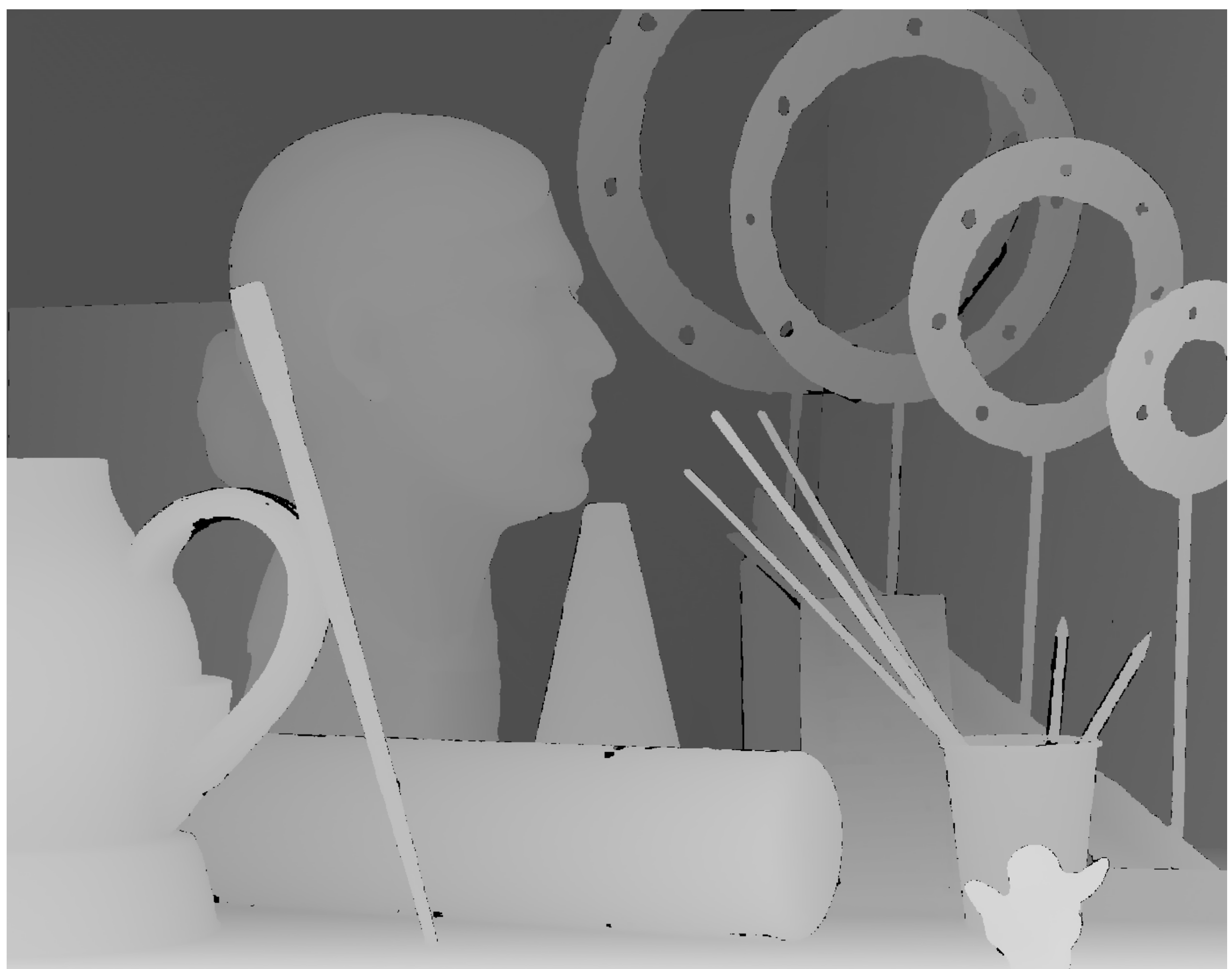


Non-Local Constraints: Ordering Fail



Non-Local Constraints: Smoothness

- We expect that disparity varies smoothly over the image, and it only greatly changes at edges.



Non-Local Constraints: Smoothness

- We can easily add a smoothness constraint E_s to the energy to minimize E obtaining a new energy to minimize called E_t:

$$E_t(x, y, d) = E(x, y, d) + \lambda E_s(x, y, d)$$

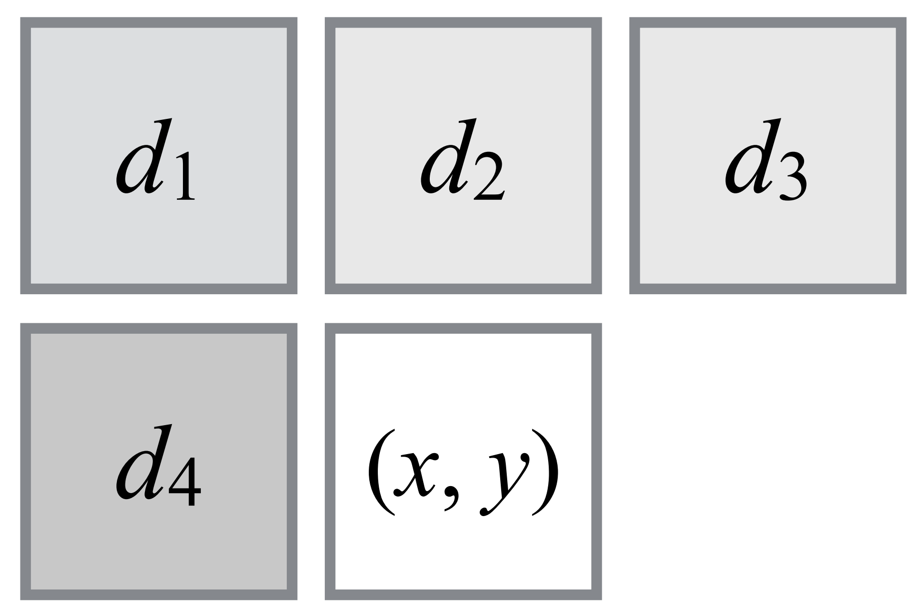
- where $\lambda > 0$ is the smoothing term, the higher the more the smoothness is enforced:
    - Typically, a value between 10% or 20% of the maximum disparity, $d_{\max}$.
    - if $\lambda = 0$ this implies $E_t = E$

Non-Local Constraints: Smoothness

- The computation of the disparity map follows (as usual) the scan order; i.e., from left to right and from top to bottom.

- So at the current pixel (x, y), we have already computed the disparity, D, at these previous locations:

  - $d_1 = D(x - 1, y - 1)$
  - $d_2 = D(x, y - 1)$
  - $d_3 = D(x + 1, y - 1)$
  - $d_4 = D(x - 1, y)$



Non-Local Constraints: Smoothness

- When defining E_s, we can enforce that the next disparity value is similar to one of the previous computed ones:

$$E_s(x, y, d) = \frac{1}{2}\left|D(x-1, y) - d\right| + \frac{1}{2}\left|D(x, y-1) - d\right|$$

Non-Local Constraints: Smoothness Example

Left (I_1)

$d_2 = 8$
$d_4 = 10$

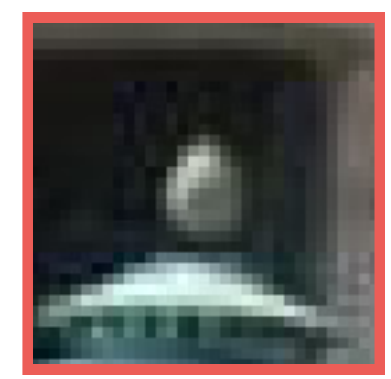
Non-Local Constraints: Smoothness Example

Left (I_1)

$d_2 = 8$
$d_4 = 10$

Non-Local Constraints: Smoothness Example

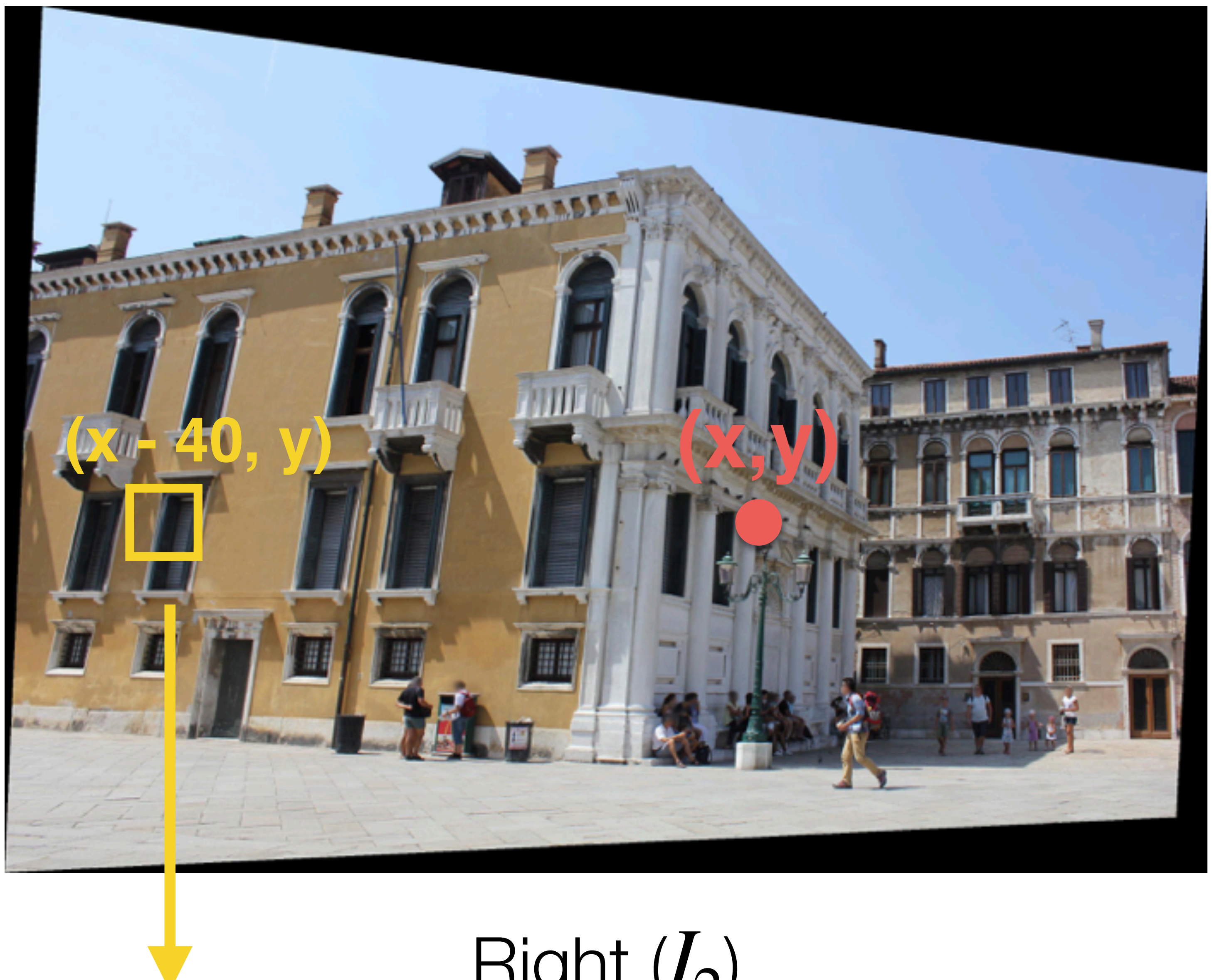
Left (I_1)



P_1

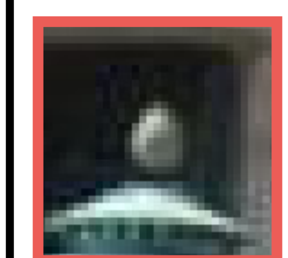
$d_2 = 8$
$d_4 = 10$

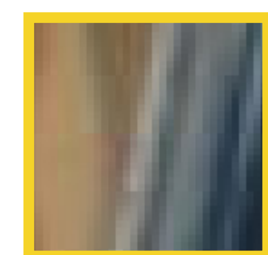
Non-Local Constraints: Smoothness Example



Right (I_2)

$d = -40$



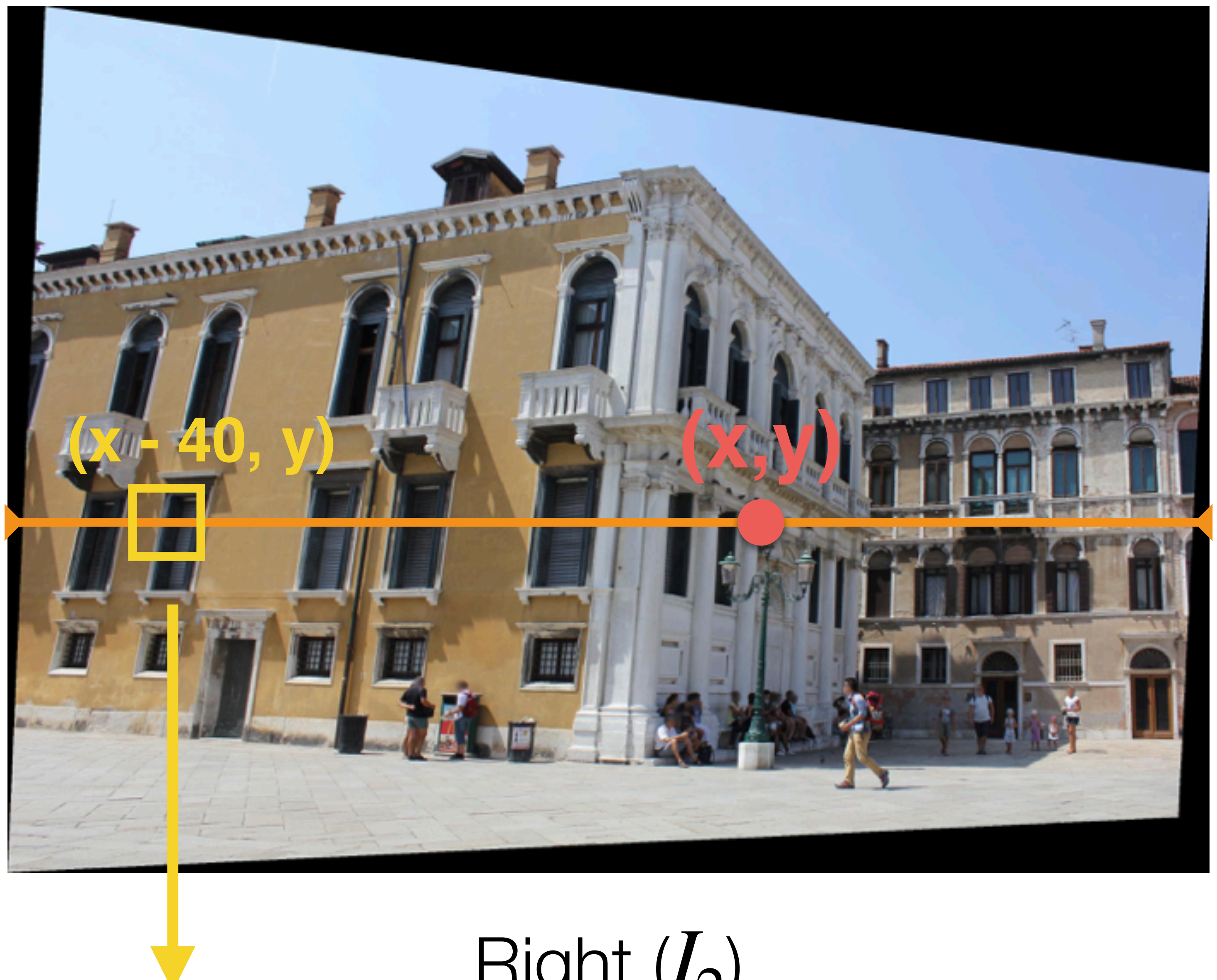


$$E = \left| P_1 - P_2 \right| = 30$$

$$E_s = 0.5\left|d_2 - (-40)\right| + 0.5\left|d_4 - (-40)\right|$$
$$= 0.5(8 + 40) + 0.5(10 + 40) = 49$$

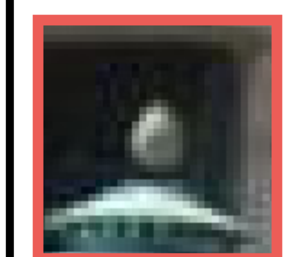
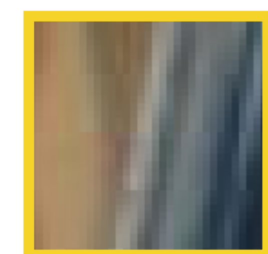
$$E_t = E + \lambda E_s = 30 + 0.2 \cdot 49 =$$
$$= 30 + 9.8 = 39.8 \quad \lambda = 0.2$$

Non-Local Constraints: Smoothness Example



Right (I_2)

$d = -40$

$E = \Big|$  -  $\Big| = 30$

P_1 P_2

$$E_s = 0.5\left|d_2 - (-40)\right| + 0.5\left|d_4 - (-40)\right|$$
$$= 0.5(8 + 40) + 0.5(10 + 40) = 49$$

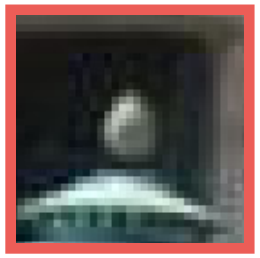
$$E_t = E + \lambda E_s = 30 + 0.2 \cdot 49 =$$
$$= 30 + 9.8 = 39.8 \quad \lambda = 0.2$$

Non-Local Constraints: Smoothness Example

Right (I_2)

$d = -16$

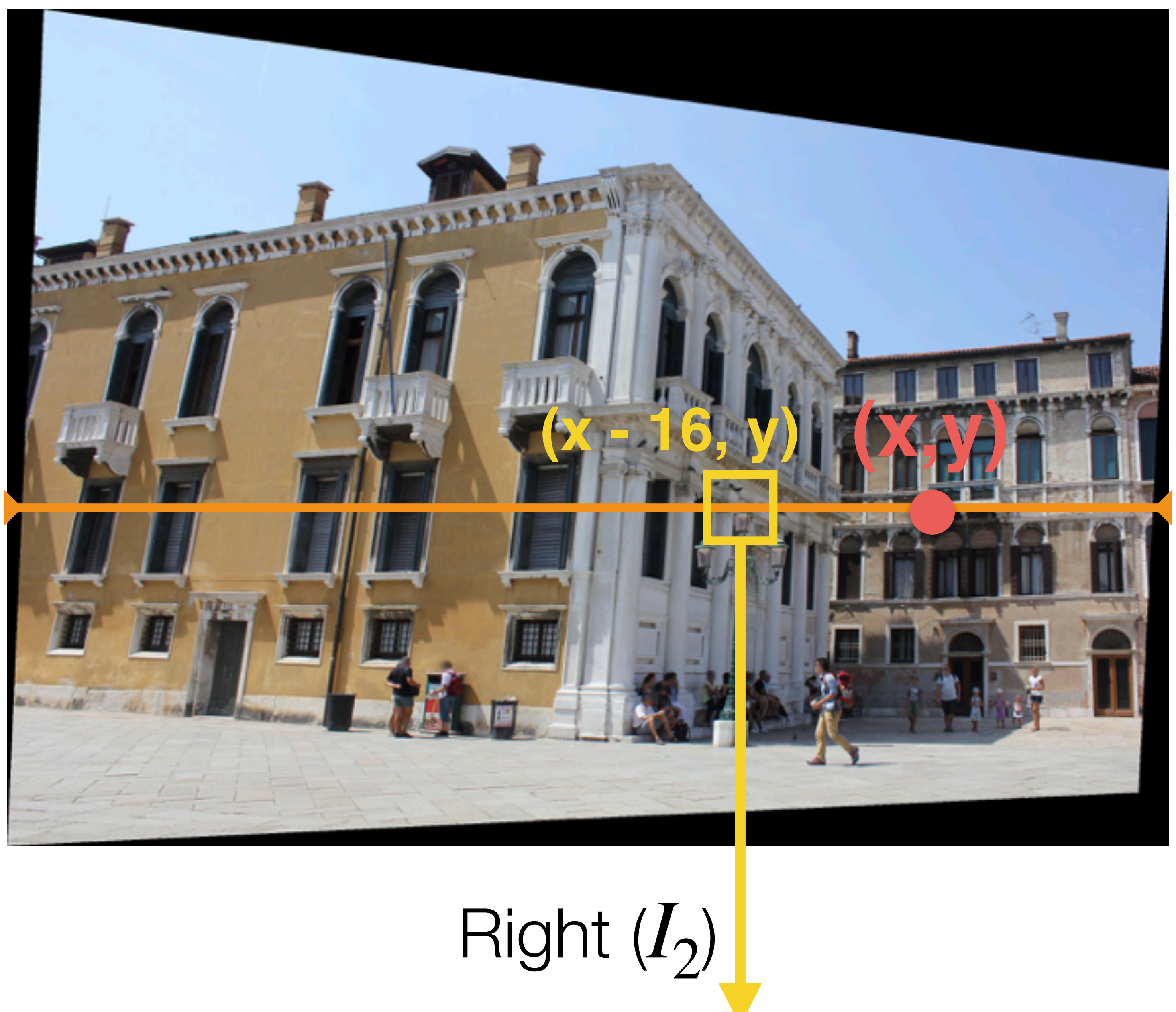
$$E = \left| P_1 - P_2 \right| = 32$$



$$E_s = 0.5\left|d_2 - (-16)\right| + 0.5\left|d_4 - (-16)\right| = 0.5(8 + 16) + 0.5(10 + 16) = 25$$

$$E_t = E + \lambda E_s = 32 + 0.2 \cdot 25 = 32 + 5 = 37 \quad \lambda = 0.2$$

Non-Local Constraints: Smoothness Example



Right (I_2)

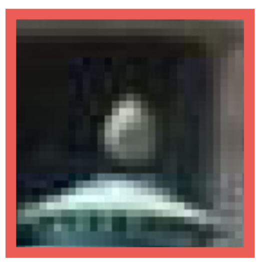
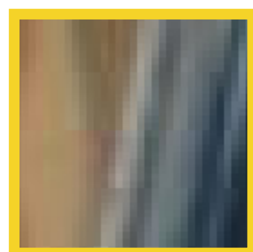
$$d = -16$$

$$E = \left| \; P_1 \; - \; P_2 \; \right| = 32$$

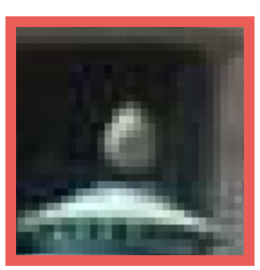
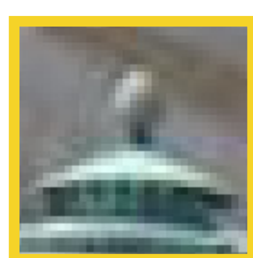
$$E_s = 0.5\left|d_2 - (-16)\right| + 0.5\left|d_4 - (-16)\right| = 0.5(8 + 16) + 0.5(10 + 16) = 25$$

$$E_t = E + \lambda E_s = 32 + 0.2 \cdot 25 = = 32 + 5 = 37 \quad \lambda = 0.2$$

Non-Local Constraints: Smoothness Example

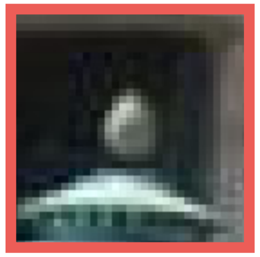
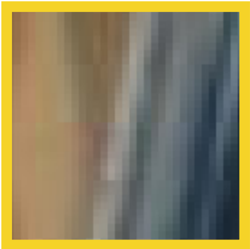
$E_t \Big($   $\Big) = 39.8$

$d = -40$

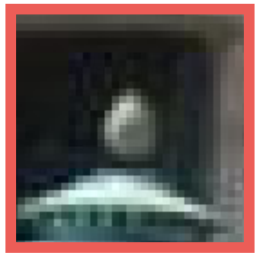
$E_t \Big($   $\Big) = 37$

$d = -16$

Non-Local Constraints: Smoothness Example

$E_t\Big($   $\Big) = 39.8$

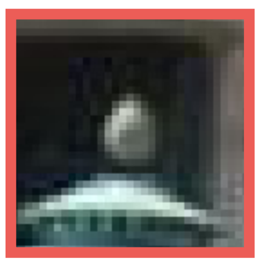
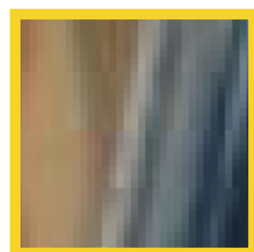
$d = -40$

$E_t\Big($   $\Big) = 37$

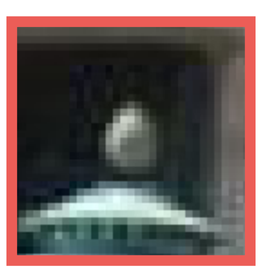
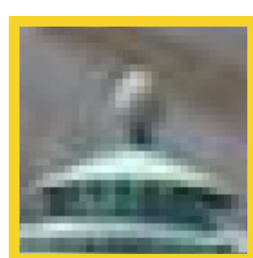
This is lower than 39.8!

$d = -16$

Non-Local Constraints: Smoothness Example

$E_t\Big($   $\Big) = 39.8$

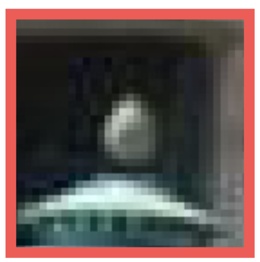
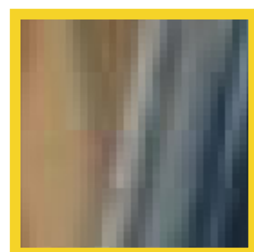
$d = -40$

$E_t\Big($   $\Big) = 37$

This is lower than 39.8!

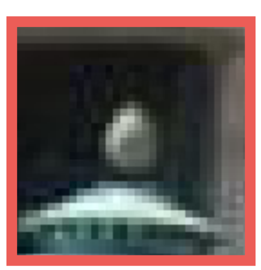
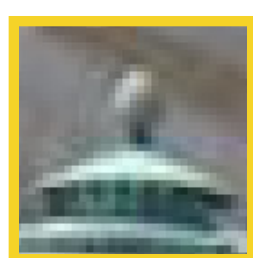
$d = -16$

Non-Local Constraints: Smoothness Example

$E_t\Big($   $\Big) = 39.8$

$d = -40$

$\Rightarrow D(x, y) = d = -16$

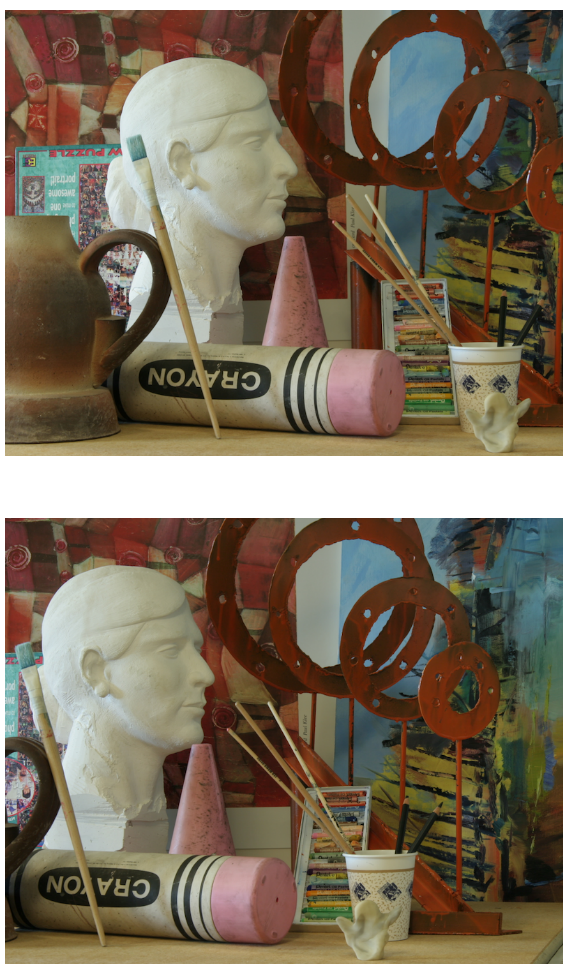
$E_t\Big($   $\Big) = 37$

This is lower than 39.8!

$d = -16$

How do we choose parameters?

Dense Matching: Choosing *n*

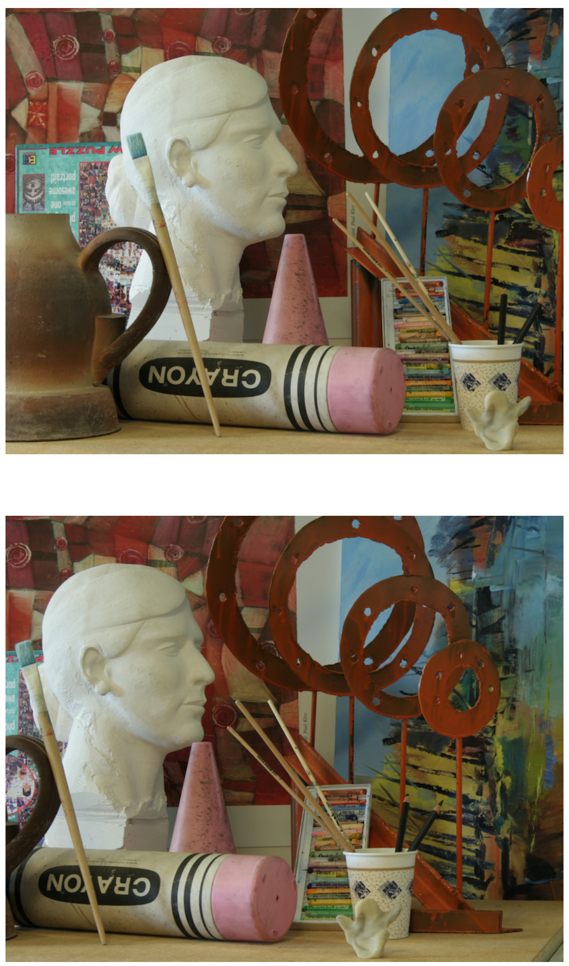


Input

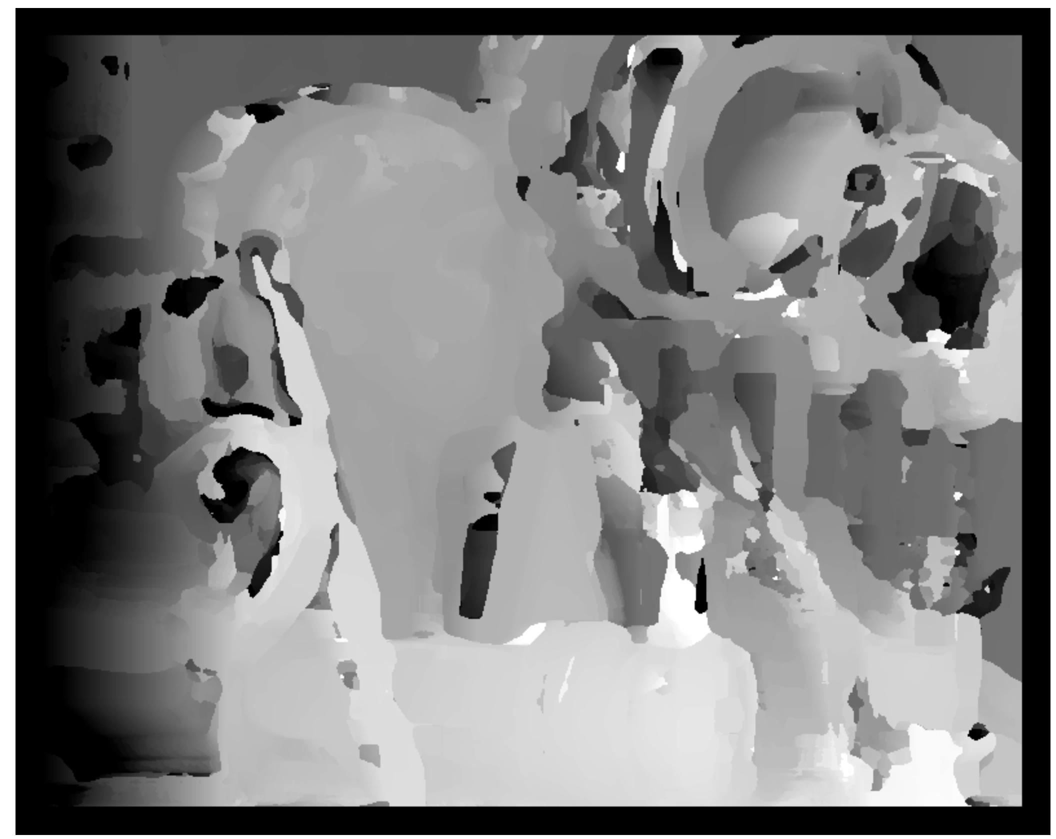
$n = 3$

- Smaller *n* —> more details but more noise!

Dense Matching: Choosing n



Input



$n = 21$

- Larger n —> less noise but less details!

Dense Matching: Choosing *n*

- A way to detect the correct a suitable windows size is to start with a size n=3.

- Iteratively, we increase n by one up to a maximum value, and we choose the window that minimizes this cost function:

$$C(n) = \overline{E_t} + \alpha \cdot \mathrm{Var}(E_t) + \frac{\beta}{n + \gamma}$$

$$\alpha = 1.5 \quad \beta = 7 \quad \gamma = -2$$

Dense Matching: Scanning the Line

- We do not have to check the whole line!

- If we have sparse matches from feature points, we have a bound to the maximum disparity, $d_{\max}$.

- We compute $d_{\max}$ as the maximum disparity that we have in input feature points.

- This means: $d \in [-d_{\max}, d_{\max}]$

Dense Matching: Scanning the Line without $d_{\max}$

Left

Right

Dense Matching: Scanning the Line with $d_{\max}$

Left

Right

Dense Matching: Limitations

- Failure cases:
  - No textures; e.g., a white wall.
  - Specular surfaces; e.g., a mirror or shiny surface.
  - Repeated occlusions; a gate.
  - Baseline is too short (e.g., very close cameras) implies high error in the disparity. This means that the two images look the same.

Handling Occlusions

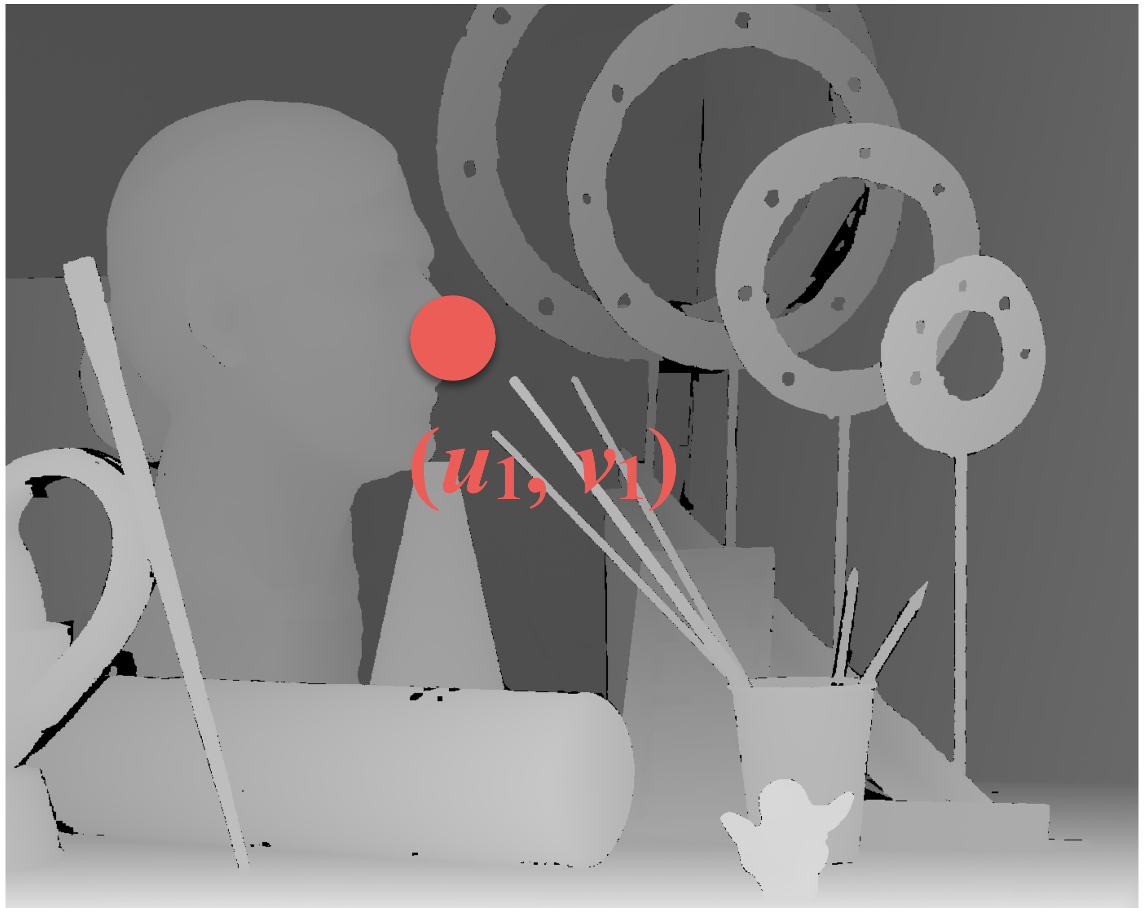
- Given two views, one view cannot “see” everything that the other does!
- In many cases we have to handle occlusions!
- If we generate two disparity/depth maps, we can use them to test if they are coherent!

Handling Occlusions: Example

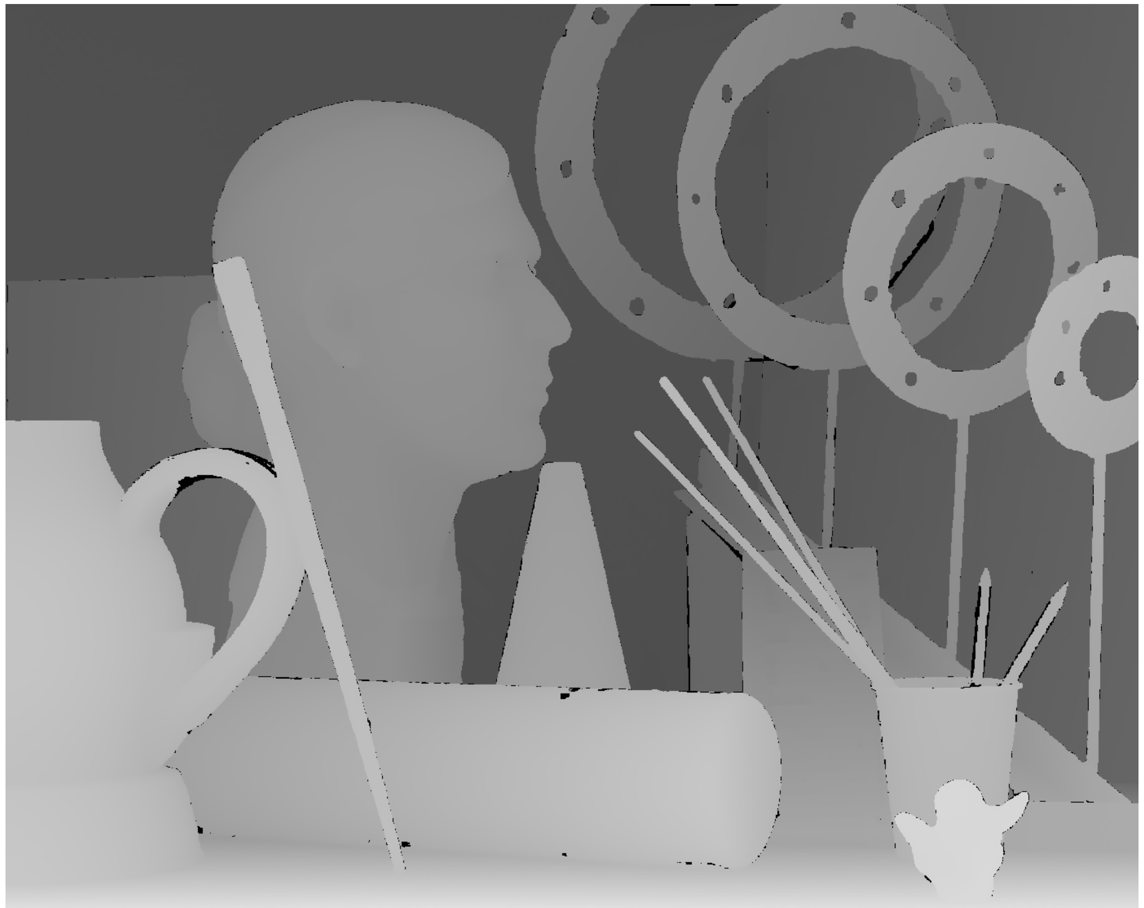
Left

Right

Handling Occlusions: Example

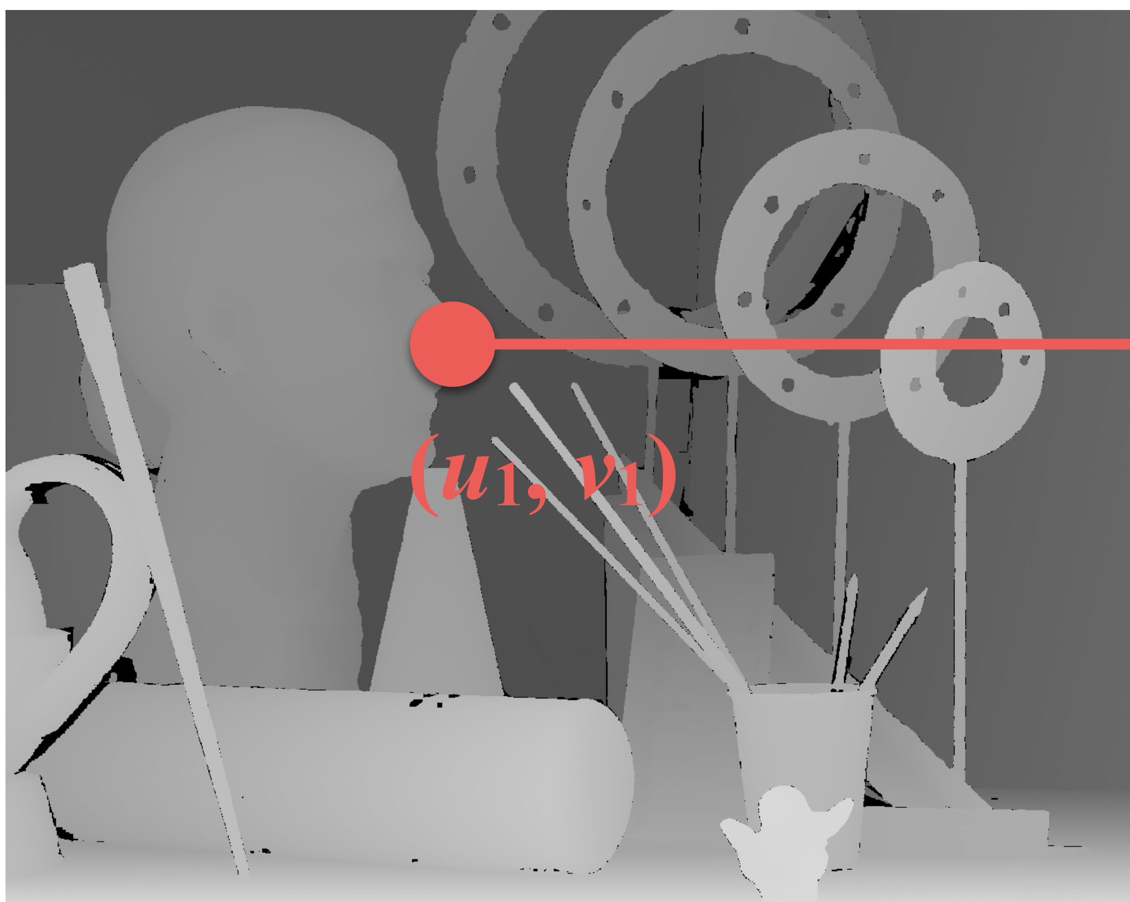


Left

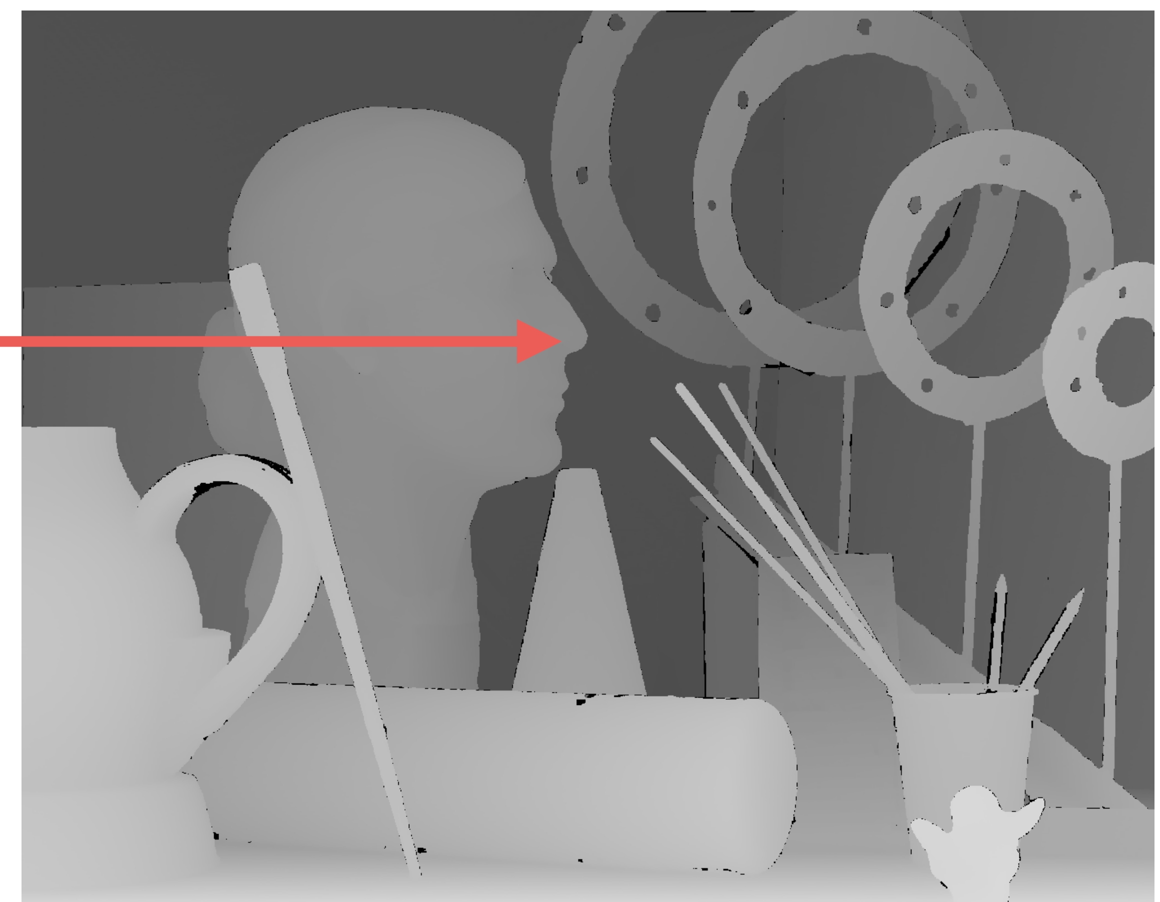


Right

Handling Occlusions: Example

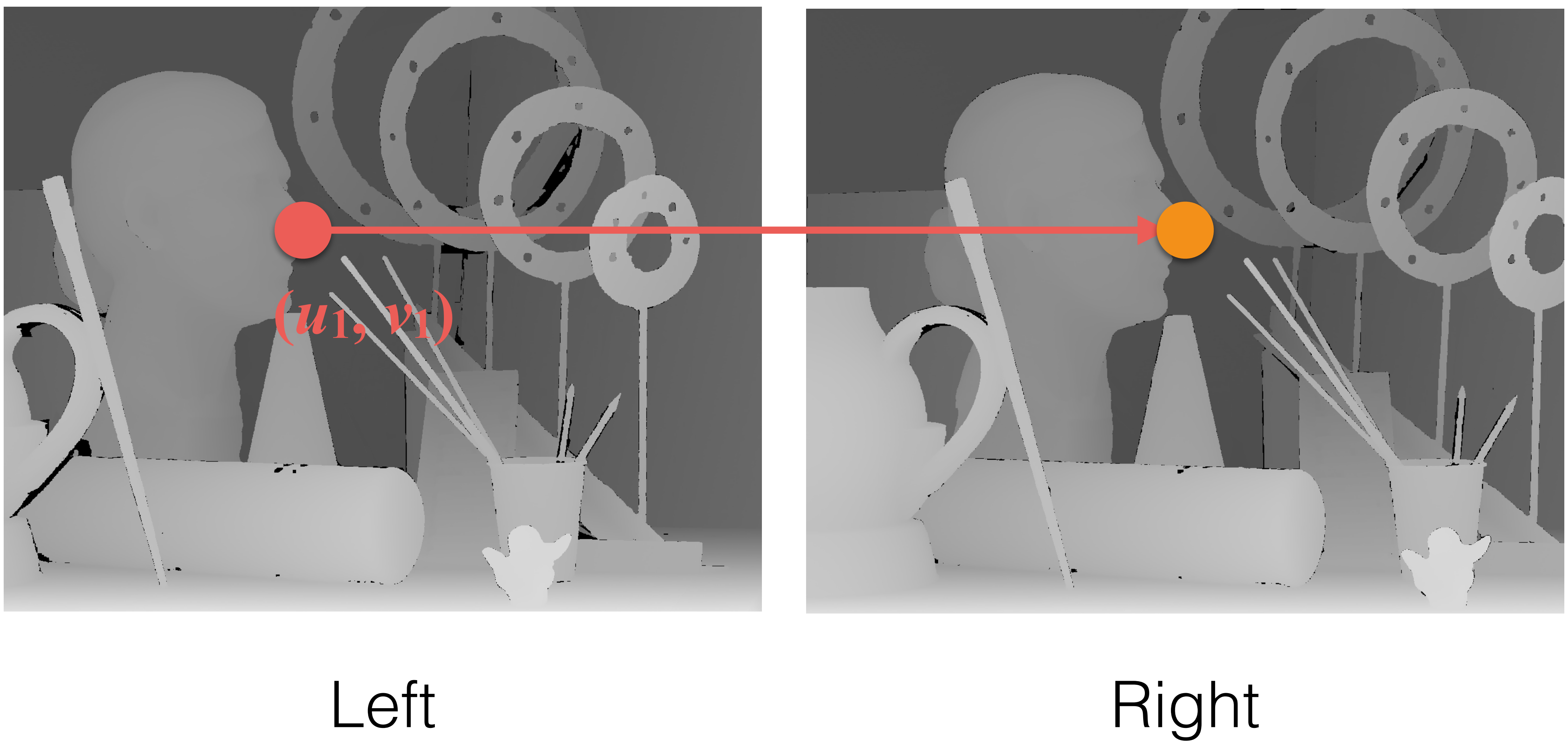


Left

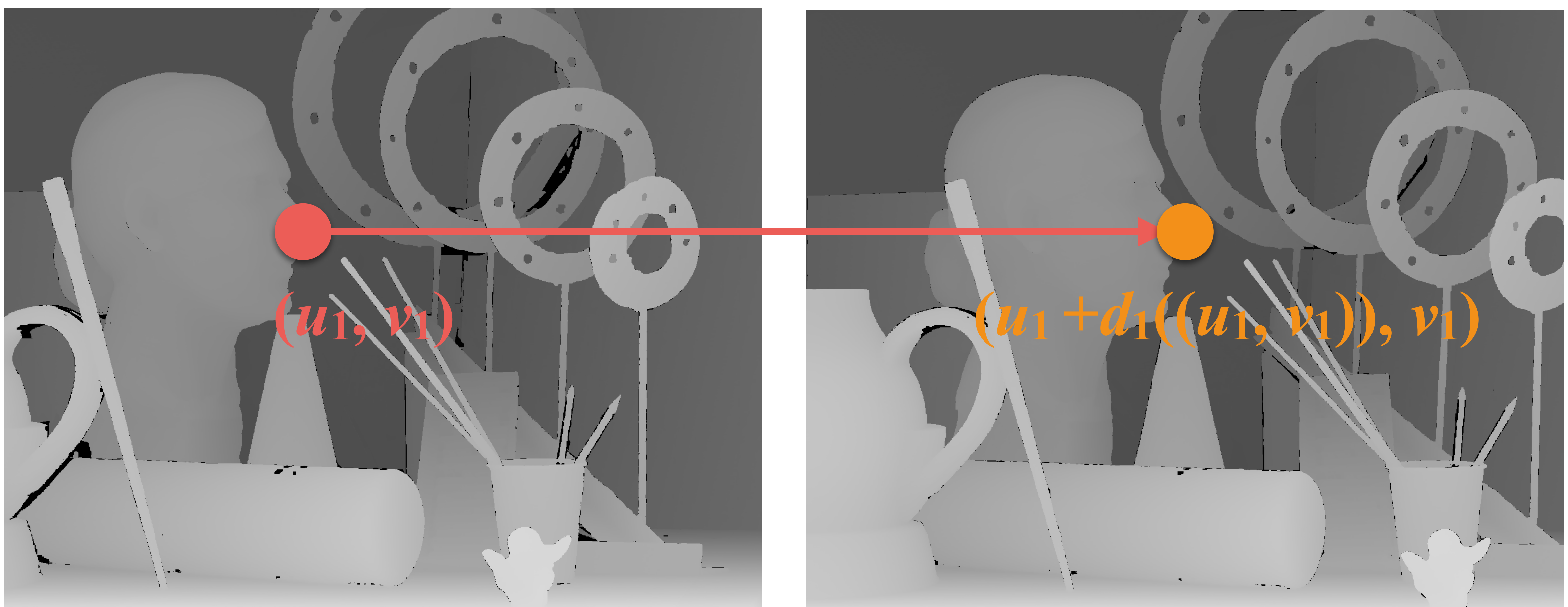


Right

Handling Occlusions: Example



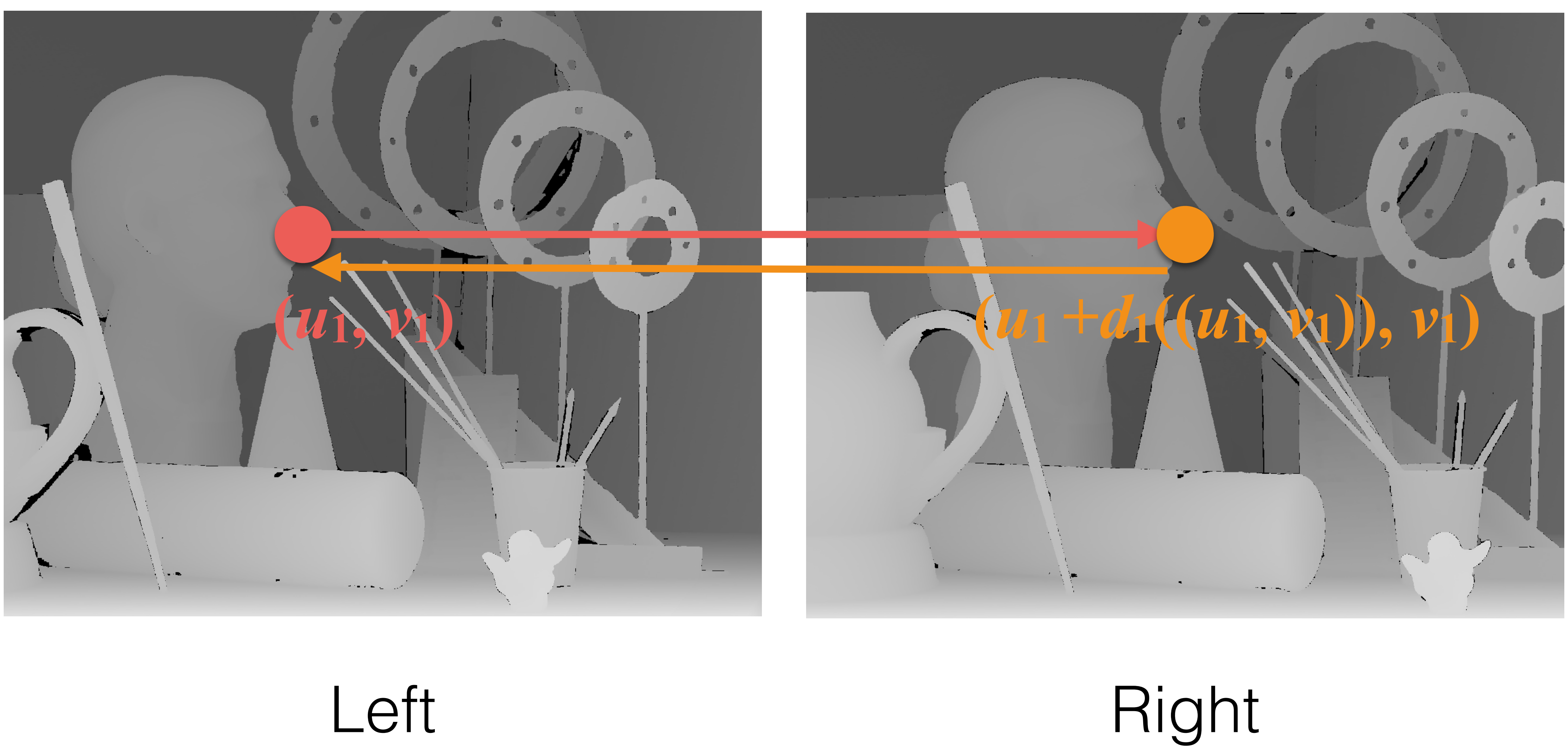
Handling Occlusions: Example



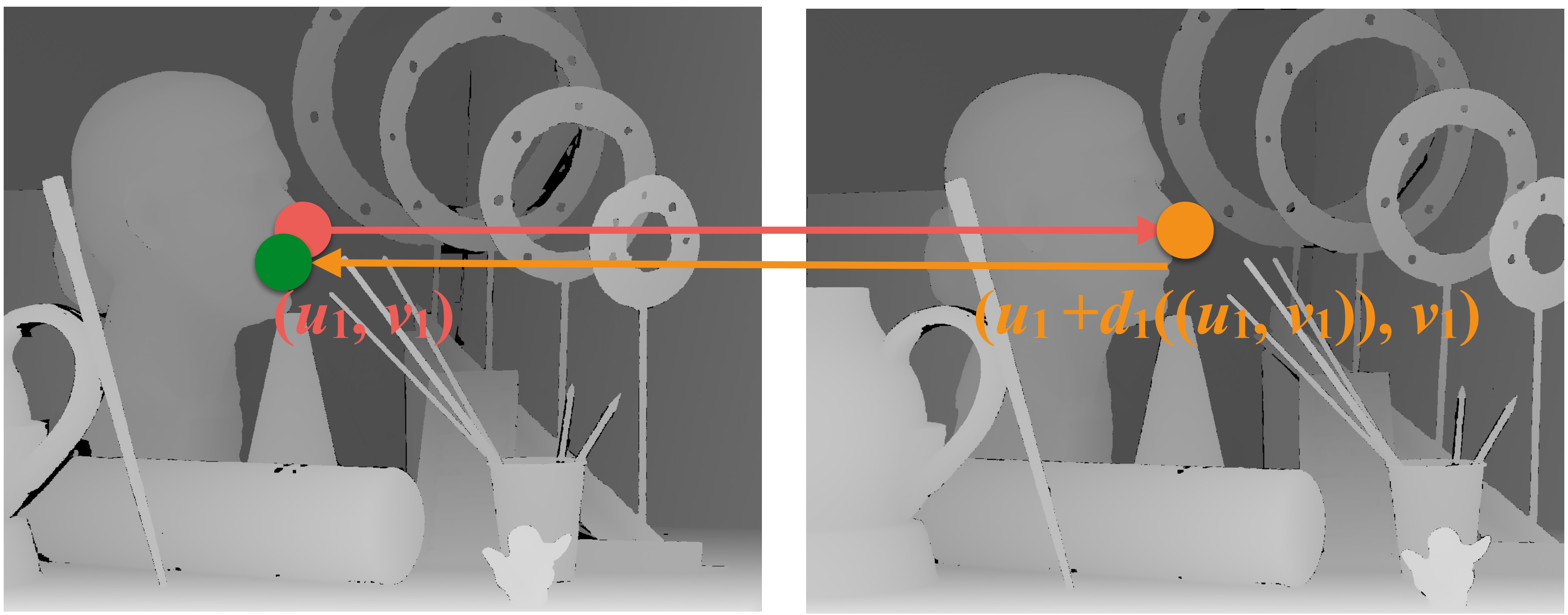
Left

Right

Handling Occlusions: Example



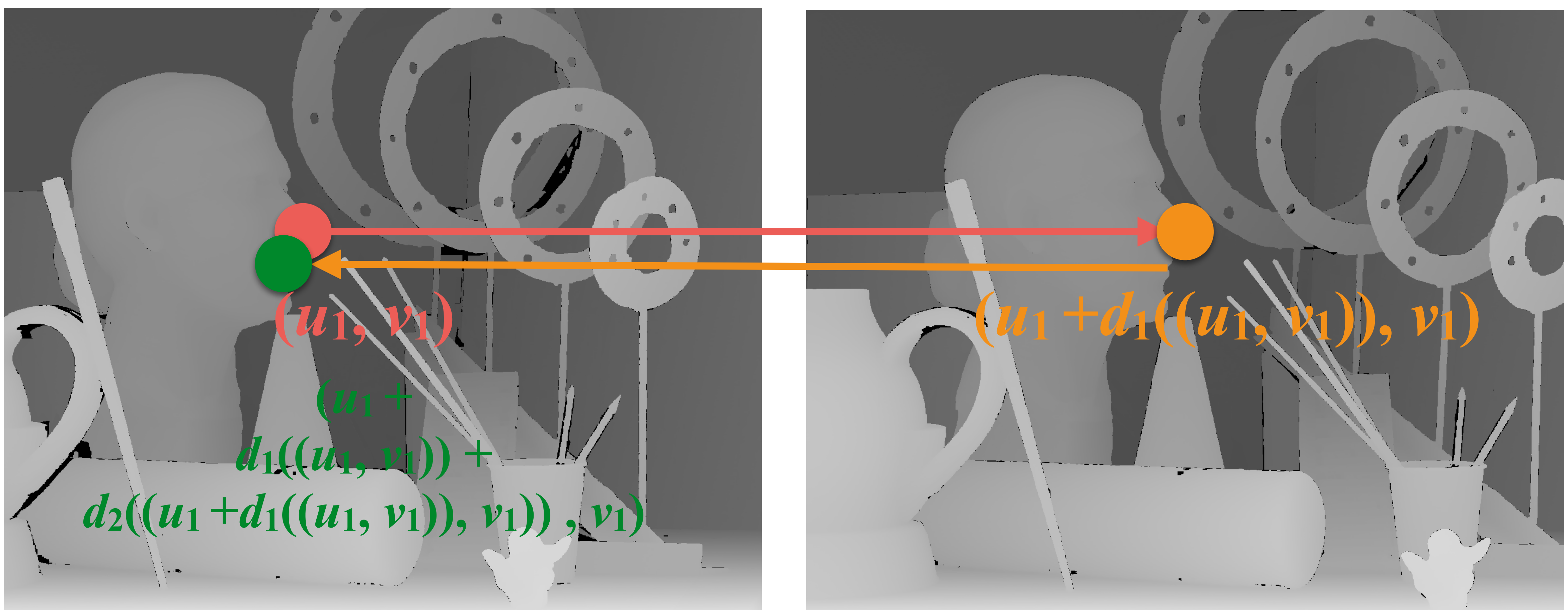
Handling Occlusions: Example



Left

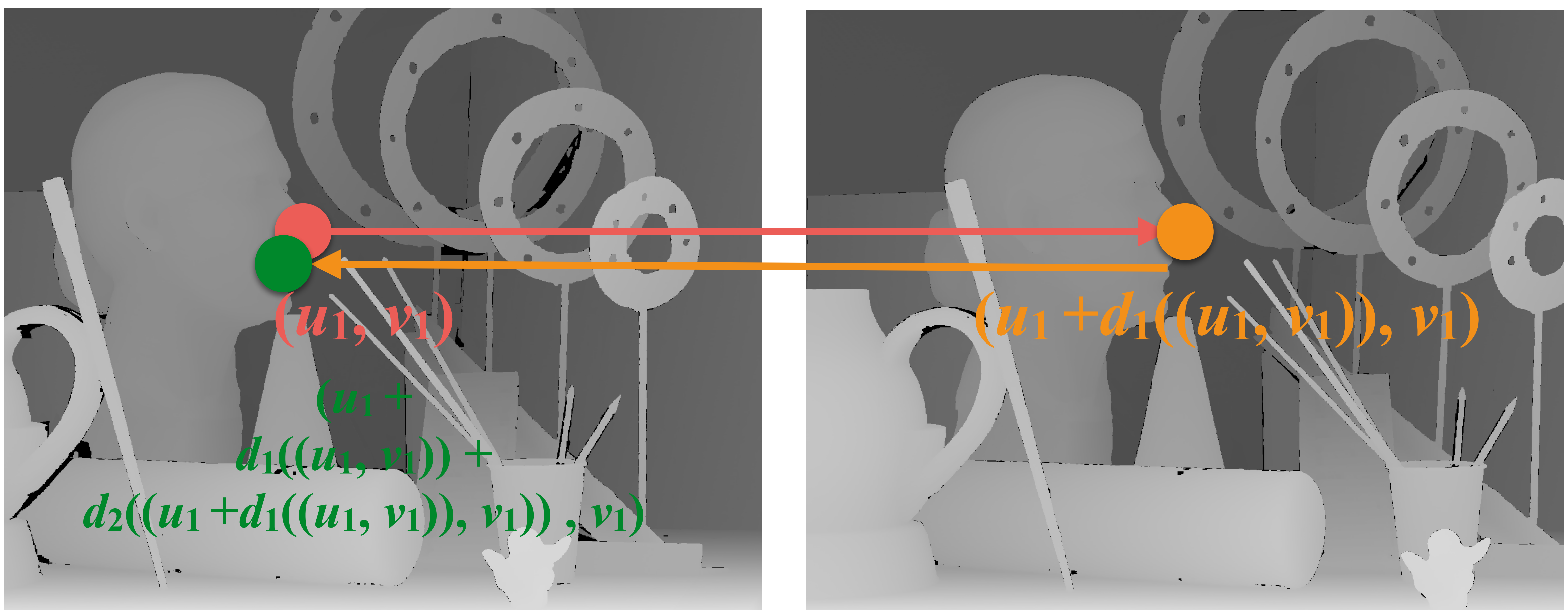
Right

Handling Occlusions: Example



Left Right

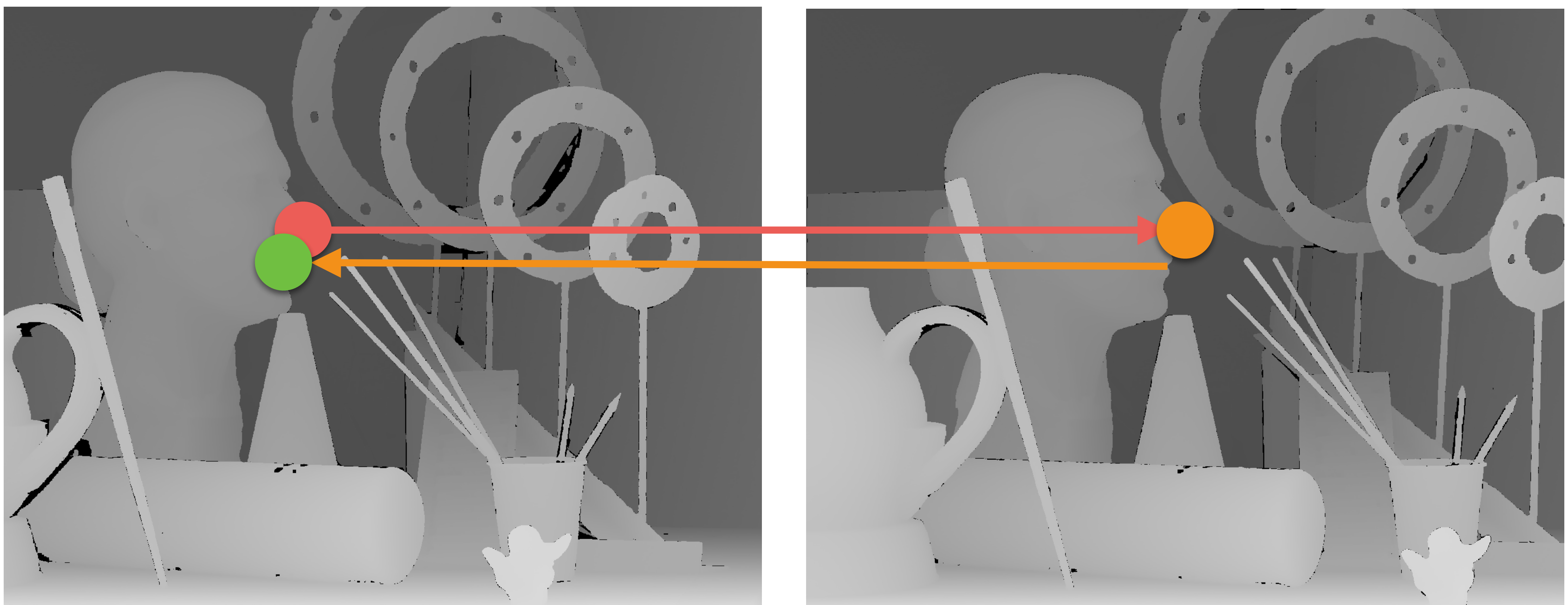
Handling Occlusions: Example



Left Right

If and are similar, both disparity values are fine.

Handling Occlusions: Example



Left

Right

Otherwise we have an occlusion and we set to ***NULL*** both values.

Handling Occlusions: Math

- We computed two ***disparity maps*** (d_1 and d_2) such that:

$$I_1(u_1, v_1) = I_2(u_1 + d_1(u_1, v_1), v_1)$$
$$I_2(u_2, v_2) = I_1(u_2 + d_2(u_2, v_2), v_2)$$

- Then the check is defined as (where t is a threshold, e.g., 1-2 pixels)

$$\begin{aligned} D &= d_1(u_1, v_1) \\ u'_2 &= u_1 + D \\ D' &= d_2(u_1 + D, v_1) \\ u'_1 &= u_2 + D' \end{aligned}$$

$$\begin{cases} \text{valid} & \text{if } |u_1 - u'_1| < t \\ \text{occlusion} & \text{otherwise} \end{cases}$$

Multi-View Stereo

Multi-View Stereo

- **Input**: ***three*** or more images of the same scene taken from different positions (no pure rotational motion!) and their camera matrices:
  - We can have sparse 3D points if this is an input from SfM.
- **Output**: either several depth maps (as many as the input images) or a densified point cloud. In the past volumes as well.

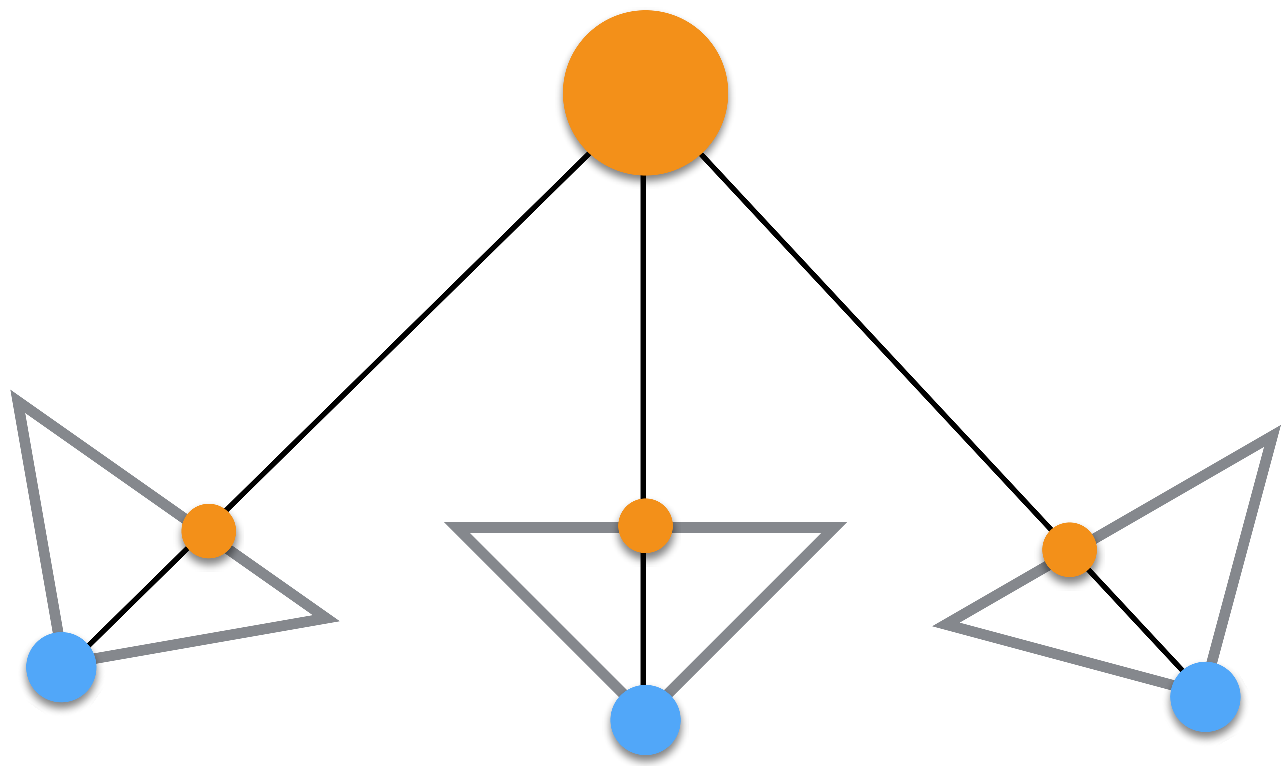
Multi-View Stereo

- There are three main approaches:
  - Computing depth-maps from n images:
    - The same formulation of Stereo, but more images!
  - Volume Carving.
  - Propagating the known 3D information of the sparse point cloud.

Multi-View Stereo: Stereo Extension

- Stereo is extended to handle multiple views.
- These views need to “see” the object or partially see it, otherwise they fail.

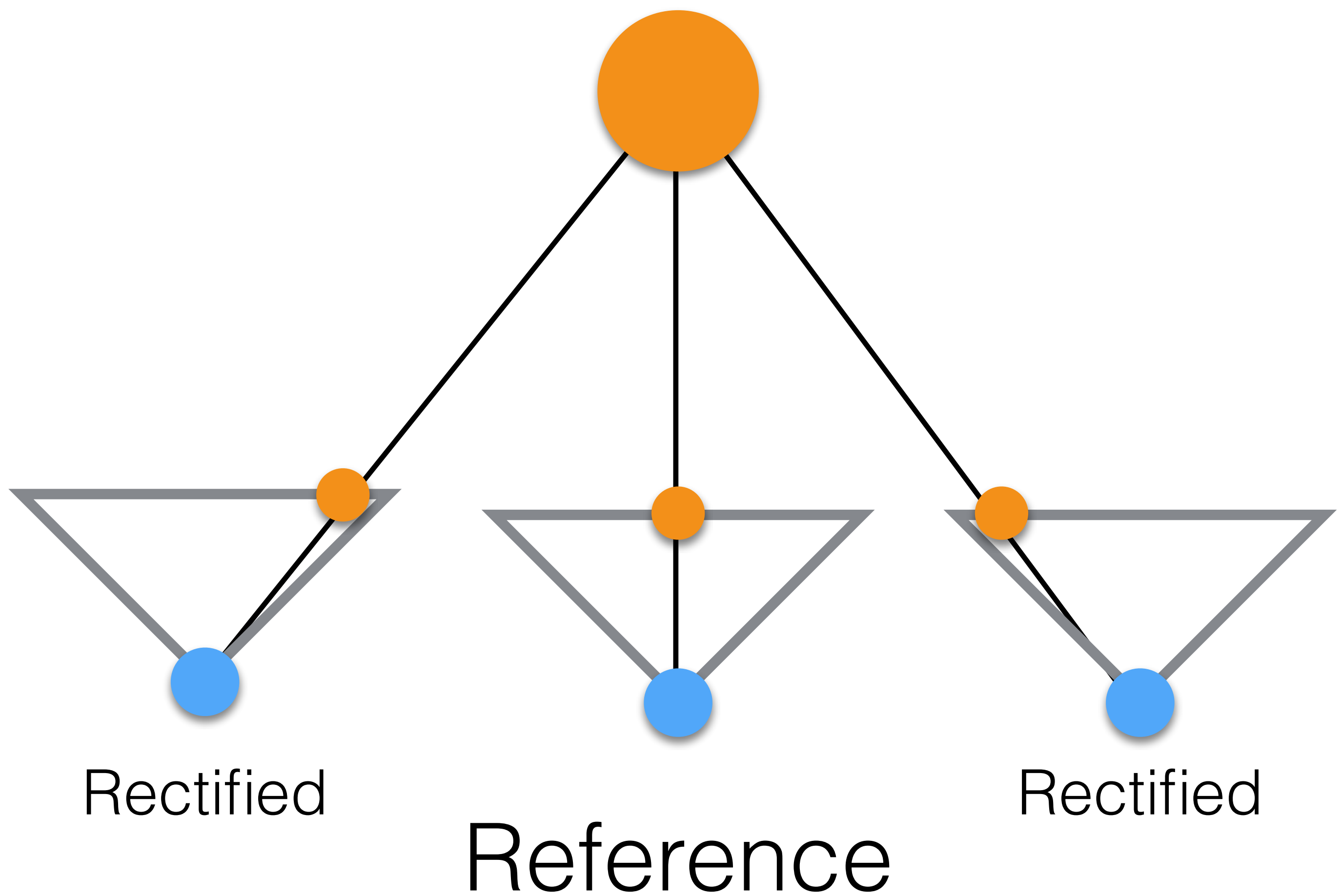
Multi-View Stereo: Stereo Extension



Reference

- Select a reference camera:
  - The one that has the most number of shared features with all other cameras.

Multi-View Stereo: Stereo Extension



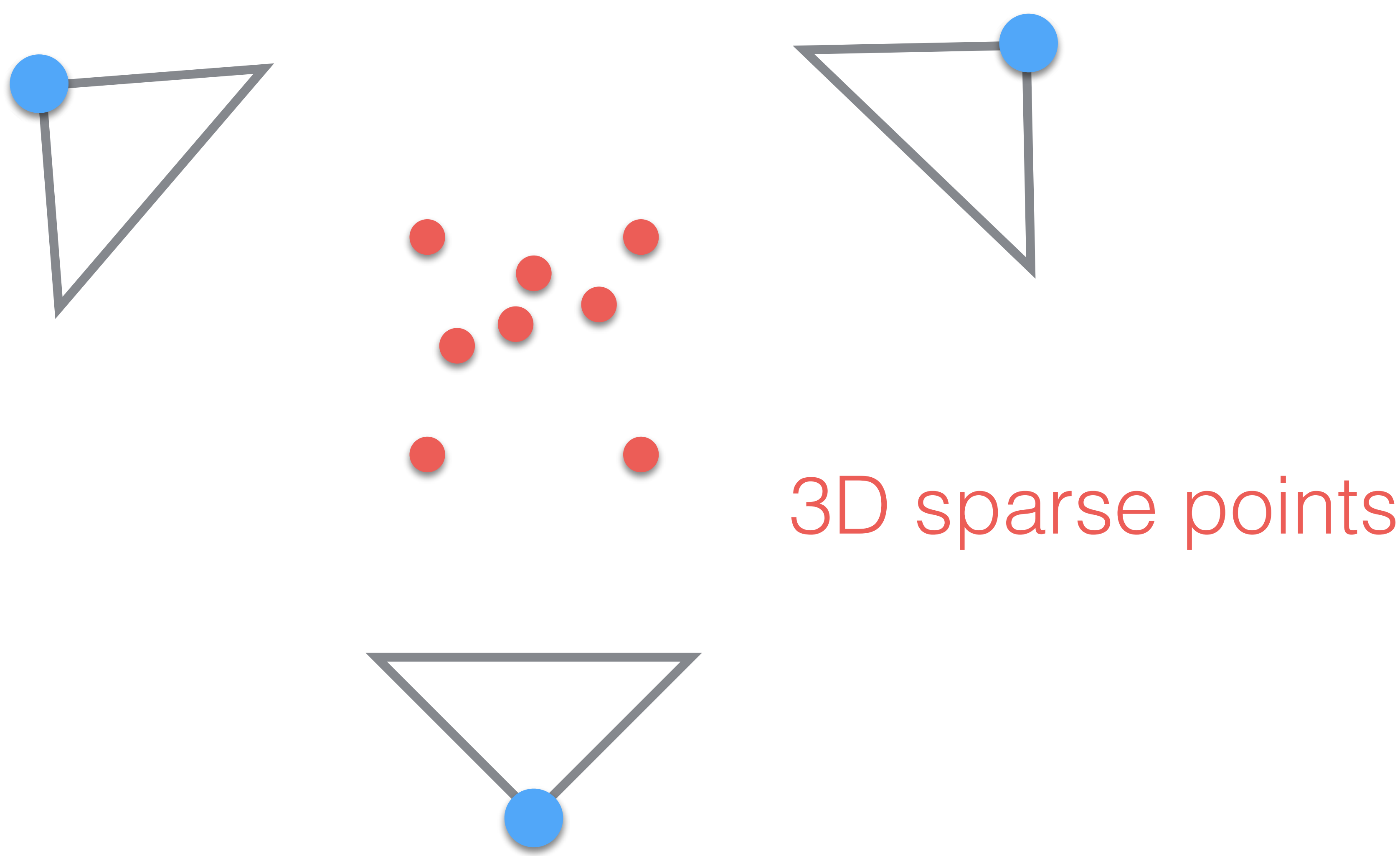
Multi-View Stereo: Stereo Extension

- Advantages:
  - We have a “single” rectification.
- Disadvantages:
  - All views need to “see” the same part of the object. This limits the whole thing to a group of cameras/views.

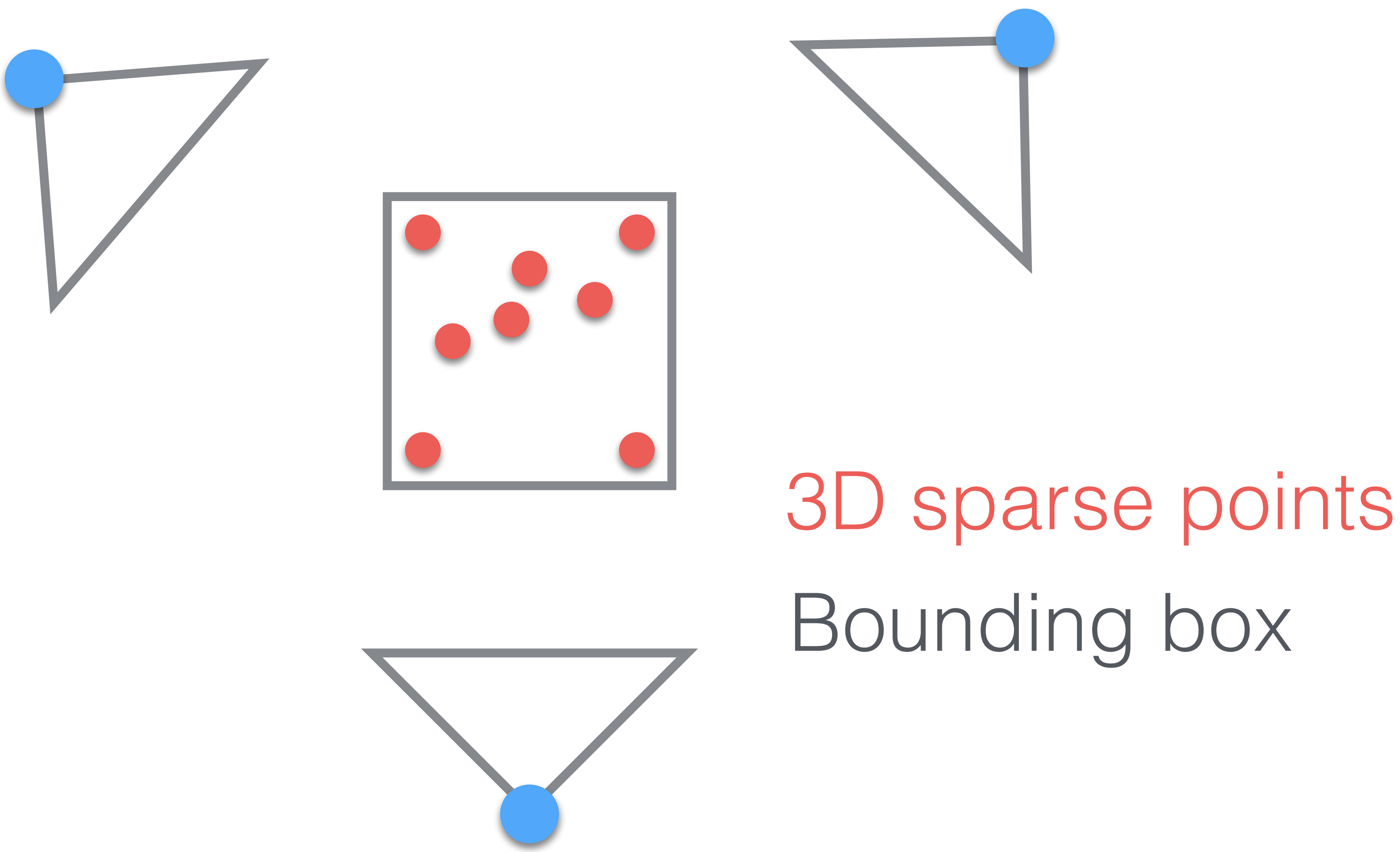
Multi-View Stereo: Space Carving

- Space carving is an algorithm with a volumetric-approach:
  - We compute the bounding box (BB) of triangulated 3D points.
  - We generate a 3D volume out of BB.
  - We carve voxel in the volume according to views.

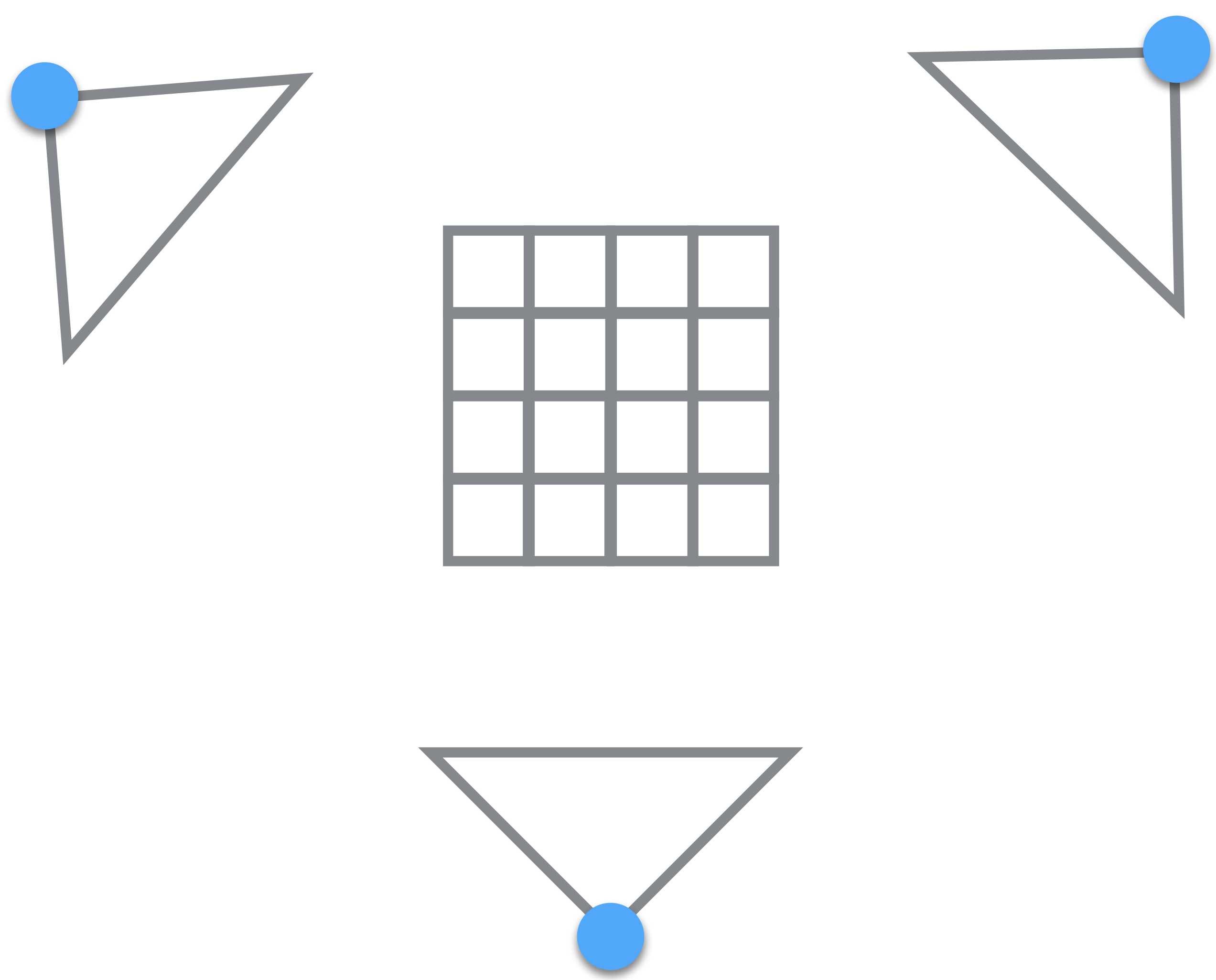
Multi-View Stereo: Space Carving



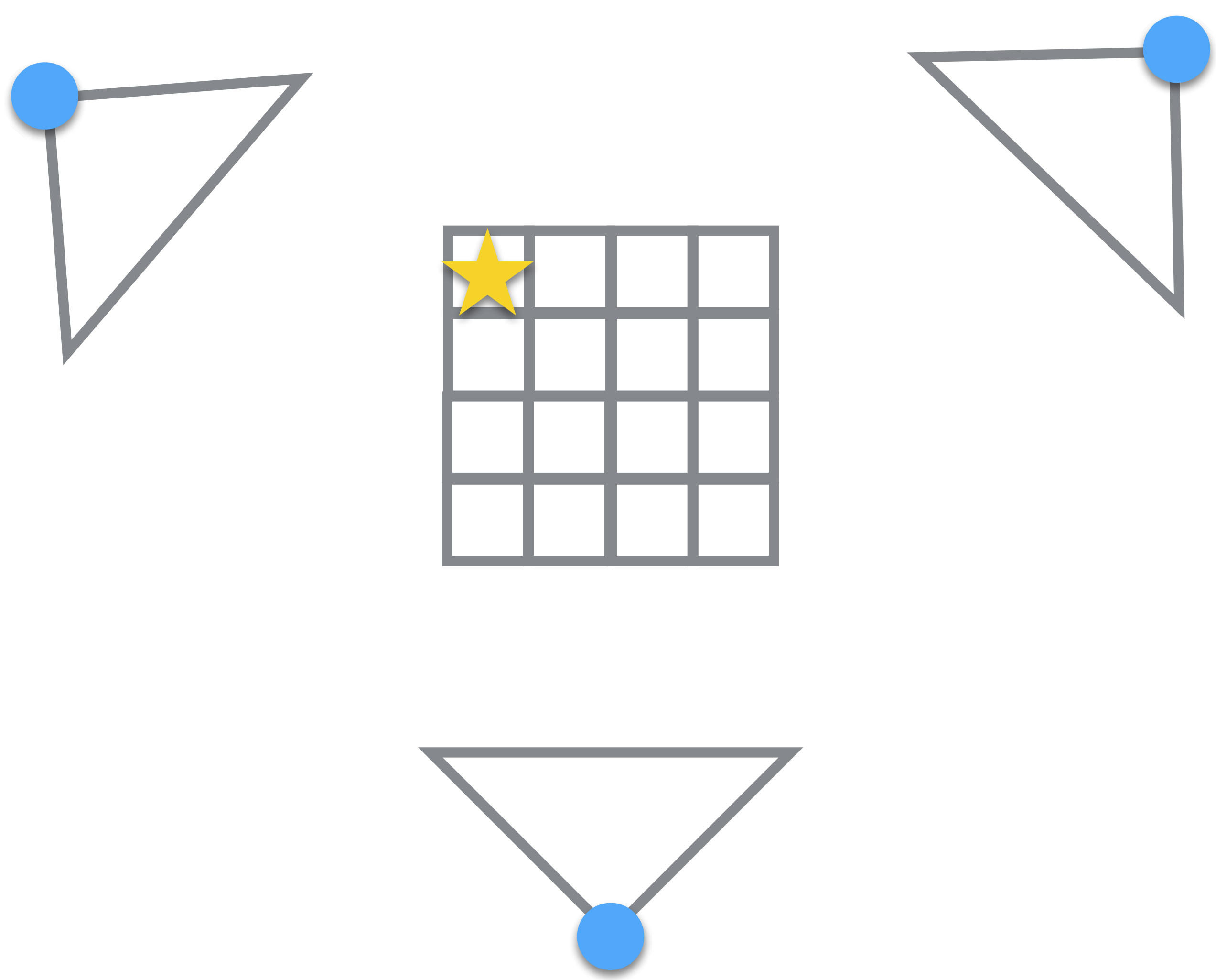
Multi-View Stereo: Space Carving



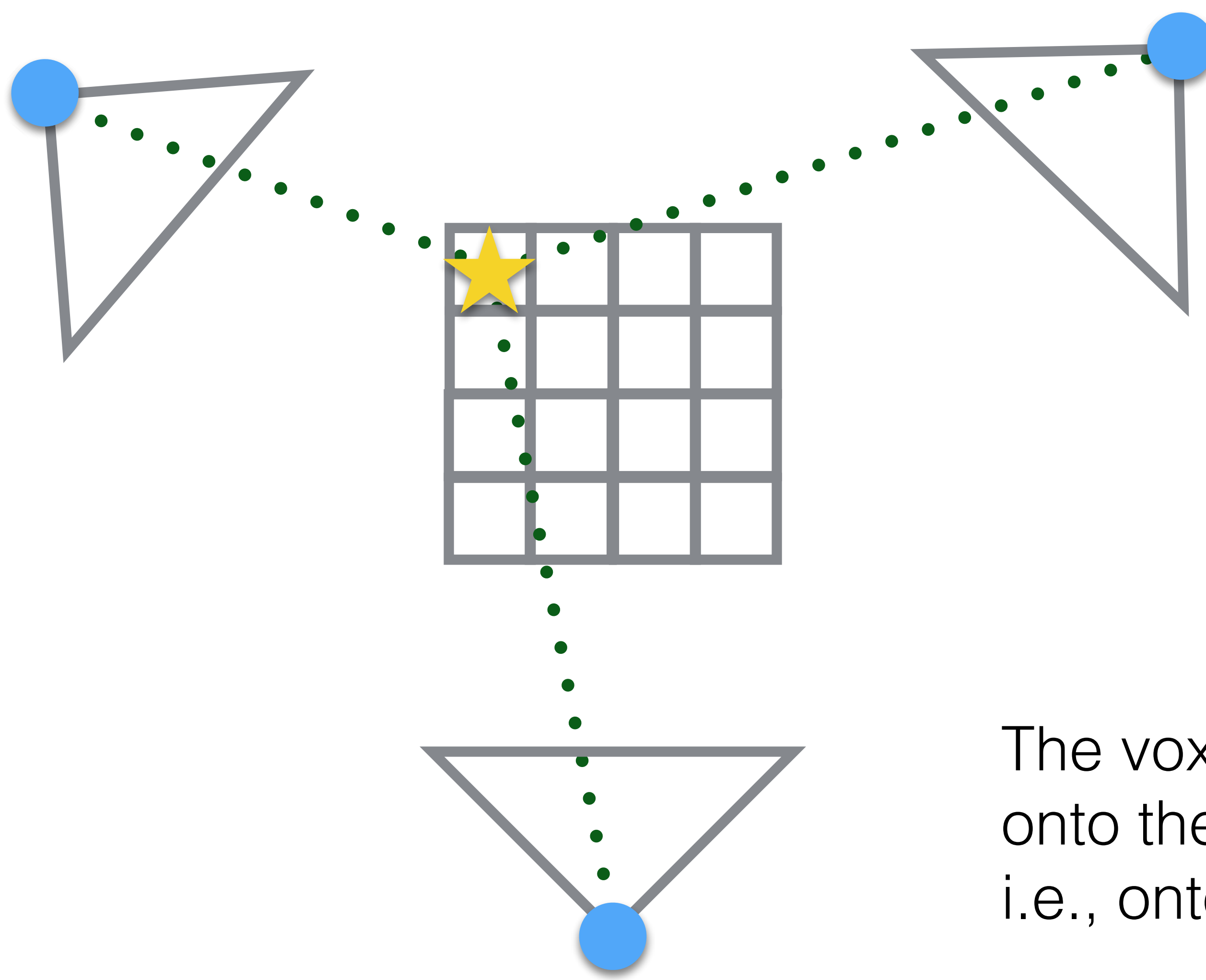
Multi-View Stereo: Space Carving



Multi-View Stereo: Space Carving

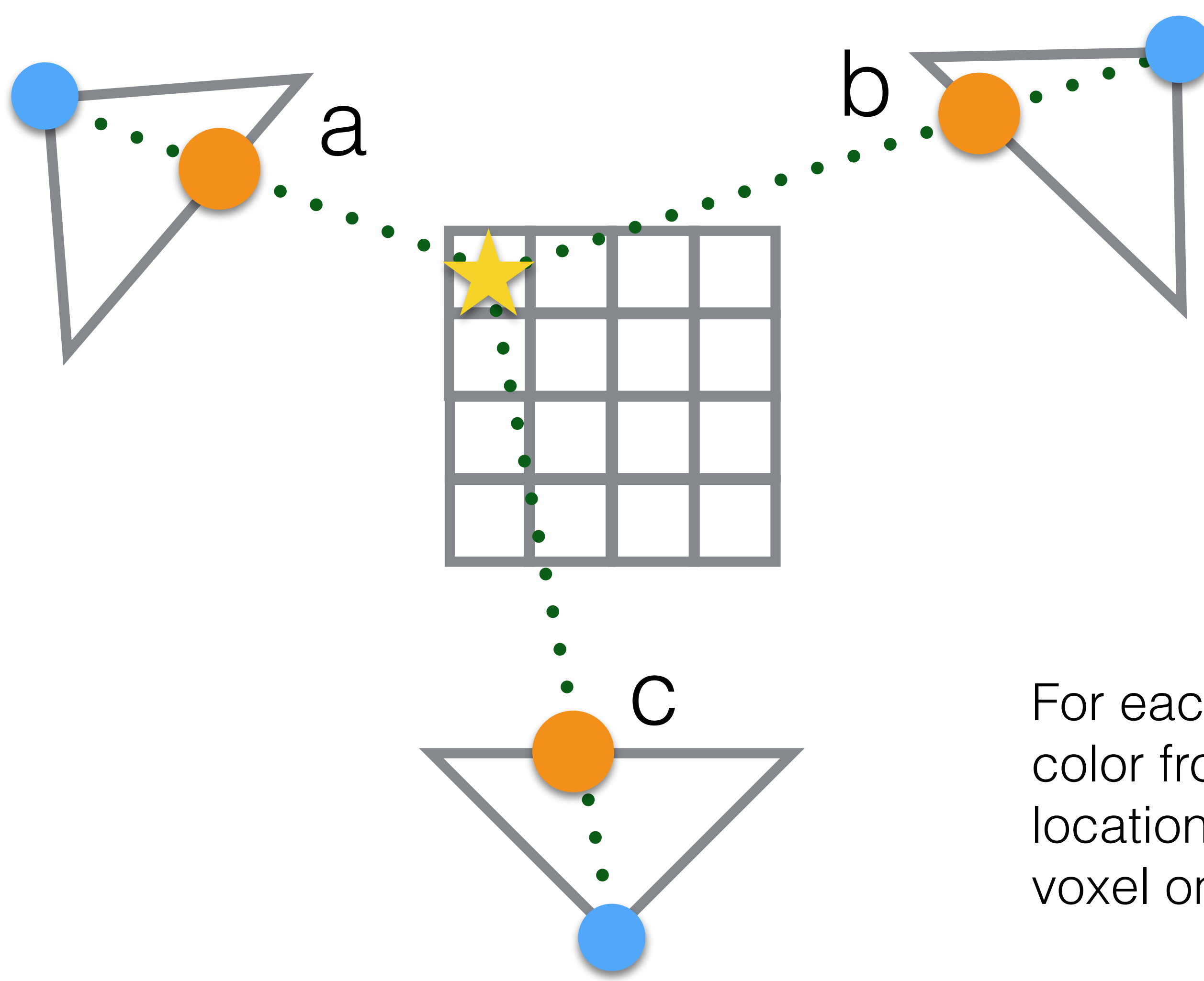


Multi-View Stereo: Space Carving



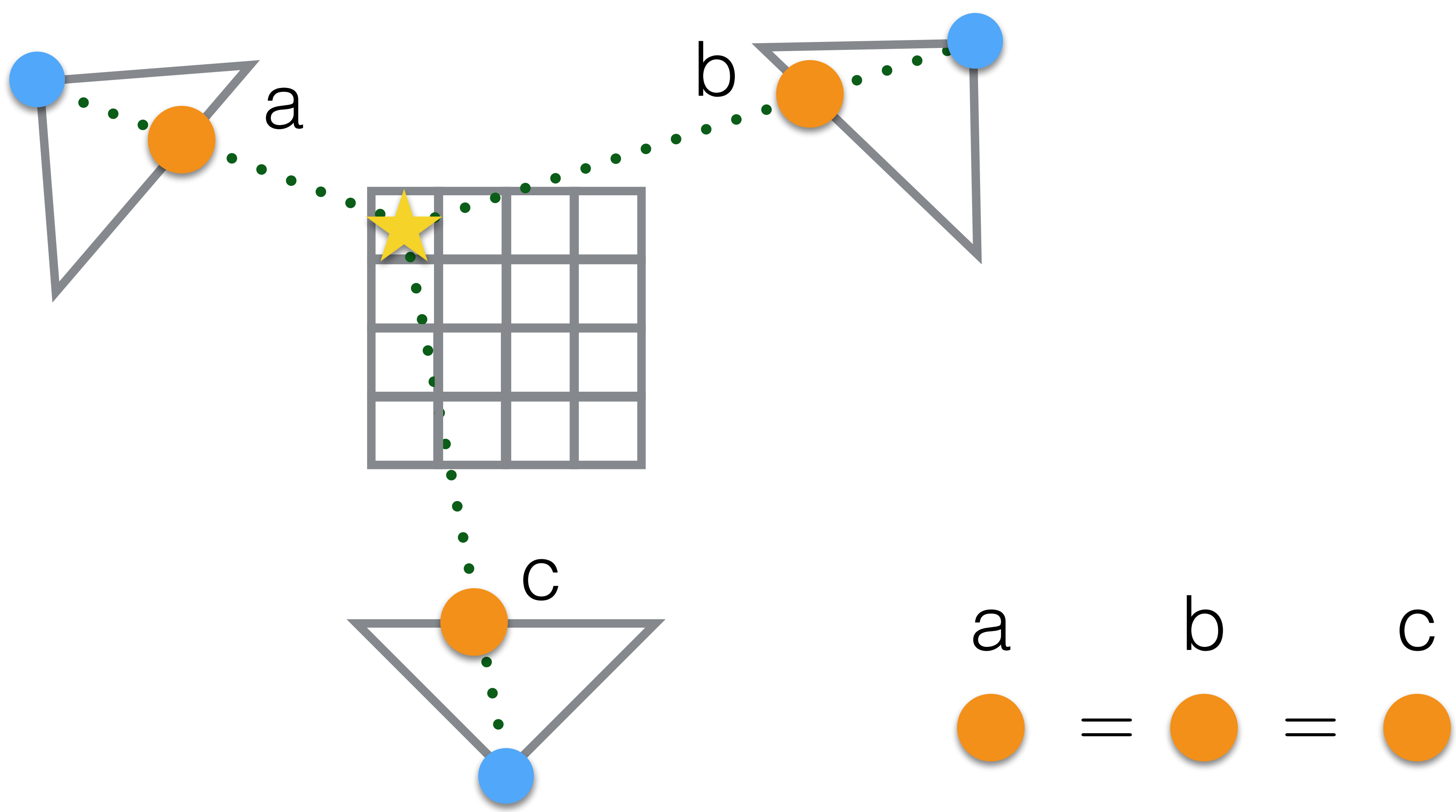
The voxel is projected
onto the calibrated camera;
i.e., onto its image-plane

Multi-View Stereo: Space Carving

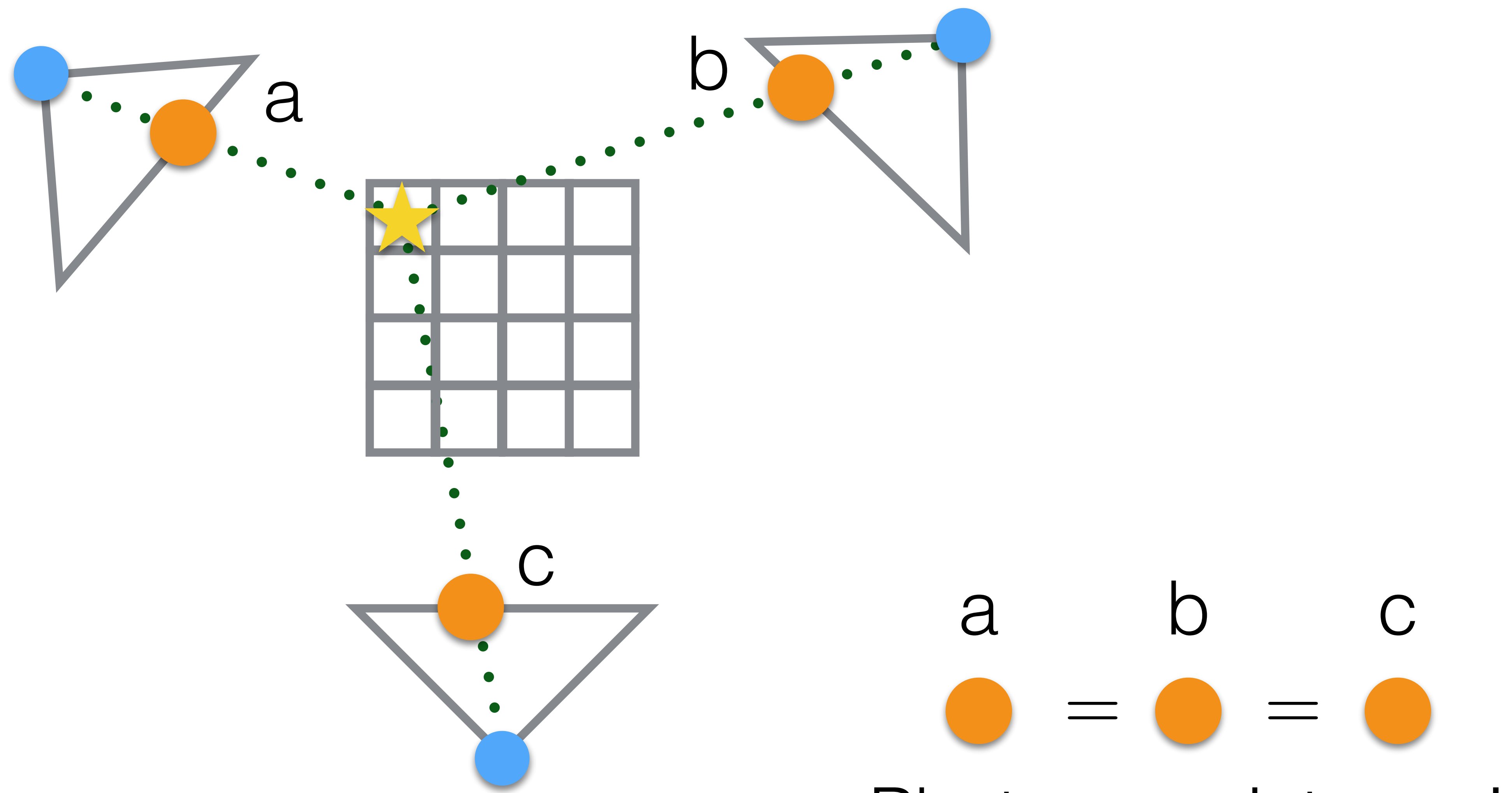


For each camera, we fetch the color from its photo at the location given by the projected voxel on its image plane.

Multi-View Stereo: Space Carving

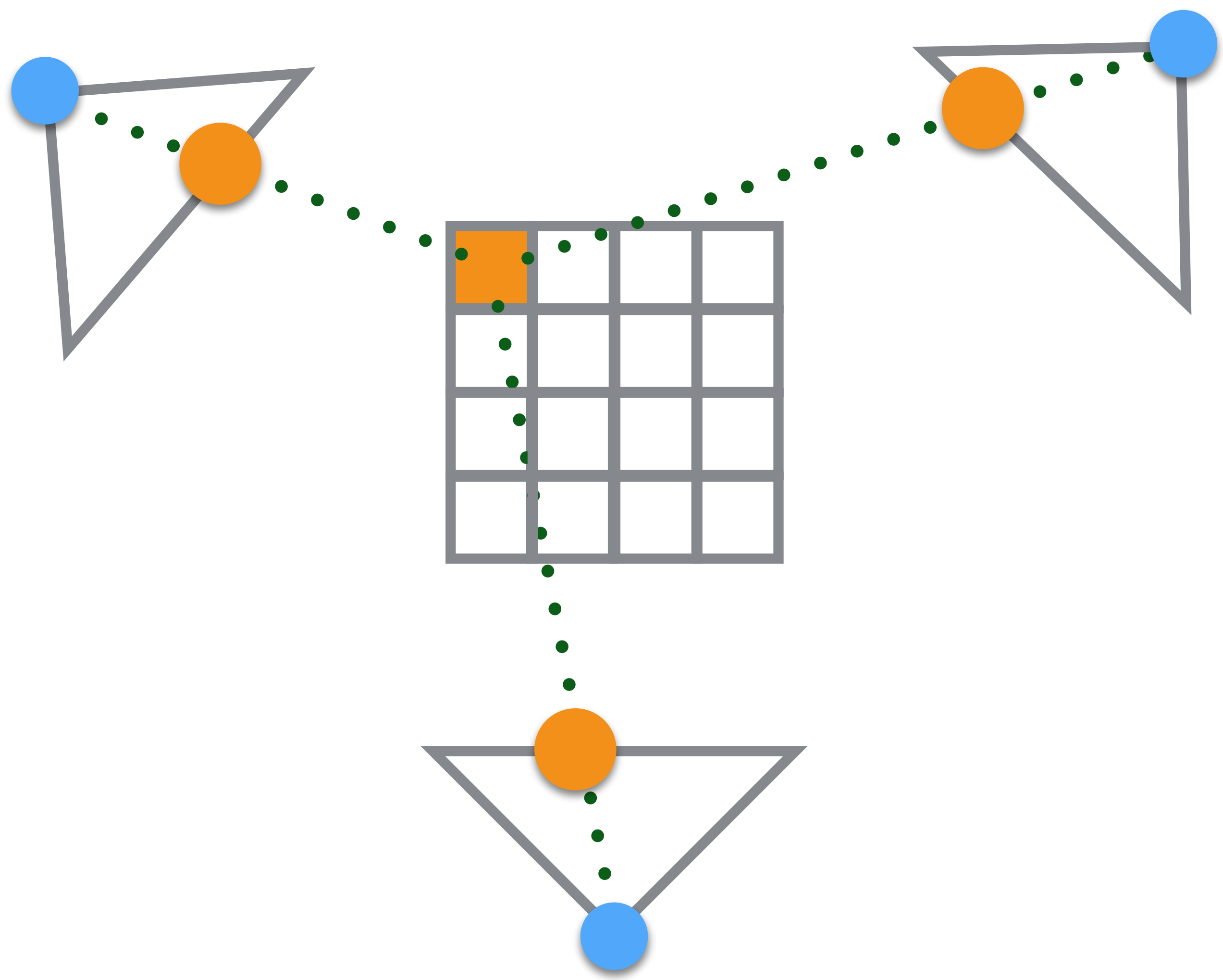


Multi-View Stereo: Space Carving

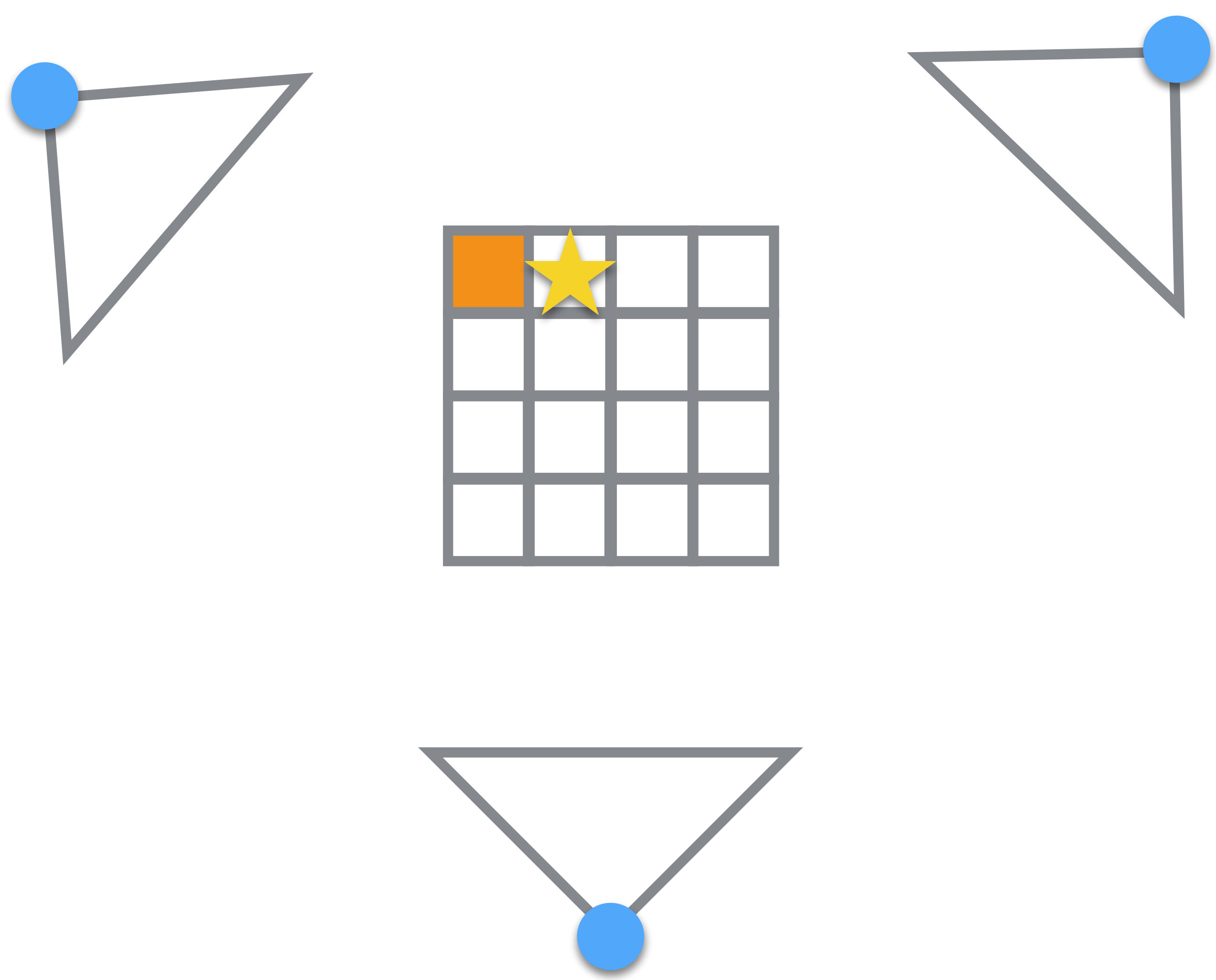


A voxel is ***photo consistent*** when all colors of its projections onto all image-planes of “*visible*” cameras appear to be similar.

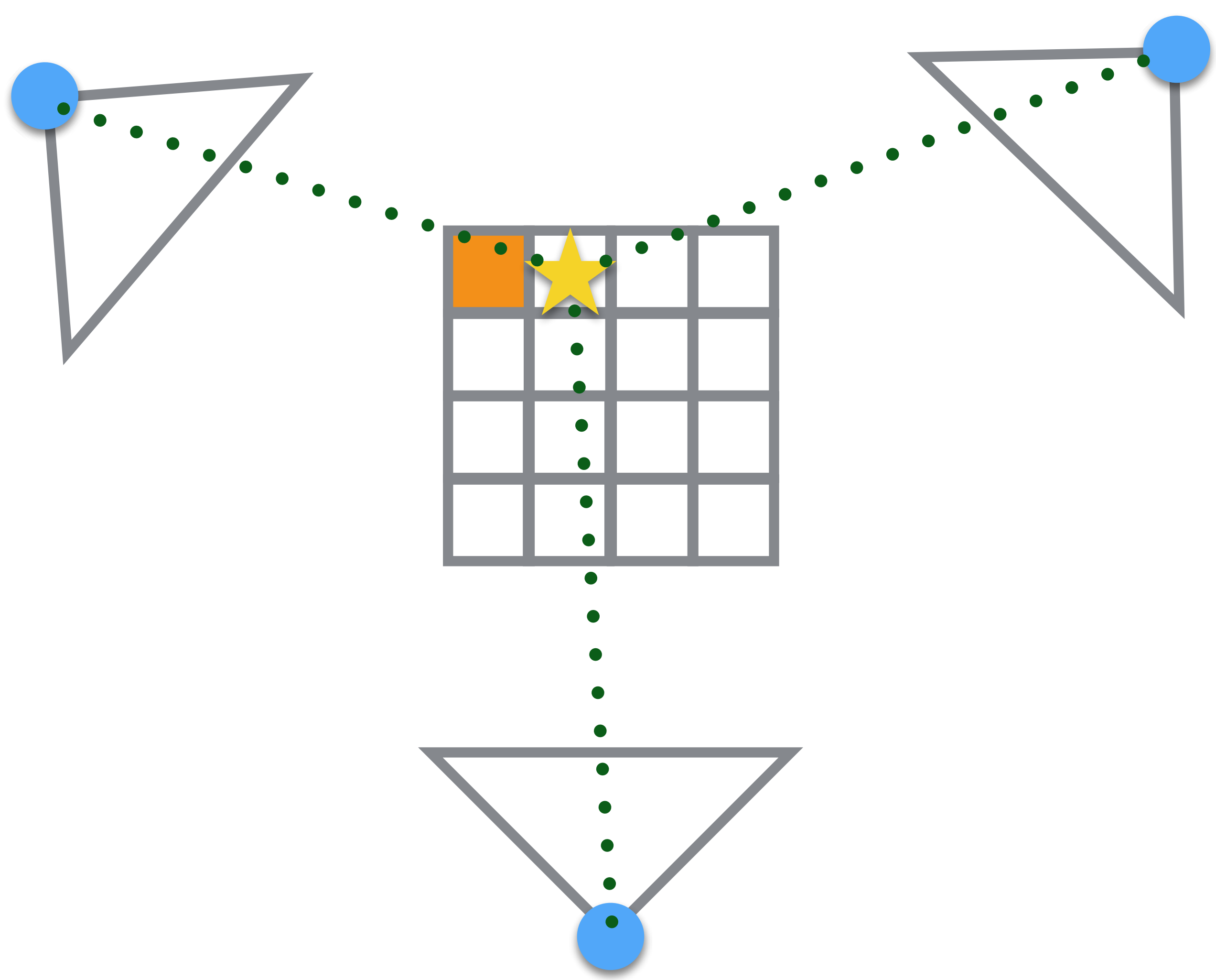
Multi-View Stereo: Space Carving



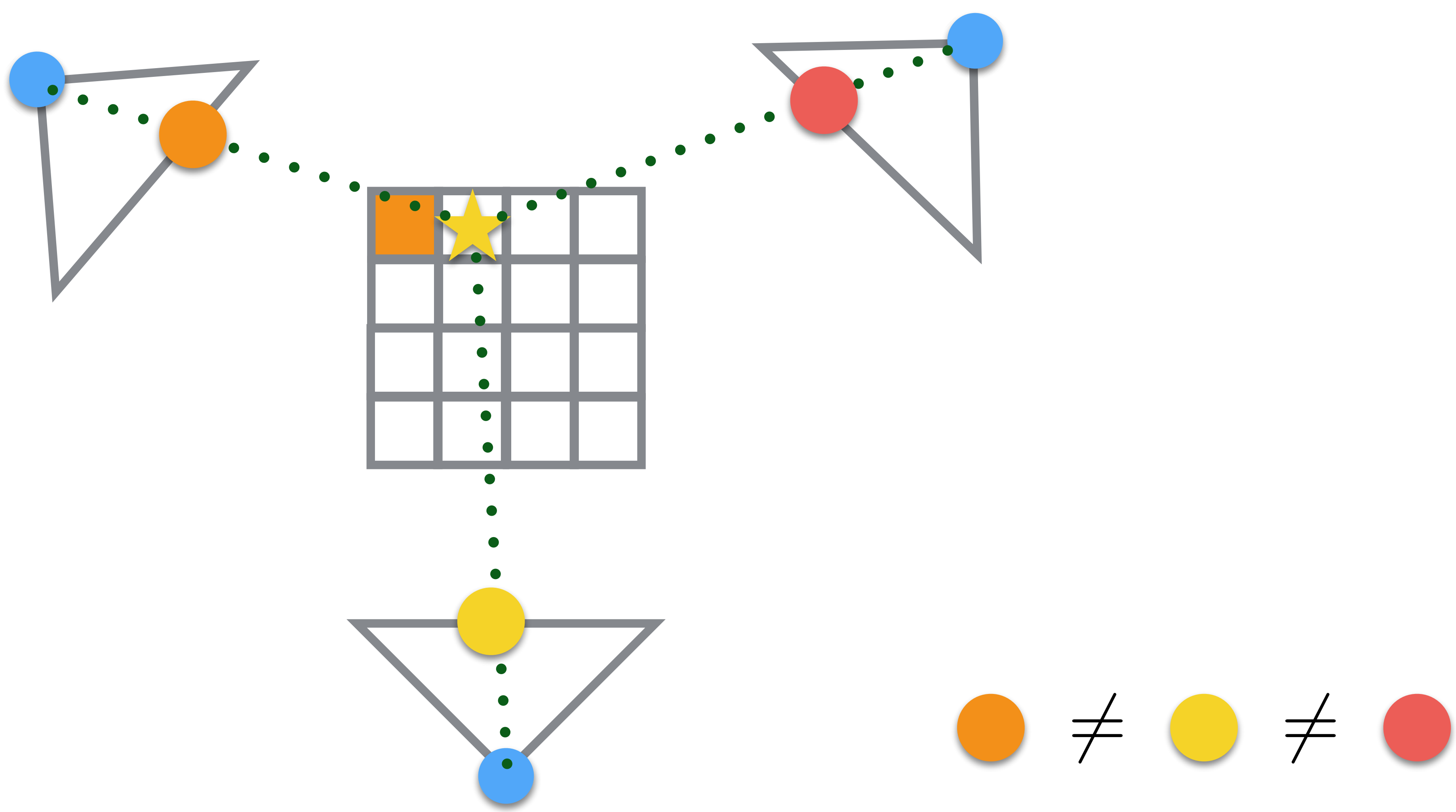
Multi-View Stereo: Space Carving



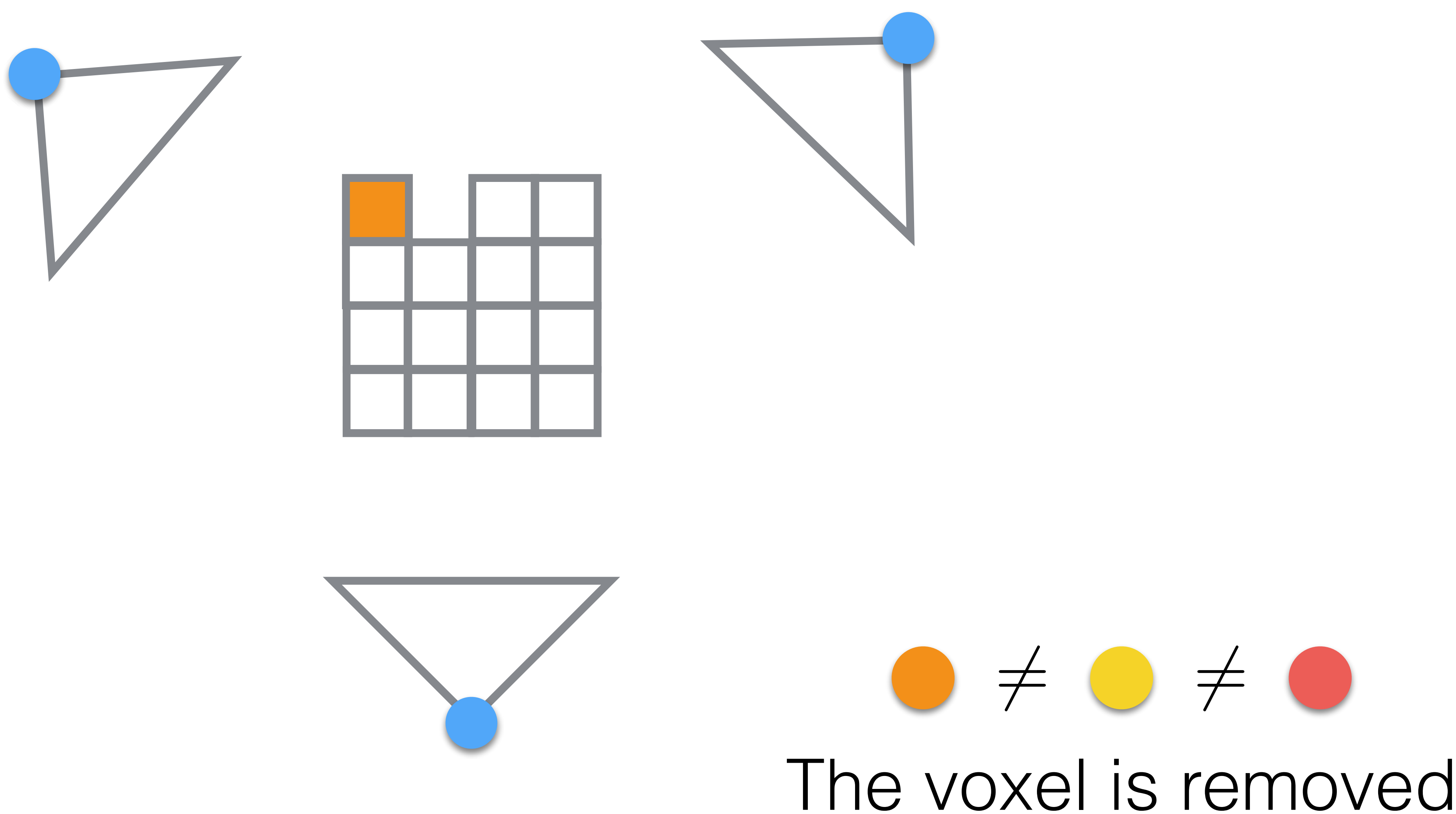
Multi-View Stereo: Space Carving



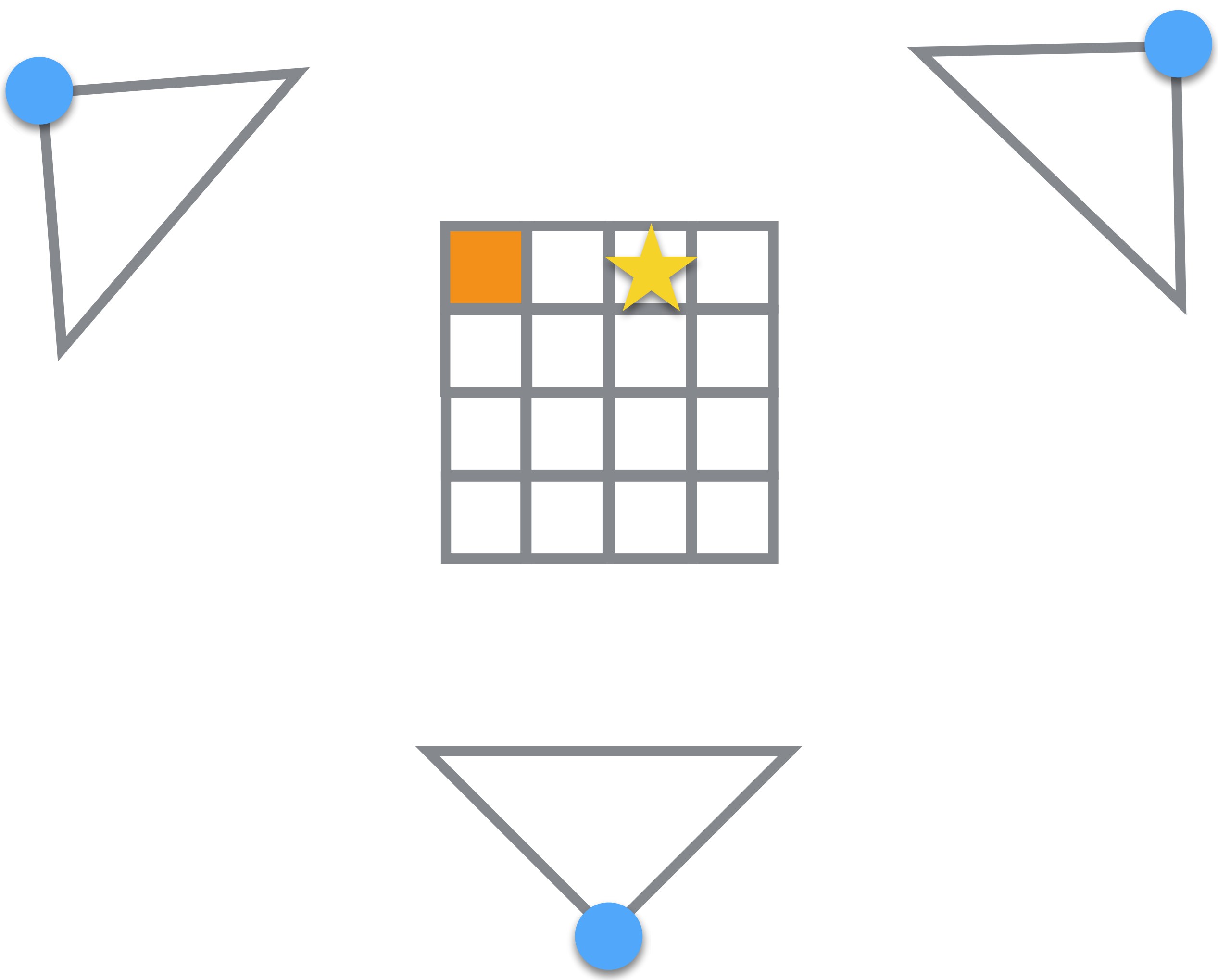
Multi-View Stereo: Space Carving



Multi-View Stereo: Space Carving



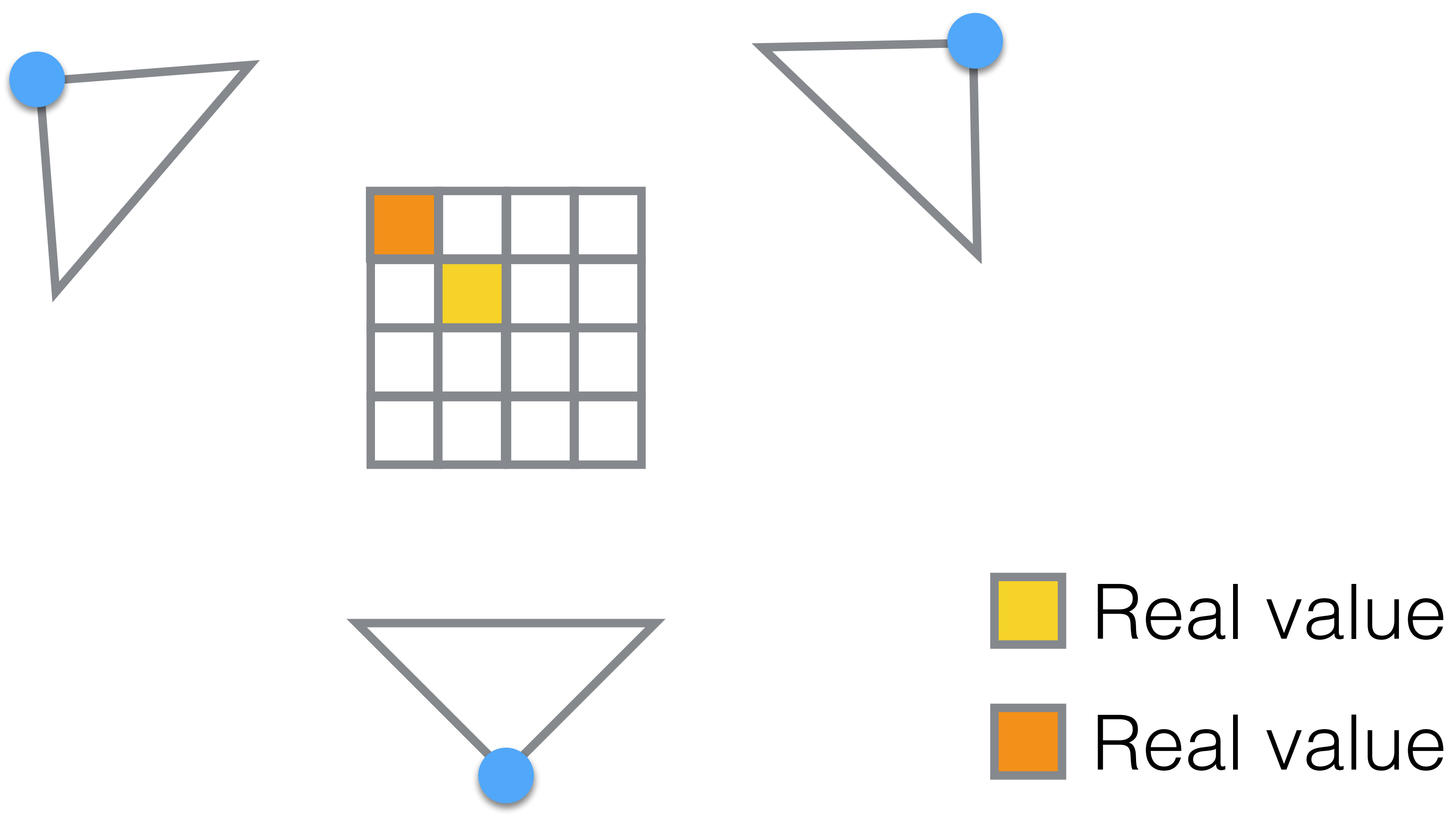
Multi-View Stereo: Space Carving



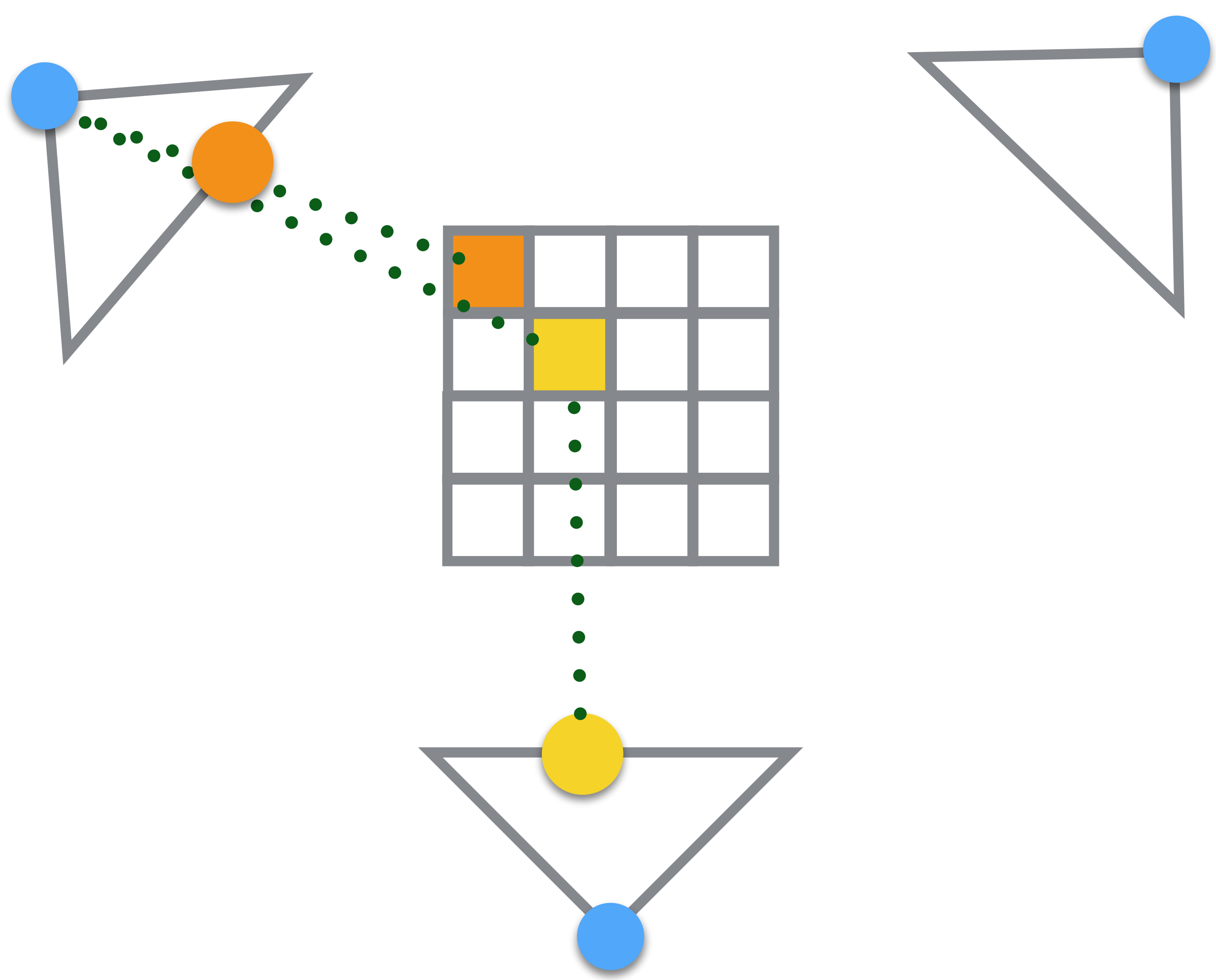
Who sees the problem here?

A voxel could be occluded, so it cannot be photo-consistent!

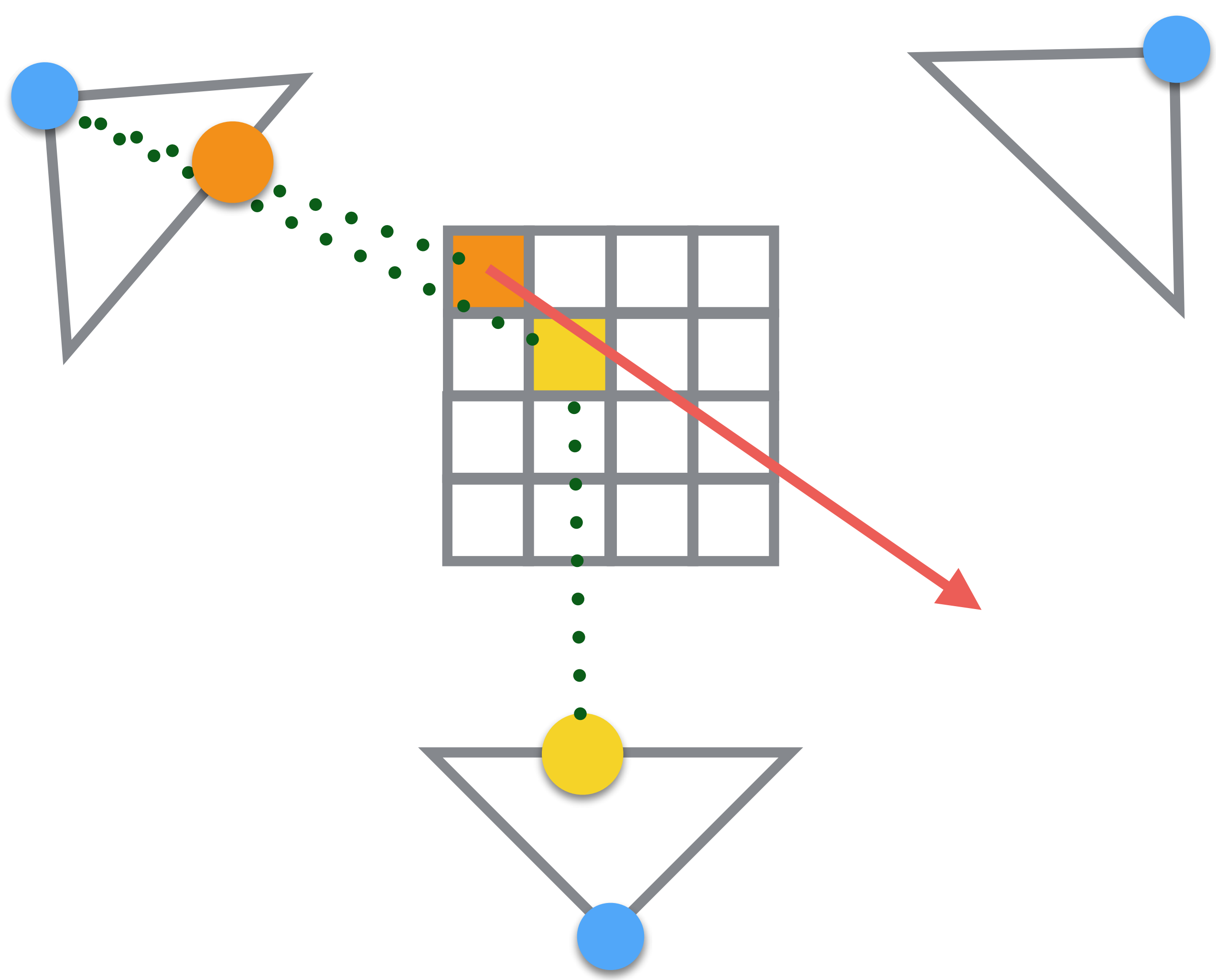
Multi-View Stereo: Space Carving



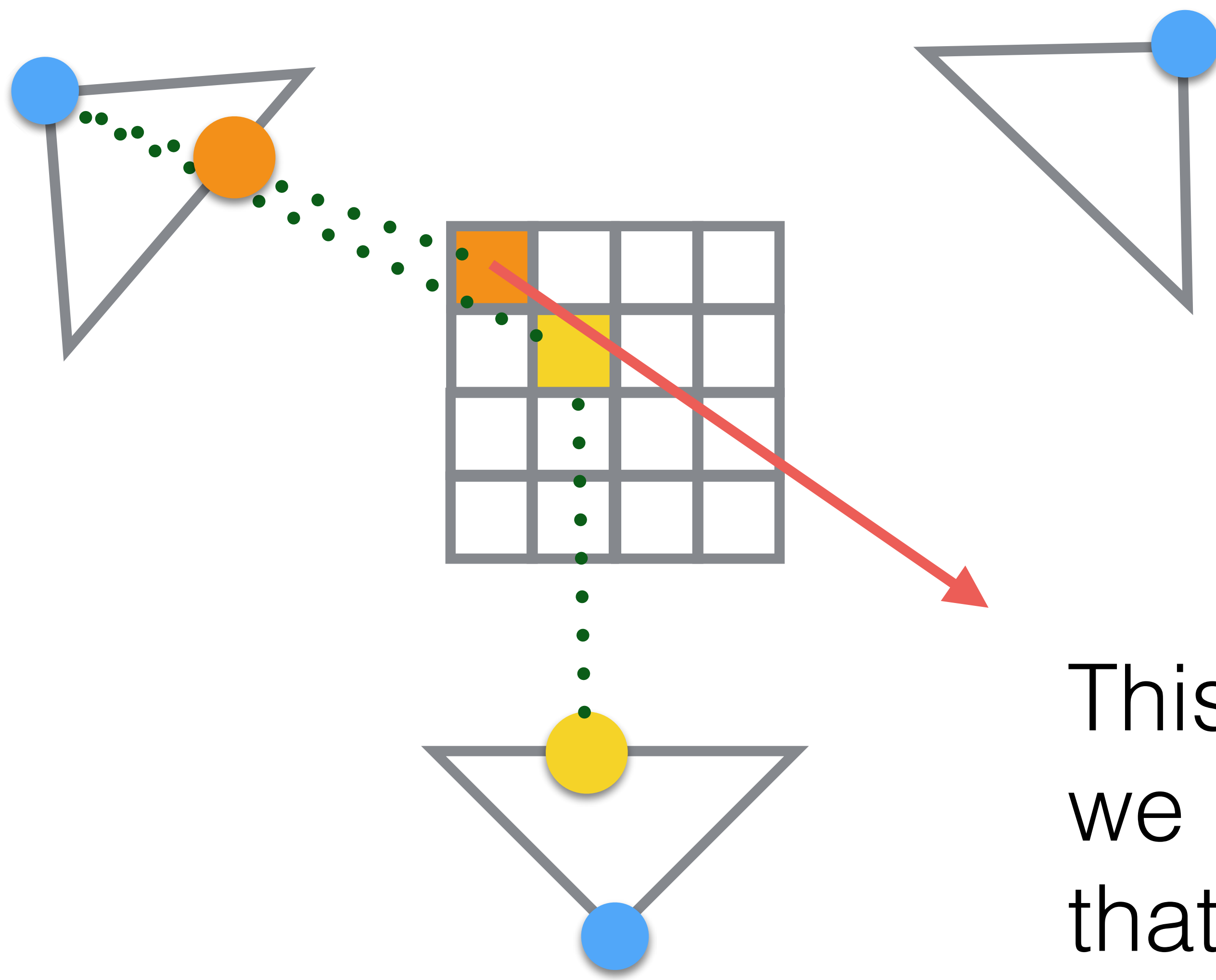
Multi-View Stereo: Space Carving



Multi-View Stereo: Space Carving



Multi-View Stereo: Space Carving

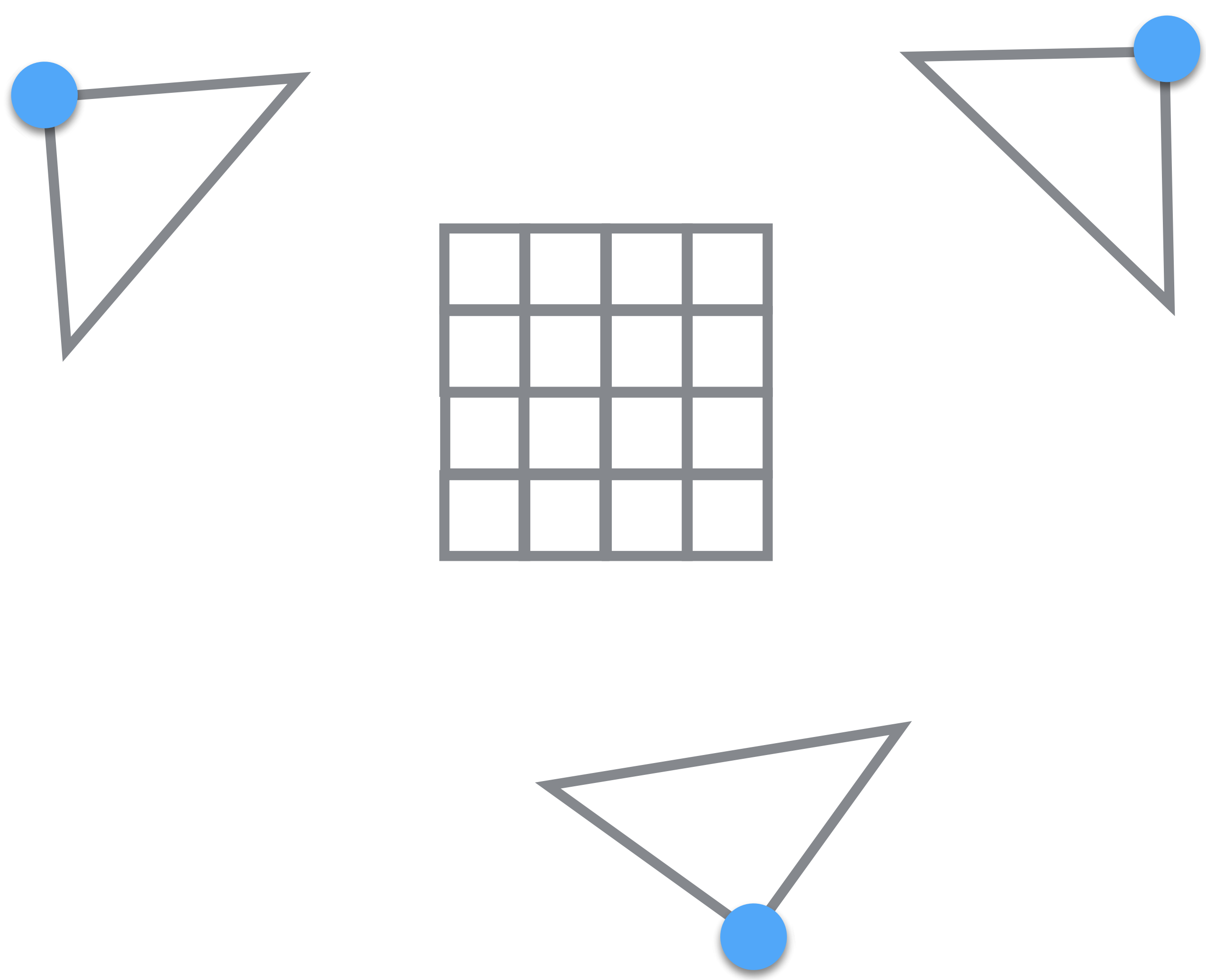


This occludes ■ ,
we may remove ■
that is real!

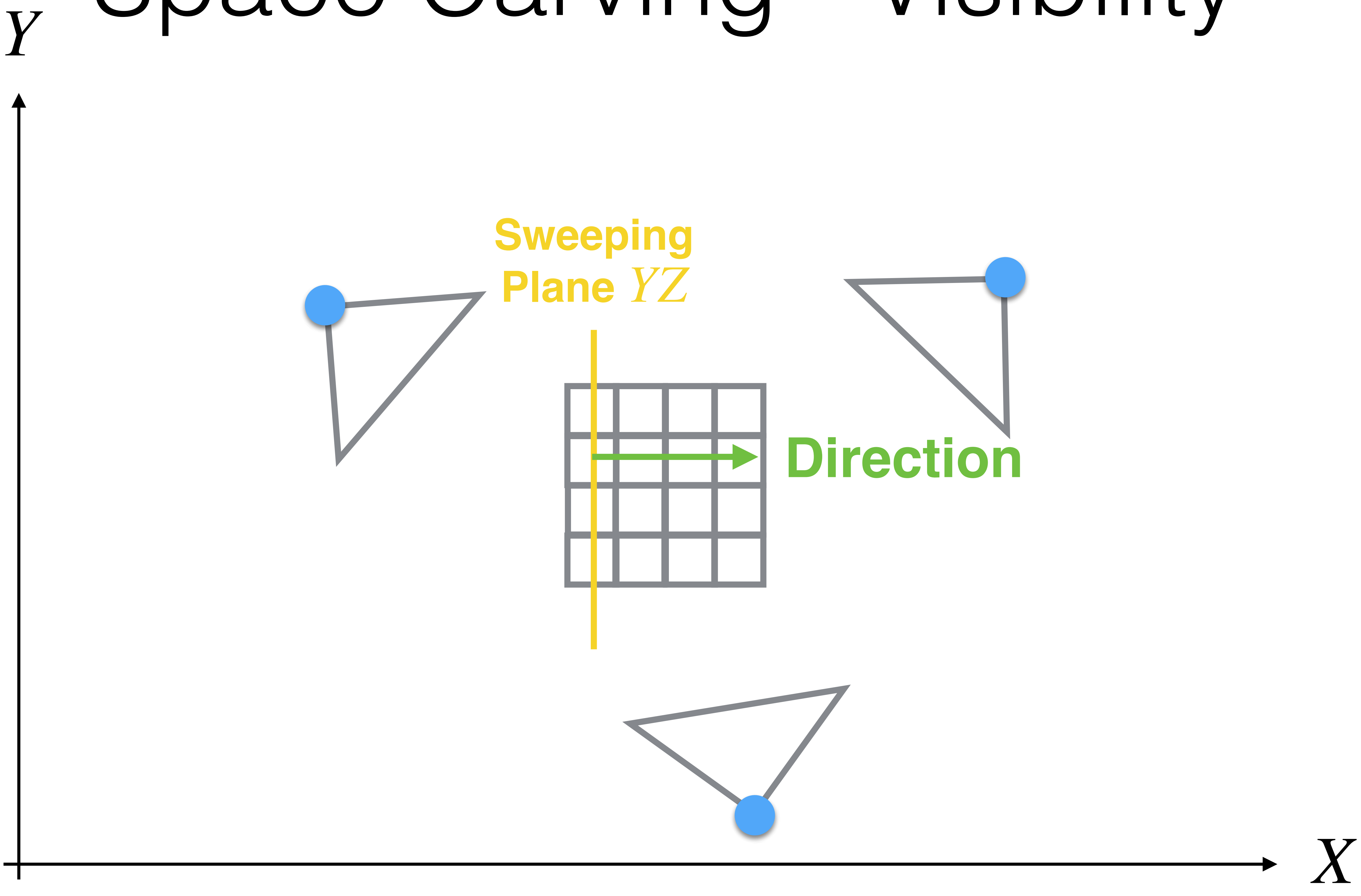
Multi-View Stereo: Space Carving

- The idea is to iterate for all six directions (back to front, and front to back) along X, Y, and Z axis.
- For each direction:
  - We sweep voxels using a plane, and every time we move the plane we determine visible views.

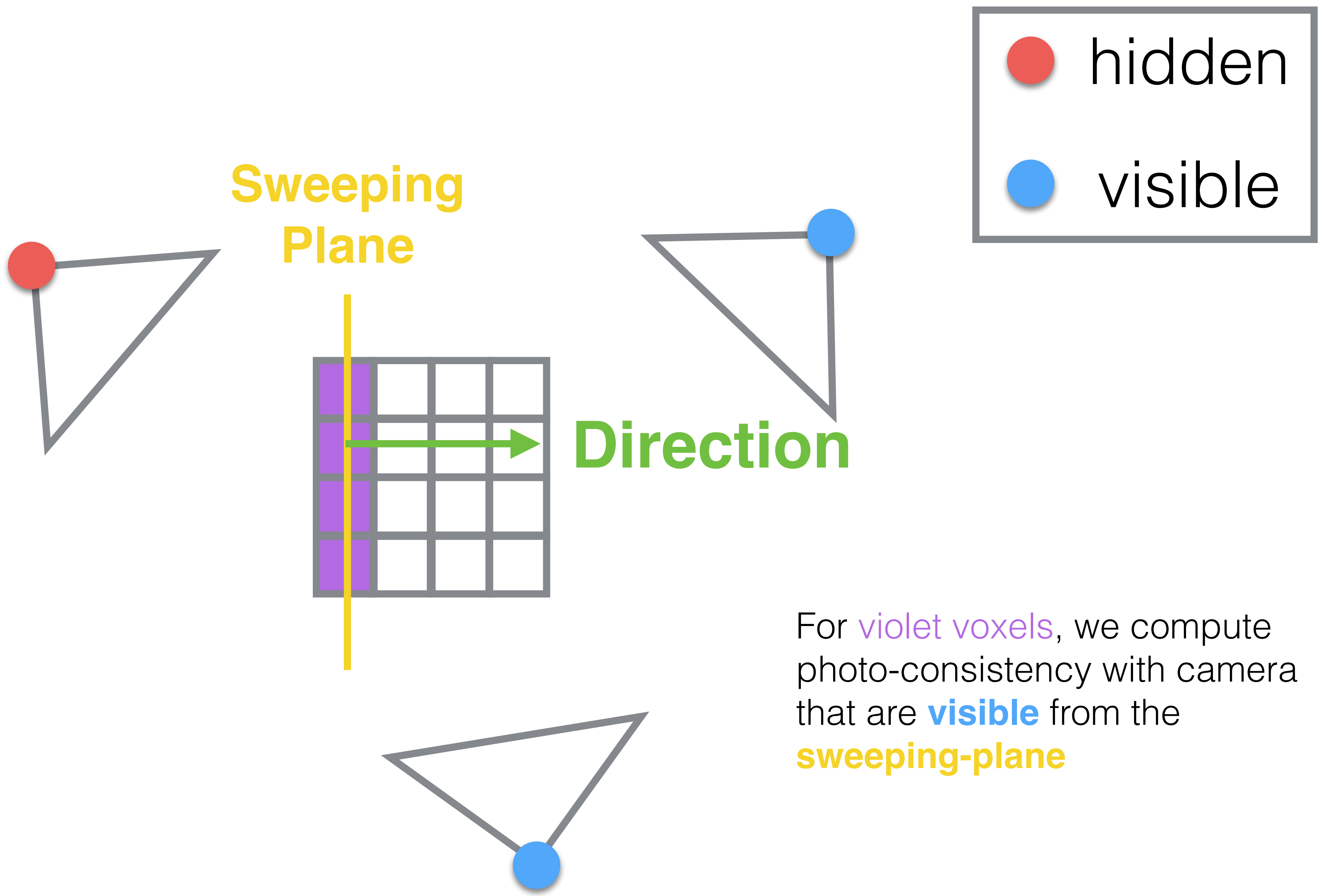
Multi-View Stereo:
Space Carving - Visibility



Multi-View Stereo: Space Carving - Visibility

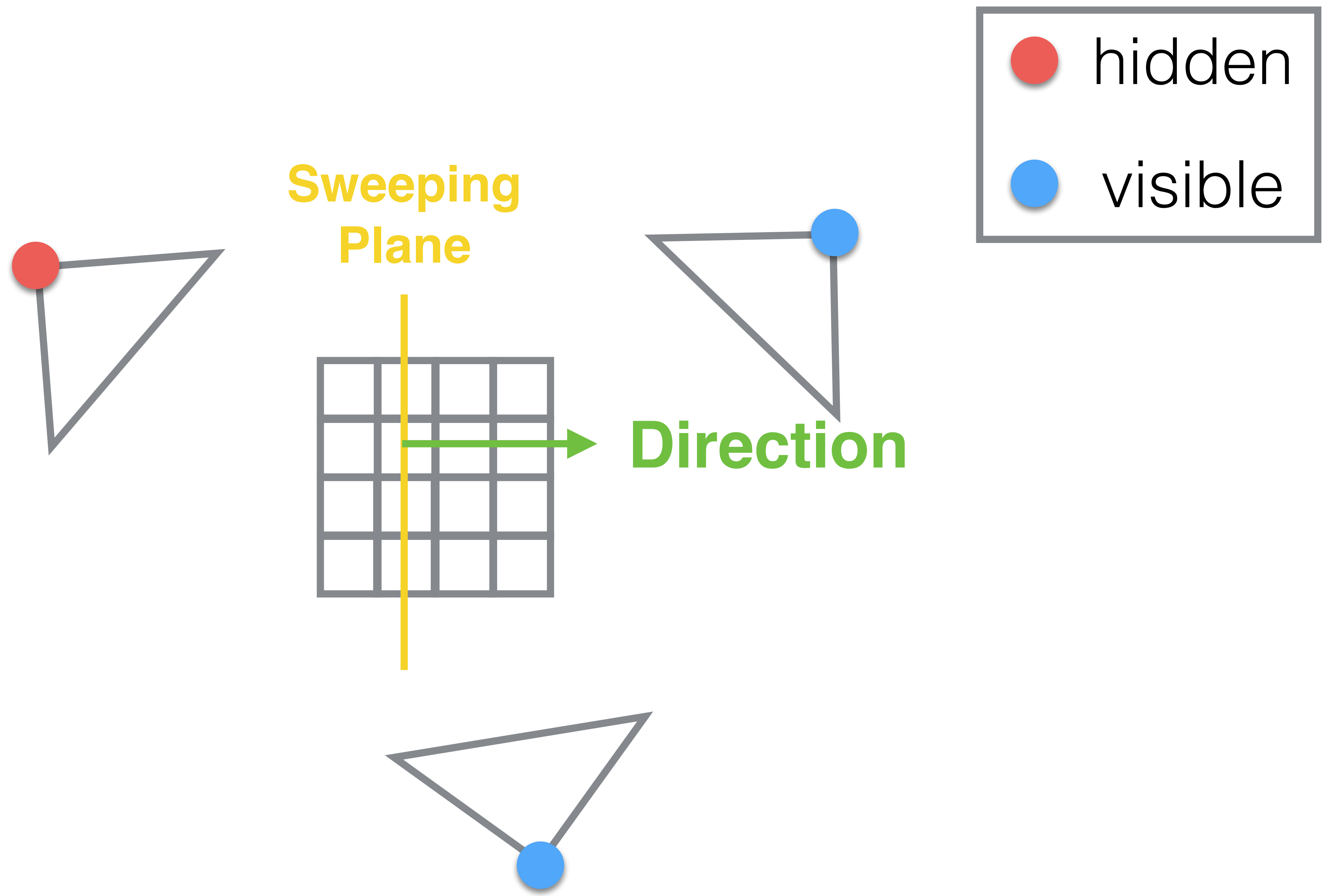


Multi-View Stereo: Space Carving - Visibility

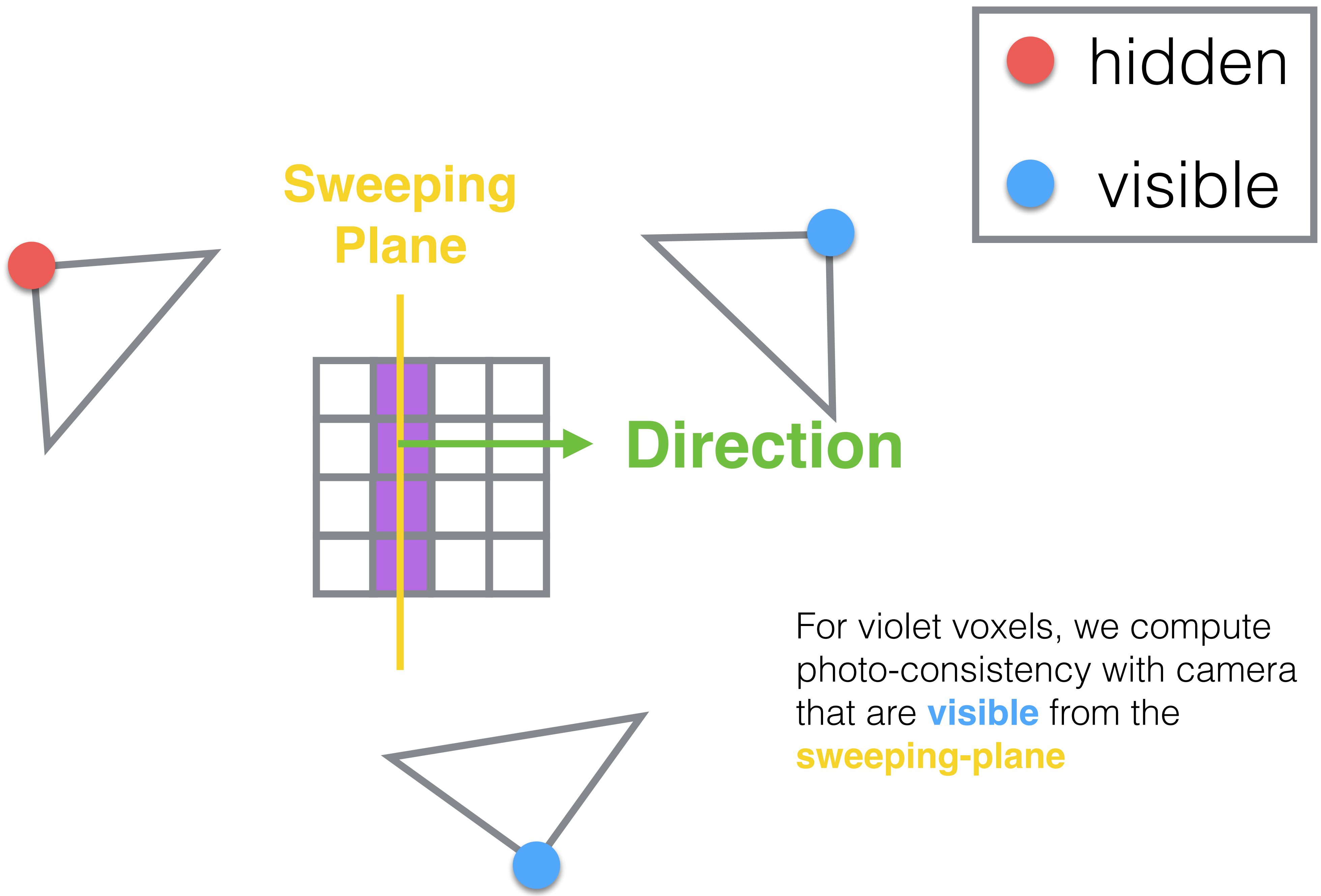


For violet voxels, we compute photo-consistency with camera that are **visible** from the **sweeping-plane**

Multi-View Stereo: Space Carving - Visibility

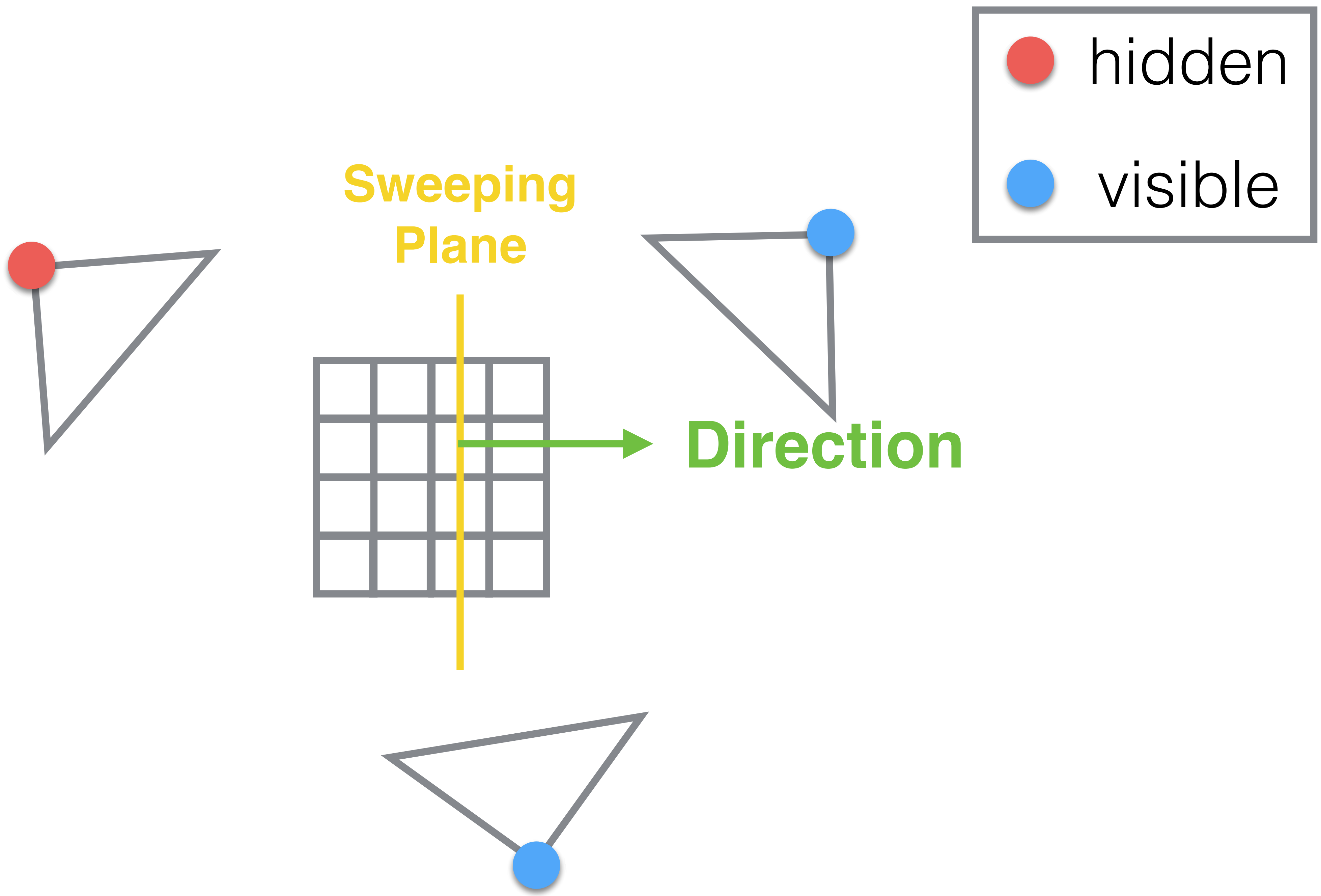


Multi-View Stereo: Space Carving - Visibility

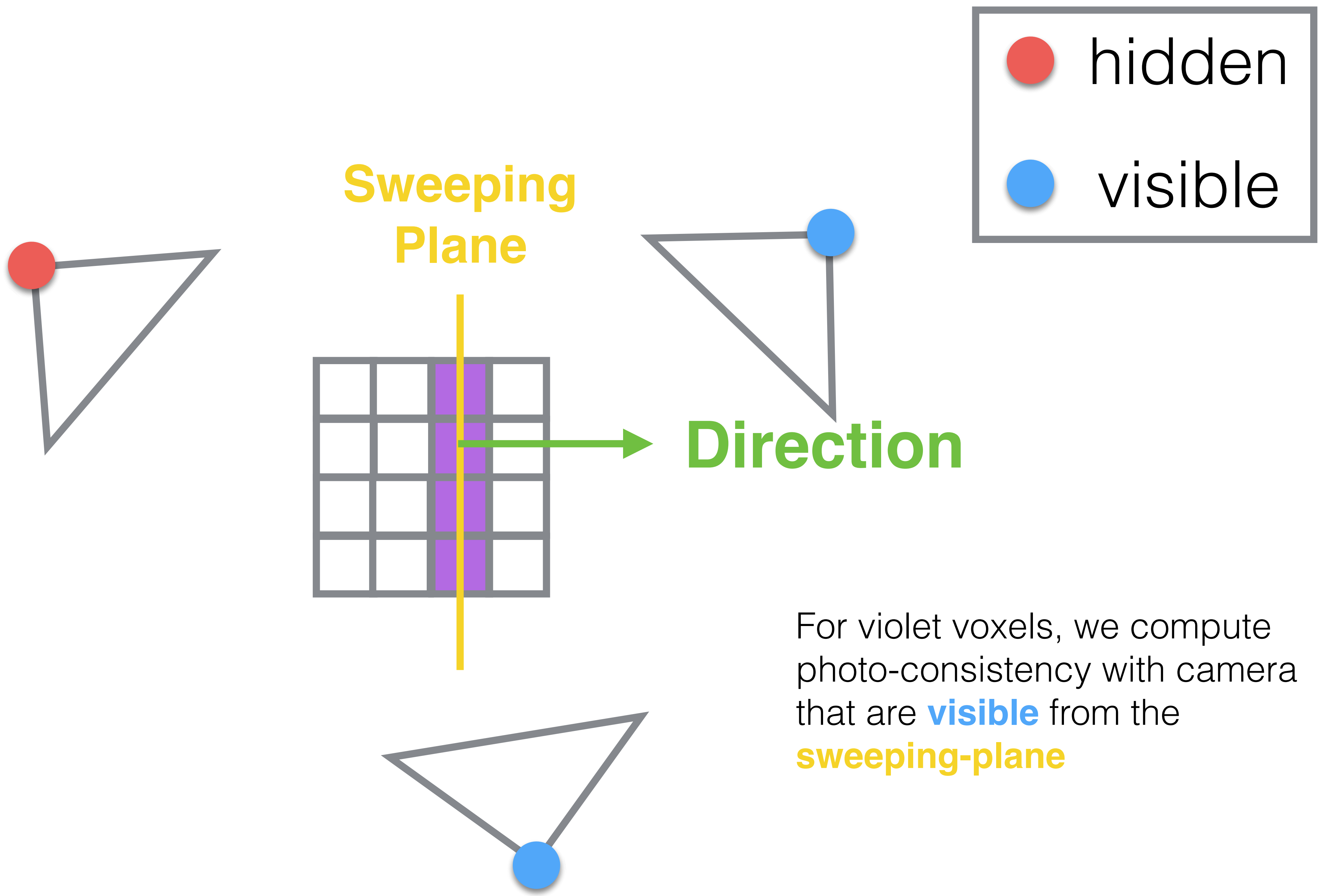


For violet voxels, we compute photo-consistency with camera that are **visible** from the **sweeping-plane**

Multi-View Stereo: Space Carving - Visibility

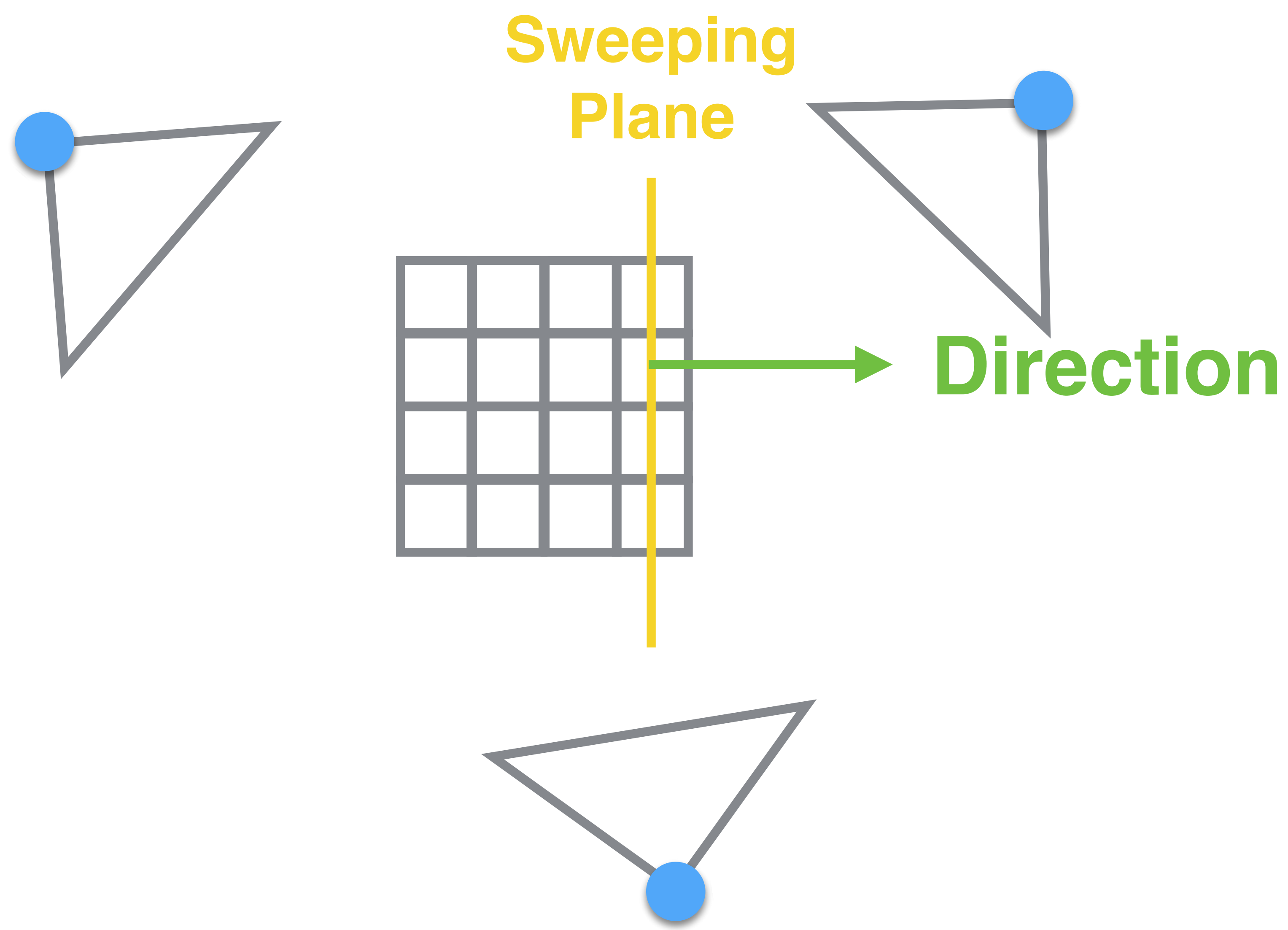


Multi-View Stereo: Space Carving - Visibility

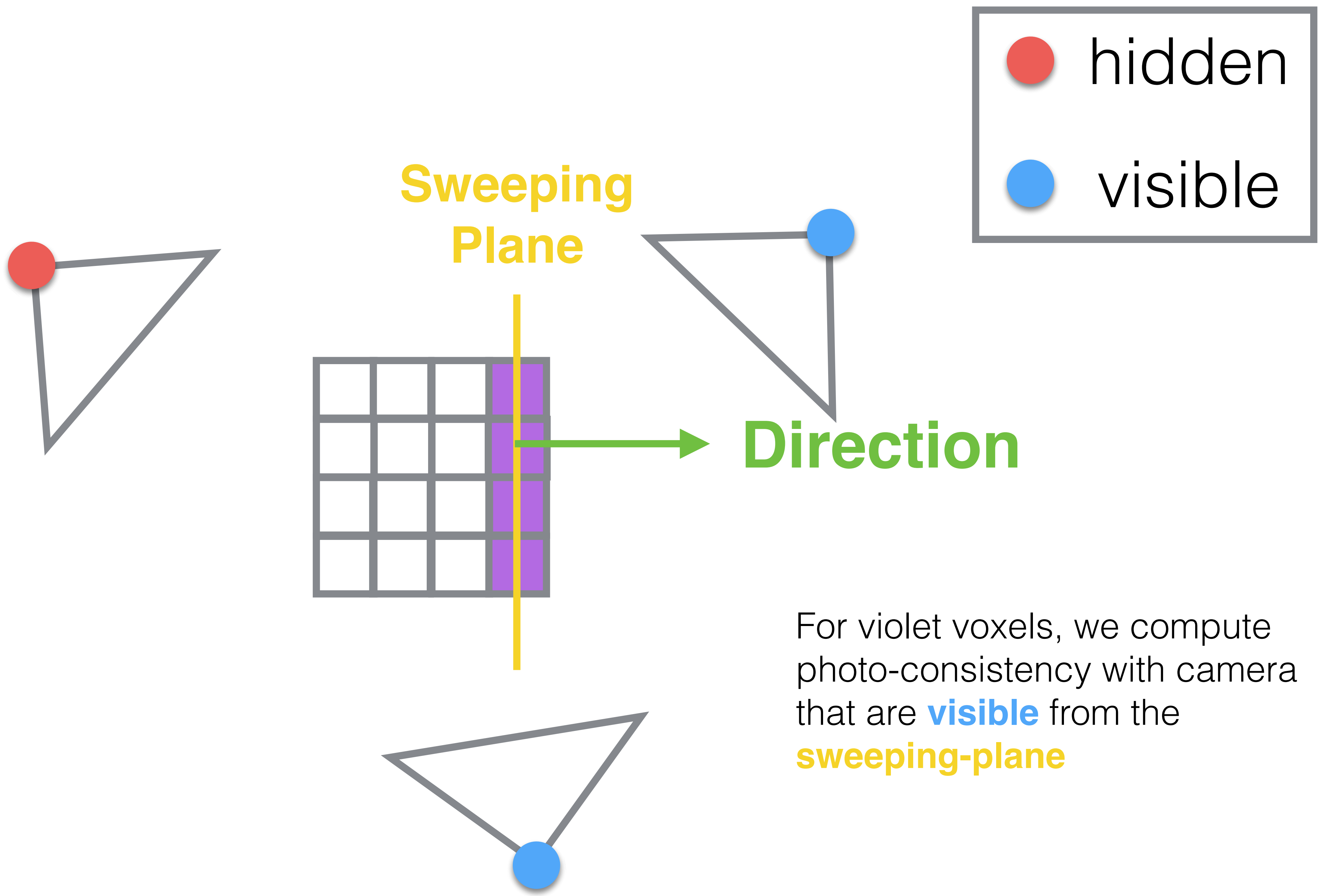


For violet voxels, we compute photo-consistency with camera that are **visible** from the **sweeping-plane**

Multi-View Stereo: Space Carving - Visibility



Multi-View Stereo: Space Carving - Visibility



For violet voxels, we compute photo-consistency with camera that are **visible** from the **sweeping-plane**

This process needs to be done again for the other direction and for Y and Z axes (both direction)!

Multi-View Stereo: Space Carving

- Once we have removed voxels, which are no photo-consistent, we run marching cubes to get the final model!

Multi-View Stereo: Space Carving Result

Photographs

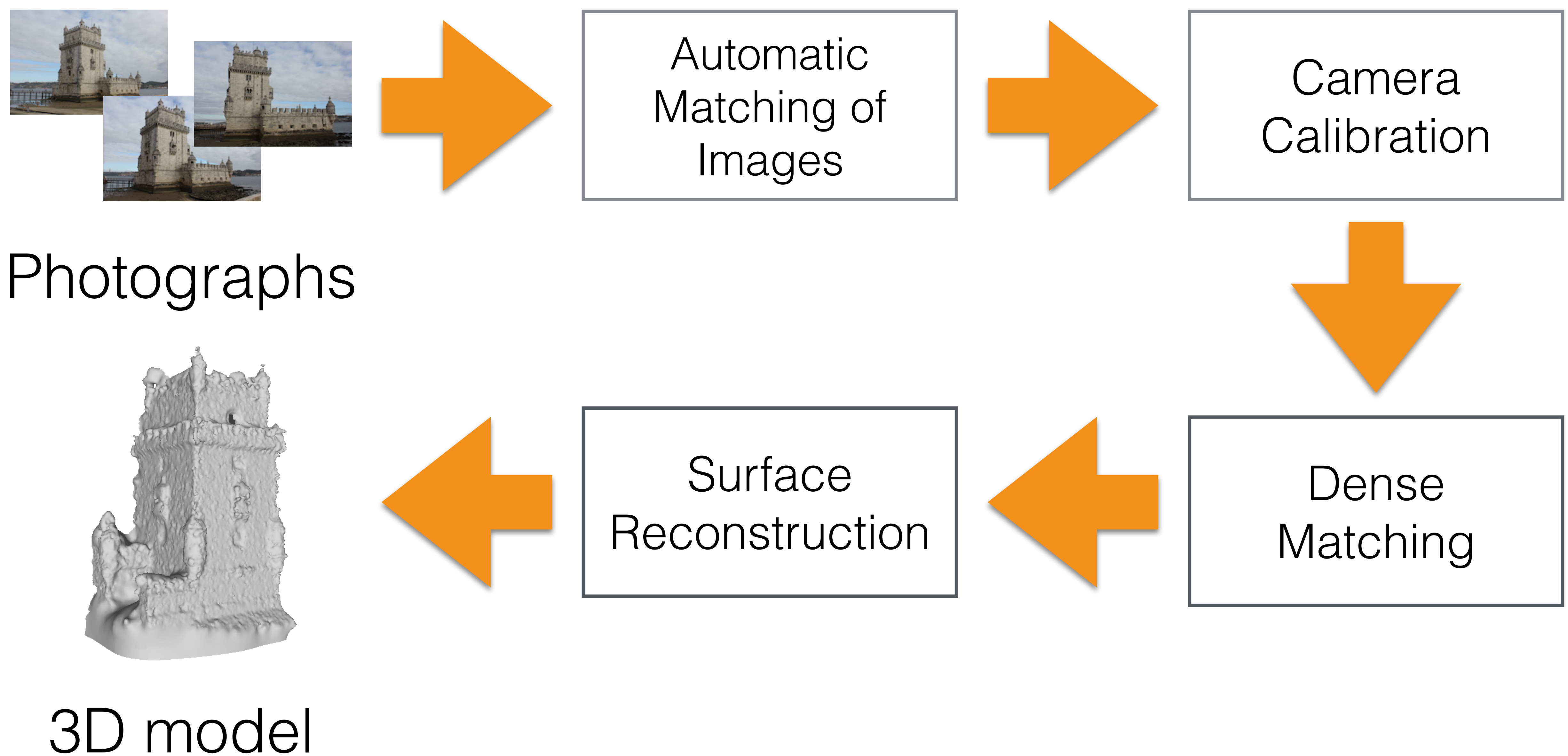
3D Model

Images courtesy of Kutulakos and Seitz

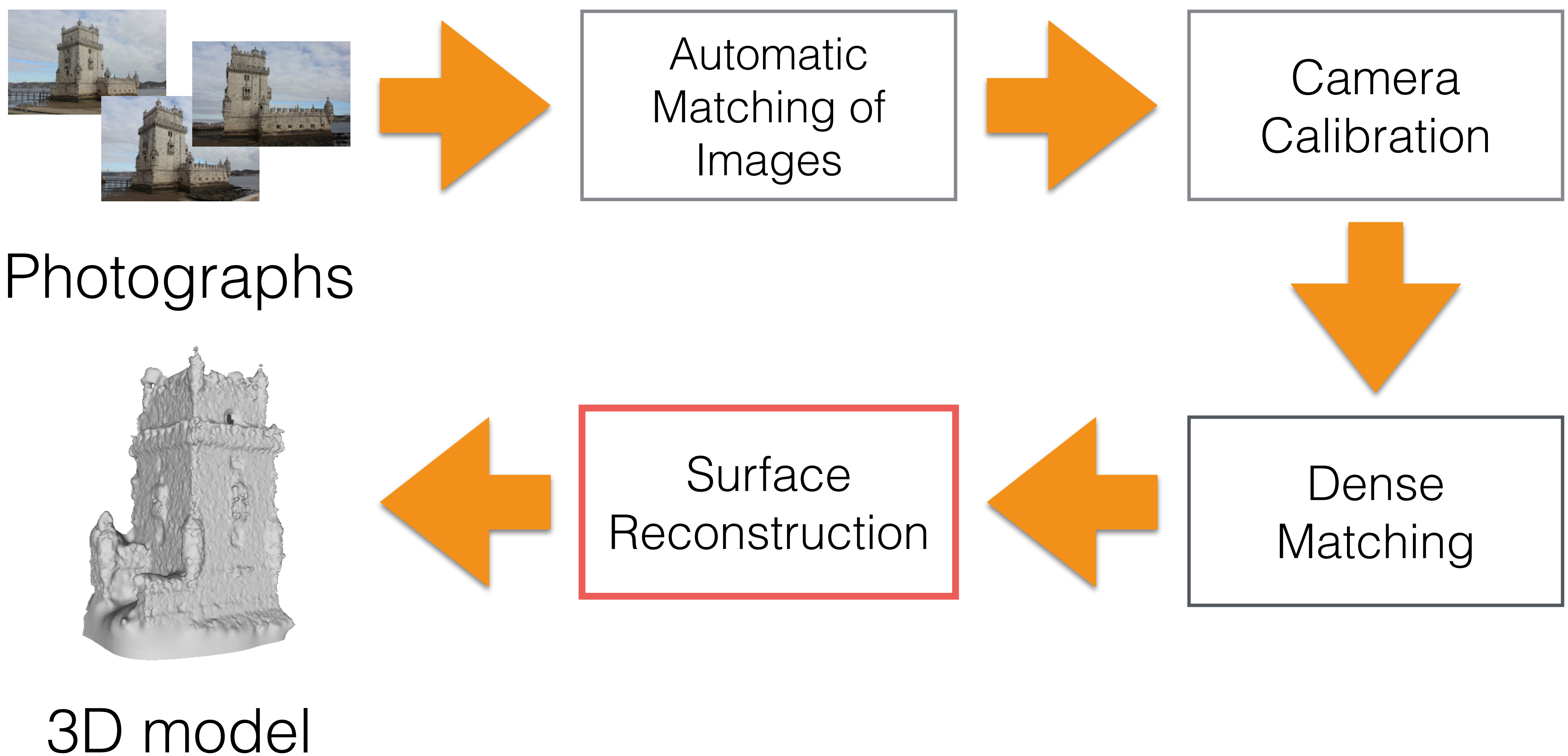
Multi-View Stereo: Space Carving

- Advantages:
  - Simple and easy to implement.
- Disadvantages:
  - It requires a lot of memory for high-quality models.

3D from Photographs



3D from Photographs



3D Reconstruction

- How do we merge all these dense points?
  - Marching cubes.
  - Poisson reconstruction.

3D Reconstruction

- Marching cubes:
  - Advantages:
    - It is fast and easy to implement.
    - It does not require to compute normals.
  - Disadvantages:
    - It requires to discretize the space using many voxels!
    - Poor results.

3D Reconstruction

- Poisson Reconstruction:
  - Advantages:
    - It creates high-quality results.
    - It can close holes.
  - Disadvantages:
    - We need to compute normals.
    - It requires both memory and time.

that’s all folks!