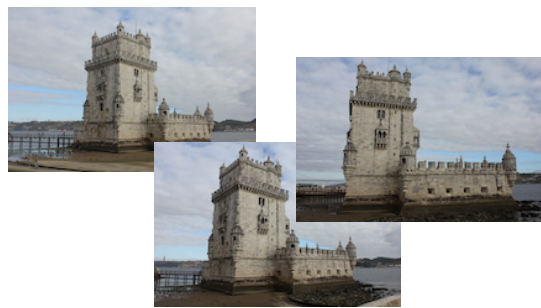


# 3D from Photographs: Stereo

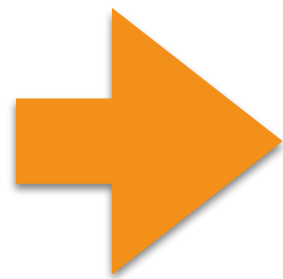
Dr Francesco Banterle  
[francesco.banterle@isti.cnr.it](mailto:francesco.banterle@isti.cnr.it)

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

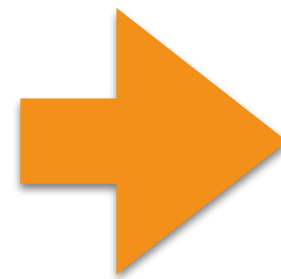
# 3D from Photographs



Photographs



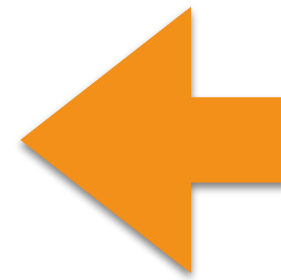
Automatic  
Matching of  
Images



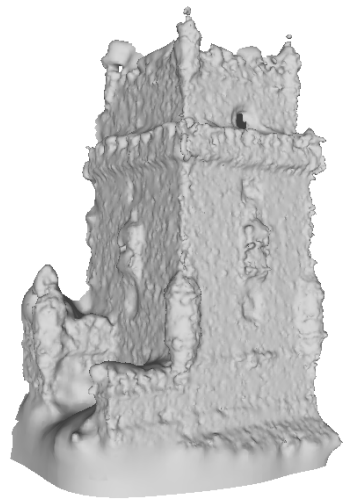
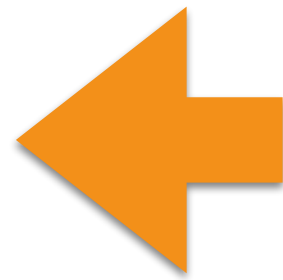
Camera  
Calibration



Dense  
Matching

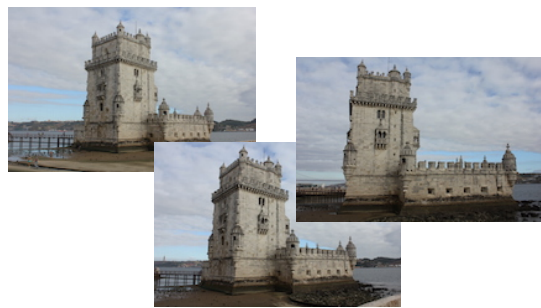


Surface  
Reconstruction



3D model

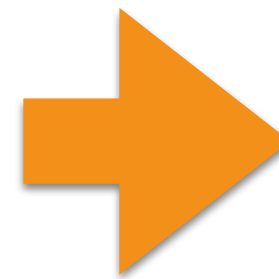
# 3D from Photographs



Photographs



Automatic  
Matching of  
Images



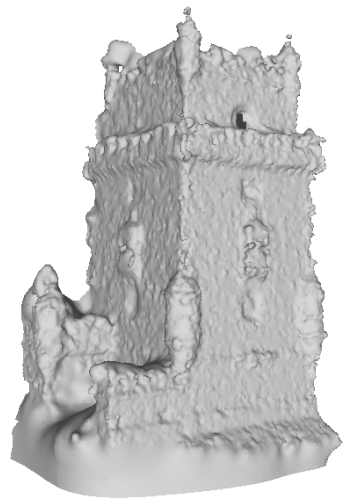
Camera  
Calibration



Dense  
Matching



Surface  
Reconstruction



3D model



# 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

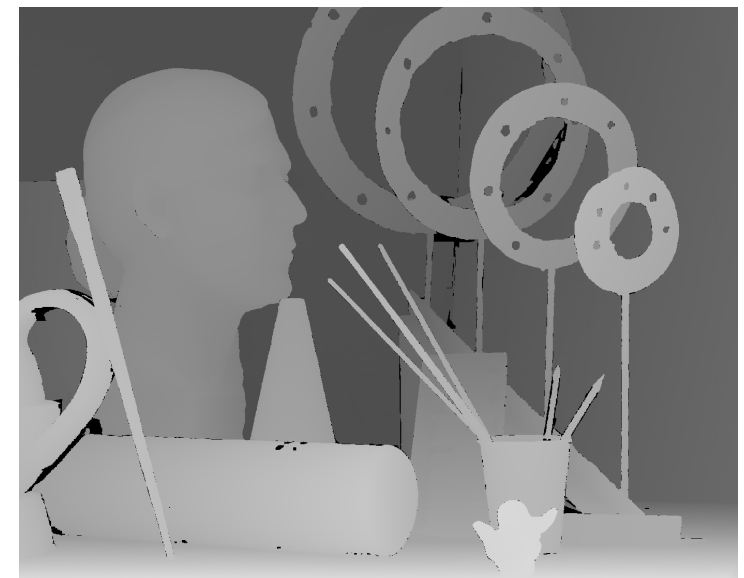
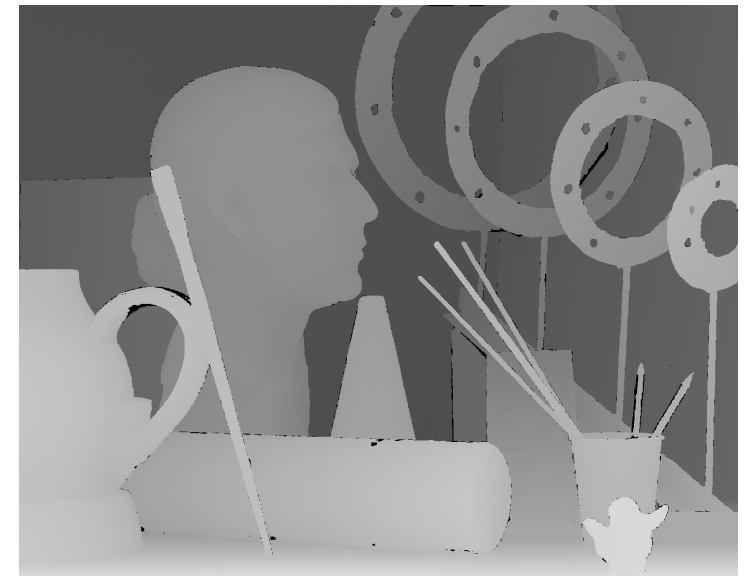
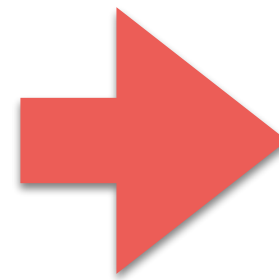


+  $P_1$



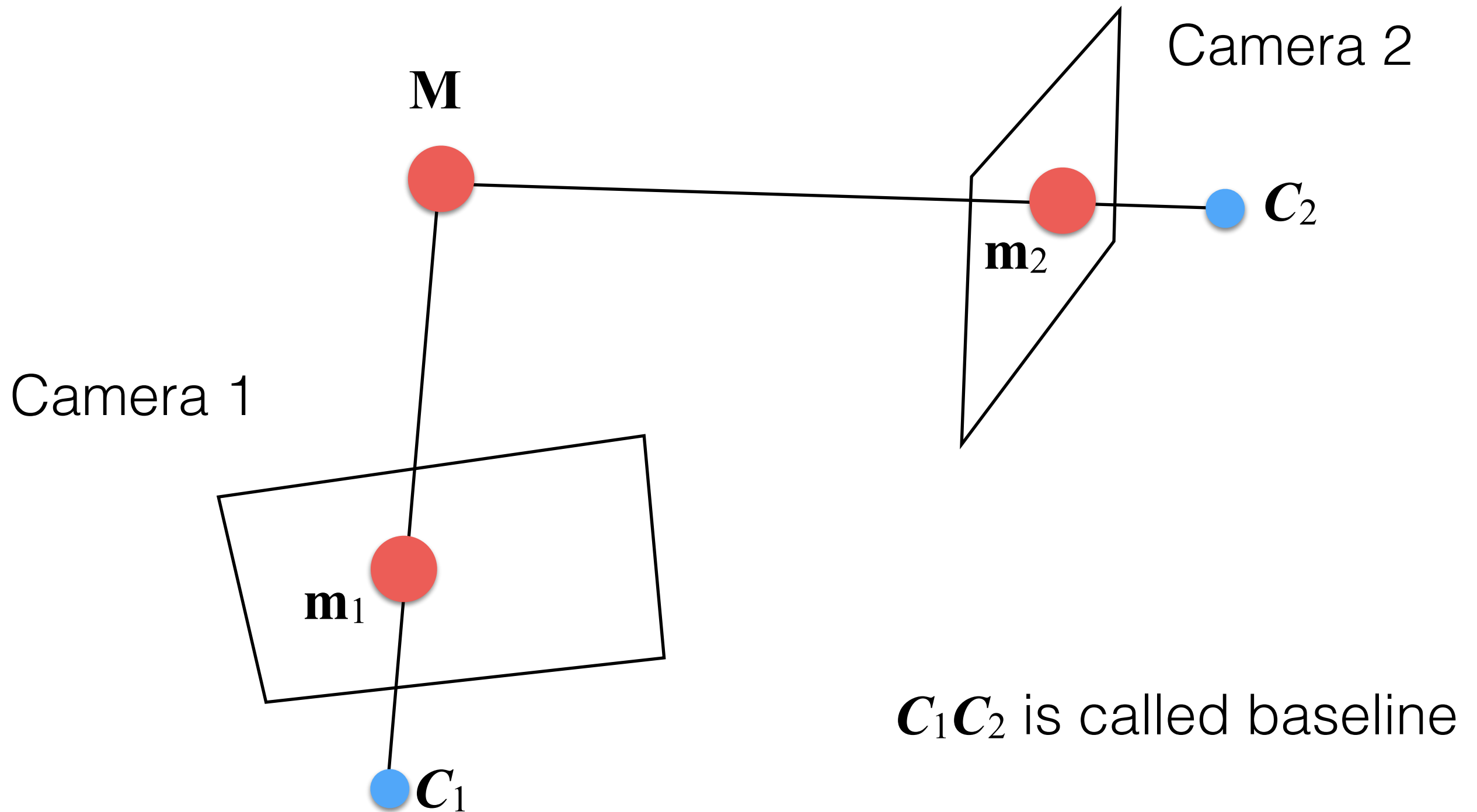
+  $P_2$

Input

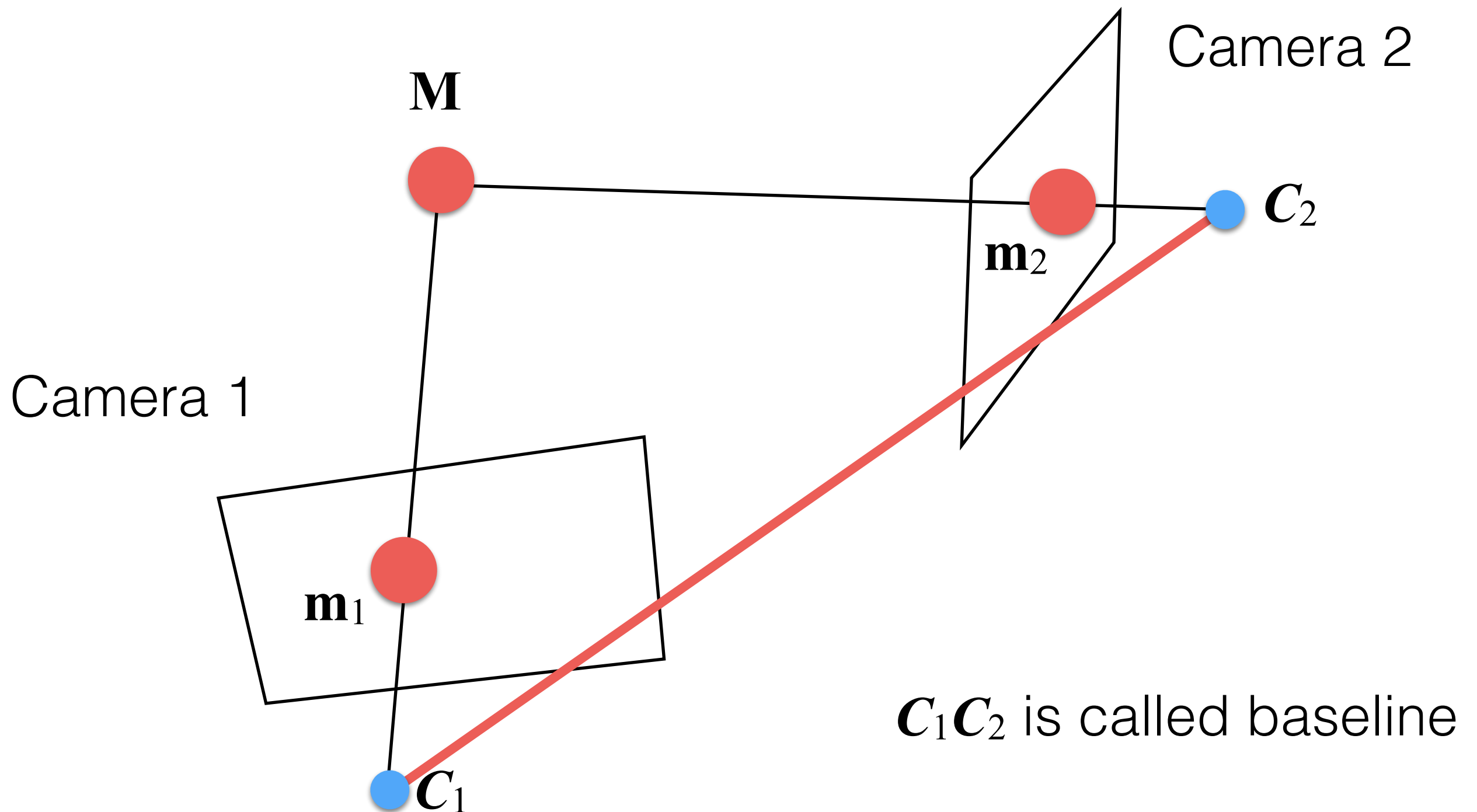


Output

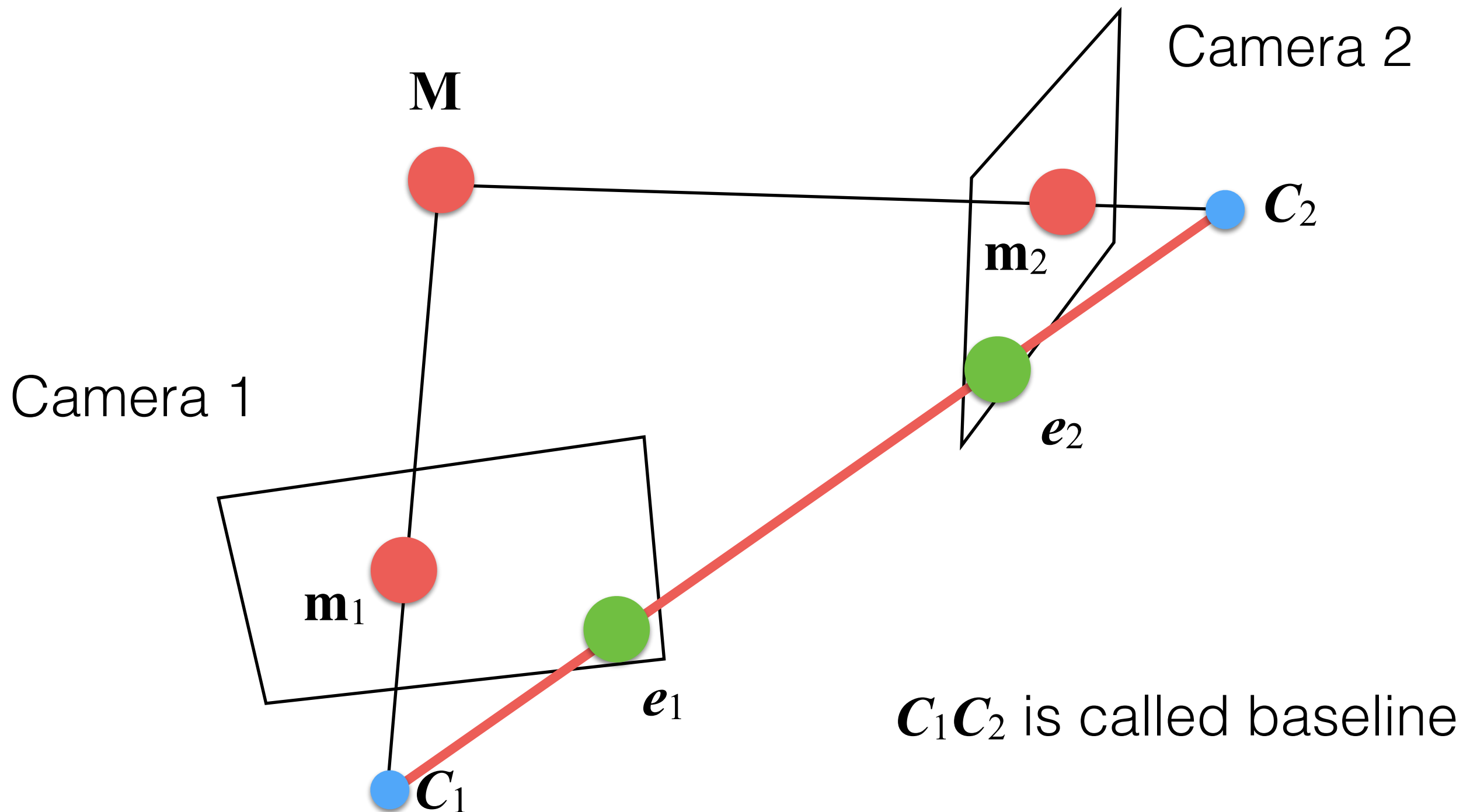
# Epipolar Geometry



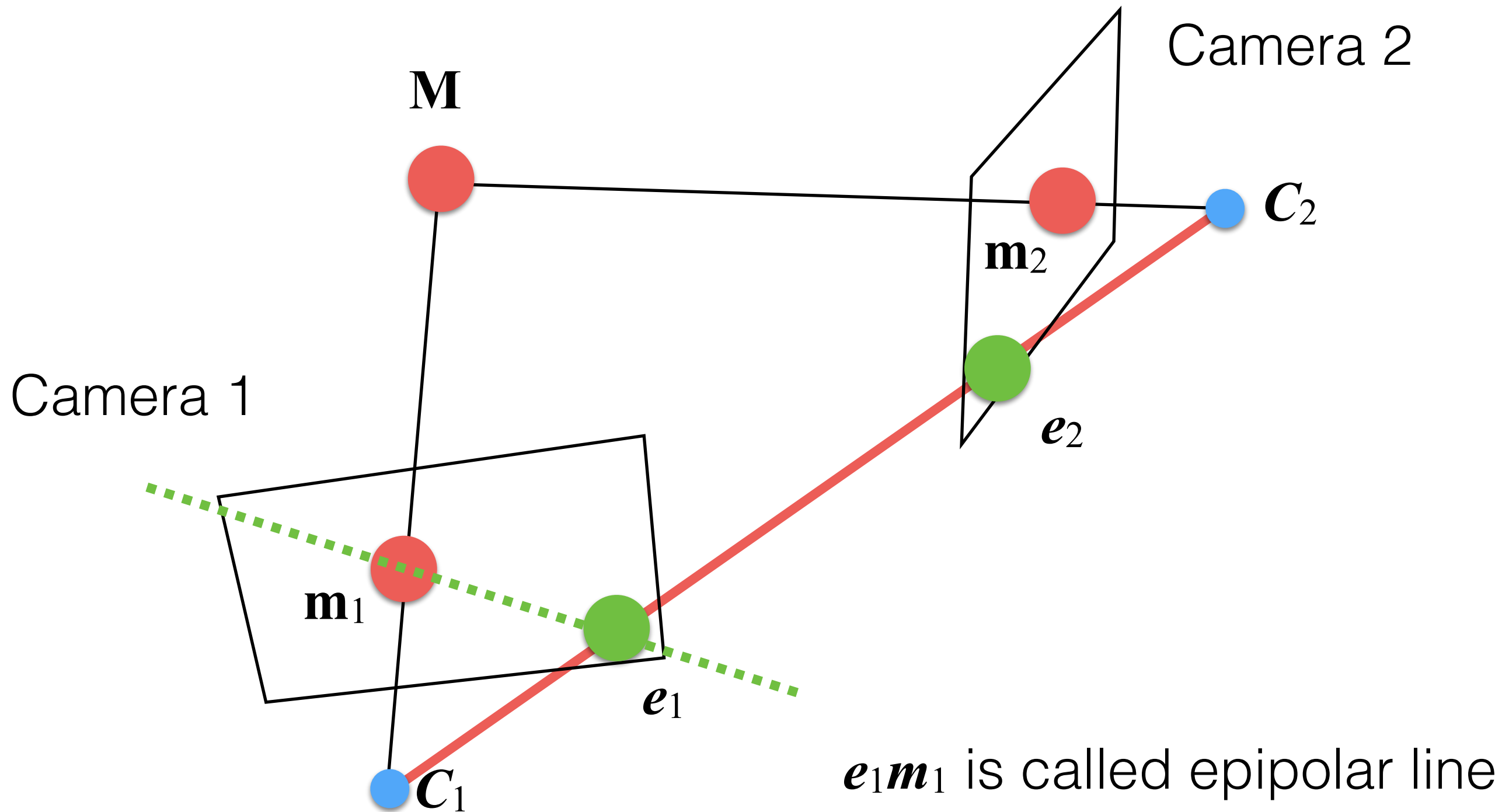
# Epipolar Geometry



# Epipolar Geometry



# Epipolar Geometry





# Epipolar Geometry: Epipolar Lines

- The epipolar line is defined as:

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

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

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

- where epipoles are defined as:

$$\mathbf{e}_1 \sim P_1 \cdot C_1$$

$$\mathbf{e}_2 \sim P_2 \cdot C_2$$

# Epipolar Geometry: Epipolar Lines

- The fundamental matrix is also defined as

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

- Knowing this, and that:

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

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

- This is an equation of a line:

$$\mathbf{l} = F \cdot \mathbf{m}_1 \leftrightarrow (l_1x + l_2y + l_3) = 0$$

$$\mathbf{l} = F^{\top} \cdot \mathbf{m}_2 \leftrightarrow (l_1x + l_2y + l_3) = 0$$

# 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



Left



Right



# Epipolar Geometry: Example 1





# Epipolar Geometry: Example 1





# Epipolar Geometry: Example 2





# Epipolar Geometry: Example 2





# Epipolar Geometry: Example 2





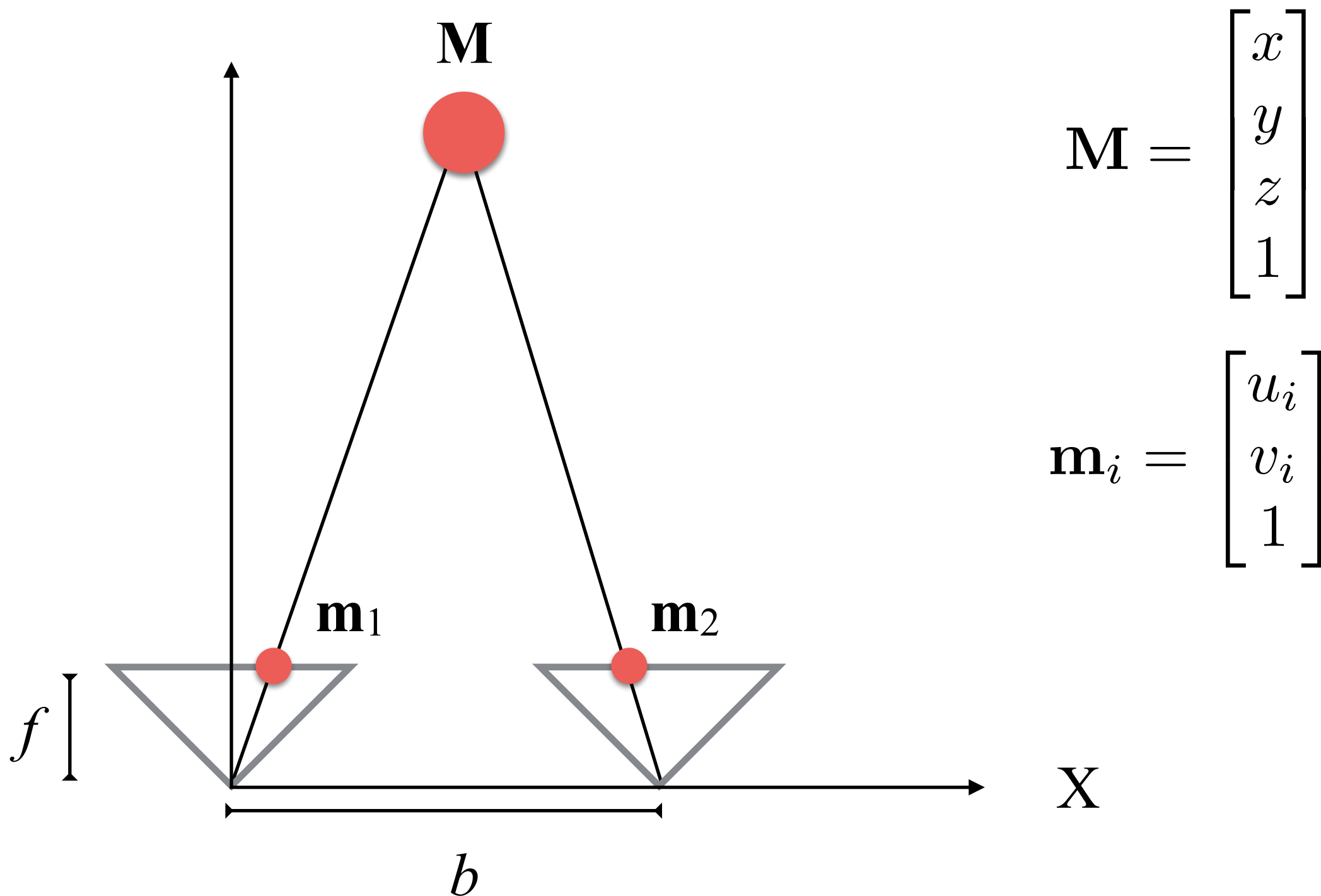
# Epipolar Geometry Example



# 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





# Epipolar Geometry: Ideal Case



Left



Right



# Epipolar Geometry: Ideal Case



Left



Right



# Epipolar Geometry: Ideal Case

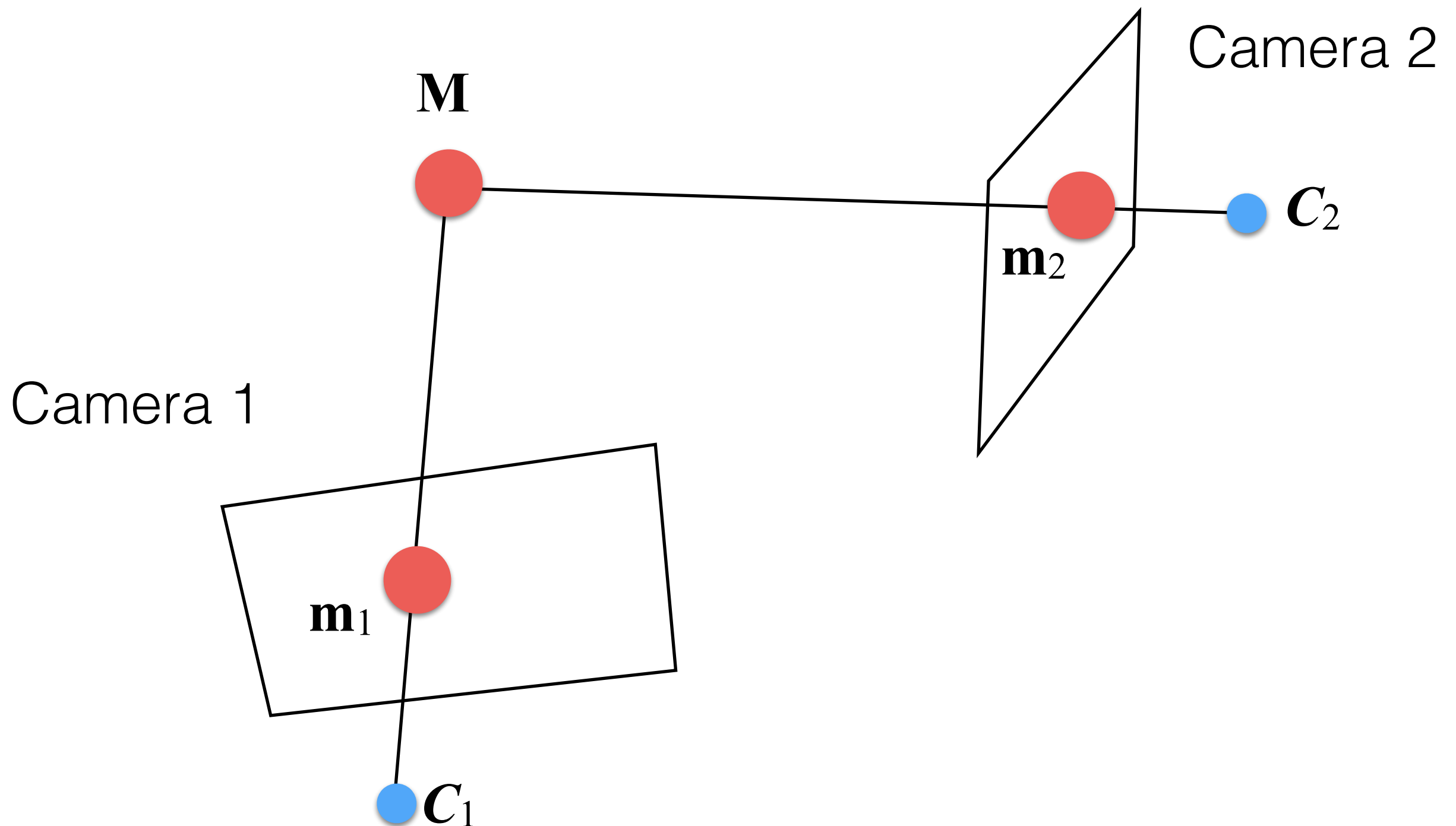


Left



Right

# Epipolar Geometry: The General Case

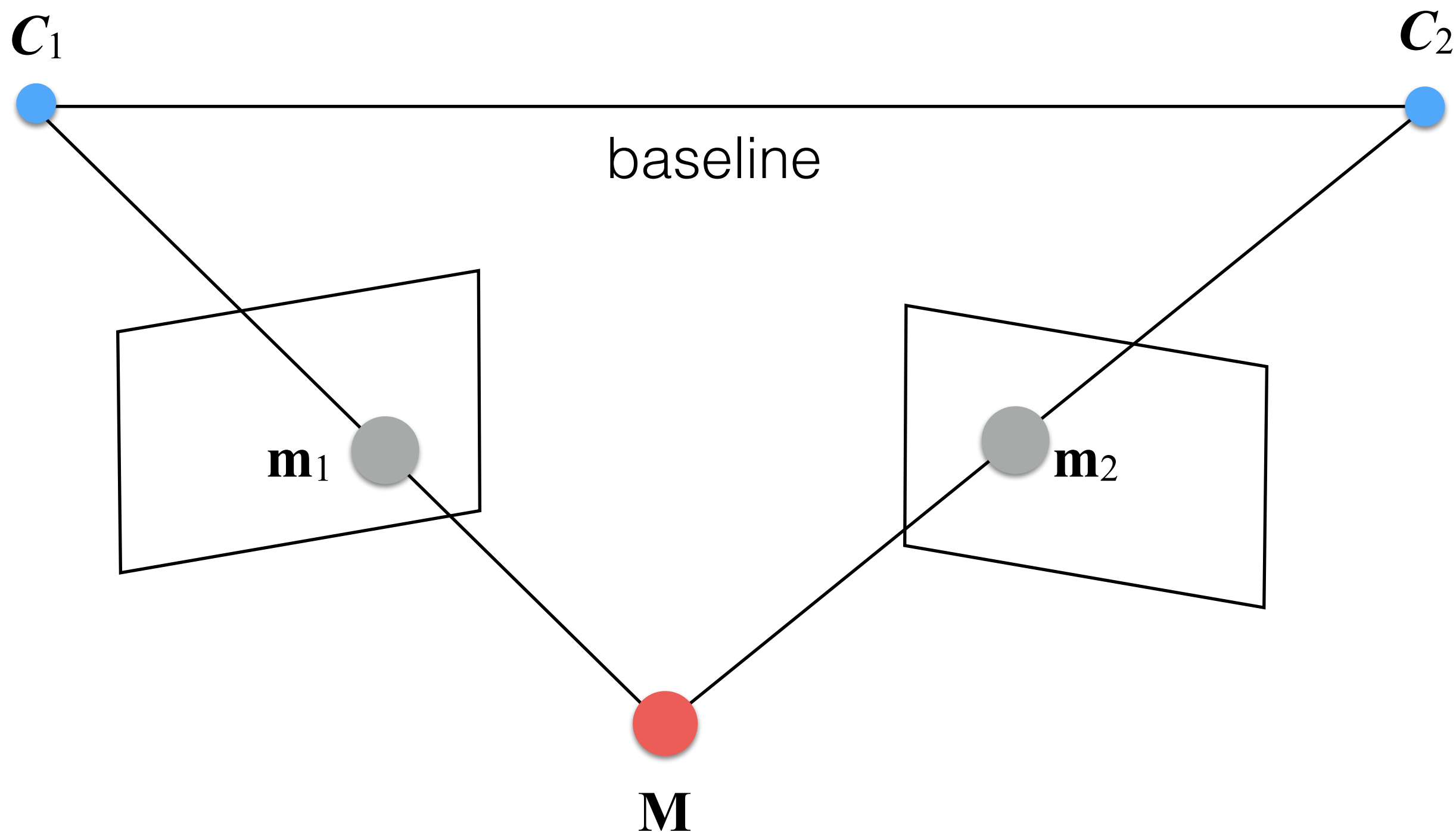




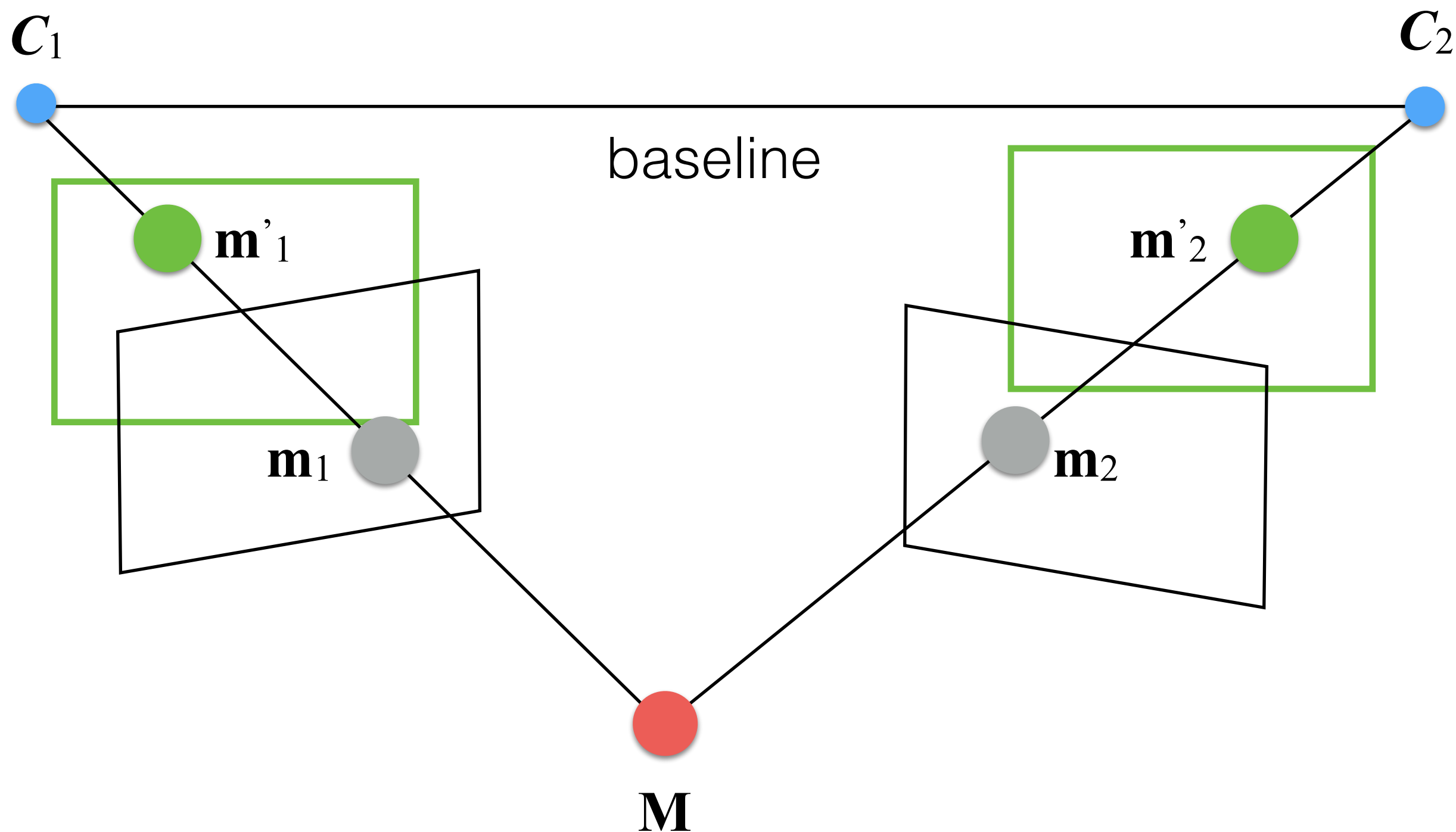
# Epipolar Geometry: Rectification

- 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:

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

# Epipolar Geometry: Rectification

- So we need to compute a new  $R$  matrix for both cameras.
- Let's see the process for the first camera:

$$R_1 = [\mathbf{r}_1 \quad \mathbf{r}_2 \quad \mathbf{r}_3]$$

- Given that the optical centers do not move, we have as  $X$  axis ( $\mathbf{r}_1$ ):

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

# Epipolar Geometry: Rectification

- The new  $Y$  axis ( $\mathbf{r}_2$ ) is defined as

$$\mathbf{r}_2 = \mathbf{r}_3^* \times \mathbf{r}_1$$

- where  $\mathbf{r}_3^*$  is the old  $\mathbf{r}_3$ .
- 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:

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

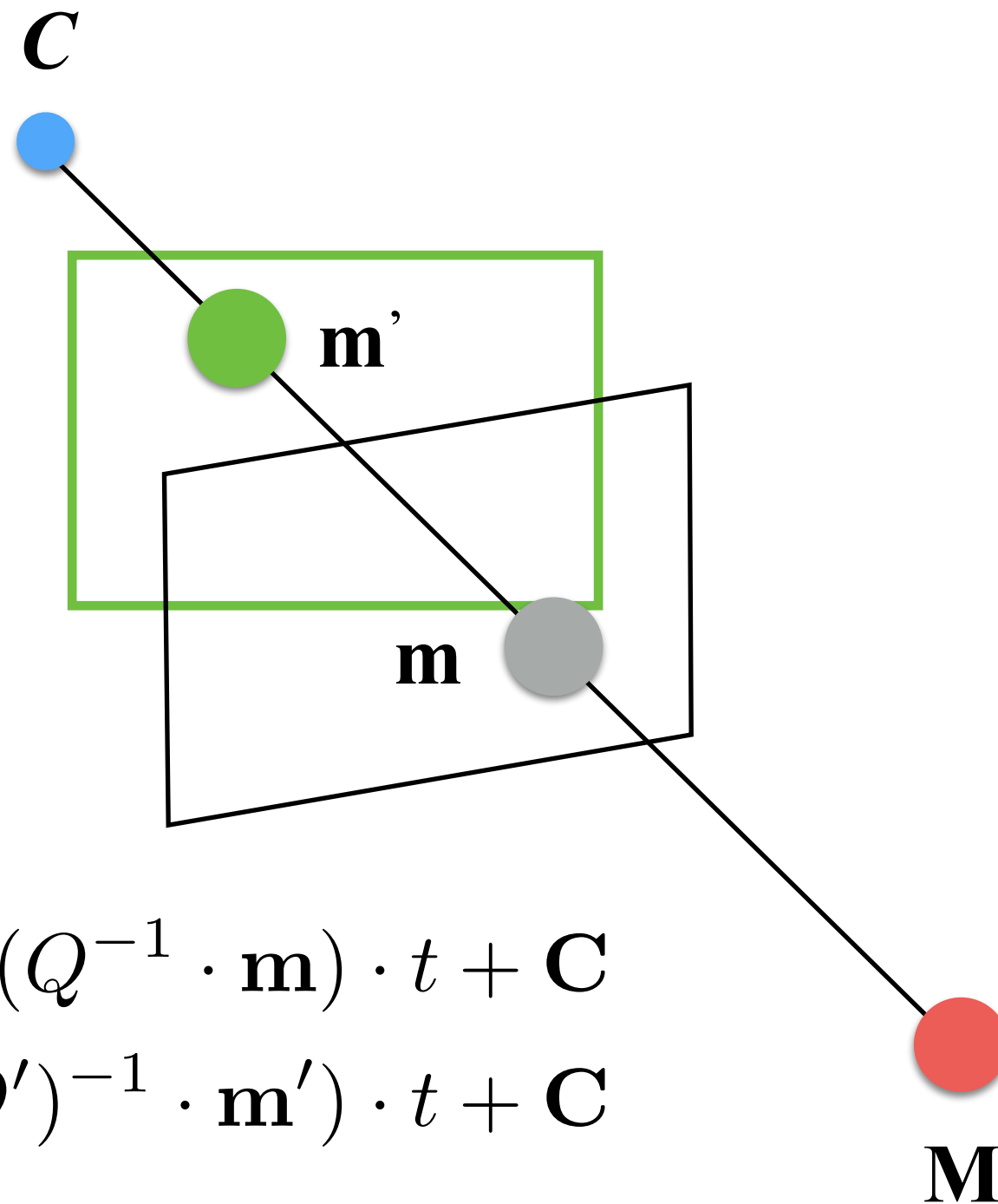
$$\mathbf{m}' \sim P' \cdot \mathbf{M}$$

- 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

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

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

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

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

# Epipolar Geometry: Rectification

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

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

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

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



# Epipolar Geometry: Rectification

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

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

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

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



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

# Epipolar Geometry: Rectification

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

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

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

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



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



# Epipolar Geometry: Rectification

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

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

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

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



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



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

# 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

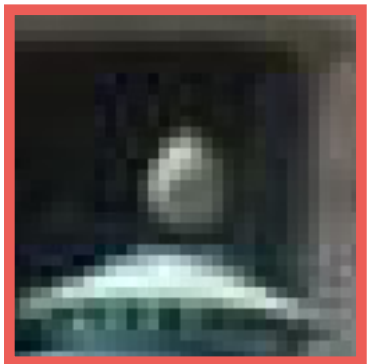


# Dense Matching



Left

# Dense Matching

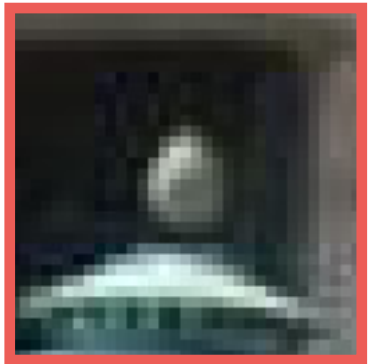


Extracted  
Patch  
(Left Image)

Left



# Dense Matching

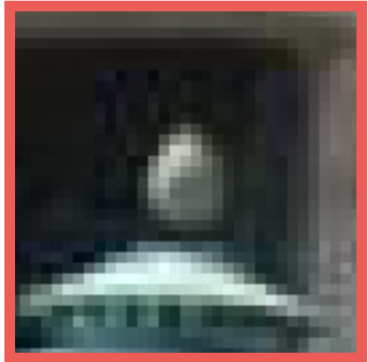


Extracted  
Patch  
(Left Image)



Right

# Dense Matching



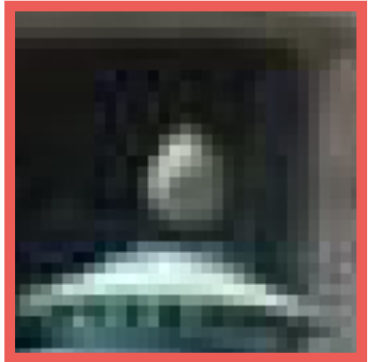
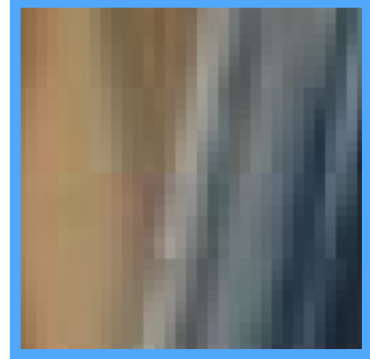
Extracted  
Patch  
(Left Image)



Right



# Dense Matching

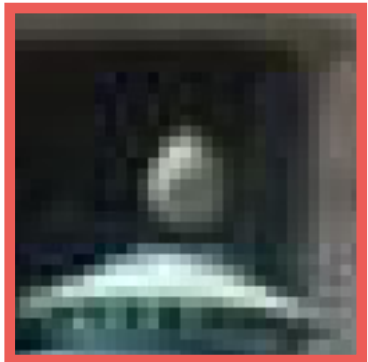


Extracted  
Patch  
(Left Image)

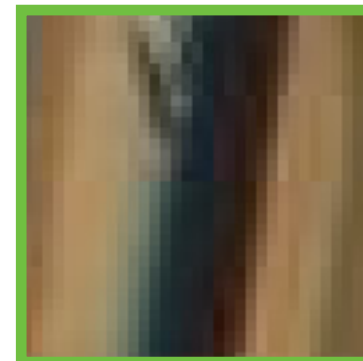
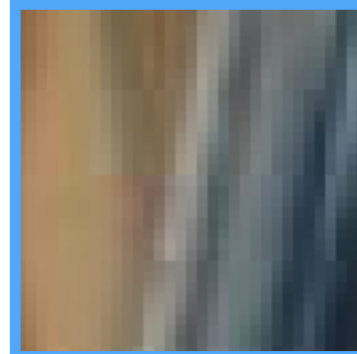


Right

# Dense Matching



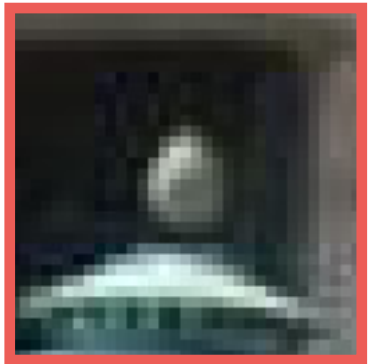
Extracted  
Patch  
(Left Image)



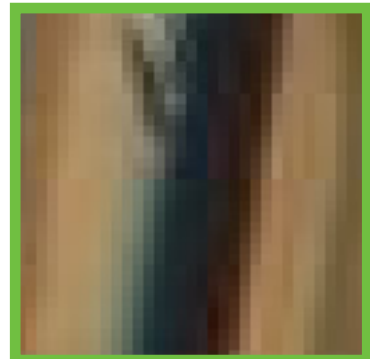
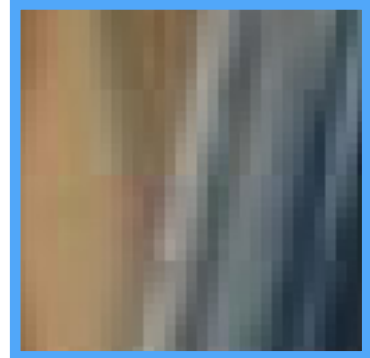
Right



# Dense Matching

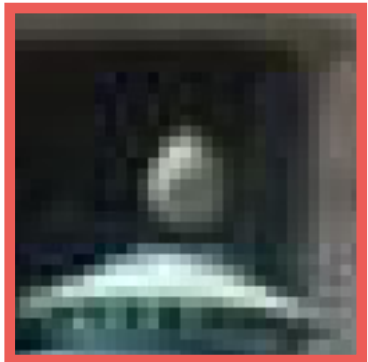


Extracted  
Patch  
(Left Image)



Right

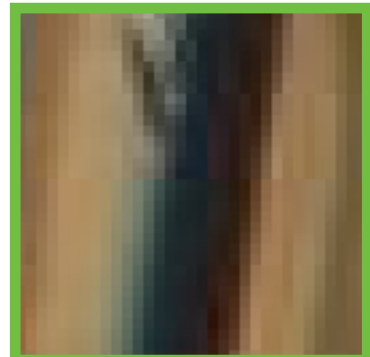
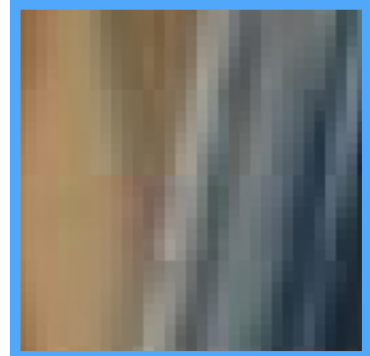
# Dense Matching



Extracted  
Patch  
(Left Image)

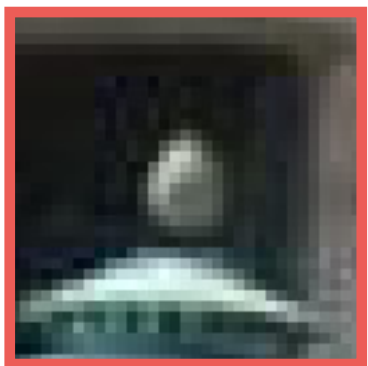


Right





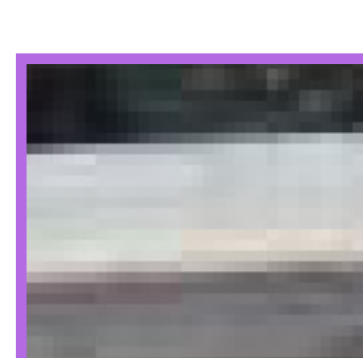
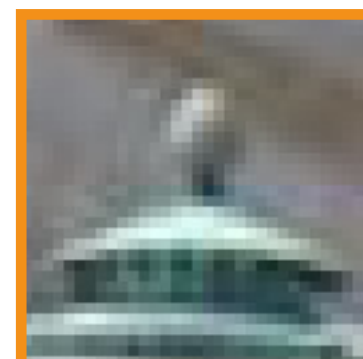
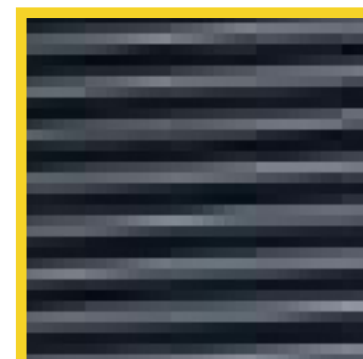
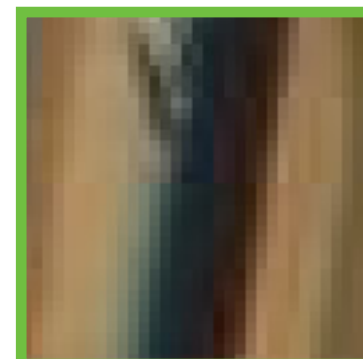
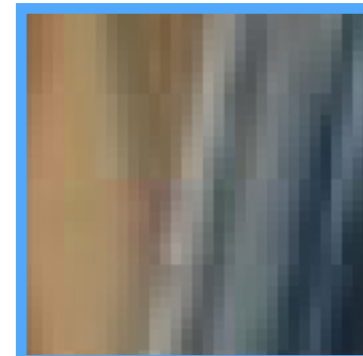
# Dense Matching



Extracted  
Patch  
(Left Image)



Right



# Dense Matching



Right

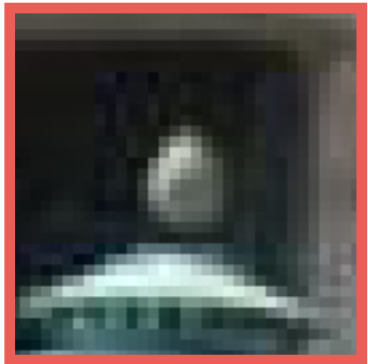


# Dense Matching



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:

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

# 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

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

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**

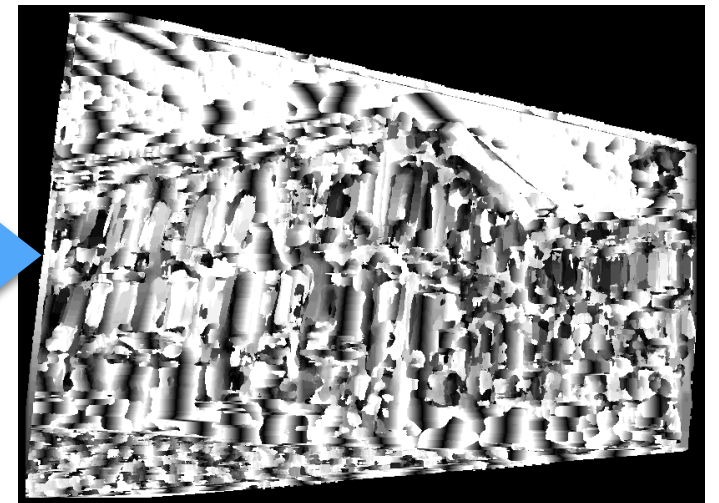


Left



Right

**Output**



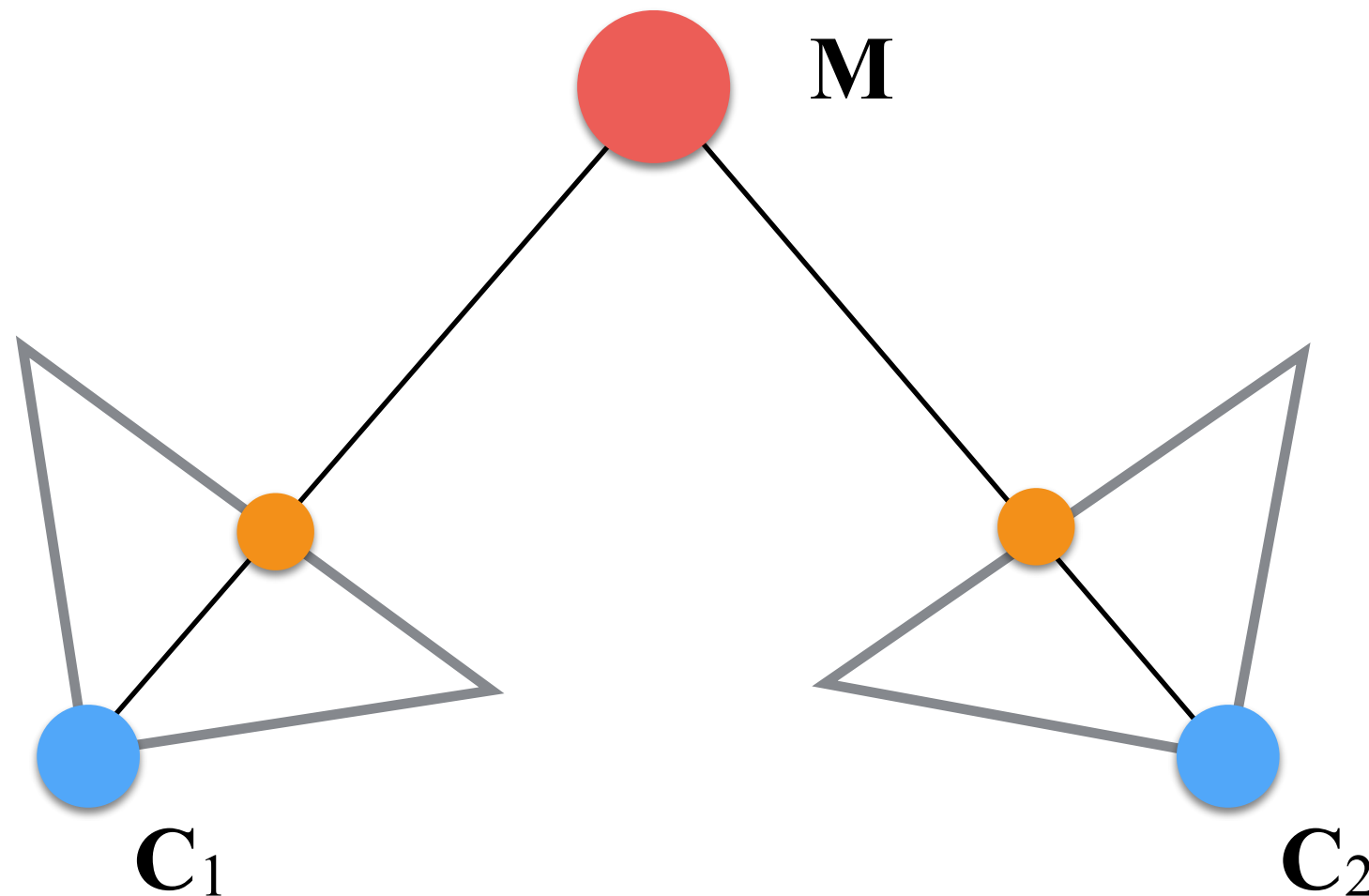
Disparity Map  
for the Left Image

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



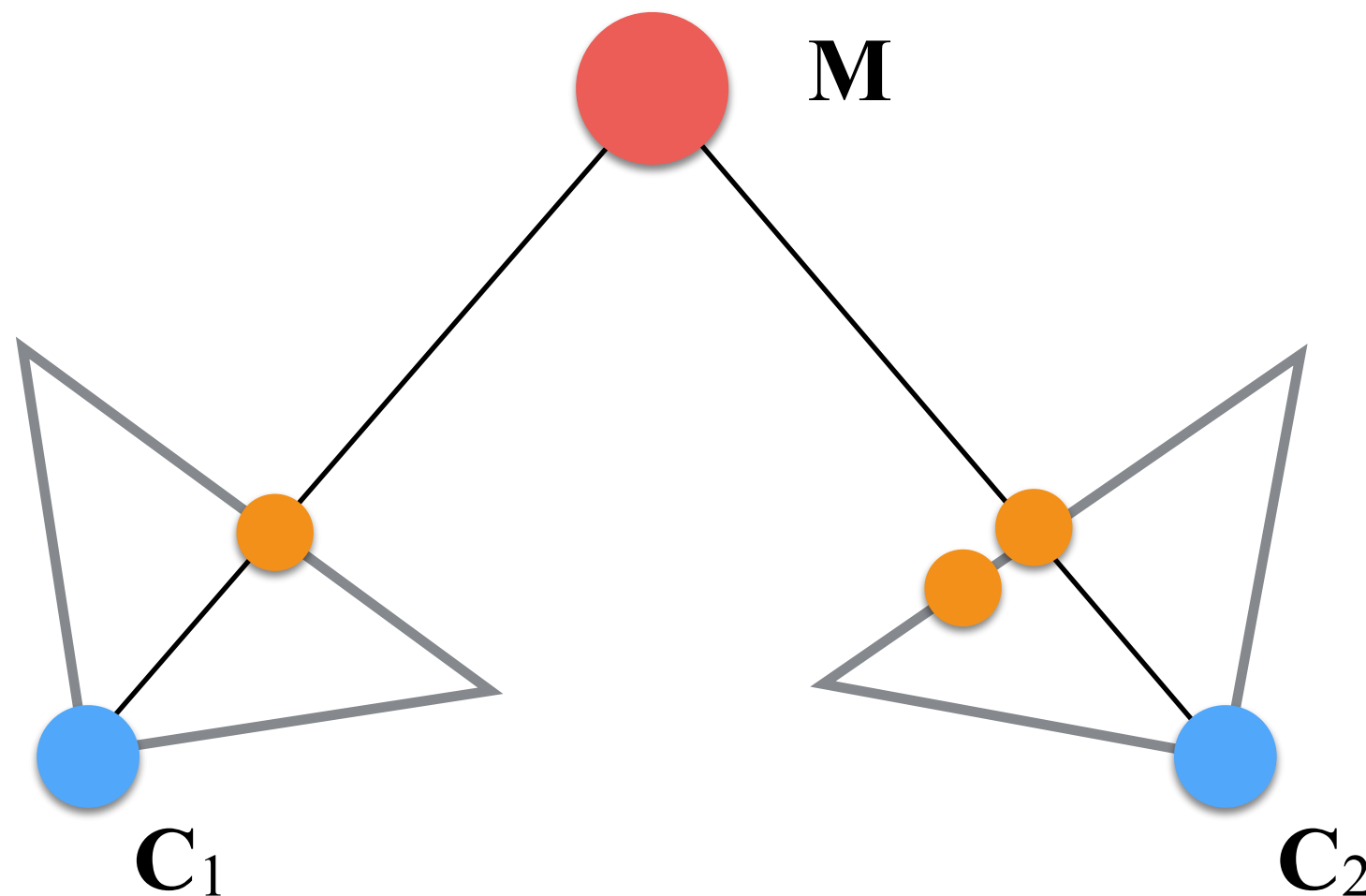
# Non-Local Constraints: Uniqueness

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



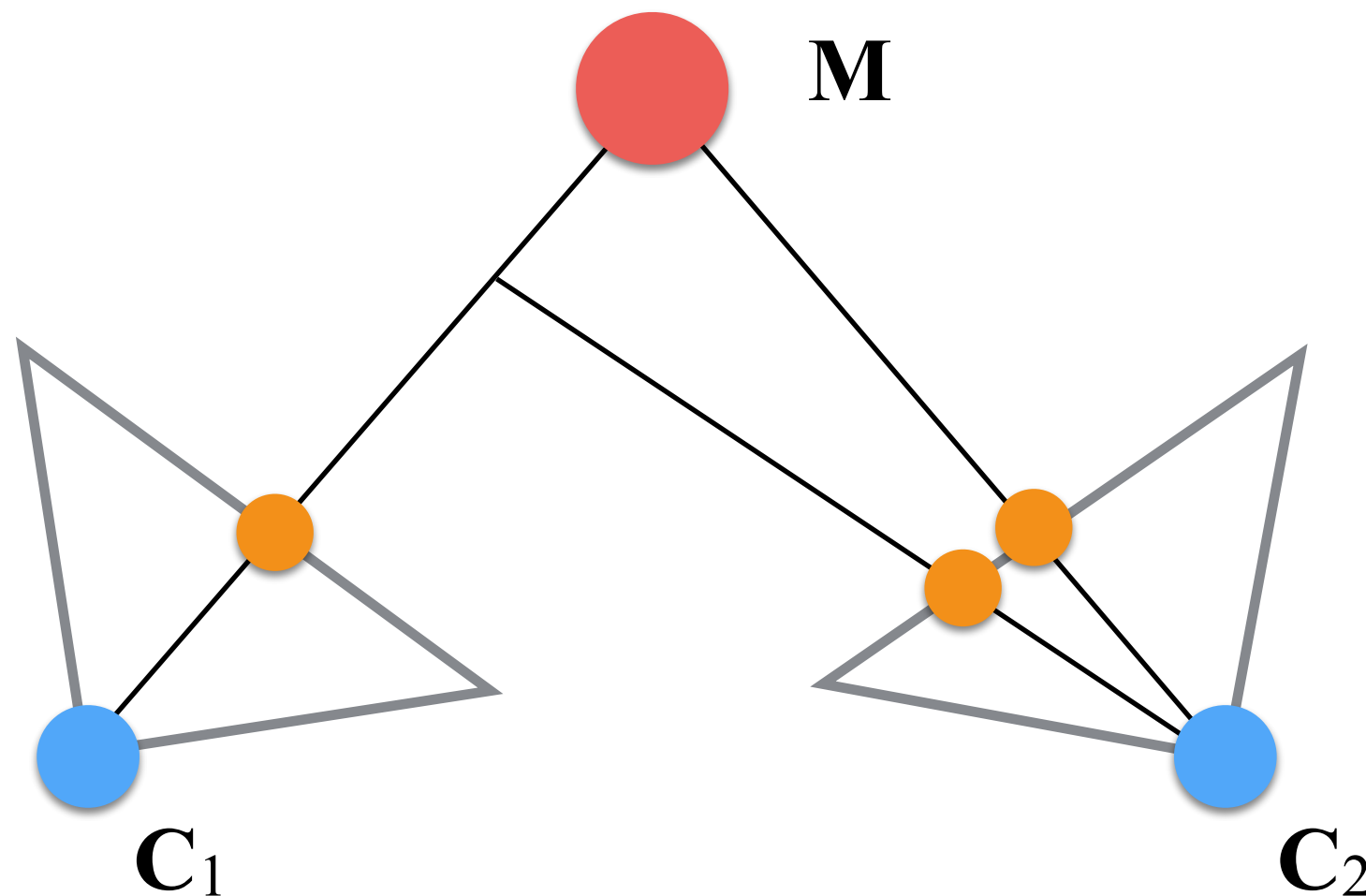
# Non-Local Constraints: Uniqueness

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



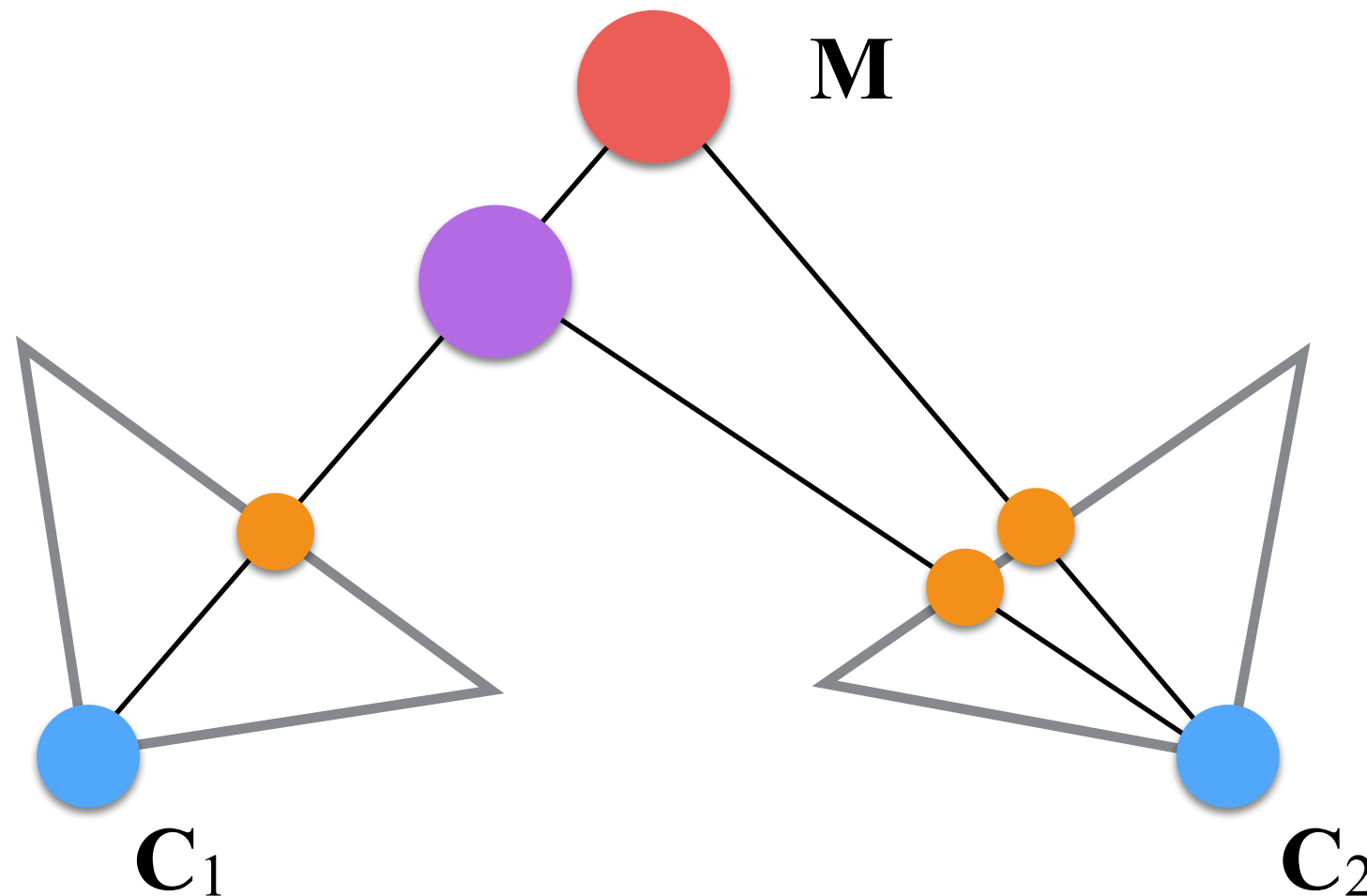
# Non-Local Constraints: Uniqueness

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



# Non-Local Constraints: Uniqueness

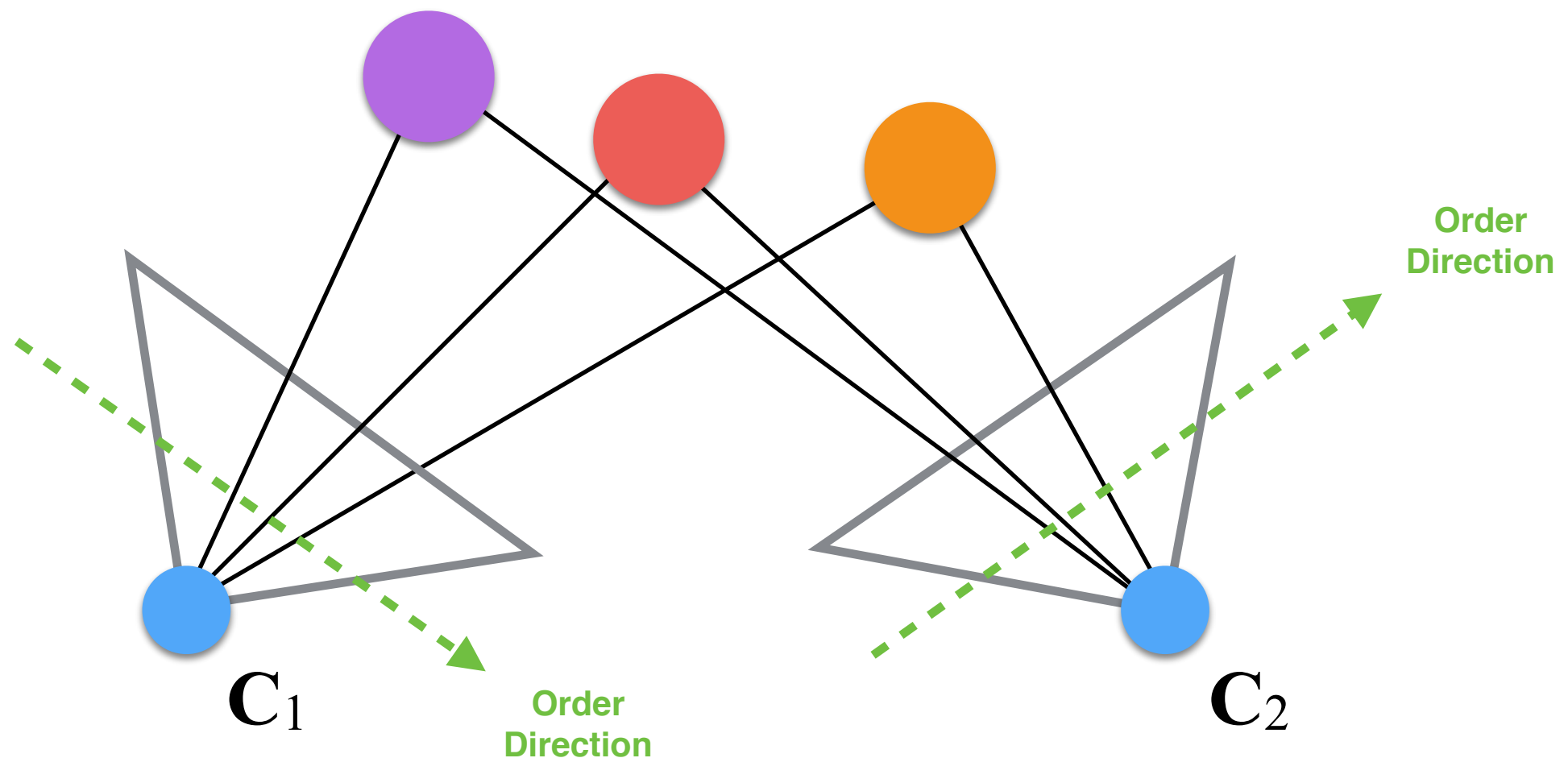
- For each point in one image, there should be at maximum one matching point in the other:





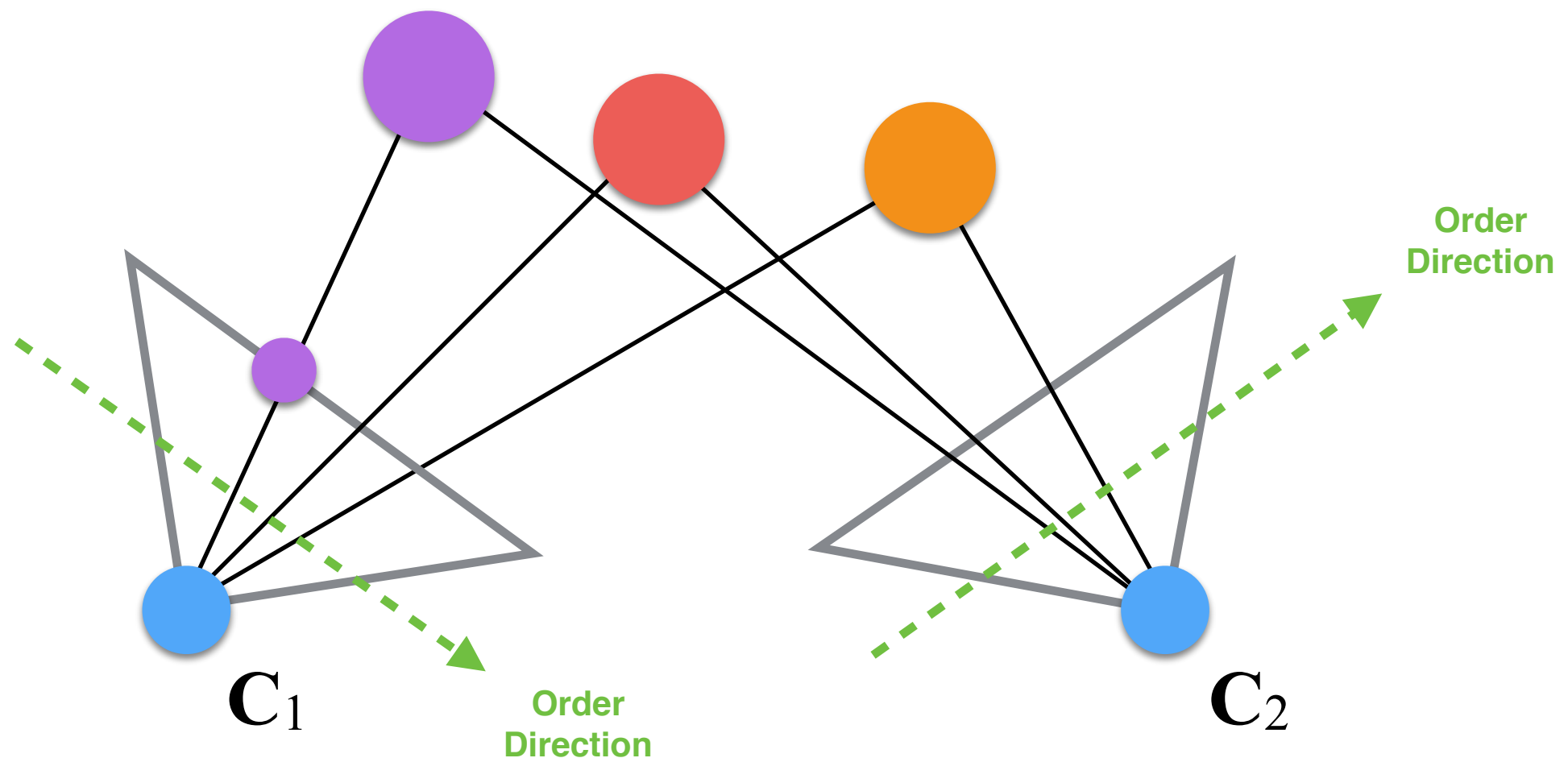
# Non-Local Constraints: Correct Ordering

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



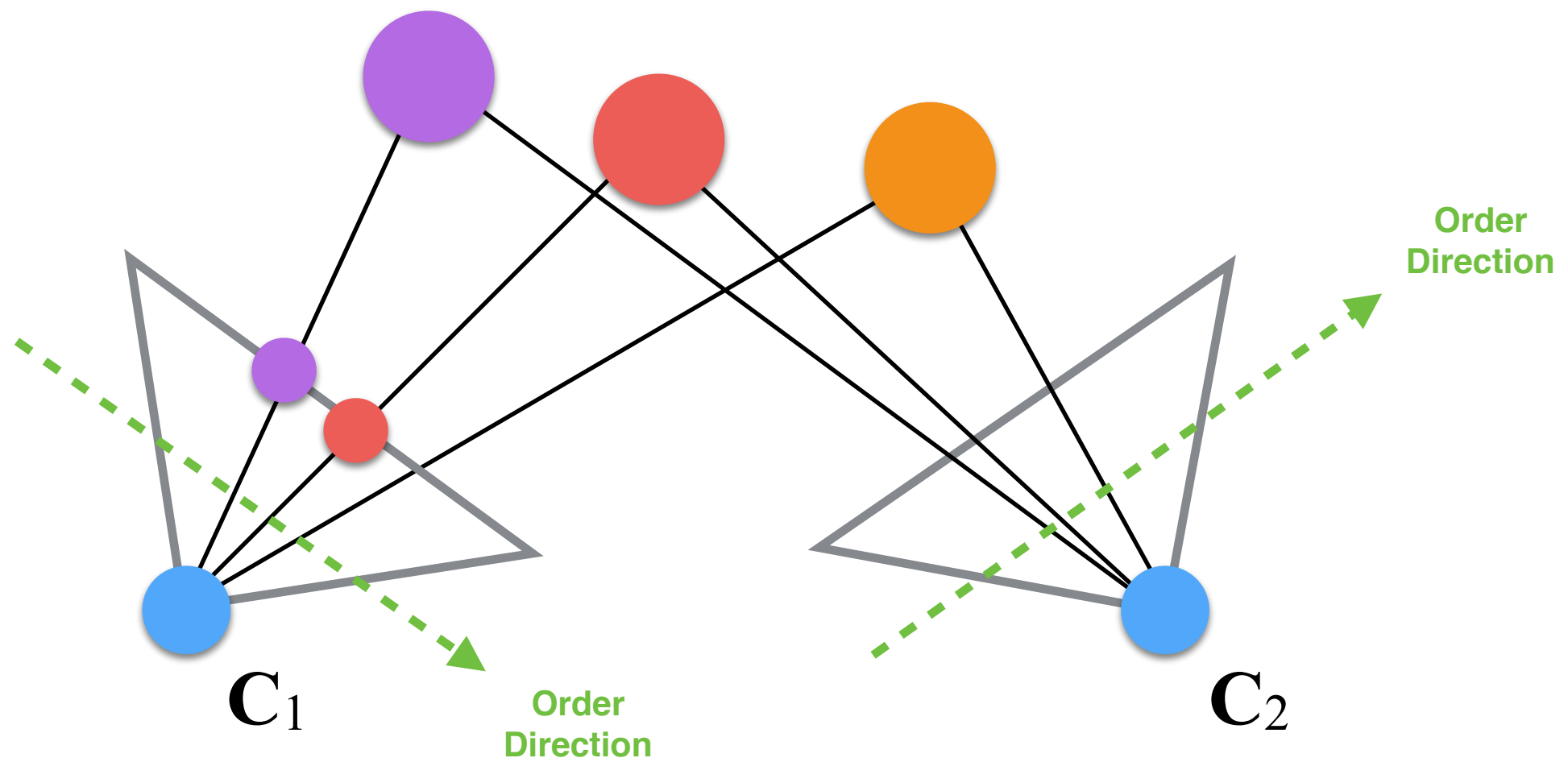
# Non-Local Constraints: Correct Ordering

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



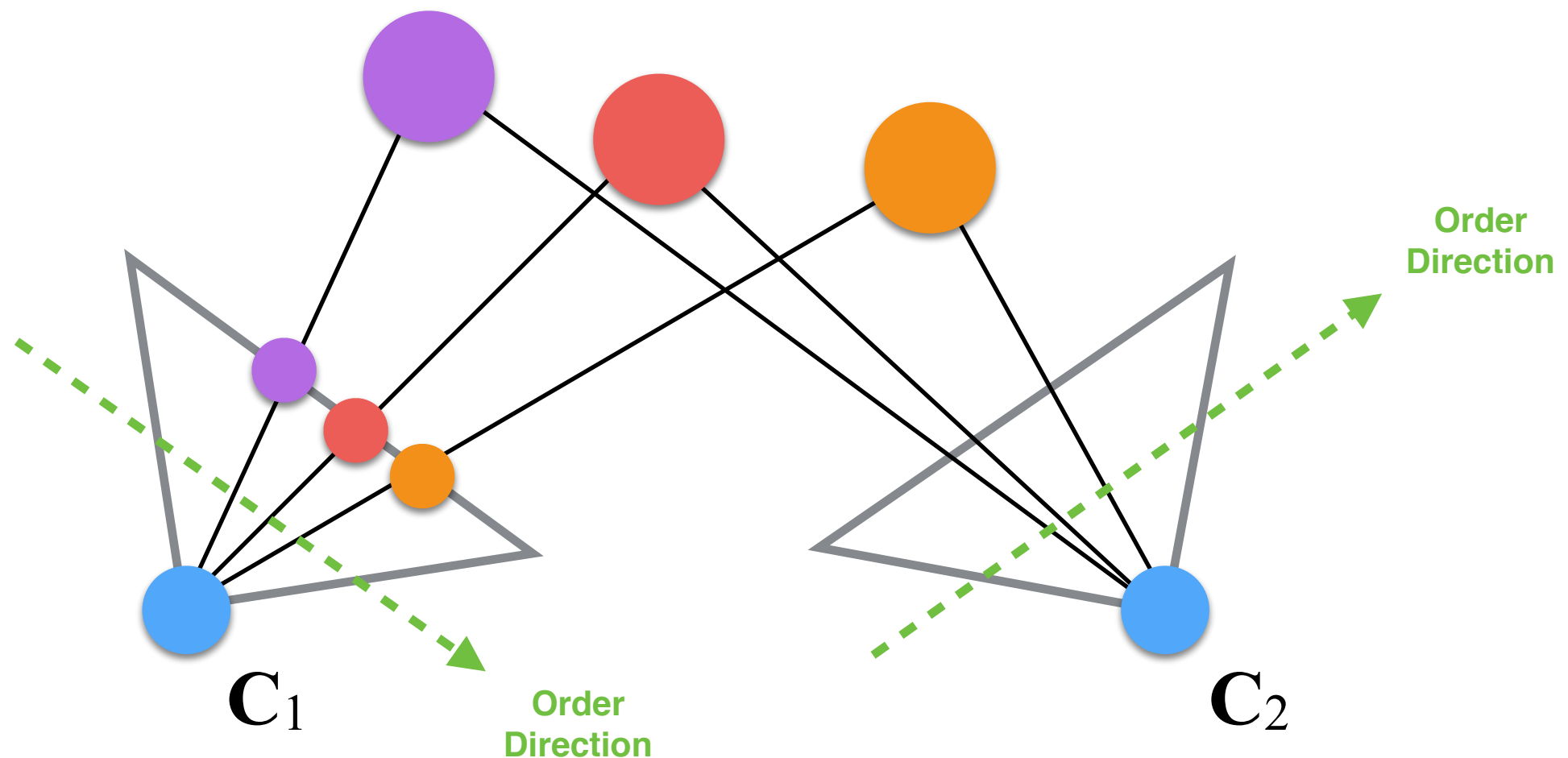
# Non-Local Constraints: Correct Ordering

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



# Non-Local Constraints: Correct Ordering

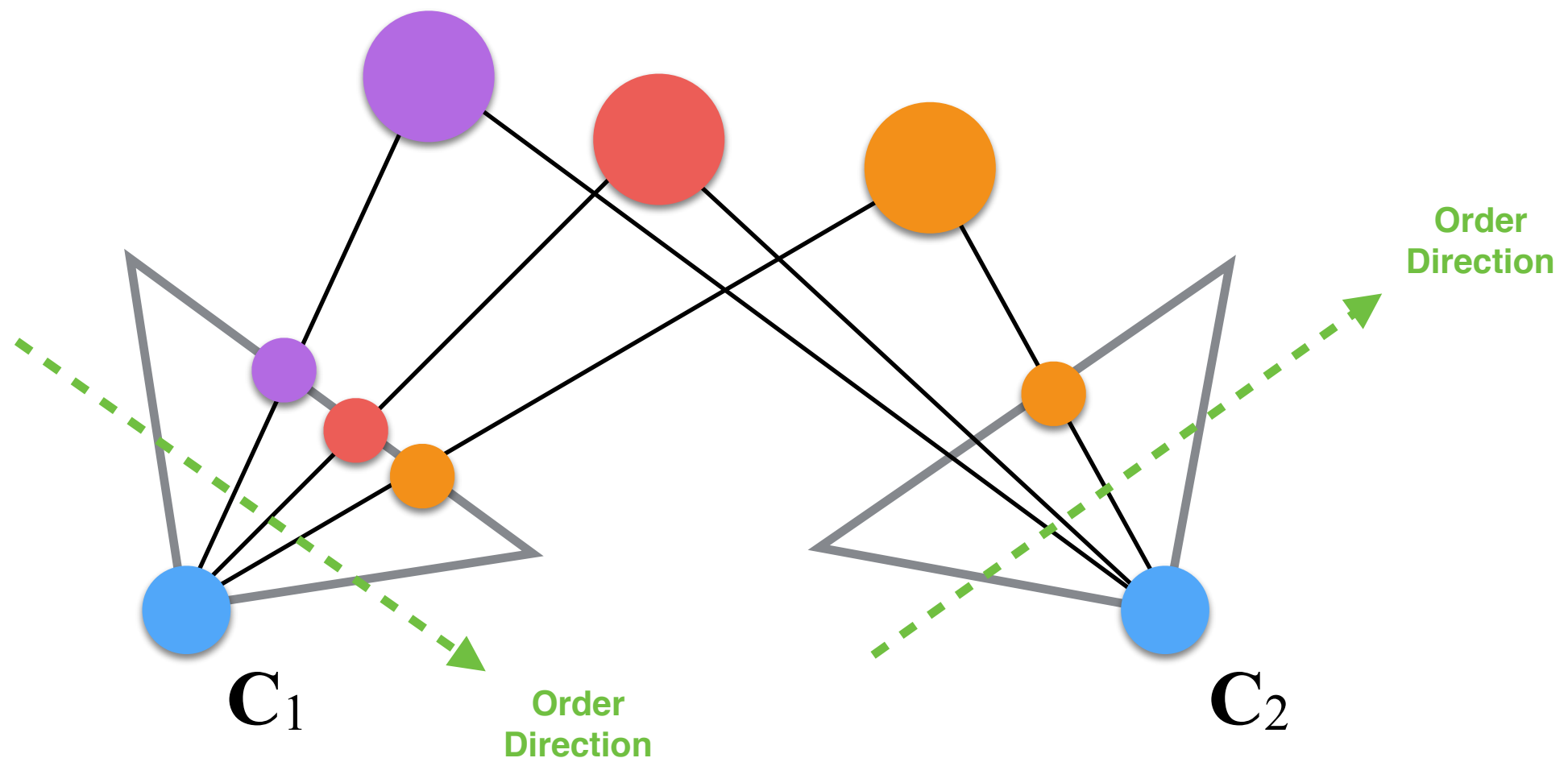
- Corresponding points should be in the same order in both views.





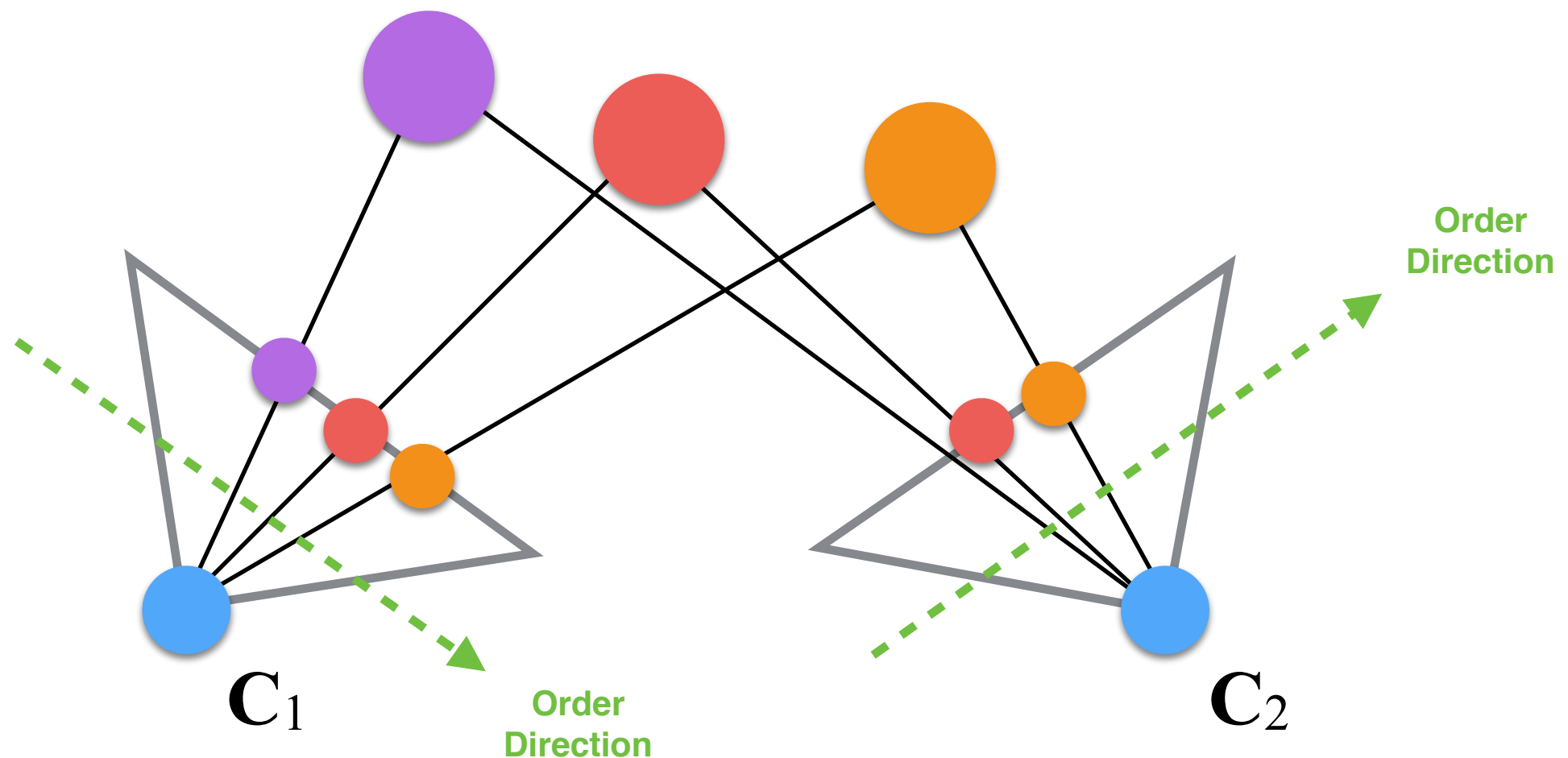
# Non-Local Constraints: Correct Ordering

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



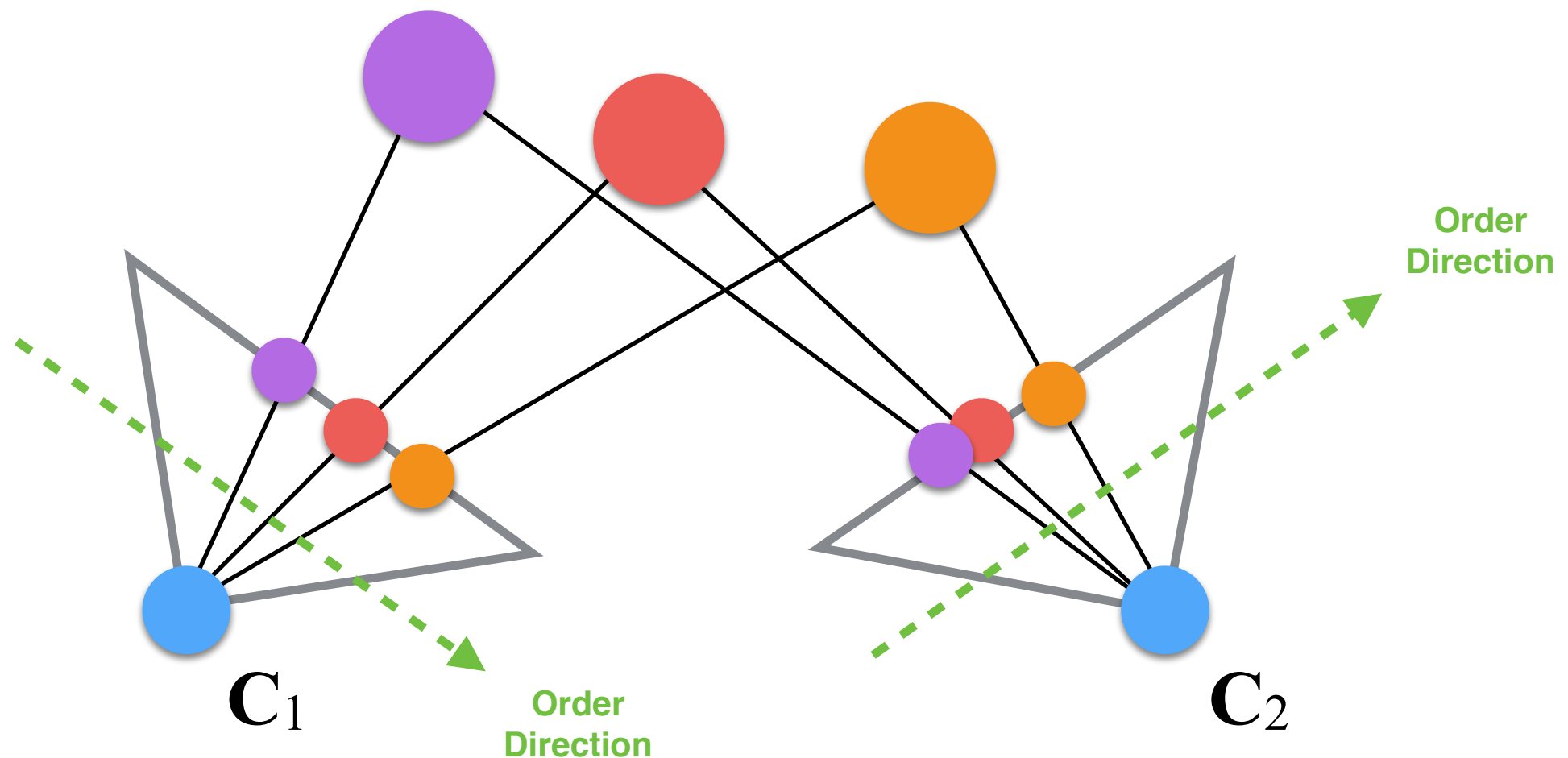
# Non-Local Constraints: Correct Ordering

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

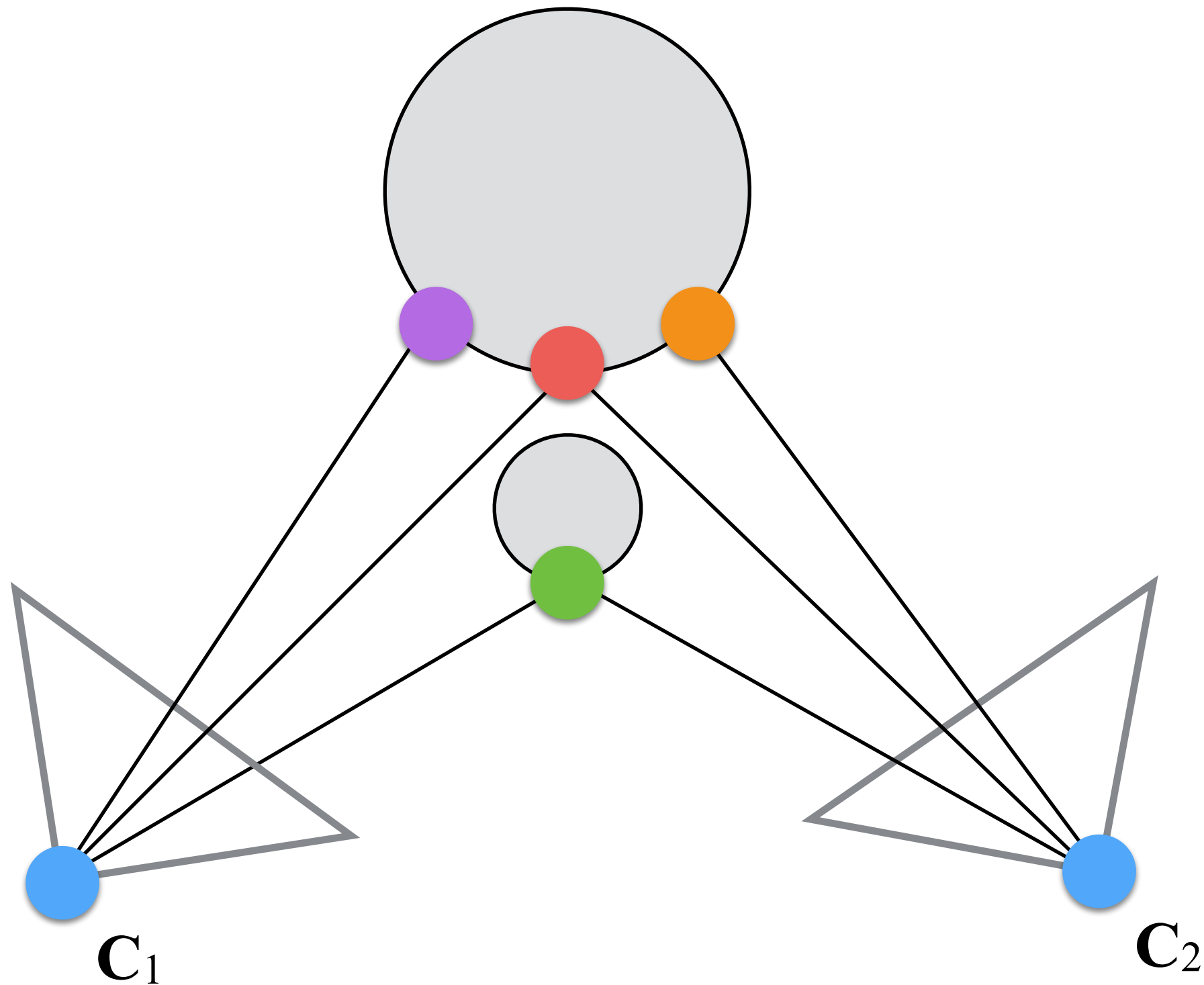


# Non-Local Constraints: Correct Ordering

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

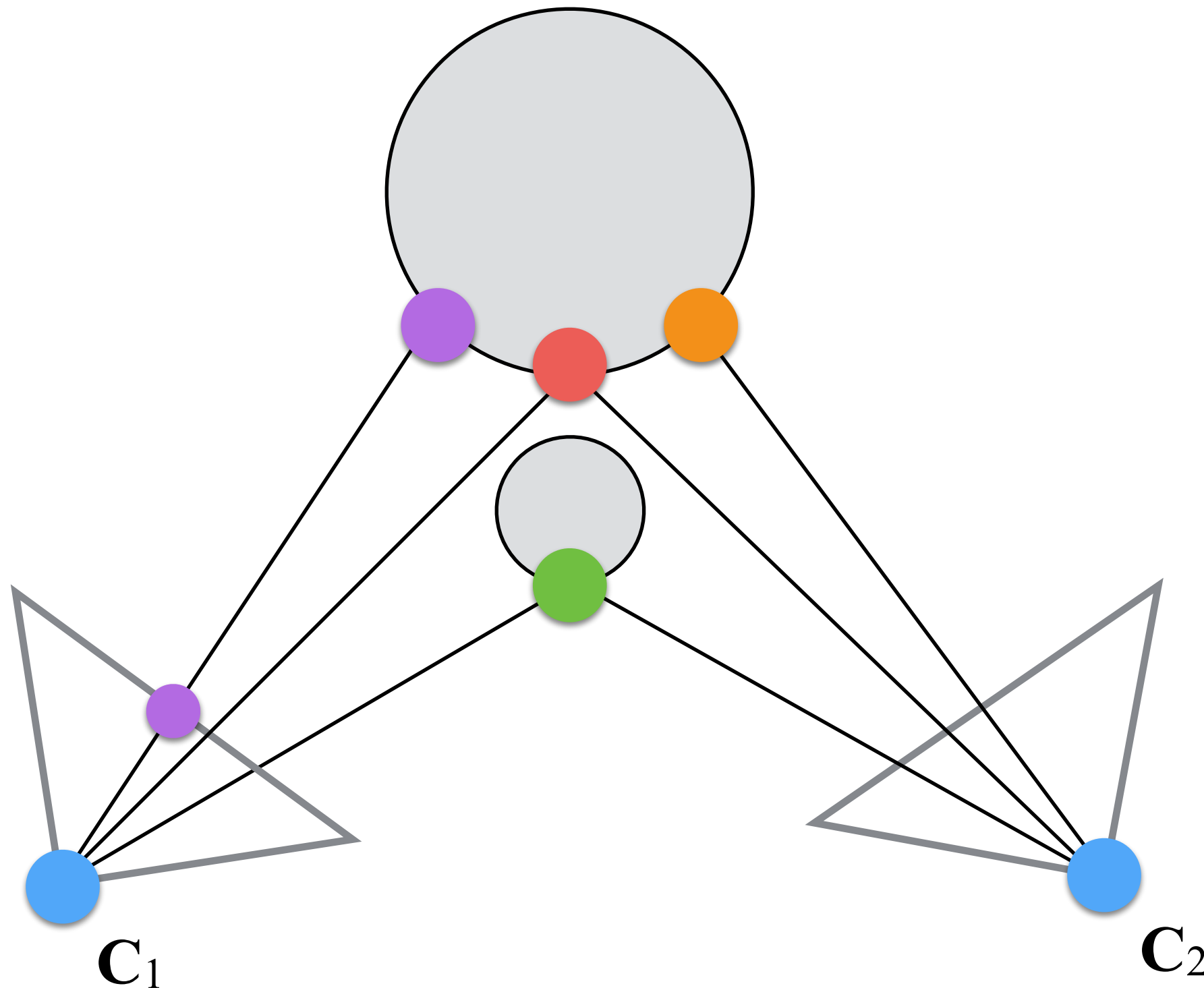


# Non-Local Constraints: Ordering Fail

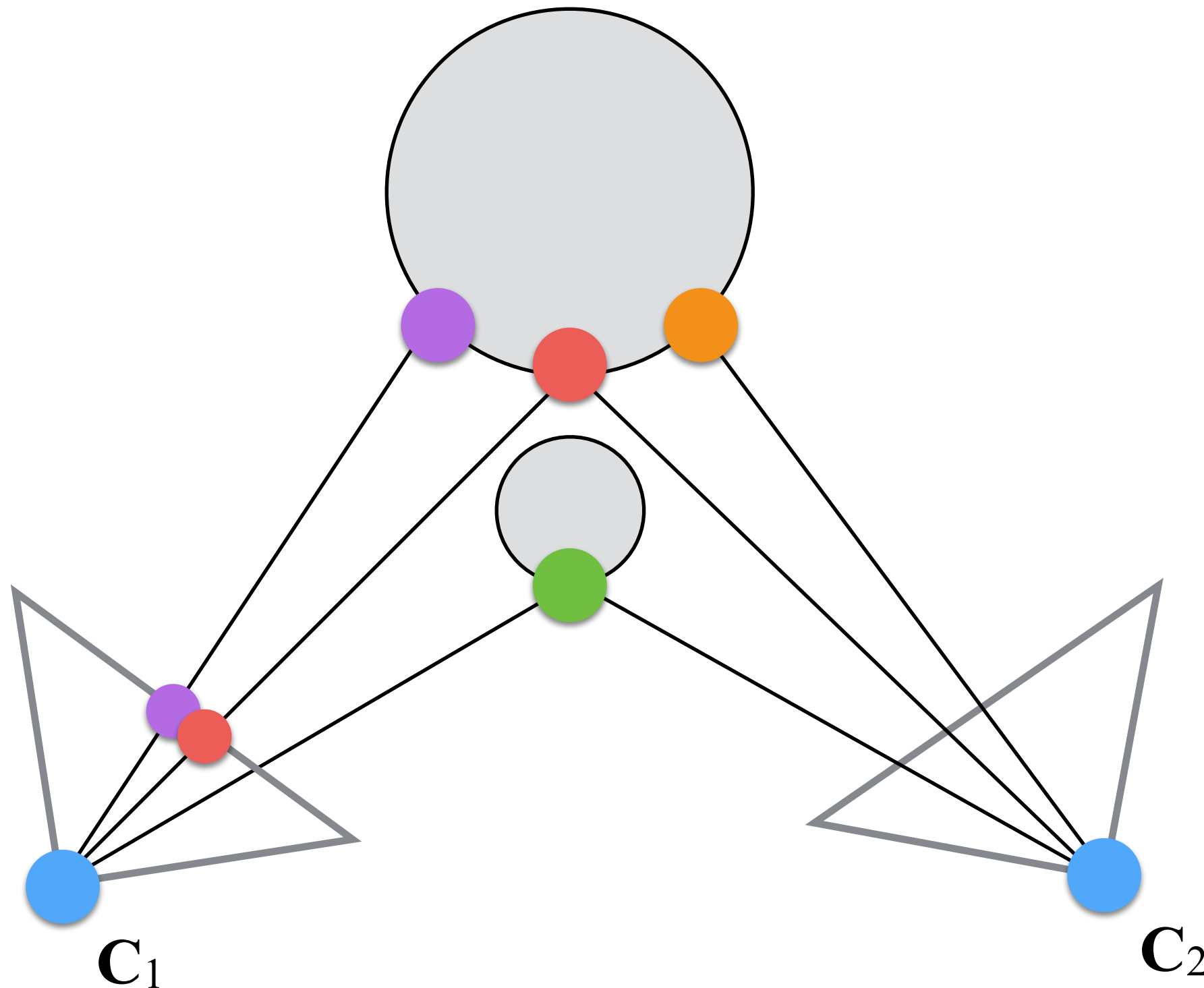


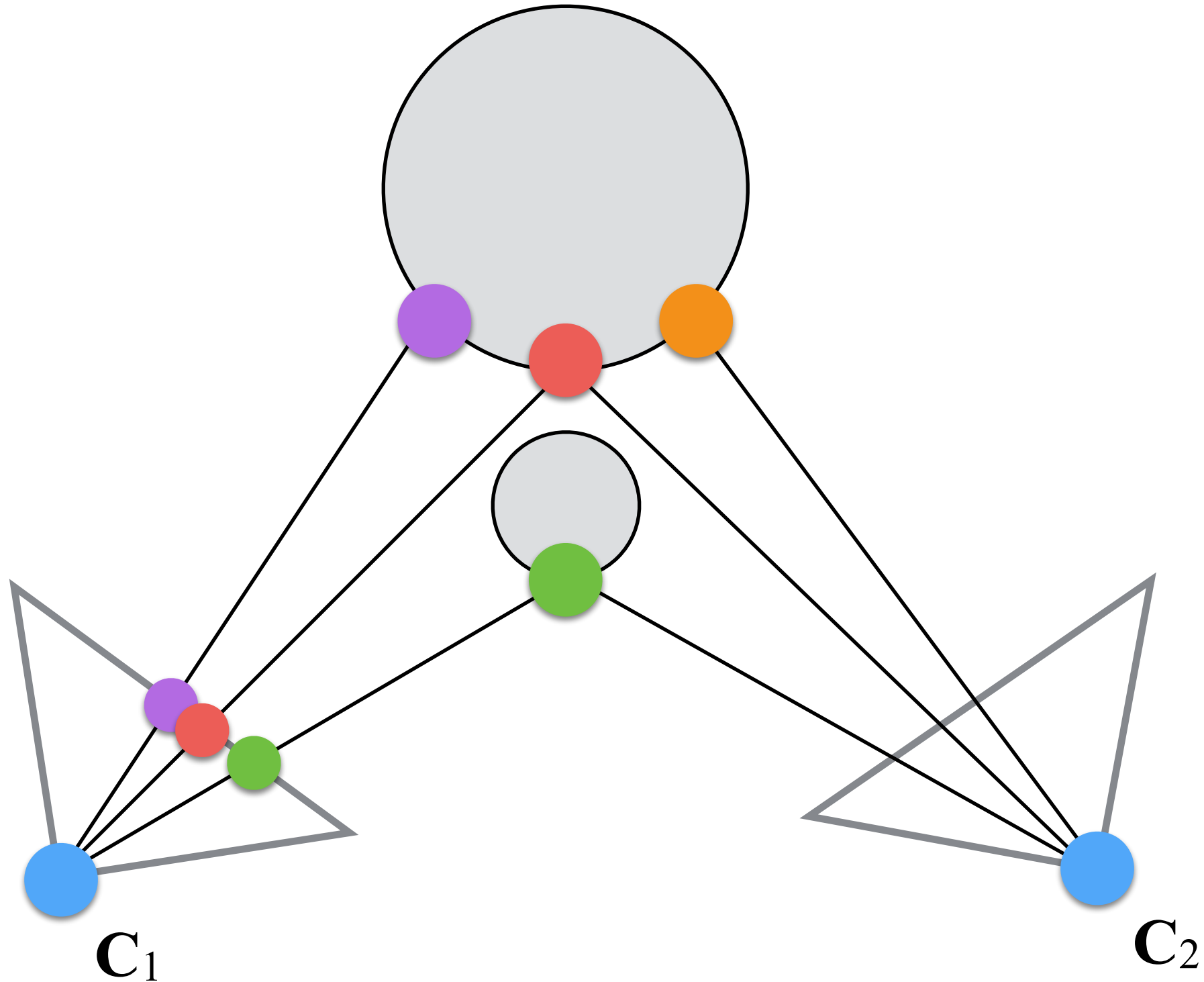


# Non-Local Constraints: Ordering Fail

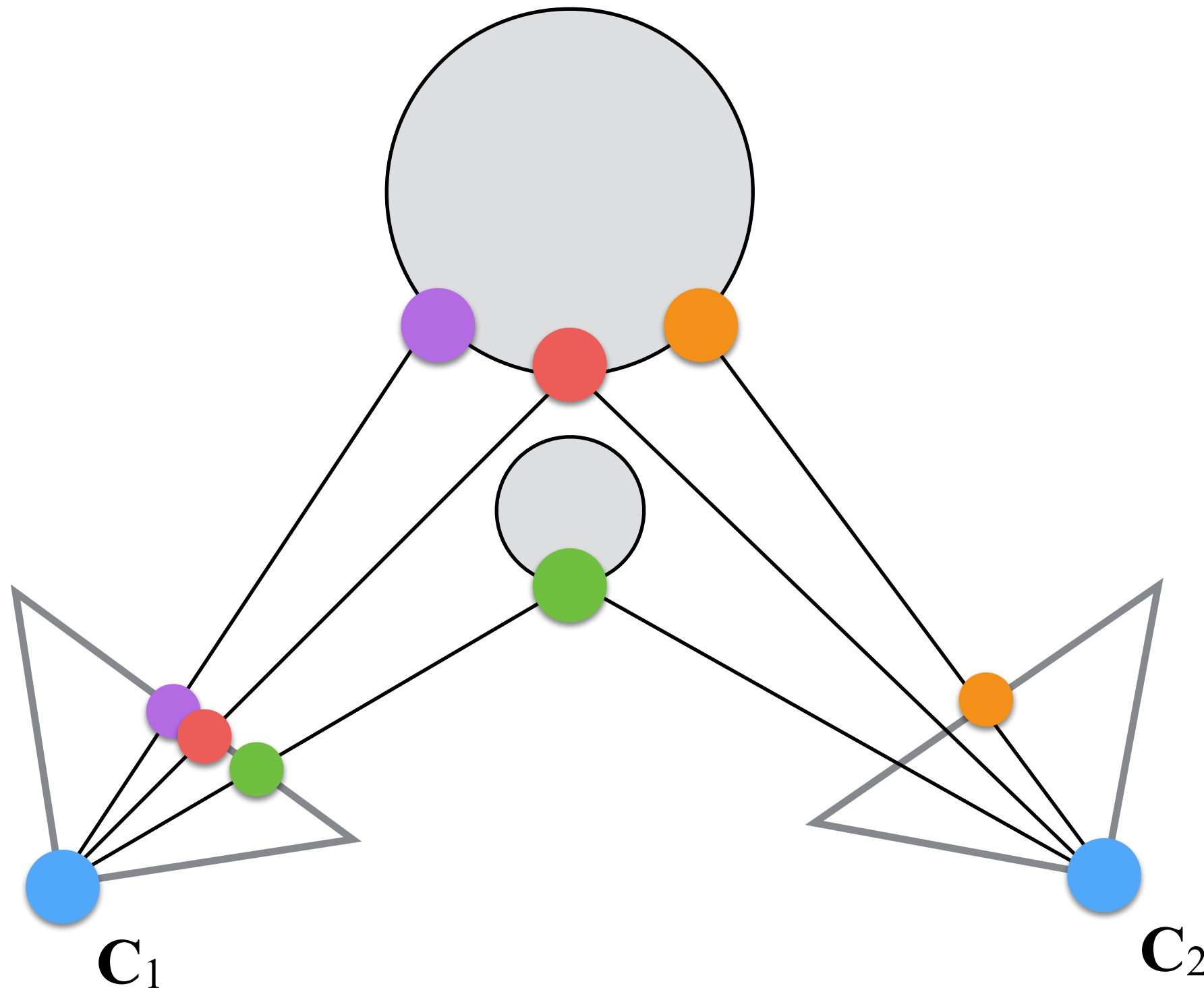


# Non-Local Constraints: Ordering Fail



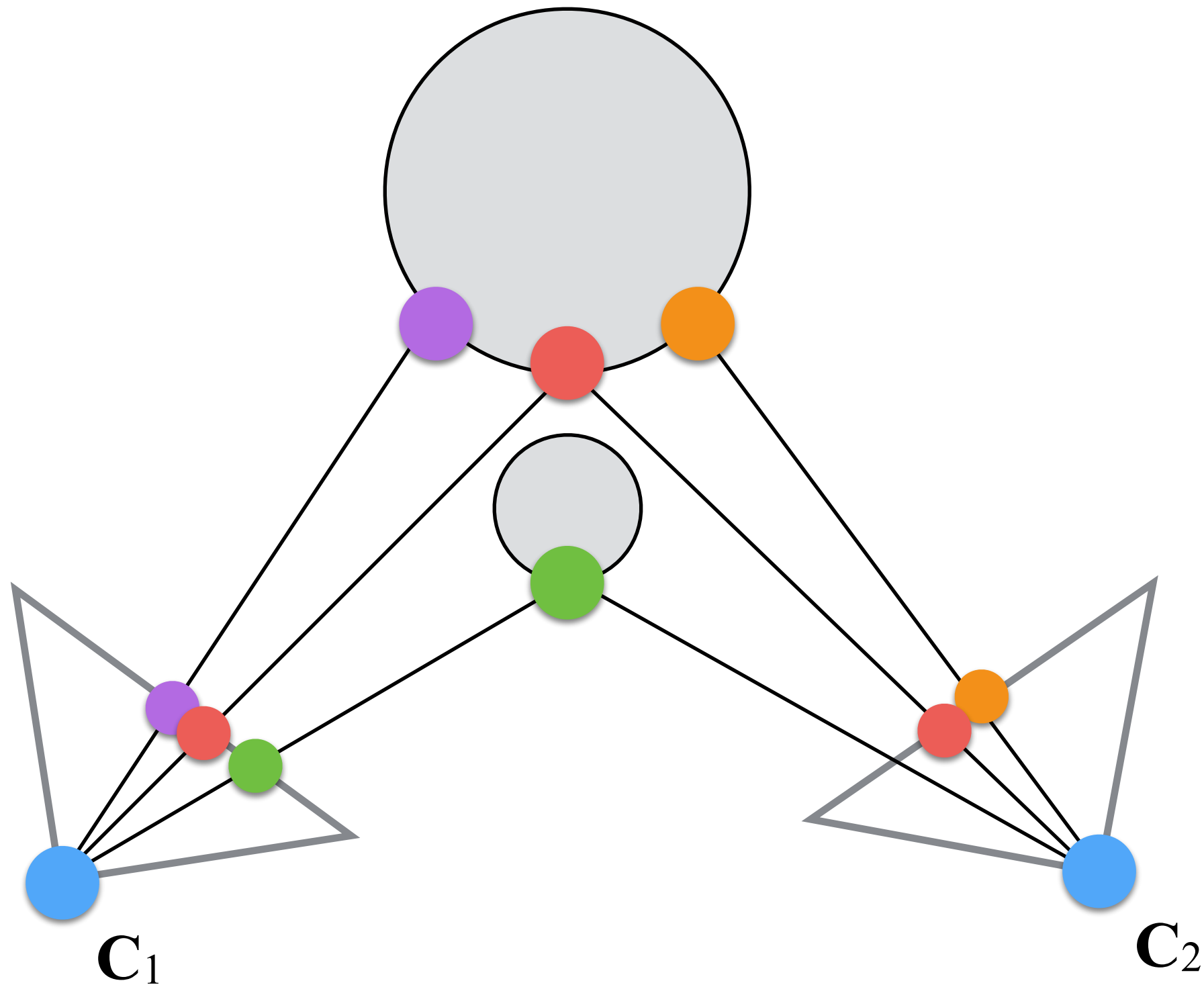


# Non-Local Constraints: Ordering Fail

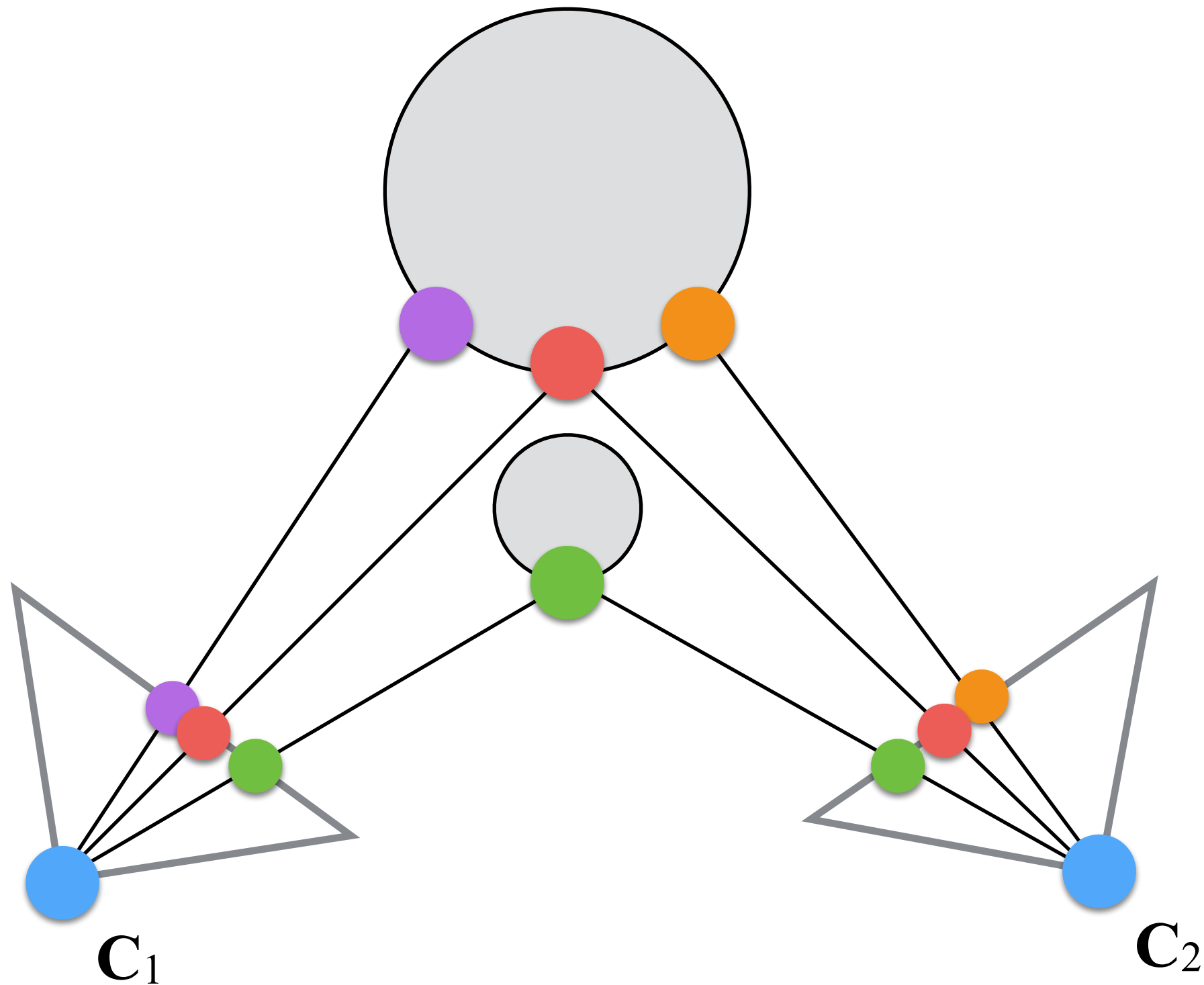




# Non-Local Constraints: Ordering Fail

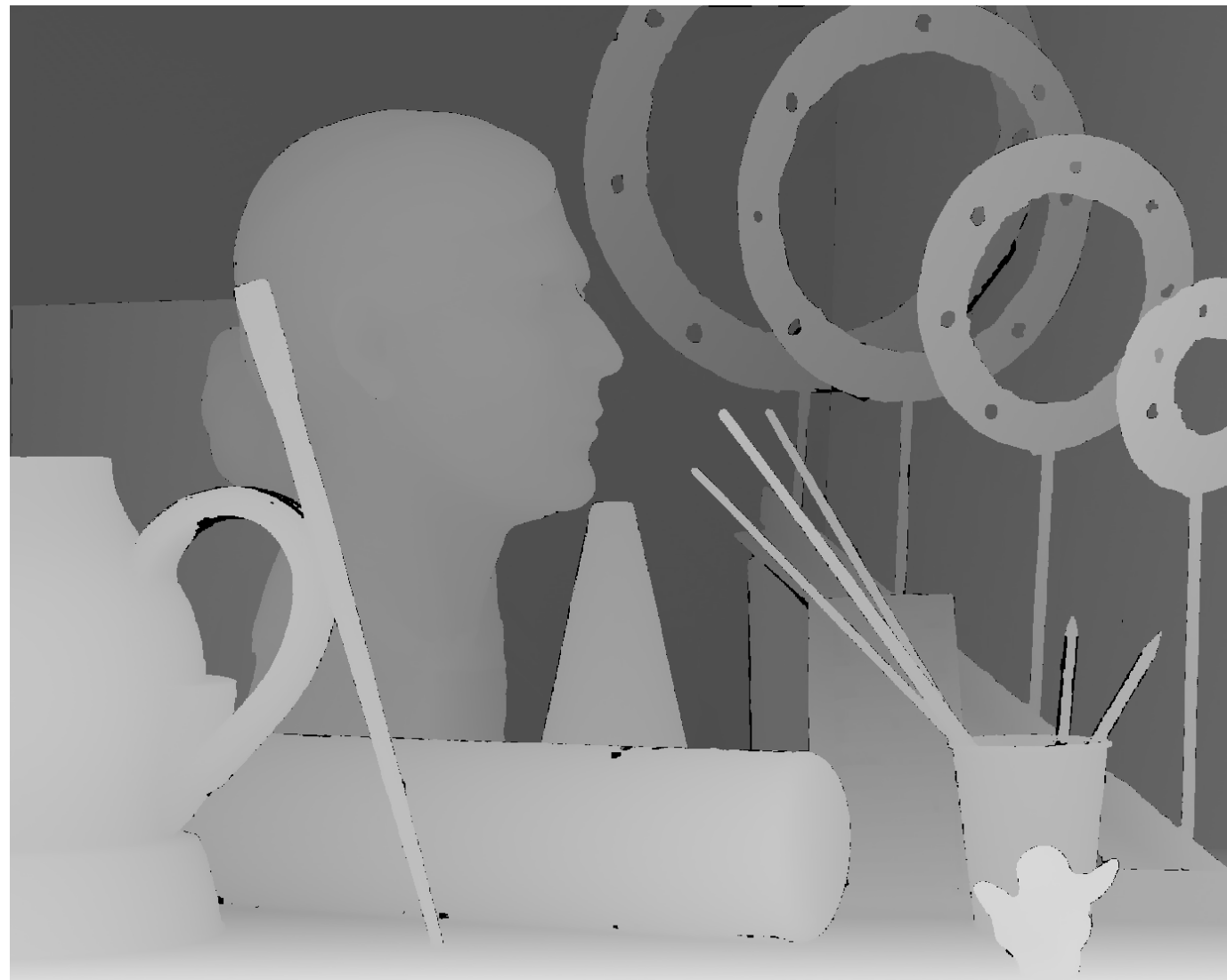


# Non-Local Constraints: Ordering Fail



# Non-Local Constraints: 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

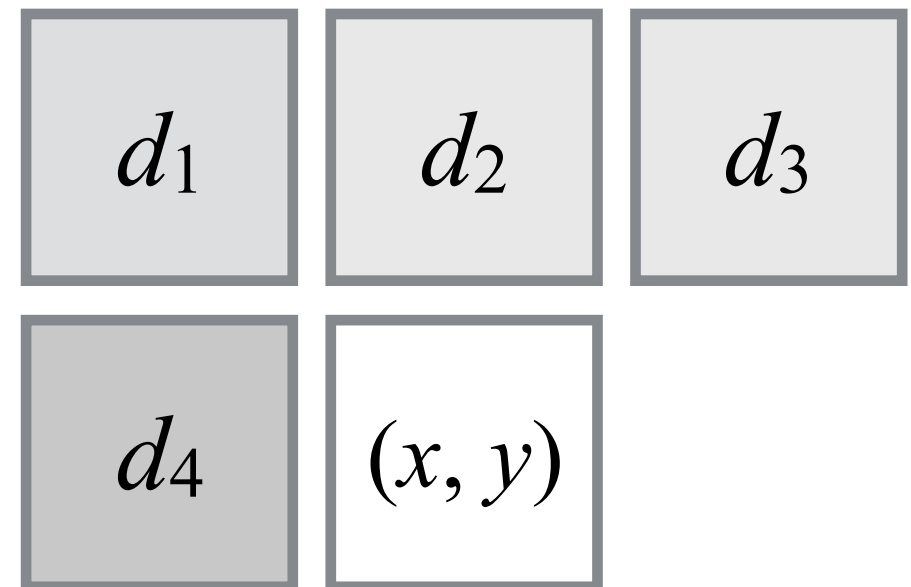
- The computation of the disparity map follows (as usual) the scan order; i.e., from left to right, from top to bottom.
- So at the current pixel  $(x, y)$ , we have already computed the disparity,  $D$ , at these previous locations:

- $d_1 = D(x - 1, y - 1)$

- $d_2 = D(x, y - 1)$

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

- $d_4 = D(x - 1, y)$



# 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:

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

# Non-Local Constraints: Smoothness Example



Left ( $I_1$ )

$$d_2 = 8$$
$$d_4 = 10$$

# Non-Local Constraints: Smoothness Example



Left ( $I_1$ )

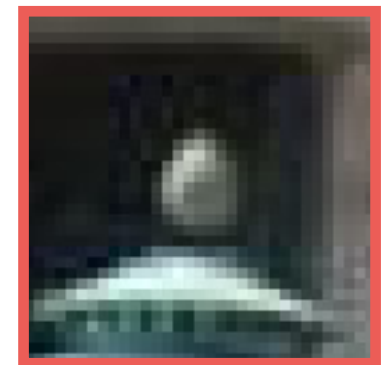
$$d_2 = 8$$
$$d_4 = 10$$



# Non-Local Constraints: Smoothness Example



Left ( $I_1$ )



$P_1$

$$d_2 = 8$$
$$d_4 = 10$$

# Non-Local Constraints: Smoothness Example



Right ( $I_2$ )

$d = -40$

$$E = \left| \begin{array}{c} \text{[Red Box]} \\ P_1 \end{array} - \begin{array}{c} \text{[Yellow Box]} \\ P_2 \end{array} \right| = 100$$

$$E_s = \frac{1}{2}|d_2 - (-40)| + \frac{1}{2}|d_4 - (-40)| = \\ = 0.5|8 - (-40)| + 0.5|10 - (-40)| = 49$$

$$E_t = E_s + \lambda E_s = 100 + 0.2 \cdot 49 = \\ = 100 + 9.8 = 109.8 \quad \lambda = 0.2$$



# Non-Local Constraints: Smoothness Example



Right ( $I_2$ )

$$d = -40$$

$$E = \left| \begin{array}{c} \text{[Red Box]} \\ P_1 \end{array} - \begin{array}{c} \text{[Yellow Box]} \\ P_2 \end{array} \right| = 100$$

$$E_s = \frac{1}{2}|d_2 - (-40)| + \frac{1}{2}|d_4 - (-40)| = \\ = 0.5|8 - (-40)| + 0.5|10 - (-40)| = 49$$

$$E_t = E_s + \lambda E_s = 100 + 0.2 \cdot 49 = \\ = 100 + 9.8 = 109.8 \quad \lambda = 0.2$$

# Non-Local Constraints: Smoothness Example



Right ( $I_2$ )

$$d = -16$$

$$E = \left| \begin{array}{c} \text{[Red Box]} \\ P_1 \end{array} - \begin{array}{c} \text{[Yellow Box]} \\ P_2 \end{array} \right| = 32$$

$$E_s = \frac{1}{2}|d_2 - (-16)| + \frac{1}{2}|d_4 - (-16)| = \\ = 0.5|8 - (-16)| + 0.5|10 - (-16)| = 25$$

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



# Non-Local Constraints: Smoothness Example



Right ( $I_2$ )

$$d = -16$$

$$E = \left| \begin{array}{c} \text{[Red Box]} \\ P_1 \end{array} - \begin{array}{c} \text{[Yellow Box]} \\ P_2 \end{array} \right| = 32$$

$$E_s = \frac{1}{2}|d_2 - (-16)| + \frac{1}{2}|d_4 - (-16)| = \\ = 0.5|8 - (-16)| + 0.5|10 - (-16)| = 25$$

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



# Non-Local Constraints: Smoothness Example

$$E_t \left( \begin{array}{c} \text{[Image 1]} \quad \text{[Image 2]} \\ d = -40 \end{array} \right) = 109.8$$

$$E_t \left( \begin{array}{c} \text{[Image 1]} \quad \text{[Image 3]} \\ d = -16 \end{array} \right) = 37$$

# Non-Local Constraints: Smoothness Example

$$E_t \left( \begin{array}{|c|c|} \hline \text{Image 1} & \text{Image 2} \\ \hline \end{array} \right) = 109.8$$

$d = -40$

$$E_t \left( \begin{array}{|c|c|} \hline \text{Image 1} & \text{Image 3} \\ \hline \end{array} \right) = 37$$

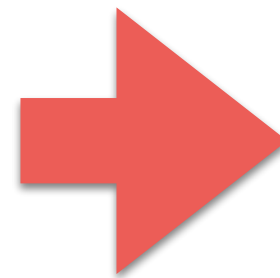
$d = -16$

**This is lower than 109.8!**

# Non-Local Constraints: Smoothness Example

$$E_t \left( \begin{array}{|c|c|} \hline \text{Image 1} & \text{Image 2} \\ \hline \end{array} \right) = 109.8$$

$d = -40$



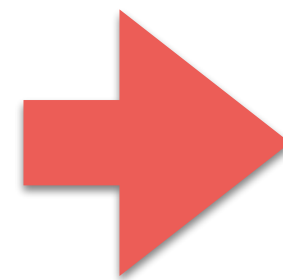
$$E_t \left( \begin{array}{|c|c|} \hline \text{Image 1} & \text{Image 3} \\ \hline \end{array} \right) = 37$$

$d = -16$

**This is lower than 109.8!**

# Non-Local Constraints: Smoothness Example

$$E_t \left( \begin{array}{c} \text{[Image 1]} \quad \text{[Image 2]} \\ d = -40 \end{array} \right) = 109.8$$



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

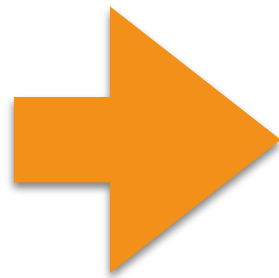
$$E_t \left( \begin{array}{c} \text{[Image 1]} \quad \text{[Image 3]} \\ d = -16 \end{array} \right) = 37$$

**This is lower than 109.8!**



How do we choose  
parameters?

# Dense Matching: Choosing $n$

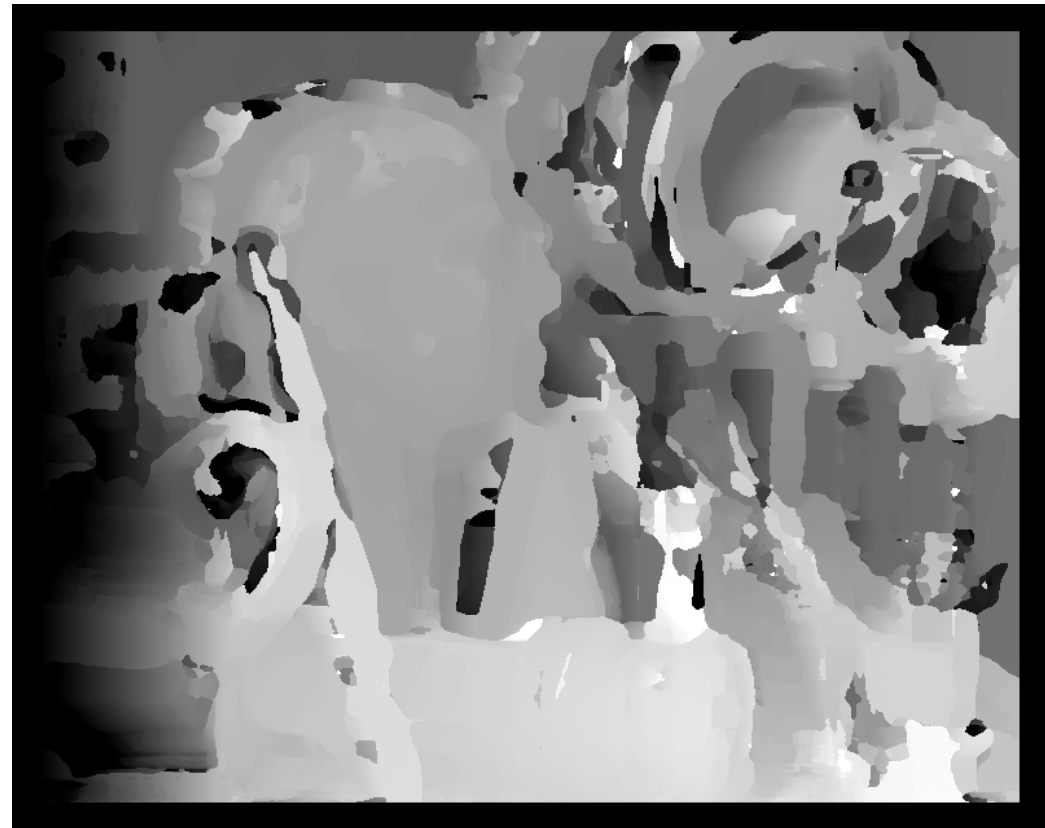
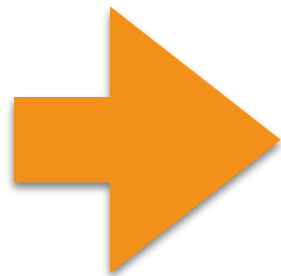


Input

$n = 3$

- Smaller  $n$   $\rightarrow$  more details but more noise!

# Dense Matching: Choosing $n$



Input

$n = 21$

- Larger  $n$   $\longrightarrow$  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 if this satisfies the following:

$$C(n) = \overline{E_t} + \alpha \cdot \text{Var}(E_t) + \frac{\beta}{n + \gamma}$$
$$\alpha = 1.5 \quad \beta = 7 \quad \gamma = -2$$

# 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



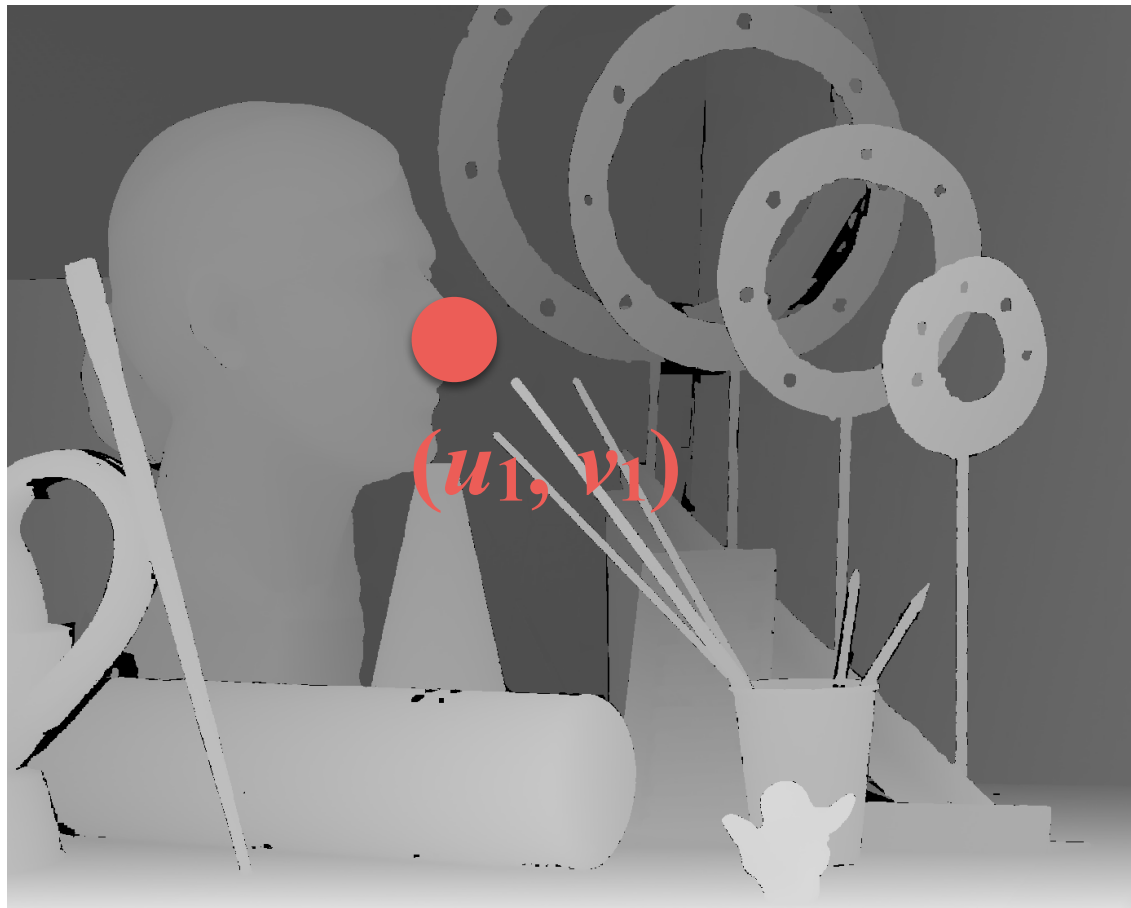
Left



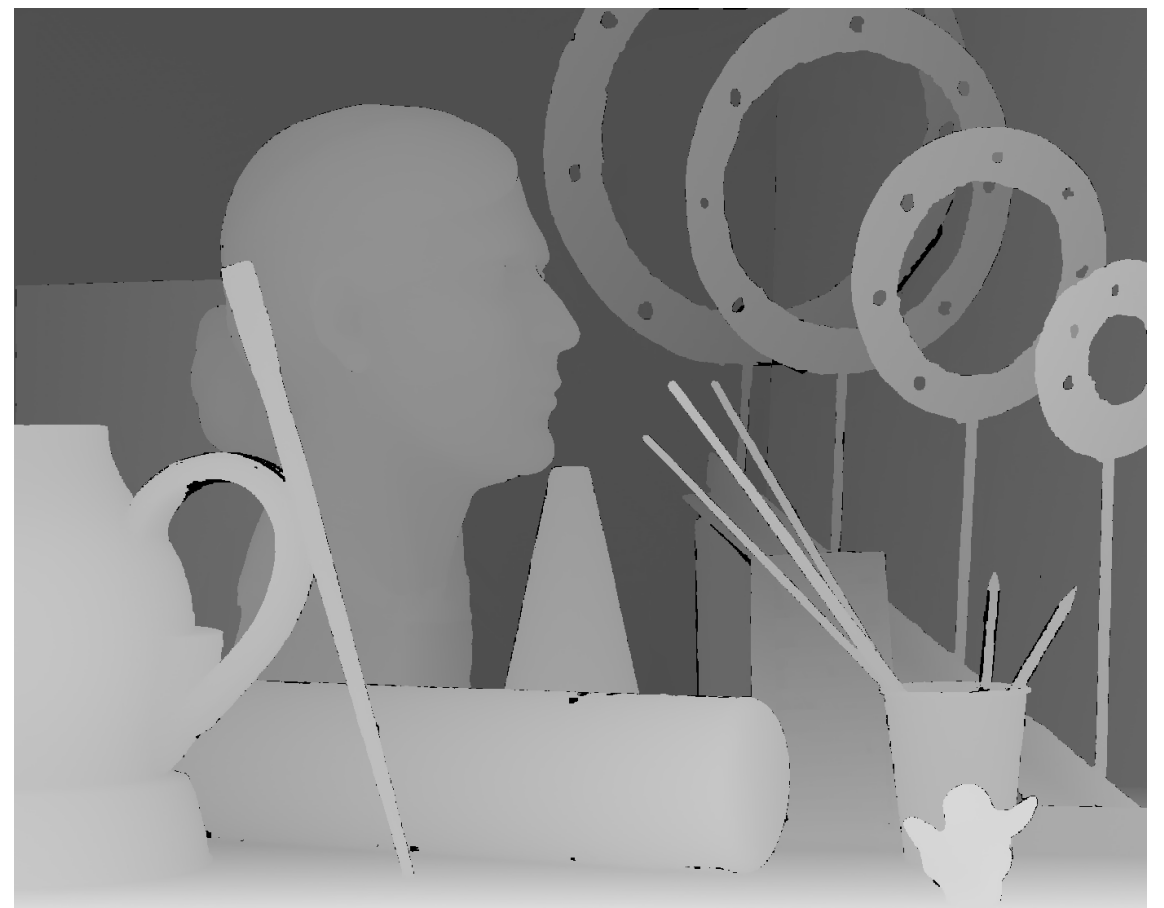
Right



# Handling Occlusions: Example

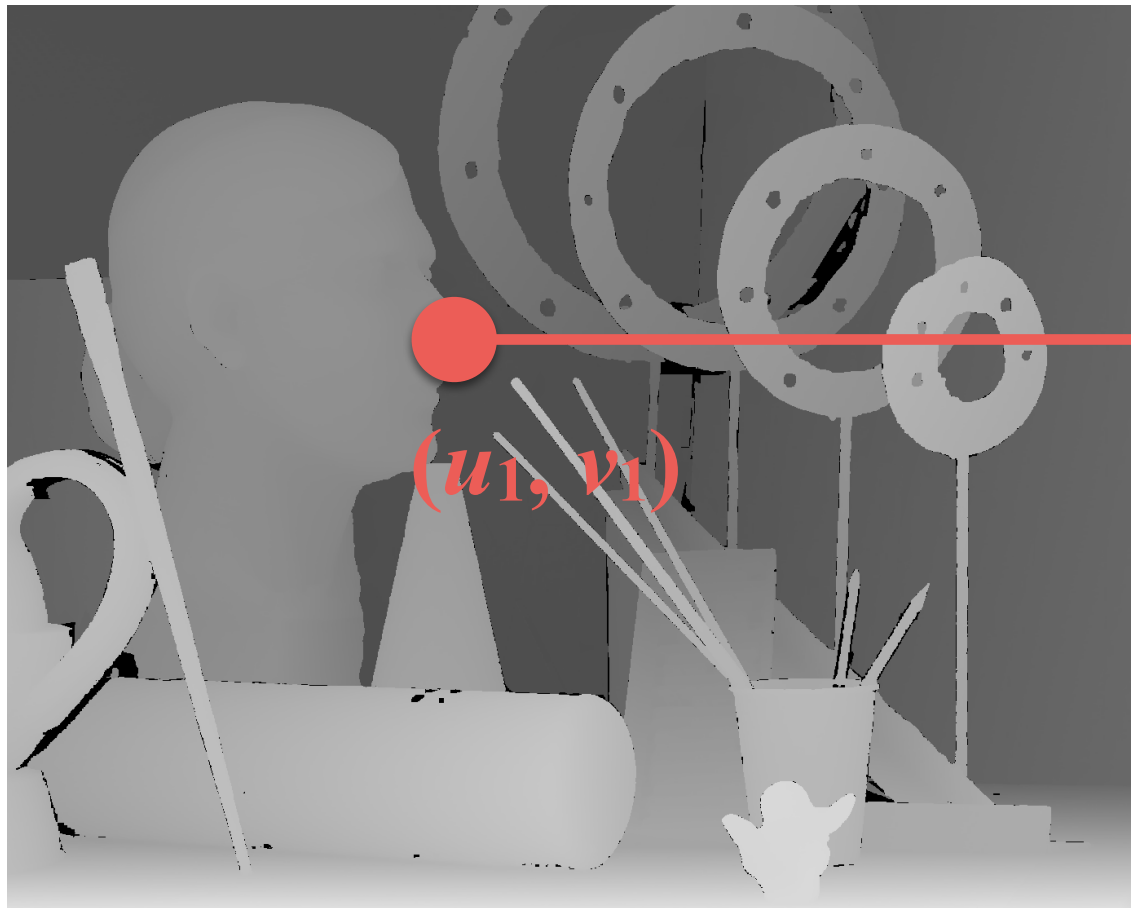


Left

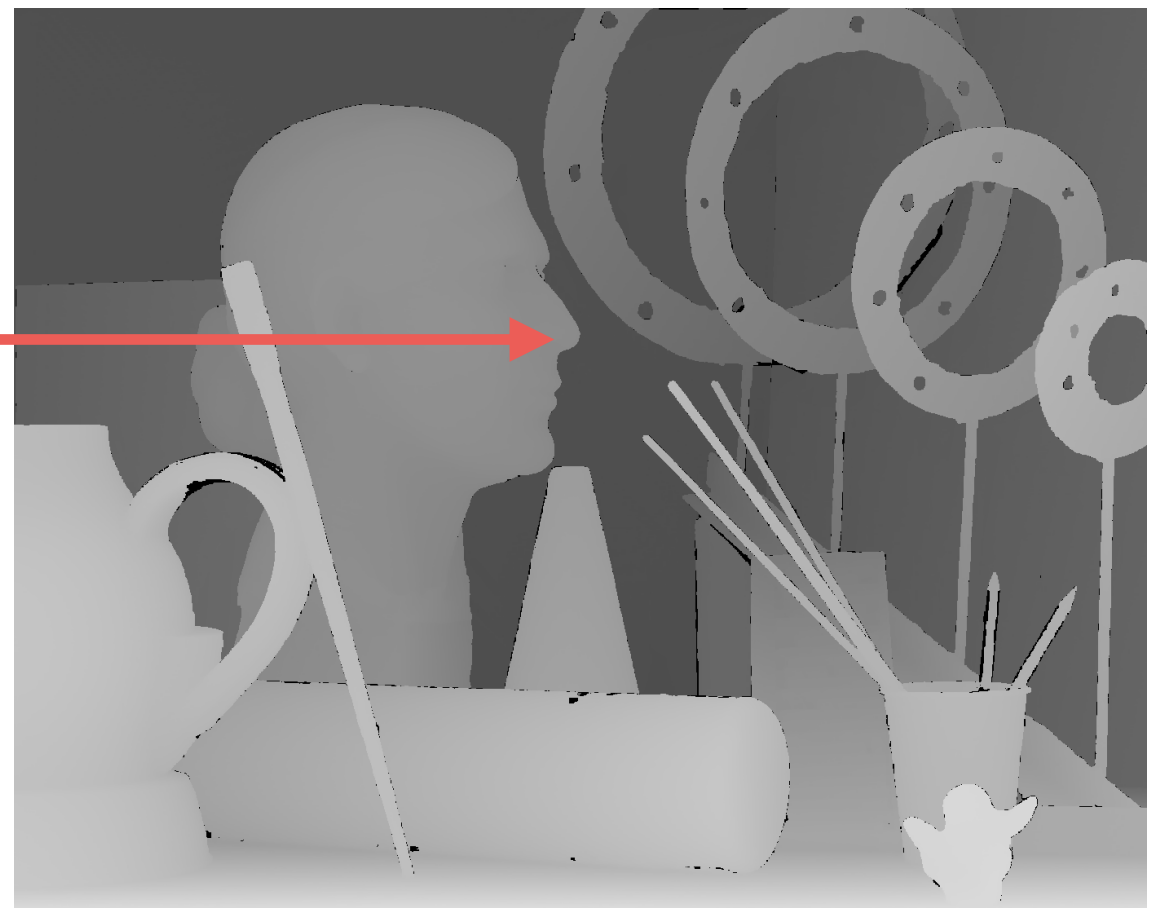


Right

# Handling Occlusions: Example

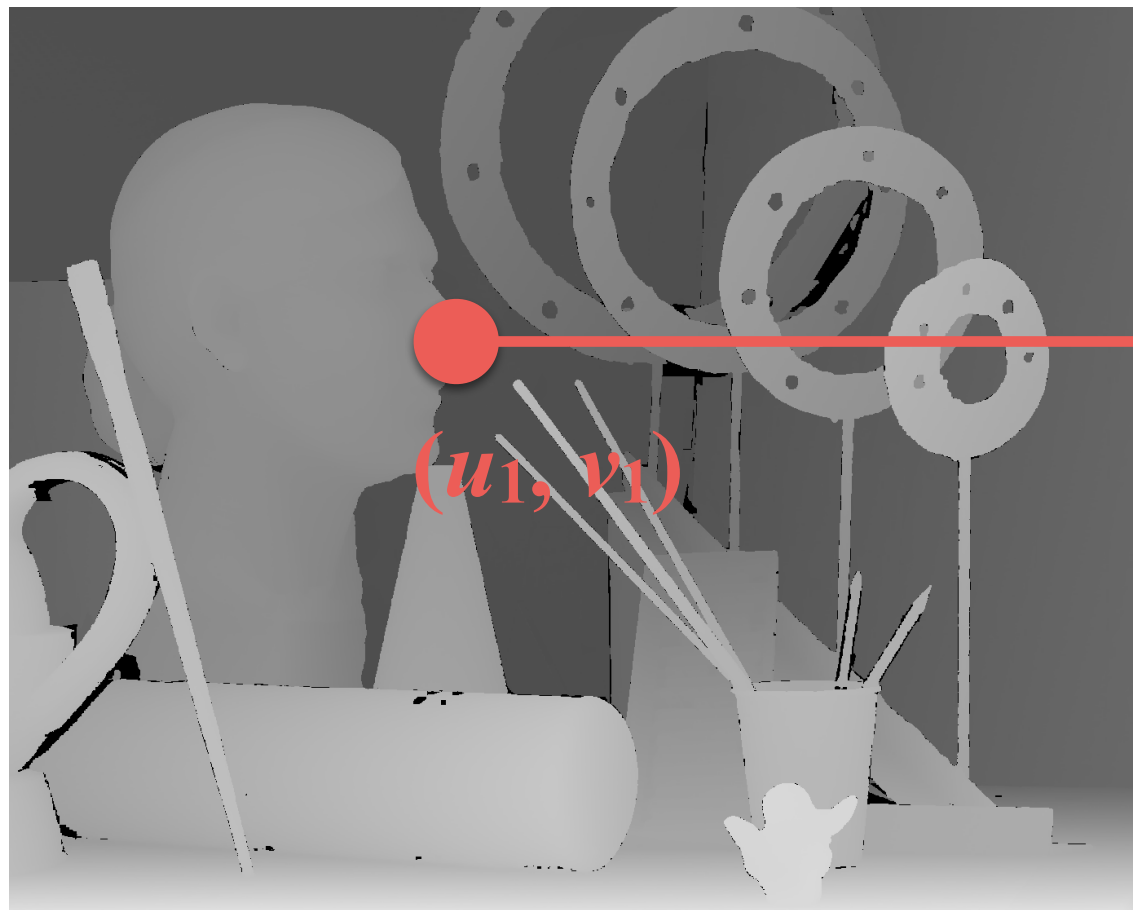


Left

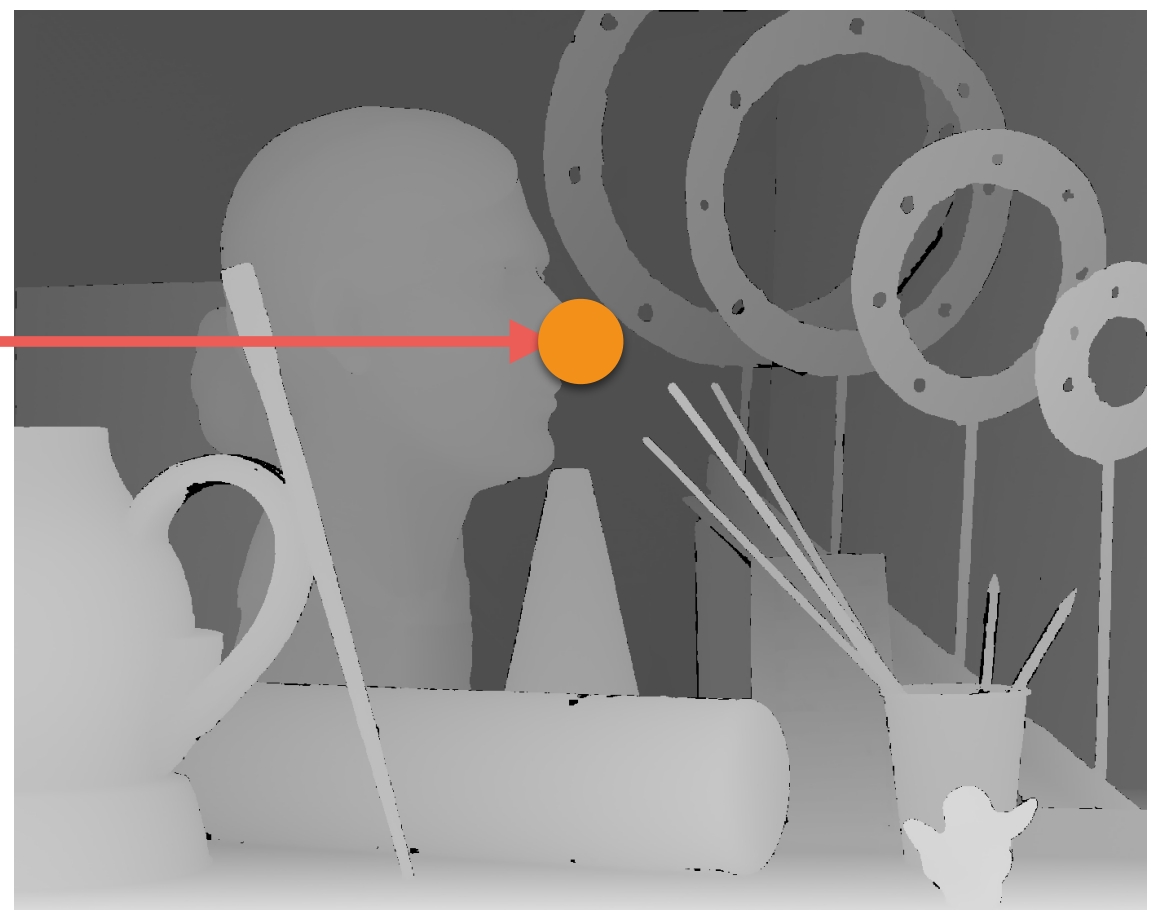


Right

# Handling Occlusions: Example

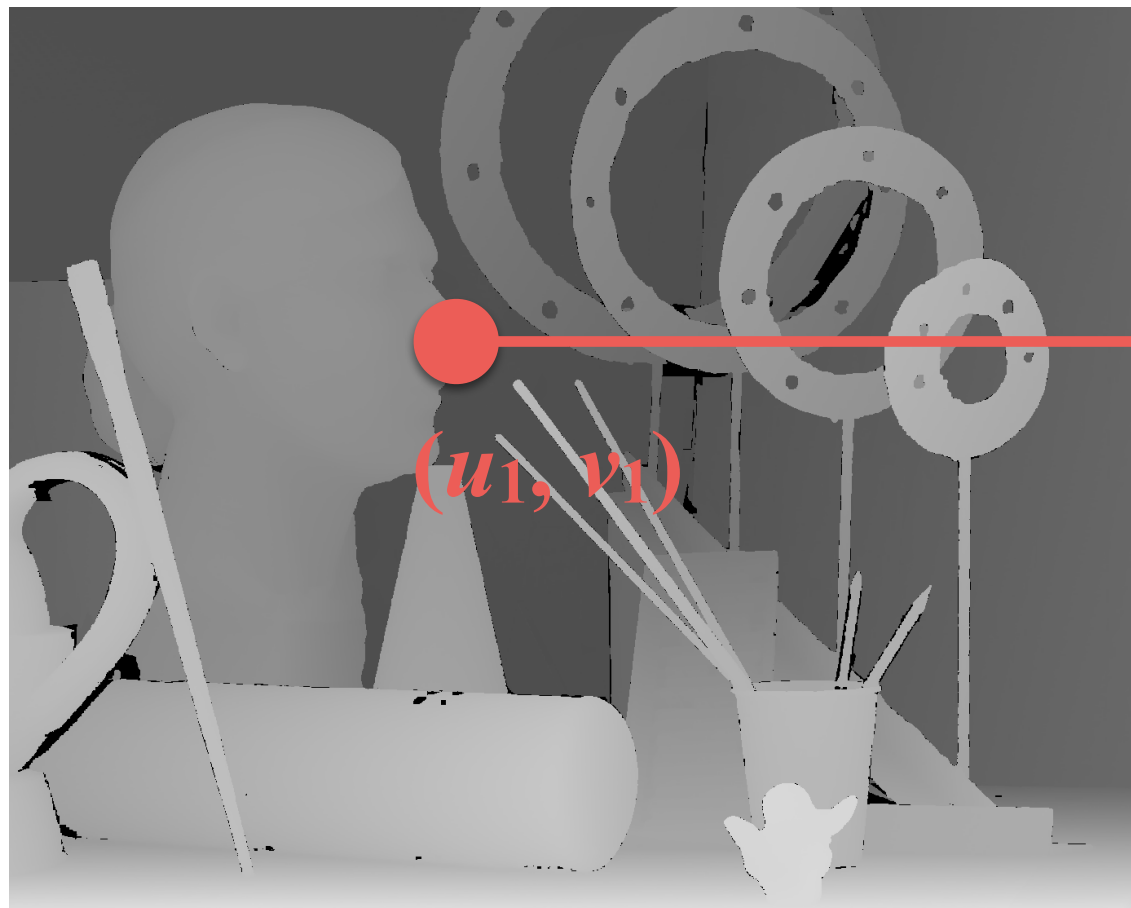


Left

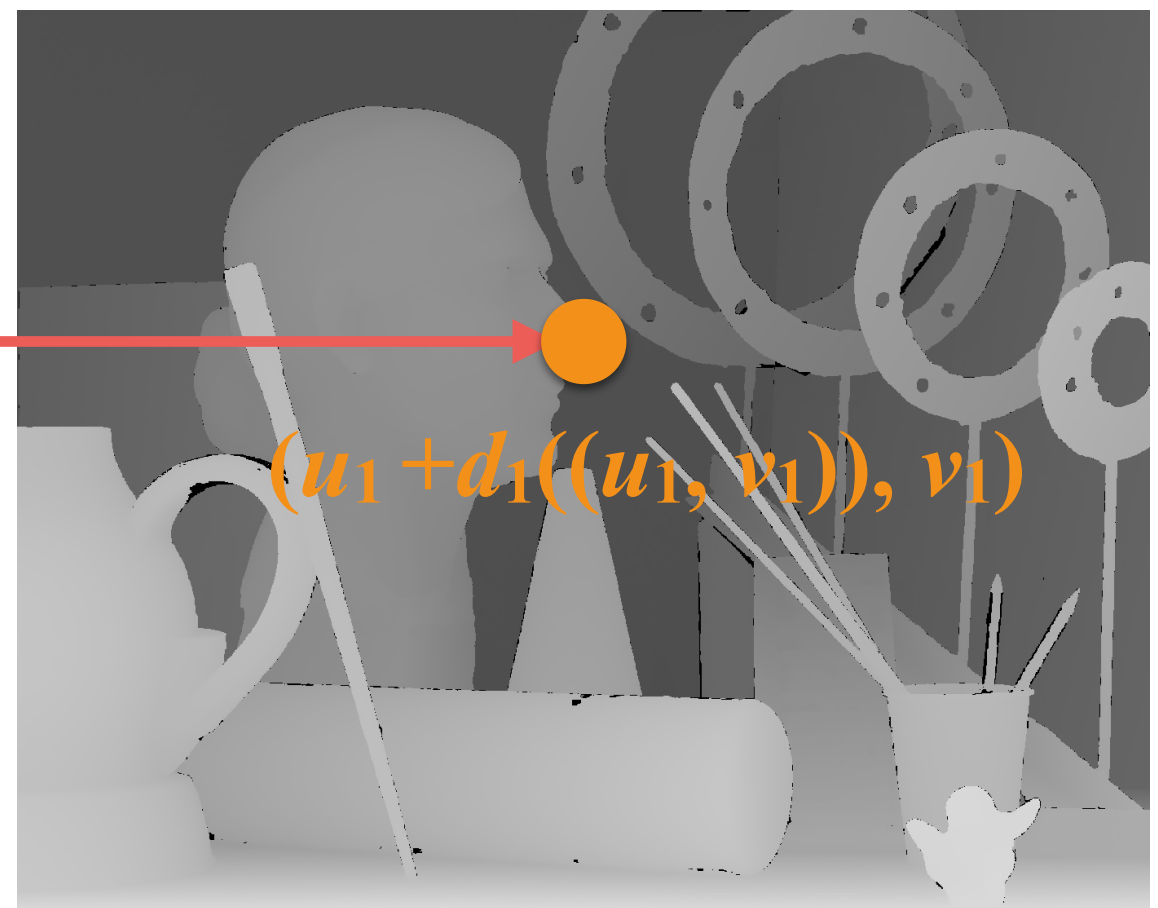


Right

# Handling Occlusions: Example



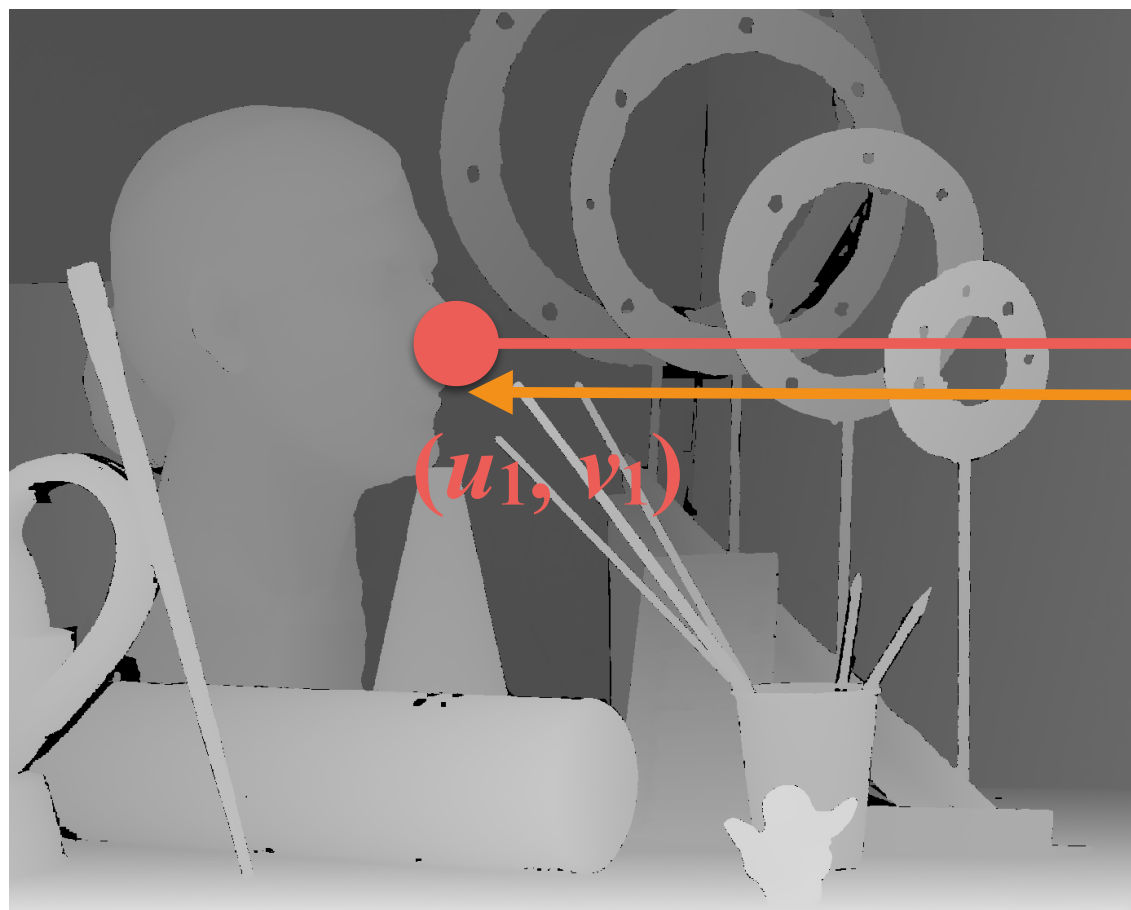
Left



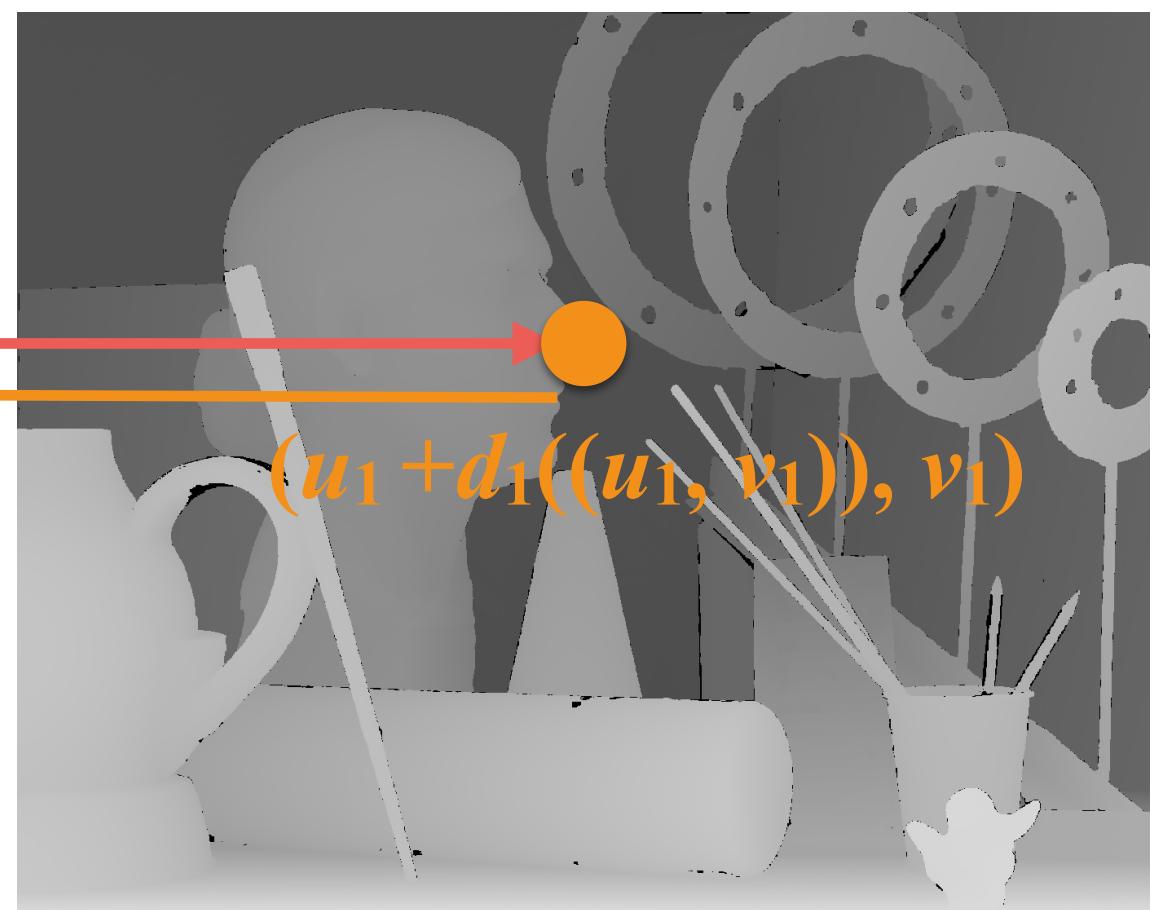
Right



# Handling Occlusions: Example

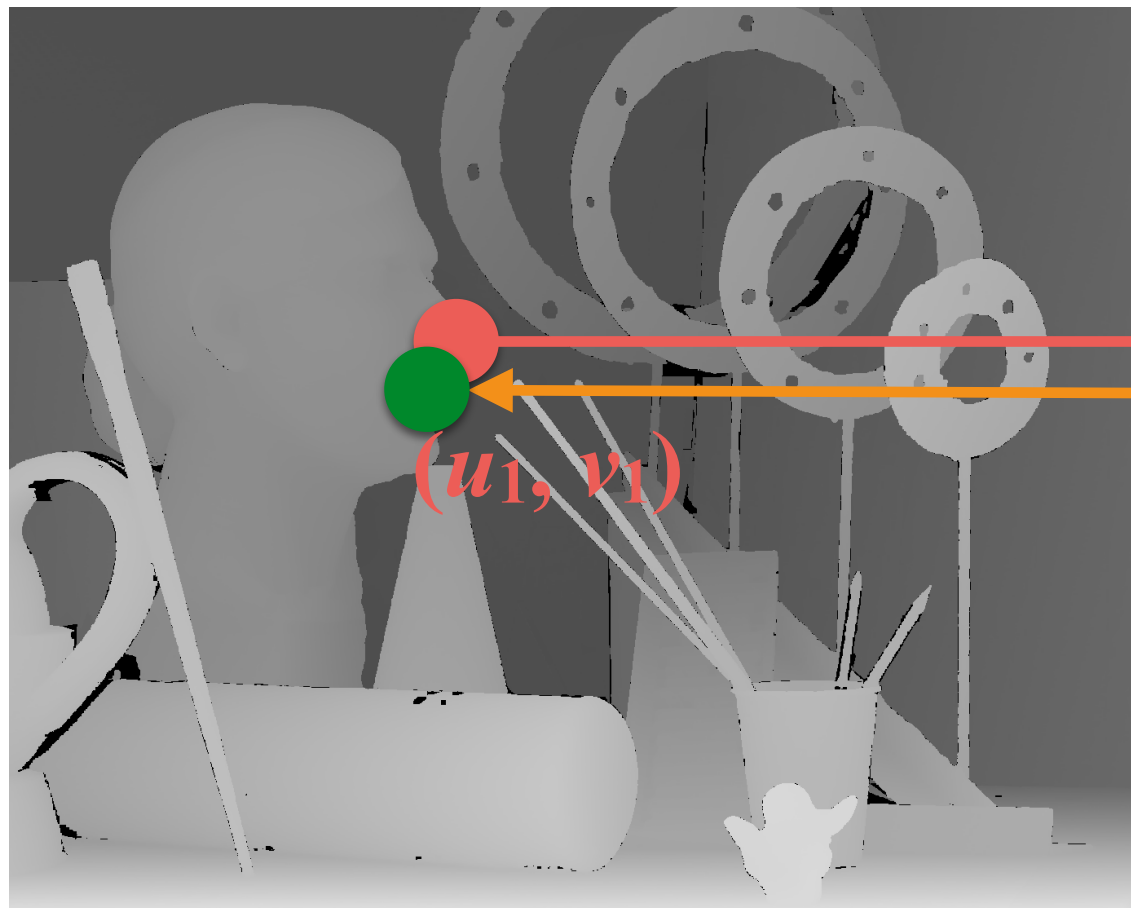


Left

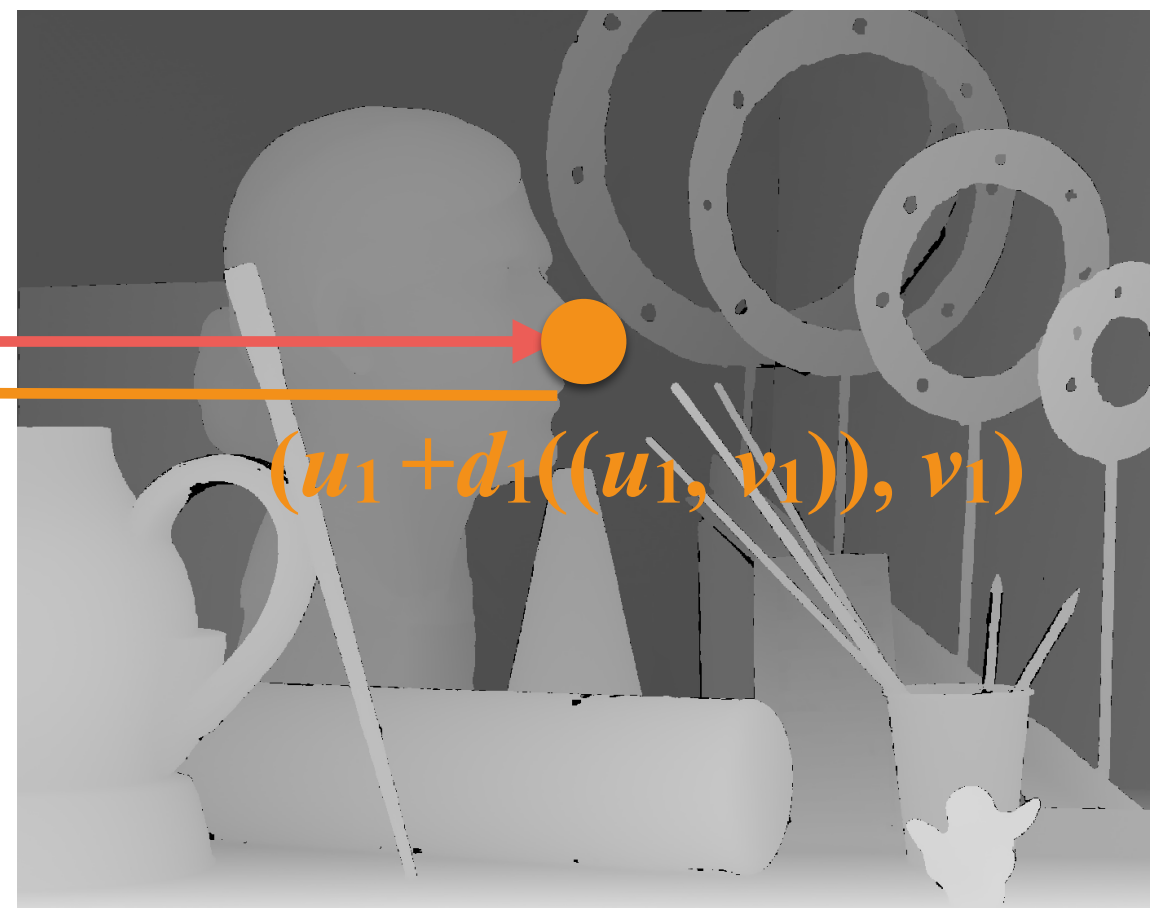


Right

# Handling Occlusions: Example

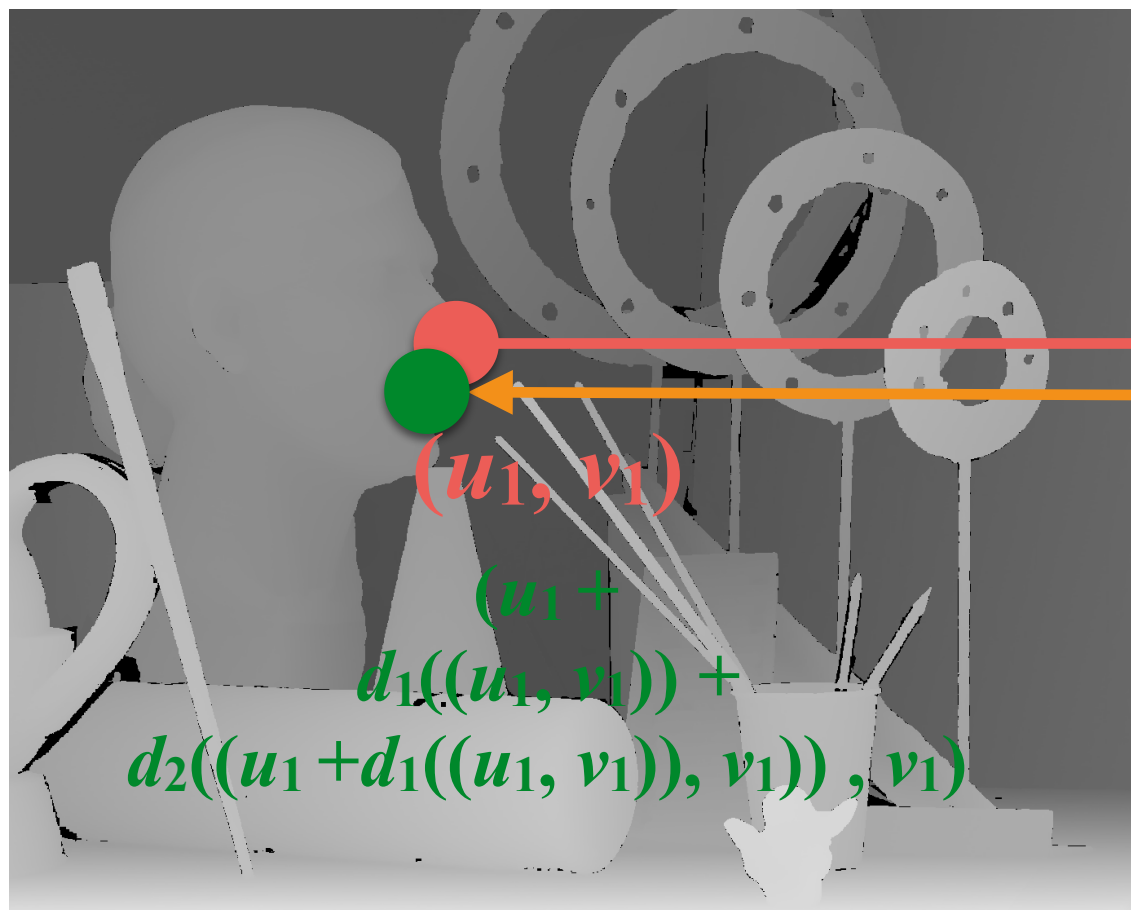


Left

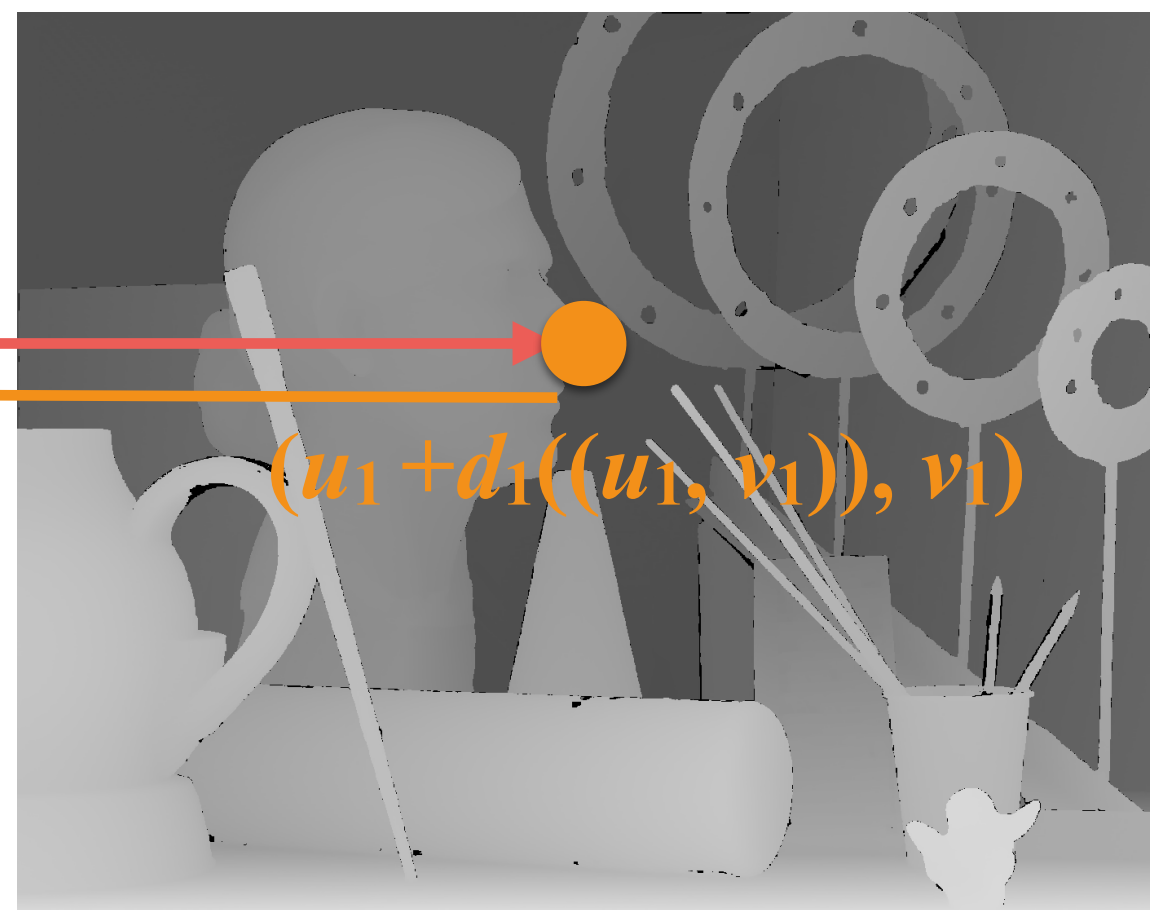


Right

# Handling Occlusions: Example

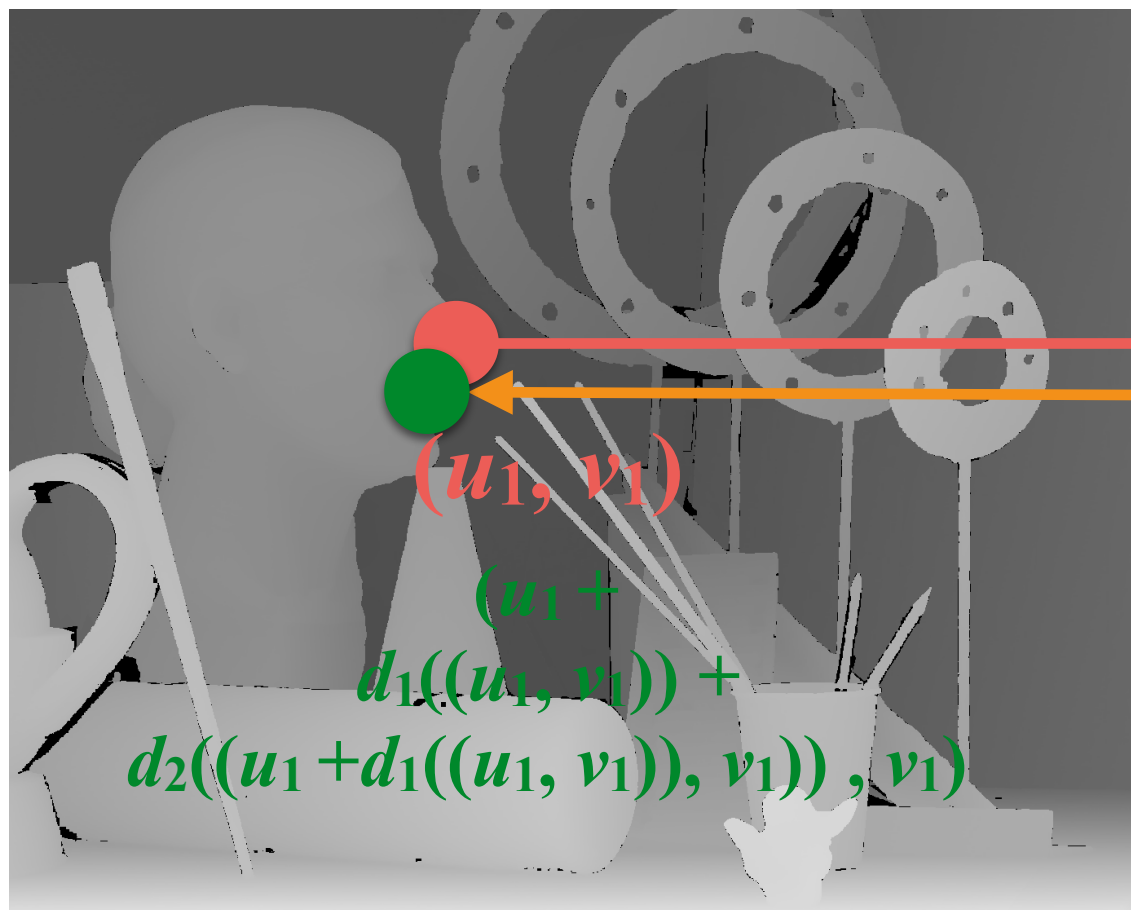


Left

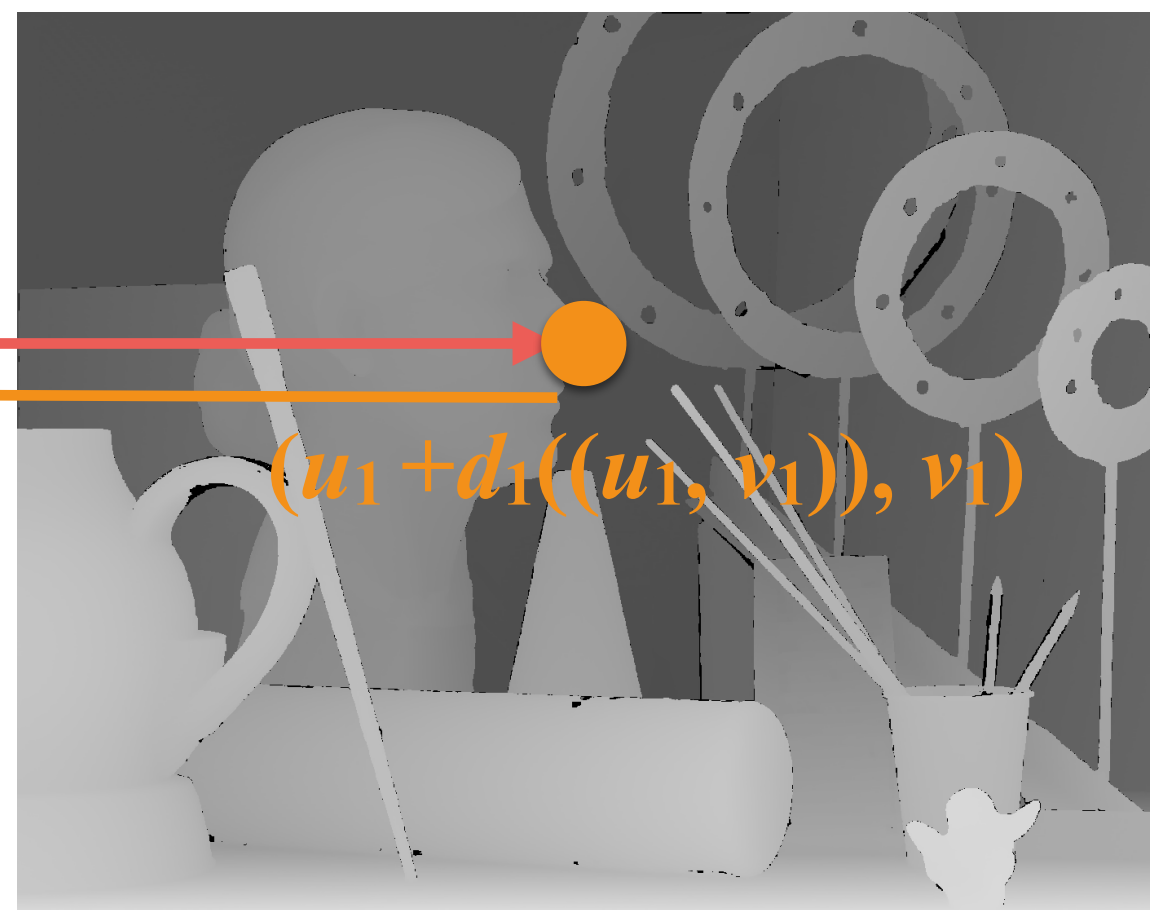


Right

# Handling Occlusions: Example



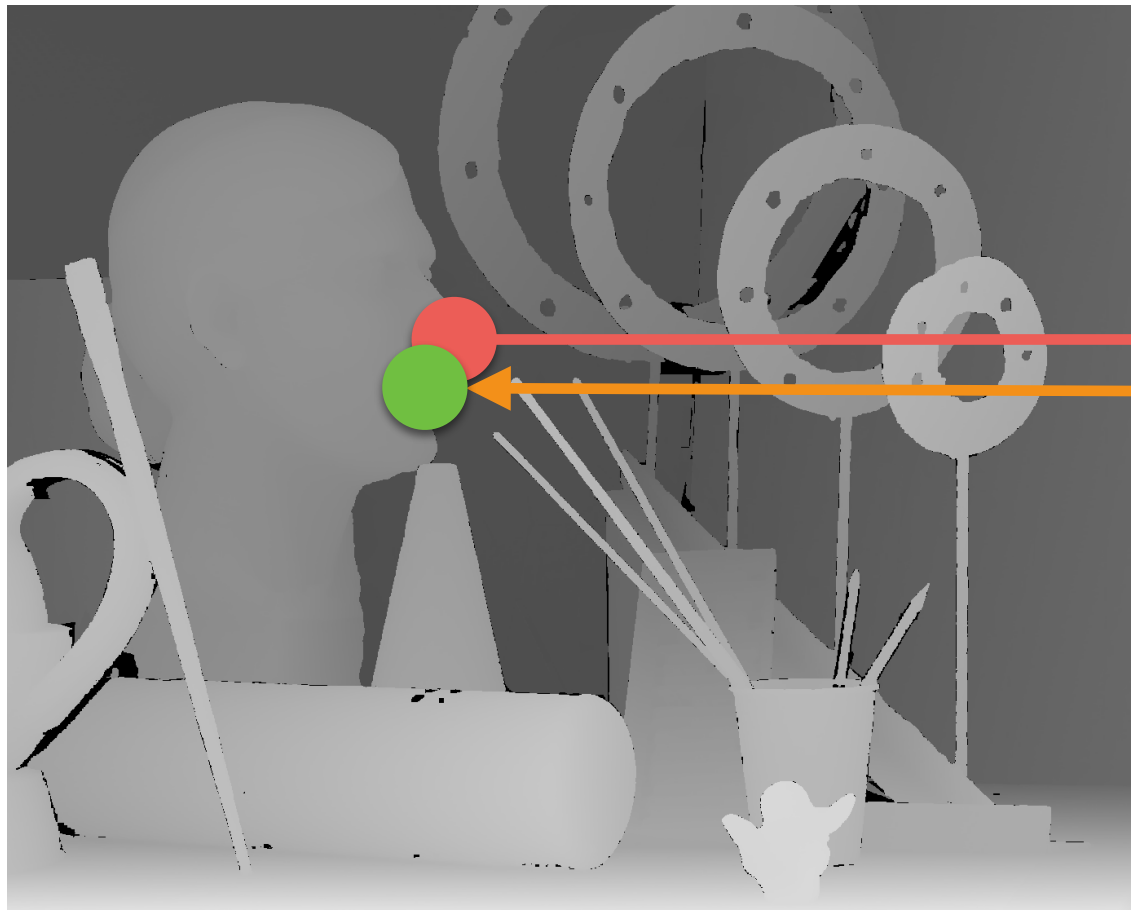
Left



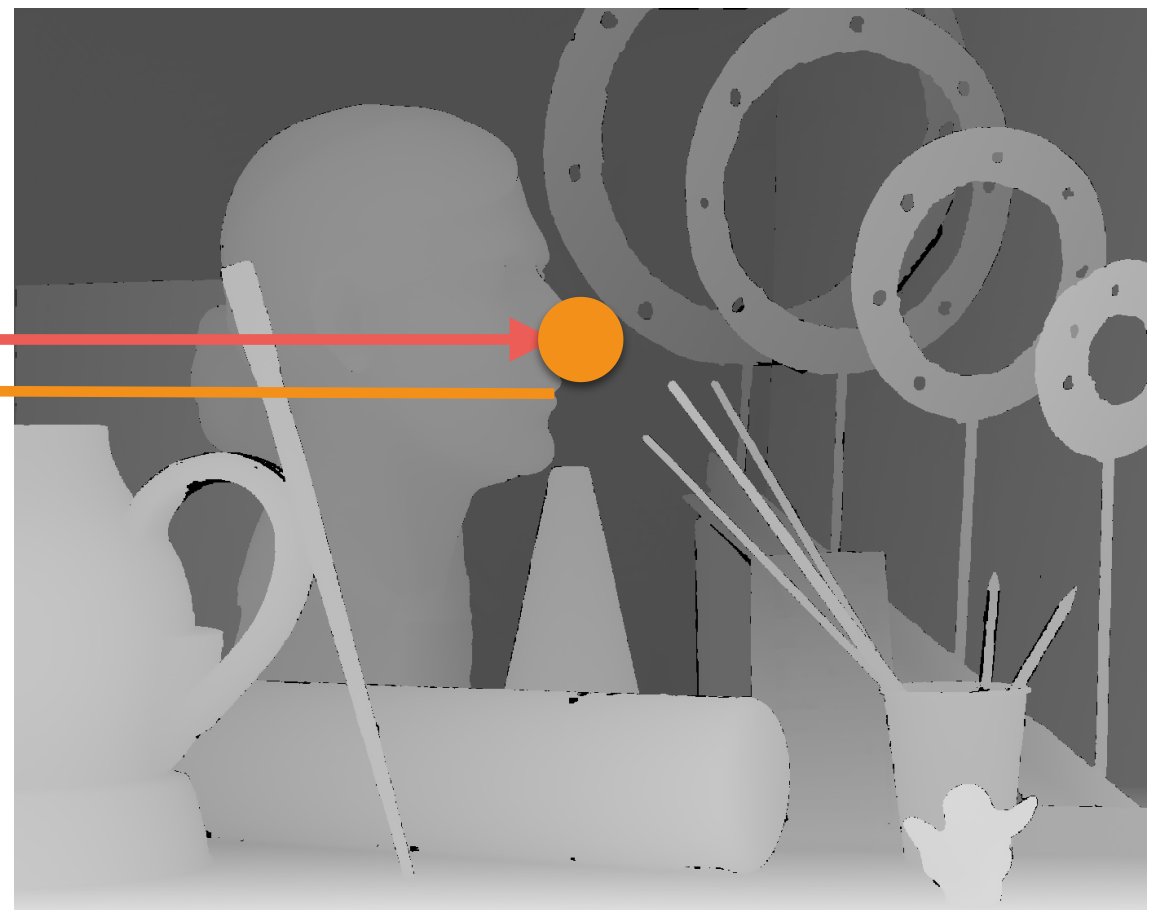
Right

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)

$$D = d_1(u_1, v_1)$$

$$u'_2 = u_1 + D$$

$$D' = d_2(u_1 + D, v_1)$$

$$u'_1 = u_2 + D'$$

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

# 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

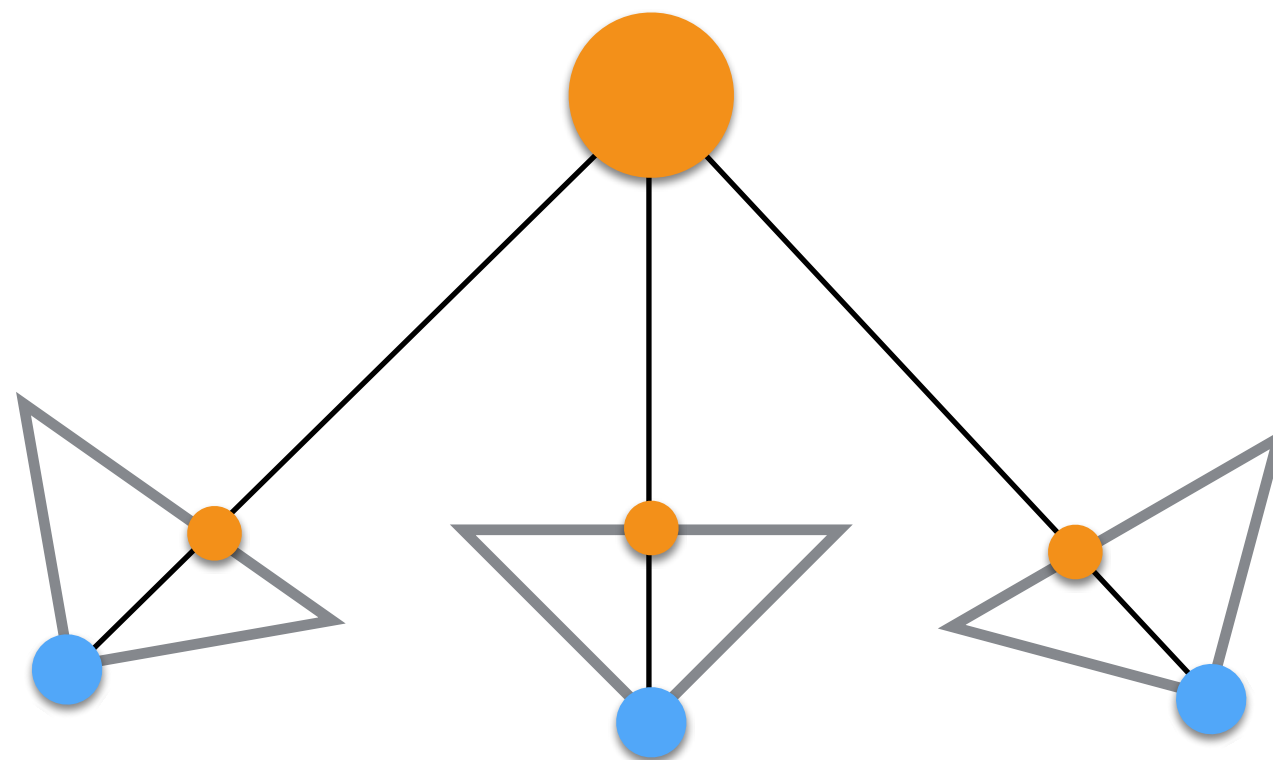
- There are three main approaches:
  - Computing depth-maps from  $n$  images:
    - The same formulation of Stereo, but more images!
  - Volume Carving.
  - Propagating the known 3D information of the sparse point cloud.

# 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.



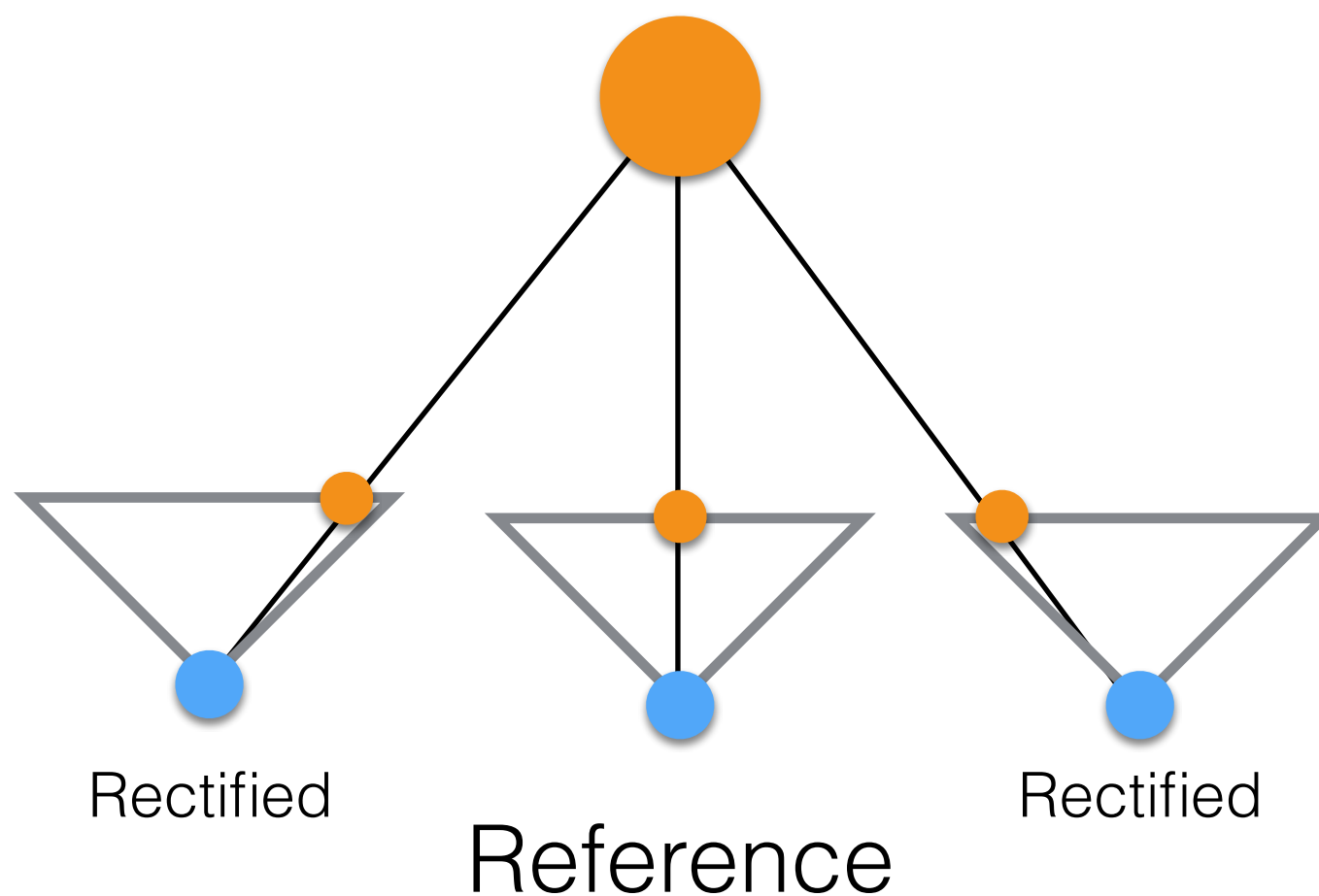
# Multi-View Stereo: Stereo Extension



Reference

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

# Multi-View Stereo: Stereo Extension



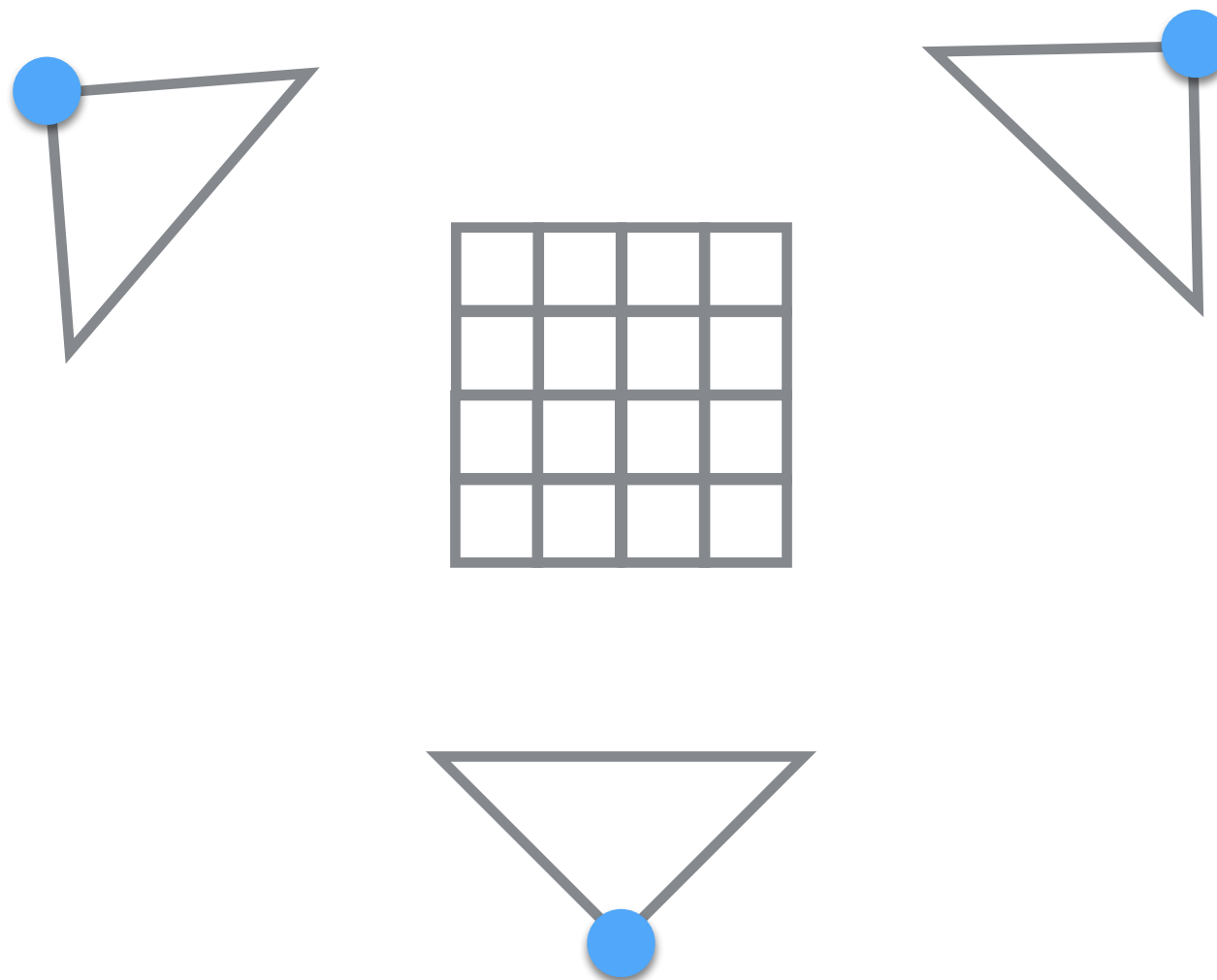
# Multi-View Stereo: Stereo Extension

- 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

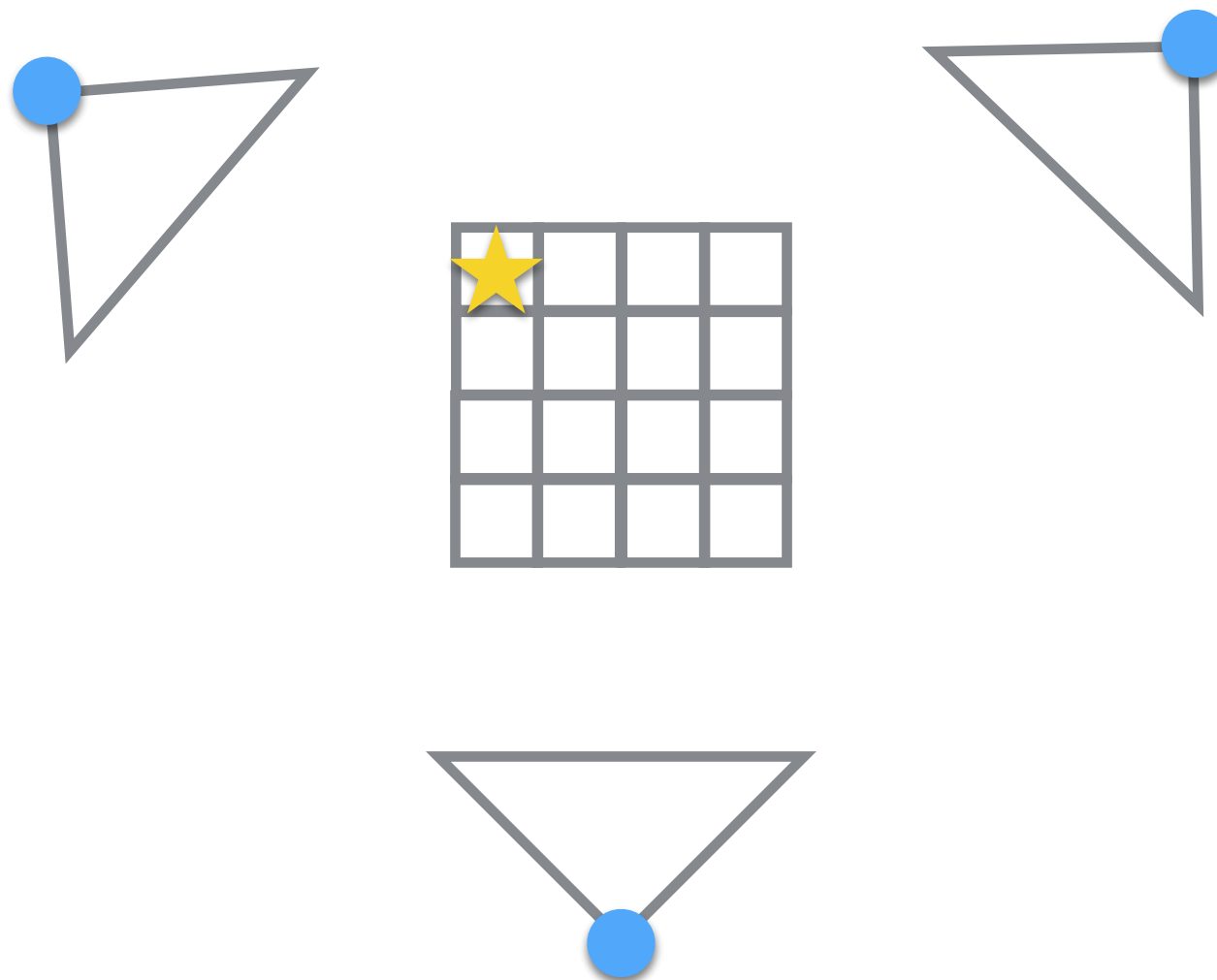
- Space carving is an algorithm with a volumetric-approach:
  - We compute the bounding box (BB) of triangulated 3D points.
  - We generate a 3D volume out of BB.
  - We carve voxel in the volume according to views.

# Multi-View Stereo: Space Carving

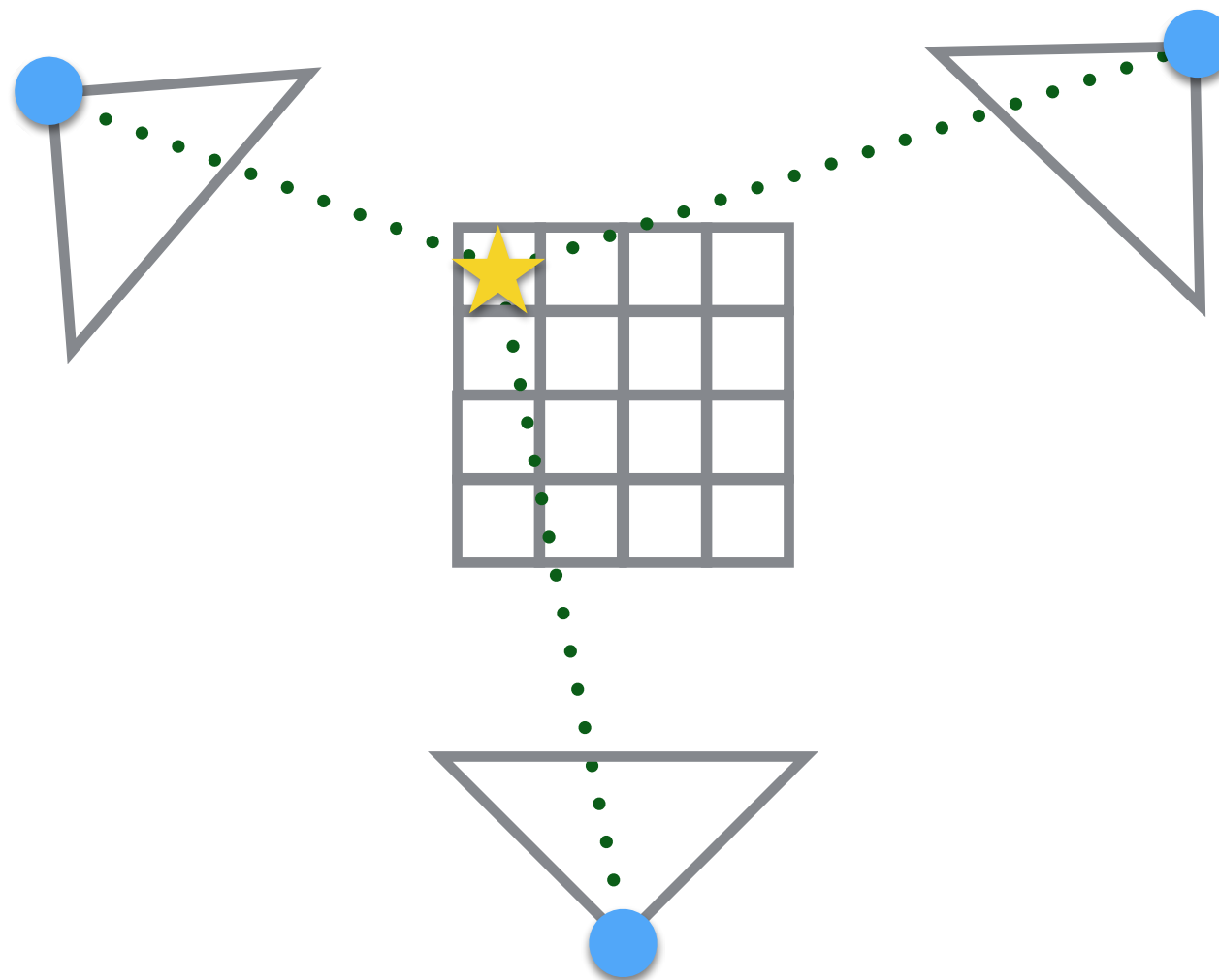




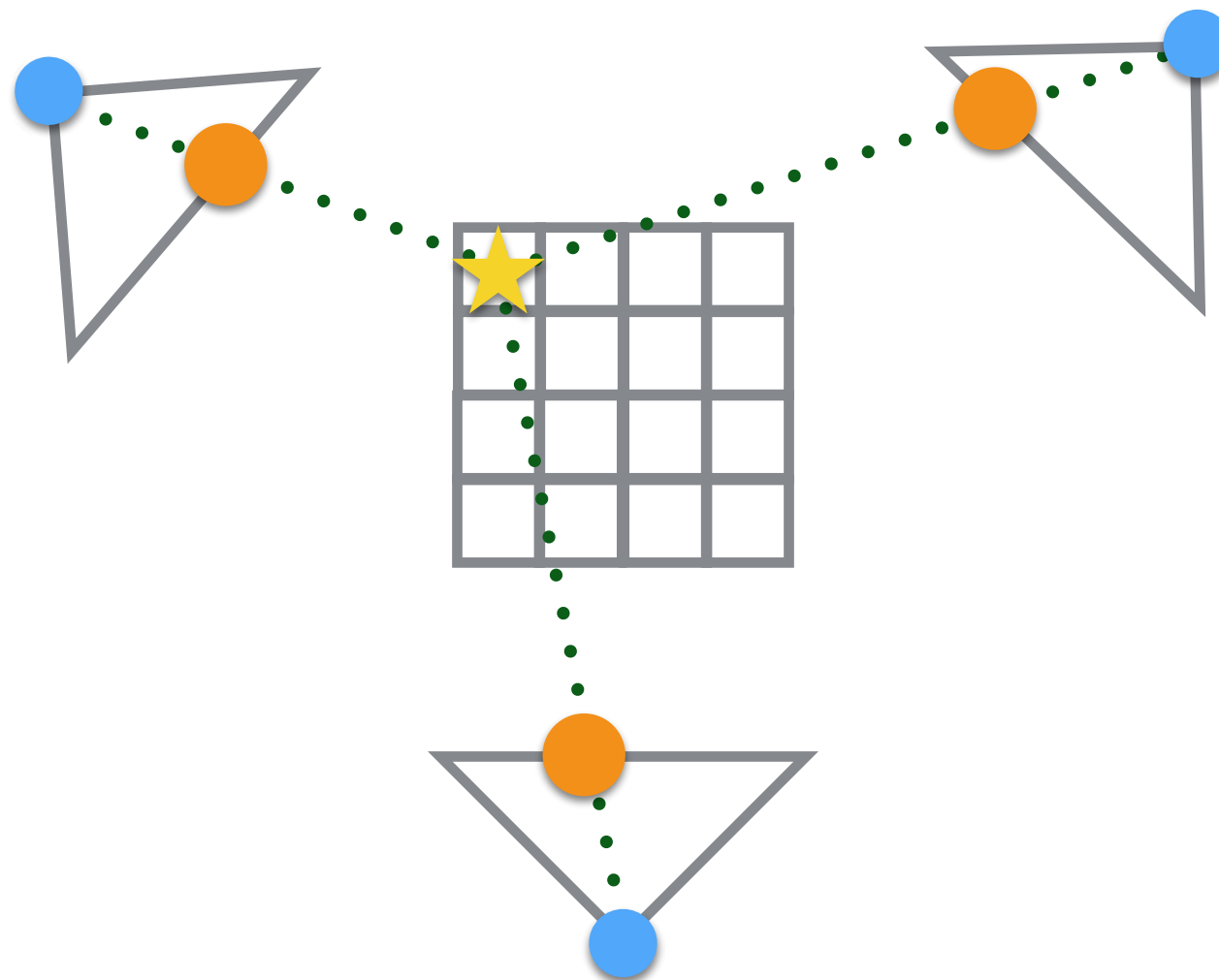
# Multi-View Stereo: Space Carving



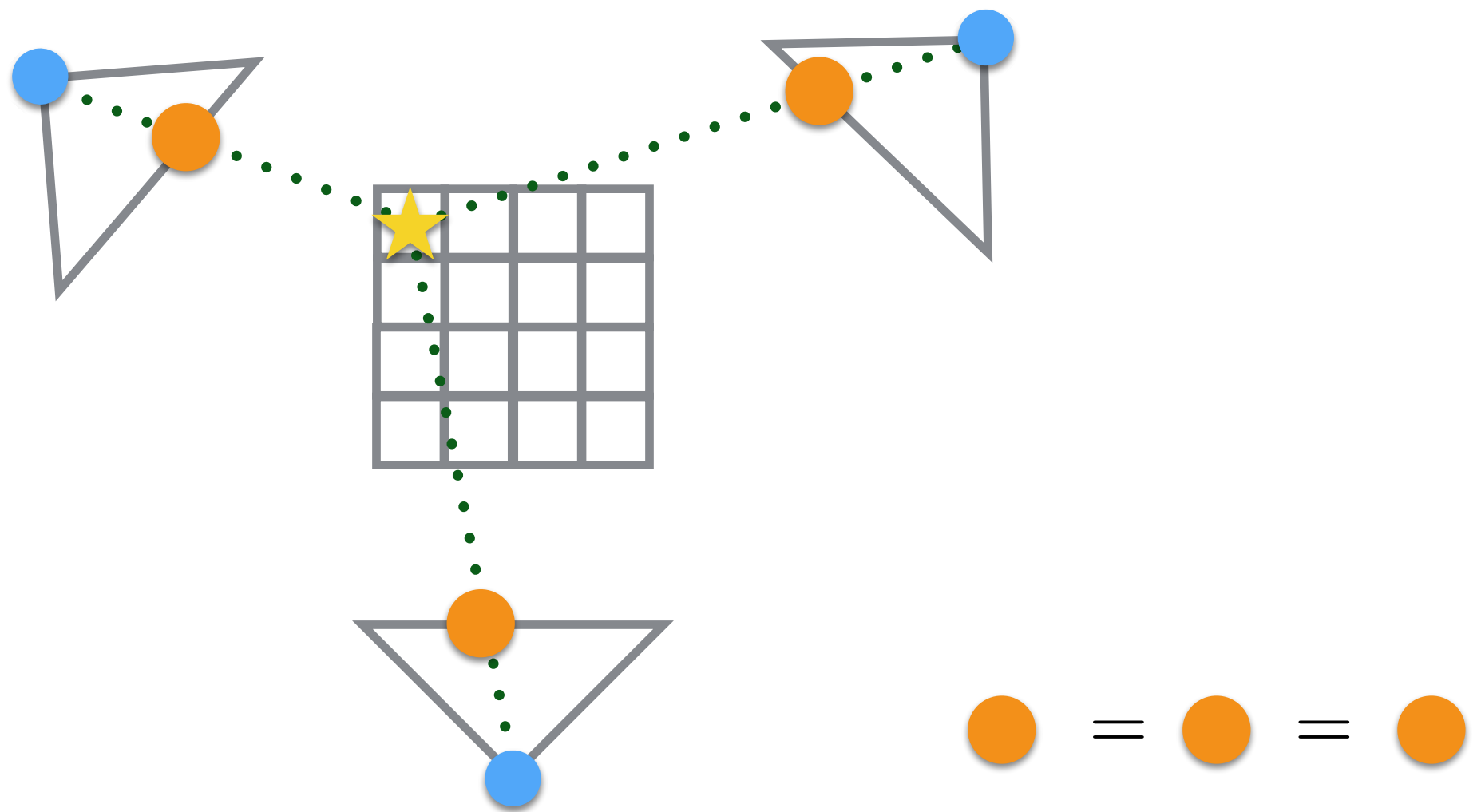
# Multi-View Stereo: Space Carving



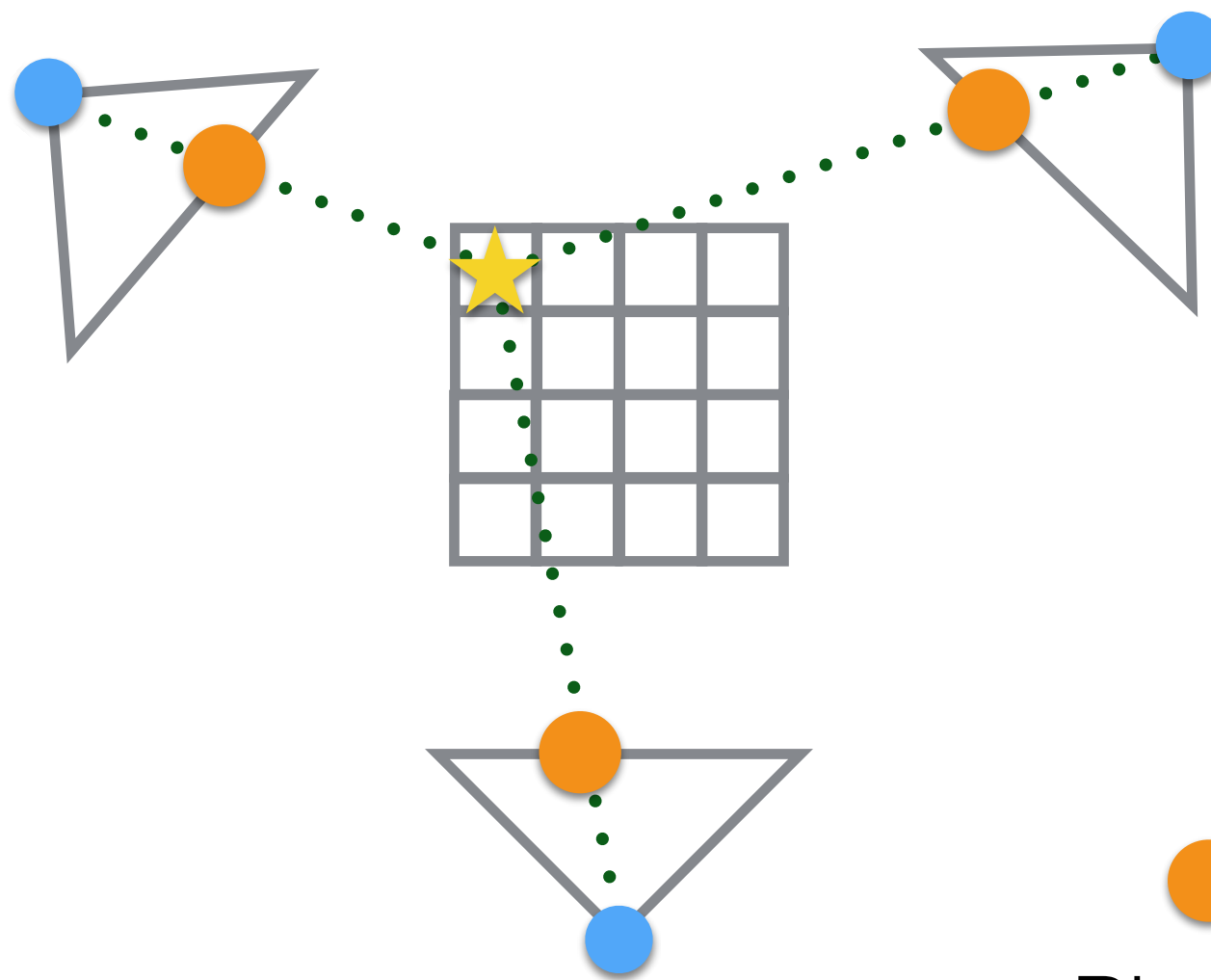
# Multi-View Stereo: Space Carving






# Multi-View Stereo: Space Carving



# Multi-View Stereo: Space Carving

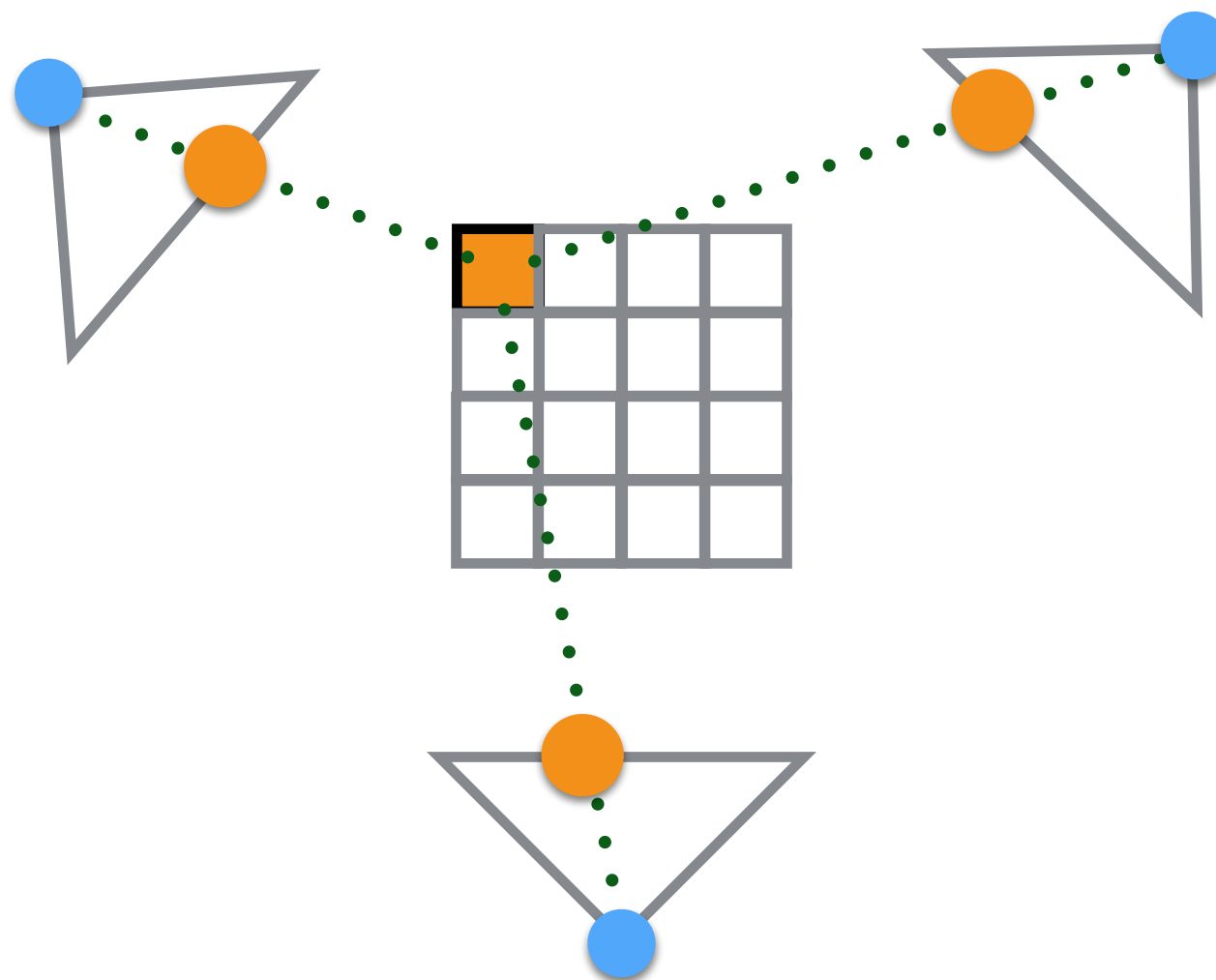


 =  =   
Photo-consistency!

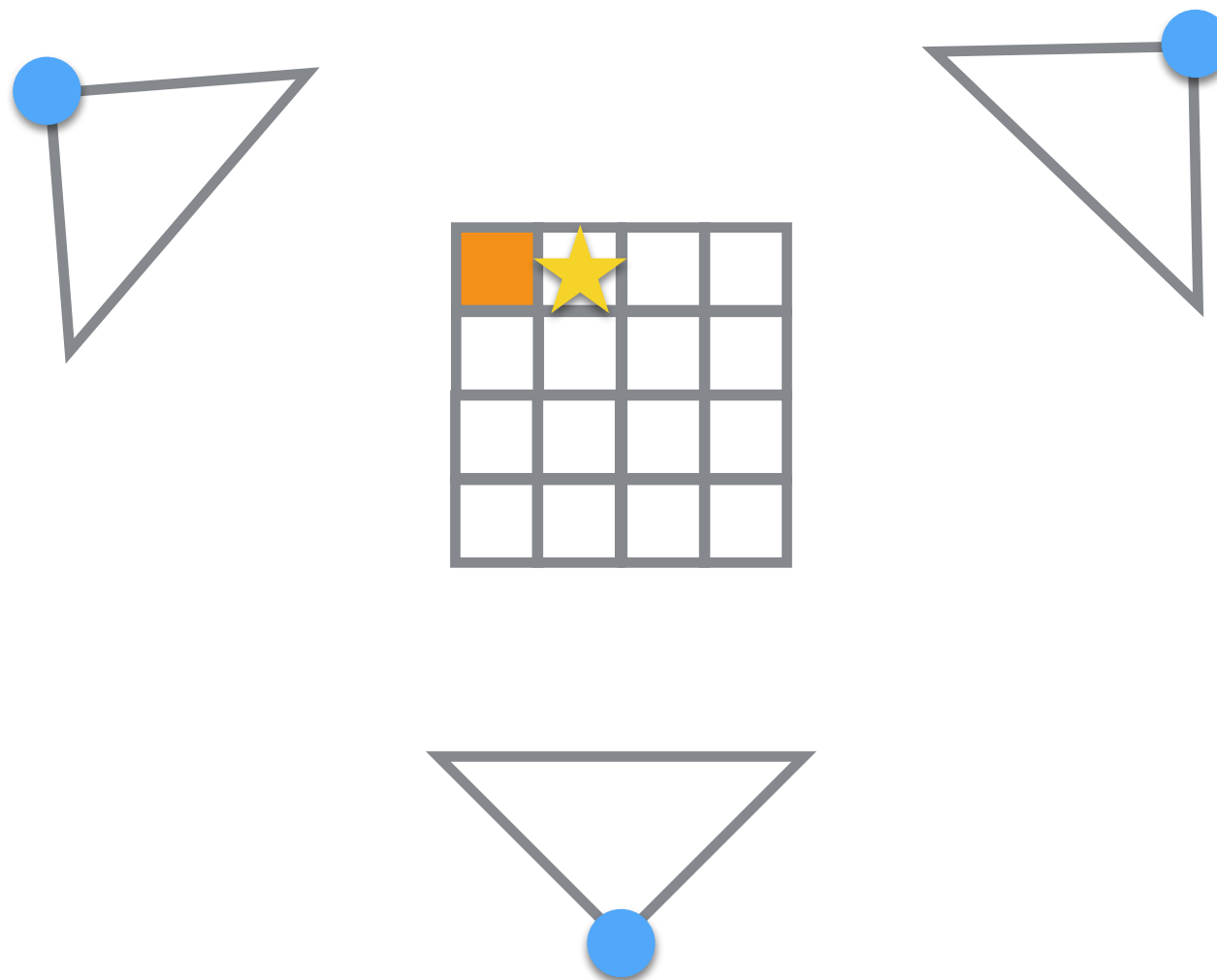


A voxel is ***photo consistent*** when its colour appears to be similar to all the cameras that “see” it.

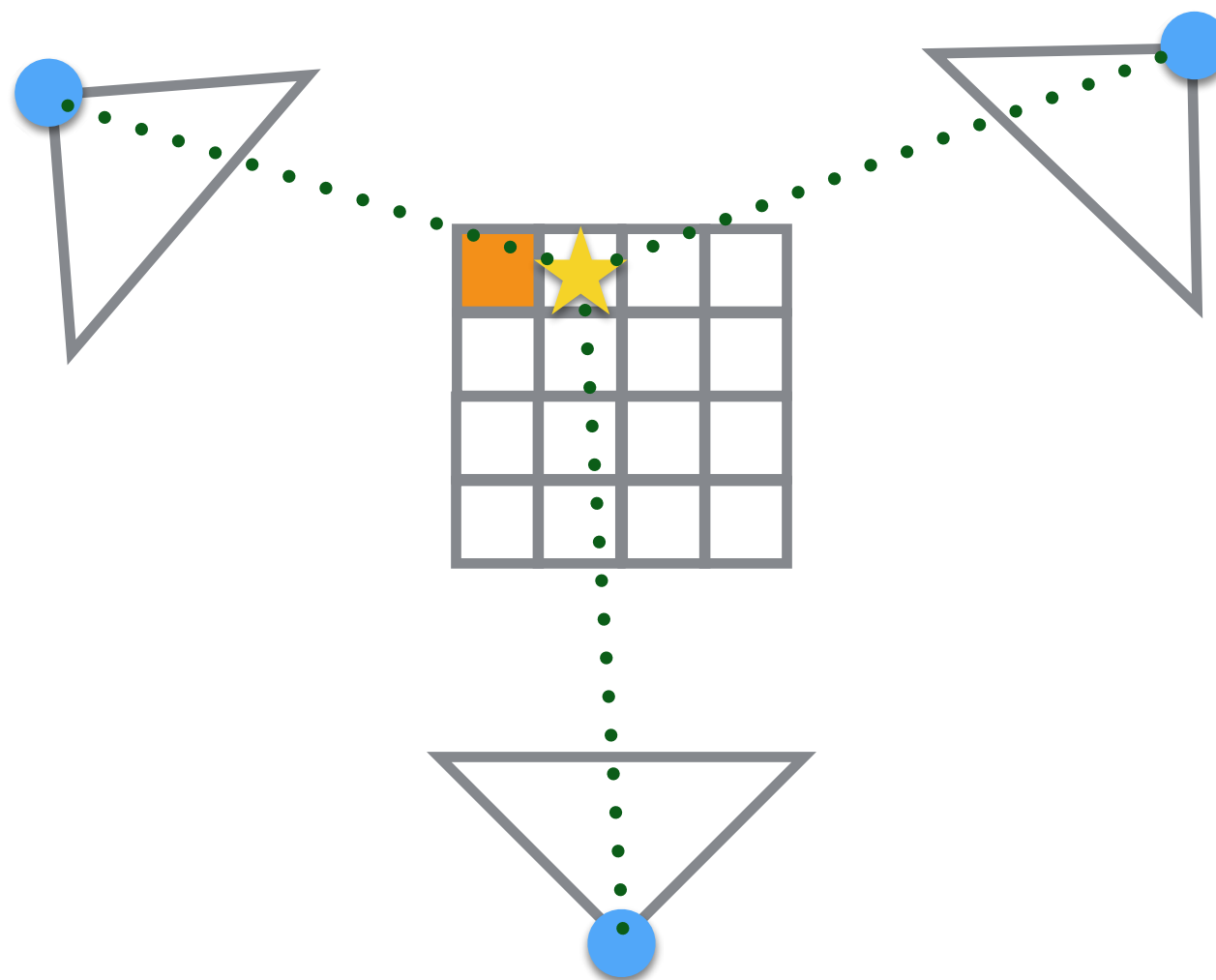
# Multi-View Stereo: Space Carving



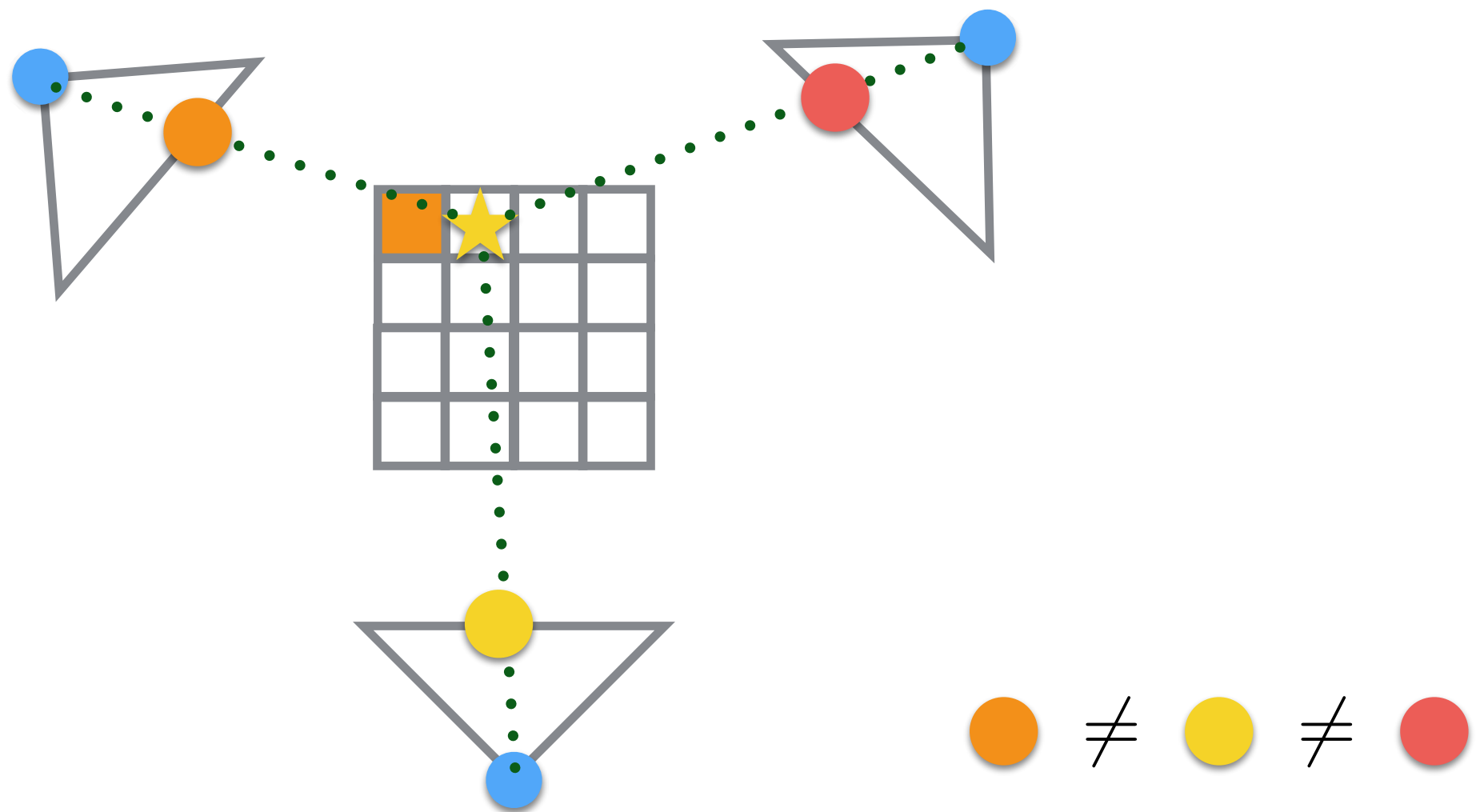
# Multi-View Stereo: Space Carving



# Multi-View Stereo: Space Carving

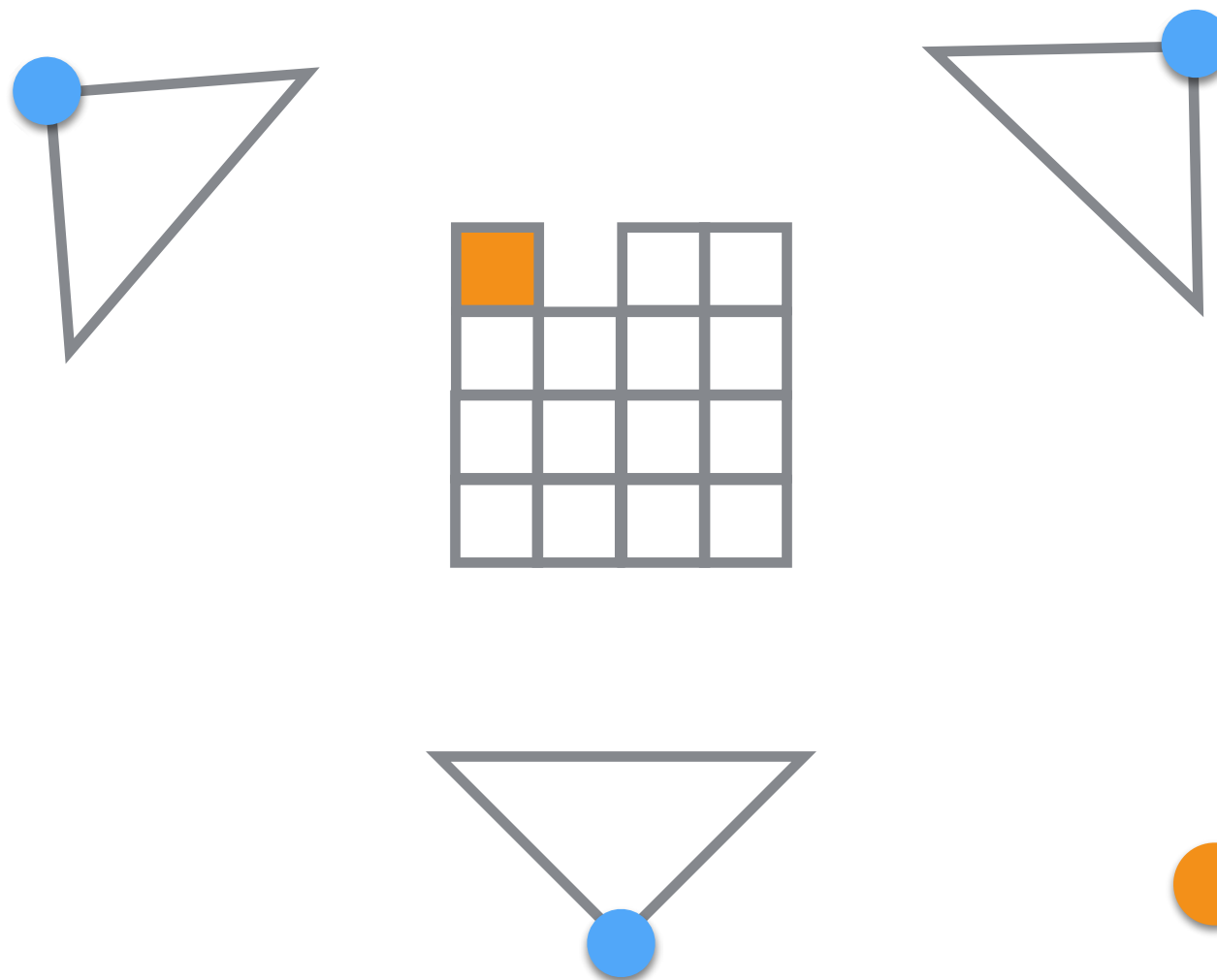


# Multi-View Stereo: Space Carving





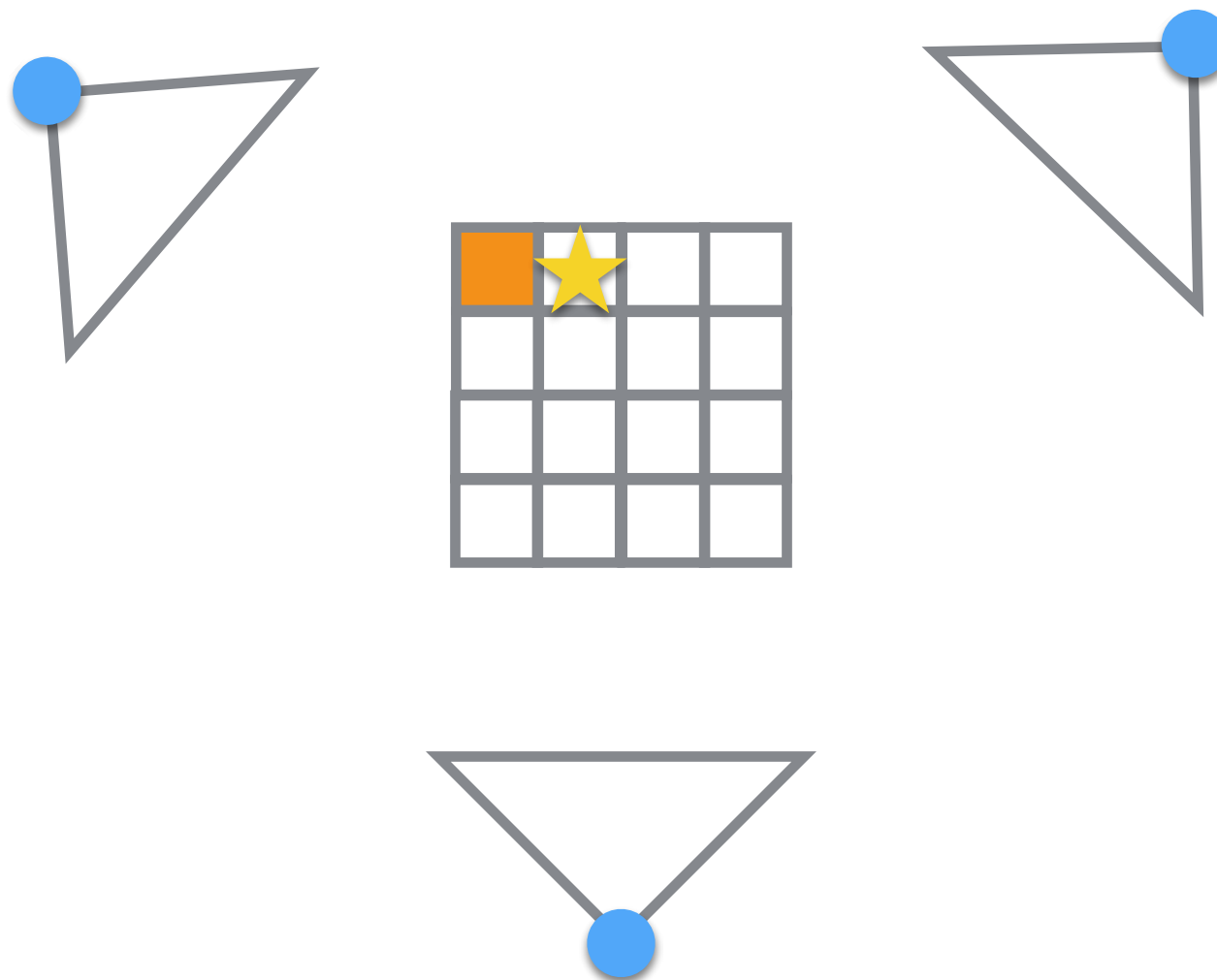
# Multi-View Stereo: Space Carving



●  $\neq$  ●  $\neq$  ●

The voxel is removed

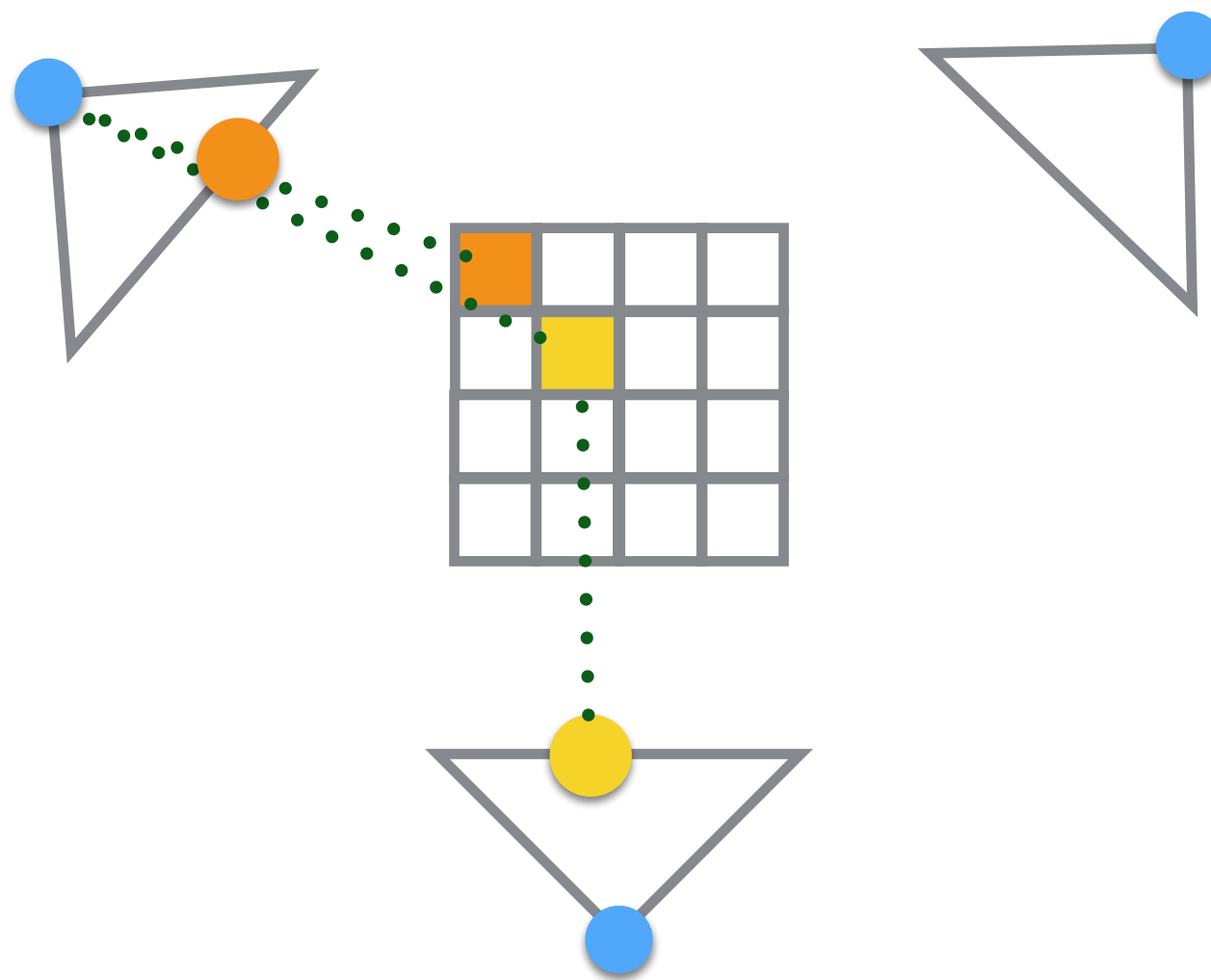
# Multi-View Stereo: Space Carving



Who sees the problem  
here?

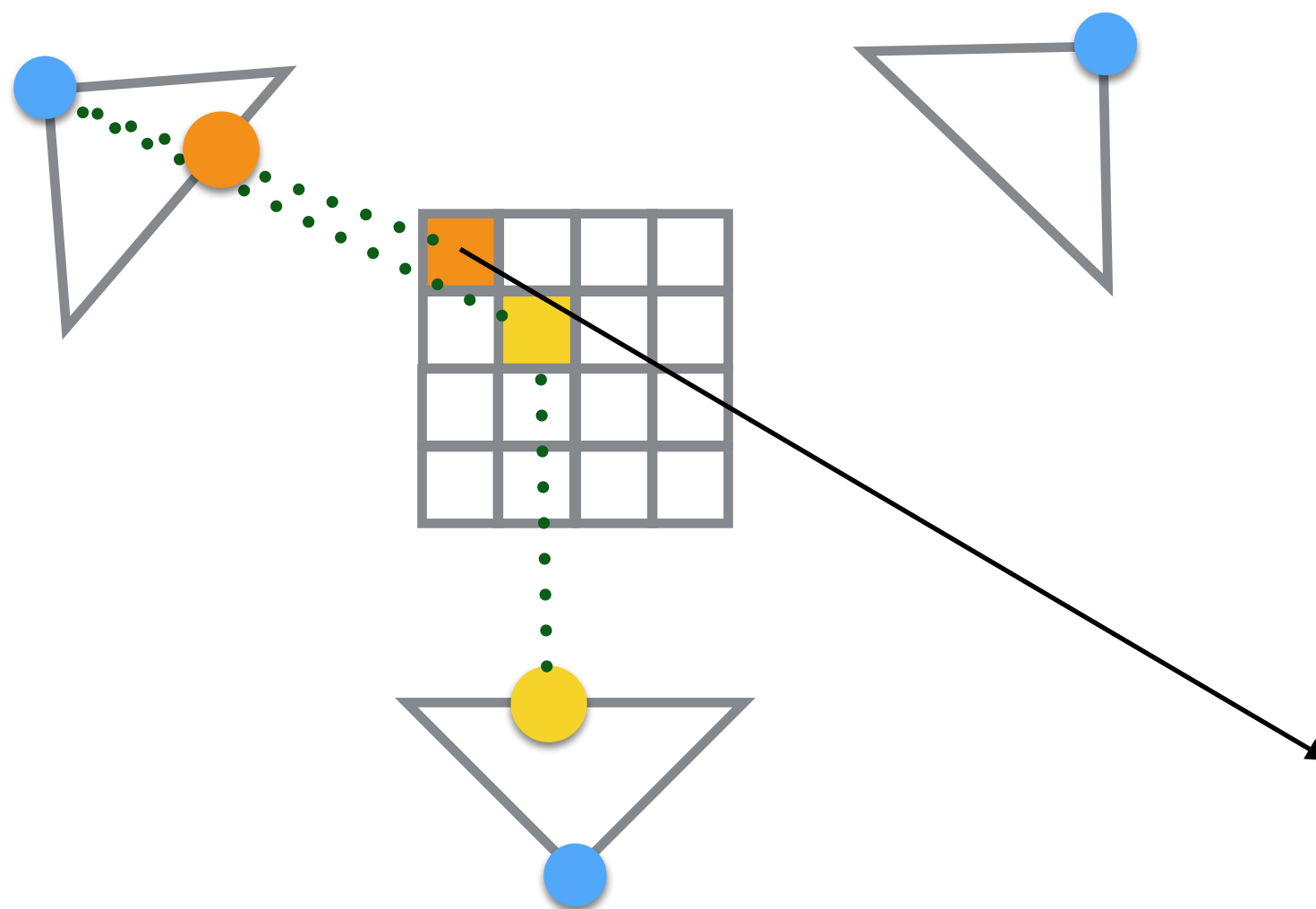
A voxel could be  
occluded, so it cannot be  
photo-consistent!

# Multi-View Stereo: Space Carving

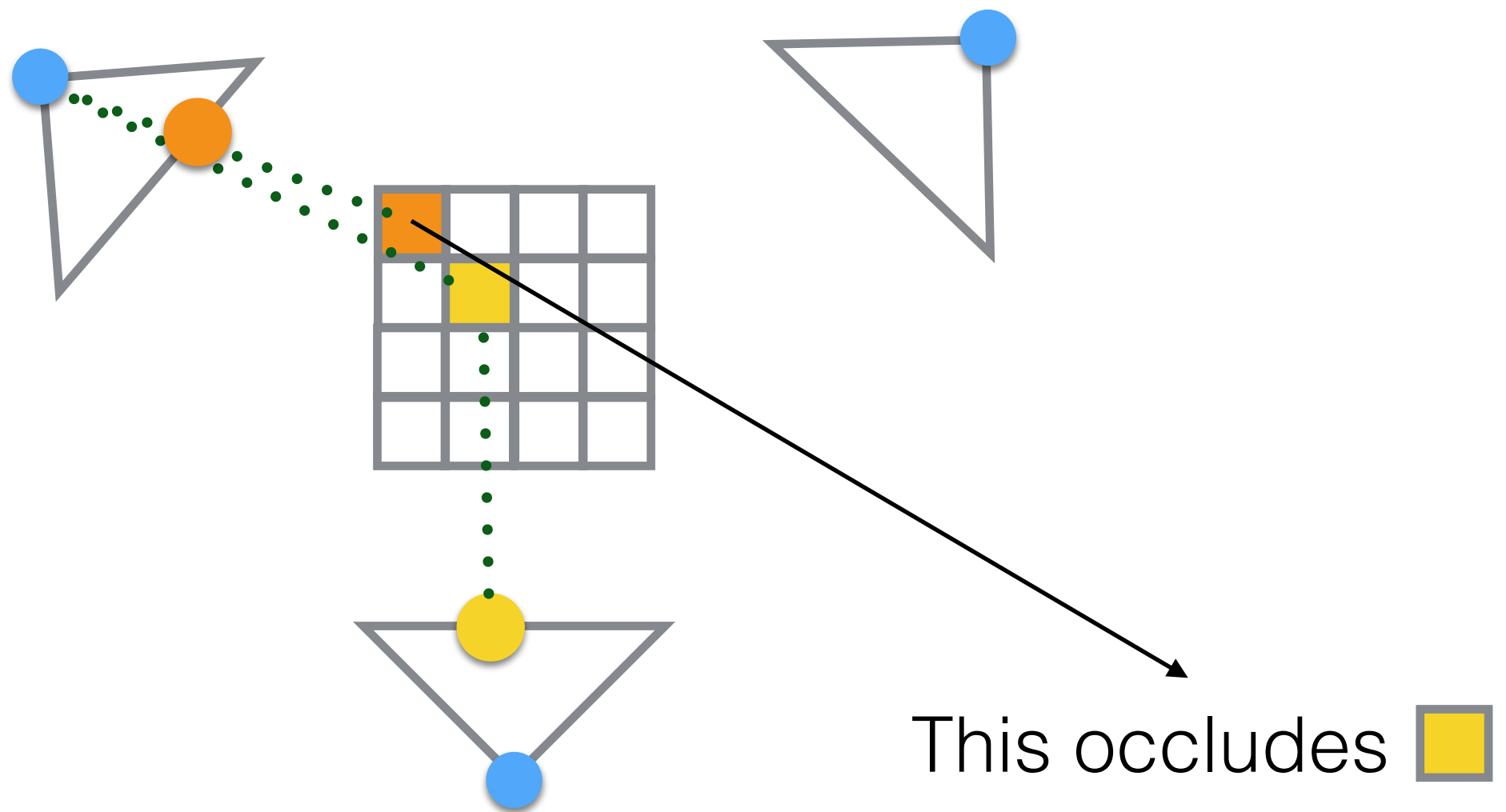




# Multi-View Stereo: Space Carving



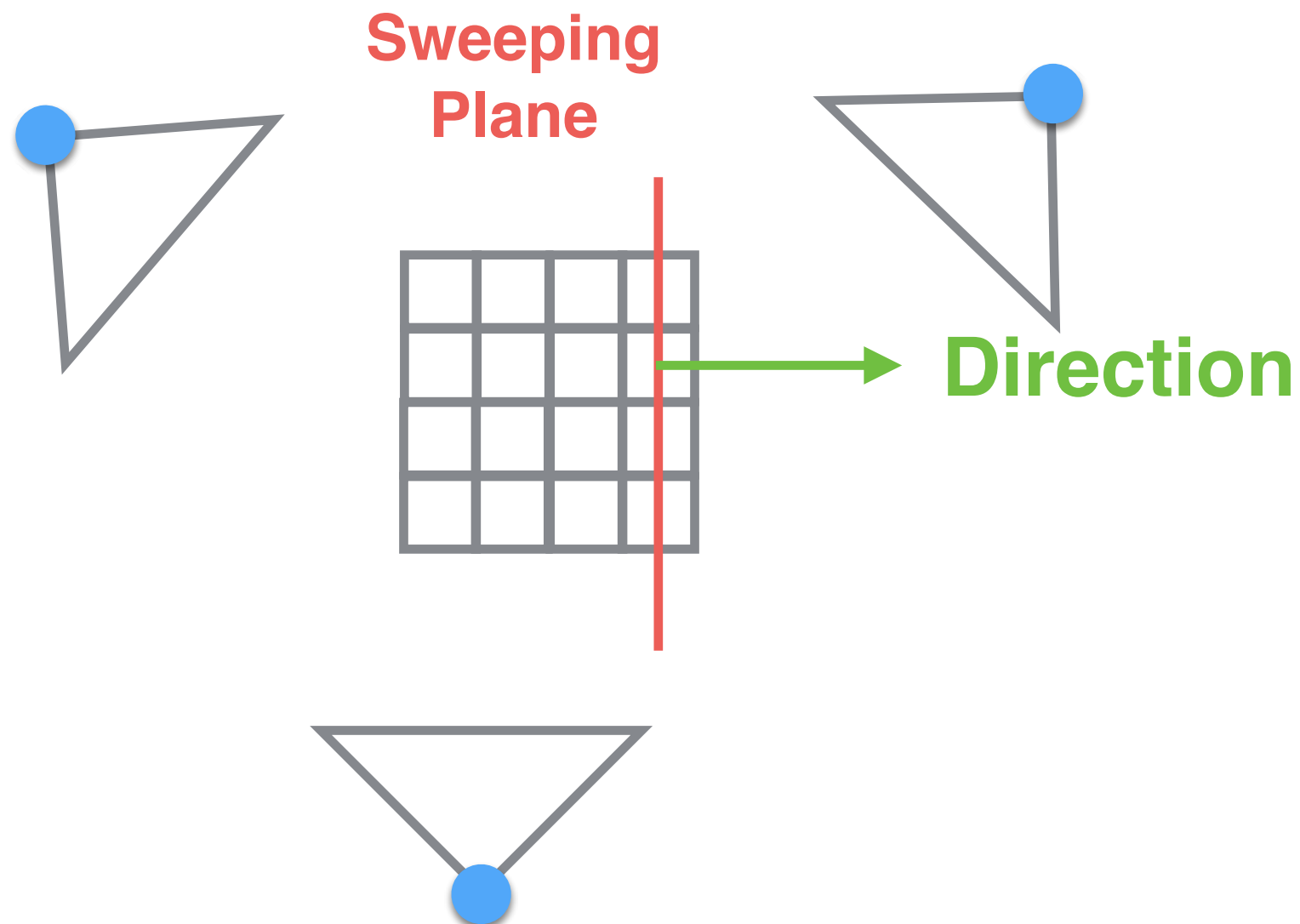
# Multi-View Stereo: Space Carving



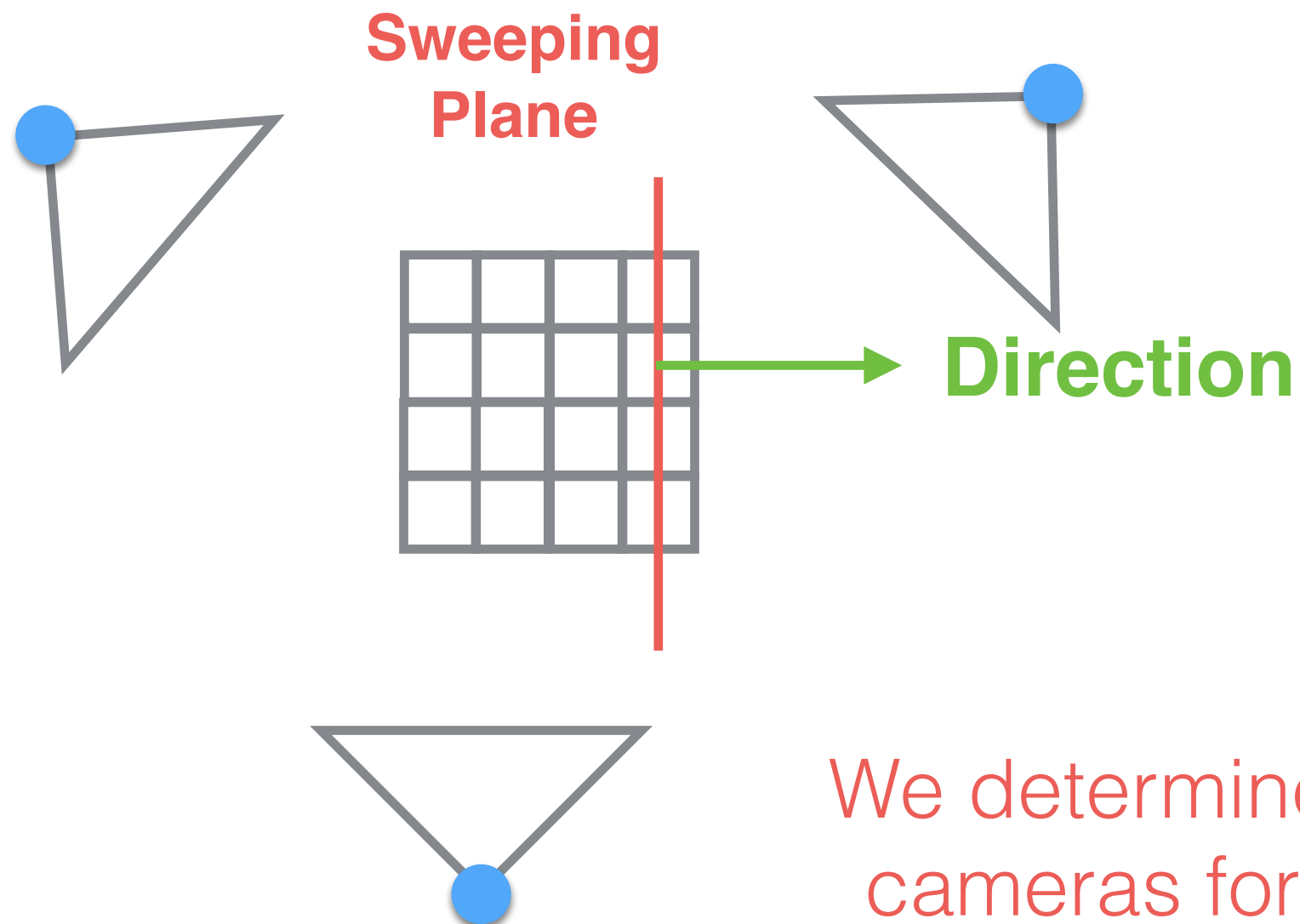
# Multi-View Stereo: Space Carving

- 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

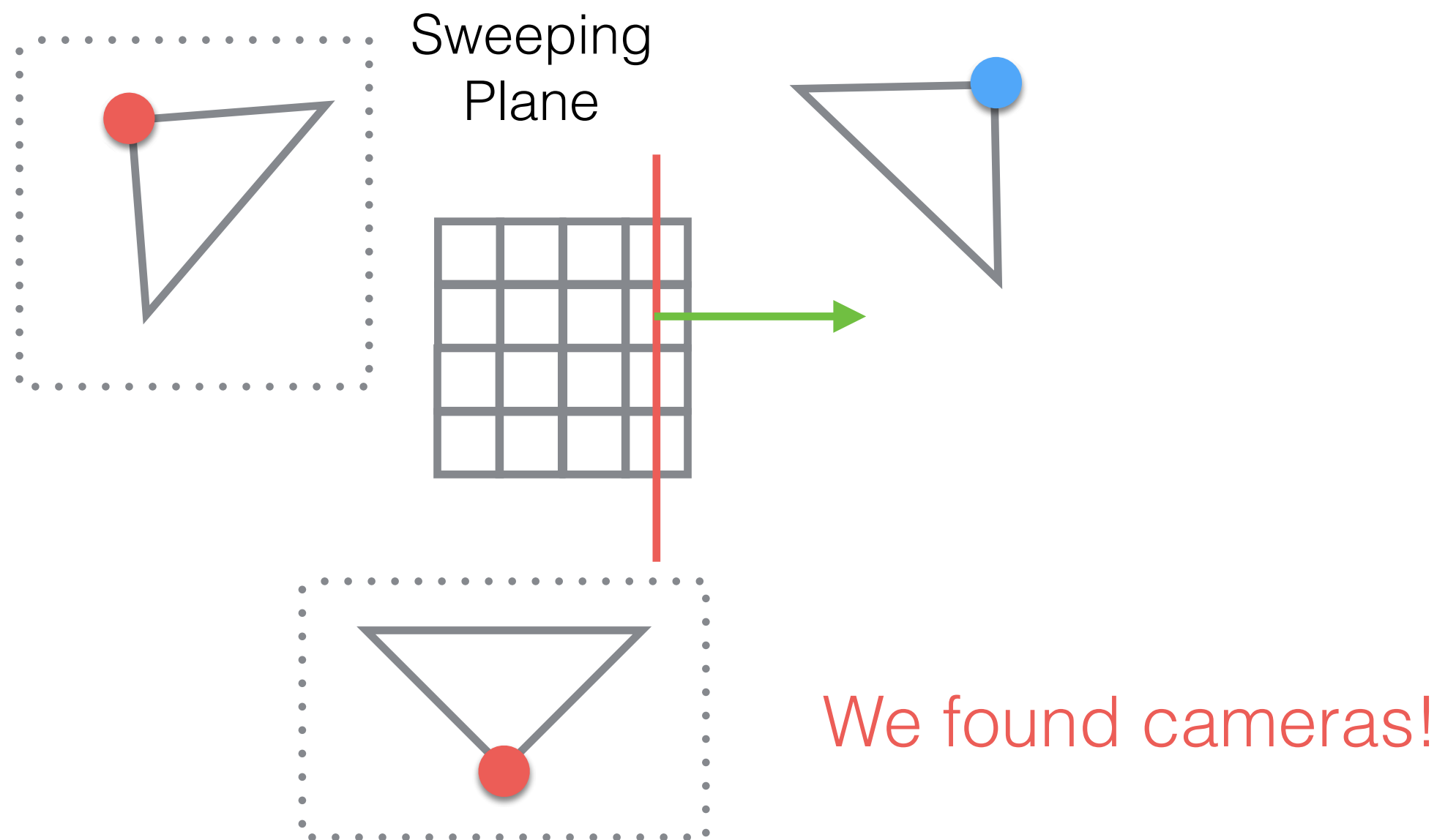


# Multi-View Stereo: Space Carving

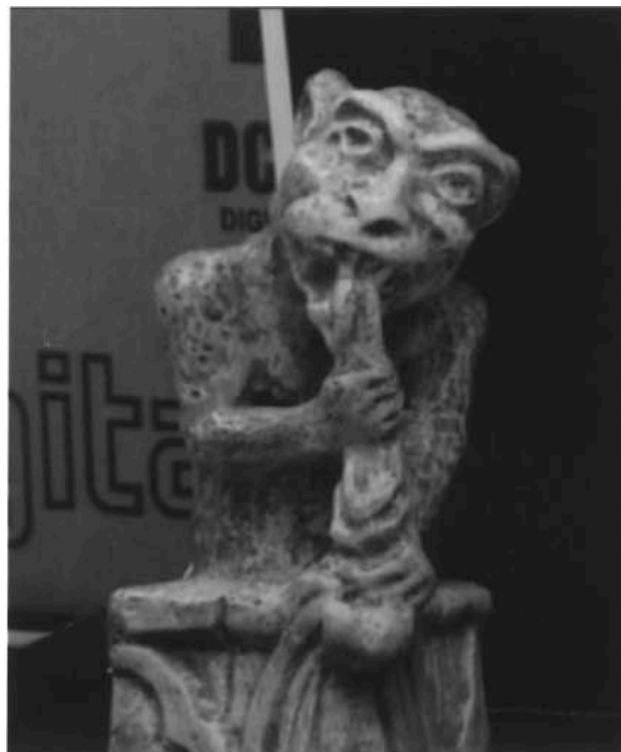




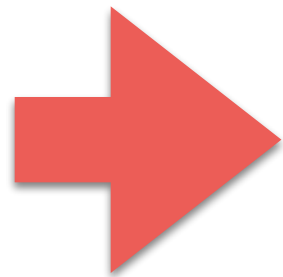
# Multi-View Stereo: Space Carving



# Multi-View Stereo: Space Carving Example



Photograph

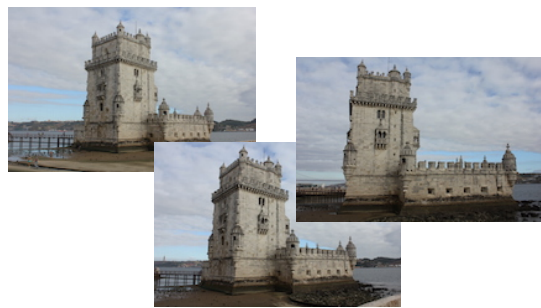


3D Model

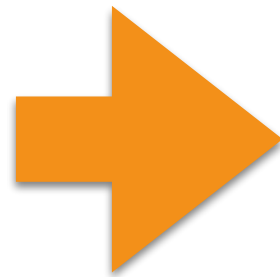
# Multi-View Stereo: Space Carving Example

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

# 3D from Photographs



Photographs



Automatic  
Matching of  
Images



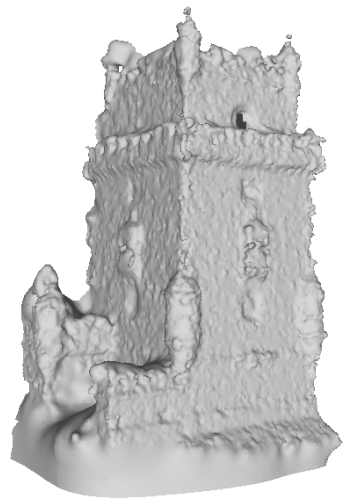
Camera  
Calibration



Dense  
Matching

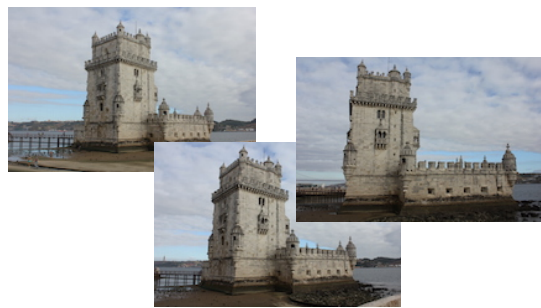


Surface  
Reconstruction

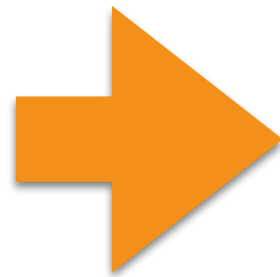


3D model

# 3D from Photographs



Photographs



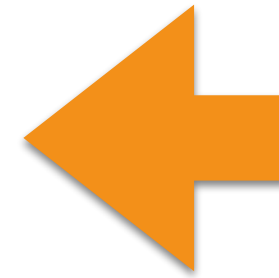
Automatic  
Matching of  
Images



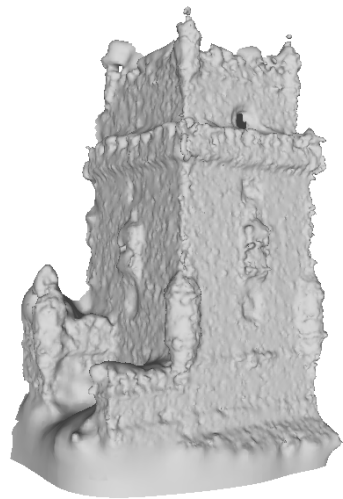
Camera  
Calibration



Dense  
Matching



Surface  
Reconstruction



3D model

# 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!

# Multi-View Stereo: Propagation

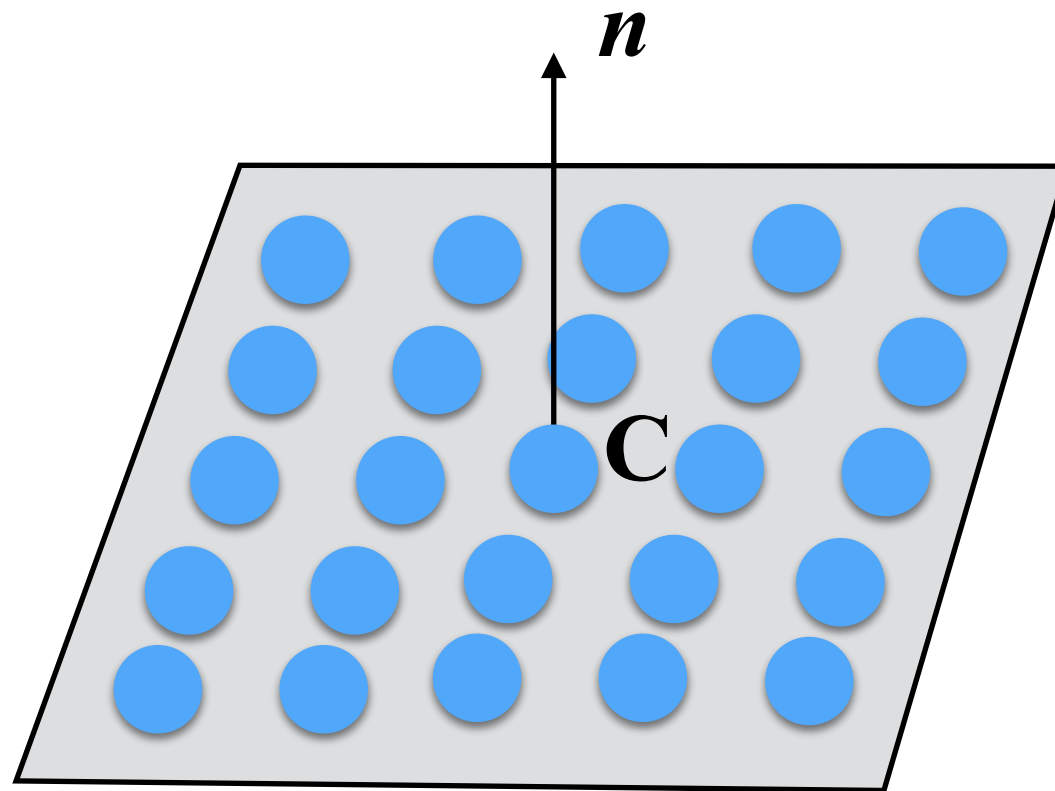
- The idea is to create a 3D plane called “*patch*” for each feature point (that is an inlier) for which we have a triangulation.
- We use only feature points that satisfies:

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

For all views where they appear.

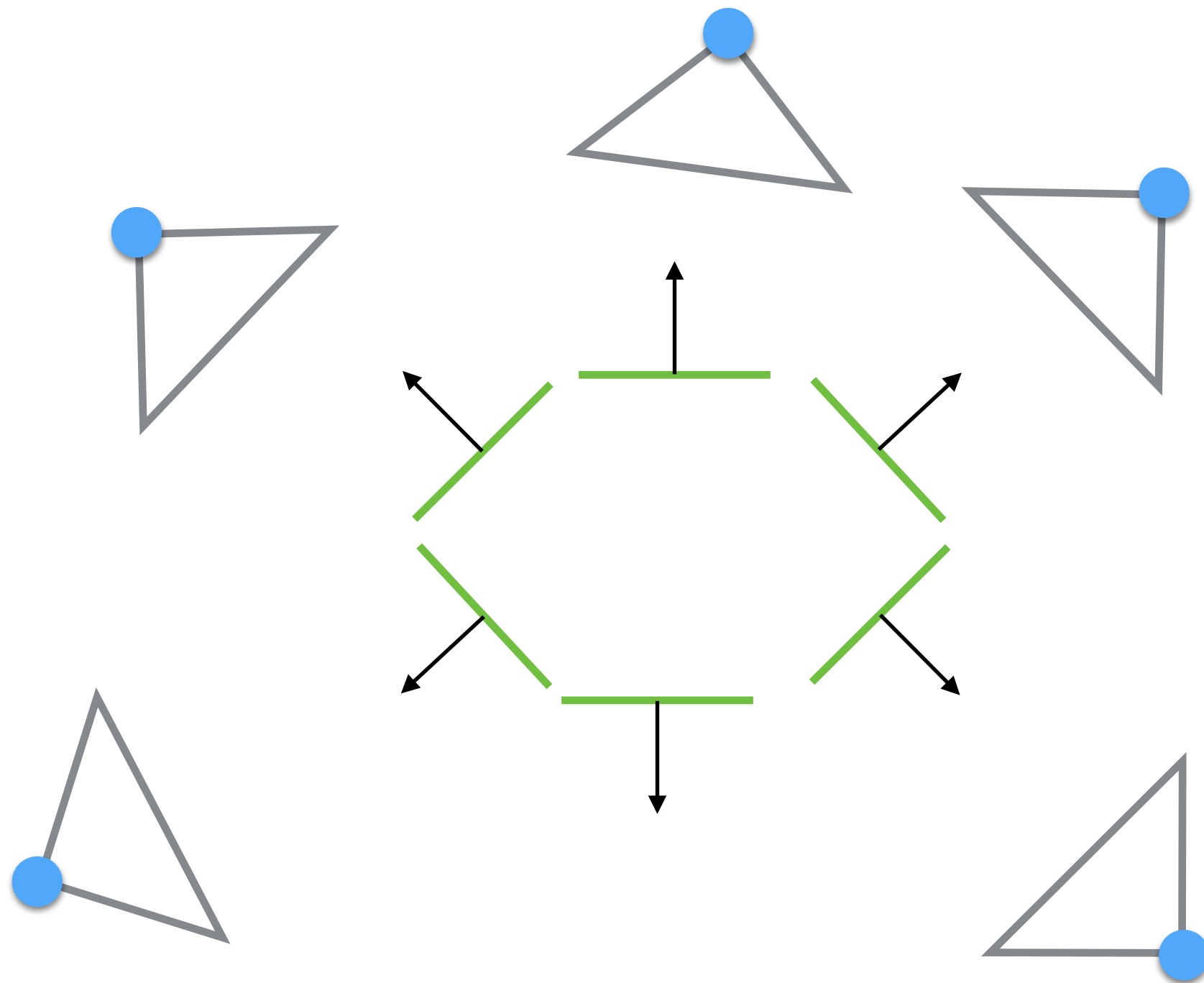
- The patch has its size expressed in pixels (5x5 or 7x7), and it takes color value from the projection of it on the best matched image *T*.

# Multi-View Stereo: Patch



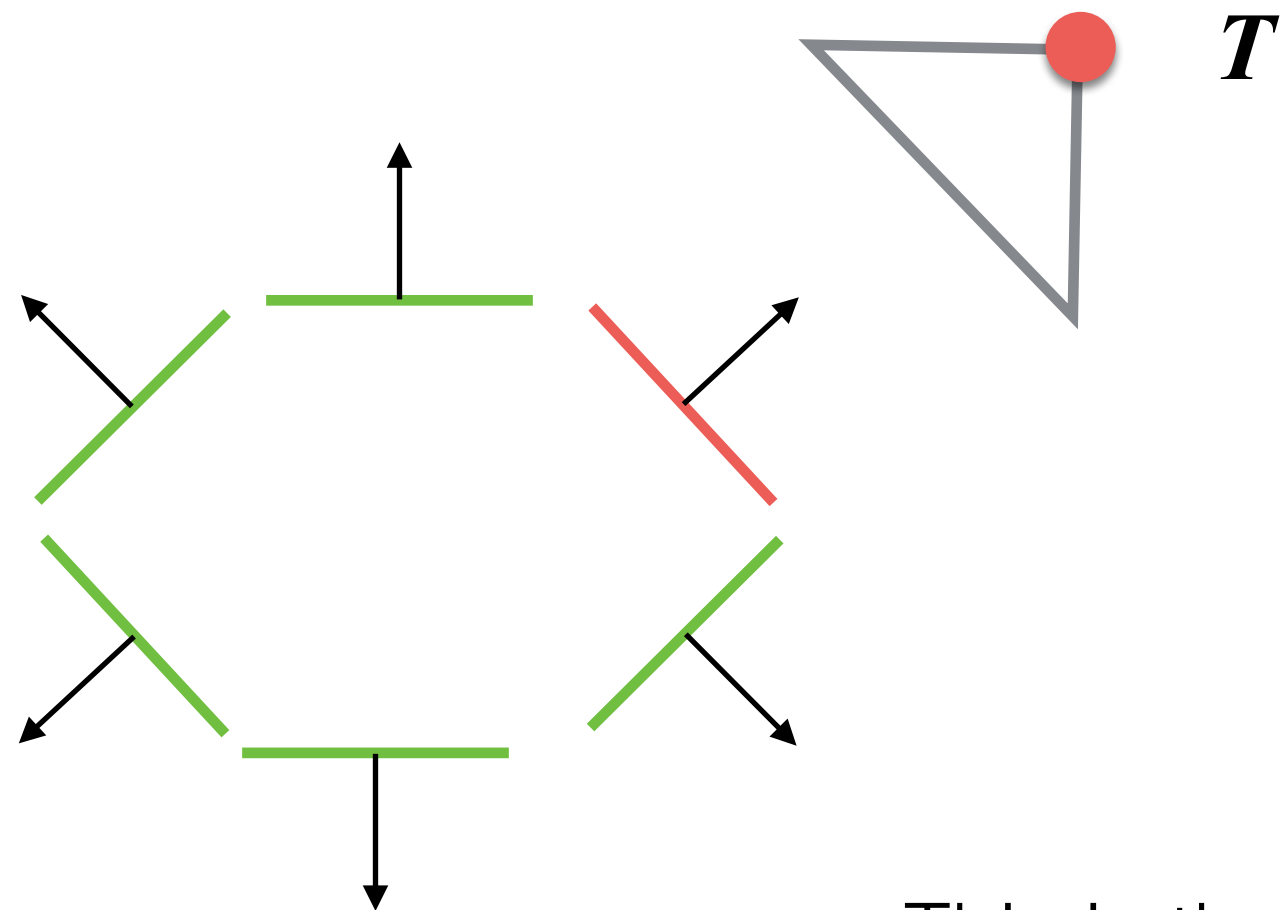
A 5x5 patch

# Multi-View Stereo: Finding the Camera for Projecting Colors to the Patch





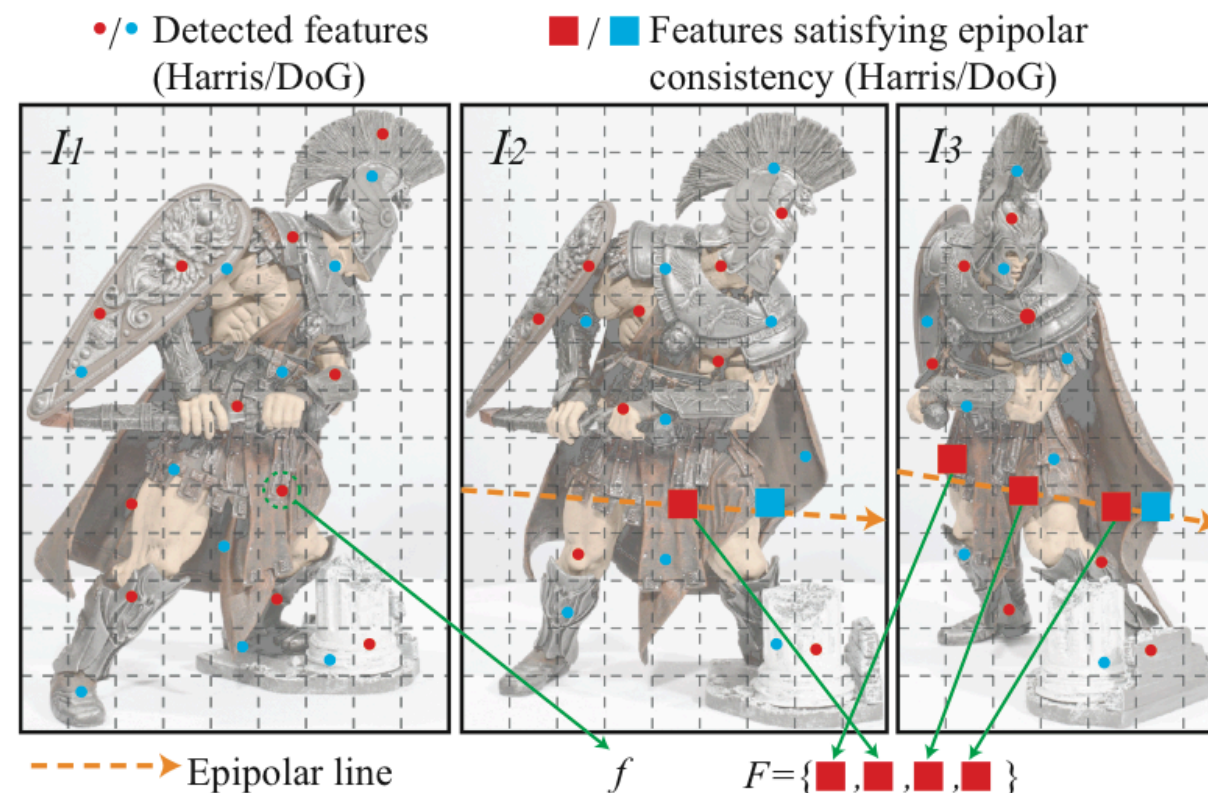
# Multi-View Stereo: Finding the Camera for Projecting Colors to the Patch



This is the closest camera (better angle)

# Multi-View Stereo: Propagation

- Input images are subdivided into cells, and patches are iteratively expanded to nearby location that are empty.
- We use visibility constraints such as epipolar lines, and photo-consistency, to filter out outliers.



# Multi-View Stereo: Propagation Example



Yasutaka Furukawa and Jean Ponce,  
[Accurate, Dense, and Robust Multi-View Stereopsis](#), CVPR 2007.