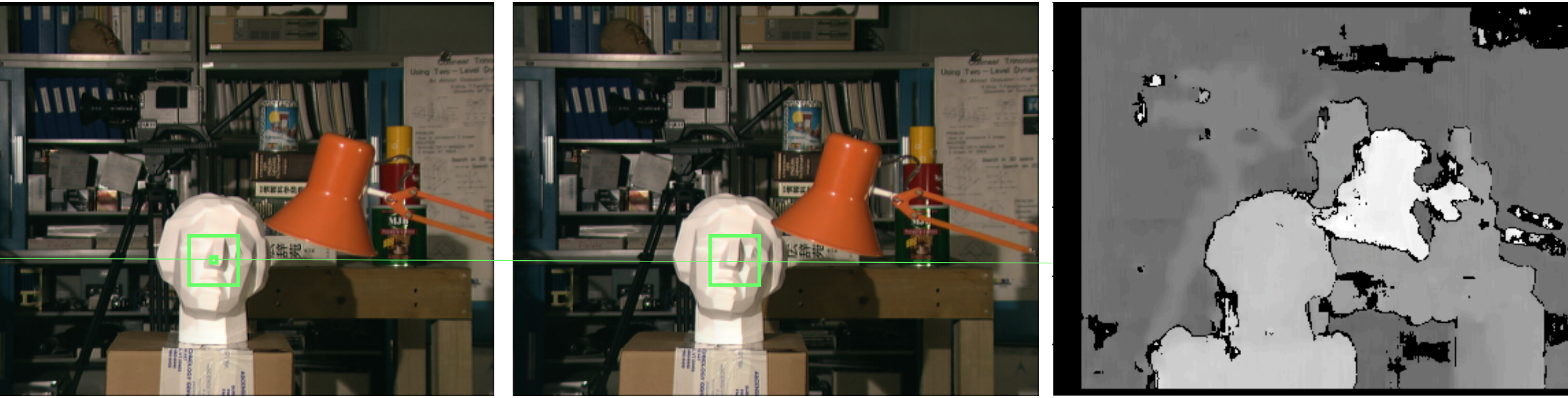
Visual Computing I:

Interactive Computer Graphics and Vision



Stereo and Dense Depth

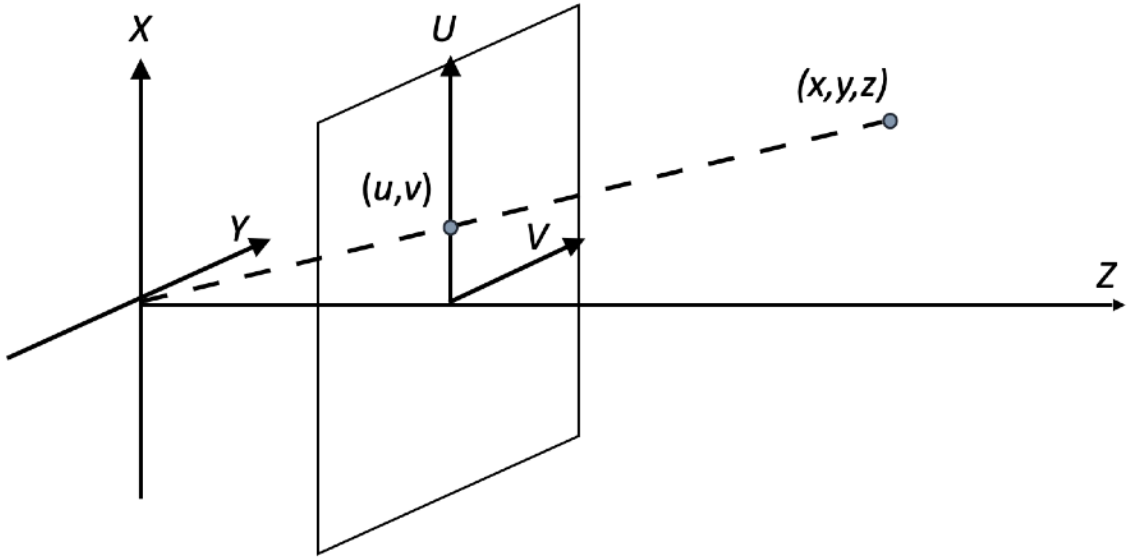
Stefanie Zollmann and Tobias Langlotz

Last time..

Camera Calibration

$$\mathbf{u} \equiv \mathrm{K}[\mathrm{R} \quad \mathbf{t}]\mathbf{x}$$

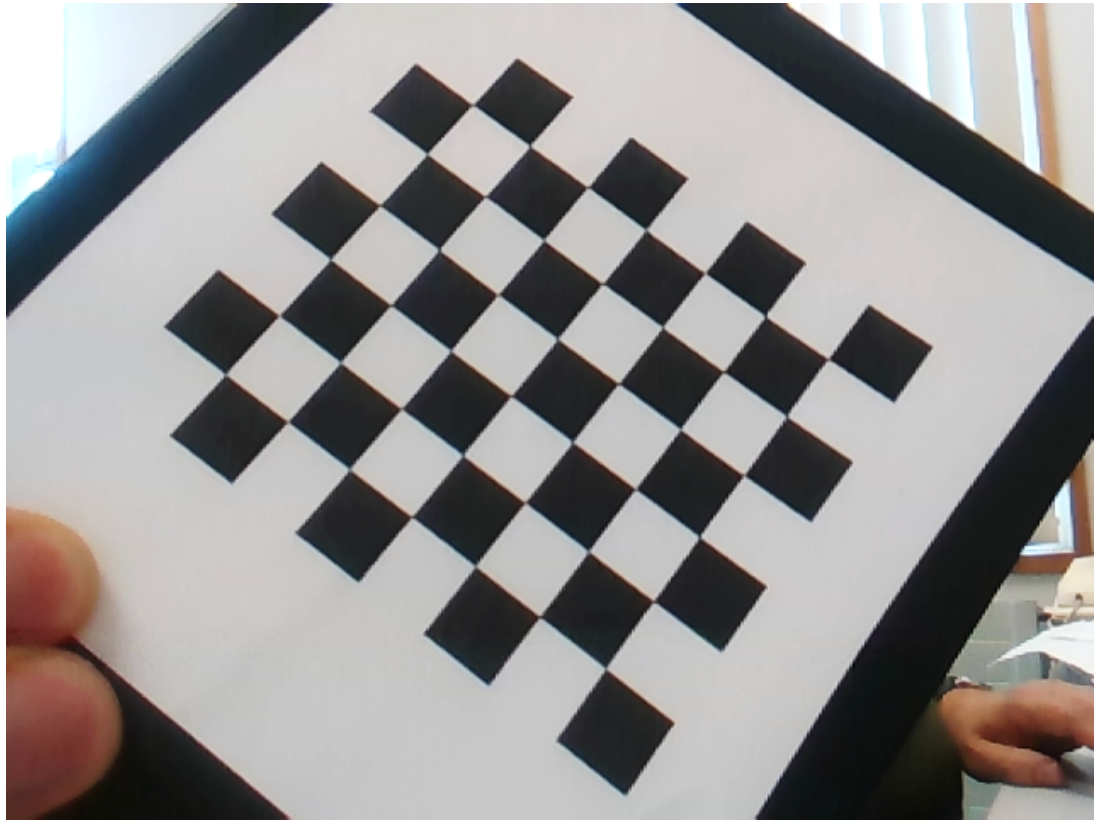
- $\mathbf{u} = [u \quad v \quad 1]^T$ is a 2D image point
- K is a 3×3 calibration matrix
- R is a 3×3 rotation matrix
- $\mathbf{t}$ is a 3D translation vector
- $\mathbf{x} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T$ is a 3D point

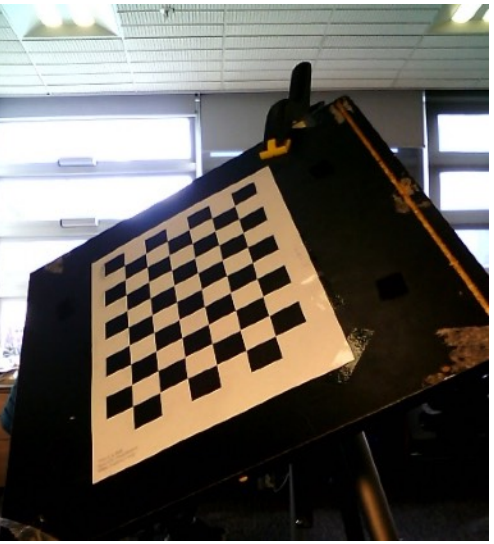


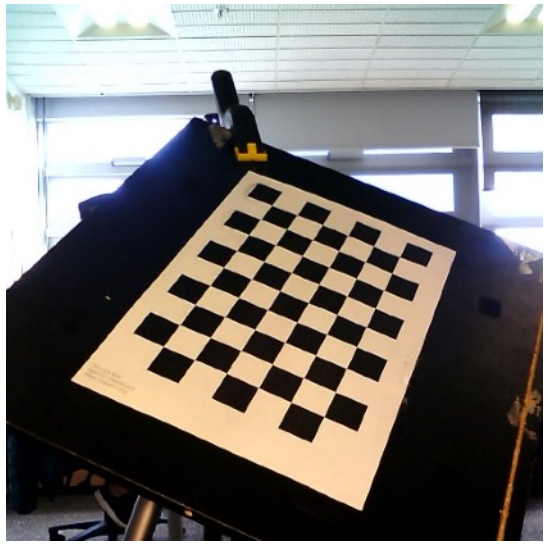
- Camera calibration: determine K

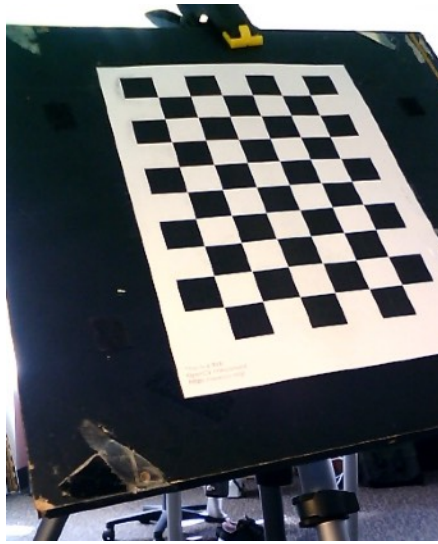
Calibration in OpenCV

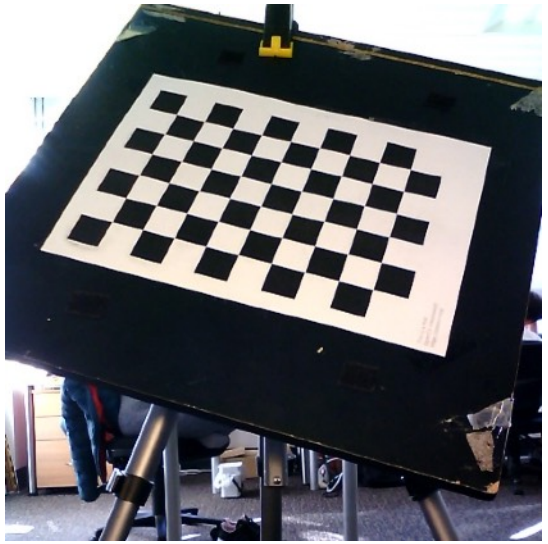
- Calibration targets:
  - Input is 3D-2D matches
  - Want easy-to find 2D points
  - Need known 3D co-ordinates
- Planar targets common
  - Easy to make with a printer
  - Chess/Checkerboards
  - Grids of dots or lines
- Is a 2D pattern enough?

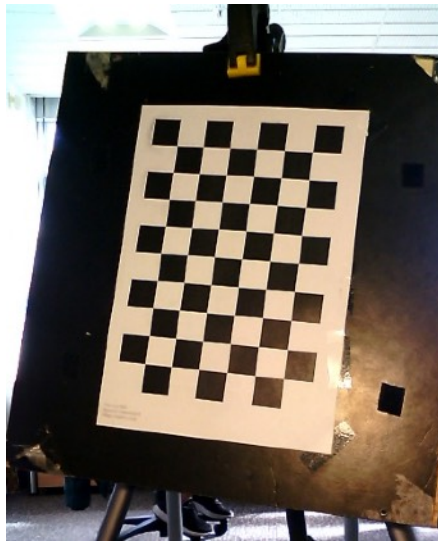










How does a calibration look like?

Do we need to calibrate our cameras for every app?

Accessing Intrinsics using ARCore

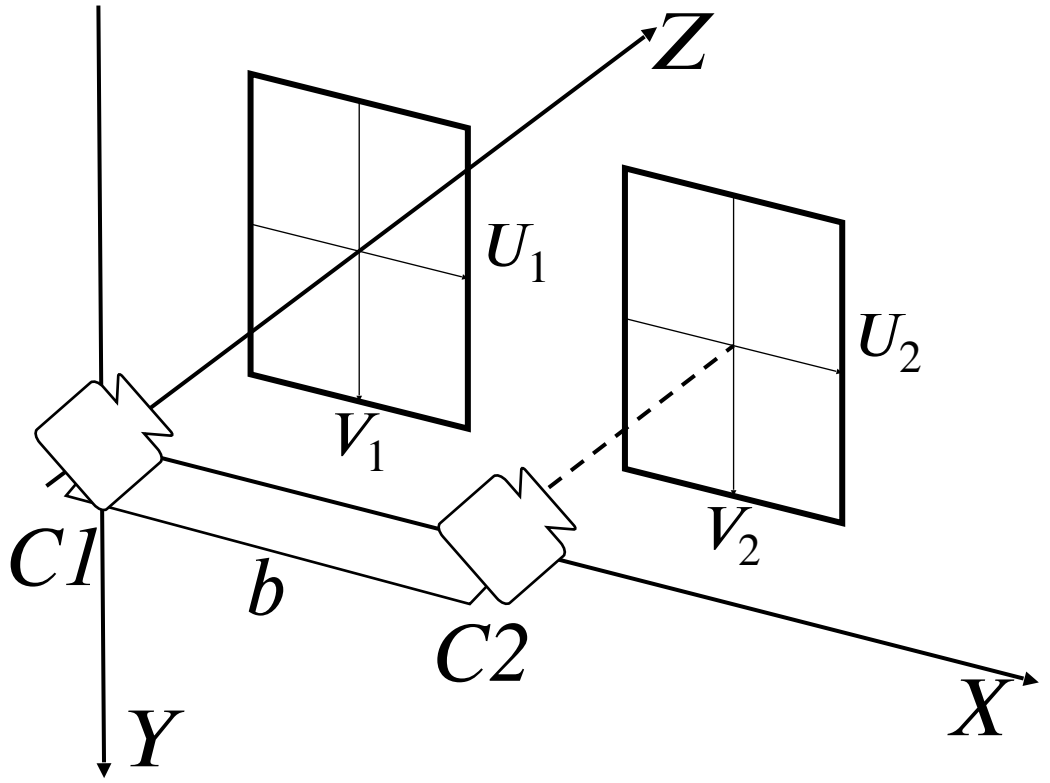
```
void Update()
    {
        // 1. Try to get the camera intrinsics
        if (cameraManager.TryGetIntrinsics(out ARCameraIntrinsics intrinsics))
        {
            // 2. The 'intrinsics' struct now holds the calibration data

            // Access focal length (fx, fy)
            Vector2 focalLength = intrinsics.focalLength;

            // Access principal point (cx, cy)
            Vector2 principalPoint = intrinsics.principalPoint;

            // Access the image resolution this corresponds to
            Vector2Int resolution = intrinsics.resolution;

            // You now have all the components of the K matrix
            // print($"Focal Length: {focalLength}, Principal Point: {principalPoint}");

            // You can construct the K matrix yourself if needed:
            // Matrix4x4 K = Matrix4x4.identity;
            // K[0, 0] = focalLength.x;
            // K[1, 1] = focalLength.y;
            // K[0, 2] = principalPoint.x;
            // K[1, 2] = principalPoint.y;
        }
    }
```

Simple Stereo

A Simple Stereo System

- Assume a set of 2 pinhole cameras
  $\mathbf{u} = \mathrm{K}[\mathrm{R} \quad \mathbf{t}]\mathbf{x}$
  - Simplify for now: $\mathrm{K} = \mathrm{I}$
- Camera 1 (C1): $\mathrm{R} = \mathrm{I}, \mathbf{t} = \mathbf{0}$
  - No rotation, looks along Z
- Camera 2 (C2): shifted along X

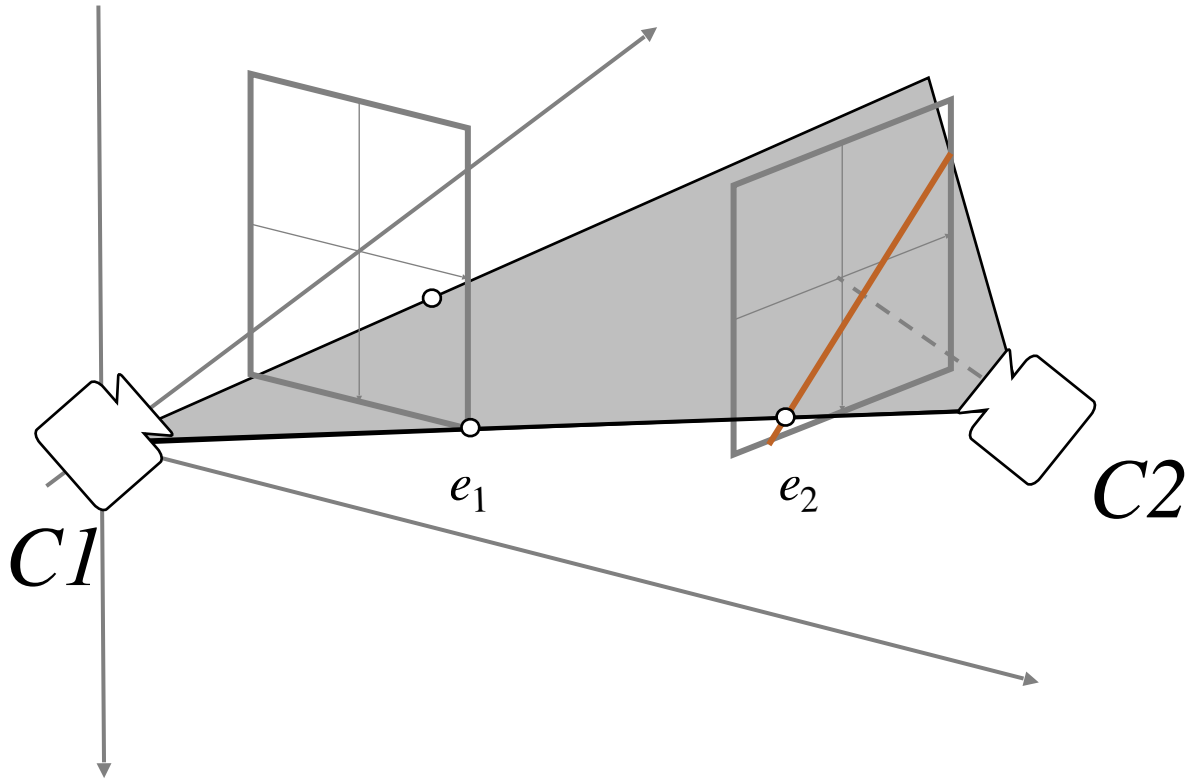
$$\mathrm{R} = \mathrm{I}, \; t = \begin{bmatrix} b \\ 0 \\ 0 \end{bmatrix}$$



More General Stereo

Epipoles and Epipolar line

- Draw a line between the camera centres
  - This crosses the image planes at the epipoles (e_1, e_2)
  - Make a plane through a point in one image and the two camera centres: epipolar plane
  - This intersects the second image at the epipolar line



Fundamental Matrix

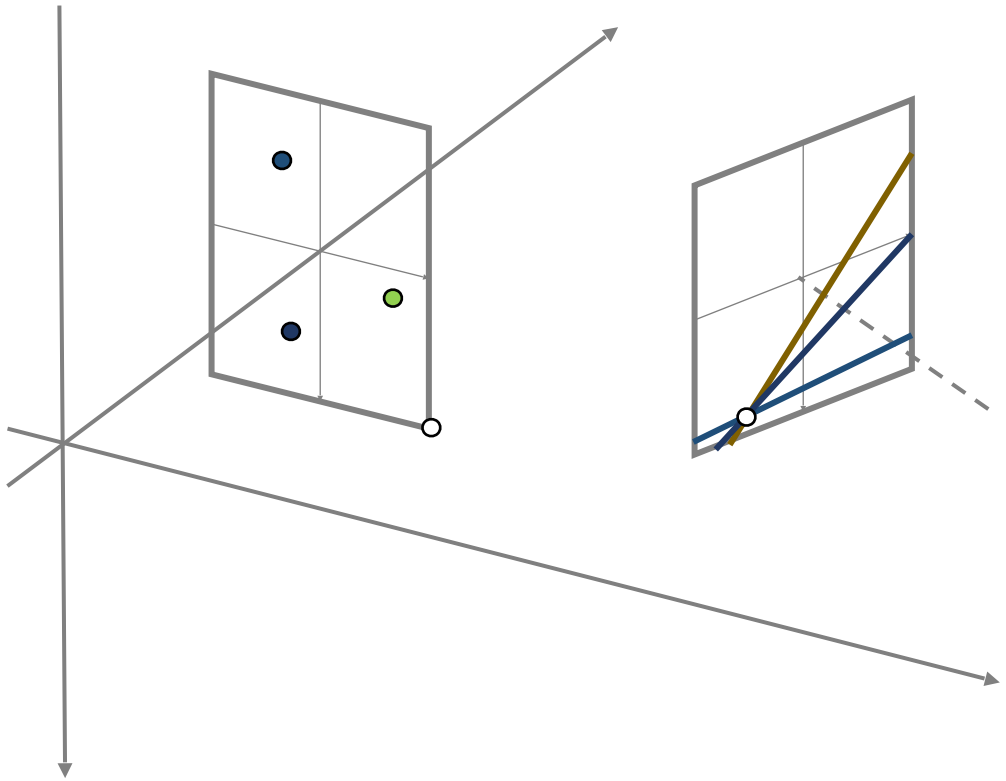
Epipolar Geometry and Fundamental Matrix

- The fundamental matrix represents this geometry
  - This is a 3×3 matrix, F
    - If we have point matches

$$\mathbf{p}_1 = \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} \leftrightarrow \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = \mathbf{p}_2$$

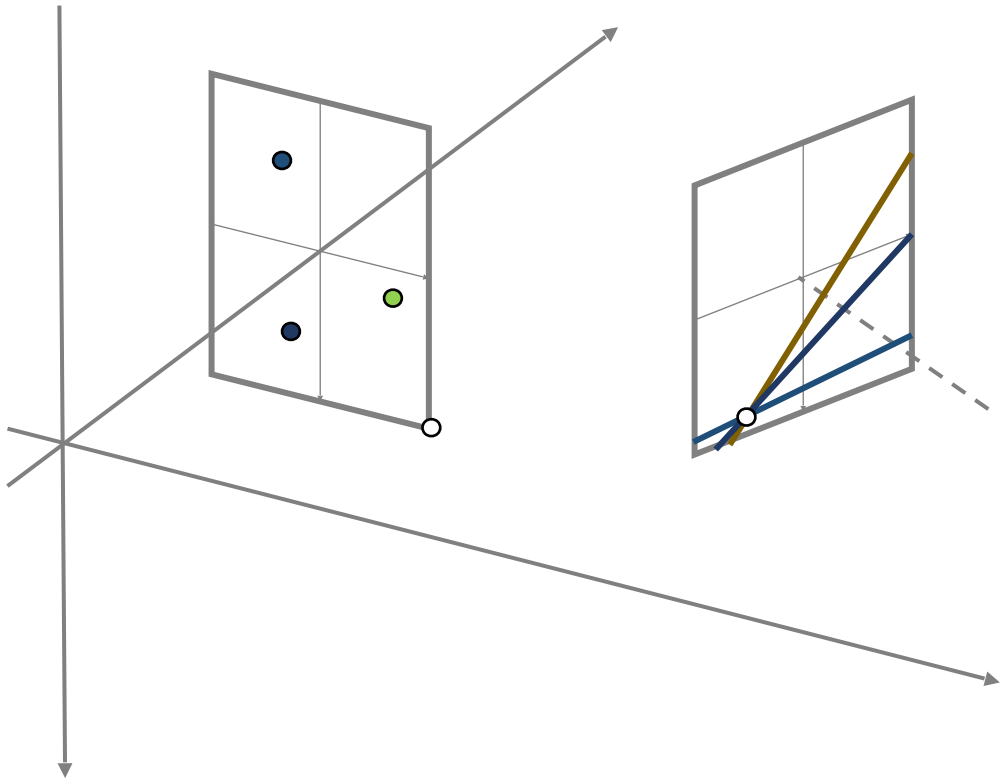
    - We can find the corresponding epipolar line in the other image

$$l_2 = \mathrm{F} p_1$$
$$l_1 = \mathrm{F}^T p_2$$



The Fundamental Matrix

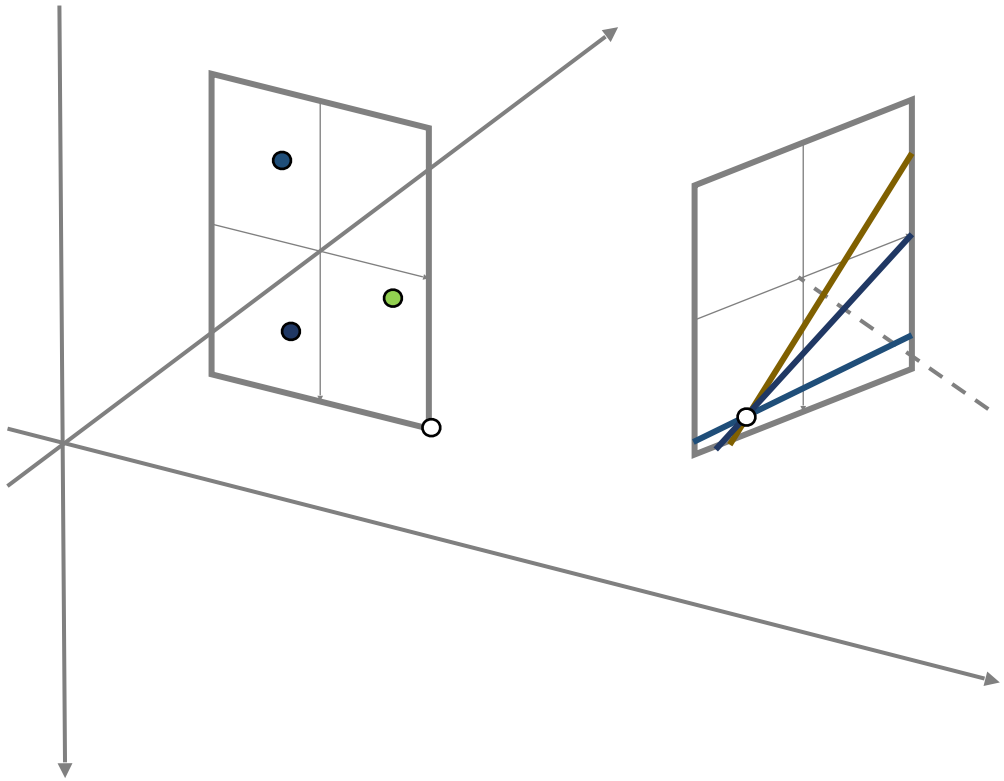
- In general: a point, $\mathbf{p}$, lies on a line, $\boldsymbol{l}$, if $p^T l = 0$
- In our case: $p_2^T l_2 = 0$
- If we now use our epipolar constraint:
  - $\boldsymbol{l}_2 = \mathrm{F} p_1$
- Then $\mathbf{p}_2^{\mathsf{T}} \mathbf{F} \mathbf{p}_1 = 0$



The Fundamental Matrix

- $\mathbf{F}$ has just seven degrees of freedom
  - Only defined up to a scale
  - Rank 2 (columns not independent)
- Can reverse the mapping

$$p_2^\top F p_1 = 0 \text{ -> } p_1^T F^T p_2 = 0$$



The Fundamental Matrix

Notes:

$$p_1 = K_1[I \mid 0]x$$
$$p_2 = K_2[R \mid t]x$$

- One way to express $\mathbf{F}$ is

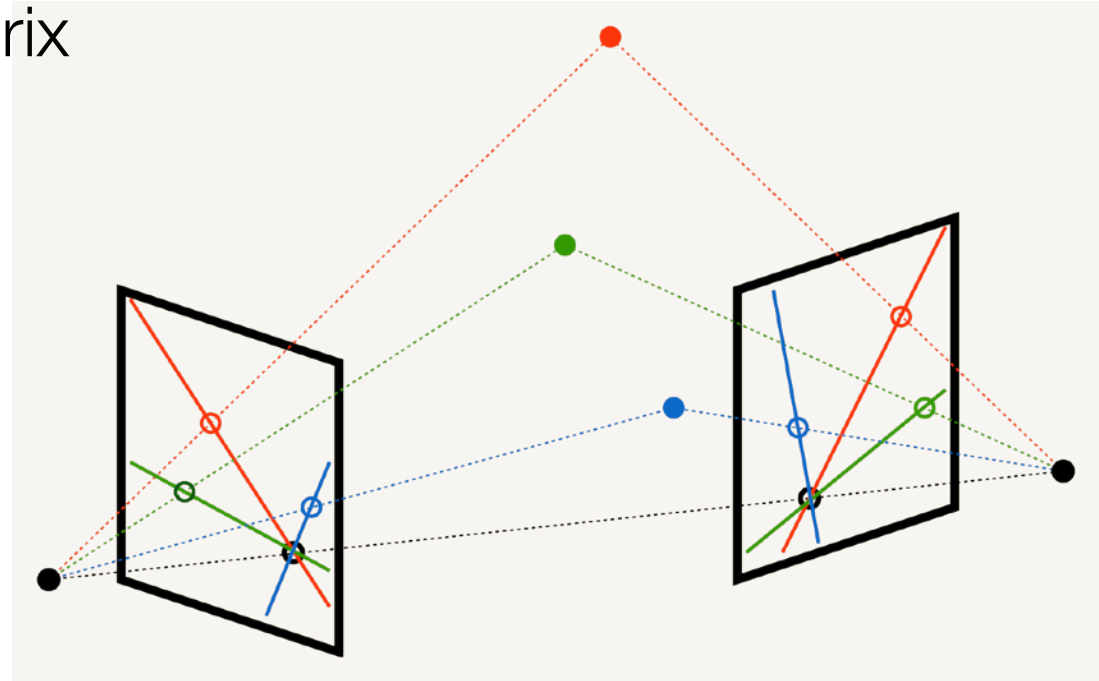
$$\mathbf{F} = \mathbf{K}_2^{-T}[\mathbf{t}]_\times \mathbf{R}\mathbf{K}_1^{-1}$$

- Here
  - $\mathbf{K}_1$ and $\mathbf{K}_2$ are the calibration matrices of the two cameras
  - $\mathbf{R}$ and $\mathbf{t}$ are the rotation and translation between them
  - $[\mathbf{t}]_\times$ is the matrix form of the cross product with $\mathbf{t}$:

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

The Essential Matrix

- For calibrated cameras
  - We can factor out $\mathbf{K}_1$ and $\mathbf{K}_2$ in $\mathbf{F} = \mathbf{K}_2^{-T}[\mathbf{t}]_\times \mathbf{R}\mathbf{K}_1^{-1}$
    - This gives us the essential matrix
      $$\mathbf{E} = \mathbf{K}_2^T \mathbf{F} \mathbf{K}_1 = [\mathbf{t}]_\times \mathbf{R}$$
  - $\mathbf{E}$ has only five DoF:
    - 3 for 3D rotation
    - 3 for 3D translation
    - Less 1 for unknown scale



Estimating the Fundamental Matrix

Fundamental Matrix

- Given matching points

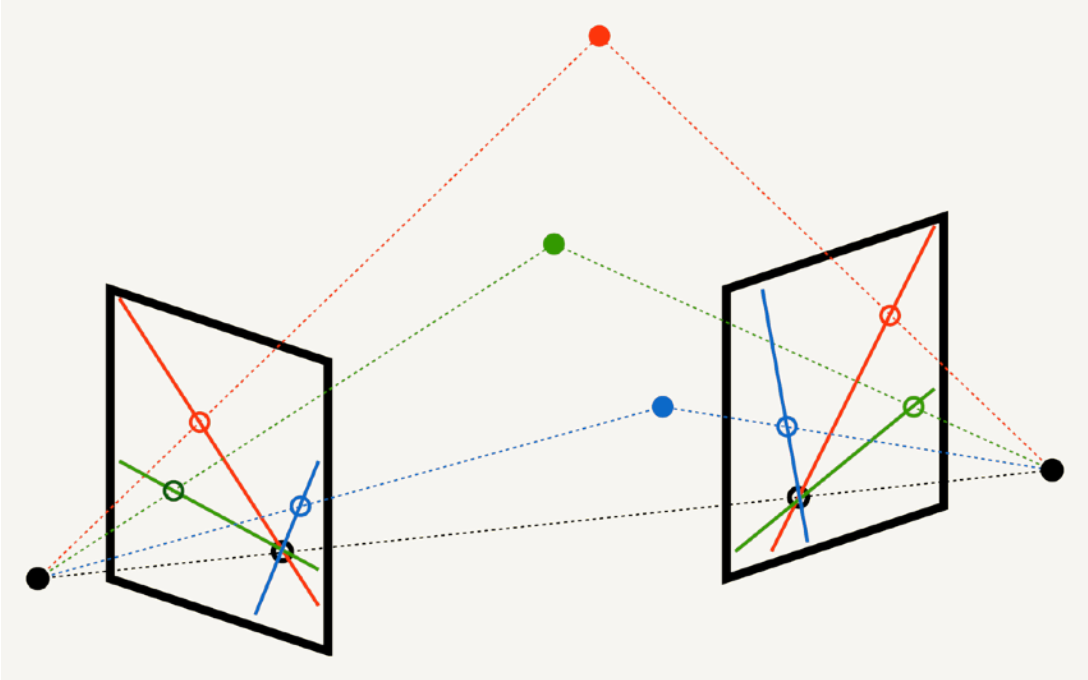
$$p = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \quad \leftrightarrow \quad \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = p'$$

- Fundamental matrix is

$$p'^{T} F p = 0$$

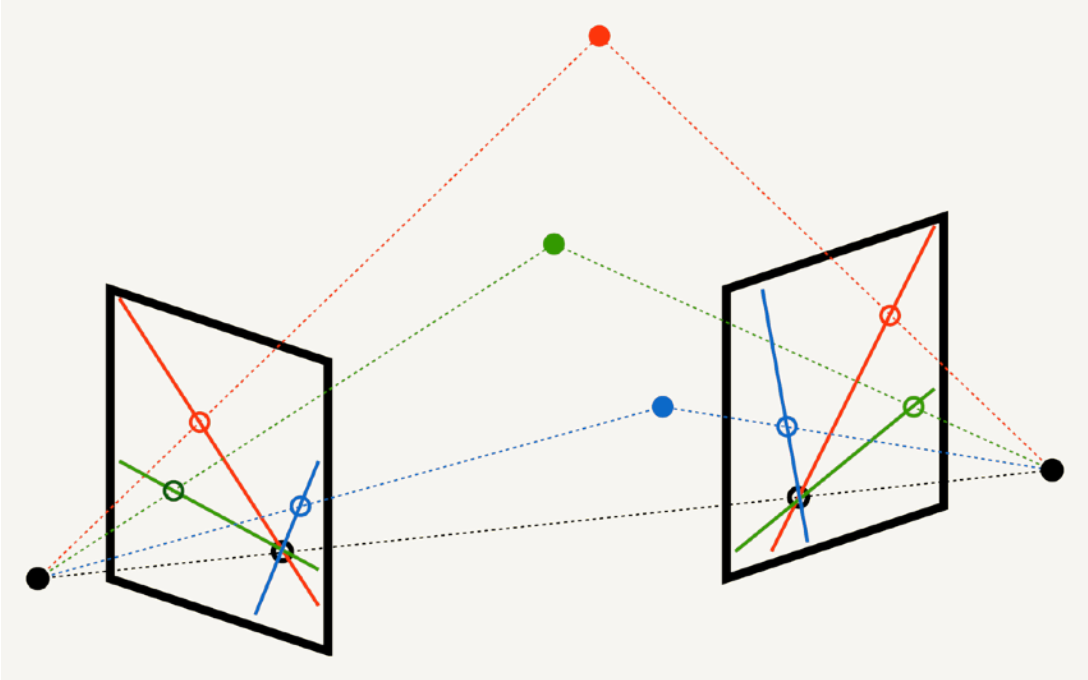
- Epipolar lines

$$\ell' = Fp \qquad \ell = Fp'$$



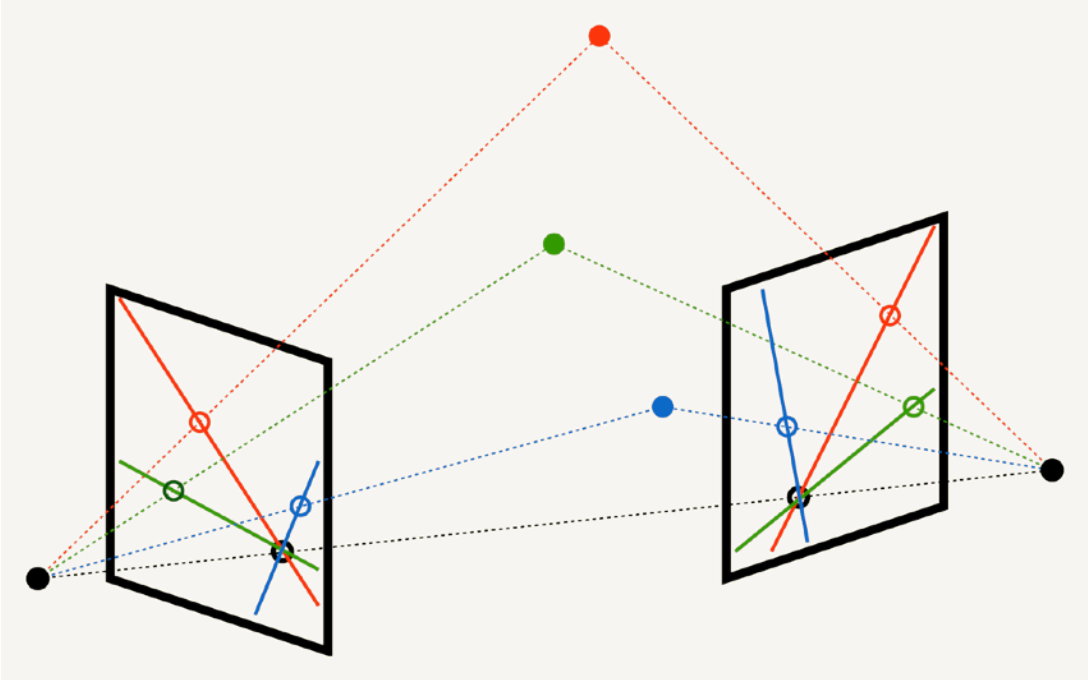
The 8-Point Algorithm

- Estimate $\mathbf{F}$ from matches
  - $\mathbf{F}$ is a 3×3 matrix – 9 values
  - Defined up to a scale – 8 DoF
  - Need 8 constraints
- $\mathbf{F}$ turns out to be rank 2
  - Only 7 degrees of freedom
  - There is a 7-point algorithm



The 8-Point Algorithm

- Many issues similar to homography estimation
  - Homogeneous systems
  - Normalizing transforms
  - RANSAC to remove outliers
- Differences:
  - Constraint $p'^{T}Fp = 0$
  - One constraint per point



The 8-Point Algorithm

- Expanding out: $p'^{T}\mathrm{F}p = 0$

$$\begin{bmatrix} u' & v' & 1 \end{bmatrix} \begin{bmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = 0 \qquad \begin{bmatrix} u' & v' & 1 \end{bmatrix} \begin{bmatrix} f_1 u + f_2 v + f_3 \\ f_4 u + f_5 v + f_6 \\ f_7 u + f_8 v + f_9 \end{bmatrix} = 0$$

Known Unknown Known

$$u\,u' f_1 + v\,u' f_2 + u' f_3 + u\,v' f_4 + v\,v' f_5 + v' f_6 + u f_7 + v f_8 + f_9 = 0$$

The Homogeneous Linear System

$$\begin{bmatrix} u_1u'_1 & v_1u'_1 & u'_1 & u_1v'_1 & v_1v'_1 & v'_1 & u_1 & v_1 & 1 \\ u_2u'_2 & v_2u'_2 & u'_2 & u_2v'_2 & v_2v'_2 & v'_2 & u_2 & v_2 & 1 \\ u_3u'_3 & v_3u'_3 & u'_3 & u_3v'_3 & v_3v'_3 & v'_3 & u_3 & v_3 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_nu'_n & v_nu'_n & u'_n & u_nv'_n & v_nv'_n & v'_n & u_n & v_n & 1 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \\ f_8 \\ f_9 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

A (Known) $\quad$ f (Unknown)

$$Af = 0$$

Normalised 8-Point Algorithm

Input: $n \geq 8$ correspondences $p_i \leftrightarrow p_i'$

Output: Fundamental matrix, F, such that $p_i'^T \mathrm{F} p_i = 0$

1. **Normalisation:** Find T and T' to centre $\tilde{p}_i = \mathrm{T} p_i$ and $\tilde{p}_i' = \mathrm{T}' p_i'$ on the origin with average length $\sqrt{2}$

2. **Direct Linear Transform:**
   1. Form the matrix A from $\tilde{\mathbf{u}}_i$ and $\tilde{\mathbf{u}}_i'$
   2. Compute the SVD of A, and find the smallest eigenvector, $\tilde{\mathbf{f}}$
   3. Reshape $\tilde{\mathbf{f}}$ to give the 3×3 fundamental matrix $\tilde{\mathrm{F}}$

3. **Denormalisation:** Final solution is $\mathrm{F} = \mathrm{T}'^T \tilde{\mathrm{F}} \mathrm{T}$

Computing the Fundamental Matrix in Practice

```
// Compute the Fundamental Matrix using RANSAC
// FM_RANSAC is robust to outliers (bad matches)
// The third parameter is the RANSAC reprojection threshold (in
pixels)
// The fourth parameter is the confidence level (e.g., 99%)
cv::Mat F = cv::findFundamentalMat(points1, points2,
cv::FM_RANSAC, 3.0, 0.99);
```

7-Point Algorithm

The 7-Point Algorithm

- Exploiting the geometric constraint that the determinant of F must be zero
- The "Two-Solution" Problem: With only 7 points, the system is underdetermined, resulting in two initial 'basis' solutions, let's call them F1 and F2
- Finding the Right Mix: The true solution F is a specific blend of these two basis solutions: $F=\alpha F1 +(1-\alpha)F2$
- The Cubic Equation: By enforcing the constraint that $\det(F)=0$, we get a cubic polynomial for the mixing value α

- Pro: Requires fewer points
- Con: More complex due to solving a cubic polynomial; can have multiple solutions; less robust to noise than the 8-point algorithm with normalisation and RANSAC

Alternative Stereo Calibration

Stereo Camera Calibration

- Alternatively use calibration pattern to recalibrate a stereo camera
- Measures the precise intrinsics (K) and extrinsics (R, t) for a fixed stereo rig
- Requires multiple views of a known calibration pattern (e.g., chessboard) as a reference
- Minimizing the reprojection error between detected and predicted pattern corners


```
double err = cv::stereoCalibrate(objectPoints,
imagePointsL, imagePointsR, K_L, distCoeffsL, K_R,
distCoeffsR, IMAGE_SIZE, R, T, E, F);
```

Stereo Rectification

Stereo Rectification

- The fundamental matrix
  - Given a point in one image gives epipolar line in the other
  - Matching point must lie on the epipolar line
  - For ideal parallel cameras, epipolar lines are just rows of the image
  - General case not so simple

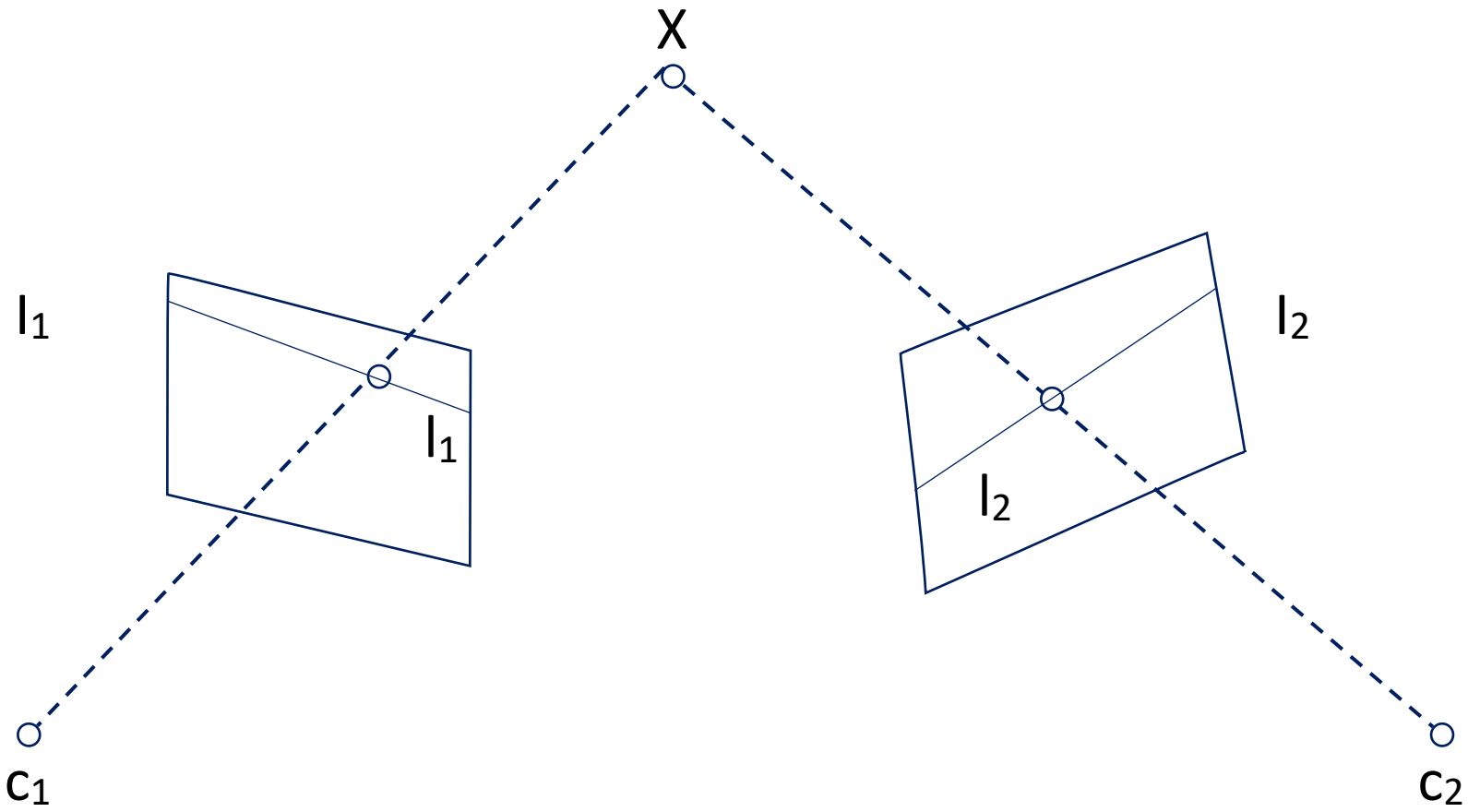
Stereo Rectification

- Goal: Transform the images to simulate a perfect, parallel camera setup
- Why?:
  - To make all epipolar lines horizontal and aligned. This simplifies the search for matching points from a complex 2D problem to a simple and fast 1D search along the same image row
- How?:
  - The algorithm uses the Fundamental (F) or Essential (E) Matrix as input to calculate a warping transformation for each image
- Result: A new pair of rectified images where finding disparity becomes straightforward, paving the way for creating a dense depth map

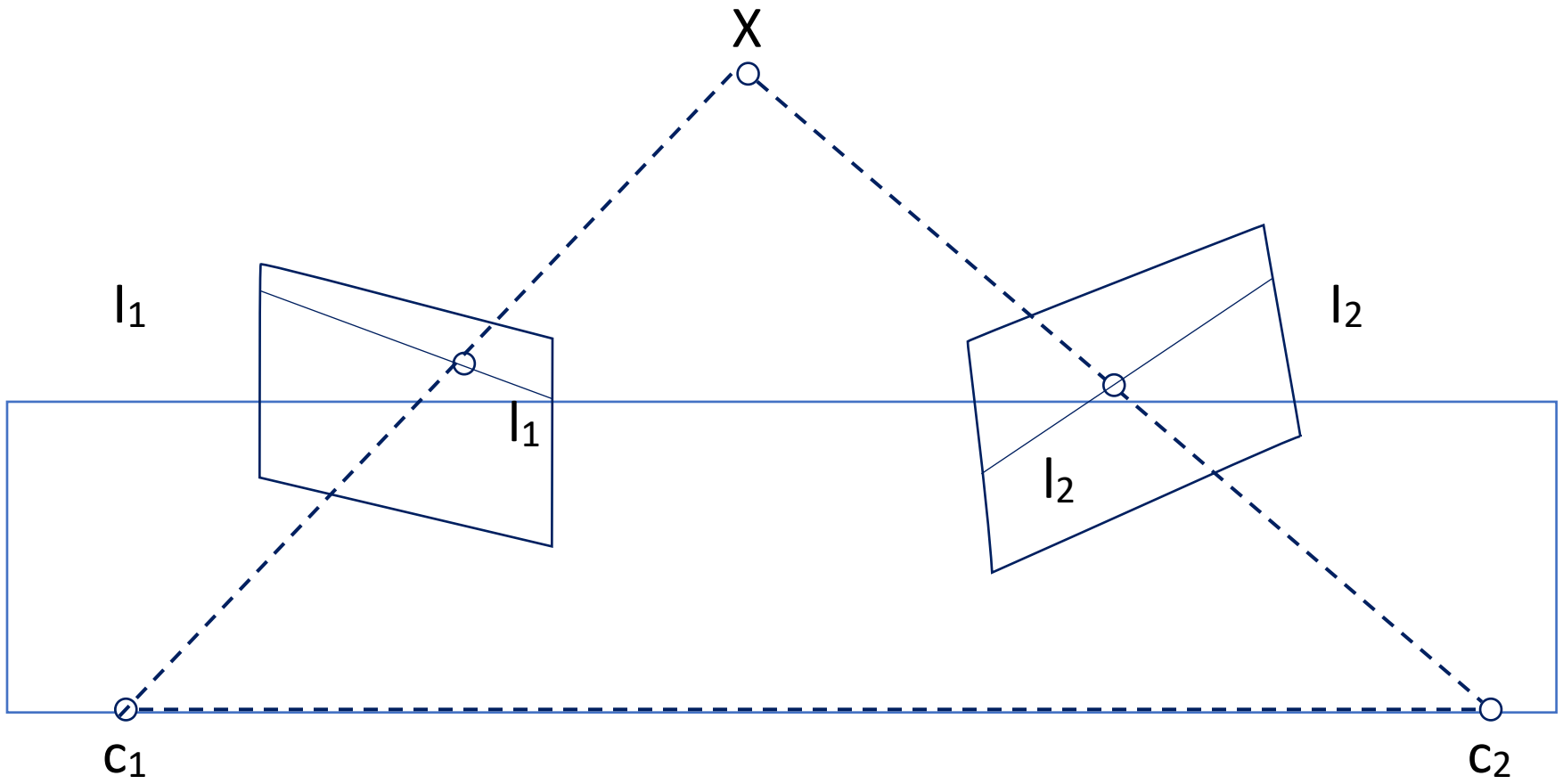
Stereo Rectification

- Given two cameras
  - Know their calibration matrices and distortion coefficients
  - Know their relative poses and therefore cameras' centres
- Take any plane parallel to the line between the camera centres
- Project the images onto this plane
- Resulting images have parallel stereo geometry

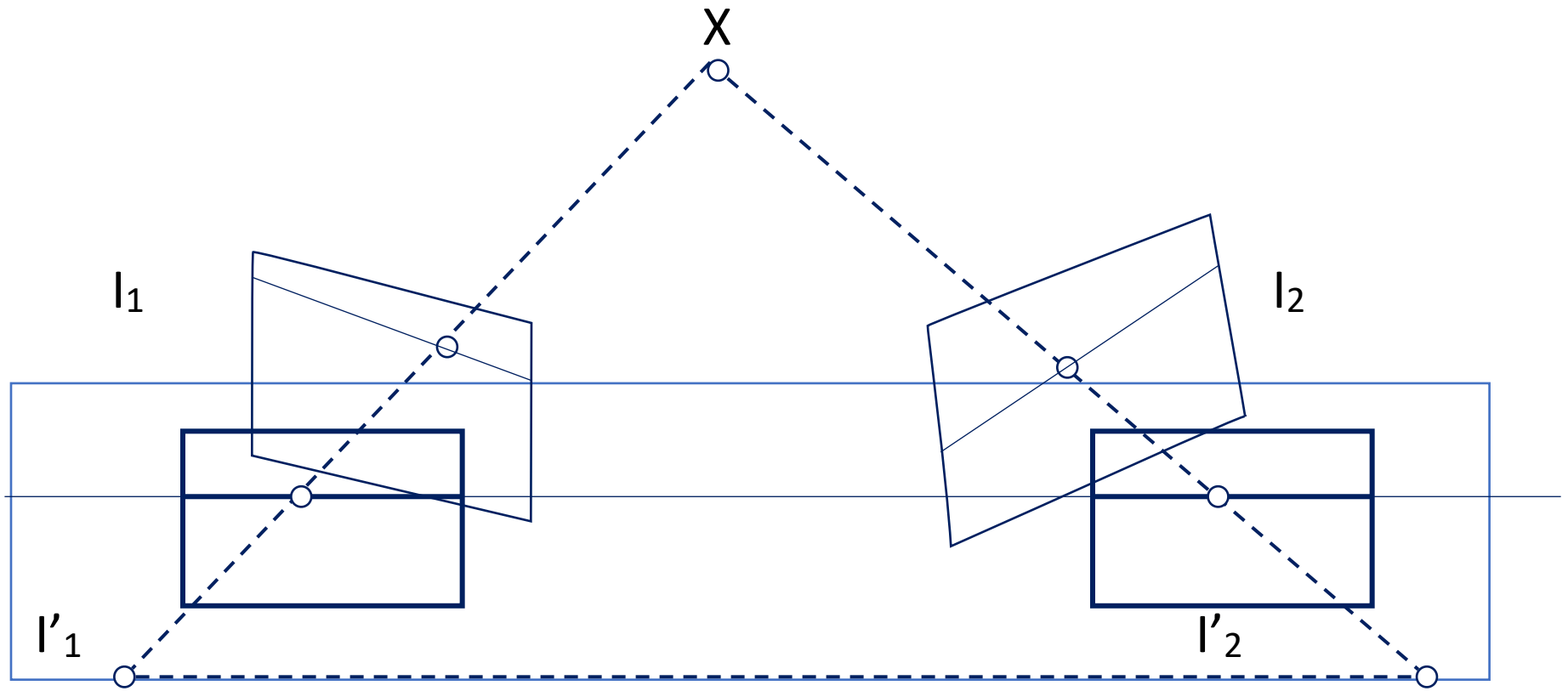
Stereo Rectification



Stereo Rectification



Stereo Rectification



Stereo Rectification

- Stereo Rectification:
  - Warping a general stereo pair to this simplified form

```
Step 1:
cv::stereoRectify computes the matrices needed to apply the rectification
transformations.

Step 2:
cv::initUndistortRectifyMap creates a mapping from the original images to the
rectified images.

Step: 3:
cv::remap applies the mapping and produces the final rectified images.
```

Stereo Rectification in OpenCV

Input

K1 and d1: Camera Calibration of first camera

K2and d2: Camera Calibration of second camera

R and T: Relative Pose of the two cameras

```
cv::stereoRectify(K1,d1,K2,d2,imgSize,R,T,R1,R2,P1,P2,Q);
```

R1 and R2: rotation between original image and rectified image plane

P1 and P2: rectified projection matrices

Q: relates disparities to depth

Output

Stereo Rectification in OpenCV

Input

K and d: Camera Calibration information

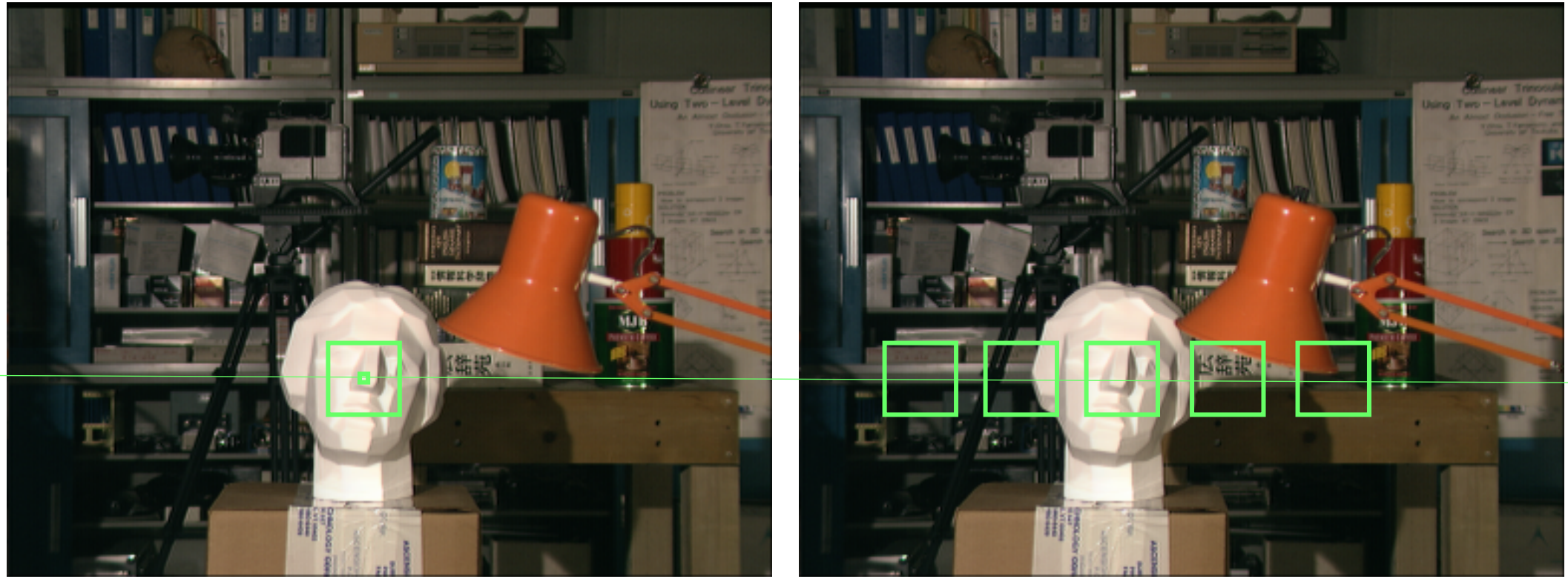
R and P: Output from stereorectify

```
cv::initUndistortRectifyMap(K,d,R,P,imgSize,
                            CV_32FC1, map1, map2);
```

map1 and map2: shift in x and shift in y

```
cv::remap(original,rectified,map1,map2,flags)
```

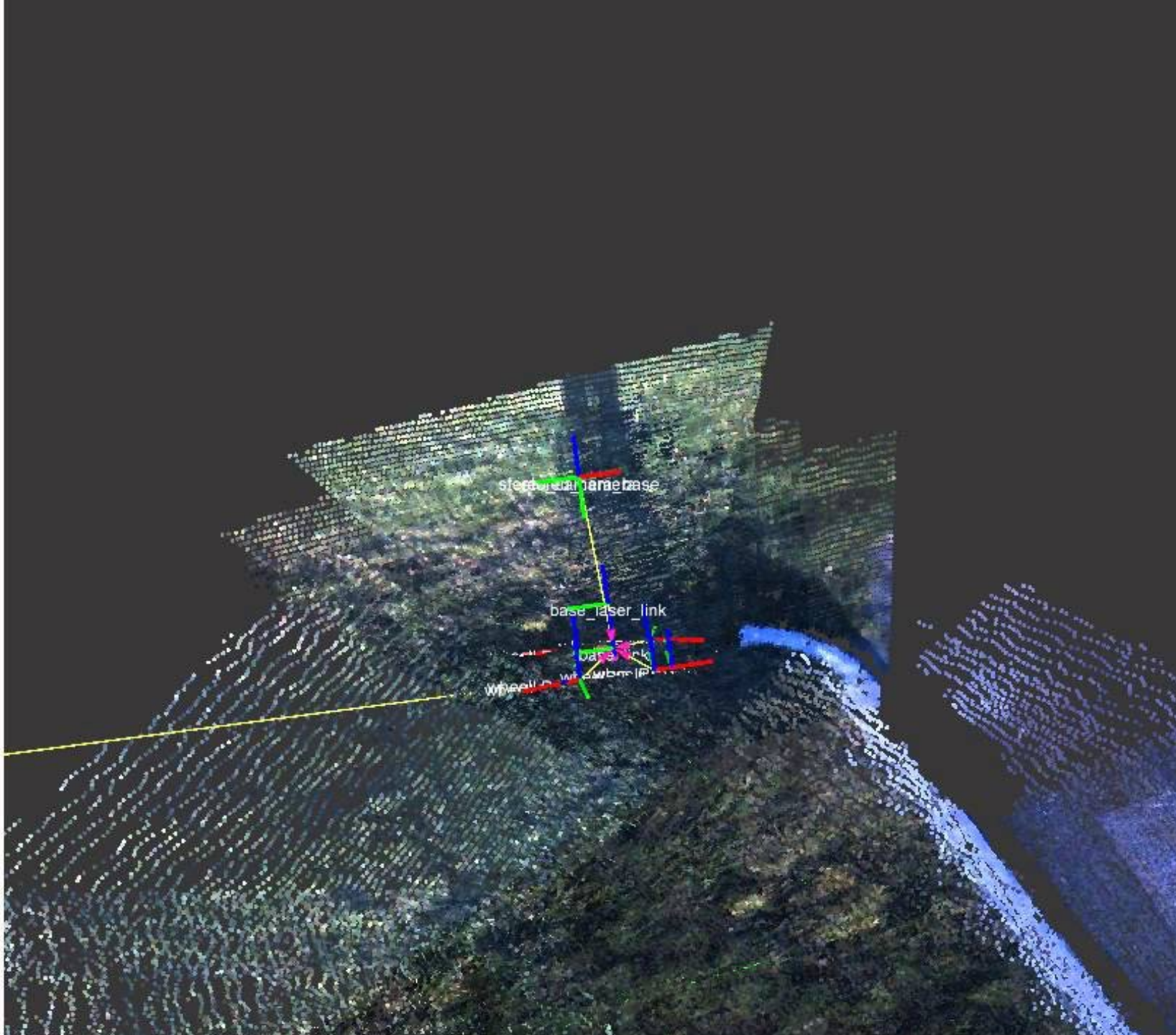
Stereo Rectification



Estimating Disparity

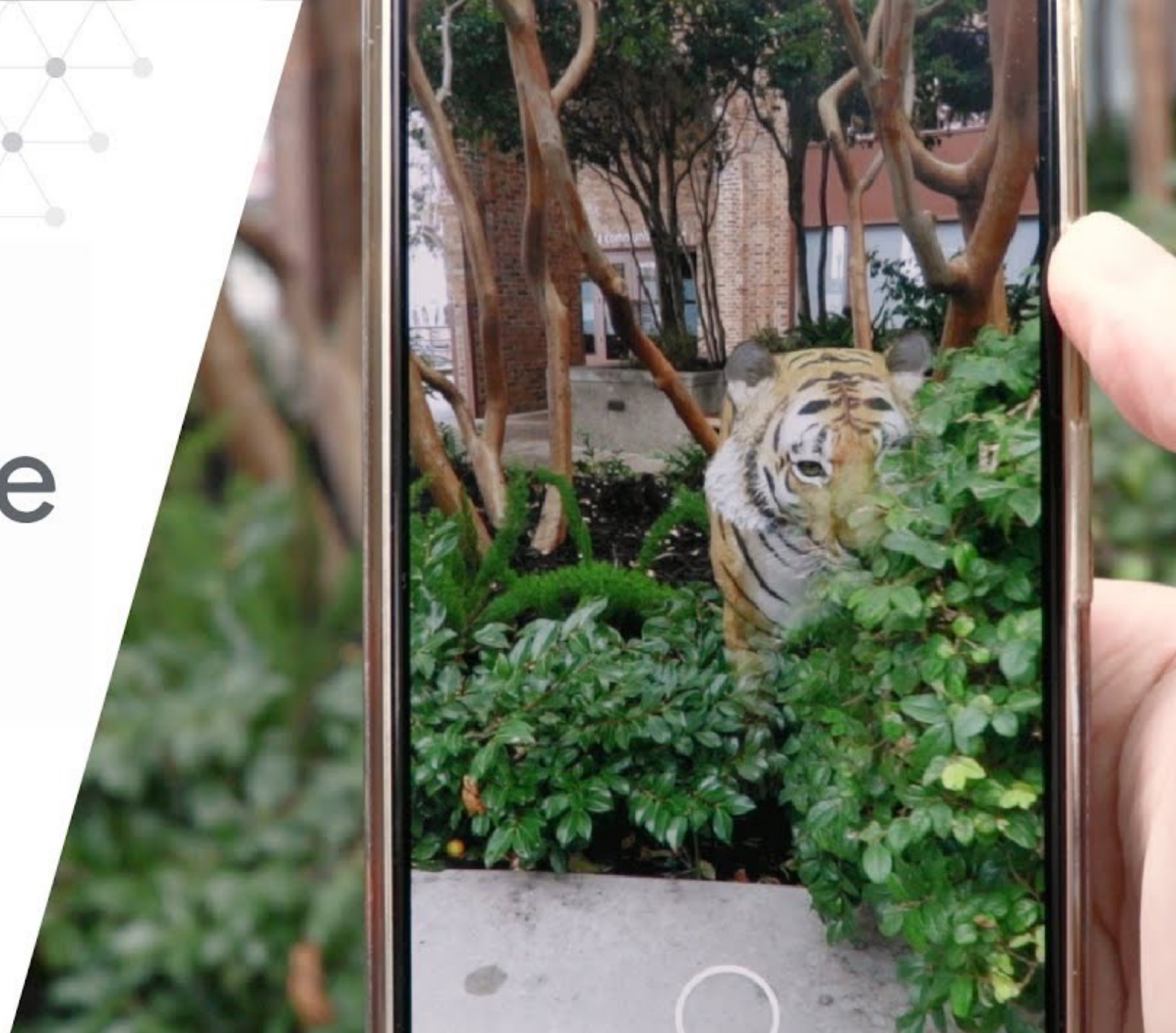
- Assuming rectified images
  - Given (u_1, v_1) in left image
  - Matching point is (u_2, v_2)
  - We have $v_2 = v_1$ so search on same row in right image
  - z is positive so $u_2 < u_1$, search from u_1 to $u_1 - D$, where D is max disparity
- How to compare points?
  - Simple block matching
  - Compare a window around the two points
  - Sum of absolute differences
  - Sum of squared differences
- Why is this OK here, but not in mosaicing?

Dense Stereo


base_laser_link

ARCore

Depth API



Dense Stereo Estimation

Stereo So Far

- We can calibrate a stereo pair of cameras
  - Find matrix $\mathbf{K}$ and distortion values $\mathbf{d}$ for each camera
  - Find relative pose, $\mathbf{R}$, $\mathbf{t}$, between the two cameras
- Can find fundamental matrix, $\mathbf{F}$
  - Gives epipolar lines of points
- If we don't know the pose
  - Can use camera calibration to convert $\mathbf{F}$ into the essential matrix, $\mathbf{E}$
  - This contains the pose, which we can extract $E = [t]_{\times}R$
- Can then rectify images
  - Make a parallel stereo pair

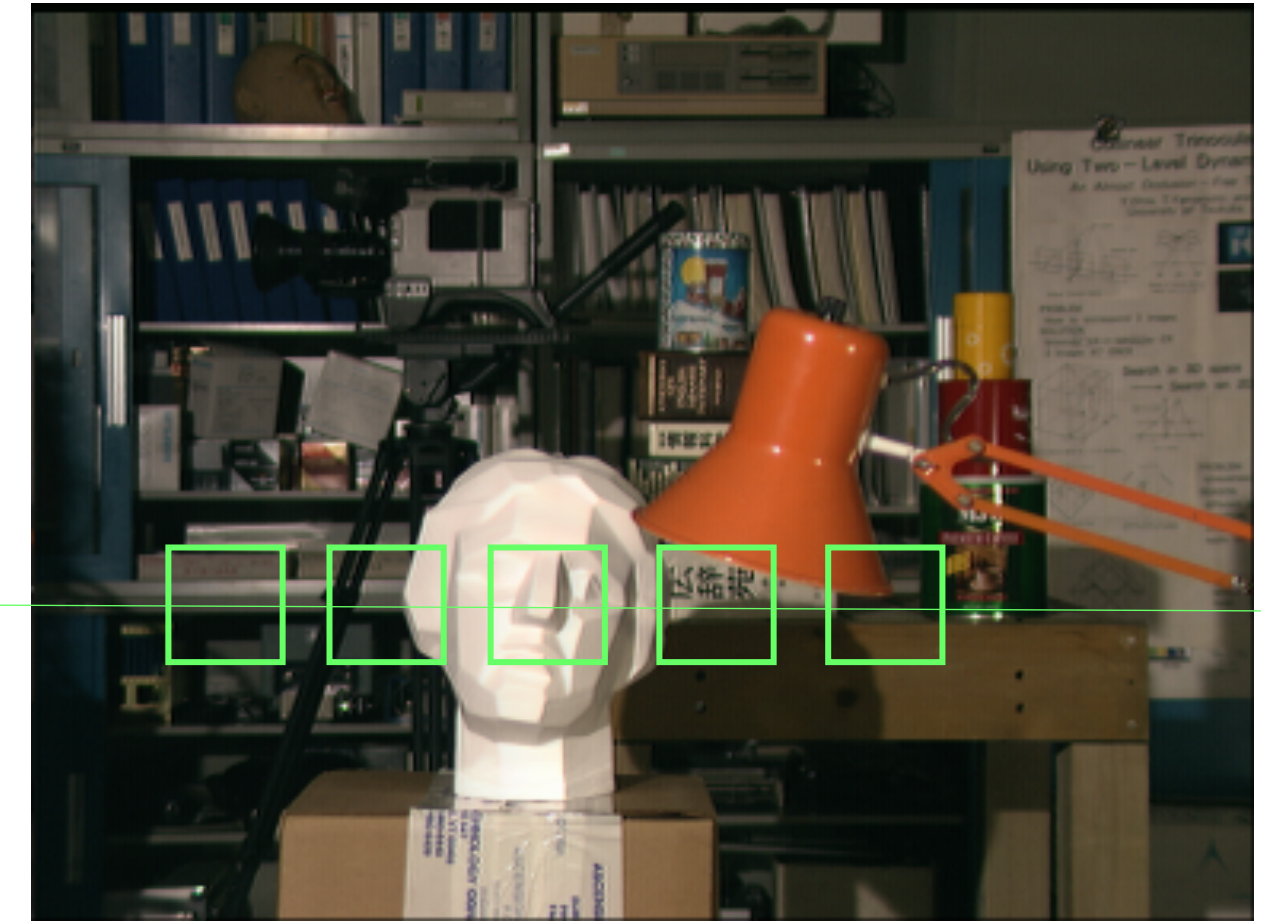
Dense Stereo Estimation

- We want to find the depth at every pixel in the image
  - For a parallel pair, this is inversely related to **disparity**
  - So for each pixel in one image, find matching pixel in the other image
  - Lies on **epipolar** lines, which are rows for parallel pair

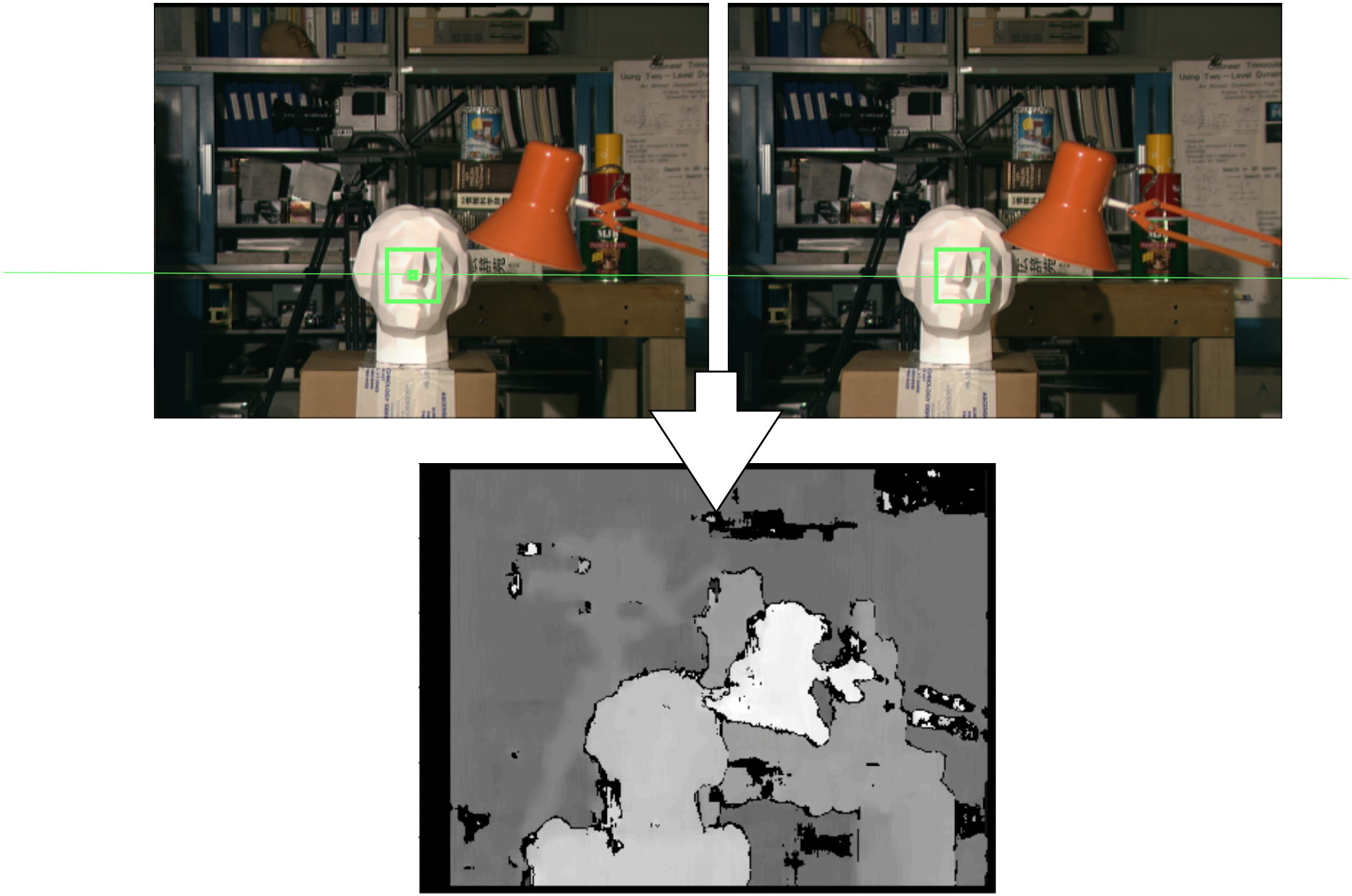
Dense Stereo Estimation

- Simple Block Matching
  - Search along epipolar line for a range of disparities,
  - Compute a matching function and find best match
  - Sum of squared differences $\sum_{u,v \in R} \left(I_1(u,v) - I_2(u+d,v)\right)^2$
  - Sum of absolute difference $\sum_{u,v \in R} \left|I_1(u,v) - I_2(u+d,v)\right|$

Block Matching



Block Matching

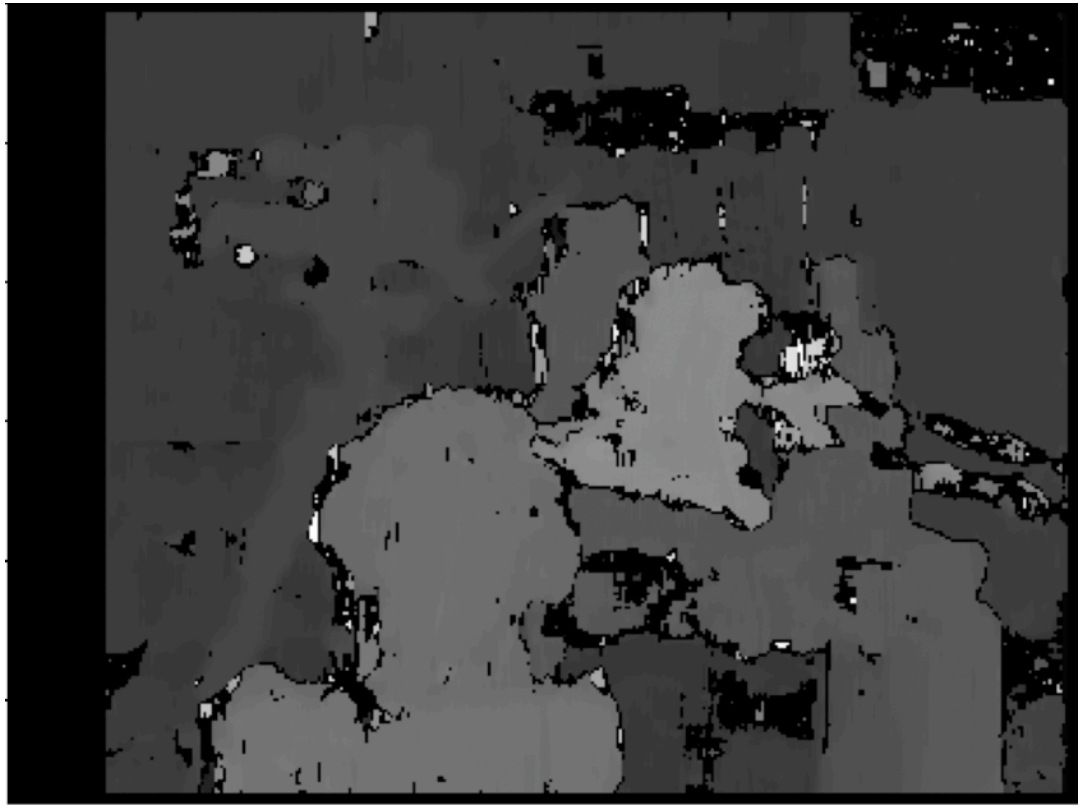


Considerations Related to Block Matching

- Image appearance change
  - Recall why we didn't use this in mosaicing, but used SIFT
  - Is this a problem here?
- Window Size
- Choice of cost functions
  - Sum of Squared Distances (SSD), or Sum of Absolute Distances (SAD)?
  - Normalised versions
  - Correlation functions

Problems with Block Matching

- Each pixel's depth is estimated independently
  - No guarantee of consistency between neighbours
  - Leads to noisy depth maps
- Add regularisation to the stereo cost function
  - Penalise sharp changes in disparity values



Regularised Cost Function

- Consists of two parts

$$C(d) = S(d) + R(d)$$

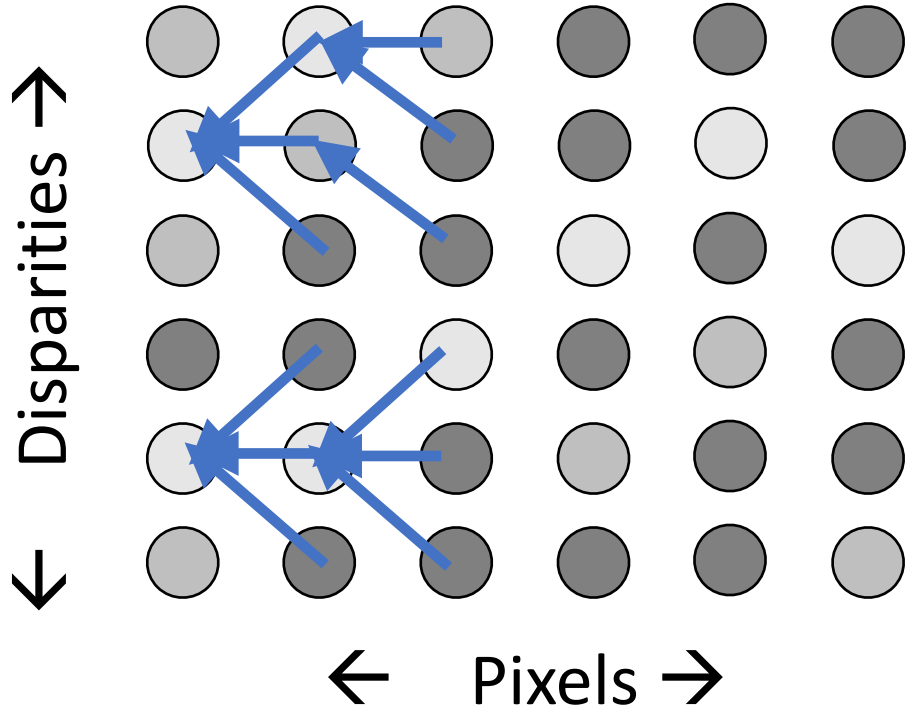
  - C is the total cost for disparity d at this pixel
  - S is the similarity between the two image patches with disparity d (e.g. SSD)
  - R is the difference between d and the disparity at all of the neighbouring pixels

Regularised Cost Function

- Minimise this across all pixels in the image
  - Each pixel's disparity affects its neighbours' cost via
  - This makes the problem NP-Hard – and we don't have small size problems
  - Need approximate solutions

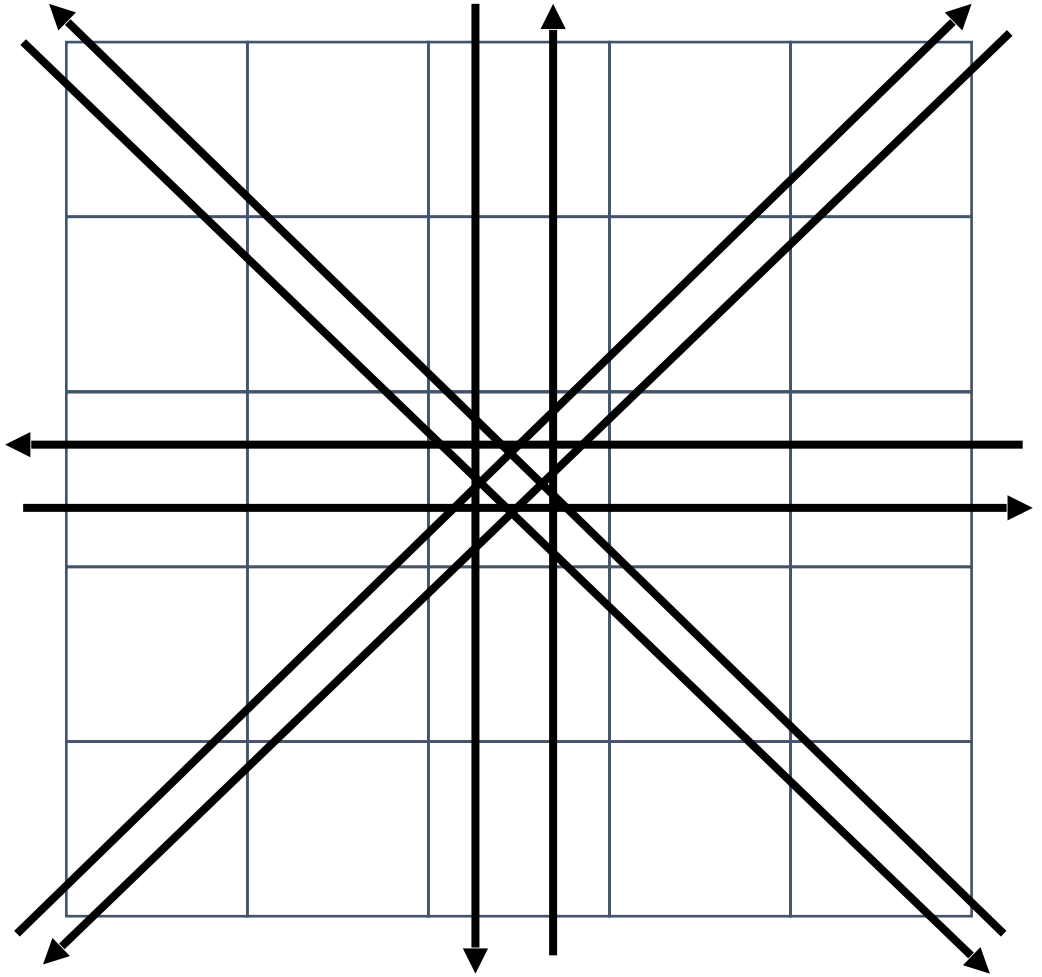
Scanline Solution

- Consider a single row of the image – a 1D case
  - This turns out to not be NP
  - Each pixel has a set of possible disparity values
  - Use dynamic programming to find optimal solution

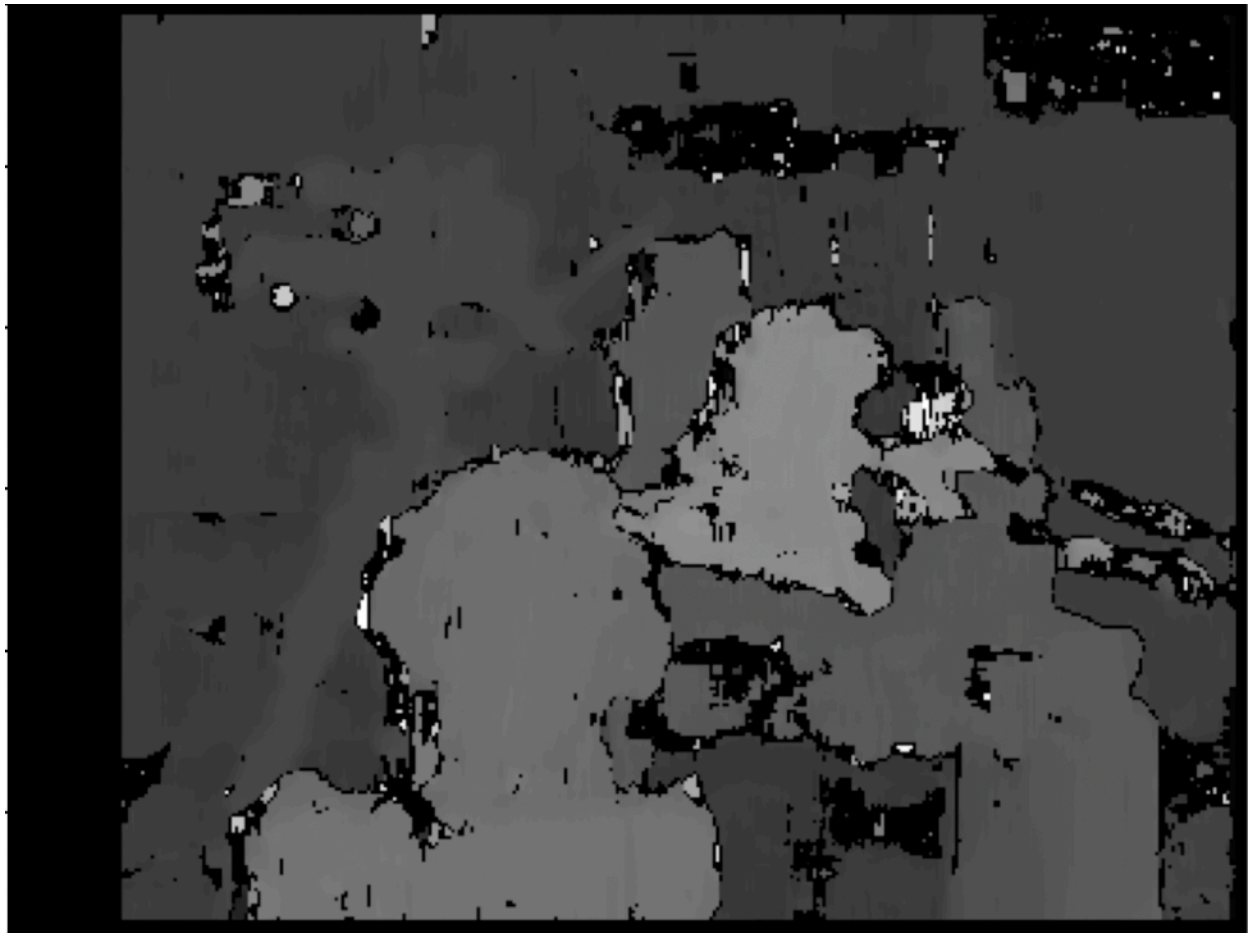


Scanline Solution

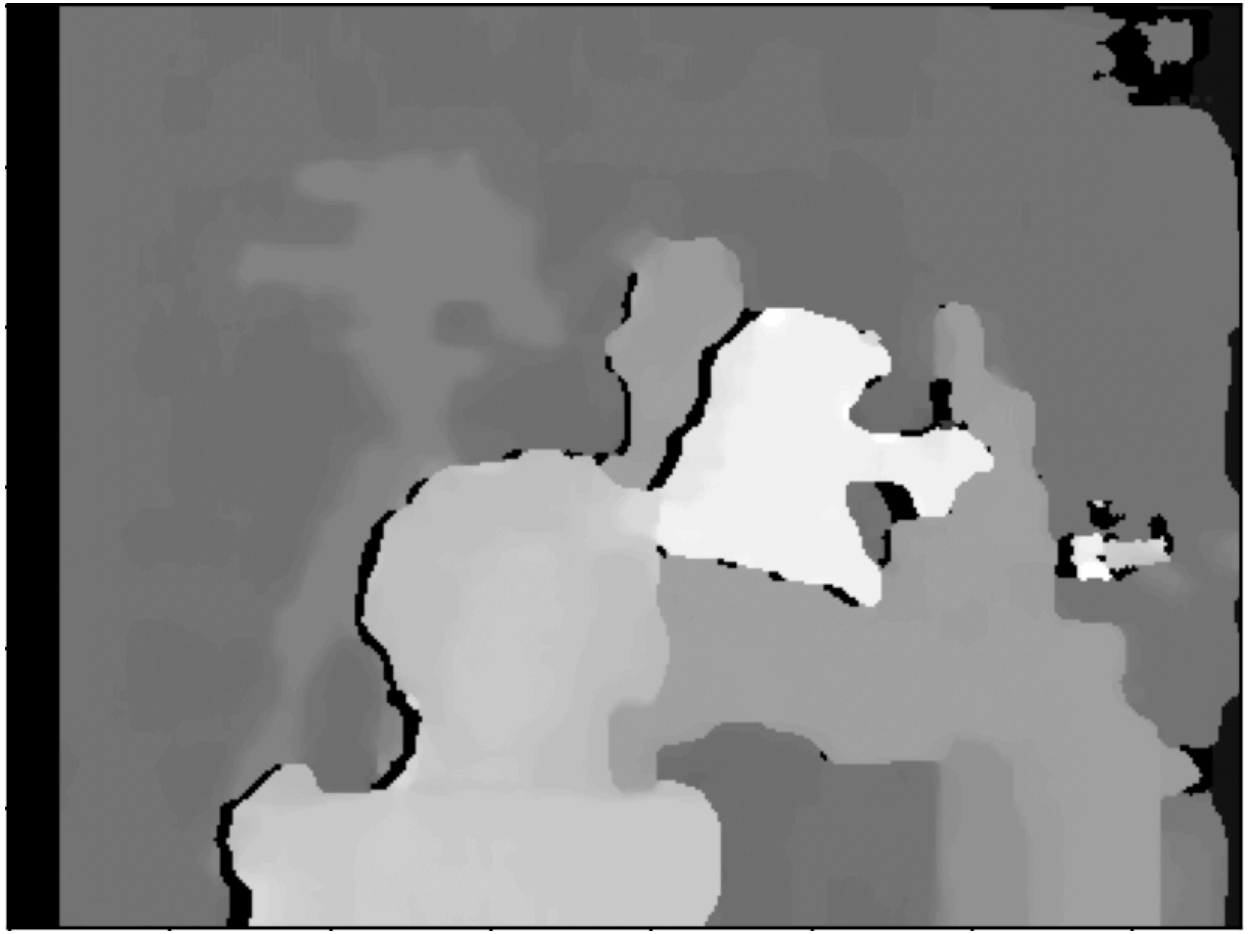
- Generalise the scanline solution to 2D
  - Follow multiple 1D lines and sum up total costs
  - 4, 8 or 16 directions common
- This is memory expensive



Disparity Computation



Simple Blockmatching



Semi-Global Blockmatching

No Stereo Camera?

- Convolutional neural networks (CNNs) have been applied to monocular depth estimation
- Common architectures include U-Net, ResNet, and DenseNet
- During training:
  - Network is given pairs of images and their corresponding ground-truth depth maps
  - Minimize the difference between predicted depth and ground truth depth

https://github.com/yuhsuanyeh/BiFuse

Next time:

The end!