計算機実験II – 3D Computer Vision

1/7, 1/21, 1/28

Meng-Yu Jennifer Kuo

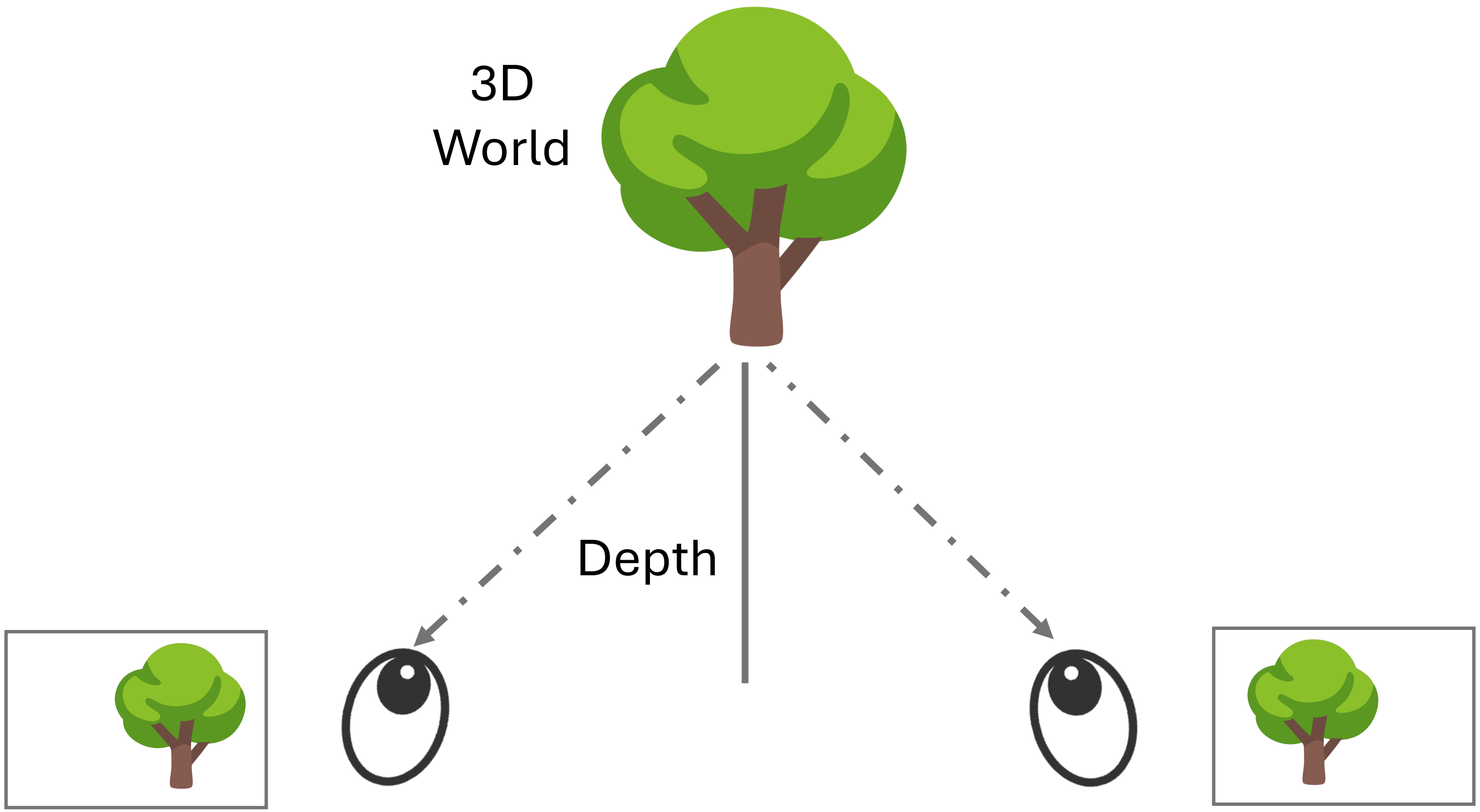
2025 Fall

Schedule

- 1/7 – Stereo Vision Part 1
- 1/21 – Stereo Vision Part 2
- 1/28 - 奈良県立医科大学の見学

- **1/31 - Final Report Deadline (submitted via LMS)**

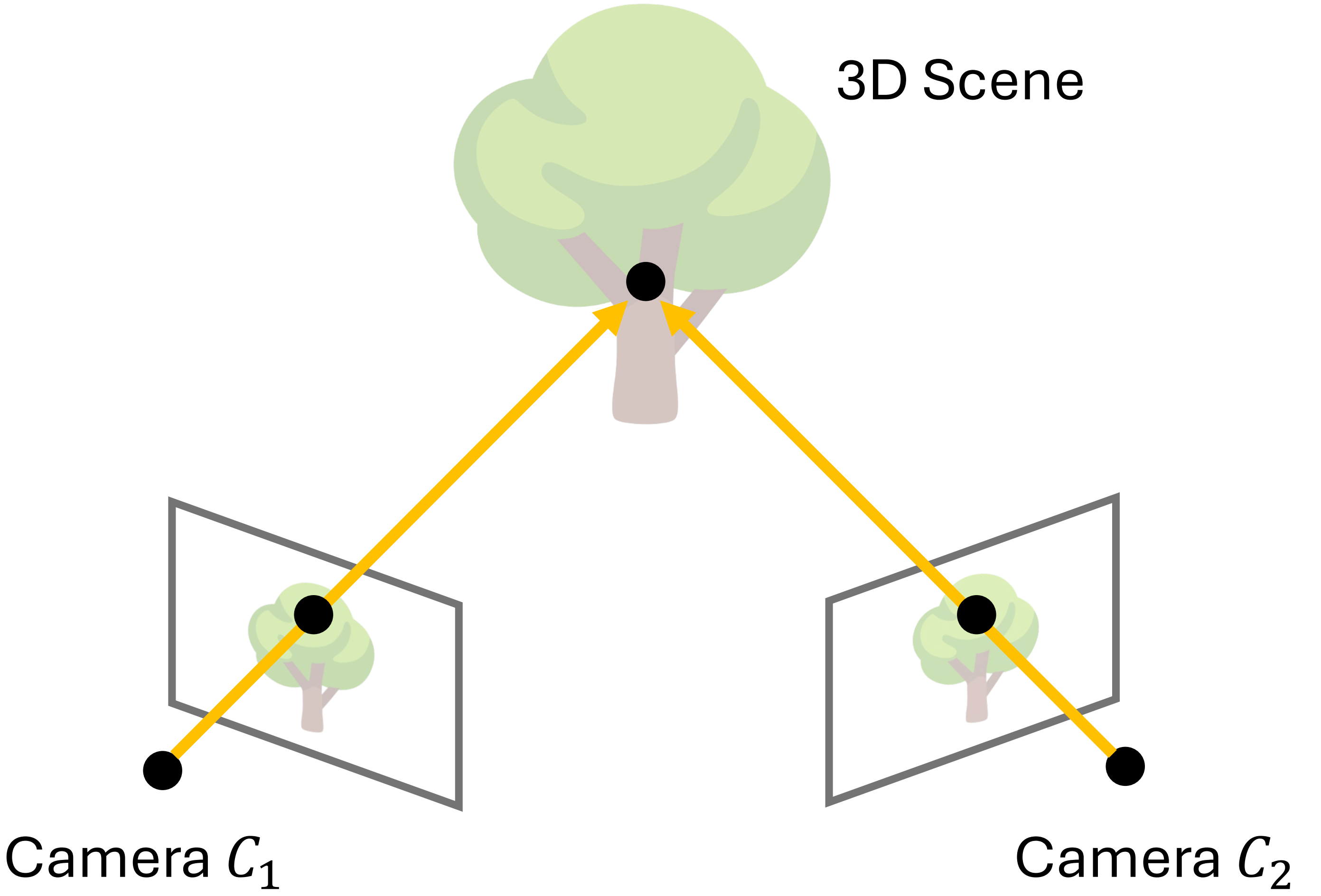
What is Stereo Vision?



Our eyes are stereo cameras

What is Stereo Vision?

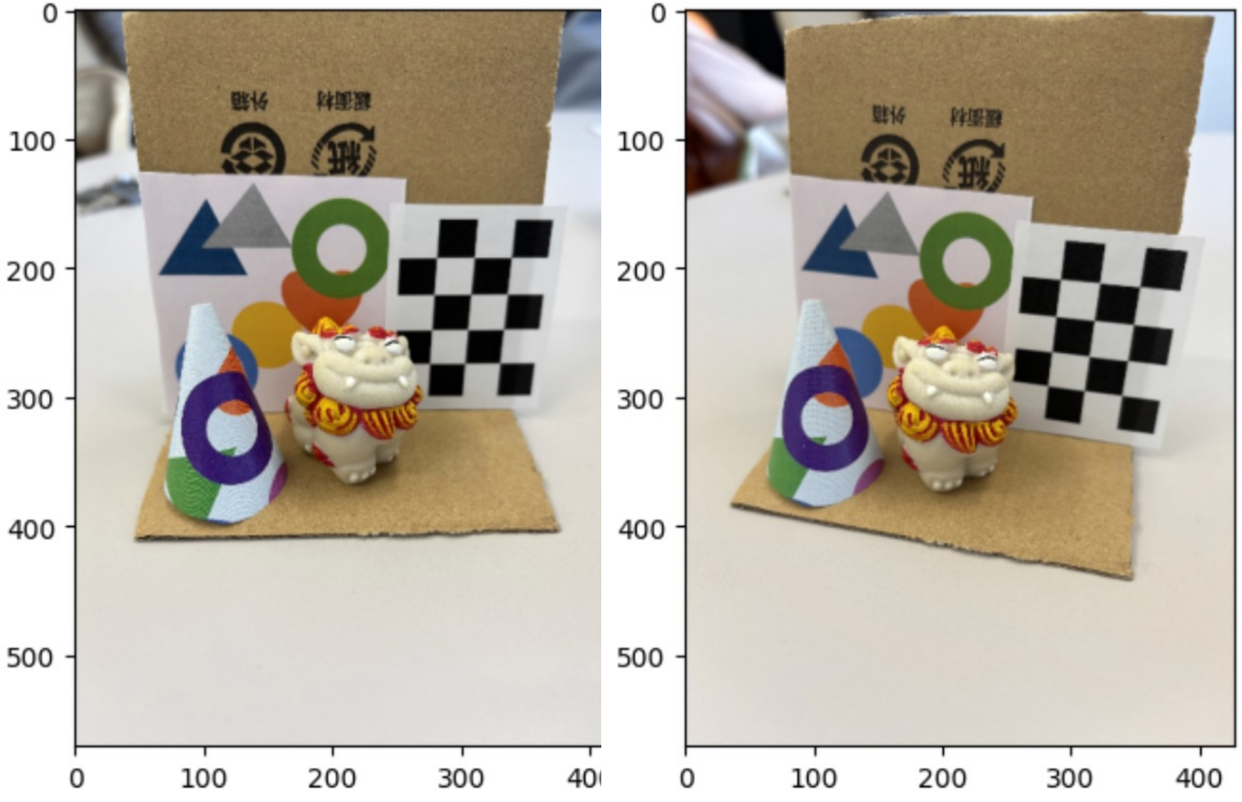
Goal: Compute **3D structure** of static scene from **2 views**



Stereo Vision – Expected Result

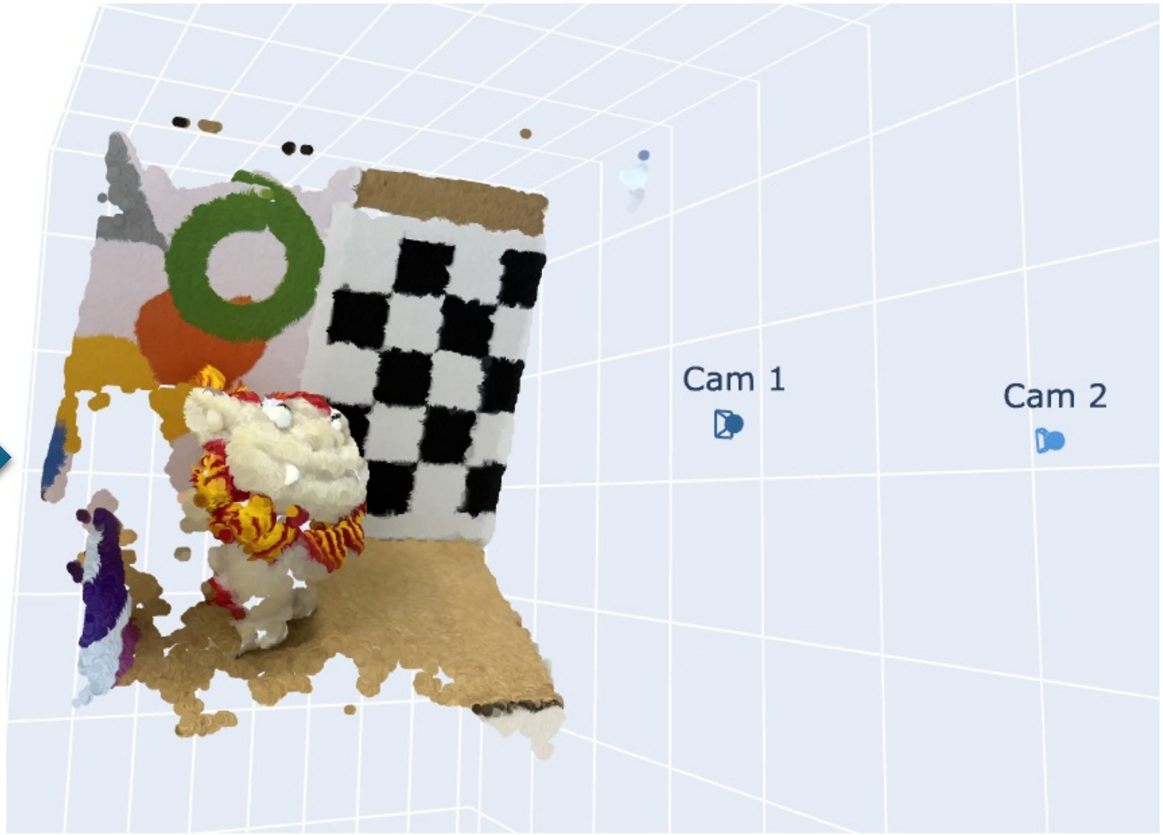
INPUT:

2-view images

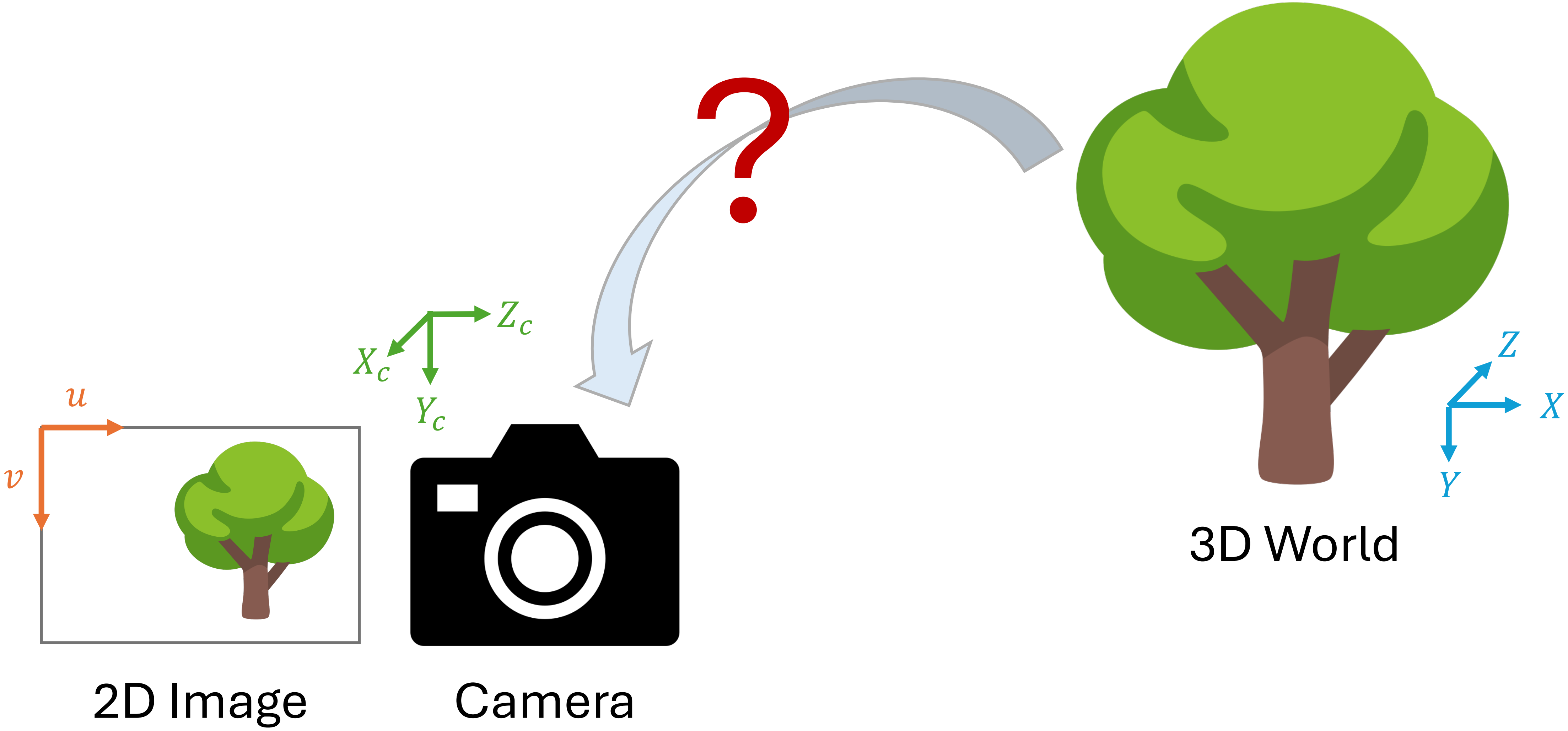


OUTPUT:

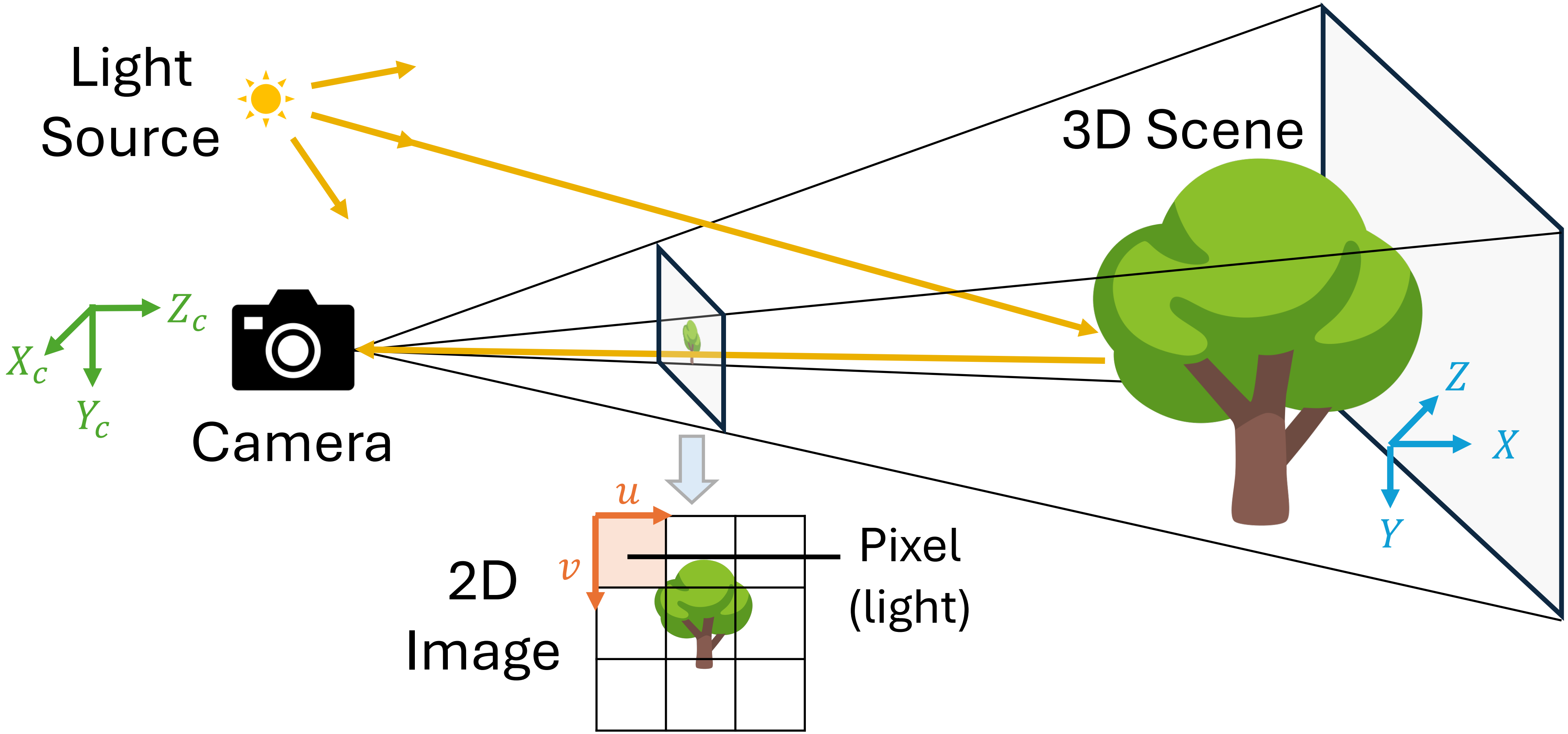
3D point cloud + RGB



How does a camera capture/project an object onto its image plane?

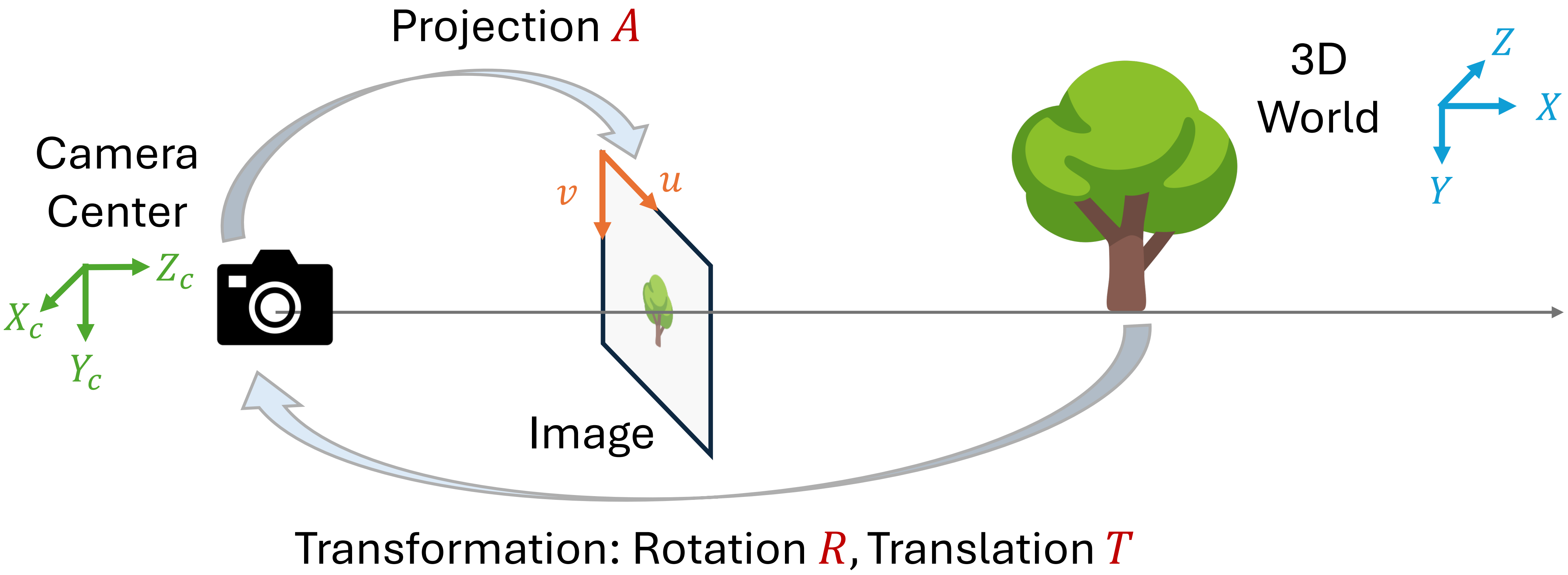


How does a camera capture/project an object onto its image plane?



➔ Let's look at this mathematically....

How does a camera capture/project an object onto its image plane?



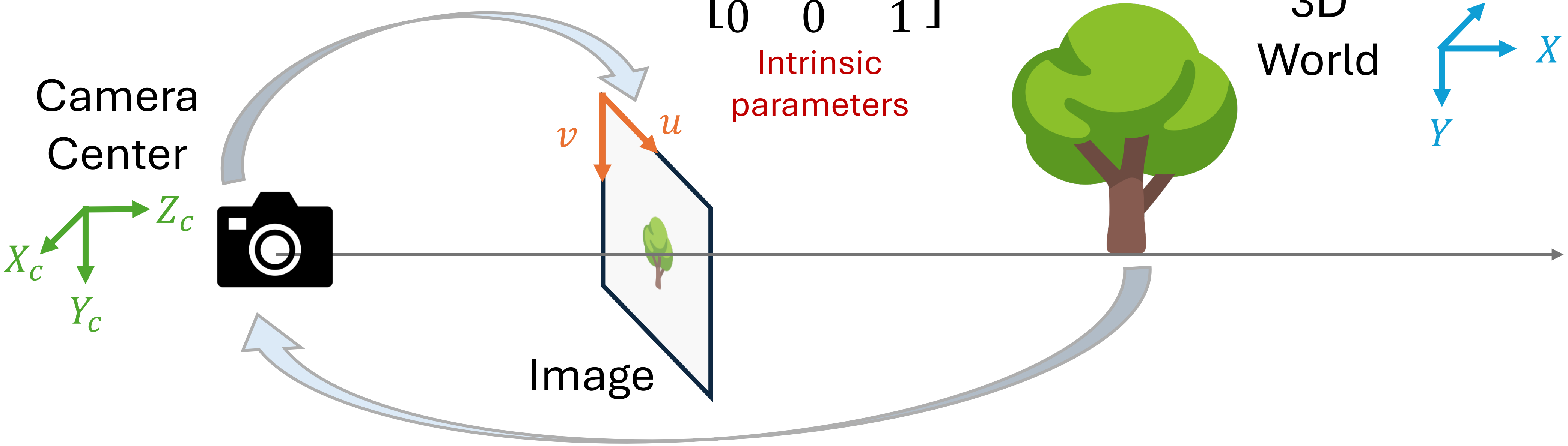
How does a camera capture/project an object onto its image plane?

Focal length

$$\text{Projection } A = \begin{bmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Image center coordinates

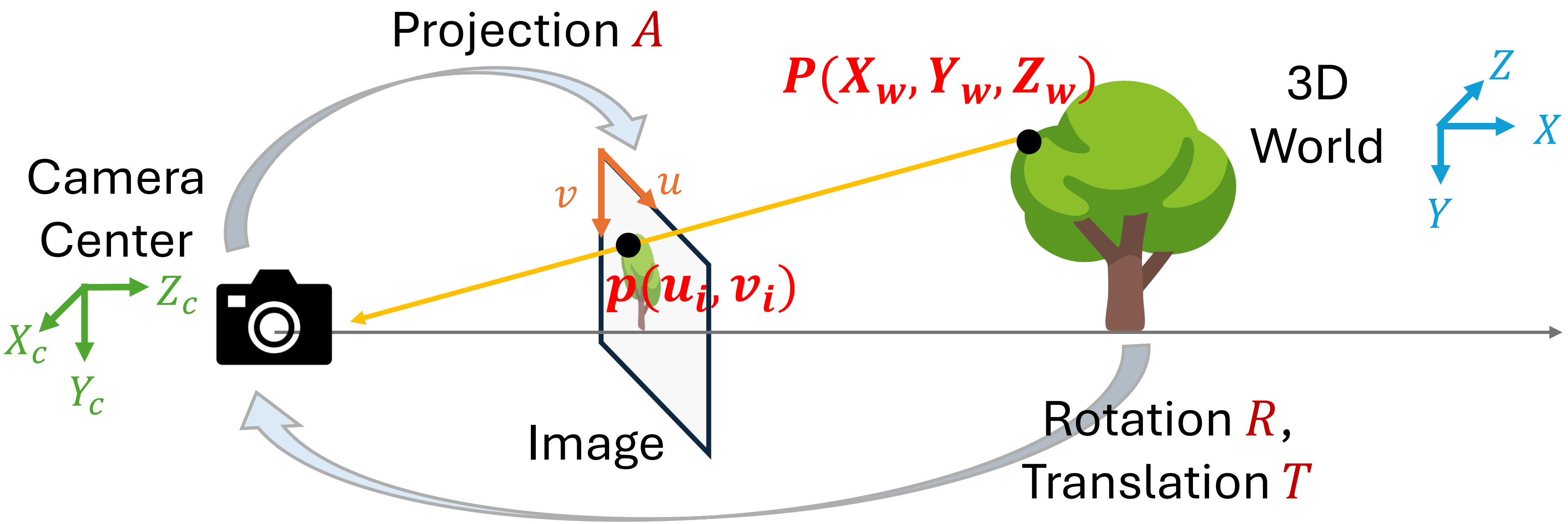
Intrinsic parameters



$$\text{Transformation: Rotation } R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \text{Translation } T = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

Extrinsic parameters

How does a camera capture/project an object onto its image plane?



Camera Projection

$$s\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = A\begin{bmatrix} R & T \end{bmatrix}\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

What is Stereo Vision?

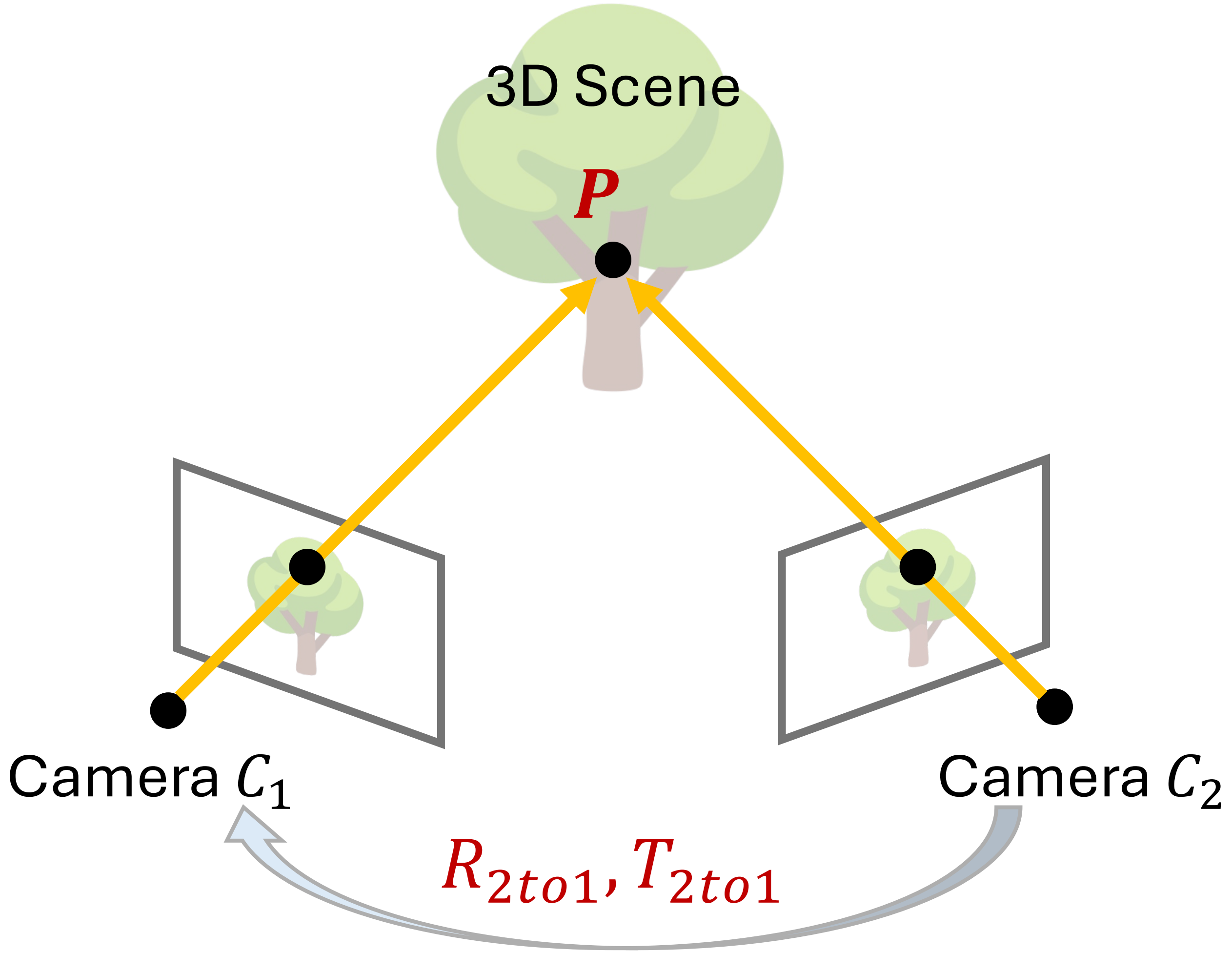
Goal: Compute **3D structure** of static scene from **2 views**

INPUT: 2 images

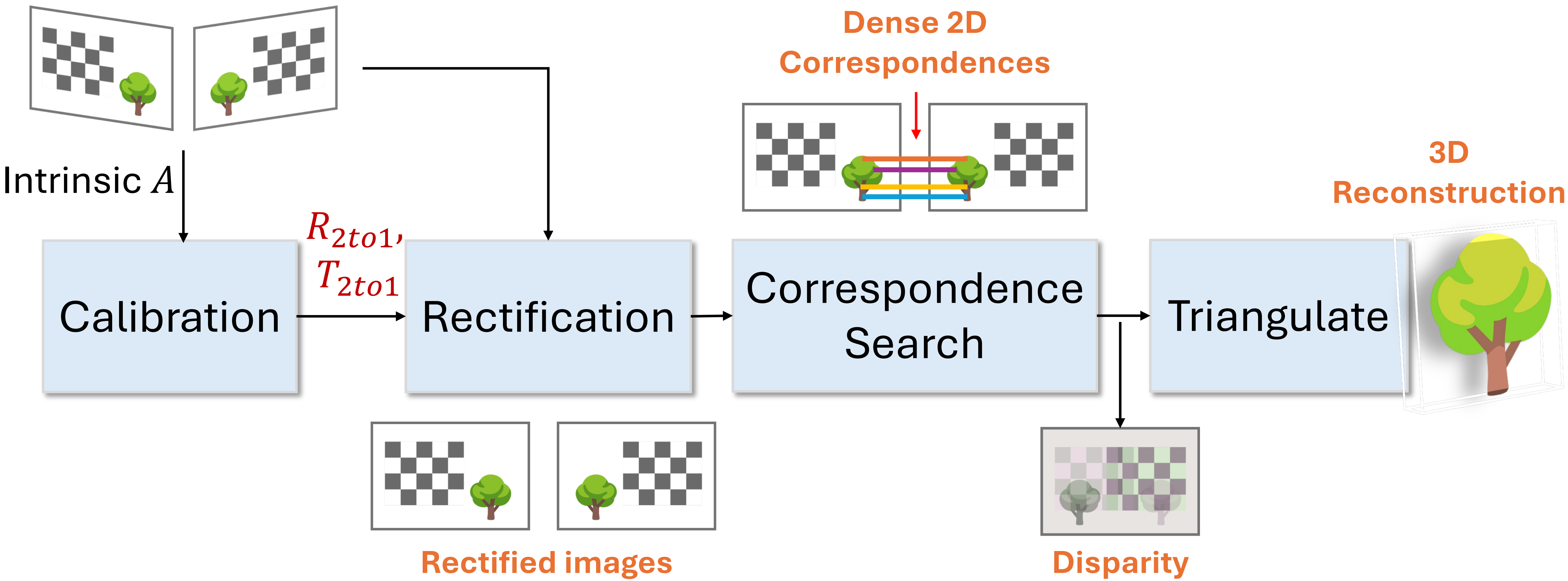
Known: Intrinsic A

OUTPUT:

1. Relative Pose: R_{2to1}, T_{2to1}
2. 3D Points $\boldsymbol{P}(X, Y, Z)$



Stereo Vision Overview



Calibration

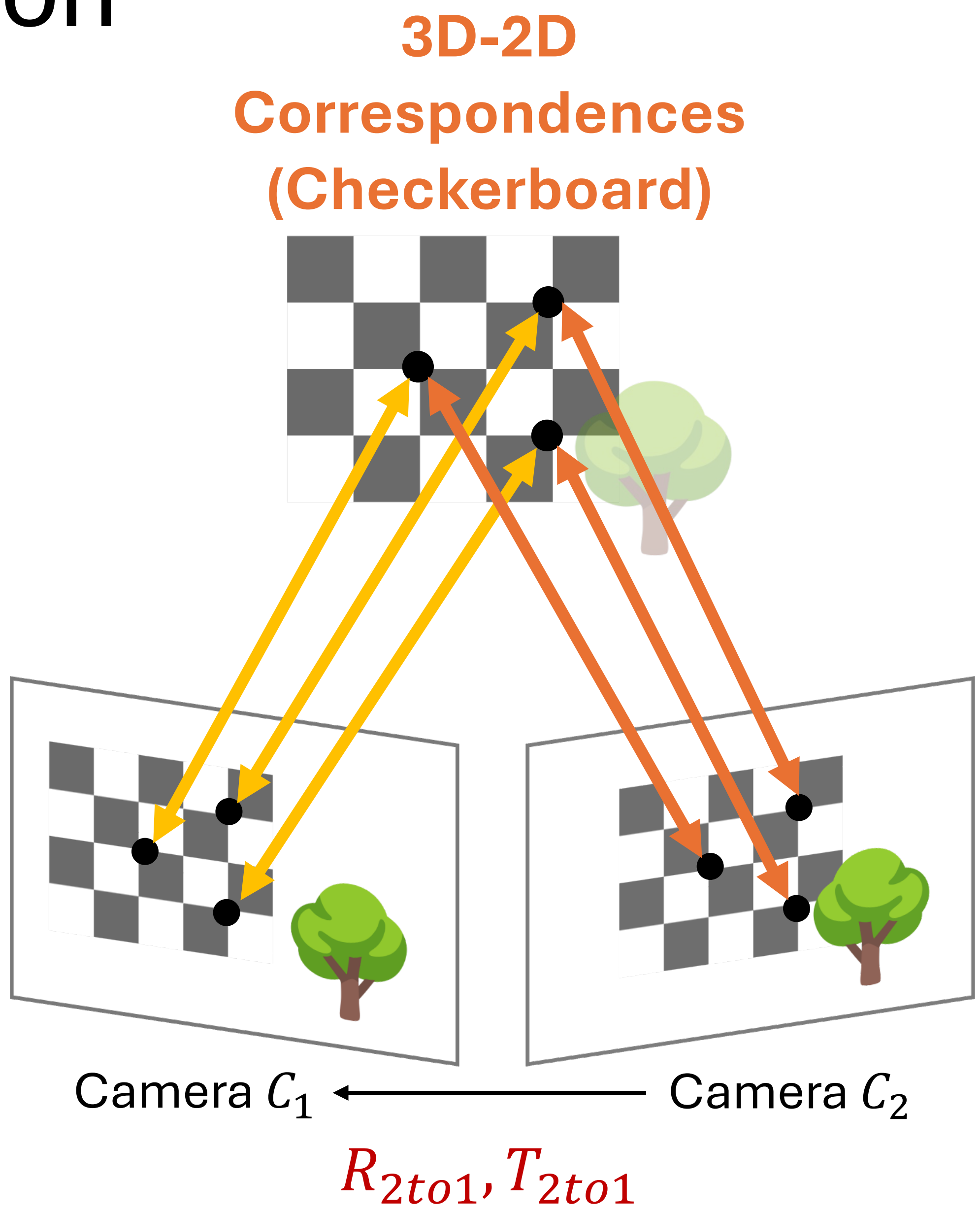
Stereo Vision – Calibration

Input:

- 3D-2D correspondences
- Intrinsic A

Output:

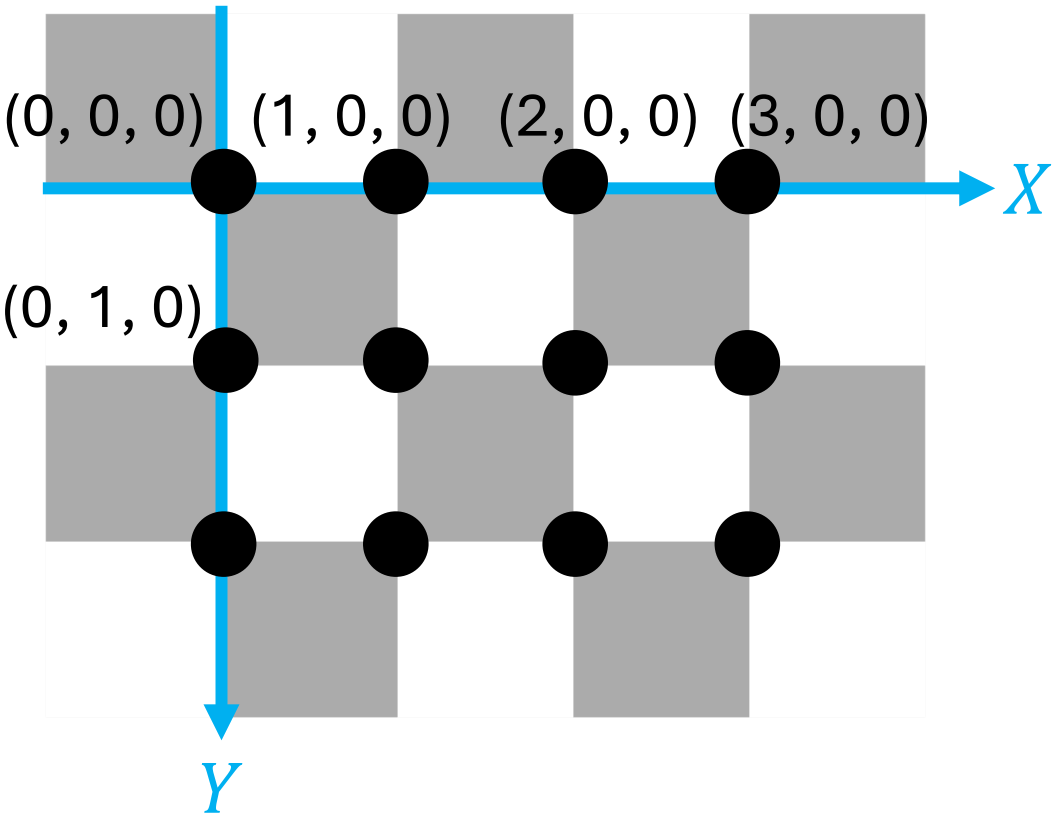
- Rotation R_{2to1}
- Translation T_{2to1}



Stereo Vision – Calibration

- As the checkerboard is flat, we set $Z = 0$

➔ the **3D (X, Y, Z) position** of corners would be: (0, 0, 0), (1, 0, 0), (2, 0, 0)…



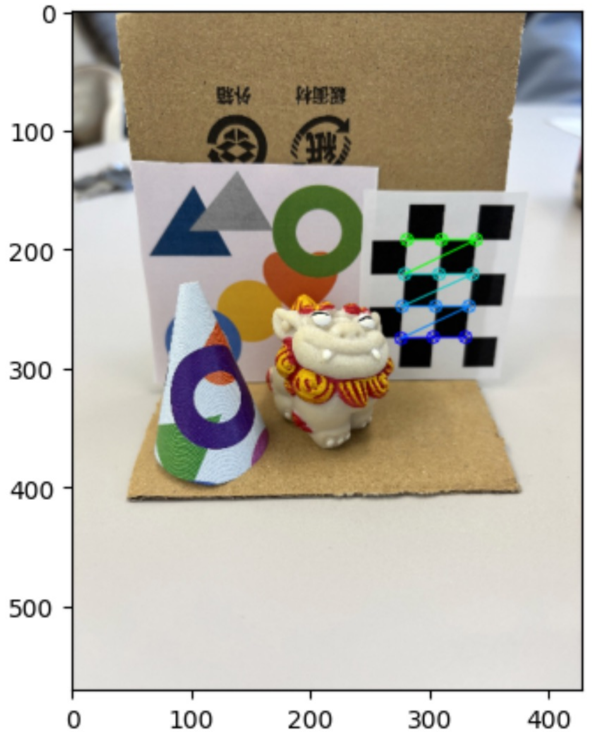
Stereo Vision – Calibration

- **2D (u, v) corner positions** in Camera images can be detected using OpenCV functions cv2.findChessboardCorners() and cv2.cornerSubPix():

```
# termination criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# Find 2D corner positions
# gray: gray scale image, corners_shape: for example -> (3, 4)
ret, corners = cv2.findChessboardCorners(gray, corners_shape, None)

if ret == True:
  # Refine detected corner positions
  corners2 = cv2.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)
```



Stereo Vision – Calibration

Given 3D-2D checkerboard corner correspondences of Camera $\boldsymbol{C_1}$

➔ You can use OpenCV function like **cv2.solvePnP()** to estimate the relative pose R_{wto1}, T_{wto1}:

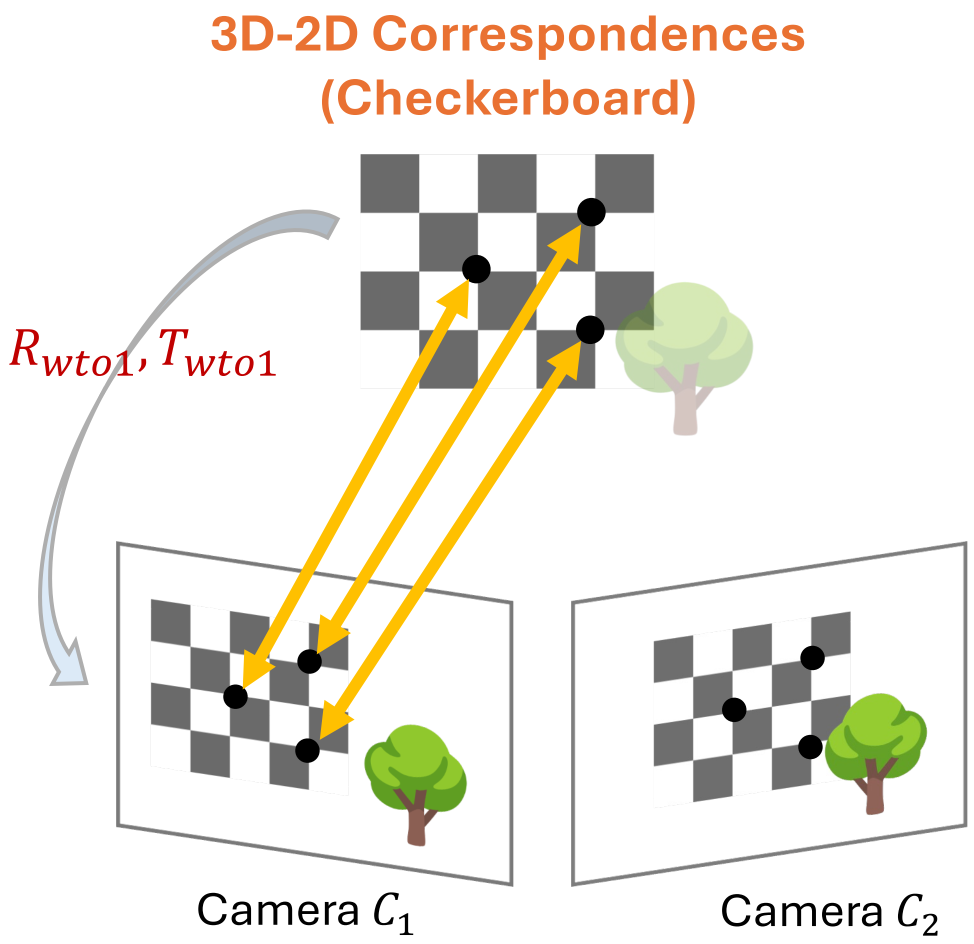
3D Corner positions

```
_, rvec_w1, tvec_w1 = cv2.solvePnP(corners3D,
                                   corners2D_in_Camera1,
                                   A, Distortion,
                                   flags=cv2.SOLVEPNP_ITERATIVE)
```

2D corner positions in Camera 1 → corners2D_in_Camera1

Intrinsic and distortion coefficients → A, Distortion

Solver option → flags=cv2.SOLVEPNP_ITERATIVE



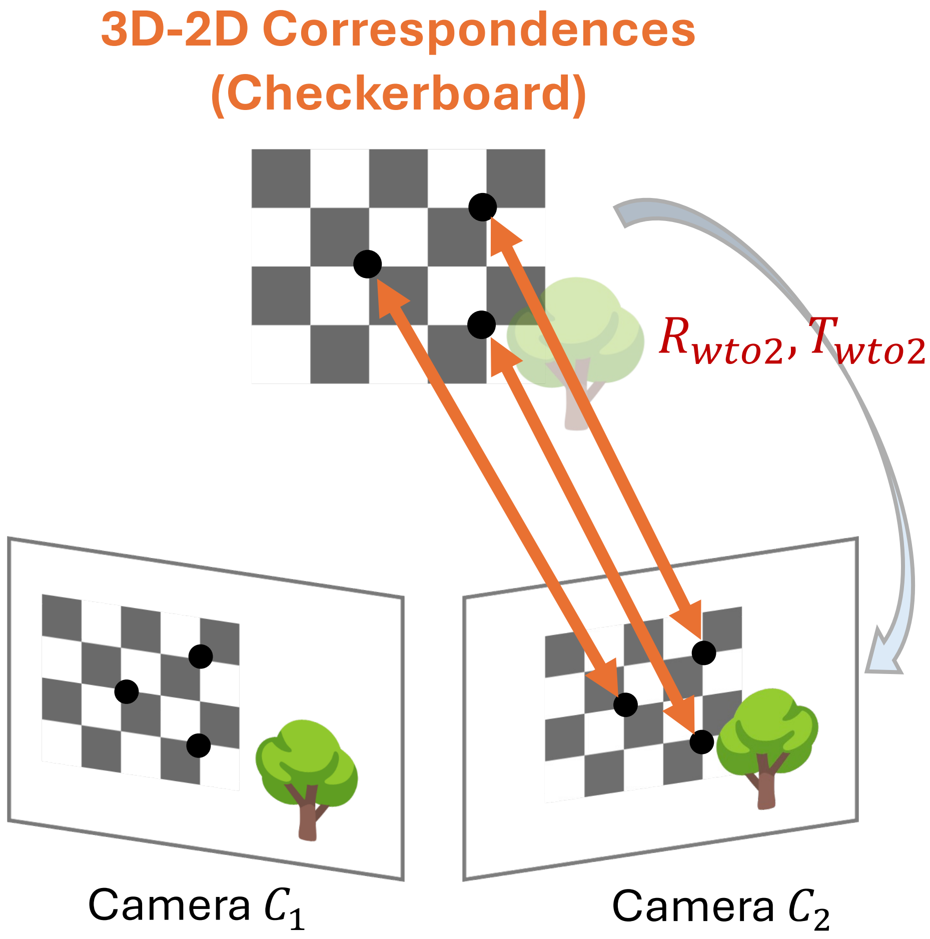
Stereo Vision – Calibration

Similarly, Given the 3D-2D correspondences of Camera $\boldsymbol{C_2}$

➔ We can estimate the relative Pose: R_{wto2}, T_{wto2}:

2D corner positions in Camera 2 →

```
_, rvec_w2, tvec_w2 = cv2.solvePnP(corners3D,
                                   corners2D_in_Camera2,
                                   A, Distortion,
                                   flags=cv2.SOLVEPNP_ITERATIVE)
```



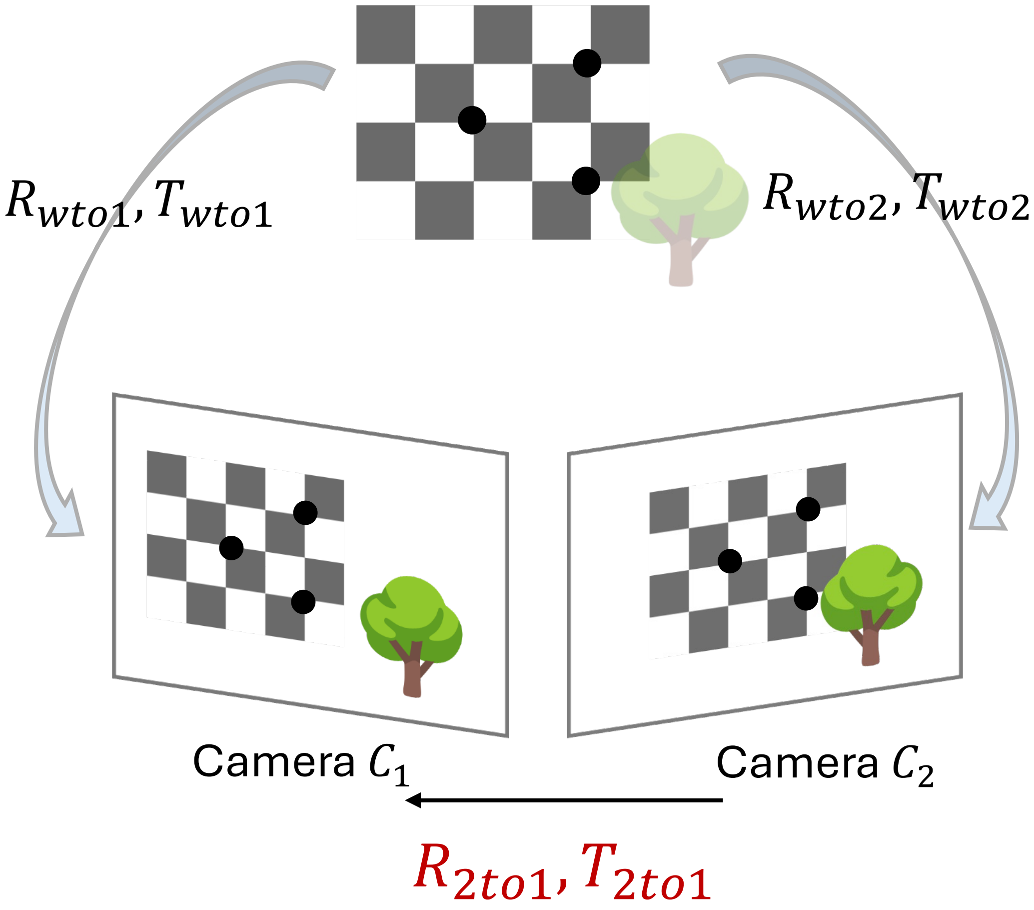
Stereo Vision – Calibration

Finally, the relative pose between

Camera C_1 & C_2 can be estimated by:

$$R_{2to1} = R_{wto1} \underset{}{\overset{\text{Dot product}}{\cdot}} R^T_{wto2}$$

$$T_{2to1} = T_{wto1} - (R_{wto1} \cdot R^T_{wto2} \cdot T_{wto2})$$



Stereo Vision – Calibration

- Please visualize the 2 camera positions and checkerboard corners in 3D using a custom function:

  - R_{wto1}, R_{wto2} in R_list
  - T_{wto1}, T_{wto2} in T_list

```
plot_cameras_with_points(R_list, T_list,
                         A, corners3D, colors='green')
```

Intrinsic parameters

3D Corner positions

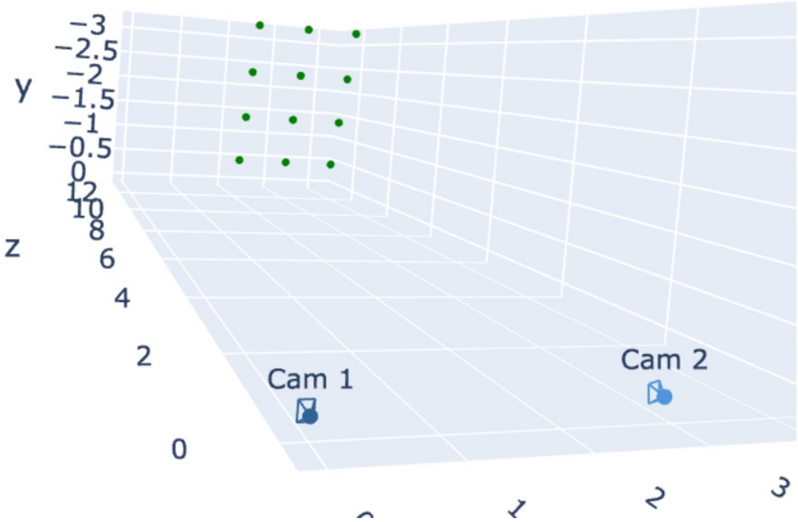


Image Rectification

Stereo Vision - Rectification

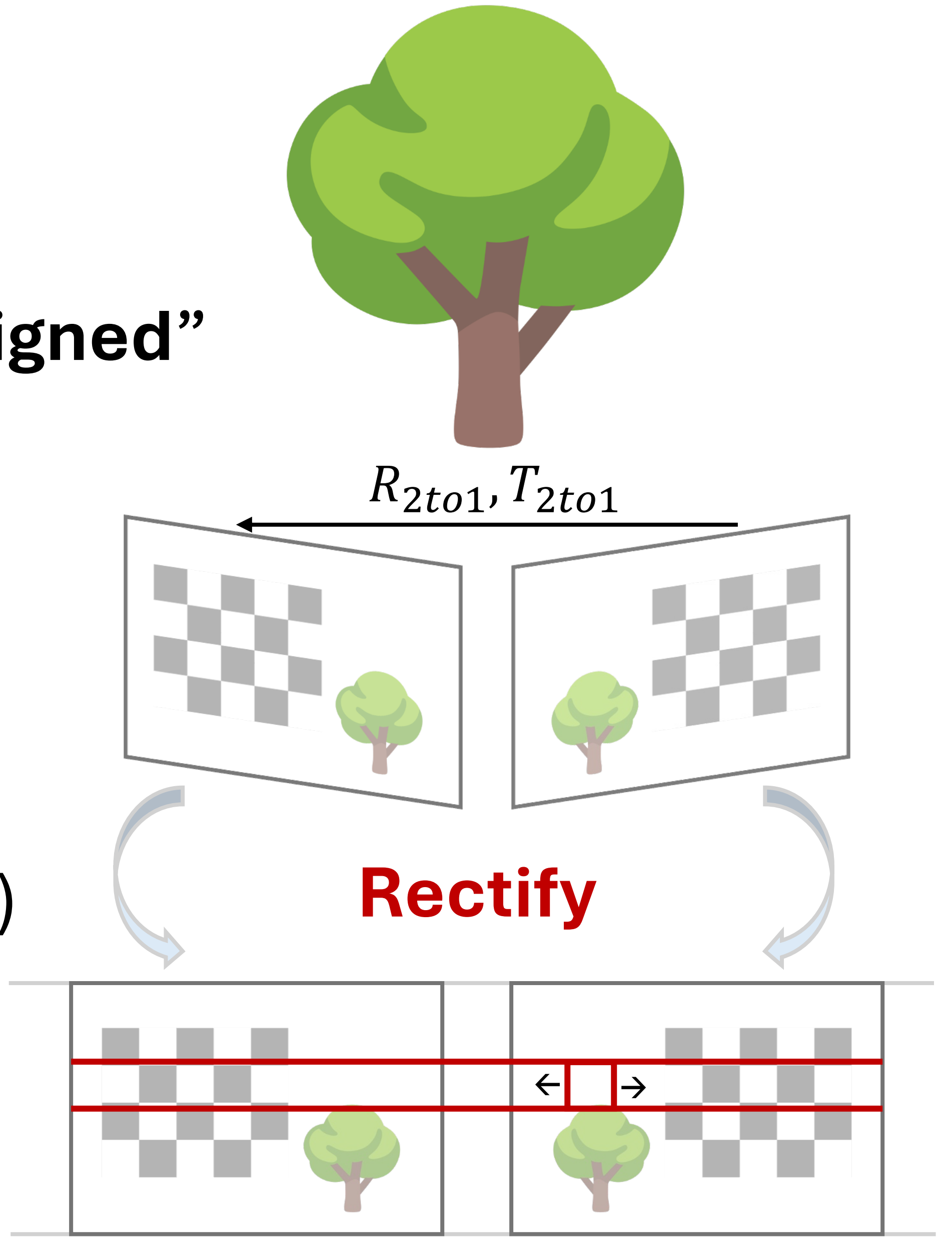
Goal: Make the two images "**horizontally aligned**"

- WHY?

  ➔ To simplify the correspondence search

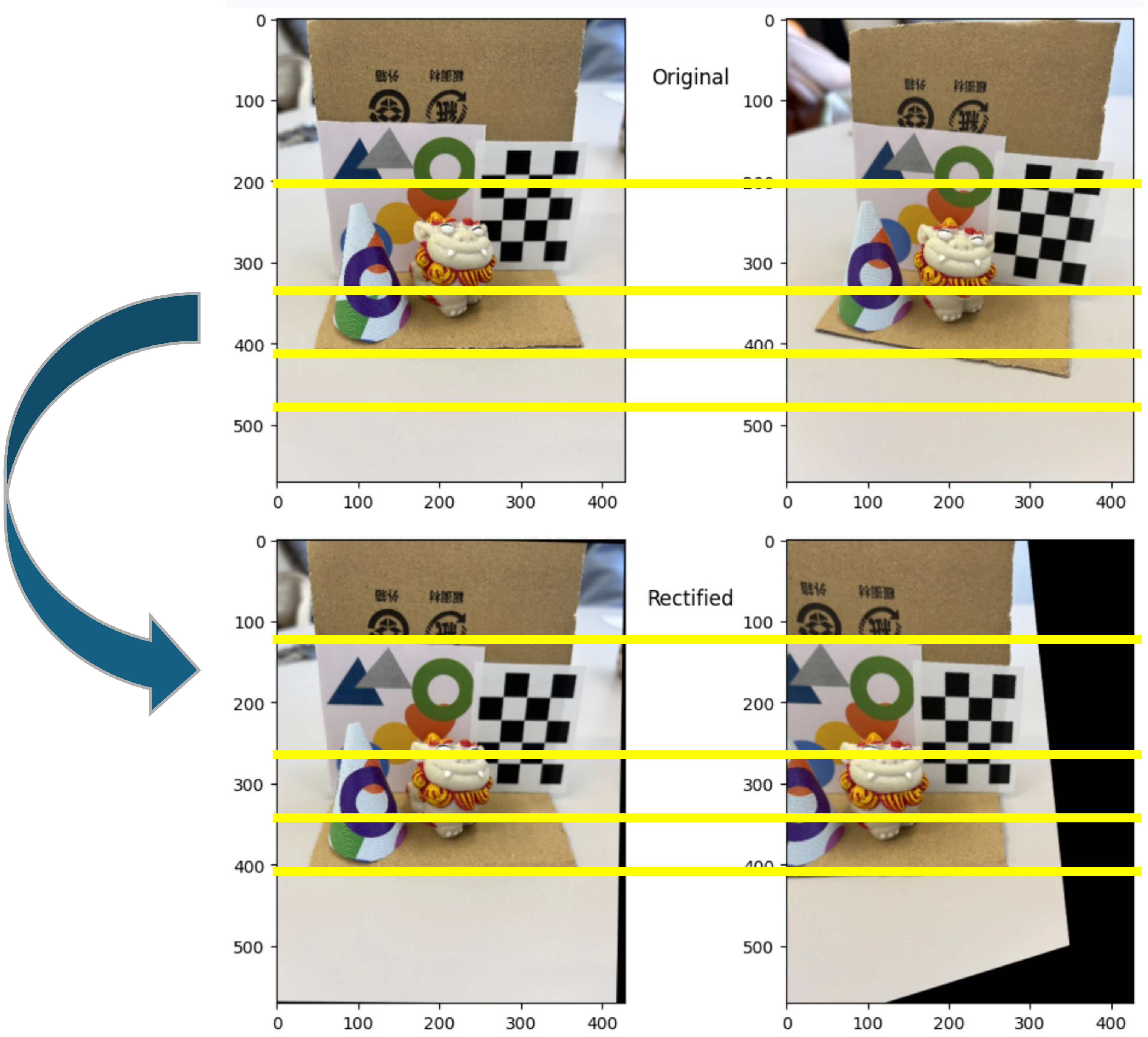
  ➔ from 2D line search to **1D search**

  (only feasible for stereo vision, i.e., 2 views)



Stereo Vision – Rectification

- Expected Results



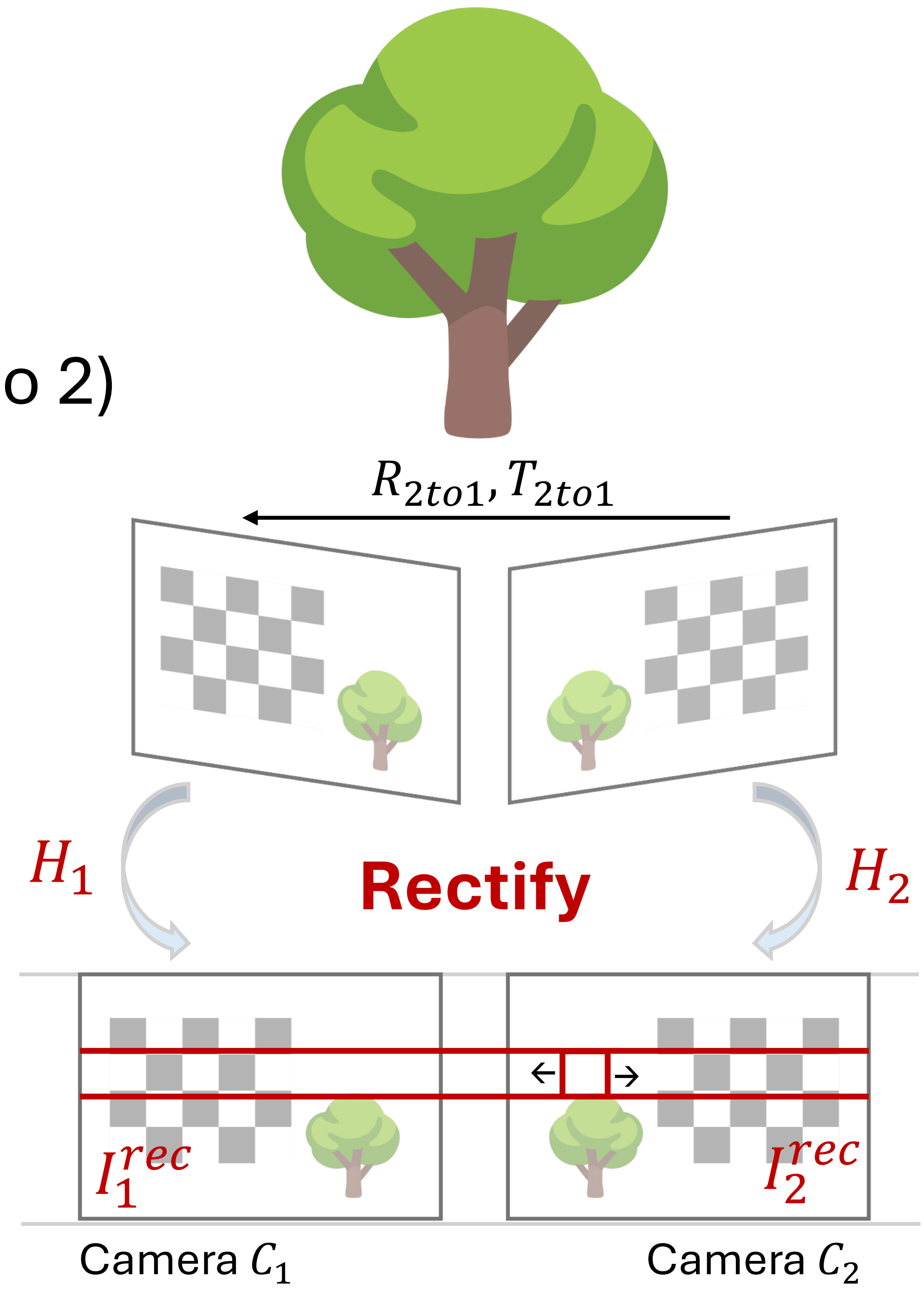
Stereo Vision - Rectification

Input: Intrinsic A, Extrinsic R, T (Camera 1 to 2)

Output:

- 3x3 Homography matrices H_1, H_2
- Rectified Images I_1^{rec}, I_2^{rec}

➔ H_1, H_2 are used to project original images to **virtual image** $I_{\{1,2\}}^{rec}$ (parallel to the baseline)



"Homography": the matrix used to map points between **2D** planes

Stereo Vision - Rectification

- Step 1: Define the New Common Camera Frame R_{rec}

➔ New X axis r_1: aligned with the baseline

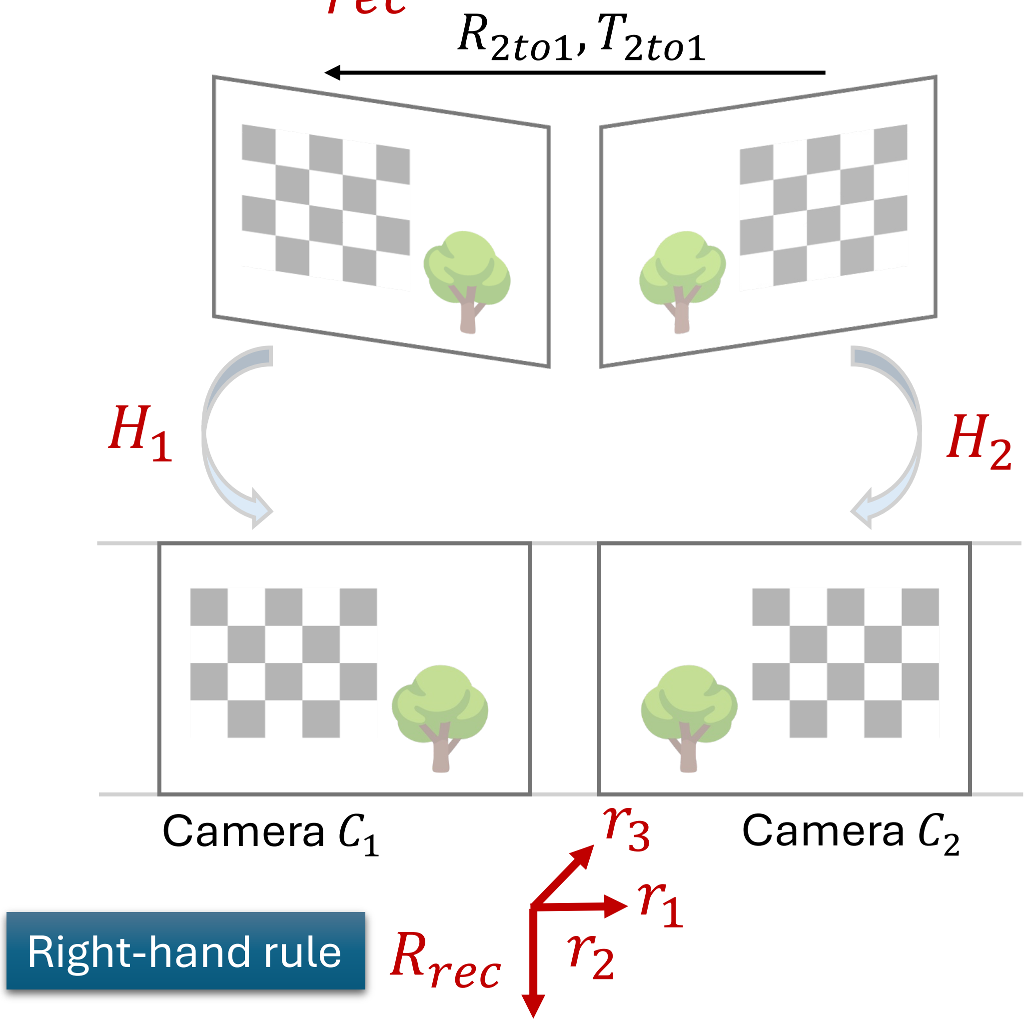
$$r_1 = \frac{T}{\|T\|}$$

➔ New Y axis r_2

$$r_2 = \frac{k \times r_1}{\|k \times r_1\|}, where\ k = [0, 0, 1]^T$$

➔ New Z axis r_3

$$r_3 = r_1 \times r_2$$

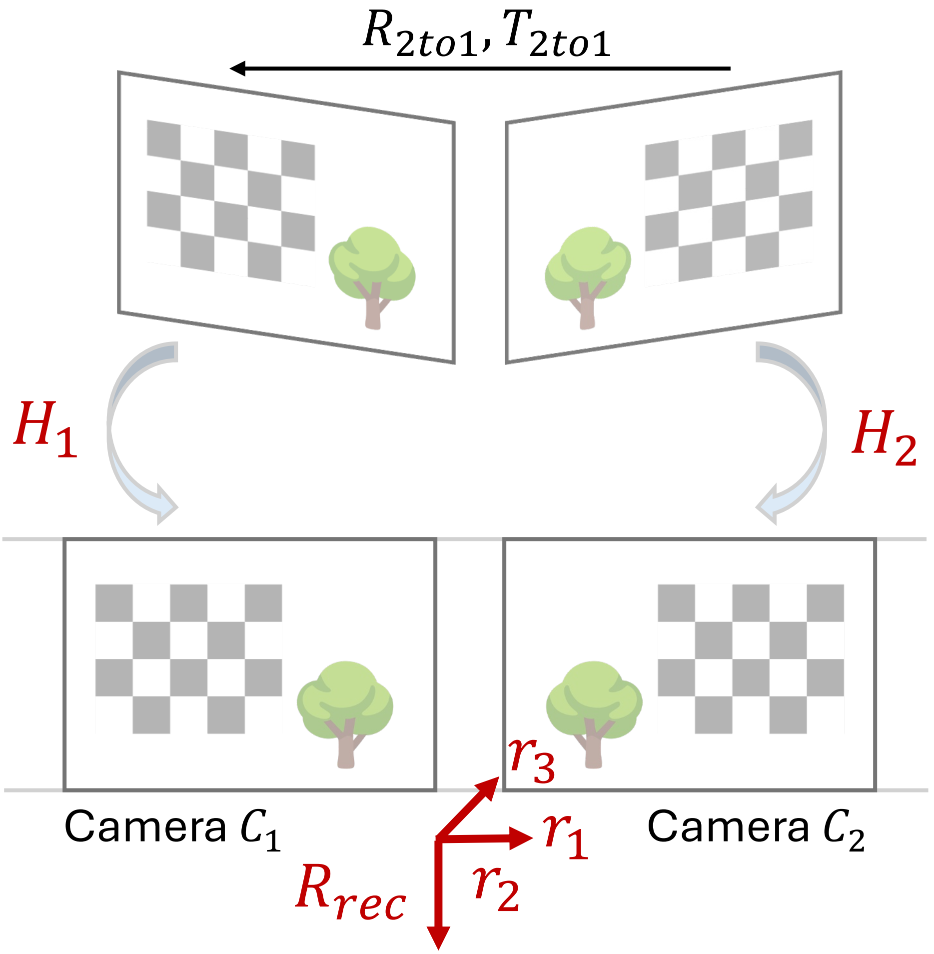


Stereo Vision - Rectification

- Step 1: Define the New Common Camera Frame R_{rec}

➔ Given new axes r_1, r_2, r_3

$$R_{rec} = \begin{pmatrix} r_1^T \\ r_2^T \\ r_3^T \end{pmatrix} = \begin{pmatrix} r_{1x} & r_{1y} & r_{1z} \\ r_{2x} & r_{2y} & r_{2z} \\ r_{3x} & r_{3y} & r_{3z} \end{pmatrix}$$



Stereo Vision – Rectification

- Step 2: Compute the Homographies H_1, H_2

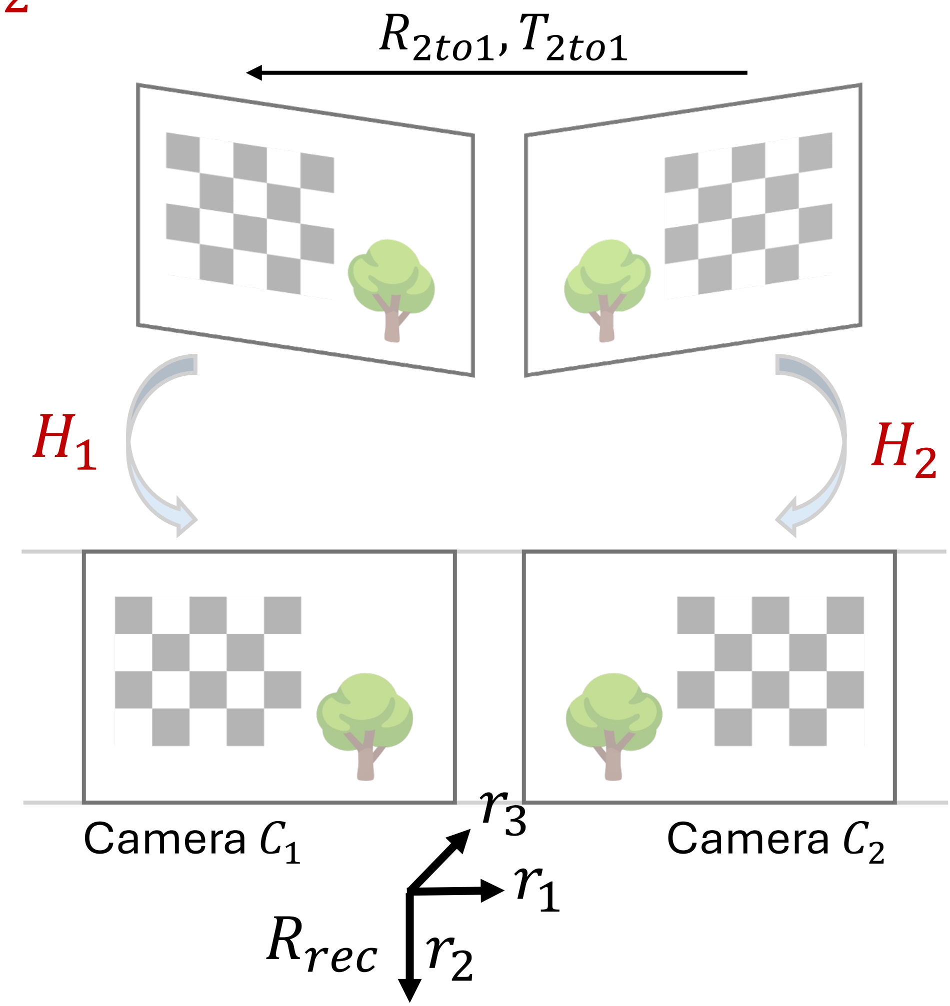
  ➔ Set Camera 1 as the origin

  - Homography H_1 for Camera 1:

$$H_1 = \underbrace{A}_{\text{Project to virtual image plane}} \cdot \underbrace{R_{rec}}_{\text{Transform to virtual space}} \cdot \underbrace{A^{-1}}_{\text{Transform to camera space}}$$

  - Homography H_2 for Camera 2:

$$H_2 = A \cdot R_{rec} \cdot \underbrace{R_{2to1}}_{\text{Transform to Camera 1 space}} \cdot A^{-1}$$

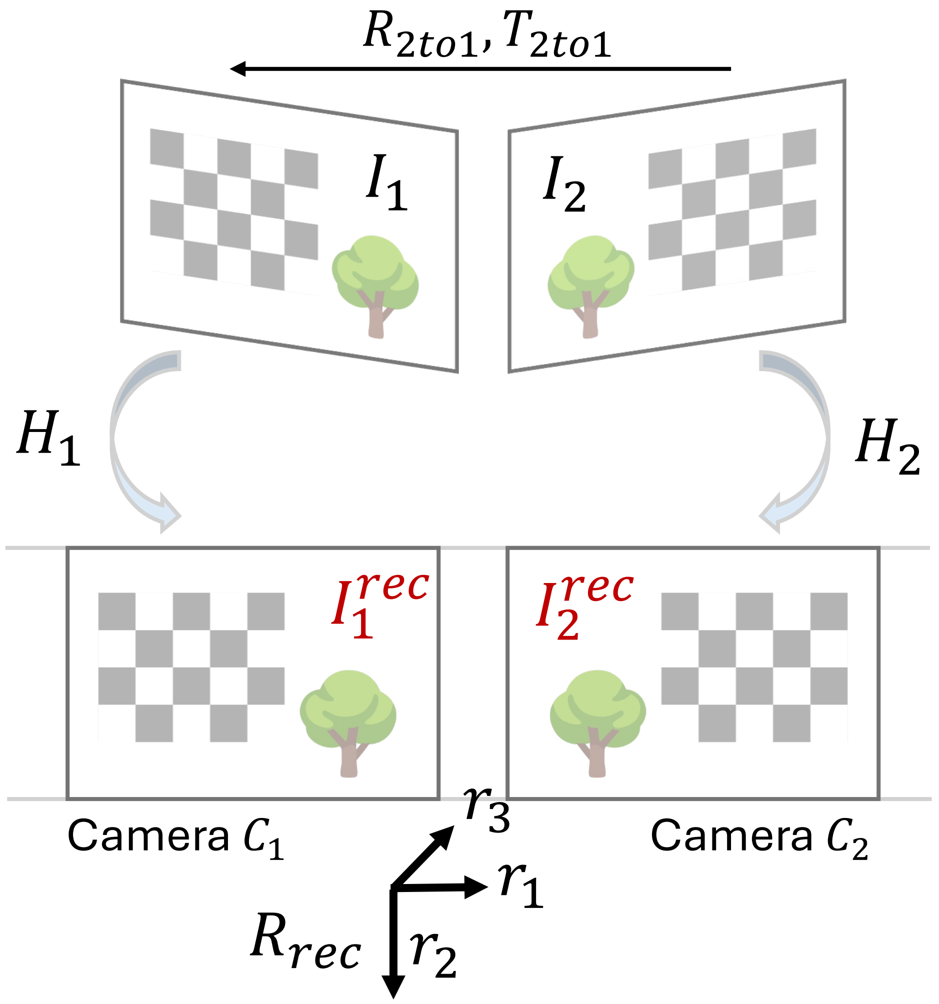


Stereo Vision – Rectification

- Step 3: Warp the Images

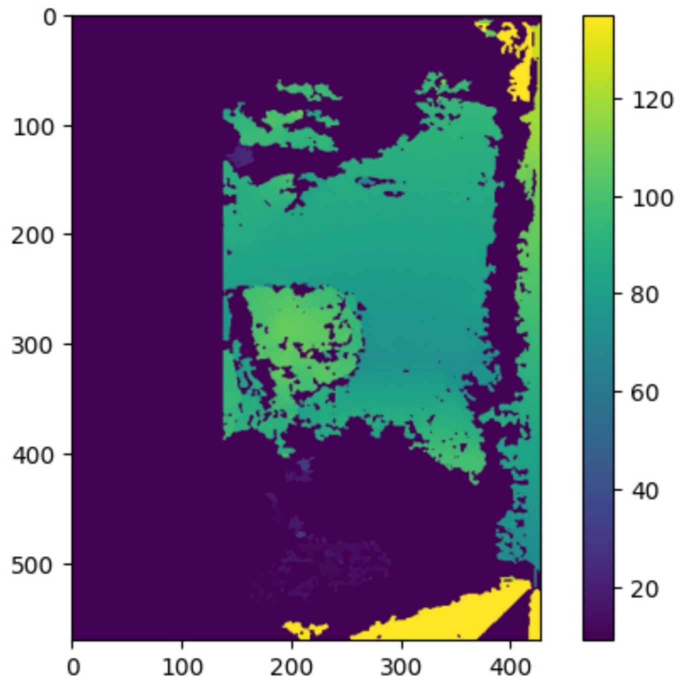
  ➔ You can use OpenCV function like **cv2.warpPerspective()** to do so:

```
cv2.warpPerspective(src=img,        ← Original image
                    M=H[i],         ← Homography matrix
                    dsize=dsize_wh, ← Image size
                    flags=cv2.INTER_LINEAR)
                          ↑
                  Bilinear interpolation
```



Correspondence Search & Triangulation

Stereo Vision –Correspondence Search



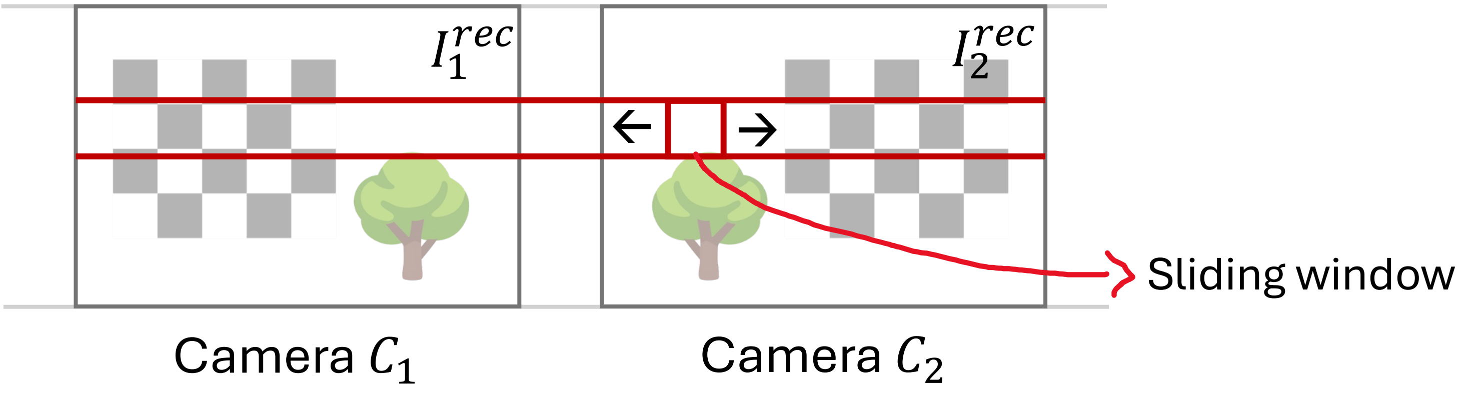
- Algorithm example: **Semi-Global Matching (SGM)**

INPUT: Rectified image pair I_1^{rec}, I_2^{rec}

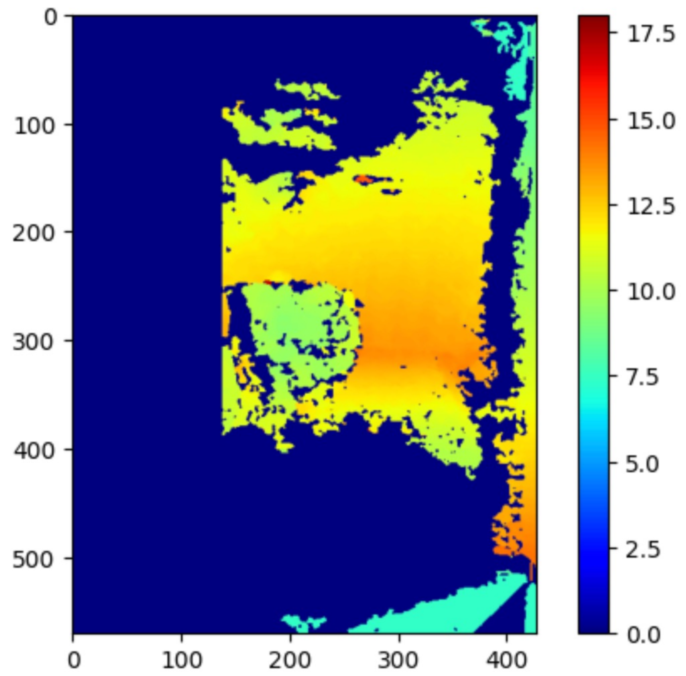
OUTPUT: Disparity Map

→ Dense 2D Correspondences $(u_{rec}, v_{rec}, 1)_{C_1} \leftrightarrow (u_{rec}, v_{rec}, 1)_{C_2}$

Basic Idea: Measure how well the local image area around a pixel in I_1^{rec} **matches** the corresponding area in I_2^{rec} → **Compute the photo-consistency!**



Stereo Vision – Triangulation



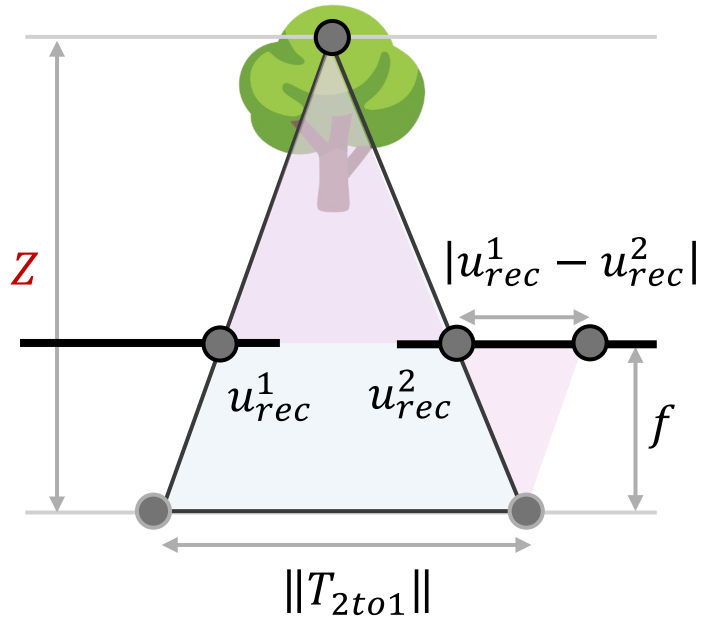
INPUT

- Disparity Map: $|u_{rec}^1 - u_{rec}^2|$
- Translation T_{2to1}, focal length f (in pixel)

OUTPUT

- Depth Map Z

- **Method**
  - Disparity-to-Depth: depth $Z = \dfrac{f * \|T_{2to1}\|}{|u_{rec}^1 - u_{rec}^2|}$



Stereo Vision – Depth to 3D Point Cloud

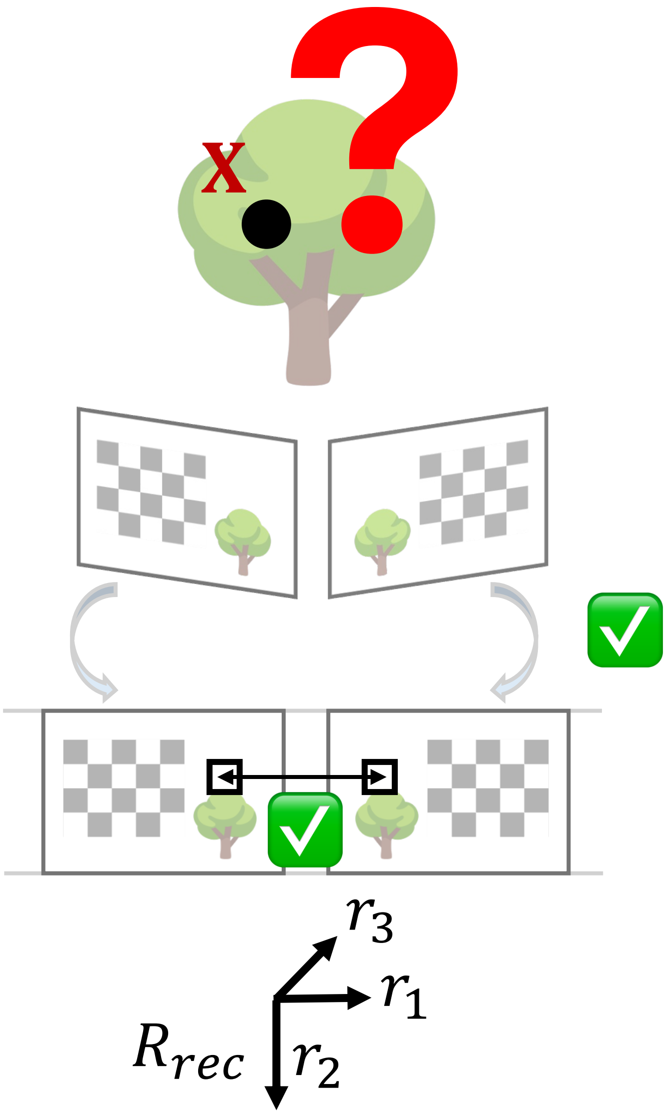
INPUT:

- Depth Map Z
- Intrinsic A, R_{rec}, $(u_{rec}, v_{rec}, 1)_{C_1}$

OUTPUT: 3D Point Cloud $\mathbf{X}$ (in Camera 1 space)

- **Method**

$$\mathbf{X}_{final} = \underbrace{R_{rec}^{T}}_{\text{Transform back to original camera 1 frame}} \cdot \underbrace{\left(Z \cdot A^{-1} \cdot \begin{pmatrix} u_{rec} \\ v_{rec} \\ 1 \end{pmatrix}_{C_1} \right)}_{\text{3D points in rectified camera 1 frame}}$$



That’s it. Good luck.