3D Scanning & Motion Capture

Exercise - 1

Felix Altenberger, Andrei Burov

Exercise 0 - Outcomes

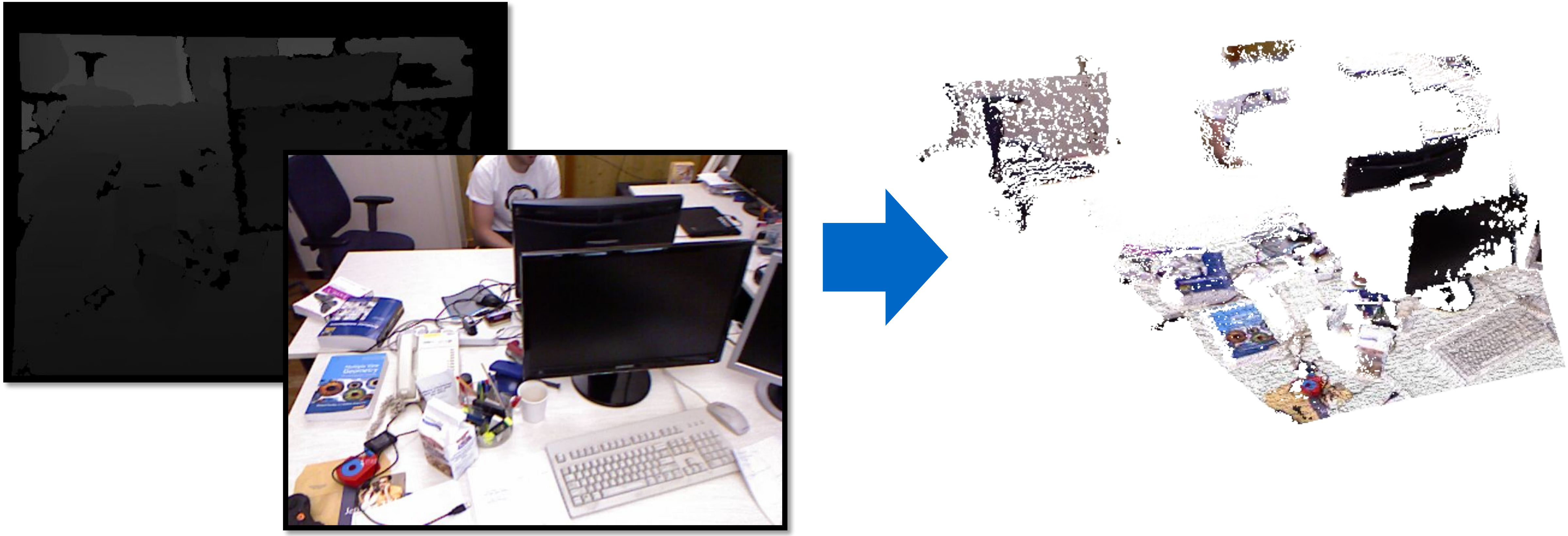
1. We’ve received 30% of expected submissions
2. Pitfalls (build output instead of app output)
3. Reach out on Moodle!
4. We hope you tested your working environment now :)
5. We will publish a PDF with matriculation numbers of successful submissions

Exercises – Overview

1. **Exercise → Camera Intrinsics, Back-projection, Meshes**
2. Exercise → Surface Representations
3. Exercise → Coarse Alignment (Procrustes)
4. Exercise → Optimization
5. Exercise → Object Alignment, ICP

Exercise 1

1. **Exercise → Camera Intrinsics, Back-projection, Meshes**



TUM-RGB-D SLAM Dataset

- https://vision.in.tum.de/data/datasets/rgbd-dataset
- 39 sequences
- Recorded using Kinect v.1
  - Structured light
  - Calibrated
  - Aligned depth and color maps
- Camera trajectory

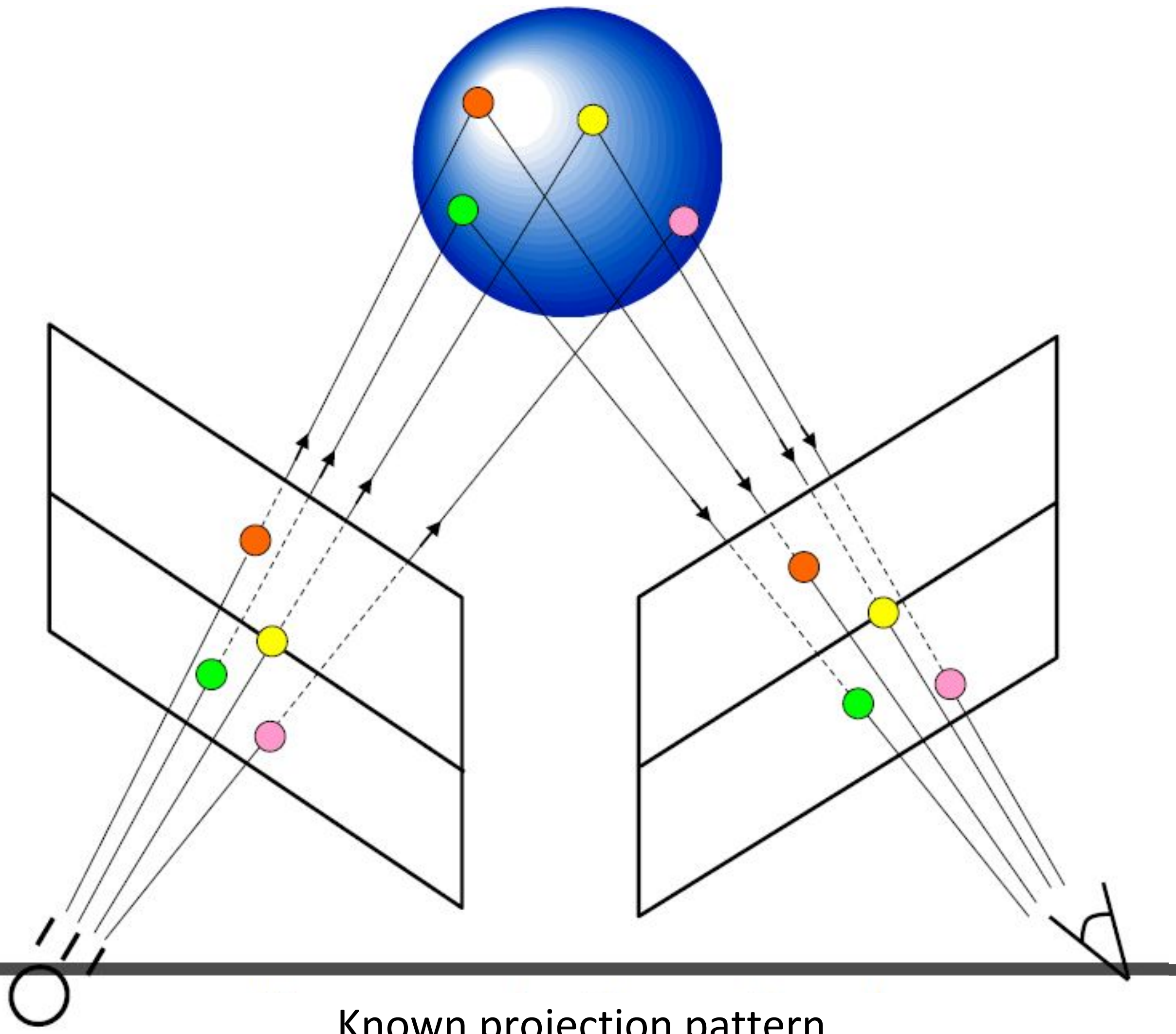



TUM-RGB-D SLAM Dataset

Scene: “fr1/xyz”

Kinect v.1 – Depth and Color Information

Kinect v.1 – Structured Light



Known projection pattern

Tasks

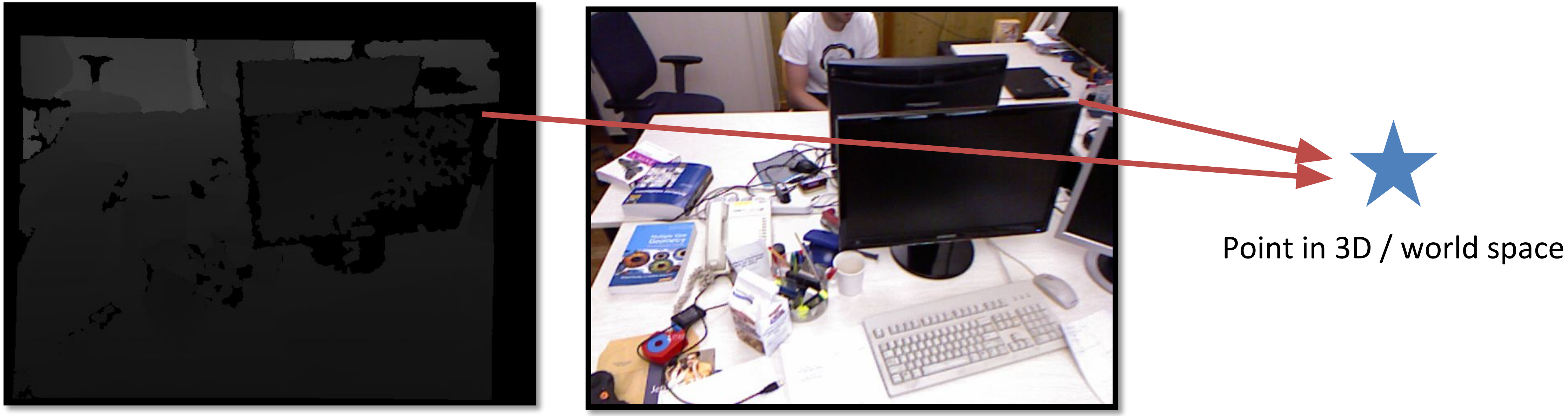
1. Project dependencies & CMake configuration
2. Back-Projection
   - Use the given intrinsics, extrinsics and the camera trajectory to project the camera observation back to world space
   - Assign the color to the back-projected points
3. Write a 3D mesh
   - Write an OFF file containing the back-projected position and color information
   - Make use of the grid structure of the observation to perform the triangulation

Task 0) Project dependencies

- Eigen http://eigen.tuxfamily.org
  - Headers-only
  - Linear Algebra library
  - Matrix, Vector, Solvers, …
  - TIP: Do not use C++'s `auto`
- FreeImage http://freeimage.sourceforge.net/
  - Support for many image formats
  - Windows: We provide a pre-compiled binary
  - Linux: `$ sudo apt-get install libfreeimage3 libfreeimage-dev`

Task 1) Back-Projection

- Use depth map, camera intrinsics and trajectory to project points from 2D → 3D.

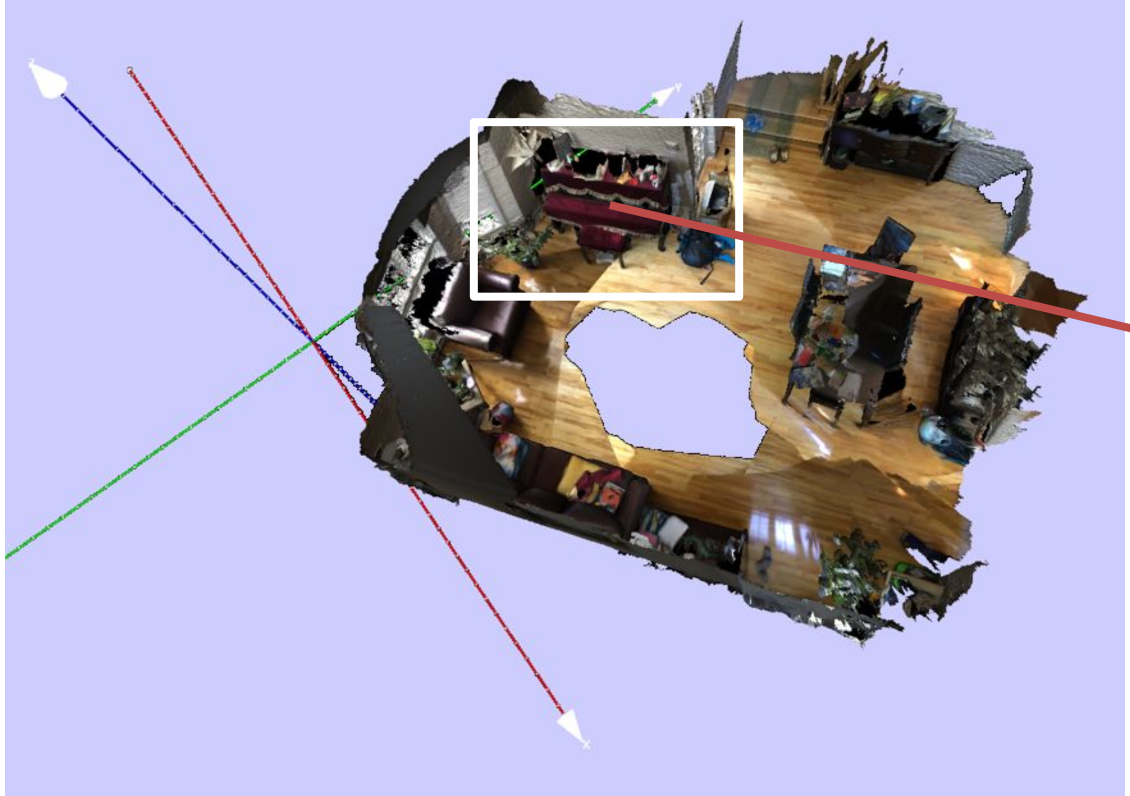


1 float / pixel (z)

4 chars / pixel (R, G, B, A)

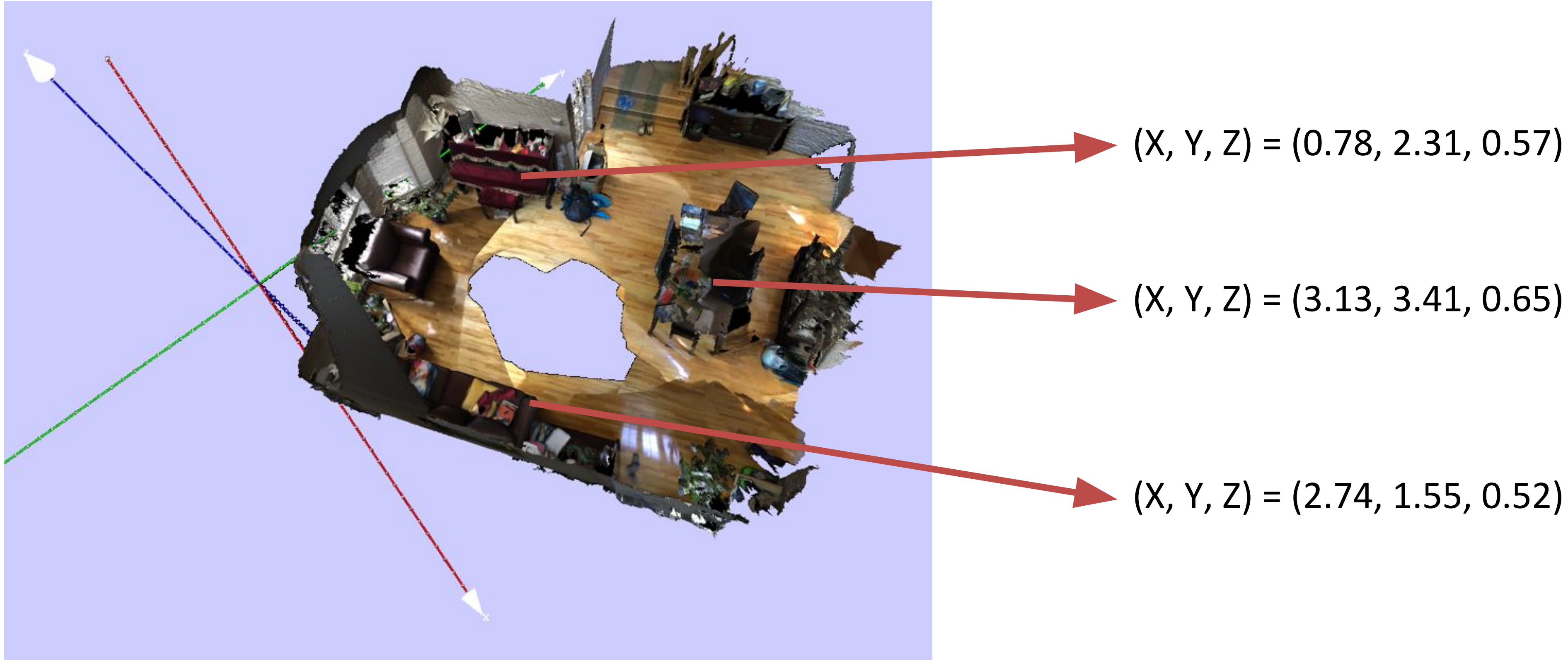
How are images synthesized?

- Given a real-world/CG scene and a camera, we want to project the 3D points in the scene to 2D pixel positions in the image



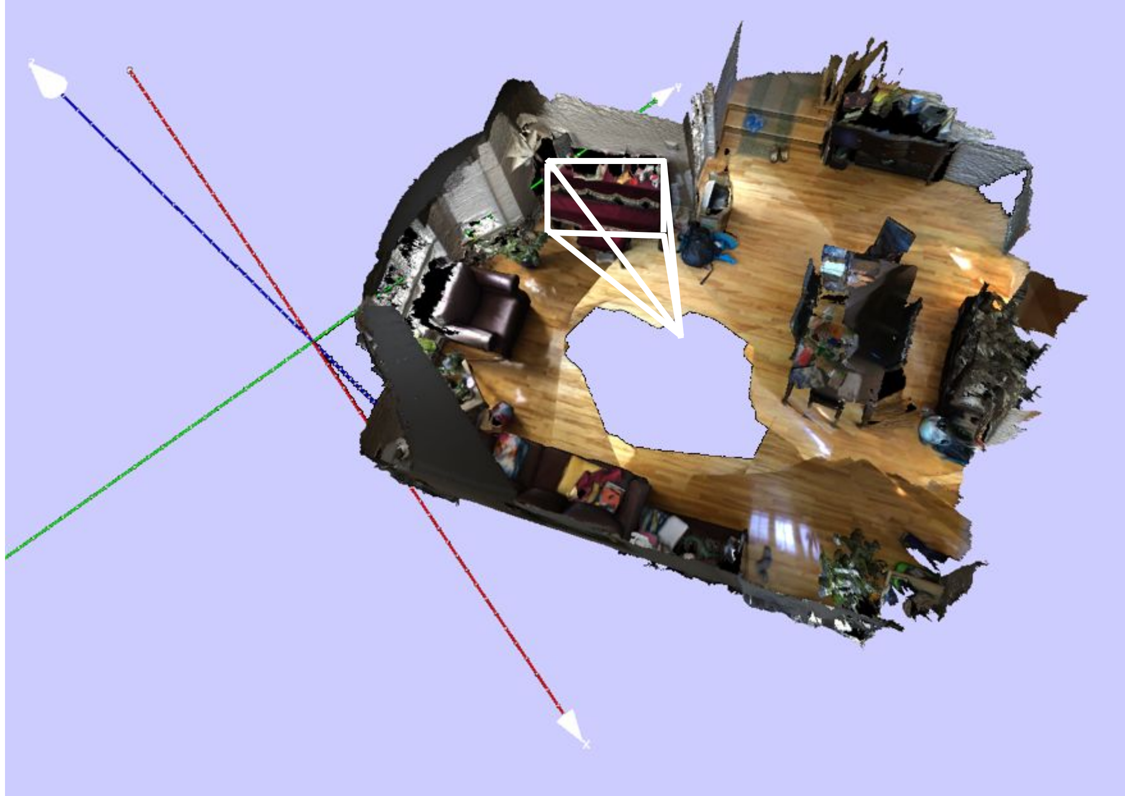
World Space

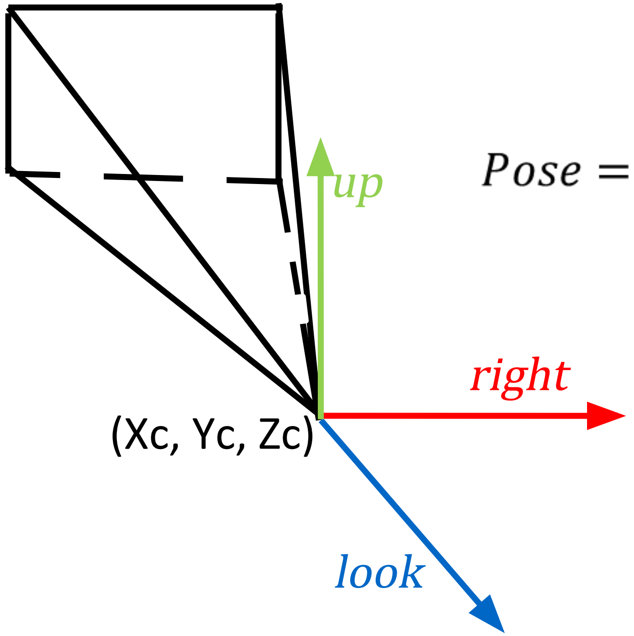
- Every point in the scene has its (X, Y, Z) coordinates



World -> Camera Space

- We place a camera at the (X, Y, Z) position in world space
- The pose/orientation is given by the right/up/look vectors

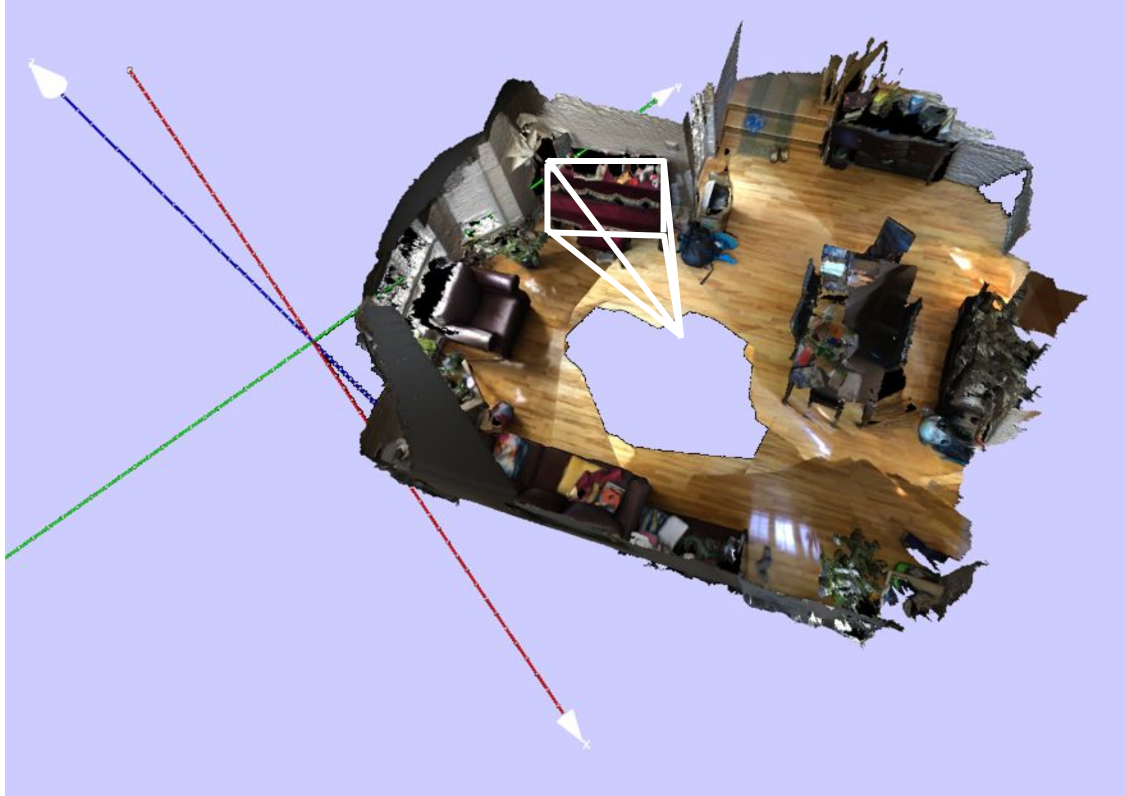




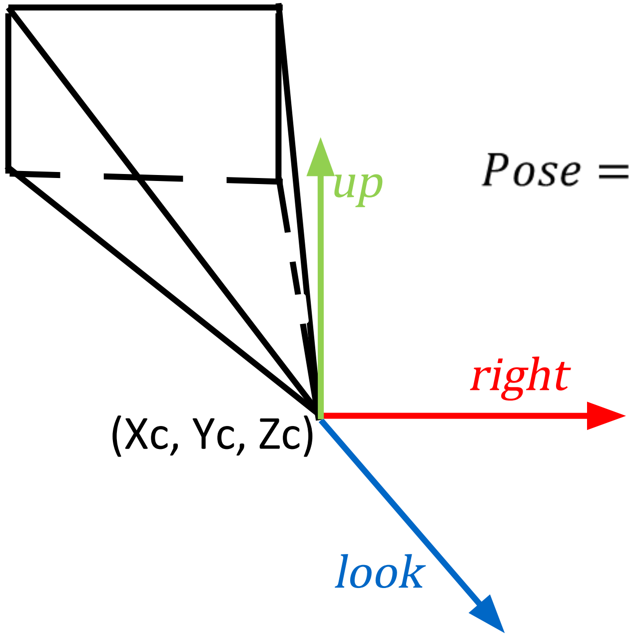
$$Pose = \begin{pmatrix} R_x & U_x & L_x & X_c \\ R_y & U_y & L_y & Y_c \\ R_z & U_z & L_z & Z_c \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

World -> Camera Space

- To transform points from world space to camera space, we need to multiply them by the camera extrinsic matrix

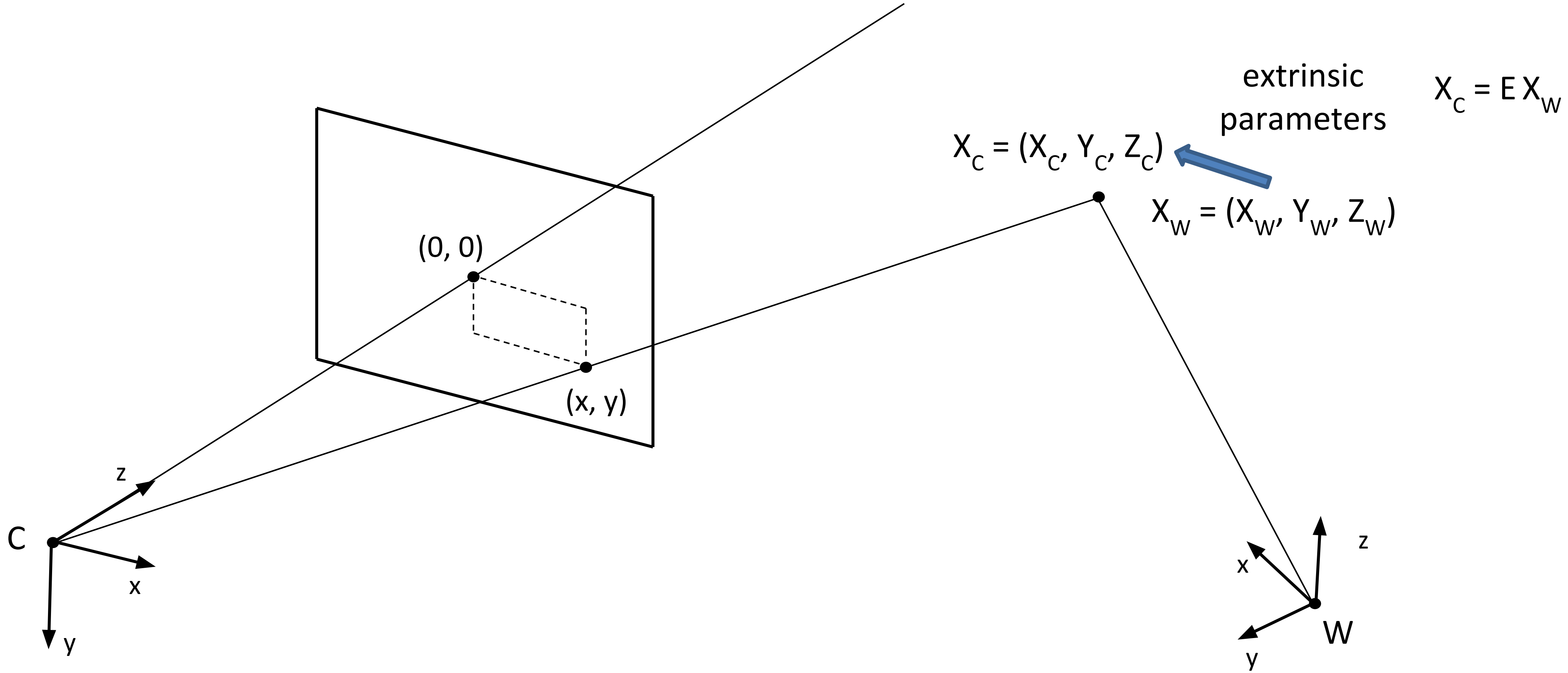


$$\boxed{Extrinsic = Pose^{-1}}$$

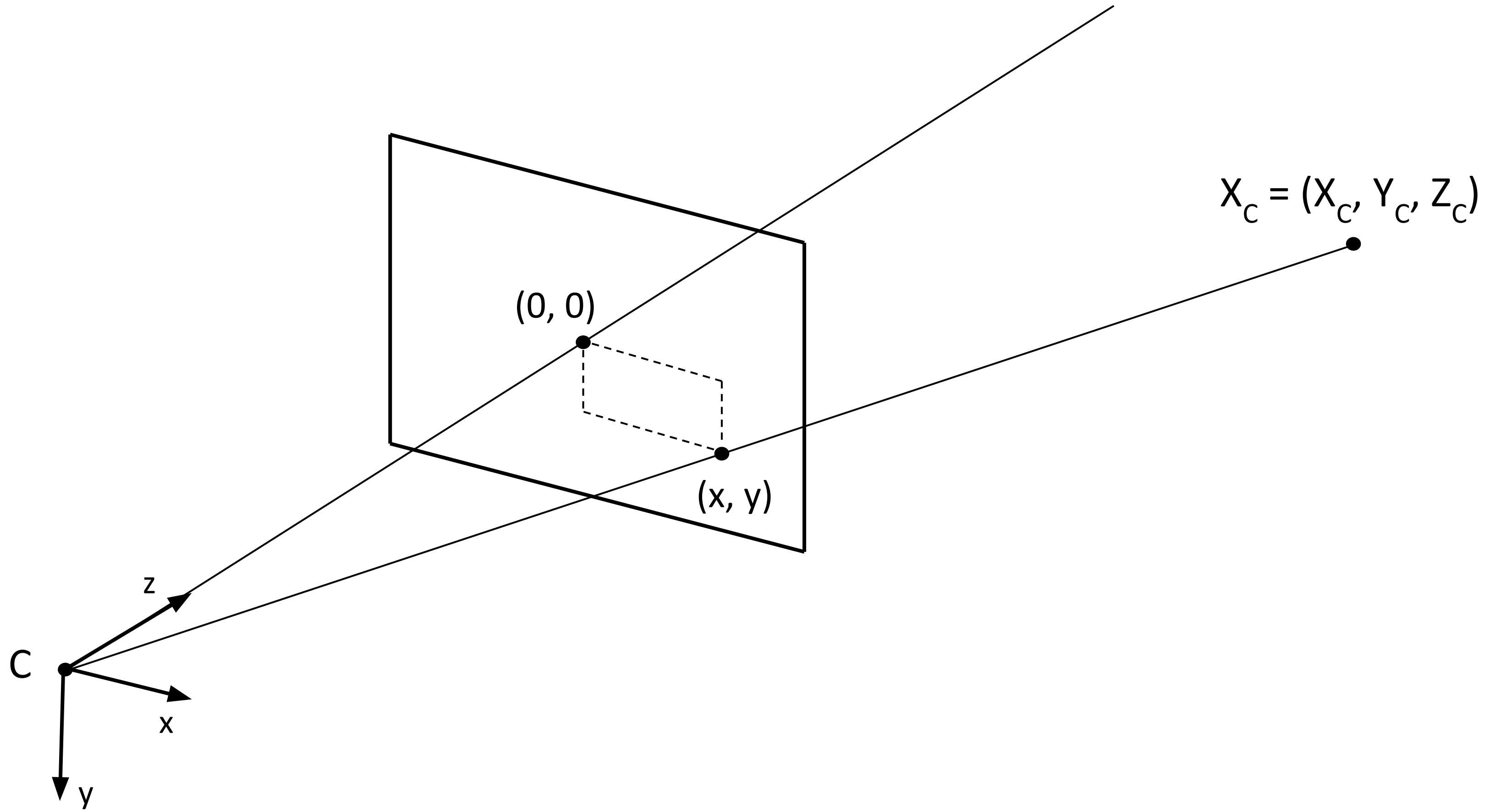


$$Pose = \begin{pmatrix} R_x & U_x & L_x & X_c \\ R_y & U_y & L_y & Y_c \\ R_z & U_z & L_z & Z_c \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

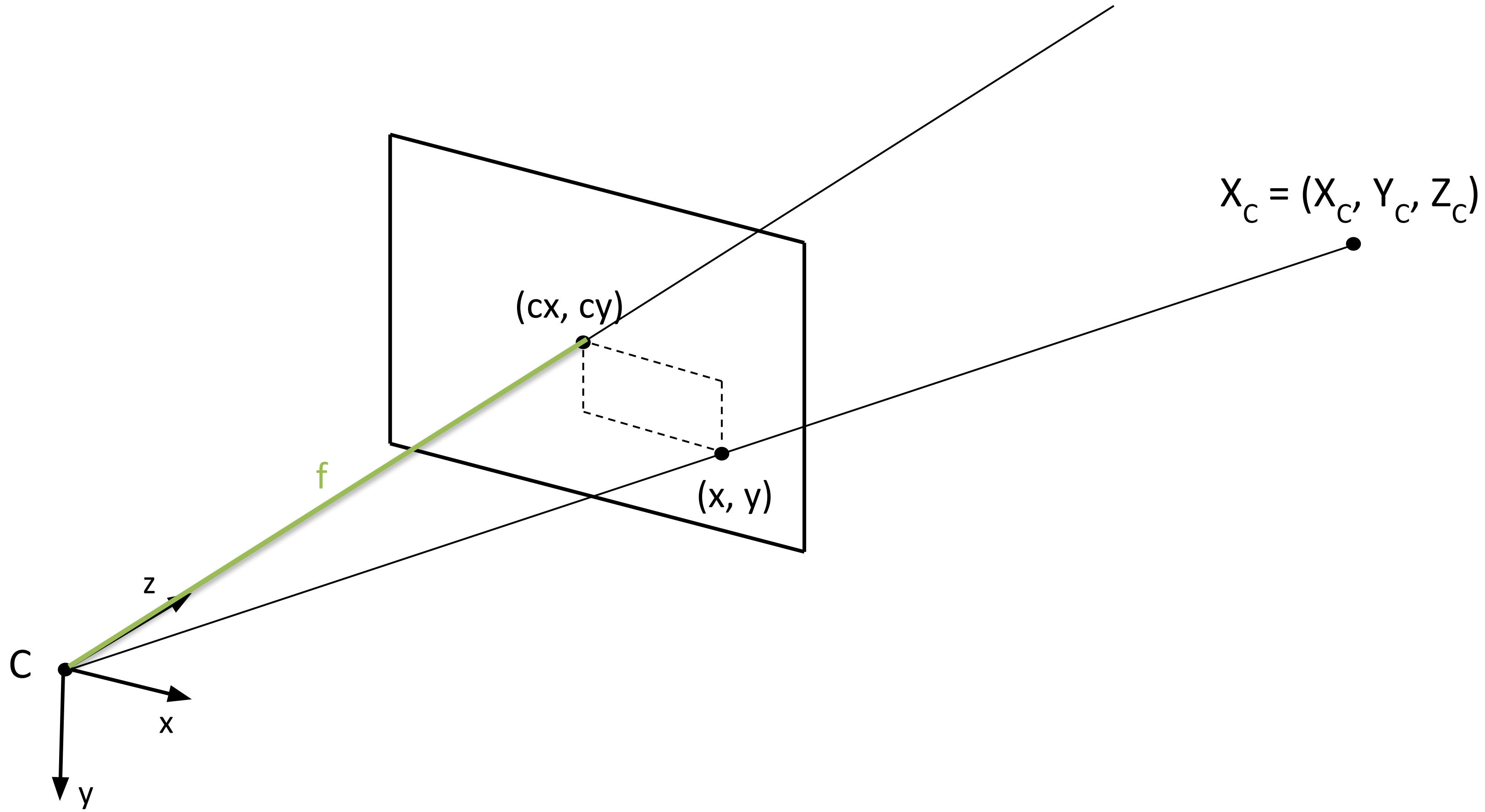
Camera Extrinsic matrix



Pinhole camera model



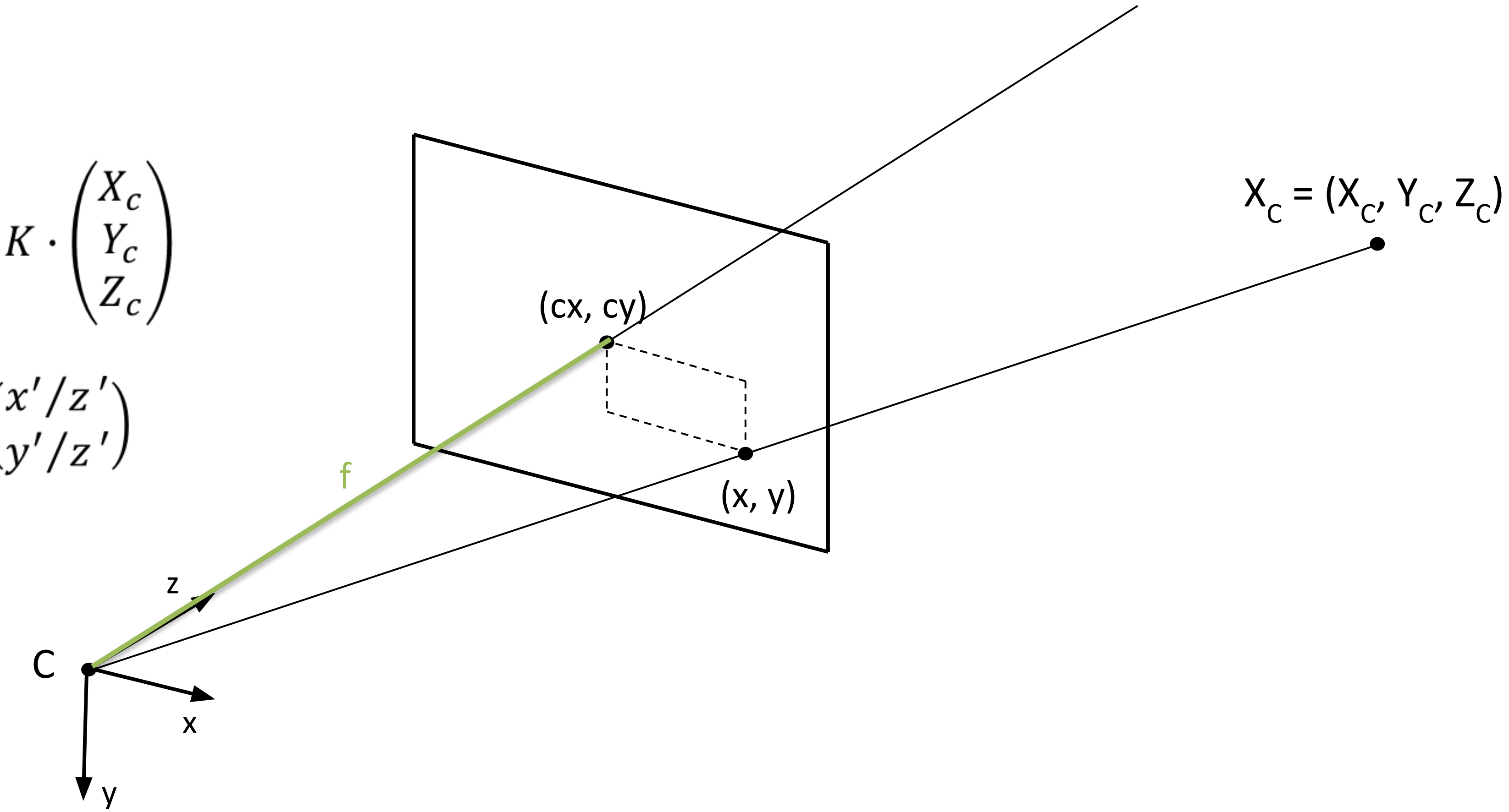
Pinhole camera model



Camera Intrinsic Matrix

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = K \cdot \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x'/z' \\ y'/z' \end{pmatrix}$$



Intrinsic matrix

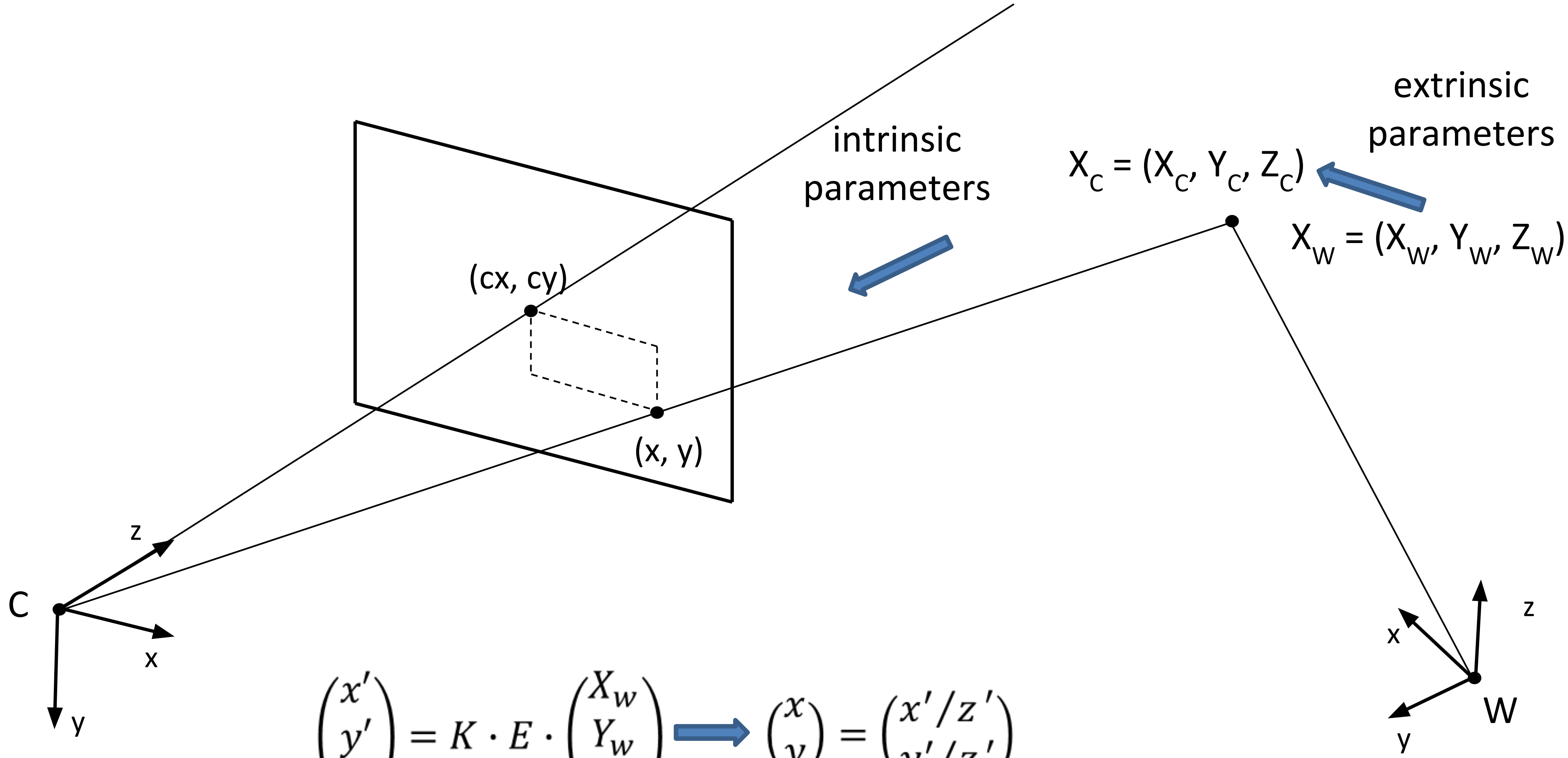
$$
\begin{aligned}
f &:= \text{focal length} = 4.1mm \\
W &:= \text{sensor width} = 4.54mm \\
H &:= \text{sensor height} = 3.42mm \\
w &:= \text{image width} = 640 \\
h &:= \text{image width} = 480 \\
c_x &:= \text{image center x} = 320 \\
c_y &:= \text{image center y} = 240
\end{aligned}
$$

$$
\text{Resulting intrisic matrix}: \begin{bmatrix} \frac{f \cdot w}{W} & 0 & c_x \\ 0 & \frac{f \cdot h}{H} & c_y \\ 0 & 0 & 1 \end{bmatrix}
$$

Perspective Projection

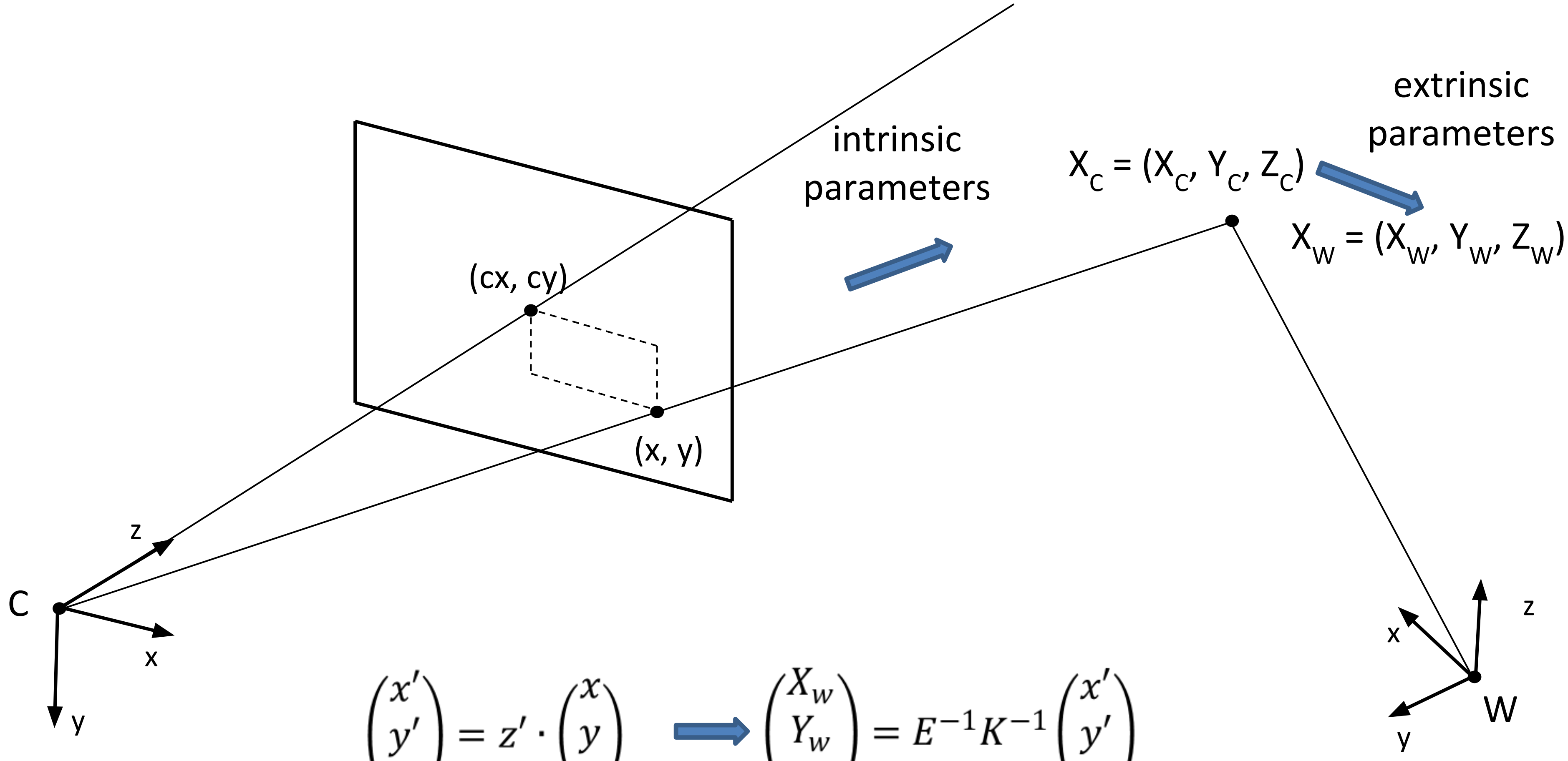
$$\begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} \xrightarrow[\textbf{Dehomogenization}]{} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} = \begin{pmatrix} x'/z' \\ y'/z' \end{pmatrix}$$

Projection Pipeline



$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = K \cdot E \cdot \begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x'/z' \\ y'/z' \end{pmatrix}$$

Projection Pipeline



$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = z' \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} = E^{-1} K^{-1} \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$$

More Than One Camera

- RGB-D Sensor like the Kinect (or your phone) have multiple cameras
- This raises the question: Which camera does the extrinsic matrix correspond to?

More Than One Camera

- Pick a point on the camera as the reference
- Each camera gets an extrinsic matrix that describes its pose relative to the reference point

Projection Pipeline (from World to Pixels)

$$p_{world} := \text{arbitary point} \in \mathbb{R}^4 = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}^T$$

$$M_{camera} := \text{camera pose} \in \mathbb{R}^{4\times4}$$

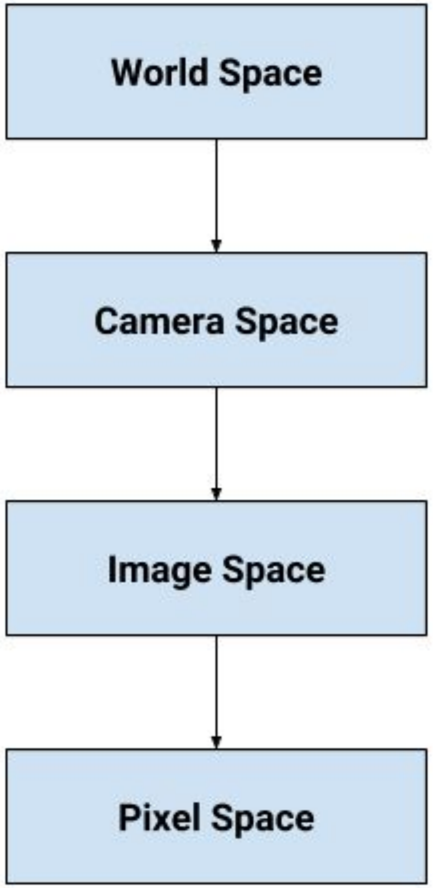
$$K := \text{intrinsic matrix} \in \mathbb{R}^{3\times3} = \begin{bmatrix} \frac{f \cdot w}{W} & 0 & c_x \\ 0 & \frac{f \cdot h}{H} & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$E := \text{extrinsic matrix} \in \mathbb{R}^{4\times4} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

$$p_{camera} = E \cdot M_{camera} \cdot p_{world} \in \mathbb{R}^4$$

$$p_{image} = K \cdot \begin{bmatrix} I_3 & 0 \end{bmatrix} \cdot p_{camera} \in \mathbb{R}^3$$

$$p_{pixel} = \frac{1}{p_{image,z}} \begin{bmatrix} p_{image,x} \\ p_{image,y} \end{bmatrix} \in \mathbb{R}^2$$

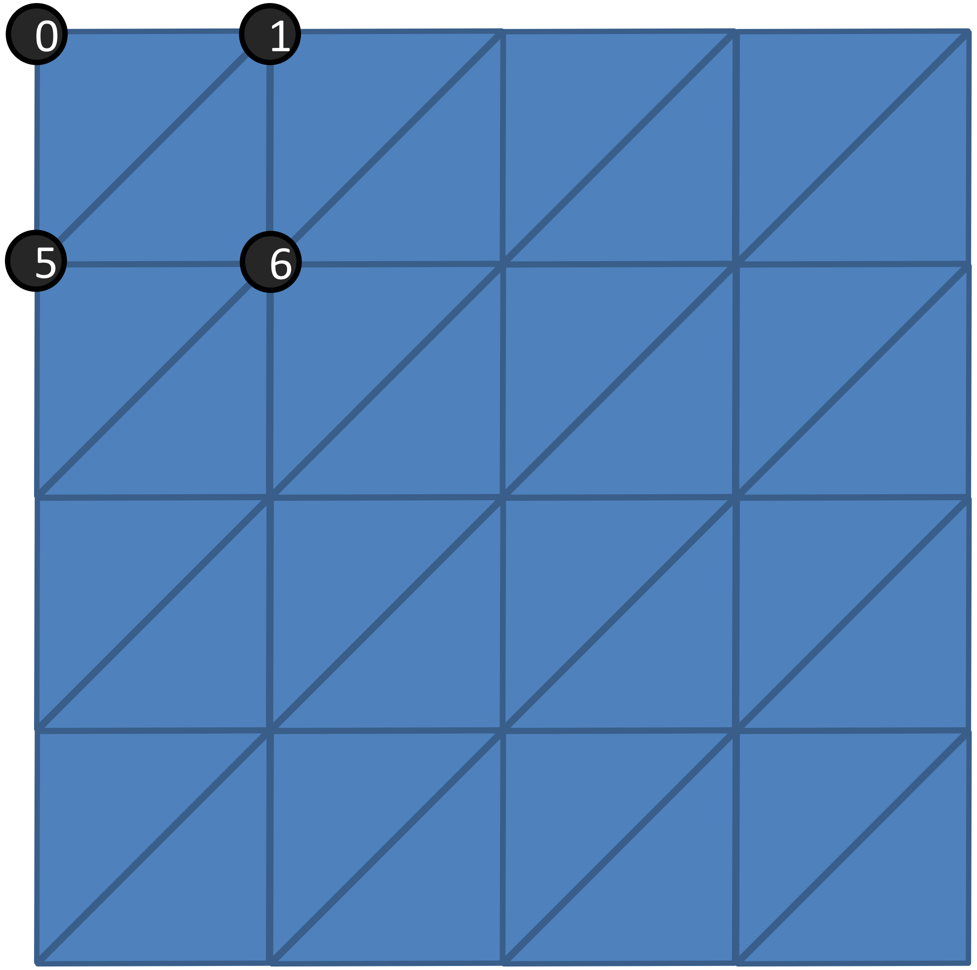


Task 2) Mesh Output

- Write OFF file
  - Simple text-based format
  - Vertices/Points:
    - Position
    - Color
  - Faces
    - Indices of vertices

```
COFF
# numVertices numFaces numEdges
4 2 0
# list of vertices
# X Y Z R G B A
0.0 1.0 0.0 255 255 255 255
0.0 0.0 0.0 255 255 255 255
1.0 0.0 0.0 255 255 255 255
1.0 1.0 0.0 255 255 255 255
# list of faces
# nVerticesPerFace idx0 idx1 idx2 ...
3 0 1 2
3 0 2 3
```

Task 2) Mesh Structure



Ensure consistent orientation of the triangles! (Usually counter-clockwise)

Example:

First triangle: 0-5-1

Second triangle: 5-6-1

Task 3) Submit your solution to Moodle

- Upload your main.cpp and a screenshot from MeshLab to Moodle
- If you worked in a group
  - **Each member** of the group should upload the solution
  - List all team members **names** and **matriculation numbers** in a separate **team_members.txt** file and upload it with your solution

See you next time!