CSC 422/522
Computer Vision and Pattern Recognition

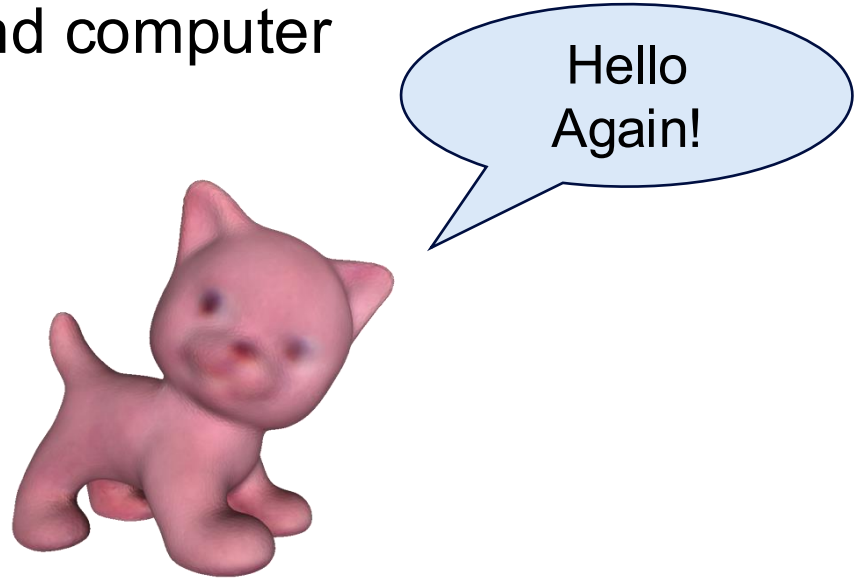
Pinhole Camera and Perspective Projection

2026 Spring

South Dakota State University

Today

- Learn basics of image formation
  - An overlapping area for computer vision and computer graphics (especially rendering)
- Focusing on the geometric side
  - Pinhole camera
  - Coordinate system
  - Perspective projection



An image rendered using POV-Ray 3.6. (Source: https://en.wikipedia.org/wiki/Rendering_(computer_graphics))

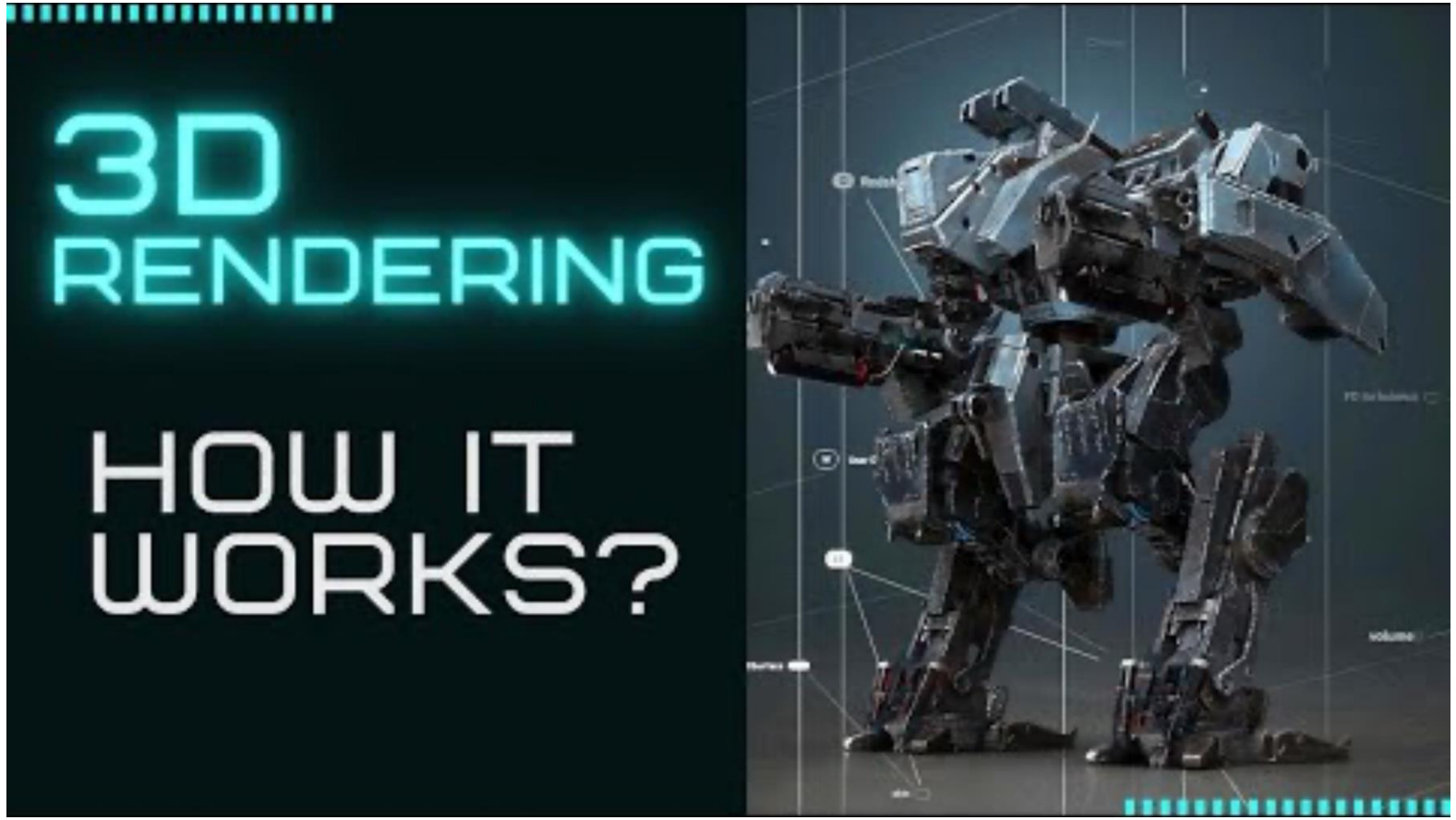
Brief Introduction: Rendering

Definition

- Rendering is the process of generating a photorealistic or non-photorealistic image from input data such as 3D models, using a computer program.

Image source:
https://upload.wikimedia.org/wikipedia/commons/thumb/3/3c/Architectural_render_02_%28Blender%29.jpg/2560px-Architectural_render_02_%28Blender%29.jpg

Example: How Rendering Works in Games



URL:
https://youtu.be/k6XVSqSVA5s?si=4-F48dzv6Xh2qq-g

Photometric Image Formation?

- In this course, we will learn how 3D geometric features in the world are projected into 2D features in an image (e.g., where are the 3D points projected in the 2D image?)
- However, images are not composed of 2D features. Instead, they are made up of discrete color or intensity values.
- Where do these values come from?

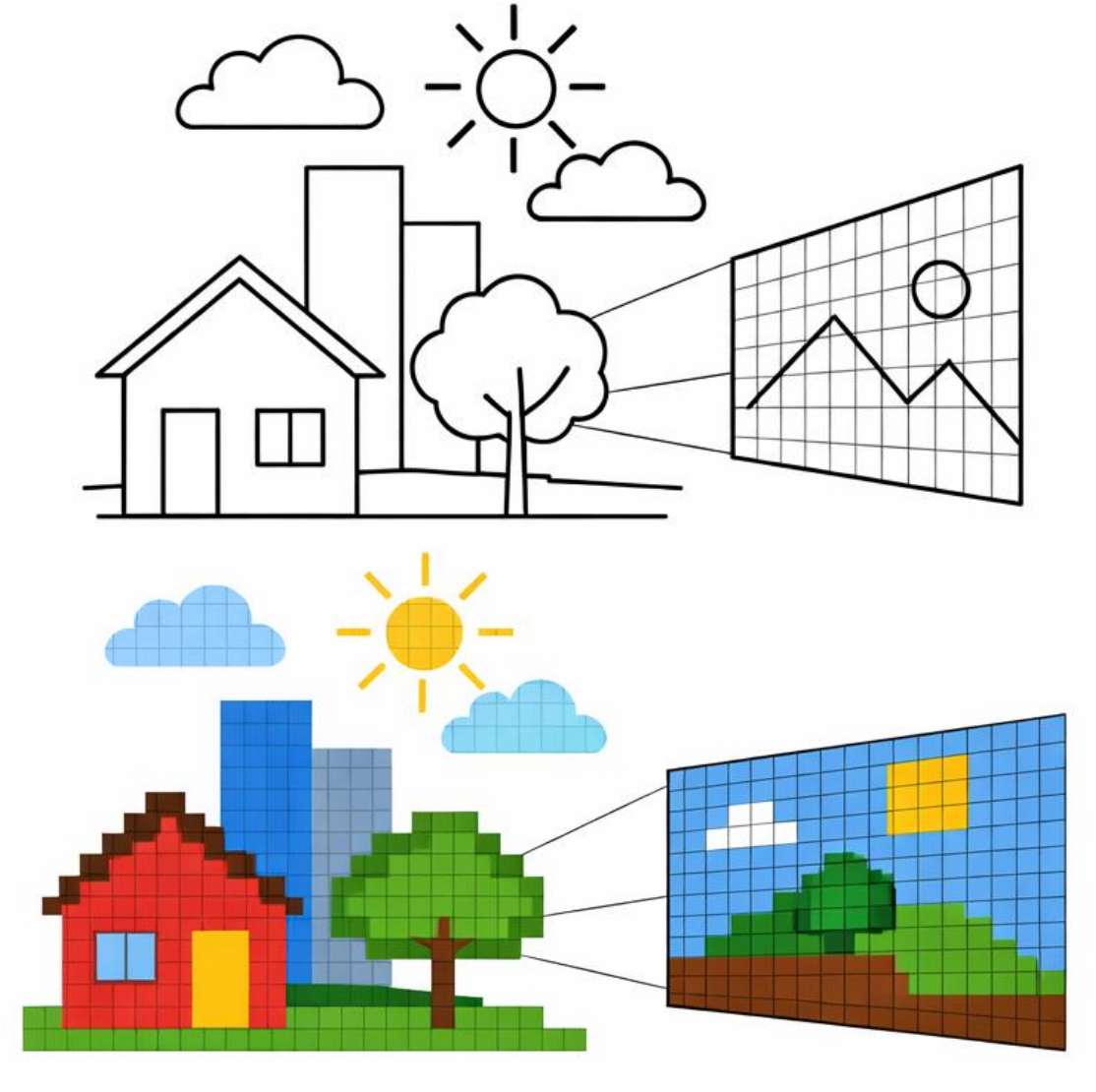


Illustration generated by ChatGPT. Is it good or not?

Photometric Image Formation?

- A simplified model of photometric image formation: Light is emitted by one or more light sources and is then reflected from an object's surface.
- A portion of this light is directed towards the camera. This simplified model ignores multiple reflections, which often occur in real-world scenes.
- Optional reading: physically based rendering (e.g., https://pbrt.org), if you are interested!

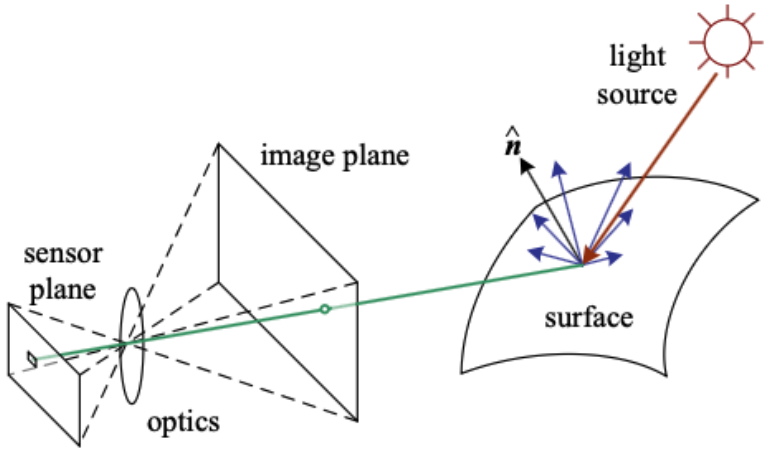
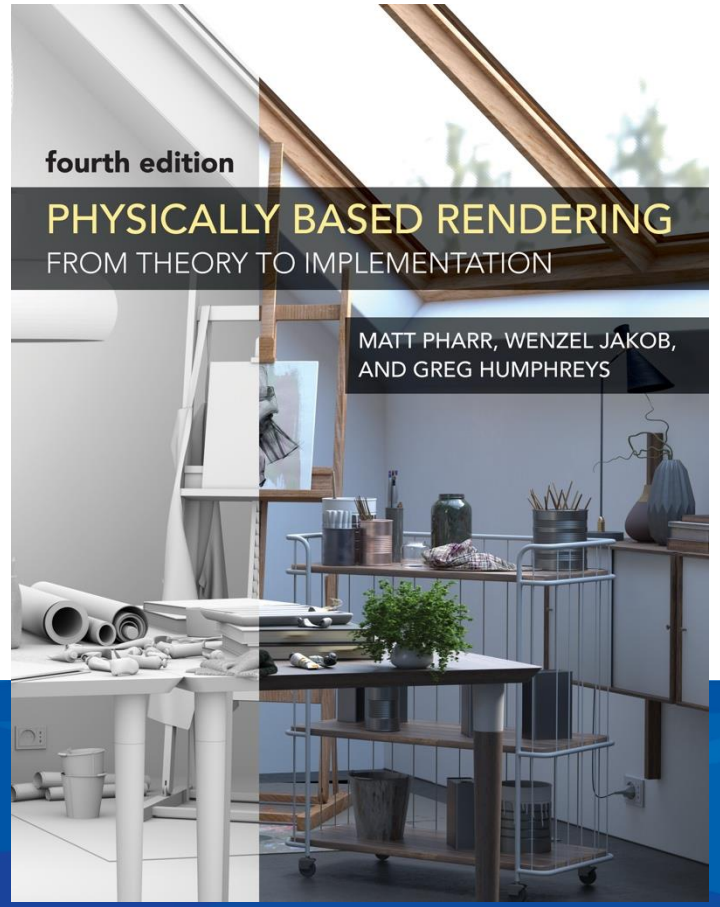


Image from Computer Vision: Algorithms and Applications, 2nd ed (Figure 2.14)



South Dakota State University

Source: https://en.wikipedia.org/wiki/Photography

Pinhole Camera: The Simplest Type of Camera

How Does a Real Camera Work?

- In a real camera, images are created when light falls on a surface sensitive to light.
  - Film camera: film
  - Digital camera: sensor or Charge-Coupled Device (CCD).
- In the real world, when light from a light source reaches an object, it is reflected in many directions. However, only one ray travels in the direction of the camera and hits the film's surface or the CCD, as illustrated.

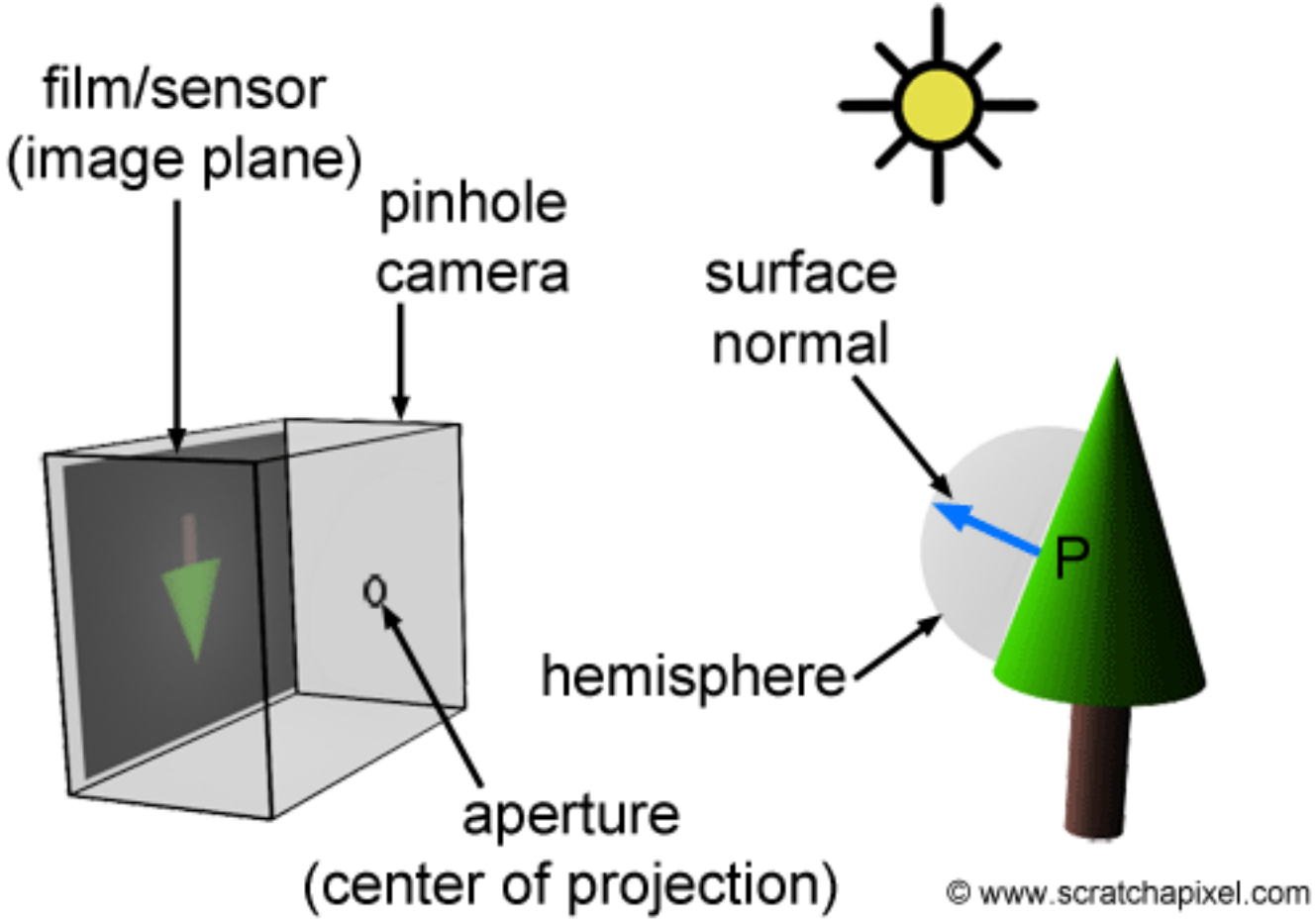
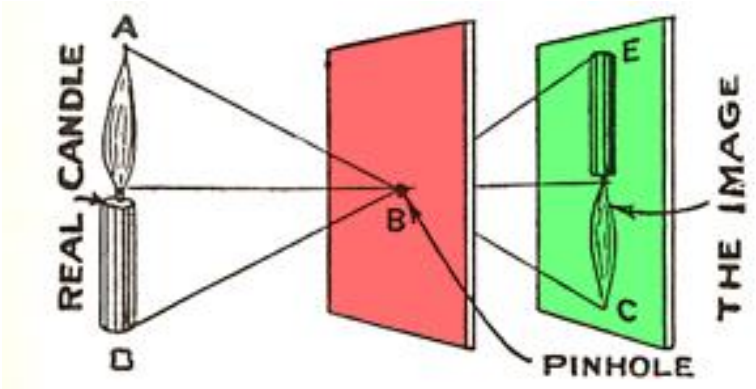
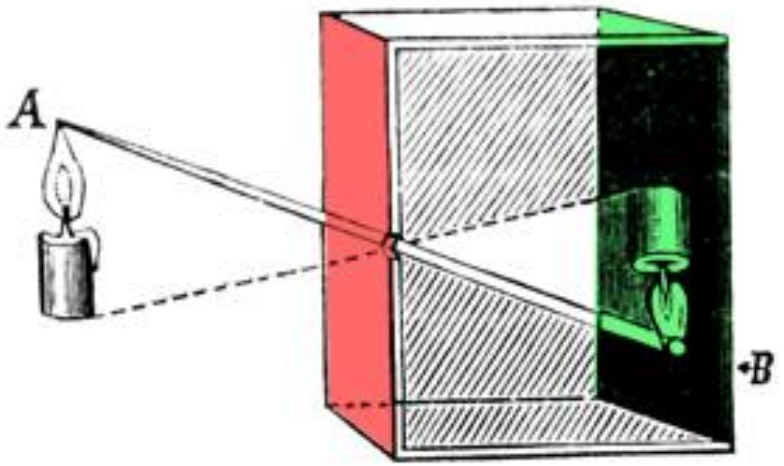


Image source: https://www.scratchapixel.com/lessons/3d-basic-rendering/3d-viewing-pinhole-camera/how-pinhole-camera-works-part-1.html

Pinhole Cameras

- The pinhole camera and camera obscura principle illustrated in 1925, in The Boy Scientist.
- This illustration depicts the simplicity and fundamental mechanics of the pinhole camera model, highlighting its significance in the evolution of photographic techniques.





Reference: https://www.scratchapixel.com/lessons/3d-basic-rendering/3d-viewing-pinhole-camera/how-pinhole-camera-works-part-1.html

Pinhole Cameras

- The simplest type of camera.
- It consists of a simple, lightproof box with a very small hole in the front, known as an **aperture**, and some light-sensitive film or paper placed inside the box on the side facing this pinhole.
- To take a picture, you simply open the aperture to expose the film to light.

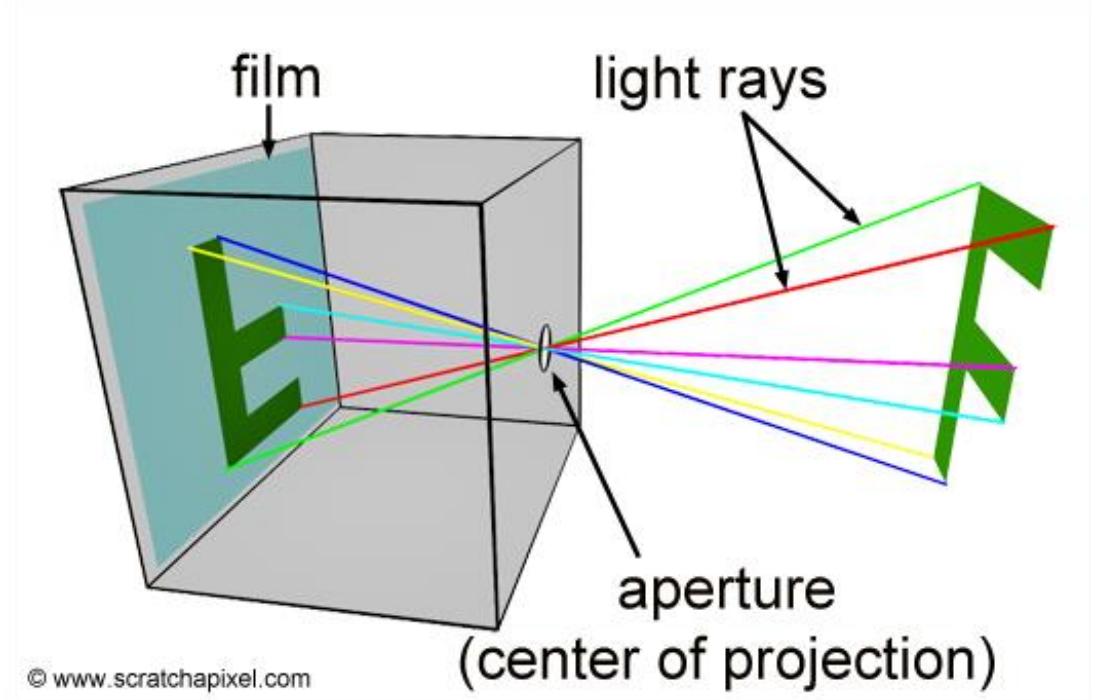


Image source: https://www.scratchapixel.com/lessons/3d-basic-rendering/3d-viewing-pinhole-camera/how-pinhole-camera-works-part-1.html

A Virtual Pinhole Camera Model (Part 1)

You may have noticed that the pinhole camera models generate inverted image.

In computer graphics and computer vision, most visualizations do something different.

(this is called a virtual image plane)

A Virtual Pinhole Camera Model (Part 2)

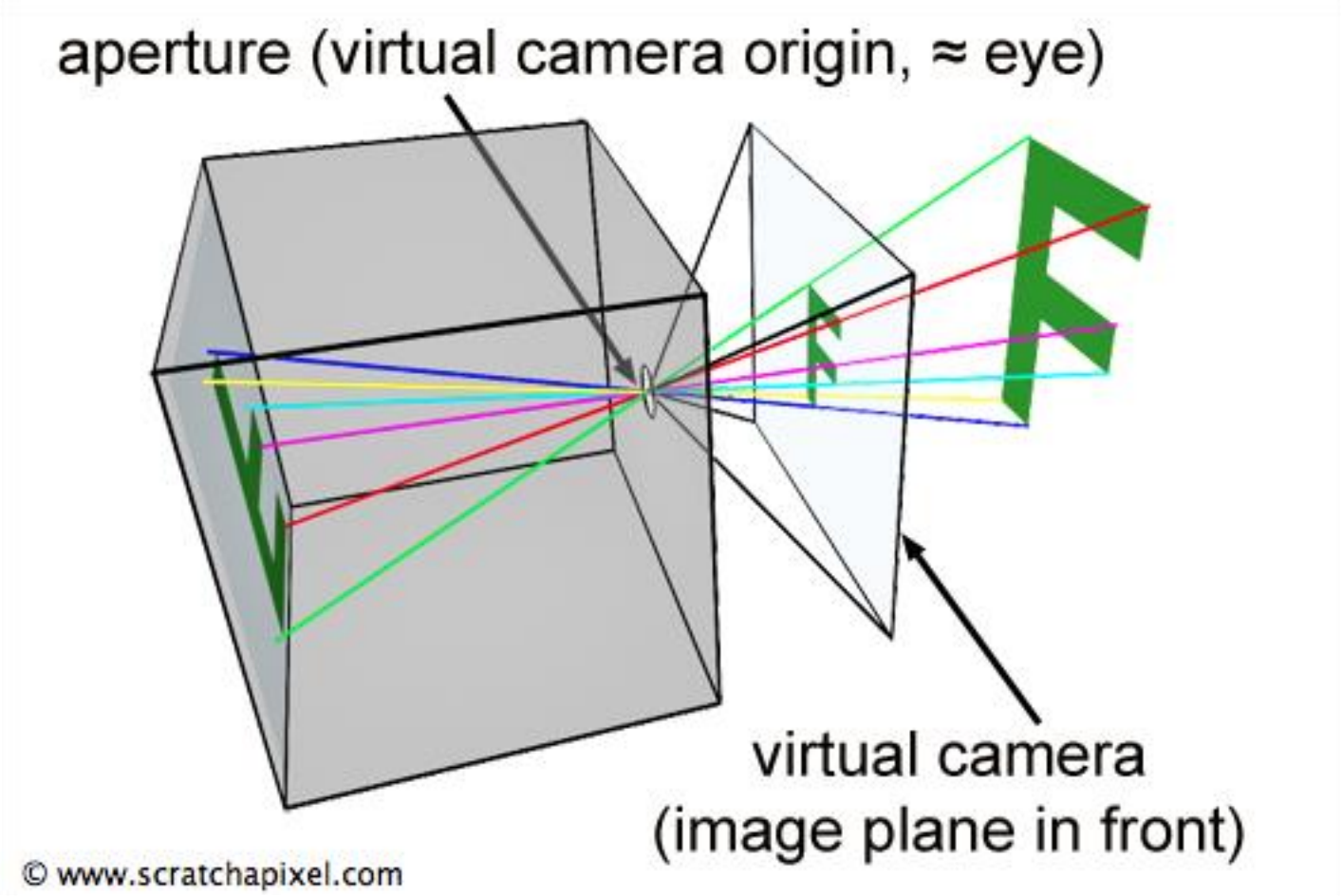
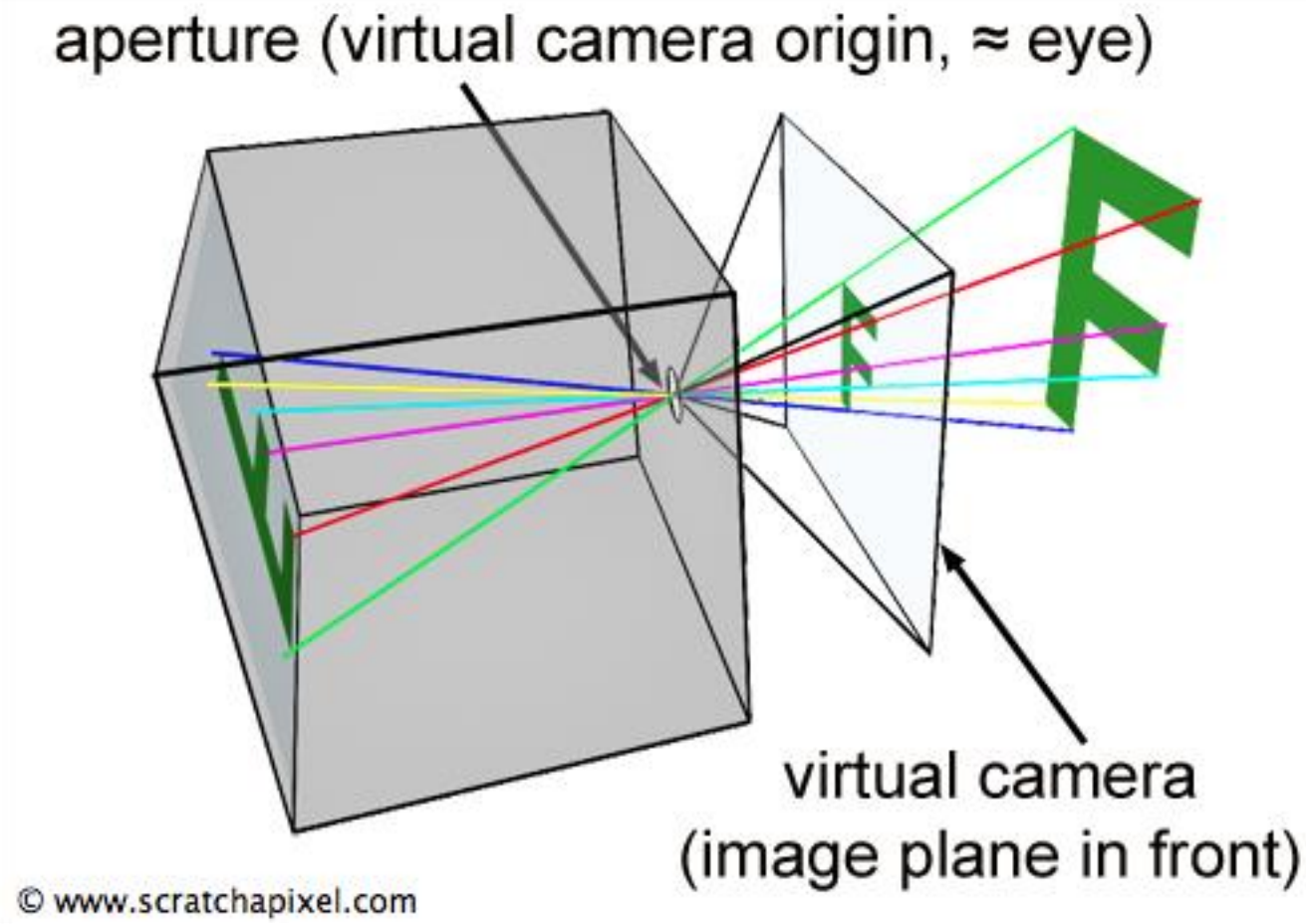


Image source:
https://www.scratchapixel.com/lessons/3d-basic-rendering/3d-viewing-pinhole-camera/virtual-pinhole-camera-model.html

A Virtual Pinhole Camera Model (Part 3)



- In real world, the image plane cannot be located in front of the aperture.
- In the **virtual** world of computers, constructing our camera in this manner is feasible.
- This way, the projected image of the scene on the image plane is not inverted.
- The real-world and virtual models are equivalent (the difference is just a sign conversion)

Image source: https://www.scratchapixel.com/lessons/3d-basic-rendering/3d-viewing-pinhole-camera/virtual-pinhole-camera-model.html

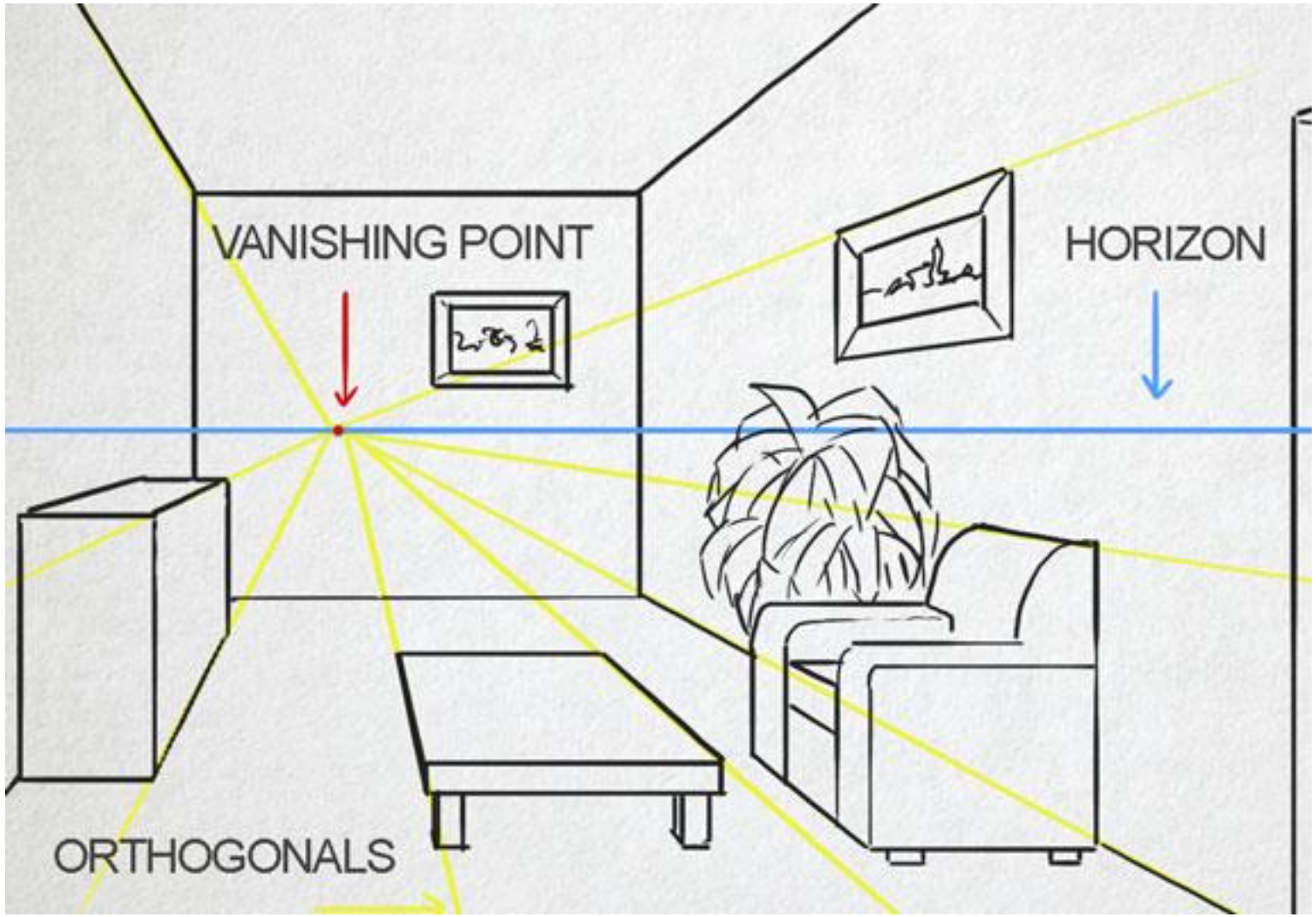


Image source: https://thevirtualinstructor.com/onepointperspective.html

Perspective Projection of a 3D Point

Goal: Compute the Location on Image Plane

Consider a 3D point $\mathbf{p}^c$ as shown in Fig. 11.1.

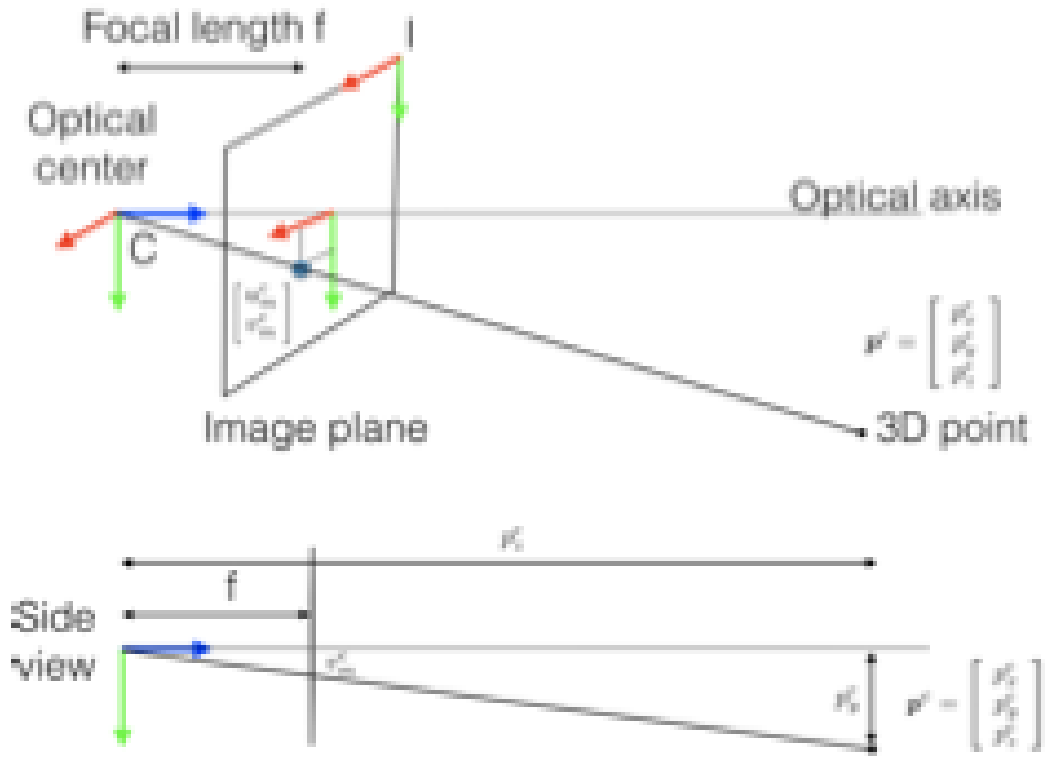
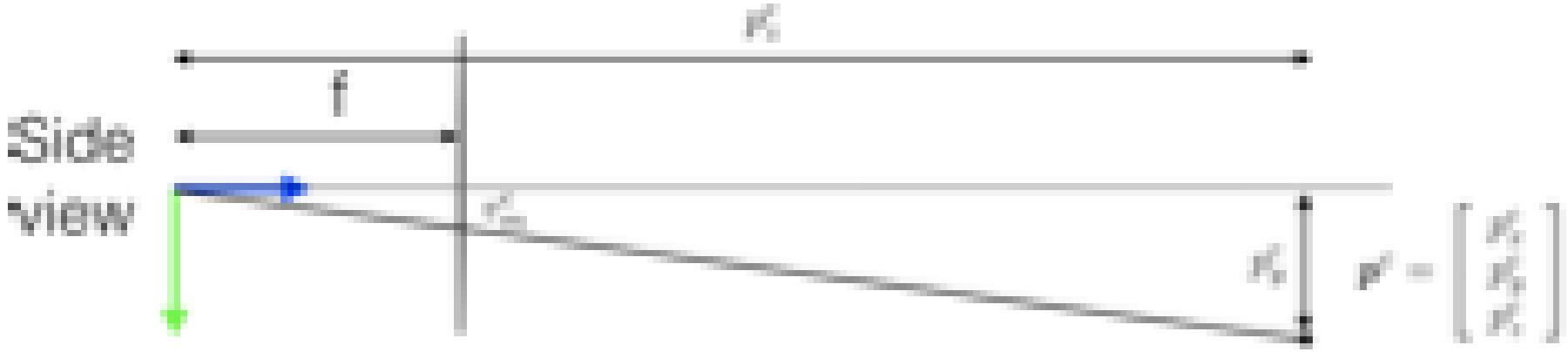


Figure 11.1: Pinhole Model.

Credit: MIT 16.485, the pdf notes can be found in https://vnav.mit.edu/material/11-ImageFormation-notes.pdf

Using the Similarity of Triangles ☺



The position (u_m^c, v_m^c) of the projection of $\boldsymbol{p}^c$ on the camera plane is:

$$u_m^c = f\frac{p_x^c}{p_z^c} \qquad v_m^c = f\frac{p_y^c}{p_z^c} \tag{11.1}$$

where f is the *focal length* (in meters), and the subscript m denotes that (u_m^c, v_m^c) are still expressed in meters (later on we convert them into pixels).

Conversion to Pixels

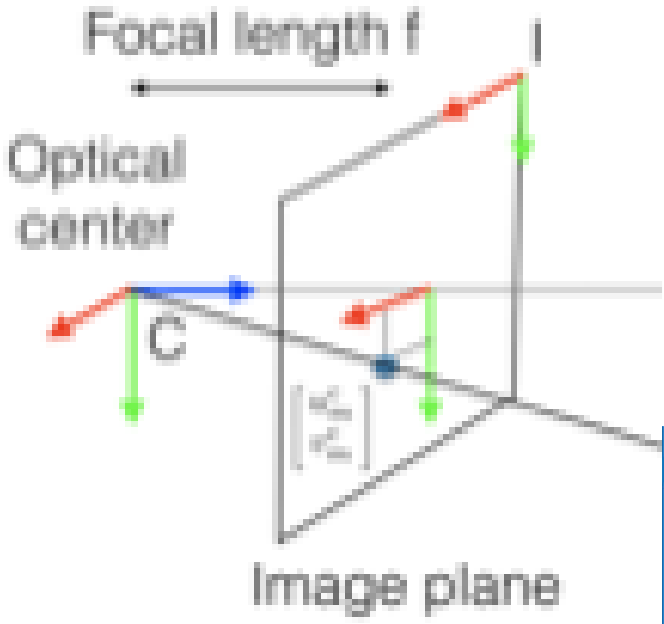
We already noticed that the quantities (u_m^c, v_m^c) are expressed in meters. Moreover, during the first lecture, we noticed that the image coordinate frame typically has origin in the top-left corner of the image (see frame "I" in Fig. 11.1).

Therefore, we can convert (u_m^c, v_m^c) to pixels and change the reference frame as follows:

$$u^I = s_x u_m^c + o_x \qquad v^I = s_y v_m^c + o_y \tag{11.5}$$

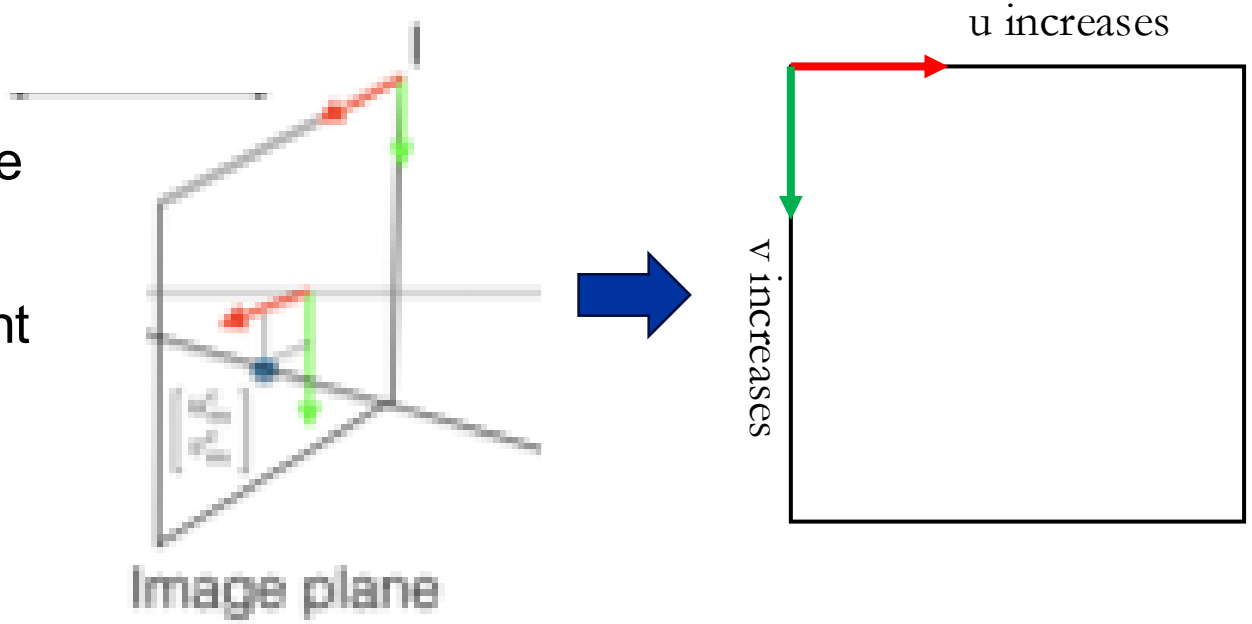
Terminologies:

- $[o_x, o_y]$ is called the principal point
- $s_x f$ is the focal length in horizontal pixels (usually denoted as f_x)
- $s_y f$ is the focal length in vertical pixels (usually denoted as f_y)



Convention of Image Coordinate System (e.g., in OpenCV)

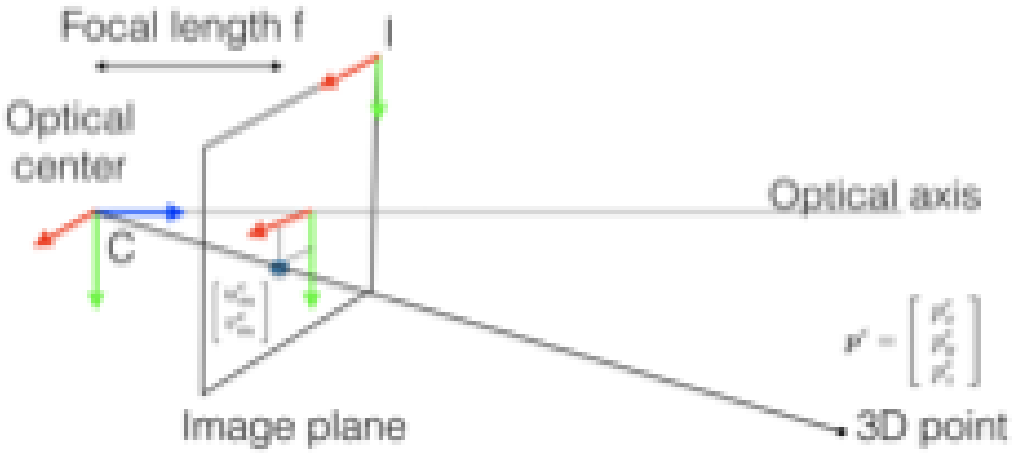
- The origin (0,0) corresponds to the top-left corner.
- The x (i.e., u) values go left-to-right (column number).
- The y (i.e., v) values go top-to-bottom (row number).



The Full Equation from 3D Point to Pixel

$$u_m^c = f\frac{p_x^c}{p_z^c} \qquad v_m^c = f\frac{p_y^c}{p_z^c} \tag{11.1}$$

$$u^I = s_x u_m^c + o_x \qquad v^I = s_y v_m^c + o_y \tag{11.5}$$



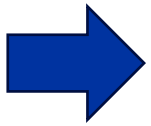
And denote: $f_x = s_x f$, $f_y = s_y f$

We can get: $u^I = s_x f \frac{p_x^c}{p_z^c} + o_x = f_x \frac{p_x^c}{p_z^c} + o_x$

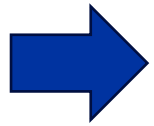
$$v^I = s_y f \frac{p_y^c}{p_z^c} + o_y = f_y \frac{p_y^c}{p_z^c} + o_y$$

Can we obtain a more concise equation?

- $u = f_x \frac{p_x^c}{p_z^c} + o_x$
- $v = f_y \frac{p_y^c}{p_z^c} + o_y$



- $up_z^c = f_x p_x^c + o_x p_z^c$
- $vp_z^c = f_y p_y^c + o_y p_z^c$



$$p_z^c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x^c \\ p_y^c \\ p_z^c \end{bmatrix}$$

We can drop the superscript "I"

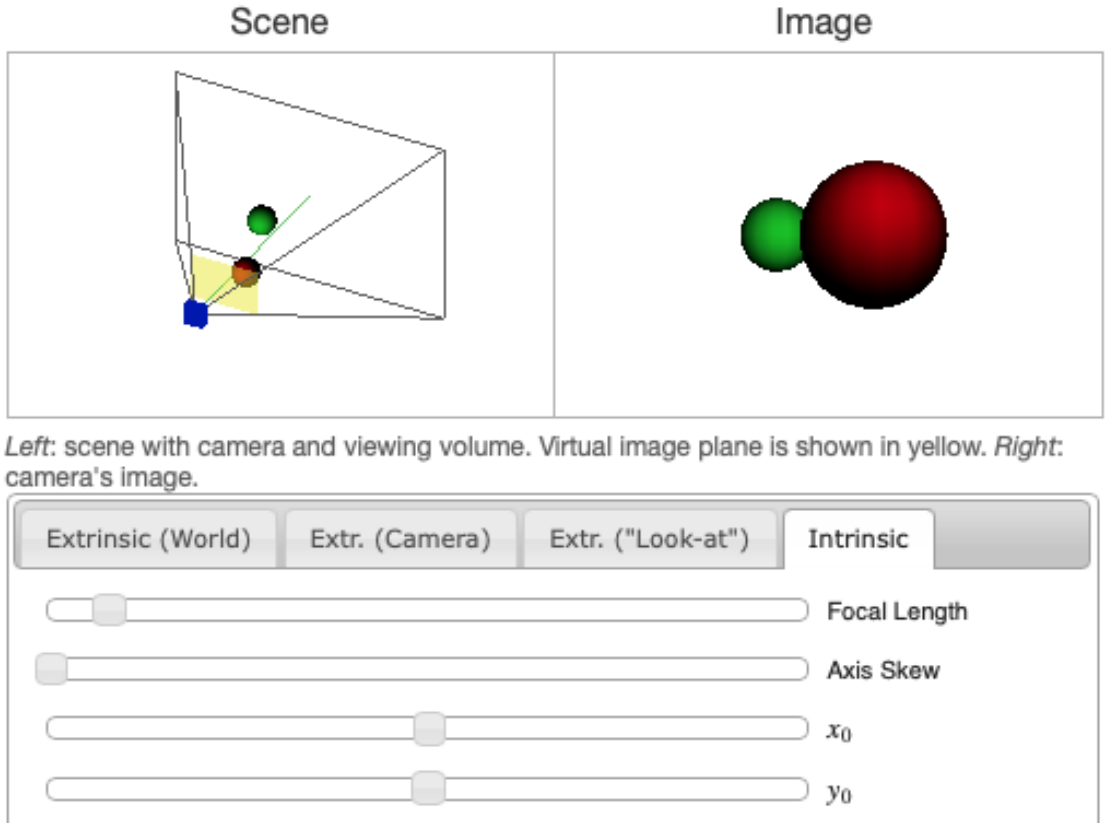
The Intrinsic Matrix *K*

$$p_z^c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x^c \\ p_y^c \\ p_z^c \end{bmatrix}$$

Note: in this course, we assume the pixels are not skewed. Otherwise, the value in the first row, second column will be non-zero.

Notations we use:

- $[o_x, o_y]$ is called the principal point
- f_x is the focal length in horizontal pixels
- f_y is the focal length in vertical pixels



Left: scene with camera and viewing volume. Virtual image plane is shown in yellow. *Right*: camera's image.

Let's try the perspective camera toy in https://ksimek.github.io/perspective_camera_toy.html

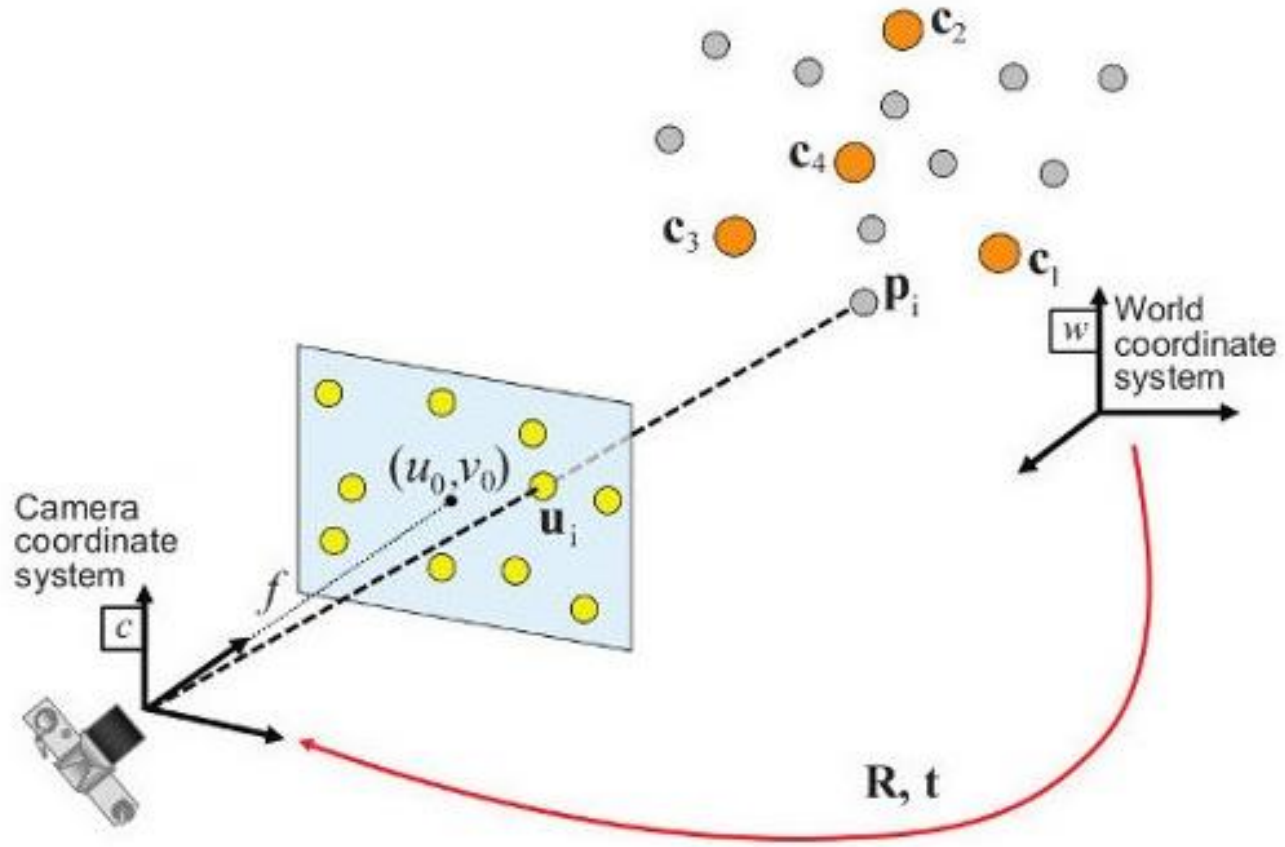


Image source: https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html

World-to-Camera Transformation

Camera and World Coordinate Systems

Camera Coordinate System

- It is attached to the camera, and by convention:
- Z-axis along the viewing direction
- X, Y axes aligned with the image plane

World Coordinate System

- It is a reference frame you choose to describe the scene, which has
- An origin (chosen arbitrarily)
- Axes (Chosen arbitrarily)

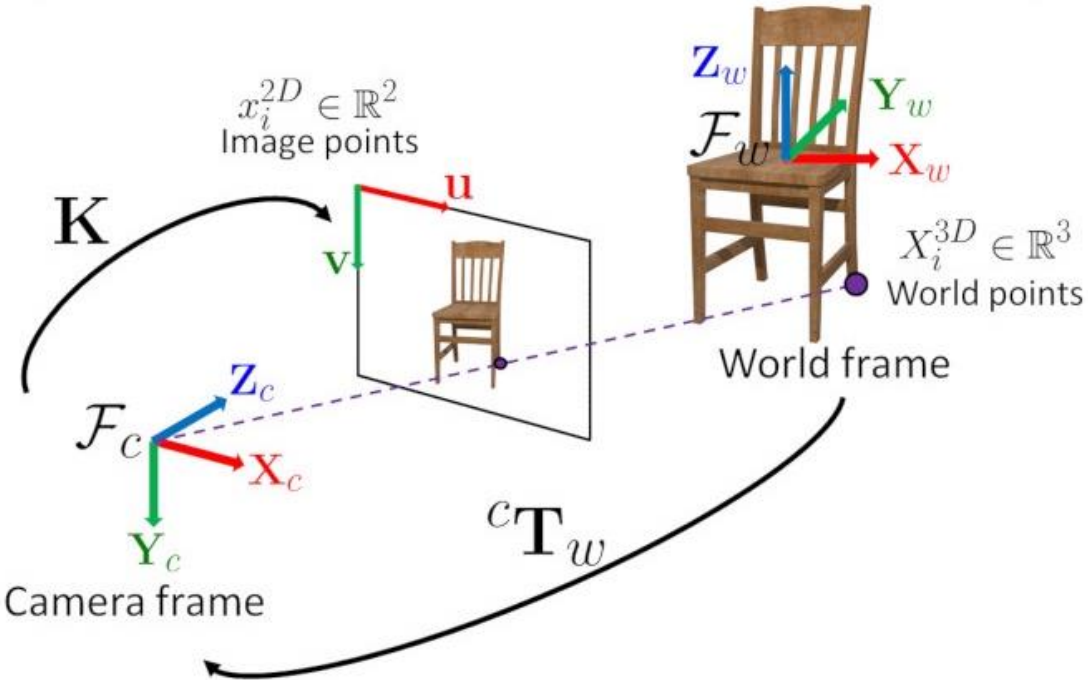
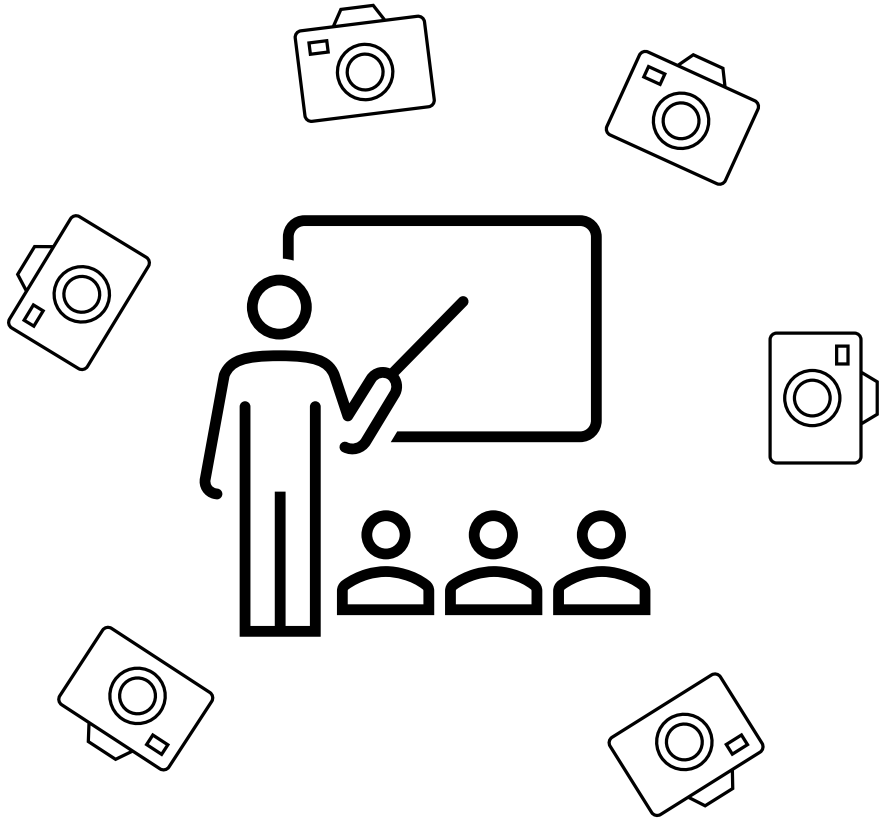


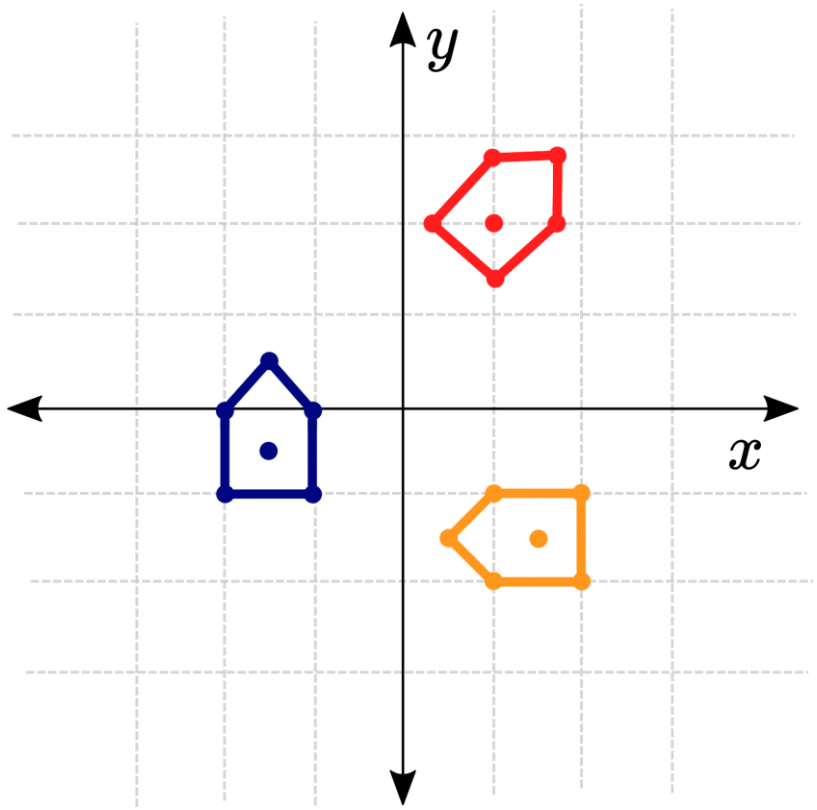
Image source: https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html

Why need different coordinate systems?

- World coordinates do NOT depend on the camera.
- Physical analogy: imaging holding a phone camera and walk around the classroom
  - The classroom doesn’t move (world coordinates)
  - You walk and change the camera orientations
  - The camera coordinate system changed



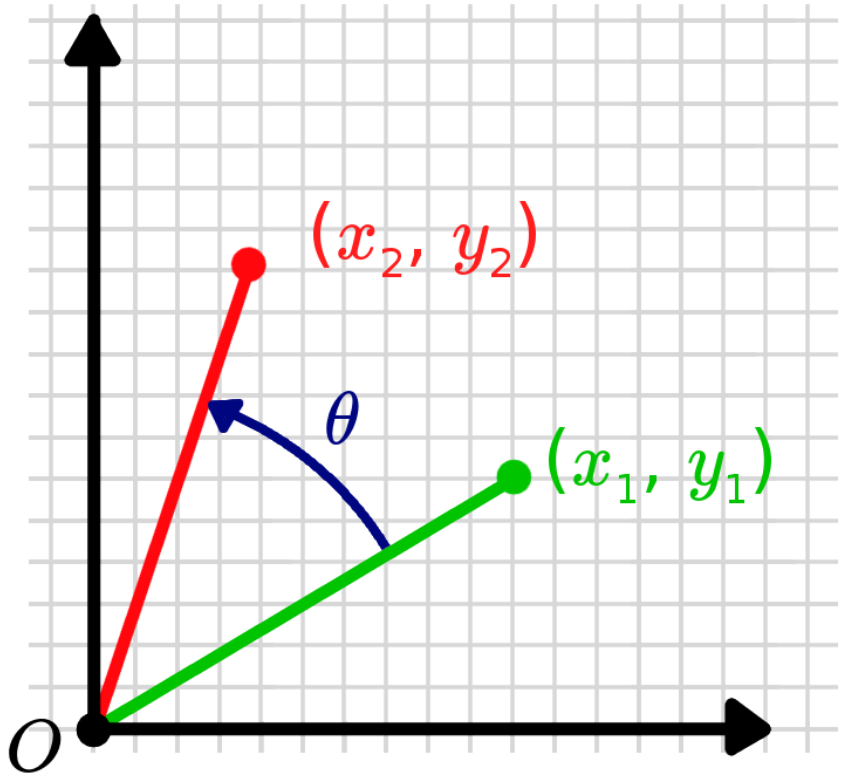
Let's start with transformations in 2D space…



We focus on **rigid** transformations including rotations and translations.

Image source: https://articulatedrobotics.xyz/tutorials/coordinate-transforms/coordinate-transforms

2D Rotation



We can use a 2x2 transformation to describe it.

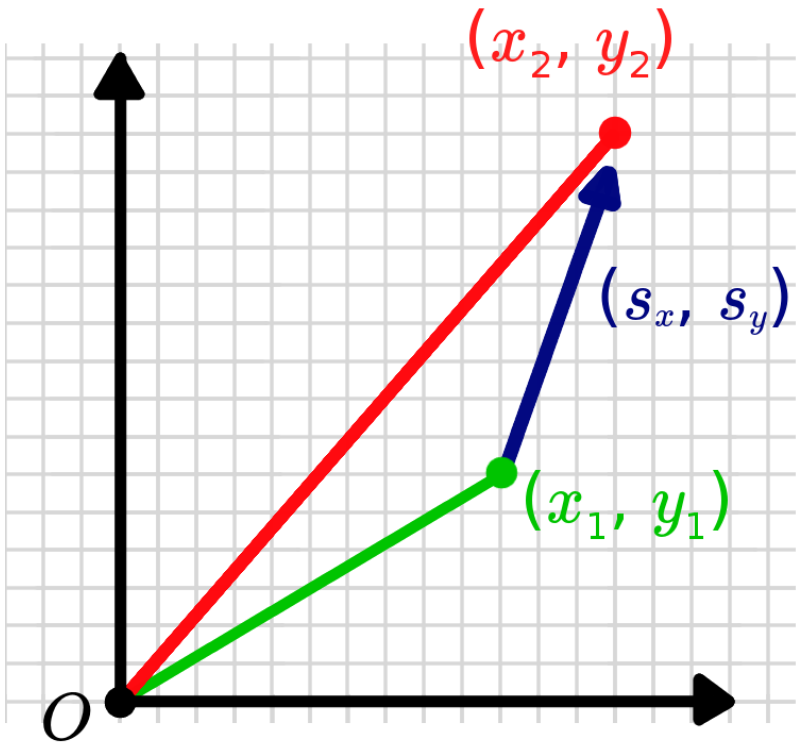
$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

For the 2D case, we can write down the rotation matrix using the rotation angle easily:

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

Image source: https://articulatedrobotics.xyz/tutorials/coordinate-transforms/rotation-matrices-2d

2D Translation



$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} s_x \\ s_y \end{bmatrix} = \begin{bmatrix} x + s_x \\ y + s_y \end{bmatrix}$$

What's the problem of the above operation?

- We would like a nice form like f(**p**) = **Ap**, where **A** is a matrix.
- But the above way to perform translation looks like f(**p**) = **p** + **b**, where **b** is a vector.
- When we need to perform multiple transformations, the f(**p**) = **Ap** form gives cleaner results.

Reference: https://articulatedrobotics.xyz/tutorials/coordinate-transforms/translations

Introducing... Homogeneous Coordinates!

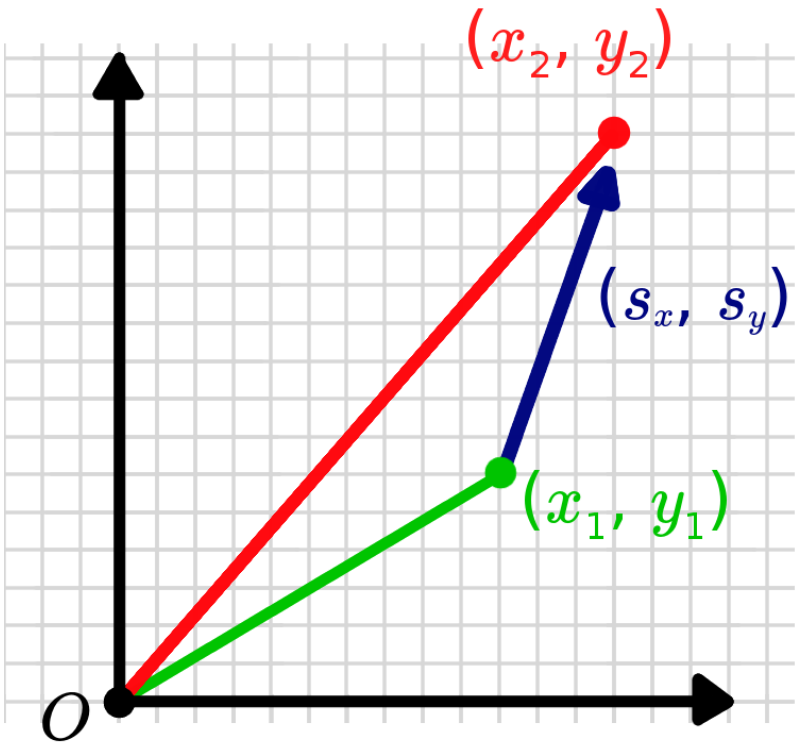
- By using a homogeneous coordinate system, we can represent both rotations and translations using a single matrix.
- The first thing we have to do is to modify our coordinates, adding a “1” onto the end of our point vector.

$$\bar{\mathbf{p}} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Note: you can add a bar (¯) above the original variable to express that you are working with the homogenous coordinates.

Reference: https://articulatedrobotics.xyz/tutorials/coordinate-transforms/translations

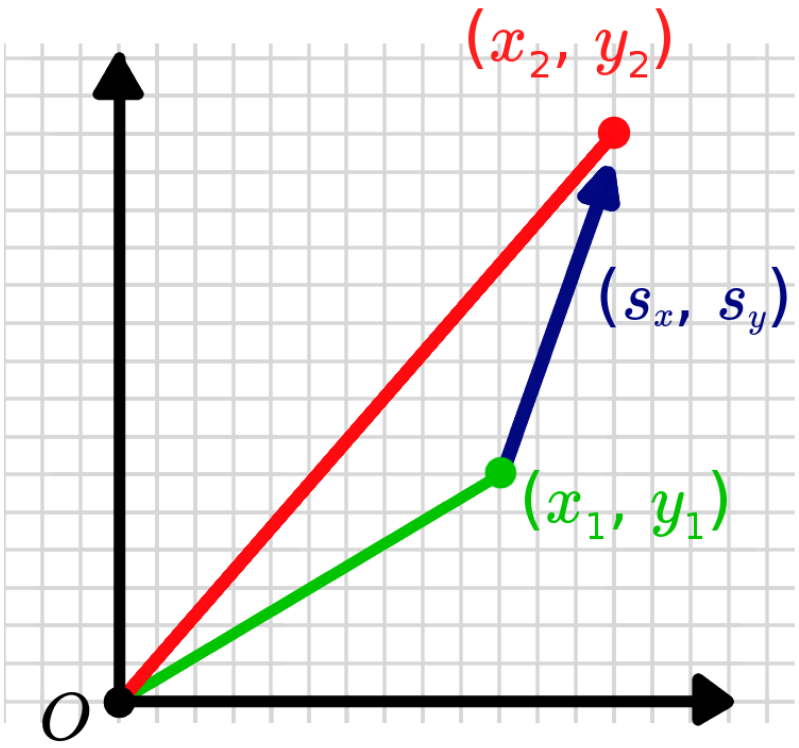
2D Translation Using Homogenous Coordinates



$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} x_1 + s_x \\ y_1 + s_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & s_x \\ 0 & 1 & s_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Reference: https://articulatedrobotics.xyz/tutorials/coordinate-transforms/translations

2D Translation Using Homogenous Coordinates

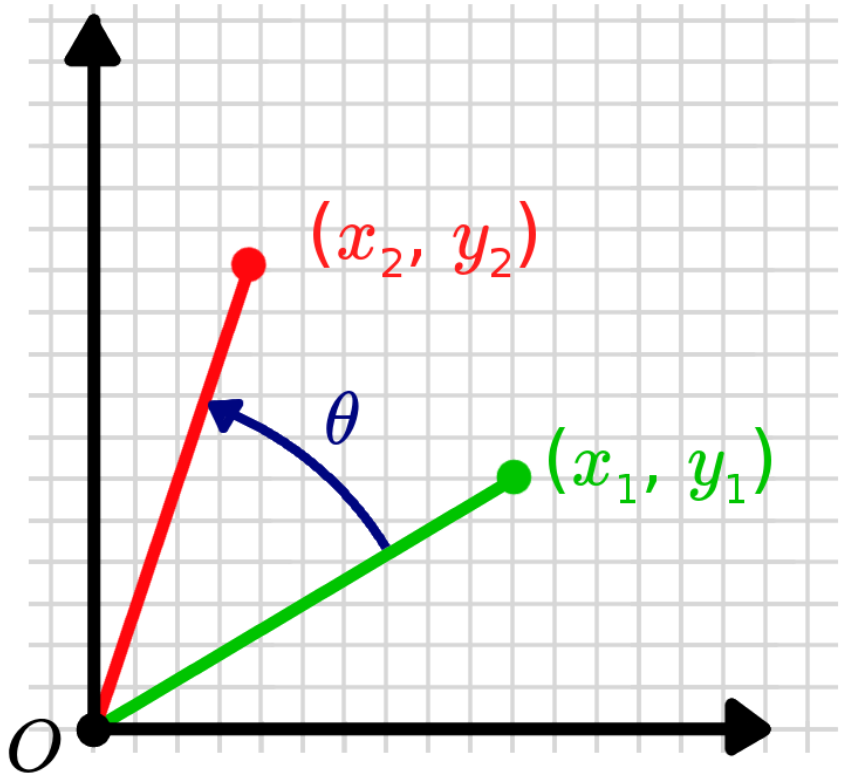


$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} x_1 + s_x \\ y_1 + s_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & s_x \\ 0 & 1 & s_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

- We get a nice form like f(**p**) = **Ap**, where **A** is a matrix ☺
- Why is it called *homogenous*? Because (x, y, 1) ~ (kx, ky, k), i.e., they represent the same point after normalization.

Reference: https://articulatedrobotics.xyz/tutorials/coordinate-transforms/translations

2D Rotation Using Homogenous Coordinates



$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & 0 \\ r_{21} & r_{22} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Image source: https://articulatedrobotics.xyz/tutorials/coordinate-transforms/rotation-matrices-2d

2D Transformation Matrix (Rotation + Translation) "T", Using Homogenous Coordinates

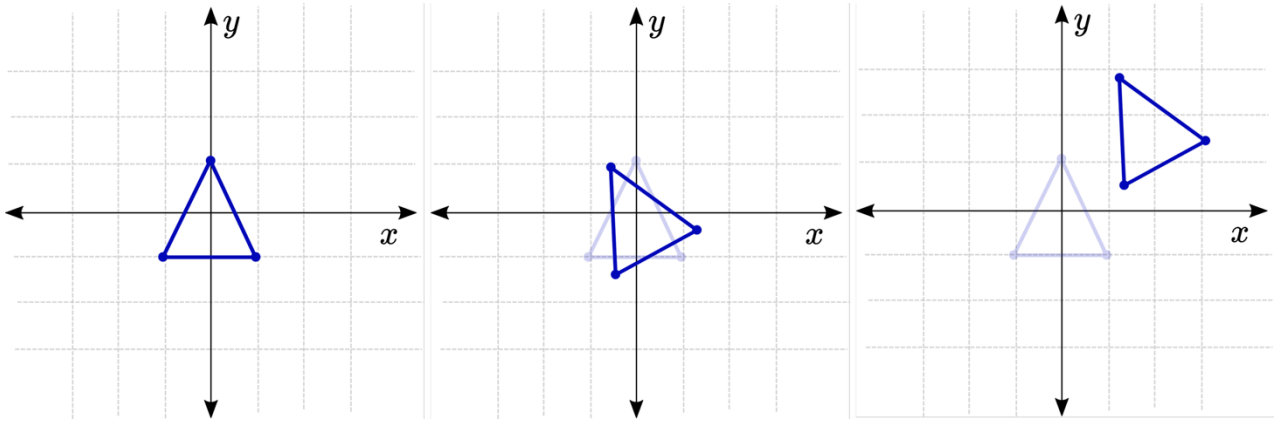


Image source: https://articulatedrobotics.xyz/tutorials/coordinate-transforms/transformation-matrices

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & s_x \\ 0 & 1 & s_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & 0 \\ r_{21} & r_{22} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

$$= \boxed{\begin{bmatrix} r_{11} & r_{12} & s_x \\ r_{21} & r_{22} & s_y \\ 0 & 0 & 1 \end{bmatrix}} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

T

Extension to n-Dimensional Space

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & s_x \\ r_{21} & r_{22} & s_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

$$T_{2\times 2} = \begin{bmatrix} R_{2\times 2} & t_{2\times 1} \\ 0_{1\times 2} & 1 \end{bmatrix}$$

$$T_{n\times n} = \begin{bmatrix} R_{n\times n} & t_{n\times 1} \\ 0_{1\times n} & 1 \end{bmatrix}$$

In 3D Space: World-to-Camera Transformation

$$T_{n\times n} = \begin{bmatrix} R_{n\times n} & t_{n\times 1} \\ 0_{1\times n} & 1 \end{bmatrix}$$

Rewriting using homogenous coordinates:

$$\tilde{\mathbf{x}}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

Extrinsic Matrix: $M_{ext} = \begin{bmatrix} R_{3\times 3} & \mathbf{t} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$

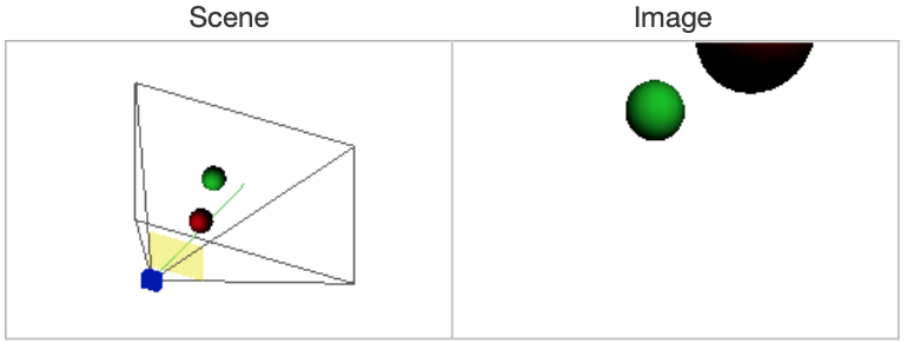
Note: some illustrations taken from https://www.youtube.com/watch?v=qByYk6JggQU

In 3D Space: World-to-Camera Transformation

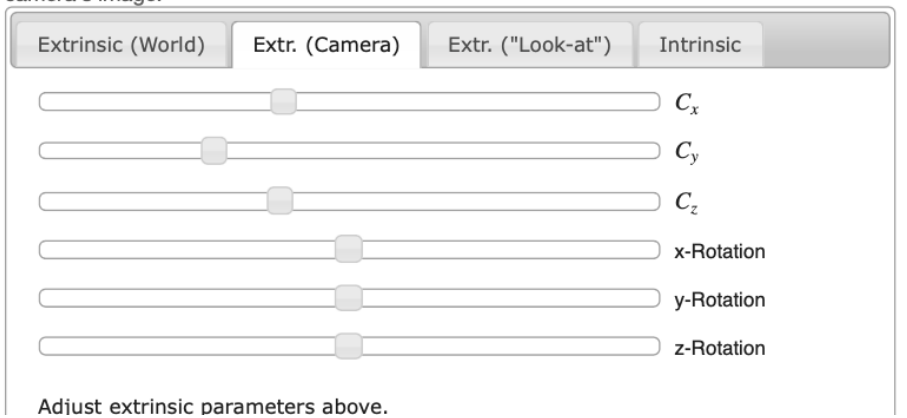
Rewriting using homogenous coordinates:

$$\tilde{\mathbf{x}}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

Extrinsic Matrix: $M_{ext} = \begin{bmatrix} R_{3\times3} & \mathbf{t} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$



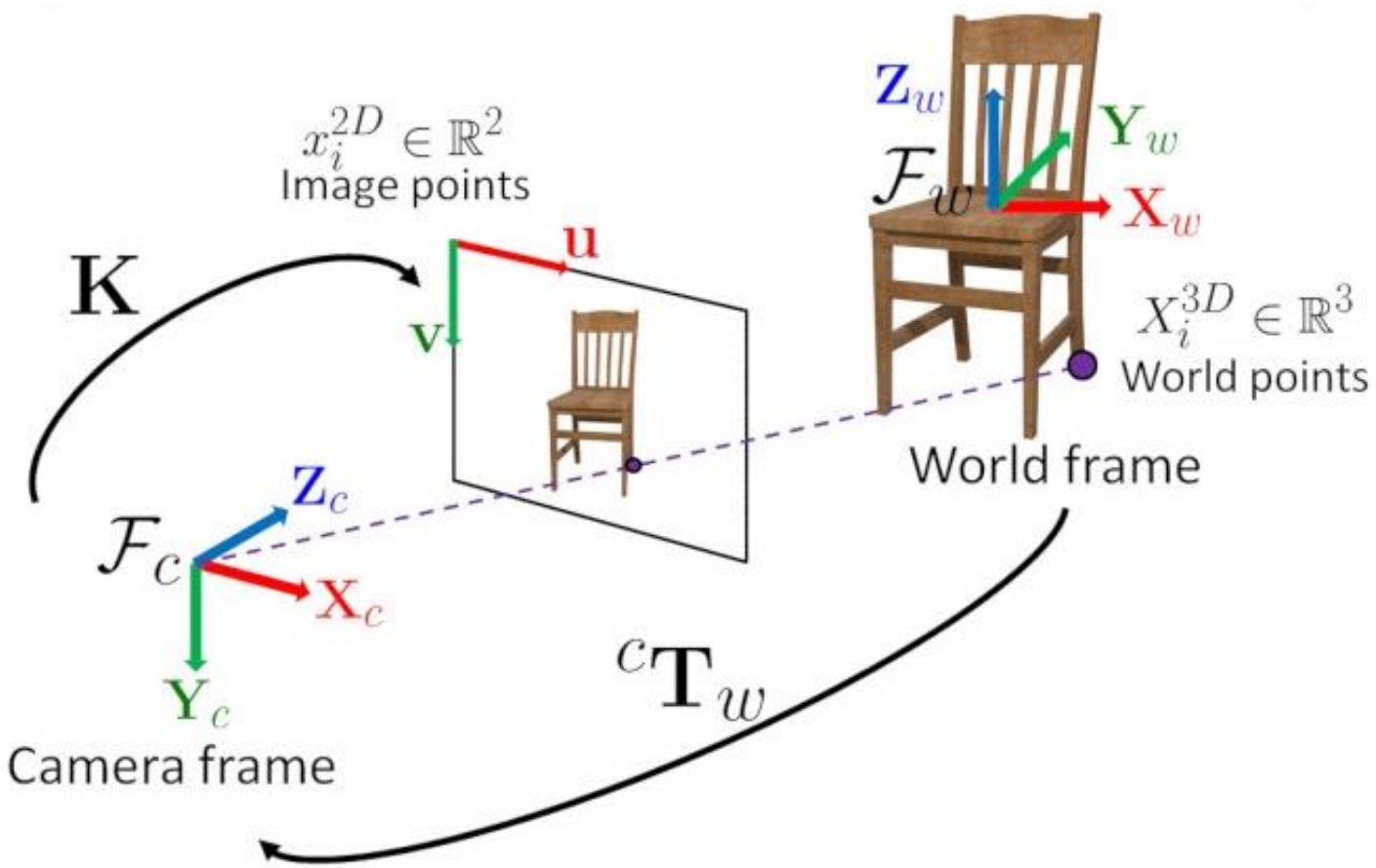
Left: scene with camera and viewing volume. Virtual image plane is shown in yellow. *Right*: camera's image.



Interactive demonstration of extrinsic parameters: https://ksimek.github.io/2012/08/22/extrinsic/

Note: some illustrations taken from https://www.youtube.com/watch?v=qByYk6JggQU

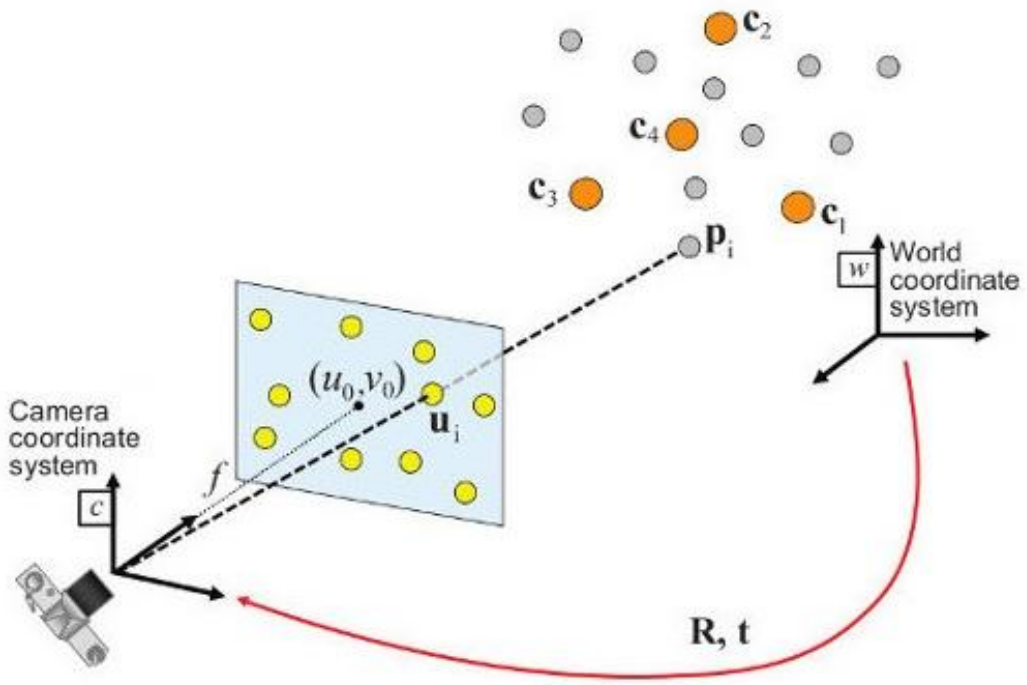
South Dakota State University

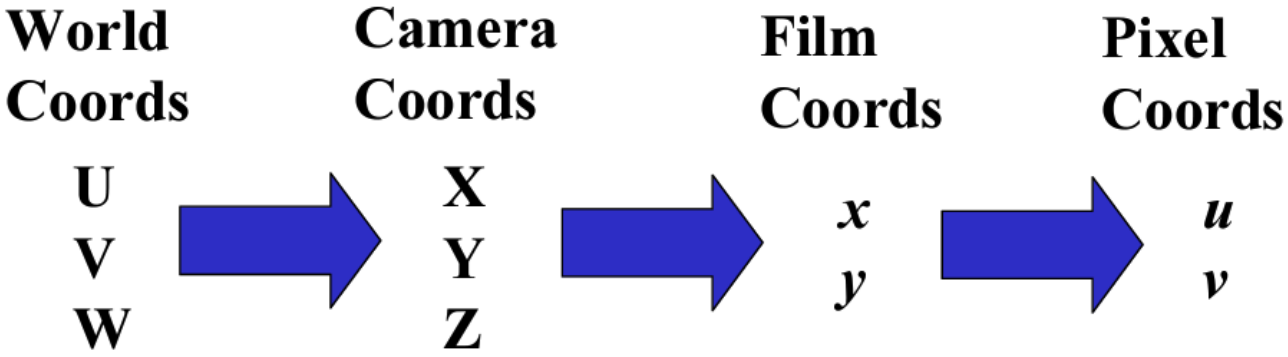


Reference: https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html

Putting It All Together: Forward Projection

Forward Projection





We want a mathematical model to describe how 3D World points get projected into 2D Pixel coordinates.

Our goal: describe this sequence of transformations by a big matrix equation!

Reference: https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html

Slide Credit: CSC492/592, 2024 Spring

Combining Extrinsic and Intrinsic Matrices

Image Coordinates

$$X_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

Intrinsic

Camera Coordinates

$$X_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}$$

Extrinsic

World Coordinates

$$X_w = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

$$z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}$$

Using homogenous coordinates:

$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}, \begin{bmatrix} z_c u \\ z_c v \\ z_c \end{bmatrix}$ represent the same point

Note: some illustrations taken from https://www.youtube.com/watch?v=qByYk6JggQU

Projection Matrix *P*

Camera to Pixel

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

$$\tilde{\mathbf{u}} = M_{int}\, \tilde{\mathbf{x}}_c$$

World to Camera

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

$$\tilde{\mathbf{x}}_c = M_{ext} \tilde{\mathbf{x}}_w$$

Combining the above two equations, we get the full projection matrix P:

$$\tilde{\mathbf{u}} = M_{int} M_{ext}\, \tilde{\mathbf{x}}_w = P\, \tilde{\mathbf{x}}_w$$

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

K. Navar

Note: some illustrations taken from https://www.youtube.com/watch?v=qByYk6JggQU

Takeaway

- Image formation?
- Perspective projection / Coordinate transformation?
- Intrinsic / Extrinsic matrix?
- Projection matrix?