COMPUTER VISION

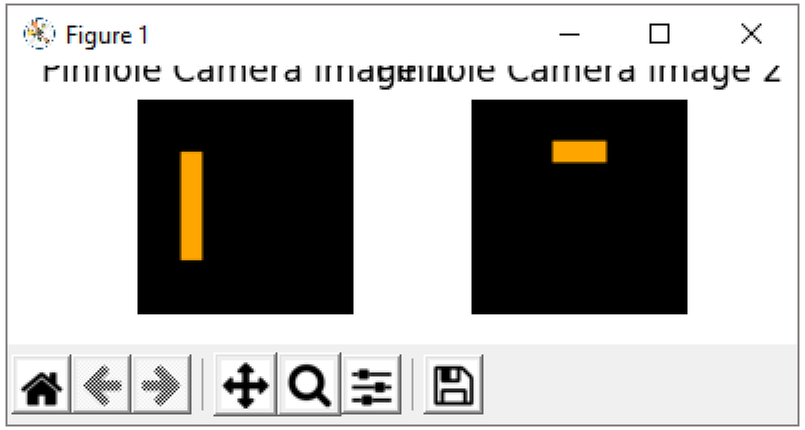
CAMERA & IMAGE FORMATION

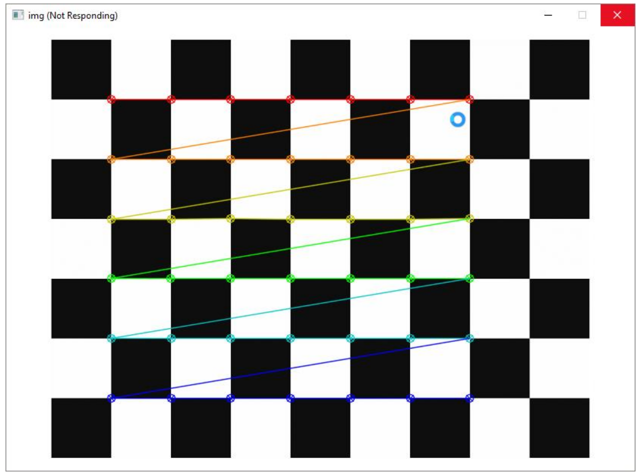
Ngo Quoc Viet-2024

CONTENTS

LECTURE OUTCOMES

- Understanding pinhole camera, perspective projection and calibration matrices.
- Implementing calibration to determine intrinsic and extrinsic matrices.
- Implementing the small augmenting application to embed 3D objects to reality scence.





$$K = \begin{bmatrix} \alpha & 0 & c_x \\ 0 & \beta & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

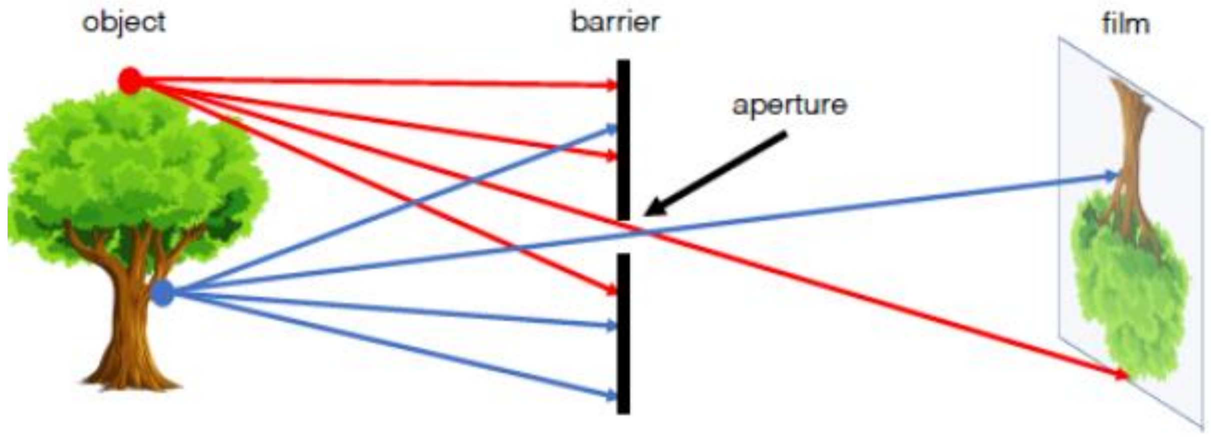
$$[\mathrm{R} \quad t] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_z \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}$$

SOME CAMERA MODELS

- **Pinhole Camera Model**: simplest mathematical model that can be applied to a lot of real photography devices.
- **Thin Lens Model**: This model incorporates a thin lens to simulate the effects of lenses on image formation
- **Fish-Eye Camera Model**: Fish-eye lenses capture a very wide field of view.
- **Spherical Camera Model**: represents cameras with spherical or panoramic lenses
- **Omni-Directional Camera Model**: Omni-directional cameras capture images from all directions simultaneously.

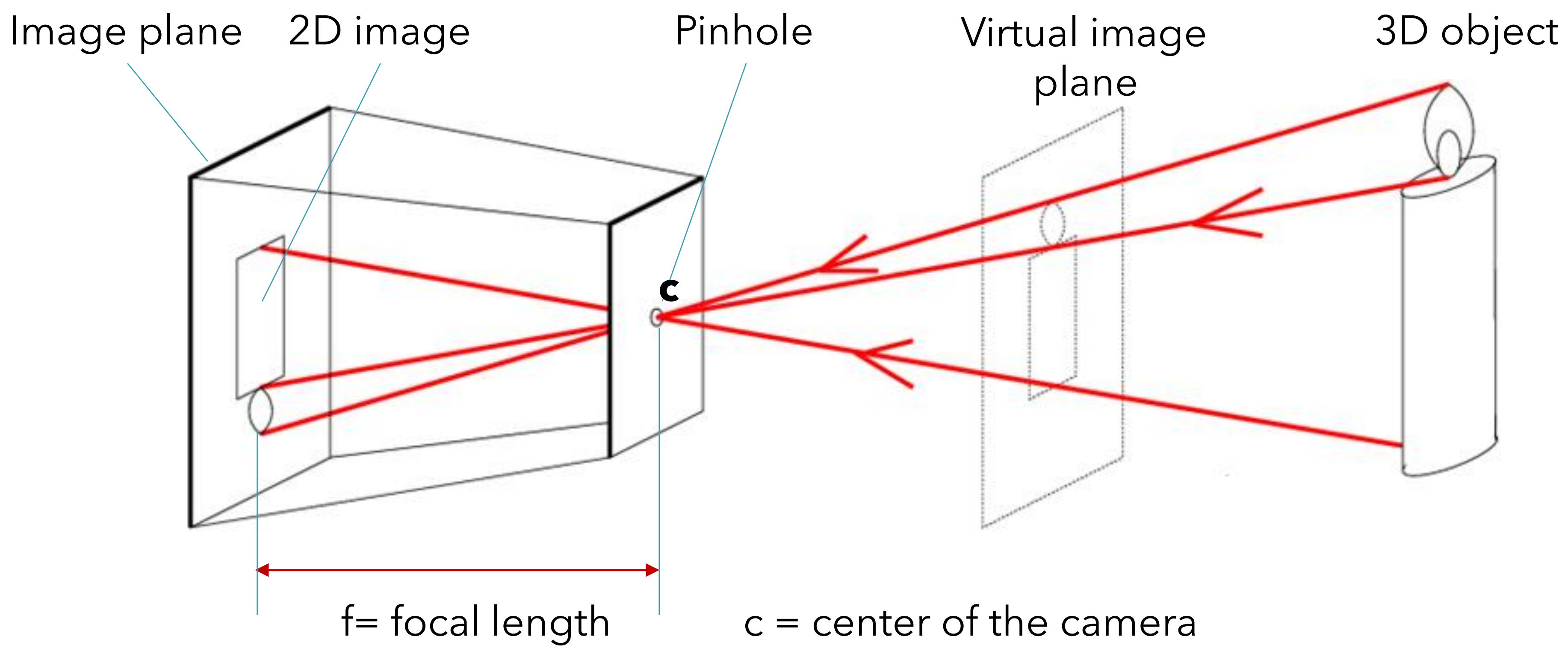
PINHOLE CAMERA MODEL

- The pinhole camera model is the simplest mathematical model that can be applied to a lot of real photography devices.
- The pinhole camera model is a simplified representation of how light travels through a small aperture to form an image on a photosensitive surface.
- The pinhole camera model provides a straightforward and intuitive way to understand the fundamental principles of image formation.



- In computer vision, the pinhole camera model is used for camera calibration (finding its intrinsic parameters-focal length and optical center, and extrinsic parameters-position and orientation of the camera). Calibration is crucial for tasks such as 3D reconstruction and object tracking.

PINHOLE CAMERA MODEL



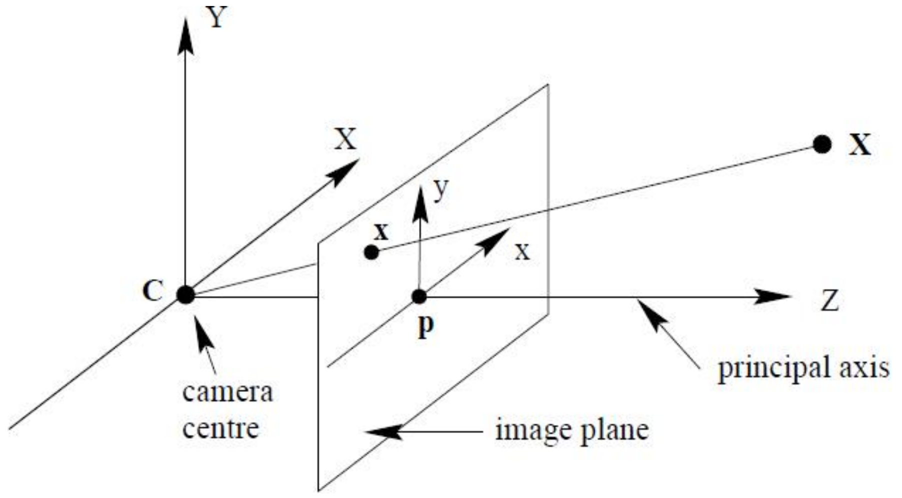
What is the transformation between images on the two image planes?

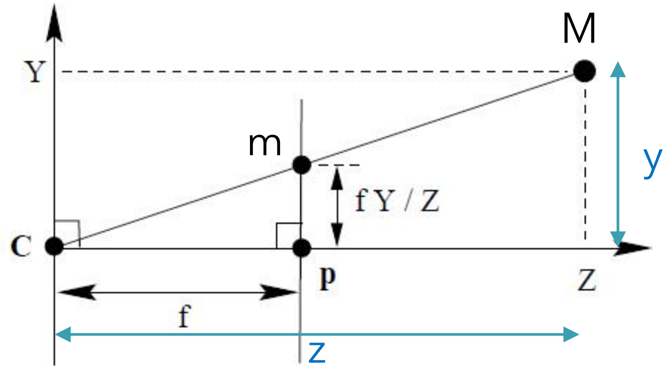
PINHOLE CAMERA MODEL

- The image plane, which is the film or medium that captures the light rays, is situated in front of the pinhole. However, in the real world, it is located behind the pinhole. This assumption makes it easier to model the projection as we don’t have to worry about inverting an image.
- All the light rays from different points converge at the pinhole, which can also be called the center of projection or the camera center.
- The idea is that the image of a point is the projection of that point on the image plane, or where the line from the camera center to the point intersects the image plane.
- What we want is a one-to-one correspondence between the points in the world and the pixels in the film.

PINHOLE CAMERA MODEL

- The pinhole camera model defines the geometric relationship between a 3D point and its 2D corresponding projection onto the image plane → perspective projection.
- 3D point $P = (X, Y, Z)^T$ projected to 2D image $p = (x, y)^T$





PINHOLE CAMERA MODEL

- Simplest form of perspective projection

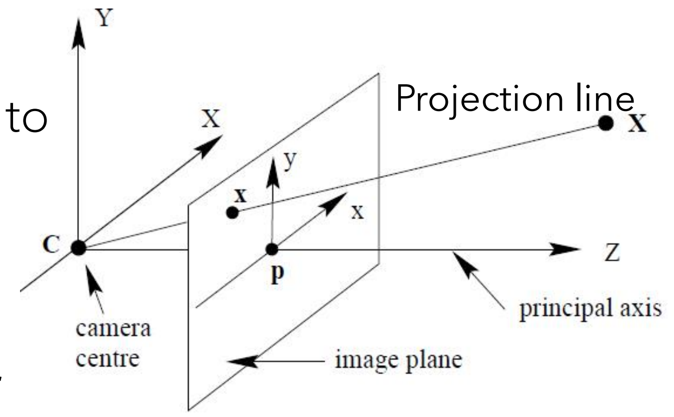
$$\frac{x}{f} = \frac{X}{Z} \Rightarrow x = f\frac{X}{Z}; \frac{y}{f} = \frac{Y}{Z} \Rightarrow y = f\frac{Y}{Z}$$

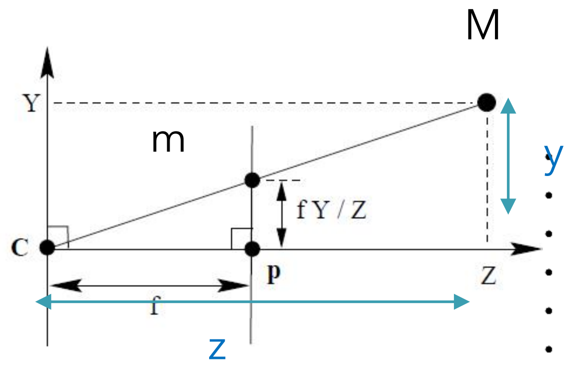
$$(x, y, 1)^T = (Xf, Yf, Z)^T$$

- Change of unit: physical measurements to pixels
- k, l: scale parameters (pixels/mm)
- f : focal length (mm). Denote: $\alpha = kf$, $\beta = lf$

$$x = kf\frac{X}{Z}, y = lf\frac{Y}{Z}$$

$$y = \alpha\frac{Y}{Z}, y = \beta\frac{Y}{Z}$$



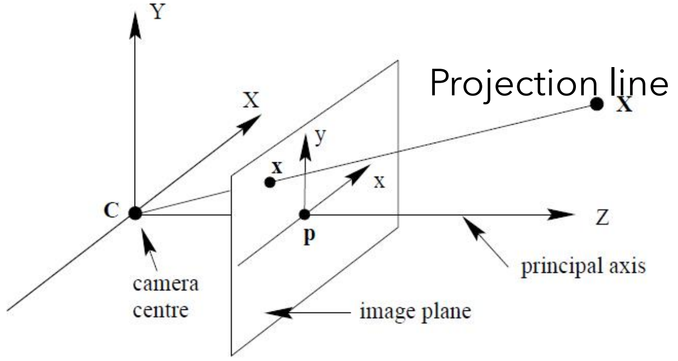


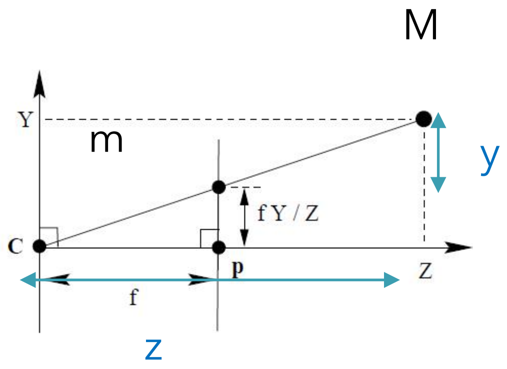
The coordinate system [x y z] centered at the pinhole P such that the axis z is perpendicular to the image plane is camera/reference coordinate system,
The line defined by *C* and *P* is called the principal/ optical axis of the camera system

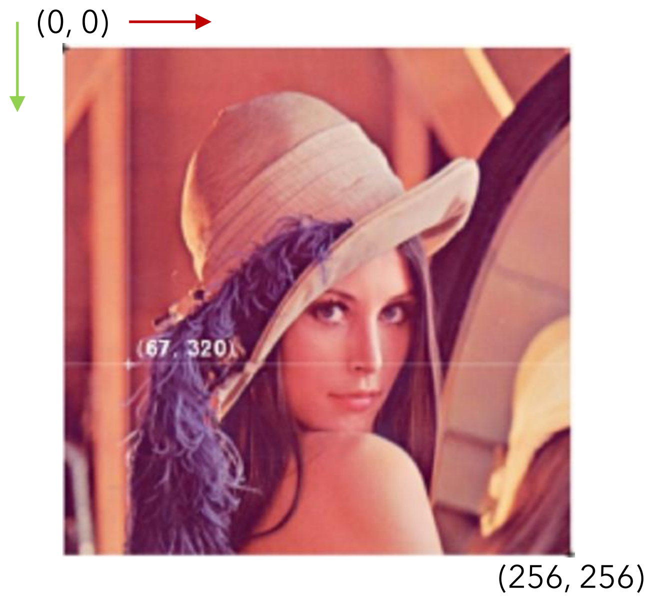
PINHOLE CAMERA MODEL

- Change of coordinate system
  - Image plane coordinates have origin at image center
  - Digital image coordinates have origin at top-left corner
  - Image center (principal point): (c_x, c_y)

$$x = \alpha\frac{X}{Z} + c_x, y = \beta\frac{Y}{Z} + c_y$$







HOMOGENEOUS COORDINATES

- Homogeneous coordinates are a mathematical tool used in computer graphics, computer vision, and other fields to represent points, vectors, and transformations in projective spaces. The term "homogeneous" implies that these coordinates are defined up to a scale factor.
- In homogeneous coordinates, a point is represented as (x, y, w), where (x, y) are the Cartesian coordinates, and w is a non-zero scale factor. Point in Cartesian is a ray in Homogeneous.
- Conversion between Cartesian and Homogeneous Coordinates

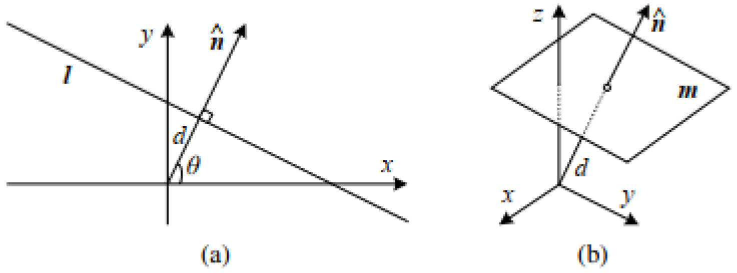
$$(x,y) \rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, (x,y,z) \rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}; \begin{bmatrix} x \\ y \\ w \end{bmatrix} \rightarrow \left(\frac{x}{w}, \frac{y}{w}\right), \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \rightarrow \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)$$

HOMOGENEOUS COORDINATES

- Projective Space: Homogeneous coordinates are particularly useful in projective geometry and projective spaces. Projective geometry deals with the properties of geometric objects that remain invariant under projective transformations
- Affine and Projective Transformations: Affine transformations (translation, rotation, scaling, and shearing) can be represented using 3x3 (2D trans formation) or 3x3 (3D transformation) matrices in homogeneous coordinates.
- Points at Infinity: In homogeneous coordinates, points at infinity can be represented naturally. For example, the point (x, y, 0) represents a point at infinity along the line through (x, y, 1).
- Homogeneous coordinates are particularly useful in projective geometry and projective spaces.

BASIC GEOMETRY IN HOMOGENEOUS COORDINATES

- 2D lines can also be represented using homogeneous coordinates $\tilde{I} = (a, b, c)$. The corresponding line equation is $\mathrm{x}.\tilde{I} = ax + by + c = 0$
- We can normalize the line equation vector so that $I = (n_x, n_y, d) = (\hat{n}, d), \|\hat{n}\| = 1$
- We can also express $\hat{n}$ as a function of rotation angle θ, $\hat{n} = (n_x, n_y) = (cos\theta, sin\theta)$



a) 2D line equation and (b) 3D plane equation, expressed in terms of the normal $\hat{n}$ and the distance to the origin *d*.

BASIC GEOMETRY IN HOMOGENEOUS COORDINATES

- Line of two points (x_i, y_i) and (x_j, y_j) given by cross product:

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \times \begin{bmatrix} x_j \\ y_j \\ 1 \end{bmatrix} = \begin{pmatrix} y_i * 1 - 1 * y_j \\ 1 * x_j - x_i * 1 \\ x_i * y_j - y_i * x_j \end{pmatrix}$$

- Intersection point of two lines given by cross product of the lines

$$q_{ij} = line_i \times line_j$$

PINHOLE CAMERA MODEL

- Using homogeneous coordinates, we can formulate $x = \alpha\frac{X}{Z} + c_x, y = \beta\frac{Y}{Z} + c_y$ in Cartesian coordinate system $\left(\frac{\alpha X}{Z}, \frac{\beta Y}{Z}\right) \Rightarrow \begin{bmatrix}\alpha X\\ \beta Y\\ Z\end{bmatrix}, \left(\frac{c_x}{1}, \frac{c_y}{1}\right) \Rightarrow \begin{bmatrix}c_x\\ c_y\\ 1\end{bmatrix}$ by

$$p = \begin{bmatrix}x\\ y\\ Z\end{bmatrix} = \begin{bmatrix}\alpha X + c_x\\ \beta Y + c_y\\ Z\end{bmatrix} = \begin{bmatrix}\alpha & 0 & c_x & 0\\ 0 & \beta & c_y & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}X\\ Y\\ Z\\ 1\end{bmatrix} = \begin{bmatrix}\alpha & 0 & c_x & 0\\ 0 & \beta & c_y & 0\\ 0 & 0 & 1 & 0\end{bmatrix}P = MP$$

- We can represent the relationship between a point in 3D space and its image coordinates by a matrix vector relationship.

PINHOLE CAMERA MODEL

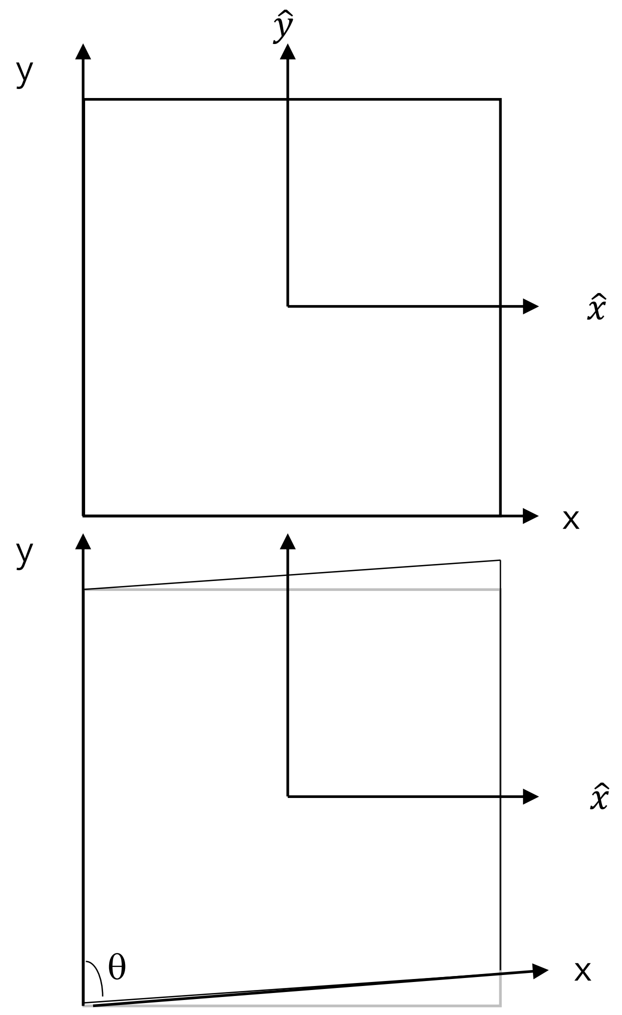
- We can decompose this transformation a bit further into

$$p = MP = \begin{bmatrix} \alpha & 0 & c_x \\ 0 & \beta & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} P$$

$$= \begin{bmatrix} \alpha & 0 & c_x \\ 0 & \beta & c_y \\ 0 & 0 & 1 \end{bmatrix} [\mathrm{I} \quad 0]\, P = K[\mathrm{I} \quad 0]\, P$$

- The matrix K is often referred to as the camera matrix. It is also called intrinsic matrix.

PINHOLE COMPLETE CAMERA MATRIX



- Image frame may not be exactly rectangular due to sensor manufacturing errors. Two parameters are currently missing: skewness and distortion (ignored)
  - θ: skew angle between x- and y-axis (normally θ=90 degree).

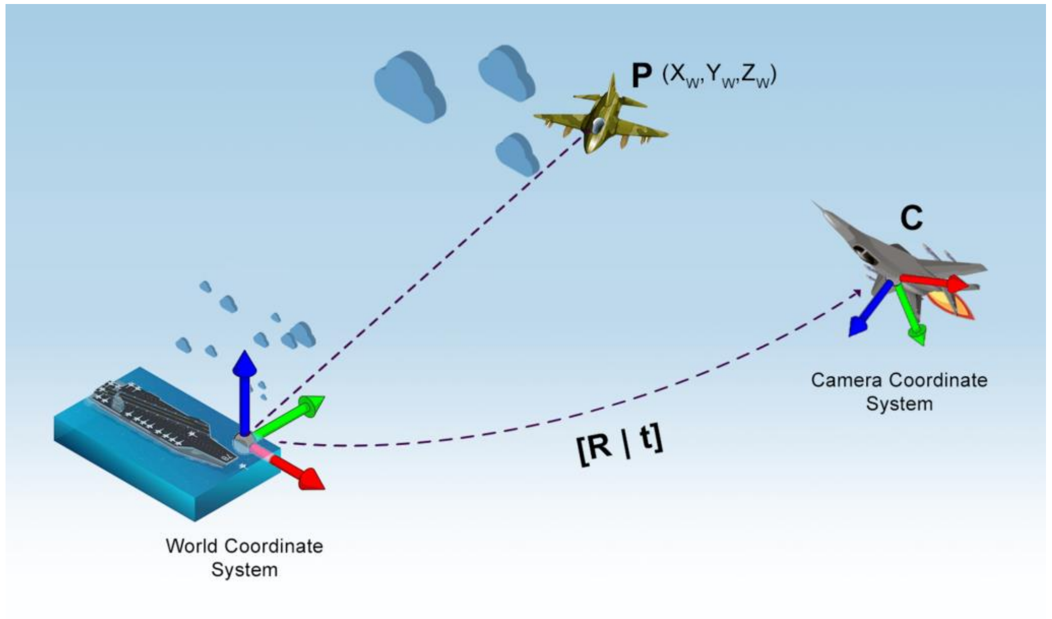
$$K = \begin{bmatrix} \alpha & \alpha cot\theta & c_x \\ 0 & \frac{\beta}{sin\theta} & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

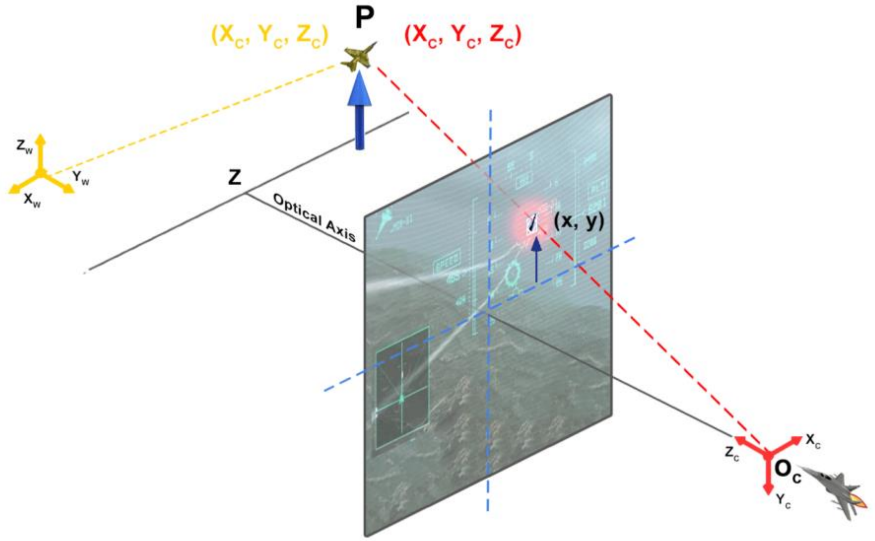
- Most cameras have zero-skew (θ=90).
- The camera matrix K has 5 degrees of freedom: k, f for focal length, (c_x, c_y) for offset and θ for skew.

WORLD COORDINATE

- We have described a mapping between a point P in the 3D camera reference system to a point p in the 2D image plane.
- But what if the information about the 3D world is available in a different coordinate system.
- We need to include an additional transformation that relates points from the world reference system to the camera reference system.
- This transformation is captured by a rotation matrix R and translation vector T.

WORLD COORDINATE





- The 3D world coordinate is projected to image plane through 2 transformations:
  - Translate and rotate camera coordinate system to 3D world coordinate system
  - Project 3D camera point P to image plane
- This transformation is captured by a rotation matrix R and translation vector T.

CAMERA TRANSLATION & ROTATION

- Translation

$$p = K[\mathrm{I} \quad t]\, P = \begin{bmatrix} \alpha & 0 & c_x \\ 0 & \beta & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_z \\ 0 & 0 & 1 & t_z \end{bmatrix} P$$

- Rotation

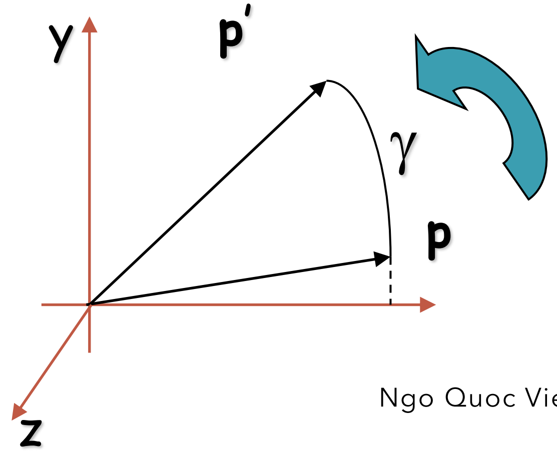
$$p = K[\mathrm{R} \quad t]\, P = \begin{bmatrix} \alpha & 0 & c_x \\ 0 & \beta & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_z \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} P$$

- These parameters R and T are known as the extrinsic parameters because they are external to and do not depend on the camera

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & -sin\alpha \\ 0 & sin\alpha & cos\alpha \end{bmatrix}$$

$$R_y(\beta) = \begin{bmatrix} cos\beta & 0 & sin\beta \\ 0 & 1 & 0 \\ -sin\beta & 0 & cos\beta \end{bmatrix}$$

$$R_z(\gamma) = \begin{bmatrix} cos\gamma & -sin\gamma & 0 \\ sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



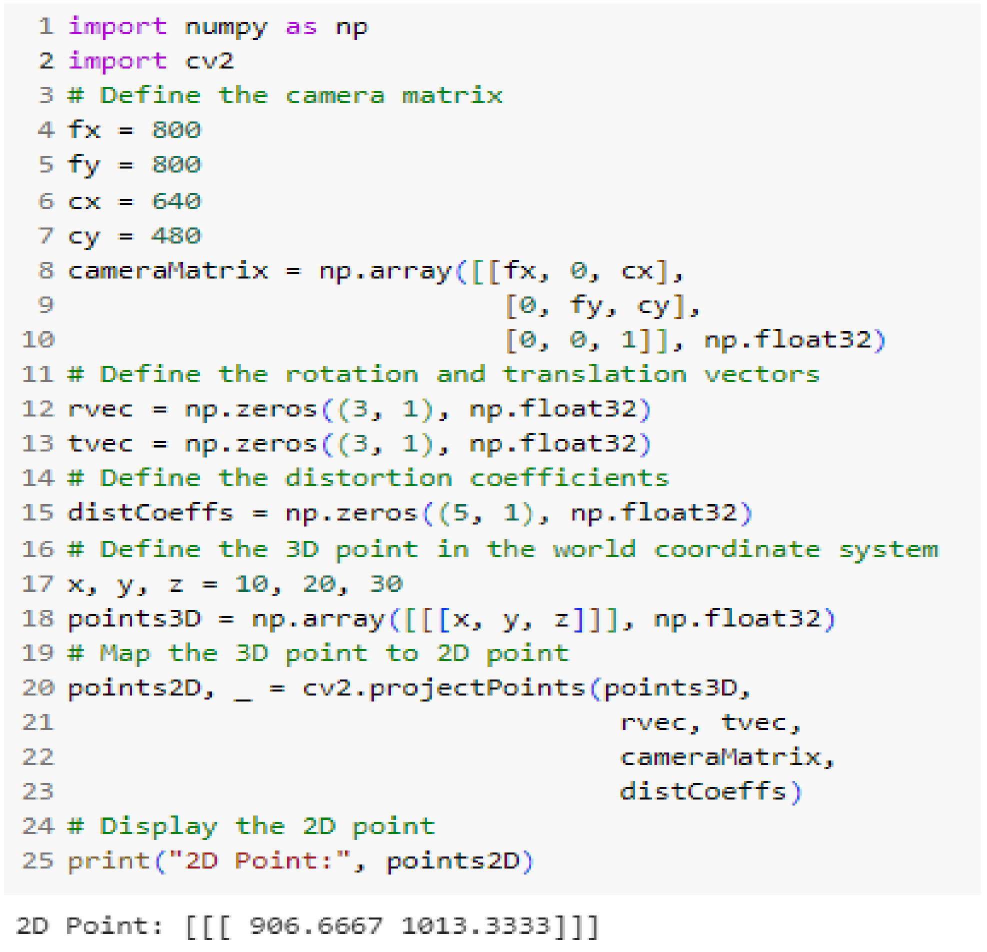
THE EXTRINSIC PARAMETERS

$$p = K[R \quad t]\, P = MP$$

- The full projection matrix M consists of the two types of parameters introduced above: **intrinsic** and **extrinsic** parameters.
- All parameters contained in the camera matrix K are the intrinsic parameters, which change as the type of camera changes.
- The extrinsic paramters include the rotation and translation, which do not depend on the camera's build.
- Overall, we find that the 3 × 4 projection matrix M has 11 degrees of freedom: 5 for intrinsic, 3 for extrinsic rotation and 3 for extrinsic translation.

MAPPING COORDINATES FROM 3D TO 2D

- The most important intrinsic parameter is the focal length, which determines how much the camera can zoom in or out. The principal point coordinates define the center of the image, where the optical axis intersects with the image plane. The image sensor size determines the field of view of the camera.
- The extrinsic parameters of a camera describe its position and orientation in 3D space. These parameters include the rotation and translation vectors, which determine the camera's position and orientation relative to the 3D scene being imaged
- When we perform a perspective projection of 3D points onto a 2D plane, we need to take into account both the intrinsic and extrinsic parameters of the camera



```python
import numpy as np
import cv2
# Define the camera matrix
fx = 800
fy = 800
cx = 640
cy = 480
cameraMatrix = np.array([[fx, 0, cx],
                         [0, fy, cy],
                         [0, 0, 1]], np.float32)
# Define the rotation and translation vectors
rvec = np.zeros((3, 1), np.float32)
tvec = np.zeros((3, 1), np.float32)
# Define the distortion coefficients
distCoeffs = np.zeros((5, 1), np.float32)
# Define the 3D point in the world coordinate system
x, y, z = 10, 20, 30
points3D = np.array([[[x, y, z]]], np.float32)
# Map the 3D point to 2D point
points2D, _ = cv2.projectPoints(points3D,
                                rvec, tvec,
                                cameraMatrix,
                                distCoeffs)
# Display the 2D point
print("2D Point:", points2D)
```

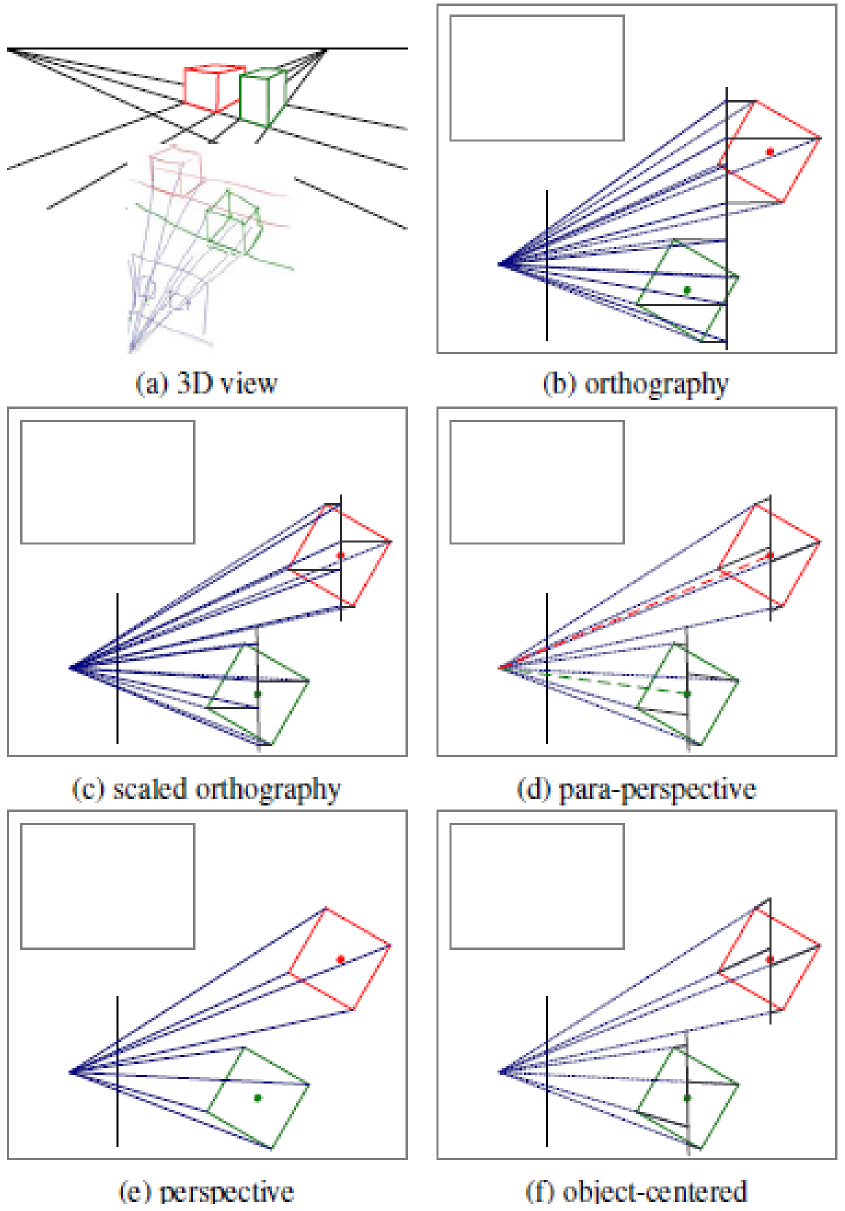
```
2D Point: [[[ 906.6667 1013.3333]]]
```

PINHOLE CAMERA MODEL

- When the **camera matrix is equal to the identity matrix**, it means that there is **no distortion or scaling in the image**. This is a special case known as the pinhole camera model, where light passes through a single point (the pinhole) to form an inverted image on the opposite side of the camera.
- Distortion coefficients are used to correct lens distortion, which can cause images to appear warped or curved. When the distortion coefficients are set to zero, it means that there is no distortion to correct for.
- In some cases, it may be appropriate to use an identity camera matrix and zero distortion coefficients. This is typically done when the camera has already been calibrated and the distortion is minimal, or when a pinhole camera model is appropriate for the task at hand.

3D TO 2D PROJECTIONS

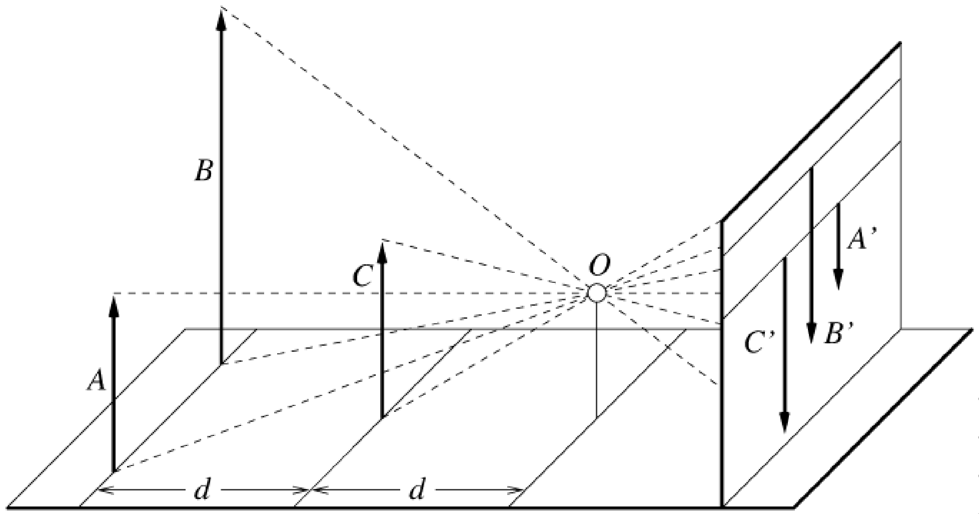
- Onthography: drop z component, $\mathbf{p} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} P.$
- Scaled-onthography: $p = [sI_{2\times 2}\ 0]P$
- Para-perspective
- Perspective: the most commonly use projection in computer graphics and computer vision.
- Object-centered


(a) 3D view
(b) orthography
(c) scaled orthography
(d) para-perspective
(e) perspective
(f) object-centered

PERSPECTIVE PROJECTION

- The most commonly used projection in computer graphics and computer vision is true 3D perspective.
- Points are projected onto the image plane by dividing them by their z component: $p = \begin{bmatrix} X/Z \\ Y/Z \\ 1 \end{bmatrix}$. In homogeneous coordinates, we have

$$p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} P$$



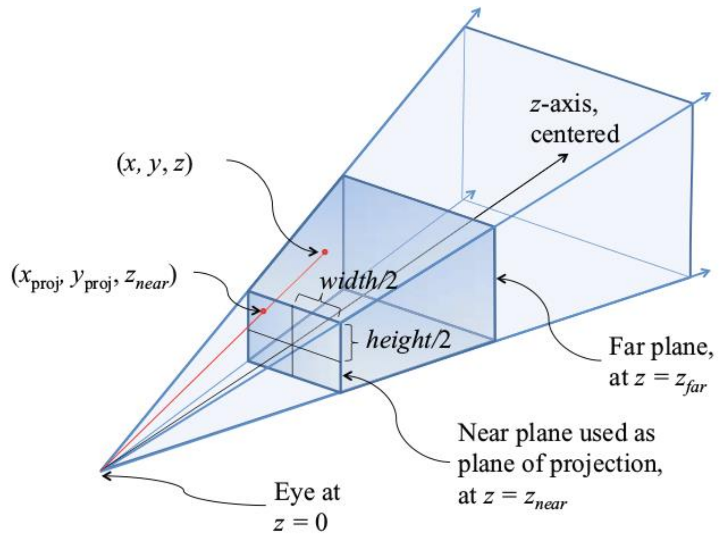
- After projection, it is not possible to recover the distance of the 3D point from the image, which makes sense for a 2D imaging sensor.

PERSPECTIVE PROJECTION

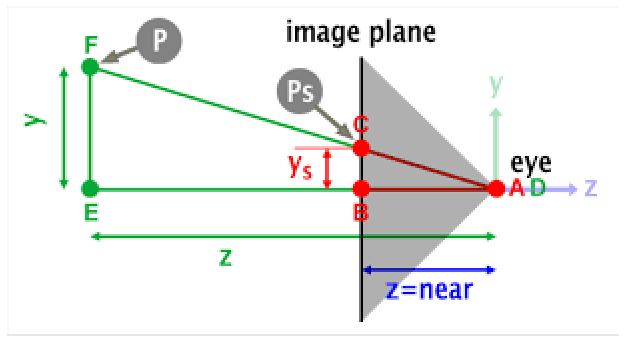
- A form often seen in computer graphics systems is a two-step projection that
  - Projects 3D coordinates into normalized device coordinates $(X, Y, Z) \in [-1,1] \times [-1,1] \times [0,1]$.
  - Rescales these coordinates to integer pixel coordinates using a viewport transformation
- The (initial) perspective projection is then represented using 4x4 matrix

$$\mathrm{p} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{-Z_{far}}{Z_{range}} & \dfrac{Z_{near}Z_{far}}{Z_{range}} \\ 0 & 0 & 0 & 0 \end{bmatrix} P, \qquad Z_{range} = Z_{far} - Z_{near}$$

where Z_{near} and Z_{far} are the near and far z clipping planes



(x, y, z)
z-axis, centered
$(x_{proj}, y_{proj}, z_{near})$
$width/2$
$height/2$
Far plane, at $z = z_{far}$
Near plane used as plane of projection, at $z = z_{near}$
Eye at $z = 0$

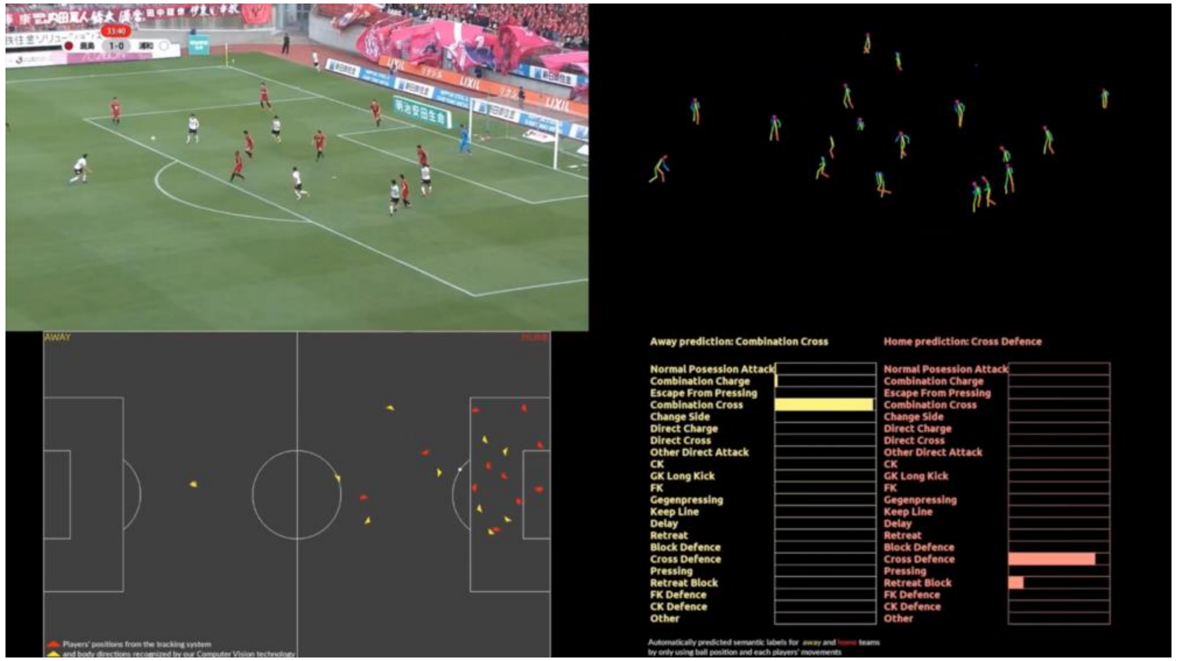


PERSPECTIVE PROJECTION

- Transform the world point $(X_w, Y_w, Z_w)$with optical center W to a new coordinate system with the camera's optical center C as the (0, 0, 0) origin. This is done using a rigid body transformation, consisting of rotation (R) and translation (t).
- Project the camera point (X_c, Y_c, Z_c) onto the optical sensor, creating new coordinates (x, y, z) in the same coordinate system. This is achieved using the camera's intrinsic parameter matrix *K*. The optical sensor is often referred to as the "image plane" or "image frame".
- Normalize (x, y, z) to pixel coordinates (u, v) by dividing by z and and adjusting the origin of the image.
- Python Code: *Perspective Projection, Mapping coordinates from 3d to 2d* in https://colab.research.google.com/

PERSPECTIVE TRANSFORMATION

- Mapping the player positions which are in the camera space to the actual soccer field coordinate and generate graph with player positions relative to the soccer field.
- What we need was an output similar to the left bottom one in the figure.

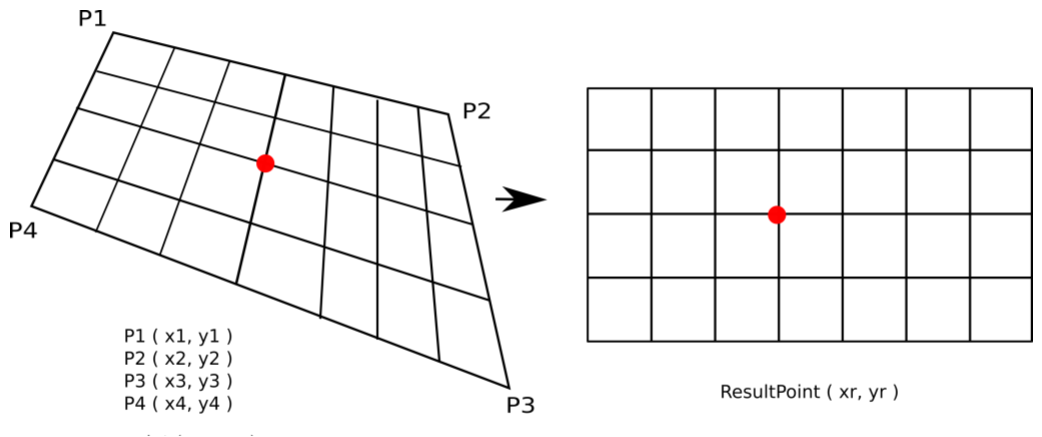

AWAY
HOME
Players' positions from the tracking system and body directions recognized by our Computer Vision technology
Away prediction: Combination Cross
Home prediction: Cross Defence
Normal Posession Attack
Combination Charge
Escape From Pressing
Combination Cross
Change Side
Direct Charge
Direct Cross
Other Direct Attack
CK
GK Long Kick
FK
Gegenpressing
Keep Line
Delay
Retreat
Block Defence
Cross Defence
Pressing
Retreat Block
FK Defence
CK Defence
Other
Automatically predicted semantic labels for away and home teams by only using ball position and each players' movements

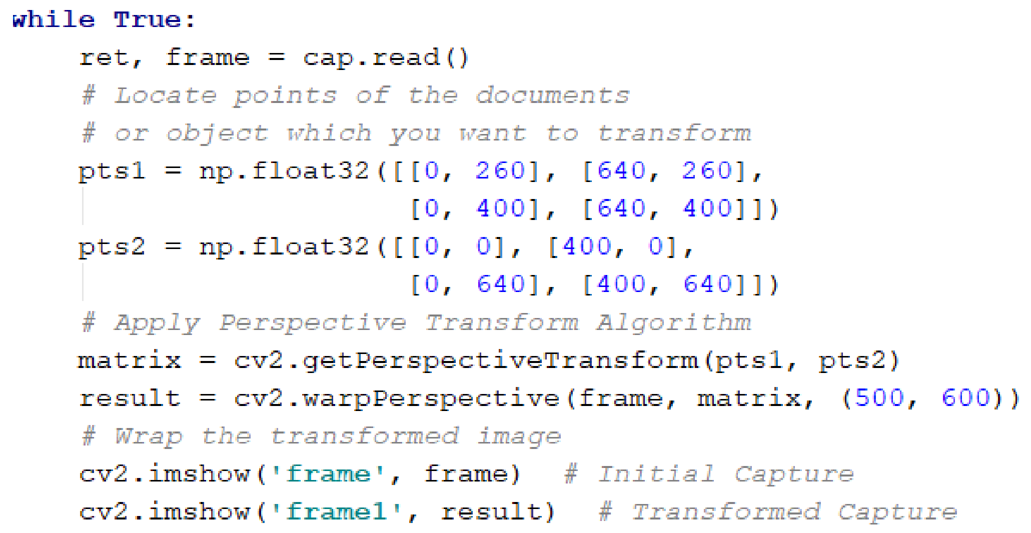
Picture from: https://naadispeaks.blog/2021/08/31/perspective-transformation-of-coordinate-points-on-polygons/

- Mathematics of Two- and Three- Point Perspective: https://people.eecs.berkeley.edu/~barsky/perspective.html

PERSPECTIVE TRANSFORMATION

- If we know the **frame-space** location coordinates of **the 4 corners of the field**, we can easily transform any point inside that polygon for a given coordinate space. It is perspective transformation.
- In Perspective Transformation, we can change the perspective of a given image or video for getting better insights into the required information.
- We use **cv2.getPerspectiveTransform** and then **cv2.warpPerspective**.
- Perspective transformation is a broader term (a wider range of operations beyond just projection) that encompasses various techniques for transforming or mapping objects from one perspective to another.



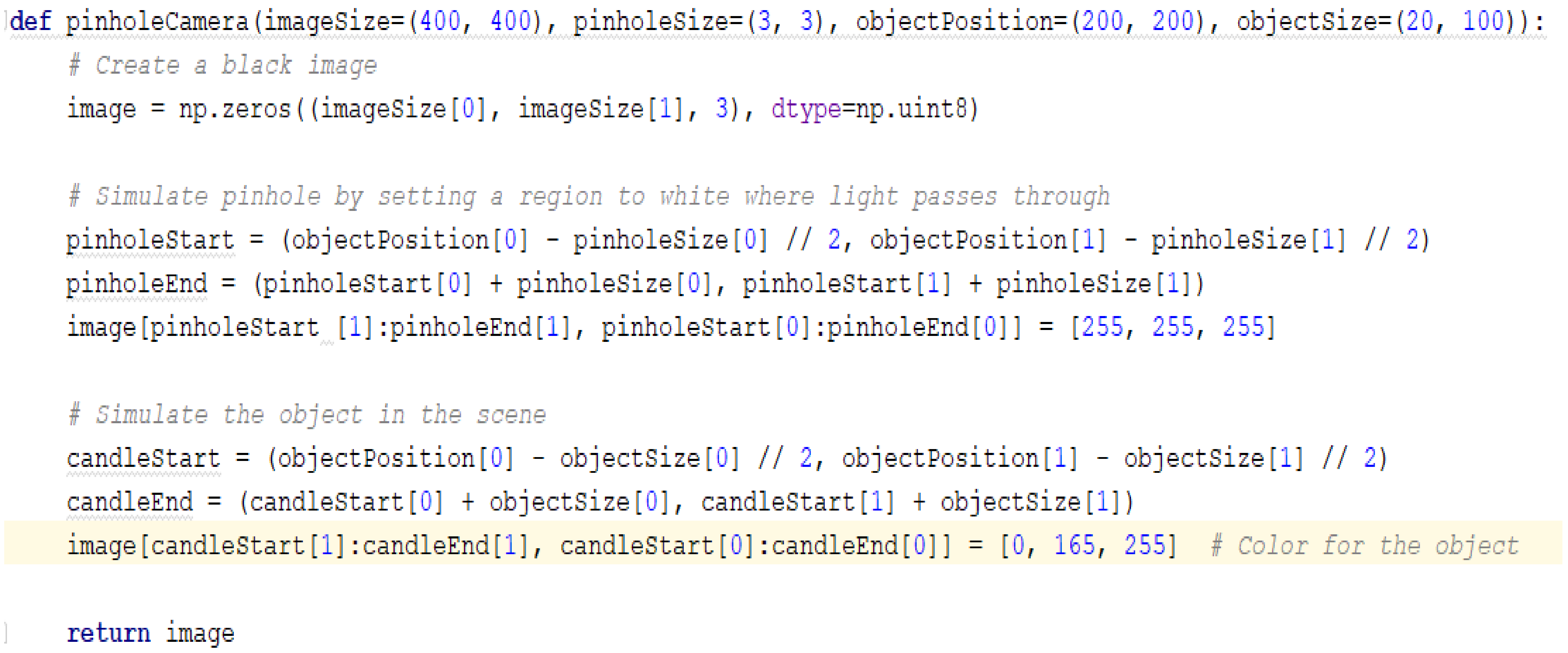


```
while True:
    ret, frame = cap.read()
    # Locate points of the documents
    # or object which you want to transform
    pts1 = np.float32([[0, 260], [640, 260],
                       [0, 400], [640, 400]])
    pts2 = np.float32([[0, 0], [400, 0],
                       [0, 640], [400, 640]])
    # Apply Perspective Transform Algorithm
    matrix = cv2.getPerspectiveTransform(pts1, pts2)
    result = cv2.warpPerspective(frame, matrix, (500, 600))
    # Wrap the transformed image
    cv2.imshow('frame', frame)  # Initial Capture
    cv2.imshow('frame1', result)  # Transformed Capture
```

IMAGE FORMATION WITH PINHOLE

- Light entry through the small pinhole.
- Inverted Image Formation: The limited rays of light passing through the pinhole create an inverted image on the opposite side of the pinhole
- Image Projection: The inverted image is projected onto the photosensitive surface (film or image sensor) .
- Please refer to Python code to simulate pinhole camera.

IMAGE FORMATION WITH PINHOLE



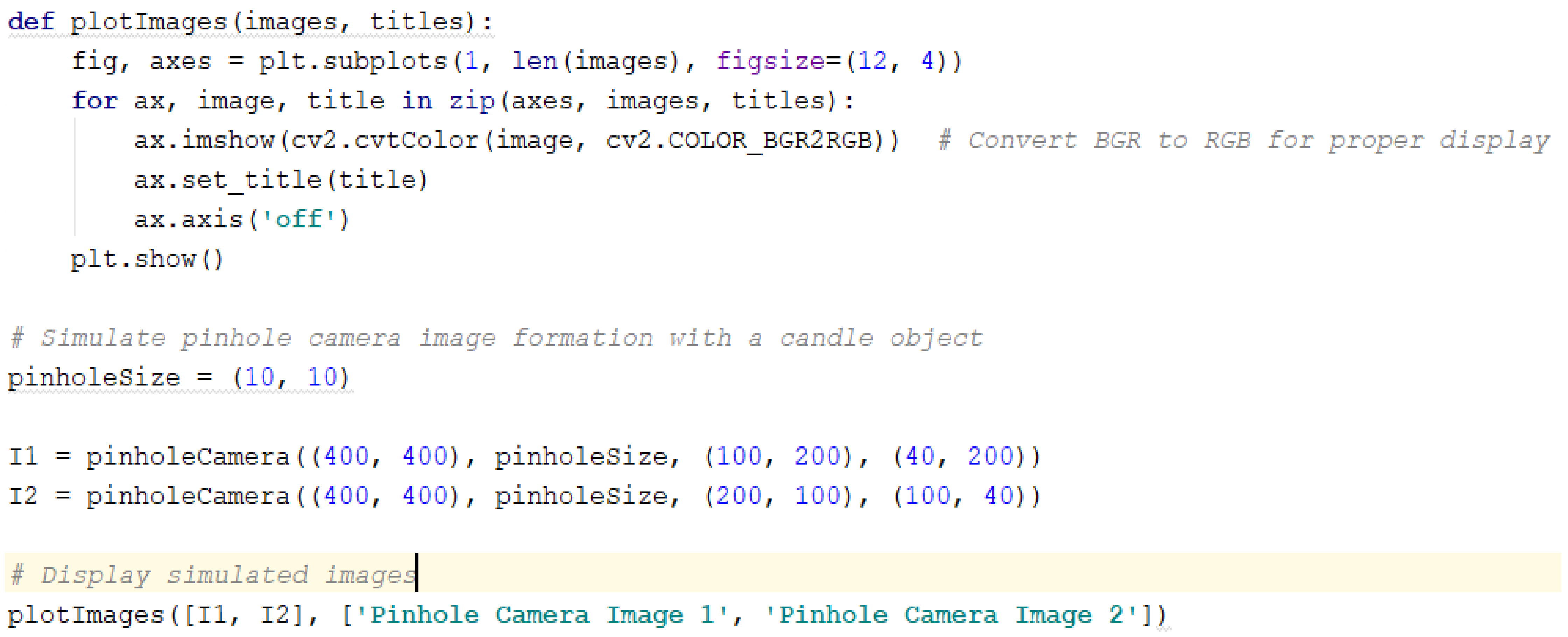
```python
def pinholeCamera(imageSize=(400, 400), pinholeSize=(3, 3), objectPosition=(200, 200), objectSize=(20, 100)):
    # Create a black image
    image = np.zeros((imageSize[0], imageSize[1], 3), dtype=np.uint8)

    # Simulate pinhole by setting a region to white where light passes through
    pinholeStart = (objectPosition[0] - pinholeSize[0] // 2, objectPosition[1] - pinholeSize[1] // 2)
    pinholeEnd = (pinholeStart[0] + pinholeSize[0], pinholeStart[1] + pinholeSize[1])
    image[pinholeStart [1]:pinholeEnd[1], pinholeStart[0]:pinholeEnd[0]] = [255, 255, 255]

    # Simulate the object in the scene
    candleStart = (objectPosition[0] - objectSize[0] // 2, objectPosition[1] - objectSize[1] // 2)
    candleEnd = (candleStart[0] + objectSize[0], candleStart[1] + objectSize[1])
    image[candleStart[1]:candleEnd[1], candleStart[0]:candleEnd[0]] = [0, 165, 255]  # Color for the object

    return image
```

IMAGE FORMATION WITH PINHOLE



```python
def plotImages(images, titles):
    fig, axes = plt.subplots(1, len(images), figsize=(12, 4))
    for ax, image, title in zip(axes, images, titles):
        ax.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))  # Convert BGR to RGB for proper display
        ax.set_title(title)
        ax.axis('off')
    plt.show()

# Simulate pinhole camera image formation with a candle object
pinholeSize = (10, 10)

I1 = pinholeCamera((400, 400), pinholeSize, (100, 200), (40, 200))
I2 = pinholeCamera((400, 400), pinholeSize, (200, 100), (100, 40))

# Display simulated images
plotImages([I1, I2], ['Pinhole Camera Image 1', 'Pinhole Camera Image 2'])
```

WHY CAMERA CALIBRATION

- Camera calibration is a crucial process in computer vision and image processing because it helps correct distortions in images caused by the camera's optical system.
- Radial Distortion Correction: Many camera lenses introduce radial distortion, causing straight lines to appear curved. Calibration helps correct this distortion, ensuring that straight lines in the real world remain straight in the image.
- Tangential Distortion Correction: This occurs when the lens is not perfectly aligned with the image sensor. Calibration corrects tangential distortion, ensuring that points away from the center of the image are accurately represented

WHY CAMERA CALIBRATION

- Computer Vision Applications: In computer vision tasks like object recognition, tracking, and 3D reconstruction, accurate knowledge of the camera's intrinsic and extrinsic parameters is crucial. Calibration provides these parameters, allowing algorithms to map image coordinates to real-world coordinates accurately.
- Augmented Reality: In augmented reality applications, where virtual objects are overlaid onto the real world, accurate camera calibration ensures proper alignment of virtual and real-world elements.
- If given an arbitrary camera, we may or may not have access to these parameters. Therefore, can we find a way to deduce them from images.

WHY CAMERA CALIBRATION

- **3D Reconstruction**: Camera calibration is fundamental for reconstructing a three-dimensional scene from multiple images. It helps establish the relationship between the 3D world and the 2D image, allowing for accurate reconstruction
- **Robotics**: Robots often use cameras for navigation and perception. Accurate calibration is necessary for robot systems to interpret visual information correctly and make informed decisions.
- **Quality Control**: In industrial applications, where cameras are used for quality control purposes, calibration ensures that defects or features are accurately detected and measured

WHAT IS CAMERA CALIBRATION

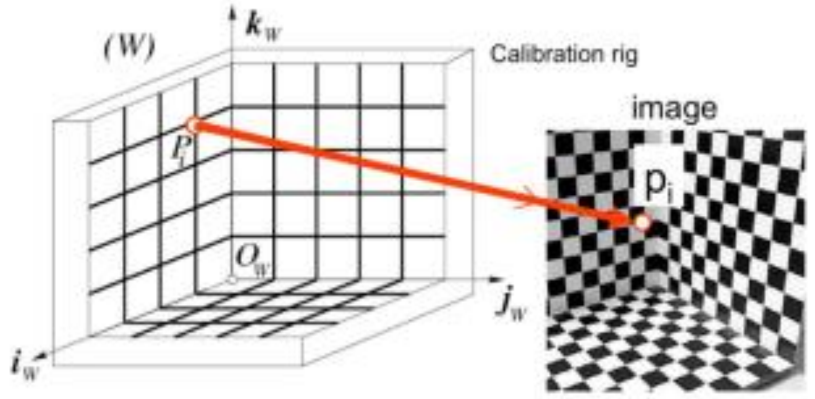
- Camera calibration is the process of determining the intrinsic and extrinsic parameters of a camera. These parameters define how a camera projects a three-dimensional (3D) scene onto a two-dimensional (2D) image.
- The calibration process typically involves **capturing images of a known calibration pattern, such as a checkerboard, from various orientations**. By analyzing the correspondences between the 3D coordinates of calibration points in the world and their 2D image coordinates, the calibration software can estimate the camera's intrinsic and extrinsic parameters.
- The calibration results enable the transformation from pixel coordinates in the image to real-world coordinates in the 3D scene.

HOW TO CALIBRATION

- We need find the intrinsic camera matrix K and the extrinsic parameters R, T by solving the

$$p = K[\mathrm{R} \quad t]\, P = MP$$

- M has 11 degrees of freedom: 5 for intrinsic, 3 for extrinsic rotation and 3 for extrinsic translation.
- This means that we need **at least 6** correspondences to solve this.



HOW TO CALIBRATION

- From the rig's known pattern, we have known points in the world reference frame $P_1, P_2, \cdots, P_n$. Finding these points in the image we take from the camera gives corresponding points in the image $p_1, p_2, \cdots, p_n$.
- We set up a linear system of equations from *n* correspondences such that for each correspondence (P_i, p_i) and camera matrix M whose rows are m_1, m_2, m_3

$$p_i = \begin{bmatrix} u_i \\ v_i \end{bmatrix} = MP_i = \begin{bmatrix} \frac{m_1 P_i}{m_3 P_i} \\ \frac{m_2 P_i}{m_3 P_i} \end{bmatrix}$$

Given n of these corresponding points, the entire linear system of equations becomes

$$\begin{aligned} u_1(m_3P_1) - m_1P_1 &= 0 \\ v_1(m_3P_1) - m_2P_1 &= 0 \\ &\vdots \\ u_n(m_3P_n) - m_1P_n &= 0 \\ v_n(m_3P_n) - m_2P_n &= 0 \end{aligned}$$

$$\begin{bmatrix} P_1^T & 0^T & -u_1P_1^T \\ 0^T & P_1^T & -v_1P_1^T \\ & \vdots & \\ P_n^T & 0^T & -u_nP_n^T \\ 0^T & P_n^T & -v_nP_n^T \end{bmatrix} \begin{bmatrix} m_1^T \\ m_2^T \\ m_3^T \end{bmatrix} = Pm = 0$$

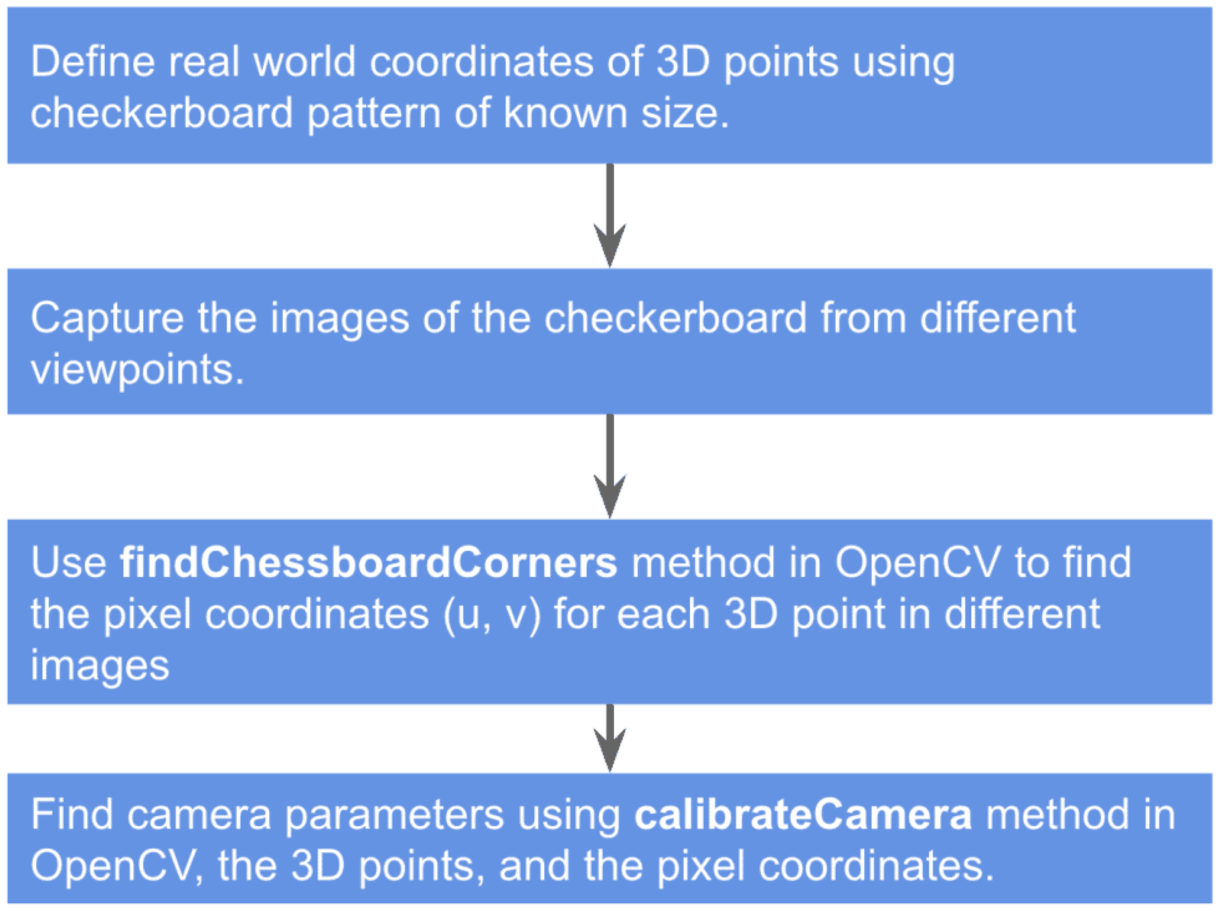
HOW TO CALIBRATION

- When $\mathbf{2n} > \mathbf{11}$ (at least 6 correspondences), our homogeneous linear system is overdetermined. **Therefore, we need linear equation system of at least** 5 and 6 **linear equations for intrinsic matrix and extrinsic parameters**.
- For such a system m = 0 is always a trivial solution. Furthemore, even if there were some other m that were a nonzero solution, then $\forall k \in R$, km is also a solution. Therefore, to constrain our solution, we complete the following minimization

$$\min_{m} \|Pm\|^2 , subject\ to\ \|m\|^2 = 1$$

- To solve this minimization problem, we simply use **singular value decomposition**. The derivation for this problem is outside the scope of this class. Please refer Section 5.3 of Hartley & Zisserman-Multiple View Geometry in CV and Section 1.3.1 of the Forsyth & Ponce textbook.

Camera Calibration Flowchart



Define real world coordinates of 3D points using checkerboard pattern of known size.

Capture the images of the checkerboard from different viewpoints.

Use **findChessboardCorners** method in OpenCV to find the pixel coordinates (u, v) for each 3D point in different images

Find camera parameters using **calibrateCamera** method in OpenCV, the 3D points, and the pixel coordinates.

Source: https://learnopencv.com/camera-calibration-using-opencv/

CAMERA CALIBRATION ALGORITHM

1. **Chessboard Setup**: Use a chessboard pattern with known square dimensions; Capture multiple images of the chessboard from different angles and distances.
2. **Image Preprocessing**: Detect chessboard corners in each image using corner detection algorithms (e.g., Harris corner detector).
3. **Corner Detection**: Find the image coordinates of the detected chessboard corners.
4. **World Coordinates**: Define the world coordinates of the chessboard corners in a 3D space; Use the known square dimensions of the chessboard.

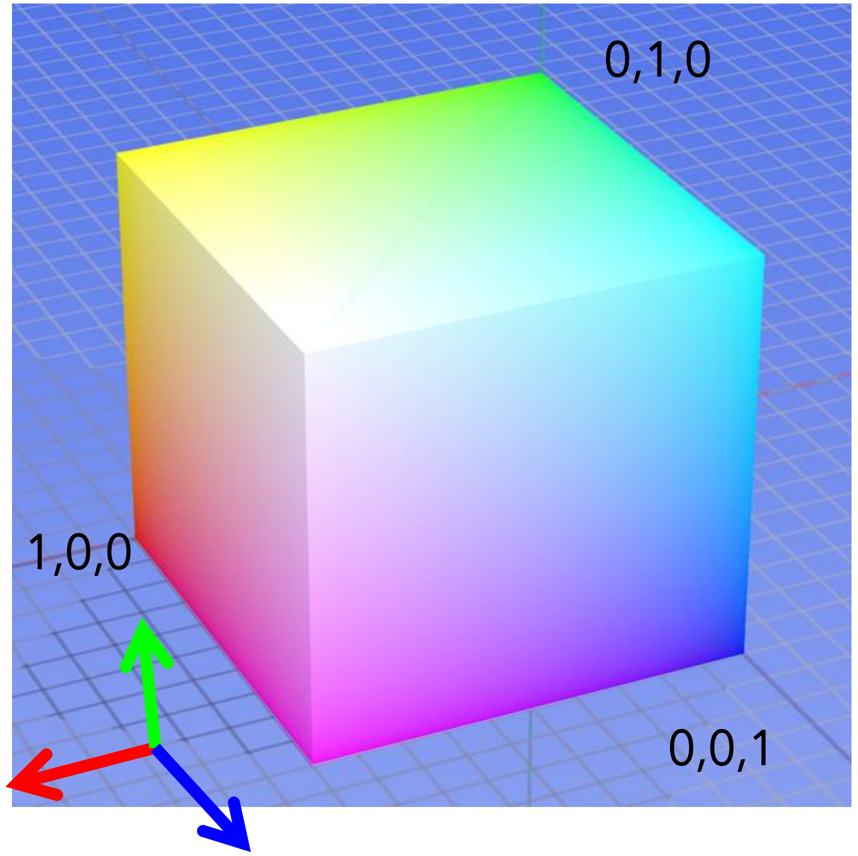
CAMERA CALIBRATION ALGORITHM

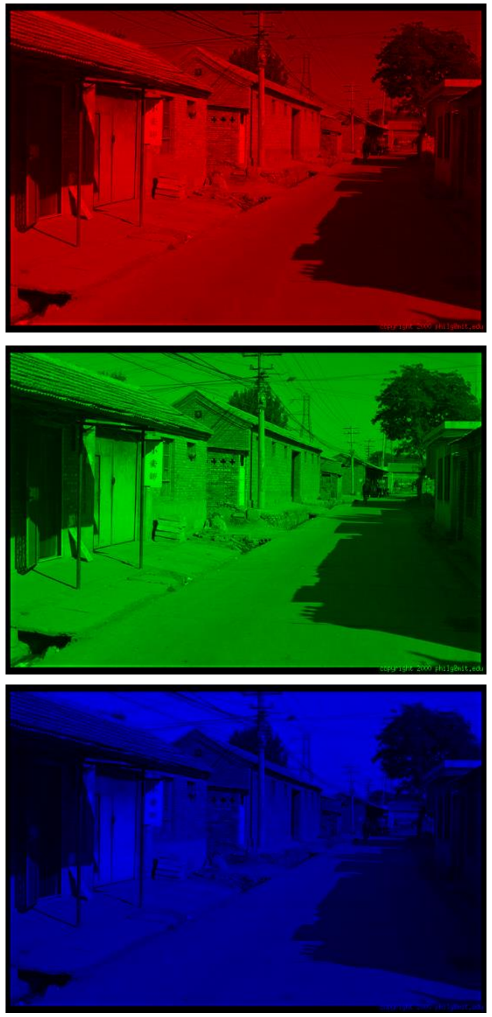
5. **Calibration Object Points**: Create a list of 3D points representing the real-world coordinates of the chessboard corners.
6. **Calibration Parameters Estimation**
   - Use the correspondences between image points and object points to estimate camera parameters;
   - Parameters include intrinsic matrix (focal length, principal point), distortion coefficients, and extrinsic parameters (rotation and translation vectors)
7. **Evaluate Calibration Quality**: Reproject the 3D world points into 2D image points using the obtained calibration parameters; Compare the reprojected points with the detected image points to assess the calibration accuracy

CAMERA CALIBRATION'S CODE

- https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html
- https://learnopencv.com/camera-calibration-using-opencv/
- https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html

COLOR SPACE-RGB

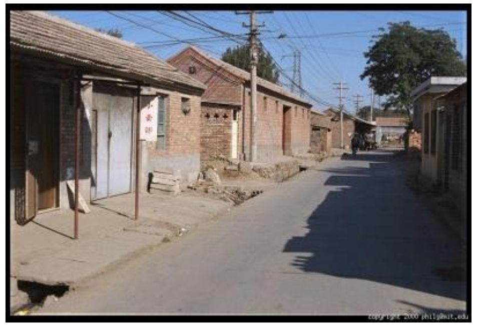




R
(G=0,B=0)

G
(R=0,B=0)

B
(R=0,G=0)

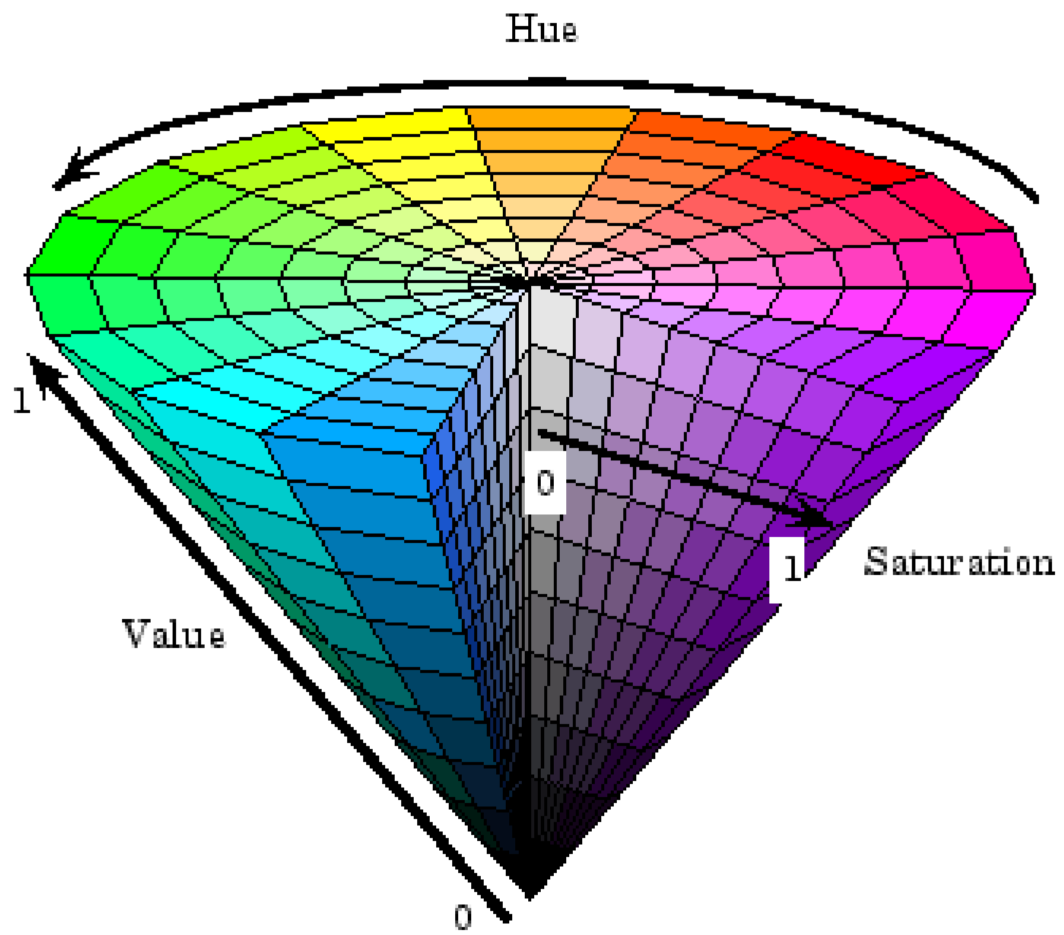


Some drawbacks

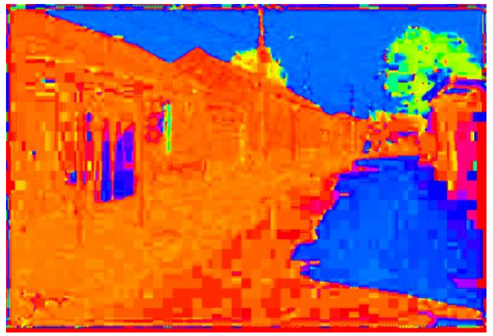
- Strongly correlated channels
- Non-perceptual

http://en.wikipedia.org/wiki/File:RGB_color_solid_cube.png

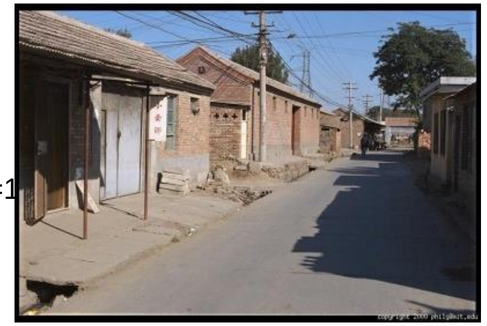
COLOR SPACES: HSV

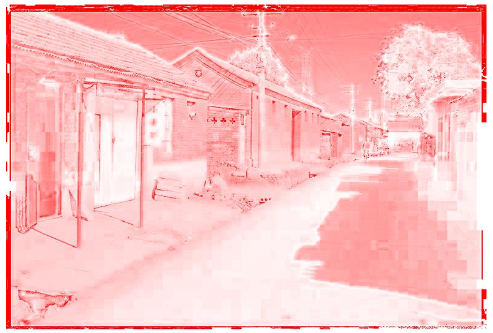


Intuitive color space

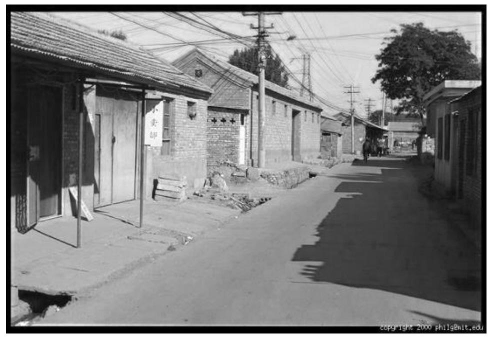


H
(S=1,V=1



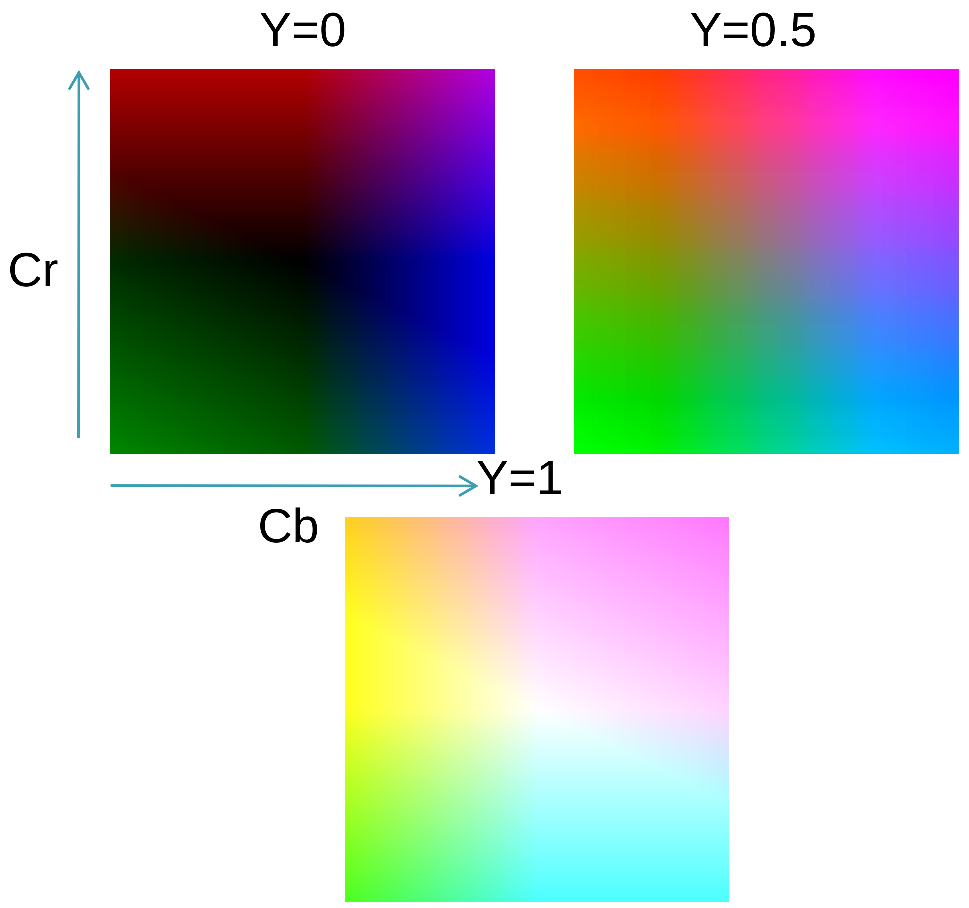


S
(H=1,V=1)

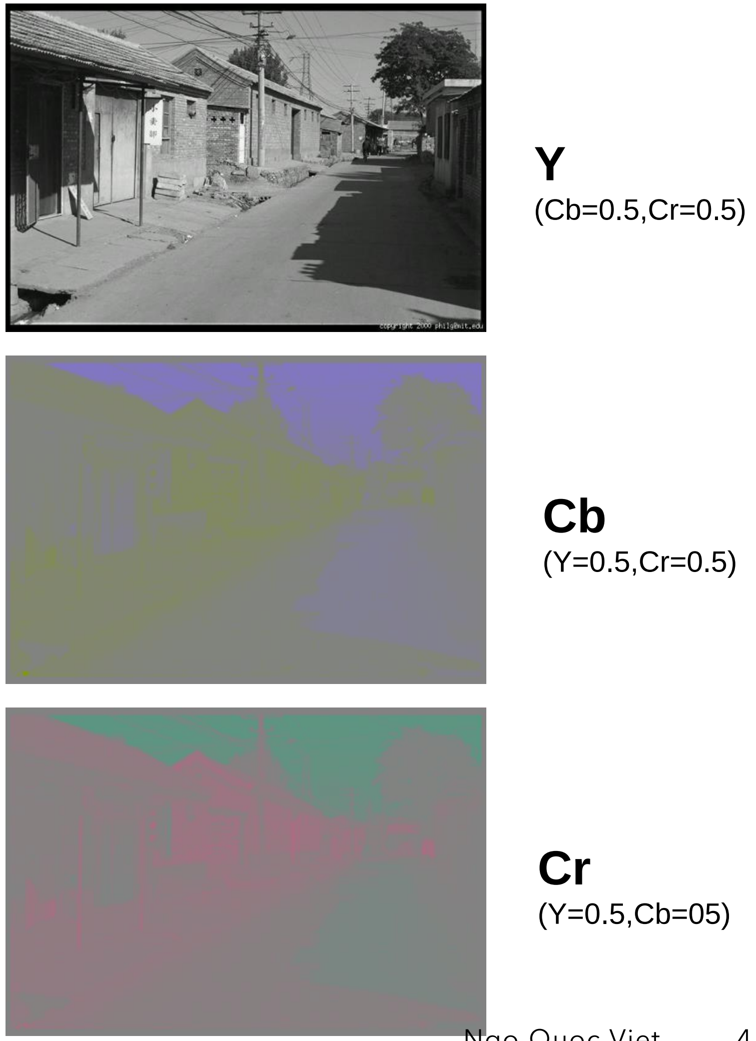


V
(H=1,S=0)

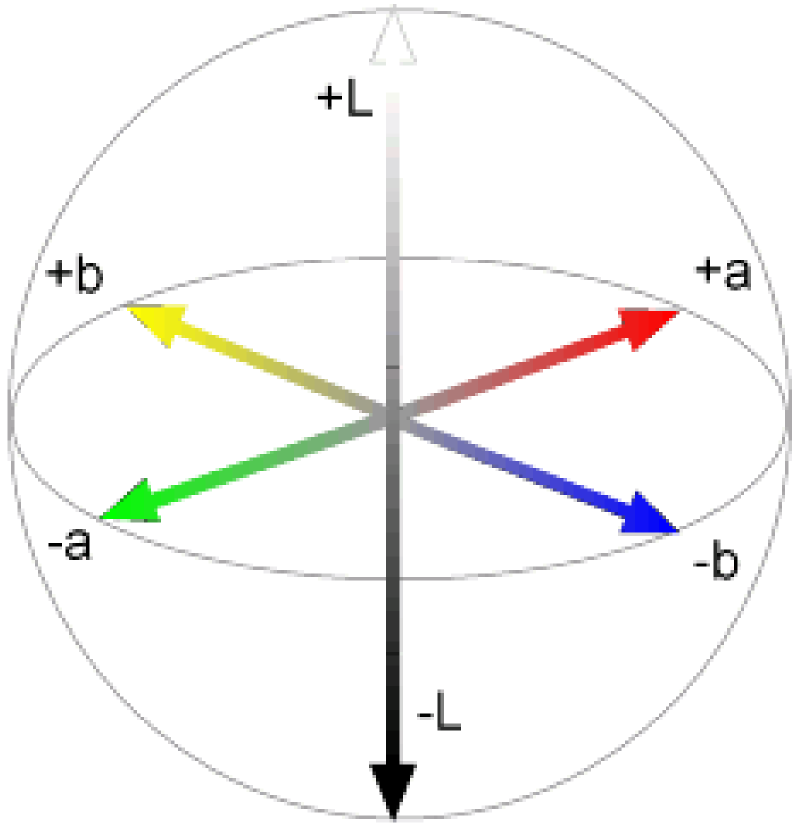
COLOR SPACES: YCBCR



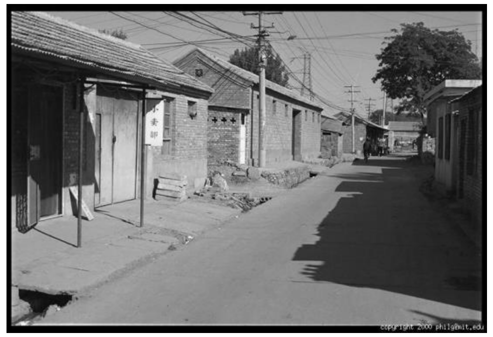
Fast to compute, good for compression, used by TV

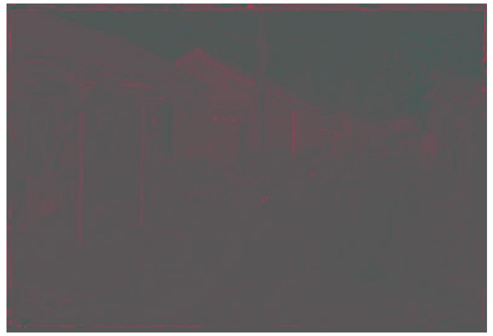


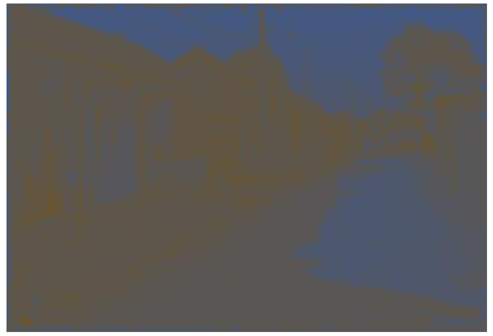
COLOR SPACES- L*A*B*



"Perceptually uniform"* color space


L
(a=0,b=0)


a
(L=65,b=0)


b
(L=65,a=0)

SUMMARY

- Pinhole camera model is the simplest mathematical model that can be applied to a lot of real photography devices.
- Homogeneous coordinates are a mathematical tool used in computer graphics, computer vision.
- Perspective projection and transformation necessary in computer vision.
- Calibration is crucial for tasks such as 3D reconstruction and object tracking. Camera calibration involves estimating the intrinsic and extrinsic parameters of a camera, which are necessary for accurate image analysis.
- Color spaces: RGB, HSV, YCbCr, L*a*b*.