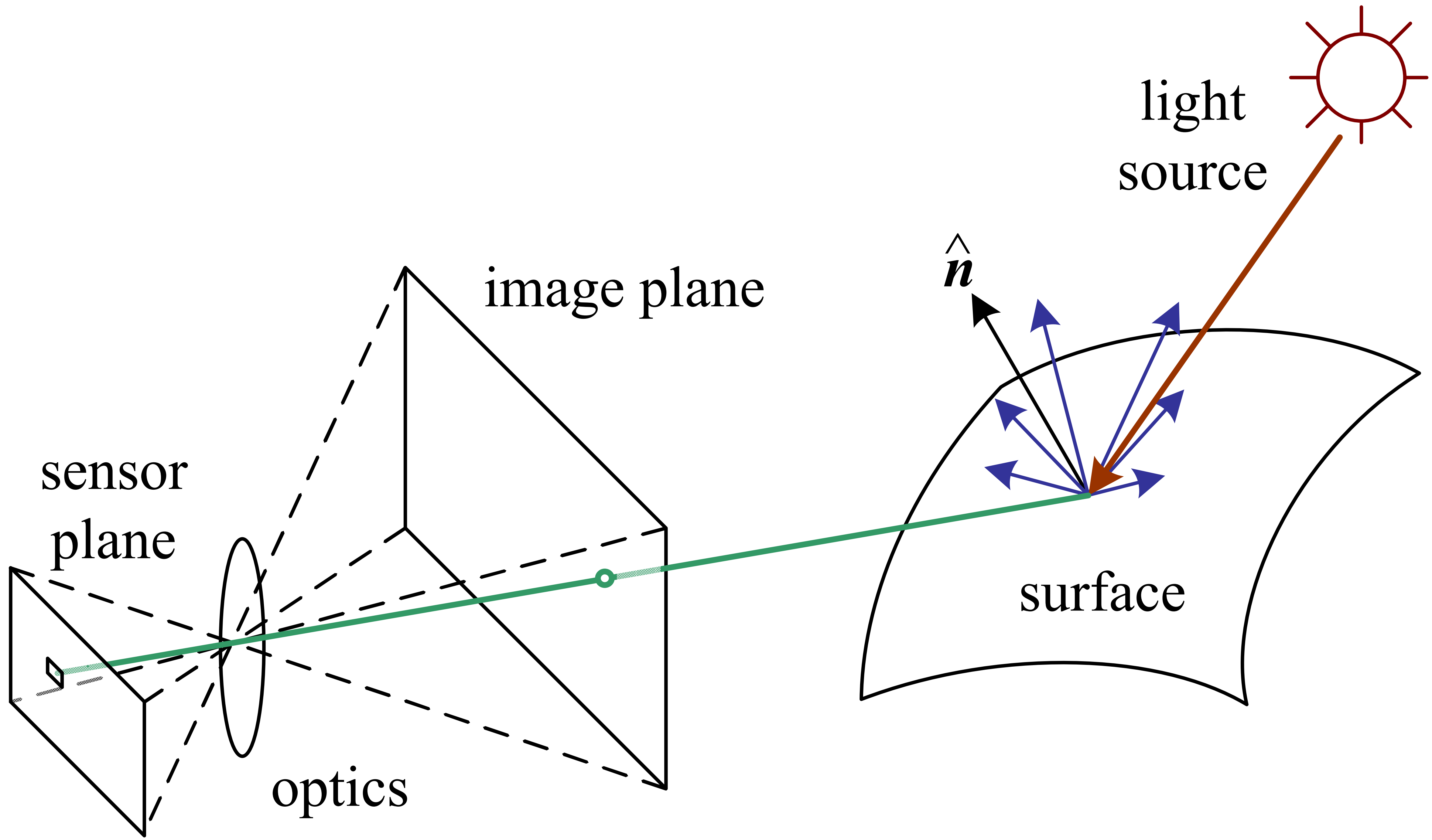
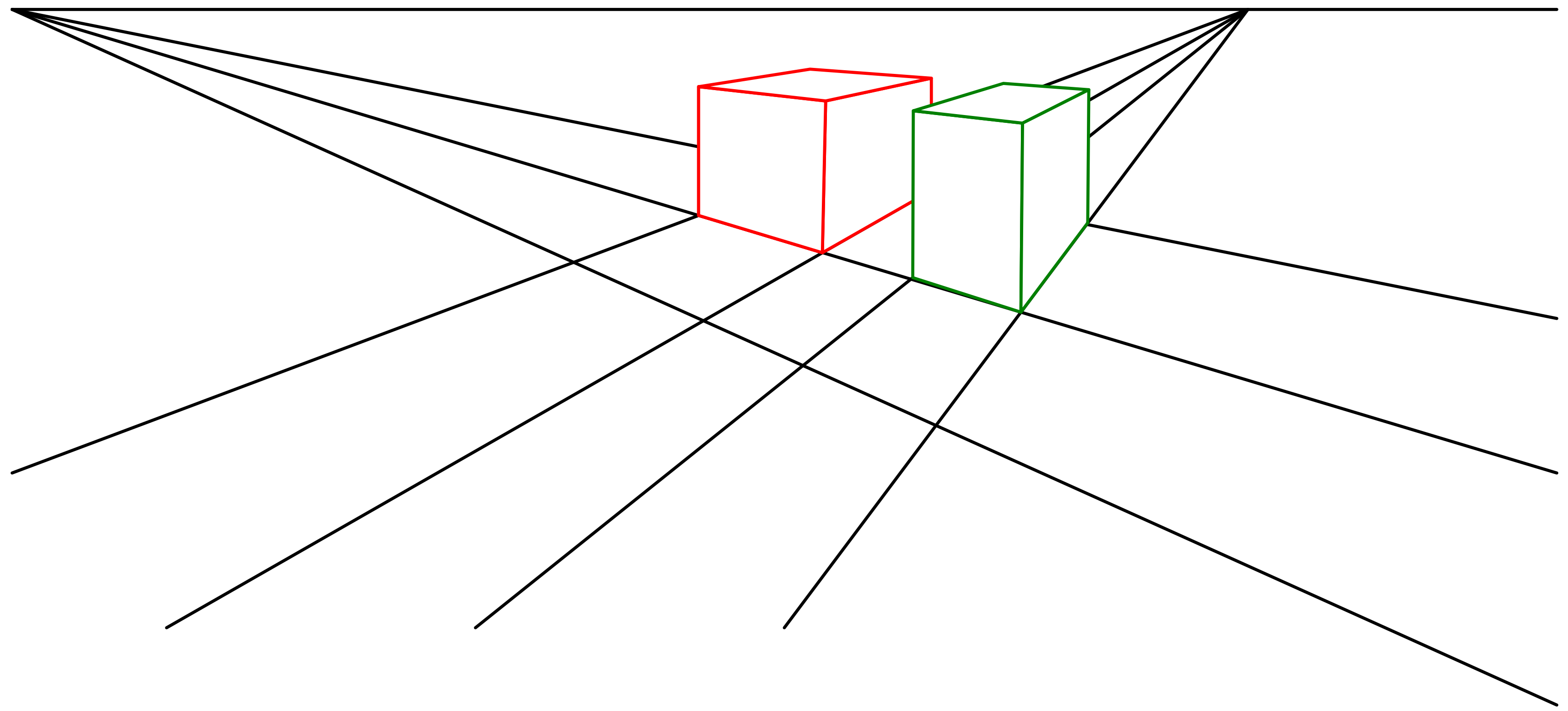
2. Image Formation



2.1 Geometric Primitives & Transformations



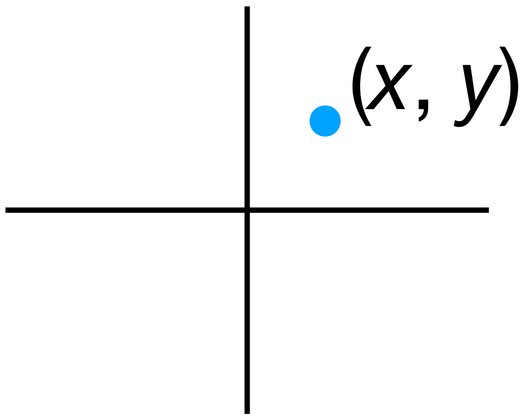
Outline

- Geometric primitives
- 2D transformations
- 3D transformations
- 3D rotations
- 3D to 2D projections
- Lens Distortions

Outline

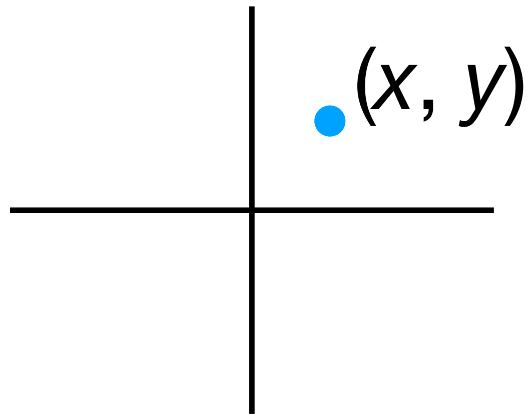
- **Geometric primitives**
- 2D transformations
- 3D transformations
- 3D rotations
- 3D to 2D projections
- Lens Distortions

2D Points



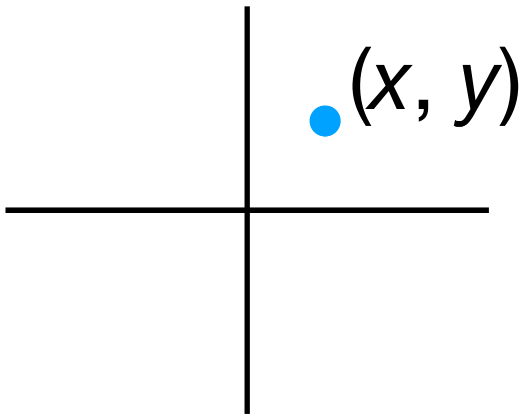
2D Points

2D point (e.g., a pixel coordinate in an image):



2D Points

2D point (e.g., a pixel coordinate in an image):

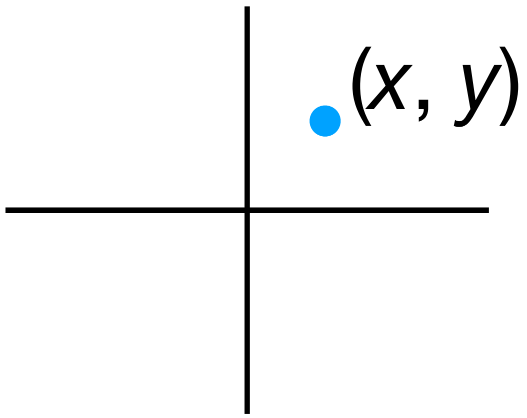


$$\boldsymbol{x} = (x, y) \in \mathcal{R}^2.$$

2D Points

2D point (e.g., a pixel coordinate in an image):

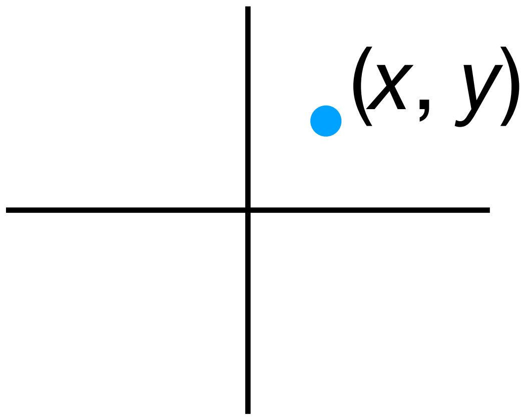
$\boldsymbol{x} = (x, y) \in \mathcal{R}^2$ or



2D Points

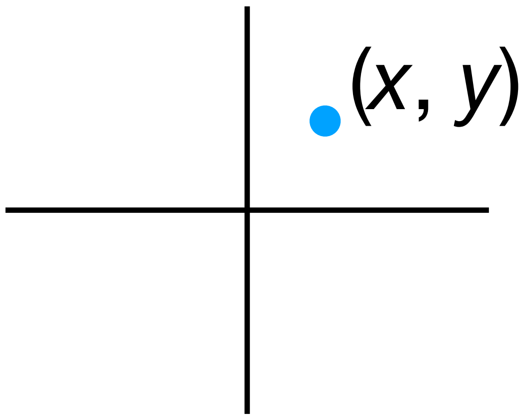
2D point (e.g., a pixel coordinate in an image):

$$\boldsymbol{x} = (x, y) \in \mathcal{R}^2 \qquad \text{or} \qquad \boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}$$



2D Points

2D point (e.g., a pixel coordinate in an image):



$$\boldsymbol{x} = (x, y) \in \mathcal{R}^2$$

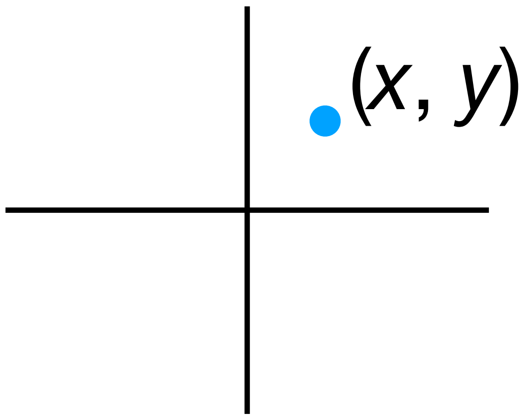
or

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}$$

In **homogeneous coordinates**:

2D Points

2D point (e.g., a pixel coordinate in an image):



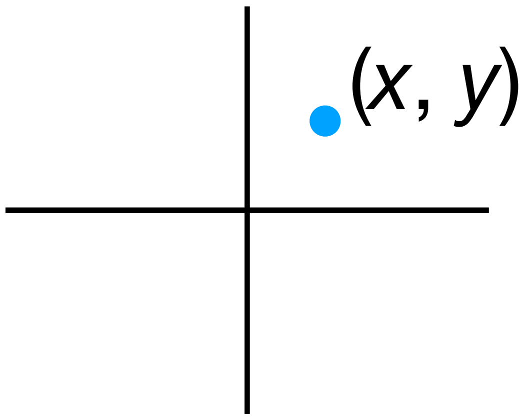
$$\boldsymbol{x} = (x, y) \in \mathcal{R}^2 \quad \text{or} \quad \boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}$$

In **homogeneous coordinates**:

$$\tilde{\boldsymbol{x}} = (\tilde{x}, \tilde{y}, \tilde{w}) \in \mathcal{P}^2$$

2D Points

2D point (e.g., a pixel coordinate in an image):



$$\boldsymbol{x} = (x, y) \in \mathcal{R}^2 \quad \text{or} \quad \boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}$$

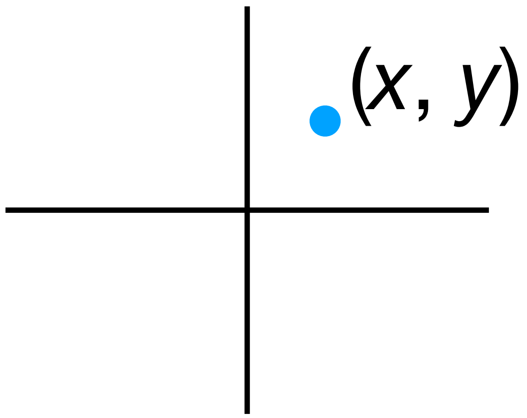
In **homogeneous coordinates**:

$$\tilde{\boldsymbol{x}} = (\tilde{x}, \tilde{y}, \tilde{w}) \in \mathcal{P}^2$$

$\mathcal{P}^2 = \mathcal{R}^3 - (0,0,0)$ is called the projective space.

2D Points

2D point (e.g., a pixel coordinate in an image):



$\boldsymbol{x} = (x, y) \in \mathcal{R}^2$ or $\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

In **homogeneous coordinates**:

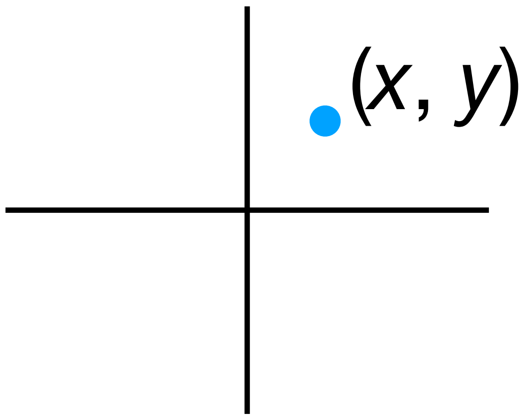
$\tilde{\boldsymbol{x}} = (\tilde{x}, \tilde{y}, \tilde{w}) \in \mathcal{P}^2$

$\mathcal{P}^2 = \mathcal{R}^3 - (0,0,0)$ is called the projective space.

Vectors that differ only by a scale considered equivalent:

2D Points

2D point (e.g., a pixel coordinate in an image):



$\boldsymbol{x} = (x, y) \in \mathcal{R}^2$ or $\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

In **homogeneous coordinates**:

$\tilde{\boldsymbol{x}} = (\tilde{x}, \tilde{y}, \tilde{w}) \in \mathcal{P}^2$

$\mathcal{P}^2 = \mathcal{R}^3 - (0,0,0)$ is called the projective space.

Vectors that differ only by a scale considered equivalent:

$s\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{x}} \;\forall s \in \mathcal{R}$

Augmented Vectors

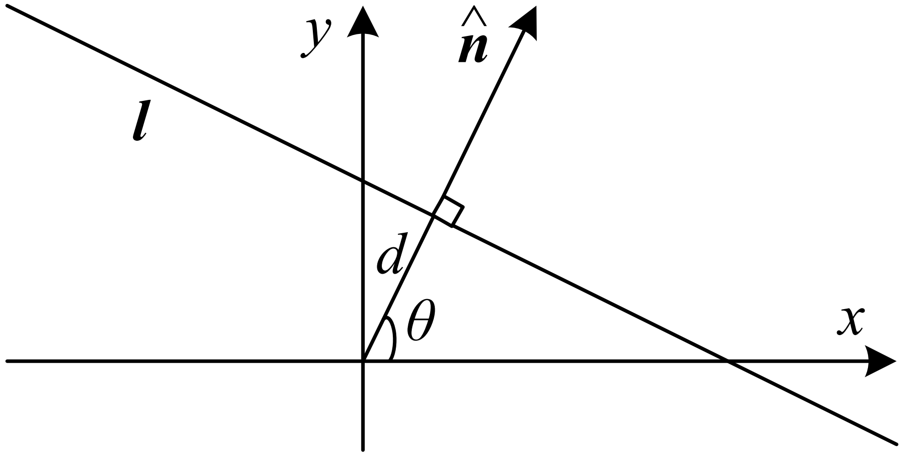
A homogenous vector can be converted back to an *inhomogeneous* vector by dividing by the last element:

$$\tilde{\boldsymbol{x}} = (\tilde{x}, \tilde{y}, \tilde{w}) = \tilde{w}(x, y, 1) = \tilde{w}\bar{\boldsymbol{x}}$$

Homogeneous vector

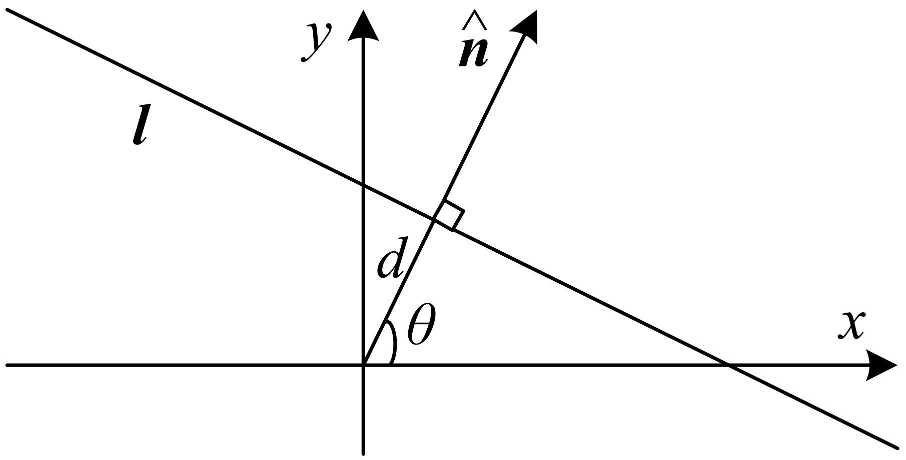
Augmented vector

2D Lines



2D Lines

2D lines can also be represented using homogeneous coordinates $\tilde{\boldsymbol{l}} = (a, b, c)$.

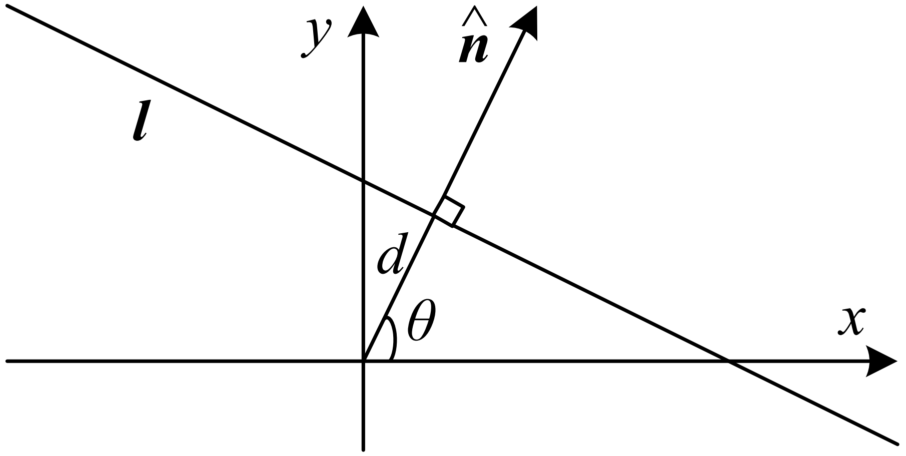


2D Lines

2D lines can also be represented using homogeneous coordinates $\tilde{\boldsymbol{l}} = (a, b, c)$.

The corresponding *line equation* is

$$\bar{\boldsymbol{x}} \cdot \tilde{\boldsymbol{l}} = ax + by + c = 0.$$



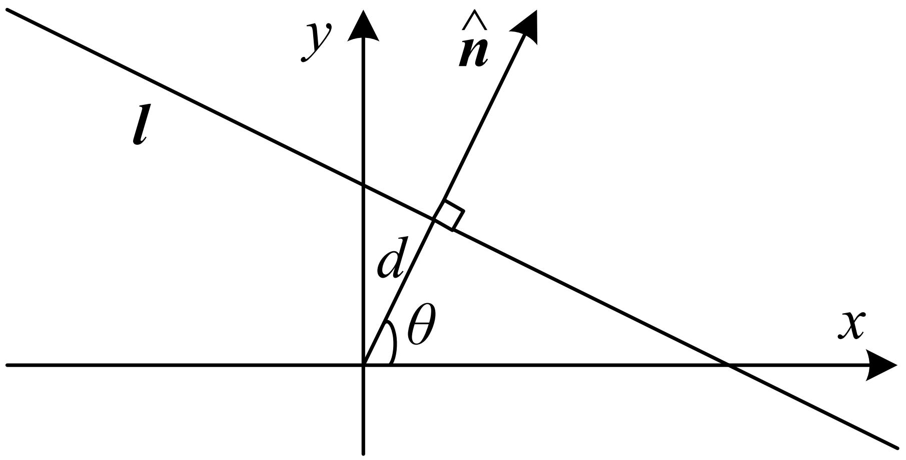
2D Lines

2D lines can also be represented using homogeneous coordinates $\tilde{\boldsymbol{l}} = (a, b, c)$.

The corresponding *line equation* is

$$\bar{\boldsymbol{x}} \cdot \tilde{\boldsymbol{l}} = ax + by + c = 0.$$

We can normalize the line equation vector so that $\boldsymbol{l} = (\hat{n}_x, \hat{n}_y, d) = (\hat{\boldsymbol{n}}, d)$ with $\|\hat{\boldsymbol{n}}\| = 1$.



2D Lines

2D lines can also be represented using homogeneous coordinates $\tilde{\boldsymbol{l}} = (a, b, c)$.

The corresponding *line equation* is

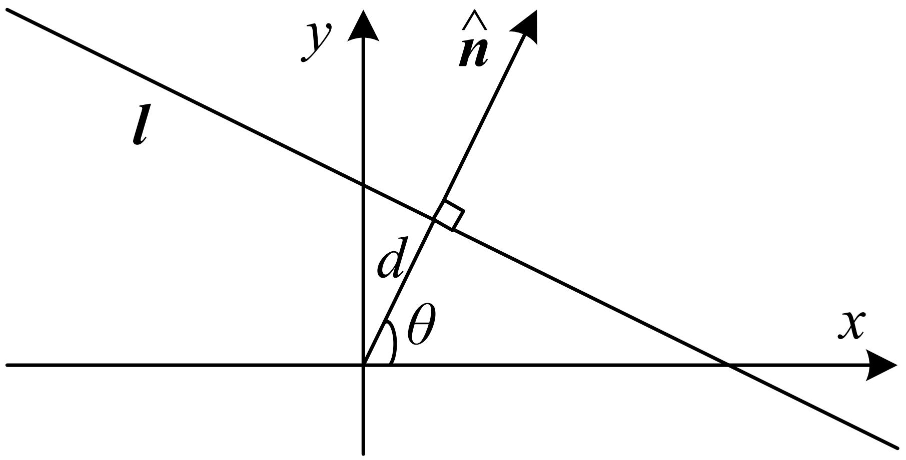
$$\bar{\boldsymbol{x}} \cdot \tilde{\boldsymbol{l}} = ax + by + c = 0.$$

We can normalize the line equation vector so that $\boldsymbol{l} = (\hat{n}_x, \hat{n}_y, d) = (\hat{\boldsymbol{n}}, d)$ with $\|\hat{\boldsymbol{n}}\| = 1$.

Then:

$\hat{\boldsymbol{n}}$ is the unit normal perpendicular to the line

$|d|$ is the distance of the line from the origin



2D Lines

2D lines can also be represented using homogeneous coordinates $\tilde{\boldsymbol{l}} = (a, b, c)$.

The corresponding *line equation* is

$$\bar{\boldsymbol{x}} \cdot \tilde{\boldsymbol{l}} = ax + by + c = 0.$$

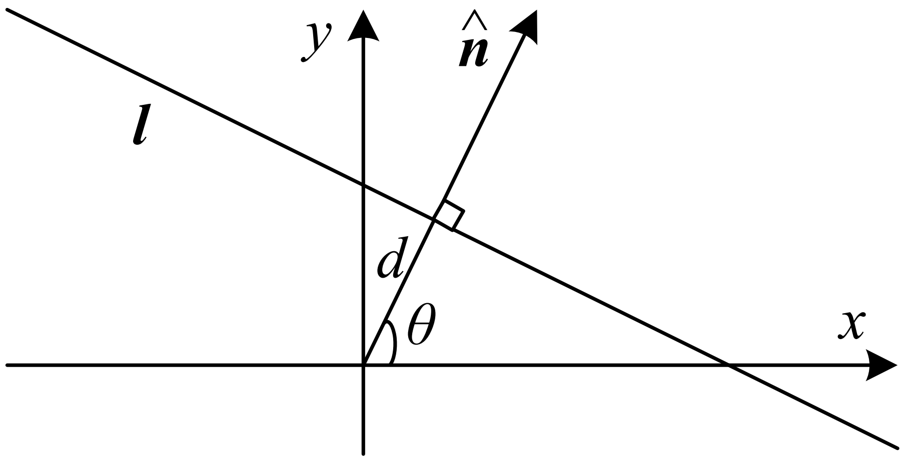
We can normalize the line equation vector so that $\boldsymbol{l} = (\hat{n}_x, \hat{n}_y, d) = (\hat{\boldsymbol{n}}, d)$ with $\|\hat{\boldsymbol{n}}\| = 1$.

Then:

$\hat{\boldsymbol{n}}$ is the unit normal perpendicular to the line

$|d|$ is the distance of the line from the origin

$$\hat{\boldsymbol{n}} = (\hat{n}_x, \hat{n}_y) = (\cos\theta, \sin\theta)$$



Intersections of 2D Lines

Intersections of 2D Lines

When using homogeneous coordinates, we can compute the intersection of two lines as

Intersections of 2D Lines

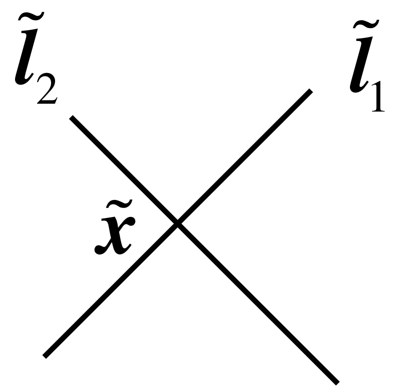
When using homogeneous coordinates, we can compute the intersection of two lines as

$$\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{l}}_1 \times \tilde{\boldsymbol{l}}_2$$

Intersections of 2D Lines

When using homogeneous coordinates, we can compute the intersection of two lines as

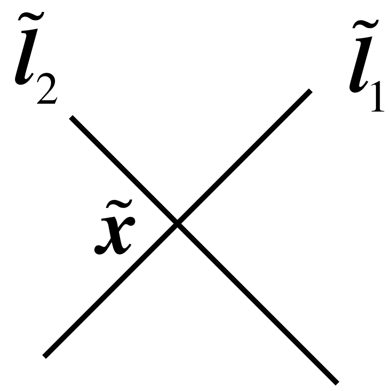
$$\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{l}}_1 \times \tilde{\boldsymbol{l}}_2$$



Intersections of 2D Lines

When using homogeneous coordinates, we can compute the intersection of two lines as

$$\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{l}}_1 \times \tilde{\boldsymbol{l}}_2$$

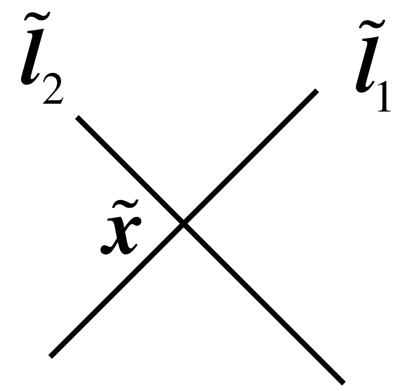


Similarly, the line joining two points can be written as

Intersections of 2D Lines

When using homogeneous coordinates, we can compute the intersection of two lines as

$$\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{l}}_1 \times \tilde{\boldsymbol{l}}_2$$



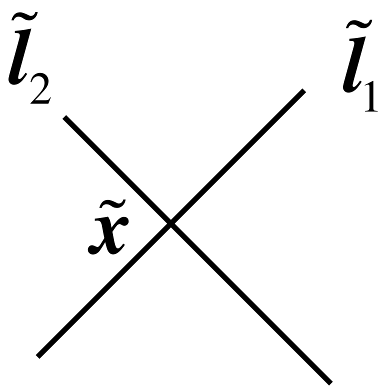
Similarly, the line joining two points can be written as

$$\tilde{\boldsymbol{l}} = \tilde{\boldsymbol{x}}_1 \times \tilde{\boldsymbol{x}}_2$$

Intersections of 2D Lines

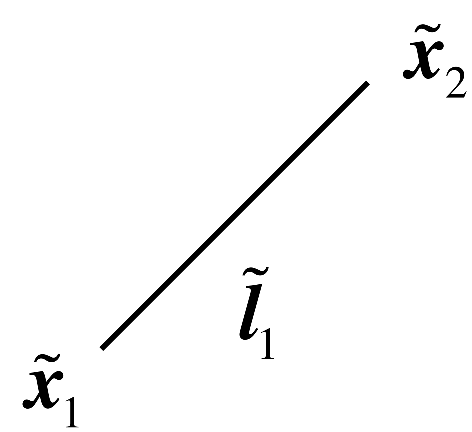
When using homogeneous coordinates, we can compute the intersection of two lines as

$$\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{l}}_1 \times \tilde{\boldsymbol{l}}_2$$



Similarly, the line joining two points can be written as

$$\tilde{\boldsymbol{l}} = \tilde{\boldsymbol{x}}_1 \times \tilde{\boldsymbol{x}}_2$$



3D Points

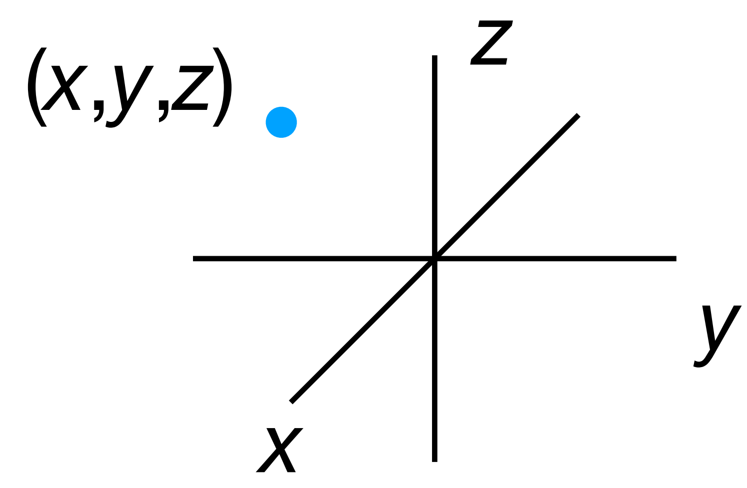
Straightforward extension from 2D:

Inhomogeneous: $\boldsymbol{x} = (x, y, z) \in \mathcal{R}^3$

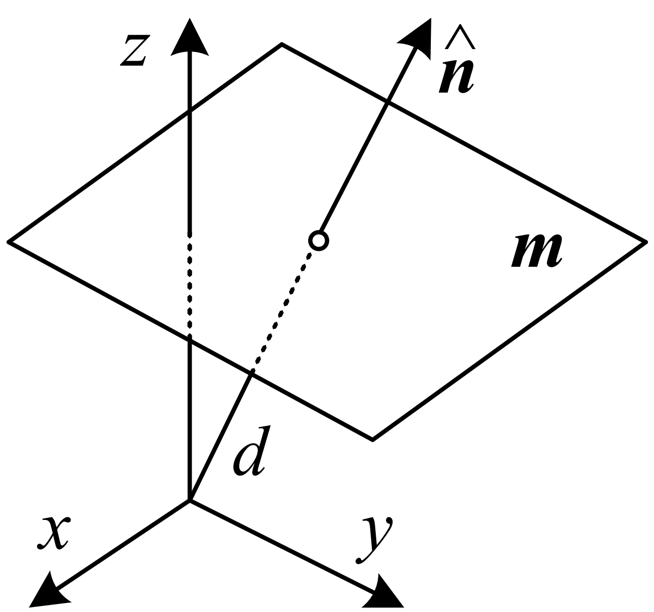
Homogeneous: $\tilde{\boldsymbol{x}} = (\tilde{x}, \tilde{y}, \tilde{z}, \tilde{w}) \in \mathcal{P}^3$

Augmented: $\bar{\boldsymbol{x}} = (x, y, z, 1)$

$$\tilde{\boldsymbol{x}} = \tilde{w}\bar{\boldsymbol{x}}$$

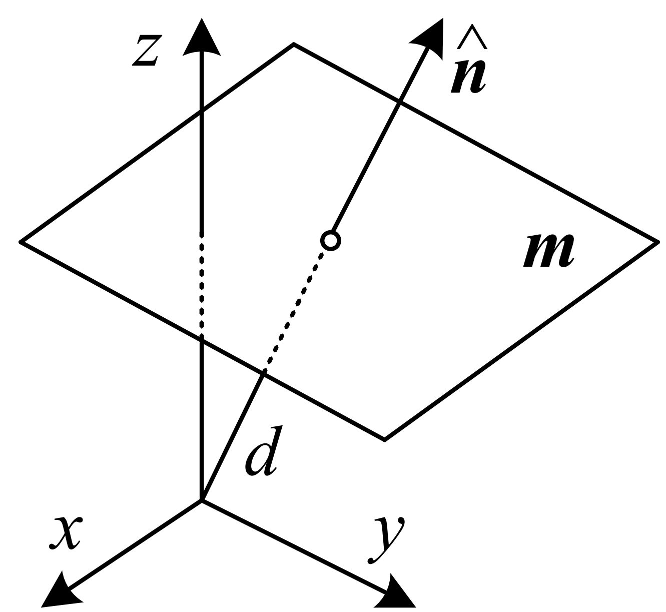


3D Planes



3D Planes

3D planes can also be represented as homogeneous coordinates $\tilde{\boldsymbol{m}} = (a, b, c, d)$

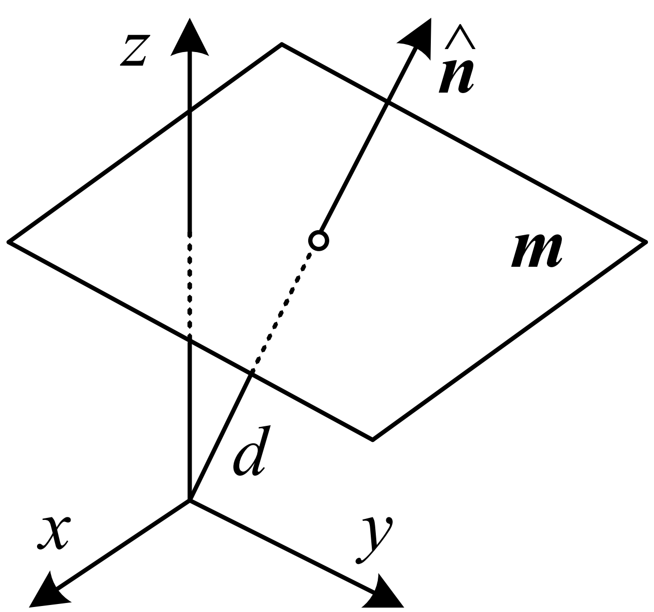


3D Planes

3D planes can also be represented as homogeneous coordinates $\tilde{\boldsymbol{m}} = (a, b, c, d)$

with a corresponding plane equation

$$\bar{\boldsymbol{x}} \cdot \tilde{\boldsymbol{m}} = ax + by + cz + d = 0$$



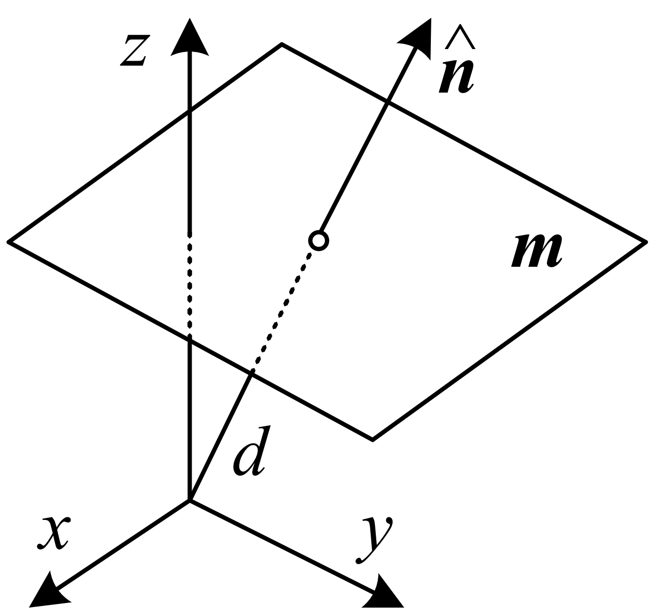
3D Planes

3D planes can also be represented as homogeneous coordinates $\tilde{\boldsymbol{m}} = (a, b, c, d)$

with a corresponding plane equation

$$\bar{\boldsymbol{x}} \cdot \tilde{\boldsymbol{m}} = ax + by + cz + d = 0$$

We can also normalize the plane equation as $\boldsymbol{m} = (\hat{n}_x, \hat{n}_y, \hat{n}_z, d) = (\hat{\boldsymbol{n}}, d)$ with $\|\hat{\boldsymbol{n}}\| = 1$.



3D Planes

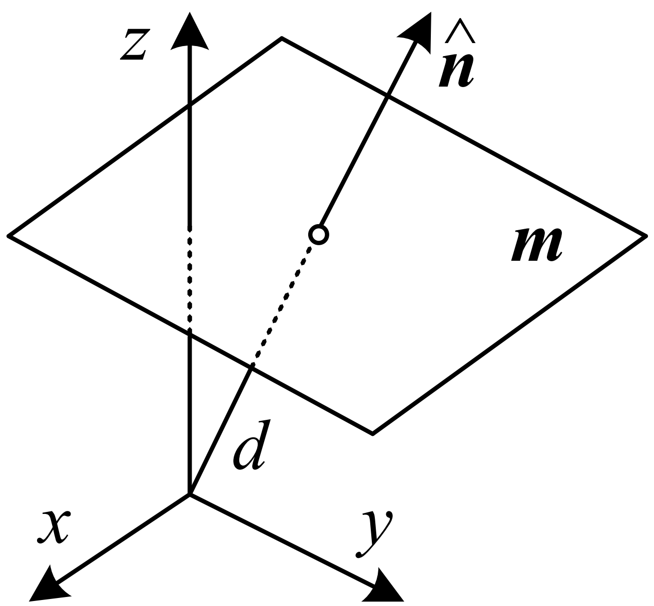
3D planes can also be represented as homogeneous coordinates $\tilde{\boldsymbol{m}} = (a, b, c, d)$

with a corresponding plane equation

$$\bar{\boldsymbol{x}} \cdot \tilde{\boldsymbol{m}} = ax + by + cz + d = 0$$

We can also normalize the plane equation as $\boldsymbol{m} = (\hat{n}_x, \hat{n}_y, \hat{n}_z, d) = (\hat{\boldsymbol{n}}, d)$ with $\|\hat{\boldsymbol{n}}\| = 1$.

$\hat{\boldsymbol{n}}$ is the *normal vector* perpendicular to the plane



3D Planes

3D planes can also be represented as homogeneous coordinates $\tilde{\boldsymbol{m}} = (a, b, c, d)$

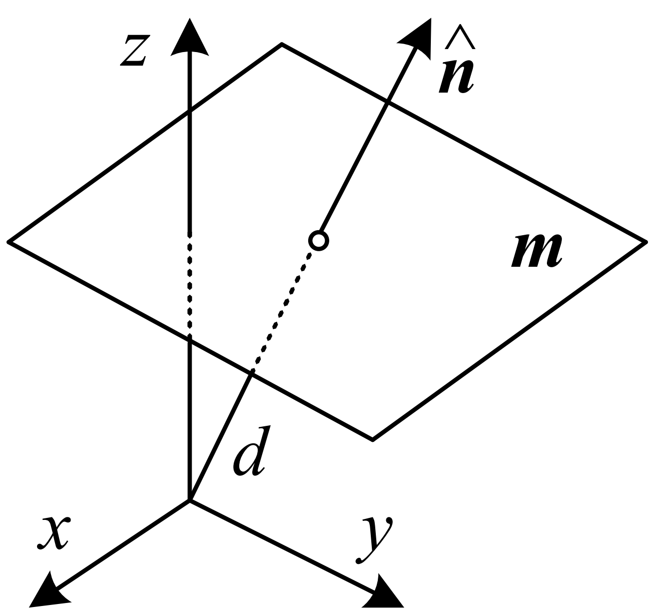
with a corresponding plane equation

$$\bar{\boldsymbol{x}} \cdot \tilde{\boldsymbol{m}} = ax + by + cz + d = 0$$

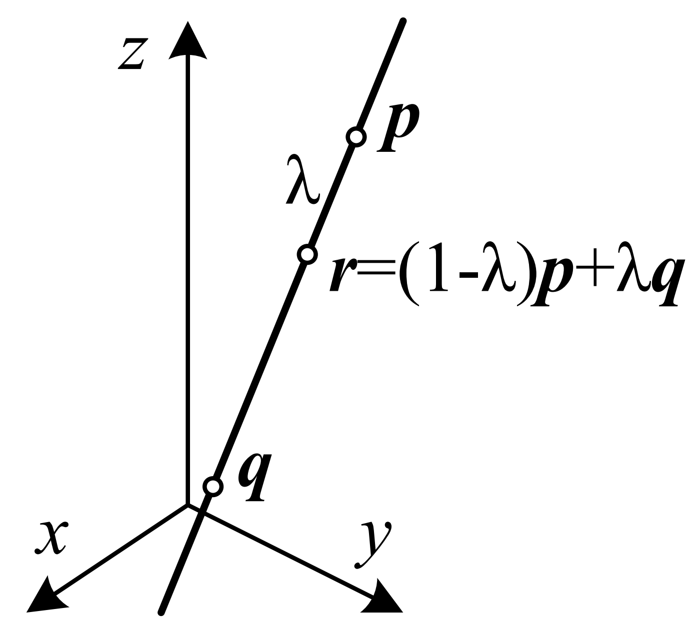
We can also normalize the plane equation as $\boldsymbol{m} = (\hat{n}_x, \hat{n}_y, \hat{n}_z, d) = (\hat{\boldsymbol{n}}, d)$ with $\|\hat{\boldsymbol{n}}\| = 1$.

$\hat{\boldsymbol{n}}$ is the *normal vector* perpendicular to the plane

$|d|$ is the distance of the plane from the origin

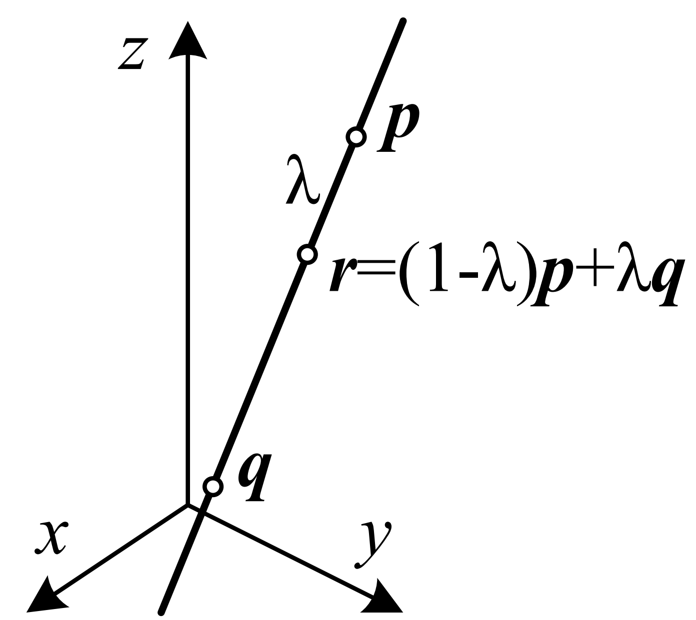


3D Lines



3D Lines

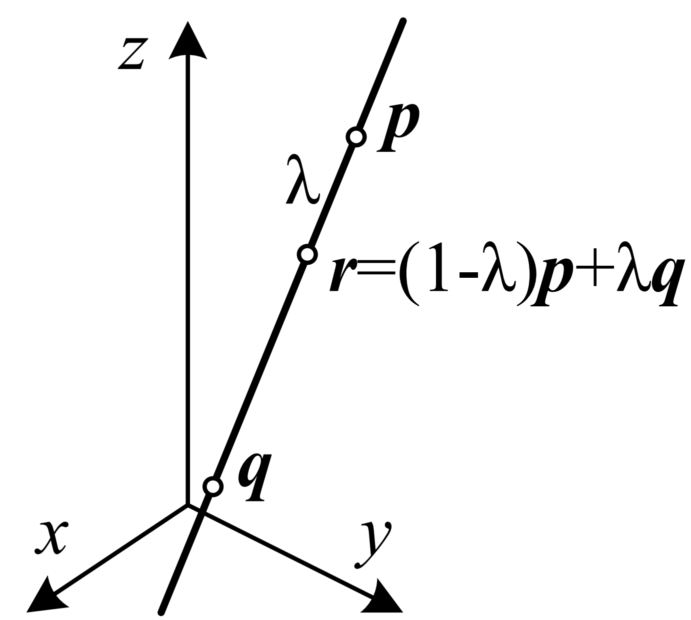
A 3D line can be represented using two points $\boldsymbol{p}$ and $\boldsymbol{q}$ that lie on the line.



3D Lines

A 3D line can be represented using two points $\boldsymbol{p}$ and $\boldsymbol{q}$ that lie on the line.

Any point $\boldsymbol{r}$ that also lies on the line can then be represented as

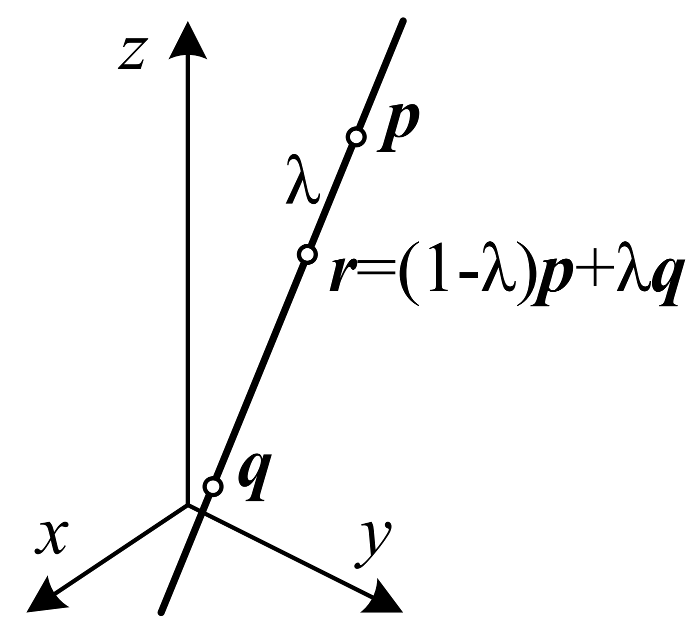


3D Lines

A 3D line can be represented using two points $\boldsymbol{p}$ and $\boldsymbol{q}$ that lie on the line.

Any point $\boldsymbol{r}$ that also lies on the line can then be represented as

$$\boldsymbol{r} = (1-\lambda)\,\boldsymbol{p} + \lambda \boldsymbol{q}$$



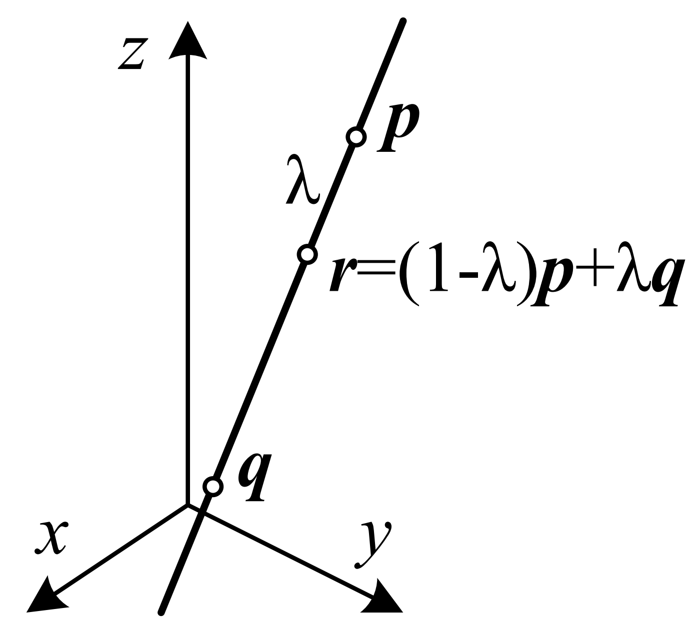
3D Lines

A 3D line can be represented using two points $\boldsymbol{p}$ and $\boldsymbol{q}$ that lie on the line.

Any point $\boldsymbol{r}$ that also lies on the line can then be represented as

$$\boldsymbol{r} = (1-\lambda)\,\boldsymbol{p} + \lambda\boldsymbol{q}$$

If we restrict $0 \leq \lambda \leq 1$, we get the *line segment* joining $\boldsymbol{p}$ and $\boldsymbol{q}$.



3D Lines

A 3D line can be represented using two points $\boldsymbol{p}$ and $\boldsymbol{q}$ that lie on the line.

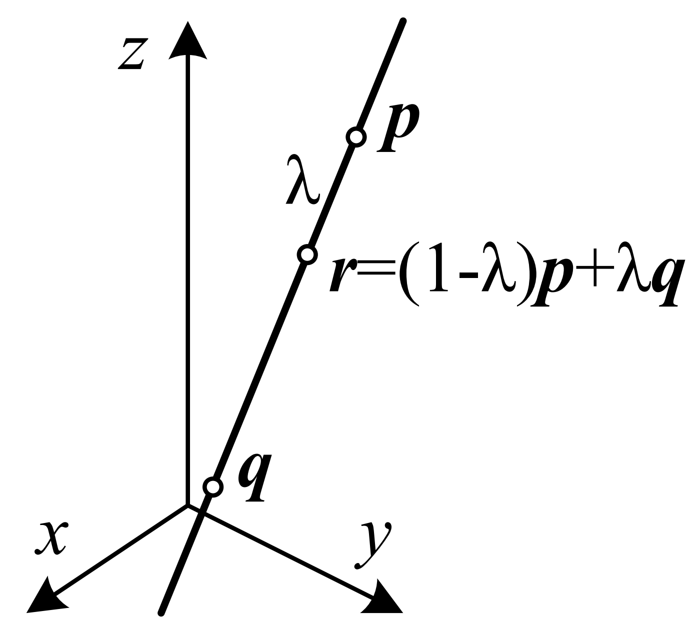
Any point $\boldsymbol{r}$ that also lies on the line can then be represented as

$$\boldsymbol{r} = (1-\lambda)\,\boldsymbol{p} + \lambda \boldsymbol{q}$$

If we restrict $0 \leq \lambda \leq 1$, we get the *line segment* joining $\boldsymbol{p}$ and $\boldsymbol{q}$.

If we use homogeneous coordinates, we can write the line as

$$\tilde{\boldsymbol{r}} = \mu \tilde{\boldsymbol{p}} + \lambda \tilde{\boldsymbol{q}}.$$



3D Lines

A 3D line can be represented using two points $\boldsymbol{p}$ and $\boldsymbol{q}$ that lie on the line.

Any point $\boldsymbol{r}$ that also lies on the line can then be represented as

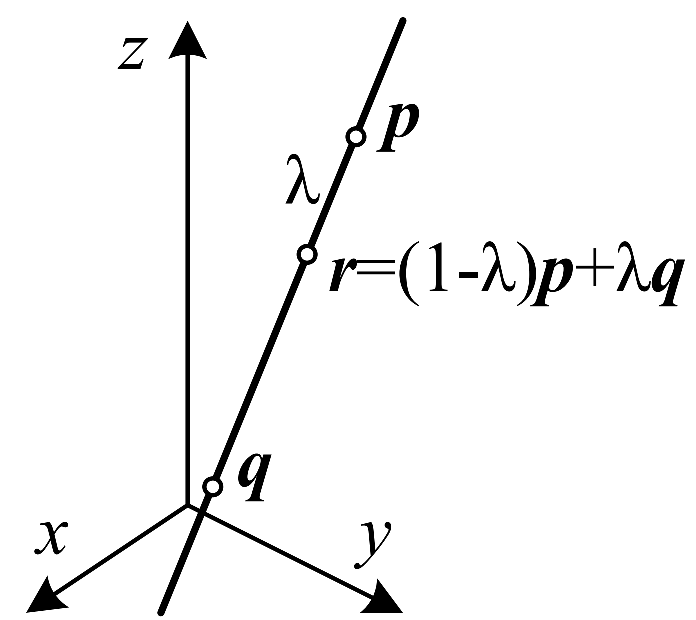
$$\boldsymbol{r} = (1-\lambda)\,\boldsymbol{p} + \lambda \boldsymbol{q}$$

If we restrict $0 \leq \lambda \leq 1$, we get the *line segment* joining $\boldsymbol{p}$ and $\boldsymbol{q}$.

If we use homogeneous coordinates, we can write the line as

$$\tilde{\boldsymbol{r}} = \mu \tilde{\boldsymbol{p}} + \lambda \tilde{\boldsymbol{q}}.$$

A special case of this is when the second point is at infinity, i.e., $\tilde{\boldsymbol{q}} = (\hat{d}_x, \hat{d}_y, \hat{d}_z, 0) = (\hat{\boldsymbol{d}}, 0)$.



3D Lines

A 3D line can be represented using two points $\boldsymbol{p}$ and $\boldsymbol{q}$ that lie on the line.

Any point $\boldsymbol{r}$ that also lies on the line can then be represented as

$$\boldsymbol{r} = (1-\lambda)\boldsymbol{p} + \lambda\boldsymbol{q}$$

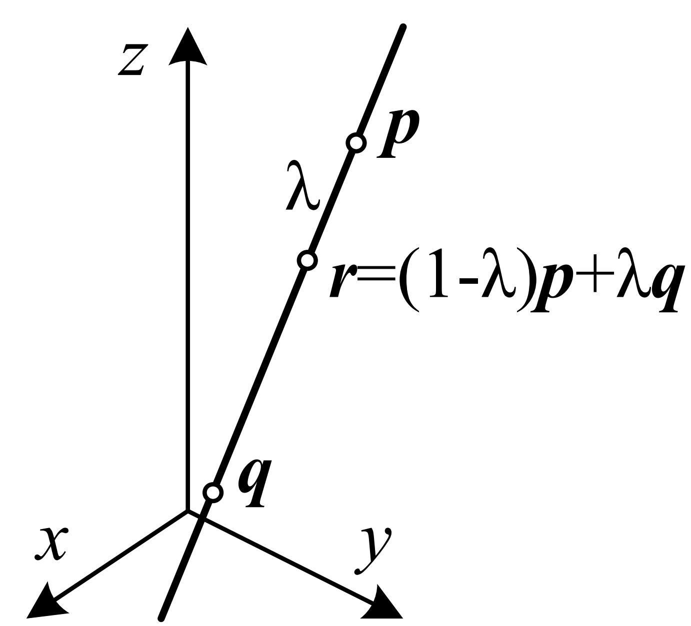
If we restrict $0 \leq \lambda \leq 1$, we get the *line segment* joining $\boldsymbol{p}$ and $\boldsymbol{q}$.

If we use homogeneous coordinates, we can write the line as

$$\tilde{\boldsymbol{r}} = \mu\tilde{\boldsymbol{p}} + \lambda\tilde{\boldsymbol{q}}.$$

A special case of this is when the second point is at infinity, i.e., $\tilde{\boldsymbol{q}} = (\hat{d}_x, \hat{d}_y, \hat{d}_z, 0) = (\hat{\boldsymbol{d}}, 0)$.

where $\hat{\boldsymbol{d}}$ is the direction of the line.



3D Lines

A 3D line can be represented using two points $\boldsymbol{p}$ and $\boldsymbol{q}$ that lie on the line.

Any point $\boldsymbol{r}$ that also lies on the line can then be represented as

$$\boldsymbol{r} = (1-\lambda)\,\boldsymbol{p} + \lambda\boldsymbol{q}$$

If we restrict $0 \leq \lambda \leq 1$, we get the *line segment* joining $\boldsymbol{p}$ and $\boldsymbol{q}$.

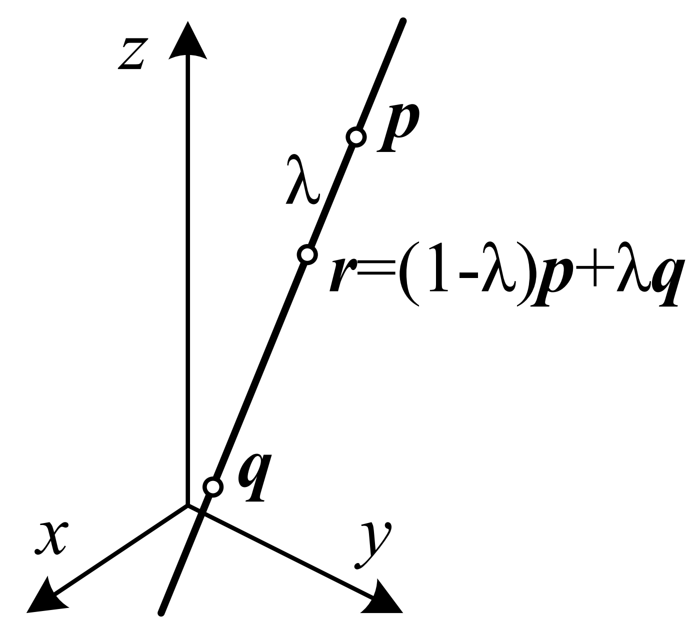
If we use homogeneous coordinates, we can write the line as

$$\tilde{\boldsymbol{r}} = \mu\tilde{\boldsymbol{p}} + \lambda\tilde{\boldsymbol{q}}.$$

A special case of this is when the second point is at infinity, i.e., $\tilde{\boldsymbol{q}} = (\hat{d}_x, \hat{d}_y, \hat{d}_z, 0) = (\hat{\boldsymbol{d}}, 0)$.

where $\hat{\boldsymbol{d}}$ is the direction of the line.

Then:



3D Lines

A 3D line can be represented using two points $\boldsymbol{p}$ and $\boldsymbol{q}$ that lie on the line.

Any point $\boldsymbol{r}$ that also lies on the line can then be represented as

$$\boldsymbol{r} = (1-\lambda)\boldsymbol{p} + \lambda\boldsymbol{q}$$

If we restrict $0 \leq \lambda \leq 1$, we get the *line segment* joining $\boldsymbol{p}$ and $\boldsymbol{q}$.

If we use homogeneous coordinates, we can write the line as

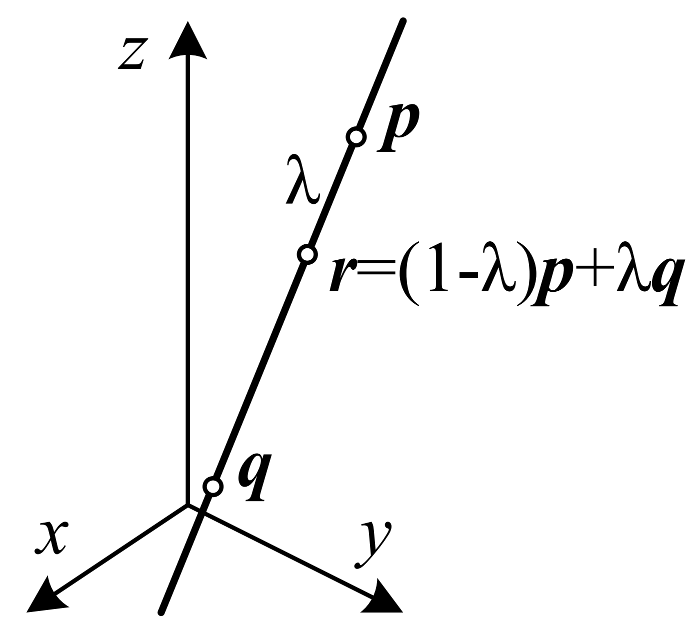
$$\tilde{\boldsymbol{r}} = \mu\tilde{\boldsymbol{p}} + \lambda\tilde{\boldsymbol{q}}.$$

A special case of this is when the second point is at infinity, i.e., $\tilde{\boldsymbol{q}} = (\hat{d}_x, \hat{d}_y, \hat{d}_z, 0) = (\hat{\boldsymbol{d}}, 0)$.

where $\hat{\boldsymbol{d}}$ is the direction of the line.

Then:

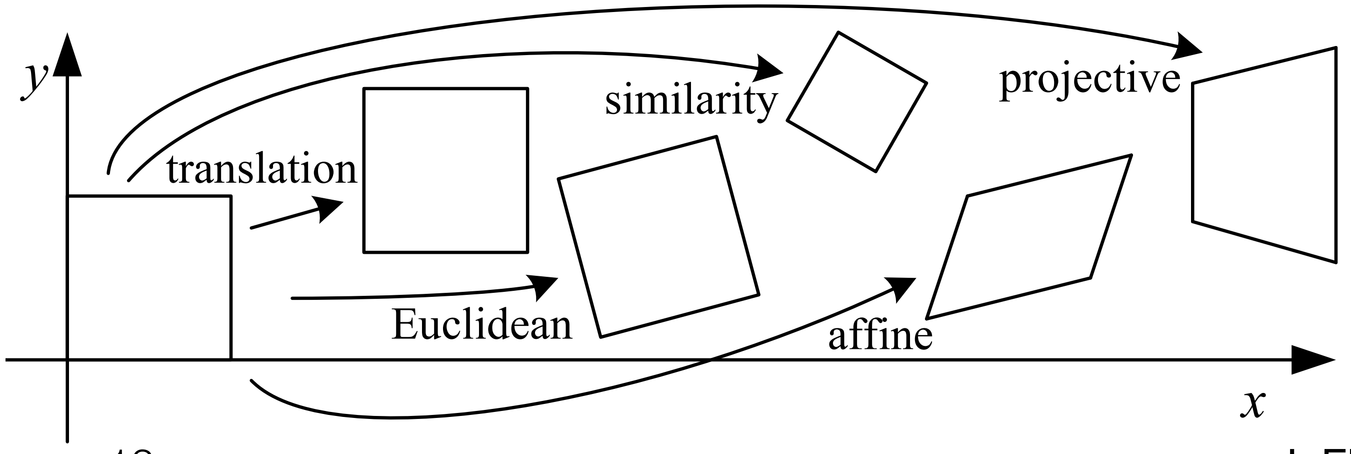
$$\boldsymbol{r} = \boldsymbol{p} + \lambda\hat{\boldsymbol{d}}$$



Outline

- ❖ Geometric primitives
- ❖ **2D transformations**
- ❖ 3D transformations
- ❖ 3D rotations
- ❖ 3D to 2D projections
- ❖ Lens Distortions

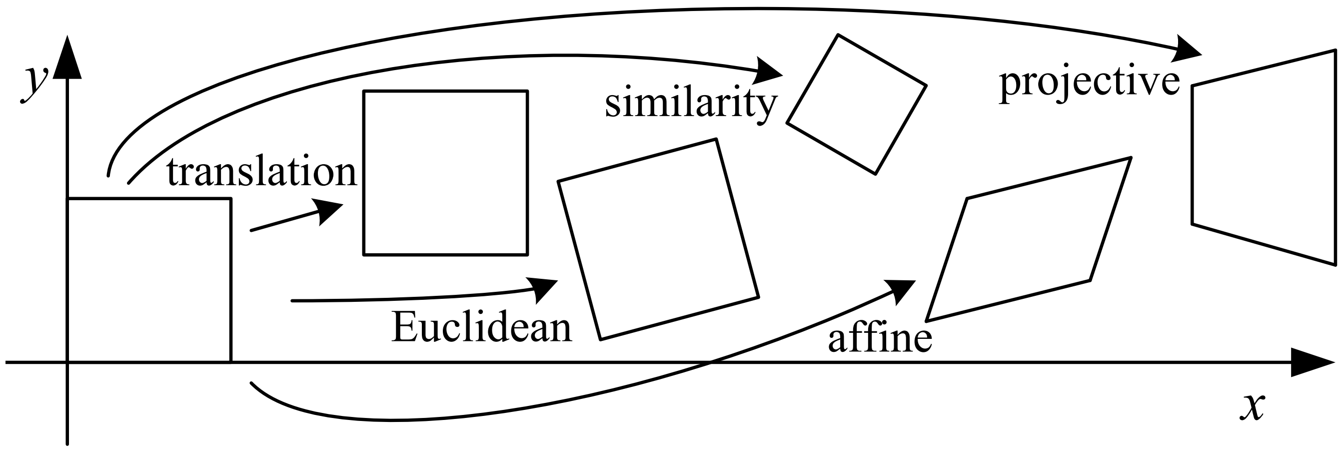
2D Translation



2D Translation

2D translations can be written as $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{t}$ or

$$\boldsymbol{x}' = \left[\begin{array}{cc} \boldsymbol{I} & \boldsymbol{t} \end{array} \right] \bar{\boldsymbol{x}}$$

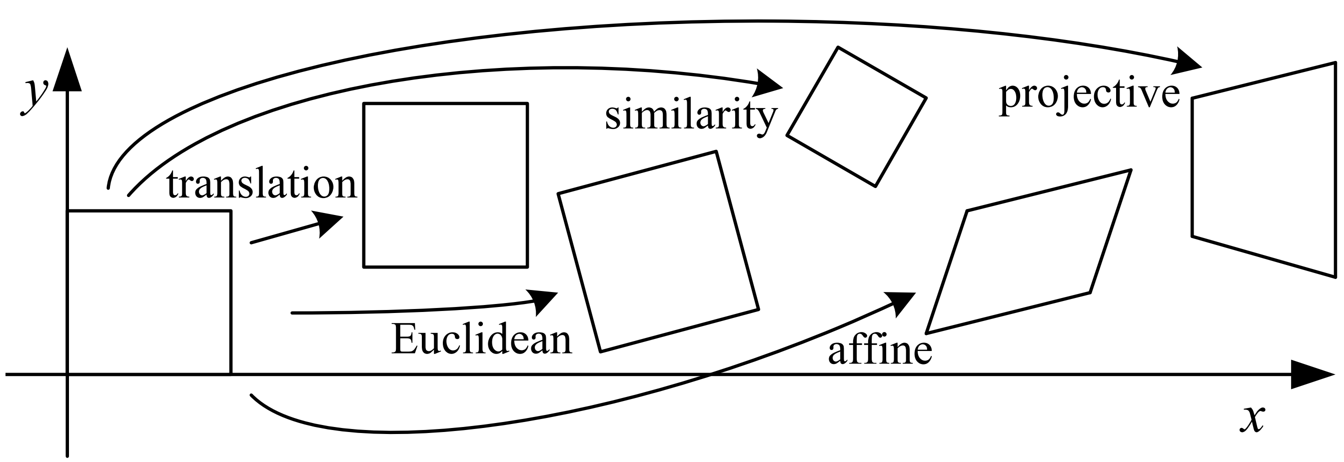


2D Translation

2D translations can be written as $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{t}$ or

$$\boldsymbol{x}' = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{t} \end{bmatrix} \bar{\boldsymbol{x}}$$

where $\boldsymbol{I}$ is the (2×2) identity matrix



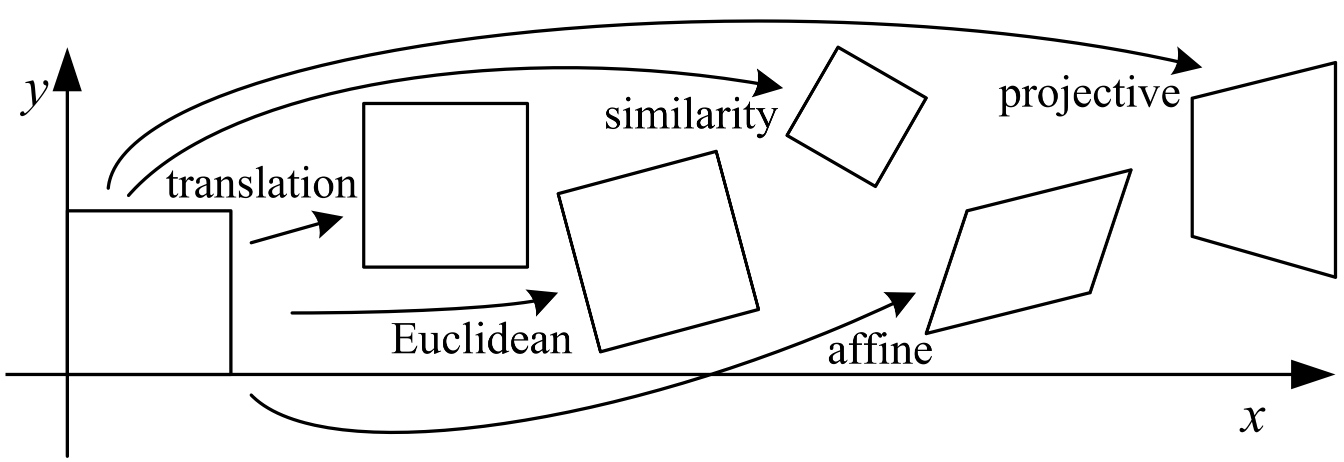
2D Translation

2D translations can be written as $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{t}$ or

$$\boldsymbol{x}' = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{t} \end{bmatrix} \bar{\boldsymbol{x}}$$

where $\boldsymbol{I}$ is the (2×2) identity matrix

or



2D Translation

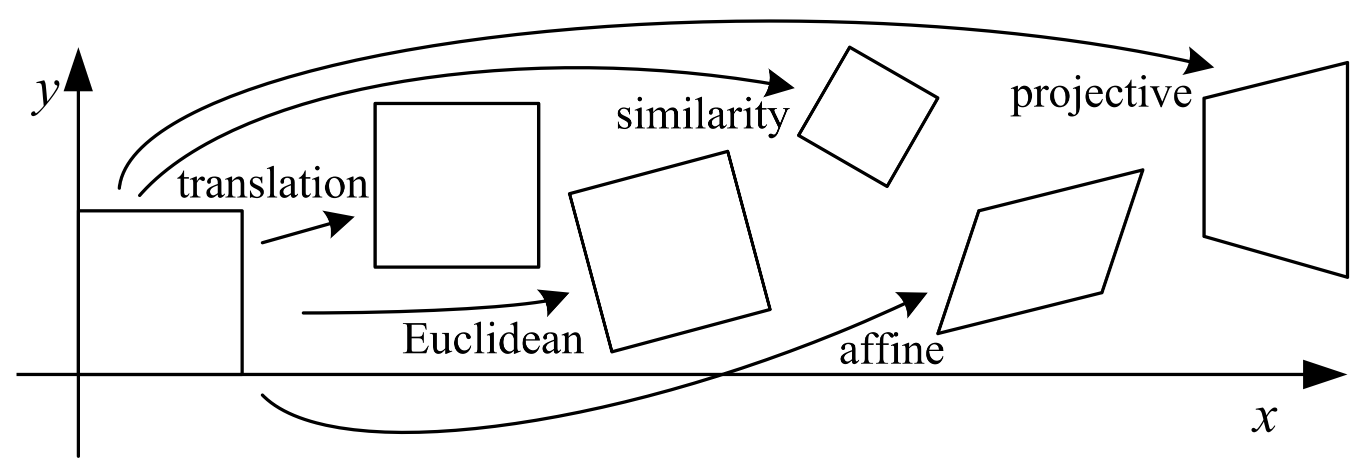
2D translations can be written as $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{t}$ or

$$\boldsymbol{x}' = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{t} \end{bmatrix} \bar{\boldsymbol{x}}$$

where $\boldsymbol{I}$ is the (2×2) identity matrix

or

$$\bar{\boldsymbol{x}}' = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \bar{\boldsymbol{x}}$$



2D Translation

2D translations can be written as $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{t}$ or

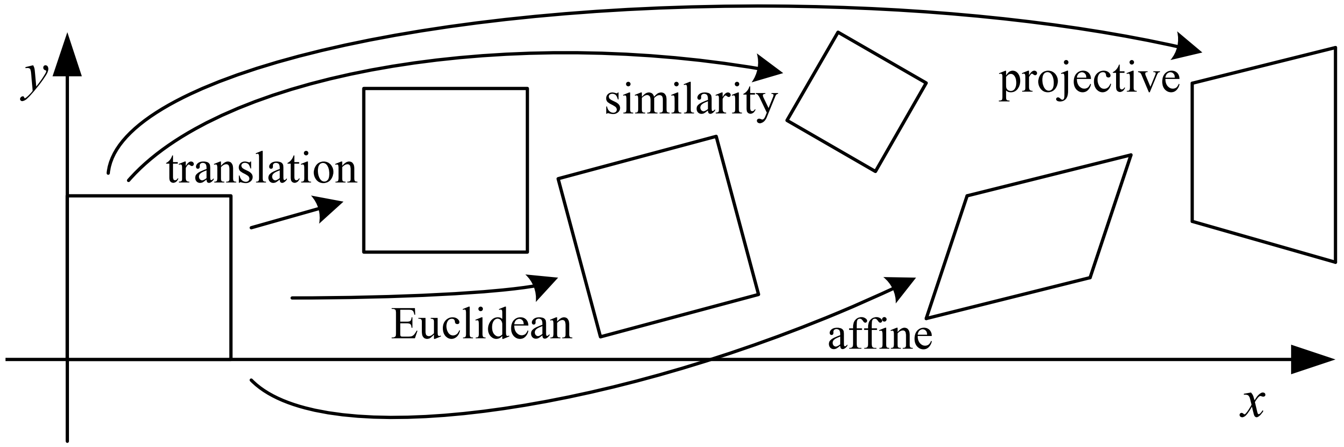
$$\boldsymbol{x}' = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{t} \end{bmatrix} \bar{\boldsymbol{x}}$$

where $\boldsymbol{I}$ is the (2×2) identity matrix

or

$$\bar{\boldsymbol{x}}' = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \bar{\boldsymbol{x}}$$

Note: Whenever an augmented vector appears on both sides, it can be replaced by a full homogenous vector.



Euclidean Transformation (2D Rotation +

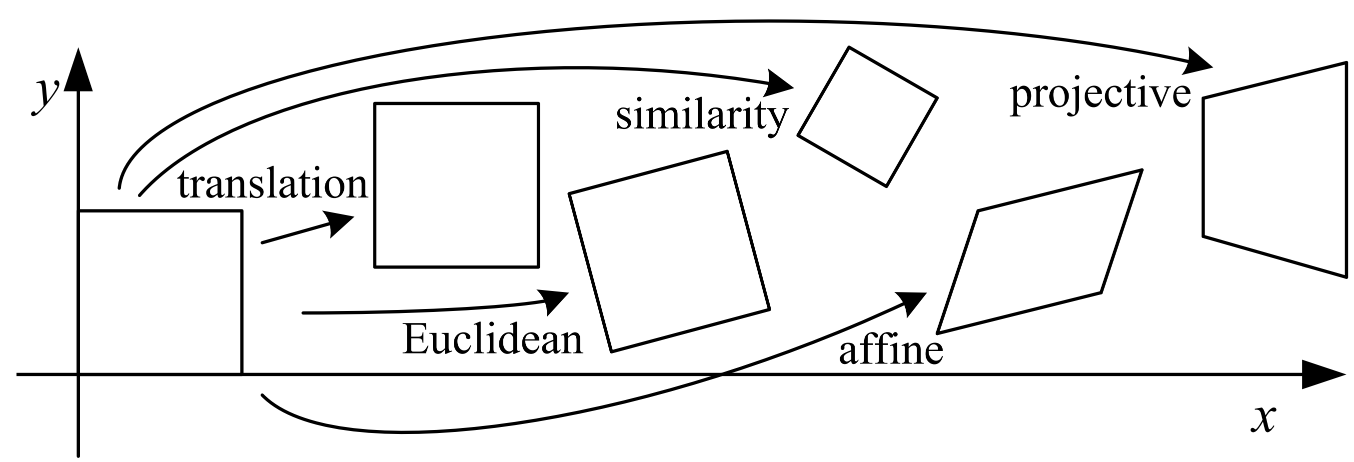
$$\boldsymbol{x}' = \boldsymbol{R}\boldsymbol{x} + \boldsymbol{t} \text{ or}$$

$$\boldsymbol{x}' = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \end{bmatrix} \bar{\boldsymbol{x}}$$

where

$$\boldsymbol{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

is an orthonormal rotation matrix with $\boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{I}$ and $|\boldsymbol{R}| = 1$.



Euclidean Transformation (2D Rotation +

$\boldsymbol{x}' = \boldsymbol{R}\boldsymbol{x} + \boldsymbol{t}$ or

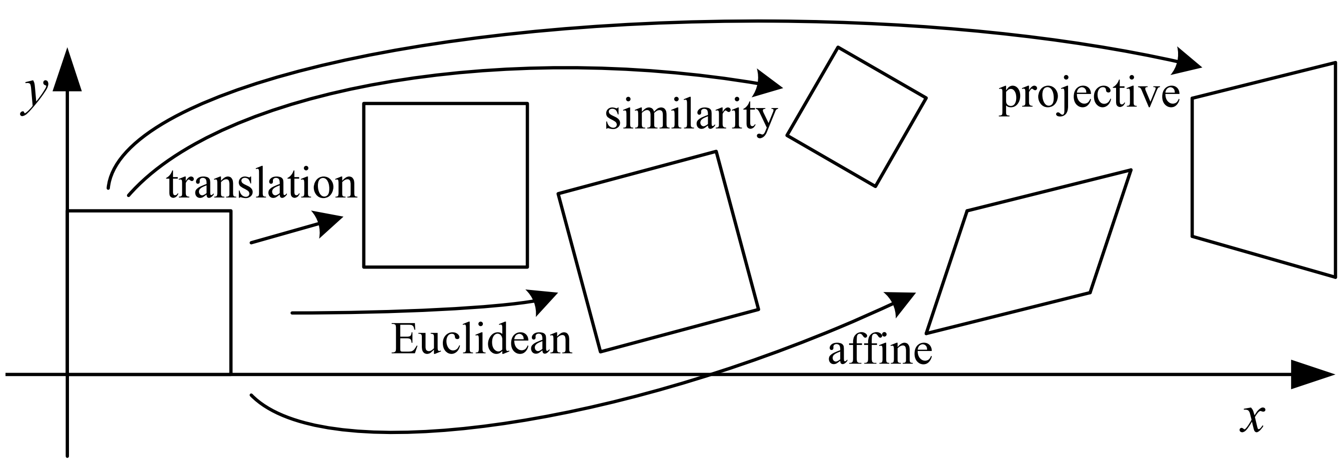
$$\boldsymbol{x}' = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \end{bmatrix} \bar{\boldsymbol{x}}$$

where

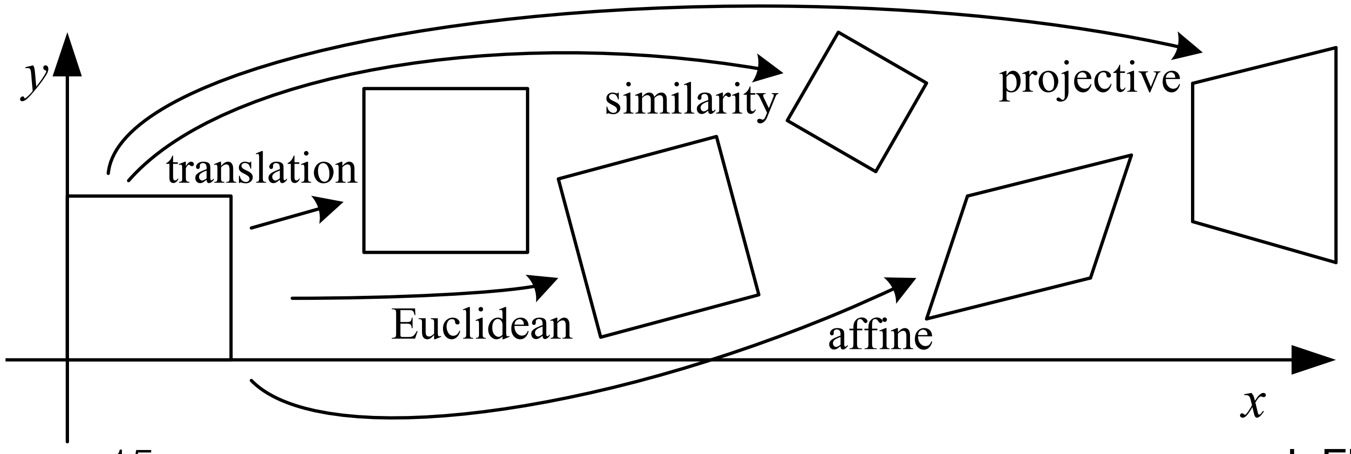
$$\boldsymbol{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

is an orthonormal rotation matrix with $\boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{I}$ and $|\boldsymbol{R}| = 1$.

Preserves Euclidean distances

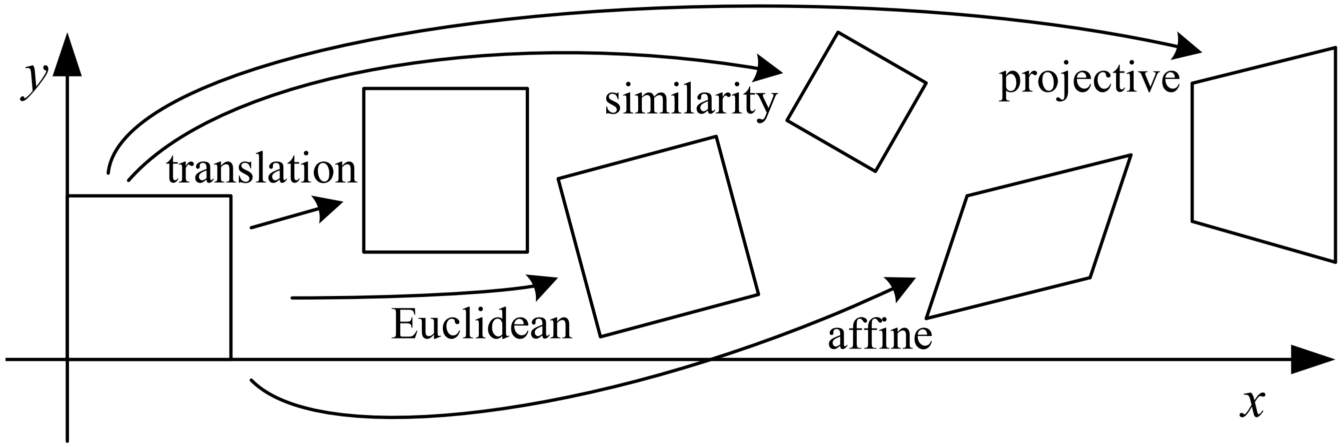


Similarity Transformation



Similarity Transformation

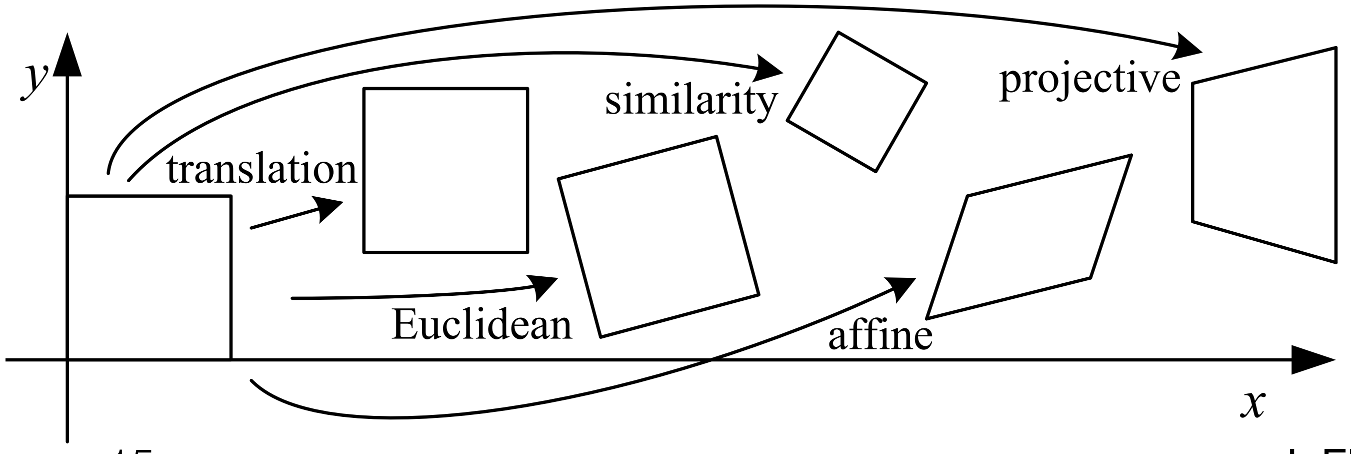
$$\boldsymbol{x}' = s\boldsymbol{R}\boldsymbol{x} + \boldsymbol{t}$$



Similarity Transformation

$$\boldsymbol{x}' = s\boldsymbol{R}\boldsymbol{x} + \boldsymbol{t}$$

$$\boldsymbol{x}' = \begin{bmatrix} s\boldsymbol{R} & \boldsymbol{t} \end{bmatrix} \bar{\boldsymbol{x}} = \begin{bmatrix} a & -b & t_x \\ b & a & t_y \end{bmatrix} \bar{\boldsymbol{x}}$$

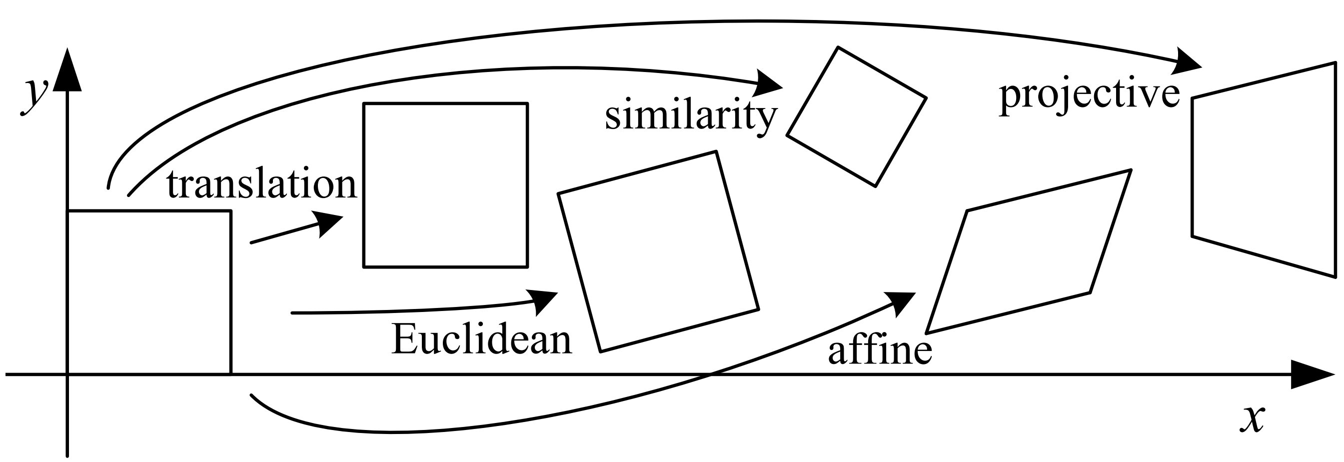


Similarity Transformation

$$\boldsymbol{x}' = s\boldsymbol{R}\boldsymbol{x} + \boldsymbol{t}$$

$$\boldsymbol{x}' = \begin{bmatrix} s\boldsymbol{R} & \boldsymbol{t} \end{bmatrix} \bar{\boldsymbol{x}} = \begin{bmatrix} a & -b & t_x \\ b & a & t_y \end{bmatrix} \bar{\boldsymbol{x}}$$

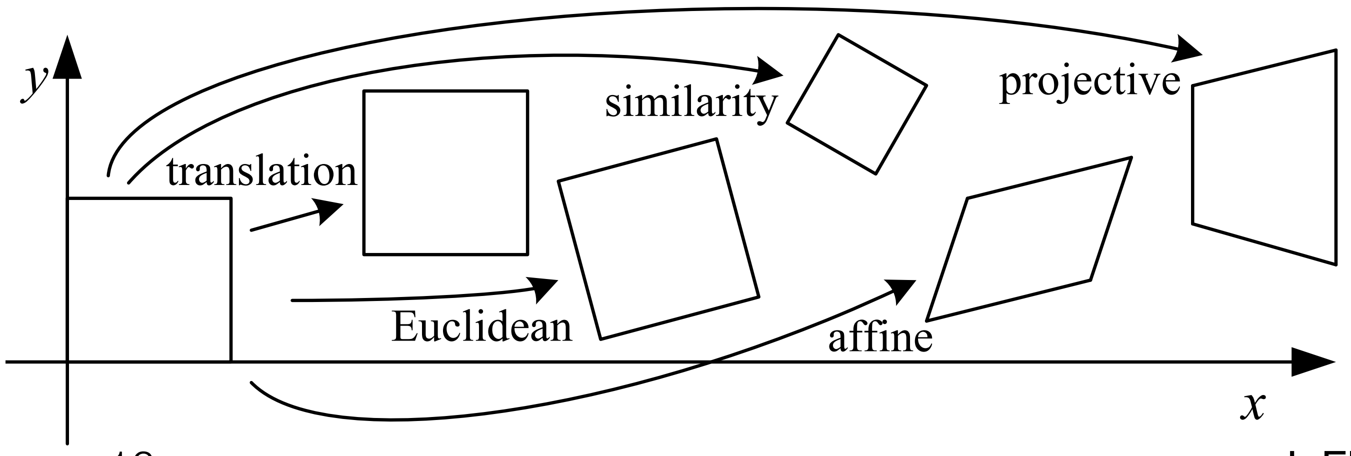
Preserves angles



Affine Transformation

$\boldsymbol{x}' = \boldsymbol{A}\bar{\boldsymbol{x}}$, where $\boldsymbol{A}$ is an arbitrary 2×3 matrix

$$\boldsymbol{x}' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \bar{\boldsymbol{x}}$$

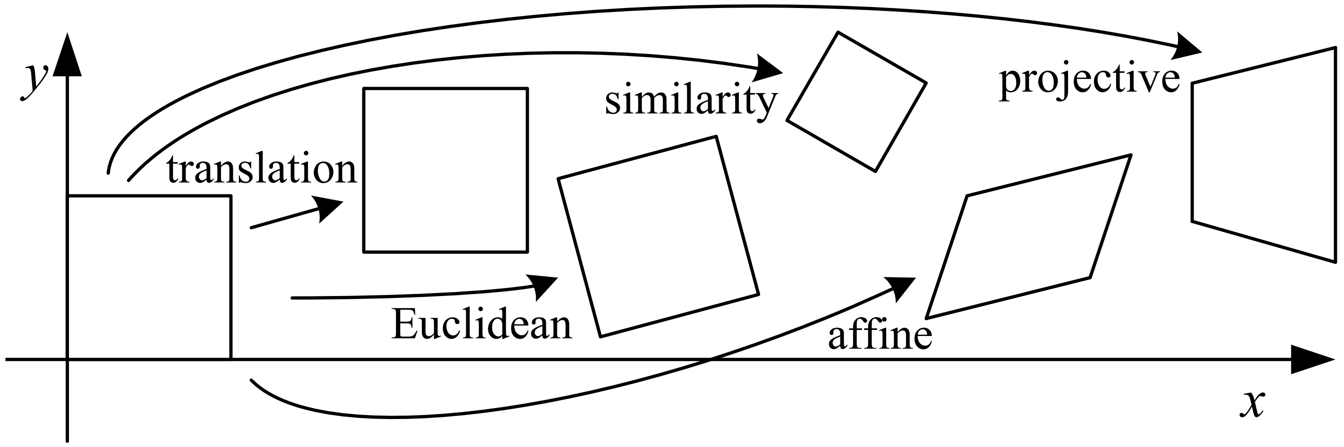


Affine Transformation

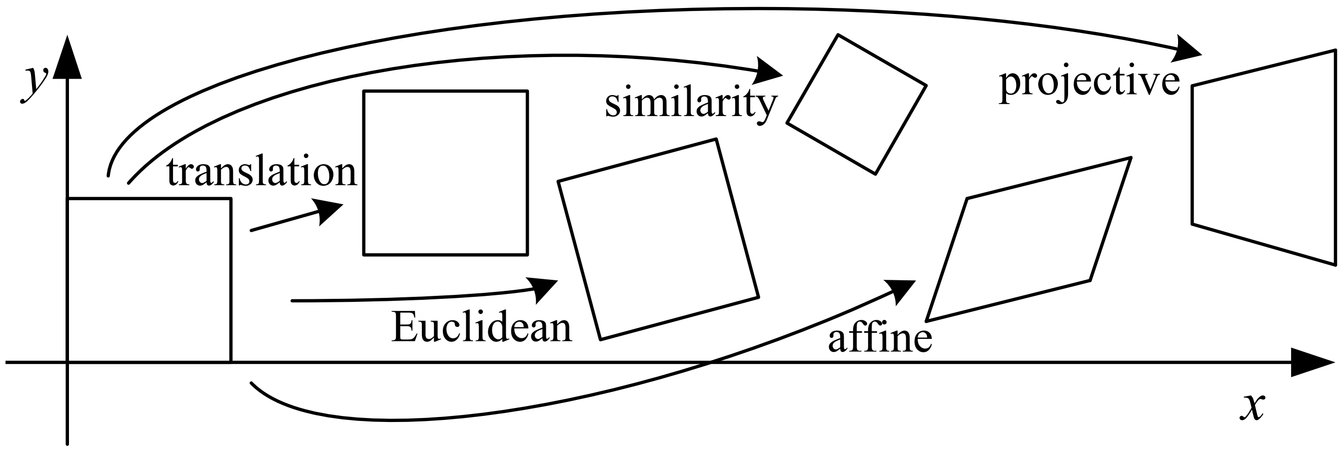
$$\boldsymbol{x}' = \boldsymbol{A}\bar{\boldsymbol{x}}, \text{ where } \boldsymbol{A} \text{ is an arbitrary } 2 \times 3 \text{ matrix}$$

$$\boldsymbol{x}' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \bar{\boldsymbol{x}}$$

Preserves parallelism

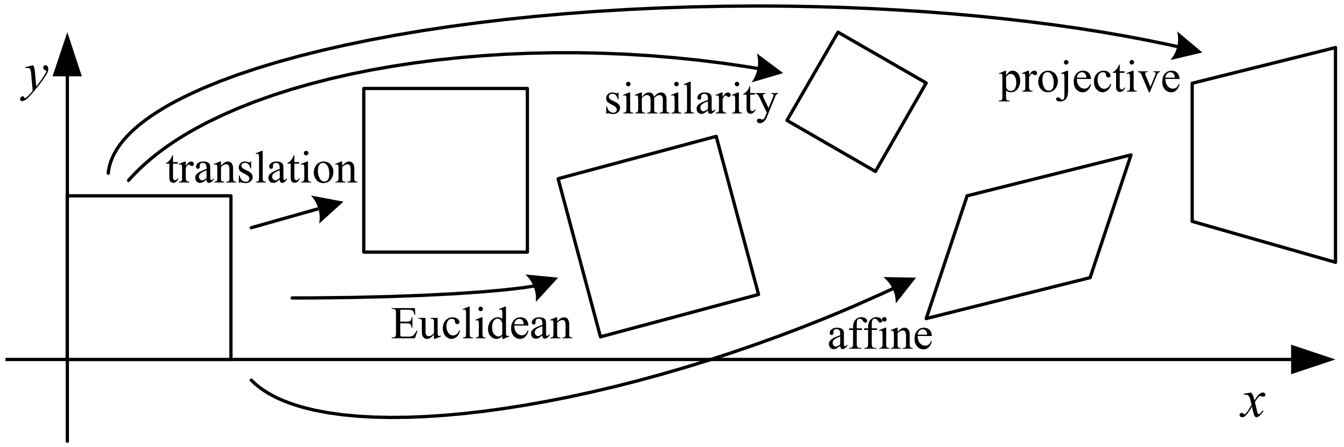


Projective Transformation (Homography)



Projective Transformation (Homography)

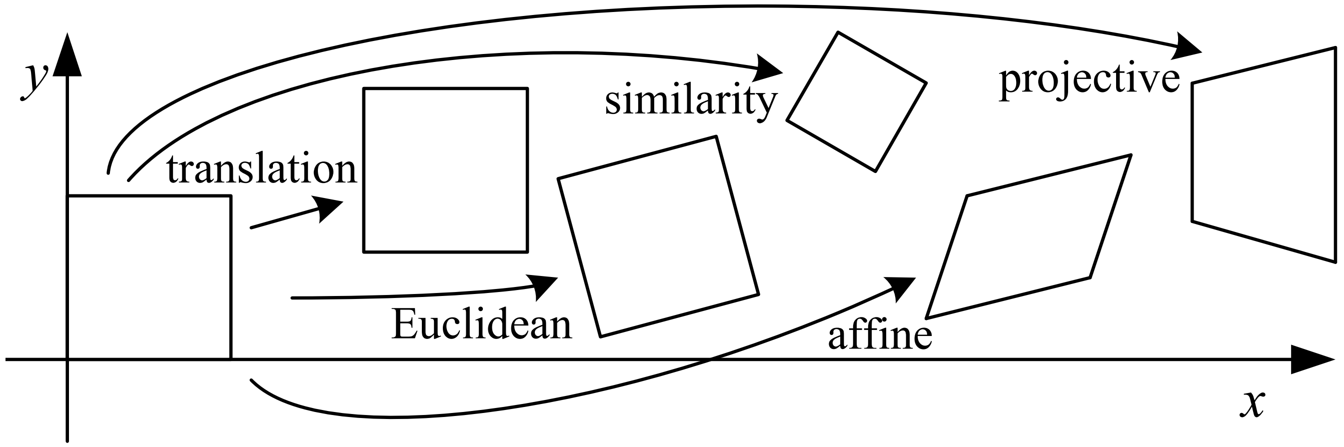
$$\tilde{\boldsymbol{x}}' = \tilde{\boldsymbol{H}}\tilde{\boldsymbol{x}}$$



Projective Transformation (Homography)

$$\tilde{\boldsymbol{x}}' = \tilde{\boldsymbol{H}}\tilde{\boldsymbol{x}}$$

where $\tilde{\boldsymbol{H}}$ is an arbitrary 3×3 matrix.



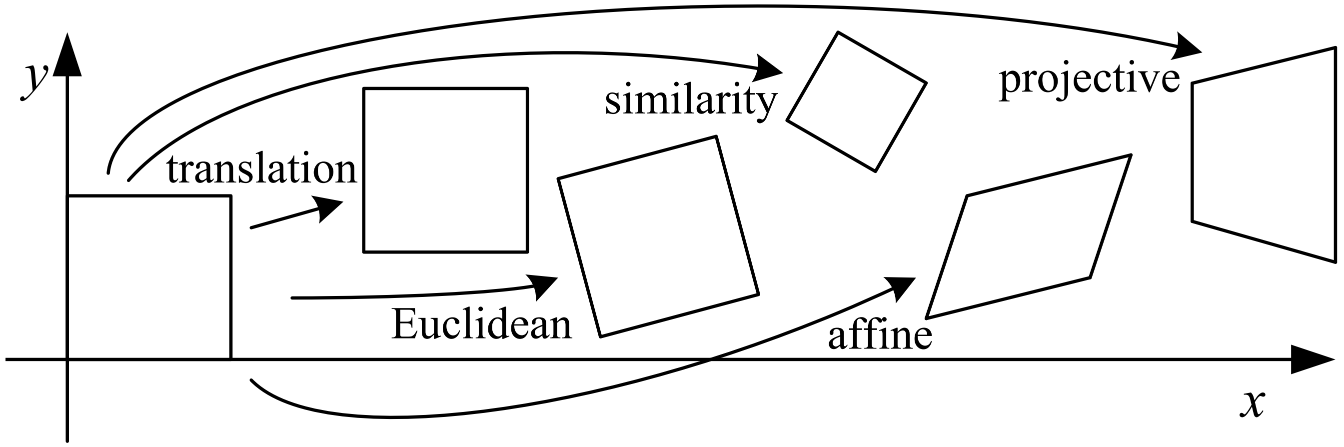
Projective Transformation (Homography)

$$\tilde{\boldsymbol{x}}' = \tilde{\boldsymbol{H}}\tilde{\boldsymbol{x}}$$

where $\tilde{\boldsymbol{H}}$ is an arbitrary 3×3 matrix.

$\tilde{\boldsymbol{H}}$ is homogenous:

Two $\tilde{\boldsymbol{H}}$ matrices that differ only by a scale factor are equivalent.



Projective Transformation (Homography)

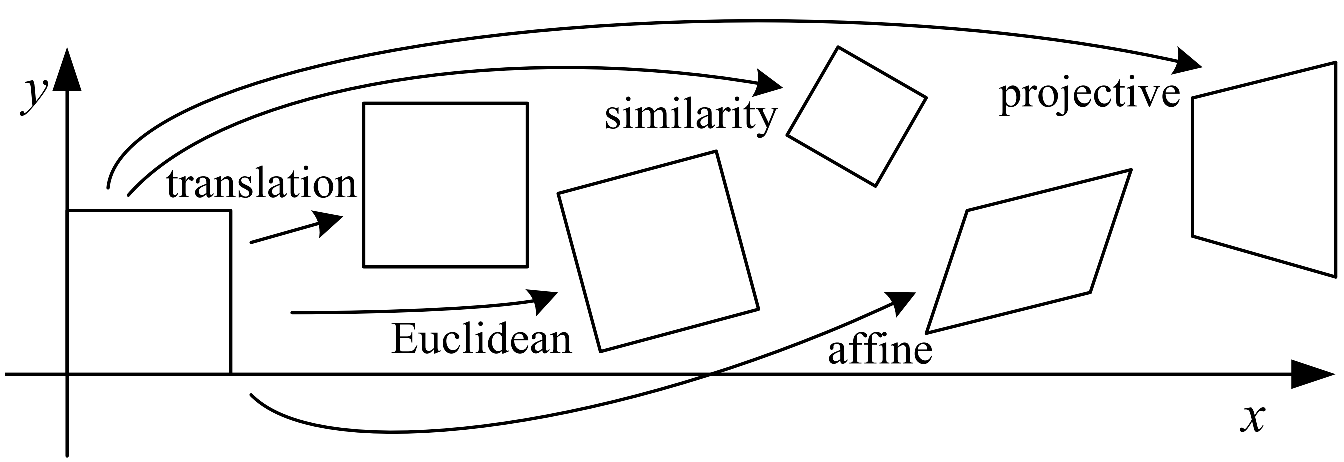
$$\tilde{\boldsymbol{x}}' = \tilde{\boldsymbol{H}} \tilde{\boldsymbol{x}}$$

where $\tilde{\boldsymbol{H}}$ is an arbitrary 3×3 matrix.

$\tilde{\boldsymbol{H}}$ is homogenous:

Two $\tilde{\boldsymbol{H}}$ matrices that differ only by a scale factor are equivalent.

$$x' = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + h_{22}} \text{ and } y' = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + h_{22}}$$



Projective Transformation (Homography)

$$\tilde{\boldsymbol{x}}' = \tilde{\boldsymbol{H}}\tilde{\boldsymbol{x}}$$

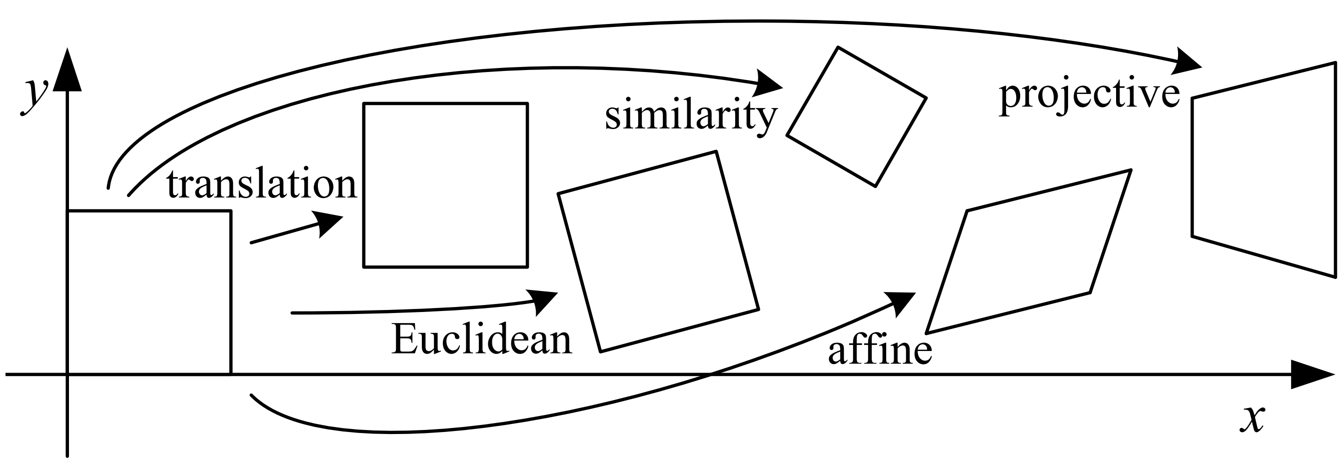
where $\tilde{\boldsymbol{H}}$ is an arbitrary 3×3 matrix.

$\tilde{\boldsymbol{H}}$ is homogenous:

Two $\tilde{\boldsymbol{H}}$ matrices that differ only by a scale factor are equivalent.

$$x' = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + h_{22}} \quad \text{and} \quad y' = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + h_{22}}$$

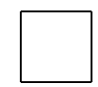
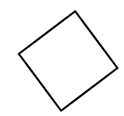
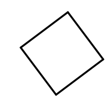
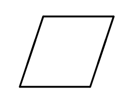
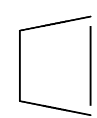
Preserves straight lines

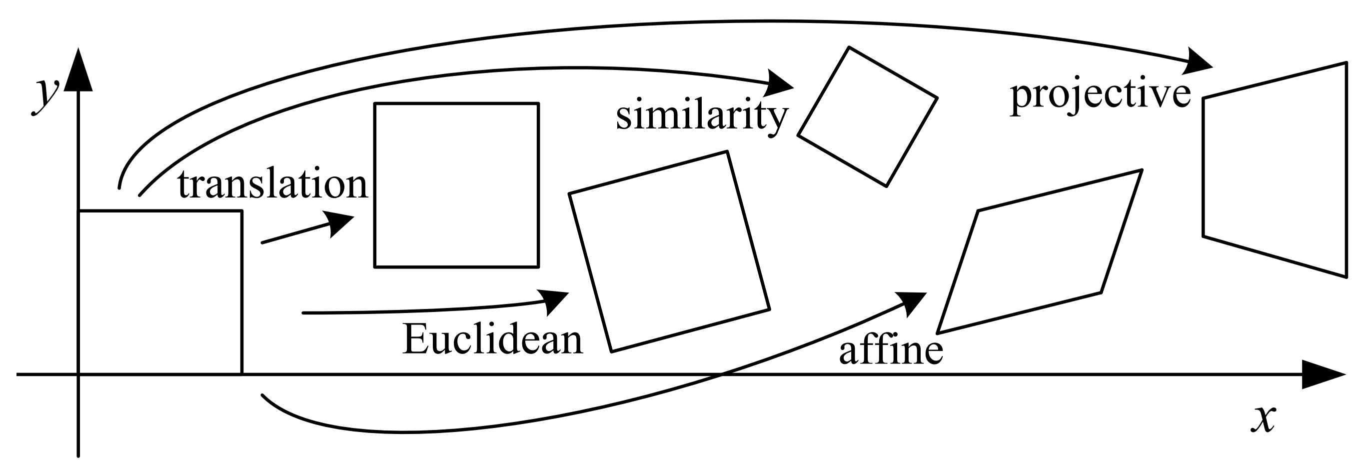


Summary of 2D Transformations

Nested set of groups

- ❖ Closed under composition
- ❖ Each transformation has an inverse that is a member of the same group

Transformation	**Matrix**	**# DoF**	**Preserves**	**Icon**
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	3	lengths	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	4	angles	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times 3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times 3}$	8	straight lines	



Co-vectors

Co-vectors

We now know how to transform points.
How do we transform lines?

Co-vectors

We now know how to transform points.
How do we transform lines?

$$\tilde{\boldsymbol{l}} \cdot \tilde{\boldsymbol{x}} = 0$$

Co-vectors

We now know how to transform points.
How do we transform lines?

$$\tilde{\boldsymbol{l}} \cdot \tilde{\boldsymbol{x}} = 0$$

$$\tilde{\boldsymbol{x}}' = \tilde{\boldsymbol{H}} \tilde{\boldsymbol{x}}$$

Co-vectors

We now know how to transform points.
How do we transform lines?

$$\tilde{\boldsymbol{l}} \cdot \tilde{\boldsymbol{x}} = 0$$

$$\tilde{\boldsymbol{x}}' = \tilde{\boldsymbol{H}} \tilde{\boldsymbol{x}}$$

$$\tilde{\boldsymbol{l}}' \cdot \tilde{\boldsymbol{x}}' = \tilde{\boldsymbol{l}}'^T \tilde{\boldsymbol{H}} \tilde{\boldsymbol{x}} = (\tilde{\boldsymbol{H}}^T \tilde{\boldsymbol{l}}')^T \tilde{\boldsymbol{x}} = \tilde{\boldsymbol{l}} \cdot \tilde{\boldsymbol{x}} = 0$$

Co-vectors

We now know how to transform points.
How do we transform lines?

$$\tilde{\boldsymbol{l}} \cdot \tilde{\boldsymbol{x}} = 0$$

$$\tilde{\boldsymbol{x}}' = \tilde{\boldsymbol{H}} \tilde{\boldsymbol{x}}$$

$$\tilde{\boldsymbol{l}}' \cdot \tilde{\boldsymbol{x}}' = \tilde{\boldsymbol{l}}'^T \tilde{\boldsymbol{H}} \tilde{\boldsymbol{x}} = (\tilde{\boldsymbol{H}}^T \tilde{\boldsymbol{l}}')^T \tilde{\boldsymbol{x}} = \tilde{\boldsymbol{l}} \cdot \tilde{\boldsymbol{x}} = 0$$

Thus

Co-vectors

We now know how to transform points.
How do we transform lines?

$$\tilde{\boldsymbol{l}} \cdot \tilde{\boldsymbol{x}} = 0$$

$$\tilde{\boldsymbol{x}}' = \tilde{\boldsymbol{H}} \tilde{\boldsymbol{x}}$$

$$\tilde{\boldsymbol{l}}' \cdot \tilde{\boldsymbol{x}}' = \tilde{\boldsymbol{l}}'^T \tilde{\boldsymbol{H}} \tilde{\boldsymbol{x}} = (\tilde{\boldsymbol{H}}^T \tilde{\boldsymbol{l}}')^T \tilde{\boldsymbol{x}} = \tilde{\boldsymbol{l}} \cdot \tilde{\boldsymbol{x}} = 0$$

Thus

$$\tilde{\boldsymbol{l}}' = \tilde{\boldsymbol{H}}^{-T} \tilde{\boldsymbol{l}}.$$

Co-vectors

We now know how to transform points.
How do we transform lines?

$$\tilde{\boldsymbol{l}} \cdot \tilde{\boldsymbol{x}} = 0$$

$$\tilde{\boldsymbol{x}}' = \tilde{\boldsymbol{H}}\tilde{\boldsymbol{x}}$$

$$\tilde{\boldsymbol{l}}' \cdot \tilde{\boldsymbol{x}}' = \tilde{\boldsymbol{l}}'^T \tilde{\boldsymbol{H}}\tilde{\boldsymbol{x}} = (\tilde{\boldsymbol{H}}^T \tilde{\boldsymbol{l}}')^T \tilde{\boldsymbol{x}} = \tilde{\boldsymbol{l}} \cdot \tilde{\boldsymbol{x}} = 0$$

Thus

$$\tilde{\boldsymbol{l}}' = \tilde{\boldsymbol{H}}^{-T} \tilde{\boldsymbol{l}}.$$

i.e., the action of a projective transformation on a *co-vector* such as a 2D line can be represented by the transposed inverse of the matrix.

Outline

- Geometric primitives
- 2D transformations
- **3D transformations**
- 3D rotations
- 3D to 2D projections
- Lens Distortions

3D Translation

3D translations can be written as $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{t}$ or

$$\boldsymbol{x}' = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{t} \end{bmatrix} \bar{\boldsymbol{x}}$$

where $\boldsymbol{I}$ is the (3×3) identity matrix

Euclidean Transformation (3D Rotation + Translation)

$$\boldsymbol{x}' = \boldsymbol{R}\boldsymbol{x} + \boldsymbol{t}$$

$$\boldsymbol{x}' = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \end{bmatrix} \bar{\boldsymbol{x}}$$

where $\boldsymbol{R}$ is a 3×3 orthonormal rotation matrix with $\boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{I}$ and $|\boldsymbol{R}| = 1$

Preserves Euclidean distances

Similarity Transformation

$$\boldsymbol{x}' = s\boldsymbol{R}\boldsymbol{x} + \boldsymbol{t}$$

$$\boldsymbol{x}' = \begin{bmatrix} s\boldsymbol{R} & \boldsymbol{t} \end{bmatrix} \bar{\boldsymbol{x}}$$

Preserves angles

Affine Transformation

$\boldsymbol{x}' = \boldsymbol{A}\bar{\boldsymbol{x}}$, where $\boldsymbol{A}$ is an arbitrary 3×4 matrix

$$\boldsymbol{x}' = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \end{bmatrix} \bar{\boldsymbol{x}}$$

Preserves parallelism

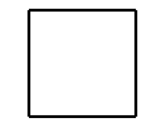
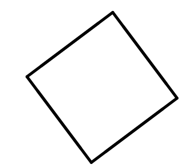
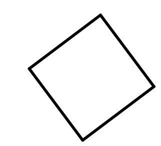
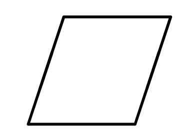
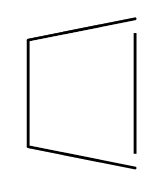
Projective Transformation (Homography)

$$\tilde{\boldsymbol{x}}' = \tilde{\boldsymbol{H}}\tilde{\boldsymbol{x}}$$

where $\tilde{\boldsymbol{H}}$ is an arbitrary 4×4 homogeneous matrix

Preserves straight lines

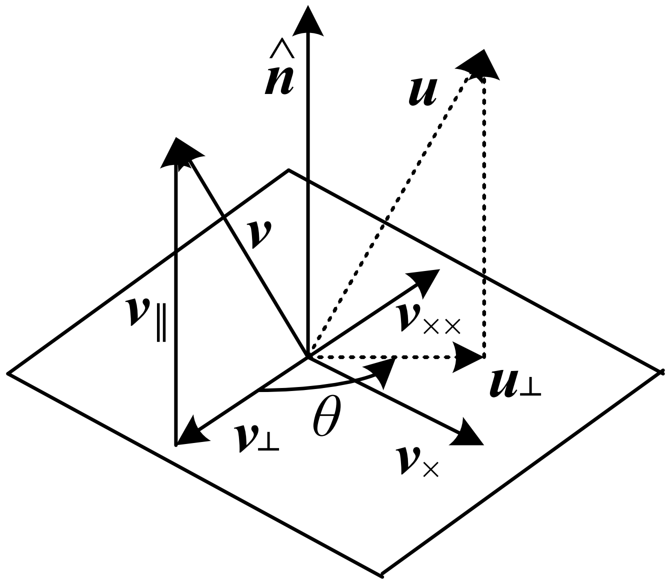
Summary of 3D Transformations

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{3\times 4}$	3	orientation	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{3\times 4}$	6	lengths	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{3\times 4}$	7	angles	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{3\times 4}$	12	parallelism	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{4\times 4}$	15	straight lines	

Outline

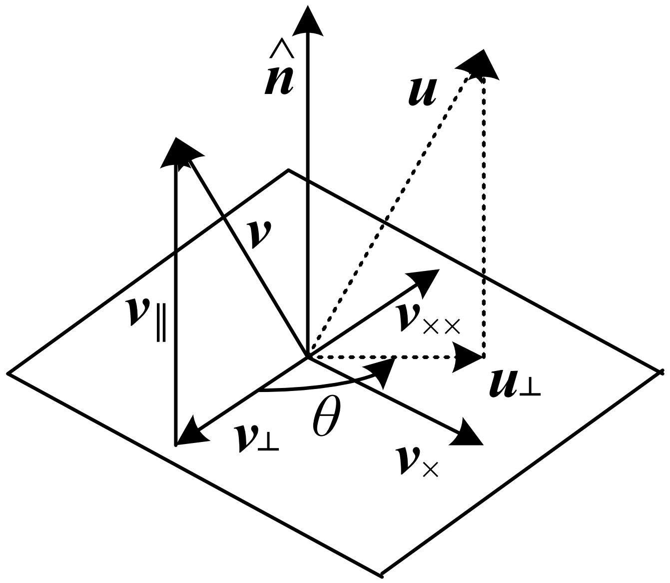
- Geometric primitives
- 2D transformations
- 3D transformations
- **3D rotations**
- 3D to 2D projections
- Lens Distortions

3D Rotations: Axis/Angle Representation



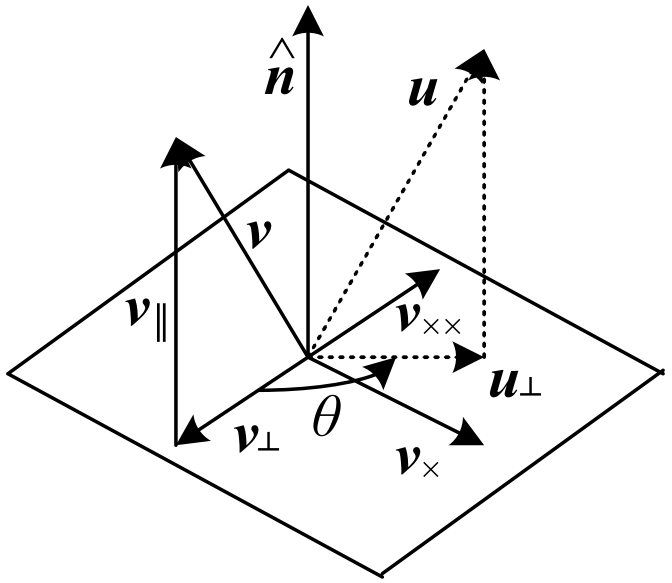
3D Rotations: Axis/Angle Representation

$$\boldsymbol{\omega} = \theta \hat{\boldsymbol{n}}$$



3D Rotations: Axis/Angle Representation

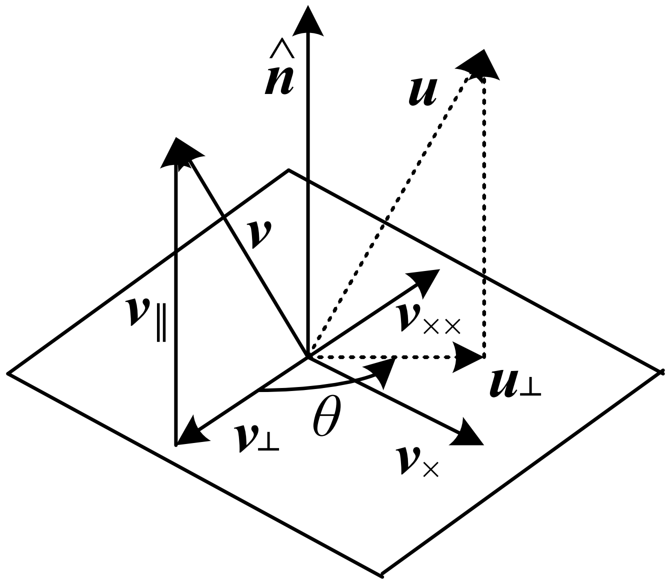
$$\boldsymbol{\omega} = \theta \hat{\boldsymbol{n}}$$



3D Rotations: Axis/Angle Representation

$$\boldsymbol{\omega} = \theta \hat{\boldsymbol{n}}$$

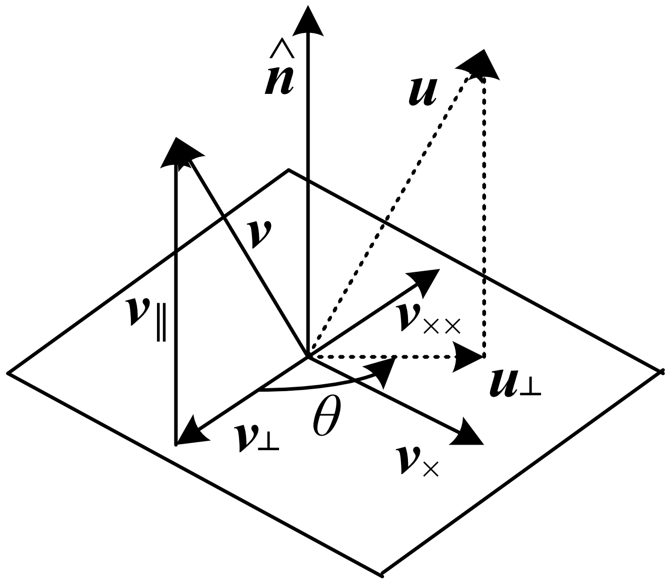
Amount of rotation



3D Rotations: Axis/Angle Representation

$$\boldsymbol{\omega} = \theta \hat{\boldsymbol{n}}$$

Amount of rotation

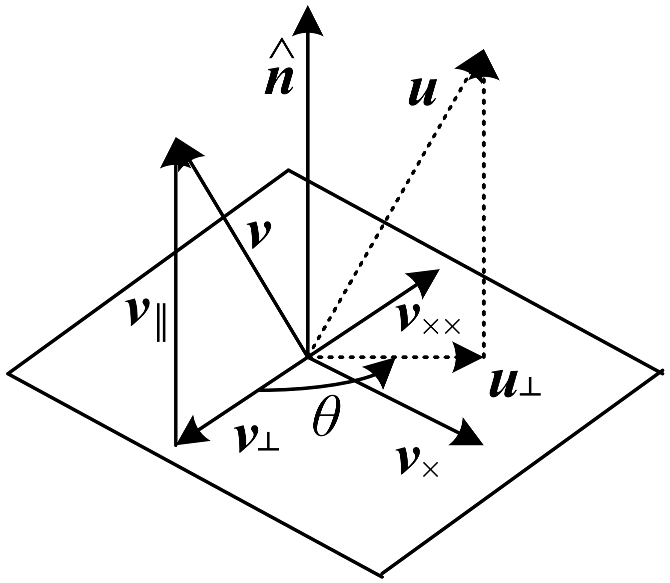


3D Rotations: Axis/Angle Representation

$$\boldsymbol{\omega} = \theta \hat{\boldsymbol{n}}$$

Amount of rotation

Axis of rotation



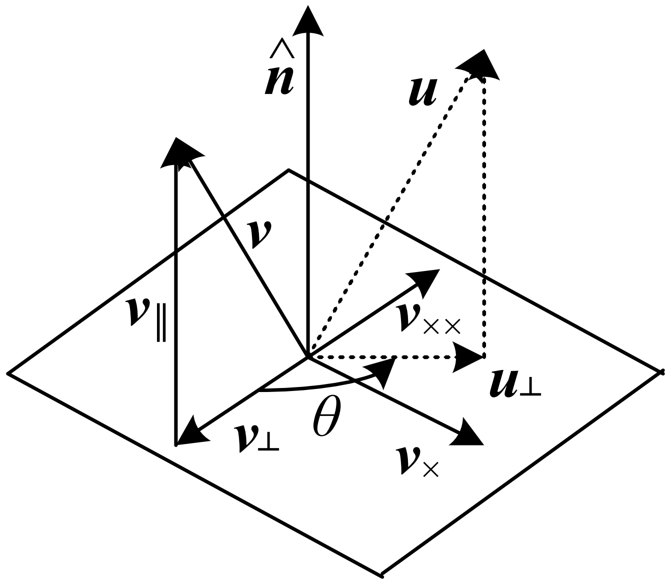
3D Rotations: Axis/Angle Representation

$$\boldsymbol{\omega} = \theta \hat{\boldsymbol{n}}$$

Amount of rotation

Axis of rotation

Let $\boldsymbol{u}$ be the result of rotating vector $\boldsymbol{v}$ about axis $\hat{\boldsymbol{n}}$ by the angle θ.



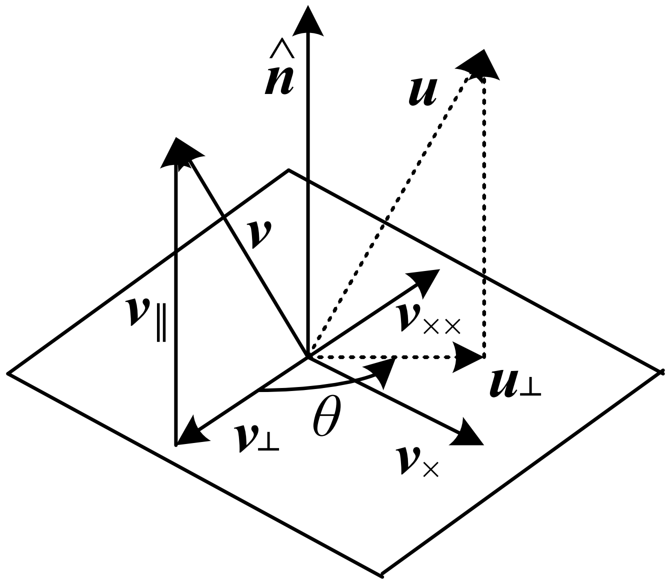
3D Rotations: Axis/Angle Representation

$$\boldsymbol{\omega} = \theta \hat{\boldsymbol{n}}$$

Amount of rotation   Axis of rotation

Let $\boldsymbol{u}$ be the result of rotating vector $\boldsymbol{v}$ about axis $\hat{\boldsymbol{n}}$ by the angle θ.

First, project the vector $\boldsymbol{v}$ onto the axis $\hat{\boldsymbol{n}}$:



3D Rotations: Axis/Angle Representation

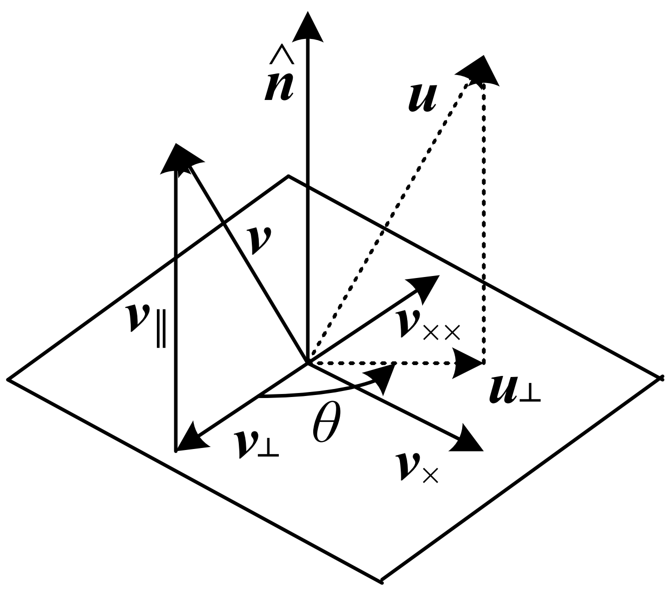
$$\boldsymbol{\omega} = \theta\hat{\boldsymbol{n}}$$

Amount of rotation   Axis of rotation

Let $\boldsymbol{u}$ be the result of rotating vector $\boldsymbol{v}$ about axis $\hat{\boldsymbol{n}}$ by the angle θ.

First, project the vector $\boldsymbol{v}$ onto the axis $\hat{\boldsymbol{n}}$:

$$\boldsymbol{v}_{\|} = \hat{\boldsymbol{n}}(\hat{\boldsymbol{n}} \cdot \boldsymbol{v}) = (\hat{\boldsymbol{n}}\hat{\boldsymbol{n}}^T)\boldsymbol{v}$$



3D Rotations: Axis/Angle Representation

$$\boldsymbol{\omega} = \theta \hat{\boldsymbol{n}}$$

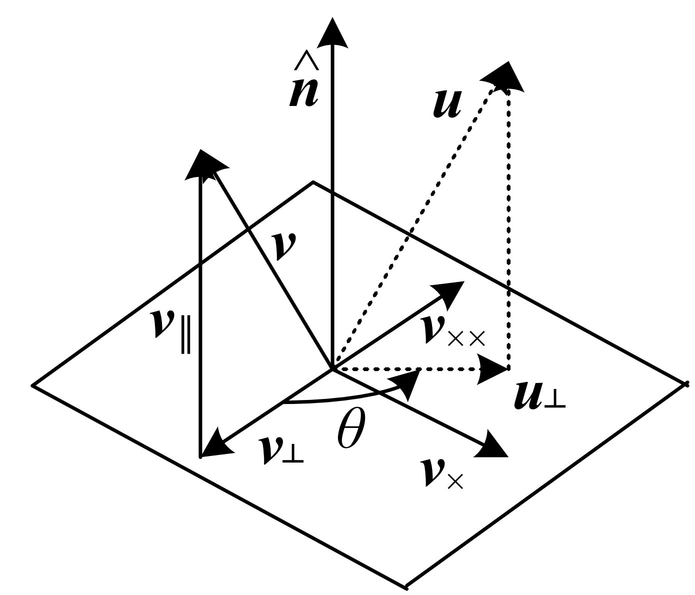
Amount of rotation   Axis of rotation

Let $\boldsymbol{u}$ be the result of rotating vector $\boldsymbol{v}$ about axis $\hat{\boldsymbol{n}}$ by the angle θ.

First, project the vector $\boldsymbol{v}$ onto the axis $\hat{\boldsymbol{n}}$:

$$\boldsymbol{v}_{\parallel} = \hat{\boldsymbol{n}}(\hat{\boldsymbol{n}} \cdot \boldsymbol{v}) = (\hat{\boldsymbol{n}}\hat{\boldsymbol{n}}^T)\boldsymbol{v}$$

Next, compute the perpendicular residual:



3D Rotations: Axis/Angle Representation

$$\boldsymbol{\omega} = \theta \hat{\boldsymbol{n}}$$

Amount of rotation

Axis of rotation

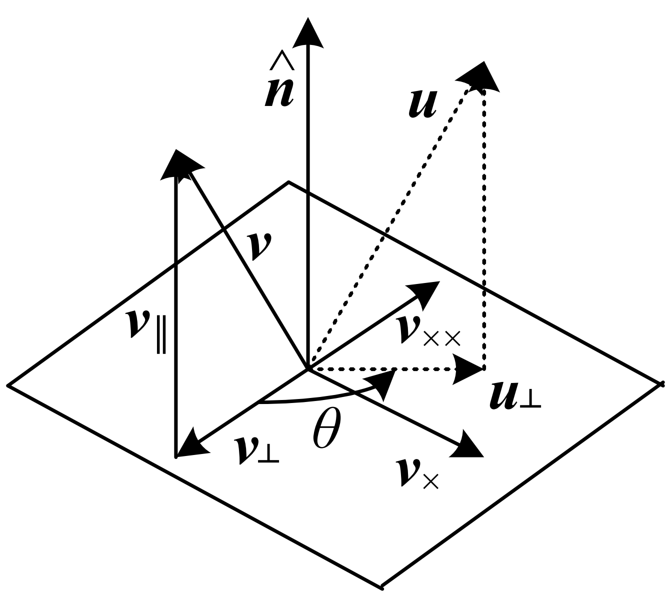
Let $\boldsymbol{u}$ be the result of rotating vector $\boldsymbol{v}$ about axis $\hat{\boldsymbol{n}}$ by the angle θ.

First, project the vector $\boldsymbol{v}$ onto the axis $\hat{\boldsymbol{n}}$:

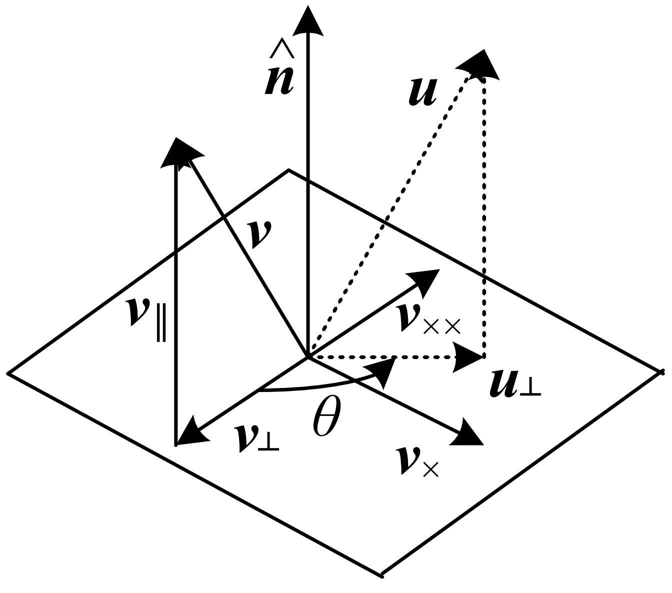
$$\boldsymbol{v}_{\parallel} = \hat{\boldsymbol{n}}(\hat{\boldsymbol{n}} \cdot \boldsymbol{v}) = (\hat{\boldsymbol{n}}\hat{\boldsymbol{n}}^T)\boldsymbol{v}$$

Next, compute the perpendicular residual:

$$\boldsymbol{v}_{\perp} = \boldsymbol{v} - \boldsymbol{v}_{\parallel} = (\boldsymbol{I} - \hat{\boldsymbol{n}}\hat{\boldsymbol{n}}^T)\boldsymbol{v}$$



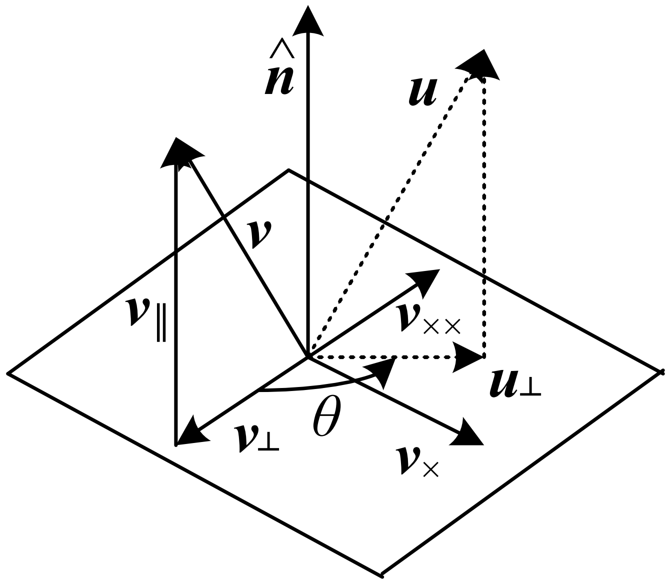
3D Rotations: Axis/Angle Representation



3D Rotations: Axis/Angle Representation

We can rotate this vector by 90° using the cross product,

$$\boldsymbol{v}_\times = \hat{\boldsymbol{n}} \times \boldsymbol{v} = [\hat{\boldsymbol{n}}]_\times \boldsymbol{v},$$

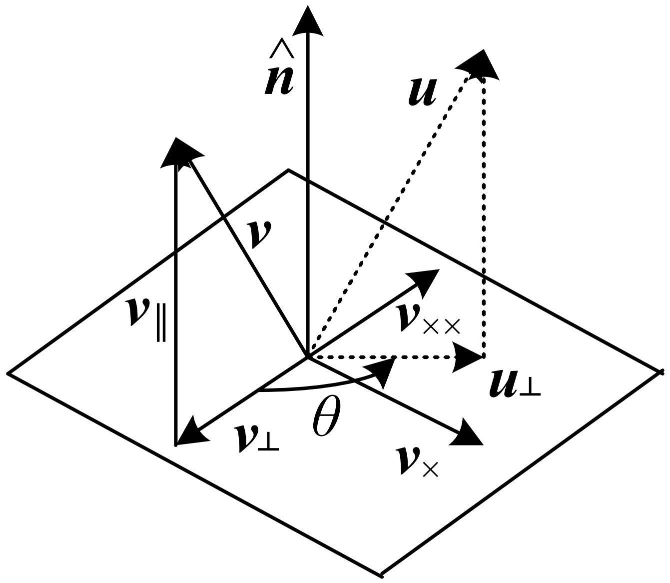


3D Rotations: Axis/Angle Representation

We can rotate this vector by 90° using the cross product,

$$\boldsymbol{v}_\times = \hat{\boldsymbol{n}} \times \boldsymbol{v} = [\hat{\boldsymbol{n}}]_\times \boldsymbol{v},$$

where $[\hat{\boldsymbol{n}}]_\times$ is the matrix form of the cross product operator with the vector $\hat{\boldsymbol{n}} = (\hat{n}_x, \hat{n}_y, \hat{n}_z)$,



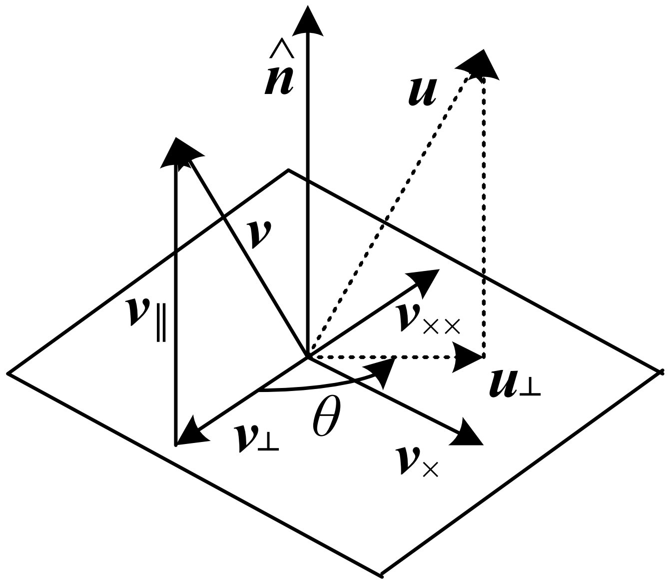
3D Rotations: Axis/Angle Representation

We can rotate this vector by 90° using the cross product,

$$\boldsymbol{v}_\times = \hat{\boldsymbol{n}} \times \boldsymbol{v} = [\hat{\boldsymbol{n}}]_\times \boldsymbol{v},$$

where $[\hat{\boldsymbol{n}}]_\times$ is the matrix form of the cross product operator with the vector $\hat{\boldsymbol{n}} = (\hat{n}_x, \hat{n}_y, \hat{n}_z)$,

$$[\hat{\boldsymbol{n}}]_\times = \begin{bmatrix} 0 & -\hat{n}_z & \hat{n}_y \\ \hat{n}_z & 0 & -\hat{n}_x \\ -\hat{n}_y & \hat{n}_x & 0 \end{bmatrix}$$



3D Rotations: Axis/Angle Representation

We can rotate this vector by $90°$ using the cross product,

$$\boldsymbol{v}_\times = \hat{\boldsymbol{n}} \times \boldsymbol{v} = [\hat{\boldsymbol{n}}]_\times \boldsymbol{v},$$

where $[\hat{\boldsymbol{n}}]_\times$ is the matrix form of the cross product operator with the vector $\hat{\boldsymbol{n}} = (\hat{n}_x, \hat{n}_y, \hat{n}_z)$,

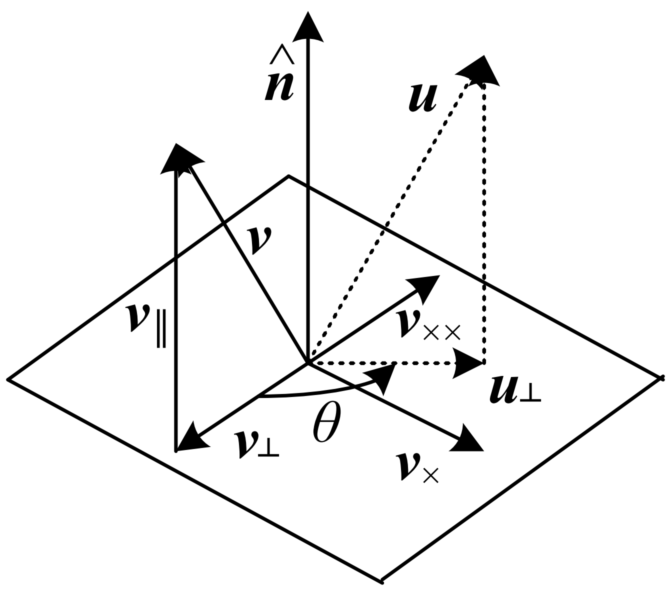
$$[\hat{\boldsymbol{n}}]_\times = \begin{bmatrix} 0 & -\hat{n}_z & \hat{n}_y \\ \hat{n}_z & 0 & -\hat{n}_x \\ -\hat{n}_y & \hat{n}_x & 0 \end{bmatrix}$$

Note that rotating this vector by another $90°$ is equivalent to taking the cross product again,

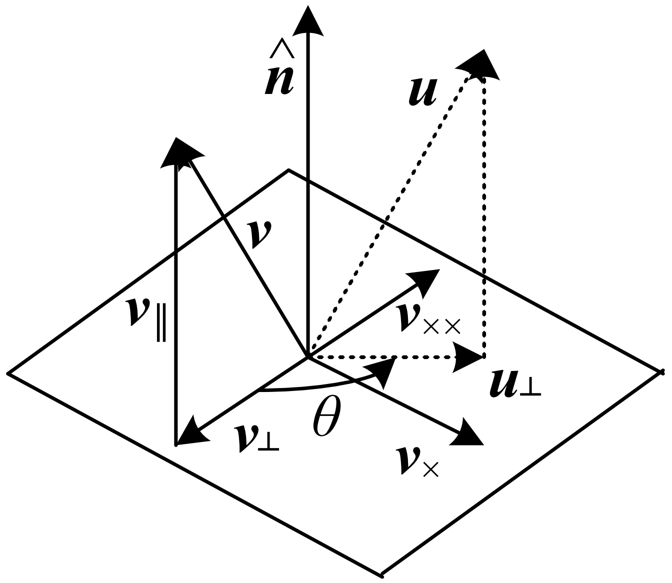
$$\boldsymbol{v}_{\times\times} = \hat{\boldsymbol{n}} \times \boldsymbol{v}_\times = [\hat{\boldsymbol{n}}]^2_\times \boldsymbol{v} = -\boldsymbol{v}_\perp,$$

and hence

$$\boldsymbol{v}_\parallel = \boldsymbol{v} - \boldsymbol{v}_\perp = \boldsymbol{v} + \boldsymbol{v}_{\times\times} = (\boldsymbol{I} + [\hat{\boldsymbol{n}}]^2_\times)\boldsymbol{v}.$$



3D Rotations: Axis/Angle Representation



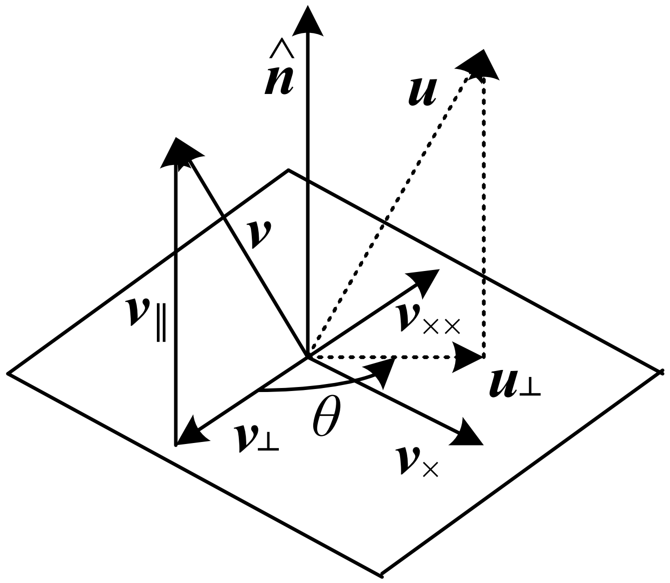
3D Rotations: Axis/Angle Representation

We can now compute the in-plane component of the rotated vector $\boldsymbol{u}$ as

$$\boldsymbol{u}_\perp = \cos\theta \boldsymbol{v}_\perp + \sin\theta \boldsymbol{v}_\times = (\sin\theta[\hat{\boldsymbol{n}}]_\times - \cos\theta[\hat{\boldsymbol{n}}]_\times^2)\boldsymbol{v}.$$

Putting all these terms together, we obtain the final rotated vector as

$$\boldsymbol{u} = \boldsymbol{u}_\perp + \boldsymbol{v}_\parallel = (\boldsymbol{I} + \sin\theta[\hat{\boldsymbol{n}}]_\times + (1-\cos\theta)[\hat{\boldsymbol{n}}]_\times^2)\boldsymbol{v}.$$



3D Rotations: Axis/Angle Representation

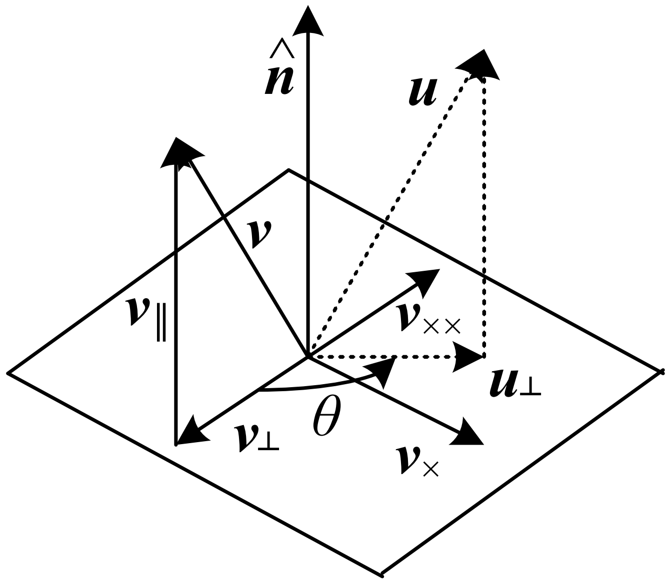
We can now compute the in-plane component of the rotated vector $\boldsymbol{u}$ as

$$\boldsymbol{u}_\perp = \cos\theta \boldsymbol{v}_\perp + \sin\theta \boldsymbol{v}_\times = (\sin\theta[\hat{\boldsymbol{n}}]_\times - \cos\theta[\hat{\boldsymbol{n}}]_\times^2)\boldsymbol{v}.$$

Putting all these terms together, we obtain the final rotated vector as

$$\boldsymbol{u} = \boldsymbol{u}_\perp + \boldsymbol{v}_\parallel = (\boldsymbol{I} + \sin\theta[\hat{\boldsymbol{n}}]_\times + (1 - \cos\theta)[\hat{\boldsymbol{n}}]_\times^2)\boldsymbol{v}.$$

We can therefore write the rotation matrix corresponding to a rotation by θ around an axis $\hat{\boldsymbol{n}}$ as



3D Rotations: Axis/Angle Representation

We can now compute the in-plane component of the rotated vector $\boldsymbol{u}$ as

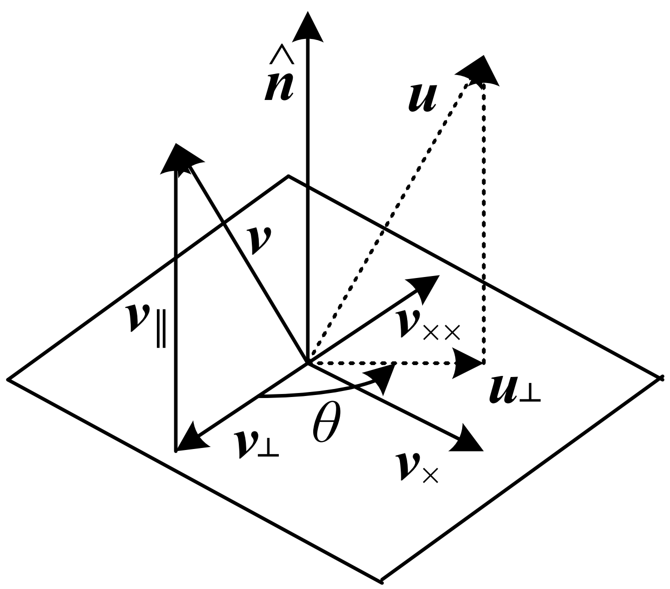
$$\boldsymbol{u}_\perp = \cos\theta\boldsymbol{v}_\perp + \sin\theta\boldsymbol{v}_\times = (\sin\theta[\hat{\boldsymbol{n}}]_\times - \cos\theta[\hat{\boldsymbol{n}}]_\times^2)\boldsymbol{v}.$$

Putting all these terms together, we obtain the final rotated vector as

$$\boldsymbol{u} = \boldsymbol{u}_\perp + \boldsymbol{v}_\parallel = (\boldsymbol{I} + \sin\theta[\hat{\boldsymbol{n}}]_\times + (1-\cos\theta)[\hat{\boldsymbol{n}}]_\times^2)\boldsymbol{v}.$$

We can therefore write the rotation matrix corresponding to a rotation by θ around an axis $\hat{\boldsymbol{n}}$ as

$$\boldsymbol{R}(\hat{\boldsymbol{n}},\theta) = \boldsymbol{I} + \sin\theta[\hat{\boldsymbol{n}}]_\times + (1-\cos\theta)[\hat{\boldsymbol{n}}]_\times^2$$



3D Rotations: Axis/Angle Representation

We can now compute the in-plane component of the rotated vector $\boldsymbol{u}$ as

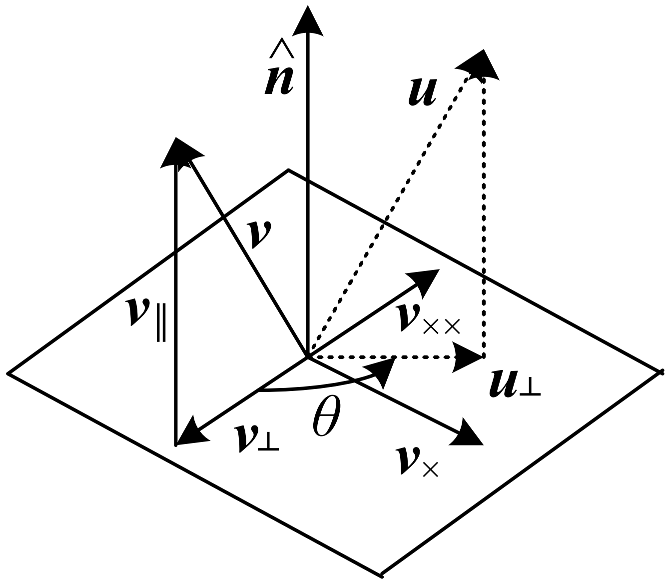
$$\boldsymbol{u}_\perp = \cos\theta \boldsymbol{v}_\perp + \sin\theta \boldsymbol{v}_\times = (\sin\theta[\hat{\boldsymbol{n}}]_\times - \cos\theta[\hat{\boldsymbol{n}}]^2_\times)\boldsymbol{v}.$$

Putting all these terms together, we obtain the final rotated vector as

$$\boldsymbol{u} = \boldsymbol{u}_\perp + \boldsymbol{v}_\| = (\boldsymbol{I} + \sin\theta[\hat{\boldsymbol{n}}]_\times + (1 - \cos\theta)[\hat{\boldsymbol{n}}]^2_\times)\boldsymbol{v}.$$

We can therefore write the rotation matrix corresponding to a rotation by θ around an axis $\hat{\boldsymbol{n}}$ as

$$\boldsymbol{R}(\hat{\boldsymbol{n}}, \theta) = \boldsymbol{I} + \sin\theta[\hat{\boldsymbol{n}}]_\times + (1 - \cos\theta)[\hat{\boldsymbol{n}}]^2_\times$$ **(Rodriquez' formula)**

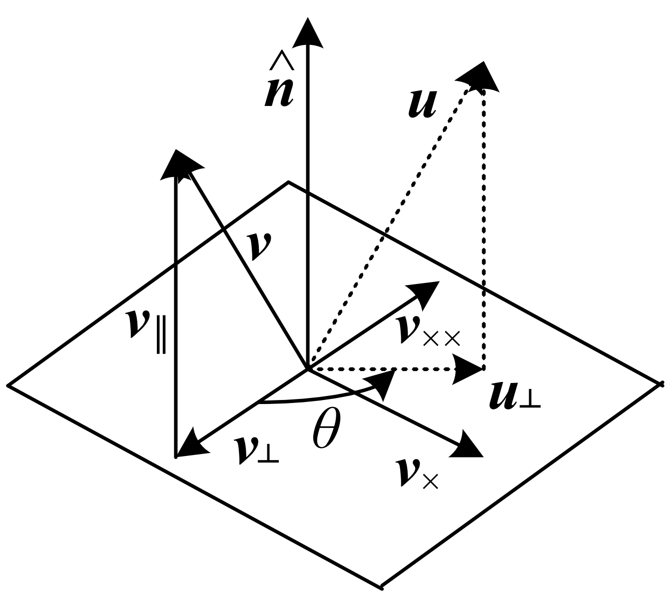


3D Rotations: Axis/Angle Representation

$$\boldsymbol{R}(\hat{\boldsymbol{n}}, \theta) = \boldsymbol{I} + \sin\theta [\hat{\boldsymbol{n}}]_\times + (1 - \cos\theta)[\hat{\boldsymbol{n}}]^2_\times$$ **(Rodriquez' formula)**

For small rotations:

$$\boldsymbol{R}(\boldsymbol{\omega}) \approx \boldsymbol{I} + \sin\theta [\hat{\boldsymbol{n}}]_\times \approx \boldsymbol{I} + [\theta \hat{\boldsymbol{n}}]_\times = \begin{bmatrix} 1 & -\omega_z & \omega_y \\ \omega_z & 1 & -\omega_x \\ -\omega_y & \omega_x & 1 \end{bmatrix}$$



Unit Quaternions

$$\boldsymbol{q} = (x, y, z, w) \text{ with } \|\boldsymbol{q}\| = 1$$

q and ***-q*** represent the same rotation.

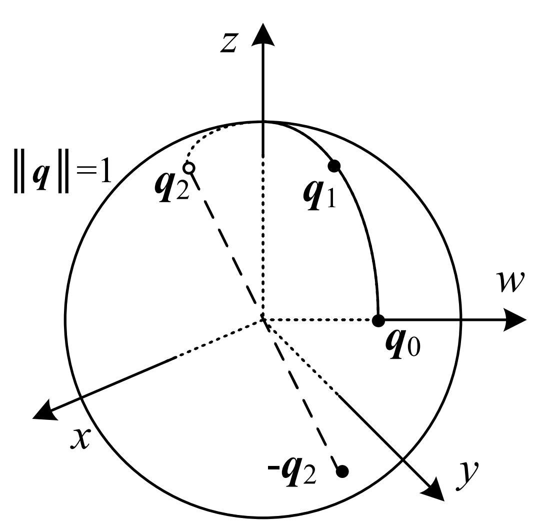
Quaternions can be derived from the axis/angle representation through the formula

$$\boldsymbol{q} = (\boldsymbol{v}, w) = (\sin\frac{\theta}{2}\hat{\boldsymbol{n}}, \cos\frac{\theta}{2}),$$

where $\hat{\boldsymbol{n}}$ and θ are the rotation axis and angle.

Rodriguez' formula now becomes (see textbook):

$$\begin{aligned} \boldsymbol{R}(\hat{\boldsymbol{n}}, \theta) &= \boldsymbol{I} + \sin\theta[\hat{\boldsymbol{n}}]_\times + (1 - \cos\theta)[\hat{\boldsymbol{n}}]_\times^2 \\ &= \boldsymbol{I} + 2w[\boldsymbol{v}]_\times + 2[\boldsymbol{v}]_\times^2. \end{aligned}$$



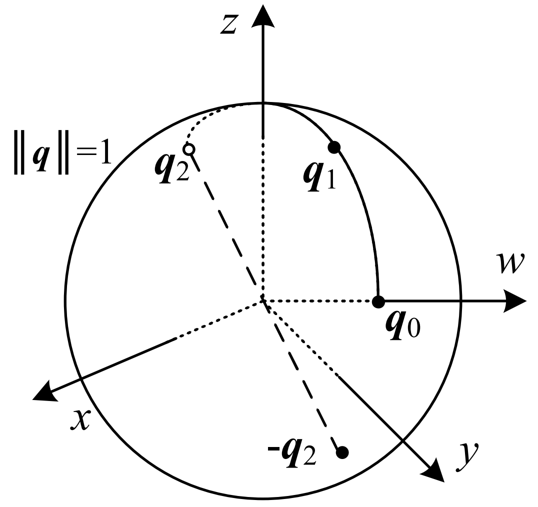
Quaternion Algebra

$$\boldsymbol{q} = (\boldsymbol{v}, w) = (\sin\frac{\theta}{2}\hat{\boldsymbol{n}}, \cos\frac{\theta}{2})$$

Composition (multiplication): $\boldsymbol{q}_2 = \boldsymbol{q}_0\boldsymbol{q}_1 = (\boldsymbol{v}_0 \times \boldsymbol{v}_1 + w_0\boldsymbol{v}_1 + w_1\boldsymbol{v}_0,\ w_0w_1 - \boldsymbol{v}_0 \cdot \boldsymbol{v}_1)$

$$\boldsymbol{R}(\boldsymbol{q}_2) = \boldsymbol{R}(\boldsymbol{q}_0)\boldsymbol{R}(\boldsymbol{q}_1).$$

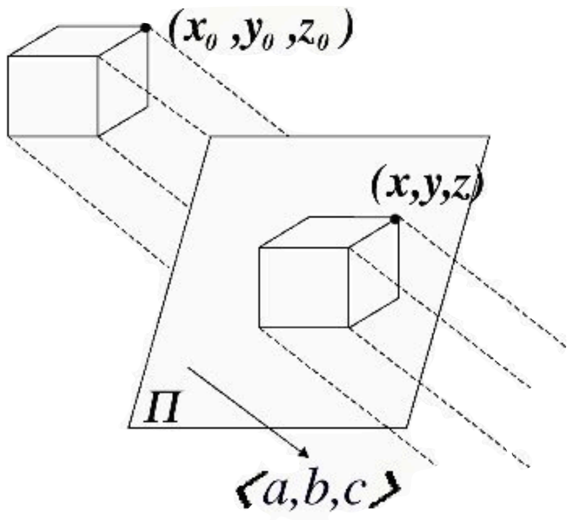
Inverse: flip the sign of *v* or *w* (but not both).
i.e., if $\boldsymbol{q} = (\boldsymbol{v}, w)$, then $\boldsymbol{q}^{-1} = (-\boldsymbol{v}, w) = (\boldsymbol{v}, -w)$.



Outline

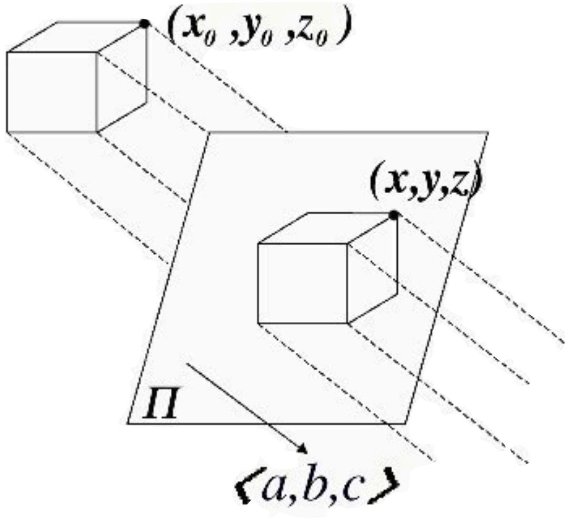
- Geometric primitives
- 2D transformations
- 3D transformations
- 3D rotations
- **3D to 2D projections**
- Lens Distortions

Orthographic (Parallel) Projection



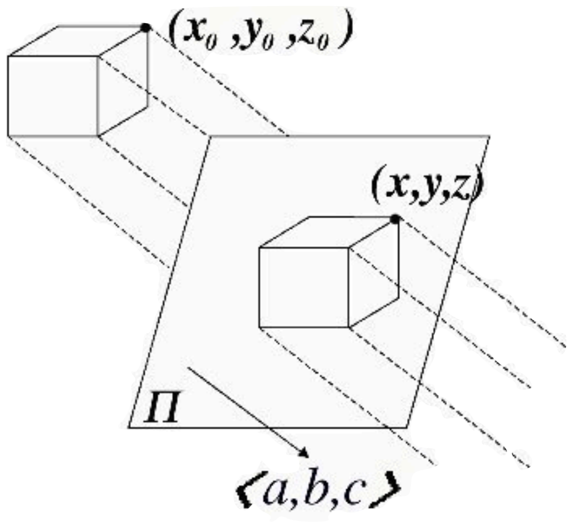
Orthographic (Parallel) Projection

- Reasonable approximation to perspective projection when % depth variation within field of view is small.



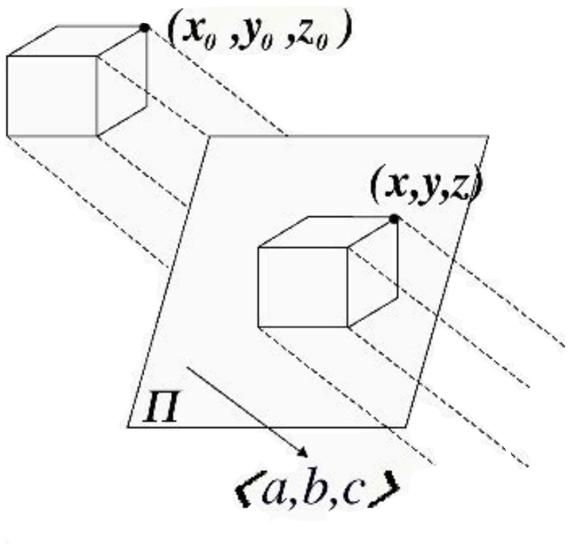
Orthographic (Parallel) Projection

- Reasonable approximation to perspective projection when % depth variation within field of view is small.
- This is often the case for telephoto lenses (long viewing distances, small field of view)



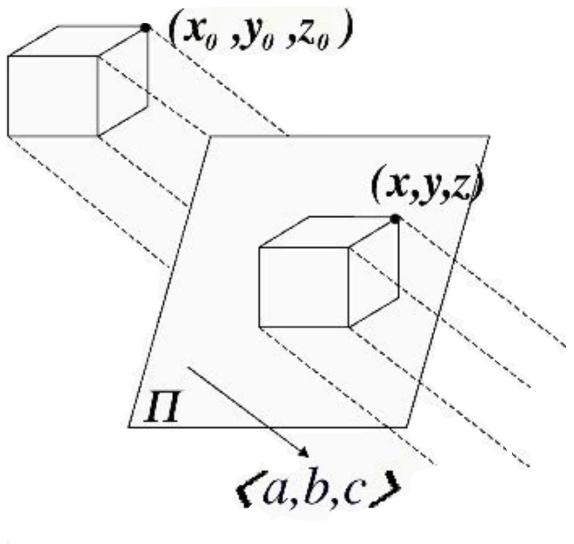
Orthographic (Parallel) Projection

- Reasonable approximation to perspective projection when % depth variation within field of view is small.
- This is often the case for telephoto lenses (long viewing distances, small field of view)
- Given camera-aligned world coordinate frame, simply drop the z component!



Orthographic (Parallel) Projection

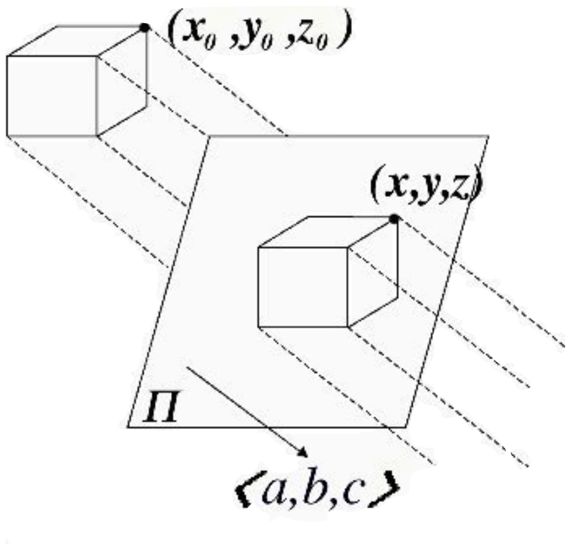
- Reasonable approximation to perspective projection when % depth variation within field of view is small.
- This is often the case for telephoto lenses (long viewing distances, small field of view)
- Given camera-aligned world coordinate frame, simply drop the z component!
- In inhomogeneous (Euclidean) coordinates:



Orthographic (Parallel) Projection

- Reasonable approximation to perspective projection when % depth variation within field of view is small.
- This is often the case for telephoto lenses (long viewing distances, small field of view)
- Given camera-aligned world coordinate frame, simply drop the z component!
- In inhomogeneous (Euclidean) coordinates:

$$\boldsymbol{x} = [\boldsymbol{I}_{2\times 2} | \mathbf{0}]\, \boldsymbol{p}$$



Orthographic (Parallel) Projection

- Reasonable approximation to perspective projection when % depth variation within field of view is small.

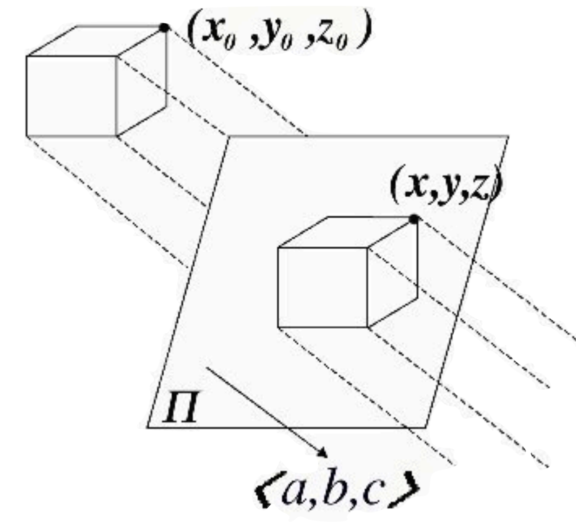
- This is often the case for telephoto lenses (long viewing distances, small field of view)

- Given camera-aligned world coordinate frame, simply drop the z component!

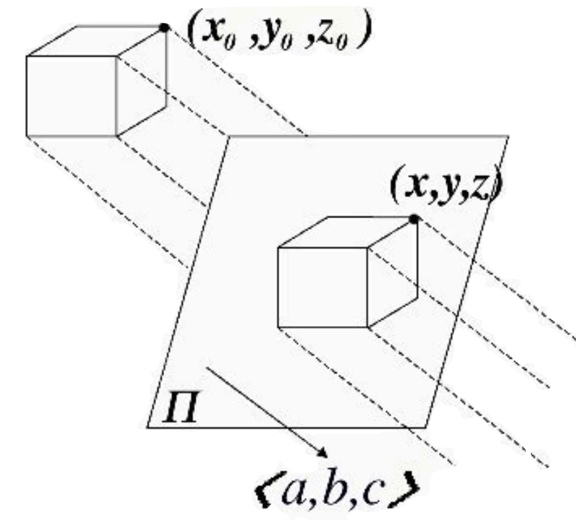
- In inhomogeneous (Euclidean) coordinates:

$$\boldsymbol{x} = [\boldsymbol{I}_{2\times 2}|\mathbf{0}]\ \boldsymbol{p}$$

where $\boldsymbol{p}$ is the 3D point and $\boldsymbol{x}$ is the projected 2D image point



Orthographic (Parallel) Projection



- Reasonable approximation to perspective projection when % depth variation within field of view is small.
- This is often the case for telephoto lenses (long viewing distances, small field of view)
- Given camera-aligned world coordinate frame, simply drop the z component!
- In inhomogeneous (Euclidean) coordinates:

  $$\boldsymbol{x} = [\boldsymbol{I}_{2\times 2}|\boldsymbol{0}]\,\boldsymbol{p}$$

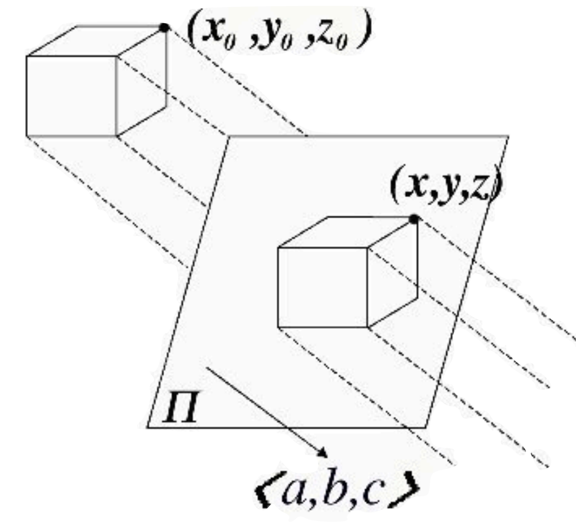
  where $\boldsymbol{p}$ is the 3D point and $\boldsymbol{x}$ is the projected 2D image point
- In practice, we also need to scale the x and y coordinates from metres to pixels:

Orthographic (Parallel) Projection

- Reasonable approximation to perspective projection when % depth variation within field of view is small.

- This is often the case for telephoto lenses (long viewing distances, small field of view)

- Given camera-aligned world coordinate frame, simply drop the z component!



- In inhomogeneous (Euclidean) coordinates:

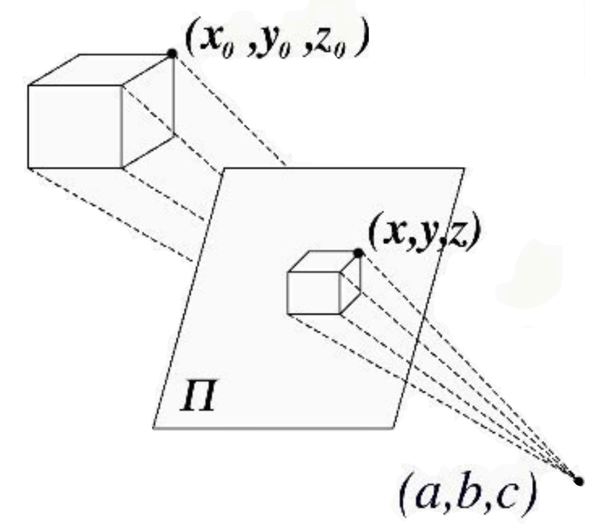
  $$\boldsymbol{x} = [\boldsymbol{I}_{2\times 2}|\boldsymbol{0}]\ \boldsymbol{p}$$

  where $\boldsymbol{p}$ is the 3D point and $\boldsymbol{x}$ is the projected 2D image point

- In practice, we also need to scale the x and y coordinates from metres to pixels:

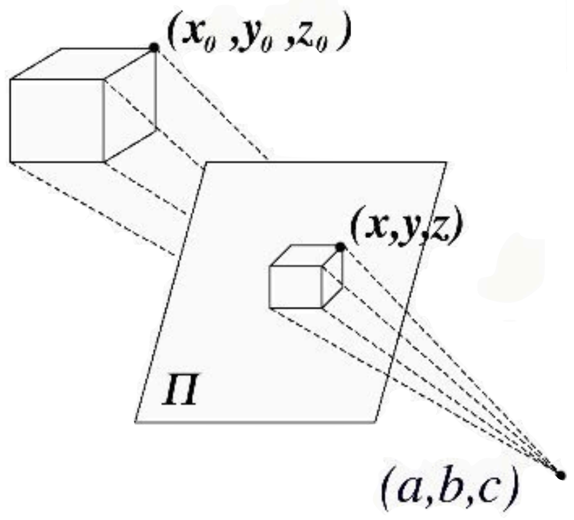
  $$\boldsymbol{x} = [s\boldsymbol{I}_{2\times 2}|\boldsymbol{0}]\ \boldsymbol{p}.$$

Perspective Projection



Perspective Projection

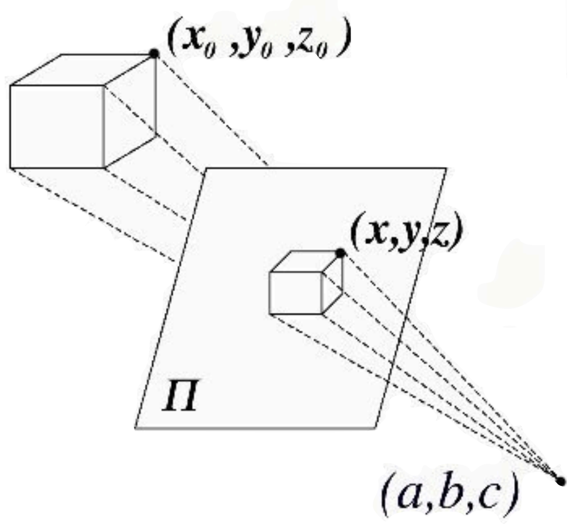
- Points projected onto image plane by dividing them by their z component.



Perspective Projection

❖ Points projected onto image plane by dividing them by their z component.

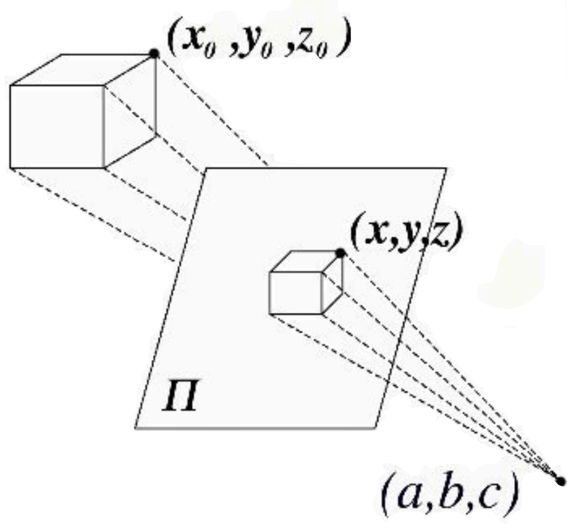
$$\bar{\boldsymbol{x}} = \mathcal{P}_z(\boldsymbol{p}) = \begin{bmatrix} x/z \\ y/z \\ 1 \end{bmatrix}$$



Perspective Projection

- Points projected onto image plane by dividing them by their z component.

$$\bar{\boldsymbol{x}} = \mathcal{P}_z(\boldsymbol{p}) = \begin{bmatrix} x/z \\ y/z \\ 1 \end{bmatrix}$$

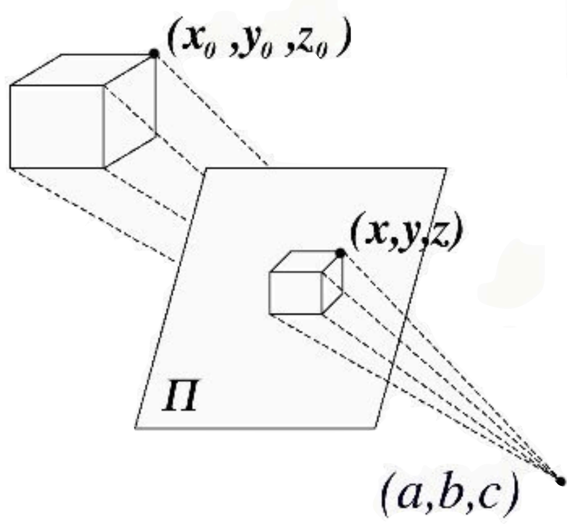


- In homogeneous coordinates:

Perspective Projection

- Points projected onto image plane by dividing them by their z component.

$$\bar{\boldsymbol{x}} = \mathcal{P}_z(\boldsymbol{p}) = \begin{bmatrix} x/z \\ y/z \\ 1 \end{bmatrix}$$



- In homogeneous coordinates:

$$\tilde{\boldsymbol{x}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tilde{\boldsymbol{p}},$$

Camera Intrinsics

$$\tilde{\boldsymbol{x}}_s = \boldsymbol{K} \left[\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{t} \end{array} \right] \bar{\boldsymbol{p}}_w = \boldsymbol{P} \bar{\boldsymbol{p}}_w$$

Camera Intrinsics

$$\tilde{\boldsymbol{x}}_s = \boldsymbol{K} \left[\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{t} \end{array} \right] \overline{\boldsymbol{p}}_w = \boldsymbol{P}\overline{\boldsymbol{p}}_w$$

3D world point

Camera Intrinsics

$$\tilde{\boldsymbol{x}}_s = \boldsymbol{K} \left[\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{t} \end{array} \right] \overline{\boldsymbol{p}}_w = \boldsymbol{P} \overline{\boldsymbol{p}}_w$$

2D image projection

3D world point

Camera Intrinsics

$$\tilde{\boldsymbol{x}}_s = \boldsymbol{K} \left[\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{t} \end{array} \right] \bar{\boldsymbol{p}}_w = \boldsymbol{P} \bar{\boldsymbol{p}}_w$$

2D image projection

3D world point

Projection matrix

Camera Intrinsics

$$\tilde{\boldsymbol{x}}_s = \boldsymbol{K} \left[\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{t} \end{array} \right] \bar{\boldsymbol{p}}_w = \boldsymbol{P} \bar{\boldsymbol{p}}_w$$

2D image projection

3D world point

Projection matrix

Extrinsic (rotation + translation) matrix

Camera Intrinsics

$$\tilde{\boldsymbol{x}}_s = \boldsymbol{K} \left[\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{t} \end{array} \right] \overline{\boldsymbol{p}}_w = \boldsymbol{P} \overline{\boldsymbol{p}}_w$$

2D image projection

Intrinsic (calibration) matrix

Extrinsic (rotation + translation) matrix

Projection matrix

3D world point

Camera Intrinsics

$$\tilde{\boldsymbol{x}}_s = \boldsymbol{K} \left[\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{t} \end{array} \right] \overline{\boldsymbol{p}}_w = \boldsymbol{P} \overline{\boldsymbol{p}}_w$$

2D image projection

3D world point

Intrinsic (calibration) matrix

Projection matrix

Extrinsic (rotation + translation) matrix

$$\boldsymbol{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Camera Intrinsics

$$\tilde{\boldsymbol{x}}_s = \boldsymbol{K} \left[\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{t} \end{array} \right] \overline{\boldsymbol{p}}_w = \boldsymbol{P} \overline{\boldsymbol{p}}_w$$

2D image projection

Intrinsic (calibration) matrix

Extrinsic (rotation + translation) matrix

Projection matrix

3D world point

$$\boldsymbol{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

f_x and f_y: encode focal length and pixel spacing, which may be slightly different in x and y dimensions.

Camera Intrinsics

$$\tilde{\boldsymbol{x}}_s = \boldsymbol{K} \left[\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{t} \end{array} \right] \overline{\boldsymbol{p}}_w = \boldsymbol{P} \overline{\boldsymbol{p}}_w$$

2D image projection

Intrinsic (calibration) matrix

Extrinsic (rotation + translation) matrix

Projection matrix

3D world point

$$\boldsymbol{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

f_x and f_y: encode focal length and pixel spacing, which may be slightly different in x and y dimensions.

c_x and c_y: encode principal point (intersection of optic axis with sensor plane) - usually very close to centre of image

Camera Intrinsics

$$\tilde{\boldsymbol{x}}_s = \boldsymbol{K} \left[\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{t} \end{array} \right] \bar{\boldsymbol{p}}_w = \boldsymbol{P} \bar{\boldsymbol{p}}_w$$

2D image projection

3D world point

Intrinsic (calibration) matrix

Projection matrix

Extrinsic (rotation + translation) matrix

$$\boldsymbol{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

f_x and f_y: encode focal length and pixel spacing, which may be slightly different in x and y dimensions.

c_x and c_y: encode principal point (intersection of optic axis with sensor plane) - usually very close to centre of image

s: encodes possible skew between sensor axes (usually close to 0).

Camera Intrinsics

$$\tilde{\boldsymbol{x}}_s = \boldsymbol{K}\left[\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{t} \end{array}\right]\overline{\boldsymbol{p}}_w = \boldsymbol{P}\overline{\boldsymbol{p}}_w$$

2D image projection

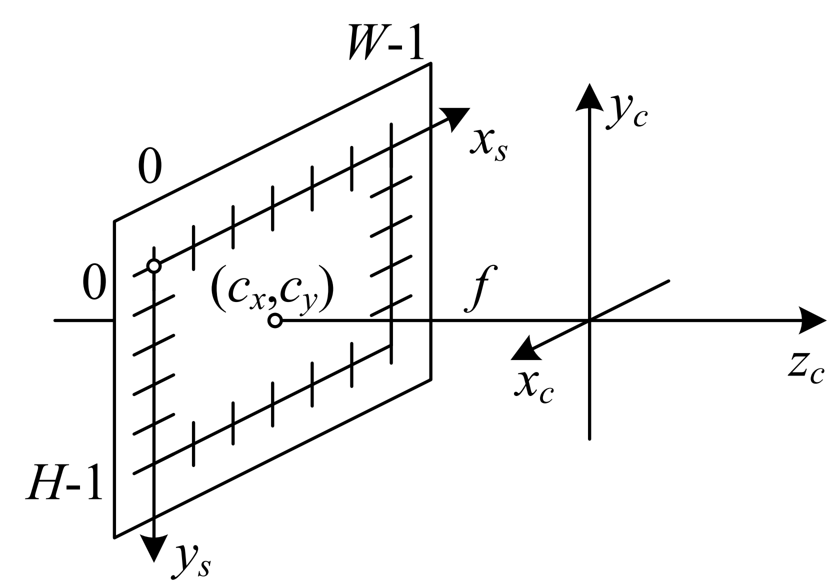
3D world point

Intrinsic (calibration) matrix

Projection matrix

Extrinsic (rotation + translation) matrix

$$\boldsymbol{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$



f_x and f_y: encode focal length and pixel spacing, which may be slightly different in x and y dimensions.

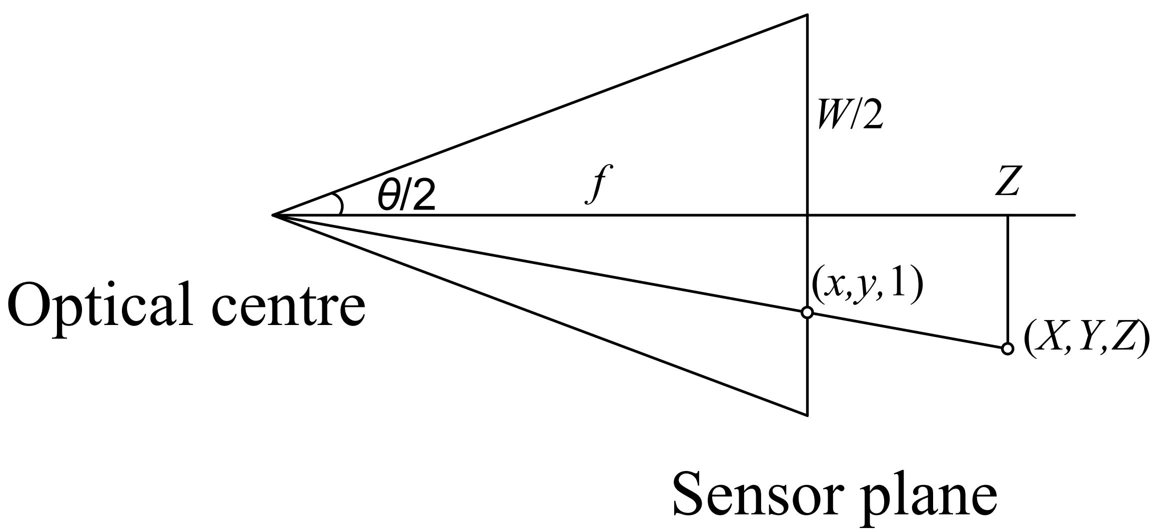
c_x and c_y: encode principal point (intersection of optic axis with sensor plane) - usually very close to centre of image

s: encodes possible skew between sensor axes (usually close to 0).

Focal Lengths

❖ Focal length can be measured either in pixels or in mm.

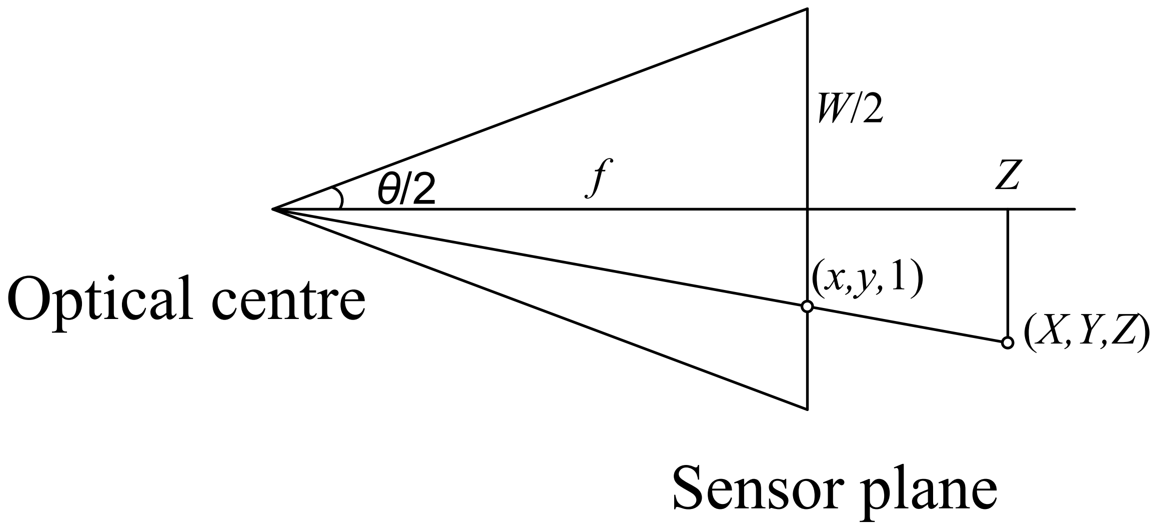
$$\tan\frac{\theta}{2} = \frac{W}{2f} \quad \text{or} \quad f = \frac{W}{2}\left[\tan\frac{\theta}{2}\right]^{-1}$$



Focal Lengths

- Focal length can be measured either in pixels or in mm.

$$\tan\frac{\theta}{2} = \frac{W}{2f} \quad \text{or} \quad f = \frac{W}{2}\left[\tan\frac{\theta}{2}\right]^{-1}$$



- Example: Consider the FLIR BlackFly S BFS-PGE-122S6C-C paired with a 10mm lens:
  - Resolution: 4096 x 3000 pixels
  - Sensor width: 1.1” = 27.94mm

Outline

- Geometric primitives
- 2D transformations
- 3D transformations
- 3D rotations
- 3D to 2D projections
- **Lens Distortions**

Lens Distortions

Lens Distortions

- In perspective projection, straight lines project to straight lines.

Lens Distortions

- In perspective projection, straight lines project to straight lines.
- This is not true in real cameras, due to lens distortions.

Lens Distortions

- In perspective projection, straight lines project to straight lines.
- This is not true in real cameras, due to lens distortions.
- Wide-angle lenses produce noticeable radial distortion

Lens Distortions

- In perspective projection, straight lines project to straight lines.
- This is not true in real cameras, due to lens distortions.
- Wide-angle lenses produce noticeable radial distortion
- Let (x_c, y_c) be image coordinates after perspective projection but before scaling by focal length and shifting but he optical centre.

Lens Distortions

- In perspective projection, straight lines project to straight lines.
- This is not true in real cameras, due to lens distortions.
- Wide-angle lenses produce noticeable radial distortion
- Let (x_c, y_c) be image coordinates after perspective projection but before scaling by focal length and shifting but he optical centre.
- Then without distortion, we should have

Lens Distortions

- In perspective projection, straight lines project to straight lines.
- This is not true in real cameras, due to lens distortions.
- Wide-angle lenses produce noticeable radial distortion
- Let (x_c, y_c) be image coordinates after perspective projection but before scaling by focal length and shifting but he optical centre.
- Then without distortion, we should have

$$
\begin{aligned}
x_c &= \frac{\boldsymbol{r}_x \cdot \boldsymbol{p} + t_x}{\boldsymbol{r}_z \cdot \boldsymbol{p} + t_z} \\
y_c &= \frac{\boldsymbol{r}_y \cdot \boldsymbol{p} + t_y}{\boldsymbol{r}_z \cdot \boldsymbol{p} + t_z}
\end{aligned}
$$

Radial Distortion

Radial Distortion

$$x_c = \frac{\boldsymbol{r}_x \cdot \boldsymbol{p} + t_x}{\boldsymbol{r}_z \cdot \boldsymbol{p} + t_z}$$

$$y_c = \frac{\boldsymbol{r}_y \cdot \boldsymbol{p} + t_y}{\boldsymbol{r}_z \cdot \boldsymbol{p} + t_z}$$

Radial Distortion

$$x_c = \frac{\boldsymbol{r}_x \cdot \boldsymbol{p} + t_x}{\boldsymbol{r}_z \cdot \boldsymbol{p} + t_z}$$

$$y_c = \frac{\boldsymbol{r}_y \cdot \boldsymbol{p} + t_y}{\boldsymbol{r}_z \cdot \boldsymbol{p} + t_z}$$

- In radial distortion, points are displaced radially by an amount that increases with their distance from the image centre

Radial Distortion

$$x_c = \frac{\boldsymbol{r}_x \cdot \boldsymbol{p} + t_x}{\boldsymbol{r}_z \cdot \boldsymbol{p} + t_z}$$

$$y_c = \frac{\boldsymbol{r}_y \cdot \boldsymbol{p} + t_y}{\boldsymbol{r}_z \cdot \boldsymbol{p} + t_z}$$

- In radial distortion, points are displaced radially by an amount that increases with their distance from the image centre
  - Barrel distortion: points are displaced away from the image centre

Radial Distortion

$$
\begin{aligned}
x_c &= \frac{\boldsymbol{r}_x \cdot \boldsymbol{p} + t_x}{\boldsymbol{r}_z \cdot \boldsymbol{p} + t_z} \\
y_c &= \frac{\boldsymbol{r}_y \cdot \boldsymbol{p} + t_y}{\boldsymbol{r}_z \cdot \boldsymbol{p} + t_z}
\end{aligned}
$$

- In radial distortion, points are displaced radially by an amount that increases with their distance from the image centre
  - Barrel distortion: points are displaced away from the image centre
  - Pincushion distortion: points are displaced towards the image centre

Radial Distortion

$$x_c = \frac{\boldsymbol{r}_x \cdot \boldsymbol{p} + t_x}{\boldsymbol{r}_z \cdot \boldsymbol{p} + t_z}$$

$$y_c = \frac{\boldsymbol{r}_y \cdot \boldsymbol{p} + t_y}{\boldsymbol{r}_z \cdot \boldsymbol{p} + t_z}$$

- In radial distortion, points are displaced radially by an amount that increases with their distance from the image centre
  - Barrel distortion: points are displaced away from the image centre
  - Pincushion distortion: points are displaced towards the image centre
  - Radial distortion can be modelled by a 4th-order perturbation on these coordinates:

Radial Distortion

$$x_c = \frac{\boldsymbol{r}_x \cdot \boldsymbol{p} + t_x}{\boldsymbol{r}_z \cdot \boldsymbol{p} + t_z}$$

$$y_c = \frac{\boldsymbol{r}_y \cdot \boldsymbol{p} + t_y}{\boldsymbol{r}_z \cdot \boldsymbol{p} + t_z}$$

- In radial distortion, points are displaced radially by an amount that increases with their distance from the image centre
  - Barrel distortion: points are displaced away from the image centre
  - Pincushion distortion: points are displaced towards the image centre
  - Radial distortion can be modelled by a 4th-order perturbation on these coordinates:

$$\hat{x}_c = x_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4)$$

$$\hat{y}_c = y_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4),$$

Outline

- Geometric primitives
- 2D transformations
- 3D transformations
- 3D rotations
- 3D to 2D projections
- Lens Distortions