Visual Recognition: Image Formation

Raquel Urtasun

TTI Chicago

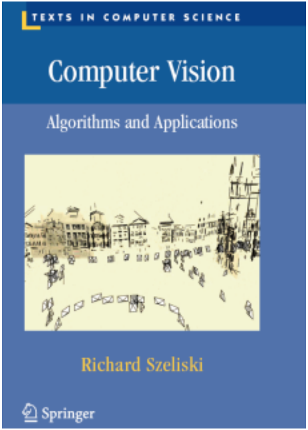
Jan 5, 2012

Today's lecture ...

- Fundamentals of image formation
- You should know about this already...
- ... so we will go fast on it
- Read about it if you are not familiar
- This will be almost all the geometry we will see in this class

Material

- Chapter 2 of Rich Szeliski book

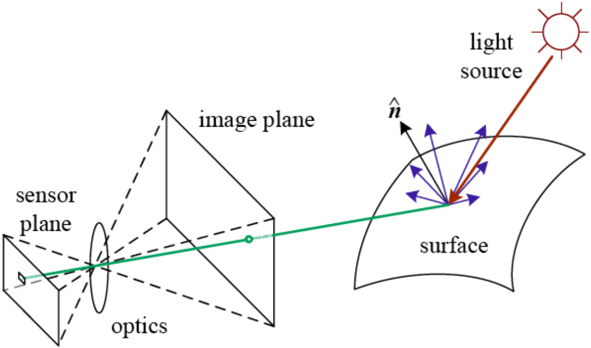


- Available online here

How is an image created?

The image formation process that produced a particular image depends on

- lighting conditions
- scene geometry,
- surface properties
- camera optics



[Source: R. Szeliski]

Geometric primitives and transformations

What are we going to see?

Basic 2D and 3D primitives:

- points
- lines
- planes

How 3D features are projected into 2D features

See [Hartley and Zisserman] book for more details

2D primitives: 2D points

- 2D points, e.g., pixel coordinate in an image, can be defined as $\mathbf{p} = (x, y) \in \Re^2$

$$\mathbf{p} = \left[\begin{array}{c} x \\ y \end{array} \right]$$

- 2D points can also be represented using homogeneous coordinates $\bar{\mathbf{p}} = (\bar{x}, \bar{y}, \bar{w}) \in \mathcal{P}^2$, with $\mathcal{P}^2 = \Re^3 - (0, 0, 0)$, the **perspective 2D space**.

2D primitives: 2D points

- 2D points, e.g., pixel coordinate in an image, can be defined as $\mathbf{p} = (x, y) \in \Re^2$

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

- 2D points can also be represented using homogeneous coordinates $\bar{\mathbf{p}} = (\bar{x}, \bar{y}, \bar{w}) \in \mathcal{P}^2$, with $\mathcal{P}^2 = \Re^3 - (0, 0, 0)$, the **perspective 2D space**.
- In **homogeneous coordinates** vectors that differ only by scale are equivalent.

2D primitives: 2D points

- 2D points, e.g., pixel coordinate in an image, can be defined as $\mathbf{p} = (x, y) \in \Re^2$

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

- 2D points can also be represented using homogeneous coordinates $\bar{\mathbf{p}} = (\bar{x}, \bar{y}, \bar{w}) \in \mathcal{P}^2$, with $\mathcal{P}^2 = \Re^3 - (0, 0, 0)$, the **perspective 2D space**.
- In **homogeneous coordinates** vectors that differ only by scale are equivalent.
- A homogeneous vector can be converted into an inhomogeneous one by dividing through by the last element

$$\tilde{\mathbf{p}} = (\tilde{x}; \tilde{y}; \tilde{w}) = \tilde{w}(x; y; 1) = \tilde{w}\bar{\mathbf{p}}$$

with $\bar{\mathbf{p}}$ an augmented vector $\bar{\mathbf{p}} = (x, y, 1)$.

2D primitives: 2D points

- 2D points, e.g., pixel coordinate in an image, can be defined as $\mathbf{p} = (x, y) \in \Re^2$

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

- 2D points can also be represented using homogeneous coordinates $\bar{\mathbf{p}} = (\bar{x}, \bar{y}, \bar{w}) \in \mathcal{P}^2$, with $\mathcal{P}^2 = \Re^3 - (0, 0, 0)$, the **perspective 2D space**.
- In **homogeneous coordinates** vectors that differ only by scale are equivalent.
- A homogeneous vector can be converted into an inhomogeneous one by dividing through by the last element

$$\tilde{\mathbf{p}} = (\tilde{x}; \tilde{y}; \tilde{w}) = \tilde{w}(x; y; 1) = \tilde{w}\bar{\mathbf{p}}$$

with $\bar{\mathbf{p}}$ an augmented vector $\bar{\mathbf{p}} = (x, y, 1)$.

- Homogeneous points whose last element is 0 are called **ideal points** or **points at infinity** and do not have an inhomogeneous representation.

2D primitives: 2D points

- 2D points, e.g., pixel coordinate in an image, can be defined as $\mathbf{p} = (x, y) \in \Re^2$

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

- 2D points can also be represented using homogeneous coordinates $\bar{\mathbf{p}} = (\bar{x}, \bar{y}, \bar{w}) \in \mathcal{P}^2$, with $\mathcal{P}^2 = \Re^3 - (0, 0, 0)$, the **perspective 2D space**.
- In **homogeneous coordinates** vectors that differ only by scale are equivalent.
- A homogeneous vector can be converted into an inhomogeneous one by dividing through by the last element

$$\tilde{\mathbf{p}} = (\tilde{x}; \tilde{y}; \tilde{w}) = \tilde{w}(x; y; 1) = \tilde{w}\bar{\mathbf{p}}$$

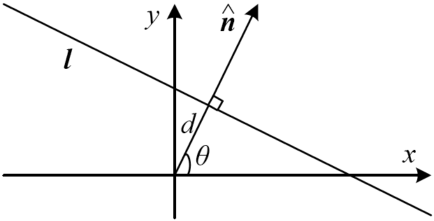
  with $\bar{\mathbf{p}}$ an augmented vector $\bar{\mathbf{p}} = (x, y, 1)$.
- Homogeneous points whose last element is 0 are called **ideal points** or **points at infinity** and do not have an inhomogeneous representation.

2D primitives: 2D lines

- 2D lines $\tilde{\mathbf{l}} = (a, b, c)$ can be represented in homogeneous coordinates

$$\bar{\mathbf{p}} \cdot \tilde{\mathbf{l}} = ax + by + c = 0$$

- If we normalize such that $\mathbf{l} = (n_x, n_y, d) = (\mathbf{n}, d)$ with $||\mathbf{n}|| = 1$, then $\mathbf{n}$ is the normal, perpendicular to the line and d is its distance to the origin.

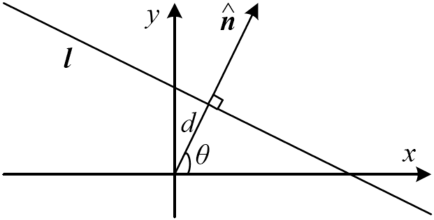


2D primitives: 2D lines

- 2D lines $\tilde{\mathbf{l}} = (a, b, c)$ can be represented in homogeneous coordinates

$$\bar{\mathbf{p}} \cdot \tilde{\mathbf{l}} = ax + by + c = 0$$

- If we normalize such that $\mathbf{l} = (n_x, n_y, d) = (\mathbf{n}, d)$ with $||\mathbf{n}|| = 1$, then $\mathbf{n}$ is the normal, perpendicular to the line and d is its distance to the origin.



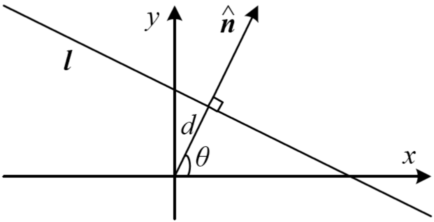
- Exception is the line at infinity, i.e., $\tilde{\mathbf{l}} = (0, 0, 1)$, which includes all (ideal) points at infinity.

2D primitives: 2D lines

- 2D lines $\tilde{\mathbf{l}} = (a, b, c)$ can be represented in homogeneous coordinates

$$\bar{\mathbf{p}} \cdot \tilde{\mathbf{l}} = ax + by + c = 0$$

- If we normalize such that $\mathbf{l} = (n_x, n_y, d) = (\mathbf{n}, d)$ with $||\mathbf{n}|| = 1$, then $\mathbf{n}$ is the normal, perpendicular to the line and d is its distance to the origin.



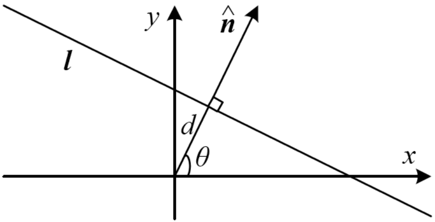
- Exception is the line at infinity, i.e., $\tilde{\mathbf{l}} = (0, 0, 1)$, which includes all (ideal) points at infinity.
- We can also express in **polar coordinates**, i.e., $\mathbf{l} = (d, \theta)$ with $\mathbf{n} = (\cos\theta, \sin\theta)$.

2D primitives: 2D lines

- 2D lines $\tilde{\mathbf{l}} = (a, b, c)$ can be represented in homogeneous coordinates

$$\bar{\mathbf{p}} \cdot \tilde{\mathbf{l}} = ax + by + c = 0$$

- If we normalize such that $\mathbf{l} = (n_x, n_y, d) = (\mathbf{n}, d)$ with $||\mathbf{n}|| = 1$, then $\mathbf{n}$ is the normal, perpendicular to the line and d is its distance to the origin.



- Exception is the line at infinity, i.e., $\tilde{\mathbf{l}} = (0, 0, 1)$, which includes all (ideal) points at infinity.
- We can also express in **polar coordinates**, i.e., $\mathbf{l} = (d, \theta)$ with $\mathbf{n} = (\cos\theta, \sin\theta)$.

2D lines and 2D points

When using homogeneous coordinates ...

- We can compute the intersection of two lines

$$\tilde{\mathbf{p}} = \tilde{\mathbf{l}}_1 \times \tilde{\mathbf{l}}_2$$

  with $\times$ the cross product.

- The cross product $\mathbf{a} \times \mathbf{b}$ is defined as a vector $\mathbf{c}$ that is perpendicular to both $\mathbf{a}$ and $\mathbf{b}$, with a direction given by the right-hand rule and a magnitude equal to the area of the parallelogram that the vectors span.

$$\mathbf{c} = |a| \cdot |b| \cdot \sin\theta \cdot \mathbf{n}$$

2D lines and 2D points

When using homogeneous coordinates ...

- We can compute the intersection of two lines

$$\tilde{\mathbf{p}} = \tilde{\mathbf{l}}_1 \times \tilde{\mathbf{l}}_2$$

  with $\times$ the cross product.

- The cross product $\mathbf{a} \times \mathbf{b}$ is defined as a vector $\mathbf{c}$ that is perpendicular to both $\mathbf{a}$ and $\mathbf{b}$, with a direction given by the right-hand rule and a magnitude equal to the area of the parallelogram that the vectors span.

$$\mathbf{c} = |a| \cdot |b| \cdot \sin\theta \cdot \mathbf{n}$$

- The line joining two points can be expressed as

$$\tilde{\mathbf{l}} = \tilde{\mathbf{p}}_1 \times \tilde{\mathbf{p}}_2$$

2D lines and 2D points

When using homogeneous coordinates ...

- We can compute the intersection of two lines

$$\tilde{\mathbf{p}} = \tilde{\mathbf{l}}_1 \times \tilde{\mathbf{l}}_2$$

  with $\times$ the cross product.

- The cross product $\mathbf{a} \times \mathbf{b}$ is defined as a vector $\mathbf{c}$ that is perpendicular to both $\mathbf{a}$ and $\mathbf{b}$, with a direction given by the right-hand rule and a magnitude equal to the area of the parallelogram that the vectors span.

$$\mathbf{c} = |a| \cdot |b| \cdot \sin\theta \cdot \mathbf{n}$$

- The line joining two points can be expressed as

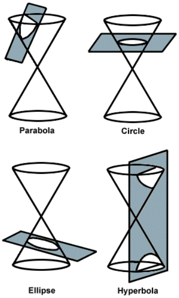
$$\tilde{\mathbf{l}} = \tilde{\mathbf{p}}_1 \times \tilde{\mathbf{p}}_2$$

2D primitives: 2D conics

- 2D conic is a curve obtained by intersecting a cone (i.e., a right circular conical surface) with a plane and can be written using a quadric equation

$$\bar{\mathbf{p}}\mathbf{Q}\bar{\mathbf{p}} = 0$$

- $\mathbf{Q}$ expresses the type of quadric.

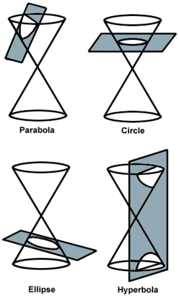


2D primitives: 2D conics

- 2D conic is a curve obtained by intersecting a cone (i.e., a right circular conical surface) with a plane and can be written using a quadric equation

$$\bar{\mathbf{p}}\mathbf{Q}\bar{\mathbf{p}} = 0$$

- $\mathbf{Q}$ expresses the type of quadric.



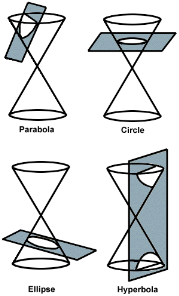
- Useful to represent human body or basic primitives and in geometry and camera calibration

2D primitives: 2D conics

- 2D conic is a curve obtained by intersecting a cone (i.e., a right circular conical surface) with a plane and can be written using a quadric equation

$$\bar{\mathbf{p}}\mathbf{Q}\bar{\mathbf{p}} = 0$$

- $\mathbf{Q}$ expresses the type of quadric.



- Useful to represent human body or basic primitives and in geometry and camera calibration

3D primitives: 3D points

- Point coordinates in 3D can be written using inhomogeneous coordinates $\mathbf{p} = (x, y, z) \in \Re^3$.
- And also in homogeneous coordinates $\tilde{\mathbf{p}} = (\tilde{x}, \tilde{y}, \tilde{z}, \tilde{w}) \in \mathcal{P}^3$.

3D primitives: 3D points

- Point coordinates in 3D can be written using inhomogeneous coordinates $\mathbf{p} = (x, y, z) \in \Re^3$.
- And also in homogeneous coordinates $\tilde{\mathbf{p}} = (\tilde{x}, \tilde{y}, \tilde{z}, \tilde{w}) \in \mathcal{P}^3$.
- It is useful to denote a 3D point using the augmented vector $\bar{\mathbf{p}} = (x, y, z, 1)$ with $\tilde{\mathbf{p}} = \tilde{w}\bar{\mathbf{x}}$.

3D primitives: 3D points

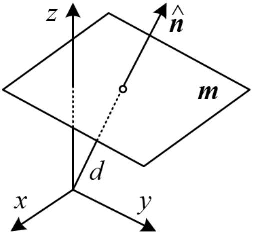
- Point coordinates in 3D can be written using inhomogeneous coordinates $\mathbf{p} = (x, y, z) \in \Re^3$.
- And also in homogeneous coordinates $\tilde{\mathbf{p}} = (\tilde{x}, \tilde{y}, \tilde{z}, \tilde{w}) \in \mathcal{P}^3$.
- It is useful to denote a 3D point using the augmented vector $\bar{\mathbf{p}} = (x, y, z, 1)$ with $\tilde{\mathbf{p}} = \tilde{w}\bar{\mathbf{x}}$.

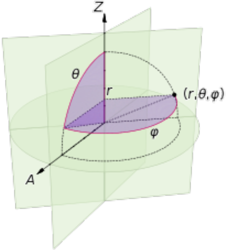
3D primitives: 3D plane

- In homogeneous coordinates, $\tilde{\mathbf{m}} = (a, b, c, d)$, the plane is

$$\bar{\mathbf{p}} \cdot \tilde{\mathbf{m}} = ax + by + cz + d = 0$$

- If we normalize such that $\mathbf{m} = (n_x, n_y, n_z, d) = (\mathbf{n}, d)$ with $||\mathbf{n}|| = 1$, then $\mathbf{n}$ is the normal, perpendicular to the plane, and d is its distance to the origin.



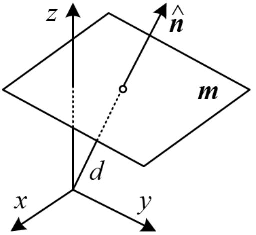


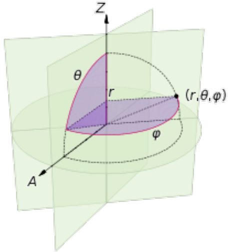
3D primitives: 3D plane

- In homogeneous coordinates, $\tilde{\mathbf{m}} = (a, b, c, d)$, the plane is

$$\bar{\mathbf{p}} \cdot \tilde{\mathbf{m}} = ax + by + cz + d = 0$$

- If we normalize such that $\mathbf{m} = (n_x, n_y, n_z, d) = (\mathbf{n}, d)$ with $||\mathbf{n}|| = 1$, then $\mathbf{n}$ is the normal, perpendicular to the plane, and d is its distance to the origin.

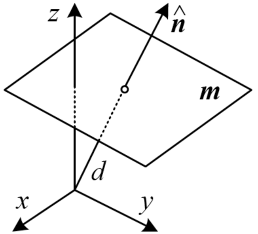


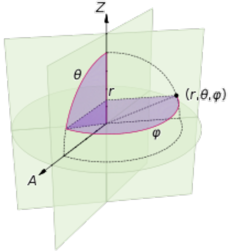


- The plane at infinity $\bar{\mathbf{m}} = (0, 0, 0, 1)$, containing all the points at infinity, cannot be normalized.

3D primitives: 3D plane

- In homogeneous coordinates, $\tilde{\mathbf{m}} = (a, b, c, d)$, the plane is
$$\bar{\mathbf{p}} \cdot \tilde{\mathbf{m}} = ax + by + cz + d = 0$$
- If we normalize such that $\mathbf{m} = (n_x, n_y, n_z, d) = (\mathbf{n}, d)$ with $||\mathbf{n}|| = 1$, then $\mathbf{n}$ is the normal, perpendicular to the plane, and d is its distance to the origin.





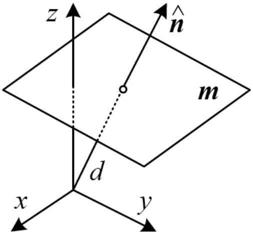
- The plane at infinity $\bar{\mathbf{m}} = (0, 0, 0, 1)$, containing all the points at infinity, cannot be normalized.
- We can express in spherical coordinates $\bar{\mathbf{n}} = (\cos\theta, \cos\phi, \sin\theta\cos\phi, \sin\phi)$

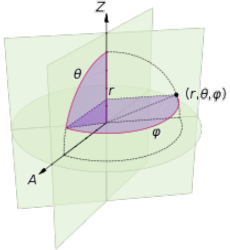
3D primitives: 3D plane

- In homogeneous coordinates, $\tilde{\mathbf{m}} = (a, b, c, d)$, the plane is

$$\bar{\mathbf{p}} \cdot \tilde{\mathbf{m}} = ax + by + cz + d = 0$$

- If we normalize such that $\mathbf{m} = (n_x, n_y, n_z, d) = (\mathbf{n}, d)$ with $||\mathbf{n}|| = 1$, then $\mathbf{n}$ is the normal, perpendicular to the plane, and d is its distance to the origin.





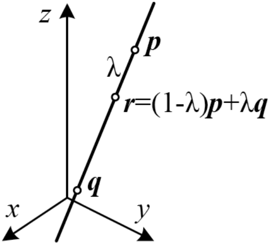
- The plane at infinity $\bar{\mathbf{m}} = (0, 0, 0, 1)$, containing all the points at infinity, cannot be normalized.
- We can express in spherical coordinates $\bar{\mathbf{n}} = (\cos\theta, \cos\phi, \sin\theta\cos\phi, \sin\phi)$

3D primitives: 3D line

- One possible representation is to use two points on the line, $(\mathbf{p}, \mathbf{q})$, then any point can be expressed as a linear combination of these two points

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

- If $0 \leq \lambda \leq 1$, then we get the line segment joining p and q.

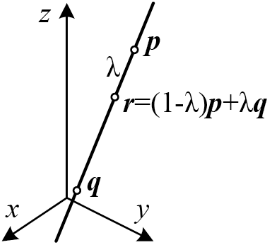


3D primitives: 3D line

- One possible representation is to use two points on the line, $(\mathbf{p}, \mathbf{q})$, then any point can be expressed as a linear combination of these two points

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

- If $0 \leq \lambda \leq 1$, then we get the line segment joining p and q.



- In homogeneous coordinates,

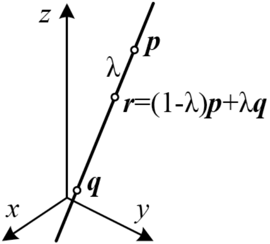
$$\tilde{\mathbf{r}} = \mu\tilde{\mathbf{p}} + \lambda\tilde{\mathbf{q}}$$

3D primitives: 3D line

- One possible representation is to use two points on the line, $(\mathbf{p}, \mathbf{q})$, then any point can be expressed as a linear combination of these two points

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

- If $0 \leq \lambda \leq 1$, then we get the line segment joining p and q.



- In homogeneous coordinates,

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

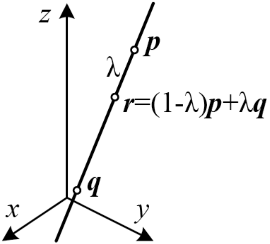
- When a point is at infinity, i.e., $\tilde{\mathbf{q}} = (d_x, d_y, d_z, 0) = (\mathbf{d}, 0)$, then $\mathbf{r} = \mathbf{p} + \lambda\mathbf{d}$

3D primitives: 3D line

- One possible representation is to use two points on the line, $(\mathbf{p}, \mathbf{q})$, then any point can be expressed as a linear combination of these two points

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

- If $0 \leq \lambda \leq 1$, then we get the line segment joining p and q.



- In homogeneous coordinates,

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

- When a point is at infinity, i.e., $\tilde{\mathbf{q}} = (d_x, d_y, d_z, 0) = (\mathbf{d}, 0)$, then $\mathbf{r} = \mathbf{p} + \lambda\mathbf{d}$

3D primitives: 3D conics

- Can be written using a quadric equation

$$\bar{\mathbf{p}}\mathbf{Q}\bar{\mathbf{p}} = 0$$

- **Q** expresses the type of quadric.
- Useful to represent human body in 3D or basic primitives

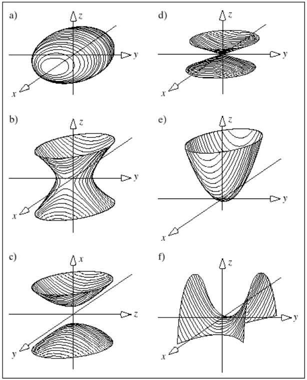
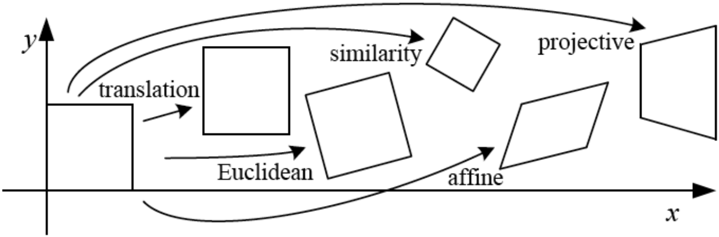


FIGURE 6.70 The six quadric surfaces: (a) Ellipsoid. (b) Hyperboloid of one sheet. (c) Hyperboloid of two sheets. (d) Elliptic cone. (e) Elliptic paraboloid. (f) Hyperbolic paraboloid.

2D Transformations



- **Translation:** can be written as $\mathbf{p}' = \mathbf{p} + \mathbf{t}$, or

$$\mathbf{p}' = \begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix} \bar{\mathbf{p}}$$

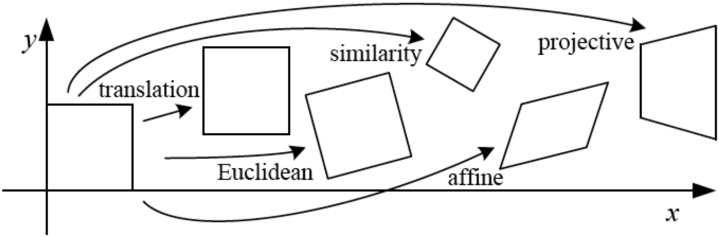
with $\mathbf{I}$ the 2×2 identity matrix, or

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

where 0 is the zero vector

- Which representation is more useful?

2D Transformations



- **Translation:** can be written as $\mathbf{p}' = \mathbf{p} + \mathbf{t}$, or

$$\mathbf{p}' = \begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix} \bar{\mathbf{p}}$$

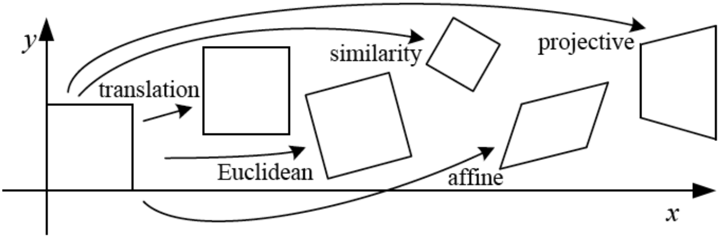
with $\mathbf{I}$ the 2×2 identity matrix, or

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

where 0 is the zero vector

- Which representation is more useful?

2D Transformations



- **2D Rigid Body Motion:** can be written as $\mathbf{p}' = \mathbf{R}\mathbf{p} + \mathbf{t}$, or

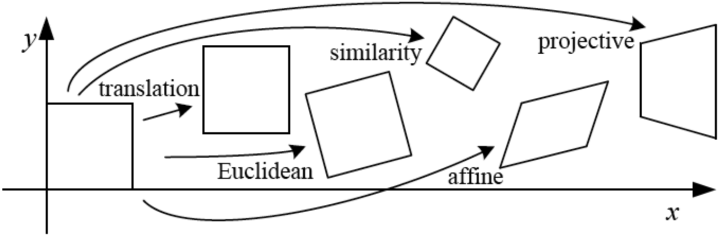
$$\mathbf{p}' = \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \bar{\mathbf{p}}$$

  with

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

  is an orthonormal rotation matrix $\mathbf{R}\mathbf{R}^T = \mathbf{I}$, and $|\mathbf{R}| = 1$
- Can also be written in homogeneous coordinates.
- Also called **2D Euclidean transformation**

2D Transformations

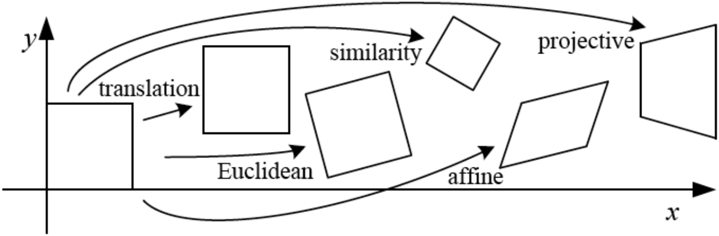


- **Similarity transform:** is $\mathbf{p}' = s\mathbf{R}\mathbf{p} + \mathbf{t}$, with s an scale factor.
- Also written as

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

  where we no longer require that $a^2 + b^2 = 1$.
- Preserves angles between lines.

2D Transformations

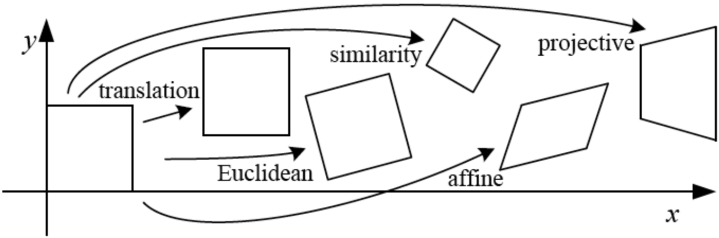


- **Affine** is $\mathbf{p}' = \mathbf{A}\bar{\mathbf{p}}$, with $\mathbf{A}$ an arbitrary 2×3 matrix, i.e.,

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

- Parallel lines remain parallel under affine transformations.

2D Transformations



- **Projective** operates on homogeneous coordinates

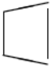
$$\bar{\mathbf{p}}' = \bar{\mathbf{H}}\bar{\mathbf{p}}$$

  with $\bar{\mathbf{H}}$ an arbitrary 3×3 matrix.
- Also known as **perspective transform** or **homography**.
- $\bar{\mathbf{H}}$ is homogeneous, i.e., it is only defined up to a scale.
- Two $\bar{\mathbf{H}}$ matrices that differ only by scale are equivalent.
- Perspective transformations preserve straight lines.

Hierarchy of Transformations

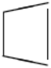
- A set of (potentially restricted) 3×3 matrices operating on 2D homogeneous coordinate vectors.
- They form a nested set of groups, i.e., they are closed under composition and have an inverse that is a member of the same group.
- Each (simpler) group is a subset of the more complex group below it.

Hierarchy of 2D Transformations

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	

- They can be applied in series
- Other transformations exist, e.g., stretch/squash

Hierarchy of 3D Transformations

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

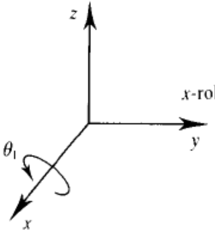
- Same as the 2D hierarchy.
- Check the book chapter for the exact definition.

Representing Rotations

- Representing 2D rotations in Euler angles is not a problem.
- However, it is a problem in 3D.
- Alternative representations: axis angles, quaternions.
- Let's see some of this representations.

Euler angles: definition

- The most popular parameterization of orientation space.
- A general rotation is described as a sequence of rotations about three mutually orthogonal coordinate axes fixed in the space.
- The rotations are applied to the space and not to the axis.

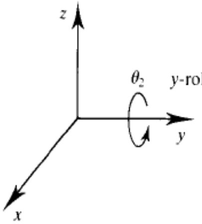


$$x\text{-roll}(\theta_1) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_1 & \sin\theta_1 & 0 \\ 0 & -\sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure: Principal rotation matrices: Rotation along the x-axis. [Souce: Watt 95]

Euler angles: definition

- The most popular parameterization of orientation space.
- A general rotation is described as a sequence of rotations about three mutually orthogonal coordinate axes fixed in the space.
- The rotations are applied to the space and not to the axis.

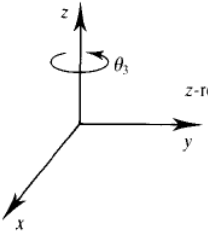


$$y\text{-roll}(\theta_2) = \begin{pmatrix} \cos\theta_2 & 0 & -\sin\theta_2 & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta_2 & 0 & \cos\theta_2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure: Principal rotation matrices: Rotation along the y-axis. [Souce: Watt 95]

Euler angles: definition

- The most popular parameterization of orientation space.
- A general rotation is described as a sequence of rotations about three mutually orthogonal coordinate axes fixed in the space.
- The rotations are applied to the space and not to the axis.



$$z\text{-roll}\,(\theta_3) = \begin{pmatrix} \cos\theta_3 & \sin\theta_3 & 0 & 0 \\ -\sin\theta_3 & \cos\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure: Principal rotation matrices: Rotation along the z-axis. [Souce: Watt 95]

Euler angles: composition

- General rotations can be done by composing rotations over these axis.
- For example, let's create a rotation matrix $\mathbf{R}(\theta_x, \theta_y, \theta_z)$ in terms of the joint angles $\theta_x, \theta_y, \theta_z$.

$$\mathbf{R}(\theta_x, \theta_y, \theta_z) = \mathbf{R}_x \cdot \mathbf{R}_y \cdot \mathbf{R}_z = \begin{pmatrix} c_y c_z & c_y s_z & -s_y & 0 \\ s_x s_y c_z - c_x s_z & s_x s_y s_z + c_x c_z & s_x c_y & 0 \\ c_x s_y c_z + s_x s_z & c_x s_y s_z - s_x c_z & c_x c_y & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

with $s_i = \sin(\theta_i)$, and $c_i = \cos(\theta_i)$.

Euler angles: composition

- General rotations can be done by composing rotations over these axis.
- For example, let's create a rotation matrix $\mathbf{R}(\theta_x, \theta_y, \theta_z)$ in terms of the joint angles $\theta_x, \theta_y, \theta_z$.

$$\mathbf{R}(\theta_x, \theta_y, \theta_z) = \mathbf{R}_x \cdot \mathbf{R}_y \cdot \mathbf{R}_z = \begin{pmatrix} c_y c_z & c_y s_z & -s_y & 0 \\ s_x s_y c_z - c_x s_z & s_x s_y s_z + c_x c_z & s_x c_y & 0 \\ c_x s_y c_z + s_x s_z & c_x s_y s_z - s_x c_z & c_x c_y & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

with $s_i = \sin(\theta_i)$, and $c_i = \cos(\theta_i)$.

- Matrix multiplication is not conmutative, the order is important

$$\mathbf{R}_x \cdot \mathbf{R}_y \cdot \mathbf{R}_z \neq \mathbf{R}_z \cdot \mathbf{R}_y \cdot \mathbf{R}_x$$

Euler angles: composition

- General rotations can be done by composing rotations over these axis.
- For example, let's create a rotation matrix $\mathbf{R}(\theta_x, \theta_y, \theta_z)$ in terms of the joint angles $\theta_x, \theta_y, \theta_z$.

$$\mathbf{R}(\theta_x, \theta_y, \theta_z) = \mathbf{R}_x \cdot \mathbf{R}_y \cdot \mathbf{R}_z = \begin{pmatrix} c_y c_z & c_y s_z & -s_y & 0 \\ s_x s_y c_z - c_x s_z & s_x s_y s_z + c_x c_z & s_x c_y & 0 \\ c_x s_y c_z + s_x s_z & c_x s_y s_z - s_x c_z & c_x c_y & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

with $s_i = \sin(\theta_i)$, and $c_i = \cos(\theta_i)$.

- Matrix multiplication is not conmutative, the order is important

$$\mathbf{R}_x \cdot \mathbf{R}_y \cdot \mathbf{R}_z \neq \mathbf{R}_z \cdot \mathbf{R}_y \cdot \mathbf{R}_x$$

- Rotations are assumed to be relative to fixed world axes, rather than local to the object.

Euler angles: composition

- General rotations can be done by composing rotations over these axis.
- For example, let's create a rotation matrix $\mathbf{R}(\theta_x, \theta_y, \theta_z)$ in terms of the joint angles $\theta_x, \theta_y, \theta_z$.

$$\mathbf{R}(\theta_x, \theta_y, \theta_z) = \mathbf{R}_x \cdot \mathbf{R}_y \cdot \mathbf{R}_z = \begin{pmatrix} c_y c_z & c_y s_z & -s_y & 0 \\ s_x s_y c_z - c_x s_z & s_x s_y s_z + c_x c_z & s_x c_y & 0 \\ c_x s_y c_z + s_x s_z & c_x s_y s_z - s_x c_z & c_x c_y & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

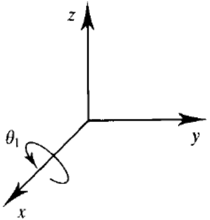
with $s_i = \sin(\theta_i)$, and $c_i = \cos(\theta_i)$.

- Matrix multiplication is not conmutative, the order is important

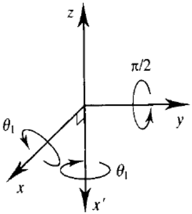
$$\mathbf{R}_x \cdot \mathbf{R}_y \cdot \mathbf{R}_z \neq \mathbf{R}_z \cdot \mathbf{R}_y \cdot \mathbf{R}_x$$

- Rotations are assumed to be relative to fixed world axes, rather than local to the object.

- **Gimbal lock**: This results when two axes effectively line up, resulting in a temporary loss of a degree of freedom.

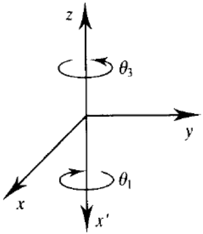


x-roll θ_1



x-roll θ_1 followed by y-roll $\pi/2$

x axis effectively gets rotated to x' axis



followed by z-roll θ_3

z-roll θ_3 same as x-roll $-\theta_1$

Euler angles: drawbacks I

- **Gimbal lock**: This results when two axes effectively line up, resulting in a temporary loss of a degree of freedom. This is a singularity in the parameterization. θ_1 and θ_3 become associated with the same DOF.

$$R(\theta_1, \frac{\pi}{2}, \theta_3) = \begin{pmatrix} 0 & 0 & -1 & 0 \\ \sin(\theta_1 - \theta_3) & \cos(\theta_1 - \theta_3) & 0 & 0 \\ \cos(\theta_1 - \theta_3) & \sin(\theta_1 - \theta_3) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

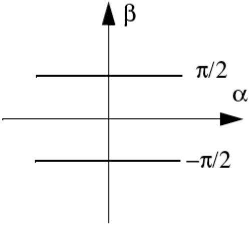


Figure: Singular locations of the Euler angles parametrization (at $\beta = \pm\pi/2$)

Euler angles: drawbacks II

- The parameterization is non-linear.
- The parameterization is modular $R(\theta) = R(\theta + 2\pi n)$, with $n \in Z$.

Euler angles: drawbacks II

- The parameterization is non-linear.
- The parameterization is modular $R(\theta) = R(\theta + 2\pi n)$, with $n \in Z$.
- The parameterization is not unique

$$\exists[\theta_4, \theta_5, \theta_6] \quad \text{such that} \quad R(\theta_1, \theta_2, \theta_3) = R(\theta_4, \theta_5, \theta_6)$$

with $\theta_i \neq \theta_{3+i}$ for all $i \in \{1, 2, 3\}$.

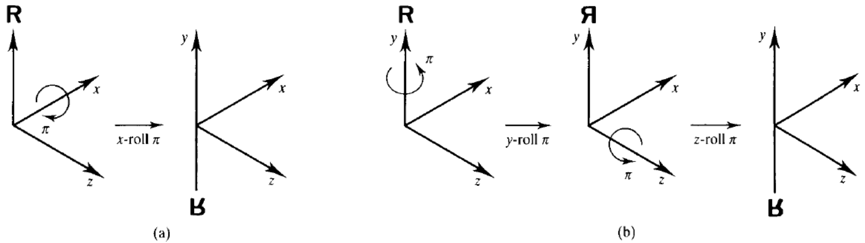


Figure: Example of two routes for the animation of the block letter R [Watt]

Euler angles: drawbacks II

- The parameterization is non-linear.
- The parameterization is modular $R(\theta) = R(\theta + 2\pi n)$, with $n \in Z$.
- The parameterization is not unique

$$\exists[\theta_4, \theta_5, \theta_6] \quad \text{such that} \quad R(\theta_1, \theta_2, \theta_3) = R(\theta_4, \theta_5, \theta_6)$$

with $\theta_i \neq \theta_{3+i}$ for all $i \in \{1, 2, 3\}$.

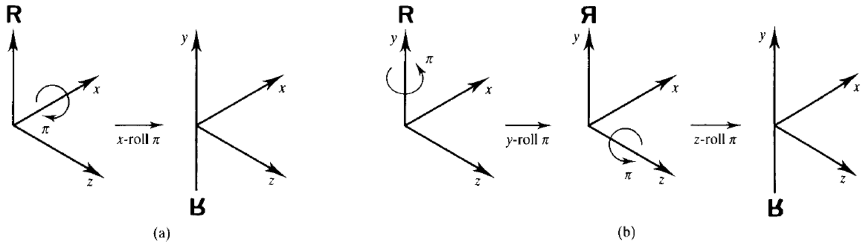


Figure: Example of two routes for the animation of the block letter R [Watt]

Axis Angles

- A rotation can be represented by a rotation axis $\mathbf{n}$, and an angle θ.
- Or equivalently by a 3D vector $\mathbf{w} = \theta\mathbf{n}$.

Axis Angles

- A rotation can be represented by a rotation axis $\mathbf{n}$, and an angle θ.
- Or equivalently by a 3D vector $\mathbf{w} = \theta\mathbf{n}$.
- It's a minimal representation.

Axis Angles

- A rotation can be represented by a rotation axis **n**, and an angle θ.
- Or equivalently by a 3D vector $\mathbf{w} = \theta \mathbf{n}$.
- It's a minimal representation.
- It has a singularity... when does it happen?

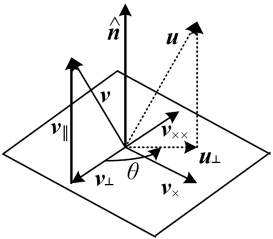


Figure: Rotation around an axis **n** by an angle θ

Axis Angles

- A rotation can be represented by a rotation axis **n**, and an angle θ.
- Or equivalently by a 3D vector $\mathbf{w} = \theta \mathbf{n}$.
- It's a minimal representation.
- It has a singularity... when does it happen?

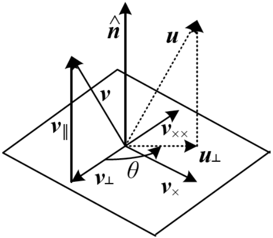


Figure: Rotation around an axis **n** by an angle θ

Quaternions

- Quaternions were invented by W.R.Hamilton in 1843.
- A quaternion has 4 components

$$\mathbf{q} = [q_w, q_x, q_y, q_z]^T$$

Quaternions

- Quaternions were invented by W.R.Hamilton in 1843.
- A quaternion has 4 components

$$\mathbf{q} = [q_w, q_x, q_y, q_z]^T$$

- They are extensions of complex numbers $a + \mathbf{i}b$ to a 3D imaginary space, **ijk**.

$$\mathbf{q} = q_w + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}$$

Quaternions

- Quaternions were invented by W.R.Hamilton in 1843.
- A quaternion has 4 components

$$\mathbf{q} = [q_w, q_x, q_y, q_z]^T$$

- They are extensions of complex numbers $a + \mathbf{i}b$ to a 3D imaginary space, **ijk**.

$$\mathbf{q} = q_w + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}$$

- With the additional properties

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{ijk} = -1$$

$$\mathbf{i} = \mathbf{jk} = -\mathbf{kj}, \quad \mathbf{j} = \mathbf{ki} = -\mathbf{ik}, \quad \mathbf{k} = \mathbf{ij} = -\mathbf{ji}$$

Quaternions

- Quaternions were invented by W.R.Hamilton in 1843.
- A quaternion has 4 components

$$\mathbf{q} = [q_w, q_x, q_y, q_z]^T$$

- They are extensions of complex numbers $a + \mathbf{i}b$ to a 3D imaginary space, $\mathbf{ijk}$.

$$\mathbf{q} = q_w + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}$$

- With the additional properties

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{ijk} = -1$$
$$\mathbf{i} = \mathbf{jk} = -\mathbf{kj}, \quad \mathbf{j} = \mathbf{ki} = -\mathbf{ik}, \quad \mathbf{k} = \mathbf{ij} = -\mathbf{ji}$$

- To represent rotations, only unit length quaternions are used

$$||\mathbf{q}||_2 = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2}$$

Quaternions

- Quaternions were invented by W.R.Hamilton in 1843.
- A quaternion has 4 components

$$\mathbf{q} = [q_w, q_x, q_y, q_z]^T$$

- They are extensions of complex numbers $a + \mathbf{i}b$ to a 3D imaginary space, $\mathbf{ijk}$.

$$\mathbf{q} = q_w + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}$$

- With the additional properties

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{ijk} = -1$$
$$\mathbf{i} = \mathbf{jk} = -\mathbf{kj}, \quad \mathbf{j} = \mathbf{ki} = -\mathbf{ik}, \quad \mathbf{k} = \mathbf{ij} = -\mathbf{ji}$$

- To represent rotations, only unit length quaternions are used

$$||\mathbf{q}||_2 = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2}$$

Quaternions

- Quaternions were invented by W.R.Hamilton in 1843.
- A quaternion has 4 components

$$\mathbf{q} = [q_w, q_x, q_y, q_z]^T$$

- They are extensions of complex numbers $a + \mathbf{i}b$ to a 3D imaginary space, **ijk**.

$$\mathbf{q} = q_w + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}$$

- With the additional properties

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{ijk} = -1$$
$$\mathbf{i} = \mathbf{jk} = -\mathbf{kj}, \quad \mathbf{j} = \mathbf{ki} = -\mathbf{ik}, \quad \mathbf{k} = \mathbf{ij} = -\mathbf{ji}$$

- To represent rotations, only unit length quaternions are used

$$||\mathbf{q}||_2 = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2}$$

Quaternions

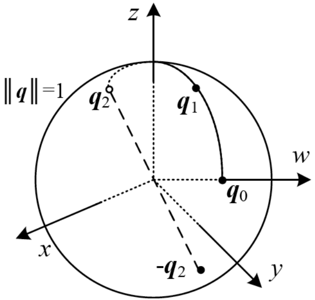


Figure: Unit quaternions live on the unit sphere $||\mathbf{q}|| = 1$. Smooth trajectory through 3 quaternions. The antipodal point to $\mathbf{q}_2$, namely $-\mathbf{q}_2$, represents the same rotation.

Quaternions

- Quaternions form a group whose underlying set is the four dimensional vector space R^4, with a multiplication operator $\circ$ that combines both the dot product and cross product of vectors.
- The identity rotation is encoded as $\mathbf{q} = [1, 0, 0, 0]^T$.

Quaternions

- Quaternions form a group whose underlying set is the four dimensional vector space R^4, with a multiplication operator $\circ$ that combines both the dot product and cross product of vectors.
- The identity rotation is encoded as $\mathbf{q} = [1, 0, 0, 0]^T$.
- The quaternion $\mathbf{q} = [q_w, q_x, q_y, q_z]^T$ encodes a rotation of $\theta = 2\cos^{-1}(q_w)$ along the unit axis $\hat{\mathbf{v}} = [qx, qy, qz]$.

Quaternions

- Quaternions form a group whose underlying set is the four dimensional vector space R^4, with a multiplication operator $\circ$ that combines both the dot product and cross product of vectors.
- The identity rotation is encoded as $\mathbf{q} = [1, 0, 0, 0]^T$.
- The quaternion $\mathbf{q} = [q_w, q_x, q_y, q_z]^T$ encodes a rotation of $\theta = 2\cos^{-1}(q_w)$ along the unit axis $\hat{\mathbf{v}} = [qx, qy, qz]$.
- Also a quaternion can represent a rotation by an angle θ around the $\hat{\mathbf{v}}$ axis as

$$\mathbf{q} = [\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\hat{\mathbf{v}}]$$

with $\hat{\mathbf{v}} = \frac{\mathbf{v}}{|\mathbf{v}|}$.

Quaternions

- Quaternions form a group whose underlying set is the four dimensional vector space R^4, with a multiplication operator $\circ$ that combines both the dot product and cross product of vectors.
- The identity rotation is encoded as $\mathbf{q} = [1, 0, 0, 0]^T$.
- The quaternion $\mathbf{q} = [q_w, q_x, q_y, q_z]^T$ encodes a rotation of $\theta = 2\cos^{-1}(q_w)$ along the unit axis $\hat{\mathbf{v}} = [qx, qy, qz]$.
- Also a quaternion can represent a rotation by an angle θ around the $\hat{\mathbf{v}}$ axis as

$$\mathbf{q} = [\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\hat{\mathbf{v}}]$$

  with $\hat{\mathbf{v}} = \frac{\mathbf{v}}{|\mathbf{v}|}$.
- If $\hat{\mathbf{v}}$ is unit length, then $\mathbf{q}$ will also be.

Quaternions

- Quaternions form a group whose underlying set is the four dimensional vector space R^4, with a multiplication operator $\circ$ that combines both the dot product and cross product of vectors.
- The identity rotation is encoded as $\mathbf{q} = [1, 0, 0, 0]^T$.
- The quaternion $\mathbf{q} = [q_w, q_x, q_y, q_z]^T$ encodes a rotation of $\theta = 2\cos^{-1}(q_w)$ along the unit axis $\hat{\mathbf{v}} = [qx, qy, qz]$.
- Also a quaternion can represent a rotation by an angle θ around the $\hat{\mathbf{v}}$ axis as

$$\mathbf{q} = [\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\hat{\mathbf{v}}]$$

  with $\hat{\mathbf{v}} = \frac{\mathbf{v}}{|\mathbf{v}|}$.
- If $\hat{\mathbf{v}}$ is unit length, then $\mathbf{q}$ will also be.
- Proof: simple exercise.

Quaternions

- Quaternions form a group whose underlying set is the four dimensional vector space R^4, with a multiplication operator $\circ$ that combines both the dot product and cross product of vectors.
- The identity rotation is encoded as $\mathbf{q} = [1, 0, 0, 0]^T$.
- The quaternion $\mathbf{q} = [q_w, q_x, q_y, q_z]^T$ encodes a rotation of $\theta = 2\cos^{-1}(q_w)$ along the unit axis $\hat{\mathbf{v}} = [qx, qy, qz]$.
- Also a quaternion can represent a rotation by an angle θ around the $\hat{\mathbf{v}}$ axis as

$$\mathbf{q} = [\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\hat{\mathbf{v}}]$$

  with $\hat{\mathbf{v}} = \frac{\mathbf{v}}{|\mathbf{v}|}$.
- If $\hat{\mathbf{v}}$ is unit length, then $\mathbf{q}$ will also be.
- Proof: simple exercise.

Quaternion to rotational matrix

- To convert a quaternion $\mathbf{q} = [q_w, q_x, q_y, q_z]$ to a rotational matrix simply compute

$$\begin{pmatrix} 1 - 2q_y^2 - 2q_z^2 & 2q_xq_y + 2q_wq_z & 2q_xq_z - 2q_wq_y & 0 \\ 2q_xq_y - 2q_wq_z & 1 - 2q_x^2 - 2q_z^2 & 2q_yq_z + 2q_wq_x & 0 \\ 2q_xq_z + 2q_wq_y & 2q_yq_z - 2q_wq_x & 1 - 2q_x^2 - 2q_y^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- A matrix can also easily be converted to quaternion. See references for the exact algorithm.

Interpretation of quaternions

- Any incremental movement along one of the orthogonal axes in curved space corresponds to an incremental rotation along an axis in real space (distances along the hypersphere correspond to angles in 3D space).
- Moving in some arbitrary direction corresponds to rotating around some arbitrary axis.

Interpretation of quaternions

- Any incremental movement along one of the orthogonal axes in curved space corresponds to an incremental rotation along an axis in real space (distances along the hypersphere correspond to angles in 3D space).
- Moving in some arbitrary direction corresponds to rotating around some arbitrary axis.
- If you move too far in one direction, you come back to where you started (corresponding to rotating 360 degrees around any one axis).

Interpretation of quaternions

- Any incremental movement along one of the orthogonal axes in curved space corresponds to an incremental rotation along an axis in real space (distances along the hypersphere correspond to angles in 3D space).
- Moving in some arbitrary direction corresponds to rotating around some arbitrary axis.
- If you move too far in one direction, you come back to where you started (corresponding to rotating 360 degrees around any one axis).
- A distance of x along the surface of the hypersphere corresponds to a rotation of angle 2x radians.

Interpretation of quaternions

- Any incremental movement along one of the orthogonal axes in curved space corresponds to an incremental rotation along an axis in real space (distances along the hypersphere correspond to angles in 3D space).
- Moving in some arbitrary direction corresponds to rotating around some arbitrary axis.
- If you move too far in one direction, you come back to where you started (corresponding to rotating 360 degrees around any one axis).
- A distance of x along the surface of the hypersphere corresponds to a rotation of angle 2x radians.
- This means that moving along a 90 degree arc on the hypersphere corresponds to rotating an object by 180 degrees.

Interpretation of quaternions

- Any incremental movement along one of the orthogonal axes in curved space corresponds to an incremental rotation along an axis in real space (distances along the hypersphere correspond to angles in 3D space).
- Moving in some arbitrary direction corresponds to rotating around some arbitrary axis.
- If you move too far in one direction, you come back to where you started (corresponding to rotating 360 degrees around any one axis).
- A distance of x along the surface of the hypersphere corresponds to a rotation of angle 2x radians.
- This means that moving along a 90 degree arc on the hypersphere corresponds to rotating an object by 180 degrees.
- Traveling 180 degrees corresponds to a 360 degree rotation, thus getting you back to where you started.

Interpretation of quaternions

- Any incremental movement along one of the orthogonal axes in curved space corresponds to an incremental rotation along an axis in real space (distances along the hypersphere correspond to angles in 3D space).
- Moving in some arbitrary direction corresponds to rotating around some arbitrary axis.
- If you move too far in one direction, you come back to where you started (corresponding to rotating 360 degrees around any one axis).
- A distance of x along the surface of the hypersphere corresponds to a rotation of angle 2x radians.
- This means that moving along a 90 degree arc on the hypersphere corresponds to rotating an object by 180 degrees.
- Traveling 180 degrees corresponds to a 360 degree rotation, thus getting you back to where you started.
- This implies that q and -q correspond to the same orientation.

Interpretation of quaternions

- Any incremental movement along one of the orthogonal axes in curved space corresponds to an incremental rotation along an axis in real space (distances along the hypersphere correspond to angles in 3D space).
- Moving in some arbitrary direction corresponds to rotating around some arbitrary axis.
- If you move too far in one direction, you come back to where you started (corresponding to rotating 360 degrees around any one axis).
- A distance of x along the surface of the hypersphere corresponds to a rotation of angle 2x radians.
- This means that moving along a 90 degree arc on the hypersphere corresponds to rotating an object by 180 degrees.
- Traveling 180 degrees corresponds to a 360 degree rotation, thus getting you back to where you started.
- This implies that q and -q correspond to the same orientation.

Quaternion operations: dot product

- The **dot product** of quaternions is simple their vector dot product

$$\mathbf{p} \cdot \mathbf{q} = \mathbf{p}_w \mathbf{q}_w + \mathbf{p}_x \mathbf{q}_x + \mathbf{p}_y \mathbf{q}_y + \mathbf{p}_z \mathbf{q}_z = |\mathbf{p}||\mathbf{q}| \cos \phi$$

- The angle between two quaternions in 4D space is half the angle one would need to rotate from one orientation to the other in 3D space.

Quaternion operations: multiplication

- **Multiplication** on quaternions can be done by expanding them into complex numbers

$$\mathbf{pq} = < s \cdot t - \mathbf{v} \cdot \mathbf{w}^T, \quad s\mathbf{w} + t\mathbf{v} + \mathbf{v} \times \mathbf{w} >$$

  where $\mathbf{p} = [s, \mathbf{v}]^T$, and $\mathbf{q} = [t, \mathbf{w}]$.

- If $\mathbf{p}$ represents a rotation and $\mathbf{q}$ represents a rotation, then $\mathbf{pq}$ represents $\mathbf{p}$ rotated by $\mathbf{q}$.

Quaternion operations: multiplication

- **Multiplication** on quaternions can be done by expanding them into complex numbers

$$\mathbf{pq} =< s \cdot t - \mathbf{v} \cdot \mathbf{w}^T, \quad s\mathbf{w} + t\mathbf{v} + \mathbf{v} \times \mathbf{w} >$$

  where $\mathbf{p} = [s, \mathbf{v}]^T$, and $\mathbf{q} = [t, \mathbf{w}]$.
- If **p** represents a rotation and **q** represents a rotation, then **pq** represents **p** rotated by **q**.
- Note that two unit quaternions multiplied together will result in another unit quaternion.

Quaternion operations: multiplication

- **Multiplication** on quaternions can be done by expanding them into complex numbers

$$\mathbf{pq} = < s \cdot t - \mathbf{v} \cdot \mathbf{w}^T, \quad s\mathbf{w} + t\mathbf{v} + \mathbf{v} \times \mathbf{w} >$$

  where $\mathbf{p} = [s, \mathbf{v}]^T$, and $\mathbf{q} = [t, \mathbf{w}]$.
- If $\mathbf{p}$ represents a rotation and $\mathbf{q}$ represents a rotation, then $\mathbf{pq}$ represents $\mathbf{p}$ rotated by $\mathbf{q}$.
- Note that two unit quaternions multiplied together will result in another unit quaternion.
- Quaternions extend the planar rotations of complex numbers to 3D rotations in space.

Quaternion operations: multiplication

- **Multiplication** on quaternions can be done by expanding them into complex numbers

$$\mathbf{pq} =< s \cdot t - \mathbf{v} \cdot \mathbf{w}^T, \quad s\mathbf{w} + t\mathbf{v} + \mathbf{v} \times \mathbf{w} >$$

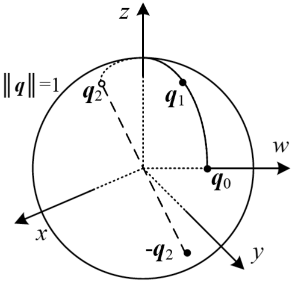
  where $\mathbf{p} = [s, \mathbf{v}]^T$, and $\mathbf{q} = [t, \mathbf{w}]$.
- If $\mathbf{p}$ represents a rotation and $\mathbf{q}$ represents a rotation, then $\mathbf{pq}$ represents $\mathbf{p}$ rotated by $\mathbf{q}$.
- Note that two unit quaternions multiplied together will result in another unit quaternion.
- Quaternions extend the planar rotations of complex numbers to 3D rotations in space.

Quaternion operations: others

- Inverse of a quaternion $\mathbf{q} = [s, \mathbf{v}]^T$

$$\mathbf{q}^{-1} = \frac{1}{|\mathbf{q}|^2}[s, -\mathbf{v}]^T$$

- Any multiple of a quaternion gives the same rotation because the effects of the magnitude are divided out.
- Very good for interpolation, Slerp.



Redundancy of the parameterizations

- The parameterizations that we have seen:
  - Rotational matrix: 9 DOF. It has 6 extra DOF.
  - Axis angles: 3 DOF for the scaled version and 4 DOF for the non-scaled. The latter has one extra DOF.
  - Quaternions: 4 DOF, 1 extra DOF.
- From the Euler theorem we know that an arbitrary rotation can be described with only 3 DOF, so those parameters extra are redundant.

Redundancy of the parameterizations

- The parameterizations that we have seen:
  - Rotational matrix: 9 DOF. It has 6 extra DOF.
  - Axis angles: 3 DOF for the scaled version and 4 DOF for the non-scaled. The latter has one extra DOF.
  - Quaternions: 4 DOF, 1 extra DOF.
- From the Euler theorem we know that an arbitrary rotation can be described with only 3 DOF, so those parameters extra are redundant.
- For rotational matrix, we can impose additional constraints if the matrix represent a rigid transform.

$$|a| = |b| = |c| = 1$$

$$a = b \times c, \quad b = c \times a, \quad c = a \times b$$

Redundancy of the parameterizations

- The parameterizations that we have seen:
  - Rotational matrix: 9 DOF. It has 6 extra DOF.
  - Axis angles: 3 DOF for the scaled version and 4 DOF for the non-scaled. The latter has one extra DOF.
  - Quaternions: 4 DOF, 1 extra DOF.
- From the Euler theorem we know that an arbitrary rotation can be described with only 3 DOF, so those parameters extra are redundant.
- For rotational matrix, we can impose additional constraints if the matrix represent a rigid transform.

$$|a| = |b| = |c| = 1$$

$$a = b \times c, \quad b = c \times a, \quad c = a \times b$$

Which rotation representation is better?

- The axis/angle representation is minimal, and hence does not require any additional constraints.
- If the angle is expressed in degrees, it is easier to understand the pose, and easier to express exact rotations and derivatives are easy to compute.

Which rotation representation is better?

- The axis/angle representation is minimal, and hence does not require any additional constraints.
- If the angle is expressed in degrees, it is easier to understand the pose, and easier to express exact rotations and derivatives are easy to compute.
- Quaternions do not have discontinuities. It is also easier to interpolate between rotations and to chain rigid transformations.

Which rotation representation is better?

- The axis/angle representation is minimal, and hence does not require any additional constraints.
- If the angle is expressed in degrees, it is easier to understand the pose, and easier to express exact rotations and derivatives are easy to compute.
- Quaternions do not have discontinuities. It is also easier to interpolate between rotations and to chain rigid transformations.

- How are 3D primitives projected onto the image plane?
- We can do this using a linear 3D to 2D projection matrix
- Different types. The most commonly used:
  - Orthography
  - Perspective

Orthographic Projection

- An orthographic projection simply drops the z component of $\mathbf{p} = (x, y, z)$ to obtain the 2D point $\mathbf{q}$

$$\mathbf{q} = \left[\begin{array}{c|c} \mathbf{I} & 0 \end{array} \right] \mathbf{p}$$

- Using homogeneous coordinates

$$\bar{\mathbf{q}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \bar{\mathbf{p}}$$

Orthographic Projection

- An orthographic projection simply drops the z component of $\mathbf{p} = (x, y, z)$ to obtain the 2D point $\mathbf{q}$

$$\mathbf{q} = \left[\begin{array}{c|c} \mathbf{I} & 0 \end{array}\right] \mathbf{p}$$

- Using homogeneous coordinates

$$\bar{\mathbf{q}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \bar{\mathbf{p}}$$

- Is an approximate model for long focal length lenses and objects whose depth is shallow relative to their distance to the camera.

Orthographic Projection

- An orthographic projection simply drops the z component of $\mathbf{p} = (x, y, z)$ to obtain the 2D point $\mathbf{q}$

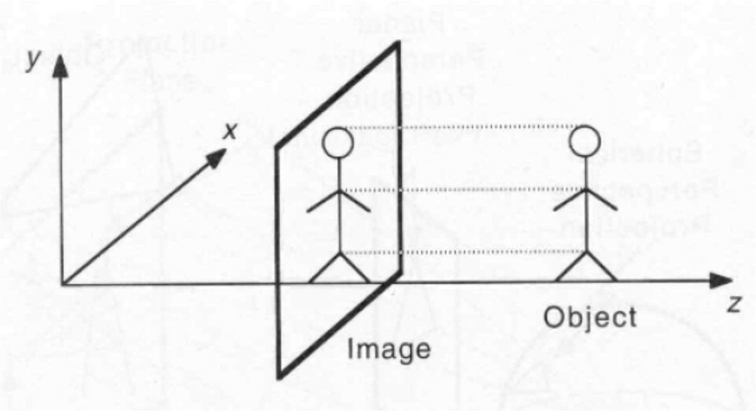
$$\mathbf{q} = \left[\begin{array}{c|c} \mathbf{I} & 0 \end{array} \right] \mathbf{p}$$

- Using homogeneous coordinates

$$\bar{\mathbf{q}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \bar{\mathbf{p}}$$

- Is an approximate model for long focal length lenses and objects whose depth is shallow relative to their distance to the camera.

Orthographic Projection



Scaled Orthographic Projection

- Scaled orthography is actually more commonly used to fit to the image

$$\mathbf{q} = \left[\begin{array}{c|c} s\mathbf{I} & 0 \end{array} \right] \mathbf{p}$$

- This model is equivalent to first projecting the world points onto a local fronto-parallel image plane and then scaling this image using regular perspective projection.

Scaled Orthographic Projection

- Scaled orthography is actually more commonly used to fit to the image

$$\mathbf{q} = \left[\begin{array}{c|c} s\mathbf{I} & 0 \end{array} \right] \mathbf{p}$$

- This model is equivalent to first projecting the world points onto a local fronto-parallel image plane and then scaling this image using regular perspective projection.
- The scaling can be the same for all parts of the scene or it can be different for objects that are being modeled independently.

Scaled Orthographic Projection

- Scaled orthography is actually more commonly used to fit to the image

$$\mathbf{q} = \begin{bmatrix} s\mathbf{I} & | & 0 \end{bmatrix} \mathbf{p}$$

- This model is equivalent to first projecting the world points onto a local fronto-parallel image plane and then scaling this image using regular perspective projection.
- The scaling can be the same for all parts of the scene or it can be different for objects that are being modeled independently.
- It is a popular model for reconstructing the 3D shape of objects far away from the camera,

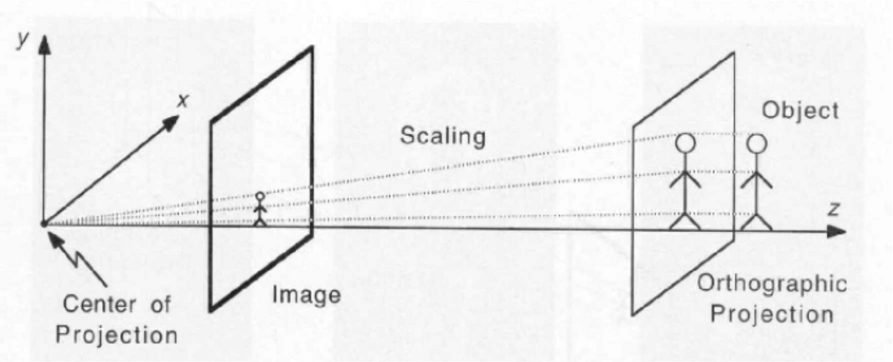
Scaled Orthographic Projection

- Scaled orthography is actually more commonly used to fit to the image

$$\mathbf{q} = \left[\begin{array}{c|c} s\mathbf{I} & 0 \end{array}\right] \mathbf{p}$$

- This model is equivalent to first projecting the world points onto a local fronto-parallel image plane and then scaling this image using regular perspective projection.
- The scaling can be the same for all parts of the scene or it can be different for objects that are being modeled independently.
- It is a popular model for reconstructing the 3D shape of objects far away from the camera,

Scaled Orthographic Projection



Perspective Projection

- Points are projected onto the image plane by dividing them by their z coordinate, i.e.,

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

- In homogeneous coordinates, the projection has a simple linear form,

$$\bar{\mathbf{q}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \bar{\mathbf{p}}$$

Perspective Projection

- Points are projected onto the image plane by dividing them by their z coordinate, i.e.,

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

- In homogeneous coordinates, the projection has a simple linear form,

$$\bar{\mathbf{q}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \bar{\mathbf{p}}$$

- We have dropped the w component, thus it is not possible to recover the distance of the 3D point from the image.

Perspective Projection

- Points are projected onto the image plane by dividing them by their z coordinate, i.e.,

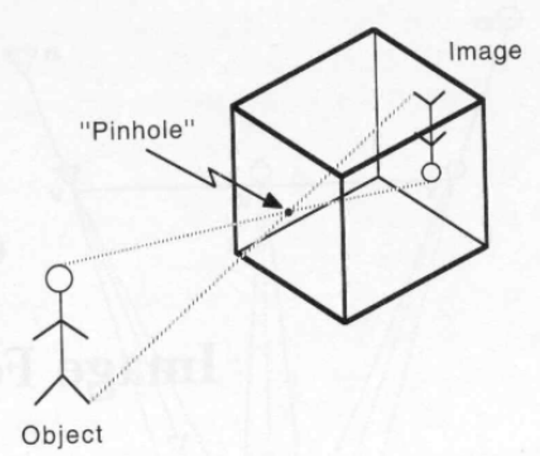
$$\mathbf{q} = \mathcal{P}_z(\mathbf{p}) = \begin{bmatrix} x/z \\ y/z \\ 1 \end{bmatrix}$$

- In homogeneous coordinates, the projection has a simple linear form,

$$\bar{\mathbf{q}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \bar{\mathbf{p}}$$

- We have dropped the w component, thus it is not possible to recover the distance of the 3D point from the image.

Perspective Projection



Perspective Projection

[Source: S. Seitz]

Camera Intrinsics

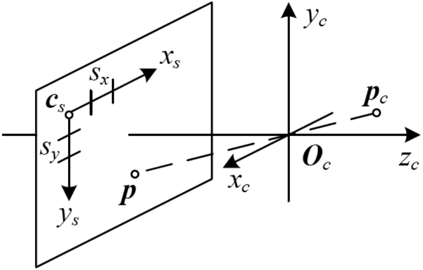
- Once we projected the 3D point through an ideal pinhole using a projection matrix, we must still transform the result according to the pixel sensor spacing and the relative position of the sensor plane to the origin.
- Image sensors return pixel values indexed by integer pixel coord (x_s, y_s).

Camera Intrinsics

- Once we projected the 3D point through an ideal pinhole using a projection matrix, we must still transform the result according to the pixel sensor spacing and the relative position of the sensor plane to the origin.
- Image sensors return pixel values indexed by integer pixel coord (x_s, y_s).
- To map pixels to 3D coordinates, we first scale the (x_s, y_s) by the pixel spacings (s_x, s_y).

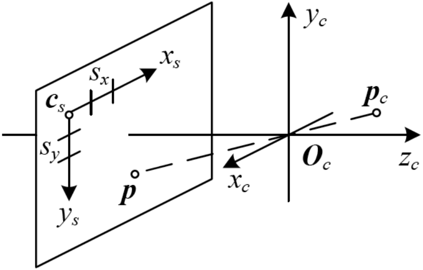
Camera Intrinsics

- Once we projected the 3D point through an ideal pinhole using a projection matrix, we must still transform the result according to the pixel sensor spacing and the relative position of the sensor plane to the origin.
- Image sensors return pixel values indexed by integer pixel coord (x_s, y_s).
- To map pixels to 3D coordinates, we first scale the (x_s, y_s) by the pixel spacings (s_x, s_y).
- We then describe the orientation of the sensor relative to the camera projection center $\mathcal{O}_c$ with an origin $\mathbf{c}_s$ and a 3D rotation $\mathbf{R}_s$.



Camera Intrinsics

- Once we projected the 3D point through an ideal pinhole using a projection matrix, we must still transform the result according to the pixel sensor spacing and the relative position of the sensor plane to the origin.
- Image sensors return pixel values indexed by integer pixel coord (x_s, y_s).
- To map pixels to 3D coordinates, we first scale the (x_s, y_s) by the pixel spacings (s_x, s_y).
- We then describe the orientation of the sensor relative to the camera projection center $\mathcal{O}_c$ with an origin $\mathbf{c}_s$ and a 3D rotation $\mathbf{R}_s$.



Camera Intrinsics

- We then describe the orientation of the sensor relative to the camera projection center $\mathcal{O}_c$ with an origin $\mathbf{c}_s$ and a 3D rotation $\mathbf{R}_s$.
- The combined 2D to 3D projection can then be written as

$$\bar{\mathbf{p}} = \begin{bmatrix} \mathbf{R}_s & | & \mathbf{c}_s \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix} = \mathbf{M}_s \bar{\mathbf{p}}_s$$

- The matrix $\mathbf{M}_s$ has 8 unknowns: 3 rotation, 3 translation, 2 scaling.

Camera Intrinsics

- We then describe the orientation of the sensor relative to the camera projection center $\mathcal{O}_c$ with an origin $\mathbf{c}_s$ and a 3D rotation $\mathbf{R}_s$.
- The combined 2D to 3D projection can then be written as

$$\bar{\mathbf{p}} = \begin{bmatrix} \mathbf{R}_s & | & \mathbf{c}_s \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix} = \mathbf{M}_s \bar{\mathbf{p}}_s$$

- The matrix $\mathbf{M}_s$ has 8 unknowns: 3 rotation, 3 translation, 2 scaling.
- We have ignored the possible skewing between the axes.

Camera Intrinsics

- We then describe the orientation of the sensor relative to the camera projection center $\mathcal{O}_c$ with an origin $\mathbf{c}_s$ and a 3D rotation $\mathbf{R}_s$.
- The combined 2D to 3D projection can then be written as

$$\bar{\mathbf{p}} = \left[\begin{array}{c|c} \mathbf{R}_s & \mathbf{c}_s \end{array} \right] \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix} = \mathbf{M}_s \bar{\mathbf{p}}_s$$

- The matrix $\mathbf{M}_s$ has 8 unknowns: 3 rotation, 3 translation, 2 scaling.
- We have ignored the possible skewing between the axes.
- In general only 7 dof, since the distance of the sensor from the origin cannot be teased apart from the sensor spacing

Camera Intrinsics

- We then describe the orientation of the sensor relative to the camera projection center $\mathcal{O}_c$ with an origin $\mathbf{c}_s$ and a 3D rotation $\mathbf{R}_s$.
- The combined 2D to 3D projection can then be written as

$$\bar{\mathbf{p}} = \left[\begin{array}{c|c} \mathbf{R}_s & \mathbf{c}_s \end{array} \right] \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix} = \mathbf{M}_s \bar{\mathbf{p}}_s$$

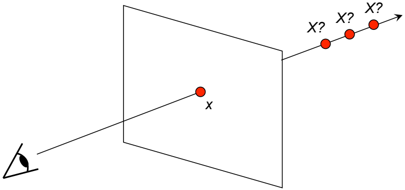
- The matrix $\mathbf{M}_s$ has 8 unknowns: 3 rotation, 3 translation, 2 scaling.
- We have ignored the possible skewing between the axes.
- In general only 7 dof, since the distance of the sensor from the origin cannot be teased apart from the sensor spacing

Camera Calibration

- We know some 3D points and we want to obtain the camera intrinsics and extrinsics.
- The relationship between the 3D pixel center $\mathbf{p}$ and the 3D camera-centered point $\mathbf{p}_c$ is given by an unknown scaling s, i.e., $\mathbf{p} = s\mathbf{p}_c$.

Camera Calibration

- We know some 3D points and we want to obtain the camera intrinsics and extrinsics.
- The relationship between the 3D pixel center $\mathbf{p}$ and the 3D camera-centered point $\mathbf{p}_c$ is given by an unknown scaling s, i.e., $\mathbf{p} = s\mathbf{p}_c$.



- We can therefore write the complete projection as

$$\bar{\mathbf{p}}_s = \alpha \mathbf{M}_s^{-1} \mathbf{p}_c = \mathbf{K}\mathbf{p}_c$$

with $\mathbf{K}$ the **calibration matrix** that describes the camera intrinsic parameters.

Camera Calibration

- We know some 3D points and we want to obtain the camera intrinsics and extrinsics.
- The relationship between the 3D pixel center $\mathbf{p}$ and the 3D camera-centered point $\mathbf{p}_c$ is given by an unknown scaling s, i.e., $\mathbf{p} = s\mathbf{p}_c$.
- We can therefore write the complete projection as

$$\bar{\mathbf{p}}_s = \alpha \mathbf{M}_s^{-1} \mathbf{p}_c = \mathbf{K}\mathbf{p}_c$$

  with $\mathbf{K}$ the **calibration matrix** that describes the camera intrinsic parameters.
- Combining with the external parameters we have

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

  with $\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$ the **camera matrix**.

Camera Calibration

- We know some 3D points and we want to obtain the camera intrinsics and extrinsics.
- The relationship between the 3D pixel center $\mathbf{p}$ and the 3D camera-centered point $\mathbf{p}_c$ is given by an unknown scaling s, i.e., $\mathbf{p} = s\mathbf{p}_c$.
- We can therefore write the complete projection as

$$\bar{\mathbf{p}}_s = \alpha \mathbf{M}_s^{-1} \mathbf{p}_c = \mathbf{K}\mathbf{p}_c$$

  with $\mathbf{K}$ the **calibration matrix** that describes the camera intrinsic parameters.
- Combining with the external parameters we have

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

  with $\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$ the **camera matrix**.
- Problem of identification, so $\mathbf{K}$ is assumed upper-triangular when estimated from 3D measurements.

Camera Calibration

- We know some 3D points and we want to obtain the camera intrinsics and extrinsics.
- The relationship between the 3D pixel center $\mathbf{p}$ and the 3D camera-centered point $\mathbf{p}_c$ is given by an unknown scaling s, i.e., $\mathbf{p} = s\mathbf{p}_c$.
- We can therefore write the complete projection as

$$\bar{\mathbf{p}}_s = \alpha \mathbf{M}_s^{-1} \mathbf{p}_c = \mathbf{K}\mathbf{p}_c$$

  with $\mathbf{K}$ the **calibration matrix** that describes the camera intrinsic parameters.
- Combining with the external parameters we have

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

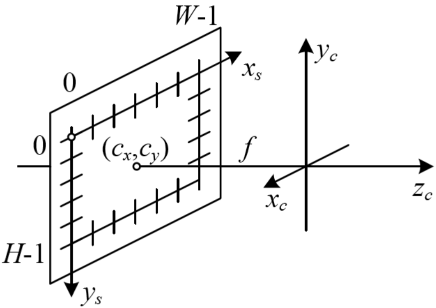
  with $\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$ the **camera matrix**.
- Problem of identification, so $\mathbf{K}$ is assumed upper-triangular when estimated from 3D measurements.

Camera Calibration

- Thus it is typically assumed to be

$$\mathbf{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \mathbf{p}_w = \mathbf{P}\mathbf{p}_w$$

with typically $f_x = f_y$ and $s = 0$.



[Source: R. Szeliski]

Camera Matrix

- We can put intrinsics and extrinsics together in a 3×4 **camera matrix**

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$$

- Or in homogeneous coordinates we have an invertible matrix

$$\bar{\mathbf{P}} = \begin{bmatrix} \mathbf{K} & 0 \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0^T & 1 \end{bmatrix} = \bar{\mathbf{K}}\mathbf{E}$$

  with **E** a 3D rigid body (Euclidean) transformation.

Camera Matrix

- We can put intrinsics and extrinsics together in a 3×4 **camera matrix**

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$$

- Or in homogeneous coordinates we have an invertible matrix

$$\bar{\mathbf{P}} = \begin{bmatrix} \mathbf{K} & 0 \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0^T & 1 \end{bmatrix} = \bar{\mathbf{K}}\mathbf{E}$$

  with **E** a 3D rigid body (Euclidean) transformation.

- We can map a 3D point in homogeneous coordinates into the image plane as

$$\mathbf{p}_s \sim \bar{\mathbf{P}}\bar{\mathbf{p}}_w$$

Camera Matrix

- We can put intrinsics and extrinsics together in a 3×4 **camera matrix**

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$$

- Or in homogeneous coordinates we have an invertible matrix

$$\bar{\mathbf{P}} = \begin{bmatrix} \mathbf{K} & 0 \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0^T & 1 \end{bmatrix} = \bar{\mathbf{K}}\mathbf{E}$$

  with **E** a 3D rigid body (Euclidean) transformation.

- We can map a 3D point in homogeneous coordinates into the image plane as

$$\mathbf{p}_s \sim \bar{\mathbf{P}}\bar{\mathbf{p}}_w$$

- After multiplication by $\bar{\mathbf{P}}$, the vector is divided by the third element of the vector to obtain the normalized form $\mathbf{p}_s = (x_s, y_s, 1, d)$.

Camera Matrix

- We can put intrinsics and extrinsics together in a 3×4 **camera matrix**

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$$

- Or in homogeneous coordinates we have an invertible matrix

$$\bar{\mathbf{P}} = \begin{bmatrix} \mathbf{K} & 0 \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0^T & 1 \end{bmatrix} = \bar{\mathbf{K}}\mathbf{E}$$

  with **E** a 3D rigid body (Euclidean) transformation.

- We can map a 3D point in homogeneous coordinates into the image plane as

$$\mathbf{p}_s \sim \bar{\mathbf{P}}\bar{\mathbf{p}}_w$$

- After multiplication by $\bar{\mathbf{P}}$, the vector is divided by the third element of the vector to obtain the normalized form $\mathbf{p}_s = (x_s, y_s, 1, d)$.

Photometric image formation

Lighting: Point Source

- To produce an image, the scene must be illuminated with one or more light sources.
- A point light source originates at a single location in space.
- In addition to its location, a point light source has an intensity and a color spectrum, i.e., a distribution over wavelengths $L(\lambda)$.
- The intensity of a light source falls off with the square of the distance between the source and the object being lit, because the same light is being spread over a larger (spherical) area.

Lighting: More complex sources

- A more complex light distribution can often be represented using an environment map.
- This representation maps incident light directions $\mathbf{v}$ to color values (or wavelengths λ)

$$L(\mathbf{v}, \lambda)$$

Reflectance and shading

- When light hits an objects surface, it is scattered and reflected.
- The **Bidirectional Reflectance Distribution Function (BRDF)** is a 4D function $f(\theta_i, \phi_i, \theta_r, \phi_r, \lambda)$ that describes how much of each wavelength λ arriving at an incident direction $\mathbf{v}_i$ is emitted in a reflected direction $\mathbf{v}_r$.
- It is reciprocal, we can exchange $\mathbf{v}_i$ and $\mathbf{v}_r$.

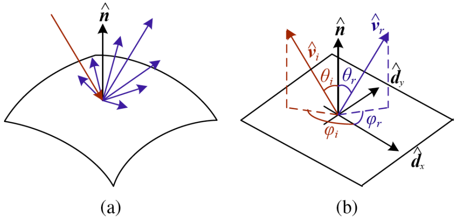


Figure: (a) Light scatters when it hits a surface. (b) The BRDF is parameterized by the angles that the incident, $\mathbf{v}_i$ and reflected, $\mathbf{v}_r$, light ray directions make with the surface coordinate frame $(d_x, d_y, \mathbf{n})$.

[Source: R. Szeliski]

BRDF and light

- For an isotropic material $f_r(\theta_i, \theta_r, |\phi_i - \phi_r|, \lambda)$ or $f_r(\mathbf{v}_i, \mathbf{v}_r, \mathbf{n}, \lambda)$
- The amount of light exiting a surface point p in a direction $\mathbf{v}_r$ under a given lighting condition is

$$L_r(\mathbf{v}_r, \lambda) = \int L_i(\mathbf{v}_i, \lambda) f_r(\mathbf{v}_i, \mathbf{v}_r, \mathbf{n}, \lambda) max(0, \cos\theta_i) d\mathbf{v}_i$$

- If the light sources are a discrete set of point light sources, then the integral is a sum

$$L_r(\mathbf{v}_r, \lambda) = \sum_i L_i(\lambda) f_r(\mathbf{v}_i, \mathbf{v}_r, \mathbf{n}, \lambda) max(0, \cos\theta_i) d\mathbf{v}_i$$

Diffuse Reflection

- Also known as **Lambertian** or **matte reflection**.
- Scatters light uniformly in all directions and is the phenomenon we most normally associate with **shading**.
- In this case the BRDF is constant

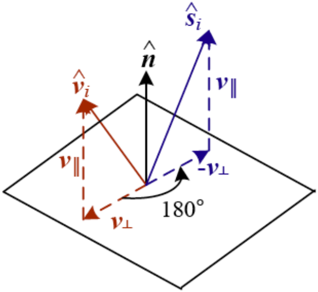
$$f_r(\mathbf{v}_i, \mathbf{v}_r, \mathbf{n}, \lambda) = f_r(\lambda)$$

- The amount of light depends on the angle between the incident light direction and the surface normal, θ_i. The **shading equation** for diffuse reflection is then

$$L_d(\mathbf{v}_r, \lambda) = \sum_i L_i(\lambda) f_d(\lambda) max(0, \mathbf{v}_i \cdot \mathbf{n})$$

Specularities

- The second major component of the BRDF is specular reflection, which depends on the direction of the outgoing light.
- Incident light rays are reflected in a direction that is rotated by 180 around the surface normal **n**.
- The amount of light reflected in a given direction $\mathbf{v}_r$ thus depends on the angle between the view direction $\mathbf{v}_r$ and the specular direction $\mathbf{s}_i$.



[Source: R. Szeliski]

Phong Model

- Combines the diffuse and specular components of reflection with another term, which he called the ambient illumination.
- This term accounts for the fact that objects are generally illuminated not only by point light sources but also by a general diffuse illumination corresponding to inter-reflection (e.g., the walls in a room) or distant sources such as the sky.
- The ambient term does not depend on surface orientation, but depends on the color of both the ambient illumination and the object

$$L = k_a(\lambda) L_a(\lambda) + k_d(\lambda) \sum_i L_i(\lambda) max(0, \mathbf{v}_i \cdot \mathbf{n}) + L_s$$

- There exists more models, we just mentioned the most used ones.

Typical Shading

Figure: Diffuse (smooth shading) and specular (shiny highlight) reflection, as well as darkening in the grooves and creases due to reduced light visibility and interreflections. Photo from the Caltech Vision Lab

[Source: R. Szeliski]

Next class ... some image fundamentals