

Euclidean Geometry

CSE 6367: Computer Vision

Instructor: William J. Beksí

Introduction

- Before we can intelligently analyze and manipulate images, we need to establish a vocabulary for describing the geometry of a scene

Introduction

- Before we can intelligently analyze and manipulate images, we need to establish a vocabulary for describing the geometry of a scene
- We'll use basic **geometric primitives** (points, lines, planes) and **geometric transformations** that project these 3D quantities into 2D image features

2D Points

- 2D points (pixel coordinates in an image) are denoted using a pair of values, $(x, y) \in \mathbb{R}^2$, as

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

or alternatively, $\mathbf{x} = [x, y]^T$

Homogeneous Coordinates

- 2D points can be represented using **homogeneous coordinates**, $\tilde{\mathbf{x}} = [\tilde{x}, \tilde{y}, \tilde{w}]^T \in \mathbb{P}^2$, where vectors that differ only by scale are considered to be equivalent

Homogeneous Coordinates

- 2D points can be represented using **homogeneous coordinates**, $\tilde{\mathbf{x}} = [\tilde{x}, \tilde{y}, \tilde{w}]^T \in \mathbb{P}^2$, where vectors that differ only by scale are considered to be equivalent
- $\mathbb{P}^2 = \mathbb{R}^3 - [0, 0, 0]^T$ is called the 2D **projective space**

Inhomogeneous Vectors

- A homogeneous vector $\tilde{\mathbf{x}}$ can be converted back into an **inhomogeneous vector** \mathbf{x} by dividing through by the last element \tilde{w} , i.e.,

$$\begin{aligned}\tilde{\mathbf{x}} &= [\tilde{x}, \tilde{y}, \tilde{w}]^T \\ &= \tilde{w}[x, y, 1]^T \\ &= \tilde{w}\bar{\mathbf{x}},\end{aligned}$$

where $\bar{\mathbf{x}} = [x, y, 1]^T$ is the **augmented vector**

Inhomogeneous Vectors

- A homogeneous vector $\tilde{\mathbf{x}}$ can be converted back into an **inhomogeneous vector** \mathbf{x} by dividing through by the last element \tilde{w} , i.e.,

$$\begin{aligned}\tilde{\mathbf{x}} &= [\tilde{x}, \tilde{y}, \tilde{w}]^T \\ &= \tilde{w}[x, y, 1]^T \\ &= \tilde{w}\bar{\mathbf{x}},\end{aligned}$$

where $\bar{\mathbf{x}} = [x, y, 1]^T$ is the **augmented vector**

- Homogeneous points whose last element is $\tilde{w} = 0$ are called **ideal points** or **points at infinity** and do not have an equivalent inhomogeneous representation

2D Lines

- 2D lines can be represented using homogeneous coordinates
 $\tilde{\mathbf{l}} = [a, b, c]^T$

2D Lines

- 2D lines can be represented using homogeneous coordinates

$$\tilde{\mathbf{l}} = [a, b, c]^T$$

- The corresponding **line equation** is

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

Normal Vector

- We can normalize the line equation vector so that

$$\begin{aligned}\mathbf{l} &= [\hat{n}_x, \hat{n}_y, d]^T \\ &= [\hat{\mathbf{n}}, d]^T\end{aligned}$$

with $\|\hat{\mathbf{n}}\| = 1$

Normal Vector

- We can normalize the line equation vector so that

$$\begin{aligned}\mathbf{l} &= [\hat{n}_x, \hat{n}_y, d]^T \\ &= [\hat{\mathbf{n}}, d]^T\end{aligned}$$

with $\|\hat{\mathbf{n}}\| = 1$

- In this case, $\hat{\mathbf{n}}$ is the **normal vector** perpendicular to the line and d is the distance to the origin

Polar Coordinates

- We can also express $\hat{\mathbf{n}}$ as a function of rotation angle θ ,

$$\begin{aligned}\hat{\mathbf{n}} &= [\hat{n}_x, \hat{n}_y]^T \\ &= [\cos \theta, \sin \theta]^T\end{aligned}$$

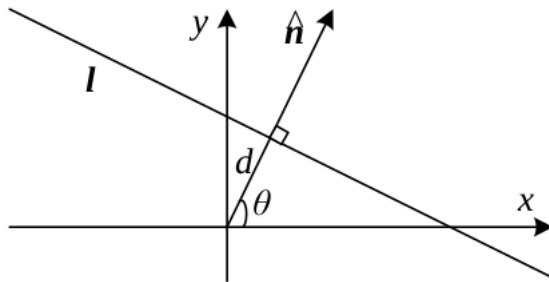
Polar Coordinates

- We can also express $\hat{\mathbf{n}}$ as a function of rotation angle θ ,

$$\begin{aligned}\hat{\mathbf{n}} &= [\hat{n}_x, \hat{n}_y]^T \\ &= [\cos \theta, \sin \theta]^T\end{aligned}$$

- The combination $[\theta, d]^T$ is known as **polar coordinates**

2D Line Equation



- 2D line equation: $\vec{x} \cdot \vec{l} = ax + by + c = 0$

Intersection of Lines

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

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

where \times is the cross product operator

Intersection of Lines

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

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

where \times is the cross product operator

- Similarly, the line joining two points can be written as

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

Intersection of Lines

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

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

where \times is the cross product operator

- Similarly, the line joining two points can be written as

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

- When trying to fit an intersection point to multiple lines (or a line to multiple points), least squares techniques can be used

2D Conics

- There are other algebraic curves that can be expressed with simple polynomial homogeneous equations

2D Conics

- There are other algebraic curves that can be expressed with simple polynomial homogeneous equations
- For example, **conic sections** (the intersection of a plane and a 3D cone) can be written using a **quadric** equation

$$\tilde{\mathbf{x}}^T Q \tilde{\mathbf{x}} = 0$$

2D Conics

- There are other algebraic curves that can be expressed with simple polynomial homogeneous equations
- For example, **conic sections** (the intersection of a plane and a 3D cone) can be written using a **quadric** equation

$$\tilde{\mathbf{x}}^T Q \tilde{\mathbf{x}} = 0$$

- Quadric equations play useful roles in the study of multiview geometry and camera calibration

3D Points

- Point coordinates in three dimensions can be written using inhomogeneous coordinates, $\mathbf{x} = [x, y, z]^T \in \mathbb{R}^3$, or homogeneous coordinates, $\tilde{\mathbf{x}} = [\tilde{x}, \tilde{y}, \tilde{z}, \tilde{w}]^T \in \mathbb{P}^3$

3D Points

- Point coordinates in three dimensions can be written using inhomogeneous coordinates, $\mathbf{x} = [x, y, z]^T \in \mathbb{R}^3$, or homogeneous coordinates, $\tilde{\mathbf{x}} = [\tilde{x}, \tilde{y}, \tilde{z}, \tilde{w}]^T \in \mathbb{P}^3$
- It is sometimes useful to denote a 3D point using the augmented vector $\bar{\mathbf{x}} = [x, y, z, 1]^T$ with $\tilde{\mathbf{x}} = \tilde{w}\bar{\mathbf{x}}$

3D Planes

- 3D planes can be represented as homogeneous coordinates $\tilde{\mathbf{m}} = [a, b, c, d]^T$ with a corresponding plane equation

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

3D Planes

- 3D planes can be represented as homogeneous coordinates $\tilde{\mathbf{m}} = [a, b, c, d]^T$ with a corresponding plane equation

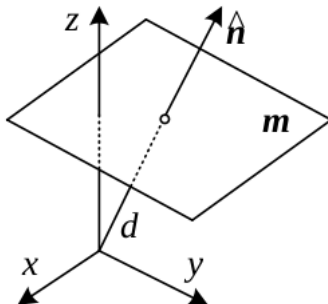
$$\bar{\mathbf{x}} \cdot \tilde{\mathbf{m}} = ax + by + cz + d = 0$$

- We can normalize the plane equation as

$$\begin{aligned}\mathbf{m} &= [\hat{n}_x, \hat{n}_y, \hat{n}_z, d]^T \\ &= [\hat{\mathbf{n}}, d]^T\end{aligned}$$

with $\|\hat{\mathbf{n}}\| = 1$

3D Plane Equation



- 3D plane equation: $\bar{\mathbf{x}} \cdot \hat{\mathbf{m}} = ax + by + cz + d = 0$

Spherical Coordinates

- We can express $\hat{\mathbf{n}}$ as a function of two angles (θ, ϕ)

$$\hat{\mathbf{n}} = [\cos \theta, \cos \phi, \sin \theta, \sin \phi]^T,$$

using **spherical coordinates**

Spherical Coordinates

- We can express $\hat{\mathbf{n}}$ as a function of two angles (θ, ϕ)

$$\hat{\mathbf{n}} = [\cos \theta, \cos \phi, \sin \theta, \sin \phi]^T,$$

using **spherical coordinates**

- Spherical coordinates are less commonly used than polar coordinates since they do not uniformly sample the space of possible normal vectors

3D Lines

- One possible representation of a 3D line is to use two points, (\mathbf{p}, \mathbf{q}) , on the line

3D Lines

- One possible representation of a 3D line is to use two points, (\mathbf{p}, \mathbf{q}) , on the line
- Any point on the line can be expressed as a linear combination of these two points

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

3D Lines

- One possible representation of a 3D line is to use two points, (\mathbf{p}, \mathbf{q}) , on the line
- Any point on the line can be expressed as a linear combination of these two points

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

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

3D Lines

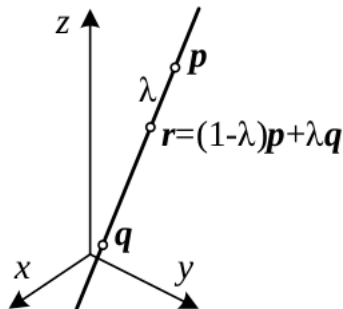
- One possible representation of a 3D line is to use two points, (\mathbf{p}, \mathbf{q}) , on the line
- Any point on the line can be expressed as a linear combination of these two points

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

- If we restrict $0 \leq \lambda \leq 1$, then we get the **line segment** joining \mathbf{p} and \mathbf{q}
- Using homogeneous coordinates, we can write the line as

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

3D Line Equation

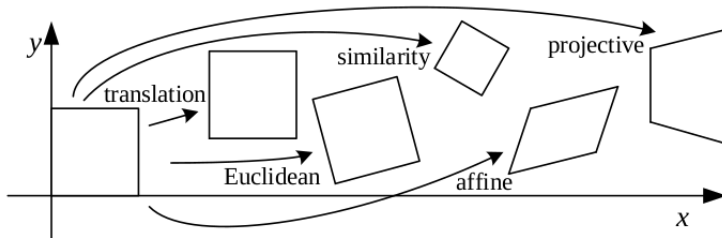


- 3D line equation: $\mathbf{r} = (1 - \lambda)\mathbf{p} + \lambda\mathbf{q}$ where $0 \leq \lambda \leq 1$

Review Question

- Find the point of intersection of the lines $3x - y - 1 = 0$ and $2x - y + 3 = 0$

2D Transformations



- Basic set of 2D planar transformations

Translation

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

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

where I is the (2×2) identity matrix or

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

where $\mathbf{0}$ is the zero vector

Rotation and Translation

- This transformation is also known as the **2D rigid body motion** or **2D Euclidean transformation** (since Euclidean distances are preserved) and can be written as

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

where

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

is an orthonormal rotation matrix with $RR^T = I$ and $\det(R) = 1$.

Scaled Rotation

- This transformation is also known as the **similarity transform** and can be expressed as

$$\begin{aligned}\mathbf{x}' &= \begin{bmatrix} sR & \mathbf{t} \end{bmatrix} \bar{\mathbf{x}} \\ &= \begin{bmatrix} a & -b & t_x \\ b & a & t_y \end{bmatrix} \bar{\mathbf{x}}\end{aligned}$$

where we no longer require that $a^2 + b^2 = 1$

Scaled Rotation

- This transformation is also known as the **similarity transform** and can be expressed as

$$\begin{aligned}\mathbf{x}' &= [sR \quad \mathbf{t}] \bar{\mathbf{x}} \\ &= \begin{bmatrix} a & -b & t_x \\ b & a & t_y \end{bmatrix} \bar{\mathbf{x}}\end{aligned}$$

where we no longer require that $a^2 + b^2 = 1$

- The similarity transform preserves angles between lines

Affine

- The affine transformation is written as $\mathbf{x}' = A\bar{\mathbf{x}}$, where A is an arbitrary 2×3 matrix, i.e.

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

Affine

- The affine transformation is written as $\mathbf{x}' = A\bar{\mathbf{x}}$, where A is an arbitrary 2×3 matrix, i.e.

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

- Parallel lines remain parallel under affine transformations

Projective

- This transformation, also known as a **perspective transform** or **homography**, operates on homogeneous coordinates,

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

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

Projective

- This transformation, also known as a **perspective transform** or **homography**, operates on homogeneous coordinates,

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

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

- Note that \tilde{H} is homogeneous, i.e. it is only defined up to a scale, and that two \tilde{H} matrices that differ only by scale are equivalent

Projective

- This transformation, also known as a **perspective transform** or **homography**, operates on homogeneous coordinates,

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

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

- Note that \tilde{H} is homogeneous, i.e. it is only defined up to a scale, and that two \tilde{H} matrices that differ only by scale are equivalent
- Perspective transformations preserve straight lines (i.e. they remain straight after the transformation)

Stretch/Squash






- This transformation changes the aspect ratio of an image,

$$x' = s_x x + t_x$$

$$y' = s_y y + t_y$$

and is a restricted form of an affine transformation

2D Coordinate Transformations

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

- Hierarchy of 2D coordinate transformations

Review Question

- Determine a matrix T that represents a translation of b units along the x-axis followed by a rotation of θ

Translation

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

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

where I is the (3×3) identity matrix

Rotation and Translation

- Also known as **3D rigid body motion** or the **3D Euclidean transformation**, rotation and translation can be written as $\mathbf{x}' = R\mathbf{x} + \mathbf{t}$ or

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

where R is a 3×3 orthonormal rotation matrix with $RR^T = I$ and $\det(R) = 1$.

Rotation and Translation

- Also known as **3D rigid body motion** or the **3D Euclidean transformation**, rotation and translation can be written as $\mathbf{x}' = R\mathbf{x} + \mathbf{t}$ or

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

where R is a 3×3 orthonormal rotation matrix with $RR^T = I$ and $\det(R) = 1$.

- Sometimes it is more convenient to describe rigid motion using

$$\mathbf{x}' = R(\mathbf{x} - \mathbf{c}) = R\mathbf{x} - R\mathbf{c}$$

where \mathbf{c} is the center of rotation (often the camera center)

Scaled Rotation

- The 3D **similarity transform** can be expressed as $\mathbf{x}' = sR\mathbf{x} + \mathbf{t}$ or

$$\mathbf{x}' = \begin{bmatrix} sR & \mathbf{t} \end{bmatrix} \bar{\mathbf{x}}$$

where s is an arbitrary scale factor

Scaled Rotation

- The 3D **similarity transform** can be expressed as $\mathbf{x}' = sR\mathbf{x} + \mathbf{t}$ or

$$\mathbf{x}' = \begin{bmatrix} sR & \mathbf{t} \end{bmatrix} \bar{\mathbf{x}}$$

where s is an arbitrary scale factor

- The transformation preserves angles between lines and planes

Affine

- The 3D affine transform is written as $\mathbf{x}' = A\bar{\mathbf{x}}$, where A is an arbitrary 3×4 matrix, i.e.

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

Affine

- The 3D affine transform is written as $\mathbf{x}' = A\bar{\mathbf{x}}$, where A is an arbitrary 3×4 matrix, i.e.

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

- Parallel lines and planes remain parallel under affine transformations

Projective

- This transformation, variously known as a 3D **perspective transform**, **homography**, or **collineation**, operates on homogeneous coordinates,

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

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

Projective

- This transformation, variously known as a 3D **perspective transform**, **homography**, or **collineation**, operates on homogeneous coordinates,

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

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

- As in 2D, the resulting homogeneous coordinate $\tilde{\mathbf{x}}'$ must be normalized in order to obtain an inhomogeneous result \mathbf{x}

Projective






- This transformation, variously known as a 3D **perspective transform**, **homography**, or **collineation**, operates on homogeneous coordinates,

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

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

- As in 2D, the resulting homogeneous coordinate $\tilde{\mathbf{x}}'$ must be normalized in order to obtain an inhomogeneous result \mathbf{x}
- Perspective transformations preserve straight lines

3D Coordinate Transformations

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

- Hierarchy of 3D coordinate transformations

Review Question

- Determine a matrix T that has been scaled by $[s_x, s_y, s_z]$

Euler Angles

- A rotation matrix can be formed as the product of three rotations around three cardinal axes, e.g. x , y , and z or x , y , and x

Euler Angles

- A rotation matrix can be formed as the product of three rotations around three cardinal axes, e.g. x , y , and z or x , y , and x
- In general this is a bad idea since the result depends on the order in which the transforms are applied

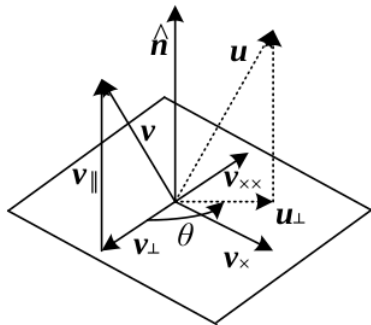
Euler Angles

- A rotation matrix can be formed as the product of three rotations around three cardinal axes, e.g. x , y , and z or x , y , and x
- In general this is a bad idea since the result depends on the order in which the transforms are applied
- Even worse, it is not always possible to move smoothly in the parameter space, i.e. one or more of the Euler angles can change dramatically in response to a small change in rotation (e.g. gimbal lock)

Euler Angles

- A rotation matrix can be formed as the product of three rotations around three cardinal axes, e.g. x , y , and z or x , y , and x
- In general this is a bad idea since the result depends on the order in which the transforms are applied
- Even worse, it is not always possible to move smoothly in the parameter space, i.e. one of more of the Euler angles can change dramatically in response to a small change in rotation (e.g. gimbal lock)
- Therefore, we will work with two alternative representations: **axis/angle** and **unit quaternions**

Axis/Angle Representation



- Example: rotation around an axis $\hat{\mathbf{n}}$ by an angle θ

Axis/Angle Representation

- A rotation can be represented by a rotation axis $\hat{\mathbf{n}}$ and angle θ , or equivalently by a 3D vector $\boldsymbol{\omega} = \theta\hat{\mathbf{n}}$

Axis/Angle Representation

- A rotation can be represented by a rotation axis $\hat{\mathbf{n}}$ and angle θ , or equivalently by a 3D vector $\boldsymbol{\omega} = \theta\hat{\mathbf{n}}$
- First, we project the vector \mathbf{v} onto the axis $\hat{\mathbf{n}}$ to obtain

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

which is the component of \mathbf{v} that is not affected by the rotation

Axis/Angle Representation

- Next, we compute the perpendicular residual of \mathbf{v} from $\hat{\mathbf{n}}$,

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

Axis/Angle Representation

- Next, we compute the perpendicular residual of \mathbf{v} from $\hat{\mathbf{n}}$,

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

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

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

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

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

Axis/Angle Representation

- Rotating this vector by another 90° is equivalent to taking the cross product again,

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

and hence

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

Axis/Angle Representation

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

$$\begin{aligned}\mathbf{u}_{\perp} &= \cos \theta \mathbf{v}_{\perp} + \sin \theta \mathbf{v}_{\times} \\ &= (\sin \theta [\hat{n}]_{\times} - \cos \theta [\hat{n}]_{\times}^2) \mathbf{v}\end{aligned}$$

Axis/Angle Representation

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

$$\begin{aligned}\mathbf{u}_{\perp} &= \cos \theta \mathbf{v}_{\perp} + \sin \theta \mathbf{v}_{\times} \\ &= (\sin \theta [\hat{n}]_{\times} - \cos \theta [\hat{n}]_{\times}^2) \mathbf{v}\end{aligned}$$

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

$$\begin{aligned}\mathbf{u} &= \mathbf{u}_{\perp} + \mathbf{v}_{\parallel} \\ &= I + \sin \theta [\hat{n}]_{\times} + (1 - \cos \theta [\hat{n}]_{\times}^2)\end{aligned}$$

Axis/Angle Representation

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

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

which is known as **Rodriguez's formula**

Axis/Angle Representation

- The product of the axis $\hat{\mathbf{n}}$ and angle θ , $\boldsymbol{\omega} = \theta\hat{\mathbf{n}} = [\omega_x, \omega_y, \omega_z]^T$, is a minimal representation for a 3D rotation

Axis/Angle Representation

- The product of the axis $\hat{\mathbf{n}}$ and angle θ , $\boldsymbol{\omega} = \theta\hat{\mathbf{n}} = [\omega_x, \omega_y, \omega_z]^T$, is a minimal representation for a 3D rotation
- Rotations through common angles, such as multiples of 90° , can be represented exactly (and converted to exact matrices) if θ is stored in degrees

Axis/Angle Representation

- Unfortunately, this representation is not unique since we can add a multiple of 360° (2π radians) to θ and get the same rotation matrix ($(\hat{\mathbf{n}}, \theta)$ and $(-\hat{\mathbf{n}}, -\theta)$ represent the same rotation)

Axis/Angle Representation

- Unfortunately, this representation is not unique since we can add a multiple of 360° (2π radians) to θ and get the same rotation matrix ($(\hat{\mathbf{n}}, \theta)$ and $(-\hat{\mathbf{n}}, -\theta)$ represent the same rotation)
- However, for small rotations (e.g. corrections to rotations), this is an excellent choice

Axis/Angle Representation

- In particular, for small (infinitesimal or instantaneous) rotations and θ expressed in radians, Rodriguez's formula simplifies to

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

which gives a nice linearized relationship between the rotation parameters ω and R

Axis/Angle Representation

- We can also write $R(\boldsymbol{\omega})\mathbf{v} \approx \mathbf{v} + \boldsymbol{\omega} \times \mathbf{v}$, which is useful when we want to compute the derivative of $R\mathbf{v}$ w.r.t $\boldsymbol{\omega}$,

$$\frac{\partial R\mathbf{v}}{\partial \boldsymbol{\omega}^T} = -[\mathbf{v}]_{\times} = \begin{bmatrix} 0 & z & -y \\ -z & 0 & x \\ y & -x & 0 \end{bmatrix}$$

Unit Quaternion Representation

- A unit quaternion is a unit length 4-vector whose components can be written as $\mathbf{q} = [q_x, q_y, q_z, q_w]^T$ or $\mathbf{q} = [x, y, z, w]^T$

Unit Quaternion Representation

- A unit quaternion is a unit length 4-vector whose components can be written as $\mathbf{q} = [q_x, q_y, q_z, q_w]^T$ or $\mathbf{q} = [x, y, z, w]^T$
- Unit quaternions live on the unit sphere $\|\mathbf{q}\| = 1$ and **antipodal** (opposite sign) quaternions, \mathbf{q} and $-\mathbf{q}$, represent the same rotation

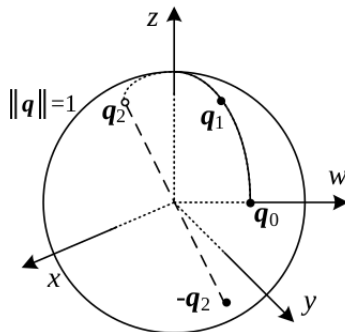
Unit Quaternion Representation

- A unit quaternion is a unit length 4-vector whose components can be written as $\mathbf{q} = [q_x, q_y, q_z, q_w]^T$ or $\mathbf{q} = [x, y, z, w]^T$
- Unit quaternions live on the unit sphere $\|\mathbf{q}\| = 1$ and **antipodal** (opposite sign) quaternions, \mathbf{q} and $-\mathbf{q}$, represent the same rotation
- The unit quaternion representation is unique and continuous

Unit Quaternion Representation

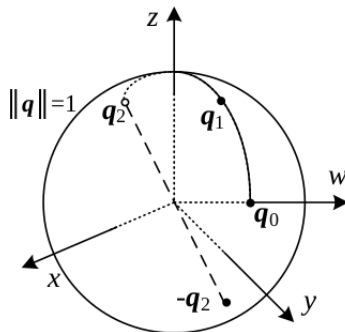
- A unit quaternion is a unit length 4-vector whose components can be written as $\mathbf{q} = [q_x, q_y, q_z, q_w]^T$ or $\mathbf{q} = [x, y, z, w]^T$
- Unit quaternions live on the unit sphere $\|\mathbf{q}\| = 1$ and **antipodal** (opposite sign) quaternions, \mathbf{q} and $-\mathbf{q}$, represent the same rotation
- The unit quaternion representation is unique and continuous
- Quaternions are a very popular representation for pose and pose interpolation in computer graphics

Unit Quaternion Representation



- Example: a smooth trajectory through the three quaternions \mathbf{q}_0 , \mathbf{q}_1 , and \mathbf{q}_2

Unit Quaternion Representation



- Example: a smooth trajectory through the three quaternions \mathbf{q}_0 , \mathbf{q}_1 , and \mathbf{q}_2
- The antipodal point to \mathbf{q}_2 , namely $-\mathbf{q}_2$, represents the same rotation as \mathbf{q}_2

Unit Quaternion Representation

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

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

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

Unit Quaternion Representation

- Using the trigonometric identities $\sin \theta = 2 \sin \frac{\theta}{2} \cos \frac{\theta}{2}$ and $(1 - \cos \theta) = 2 \sin^2 \frac{\theta}{2}$, Rodriguez's formula can be converted to

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

Unit Quaternion Representation

- This suggests a quick way to rotate a vector \mathbf{v} by a quaternion using a series of cross products, scalings, and additions

Unit Quaternion Representation

- This suggests a quick way to rotate a vector \mathbf{v} by a quaternion using a series of cross products, scalings, and additions
- To obtain a formula for $\mathbf{R}(\mathbf{q})$ as a function of $[x, y, z, w]$, recall that

$$[v]_{\times} = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \text{ and } [v]_{\times}^2 = \begin{bmatrix} -y^2 - z^2 & xy & xz \\ xy & -x^2 - z^2 & yz \\ xz & yz & -x^2 - y^2 \end{bmatrix}$$

thus we obtain

$$\mathbf{R}(\mathbf{q}) = \begin{bmatrix} 1 - 2(y^2 + z^2) & 2(xy - zw) & 2(xz + yw) \\ 2(xy + zw) & 1 - 2(x^2 + z^2) & 2(yz - xw) \\ 2(xz - yw) & 2(yz + xw) & 1 - 2(x^2 + y^2) \end{bmatrix}$$

Unit Quaternion Representation

- The nicest aspect of unit quaternions is that there is a simple algebra for composing rotations expressed as unit quaternions

Unit Quaternion Representation

- The nicest aspect of unit quaternions is that there is a simple algebra for composing rotations expressed as unit quaternions
- Given two quaternions $\mathbf{q}_0 = (\mathbf{v}_0, w_0)$ and $\mathbf{q}_1 = (\mathbf{v}_1, w_1)$, the **quaternion multiply** operator is defined as

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

with the property that $\mathbf{R}(\mathbf{q}_2) = \mathbf{R}(\mathbf{q}_0)\mathbf{R}(\mathbf{q}_1)$

Unit Quaternion Representation

- The nicest aspect of unit quaternions is that there is a simple algebra for composing rotations expressed as unit quaternions
- Given two quaternions $\mathbf{q}_0 = (\mathbf{v}_0, w_0)$ and $\mathbf{q}_1 = (\mathbf{v}_1, w_1)$, the **quaternion multiply** operator is defined as

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

with the property that $\mathbf{R}(\mathbf{q}_2) = \mathbf{R}(\mathbf{q}_0)\mathbf{R}(\mathbf{q}_1)$

- Note that quaternion multiplication is *not* commutative, just as 3D rotations and matrix multiplications are not

Unit Quaternion Representation

- Taking the inverse of a quaternion is easy: just flip the sign of \mathbf{v} or w (but not both!)

Unit Quaternion Representation

- Taking the inverse of a quaternion is easy: just flip the sign of \mathbf{v} or w (but not both!)
- Thus, we can define **quaternion division** as

$$\mathbf{q}_2 = \mathbf{q}_0 / \mathbf{q}_1 = \mathbf{q}_0 \mathbf{q}_1^{-1} = (\mathbf{v}_0 \times \mathbf{v}_1 + w_0 \mathbf{v}_1 - w_1 \mathbf{v}_0, -w_0 w_1 - \mathbf{v}_0 \cdot \mathbf{v}_1)$$

Unit Quaternion Representation

- Taking the inverse of a quaternion is easy: just flip the sign of \mathbf{v} or w (but not both!)
- Thus, we can define **quaternion division** as

$$\mathbf{q}_2 = \mathbf{q}_0 / \mathbf{q}_1 = \mathbf{q}_0 \mathbf{q}_1^{-1} = (\mathbf{v}_0 \times \mathbf{v}_1 + w_0 \mathbf{v}_1 - w_1 \mathbf{v}_0, -w_0 w_1 - \mathbf{v}_0 \cdot \mathbf{v}_1)$$

- This is useful when the **incremental rotation** between two rotations is desired

Unit Quaternion Representation

- If we want to determine a rotation that is partway between two given rotations, we can compute the incremental rotation, take a fraction of the angle, and compute the new rotation

Unit Quaternion Representation

- If we want to determine a rotation that is partway between two given rotations, we can compute the incremental rotation, take a fraction of the angle, and compute the new rotation
- This procedure is called **spherical linear interpolation** or **slerp**

Unit Quaternion Representation

procedure *slerp*(q_0, q_1, α):

1. $q_r = q_1/q_0 = (v_r, w_r)$
2. if $w_r < 0$ then $q_r \leftarrow -q_r$
3. $\theta_r = 2 \tan^{-1}(\|v_r\|/w_r)$
4. $\hat{n}_r = \mathcal{N}(v_r) = v_r/\|v_r\|$
5. $\theta_\alpha = \alpha \theta_r$
6. $q_\alpha = (\sin \frac{\theta_\alpha}{2} \hat{n}_r, \cos \frac{\theta_\alpha}{2})$
7. **return** $q_2 = q_\alpha q_0$

- Spherical linear interpolation (slerp): the axis and total angle are first computed from the quaternion ratio, then an incremental quaternion is computed and multiplied by the starting rotation quaternion

Which Rotation Representation is Better?

- The choice of representation for 3D rotations depends partly on the application

Which Rotation Representation is Better?

- The choice of representation for 3D rotations depends partly on the application
- Axis/angle: representation is minimal, does not require any additional constraints on the parameters, if the angle is expressed in degrees it is easier to understand the pose (e.g. 90° twist around x -axis), easier to express exact rotations, when angle is in radians it is easy to compute the derivatives of R w.r.t ω

Which Rotation Representation is Better?

- The choice of representation for 3D rotations depends partly on the application
- Axis/angle: representation is minimal, does not require any additional constraints on the parameters, if the angle is expressed in degrees it is easier to understand the pose (e.g. 90° twist around x-axis), easier to express exact rotations, when angle is in radians it is easy to compute the derivatives of R w.r.t ω
- Unit quaternions: allow for keeping track of a smoothly moving camera (no discontinuities in the representation), easier to interpolate between rotations and chain them to rigid transformations

Review Questions

- Using Rodriguez's formula, find the rotation matrix corresponding to a rotation of 78° about an axis that passes through the point $[1, 3.7, 8]$

Review Questions

- Using Rodriguez's formula, find the rotation matrix corresponding to a rotation of 78° about an axis that passes through the point $[1, 3.7, 8]$
- Determine the quaternion that represents a rotation of 90° about the z-axis

3D to 2D Projections

- Now that we know how to represent 2D and 3D geometric primitives and how to transform them spatially, we need to specify how 3D primitives are projected onto the image plane

3D to 2D Projections

- Now that we know how to represent 2D and 3D geometric primitives and how to transform them spatially, we need to specify how 3D primitives are projected onto the image plane
- We can do this using a linear 3D to 2D projection matrix

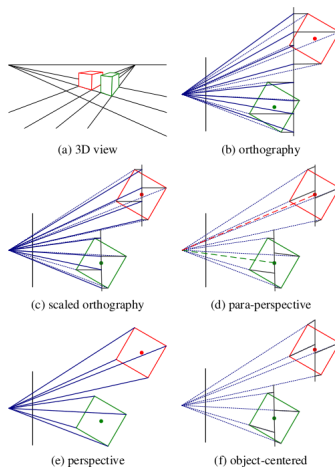
3D to 2D Projections

- Now that we know how to represent 2D and 3D geometric primitives and how to transform them spatially, we need to specify how 3D primitives are projected onto the image plane
- We can do this using a linear 3D to 2D projection matrix
- The simplest model is orthography which requires no division to get the final (inhomogeneous) result

3D to 2D Projections

- Now that we know how to represent 2D and 3D geometric primitives and how to transform them spatially, we need to specify how 3D primitives are projected onto the image plane
- We can do this using a linear 3D to 2D projection matrix
- The simplest model is orthography which requires no division to get the final (inhomogeneous) result
- The more commonly used model is perspective since this more accurately models the behavior of real cameras

Commonly used Projection Models



Orthography and Para-Perspective

- An orthographic projection simply drops the z component of the three-dimensional coordinate \mathbf{p} to obtain the 2D point \mathbf{x} and can be written as

$$\mathbf{x} = [I_{2 \times 2} | \mathbf{0}] \mathbf{p}$$

Orthography and Para-Perspective

- An orthographic projection simply drops the z component of the three-dimensional coordinate \mathbf{p} to obtain the 2D point \mathbf{x} and can be written as

$$\mathbf{x} = [I_{2 \times 2} | \mathbf{0}] \mathbf{p}$$

- Using homogeneous (projective) coordinates we can write this as

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

i.e. we drop the z component but keep the w component

Orthography and Para-Perspective

- **Orthography** is an approximate model for long focal length (telephoto) lenses and objects whose depth is *shallow* relative to their distance to the camera

Orthography and Para-Perspective

- **Orthography** is an approximate model for long focal length (telephoto) lenses and objects whose depth is *shallow* relative to their distance to the camera
- It is exact only for *telecentric* lenses

Orthography and Para-Perspective

- In practice, world coordinates need to be scaled to fit onto an image sensor

Orthography and Para-Perspective

- In practice, world coordinates need to be scaled to fit onto an image sensor
- For this reason, **scaled orthography** is more commonly used,

$$\mathbf{x} = [s/2 \times 2 | \mathbf{0}] \mathbf{p}$$

Orthography and Para-Perspective

- In practice, world coordinates need to be scaled to fit onto an image sensor
- For this reason, **scaled orthography** is more commonly used,

$$\mathbf{x} = [sI_{2 \times 2} | \mathbf{0}] \mathbf{p}$$

- 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

Orthography and Para-Perspective

- In practice, world coordinates need to be scaled to fit onto an image sensor
- For this reason, **scaled orthography** is more commonly used,

$$\mathbf{x} = [s/2 \times 2 | 0] \mathbf{p}$$

- 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 (b) or it can be different for objects that are being modeled independently (c)

Orthography and Para-Perspective

- A closely related projection model is **para-perspective**

Orthography and Para-Perspective

- A closely related projection model is **para-perspective**
- In this model, object points are again first projected onto a local reference parallel to the image plane

Orthography and Para-Perspective

- A closely related projection model is **para-perspective**
- In this model, object points are again first projected onto a local reference parallel to the image plane
- However, rather than being projected orthogonally to this plane they are projected *parallel* to the line of sight to the object center (d)

Orthography and Para-Perspective

- A closely related projection model is **para-perspective**
- In this model, object points are again first projected onto a local reference plane parallel to the image plane
- However, rather than being projected orthogonally to this plane they are projected *parallel* to the line of sight to the object center (d)
- This is followed by the usual projection onto the final image plane which amounts to a scaling

Orthography and Para-Perspective

- The combination of these two projections is therefore *affine* and can be written as

$$\tilde{\mathbf{x}} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tilde{\mathbf{p}}$$

Orthography and Para-Perspective

- The combination of these two projections is therefore *affine* and can be written as

$$\tilde{\mathbf{x}} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tilde{\mathbf{p}}$$

- Note that parallel lines in 3D remain parallel after projection (b-d)

Orthography and Para-Perspective

- The combination of these two projections is therefore *affine* and can be written as

$$\tilde{\mathbf{x}} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tilde{\mathbf{p}}$$

- Note that parallel lines in 3D remain parallel after projection (b-d)
- Para-perspective provides a more accurate projection model than scaled orthography without incurring the added complexity of per-pixel perspective division

Perspective

- The most commonly used projection in computer graphics and computer vision is true 3D **perspective** (e)

Perspective

- The most commonly used projection in computer graphics and computer vision is true 3D **perspective** (e)
- Points are projected onto the image plane by dividing them by their z component

Perspective

- The most commonly used projection in computer graphics and computer vision is true 3D **perspective** (e)
- Points are projected onto the image plane by dividing them by their z component
- Using homogeneous coordinates, this can be written as

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

Perspective

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

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

i.e. we can drop the w component of \mathbf{p}

Perspective

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

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

i.e. we can drop the w component of \mathbf{p}

- Thus, after projection it is not possible to recover the *distance* of the 3D point from the image

Summary

- We've introduced the basic 2D and 3D primitives used in computer vision

Summary

- We've introduced the basic 2D and 3D primitives used in computer vision
- We have also described how 3D features are projected onto 2D features

Summary

- We've introduced the basic 2D and 3D primitives used in computer vision
- We have also described how 3D features are projected onto 2D features
- Next, we will study geometric properties that are invariant w.r.t projective transformations (i.e. projective geometry)