3D Rotation and 3D Euclidean Transformation Formalisms

Computational Photography

CSE 291

Lecture 10

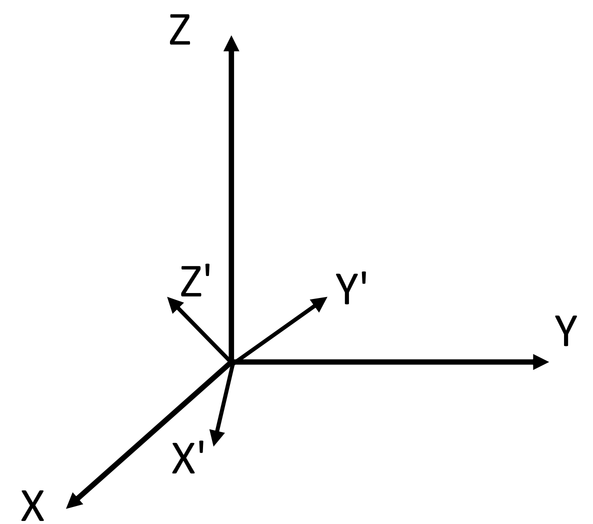
Announcements

- Assignment 4 is due today, 11:59 PM
- Initial project proposal is due May 9, 11:59 PM
- Revised project proposal is due May 16, 11:59 PM
- Draft project report due May 30, 11:59 PM
- Final project report due June 7, 11:59 PM

3D rotation

- Rigid body rotation in 3 dimensions

3D rotations map between
coordinate frames



3D rotation

- Preserves
  - Origin (i.e., rotates about origin)
  - Euclidean distance
  - Relative orientation
- Special orthogonal group in 3 dimensions, SO(3)
  - Multiple representations

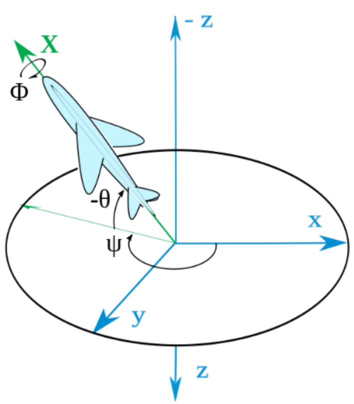
3D rotation

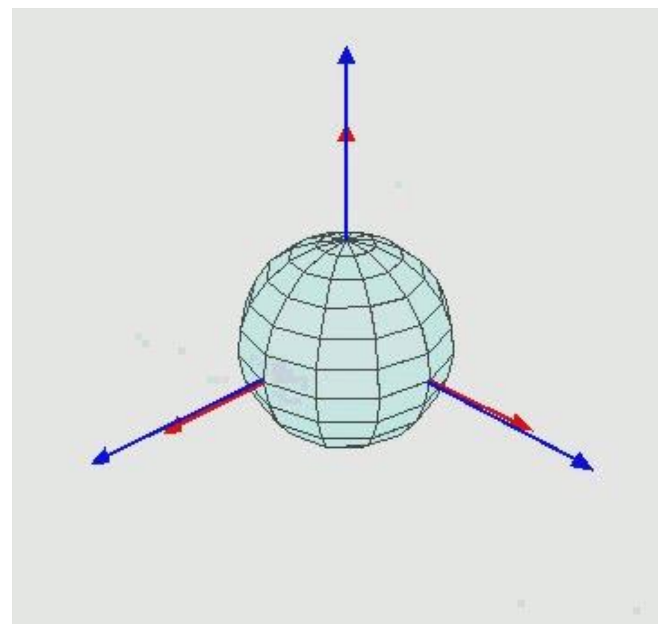
- Formalisms and example uses
  - Euler angles: platform or gimbal orientation (e.g., yaw-pitch-roll)
  - Angle-axis (Euler axis and angle): nonlinear optimization, robotics
  - Quaternion: many compositions of rotations (e.g., game engines)
  - Rotation matrix: everywhere else (and the above)

3D rotation, Euler angles

- 3 parameters (3 angles)
- A sequence of 3 elemental rotations
- 12 possible sequences

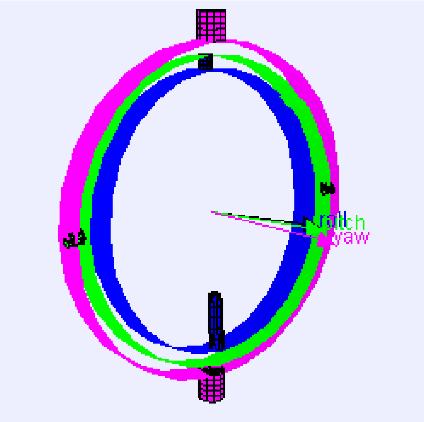
X-Y-X	Y-X-Y	Z-X-Y	
X-Y-Z	Y-X-Z	Z-X-Z	Euler Angles
X-Z-X	Y-Z-X	Z-Y-X	Tait-Bryan Angles
X-Z-Y	Y-Z-Y	Z-Y-Z	



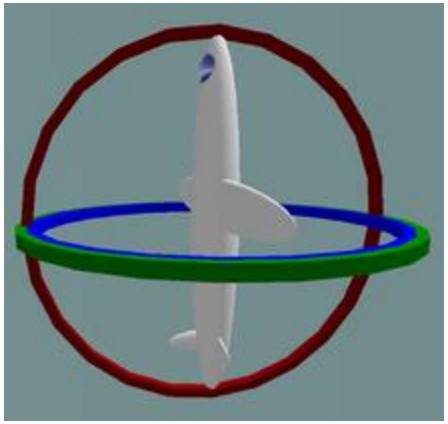


3D rotation, Euler angles

- Gimbal lock
  - Two of the three gimbals are in the same plane
  - One degree of freedom is lost
    - “Locked” into 2D rotation



No gimbal lock

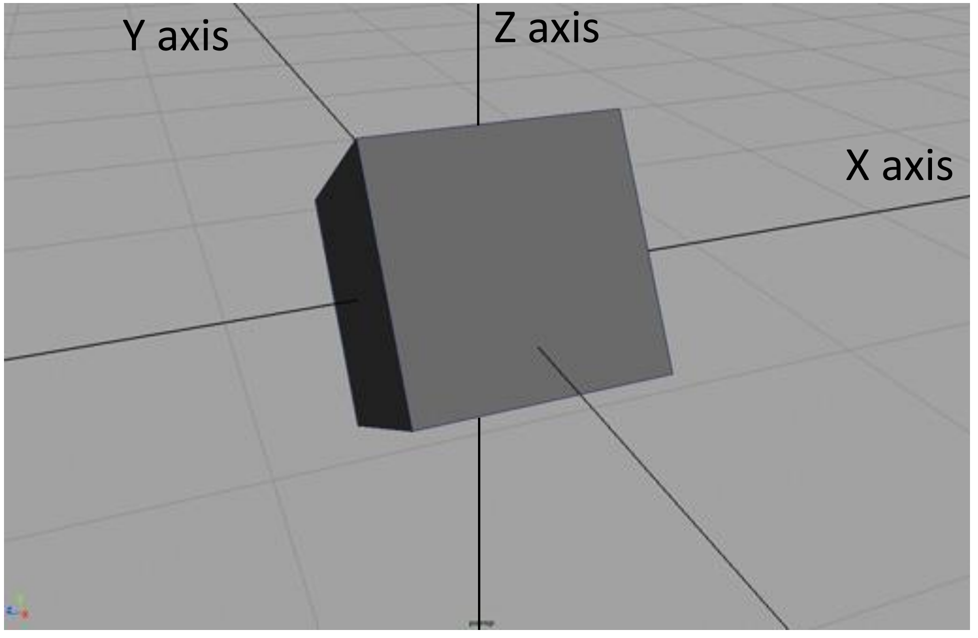


Gimbal lock

3D rotation, rotation matrix
3D rotation about X-axis

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

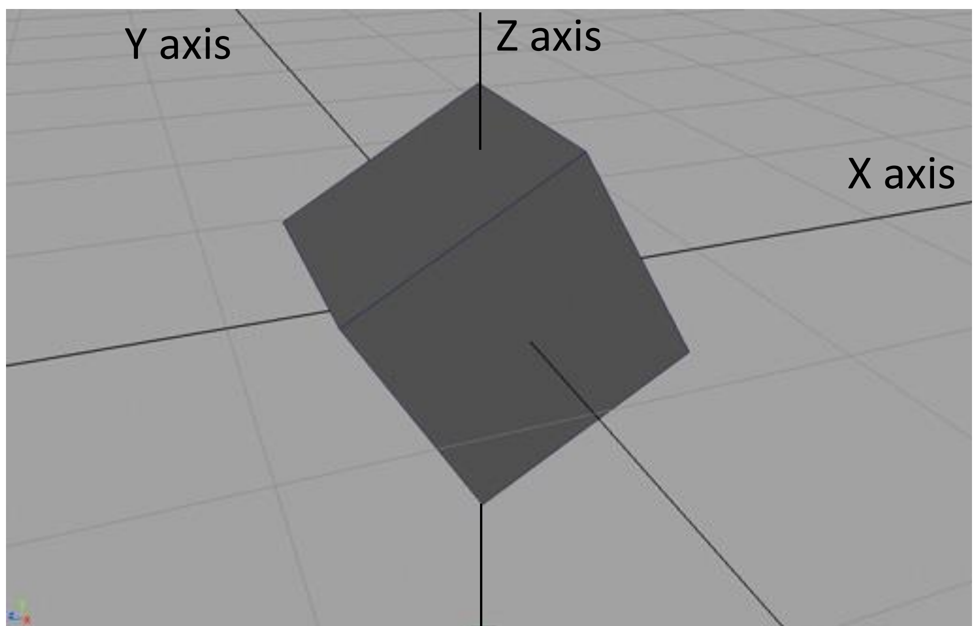
$$\mathbf{X}' = \mathrm{R}_X(\alpha)\mathbf{X}$$



3D rotation, rotation matrix
3D rotation about Y-axis

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

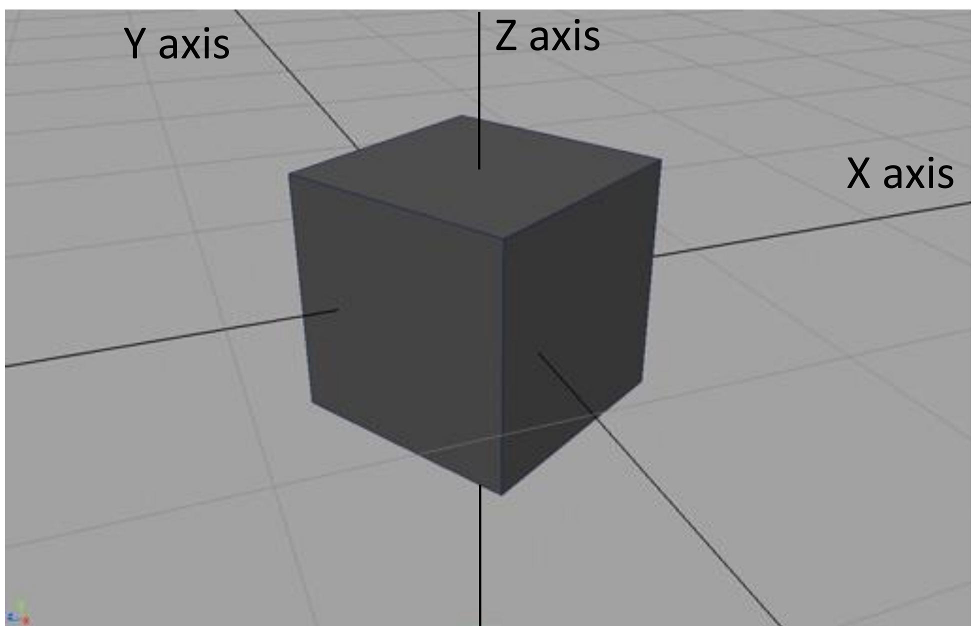
$$\mathbf{X}' = \mathrm{R}_Y(\beta)\mathbf{X}$$



3D rotation, rotation matrix
3D rotation about Z-axis

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

$$\mathbf{X}' = \mathrm{R}_Z(\gamma)\mathbf{X}$$



3D rotation, rotation matrix

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

$$\mathbf{X}' = \mathrm{R}\mathbf{X}$$

$$\text{where } \mathrm{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

3x3 special orthogonal matrix

$$\mathrm{R} = \mathrm{R}_Z(\gamma)\mathrm{R}_Y(\beta)\mathrm{R}_X(\alpha)$$

Composition of rotations

Rotation matrix

- A rotation matrix is a special orthogonal matrix
  - Properties of special orthogonal matrices

$$\mathrm{R}^\top \mathrm{R} = \mathrm{R}\mathrm{R}^\top = \mathrm{I}$$

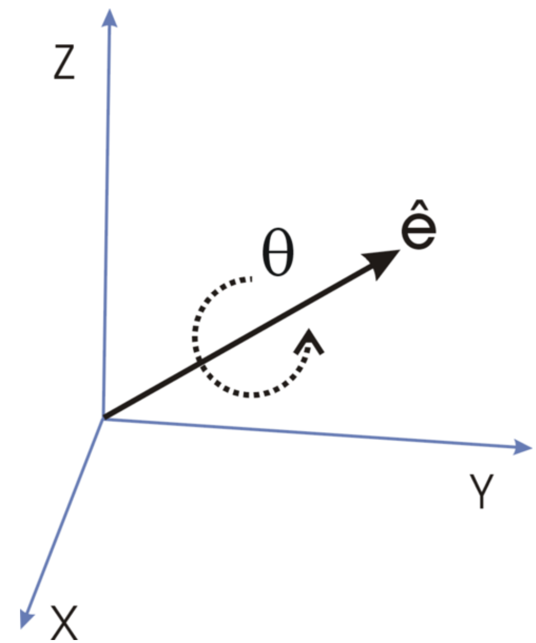
$$\det(\mathrm{R}) = +1$$

$$\mathrm{R}^\top = \mathrm{R}^{-1}$$

The inverse of a special orthogonal matrix is also a special orthogonal matrix

3D rotation, angle-axis representation

- Euler's rotation theorem
  - Any rotation of a rigid body in 3D is equivalent to a pure rotation about a single fixed axis
- 3 parameters, 3 degrees of freedom
  - Axis of rotation defined by a unit 3-vector (2 degrees of freedom) multiplied by angle of rotation about the axis (1 degree of freedom)



$$\theta = \|\boldsymbol{\omega}\|$$

$$\hat{\mathrm{e}} = \frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|}$$

$$(\omega_1, \omega_2, \omega_3)^\top = \theta(\hat{e}_1, \hat{e}_2, \hat{e}_3)^\top$$

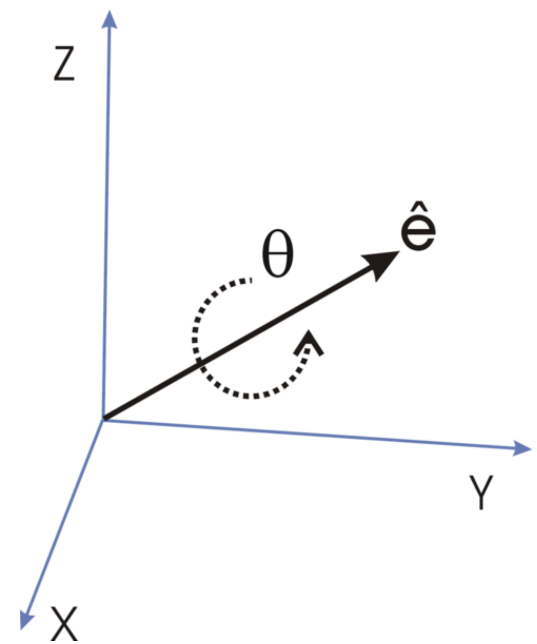
$$\boldsymbol{\omega} = \theta\hat{\mathrm{e}}$$

Angle-axis coordinates

where $\boldsymbol{\omega} = (\omega_1, \omega_2, \omega_3)^\top$

3D rotation, angle-axis representation

- Invert rotation by negating angle-axis coordinates
- Interpolation between 3D rotations
  - Spherical linear interpolation (Slerp)
    - Interpolate rotation through the angle about the axis



Matrix logarithm of 3x3 special orthogonal matrix

- Rotation matrix to angle-axis representation

3x3 skew-symmetric matrix

3x3 special orthogonal matrix

$$\hat{\boldsymbol{\omega}} = \log(\mathrm{R}) \quad \text{where } \hat{\boldsymbol{\omega}} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \qquad \mathrm{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

$$\boldsymbol{\omega} = (\omega_1, \omega_2, \omega_3)^\top$$

$$\mathrm{R} \mapsto \log(\mathrm{R})$$

$\mathrm{SO}(3) \mapsto \mathrm{so}(3)$ called little so(3)

SO(3) is a Lie group
so(3) is its Lie algebra

Matrix exponent of 3x3 skew-symmetric matrix

- Angle-axis representation to rotation matrix

$$\boldsymbol{\omega} = (\omega_1, \omega_2, \omega_3)^\top$$

3x3 special orthogonal matrix

3x3 skew-symmetric matrix

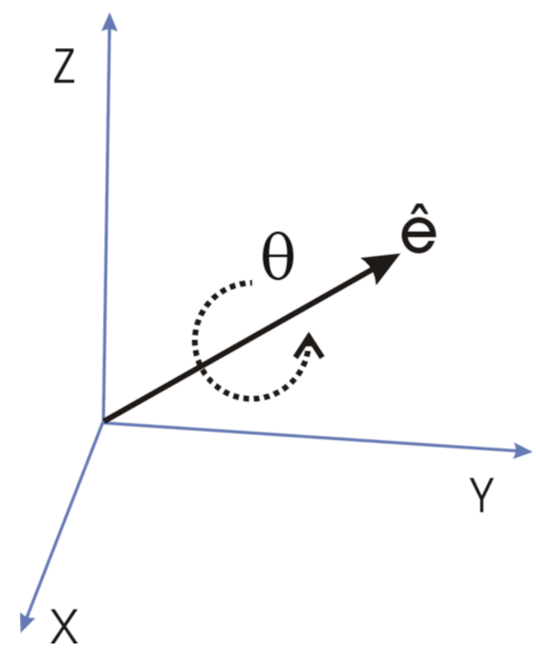
$$\mathrm{R} = \exp(\hat{\boldsymbol{\omega}}) \quad \text{where } \mathrm{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \qquad \hat{\boldsymbol{\omega}} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}$$

$$\hat{\boldsymbol{\omega}} \mapsto \exp(\hat{\boldsymbol{\omega}})$$

$$\mathrm{so}(3) \mapsto \mathrm{SO}(3)$$

3D rotation, quaternion representation

- Euler's rotation theorem
  - Any rotation of a rigid body in 3D is equivalent to a pure rotation about a single fixed axis
- 4 parameters, 3 degrees of freedom
  - Homogeneous vector (defined up to nonzero scale)
    - Real part and imaginary part



$$\mathbf{q} = \left(\cos\left(\frac{\theta}{2}\right), \sin\left(\frac{\theta}{2}\right) \hat{\mathrm{e}}^\top \right)^\top$$

3D rotation, quaternion representation

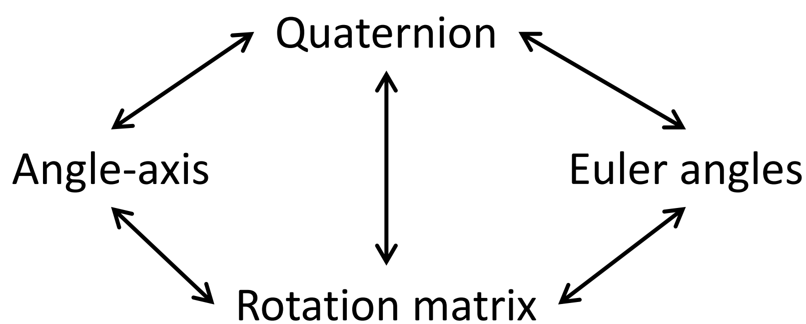
- 4 parameters (real parts; *a*, *b*, *c*, and *d*)
  - Homogeneous 4-vector (i.e., defined up to scale)
- $a + bi + cj + dk$, where
  - $i^2 = j^2 = k^2 = ijk = -1$
  - $ij = -ji = k$
  - $jk = -kj = i$
  - $ki = -ik = j$

Hamilton's rules

- Real and imaginary parts
- Commonly a unit 4-vector (called a versor; 3 degrees of freedom), but not necessary
- Compose rotations using the Hamilton product (not commutative)
- Invert rotation using complex conjugate

3D rotation

- Conversions between formalisms



Rotation matrix to angle-axis

```
template<typename T>
const AngleAxis3D<T> rotationMatrixToAngleAxis( const SmallMatrix<T, 3, 3>& R )
{
    using std::atan2;

    SmallMatrix<T, 3, 1> vhat;
    vhat[0] = R[2][1] - R[1][2];
    vhat[1] = R[0][2] - R[2][0];
    vhat[2] = R[1][0] - R[0][1];
    const SmallMatrix<T, 3, 3> RminusI = R - identity<T, 3, 3>();
    const SmallMatrix<T, 3, 1> s = svd( RminusI );
    const T tolerance = 3 * epsilon( s[0] );
    if ( tolerance >= s[1] )
    {
        // Zero or near zero rotation
        return AngleAxis3D<T>( T{ 0.5 } * vhat );
    }
    else
    {
        // (R - I) * v = 0
        // Null vector of (R - I)
        const SmallMatrix<T, 3, 1> v = nullVector( RminusI );

        const T sin_theta = ( trans( v ) * vhat ) / 2;
        const T cos_theta = ( trace( R ) - 1 ) / 2;
        const T theta = atan2( sin_theta, cos_theta );

        return AngleAxis3D<T>( theta * v );
    }
}
```

Angle-axis to rotation matrix

```cpp
template<typename T>
inline const SmallMatrix<T, 3, 3> angleAxisToRotationMatrix(
    const AngleAxis3D<T>& omega )
{
    using std::cos;

    const SmallMatrix<T, 3, 3> I = identity<T, 3, 3>();
    const SmallMatrix<T, 3, 1> omega_vec = omega.asVector();
    const T theta = omega.angle();
    const T cos_theta = cos( theta );
    const T temp = ( 1 - cos_theta ) / ( theta * theta );
    if ( !isFinite( temp ) )
    {
        // Zero or near zero rotation
        return I + skewSym( omega_vec );
    }
    else
    {
        return cos_theta * I + sinc( theta ) * skewSym( omega_vec ) +
            temp * omega_vec * trans( omega_vec );
    }
}
```

Rotation matrix to quaternion

```
template<typename T>
inline const Quaternion3D<T> rotationMatrixToQuaternion(
    const SmallMatrix<T, 3, 3>& R )
{
    using std::sqrt;

    const T trR = trace( R );
    if ( 0 < trR )
    {
        const T a = sqrt( trR + 1 );
        const T b = 1 / ( 2 * a );
        return Quaternion3D<T>( a / 2,
                                ( R[2][1] - R[1][2] ) * b,
                                ( R[0][2] - R[2][0] ) * b,
                                ( R[1][0] - R[0][1] ) * b );
    }
    // else
    int i = 0;
    if ( R[1][1] > R[0][0] ) i = 1;
    if ( R[2][2] > R[i][i] ) i = 2;
    const int j = ( i + 1 ) % 3;
    const int k = ( j + 1 ) % 3;

    const T a = sqrt( std::max<T>( 0, R[i][i] - R[j][j] - R[k][k] + 1 ) );
    const T b = 1 / ( 2 * a );
    Quaternion3D<T> q;
    q[0]     = ( R[k][j] - R[j][k] ) * b;
    q[i + 1] = a / 2;
    q[j + 1] = ( R[j][i] + R[i][j] ) * b;
    q[k + 1] = ( R[k][i] + R[i][k] ) * b;
    return q;
}
```

Quaternion to rotation matrix

```
template<typename T>
inline const SmallMatrix<T, 3, 3> quaternionToRotationMatrix(
    const Quaternion3D<T>& q )
{
    const T& a = q.a();
    const T& b = q.b();
    const T& c = q.c();
    const T& d = q.d();

    const T aa = a * a;
    const T ab = a * b;
    const T ac = a * c;
    const T ad = a * d;
    const T bb = b * b;
    const T bc = b * c;
    const T bd = b * d;
    const T cc = c * c;
    const T cd = c * d;
    const T dd = d * d;

    const T s = 1 / ( aa + bb + cc + dd ); // = 1 / ||q||^2
    const T two_s = 2 * s;                 // = 2 / ||q||^2

    SmallMatrix<T, 3, 3> R;
    // Row 1
    R[0][0] = s * ( aa + bb - cc - dd );
    R[0][1] = two_s * ( bc - ad );
    R[0][2] = two_s * ( bd + ac );
    // Row 2
    R[1][0] = two_s * ( bc + ad );
    R[1][1] = s * ( aa - bb + cc - dd );
    R[1][2] = two_s * ( cd - ab );
    // Row 3
    R[2][0] = two_s * ( bd - ac );
    R[2][1] = two_s * ( cd + ab );
    R[2][2] = s * ( aa - bb - cc + dd );
    return R;
}
```

Angle-axis to quaternion

```
template<typename T>
inline const Quaternion3D<T> angleAxisToQuaternion(
    const AngleAxis3D<T>& omega )
{
    using std::cos;

    // q = (a, b^T)^T
    const SmallMatrix<T, 3, 1> omega_vec = omega.asVector();
    const T theta_div_2 = norm( omega_vec ) / 2; // 0 <= theta_div_2 <= pi/2
    const T a = cos( theta_div_2 );              // 1 >= a           >= 0
    const SmallMatrix<T, 3, 1> b = sinc( theta_div_2 ) / 2 * omega_vec;
    return Quaternion3D<T>( a, b[0], b[1], b[2] );
}
```

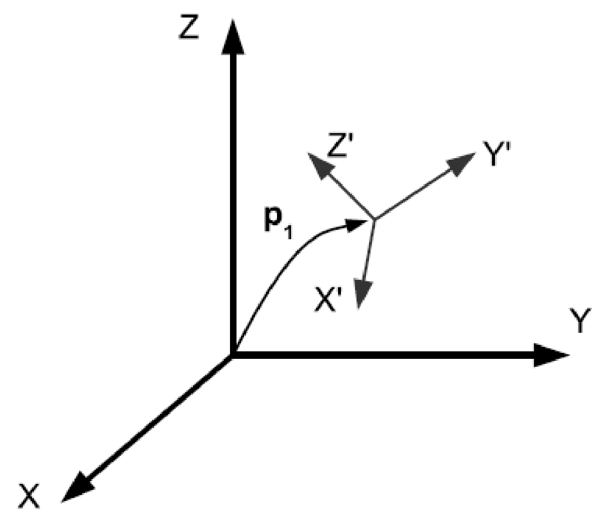
Quaternion to angle-axis

```
template<typename T>
inline const AngleAxis3D<T> quaternionToAngleAxis( const Quaternion3D<T>& q )
{
    using std::acos;

    // Scale quaternion such that it is a unit quaternion with nonnegative
    // first element
    SmallMatrix<T, 4, 1> q_vec( q.ptrData(), q.stride2() );
    q_vec /= copySign( norm( q_vec ), q_vec[0] );

    return AngleAxis3D<T>( 2 / sinc( acos( q_vec[0] ) ) *
        SmallMatrix<T, 3, 1>( &q_vec[1], q_vec.stride2() ) );
}
```

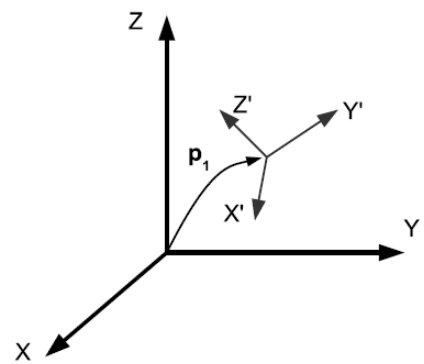
3D Euclidean transformation

- Rigid body transformation in 3 dimensions
- Embodies 3D rotation and 3D translation
- Also called a *pose*

3D Euclidean transformations map between coordinate frames



3D Euclidean transformation



- Preserves
  - Euclidean distance
  - Relative orientation
- Special Euclidean group in 3 dimensions, SE(3)
  - 3D rotation
    - Special orthogonal group in 3 dimensions, SO(3)
    - Multiple representations
  - 3D translation
    - Single representation, a 3-vector

3D Euclidean transformation

- Formalisms and example uses
  - Euler angles and position: platform position and orientation
  - Twist: nonlinear optimization, robotics
  - Dual quaternion: many compositions of transformations (e.g., game engines, computer animation)
  - Homogeneous transformation matrix: everywhere else (and the above)

3D Euclidean transformation, homogeneous transformation matrix

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{X}' \\ 1 \end{bmatrix} = \begin{bmatrix} \mathtt{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{X}' \\ 1 \end{bmatrix} = \mathtt{H}_\mathrm{E} \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix}$$

3x3 special orthogonal matrix

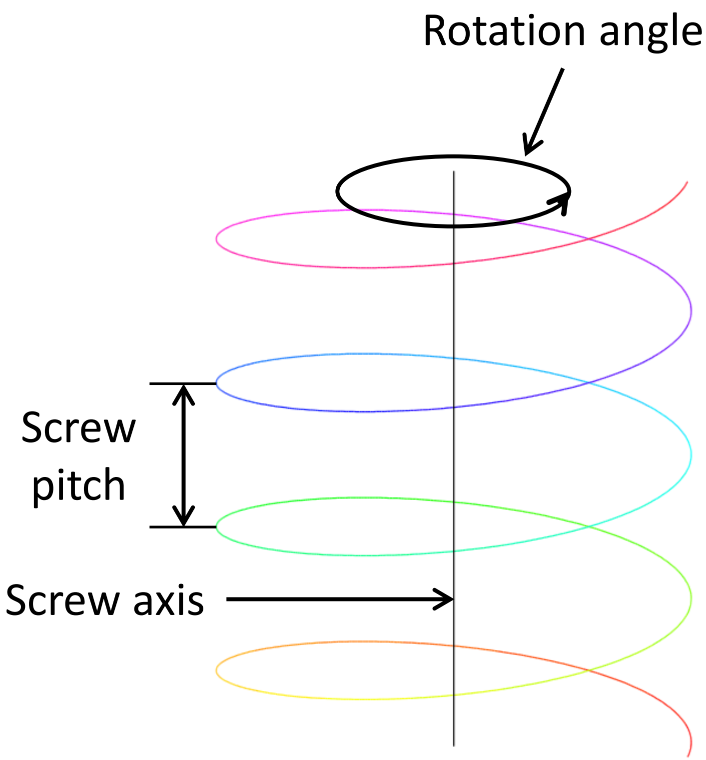
$$\text{where } \mathtt{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \text{ and } \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$$

$$\mathtt{H}_\mathrm{E} = \begin{bmatrix} \mathtt{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix}$$

4x4 homogeneous transformation matrix

3D Euclidean transformation, screw theory

- Chasles' theorem
  - Each Euclidean displacement in three-dimensional space has a screw axis, and the displacement can be decomposed into a rotation about and a slide along this screw axis
- Screw parameters
- Twist representation



Matrix logarithm of 4x4 homogeneous transformation matrix

- Homogeneous transformation matrix to twist representation

$$\hat{\xi} = \log(\mathrm{H_E}) \qquad \text{where } \hat{\xi} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 & v_1 \\ \omega_3 & 0 & -\omega_1 & v_2 \\ -\omega_2 & \omega_1 & 0 & v_3 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \mathrm{H_E} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\hat{\xi} = \begin{bmatrix} \hat{\omega} & \mathbf{v} \\ \mathbf{0}^\top & 0 \end{bmatrix}$$

$$\xi = (\omega_1, \omega_2, \omega_3, v_1, v_2, v_3)^\top$$
$$\xi = (\boldsymbol{\omega}^\top, \mathbf{v}^\top)^\top$$

Twist coordinates

$$\mathrm{H_E} \mapsto \log(\mathrm{H_E})$$
$$\mathrm{SE}(3) \mapsto \mathrm{se}(3)$$

called little se(3)

SE(3) is a Lie group
se(3) is its Lie algebra

Matrix exponent of 4x4 twist matrix

- Twist representation to homogeneous transformation matrix

$$\boldsymbol{\xi} = (\omega_1, \omega_2, \omega_3, v_1, v_2, v_3)^\top$$

$$\boldsymbol{\xi} = (\boldsymbol{\omega}^\top, \mathbf{v}^\top)^\top$$

$$\mathrm{H_E} = \exp(\hat{\boldsymbol{\xi}}) \quad \text{where } \mathrm{H_E} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \hat{\boldsymbol{\xi}} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 & v_1 \\ \omega_3 & 0 & -\omega_1 & v_2 \\ -\omega_2 & \omega_1 & 0 & v_3 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

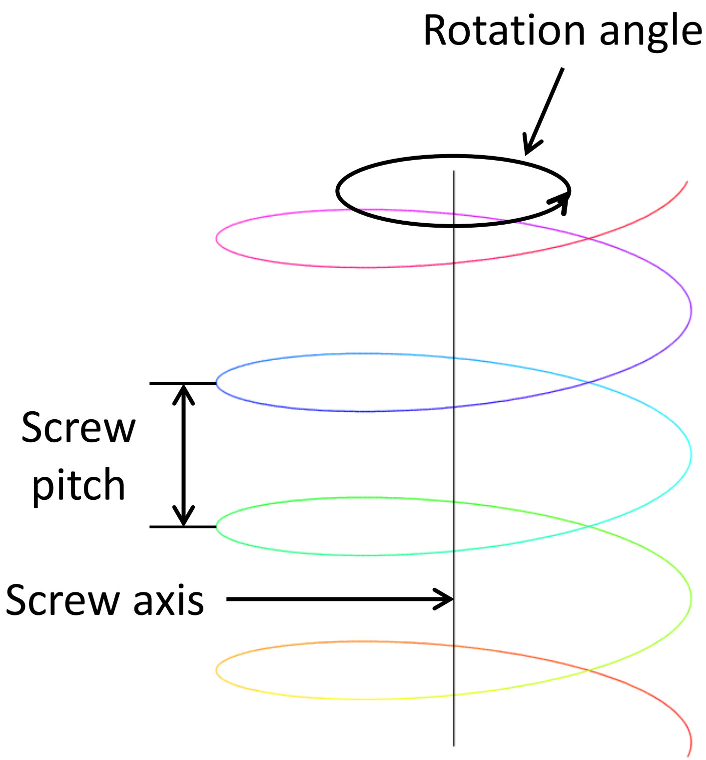
$$\mathrm{H_E} = \begin{bmatrix} \mathrm{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \qquad \hat{\boldsymbol{\xi}} = \begin{bmatrix} \hat{\boldsymbol{\omega}} & \mathbf{v} \\ \mathbf{0}^\top & 0 \end{bmatrix}$$

$$\hat{\boldsymbol{\xi}} \mapsto \exp(\hat{\boldsymbol{\xi}})$$

$$\mathrm{se}(3) \mapsto \mathrm{SE}(3)$$

3D Euclidean transformation, twist representation

- Invert Euclidean transformation by negating twist coordinates
- Interpolation between 3D Euclidean transformations
  - Screw linear interpolation
    - Interpolate rotation through the angle about and slide along the axis



3D Euclidean transformation, dual quaternion representation

- Dual number $x = x_{\text{real}} + \epsilon x_{\text{dual}}, \text{ where } \epsilon^2 = 0$
  - Real part and dual part
  - Similar to complex numbers (real part and imaginary part)
- Dual quaternion $\mathrm{q} = \mathrm{q}_{\text{real}} + \epsilon \mathrm{q}_{\text{dual}}, \text{ where } \epsilon^2 = 0$
  - Real part embodies rotation
  - Dual part

$$\mathrm{q}_{\text{dual}} = \frac{1}{2}\mathrm{q}_{\mathrm{t}}\mathrm{q}_{\text{real}}, \text{ where } \mathrm{q}_{\mathrm{t}} = (0, \mathrm{t}^\top)^\top$$

Multiply (compose transformations) using Hamilton product

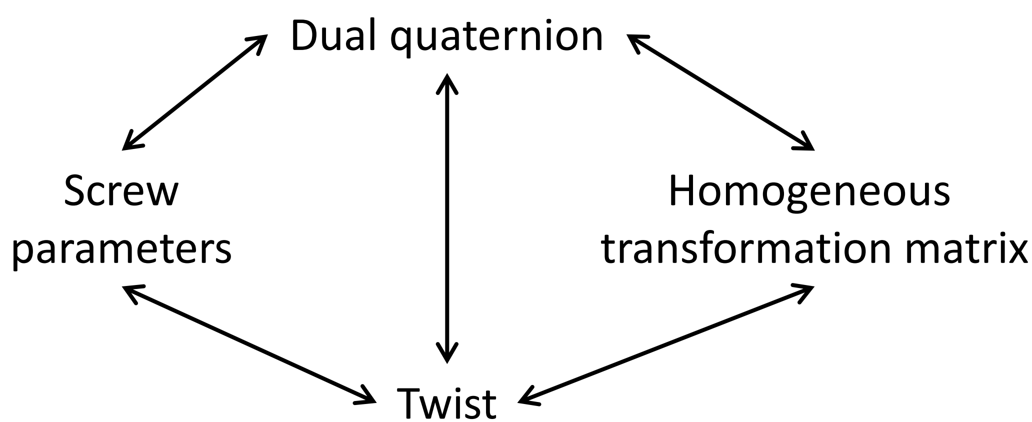
3D Euclidean transformation, dual quaternion representation

- Compose Euclidean transformations using dual number multiplication and the Hamilton product

```
template<typename T>
inline const DualQuaternion3D<T> operator*( const DualQuaternion3D<T>& q1,
    const DualQuaternion3D<T>& q2 )
{
    return DualQuaternion3D<T>( q1.real() * q2.real(),
        q1.real() * q2.dual() + q1.dual() * q2.real() );
}
```

- Invert Euclidean transformation using complex conjugate of real and dual parts

Conversion between 3D Euclidean transformation formalisms



Analogs

3D rotation formalism	$\longleftrightarrow$	**3D Euclidean transformation formalism**
Rotation matrix	$\longleftrightarrow$	Homogeneous transformation matrix
Angle-axis	$\longleftrightarrow$	Twist
Quaternion	$\longleftrightarrow$	Dual Quaternion

Summary

- 3D Euclidean transformation formalisms are analogous to 3D rotation formalisms
- Elegant mathematical relationship between the different formalisms
- Advice
  – Use the representation that is best suited to the application
  – Do not perform calculations using Euler angles
    - Only use for storage, data transfer, or user interface