

Documentation for Sphere.h and Sphere.c

Steven Andrews, © 2005

See the document “LibDoc” for general information about this and other libraries.

Header file

```
#ifndef __Sphere_h
#define __Sphere_h

/*
Cart = Cartesian coordinates (x, y, z)
Sc = Spherical coordinates (r, theta, phi)
Dcm = Direction cosine matrix (A00, A01, A02, A10, A11, A12, A20, A21, A22)
Eax = Euler angles x-convention (theta, phi, psi)
Eay = Euler angles y-convention (theta, phi, chi)
Ep = Euler parameters (e0, e1, e2, e3)
Xyz = xyz- or ypr- or Tait-Bryan convention (yaw, pitch, roll): rotate on z,
then y, then x
*/

void Sph_Cart2Sc(double *Cart, double *Sc);
void Sph_Sc2Cart(double *Sc, double *Cart);
void Sph_Eay2Ep(double *Eay, double *Ep);
void Sph_Xyz2Xyz(double *Xyz1, double *Xyz2);

void Sph_Eax2Dcm(double *Eax, double *Dcm);
void Sph_Eay2Dcm(double *Eay, double *Dcm);
void Sph_Xyz2Dcm(double *Xyz, double *Dcm);
void Sph_Xyz2Dcmt(double *Xyz, double *Dcmt);
void Sph_Dcm2Xyz(double *Dcm, double *Xyz);
void Sph_Dcm2Dcm(double *Dcm1, double *Dcm2);
void Sph_Dcm2Dcmt(double *Dcm1, double *Dcm2);

void Sph_DcmxDcm(double *Dcm1, double *Dcm2, double *Dcm3);
void Sph_DcmxDcmt(double *Dcm1, double *Dcmt, double *Dcm3);
void Sph_DcmtxDcm(double *Dcmt, double *Dcm2, double *Dcm3);

void Sph_One2Dcm(double *Dcm);
void Sph_Xyz2Xyzr(double *Xyz, double *Xyzr);
void Sph_Dcm2Dcmr(double *Dcm, double *Dcmr);
void Sph_Rot2Dcm(char axis, double angle, double *Dcm);
void Sph_Newz2Dcm(double *Newz, double psi, double *Dcm);

void Sph_DcmtxUnit(double *Dcmt, char unit, double *vect, double *add, double
mult);

#endif
```

Includes: <stdio.h>, "random.h", "Sphere.h"

Example program: MinWalk.c

History: Written 2/05. Documented 7/05. Added Eax2Dcm, Eay2Dcm, Newz2Dcm on 10/24/07. Added Sph_DcmtxUnit 5/55/12. Added Sph_Xyz2Dcmt 5/28/12.

Description

This is a collection of routines for manipulating rotational coordinates using a variety of conventions. Note that some coordinates are for vectors (e.g. spherical coordinates) whereas others are for transformations (e.g. Euler angles). Most of the math here is described in Goldstein.

If two different function arguments are the same size, such as two vectors or two matrices, then they are always allowed to point to the same memory. For example to invert the direction cosine matrix dcm in-place, the function call is Sph_Dcm2Dcmt(dcm,dcm). While input angles are never required to be clamped to fixed domains, the output angle ranges are always clamped, as listed below. Input direction cosine matrices are assumed to be valid and are not checked. The following descriptions of the conventions uses **A** as a direction cosine matrix and the matrix definitions:

$$\mathbf{X}(a) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos a & \sin a \\ 0 & -\sin a & \cos a \end{bmatrix} \quad \mathbf{Y}(a) = \begin{bmatrix} \cos a & 0 & -\sin a \\ 0 & 1 & 0 \\ \sin a & 0 & \cos a \end{bmatrix} \quad \mathbf{Z}(a) = \begin{bmatrix} \cos a & \sin a & 0 \\ -\sin a & \cos a & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Cartesian coordinates (Cart)

Vector is $[x,y,z]$, all of which are on $(-\infty,\infty)$.

Spherical coordinates (Sc)

Vector is $[r,\theta,\phi]$. r is on $[0,\infty)$, θ is on $[0,\pi]$, and ϕ is on $[0,2\pi)$.

Direction cosine matrix (Dcm)

Matrix is given as a 9 element array, which lists the matrix row by row. This is useful for all coordinate transformations and is not associated with any particular convention.

Direction cosine matrix transpose (Dcmt)

This is entered as a normal, non-transposed, direction cosine matrix. However, it is interpreted as a transposed direction cosine matrix in the code.

Euler angle x-convention (Eax)

Vector is $[\theta,\phi,\psi]$. θ is on $[0,\pi]$, ϕ is on $[0,2\pi)$, ψ is on $[0,2\pi)$. $\mathbf{A} = \mathbf{Z}(\psi)\mathbf{X}(\theta)\mathbf{Z}(\phi)$.

Euler angle y-convention (Eay)

Vector is $[\theta,\phi,\chi]$. θ is on $[0,\pi]$, ϕ is on $[0,2\pi)$, χ is on $[0,2\pi)$. $\mathbf{A} = \mathbf{Z}(\chi)\mathbf{Y}(\theta)\mathbf{Z}(\phi)$.

Euler parameters (Eap)

Vector is $[e_0,e_1,e_2,e_3]$.

Yaw-pitch-roll (Xyz)

Vector is $[\phi, \theta, \psi]$. All are on $[-\pi, \pi)$. $\mathbf{A} = \mathbf{X}(\psi)\mathbf{Y}(\theta)\mathbf{Z}(\phi)$.

Code documentation

Typical parameter names

cf	$\cos(\phi)$
cq	$\cos(\theta)$
cy	$\cos(\psi)$ or $\cos(\chi)$
sf	$\sin(\phi)$
sq	$\sin(\theta)$
sy	$\sin(\psi)$ or $\sin(\chi)$

Internal macros and global variables

```
#define PI 3.14159265358979323846  
     $\pi$ .
```

```
double Work[9], Work2[9];  
    Scratch-space.
```

Externally accessible functions

```
void Sph_Cart2Sc(double *Cart, double *Sc);  
    Converts Cartesian coordinates to spherical coordinates.
```

```
void Sph_Sc2Cart(double *Sc, double *Cart);  
    Converts spherical coordinates to Cartesian coordinates.
```

```
void Sph_Eay2Ep(double *Eay, double *Ep);  
    Converts Euler angle y-convention transformation to Euler parameters. Equations  
    from Goldstein p. 608.
```

```
void Sph_Xyz2Xyz(double *Xyz1, double *Xyz2);  
    Copies yaw-pitch-roll vector Xyz1 to Xyz2, and clamps angles in the process.
```

```
void Sph_Eax2Dcm(double *Eax, double *Dcm);  
    Calculates direction cosine matrix from Euler angle x-convention vector. Equations  
    from Wolfram MathWorld.
```

```
void Sph_Eay2Dcm(double *Eay, double *Dcm);
```

Calculates direction cosine matrix from Euler angle y-convention vector. Equations from Wolfram MathWorld.

void Sph_Xyz2Dcm(double *Xyz, double *Dcm);

Calculates direction cosine matrix from yaw-pitch-roll vector. Equations from Goldstein p. 609. $\mathbf{A} = \mathbf{X}(\psi)\mathbf{Y}(\theta)\mathbf{Z}(\phi)$.

void Sph_Xyz2Dcmt(double *Xyz, double *Dcmt);

Calculates transposed direction cosine matrix from yaw-pitch-roll vector. This is just Sph_Xyz2Dcm, but for a transposed result.

void Sph_Dcm2Xyz(double *Dcm, double *Xyz);

Calculates yaw-pitch-roll vector from a direction cosine matrix. Equations are derived from Goldstein p. 609.

void Sph_Dcm2Dcm(double *Dcm1, double *Dcm2);

Copies direction cosine matrix Dcm1 to a new one in Dcm2.

void Sph_Dcm2Dcmt(double *Dcm1, double *Dcm2);

Transposes direction cosine matrix Dcm1 to yield matrix inverse in Dcm2. $\mathbf{A}_2 = \mathbf{A}_1^{-1}$.

void Sph_DcmxDcm(double *Dcm1, double *Dcm2, double *Dcm3);

Matrix multiplies Dcm1 by Dcm2 and returns result in Dcm3. Note that the transformation is Dcm2 first, then Dcm1, which occurs in the new coordinate system. $\mathbf{A}_3 = \mathbf{A}_1\mathbf{A}_2$.

void Sph_DcmxDcmt(double *Dcm1, double *Dcmt, double *Dcm3);

Matrix multiplies Dcm1 by the transpose of Dcmt and returns result in Dcm3 (Dcmt is entered as an untransposed matrix). Essentially, this is a negative rotation of Dcmt followed by a positive rotation of Dcm1. $\mathbf{A}_3 = \mathbf{A}_1\mathbf{A}_2^{-1}$.

void Sph_DcmtxDcm(double *Dcmt, double *Dcm2, double *Dcm3);

Matrix multiplies the transpose of Dcmt by Dcm2 and returns the result in Dcm3 (Dcmt is entered as an untransposed matrix). Essentially, this is a positive rotation of Dcm2 followed by a negative rotation of Dcmt. $\mathbf{A}_3 = \mathbf{A}_1^{-1}\mathbf{A}_2$.

void Sph_One2Dcm(double *Dcm);

Returns the identity direction cosine matrix. $A_{ij} = \delta_{ij}$.

void Sph_Xyz2Xyzr(double *Xyz, double *Xyzr);

Converts the forwards-direction yaw-pitch-roll vector Xyz to a relative direction change, but for travel in the reverse direction. For example, suppose an airplane performs the direction change that corresponds to Xyz. If it then turns around, with the local z-vector as it was initially, but with both x- and y-vectors reversed (180° yaw), then it needs to execute rotation Xyzr to retrace its original track. $\mathbf{A} = \mathbf{Z}^{-1}(\phi)\mathbf{Y}(\theta)\mathbf{X}(\psi)$. Note that this reverses a relative direction change between two

vectors and does not reverse an absolute vector (the airplane traveling west being converted to it traveling east).

`void Sph_Dcm2Dcmr(double *Dcm, double *Dcmr);`

Converts an absolute dcm to a dcm in the reverse direction. This reverses the local x and y directions, while preserving the local z direction. This is unlike `Sph_Xyz2Xyzr` in that this is for absolute directions while that one was for relative directions. $\mathbf{A}_r = \mathbf{Z}(\pi)\mathbf{A}$.

`void Sph_Rot2Dcm(char axis, double angle, double *Dcm);`

Returns the direction cosine matrix that corresponds to rotation by angle angle about axis axis , where this latter parameter is the character 'x', 'y', or 'z' (or upper-case). $\mathbf{A} = \mathbf{X}(a)$ or $\mathbf{A} = \mathbf{Y}(a)$ or $\mathbf{A} = \mathbf{Z}(a)$.

`void Sph_Newz2Dcm(double *Newz, double psi, double *Dcm);`

Returns the direction cosine matrix that can be used to rotate the coordinate system such that the original z -axis will line up with the vector Newz . The length of Newz is irrelevant; it does not need to be normalized. Additional rotation about the new z -axis is entered with psi . This works as follows: Newz is converted to spherical coordinates θ and ϕ , then the d.c.m. is $\mathbf{A} = \mathbf{Z}(\psi - \phi) \mathbf{X}(\theta) \mathbf{Z}(\phi)$, which is transposed to yield the active matrix.

`void Sph_DcmtxUnit(double *Dcmt, char axis, double *vect, double *add, double mult);`

Multiplies the transpose of Dcmt (entered as a non-transposed direction cosine matrix) with the unit vector for axis axis (entered as 'x', 'y', or 'z', or upper case) and returns the result in the 3-dimensional vector vect . This multiplies the result by the scalar mult . If add is non-NULL, this adds add to vect before returning the result.