THUNDER

Generated by Doxygen 1.8.5

Thu Sep 13 2018 13:32:23

Contents

Chapter 1

File Index

1.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 2

File Documentation

2.1 include/Complex.h File Reference

Complex.h defines complex number related operations like +,-,$*$,/,|a|, $|a|^2$ and so on.

```
#include <gsl/gsl_complex.h>
#include <gsl/gsl_complex_math.h>
#include <math.h>
#include "Config.h"
#include "Precision.h"
#include "Typedef.h"
```

Functions

- Complex COMPLEX_POLAR (const RFLOAT phi)
- Complex CONJUGATE (const Complex &a)

  Get the conjugate result based on value a.
- RFLOAT ABS (const Complex &a)

  Calculate the $|a|$ of complex number a.
- RFLOAT **ABS2** (const Complex &a)
- Complex COMPLEX (RFLOAT a, RFLOAT b)

  Construct a complex number with a as real part and b as image part.
- RFLOAT REAL (const Complex &a)

  Get the real part of the complex number a.
- RFLOAT IMAG (const Complex &a)

  Get the image part of the complex number a.
- RFLOAT gsl_real_imag_sum (const Complex &a)

  Calcuate the sum of the complex number a's real part and image part.
- Complex operator+ (const Complex &a, const Complex &b)

  Implement the add operation between two complex numbers, e.g. c = a + b, where a and b are complex numbers.
- Complex operator- (const Complex &a, const Complex &b)

  Implement the sub operation between two complex numbers, e.g. c = a - b, where a and b are complex numbers.
- Complex operator$*$ (const Complex &a, const Complex &b)

  Implement the mul operation between two complex numbers, e.g. c = a $*$ b, where a and b are complex numbers.
- Complex operator/ (const Complex &a, const Complex &b)

  Implement the div operation between two complex numbers, e.g. c = a / b, where a and b are complex numbers.
- void operator+= (Complex &a, const Complex &b)

  Implement the += operation between two complex numbers, e.g. a += b, where a and b are complex numbers.

- void operator-= (Complex &a, const Complex &b)

    Implement the -= operation between two complex numbers, e.g. a -= b, where a and b are complex numbers.
- void operator$*$= (Complex &a, const Complex &b)

    Implement the $*$= operation between two complex numbers, e.g. a $*$= b, where a and b are complex numbers.
- void operator/= (Complex &a, const Complex &b)

    Implement the /= operation between two complex numbers, e.g. a /= b, where a and b are complex numbers.
- Complex operator$*$ (const Complex a, const RFLOAT x)

    Implement the mul operation between a complex number and a RFLOAT number, e.g. c = a + b, where a is a complex number and b is a RFLOAT number.
- Complex **operator$*$** (const RFLOAT x, const Complex a)
- void **operator$*$=** (Complex &a, const RFLOAT x)
- Complex **operator/** (const Complex a, const RFLOAT x)
- void **operator/=** (Complex &a, const RFLOAT x)

2.1.1 Detailed Description

Complex.h defines complex number related operations like +,-,$*$,/,|a|, $|a|^2$ and so on.

2.1.2 Function Documentation

2.1.2.1 RFLOAT ABS (const Complex & *a*) [inline]

Calculate the |a| of complex number a.

Returns

The |a| of complex number a.

Parameters

in	*a*	The number whose |a| needs to be calculated

2.1.2.2 Complex COMPLEX (RFLOAT *a*, RFLOAT *b*) [inline]

Construct a complex number with a as real part and b as image part.

Returns

A initialized complex number.

Parameters

in	*a*	Real part
in	*b*	Image part

2.1.2.3 Complex COMPLEX_POLAR (const RFLOAT *phi*) [inline]

Get the complex number representation $\mathbf{c}$, given the angle ϕ in polar coordinate.

Returns

Complex polar representation.

Parameters

in	*phi*	ϕ - Angle value ϕ to be converted

2.1.2.4 Complex CONJUGATE (const Complex & *a*) [inline]

Get the conjugate result based on value a.

Returns

Conjugate of a.

Parameters

in	*a*	Complex number whose conjuate value needs to be returned

2.1.2.5 RFLOAT gsl_real_imag_sum (const Complex & *a*) [inline]

Calcuate the sum of the complex number a's real part and image part.

Returns

The sum of a's real part and image part

Parameters

in	*a*	Complex number to be operated

2.1.2.6 RFLOAT IMAG (const Complex & *a*) [inline]

Get the image part of the complex number a.

Returns

Complex number a's image part

Parameters

in	*a*	Complex number to be operated

2.1.2.7 Complex operator$*$ (const Complex & *a*, const Complex & *b*) [inline]

Implement the mul operation between two complex numbers, e.g. c = a $*$ b, where a and b are complex numbers.

Returns

the result of c = a $*$ b.

Parameters

in	*a*	First operand used to perform sub operation between two complex numbers

in	*b*	Second operand used to perform sub operation between two complex numbers

2.1.2.8 Complex operator∗ (const Complex *a*, const RFLOAT *x*) [inline]

Implement the mul operation between a complex number and a RFLOAT number, e.g. c = a + b, where a is a complex number and b is a RFLOAT number.

Returns

the result of c = a ∗ b.

Parameters

in	*a*	First operand with type of complex used to perform mul operation.
in	*x*	Second operand with type of RFLOAT used to perform mul operation.

2.1.2.9 void operator∗= (Complex & *a*, const Complex & *b*) [inline]

Implement the ∗= operation between two complex numbers, e.g. a ∗= b, where a and b are complex numbers.

Returns

the result of a ∗= b.

Parameters

in	*a*	First operand used to perform ∗= operation between two complex numbers
in	*b*	Second operand used to perform ∗= operation between two complex numbers

2.1.2.10 Complex operator+ (const Complex & *a*, const Complex & *b*) [inline]

Implement the add operation between two complex numbers, e.g. c = a + b, where a and b are complex numbers.

Returns

the result of c = a + b.

Parameters

in	*a*	First operand used to perform add operation between two complex numbers
in	*b*	Second operand used to perform add operation between two complex numbers

2.1.2.11 void operator+= (Complex & *a*, const Complex & *b*) [inline]

Implement the += operation between two complex numbers, e.g. a += b, where a and b are complex numbers.

Returns

the result of a += b.

Parameters

in	*a*	First operand used to perform += operation between two complex numbers
in	*b*	Second operand used to perform += operation between two complex numbers

2.1.2.12 Complex operator- (const Complex & *a*, const Complex & *b*) [inline]

Implement the sub operation between two complex numbers, e.g. c = a - b, where a and b are complex numbers.

Returns

the result of c = a - b.

Parameters

in	*a*	First operand used to perform sub operation between two complex numbers
in	*b*	Second operand used to perform sub operation between two complex numbers

2.1.2.13 void operator-= (Complex & *a*, const Complex & *b*) [inline]

Implement the -= operation between two complex numbers, e.g. a -= b, where a and b are complex numbers.

Returns

the result of a -= b.

Parameters

in	*a*	First operand used to perform -= operation between two complex numbers
in	*b*	Second operand used to perform -= operation between two complex numbers

2.1.2.14 Complex operator/ (const Complex & *a*, const Complex & *b*) [inline]

Implement the div operation between two complex numbers, e.g. c = a / b, where a and b are complex numbers.

Returns

the result of c = a / b.

Parameters

in	*a*	First operand used to perform div operation between two complex numbers
in	*b*	Second operand used to perform div operation between two complex numbers

2.1.2.15 void operator/= (Complex & *a*, const Complex & *b*) [inline]

Implement the /= operation between two complex numbers, e.g. a /= b, where a and b are complex numbers.

Returns

the result of a /= b.

Parameters

in	*a*	First operand used to perform /= operation between two complex numbers
in	*b*	Second operand used to perform /= operation between two complex numbers

2.1.2.16 RFLOAT REAL (const Complex & *a*) `[inline]`

Get the real part of the complex number a.

Returns

Complex number a's real part

Parameters

in	*a*	Complex number to be operated

2.2 include/Geometry/Euler.h File Reference

Euler.h contains several functions, for operations of quaternions, converting between Euler angles, rotation matrices and unit quaternions and sampling rotation matrices from even distribution.

```
#include <cmath>
#include <gsl/gsl_math.h>
#include "Macro.h"
#include "Typedef.h"
#include "Precision.h"
#include "Random.h"
#include "Functions.h"
```

Functions

- void quaternion_mul (dvec4 &dst, const dvec4 &a, const dvec4 &b)

  Calculate the product of two quaternions $\mathbf{q_1}$ *and* $\mathbf{q_2}$*.*
- dvec4 quaternion_conj (const dvec4 &quat)

  Calculate the conjugate quaternion of a quaternion.
- void angle (double &phi, double &theta, const dvec3 &src)

  Calculate ϕ *and* θ *given a certain direction* $\mathbf{v}$*.*
- void angle (double &phi, double &theta, double &psi, const dmat33 &src)

  Calculate ϕ*,* θ *and* ψ *of the rotation represented by the rotation matrix* $\mathbf{R}$*.*
- void angle (double &phi, double &theta, double &psi, const dvec4 &src)

  Calculate ϕ*,* θ *and* ψ *of the rotation represented by the unit quaternion* $\mathbf{q}$*.*
- void quaternion (dvec4 &dst, const double phi, const double theta, const double psi)

  Calculate the unit quaternion $\mathbf{q}$ *for representing the rotation, given 3 Euler angles* ϕ*,* θ *and* ψ*.*
- void quaternion (dvec4 &dst, const double phi, const dvec3 &axis)

  Calculate the unit quaternion $\mathbf{q}$ *for representing the rotation, given the rotation axis* $\mathbf{r}$ *and the rotation angle around this axis* ϕ*.*
- void quaternion (dvec4 &dst, const dmat33 &src)

  Calculate the unit quaternion $\mathbf{q}$ *for representing the rotation, given the rotation matrix* $\mathbf{R}$*.*
- void rotate2D (dmat22 &dst, const dvec2 &vec)

  Calculate the rotation matrix (2D) $\mathbf{R}$*, which rotates the unit vector* $\mathbf{v_0} = \{1,0\}$ *to the given unit vector* $\mathbf{v}$*.*

- void rotate2D (dmat22 &dst, const double phi)

  Calculate the rotation matrix (2D) $\mathbf{R}$*, given the rotation angle* ϕ*.*
- void direction (dvec3 &dst, const double phi, const double theta)

  Caclulate the unit direction vector $\mathbf{v}$*, given the rotation angle* ϕ *and* θ*.*
- void rotate3D (dmat33 &dst, const double phi, const double theta, const double psi)

  Caclulate the rotation matrix $\mathbf{R}$*, given the rotation angle* ϕ*,* θ *and* ψ*.*
- void rotate3D (dmat33 &dst, const dvec4 &src)

  Calculate the rotation matrix $\mathbf{R}$*, given the unit quaternion* $\mathbf{q}$ *which represents this rotation.*
- void rotate3DX (dmat33 &dst, const double phi)

  Calculate the rotation matrix $\mathbf{R}$ *which represents the rotation along X-axis with rotation angle* ϕ*.*
- void rotate3DY (dmat33 &dst, const double phi)

  Calculate the rotation matrix $\mathbf{R}$ *which represents the rotation along Y-axis with rotation angle* ϕ*.*
- void rotate3DZ (dmat33 &dst, const double phi)

  Calculate the rotation matrix $\mathbf{R}$ *which represents the rotation along Z-axis with rotation angle* ϕ*.*
- void alignZ (dmat33 &dst, const dvec3 &vec)

  Calculate the rotation matrix $\mathbf{R}$ *which aligns a direction vector* $\mathbf{v}$ *to Z-axis.*
- void rotate3D (dmat33 &dst, const double phi, const dvec3 &axis)

  Calculate the rotation matrix $\mathbf{R}$ *which represents the rotation along the axis* $\mathbf{v}$ *with rotation angle* ϕ*.*
- void reflect3D (dmat33 &dst, const dvec3 &plane)

  Calculate the transformation matrix $\mathbf{M}$ *of reflection against a certian plane, which is represented by its normal vector* $\mathbf{n}$*.*
- void swingTwist (dvec4 &swing, dvec4 &twist, const dvec4 &src, const dvec3 &vec)

  Calculate the two quaternions $\mathbf{q_s}$ *and* $\mathbf{q_t}$*, which represent swing and twist along axis* $\mathbf{v}$ *respectively, representing the rotation represented by quaternion* $\mathbf{q}$*.*
- void randRotate2D (dmat22 &rot)

  Sample a 2D rotation matrix $\mathbf{R}$ *from even distribution.*
- void randRotate3D (dmat33 &rot)

  Sample a 3D rotation matrix $\mathbf{R}$ *from even distribution.*

2.2.1 Detailed Description

Euler.h contains several functions, for operations of quaternions, converting between Euler angles, rotation matrices and unit quaternions and sampling rotation matrices from even distribution. Quaternions are a number system that extends the complex numbers. Unit quaternions provide a convenient mathematical notation for representing rotations of objects in 3D. Compared to Euler angles, they are simpler to compose and aovid the problem of glimbal lock. Compared to rotation matrices, they are more compact and more efficient. Moroever, unlike Euler angles, unit quaternions do not rely on the choosing and order of the rotation axes.

To be noticed, Euler angles in this file follow the standard of ZXZ Euler system. In other words, Euler angle set $\{\phi, \theta, \psi\}$ stands for rotating along Z axis with ϕ, followed by rotating along X axis with θ, and followed by rotating along Z axis with ψ.

2.2.2 Function Documentation

2.2.2.1 void alignZ (dmat33 & *dst,* const dvec3 & *vec*)

Calculate the rotation matrix $\mathbf{R}$ which aligns a direction vector $\mathbf{v}$ to Z-axis.

Parameters

out	*dst*	$\mathbf{R}$
in	*vec*	$\mathbf{v}$

2.2.2.2 void angle (double & *phi,* double & *theta,* const dvec3 & *src*)

Calculate ϕ and θ given a certain direction $\mathbf{v}$.

$\mathbf{v}$ must be a unit vector. Output value ϕ ranges $[0, 2\pi)$, and θ ranges $[0, \pi]$.

Parameters

out	*phi*	ϕ
out	*theta*	θ
in	*src*	$\mathbf{v}$

2.2.2.3 void angle (double & *phi,* double & *theta,* double & *psi,* const dmat33 & *src*)

Calculate ϕ, θ and ψ of the rotation represented by the rotation matrix $\mathbf{R}$.

$\mathbf{R}$ must be an orthogonal matirx and determinant of which equals to 1. In other words, $RR^T = I$ and $\det A = 1$. Output value ϕ ranges $[0, 2\pi)$, θ ranges $[0, \pi]$, and ψ ranges $[0, 2\pi)$.

Parameters

out	*phi*	ϕ
out	*theta*	θ
out	*psi*	ψ
in	*src*	$\mathbf{R}$

2.2.2.4 void angle (double & *phi,* double & *theta,* double & *psi,* const dvec4 & *src*)

Calculate ϕ, θ and ψ of the rotation represented by the unit quaternion $\mathbf{q}$.

Parameters

out	*phi*	ϕ
out	*theta*	θ
out	*psi*	ψ
in	*src*	$\mathbf{q}$

2.2.2.5 void direction (dvec3 & *dst,* const double *phi,* const double *theta*)

Caclulate the unit direction vector $\mathbf{v}$, given the rotation angle ϕ and θ.

Parameters

out	*dst*	$\mathbf{v}$
in	*phi*	ϕ
in	*theta*	θ

2.2.2.6 void quaternion (dvec4 & *dst,* const double *phi,* const double *theta,* const double *psi*)

Calculate the unit quaternion $\mathbf{q}$ for representing the rotation, given 3 Euler angles ϕ, θ and ψ.

Parameters

out	*dst*	$\mathbf{q}$
in	*phi*	ϕ
in	*theta*	θ
in	*psi*	ψ

2.2.2.7 void quaternion (dvec4 & *dst*, const double *phi*, const dvec3 & *axis*)

Calculate the unit quaternion $\mathbf{q}$ for representing the rotation, given the rotation axis $\mathbf{r}$ and the rotation angle around this axis ϕ.

This rotation axis $\mathbf{r}$ must be a unit vector, while the rotation angle ϕ ranges $(-\infty, +\infty)$.

Parameters

out	*dst*	$\mathbf{q}$
in	*phi*	ϕ
in	*axis*	$\mathbf{r}$

2.2.2.8 void quaternion (dvec4 & *dst*, const dmat33 & *src*)

Calculate the unit quaternion $\mathbf{q}$ for representing the rotation, given the rotation matrix $\mathbf{R}$.

Parameters

out	*dst*	$\mathbf{q}$
in	*src*	$\mathbf{R}$

2.2.2.9 dvec4 quaternion_conj (const dvec4 & *quat*)

Calculate the conjugate quaternion of a quaternion.

Returns

the conjugate quaternion

Parameters

in	*quat*	a quaternion

2.2.2.10 void quaternion_mul (dvec4 & *dst*, const dvec4 & *a*, const dvec4 & *b*)

Calculate the product of two quaternions $\mathbf{q_1}$ and $\mathbf{q_2}$.

Assuming that $\mathbf{q_1} = (w_1, x_1, y_1, z_1)$ and $\mathbf{q_2} = (w_2, x_2, y_2, z_2)$, the product can be calculated as

$$\begin{pmatrix} w_1 \\ x_1 \\ y_1 \\ z_1 \end{pmatrix} \times \begin{pmatrix} w_2 \\ x_2 \\ y_2 \\ z_2 \end{pmatrix} = \begin{pmatrix} w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2 \\ w_1 x_2 + x_1 w_2 + y_1 z_2 - z_1 y_2 \\ w_1 y_2 - x_1 z_2 + y_1 w_2 + z_1 x_2 \\ w_1 z_2 + x_1 y_2 - y_1 x_2 + z_1 w_2 \end{pmatrix}$$

Parameters

out	*dst*	product, a quaternion
in	*a*	left multiplier, $\mathbf{q_1}$
in	*b*	right multiplier, $\mathbf{q_2}$

2.2.2.11 void randRotate2D (dmat22 & *rot*)

Sample a 2D rotation matrix $\mathbf{R}$ from even distribution.

Parameters

out	*rot*	$\mathbf{R}$

2.2.2.12 void randRotate3D (dmat33 & *rot*)

Sample a 3D rotation matrix $\mathbf{R}$ from even distribution.

Parameters

out	*rot*	$\mathbf{R}$

2.2.2.13 void reflect3D (dmat33 & *dst*, const dvec3 & *plane*)

Calculate the transformation matrix $\mathbf{M}$ of reflection against a certian plane, which is represented by its normal vector $\mathbf{n}$.

Parameters

out	*dst*	$\mathbf{M}$
in	*plane*	$\mathbf{n}$

2.2.2.14 void rotate2D (dmat22 & *dst*, const dvec2 & *vec*)

Calculate the rotation matrix (2D) $\mathbf{R}$, which rotates the unit vector $\mathbf{v_0} = \{1, 0\}$ to the given unit vector $\mathbf{v}$.

Parameters

out	*dst*	$\mathbf{R}$
in	*vec*	$\mathbf{v}$

2.2.2.15 void rotate2D (dmat22 & *dst*, const double *phi*)

Calculate the rotation matrix (2D) $\mathbf{R}$, given the rotation angle ϕ.

Parameters

out	*dst*	$\mathbf{R}$
in	*phi*	ϕ

2.2.2.16 void rotate3D (dmat33 & *dst*, const double *phi*, const double *theta*, const double *psi*)

Caclulate the rotation matrix $\mathbf{R}$, given the rotation angle ϕ, θ and ψ.

Parameters

out	*dst*	$\mathbf{R}$
in	*phi*	ϕ
in	*theta*	θ
in	*psi*	ψ

2.2.2.17 void rotate3D (dmat33 & *dst*, const dvec4 & *src*)

Calculate the rotation matrix $\mathbf{R}$, given the unit quaternion $\mathbf{q}$ which represents this rotation.

Parameters

out	*dst*	$\mathbf{R}$
in	*src*	$\mathbf{q}$

2.2.2.18 void rotate3D (dmat33 & *dst*, const double *phi*, const dvec3 & *axis*)

Calculate the rotation matrix $\mathbf{R}$ which represents the rotation along the axis $\mathbf{v}$ with rotation angle ϕ.

Parameters

out	*dst*	$\mathbf{R}$
in	*phi*	ϕ
in	*axis*	$\mathbf{v}$

2.2.2.19 void rotate3DX (dmat33 & *dst*, const double *phi*)

Calculate the rotation matrix $\mathbf{R}$ which represents the rotation along X-axis with rotation angle ϕ.

Parameters

out	*dst*	$\mathbf{R}$
in	*phi*	ϕ

2.2.2.20 void rotate3DY (dmat33 & *dst*, const double *phi*)

Calculate the rotation matrix $\mathbf{R}$ which represents the rotation along Y-axis with rotation angle ϕ.

Parameters

out	*dst*	$\mathbf{R}$
in	*phi*	ϕ

2.2.2.21 void rotate3DZ (dmat33 & *dst*, const double *phi*)

Calculate the rotation matrix $\mathbf{R}$ which represents the rotation along Z-axis with rotation angle ϕ.

Parameters

out	*dst*	$\mathbf{R}$

in	*phi*	ϕ

2.2.2.22 void swingTwist (dvec4 & *swing,* dvec4 & *twist,* const dvec4 & *src,* const dvec3 & *vec*)

Calculate the two quaternions $\mathbf{q_s}$ and $\mathbf{q_t}$, which represent swing and twist along axis $\mathbf{v}$ respectively, representing the rotation represented by quaternion $\mathbf{q}$.

Parameters

out	*swing*	$\mathbf{q_s}$
out	*twist*	$\mathbf{q_t}$
in	*src*	$\mathbf{q}$
in	*vec*	$\mathbf{v}$

Index