SL: Module Quaternion

— Version sl-0.1.4 / Quaternion-1.1 —

Module Quaternion and associated rotations manipulations

Pierre Aubert paubert@mathinsa.insa-lyon.fr

Contents

Abstract

This module provide an easy way to deal with 3D rotations with the use of the quaternions field. Quaternions, pure Quaternion, *i.e.* rotations vectors and conversion to and from 3D rotations matrices are provided. The expected applications are robotics and finite elements methods applied to solids mechanics.

1 Introduction

to do

4 Example

BRYANT *'s Angles*

2 Code documentation

List of all public methods

Names

2.1 namespace **sl**

Names

2.1.1 class **Quaternion**

Modeling of a quaternion

Public Members

2.1.1.1
Explanation

Read the documentation

2.1.1.2
Constructor(s), Destructors(s)

Names

Quaternion (T const q1 = T(0.0), T const q2 = T(0.0), T const q3 = T(0.0), T const q4 = T(0.0))
default constructor

Quaternion (Quaternion <T2> const& mq)
default copy constructor

2.1.1.2.1
Quaternion (Matrix<T> const& rot)

construct a quaternion from a rotation matrix @doc we construct a pur quaternion, such that q_1, q_2, q_3 is the vector defining the rotation and such that the norm of the quaternion gives the amplitude of this rotation. see We use an algorithm from Spurrier.

2.1.1.3 Access to composant

Names

T	**q1** () const	*get real part* q_0
T	**q2** () const	*get* q_1
T	**q3** () const	*get* q_2
T	**q4** () const	*get* q_3
T	**real** () const	*get real part*
Bool	**isPur** () const	*return true if* q *is pur*

2.1.1.4 Simple arithmetics operators

Names

&	**operator =** (T const x)	*Affectation* $q \leftarrow x$
&	**operator +=** (T const x)	*Add a real* $q \leftarrow q + x$
&	**operator -=** (T const x)	*Substract a real* $q \leftarrow q - x$
&	**operator *=** (T const x)	*Multiply by a real* $q \leftarrow q * x$
&	**operator /=** (T const x)	*Divide by a real* $q \leftarrow q/x$
&	**operator =** (Quaternion <T2> const& z)	$q = z$
&	**operator +=** (Quaternion <T2> const& z)	$q+ = z$
&	**operator -=** (Quaternion <T2> const& z)	$q- = z$
&	**operator *=** (Quaternion <T2> const& y)	$q* = z$
&	**operator /=** (Quaternion <T2> const& z)	

$$q / = z$$

2.1.1.5

arithmetic operations between two quaternions

Names

const&	**operator +** ()	*+q unary plus*
	operator - ()	*−q unary minus*

2.1.2

namespace **IO operations for Quaternions**

Names

std::ostream&
operator << (std::ostream& os, Quaternion <T> const& z)
I/O operators <<

std::istream& **operator**>> (std::istream& is, Quaternion <T> & z)
I/O operators> >

2.1.3

namespace **Misc functions define over quaternion's field**

Names

operator˜ (Quaternion <T> const& z)
return the conjugate of a quaternion : $\bar{q}$.

conj (Quaternion <T> const& z)
return the conjugate a quaternion : $\bar{q}$.

2.1.3.1 T **norm2** (Quaternion <T> const& z) *return the 2−norm of a quaternion :* $N_2(q) = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}$ *It's a true norm in a mathematical view-point* .. 11

2.1.3.2 T **norm** (Quaternion <T> const& z)
return the norm of a quaternion : $N(q) = q_0^2 + q_1^2 + q_2^2 + q_3^2$ *It's* not *a true norm in a mathematical view-point* .. 11

2.1.3.3 **prod_scal** (Quaternion <T> const& x, Quaternion <T> const& y)
return the scalar product of two quaternions: $<q_1,\ q_2> = \frac{1}{2}(\bar{q}_1^i q_2^i + \bar{q}_2^i q_1^i)$ ... 12

2.1.3.4 **commutator** (Quaternion <T> const& x, Quaternion <T> const& y)
return the commutator of two quaternions: $[q_1,\ q_2] = q_1 q_2 - q_2 q_1$ 12

inv (Quaternion <T> const& q)
return the inverse of a quaternion : $q^{-1} = \frac{1}{q}$.

promoted_t **operator+** (Quaternion <T1> const& z1, Quaternion <T2> const& z2)
$q2 + q3$ *addition of two quaternions*

promoted_t **operator+** (Quaternion <T> const& z, T const x)
$q2 + x$ *addition of a quaternion and a real*

promoted_t **operator+** (T const x, Quaternion <T> const& z)
$x + q2$ *addition of a real and a quaternion*

promoted_t **operator-** (Quaternion <T1> const& z1, Quaternion <T2> const& z2)
$q2 - q3$ *substraction of two quaternion*

promoted_t **operator-** (Quaternion <T> const& z, T const x)
$q2 - x$ *substraction of a quaternion and a real*

promoted_t **operator-** (T const x, Quaternion <T> const& z)
$x - q2$ *substraction of a real and a quaternion*

promoted_t **operator*** (Quaternion <T1> const& x, Quaternion <T2> const& y)
$q2 * q3$ *multiplication of two quaternions*

promoted_t **operator*** (Quaternion <T> const& z, T const x)

$q2 * x$ multiplication of a real and a quaternion

promoted_t **operator*** (T const x, Quaternion <T> const& z)
$x * q2$ multiplication of a real and a quaternion

promoted_t **operator/** (Quaternion <T1> const& z1, Quaternion <T2> const& z2)
$q2/q3$ division of two quaternions

promoted_t **operator/** (Quaternion <T> const& z, T const x)
$q2/x$ division of a quaternion by a real

2.1.3.1

T **norm2** (Quaternion <T> const& z)

return the 2−norm of a quaternion : $N_2(q) = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}$ It's a true norm in a mathematical view-point

return the 2−norm of a quaternion : $N_2(q) = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}$ It's a true norm in a mathematical view-point.

2.1.3.2

T **norm** (Quaternion <T> const& z)

return the norm of a quaternion : $N(q) = q_0^2 + q_1^2 + q_2^2 + q_3^2$ It's not *a true norm in a mathematical view-point*

return the norm of a quaternion : $N(q) = q_0^2 + q_1^2 + q_2^2 + q_3^2$ It's *not* a true norm in a mathematical view-point.

2.1.3.3

prod_scal (Quaternion <T> const& x, Quaternion <T> const& y)

return the scalar product of two quaternions: $<q_1, q_2> = \frac{1}{2}(\bar{q}_1^i q_2^i + \bar{q}_2^i q_1^i)$

return the scalar product of two quaternions: $<q_1, q_2> = \frac{1}{2}(\bar{q}_1^i q_2^i + \bar{q}_2^i q_1^i)$

2.1.3.4

commutator (Quaternion <T> const& x, Quaternion <T> const& y)

return the commutator of two quaternions: $[q_1, q_2] = q_1 q_2 - q_2 q_1$

return the commutator of two quaternions: $[q_1, q_2] = q_1 q_2 - q_2 q_1$

2.1.4

class **QuaternionFormat**

Quaternion Format gives a format for IO

Public Members

2.1.4.1

Explanation

QuaternionFormat let you define your favorite format for loading, store and display Quaternion.

For example, let

```
my_left      = '[';
my_separator = ',';
my_open      = '(';
my_close     = ')';
my_right     = ']';
```

leads to

```
[ q1, (q2, q3, q4)  ]
```

2.1.4.2

members(s)

Names

void **set_fmt** (Char l, Char s, Char o, Char c, Char r)
set format according to specifications

void **get_fmt** (Char &l, Char &s, Char &o, Char &c, Char &r)
get format according to specifications

3 Conclusion

to do

Class Graph