

THUNDER

Generated by Doxygen 1.8.5

Wed Sep 12 2018 15:40:59

Contents

1	Hierarchical Index	1
1.1	Class Hierarchy	1
2	Class Index	3
2.1	Class List	3
3	File Index	5
3.1	File List	5
4	Class Documentation	7
4.1	_complex_float_t Struct Reference	7
4.2	Parallel Class Reference	7
4.2.1	Constructor & Destructor Documentation	8
4.2.1.1	Parallel	8
4.2.1.2	~Parallel	8
4.2.2	Member Function Documentation	8
4.2.2.1	commRank	8
4.2.2.2	commSize	8
4.2.2.3	hemi	8
4.2.2.4	isA	8
4.2.2.5	isB	8
4.2.2.6	isMaster	8
4.2.2.7	setCommRank	8
4.2.2.8	setCommSize	9
4.2.2.9	setHemi	9
4.2.2.10	setMPIEnv	9
4.2.2.11	setMPIEnv	9
4.2.3	Member Data Documentation	9
4.2.3.1	_commRank	9
4.2.3.2	_commSize	9
4.2.3.3	_hemi	9
5	File Documentation	11

5.1	include/Complex.h File Reference	11
5.1.1	Detailed Description	12
5.2	include/Geometry/Euler.h File Reference	12
5.2.1	Detailed Description	13
5.2.2	Function Documentation	13
5.2.2.1	alignZ	13
5.2.2.2	angle	13
5.2.2.3	angle	14
5.2.2.4	angle	14
5.2.2.5	direction	14
5.2.2.6	quaternion	14
5.2.2.7	quaternion	14
5.2.2.8	quaternion	15
5.2.2.9	quaternion_conj	15
5.2.2.10	quaternion_mul	15
5.2.2.11	randRotate2D	15
5.2.2.12	randRotate3D	15
5.2.2.13	reflect3D	16
5.2.2.14	rotate2D	16
5.2.2.15	rotate2D	16
5.2.2.16	rotate3D	16
5.2.2.17	rotate3D	16
5.2.2.18	rotate3D	17
5.2.2.19	rotate3DX	18
5.2.2.20	rotate3DY	18
5.2.2.21	rotate3DZ	18
5.2.2.22	swingTwist	18

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

_complex_float_t	7
noncopyable	
Parallel	7

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

_complex_float_t	7
Parallel	7

Chapter 3

File Index

3.1 File List

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

include/ Complex.h	
Complex.h defines complex number related operations like +, -, *, /, $ a $, $ a ^2$ and so on	11
include/ Parallel.h	??
include/ Precision.h	??
include/Geometry/ Euler.h	
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	12

Chapter 4

Class Documentation

4.1 `_complex_float_t` Struct Reference

Public Attributes

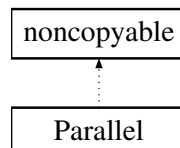
- double **dat** [2]

The documentation for this struct was generated from the following file:

- include/Precision.h

4.2 Parallel Class Reference

Inheritance diagram for Parallel:



Public Member Functions

- `Parallel ()`
- `~Parallel ()`
- void `setMPIEnv ()`
- void `setMPIEnv` (const int `commSize`, const int `commRank`, const MPI_Comm &`hemi`, const MPI_Comm &`slav`)
- bool `isMaster ()` const
- bool `isA ()` const
- bool `isB ()` const
- int `commSize ()` const
- void `setCommSize` (const int `commSize`)
- int `commRank ()` const
- void `setCommRank` (const int `commRank`)
- MPI_Comm `hemi ()` const
- void `setHemi` (const MPI_Comm &`hemi`)
- MPI_Comm `slav ()` const
- void `setSlav` (const MPI_Comm &`slav`)

Protected Attributes

- `int _commSize`
- `int _commRank`
- `MPI_Comm _hemi`
- `MPI_Comm _slav`

4.2.1 Constructor & Destructor Documentation

4.2.1.1 `Parallel::Parallel ()`

default constructor

4.2.1.2 `Parallel::~~Parallel ()`

default destructor

4.2.2 Member Function Documentation

4.2.2.1 `int Parallel::commRank () const`

This function returns the rank ID of the current process in `MPI_COMM_WORLD`.

4.2.2.2 `int Parallel::commSize () const`

This function returns the number of processes in `MPI_COMM_WORLD`.

4.2.2.3 `MPI_Comm Parallel::hemi () const`

This function returns the hemisphere of the current process.

4.2.2.4 `bool Parallel::isA () const`

This function returns whether the current process is in hemisphere A or not.

4.2.2.5 `bool Parallel::isB () const`

This function returns whether the current process is in hemisphere B or not.

4.2.2.6 `bool Parallel::isMaster () const`

This function returns whether the current process is the master process or not.

4.2.2.7 `void Parallel::setCommRank (const int commRank)`

This function sets the rank ID of the current process in `MPI_COMM_WORLD`.

Parameters

<i>commRank</i>	the rank ID of the current process in MPI_COMM_WORLD
-----------------	--

4.2.2.8 void Parallel::setCommSize (const int *commSize*)

This function sets the number of processes in MPI_COMM_WORLD.

Parameters

<i>commSize</i>	the number of processes in MPI_COMM_WORLD
-----------------	---

4.2.2.9 void Parallel::setHemi (const MPI_Comm & *hemi*)

This function sets the hemisphere of the current process.

Parameters

<i>hemi</i>	the hemisphere of the current process
-------------	---------------------------------------

4.2.2.10 void Parallel::setMPIEnv ()

This function detects the number of processes in MPI_COMM_WORLD and the rank ID of the current process in MPI_COMM_WORLD. Moreover, it will assign all process in MPI_COMM_WORLD into three parts: master, hemisphere A and hemisphere B.

4.2.2.11 void Parallel::setMPIEnv (const int *commSize*, const int *commRank*, const MPI_Comm & *hemi*, const MPI_Comm & *slav*)

This function inherits the MPI information by parameters.

Parameters

<i>commSize</i>	the numbbber of process in MPI_COMM_WORLD
<i>commRank</i>	the rank ID of the current process in MPI_COMM_WORLD
<i>hemi</i>	the hemisphere of the current process

4.2.3 Member Data Documentation

4.2.3.1 int Parallel::_commRank [protected]

the rank ID of the current process in MPI_COMM_WORLD

4.2.3.2 int Parallel::_commSize [protected]

number of processes in MPI_COMM_WORLD

4.2.3.3 MPI_Comm Parallel::_hemi [protected]

communicator of hemisphere A(B)

The documentation for this class was generated from the following file:

- include/Parallel.h

Chapter 5

File Documentation

5.1 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](#) **ts_complex_polar** (const RFLOAT r, const RFLOAT phi)
- [Complex](#) **CONJUGATE** (const [Complex](#) &a)
- RFLOAT **ABS** (const [Complex](#) &a)
- RFLOAT **ABS2** (const [Complex](#) &a)
- [Complex](#) **COMPLEX** (RFLOAT a, RFLOAT b)
- RFLOAT **REAL** (const [Complex](#) &a)
- RFLOAT **IMAG** (const [Complex](#) &a)
- RFLOAT **gsl_real** (const [Complex](#) &a)
- RFLOAT **gsl_imag** (const [Complex](#) &a)
- RFLOAT **gsl_real_imag_sum** (const [Complex](#) &a)
- [Complex](#) **operator-** (const [Complex](#) &a)
- [Complex](#) **operator+** (const [Complex](#) &a, const [Complex](#) &b)
- [Complex](#) **operator-** (const [Complex](#) &a, const [Complex](#) &b)
- [Complex](#) **operator*** (const [Complex](#) &a, const [Complex](#) &b)
- [Complex](#) **operator/** (const [Complex](#) &a, const [Complex](#) &b)
- void **operator+=** ([Complex](#) &a, const [Complex](#) b)
- void **operator-=** ([Complex](#) &a, const [Complex](#) b)
- void **operator*=** ([Complex](#) &a, const [Complex](#) b)
- void **operator/=** ([Complex](#) &a, const [Complex](#) b)
- [Complex](#) **operator*** (const [Complex](#) a, const RFLOAT x)
- [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)

5.1.1 Detailed Description

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

5.2 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)
Calculate 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)
Calculate 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.

5.2.1 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 avoid the problem of gimbal lock. Compared to rotation matrices, they are more compact and more efficient. Moreover, 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 .

5.2.2 Function Documentation

5.2.2.1 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}

5.2.2.2 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}

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

Parameters

out	<i>phi</i>	ϕ
out	<i>theta</i>	θ
out	<i>psi</i>	ψ
in	<i>src</i>	\mathbf{R}

5.2.2.4 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	<i>phi</i>	ϕ
out	<i>theta</i>	θ
out	<i>psi</i>	ψ
in	<i>src</i>	\mathbf{q}

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

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

Parameters

out	<i>dst</i>	\mathbf{v}
in	<i>phi</i>	ϕ
in	<i>theta</i>	θ

5.2.2.6 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	<i>dst</i>	\mathbf{q}
in	<i>phi</i>	ϕ
in	<i>theta</i>	θ
in	<i>psi</i>	ψ

5.2.2.7 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	<i>dst</i>	q
in	<i>phi</i>	ϕ
in	<i>axis</i>	r

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

Calculate the unit quaternion **q** for representing the rotation, given the rotation matrix **R**.

Parameters

out	<i>dst</i>	q
in	<i>src</i>	R

5.2.2.9 dvec4 quaternion_conj (const dvec4 & *quat*)

Calculate the conjugate quaternion of a quaternion.

Returns

the conjugate quaternion

Parameters

in	<i>quat</i>	a quaternion
----	-------------	--------------

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

Calculate the product of two quaternions **q₁** and **q₂**.

Assuming that **q₁** = (*w₁*, *x₁*, *y₁*, *z₁*) and **q₂** = (*w₂*, *x₂*, *y₂*, *z₂*), 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	<i>dst</i>	product, a quaternion
in	<i>a</i>	left multiplier, q₁
in	<i>b</i>	right multiplier, q₂

5.2.2.11 void randRotate2D (dmat22 & *rot*)

Sample a 2D rotation matrix **R** from even distribution.

Parameters

out	<i>rot</i>	R
-----	------------	----------

5.2.2.12 void randRotate3D (dmat33 & *rot*)

Sample a 3D rotation matrix **R** from even distribution.

Parameters

out	rot	R
-----	-----	----------

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

Calculate the transformation matrix **M** of reflection against a certian plane, which is represented by its normal vector **n**.

Parameters

out	<i>dst</i>	M
in	<i>plane</i>	n

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

Calculate the rotation matrix (2D) **R**, which rotates the unit vector $\mathbf{v}_0 = \{1, 0\}$ to the given unit vector **v**.

Parameters

out	<i>dst</i>	R
in	<i>vec</i>	v

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

Calculate the rotation matrix (2D) **R**, given the rotation angle ϕ .

Parameters

out	<i>dst</i>	R
in	<i>phi</i>	ϕ

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

Caclulate the rotation matrix **R**, given the rotation angle ϕ , θ and ψ .

Parameters

out	<i>dst</i>	R
in	<i>phi</i>	ϕ
in	<i>theta</i>	θ
in	<i>psi</i>	ψ

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

Calculate the rotation matrix **R**, given the unit quaternion **q** which represents this rotation.

Parameters

out	<i>dst</i>	R
in	<i>src</i>	q

5.2.2.18 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	<i>dst</i>	R
in	<i>phi</i>	ϕ
in	<i>axis</i>	v

5.2.2.19 void rotate3DX (dmat33 & dst, const double phi)

Calculate the rotation matrix **R** which represents the rotation along X-axis with rotation angle ϕ .

Parameters

out	<i>dst</i>	R
in	<i>phi</i>	ϕ

5.2.2.20 void rotate3DY (dmat33 & dst, const double phi)

Calculate the rotation matrix **R** which represents the rotation along Y-axis with rotation angle ϕ .

Parameters

out	<i>dst</i>	R
in	<i>phi</i>	ϕ

5.2.2.21 void rotate3DZ (dmat33 & dst, const double phi)

Calculate the rotation matrix **R** which represents the rotation along Z-axis with rotation angle ϕ .

Parameters

out	<i>dst</i>	R
in	<i>phi</i>	ϕ

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

Calculate the two quaternions **q_s** and **q_t**, which represent swing and twist along axis **v** respectively, representing the rotation represented by quaternion **q**.

Parameters

out	<i>swing</i>	q_s
out	<i>twist</i>	q_t
in	<i>src</i>	q
in	<i>vec</i>	v

Index

- ~Parallel
 - Parallel, [8](#)
- _commRank
 - Parallel, [9](#)
- _commSize
 - Parallel, [9](#)
- _complex_float_t, [7](#)
- _hemi
 - Parallel, [9](#)
- alignZ
 - Euler.h, [13](#)
- angle
 - Euler.h, [13](#), [14](#)
- commRank
 - Parallel, [8](#)
- commSize
 - Parallel, [8](#)
- direction
 - Euler.h, [14](#)
- Euler.h
 - alignZ, [13](#)
 - angle, [13](#), [14](#)
 - direction, [14](#)
 - quaternion, [14](#), [15](#)
 - quaternion_conj, [15](#)
 - quaternion_mul, [15](#)
 - randRotate2D, [15](#)
 - randRotate3D, [15](#)
 - reflect3D, [16](#)
 - rotate2D, [16](#)
 - rotate3D, [16](#)
 - rotate3DX, [18](#)
 - rotate3DY, [18](#)
 - rotate3DZ, [18](#)
 - swingTwist, [18](#)
- hemi
 - Parallel, [8](#)
- include/Complex.h, [11](#)
- include/Geometry/Euler.h, [12](#)
- isA
 - Parallel, [8](#)
- isB
 - Parallel, [8](#)
- isMaster
 - Parallel, [8](#)
- Parallel, [7](#)
 - ~Parallel, [8](#)
 - _commRank, [9](#)
 - _commSize, [9](#)
 - _hemi, [9](#)
 - commRank, [8](#)
 - commSize, [8](#)
 - hemi, [8](#)
 - isA, [8](#)
 - isB, [8](#)
 - isMaster, [8](#)
 - Parallel, [8](#)
 - setCommRank, [8](#)
 - setCommSize, [9](#)
 - setHemi, [9](#)
 - setMPIEnv, [9](#)
- quaternion
 - Euler.h, [14](#), [15](#)
- quaternion_conj
 - Euler.h, [15](#)
- quaternion_mul
 - Euler.h, [15](#)
- randRotate2D
 - Euler.h, [15](#)
- randRotate3D
 - Euler.h, [15](#)
- reflect3D
 - Euler.h, [16](#)
- rotate2D
 - Euler.h, [16](#)
- rotate3D
 - Euler.h, [16](#)
- rotate3DX
 - Euler.h, [18](#)
- rotate3DY
 - Euler.h, [18](#)
- rotate3DZ
 - Euler.h, [18](#)
- setCommRank
 - Parallel, [8](#)
- setCommSize
 - Parallel, [9](#)
- setHemi
 - Parallel, [9](#)
- setMPIEnv
 - Parallel, [9](#)
- swingTwist
 - Euler.h, [18](#)