DataTypeExtensions

Generated by Doxygen 1.7.6.1

Sun Mar 23 2014 21:34:10

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

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

Chapter 3

Class Documentation

3.1 Geometrics::Circle< T > Class Template Reference

Public Member Functions

- Circle (const Point< T > &center, const T radius)
- Circle (const Circle &orig)

Public Attributes

- T **_radius**
- Point< T > **_center**

template<class T = int> class Geometrics::Circle< T >

3.1.1 Constructor & Destructor Documentation

3.1.1.1 template<class T = int> Geometrics::Circle< T >::Circle (const Point< T > & *center,* const T *radius*) [inline]

Creates an Circle with the given center and radius.

Parameters

center	The center of the Circle, represanted as Point.
radius	The radius of the Circle.

3.1.1.2 template<class T = int> Geometrics::Circle< T >::Circle (const Circle< T > & *orig*) [inline]

A copy of the given Circle is generated.

Parameters

orig	The Circle, which should be copied.

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

- CPlusPlusFrameWork/Geometrics/Circle.h

3.2 Geometrics::Point< T > Class Template Reference

`#include <Point.h>`

Public Member Functions

- Point (int dim,...)
- Point (const Point &orig)
- Point (const int dim, const T value)
- virtual ~Point ()
- const int & getDim ()
- Point & operator= (Point p)
- bool operator== (const Point &p)
- bool operator!= (const Point &p)
- const Point operator+ (const Point &p)
- Point & operator+= (const Point &p)
- const Point operator- (const Point &p)
- Point & operator-= (const Point &p)
- const T operator* (const Point &p)
- T & operator[] (const int &i)
- const T & operator[] (const int &i) const
- T & at (const int &i)
- const T & at (const int &i) const

Private Member Functions

- void swap (Point &p)

Private Attributes

- T * _coordinates
- int _dim

3.2.1 Detailed Description

template<class T = int>class Geometrics::Point< T >

The Point class describes an _dim dimensional point. The point is stored as an array.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 template<class T = int> Geometrics::Point< T >::Point (int *dim*, ...) [inline]

The first constructor.

Parameters

dim	The dimension of the Point
...	The coordinates of the Point given as dynamic parameter list.

3.2.2.2 template<class T = int> Geometrics::Point< T >::Point (const Point< T > & *orig*) [inline]

Make a deep copy of the given Point.

Parameters

orig	The original point, which should be copied.

3.2.2.3 template<class T = int> Geometrics::Point< T >::Point (const int *dim*, const T *value*) [inline]

The second constructor

Parameters

dim	The dimension of the Point.
value	All coordinates are set to that value.

3.2.2.4 template<class T = int> virtual Geometrics::Point< T >::~Point ( ) [inline, virtual]

The destructor, which deletes the array, storing the coordinates.

3.2.3 Member Function Documentation

3.2.3.1 template<class T = int> T& Geometrics::Point< T >::at (const int & *i*) [inline]

A function, doing th same like the [] operator. Non-Const variante.

Parameters

i	is the coordinate index

Returns

The value of the coordinate with the index i.

3.2.3.2 template<class T = int> const T& Geometrics::Point< T >::at (const int & *i*) const [inline]

A function, doing th same like the [] operator. Const variante.

Parameters

i	is the coordinate index

Returns

The value of the coordinate with the index i.

3.2.3.3 template<class T = int> const int& Geometrics::Point< T >::getDim ( ) [inline]

Return the dimension of the Point

Returns

The dimension of the Point.

3.2.3.4 template<class T = int> bool Geometrics::Point< T >::operator!= (const Point< T > & *p*) [inline]

Overloading the != operator.

Parameters

p	The other Point.

Returns

True, if not all the coordinates of both Points are equal.

3.2.3.5 template<class T = int> const T Geometrics::Point< T >::operator∗ (const Point< T > & *p*) [inline]

Overloading the ∗ operator. Calculate the dot product of two Points(p1, p2). Throw an assertion, if the dimension of the vectors are not the same.

Parameters

p	The other Point.

Returns

T The dot product of the two Points.

3.2.3.6 template<class T = int> const Point Geometrics::Point< T >::operator+ (const Point< T > & *p*) [inline]

Overloading the + operator. Add two Points p1 and p2. Throw an assertion, if the dimension of the Points are not the same.

Parameters

p	The other Point.

Returns

Point The Point(p3), where for all coordinates i, it holds: p3[i] = p1[i] + p2[i].

3.2.3.7 template<class T = int> Point& Geometrics::Point< T >::operator+= (const Point< T > & *p*) [inline]

Overloading the += operator. Add the Point p to the Point, standing before the += operator. Throw an assertion, if the dimension of the Points are not the same.

Parameters

p	The other Point.

Returns

The modified Point, standing before the += operator.

3.2.3.8 template<class T = int> const Point Geometrics::Point< T >::operator- (const Point< T > & *p*) [inline]

Overloading the - operator. Add two Points p1 and p2. Throw an assertion, if the dimension of the vectors are not the same.

Parameters

p	The other Point.

Returns

Point The Point(p3), where for all coordinate i, it holds: p3[i] = p1[i] - p2[i].

3.2.3.9 template<class T = int> Point& Geometrics::Point< T >::operator-= (const Point< T > & *p*) [inline]

Overloading the -= operator. Subtract the Point p from the Point, standing before the -= operator. Throw an assertion, if the dimension of the Points are not the same.

Parameters

p	The other Point.

Returns

The modified Point, standing before the -= operator.

3.2.3.10 template<class T = int> Point& Geometrics::Point< T >::operator= (Point< T > *p*) [inline]

The Point p is assigned to this, by swapping all the members of this and the deep copy of p.

Parameters

p	The point right to the =. It's wanted, that p is passed by value.

Returns

This, with the members' values of p.

3.2.3.11 template<class T = int> bool Geometrics::Point< T >::operator== (const Point< T > & *p*) [inline]

Overloading the == operator.

Parameters

p	The other Point.

Returns

True, if all the coordinates of both Points are the same.

3.2.3.12 template<class T = int> T& Geometrics::Point< T >::operator[] (const int & *i*) [inline]

Overloading the [] operator. Non-Const variante.

Parameters

i	is the coordinate index

Returns

The value of the coordinate with the index i.

3.2.3.13 template<class T = int> const T& Geometrics::Point< T >::operator[] (const int & *i*) const [inline]

Overloading the [] operator. Const variante.

Parameters

i	is the coordinate index

Returns

The value of the coordinate with the index i.

3.2.3.14 template<class T = int> void Geometrics::Point< T >::swap (Point< T > & *p*) [inline, private]

Swap all the members of p and this.

Parameters

p	The Point for the swapping

3.2.4 Member Data Documentation

3.2.4.1 template<class T = int> T* Geometrics::Point< T >::_coordinates [private]

The coordinates of the Point as array.

3.2.4.2 template<class T = int> int Geometrics::Point< T >::_dim [private]

The dimension of the Point.

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

- CPlusPlusFrameWork/Geometrics/Point.h

3.3 Geometrics::Point< float > Class Template Reference

`#include <Point.h>`

Public Member Functions

- Point (int dim,...)
- Point (const Point &orig)
- Point (const int dim, const float value)
- virtual ~Point ()
- const int & getDim ()
- Point & operator= (Point p)
- bool operator== (const Point &p)
- bool operator!= (const Point &p)
- const Point operator+ (const Point &p)
- Point & operator+= (const Point &p)
- const Point operator- (const Point &p)
- Point & operator-= (const Point &p)
- const float operator* (const Point &p)
- float & operator[] (const int &i)
- const float & operator[] (const int &i) const
- float & at (const int &i)
- const float & at (const int &i) const

Private Member Functions

- void swap (Point &p)

Private Attributes

- float * _coordinates
- int _dim

3.3.1 Detailed Description

template<>class Geometrics::Point< float >

Specialication for float. This is needed because the dynamic parameter list has problems with float values. For that, va_arg uses double and after that this double value is converted to a float value.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Geometrics::Point< float >::Point (int *dim*, ...) [inline]

The first constructor.

Parameters

dim	The dimension of the Vector.
...	The coordinates of the Point as dynamic parameter list.

3.3.2.2 Geometrics::Point< float >::Point (const Point< float > & *orig*) [inline]

Make a deep copy of the given Point.

Parameters

orig	The original point, which should be copied.

3.3.2.3 Geometrics::Point< float >::Point (const int *dim*, const float *value*) [inline]

The second constructor

Parameters

dim	The dimension of the Point.
value	All coordinates are set to that value.

3.3.2.4 virtual Geometrics::Point< float >::~Point ( ) [inline, virtual]

The destructor, which deletes the array, storing the coordinates.

3.3.3 Member Function Documentation

3.3.3.1 float& Geometrics::Point< float >::at (const int & *i*) [inline]

A function, doing th same like the [] operator. Non-Const variante.

Parameters

i	is the coordinate index

Returns

The value of the coordinate with the index i.

3.3.3.2 const float& Geometrics::Point< float >::at (const int & *i*) const [inline]

A function, doing th same like the [] operator. Const variante.

Parameters

i	is the coordinate index

Returns

The value of the coordinate with the index i.

3.3.3.3 const int& Geometrics::Point< float >::getDim ( ) [inline]

Return the dimension of the Point

Returns

The dimension of the Point.

3.3.3.4 bool Geometrics::Point< float >::operator!= (const Point< float > & *p*) [inline]

Overloading the != operator.

Parameters

p	The other Point.

Returns

True, if not all the coordinates of both Points are equal.

3.3.3.5 const float Geometrics::Point< float >::operator∗ (const Point< float > & *p*) [inline]

Overloading the ∗ operator. Calculate the dot product of two Points(p1, p2). Throw an assertion, if the dimension of the vectors are not the same.

Parameters

p	The other Point.

Returns

T The dot product of the two Points.

3.3.3.6 const Point Geometrics::Point< float >::operator+ (const Point< float > & *p*) [inline]

Overloading the + operator. Add two Points p1 and p2. Throw an assertion, if the dimension of the Points are not the same.

Parameters

p	The other Point.

Returns

Point The Point(p3), where for all coordinates i, it holds: p3[i] = p1[i] + p2[i].

3.3.3.7 Point& Geometrics::Point< float >::operator+= (const Point< float > & *p*) [inline]

Overloading the += operator. Add the Point p to the Point, standing before the += operator. Throw an assertion, if the dimension of the Points are not the same.

Parameters

p	The other Point.

Returns

The modified Point, standing before the += operator.

3.3.3.8 const Point Geometrics::Point< float >::operator- (const Point< float > & *p*) [inline]

Overloading the - operator. Add two Points p1 and p2. Throw an assertion, if the dimension of the vectors are not the same.

Parameters

p	The other Point.

Returns

Point The Point(p3), where for all coordinate i, it holds: p3[i] = p1[i] - p2[i].

3.3.3.9 Point& Geometrics::Point< float >::operator-= (const Point< float > & *p*) [inline]

Overloading the -= operator. Subtract the Point p from the Point, standing before the -= operator. Throw an assertion, if the dimension of the Points are not the same.

Parameters

p	The other Point.

Returns

The modified Point, standing before the -= operator.

3.3.3.10 Point& Geometrics::Point< float >::operator= (Point< float > *p*) [inline]

The Point p is assigned to this, by swapping all the members of this and the deep copy of p.

Parameters

p	The point right to the =. It's wanted, that p is passed by value.

Returns

This, with the members' values of p.

3.3.3.11 bool Geometrics::Point< float >::operator== (const Point< float > & *p*) [inline]

Overloading the == operator.

Parameters

p	The other Point.

Returns

True, if all the coordinates of both Points are the same.

3.3.3.12 float& Geometrics::Point< float >::operator[] (const int & *i*) [inline]

Overloading the [] operator. Non-Const variante.

Parameters

i	is the coordinate index

Returns

The value of the coordinate with the index i.

3.3.3.13 const float& Geometrics::Point< float >::operator[] (const int & *i*) const [inline]

Overloading the [] operator. Const variante.

Parameters

i	is the coordinate index

Returns

The value of the coordinate with the index i.

3.3.3.14 void Geometrics::Point< float >::swap (Point< float > & *p*) [inline, private]

Swap all the members of p and this.

Parameters

p	The Point for the swapping

3.3.4 Member Data Documentation

3.3.4.1 float* Geometrics::Point< float >::_coordinates [private]

The coordinates of the Point as array.

3.3.4.2 int Geometrics::Point< float >::_dim [private]

The dimension of the Point.

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

- CPlusPlusFrameWork/Geometrics/Point.h

3.4 Geometrics::Quaternion Class Reference

Public Member Functions

- **Quaternion** (float inW, float inX, float inY, float inZ)
- Quaternion (float alpha, float beta, float gamma)
- template<typename T >
  Quaternion (float angle, Vec3< T > const &axis)
- template<typename T , typename U >
  Quaternion (Vec3< T > const &v1, Vec3< U > const &v2)
- Quaternion operator* (Quaternion const &rOp) const
- Quaternion operator+ (Quaternion const &rOP) const
- void normalize ()
- bool isNormalized () const
- float angle (Quaternion const &toQuat) const
- Quaternion slerp (Quaternion const &destQt, float t, float eps=0.01) const
- Quaternion lerp (Quaternion const &destQt, float t) const
- void toByteArray (byte *bArray) const
- float rotAngleInDeg ()

Public Attributes

- float **w**
- float **x**
- float **y**
- float **z**

3.4.1 Constructor & Destructor Documentation

3.4.1.1 Geometrics::Quaternion::Quaternion (float *alpha,* float *beta,* float *gamma*)

Constructor from Euler angles. (I have too re-check the angle sequence sometimes)

Parameters

alpha	Rotation around the z axis (yaw)
beta	Rotation around the y axis (pitch)
gamma	Rotation around the x axis (roll)

3.4.1.2 template<typename T > Geometrics::Quaternion::Quaternion (float *angle,* Vec3< T > const & *axis*)

Constructor from angle and rotation axis

Parameters

angle	Rotation magnitude
gamma	Rotation axis

3.4.1.3 template<typename T , typename U > Geometrics::Quaternion::Quaternion (Vec3< T > const & *v1,* Vec3< U > const & *v2*)

Constructor from two vectors. The resulting quaternion represents the rotation between the vectors.

Parameters

v1	First vector
v2	Second vector

3.4.2 Member Function Documentation

3.4.2.1 float Geometrics::Quaternion::angle (Quaternion const & *toQuat*) const

Calculates the angle between the given and the underlying quaternion in 4D space. Has nothing to do with rotations in 3D space.

Parameters

toQuat	The quaternion to which the angle is calculated

Returns

The angle between the two quaternions

3.4.2.2 bool Geometrics::Quaternion::isNormalized ( ) const

Returns whether the Quaternion is normalized.

Returns

True, if normalized.

3.4.2.3 Quaternion Geometrics::Quaternion::lerp (Quaternion const & *destQt*, float *t*) const

Computes a [l]inear int[erp]olation between the given and the underlying quaternion and returns the resulting rotation as a new quaternion. This method is mainly used by SLERP, usually there is no application where to call it manually. It is necessary to normalize the quaternion beforehand!

Parameters

destQt	The quaternion on the other side of the interpolation
t	"Time", the interpolation value between 0 and 1

Returns

The resulting rotation as a quaternion

3.4.2.4 void Geometrics::Quaternion::normalize ( )

Normalizes the Quaternion in place (not a copy that is returned). This is necessary for almost all quaternion operations before executing.

3.4.2.5 Quaternion Geometrics::Quaternion::operator∗ (Quaternion const & *rOp*) const

Quaternion Multiplication Operator. Multiplication of two quaternions corresponds to a combined resulting rotation. Note that a quaternion multiplication is non-commutative. It is necessary to normalize the quaternion beforehand!

Parameters

rOp	Right hand side operand (Quaternion)

Returns

A new quaternion.

3.4.2.6 Quaternion Geometrics::Quaternion::operator+ (Quaternion const & *rOp*) const

Quaternion Addition Operator. Addition of two Quaternions does NOT result in an addition of the respective rotations. Read up quaternions! It is necessary to normalize the quaternion beforehand!

Parameters

rOp	Right hand side operand (Quaternion)

Returns

A new quaternion.

3.4.2.7 float Geometrics::Quaternion::rotAngleInDeg ( )

Returns the angle of the rotation represented by the quaternion. It is necessary to normalize the quaternion beforehand!

Returns

The angle of the rotation.

3.4.2.8 Quaternion Geometrics::Quaternion::slerp (Quaternion const & *destQt*, float *t*, float *eps* = `0.01`) const

Computes a [s]pherical [l]inear int[erp]olation between the given and the underlying quaternion and returns the resulting rotation as a new quaternion. It is necessary to normalize the quaternion beforehand!

Parameters

destQt	The quaternion of the other side of the interpolation
t	"Time", the interpolation value between 0 and 1
eps	Angular threshold where to begin with LERP

Returns

The resulting rotation as a quaternion

3.4.2.9 void Geometrics::Quaternion::toByteArray (byte * *bArray*) const

Serializes the quaternion. Make sure to allocate enough space for four floats.

Parameters

bArray	The byte array to be filled

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

- CPlusPlusFrameWork/Geometrics/Quaternion.h

3.5 Vec3< T > Struct Template Reference

Public Member Functions

- **Vec3** (T inX, T inY, T inZ)
- float **norm2** () const
- template<typename U >
  U **dot** (Vec3< U > const &v) const
- template<typename U >
  Vec3< U > **cross** (Vec3< U > const &v) const

Public Attributes

- T **x**
- T **y**
- T **z**

template<typename T> struct Vec3< T >

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

- CPlusPlusFrameWork/Geometrics/Vec3.h

Chapter 4

File Documentation

4.1 CPlusPlusFrameWork/Geometrics/Circle.h File Reference

Classes

- class Geometrics::Circle< T >

4.1.1 Detailed Description

Author

michael <rudolphmichael42@gmail.com>

Version

1.0.0

Copyright

Copyright 23. März 2014 University of Freiburg

4.1.2 LICENSE

4.1.3 DESCRIPTION

description

4.2 CPlusPlusFrameWork/Geometrics/Point.h File Reference

Classes

- class Geometrics::Point< T >
- class Geometrics::Point< float >

4.2.1 Detailed Description

Author

michael <rudolphmichael42@gmail.com>

Version

1.0.0

Copyright

Copyright 5. März 2014 University of Freiburg

4.2.2 LICENSE

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details at http://www.gnu.org/copyleft/gpl.html

4.2.3 DESCRIPTION

The Point class describes an _dim dimensional point. The point is stored as an array.