Geometrics

1.0

Generated by Doxygen 1.7.6.1

Mon Mar 10 2014 22:12:13

Contents

1 Class Index **1**

1.1 Class List 1

2 Class Documentation **3**

2.1 Geometrics::Quaternion Class Reference 3

2.1.1 Constructor & Destructor Documentation 4

2.1.1.1 Quaternion 4

2.1.1.2 Quaternion 4

2.1.1.3 Quaternion 4

2.1.2 Member Function Documentation 4

2.1.2.1 angle 4

2.1.2.2 isNormalized 5

2.1.2.3 lerp 5

2.1.2.4 normalize 5

2.1.2.5 operator∗ 5

2.1.2.6 operator+ 6

2.1.2.7 rotAngleInDeg 6

2.1.2.8 slerp 6

2.1.2.9 toByteArray 7

2.2 Vec3< T > Struct Template Reference 7

2.3 Geometrics::Vector< T > Class Template Reference 7

2.3.1 Constructor & Destructor Documentation 8

2.3.1.1 Vector 8

2.3.1.2 Vector 8

2.3.1.3 ∼Vector 8

2.3.2 Member Function Documentation 9

2.3.2.1 operator!= . . . 9
2.3.2.2 operator∗ . . . 9
2.3.2.3 operator∗= . . . 9
2.3.2.4 operator+ . . . 10
2.3.2.5 operator+= . . . 10
2.3.2.6 operator- . . . 10
2.3.2.7 operator-= . . . 11
2.3.2.8 operator== . . . 11
2.3.2.9 operator[] . . . 11
2.3.2.10 operator[] . . . 11
2.3.3 Member Data Documentation . . . 12
2.3.3.1 _coordinates . . . 12
2.3.3.2 _delocate . . . 12
2.3.3.3 _dim . . . 12

Chapter 1

Class Index

1.1 Class List

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

Geometrics::Quaternion 3
Vec3< T > 7
Geometrics::Vector< T > 7

Chapter 2

Class Documentation

2.1 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**

2.1.1 Constructor & Destructor Documentation

2.1.1.1 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)

2.1.1.2 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

2.1.1.3 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

2.1.2 Member Function Documentation

2.1.2.1 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

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

Returns whether the Quaternion is normalized.

Returns

True, if normalized.

2.1.2.3 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

2.1.2.4 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.

2.1.2.5 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.

2.1.2.6 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.

2.1.2.7 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.

2.1.2.8 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

2.1.2.9 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:

- Geometrics/Quaternion.h

2.2 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:

- Geometrics/Vec3.h

2.3 Geometrics::Vector< T > Class Template Reference

Public Member Functions

- Vector (T coordinates[], const int dimension)
- Vector (const int dim, const T value)
- virtual ~Vector ()
- bool operator== (const Vector &v)
- bool operator!= (const Vector &v)

- const Vector operator+ (const Vector &v)
- Vector & operator+= (const Vector &v)
- const Vector operator- (const Vector &v)
- Vector & operator-= (const Vector &v)
- const Vector operator∗ (const Vector &v)
- Vector & operator∗= (const Vector &v)
- T & operator[] (const int &i)
- const T & operator[] (const int &i) const

Private Attributes

- T ∗ _coordinates
- int _dim
- bool _delocate

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

2.3.1 Constructor & Destructor Documentation

2.3.1.1 template<class T = int> Geometrics::Vector< T >::Vector (T *coordinates[]*, const int *dimension*) [inline]

The first constructor.

Parameters

coordinates	The coordinates of the Vector.
dimension	The dimension of the Vector.

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

The second constructor

Parameters

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

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

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

2.3.2 Member Function Documentation

2.3.2.1 template<class T = int> bool Geometrics::Vector< T >::operator!= (const Vector< T > & *v*) [inline]

Overloading the != operator.

Parameters

v	The other Vector.

Returns

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

2.3.2.2 template<class T = int> const Vector Geometrics::Vector< T >::operator* (const Vector< T > & *v*) [inline]

Overloading the * operator. Calculate the scalar product of two vectors(v1, v2). Throw an assertion, if the dimension of the vectors are not the same.

Parameters

v	The other Vector.

Returns

Vector v3, where all coordinate i holds: v3[i] = v1[i] * v2[i].

2.3.2.3 template<class T = int> Vector& Geometrics::Vector< T >::operator*= (const Vector< T > & *v*) [inline]

Overloading the *= operator. Calculate the scalar product of two vectors(v1, v2). Throw an assertion, if the dimension of the vectors are not the same.

Parameters

v	The other Vector.

Returns

Vector v3, where all coordinate i holds: v3[i] = v1[i] ∗ v2[i].

2.3.2.4 template<class T = int> const Vector Geometrics::Vector< T >::operator+ (const Vector< T > & *v*) [inline]

Overloading the + operator. Add two vector v1 and v2. Throw an assertion, if the dimension of the vectors are not the same.

Parameters

v	The other Vector.

Returns

Vector v3, where all coordinate i holds: v3[i] = v1[i] + v2[i].

2.3.2.5 template<class T = int> Vector& Geometrics::Vector< T >::operator+= (const Vector< T > & *v*) [inline]

Overloading the += operator. Add the vector v to the vector, standing before the += operator. Throw an assertion, if the dimension of the vectors are not the same.

Parameters

v	The other Vector.

Returns

The modified vector, standing before the += operator.

2.3.2.6 template<class T = int> const Vector Geometrics::Vector< T >::operator- (const Vector< T > & *v*) [inline]

Overloading the - operator. Add two vector v1 and v2. Throw an assertion, if the dimension of the vectors are not the same.

Parameters

v	The other Vector.

Returns

Vector v3, where all coordinate i holds: v3[i] = v1[i] - v2[i].

2.3.2.7 template<class T = int> Vector& Geometrics::Vector< T >::operator-= (const Vector< T > & *v*) [inline]

Overloading the -= operator. Subtract the vector v from the vector, standing before the -= operator. Throw an assertion, if the dimension of the vectors are not the same.

Parameters

v	The other Vector.

Returns

The modified vector, standing before the -= operator.

2.3.2.8 template<class T = int> bool Geometrics::Vector< T >::operator== (const Vector< T > & *v*) [inline]

Overloading the == operator.

Parameters

v	The other Vector.

Returns

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

2.3.2.9 template<class T = int> T& Geometrics::Vector< 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.

2.3.2.10 template<class T = int> const T& Geometrics::Vector< 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.

2.3.3 Member Data Documentation

2.3.3.1 template<class T = int> T* Geometrics::Vector< T >::_coordinates [private]

The coordinates of the vector as array.

2.3.3.2 template<class T = int> bool Geometrics::Vector< T >::_delocate [private]

True, if an array was allocated with new.

2.3.3.3 template<class T = int> int Geometrics::Vector< T >::_dim [private]

The dimension of the vector.

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

- Geometrics/Vector.h