

# GenericMathTemplateLibrary Reference Manual

## 0.0.5

Generated by Doxygen 1.2.15

Thu Jun 13 18:17:01 2002



---

# Contents

<b>1</b>	<b>Generic Math Template Library</b>	<b>1</b>
1.1	Using This Reference Guide . . . . .	1
1.2	Quickly Understanding The GMTL API . . . . .	1
1.3	A Light Overview Of GMTL . . . . .	2
<b>2</b>	<b>GenericMathTemplateLibrary Module Index</b>	<b>5</b>
2.1	GenericMathTemplateLibrary Modules . . . . .	5
<b>3</b>	<b>GenericMathTemplateLibrary Namespace Index</b>	<b>7</b>
3.1	GenericMathTemplateLibrary Namespace List . . . . .	7
<b>4</b>	<b>GenericMathTemplateLibrary Hierarchical Index</b>	<b>9</b>
4.1	GenericMathTemplateLibrary Class Hierarchy . . . . .	9
<b>5</b>	<b>GenericMathTemplateLibrary Compound Index</b>	<b>11</b>
5.1	GenericMathTemplateLibrary Compound List . . . . .	11
<b>6</b>	<b>GenericMathTemplateLibrary File Index</b>	<b>13</b>
6.1	GenericMathTemplateLibrary File List . . . . .	13
<b>7</b>	<b>GenericMathTemplateLibrary Page Index</b>	<b>15</b>
7.1	GenericMathTemplateLibrary Related Pages . . . . .	15
<b>8</b>	<b>GenericMathTemplateLibrary Module Documentation</b>	<b>17</b>
8.1	Global Flags: Xelt, XYZ, etc... . . . . .	17

---

8.2	C Math Abstraction: sin, cos, tan, Min, Max, PI . . . . .	20
8.3	Abstract Data Types: Matrix, Vec, Quat, Coord, Sphere, Plane . . . . .	38
8.4	Mathematical Operations: add(...), sub(...), mul(...), div(...), invert(...), dot(...), cross(...) . . . . .	40
8.5	Spacial Transformers: xform( ... ), operator( ... ). . . . .	89
8.6	Comparison: isEqual(...), isEquiv(...), ==, != . . . . .	99
8.7	Generators: make( ... ), set( ... ). . . . .	118
8.8	Interpolation: lerp(...), slerp(...) . . . . .	158
8.9	Output Stream Methods: operator<<( ... ). . . . .	163
8.10	Template Metaprogramming Utilities . . . . .	168
8.11	Template Metaprogramming Utilities (Helpers) . . . . .	169
<b>9</b>	<b>GenericMathTemplateLibrary Namespace Documentation</b>	<b>171</b>
9.1	gmtl Namespace Reference . . . . .	171
9.2	gmtl::Math Namespace Reference . . . . .	233
<b>10</b>	<b>GenericMathTemplateLibrary Class Documentation</b>	<b>237</b>
10.1	gmtl::AABBox< DATA_TYPE > Class Template Reference . . . . .	237
10.2	gmtl::AxisAngle< DATA_TYPE > Class Template Reference . . . . .	245
10.3	gmtl::CompareIndexPointProjections Struct Reference . . . . .	252
10.4	gmtl::Coord< POS_TYPE, ROT_TYPE > Class Template Reference . . . . .	254
10.5	gmtl::Eigen Class Reference . . . . .	260
10.6	gmtl::EulerAngle< DATA_TYPE, ROTATION_ORDER > Class Template Reference . . . . .	281
10.7	gmtl::LineSeg< DATA_TYPE > Class Template Reference . . . . .	287
10.8	gmtl::Matrix< DATA_TYPE, ROWS, COLS > Class Template Reference . . . . .	294
10.9	gmtl::OOBox Class Reference . . . . .	308
10.10	gmtl::Plane< DATA_TYPE > Class Template Reference . . . . .	315
10.11	gmtl::Point< DATA_TYPE, SIZE > Class Template Reference . . . . .	322
10.12	gmtl::Quat< DATA_TYPE > Class Template Reference . . . . .	327
10.13	gmtl::RotationOrderBase Struct Reference . . . . .	333
10.14	gmtl::Sphere< DATA_TYPE > Class Template Reference . . . . .	335
10.15	gmtl::Tri< DATA_TYPE > Class Template Reference . . . . .	341
10.16	gmtl::Type2Type< T > Struct Template Reference . . . . .	346

10.17gmtl::Vec< DATA_TYPE, SIZE > Class Template Reference . . . . .	347
10.18gmtl::VecBase< DATA_TYPE, SIZE > Class Template Reference . . . . .	352
10.19gmtl::XYZ Struct Reference . . . . .	361
10.20gmtl::ZXY Struct Reference . . . . .	363
10.21gmtl::ZYX Struct Reference . . . . .	365
<b>11 GenericMathTemplateLibrary File Documentation</b>	<b>367</b>
11.1 AABox.h File Reference . . . . .	367
11.2 Assert.h File Reference . . . . .	368
11.3 AxisAngle.h File Reference . . . . .	369
11.4 AxisAngleOps.h File Reference . . . . .	370
11.5 Comparitors.h File Reference . . . . .	371
11.6 Containment.h File Reference . . . . .	372
11.7 Coord.h File Reference . . . . .	373
11.8 CoordOps.h File Reference . . . . .	374
11.9 Defines.h File Reference . . . . .	375
11.10Eigen.h File Reference . . . . .	376
11.11EulerAngle.h File Reference . . . . .	377
11.12EulerAngleOps.h File Reference . . . . .	378
11.13GaussPointsFit.h File Reference . . . . .	379
11.14Generate.h File Reference . . . . .	380
11.15gmtl.doxygen File Reference . . . . .	382
11.16gmtl.h File Reference . . . . .	383
11.17Intersection.h File Reference . . . . .	384
11.18LineSeg.h File Reference . . . . .	385
11.19LineSegOps.h File Reference . . . . .	386
11.20Math.h File Reference . . . . .	387
11.21Matrix.h File Reference . . . . .	388
11.22MatrixOps.h File Reference . . . . .	389
11.23Meta.h File Reference . . . . .	390
11.24OOBox.h File Reference . . . . .	391

11.25OpenSGConvert.h File Reference . . . . .	392
11.26Output.h File Reference . . . . .	393
11.27Plane.h File Reference . . . . .	394
11.28PlaneOps.h File Reference . . . . .	395
11.29Point.h File Reference . . . . .	396
11.30Quat.h File Reference . . . . .	397
11.31QuatOps.h File Reference . . . . .	398
11.32Sphere.h File Reference . . . . .	399
11.33SphereOps.h File Reference . . . . .	400
11.34Tri.h File Reference . . . . .	401
11.35TriOps.h File Reference . . . . .	402
11.36Vec.h File Reference . . . . .	403
11.37VecBase.h File Reference . . . . .	404
11.38VecOps.h File Reference . . . . .	405
11.39Version.h File Reference . . . . .	406
11.40Xforms.h File Reference . . . . .	410
 <b>12 GenericMathTemplateLibrary Page Documentation</b>	 <b>411</b>
12.1 Todo List . . . . .	411

---

## Chapter 1

# Generic Math Template Library

### 1.1 Using This Reference Guide

Welcome to GMTL. To use this reference guide effectively, we suggest you see the [Modules](#) section first. The [Modules](#) section provides the most intuitive navigation of the reference guide because this section is structured very similar GMTL. Be sure to read the GMTL Programmer's Guide (available on the [GMTL web site](#)) to understand the philosophy behind GMTL. Understanding abstractly what GMTL is and why it is designed this way will make your life with GMTL very easy. Lastly, you should subscribe to the mailing lists so that you can ask questions, or propose extensions to the library.

### 1.2 Quickly Understanding The GMTL API

The GMTL API has two aspects you should keep in mind. The *data* types, and the *operations* on the data.

All data types and operations are defined in the `gmtl` namespace. Thus all types must be prefixed with the `gmtl::` scope or a `using gmtl;` command can be used to bring all of the GMTL functionality into the local scope.

---

### 1.2.1 Supplied GMTL Math Types

GMTL comes with many math data types: Vec, Point, Matrix, Quat, Coord, Sphere. Please read the programmer's guide for more detailed information. Or read on for a light overview on what GMTL is.

## 1.3 A Light Overview Of GMTL

GMTL stands for (G)eneric (M)ath (T)emplate (L)ibrary. It is a math library designed to be high-performance, extensible, and generic. The design is based upon discussion with many experts in the field of computer graphics and virtual reality and is the culmination of many previous graphics math library efforts. GMTL gives the graphics programmer several core math types and a rich library of graphics/math operations on those types.

### 1.3.1 Design

The design of GMTL allows extensibility while maintaining a stable core. Core data types are separated from operations. This allows anyone to write their own math routines to extend or replace parts of the GMTL. This feature allows a very stable core set of math primitives that seldom change due to extensions, maintainance, or programmer error.

All math primitives in GMTL use generic programming techniques to give the programmer many options to define their data. For example, matrices and vectors can be any dimension and any type. GMTL suffers no loss of performance due to these generalities because the parameter choices made are bound at *compile time*.

### 1.3.2 Implementation

GMTL is implemented using generic programming and template metaprogramming. Generic programming allows selection by the user of size and type information for all data types in GMTL. For example, the generic Matrix type allows a programmer to select between any size ( $N \times M$ ) and any datatype (float, double, int...). The selection of these parameters is done through *template parameters*. To ease the use of these parameters, the system declares several typedefs that capture commonly used options.

Requested data types are statically bound and optimized by the compiler. The operations supplied with GMTL are implemented generically using a technique called



*template metaprogramming*. Template metaprogramming allows things such as loops to be unrolled and conditionals to be evaluated *by the compiler*. Things such as loops and conditionals are evaluated statically, rather than at runtime. In addition, advanced optimizations can be performed that do this such as eliminate temporary variables and other intermediate computations. The result is compiled code that can behave as fast (or faster) than using traditional hand-coding methods such as loop unrolling, etc...

### 1.3.3 Testing

GMTL has an integrated test suite included in the source code distribution. The suite tests GMTL for correctness as well as performance degradation. The GMTL developers have put much time and effort into the test suite because we think that it will ensure that the code stays stable when changes are made, and that changes don't introduce performance hits. The bottom line is, if any behaviour changes in GMTL we want to know about it before it bites us. As a result of this philosophy, any contributions to GMTL also need to be well tested. Submissions will not be accepted without tests for correctness and performance.



---

## Chapter 2

# GenericMathTemplateLibrary Module Index

### 2.1 GenericMathTemplateLibrary Modules

Here is a list of all modules:

Global Flags: Xelt, XYZ, etc...	17
C Math Abstraction: sin, cos, tan, Min, Max, PI	20
Abstract Data Types: Matrix, Vec, Quat, Coord, Sphere, Plane	38
Mathematical Operations: add(...), sub(...), mul(...), div(...), invert(...), dot(...), cross(...)	40
Spacial Transformers: xform( ... ), operator( ... )	89
Comparison: isEqual(...), isEquiv(...), ==, !=	99
Generators: make( ... ), set( ... )	118
Interpolation: lerp(...), slerp(...)	158
Output Stream Methods: operator<<( ... )	163
Template Metaprogramming Utilities	168
Template Metaprogramming Utilities (Helpers)	169

---



---

## Chapter 3

# GenericMathTemplateLibrary Namespace Index

### 3.1 GenericMathTemplateLibrary Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">gmtl</a>	.....	<a href="#">171</a>
<a href="#">gmtl::Math</a>	.....	<a href="#">233</a>



---

## Chapter 4

# GenericMathTemplateLibrary Hierarchical Index

### 4.1 GenericMathTemplateLibrary Class Hierarchy

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

gmtl::AABBox< DATA_TYPE > . . . . .	237
gmtl::CompareIndexPointProjections . . . . .	252
gmtl::Coord< POS_TYPE, ROT_TYPE > . . . . .	254
gmtl::Eigen . . . . .	260
gmtl::EulerAngle< DATA_TYPE, ROTATION_ORDER > . . . . .	281
gmtl::LineSeg< DATA_TYPE > . . . . .	287
gmtl::Matrix< DATA_TYPE, ROWS, COLS > . . . . .	294
gmtl::OOBox . . . . .	308
gmtl::Plane< DATA_TYPE > . . . . .	315
gmtl::Quat< DATA_TYPE > . . . . .	327
gmtl::RotationOrderBase . . . . .	333
gmtl::XYZ . . . . .	361
gmtl::ZXY . . . . .	363
gmtl::ZYX . . . . .	365
gmtl::Sphere< DATA_TYPE > . . . . .	335
gmtl::Tri< DATA_TYPE > . . . . .	341
gmtl::Type2Type< T > . . . . .	346
gmtl::VecBase< DATA_TYPE, SIZE > . . . . .	352
gmtl::Point< DATA_TYPE, SIZE > . . . . .	322
gmtl::Vec< DATA_TYPE, SIZE > . . . . .	347
gmtl::VecBase< DATA_TYPE, 4 > . . . . .	352
gmtl::AxisAngle< DATA_TYPE > . . . . .	245

---





---

## Chapter 5

# GenericMathTemplateLibrary Compound Index

### 5.1 GenericMathTemplateLibrary Compound List

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

<a href="#">gmtl::AABBox&lt; DATA_TYPE &gt;</a> (Describes an axially aligned box in 3D space)	<a href="#">237</a>
<a href="#">gmtl::AxisAngle&lt; DATA_TYPE &gt;</a> ( <a href="#">AxisAngle</a> : Represents a "twist about an axis" <a href="#">AxisAngle</a> is used to specify a rotation in 3-space)	<a href="#">245</a>
<a href="#">gmtl::CompareIndexPointProjections</a>	<a href="#">252</a>
<a href="#">gmtl::Coord&lt; POS_TYPE, ROT_TYPE &gt;</a> ( <a href="#">Coord</a> is a position/rotation pair)	<a href="#">254</a>
<a href="#">gmtl::Eigen</a>	<a href="#">260</a>
<a href="#">gmtl::EulerAngle&lt; DATA_TYPE, ROTATION_ORDER &gt;</a> ( <a href="#">EulerAngle</a> : Represents a group of euler angles)	<a href="#">281</a>
<a href="#">gmtl::LineSeg&lt; DATA_TYPE &gt;</a> (Describes a line segment)	<a href="#">287</a>
<a href="#">gmtl::Matrix&lt; DATA_TYPE, ROWS, COLS &gt;</a> ( <a href="#">Matrix</a> : 4x4 <a href="#">Matrix</a> class (OpenGL ordering))	<a href="#">294</a>
<a href="#">gmtl::OOBox</a>	<a href="#">308</a>
<a href="#">gmtl::Plane&lt; DATA_TYPE &gt;</a> ( <a href="#">Plane</a> : Defines a geometrical plane)	<a href="#">315</a>
<a href="#">gmtl::Point&lt; DATA_TYPE, SIZE &gt;</a> ( <a href="#">Point</a> Use points when you need to represent a position)	<a href="#">322</a>
<a href="#">gmtl::Quat&lt; DATA_TYPE &gt;</a> ( <a href="#">Quat</a> : Class to encapsulate quaternion behaviors)	<a href="#">327</a>
<a href="#">gmtl::RotationOrderBase</a> (Base class for Rotation orders)	<a href="#">333</a>
<a href="#">gmtl::Sphere&lt; DATA_TYPE &gt;</a> (Describes a sphere in 3D space by its center point and its radius)	<a href="#">335</a>
<a href="#">gmtl::Tri&lt; DATA_TYPE &gt;</a> (This class defines a triangle as a set of 3 points order in CCW fashion)	<a href="#">341</a>

---

<a href="#">gmtl::Type2Type&lt; T &gt;</a> (A lightweight identifier you can pass to overloaded functions to typefy them) . . . . .	346
<a href="#">gmtl::Vec&lt; DATA_TYPE, SIZE &gt;</a> (A representation of a vector with SIZE components using DATA_TYPE as the data type for each component)	347
<a href="#">gmtl::VecBase&lt; DATA_TYPE, SIZE &gt;</a> (Base type for vector-like objects including Points and Vectors) . . . . .	352
<a href="#">gmtl::XYZ</a> (XYZ Rotation order) . . . . .	361
<a href="#">gmtl::ZXY</a> (ZXY Rotation order) . . . . .	363
<a href="#">gmtl::ZYX</a> (ZYX Rotation order) . . . . .	365

---

## Chapter 6

# GenericMathTemplateLibrary File Index

### 6.1 GenericMathTemplateLibrary File List

Here is a list of all files with brief descriptions:

<a href="#">AABox.h</a>	367
<a href="#">Assert.h</a>	368
<a href="#">AxisAngle.h</a>	369
<a href="#">AxisAngleOps.h</a>	370
<a href="#">Comparitors.h</a>	371
<a href="#">Containment.h</a>	372
<a href="#">Coord.h</a>	373
<a href="#">CoordOps.h</a>	374
<a href="#">Defines.h</a>	375
<a href="#">Eigen.h</a>	376
<a href="#">EulerAngle.h</a>	377
<a href="#">EulerAngleOps.h</a>	378
<a href="#">GaussPointsFit.h</a>	379
<a href="#">Generate.h</a>	380
<a href="#">gmtl.doxygen</a> (This file for documentation purposes only (for doxygen generation))	382
<a href="#">gmtl.h</a>	383
<a href="#">Intersection.h</a>	384
<a href="#">LineSeg.h</a>	385
<a href="#">LineSegOps.h</a>	386
<a href="#">Math.h</a>	387
<a href="#">Matrix.h</a>	388

---

<a href="#">MatrixOps.h</a>	389
<a href="#">Meta.h</a>	390
<a href="#">OOBox.h</a>	391
<a href="#">OpenSGConvert.h</a> (GMTL to OpenSG conversion functions)	392
<a href="#">Output.h</a>	393
<a href="#">Plane.h</a>	394
<a href="#">PlaneOps.h</a>	395
<a href="#">Point.h</a>	396
<a href="#">Quat.h</a>	397
<a href="#">QuatOps.h</a>	398
<a href="#">Sphere.h</a>	399
<a href="#">SphereOps.h</a>	400
<a href="#">Tri.h</a>	401
<a href="#">TriOps.h</a>	402
<a href="#">Vec.h</a>	403
<a href="#">VecBase.h</a>	404
<a href="#">VecOps.h</a>	405
<a href="#">Version.h</a>	406
<a href="#">Xforms.h</a>	410

---

## Chapter 7

# GenericMathTemplateLibrary

## Page Index

### 7.1 GenericMathTemplateLibrary Related Pages

Here is a list of all related documentation pages:

Todo List . . . . .	<a href="#">411</a>
---------------------	---------------------



---

## Chapter 8

# GenericMathTemplateLibrary Module Documentation

### 8.1 Global Flags: Xelt, XYZ, etc...

Constant Static Global Flags.

#### Compounds

- struct [RotationOrderBase](#)  
*Base class for Rotation orders.*
- struct [XYZ](#)  
*XYZ Rotation order.*
- struct [ZXY](#)  
*ZXY Rotation order.*
- struct [ZYX](#)  
*ZYX Rotation order.*

#### Constants

- const float [GMTL\\_EPSILON](#) = 1.0e-6f
-

- const float `GMTL_MAT_EQUAL_EPSILON` = 0.001f
- const float `GMTL_VEC_EQUAL_EPSILON` = 0.0001f

## Enumerations

- enum `VectorIndex` { `Xelt` = 0, `Yelt` = 1, `Zelt` = 2, `Welt` = 3 }  
*use the values in this enum to index vector data types (such as `Vec`, `Point`, `Quat`).*
- enum `PlaneSide` { `ON_PLANE`, `POS_SIDE`, `NEG_SIDE` }  
*Used to describe where a point lies in relationship to a plane.*

### 8.1.1 Detailed Description

Constant Static Global Flags.

### 8.1.2 Enumeration Type Documentation

#### 8.1.2.1 enum `gmtl::PlaneSide`

Used to describe where a point lies in relationship to a plane.

`ON_PLANE` means the point lies on the plane. `POS_SIDE` means the point lies on the side that the normal points. `NEG_SIDE` means the point lies on the side away from the normal.

#### Enumeration values:

**`ON_PLANE`**

**`POS_SIDE`**

**`NEG_SIDE`**

Definition at line 61 of file `Defines.h`.

Referenced by `gmtl::whichSide()`.

```

62     {
63         ON_PLANE ,
64         POS_SIDE ,
65         NEG_SIDE
66     } ;

```



### 8.1.2.2 enum gmtl::VectorIndex

use the values in this enum to index vector data types (such as [Vec](#), [Point](#), [Quat](#)).

”Example (access elements in a Vec3f):”

```
Vec3f vec;  
vec[Xelt] = 1.0f;  
vec[Yelt] = 3.0f;  
vec[Zelt] = 2.0f;
```

Enumeration values:

**Xelt**

**Yelt**

**Zelt**

**Welt**

Definition at line 52 of file Defines.h.

```
52 { Xelt = 0, Yelt = 1, Zelt = 2, Welt = 3 };
```

## 8.1.3 Variable Documentation

### 8.1.3.1 const float gmtl::GMTL\_EPSILON = 1.0e-6f

Definition at line 72 of file Defines.h.

### 8.1.3.2 const float gmtl::GMTL\_MAT\_EQUAL\_EPSILON = 0.001f

Definition at line 73 of file Defines.h.

### 8.1.3.3 const float gmtl::GMTL\_VEC\_EQUAL\_EPSILON = 0.0001f

Definition at line 74 of file Defines.h.

## 8.2 C Math Abstraction: sin, cos, tan, Min, Max, PI

We've abstracted C math to be cross platform and typesafe.

### C Math Abstraction

- `template<typename T> T abs (T iValue)`
- `template<typename T> T ceil (T fValue)`
- `float ceil (float fValue)`
- `double ceil (double fValue)`
- `template<typename T> T floor (T fValue)`
- `float floor (float fValue)`
- `double floor (double fValue)`
- `template<typename T> T sign (int iValue)`
- `template<typename T> T zeroClamp (T value, T eps=T(0))`  
*Clamps the given value down to zero if it is within epsilon of zero.*
- `template<typename T> T aCos (T fValue)`
- `float aCos (float fValue)`
- `double aCos (double fValue)`
- `template<typename T> T aSin (T fValue)`
- `float aSin (float fValue)`
- `double aSin (double fValue)`
- `template<typename T> T aTan (T fValue)`
- `double aTan (double fValue)`
- `float aTan (float fValue)`
- `template<typename T> T atan2 (T fY, T fX)`
- `float aTan2 (float fY, float fX)`
- `double aTan2 (double fY, double fX)`
- `template<typename T> T cos (T fValue)`
- `float cos (float fValue)`
- `double cos (double fValue)`
- `template<typename T> T exp (T fValue)`
- `float exp (float fValue)`
- `double exp (double fValue)`
- `template<typename T> T log (T fValue)`
- `double log (double fValue)`
- `float log (float fValue)`
- `double pow (double fBase, double fExponent)`

- float **pow** (float fBase, float fExponent)
- template<typename T> T **sin** (T fValue)
- double **sin** (double fValue)
- float **sin** (float fValue)
- template<typename T> T **tan** (T fValue)
- double **tan** (double fValue)
- float **tan** (float fValue)
- template<typename T> T **sqr** (T fValue)
- template<typename T> T **sqrt** (T fValue)
- double **sqrt** (double fValue)
- float **unitRandom** ()  
*get a random number between 0 and 1.*
- float **rangeRandom** (float x1, float x2)  
*return a random number between x1 and x2 RETURNS: random number between x1 and x2.*
- float **deg2Rad** (float fVal)
- double **deg2Rad** (double fVal)
- float **rad2Deg** (float fVal)
- double **rad2Deg** (double fVal)
- template<class T> bool **isEqual** (const T &a, const T &b, const T &tolerance)  
*Is almost equal? test for equality within some tolerance...*
- template<class T> T **trunc** (T val)  
*cut off the digits after the decimal place.*
- template<class T> T **round** (T p)  
*round to nearest integer.*
- template<class T> T **Min** (const T &x, const T &y)  
*min returns the minimum of 2 values.*
- template<class T> T **Min** (const T &x, const T &y, const T &z)  
*min returns the minimum of 3 values.*
- template<class T> T **Min** (const T &w, const T &x, const T &y, const T &z)  
*min returns the minimum of 4 values.*
- template<class T> T **Max** (const T &x, const T &y)  
*max returns the maximum of 2 values.*
- template<class T> T **Max** (const T &x, const T &y, const T &z)

*max returns the maximum of 3 values.*

- `template<class T> T Max (const T &w, const T &x, const T &y, const T &z)`  
*max returns the maximum of 4 values.*
- `template<class T> T factorial (T rhs)`  
*Compute the factorial.*

## Mathematical constants

- `const float PI = 3.14159265358979323846f`
- `const float PI\_OVER\_2 = 1.57079632679489661923f`
- `const float PI\_OVER\_4 = 0.78539816339744830962f`

### 8.2.1 Detailed Description

We've abstracted C math to be cross platform and typesafe.

### 8.2.2 Function Documentation

#### 8.2.2.1 `template<typename T> T abs (T iValue) [inline]`

Definition at line 80 of file Math.h.

Referenced by `gmtl::invertFull()`, `gmtl::isOnVolume()`, `gmtl::Eigen::QLAlgorithm()`, `gmtl::set()`, `gmtl::TestIntersect()`, `gmtl::TestIntersectOBB()`, `gmtl::Eigen::TridiagonalN()`, and `gmtl::Math::zeroClamp()`.

```
81 {
82     return T( iValue >= ((T)0) ? iValue : -iValue );
83 }
```

#### 8.2.2.2 `double aCos (double fValue) [inline]`

Definition at line 149 of file Math.h.

Referenced by `gmtl::set()`, and `gmtl::setRot()`.

```

150 {
151     if ( -1.0 < fValue )
152     {
153         if ( fValue < 1.0 )
154             return double( ::acos( fValue ) );
155         else
156             return 0.0;
157     }
158     else
159     {
160         return (double)gmtl::Math::PI;
161     }
162 }

```

### 8.2.2.3 float aCos (float *fValue*) [inline]

Definition at line 134 of file Math.h.

```

135 {
136     if ( -1.0f < fValue )
137     {
138         if ( fValue < 1.0f )
139             return float( ::acosf( fValue ) );
140         else
141             return 0.0f;
142     }
143     else
144     {
145         return (float)gmtl::Math::PI;
146     }
147 }

```

### 8.2.2.4 template<typename T> T aCos (T *fValue*) [inline]

Referenced by gmtl::makeXRot(), gmtl::makeYRot(), gmtl::makeZRot(), and gmtl::slerp().

### 8.2.2.5 double aSin (double *fValue*) [inline]

Definition at line 182 of file Math.h.

Referenced by gmtl::set().

```

183 {

```

```

184     if ( -1.0 < fValue )
185     {
186         if ( fValue < 1.0 )
187             return double( ::asin( fValue ) );
188         else
189             return (double)-gmtl::Math::PI_OVER_2;
190     }
191     else
192     {
193         return (double)gmtl::Math::PI_OVER_2;
194     }
195 }

```

#### 8.2.2.6 float aSin (float *fValue*) [inline]

Definition at line 167 of file Math.h.

```

168 {
169     if ( -1.0f < fValue )
170     {
171         if ( fValue < 1.0f )
172             return float( ::asinf( fValue ) );
173         else
174             return (float)-gmtl::Math::PI_OVER_2;
175     }
176     else
177     {
178         return (float)gmtl::Math::PI_OVER_2;
179     }
180 }

```

#### 8.2.2.7 template<typename T> T aSin (T *fValue*) [inline]

#### 8.2.2.8 float aTan (float *fValue*) [inline]

Definition at line 204 of file Math.h.

```

205 {
206     return float( ::atanf( fValue ) );
207 }

```

**8.2.2.9 double aTan (double *fValue*) [inline]**

Definition at line 199 of file Math.h.

```
200 {  
201     return ::atan( fValue );  
202 }
```

**8.2.2.10 template<typename T> T aTan (T *fValue*) [inline]****8.2.2.11 double aTan2 (double *fY*, double *fX*) [inline]**

Definition at line 218 of file Math.h.

References gmtl::Math::atan2().

Referenced by gmtl::set().

```
219 {  
220     return double( ::atan2( fY, fX ) );  
221 }
```

**8.2.2.12 float aTan2 (float *fY*, float *fX*) [inline]**

Definition at line 213 of file Math.h.

```
214 {  
215     return float( ::atan2f( fY, fX ) );  
216 }
```

**8.2.2.13 template<typename T> T atan2 (T *fY*, T *fX*) [inline]**

Referenced by gmtl::Math::aTan2().

**8.2.2.14 double ceil (double *fValue*) [inline]**

Definition at line 91 of file Math.h.

References gmtl::Math::ceil().

```
92 {  
93     return double( ::ceil( fValue ) );  
94 }
```

**8.2.2.15 float ceil (float *fValue*) [inline]**

Definition at line 87 of file Math.h.

```
88 {  
89     return float( ::ceilf( fValue ) );  
90 }
```

**8.2.2.16 template<typename T> T ceil (T *fValue*) [inline]**

Referenced by gmtl::Math::ceil().

**8.2.2.17 double cos (double *fValue*) [inline]**

Definition at line 231 of file Math.h.

References gmtl::Math::cos().

Referenced by gmtl::exp(), gmtl::set(), and gmtl::setRot().

```
232 {  
233     return double( ::cos( fValue ) );  
234 }
```

**8.2.2.18 float cos (float *fValue*) [inline]**

Definition at line 226 of file Math.h.



```
227 {  
228     return float( ::cosf( fValue ) );  
229 }
```

#### 8.2.2.19 `template<typename T> T cos (T fValue) [inline]`

Referenced by `gmtl::Math::cos()`.

#### 8.2.2.20 `double deg2Rad (double fVal) [inline]`

Definition at line 342 of file `Math.h`.

```
343 {  
344     return double( fVal * (double)(gmtl::Math::PI/180.0) );  
345 }
```

#### 8.2.2.21 `float deg2Rad (float fVal) [inline]`

Definition at line 338 of file `Math.h`.

```
339 {  
340     return float( fVal * (float)(gmtl::Math::PI/180.0) );  
341 }
```

#### 8.2.2.22 `double exp (double fValue) [inline]`

Definition at line 242 of file `Math.h`.

References `gmtl::Math::exp()`.

```
243 {  
244     return double( ::exp( fValue ) );  
245 }
```

**8.2.2.23 float exp (float *fValue*) [inline]**

Definition at line 238 of file Math.h.

```
239 {  
240     return float( ::expf( fValue ) );  
241 }
```

**8.2.2.24 template<typename T> T exp (T *fValue*) [inline]**

Referenced by gmtl::Math::exp().

**8.2.2.25 template<class T> T factorial (T *rhs*) [inline]**

Compute the factorial.

give - an object who's type has operator++, operator=, operator<=, and operator \*= defined. it should be a single valued scalar type such as an int, float, double etc....

NOTE: This could be faster with a lookup table, but then wouldn't work templated :  
kevin

Definition at line 425 of file Math.h.

```
426 {  
427     T lhs = (T)1;  
428  
429     for( T x = (T)1; x <= rhs; ++x )  
430     {  
431         lhs *= x;  
432     }  
433  
434     return lhs;  
435 }
```

**8.2.2.26 double floor (double *fValue*) [inline]**

Definition at line 102 of file Math.h.

References gmtl::Math::floor().

```
103 {  
104     return double( ::floor( fValue ) );  
105 }
```

**8.2.2.27 float floor (float *fValue*)** [inline]

Definition at line 98 of file Math.h.

```
99 {  
100     return float( ::floorf( fValue ) );  
101 }
```

**8.2.2.28 template<typename T> T floor (T *fValue*)** [inline]

Referenced by gmtl::Math::floor(), and gmtl::Math::trunc().

**8.2.2.29 template<class T> bool isEqual (const T & *a*, const T & *b*, const T & *tolerance*)** [inline]

Is almost equal? test for equality within some tolerance...

@PRE: tolerance must be  $\geq 0$

Definition at line 362 of file Math.h.

References gmtlASSERT.

```
363 {  
364     gmtlASSERT( tolerance  $\geq$  (T)0 );  
365     return bool( gmtl::Math::abs( a - b )  $\leq$  tolerance );  
366 }
```

**8.2.2.30 float log (float *fValue*)** [inline]

Definition at line 253 of file Math.h.

```
254 {  
255     return float( ::logf( fValue ) );  
256 }
```

**8.2.2.31 double log (double *fValue*)** [inline]

Definition at line 249 of file Math.h.

References gmtl::Math::log().

```

250 {
251     return double( ::log( fValue ) );
252 }

```

#### 8.2.2.32 `template<typename T> T log (T fValue)` [inline]

Referenced by `gmtl::Math::log()`.

#### 8.2.2.33 `template<class T> T Max (const T & w, const T & x, const T & y, const T & z)` [inline]

max returns the maximum of 4 values.

Definition at line 414 of file Math.h.

References `gmtl::Math::Max()`.

```

415 {
416     return gmtl::Math::Max( gmtl::Math::Max( w, x ), gmtl::Math::Max( y, z ) );
417 }

```

#### 8.2.2.34 `template<class T> T Max (const T & x, const T & y, const T & z)` [inline]

max returns the maximum of 3 values.

Definition at line 408 of file Math.h.

References `gmtl::Math::Max()`.

```

409 {
410     return Max( gmtl::Math::Max( x, y ), z );
411 }

```

#### 8.2.2.35 `template<class T> T Max (const T & x, const T & y)` [inline]

max returns the maximum of 2 values.

Definition at line 402 of file Math.h.

Referenced by `gmtl::Math::Max()`.

```

403 {
404     return ( x > y ) ? x : y;
405 }

```

#### 8.2.2.36 `template<class T> T Min (const T & w, const T & x, const T & y, const T & z) [inline]`

min returns the minimum of 4 values.

Definition at line 395 of file Math.h.

References gmtl::Math::Min().

Referenced by gmtl::identity(), gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::Matrix(), gmtl::setScale(), and gmtl::zero().

```

396 {
397     return gmtl::Math::Min( gmtl::Math::Min( w, x ), gmtl::Math::Min( y, z ) );
398 }

```

#### 8.2.2.37 `template< class T> T Min (const T & x, const T & y, const T & z) [inline]`

min returns the minimum of 3 values.

Definition at line 389 of file Math.h.

References gmtl::Math::Min().

```

390 {
391     return Min( gmtl::Math::Min( x, y ), z );
392 }

```

#### 8.2.2.38 `template<class T> T Min (const T & x, const T & y) [inline]`

min returns the minimum of 2 values.

Definition at line 383 of file Math.h.

Referenced by gmtl::Math::Min().

```
384 {  
385     return ( x > y ) ? y : x;  
386 }
```

#### **8.2.2.39 float pow (float *fBase*, float *fExponent*)** [inline]

Definition at line 262 of file Math.h.

```
263 {  
264     return float( ::powf( fBase, fExponent ) );  
265 }
```

#### **8.2.2.40 double pow (double *fBase*, double *fExponent*)** [inline]

Definition at line 258 of file Math.h.

```
259 {  
260     return double( ::pow( fBase, fExponent ) );  
261 }
```

#### **8.2.2.41 double rad2Deg (double *fVal*)** [inline]

Definition at line 351 of file Math.h.

```
352 {  
353     return double( fVal * (double)(180.0/gmtl::Math::PI) );  
354 }
```

#### **8.2.2.42 float rad2Deg (float *fVal*)** [inline]

Definition at line 347 of file Math.h.

```
348 {  
349     return float( fVal * (float)(180.0/gmtl::Math::PI) );  
350 }
```

**8.2.2.43 float rangeRandom (float x1, float x2) [inline]**

return a random number between x1 and x2 RETURNS: random number between x1 and x2.

Definition at line 323 of file Math.h.

References gmtl::Math::unitRandom().

```

324 {
325     float r = gmtl::Math::unitRandom();
326     float size = x2 - x1;
327     return float( r * size + x1 );
328 }
```

**8.2.2.44 template<class T> T round (T p) [inline]**

round to nearest integer.

Definition at line 376 of file Math.h.

```

377 {
378     return T( gmtl::Math::floor( p + (T)0.5 ) );
379 }
```

**8.2.2.45 template<typename T> T sign (int iValue) [inline]**

Definition at line 108 of file Math.h.

```

109 {
110     return ( iValue > ((T)0) ? ((T)+1) : ( iValue < ((T)0) ? ((T)-1) : ((T)0) );
111 }
```

**8.2.2.46 float sin (float fValue) [inline]**

Definition at line 274 of file Math.h.

Referenced by gmtl::exp(), gmtl::set(), and gmtl::setRot().

```

275 {
276     return float( ::sinf( fValue ) );
277 }
```

**8.2.2.47 double sin (double *fValue*) [inline]**

Definition at line 269 of file Math.h.

References gmtl::Math::sin().

```
270 {  
271     return double( ::sin( fValue ) );  
272 }
```

**8.2.2.48 template<typename T> T sin (T *fValue*) [inline]**

Referenced by gmtl::Math::sin(), and gmtl::slerp().

**8.2.2.49 template<typename T> T sqr (T *fValue*) [inline]**

Definition at line 294 of file Math.h.

```
295 {  
296     return T( fValue * fValue );  
297 }
```

**8.2.2.50 double sqrt (double *fValue*) [inline]**

Definition at line 306 of file Math.h.

References gmtl::Math::sqrt().

Referenced by gmtl::exp(), gmtl::length(), gmtl::log(), gmtl::makeVolume(), gmtl::Eigen::QLAlgorithm(), gmtl::set(), gmtl::Eigen::Tridiagonal3(), gmtl::Eigen::Tridiagonal4(), and gmtl::Eigen::TridiagonalN().

```
307 {  
308     return double( ::sqrt( fValue ) );  
309 }
```



**8.2.2.51** `template<typename T> T sqrt (T fValue)` [inline]

Definition at line 301 of file Math.h.

Referenced by `gmtl::Math::sqrt()`.

```
302 {  
303     return T( ::sqrtf( ((float)fValue) ) );  
304 }
```

**8.2.2.52** `float tan (float fValue)` [inline]

Definition at line 287 of file Math.h.

```
288 {  
289     return float( ::tanf( fValue ) );  
290 }
```

**8.2.2.53** `double tan (double fValue)` [inline]

Definition at line 282 of file Math.h.

References `gmtl::Math::tan()`.

```
283 {  
284     return double( ::tan( fValue ) );  
285 }
```

**8.2.2.54** `template<typename T> T tan (T fValue)` [inline]

Referenced by `gmtl::Math::tan()`.

**8.2.2.55** `template<class T> T trunc (T val)` [inline]

cut off the digits after the decimal place.

Definition at line 370 of file Math.h.

References `gmtl::Math::floor()`.

```

371 {
372     return T( (val < ((T)0)) ? gmtl::Math::ceil( val ) : gmtl::Math::floor( val ) );
373 }

```

#### 8.2.2.56 float unitRandom() [inline]

get a random number between 0 and 1.

**Postcondition:**

returns number between 0 and 1

Definition at line 315 of file Math.h.

Referenced by gmtl::Math::rangeRandom().

```

316 {
317     return float(::rand())/float(RAND_MAX);
318 }

```

#### 8.2.2.57 template<typename T> T zeroClamp (T value, T eps = T(0)) [inline]

Clamps the given value down to zero if it is within epsilon of zero.

**Parameters:**

*value* the value to clamp

*eps* the epsilon tolerance or zero by default

**Returns:**

zero if the value is close to 0, the value otherwise

Definition at line 122 of file Math.h.

References gmtl::Math::abs().

```

123 {
124     return ( (gmtl::Math::abs(value) <= eps) ? T(0) : value );
125 }

```

### 8.2.3 Variable Documentation

**8.2.3.1** `const float gmtl::Math::PI = 3.14159265358979323846f`

Definition at line 69 of file Math.h.

**8.2.3.2** `const float gmtl::Math::PI_OVER_2 = 1.57079632679489661923f`

Definition at line 70 of file Math.h.

**8.2.3.3** `const float gmtl::Math::PI_OVER_4 = 0.78539816339744830962f`

Definition at line 71 of file Math.h.

## 8.3 Abstract Data Types: Matrix, Vec, Quat, Coord, Sphere, Plane

GMTL comes with many math data types: Vec, Point, Matrix, Quat, Coord, Sphere.

### Compounds

- class [AABox](#)  
*Describes an axially aligned box in 3D space.*
- class [AxisAngle](#)  
*[AxisAngle](#): Represents a "twist about an axis" [AxisAngle](#) is used to specify a rotation in 3-space.*
- class [Coord](#)  
*coord is a position/rotation pair.*
- class [EulerAngle](#)  
*[EulerAngle](#): Represents a group of euler angles.*
- class [Matrix](#)  
*[Matrix](#): 4x4 [Matrix](#) class (OpenGL ordering).*
- class [Plane](#)  
*[Plane](#): Defines a geometrical plane.*
- class [Point](#)  
*[Point](#) Use points when you need to represent a position.*
- class [Quat](#)  
*[Quat](#): Class to encapsulate quaternion behaviors.*
- class [Sphere](#)  
*Describes a sphere in 3D space by its center point and its radius.*
- class [Tri](#)  
*This class defines a triangle as a set of 3 points order in CCW fashion.*
- class [Vec](#)

*A representation of a vector with `SIZE` components using `DATA_TYPE` as the data type for each component.*

### **8.3.1 Detailed Description**

GMTL comes with many math data types: Vec, Point, Matrix, Quat, Coord, Sphere.

## 8.4 Mathematical Operations: add(...), sub(...), mul(...), div(...), invert(...), dot(...), cross(...)

Implements fundamental mathematical operations such as +, -, \*, invert, dot product.

### Matrix Operations

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & identity (Matrix< DATA_TYPE, ROWS, COLS > &result)`  
*Make identity matrix out the matrix.*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & zero (Matrix< DATA_TYPE, ROWS, COLS > &result)`  
*zero out the matrix.*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned INTERNAL, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & mult (Matrix< DATA_TYPE, ROWS, COLS > &result, const Matrix< DATA_TYPE, ROWS, INTERNAL > &lhs, const Matrix< DATA_TYPE, INTERNAL, COLS > &rhs)`  
*matrix multiply.*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned INTERNAL, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > operator * (const Matrix< DATA_TYPE, ROWS, INTERNAL > &lhs, const Matrix< DATA_TYPE, INTERNAL, COLS > &rhs)`  
*matrix \* matrix.*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & sub (Matrix< DATA_TYPE, ROWS, COLS > &result, const Matrix< DATA_TYPE, ROWS, COLS > &lhs, const Matrix< DATA_TYPE, ROWS, COLS > &rhs)`  
*matrix subtraction (algebraic operation for matrix).*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & add (Matrix< DATA_TYPE, ROWS, COLS > &result, const Matrix< DATA_TYPE, ROWS, COLS > &lhs, const Matrix< DATA_TYPE, ROWS, COLS > &rhs)`

*matrix addition (algebraic operation for matrix).*

- `template<typename DATA_TYPE, unsigned SIZE> Matrix< DATA_TYPE, SIZE, SIZE > & postMult (Matrix< DATA_TYPE, SIZE, SIZE > &result, const Matrix< DATA_TYPE, SIZE, SIZE > &operand)`

*matrix postmultiply.*

- `template<typename DATA_TYPE, unsigned SIZE> Matrix< DATA_TYPE, SIZE, SIZE > & preMult (Matrix< DATA_TYPE, SIZE, SIZE > &result, const Matrix< DATA_TYPE, SIZE, SIZE > &operand)`

*matrix preMultiply.*

- `template<typename DATA_TYPE, unsigned SIZE> Matrix< DATA_TYPE, SIZE, SIZE > & operator *= (Matrix< DATA_TYPE, SIZE, SIZE > &result, const Matrix< DATA_TYPE, SIZE, SIZE > &operand)`

*matrix postmult (operator \* =).*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & mult (Matrix< DATA_TYPE, ROWS, COLS > &result, const Matrix< DATA_TYPE, ROWS, COLS > &mat, float scalar)`

*matrix scalar mult.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & mult (Matrix< DATA_TYPE, ROWS, COLS > &result, DATA_TYPE scalar)`

*matrix scalar mult.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & operator *= (Matrix< DATA_TYPE, ROWS, COLS > &result, DATA_TYPE scalar)`

*matrix scalar mult (operator \* =).*

- `template<typename DATA_TYPE, unsigned SIZE> Matrix< DATA_TYPE, SIZE, SIZE > & transpose (Matrix< DATA_TYPE, SIZE, SIZE > &result)`

*matrix transpose in place.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & transpose (Matrix< DATA_TYPE, ROWS, COLS > &result, const Matrix< DATA_TYPE, COLS, ROWS > &source)`

*matrix transpose from one type to another (i.e.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & invertFull (Matrix< DATA_TYPE, ROWS, COLS > &result, const Matrix< DATA_TYPE, ROWS, COLS > &src)`

*full matrix inversion.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & invert (Matrix< DATA_TYPE, ROWS, COLS > &result, const Matrix< DATA_TYPE, ROWS, COLS > &src)`

*smart matrix inversion.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & invert (Matrix< DATA_TYPE, ROWS, COLS > &result)`

*smart matrix inversion (in place) Does matrix inversion by intelligently selecting what type of inversion to use depending on the types of operations your Matrix has been through.*

## Plane Operations

- `template<class DATA_TYPE> DATA_TYPE distance (const Plane< DATA_TYPE > &plane, const Point< DATA_TYPE, 3 > &pt)`

*Computes the distance from the plane to the point.*

- `template<class DATA_TYPE> PlaneSide whichSide (const Plane< DATA_TYPE > &plane, const Point< DATA_TYPE, 3 > &pt)`

*Determines which side of the plane the given point lies.*

- `template<class DATA_TYPE> PlaneSide whichSide (const Plane< DATA_TYPE > &plane, const Point< DATA_TYPE, 3 > &pt, const DATA_TYPE &eps)`

*Determines which side of the plane the given point lies with the given epsilon tolerance.*

- `template<class DATA_TYPE> DATA_TYPE findNearestPt (const Plane< DATA_TYPE > &plane, const Point< DATA_TYPE, 3 > &pt, Point< DATA_TYPE, 3 > &result)`

*Finds the point on the plane that is nearest to the given point.*



## Quat Operations

- `template<typename DATA_TYPE> Quat< DATA_TYPE > & mult (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`  
*product of two quaternions (quaternion product) multiplication of quats is much like multiplication of typical complex numbers.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > operator * (const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`  
*product of two quaternions (quaternion product).*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & operator *= (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q2)`  
*quaternion postmult.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & negate (Quat< DATA_TYPE > &result)`  
*Vector negation - negate each element in the quaternion vector.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > operator- (const Quat< DATA_TYPE > &quat)`  
*Vector negation - (operator-) return a temporary that is the negative of the given quat.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & mult (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q, DATA_TYPE s)`  
*vector scalar multiplication.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > operator * (const Quat< DATA_TYPE > &q, DATA_TYPE s)`  
*vector scalar multiplication.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & operator *= (Quat< DATA_TYPE > &q, DATA_TYPE s)`  
*vector scalar multiplication.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & div (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`  
*quotient of two quaternions.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & div (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q, DATA_TYPE s)`  
*quaternion vector scale.*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > operator/ (const Quat< DATA_TYPE > &q, DATA_TYPE s)`  
*vector scalar division.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & operator/= (const Quat< DATA_TYPE > &q, DATA_TYPE s)`  
*vector scalar division.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & add (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`  
*vector addition.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > operator+ (const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`  
*vector addition.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & operator+= (Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`  
*vector addition.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & sub (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`  
*vector subtraction.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > operator- (const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`  
*vector subtraction.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & operator-= (Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`  
*vector subtraction.*
- `template<typename DATA_TYPE> DATA_TYPE dot (const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`  
*vector dot product between two quaternions.*
- `template<typename DATA_TYPE> DATA_TYPE lengthSquared (const Quat< DATA_TYPE > &q)`  
*quaternion "norm" (also known as vector length squared) using this can be faster than using length for some operations...*

- `template<typename DATA_TYPE> DATA_TYPE length (const Quat< DATA_TYPE > &q)`

*quaternion "absolute" (also known as vector length or magnitude) using this can be faster than using `length` for some operations...*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > & normalize (Quat< DATA_TYPE > &result)`

*set self to the normalized quaternion of self.*

- `template<typename DATA_TYPE> bool isNormalized (const Quat< DATA_TYPE > &q1, const DATA_TYPE eps=(DATA_TYPE) 0.0001f)`

*Determines if the given vector is normalized within the given tolerance.*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > & conj (Quat< DATA_TYPE > &result)`

*quaternion complex conjugate.*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > & invert (Quat< DATA_TYPE > &result)`

*quaternion multiplicative inverse.*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > & exp (Quat< DATA_TYPE > &result)`

*complex exponentiation.*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > & log (Quat< DATA_TYPE > &result)`

*complex logarithm.*

- `template<typename DATA_TYPE> void squad (Quat< DATA_TYPE > &result, DATA_TYPE t, const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2, const Quat< DATA_TYPE > &a, const Quat< DATA_TYPE > &b)`

*WARNING: not implemented (do not use).*

- `template<typename DATA_TYPE> void meanTangent (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2, const Quat< DATA_TYPE > &q3)`

*WARNING: not implemented (do not use).*

## Triangle Operations

- `template<class DATA_TYPE> Point< DATA_TYPE, 3 > center (const Tri< DATA_TYPE > &tri)`

*Computes the point at the center of the given triangle.*

- `template<class DATA_TYPE> Vec< DATA_TYPE, 3 > normal (const Tri< DATA_TYPE > &tri)`

*Computes the normal for this triangle.*

## Vector/Point Operations

- `template<typename DATA_TYPE, unsigned SIZE> Vec< DATA_TYPE, SIZE > operator- (const VecBase< DATA_TYPE, SIZE > &v1)`

*Negates v1.*

- `template<class DATA_TYPE, unsigned SIZE> VecBase< DATA_TYPE, SIZE > & operator+= (VecBase< DATA_TYPE, SIZE > &v1, const VecBase< DATA_TYPE, SIZE > &v2)`

*Adds v2 to v1 and stores the result in v1.*

- `template<class DATA_TYPE, unsigned SIZE> VecBase< DATA_TYPE, SIZE > operator+ (const VecBase< DATA_TYPE, SIZE > &v1, const VecBase< DATA_TYPE, SIZE > &v2)`

*Adds v2 to v1 and returns the result.*

- `template<class DATA_TYPE, unsigned SIZE> VecBase< DATA_TYPE, SIZE > & operator-= (VecBase< DATA_TYPE, SIZE > &v1, const VecBase< DATA_TYPE, SIZE > &v2)`

*Subtracts v2 from v1 and stores the result in v1.*

- `template<class DATA_TYPE, unsigned SIZE> Vec< DATA_TYPE, SIZE > operator- (const VecBase< DATA_TYPE, SIZE > &v1, const VecBase< DATA_TYPE, SIZE > &v2)`

*Subtracts v2 from v1 and returns the result.*

- `template<class DATA_TYPE, unsigned SIZE, class SCALAR_TYPE> VecBase< DATA_TYPE, SIZE > & operator *= (VecBase< DATA_TYPE, SIZE > &v1, const SCALAR_TYPE &scalar)`

*Multiplies v1 by a scalar value and stores the result in v1.*

- `template<class DATA_TYPE, unsigned SIZE, class SCALAR_TYPE> VecBase< DATA_TYPE, SIZE > operator * (const VecBase< DATA_TYPE, SIZE > &v1, const SCALAR_TYPE &scalar)`

*Multiplies v1 by a scalar value and returns the result.*

- `template<class DATA_TYPE, unsigned SIZE, class SCALAR_TYPE> VecBase< DATA_TYPE, SIZE > operator * (const SCALAR_TYPE &scalar, const VecBase< DATA_TYPE, SIZE > &v1)`

*Multiplies v1 by a scalar value and returns the result.*

- `template<class DATA_TYPE, unsigned SIZE, class SCALAR_TYPE> VecBase< DATA_TYPE, SIZE > & operator /= (VecBase< DATA_TYPE, SIZE > &v1, const SCALAR_TYPE &scalar)`

*Divides v1 by a scalar value and stores the result in v1.*

- `template<class DATA_TYPE, unsigned SIZE, class SCALAR_TYPE> VecBase< DATA_TYPE, SIZE > operator / (const VecBase< DATA_TYPE, SIZE > &v1, const SCALAR_TYPE &scalar)`

*Divides v1 by a scalar value and returns the result.*

## Vector Operations

- `template<class DATA_TYPE, unsigned SIZE> DATA_TYPE dot (const Vec< DATA_TYPE, SIZE > &v1, const Vec< DATA_TYPE, SIZE > &v2)`

*Computes dot product of v1 and v2 and returns the result.*

- `template<class DATA_TYPE, unsigned SIZE> DATA_TYPE length (const Vec< DATA_TYPE, SIZE > &v1)`

*Computes the length of the given vector.*

- `template<class DATA_TYPE, unsigned SIZE> DATA_TYPE lengthSquared (const Vec< DATA_TYPE, SIZE > &v1)`

*Computes the square of the length of the given vector.*

- `template<class DATA_TYPE, unsigned SIZE> DATA_TYPE normalize (Vec< DATA_TYPE, SIZE > &v1)`

*Normalizes the given vector in place causing it to be of unit length.*

- `template<class DATA_TYPE, unsigned SIZE> bool isNormalized (const Vec< DATA_TYPE, SIZE > &v1, const DATA_TYPE eps=(DATA_TYPE) 0.0001)`

*Determines if the given vector is normalized within the given tolerance.*

- `template<class DATA_TYPE> Vec< DATA_TYPE, 3 > cross (const Vec< DATA_TYPE, 3 > &v1, const Vec< DATA_TYPE, 3 > &v2)`

*Computes the cross product between v1 and v2 and returns the result.*

- `template<class DATA_TYPE> Vec< DATA_TYPE, 3 > & cross (Vec< DATA_TYPE, 3 > &result, const Vec< DATA_TYPE, 3 > &v1, const Vec< DATA_TYPE, 3 > &v2)`

*Computes the cross product between v1 and v2 and stores the result in result.*

### 8.4.1 Detailed Description

Implements fundamental mathematical operations such as +, -, \*, invert, dot product.

### 8.4.2 Function Documentation

- 8.4.2.1** `template<typename DATA_TYPE> Quat<DATA_TYPE>& add (Quat< DATA_TYPE > & result, const Quat< DATA_TYPE > & q1, const Quat< DATA_TYPE > & q2)`

vector addition.

**See also:**

[Quat](#)

Definition at line 247 of file QuatOps.h.

```

248     {
249         result[0] = q1[0] + q2[0];
250         result[1] = q1[1] + q2[1];
251         result[2] = q1[2] + q2[2];
252         result[3] = q1[3] + q2[3];
253         return result;
254     }
```

**8.4.2.2** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS>`  
`Matrix<DATA_TYPE, ROWS, COLS> & add (Matrix< DATA_TYPE,`  
`ROWS, COLS > & result, const Matrix< DATA_TYPE, ROWS,`  
`COLS > & lhs, const Matrix< DATA_TYPE, ROWS, COLS > & rhs)`  
`[inline]`

matrix addition (algebraic operation for matrix).

@PRE: if lhs is  $m \times n$ , and rhs is  $m \times n$ , then result is  $m \times n$  (mult func undefined otherwise) @POST: returns a  $m \times n$  matrix TODO: **enforce the sizes with templates...**

Definition at line 162 of file MatrixOps.h.

Referenced by `gmtl::operator+()`, and `gmtl::operator+=()`.

```

165  {
166      // p. 150 Numerical Analysis (second ed.)
167      // if A is m x n, and B is m x n, then AB is m x n
168      // (A - B)ij = (a)ij + (b)ij      (where: 1 <= i <= m, 1 <= j <= n)
169      for (unsigned int i = 0; i < ROWS; ++i)          // 1 <= i <= m
170          for (unsigned int j = 0; j < COLS; ++j)      // 1 <= j <= n
171              result( i, j ) = lhs( i, j ) + rhs( i, j );
172
173      return result;
174  }
```

**8.4.2.3** `template<class DATA_TYPE> Point<DATA_TYPE, 3> center (const`  
`Tri< DATA_TYPE > & tri)`

Computes the point at the center of the given triangle.

**Parameters:**

*tri* the triangle to find the center of

**Returns:**

the point at the center of the triangle

Definition at line 55 of file TriOps.h.

Referenced by `gmtl::Sphere< DATA_TYPE >::setCenter()`, and `gmtl::Sphere< DATA_TYPE >::Sphere()`.

```

56  {
57      const float one_third = (1.0f/3.0f);
58      return (tri[0] + tri[1] + tri[2]) * one_third;
59  }
```

#### 8.4.2.4 `template<typename DATA_TYPE> Quat<DATA_TYPE>& conj` `(Quat< DATA_TYPE > & result)`

quaternion complex conjugate.

**Postcondition:**

set result to the complex conjugate of result.

$q* = [s, -v]$

$result'[x,y,z,w] == result[-x,-y,-z,w]$

**See also:**

[Quat](#)

Definition at line 396 of file QuatOps.h.

References `gmtl::Xelt`, `gmtl::Yelt`, and `gmtl::Zelt`.

Referenced by `gmtl::div()`, `gmtl::invert()`, and `gmtl::makeConj()`.

```

397     {
398         result[Xelt] = -result[Xelt];
399         result[Yelt] = -result[Yelt];
400         result[Zelt] = -result[Zelt];
401         return result;
402     }
```

#### 8.4.2.5 `template<class DATA_TYPE> Vec<DATA_TYPE,3>& cross (Vec<` `DATA_TYPE, 3 > & result, const Vec< DATA_TYPE, 3 > & v1, const` `Vec< DATA_TYPE, 3 > & v2)`

Computes the cross product between v1 and v2 and stores the result in result.

The result is also returned by reference. Use this when you want to reuse an existing [Vec](#) to store the result. Note that this only applies to 3-dimensional vectors.

**Postcondition:**

$result = v1 \times v2$

**Parameters:**

*result* filled with the result of the cross product between v1 and v2

*v1* the first vector

*v2* the second vector

**Returns:**

a reference to result for convenience



Definition at line 388 of file VecOps.h.

References gmtl::VecBase< DATA\_TYPE, SIZE >::set(), gmtl::Xelt, gmtl::Yelt, and gmtl::Zelt.

Referenced by gmtl::Plane< DATA\_TYPE >::Plane().

```

390 {
391     result.set( (v1[Yelt]*v2[Zelt]) - (v1[Zelt]*v2[Yelt]),
392               (v1[Zelt]*v2[Xelt]) - (v1[Xelt]*v2[Zelt]),
393               (v1[Xelt]*v2[Yelt]) - (v1[Yelt]*v2[Xelt]) );
394     return result;
395 }
```

#### 8.4.2.6 template<class DATA\_TYPE> Vec<DATA\_TYPE,3> cross (const Vec<DATA\_TYPE, 3 > & v1, const Vec< DATA\_TYPE, 3 > & v2)

Computes the cross product between v1 and v2 and returns the result.

Note that this only applies to 3-dimensional vectors.

##### Postcondition:

result = v1 x v2

##### Parameters:

v1 the first vector

v2 the second vector

##### Returns:

the result of the cross product between v1 and v2

Definition at line 366 of file VecOps.h.

References gmtl::Xelt, gmtl::Yelt, and gmtl::Zelt.

Referenced by gmtl::makeXRot(), gmtl::makeYRot(), gmtl::makeZRot(), gmtl::normal(), and gmtl::setRot().

```

367 {
368     return Vec<DATA_TYPE,3>( ((v1[Yelt]*v2[Zelt]) - (v1[Zelt]*v2[Yelt])),
369                             ((v1[Zelt]*v2[Xelt]) - (v1[Xelt]*v2[Zelt])),
370                             ((v1[Xelt]*v2[Yelt]) - (v1[Yelt]*v2[Xelt])) );
371 }
```

#### 8.4.2.7 `template<class DATA_TYPE> DATA_TYPE distance (const Plane< DATA_TYPE > & plane, const Point< DATA_TYPE, 3 > & pt)`

Computes the distance from the plane to the point.

##### Parameters:

*plane* the plane to compare the point to it

*pt* a point in space

##### Returns:

the distance from the point to the plane

Definition at line 59 of file PlaneOps.h.

References `gmatl::dot()`.

```
60 {
61     return ( dot(plane.mNorm, static_cast< Vec<DATA_TYPE, 3> >(pt)) - plane.mOffset );
62 }
```

#### 8.4.2.8 `template<typename DATA_TYPE> Quat<DATA_TYPE>& div (Quat< DATA_TYPE > & result, const Quat< DATA_TYPE > & q, DATA_TYPE s)`

quaternion vector scale.

##### Postcondition:

$\text{result} = q / s$

##### See also:

[Quat](#)

Definition at line 213 of file QuatOps.h.

```
214     {
215         result[0] = q[0] / s;
216         result[1] = q[1] / s;
217         result[2] = q[2] / s;
218         result[3] = q[3] / s;
219         return result;
220     }
```

**8.4.2.9** `template<typename DATA_TYPE> Quat<DATA_TYPE>& div  
 (Quat< DATA_TYPE > & result, const Quat< DATA_TYPE > & q1,  
 const Quat< DATA_TYPE > & q2)`

quotient of two quaternions.

**Postcondition:**

$\text{result} = q1 / q2$

**See also:**

[Quat](#)

Definition at line 190 of file QuatOps.h.

References `gmtl::conj()`, `gmtl::mult()`, and `gmtl::Welt`.

Referenced by `gmtl::operator/()`, and `gmtl::operator/=(())`.

```

191     {
192         Quat<DATA_TYPE> q2_inv( q2 ), r, s;
193
194         // conj the quat
195         conj( q2_inv );
196
197         mult( r, q1, q2_inv );
198         mult( s, q2_inv, q2_inv );
199
200         float sw_inv = 1.0f / s[Welt];
201         result[0] = r[0] * sw_inv;
202         result[1] = r[1] * sw_inv;
203         result[2] = r[2] * sw_inv;
204         result[3] = r[3] * sw_inv;
205         return result;
206     }

```

**8.4.2.10** `template<typename DATA_TYPE> DATA_TYPE dot (const Quat<  
 DATA_TYPE > & q1, const Quat< DATA_TYPE > & q2)`

vector dot product between two quaternions.

get the lengthSquared between two quat vectors...

**Postcondition:**

$N(q) = x1*x2 + y1*y2 + z1*z2 + w1*w2$   
 $\text{result} = x1*x2 + y1*y2 + z1*z2 + w1*w2$

**See also:**

[Quat](#)

Definition at line 318 of file QuatOps.h.

Referenced by gmtl::distance(), gmtl::findNearestPt(), gmtl::lengthSquared(), gmtl::lerp(), gmtl::setDirCos(), gmtl::setRot(), gmtl::slerp(), gmtl::TestIntersect(), and gmtl::TestIntersectOBB().

```

319     {
320         return DATA_TYPE( (q1[0] * q2[0]) +
321                             (q1[1] * q2[1]) +
322                             (q1[2] * q2[2]) +
323                             (q1[3] * q2[3]) );
324     }

```

**8.4.2.11** `template<class DATA_TYPE, unsigned SIZE> DATA_TYPE dot  
(const Vec< DATA_TYPE, SIZE > & v1, const Vec< DATA_TYPE,  
SIZE > & v2)`

Computes dot product of v1 and v2 and returns the result.

**Parameters:**

**v1** the first vector  
**v2** the second vector

**Returns:**

the dotproduct of v1 and v2

Definition at line 263 of file VecOps.h.

Referenced by gmtl::Plane< DATA\_TYPE >::Plane().

```

264 {
265     DATA_TYPE ret_val(0);
266     for(unsigned i=0;i<SIZE;++i)
267     {
268         ret_val += (v1[i] * v2[i]);
269     }
270     return ret_val;
271 }

```

**8.4.2.12** `template<typename DATA_TYPE> Quat<DATA_TYPE>& exp  
(Quat< DATA_TYPE > & result)`

complex exponentiation.

**Precondition:**

safe to pass self as argument

**Postcondition:**

sets self to the exponentiation of quat

**See also:**

[Quat](#)

Definition at line 434 of file QuatOps.h.

References `gml::Math::cos()`, `gml::Math::sin()`, `gml::Math::sqrt()`, `gml::Welt`, `gml::Xelt`, `gml::Yelt`, and `gml::Zelt`.

```

435  {
436      DATA_TYPE len1, len2;
437
438      len1 = Math::sqrt( result[Xelt] * result[Xelt] +
439                          result[Yelt] * result[Yelt] +
440                          result[Zelt] * result[Zelt] );
441      if (len1 > (DATA_TYPE)0.0)
442          len2 = Math::sin( len1 ) / len1;
443      else
444          len2 = (DATA_TYPE)1.0;
445
446      result[Xelt] = result[Xelt] * len2;
447      result[Yelt] = result[Yelt] * len2;
448      result[Zelt] = result[Zelt] * len2;
449      result[Welt] = Math::cos( len1 );
450
451      return result;
452  }
```

**8.4.2.13** `template<class DATA_TYPE> DATA_TYPE findNearestPt (const Plane< DATA_TYPE > &plane, const Point< DATA_TYPE, 3 > &pt, Point< DATA_TYPE, 3 > &result)`

Finds the point on the plane that is nearest to the given point.

As a convenience, the distance between pt and result is returned.

**Parameters:**

*plane* [in] the plane to compare the point to

*pt* [in] the point to test

*result* [out] the point on plane closest to pt

**Returns:**

the distance between pt and result

Definition at line 125 of file PlaneOps.h.

References `gmtl::dot()`, `gmtlASSERT`, and `gmtl::isNormalized()`.

```

128 {
129     // GGI: p297
130     // GGII: p223
131     gmtlASSERT( isNormalized(plane.mNorm) );    // Assert: Normalized
132     DATA_TYPE dist_to_plane(0);
133     dist_to_plane = plane.mOffset + dot( plane.mNorm, static_cast< Vec<DATA_TYPE, 3> >(pt) );
134     result = pt - (plane.mNorm * dist_to_plane);
135     return dist_to_plane;
136 }
```

#### 8.4.2.14 `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& identity (Matrix<DATA_TYPE, ROWS, COLS > &result) [inline]`

Make identity matrix out the matrix.

make sure every elt is 0.

Definition at line 55 of file MatrixOps.h.

References `gmtl::Math::Min()`.

Referenced by `gmtl::set()`.

```

56     {
57         if(result.mState != Matrix<DATA_TYPE, ROWS, COLS>::IDENTITY)    // if not already ident
58         {
59             // TODO: mp
60             for (unsigned int r = 0; r < ROWS; ++r)
61                 for (unsigned int c = 0; c < COLS; ++c)
62                     result( r, c ) = (DATA_TYPE)0.0;
63
64             // TODO: mp
65             for (unsigned int x = 0; x < Math::Min( COLS, ROWS ); ++x)
66                 result( x, x ) = (DATA_TYPE)1.0;
67
68             result.mState = Matrix<DATA_TYPE, ROWS, COLS>::IDENTITY;
69             result.mState = Matrix<DATA_TYPE, ROWS, COLS>::FULL;
70         }
71
72         return result;
73     }
```

#### 8.4.2.15 `template<typename DATA_TYPE> Quat<DATA_TYPE>& invert (Quat< DATA_TYPE > & result)`

quaternion multiplicative inverse.

##### Postcondition:

self becomes the multiplicative inverse of self

$1/q = q^* / N(q)$

##### See also:

[Quat](#)

Definition at line 410 of file QuatOps.h.

References `gmtl::conj()`, `gmtl::lengthSquared()`, `gmtl::Welt`, `gmtl::Xelt`, `gmtl::Yelt`, and `gmtl::Zelt`.

```

411  {
412      // do result = conj( q ) / norm( q )
413      conj( result );
414
415      // return if norm() is near 0 (divide by 0 would result in NaN)
416      DATA_TYPE l = lengthSquared( result );
417      if ( l < (DATA_TYPE)0.0001 )
418          return result;
419
420      DATA_TYPE l_inv = ((DATA_TYPE)1.0) / l;
421      result[Xelt] *= l_inv;
422      result[Yelt] *= l_inv;
423      result[Zelt] *= l_inv;
424      result[Welt] *= l_inv;
425      return result;
426  }
```

#### 8.4.2.16 `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& invert (Matrix< DATA_TYPE, ROWS, COLS > & result) [inline]`

smart matrix inversion (in place) Does matrix inversion by intelligently selecting what type of inversion to use depending on the types of operations your [Matrix](#) has been through.

5 types of inversion: FULL, AFFINE, ORTHONORMAL, ORTHOGONAL, IDENTITY.

Check for error with [Matrix::isError\(\)](#). @POST: result' = inv( result ) @POST: If inversion failed, then error bit is set within the [Matrix](#).

Definition at line 426 of file MatrixOps.h.

References gmtl::invert().

```

427     {
428         return invert( result, result );
429     }

```

**8.4.2.17** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& invert (Matrix<DATA_TYPE, ROWS, COLS> & result, const Matrix<DATA_TYPE, ROWS, COLS> & src) [inline]`

smart matrix inversion.

Does matrix inversion by intelligently selecting what type of inversion to use depending on the types of operations your [Matrix](#) has been through.

5 types of inversion: FULL, AFFINE, ORTHONORMAL, ORTHOGONAL, IDENTITY.

Check for error with [Matrix::isError\(\)](#). @POST: result' = inv( result ) @POST: If inversion failed, then error bit is set within the [Matrix](#).

Definition at line 407 of file MatrixOps.h.

References gmtl::invertFull().

Referenced by gmtl::invert(), gmtl::makeInverse(), and gmtl::makeInvert().

```

408     {
409         if (src.mState == Matrix<DATA_TYPE, ROWS, COLS>::IDENTITY )
410             return result = src;
411         else
412             return invertFull( result, src );
413     }

```

**8.4.2.18** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& invertFull (Matrix<DATA_TYPE, ROWS, COLS> & result, const Matrix<DATA_TYPE, ROWS, COLS> & src) [inline]`

full matrix inversion.

Check for error with [Matrix::isError\(\)](#). @POST: result' = inv( result ) @POST: If inversion failed, then error bit is set within the [Matrix](#).



Definition at line 288 of file MatrixOps.h.

References gmtl::Math::abs().

Referenced by gmtl::invert().

```

289 {
290     /*-----*
291     | mat_inv: Compute the inverse of a n x n matrix, using the maximum pivot |
292     | strategy. n <= MAX1. |
293     *-----*
294
295     Parameters:
296         a      a n x n square matrix
297         b      inverse of input a.
298         n      dimension of matrix a.
299     */
300
301     const DATA_TYPE* a = src.getData();
302     DATA_TYPE* b = result.mData;
303
304     int n = 4;
305     int i, j, k;
306     int r[ 4], c[ 4], row[ 4], col[ 4];
307     DATA_TYPE m[ 4][ 4*2], pivot, max_m, tmp_m, fac;
308
309     /* Initialization */
310     for ( i = 0; i < n; i ++ )
311     {
312         r[ i] = c[ i] = 0;
313         row[ i] = col[ i] = 0;
314     }
315
316     /* Set working matrix */
317     for ( i = 0; i < n; i++ )
318     {
319         for ( j = 0; j < n; j++ )
320         {
321             m[ i][ j] = a[ i * n + j];
322             m[ i][ j + n] = ( i == j ) ? (DATA_TYPE)1.0 : (DATA_TYPE)0.0 ;
323         }
324     }
325
326     /* Begin of loop */
327     for ( k = 0; k < n; k++ )
328     {
329         /* Choosing the pivot */
330         for ( i = 0, max_m = 0; i < n; i++ )
331         {
332             if ( row[ i] )
333                 continue;
334             for ( j = 0; j < n; j++ )
335             {
336                 if ( col[ j] )
337                     continue;

```

```

338         tmp_m = gmtl::Math::abs( m[ i][ j]);
339         if ( tmp_m > max_m)
340         {
341             max_m = tmp_m;
342             r[ k] = i;
343             c[ k] = j;
344         }
345     }
346 }
347 row[ r[k] ] = col[ c[k] ] = 1;
348 pivot = m[ r[ k] ][ c[ k] ];
349
350
351 if ( gmtl::Math::abs( pivot) <= 1e-20)
352 {
353     std::cerr << "**** pivot = %f in mat_inv. ****\n";
354     result.setError();
355     return result;
356 }
357
358 /* Normalization */
359 for ( j = 0; j < 2*n; j++ )
360 {
361     if ( j == c[ k] )
362         m[ r[ k] ][ j] = (DATA_TYPE)1.0;
363     else
364         m[ r[ k] ][ j] /= pivot;
365 }
366
367 /* Reduction */
368 for ( i = 0; i < n; i++ )
369 {
370     if ( i == r[ k] )
371         continue;
372
373     for ( j=0, fac = m[ i][ c[k]]; j < 2*n; j++ )
374     {
375         if ( j == c[ k] )
376             m[ i][ j] = (DATA_TYPE)0.0;
377         else
378             m[ i][ j] -= fac * m[ r[k] ][ j];
379     }
380 }
381 }
382
383 /* Assign inverse to a matrix */
384 for ( i = 0; i < n; i++ )
385     for ( j = 0; j < n; j++ )
386         row[ i] = ( c[ j] == i ) ? r[ j] : row[ i];
387
388 for ( i = 0; i < n; i++ )
389     for ( j = 0; j < n; j++ )
390         b[ i * n + j] = m[ row[ i] ][ j + n];
391
392 // It worked

```

```

393     return result;
394 }

```

**8.4.2.19** `template<class DATA_TYPE, unsigned SIZE> bool isNormalized  
(const Vec< DATA_TYPE, SIZE > & v1, const DATA_TYPE eps =  
(DATA_TYPE)0.0001)`

Determines if the given vector is normalized within the given tolerance.

The vector is normalized if its lengthSquared is 1.

**Parameters:**

*v1* the vector to test

*eps* the epsilon tolerance

**Returns:**

true if the vector is normalized, false otherwise

Definition at line 348 of file VecOps.h.

References gmtl::isEqual(), and gmtl::lengthSquared().

```

350 {
351     return Math::isEqual( lengthSquared( v1 ), (DATA_TYPE)1.0, eps );
352 }

```

**8.4.2.20** `template<typename DATA_TYPE> bool isNormalized (const Quat<  
DATA_TYPE > & q1, const DATA_TYPE eps = (DATA_TYPE)0.0001f)`

Determines if the given vector is normalized within the given tolerance.

The vector is normalized if its lengthSquared is 1.

**Parameters:**

*v1* the vector to test

*eps* the epsilon tolerance

**Returns:**

true if the vector is normalized, false otherwise

Definition at line 384 of file QuatOps.h.

References gmtl::isEqual(), and gmtl::lengthSquared().

Referenced by gmtl::findNearestPt().

```

385     {
386         return Math::isEqual( lengthSquared( q1 ), DATA_TYPE(1), eps );
387     }

```

#### 8.4.2.21 `template<class DATA_TYPE, unsigned SIZE> DATA_TYPE length (const Vec< DATA_TYPE, SIZE > & v1)`

Computes the length of the given vector.

##### Parameters:

`v1` the vector with which to compute the length

##### Returns:

the length of `v1`

Definition at line 281 of file VecOps.h.

References gmtl::lengthSquared(), and gmtl::Math::sqrt().

Referenced by gmtl::LineSeg< DATA\_TYPE >::getLength().

```

282 {
283     DATA_TYPE ret_val = lengthSquared(v1);
284     if (ret_val == 0.0f)
285         return 0.0f;
286     else
287         return Math::sqrt(ret_val);
288 }

```

#### 8.4.2.22 `template<typename DATA_TYPE> DATA_TYPE length (const Quat< DATA_TYPE > & q)`

quaternion "absolute" (also known as vector length or magnitude) using this can be faster than using length for some operations...

##### Postcondition:

returns the magnitude of the 4D vector.

result = sqrt( lengthSquared( q ) )

See also:

[Quat](#)

Definition at line 346 of file QuatOps.h.

References `gmtl::lengthSquared()`, and `gmtl::Math::sqrt()`.

Referenced by `gmtl::isInVolume()`, `gmtl::isOnVolume()`, `gmtl::log()`, `gmtl::normalize()`, and `gmtl::xform()`.

```

347 {
348     return Math::sqrt( lengthSquared( q ) );
349 }
```

#### 8.4.2.23 `template<class DATA_TYPE, unsigned SIZE> DATA_TYPE lengthSquared (const Vec< DATA_TYPE, SIZE > & v1)`

Computes the square of the length of the given vector.

This can be used in many calculations instead of `length` to increase speed by saving you an expensive `sqrt` call.

**Parameters:**

***v1*** the vector with which to compute the squared length

**Returns:**

the square of the length of *v1*

Definition at line 300 of file VecOps.h.

```

301 {
302     DATA_TYPE ret_val(0);
303     for(unsigned i=0;i<SIZE;++i)
304     {
305         ret_val += (v1[i] * v1[i]);
306     }
307     return ret_val;
308 }
309 }
```

#### 8.4.2.24 `template<typename DATA_TYPE> DATA_TYPE lengthSquared (const Quat< DATA_TYPE > & q)`

quaternion "norm" (also known as vector length squared) using this can be faster than using `length` for some operations...

**Postcondition:**

returns the vector length squared

$$N(q) = x^2 + y^2 + z^2 + w^2$$

result = x\*x + y\*y + z\*z + w\*w

**See also:**

[Quat](#)

Definition at line 334 of file QuatOps.h.

References gmtl::dot().

Referenced by gmtl::invert(), gmtl::isNormalized(), gmtl::length(), gmtl::makeVolume(), gmtl::set(), and gmtl::setRot().

```

335     {
336         return dot( q, q );
337     }
```

#### 8.4.2.25 `template<typename DATA_TYPE> Quat<DATA_TYPE> & log(Quat< DATA_TYPE > & result)`

complex logarithm.

**Postcondition:**

sets self to the log of quat

**See also:**

[Quat](#)

Definition at line 459 of file QuatOps.h.

References gmtl::isEqual(), gmtl::length(), gmtl::Math::sqrt(), gmtl::Welt, gmtl::Xelt, gmtl::Yelt, and gmtl::Zelt.

```

460     {
461         DATA_TYPE length;
462
463         length = Math::sqrt( result[Xelt] * result[Xelt] +
464                             result[Yelt] * result[Yelt] +
465                             result[Zelt] * result[Zelt] );
466
467         // avoid divide by 0
468         if (Math::isEqual( result[Welt], (DATA_TYPE)0.0, (DATA_TYPE)0.00001 ) == false)
469             length = Math::atan( length / result[Welt] );
470         else
```

```

471         length = Math::PI_OVER_2;
472
473         result[Welt] = (DATA_TYPE)0.0;
474         result[Xelt] = result[Xelt] * length;
475         result[Yelt] = result[Yelt] * length;
476         result[Zelt] = result[Zelt] * length;
477         return result;
478     }

```

**8.4.2.26** `template<typename DATA_TYPE> void meanTangent (Quat< DATA_TYPE > & result, const Quat< DATA_TYPE > & q1, const Quat< DATA_TYPE > & q2, const Quat< DATA_TYPE > & q3)`

WARNING: not implemented (do not use).

Definition at line 489 of file QuatOps.h.

References gmtlASSERT.

```

490     {
491         gmtlASSERT( false );
492     }

```

**8.4.2.27** `template<typename DATA_TYPE> Quat<DATA_TYPE>& mult (Quat< DATA_TYPE > & result, const Quat< DATA_TYPE > & q, DATA_TYPE s)`

vector scalar multiplication.

**Postcondition:**

`result' = [qx*s, qy*s, qz*s, qw*s]`

**See also:**

[Quat](#)

Definition at line 155 of file QuatOps.h.

```

156     {
157         result[0] = q[0] * s;
158         result[1] = q[1] * s;
159         result[2] = q[2] * s;
160         result[3] = q[3] * s;
161         return result;
162     }

```

**8.4.2.28** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& mult (Matrix<DATA_TYPE, ROWS, COLS > & result, DATA_TYPE scalar) [inline]`

matrix scalar mult.

mult each elt in a matrix by a scalar value. @POST: result \*= scalar

Definition at line 227 of file MatrixOps.h.

```
228    {
229        for (unsigned i = 0; i < ROWS * COLS; ++i)
230            result[i] *= scalar;
231        return result;
232    }
```

**8.4.2.29** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& mult (Matrix<DATA_TYPE, ROWS, COLS > & result, const Matrix< DATA_TYPE, ROWS, COLS > & mat, float scalar) [inline]`

matrix scalar mult.

mult each elt in a matrix by a scalar value. @POST: result = mat \* scalar

Definition at line 215 of file MatrixOps.h.

```
216    {
217        for (unsigned i = 0; i < ROWS * COLS; ++i)
218            result[i] = mat[i] * scalar;
219        return result;
220    }
```

**8.4.2.30** `template<typename DATA_TYPE, unsigned ROWS, unsigned INTERNAL, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& mult (Matrix< DATA_TYPE, ROWS, COLS > & result, const Matrix< DATA_TYPE, ROWS, INTERNAL > & lhs, const Matrix< DATA_TYPE, INTERNAL, COLS > & rhs) [inline]`

matrix multiply.



@PRE: if lhs is  $m \times p$ , and rhs is  $p \times n$ , then result is  $m \times n$  (mult func undefined otherwise) @POST: returns a  $m \times n$  matrix

Definition at line 105 of file MatrixOps.h.

References gmtl::zero().

Referenced by gmtl::div(), gmtl::operator \*(), gmtl::operator \*=( ), gmtl::postMult(), and gmtl::preMult().

```

108  {
109      Matrix<DATA_TYPE, ROWS, COLS> ret_mat; // prevent aliasing
110      zero( ret_mat );
111
112      // p. 150 Numerical Analysis (second ed.)
113      // if A is m x p, and B is p x n, then AB is m x n
114      // (AB)ij = [k = 1 to p] (a)ik (b)kj      (where: 1 <= i <= m, 1 <= j <= n)
115      for (unsigned int i = 0; i < ROWS; ++i)          // 1 <= i <= m
116      for (unsigned int j = 0; j < COLS; ++j)          // 1 <= j <= n
117      for (unsigned int k = 0; k < INTERNAL; ++k)      // [k = 1 to p]
118          ret_mat( i, j ) += lhs( i, k ) * rhs( k, j );
119
120      return result = ret_mat;
121  }
```

**8.4.2.31** `template<typename DATA_TYPE> Quat<DATA_TYPE>& mult`  
`(Quat< DATA_TYPE > & result, const Quat< DATA_TYPE > & q1,`  
`const Quat< DATA_TYPE > & q2)`

product of two quaternions (quaternion product) multiplication of quats is much like multiplication of typical complex numbers.

**Postcondition:**

$q1q2 = (s1 + v1)(s2 + v2)$   
 $this' = q1 * q2$  (grassman product)

**See also:**

[Quat](#)

Definition at line 55 of file QuatOps.h.

References gmtl::Welt, gmtl::Xelt, gmtl::Yelt, and gmtl::Zelt.

```

56  {
57      // Here is the easy to understand equation: (grassman product)
58      // scalar_component = q1.s * q2.s - dot(q1.v, q2.v)
59      // vector_component = q2.v * q1.s + q1.v * q2.s + cross(q1.v, q2.v)
```

```

60
61      // Here is another version (euclidean product, just FYI)...
62      // scalar_component = q1.s * q2.s + dot(q1.v, q2.v)
63      // vector_component = q2.v * q1.s - q1.v * q2.s - cross(q1.v, q2.v)
64
65      // Here it is, using vector algebra (grassman product)
66      /*
67      const float& w1( q1[Welt] ), w2( q2[Welt] );
68      Vec3 v1( q1[Xelt], q1[Yelt], q1[Zelt] ), v2( q2[Xelt], q2[Yelt], q2[Zelt] );
69
70      float w = w1 * w2 - v1.dot( v2 );
71      Vec3 v = (w1 * v2) + (w2 * v1) + v1.cross( v2 );
72
73      vec[Welt] = w;
74      vec[Xelt] = v[0];
75      vec[Yelt] = v[1];
76      vec[Zelt] = v[2];
77      */
78
79      // Here is the same, only expanded... (grassman product)
80      Quat<DATA_TYPE> temporary; // avoid aliasing problems...
81      temporary[Xelt] = q1[Welt]*q2[Xelt] + q1[Xelt]*q2[Welt] + q1[Yelt]*q2[Zelt] - q1[Zelt]*q2[Yelt];
82      temporary[Yelt] = q1[Welt]*q2[Yelt] + q1[Yelt]*q2[Welt] + q1[Zelt]*q2[Xelt] - q1[Xelt]*q2[Zelt];
83      temporary[Zelt] = q1[Welt]*q2[Zelt] + q1[Zelt]*q2[Welt] + q1[Xelt]*q2[Yelt] - q1[Yelt]*q2[Xelt];
84      temporary[Welt] = q1[Welt]*q2[Welt] - q1[Xelt]*q2[Xelt] - q1[Yelt]*q2[Yelt] - q1[Zelt]*q2[Zelt];
85
86      // use a temporary, in case q1 or q2 is the same as self.
87      result[Xelt] = temporary[Xelt];
88      result[Yelt] = temporary[Yelt];
89      result[Zelt] = temporary[Zelt];
90      result[Welt] = temporary[Welt];
91
92      // don't normalize, because it might not be rotation arithmetic we're doing
93      // (only rotation quats have unit length)
94      return result;
95  }

```

#### 8.4.2.32 `template<typename DATA_TYPE> Quat<DATA_TYPE>& negate` (`Quat< DATA_TYPE > &result`)

Vector negation - negate each element in the quaternion vector.

the negative of a rotation quaternion is geometrically equivalent to the original. there exist 2 quats for every possible rotation.

##### Postcondition:

returns the negation of the given quat.

Definition at line 130 of file QuatOps.h.

```

131  {
132      result[0] = -result[0];
133      result[1] = -result[1];
134      result[2] = -result[2];
135      result[3] = -result[3];
136      return result;
137  }

```

#### 8.4.2.33 `template<class DATA_TYPE> Vec<DATA_TYPE, 3> normal (const Tri< DATA_TYPE > & tri)`

Computes the normal for this triangle.

##### Parameters:

*tri* the triangle for which to compute the normal

##### Returns:

the normal vector for tri

Definition at line 69 of file TriOps.h.

References gmtl::cross(), gmtl::normal(), and gmtl::normalize().

Referenced by gmtl::normal().

```

70 {
71     Vec<DATA_TYPE, 3> normal = cross( tri[1] - tri[0], tri[2] - tri[0] );
72     normalize( normal );
73     return normal;
74 }

```

#### 8.4.2.34 `template<class DATA_TYPE, unsigned SIZE> DATA_TYPE normalize (Vec< DATA_TYPE, SIZE > & v1)`

Normalizes the given vector in place causing it to be of unit length.

If the vector is already of length 1.0, nothing is done. For convenience, the original length of the vector is returned.

##### Postcondition:

`length(v1) == 1.0`

**Parameters:**

**v1** the vector to normalize

**Returns:**

the length of v1 before it was normalized

Definition at line 323 of file VecOps.h.

References gmtl::length().

Referenced by gmtl::Plane< DATA\_TYPE >::Plane().

```

324 {
325     DATA_TYPE len = length(v1);
326
327     if(len != 0.0f)
328     {
329         for(unsigned i=0;i<SIZE;++i)
330         {
331             v1[i] /= len;
332         }
333     }
334
335     return len;
336 }
```

#### 8.4.2.35 `template<typename DATA_TYPE> Quat<DATA_TYPE>& normalize(Quat< DATA_TYPE > &result)`

set self to the normalized quaternion of self.

**Precondition:**

magnitude should be > 0, otherwise no calculation is done.

**Postcondition:**

result' = normalize( result ), where normalize makes length( result ) == 1

**See also:**

[Quat](#)

Definition at line 357 of file QuatOps.h.

References gmtl::length(), gmtl::Welt, gmtl::Xelt, gmtl::Yelt, and gmtl::Zelt.

Referenced by gmtl::extendVolume(), gmtl::makeNormal(), gmtl::makeXRot(), gmtl::makeYRot(), gmtl::makeZRot(), gmtl::normal(), gmtl::set(), and gmtl::setRot().

```

358  {
359      DATA_TYPE l = length( result );
360
361      // return if no magnitude (already as normalized as possible)
362      if (l < (DATA_TYPE)0.0001)
363          return result;
364
365      DATA_TYPE l_inv = ((DATA_TYPE)1.0) / l;
366      result[Xelt] *= l_inv;
367      result[Yelt] *= l_inv;
368      result[Zelt] *= l_inv;
369      result[Welt] *= l_inv;
370
371      return result;
372  }

```

**8.4.2.36** `template<class DATA_TYPE, unsigned SIZE, class SCALAR_TYPE>`  
`VecBase<DATA_TYPE, SIZE> operator * (const SCALAR_TYPE &`  
`scalar, const VecBase< DATA_TYPE, SIZE > & v1)`

Multiplies v1 by a scalar value and returns the result.

Thus  $\text{result} = \text{scalar} * \text{v1}$ . This is equivalent to  $\text{result} = \text{v1} * \text{scalar}$ .

**Parameters:**

*scalar* the amount by which to scale v1

*v1* the vector to scale

**Returns:**

the result of multiplying v1 by scalar

Definition at line 197 of file VecOps.h.

```

199  {
200      VecBase<DATA_TYPE, SIZE> ret_val(v1);
201      ret_val *= scalar;
202      return ret_val;
203
204      //return VecBase<DATA_TYPE, SIZE>(v1) *= scalar;
205  }

```

**8.4.2.37** `template<class DATA_TYPE, unsigned SIZE, class SCALAR_TYPE>  
VecBase<DATA_TYPE, SIZE> operator * (const VecBase<  
 DATA_TYPE, SIZE > & v1, const SCALAR_TYPE & scalar)`

Multiplies v1 by a scalar value and returns the result.

Thus result = v1 \* scalar.

**Parameters:**

*v1* the vector to scale

*scalar* the amount by which to scale v1

**Returns:**

the result of multiplying v1 by scalar

Definition at line 177 of file VecOps.h.

```
179 {
180     VecBase<DATA_TYPE, SIZE> ret_val(v1);
181     ret_val *= scalar;
182     return ret_val;
183
184     //return VecBase<DATA_TYPE, SIZE>(v1) *= scalar;
185 }
```

**8.4.2.38** `template<typename DATA_TYPE> Quat<DATA_TYPE> operator *  
 (const Quat< DATA_TYPE > & q, DATA_TYPE s)`

vector scalar multiplication.

**Postcondition:**

result' = [qx\*s, qy\*s, qz\*s, qw\*s]

**See also:**

[Quat](#)

Definition at line 169 of file QuatOps.h.

References gmtl::mult().

```
170     {
171         Quat<DATA_TYPE> temporary;
172         return mult( temporary, q, s );
173     }
```

**8.4.2.39** `template<typename DATA_TYPE> Quat<DATA_TYPE> operator *  
(const Quat< DATA_TYPE > & q1, const Quat< DATA_TYPE > &  
q2)`

product of two quaternions (quaternion product).

**Postcondition:**

this' = q1 \* q2 (grassman product)

**See also:**

**Quat**

**Todo:**

metaprogramming on quat **operator \***()

Definition at line 103 of file QuatOps.h.

References gmtl::Welt, gmtl::Xelt, gmtl::Yelt, and gmtl::Zelt.

```

104     {
105         // (grassman product - see mult() for discussion)
106         // don't normalize, because it might not be rotation arithmetic we're doing
107         // (only rotation quats have unit length)
108         return Quat<DATA_TYPE>( q1[Welt]*q2[Xelt] + q1[Xelt]*q2[Welt] + q1[Yelt]*q2[Zelt] - q1[Zelt]*q2[Yelt]
109                                q1[Welt]*q2[Yelt] + q1[Yelt]*q2[Welt] + q1[Zelt]*q2[Xelt] - q1[Xelt]*q2[Zelt]
110                                q1[Welt]*q2[Zelt] + q1[Zelt]*q2[Welt] + q1[Xelt]*q2[Yelt] - q1[Yelt]*q2[Xelt]
111                                q1[Welt]*q2[Welt] - q1[Xelt]*q2[Xelt] - q1[Yelt]*q2[Yelt] - q1[Zelt]*q2[Zelt]
112     }
```

**8.4.2.40** `template<typename DATA_TYPE, unsigned ROWS, unsigned  
INTERNAL, unsigned COLS> Matrix<DATA_TYPE, ROWS,  
COLS> operator * (const Matrix< DATA_TYPE, ROWS, INTERNAL  
> & lhs, const Matrix< DATA_TYPE, INTERNAL, COLS > & rhs)  
[inline]`

matrix \* matrix.

@PRE: if lhs is m x p, and rhs is p x n, then result is m x n (mult func undefined otherwise) @POST: returns a m x n matrix == lhs \* rhs returns a temporary, is slower.

Definition at line 129 of file MatrixOps.h.

References gmtl::mult().

```

131     {
132         Matrix<DATA_TYPE, ROWS, COLS> temporary;
133         return mult( temporary, lhs, rhs );
134     }
```

**8.4.2.41** `template<class DATA_TYPE, unsigned SIZE, class SCALAR_TYPE>  
VecBase<DATA_TYPE, SIZE>& operator *=(VecBase<  
 DATA_TYPE, SIZE > & v1, const SCALAR_TYPE & scalar)`

Multiplies *v1* by a scalar value and stores the result in *v1*.

This is equivalent to the expression `v1 = v1 * scalar`.

**Parameters:**

*v1* the vector to scale

*scalar* the amount by which to scale *v1*

**Returns:**

*v1* after it has been multiplied by *scalar*

Definition at line 156 of file `VecOps.h`.

```
158 {
159     for(unsigned i=0;i<SIZE;++i)
160     {
161         v1[i] *= scalar;
162     }
163
164     return v1;
165 }
```

**8.4.2.42** `template<typename DATA_TYPE> Quat<DATA_TYPE>& operator  
 *=(Quat< DATA_TYPE > & q, DATA_TYPE s)`

vector scalar multiplication.

**Postcondition:**

`result' = [resultx*s, resulty*s, resultz*s, resultw*s]`

**See also:**

[Quat](#)

Definition at line 180 of file `QuatOps.h`.

References `gmtl::mult()`.



```

181  {
182      return mult( q, q, s );
183  }

```

**8.4.2.43** `template<typename DATA_TYPE> Quat<DATA_TYPE>& operator  
*= (Quat< DATA_TYPE > & result, const Quat< DATA_TYPE > &  
q2)`

quaternion postmult.

**Postcondition:**

result' = result \* q2

**See also:**

[Quat](#)

Definition at line 119 of file QuatOps.h.

References gmtl::mult().

```

120  {
121      return mult( result, result, q2 );
122  }

```

**8.4.2.44** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS>  
Matrix<DATA_TYPE, ROWS, COLS>& operator *= (Matrix<  
DATA_TYPE, ROWS, COLS > & result, DATA_TYPE scalar)  
[inline]`

matrix scalar mult (operator \*=").

multiply matrix elements by a scalar @POST: result \*= scalar

Definition at line 239 of file MatrixOps.h.

References gmtl::mult().

```

240  {
241      return mult( result, scalar );
242  }

```

**8.4.2.45** `template<typename DATA_TYPE, unsigned SIZE>`  
`Matrix<DATA_TYPE, SIZE, SIZE> & operator *= (Matrix<`  
`DATA_TYPE, SIZE, SIZE > & result, const Matrix< DATA_TYPE,`  
`SIZE, SIZE > & operand) [inline]`

matrix postmult (operator \*=").

does a postmult on the matrix. @PRE: args must both be n x n (this function is undefined otherwise) @POST: result' = result \* operand

Definition at line 204 of file MatrixOps.h.

References gmtl::postMult().

```
206     {
207         return postMult( result, operand );
208     }
```

**8.4.2.46** `template<class DATA_TYPE, unsigned SIZE>`  
`VecBase<DATA_TYPE, SIZE> operator+ (const VecBase<`  
`DATA_TYPE, SIZE > & v1, const VecBase< DATA_TYPE, SIZE > &`  
`v2)`

Adds v2 to v1 and returns the result.

Thus result = v1 + v2.

**Parameters:**

**v1** the first vector

**v2** the second vector

**Returns:**

the result of adding v2 to v1

Definition at line 100 of file VecOps.h.

```
102 {
103     VecBase<DATA_TYPE, SIZE> ret_val(v1);
104     ret_val += v2;
105     return ret_val;
106 }
```

**8.4.2.47** `template<typename DATA_TYPE> Quat<DATA_TYPE> operator+  
(const Quat< DATA_TYPE > & q1, const Quat< DATA_TYPE > &  
q2)`

vector addition.

**Postcondition:**

`result' = [qx+s, qy+s, qz+s, qw+s]`

**See also:**

[Quat](#)

Definition at line 261 of file QuatOps.h.

References `gmtl::add()`.

```
262 {
263     Quat<DATA_TYPE> temporary;
264     return add( temporary, q1, q2 );
265 }
```

**8.4.2.48** `template<class DATA_TYPE, unsigned SIZE>  
VecBase<DATA_TYPE, SIZE>& operator+=( VecBase<  
DATA_TYPE, SIZE > & v1, const VecBase< DATA_TYPE, SIZE > &  
v2)`

Adds *v2* to *v1* and stores the result in *v1*.

This is equivalent to the expression `v1 = v1 + v2`.

**Parameters:**

*v1* the first vector

*v2* the second vector

**Returns:**

*v1* after *v2* has been added to it

Definition at line 80 of file VecOps.h.

```
82 {
83     for(unsigned i=0;i<SIZE;++i)
84     {
85         v1[i] += v2[i];
86     }
87
88     return v1;
89 }
```

**8.4.2.49** `template<typename DATA_TYPE> Quat<DATA_TYPE>&  
operator+= (Quat< DATA_TYPE > & q1, const Quat< DATA_TYPE  
> & q2)`

vector addition.

**Postcondition:**

`result' = [resultx+s, resulty+s, resultz+s, resultw+s]`

**See also:**

[Quat](#)

Definition at line 272 of file QuatOps.h.

References `gmtl::add()`.

```
273     {
274         return add( q1, q1, q2 );
275     }
```

**8.4.2.50** `template<class DATA_TYPE, unsigned SIZE> Vec<DATA_TYPE,  
SIZE> operator- (const VecBase< DATA_TYPE, SIZE > & v1, const  
VecBase< DATA_TYPE, SIZE > & v2)`

Subtracts *v2* from *v1* and returns the result.

Thus `result = v1 - v2`.

**Parameters:**

*v1* the first vector

*v2* the second vector

**Returns:**

the result of subtracting *v2* from *v1*

Definition at line 138 of file VecOps.h.

```
140 {
141     Vec<DATA_TYPE, SIZE> ret_val(v1);
142     ret_val -= v2;
143     return ret_val;
144 }
```

**8.4.2.51** `template<typename DATA_TYPE> Quat<DATA_TYPE> operator-  
(const Quat< DATA_TYPE > & q1, const Quat< DATA_TYPE > &  
q2)`

vector subtraction.

**Postcondition:**

result' = [qx-s, qy-s, qz-s, qw-s]

**See also:**

[Quat](#)

Definition at line 295 of file QuatOps.h.

References `gmtl::sub()`.

```
296     {
297         Quat<DATA_TYPE> temporary;
298         return sub( temporary, q1, q2 );
299     }
```

**8.4.2.52** `template<typename DATA_TYPE> Quat<DATA_TYPE> operator-  
(const Quat< DATA_TYPE > & quat)`

Vector negation - (operator-) return a temporary that is the negative of the given quat.

the negative of a rotation quaternion is geometrically equivalent to the original. there exist 2 quats for every possible rotation.

**Postcondition:**

returns the negation of the given quat

Definition at line 145 of file QuatOps.h.

```
146     {
147         return Quat<DATA_TYPE>( -quat[0], -quat[1], -quat[2], -quat[3] );
148     }
```

**8.4.2.53** `template<typename DATA_TYPE, unsigned SIZE>  
Vec<DATA_TYPE, SIZE> operator- (const VecBase< DATA_TYPE,  
SIZE > & v1)`

Negates v1.

The result = -v1.

**Parameters:**

**v1** the vector.

**Returns:**

the result of negating v1.

Definition at line 60 of file VecOps.h.

```

61 {
62     Vec<DATA_TYPE, SIZE> ret_val;
63     for ( unsigned i=0; i < SIZE; ++i )
64     {
65         ret_val[i] = -v1[i];
66     }
67     return ret_val;
68 }
```

**8.4.2.54** `template<class DATA_TYPE, unsigned SIZE>`  
`VecBase<DATA_TYPE, SIZE>& operator-= (VecBase< DATA_TYPE,`  
`SIZE > & v1, const VecBase< DATA_TYPE, SIZE > & v2)`

Subtracts v2 from v1 and stores the result in v1.

This is equivalent to the expression `v1 = v1 - v2`.

**Parameters:**

**v1** the first vector

**v2** the second vector

**Returns:**

v1 after v2 has been subtracted from it

Definition at line 118 of file VecOps.h.

```

120 {
121     for(unsigned i=0;i<SIZE;++i)
122     {
123         v1[i] -= v2[i];
124     }
125
126     return v1;
127 }
```

**8.4.2.55** `template<typename DATA_TYPE> Quat<DATA_TYPE>&  
operator- ( Quat< DATA_TYPE > & q1, const Quat< DATA_TYPE >  
& q2)`

vector subtraction.

**Postcondition:**

`result' = [resultx-s, resulty-s, resultz-s, resultw-s]`

**See also:**

[Quat](#)

Definition at line 306 of file QuatOps.h.

References `gmtl::sub()`.

```
307     {
308         return sub( q1, q1, q2 );
309     }
```

**8.4.2.56** `template<class DATA_TYPE, unsigned SIZE, class SCALAR_TYPE>  
VecBase<DATA_TYPE, SIZE> operator/ (const VecBase<  
DATA_TYPE, SIZE > & v1, const SCALAR_TYPE & scalar)`

Divides *v1* by a scalar value and returns the result.

Thus `result = v1 / scalar`.

**Parameters:**

*v1* the vector to scale

*scalar* the amount by which to scale *v1*

**Returns:**

the result of dividing *v1* by *scalar*

Definition at line 238 of file VecOps.h.

```
240 {
241     VecBase<DATA_TYPE, SIZE> ret_val(v1);
242     ret_val /= scalar;
243     return ret_val;
244     // return VecBase<DATA_TYPE, SIZE>(v1)( /= scalar;
245 }
```

#### 8.4.2.57 `template<typename DATA_TYPE> Quat<DATA_TYPE> operator/ (const Quat< DATA_TYPE > & q, DATA_TYPE s)`

vector scalar division.

##### Postcondition:

result' = [qx/s, qy/s, qz/s, qw/s]

##### See also:

[Quat](#)

Definition at line 227 of file QuatOps.h.

References `gmtl::div()`.

```
228     {
229         Quat<DATA_TYPE> temporary;
230         return div( temporary, q, s );
231     }
```

#### 8.4.2.58 `template<class DATA_TYPE, unsigned SIZE, class SCALAR_TYPE> VecBase<DATA_TYPE, SIZE>& operator/= ( VecBase< DATA_TYPE, SIZE > & v1, const SCALAR_TYPE & scalar)`

Divides *v1* by a scalar value and stores the result in *v1*.

This is equivalent to the expression `v1 = v1 / scalar`.

##### Parameters:

*v1* the vector to scale

*scalar* the amount by which to scale *v1*

##### Returns:

*v1* after it has been divided by *scalar*

Definition at line 217 of file VecOps.h.

```
219 {
220     for(unsigned i=0;i<SIZE;++i)
221     {
222         v1[i] /= scalar;
223     }
224
225     return v1;
226 }
```



**8.4.2.59** `template<typename DATA_TYPE> Quat<DATA_TYPE>&  
operator/=(const Quat< DATA_TYPE > & q, DATA_TYPE s)`

vector scalar division.

**Postcondition:**

`result' = [resultx/s, resulty/s, resultz/s, resultw/s]`

**See also:**

[Quat](#)

Definition at line 238 of file QuatOps.h.

References `gmtl::div()`.

```
239  {
240      return div( q, q, s );
241  }
```

**8.4.2.60** `template<typename DATA_TYPE, unsigned SIZE>  
Matrix<DATA_TYPE, SIZE, SIZE>& postMult (Matrix<  
DATA_TYPE, SIZE, SIZE > & result, const Matrix< DATA_TYPE,  
SIZE, SIZE > & operand) [inline]`

matrix postmultiply.

@PRE: args must both be n x n (this function is undefined otherwise) @POST: result'  
= result \* operand

Definition at line 181 of file MatrixOps.h.

References `gmtl::mult()`.

Referenced by `gmtl::operator *=(())`.

```
183  {
184      return mult( result, result, operand );
185  }
```

**8.4.2.61** `template<typename DATA_TYPE, unsigned SIZE>  
Matrix<DATA_TYPE, SIZE, SIZE>& preMult (Matrix<  
DATA_TYPE, SIZE, SIZE > & result, const Matrix< DATA_TYPE,  
SIZE, SIZE > & operand) [inline]`

matrix preMultiply.

@PRE: args must both be n x n (this function is undefined otherwise) @POST: result' = operand \* result

Definition at line 192 of file MatrixOps.h.

References gmtl::mult().

```

194     {
195         return mult( result, operand, result );
196     }
```

**8.4.2.62** `template<typename DATA_TYPE> void squad (Quat< DATA_TYPE > & result, DATA_TYPE t, const Quat< DATA_TYPE > & q1, const Quat< DATA_TYPE > & q2, const Quat< DATA_TYPE > & a, const Quat< DATA_TYPE > & b)`

WARNING: not implemented (do not use).

Definition at line 482 of file QuatOps.h.

References gmtlASSERT.

```

483     {
484         gmtlASSERT( false );
485     }
```

**8.4.2.63** `template<typename DATA_TYPE> Quat<DATA_TYPE>& sub (Quat< DATA_TYPE > & result, const Quat< DATA_TYPE > & q1, const Quat< DATA_TYPE > & q2)`

vector subtraction.

**See also:**

[Quat](#)

Definition at line 281 of file QuatOps.h.

```

282     {
283         result[0] = q1[0] - q2[0];
284         result[1] = q1[1] - q2[1];
285         result[2] = q1[2] - q2[2];
286         result[3] = q1[3] - q2[3];
287         return result;
288     }
```

**8.4.2.64** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS>  
 Matrix<DATA_TYPE, ROWS, COLS>& sub (Matrix< DATA_TYPE,  
 ROWS, COLS > & result, const Matrix< DATA_TYPE, ROWS, COLS  
 > & lhs, const Matrix< DATA_TYPE, ROWS, COLS > & rhs)  
 [inline]`

matrix subtraction (algebraic operation for matrix).

@PRE: if lhs is  $m \times n$ , and rhs is  $m \times n$ , then result is  $m \times n$  (mult func undefined otherwise) @POST: returns a  $m \times n$  matrix TODO: **enforce the sizes with templates...**

Definition at line 142 of file MatrixOps.h.

Referenced by `gmtl::operator-()`, and `gmtl::operator-=()`.

```

145  {
146      // p. 150 Numerical Analysis (second ed.)
147      // if A is  $m \times n$ , and B is  $m \times n$ , then AB is  $m \times n$ 
148      // (A - B)ij = (a)ij - (b)ij (where:  $1 \leq i \leq m, 1 \leq j \leq n$ )
149      for (unsigned int i = 0; i < ROWS; ++i) //  $1 \leq i \leq m$ 
150          for (unsigned int j = 0; j < COLS; ++j) //  $1 \leq j \leq n$ 
151              result( i, j ) = lhs( i, j ) - rhs( i, j );
152
153      return result;
154  }
```

**8.4.2.65** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS>  
 Matrix<DATA_TYPE, ROWS, COLS>& transpose (Matrix<  
 DATA_TYPE, ROWS, COLS > & result, const Matrix< DATA_TYPE,  
 COLS, ROWS > & source)`

matrix transpose from one type to another (i.e.

$3 \times 4$  to  $4 \times 3$ ) @PRE: source needs to be an  $M \times N$  matrix, while dest needs to be  $N \times M$

@POST: flip along diagonal

Definition at line 264 of file MatrixOps.h.

```

265  {
266      // in case result is == source... :(
267      Matrix<DATA_TYPE, COLS, ROWS> temp = source;
268
269      // p. 149 Numerical Analysis (second ed.)
270      for (unsigned i = 0; i < ROWS; ++i)
271      {
272          for (unsigned j = 0; j < COLS; ++j)
273          {
274              result( i, j ) = temp( j, i );
275          }
276      }
```

```

276     }
277
278     return result;
279 }

```

**8.4.2.66** `template<typename DATA_TYPE, unsigned SIZE>`  
`Matrix<DATA_TYPE, SIZE, SIZE>& transpose (Matrix<`  
`DATA_TYPE, SIZE, SIZE > & result)`

matrix transpose in place.

@PRE: needs to be an N x N matrix @POST: flip along diagonal

Definition at line 249 of file MatrixOps.h.

Referenced by gmtl::makeTranspose().

```

250 {
251     // p. 27 game programming gems #1
252     for (unsigned c = 0; c < SIZE; ++c)
253         for (unsigned r = c + 1; r < SIZE; ++r)
254             std::swap( result( r, c ), result( c, r ) );
255
256     return result;
257 }

```

**8.4.2.67** `template<class DATA_TYPE> PlaneSide` whichSide (const `Plane<`  
`DATA_TYPE > & plane`, const `Point< DATA_TYPE, 3 > & pt`, const  
`DATA_TYPE & eps`)

Determines which side of the plane the given point lies with the given epsilon tolerance.

**Parameters:**

*plane* the plane to compare the point to

*pt* the point to test

*eps* the epsilon tolerance to use while testing

**Returns:**

the PlaneSide enum describing on which side of the plane the point lies

Definition at line 100 of file PlaneOps.h.

References gmtl::distance(), gmtl::NEG\_SIDE, gmtl::ON\_PLANE, gmtl::PlaneSide, and gmtl::POS\_SIDE.

```

103 {
104     DATA_TYPE dist = distance( plane, pt );
105
106     if ( dist < eps )
107         return NEG_SIDE;
108     else if ( dist > eps )
109         return POS_SIDE;
110     else
111         return ON_PLANE;
112 }

```

#### 8.4.2.68 `template<class DATA_TYPE> PlaneSide whichSide (const Plane<DATA_TYPE> &plane, const Point<DATA_TYPE, 3> &pt)`

Determines which side of the plane the given point lies.

This operation is done with ZERO tolerance.

##### Parameters:

*plane* the plane to compare the point to

*pt* the point to test

##### Returns:

the PlaneSide enum describing on which side of the plane the point lies

Definition at line 75 of file PlaneOps.h.

References `gmtl::distance()`, `gmtl::NEG_SIDE`, `gmtl::ON_PLANE`, `gmtl::PlaneSide`, and `gmtl::POS_SIDE`.

```

77 {
78     DATA_TYPE dist = distance( plane, pt );
79
80     if ( dist < DATA_TYPE(0) )
81         return NEG_SIDE;
82     else if ( dist > DATA_TYPE(0) )
83         return POS_SIDE;
84     else
85         return ON_PLANE;
86 }

```

**8.4.2.69** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS>`  
`Matrix<DATA_TYPE, ROWS, COLS>& zero (Matrix< DATA_TYPE,`  
`ROWS, COLS > &result) [inline]`

zero out the matrix.

make sure every elt is 0.

Definition at line 80 of file MatrixOps.h.

References `gmtl::Math::Min()`.

Referenced by `gmtl::mult()`.

```

81  {
82      if (result.mState == Matrix<DATA_TYPE, ROWS, COLS>::IDENTITY)
83      {
84          for (unsigned int x = 0; x < Math::Min( ROWS, COLS ); ++x)
85          {
86              result( x, x ) = (DATA_TYPE)0;
87          }
88      }
89      else
90      {
91          for (unsigned int x = 0; x < ROWS*COLS; ++x)
92          {
93              result[x] = (DATA_TYPE)0;
94          }
95      }
96      return result;
97  }
```

## 8.5 Spatial Transformers: xform( ... ), operator( ... ).

Transform points and vectors by Matrices and Quaternions.

### Vector Transform (Quaternion)

- `template<typename DATA_TYPE> VecBase< DATA_TYPE, 3 > & xform (VecBase< DATA_TYPE, 3 > &result, const Quat< DATA_TYPE > &rot, const VecBase< DATA_TYPE, 3 > &vector)`

*transform a vector by a rotation quaternion.*

- `template<typename DATA_TYPE> VecBase< DATA_TYPE, 3 > operator * (const Quat< DATA_TYPE > &rot, const VecBase< DATA_TYPE, 3 > &vector)`

*transform a vector by a rotation quaternion.*

### Vector Transform (Matrix)

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Vec< DATA_TYPE, COLS > & xform (Vec< DATA_TYPE, COLS > &result, const Matrix< DATA_TYPE, ROWS, COLS > &matrix, const Vec< DATA_TYPE, COLS > &vector)`

*xform a vector by a matrix.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Vec< DATA_TYPE, COLS > operator * (const Matrix< DATA_TYPE, ROWS, COLS > &matrix, const Vec< DATA_TYPE, COLS > &vector)`

*matrix \* vector xform.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, unsigned VEC_SIZE> Vec< DATA_TYPE, VEC_SIZE > & xform (Vec< DATA_TYPE, VEC_SIZE > &result, const Matrix< DATA_TYPE, ROWS, COLS > &matrix, const Vec< DATA_TYPE, VEC_SIZE > &vector)`

*partially transform a partially specified vector by a matrix, assumes last elt of vector is 0 (the 0 makes it only partially transformed).*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, unsigned COLS_MINUS_ONE> Vec< DATA_TYPE, COLS_MINUS_ONE > operator *`

(const Matrix< DATA\_TYPE, ROWS, COLS > &matrix, const Vec< DATA\_TYPE, COLS\_MINUS\_ONE > &vector)

*matrix \* partial vector, assumes last elt of vector is 0 (partial transform).*

## Point Transform (Matrix)

- template<typename DATA\_TYPE, unsigned ROWS, unsigned COLS> Point< DATA\_TYPE, COLS > & [xform](#) (Point< DATA\_TYPE, COLS > &result, const Matrix< DATA\_TYPE, ROWS, COLS > &matrix, const Point< DATA\_TYPE, COLS > &point)

*transform point by a matrix.*

- template<typename DATA\_TYPE, unsigned ROWS, unsigned COLS> Point< DATA\_TYPE, COLS > [operator \\*](#) (const Matrix< DATA\_TYPE, ROWS, COLS > &matrix, const Point< DATA\_TYPE, COLS > &point)

*matrix \* point.*

- template<typename DATA\_TYPE, unsigned ROWS, unsigned COLS, unsigned PNT\_SIZE> Point< DATA\_TYPE, PNT\_SIZE > & [xform](#) (Point< DATA\_TYPE, PNT\_SIZE > &result, const Matrix< DATA\_TYPE, ROWS, COLS > &matrix, const Point< DATA\_TYPE, PNT\_SIZE > &point)

*transform a partially specified point by a matrix, assumes last elt of point is 1.*

- template<typename DATA\_TYPE, unsigned ROWS, unsigned COLS, unsigned COLS\_MINUS\_ONE> Point< DATA\_TYPE, COLS\_MINUS\_ONE > [operator \\*](#) (const Matrix< DATA\_TYPE, ROWS, COLS > &matrix, const Point< DATA\_TYPE, COLS\_MINUS\_ONE > &point)

*matrix \* partially specified point.*

### 8.5.1 Detailed Description

Transform points and vectors by Matrices and Quaternions.

Note that xform is defined differently for Point and Vec. By Point is a full xform, by Vec is only a rotation.

### 8.5.2 Function Documentation



**8.5.2.1** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, unsigned COLS_MINUS_ONE> Point<DATA_TYPE, COLS_MINUS_ONE> operator * (const Matrix< DATA_TYPE, ROWS, COLS > & matrix, const Point< DATA_TYPE, COLS_MINUS_ONE > & point) [inline]`

matrix \* partially specified point.

multiplication of [m x k] matrix by a [k-1 x 1] matrix (also known as a [Point](#) [with w == 1 for points by definition] ).

**Postcondition:**

the [k-1 x 1] vector you pass in is treated as a [point, 1.0]

This results in a full matrix xform of the point.

Definition at line 286 of file Xforms.h.

References gmtl::xform().

```
287 {
288     Point<DATA_TYPE, COLS_MINUS_ONE> temporary;
289     return xform( temporary, matrix, point );
290 }
```

**8.5.2.2** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Point<DATA_TYPE, COLS> operator * (const Matrix< DATA_TYPE, ROWS, COLS > & matrix, const Point< DATA_TYPE, COLS > & point) [inline]`

matrix \* point.

multiplication of [m x k] matrix by a [k x 1] matrix (also known as a [Point](#)...).

**Postcondition:**

This results in a full matrix xform of the point.

returns a point same size as the matrix rows... (p[r][1] = m[r][k] \* p[k][1])

Definition at line 233 of file Xforms.h.

References gmtl::xform().

```
234 {
235     Point<DATA_TYPE, COLS> temporary;
236     return xform( temporary, matrix, point );
237 }
```

**8.5.2.3** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, unsigned COLS_MINUS_ONE> Vec<DATA_TYPE, COLS_MINUS_ONE> operator * (const Matrix< DATA_TYPE, ROWS, COLS > & matrix, const Vec< DATA_TYPE, COLS_MINUS_ONE > & vector) [inline]`

matrix \* partial vector, assumes last elt of vector is 0 (partial transform).

multiplication of [m x k] matrix by a [k-1 x 1] matrix (also known as a Vector [with w == 0 for vectors by definition] ).

**Postcondition:**

the [k-1 x 1] vector you pass in is treated as a [vector, 0.0]

This ends up being a partial xform using only the rotation from the matrix (vector xformed result is untranslated).

Definition at line 193 of file Xforms.h.

References gmtl::xform().

```

194     {
195         Vec<DATA_TYPE, COLS_MINUS_ONE> temporary;
196         return xform( temporary, matrix, vector );
197     }
```

**8.5.2.4** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Vec<DATA_TYPE, COLS> operator * (const Matrix< DATA_TYPE, ROWS, COLS > & matrix, const Vec< DATA_TYPE, COLS > & vector) [inline]`

matrix \* vector xform.

multiplication of [m x k] matrix by a [k x 1] matrix (also known as a Vector...).

**Postcondition:**

This results in a full matrix xform of the vector (assumes you know what you are doing - i.e. that you know that the last component of a vector by definition is 0.0, and changing this might make the xform different that what you may expect).

returns a vec same size as the matrix rows... (v[r][1] = m[r][k] \* v[k][1])

Definition at line 139 of file Xforms.h.

References gmtl::xform().

```

140     {
```

```

141      // do a standard [m x k] by [k x n] matrix multiplication (where n == 0).
142      Vec<DATA_TYPE, COLS> temporary;
143      return xform( temporary, matrix, vector );
144  }

```

**8.5.2.5** `template<typename DATA_TYPE> VecBase<DATA_TYPE, 3>`  
`operator * (const Quat< DATA_TYPE > & rot, const VecBase<`  
`DATA_TYPE, 3 > & vector) [inline]`

transform a vector by a rotation quaternion.

**Precondition:**

give a vector, and a rotation quaternion (by definition, a rotation quaternion is normalized).

**Postcondition:**

$v' = q P(v) q^*$  (where result is  $v'$ , rot is  $q$ , and vector is  $v$ .  $q^*$  is  $\text{conj}(q)$ , and  $P(v)$  is pure quaternion made from  $v$ )

Definition at line 95 of file Xforms.h.

References gmtl::xform().

```

96  {
97      VecBase<DATA_TYPE, 3> temporary;
98      return xform( temporary, rot, vector );
99  }

```

**8.5.2.6** `template<typename DATA_TYPE, unsigned ROWS, unsigned`  
`COLS, unsigned PNT_SIZE> Point<DATA_TYPE, PNT_SIZE>&`  
`xform (Point< DATA_TYPE, PNT_SIZE > & result, const Matrix<`  
`DATA_TYPE, ROWS, COLS > & matrix, const Point< DATA_TYPE,`  
`PNT_SIZE > & point) [inline]`

transform a partially specified point by a matrix, assumes last elt of point is 1.

Transforms a point with a matrix, uses multiplication of [m x k] matrix by a [k-1 x 1] matrix (also known as a `Point` [with  $w == 1$  for points by definition] ).

**Postcondition:**

the [k-1 x 1] point you pass in is treated as [point, 1.0]  
This results in a full matrix xform of the point.

**Todo:**

we need a PointOps.h operator \*=(scalar) function

Definition at line 249 of file Xforms.h.

References gmtlASSERT, and gmtl::isEqual().

```

250     {
251         gmtlASSERT( PNT_SIZE == COLS - 1 && "The precondition of this method is that the vect
252
253         // copy the point to the correct size.
254         Point<DATA_TYPE, PNT_SIZE+1> temp_point, temp_result;
255         for (unsigned x = 0; x < PNT_SIZE; ++x)
256             temp_point[x] = point[x];
257         temp_point[PNT_SIZE] = (DATA_TYPE)1.0; // by definition of a point, set the last uns
258
259         // transform it.
260         xform<DATA_TYPE, ROWS, COLS>( temp_result, matrix, temp_point );
261
262         // convert result back to pnt<DATA_TYPE, PNT_SIZE>
263         // some matrices will make the W param large even if this is a true vector,
264         // we'll need to redistribute it to the other elts if W param is non-zero
265         if (Math::isEqual( temp_result[PNT_SIZE], (DATA_TYPE)0, (DATA_TYPE)0.0001 ) == false
266         {
267             DATA_TYPE w_coord_div = DATA_TYPE( 1.0 ) / temp_result[PNT_SIZE];
268             for (unsigned x = 0; x < PNT_SIZE; ++x)
269                 result[x] = temp_result[x] * w_coord_div;
270         }
271         else
272         {
273             for (unsigned x = 0; x < PNT_SIZE; ++x)
274                 result[x] = temp_result[x];
275         }
276
277         return result;
278     }

```

**8.5.2.7** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, unsigned VEC_SIZE> Vec<DATA_TYPE, VEC_SIZE>& xform (Vec<DATA_TYPE, VEC_SIZE> & result, const Matrix< DATA_TYPE, ROWS, COLS> & matrix, const Vec< DATA_TYPE, VEC_SIZE> & vector)` [inline]

partially transform a partially specified vector by a matrix, assumes last elt of vector is 0 (the 0 makes it only partially transformed).

Transforms a vector with a matrix, uses multiplication of [m x k] matrix by a [k-1 x 1] matrix (also known as a Vector [with w == 0 for vectors by definition] ).

**Postcondition:**

the  $[k-1 \times 1]$  vector you pass in is treated as a  $[\text{vector}, 0.0]$

This ends up being a partial xform using only the rotation from the matrix (vector xformed result is untranslated).

Definition at line 155 of file Xforms.h.

References gmtlASSERT, and gmtl::isEqual().

```

156  {
157      gmtlASSERT( VEC_SIZE == COLS - 1 );
158      // do a standard [m x k] by [k x n] matrix multiplication (where n == 0).
159
160      // copy the point to the correct size.
161      Vec<DATA_TYPE, COLS> temp_vector, temp_result;
162      for (unsigned x = 0; x < VEC_SIZE; ++x)
163          temp_vector[x] = vector[x];
164      temp_vector[COLS-1] = (DATA_TYPE)0.0; // by definition of a vector, set the last unspecified elt
165
166      // transform it.
167      xform<DATA_TYPE, ROWS, COLS>( temp_result, matrix, temp_vector );
168
169      // convert result back to vec<DATA_TYPE, VEC_SIZE>
170      // some matrices will make the W param large even if this is a true vector,
171      // we'll need to redistribute it to the other elts if W param is non-zero
172      if (Math::isEqual( temp_result[VEC_SIZE], (DATA_TYPE)0, (DATA_TYPE)0.0001 ) == false)
173      {
174          DATA_TYPE w_coord_div = DATA_TYPE( 1.0 ) / temp_result[VEC_SIZE];
175          for (unsigned x = 0; x < VEC_SIZE; ++x)
176              result[x] = temp_result[x] * w_coord_div;
177      }
178      else
179      {
180          for (unsigned x = 0; x < VEC_SIZE; ++x)
181              result[x] = temp_result[x];
182      }
183
184      return result;
185  }

```

**8.5.2.8** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS>`  
`Point<DATA_TYPE, COLS>& xform (Point< DATA_TYPE, COLS >`  
`& result, const Matrix< DATA_TYPE, ROWS, COLS > & matrix, const`  
`Point< DATA_TYPE, COLS > & point) [inline]`

transform point by a matrix.

multiplication of  $[m \times k]$  matrix by a  $[k \times 1]$  matrix (also known as a [Point...](#)).

**Postcondition:**

This results in a full matrix xform of the point.

returns a point same size as the matrix rows... ( $p[r][1] = m[r][k] * p[k][1]$ )

Definition at line 213 of file Xforms.h.

```

214     {
215         // do a standard [m x k] by [k x n] matrix multiplication (n == 1).
216
217         // reset point to zero...
218         result = Point<DATA_TYPE, COLS>();
219
220         for (unsigned iRow = 0; iRow < ROWS; ++iRow)
221             for (unsigned iCol = 0; iCol < COLS; ++iCol)
222                 result[iRow] += matrix( iRow, iCol ) * point[iCol];
223
224         return result;
225     }
```

**8.5.2.9** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS>  
Vec<DATA_TYPE, COLS>& xform (Vec< DATA_TYPE, COLS > &  
result, const Matrix< DATA_TYPE, ROWS, COLS > & matrix, const  
Vec< DATA_TYPE, COLS > & vector) [inline]`

xform a vector by a matrix.

Transforms a vector with a matrix, uses multiplication of [m x k] matrix by a [k x 1] matrix (the later also known as a Vector...).

**Postcondition:**

This results in a full matrix xform of the vector (assumes you know what you are doing - i.e. that you know that the last component of a vector by definition is 0.0, and changing this might make the xform different than what you may expect).

returns a point same size as the matrix rows... ( $v[r][1] = m[r][k] * v[k][1]$ )

Definition at line 116 of file Xforms.h.

```

117     {
118         // do a standard [m x k] by [k x n] matrix multiplication (where n == 0).
119
120         // reset vec to zero...
121         result = Vec<DATA_TYPE, COLS>();
122
123         for (unsigned iRow = 0; iRow < ROWS; ++iRow)
124             for (unsigned iCol = 0; iCol < COLS; ++iCol)
125                 result[iRow] += matrix( iRow, iCol ) * vector[iCol];
126     }
```

```

127     return result;
128 }

```

**8.5.2.10** `template<typename DATA_TYPE> VecBase<DATA_TYPE, 3>& xform (VecBase< DATA_TYPE, 3 > & result, const Quat< DATA_TYPE > & rot, const VecBase< DATA_TYPE, 3 > & vector)`  
`[inline]`

transform a vector by a rotation quaternion.

**Precondition:**

give a vector, and a rotation quaternion (by definition, a rotation quaternion is normalized).

**Postcondition:**

$v' = q P(v) q^*$  (where result is  $v'$ , rot is  $q$ , and vector is  $v$ .  $q^*$  is  $\text{conj}(q)$ , and  $P(v)$  is pure quaternion made from  $v$ )

**See also:**

game programming gems #1 p199 , shoemake siggraph notes

**Note:**

for the implementation, `inv` and `conj` should both work for the " $q^*$ " in " $Rv = q P(v) q^*$ " but `conj` is actually faster so we usually choose that.  
also note, that if the input quat wasn't normalized (and thus isn't a rotation quat), then this might not give the correct result, since `conj` and `invert` is only equiv when normalized...

Definition at line 64 of file Xforms.h.

References `gmtl::ASSERT`, `gmtl::length()`, `gmtl::Welt`, `gmtl::Xelt`, `gmtl::Yelt`, and `gmtl::Zelt`.

Referenced by `gmtl::makeXRot()`, `gmtl::makeYRot()`, `gmtl::makeZRot()`, and `gmtl::operator*()`.

```

65     {
66         // check preconditions...
67         gmtl::ASSERT( Math::isEqual( length( rot ), (DATA_TYPE)1.0, (DATA_TYPE)0.0001 ) && "must pass a rotation quaternion" );
68
69         // easiest to write and understand (slowest too)
70         //return result_vec = makeVec( rot * makePure( vector ) * makeConj( rot ) );
71
72         // completely hand expanded
73         // (faster by 28% in gcc 2.96 debug mode.)

```

---

```

74      // (faster by 35% in gcc 2.96 opt3 mode (78% for doubles))
75      Quat<DATA_TYPE> rot_conj( -rot[Xelt], -rot[Yelt], -rot[Zelt], rot[Welt] );
76      Quat<DATA_TYPE> pure( vector[0], vector[1], vector[2], (DATA_TYPE)0.0 );
77      Quat<DATA_TYPE> temp(
78          pure[Welt]*rot_conj[Xelt] + pure[Xelt]*rot_conj[Welt] + pure[Yelt]*rot_conj[Zelt] -
79          pure[Welt]*rot_conj[Yelt] + pure[Yelt]*rot_conj[Welt] + pure[Zelt]*rot_conj[Xelt] -
80          pure[Welt]*rot_conj[Zelt] + pure[Zelt]*rot_conj[Welt] + pure[Xelt]*rot_conj[Yelt] -
81          pure[Welt]*rot_conj[Welt] - pure[Xelt]*rot_conj[Xelt] - pure[Yelt]*rot_conj[Yelt] -
82
83      result.set(
84          rot[Welt]*temp[Xelt] + rot[Xelt]*temp[Welt] + rot[Yelt]*temp[Zelt] - rot[Zelt]*temp
85          rot[Welt]*temp[Yelt] + rot[Yelt]*temp[Welt] + rot[Zelt]*temp[Xelt] - rot[Xelt]*temp
86          rot[Welt]*temp[Zelt] + rot[Zelt]*temp[Welt] + rot[Xelt]*temp[Yelt] - rot[Yelt]*temp
87      return result;
88  }
```



## 8.6 Comparison: isEqual(...), isEquiv(...), ==, !=

Tests for equality between GMTL data types.

### AxisAngle Comparitors

- `template<class DATA_TYPE> bool operator== (const AxisAngle< DATA_TYPE > &v1, const AxisAngle< DATA_TYPE > &v2)`  
*Compares v1 and v2 to see if they are exactly the same with zero tolerance.*
- `template<class DATA_TYPE> bool operator!= (const AxisAngle< DATA_TYPE > &v1, const AxisAngle< DATA_TYPE > &v2)`  
*Compares v1 and v2 to see if they are NOT exactly the same with zero tolerance.*
- `template<class DATA_TYPE> bool isEqual (const AxisAngle< DATA_TYPE > &v1, const AxisAngle< DATA_TYPE > &v2, const DATA_TYPE &eps=(DATA_TYPE) 0)`  
*Compares v1 and v2 to see if they are the same within the given epsilon tolerance.*

### Coord Comparitors

- `template<typename POS_TYPE, typename ROT_TYPE> bool operator== (const Coord< POS_TYPE, ROT_TYPE > &q1, const Coord< POS_TYPE, ROT_TYPE > &q2)`  
*Compare two quaternions for equality.*
- `template<typename POS_TYPE, typename ROT_TYPE> bool operator!= (const Coord< POS_TYPE, ROT_TYPE > &q1, const Coord< POS_TYPE, ROT_TYPE > &q2)`  
*Compare two quaternions for not-equality.*
- `template<typename POS_TYPE, typename ROT_TYPE> bool isEqual (const Coord< POS_TYPE, ROT_TYPE > &q1, const Coord< POS_TYPE, ROT_TYPE > &q2, typename Coord< POS_TYPE, ROT_TYPE >::DataType tol=(typename Coord< POS_TYPE, ROT_TYPE >::DataType) 0.0)`  
*Compare two quaternions for equality with tolerance.*

## EulerAngle Comparitors

- `template<class DATA_TYPE, typename ROT_ORDER> bool operator== (const EulerAngle< DATA_TYPE, ROT_ORDER > &v1, const EulerAngle< DATA_TYPE, ROT_ORDER > &v2)`

*Compares v1 and v2 to see if they are exactly the same with zero tolerance.*

- `template<class DATA_TYPE, typename ROT_ORDER> bool operator!= (const EulerAngle< DATA_TYPE, ROT_ORDER > &v1, const EulerAngle< DATA_TYPE, ROT_ORDER > &v2)`

*Compares v1 and v2 to see if they are NOT exactly the same with zero tolerance.*

- `template<class DATA_TYPE, typename ROT_ORDER> bool isEqual (const EulerAngle< DATA_TYPE, ROT_ORDER > &v1, const EulerAngle< DATA_TYPE, ROT_ORDER > &v2, const DATA_TYPE &eps=(DATA_TYPE) 0)`

*Compares v1 and v2 to see if they are the same within the given epsilon tolerance.*

## Matrix Comparitors

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> bool operator== (const Matrix< DATA_TYPE, ROWS, COLS > &lhs, const Matrix< DATA_TYPE, ROWS, COLS > &rhs)`

*Compare two mats.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> bool operator!= (const Matrix< DATA_TYPE, ROWS, COLS > &lhs, const Matrix< DATA_TYPE, ROWS, COLS > &rhs)`

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> bool isEqual (const Matrix< DATA_TYPE, ROWS, COLS > &lhs, const Matrix< DATA_TYPE, ROWS, COLS > &rhs, const DATA_TYPE &eps=(DATA_TYPE) 0)`

*Compare two vectors with a tolerance.*

## Plane Comparitors

- `template<class DATA_TYPE> bool operator== (const Plane< DATA_TYPE > &p1, const Plane< DATA_TYPE > &p2)`

*Compare two planes to see if they are EXACTLY the same.*

- `template<class DATA_TYPE> bool operator!= (const Plane< DATA_TYPE > &p1, const Plane< DATA_TYPE > &p2)`

*Compare two planes to see if they are not EXACTLY the same.*

- `template<class DATA_TYPE> bool isEqual (const Plane< DATA_TYPE > &p1, const Plane< DATA_TYPE > &p2, const DATA_TYPE &eps)`

*Compare two planes to see if they are the same within the given tolerance.*

## Quat Comparisons

- `template<typename DATA_TYPE> bool operator== (const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`

*Compare two quaternions for equality.*

- `template<typename DATA_TYPE> bool operator!= (const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`

*Compare two quaternions for not-equality.*

- `template<typename DATA_TYPE> bool isEqual (const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2, DATA_TYPE tol=0.0)`

*Compare two quaternions for equality with tolerance.*

- `template<typename DATA_TYPE> bool isEquiv (const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2, DATA_TYPE tol=0.0)`

*Compare two quaternions for geometric equivalence (with tolerance).*

## Sphere Comparitors

- `template<class DATA_TYPE> bool operator== (const Sphere< DATA_TYPE > &s1, const Sphere< DATA_TYPE > &s2)`

*Compare two spheres to see if they are EXACTLY the same.*

- `template<class DATA_TYPE> bool operator!= (const Sphere< DATA_TYPE > &s1, const Sphere< DATA_TYPE > &s2)`

*Compare two spheres to see if they are not EXACTLY the same.*

- `template<class DATA_TYPE> bool isEqual (const Sphere< DATA_TYPE > &s1, const Sphere< DATA_TYPE > &s2, const DATA_TYPE &eps)`

*Compare two spheres to see if they are the same within the given tolerance.*

## Triangle Comparitors

- `template<class DATA_TYPE> bool operator== (const Tri< DATA_TYPE > &tri1, const Tri< DATA_TYPE > &tri2)`

*Compare two triangles to see if they are EXACTLY the same.*

- `template<class DATA_TYPE> bool operator!= (const Tri< DATA_TYPE > &tri1, const Tri< DATA_TYPE > &tri2)`

*Compare two triangle to see if they are not EXACTLY the same.*

- `template<class DATA_TYPE> bool isEqual (const Tri< DATA_TYPE > &tri1, const Tri< DATA_TYPE > &tri2, const DATA_TYPE &eps)`

*Compare two triangles to see if they are the same within the given tolerance.*

## Vector Comparitors

- `template<class DATA_TYPE, unsigned SIZE> bool operator== (const VecBase< DATA_TYPE, SIZE > &v1, const VecBase< DATA_TYPE, SIZE > &v2)`

*Compares v1 and v2 to see if they are exactly the same with zero tolerance.*

- `template<class DATA_TYPE, unsigned SIZE> bool operator!= (const VecBase< DATA_TYPE, SIZE > &v1, const VecBase< DATA_TYPE, SIZE > &v2)`

*Compares v1 and v2 to see if they are NOT exactly the same with zero tolerance.*

- `template<class DATA_TYPE, unsigned SIZE> bool isEqual (const VecBase< DATA_TYPE, SIZE > &v1, const VecBase< DATA_TYPE, SIZE > &v2, const DATA_TYPE &eps)`

*Compares v1 and v2 to see if they are the same within the given epsilon tolerance.*

### 8.6.1 Detailed Description

Tests for equality between GMTL data types.

### 8.6.2 Function Documentation

**8.6.2.1** `template<class DATA_TYPE, unsigned SIZE> bool isEqual (const VecBase< DATA_TYPE, SIZE > & v1, const VecBase< DATA_TYPE, SIZE > & v2, const DATA_TYPE & eps) [inline]`

Compares v1 and v2 to see if they are the same within the given epsilon tolerance.

**Precondition:**

eps must be  $\geq 0$

**Parameters:**

*v1* the first vector

*v2* the second vector

*eps* the epsilon tolerance value

**Returns:**

true if v1 equals v2; false if they differ

Definition at line 498 of file VecOps.h.

References gmtlASSERT.

Referenced by gmtl::isEqual(), gmtl::isNormalized(), gmtl::log(), gmtl::setRot(), and gmtl::xform().

```

500 {
501     gmtlASSERT(eps >= 0);
502
503     for(unsigned i=0;i<SIZE;++i)
504     {
505         if (fabs(v1[i] - v2[i]) > eps)
506         {
507             return false;
508         }
509     }
510     return true;
511 }
```

**8.6.2.2** `template<class DATA_TYPE> bool isEqual (const Tri< DATA_TYPE > & tri1, const Tri< DATA_TYPE > & tri2, const DATA_TYPE & eps)`

Compare two triangles to see if they are the same within the given tolerance.

**Parameters:**

*tri1* the first triangle to compare

*tri2* the second triangle to compare  
*eps* the tolerance value to use

**Precondition:**

eps must be  $\geq 0$

**Returns:**

true if they are equal, false otherwise

Definition at line 126 of file TriOps.h.

References gmtlASSERT, and gmtl::isEqual().

```
128 {
129     gmtlASSERT( eps >= 0 );
130     return ( isEqual(tri1[0], tri2[0], eps) &&
131             isEqual(tri1[1], tri2[1], eps) &&
132             isEqual(tri1[2], tri2[2], eps) );
133 }
```

**8.6.2.3** `template<class DATA_TYPE> bool isEqual (const Sphere<  
 DATA_TYPE > & s1, const Sphere< DATA_TYPE > & s2, const  
 DATA_TYPE & eps) [inline]`

Compare two spheres to see if they are the same within the given tolerance.

**Parameters:**

*s1* the first sphere to compare  
*s2* the second sphere to compare  
*eps* the tolerance value to use

**Precondition:**

eps must be  $\geq 0$

**Returns:**

true if they are equal, false otherwise

Definition at line 91 of file SphereOps.h.

References gmtlASSERT, and gmtl::isEqual().

```
92 {
93     gmtlASSERT( eps >= 0 );
94     return ( (isEqual(s1.mCenter, s2.mCenter, eps)) &&
95             (Math::isEqual(s1.mRadius, s2.mRadius, eps)) );
96 }
```

**8.6.2.4** `template<typename DATA_TYPE> bool isEqual (const Quat< DATA_TYPE > & q1, const Quat< DATA_TYPE > & q2, DATA_TYPE tol = 0.0)`

Compare two quaternions for equality with tolerance.

Definition at line 636 of file QuatOps.h.

```

637 {
638     return bool( Math::isEqual( q1[0], q2[0], tol ) &&
639                 Math::isEqual( q1[1], q2[1], tol ) &&
640                 Math::isEqual( q1[2], q2[2], tol ) &&
641                 Math::isEqual( q1[3], q2[3], tol ) );
642 }
```

**8.6.2.5** `template<class DATA_TYPE> bool isEqual (const Plane< DATA_TYPE > & p1, const Plane< DATA_TYPE > & p2, const DATA_TYPE & eps) [inline]`

Compare two planes to see if they are the same within the given tolerance.

**Parameters:**

*p1* the first plane to compare  
*p2* the second plane to compare  
*eps* the tolerance value to use

**Precondition:**

eps must be  $\geq 0$

**Returns:**

true if they are equal, false otherwise

Definition at line 186 of file PlaneOps.h.

References gmtlASSERT, and gmtl::isEqual().

```

188 {
189     gmtlASSERT( eps >= 0 );
190     return ( (isEqual(p1.mNorm, p2.mNorm, eps)) &&
191             (Math::isEqual(p1.mOffset, p2.mOffset, eps)) );
192 }
```

**8.6.2.6** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> bool isEqual (const Matrix< DATA_TYPE, ROWS, COLS > & lhs, const Matrix< DATA_TYPE, ROWS, COLS > & rhs, const DATA_TYPE & eps = (DATA_TYPE)0) [inline]`

Compare two vectors with a tolerance.

**Precondition:**

eps must be  $\geq 0$

Definition at line 469 of file MatrixOps.h.

References `gmtlASSERT`, and `gmtl::isEqual()`.

```

470     {
471         gmtlASSERT( eps >= (DATA_TYPE)0 );
472
473         for (unsigned int i = 0; i < ROWS*COLS; ++i)
474         {
475             if (!Math::isEqual( lhs[i], rhs[i], eps ))
476                 return false;
477         }
478         return true;
479     }

```

**8.6.2.7** `template<class DATA_TYPE, typename ROT_ORDER> bool isEqual (const EulerAngle< DATA_TYPE, ROT_ORDER > & v1, const EulerAngle< DATA_TYPE, ROT_ORDER > & v2, const DATA_TYPE & eps = (DATA_TYPE)0) [inline]`

Compares v1 and v2 to see if they are the same within the given epsilon tolerance.

**Precondition:**

eps must be  $\geq 0$

**Parameters:**

*v1* the first rotation

*v2* the second rotation

*eps* the epsilon tolerance value

**Returns:**

true if v1 equals v2; false if they differ



Definition at line 62 of file EulerAngleOps.h.

References gmtlASSERT, and gmtl::isEqual().

```

65 {
66     gmtlASSERT(eps >= (DATA_TYPE)0);
67
68     // @todo metaprogramming.
69     if (!Math::isEqual( v1[0], v2[0], eps )) return false;
70     if (!Math::isEqual( v1[1], v2[1], eps )) return false;
71     if (!Math::isEqual( v1[2], v2[2], eps )) return false;
72     return true;
73 }
```

**8.6.2.8** `template<typename POS_TYPE, typename ROT_TYPE> bool isEqual`  
`(const Coord< POS_TYPE, ROT_TYPE > & q1, const Coord<`  
`POS_TYPE, ROT_TYPE > & q2, typename Coord< POS_TYPE,`  
`ROT_TYPE >::DataType tol = (typename Coord<POS_TYPE,`  
`ROT_TYPE>::DataType)0.0)`

Compare two quaternions for equality with tolerance.

Definition at line 71 of file CoordOps.h.

References gmtl::isEqual().

```

75 {
76     return bool( isEqual( q1.getPos(), q2.getPos(), tol ) &&
77                   isEqual( q1.getRot(), q2.getRot(), tol ) );
78 }
```

**8.6.2.9** `template<class DATA_TYPE> bool isEqual (const AxisAngle<`  
`DATA_TYPE > & v1, const AxisAngle< DATA_TYPE > & v2, const`  
`DATA_TYPE & eps = (DATA_TYPE)0) [inline]`

Compares v1 and v2 to see if they are the same within the given epsilon tolerance.

**Precondition:**

eps must be >= 0

**Parameters:**

v1 the first vector

v2 the second vector

*eps* the epsilon tolerance value

**Returns:**

true if v1 equals v2; false if they differ

Definition at line 63 of file AxisAngleOps.h.

References gmtlASSERT, and gmtl::isEqual().

Referenced by gmtl::isEqual(), and gmtl::isEquiv().

```

66 {
67     gmtlASSERT( eps >= (DATA_TYPE)0 );
68
69     // @todo metaprogramming.
70     if (!Math::isEqual( v1[0], v2[0], eps )) return false;
71     if (!Math::isEqual( v1[1], v2[1], eps )) return false;
72     if (!Math::isEqual( v1[2], v2[2], eps )) return false;
73     if (!Math::isEqual( v1[3], v2[3], eps )) return false;
74     return true;
75 }
```

**8.6.2.10** `template<typename DATA_TYPE> bool isEquiv (const Quat< DATA_TYPE > & q1, const Quat< DATA_TYPE > & q2, DATA_TYPE tol = 0.0)`

Compare two quaternions for geometric equivalence (with tolerance).

there exist 2 quats for every possible rotation: the original, and its negative. the negative of a rotation quaternion is geometrically equivalent to the original.

Definition at line 650 of file QuatOps.h.

References gmtl::isEqual().

```

651     {
652         return bool( isEqual( q1, q2, tol ) || isEqual( q1, -q2, tol ) );
653     }
```

**8.6.2.11** `template<class DATA_TYPE, unsigned SIZE> bool operator!= (const VecBase< DATA_TYPE, SIZE > & v1, const VecBase< DATA_TYPE, SIZE > & v2) [inline]`

Compares v1 and v2 to see if they are NOT exactly the same with zero tolerance.

**Parameters:***v1* the first vector*v2* the second vector**Returns:**true if *v1* does not equal *v2*; false if they are equal

Definition at line 479 of file VecOps.h.

```
481 {  
482     return(! (v1 == v2));  
483 }
```

**8.6.2.12** `template<class DATA_TYPE> bool operator!=(const Tri< DATA_TYPE > & tri1, const Tri< DATA_TYPE > & tri2)`

Compare two triangle to see if they are not EXACTLY the same.

In other words, this comparison is done with zero tolerance.

**Parameters:***tri1* the first triangle to compare*tri2* the second triangle to compare**Returns:**

true if they are not equal, false otherwise

Definition at line 109 of file TriOps.h.

```
110 {  
111     return (! (tri1 == tri2));  
112 }
```

**8.6.2.13** `template<class DATA_TYPE> bool operator!=(const Sphere< DATA_TYPE > & s1, const Sphere< DATA_TYPE > & s2)  
[inline]`

Compare two spheres to see if they are not EXACTLY the same.

In other words, this comparison is done with zero tolerance.

**Parameters:**

- s1* the first sphere to compare
- s2* the second sphere to compare

**Returns:**

true if they are not equal, false otherwise

Definition at line 74 of file SphereOps.h.

```

75 {
76     return (! (s1 == s2));
77 }
```

**8.6.2.14** `template<typename DATA_TYPE> bool operator!=(const Quat<DATA_TYPE> & q1, const Quat<DATA_TYPE> & q2) [inline]`

Compare two quaternions for not-equality.

**See also:**

`isEqual( Quat, Quat )`

Definition at line 628 of file QuatOps.h.

References `gmtl::operator==( )`.

```

629 {
630     return !operator==( q1, q2 );
631 }
```

**8.6.2.15** `template<class DATA_TYPE> bool operator!=(const Plane<DATA_TYPE> & p1, const Plane<DATA_TYPE> & p2) [inline]`

Compare two planes to see if they are not EXACTLY the same.

In other words, this comparison is done with zero tolerance.

**Parameters:**

- p1* the first plane to compare
- p2* the second plane to compare

**Returns:**

true if they are not equal, false otherwise

Definition at line 169 of file PlaneOps.h.

```
170 {
171     return (! (p1 == p2));
172 }
```

**8.6.2.16** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> bool operator!=(const Matrix< DATA_TYPE, ROWS, COLS > &lhs, const Matrix< DATA_TYPE, ROWS, COLS > &rhs) [inline]`

Definition at line 460 of file MatrixOps.h.

```
461 {
462     return bool( !(lhs == rhs) );
463 }
```

**8.6.2.17** `template<class DATA_TYPE, typename ROT_ORDER> bool operator!=(const EulerAngle< DATA_TYPE, ROT_ORDER > &v1, const EulerAngle< DATA_TYPE, ROT_ORDER > &v2) [inline]`

Compares v1 and v2 to see if they are NOT exactly the same with zero tolerance.

**Parameters:**

**v1** the first rotation

**v2** the second rotation

**Returns:**

true if v1 does not equal v2; false if they are equal

Definition at line 43 of file EulerAngleOps.h.

```
45 {
46     return(! (v1 == v2));
47 }
```

**8.6.2.18** `template<typename POS_TYPE, typename ROT_TYPE> bool  
operator!=(const Coord< POS_TYPE, ROT_TYPE > & q1, const  
Coord< POS_TYPE, ROT_TYPE > & q2) [inline]`

Compare two quaternions for not-equality.

**See also:**

`isEqual( Coord, Coord )`

Definition at line 62 of file CoordOps.h.

References `gmtl::operator==( )`.

```
64 {
65     return !operator==( q1, q2 );
66 }
```

**8.6.2.19** `template<class DATA_TYPE> bool operator!=(const AxisAngle<  
DATA_TYPE > & v1, const AxisAngle< DATA_TYPE > & v2)  
[inline]`

Compares v1 and v2 to see if they are NOT exactly the same with zero tolerance.

**Parameters:**

**v1** the first vector

**v2** the second vector

**Returns:**

true if v1 does not equal v2; false if they are equal

Definition at line 44 of file AxisAngleOps.h.

```
46 {
47     return !(v1 == v2);
48 }
```

**8.6.2.20** `template<class DATA_TYPE, unsigned SIZE> bool operator==(const  
VecBase< DATA_TYPE, SIZE > & v1, const VecBase< DATA_TYPE,  
SIZE > & v2) [inline]`

Compares v1 and v2 to see if they are exactly the same with zero tolerance.

**Parameters:**

*v1* the first vector  
*v2* the second vector

**Returns:**

true if v1 equals v2; false if they differ

Definition at line 449 of file VecOps.h.

```

451 {
452     for(unsigned i=0;i<SIZE;++i)
453     {
454         if(v1[i] != v2[i])
455         {
456             return false;
457         }
458     }
459     return true;
460
461     /* Would like this
462     return(vec[0] == _v[0] &&
463            vec[1] == _v[1] &&
464            vec[2] == _v[2]);
465     */
466 }

```

### 8.6.2.21 `template<class DATA_TYPE> bool operator==(const Tri<DATA_TYPE> & tri1, const Tri<DATA_TYPE> & tri2)`

Compare two triangles to see if they are EXACTLY the same.

In other words, this comparison is done with zero tolerance.

**Parameters:**

*tri1* the first triangle to compare  
*tri2* the second triangle to compare

**Returns:**

true if they are equal, false otherwise

Definition at line 92 of file TriOps.h.

```

93 {
94     return ( (tri1[0] == tri2[0]) &&
95             (tri1[1] == tri2[1]) &&
96             (tri1[2] == tri2[2]) );
97 }

```

**8.6.2.22** `template<class DATA_TYPE> bool operator==(const Sphere<  
DATA_TYPE > & s1, const Sphere< DATA_TYPE > & s2)  
[inline]`

Compare two spheres to see if they are EXACTLY the same.

In other words, this comparison is done with zero tolerance.

**Parameters:**

*s1* the first sphere to compare

*s2* the second sphere to compare

**Returns:**

true if they are equal, false otherwise

Definition at line 59 of file SphereOps.h.

```
60 {
61     return ( (s1.mCenter == s2.mCenter) && (s1.mRadius == s2.mRadius) );
62 }
```

**8.6.2.23** `template<typename DATA_TYPE> bool operator==(const Quat<  
DATA_TYPE > & q1, const Quat< DATA_TYPE > & q2) [inline]`

Compare two quaternions for equality.

**See also:**

`isEqual( Quat, Quat )`

Definition at line 616 of file QuatOps.h.

```
617     {
618         return bool( q1[0] == q2[0] &&
619                     q1[1] == q2[1] &&
620                     q1[2] == q2[2] &&
621                     q1[3] == q2[3] );
622     }
```



**8.6.2.24** `template<class DATA_TYPE> bool operator==(const Plane<DATA_TYPE> & p1, const Plane<DATA_TYPE> & p2) [inline]`

Compare two planes to see if they are EXACTLY the same.

In other words, this comparison is done with zero tolerance.

**Parameters:**

*p1* the first plane to compare

*p2* the second plane to compare

**Returns:**

true if they are equal, false otherwise

Definition at line 154 of file PlaneOps.h.

```
155 {
156     return ( (p1.mNorm == p2.mNorm) && (p1.mOffset == p2.mOffset) );
157 }
```

**8.6.2.25** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> bool operator==(const Matrix<DATA_TYPE, ROWS, COLS> & lhs, const Matrix<DATA_TYPE, ROWS, COLS> & rhs) [inline]`

Compare two mats.

Definition at line 440 of file MatrixOps.h.

```
441 {
442     for (unsigned int i = 0; i < ROWS*COLS; ++i)
443     {
444         if (lhs[i] != rhs[i])
445         {
446             return false;
447         }
448     }
449     return true;
450
451     /* Would like this
452     return( lhs[0] == rhs[0] &&
453            lhs[1] == rhs[1] &&
454            lhs[2] == rhs[2] );
455     */
456 }
```

**8.6.2.26** `template<class DATA_TYPE, typename ROT_ORDER> bool  
operator==(const EulerAngle< DATA_TYPE, ROT_ORDER > & v1,  
const EulerAngle< DATA_TYPE, ROT_ORDER > & v2) [inline]`

Compares v1 and v2 to see if they are exactly the same with zero tolerance.

**Parameters:**

**v1** the first rotation

**v2** the second rotation

**Returns:**

true if v1 equals v2; false if they differ

Definition at line 23 of file EulerAngleOps.h.

```
25 {
26     // @todo metaprogramming.
27     if (v1[0] != v2[0]) return false;
28     if (v1[1] != v2[1]) return false;
29     if (v1[2] != v2[2]) return false;
30     return true;
31 }
```

**8.6.2.27** `template<typename POS_TYPE, typename ROT_TYPE> bool  
operator==(const Coord< POS_TYPE, ROT_TYPE > & q1, const  
Coord< POS_TYPE, ROT_TYPE > & q2) [inline]`

Compare two quaternions for equality.

**See also:**

isEqual( Coord, Coord )

Definition at line 51 of file CoordOps.h.

```
53 {
54     return bool( q1.getPos() == q2.getPos() &&
55                 q1.getRot() == q2.getRot() );
56 }
```

```
8.6.2.28 template<class DATA_TYPE> bool operator==(const AxisAngle<
DATA_TYPE > & v1, const AxisAngle< DATA_TYPE > & v2)
[inline]
```

Compares v1 and v2 to see if they are exactly the same with zero tolerance.

**Parameters:**

**v1** the first vector

**v2** the second vector

**Returns:**

true if v1 equals v2; false if they differ

Definition at line 23 of file AxisAngleOps.h.

Referenced by gmtl::operator!=().

```
25 {
26     // @todo metaprogramming.
27     if (v1[0] != v2[0]) return false;
28     if (v1[1] != v2[1]) return false;
29     if (v1[2] != v2[2]) return false;
30     if (v1[3] != v2[3]) return false;
31     return true;
32 }
```

## 8.7 Generators: make( ... ), set( ... ).

Make get and set functions for all math types in [gmtl](#).

### Generic Generators (any type)

- `template<typename TARGET_TYPE, typename SOURCE_TYPE> TARGET_TYPE make (const SOURCE_TYPE &src, Type2Type< TARGET_TYPE > t=Type2Type< TARGET_TYPE >())`

*Construct an object from another object of a different type.*

- `template<typename ROTATION_TYPE, typename SOURCE_TYPE> ROTATION_TYPE makeRot (const SOURCE_TYPE &coord, Type2Type< ROTATION_TYPE > t=Type2Type< ROTATION_TYPE >())`

*Create a rotation datatype from another rotation datatype.*

- `template<typename ROTATION_TYPE> ROTATION_TYPE makeDirCos (const Vec< typename ROTATION_TYPE::DataType, 3 > &xDestAxis, const Vec< typename ROTATION_TYPE::DataType, 3 > &yDestAxis, const Vec< typename ROTATION_TYPE::DataType, 3 > &zDestAxis, const Vec< typename ROTATION_TYPE::DataType, 3 > &xSrcAxis=Vec< typename ROTATION_TYPE::DataType, 3 >(1, 0, 0), const Vec< typename ROTATION_TYPE::DataType, 3 > &ySrcAxis=Vec< typename ROTATION_TYPE::DataType, 3 >(0, 1, 0), const Vec< typename ROTATION_TYPE::DataType, 3 > &zSrcAxis=Vec< typename ROTATION_TYPE::DataType, 3 >(0, 0, 1), Type2Type< ROTATION_TYPE > t=Type2Type< ROTATION_TYPE >())`

*Create a rotation matrix or quaternion (or any other rotation data type) using direction cosines.*

- `template<typename TRANS_TYPE, typename SRC_TYPE> TRANS_TYPE makeTrans (const SRC_TYPE &arg, Type2Type< TRANS_TYPE > t=Type2Type< TRANS_TYPE >())`

*Make a translation datatype from another translation datatype.*

- `template<typename ROTATION_TYPE> ROTATION_TYPE makeRot (const Vec< typename ROTATION_TYPE::DataType, 3 > &from, const Vec< typename ROTATION_TYPE::DataType, 3 > &to)`

*Create a rotation datatype that will xform first vector to the second.*

- `template<typename DEST_TYPE, typename DATA_TYPE> DEST_TYPE & setRot (DEST_TYPE &result, const Vec< DATA_TYPE, 3 > &from, const Vec< DATA_TYPE, 3 > &to)`

*set a rotation datatype that will xform first vector to the second.*

## Vec Generators

- `template<typename DATA_TYPE> Vec< DATA_TYPE, 3 > makeVec (const Quat< DATA_TYPE > &quat)`

*create a vector from the vector component of a quaternion.*

- `template<typename DATA_TYPE, unsigned SIZE> Vec< DATA_TYPE, SIZE > makeNormal (Vec< DATA_TYPE, SIZE > vec)`

*create a normalized vector from the given vector.*

- `template<typename VEC_TYPE, typename DATA_TYPE, unsigned ROWS, unsigned COLS> VEC_TYPE & setTrans (VEC_TYPE &result, const Matrix< DATA_TYPE, ROWS, COLS > &arg)`

*Set vector using translation portion of the matrix.*

## Quat Generators

- `template<typename DATA_TYPE> Quat< DATA_TYPE > & setPure (Quat< DATA_TYPE > &quat, const Vec< DATA_TYPE, 3 > &vec)`

*Set pure quaternion.*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > makePure (const Vec< DATA_TYPE, 3 > &vec)`

*create a pure quaternion.*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > makeNormal (const Quat< DATA_TYPE > &quat)`

*create a pure quaternion.*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > makeConj (const Quat< DATA_TYPE > &quat)`

*quaternion complex conjugate.*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > makeInvert (const Quat< DATA_TYPE > &quat)`

*create quaternion from the inverse of another quaternion.*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > & set (Quat< DATA_TYPE > &result, const AxisAngle< DATA_TYPE > &axisAngle)`

*Convert an AxisAngle to a Quat.*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > & setRot (Quat< DATA_TYPE > &result, const AxisAngle< DATA_TYPE > &axisAngle)`

*Redundant duplication of the set(quat,axisangle) function, this is provided only for template compatibility.*

- `template<typename DATA_TYPE, typename ROT_ORDER> Quat< DATA_TYPE > & set (Quat< DATA_TYPE > &result, const EulerAngle< DATA_TYPE, ROT_ORDER > &euler)`

*Convert an EulerAngle rotation to a Quaternion rotation.*

- `template<typename DATA_TYPE, typename ROT_ORDER> Quat< DATA_TYPE > & setRot (Quat< DATA_TYPE > &result, const EulerAngle< DATA_TYPE, ROT_ORDER > &euler)`

*Redundant duplication of the set(quat,eulerangle) function, this is provided only for template compatibility.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Quat< DATA_TYPE > & set (Quat< DATA_TYPE > &quat, const Matrix< DATA_TYPE, ROWS, COLS > &mat)`

*Convert a Matrix to a Quat.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Quat< DATA_TYPE > & setRot (Quat< DATA_TYPE > &result, const Matrix< DATA_TYPE, ROWS, COLS > &mat)`

*Redundant duplication of the set(quat,mat) function, this is provided only for template compatibility.*

## AxisAngle Generators

- `template<typename DATA_TYPE> AxisAngle< DATA_TYPE > & set (AxisAngle< DATA_TYPE > &axisAngle, Quat< DATA_TYPE > quat)`

*Convert a rotation quaternion to an AxisAngle.*

- `template<typename DATA_TYPE> AxisAngle< DATA_TYPE > & setRot (AxisAngle< DATA_TYPE > &result, Quat< DATA_TYPE > quat)`

*Redundant duplication of the set(axisangle,quat) function, this is provided only for template compatibility.*

- `template<typename DATA_TYPE> AxisAngle< DATA_TYPE > makeNormal`  
`(const AxisAngle< DATA_TYPE > &a)`  
*make a normalized axisangle.*

## EulerAngle Generators

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, typename ROT_ORDER> EulerAngle< DATA_TYPE, ROT_ORDER > & set` `(EulerAngle< DATA_TYPE, ROT_ORDER > &euler, const Matrix< DATA_TYPE, ROWS, COLS > &mat)`  
*Convert Matrix to EulerAngle.*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, typename ROT_ORDER> EulerAngle< DATA_TYPE, ROT_ORDER > & setRot` `(EulerAngle< DATA_TYPE, ROT_ORDER > &result, const Matrix< DATA_TYPE, ROWS, COLS > &mat)`  
*Redundant duplication of the `set(eulerangle,quat)` function, this is provided only for template compatibility.*

## Matrix Generators

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, unsigned SIZE> Matrix< DATA_TYPE, ROWS, COLS > & setTrans` `(Matrix< DATA_TYPE, ROWS, COLS > &result, const Vec< DATA_TYPE, SIZE > &trans)`  
*Set matrix translation from vec.*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, unsigned SIZE> Matrix< DATA_TYPE, ROWS, COLS > & setScale` `(Matrix< DATA_TYPE, ROWS, COLS > &result, const Vec< DATA_TYPE, SIZE > &scale)`  
*Set the scale part of a matrix.*
- `template<typename MATRIX_TYPE, unsigned SIZE> MATRIX_TYPE makeScale` `(const Vec< typename MATRIX_TYPE::DataType, SIZE > &scale, Type2Type< MATRIX_TYPE > t=Type2Type< MATRIX_TYPE >())`  
*Create a scale matrix.*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & setScale` `(Matrix< DATA_TYPE, ROWS, COLS > &result, const DATA_TYPE scale)`  
*Create a scale matrix.*

- `template<typename MATRIX_TYPE> MATRIX_TYPE makeScale (const typename MATRIX_TYPE::DataType scale, Type2Type< MATRIX_TYPE > t=Type2Type< MATRIX_TYPE >())`

*Create a scale matrix.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & setRot (Matrix< DATA_TYPE, ROWS, COLS > &result, const AxisAngle< DATA_TYPE > &axisAngle)`

*Set the rotation portion of a rotation matrix using an axis and an angle (in radians).*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & set (Matrix< DATA_TYPE, ROWS, COLS > &result, const AxisAngle< DATA_TYPE > &axisAngle)`

*Convert an AxisAngle to a rotation matrix.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, typename ROT_ORDER> Matrix< DATA_TYPE, ROWS, COLS > & setRot (Matrix< DATA_TYPE, ROWS, COLS > &result, const EulerAngle< DATA_TYPE, ROT_ORDER > &euler)`

*Set (only) the rotation part of a matrix using an EulerAngle (angles are in radians).*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, typename ROT_ORDER> Matrix< DATA_TYPE, ROWS, COLS > & set (Matrix< DATA_TYPE, ROWS, COLS > &result, const EulerAngle< DATA_TYPE, ROT_ORDER > &euler)`

*Convert an EulerAngle to a rotation matrix.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> float makeYRot (const Matrix< DATA_TYPE, ROWS, COLS > &mat)`

*Extracts the yaw information from the matrix.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> float makeXRot (const Matrix< DATA_TYPE, ROWS, COLS > &mat)`

*Extracts the pitch information from the matrix.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> float makeZRot (const Matrix< DATA_TYPE, ROWS, COLS > &mat)`

*Extracts the roll information from the matrix.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & setDirCos (Matrix< DATA_TYPE, ROWS, COLS > &result, const Vec< DATA_TYPE, 3 > &xDestAxis, const`



```
Vec< DATA_TYPE, 3 > &yDestAxis, const Vec< DATA_TYPE, 3 > &zDest-
Axis, const Vec< DATA_TYPE, 3 > &xSrcAxis=Vec< DATA_TYPE, 3 >(1, 0,
0), const Vec< DATA_TYPE, 3 > &ySrcAxis=Vec< DATA_TYPE, 3 >(0, 1,
0), const Vec< DATA_TYPE, 3 > &zSrcAxis=Vec< DATA_TYPE, 3 >(0, 0,
1))
```

*create a rotation matrix that will rotate from SrcAxis to DestAxis.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & setAxes (Matrix< DATA_TYPE, ROWS, COLS > &result, const Vec< DATA_TYPE, 3 > &xAxis, const Vec< DATA_TYPE, 3 > &yAxis, const Vec< DATA_TYPE, 3 > &zAxis)`

*set the matrix given the raw coordinate axes.*

- `template<typename ROTATION_TYPE> ROTATION_TYPE makeAxes (const Vec< typename ROTATION_TYPE::DataType, 3 > &xAxis, const Vec< typename ROTATION_TYPE::DataType, 3 > &yAxis, const Vec< typename ROTATION_TYPE::DataType, 3 > &zAxis, Type2Type< ROTATION_TYPE > t=Type2Type< ROTATION_TYPE >())`

*set the matrix given the raw coordinate axes.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > makeTranspose (const Matrix< DATA_TYPE, ROWS, COLS > &m)`

*create a matrix transposed from the source.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > makeInverse (const Matrix< DATA_TYPE, ROWS, COLS > src, Type2Type< Matrix< DATA_TYPE, ROWS, COLS > > t=Type2Type< Matrix< DATA_TYPE, ROWS, COLS > >())`

*Creates a matrix that is the inverse of the given source matrix.*

- `template<typename DATATYPE, typename POS_TYPE, typename ROT_TYPE, unsigned MATCOLS, unsigned MATROWS> Matrix< DATATYPE, MATROWS, MATCOLS > & set (Matrix< DATATYPE, MATROWS, MATCOLS > &mat, const Coord< POS_TYPE, ROT_TYPE > &coord)`

*Convert a Coord to a Matrix.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & setRot (Matrix< DATA_TYPE, ROWS, COLS > &mat, const Quat< DATA_TYPE > &q)`

*Set the rotation portion of a matrix (3x3) from a rotation quaternion.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & set (Matrix< DATA_TYPE, ROWS, COLS > &mat, const Quat< DATA_TYPE > &q)`

*Convert a Quat to a rotation Matrix.*

## Coord Generators

- `template<typename DATATYPE, typename POS_TYPE, typename ROT_TYPE, unsigned MATCOLS, unsigned MATROWS> Coord< POS_TYPE, ROT_TYPE > & set (Coord< POS_TYPE, ROT_TYPE > &eulercoord, const Matrix< DATATYPE, MATROWS, MATCOLS > &mat)`

*convert Matrix to Coord.*

- `template<typename DATATYPE, typename POS_TYPE, typename ROT_TYPE, unsigned MATCOLS, unsigned MATROWS> Coord< POS_TYPE, ROT_TYPE > & setRot (Coord< POS_TYPE, ROT_TYPE > &result, const Matrix< DATATYPE, MATROWS, MATCOLS > &mat)`

*Redundant duplication of the set(coord,mat) function, this is provided only for template compatibility.*

### 8.7.1 Detailed Description

Make get and set functions for all math types in [gmtl](#).

### 8.7.2 Function Documentation

**8.7.2.1** `template<typename TARGET_TYPE, typename SOURCE_TYPE> TARGET_TYPE make (const SOURCE_TYPE & src, Type2Type< TARGET_TYPE > t = Type2Type< TARGET_TYPE >()) [inline]`

Construct an object from another object of a different type.

This allows us to automatically convert from any type to any other type.

**Precondition:**

must have a [set\(\)](#) function defined that converts between the two types.

**See also:**

`OpenSGGenerate.h` for an example

Definition at line 68 of file `Generate.h`.

References `gmtl::ignore_unused_variable_warning()`, and `gmtl::set()`.

```

70  {
71      gmtl::ignore_unused_variable_warning(t);
72      TARGET_TYPE target;
73      return set( target, src );
74  }

```

**8.7.2.2** `template<typename ROTATION_TYPE> ROTATION_TYPE  
makeAxes( const Vec< typename ROTATION_TYPE::DataType, 3  
> & xAxis, const Vec< typename ROTATION_TYPE::DataType, 3  
> & yAxis, const Vec< typename ROTATION_TYPE::DataType,  
3 > & zAxis, Type2Type< ROTATION_TYPE > t = Type2Type<  
ROTATION_TYPE >()) [inline]`

set the matrix given the raw coordinate axes.

**Postcondition:**

this function only produces 3x3, 3x4, 4x3, and 4x4 matrices, and is undefined otherwise  
these axes are copied direct to the 3x3 in the matrix

Definition at line 1013 of file Generate.h.

References gmtl::ignore\_unused\_variable\_warning(), and gmtl::setAxes().

```

1017  {
1018      gmtl::ignore_unused_variable_warning(t);
1019      ROTATION_TYPE temporary;
1020      return setAxes( temporary, xAxis, yAxis, zAxis );
1021  }

```

**8.7.2.3** `template<typename DATA_TYPE> Quat<DATA_TYPE> makeConj  
(const Quat< DATA_TYPE > & quat) [inline]`

quaternion complex conjugate.

**Postcondition:**

set result to the complex conjugate of result.  
result'[x,y,z,w] == result[-x,-y,-z,w]

**See also:**

[Quat](#)

Definition at line 297 of file Generate.h.

References gmtl::conj().

```

298     {
299         Quat<DATA_TYPE> temporary( quat );
300         return conj( temporary );
301     }

```

**8.7.2.4** `template<typename ROTATION_TYPE> ROTATION_TYPE  
makeDirCos (const Vec< typename ROTATION_TYPE::DataType, 3 >  
& xDestAxis, const Vec< typename ROTATION_TYPE::DataType, 3 >  
& yDestAxis, const Vec< typename ROTATION_TYPE::DataType, 3 >  
& zDestAxis, const Vec< typename ROTATION_TYPE::DataType,  
3 > & xSrcAxis = Vec<typename ROTATION_TYPE::DataType,  
3>(1,0,0), const Vec< typename ROTATION_TYPE::DataType,  
3 > & ySrcAxis = Vec<typename ROTATION_TYPE::DataType,  
3>(0,1,0), const Vec< typename ROTATION_TYPE::DataType, 3 > &  
zSrcAxis = Vec<typename ROTATION_TYPE::DataType, 3>(0,0,1),  
Type2Type< ROTATION_TYPE > t = Type2Type< ROTATION_TYPE  
>()) [inline]`

Create a rotation matrix or quaternion (or any other rotation data type) using direction cosines.

**Parameters:**

*DestAxis* required to specify

*SrcAxis* optional to specify

**Precondition:**

specify 1 axis (3 vectors), or 2 axes (6 vectors).

**Postcondition:**

Creates a rotation from SrcAxis to DestAxis

this function only produces 3x3, 3x4, 4x3, and 4x4 matrices, and is undefined otherwise

Definition at line 97 of file Generate.h.

References gmtl::ignore\_unused\_variable\_warning(), and gmtl::setDirCos().

```

104     {
105         gmtl::ignore_unused_variable_warning(t);
106         ROTATION_TYPE temporary;
107         return setDirCos( temporary, xDestAxis, yDestAxis, zDestAxis, xSrcAxis, ySrcAxis, zSrcAxis );
108     }

```

**8.7.2.5** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS> makeInverse (const Matrix< DATA_TYPE, ROWS, COLS > src, Type2Type< Matrix< DATA_TYPE, ROWS, COLS > > t = Type2Type< Matrix< DATA_TYPE, ROWS, COLS > >()) [inline]`

Creates a matrix that is the inverse of the given source matrix.

**Parameters:**

*src* the matrix to compute the inverse of

**Returns:**

the inverse of source

Definition at line 1042 of file Generate.h.

References `gmtl::invert()`.

```
1044     {
1045         Matrix<DATA_TYPE, ROWS, COLS> result;
1046         return invert( result, src );
1047     }
```

**8.7.2.6** `template<typename DATA_TYPE> Quat<DATA_TYPE> makeInvert (const Quat< DATA_TYPE > & quat) [inline]`

create quaternion from the inverse of another quaternion.

**Postcondition:**

returns the multiplicative inverse of quat

**See also:**

[Quat](#)

Definition at line 308 of file Generate.h.

References `gmtl::invert()`.

```
309     {
310         Quat<DATA_TYPE> temporary( quat );
311         return invert( temporary );
312     }
```

### 8.7.2.7 `template<typename DATA_TYPE> AxisAngle<DATA_TYPE> makeNormal (const AxisAngle< DATA_TYPE > & a)`

make a normalized axisangle.

Definition at line 549 of file Generate.h.

References gmtl::makeNormal().

```
550     {
551         return AxisAngle<DATA_TYPE>( a.getAngle(), makeNormal( a.getAxis() ) );
552     }
```

### 8.7.2.8 `template<typename DATA_TYPE> Quat<DATA_TYPE> makeNormal (const Quat< DATA_TYPE > & quat) [inline]`

create a pure quaternion.

**Postcondition:**

`quat = [v,0] = [v0,v1,v2,0]`

Definition at line 285 of file Generate.h.

References gmtl::normalize().

```
286     {
287         Quat<DATA_TYPE> temporary( quat );
288         return normalize( temporary );
289     }
```

### 8.7.2.9 `template<typename DATA_TYPE, unsigned SIZE> Vec<DATA_TYPE, SIZE> makeNormal (Vec< DATA_TYPE, SIZE > vec) [inline]`

create a normalized vector from the given vector.

Definition at line 210 of file Generate.h.

References gmtl::normalize().

Referenced by gmtl::makeNormal().

```
211     {
212         normalize( vec );
213         return vec;
214     }
```

**8.7.2.10** `template<typename DATA_TYPE> Quat<DATA_TYPE> makePure  
(const Vec< DATA_TYPE, 3 > & vec) [inline]`

create a pure quaternion.

**Postcondition:**

`quat = [v,0] = [v0,v1,v2,0]`

Definition at line 276 of file Generate.h.

```
277 {
278     return Quat<DATA_TYPE>( vec[0], vec[1], vec[2], 0 );
279 }
```

**8.7.2.11** `template<typename ROTATION_TYPE> ROTATION_TYPE  
makeRot (const Vec< typename ROTATION_TYPE::DataType, 3 > &  
from, const Vec< typename ROTATION_TYPE::DataType, 3 > & to)  
[inline]`

Create a rotation datatype that will xform first vector to the second.

This function creates a temporary.

Definition at line 138 of file Generate.h.

References `gmtl::setRot()`.

```
140 {
141     ROTATION_TYPE temporary;
142     return setRot( temporary, from, to );
143 }
```

**8.7.2.12** `template<typename ROTATION_TYPE, typename SOURCE_TYPE>  
ROTATION_TYPE makeRot (const SOURCE_TYPE &  
coord, Type2Type< ROTATION_TYPE > t = Type2Type<  
ROTATION_TYPE >()) [inline]`

Create a rotation datatype from another rotation datatype.

**Postcondition:**

converts the source rotation to a to another type (usually [Matrix](#), [Quat](#), Euler, [AxisAngle](#)),  
returns a temporary object.

Definition at line 81 of file Generate.h.

References `gmtl::ignore_unused_variable_warning()`, and `gmtl::set()`.

```

83     {
84         gmtl::ignore_unused_variable_warning(t);
85         ROTATION_TYPE temporary;
86         return set( temporary, coord );
87     }
```

**8.7.2.13** `template<typename MATRIX_TYPE> MATRIX_TYPE makeScale  
(const typename MATRIX_TYPE::DataType scale, Type2Type<  
MATRIX_TYPE > t = Type2Type< MATRIX_TYPE >()) [inline]`

Create a scale matrix.

Definition at line 740 of file Generate.h.

References `gmtl::ignore_unused_variable_warning()`, and `gmtl::setScale()`.

```

742     {
743         gmtl::ignore_unused_variable_warning(t);
744         MATRIX_TYPE temporary;
745         return setScale( temporary, scale );
746     }
```

**8.7.2.14** `template<typename MATRIX_TYPE, unsigned SIZE>  
MATRIX_TYPE makeScale (const Vec< typename  
MATRIX_TYPE::DataType, SIZE > & scale, Type2Type<  
MATRIX_TYPE > t = Type2Type< MATRIX_TYPE >()) [inline]`

Create a scale matrix.

Definition at line 717 of file Generate.h.

References `gmtl::ignore_unused_variable_warning()`, and `gmtl::setScale()`.

```

719     {
720         gmtl::ignore_unused_variable_warning(t);
```



```

721     MATRIX_TYPE temporary;
722     return setScale( temporary, scale );
723 }

```

**8.7.2.15** `template<typename TRANS_TYPE, typename SRC_TYPE>`  
`TRANS_TYPE makeTrans (const SRC_TYPE & arg, Type2Type<`  
`TRANS_TYPE > t = Type2Type< TRANS_TYPE >()) [inline]`

Make a translation datatype from another translation datatype.

Typically this is from [Matrix](#) to [Vec](#) or [Vec](#) to [Matrix](#). This function reads only translation information from the src datatype.

**Parameters:**

*arg* the matrix to extract the translation from

**Precondition:**

if making an n x n matrix, then for

- **vector is homogeneous:** SIZE of vector needs to equal number of [Matrix](#) ROWS - 1
- **vector has scale component:** SIZE of vector needs to equal number of [Matrix](#) ROWS

if making an n x n+1 matrix, then for

- **vector is homogeneous:** SIZE of vector needs to equal number of [Matrix](#) ROWS
- **vector has scale component:** SIZE of vector needs to equal number of [Matrix](#) ROWS + 1

**Postcondition:**

if preconditions are not met, then function is undefined (will not compile)

Definition at line 126 of file Generate.h.

References `gmtl::ignore_unused_variable_warning()`, and `gmtl::setTrans()`.

```

128 {
129     gmtl::ignore_unused_variable_warning(t);
130     TRANS_TYPE temporary;
131     return setTrans( temporary, arg );
132 }

```

**8.7.2.16** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS>`  
`Matrix<DATA_TYPE, ROWS, COLS> makeTranspose (const`  
`Matrix< DATA_TYPE, ROWS, COLS > & m) [inline]`

create a matrix transposed from the source.

**Postcondition:**

returns the transpose of m

**See also:**

[Quat](#)

Definition at line 1028 of file Generate.h.

References `gmtl::transpose()`.

```
1029  {
1030      Matrix<DATA_TYPE, ROWS, COLS> temporary( m );
1031      return transpose( temporary );
1032  }
```

**8.7.2.17** `template<typename DATA_TYPE> Vec<DATA_TYPE, 3> makeVec`  
`(const Quat< DATA_TYPE > & quat) [inline]`

create a vector from the vector component of a quaternion.

**Postcondition:**

$\text{quat} = [v, 0] = [v_0, v_1, v_2, 0]$

**Todo:**

should this be called convert?

Definition at line 202 of file Generate.h.

References `gmtl::Xelt`, `gmtl::Yelt`, and `gmtl::Zelt`.

```
203  {
204      return Vec<DATA_TYPE, 3>( quat[Xelt], quat[Yelt], quat[Zelt] );
205  }
```

**8.7.2.18** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS>`  
`float makeXRot (const Matrix< DATA_TYPE, ROWS, COLS > &`  
`mat) [inline]`

Extracts the pitch information from the matrix.

**Postcondition:**

Returned value is from -180 to 180, where 0 is none.

Definition at line 888 of file Generate.h.

References `gmtl::Math::aCos()`, `gmtl::cross()`, `gmtl::normalize()`, and `gmtl::xform()`.

```

889  {
890      const gmtl::Vec3f forward_point(0.0f, 0.0f, -1.0f);    // -Z
891      const gmtl::Vec3f origin_point(0.0f, 0.0f, 0.0f);
892      gmtl::Vec3f end_point, start_point;
893
894      gmtl::xform(end_point, mat, forward_point);
895      gmtl::xform(start_point, mat, origin_point);
896      gmtl::Vec3f direction_vector = end_point - start_point;
897
898      // Constrain the direction to YZ-plane only.
899      direction_vector[0] = 0.0f;                          // Eliminate X value
900      gmtl::normalize(direction_vector);
901      float x_rot = gmtl::Math::aCos(gmtl::dot(direction_vector,
902                                              forward_point));
903
904      gmtl::Vec3f which_side = gmtl::cross(forward_point, direction_vector);
905
906      // If direction vector to "bottom" (negative) side of forward
907      if ( which_side[0] < 0.0f )
908      {
909          x_rot = -x_rot;
910      }
911
912      return x_rot;
913  }
```

**8.7.2.19** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS>`  
`float makeYRot (const Matrix< DATA_TYPE, ROWS, COLS > &`  
`mat) [inline]`

Extracts the yaw information from the matrix.

**Postcondition:**

Returned value is from -180 to 180, where 0 is none.

Definition at line 856 of file Generate.h.

References `gmtl::Math::aCos()`, `gmtl::cross()`, `gmtl::normalize()`, and `gmtl::xform()`.

```

857     {
858         const gmtl::Vec3f forward_point(0.0f, 0.0f, -1.0f);    // -Z
859         const gmtl::Vec3f origin_point(0.0f, 0.0f, 0.0f);
860         gmtl::Vec3f end_point, start_point;
861
862         gmtl::xform(end_point, mat, forward_point);
863         gmtl::xform(start_point, mat, origin_point);
864         gmtl::Vec3f direction_vector = end_point - start_point;
865
866         // Constrain the direction to XZ-plane only.
867         direction_vector[1] = 0.0f;                          // Eliminate Y value
868         gmtl::normalize(direction_vector);
869         float y_rot = gmtl::Math::aCos(gmtl::dot(direction_vector,
870                                                  forward_point));
871
872         gmtl::Vec3f which_side = gmtl::cross(forward_point, direction_vector);
873
874         // If direction vector to "right" (negative) side of forward
875         if ( which_side[1] < 0.0f )
876         {
877             y_rot = -y_rot;
878         }
879
880         return y_rot;
881     }

```

**8.7.2.20** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS>`  
**`float makeZRot (const Matrix< DATA_TYPE, ROWS, COLS > &`**  
**`mat) [inline]`**

Extracts the roll information from the matrix.

**Postcondition:**

Returned value is from -180 to 180, where 0 is no roll.

Definition at line 920 of file Generate.h.

References `gmtl::Math::aCos()`, `gmtl::cross()`, `gmtl::normalize()`, and `gmtl::xform()`.

```

921     {
922         const gmtl::Vec3f forward_point(0.0f, 0.0f, -1.0f);    // -Z
923         const gmtl::Vec3f origin_point(0.0f, 0.0f, 0.0f);
924         gmtl::Vec3f end_point, start_point;
925
926         gmtl::xform(end_point, mat, forward_point);

```

```

927     gmtl::xform(start_point, mat, origin_point);
928     gmtl::Vec3f direction_vector = end_point - start_point;
929
930     // Constrain the direction to XY-plane only.
931     direction_vector[2] = 0.0f;           // Eliminate Z value
932     gmtl::normalize(direction_vector);
933     float z_rot = gmtl::Math::aCos(gmtl::dot(direction_vector,
934                                             forward_point));
935
936     gmtl::Vec3f which_side = gmtl::cross(forward_point, direction_vector);
937
938     // If direction vector to "right" (negative) side of forward
939     if ( which_side[2] < 0.0f )
940     {
941         z_rot = -z_rot;
942     }
943
944     return z_rot;
945 }

```

**8.7.2.21** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS>`  
`Matrix<DATA_TYPE, ROWS, COLS>& set (Matrix< DATA_TYPE,`  
`ROWS, COLS > & mat, const Quat< DATA_TYPE > & q)`

Convert a [Quat](#) to a rotation [Matrix](#).

**Precondition:**

only 3x3, 3x4, 4x3, or 4x4 matrices are allowed, function is undefined otherwise.

**Postcondition:**

[Matrix](#) is entirely overwritten.

**Todo:**

Implement using setRot

Definition at line 1111 of file Generate.h.

References gmtl::setRot().

```

1112     {
1113         setRot( mat, q );
1114
1115         if (ROWS == 4)
1116         {
1117             mat( 3, 0 ) = DATA_TYPE(0.0);
1118             mat( 3, 1 ) = DATA_TYPE(0.0);
1119             mat( 3, 2 ) = DATA_TYPE(0.0);

```

```

1120     }
1121
1122     if (COLS == 4)
1123     {
1124         mat( 0, 3 ) = DATA_TYPE(0.0);
1125         mat( 1, 3 ) = DATA_TYPE(0.0);
1126         mat( 2, 3 ) = DATA_TYPE(0.0);
1127     }
1128
1129     if (ROWS == 4 && COLS == 4)
1130         mat( 3, 3 ) = DATA_TYPE(1.0);
1131
1132     return mat;
1133 }

```

**8.7.2.22** `template<typename DATATYPE, typename POS_TYPE, typename ROT_TYPE, unsigned MATCOLS, unsigned MATROWS>`  
`Matrix<DATATYPE, MATROWS, MATCOLS>& set (Matrix<`  
`DATATYPE, MATROWS, MATCOLS > & mat, const Coord<`  
`POS_TYPE, ROT_TYPE > & coord) [inline]`

Convert a [Coord](#) to a [Matrix](#).

See also:

[Coord](#) , [Matrix](#)

Definition at line 1054 of file Generate.h.

References `gmtl::identity()`, `gmtl::setRot()`, and `gmtl::setTrans()`.

```

1056     {
1057         // set to ident first...
1058         gmtl::identity( mat );
1059
1060         // set translation
1061         // @todo metaprogram this out for 3x3 and 2x2 matrices
1062         if (MATCOLS == 4)
1063         {
1064             setTrans( mat, coord.getPos() );// filtered (only sets the trans part).
1065         }
1066         setRot( mat, coord.getRot() ); // filtered (only sets the rot part).
1067         return mat;
1068     }

```

**8.7.2.23** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, typename ROT_ORDER> Matrix<DATA_TYPE, ROWS, COLS>& set (Matrix< DATA_TYPE, ROWS, COLS > & result, const EulerAngle< DATA_TYPE, ROT_ORDER > & euler) [inline]`

Convert an [EulerAngle](#) to a rotation matrix.

**Postcondition:**

this function only writes to 3x3, 3x4, 4x3, and 4x4 matrices, and is undefined otherwise

Definition at line 845 of file Generate.h.

References `gmtl::identity()`, and `gmtl::setRot()`.

```
846  {
847      gmtl::identity( result );
848      return setRot( result, euler );
849  }
```

**8.7.2.24** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& set (Matrix< DATA_TYPE, ROWS, COLS > & result, const AxisAngle< DATA_TYPE > & axisAngle) [inline]`

Convert an [AxisAngle](#) to a rotation matrix.

**Postcondition:**

this function only writes to 3x3, 3x4, 4x3, and 4x4 matrices, and is undefined otherwise

**Precondition:**

[AxisAngle](#) must be normalized (the axis part), results are undefined if not.

Definition at line 786 of file Generate.h.

References `gmtl::identity()`, and `gmtl::setRot()`.

```
787  {
788      gmtl::identity( result );
789      return setRot( result, axisAngle );
790  }
```

**8.7.2.25** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS>`  
`Quat<DATA_TYPE>& set (Quat< DATA_TYPE > & quat, const`  
`Matrix< DATA_TYPE, ROWS, COLS > & mat) [inline]`

Convert a [Matrix](#) to a [Quat](#).

Sets the rotation quaternion using the given matrix (3x3, 3x4, 4x3, or 4x4 are all valid sizes).

Definition at line 414 of file Generate.h.

References `gmlASSERT`, `gml::Math::sqrt()`, `gml::Welt`, `gml::Xelt`, `gml::Yelt`, and `gml::Zelt`.

```

415     {
416         gmlASSERT( ((ROWS == 3 && COLS == 3) ||
417                     (ROWS == 3 && COLS == 4) ||
418                     (ROWS == 4 && COLS == 3) ||
419                     (ROWS == 4 && COLS == 4)) &&
420                     "pre conditions not met on set( quat, pos, mat ) which only sets a quaterni
421
422     DATA_TYPE tr( mat( 0, 0 ) + mat( 1, 1 ) + mat( 2, 2 ) ), s( 0.0f );
423
424     // If diagonal is positive
425     if (tr > (DATA_TYPE)0.0)
426     {
427         s = Math::sqrt( tr + (DATA_TYPE)1.0 );
428         quat[Welt] = s * (DATA_TYPE)0.5;
429         s = DATA_TYPE(0.5) / s;
430
431         quat[Xelt] = (mat( 2, 1 ) - mat( 1, 2 )) * s;
432         quat[Yelt] = (mat( 0, 2 ) - mat( 2, 0 )) * s;
433         quat[Zelt] = (mat( 1, 0 ) - mat( 0, 1 )) * s;
434     }
435
436     // when Diagonal is negative
437     else
438     {
439         static const unsigned int nxt[3] = { 1, 2, 0 };
440         unsigned int i( Xelt ), j, k;
441
442         if (mat( 1, 1 ) > mat( 0, 0 ))
443             i = 1;
444
445         if (mat( 2, 2 ) > mat( i, i ))
446             i = 2;
447
448         j = nxt[i];
449         k = nxt[j];
450
451         s = Math::sqrt( (mat( i, i )-(mat( j, j )+mat( k, k ))) + (DATA_TYPE)1.0 );
452
453         DATA_TYPE q[4];
454         q[i] = s * (DATA_TYPE)0.5;

```



```

455
456         if (s != (DATA_TYPE)0.0)
457             s = DATA_TYPE(0.5) / s;
458
459         q[3] = (mat( k, j ) - mat( j, k )) * s;
460         q[j] = (mat( j, i ) + mat( i, j )) * s;
461         q[k] = (mat( k, i ) + mat( i, k )) * s;
462
463         quat[Xelt] = q[0];
464         quat[Yelt] = q[1];
465         quat[Zelt] = q[2];
466         quat[Welt] = q[3];
467     }
468
469     return quat;
470 }

```

**8.7.2.26** `template<typename DATA_TYPE, typename ROT_ORDER>`  
`Quat<DATA_TYPE>& set (Quat< DATA_TYPE > &result, const`  
`EulerAngle< DATA_TYPE, ROT_ORDER > &euler) [inline]`

Convert an [EulerAngle](#) rotation to a Quaternion rotation.

Sets a rotation quaternion using euler angles (each angle in radians).

**Precondition:**

pass in your angles in the same order as the RotationOrder you specify

Definition at line 350 of file Generate.h.

References `gmtl::Math::cos()`, `gmtl::ASSERT`, `gmtl::normalize()`, `gmtl::Math::sin()`,  
`gmtl::Welt`, `gmtl::Xelt`, `gmtl::Yelt`, and `gmtl::Zelt`.

```

351     {
352         // this might be faster if put into the switch statement... (testme)
353         const int& order = ROT_ORDER::ID;
354         const DATA_TYPE xRot = (order == XYZ::ID) ? euler[0] : ((order == ZXY::ID) ? euler[1] : euler[2]);
355         const DATA_TYPE yRot = (order == XYZ::ID) ? euler[1] : ((order == ZXY::ID) ? euler[2] : euler[1]);
356         const DATA_TYPE zRot = (order == XYZ::ID) ? euler[2] : ((order == ZXY::ID) ? euler[0] : euler[0]);
357
358         // this could be written better for each rotation order, but this is really general...
359         Quat<DATA_TYPE> qx, qy, qz;
360
361         // precompute half angles
362         DATA_TYPE xOver2 = xRot * (DATA_TYPE)0.5;
363         DATA_TYPE yOver2 = yRot * (DATA_TYPE)0.5;
364         DATA_TYPE zOver2 = zRot * (DATA_TYPE)0.5;
365
366         // set the pitch quat

```

```

367     qx[Xelt] = Math::sin( xOver2 );
368     qx[Yelt] = (DATA_TYPE)0.0;
369     qx[Zelt] = (DATA_TYPE)0.0;
370     qx[Welt] = Math::cos( xOver2 );
371
372     // set the yaw quat
373     qy[Xelt] = (DATA_TYPE)0.0;
374     qy[Yelt] = Math::sin( yOver2 );
375     qy[Zelt] = (DATA_TYPE)0.0;
376     qy[Welt] = Math::cos( yOver2 );
377
378     // set the roll quat
379     qz[Xelt] = (DATA_TYPE)0.0;
380     qz[Yelt] = (DATA_TYPE)0.0;
381     qz[Zelt] = Math::sin( zOver2 );
382     qz[Welt] = Math::cos( zOver2 );
383
384     // compose the three in pyr order...
385     switch (order)
386     {
387     case XYZ::ID: result = qx * qy * qz; break;
388     case ZYX::ID: result = qz * qy * qx; break;
389     case ZXY::ID: result = qz * qx * qy; break;
390     default:
391         gmtlASSERT( false && "unknown rotation order passed to setRot" );
392         break;
393     }
394
395     // ensure the quaternion is normalized
396     normalize( result );
397     return result;
398 }

```

**8.7.2.27** `template<typename DATA_TYPE> Quat<DATA_TYPE>& set`  
`(Quat< DATA_TYPE > &result, const AxisAngle< DATA_TYPE > &`  
`axisAngle) [inline]`

Convert an [AxisAngle](#) to a [Quat](#).

sets a rotation quaternion from an angle and an axis.

**Precondition:**

AxisAngle::axis must be normalized to length == 1 prior to calling this.

**Postcondition:**

$q = [ \cos(\text{rad}/2), \sin(\text{rad}/2) * [x,y,z] ]$

Definition at line 319 of file Generate.h.

References gmtl::Math::cos(), gmtl::ASSERT, gmtl::lengthSquared(), gmtl::Math::sin(), gmtl::Welt, gmtl::Xelt, gmtl::Yelt, and gmtl::Zelt.

```

320  {
321      gmtl::ASSERT( (Math::isEqual( lengthSquared( axisAngle.GetAxis() ), (DATA_TYPE)1.0, (DATA_TYPE)0.00000001 ) ),
322                  "you must pass in a normalized vector to setRot( quat, rad, vec )" );
323
324      DATA_TYPE half_angle = axisAngle.getAngle() * (DATA_TYPE)0.5;
325      DATA_TYPE sin_half_angle = Math::sin( half_angle );
326
327      result[Welt] = Math::cos( half_angle );
328      result[Xelt] = sin_half_angle * axisAngle.GetAxis()[0];
329      result[Yelt] = sin_half_angle * axisAngle.GetAxis()[1];
330      result[Zelt] = sin_half_angle * axisAngle.GetAxis()[2];
331
332      // should automagically be normalized (unit magnitude) now...
333      return result;
334  }

```

**8.7.2.28** `template<typename DATATYPE, typename POS_TYPE, typename ROT_TYPE, unsigned MATCOLS, unsigned MATROWS>  
Coord<POS_TYPE, ROT_TYPE>& set ( Coord< POS_TYPE,  
ROT_TYPE > & eulercoord, const Matrix< DATATYPE, MATROWS,  
MATCOLS > & mat) [inline]`

convert [Matrix](#) to [Coord](#).

Definition at line 1144 of file Generate.h.

References gmtl::set(), and gmtl::setTrans().

Referenced by gmtl::AxisAngle< DATA\_TYPE >::set().

```

1145  {
1146      gmtl::setTrans( eulercoord.pos(), mat );
1147      gmtl::set( eulercoord.rot(), mat );
1148      return eulercoord;
1149  }

```

**8.7.2.29** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, typename ROT_ORDER> EulerAngle<DATA_TYPE,  
ROT_ORDER>& set ( EulerAngle< DATA_TYPE, ROT_ORDER  
> & euler, const Matrix< DATA_TYPE, ROWS, COLS > & mat)  
[inline]`

Convert [Matrix](#) to [EulerAngle](#).

Set the Euler Angle from the given rotation portion (3x3) of the matrix.

**Parameters:**

*input* order, mat  
*output* param0, param1, param2

**Precondition:**

pass in your args in the same order as the RotationOrder you specify

**Postcondition:**

this function only reads 3x3, 3x4, 4x3, and 4x4 matrices, and is undefined otherwise NOTE: Angles are returned in radians (this is always true in GMTL).

Definition at line 571 of file Generate.h.

References gmtl::Math::aSin(), gmtl::Math::aTan2(), gmtl::Math::cos(), gmtlASSERT, and gmtl::Math::sin().

```

573     {
574         // @todo set this a compile time assert...
575         gmtlASSERT( ROWS >= 3 && COLS >= 3 && ROWS <= 4 && COLS <= 4 &&
576             "this is undefined for Matrix smaller than 3x3 or bigger than 4x4" );
577
578         DATA_TYPE sx;
579         DATA_TYPE cz;
580
581         // @todo metaprogram this!
582         const int& order = ROT_ORDER::ID;
583         switch (order)
584         {
585             case XYZ::ID:
586             {
587                 euler[2] = Math::aTan2( -mat(0,1), mat(0,0) );           // -(cy*sz)/(cy*cz) - cy fa
588                 euler[0] = Math::aTan2( -mat(1,2), mat(2,2) );           // -(sx*cy)/(cx*cy) - cy fa
589                 cz = Math::cos( euler[2] );
590                 euler[1] = Math::aTan2( mat(0,2), mat(0,0) / cz );       // (sy)/((cy*cz)/cz)
591             }
592             break;
593             case ZYX::ID:
594             {
595                 euler[0] = Math::aTan2( mat(1,0), mat(0,0) );           // (cy*sz)/(cy*cz) - cy fa
596                 euler[2] = Math::aTan2( mat(2,1), mat(2,2) );           // (sx*cy)/(cx*cy) - cy fa
597                 sx = Math::sin( euler[2] );
598                 euler[1] = Math::aTan2( -mat(2,0), mat(2,1) / sx );     // -(-sy)/((sx*cy)/sx)
599             }
600             break;
601             case ZXY::ID:
602             {
603                 // Extract the rotation directly from the matrix
604                 DATA_TYPE x_angle;
605                 DATA_TYPE y_angle;

```

```

606         DATA_TYPE z_angle;
607         DATA_TYPE cos_y, sin_y;
608         DATA_TYPE cos_x, sin_x;
609         DATA_TYPE cos_z, sin_z;
610
611         sin_x = mat(2,1);
612         x_angle = Math::aSin( sin_x );      // Get x angle
613         cos_x = Math::cos( x_angle );
614
615         // Check if cos_x = Zero
616         if (cos_x != 0.0f)      // ASSERT: cos_x != 0
617         {
618             // Get y Angle
619             cos_y = mat(2,2) / cos_x;
620             sin_y = -mat(2,0) / cos_x;
621             y_angle = Math::aTan2( cos_y, sin_y );
622
623             // Get z Angle
624             cos_z = mat(1,1) / cos_x;
625             sin_z = -mat(0,1) / cos_x;
626             z_angle = Math::aTan2( cos_z, sin_z );
627         }
628         else
629         {
630             // Arbitrarily set z_angle = 0
631             z_angle = 0;
632
633             // Get y Angle
634             cos_y = mat(0,0);
635             sin_y = mat(1,0);
636             y_angle = Math::aTan2( cos_y, sin_y );
637         }
638
639         euler[1] = x_angle;
640         euler[2] = y_angle;
641         euler[0] = z_angle;
642     }
643     break;
644     default:
645         gmtlASSERT( false && "unknown rotation order passed to setRot" );
646         break;
647     }
648     return euler;
649 }

```

**8.7.2.30** `template<typename DATA_TYPE> AxisAngle<DATA_TYPE>& set(AxisAngle< DATA_TYPE > & axisAngle, Quat< DATA_TYPE > quat)` [inline]

Convert a rotation quaternion to an [AxisAngle](#).

**Postcondition:**

returns an angle in radians, and a normalized axis equivalent to the quaternion's rotation.

returns rad and xyz such that

- $\text{rad} = \text{acos}(w) * 2.0$
- $\text{vec} = v / (\text{asin}(w) * 2.0)$  [where  $v$  is the xyz or vector component of the quat]

axisAngle = quat;

Definition at line 498 of file Generate.h.

References gmtl::Math::abs(), gmtl::Math::aCos(), gmtlASSERT, gmtl::normalize(), gmtl::Math::sin(), gmtl::Welt, gmtl::Xelt, gmtl::Yelt, and gmtl::Zelt.

```

499     {
500         // set sure we don't get a NaN result from acos...
501         if (Math::abs( quat[Welt] ) > (DATA_TYPE)1.0)
502         {
503             gmtl::normalize( quat );
504         }
505         gmtlASSERT( Math::abs( quat[Welt] ) <= (DATA_TYPE)1.0 && "acos returns NaN when quat
506
507         // [acos( w ) * 2.0, v / (asin( w ) * 2.0)]
508
509         // set the angle - aCos is mathematically defined to be between 0 and PI
510         DATA_TYPE rad = Math::aCos( quat[Welt] ) * (DATA_TYPE)2.0;
511         axisAngle.setAngle( rad );
512
513         // set the axis: (use sin(rad) instead of asin(w))
514         DATA_TYPE sin_half_angle = Math::sin( rad * (DATA_TYPE)0.5 );
515         if (sin_half_angle >= (DATA_TYPE)0.0001) // because (PI >= rad >= 0)
516         {
517             DATA_TYPE sin_half_angle_inv = DATA_TYPE(1.0) / sin_half_angle;
518             axisAngle.setAxis( gmtl::Vec3f(
519                                     quat[Xelt] * sin_half_angle_inv,
520                                     quat[Yelt] * sin_half_angle_inv,
521                                     quat[Zelt] * sin_half_angle_inv ) );
522         }
523
524         // avoid NAN
525         else
526         {
527             // one of the terms should be a 1,
528             // so we can maintain unit-ness
529             // in case w is 0 (which here w is 0)
530             axisAngle.setAxis( gmtl::Vec3f(
531                                     DATA_TYPE( 1.0 ) /*- gmtl::Math::abs( quat[Welt] )*/ ,
532                                     (DATA_TYPE)0.0,
533                                     (DATA_TYPE)0.0 ) );
534         }
535         return axisAngle;
536     }

```

**8.7.2.31** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& setAxes (Matrix<DATA_TYPE, ROWS, COLS > & result, const Vec< DATA_TYPE, 3 > & xAxis, const Vec< DATA_TYPE, 3 > & yAxis, const Vec< DATA_TYPE, 3 > & zAxis) [inline]`

set the matrix given the raw coordinate axes.

**Postcondition:**

this function only produces 3x3, 3x4, 4x3, and 4x4 matrices, and is undefined otherwise  
these axes are copied direct to the 3x3 in the matrix

Definition at line 985 of file Generate.h.

References `gmtlASSERT`.

Referenced by `gmtl::makeAxes()`.

```

989     {
990         // @todo set this a compile time assert...
991         gmtlASSERT( ROWS >= 3 && COLS >= 3 && ROWS <= 4 && COLS <= 4 && "this is undefined for Matrix sm
992
993         result( 0, 0 ) = xAxis[0];
994         result( 1, 0 ) = xAxis[1];
995         result( 2, 0 ) = xAxis[2];
996
997         result( 0, 1 ) = yAxis[0];
998         result( 1, 1 ) = yAxis[1];
999         result( 2, 1 ) = yAxis[2];
1000
1001         result( 0, 2 ) = zAxis[0];
1002         result( 1, 2 ) = zAxis[1];
1003         result( 2, 2 ) = zAxis[2];
1004
1005         return result;
1006     }
```

**8.7.2.32** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS>  
 Matrix<DATA_TYPE, ROWS, COLS> & setDirCos (Matrix<  
 DATA_TYPE, ROWS, COLS > & result, const Vec< DATA_TYPE, 3  
 > & xDestAxis, const Vec< DATA_TYPE, 3 > & yDestAxis, const  
 Vec< DATA_TYPE, 3 > & zDestAxis, const Vec< DATA_TYPE, 3 > &  
 xSrcAxis = Vec<DATA_TYPE, 3>(1,0,0), const Vec< DATA_TYPE, 3 >  
 & ySrcAxis = Vec<DATA_TYPE, 3>(0,1,0), const Vec< DATA_TYPE,  
 3 > & zSrcAxis = Vec<DATA_TYPE, 3>(0,0,1)) [inline]`

create a rotation matrix that will rotate from SrcAxis to DestAxis.

xSrcAxis, ySrcAxis, zSrcAxis is the base rotation to go from and defaults to xSrc-Axis(1,0,0), ySrcAxis(0,1,0), zSrcAxis(0,0,1) if you only pass in 3 axes.

**Precondition:**

pass in 3 axes, and setDirCos will give you the rotation from MATRIX.IDENTITY to DestAxis

pass in 6 axes, and setDirCos will give you the rotation from your 3-axis rotation to your second 3-axis rotation

**Postcondition:**

this function only produces 3x3, 3x4, 4x3, and 4x4 matrices

Definition at line 954 of file Generate.h.

References gmtl::dot(), and gmtl::ASSERT.

Referenced by gmtl::makeDirCos().

```

960     {
961         // @todo set this a compile time assert...
962         gmtl::ASSERT( ROWS >= 3 && COLS >= 3 && ROWS <= 4 && COLS <= 4 && "this is undefined for"
963
964         DATA_TYPE Xa, Xb, Xy;    // Direction cosines of the secondary x-axis
965         DATA_TYPE Ya, Yb, Yy;    // Direction cosines of the secondary y-axis
966         DATA_TYPE Za, Zb, Zy;    // Direction cosines of the secondary z-axis
967
968         Xa = dot(xDestAxis, xSrcAxis); Xb = dot(xDestAxis, ySrcAxis); Xy = dot(xDestAxis, zSrcAxis);
969         Ya = dot(yDestAxis, xSrcAxis); Yb = dot(yDestAxis, ySrcAxis); Yy = dot(yDestAxis, zSrcAxis);
970         Za = dot(zDestAxis, xSrcAxis); Zb = dot(zDestAxis, ySrcAxis); Zy = dot(zDestAxis, zSrcAxis);
971
972         // Set the matrix correctly
973         result( 0, 0 ) = Xa; result( 0, 1 ) = Xb; result( 0, 2 ) = Xy;
974         result( 1, 0 ) = Ya; result( 1, 1 ) = Yb; result( 1, 2 ) = Yy;
975         result( 2, 0 ) = Za; result( 2, 1 ) = Zb; result( 2, 2 ) = Zy;
976
977         return result;
978     }

```



**8.7.2.33** `template<typename DATA_TYPE> Quat<DATA_TYPE>& setPure  
(Quat< DATA_TYPE > & quat, const Vec< DATA_TYPE, 3 > & vec)  
[inline]`

Set pure quaternion.

**Todo:**

Write test case for setPure

Definition at line 266 of file Generate.h.

```
267 {
268     quat.set( vec[0], vec[1], vec[2], 0 );
269     return quat;
270 }
```

**8.7.2.34** `template<typename DATATYPE, typename POS_TYPE, typename  
ROT_TYPE, unsigned MATCOLS, unsigned MATROWS>  
Coord<POS_TYPE, ROT_TYPE>& setRot (Coord< POS_TYPE,  
ROT_TYPE > & result, const Matrix< DATATYPE, MATROWS,  
MATCOLS > & mat) [inline]`

Redundant duplication of the set(coord,mat) function, this is provided only for template compatibility.

unless you're writing template functions, you should use set(coord,mat) for clarity.

Definition at line 1155 of file Generate.h.

References gmtl::set().

```
1156 {
1157     return set( result, mat );
1158 }
```

**8.7.2.35** `template<typename DATA_TYPE, unsigned ROWS, unsigned  
COLS> Matrix<DATA_TYPE, ROWS, COLS>& setRot (Matrix<  
DATA_TYPE, ROWS, COLS > & mat, const Quat< DATA_TYPE > &  
q)`

Set the rotation portion of a matrix (3x3) from a rotation quaternion.

**Precondition:**

only 3x3, 3x4, 4x3, or 4x4 matrices are allowed, function is undefined otherwise.

Definition at line 1074 of file Generate.h.

References gmtlASSERT, gmtl::Welt, gmtl::Xelt, gmtl::Yelt, and gmtl::Zelt.

```

1075 {
1076     gmtlASSERT( (ROWS == 3 && COLS == 3) ||
1077                (ROWS == 3 && COLS == 4) ||
1078                (ROWS == 4 && COLS == 3) ||
1079                (ROWS == 4 && COLS == 4) ) &&
1080                "pre conditions not met on set( mat, quat ) which only sets a quaternion to
1081
1082     // From Watt & Watt
1083     DATA_TYPE wx, wy, wz, xx, yy, yz, xy, xz, zz, xs, ys, zs;
1084
1085     xs = q[Xelt] + q[Xelt]; ys = q[Yelt] + q[Yelt]; zs = q[Zelt] + q[Zelt];
1086     xx = q[Xelt] * xs;      xy = q[Xelt] * ys;      xz = q[Xelt] * zs;
1087     yy = q[Yelt] * ys;      yz = q[Yelt] * zs;      zz = q[Zelt] * zs;
1088     wx = q[Welt] * xs;      wy = q[Welt] * ys;      wz = q[Welt] * zs;
1089
1090     mat( 0, 0 ) = DATA_TYPE(1.0) - (yy + zz);
1091     mat( 1, 0 ) = xy + wz;
1092     mat( 2, 0 ) = xz - wy;
1093
1094     mat( 0, 1 ) = xy - wz;
1095     mat( 1, 1 ) = DATA_TYPE(1.0) - (xx + zz);
1096     mat( 2, 1 ) = yz + wx;
1097
1098     mat( 0, 2 ) = xz + wy;
1099     mat( 1, 2 ) = yz - wx;
1100     mat( 2, 2 ) = DATA_TYPE(1.0) - (xx + yy);
1101
1102     return mat;
1103 }
```

**8.7.2.36** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS,  
typename ROT_ORDER> Matrix<DATA_TYPE, ROWS, COLS> &  
setRot (Matrix< DATA_TYPE, ROWS, COLS > &result, const  
EulerAngle< DATA_TYPE, ROT_ORDER > &euler) [inline]`

Set (only) the rotation part of a matrix using an [EulerAngle](#) (angles are in radians).

**Postcondition:**

this function only produces 3x3, 3x4, 4x3, and 4x4 matrices, and is undefined otherwise

**See also:**

[EulerAngle](#) for angle ordering (usually ordered based on RotationOrder)

Definition at line 797 of file Generate.h.

References gmtl::Math::cos(), gmtlASSERT, and gmtl::Math::sin().

```

798 {
799     // @todo set this a compile time assert...
800     gmtlASSERT( ROWS >= 3 && COLS >= 3 && ROWS <= 4 && COLS <= 4 && "this is undefined for Matrix size" );
801
802     // this might be faster if put into the switch statement... (testme)
803     const int& order = ROT_ORDER::ID;
804     const float xRot = (order == XYZ::ID) ? euler[0] : ((order == ZXY::ID) ? euler[1] : euler[2]);
805     const float yRot = (order == XYZ::ID) ? euler[1] : ((order == ZXY::ID) ? euler[2] : euler[0]);
806     const float zRot = (order == XYZ::ID) ? euler[2] : ((order == ZXY::ID) ? euler[0] : euler[1]);
807
808     float sx = Math::sin( xRot ); float cx = Math::cos( xRot );
809     float sy = Math::sin( yRot ); float cy = Math::cos( yRot );
810     float sz = Math::sin( zRot ); float cz = Math::cos( zRot );
811
812     // @todo metaprogram this!
813     switch (order)
814     {
815     case XYZ::ID:
816         // Derived by simply multiplying out the matrices by hand X * Y * Z
817         result( 0, 0 ) = cy*cz; result( 0, 1 ) = -cy*sz; result( 0, 2 ) = sy;
818         result( 1, 0 ) = sx*sy*cz + cx*sz; result( 1, 1 ) = -sx*sy*sz + cx*cz; result( 1, 2 ) = -sx*cy;
819         result( 2, 0 ) = -cx*sy*cz + sx*sz; result( 2, 1 ) = cx*sy*sz + sx*cz; result( 2, 2 ) = cx*cy;
820         break;
821     case ZYX::ID:
822         // Derived by simply multiplying out the matrices by hand Z * Y * X
823         result( 0, 0 ) = cy*cz; result( 0, 1 ) = -cx*sz + sx*sy*cz; result( 0, 2 ) = sx*sz + cx*sy*cz;
824         result( 1, 0 ) = cy*sz; result( 1, 1 ) = cx*cz + sx*sy*sz; result( 1, 2 ) = -sx*cz + cx*sy*sz;
825         result( 2, 0 ) = -sy; result( 2, 1 ) = sx*cy; result( 2, 2 ) = cx*cy;
826         break;
827     case ZXY::ID:
828         // Derived by simply multiplying out the matrices by hand Z * X * Y
829         result( 0, 0 ) = cy*cz - sx*sy*sz; result( 0, 1 ) = -cx*sz; result( 0, 2 ) = sy*cz + sx*cy*sz;
830         result( 1, 0 ) = cy*sz + sx*sy*cz; result( 1, 1 ) = cx*cz; result( 1, 2 ) = sy*sz - sx*cy*cz;
831         result( 2, 0 ) = -cx*sy; result( 2, 1 ) = sx; result( 2, 2 ) = cx*cy;
832         break;
833     default:
834         gmtlASSERT( false && "unknown rotation order passed to setRot" );
835         break;
836     }
837
838     return result;
839 }

```

**8.7.2.37** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS> & setRot ( Matrix<DATA_TYPE, ROWS, COLS > & result, const AxisAngle<DATA_TYPE > & axisAngle) [inline]`

Set the rotation portion of a rotation matrix using an axis and an angle (in radians).

Only writes to the rotation matrix (3x3) defined by the rotation part of M

**Postcondition:**

this function only writes to 3x3, 3x4, 4x3, and 4x4 matrices, and is undefined otherwise

**Precondition:**

you must pass a normalized vector in for the axis, results are undefined if not.

Definition at line 756 of file Generate.h.

References `gmtl::Math::cos()`, `gmtlASSERT`, `gmtl::lengthSquared()`, and `gmtl::Math::sin()`.

```

757     {
758         /* @todo set this a compile time assert... */
759         gmtlASSERT( ROWS >= 3 && COLS >= 3 && ROWS <= 4 && COLS <= 4 &&
760             "this func is undefined for Matrix smaller than 3x3 or bigger than 4x4" );
761         gmtlASSERT( Math::isEqual( lengthSquared( axisAngle.getAxis() ), (DATA_TYPE)1.0, (DATA_TYPE)0.000001 ) &&
762             "you must pass in a normalized vector to setRot( mat, rad, vec )" );
763
764         // GGI: pg 466
765         DATA_TYPE s = Math::sin( axisAngle.getAngle() );
766         DATA_TYPE c = Math::cos( axisAngle.getAngle() );
767         DATA_TYPE t = DATA_TYPE( 1.0 ) - c;
768         DATA_TYPE x = axisAngle.getAxis()[0];
769         DATA_TYPE y = axisAngle.getAxis()[1];
770         DATA_TYPE z = axisAngle.getAxis()[2];
771
772         /* From: Introduction to robotic. Craig. Pg. 52 */
773         result( 0, 0 ) = (t*x*x)+c;      result( 0, 1 ) = (t*x*y)-(s*z); result( 0, 2 ) = (t*x*z)+(s*y);
774         result( 1, 0 ) = (t*x*y)+(s*z); result( 1, 1 ) = (t*y*y)+c;      result( 1, 2 ) = (t*y*z)-(s*x);
775         result( 2, 0 ) = (t*x*z)-(s*y); result( 2, 1 ) = (t*y*z)+(s*x); result( 2, 2 ) = (t*z*z)+c;
776
777         return result;
778     }

```

**8.7.2.38** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, typename ROT_ORDER> EulerAngle<DATA_TYPE, ROT_ORDER> & setRot (EulerAngle< DATA_TYPE, ROT_ORDER > & result, const Matrix< DATA_TYPE, ROWS, COLS > & mat) [inline]`

Redundant duplication of the set(eulerangle,quat) function, this is provided only for template compatibility.

unless you're writing template functions, you should use set(eulerangle,quat) for clarity.

Definition at line 655 of file Generate.h.

References gmtl::set().

```
656 {
657     return set( result, mat );
658 }
```

**8.7.2.39** `template<typename DATA_TYPE> AxisAngle<DATA_TYPE> & setRot (AxisAngle< DATA_TYPE > & result, Quat< DATA_TYPE > quat) [inline]`

Redundant duplication of the set(axisangle,quat) function, this is provided only for template compatibility.

unless you're writing template functions, you should use set(axisangle,quat) for clarity.

Definition at line 542 of file Generate.h.

References gmtl::set().

```
543 {
544     return set( result, quat );
545 }
```

**8.7.2.40** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Quat<DATA_TYPE> & setRot (Quat< DATA_TYPE > & result, const Matrix< DATA_TYPE, ROWS, COLS > & mat) [inline]`

Redundant duplication of the set(quat,mat) function, this is provided only for template compatibility.

unless you're writing template functions, you should use `set(quat,mat)`.

Definition at line 476 of file `Generate.h`.

References `gmtl::set()`.

```
477     {
478         return set( result, mat );
479     }
```

**8.7.2.41** `template<typename DATA_TYPE, typename ROT_ORDER>`  
`Quat<DATA_TYPE>& setRot (Quat< DATA_TYPE > &result, const`  
`EulerAngle< DATA_TYPE, ROT_ORDER > &euler) [inline]`

Redundant duplication of the `set(quat,eulerangle)` function, this is provided only for template compatibility.

unless you're writing template functions, you should use `set(quat,eulerangle)`.

Definition at line 404 of file `Generate.h`.

References `gmtl::set()`.

```
405     {
406         return set( result, euler );
407     }
```

**8.7.2.42** `template<typename DATA_TYPE> Quat<DATA_TYPE>& setRot`  
`(Quat< DATA_TYPE > &result, const AxisAngle< DATA_TYPE > &`  
`axisAngle) [inline]`

Redundant duplication of the `set(quat,axisangle)` function, this is provided only for template compatibility.

unless you're writing template functions, you should use `set(quat,axisangle)`.

Definition at line 340 of file `Generate.h`.

References `gmtl::set()`.

```
341     {
342         return set( result, axisAngle );
343     }
```

**8.7.2.43** `template<typename DEST_TYPE, typename DATA_TYPE>`  
`DEST_TYPE& setRot( DEST_TYPE & result, const Vec<`  
`DATA_TYPE, 3 > & from, const Vec< DATA_TYPE, 3 > & to)`  
`[inline]`

set a rotation datatype that will xform first vector to the second.

**Precondition:**

each vec needs to be normalized.

**Postcondition:**

generate rotation datatype that is the rotation between the vectors. @note: only sets the rotation component of result, if result is a matrix, only sets the 3x3.

Definition at line 151 of file Generate.h.

References `gmtl::Math::aCos()`, `gmtl::cross()`, `gmtl::dot()`, `gmtl::ASSERT`, `gmtl::isEqual()`, and `gmtl::normalize()`.

Referenced by `gmtl::makeRot()`, and `gmtl::set()`.

```

152 {
153     // @todo should assert that DEST_TYPE::DataType == DATA_TYPE
154     const DATA_TYPE epsilon = (DATA_TYPE)0.00001;
155
156     gmtl::ASSERT( gmtl::Math::isEqual( gmtl::length( from ), (DATA_TYPE)1.0, epsilon ) &&
157                 gmtl::Math::isEqual( gmtl::length( to ), (DATA_TYPE)1.0, epsilon ) &&
158                 "input params not normalized" );
159
160     DATA_TYPE cosangle = dot( from, to );
161
162     // if cosangle is close to 1, so the vectors are close to being coincident
163     // Need to generate an angle of zero with any vector we like
164     // We'll choose identity (no rotation)
165     if ( Math::isEqual( cosangle, (DATA_TYPE)1.0, epsilon ) )
166     {
167         return result = DEST_TYPE();
168     }
169
170     // vectors are close to being opposite, so rotate one a little...
171     else if ( Math::isEqual( cosangle, (DATA_TYPE)-1.0, epsilon ) )
172     {
173         Vec<DATA_TYPE, 3> to_rot( to[0] + (DATA_TYPE)0.3, to[1] - (DATA_TYPE)0.15, to[2] - (DATA_TYPE)0.15 );
174         normalize( cross( axis, from, to_rot ) ); // setRot requires normalized vec
175         DATA_TYPE angle = Math::aCos( cosangle );
176         return setRot( result, gmtl::AxisAngle<DATA_TYPE>( angle, axis ) );
177     }
178
179     // This is the usual situation - take a cross-product of vec1 and vec2
180     // and that is the axis around which to rotate.
181     else
182     {

```

```

183         Vec<DATA_TYPE, 3> axis;
184         normalize( cross( axis, from, to ) ); // setRot requires normalized vec
185         DATA_TYPE angle = Math::aCos( cosangle );
186         return setRot( result, gmtl::AxisAngle<DATA_TYPE>( angle, axis ) );
187     }
188 }

```

**8.7.2.44** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& setScale( Matrix<DATA_TYPE, ROWS, COLS> & result, const DATA_TYPE scale)` [inline]

Create a scale matrix.

Definition at line 730 of file Generate.h.

References `gmtl::Math::Min()`.

```

731     {
732         for (unsigned x = 0; x < Math::Min( ROWS, COLS, Math::Max( ROWS, COLS ) - 1 ); ++x)
733             result( x, x ) = scale;
734         return result;
735     }

```

**8.7.2.45** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, unsigned SIZE> Matrix<DATA_TYPE, ROWS, COLS>& setScale( Matrix<DATA_TYPE, ROWS, COLS> & result, const Vec<DATA_TYPE, SIZE> & scale)` [inline]

Set the scale part of a matrix.

Definition at line 706 of file Generate.h.

References `gmtlASSERT`.

Referenced by `gmtl::makeScale()`.

```

707     {
708         gmtlASSERT( ((SIZE == (ROWS-1) && SIZE == (COLS-1)) || (SIZE == (ROWS-1) && SIZE == (COLS-1))) );
709         for (unsigned x = 0; x < SIZE; ++x)
710             result( x, x ) = scale[x];
711         return result;
712     }

```



**8.7.2.46** `template<typename VEC_TYPE, typename DATA_TYPE, unsigned ROWS, unsigned COLS> VEC_TYPE& setTrans (VEC_TYPE & result, const Matrix< DATA_TYPE, ROWS, COLS > & arg)`  
`[inline]`

Set vector using translation portion of the matrix.

**Precondition:**

if making an  $n \times n$  matrix, then for

- **vector is homogeneous:** SIZE of vector needs to equal number of [Matrix](#) ROWS - 1
- **vector has scale component:** SIZE of vector needs to equal number of [Matrix](#) ROWS if making an  $n \times n+1$  matrix, then for
- **vector is homogeneous:** SIZE of vector needs to equal number of [Matrix](#) ROWS
- **vector has scale component:** SIZE of vector needs to equal number of [Matrix](#) ROWS + 1

**Postcondition:**

if preconditions are not met, then function is undefined (will not compile)

Definition at line 226 of file Generate.h.

References gmtlASSERT.

Referenced by gmtl::makeTrans(), and gmtl::set().

```

227 {
228     // ASSERT: There are as many
229
230     // if n x n then (homogeneous case) vecsize == rows-1 or (scale component case) vecsize == rows
231     // if n x n+1 then (homogeneous case) vecsize == rows or (scale component case) vecsize == rows+1
232     gmtlASSERT( ((ROWS == COLS && ( VEC_TYPE::Size == (ROWS-1) || VEC_TYPE::Size == ROWS)) ||
233                 (COLS == (ROWS+1) && ( VEC_TYPE::Size == ROWS || VEC_TYPE::Size == (ROWS+1)))) &&
234                 "preconditions not met for vector size in call to makeTrans. Read your documentation."
235
236     // homogeneous case...
237     if ((ROWS == COLS && VEC_TYPE::Size == ROWS) // Square matrix and vec so assume homogeneous
238         || (COLS == (ROWS+1) && VEC_TYPE::Size == (ROWS+1))) // ex: 3x4 with vec4
239     {
240         result[VEC_TYPE::Size-1] = 1.0f;
241     }
242
243     // non-homogeneous case... (SIZE == ROWS),
244     //else
245     //{
246
247     for (unsigned x = 0; x < COLS - 1; ++x)
248     {
249         result[x] = arg( x, COLS - 1 );

```

```

250     }
251
252     return result;
253 }
```

**8.7.2.47** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, unsigned SIZE> Matrix<DATA_TYPE, ROWS, COLS>& setTrans(Matrix< DATA_TYPE, ROWS, COLS > &result, const Vec< DATA_TYPE, SIZE > &trans) [inline]`

Set matrix translation from vec.

**Precondition:**

if making an n x n matrix, then for

- **vector is homogeneous:** SIZE of vector needs to equal number of [Matrix](#) ROWS - 1
- **vector has scale component:** SIZE of vector needs to equal number of [Matrix](#) ROWS if making an n x n+1 matrix, then for
- **vector is homogeneous:** SIZE of vector needs to equal number of [Matrix](#) ROWS
- **vector has scale component:** SIZE of vector needs to equal number of [Matrix](#) ROWS + 1

**Postcondition:**

if preconditions are not met, then function is undefined (will not compile)

Definition at line 676 of file Generate.h.

References [gmtlASSERT](#).

```

678 {
679     /* @todo make this a compile time assert... */
680     // if n x n then (homogeneous case) vecsize == rows-1 or (scale component case) vecsize == rows
681     // if n x n+1 then (homogeneous case) vecsize == rows or (scale component case) vecsize == rows+1
682     gmtlASSERT( ((ROWS == COLS && (SIZE == (ROWS-1) || SIZE == ROWS)) ||
683                 (COLS == (ROWS+1) && (SIZE == ROWS || SIZE == (ROWS+1)))) &&
684                 "preconditions not met for vector size in call to makeTrans. Read your documentation" );
685
686     // homogeneous case...
687     if ((ROWS == COLS && SIZE == ROWS) /* Square matrix and vec so assume homogeneous vector */
688         || (COLS == (ROWS+1) && SIZE == (ROWS+1))) /* ex: 3x4 with vec4 */
689     {
690         for (unsigned x = 0; x < COLS - 1; ++x)
691             result( x, COLS - 1 ) = trans[x] / trans[SIZE-1];
692     }
693 }
```

---

```
694     // non-homogeneous case...
695     else
696     {
697         for (unsigned x = 0; x < COLS - 1; ++x)
698             result( x, COLS - 1 ) = trans[x];
699     }
700     return result;
701 }
```

## 8.8 Interpolation: lerp(...), slerp(...)

Functions to interpolate between two values.

### Scalar type interpolation (for doubles, floats, etc...)

- `template<class T, typename U> void lerp (T &result, const U &lerp, const T &a, const T &b)`

*Linear Interpolation between number [a] and [b].*

### Quaternion Interpolation

- `template<typename DATA_TYPE> Quat< DATA_TYPE > & slerp (Quat< DATA_TYPE > &result, const DATA_TYPE t, const Quat< DATA_TYPE > &from, const Quat< DATA_TYPE > &to)`

*spherical linear interpolation between two rotation quaternions.*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > & lerp (Quat< DATA_TYPE > &result, const DATA_TYPE t, const Quat< DATA_TYPE > &from, const Quat< DATA_TYPE > &to)`

*linear interpolation between two quaternions.*

### Vector Interpolation

- `template<typename DATA_TYPE, unsigned SIZE> VecBase< DATA_TYPE, SIZE > & lerp (VecBase< DATA_TYPE, SIZE > &result, const DATA_TYPE &lerpVal, const VecBase< DATA_TYPE, SIZE > &from, const VecBase< DATA_TYPE, SIZE > &to)`

*Linearly interpolates between to vectors.*

#### 8.8.1 Detailed Description

Functions to interpolate between two values.

## 8.8.2 Function Documentation

**8.8.2.1** `template<typename DATA_TYPE> Quat<DATA_TYPE>& lerp(  
Quat< DATA_TYPE > & result, const DATA_TYPE t, const Quat<  
DATA_TYPE > & from, const Quat< DATA_TYPE > & to)`

linear interpolation between two quaternions.

*t* is a value between 0 and 1 that interpolates between *from* and *to*.

**Precondition:**

no aliasing problems to worry about ("result" can be "from" or "to" param). References:

- From Adv Anim and Rendering Tech. Pg 364

**See also:**

[Quat](#)

Definition at line 571 of file QuatOps.h.

References `gmatl::dot()`, `gmatl::Welt`, `gmatl::Xelt`, `gmatl::Yelt`, and `gmatl::Zelt`.

```

572     {
573         // just an alias to match q
574         const Quat<DATA_TYPE>& p = from;
575
576         // calc cosine theta
577         DATA_TYPE cosom = dot( from, to );
578
579         // adjust signs (if necessary)
580         Quat<DATA_TYPE> q;
581         if (cosom < (DATA_TYPE)0.0)
582         {
583             q[0] = -to[0];    // Reverse all signs
584             q[1] = -to[1];
585             q[2] = -to[2];
586             q[3] = -to[3];
587         }
588         else
589         {
590             q = to;
591         }
592
593         // do linear interp
594         DATA_TYPE sclp, sclq;
595         sclp = (DATA_TYPE)1.0 - t;
596         sclq = t;
597     }

```

```

598     result[Xelt] = sclp * p[Xelt] + sclq * q[Xelt];
599     result[Yelt] = sclp * p[Yelt] + sclq * q[Yelt];
600     result[Zelt] = sclp * p[Zelt] + sclq * q[Zelt];
601     result[Welt] = sclp * p[Welt] + sclq * q[Welt];
602     return result;
603 }

```

**8.8.2.2** `template<typename DATA_TYPE, unsigned SIZE>`  
`VecBase<DATA_TYPE, SIZE> & lerp (VecBase< DATA_TYPE, SIZE`  
`> & result, const DATA_TYPE & lerpVal, const VecBase< DATA_TYPE,`  
`SIZE > & from, const VecBase< DATA_TYPE, SIZE > & to)`

Linearly interpolates between to vectors.

**Precondition:**

lerpVal is a value between 0 and 1 that interpolates between from and to.

**Postcondition:**

undefined if lerpVal < 0 or lerpVal > 1

**Parameters:**

*result* the result of the linear interpolation

*lerpVal* the value to interpolate between from and to

*from* the vector at lerpVal 0

*to* the vector at lerpVal 1

**Returns:**

a reference to result for convenience

Definition at line 418 of file VecOps.h.

```

422 {
424     for (unsigned int x = 0; x < SIZE; ++x)
425     {
426         Math::lerp( result[x], lerpVal, from[x], to[x] );
427     }
428     return result;
429 }

```

**8.8.2.3** `template<class T, typename U> void lerp (T & result, const U & lerp,  
const T & a, const T & b) [inline]`

Linear Interpolation between number [a] and [b].

**Precondition:**

use double or float only...

Definition at line 447 of file Math.h.

References gmtl::Math::lerp().

Referenced by gmtl::Math::lerp().

```
448 {
449     T size = b - a;
450     result = ((U)a) + (((U)size) * lerp);
451 }
```

**8.8.2.4** `template<typename DATA_TYPE> Quat<DATA_TYPE>& slerp  
(Quat< DATA_TYPE > & result, const DATA_TYPE t, const Quat<  
DATA_TYPE > & from, const Quat< DATA_TYPE > & to)`

spherical linear interpolation between two rotation quaternions.

t is a value between 0 and 1 that interpolates between from and to.

**Precondition:**

no aliasing problems to worry about ("result" can be "from" or "to" param). References:

- From Adv Anim and Rendering Tech. Pg 364

**See also:**

[Quat](#)

Definition at line 514 of file QuatOps.h.

References gmtl::Math::aCos(), gmtl::dot(), gmtl::Math::sin(), gmtl::Welt, gmtl::Xelt, gmtl::Yelt, and gmtl::Zelt.

```
515 {
516     const Quat<DATA_TYPE>& p = from; // just an alias to match q
517
518     // calc cosine theta
519     DATA_TYPE cosom = dot( from, to );
```

```

520
521     // adjust signs (if necessary)
522     Quat<DATA_TYPE> q;
523     if (cosom < (DATA_TYPE)0.0)
524     {
525         cosom = -cosom;
526         q[0] = -to[0];    // Reverse all signs
527         q[1] = -to[1];
528         q[2] = -to[2];
529         q[3] = -to[3];
530     }
531     else
532     {
533         q = to;
534     }
535
536     // Calculate coefficients
537     DATA_TYPE sclp, sclq;
538     if ((DATA_TYPE)1.0 - cosom) > (DATA_TYPE)0.0001) // 0.0001 -> some epsilon
539     {
540         // Standard case (slerp)
541         DATA_TYPE omega, sinom;
542         omega = gmtl::Math::aCos( cosom ); // extract theta from dot product's cos theta
543         sinom = gmtl::Math::sin( omega );
544         sclp = gmtl::Math::sin( ((DATA_TYPE)1.0 - t) * omega ) / sinom;
545         sclq = gmtl::Math::sin( t * omega ) / sinom;
546     }
547     else
548     {
549         // Very close, do linear interp (because it's faster)
550         sclp = (DATA_TYPE)1.0 - t;
551         sclq = t;
552     }
553
554     result[Xelt] = sclp * p[Xelt] + sclq * q[Xelt];
555     result[Yelt] = sclp * p[Yelt] + sclq * q[Yelt];
556     result[Zelt] = sclp * p[Zelt] + sclq * q[Zelt];
557     result[Welt] = sclp * p[Welt] + sclq * q[Welt];
558     return result;
559 }

```



## 8.9 Output Stream Methods: operator<<( ... ).

Output GMTL data types to an ostream.

### Output Stream Operators

- `template<class DATA_TYPE, unsigned SIZE> std::ostream & operator<< (std::ostream &out, const VecBase< DATA_TYPE, SIZE > &v)`

*Outputs a string representation of the given VecBase type to the given output stream.*

- `template<class DATA_TYPE, unsigned ROWS, unsigned COLS> std::ostream & operator<< (std::ostream &out, const Matrix< DATA_TYPE, ROWS, COLS > &m)`

*Outputs a string representation of the given Matrix to the given output stream.*

- `template<typename DATA_TYPE> std::ostream & operator<< (std::ostream &out, const Quat< DATA_TYPE > &q)`

*Outputs a string representation of the given Matrix to the given output stream.*

- `template<typename DATA_TYPE> std::ostream & operator<< (std::ostream &out, const Tri< DATA_TYPE > &t)`

*Outputs a string representation of the given Tri to the given output stream.*

- `template<typename DATA_TYPE> std::ostream & operator<< (std::ostream &out, const Plane< DATA_TYPE > &p)`

*Outputs a string representation of the given Plane to the given output stream.*

- `template<typename DATA_TYPE> std::ostream & operator<< (std::ostream &out, const Sphere< DATA_TYPE > &s)`

*Outputs a string representation of the given Sphere to the given output stream.*

### 8.9.1 Detailed Description

Output GMTL data types to an ostream.

std::ostream& operator<< methods...

## 8.9.2 Function Documentation

### 8.9.2.1 `template<typename DATA_TYPE> std::ostream& operator<< (std::ostream & out, const Sphere< DATA_TYPE > & s)`

Outputs a string representation of the given [Sphere](#) to the given output stream.

The output is formatted such that `Sphere<int>( Point<int, 3>(1,2,3), 4 )` will appear as "(1, 2, 3), 4)".

**Parameters:**

*out* the stream to write to

*s* the [Sphere](#) to output

**Returns:**

out after it has been written to

Definition at line 186 of file Output.h.

```
187     {
188         out << s.mCenter << ", " << s.mRadius;
189         return out;
190     }
```

### 8.9.2.2 `template<typename DATA_TYPE> std::ostream& operator<< (std::ostream & out, const Plane< DATA_TYPE > & p)`

Outputs a string representation of the given [Plane](#) to the given output stream.

The output is formatted such that `Plane<int>( Vec<int, 3>(1,2,3), 4 )` will appear as "(1, 2, 3), 4)".

**Parameters:**

*out* the stream to write to

*p* the [Plane](#) to output

**Returns:**

out after it has been written to

Definition at line 165 of file Output.h.

```

166     {
167         out << p.mNorm << ", " << p.mOffset;
168         return out;
169     }

```

### 8.9.2.3 `template<typename DATA_TYPE> std::ostream& operator<<` `(std::ostream & out, const Tri< DATA_TYPE > & t)`

Outputs a string representation of the given `Tri` to the given output stream.

The output is formatted such that `Tri<int>( Point<int, 3>(1,2,3), Point<int, 3>(4,5,6), Point<int, 3>(7,8,9) )` will appear as `"(1, 2, 3), (4, 5, 6), (7, 8, 9)"`.

#### Parameters:

*out* the stream to write to

*t* the `Tri` to output

#### Returns:

out after it has been written to

Definition at line 144 of file Output.h.

```

145     {
146         out << t[0] << ", " << t[1] << ", " << t[2];
147         return out;
148     }

```

### 8.9.2.4 `template<typename DATA_TYPE> std::ostream& operator<<` `(std::ostream & out, const Quat< DATA_TYPE > & q)`

Outputs a string representation of the given `Matrix` to the given output stream.

The output is formatted such that `Quat<int>(1,2,3,4)` will appear as `"(1, 2, 3, 4)"`.

#### Parameters:

*out* the stream to write to

*q* the `Quat` to output

#### Returns:

out after it has been written to

Definition at line 122 of file Output.h.

```

123     {
124         out << q.mData;
125         return out;
126     }

```

#### 8.9.2.5 `template<class DATA_TYPE, unsigned ROWS, unsigned COLS> std::ostream& operator<< (std::ostream & out, const Matrix< DATA_TYPE, ROWS, COLS > & m)`

Outputs a string representation of the given [Matrix](#) to the given output stream.

The output is formatted along the lines of:

```

| 1 2 3 4 |
| 5 6 7 8 |
| 9 10 11 12 |

```

#### Parameters:

*out* the stream to write to

*m* the [Matrix](#) to output

#### Returns:

out after it has been written to

Definition at line 96 of file Output.h.

```

98     {
99         for ( unsigned row=0; row<ROWS; ++row )
100         {
101             out << "|";
102             for ( unsigned col=0; col<COLS; ++col )
103             {
104                 out << " " << m(row, col);
105             }
106             out << " |" << std::endl;
107         }
108         return out;
109     }

```

**8.9.2.6** `template<class DATA_TYPE, unsigned SIZE> std::ostream&  
operator<< (std::ostream & out, const VecBase< DATA_TYPE, SIZE  
> & v)`

Outputs a string representation of the given `VecBase` type to the given output stream.

This works for both `Point` and `Vec` types. The output is formatted such that `Vec<int, 4>(1,2,3,4)` will appear as "(1, 2, 3, 4)".

**Parameters:**

*out* the stream to write to

*v* the `VecBase` type to output

**Returns:**

out after it has been written to

Definition at line 65 of file Output.h.

```
67     {  
68         out << "(";  
69         for ( unsigned i=0; i<SIZE; ++i )  
70             {  
71                 if ( i != 0 )  
72                     {  
73                         out << ", ";  
74                     }  
75                     out << v[i];  
76             }  
77         out << ")";  
78         return out;  
79     }
```

## 8.10 Template Metaprogramming Utilities

### Compounds

- struct [Type2Type](#)

*A lightweight identifier you can pass to overloaded functions to typefy them.*

## 8.11 Template Metaprogramming Utilities (Helpers)

### [NOHEADER]

- `template<class T> void ignore\_unused\_variable\_warning (const T &)`

#### 8.11.1 Function Documentation

##### 8.11.1.1 `template<class T> void ignore_unused_variable_warning (const T &)` [inline]

Definition at line 58 of file Meta.h.

Referenced by `gmtl::make()`, `gmtl::makeAxes()`, `gmtl::makeDirCos()`, `gmtl::makeRot()`, `gmtl::makeScale()`, and `gmtl::makeTrans()`.

```
58 { }
```





---

## Chapter 9

# GenericMathTemplateLibrary Namespace Documentation

### 9.1 gmtl Namespace Reference

#### Compounds

- class [AABox](#)  
*Describes an axially aligned box in 3D space.*
  - class [AxisAngle](#)  
*[AxisAngle](#): Represents a "twist about an axis" [AxisAngle](#) is used to specify a rotation in 3-space.*
  - struct [CompareIndexPointProjections](#)
  - class [Coord](#)  
*coord is a position/rotation pair.*
  - class [Eigen](#)
  - class [EulerAngle](#)  
*[EulerAngle](#): Represents a group of euler angles.*
  - class [LineSeg](#)  
*Describes a line segment.*
-

- class [Matrix](#)  
*Matrix: 4x4 [Matrix](#) class (OpenGL ordering).*
- class [OOBox](#)
- class [Plane](#)  
*Plane: Defines a geometrical plane.*
- class [Point](#)  
*Point Use points when you need to represent a position.*
- class [Quat](#)  
*Quat: Class to encapsulate quaternion behaviors.*
- struct [RotationOrderBase](#)  
*Base class for Rotation orders.*
- class [Sphere](#)  
*Describes a sphere in 3D space by its center point and its radius.*
- class [Tri](#)  
*This class defines a triangle as a set of 3 points order in CCW fashion.*
- struct [Type2Type](#)  
*A lightweight identifier you can pass to overloaded functions to typefy them.*
- class [Vec](#)  
*A representation of a vector with SIZE components using DATA\_TYPE as the data type for each component.*
- class [VecBase](#)  
*Base type for vector-like objects including Points and Vectors.*
- struct [XYZ](#)  
*XYZ Rotation order.*
- struct [ZXY](#)  
*ZXY Rotation order.*
- struct [ZYX](#)  
*ZYX Rotation order.*

## AxisAngle Comparitors

- `template<class DATA_TYPE> bool operator== (const AxisAngle< DATA_TYPE > &v1, const AxisAngle< DATA_TYPE > &v2)`  
*Compares v1 and v2 to see if they are exactly the same with zero tolerance.*
- `template<class DATA_TYPE> bool operator!= (const AxisAngle< DATA_TYPE > &v1, const AxisAngle< DATA_TYPE > &v2)`  
*Compares v1 and v2 to see if they are NOT exactly the same with zero tolerance.*
- `template<class DATA_TYPE> bool isEqual (const AxisAngle< DATA_TYPE > &v1, const AxisAngle< DATA_TYPE > &v2, const DATA_TYPE &eps=(DATA_TYPE) 0)`  
*Compares v1 and v2 to see if they are the same within the given epsilon tolerance.*

## Coord Comparitors

- `template<typename POS_TYPE, typename ROT_TYPE> bool operator== (const Coord< POS_TYPE, ROT_TYPE > &q1, const Coord< POS_TYPE, ROT_TYPE > &q2)`  
*Compare two quaternions for equality.*
- `template<typename POS_TYPE, typename ROT_TYPE> bool operator!= (const Coord< POS_TYPE, ROT_TYPE > &q1, const Coord< POS_TYPE, ROT_TYPE > &q2)`  
*Compare two quaternions for not-equality.*
- `template<typename POS_TYPE, typename ROT_TYPE> bool isEqual (const Coord< POS_TYPE, ROT_TYPE > &q1, const Coord< POS_TYPE, ROT_TYPE > &q2, typename Coord< POS_TYPE, ROT_TYPE >::DataType tol=(typename Coord< POS_TYPE, ROT_TYPE >::DataType) 0.0)`  
*Compare two quaternions for equality with tolerance.*

## EulerAngle Comparitors

- `template<class DATA_TYPE, typename ROT_ORDER> bool operator== (const EulerAngle< DATA_TYPE, ROT_ORDER > &v1, const EulerAngle< DATA_TYPE, ROT_ORDER > &v2)`  
*Compares v1 and v2 to see if they are exactly the same with zero tolerance.*

- `template<class DATA_TYPE, typename ROT_ORDER> bool operator!= (const EulerAngle< DATA_TYPE, ROT_ORDER > &v1, const EulerAngle< DATA_TYPE, ROT_ORDER > &v2)`

*Compares v1 and v2 to see if they are NOT exactly the same with zero tolerance.*

- `template<class DATA_TYPE, typename ROT_ORDER> bool isEqual (const EulerAngle< DATA_TYPE, ROT_ORDER > &v1, const EulerAngle< DATA_TYPE, ROT_ORDER > &v2, const DATA_TYPE &eps=(DATA_TYPE) 0)`

*Compares v1 and v2 to see if they are the same within the given epsilon tolerance.*

## Generic Generators (any type)

- `template<typename TARGET_TYPE, typename SOURCE_TYPE> TARGET_TYPE make (const SOURCE_TYPE &src, Type2Type< TARGET_TYPE > t=Type2Type< TARGET_TYPE >())`

*Construct an object from another object of a different type.*

- `template<typename ROTATION_TYPE, typename SOURCE_TYPE> ROTATION_TYPE makeRot (const SOURCE_TYPE &coord, Type2Type< ROTATION_TYPE > t=Type2Type< ROTATION_TYPE >())`

*Create a rotation datatype from another rotation datatype.*

- `template<typename ROTATION_TYPE> ROTATION_TYPE makeDirCos (const Vec< typename ROTATION_TYPE::DataType, 3 > &xDestAxis, const Vec< typename ROTATION_TYPE::DataType, 3 > &yDestAxis, const Vec< typename ROTATION_TYPE::DataType, 3 > &zDestAxis, const Vec< typename ROTATION_TYPE::DataType, 3 > &xSrcAxis=Vec< typename ROTATION_TYPE::DataType, 3 >(1, 0, 0), const Vec< typename ROTATION_TYPE::DataType, 3 > &ySrcAxis=Vec< typename ROTATION_TYPE::DataType, 3 >(0, 1, 0), const Vec< typename ROTATION_TYPE::DataType, 3 > &zSrcAxis=Vec< typename ROTATION_TYPE::DataType, 3 >(0, 0, 1), Type2Type< ROTATION_TYPE > t=Type2Type< ROTATION_TYPE >())`

*Create a rotation matrix or quaternion (or any other rotation data type) using direction cosines.*

- `template<typename TRANS_TYPE, typename SRC_TYPE> TRANS_TYPE makeTrans (const SRC_TYPE &arg, Type2Type< TRANS_TYPE > t=Type2Type< TRANS_TYPE >())`

*Make a translation datatype from another translation datatype.*

- `template<typename ROTATION_TYPE> ROTATION_TYPE makeRot (const Vec< typename ROTATION_TYPE::DataType, 3 > &from, const Vec< typename ROTATION_TYPE::DataType, 3 > &to)`

*Create a rotation datatype that will xform first vector to the second.*

- `template<typename DEST_TYPE, typename DATA_TYPE> DEST_TYPE & setRot (DEST_TYPE &result, const Vec< DATA_TYPE, 3 > &from, const Vec< DATA_TYPE, 3 > &to)`

*set a rotation datatype that will xform first vector to the second.*

## Vec Generators

- `template<typename DATA_TYPE> Vec< DATA_TYPE, 3 > makeVec (const Quat< DATA_TYPE > &quat)`

*create a vector from the vector component of a quaternion.*

- `template<typename DATA_TYPE, unsigned SIZE> Vec< DATA_TYPE, SIZE > makeNormal (Vec< DATA_TYPE, SIZE > vec)`

*create a normalized vector from the given vector.*

- `template<typename VEC_TYPE, typename DATA_TYPE, unsigned ROWS, unsigned COLS> VEC_TYPE & setTrans (VEC_TYPE &result, const Matrix< DATA_TYPE, ROWS, COLS > &arg)`

*Set vector using translation portion of the matrix.*

## Quat Generators

- `template<typename DATA_TYPE> Quat< DATA_TYPE > & setPure (Quat< DATA_TYPE > &quat, const Vec< DATA_TYPE, 3 > &vec)`

*Set pure quaternion.*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > makePure (const Vec< DATA_TYPE, 3 > &vec)`

*create a pure quaternion.*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > makeNormal (const Quat< DATA_TYPE > &quat)`

*create a pure quaternion.*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > makeConj (const Quat< DATA_TYPE > &quat)`

*quaternion complex conjugate.*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > makeInvert (const Quat< DATA_TYPE > &quat)`  
*create quaternion from the inverse of another quaternion.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & set (Quat< DATA_TYPE > &result, const AxisAngle< DATA_TYPE > &axisAngle)`  
*Convert an [AxisAngle](#) to a [Quat](#).*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & setRot (Quat< DATA_TYPE > &result, const AxisAngle< DATA_TYPE > &axisAngle)`  
*Redundant duplication of the `set(quat,axisangle)` function, this is provided only for template compatibility.*
- `template<typename DATA_TYPE, typename ROT_ORDER> Quat< DATA_TYPE > & set (Quat< DATA_TYPE > &result, const EulerAngle< DATA_TYPE, ROT_ORDER > &euler)`  
*Convert an [EulerAngle](#) rotation to a [Quaternion](#) rotation.*
- `template<typename DATA_TYPE, typename ROT_ORDER> Quat< DATA_TYPE > & setRot (Quat< DATA_TYPE > &result, const EulerAngle< DATA_TYPE, ROT_ORDER > &euler)`  
*Redundant duplication of the `set(quat,eulerangle)` function, this is provided only for template compatibility.*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Quat< DATA_TYPE > & set (Quat< DATA_TYPE > &quat, const Matrix< DATA_TYPE, ROWS, COLS > &mat)`  
*Convert a [Matrix](#) to a [Quat](#).*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Quat< DATA_TYPE > & setRot (Quat< DATA_TYPE > &result, const Matrix< DATA_TYPE, ROWS, COLS > &mat)`  
*Redundant duplication of the `set(quat,mat)` function, this is provided only for template compatibility.*

## AxisAngle Generators

- `template<typename DATA_TYPE> AxisAngle< DATA_TYPE > & set (AxisAngle< DATA_TYPE > &axisAngle, Quat< DATA_TYPE > quat)`  
*Convert a rotation quaternion to an [AxisAngle](#).*
- `template<typename DATA_TYPE> AxisAngle< DATA_TYPE > & setRot (AxisAngle< DATA_TYPE > &result, Quat< DATA_TYPE > quat)`

*Redundant duplication of the set(axisangle,quat) function, this is provided only for template compatibility.*

- `template<typename DATA_TYPE> AxisAngle< DATA_TYPE > makeNormal(const AxisAngle< DATA_TYPE > &a)`  
*make a normalized axisangle.*

## EulerAngle Generators

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, type-name ROT_ORDER> EulerAngle< DATA_TYPE, ROT_ORDER > & set(EulerAngle< DATA_TYPE, ROT_ORDER > &euler, const Matrix< DATA_TYPE, ROWS, COLS > &mat)`  
*Convert [Matrix](#) to [EulerAngle](#).*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, type-name ROT_ORDER> EulerAngle< DATA_TYPE, ROT_ORDER > & setRot(EulerAngle< DATA_TYPE, ROT_ORDER > &result, const Matrix< DATA_TYPE, ROWS, COLS > &mat)`  
*Redundant duplication of the set(eulerangle,quat) function, this is provided only for template compatibility.*

## Matrix Generators

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, unsigned SIZE> Matrix< DATA_TYPE, ROWS, COLS > & setTrans(Matrix< DATA_TYPE, ROWS, COLS > &result, const Vec< DATA_TYPE, SIZE > &trans)`  
*Set matrix translation from vec.*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, unsigned SIZE> Matrix< DATA_TYPE, ROWS, COLS > & setScale(Matrix< DATA_TYPE, ROWS, COLS > &result, const Vec< DATA_TYPE, SIZE > &scale)`  
*Set the scale part of a matrix.*
- `template<typename MATRIX_TYPE, unsigned SIZE> MATRIX_TYPE makeScale(const Vec< typename MATRIX_TYPE::DataType, SIZE > &scale, Type2Type< MATRIX_TYPE > t=Type2Type< MATRIX_TYPE >())`  
*Create a scale matrix.*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & setScale(Matrix< DATA_TYPE, ROWS, COLS > &result, const DATA_TYPE scale)`

*Create a scale matrix.*

- `template<typename MATRIX_TYPE> MATRIX_TYPE makeScale (const typename MATRIX_TYPE::DataType scale, Type2Type< MATRIX_TYPE > t=Type2Type< MATRIX_TYPE >())`

*Create a scale matrix.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & setRot (Matrix< DATA_TYPE, ROWS, COLS > &result, const AxisAngle< DATA_TYPE > &axisAngle)`

*Set the rotation portion of a rotation matrix using an axis and an angle (in radians).*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & set (Matrix< DATA_TYPE, ROWS, COLS > &result, const AxisAngle< DATA_TYPE > &axisAngle)`

*Convert an [AxisAngle](#) to a rotation matrix.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, typename ROT_ORDER> Matrix< DATA_TYPE, ROWS, COLS > & setRot (Matrix< DATA_TYPE, ROWS, COLS > &result, const EulerAngle< DATA_TYPE, ROT_ORDER > &euler)`

*Set (only) the rotation part of a matrix using an [EulerAngle](#) (angles are in radians).*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, typename ROT_ORDER> Matrix< DATA_TYPE, ROWS, COLS > & set (Matrix< DATA_TYPE, ROWS, COLS > &result, const EulerAngle< DATA_TYPE, ROT_ORDER > &euler)`

*Convert an [EulerAngle](#) to a rotation matrix.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> float makeYRot (const Matrix< DATA_TYPE, ROWS, COLS > &mat)`

*Extracts the yaw information from the matrix.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> float makeXRot (const Matrix< DATA_TYPE, ROWS, COLS > &mat)`

*Extracts the pitch information from the matrix.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> float makeZRot (const Matrix< DATA_TYPE, ROWS, COLS > &mat)`

*Extracts the roll information from the matrix.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & setDirCos (Matrix< DATA_TYPE, ROWS, COLS > &result, const Vec< DATA_TYPE, 3 > &xDestAxis, const`



`Vec< DATA_TYPE, 3 > &yDestAxis, const Vec< DATA_TYPE, 3 > &zDestAxis, const Vec< DATA_TYPE, 3 > &xSrcAxis=Vec< DATA_TYPE, 3 >(1, 0, 0), const Vec< DATA_TYPE, 3 > &ySrcAxis=Vec< DATA_TYPE, 3 >(0, 1, 0), const Vec< DATA_TYPE, 3 > &zSrcAxis=Vec< DATA_TYPE, 3 >(0, 0, 1))`

*create a rotation matrix that will rotate from SrcAxis to DestAxis.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & setAxes (Matrix< DATA_TYPE, ROWS, COLS > &result, const Vec< DATA_TYPE, 3 > &xAxis, const Vec< DATA_TYPE, 3 > &yAxis, const Vec< DATA_TYPE, 3 > &zAxis)`

*set the matrix given the raw coordinate axes.*

- `template<typename ROTATION_TYPE> ROTATION_TYPE makeAxes (const Vec< typename ROTATION_TYPE::DataType, 3 > &xAxis, const Vec< typename ROTATION_TYPE::DataType, 3 > &yAxis, const Vec< typename ROTATION_TYPE::DataType, 3 > &zAxis, Type2Type< ROTATION_TYPE > t=Type2Type< ROTATION_TYPE >())`

*set the matrix given the raw coordinate axes.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > makeTranspose (const Matrix< DATA_TYPE, ROWS, COLS > &m)`

*create a matrix transposed from the source.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > makeInverse (const Matrix< DATA_TYPE, ROWS, COLS > src, Type2Type< Matrix< DATA_TYPE, ROWS, COLS > > t=Type2Type< Matrix< DATA_TYPE, ROWS, COLS > >())`

*Creates a matrix that is the inverse of the given source matrix.*

- `template<typename DATATYPE, typename POS_TYPE, typename ROT_TYPE, unsigned MATCOLS, unsigned MATROWS> Matrix< DATATYPE, MATROWS, MATCOLS > & set (Matrix< DATATYPE, MATROWS, MATCOLS > &mat, const Coord< POS_TYPE, ROT_TYPE > &coord)`

*Convert a Coord to a Matrix.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & setRot (Matrix< DATA_TYPE, ROWS, COLS > &mat, const Quat< DATA_TYPE > &q)`

*Set the rotation portion of a matrix (3x3) from a rotation quaternion.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & set (Matrix< DATA_TYPE, ROWS, COLS > &mat, const Quat< DATA_TYPE > &q)`

Convert a *Quat* to a rotation *Matrix*.

## Coord Generators

- `template<typename DATATYPE, typename POS_TYPE, typename ROT_TYPE, unsigned MATCOLS, unsigned MATROWS> Coord< POS_TYPE, ROT_TYPE > & set (Coord< POS_TYPE, ROT_TYPE > &eulercoord, const Matrix< DATATYPE, MATROWS, MATCOLS > &mat)`

*convert Matrix to Coord.*

- `template<typename DATATYPE, typename POS_TYPE, typename ROT_TYPE, unsigned MATCOLS, unsigned MATROWS> Coord< POS_TYPE, ROT_TYPE > & setRot (Coord< POS_TYPE, ROT_TYPE > &result, const Matrix< DATATYPE, MATROWS, MATCOLS > &mat)`

*Redundant duplication of the set(coord,mat) function, this is provided only for template compatibility.*

## Matrix Operations

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & identity (Matrix< DATA_TYPE, ROWS, COLS > &result)`

*Make identity matrix out the matrix.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & zero (Matrix< DATA_TYPE, ROWS, COLS > &result)`

*zero out the matrix.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned INTERNAL, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & mult (Matrix< DATA_TYPE, ROWS, COLS > &result, const Matrix< DATA_TYPE, ROWS, INTERNAL > &lhs, const Matrix< DATA_TYPE, INTERNAL, COLS > &rhs)`

*matrix multiply.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned INTERNAL, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > operator * (const Matrix< DATA_TYPE, ROWS, INTERNAL > &lhs, const Matrix< DATA_TYPE, INTERNAL, COLS > &rhs)`

*matrix \* matrix.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & sub (Matrix< DATA_TYPE, ROWS, COLS > &result, const Matrix< DATA_TYPE, ROWS, COLS > &lhs, const Matrix< DATA_TYPE, ROWS, COLS > &rhs)`  
*matrix subtraction (algebraic operation for matrix).*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & add (Matrix< DATA_TYPE, ROWS, COLS > &result, const Matrix< DATA_TYPE, ROWS, COLS > &lhs, const Matrix< DATA_TYPE, ROWS, COLS > &rhs)`  
*matrix addition (algebraic operation for matrix).*
- `template<typename DATA_TYPE, unsigned SIZE> Matrix< DATA_TYPE, SIZE, SIZE > & postMult (Matrix< DATA_TYPE, SIZE, SIZE > &result, const Matrix< DATA_TYPE, SIZE, SIZE > &operand)`  
*matrix postmultiply.*
- `template<typename DATA_TYPE, unsigned SIZE> Matrix< DATA_TYPE, SIZE, SIZE > & preMult (Matrix< DATA_TYPE, SIZE, SIZE > &result, const Matrix< DATA_TYPE, SIZE, SIZE > &operand)`  
*matrix preMultiply.*
- `template<typename DATA_TYPE, unsigned SIZE> Matrix< DATA_TYPE, SIZE, SIZE > & operator *= (Matrix< DATA_TYPE, SIZE, SIZE > &result, const Matrix< DATA_TYPE, SIZE, SIZE > &operand)`  
*matrix postmult (operator \* =).*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & mult (Matrix< DATA_TYPE, ROWS, COLS > &result, const Matrix< DATA_TYPE, ROWS, COLS > &mat, float scalar)`  
*matrix scalar mult.*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & mult (Matrix< DATA_TYPE, ROWS, COLS > &result, DATA_TYPE scalar)`  
*matrix scalar mult.*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & operator *= (Matrix< DATA_TYPE, ROWS, COLS > &result, DATA_TYPE scalar)`  
*matrix scalar mult (operator \* =).*

- `template<typename DATA_TYPE, unsigned SIZE> Matrix< DATA_TYPE, SIZE, SIZE > & transpose (Matrix< DATA_TYPE, SIZE, SIZE > &result)`  
*matrix transpose in place.*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & transpose (Matrix< DATA_TYPE, ROWS, COLS > &result, const Matrix< DATA_TYPE, COLS, ROWS > &source)`  
*matrix transpose from one type to another (i.e.*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & invertFull (Matrix< DATA_TYPE, ROWS, COLS > &result, const Matrix< DATA_TYPE, ROWS, COLS > &src)`  
*full matrix inversion.*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & invert (Matrix< DATA_TYPE, ROWS, COLS > &result, const Matrix< DATA_TYPE, ROWS, COLS > &src)`  
*smart matrix inversion.*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix< DATA_TYPE, ROWS, COLS > & invert (Matrix< DATA_TYPE, ROWS, COLS > &result)`  
*smart matrix inversion (in place) Does matrix inversion by intelligently selecting what type of inversion to use depending on the types of operations your [Matrix](#) has been through.*

## Matrix Comparitors

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> bool operator== (const Matrix< DATA_TYPE, ROWS, COLS > &lhs, const Matrix< DATA_TYPE, ROWS, COLS > &rhs)`  
*Compare two mats.*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> bool operator!= (const Matrix< DATA_TYPE, ROWS, COLS > &lhs, const Matrix< DATA_TYPE, ROWS, COLS > &rhs)`
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> bool isEqual (const Matrix< DATA_TYPE, ROWS, COLS > &lhs, const Matrix< DATA_TYPE, ROWS, COLS > &rhs, const DATA_TYPE &eps=(DATA_TYPE) 0)`

*Compare two vectors with a tolerance.*

## [NOHEADER]

- `template<class T> void ignore_unused_variable_warning (const T &)`

## Output Stream Operators

- `template<class DATA_TYPE, unsigned SIZE> std::ostream & operator<< (std::ostream &out, const VecBase< DATA_TYPE, SIZE > &v)`

*Outputs a string representation of the given VecBase type to the given output stream.*

- `template<class DATA_TYPE, unsigned ROWS, unsigned COLS> std::ostream & operator<< (std::ostream &out, const Matrix< DATA_TYPE, ROWS, COLS > &m)`

*Outputs a string representation of the given Matrix to the given output stream.*

- `template<typename DATA_TYPE> std::ostream & operator<< (std::ostream &out, const Quat< DATA_TYPE > &q)`

*Outputs a string representation of the given Matrix to the given output stream.*

- `template<typename DATA_TYPE> std::ostream & operator<< (std::ostream &out, const Tri< DATA_TYPE > &t)`

*Outputs a string representation of the given Tri to the given output stream.*

- `template<typename DATA_TYPE> std::ostream & operator<< (std::ostream &out, const Plane< DATA_TYPE > &p)`

*Outputs a string representation of the given Plane to the given output stream.*

- `template<typename DATA_TYPE> std::ostream & operator<< (std::ostream &out, const Sphere< DATA_TYPE > &s)`

*Outputs a string representation of the given Sphere to the given output stream.*

## Plane Operations

- `template<class DATA_TYPE> DATA_TYPE distance (const Plane< DATA_TYPE > &plane, const Point< DATA_TYPE, 3 > &pt)`

*Computes the distance from the plane to the point.*

- `template<class DATA_TYPE> PlaneSide whichSide (const Plane< DATA_TYPE > &plane, const Point< DATA_TYPE, 3 > &pt)`

*Determines which side of the plane the given point lies.*

- `template<class DATA_TYPE> PlaneSide whichSide (const Plane< DATA_TYPE > &plane, const Point< DATA_TYPE, 3 > &pt, const DATA_TYPE &eps)`

*Determines which side of the plane the given point lies with the given epsilon tolerance.*

- `template<class DATA_TYPE> DATA_TYPE findNearestPt (const Plane< DATA_TYPE > &plane, const Point< DATA_TYPE, 3 > &pt, Point< DATA_TYPE, 3 > &result)`

*Finds the point on the plane that is nearest to the given point.*

## Plane Comparitors

- `template<class DATA_TYPE> bool operator== (const Plane< DATA_TYPE > &p1, const Plane< DATA_TYPE > &p2)`

*Compare two planes to see if they are EXACTLY the same.*

- `template<class DATA_TYPE> bool operator!= (const Plane< DATA_TYPE > &p1, const Plane< DATA_TYPE > &p2)`

*Compare two planes to see if they are not EXACTLY the same.*

- `template<class DATA_TYPE> bool isEqual (const Plane< DATA_TYPE > &p1, const Plane< DATA_TYPE > &p2, const DATA_TYPE &eps)`

*Compare two planes to see if they are the same within the given tolerance.*

## Quat Operations

- `template<typename DATA_TYPE> Quat< DATA_TYPE > & mult (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`

*product of two quaternions (quaternion product) multiplication of quats is much like multiplication of typical complex numbers.*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > operator * (const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`

*product of two quaternions (quaternion product).*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > & operator *= (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q2)`  
*quaternion postmult.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & negate (Quat< DATA_TYPE > &result)`  
*Vector negation - negate each element in the quaternion vector.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > operator- (const Quat< DATA_TYPE > &quat)`  
*Vector negation - (operator-) return a temporary that is the negative of the given quat.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & mult (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q, DATA_TYPE s)`  
*vector scalar multiplication.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > operator * (const Quat< DATA_TYPE > &q, DATA_TYPE s)`  
*vector scalar multiplication.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & operator *= (Quat< DATA_TYPE > &q, DATA_TYPE s)`  
*vector scalar multiplication.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & div (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`  
*quotient of two quaternions.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & div (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q, DATA_TYPE s)`  
*quaternion vector scale.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > operator/ (const Quat< DATA_TYPE > &q, DATA_TYPE s)`  
*vector scalar division.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & operator/= (const Quat< DATA_TYPE > &q, DATA_TYPE s)`  
*vector scalar division.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & add (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`

*vector addition.*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > operator+ (const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`

*vector addition.*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > & operator+= (Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`

*vector addition.*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > & sub (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`

*vector subtraction.*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > operator- (const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`

*vector subtraction.*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > & operator-= (Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`

*vector subtraction.*

- `template<typename DATA_TYPE> DATA_TYPE dot (const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`

*vector dot product between two quaternions.*

- `template<typename DATA_TYPE> DATA_TYPE lengthSquared (const Quat< DATA_TYPE > &q)`

*quaternion "norm" (also known as vector length squared) using this can be faster than using length for some operations...*

- `template<typename DATA_TYPE> DATA_TYPE length (const Quat< DATA_TYPE > &q)`

*quaternion "absolute" (also known as vector length or magnitude) using this can be faster than using length for some operations...*

- `template<typename DATA_TYPE> Quat< DATA_TYPE > & normalize (Quat< DATA_TYPE > &result)`

*set self to the normalized quaternion of self.*

- `template<typename DATA_TYPE> bool isNormalized (const Quat< DATA_TYPE > &q1, const DATA_TYPE eps=(DATA_TYPE) 0.0001f)`

*Determines if the given vector is normalized within the given tolerance.*



- `template<typename DATA_TYPE> Quat< DATA_TYPE > & conj (Quat< DATA_TYPE > &result)`  
*quaternion complex conjugate.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & invert (Quat< DATA_TYPE > &result)`  
*quaternion multiplicative inverse.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & exp (Quat< DATA_TYPE > &result)`  
*complex exponentiation.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & log (Quat< DATA_TYPE > &result)`  
*complex logarithm.*
- `template<typename DATA_TYPE> void squad (Quat< DATA_TYPE > &result, DATA_TYPE t, const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2, const Quat< DATA_TYPE > &a, const Quat< DATA_TYPE > &b)`  
*WARNING: not implemented (do not use).*
- `template<typename DATA_TYPE> void meanTangent (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2, const Quat< DATA_TYPE > &q3)`  
*WARNING: not implemented (do not use).*

## Quaternion Interpolation

- `template<typename DATA_TYPE> Quat< DATA_TYPE > & slerp (Quat< DATA_TYPE > &result, const DATA_TYPE t, const Quat< DATA_TYPE > &from, const Quat< DATA_TYPE > &to)`  
*spherical linear interpolation between two rotation quaternions.*
- `template<typename DATA_TYPE> Quat< DATA_TYPE > & lerp (Quat< DATA_TYPE > &result, const DATA_TYPE t, const Quat< DATA_TYPE > &from, const Quat< DATA_TYPE > &to)`  
*linear interpolation between two quaternions.*

## Quat Comparisons

- `template<typename DATA_TYPE> bool operator== (const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`  
*Compare two quaternions for equality.*
- `template<typename DATA_TYPE> bool operator!= (const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)`  
*Compare two quaternions for not-equality.*
- `template<typename DATA_TYPE> bool isEqual (const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2, DATA_TYPE tol=0.0)`  
*Compare two quaternions for equality with tolerance.*
- `template<typename DATA_TYPE> bool isEquiv (const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2, DATA_TYPE tol=0.0)`  
*Compare two quaternions for geometric equivalence (with tolerance).*

## Sphere Comparitors

- `template<class DATA_TYPE> bool operator== (const Sphere< DATA_TYPE > &s1, const Sphere< DATA_TYPE > &s2)`  
*Compare two spheres to see if they are EXACTLY the same.*
- `template<class DATA_TYPE> bool operator!= (const Sphere< DATA_TYPE > &s1, const Sphere< DATA_TYPE > &s2)`  
*Compare two spheres to see if they are not EXACTLY the same.*
- `template<class DATA_TYPE> bool isEqual (const Sphere< DATA_TYPE > &s1, const Sphere< DATA_TYPE > &s2, const DATA_TYPE &eps)`  
*Compare two spheres to see if they are the same within the given tolerance.*

## Triangle Operations

- `template<class DATA_TYPE> Point< DATA_TYPE, 3 > center (const Tri< DATA_TYPE > &tri)`  
*Computes the point at the center of the given triangle.*
- `template<class DATA_TYPE> Vec< DATA_TYPE, 3 > normal (const Tri< DATA_TYPE > &tri)`  
*Computes the normal for this triangle.*

## Triangle Comparitors

- `template<class DATA_TYPE> bool operator== (const Tri< DATA_TYPE > &tri1, const Tri< DATA_TYPE > &tri2)`  
*Compare two triangles to see if they are EXACTLY the same.*
- `template<class DATA_TYPE> bool operator!= (const Tri< DATA_TYPE > &tri1, const Tri< DATA_TYPE > &tri2)`  
*Compare two triangle to see if they are not EXACTLY the same.*
- `template<class DATA_TYPE> bool isEqual (const Tri< DATA_TYPE > &tri1, const Tri< DATA_TYPE > &tri2, const DATA_TYPE &eps)`  
*Compare two triangles to see if they are the same within the given tolerance.*

## Vector/Point Operations

- `template<typename DATA_TYPE, unsigned SIZE> Vec< DATA_TYPE, SIZE > operator- (const VecBase< DATA_TYPE, SIZE > &v1)`  
*Negates v1.*
- `template<class DATA_TYPE, unsigned SIZE> VecBase< DATA_TYPE, SIZE > & operator+= (VecBase< DATA_TYPE, SIZE > &v1, const VecBase< DATA_TYPE, SIZE > &v2)`  
*Adds v2 to v1 and stores the result in v1.*
- `template<class DATA_TYPE, unsigned SIZE> VecBase< DATA_TYPE, SIZE > operator+ (const VecBase< DATA_TYPE, SIZE > &v1, const VecBase< DATA_TYPE, SIZE > &v2)`  
*Adds v2 to v1 and returns the result.*
- `template<class DATA_TYPE, unsigned SIZE> VecBase< DATA_TYPE, SIZE > & operator-= (VecBase< DATA_TYPE, SIZE > &v1, const VecBase< DATA_TYPE, SIZE > &v2)`  
*Subtracts v2 from v1 and stores the result in v1.*
- `template<class DATA_TYPE, unsigned SIZE> Vec< DATA_TYPE, SIZE > operator- (const VecBase< DATA_TYPE, SIZE > &v1, const VecBase< DATA_TYPE, SIZE > &v2)`  
*Subtracts v2 from v1 and returns the result.*
- `template<class DATA_TYPE, unsigned SIZE, class SCALAR_TYPE> VecBase< DATA_TYPE, SIZE > & operator*= (VecBase< DATA_TYPE, SIZE > &v1, const SCALAR_TYPE &scalar)`

*Multiplies v1 by a scalar value and stores the result in v1.*

- `template<class DATA_TYPE, unsigned SIZE, class SCALAR_TYPE>  
VecBase< DATA_TYPE, SIZE > operator * (const VecBase< DATA_TYPE,  
SIZE > &v1, const SCALAR_TYPE &scalar)`

*Multiplies v1 by a scalar value and returns the result.*

- `template<class DATA_TYPE, unsigned SIZE, class SCALAR_TYPE>  
VecBase< DATA_TYPE, SIZE > operator * (const SCALAR_TYPE &scalar,  
const VecBase< DATA_TYPE, SIZE > &v1)`

*Multiplies v1 by a scalar value and returns the result.*

- `template<class DATA_TYPE, unsigned SIZE, class SCALAR_TYPE>  
VecBase< DATA_TYPE, SIZE > & operator/= (VecBase< DATA_TYPE, SIZE  
> &v1, const SCALAR_TYPE &scalar)`

*Divides v1 by a scalar value and stores the result in v1.*

- `template<class DATA_TYPE, unsigned SIZE, class SCALAR_TYPE>  
VecBase< DATA_TYPE, SIZE > operator/ (const VecBase< DATA_TYPE,  
SIZE > &v1, const SCALAR_TYPE &scalar)`

*Divides v1 by a scalar value and returns the result.*

## Vector Operations

- `template<class DATA_TYPE, unsigned SIZE> DATA_TYPE dot (const Vec<  
DATA_TYPE, SIZE > &v1, const Vec< DATA_TYPE, SIZE > &v2)`

*Computes dot product of v1 and v2 and returns the result.*

- `template<class DATA_TYPE, unsigned SIZE> DATA_TYPE length (const  
Vec< DATA_TYPE, SIZE > &v1)`

*Computes the length of the given vector.*

- `template<class DATA_TYPE, unsigned SIZE> DATA_TYPE lengthSquared  
(const Vec< DATA_TYPE, SIZE > &v1)`

*Computes the square of the length of the given vector.*

- `template<class DATA_TYPE, unsigned SIZE> DATA_TYPE normalize (Vec<  
DATA_TYPE, SIZE > &v1)`

*Normalizes the given vector in place causing it to be of unit length.*

- `template<class DATA_TYPE, unsigned SIZE> bool isNormalized (const Vec<  
DATA_TYPE, SIZE > &v1, const DATA_TYPE eps=(DATA_TYPE) 0.0001)`

*Determines if the given vector is normalized within the given tolerance.*

- `template<class DATA_TYPE> Vec< DATA_TYPE, 3 > cross (const Vec< DATA_TYPE, 3 > &v1, const Vec< DATA_TYPE, 3 > &v2)`

*Computes the cross product between v1 and v2 and returns the result.*

- `template<class DATA_TYPE> Vec< DATA_TYPE, 3 > & cross (Vec< DATA_TYPE, 3 > &result, const Vec< DATA_TYPE, 3 > &v1, const Vec< DATA_TYPE, 3 > &v2)`

*Computes the cross product between v1 and v2 and stores the result in result.*

## Vector Interpolation

- `template<typename DATA_TYPE, unsigned SIZE> VecBase< DATA_TYPE, SIZE > & lerp (VecBase< DATA_TYPE, SIZE > &result, const DATA_TYPE &lerpVal, const VecBase< DATA_TYPE, SIZE > &from, const VecBase< DATA_TYPE, SIZE > &to)`

*Linearly interpolates between to vectors.*

## Vector Comparitors

- `template<class DATA_TYPE, unsigned SIZE> bool operator== (const VecBase< DATA_TYPE, SIZE > &v1, const VecBase< DATA_TYPE, SIZE > &v2)`

*Compares v1 and v2 to see if they are exactly the same with zero tolerance.*

- `template<class DATA_TYPE, unsigned SIZE> bool operator!= (const VecBase< DATA_TYPE, SIZE > &v1, const VecBase< DATA_TYPE, SIZE > &v2)`

*Compares v1 and v2 to see if they are NOT exactly the same with zero tolerance.*

- `template<class DATA_TYPE, unsigned SIZE> bool isEqual (const VecBase< DATA_TYPE, SIZE > &v1, const VecBase< DATA_TYPE, SIZE > &v2, const DATA_TYPE &eps)`

*Compares v1 and v2 to see if they are the same within the given epsilon tolerance.*

## Vector Transform (Quaternion)

- `template<typename DATA_TYPE> VecBase< DATA_TYPE, 3 > & xform (VecBase< DATA_TYPE, 3 > &result, const Quat< DATA_TYPE > &rot, const`

`VecBase< DATA_TYPE, 3 > &vector)`

*transform a vector by a rotation quaternion.*

- `template<typename DATA_TYPE> VecBase< DATA_TYPE, 3 > operator * (const Quat< DATA_TYPE > &rot, const VecBase< DATA_TYPE, 3 > &vector)`

*transform a vector by a rotation quaternion.*

## Vector Transform (Matrix)

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Vec< DATA_TYPE, COLS > & xform (Vec< DATA_TYPE, COLS > &result, const Matrix< DATA_TYPE, ROWS, COLS > &matrix, const Vec< DATA_TYPE, COLS > &vector)`

*xform a vector by a matrix.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Vec< DATA_TYPE, COLS > operator * (const Matrix< DATA_TYPE, ROWS, COLS > &matrix, const Vec< DATA_TYPE, COLS > &vector)`

*matrix \* vector xform.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, unsigned VEC_SIZE> Vec< DATA_TYPE, VEC_SIZE > & xform (Vec< DATA_TYPE, VEC_SIZE > &result, const Matrix< DATA_TYPE, ROWS, COLS > &matrix, const Vec< DATA_TYPE, VEC_SIZE > &vector)`

*partially transform a partially specified vector by a matrix, assumes last elt of vector is 0 (the 0 makes it only partially transformed).*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, unsigned COLS_MINUS_ONE> Vec< DATA_TYPE, COLS_MINUS_ONE > operator * (const Matrix< DATA_TYPE, ROWS, COLS > &matrix, const Vec< DATA_TYPE, COLS_MINUS_ONE > &vector)`

*matrix \* partial vector, assumes last elt of vector is 0 (partial transform).*

## Point Transform (Matrix)

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Point< DATA_TYPE, COLS > & xform (Point< DATA_TYPE, COLS > &result, const Matrix< DATA_TYPE, ROWS, COLS > &matrix, const Point< DATA_TYPE, COLS > &point)`

*transform point by a matrix.*

- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Point<DATA_TYPE, COLS> operator * (const Matrix<DATA_TYPE, ROWS, COLS> &matrix, const Point<DATA_TYPE, COLS> &point)`  
*matrix \* point.*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, unsigned PNT_SIZE> Point<DATA_TYPE, PNT_SIZE> & xform (Point<DATA_TYPE, PNT_SIZE> &result, const Matrix<DATA_TYPE, ROWS, COLS> &matrix, const Point<DATA_TYPE, PNT_SIZE> &point)`  
*transform a partially specified point by a matrix, assumes last elt of point is 1.*
- `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS, unsigned COLS_MINUS_ONE> Point<DATA_TYPE, COLS_MINUS_ONE> operator * (const Matrix<DATA_TYPE, ROWS, COLS> &matrix, const Point<DATA_TYPE, COLS_MINUS_ONE> &point)`  
*matrix \* partially specified point.*

## Constants

- `const float GMTL_EPSILON = 1.0e-6f`
- `const float GMTL_MAT_EQUAL_EPSILON = 0.001f`
- `const float GMTL_VEC_EQUAL_EPSILON = 0.0001f`

## Typedefs

- `typedef AABox<float> AABoxf`
- `typedef AABox<double> AABoxd`
- `typedef AxisAngle<float> AxisAnglef`
- `typedef AxisAngle<double> AxisAngled`
- `typedef Coord<Vec3d, EulerAngleXYZd> CoordVec3EulerAngleXYZd`
- `typedef Coord<Vec3f, EulerAngleXYZf> CoordVec3EulerAngleXYZf`
- `typedef Coord<Vec4d, EulerAngleXYZd> CoordVec4EulerAngleXYZd`
- `typedef Coord<Vec4f, EulerAngleXYZf> CoordVec4EulerAngleXYZf`
- `typedef Coord<Vec3d, EulerAngleZXYd> CoordVec3EulerAngleZXYd`
- `typedef Coord<Vec3f, EulerAngleZXYf> CoordVec3EulerAngleZXYf`
- `typedef Coord<Vec4d, EulerAngleZXYd> CoordVec4EulerAngleZXYd`
- `typedef Coord<Vec4f, EulerAngleZXYf> CoordVec4EulerAngleZXYf`
- `typedef Coord<Vec3d, EulerAngleZXYd> CoordVec3EulerAngleZXYd`
- `typedef Coord<Vec3f, EulerAngleZXYf> CoordVec3EulerAngleZXYf`
- `typedef Coord<Vec4d, EulerAngleZXYd> CoordVec4EulerAngleZXYd`

- typedef [Coord](#)< [Vec4f](#), [EulerAngleZXYf](#) > [CoordVec4EulerAngleZXYf](#)
- typedef [Coord](#)< [Vec3d](#), [AxisAngled](#) > [CoordVec3AxisAngled](#)
- typedef [Coord](#)< [Vec3f](#), [AxisAnglef](#) > [CoordVec3AxisAnglef](#)
- typedef [Coord](#)< [Vec4d](#), [AxisAngled](#) > [CoordVec4AxisAngled](#)
- typedef [Coord](#)< [Vec4f](#), [AxisAnglef](#) > [CoordVec4AxisAnglef](#)
- typedef [EulerAngle](#)< float, [XYZ](#) > [EulerAngleXYZf](#)
- typedef [EulerAngle](#)< double, [XYZ](#) > [EulerAngleXYZd](#)
- typedef [EulerAngle](#)< float, [ZYX](#) > [EulerAngleZYXf](#)
- typedef [EulerAngle](#)< double, [ZYX](#) > [EulerAngleZYXd](#)
- typedef [EulerAngle](#)< float, [ZXY](#) > [EulerAngleZXYf](#)
- typedef [EulerAngle](#)< double, [ZXY](#) > [EulerAngleZXYd](#)
- typedef [LineSeg](#)< float > [LineSegf](#)
- typedef [LineSeg](#)< double > [LineSegd](#)
- typedef [Matrix](#)< float, 2, 2 > [Matrix22f](#)
- typedef [Matrix](#)< double, 2, 2 > [Matrix22d](#)
- typedef [Matrix](#)< float, 2, 3 > [Matrix23f](#)
- typedef [Matrix](#)< double, 2, 3 > [Matrix23d](#)
- typedef [Matrix](#)< float, 3, 3 > [Matrix33f](#)
- typedef [Matrix](#)< double, 3, 3 > [Matrix33d](#)
- typedef [Matrix](#)< float, 3, 4 > [Matrix34f](#)
- typedef [Matrix](#)< double, 3, 4 > [Matrix34d](#)
- typedef [Matrix](#)< float, 4, 4 > [Matrix44f](#)
- typedef [Matrix](#)< double, 4, 4 > [Matrix44d](#)
- typedef [Plane](#)< float > [Planef](#)
- typedef [Plane](#)< double > [Planed](#)
- typedef [Point](#)< float, 3 > [Point3f](#)
- typedef [Point](#)< double, 3 > [Point3d](#)
- typedef [Point](#)< float, 4 > [Point4f](#)
- typedef [Point](#)< double, 4 > [Point4d](#)
- typedef [Quat](#)< float > [Quatf](#)
- typedef [Quat](#)< double > [Quatd](#)
- typedef [Sphere](#)< float > [Spheref](#)
- typedef [Sphere](#)< double > [Sphered](#)
- typedef [Vec](#)< float, 2 > [Vec2f](#)
- typedef [Vec](#)< double, 2 > [Vec2d](#)
- typedef [Vec](#)< float, 3 > [Vec3f](#)
- typedef [Vec](#)< double, 3 > [Vec3d](#)
- typedef [Vec](#)< float, 4 > [Vec4f](#)
- typedef [Vec](#)< double, 4 > [Vec4d](#)



## Enumerations

- enum [VectorIndex](#) { [Xelt](#) = 0, [Yelt](#) = 1, [Zelt](#) = 2, [Welt](#) = 3 }  
*use the values in this enum to index vector data types (such as [Vec](#), [Point](#), [Quat](#)).*
- enum [PlaneSide](#) { [ON\\_PLANE](#), [POS\\_SIDE](#), [NEG\\_SIDE](#) }  
*Used to describe where a point lies in relationship to a plane.*

## Functions

- const [AxisAngle](#)< float > [AXISANGLE\\_IDENTITYF](#) (0.0f, 1.0f, 0.0f, 0.0f)
- const [AxisAngle](#)< double > [AXISANGLE\\_IDENTITYD](#) (0.0, 1.0, 0.0, 0.0)
- template<class DATA\_TYPE> bool [isInVolume](#) (const [Sphere](#)< DATA\_TYPE > &container, const [Point](#)< DATA\_TYPE, 3 > &pt)  
*Tests if the given point is inside or on the surface of the given spherical volume.*
- template<class DATA\_TYPE> bool [isInVolume](#) (const [Sphere](#)< DATA\_TYPE > &container, const [Sphere](#)< DATA\_TYPE > &sphere)  
*Tests if the given sphere is completely inside or on the surface of the given spherical volume.*
- template<class DATA\_TYPE> void [extendVolume](#) ([Sphere](#)< DATA\_TYPE > &container, const [Point](#)< DATA\_TYPE, 3 > &pt)  
*Modifies the existing sphere to tightly enclose itself and the given point.*
- template<class DATA\_TYPE> void [extendVolume](#) ([Sphere](#)< DATA\_TYPE > &container, const [Sphere](#)< DATA\_TYPE > &sphere)  
*Modifies the container to tightly enclose itself and the given sphere.*
- template<class DATA\_TYPE> void [makeVolume](#) ([Sphere](#)< DATA\_TYPE > &container, const std::vector< [Point](#)< DATA\_TYPE, 3 > > &pts)  
*Modifies the given sphere to tightly enclose all points in the given std::vector.*
- template<class DATA\_TYPE> bool [isOnVolume](#) (const [Sphere](#)< DATA\_TYPE > &container, const [Point](#)< DATA\_TYPE, 3 > &pt)  
*Tests if the given point is on the surface of the container with zero tolerance.*
- template<class DATA\_TYPE> bool [isOnVolume](#) (const [Sphere](#)< DATA\_TYPE > &container, const [Point](#)< DATA\_TYPE, 3 > &pt, const DATA\_TYPE &tol)  
*Tests if the given point is on the surface of the container with the given tolerance.*

- const [EulerAngle](#)< float, [XYZ](#) > [EULERANGLE\\_IDENTITY\\_XYZF](#) (0.0f, 0.0f, 0.0f)
- const [EulerAngle](#)< double, [XYZ](#) > [EULERANGLE\\_IDENTITY\\_XYZD](#) (0.0, 0.0, 0.0)
- const [EulerAngle](#)< float, [ZYX](#) > [EULERANGLE\\_IDENTITY\\_ZYXF](#) (0.0f, 0.0f, 0.0f)
- const [EulerAngle](#)< double, [ZYX](#) > [EULERANGLE\\_IDENTITY\\_ZYXD](#) (0.0, 0.0, 0.0)
- const [EulerAngle](#)< float, [ZXY](#) > [EULERANGLE\\_IDENTITY\\_ZXYF](#) (0.0f, 0.0f, 0.0f)
- const [EulerAngle](#)< double, [ZXY](#) > [EULERANGLE\\_IDENTITY\\_ZXYD](#) (0.0, 0.0, 0.0)
- [Matrix44f](#) & [set](#) ([Matrix44f](#) &mat, const osg::Matrix &osg\_mat)

*Convert an opensg matrix to a [gml::Matrix](#).*

- osg::Matrix & [set](#) (osg::Matrix &osg\_mat, const [Matrix44f](#) &mat)
- void [GaussPointsFit](#) (int iQuantity, const Point3 \*akPoint, Point3 &rkCenter, Vec3 akAxis[3], float afExtent[3])
- bool [GaussPointsFit](#) (int iQuantity, const Vec3 \*akPoint, const bool \*abValid, Vec3 &rkCenter, Vec3 akAxis[3], float afExtent[3])
- bool [TestIntersect](#) (const [OOBox](#) &box0, const [OOBox](#) &box1)

*Test for intersection of two OOB's.*

- bool [TestIntersect](#) (float time, const [OOBox](#) &box0, const Vec3 &vel0, const [OOBox](#) &box1, const Vec3 &vel1)
- bool [TestIntersect](#) (float time, const [OOBox](#) &box0, const Vec3 &vel0, const [OOBox](#) &box1, const Vec3 &vel1, float &tFirstContact)
- template<bool FIND\_CONTACT> bool [dynObbFind0](#) (const float radii0, const float radiiT, const float rCenterSep, const float dt, float &tMaxContact)
- template<bool FIND\_CONTACT> bool [dynObbFind1](#) (const float radii0, const float rt0, const float rt1, const float rt2, const float rt3, const float rCenterSep, const float dt, float &tMaxContact)
- template<bool FIND\_CONTACT> bool [TestIntersectOBB](#) (float time, const [OOBox](#) &box0, const Vec3 &vel0, const [OOBox](#) &box1, const Vec3 &vel1, float &tFirstContact)
- template<class DATA\_TYPE> [Point](#)< DATA\_TYPE, 3 > [findNearestPt](#) (const [LineSeg](#)< DATA\_TYPE > &lineseg, const [Point](#)< DATA\_TYPE, 3 > &pt)

*Finds the closest point on the line segment to a given point.*

- template<class DATA\_TYPE> DATA\_TYPE [distance](#) (const [LineSeg](#)< DATA\_TYPE > &lineseg, const [Point](#)< DATA\_TYPE, 3 > &pt)

*Computes the shortest distance from the line segment to the given point.*

- `template<class DATA_TYPE> bool operator== (const LineSeg< DATA_TYPE > &ls1, const LineSeg< DATA_TYPE > &ls2)`  
*Compare two line segments to see if they are EXACTLY the same.*
- `template<class DATA_TYPE> bool operator!= (const LineSeg< DATA_TYPE > &ls1, const LineSeg< DATA_TYPE > &ls2)`  
*Compare two line segments to see if they are not EXACTLY the same.*
- `template<class DATA_TYPE> bool isEqual (const LineSeg< DATA_TYPE > &ls1, const LineSeg< DATA_TYPE > &ls2, const DATA_TYPE &eps)`  
*Compare two line segments to see if they are the same within the given tolerance.*
- `const Quat< float > QUAT_MULT_IDENTITYF (0.0f, 0.0f, 0.0f, 1.0f)`
- `const Quat< float > QUAT_ADD_IDENTITYF (0.0f, 0.0f, 0.0f, 0.0f)`
- `const Quat< float > QUAT_IDENTITYF (QUAT_MULT_IDENTITYF)`
- `const Quat< double > QUAT_MULT_IDENTITYD (0.0, 0.0, 0.0, 1.0)`
- `const Quat< double > QUAT_ADD_IDENTITYD (0.0, 0.0, 0.0, 0.0)`
- `const Quat< double > QUAT_IDENTITYD (QUAT_MULT_IDENTITYD)`

## Variables

- `const Matrix22f MAT_IDENTITY22F = Matrix22f()`  
*32bit floating point 2x2 identity matrix.*
- `const Matrix22d MAT_IDENTITY22D = Matrix22d()`  
*64bit floating point 2x2 identity matrix.*
- `const Matrix23f MAT_IDENTITY23F = Matrix23f()`  
*32bit floating point 2x2 identity matrix.*
- `const Matrix23d MAT_IDENTITY23D = Matrix23d()`  
*64bit floating point 2x2 identity matrix.*
- `const Matrix33f MAT_IDENTITY33F = Matrix33f()`  
*32bit floating point 3x3 identity matrix.*
- `const Matrix33d MAT_IDENTITY33D = Matrix33d()`  
*64bit floating point 3x3 identity matrix.*
- `const Matrix34f MAT_IDENTITY34F = Matrix34f()`  
*32bit floating point 3x4 identity matrix.*

- const [Matrix34d](#) MAT\_IDENTITY34D = [Matrix34d](#)()  
*64bit floating point 3x4 identity matrix.*
- const [Matrix44f](#) MAT\_IDENTITY44F = [Matrix44f](#)()  
*32bit floating point 4x4 identity matrix.*
- const [Matrix44d](#) MAT\_IDENTITY44D = [Matrix44d](#)()  
*64bit floating point 4x4 identity matrix.*
- const char \* [version](#) = GMTL\_XSTR(GMTL\_VERSION\_STRING)

### 9.1.1 Typedef Documentation

#### 9.1.1.1 typedef [AABox](#)<double> gmtl::AABoxd

Definition at line 165 of file AABox.h.

#### 9.1.1.2 typedef [AABox](#)<float> gmtl::AABoxf

Definition at line 164 of file AABox.h.

#### 9.1.1.3 typedef [AxisAngle](#)<double> gmtl::AxisAngled

Definition at line 147 of file AxisAngle.h.

#### 9.1.1.4 typedef [AxisAngle](#)<float> gmtl::AxisAnglef

Definition at line 146 of file AxisAngle.h.

#### 9.1.1.5 typedef [Coord](#)<[Vec3d](#), [AxisAngled](#)> gmtl::CoordVec3AxisAngled

Definition at line 88 of file Coord.h.

#### 9.1.1.6 typedef [Coord](#)<[Vec3f](#), [AxisAnglef](#)> gmtl::CoordVec3AxisAnglef

Definition at line 89 of file Coord.h.

**9.1.1.7** typedef [Coord](#)<[Vec3d](#), [EulerAngleXYZd](#)>  
gmtl::CoordVec3EulerAngleXYZd

Definition at line 73 of file Coord.h.

**9.1.1.8** typedef [Coord](#)<[Vec3f](#), [EulerAngleXYZf](#)>  
gmtl::CoordVec3EulerAngleXYZf

Definition at line 74 of file Coord.h.

**9.1.1.9** typedef [Coord](#)<[Vec3d](#), [EulerAngleZXYd](#)>  
gmtl::CoordVec3EulerAngleZXYd

Definition at line 83 of file Coord.h.

**9.1.1.10** typedef [Coord](#)<[Vec3f](#), [EulerAngleZXYf](#)>  
gmtl::CoordVec3EulerAngleZXYf

Definition at line 84 of file Coord.h.

**9.1.1.11** typedef [Coord](#)<[Vec3d](#), [EulerAngleZYXd](#)>  
gmtl::CoordVec3EulerAngleZYXd

Definition at line 78 of file Coord.h.

**9.1.1.12** typedef [Coord](#)<[Vec3f](#), [EulerAngleZYXf](#)>  
gmtl::CoordVec3EulerAngleZYXf

Definition at line 79 of file Coord.h.

**9.1.1.13** typedef [Coord](#)<[Vec4d](#), [AxisAngled](#)> gmtl::CoordVec4AxisAngled

Definition at line 90 of file Coord.h.

**9.1.1.14** typedef [Coord](#)<[Vec4f](#), [AxisAnglef](#)> gmtl::CoordVec4AxisAnglef

Definition at line 91 of file Coord.h.

**9.1.1.15**   `typedef Coord<Vec4d, EulerAngleXYZd>  
                  gmtl::CoordVec4EulerAngleXYZd`

Definition at line 75 of file `Coord.h`.

**9.1.1.16**   `typedef Coord<Vec4f, EulerAngleXYZf>  
                  gmtl::CoordVec4EulerAngleXYZf`

Definition at line 76 of file `Coord.h`.

**9.1.1.17**   `typedef Coord<Vec4d, EulerAngleZXYd>  
                  gmtl::CoordVec4EulerAngleZXYd`

Definition at line 85 of file `Coord.h`.

**9.1.1.18**   `typedef Coord<Vec4f, EulerAngleZXYf>  
                  gmtl::CoordVec4EulerAngleZXYf`

Definition at line 86 of file `Coord.h`.

**9.1.1.19**   `typedef Coord<Vec4d, EulerAngleZYXd>  
                  gmtl::CoordVec4EulerAngleZYXd`

Definition at line 80 of file `Coord.h`.

**9.1.1.20**   `typedef Coord<Vec4f, EulerAngleZYXf>  
                  gmtl::CoordVec4EulerAngleZYXf`

Definition at line 81 of file `Coord.h`.

**9.1.1.21**   `typedef EulerAngle<double, XYZ> gmtl::EulerAngleXYZd`

Definition at line 151 of file `EulerAngle.h`.

**9.1.1.22**   `typedef EulerAngle<float, XYZ> gmtl::EulerAngleXYZf`

Definition at line 150 of file `EulerAngle.h`.

**9.1.1.23** typedef [EulerAngle](#)<double, [ZXY](#)> gmtl::EulerAngleZXYd

Definition at line 155 of file EulerAngle.h.

**9.1.1.24** typedef [EulerAngle](#)<float, [ZXY](#)> gmtl::EulerAngleZXYf

Definition at line 154 of file EulerAngle.h.

**9.1.1.25** typedef [EulerAngle](#)<double, [ZYX](#)> gmtl::EulerAngleZYXd

Definition at line 153 of file EulerAngle.h.

**9.1.1.26** typedef [EulerAngle](#)<float, [ZYX](#)> gmtl::EulerAngleZYXf

Definition at line 152 of file EulerAngle.h.

**9.1.1.27** typedef [LineSeg](#)<double> gmtl::LineSegd

Definition at line 161 of file LineSeg.h.

**9.1.1.28** typedef [LineSeg](#)<float> gmtl::LineSegf

Definition at line 160 of file LineSeg.h.

**9.1.1.29** typedef [Matrix](#)<double, 2, 2> gmtl::Matrix22d

Definition at line 343 of file Matrix.h.

**9.1.1.30** typedef [Matrix](#)<float, 2, 2> gmtl::Matrix22f

Definition at line 342 of file Matrix.h.

**9.1.1.31** typedef [Matrix](#)<double, 2, 3> gmtl::Matrix23d

Definition at line 345 of file Matrix.h.

**9.1.1.32** typedef [Matrix](#)<float, 2, 3> gmtl::Matrix23f

Definition at line 344 of file Matrix.h.

**9.1.1.33** typedef **Matrix**<double, 3, 3> gmtl::Matrix33d

Definition at line 347 of file Matrix.h.

**9.1.1.34** typedef **Matrix**<float, 3, 3> gmtl::Matrix33f

Definition at line 346 of file Matrix.h.

**9.1.1.35** typedef **Matrix**<double, 3, 4> gmtl::Matrix34d

Definition at line 349 of file Matrix.h.

**9.1.1.36** typedef **Matrix**<float, 3, 4> gmtl::Matrix34f

Definition at line 348 of file Matrix.h.

**9.1.1.37** typedef **Matrix**<double, 4, 4> gmtl::Matrix44d

Definition at line 351 of file Matrix.h.

**9.1.1.38** typedef **Matrix**<float, 4, 4> gmtl::Matrix44f

Definition at line 350 of file Matrix.h.

Referenced by set().

**9.1.1.39** typedef **Plane**<double> gmtl::Planed

Definition at line 186 of file Plane.h.

**9.1.1.40** typedef **Plane**<float> gmtl::Planef

Definition at line 185 of file Plane.h.

**9.1.1.41** typedef **Point**<double,3> gmtl::Point3d

Definition at line 112 of file Point.h.



**9.1.1.42** typedef [Point](#)<float,3> gmtl::Point3f

Definition at line 111 of file Point.h.

**9.1.1.43** typedef [Point](#)<double,4> gmtl::Point4d

Definition at line 114 of file Point.h.

**9.1.1.44** typedef [Point](#)<float,4> gmtl::Point4f

Definition at line 113 of file Point.h.

**9.1.1.45** typedef [Quat](#)<double> gmtl::Quatd

Definition at line 170 of file Quat.h.

**9.1.1.46** typedef [Quat](#)<float> gmtl::Quatf

Definition at line 169 of file Quat.h.

**9.1.1.47** typedef [Sphere](#)<double> gmtl::Sphered

Definition at line 136 of file Sphere.h.

**9.1.1.48** typedef [Sphere](#)<float> gmtl::Spheref

Definition at line 135 of file Sphere.h.

**9.1.1.49** typedef [Vec](#)<double,2> gmtl::Vec2d

Definition at line 123 of file Vec.h.

**9.1.1.50** typedef [Vec](#)<float,2> gmtl::Vec2f

Definition at line 122 of file Vec.h.

**9.1.1.51** typedef [Vec](#)<double,3> gmtl::Vec3d

Definition at line 125 of file Vec.h.

**9.1.1.52** typedef [Vec](#)<float,3> gmtl::Vec3f

Definition at line 124 of file Vec.h.

**9.1.1.53** typedef [Vec](#)<double,4> gmtl::Vec4d

Definition at line 127 of file Vec.h.

**9.1.1.54** typedef [Vec](#)<float,4> gmtl::Vec4f

Definition at line 126 of file Vec.h.

**9.1.2 Function Documentation**

**9.1.2.1** const [AxisAngle](#)<double> AXISANGLE\_IDENTITYD (0. 0, 1. 0, 0. 0, 0. 0)

**9.1.2.2** const [AxisAngle](#)<float> AXISANGLE\_IDENTITYF (0. 0f, 1. 0f, 0. 0f, 0. 0f)

**9.1.2.3** template<class DATA\_TYPE> DATA\_TYPE distance (const [LineSeg](#)<DATA\_TYPE> & *lineseg*, const [Point](#)<DATA\_TYPE, 3> & *pt*)  
[inline]

Computes the shortest distance from the line segment to the given point.

**Parameters:**

*lineseg* the line segment to test

*pt* the point which to test against lineseg

**Returns:**

the shortest distance from pt to lineseg

Definition at line 68 of file LineSegOps.h.

References findNearestPt().

Referenced by whichSide().

```

70 {
71     return ( pt - findNearestPt( lineseg, pt ) );
72 }
```

#### 9.1.2.4 `template<bool FIND_CONTACT> bool dynObbFind0 (const float radii0, const float radiiT, const float rCenterSep, const float dt, float & tMaxContact) [inline]`

Definition at line 228 of file Intersection.h.

```

230     {
231         if(radii0 > rCenterSep)
232         {
233             if( radiiT > rCenterSep)
234                 return false;
235             if(FIND_CONTACT)
236             {
237                 float tmp(dt*((rCenterSep-radii0)/(radiiT-radii0)));
238                 if(tmp > tMaxContact)
239                 {
240                     tMaxContact = tmp;
241                 }
242             }
243         }
244         else if( radii0 < -rCenterSep)
245         {
246             if( radiiT < -rCenterSep)
247                 return false;
248             if(FIND_CONTACT)
249             {
250                 float tmp(-dt*((rCenterSep+radii0)/(radiiT-radii0)));
251                 if(tmp > tMaxContact)
252                 {
253                     tMaxContact = tmp;
254                 }
255             }
256         }
257         return true;
258     }
259 }
```

**9.1.2.5** `template<bool FIND_CONTACT> bool dynObbFind1 (const float radii0, const float rt0, const float rt1, const float rt2, const float rt3, const float rCenterSep, const float dt, float & tMaxContact) [inline]`

Definition at line 262 of file Intersection.h.

```

264     {
265         float radiiT;
266
267         if(radii0 > rCenterSep)
268         {
269             radiiT = (rt0*rt1)-(rt2*rt3);
270             if( radiiT > rCenterSep)
271                 return false;
272             if(FIND_CONTACT)
273             {
274                 float tmp(dt*((rCenterSep-radii0)/(radiiT-radii0)));
275                 if(tmp > tMaxContact)
276                 {
277                     tMaxContact = tmp;
278                 }
279             }
280         }
281         else if( radii0 < -rCenterSep)
282         {
283             radiiT = (rt0*rt1)-(rt2*rt3);
284             if( radiiT < -rCenterSep)
285                 return false;
286             if(FIND_CONTACT)
287             {
288                 float tmp(-dt*((rCenterSep+radii0)/(radiiT-radii0)));
289                 if(tmp > tMaxContact)
290                 {
291                     tMaxContact = tmp;
292                 }
293             }
294         }
295
296         return true;
297     }

```

**9.1.2.6** `const EulerAngle<double, XYZ> EULERANGLE_IDENTITY_XYZD (0. 0, 0. 0, 0. 0)`

**9.1.2.7** `const EulerAngle<float, XYZ> EULERANGLE_IDENTITY_XYZF (0. 0f, 0. 0f, 0. 0f)`

**9.1.2.8**    `const EulerAngle<double, ZXY> EULERANGLE_IDENTITY_ZXYD  
(0. 0, 0. 0, 0. 0)`

**9.1.2.9**    `const EulerAngle<float, ZXY> EULERANGLE_IDENTITY_ZXYF (0.  
0f, 0. 0f, 0. 0f)`

**9.1.2.10**   `const EulerAngle<double, ZYX> EULERANGLE_IDENTITY_ZYXD  
(0. 0, 0. 0, 0. 0)`

**9.1.2.11**   `const EulerAngle<float, ZYX> EULERANGLE_IDENTITY_ZYXF  
(0. 0f, 0. 0f, 0. 0f)`

**9.1.2.12**   `template<class DATA_TYPE> void extendVolume (Sphere<  
DATA_TYPE > & container, const Sphere< DATA_TYPE > & sphere)`

Modifies the container to tightly enclose itself and the given sphere.

**Parameters:**

*container* [in,out] the sphere that will be extended

*sphere* [in] the sphere which container should contain

Definition at line 133 of file Containment.h.

References `isInVolume()`, `gmtl::Sphere< DATA_TYPE >::mCenter`, `gmtl::Sphere< DATA_TYPE >::mRadius`, and `normalize()`.

```

135 {
136     // check if we already contain the sphere
137     if ( isInVolume( container, sphere ) )
138     {
139         return;
140     }
141
```

```

142 // make a vector pointing from the center of container to sphere. this is the
143 // direction in which we need to move container's center
144 Vec<DATA_TYPE, 3> dir = sphere.mCenter - container.mCenter;
145 DATA_TYPE len = normalize( dir );
146
147 // compute what the new radius should be
148 DATA_TYPE newRadius = (len + sphere.mRadius + container.mRadius) *
149                        DATA_TYPE(0.5);
150
151 // compute the new center for container
152 Point<DATA_TYPE, 3> newCenter = container.mCenter +
153                               (dir * (newRadius - container.mRadius));
154
155 // modify container to its new values
156 container.mCenter = newCenter;
157 container.mRadius = newRadius;
158 }

```

#### 9.1.2.13 `template<class DATA_TYPE> void extendVolume (Sphere<DATA_TYPE > & container, const Point< DATA_TYPE, 3 > & pt)`

Modifies the existing sphere to tightly enclose itself and the given point.

##### Parameters:

*container* [in,out] the sphere that will be extended

*pt* [in] the point which the sphere should contain

Definition at line 100 of file Containment.h.

References `isInVolume()`, `gmtl::Sphere< DATA_TYPE >::mCenter`, `gmtl::Sphere< DATA_TYPE >::mRadius`, and `normalize()`.

```

102 {
103 // check if we already contain the point
104 if ( isInVolume( container, pt ) )
105 {
106     return;
107 }
108
109 // make a vector pointing from the center of the sphere to pt. this is the
110 // direction in which we need to move the sphere's center
111 Vec<DATA_TYPE, 3> dir = pt - container.mCenter;
112 DATA_TYPE len = normalize( dir );
113
114 // compute what the new radius should be
115 DATA_TYPE newRadius = (len + container.mRadius) * DATA_TYPE(0.5);
116
117 // compute the new center for the sphere

```

```

118     Point<DATA_TYPE, 3> newCenter = container.mCenter +
119                                     (dir * (newRadius - container.mRadius));
120
121     // modify container to its new values
122     container.mCenter = newCenter;
123     container.mRadius = newRadius;
124 }

```

#### 9.1.2.14 `template<class DATA_TYPE> Point<DATA_TYPE, 3> findNearestPt(const LineSeg< DATA_TYPE > & lineseg, const Point< DATA_TYPE, 3> & pt)`

Finds the closest point on the line segment to a given point.

##### Parameters:

*lineseg* the line segment to test

*pt* the point which to test against lineseg

##### Returns:

the point on the line segment closest to pt

Definition at line 51 of file LineSegOps.h.

References `dot()`, `gmtl::LineSeg< DATA_TYPE >::mDir`, and `gmtl::LineSeg< DATA_TYPE >::mOrigin`.

Referenced by `distance()`.

```

53 {
54     // result = origin + dir * dot((pt-origin), dir)
55     return ( lineseg.mOrigin + lineseg.mDir *
56             dot(pt - lineseg.mOrigin, lineseg.mDir) );
57 }

```

#### 9.1.2.15 `bool gmtl::GaussPointsFit(int iQuantity, const Vec3 * akPoint, const bool * abValid, Vec3 & rkCenter, Vec3 akAxis[3], float afExtent[3])`

Definition at line 139 of file GaussPointsFit.h.

References `gmtl::Eigen::GetEigenvalue()`, `gmtl::Eigen::GetEigenvector()`, `gmtl::Eigen::IncrSortEigenStuff3()`, `gmtl::Eigen::Matrix()`, `Xelt`, `Yelt`, and `Zelt`.

```

142 {
143     // compute mean of points
144     rkCenter = ZeroVec3;
145     int i, iValidQuantity = 0;
146     for (i = 0; i < iQuantity; i++)
147     {
148         if ( abValid[i] )
149         {
150             rkCenter += akPoint[i];
151             iValidQuantity++;
152         }
153     }
154     if ( iValidQuantity == 0 )
155         return false;
156
157     float fInvQuantity = 1.0/iValidQuantity;
158     rkCenter *= fInvQuantity;
159
160     // compute covariances of points
161     float fSumXX = 0.0, fSumXY = 0.0, fSumXZ = 0.0;
162     float fSumYY = 0.0, fSumYZ = 0.0, fSumZZ = 0.0;
163     for (i = 0; i < iQuantity; i++)
164     {
165         if ( abValid[i] )
166         {
167             Vec3 kDiff = akPoint[i] - rkCenter;
168             fSumXX += kDiff[Xelt]*kDiff[Xelt];
169             fSumXY += kDiff[Xelt]*kDiff[Yelt];
170             fSumXZ += kDiff[Xelt]*kDiff[Zelt];
171             fSumYY += kDiff[Yelt]*kDiff[Yelt];
172             fSumYZ += kDiff[Yelt]*kDiff[Zelt];
173             fSumZZ += kDiff[Zelt]*kDiff[Zelt];
174         }
175     }
176     fSumXX *= fInvQuantity;
177     fSumXY *= fInvQuantity;
178     fSumXZ *= fInvQuantity;
179     fSumYY *= fInvQuantity;
180     fSumYZ *= fInvQuantity;
181     fSumZZ *= fInvQuantity;
182
183     // compute eigenvectors for covariance matrix
184     Eigen kES(3);
185     kES.Matrix(0,0) = fSumXX;
186     kES.Matrix(0,1) = fSumXY;
187     kES.Matrix(0,2) = fSumXZ;
188     kES.Matrix(1,0) = fSumXY;
189     kES.Matrix(1,1) = fSumYY;
190     kES.Matrix(1,2) = fSumYZ;
191     kES.Matrix(2,0) = fSumXZ;
192     kES.Matrix(2,1) = fSumYZ;
193     kES.Matrix(2,2) = fSumZZ;
194     kES.IncrSortEigenStuff3();
195
196     akAxis[0][Xelt] = kES.GetEigenvector(0,0);

```



```

197     akAxis[0][Yelt] = kES.GetEigenvector(1,0);
198     akAxis[0][Zelt] = kES.GetEigenvector(2,0);
199
200     akAxis[1][Xelt] = kES.GetEigenvector(0,1);
201     akAxis[1][Yelt] = kES.GetEigenvector(1,1);
202     akAxis[1][Zelt] = kES.GetEigenvector(2,1);
203
204     akAxis[2][Xelt] = kES.GetEigenvector(0,2);
205     akAxis[2][Yelt] = kES.GetEigenvector(1,2);
206     akAxis[2][Zelt] = kES.GetEigenvector(2,2);
207
208     afExtent[0] = kES.GetEigenvalue(0);
209     afExtent[1] = kES.GetEigenvalue(1);
210     afExtent[2] = kES.GetEigenvalue(2);
211
212     return true;
213 }

```

#### 9.1.2.16 void gmtl::GaussPointsFit (int *iQuantity*, const Point3 \* *akPoint*, Point3 & *rkCenter*, Vec3 *akAxis*[3], float *afExtent*[3])

Definition at line 76 of file GaussPointsFit.h.

References gmtl::Eigen::GetEigenvalue(), gmtl::Eigen::GetEigenvector(), gmtl::Eigen::IncrSortEigenStuff3(), gmtl::Eigen::Matrix(), Xelt, Yelt, and Zelt.

```

78 {
79     // compute mean of points
80     rkCenter = akPoint[0];
81     unsigned i;
82     for (i = 1; i < iQuantity; i++)
83         rkCenter += akPoint[i];
84     float fInvQuantity = 1.0f/iQuantity;
85     rkCenter *= fInvQuantity;
86
87     // compute covariances of points
88     float fSumXX = 0.0, fSumXY = 0.0, fSumXZ = 0.0;
89     float fSumYY = 0.0, fSumYZ = 0.0, fSumZZ = 0.0;
90     for (i = 0; i < iQuantity; i++)
91     {
92         Vec3 kDiff = akPoint[i] - rkCenter;
93         fSumXX += kDiff[Xelt]*kDiff[Xelt];
94         fSumXY += kDiff[Xelt]*kDiff[Yelt];
95         fSumXZ += kDiff[Xelt]*kDiff[Zelt];
96         fSumYY += kDiff[Yelt]*kDiff[Yelt];
97         fSumYZ += kDiff[Yelt]*kDiff[Zelt];
98         fSumZZ += kDiff[Zelt]*kDiff[Zelt];
99     }
100     fSumXX *= fInvQuantity;
101     fSumXY *= fInvQuantity;

```

```

102     fSumXZ *= fInvQuantity;
103     fSumYY *= fInvQuantity;
104     fSumYZ *= fInvQuantity;
105     fSumZZ *= fInvQuantity;
106
107     // compute eigenvectors for covariance matrix
108     gmtl::Eigen kES(3);
109     kES.Matrix(0,0) = fSumXX;
110     kES.Matrix(0,1) = fSumXY;
111     kES.Matrix(0,2) = fSumXZ;
112     kES.Matrix(1,0) = fSumXY;
113     kES.Matrix(1,1) = fSumYY;
114     kES.Matrix(1,2) = fSumYZ;
115     kES.Matrix(2,0) = fSumXZ;
116     kES.Matrix(2,1) = fSumYZ;
117     kES.Matrix(2,2) = fSumZZ;
118     kES.IncrSortEigenStuff3();
119
120     akAxis[0][Xelt] = kES.GetEigenvector(0,0);
121     akAxis[0][Yelt] = kES.GetEigenvector(1,0);
122     akAxis[0][Zelt] = kES.GetEigenvector(2,0);
123
124     akAxis[1][Xelt] = kES.GetEigenvector(0,1);
125     akAxis[1][Yelt] = kES.GetEigenvector(1,1);
126     akAxis[1][Zelt] = kES.GetEigenvector(2,1);
127
128     akAxis[2][Xelt] = kES.GetEigenvector(0,2);
129     akAxis[2][Yelt] = kES.GetEigenvector(1,2);
130     akAxis[2][Zelt] = kES.GetEigenvector(2,2);
131
132     afExtent[0] = kES.GetEigenvalue(0);
133     afExtent[1] = kES.GetEigenvalue(1);
134     afExtent[2] = kES.GetEigenvalue(2);
135 }

```

**9.1.2.17** `template<class DATA_TYPE> bool isEqual (const LineSeg< DATA_TYPE > & ls1, const LineSeg< DATA_TYPE > & ls2, const DATA_TYPE & eps) [inline]`

Compare two line segments to see if they are the same within the given tolerance.

**Parameters:**

- ls1* the first lineseg to compare
- ls2* the second lineseg to compare
- pre* the tolerance value to use

**Precondition:**

- eps* must be  $\geq 0$

**Returns:**

true if they are equal, false otherwise

Definition at line 119 of file LineSegOps.h.

References gmtlASSERT, isEqual(), gmtl::LineSeg< DATA\_TYPE >::mDir, and gmtl::LineSeg< DATA\_TYPE >::mOrigin.

```

122 {
123     gmtlASSERT( eps >= 0 );
124     return ( (isEqual(ls1.mOrigin, ls2.mOrigin, eps)) &&
125             (isEqual(ls1.mDir, ls2.mDir, eps)) );
126 }
```

#### 9.1.2.18 template<class DATA\_TYPE> bool isInVolume (const Sphere< DATA\_TYPE > & container, const Sphere< DATA\_TYPE > & sphere)

Tests if the given sphere is completely inside or on the surface of the given spherical volume.

**Parameters:**

*container* the sphere acting as the container

*sphere* the sphere that may be inside container

**Returns:**

true if sphere is inside container, false otherwise

Definition at line 82 of file Containment.h.

References length(), gmtl::Sphere< DATA\_TYPE >::mCenter, and gmtl::Sphere< DATA\_TYPE >::mRadius.

```

84 {
85     // the sphere is inside container if the distance between the centers of the
86     // spheres plus the radius of the inner sphere is less than or equal to the
87     // radius of the containing sphere.
88     // |sphere.center - container.center| + sphere.radius <= container.radius
89     return ( length(sphere.mCenter - container.mCenter) + sphere.mRadius
90             <= container.mRadius );
91 }
```

### 9.1.2.19 `template<class DATA_TYPE> bool isInVolume (const Sphere< DATA_TYPE > & container, const Point< DATA_TYPE, 3 > & pt)`

Tests if the given point is inside or on the surface of the given spherical volume.

#### Parameters:

*container* the sphere to test against

*pt* the point to test with

#### Returns:

true if pt is inside container, false otherwise

Definition at line 62 of file Containment.h.

References `length()`, `gmtl::Sphere< DATA_TYPE >::mCenter`, and `gmtl::Sphere< DATA_TYPE >::mRadius`.

Referenced by `extendVolume()`.

```

64 {
65     // The point is inside the sphere if the vector computed from the center of
66     // the sphere to the point has a magnitude less than or equal to the radius
67     // of the sphere.
68     // |pt - center| <= radius
69     return ( length(pt - container.mCenter) <= container.mRadius );
70 }
```

### 9.1.2.20 `template<class DATA_TYPE> bool isOnVolume (const Sphere< DATA_TYPE > & container, const Point< DATA_TYPE, 3 > & pt, const DATA_TYPE & tol)`

Tests if the given point is on the surface of the container with the given tolerance.

#### Parameters:

*container* the container to test against

*pt* the test point

*tol* the epsilon tolerance

#### Returns:

true if pt is on the surface of container, false otherwise

Definition at line 301 of file Containment.h.

References `gmtl::Math::abs()`, `gmtl::ASSERT`, `length()`, `gtml::Sphere< DATA_TYPE >::mCenter`, and `gtml::Sphere< DATA_TYPE >::mRadius`.

```

304 {
305     gmtlASSERT( tol >= 0 && "tolerance must be positive" );
306
307     // abs( |center-pt| - radius ) < tol
308     return ( Math::abs( length(container.mCenter - pt) - container.mRadius )
309             <= tol );
310 }

```

#### 9.1.2.21 `template<class DATA_TYPE> bool isOnVolume (const Sphere< DATA_TYPE > & container, const Point< DATA_TYPE, 3 > & pt)`

Tests if the given point is on the surface of the container with zero tolerance.

##### Parameters:

*container* the container to test against

*pt* the test point

##### Returns:

true if *pt* is on the surface of container, false otherwise

Definition at line 283 of file Containment.h.

References `length()`, `gmtl::Sphere< DATA_TYPE >::mCenter`, and `gmtl::Sphere< DATA_TYPE >::mRadius`.

```

285 {
286     // |center - pt| - radius == 0
287     return ( length(container.mCenter - pt) - container.mRadius == 0 );
288 }

```

#### 9.1.2.22 `template<class DATA_TYPE> void makeVolume (Sphere< DATA_TYPE > & container, const std::vector< Point< DATA_TYPE, 3 > > & pts)`

Modifies the given sphere to tightly enclose all points in the given `std::vector`.

This operation is  $O(n)$  and uses `sqrt(..)` liberally. :(

##### Parameters:

*container* [out] the sphere that will be modified to tightly enclose all the points in *pts*

*pts* [in] the list of points to contain

**Precondition:**

pts must contain at least 2 points

Definition at line 171 of file Containment.h.

References gmtlASSERT, lengthSquared(), gmtl::Sphere< DATA\_TYPE >::mCenter, gmtl::Sphere< DATA\_TYPE >::mRadius, and gmtl::Math::sqrt().

```

173 {
174     gmtlASSERT( pts.size() > 0  && "pts must contain at least 1 point" );
175
176     // Implementation based on the Sphere Centered at Average of Points algorithm
177     // found in "3D Game Engine Design" by Devud G, Eberly (pg. 27)
178     std::vector< Point<DATA_TYPE, 3> >::const_iterator itr = pts.begin();
179
180     // compute the average of the points as the center
181     Point<DATA_TYPE, 3> sum = *itr;
182     ++itr;
183     while ( itr != pts.end() )
184     {
185         sum += *itr;
186         ++itr;
187     }
188     container.mCenter = sum / pts.size();
189
190     // compute the distance from the computed center to point furthest from that
191     // center as the radius
192     DATA_TYPE radiusSqr(0);
193     for ( itr = pts.begin(); itr != pts.end(); ++itr )
194     {
195         float len = lengthSquared( *itr - container.mCenter );
196         if ( len > radiusSqr )
197             radiusSqr = len;
198     }
199
200     container.mRadius = Math::sqrt( radiusSqr );
201 }
```

**9.1.2.23** `template<class DATA_TYPE> bool operator!=(const LineSeg< DATA_TYPE > & ls1, const LineSeg< DATA_TYPE > & ls2)`  
`[inline]`

Compare two line segments to see if they are not EXACTLY the same.

In other words, this comparison is done with zero tolerance.

**Parameters:**

*ls1* the first lineseg to compare

*ls2* the second lineseg to compare

**Returns:**

true if they are not equal, false otherwise

Definition at line 100 of file LineSegOps.h.

```
102 {
103     return ( ! (ls1 == ls2) );
104 }
```

**9.1.2.24** `template<class DATA_TYPE> bool operator==(const LineSeg< DATA_TYPE > & ls1, const LineSeg< DATA_TYPE > & ls2)`  
[inline]

Compare two line segments to see if they are EXACTLY the same.

In other words, this comparison is done with zero tolerance.

**Parameters:**

*ls1* the first lineseg to compare

*ls2* the second lineseg to compare

**Returns:**

true if they are equal, false otherwise

Definition at line 85 of file LineSegOps.h.

References `gmtl::LineSeg< DATA_TYPE >::mDir`, and `gmtl::LineSeg< DATA_TYPE >::mOrigin`.

```
86 {
87     return ( (ls1.mOrigin == ls2.mOrigin) && (ls1.mDir == ls2.mDir) );
88 }
```

**9.1.2.25** `const Quat<double> QUAT_ADD_IDENTITYD (0. 0, 0. 0, 0. 0, 0. 0)`

**9.1.2.26** `const Quat<float> QUAT_ADD_IDENTITYF (0. 0f, 0. 0f, 0. 0f, 0. 0f)`

**9.1.2.27** `const Quat<double> QUAT_IDENTITYD  
(QUAT_MULT_IDENTITYD)`

**9.1.2.28** `const Quat<float> QUAT_IDENTITYF (QUAT_MULT_IDENTITYF)`

**9.1.2.29** `const Quat<double> QUAT_MULT_IDENTITYD (0. 0, 0. 0, 0. 0, 1. 0)`

**9.1.2.30** `const Quat<float> QUAT_MULT_IDENTITYF (0. 0f, 0. 0f, 0. 0f, 1. 0f)`

**9.1.2.31** `osg::Matrix& set (osg::Matrix & osg_mat, const Matrix44f & mat)  
[inline]`

Definition at line 26 of file OpenSGConvert.h.

References [Matrix44f](#).

```
27 {
28     osg_mat.setValue( mat.getData() );
29     return osg_mat;
30 }
```

**9.1.2.32** `Matrix44f& set (Matrix44f & mat, const osg::Matrix & osg_mat)  
[inline]`

Convert an opensg matrix to a [gmtl::Matrix](#).

Definition at line 19 of file OpenSGConvert.h.

References [Matrix44f](#), and `gmtl::Matrix< DATA_TYPE, ROWS, COLS >::set()`.

Referenced by `make()`, `makeRot()`, `set()`, and `setRot()`.

```
20 {
21     mat.set(osg_mat.getValues());
22     return mat;
23 }
```



### 9.1.2.33 `bool gmtl::TestIntersect (float time, const OOBox & box0, const Vec3 & vel0, const OOBox & box1, const Vec3 & vel1, float & tFirstContact)`

Definition at line 771 of file Intersection.h.

```

775     {
776         return TestIntersectOBB<true>(time, box0, vel0, box1, vel1, tFirstContact);
777     }
```

### 9.1.2.34 `bool gmtl::TestIntersect (float time, const OOBox & box0, const Vec3 & vel0, const OOBox & box1, const Vec3 & vel1)`

Definition at line 763 of file Intersection.h.

```

766     {
767         float unused;
768         return TestIntersectOBB<false>(time, box0, vel0, box1, vel1, unused);
769     }
```

### 9.1.2.35 `bool gmtl::TestIntersect (const OOBox & box0, const OOBox & box1)`

Test for intersection of two OOB's.

#### Parameters:

***box0*** First box

***box1*** Second box

#### Returns:

True - Boxes intersect

Definition at line 70 of file Intersection.h.

References `gmtl::Math::abs()`, `gmtl::OOBox::axes()`, `gmtl::OOBox::center()`, `dot()`, and `gmtl::OOBox::halfLens()`.

```

71     {
72         // convenience variables
73         const Vec3* aAxes = box0.axes();
```

```

74     const Vec3* bAxes = box1.axes();
75     const float* aExtents = box0.halfLens();
76     const float* bExtents = box1.halfLens();
77
78     // compute difference of box centers, D = C1-C0
79     Vec3 dist = box1.center() - box0.center();
80
81     float rMat[3][3];      // matrix rMat = A^T B, c_{ij} = Dot(A_i,B_j)
82     float rMatAbs[3][3];  // |c_{ij}|
83     // NOTE: Since it is not a Matrix4, I am using a different ordering here
84     float aDotD[3];        // Dot(A_i,D)
85     float fR0, fR1, fR;    // interval radii and distance between centers
86     float fR01;            // = R0 + R1
87
88     // axis C0+t*A0
89     rMat[0][0] = aAxes[0].dot(bAxes[0]);
90     rMat[0][1] = aAxes[0].dot(bAxes[1]);
91     rMat[0][2] = aAxes[0].dot(bAxes[2]);
92     aDotD[0] = aAxes[0].dot(dist);
93     rMatAbs[0][0] = Math::abs(rMat[0][0]);
94     rMatAbs[0][1] = Math::abs(rMat[0][1]);
95     rMatAbs[0][2] = Math::abs(rMat[0][2]);
96     fR = Math::abs(aDotD[0]);
97     fR1 = bExtents[0]*rMatAbs[0][0]+bExtents[1]*rMatAbs[0][1]+bExtents[2]*rMatAbs[0][2];
98     fR01 = aExtents[0] + fR1;
99     if ( fR > fR01 )
100         return false;
101
102     // axis C0+t*A1
103     rMat[1][0] = aAxes[1].dot(bAxes[0]);
104     rMat[1][1] = aAxes[1].dot(bAxes[1]);
105     rMat[1][2] = aAxes[1].dot(bAxes[2]);
106     aDotD[1] = aAxes[1].dot(dist);
107     rMatAbs[1][0] = Math::abs(rMat[1][0]);
108     rMatAbs[1][1] = Math::abs(rMat[1][1]);
109     rMatAbs[1][2] = Math::abs(rMat[1][2]);
110     fR = Math::abs(aDotD[1]);
111     fR1 = bExtents[0]*rMatAbs[1][0]+bExtents[1]*rMatAbs[1][1]+bExtents[2]*rMatAbs[1][2];
112     fR01 = aExtents[1] + fR1;
113     if ( fR > fR01 )
114         return false;
115
116     // axis C0+t*A2
117     rMat[2][0] = aAxes[2].dot(bAxes[0]);
118     rMat[2][1] = aAxes[2].dot(bAxes[1]);
119     rMat[2][2] = aAxes[2].dot(bAxes[2]);
120     aDotD[2] = aAxes[2].dot(dist);
121     rMatAbs[2][0] = Math::abs(rMat[2][0]);
122     rMatAbs[2][1] = Math::abs(rMat[2][1]);
123     rMatAbs[2][2] = Math::abs(rMat[2][2]);
124     fR = Math::abs(aDotD[2]);
125     fR1 = bExtents[0]*rMatAbs[2][0]+bExtents[1]*rMatAbs[2][1]+bExtents[2]*rMatAbs[2][2];
126     fR01 = aExtents[2] + fR1;
127     if ( fR > fR01 )
128         return false;

```

```

129
130     // axis C0+t*B0
131     fR = Math::abs(bAxes[0].dot(dist));
132     fR0 = aExtents[0]*rMatAbs[0][0]+aExtents[1]*rMatAbs[1][0]+aExtents[2]*rMatAbs[2][0];
133     fR01 = fR0 + bExtents[0];
134     if ( fR > fR01 )
135         return false;
136
137     // axis C0+t*B1
138     fR = Math::abs(bAxes[1].dot(dist));
139     fR0 = aExtents[0]*rMatAbs[0][1]+aExtents[1]*rMatAbs[1][1]+aExtents[2]*rMatAbs[2][1];
140     fR01 = fR0 + bExtents[1];
141     if ( fR > fR01 )
142         return false;
143
144     // axis C0+t*B2
145     fR = Math::abs(bAxes[2].dot(dist));
146     fR0 = aExtents[0]*rMatAbs[0][2]+aExtents[1]*rMatAbs[1][2]+aExtents[2]*rMatAbs[2][2];
147     fR01 = fR0 + bExtents[2];
148     if ( fR > fR01 )
149         return false;
150
151     // axis C0+t*A0xB0
152     fR = Math::abs(aDotD[2]*rMat[1][0]-aDotD[1]*rMat[2][0]);
153     fR0 = aExtents[1]*rMatAbs[2][0] + aExtents[2]*rMatAbs[1][0];
154     fR1 = bExtents[1]*rMatAbs[0][2] + bExtents[2]*rMatAbs[0][1];
155     fR01 = fR0 + fR1;
156     if ( fR > fR01 )
157         return false;
158
159     // axis C0+t*A0xB1
160     fR = Math::abs(aDotD[2]*rMat[1][1]-aDotD[1]*rMat[2][1]);
161     fR0 = aExtents[1]*rMatAbs[2][1] + aExtents[2]*rMatAbs[1][1];
162     fR1 = bExtents[0]*rMatAbs[0][2] + bExtents[2]*rMatAbs[0][0];
163     fR01 = fR0 + fR1;
164     if ( fR > fR01 )
165         return false;
166
167     // axis C0+t*A0xB2
168     fR = Math::abs(aDotD[2]*rMat[1][2]-aDotD[1]*rMat[2][2]);
169     fR0 = aExtents[1]*rMatAbs[2][2] + aExtents[2]*rMatAbs[1][2];
170     fR1 = bExtents[0]*rMatAbs[0][1] + bExtents[1]*rMatAbs[0][0];
171     fR01 = fR0 + fR1;
172     if ( fR > fR01 )
173         return false;
174
175     // axis C0+t*A1xB0
176     fR = Math::abs(aDotD[0]*rMat[2][0]-aDotD[2]*rMat[0][0]);
177     fR0 = aExtents[0]*rMatAbs[2][0] + aExtents[2]*rMatAbs[0][0];
178     fR1 = bExtents[1]*rMatAbs[1][2] + bExtents[2]*rMatAbs[1][1];
179     fR01 = fR0 + fR1;
180     if ( fR > fR01 )
181         return false;
182
183     // axis C0+t*A1xB1

```

```

184     fR = Math::abs(aDotD[0]*rMat[2][1]-aDotD[2]*rMat[0][1]);
185     fR0 = aExtents[0]*rMatAbs[2][1] + aExtents[2]*rMatAbs[0][1];
186     fR1 = bExtents[0]*rMatAbs[1][2] + bExtents[2]*rMatAbs[1][0];
187     fR01 = fR0 + fR1;
188     if ( fR > fR01 )
189         return false;
190
191     // axis C0+t*A1xB2
192     fR = Math::abs(aDotD[0]*rMat[2][2]-aDotD[2]*rMat[0][2]);
193     fR0 = aExtents[0]*rMatAbs[2][2] + aExtents[2]*rMatAbs[0][2];
194     fR1 = bExtents[0]*rMatAbs[1][1] + bExtents[1]*rMatAbs[1][0];
195     fR01 = fR0 + fR1;
196     if ( fR > fR01 )
197         return false;
198
199     // axis C0+t*A2xB0
200     fR = Math::abs(aDotD[1]*rMat[0][0]-aDotD[0]*rMat[1][0]);
201     fR0 = aExtents[0]*rMatAbs[1][0] + aExtents[1]*rMatAbs[0][0];
202     fR1 = bExtents[1]*rMatAbs[2][2] + bExtents[2]*rMatAbs[2][1];
203     fR01 = fR0 + fR1;
204     if ( fR > fR01 )
205         return false;
206
207     // axis C0+t*A2xB1
208     fR = Math::abs(aDotD[1]*rMat[0][1]-aDotD[0]*rMat[1][1]);
209     fR0 = aExtents[0]*rMatAbs[1][1] + aExtents[1]*rMatAbs[0][1];
210     fR1 = bExtents[0]*rMatAbs[2][2] + bExtents[2]*rMatAbs[2][0];
211     fR01 = fR0 + fR1;
212     if ( fR > fR01 )
213         return false;
214
215     // axis C0+t*A2xB2
216     fR = Math::abs(aDotD[1]*rMat[0][2]-aDotD[0]*rMat[1][2]);
217     fR0 = aExtents[0]*rMatAbs[1][2] + aExtents[1]*rMatAbs[0][2];
218     fR1 = bExtents[0]*rMatAbs[2][1] + bExtents[1]*rMatAbs[2][0];
219     fR01 = fR0 + fR1;
220     if ( fR > fR01 )
221         return false;
222
223     return true;
224
225 }

```

**9.1.2.36** `template<bool FIND_CONTACT> bool TestIntersectOBB (float time,  
const OBox & box0, const Vec3 & vel0, const OBox & box1, const  
Vec3 & vel1, float & tFirstContact) [inline]`

Definition at line 302 of file Intersection.h.

References `gmtl::Math::abs()`, `gmtl::OBox::axes()`, `gmtl::OBox::center()`, `dot()`,  
and `gmtl::OBox::halfLens()`.

```

306 {
307     // convenience variables
308     const Vec3* aAxes = box0.axes();
309     const Vec3* bAxes = box1.axes();
310     const float* aExtents = box0.halfLens();
311     const float* bExtents = box1.halfLens();
312
313     // Compute relative velocity of box1 with respect to box0 so that box0
314     // may as well be stationary.
315     Vec3 kW = vel1 - vel0;
316
317     // Compute difference of box centers at time 0 and time 'fTime'.
318     Vec3 dist0 = box1.center() - box0.center();
319     Vec3 dist1 = dist0 + time*kW;
320
321     float rMat[3][3];          // matrix C = A^T B, c_{ij} = dot(A_i,B_j)
322     float rMatAbs[3][3];      // |c_{ij}|
323     float aDotD0[3];          // dot(A_i,D0)
324     float aDotD1[3];          // dot(A_i,D1)
325     float bDotD0[3];          // dot(B_i,D0)
326     float bDotD1[3];          // dot(B_i,D1)
327     float fR0, fR1, fR;       // interval radii and distance between centers
328     float fR01;               // = R0 + R1
329
330     // Track minimum time
331     if(FIND_CONTACT)
332     {
333         tFirstContact = 0.0f;
334     }
335
336     // axis C0+t*A0
337     rMat[0][0] = aAxes[0].dot(bAxes[0]);
338     rMat[0][1] = aAxes[0].dot(bAxes[1]);
339     rMat[0][2] = aAxes[0].dot(bAxes[2]);
340     aDotD0[0] = aAxes[0].dot(dist0);
341     aDotD1[0] = aAxes[0].dot(dist1);
342     rMatAbs[0][0] = Math::abs(rMat[0][0]);
343     rMatAbs[0][1] = Math::abs(rMat[0][1]);
344     rMatAbs[0][2] = Math::abs(rMat[0][2]);
345     fR1 = bExtents[0]*rMatAbs[0][0]+bExtents[1]*rMatAbs[0][1]+bExtents[2]*rMatAbs[0][2];
346     fR01 = aExtents[0] + fR1;
347     //if(!FIND_CONTACT)
348     //{
349         if(!dynObbFind0<FIND_CONTACT>(aDotD0[0], aDotD1[0], fR01, time, tFirstContact))
350             return false;
351     //}
352     //else
353     //{
354
355     //}
356     /*
357     if ( aDotD0[0] > fR01 )
358     {
359         if ( aDotD1[0] > fR01 )
360             return false;

```

```

361     }
362     else if ( aDotD0[0] < -fR01 )
363     {
364         if ( aDotD1[0] < -fR01 )
365             return false;
366     }
367     */
368
369     // axis C0+t*A1
370     rMat[1][0] = aAxes[1].dot(bAxes[0]);
371     rMat[1][1] = aAxes[1].dot(bAxes[1]);
372     rMat[1][2] = aAxes[1].dot(bAxes[2]);
373     aDotD0[1] = aAxes[1].dot(dist0);
374     aDotD1[1] = aAxes[1].dot(dist1);
375     rMatAbs[1][0] = Math::abs(rMat[1][0]);
376     rMatAbs[1][1] = Math::abs(rMat[1][1]);
377     rMatAbs[1][2] = Math::abs(rMat[1][2]);
378     fR1 = bExtents[0]*rMatAbs[1][0]+bExtents[1]*rMatAbs[1][1]+bExtents[2]*rMatAbs[1][2];
379     fR01 = aExtents[1] + fR1;
380     if(!dynObbFind0<FIND_CONTACT>(aDotD0[1], aDotD1[1], fR01, time, tFirstContact))
381         return false;
382     /*
383     if ( aDotD0[1] > fR01 )
384     {
385         if ( aDotD1[1] > fR01 )
386             return false;
387     }
388     else if ( aDotD0[1] < -fR01 )
389     {
390         if ( aDotD1[1] < -fR01 )
391             return false;
392     }
393     */
394
395     // axis C0+t*A2
396     rMat[2][0] = aAxes[2].dot(bAxes[0]);
397     rMat[2][1] = aAxes[2].dot(bAxes[1]);
398     rMat[2][2] = aAxes[2].dot(bAxes[2]);
399     aDotD0[2] = aAxes[2].dot(dist0);
400     aDotD1[2] = aAxes[2].dot(dist1);
401     rMatAbs[2][0] = Math::abs(rMat[2][0]);
402     rMatAbs[2][1] = Math::abs(rMat[2][1]);
403     rMatAbs[2][2] = Math::abs(rMat[2][2]);
404     fR1 = bExtents[0]*rMatAbs[2][0]+bExtents[1]*rMatAbs[2][1]+bExtents[2]*rMatAbs[2][2];
405     fR01 = aExtents[2] + fR1;
406     if(!dynObbFind0<FIND_CONTACT>(aDotD0[2], aDotD1[2], fR01, time, tFirstContact))
407         return false;
408     /*
409     if ( aDotD0[2] > fR01 )
410     {
411         if ( aDotD1[2] > fR01 )
412             return false;
413     }
414     else if ( aDotD0[2] < -fR01 )
415     {

```

```

416         if ( aDotD1[2] < -fR01 )
417             return false;
418     }
419     */
420
421     // axis C0+t*B0
422     bDotD0[0] = bAxes[0].dot(dist0);
423     bDotD1[0] = bAxes[0].dot(dist1);
424     //fR = bAxes[0].dot(dist0);
425     fR0 = aExtents[0]*rMatAbs[0][0]+aExtents[1]*rMatAbs[1][0]+aExtents[2]*rMatAbs[2][0];
426     fR01 = fR0 + bExtents[0];
427     if(!dynObbFind0<FIND_CONTACT>(bDotD0[0], bDotD1[0], fR01, time, tFirstContact))
428         return false;
429     /*
430     if ( fR > fR01 )
431     {
432         fR = bAxes[0].dot(dist1);
433         if ( fR > fR01 )
434             return false;
435     }
436     else if ( fR < -fR01 )
437     {
438         fR = bAxes[0].dot(dist1);
439         if ( fR < -fR01 )
440             return false;
441     }
442     */
443
444     // axis C0+t*B1
445     bDotD0[1] = bAxes[1].dot(dist0);
446     bDotD1[1] = bAxes[1].dot(dist1);
447     //fR = bAxes[1].dot(dist0);
448     fR0 = aExtents[0]*rMatAbs[0][1]+aExtents[1]*rMatAbs[1][1]+aExtents[2]*rMatAbs[2][1];
449     fR01 = fR0 + bExtents[1];
450     if(!dynObbFind0<FIND_CONTACT>(bDotD0[1], bDotD1[1], fR01, time, tFirstContact))
451         return false;
452
453     /*
454     if ( fR > fR01 )
455     {
456         fR = bAxes[1].dot(dist1);
457         if ( fR > fR01 )
458             return false;
459     }
460     else if ( fR < -fR01 )
461     {
462         fR = bAxes[1].dot(dist1);
463         if ( fR < -fR01 )
464             return false;
465     }
466     */
467
468     // axis C0+t*B2
469     bDotD0[2] = bAxes[2].dot(dist0);
470     bDotD1[2] = bAxes[2].dot(dist1);

```

```

471 //fR = bAxes[2].dot(dist0);
472 fR0 = aExtents[0]*rMatAbs[0][2]+aExtents[1]*rMatAbs[1][2]+aExtents[2]*rMatAbs[2][2];
473 fR01 = fR0 + bExtents[2];
474 if(!dynObbFind0<FIND_CONTACT>(bDotD0[2], bDotD1[2], fR01, time, tFirstContact))
475     return false;
476
477 /*
478 if ( fR > fR01 )
479 {
480     fR = bAxes[2].dot(dist1);
481     if ( fR > fR01 )
482         return false;
483 }
484 else if ( fR < -fR01 )
485 {
486     fR = bAxes[2].dot(dist1);
487     if ( fR < -fR01 )
488         return false;
489 }
490 */
491
492 // axis C0+t*A0xB0
493 fR = aDotD0[2]*rMat[1][0]-aDotD0[1]*rMat[2][0];
494 fR0 = aExtents[1]*rMatAbs[2][0] + aExtents[2]*rMatAbs[1][0];
495 fR1 = bExtents[1]*rMatAbs[0][2] + bExtents[2]*rMatAbs[0][1];
496 fR01 = fR0 + fR1;
497
498 if(!dynObbFind1<FIND_CONTACT>(fR, aDotD1[2], rMat[1][0], aDotD1[1], rMat[2][0], fR01)
499     return false;
500
501 /*
502 if ( fR > fR01 )
503 {
504     fR = aDotD1[2]*rMat[1][0]-aDotD1[1]*rMat[2][0];
505     if ( fR > fR01 )
506         return false;
507 }
508 else if ( fR < -fR01 )
509 {
510     fR = aDotD1[2]*rMat[1][0]-aDotD1[1]*rMat[2][0];
511     if ( fR < -fR01 )
512         return false;
513 }
514 */
515
516 // axis C0+t*A0xB1
517 fR = aDotD0[2]*rMat[1][1]-aDotD0[1]*rMat[2][1];
518 fR0 = aExtents[1]*rMatAbs[2][1] + aExtents[2]*rMatAbs[1][1];
519 fR1 = bExtents[0]*rMatAbs[0][2] + bExtents[2]*rMatAbs[0][0];
520 fR01 = fR0 + fR1;
521 if(!dynObbFind1<FIND_CONTACT>(fR, aDotD1[2], rMat[1][1], aDotD1[1], rMat[2][1], fR01)
522     return false;
523
524 /*if ( fR > fR01 )
525 {
526     fR = aDotD1[2]*rMat[1][1]-aDotD1[1]*rMat[2][1];

```



```

526         if ( fR > fR01 )
527             return false;
528     }
529     else if ( fR < -fR01 )
530     {
531         fR = aDotD1[2]*rMat[1][1]-aDotD1[1]*rMat[2][1];
532         if ( fR < -fR01 )
533             return false;
534     }*/
535
536     // axis C0+t*A0xB2
537     fR = aDotD0[2]*rMat[1][2]-aDotD0[1]*rMat[2][2];
538     fR0 = aExtents[1]*rMatAbs[2][2] + aExtents[2]*rMatAbs[1][2];
539     fR1 = bExtents[0]*rMatAbs[0][1] + bExtents[1]*rMatAbs[0][0];
540     fR01 = fR0 + fR1;
541     if(!dynObbFind1<FIND_CONTACT>(fR,aDotD1[2],rMat[1][2],aDotD1[1],rMat[2][2],fR01, time, tFirstCor
542         return false;
543
544     /*if ( fR > fR01 )
545     {
546         fR = aDotD1[2]*rMat[1][2]-aDotD1[1]*rMat[2][2];
547         if ( fR > fR01 )
548             return false;
549     }
550     else if ( fR < -fR01 )
551     {
552         fR = aDotD1[2]*rMat[1][2]-aDotD1[1]*rMat[2][2];
553         if ( fR < -fR01 )
554             return false;
555     }*/
556
557     // axis C0+t*A1xB0
558     fR = aDotD0[0]*rMat[2][0]-aDotD0[2]*rMat[0][0];
559     fR0 = aExtents[0]*rMatAbs[2][0] + aExtents[2]*rMatAbs[0][0];
560     fR1 = bExtents[1]*rMatAbs[1][2] + bExtents[2]*rMatAbs[1][1];
561     fR01 = fR0 + fR1;
562     if(!dynObbFind1<FIND_CONTACT>(fR,aDotD1[0],rMat[2][0],aDotD1[2],rMat[0][0],fR01, time, tFirstCor
563         return false;
564     /*if ( fR > fR01 )
565     {
566         fR = aDotD1[0]*rMat[2][0]-aDotD1[2]*rMat[0][0];
567         if ( fR > fR01 )
568             return false;
569     }
570     else if ( fR < -fR01 )
571     {
572         fR = aDotD1[0]*rMat[2][0]-aDotD1[2]*rMat[0][0];
573         if ( fR < -fR01 )
574             return false;
575     }*/
576
577     // axis C0+t*A1xB1
578     fR = aDotD0[0]*rMat[2][1]-aDotD0[2]*rMat[0][1];
579     fR0 = aExtents[0]*rMatAbs[2][1] + aExtents[2]*rMatAbs[0][1];
580     fR1 = bExtents[0]*rMatAbs[1][2] + bExtents[2]*rMatAbs[1][0];

```

```

581     fR01 = fR0 + fR1;
582     if(!dynObbFind1<FIND_CONTACT>(fR,aDotD1[0],rMat[2][1],aDotD1[2],rMat[0][1],fR01, time))
583         return false;
584     /*if ( fR > fR01 )
585     {
586         fR = aDotD1[0]*rMat[2][1]-aDotD1[2]*rMat[0][1];
587         if ( fR > fR01 )
588             return false;
589     }
590     else if ( fR < -fR01 )
591     {
592         fR = aDotD1[0]*rMat[2][1]-aDotD1[2]*rMat[0][1];
593         if ( fR < -fR01 )
594             return false;
595     }*/
596
597     // axis C0+t*A1xB2
598     fR = aDotD0[0]*rMat[2][2]-aDotD0[2]*rMat[0][2];
599     fR0 = aExtents[0]*rMatAbs[2][2] + aExtents[2]*rMatAbs[0][2];
600     fR1 = bExtents[0]*rMatAbs[1][1] + bExtents[1]*rMatAbs[1][0];
601     fR01 = fR0 + fR1;
602     if(!dynObbFind1<FIND_CONTACT>(fR,aDotD1[0],rMat[2][2],aDotD1[2],rMat[0][2],fR01, time))
603         return false;
604     /*if ( fR > fR01 )
605     {
606         fR = aDotD1[0]*rMat[2][2]-aDotD1[2]*rMat[0][2];
607         if ( fR > fR01 )
608             return false;
609     }
610     else if ( fR < -fR01 )
611     {
612         fR = aDotD1[0]*rMat[2][2]-aDotD1[2]*rMat[0][2];
613         if ( fR < -fR01 )
614             return false;
615     }
616     */
617
618     // axis C0+t*A2xB0
619     fR = aDotD0[1]*rMat[0][0]-aDotD0[0]*rMat[1][0];
620     fR0 = aExtents[0]*rMatAbs[1][0] + aExtents[1]*rMatAbs[0][0];
621     fR1 = bExtents[1]*rMatAbs[2][2] + bExtents[2]*rMatAbs[2][1];
622     fR01 = fR0 + fR1;
623     if(!dynObbFind1<FIND_CONTACT>(fR,aDotD1[1],rMat[0][0],aDotD1[0],rMat[1][0],fR01, time))
624         return false;
625     /*if ( fR > fR01 )
626     {
627         fR = aDotD1[1]*rMat[0][0]-aDotD1[0]*rMat[1][0];
628         if ( fR > fR01 )
629             return false;
630     }
631     else if ( fR < -fR01 )
632     {
633         fR = aDotD1[1]*rMat[0][0]-aDotD1[0]*rMat[1][0];
634         if ( fR < -fR01 )
635             return false;

```

```

636     */
637
638     // axis C0+t*A2xB1
639     fR = aDotD0[1]*rMat[0][1]-aDotD0[0]*rMat[1][1];
640     fR0 = aExtents[0]*rMatAbs[1][1] + aExtents[1]*rMatAbs[0][1];
641     fR1 = bExtents[0]*rMatAbs[2][2] + bExtents[2]*rMatAbs[2][0];
642     fR01 = fR0 + fR1;
643     if(!dynObbFind1<FIND_CONTACT>(fR,aDotD1[1],rMat[0][1],aDotD1[0],rMat[1][1],fR01, time, tFirstCor
644         return false;
645     /*if ( fR > fR01 )
646     {
647         fR = aDotD1[1]*rMat[0][1]-aDotD1[0]*rMat[1][1];
648         if ( fR > fR01 )
649             return false;
650     }
651     else if ( fR < -fR01 )
652     {
653         fR = aDotD1[1]*rMat[0][1]-aDotD1[0]*rMat[1][1];
654         if ( fR < -fR01 )
655             return false;
656     }*/
657
658     // axis C0+t*A2xB2
659     fR = aDotD0[1]*rMat[0][2]-aDotD0[0]*rMat[1][2];
660     fR0 = aExtents[0]*rMatAbs[1][2] + aExtents[1]*rMatAbs[0][2];
661     fR1 = bExtents[0]*rMatAbs[2][1] + bExtents[1]*rMatAbs[2][0];
662     fR01 = fR0 + fR1;
663     if(!dynObbFind1<FIND_CONTACT>(fR,aDotD1[1],rMat[0][2],aDotD1[0],rMat[1][2],fR01, time, tFirstCor
664         return false;
665     /*if ( fR > fR01 )
666     {
667         fR = aDotD1[1]*rMat[0][2]-aDotD1[0]*rMat[1][2];
668         if ( fR > fR01 )
669             return false;
670     }
671     else if ( fR < -fR01 )
672     {
673         fR = aDotD1[1]*rMat[0][2]-aDotD1[0]*rMat[1][2];
674         if ( fR < -fR01 )
675             return false;
676     }*/
677
678     // At this point none of the 15 axes separate the boxes. It is still
679     // possible that they are separated as viewed in any plane orthogonal
680     // to the relative direction of motion W. In the worst case, the two
681     // projected boxes are hexagons. This requires three separating axis
682     // tests per box.
683     Vec3 kWxD0 = kW.cross(dist0);
684     float wDotA[3], wDotB[3];
685
686     // axis C0 + t*WxA0
687     wDotA[1] = kW.dot(aAxes[1]);
688     wDotA[2] = kW.dot(aAxes[2]);
689     fR = Math::abs(aAxes[0].dot(kWxD0));
690     fR0 = aExtents[1]*wDotA[2] + aExtents[2]*wDotA[1];

```

```

691     fR1 =
692         bExtents[0]*Math::abs(rMat[1][0]*wDotA[2] - rMat[2][0]*wDotA[1]) +
693         bExtents[1]*Math::abs(rMat[1][1]*wDotA[2] - rMat[2][1]*wDotA[1]) +
694         bExtents[2]*Math::abs(rMat[1][2]*wDotA[2] - rMat[2][2]*wDotA[1]);
695     fR01 = fR0 + fR1;
696     if ( fR > fR01 )
697         return false;
698
699     // axis C0 + t*WxA1
700     wDotA[0] = kW.dot(aAxes[0]);
701     fR = Math::abs(aAxes[1].dot(kWxD0));
702     fR0 = aExtents[2]*wDotA[0] + aExtents[0]*wDotA[2];
703     fR1 =
704         bExtents[0]*Math::abs(rMat[2][0]*wDotA[0] - rMat[0][0]*wDotA[2]) +
705         bExtents[1]*Math::abs(rMat[2][1]*wDotA[0] - rMat[0][1]*wDotA[2]) +
706         bExtents[2]*Math::abs(rMat[2][2]*wDotA[0] - rMat[0][2]*wDotA[2]);
707     fR01 = fR0 + fR1;
708     if ( fR > fR01 )
709         return false;
710
711     // axis C0 + t*WxA2
712     fR = Math::abs(aAxes[2].dot(kWxD0));
713     fR0 = aExtents[0]*wDotA[1] + aExtents[1]*wDotA[0];
714     fR1 =
715         bExtents[0]*Math::abs(rMat[0][0]*wDotA[1] - rMat[1][0]*wDotA[0]) +
716         bExtents[1]*Math::abs(rMat[0][1]*wDotA[1] - rMat[1][1]*wDotA[0]) +
717         bExtents[2]*Math::abs(rMat[0][2]*wDotA[1] - rMat[1][2]*wDotA[0]);
718     fR01 = fR0 + fR1;
719     if ( fR > fR01 )
720         return false;
721
722     // axis C0 + t*WxB0
723     wDotB[1] = kW.dot(bAxes[1]);
724     wDotB[2] = kW.dot(bAxes[2]);
725     fR = Math::abs(bAxes[0].dot(kWxD0));
726     fR0 =
727         aExtents[0]*Math::abs(rMat[0][1]*wDotB[2] - rMat[0][2]*wDotB[1]) +
728         aExtents[1]*Math::abs(rMat[1][1]*wDotB[2] - rMat[1][2]*wDotB[1]) +
729         aExtents[2]*Math::abs(rMat[2][1]*wDotB[2] - rMat[2][2]*wDotB[1]);
730     fR1 = bExtents[1]*wDotB[2] + bExtents[2]*wDotB[1];
731     fR01 = fR0 + fR1;
732     if ( fR > fR01 )
733         return false;
734
735     // axis C0 + t*WxB1
736     wDotB[0] = kW.dot(bAxes[0]);
737     fR = Math::abs(bAxes[1].dot(kWxD0));
738     fR0 =
739         aExtents[0]*Math::abs(rMat[0][2]*wDotB[0] - rMat[0][0]*wDotB[2]) +
740         aExtents[1]*Math::abs(rMat[1][2]*wDotB[0] - rMat[1][0]*wDotB[2]) +
741         aExtents[2]*Math::abs(rMat[2][2]*wDotB[0] - rMat[2][0]*wDotB[2]);
742     fR1 = bExtents[2]*wDotB[0] + bExtents[0]*wDotB[2];
743     fR01 = fR0 + fR1;
744     if ( fR > fR01 )
745         return false;

```

```

746
747     // axis C0 + t*WxB2
748     fR = Math::abs(bAxes[2].dot(kWxD0));
749     fR0 =
750         aExtents[0]*Math::abs(rMat[0][0]*wDotB[1] - rMat[0][1]*wDotB[0]) +
751         aExtents[1]*Math::abs(rMat[1][0]*wDotB[1] - rMat[1][1]*wDotB[0]) +
752         aExtents[2]*Math::abs(rMat[2][0]*wDotB[1] - rMat[2][1]*wDotB[0]);
753     fR1 = bExtents[0]*wDotB[1] + bExtents[1]*wDotB[0];
754     fR01 = fR0 + fR1;
755     if ( fR > fR01 )
756         return false;
757
758     return true;
759
760 }
```

### 9.1.3 Variable Documentation

#### 9.1.3.1 `const Matrix22d gmtl::MAT_IDENTITY22D = Matrix22d()`

64bit floating point 2x2 identity matrix.

Definition at line 357 of file Matrix.h.

#### 9.1.3.2 `const Matrix22f gmtl::MAT_IDENTITY22F = Matrix22f()`

32bit floating point 2x2 identity matrix.

Definition at line 354 of file Matrix.h.

#### 9.1.3.3 `const Matrix23d gmtl::MAT_IDENTITY23D = Matrix23d()`

64bit floating point 2x2 identity matrix.

Definition at line 363 of file Matrix.h.

#### 9.1.3.4 `const Matrix23f gmtl::MAT_IDENTITY23F = Matrix23f()`

32bit floating point 2x2 identity matrix.

Definition at line 360 of file Matrix.h.

**9.1.3.5** `const Matrix33d gmtl::MAT_IDENTITY33D = Matrix33d()`

64bit floating point 3x3 identity matrix.

Definition at line 369 of file Matrix.h.

**9.1.3.6** `const Matrix33f gmtl::MAT_IDENTITY33F = Matrix33f()`

32bit floating point 3x3 identity matrix.

Definition at line 366 of file Matrix.h.

**9.1.3.7** `const Matrix34d gmtl::MAT_IDENTITY34D = Matrix34d()`

64bit floating point 3x4 identity matrix.

Definition at line 375 of file Matrix.h.

**9.1.3.8** `const Matrix34f gmtl::MAT_IDENTITY34F = Matrix34f()`

32bit floating point 3x4 identity matrix.

Definition at line 372 of file Matrix.h.

**9.1.3.9** `const Matrix44d gmtl::MAT_IDENTITY44D = Matrix44d()`

64bit floating point 4x4 identity matrix.

Definition at line 381 of file Matrix.h.

**9.1.3.10** `const Matrix44f gmtl::MAT_IDENTITY44F = Matrix44f()`

32bit floating point 4x4 identity matrix.

Definition at line 378 of file Matrix.h.

**9.1.3.11** `const char* gmtl::version = GMTL_XSTR(GMTL_VERSION_-  
STRING)`

Definition at line 140 of file Version.h.

## 9.2 gmtl::Math Namespace Reference

### C Math Abstraction

- `template<typename T> T abs (T iValue)`
- `template<typename T> T ceil (T fValue)`
- `float ceil (float fValue)`
- `double ceil (double fValue)`
- `template<typename T> T floor (T fValue)`
- `float floor (float fValue)`
- `double floor (double fValue)`
- `template<typename T> T sign (int iValue)`
- `template<typename T> T zeroClamp (T value, T eps=T(0))`  
*Clamps the given value down to zero if it is within epsilon of zero.*

- `template<typename T> T aCos (T fValue)`
- `float aCos (float fValue)`
- `double aCos (double fValue)`
- `template<typename T> T aSin (T fValue)`
- `float aSin (float fValue)`
- `double aSin (double fValue)`
- `template<typename T> T aTan (T fValue)`
- `double aTan (double fValue)`
- `float aTan (float fValue)`
- `template<typename T> T atan2 (T fY, T fX)`
- `float aTan2 (float fY, float fX)`
- `double aTan2 (double fY, double fX)`
- `template<typename T> T cos (T fValue)`
- `float cos (float fValue)`
- `double cos (double fValue)`
- `template<typename T> T exp (T fValue)`
- `float exp (float fValue)`
- `double exp (double fValue)`
- `template<typename T> T log (T fValue)`
- `double log (double fValue)`
- `float log (float fValue)`
- `double pow (double fBase, double fExponent)`
- `float pow (float fBase, float fExponent)`
- `template<typename T> T sin (T fValue)`

- double [sin](#) (double fValue)
- float [sin](#) (float fValue)
- template<typename T> T [tan](#) (T fValue)
- double [tan](#) (double fValue)
- float [tan](#) (float fValue)
- template<typename T> T [sqr](#) (T fValue)
- template<typename T> T [sqrt](#) (T fValue)
- double [sqrt](#) (double fValue)
- float [unitRandom](#) ()  
*get a random number between 0 and 1.*
- float [rangeRandom](#) (float x1, float x2)  
*return a random number between x1 and x2 RETURNS: random number between x1 and x2.*
- float [deg2Rad](#) (float fVal)
- double [deg2Rad](#) (double fVal)
- float [rad2Deg](#) (float fVal)
- double [rad2Deg](#) (double fVal)
- template<class T> bool [isEqual](#) (const T &a, const T &b, const T &tolerance)  
*Is almost equal? test for equality within some tolerance...*
- template<class T> T [trunc](#) (T val)  
*cut off the digits after the decimal place.*
- template<class T> T [round](#) (T p)  
*round to nearest integer.*
- template<class T> T [Min](#) (const T &x, const T &y)  
*min returns the minimum of 2 values.*
- template<class T> T [Min](#) (const T &x, const T &y, const T &z)  
*min returns the minimum of 3 values.*
- template<class T> T [Min](#) (const T &w, const T &x, const T &y, const T &z)  
*min returns the minimum of 4 values.*
- template<class T> T [Max](#) (const T &x, const T &y)  
*max returns the maximum of 2 values.*
- template<class T> T [Max](#) (const T &x, const T &y, const T &z)  
*max returns the maximum of 3 values.*



- `template<class T> T Max (const T &w, const T &x, const T &y, const T &z)`  
*max returns the maximum of 4 values.*
- `template<class T> T factorial (T rhs)`  
*Compute the factorial.*

### Scalar type interpolation (for doubles, floats, etc...)

- `template<class T, typename U> void lerp (T &result, const U &lerp, const T &a, const T &b)`  
*Linear Interpolation between number [a] and [b].*

### Mathematical constants

- `const float PI = 3.14159265358979323846f`
- `const float PI\_OVER\_2 = 1.57079632679489661923f`
- `const float PI\_OVER\_4 = 0.78539816339744830962f`



---

## Chapter 10

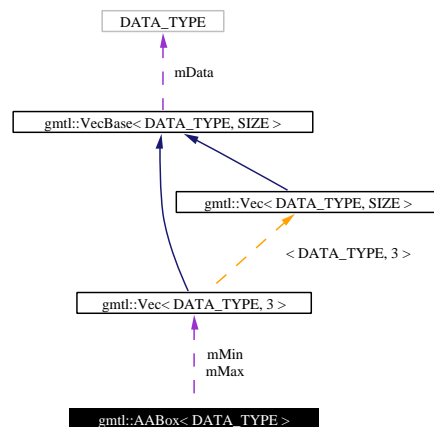
# GenericMathTemplateLibrary Class Documentation

### 10.1 gmtl::AABBox< DATA\_TYPE > Class Template Reference

Describes an axially aligned box in 3D space.

```
#include <AABBox.h>
```

Collaboration diagram for gmtl::AABBox< DATA\_TYPE >:



## Public Types

- typedef DATA\_TYPE [DataType](#)

## Public Methods

- [AABox](#) ()  
*Creates a new empty box.*
- [AABox](#) (const [Vec](#)< DATA\_TYPE, 3 > &min, const [Vec](#)< DATA\_TYPE, 3 > &max)  
*Creates a new box with the given min and max points.*
- [AABox](#) (const [AABox](#)< DATA\_TYPE > &box)  
*Constructs a duplicate of the given box.*
- const [Vec](#)< DATA\_TYPE, 3 > & [getMin](#) () const  
*Gets the minimum point of the box.*
- const [Vec](#)< DATA\_TYPE, 3 > & [getMax](#) () const  
*Gets the maximum point of the box.*
- bool [isEmpty](#) () const  
*Tests if this box is empty.*
- void [setMin](#) (const [Vec](#)< DATA\_TYPE, 3 > &min)  
*Sets the minimum point of the box.*
- void [setMax](#) (const [Vec](#)< DATA\_TYPE, 3 > &max)  
*Sets the maximum point of the box.*
- void [setEmpty](#) (bool empty)  
*Sets the empty flag on this box.*

## Public Attributes

- [Vec](#)< DATA\_TYPE, 3 > [mMin](#)  
*The minimum point of the box.*
- [Vec](#)< DATA\_TYPE, 3 > [mMax](#)  
*The maximum point on the box.*

- bool `mEmpty`  
*Flag for empty box.*

### 10.1.1 Detailed Description

**template<class DATA\_TYPE> class gmtl::AABox< DATA\_TYPE >**

Describes an axially aligned box in 3D space.

This is usually used for graphics applications. It is defined by its minimum and maximum points.

**Parameters:**

*DATA\_TYPE* the internal type used for the points

Definition at line 51 of file AABox.h.

### 10.1.2 Member Typedef Documentation

**10.1.2.1 template<class DATA\_TYPE> typedef DATA\_TYPE gmtl::AABox< DATA\_TYPE >::DataType**

Definition at line 54 of file AABox.h.

### 10.1.3 Constructor & Destructor Documentation

**10.1.3.1 template<class DATA\_TYPE> gmtl::AABox< DATA\_TYPE >::AABox() [inline]**

Creates a new empty box.

Definition at line 60 of file AABox.h.

References gmtl::AABox< DATA\_TYPE >::mEmpty, gmtl::AABox< DATA\_TYPE >::mMax, and gmtl::AABox< DATA\_TYPE >::mMin.

```
61         : mMin(0,0,0), mMax(0,0,0), mEmpty(true)
62         {}
```

**10.1.3.2** `template<class DATA_TYPE> gmtl::AABBox< DATA_TYPE  
>::AABBox (const Vec< DATA_TYPE, 3 > & min, const Vec<  
DATA_TYPE, 3 > & max) [inline]`

Creates a new box with the given min and max points.

**Parameters:**

*min* the minimum point on the box

*max* the maximum point on the box

**Precondition:**

all elements of min are less than max

bot min and max are not zero

Definition at line 73 of file AABox.h.

References gmtl::AABBox< DATA\_TYPE >::mEmpty, gmtl::AABBox< DATA\_TYPE  
>::mMax, and gmtl::AABBox< DATA\_TYPE >::mMin.

```
74         : mMin(min), mMax(max), mEmpty(false)
75     {}
```

**10.1.3.3** `template<class DATA_TYPE> gmtl::AABBox< DATA_TYPE  
>::AABBox (const AABBox< DATA_TYPE > & box) [inline]`

Constructs a duplicate of the given box.

**Parameters:**

*box* the box to make a copy of

Definition at line 82 of file AABox.h.

References gmtl::AABBox< DATA\_TYPE >::mEmpty, gmtl::AABBox< DATA\_TYPE  
>::mMax, and gmtl::AABBox< DATA\_TYPE >::mMin.

```
83         : mMin(box.mMin), mMax(box.mMax), mEmpty(box.mEmpty)
84     {}
```

## 10.1.4 Member Function Documentation

**10.1.4.1** `template<class DATA_TYPE> const Vec<DATA_TYPE, 3>& gmtl::AABox< DATA_TYPE >::getMax () const [inline]`

Gets the maximum point of the box.

**Returns:**

the max point

Definition at line 101 of file AABox.h.

References gmtl::AABox< DATA\_TYPE >::mMax.

```
102      {  
103          return mMax;  
104      }
```

**10.1.4.2** `template<class DATA_TYPE> const Vec<DATA_TYPE, 3>& gmtl::AABox< DATA_TYPE >::getMin () const [inline]`

Gets the minimum point of the box.

**Returns:**

the min point

Definition at line 91 of file AABox.h.

References gmtl::AABox< DATA\_TYPE >::mMin.

```
92      {  
93          return mMin;  
94      }
```

**10.1.4.3** `template<class DATA_TYPE> bool gmtl::AABox< DATA_TYPE >::isEmpty () const [inline]`

Tests if this box is empty.

**Returns:**

true if the box is empty, false otherwise

Definition at line 111 of file AABox.h.

References gmtl::AABox< DATA\_TYPE >::mEmpty.

```

112     {
113         return mEmpty;
114     }

```

#### 10.1.4.4 `template<class DATA_TYPE> void gmtl::AABox< DATA_TYPE >::setEmpty (bool empty)` `[inline]`

Sets the empty flag on this box.

##### Parameters:

*empty* true to make the box empty, false otherwise

Definition at line 141 of file AABox.h.

References gmtl::AABox< DATA\_TYPE >::mEmpty.

```

142     {
143         mEmpty = empty;
144     }

```

#### 10.1.4.5 `template<class DATA_TYPE> void gmtl::AABox< DATA_TYPE >::setMax (const Vec< DATA_TYPE, 3 > & max)` `[inline]`

Sets the maximum point of the box.

##### Parameters:

*max* the max point

Definition at line 131 of file AABox.h.

References gmtl::AABox< DATA\_TYPE >::mMax.

```

132     {
133         mMax = max;
134     }

```



**10.1.4.6** `template<class DATA_TYPE> void gmtl::AABox< DATA_TYPE  
>::setMin (const Vec< DATA_TYPE, 3 > & min) [inline]`

Sets the minimum point of the box.

**Parameters:**

*min* the min point

Definition at line 121 of file AABox.h.

References gmtl::AABox< DATA\_TYPE >::mMin.

```
122      {
123          mMin = min;
124      }
```

## 10.1.5 Member Data Documentation

**10.1.5.1** `template<class DATA_TYPE> bool gmtl::AABox< DATA_TYPE  
>::mEmpty`

Flag for empty box.

True if the box is empty.

Definition at line 160 of file AABox.h.

Referenced by gmtl::AABox< DATA\_TYPE >::AABox(), gmtl::AABox< DATA\_TYPE >::isEmpty(), and gmtl::AABox< DATA\_TYPE >::setEmpty().

**10.1.5.2** `template<class DATA_TYPE> Vec<DATA_TYPE, 3> gmtl::AABox<  
DATA_TYPE >::mMax`

The maximum point on the box.

Definition at line 155 of file AABox.h.

Referenced by gmtl::AABox< DATA\_TYPE >::AABox(), gmtl::AABox< DATA\_TYPE >::getMax(), and gmtl::AABox< DATA\_TYPE >::setMax().

**10.1.5.3** `template<class DATA_TYPE> Vec<DATA_TYPE, 3> gmtl::AABox<  
DATA_TYPE >::mMin`

The minimum point of the box.

Definition at line 150 of file AABox.h.

Referenced by `gmtl::AABox< DATA_TYPE >::AABox()`, `gmtl::AABox< DATA_TYPE >::getMin()`, and `gmtl::AABox< DATA_TYPE >::setMin()`.

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

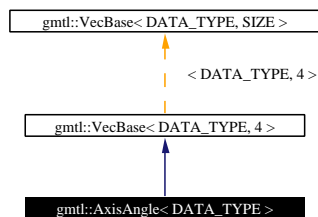
- [AABox.h](#)

## 10.2 gmtl::AxisAngle< DATA\_TYPE > Class Template Reference

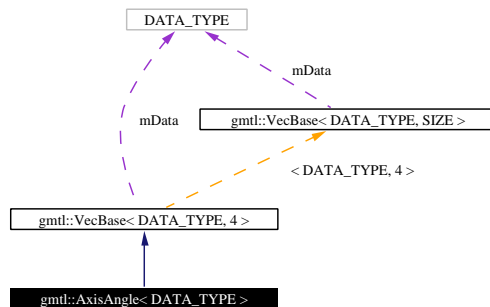
**AxisAngle**: Represents a "twist about an axis" **AxisAngle** is used to specify a rotation in 3-space.

```
#include <AxisAngle.h>
```

Inheritance diagram for gmtl::AxisAngle:



Collaboration diagram for gmtl::AxisAngle< DATA\_TYPE >:



### Public Types

- enum { `Size = 4` }

### Public Methods

- `AxisAngle()`

*default constructor.*

- [AxisAngle](#) (const [AxisAngle](#) &e)

*copy constructor.*

- [AxisAngle](#) (const DATA\_TYPE &rad\_angle, const DATA\_TYPE &x, const DATA\_TYPE &y, const DATA\_TYPE &z)

*data constructor (angle/x,y,z).*

- [AxisAngle](#) (const DATA\_TYPE &rad\_angle, const [Vec](#)< DATA\_TYPE, 3 > &axis)

*data constructor (angle/Vec3).*

- void [set](#) (const DATA\_TYPE &rad\_angle, const DATA\_TYPE &x, const DATA\_TYPE &y, const DATA\_TYPE &z)

*set raw data.*

- void [set](#) (const DATA\_TYPE &rad\_angle, const [Vec](#)< DATA\_TYPE, 3 > &axis)

*set data.*

- void [setAxis](#) (const [Vec](#)< DATA\_TYPE, 3 > &axis)

*set the axis portion of the [AxisAngle](#).*

- void [setAngle](#) (const DATA\_TYPE &rad\_angle)

*get the angle part of the axisangle.*

- [Vec](#)< DATA\_TYPE, 3 > [getAxis](#) () const

*get the axis portion of the axis angle.*

- const DATA\_TYPE & [getAngle](#) () const

*get the angle part of the axisangle.*

### 10.2.1 Detailed Description

**template<typename DATA\_TYPE> class gmtl::AxisAngle< DATA\_TYPE >**

[AxisAngle](#): Represents a "twist about an axis" [AxisAngle](#) is used to specify a rotation in 3-space.

To some people this rotation format can be more intuitive to specify than [Matrix](#), [Quat](#), or [EulerAngle](#) formatted rotation.

[AxisAngle](#) is very similar to [Quat](#), except it is human readable. For efficiency, you should use [Quat](#) instead ([Quat](#) or [Matrix](#) are preferred).

The internal data format is an array of 4 DATA\_TYPE values. Angle is first, the axis is the last 3.

**Precondition:**

angles are in radians, the axis is usually normalized by the user.

**See also:**

[AxisAnglef](#), [AxisAngled](#), [Matrix](#), [Quat](#), [EulerAngle](#)

Definition at line 64 of file AxisAngle.h.

## 10.2.2 Member Enumeration Documentation

### 10.2.2.1 template<typename DATA\_TYPE> anonymous enum

**Enumeration values:**

**Size**

Definition at line 67 of file AxisAngle.h.

```
67 { Size = 4 };
```

## 10.2.3 Constructor & Destructor Documentation

### 10.2.3.1 template<typename DATA\_TYPE> gmtl::AxisAngle< DATA\_TYPE >::AxisAngle () [inline]

default constructor.

initializes to identity rotation (no rotation).

Definition at line 70 of file AxisAngle.h.

```
70         :
71         VecBase<DATA_TYPE, 4>( (DATA_TYPE)0.0, (DATA_TYPE)1.0,
72                                (DATA_TYPE)0.0, (DATA_TYPE)0.0 )
73     {
74     }
```

### 10.2.3.2 `template<typename DATA_TYPE> gmtl::AxisAngle< DATA_TYPE >::AxisAngle (const AxisAngle< DATA_TYPE > &e) [inline]`

copy constructor.

Definition at line 77 of file AxisAngle.h.

```

77                                     : VecBase<DATA_TYPE, 4>( e )
78     {
79     }
```

### 10.2.3.3 `template<typename DATA_TYPE> gmtl::AxisAngle< DATA_TYPE >::AxisAngle (const DATA_TYPE &rad_angle, const DATA_TYPE &x, const DATA_TYPE &y, const DATA_TYPE &z) [inline]`

data constructor (angle/x,y,z).

angles are in radians.

Definition at line 82 of file AxisAngle.h.

```

83                                     :
84     VecBase<DATA_TYPE, 4>( rad_angle, x, y, z )
85     {
86     }
```

### 10.2.3.4 `template<typename DATA_TYPE> gmtl::AxisAngle< DATA_TYPE >::AxisAngle (const DATA_TYPE &rad_angle, const Vec< DATA_TYPE, 3 > &axis) [inline]`

data constructor (angle/Vec3).

angles are in radians.

Definition at line 89 of file AxisAngle.h.

```

89                                     :
90     VecBase<DATA_TYPE, 4>( rad_angle, axis[0], axis[1], axis[2] )
91     {
92     }
```

## 10.2.4 Member Function Documentation

**10.2.4.1** `template<typename DATA_TYPE> const DATA_TYPE&  
gmtl::AxisAngle< DATA_TYPE >::getAngle () const [inline]`

get the angle part of the axisangle.

**Postcondition:**

returned in radians

Definition at line 137 of file AxisAngle.h.

```
138 {
139     return VecBase<DATA_TYPE, 4>::operator[] ( 0 );
140 }
```

**10.2.4.2** `template<typename DATA_TYPE> Vec<DATA_TYPE, 3>  
gmtl::AxisAngle< DATA_TYPE >::getAxis () const [inline]`

get the axis portion of the axis angle.

**Postcondition:**

returned as a vector, may or may not be normalized.

Definition at line 127 of file AxisAngle.h.

```
128 {
129     return Vec<DATA_TYPE, 3>( VecBase<DATA_TYPE, 4>::operator[] ( 1 ),
130                               VecBase<DATA_TYPE, 4>::operator[] ( 2 ),
131                               VecBase<DATA_TYPE, 4>::operator[] ( 3 ) );
132 }
```

**10.2.4.3** `template<typename DATA_TYPE> void gmtl::AxisAngle<  
DATA_TYPE >::set (const DATA_TYPE & rad_angle, const Vec<  
DATA_TYPE, 3 > & axis) [inline]`

set data.

angles are in radians.

Definition at line 102 of file AxisAngle.h.

References `gmtl::set()`.

```

103     {
104         VecBase<DATA_TYPE, 4>::set( rad_angle, axis[0], axis[1], axis[2] );
105     }
```

**10.2.4.4** `template<typename DATA_TYPE> void gmtl::AxisAngle<DATA_TYPE >::set (const DATA_TYPE & rad_angle, const DATA_TYPE & x, const DATA_TYPE & y, const DATA_TYPE & z)`  
[inline]

set raw data.

angles are in radians.

Reimplemented from `gmtl::VecBase< DATA_TYPE, 4 >`.

Definition at line 95 of file AxisAngle.h.

References `gmtl::set()`.

```

97     {
98         VecBase<DATA_TYPE, 4>::set( rad_angle, x, y, z );
99     }
```

**10.2.4.5** `template<typename DATA_TYPE> void gmtl::AxisAngle<DATA_TYPE >::setAngle (const DATA_TYPE & rad_angle)`  
[inline]

get the angle part of the axisangle.

**Postcondition:**

returned in radians

Definition at line 119 of file AxisAngle.h.

```

120     {
121         VecBase<DATA_TYPE, 4>::operator[] ( 0 ) = rad_angle;
122     }
```



**10.2.4.6** `template<typename DATA_TYPE> void gmtl::AxisAngle<  
DATA_TYPE >::setAxis (const Vec< DATA_TYPE, 3 > & axis)  
[inline]`

set the axis portion of the [AxisAngle](#).

Definition at line 109 of file AxisAngle.h.

```
110     {  
111         VecBase<DATA_TYPE, 4>::operator[] ( 1 ) = axis[0];  
112         VecBase<DATA_TYPE, 4>::operator[] ( 2 ) = axis[1];  
113         VecBase<DATA_TYPE, 4>::operator[] ( 3 ) = axis[2];  
114     }
```

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

- [AxisAngle.h](#)

## 10.3 gmtl::CompareIndexPointProjections Struct Reference

```
#include <Comparitors.h>
```

### Public Methods

- [CompareIndexPointProjections\(\)](#)
- bool [operator\(\)](#) (const unsigned x, const unsigned y)

### Public Attributes

- const std::vector< Point3 > \* [points](#)
- gmtl::Vec3 [sortDir](#)

### 10.3.1 Constructor & Destructor Documentation

#### 10.3.1.1 gmtl::CompareIndexPointProjections::CompareIndexPointProjections() [inline]

Definition at line 54 of file Comparitors.h.

References [points](#).

```
54                                     : points(NULL)
55     { ; }
```

### 10.3.2 Member Function Documentation

#### 10.3.2.1 bool gmtl::CompareIndexPointProjections::operator() (const unsigned x, const unsigned y) [inline]

Definition at line 57 of file Comparitors.h.

References [points](#), and [sortDir](#).

```
58     {  
59         float xVal = sortDir.dot((*points)[x]);  
60         float yVal = sortDir.dot((*points)[y]);  
61  
62         return (xVal < yVal);  
63     }
```

### 10.3.3 Member Data Documentation

#### 10.3.3.1 `const std::vector<Point3>* gmtl::CompareIndexPointProjections::points`

Definition at line 65 of file Comparitors.h.

Referenced by `CompareIndexPointProjections()`, and `operator()()`.

#### 10.3.3.2 `gmtl::Vec3 gmtl::CompareIndexPointProjections::sortDir`

Definition at line 66 of file Comparitors.h.

Referenced by `operator()()`.

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

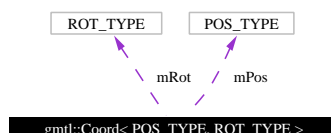
- [Comparitors.h](#)

## 10.4 gmtl::Coord< POS\_TYPE, ROT\_TYPE > Class Template Reference

coord is a position/rotation pair.

```
#include <Coord.h>
```

Collaboration diagram for gmtl::Coord< POS\_TYPE, ROT\_TYPE >:



### Public Types

- typedef POS\_TYPE::DataType [DataType](#)
- typedef POS\_TYPE [PosDataType](#)
- typedef ROT\_TYPE [RotDataType](#)
- enum { [PosSize](#) = POS\_TYPE::Size, [RotSize](#) = ROT\_TYPE::Size }

### Public Methods

- [Coord](#) ()
- [Coord](#) (const Coord< POS\_TYPE, ROT\_TYPE > &coord)
- [Coord](#) (const POS\_TYPE &pos, const ROT\_TYPE &rot)
- const POS\_TYPE & [getPos](#) () const
- const ROT\_TYPE & [getRot](#) () const
- POS\_TYPE & [pos](#) ()

*accessor to the position element.*

- ROT\_TYPE & [rot](#) ()

*accessor to the rotation element.*

## Public Attributes

- POS\_TYPE [mPos](#)  
*const accessor to the rotation element.*
- ROT\_TYPE [mRot](#)

### 10.4.1 Detailed Description

**template<typename POS\_TYPE, typename ROT\_TYPE> class gmtl::Coord< POS\_TYPE, ROT\_TYPE >**

coord is a position/rotation pair.

coord consists of a position element and a rotation element.

**”How to define an [Point](#)/Euler pair (32 bit float precision):”**

```
Coord<float, 3, 3> myEulerCoord;
```

**”Or use the built in typedefs:”**

```
CoordVec3dEuler myEulerCoord;  
CoordVec4fEuler myOtherEulerCoord;
```

**See also:**

[Vec](#), [AxisAngle](#), [EulerAngle](#)

Definition at line 28 of file Coord.h.

### 10.4.2 Member Typedef Documentation

**10.4.2.1 template<typename POS\_TYPE, typename ROT\_TYPE> typedef POS\_TYPE::DataType gmtl::Coord< POS\_TYPE, ROT\_TYPE >::DataType**

Definition at line 35 of file Coord.h.

**10.4.2.2** `template<typename POS_TYPE, typename ROT_TYPE> typedef  
POS_TYPE gmtl::Coord< POS_TYPE, ROT_TYPE >::PosDataType`

Definition at line 36 of file Coord.h.

**10.4.2.3** `template<typename POS_TYPE, typename ROT_TYPE> typedef  
ROT_TYPE gmtl::Coord< POS_TYPE, ROT_TYPE >::RotDataType`

Definition at line 37 of file Coord.h.

### 10.4.3 Member Enumeration Documentation

**10.4.3.1** `template<typename POS_TYPE, typename ROT_TYPE> anonymous  
enum`

Enumeration values:

**PosSize**

**RotSize**

Definition at line 38 of file Coord.h.

```
39     {
40         PosSize = POS_TYPE::Size,
41         RotSize = ROT_TYPE::Size
42     };
```

### 10.4.4 Constructor & Destructor Documentation

**10.4.4.1** `template<typename POS_TYPE, typename ROT_TYPE>  
gmtl::Coord< POS_TYPE, ROT_TYPE >::Coord() [inline]`

Definition at line 31 of file Coord.h.

References `gmtl::Coord< POS_TYPE, ROT_TYPE >::mPos`, and `gmtl::Coord< POS_TYPE, ROT_TYPE >::mRot`.

```
31         : mPos(), mRot()
32     {
33     }
```

**10.4.4.2** `template<typename POS_TYPE, typename ROT_TYPE>  
gmtl::Coord< POS_TYPE, ROT_TYPE >::Coord (const Coord<  
POS_TYPE, ROT_TYPE > & coord) [inline]`

Definition at line 44 of file Coord.h.

References gmtl::Coord< POS\_TYPE, ROT\_TYPE >::mPos, and gmtl::Coord< POS\_TYPE, ROT\_TYPE >::mRot.

```
44                                     : mPos( coord.mPos ), mRot( coord.mRot )
45     {
46     }
```

**10.4.4.3** `template<typename POS_TYPE, typename ROT_TYPE>  
gmtl::Coord< POS_TYPE, ROT_TYPE >::Coord (const POS_TYPE  
& pos, const ROT_TYPE & rot) [inline]`

Definition at line 48 of file Coord.h.

References gmtl::Coord< POS\_TYPE, ROT\_TYPE >::mPos, gmtl::Coord< POS\_TYPE, ROT\_TYPE >::mRot, gmtl::Coord< POS\_TYPE, ROT\_TYPE >::pos(), and gmtl::Coord< POS\_TYPE, ROT\_TYPE >::rot().

```
48                                     : mPos( pos ), mRot( rot )
49     {
50     }
```

## 10.4.5 Member Function Documentation

**10.4.5.1** `template<typename POS_TYPE, typename ROT_TYPE> const  
POS_TYPE& gmtl::Coord< POS_TYPE, ROT_TYPE >::getPos ()  
const [inline]`

Definition at line 52 of file Coord.h.

References gmtl::Coord< POS\_TYPE, ROT\_TYPE >::mPos.

```
52 { return mPos; }
```

**10.4.5.2** `template<typename POS_TYPE, typename ROT_TYPE> const  
ROT_TYPE& gmtl::Coord< POS_TYPE, ROT_TYPE >::getRot ()  
const [inline]`

Definition at line 53 of file Coord.h.

References gmtl::Coord< POS\_TYPE, ROT\_TYPE >::mRot.

```
53 { return mRot; }
```

**10.4.5.3** `template<typename POS_TYPE, typename ROT_TYPE>  
POS_TYPE& gmtl::Coord< POS_TYPE, ROT_TYPE >::pos ()  
[inline]`

accessor to the position element.

Definition at line 59 of file Coord.h.

References gmtl::Coord< POS\_TYPE, ROT\_TYPE >::mPos.

Referenced by gmtl::Coord< POS\_TYPE, ROT\_TYPE >::Coord().

```
59 { return mPos; }
```

**10.4.5.4** `template<typename POS_TYPE, typename ROT_TYPE>  
ROT_TYPE& gmtl::Coord< POS_TYPE, ROT_TYPE >::rot ()  
[inline]`

accessor to the rotation element.

Definition at line 62 of file Coord.h.

References gmtl::Coord< POS\_TYPE, ROT\_TYPE >::mRot.

Referenced by gmtl::Coord< POS\_TYPE, ROT\_TYPE >::Coord().

```
62 { return mRot; }
```

## 10.4.6 Member Data Documentation



**10.4.6.1    template<typename POS\_TYPE, typename ROT\_TYPE> POS\_TYPE  
gmtl::Coord< POS\_TYPE, ROT\_TYPE >::mPos**

const accessor to the rotation element.

Definition at line 71 of file Coord.h.

Referenced by gmtl::Coord< POS\_TYPE, ROT\_TYPE >::Coord(), gmtl::Coord< POS\_TYPE, ROT\_TYPE >::getPos(), and gmtl::Coord< POS\_TYPE, ROT\_TYPE >::pos().

**10.4.6.2    template<typename POS\_TYPE, typename ROT\_TYPE> ROT\_TYPE  
gmtl::Coord< POS\_TYPE, ROT\_TYPE >::mRot**

Definition at line 72 of file Coord.h.

Referenced by gmtl::Coord< POS\_TYPE, ROT\_TYPE >::Coord(), gmtl::Coord< POS\_TYPE, ROT\_TYPE >::getRot(), and gmtl::Coord< POS\_TYPE, ROT\_TYPE >::rot().

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

- [Coord.h](#)

## 10.5 gmtl::Eigen Class Reference

```
#include <Eigen.h>
```

### Public Methods

- [Eigen](#) (int iSize)
- [~Eigen](#) ()
- float & [Matrix](#) (int iRow, int iCol)
- void [SetMatrix](#) (float \*\*aafMat)
- float [GetEigenvalue](#) (int i) const
- float [GetEigenvector](#) (int iRow, int iCol) const
- float \* [GetEigenvalue](#) ()
- float \*\* [GetEigenvector](#) ()
- void [EigenStuff2](#) ()
- void [EigenStuff3](#) ()
- void [EigenStuff4](#) ()
- void [EigenStuffN](#) ()
- void [EigenStuff](#) ()
- void [DecrSortEigenStuff2](#) ()
- void [DecrSortEigenStuff3](#) ()
- void [DecrSortEigenStuff4](#) ()
- void [DecrSortEigenStuffN](#) ()
- void [DecrSortEigenStuff](#) ()
- void [IncrSortEigenStuff2](#) ()
- void [IncrSortEigenStuff3](#) ()
- void [IncrSortEigenStuff4](#) ()
- void [IncrSortEigenStuffN](#) ()
- void [IncrSortEigenStuff](#) ()

### Static Protected Methods

- void [Tridiagonal2](#) (float \*\*aafMat, float \*afDiag, float \*afSubd)
- void [Tridiagonal3](#) (float \*\*aafMat, float \*afDiag, float \*afSubd)
- void [Tridiagonal4](#) (float \*\*aafMat, float \*afDiag, float \*afSubd)
- void [TridiagonalN](#) (int iSize, float \*\*aafMat, float \*afDiag, float \*afSubd)
- bool [QLAlgorithm](#) (int iSize, float \*afDiag, float \*afSubd, float \*\*aafMat)
- void [DecreasingSort](#) (int iSize, float \*afEigval, float \*\*aafEigvec)
- void [IncreasingSort](#) (int iSize, float \*afEigval, float \*\*aafEigvec)

## Protected Attributes

- int [m\\_iSize](#)
- float \*\* [m\\_aafMat](#)
- float \* [m\\_afDiag](#)
- float \* [m\\_afSubd](#)

### 10.5.1 Constructor & Destructor Documentation

#### 10.5.1.1 gmtl::Eigen::Eigen (int *iSize*)

Definition at line 144 of file Eigen.h.

References [m\\_aafMat](#), [m\\_afDiag](#), [m\\_afSubd](#), and [m\\_iSize](#).

```
145 {  
146     assert( iSize >= 2 );  
147     m_iSize = iSize;  
148  
149     m_aafMat = new float*[m_iSize];  
150     for (int i = 0; i < m_iSize; i++)  
151         m_aafMat[i] = new float[m_iSize];  
152  
153     m_afDiag = new float[m_iSize];  
154     m_afSubd = new float[m_iSize];  
155 }
```

#### 10.5.1.2 gmtl::Eigen::~~Eigen ()

Definition at line 157 of file Eigen.h.

References [m\\_aafMat](#), [m\\_afDiag](#), [m\\_afSubd](#), and [m\\_iSize](#).

```
158 {  
159     delete[] m_afSubd;  
160     delete[] m_afDiag;  
161     for (int i = 0; i < m_iSize; i++)  
162         delete[] m_aafMat[i];  
163     delete[] m_aafMat;  
164 }
```

## 10.5.2 Member Function Documentation

### 10.5.2.1 void gmtl::Eigen::DecreasingSort (int *iSize*, float \* *afEigval*, float \*\* *aafEigvec*) [static, protected]

Definition at line 551 of file Eigen.h.

Referenced by DecrSortEigenStuff(), DecrSortEigenStuff2(), DecrSortEigenStuff3(), DecrSortEigenStuff4(), and DecrSortEigenStuffN().

```

553 {
554     // sort eigenvalues in decreasing order, e[0] >= ... >= e[iSize-1]
555     for (int i0 = 0, i1; i0 <= iSize-2; i0++)
556     {
557         // locate maximum eigenvalue
558         i1 = i0;
559         float fMax = afEigval[i1];
560         int i2;
561         for (i2 = i0+1; i2 < iSize; i2++)
562         {
563             if ( afEigval[i2] > fMax )
564             {
565                 i1 = i2;
566                 fMax = afEigval[i1];
567             }
568         }
569         if ( i1 != i0 )
570         {
571             // swap eigenvalues
572             afEigval[i1] = afEigval[i0];
573             afEigval[i0] = fMax;
574             // swap eigenvectors
575             for (i2 = 0; i2 < iSize; i2++)
576             {
577                 float fTmp = aafEigvec[i2][i0];
578                 aafEigvec[i2][i0] = aafEigvec[i2][i1];
579                 aafEigvec[i2][i1] = fTmp;
580             }
581         }
582     }
583 }
584 }
585 }
```

### 10.5.2.2 void gmtl::Eigen::DecrSortEigenStuff ()

Definition at line 704 of file Eigen.h.

References DecreasingSort(), m\_aafMat, m\_afDiag, m\_afSubd, m\_iSize, QLAlgorithm(), Tridiagonal2(), Tridiagonal3(), Tridiagonal4(), and TridiagonalN().

```

705 {
706     switch ( m_iSize )
707     {
708         case 2:
709             Tridiagonal2(m_aafMat,m_afDiag,m_afSubd);
710             break;
711         case 3:
712             Tridiagonal3(m_aafMat,m_afDiag,m_afSubd);
713             break;
714         case 4:
715             Tridiagonal4(m_aafMat,m_afDiag,m_afSubd);
716             break;
717         default:
718             TridiagonalN(m_iSize,m_aafMat,m_afDiag,m_afSubd);
719             break;
720     }
721     QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
722     DecreasingSort(m_iSize,m_afDiag,m_aafMat);
723 }
```

#### 10.5.2.3 void gmtl::Eigen::DecrSortEigenStuff2 ()

Definition at line 676 of file Eigen.h.

References DecreasingSort(), m\_aafMat, m\_afDiag, m\_afSubd, m\_iSize, QLAlgorithm(), and Tridiagonal2().

```

677 {
678     Tridiagonal2(m_aafMat,m_afDiag,m_afSubd);
679     QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
680     DecreasingSort(m_iSize,m_afDiag,m_aafMat);
681 }
```

#### 10.5.2.4 void gmtl::Eigen::DecrSortEigenStuff3 ()

Definition at line 683 of file Eigen.h.

References DecreasingSort(), m\_aafMat, m\_afDiag, m\_afSubd, m\_iSize, QLAlgorithm(), and Tridiagonal3().

```

684 {
685     Tridiagonal3(m_aafMat,m_afDiag,m_afSubd);
```

```

686     QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
687     DecreasingSort(m_iSize,m_afDiag,m_aafMat);
688 }

```

#### 10.5.2.5 void gmtl::Eigen::DecrSortEigenStuff4 ()

Definition at line 690 of file Eigen.h.

References [DecreasingSort\(\)](#), [m\\_aafMat](#), [m\\_afDiag](#), [m\\_afSubd](#), [m\\_iSize](#), [QLAlgorithm\(\)](#), and [Tridiagonal4\(\)](#).

```

691 {
692     Tridiagonal4(m_aafMat,m_afDiag,m_afSubd);
693     QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
694     DecreasingSort(m_iSize,m_afDiag,m_aafMat);
695 }

```

#### 10.5.2.6 void gmtl::Eigen::DecrSortEigenStuffN ()

Definition at line 697 of file Eigen.h.

References [DecreasingSort\(\)](#), [m\\_aafMat](#), [m\\_afDiag](#), [m\\_afSubd](#), [m\\_iSize](#), [QLAlgorithm\(\)](#), and [TridiagonalN\(\)](#).

```

698 {
699     TridiagonalN(m_iSize,m_aafMat,m_afDiag,m_afSubd);
700     QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
701     DecreasingSort(m_iSize,m_afDiag,m_aafMat);
702 }

```

#### 10.5.2.7 void gmtl::Eigen::EigenStuff ()

Definition at line 656 of file Eigen.h.

References [m\\_aafMat](#), [m\\_afDiag](#), [m\\_afSubd](#), [m\\_iSize](#), [QLAlgorithm\(\)](#), [Tridiagonal2\(\)](#), [Tridiagonal3\(\)](#), [Tridiagonal4\(\)](#), and [TridiagonalN\(\)](#).

```

657 {
658     switch ( m_iSize )
659     {

```

```

660         case 2:
661             Tridiagonal2(m_aafMat,m_afDiag,m_afSubd);
662             break;
663         case 3:
664             Tridiagonal3(m_aafMat,m_afDiag,m_afSubd);
665             break;
666         case 4:
667             Tridiagonal4(m_aafMat,m_afDiag,m_afSubd);
668             break;
669         default:
670             TridiagonalN(m_iSize,m_aafMat,m_afDiag,m_afSubd);
671             break;
672     }
673     QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
674 }

```

#### 10.5.2.8 void gmtl::Eigen::EigenStuff2 ()

Definition at line 632 of file Eigen.h.

References `m_aafMat`, `m_afDiag`, `m_afSubd`, `m_iSize`, `QLAlgorithm()`, and `Tridiagonal2()`.

```

633 {
634     Tridiagonal2(m_aafMat,m_afDiag,m_afSubd);
635     QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
636 }

```

#### 10.5.2.9 void gmtl::Eigen::EigenStuff3 ()

Definition at line 638 of file Eigen.h.

References `m_aafMat`, `m_afDiag`, `m_afSubd`, `m_iSize`, `QLAlgorithm()`, and `Tridiagonal3()`.

```

639 {
640     Tridiagonal3(m_aafMat,m_afDiag,m_afSubd);
641     QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
642 }

```

**10.5.2.10 void gmtl::Eigen::EigenStuff4 ()**

Definition at line 644 of file Eigen.h.

References `m_aafMat`, `m_afDiag`, `m_afSubd`, `m_iSize`, `QLAlgorithm()`, and `Tridiagonal4()`.

```
645 {
646     Tridiagonal4(m_aafMat,m_afDiag,m_afSubd);
647     QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
648 }
```

**10.5.2.11 void gmtl::Eigen::EigenStuffN ()**

Definition at line 650 of file Eigen.h.

References `m_aafMat`, `m_afDiag`, `m_afSubd`, `m_iSize`, `QLAlgorithm()`, and `TridiagonalN()`.

```
651 {
652     TridiagonalN(m_iSize,m_aafMat,m_afDiag,m_afSubd);
653     QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
654 }
```

**10.5.2.12 float \* gmtl::Eigen::GetEigenvalue () [inline]**

Definition at line 128 of file Eigen.h.

References `m_afDiag`.

```
129 {
130     return m_afDiag;
131 }
```

**10.5.2.13 float gmtl::Eigen::GetEigenvalue (int *i*) const [inline]**

Definition at line 118 of file Eigen.h.

References `m_afDiag`.

Referenced by `gmtl::GaussPointsFit()`.



```

119 {
120     return m_afDiag[i];
121 }

```

#### 10.5.2.14 float\*\* gmtl::Eigen::GetEigenvector() [inline]

Definition at line 133 of file Eigen.h.

References m\_aafMat.

```

134 {
135     return m_aafMat;
136 }

```

#### 10.5.2.15 float gmtl::Eigen::GetEigenvector(int iRow, int iCol) const [inline]

Definition at line 123 of file Eigen.h.

References m\_aafMat.

Referenced by gmtl::GaussPointsFit().

```

124 {
125     return m_aafMat[iRow][iCol];
126 }

```

#### 10.5.2.16 void gmtl::Eigen::IncreasingSort(int iSize, float \* afEigval, float \*\* aafEigvec) [static, protected]

Definition at line 587 of file Eigen.h.

Referenced by IncrSortEigenStuff(), IncrSortEigenStuff2(), IncrSortEigenStuff3(), IncrSortEigenStuff4(), and IncrSortEigenStuffN().

```

589 {
590     // sort eigenvalues in increasing order, e[0] <= ... <= e[iSize-1]
591     for (int i0 = 0, i1; i0 <= iSize-2; i0++)
592     {
593         // locate minimum eigenvalue

```

```

594         i1 = i0;
595         float fMin = afEigval[i1];
596         int i2;
597         for (i2 = i0+1; i2 < iSize; i2++)
598         {
599             if ( afEigval[i2] < fMin )
600             {
601                 i1 = i2;
602                 fMin = afEigval[i1];
603             }
604         }
605
606         if ( i1 != i0 )
607         {
608             // swap eigenvalues
609             afEigval[i1] = afEigval[i0];
610             afEigval[i0] = fMin;
611
612             // swap eigenvectors
613             for (i2 = 0; i2 < iSize; i2++)
614             {
615                 float fTmp = aafEigvec[i2][i0];
616                 aafEigvec[i2][i0] = aafEigvec[i2][i1];
617                 aafEigvec[i2][i1] = fTmp;
618             }
619         }
620     }
621 }

```

### 10.5.2.17 void gmtl::Eigen::IncrSortEigenStuff ()

Definition at line 753 of file Eigen.h.

References [IncreasingSort\(\)](#), [m\\_aafMat](#), [m\\_afDiag](#), [m\\_afSubd](#), [m\\_iSize](#), [QLAlgorithm\(\)](#), [Tridiagonal2\(\)](#), [Tridiagonal3\(\)](#), [Tridiagonal4\(\)](#), and [TridiagonalN\(\)](#).

```

754 {
755     switch ( m_iSize )
756     {
757         case 2:
758             Tridiagonal2(m_aafMat,m_afDiag,m_afSubd);
759             break;
760         case 3:
761             Tridiagonal3(m_aafMat,m_afDiag,m_afSubd);
762             break;
763         case 4:
764             Tridiagonal4(m_aafMat,m_afDiag,m_afSubd);
765             break;
766         default:
767             TridiagonalN(m_iSize,m_aafMat,m_afDiag,m_afSubd);

```

```

768         break;
769     }
770     QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
771     IncreasingSort(m_iSize,m_afDiag,m_aafMat);
772 }

```

#### 10.5.2.18 void gmtl::Eigen::IncrSortEigenStuff2 ()

Definition at line 725 of file Eigen.h.

References [IncreasingSort\(\)](#), [m\\_aafMat](#), [m\\_afDiag](#), [m\\_afSubd](#), [m\\_iSize](#), [QLAlgorithm\(\)](#), and [Tridiagonal2\(\)](#).

```

726 {
727     Tridiagonal2(m_aafMat,m_afDiag,m_afSubd);
728     QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
729     IncreasingSort(m_iSize,m_afDiag,m_aafMat);
730 }

```

#### 10.5.2.19 void gmtl::Eigen::IncrSortEigenStuff3 ()

Definition at line 732 of file Eigen.h.

References [IncreasingSort\(\)](#), [m\\_aafMat](#), [m\\_afDiag](#), [m\\_afSubd](#), [m\\_iSize](#), [QLAlgorithm\(\)](#), and [Tridiagonal3\(\)](#).

Referenced by [gmtl::GaussPointsFit\(\)](#).

```

733 {
734     Tridiagonal3(m_aafMat,m_afDiag,m_afSubd);
735     QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
736     IncreasingSort(m_iSize,m_afDiag,m_aafMat);
737 }

```

#### 10.5.2.20 void gmtl::Eigen::IncrSortEigenStuff4 ()

Definition at line 739 of file Eigen.h.

References [IncreasingSort\(\)](#), [m\\_aafMat](#), [m\\_afDiag](#), [m\\_afSubd](#), [m\\_iSize](#), [QLAlgorithm\(\)](#), and [Tridiagonal4\(\)](#).

```

740 {
741     Tridiagonal4(m_aafMat,m_afDiag,m_afSubd);
742     QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
743     IncreasingSort(m_iSize,m_afDiag,m_aafMat);
744 }

```

#### 10.5.2.21 void gmtl::Eigen::IncrSortEigenStuffN ()

Definition at line 746 of file Eigen.h.

References `IncreasingSort()`, `m_aafMat`, `m_afDiag`, `m_afSubd`, `m_iSize`, `QLAlgorithm()`, and `TridiagonalN()`.

```

747 {
748     TridiagonalN(m_iSize,m_aafMat,m_afDiag,m_afSubd);
749     QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
750     IncreasingSort(m_iSize,m_afDiag,m_aafMat);
751 }

```

#### 10.5.2.22 float & gmtl::Eigen::Matrix (int iRow, int iCol) [inline]

Definition at line 113 of file Eigen.h.

References `m_aafMat`.

Referenced by `gmtl::GaussPointsFit()`.

```

114 {
115     return m_aafMat[iRow][iCol];
116 }

```

#### 10.5.2.23 bool gmtl::Eigen::QLAlgorithm (int iSize, float \* afDiag, float \* afSubd, float \*\* aafMat) [static, protected]

Definition at line 479 of file Eigen.h.

References `gmtl::Math::abs()`, `m_aafMat`, `m_afDiag`, `m_afSubd`, and `gmtl::Math::sqrt()`.

Referenced by `DecrSortEigenStuff()`, `DecrSortEigenStuff2()`, `DecrSortEigenStuff3()`, `DecrSortEigenStuff4()`, `DecrSortEigenStuffN()`, `EigenStuff()`, `EigenStuff2()`, `EigenStuff3()`, `EigenStuff4()`, `EigenStuffN()`, `IncrSortEigenStuff()`, `IncrSortEigenStuff2()`, `IncrSortEigenStuff3()`, `IncrSortEigenStuff4()`, and `IncrSortEigenStuffN()`.

```

481 {
482     const int iMaxIter = 32;
483
484     for (int i0 = 0; i0 < iSize; i0++)
485     {
486         int i1;
487         for (i1 = 0; i1 < iMaxIter; i1++)
488         {
489             int i2;
490             for (i2 = i0; i2 <= iSize-2; i2++)
491             {
492                 float fTmp =
493                     Math::abs(m_afDiag[i2])+ Math::abs(m_afDiag[i2+1]);
494                 if ( Math::abs(m_afSubd[i2]) + fTmp == fTmp )
495                     break;
496             }
497             if ( i2 == i0 )
498                 break;
499
500             float fG = (m_afDiag[i0+1]-m_afDiag[i0])/(2.0*m_afSubd[i0]);
501             float fR = Math::sqrt(fG*fG+1.0);
502             if ( fG < 0.0 )
503                 fG = m_afDiag[i2]-m_afDiag[i0]+m_afSubd[i0]/(fG-fR);
504             else
505                 fG = m_afDiag[i2]-m_afDiag[i0]+m_afSubd[i0]/(fG+fR);
506             float fSin = 1.0, fCos = 1.0, fP = 0.0;
507             for (int i3 = i2-1; i3 >= i0; i3--)
508             {
509                 float fF = fSin*m_afSubd[i3];
510                 float fB = fCos*m_afSubd[i3];
511                 if ( Math::abs(fF) >= Math::abs(fG) )
512                 {
513                     fCos = fG/fF;
514                     fR = sqrt(fCos*fCos+1.0);
515                     m_afSubd[i3+1] = fF*fR;
516                     fSin = 1.0/fR;
517                     fCos *= fSin;
518                 }
519                 else
520                 {
521                     fSin = fF/fG;
522                     fR = Math::sqrt(fSin*fSin+1.0);
523                     m_afSubd[i3+1] = fG*fR;
524                     fCos = 1.0/fR;
525                     fSin *= fCos;
526                 }
527                 fG = m_afDiag[i3+1]-fP;
528                 fR = (m_afDiag[i3]-fG)*fSin+2.0*fB*fCos;
529                 fP = fSin*fR;
530                 m_afDiag[i3+1] = fG+fP;
531                 fG = fCos*fR-fB;
532
533                 for (int i4 = 0; i4 < iSize; i4++)
534                 {
535                     fF = m_aafMat[i4][i3+1];

```

```

536             m_aafMat[i4][i3+1] = fSin*m_aafMat[i4][i3]+fCos*fF;
537             m_aafMat[i4][i3] = fCos*m_aafMat[i4][i3]-fSin*fF;
538         }
539     }
540     m_afDiag[i0] -= fP;
541     m_afSubd[i0] = fG;
542     m_afSubd[i2] = 0.0;
543 }
544 if ( i1 == iMaxIter )
545     return false;
546 }
547
548 return true;
549 }
```

#### 10.5.2.24 void gmtl::Eigen::SetMatrix (float \*\* *aafMat*)

Definition at line 623 of file Eigen.h.

References `m_aafMat`, and `m_iSize`.

```

624 {
625     for (int iRow = 0; iRow < m_iSize; iRow++)
626     {
627         for (int iCol = 0; iCol < m_iSize; iCol++)
628             m_aafMat[iRow][iCol] = aafMat[iRow][iCol];
629     }
630 }
```

#### 10.5.2.25 void gmtl::Eigen::Tridiagonal2 (float \*\* *aafMat*, float \* *afDiag*, float \* *afSubd*) [static, protected]

Definition at line 166 of file Eigen.h.

References `m_aafMat`, `m_afDiag`, and `m_afSubd`.

Referenced by `DecrSortEigenStuff()`, `DecrSortEigenStuff2()`, `EigenStuff()`, `EigenStuff2()`, `IncrSortEigenStuff()`, and `IncrSortEigenStuff2()`.

```

168 {
169     // matrix is already tridiagonal
170     m_afDiag[0] = m_aafMat[0][0];
171     m_afDiag[1] = m_aafMat[1][1];
172     m_afSubd[0] = m_aafMat[0][1];
173     m_afSubd[1] = 0.0;
174     m_aafMat[0][0] = 1.0;
```

```

175     m_aafMat[0][1] = 0.0;
176     m_aafMat[1][0] = 0.0;
177     m_aafMat[1][1] = 1.0;
178 }

```

#### 10.5.2.26 void gmtl::Eigen::Tridiagonal3 (float \*\* aafMat, float \* afDiag, float \* afSubd) [static, protected]

Definition at line 180 of file Eigen.h.

References `m_aafMat`, `m_afDiag`, `m_afSubd`, and `gmtl::Math::sqrt()`.

Referenced by `DecrSortEigenStuff()`, `DecrSortEigenStuff3()`, `EigenStuff()`, `EigenStuff3()`, `IncrSortEigenStuff()`, and `IncrSortEigenStuff3()`.

```

182 {
183     float fM00 = m_aafMat[0][0];
184     float fM01 = m_aafMat[0][1];
185     float fM02 = m_aafMat[0][2];
186     float fM11 = m_aafMat[1][1];
187     float fM12 = m_aafMat[1][2];
188     float fM22 = m_aafMat[2][2];
189
190     m_afDiag[0] = fM00;
191     m_afSubd[2] = 0.0;
192     if ( fM02 != 0.0 )
193     {
194         float fLength = Math::sqrt(fM01*fM01+fM02*fM02);
195         float fInvLength = 1.0/fLength;
196         fM01 *= fInvLength;
197         fM02 *= fInvLength;
198         float fQ = 2.0*fM01*fM12+fM02*(fM22-fM11);
199         m_afDiag[1] = fM11+fM02*fQ;
200         m_afDiag[2] = fM22-fM02*fQ;
201         m_afSubd[0] = fLength;
202         m_afSubd[1] = fM12-fM01*fQ;
203         m_aafMat[0][0] = 1.0; m_aafMat[0][1] = 0.0; m_aafMat[0][2] = 0.0;
204         m_aafMat[1][0] = 0.0; m_aafMat[1][1] = fM01; m_aafMat[1][2] = fM02;
205         m_aafMat[2][0] = 0.0; m_aafMat[2][1] = fM02; m_aafMat[2][2] = -fM01;
206     }
207     else
208     {
209         m_afDiag[1] = fM11;
210         m_afDiag[2] = fM22;
211         m_afSubd[0] = fM01;
212         m_afSubd[1] = fM12;
213         m_aafMat[0][0] = 1.0; m_aafMat[0][1] = 0.0; m_aafMat[0][2] = 0.0;
214         m_aafMat[1][0] = 0.0; m_aafMat[1][1] = 1.0; m_aafMat[1][2] = 0.0;
215         m_aafMat[2][0] = 0.0; m_aafMat[2][1] = 0.0; m_aafMat[2][2] = 1.0;
216     }
217 }

```

### 10.5.2.27 void gmtl::Eigen::Tridiagonal4 (float \*\* *aafMat*, float \* *afDiag*, float \* *afSubd*) [static, protected]

Definition at line 219 of file Eigen.h.

References `m_aafMat`, `m_afDiag`, `m_afSubd`, and `gmtl::Math::sqrt()`.

Referenced by `DecrSortEigenStuff()`, `DecrSortEigenStuff4()`, `EigenStuff()`, `EigenStuff4()`, `IncrSortEigenStuff()`, and `IncrSortEigenStuff4()`.

```

221 {
222     // save matrix M
223     float fM00 = m_aafMat[0][0];
224     float fM01 = m_aafMat[0][1];
225     float fM02 = m_aafMat[0][2];
226     float fM03 = m_aafMat[0][3];
227     float fM11 = m_aafMat[1][1];
228     float fM12 = m_aafMat[1][2];
229     float fM13 = m_aafMat[1][3];
230     float fM22 = m_aafMat[2][2];
231     float fM23 = m_aafMat[2][3];
232     float fM33 = m_aafMat[3][3];
233
234     m_afDiag[0] = fM00;
235     m_afSubd[3] = 0.0;
236
237     m_aafMat[0][0] = 1.0;
238     m_aafMat[0][1] = 0.0;
239     m_aafMat[0][2] = 0.0;
240     m_aafMat[0][3] = 0.0;
241     m_aafMat[1][0] = 0.0;
242     m_aafMat[2][0] = 0.0;
243     m_aafMat[3][0] = 0.0;
244
245     float fLength, fInvLength;
246
247     if ( fM02 != 0.0 || fM03 != 0.0 )
248     {
249         float fQ11, fQ12, fQ13;
250         float fQ21, fQ22, fQ23;
251         float fQ31, fQ32, fQ33;
252
253         // build column Q1
254         fLength = Math::sqrt(fM01*fM01 + fM02*fM02 + fM03*fM03);
255         fInvLength = 1.0/fLength;
256         fQ11 = fM01*fInvLength;
257         fQ21 = fM02*fInvLength;
258         fQ31 = fM03*fInvLength;
259
260         m_afSubd[0] = fLength;
261

```



```

262         // compute S*Q1
263         float fV0 = fM11*fQ11+fM12*fQ21+fM13*fQ31;
264         float fV1 = fM12*fQ11+fM22*fQ21+fM23*fQ31;
265         float fV2 = fM13*fQ11+fM23*fQ21+fM33*fQ31;
266
267         m_afDiag[1] = fQ11*fV0+fQ21*fV1+fQ31*fV2;
268
269         // build column Q3 = Q1x(S*Q1)
270         fQ13 = fQ21*fV2-fQ31*fV1;
271         fQ23 = fQ31*fV0-fQ11*fV2;
272         fQ33 = fQ11*fV1-fQ21*fV0;
273         fLength = Math::sqrt(fQ13*fQ13+fQ23*fQ23+fQ33*fQ33);
274         if ( fLength > 0.0 )
275         {
276             fInvLength = 1.0/fLength;
277             fQ13 *= fInvLength;
278             fQ23 *= fInvLength;
279             fQ33 *= fInvLength;
280
281             // build column Q2 = Q3xQ1
282             fQ12 = fQ23*fQ31-fQ33*fQ21;
283             fQ22 = fQ33*fQ11-fQ13*fQ31;
284             fQ32 = fQ13*fQ21-fQ23*fQ11;
285
286             fV0 = fQ12*fM11+fQ22*fM12+fQ32*fM13;
287             fV1 = fQ12*fM12+fQ22*fM22+fQ32*fM23;
288             fV2 = fQ12*fM13+fQ22*fM23+fQ32*fM33;
289             m_afSubd[1] = fQ11*fV0+fQ21*fV1+fQ31*fV2;
290             m_afDiag[2] = fQ12*fV0+fQ22*fV1+fQ32*fV2;
291             m_afSubd[2] = fQ13*fV0+fQ23*fV1+fQ33*fV2;
292
293             fV0 = fQ13*fM11+fQ23*fM12+fQ33*fM13;
294             fV1 = fQ13*fM12+fQ23*fM22+fQ33*fM23;
295             fV2 = fQ13*fM13+fQ23*fM23+fQ33*fM33;
296             m_afDiag[3] = fQ13*fV0+fQ23*fV1+fQ33*fV2;
297         }
298         else
299         {
300             // S*Q1 parallel to Q1, choose any valid Q2 and Q3
301             m_afSubd[1] = 0;
302
303             fLength = fQ21*fQ21+fQ31*fQ31;
304             if ( fLength > 0.0 )
305             {
306                 fInvLength = 1.0/fLength;
307                 float fTmp = fQ11-1.0;
308                 fQ12 = -fQ21;
309                 fQ22 = 1.0+fTmp*fQ21*fQ21*fInvLength;
310                 fQ32 = fTmp*fQ21*fQ31*fInvLength;
311
312                 fQ13 = -fQ31;
313                 fQ23 = fQ32;
314                 fQ33 = 1.0+fTmp*fQ31*fQ31*fInvLength;
315
316                 fV0 = fQ12*fM11+fQ22*fM12+fQ32*fM13;

```

```

317         fV1 = fQ12*fM12+fQ22*fM22+fQ32*fM23;
318         fV2 = fQ12*fM13+fQ22*fM23+fQ32*fM33;
319         m_afDiag[2] = fQ12*fV0+fQ22*fV1+fQ32*fV2;
320         m_afSubd[2] = fQ13*fV0+fQ23*fV1+fQ33*fV2;
321
322         fV0 = fQ13*fM11+fQ23*fM12+fQ33*fM13;
323         fV1 = fQ13*fM12+fQ23*fM22+fQ33*fM23;
324         fV2 = fQ13*fM13+fQ23*fM23+fQ33*fM33;
325         m_afDiag[3] = fQ13*fV0+fQ23*fV1+fQ33*fV2;
326     }
327     else
328     {
329         // Q1 = (+-1,0,0)
330         fQ12 = 0.0; fQ22 = 1.0; fQ32 = 0.0;
331         fQ13 = 0.0; fQ23 = 0.0; fQ33 = 1.0;
332
333         m_afDiag[2] = fM22;
334         m_afDiag[3] = fM33;
335         m_afSubd[2] = fM23;
336     }
337 }
338
339 m_aafMat[1][1] = fQ11; m_aafMat[1][2] = fQ12; m_aafMat[1][3] = fQ13;
340 m_aafMat[2][1] = fQ21; m_aafMat[2][2] = fQ22; m_aafMat[2][3] = fQ23;
341 m_aafMat[3][1] = fQ31; m_aafMat[3][2] = fQ32; m_aafMat[3][3] = fQ33;
342 }
343 else
344 {
345     m_afDiag[1] = fM11;
346     m_afSubd[0] = fM01;
347     m_aafMat[1][1] = 1.0;
348     m_aafMat[2][1] = 0.0;
349     m_aafMat[3][1] = 0.0;
350
351     if ( fM13 != 0.0 )
352     {
353         fLength = Math::sqrt(fM12*fM12+fM13*fM13);
354         fInvLength = 1.0/fLength;
355         fM12 *= fInvLength;
356         fM13 *= fInvLength;
357         float fQ = 2.0*fM12*fM23+fM13*(fM33-fM22);
358
359         m_afDiag[2] = fM22+fM13*fQ;
360         m_afDiag[3] = fM33-fM13*fQ;
361         m_afSubd[1] = fLength;
362         m_afSubd[2] = fM23-fM12*fQ;
363         m_aafMat[1][2] = 0.0;
364         m_aafMat[1][3] = 0.0;
365         m_aafMat[2][2] = fM12;
366         m_aafMat[2][3] = fM13;
367         m_aafMat[3][2] = fM13;
368         m_aafMat[3][3] = -fM12;
369     }
370     else
371     {

```

```

372         m_afDiag[2] = fM22;
373         m_afDiag[3] = fM33;
374         m_afSubd[1] = fM12;
375         m_afSubd[2] = fM23;
376         m_aafMat[1][2] = 0.0;
377         m_aafMat[1][3] = 0.0;
378         m_aafMat[2][2] = 1.0;
379         m_aafMat[2][3] = 0.0;
380         m_aafMat[3][2] = 0.0;
381         m_aafMat[3][3] = 1.0;
382     }
383 }
384 }

```

#### 10.5.2.28 void gmtl::Eigen::TridiagonalN (int iSize, float \*\* aafMat, float \* afDiag, float \* afSubd) [static, protected]

Definition at line 386 of file Eigen.h.

References gmtl::Math::abs(), m\_aafMat, m\_afDiag, m\_afSubd, and gmtl::Math::sqrt().

Referenced by DecrSortEigenStuff(), DecrSortEigenStuffN(), EigenStuff(), EigenStuffN(), IncrSortEigenStuff(), and IncrSortEigenStuffN().

```

388 {
389     int i0, i1, i2, i3;
390
391     for (i0 = iSize-1, i3 = iSize-2; i0 >= 1; i0--, i3--)
392     {
393         float fH = 0.0, fScale = 0.0;
394
395         if ( i3 > 0 )
396         {
397             for (i2 = 0; i2 <= i3; i2++)
398                 fScale += Math::abs(m_aafMat[i0][i2]);
399             if ( fScale == 0 )
400             {
401                 m_afSubd[i0] = m_aafMat[i0][i3];
402             }
403             else
404             {
405                 float fInvScale = 1.0/fScale;
406                 for (i2 = 0; i2 <= i3; i2++)
407                 {
408                     m_aafMat[i0][i2] *= fInvScale;
409                     fH += m_aafMat[i0][i2]*m_aafMat[i0][i2];
410                 }
411                 float fF = m_aafMat[i0][i3];
412                 float fG = Math::sqrt(fH);
413                 if ( fF > 0.0 )

```

```

414         fG = -fG;
415         m_afSubd[i0] = fScale*fG;
416         fH -= fF*fG;
417         m_aafMat[i0][i3] = fF-fG;
418         fF = 0.0;
419         float fInvH = 1.0/fH;
420         for (i1 = 0; i1 <= i3; i1++)
421         {
422             m_aafMat[i1][i0] = m_aafMat[i0][i1]*fInvH;
423             fG = 0.0;
424             for (i2 = 0; i2 <= i1; i2++)
425                 fG += m_aafMat[i1][i2]*m_aafMat[i0][i2];
426             for (i2 = i1+1; i2 <= i3; i2++)
427                 fG += m_aafMat[i2][i1]*m_aafMat[i0][i2];
428             m_afSubd[i1] = fG*fInvH;
429             fF += m_afSubd[i1]*m_aafMat[i0][i1];
430         }
431         float fHalfFdivH = 0.5*fF*fInvH;
432         for (i1 = 0; i1 <= i3; i1++)
433         {
434             fF = m_aafMat[i0][i1];
435             fG = m_afSubd[i1] - fHalfFdivH*fF;
436             m_afSubd[i1] = fG;
437             for (i2 = 0; i2 <= i1; i2++)
438             {
439                 m_aafMat[i1][i2] -= fF*m_afSubd[i2] +
440                     fG*m_aafMat[i0][i2];
441             }
442         }
443     }
444 }
445 else
446 {
447     m_afSubd[i0] = m_aafMat[i0][i3];
448 }
449
450 m_afDiag[i0] = fH;
451 }
452
453 m_afDiag[0] = m_afSubd[0] = 0;
454 for (i0 = 0, i3 = -1; i0 <= iSize-1; i0++, i3++)
455 {
456     if ( m_afDiag[i0] )
457     {
458         for (i1 = 0; i1 <= i3; i1++)
459         {
460             float fSum = 0;
461             for (i2 = 0; i2 <= i3; i2++)
462                 fSum += m_aafMat[i0][i2]*m_aafMat[i2][i1];
463             for (i2 = 0; i2 <= i3; i2++)
464                 m_aafMat[i2][i1] -= fSum*m_aafMat[i2][i0];
465         }
466     }
467     m_afDiag[i0] = m_aafMat[i0][i0];
468     m_aafMat[i0][i0] = 1;

```

```

469         for (i1 = 0; i1 <= i3; i1++)
470             m_aafMat[i1][i0] = m_aafMat[i0][i1] = 0;
471     }
472
473     // re-ordering if Eigen::QLAlgorithm is used subsequently
474     for (i0 = 1, i3 = 0; i0 < iSize; i0++, i3++)
475         m_afSubd[i3] = m_afSubd[i0];
476     m_afSubd[iSize-1] = 0;
477 }

```

### 10.5.3 Member Data Documentation

#### 10.5.3.1 float\*\* gmtl::Eigen::m\_aafMat [protected]

Definition at line 85 of file Eigen.h.

Referenced by DecrSortEigenStuff(), DecrSortEigenStuff2(), DecrSortEigenStuff3(), DecrSortEigenStuff4(), DecrSortEigenStuffN(), Eigen(), EigenStuff(), EigenStuff2(), EigenStuff3(), EigenStuff4(), EigenStuffN(), GetEigenvector(), IncrSortEigenStuff(), IncrSortEigenStuff2(), IncrSortEigenStuff3(), IncrSortEigenStuff4(), IncrSortEigenStuffN(), Matrix(), QLAlgorithm(), SetMatrix(), Tridiagonal2(), Tridiagonal3(), Tridiagonal4(), TridiagonalN(), and ~Eigen().

#### 10.5.3.2 float\* gmtl::Eigen::m\_afDiag [protected]

Definition at line 86 of file Eigen.h.

Referenced by DecrSortEigenStuff(), DecrSortEigenStuff2(), DecrSortEigenStuff3(), DecrSortEigenStuff4(), DecrSortEigenStuffN(), Eigen(), EigenStuff(), EigenStuff2(), EigenStuff3(), EigenStuff4(), EigenStuffN(), GetEigenvalue(), IncrSortEigenStuff(), IncrSortEigenStuff2(), IncrSortEigenStuff3(), IncrSortEigenStuff4(), IncrSortEigenStuffN(), QLAlgorithm(), Tridiagonal2(), Tridiagonal3(), Tridiagonal4(), TridiagonalN(), and ~Eigen().

#### 10.5.3.3 float\* gmtl::Eigen::m\_afSubd [protected]

Definition at line 87 of file Eigen.h.

Referenced by DecrSortEigenStuff(), DecrSortEigenStuff2(), DecrSortEigenStuff3(), DecrSortEigenStuff4(), DecrSortEigenStuffN(), Eigen(), EigenStuff(), EigenStuff2(), EigenStuff3(), EigenStuff4(), EigenStuffN(), IncrSortEigenStuff(), IncrSortEigenStuff2(), IncrSortEigenStuff3(), IncrSortEigenStuff4(), IncrSortEigenStuffN(), QLAlgorithm(), Tridiagonal2(), Tridiagonal3(), Tridiagonal4(), TridiagonalN(), and ~Eigen().

#### 10.5.3.4 `int gmtl::Eigen::m_jSize` [protected]

Definition at line 84 of file Eigen.h.

Referenced by `DecrSortEigenStuff()`, `DecrSortEigenStuff2()`, `DecrSortEigenStuff3()`, `DecrSortEigenStuff4()`, `DecrSortEigenStuffN()`, `Eigen()`, `EigenStuff()`, `EigenStuff2()`, `EigenStuff3()`, `EigenStuff4()`, `EigenStuffN()`, `IncrSortEigenStuff()`, `IncrSortEigenStuff2()`, `IncrSortEigenStuff3()`, `IncrSortEigenStuff4()`, `IncrSortEigenStuffN()`, `SetMatrix()`, and `~Eigen()`.

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

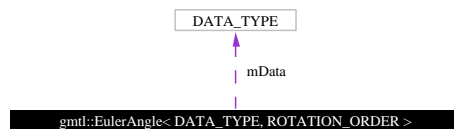
- [Eigen.h](#)

## 10.6 gmtl::EulerAngle< DATA\_TYPE, ROTATION\_ORDER > Class Template Reference

**EulerAngle**: Represents a group of euler angles.

```
#include <EulerAngle.h>
```

Collaboration diagram for gmtl::EulerAngle< DATA\_TYPE, ROTATION\_ORDER >:



### Public Types

- enum { **Size** = 3, **Order** = ROTATION\_ORDER::ID }

### Public Methods

- EulerAngle** ()  
*default constructor.*
- EulerAngle** (const EulerAngle &e)  
*copy constructor.*
- EulerAngle** (DATA\_TYPE p0, DATA\_TYPE p1, DATA\_TYPE p2)  
*data constructor.*
- void **set** (const DATA\_TYPE &p0, const DATA\_TYPE &p1, const DATA\_TYPE &p2)  
*set data.*
- DATA\_TYPE & **operator[]** (const unsigned i)  
*Gets the ith component in this EulerAngle.*
- const DATA\_TYPE & **operator[]** (const unsigned i) const

- `DATA_TYPE * getData ()`

*Gets the internal array of the components.*

- `const DATA_TYPE * getData () const`

*Gets the internal array of the components (const version).*

### 10.6.1 Detailed Description

```
template<typename DATA_TYPE, typename ROTATION_ORDER> class
gmml::EulerAngle< DATA_TYPE, ROTATION_ORDER >
```

[EulerAngle](#): Represents a group of euler angles.

Euler angle can be used to represent rotations in 3-space.

To some people this rotation format can be more intuitive to specify than [Matrix](#), [Quat](#), or [AxisAngle](#) formatted rotation.

For efficiency and to minimize problems from gimbal-lock, you should use one of the other rotation formats instead ([Quat](#) or [Matrix](#) are preferred).

The internal data format is an array of 3 `DATA_TYPE` angle values, plus a Rotation-Order that specifies how to build a rotation transform from the 3 angle value.

IMPORTANT: The 3 angles are in the order set `getOrder()`, not [XYZ](#). The values do not swap when order is changed after setting the angles.

**Precondition:**

all angles are in radians.

**See also:**

`EulerAnglef`, `EulerAngled`, [Matrix](#), [Quat](#), [AxisAngle](#)

**Todo:**

bug: might not want to derive from `vec`, otherwise `EulerXYZ == EulerZYX` works, when it shouldn't even compile...

Definition at line 69 of file `EulerAngle.h`.

### 10.6.2 Member Enumeration Documentation



### 10.6.2.1 `template<typename DATA_TYPE, typename ROTATION_ORDER>` anonymous enum

Enumeration values:

Size

Order

Definition at line 72 of file EulerAngle.h.

```
72 { Size = 3, Order = ROTATION_ORDER::ID };
```

## 10.6.3 Constructor & Destructor Documentation

### 10.6.3.1 `template<typename DATA_TYPE, typename ROTATION_ORDER>` `gmtl::EulerAngle< DATA_TYPE, ROTATION_ORDER` `>::EulerAngle () [inline]`

default constructor.

initializes to identity rotation (no rotation).

Definition at line 75 of file EulerAngle.h.

```
76 {
77     assert( ROTATION_ORDER::IS_ROTORDER == 1 &&
78         "you must specify a RotatoionOrder derived type for the rotationorder in euler angle." );
79     mData[0] = DATA_TYPE( 0 );
80     mData[1] = DATA_TYPE( 0 );
81     mData[2] = DATA_TYPE( 0 );
82 }
```

### 10.6.3.2 `template<typename DATA_TYPE, typename ROTATION_ORDER>` `gmtl::EulerAngle< DATA_TYPE, ROTATION_ORDER` `>::EulerAngle (const EulerAngle< DATA_TYPE,` `ROTATION_ORDER > & e) [inline]`

copy constructor.

Definition at line 85 of file EulerAngle.h.

References `gmtl::EulerAngle< DATA_TYPE, ROTATION_ORDER >::mData`.

```

86  {
87      mData[0] = e.mData[0];
88      mData[1] = e.mData[1];
89      mData[2] = e.mData[2];
90  }

```

**10.6.3.3** `template<typename DATA_TYPE, typename ROTATION_ORDER>  
gmtl::EulerAngle< DATA_TYPE, ROTATION_ORDER  
>::EulerAngle (DATA_TYPE p0, DATA_TYPE p1, DATA_TYPE p2)  
[inline]`

data constructor.

angles are in radians.

Definition at line 93 of file EulerAngle.h.

```

94  {
95      mData[0] = p0;
96      mData[1] = p1;
97      mData[2] = p2;
98  }

```

## 10.6.4 Member Function Documentation

**10.6.4.1** `template<typename DATA_TYPE, typename ROTATION_ORDER>  
const DATA_TYPE* gmtl::EulerAngle< DATA_TYPE,  
ROTATION_ORDER >::getData () const [inline]`

Gets the internal array of the components (const version).

**Returns:**

a pointer to the component array with length SIZE

Definition at line 136 of file EulerAngle.h.

```

136 { return mData; }

```

**10.6.4.2** `template<typename DATA_TYPE, typename ROTATION_ORDER>  
DATA_TYPE* gmtl::EulerAngle< DATA_TYPE, ROTATION_ORDER  
>::getData () [inline]`

Gets the internal array of the components.

**Returns:**

a pointer to the component array with length SIZE

Definition at line 131 of file EulerAngle.h.

```
131 { return mData; }
```

**10.6.4.3** `[]`

`template<typename DATA_TYPE, typename ROTATION_ORDER> const DATA_-  
TYPE& gmtl::EulerAngle< DATA_TYPE, ROTATION_ORDER >::operator[] (const  
unsigned i) const [inline]`

Definition at line 120 of file EulerAngle.h.

References `gmtlASSERT`, and `gmtl::EulerAngle< DATA_TYPE, ROTATION_-  
ORDER >::Size`.

```
121 {  
122     gmtlASSERT( i < Size );  
123     return mData[i];  
124 }
```

**10.6.4.4** `[]`

`template<typename DATA_TYPE, typename ROTATION_ORDER> DATA_TYPE&  
gmtl::EulerAngle< DATA_TYPE, ROTATION_ORDER >::operator[] (const unsigned  
i) [inline]`

Gets the *i*th component in this [EulerAngle](#).

**Parameters:**

*i* the zero-based index of the component to access.

**Precondition:**

$0 \leq i < 3$

**Returns:**

a reference to the *i*th component

Definition at line 115 of file EulerAngle.h.

References `gmtlASSERT`, and `gmtl::EulerAngle< DATA_TYPE, ROTATION_ORDER >::Size`.

```

116    {
117        gmtlASSERT( i < Size );
118        return mData[i];
119    }
```

**10.6.4.5** `template<typename DATA_TYPE, typename ROTATION_ORDER>`  
`void gmtl::EulerAngle< DATA_TYPE, ROTATION_ORDER >::set`  
`(const DATA_TYPE & p0, const DATA_TYPE & p1, const DATA_TYPE`  
`& p2) [inline]`

set data.

angles are in radians.

Definition at line 101 of file EulerAngle.h.

```

103    {
104        mData[0] = p0;
105        mData[1] = p1;
106        mData[2] = p2;
107    }
```

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

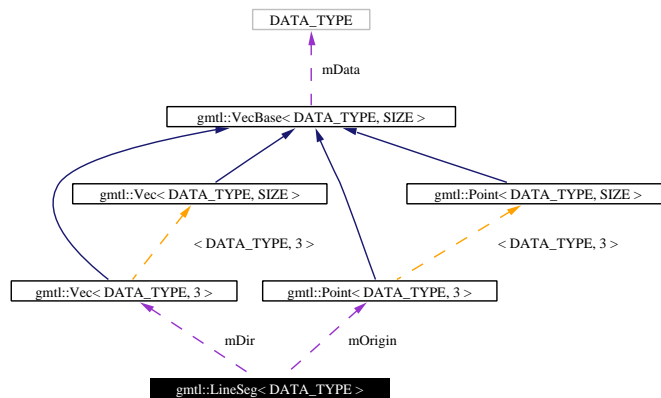
- [EulerAngle.h](#)

## 10.7 gmtl::LineSeg< DATA\_TYPE > Class Template Reference

Describes a line segment.

```
#include <LineSeg.h>
```

Collaboration diagram for gmtl::LineSeg< DATA\_TYPE >:



### Public Methods

- [LineSeg](#) ()  
*Constructs a line segment at the origin with a zero vector.*
- [LineSeg](#) (const [Point](#)< DATA\_TYPE, 3 > &origin, const [Vec](#)< DATA\_TYPE, 3 > &dir)  
*Constructs a line segment with the given origin and vector.*
- [LineSeg](#) (const [Point](#)< DATA\_TYPE, 3 > &beg, const [Point](#)< DATA\_TYPE, 3 > &end)  
*Constructs a line segment with the given beginning and ending points.*
- [LineSeg](#) (const LineSeg &lineseg)  
*Constructs an exact duplicate of the given line segment.*
- const [Point](#)< DATA\_TYPE, 3 > & [getOrigin](#) () const

*Gets the origin of the line segment.*

- void `setOrigin` (const `Point`< DATA\_TYPE, 3 > &origin)  
*Sets the origin point for this line segment.*
- const `Vec`< DATA\_TYPE, 3 > & `getDir` () const  
*Gets the vector describing the direction and length of the line segment.*
- void `setDir` (const `Vec`< DATA\_TYPE, 3 > &dir)  
*Sets the vector describing the direction and length of the line segment.*
- const DATA\_TYPE & `getLength` () const  
*Gets the length of this line segment.*

## Public Attributes

- `Point`< DATA\_TYPE, 3 > `mOrigin`  
*The origin of the line segment.*
- `Vec`< DATA\_TYPE, 3 > `mDir`  
*The vector along which the line segment lies.*

### 10.7.1 Detailed Description

`template<class DATA_TYPE> class gmtl::LineSeg< DATA_TYPE >`

Describes a line segment.

This is represented by a point origin O and a vector spanning the length of the line segment originating at O. Thus any point on the line segment can be described as

$$P(s) = O + Vs$$

where  $0 \leq s \leq 1$

#### Parameters:

**DATA\_TYPE** the internal type used for the point and vector

Definition at line 56 of file LineSeg.h.

### 10.7.2 Constructor & Destructor Documentation

### 10.7.2.1 `template<class DATA_TYPE> gmtl::LineSeg< DATA_TYPE >::LineSeg () [inline]`

Constructs a line segment at the origin with a zero vector.

Definition at line 62 of file LineSeg.h.

```
63     {}
```

### 10.7.2.2 `template<class DATA_TYPE> gmtl::LineSeg< DATA_TYPE >::LineSeg (const Point< DATA_TYPE, 3 > & origin, const Vec< DATA_TYPE, 3 > & dir) [inline]`

Constructs a line segment with the given origin and vector.

#### Parameters:

*origin* the point at which the line segment starts

*dir* the vector describing the direction and length of the line segment starting at origin

Definition at line 72 of file LineSeg.h.

References gmtl::LineSeg< DATA\_TYPE >::mDir, and gmtl::LineSeg< DATA\_TYPE >::mOrigin.

```
73     : mOrigin( origin ), mDir( dir )
74     {}
```

### 10.7.2.3 `template<class DATA_TYPE> gmtl::LineSeg< DATA_TYPE >::LineSeg (const Point< DATA_TYPE, 3 > & beg, const Point< DATA_TYPE, 3 > & end) [inline]`

Constructs a line segment with the given beginning and ending points.

#### Parameters:

*beg* the point at the beginning of the line segment

*end* the point at the end of the line segment

Definition at line 82 of file LineSeg.h.

References gmtl::LineSeg< DATA\_TYPE >::mDir, and gmtl::LineSeg< DATA\_TYPE >::mOrigin.

```

83      : mOrigin( beg )
84      {
85          mDir = end - beg;
86      }

```

#### 10.7.2.4 `template<class DATA_TYPE> gmtl::LineSeg< DATA_TYPE >::LineSeg (const LineSeg< DATA_TYPE > & lineseg) [inline]`

Constructs an exact duplicate of the given line segment.

##### Parameters:

*lineseg* the line segment to copy

Definition at line 93 of file LineSeg.h.

References gmtl::LineSeg< DATA\_TYPE >::mDir, and gmtl::LineSeg< DATA\_TYPE >::mOrigin.

```

94      {
95          mOrigin = lineseg.mOrigin;
96          mDir = lineseg.mDir;
97      }

```

### 10.7.3 Member Function Documentation

#### 10.7.3.1 `template<class DATA_TYPE> const Vec<DATA_TYPE, 3>& gmtl::LineSeg< DATA_TYPE >::getDir () const [inline]`

Gets the vector describing the direction and length of the line segment.

##### Returns:

the line segment's vector

Definition at line 124 of file LineSeg.h.

References gmtl::LineSeg< DATA\_TYPE >::mDir.

```

125      {
126          return mDir;
127      }

```



### 10.7.3.2 `template<class DATA_TYPE> const DATA_TYPE& gmtl::LineSeg< DATA_TYPE >::getLength () const` [inline]

Gets the length of this line segment.

Definition at line 142 of file LineSeg.h.

References gmtl::length().

```
143     {  
144         return length( dir );  
145     }
```

### 10.7.3.3 `template<class DATA_TYPE> const Point<DATA_TYPE, 3>& gmtl::LineSeg< DATA_TYPE >::getOrigin () const` [inline]

Gets the origin of the line segment.

#### Returns:

the point at the beginning of the line

Definition at line 104 of file LineSeg.h.

References gmtl::LineSeg< DATA\_TYPE >::mOrigin.

```
105     {  
106         return mOrigin;  
107     }
```

### 10.7.3.4 `template<class DATA_TYPE> void gmtl::LineSeg< DATA_TYPE >::setDir (const Vec< DATA_TYPE, 3 > & dir)` [inline]

Sets the vector describing the direction and length of the line segment.

#### Parameters:

*dir* the line segment's vector

Definition at line 134 of file LineSeg.h.

References gmtl::LineSeg< DATA\_TYPE >::mDir.

```

135     {
136         mDir = dir;
137     }

```

#### 10.7.3.5 `template<class DATA_TYPE> void gmtl::LineSeg< DATA_TYPE >::setOrigin (const Point< DATA_TYPE, 3 > &origin) [inline]`

Sets the origin point for this line segment.

##### Parameters:

*origin* the point at which the line segment starts

Definition at line 114 of file LineSeg.h.

References `gmtl::LineSeg< DATA_TYPE >::mOrigin`.

```

115     {
116         mOrigin = origin;
117     }

```

### 10.7.4 Member Data Documentation

#### 10.7.4.1 `template<class DATA_TYPE> Vec<DATA_TYPE, 3> gmtl::LineSeg< DATA_TYPE >::mDir`

The vector along which the line segment lies.

Definition at line 156 of file LineSeg.h.

Referenced by `gmtl::findNearestPt()`, `gmtl::LineSeg< DATA_TYPE >::getDir()`, `gmtl::isEqual()`, `gmtl::LineSeg< DATA_TYPE >::LineSeg()`, `gmtl::operator==()`, and `gmtl::LineSeg< DATA_TYPE >::setDir()`.

#### 10.7.4.2 `template<class DATA_TYPE> Point<DATA_TYPE, 3> gmtl::LineSeg< DATA_TYPE >::mOrigin`

The origin of the line segment.

Definition at line 151 of file LineSeg.h.

Referenced by `gmtl::findNearestPt()`, `gmtl::LineSeg< DATA_TYPE >::getOrigin()`, `gmtl::isEqual()`, `gmtl::LineSeg< DATA_TYPE >::LineSeg()`, `gmtl::operator==(())`, and `gmtl::LineSeg< DATA_TYPE >::setOrigin()`.

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

- [LineSeg.h](#)

## 10.8 gmtl::Matrix< DATA\_TYPE, ROWS, COLS > Class Template Reference

[Matrix](#): 4x4 [Matrix](#) class (OpenGL ordering).

```
#include <Matrix.h>
```

Collaboration diagram for gmtl::Matrix< DATA\_TYPE, ROWS, COLS >:



### Public Types

- typedef DATA\_TYPE [DataType](#)  
*use this to declare single value types of the same type as this matrix.*
  - enum { [Rows](#) = ROWS, [Cols](#) = COLS }
  - enum [XformState](#) { [IDENTITY](#) = 1, [ORTHOGONAL](#) = 2, [ORTHONORMAL](#) = 4, [AFFINE](#) = 8, [FULL](#) = 16, [XFORM\\_ERROR](#) = 32 }
- describes the xforms that this matrix has been through.*

### Public Methods

- [Matrix](#) ()  
*Default Constructor (Identity constructor).*
- [Matrix](#) (const Matrix< DATA\_TYPE, ROWS, COLS > &matrix)  
*copy constructor.*
- void [set](#) (DATA\_TYPE v00, DATA\_TYPE v01, DATA\_TYPE v10, DATA\_TYPE v11)  
*element wise setter for 2x2.*

- void [set](#) (DATA\_TYPE v00, DATA\_TYPE v01, DATA\_TYPE v02, DATA\_TYPE v10, DATA\_TYPE v11, DATA\_TYPE v12)  
*element wise setter for 2x3.*
- void [set](#) (DATA\_TYPE v00, DATA\_TYPE v01, DATA\_TYPE v02, DATA\_TYPE v10, DATA\_TYPE v11, DATA\_TYPE v12, DATA\_TYPE v20, DATA\_TYPE v21, DATA\_TYPE v22)  
*element wise setter for 3x3.*
- void [set](#) (DATA\_TYPE v00, DATA\_TYPE v01, DATA\_TYPE v02, DATA\_TYPE v03, DATA\_TYPE v10, DATA\_TYPE v11, DATA\_TYPE v12, DATA\_TYPE v13, DATA\_TYPE v20, DATA\_TYPE v21, DATA\_TYPE v22, DATA\_TYPE v23)  
*element wise setter for 3x4.*
- void [set](#) (DATA\_TYPE v00, DATA\_TYPE v01, DATA\_TYPE v02, DATA\_TYPE v03, DATA\_TYPE v10, DATA\_TYPE v11, DATA\_TYPE v12, DATA\_TYPE v13, DATA\_TYPE v20, DATA\_TYPE v21, DATA\_TYPE v22, DATA\_TYPE v23, DATA\_TYPE v30, DATA\_TYPE v31, DATA\_TYPE v32, DATA\_TYPE v33)  
*element wise setter for 4x4.*
- void [set](#) (const DATA\_TYPE \*data)  
*set the matrix to the given data.*
- void [setTranspose](#) (const DATA\_TYPE \*data)  
*set the matrix to the transpose of the given data.*
- DATA\_TYPE & [operator\(\)](#) (const unsigned row, const unsigned column)  
*access [row, col] in the matrix.*
- const DATA\_TYPE & [operator\(\)](#) (const unsigned row, const unsigned column) const  
*access [row, col] in the matrix (const version).*
- DATA\_TYPE & [operator\[\]](#) (const unsigned i)  
*bracket operator.*
- const DATA\_TYPE & [operator\[\]](#) (const unsigned i) const  
*bracket operator.*
- const DATA\_TYPE \* [getData](#) () const  
*Get a DATA\_TYPE pointer to the matrix data RETVAL: Returns a ptr to the head of the matrix data.*
- bool [isError](#) ()
- void [setError](#) ()

## Public Attributes

- DATA\_TYPE [mData](#) [COLS \*ROWS]

*Column major.*

- char [mState](#)

*describes what xforms are in this matrix.*

### 10.8.1 Detailed Description

**template<typename DATA\_TYPE, unsigned ROWS, unsigned COLS> class gmtl::Matrix< DATA\_TYPE, ROWS, COLS >**

[Matrix](#): 4x4 [Matrix](#) class (OpenGL ordering).

C/C++ uses matrices in row major order. In other words the access indices look like:  
mat[row][col]

(0,0) (0,1) (0,2) (0,3) <=== Array

(1,0) (1,1) (1,2) (1,3) <=== Array

(2,0) (2,1) (2,2) (2,3) <=== Array

(3,0) (3,1) (3,2) (3,3) <=== Array

OpenGL ordering specifies that the matrix has to be column major in memory, so we need to access it more like:

NOTE: The given indexes are what the cells have to be called in C/C++ notation. Since we are putting the columns into memory back-to-back.

(0,0) (1,0) (2,0) (3,0)

(0,1) (1,1) (2,1) (3,1)

(0,2) (1,2) (2,2) (3,2)

(0,3) (1,3) (2,3) (3,3)

^ ^ ^ ^

===== Arrays

So basically OpenGL ordering is the Transpose of the way C++ accesses the array

**See also:**

[Matrix44f](#) , [Matrix44d](#)

Definition at line 74 of file Matrix.h.

## 10.8.2 Member Typedef Documentation

**10.8.2.1** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS>  
typedef DATA_TYPE gmtl::Matrix< DATA_TYPE, ROWS, COLS  
>::DataType`

use this to declare single value types of the same type as this matrix.

Definition at line 79 of file Matrix.h.

## 10.8.3 Member Enumeration Documentation

**10.8.3.1** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS>  
anonymous enum`

Enumeration values:

**Rows**

**Cols**

Definition at line 80 of file Matrix.h.

```
81     {  
82         Rows = ROWS, Cols = COLS  
83     };
```

**10.8.3.2** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS>  
enum gmtl::Matrix::XformState`

describes the xforms that this matrix has been through.

Enumeration values:

**IDENTITY**

**ORTHOGONAL**

**ORTHONORMAL**

**AFFINE**

**FULL**

## XFORM\_ERROR

Definition at line 86 of file Matrix.h.

```

87     {
88         IDENTITY = 1,
89         ORTHOGONAL = 2,
90         ORTHONORMAL = 4,
91         AFFINE = 8,
92         FULL = 16,
93         XFORM_ERROR = 32 // error bit
94     };

```

## 10.8.4 Constructor & Destructor Documentation

### 10.8.4.1 `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> gmtl::Matrix< DATA_TYPE, ROWS, COLS >::Matrix () [inline]`

Default Constructor (Identity constructor).

Definition at line 97 of file Matrix.h.

References `gmtl::Matrix< DATA_TYPE, ROWS, COLS >::FULL`, `gmtl::Math::Min()`, `gmtl::Matrix< DATA_TYPE, ROWS, COLS >::mState`, and `gmtl::Matrix< DATA_TYPE, ROWS, COLS >::operator()`.

```

98     {
99         for (unsigned int r = 0; r < ROWS; ++r)
100             for (unsigned int c = 0; c < COLS; ++c)
101                 this->operator()( r, c ) = (DATA_TYPE)0.0;
102
103         for (unsigned int x = 0; x < Math::Min( COLS, ROWS ); ++x)
104             this->operator()( x, x ) = (DATA_TYPE)1.0;
105
106         mState = FULL;
107     };

```

### 10.8.4.2 `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> gmtl::Matrix< DATA_TYPE, ROWS, COLS >::Matrix (const Matrix< DATA_TYPE, ROWS, COLS > & matrix) [inline]`

copy constructor.



Definition at line 113 of file Matrix.h.

References gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::getData(), gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::mState, and gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::set().

```

114     {
115         this->set( matrix.getData() );
116         mState = matrix.mState;
117     }

```

## 10.8.5 Member Function Documentation

### 10.8.5.1 `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> const DATA_TYPE* gmtl::Matrix< DATA_TYPE, ROWS, COLS >::getData () const [inline]`

Get a DATA\_TYPE pointer to the matrix data RETVAL: Returns a ptr to the head of the matrix data.

Definition at line 321 of file Matrix.h.

References gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::mData.

Referenced by gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::Matrix().

```

321 { return (DATA_TYPE*)mData; }

```

### 10.8.5.2 `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> bool gmtl::Matrix< DATA_TYPE, ROWS, COLS >::isError () [inline]`

Definition at line 323 of file Matrix.h.

References gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::mState, and gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::XFORM\_ERROR.

```

324     {
325         return mState & XFORM_ERROR;
326     }

```

**10.8.5.3** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS>  
const DATA_TYPE& gmtl::Matrix< DATA_TYPE, ROWS, COLS  
>::operator() (const unsigned row, const unsigned column) const  
[inline]`

access [row, col] in the matrix (const version).

Definition at line 298 of file Matrix.h.

References gmtlASSERT, and gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::mData.

```
299     {
300         gmtlASSERT( (row < ROWS) && (column < COLS) );
301         return mData[column*ROWS + row];
302     }
```

**10.8.5.4** `template<typename DATA_TYPE, unsigned ROWS, unsigned  
COLS> DATA_TYPE& gmtl::Matrix< DATA_TYPE, ROWS,  
COLS >::operator() (const unsigned row, const unsigned column)  
[inline]`

access [row, col] in the matrix.

Definition at line 291 of file Matrix.h.

References gmtlASSERT, and gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::mData.

Referenced by gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::Matrix(), and gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::setTranspose().

```
292     {
293         gmtlASSERT( (row < ROWS) && (column < COLS) );
294         return mData[column*ROWS + row];
295     }
```

#### **10.8.5.5 ]**

`template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> const DATA_  
TYPE& gmtl::Matrix< DATA_TYPE, ROWS, COLS >::operator[] (const unsigned i)  
const [inline]`

bracket operator.

Definition at line 312 of file Matrix.h.

References gmtlASSERT, and gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::mData.

```

313     {
314         gmtlASSERT( i < (ROWS*COLS) );
315         return mData[i];
316     }

```

#### 10.8.5.6 ]

template<typename DATA\_TYPE, unsigned ROWS, unsigned COLS> DATA\_TYPE& gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::operator[] (const unsigned i) [inline]

bracket operator.

Definition at line 305 of file Matrix.h.

References gmtlASSERT, and gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::mData.

```

306     {
307         gmtlASSERT( i < (ROWS*COLS) );
308         return mData[i];
309     }

```

#### 10.8.5.7 template<typename DATA\_TYPE, unsigned ROWS, unsigned COLS> void gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::set (const DATA\_TYPE \* data) [inline]

set the matrix to the given data.

This function is useful to copy matrix data from another math library.

**”Example (to a matrix using an external math library):”**

```

pfMatrix other_matrix;
other_matrix.setRot( 90, 1, 0, 0 );

gmtl::Matrix44f mat;
mat.set( other_matrix.getFloatPtr() );

```

WARNING: this isn’t really safe, size and datatype are not enforced by the compiler.

#### Precondition:

data is in the native format of the gmtl::Matrix class, if not, then you might be able to use the setTranspose function.

i.e. in a 4x4 data[0-3] is the 1st column, data[4-7] is 2nd, etc...

Definition at line 251 of file Matrix.h.

References gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::FULL, gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::mData, and gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::mState.

```

252     {
253     for (unsigned int x = 0; x < ROWS * COLS; ++x)
254         mData[x] = data[x];
255     mState = FULL;
256     }

```

**10.8.5.8** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS>  
void gmtl::Matrix< DATA_TYPE, ROWS, COLS >::set (DATA_TYPE  
v00, DATA_TYPE v01, DATA_TYPE v02, DATA_TYPE v03,  
DATA_TYPE v10, DATA_TYPE v11, DATA_TYPE v12, DATA_TYPE  
v13, DATA_TYPE v20, DATA_TYPE v21, DATA_TYPE v22,  
DATA_TYPE v23, DATA_TYPE v30, DATA_TYPE v31, DATA_TYPE  
v32, DATA_TYPE v33) [inline]`

element wise setter for 4x4.

#### Todo:

needs mp!! currently no way for a 4x3, ....

Definition at line 199 of file Matrix.h.

References gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::FULL, gmtl::ASSERT, gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::mData, and gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::mState.

```

203     {
204     gmtl::ASSERT( ROWS == 4 && COLS == 4 );// could be compile time...
205     mData[0] = v00;
206     mData[1] = v10;
207     mData[2] = v20;
208     mData[4] = v01;
209     mData[5] = v11;
210     mData[6] = v21;
211     mData[8] = v02;
212     mData[9] = v12;
213     mData[10] = v22;
214
215     // right row

```

```

216         mData[12] = v03;
217         mData[13] = v13;
218         mData[14] = v23;
219
220         // bottom row
221         mData[3]  = v30;
222         mData[7]  = v31;
223         mData[11] = v32;
224         mData[15] = v33;
225         mState = FULL;
226     }

```

**10.8.5.9** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> void gmtl::Matrix< DATA_TYPE, ROWS, COLS >::set (DATA_TYPE v00, DATA_TYPE v01, DATA_TYPE v02, DATA_TYPE v03, DATA_TYPE v10, DATA_TYPE v11, DATA_TYPE v12, DATA_TYPE v13, DATA_TYPE v20, DATA_TYPE v21, DATA_TYPE v22, DATA_TYPE v23) [inline]`

element wise setter for 3x4.

**Todo:**

needs mp!! currently no way for a 4x3, ....

Definition at line 174 of file Matrix.h.

References `gmtl::Matrix< DATA_TYPE, ROWS, COLS >::FULL`, `gmtlASSERT`, `gmtl::Matrix< DATA_TYPE, ROWS, COLS >::mData`, and `gmtl::Matrix< DATA_TYPE, ROWS, COLS >::mState`.

```

177     {
178         gmtlASSERT( ROWS == 3 && COLS == 4 );// could be compile time...
179         mData[0] = v00;
180         mData[1] = v10;
181         mData[2] = v20;
182         mData[3] = v01;
183         mData[4] = v11;
184         mData[5] = v21;
185         mData[6] = v02;
186         mData[7] = v12;
187         mData[8] = v22;
188
189         // right row
190         mData[9]  = v03;
191         mData[10] = v13;
192         mData[11] = v23;
193         mState = FULL;
194     }

```

**10.8.5.10** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> void gmtl::Matrix< DATA_TYPE, ROWS, COLS >::set (DATA_TYPE v00, DATA_TYPE v01, DATA_TYPE v02, DATA_TYPE v10, DATA_TYPE v11, DATA_TYPE v12, DATA_TYPE v20, DATA_TYPE v21, DATA_TYPE v22) [inline]`

element wise setter for 3x3.

**Todo:**

needs mp!!

Definition at line 152 of file Matrix.h.

References `gmtl::Matrix< DATA_TYPE, ROWS, COLS >::FULL`, `gmtlASSERT`, `gmtl::Matrix< DATA_TYPE, ROWS, COLS >::mData`, and `gmtl::Matrix< DATA_TYPE, ROWS, COLS >::mState`.

```

155     {
156         gmtlASSERT( ROWS == 3 && COLS == 3 ); // could be at compile time...
157         mData[0] = v00;
158         mData[1] = v10;
159         mData[2] = v20;
160
161         mData[3] = v01;
162         mData[4] = v11;
163         mData[5] = v21;
164
165         mData[6] = v02;
166         mData[7] = v12;
167         mData[8] = v22;
168         mState = FULL;
169     }
```

**10.8.5.11** `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> void gmtl::Matrix< DATA_TYPE, ROWS, COLS >::set (DATA_TYPE v00, DATA_TYPE v01, DATA_TYPE v02, DATA_TYPE v10, DATA_TYPE v11, DATA_TYPE v12) [inline]`

element wise setter for 2x3.

**Todo:**

needs mp!!

Definition at line 136 of file Matrix.h.

References gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::FULL, gmtlASSERT, gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::mData, and gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::mState.

```

138     {
139         gmtlASSERT( ROWS == 2 && COLS == 3 ); // could be at compile time...
140         mData[0] = v00;
141         mData[1] = v10;
142         mData[2] = v01;
143         mData[3] = v11;
144         mData[4] = v02;
145         mData[5] = v12;
146         mState = FULL;
147     }

```

#### 10.8.5.12 template<typename DATA\_TYPE, unsigned ROWS, unsigned COLS> void gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::set (DATA\_TYPE v00, DATA\_TYPE v01, DATA\_TYPE v10, DATA\_TYPE v11) [inline]

element wise setter for 2x2.

#### Todo:

needs mp!!

Definition at line 122 of file Matrix.h.

References gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::FULL, gmtlASSERT, gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::mData, and gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::mState.

Referenced by gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::Matrix(), and gmtl::set().

```

124     {
125         gmtlASSERT( ROWS == 2 && COLS == 2 ); // could be at compile time...
126         mData[0] = v00;
127         mData[1] = v10;
128         mData[2] = v01;
129         mData[3] = v11;
130         mState = FULL;
131     }

```

#### 10.8.5.13 `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> void gmtl::Matrix< DATA_TYPE, ROWS, COLS >::setError () [inline]`

Definition at line 327 of file Matrix.h.

References `gmtl::Matrix< DATA_TYPE, ROWS, COLS >::mState`, and `gmtl::Matrix< DATA_TYPE, ROWS, COLS >::XFORM_ERROR`.

```
328     {
329         mState |= XFORM_ERROR;
330     }
```

#### 10.8.5.14 `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> void gmtl::Matrix< DATA_TYPE, ROWS, COLS >::setTranspose (const DATA_TYPE * data) [inline]`

set the matrix to the transpose of the given data.

normally `set()` takes raw matrix data in column by column order, this function allows you to pass in row by row data.

Normally you'll use this function if you want to use a float array to init the matrix (see code example).

**"Example (to set a [15 -4 20] translation using float array):"**

```
float data[] = { 1, 0, 0, 15,
                0, 1, 0, -4,
                0, 0, 1, 20,
                0, 0, 0, 1  };
gmtl::Matrix44f mat;
mat.setTranspose( data );
```

WARNING: this isn't really safe, size and datatype are not enforced by the compiler.

#### **Precondition:**

ptr is in the transpose of the native format of the `Matrix` class  
i.e. in a 4x4 data[0-3] is the 1st row, data[4-7] is 2nd, etc...

Definition at line 281 of file Matrix.h.

References `gmtl::Matrix< DATA_TYPE, ROWS, COLS >::FULL`, `gmtl::Matrix< DATA_TYPE, ROWS, COLS >::mState`, and `gmtl::Matrix< DATA_TYPE, ROWS, COLS >::operator()`.



```

282  {
284      for (unsigned int r = 0; r < ROWS; ++r)
285          for (unsigned int c = 0; c < COLS; ++c)
286              this->operator()( r, c ) = data[(r * COLS) + c];
287      mState = FULL;
288  }

```

## 10.8.6 Member Data Documentation

### 10.8.6.1 `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> DATA_TYPE gmtl::Matrix< DATA_TYPE, ROWS, COLS >::mData[COLS*ROWS]`

Column major.

In other words {Column1, Column2, Column3, Column4} in memory access element mData[column][row]

Definition at line 336 of file Matrix.h.

Referenced by gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::getData(), gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::operator(), gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::operator[], and gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::set().

### 10.8.6.2 `template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> char gmtl::Matrix< DATA_TYPE, ROWS, COLS >::mState`

describes what xforms are in this matrix.

Definition at line 339 of file Matrix.h.

Referenced by gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::isError(), gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::Matrix(), gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::set(), gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::setError(), and gmtl::Matrix< DATA\_TYPE, ROWS, COLS >::setTranspose().

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

- [Matrix.h](#)

## 10.9 gmtl::OOBox Class Reference

```
#include <OOBox.h>
```

### Public Methods

- [OOBox](#) ()
- [OOBox](#) (OOBox &box)
- Point3 & [center](#) ()
- const Point3 & [center](#) () const
- Vec3 & [axis](#) (int i)
- const Vec3 & [axis](#) (int i) const
- Vec3 \* [axes](#) ()
- const Vec3 \* [axes](#) () const
- float & [halfLen](#) (int i)
- const float & [halfLen](#) (int i) const
- float \* [halfLens](#) ()
- const float \* [halfLens](#) () const
- OOBox & [operator=](#) (const OOBox &box)
- bool [operator==](#) (const OOBox &box) const
- void [getVerts](#) (Point3 verts[8]) const
- void [mergeWith](#) (const OOBox &box)
- void [ident](#) ()

### Public Attributes

- Point3 [mCenter](#)
- Vec3 [mAxis](#) [3]
- float [mHalfLen](#) [3]

### 10.9.1 Constructor & Destructor Documentation

### 10.9.1.1 gmtl::OOBox::OOBox() [inline]

Definition at line 52 of file OOBBox.h.

References `ident()`.

```
53     { ident(); }
```

### 10.9.1.2 gmtl::OOBox::OOBox(OOBBox & box) [inline]

Definition at line 106 of file OOBBox.h.

References `mAxis`, `mCenter`, and `mHalfLen`.

```
107 {  
108     mCenter = box.mCenter;  
109     mAxis[0] = box.mAxis[0];  
110     mAxis[1] = box.mAxis[1];  
111     mAxis[2] = box.mAxis[2];  
112     mHalfLen[0] = box.mHalfLen[0];  
113     mHalfLen[1] = box.mHalfLen[1];  
114     mHalfLen[2] = box.mHalfLen[2];  
115 }
```

## 10.9.2 Member Function Documentation

### 10.9.2.1 const Vec3 \* gmtl::OOBox::axes() const [inline]

Definition at line 143 of file OOBBox.h.

References `mAxis`.

```
144 {  
145     return mAxis;  
146 }
```

### 10.9.2.2 Vec3 \* gmtl::OOBox::axes() [inline]

Definition at line 138 of file OOBBox.h.

References mAxis.

Referenced by gmtl::TestIntersect(), and gmtl::TestIntersectOBB().

```
139 {  
140     return mAxis;  
141 }
```

#### **10.9.2.3** `const Vec3 & gmtl::OOBox::axis(int i) const` [inline]

Definition at line 133 of file OOBox.h.

References mAxis.

```
134 {  
135     return mAxis[i];  
136 }
```

#### **10.9.2.4** `Vec3 & gmtl::OOBox::axis(int i)` [inline]

Definition at line 128 of file OOBox.h.

References mAxis.

```
129 {  
130     return mAxis[i];  
131 }
```

#### **10.9.2.5** `const Point3 & gmtl::OOBox::center() const` [inline]

Definition at line 123 of file OOBox.h.

References mCenter.

```
124 {  
125     return mCenter;  
126 }
```

**10.9.2.6 Point3 & gmtl::OOBox::center () [inline]**

Definition at line 118 of file OOBBox.h.

References mCenter.

Referenced by gmtl::TestIntersect(), and gmtl::TestIntersectOBB().

```
119 {
120     return mCenter;
121 }
```

**10.9.2.7 void gmtl::OOBox::getVerts (Point3 verts[8]) const [inline]**

Definition at line 193 of file OOBBox.h.

References mAxis, mCenter, and mHalfLen.

```
194 {
195     Vec3 x_half_axis = mAxis[0]*mHalfLen[0];
196     Vec3 y_half_axis = mAxis[1]*mHalfLen[1];
197     Vec3 z_half_axis = mAxis[2]*mHalfLen[2];
198
199     verts[0] = mCenter - x_half_axis - y_half_axis - z_half_axis;
200     verts[1] = mCenter + x_half_axis - y_half_axis - z_half_axis;
201     verts[2] = mCenter + x_half_axis + y_half_axis - z_half_axis;
202     verts[3] = mCenter - x_half_axis + y_half_axis - z_half_axis;
203     verts[4] = mCenter - x_half_axis - y_half_axis + z_half_axis;
204     verts[5] = mCenter + x_half_axis - y_half_axis + z_half_axis;
205     verts[6] = mCenter + x_half_axis + y_half_axis + z_half_axis;
206     verts[7] = mCenter - x_half_axis + y_half_axis + z_half_axis;
207 }
```

**10.9.2.8 const float & gmtl::OOBox::halfLen (int i) const [inline]**

Definition at line 153 of file OOBBox.h.

References mHalfLen.

```
154 {
155     return mHalfLen[i];
156 }
```

**10.9.2.9 float & gmtl::OOBox::halfLen (int *i*) [inline]**

Definition at line 148 of file OOBBox.h.

References mHalfLen.

```
149 {  
150     return mHalfLen[i];  
151 }
```

**10.9.2.10 const float \* gmtl::OOBox::halfLens () const [inline]**

Definition at line 163 of file OOBBox.h.

References mHalfLen.

```
164 {  
165     return mHalfLen;  
166 }
```

**10.9.2.11 float \* gmtl::OOBox::halfLens () [inline]**

Definition at line 158 of file OOBBox.h.

References mHalfLen.

Referenced by gmtl::TestIntersect(), and gmtl::TestIntersectOBB().

```
159 {  
160     return mHalfLen;  
161 }
```

**10.9.2.12 void gmtl::OOBox::ident () [inline]**

Definition at line 86 of file OOBBox.h.

References mAxis, mCenter, and mHalfLen.

Referenced by OOBBox().

```

87     {
88         mCenter = ZeroVec3;
89         mAxis[0] = XUnitVec3;
90         mAxis[1] = YUnitVec3;
91         mAxis[2] = ZUnitVec3;
92         mHalfLen[0] = mHalfLen[1] = mHalfLen[2] = 0.0f;
93     }

```

#### 10.9.2.13 void gmtl::OOBox::mergeWith (const OOBox & *box*)

#### 10.9.2.14 OOBox & gmtl::OOBox::operator= (const OOBox & *box*) [inline]

Definition at line 169 of file OOBox.h.

References mAxis, mCenter, and mHalfLen.

```

170 {
171     mCenter = box.mCenter;
172     mAxis[0] = box.mAxis[0];
173     mAxis[1] = box.mAxis[1];
174     mAxis[2] = box.mAxis[2];
175     mHalfLen[0] = box.mHalfLen[0];
176     mHalfLen[1] = box.mHalfLen[1];
177     mHalfLen[2] = box.mHalfLen[2];
178     return *this;
179 }

```

#### 10.9.2.15 bool gmtl::OOBox::operator== (const OOBox & *box*) const [inline]

Definition at line 182 of file OOBox.h.

References mAxis, mCenter, and mHalfLen.

```

183 {
184     return ((mCenter == box.mCenter) &&
185         (mAxis[0] == box.mAxis[0]) &&
186         (mAxis[1] == box.mAxis[1]) &&
187         (mAxis[2] == box.mAxis[2]) &&
188         (mHalfLen[0] == box.mHalfLen[0]) &&
189         (mHalfLen[1] == box.mHalfLen[1]) &&

```

```
190             (mHalfLen[2] == box.mHalfLen[2]));  
191 }
```

### 10.9.3 Member Data Documentation

#### 10.9.3.1 Vec3 gmtl::OOBox::mAxis[3]

Definition at line 97 of file OOBox.h.

Referenced by axes(), axis(), getVerts(), ident(), OOBox(), operator=(), and operator==().

#### 10.9.3.2 Point3 gmtl::OOBox::mCenter

Definition at line 96 of file OOBox.h.

Referenced by center(), getVerts(), ident(), OOBox(), operator=(), and operator==().

#### 10.9.3.3 float gmtl::OOBox::mHalfLen[3]

Definition at line 98 of file OOBox.h.

Referenced by getVerts(), halfLen(), halfLens(), ident(), OOBox(), operator=(), and operator==().

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

- [OOBox.h](#)

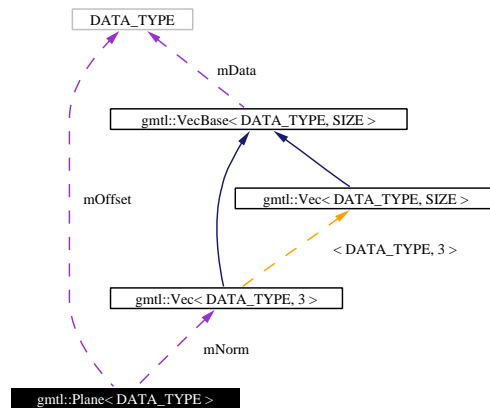


## 10.10 gmtl::Plane< DATA\_TYPE > Class Template Reference

**Plane:** Defines a geometrical plane.

```
#include <Plane.h>
```

Collaboration diagram for gmtl::Plane< DATA\_TYPE >:



### Public Methods

- **Plane** ()  
*Creates an uninitialized **Plane**.*
- **Plane** (const **Point**< DATA\_TYPE, 3 > &pt1, const **Point**< DATA\_TYPE, 3 > &pt2, const **Point**< DATA\_TYPE, 3 > &pt3)  
*Creates a plane that the given points lie on.*
- **Plane** (const **Vec**< DATA\_TYPE, 3 > &norm, const **Point**< DATA\_TYPE, 3 > &pt)  
*Creates a plane with the given normal on which pt resides.*
- **Plane** (const **Vec**< DATA\_TYPE, 3 > &norm, const DATA\_TYPE &dPlane-Const)  
*Creates a plane with the given normal and offset.*

- [Plane](#) (const Plane< DATA\_TYPE > &plane)  
*Creates an exact duplicate of the given plane.*
- const [Vec](#)< DATA\_TYPE, 3 > & [getNormal](#) () const  
*Gets the normal for this plane.*
- void [setNormal](#) (const [Vec](#)< DATA\_TYPE, 3 > &norm)  
*Sets the normal for this plane to the given vector.*
- const DATA\_TYPE & [getOffset](#) () const  
*Gets the offset of this plane from the origin such that the offset is the negative distance from the origin.*
- void [setOffset](#) (const DATA\_TYPE &offset)  
*Sets the offset of this plane from the origin.*

## Public Attributes

- [Vec](#)< DATA\_TYPE, 3 > [mNorm](#)  
*The normal for this vector.*
- DATA\_TYPE [mOffset](#)  
*This plane's offset from the origin such that for any point pt,  $\text{dot}(pt, mNorm) = mOffset$ .*

### 10.10.1 Detailed Description

**template<class DATA\_TYPE> class gmtl::Plane< DATA\_TYPE >**

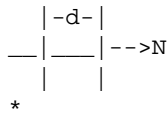
[Plane](#): Defines a geometrical plane.

All points on the plane satisfy the equation  $\text{dot}(\text{Pt}, \text{Normal}) = \text{offset}$  normal is assumed to be normalized

NOTE: Some plane implementation store D instead of offset. Thus those implementation have opposite sign from what we have

pg. 309 Computer Graphics 2nd Edition Hearn Baker

$$\begin{array}{l} N \cdot P = -D \\ | \end{array}$$



Definition at line 65 of file Plane.h.

## 10.10.2 Constructor & Destructor Documentation

### 10.10.2.1 `template<class DATA_TYPE> gmtl::Plane< DATA_TYPE >::Plane()` [inline]

Creates an uninitialized [Plane](#).

In other words, the normal is (0,0,0) and the offset is 0.

Definition at line 72 of file Plane.h.

References `gmtl::Plane< DATA_TYPE >::mOffset`.

```
73         : mOffset( 0 )
74     {}
```

### 10.10.2.2 `template<class DATA_TYPE> gmtl::Plane< DATA_TYPE >::Plane(const Point< DATA_TYPE, 3 > &pt1, const Point< DATA_TYPE, 3 > &pt2, const Point< DATA_TYPE, 3 > &pt3)` [inline]

Creates a plane that the given points lie on.

#### Parameters:

*pt1* a point on the plane

*pt2* a point on the plane

*pt3* a point on the plane

Definition at line 83 of file Plane.h.

References `gmtl::cross()`, `gmtl::dot()`, `gmtl::Plane< DATA_TYPE >::mNorm`, `gmtl::Plane< DATA_TYPE >::mOffset`, and `gmtl::normalize()`.

```
85     {
```

```

86     Vec<DATA_TYPE, 3> vec12( pt2-pt1 );
87     Vec<DATA_TYPE, 3> vec13( pt3-pt1 );
88
89     mNorm = cross( vec12, vec13 );
90     normalize( mNorm );
91
92     mOffset = dot( static_cast< Vec<DATA_TYPE, 3> >(pt1), mNorm ); // Graphics Gems I: Pa
93 }

```

**10.10.2.3** `template<class DATA_TYPE> gmtl::Plane< DATA_TYPE >::Plane`  
`(const Vec< DATA_TYPE, 3 > & norm, const Point< DATA_TYPE, 3`  
`> & pt) [inline]`

Creates a plane with the given normal on which pt resides.

**Parameters:**

*norm* the normal of the plane  
*pt* a point that lies on the plane

Definition at line 101 of file Plane.h.

References gmtl::dot(), gmtl::Plane< DATA\_TYPE >::mNorm, and gmtl::Plane< DATA\_TYPE >::mOffset.

```

102     : mNorm( norm )
103     {
104     mOffset = dot( static_cast< Vec<DATA_TYPE, 3> >(pt), norm );
105     }

```

**10.10.2.4** `template<class DATA_TYPE> gmtl::Plane< DATA_TYPE >::Plane`  
`(const Vec< DATA_TYPE, 3 > & norm, const DATA_TYPE &`  
`dPlaneConst) [inline]`

Creates a plane with the given normal and offset.

**Parameters:**

*norm* the normal of the plane  
*dPlaneConst* the plane offset constant

Definition at line 113 of file Plane.h.

References gmtl::Plane< DATA\_TYPE >::mNorm, and gmtl::Plane< DATA\_TYPE >::mOffset.

```

114         : mNorm( norm ), mOffset( dPlaneConst )
115     {}

```

#### 10.10.2.5 `template<class DATA_TYPE> gmtl::Plane< DATA_TYPE >::Plane` `(const Plane< DATA_TYPE > &plane) [inline]`

Creates an exact duplicate of the given plane.

##### Parameters:

*plane* the plane to copy

Definition at line 122 of file Plane.h.

References gmtl::Plane< DATA\_TYPE >::mNorm, and gmtl::Plane< DATA\_TYPE >::mOffset.

```

123         : mNorm( plane.mNorm ), mOffset( plane.mOffset )
124     {}

```

### 10.10.3 Member Function Documentation

#### 10.10.3.1 `template<class DATA_TYPE> const Vec<DATA_TYPE, 3>&` `gmtl::Plane< DATA_TYPE >::getNormal() const [inline]`

Gets the normal for this plane.

##### Returns:

this plane's normal

Definition at line 131 of file Plane.h.

References gmtl::Plane< DATA\_TYPE >::mNorm.

```

132     {
133         return mNorm;
134     }

```

### 10.10.3.2 `template<class DATA_TYPE> const DATA_TYPE& gmtl::Plane< DATA_TYPE >::getOffset () const` [inline]

Gets the offset of this plane from the origin such that the offset is the negative distance from the origin.

**Returns:**

this plane's offset

Definition at line 154 of file Plane.h.

References gmtl::Plane< DATA\_TYPE >::mOffset.

```
155     {
156         return mOffset;
157     }
```

### 10.10.3.3 `template<class DATA_TYPE> void gmtl::Plane< DATA_TYPE >::setNormal (const Vec< DATA_TYPE, 3 > & norm)` [inline]

Sets the normal for this plane to the given vector.

**Parameters:**

*norm* the new normal

**Precondition:**

$|\text{norm}| = 1$

Definition at line 143 of file Plane.h.

References gmtl::Plane< DATA\_TYPE >::mNorm.

```
144     {
145         mNorm = norm;
146     }
```

### 10.10.3.4 `template<class DATA_TYPE> void gmtl::Plane< DATA_TYPE >::setOffset (const DATA_TYPE & offset)` [inline]

Sets the offset of this plane from the origin.

**Parameters:***offset* the new offset

Definition at line 164 of file Plane.h.

References gmtl::Plane&lt; DATA\_TYPE &gt;::mOffset.

```

165     {
166         mOffset = offset;
167     }

```

**10.10.4 Member Data Documentation****10.10.4.1 template<class DATA\_TYPE> [Vec<DATA\\_TYPE, 3>](#) gmtl::Plane< DATA\_TYPE >::mNorm**

The normal for this vector.

For any point on the plane,  $\text{dot}(\text{pt}, \text{mNorm}) = \text{mOffset}$ .

Definition at line 175 of file Plane.h.

Referenced by gmtl::Plane&lt; DATA\_TYPE &gt;::getNormal(), gmtl::Plane&lt; DATA\_TYPE &gt;::Plane(), and gmtl::Plane&lt; DATA\_TYPE &gt;::setNormal().

**10.10.4.2 template<class DATA\_TYPE> DATA\_TYPE gmtl::Plane< DATA\_TYPE >::mOffset**This plane's offset from the origin such that for any point  $\text{pt}$ ,  $\text{dot}(\text{pt}, \text{mNorm}) = \text{mOffset}$ .Note that  $\text{mOffset} = -D$  (neg dist from the origin).

Definition at line 182 of file Plane.h.

Referenced by gmtl::Plane&lt; DATA\_TYPE &gt;::getOffset(), gmtl::Plane&lt; DATA\_TYPE &gt;::Plane(), and gmtl::Plane&lt; DATA\_TYPE &gt;::setOffset().

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

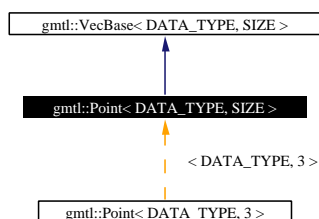
- [Plane.h](#)

## 10.11 gmtl::Point< DATA\_TYPE, SIZE > Class Template Reference

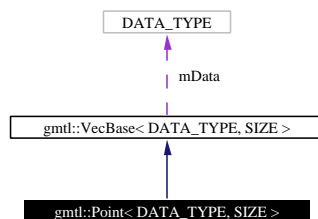
**Point** Use points when you need to represent a position.

```
#include <Point.h>
```

Inheritance diagram for gmtl::Point:



Collaboration diagram for gmtl::Point< DATA\_TYPE, SIZE >:



### Public Types

- typedef DATA\_TYPE [DataType](#)  
*The datatype used for the components of this [VecBase](#).*
- typedef [VecBase](#)< DATA\_TYPE, SIZE > [BaseType](#)  
*Placeholder for the base type.*
- enum { [Size](#) = SIZE }



## Public Methods

- [Point](#) ()

*Default constructor.*

### Value constructors

*Construct with copy of rVec*

- [Point](#) (const Point< DATA\_TYPE, SIZE > &rVec)
- [Point](#) (const [VecBase](#)< DATA\_TYPE, SIZE > &rVec)
- [Point](#) (const DATA\_TYPE &val0, const DATA\_TYPE &val1)
- [Point](#) (const DATA\_TYPE &val0, const DATA\_TYPE &val1, const DATA\_TYPE &val2)
- [Point](#) (const DATA\_TYPE &val0, const DATA\_TYPE &val1, const DATA\_TYPE &val2, const DATA\_TYPE &val3)

### 10.11.1 Detailed Description

**template<class DATA\_TYPE, unsigned SIZE> class gmtl::Point< DATA\_TYPE, SIZE >**

[Point](#) Use points when you need to represent a position.

Don't use points to represent a Vector. One difference you should note is that ceratain matrix operations are different between [Point](#) and [Vec](#) such as xform and operator \*. A [Vec](#) xform by matrix is simply a rotation, while a [Point](#) xformed by a matrix is a full matrix transform (rotation, skew, translation, scale).

**See also:**

[Point3f](#) , [Point4f](#) , [Point3d](#) , [Point4f](#)

Definition at line 58 of file Point.h.

### 10.11.2 Member Typedef Documentation

**10.11.2.1    template<class DATA\_TYPE, unsigned SIZE> typedef  
              [VecBase](#)<DATA\_TYPE, SIZE> gmtl::Point< DATA\_TYPE, SIZE  
              >::BaseType**

Placeholder for the base type.

Definition at line 65 of file Point.h.

### 10.11.2.2 `template<class DATA_TYPE, unsigned SIZE> typedef DATA_TYPE gmtl::Point< DATA_TYPE, SIZE >::DataType`

The datatype used for the components of this [VecBase](#).

Reimplemented from [gmtl::VecBase< DATA\\_TYPE, SIZE >](#).

Definition at line 61 of file Point.h.

## 10.11.3 Member Enumeration Documentation

### 10.11.3.1 `template<class DATA_TYPE, unsigned SIZE> anonymous enum`

Enumeration values:

**Size**

Definition at line 62 of file Point.h.

```
62 { Size = SIZE };
```

## 10.11.4 Constructor & Destructor Documentation

### 10.11.4.1 `template<class DATA_TYPE, unsigned SIZE> gmtl::Point< DATA_TYPE, SIZE >::Point () [inline]`

Default constructor.

Definition at line 70 of file Point.h.

```
71 {
72     for (unsigned i = 0; i < SIZE; ++i)
73         mData[i] = (DATA_TYPE)0;
74 }
```

### 10.11.4.2 `template<class DATA_TYPE, unsigned SIZE> gmtl::Point< DATA_TYPE, SIZE >::Point (const Point< DATA_TYPE, SIZE > & rVec) [inline]`

Definition at line 80 of file Point.h.

```

81      : BaseType(static_cast<BaseType>(rVec))
82      {;}

```

**10.11.4.3** `template<class DATA_TYPE, unsigned SIZE> gmtl::Point< DATA_TYPE, SIZE >::Point (const VecBase< DATA_TYPE, SIZE > & rVec) [inline]`

Definition at line 83 of file Point.h.

```

84      : BaseType(rVec)
85      {;}

```

**10.11.4.4** `template<class DATA_TYPE, unsigned SIZE> gmtl::Point< DATA_TYPE, SIZE >::Point (const DATA_TYPE & val0, const DATA_TYPE & val1) [inline]`

Definition at line 86 of file Point.h.

```

87      : BaseType(val0, val1)
88      {
89          // @todo need compile time assert
90          gmtlASSERT( SIZE == 2 && "out of bounds element access in Point" );
91      }

```

**10.11.4.5** `template<class DATA_TYPE, unsigned SIZE> gmtl::Point< DATA_TYPE, SIZE >::Point (const DATA_TYPE & val0, const DATA_TYPE & val1, const DATA_TYPE & val2) [inline]`

Definition at line 93 of file Point.h.

```

94      : BaseType(val0, val1, val2)
95      {
96          // @todo need compile time assert
97          gmtlASSERT( SIZE == 3 && "out of bounds element access in Point" );
98      }

```

**10.11.4.6** `template<class DATA_TYPE, unsigned SIZE> gmtl::Point<  
DATA_TYPE, SIZE >::Point (const DATA_TYPE & val0, const  
DATA_TYPE & val1, const DATA_TYPE & val2, const DATA_TYPE  
& val3) [inline]`

Definition at line 100 of file Point.h.

```
101     : BaseType(val0, val1, val2, val3)
102     {
103         // @todo need compile time assert
104         gmtlASSERT( SIZE == 4 && "out of bounds element access in Point" );
105     }
```

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

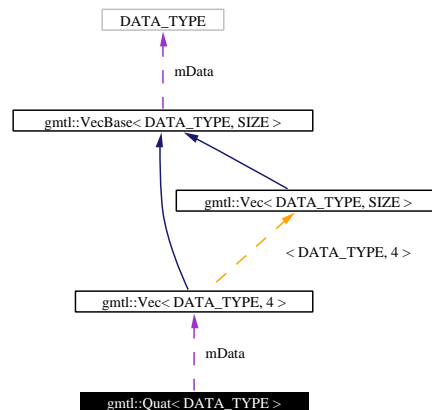
- [Point.h](#)

## 10.12 gmtl::Quat< DATA\_TYPE > Class Template Reference

**Quat**: Class to encapsulate quaternion behaviors.

```
#include <Quat.h>
```

Collaboration diagram for gmtl::Quat< DATA\_TYPE >:



### Public Types

- typedef `DATA_TYPE` `DataType`  
use this to declare single value types of the same type as this matrix.

### Public Methods

- `Quat` (`const DATA_TYPE x=(DATA_TYPE) 0.0, const DATA_TYPE y=(DATA_TYPE) 0.0, const DATA_TYPE z=(DATA_TYPE) 0.0, const DATA_TYPE w=(DATA_TYPE) 1.0`)  
default constructor, initializes to quaternion multiplication identity  $[x,y,z,w] == [0,0,0,1]$ .
- `Quat` (`const Quat< DATA_TYPE > &q`)  
copy constructor.

- void [set](#) (const DATA\_TYPE x, const DATA\_TYPE y, const DATA\_TYPE z, const DATA\_TYPE w)
- void [get](#) (DATA\_TYPE &x, DATA\_TYPE &y, DATA\_TYPE &z, DATA\_TYPE &w)  
*get the raw data elements of the quaternion.*
- DATA\_TYPE & [operator\[\]](#) (const int x)  
*bracket operator.*
- const DATA\_TYPE & [operator\[\]](#) (const int x) const  
*bracket operator(const version).*
- const DATA\_TYPE \* [getData](#) () const  
*Get a DATA\_TYPE pointer to the quat internal data.*

## Public Attributes

- [Vec](#)< DATA\_TYPE, 4 > [mData](#)

### 10.12.1 Detailed Description

**template<typename DATA\_TYPE> class gmtl::Quat< DATA\_TYPE >**

[Quat](#): Class to encapsulate quaternion behaviors.

this Quaternion is ordered in memory: x,y,z,w.

**See also:**

[Quatf](#) , [Quatd](#)

Note: The code for most of these routines was built using the following references

References:

- Advanced Animation and Rendering Techniques: pp363-365
- Animating Rotation with Quaternion Curves, Ken Shoemake, SIGGRAPH Proceedings Vol 19, Number 3, 1985
- Quaternion Calculus for Animation, Ken Shoemake SIGGRAPH course notes 1989
- Game Developer Magazine: Feb 98, pg.34-42
- Motivation for the use of Quaternions to perform transformations  
<http://www.rust.net/~kgeoinfo/quat1.htm>

- On quaternions; or on a new system of imaginaries in algebra, Sir William Rowan Hamilton, Philosophical Magazine, xxv, pp. 10-13 (July 1844)
- You also can find more on quaternions at
  - [http://www.gamasutra.com/features/19980703/quaternions\\_01.htm](http://www.gamasutra.com/features/19980703/quaternions_01.htm) and at
  - <http://archive.ncsa.uiuc.edu/VEG/VPS/emtc/quaternions/index.html>
- Or search on google....

Definition at line 76 of file Quat.h.

## 10.12.2 Member Typedef Documentation

### 10.12.2.1 `template<typename DATA_TYPE> typedef DATA_TYPE gmtl::Quat< DATA_TYPE >::DataType`

use this to declare single value types of the same type as this matrix.

Definition at line 81 of file Quat.h.

## 10.12.3 Constructor & Destructor Documentation

### 10.12.3.1 `template<typename DATA_TYPE> gmtl::Quat< DATA_TYPE >::Quat (const DATA_TYPE x = (DATA_TYPE)0.0, const DATA_TYPE y = (DATA_TYPE)0.0, const DATA_TYPE z = (DATA_TYPE)0.0, const DATA_TYPE w = (DATA_TYPE)1.0) [inline]`

default constructor, initializes to quaternion multiplication identity [x,y,z,w] == [0,0,0,1].

NOTE: the addition identity is [0,0,0,0]

### 10.12.3.2 `template<typename DATA_TYPE> gmtl::Quat< DATA_TYPE >::Quat (const Quat< DATA_TYPE > & q) [inline]`

copy constructor.

## 10.12.4 Member Function Documentation

**10.12.4.1** `template<typename DATA_TYPE> void gmtl::Quat< DATA_TYPE >::get (DATA_TYPE & x, DATA_TYPE & y, DATA_TYPE & z, DATA_TYPE & w) [inline]`

get the raw data elements of the quaternion.

**Postcondition:**

returns  $[\sin(\theta/2) * x, \sin(\theta/2) * y, \sin(\theta/2) * z, \cos(\theta/2)]$  with  $\theta$  in radians.

Definition at line 107 of file Quat.h.

References `gmtl::Quat< DATA_TYPE >::mData`, `gmtl::Welt`, `gmtl::Xelt`, `gmtl::Yelt`, and `gmtl::Zelt`.

```

108     {
109         x = mData[Xelt];
110         y = mData[Yelt];
111         z = mData[Zelt];
112         w = mData[Welt];
113     }
```

**10.12.4.2** `template<typename DATA_TYPE> const DATA_TYPE* gmtl::Quat< DATA_TYPE >::getData () const [inline]`

Get a DATA\_TYPE pointer to the quat internal data.

**Postcondition:**

Returns a ptr to the head of the quat data

Definition at line 155 of file Quat.h.

References `gmtl::VecBase< DATA_TYPE, SIZE >::getData()`, and `gmtl::Quat< DATA_TYPE >::mData`.

```

155 { return (DATA_TYPE*)mData.getData(); }
```



**10.12.4.3** ]

```
template<typename DATA_TYPE> const DATA_TYPE& gmtl::Quat< DATA_TYPE
>::operator[] (const int x) const [inline]
```

bracket operator(const version).

raw data accessor.

**”Example (access raw data element in a [Quat](#)):”**

```
Quatf q;
float rads = acos( q[Welt] ) / 2.0f;
```

**See also:**

[VectorIndex](#)

Definition at line 146 of file Quat.h.

References gmtlASSERT, and gmtl::Quat< DATA\_TYPE >::mData.

```
147 {
148     gmtlASSERT( x >= 0 && x < 4 && "out of bounds error" );
149     return mData[x];
150 }
```

**10.12.4.4** ]

```
template<typename DATA_TYPE> DATA_TYPE& gmtl::Quat< DATA_TYPE
>::operator[] (const int x) [inline]
```

bracket operator.

raw data accessor.

**”Example (access raw data element in a [Quat](#)):”**

```
Quatf q;
q[Xelt] = 0.001231176f;
q[Yelt] = 0.1222f;
q[Zelt] = 0.721f;
q[Welt] = 0.982323f;
```

**See also:**

[VectorIndex](#)

Definition at line 129 of file Quat.h.

References `gmmlASSERT`, and `gmml::Quat< DATA_TYPE >::mData`.

```

130    {
131        gmmlASSERT( x >= 0 && x < 4 && "out of bounds error" );
132        return mData[x];
133    }
```

#### 10.12.4.5 `template<typename DATA_TYPE> void gmml::Quat< DATA_TYPE >::set (const DATA_TYPE x, const DATA_TYPE y, const DATA_TYPE z, const DATA_TYPE w) [inline]`

Definition at line 99 of file Quat.h.

References `gmml::Quat< DATA_TYPE >::mData`, and `gmml::VecBase< DATA_TYPE, SIZE >::set()`.

```

100    {
101        mData.set( x, y, z, w );
102    }
```

### 10.12.5 Member Data Documentation

#### 10.12.5.1 `template<typename DATA_TYPE> Vec<DATA_TYPE, 4> gmml::Quat< DATA_TYPE >::mData`

Definition at line 159 of file Quat.h.

Referenced by `gmml::Quat< DATA_TYPE >::get()`, `gmml::Quat< DATA_TYPE >::getData()`, `gmml::Quat< DATA_TYPE >::operator[]()`, and `gmml::Quat< DATA_TYPE >::set()`.

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

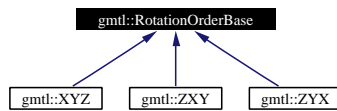
- [Quat.h](#)

## 10.13 gmtl::RotationOrderBase Struct Reference

Base class for Rotation orders.

```
#include <Math.h>
```

Inheritance diagram for gmtl::RotationOrderBase:



### Public Types

- enum { [IS\\_ROTORDER](#) = 1 }

#### 10.13.1 Detailed Description

Base class for Rotation orders.

See also:

[XYZ](#), [ZYX](#), [ZXY](#)

Definition at line 49 of file Math.h.

#### 10.13.2 Member Enumeration Documentation

##### 10.13.2.1 anonymous enum

Enumeration values:

**IS\_ROTORDER**

Definition at line 49 of file Math.h.

```
49 { enum { IS_ROTORDER = 1 }; };
```

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

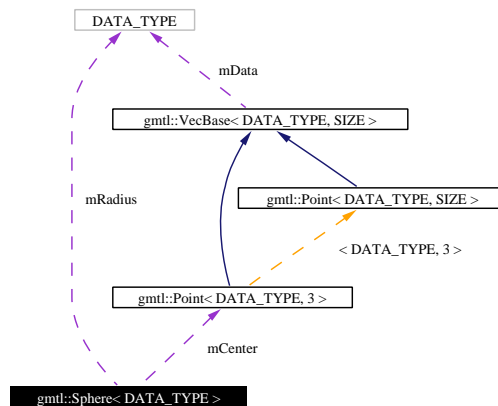
- [Math.h](#)

## 10.14 gmtl::Sphere< DATA\_TYPE > Class Template Reference

Describes a sphere in 3D space by its center point and its radius.

```
#include <Sphere.h>
```

Collaboration diagram for gmtl::Sphere< DATA\_TYPE >:



### Public Types

- typedef DATA\_TYPE [DataType](#)

### Public Methods

- [Sphere](#) ()  
*Constructs a sphere centered at the origin with a radius of 0.*
- [Sphere](#) (const [Point](#)< DATA\_TYPE, 3 > &center, const DATA\_TYPE &radius)  
*Constructs a sphere with the given center and radius.*
- [Sphere](#) (const Sphere< DATA\_TYPE > &sphere)  
*Constructs a duplicate of the given sphere.*

- `const Point< DATA_TYPE, 3 > & getCenter () const`  
*Gets the center of the sphere.*
- `const DATA_TYPE & getRadius () const`  
*Gets the radius of the sphere.*
- `void setCenter (const Point< DATA_TYPE, 3 > &center)`  
*Sets the center point of the sphere.*
- `void setRadius (const DATA_TYPE &radius)`  
*Sets the radius of the sphere.*

### Public Attributes

- `Point< DATA_TYPE, 3 > mCenter`  
*The center of the sphere.*
- `DATA_TYPE mRadius`  
*The radius of the sphere.*

### 10.14.1 Detailed Description

`template<class DATA_TYPE> class gmtl::Sphere< DATA_TYPE >`

Describes a sphere in 3D space by its center point and its radius.

#### Parameters:

*DATA\_TYPE* the internal type used for the point and radius

Definition at line 50 of file Sphere.h.

### 10.14.2 Member Typedef Documentation

**10.14.2.1** `template<class DATA_TYPE> typedef DATA_TYPE gmtl::Sphere< DATA_TYPE >::DataType`

Definition at line 53 of file Sphere.h.

### 10.14.3 Constructor & Destructor Documentation

#### 10.14.3.1 `template<class DATA_TYPE> gmtl::Sphere< DATA_TYPE >::Sphere() [inline]`

Constructs a sphere centered at the origin with a radius of 0.

Definition at line 59 of file Sphere.h.

References gmtl::Sphere< DATA\_TYPE >::mRadius.

```
60      : mRadius( 0 )
61      {}
```

#### 10.14.3.2 `template<class DATA_TYPE> gmtl::Sphere< DATA_TYPE >::Sphere(const Point< DATA_TYPE, 3 > & center, const DATA_TYPE & radius) [inline]`

Constructs a sphere with the given center and radius.

##### Parameters:

*center* the point at which to center the sphere

*radius* the radius of the sphere

Definition at line 69 of file Sphere.h.

References gmtl::center(), gmtl::Sphere< DATA\_TYPE >::mCenter, and gmtl::Sphere< DATA\_TYPE >::mRadius.

```
70      : mCenter( center ), mRadius( radius )
71      {}
```

#### 10.14.3.3 `template<class DATA_TYPE> gmtl::Sphere< DATA_TYPE >::Sphere(const Sphere< DATA_TYPE > & sphere) [inline]`

Constructs a duplicate of the given sphere.

##### Parameters:

*sphere* the sphere to make a copy of

Definition at line 78 of file Sphere.h.

References gmtl::Sphere< DATA\_TYPE >::mCenter, and gmtl::Sphere< DATA\_TYPE >::mRadius.

```
79         : mCenter( sphere.mCenter ), mRadius( sphere.mRadius )
80     {}
```

## 10.14.4 Member Function Documentation

### 10.14.4.1 `template<class DATA_TYPE> const Point<DATA_TYPE, 3>& gmtl::Sphere< DATA_TYPE >::getCenter () const` [inline]

Gets the center of the sphere.

**Returns:**

the center point of the sphere

Definition at line 87 of file Sphere.h.

References gmtl::Sphere< DATA\_TYPE >::mCenter.

```
88     {
89         return mCenter;
90     }
```

### 10.14.4.2 `template<class DATA_TYPE> const DATA_TYPE& gmtl::Sphere< DATA_TYPE >::getRadius () const` [inline]

Gets the radius of the sphere.

**Returns:**

the radius of the sphere

Definition at line 97 of file Sphere.h.

References gmtl::Sphere< DATA\_TYPE >::mRadius.

```
98     {
99         return mRadius;
100    }
```



**10.14.4.3** `template<class DATA_TYPE> void gmtl::Sphere< DATA_TYPE  
>::setCenter (const Point< DATA_TYPE, 3 > &center) [inline]`

Sets the center point of the sphere.

**Parameters:**

*center* the new point at which to center the sphere

Definition at line 107 of file Sphere.h.

References gmtl::center(), and gmtl::Sphere< DATA\_TYPE >::mCenter.

```
108    {
109        mCenter = center;
110    }
```

**10.14.4.4** `template<class DATA_TYPE> void gmtl::Sphere< DATA_TYPE  
>::setRadius (const DATA_TYPE &radius) [inline]`

Sets the radius of the sphere.

**Parameters:**

*radius* the new radius of the sphere

Definition at line 117 of file Sphere.h.

References gmtl::Sphere< DATA\_TYPE >::mRadius.

```
118    {
119        mRadius = radius;
120    }
```

## 10.14.5 Member Data Documentation

**10.14.5.1** `template<class DATA_TYPE> Point<DATA_TYPE, 3>  
gmtl::Sphere< DATA_TYPE >::mCenter`

The center of the sphere.

Definition at line 126 of file Sphere.h.

Referenced by gmtl::extendVolume(), gmtl::Sphere< DATA\_TYPE >::getCenter(), gmtl::isInVolume(), gmtl::isOnVolume(), gmtl::makeVolume(), gmtl::Sphere< DATA\_TYPE >::setCenter(), and gmtl::Sphere< DATA\_TYPE >::Sphere().

#### 10.14.5.2 `template<class DATA_TYPE> DATA_TYPE gmtl::Sphere<DATA_TYPE >::mRadius`

The radius of the sphere.

Definition at line 131 of file Sphere.h.

Referenced by `gmtl::extendVolume()`, `gmtl::Sphere< DATA_TYPE >::getRadius()`, `gmtl::isInVolume()`, `gmtl::isOnVolume()`, `gmtl::makeVolume()`, `gmtl::Sphere< DATA_TYPE >::setRadius()`, and `gmtl::Sphere< DATA_TYPE >::Sphere()`.

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

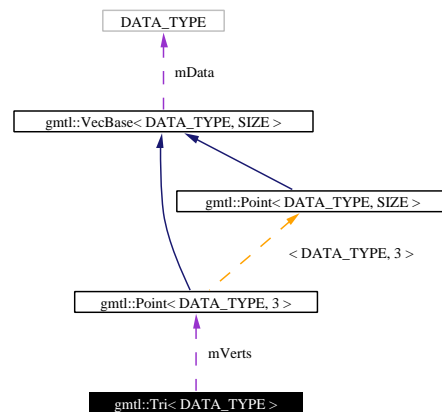
- [Sphere.h](#)

## 10.15 gmtl::Tri< DATA\_TYPE > Class Template Reference

This class defines a triangle as a set of 3 points order in CCW fashion.

```
#include <Tri.h>
```

Collaboration diagram for gmtl::Tri< DATA\_TYPE >:



### Public Methods

- **Tri ()**  
Constructs a new triangle with all vertices at the origin.
- **Tri (const Point< DATA\_TYPE, 3 > &p1, const Point< DATA\_TYPE, 3 > &p2, const Point< DATA\_TYPE, 3 > &p3)**  
Constructs a new triangle with the given points.
- **Tri (const Tri< DATA\_TYPE > &tri)**  
Constructs a duplicate of the given triangle.
- **Vec< DATA\_TYPE, 3 > edge (int idx) const**  
Gets the nth edge of the triangle where edge0 corresponds to the vector from vertex 0 to 1, edge1 corresponds to the vector from vertex 1 to 2 and edge2 corresponds to the vector from vertex 2 to vertex 0.

- `Point< DATA_TYPE, 3 > & operator[] (int idx)`
- `const Point< DATA_TYPE, 3 > & operator[] (int idx) const`

### 10.15.1 Detailed Description

`template<class DATA_TYPE> class gmtl::Tri< DATA_TYPE >`

This class defines a triangle as a set of 3 points order in CCW fashion.

Triangle points are  $\text{tri}(s,t) = b+s*e_0+t*e_1$  where  $0 \leq s \leq 1$ ,  $0 \leq t \leq 1$ , and  $0 \leq s+t \leq 1$ .

Definition at line 52 of file Tri.h.

### 10.15.2 Constructor & Destructor Documentation

**10.15.2.1** `template<class DATA_TYPE> gmtl::Tri< DATA_TYPE >::Tri ()`  
[inline]

Constructs a new triangle with all vertices at the origin.

Definition at line 58 of file Tri.h.

```
58 {}
```

**10.15.2.2** `template<class DATA_TYPE> gmtl::Tri< DATA_TYPE >::Tri (const`  
`Point< DATA_TYPE, 3 > & p1, const Point< DATA_TYPE, 3 > & p2,`  
`const Point< DATA_TYPE, 3 > & p3) [inline]`

Constructs a new triangle with the given points.

The points must be passed in in CCW order.

**Parameters:**

*p1* vertex0

*p2* vertex1

*p3* vertex2

**Precondition:**

p1, p2, p3 must be in CCW order

Definition at line 70 of file Tri.h.

```
72    {  
73        mVerts[0] = p1;  
74        mVerts[1] = p2;  
75        mVerts[2] = p3;  
76    }
```

### 10.15.2.3 `template<class DATA_TYPE> gmtl::Tri< DATA_TYPE >::Tri (const Tri< DATA_TYPE > & tri) [inline]`

Constructs a duplicate of the given triangle.

**Parameters:**

*tri* the triangle to copy

Definition at line 83 of file Tri.h.

```
84    {  
85        mVerts[0] = tri[0];  
86        mVerts[1] = tri[1];  
87        mVerts[2] = tri[2];  
88    }
```

## 10.15.3 Member Function Documentation

### 10.15.3.1 `template<class DATA_TYPE> Vec<DATA_TYPE, 3> gmtl::Tri< DATA_TYPE >::edge (int idx) const [inline]`

Gets the nth edge of the triangle where edge0 corresponds to the vector from vertex 0 to 1, edge1 corresponds to the vector from vertex 1 to 2 and edge2 corresponds to the vector from vertex 2 to vertex 0.

**Parameters:**

*idx* the ordered edge index

**Precondition:**

$0 \leq \text{idx} \leq 2$

**Returns:**

a vector from vertex idx to vertex (idx+1)mod size

Definition at line 121 of file Tri.h.

References gmtlASSERT.

```

122    {
123        gmtlASSERT( (0 <= idx) && (idx <= 2) );
124        int idx2 = ( idx == 2 ) ? 0 : idx + 1;
125        return (mVerts[idx2] - mVerts[idx]);
126    }
```

### 10.15.3.2 ]

template<class DATA\_TYPE> const [Point](#)<DATA\_TYPE, 3>& gmtl::Tri< DATA\_TYPE >::operator[] (int *idx*) const [inline]

Definition at line 104 of file Tri.h.

References gmtlASSERT.

```

105    {
106        gmtlASSERT( (0 <= idx) && (idx <= 2) );
107        return mVerts[idx];
108    }
```

### 10.15.3.3 ]

template<class DATA\_TYPE> [Point](#)<DATA\_TYPE, 3>& gmtl::Tri< DATA\_TYPE >::operator[] (int *idx*) [inline]

#### Parameters:

*idx* the index to the vertex in the triangle

#### Precondition:

$0 \leq \text{idx} \leq 2$

#### Returns:

the nth vertex

Definition at line 99 of file Tri.h.

References gmtlASSERT.

```
100     {  
101         gmtlASSERT( (0 <= idx) && (idx <= 2) );  
102         return mVerts[idx];  
103     }
```

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

- [Tri.h](#)

## 10.16 gmtl::Type2Type< T > Struct Template Reference

A lightweight identifier you can pass to overloaded functions to typefy them.

```
#include <Meta.h>
```

### Public Types

- typedef T [OriginalType](#)

#### 10.16.1 Detailed Description

```
template<typename T> struct gmtl::Type2Type< T >
```

A lightweight identifier you can pass to overloaded functions to typefy them.

[Type2Type](#) lets you transport the type information about T to functions

Definition at line 49 of file Meta.h.

#### 10.16.2 Member Typedef Documentation

**10.16.2.1** `template<typename T> typedef T gmtl::Type2Type< T >::OriginalType`

Definition at line 51 of file Meta.h.

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

- [Meta.h](#)

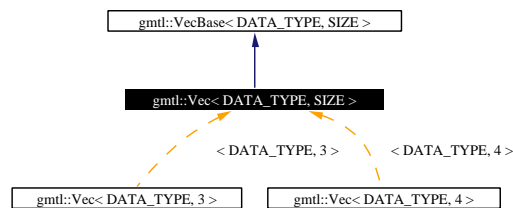


## 10.17 gmtl::Vec< DATA\_TYPE, SIZE > Class Template Reference

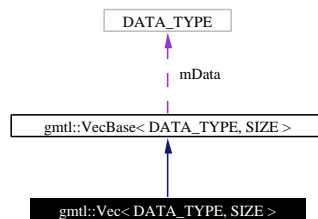
A representation of a vector with SIZE components using DATA\_TYPE as the data type for each component.

```
#include <Vec.h>
```

Inheritance diagram for gmtl::Vec:



Collaboration diagram for gmtl::Vec< DATA\_TYPE, SIZE >:



### Public Types

- typedef DATA\_TYPE [DataType](#)  
The datatype used for the components of this [Vec](#).
- typedef [VecBase](#)< DATA\_TYPE, SIZE > [BaseType](#)  
The superclass type.
- enum { [Size](#) = SIZE }  
The number of components this [Vec](#) has.

## Public Methods

- [Vec](#) ()

*Default constructor.*

### Value constructors

- [Vec](#) (const [Vec](#)< DATA\_TYPE, SIZE > &rVec)  
*Make an exact copy of the given [Vec](#) object.*
- [Vec](#) (const [VecBase](#)< DATA\_TYPE, SIZE > &rVec)
- [Vec](#) (const DATA\_TYPE &val0, const DATA\_TYPE &val1)  
*Creates a new [Vec](#) initialized to the given values.*
- [Vec](#) (const DATA\_TYPE &val0, const DATA\_TYPE &val1, const DATA\_TYPE &val2)
- [Vec](#) (const DATA\_TYPE &val0, const DATA\_TYPE &val1, const DATA\_TYPE &val2, const DATA\_TYPE &val3)

### 10.17.1 Detailed Description

```
template<class DATA_TYPE, unsigned SIZE> class gmtl::Vec< DATA_TYPE,
SIZE >
```

A representation of a vector with SIZE components using DATA\_TYPE as the data type for each component.

#### Parameters:

**DATA\_TYPE** the datatype to use for the components

**SIZE** the number of components this [VecBase](#) has

#### See also:

[Vec3f](#) , [Vec4f](#) , [Vec3d](#) , [Vec4f](#)

Definition at line 56 of file Vec.h.

### 10.17.2 Member Typedef Documentation

**10.17.2.1** `template<class DATA_TYPE, unsigned SIZE> typedef  
VecBase<DATA_TYPE, SIZE> gmtl::Vec< DATA_TYPE, SIZE  
>::BaseType`

The superclass type.

Definition at line 66 of file Vec.h.

**10.17.2.2** `template<class DATA_TYPE, unsigned SIZE> typedef DATA_TYPE  
gmtl::Vec< DATA_TYPE, SIZE >::DataType`

The datatype used for the components of this [Vec](#).

Reimplemented from [gmtl::VecBase< DATA\\_TYPE, SIZE >](#).

Definition at line 60 of file Vec.h.

## 10.17.3 Member Enumeration Documentation

**10.17.3.1** `template<class DATA_TYPE, unsigned SIZE> anonymous enum`

The number of components this [Vec](#) has.

**Enumeration values:**  
**Size**

Definition at line 63 of file Vec.h.

```
63 { Size = SIZE };
```

## 10.17.4 Constructor & Destructor Documentation

**10.17.4.1** `template<class DATA_TYPE, unsigned SIZE> gmtl::Vec<  
DATA_TYPE, SIZE >::Vec () [inline]`

Default constructor.

All components are initialized to zero.

Definition at line 72 of file Vec.h.

```

73     {
74         for (unsigned i = 0; i < SIZE; ++i)
75             mData[i] = (DATA_TYPE)0;
76     }

```

**10.17.4.2** `template<class DATA_TYPE, unsigned SIZE> gmtl::Vec<DATA_TYPE, SIZE >::Vec (const Vec< DATA_TYPE, SIZE > & rVec) [inline]`

Make an exact copy of the given [Vec](#) object.

**Parameters:**

*rVec* the [Vec](#) object to copy

Definition at line 85 of file Vec.h.

```

86         : BaseType( static_cast<BaseType>( rVec ) )
87     {
88     }

```

**10.17.4.3** `template<class DATA_TYPE, unsigned SIZE> gmtl::Vec<DATA_TYPE, SIZE >::Vec (const VecBase< DATA_TYPE, SIZE > & rVec) [inline]`

Definition at line 90 of file Vec.h.

```

91         : BaseType( rVec )
92     {
93     }

```

**10.17.4.4** `template<class DATA_TYPE, unsigned SIZE> gmtl::Vec<DATA_TYPE, SIZE >::Vec (const DATA_TYPE & val0, const DATA_TYPE & val1) [inline]`

Creates a new [Vec](#) initialized to the given values.

Definition at line 98 of file Vec.h.

```
99      : BaseType(val0, val1)
100    {
101        // @todo compile time assert is needed here
102        gmtlASSERT( SIZE == 2 && "out of bounds element access in Point" );
103    }
```

#### 10.17.4.5 `template<class DATA_TYPE, unsigned SIZE> gmtl::Vec< DATA_TYPE, SIZE >::Vec (const DATA_TYPE & val0, const DATA_TYPE & val1, const DATA_TYPE & val2) [inline]`

Definition at line 105 of file Vec.h.

```
106      : BaseType(val0, val1, val2)
107    {
108        // @todo compile time assert is needed here
109        gmtlASSERT( SIZE == 3 && "out of bounds element access in Point" );
110    }
```

#### 10.17.4.6 `template<class DATA_TYPE, unsigned SIZE> gmtl::Vec< DATA_TYPE, SIZE >::Vec (const DATA_TYPE & val0, const DATA_TYPE & val1, const DATA_TYPE & val2, const DATA_TYPE & val3) [inline]`

Definition at line 112 of file Vec.h.

```
113      : BaseType(val0, val1, val2, val3)
114    {
115        // @todo compile time assert is needed here
116        gmtlASSERT( SIZE == 4 && "out of bounds element access in Point" );
117    }
```

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

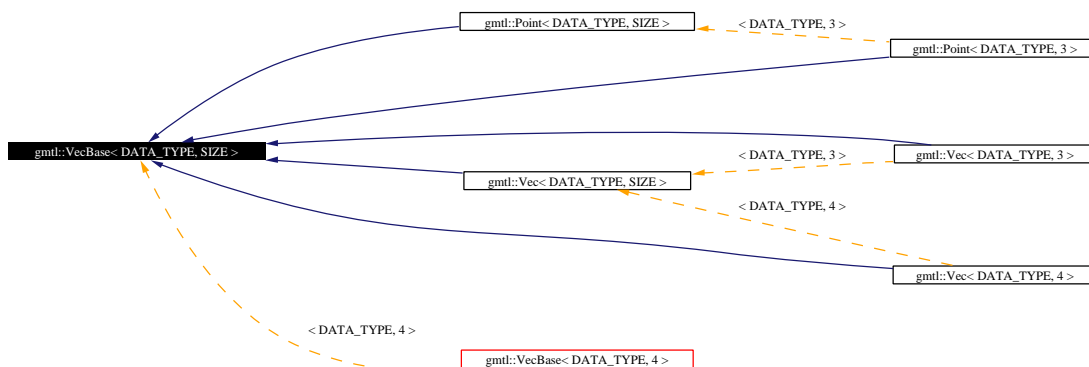
- [Vec.h](#)

## 10.18 gmtl::VecBase< DATA\_TYPE, SIZE > Class Template Reference

Base type for vector-like objects including Points and Vectors.

```
#include <VecBase.h>
```

Inheritance diagram for gmtl::VecBase:



Collaboration diagram for gmtl::VecBase< DATA\_TYPE, SIZE >:



### Public Types

- typedef DATA\_TYPE [DataType](#)

*The datatype used for the components of this [VecBase](#).*

- enum { [Size](#) = SIZE }

*The number of components this [VecBase](#) has.*

## Public Methods

- [VecBase](#) ()  
*Default constructor.*
- [VecBase](#) (const VecBase< DATA\_TYPE, SIZE > &rVec)  
*Makes an exact copy of the given [VecBase](#) object.*
- void [set](#) (const DATA\_TYPE \*dataPtr)  
*Sets the components in this [VecBase](#) using the given array.*
- [VecBase](#) (const DATA\_TYPE &val0, const DATA\_TYPE &val1)  
*Creates a new [VecBase](#) initialized to the given values.*
- [VecBase](#) (const DATA\_TYPE &val0, const DATA\_TYPE &val1, const DATA\_TYPE &val2)
- [VecBase](#) (const DATA\_TYPE &val0, const DATA\_TYPE &val1, const DATA\_TYPE &val2, const DATA\_TYPE &val3)
- void [set](#) (const DATA\_TYPE &val0)  
*Sets the components in this [VecBase](#) to the given values.*
- void [set](#) (const DATA\_TYPE &val0, const DATA\_TYPE &val1)
- void [set](#) (const DATA\_TYPE &val0, const DATA\_TYPE &val1, const DATA\_TYPE &val2)
- void [set](#) (const DATA\_TYPE &val0, const DATA\_TYPE &val1, const DATA\_TYPE &val2, const DATA\_TYPE &val3)
- DATA\_TYPE & [operator\[\]](#) (const unsigned i)  
*Gets the ith component in this [VecBase](#).*
- const DATA\_TYPE & [operator\[\]](#) (const unsigned i) const
- DATA\_TYPE \* [getData](#) ()  
*Gets the internal array of the components.*
- const DATA\_TYPE \* [getData](#) () const

## Public Attributes

- DATA\_TYPE [mData](#) [SIZE]  
*The array of components.*

### 10.18.1 Detailed Description

```
template<class DATA_TYPE, unsigned SIZE> class gmtl::VecBase< DATA_
TYPE, SIZE >
```

Base type for vector-like objects including Points and Vectors.

It is templated on the component datatype as well as the number of components that make it up.

**Parameters:**

*DATA\_TYPE* the datatype to use for the components

*SIZE* the number of components this [VecBase](#) has

Definition at line 52 of file VecBase.h.

### 10.18.2 Member Typedef Documentation

**10.18.2.1** `template<class DATA_TYPE, unsigned SIZE> typedef DATA_TYPE gmtl::VecBase< DATA_TYPE, SIZE >::DataType`

The datatype used for the components of this [VecBase](#).

Reimplemented in [gmtl::Point< DATA\\_TYPE, SIZE >](#), [gmtl::Vec< DATA\\_TYPE, SIZE >](#), [gmtl::Point< DATA\\_TYPE, 3 >](#), [gmtl::Vec< DATA\\_TYPE, 3 >](#), and [gmtl::Vec< DATA\\_TYPE, 4 >](#).

Definition at line 56 of file VecBase.h.

### 10.18.3 Member Enumeration Documentation

**10.18.3.1** `template<class DATA_TYPE, unsigned SIZE> anonymous enum`

The number of components this [VecBase](#) has.

**Enumeration values:**

**Size**

Definition at line 59 of file VecBase.h.

```
59 { Size = SIZE };
```



## 10.18.4 Constructor & Destructor Documentation

### 10.18.4.1 `template<class DATA_TYPE, unsigned SIZE> gmtl::VecBase< DATA_TYPE, SIZE >::VecBase () [inline]`

Default constructor.

Does nothing, leaves data alone. This is for performance because this constructor is called by derived class constructors Even when they just want to set the data directly

Definition at line 68 of file VecBase.h.

```
68 {}
```

### 10.18.4.2 `template<class DATA_TYPE, unsigned SIZE> gmtl::VecBase< DATA_TYPE, SIZE >::VecBase (const VecBase< DATA_TYPE, SIZE > & rVec)`

Makes an exact copy of the given [VecBase](#) object.

**Parameters:**

*rVec* the [VecBase](#) object to copy

Definition at line 144 of file VecBase.h.

References `gmtl::VecBase< DATA_TYPE, SIZE >::mData`.

```
145 {
146     for(unsigned i=0;i<SIZE;++i)
147         mData[i] = rVec.mData[i];
148 }
```

### 10.18.4.3 `template<class DATA_TYPE, unsigned SIZE> gmtl::VecBase< DATA_TYPE, SIZE >::VecBase (const DATA_TYPE & val0, const DATA_TYPE & val1)`

Creates a new [VecBase](#) initialized to the given values.

Definition at line 151 of file VecBase.h.

References `gmtl::ASSERT`, and `gmtl::VecBase< DATA_TYPE, SIZE >::mData`.

```

152 {
153     // @todo need compile time assert
154     gmtlASSERT( SIZE == 2 && "out of bounds element access in VecBase" );
155     mData[0] = val0;
156     mData[1] = val1;
157 }

```

#### 10.18.4.4 `template<class DATA_TYPE, unsigned SIZE> gmtl::VecBase<DATA_TYPE, SIZE >::VecBase (const DATA_TYPE & val0, const DATA_TYPE & val1, const DATA_TYPE & val2)`

Definition at line 160 of file VecBase.h.

References gmtlASSERT, and gmtl::VecBase< DATA\_TYPE, SIZE >::mData.

```

161 {
162     // @todo need compile time assert
163     gmtlASSERT( SIZE == 3 && "out of bounds element access in VecBase" );
164     mData[0] = val0;
165     mData[1] = val1;
166     mData[2] = val2;
167 }

```

#### 10.18.4.5 `template<class DATA_TYPE, unsigned SIZE> gmtl::VecBase<DATA_TYPE, SIZE >::VecBase (const DATA_TYPE & val0, const DATA_TYPE & val1, const DATA_TYPE & val2, const DATA_TYPE & val3)`

Definition at line 170 of file VecBase.h.

References gmtlASSERT, and gmtl::VecBase< DATA\_TYPE, SIZE >::mData.

```

171 {
172     // @todo need compile time assert
173     gmtlASSERT( SIZE == 4 && "out of bounds element access in VecBase" );
174     mData[0] = val0;
175     mData[1] = val1;
176     mData[2] = val2;
177     mData[3] = val3;
178 }

```

### 10.18.5 Member Function Documentation

#### 10.18.5.1 `template<class DATA_TYPE, unsigned SIZE> const DATA_TYPE* gmtl::VecBase< DATA_TYPE, SIZE >::getData() const` [inline]

Definition at line 133 of file VecBase.h.

```
134    { return mData; }
```

#### 10.18.5.2 `template<class DATA_TYPE, unsigned SIZE> DATA_TYPE* gmtl::VecBase< DATA_TYPE, SIZE >::getData()` [inline]

Gets the internal array of the components.

##### Returns:

a pointer to the component array with length SIZE

Definition at line 131 of file VecBase.h.

Referenced by `gmtl::Quat< DATA_TYPE >::getData()`.

```
132    { return mData; }
```

#### 10.18.5.3 ]

`template<class DATA_TYPE, unsigned SIZE> const DATA_TYPE& gmtl::VecBase< DATA_TYPE, SIZE >::operator[] (const unsigned i) const` [inline]

Definition at line 118 of file VecBase.h.

```
119    {
120        gmtlASSERT(i < SIZE);
121        return mData[i];
122    }
```

#### 10.18.5.4 ]

`template<class DATA_TYPE, unsigned SIZE> DATA_TYPE& gmtl::VecBase< DATA_TYPE, SIZE >::operator[] (const unsigned i)` [inline]

Gets the ith component in this [VecBase](#).

**Parameters:**

*i* the zero-based index of the component to access.

**Precondition:**

$i < \text{SIZE}$

**Returns:**

a reference to the *i*th component

Definition at line 113 of file VecBase.h.

```

114     {
115         gmtlASSERT(i < SIZE);
116         return mData[i];
117     }

```

**10.18.5.5** `template<class DATA_TYPE, unsigned SIZE> void gmtl::VecBase<DATA_TYPE, SIZE >::set (const DATA_TYPE & val0, const DATA_TYPE & val1, const DATA_TYPE & val2, const DATA_TYPE & val3) [inline]`

Reimplemented in [gmtl::AxisAngle< DATA\\_TYPE >](#).

Definition at line 210 of file VecBase.h.

References gmtlASSERT, and gmtl::VecBase< DATA\_TYPE, SIZE >::mData.

```

211 {
212     gmtlASSERT( SIZE >= 4 && "out of bounds element access in VecBase" );
213     mData[0] = val0;
214     mData[1] = val1;
215     mData[2] = val2;
216     mData[3] = val3;
217 }

```

**10.18.5.6** `template<class DATA_TYPE, unsigned SIZE> void gmtl::VecBase<DATA_TYPE, SIZE >::set (const DATA_TYPE & val0, const DATA_TYPE & val1, const DATA_TYPE & val2) [inline]`

Definition at line 202 of file VecBase.h.

References gmtlASSERT, and gmtl::VecBase< DATA\_TYPE, SIZE >::mData.

```

203 {
204     gmtlASSERT( SIZE >= 3 && "out of bounds element access in VecBase" );
205     mData[0] = val0;
206     mData[1] = val1;
207     mData[2] = val2;
208 }

```

#### 10.18.5.7 `template<class DATA_TYPE, unsigned SIZE> void gmtl::VecBase< DATA_TYPE, SIZE >::set (const DATA_TYPE & val0, const DATA_TYPE & val1) [inline]`

Definition at line 195 of file VecBase.h.

References `gmtlASSERT`, and `gmtl::VecBase< DATA_TYPE, SIZE >::mData`.

```

196 {
197     gmtlASSERT( SIZE >= 2 && "out of bounds element access in VecBase" );
198     mData[0] = val0;
199     mData[1] = val1;
200 }

```

#### 10.18.5.8 `template<class DATA_TYPE, unsigned SIZE> void gmtl::VecBase< DATA_TYPE, SIZE >::set (const DATA_TYPE & val0) [inline]`

Sets the components in this [VecBase](#) to the given values.

Definition at line 189 of file VecBase.h.

References `gmtlASSERT`, and `gmtl::VecBase< DATA_TYPE, SIZE >::mData`.

```

190 {
191     gmtlASSERT( SIZE >= 1 && "out of bounds element access in VecBase" );
192     mData[0] = val0;
193 }

```

#### 10.18.5.9 `template<class DATA_TYPE, unsigned SIZE> void gmtl::VecBase< DATA_TYPE, SIZE >::set (const DATA_TYPE * dataPtr) [inline]`

Sets the components in this [VecBase](#) using the given array.

**Parameters:**

*dataPtr* the array containing the values to copy

**Precondition:**

dataPtr has at least SIZE elements

Definition at line 183 of file VecBase.h.

References gmtl::VecBase< DATA\_TYPE, SIZE >::mData.

Referenced by gmtl::cross(), and gmtl::Quat< DATA\_TYPE >::set().

```
184 {
185     for(unsigned i=0;i<SIZE;++i)
186         mData[i] = dataPtr[i];
187 }
```

## 10.18.6 Member Data Documentation

### 10.18.6.1 `template<class DATA_TYPE, unsigned SIZE> DATA_TYPE gmtl::VecBase< DATA_TYPE, SIZE >::mData[SIZE]`

The array of components.

Definition at line 139 of file VecBase.h.

Referenced by gmtl::VecBase< DATA\_TYPE, 4 >::getData(), gmtl::VecBase< DATA\_TYPE, 4 >::operator[](), gmtl::Point< DATA\_TYPE, 3 >::Point(), gmtl::VecBase< DATA\_TYPE, SIZE >::set(), gmtl::Vec< DATA\_TYPE, 4 >::Vec(), and gmtl::VecBase< DATA\_TYPE, SIZE >::VecBase().

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

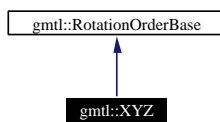
- [VecBase.h](#)

## 10.19 gmtl::XYZ Struct Reference

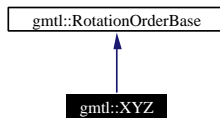
[XYZ](#) Rotation order.

```
#include <Math.h>
```

Inheritance diagram for gmtl::XYZ:



Collaboration diagram for gmtl::XYZ:



### Public Types

- enum { [ID](#) = 0 }

#### 10.19.1 Detailed Description

[XYZ](#) Rotation order.

Definition at line 53 of file Math.h.

#### 10.19.2 Member Enumeration Documentation

##### 10.19.2.1 anonymous enum

Enumeration values:

[ID](#)

Definition at line 53 of file Math.h.

```
53 : public RotationOrderBase { enum { ID = 0 }; };
```

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

- [Math.h](#)

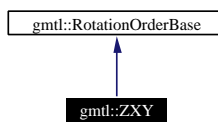


## 10.20 gmtl::ZXY Struct Reference

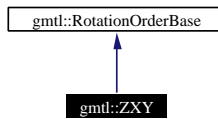
[ZXY](#) Rotation order.

```
#include <Math.h>
```

Inheritance diagram for gmtl::ZXY:



Collaboration diagram for gmtl::ZXY:



### Public Types

- enum { [ID](#) = 2 }

#### 10.20.1 Detailed Description

[ZXY](#) Rotation order.

Definition at line 61 of file Math.h.

#### 10.20.2 Member Enumeration Documentation

##### 10.20.2.1 anonymous enum

Enumeration values:

[ID](#)

Definition at line 61 of file Math.h.

```
61 : public RotationOrderBase { enum { ID = 2 }; };
```

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

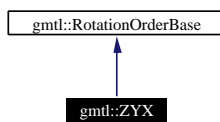
- [Math.h](#)

## 10.21 gmtl::ZYX Struct Reference

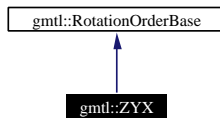
[ZYX](#) Rotation order.

```
#include <Math.h>
```

Inheritance diagram for gmtl::ZYX:



Collaboration diagram for gmtl::ZYX:



### Public Types

- enum { [ID](#) = 1 }

#### 10.21.1 Detailed Description

[ZYX](#) Rotation order.

Definition at line 57 of file Math.h.

#### 10.21.2 Member Enumeration Documentation

##### 10.21.2.1 anonymous enum

Enumeration values:

[ID](#)

Definition at line 57 of file Math.h.

```
57 : public RotationOrderBase { enum { ID = 1 }; };
```

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

- [Math.h](#)

---

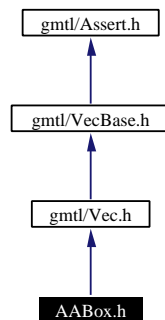
## Chapter 11

# GenericMathTemplateLibrary File Documentation

### 11.1 AABox.h File Reference

```
#include <gmtl/Vec.h>
```

Include dependency graph for AABox.h:

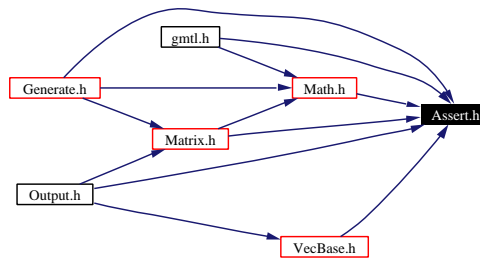


### Namespaces

- namespace [gmtl](#)
-

## 11.2 Assert.h File Reference

This graph shows which files directly or indirectly include this file:



### Defines

- #define `gmdlASSERT(val) ((void)0)`

### 11.2.1 Define Documentation

#### 11.2.1.1 #define gmdlASSERT(val) ((void)0)

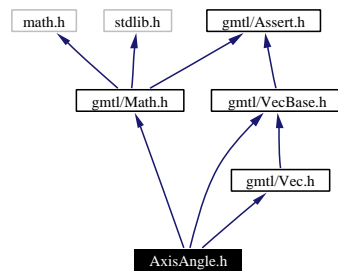
Definition at line 9 of file Assert.h.

Referenced by `gmdl::Tri< DATA_TYPE >::edge()`, `gmdl::findNearestPt()`, `gmdl::Math::isEqual()`, `gmdl::isEqual()`, `gmdl::isOnVolume()`, `gmdl::makeVolume()`, `gmdl::meanTangent()`, `gmdl::Matrix< DATA_TYPE, ROWS, COLS >::operator()`, `gmdl::VecBase< DATA_TYPE, 4 >::operator[]()`, `gmdl::Tri< DATA_TYPE >::operator[]()`, `gmdl::Quat< DATA_TYPE >::operator[]()`, `gmdl::Matrix< DATA_TYPE, ROWS, COLS >::operator[]()`, `gmdl::EulerAngle< DATA_TYPE, ROTATION_ORDER >::operator[]()`, `gmdl::Point< DATA_TYPE, 3 >::Point()`, `gmdl::VecBase< DATA_TYPE, SIZE >::set()`, `gmdl::Matrix< DATA_TYPE, ROWS, COLS >::set()`, `gmdl::set()`, `gmdl::setAxes()`, `gmdl::setDirCos()`, `gmdl::setRot()`, `gmdl::setScale()`, `gmdl::setTrans()`, `gmdl::squad()`, `gmdl::Vec< DATA_TYPE, 4 >::Vec()`, `gmdl::VecBase< DATA_TYPE, SIZE >::VecBase()`, and `gmdl::xform()`.

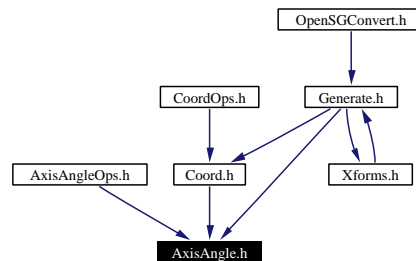
## 11.3 AxisAngle.h File Reference

```
#include <gmtl/Math.h>
#include <gmtl/VecBase.h>
#include <gmtl/Vec.h>
```

Include dependency graph for AxisAngle.h:



This graph shows which files directly or indirectly include this file:



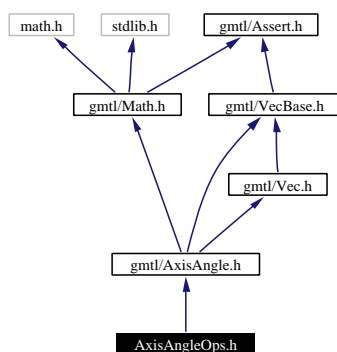
## Namespaces

- namespace `gmtl`

## 11.4 AxisAngleOps.h File Reference

```
#include "gmtl/AxisAngle.h"
```

Include dependency graph for AxisAngleOps.h:



### Namespaces

- namespace `gmtl`

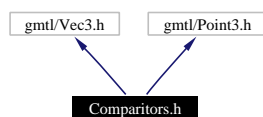


## 11.5 Comparitors.h File Reference

```
#include <gmtl/Vec3.h>
```

```
#include <gmtl/Point3.h>
```

Include dependency graph for Comparitors.h:



### Namespaces

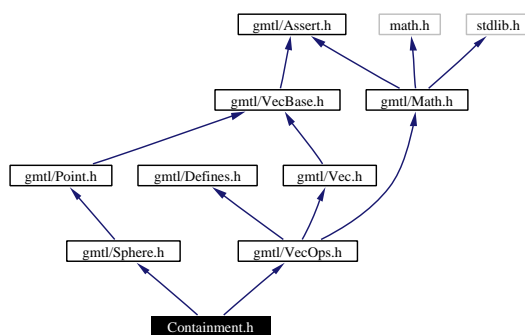
- namespace [gmtl](#)

## 11.6 Containment.h File Reference

```
#include <gmtl/Sphere.h>
```

```
#include <gmtl/VecOps.h>
```

Include dependency graph for Containment.h:



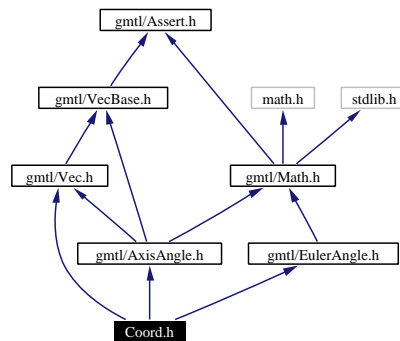
### Namespaces

- namespace `gmtl`

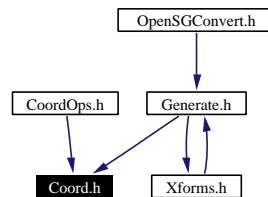
## 11.7 Coord.h File Reference

```
#include <gmtl/Vec.h>
#include <gmtl/AxisAngle.h>
#include <gmtl/EulerAngle.h>
```

Include dependency graph for Coord.h:



This graph shows which files directly or indirectly include this file:



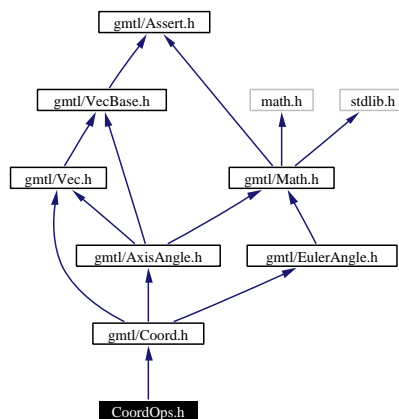
## Namespaces

- namespace [gmtl](#)

## 11.8 CoordOps.h File Reference

```
#include <gmtl/Coord.h>
```

Include dependency graph for CoordOps.h:

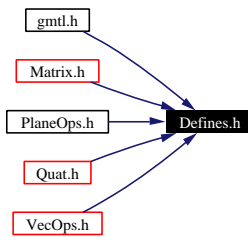


### Namespaces

- namespace `gmtl`

## 11.9 Defines.h File Reference

This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [gmtl](#)

### Defines

- `#define` [GMTL\\_NEAR](#)(x, y, eps) (gmtl::Math::abs((x)-(y))<(eps))

#### 11.9.1 Define Documentation

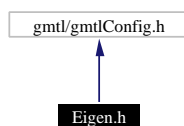
##### 11.9.1.1 `#define` GMTL\_NEAR(x, y, eps) (gmtl::Math::abs((x)-(y))<(eps))

Definition at line 77 of file Defines.h.

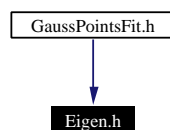
## 11.10 Eigen.h File Reference

```
#include <gmtl/gmtlConfig.h>
```

Include dependency graph for Eigen.h:



This graph shows which files directly or indirectly include this file:



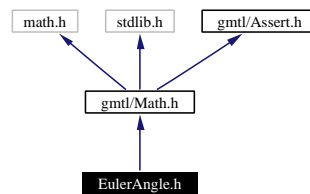
### Namespaces

- namespace [gmtl](#)

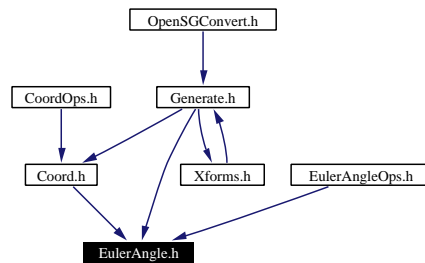
## 11.11 EulerAngle.h File Reference

```
#include <gmtl/Math.h>
```

Include dependency graph for EulerAngle.h:



This graph shows which files directly or indirectly include this file:



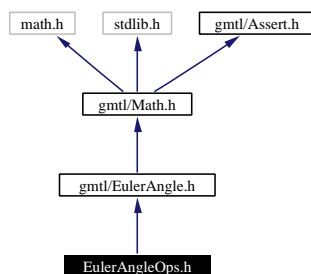
## Namespaces

- namespace [gmtl](#)

## 11.12 EulerAngleOps.h File Reference

```
#include "gmtl/EulerAngle.h"
```

Include dependency graph for EulerAngleOps.h:



### Namespaces

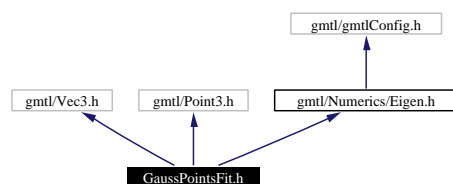
- namespace `gmtl`



## 11.13 GaussPointsFit.h File Reference

```
#include <gmtl/Vec3.h>  
#include <gmtl/Point3.h>  
#include <gmtl/Numerics/Eigen.h>
```

Include dependency graph for GaussPointsFit.h:

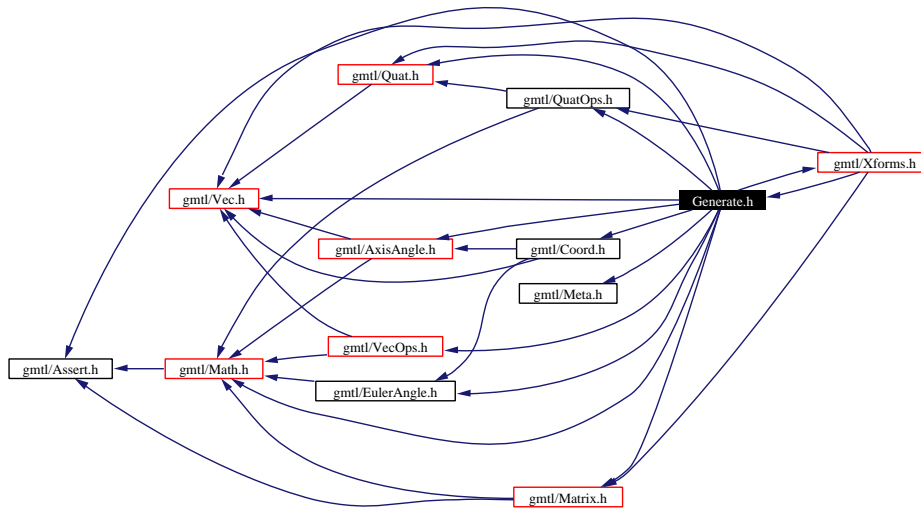


### Namespaces

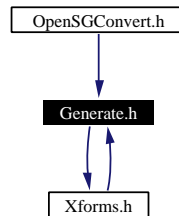
- namespace `gmtl`

## 11.14 Generate.h File Reference

```
#include <gmtl/Assert.h>
#include <gmtl/Vec.h>
#include <gmtl/VecOps.h>
#include <gmtl/Quat.h>
#include <gmtl/QuatOps.h>
#include <gmtl/Coord.h>
#include <gmtl/Matrix.h>
#include <gmtl/Meta.h>
#include <gmtl/Math.h>
#include <gmtl/Xforms.h>
#include <gmtl/EulerAngle.h>
#include <gmtl/AxisAngle.h>
Include dependency graph for Generate.h:
```



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [gmtl](#)

## 11.15 gmtl.doxygen File Reference

this file for documentation purposes only (for doxygen generation).

### 11.15.1 Detailed Description

this file for documentation purposes only (for doxygen generation).

Definition in file [gmtl.doxygen](#).

## 11.16 gmtl.h File Reference

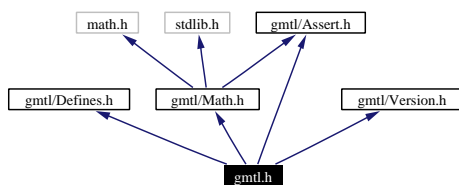
```
#include <gmtl/Defines.h>
```

```
#include <gmtl/Math.h>
```

```
#include <gmtl/Assert.h>
```

```
#include <gmtl/Version.h>
```

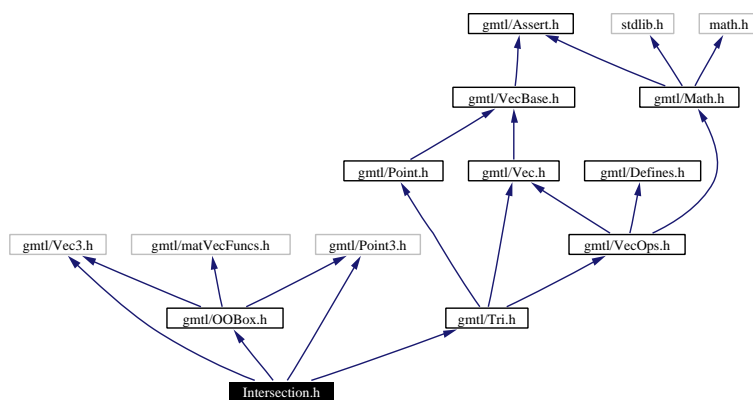
Include dependency graph for gmtl.h:



## 11.17 Intersection.h File Reference

```
#include <gmtl/Vec3.h>
#include <gmtl/Point3.h>
#include <gmtl/OOBox.h>
#include <gmtl/Tri.h>
```

Include dependency graph for Intersection.h:



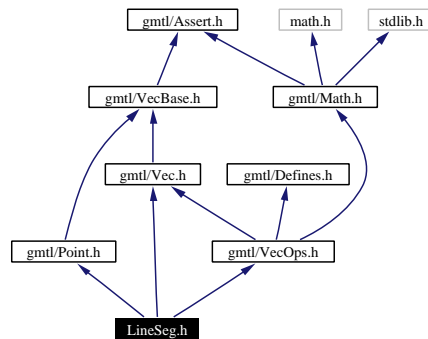
## Namespaces

- namespace `gmtl`

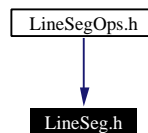
## 11.18 LineSeg.h File Reference

```
#include <gmtl/Point.h>
#include <gmtl/Vec.h>
#include <gmtl/VecOps.h>
```

Include dependency graph for LineSeg.h:



This graph shows which files directly or indirectly include this file:



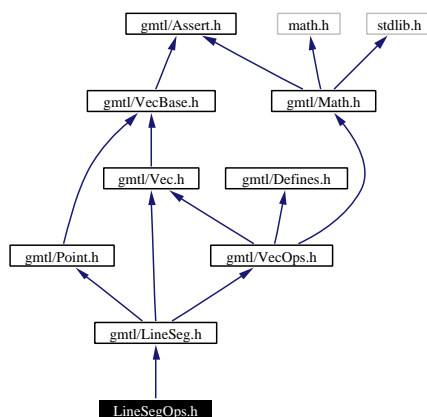
## Namespaces

- namespace `gmtl`

## 11.19 LineSegOps.h File Reference

```
#include <gmtl/LineSeg.h>
```

Include dependency graph for LineSegOps.h:



### Namespaces

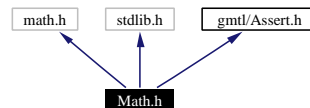
- namespace [gmtl](#)



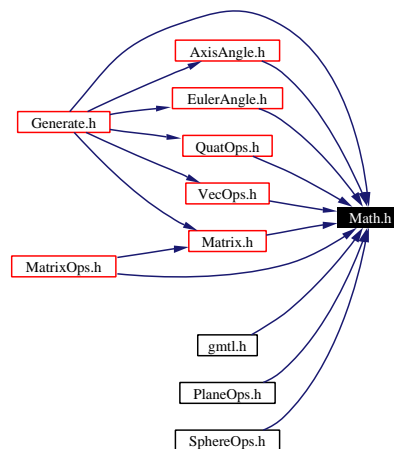
## 11.20 Math.h File Reference

```
#include <math.h>
#include <stdlib.h>
#include <gmtl/Assert.h>
```

Include dependency graph for Math.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [gmtl](#)
- namespace [gmtl::Math](#)

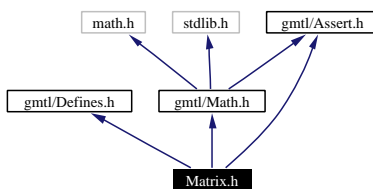
## 11.21 Matrix.h File Reference

```
#include <gmtl/Defines.h>
```

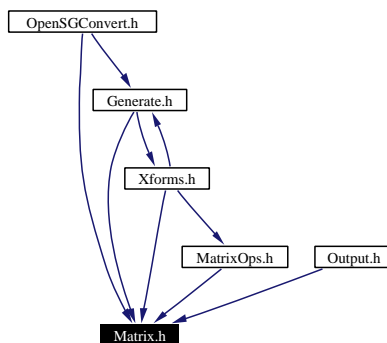
```
#include <gmtl/Math.h>
```

```
#include <gmtl/Assert.h>
```

Include dependency graph for Matrix.h:



This graph shows which files directly or indirectly include this file:



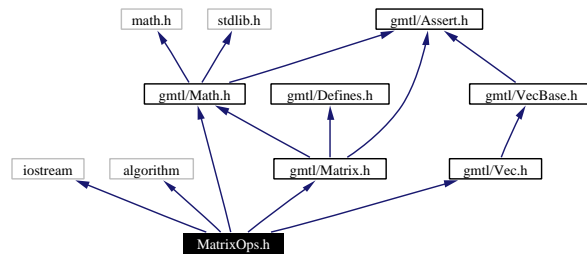
## Namespaces

- namespace [gmtl](#)

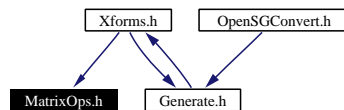
## 11.22 MatrixOps.h File Reference

```
#include <iostream>
#include <algorithm>
#include <gmtl/Matrix.h>
#include <gmtl/Math.h>
#include <gmtl/Vec.h>
```

Include dependency graph for MatrixOps.h:



This graph shows which files directly or indirectly include this file:

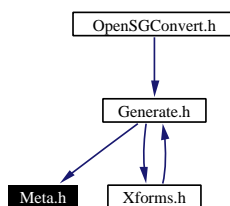


## Namespaces

- namespace [gmtl](#)

## 11.23 Meta.h File Reference

This graph shows which files directly or indirectly include this file:



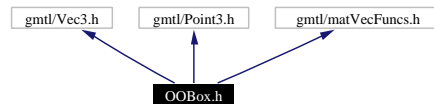
### Namespaces

- namespace [gmtl](#)

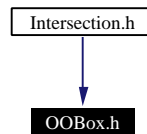
## 11.24 OOBox.h File Reference

```
#include <gmtl/Vec3.h>
#include <gmtl/Point3.h>
#include <gmtl/matVecFuncs.h>
```

Include dependency graph for OOBox.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

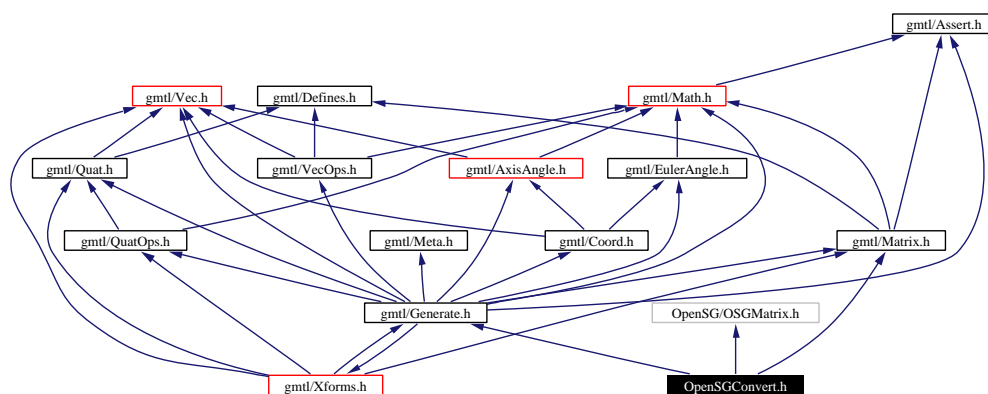
- namespace [gmtl](#)

## 11.25 OpenSGConvert.h File Reference

GMTL to OpenSG conversion functions.

```
#include <gmtl/Matrix.h>
#include <gmtl/Generate.h>
#include <OpenSG/OSGMatrix.h>
```

Include dependency graph for OpenSGConvert.h:



### Namespaces

- namespace [gmtl](#)

### 11.25.1 Detailed Description

GMTL to OpenSG conversion functions.

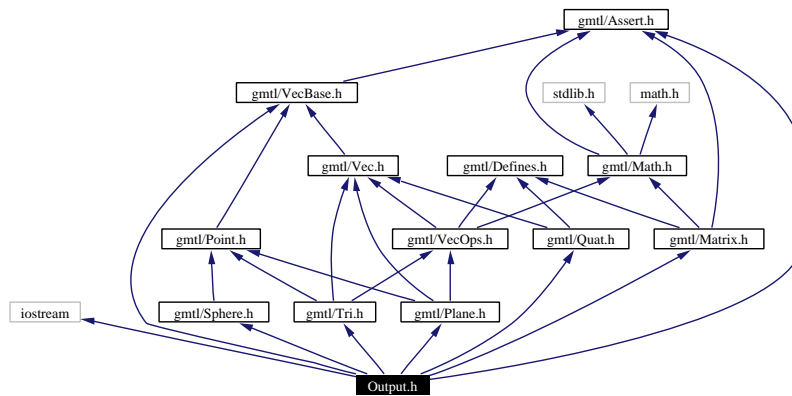
methods to convert between gmtl and opensg matrix classes

Definition in file [OpenSGConvert.h](#).

## 11.26 Output.h File Reference

```
#include <iostream>
#include <gmtl/Assert.h>
#include <gmtl/VecBase.h>
#include <gmtl/Matrix.h>
#include <gmtl/Quat.h>
#include <gmtl/Tri.h>
#include <gmtl/Plane.h>
#include <gmtl/Sphere.h>
```

Include dependency graph for Output.h:



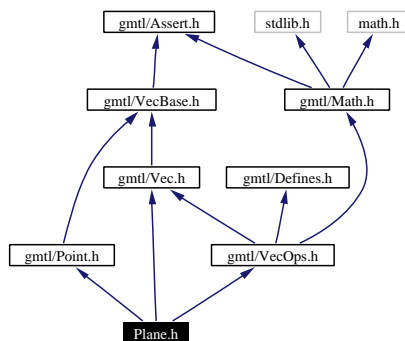
## Namespaces

- namespace [gmtl](#)

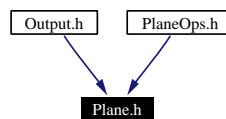
## 11.27 Plane.h File Reference

```
#include <gmtl/Vec.h>
#include <gmtl/Point.h>
#include <gmtl/VecOps.h>
```

Include dependency graph for Plane.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [gmtl](#)



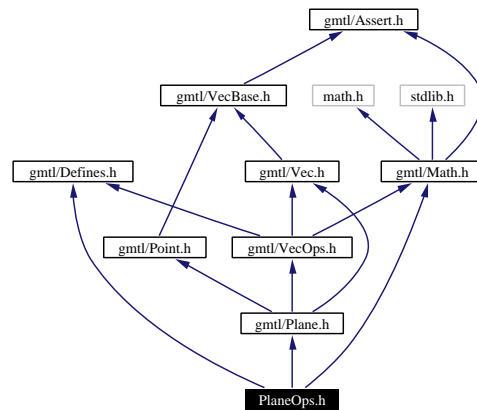
## 11.28 PlaneOps.h File Reference

```
#include <gmtl/Defines.h>
```

```
#include <gmtl/Plane.h>
```

```
#include <gmtl/Math.h>
```

Include dependency graph for PlaneOps.h:



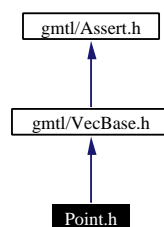
### Namespaces

- namespace `gmtl`

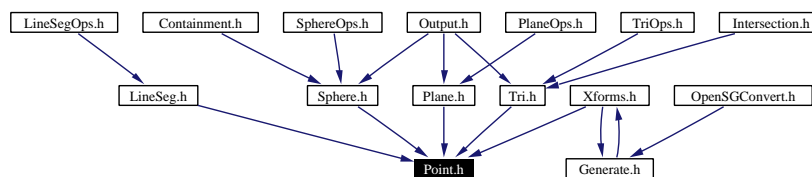
## 11.29 Point.h File Reference

```
#include <gmtl/VecBase.h>
```

Include dependency graph for Point.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

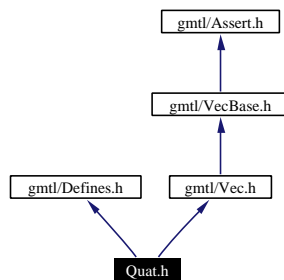
- namespace [gmtl](#)

## 11.30 Quat.h File Reference

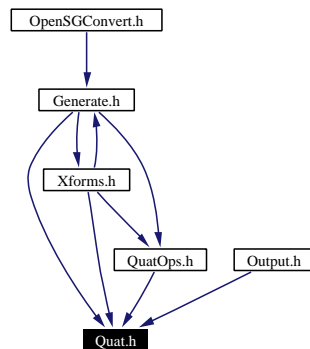
```
#include <gmtl/Defines.h>
```

```
#include <gmtl/Vec.h>
```

Include dependency graph for Quat.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

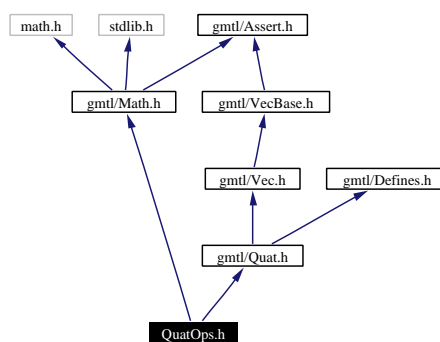
- namespace [gmtl](#)

## 11.31 QuatOps.h File Reference

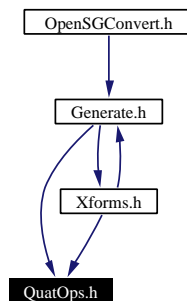
```
#include "gmtl/Math.h"
```

```
#include "gmtl/Quat.h"
```

Include dependency graph for QuatOps.h:



This graph shows which files directly or indirectly include this file:



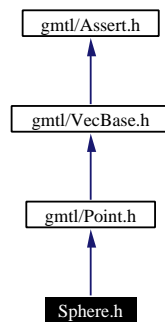
## Namespaces

- namespace [gmtl](#)

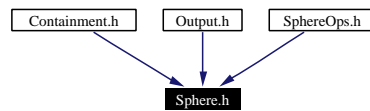
## 11.32 Sphere.h File Reference

```
#include <gmtl/Point.h>
```

Include dependency graph for Sphere.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

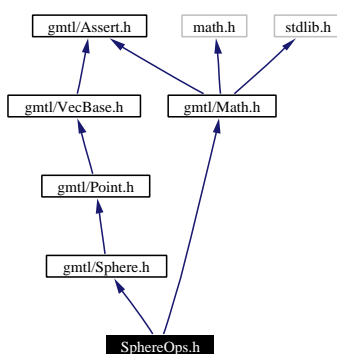
- namespace [gmtl](#)

## 11.33 SphereOps.h File Reference

```
#include <gmtl/Sphere.h>
```

```
#include <gmtl/Math.h>
```

Include dependency graph for SphereOps.h:



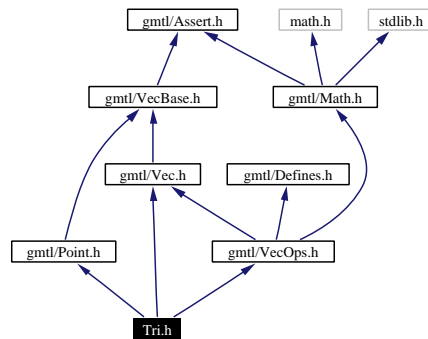
### Namespaces

- namespace `gmtl`

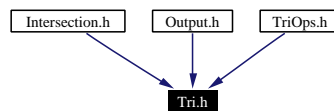
## 11.34 Tri.h File Reference

```
#include <gmtl/Point.h>
#include <gmtl/Vec.h>
#include <gmtl/VecOps.h>
```

Include dependency graph for Tri.h:



This graph shows which files directly or indirectly include this file:



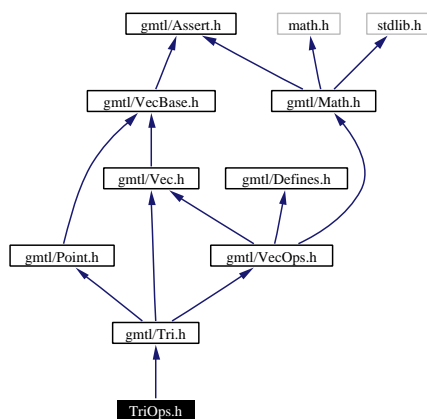
## Namespaces

- namespace `gmtl`

## 11.35 TriOps.h File Reference

```
#include <gmtl/Tri.h>
```

Include dependency graph for TriOps.h:



### Namespaces

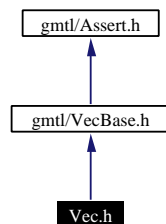
- namespace [gmtl](#)



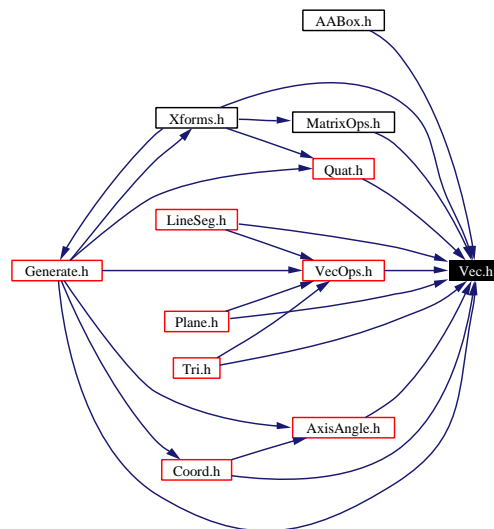
## 11.36 Vec.h File Reference

```
#include <gmtl/VecBase.h>
```

Include dependency graph for Vec.h:



This graph shows which files directly or indirectly include this file:



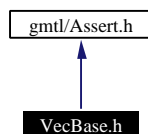
## Namespaces

- namespace [gmtl](#)

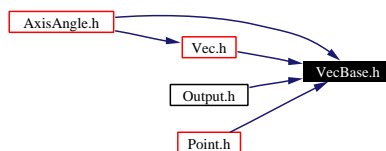
## 11.37 VecBase.h File Reference

```
#include "gmtl/Assert.h"
```

Include dependency graph for VecBase.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace `gmtl`

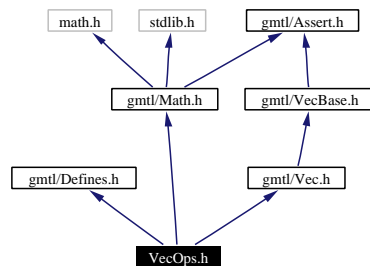
## 11.38 VecOps.h File Reference

```
#include "gmtl/Defines.h"
```

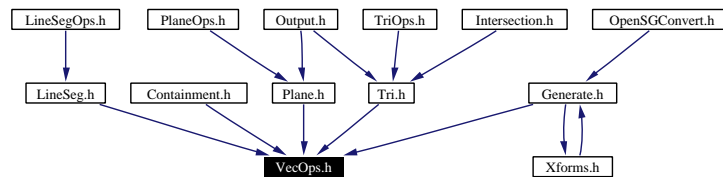
```
#include "gmtl/Math.h"
```

```
#include "gmtl/Vec.h"
```

Include dependency graph for VecOps.h:



This graph shows which files directly or indirectly include this file:

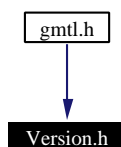


## Namespaces

- namespace [gmtl](#)

## 11.39 Version.h File Reference

This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace `gmtl`

### Defines

- `#define GMTL_VERSION_MAJOR 0`  
*This is the "human-readable" GMTL version string.*
- `#define GMTL_VERSION_MINOR 0`
- `#define GMTL_VERSION_PATCH 5`
- `#define GMTL_GLUE(a, b) a ## b`
- `#define GMTL_XGLUE(a, b) GMTL_GLUE(a,b)`
- `#define GMTL_STR(s) # s`
- `#define GMTL_XSTR(s) GMTL_STR(s)`
- `#define GMTL_DOT(a, b) a ## . ## b`
- `#define GMTL_XDOT(a, b) GMTL_DOT(a,b)`
- `#define GMTL_ZEROFILL(a) 0 ## a`
- `#define GMTL_XZEROFILL(a) GMTL_ZEROFILL(a)`
- `#define GMTL_VERSION_MAJOR_FILLED GMTL_XZEROFILL(GMTL_-XZEROFILL(GMTL_VERSION_MAJOR))`
- `#define GMTL_VERSION_MINOR_FILLED GMTL_XZEROFILL(GMTL_-XZEROFILL(GMTL_VERSION_MINOR))`
- `#define GMTL_VERSION_PATCH_FILLED GMTL_XZEROFILL(GMTL_-XZEROFILL(GMTL_VERSION_PATCH))`
- `#define GMTL_VERSION`  
*The is the preprocessor-friendly version string.*
- `#define GMTL_VERSION_STRING`

## 11.39.1 Define Documentation

### 11.39.1.1 `#define GMTL_DOT(a, b) a ## . ## b`

Definition at line 76 of file Version.h.

### 11.39.1.2 `#define GMTL_GLUE(a, b) a ## b`

Definition at line 68 of file Version.h.

### 11.39.1.3 `#define GMTL_STR(s) # s`

Definition at line 72 of file Version.h.

### 11.39.1.4 `#define GMTL_VERSION`

**Value:**

```
GMTL_XGLUE( \
    GMTL_XGLUE(GMTL_VERSION_MAJOR_FILLED, GMTL_VERSION_MINOR_FILLED), \
    GMTL_VERSION_PATCH_FILLED \
)
```

This is the preprocessor-friendly version string.

It is in the form of <major><minor><patch>. Each part has exactly 3 digits.

Definition at line 122 of file Version.h.

### 11.39.1.5 `#define GMTL_VERSION_MAJOR 0`

This is the "human-readable" GMTL version string.

It is of the form <major><minor><patch>. Each part has exactly 3 digits.

Definition at line 52 of file Version.h.

### 11.39.1.6 `#define GMTL_VERSION_MAJOR_FILLED GMTL_XZEROFILL(GMTL_XZEROFILL(GMTL_VERSION_MAJOR))`

Definition at line 85 of file Version.h.

**11.39.1.7 #define GMTL\_VERSION\_MINOR 0**

Definition at line 53 of file Version.h.

**11.39.1.8 #define GMTL\_VERSION\_MINOR\_FILLED GMTL\_XZEROFILL(GMTL\_XZEROFILL(GMTL\_VERSION\_MINOR))**

Definition at line 95 of file Version.h.

**11.39.1.9 #define GMTL\_VERSION\_PATCH 5**

Definition at line 54 of file Version.h.

**11.39.1.10 #define GMTL\_VERSION\_PATCH\_FILLED GMTL\_XZEROFILL(GMTL\_XZEROFILL(GMTL\_VERSION\_PATCH))**

Definition at line 105 of file Version.h.

**11.39.1.11 #define GMTL\_VERSION\_STRING**

**Value:**

```
GMTL_XDOT( \
    GMTL_XDOT(GMTL_VERSION_MAJOR, GMTL_VERSION_MINOR), \
    GMTL_VERSION_PATCH \
)
```

Definition at line 129 of file Version.h.

**11.39.1.12 #define GMTL\_XDOT(a, b) GMTL\_DOT(a,b)**

Definition at line 77 of file Version.h.

**11.39.1.13 #define GMTL\_XGLUE(a, b) GMTL\_GLUE(a,b)**

Definition at line 69 of file Version.h.

**11.39.1.14 #define GMTL\_XSTR(s) GMTL\_STR(s)**

Definition at line 73 of file Version.h.

**11.39.1.15   #define GMTL\_XZEROFILL(a) GMTL\_ZEROFILL(a)**

Definition at line 81 of file Version.h.

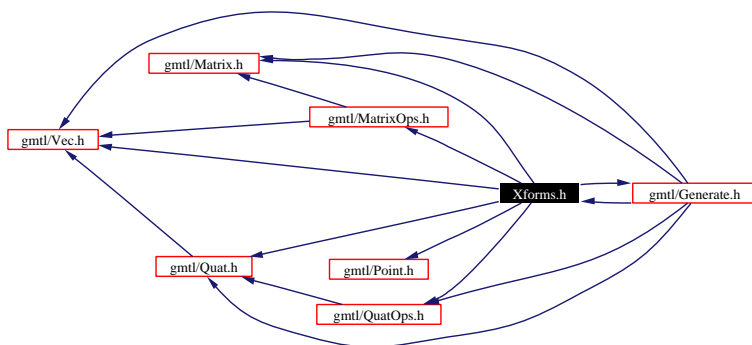
**11.39.1.16   #define GMTL\_ZEROFILL(a) 0 ## a**

Definition at line 80 of file Version.h.

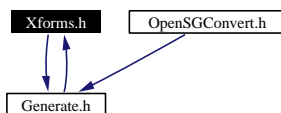
## 11.40 Xforms.h File Reference

```
#include <gmtl/Point.h>
#include <gmtl/Vec.h>
#include <gmtl/Matrix.h>
#include <gmtl/MatrixOps.h>
#include <gmtl/Quat.h>
#include <gmtl/QuatOps.h>
#include <gmtl/Generate.h>
```

Include dependency graph for Xforms.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [gmtl](#)



---

## Chapter 12

# GenericMathTemplateLibrary Page Documentation

### 12.1 Todo List

Member `gmtl::Coord::pos()` what about having a `pos`, and a `const_pos` naming convention?

what about having a `rot`, and a `const_rot` naming convention?

Class `gmtl::EulerAngle` bug: might not want to derive from `vec`, otherwise Euler-XYZ == EulerZXY works, when it shouldn't even compile...

Member `gmtl::Matrix::operator()(const unsigned row, const unsigned column)`  
metaprogramming

Member `gmtl::Matrix::set(const DATA_TYPE *data)` implement this!

Member `gmtl::Matrix::set(DATA_TYPE v00, DATA_TYPE v01, DATA_TYPE v02, DATA_TYPE v03, DATA_TYPE v10, DATA_TYPE v11, DATA_TYPE v12, DATA_TYPE v13, DATA_TYPE v20, DATA_TYPE v21, DATA_TYPE v22, DATA_TYPE v23)`  
needs mp!! currently no way for a 4x3, ....

Member `gmtl::Matrix::set(DATA_TYPE v00, DATA_TYPE v01, DATA_TYPE v02, DATA_TYPE v03, DATA_TYPE v10, DATA_TYPE v11, DATA_TYPE v12, DATA_TYPE v13, DATA_TYPE v20, DATA_TYPE v21, DATA_TYPE v22, DATA_TYPE v23, DATA_TYPE v30, DATA_TYPE v31, DATA_TYPE v32, DATA_TYPE v33)`  
needs mp!! currently no way for a 4x3, ....

Member `gmtl::Matrix::set(DATA_TYPE v00, DATA_TYPE v01, DATA_TYPE v02, DATA_TYPE v10, DATA_TYPE v11, DATA_TYPE v12, DATA_TYPE v20, DATA_TYPE v21, DATA_TYPE v22, DATA_TYPE v30, DATA_TYPE v31, DATA_TYPE v32, DATA_TYPE v33, DATA_TYPE v40, DATA_TYPE v41, DATA_TYPE v42, DATA_TYPE v43)`  
needs mp!!

---

Member **gmtl::Matrix::set**(DATA\_TYPE v00, DATA\_TYPE v01, DATA\_TYPE v02, DATA\_TYPE v10, DATA\_TYPE v11, DATA\_TYPE v12)  
needs mp!!

Member **gmtl::Matrix::set**(DATA\_TYPE v00, DATA\_TYPE v01, DATA\_TYPE v10, DATA\_TYPE v11, DATA\_TYPE v12, DATA\_TYPE v20, DATA\_TYPE v21, DATA\_TYPE v22)  
needs mp!!

Member **gmtl::Matrix::setTranspose**(const DATA\_TYPE \*data) mp

Member **gmtl::operator \***(const Quat< DATA\_TYPE > &q1, const Quat< DATA\_TYPE > &q2)  
metaprogramming on quat **operator \***()

Member **gmtl::xform**(Point< DATA\_TYPE, PNT\_SIZE > &result, const Matrix< DATA\_TYPE, ROWS, COLS > &mat)  
we need a PointOps.h operator \*=(scalar) function

Member **gmtl::makeVec**(const Quat< DATA\_TYPE > &quat) should this be  
called convert?

Member **gmtl::setPure**(Quat< DATA\_TYPE > &quat, const Vec< DATA\_TYPE, 3 > &vec)  
Write test case for setPure

Member **gmtl::set**(Matrix< DATA\_TYPE, ROWS, COLS > &mat, const Quat< DATA\_TYPE > &q)  
Implement using setRot

---

# Index

- ~Eigen
    - gmtl::Eigen, [261](#)
  - AABox
    - gmtl::AABox, [239](#), [240](#)
  - AABox.h, [367](#)
  - AABoxd
    - gmtl, [198](#)
  - AABoxf
    - gmtl, [198](#)
  - abs
    - Math, [22](#)
  - Abstract Data Types: Matrix, Vec,  
Quat, Coord, Sphere, Plane,  
[38](#)
  - aCos
    - Math, [22](#), [23](#)
  - add
    - Ops, [48](#)
  - AFFINE
    - gmtl::Matrix, [297](#)
  - aSin
    - Math, [23](#), [24](#)
  - Assert.h, [368](#)
    - gmtlASSERT, [368](#)
  - aTan
    - Math, [24](#), [25](#)
  - aTan2
    - Math, [25](#)
  - atan2
    - Math, [25](#)
  - axes
    - gmtl::OOBox, [309](#)
  - axis
    - gmtl::OOBox, [310](#)
  - AxisAngle
    - gmtl::AxisAngle, [247](#), [248](#)
  - AxisAngle.h, [369](#)
  - AXISANGLE\_IDENTITYD
    - gmtl, [204](#)
  - AXISANGLE\_IDENTITYF
    - gmtl, [204](#)
  - AxisAngled
    - gmtl, [198](#)
  - AxisAnglef
    - gmtl, [198](#)
  - AxisAngleOps.h, [370](#)
  - BaseType
    - gmtl::Point, [323](#)
    - gmtl::Vec, [348](#)
  - C Math Abstraction: sin, cos, tan, Min,  
Max, PI, [20](#)
  - ceil
    - Math, [25](#), [26](#)
  - center
    - gmtl::OOBox, [310](#)
    - Ops, [49](#)
  - Cols
    - gmtl::Matrix, [297](#)
  - Compare
    - isEqual, [102–107](#)
    - isEquiv, [108](#)
    - operator!=, [108–112](#)
    - operator==, [112–116](#)
  - CompareIndexPointProjections
    - gmtl::CompareIndexPoint-  
Projections, [252](#)
  - Comparison: isEqual(...), isEquiv(...),  
==,  
Comparison: isEqual(...), is-  
Equiv(...), ==,  
=, [99](#)
-

- Comparitors.h, 371
- conj
  - Ops, 49
- Containment.h, 372
- Coord
  - gmtl::Coord, 256, 257
- Coord.h, 373
- CoordOps.h, 374
- CoordVec3AxisAngled
  - gmtl, 198
- CoordVec3AxisAnglef
  - gmtl, 198
- CoordVec3EulerAngleXYZd
  - gmtl, 198
- CoordVec3EulerAngleXYZf
  - gmtl, 199
- CoordVec3EulerAngleZXYd
  - gmtl, 199
- CoordVec3EulerAngleZXYf
  - gmtl, 199
- CoordVec3EulerAngleZYPd
  - gmtl, 199
- CoordVec3EulerAngleZYPf
  - gmtl, 199
- CoordVec4AxisAngled
  - gmtl, 199
- CoordVec4AxisAnglef
  - gmtl, 199
- CoordVec4EulerAngleXYZd
  - gmtl, 199
- CoordVec4EulerAngleXYZf
  - gmtl, 200
- CoordVec4EulerAngleZXYd
  - gmtl, 200
- CoordVec4EulerAngleZXYf
  - gmtl, 200
- CoordVec4EulerAngleZYPd
  - gmtl, 200
- CoordVec4EulerAngleZYPf
  - gmtl, 200
- cos
  - Math, 26, 27
- cross
  - Ops, 50, 51
- DataType
  - gmtl::AABBox, 239
  - gmtl::Coord, 255
  - gmtl::Matrix, 297
  - gmtl::Point, 323
  - gmtl::Quat, 329
  - gmtl::Sphere, 336
  - gmtl::Vec, 349
  - gmtl::VecBase, 354
- DecreasingSort
  - gmtl::Eigen, 262
- DecrSortEigenStuff
  - gmtl::Eigen, 262
- DecrSortEigenStuff2
  - gmtl::Eigen, 263
- DecrSortEigenStuff3
  - gmtl::Eigen, 263
- DecrSortEigenStuff4
  - gmtl::Eigen, 264
- DecrSortEigenStuffN
  - gmtl::Eigen, 264
- Defines
  - GMTL\_EPSILON, 19
  - GMTL\_MAT\_EQUAL\_-EPSILON, 19
  - GMTL\_VEC\_EQUAL\_-EPSILON, 19
  - NEG\_SIDE, 18
  - ON\_PLANE, 18
  - PlaneSide, 18
  - POS\_SIDE, 18
  - VectorIndex, 18
  - Welt, 19
  - Xelt, 19
  - Yelt, 19
  - Zelt, 19
- Defines.h, 375
  - GMTL\_NEAR, 375
- deg2Rad
  - Math, 27
- distance
  - gmtl, 204
  - Ops, 51
- div
  - Ops, 52
- dot
  - Ops, 53, 54

- dynObbFind0
  - gmtl, 205
- dynObbFind1
  - gmtl, 205
- edge
  - gmtl::Tri, 343
- Eigen
  - gmtl::Eigen, 261
- Eigen.h, 376
- EigenStuff
  - gmtl::Eigen, 264
- EigenStuff2
  - gmtl::Eigen, 265
- EigenStuff3
  - gmtl::Eigen, 265
- EigenStuff4
  - gmtl::Eigen, 265
- EigenStuffN
  - gmtl::Eigen, 266
- EulerAngle
  - gmtl::EulerAngle, 283, 284
- EulerAngle.h, 377
- EULERANGLE\_IDENTITY\_XYZD
  - gmtl, 206
- EULERANGLE\_IDENTITY\_XYZF
  - gmtl, 206
- EULERANGLE\_IDENTITY\_ZXYD
  - gmtl, 207
- EULERANGLE\_IDENTITY\_ZXYF
  - gmtl, 207
- EULERANGLE\_IDENTITY\_ZYXD
  - gmtl, 207
- EULERANGLE\_IDENTITY\_ZYXF
  - gmtl, 207
- EulerAngleOps.h, 378
- EulerAngleXYZd
  - gmtl, 200
- EulerAngleXYZf
  - gmtl, 200
- EulerAngleZXYd
  - gmtl, 200
- EulerAngleZXYf
  - gmtl, 201
- EulerAngleZYXd
  - gmtl, 201
- EulerAngleZYXf
  - gmtl, 201
- exp
  - Math, 27, 28
  - Ops, 54
- extendVolume
  - gmtl, 207, 208
- factorial
  - Math, 28
- findNearestPt
  - gmtl, 209
  - Ops, 55
- floor
  - Math, 28, 29
- FULL
  - gmtl::Matrix, 297
- GaussPointsFit
  - gmtl, 209, 211
- GaussPointsFit.h, 379
- Generate
  - make, 124
  - makeAxes, 125
  - makeConj, 125
  - makeDirCos, 126
  - makeInverse, 127
  - makeInvert, 127
  - makeNormal, 127, 128
  - makePure, 129
  - makeRot, 129
  - makeScale, 130
  - makeTrans, 131
  - makeTranspose, 131
  - makeVec, 132
  - makeXRot, 132
  - makeYRot, 133
  - makeZRot, 134
  - set, 135–137, 139–141, 143
  - setAxes, 145
  - setDirCos, 145
  - setPure, 146
  - setRot, 147–152
  - setScale, 154
  - setTrans, 154, 156
- Generate.h, 380

- Generators: make( ... ), set( ... ), 118
- get
  - gmtl::Quat, 330
- getAngle
  - gmtl::AxisAngle, 249
- getAxis
  - gmtl::AxisAngle, 249
- getCenter
  - gmtl::Sphere, 338
- getData
  - gmtl::EulerAngle, 284
  - gmtl::Matrix, 299
  - gmtl::Quat, 330
  - gmtl::VecBase, 356, 357
- getDir
  - gmtl::LineSeg, 290
- GetEigenvalue
  - gmtl::Eigen, 266
- GetEigenvector
  - gmtl::Eigen, 267
- getLength
  - gmtl::LineSeg, 290
- getMax
  - gmtl::AABBox, 240
- getMin
  - gmtl::AABBox, 241
- getNormal
  - gmtl::Plane, 319
- getOffset
  - gmtl::Plane, 319
- getOrigin
  - gmtl::LineSeg, 291
- getPos
  - gmtl::Coord, 257
- getRadius
  - gmtl::Sphere, 338
- getRot
  - gmtl::Coord, 257
- getVerts
  - gmtl::OBox, 311
- Global Flags: Xelt, XYZ, etc..., 17
- gmtl, 171
  - AABBoxd, 198
  - AABBoxf, 198
  - AXISANGLE.IDENTITYD, 204
  - AXISANGLE.IDENTITYF, 204
  - AxisAngled, 198
  - AxisAnglef, 198
  - CoordVec3AxisAngled, 198
  - CoordVec3AxisAnglef, 198
  - CoordVec3EulerAngleXYZd, 198
  - CoordVec3EulerAngleXYZf, 199
  - CoordVec3EulerAngleZXYd, 199
  - CoordVec3EulerAngleZXYf, 199
  - CoordVec3EulerAngleZXYd, 199
  - CoordVec3EulerAngleZYXf, 199
  - CoordVec4AxisAngled, 199
  - CoordVec4AxisAnglef, 199
  - CoordVec4EulerAngleXYZd, 199
  - CoordVec4EulerAngleXYZf, 200
  - CoordVec4EulerAngleZXYd, 200
  - CoordVec4EulerAngleZXYf, 200
  - CoordVec4EulerAngleZYXd, 200
  - CoordVec4EulerAngleZYXf, 200
  - distance, 204
  - dynObbFind0, 205
  - dynObbFind1, 205
  - EULERANGLE.IDENTITY\_-XYZD, 206
  - EULERANGLE.IDENTITY\_-XYZF, 206
  - EULERANGLE.IDENTITY\_-ZXYD, 207
  - EULERANGLE.IDENTITY\_-ZXYF, 207
  - EULERANGLE.IDENTITY\_-ZYXD, 207
  - EULERANGLE.IDENTITY\_-ZYXF, 207
  - EulerAngleXYZd, 200
  - EulerAngleXYZf, 200
  - EulerAngleZXYd, 200
  - EulerAngleZXYf, 201
  - EulerAngleZYXd, 201
  - EulerAngleZYXf, 201
  - extendVolume, 207, 208

- findNearestPt, 209
- GaussPointsFit, 209, 211
- isEqual, 212
- isInVolume, 213
- isOnVolume, 214, 215
- LineSegd, 201
- LineSegf, 201
- makeVolume, 215
- MAT\_IDENTITY22D, 231
- MAT\_IDENTITY22F, 231
- MAT\_IDENTITY23D, 231
- MAT\_IDENTITY23F, 231
- MAT\_IDENTITY33D, 231
- MAT\_IDENTITY33F, 232
- MAT\_IDENTITY34D, 232
- MAT\_IDENTITY34F, 232
- MAT\_IDENTITY44D, 232
- MAT\_IDENTITY44F, 232
- Matrix22d, 201
- Matrix22f, 201
- Matrix23d, 201
- Matrix23f, 201
- Matrix33d, 201
- Matrix33f, 202
- Matrix34d, 202
- Matrix34f, 202
- Matrix44d, 202
- Matrix44f, 202
- operator!=, 216
- operator==, 217
- Planed, 202
- Planef, 202
- Point3d, 202
- Point3f, 202
- Point4d, 203
- Point4f, 203
- QUAT\_ADD\_IDENTITYD, 217
- QUAT\_ADD\_IDENTITYF, 217
- QUAT\_IDENTITYD, 217
- QUAT\_IDENTITYF, 218
- QUAT\_MULT\_IDENTITYD, 218
- QUAT\_MULT\_IDENTITYF, 218
- Quatd, 203
- Quatf, 203
- set, 218
- Sphered, 203
- Spheref, 203
- TestIntersect, 219
- TestIntersectOBB, 222
- Vec2d, 203
- Vec2f, 203
- Vec3d, 203
- Vec3f, 203
- Vec4d, 204
- Vec4f, 204
- version, 232
- gmtl.doxxygen, 382
- gmtl.h, 383
- gmtl::AABBox, 237
  - AABBox, 239, 240
  - DataType, 239
  - getMax, 240
  - getMin, 241
  - isEmpty, 241
  - mEmpty, 243
  - mMax, 243
  - mMin, 243
  - setEmpty, 242
  - setMax, 242
  - setMin, 242
- gmtl::AxisAngle
  - Size, 247
- gmtl::AxisAngle, 245
  - AxisAngle, 247, 248
  - getAngle, 249
  - getAxis, 249
  - set, 249, 250
  - setAngle, 250
  - setAxis, 250
- gmtl::CompareIndexPointProjections, 252
  - CompareIndexPointProjections, 252
  - operator(), 252
  - points, 253
  - sortDir, 253
- gmtl::Coord, 254
  - Coord, 256, 257
  - DataType, 255
  - getPos, 257
  - getRot, 257
  - mPos, 258

- mRot, [259](#)
- pos, [258](#)
- PosDataType, [255](#)
- PosSize, [256](#)
- rot, [258](#)
- RotDataType, [256](#)
- RotSize, [256](#)
- gmtl::Eigen, [260](#)
  - ~Eigen, [261](#)
  - DecreasingSort, [262](#)
  - DecrSortEigenStuff, [262](#)
  - DecrSortEigenStuff2, [263](#)
  - DecrSortEigenStuff3, [263](#)
  - DecrSortEigenStuff4, [264](#)
  - DecrSortEigenStuffN, [264](#)
  - Eigen, [261](#)
  - EigenStuff, [264](#)
  - EigenStuff2, [265](#)
  - EigenStuff3, [265](#)
  - EigenStuff4, [265](#)
  - EigenStuffN, [266](#)
  - GetEigenvalue, [266](#)
  - GetEigenvector, [267](#)
  - IncreasingSort, [267](#)
  - IncrSortEigenStuff, [268](#)
  - IncrSortEigenStuff2, [269](#)
  - IncrSortEigenStuff3, [269](#)
  - IncrSortEigenStuff4, [269](#)
  - IncrSortEigenStuffN, [270](#)
  - m\_aafMat, [279](#)
  - m\_afDiag, [279](#)
  - m\_afSubd, [279](#)
  - m\_iSize, [279](#)
  - Matrix, [270](#)
  - QLAlgorithm, [270](#)
  - SetMatrix, [272](#)
  - Tridiagonal2, [272](#)
  - Tridiagonal3, [273](#)
  - Tridiagonal4, [274](#)
  - TridiagonalN, [277](#)
- gmtl::EulerAngle
  - Order, [283](#)
  - Size, [283](#)
- gmtl::EulerAngle, [281](#)
  - EulerAngle, [283](#), [284](#)
  - getData, [284](#)
  - operator[], [285](#)
  - set, [286](#)
- gmtl::LineSeg, [287](#)
  - getDir, [290](#)
  - getLength, [290](#)
  - getOrigin, [291](#)
  - LineSeg, [288–290](#)
  - mDir, [292](#)
  - mOrigin, [292](#)
  - setDir, [291](#)
  - setOrigin, [292](#)
- gmtl::Math, [233](#)
- gmtl::Matrix, [294](#)
  - AFFINE, [297](#)
  - Cols, [297](#)
  - DataType, [297](#)
  - FULL, [297](#)
  - getData, [299](#)
  - IDENTITY, [297](#)
  - isError, [299](#)
  - Matrix, [298](#)
  - mData, [307](#)
  - mState, [307](#)
  - operator(), [299](#), [300](#)
  - operator[], [300](#), [301](#)
  - ORTHOGONAL, [297](#)
  - ORTHONORMAL, [297](#)
  - Rows, [297](#)
  - set, [301–305](#)
  - setError, [305](#)
  - setTranspose, [306](#)
  - XFORM\_ERROR, [297](#)
  - XformState, [297](#)
- gmtl::OOBox, [308](#)
  - axes, [309](#)
  - axis, [310](#)
  - center, [310](#)
  - getVerts, [311](#)
  - halfLen, [311](#)
  - halfLens, [312](#)
  - ident, [312](#)
  - mAxis, [314](#)
  - mCenter, [314](#)
  - mergeWith, [313](#)
  - mHalfLen, [314](#)
  - OOBox, [308](#), [309](#)



- operator=, 313
- operator==, 313
- gmtl::Plane, 315
  - getNormal, 319
  - getOffset, 319
  - mNorm, 321
  - mOffset, 321
  - Plane, 317–319
  - setNormal, 320
  - setOffset, 320
- gmtl::Point, 322
  - BaseType, 323
  - DataType, 323
  - Point, 324, 325
  - Size, 324
- gmtl::Quat, 327
  - DataType, 329
  - get, 330
  - getData, 330
  - mData, 332
  - operator[], 330, 331
  - Quat, 329
  - set, 332
- gmtl::RotationOrderBase
  - IS\_ROTORDER, 333
- gmtl::RotationOrderBase, 333
- gmtl::Sphere, 335
  - DataType, 336
  - getCenter, 338
  - getRadius, 338
  - mCenter, 339
  - mRadius, 339
  - setCenter, 338
  - setRadius, 339
  - Sphere, 337
- gmtl::Tri, 341
  - edge, 343
  - operator[], 344
  - Tri, 342, 343
- gmtl::Type2Type, 346
  - OriginalType, 346
- gmtl::Vec, 347
  - BaseType, 348
  - DataType, 349
  - Size, 349
  - Vec, 349–351
- gmtl::VecBase
  - Size, 354
- gmtl::VecBase, 352
  - DataType, 354
  - getData, 356, 357
  - mData, 360
  - operator[], 357
  - set, 358, 359
  - VecBase, 355, 356
- gmtl::XYZ, 361
  - ID, 361
- gmtl::ZXY, 363
  - ID, 363
- gmtl::ZYX, 365
  - ID, 365
- GMTL\_DOT
  - Version.h, 407
- GMTL\_EPSILON
  - Defines, 19
- GMTL\_GLUE
  - Version.h, 407
- GMTL\_MAT\_EQUAL\_EPSILON
  - Defines, 19
- GMTL\_NEAR
  - Defines.h, 375
- GMTL\_STR
  - Version.h, 407
- GMTL\_VEC\_EQUAL\_EPSILON
  - Defines, 19
- GMTL\_VERSION
  - Version.h, 407
- GMTL\_VERSION\_MAJOR
  - Version.h, 407
- GMTL\_VERSION\_MAJOR\_FILLED
  - Version.h, 407
- GMTL\_VERSION\_MINOR
  - Version.h, 407
- GMTL\_VERSION\_MINOR\_FILLED
  - Version.h, 408
- GMTL\_VERSION\_PATCH
  - Version.h, 408
- GMTL\_VERSION\_PATCH\_FILLED
  - Version.h, 408
- GMTL\_VERSION\_STRING
  - Version.h, 408
- GMTL\_XDOT

- Version.h, 408
- GMTL\_XGLUE
  - Version.h, 408
- GMTL\_XSTR
  - Version.h, 408
- GMTL\_XZEROFILL
  - Version.h, 408
- GMTL\_ZEROFILL
  - Version.h, 409
- gmtlASSERT
  - Assert.h, 368
- halfLen
  - gmtl::OOBox, 311
- halfLens
  - gmtl::OOBox, 312
- HelperMeta
  - ignore\_unused\_variable\_warning, 169
- ID
  - gmtl::XYZ, 361
  - gmtl::ZXY, 363
  - gmtl::ZYX, 365
- ident
  - gmtl::OOBox, 312
- IDENTITY
  - gmtl::Matrix, 297
- identity
  - Ops, 56
- ignore\_unused\_variable\_warning
  - HelperMeta, 169
- IncreasingSort
  - gmtl::Eigen, 267
- IncrSortEigenStuff
  - gmtl::Eigen, 268
- IncrSortEigenStuff2
  - gmtl::Eigen, 269
- IncrSortEigenStuff3
  - gmtl::Eigen, 269
- IncrSortEigenStuff4
  - gmtl::Eigen, 269
- IncrSortEigenStuffN
  - gmtl::Eigen, 270
- Interp
  - lerp, 159, 160
  - slerp, 161
- Interpolation: lerp(...), slerp(...), 158
- Intersection.h, 384
- invert
  - Ops, 56–58
- invertFull
  - Ops, 58
- IS\_ROTORDER
  - gmtl::RotationOrderBase, 333
- isEmpty
  - gmtl::AABBox, 241
- isEqual
  - Compare, 102–107
  - gmtl, 212
  - Math, 29
- isEquiv
  - Compare, 108
- isError
  - gmtl::Matrix, 299
- isInVolume
  - gmtl, 213
- isNormalized
  - Ops, 61
- isOnVolume
  - gmtl, 214, 215
- length
  - Ops, 62
- lengthSquared
  - Ops, 63
- lerp
  - Interp, 159, 160
- LineSeg
  - gmtl::LineSeg, 288–290
- LineSeg.h, 385
- LineSegd
  - gmtl, 201
- LineSegf
  - gmtl, 201
- LineSegOps.h, 386
- log
  - Math, 29, 30
  - Ops, 64
- m\_aafMat
  - gmtl::Eigen, 279

- m\_afDiag
  - gmtl::Eigen, [279](#)
- m\_afSubd
  - gmtl::Eigen, [279](#)
- m\_iSize
  - gmtl::Eigen, [279](#)
- make
  - Generate, [124](#)
- makeAxes
  - Generate, [125](#)
- makeConj
  - Generate, [125](#)
- makeDirCos
  - Generate, [126](#)
- makeInverse
  - Generate, [127](#)
- makeInvert
  - Generate, [127](#)
- makeNormal
  - Generate, [127](#), [128](#)
- makePure
  - Generate, [129](#)
- makeRot
  - Generate, [129](#)
- makeScale
  - Generate, [130](#)
- makeTrans
  - Generate, [131](#)
- makeTranspose
  - Generate, [131](#)
- makeVec
  - Generate, [132](#)
- makeVolume
  - gmtl, [215](#)
- makeXRot
  - Generate, [132](#)
- makeYRot
  - Generate, [133](#)
- makeZRot
  - Generate, [134](#)
- MAT\_IDENTITY22D
  - gmtl, [231](#)
- MAT\_IDENTITY22F
  - gmtl, [231](#)
- MAT\_IDENTITY23D
  - gmtl, [231](#)
- MAT\_IDENTITY23F
  - gmtl, [231](#)
- MAT\_IDENTITY33D
  - gmtl, [231](#)
- MAT\_IDENTITY33F
  - gmtl, [232](#)
- MAT\_IDENTITY34D
  - gmtl, [232](#)
- MAT\_IDENTITY34F
  - gmtl, [232](#)
- MAT\_IDENTITY44D
  - gmtl, [232](#)
- MAT\_IDENTITY44F
  - gmtl, [232](#)
- Math
  - abs, [22](#)
  - aCos, [22](#), [23](#)
  - aSin, [23](#), [24](#)
  - aTan, [24](#), [25](#)
  - aTan2, [25](#)
  - atan2, [25](#)
  - ceil, [25](#), [26](#)
  - cos, [26](#), [27](#)
  - deg2Rad, [27](#)
  - exp, [27](#), [28](#)
  - factorial, [28](#)
  - floor, [28](#), [29](#)
  - isEqual, [29](#)
  - log, [29](#), [30](#)
  - Max, [30](#)
  - Min, [31](#)
  - PI, [36](#)
  - PI\_OVER\_2, [37](#)
  - PI\_OVER\_4, [37](#)
  - pow, [32](#)
  - rad2Deg, [32](#)
  - rangeRandom, [32](#)
  - round, [33](#)
  - sign, [33](#)
  - sin, [33](#), [34](#)
  - sqr, [34](#)
  - sqrt, [34](#)
  - tan, [35](#)
  - trunc, [35](#)
  - unitRandom, [36](#)
  - zeroClamp, [36](#)

- Math.h, [387](#)
- Mathematical Operations: `add(...)`,  
`sub(...)`, `mul(...)`, `div(...)`, `invert(...)`, `dot(...)`, `cross(...)`, [40](#)
- Matrix
  - `gmtl::Eigen`, [270](#)
  - `gmtl::Matrix`, [298](#)
- Matrix.h, [388](#)
- Matrix22d
  - `gmtl`, [201](#)
- Matrix22f
  - `gmtl`, [201](#)
- Matrix23d
  - `gmtl`, [201](#)
- Matrix23f
  - `gmtl`, [201](#)
- Matrix33d
  - `gmtl`, [201](#)
- Matrix33f
  - `gmtl`, [202](#)
- Matrix34d
  - `gmtl`, [202](#)
- Matrix34f
  - `gmtl`, [202](#)
- Matrix44d
  - `gmtl`, [202](#)
- Matrix44f
  - `gmtl`, [202](#)
- MatrixOps.h, [389](#)
- Max
  - Math, [30](#)
- mAxis
  - `gmtl::OOBox`, [314](#)
- mCenter
  - `gmtl::OOBox`, [314](#)
  - `gmtl::Sphere`, [339](#)
- mData
  - `gmtl::Matrix`, [307](#)
  - `gmtl::Quat`, [332](#)
  - `gmtl::VecBase`, [360](#)
- mDir
  - `gmtl::LineSeg`, [292](#)
- meanTangent
  - Ops, [65](#)
- mEmpty
  - `gmtl::AABBox`, [243](#)
- mergeWith
  - `gmtl::OOBox`, [313](#)
- Meta.h, [390](#)
- mHalfLen
  - `gmtl::OOBox`, [314](#)
- Min
  - Math, [31](#)
- mMax
  - `gmtl::AABBox`, [243](#)
- mMin
  - `gmtl::AABBox`, [243](#)
- mNorm
  - `gmtl::Plane`, [321](#)
- mOffset
  - `gmtl::Plane`, [321](#)
- mOrigin
  - `gmtl::LineSeg`, [292](#)
- mPos
  - `gmtl::Coord`, [258](#)
- mRadius
  - `gmtl::Sphere`, [339](#)
- mRot
  - `gmtl::Coord`, [259](#)
- mState
  - `gmtl::Matrix`, [307](#)
- mult
  - Ops, [65–67](#)
- NEG\_SIDE
  - Defines, [18](#)
- negate
  - Ops, [68](#)
- normal
  - Ops, [69](#)
- normalize
  - Ops, [69, 70](#)
- ON\_PLANE
  - Defines, [18](#)
- OOBox
  - `gmtl::OOBox`, [308, 309](#)
- OOBox.h, [391](#)
- OpenSGConvert.h, [392](#)
- operator `*`
  - Ops, [71–73](#)
  - Transforms, [90–93](#)

- operator \*=
  - Ops, [74](#), [75](#)
- operator !=
  - Compare, [108–112](#)
  - gmtl, [216](#)
- operator()
  - gmtl::CompareIndexPoint-  
Projections, [252](#)
  - gmtl::Matrix, [299](#), [300](#)
- operator+
  - Ops, [76](#)
- operator +=
  - Ops, [77](#), [78](#)
- operator-
  - Ops, [78](#), [79](#)
- operator-=
  - Ops, [80](#)
- operator/
  - Ops, [81](#)
- operator/=
  - Ops, [82](#)
- operator <<
  - Output, [164–166](#)
- operator=
  - gmtl::OBox, [313](#)
- operator==
  - Compare, [112–116](#)
  - gmtl, [217](#)
  - gmtl::OBox, [313](#)
- operator[]
  - gmtl::EulerAngle, [285](#)
  - gmtl::Matrix, [300](#), [301](#)
  - gmtl::Quat, [330](#), [331](#)
  - gmtl::Tri, [344](#)
  - gmtl::VecBase, [357](#)
- Ops
  - add, [48](#)
  - center, [49](#)
  - conj, [49](#)
  - cross, [50](#), [51](#)
  - distance, [51](#)
  - div, [52](#)
  - dot, [53](#), [54](#)
  - exp, [54](#)
  - findNearestPt, [55](#)
  - identity, [56](#)
  - invert, [56–58](#)
  - invertFull, [58](#)
  - isNormalized, [61](#)
  - length, [62](#)
  - lengthSquared, [63](#)
  - log, [64](#)
  - meanTangent, [65](#)
  - mult, [65–67](#)
  - negate, [68](#)
  - normal, [69](#)
  - normalize, [69](#), [70](#)
  - operator \*, [71–73](#)
  - operator \*=, [74](#), [75](#)
  - operator+, [76](#)
  - operator+=, [77](#), [78](#)
  - operator-, [78](#), [79](#)
  - operator-=, [80](#)
  - operator/, [81](#)
  - operator/=: [82](#)
  - postMult, [83](#)
  - preMult, [83](#)
  - squad, [84](#)
  - sub, [84](#)
  - transpose, [85](#), [86](#)
  - whichSide, [86](#), [87](#)
  - zero, [87](#)
- Order
  - gmtl::EulerAngle, [283](#)
- OriginalType
  - gmtl::Type2Type, [346](#)
- ORTHOGONAL
  - gmtl::Matrix, [297](#)
- ORTHONORMAL
  - gmtl::Matrix, [297](#)
- Output
  - operator <<, [164–166](#)
- Output Stream Methods: operator <<(
  - ... ), [163](#)
- Output.h, [393](#)
- PI
  - Math, [36](#)
- PI\_OVER\_2
  - Math, [37](#)
- PI\_OVER\_4
  - Math, [37](#)

- Plane
  - gmtl::Plane, [317–319](#)
- Plane.h, [394](#)
- Planned
  - gmtl, [202](#)
- Planef
  - gmtl, [202](#)
- PlaneOps.h, [395](#)
- PlaneSide
  - Defines, [18](#)
- Point
  - gmtl::Point, [324](#), [325](#)
- Point.h, [396](#)
- Point3d
  - gmtl, [202](#)
- Point3f
  - gmtl, [202](#)
- Point4d
  - gmtl, [203](#)
- Point4f
  - gmtl, [203](#)
- points
  - gmtl::CompareIndexPoint-  
Projections, [253](#)
- pos
  - gmtl::Coord, [258](#)
- POS\_SIDE
  - Defines, [18](#)
- PosDataType
  - gmtl::Coord, [255](#)
- PosSize
  - gmtl::Coord, [256](#)
- postMult
  - Ops, [83](#)
- pow
  - Math, [32](#)
- preMult
  - Ops, [83](#)
- QLAlgorithm
  - gmtl::Eigen, [270](#)
- Quat
  - gmtl::Quat, [329](#)
- Quat.h, [397](#)
- QUAT\_ADD\_IDENTITYD
  - gmtl, [217](#)
- QUAT\_ADD\_IDENTITYF
  - gmtl, [217](#)
- QUAT\_IDENTITYD
  - gmtl, [217](#)
- QUAT\_IDENTITYF
  - gmtl, [218](#)
- QUAT\_MULT\_IDENTITYD
  - gmtl, [218](#)
- QUAT\_MULT\_IDENTITYF
  - gmtl, [218](#)
- Quatd
  - gmtl, [203](#)
- Quatf
  - gmtl, [203](#)
- QuatOps.h, [398](#)
- rad2Deg
  - Math, [32](#)
- rangeRandom
  - Math, [32](#)
- rot
  - gmtl::Coord, [258](#)
- RotDataType
  - gmtl::Coord, [256](#)
- RotSize
  - gmtl::Coord, [256](#)
- round
  - Math, [33](#)
- Rows
  - gmtl::Matrix, [297](#)
- set
  - Generate, [135–137](#), [139–141](#),  
[143](#)
  - gmtl, [218](#)
  - gmtl::AxisAngle, [249](#), [250](#)
  - gmtl::EulerAngle, [286](#)
  - gmtl::Matrix, [301–305](#)
  - gmtl::Quat, [332](#)
  - gmtl::VecBase, [358](#), [359](#)
- setAngle
  - gmtl::AxisAngle, [250](#)
- setAxes
  - Generate, [145](#)
- setAxis
  - gmtl::AxisAngle, [250](#)

- setCenter
  - gmtl::Sphere, 338
- setDir
  - gmtl::LineSeg, 291
- setDirCos
  - Generate, 145
- setEmpty
  - gmtl::AABBox, 242
- setError
  - gmtl::Matrix, 305
- SetMatrix
  - gmtl::Eigen, 272
- setMax
  - gmtl::AABBox, 242
- setMin
  - gmtl::AABBox, 242
- setNormal
  - gmtl::Plane, 320
- setOffset
  - gmtl::Plane, 320
- setOrigin
  - gmtl::LineSeg, 292
- setPure
  - Generate, 146
- setRadius
  - gmtl::Sphere, 339
- setRot
  - Generate, 147–152
- setScale
  - Generate, 154
- setTrans
  - Generate, 154, 156
- setTranspose
  - gmtl::Matrix, 306
- sign
  - Math, 33
- sin
  - Math, 33, 34
- Size
  - gmtl::AxisAngle, 247
  - gmtl::EulerAngle, 283
  - gmtl::Point, 324
  - gmtl::Vec, 349
  - gmtl::VecBase, 354
- slerp
  - Interp, 161
- sortDir
  - gmtl::CompareIndexPoint-  
Projections, 253
- Spacial Transformers: xform( ... ), op-  
erator( ... ), 89
- Sphere
  - gmtl::Sphere, 337
- Sphere.h, 399
- Sphered
  - gmtl, 203
- Spheref
  - gmtl, 203
- SphereOps.h, 400
- sqr
  - Math, 34
- sqrt
  - Math, 34
- squad
  - Ops, 84
- sub
  - Ops, 84
- tan
  - Math, 35
- Template Metaprogramming Utilities,  
168
- Template Metaprogramming Utilities  
(Helpers), 169
- TestIntersect
  - gmtl, 219
- TestIntersectOBB
  - gmtl, 222
- Transforms
  - operator \*, 90–93
  - xform, 93–97
- transpose
  - Ops, 85, 86
- Tri
  - gmtl::Tri, 342, 343
- Tri.h, 401
- Tridiagonal2
  - gmtl::Eigen, 272
- Tridiagonal3
  - gmtl::Eigen, 273
- Tridiagonal4
  - gmtl::Eigen, 274

- TridiagonalN
  - gmtl::Eigen, [277](#)
- TriOps.h, [402](#)
- trunc
  - Math, [35](#)
- unitRandom
  - Math, [36](#)
- Vec
  - gmtl::Vec, [349–351](#)
- Vec.h, [403](#)
- Vec2d
  - gmtl, [203](#)
- Vec2f
  - gmtl, [203](#)
- Vec3d
  - gmtl, [203](#)
- Vec3f
  - gmtl, [203](#)
- Vec4d
  - gmtl, [204](#)
- Vec4f
  - gmtl, [204](#)
- VecBase
  - gmtl::VecBase, [355](#), [356](#)
- VecBase.h, [404](#)
- VecOps.h, [405](#)
- VectorIndex
  - Defines, [18](#)
- version
  - gmtl, [232](#)
- Version.h, [406](#)
  - GMTL\_DOT, [407](#)
  - GMTL\_GLUE, [407](#)
  - GMTL\_STR, [407](#)
  - GMTL\_VERSION, [407](#)
  - GMTL\_VERSION\_MAJOR, [407](#)
  - GMTL\_VERSION\_MAJOR\_-  
FILLED, [407](#)
  - GMTL\_VERSION\_MINOR, [407](#)
  - GMTL\_VERSION\_MINOR\_-  
FILLED, [408](#)
  - GMTL\_VERSION\_PATCH, [408](#)
  - GMTL\_VERSION\_PATCH\_-  
FILLED, [408](#)
- GMTL\_VERSION\_STRING, [408](#)
- GMTL\_XDOT, [408](#)
- GMTL\_XGLUE, [408](#)
- GMTL\_XSTR, [408](#)
- GMTL\_XZEROFILL, [408](#)
- GMTL\_ZEROFILL, [409](#)
- Welt
  - Defines, [19](#)
- whichSide
  - Ops, [86](#), [87](#)
- Xelt
  - Defines, [19](#)
- xform
  - Transforms, [93–97](#)
- XFORM\_ERROR
  - gmtl::Matrix, [297](#)
- Xforms.h, [410](#)
- XformState
  - gmtl::Matrix, [297](#)
- Yelt
  - Defines, [19](#)
- Zelt
  - Defines, [19](#)
- zero
  - Ops, [87](#)
- zeroClamp
  - Math, [36](#)