GenericMathTemplateLibrary Reference Manual

0.0.5

Generated by Doxygen 1.2.15

Thu Jun 13 18:17:01 2002

Contents

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 mantaining 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 x 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) then 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:

Chapter 3

GenericMathTemplateLibrary Namespace Index

3.1 GenericMathTemplateLibrary Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 4

GenericMathTemplateLibrary Hierarchical Index

4.1 GenericMathTemplateLibrary Class Hierarchy

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

Chapter 5

GenericMathTemplateLibrary Compound Index

5.1 GenericMathTemplateLibrary Compound List

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

Chapter 6

GenericMathTemplateLibrary File Index

6.1 GenericMathTemplateLibrary File List

Here is a list of all files with brief descriptions:

Chapter 7

GenericMathTemplateLibrary Page Index

7.1 GenericMathTemplateLibrary Related Pages

Here is a list of all related documentation pages:

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().

```
{
   ON_PLANE,
   POS_SIDE,
   NEG_SIDE
};
```

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().

```
{
    if ( -1.0 < fValue )
    {
        if ( fValue < 1.0 )
            return double( ::acos( fValue ) );
        else
            return 0.0;
    }
    else
    {
        return (double)gmtl::Math::PI;
    }
}
```

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

Definition at line 134 of file Math.h.

```
{
    if ( -1.0f < fValue )
    {
        if ( fValue < 1.0f )
            return float( ::acosf( fValue ) );
        else
            return 0.0f;
    }
    else
    {
        return (float)gmtl::Math::PI;
    }
}
```

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().

```
{
```

```
    if ( -1.0 < fValue )
    {
        if ( fValue < 1.0 )
            return double( ::asin( fValue ) );
        else
            return (double)-gmtl::Math::PI_OVER_2;
    }
    else
    {
        return (double)gmtl::Math::PI_OVER_2;
    }
}
```

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

Definition at line 167 of file Math.h.

```
{
    if ( -1.0f < fValue )
    {
        if ( fValue < 1.0f )
            return float( ::asinf( fValue ) );
        else
            return (float)-gmtl::Math::PI_OVER_2;
    }
    else
    {
        return (float)gmtl::Math::PI_OVER_2;
    }
}
```

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.

```
{
    return float( ::atanf( fValue ) );
}
```

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

Definition at line 199 of file Math.h.

```
{
    return ::atan( fValue );
}
```

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().

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

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

Definition at line 213 of file Math.h.

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

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().

```
{
    return double( ::ceil( fValue ) );
}
```

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

Definition at line 87 of file Math.h.

```
{
    return float( ::ceilf( fValue ) );
}
```

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().

```
{
    return double( ::cos( fValue ) );
}
```

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

Definition at line 226 of file Math.h.

```
{
    return float( ::cosf( fValue ) );
}
```

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.

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

8.2.2.21 float deg2Rad (float *fVal*) [inline]

Definition at line 338 of file Math.h.

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

8.2.2.22 double exp (double *fValue*) [inline]

Definition at line 242 of file Math.h.

References gmtl::Math::exp().

```
{
    return double( ::exp( fValue ) );
}
```

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

Definition at line 238 of file Math.h.

```
{
    return float( ::expf( fValue ) );
}
```

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.

```
{
   T lhs = (T)1;

   for( T x = (T)1; x <= rhs; ++x )
   {
      lhs *= x;
   }

   return lhs;
}
```

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

Definition at line 102 of file Math.h.

References gmtl::Math::floor().

```
{
    return double( ::floor( fValue ) );
}
```

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

Definition at line 98 of file Math.h.

```
{
    return float( ::floorf( fValue ) );
}
```

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 >= 0

Definition at line 362 of file Math.h.

References gmtlASSERT.

```
{
   gmtlASSERT( tolerance >= (T)0 );
   return bool( gmtl::Math::abs( a - b ) <= tolerance );
}
```

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

Definition at line 253 of file Math.h.

```
{
    return float( ::logf( fValue ) );
}
```

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

Definition at line 249 of file Math.h.

References gmtl::Math::log().

```
{
    return double( ::log( fValue ) );
}
```

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().

```
{
   return gmtl::Math::Max( gmtl::Math::Max( w, x ), gmtl::Math::Max( y, z ) );
}
```

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().

```
{
   return Max( gmtl::Math::Max( x, y ), z );
}
```

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().

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

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().

```
{
   return gmtl::Math::Min( gmtl::Math::Min( w, x ), gmtl::Math::Min( y, z ) );
}
```

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().

```
{
   return Min( gmtl::Math::Min( x, y ), z );
}
```

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().

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

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

Definition at line 262 of file Math.h.

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

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

Definition at line 258 of file Math.h.

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

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

Definition at line 351 of file Math.h.

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

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

Definition at line 347 of file Math.h.

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

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().

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

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

round to nearest integer.

Definition at line 376 of file Math.h.

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

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

Definition at line 108 of file Math.h.

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

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().

```
{
    return float( ::sinf( fValue ) );
}
```

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

Definition at line 269 of file Math.h.

References gmtl::Math::sin().

```
{
    return double( ::sin( fValue ) );
}
```

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.

```
{
    return T( fValue * fValue );
}
```

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().

```
{
    return double( ::sqrt( fValue ) );
}
```

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().

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

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

Definition at line 287 of file Math.h.

```
{
    return float( ::tanf( fValue ) );
}
```

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

Definition at line 282 of file Math.h.

References gmtl::Math::tan().

```
{
    return double( ::tan( fValue ) );
}
```

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().

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

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().

```
{
   return float(::rand())/float(RAND_MAX);
}
```

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().

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

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.

```
{
   result[0] = q1[0] + q2[0];
   result[1] = q1[1] + q2[1];
   result[2] = q1[2] + q2[2];
   result[3] = q1[3] + q2[3];
   return result;
}
```

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

Definition at line 162 of file MatrixOps.h.

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

```
   {
      // p. 150 Numerical Analysis (second ed.)
      // if A is m x n, and B is m x n, then AB is m x n
      // (A - B)ij  = (a)ij + (b)ij     (where:  1 <= i <= m, 1 <= j <= n)
      for (unsigned int i = 0; i < ROWS; ++i)          // 1 <= i <= m
      for (unsigned int j = 0; j < COLS; ++j)          // 1 <= j <= n
         result( i, j ) = lhs( i, j ) + rhs( i, j );

      return result;
   }
```

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().

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

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().

```
{
   result[Xelt] = -result[Xelt];
   result[Yelt] = -result[Yelt];
   result[Zelt] = -result[Zelt];
   return result;
}
```

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 x 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().

```
{
   result.set( (v1[Yelt]*v2[Zelt]) - (v1[Zelt]*v2[Yelt]),
               (v1[Zelt]*v2[Xelt]) - (v1[Xelt]*v2[Zelt]),
               (v1[Xelt]*v2[Yelt]) - (v1[Yelt]*v2[Xelt]) );
   return result;
}
```

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().

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

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 gmtl::dot().

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

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:

result = q / s

See also:

Quat

Definition at line 213 of file QuatOps.h.

```
{
   result[0] = q[0] / s;
   result[1] = q[1] / s;
   result[2] = q[2] / s;
   result[3] = q[3] / s;
   return result;
}
```

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:
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/=().

```
{
   Quat<DATA_TYPE> q2_inv( q2 ), r, s;

   // conj the quat
   conj( q2_inv );

   mult( r, q1, q2_inv );
   mult( s, q2_inv, q2_inv );

   float sw_inv = 1.0f / s[Welt];
   result[0] = r[0] * sw_inv;
   result[1] = r[1] * sw_inv;
   result[2] = r[2] * sw_inv;
   result[3] = r[3] * sw_inv;
   return result;
}
```

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
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().

```
{
   return DATA_TYPE( (q1[0] * q2[0]) +
                     (q1[1] * q2[1]) +
                     (q1[2] * q2[2]) +
                     (q1[3] * q2[3])  );
}
```

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().

```
{
   DATA_TYPE ret_val(0);
   for(unsigned i=0;i<SIZE;++i)
   {
      ret_val += (v1[i] * v2[i]);
   }
   return ret_val;
}
```

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 gmtl::Math::cos(), gmtl::Math::sin(), gmtl::Math::sqrt(), gmtl::Welt, gmtl::Xelt, gmtl::Yelt, and gmtl::Zelt.

```
{
   DATA_TYPE len1, len2;

   len1 = Math::sqrt( result[Xelt] * result[Xelt] +
                      result[Yelt] * result[Yelt] +
                      result[Zelt] * result[Zelt] );
   if (len1 > (DATA_TYPE)0.0)
      len2 = Math::sin( len1 ) / len1;
   else
      len2 = (DATA_TYPE)1.0;

   result[Xelt] = result[Xelt] * len2;
   result[Yelt] = result[Yelt] * len2;
   result[Zelt] = result[Zelt] * len2;
   result[Welt] = Math::cos( len1 );

   return result;
}
```

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().

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

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().

```
  {
     if(result.mState != Matrix<DATA_TYPE, ROWS, COLS>::IDENTITY)   // if not already ident
     {
        // TODO: mp
        for (unsigned int r = 0; r < ROWS; ++r)
        for (unsigned int c = 0; c < COLS; ++c)
           result( r, c ) = (DATA_TYPE)0.0;

        // TODO: mp
        for (unsigned int x = 0; x < Math::Min( COLS, ROWS ); ++x)
           result( x, x ) = (DATA_TYPE)1.0;

//         result.mState = Matrix<DATA_TYPE, ROWS, COLS>::IDENTITY;
        result.mState = Matrix<DATA_TYPE, ROWS, COLS>::FULL;
     }

     return result;
  }
```

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.

```
{
   // do result = conj( q ) / norm( q )
   conj( result );

   // return if norm() is near 0 (divide by 0 would result in NaN)
   DATA_TYPE l = lengthSquared( result );
   if (l < (DATA_TYPE)0.0001)
      return result;

   DATA_TYPE l_inv = ((DATA_TYPE)1.0) / l;
   result[Xelt] *= l_inv;
   result[Yelt] *= l_inv;
   result[Zelt] *= l_inv;
   result[Welt] *= l_inv;
   return result;
}
```

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().

```
{
   return invert( result, result );
}
```

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().

```
{
   if (src.mState == Matrix<DATA_TYPE, ROWS, COLS>::IDENTITY )
      return result = src;
   else
      return invertFull( result, src );
}
```

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().

```
{
   /*---------------------------------------------------------------------------*
    | mat_inv: Compute the inverse of a n x n matrix, using the maximum pivot   |
    |          strategy.  n <= MAX1.                                            |
    *---------------------------------------------------------------------------*

      Parameters:
          a        a n x n square matrix
          b        inverse of input a.
          n        dimenstion of matrix a.
   */

   const DATA_TYPE* a = src.getData();
   DATA_TYPE* b = result.mData;

   int   n = 4;
   int    i, j, k;
   int    r[ 4], c[ 4], row[ 4], col[ 4];
   DATA_TYPE  m[ 4][ 4*2], pivot, max_m, tmp_m, fac;

   /* Initialization */
   for ( i = 0; i < n; i ++ )
   {
      r[ i] = c[ i] = 0;
      row[ i] = col[ i] = 0;
   }

   /* Set working matrix */
   for ( i = 0; i < n; i++ )
   {
      for ( j = 0; j < n; j++ )
      {
         m[ i][ j] = a[ i * n + j];
         m[ i][ j + n] = ( i == j ) ? (DATA_TYPE)1.0 : (DATA_TYPE)0.0 ;
      }
   }

   /* Begin of loop */
   for ( k = 0; k < n; k++ )
   {
      /* Choosing the pivot */
      for ( i = 0, max_m = 0; i < n; i++ )
      {
         if ( row[ i]  )
            continue;
         for ( j = 0; j < n; j++ )
         {
            if ( col[ j] )
               continue;
```

```
                tmp_m = gmtl::Math::abs( m[ i][ j]);
                if ( tmp_m > max_m)
                {
                   max_m = tmp_m;
                   r[ k] = i;
                   c[ k] = j;
                }
             }
          }
          row[ r[k] ] = col[ c[k] ] = 1;
          pivot = m[ r[ k] ][ c[ k] ];


          if ( gmtl::Math::abs( pivot) <= 1e-20)
          {
             std::cerr << "*** pivot = %f in mat_inv. ***\n";
             result.setError();
             return result;
          }

          /* Normalization */
          for ( j = 0; j < 2*n; j++ )
          {
             if ( j == c[ k] )
                m[ r[ k]][ j] = (DATA_TYPE)1.0;
             else
                m[ r[ k]][ j] /= pivot;
          }

          /* Reduction */
          for ( i = 0; i < n; i++ )
          {
             if ( i == r[ k] )
                continue;

             for ( j=0, fac = m[ i][ c[k]]; j < 2*n; j++ )
             {
                if ( j == c[ k] )
                   m[ i][ j] = (DATA_TYPE)0.0;
                else
                   m[ i][ j] -= fac * m[ r[k]][ j];
             }
          }
       }

       /* Assign inverse to a matrix */
       for ( i = 0; i < n; i++ )
          for ( j = 0; j < n; j++ )
             row[ i] = ( c[ j] == i ) ? r[ j] : row[ i];

       for ( i = 0; i < n; i++ )
          for ( j = 0; j < n; j++ )
             b[ i * n +  j] = m[ row[ i]][ j + n];

       // It worked
```

```
    return result;
}
```

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().

```
{
   return Math::isEqual( lengthSquared( v1 ), (DATA_TYPE)1.0, eps );
}
```

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().

```
{
   return Math::isEqual( lengthSquared( q1 ), DATA_TYPE(1), eps );
}
```

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().

```
{
   DATA_TYPE ret_val = lengthSquared(v1);
   if (ret_val == 0.0f)
      return 0.0f;
   else
      return Math::sqrt(ret_val);
}
```

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
308    return ret_val;
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().

```
{
   return dot( q, q );
}
```

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.

```
{
   DATA_TYPE length;

   length = Math::sqrt( result[Xelt] * result[Xelt] +
                        result[Yelt] * result[Yelt] +
                        result[Zelt] * result[Zelt] );

   // avoid divide by 0
   if (Math::isEqual( result[Welt], (DATA_TYPE)0.0, (DATA_TYPE)0.00001 ) == false)
      length = Math::atan( length / result[Welt] );
   else
```

```
        length = Math::PI_OVER_2;

    result[Welt] = (DATA_TYPE)0.0;
    result[Xelt] = result[Xelt] * length;
    result[Yelt] = result[Yelt] * length;
    result[Zelt] = result[Zelt] * length;
    return result;
}
```

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.

```
{
    gmtlASSERT( false );
}
```

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.

```
{
    result[0] = q[0] * s;
    result[1] = q[1] * s;
    result[2] = q[2] * s;
    result[3] = q[3] * s;
    return result;
}
```

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.

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

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.

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

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 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

Definition at line 105 of file MatrixOps.h.

References gmtl::zero().

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

```
{
   Matrix<DATA_TYPE, ROWS, COLS> ret_mat; // prevent aliasing
   zero( ret_mat );

   // p. 150 Numerical Analysis (second ed.)
   // if A is m x p, and B is p x n, then AB is m x n
   // (AB)ij  =  [k = 1 to p] (a)ik (b)kj     (where:  1 <= i <= m, 1 <= j <= n)
   for (unsigned int i = 0; i < ROWS; ++i)           // 1 <= i <= m
   for (unsigned int j = 0; j < COLS; ++j)           // 1 <= j <= n
   for (unsigned int k = 0; k < INTERNAL; ++k)       // [k = 1 to p]
      ret_mat( i, j ) += lhs( i, k ) * rhs( k, j );

   return result = ret_mat;
}
```

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.

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

```

    // Here is another version (euclidean product, just FYI)...
    // scalar_component = q1.s * q2.s + dot(q1.v, q2.v)
    // vector_component = q2.v * q1.s - q1.v * q2.s - cross(q1.v, q2.v)

    // Here it is, using vector algebra (grassman product)
    /*
    const float& w1( q1[Welt] ), w2( q2[Welt] );
    Vec3 v1( q1[Xelt], q1[Yelt], q1[Zelt] ), v2( q2[Xelt], q2[Yelt], q2[Zelt] );

    float w = w1 * w2 - v1.dot( v2 );
    Vec3 v = (w1 * v2) + (w2 * v1) + v1.cross( v2 );

    vec[Welt] = w;
    vec[Xelt] = v[0];
    vec[Yelt] = v[1];
    vec[Zelt] = v[2];
    */

    // Here is the same, only expanded... (grassman product)
    Quat<DATA_TYPE> temporary; // avoid aliasing problems...
    temporary[Xelt] = q1[Welt]*q2[Xelt] + q1[Xelt]*q2[Welt] + q1[Yelt]*q2[Zelt] - q1[Zelt
    temporary[Yelt] = q1[Welt]*q2[Yelt] + q1[Yelt]*q2[Welt] + q1[Zelt]*q2[Xelt] - q1[Xelt
    temporary[Zelt] = q1[Welt]*q2[Zelt] + q1[Zelt]*q2[Welt] + q1[Xelt]*q2[Yelt] - q1[Yelt
    temporary[Welt] = q1[Welt]*q2[Welt] - q1[Xelt]*q2[Xelt] - q1[Yelt]*q2[Yelt] - q1[Zelt

    // use a temporary, in case q1 or q2 is the same as self.
    result[Xelt] = temporary[Xelt];
    result[Yelt] = temporary[Yelt];
    result[Zelt] = temporary[Zelt];
    result[Welt] = temporary[Welt];

    // don’t normalize, because it might not be rotation arithmetic we’re doing
    // (only rotation quats have unit length)
    return result;
}
```

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 equivelent 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.

```
   {
      result[0] = -result[0];
      result[1] = -result[1];
      result[2] = -result[2];
      result[3] = -result[3];
      return result;
   }
```

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().

```
{
   Vec<DATA_TYPE, 3> normal = cross( tri[1] - tri[0], tri[2] - tri[0] );
   normalize( normal );
   return normal;
}
```

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().

```
{
   DATA_TYPE len = length(v1);

   if(len != 0.0f)
   {
      for(unsigned i=0;i<SIZE;++i)
      {
         v1[i] /= len;
      }
   }

   return len;
}
```

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().

```
{
   DATA_TYPE l = length( result );

   // return if no magnitude (already as normalized as possible)
   if (l < (DATA_TYPE)0.0001)
      return result;

   DATA_TYPE l_inv = ((DATA_TYPE)1.0) / l;
   result[Xelt] *= l_inv;
   result[Yelt] *= l_inv;
   result[Zelt] *= l_inv;
   result[Welt] *= l_inv;

   return result;
}
```

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 result = scalar ∗ v1. This is equivalent to result = v1 ∗ 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.

```
{
   VecBase<DATA_TYPE, SIZE> ret_val(v1);
   ret_val *= scalar;
   return ret_val;

   //return VecBase<DATA_TYPE, SIZE>(v1) *= scalar;
}
```

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.

```
{
   VecBase<DATA_TYPE, SIZE> ret_val(v1);
   ret_val *= scalar;
   return ret_val;

   //return VecBase<DATA_TYPE, SIZE>(v1) *= scalar;
}
```

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().

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

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.

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

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().

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

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 mutiplied by scalar

Definition at line 156 of file VecOps.h.

```
{
   for(unsigned i=0;i<SIZE;++i)
   {
      v1[i] *= scalar;
   }

   return v1;
}
```

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().

```
{
   return mult( q, q, s );
}
```

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().

```
{
   return mult( result, result, q2 );
}
```

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().

```
{
   return mult( result, scalar );
}
```

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().

```
{
   return postMult( result, operand );
}
```

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.

```
{
   VecBase<DATA_TYPE, SIZE> ret_val(v1);
   ret_val += v2;
   return ret_val;
}
```

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().

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

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.

```
{
   for(unsigned i=0;i<SIZE;++i)
   {
      v1[i] += v2[i];
   }

   return v1;
}
```

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().

```
{
   return add( q1, q1, q2 );
}
```

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.

```
{
   Vec<DATA_TYPE, SIZE> ret_val(v1);
   ret_val -= v2;
   return ret_val;
}
```

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().

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

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 equivelent 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.

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

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.

```
{
   Vec<DATA_TYPE, SIZE> ret_val;
   for ( unsigned i=0; i < SIZE; ++i )
   {
      ret_val[i] = -v1[i];
   }
   return ret_val;
}
```

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.

```
{
   for(unsigned i=0;i<SIZE;++i)
   {
      v1[i] -= v2[i];
   }

   return v1;
}
```

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().

```
{
   return sub( q1, q1, q2 );
}
```

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.

```
{
   VecBase<DATA_TYPE, SIZE> ret_val(v1);
   ret_val /= scalar;
   return ret_val;
   // return VecBase<DATA_TYPE, SIZE>(v1)( /= scalar;
}
```

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().

```
{
   return div( q, q, s );
}
```

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 ∗=().

```
{
   return mult( result, result, operand );
}
```

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().

```
{
   return mult( result, operand, result );
}
```

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.

```
{
   gmtlASSERT( false );
}
```

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.

```
{
   result[0] = q1[0] - q2[0];
   result[1] = q1[1] - q2[1];
   result[2] = q1[2] - q2[2];
   result[3] = q1[3] - q2[3];
   return result;
}
```

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

Definition at line 142 of file MatrixOps.h.

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

```
{
   // p. 150 Numerical Analysis (second ed.)
   // if A is m x n, and B is m x n, then AB is m x n
   // (A - B)ij  = (a)ij - (b)ij     (where:  1 <= i <= m, 1 <= j <= n)
   for (unsigned int i = 0; i < ROWS; ++i)           // 1 <= i <= m
   for (unsigned int j = 0; j < COLS; ++j)           // 1 <= j <= n
      result( i, j ) = lhs( i, j ) - rhs( i, j );

   return result;
}
```

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.

3x4 to 4x3) @PRE: source needs to be an M x N matrix, while dest needs to be N x M @POST: flip along diagonal

Definition at line 264 of file MatrixOps.h.

```
{
   // in case result is == source... :(
   Matrix<DATA_TYPE, COLS, ROWS> temp = source;

   // p. 149 Numerical Analysis (second ed.)
   for (unsigned i = 0; i < ROWS; ++i)
   {
      for (unsigned j = 0; j < COLS; ++j)
      {
         result( i, j ) = temp( j, i );
      }
```

```
   }

   return result;
}
```

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().

```
{
   // p. 27 game programming gems #1
   for (unsigned c = 0; c < SIZE; ++c)
      for (unsigned r = c + 1; r < SIZE; ++r)
         std::swap( result( r, c ), result( c, r ) );

   return result;
}
```

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.

```
{
   DATA_TYPE dist = distance( plane, pt );

   if ( dist < eps )
      return NEG_SIDE;
   else if ( dist > eps )
      return POS_SIDE;
   else
      return ON_PLANE;
}
```

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.

```
{
   DATA_TYPE dist = distance( plane, pt );

   if ( dist < DATA_TYPE(0) )
      return NEG_SIDE;
   else if ( dist > DATA_TYPE(0) )
      return POS_SIDE;
   else
      return ON_PLANE;
}
```

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().

```
{
   if (result.mState == Matrix<DATA_TYPE, ROWS, COLS>::IDENTITY)
   {
      for (unsigned int x = 0; x < Math::Min( ROWS, COLS ); ++x)
      {
         result( x, x ) = (DATA_TYPE)0;
      }
   }
   else
   {
      for (unsigned int x = 0; x < ROWS*COLS; ++x)
      {
         result[x] = (DATA_TYPE)0;
      }
   }
   return result;
}
```

8.5 Spacial 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().

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

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().

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

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().

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

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().

```
{
```

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

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 conj(q), and P(v) is pure quaternion made from v)

Definition at line 95 of file Xforms.h.

References gmtl::xform().

```
{
   VecBase<DATA_TYPE, 3> temporary;
   return xform( temporary, rot, vector );
}
```

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().

```
{
   gmtlASSERT( PNT_SIZE == COLS - 1 && "The precondition of this method is that the vect

   // copy the point to the correct size.
   Point<DATA_TYPE, PNT_SIZE+1> temp_point, temp_result;
   for (unsigned x = 0; x < PNT_SIZE; ++x)
      temp_point[x] = point[x];
   temp_point[PNT_SIZE] = (DATA_TYPE)1.0; // by definition of a point, set the last unsp

   // transform it.
   xform<DATA_TYPE, ROWS, COLS>( temp_result, matrix, temp_point );

   // convert result back to pnt<DATA_TYPE, PNT_SIZE>
   // some matrices will make the W param large even if this is a true vector,
   // we'll need to redistribute it to the other elts if W param is non-zero
   if (Math::isEqual( temp_result[PNT_SIZE], (DATA_TYPE)0, (DATA_TYPE)0.0001 ) == false)
   {
      DATA_TYPE w_coord_div = DATA_TYPE( 1.0 ) / temp_result[PNT_SIZE];
      for (unsigned x = 0; x < PNT_SIZE; ++x)
         result[x] = temp_result[x] * w_coord_div;
   }
   else
   {
      for (unsigned x = 0; x < PNT_SIZE; ++x)
         result[x] = temp_result[x];
   }

   return result;
}
```

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 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 155 of file Xforms.h.

References gmtlASSERT, and gmtl::isEqual().

```
{
   gmtlASSERT( VEC_SIZE == COLS - 1 );
   // do a standard [m x k] by [k x n] matrix multiplication (where n == 0).

   // copy the point to the correct size.
   Vec<DATA_TYPE, COLS> temp_vector, temp_result;
   for (unsigned x = 0; x < VEC_SIZE; ++x)
      temp_vector[x] = vector[x];
   temp_vector[COLS-1] = (DATA_TYPE)0.0; // by definition of a vector, set the last unspecified elt

   // transform it.
   xform<DATA_TYPE, ROWS, COLS>( temp_result, matrix, temp_vector );

   // convert result back to vec<DATA_TYPE, VEC_SIZE>
   // some matrices will make the W param large even if this is a true vector,
   // we'll need to redistribute it to the other elts if W param is non-zero
   if (Math::isEqual( temp_result[VEC_SIZE], (DATA_TYPE)0, (DATA_TYPE)0.0001 ) == false)
   {
      DATA_TYPE w_coord_div = DATA_TYPE( 1.0 ) / temp_result[VEC_SIZE];
      for (unsigned x = 0; x < VEC_SIZE; ++x)
         result[x] = temp_result[x] * w_coord_div;
   }
   else
   {
      for (unsigned x = 0; x < VEC_SIZE; ++x)
         result[x] = temp_result[x];
   }

   return result;
}
```

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 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 213 of file Xforms.h.

```
{
   // do a standard [m x k] by [k x n] matrix multiplication (n == 1).

   // reset point to zero...
   result = Point<DATA_TYPE, COLS>();

   for (unsigned iRow = 0; iRow < ROWS; ++iRow)
   for (unsigned iCol = 0; iCol < COLS; ++iCol)
      result[iRow] += matrix( iRow, iCol ) * point[iCol];

   return result;
}
```

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.

```
{
   // do a standard [m x k] by [k x n] matrix multiplication (where n == 0).

   // reset vec to zero...
   result = Vec<DATA_TYPE, COLS>();

   for (unsigned iRow = 0; iRow < ROWS; ++iRow)
   for (unsigned iCol = 0; iCol < COLS; ++iCol)
      result[iRow] += matrix( iRow, iCol ) * vector[iCol];

```

```
    return result;
}
```

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

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

```
{
   // check preconditions...
   gmtlASSERT( Math::isEqual( length( rot ), (DATA_TYPE)1.0, (DATA_TYPE)0.0001 ) && "must pass a rot

   // easiest to write and understand (slowest too)
   //return result_vec = makeVec( rot * makePure( vector ) * makeConj( rot ) );

   // completely hand expanded
   // (faster by 28% in gcc 2.96 debug mode.)
```

```
    // (faster by 35% in gcc 2.96 opt3 mode (78% for doubles))
    Quat<DATA_TYPE> rot_conj( -rot[Xelt], -rot[Yelt], -rot[Zelt], rot[Welt] );
    Quat<DATA_TYPE> pure( vector[0], vector[1], vector[2], (DATA_TYPE)0.0 );
    Quat<DATA_TYPE> temp(
       pure[Welt]*rot_conj[Xelt] + pure[Xelt]*rot_conj[Welt] + pure[Yelt]*rot_conj[Zelt] -
       pure[Welt]*rot_conj[Yelt] + pure[Yelt]*rot_conj[Welt] + pure[Zelt]*rot_conj[Xelt] -
       pure[Welt]*rot_conj[Zelt] + pure[Zelt]*rot_conj[Welt] + pure[Xelt]*rot_conj[Yelt] -
       pure[Welt]*rot_conj[Welt] - pure[Xelt]*rot_conj[Xelt] - pure[Yelt]*rot_conj[Yelt] -

    result.set(
       rot[Welt]*temp[Xelt] + rot[Xelt]*temp[Welt] + rot[Yelt]*temp[Zelt] - rot[Zelt]*temp
       rot[Welt]*temp[Yelt] + rot[Yelt]*temp[Welt] + rot[Zelt]*temp[Xelt] - rot[Xelt]*temp
       rot[Welt]*temp[Zelt] + rot[Zelt]*temp[Welt] + rot[Xelt]*temp[Yelt] - rot[Yelt]*temp
    return result;
}
```

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 equivelence (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 >= 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().

```
{
   gmtlASSERT(eps >= 0);

   for(unsigned i=0;i<SIZE;++i)
   {
      if (fabs(v1[i] - v2[i]) > eps)
      {
         return false;
      }
   }
   return true;
}
```

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 >= 0

Returns:
true if they are equal, false otherwise

Definition at line 126 of file TriOps.h.

References gmtlASSERT, and gmtl::isEqual().

```
{
   gmtlASSERT( eps >= 0 );
   return ( isEqual(tri1[0], tri2[0], eps) &&
            isEqual(tri1[1], tri2[1], eps) &&
            isEqual(tri1[2], tri2[2], eps) );
}
```

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 >= 0

Returns:
true if they are equal, false otherwise

Definition at line 91 of file SphereOps.h.

References gmtlASSERT, and gmtl::isEqual().

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

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.

```
{
   return bool( Math::isEqual( q1[0], q2[0], tol ) &&
                Math::isEqual( q1[1], q2[1], tol ) &&
                Math::isEqual( q1[2], q2[2], tol ) &&
                Math::isEqual( q1[3], q2[3], tol )    );
}
```

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 >= 0

Returns:

true if they are equal, false otherwise

Definition at line 186 of file PlaneOps.h.

References gmtlASSERT, and gmtl::isEqual().

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

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 >= 0

Definition at line 469 of file MatrixOps.h.

References gmtlASSERT, and gmtl::isEqual().

```
{
   gmtlASSERT( eps >= (DATA_TYPE)0 );

   for (unsigned int i = 0; i < ROWS*COLS; ++i)
   {
      if (!Math::isEqual( lhs[i], rhs[i], eps ))
         return false;
   }
   return true;
}
```

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 >= 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().

```
{
   gmtlASSERT(eps >= (DATA_TYPE)0);

   // @todo metaprogramming.
   if (!Math::isEqual( v1[0], v2[0], eps )) return false;
   if (!Math::isEqual( v1[1], v2[1], eps )) return false;
   if (!Math::isEqual( v1[2], v2[2], eps )) return false;
   return true;
}
```

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().

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

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().

```
{
   gmtlASSERT( eps >= (DATA_TYPE)0 );

   // @todo metaprogramming.
   if (!Math::isEqual( v1[0], v2[0], eps )) return false;
   if (!Math::isEqual( v1[1], v2[1], eps )) return false;
   if (!Math::isEqual( v1[2], v2[2], eps )) return false;
   if (!Math::isEqual( v1[3], v2[3], eps )) return false;
   return true;
}
```

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 equivelence (with tolerance).

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

Definition at line 650 of file QuatOps.h.

References gmtl::isEqual().

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

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.

```
{
   return(! (v1 == v2));
}
```

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.

```
{
   return (! (tri1 == tri2));
}
```

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.

```
{
   return (! (s1 == s2));
}
```

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==().

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

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.

```
{
   return (! (p1 == p2));
}
```

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.

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

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.

```
{
   return(! (v1 == v2));
}
```

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==().

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

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.

```
{
   return !(v1 == v2);
}
```

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.

```
{
   for(unsigned i=0;i<SIZE;++i)
   {
      if(v1[i] != v2[i])
      {
         return false;
      }
   }

   return true;

   /*  Would like this
   return(vec[0] == _v[0] &&
          vec[1] == _v[1] &&
          vec[2] == _v[2]);
          */
}
```

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.

```
{
   return ( (tri1[0] == tri2[0]) &&
            (tri1[1] == tri2[1]) &&
            (tri1[2] == tri2[2]) );
}
```

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.

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

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.

```
{
   return bool( q1[0] == q2[0] &&
                q1[1] == q2[1] &&
                q1[2] == q2[2] &&
                q1[3] == q2[3]  );
}
```

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.

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

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.

```
{
   for (unsigned int i = 0; i < ROWS*COLS; ++i)
   {
      if (lhs[i] != rhs[i])
      {
         return false;
      }
   }

   return true;

   /*  Would like this
   return( lhs[0] == rhs[0] &&
           lhs[1] == rhs[1] &&
           lhs[2] == rhs[2] );
   */
}
```

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.

```
{
   // @todo metaprogramming.
   if (v1[0] != v2[0]) return false;
   if (v1[1] != v2[1]) return false;
   if (v1[2] != v2[2]) return false;
   return true;
}
```

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.

```
   {
      return bool( q1.getPos() == q2.getPos() &&
                   q1.getRot() == q2.getRot() );
   }
```

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!=().

```
{
   // @todo metaprogramming.
   if (v1[0] != v2[0]) return false;
   if (v1[1] != v2[1]) return false;
   if (v1[2] != v2[2]) return false;
   if (v1[3] != v2[3]) return false;
   return true;
}
```

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> ROTA-TION_TYPE makeRot (const SOURCE_TYPE &coord, Type2Type< ROTA-TION_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::Data-Type, 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 (Axis-Angle< 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 > &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.

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().

```
{
   gmtl::ignore_unused_variable_warning(t);
   TARGET_TYPE target;
   return set( target, src );
}
```

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().

```
{
   gmtl::ignore_unused_variable_warning(t);
   ROTATION_TYPE temporary;
   return setAxes( temporary, xAxis, yAxis, zAxis );
}
```

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().

```
{
   Quat<DATA_TYPE> temporary( quat );
   return conj( temporary );
}
```

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().

```
{
   gmtl::ignore_unused_variable_warning(t);
   ROTATION_TYPE temporary;
   return setDirCos( temporary, xDestAxis, yDestAxis, zDestAxis, xSrcAxis, ySrcAxis, zS
}
```

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().

```
{
   Matrix<DATA_TYPE, ROWS, COLS> result;
   return invert( result, src );
}
```

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().

```
{
   Quat<DATA_TYPE> temporary( quat );
   return invert( temporary );
}
```

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().

```
{
   return AxisAngle<DATA_TYPE>( a.getAngle(), makeNormal( a.getAxis() ) );
}
```

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().

```
{
   Quat<DATA_TYPE> temporary( quat );
   return normalize( temporary );
}
```

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().

```
{
   normalize( vec );
   return vec;
}
```

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.

```
{
   return Quat<DATA_TYPE>( vec[0], vec[1], vec[2], 0 );
}
```

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().

```
{
   ROTATION_TYPE temporary;
   return setRot( temporary, from, to );
}
```

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().

```
{
   gmtl::ignore_unused_variable_warning(t);
   ROTATION_TYPE temporary;
   return set( temporary, coord );
}
```

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().

```
{
   gmtl::ignore_unused_variable_warning(t);
   MATRIX_TYPE temporary;
   return setScale( temporary, scale );
}
```

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().

```
{
   gmtl::ignore_unused_variable_warning(t);
```

```
    MATRIX_TYPE temporary;
    return setScale( temporary, scale );
}
```

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().

```
{
    gmtl::ignore_unused_variable_warning(t);
    TRANS_TYPE temporary;
    return setTrans( temporary, arg );
}
```

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().

```
{
   Matrix<DATA_TYPE, ROWS, COLS> temporary( m );
   return transpose( temporary );
}
```

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:
quat = [v,0] = [v0,v1,v2,0]

Todo:
should this be called convert?

Definition at line 202 of file Generate.h.

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

```
{
   return Vec<DATA_TYPE, 3>( quat[Xelt], quat[Yelt], quat[Zelt] );
}
```

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().

```
{
   const gmtl::Vec3f forward_point(0.0f, 0.0f, -1.0f);   // -Z
   const gmtl::Vec3f origin_point(0.0f, 0.0f, 0.0f);
   gmtl::Vec3f end_point, start_point;

   gmtl::xform(end_point, mat, forward_point);
   gmtl::xform(start_point, mat, origin_point);
   gmtl::Vec3f direction_vector = end_point - start_point;

   // Constrain the direction to YZ-plane only.
   direction_vector[0] = 0.0f;                  // Eliminate X value
   gmtl::normalize(direction_vector);
   float x_rot = gmtl::Math::aCos(gmtl::dot(direction_vector,
                                            forward_point));

   gmtl::Vec3f which_side = gmtl::cross(forward_point, direction_vector);

   // If direction vector to "bottom" (negative) side of forward
   if ( which_side[0] < 0.0f )
   {
      x_rot = -x_rot;
   }

   return x_rot;
}
```

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().

```
{
   const gmtl::Vec3f forward_point(0.0f, 0.0f, -1.0f);   // -Z
   const gmtl::Vec3f origin_point(0.0f, 0.0f, 0.0f);
   gmtl::Vec3f end_point, start_point;

   gmtl::xform(end_point, mat, forward_point);
   gmtl::xform(start_point, mat, origin_point);
   gmtl::Vec3f direction_vector = end_point - start_point;

   // Constrain the direction to XZ-plane only.
   direction_vector[1] = 0.0f;                  // Eliminate Y value
   gmtl::normalize(direction_vector);
   float y_rot = gmtl::Math::aCos(gmtl::dot(direction_vector,
                                            forward_point));

   gmtl::Vec3f which_side = gmtl::cross(forward_point, direction_vector);

   // If direction vector to "right" (negative) side of forward
   if ( which_side[1] < 0.0f )
   {
      y_rot = -y_rot;
   }

   return y_rot;
}
```

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().

```
{
   const gmtl::Vec3f forward_point(0.0f, 0.0f, -1.0f);   // -Z
   const gmtl::Vec3f origin_point(0.0f, 0.0f, 0.0f);
   gmtl::Vec3f end_point, start_point;

   gmtl::xform(end_point, mat, forward_point);
```

```
    gmtl::xform(start_point, mat, origin_point);
    gmtl::Vec3f direction_vector = end_point - start_point;

    // Constrain the direction to XY-plane only.
    direction_vector[2] = 0.0f;                  // Eliminate Z value
    gmtl::normalize(direction_vector);
    float z_rot = gmtl::Math::aCos(gmtl::dot(direction_vector,
                                             forward_point));

    gmtl::Vec3f which_side = gmtl::cross(forward_point, direction_vector);

    // If direction vector to "right" (negative) side of forward
    if ( which_side[2] < 0.0f )
    {
       z_rot = -z_rot;
    }

    return z_rot;
 }
```

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().

```
 {
    setRot( mat, q );

    if (ROWS == 4)
    {
       mat( 3, 0 ) = DATA_TYPE(0.0);
       mat( 3, 1 ) = DATA_TYPE(0.0);
       mat( 3, 2 ) = DATA_TYPE(0.0);
```

```
    }

    if (COLS == 4)
    {
       mat( 0, 3 ) = DATA_TYPE(0.0);
       mat( 1, 3 ) = DATA_TYPE(0.0);
       mat( 2, 3 ) = DATA_TYPE(0.0);
    }

    if (ROWS == 4 && COLS == 4)
       mat( 3, 3 ) = DATA_TYPE(1.0);

    return mat;
 }
```

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().

```
 {
    // set to ident first...
    gmtl::identity( mat );

    // set translation
    // @todo metaprogram this out for 3x3 and 2x2 matrices
    if (MATCOLS == 4)
    {
       setTrans( mat, coord.getPos() );// filtered (only sets the trans part).
    }
    setRot( mat, coord.getRot() ); // filtered (only sets the rot part).
    return mat;
 }
```

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().

```
{
   gmtl::identity( result );
   return setRot( result, euler );
}
```

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().

```
{
   gmtl::identity( result );
   return setRot( result, axisAngle );
}
```

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 gmtlASSERT, gmtl::Math::sqrt(), gmtl::Welt, gmtl::Xelt, gmtl::Yelt, and gmtl::Zelt.

```
   {
      gmtlASSERT( ((ROWS == 3 && COLS == 3) ||
                   (ROWS == 3 && COLS == 4) ||
                   (ROWS == 4 && COLS == 3) ||
                   (ROWS == 4 && COLS == 4)) &&
                   "pre conditions not met on set( quat, pos, mat ) which only sets a quaternio

      DATA_TYPE tr( mat( 0, 0 ) + mat( 1, 1 ) + mat( 2, 2 ) ), s( 0.0f );

      // If diagonal is positive
      if (tr > (DATA_TYPE)0.0)
      {
         s = Math::sqrt( tr + (DATA_TYPE)1.0 );
         quat[Welt] = s * (DATA_TYPE)0.5;
         s = DATA_TYPE(0.5) / s;

         quat[Xelt] = (mat( 2, 1 ) - mat( 1, 2 )) * s;
         quat[Yelt] = (mat( 0, 2 ) - mat( 2, 0 )) * s;
         quat[Zelt] = (mat( 1, 0 ) - mat( 0, 1 )) * s;
      }

      // when Diagonal is negative
      else
      {
         static const unsigned int nxt[3] = { 1, 2, 0 };
         unsigned int i( Xelt ), j, k;

         if (mat( 1, 1 ) > mat( 0, 0 ))
            i = 1;

         if (mat( 2, 2 ) > mat( i, i ))
            i = 2;

         j = nxt[i];
         k = nxt[j];

         s = Math::sqrt( (mat( i, i )-(mat( j, j )+mat( k, k ))) + (DATA_TYPE)1.0 );

         DATA_TYPE q[4];
         q[i] = s * (DATA_TYPE)0.5;
```

```

      if (s != (DATA_TYPE)0.0)
         s = DATA_TYPE(0.5) / s;

      q[3] = (mat( k, j ) - mat( j, k )) * s;
      q[j] = (mat( j, i ) + mat( i, j )) * s;
      q[k] = (mat( k, i ) + mat( i, k )) * s;

      quat[Xelt] = q[0];
      quat[Yelt] = q[1];
      quat[Zelt] = q[2];
      quat[Welt] = q[3];
   }

   return quat;
}
```

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(), gmtlASSERT, gmtl::normalize(), gmtl::Math::sin(), gmtl::Welt, gmtl::Xelt, gmtl::Yelt, and gmtl::Zelt.

```
{
   // this might be faster if put into the switch statement... (testme)
   const int& order = ROT_ORDER::ID;
   const DATA_TYPE xRot = (order == XYZ::ID) ? euler[0] : ((order == ZXY::ID) ? euler[1] : euler[2]
   const DATA_TYPE yRot = (order == XYZ::ID) ? euler[1] : ((order == ZXY::ID) ? euler[2] : euler[1]
   const DATA_TYPE zRot = (order == XYZ::ID) ? euler[2] : ((order == ZXY::ID) ? euler[0] : euler[0]

   // this could be written better for each rotation order, but this is really general...
   Quat<DATA_TYPE> qx, qy, qz;

   // precompute half angles
   DATA_TYPE xOver2 = xRot * (DATA_TYPE)0.5;
   DATA_TYPE yOver2 = yRot * (DATA_TYPE)0.5;
   DATA_TYPE zOver2 = zRot * (DATA_TYPE)0.5;

   // set the pitch quat
```

```
   qx[Xelt] = Math::sin( xOver2 );
   qx[Yelt] = (DATA_TYPE)0.0;
   qx[Zelt] = (DATA_TYPE)0.0;
   qx[Welt] = Math::cos( xOver2 );

   // set the yaw quat
   qy[Xelt] = (DATA_TYPE)0.0;
   qy[Yelt] = Math::sin( yOver2 );
   qy[Zelt] = (DATA_TYPE)0.0;
   qy[Welt] = Math::cos( yOver2 );

   // set the roll quat
   qz[Xelt] = (DATA_TYPE)0.0;
   qz[Yelt] = (DATA_TYPE)0.0;
   qz[Zelt] = Math::sin( zOver2 );
   qz[Welt] = Math::cos( zOver2 );

   // compose the three in pyr order...
   switch (order)
   {
   case XYZ::ID: result = qx * qy * qz; break;
   case ZYX::ID: result = qz * qy * qx; break;
   case ZXY::ID: result = qz * qx * qy; break;
   default:
      gmtlASSERT( false && "unknown rotation order passed to setRot" );
      break;
   }

   // ensure the quaternion is normalized
   normalize( result );
   return result;
}
```

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(rad/2), sin(rad/2) $*$ [x,y,z]]

Definition at line 319 of file Generate.h.

References gmtl::Math::cos(), gmtlASSERT, gmtl::lengthSquared(), gmtl::Math::sin(), gmtl::Welt, gmtl::Xelt, gmtl::Yelt, and gmtl::Zelt.

```
{
   gmtlASSERT( (Math::isEqual( lengthSquared( axisAngle.getAxis() ), (DATA_TYPE)1.0, (DATA_TYPE)0.0
                "you must pass in a normalized vector to setRot( quat, rad, vec )" );

   DATA_TYPE half_angle = axisAngle.getAngle() * (DATA_TYPE)0.5;
   DATA_TYPE sin_half_angle = Math::sin( half_angle );

   result[Welt] = Math::cos( half_angle );
   result[Xelt] = sin_half_angle * axisAngle.getAxis()[0];
   result[Yelt] = sin_half_angle * axisAngle.getAxis()[1];
   result[Zelt] = sin_half_angle * axisAngle.getAxis()[2];

   // should automagically be normalized (unit magnitude) now...
   return result;
}
```

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().

```
{
   gmtl::setTrans( eulercoord.pos(), mat );
   gmtl::set( eulercoord.rot(), mat );
   return eulercoord;
}
```

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().

```
{
   // @todo set this a compile time assert...
   gmtlASSERT( ROWS >= 3 && COLS >= 3 && ROWS <= 4 && COLS <= 4 &&
         "this is undefined for Matrix smaller than 3x3 or bigger than 4x4" );

   DATA_TYPE sx;
   DATA_TYPE cz;

   // @todo metaprogram this!
   const int& order = ROT_ORDER::ID;
   switch (order)
   {
   case XYZ::ID:
      {
         euler[2] = Math::aTan2( -mat(0,1), mat(0,0) );       // -(-cy*sz)/(cy*cz) - cy
         euler[0] = Math::aTan2( -mat(1,2), mat(2,2) );       // -(sx*cy)/(cx*cy) - cy f
         cz = Math::cos( euler[2] );
         euler[1] = Math::aTan2( mat(0,2), mat(0,0) / cz );   // (sy)/((cy*cz)/cz)
      }
      break;
   case ZYX::ID:
      {
         euler[0] = Math::aTan2( mat(1,0), mat(0,0) );        // (cy*sz)/(cy*cz) - cy fa
         euler[2] = Math::aTan2( mat(2,1), mat(2,2) );        // (sx*cy)/(cx*cy) - cy fa
         sx = Math::sin( euler[2] );
         euler[1] = Math::aTan2( -mat(2,0), mat(2,1) / sx );  // -(-sy)/((sx*cy)/sx)
      }
      break;
   case ZXY::ID:
      {
         // Extract the rotation directly from the matrix
         DATA_TYPE x_angle;
         DATA_TYPE y_angle;
```

```
         DATA_TYPE z_angle;
         DATA_TYPE cos_y, sin_y;
         DATA_TYPE cos_x, sin_x;
         DATA_TYPE cos_z, sin_z;

         sin_x = mat(2,1);
         x_angle = Math::aSin( sin_x );     // Get x angle
         cos_x = Math::cos( x_angle );

         // Check if cos_x = Zero
         if (cos_x != 0.0f)     // ASSERT: cos_x != 0
         {
               // Get y Angle
            cos_y = mat(2,2) / cos_x;
            sin_y = -mat(2,0) / cos_x;
            y_angle = Math::aTan2( cos_y, sin_y );

               // Get z Angle
            cos_z = mat(1,1) / cos_x;
            sin_z = -mat(0,1) / cos_x;
            z_angle = Math::aTan2( cos_z, sin_z );
         }
         else
         {
            // Arbitrarily set z_angle = 0
            z_angle = 0;

               // Get y Angle
            cos_y = mat(0,0);
            sin_y = mat(1,0);
            y_angle = Math::aTan2( cos_y, sin_y );
         }

         euler[1] = x_angle;
         euler[2] = y_angle;
         euler[0] = z_angle;
      }
      break;
   default:
      gmtlASSERT( false && "unknown rotation order passed to setRot" );
      break;
   }
   return euler;
}
```

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 equivilent to the quaternion's rotation.

returns rad and xyz such that

- rad = acos(w) ∗ 2.0
- vec = v / (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.

```
{
   // set sure we don't get a NaN result from acos...
   if (Math::abs( quat[Welt] ) > (DATA_TYPE)1.0)
   {
      gmtl::normalize( quat );
   }
   gmtlASSERT( Math::abs( quat[Welt] ) <= (DATA_TYPE)1.0 && "acos returns NaN when quat

   // [acos( w ) * 2.0, v / (asin( w ) * 2.0)]

   // set the angle - aCos is mathematically defined to be between 0 and PI
   DATA_TYPE rad = Math::aCos( quat[Welt] ) * (DATA_TYPE)2.0;
   axisAngle.setAngle( rad );

   // set the axis: (use sin(rad) instead of asin(w))
   DATA_TYPE sin_half_angle = Math::sin( rad * (DATA_TYPE)0.5 );
   if (sin_half_angle >= (DATA_TYPE)0.0001) // because (PI >= rad >= 0)
   {
      DATA_TYPE sin_half_angle_inv = DATA_TYPE(1.0) / sin_half_angle;
      axisAngle.setAxis( gmtl::Vec3f(
                         quat[Xelt] * sin_half_angle_inv,
                         quat[Yelt] * sin_half_angle_inv,
                         quat[Zelt] * sin_half_angle_inv ) );
   }

   // avoid NAN
   else
   {
      // one of the terms should be a 1,
      // so we can maintain unit-ness
      // in case w is 0 (which here w is 0)
      axisAngle.setAxis( gmtl::Vec3f(
                          DATA_TYPE( 1.0 ) /*- gmtl::Math::abs( quat[Welt] )*/,
                         (DATA_TYPE)0.0,
                         (DATA_TYPE)0.0 ) );
   }
   return axisAngle;
}
```

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().

```
{
   // @todo set this a compile time assert...
   gmtlASSERT( ROWS >= 3 && COLS >= 3 && ROWS <= 4 && COLS <= 4 && "this is undefined for Matrix sm

   result( 0, 0 ) = xAxis[0];
   result( 1, 0 ) = xAxis[1];
   result( 2, 0 ) = xAxis[2];

   result( 0, 1 ) = yAxis[0];
   result( 1, 1 ) = yAxis[1];
   result( 2, 1 ) = yAxis[2];

   result( 0, 2 ) = zAxis[0];
   result( 1, 2 ) = zAxis[1];
   result( 2, 2 ) = zAxis[2];

   return result;
}
```

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 xSrcAxis(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 gmtlASSERT.

Referenced by gmtl::makeDirCos().

```
{
   // @todo set this a compile time assert...
   gmtlASSERT( ROWS >= 3 && COLS >= 3 && ROWS <= 4 && COLS <= 4 && "this is undefined fo

   DATA_TYPE Xa, Xb, Xy;    // Direction cosines of the secondary x-axis
   DATA_TYPE Ya, Yb, Yy;    // Direction cosines of the secondary y-axis
   DATA_TYPE Za, Zb, Zy;    // Direction cosines of the secondary z-axis

   Xa = dot(xDestAxis, xSrcAxis);  Xb = dot(xDestAxis, ySrcAxis);  Xy = dot(xDestAxis, z
   Ya = dot(yDestAxis, xSrcAxis);  Yb = dot(yDestAxis, ySrcAxis);  Yy = dot(yDestAxis, z
   Za = dot(zDestAxis, xSrcAxis);  Zb = dot(zDestAxis, ySrcAxis);  Zy = dot(zDestAxis, z

   // Set the matrix correctly
   result( 0, 0 ) = Xa; result( 0, 1 ) = Xb; result( 0, 2 ) = Xy;
   result( 1, 0 ) = Ya; result( 1, 1 ) = Yb; result( 1, 2 ) = Yy;
   result( 2, 0 ) = Za; result( 2, 1 ) = Zb; result( 2, 2 ) = Zy;

   return result;
}
```

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.

```
{
   quat.set( vec[0], vec[1], vec[2], 0 );
   return quat;
}
```

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().

```
{
   return set( result, mat );
}
```

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.

```
{
   gmtlASSERT( ((ROWS == 3 && COLS == 3) ||
            (ROWS == 3 && COLS == 4) ||
            (ROWS == 4 && COLS == 3) ||
            (ROWS == 4 && COLS == 4)) &&
            "pre conditions not met on set( mat, quat ) which only sets a quaternion to

   // From Watt & Watt
   DATA_TYPE wx, wy, wz, xx, yy, yz, xy, xz, zz, xs, ys, zs;

   xs = q[Xelt] + q[Xelt]; ys = q[Yelt] + q[Yelt]; zs = q[Zelt] + q[Zelt];
   xx = q[Xelt] * xs;      xy = q[Xelt] * ys;      xz = q[Xelt] * zs;
   yy = q[Yelt] * ys;      yz = q[Yelt] * zs;      zz = q[Zelt] * zs;
   wx = q[Welt] * xs;      wy = q[Welt] * ys;      wz = q[Welt] * zs;

   mat( 0, 0 ) = DATA_TYPE(1.0) - (yy + zz);
   mat( 1, 0 ) = xy + wz;
   mat( 2, 0 ) = xz - wy;

   mat( 0, 1 ) = xy - wz;
   mat( 1, 1 ) = DATA_TYPE(1.0) - (xx + zz);
   mat( 2, 1 ) = yz + wx;

   mat( 0, 2 ) = xz + wy;
   mat( 1, 2 ) = yz - wx;
   mat( 2, 2 ) = DATA_TYPE(1.0) - (xx + yy);

   return mat;
}
```

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().

```
{
   // @todo set this a compile time assert...
   gmtlASSERT( ROWS >= 3 && COLS >= 3 && ROWS <= 4 && COLS <= 4 && "this is undefined for Matrix sm

   // this might be faster if put into the switch statement... (testme)
   const int& order = ROT_ORDER::ID;
   const float xRot = (order == XYZ::ID) ? euler[0] : ((order == ZXY::ID) ? euler[1] : euler[2]);
   const float yRot = (order == XYZ::ID) ? euler[1] : ((order == ZXY::ID) ? euler[2] : euler[1]);
   const float zRot = (order == XYZ::ID) ? euler[2] : ((order == ZXY::ID) ? euler[0] : euler[0]);

   float sx = Math::sin( xRot );  float cx = Math::cos( xRot );
   float sy = Math::sin( yRot );  float cy = Math::cos( yRot );
   float sz = Math::sin( zRot );  float cz = Math::cos( zRot );

   // @todo metaprogram this!
   switch (order)
   {
   case XYZ::ID:
      // Derived by simply multiplying out the matrices by hand X * Y * Z
      result( 0, 0 ) = cy*cz;             result( 0, 1 ) = -cy*sz;            result( 0, 2 ) = sy;
      result( 1, 0 ) = sx*sy*cz + cx*sz;  result( 1, 1 ) = -sx*sy*sz + cx*cz; result( 1, 2 ) = -sx*
      result( 2, 0 ) = -cx*sy*cz + sx*sz; result( 2, 1 ) = cx*sy*sz + sx*cz;  result( 2, 2 ) = cx*c
      break;
   case ZYX::ID:
      // Derived by simply multiplying out the matrices by hand Z * Y * Z
      result( 0, 0 ) = cy*cz; result( 0, 1 ) = -cx*sz + sx*sy*cz; result( 0, 2 ) = sx*sz + cx*sy*cz
      result( 1, 0 ) = cy*sz; result( 1, 1 ) = cx*cz + sx*sy*sz;  result( 1, 2 ) = -sx*cz + cx*sy*s
      result( 2, 0 ) = -sy;   result( 2, 1 ) = sx*cy;             result( 2, 2 ) = cx*cy;
      break;
   case ZXY::ID:
      // Derived by simply multiplying out the matrices by hand Z * X * Y
      result( 0, 0 ) = cy*cz - sx*sy*sz; result( 0, 1 ) = -cx*sz; result( 0, 2 ) = sy*cz + sx*cy*sz
      result( 1, 0 ) = cy*sz + sx*sy*cz; result( 1, 1 ) = cx*cz;  result( 1, 2 ) = sy*sz - sx*cy*cz
      result( 2, 0 ) = -cx*sy;           result( 2, 1 ) = sx;     result( 2, 2 ) = cx*cy;
      break;
   default:
      gmtlASSERT( false && "unknown rotation order passed to setRot" );
      break;
   }

   return result;
}
```

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().

```
{
   /* @todo set this a compile time assert... */
   gmtlASSERT( ROWS >= 3 && COLS >= 3 && ROWS <= 4 && COLS <= 4 &&
                  "this func is undefined for Matrix smaller than 3x3 or bigger than 4x4
   gmtlASSERT( Math::isEqual( lengthSquared( axisAngle.getAxis() ), (DATA_TYPE)1.0, (DAT
                  "you must pass in a normalized vector to setRot( mat, rad, vec )" );

   // GGI: pg 466
   DATA_TYPE s = Math::sin( axisAngle.getAngle() );
   DATA_TYPE c = Math::cos( axisAngle.getAngle() );
   DATA_TYPE t = DATA_TYPE( 1.0 ) - c;
   DATA_TYPE x = axisAngle.getAxis()[0];
   DATA_TYPE y = axisAngle.getAxis()[1];
   DATA_TYPE z = axisAngle.getAxis()[2];

   /* From: Introduction to robotic.  Craig.  Pg. 52 */
   result( 0, 0 ) = (t*x*x)+c;     result( 0, 1 ) = (t*x*y)-(s*z); result( 0, 2 ) = (t*x
   result( 1, 0 ) = (t*x*y)+(s*z); result( 1, 1 ) = (t*y*y)+c;     result( 1, 2 ) = (t*y
   result( 2, 0 ) = (t*x*z)-(s*y); result( 2, 1 ) = (t*y*z)+(s*x); result( 2, 2 ) = (t*z

   return result;
}
```

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().

```
{
   return set( result, mat );
}
```

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().

```
{
   return set( result, euler );
}
```

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().

```
{
   return set( result, axisAngle );
}
```

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(), gmtlASSERT, gmtl::isEqual(), and gmtl::normalize().

Referenced by gmtl::makeRot(), and gmtl::set().

```
{
   // @todo should assert that DEST_TYPE::DataType == DATA_TYPE
   const DATA_TYPE epsilon = (DATA_TYPE)0.00001;

   gmtlASSERT( gmtl::Math::isEqual( gmtl::length( from ), (DATA_TYPE)1.0, epsilon ) &&
               gmtl::Math::isEqual( gmtl::length( to ), (DATA_TYPE)1.0, epsilon ) &&
               "input params not normalized" );

   DATA_TYPE cosangle = dot( from, to );

   // if cosangle is close to 1, so the vectors are close to being coincident
   // Need to generate an angle of zero with any vector we like
   // We'll choose identity (no rotation)
   if ( Math::isEqual( cosangle, (DATA_TYPE)1.0, epsilon ) )
   {
      return result = DEST_TYPE();
   }

   // vectors are close to being opposite, so rotate one a little...
   else if ( Math::isEqual( cosangle, (DATA_TYPE)-1.0, epsilon ) )
   {
      Vec<DATA_TYPE, 3> to_rot( to[0] + (DATA_TYPE)0.3, to[1] - (DATA_TYPE)0.15, to[2] - (DATA_TYPE
      normalize( cross( axis, from, to_rot ) ); // setRot requires normalized vec
      DATA_TYPE angle = Math::aCos( cosangle );
      return setRot( result, gmtl::AxisAngle<DATA_TYPE>( angle, axis ) );
   }

   // This is the usual situation - take a cross-product of vec1 and vec2
   // and that is the axis around which to rotate.
   else
   {
```

```
            Vec<DATA_TYPE, 3> axis;
            normalize( cross( axis, from, to ) ); // setRot requires normalized vec
            DATA_TYPE angle = Math::aCos( cosangle );
            return setRot( result, gmtl::AxisAngle<DATA_TYPE>( angle, axis ) );
         }
      }
```

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().

```
      {
         for (unsigned x = 0; x < Math::Min( ROWS, COLS, Math::Max( ROWS, COLS ) - 1 ); ++x) /
            result( x, x ) = scale;
         return result;
      }
```

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().

```
      {
         gmtlASSERT( ((SIZE == (ROWS-1) && SIZE == (COLS-1)) || (SIZE == (ROWS-1) && SIZE == (
         for (unsigned x = 0; x < SIZE; ++x)
            result( x, x ) = scale[x];
         return result;
      }
```

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 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 226 of file Generate.h.

References gmtlASSERT.

Referenced by gmtl::makeTrans(), and gmtl::set().

```
{
   // ASSERT: There are as many

   // if n x n   then (homogeneous case) vecsize == rows-1 or (scale component case) vecsize == row
   // if n x n+1 then (homogeneous case) vecsize == rows   or (scale component case) vecsize == row
   gmtlASSERT( ((ROWS == COLS && ( VEC_TYPE::Size == (ROWS-1) ||  VEC_TYPE::Size == ROWS)) ||
            (COLS == (ROWS+1) && ( VEC_TYPE::Size == ROWS ||  VEC_TYPE::Size == (ROWS+1)))) &&
           "preconditions not met for vector size in call to makeTrans.  Read your documentation."

   // homogeneous case...
   if ((ROWS == COLS &&  VEC_TYPE::Size == ROWS)              // Square matrix and vec so assume ho
       || (COLS == (ROWS+1) &&  VEC_TYPE::Size == (ROWS+1)))  // ex: 3x4 with vec4
   {
      result[VEC_TYPE::Size-1] = 1.0f;
   }

   // non-homogeneous case... (SIZE == ROWS),
   //else
   //{}

   for (unsigned x = 0; x < COLS - 1; ++x)
   {
      result[x] = arg( x, COLS - 1 );
```

```
   }

   return result;
}
```

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.

```
{
   /* @todo make this a compile time assert... */
   // if n x n   then (homogeneous case) vecsize == rows-1 or (scale component case) vec
   // if n x n+1 then (homogeneous case) vecsize == rows   or (scale component case) vec
   gmtlASSERT( ((ROWS == COLS && (SIZE == (ROWS-1) || SIZE == ROWS)) ||
             (COLS == (ROWS+1) && (SIZE == ROWS || SIZE == (ROWS+1)))) &&
            "preconditions not met for vector size in call to makeTrans.  Read your docum

   // homogeneous case...
   if ((ROWS == COLS && SIZE == ROWS) /* Square matrix and vec so assume homogeneous vec
       || (COLS == (ROWS+1) && SIZE == (ROWS+1)))  /* ex: 3x4 with vec4 */
   {
      for (unsigned x = 0; x < COLS - 1; ++x)
         result( x, COLS - 1 ) = trans[x] / trans[SIZE-1];
   }

```

```
      // non-homogeneous case...
      else
      {
         for (unsigned x = 0; x < COLS - 1; ++x)
            result( x, COLS - 1 ) = trans[x];
      }
      return result;
   }
```

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 gmtl::dot(), gmtl::Welt, gmtl::Xelt, gmtl::Yelt, and gmtl::Zelt.

```
{
   // just an alias to match q
   const Quat<DATA_TYPE>& p = from;

   // calc cosine theta
   DATA_TYPE cosom = dot( from, to );

   // adjust signs (if necessary)
   Quat<DATA_TYPE> q;
   if (cosom < (DATA_TYPE)0.0)
   {
      q[0] = -to[0];   // Reverse all signs
      q[1] = -to[1];
      q[2] = -to[2];
      q[3] = -to[3];
   }
   else
   {
      q = to;
   }

   // do linear interp
   DATA_TYPE sclp, sclq;
   sclp = (DATA_TYPE)1.0 - t;
   sclq = t;

```

```
      result[Xelt] = sclp * p[Xelt] + sclq * q[Xelt];
      result[Yelt] = sclp * p[Yelt] + sclq * q[Yelt];
      result[Zelt] = sclp * p[Zelt] + sclq * q[Zelt];
      result[Welt] = sclp * p[Welt] + sclq * q[Welt];
      return result;
   }
```

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.

```
{
   for (unsigned int x = 0; x < SIZE; ++x)
   {
      Math::lerp( result[x], lerpVal, from[x], to[x] );
   }
   return result;
}
```

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().

```
{
    T size = b - a;
    result = ((U)a) + (((U)size) * lerp);
}
```

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.

```
{
   const Quat<DATA_TYPE>& p = from; // just an alias to match q

   // calc cosine theta
   DATA_TYPE cosom = dot( from, to );
```

```

      // adjust signs (if necessary)
      Quat<DATA_TYPE> q;
      if (cosom < (DATA_TYPE)0.0)
      {
         cosom = -cosom;
         q[0] = -to[0];   // Reverse all signs
         q[1] = -to[1];
         q[2] = -to[2];
         q[3] = -to[3];
      }
      else
      {
         q = to;
      }

      // Calculate coefficients
      DATA_TYPE sclp, sclq;
      if (((DATA_TYPE)1.0 - cosom) > (DATA_TYPE)0.0001) // 0.0001 -> some epsillon
      {
         // Standard case (slerp)
         DATA_TYPE omega, sinom;
         omega = gmtl::Math::aCos( cosom ); // extract theta from dot product's cos theta
         sinom = gmtl::Math::sin( omega );
         sclp  = gmtl::Math::sin( ((DATA_TYPE)1.0 - t) * omega ) / sinom;
         sclq  = gmtl::Math::sin( t * omega ) / sinom;
      }
      else
      {
         // Very close, do linear interp (because it's faster)
         sclp = (DATA_TYPE)1.0 - t;
         sclq = t;
      }

      result[Xelt] = sclp * p[Xelt] + sclq * q[Xelt];
      result[Yelt] = sclp * p[Yelt] + sclq * q[Yelt];
      result[Zelt] = sclp * p[Zelt] + sclq * q[Zelt];
      result[Welt] = sclp * p[Welt] + sclq * q[Welt];
      return result;
   }
```

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.

```
{
    out << s.mCenter << ", " << s.mRadius;
    return out;
}
```

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.

```
{
   out << q.mData;
   return out;
}
```

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.

```
{
   for ( unsigned row=0; row<ROWS; ++row )
    {
       out << "|";
       for ( unsigned col=0; col<COLS; ++col )
       {
          out << " " << m(row, col);
       }
       out << " |" << std::endl;
    }
    return out;
}
```

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.

```
{
   out << "(";
   for ( unsigned i=0; i<SIZE; ++i )
   {
      if ( i != 0 )
      {
         out << ", ";
      }
      out << v[i];
   }
   out << ")";
   return out;
}
```

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, 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 > &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 equivelence (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, EulerAngleZYXd > CoordVec3EulerAngleZYXd
- typedef Coord< Vec3f, EulerAngleZYXf > CoordVec3EulerAngleZYXf
- typedef Coord< Vec4d, EulerAngleZYXd > CoordVec4EulerAngleZYXd
- typedef Coord< Vec4f, EulerAngleZYXf > CoordVec4EulerAngleZYXf
- 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 of 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 gmtl::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 the 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().

```
{
   return ( pt - findNearestPt( lineseg, pt ) );
}
```

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.

```
   {
      if(radii0 > rCenterSep)
      {
         if( radiiT > rCenterSep)
            return false;
         if(FIND_CONTACT)
         {
            float tmp(dt*((rCenterSep-radii0)/(radiiT-radii0)));
            if(tmp > tMaxContact)
            {
               tMaxContact = tmp;
            }
         }
      }
      else if( radii0 < -rCenterSep)
      {
         if( radiiT < -rCenterSep)
            return false;
         if(FIND_CONTACT)
         {
            float tmp(-dt*((rCenterSep+radii0)/(radiiT-radii0)));
            if(tmp > tMaxContact)
            {
               tMaxContact = tmp;
            }
         }
      }

      return true;
   }
```

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.

```
{
   float radiiT;

   if(radii0 > rCenterSep)
   {
      radiiT = (rt0*rt1)-(rt2*rt3);
      if( radiiT > rCenterSep)
         return false;
      if(FIND_CONTACT)
      {
         float tmp(dt*((rCenterSep-radii0)/(radiiT-radii0)));
         if(tmp > tMaxContact)
         {
            tMaxContact = tmp;
         }
      }
   }
   else if( radii0 < -rCenterSep)
   {
      radiiT = (rt0*rt1)-(rt2*rt3);
      if( radiiT < -rCenterSep)
         return false;
      if(FIND_CONTACT)
      {
         float tmp(-dt*((rCenterSep+radii0)/(radiiT-radii0)));
         if(tmp > tMaxContact)
         {
            tMaxContact = tmp;
         }
      }
   }

   return true;
}
```

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().

```
{
   // check if we already contain the sphere
   if ( isInVolume( container, sphere ) )
   {
      return;
   }

```

```
   // make a vector pointing from the center of container to sphere. this is the
   // direction in which we need to move container's center
   Vec<DATA_TYPE, 3> dir = sphere.mCenter - container.mCenter;
   DATA_TYPE len = normalize( dir );

   // compute what the new radius should be
   DATA_TYPE newRadius = (len + sphere.mRadius + container.mRadius) *
                         DATA_TYPE(0.5);

   // compute the new center for container
   Point<DATA_TYPE, 3> newCenter = container.mCenter +
                                   (dir * (newRadius - container.mRadius));

   // modify container to its new values
   container.mCenter = newCenter;
   container.mRadius = newRadius;
}
```

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().

```
{
   // check if we already contain the point
   if ( isInVolume( container, pt ) )
   {
      return;
   }

   // make a vector pointing from the center of the sphere to pt. this is the
   // direction in which we need to move the sphere's center
   Vec<DATA_TYPE, 3> dir = pt - container.mCenter;
   DATA_TYPE len = normalize( dir );

   // compute what the new radius should be
   DATA_TYPE newRadius =  (len + container.mRadius) * DATA_TYPE(0.5);

   // compute the new center for the sphere
```

```
   Point<DATA_TYPE, 3> newCenter = container.mCenter +
                                   (dir * (newRadius - container.mRadius));

   // modify container to its new values
   container.mCenter = newCenter;
   container.mRadius = newRadius;
}
```

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().

```
{
  // result = origin + dir * dot((pt-origin), dir)
  return ( lineseg.mOrigin + lineseg.mDir *
           dot(pt - lineseg.mOrigin, lineseg.mDir) );
}
```

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.

```
{
    // compute mean of points
    rkCenter = ZeroVec3;
    int i, iValidQuantity = 0;
    for (i = 0; i < iQuantity; i++)
    {
        if ( abValid[i] )
        {
            rkCenter += akPoint[i];
            iValidQuantity++;
        }
    }
    if ( iValidQuantity == 0 )
        return false;

    float fInvQuantity = 1.0/iValidQuantity;
    rkCenter *= fInvQuantity;

    // compute covariances of points
    float fSumXX = 0.0, fSumXY = 0.0, fSumXZ = 0.0;
    float fSumYY = 0.0, fSumYZ = 0.0, fSumZZ = 0.0;
    for (i = 0; i < iQuantity; i++)
    {
        if ( abValid[i] )
        {
            Vec3 kDiff = akPoint[i] - rkCenter;
            fSumXX += kDiff[Xelt]*kDiff[Xelt];
            fSumXY += kDiff[Xelt]*kDiff[Yelt];
            fSumXZ += kDiff[Xelt]*kDiff[Zelt];
            fSumYY += kDiff[Yelt]*kDiff[Yelt];
            fSumYZ += kDiff[Yelt]*kDiff[Zelt];
            fSumZZ += kDiff[Zelt]*kDiff[Zelt];
        }
    }
    fSumXX *= fInvQuantity;
    fSumXY *= fInvQuantity;
    fSumXZ *= fInvQuantity;
    fSumYY *= fInvQuantity;
    fSumYZ *= fInvQuantity;
    fSumZZ *= fInvQuantity;

    // compute eigenvectors for covariance matrix
    Eigen kES(3);
    kES.Matrix(0,0) = fSumXX;
    kES.Matrix(0,1) = fSumXY;
    kES.Matrix(0,2) = fSumXZ;
    kES.Matrix(1,0) = fSumXY;
    kES.Matrix(1,1) = fSumYY;
    kES.Matrix(1,2) = fSumYZ;
    kES.Matrix(2,0) = fSumXZ;
    kES.Matrix(2,1) = fSumYZ;
    kES.Matrix(2,2) = fSumZZ;
    kES.IncrSortEigenStuff3();

    akAxis[0][Xelt] = kES.GetEigenvector(0,0);
```

```
    akAxis[0][Yelt] = kES.GetEigenvector(1,0);
    akAxis[0][Zelt] = kES.GetEigenvector(2,0);

    akAxis[1][Xelt] = kES.GetEigenvector(0,1);
    akAxis[1][Yelt] = kES.GetEigenvector(1,1);
    akAxis[1][Zelt] = kES.GetEigenvector(2,1);

    akAxis[2][Xelt] = kES.GetEigenvector(0,2);
    akAxis[2][Yelt] = kES.GetEigenvector(1,2);
    akAxis[2][Zelt] = kES.GetEigenvector(2,2);

    afExtent[0] = kES.GetEigenvalue(0);
    afExtent[1] = kES.GetEigenvalue(1);
    afExtent[2] = kES.GetEigenvalue(2);

    return true;
}
```

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.

```
{
    // compute mean of points
    rkCenter = akPoint[0];
    unsigned i;
    for (i = 1; i < iQuantity; i++)
        rkCenter += akPoint[i];
    float fInvQuantity = 1.0f/iQuantity;
    rkCenter *= fInvQuantity;

    // compute covariances of points
    float fSumXX = 0.0, fSumXY = 0.0, fSumXZ = 0.0;
    float fSumYY = 0.0, fSumYZ = 0.0, fSumZZ = 0.0;
    for (i = 0; i < iQuantity; i++)
    {
        Vec3 kDiff = akPoint[i] - rkCenter;
        fSumXX += kDiff[Xelt]*kDiff[Xelt];
        fSumXY += kDiff[Xelt]*kDiff[Yelt];
        fSumXZ += kDiff[Xelt]*kDiff[Zelt];
        fSumYY += kDiff[Yelt]*kDiff[Yelt];
        fSumYZ += kDiff[Yelt]*kDiff[Zelt];
        fSumZZ += kDiff[Zelt]*kDiff[Zelt];
    }
    fSumXX *= fInvQuantity;
    fSumXY *= fInvQuantity;
```

```
    fSumXZ *= fInvQuantity;
    fSumYY *= fInvQuantity;
    fSumYZ *= fInvQuantity;
    fSumZZ *= fInvQuantity;

    // compute eigenvectors for covariance matrix
    gmtl::Eigen kES(3);
    kES.Matrix(0,0) = fSumXX;
    kES.Matrix(0,1) = fSumXY;
    kES.Matrix(0,2) = fSumXZ;
    kES.Matrix(1,0) = fSumXY;
    kES.Matrix(1,1) = fSumYY;
    kES.Matrix(1,2) = fSumYZ;
    kES.Matrix(2,0) = fSumXZ;
    kES.Matrix(2,1) = fSumYZ;
    kES.Matrix(2,2) = fSumZZ;
    kES.IncrSortEigenStuff3();

    akAxis[0][Xelt] = kES.GetEigenvector(0,0);
    akAxis[0][Yelt] = kES.GetEigenvector(1,0);
    akAxis[0][Zelt] = kES.GetEigenvector(2,0);

    akAxis[1][Xelt] = kES.GetEigenvector(0,1);
    akAxis[1][Yelt] = kES.GetEigenvector(1,1);
    akAxis[1][Zelt] = kES.GetEigenvector(2,1);

    akAxis[2][Xelt] = kES.GetEigenvector(0,2);
    akAxis[2][Yelt] = kES.GetEigenvector(1,2);
    akAxis[2][Zelt] = kES.GetEigenvector(2,2);

    afExtent[0] = kES.GetEigenvalue(0);
    afExtent[1] = kES.GetEigenvalue(1);
    afExtent[2] = kES.GetEigenvalue(2);
}
```

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 the 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 >= 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.

```
{
  gmtlASSERT( eps >= 0 );
  return ( (isEqual(ls1.mOrigin, ls2.mOrigin, eps)) &&
           (isEqual(ls1.mDir, ls2.mDir, eps)) );
}
```

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.

```
{
  // the sphere is inside container if the distance between the centers of the
  // spheres plus the radius of the inner sphere is less than or equal to the
  // radius of the containing sphere.
  // |sphere.center - container.center| + sphere.radius <= container.radius
  return ( length(sphere.mCenter - container.mCenter) + sphere.mRadius
           <= container.mRadius );
}
```

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().

```
{
   // The point is inside the sphere if the vector computed from the center of
   // the sphere to the point has a magnitude less than or equal to the radius
   // of the sphere.
   // |pt - center| <= radius
   return ( length(pt - container.mCenter) <= container.mRadius );
}
```

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 of 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(), gmtlASSERT, length(), gmtl::Sphere< DATA_TYPE >::mCenter, and gmtl::Sphere< DATA_TYPE >::mRadius.

```
{
   gmtlASSERT( tol >= 0 && "tolerance must be positive" );

   // abs( |center-pt| - radius ) < tol
   return ( Math::abs( length(container.mCenter - pt) - container.mRadius )
               <= tol );
}
```

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.

```
{
   // |center - pt| - radius == 0
   return ( length(container.mCenter - pt) - container.mRadius == 0 );
}
```

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().

```
{
   gmtlASSERT( pts.size() > 0  && "pts must contain at least 1 point" );

   // Implementation based on the Sphere Centered at Average of Points algorithm
   // found in "3D Game Engine Design" by Devud G, Eberly (pg. 27)
   std::vector< Point<DATA_TYPE, 3> >::const_iterator itr = pts.begin();

   // compute the average of the points as the center
   Point<DATA_TYPE, 3> sum = *itr;
   ++itr;
   while ( itr != pts.end() )
   {
      sum += *itr;
      ++itr;
   }
   container.mCenter = sum / pts.size();

   // compute the distance from the computed center to point furthest from that
   // center as the radius
   DATA_TYPE radiusSqr(0);
   for ( itr = pts.begin(); itr != pts.end(); ++itr )
   {
      float len = lengthSquared( *itr - container.mCenter );
      if ( len > radiusSqr )
         radiusSqr = len;
   }

   container.mRadius = Math::sqrt( radiusSqr );
}
```

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.

```
{
   return ( ! (ls1 == ls2) );
}
```

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.

```
{
   return ( (ls1.mOrigin == ls2.mOrigin) && (ls1.mDir == ls2.mDir) );
}
```

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.

```
{
   osg_mat.setValue( mat.getData() );
   return osg_mat;
}
```

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().

```
{
   mat.set(osg_mat.getValues());
   return mat;
}
```

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.

```
{
   return TestIntersectOBB<true>(time, box0, vel0, box1, vel1, tFirstContact);
}
```

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.

```
{
   float unused;
   return TestIntersectOBB<false>(time, box0, vel0, box1, vel1, unused);
}
```

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().

```
{
   // convenience variables
   const Vec3* aAxes = box0.axes();
```

```
    const Vec3* bAxes = box1.axes();
    const float* aExtents = box0.halfLens();
    const float* bExtents = box1.halfLens();

    // compute difference of box centers, D = C1-C0
    Vec3 dist = box1.center() - box0.center();

    float rMat[3][3];     // matrix rMat = A^T B, c_{ij} = Dot(A_i,B_j)
    float rMatAbs[3][3];  // |c_{ij}|
    // NOTE: Since it is not a Matrix4, I am using a different ordering here
    float aDotD[3];        // Dot(A_i,D)
    float fR0, fR1, fR;   // interval radii and distance between centers
    float fR01;           // = R0 + R1

    // axis C0+t*A0
    rMat[0][0] = aAxes[0].dot(bAxes[0]);
    rMat[0][1] = aAxes[0].dot(bAxes[1]);
    rMat[0][2] = aAxes[0].dot(bAxes[2]);
    aDotD[0] = aAxes[0].dot(dist);
    rMatAbs[0][0] = Math::abs(rMat[0][0]);
    rMatAbs[0][1] = Math::abs(rMat[0][1]);
    rMatAbs[0][2] = Math::abs(rMat[0][2]);
    fR = Math::abs(aDotD[0]);
    fR1 = bExtents[0]*rMatAbs[0][0]+bExtents[1]*rMatAbs[0][1]+bExtents[2]*rMatAbs[0][2];
    fR01 = aExtents[0] + fR1;
    if ( fR > fR01 )
        return false;

    // axis C0+t*A1
    rMat[1][0] = aAxes[1].dot(bAxes[0]);
    rMat[1][1] = aAxes[1].dot(bAxes[1]);
    rMat[1][2] = aAxes[1].dot(bAxes[2]);
    aDotD[1] = aAxes[1].dot(dist);
    rMatAbs[1][0] = Math::abs(rMat[1][0]);
    rMatAbs[1][1] = Math::abs(rMat[1][1]);
    rMatAbs[1][2] = Math::abs(rMat[1][2]);
    fR = Math::abs(aDotD[1]);
    fR1 = bExtents[0]*rMatAbs[1][0]+bExtents[1]*rMatAbs[1][1]+bExtents[2]*rMatAbs[1][2];
    fR01 = aExtents[1] + fR1;
    if ( fR > fR01 )
        return false;

    // axis C0+t*A2
    rMat[2][0] = aAxes[2].dot(bAxes[0]);
    rMat[2][1] = aAxes[2].dot(bAxes[1]);
    rMat[2][2] = aAxes[2].dot(bAxes[2]);
    aDotD[2] = aAxes[2].dot(dist);
    rMatAbs[2][0] = Math::abs(rMat[2][0]);
    rMatAbs[2][1] = Math::abs(rMat[2][1]);
    rMatAbs[2][2] = Math::abs(rMat[2][2]);
    fR = Math::abs(aDotD[2]);
    fR1 = bExtents[0]*rMatAbs[2][0]+bExtents[1]*rMatAbs[2][1]+bExtents[2]*rMatAbs[2][2];
    fR01 = aExtents[2] + fR1;
    if ( fR > fR01 )
        return false;
```

```

   // axis C0+t*B0
   fR = Math::abs(bAxes[0].dot(dist));
   fR0 = aExtents[0]*rMatAbs[0][0]+aExtents[1]*rMatAbs[1][0]+aExtents[2]*rMatAbs[2][0];
   fR01 = fR0 + bExtents[0];
   if ( fR > fR01 )
      return false;

   // axis C0+t*B1
   fR = Math::abs(bAxes[1].dot(dist));
   fR0 = aExtents[0]*rMatAbs[0][1]+aExtents[1]*rMatAbs[1][1]+aExtents[2]*rMatAbs[2][1];
   fR01 = fR0 + bExtents[1];
   if ( fR > fR01 )
      return false;

   // axis C0+t*B2
   fR = Math::abs(bAxes[2].dot(dist));
   fR0 = aExtents[0]*rMatAbs[0][2]+aExtents[1]*rMatAbs[1][2]+aExtents[2]*rMatAbs[2][2];
   fR01 = fR0 + bExtents[2];
   if ( fR > fR01 )
      return false;

   // axis C0+t*A0xB0
   fR = Math::abs(aDotD[2]*rMat[1][0]-aDotD[1]*rMat[2][0]);
   fR0 = aExtents[1]*rMatAbs[2][0] + aExtents[2]*rMatAbs[1][0];
   fR1 = bExtents[1]*rMatAbs[0][2] + bExtents[2]*rMatAbs[0][1];
   fR01 = fR0 + fR1;
   if ( fR > fR01 )
      return false;

   // axis C0+t*A0xB1
   fR = Math::abs(aDotD[2]*rMat[1][1]-aDotD[1]*rMat[2][1]);
   fR0 = aExtents[1]*rMatAbs[2][1] + aExtents[2]*rMatAbs[1][1];
   fR1 = bExtents[0]*rMatAbs[0][2] + bExtents[2]*rMatAbs[0][0];
   fR01 = fR0 + fR1;
   if ( fR > fR01 )
      return false;

   // axis C0+t*A0xB2
   fR = Math::abs(aDotD[2]*rMat[1][2]-aDotD[1]*rMat[2][2]);
   fR0 = aExtents[1]*rMatAbs[2][2] + aExtents[2]*rMatAbs[1][2];
   fR1 = bExtents[0]*rMatAbs[0][1] + bExtents[1]*rMatAbs[0][0];
   fR01 = fR0 + fR1;
   if ( fR > fR01 )
      return false;

   // axis C0+t*A1xB0
   fR = Math::abs(aDotD[0]*rMat[2][0]-aDotD[2]*rMat[0][0]);
   fR0 = aExtents[0]*rMatAbs[2][0] + aExtents[2]*rMatAbs[0][0];
   fR1 = bExtents[1]*rMatAbs[1][2] + bExtents[2]*rMatAbs[1][1];
   fR01 = fR0 + fR1;
   if ( fR > fR01 )
      return false;

   // axis C0+t*A1xB1
```

```
      fR = Math::abs(aDotD[0]*rMat[2][1]-aDotD[2]*rMat[0][1]);
      fR0 = aExtents[0]*rMatAbs[2][1] + aExtents[2]*rMatAbs[0][1];
      fR1 = bExtents[0]*rMatAbs[1][2] + bExtents[2]*rMatAbs[1][0];
      fR01 = fR0 + fR1;
      if ( fR > fR01 )
         return false;

      // axis C0+t*A1xB2
      fR = Math::abs(aDotD[0]*rMat[2][2]-aDotD[2]*rMat[0][2]);
      fR0 = aExtents[0]*rMatAbs[2][2] + aExtents[2]*rMatAbs[0][2];
      fR1 = bExtents[0]*rMatAbs[1][1] + bExtents[1]*rMatAbs[1][0];
      fR01 = fR0 + fR1;
      if ( fR > fR01 )
         return false;

      // axis C0+t*A2xB0
      fR = Math::abs(aDotD[1]*rMat[0][0]-aDotD[0]*rMat[1][0]);
      fR0 = aExtents[0]*rMatAbs[1][0] + aExtents[1]*rMatAbs[0][0];
      fR1 = bExtents[1]*rMatAbs[2][2] + bExtents[2]*rMatAbs[2][1];
      fR01 = fR0 + fR1;
      if ( fR > fR01 )
         return false;

      // axis C0+t*A2xB1
      fR = Math::abs(aDotD[1]*rMat[0][1]-aDotD[0]*rMat[1][1]);
      fR0 = aExtents[0]*rMatAbs[1][1] + aExtents[1]*rMatAbs[0][1];
      fR1 = bExtents[0]*rMatAbs[2][2] + bExtents[2]*rMatAbs[2][0];
      fR01 = fR0 + fR1;
      if ( fR > fR01 )
         return false;

      // axis C0+t*A2xB2
      fR = Math::abs(aDotD[1]*rMat[0][2]-aDotD[0]*rMat[1][2]);
      fR0 = aExtents[0]*rMatAbs[1][2] + aExtents[1]*rMatAbs[0][2];
      fR1 = bExtents[0]*rMatAbs[2][1] + bExtents[1]*rMatAbs[2][0];
      fR01 = fR0 + fR1;
      if ( fR > fR01 )
         return false;

      return true;

   }
```

9.1.2.36 template<bool FIND_CONTACT> bool TestIntersectOBB (float *time*, const OOBox & *box0*, const Vec3 & *vel0*, const OOBox & *box1*, const Vec3 & *vel1*, float & *tFirstContact*) `[inline]`

Definition at line 302 of file Intersection.h.

References gmtl::Math::abs(), gmtl::OOBox::axes(), gmtl::OOBox::center(), dot(), and gmtl::OOBox::halfLens().

```
{
   // convenience variables
   const Vec3* aAxes = box0.axes();
   const Vec3* bAxes = box1.axes();
   const float* aExtents = box0.halfLens();
   const float* bExtents = box1.halfLens();

   // Compute relative velocity of box1 with respect to box0 so that box0
   // may as well be stationary.
   Vec3 kW = vel1 - vel0;

   // Compute difference of box centers at time 0 and time 'fTime'.
   Vec3 dist0 = box1.center() - box0.center();
   Vec3 dist1 = dist0 + time*kW;

   float rMat[3][3];       // matrix C = A^T B, c_{ij} = dot(A_i,B_j)
   float rMatAbs[3][3];    // |c_{ij}|
   float aDotD0[3];        // dot(A_i,D0)
   float aDotD1[3];        // dot(A_i,D1)
   float bDotD0[3];        // dot(B_i,D0)
   float bDotD1[3];        // dot(B_i,D1)
   float fR0, fR1, fR;     // interval radii and distance between centers
   float fR01;             // = R0 + R1

   // Track minimum time
   if(FIND_CONTACT)
   {
      tFirstContact = 0.0f;
   }

   // axis C0+t*A0
   rMat[0][0] = aAxes[0].dot(bAxes[0]);
   rMat[0][1] = aAxes[0].dot(bAxes[1]);
   rMat[0][2] = aAxes[0].dot(bAxes[2]);
   aDotD0[0] = aAxes[0].dot(dist0);
   aDotD1[0] = aAxes[0].dot(dist1);
   rMatAbs[0][0] = Math::abs(rMat[0][0]);
   rMatAbs[0][1] = Math::abs(rMat[0][1]);
   rMatAbs[0][2] = Math::abs(rMat[0][2]);
   fR1 = bExtents[0]*rMatAbs[0][0]+bExtents[1]*rMatAbs[0][1]+bExtents[2]*rMatAbs[0][2];
   fR01 = aExtents[0] + fR1;
   //if(!FIND_CONTACT)
   //{
      if(!dynObbFind0<FIND_CONTACT>(aDotD0[0], aDotD1[0], fR01, time, tFirstContact))
         return false;
   //}
   //else
   //{

   //}
   /*
   if ( aDotD0[0] > fR01 )
   {
      if ( aDotD1[0] > fR01 )
         return false;
```

```
    }
    else if ( aDotD0[0] < -fR01 )
    {
       if ( aDotD1[0] < -fR01 )
          return false;
    }
    */

    // axis C0+t*A1
    rMat[1][0] = aAxes[1].dot(bAxes[0]);
    rMat[1][1] = aAxes[1].dot(bAxes[1]);
    rMat[1][2] = aAxes[1].dot(bAxes[2]);
    aDotD0[1] = aAxes[1].dot(dist0);
    aDotD1[1] = aAxes[1].dot(dist1);
    rMatAbs[1][0] = Math::abs(rMat[1][0]);
    rMatAbs[1][1] = Math::abs(rMat[1][1]);
    rMatAbs[1][2] = Math::abs(rMat[1][2]);
    fR1 = bExtents[0]*rMatAbs[1][0]+bExtents[1]*rMatAbs[1][1]+bExtents[2]*rMatAbs[1][2];
    fR01 = aExtents[1] + fR1;
    if(!dynObbFind0<FIND_CONTACT>(aDotD0[1], aDotD1[1], fR01, time, tFirstContact))
       return false;
    /*
    if ( aDotD0[1] > fR01 )
    {
       if ( aDotD1[1] > fR01 )
          return false;
    }
    else if ( aDotD0[1] < -fR01 )
    {
       if ( aDotD1[1] < -fR01 )
          return false;
    }
    */

    // axis C0+t*A2
    rMat[2][0] = aAxes[2].dot(bAxes[0]);
    rMat[2][1] = aAxes[2].dot(bAxes[1]);
    rMat[2][2] = aAxes[2].dot(bAxes[2]);
    aDotD0[2] = aAxes[2].dot(dist0);
    aDotD1[2] = aAxes[2].dot(dist1);
    rMatAbs[2][0] = Math::abs(rMat[2][0]);
    rMatAbs[2][1] = Math::abs(rMat[2][1]);
    rMatAbs[2][2] = Math::abs(rMat[2][2]);
    fR1 = bExtents[0]*rMatAbs[2][0]+bExtents[1]*rMatAbs[2][1]+bExtents[2]*rMatAbs[2][2];
    fR01 = aExtents[2] + fR1;
    if(!dynObbFind0<FIND_CONTACT>(aDotD0[2], aDotD1[2], fR01, time, tFirstContact))
       return false;
    /*
    if ( aDotD0[2] > fR01 )
    {
       if ( aDotD1[2] > fR01 )
          return false;
    }
    else if ( aDotD0[2] < -fR01 )
    {
```

```
         if ( aDotD1[2] < -fR01 )
            return false;
      }
      */

      // axis C0+t*B0
      bDotD0[0] = bAxes[0].dot(dist0);
      bDotD1[0] = bAxes[0].dot(dist1);
      //fR = bAxes[0].dot(dist0);
      fR0 = aExtents[0]*rMatAbs[0][0]+aExtents[1]*rMatAbs[1][0]+aExtents[2]*rMatAbs[2][0];
      fR01 = fR0 + bExtents[0];
      if(!dynObbFind0<FIND_CONTACT>(bDotD0[0], bDotD1[0], fR01, time, tFirstContact))
         return false;
      /*
      if ( fR > fR01 )
      {
         fR = bAxes[0].dot(dist1);
         if ( fR > fR01)
            return false;
      }
      else if ( fR < -fR01 )
      {
         fR = bAxes[0].dot(dist1);
         if ( fR < -fR01 )
            return false;
      }
      */

      // axis C0+t*B1
      bDotD0[1] = bAxes[1].dot(dist0);
      bDotD1[1] = bAxes[1].dot(dist1);
      //fR = bAxes[1].dot(dist0);
      fR0 = aExtents[0]*rMatAbs[0][1]+aExtents[1]*rMatAbs[1][1]+aExtents[2]*rMatAbs[2][1];
      fR01 = fR0 + bExtents[1];
      if(!dynObbFind0<FIND_CONTACT>(bDotD0[1], bDotD1[1], fR01, time, tFirstContact))
         return false;

      /*
      if ( fR > fR01 )
      {
         fR = bAxes[1].dot(dist1);
         if ( fR > fR01 )
            return false;
      }
      else if ( fR < -fR01 )
      {
         fR = bAxes[1].dot(dist1);
         if ( fR < -fR01 )
            return false;
      }
      */

      // axis C0+t*B2
      bDotD0[2] = bAxes[2].dot(dist0);
      bDotD1[2] = bAxes[2].dot(dist1);
```

```
      //fR = bAxes[2].dot(dist0);
      fR0 = aExtents[0]*rMatAbs[0][2]+aExtents[1]*rMatAbs[1][2]+aExtents[2]*rMatAbs[2][2];
      fR01 = fR0 + bExtents[2];
      if(!dynObbFind0<FIND_CONTACT>(bDotD0[2], bDotD1[2], fR01, time, tFirstContact))
         return false;

      /*
      if ( fR > fR01 )
      {
         fR = bAxes[2].dot(dist1);
         if ( fR > fR01 )
            return false;
      }
      else if ( fR < -fR01 )
      {
         fR = bAxes[2].dot(dist1);
         if ( fR < -fR01 )
            return false;
      }
      */

      // axis C0+t*A0xB0
      fR = aDotD0[2]*rMat[1][0]-aDotD0[1]*rMat[2][0];
      fR0 = aExtents[1]*rMatAbs[2][0] + aExtents[2]*rMatAbs[1][0];
      fR1 = bExtents[1]*rMatAbs[0][2] + bExtents[2]*rMatAbs[0][1];
      fR01 = fR0 + fR1;

      if(!dynObbFind1<FIND_CONTACT>(fR, aDotD1[2], rMat[1][0], aDotD1[1], rMat[2][0], fR01,
         return false;
      /*
      if ( fR > fR01 )
      {
         fR = aDotD1[2]*rMat[1][0]-aDotD1[1]*rMat[2][0];
         if ( fR > fR01 )
            return false;
      }
      else if ( fR < -fR01 )
      {
         fR = aDotD1[2]*rMat[1][0]-aDotD1[1]*rMat[2][0];
         if ( fR < -fR01 )
            return false;
      }
      */

      // axis C0+t*A0xB1
      fR = aDotD0[2]*rMat[1][1]-aDotD0[1]*rMat[2][1];
      fR0 = aExtents[1]*rMatAbs[2][1] + aExtents[2]*rMatAbs[1][1];
      fR1 = bExtents[0]*rMatAbs[0][2] + bExtents[2]*rMatAbs[0][0];
      fR01 = fR0 + fR1;
      if(!dynObbFind1<FIND_CONTACT>(fR, aDotD1[2], rMat[1][1], aDotD1[1], rMat[2][1], fR01,
         return false;

      /*if ( fR > fR01 )
      {
         fR = aDotD1[2]*rMat[1][1]-aDotD1[1]*rMat[2][1];
```

```
         if ( fR > fR01 )
            return false;
      }
      else if ( fR < -fR01 )
      {
         fR = aDotD1[2]*rMat[1][1]-aDotD1[1]*rMat[2][1];
         if ( fR < -fR01 )
            return false;
      }*/

      // axis C0+t*A0xB2
      fR = aDotD0[2]*rMat[1][2]-aDotD0[1]*rMat[2][2];
      fR0 = aExtents[1]*rMatAbs[2][2] + aExtents[2]*rMatAbs[1][2];
      fR1 = bExtents[0]*rMatAbs[0][1] + bExtents[1]*rMatAbs[0][0];
      fR01 = fR0 + fR1;
      if(!dynObbFind1<FIND_CONTACT>(fR,aDotD1[2],rMat[1][2],aDotD1[1],rMat[2][2],fR01, time, tFirstCon
         return false;

      /*if ( fR > fR01 )
      {
         fR = aDotD1[2]*rMat[1][2]-aDotD1[1]*rMat[2][2];
         if ( fR > fR01 )
            return false;
      }
      else if ( fR < -fR01 )
      {
         fR = aDotD1[2]*rMat[1][2]-aDotD1[1]*rMat[2][2];
         if ( fR < -fR01 )
            return false;
      }*/

      // axis C0+t*A1xB0
      fR = aDotD0[0]*rMat[2][0]-aDotD0[2]*rMat[0][0];
      fR0 = aExtents[0]*rMatAbs[2][0] + aExtents[2]*rMatAbs[0][0];
      fR1 = bExtents[1]*rMatAbs[1][2] + bExtents[2]*rMatAbs[1][1];
      fR01 = fR0 + fR1;
      if(!dynObbFind1<FIND_CONTACT>(fR,aDotD1[0],rMat[2][0],aDotD1[2],rMat[0][0],fR01, time, tFirstCon
         return false;
      /*if ( fR > fR01 )
      {
         fR = aDotD1[0]*rMat[2][0]-aDotD1[2]*rMat[0][0];
         if ( fR > fR01 )
            return false;
      }
      else if ( fR < -fR01 )
      {
         fR = aDotD1[0]*rMat[2][0]-aDotD1[2]*rMat[0][0];
         if ( fR < -fR01 )
            return false;
      }*/

      // axis C0+t*A1xB1
      fR = aDotD0[0]*rMat[2][1]-aDotD0[2]*rMat[0][1];
      fR0 = aExtents[0]*rMatAbs[2][1] + aExtents[2]*rMatAbs[0][1];
      fR1 = bExtents[0]*rMatAbs[1][2] + bExtents[2]*rMatAbs[1][0];
```

```
      fR01 = fR0 + fR1;
      if(!dynObbFind1<FIND_CONTACT>(fR,aDotD1[0],rMat[2][1],aDotD1[2],rMat[0][1],fR01, time
         return false;
      /*if ( fR > fR01 )
      {
         fR = aDotD1[0]*rMat[2][1]-aDotD1[2]*rMat[0][1];
         if ( fR > fR01 )
            return false;
      }
      else if ( fR < -fR01 )
      {
         fR = aDotD1[0]*rMat[2][1]-aDotD1[2]*rMat[0][1];
         if ( fR < -fR01 )
            return false;
      }*/

      // axis C0+t*A1xB2
      fR = aDotD0[0]*rMat[2][2]-aDotD0[2]*rMat[0][2];
      fR0 = aExtents[0]*rMatAbs[2][2] + aExtents[2]*rMatAbs[0][2];
      fR1 = bExtents[0]*rMatAbs[1][1] + bExtents[1]*rMatAbs[1][0];
      fR01 = fR0 + fR1;
      if(!dynObbFind1<FIND_CONTACT>(fR,aDotD1[0],rMat[2][2],aDotD1[2],rMat[0][2],fR01, time
         return false;
      /*if ( fR > fR01 )
      {
         fR = aDotD1[0]*rMat[2][2]-aDotD1[2]*rMat[0][2];
         if ( fR > fR01 )
            return false;
      }
      else if ( fR < -fR01 )
      {
         fR = aDotD1[0]*rMat[2][2]-aDotD1[2]*rMat[0][2];
         if ( fR < -fR01 )
            return false;
      }
      */

      // axis C0+t*A2xB0
      fR = aDotD0[1]*rMat[0][0]-aDotD0[0]*rMat[1][0];
      fR0 = aExtents[0]*rMatAbs[1][0] + aExtents[1]*rMatAbs[0][0];
      fR1 = bExtents[1]*rMatAbs[2][2] + bExtents[2]*rMatAbs[2][1];
      fR01 = fR0 + fR1;
      if(!dynObbFind1<FIND_CONTACT>(fR,aDotD1[1],rMat[0][0],aDotD1[0],rMat[1][0],fR01, time
         return false;
      /*if ( fR > fR01 )
      {
         fR = aDotD1[1]*rMat[0][0]-aDotD1[0]*rMat[1][0];
         if ( fR > fR01 )
            return false;
      }
      else if ( fR < -fR01 )
      {
         fR = aDotD1[1]*rMat[0][0]-aDotD1[0]*rMat[1][0];
         if ( fR < -fR01 )
            return false;
```

```
      }*/

      // axis C0+t*A2xB1
      fR = aDotD0[1]*rMat[0][1]-aDotD0[0]*rMat[1][1];
      fR0 = aExtents[0]*rMatAbs[1][1] + aExtents[1]*rMatAbs[0][1];
      fR1 = bExtents[0]*rMatAbs[2][2] + bExtents[2]*rMatAbs[2][0];
      fR01 = fR0 + fR1;
      if(!dynObbFind1<FIND_CONTACT>(fR,aDotD1[1],rMat[0][1],aDotD1[0],rMat[1][1],fR01, time, tFirstCon
         return false;
      /*if ( fR > fR01 )
      {
         fR = aDotD1[1]*rMat[0][1]-aDotD1[0]*rMat[1][1];
         if ( fR > fR01 )
            return false;
      }
      else if ( fR < -fR01 )
      {
         fR = aDotD1[1]*rMat[0][1]-aDotD1[0]*rMat[1][1];
         if ( fR < -fR01 )
            return false;
      }*/

      // axis C0+t*A2xB2
      fR = aDotD0[1]*rMat[0][2]-aDotD0[0]*rMat[1][2];
      fR0 = aExtents[0]*rMatAbs[1][2] + aExtents[1]*rMatAbs[0][2];
      fR1 = bExtents[0]*rMatAbs[2][1] + bExtents[1]*rMatAbs[2][0];
      fR01 = fR0 + fR1;
      if(!dynObbFind1<FIND_CONTACT>(fR,aDotD1[1],rMat[0][2],aDotD1[0],rMat[1][2],fR01, time, tFirstCon
         return false;
      /*if ( fR > fR01 )
      {
         fR = aDotD1[1]*rMat[0][2]-aDotD1[0]*rMat[1][2];
         if ( fR > fR01 )
            return false;
      }
      else if ( fR < -fR01 )
      {
         fR = aDotD1[1]*rMat[0][2]-aDotD1[0]*rMat[1][2];
         if ( fR < -fR01 )
            return false;
      }*/

      // At this point none of the 15 axes separate the boxes.  It is still
      // possible that they are separated as viewed in any plane orthogonal
      // to the relative direction of motion W.  In the worst case, the two
      // projected boxes are hexagons.  This requires three separating axis
      // tests per box.
      Vec3 kWxD0 = kW.cross(dist0);
      float wDotA[3], wDotB[3];

      // axis C0 + t*WxA0
      wDotA[1] = kW.dot(aAxes[1]);
      wDotA[2] = kW.dot(aAxes[2]);
      fR = Math::abs(aAxes[0].dot(kWxD0));
      fR0 = aExtents[1]*wDotA[2] + aExtents[2]*wDotA[1];
```

```
    fR1 =
       bExtents[0]*Math::abs(rMat[1][0]*wDotA[2] - rMat[2][0]*wDotA[1]) +
       bExtents[1]*Math::abs(rMat[1][1]*wDotA[2] - rMat[2][1]*wDotA[1]) +
       bExtents[2]*Math::abs(rMat[1][2]*wDotA[2] - rMat[2][2]*wDotA[1]);
    fR01 = fR0 + fR1;
    if ( fR > fR01 )
       return false;

    // axis C0 + t*WxA1
    wDotA[0] = kW.dot(aAxes[0]);
    fR = Math::abs(aAxes[1].dot(kWxD0));
    fR0 = aExtents[2]*wDotA[0] + aExtents[0]*wDotA[2];
    fR1 =
       bExtents[0]*Math::abs(rMat[2][0]*wDotA[0] - rMat[0][0]*wDotA[2]) +
       bExtents[1]*Math::abs(rMat[2][1]*wDotA[0] - rMat[0][1]*wDotA[2]) +
       bExtents[2]*Math::abs(rMat[2][2]*wDotA[0] - rMat[0][2]*wDotA[2]);
    fR01 = fR0 + fR1;
    if ( fR > fR01 )
       return false;

    // axis C0 + t*WxA2
    fR = Math::abs(aAxes[2].dot(kWxD0));
    fR0 = aExtents[0]*wDotA[1] + aExtents[1]*wDotA[0];
    fR1 =
       bExtents[0]*Math::abs(rMat[0][0]*wDotA[1] - rMat[1][0]*wDotA[0]) +
       bExtents[1]*Math::abs(rMat[0][1]*wDotA[1] - rMat[1][1]*wDotA[0]) +
       bExtents[2]*Math::abs(rMat[0][2]*wDotA[1] - rMat[1][2]*wDotA[0]);
    fR01 = fR0 + fR1;
    if ( fR > fR01 )
       return false;

    // axis C0 + t*WxB0
    wDotB[1] = kW.dot(bAxes[1]);
    wDotB[2] = kW.dot(bAxes[2]);
    fR = Math::abs(bAxes[0].dot(kWxD0));
    fR0 =
       aExtents[0]*Math::abs(rMat[0][1]*wDotB[2] - rMat[0][2]*wDotB[1]) +
       aExtents[1]*Math::abs(rMat[1][1]*wDotB[2] - rMat[1][2]*wDotB[1]) +
       aExtents[2]*Math::abs(rMat[2][1]*wDotB[2] - rMat[2][2]*wDotB[1]);
    fR1 = bExtents[1]*wDotB[2] + bExtents[2]*wDotB[1];
    fR01 = fR0 + fR1;
    if ( fR > fR01 )
       return false;

    // axis C0 + t*WxB1
    wDotB[0] = kW.dot(bAxes[0]);
    fR = Math::abs(bAxes[1].dot(kWxD0));
    fR0 =
       aExtents[0]*Math::abs(rMat[0][2]*wDotB[0] - rMat[0][0]*wDotB[2]) +
       aExtents[1]*Math::abs(rMat[1][2]*wDotB[0] - rMat[1][0]*wDotB[2]) +
       aExtents[2]*Math::abs(rMat[2][2]*wDotB[0] - rMat[2][0]*wDotB[2]);
    fR1 = bExtents[2]*wDotB[0] + bExtents[0]*wDotB[2];
    fR01 = fR0 + fR1;
    if ( fR > fR01 )
       return false;
```

```

    // axis C0 + t*WxB2
    fR = Math::abs(bAxes[2].dot(kWxD0));
    fR0 =
       aExtents[0]*Math::abs(rMat[0][0]*wDotB[1] - rMat[0][1]*wDotB[0]) +
       aExtents[1]*Math::abs(rMat[1][0]*wDotB[1] - rMat[1][1]*wDotB[0]) +
       aExtents[2]*Math::abs(rMat[2][0]*wDotB[1] - rMat[2][1]*wDotB[0]);
    fR1 = bExtents[0]*wDotB[1] + bExtents[1]*wDotB[0];
    fR01 = fR0 + fR1;
    if ( fR > fR01 )
       return false;

    return true;

 }
```

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

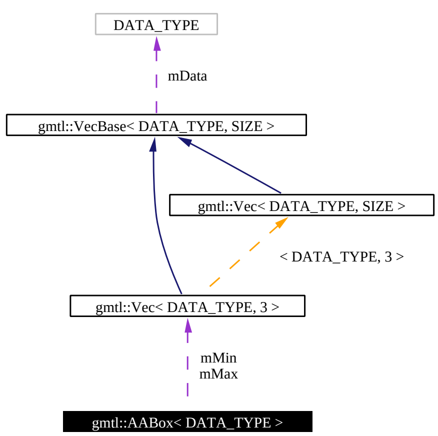
GenericMathTemplateLibrary Class Documentation

10.1 gmtl::AABox< DATA_TYPE > Class Template Reference

Describes an axially aligned box in 3D space.

`#include <AABox.h>`

Collaboration diagram for gmtl::AABox< 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)

  Construcst 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.

```
      : mMin(0,0,0), mMax(0,0,0), mEmpty(true)
   {}
```

10.1.3.2 template<class DATA_TYPE> gmtl::AABox< DATA_TYPE >::AABox (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::AABox< DATA_TYPE >::mEmpty, gmtl::AABox< DATA_TYPE >::mMax, and gmtl::AABox< DATA_TYPE >::mMin.

```
74          : mMin(min), mMax(max), mEmpty(false)
75       {}
```

10.1.3.3 template<class DATA_TYPE> gmtl::AABox< DATA_TYPE >::AABox (const AABox< DATA_TYPE > & *box*) `[inline]`

Construcst a duplicate of the given box.

Parameters:

box the box the make a copy of

Definition at line 82 of file AABox.h.

References gmtl::AABox< DATA_TYPE >::mEmpty, gmtl::AABox< DATA_TYPE >::mMax, and gmtl::AABox< 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.

```
{
   return mMax;
}
```

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.

```
{
   return mMin;
}
```

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.

```
{
   return mEmpty;
}
```

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.

```
{
   mEmpty = empty;
}
```

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.

```
{
   mMax = max;
}
```

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.

```
{
    mMin = min;
}
```

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:

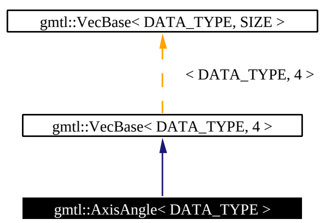
- AABox.h

10.2 gmtl::AxisAngle< DATA_TYPE > Class Template Reference

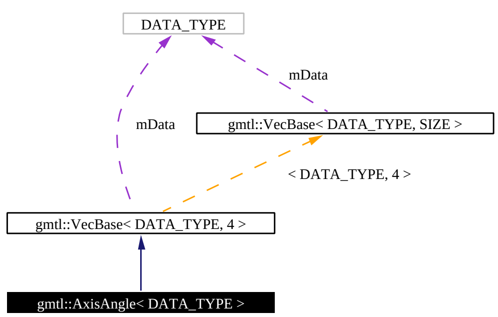
AxisAngle: Represents a "twist about an axis" AxisAngle is used to specify a rotation in 3-space.

```
#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.

```
{
   return VecBase<DATA_TYPE, 4>::operator[]( 0 );
}
```

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.

```
{
   return Vec<DATA_TYPE, 3>( VecBase<DATA_TYPE, 4>::operator[]( 1 ),
                             VecBase<DATA_TYPE, 4>::operator[]( 2 ),
                             VecBase<DATA_TYPE, 4>::operator[]( 3 ) );
}
```

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().

```
{
   VecBase<DATA_TYPE, 4>::set( rad_angle, axis[0], axis[1], axis[2] );
}
```

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().

```
{
   VecBase<DATA_TYPE, 4>::set( rad_angle, x, y, z );
}
```

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.

```
{
   VecBase<DATA_TYPE, 4>::operator[]( 0 ) = rad_angle;
}
```

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.

```
{
   VecBase<DATA_TYPE, 4>::operator[]( 1 ) = axis[0];
   VecBase<DATA_TYPE, 4>::operator[]( 2 ) = axis[1];
   VecBase<DATA_TYPE, 4>::operator[]( 3 ) = axis[2];
}
```

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.

```
      {
         float xVal = sortDir.dot((*points)[x]);
         float yVal = sortDir.dot((*points)[y]);

         return (xVal < yVal);
      }
```

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:

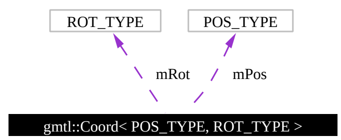
- Comparitors.h

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.

```
{
    assert( iSize >= 2 );
    m_iSize = iSize;

    m_aafMat = new float*[m_iSize];
    for (int i = 0; i < m_iSize; i++)
        m_aafMat[i] = new float[m_iSize];

    m_afDiag = new float[m_iSize];
    m_afSubd = new float[m_iSize];
}
```

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.

```
{
    delete[] m_afSubd;
    delete[] m_afDiag;
    for (int i = 0; i < m_iSize; i++)
        delete[] m_aafMat[i];
    delete[] m_aafMat;
}
```

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().

```
{
    // sort eigenvalues in decreasing order, e[0] >= ... >= e[iSize-1]
    for (int i0 = 0, i1; i0 <= iSize-2; i0++)
    {
        // locate maximum eigenvalue
        i1 = i0;
        float fMax = afEigval[i1];
        int i2;
        for (i2 = i0+1; i2 < iSize; i2++)
        {
            if ( afEigval[i2] > fMax )
            {
                i1 = i2;
                fMax = afEigval[i1];
            }
        }

        if ( i1 != i0 )
        {
            // swap eigenvalues
            afEigval[i1] = afEigval[i0];
            afEigval[i0] = fMax;

            // swap eigenvectors
            for (i2 = 0; i2 < iSize; i2++)
            {
                float fTmp = aafEigvec[i2][i0];
                aafEigvec[i2][i0] = aafEigvec[i2][i1];
                aafEigvec[i2][i1] = fTmp;
            }
        }
    }
}
```

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().

```
{
    switch ( m_iSize )
    {
        case 2:
            Tridiagonal2(m_aafMat,m_afDiag,m_afSubd);
            break;
        case 3:
            Tridiagonal3(m_aafMat,m_afDiag,m_afSubd);
            break;
        case 4:
            Tridiagonal4(m_aafMat,m_afDiag,m_afSubd);
            break;
        default:
            TridiagonalN(m_iSize,m_aafMat,m_afDiag,m_afSubd);
            break;
    }
    QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
    DecreasingSort(m_iSize,m_afDiag,m_aafMat);
}
```

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().

```
{
    Tridiagonal2(m_aafMat,m_afDiag,m_afSubd);
    QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
    DecreasingSort(m_iSize,m_afDiag,m_aafMat);
}
```

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().

```
{
    Tridiagonal3(m_aafMat,m_afDiag,m_afSubd);
```

```
    QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
    DecreasingSort(m_iSize,m_afDiag,m_aafMat);
}
```

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().

```
{
    Tridiagonal4(m_aafMat,m_afDiag,m_afSubd);
    QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
    DecreasingSort(m_iSize,m_afDiag,m_aafMat);
}
```

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().

```
{
    TridiagonalN(m_iSize,m_aafMat,m_afDiag,m_afSubd);
    QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
    DecreasingSort(m_iSize,m_afDiag,m_aafMat);
}
```

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().

```
{
    switch ( m_iSize )
    {
```

```
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().

```
{
    return m_afDiag[i];
}
```

10.5.2.14 float ** gmtl::Eigen::GetEigenvector () [inline]

Definition at line 133 of file Eigen.h.

References m_aafMat.

```
{
    return m_aafMat;
}
```

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().

```
{
    return m_aafMat[iRow][iCol];
}
```

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().

```
{
    // sort eigenvalues in increasing order, e[0] <= ... <= e[iSize-1]
    for (int i0 = 0, i1; i0 <= iSize-2; i0++)
    {
        // locate minimum eigenvalue
```

```
        i1 = i0;
        float fMin = afEigval[i1];
        int i2;
        for (i2 = i0+1; i2 < iSize; i2++)
        {
            if ( afEigval[i2] < fMin )
            {
                i1 = i2;
                fMin = afEigval[i1];
            }
        }

        if ( i1 != i0 )
        {
            // swap eigenvalues
            afEigval[i1] = afEigval[i0];
            afEigval[i0] = fMin;

            // swap eigenvectors
            for (i2 = 0; i2 < iSize; i2++)
            {
                float fTmp = aafEigvec[i2][i0];
                aafEigvec[i2][i0] = aafEigvec[i2][i1];
                aafEigvec[i2][i1] = fTmp;
            }
        }
    }
}
```

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().

```
{
    switch ( m_iSize )
    {
        case 2:
            Tridiagonal2(m_aafMat,m_afDiag,m_afSubd);
            break;
        case 3:
            Tridiagonal3(m_aafMat,m_afDiag,m_afSubd);
            break;
        case 4:
            Tridiagonal4(m_aafMat,m_afDiag,m_afSubd);
            break;
        default:
            TridiagonalN(m_iSize,m_aafMat,m_afDiag,m_afSubd);
```

```
            break;
    }
    QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
    IncreasingSort(m_iSize,m_afDiag,m_aafMat);
}
```

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().

```
{
    Tridiagonal2(m_aafMat,m_afDiag,m_afSubd);
    QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
    IncreasingSort(m_iSize,m_afDiag,m_aafMat);
}
```

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().

```
{
    Tridiagonal3(m_aafMat,m_afDiag,m_afSubd);
    QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
    IncreasingSort(m_iSize,m_afDiag,m_aafMat);
}
```

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().

```
{
    Tridiagonal4(m_aafMat,m_afDiag,m_afSubd);
    QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
    IncreasingSort(m_iSize,m_afDiag,m_aafMat);
}
```

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().

```
{
    TridiagonalN(m_iSize,m_aafMat,m_afDiag,m_afSubd);
    QLAlgorithm(m_iSize,m_afDiag,m_afSubd,m_aafMat);
    IncreasingSort(m_iSize,m_afDiag,m_aafMat);
}
```

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().

```
{
    return m_aafMat[iRow][iCol];
}
```

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().

```
{
    const int iMaxIter = 32;

    for (int i0 = 0; i0 < iSize; i0++)
    {
        int i1;
        for (i1 = 0; i1 < iMaxIter; i1++)
        {
            int i2;
            for (i2 = i0; i2 <= iSize-2; i2++)
            {
                float fTmp =
                    Math::abs(m_afDiag[i2])+ Math::abs(m_afDiag[i2+1]);
                if ( Math::abs(m_afSubd[i2]) + fTmp == fTmp )
                    break;
            }
            if ( i2 == i0 )
                break;

            float fG = (m_afDiag[i0+1]-m_afDiag[i0])/(2.0*m_afSubd[i0]);
            float fR = Math::sqrt(fG*fG+1.0);
            if ( fG < 0.0 )
                fG = m_afDiag[i2]-m_afDiag[i0]+m_afSubd[i0]/(fG-fR);
            else
                fG = m_afDiag[i2]-m_afDiag[i0]+m_afSubd[i0]/(fG+fR);
            float fSin = 1.0, fCos = 1.0, fP = 0.0;
            for (int i3 = i2-1; i3 >= i0; i3--)
            {
                float fF = fSin*m_afSubd[i3];
                float fB = fCos*m_afSubd[i3];
                if ( Math::abs(fF) >= Math::abs(fG) )
                {
                    fCos = fG/fF;
                    fR = sqrt(fCos*fCos+1.0);
                    m_afSubd[i3+1] = fF*fR;
                    fSin = 1.0/fR;
                    fCos *= fSin;
                }
                else
                {
                    fSin = fF/fG;
                    fR = Math::sqrt(fSin*fSin+1.0);
                    m_afSubd[i3+1] = fG*fR;
                    fCos = 1.0/fR;
                    fSin *= fCos;
                }
                fG = m_afDiag[i3+1]-fP;
                fR = (m_afDiag[i3]-fG)*fSin+2.0*fB*fCos;
                fP = fSin*fR;
                m_afDiag[i3+1] = fG+fP;
                fG = fCos*fR-fB;

                for (int i4 = 0; i4 < iSize; i4++)
                {
                    fF = m_aafMat[i4][i3+1];
```

```
                    m_aafMat[i4][i3+1] = fSin*m_aafMat[i4][i3]+fCos*fF;
                    m_aafMat[i4][i3] = fCos*m_aafMat[i4][i3]-fSin*fF;
                }
            }
            m_afDiag[i0] -= fP;
            m_afSubd[i0] = fG;
            m_afSubd[i2] = 0.0;
        }
        if ( i1 == iMaxIter )
            return false;
    }

    return true;
}
```

10.5.2.24 void gmtl::Eigen::SetMatrix (float ∗∗ *aafMat*)

Definition at line 623 of file Eigen.h.

References m_aafMat, and m_iSize.

```
{
    for (int iRow = 0; iRow < m_iSize; iRow++)
    {
        for (int iCol = 0; iCol < m_iSize; iCol++)
            m_aafMat[iRow][iCol] = aafMat[iRow][iCol];
    }
}
```

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().

```
{
    // matrix is already tridiagonal
    m_afDiag[0] = m_aafMat[0][0];
    m_afDiag[1] = m_aafMat[1][1];
    m_afSubd[0] = m_aafMat[0][1];
    m_afSubd[1] = 0.0;
    m_aafMat[0][0] = 1.0;
```

```
    m_aafMat[0][1] = 0.0;
    m_aafMat[1][0] = 0.0;
    m_aafMat[1][1] = 1.0;
}
```

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().

```
{
    float fM00 = m_aafMat[0][0];
    float fM01 = m_aafMat[0][1];
    float fM02 = m_aafMat[0][2];
    float fM11 = m_aafMat[1][1];
    float fM12 = m_aafMat[1][2];
    float fM22 = m_aafMat[2][2];

    m_afDiag[0] = fM00;
    m_afSubd[2] = 0.0;
    if ( fM02 != 0.0 )
    {
        float fLength = Math::sqrt(fM01*fM01+fM02*fM02);
        float fInvLength = 1.0/fLength;
        fM01 *= fInvLength;
        fM02 *= fInvLength;
        float fQ = 2.0*fM01*fM12+fM02*(fM22-fM11);
        m_afDiag[1] = fM11+fM02*fQ;
        m_afDiag[2] = fM22-fM02*fQ;
        m_afSubd[0] = fLength;
        m_afSubd[1] = fM12-fM01*fQ;
        m_aafMat[0][0] = 1.0; m_aafMat[0][1] = 0.0;  m_aafMat[0][2] = 0.0;
        m_aafMat[1][0] = 0.0; m_aafMat[1][1] = fM01; m_aafMat[1][2] = fM02;
        m_aafMat[2][0] = 0.0; m_aafMat[2][1] = fM02; m_aafMat[2][2] = -fM01;
    }
    else
    {
        m_afDiag[1] = fM11;
        m_afDiag[2] = fM22;
        m_afSubd[0] = fM01;
        m_afSubd[1] = fM12;
        m_aafMat[0][0] = 1.0; m_aafMat[0][1] = 0.0; m_aafMat[0][2] = 0.0;
        m_aafMat[1][0] = 0.0; m_aafMat[1][1] = 1.0; m_aafMat[1][2] = 0.0;
        m_aafMat[2][0] = 0.0; m_aafMat[2][1] = 0.0; m_aafMat[2][2] = 1.0;
    }
}
```

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().

```
{
    // save matrix M
    float fM00 = m_aafMat[0][0];
    float fM01 = m_aafMat[0][1];
    float fM02 = m_aafMat[0][2];
    float fM03 = m_aafMat[0][3];
    float fM11 = m_aafMat[1][1];
    float fM12 = m_aafMat[1][2];
    float fM13 = m_aafMat[1][3];
    float fM22 = m_aafMat[2][2];
    float fM23 = m_aafMat[2][3];
    float fM33 = m_aafMat[3][3];

    m_afDiag[0] = fM00;
    m_afSubd[3] = 0.0;

    m_aafMat[0][0] = 1.0;
    m_aafMat[0][1] = 0.0;
    m_aafMat[0][2] = 0.0;
    m_aafMat[0][3] = 0.0;
    m_aafMat[1][0] = 0.0;
    m_aafMat[2][0] = 0.0;
    m_aafMat[3][0] = 0.0;

    float fLength, fInvLength;

    if ( fM02 != 0.0 || fM03 != 0.0 )
    {
        float fQ11, fQ12, fQ13;
        float fQ21, fQ22, fQ23;
        float fQ31, fQ32, fQ33;

        // build column Q1
        fLength = Math::sqrt(fM01*fM01 + fM02*fM02 + fM03*fM03);
        fInvLength = 1.0/fLength;
        fQ11 = fM01*fInvLength;
        fQ21 = fM02*fInvLength;
        fQ31 = fM03*fInvLength;

        m_afSubd[0] = fLength;

```

```
        // compute S*Q1
        float fV0 = fM11*fQ11+fM12*fQ21+fM13*fQ31;
        float fV1 = fM12*fQ11+fM22*fQ21+fM23*fQ31;
        float fV2 = fM13*fQ11+fM23*fQ21+fM33*fQ31;

        m_afDiag[1] = fQ11*fV0+fQ21*fV1+fQ31*fV2;

        // build column Q3 = Q1x(S*Q1)
        fQ13 = fQ21*fV2-fQ31*fV1;
        fQ23 = fQ31*fV0-fQ11*fV2;
        fQ33 = fQ11*fV1-fQ21*fV0;
        fLength = Math::sqrt(fQ13*fQ13+fQ23*fQ23+fQ33*fQ33);
        if ( fLength > 0.0 )
        {
            fInvLength = 1.0/fLength;
            fQ13 *= fInvLength;
            fQ23 *= fInvLength;
            fQ33 *= fInvLength;

            // build column Q2 = Q3xQ1
            fQ12 = fQ23*fQ31-fQ33*fQ21;
            fQ22 = fQ33*fQ11-fQ13*fQ31;
            fQ32 = fQ13*fQ21-fQ23*fQ11;

            fV0 = fQ12*fM11+fQ22*fM12+fQ32*fM13;
            fV1 = fQ12*fM12+fQ22*fM22+fQ32*fM23;
            fV2 = fQ12*fM13+fQ22*fM23+fQ32*fM33;
            m_afSubd[1] = fQ11*fV0+fQ21*fV1+fQ31*fV2;
            m_afDiag[2] = fQ12*fV0+fQ22*fV1+fQ32*fV2;
            m_afSubd[2] = fQ13*fV0+fQ23*fV1+fQ33*fV2;

            fV0 = fQ13*fM11+fQ23*fM12+fQ33*fM13;
            fV1 = fQ13*fM12+fQ23*fM22+fQ33*fM23;
            fV2 = fQ13*fM13+fQ23*fM23+fQ33*fM33;
            m_afDiag[3] = fQ13*fV0+fQ23*fV1+fQ33*fV2;
        }
        else
        {
            // S*Q1 parallel to Q1, choose any valid Q2 and Q3
            m_afSubd[1] = 0;

            fLength = fQ21*fQ21+fQ31*fQ31;
            if ( fLength > 0.0 )
            {
                fInvLength = 1.0/fLength;
                float fTmp = fQ11-1.0;
                fQ12 = -fQ21;
                fQ22 = 1.0+fTmp*fQ21*fQ21*fInvLength;
                fQ32 = fTmp*fQ21*fQ31*fInvLength;

                fQ13 = -fQ31;
                fQ23 = fQ32;
                fQ33 = 1.0+fTmp*fQ31*fQ31*fInvLength;

                fV0 = fQ12*fM11+fQ22*fM12+fQ32*fM13;
```

```
                fV1 = fQ12*fM12+fQ22*fM22+fQ32*fM23;
                fV2 = fQ12*fM13+fQ22*fM23+fQ32*fM33;
                m_afDiag[2] = fQ12*fV0+fQ22*fV1+fQ32*fV2;
                m_afSubd[2] = fQ13*fV0+fQ23*fV1+fQ33*fV2;

                fV0 = fQ13*fM11+fQ23*fM12+fQ33*fM13;
                fV1 = fQ13*fM12+fQ23*fM22+fQ33*fM23;
                fV2 = fQ13*fM13+fQ23*fM23+fQ33*fM33;
                m_afDiag[3] = fQ13*fV0+fQ23*fV1+fQ33*fV2;
            }
            else
            {
                // Q1 = (+-1,0,0)
                fQ12 = 0.0; fQ22 = 1.0; fQ32 = 0.0;
                fQ13 = 0.0; fQ23 = 0.0; fQ33 = 1.0;

                m_afDiag[2] = fM22;
                m_afDiag[3] = fM33;
                m_afSubd[2] = fM23;
            }
        }

        m_aafMat[1][1] = fQ11; m_aafMat[1][2] = fQ12; m_aafMat[1][3] = fQ13;
        m_aafMat[2][1] = fQ21; m_aafMat[2][2] = fQ22; m_aafMat[2][3] = fQ23;
        m_aafMat[3][1] = fQ31; m_aafMat[3][2] = fQ32; m_aafMat[3][3] = fQ33;
    }
    else
    {
        m_afDiag[1] = fM11;
        m_afSubd[0] = fM01;
        m_aafMat[1][1] = 1.0;
        m_aafMat[2][1] = 0.0;
        m_aafMat[3][1] = 0.0;

        if ( fM13 != 0.0 )
        {
            fLength = Math::sqrt(fM12*fM12+fM13*fM13);
            fInvLength = 1.0/fLength;
            fM12 *= fInvLength;
            fM13 *= fInvLength;
            float fQ = 2.0*fM12*fM23+fM13*(fM33-fM22);

            m_afDiag[2] = fM22+fM13*fQ;
            m_afDiag[3] = fM33-fM13*fQ;
            m_afSubd[1] = fLength;
            m_afSubd[2] = fM23-fM12*fQ;
            m_aafMat[1][2] = 0.0;
            m_aafMat[1][3] = 0.0;
            m_aafMat[2][2] = fM12;
            m_aafMat[2][3] = fM13;
            m_aafMat[3][2] = fM13;
            m_aafMat[3][3] = -fM12;
        }
        else
        {
```

```
            m_afDiag[2] = fM22;
            m_afDiag[3] = fM33;
            m_afSubd[1] = fM12;
            m_afSubd[2] = fM23;
            m_aafMat[1][2] = 0.0;
            m_aafMat[1][3] = 0.0;
            m_aafMat[2][2] = 1.0;
            m_aafMat[2][3] = 0.0;
            m_aafMat[3][2] = 0.0;
            m_aafMat[3][3] = 1.0;
        }
    }
}
```

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().

```
{
    int i0, i1, i2, i3;

    for (i0 = iSize-1, i3 = iSize-2; i0 >= 1; i0--, i3--)
    {
        float fH = 0.0, fScale = 0.0;

        if ( i3 > 0 )
        {
            for (i2 = 0; i2 <= i3; i2++)
                fScale += Math::abs(m_aafMat[i0][i2]);
            if ( fScale == 0 )
            {
                m_afSubd[i0] = m_aafMat[i0][i3];
            }
            else
            {
                float fInvScale = 1.0/fScale;
                for (i2 = 0; i2 <= i3; i2++)
                {
                    m_aafMat[i0][i2] *= fInvScale;
                    fH += m_aafMat[i0][i2]*m_aafMat[i0][i2];
                }
                float fF = m_aafMat[i0][i3];
                float fG = Math::sqrt(fH);
                if ( fF > 0.0 )
```

```
                    fG = -fG;
                m_afSubd[i0] = fScale*fG;
                fH -= fF*fG;
                m_aafMat[i0][i3] = fF-fG;
                fF = 0.0;
                float fInvH = 1.0/fH;
                for (i1 = 0; i1 <= i3; i1++)
                {
                    m_aafMat[i1][i0] = m_aafMat[i0][i1]*fInvH;
                    fG = 0.0;
                    for (i2 = 0; i2 <= i1; i2++)
                        fG += m_aafMat[i1][i2]*m_aafMat[i0][i2];
                    for (i2 = i1+1; i2 <= i3; i2++)
                        fG += m_aafMat[i2][i1]*m_aafMat[i0][i2];
                    m_afSubd[i1] = fG*fInvH;
                    fF += m_afSubd[i1]*m_aafMat[i0][i1];
                }
                float fHalfFdivH = 0.5*fF*fInvH;
                for (i1 = 0; i1 <= i3; i1++)
                {
                    fF = m_aafMat[i0][i1];
                    fG = m_afSubd[i1] - fHalfFdivH*fF;
                    m_afSubd[i1] = fG;
                    for (i2 = 0; i2 <= i1; i2++)
                    {
                        m_aafMat[i1][i2] -= fF*m_afSubd[i2] +
                            fG*m_aafMat[i0][i2];
                    }
                }
            }
        }
        else
        {
            m_afSubd[i0] = m_aafMat[i0][i3];
        }

        m_afDiag[i0] = fH;
    }

    m_afDiag[0] = m_afSubd[0] = 0;
    for (i0 = 0, i3 = -1; i0 <= iSize-1; i0++, i3++)
    {
        if ( m_afDiag[i0] )
        {
            for (i1 = 0; i1 <= i3; i1++)
            {
                float fSum = 0;
                for (i2 = 0; i2 <= i3; i2++)
                    fSum += m_aafMat[i0][i2]*m_aafMat[i2][i1];
                for (i2 = 0; i2 <= i3; i2++)
                    m_aafMat[i2][i1] -= fSum*m_aafMat[i2][i0];
            }
        }
        m_afDiag[i0] = m_aafMat[i0][i0];
        m_aafMat[i0][i0] = 1;
```

```
        for (i1 = 0; i1 <= i3; i1++)
            m_aafMat[i1][i0] = m_aafMat[i0][i1] = 0;
    }

    // re-ordering if Eigen::QLAlgorithm is used subsequently
    for (i0 = 1, i3 = 0; i0 < iSize; i0++, i3++)
        m_afSubd[i3] = m_afSubd[i0];
    m_afSubd[iSize-1] = 0;
}
```

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_iSize [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:

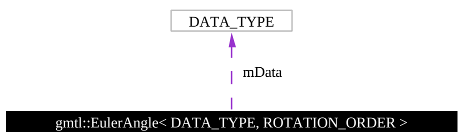
- Eigen.h

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 gmtl::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 RotatoinOrder 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 ith component in this EulerAngle.

Parameters:
i the zero-based index of the component to access.

Precondition:
0 <= i < 3

Returns:

a reference to the ith component

Definition at line 115 of file EulerAngle.h.

References gmtlASSERT, and gmtl::EulerAngle< DATA_TYPE, ROTATION_ORDER >::Size.

```
{
   gmtlASSERT( i < Size );
   return mData[i];
}
```

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.

```
{
   mData[0] = p0;
   mData[1] = p1;
   mData[2] = p2;
}
```

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

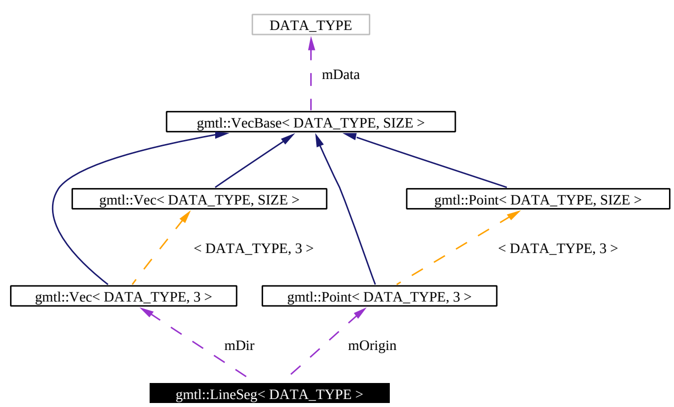
- EulerAngle.h

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 segement originating at O. Thus any point on the line segment can be described as

P(s) = O + Vs

where 0 <= s <= 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.

```
{}
```

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.

```
    : mOrigin( origin ), mDir( dir )
{}
```

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.

```
    : mOrigin( beg )
{
    mDir = end - beg;
}
```

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.

```
{
    mOrigin = lineseg.mOrigin;
    mDir = lineseg.mDir;
}
```

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.

```
{
    return mDir;
}
```

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().

```
{
   return length( dir );
}
```

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.

```
{
   return mOrigin;
}
```

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.

```
{
    mDir = dir;
}
```

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.

```
{
    mOrigin = origin;
}
```

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:

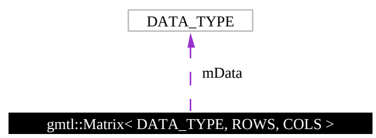
- LineSeg.h

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.

```
{
    Rows = ROWS, Cols = COLS
};
```

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.

```
{
   IDENTITY = 1,
   ORTHOGONAL = 2,
   ORTHONORMAL = 4,
   AFFINE = 8,
   FULL = 16,
   XFORM_ERROR = 32 // error bit
};
```

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()().

```
{
    for (unsigned int r = 0; r < ROWS; ++r)
    for (unsigned int c = 0; c < COLS; ++c)
       this->operator()( r, c ) = (DATA_TYPE)0.0;

    for (unsigned int x = 0; x < Math::Min( COLS, ROWS ); ++x)
       this->operator()( x, x ) = (DATA_TYPE)1.0;

    mState = FULL;
 };
```

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.

```
{
   gmtlASSERT( (row < ROWS) && (column < COLS) );
   return mData[column*ROWS + row];
}
```

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().

```
{
   gmtlASSERT( (row < ROWS) && (column < COLS) );
   return mData[column*ROWS + row];
}
```

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.

```
{
   gmtlASSERT( i < (ROWS*COLS) );
   return mData[i];
}
```

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.

```
{
   gmtlASSERT( i < (ROWS*COLS) );
   return mData[i];
}
```

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.

```
{
   for (unsigned int x = 0; x < ROWS * COLS; ++x)
      mData[x] = data[x];
   mState = FULL;
}
```

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, gmtlASSERT, gmtl::Matrix< DATA_TYPE, ROWS, COLS >::mData, and gmtl::Matrix< DATA_TYPE, ROWS, COLS >::mState.

```
{
   gmtlASSERT( ROWS == 4 && COLS == 4 );// could be compile time...
   mData[0]  = v00;
   mData[1]  = v10;
   mData[2]  = v20;
   mData[4]  = v01;
   mData[5]  = v11;
   mData[6]  = v21;
   mData[8]  = v02;
   mData[9]  = v12;
   mData[10] = v22;

   // right row
```

```
    mData[12] = v03;
    mData[13] = v13;
    mData[14] = v23;

    // bottom row
    mData[3]  = v30;
    mData[7]  = v31;
    mData[11] = v32;
    mData[15] = v33;
    mState = FULL;
 }
```

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.

```
 {
    gmtlASSERT( ROWS == 3 && COLS == 4 );// could be compile time...
    mData[0] = v00;
    mData[1] = v10;
    mData[2] = v20;
    mData[3] = v01;
    mData[4] = v11;
    mData[5] = v21;
    mData[6] = v02;
    mData[7] = v12;
    mData[8] = v22;

    // right row
    mData[9]  = v03;
    mData[10] = v13;
    mData[11] = v23;
    mState = FULL;
 }
```

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.

```
{
   gmtlASSERT( ROWS == 3 && COLS == 3 ); // could be at compile time...
   mData[0] = v00;
   mData[1] = v10;
   mData[2] = v20;

   mData[3] = v01;
   mData[4] = v11;
   mData[5] = v21;

   mData[6] = v02;
   mData[7] = v12;
   mData[8] = v22;
   mState = FULL;
}
```

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.

```
{
   gmtlASSERT( ROWS == 2 && COLS == 3 ); // could be at compile time...
   mData[0] = v00;
   mData[1] = v10;
   mData[2] = v01;
   mData[3] = v11;
   mData[4] = v02;
   mData[5] = v12;
   mState = FULL;
}
```

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().

```
{
   gmtlASSERT( ROWS == 2 && COLS == 2 ); // could be at compile time...
   mData[0] = v00;
   mData[1] = v10;
   mData[2] = v01;
   mData[3] = v11;
   mState = FULL;
}
```

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.

```
{
   mState |= XFORM_ERROR;
}
```

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()().

```
{
   for (unsigned int r = 0; r < ROWS; ++r)
   for (unsigned int c = 0; c < COLS; ++c)
      this->operator()( r, c ) = data[(r * COLS) + c];
   mState = FULL;
}
```

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 OOBox.h.

References ident().

```
53    { ident(); }
```

10.9.1.2 gmtl::OOBox::OOBox (OOBox & *box*) `[inline]`

Definition at line 106 of file OOBox.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 OOBox.h.

References mAxis.

```
144 {
145    return mAxis;
146 }
```

10.9.2.2 Vec3 ∗ gmtl::OOBox::axes () `[inline]`

Definition at line 138 of file OOBox.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 OOBox.h.

References mCenter.

Referenced by gmtl::TestIntersect(), and gmtl::TestIntersectOBB().

```
{
   return mCenter;
}
```

10.9.2.7 void gmtl::OOBox::getVerts (Point3 *verts*[8]) const [inline]

Definition at line 193 of file OOBox.h.

References mAxis, mCenter, and mHalfLen.

```
{
   Vec3 x_half_axis = mAxis[0]*mHalfLen[0];
   Vec3 y_half_axis = mAxis[1]*mHalfLen[1];
   Vec3 z_half_axis = mAxis[2]*mHalfLen[2];

   verts[0] = mCenter - x_half_axis - y_half_axis - z_half_axis;
   verts[1] = mCenter + x_half_axis - y_half_axis - z_half_axis;
   verts[2] = mCenter + x_half_axis + y_half_axis - z_half_axis;
   verts[3] = mCenter - x_half_axis + y_half_axis - z_half_axis;
   verts[4] = mCenter - x_half_axis - y_half_axis + z_half_axis;
   verts[5] = mCenter + x_half_axis - y_half_axis + z_half_axis;
   verts[6] = mCenter + x_half_axis + y_half_axis + z_half_axis;
   verts[7] = mCenter - x_half_axis + y_half_axis + z_half_axis;
}
```

10.9.2.8 const float & gmtl::OOBox::halfLen (int *i*) const [inline]

Definition at line 153 of file OOBox.h.

References mHalfLen.

```
{
   return mHalfLen[i];
}
```

10.9.2.9 float & gmtl::OOBox::halfLen (int *i*) [inline]

Definition at line 148 of file OOBox.h.

References mHalfLen.

```
{
   return mHalfLen[i];
}
```

10.9.2.10 const float ∗ gmtl::OOBox::halfLens () const [inline]

Definition at line 163 of file OOBox.h.

References mHalfLen.

```
{
   return mHalfLen;
}
```

10.9.2.11 float ∗ gmtl::OOBox::halfLens () [inline]

Definition at line 158 of file OOBox.h.

References mHalfLen.

Referenced by gmtl::TestIntersect(), and gmtl::TestIntersectOBB().

```
{
   return mHalfLen;
}
```

10.9.2.12 void gmtl::OOBox::ident () [inline]

Definition at line 86 of file OOBox.h.

References mAxis, mCenter, and mHalfLen.

Referenced by OOBox().

```
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]) &&
```

```
            (mHalfLen[2] == box.mHalfLen[2]));
}
```

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:

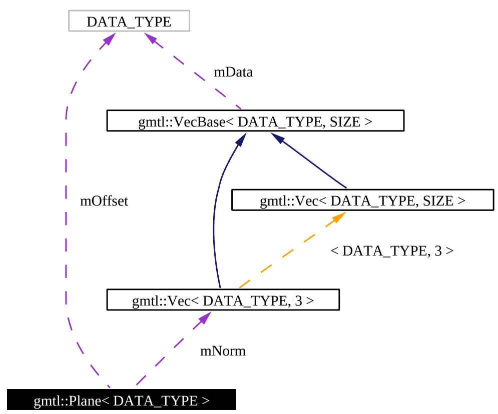
- OOBox.h

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 &dPlaneConst)

  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, 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 satify the equation dot(Pt,Normal) = 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

```
N dot P = -D
 |
```

```
    |-d-|
 __|___|-->N
    |   |
 *
```

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.

```
      : mOffset( 0 )
   {}
```

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().

```
   {
```

```
    Vec<DATA_TYPE, 3> vec12( pt2-pt1 );
    Vec<DATA_TYPE, 3> vec13( pt3-pt1 );

    mNorm = cross( vec12, vec13 );
    normalize( mNorm );

    mOffset = dot( static_cast< Vec<DATA_TYPE, 3> >(pt1), mNorm );  // Graphics Gems I: Pa
}
```

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.

```
    : mNorm( norm )
{
    mOffset = dot( static_cast< Vec<DATA_TYPE, 3> >(pt), norm );
}
```

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.

```
    : mNorm( norm ), mOffset( dPlaneConst )
{}
```

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.

```
    : mNorm( plane.mNorm ), mOffset( plane.mOffset )
{}
```

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.

```
{
    return mNorm;
}
```

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:
|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< DATA_TYPE >::mOffset.

```
{
   mOffset = offset;
}
```

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, dot(pt, mNorm) = mOffset.

Definition at line 175 of file Plane.h.

Referenced by gmtl::Plane< DATA_TYPE >::getNormal(), gmtl::Plane< DATA_TYPE >::Plane(), and gmtl::Plane< DATA_TYPE >::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 pt, dot(pt, mNorm) = mOffset.

Note that mOffset = -D (neg dist from the origin).

Definition at line 182 of file Plane.h.

Referenced by gmtl::Plane< DATA_TYPE >::getOffset(), gmtl::Plane< DATA_TYPE >::Plane(), and gmtl::Plane< DATA_TYPE >::setOffset().

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

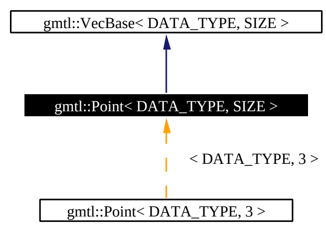
- Plane.h

10.11 gmtl::Point< DATA_TYPE, SIZE > Class Template Reference

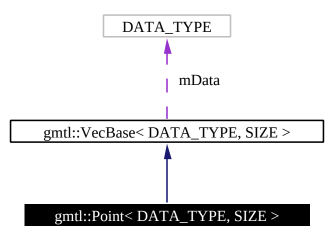
Point Use points when you need to represent a position.

```
#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.

```
      : BaseType(static_cast<BaseType>(rVec))
   {;}
```

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.

```
      : BaseType(rVec)
   {;}
```

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.

```
   : BaseType(val0, val1)
   {
      // @todo need compile time assert
      gmtlASSERT( SIZE == 2 && "out of bounds element access in Point" );
   }
```

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.

```
   : BaseType(val0, val1, val2)
   {
      // @todo need compile time assert
      gmtlASSERT( SIZE == 3 && "out of bounds element access in Point" );
   }
```

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.

```
    : BaseType(val0, val1, val2, val3)
    {
       // @todo need compile time assert
       gmtlASSERT( SIZE == 4 && "out of bounds element access in Point" );
    }
```

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

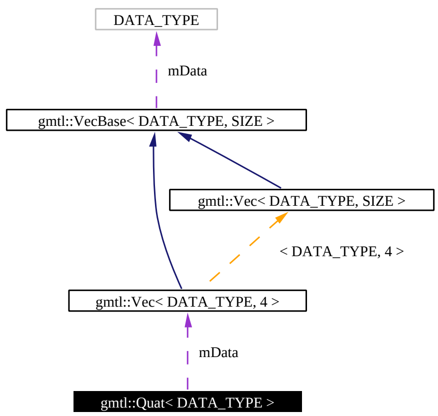
- Point.h

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 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) ∗ x, sin(theta/2) ∗ x, sin(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 gmtlASSERT, and gmtl::Quat< DATA_TYPE >::mData.

```
{
   gmtlASSERT( x >= 0 && x < 4 && "out of bounds error" );
   return mData[x];
}
```

10.12.4.5 template<typename DATA_TYPE> void gmtl::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 gmtl::Quat< DATA_TYPE >::mData, and gmtl::VecBase< DATA_TYPE, SIZE >::set().

```
{
   mData.set( x, y, z, w );
}
```

10.12.5 Member Data Documentation

10.12.5.1 template<typename DATA_TYPE> Vec<DATA_TYPE, 4> gmtl::Quat< DATA_TYPE >::mData

Definition at line 159 of file Quat.h.

Referenced by gmtl::Quat< DATA_TYPE >::get(), gmtl::Quat< DATA_TYPE >::getData(), gmtl::Quat< DATA_TYPE >::operator[ ](), and gmtl::Quat< DATA_TYPE >::set().

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

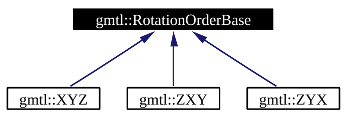
- Quat.h

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:

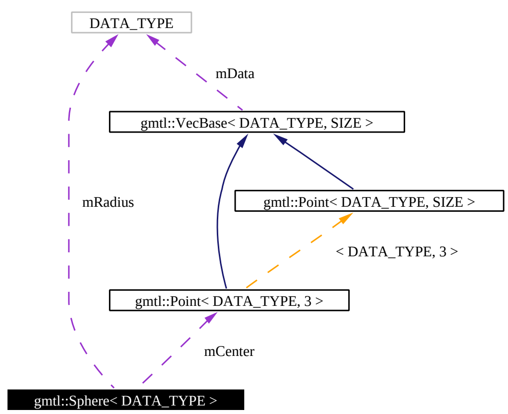
- Math.h

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.

```
    : mCenter( sphere.mCenter ), mRadius( sphere.mRadius )
{}
```

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.

```
{
    return mCenter;
}
```

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.

```
{
    return mRadius;
}
```

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.

```
{
   mCenter = center;
}
```

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.

```
{
   mRadius = radius;
}
```

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:

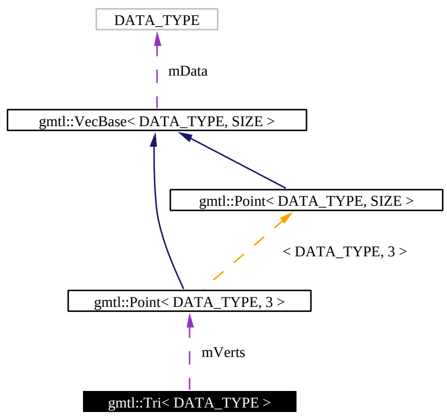
- Sphere.h

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 corresponsds 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 tri(s,t) = b+s∗e0+t∗e1 where 0 <= s <= 1, 0 <= t <= 1, and 0 <= s+t <= 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.

```
{
   mVerts[0] = p1;
   mVerts[1] = p2;
   mVerts[2] = p3;
}
```

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.

```
{
   mVerts[0] = tri[0];
   mVerts[1] = tri[1];
   mVerts[2] = tri[2];
}
```

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 corresponsds to the vector from vertex 2 to vertex 0.

Parameters:

idx the ordered edge index

Precondition:

0 <= idx <= 2

Returns:

a vector from vertex idx to vertex (idx+1)mod size

Definition at line 121 of file Tri.h.

References gmtlASSERT.

```
{
   gmtlASSERT( (0 <= idx) && (idx <= 2) );
   int idx2 = ( idx == 2 ) ? 0 : idx + 1;
   return (mVerts[idx2] - mVerts[idx]);
}
```

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.

```
{
   gmtlASSERT( (0 <= idx) && (idx <= 2) );
   return mVerts[idx];
}
```

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 <= idx <= 2

Returns:

the nth vertex

Definition at line 99 of file Tri.h.

References gmtlASSERT.

```
{
   gmtlASSERT( (0 <= idx) && (idx <= 2) );
   return mVerts[idx];
}
```

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:

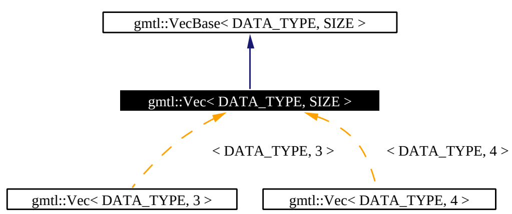
- Meta.h

10.17 gmtl::Vec< DATA_TYPE, SIZE > Class Template Reference

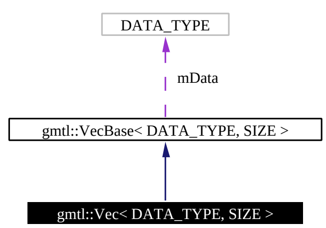
A representation of a vector with SIZE components using DATA_TYPE as the data type for each component.

```
#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.

```
{
   for (unsigned i = 0; i < SIZE; ++i)
      mData[i] = (DATA_TYPE)0;
}
```

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.

```
   : BaseType( static_cast<BaseType>( rVec ) )
{
}
```

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.

```
   : BaseType( rVec )
{
}
```

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.

```
   : BaseType(val0, val1)
    {
       // @todo compile time assert is needed here
       gmtlASSERT( SIZE == 2 && "out of bounds element access in Point" );
    }
```

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.

```
    : BaseType(val0, val1, val2)
    {
       // @todo compile time assert is needed here
       gmtlASSERT( SIZE == 3 && "out of bounds element access in Point" );
    }
```

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.

```
    : BaseType(val0, val1, val2, val3)
    {
       // @todo compile time assert is needed here
       gmtlASSERT( SIZE == 4 && "out of bounds element access in Point" );
    }
```

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

- Vec.h

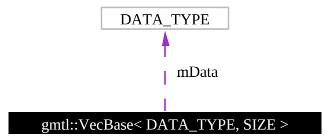
10.18 gmtl::VecBase< DATA_TYPE, SIZE > Class Template Reference

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 gmtlASSERT, and gmtl::VecBase< DATA_TYPE, SIZE >::mData.

```
{
   // @todo need compile time assert
   gmtlASSERT( SIZE == 2 && "out of bounds element access in VecBase" );
   mData[0] = val0;
   mData[1] = val1;
}
```

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.

```
{
   // @todo need compile time assert
   gmtlASSERT( SIZE == 3 && "out of bounds element access in VecBase" );
   mData[0] = val0;
   mData[1] = val1;
   mData[2] = val2;
}
```

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.

```
{
   // @todo need compile time assert
   gmtlASSERT( SIZE == 4 && "out of bounds element access in VecBase" );
   mData[0] = val0;
   mData[1] = val1;
   mData[2] = val2;
   mData[3] = val3;
}
```

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 < SIZE

Returns:

a reference to the ith component

Definition at line 113 of file VecBase.h.

```
{
   gmtlASSERT(i < SIZE);
   return mData[i];
}
```

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.

```
{
   gmtlASSERT( SIZE >= 4 && "out of bounds element access in VecBase" );
   mData[0] = val0;
   mData[1] = val1;
   mData[2] = val2;
   mData[3] = val3;
}
```

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.

```
{
   gmtlASSERT( SIZE >= 3 && "out of bounds element access in VecBase" );
   mData[0] = val0;
   mData[1] = val1;
   mData[2] = val2;
}
```

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.

```
{
   gmtlASSERT( SIZE >= 2 && "out of bounds element access in VecBase" );
   mData[0] = val0;
   mData[1] = val1;
}
```

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.

```
{
   gmtlASSERT( SIZE >= 1 && "out of bounds element access in VecBase" );
   mData[0] = val0;
}
```

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:

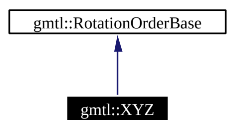
- VecBase.h

10.19 gmtl::XYZ Struct Reference

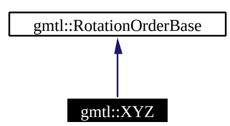
XYZ Rotation order.

#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:

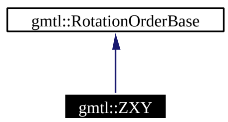
- Math.h

10.20 gmtl::ZXY Struct Reference

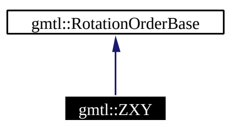
ZXY Rotation order.

`#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:

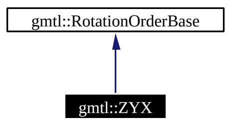
- Math.h

10.21 gmtl::ZYX Struct Reference

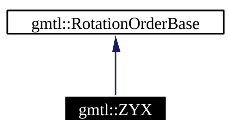
ZYX Rotation order.

`#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

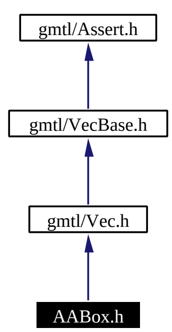
Chapter 11

GenericMathTemplateLibrary File Documentation

11.1 AABox.h File Reference

`#include <gmtl/Vec.h>`

Include dependency graph for AABox.h:

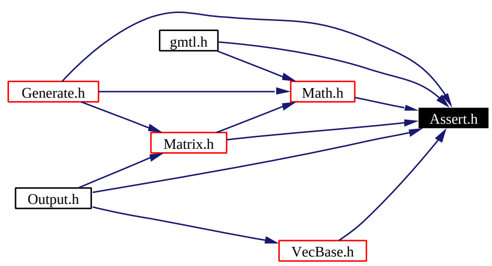


Namespaces

- namespace gmtl

11.2 Assert.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define gmtlASSERT(val) ((void)0)

11.2.1 Define Documentation

11.2.1.1 #define gmtlASSERT(val) ((void)0)

Definition at line 9 of file Assert.h.

Referenced by gmtl::Tri< DATA_TYPE >::edge(), gmtl::findNearestPt(), gmtl::Math::isEqual(), gmtl::isEqual(), gmtl::isOnVolume(), gmtl::makeVolume(), gmtl::meanTangent(), gmtl::Matrix< DATA_TYPE, ROWS, COLS >::operator()(), gmtl::VecBase< DATA_TYPE, 4 >::operator[ ](), gmtl::Tri< DATA_TYPE >::operator[ ](), gmtl::Quat< DATA_TYPE >::operator[ ](), gmtl::Matrix< DATA_TYPE, ROWS, COLS >::operator[ ](), gmtl::EulerAngle< DATA_TYPE, ROTATION_ORDER >::operator[ ](), gmtl::Point< DATA_TYPE, 3 >::Point(), gmtl::VecBase< DATA_TYPE, SIZE >::set(), gmtl::Matrix< DATA_TYPE, ROWS, COLS >::set(), gmtl::set(), gmtl::setAxes(), gmtl::setDirCos(), gmtl::setRot(), gmtl::setScale(), gmtl::setTrans(), gmtl::squad(), gmtl::Vec< DATA_TYPE, 4 >::Vec(), gmtl::VecBase< DATA_TYPE, SIZE >::VecBase(), and gmtl::xform().

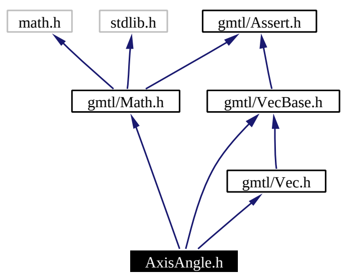
11.3 AxisAngle.h File Reference

`#include <gmtl/Math.h>`

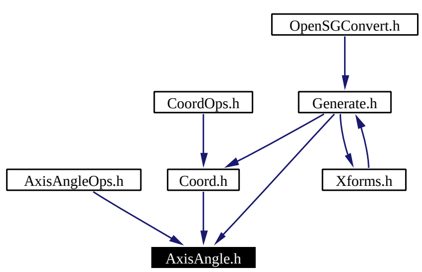
`#include <gmtl/VecBase.h>`

`#include <gmtl/Vec.h>`

Include dependency graph for AxisAngle.h:



This graph shows which files directly or indirectly include this file:



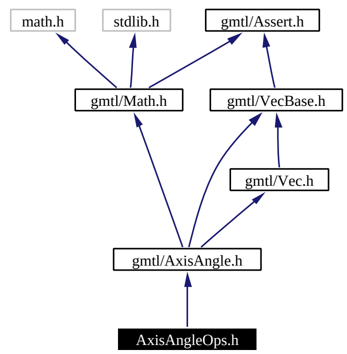
Namespaces

- namespace gmtl

11.4 AxisAngleOps.h File Reference

`#include "gmtl/AxisAngle.h"`

Include dependency graph for AxisAngleOps.h:



Namespaces

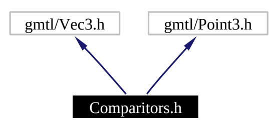
- namespace gmtl

11.5 Comparitors.h File Reference

`#include <gmtl/Vec3.h>`

`#include <gmtl/Point3.h>`

Include dependency graph for Comparitors.h:



Namespaces

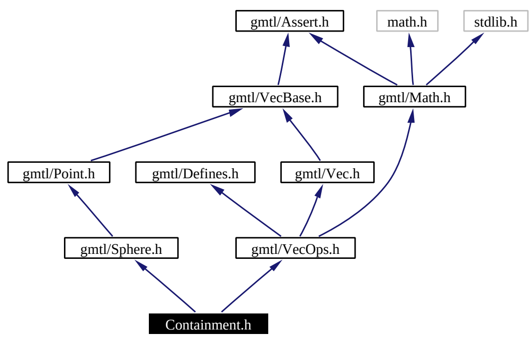
- namespace gmtl

11.6 Containment.h File Reference

`#include <gmtl/Sphere.h>`

`#include <gmtl/VecOps.h>`

Include dependency graph for Containment.h:



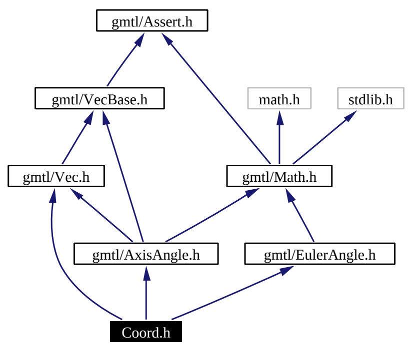
Namespaces

- namespace gmtl

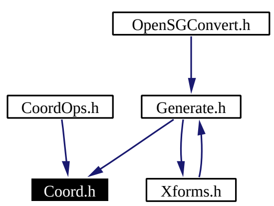
11.7 Coord.h File Reference

```
#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:



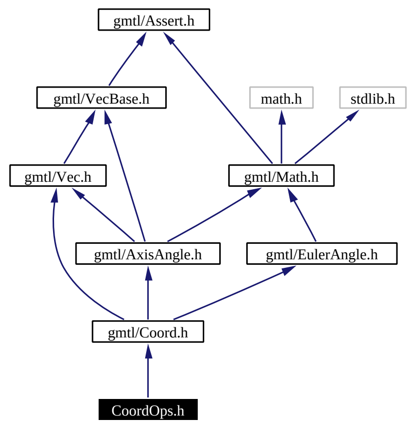
Namespaces

- namespace gmtl

11.8 CoordOps.h File Reference

`#include <gmtl/Coord.h>`

Include dependency graph for CoordOps.h:

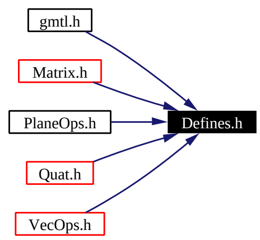


Namespaces

- 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

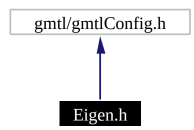
11.9.1.1 #define GMTL_NEAR(x, y, eps) (gmtl::Math::abs((x)-(y))<(eps))

Definition at line 77 of file Defines.h.

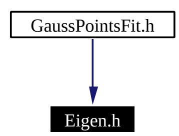
11.10 Eigen.h File Reference

`#include <gmtl/gmtlConfig.h>`

Include dependency graph for Eigen.h:



This graph shows which files directly or indirectly include this file:



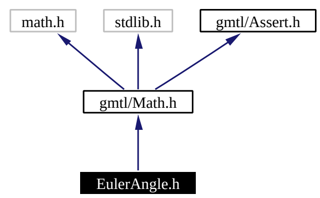
Namespaces

- namespace gmtl

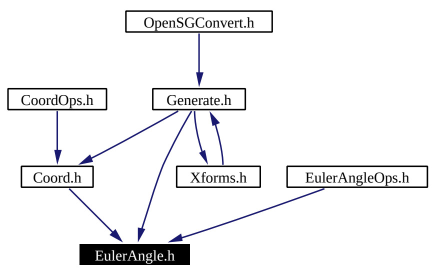
11.11 EulerAngle.h File Reference

`#include <gmtl/Math.h>`

Include dependency graph for EulerAngle.h:



This graph shows which files directly or indirectly include this file:



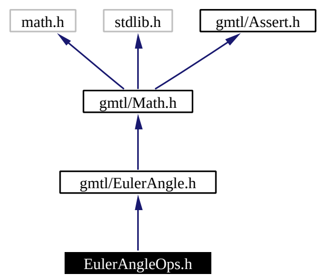
Namespaces

- namespace gmtl

11.12 EulerAngleOps.h File Reference

`#include "gmtl/EulerAngle.h"`

Include dependency graph for EulerAngleOps.h:



Namespaces

- namespace gmtl

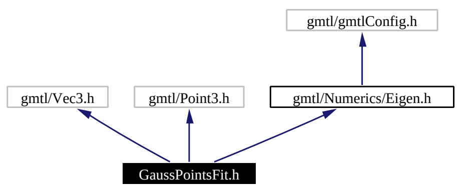
11.13 GaussPointsFit.h File Reference

`#include <gmtl/Vec3.h>`

`#include <gmtl/Point3.h>`

`#include <gmtl/Numerics/Eigen.h>`

Include dependency graph for GaussPointsFit.h:



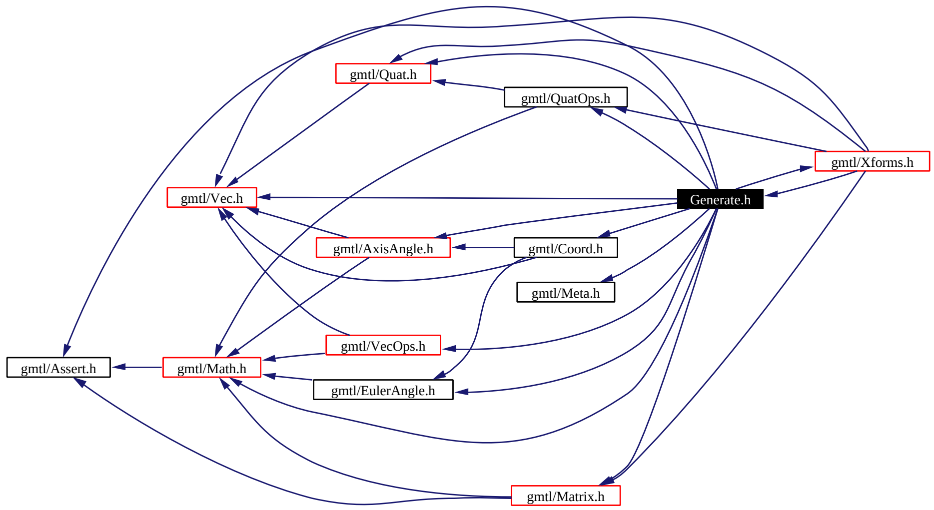
Namespaces

- namespace gmtl

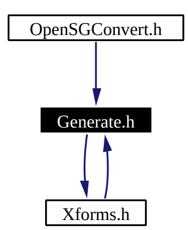
11.14 Generate.h File Reference

```
#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

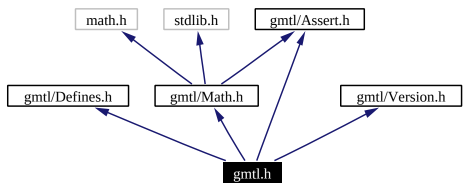
this file for documentation purposes only (for doxygen generation).

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>
```

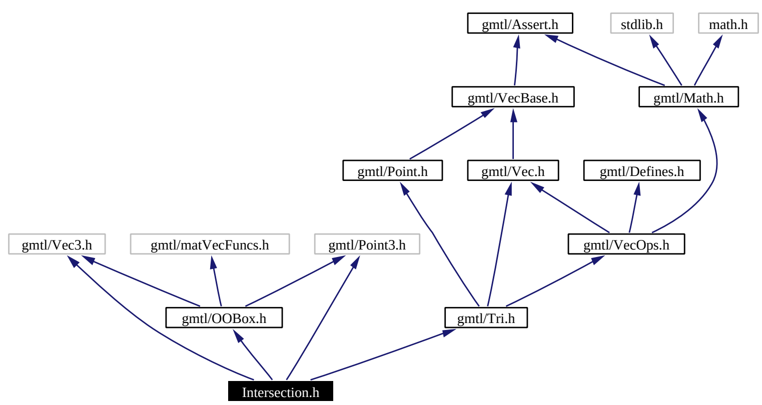
Include dependency graph for gmtl.h:



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:



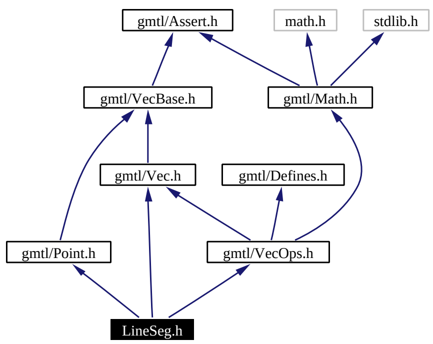
Namespaces

- namespace gmtl

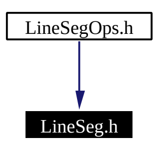
11.18 LineSeg.h File Reference

```
#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:



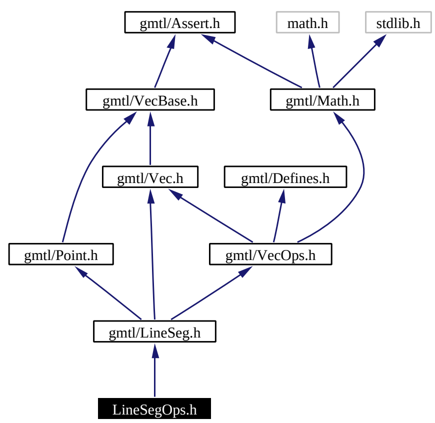
Namespaces

- namespace gmtl

11.19 LineSegOps.h File Reference

`#include <gmtl/LineSeg.h>`

Include dependency graph for LineSegOps.h:



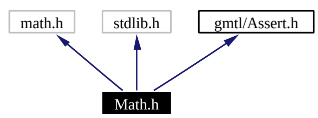
Namespaces

- namespace gmtl

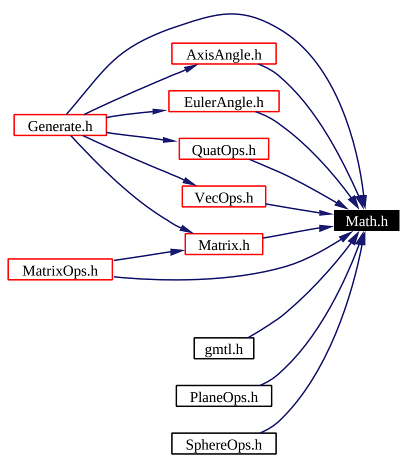
11.20 Math.h File Reference

```
#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

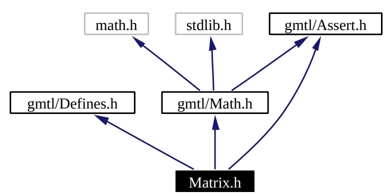
11.21 Matrix.h File Reference

`#include <gmtl/Defines.h>`

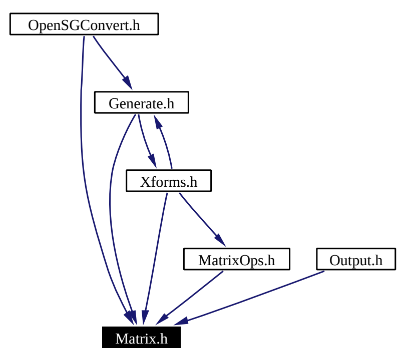
`#include <gmtl/Math.h>`

`#include <gmtl/Assert.h>`

Include dependency graph for Matrix.h:



This graph shows which files directly or indirectly include this file:



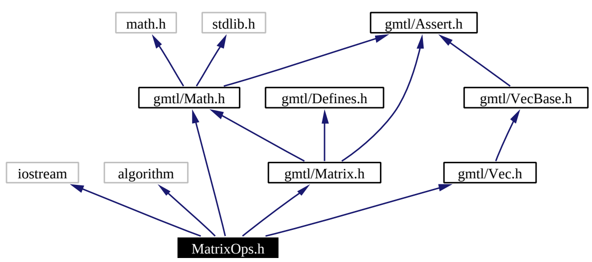
Namespaces

- namespace gmtl

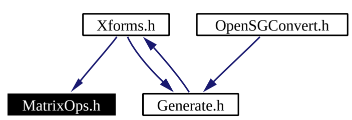
11.22 MatrixOps.h File Reference

```
#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:

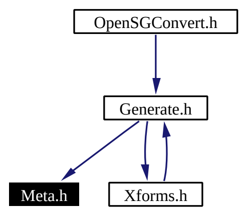


Namespaces

- namespace gmtl

11.23 Meta.h File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace gmtl

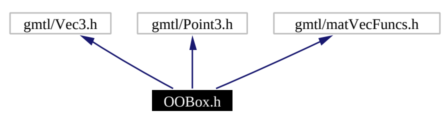
11.24 OOBox.h File Reference

`#include <gmtl/Vec3.h>`

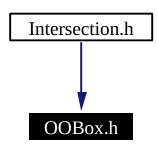
`#include <gmtl/Point3.h>`

`#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

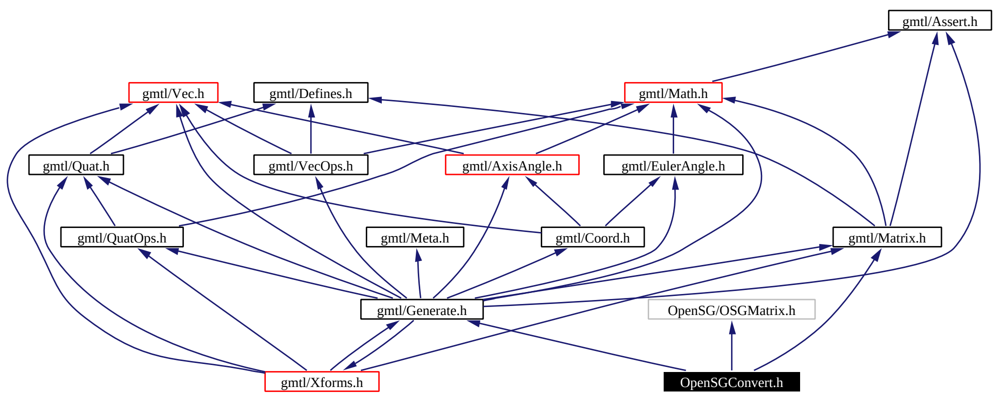
GMTL to OpenSG conversion functions.

`#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 gtml 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>`

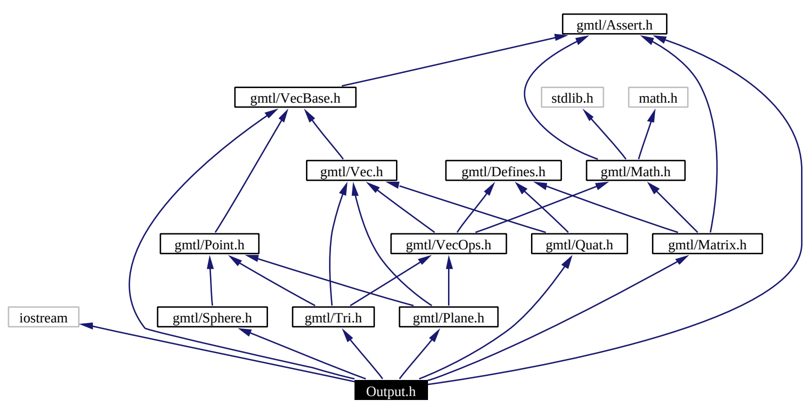
`#include <gmtl/Quat.h>`

`#include <gmtl/Tri.h>`

`#include <gmtl/Plane.h>`

`#include <gmtl/Sphere.h>`

Include dependency graph for Output.h:



Namespaces

- namespace gmtl

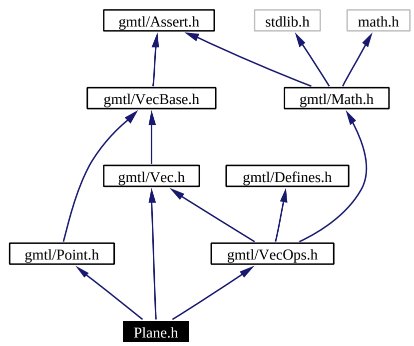
11.27 Plane.h File Reference

`#include <gmtl/Vec.h>`

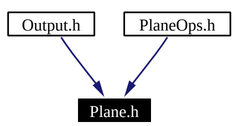
`#include <gmtl/Point.h>`

`#include <gmtl/VecOps.h>`

Include dependency graph for Plane.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace gmtl

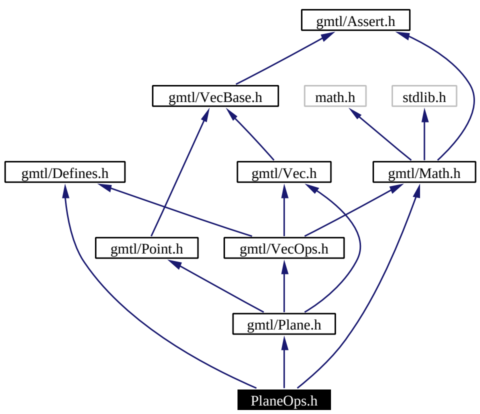
11.28 PlaneOps.h File Reference

`#include <gmtl/Defines.h>`

`#include <gmtl/Plane.h>`

`#include <gmtl/Math.h>`

Include dependency graph for PlaneOps.h:



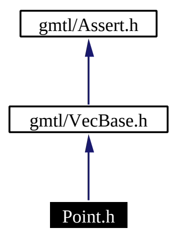
Namespaces

- namespace gmtl

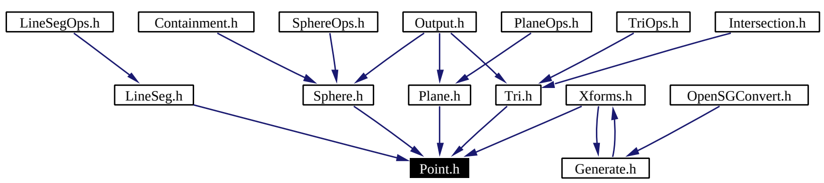
11.29 Point.h File Reference

`#include <gmtl/VecBase.h>`

Include dependency graph for Point.h:



This graph shows which files directly or indirectly include this file:



Namespaces

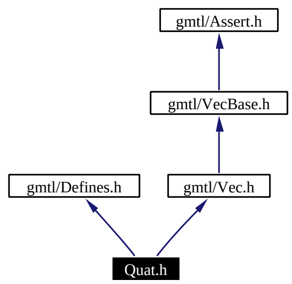
- namespace gmtl

11.30 Quat.h File Reference

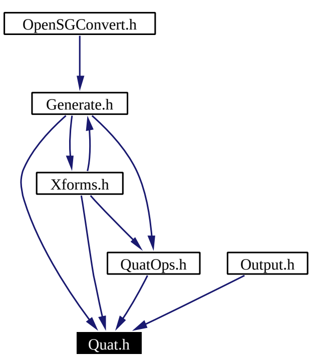
`#include <gmtl/Defines.h>`

`#include <gmtl/Vec.h>`

Include dependency graph for Quat.h:



This graph shows which files directly or indirectly include this file:



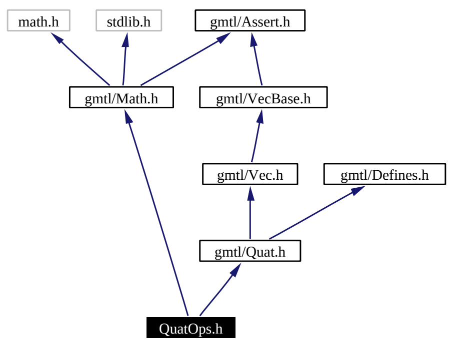
Namespaces

- namespace gmtl

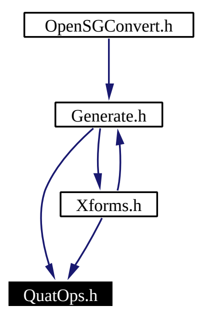
11.31 QuatOps.h File Reference

```
#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:



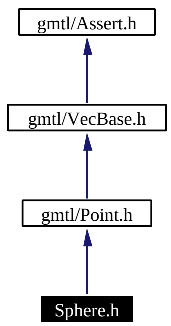
Namespaces

- namespace gmtl

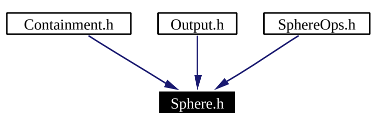
11.32 Sphere.h File Reference

`#include <gmtl/Point.h>`

Include dependency graph for Sphere.h:



This graph shows which files directly or indirectly include this file:



Namespaces

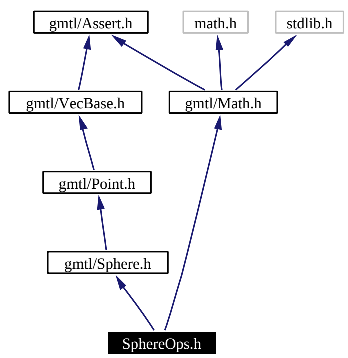
- namespace gmtl

11.33 SphereOps.h File Reference

`#include <gmtl/Sphere.h>`

`#include <gmtl/Math.h>`

Include dependency graph for SphereOps.h:



Namespaces

- namespace gmtl

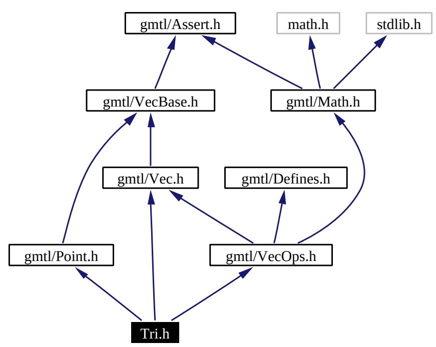
11.34 Tri.h File Reference

`#include <gmtl/Point.h>`

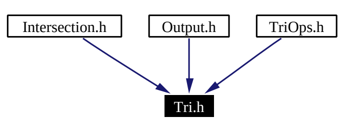
`#include <gmtl/Vec.h>`

`#include <gmtl/VecOps.h>`

Include dependency graph for Tri.h:



This graph shows which files directly or indirectly include this file:



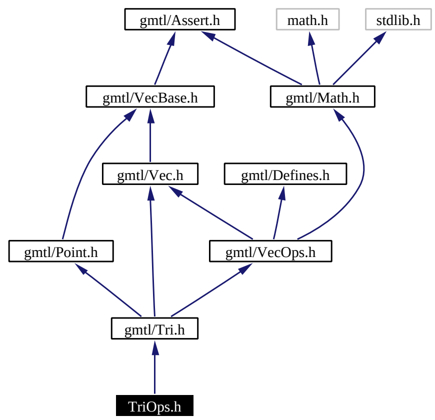
Namespaces

- namespace gmtl

11.35 TriOps.h File Reference

`#include <gmtl/Tri.h>`

Include dependency graph for TriOps.h:



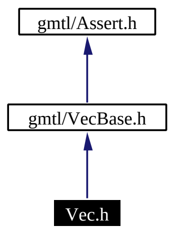
Namespaces

- namespace gmtl

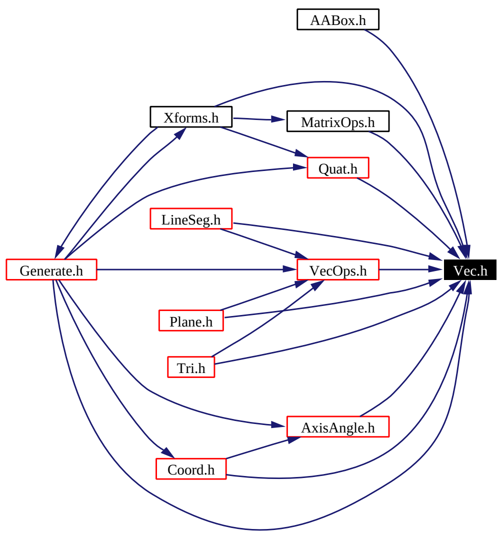
11.36 Vec.h File Reference

`#include <gmtl/VecBase.h>`

Include dependency graph for Vec.h:



This graph shows which files directly or indirectly include this file:



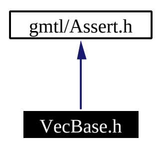
Namespaces

- namespace gmtl

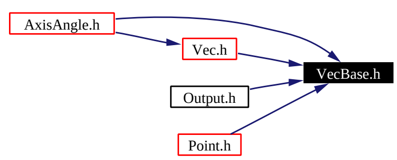
11.37 VecBase.h File Reference

`#include "gmtl/Assert.h"`

Include dependency graph for VecBase.h:



This graph shows which files directly or indirectly include this file:



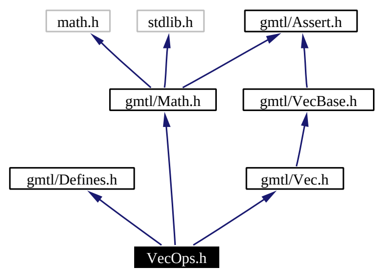
Namespaces

- namespace gmtl

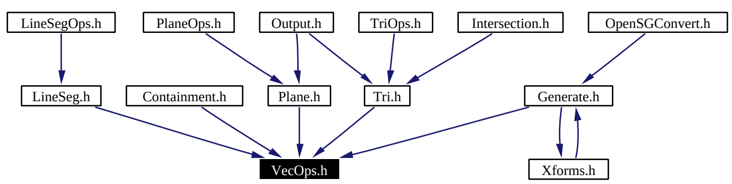
11.38 VecOps.h File Reference

```
#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:

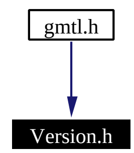


Namespaces

- 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 \
   )
```

The 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.

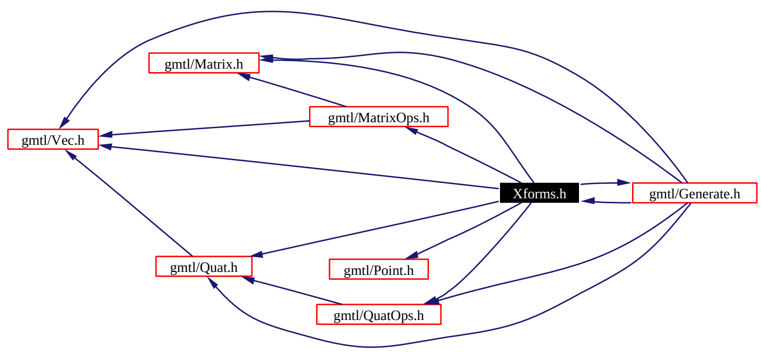
11.39.1.16 #define GMTL_ZEROFILL(a) 0 ## a

Definition at line 80 of file Version.h.

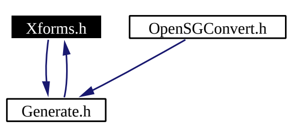
11.40 Xforms.h File Reference

```
#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 EulerXYZ == EulerZYX works, when it shouldn't even compile...

Member gmtl::Matrix::operator()(const unsigned row, const unsigned column) metaprog

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
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
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
needs mp!!

Member gmtl::Matrix::set(DATA_TYPE v00, DATA_TYPE v01, DATA_TYPE v02, DATA_TYPE v10, DAT
needs mp!!

Member gmtl::Matrix::set(DATA_TYPE v00, DATA_TYPE v01, DATA_TYPE v10, DATA_TYPE v11)
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, C
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