GenericMathTemplateLibrary
0.6.1

Generated by Doxygen 1.7.1

Sun Sep 19 2010 14:32:20

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.

Please see the GMTL FAQ for more information.

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

Todo List

Member gmtl::Coord< POS_TYPE, ROT_TYPE >::pos() what about having a pos, and a const_pos naming convention?

what about having a rot, and a const_rot naming convention?

Member gmtl::lerp(VecBase< DATA_TYPE, SIZE > &result, const DATA_TYPE &lerpVal, const VecBase< DATA_T
metaprogramming...

Member gmtl::Matrix< DATA_TYPE, ROWS, COLS >::Matrix() mp

mp

Set initial state to IDENTITY and test other stuff

Member gmtl::Matrix< DATA_TYPE, ROWS, COLS >::set(DATA_TYPE v00, DATA_TYPE v01, DATA_TYPE v02, D
needs mp!!

Member gmtl::Matrix< DATA_TYPE, ROWS, COLS >::set(const DATA_TYPE ∗data)
implement this!

mp

Member gmtl::Matrix< DATA_TYPE, ROWS, COLS >::set(DATA_TYPE v00, DATA_TYPE v01, DATA_TYPE v02, D
needs mp!!

Member gmtl::Matrix< DATA_TYPE, ROWS, COLS >::set(DATA_TYPE v00, DATA_TYPE v01, DATA_TYPE v10, D
needs mp!!

Member gmtl::Matrix< DATA_TYPE, ROWS, COLS >::set(DATA_TYPE v00, DATA_TYPE v01, DATA_
needs mp!! currently no way for a 4x3, ...

Member gmtl::Matrix< DATA_TYPE, ROWS, COLS >::set(DATA_TYPE v00, DATA_TYPE v01, DATA_
needs mp!! currently no way for a 4x3, ...

Member gmtl::Matrix< DATA_TYPE, ROWS, COLS >::setTranspose(const DATA_TYPE *data)
metaprog

Member gmtl::operator*(const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)
metaprogramming on quat operator*()

Member gmtl::set(Matrix< DATA_TYPE, ROWS, COLS > &mat, const Quat< DATA_TYPE > &q)
Implement using setRot

Member gmtl::xform(Point< DATA_TYPE, PNT_SIZE > &result, const Matrix< DATA_TYPE, ROWS, C
we need a PointOps.h operator*=(scalar) function

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 5

Class Index

5.1 Class Hierarchy

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

Chapter 6

Class Index

6.1 Class List

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

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

Chapter 8

Module Documentation

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

Constant Static Global Flags.

Classes

- struct gmtl::RotationOrderBase

    Base class for Rotation orders.

- struct gmtl::XYZ

    XYZ Rotation order.

- struct gmtl::ZYX

    ZYX Rotation order.

- struct gmtl::ZXY

    ZXY Rotation order.

Enumerations

- enum gmtl::VectorIndex { gmtl::Xelt = 0, gmtl::Yelt = 1, gmtl::Zelt = 2, gmtl::Welt = 3 }

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

- enum gmtl::PlaneSide { gmtl::ON_PLANE, gmtl::POS_SIDE, gmtl::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 same side as the surface normal. NEG_SIDE means the point lies on the opposite side as the ssurface normal.

Enumerator:

ON_PLANE

POS_SIDE

NEG_SIDE

Definition at line 32 of file Defines.h.

```
{
   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;
```

Enumerator:

Xelt

Yelt

Zelt

Welt

Definition at line 23 of file Defines.h.

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

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

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

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

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

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

Classes

- class gmtl::AABox< DATA_TYPE >

  Describes an axially aligned box in 3D space.

- class gmtl::AxisAngle< DATA_TYPE >

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

- class gmtl::Coord< POS_TYPE, ROT_TYPE >

  coord is a position/rotation pair.

- class gmtl::EulerAngle< DATA_TYPE, ROTATION_ORDER >

  EulerAngle: Represents a group of euler angles.

- class gmtl::Frustum< DATA_TYPE >

  This class defines a View Frustum Volume as a set of 6 planes.

- class gmtl::Matrix< DATA_TYPE, ROWS, COLS >

  State tracked NxM dimensional Matrix (ordered in memory by Column).

- class gmtl::Plane< DATA_TYPE >

  Plane: Defines a geometrical plane.

- class gmtl::Point< DATA_TYPE, SIZE >

  Point Use points when you need to represent a position.

- class gmtl::Quat< DATA_TYPE >

  Quat: Class to encapsulate quaternion behaviors.

- class gmtl::Sphere< DATA_TYPE >

  Describes a sphere in 3D space by its center point and its radius.

- class gmtl::Tri< DATA_TYPE >

  This class defines a triangle as a set of 3 points order in CCW fashion.

- class gmtl::Vec< DATA_TYPE, SIZE >

  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.

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

8.5 Spacial Transformers: xform(...), operator(...).

Transform points and vectors by Matrices and Quaternions.

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.

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

Tests for equality between GMTL data types.

Tests for equality between GMTL data types.

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

Make get and set functions for all math types in gmtl.

Make get and set functions for all math types in gmtl.

8.8 Interpolation: lerp(...), slerp(...)

Functions to interpolate between two values.

Functions to interpolate between two values.

8.9 Output Stream Methods: operator<<(...).

Output GMTL data types to an ostream.

Output GMTL data types to an ostream. std::ostream& operator<< methods...

8.10 Template Metaprogramming Utilities

Classes

- struct gmtl::Type2Type< T >

  A lightweight identifier you can pass to overloaded functions to typefy them.

8.11 Template Metaprogramming Utilities (Helpers)

- template<class T >
  void gmtl::ignore_unused_variable_warning (const T &)

8.11.1 Function Documentation

8.11.1.1 template< class T > void gmtl::ignore_unused_variable_warning (const T &) [inline]

Definition at line 56 of file Meta.h.

```
{ }
```

Chapter 9

Namespace Documentation

9.1 gmtl Namespace Reference

Meta programming classes.

Namespaces

- namespace helpers
- namespace Math
- namespace meta
- namespace output

Classes

- 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 EulerAngle

  EulerAngle: Represents a group of euler angles.

- class Frustum

  This class defines a View Frustum Volume as a set of 6 planes.

- class LineSeg

  Describes a line segment.

- struct RotationOrderBase

  Base class for Rotation orders.

- struct XYZ

  XYZ Rotation order.

- struct ZYX

  ZYX Rotation order.

- struct ZXY

  ZXY Rotation order.

- class Matrix

  State tracked NxM dimensional Matrix (ordered in memory by Column).

- class Eigen
- class OOBox
- class ParametricCurve

  A base representation of a parametric curve with SIZE component using DATA_TYPE as the data type, ORDER as the order for each component.

- class LinearCurve

  A representation of a line with order set to 2.

- class QuadraticCurve

  A representation of a quadratic curve with order set to 3.

- class CubicCurve

  A representation of a cubic curve with order set to 4.

- 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 Ray

  Describes a ray.

- 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.

- struct CompileTimeError< true >
- 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.

- class VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >

  Specialized version of VecBase that is actually used for all user interaction with a traditional vector.

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 Coord< Vec3f, EulerAngleXYZf > Coord3fXYZ

    3 elt types

- typedef Coord< Vec3f, EulerAngleZYXf > Coord3fZYX
- typedef Coord< Vec3f, EulerAngleZXYf > Coord3fZXY
- typedef Coord< Vec3d, EulerAngleXYZd > Coord3dXYZ
- typedef Coord< Vec3d, EulerAngleZYXd > Coord3dZYX
- typedef Coord< Vec3d, EulerAngleZXYd > Coord3dZXY
- typedef Coord< Vec4f, EulerAngleXYZf > Coord4fXYZ

    4 elt types

- typedef Coord< Vec4f, EulerAngleZYXf > Coord4fZYX
- typedef Coord< Vec4f, EulerAngleZXYf > Coord4fZXY
- typedef Coord< Vec4d, EulerAngleXYZd > Coord4dXYZ
- typedef Coord< Vec4d, EulerAngleZYXd > Coord4dZYX
- typedef Coord< Vec4d, EulerAngleZXYd > Coord4dZXY
- typedef Coord< Vec3f, Quatf > Coord3fQuat

    3 elt types

- typedef Coord< Vec3d, Quatd > Coord3dQuat
- typedef Coord< Vec4f, Quatf > Coord4fQuat

    4 elt types

- typedef Coord< Vec4d, Quatd > Coord4dQuat
- typedef Coord< Vec3f, AxisAnglef > Coord3fAxisAngle

    3 elt types

- typedef Coord< Vec3d, AxisAngled > Coord3dAxisAngle
- typedef Coord< Vec4f, AxisAnglef > Coord4fAxisAngle

    4 elt types

- typedef Coord< Vec4d, AxisAngled > Coord4dAxisAngle
- 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 Frustum< float > Frustumf
- typedef Frustum< double > Frustumd
- 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 LinearCurve< float, 1 > LinearCurve1f
- typedef LinearCurve< float, 2 > LinearCurve2f
- typedef LinearCurve< float, 3 > LinearCurve3f
- typedef LinearCurve< double, 1 > LinearCurve1d
- typedef LinearCurve< double, 2 > LinearCurve2d
- typedef LinearCurve< double, 3 > LinearCurve3d
- typedef QuadraticCurve< float, 1 > QuadraticCurve1f
- typedef QuadraticCurve< float, 2 > QuadraticCurve2f
- typedef QuadraticCurve< float, 3 > QuadraticCurve3f
- typedef QuadraticCurve< double, 1 > QuadraticCurve1d
- typedef QuadraticCurve< double, 2 > QuadraticCurve2d
- typedef QuadraticCurve< double, 3 > QuadraticCurve3d
- typedef CubicCurve< float, 1 > CubicCurve1f
- typedef CubicCurve< float, 2 > CubicCurve2f
- typedef CubicCurve< float, 3 > CubicCurve3f
- typedef CubicCurve< double, 1 > CubicCurve1d
- typedef CubicCurve< double, 2 > CubicCurve2d
- typedef CubicCurve< double, 3 > CubicCurve3d
- typedef Plane< float > Planef
- typedef Plane< double > Planed
- typedef Point< int, 2 > Point2i
- typedef Point< float, 2 > Point2f
- typedef Point< double, 2 > Point2d
- typedef Point< int, 3 > Point3i
- typedef Point< float, 3 > Point3f
- typedef Point< double, 3 > Point3d

- typedef Point< int, 4 > Point4i
- typedef Point< float, 4 > Point4f
- typedef Point< double, 4 > Point4d
- typedef Quat< float > Quatf
- typedef Quat< double > Quatd
- typedef Ray< float > Rayf
- typedef Ray< double > Rayd
- typedef Sphere< float > Spheref
- typedef Sphere< double > Sphered
- typedef Tri< float > Trif
- typedef Tri< double > Trid
- typedef Tri< int > Trii
- typedef Vec< int, 2 > Vec2i
- typedef Vec< float, 2 > Vec2f
- typedef Vec< double, 2 > Vec2d
- typedef Vec< int, 3 > Vec3i
- typedef Vec< float, 3 > Vec3f
- typedef Vec< double, 3 > Vec3d
- typedef Vec< int, 4 > Vec4i
- 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)

  Modifies the given sphere to tightly enclose all spheres in the given std::vector.

- 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.

- template<class DATA_TYPE >
  bool isInVolume (const AABox< DATA_TYPE > &container, const Point< DATA_TYPE, 3 > &pt)

  Tests if the given point is inside (or on) the surface of the given AABox volume.

- template<class DATA_TYPE >
  bool isInVolumeExclusive (const AABox< DATA_TYPE > &container, const Point< DATA_TYPE, 3 > &pt)

  Tests if the given point is inside (not on) the surface of the given AABox volume.

- template<class DATA_TYPE >
  bool isInVolume (const AABox< DATA_TYPE > &container, const AABox< DATA_TYPE > &box)

Tests if the given AABox is completely inside or on the surface of the given AABox container.

- template< class DATA_TYPE >
void extendVolume (AABox< DATA_TYPE > &container, const Point< DATA_TYPE, 3 > &pt)

  Modifies the existing AABox to tightly enclose itself and the given point.

- template< class DATA_TYPE >
void extendVolume (AABox< DATA_TYPE > &container, const AABox< DATA_TYPE > &box)

  Modifies the container to tightly enclose itself and the given AABox.

- template< class DATA_TYPE >
void makeVolume (AABox< DATA_TYPE > &box, const Sphere< DATA_-TYPE > &sph)

  Creates an AABox that tightly encloses the given Sphere.

- template< typename T >
bool isInVolume (const Frustum< T > &f, const Point< T, 3 > &p, unsigned int &idx)
- template< typename T >
bool isInVolume (const Frustum< T > &f, const Sphere< T > &s)
- template< typename T >
bool isInVolume (const Frustum< T > &f, const AABox< T > &box)
- template< typename T >
bool isInVolume (const Frustum< T > &f, const Tri< T > &tri)
- 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 &osgMat)

  Converts an OpenSG matrix to a gmtl::Matrix.

- OSG::Matrix & set (OSG::Matrix &osgMat, const Matrix44f &mat)

  Converts a GMTL matrix to an OpenSG matrix.

- 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])
- template<class DATA_TYPE >
  void normalize (Frustum< DATA_TYPE > &f)
- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  void setRow (Vec< DATA_TYPE, COLS > &dest, const Matrix< DATA_-TYPE, ROWS, COLS > &src, unsigned row)

  Accesses a particular row in the matrix by copying the values in the row into the given vector.

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  Vec< DATA_TYPE, COLS > makeRow (const Matrix< DATA_TYPE, ROWS, COLS > &src, unsigned row)

  Accesses a particular row in the matrix by creating a new vector containing the values in the given matrix.

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  void setColumn (Vec< DATA_TYPE, ROWS > &dest, const Matrix< DATA_-TYPE, ROWS, COLS > &src, unsigned col)

  Accesses a particular column in the matrix by copying the values in the column into the given vector.

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  Vec< DATA_TYPE, ROWS > makeColumn (const Matrix< DATA_TYPE, ROWS, COLS > &src, unsigned col)

  Accesses a particular column in the matrix by creating a new vector containing the values in the given matrix.

- template<class DATA_TYPE >
  bool intersect (const AABox< DATA_TYPE > &box1, const AABox< DATA_-TYPE > &box2)

  Tests if the given AABoxes intersect with each other.

- template<class DATA_TYPE >
  bool intersect (const AABox< DATA_TYPE > &box, const Point< DATA_-TYPE, 3 > &point)

  Tests if the given AABox and point intersect with each other.

- template<class DATA_TYPE >
  bool intersect (const AABox< DATA_TYPE > &box1, const Vec< DATA_-TYPE, 3 > &path1, const AABox< DATA_TYPE > &box2, const Vec<

DATA_TYPE, 3 > &path2, DATA_TYPE &firstContact, DATA_TYPE &secondContact)

Tests if the given AABoxes intersect if moved along the given paths.

- template<class DATA_TYPE >
bool intersectAABoxRay (const AABox< DATA_TYPE > &box, const Ray< DATA_TYPE > &ray, DATA_TYPE &tIn, DATA_TYPE &tOut)

  Given an axis-aligned bounding box and a ray (or subclass thereof), returns whether the ray intersects the box, and if so, `tIn` and `tOut` are set to the parametric terms on the ray where the segment enters and exits the box respectively.

- template<class DATA_TYPE >
bool intersect (const AABox< DATA_TYPE > &box, const LineSeg< DATA_TYPE > &seg, unsigned int &numHits, DATA_TYPE &tIn, DATA_TYPE &tOut)

  Given a line segment and an axis-aligned bounding box, returns whether the line intersects the box, and if so, `tIn` and `tOut` are set to the parametric terms on the line segment where the segment enters and exits the box respectively.

- template<class DATA_TYPE >
bool intersect (const LineSeg< DATA_TYPE > &seg, const AABox< DATA_TYPE > &box, unsigned int &numHits, DATA_TYPE &tIn, DATA_TYPE &tOut)

  Given a line segment and an axis-aligned bounding box, returns whether the line intersects the box, and if so, `tIn` and `tOut` are set to the parametric terms on the line segment where the segment enters and exits the box respectively.

- template<class DATA_TYPE >
bool intersect (const AABox< DATA_TYPE > &box, const Ray< DATA_TYPE > &ray, unsigned int &numHits, DATA_TYPE &tIn, DATA_TYPE &tOut)

  Given a ray and an axis-aligned bounding box, returns whether the ray intersects the box, and if so, `tIn` and `tOut` are set to the parametric terms on the ray where it enters and exits the box respectively.

- template<class DATA_TYPE >
bool intersect (const Ray< DATA_TYPE > &ray, const AABox< DATA_TYPE > &box, unsigned int &numHits, DATA_TYPE &tIn, DATA_TYPE &tOut)

  Given a ray and an axis-aligned bounding box, returns whether the ray intersects the box, and if so, `tIn` and `tOut` are set to the parametric terms on the ray where it enters and exits the box respectively.

- template<class DATA_TYPE >
bool intersect (const Sphere< DATA_TYPE > &sph1, const Vec< DATA_TYPE, 3 > &path1, const Sphere< DATA_TYPE > &sph2, const Vec<

DATA_TYPE, 3 > &path2, DATA_TYPE &firstContact, DATA_TYPE &secondContact)

Tests if the given Spheres intersect if moved along the given paths.

- template<class DATA_TYPE >
bool intersect (const AABox< DATA_TYPE > &box, const Sphere< DATA_TYPE > &sph)

  Tests if the given AABox and Sphere intersect with each other.

- template<class DATA_TYPE >
bool intersect (const Sphere< DATA_TYPE > &sph, const AABox< DATA_TYPE > &box)

  Tests if the given AABox and Sphere intersect with each other.

- template<class DATA_TYPE >
bool intersect (const Sphere< DATA_TYPE > &sphere, const Point< DATA_TYPE, 3 > &point)

  intersect point/sphere.

- template<typename T >
bool intersect (const Sphere< T > &sphere, const Ray< T > &ray, int &numhits, T &t0, T &t1)

  intersect ray/sphere-shell (not volume).

- template<typename T >
bool intersect (const Sphere< T > &sphere, const LineSeg< T > &lineseg, int &numhits, T &t0, T &t1)

  intersect LineSeg/Sphere-shell (not volume).

- template<typename T >
bool intersectVolume (const Sphere< T > &sphere, const LineSeg< T > &ray, int &numhits, T &t0, T &t1)

  intersect lineseg/sphere-volume.

- template<typename T >
bool intersectVolume (const Sphere< T > &sphere, const Ray< T > &ray, int &numhits, T &t0, T &t1)

  intersect ray/sphere-volume.

- template<class DATA_TYPE >
bool intersect (const Plane< DATA_TYPE > &plane, const Ray< DATA_TYPE > &ray, DATA_TYPE &t)

  Tests if the given plane and ray intersect with each other.

- template<class DATA_TYPE >
bool intersect (const Plane< DATA_TYPE > &plane, const LineSeg< DATA_-TYPE > &seg, DATA_TYPE &t)

  Tests if the given plane and lineseg intersect with each other.

- template<class DATA_TYPE >
bool intersect (const Tri< DATA_TYPE > &tri, const Ray< DATA_TYPE > &ray, float &u, float &v, float &t)

  Tests if the given triangle and ray intersect with each other.

- template<class DATA_TYPE >
bool intersectDoubleSided (const Tri< DATA_TYPE > &tri, const Ray< DATA_TYPE > &ray, DATA_TYPE &u, DATA_TYPE &v, DATA_TYPE &t)

  Tests if the given triangle intersects with the given ray, from both sides.

- template<class DATA_TYPE >
bool intersect (const Tri< DATA_TYPE > &tri, const LineSeg< DATA_TYPE > &lineseg, DATA_TYPE &u, DATA_TYPE &v, DATA_TYPE &t)

  Tests if the given triangle and line segment intersect with each other.

- 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 >
DATA_TYPE distanceSquared (const LineSeg< DATA_TYPE > &lineseg, const Point< DATA_TYPE, 3 > &pt)

  Computes the shortest distance from the line segment to the given point.

- int combineMatrixStates (int state1, int state2)

  utility function for use by matrix operations.

- template<typename DATA_TYPE_OUT , typename DATA_TYPE_IN , unsigned ROWS, unsigned COLS>
gmtl::Matrix< DATA_TYPE_OUT, ROWS, COLS > convertTo (const gmtl::Matrix< DATA_TYPE_IN, ROWS, COLS > &in)

  Converts a matrix of one data type to another, such as gmtl::Matrix44f to gmtl::Matrix44d.

- 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)
- template< class DATA_TYPE >
  bool operator== (const Ray< DATA_TYPE > &ls1, const Ray< DATA_TYPE > &ls2)

  Compare two line segments to see if they are EXACTLY the same.

- template< class DATA_TYPE >
  bool operator!= (const Ray< DATA_TYPE > &ls1, const Ray< DATA_TYPE > &ls2)

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

- template< class DATA_TYPE >
  bool isEqual (const Ray< DATA_TYPE > &ls1, const Ray< DATA_TYPE > &ls2, const DATA_TYPE &eps)

  Compare two line segments to see if the are the same within the given tolerance.

- const char * getVersion ()
- template< typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  Ray< DATA_TYPE > & xform (Ray< DATA_TYPE > &result, const Matrix< DATA_TYPE, ROWS, COLS > &matrix, const Ray< DATA_TYPE > &ray)

  transform ray by a matrix.

- template< typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  Ray< DATA_TYPE > operator* (const Matrix< DATA_TYPE, ROWS, COLS > &matrix, const Ray< DATA_TYPE > &ray)

  *ray * a matrix multiplication of [m x k] matrix by a ray.*

- template< typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  Ray< DATA_TYPE > & operator*= (Ray< DATA_TYPE > &ray, const Matrix< DATA_TYPE, ROWS, COLS > &matrix)

  *ray *= a matrix multiplication of [m x k] matrix by a ray.*

- template< typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  LineSeg< DATA_TYPE > & xform (LineSeg< DATA_TYPE > &result, const Matrix< DATA_TYPE, ROWS, COLS > &matrix, const LineSeg< DATA_TYPE > &seg)

  transform seg by a matrix.

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
LineSeg< DATA_TYPE > operator∗ (const Matrix< DATA_TYPE, ROWS, COLS > &matrix, const LineSeg< DATA_TYPE > &seg)

    seg ∗ a matrix multiplication of [m x k] matrix by a seg.

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
LineSeg< DATA_TYPE > & operator∗= (LineSeg< DATA_TYPE > &seg, const Matrix< DATA_TYPE, ROWS, COLS > &matrix)

    seg ∗= a matrix multiplication of [m x k] matrix by a seg.

AABox Comparitors

- template<class DATA_TYPE >
bool operator== (const AABox< DATA_TYPE > &b1, const AABox< DATA_TYPE > &b2)

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

- template<class DATA_TYPE >
bool operator!= (const AABox< DATA_TYPE > &b1, const AABox< DATA_TYPE > &b2)

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

- template<class DATA_TYPE >
bool isEqual (const AABox< DATA_TYPE > &b1, const AABox< DATA_-TYPE > &b2, const DATA_TYPE &eps)

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

AxisAngle Comparitors

- template<class DATA_TYPE >
bool operator== (const AxisAngle< DATA_TYPE > &a1, const AxisAngle< DATA_TYPE > &a2)

    Compares 2 AxisAngles to see if they are exactly the same.

- template<class DATA_TYPE >
bool operator!= (const AxisAngle< DATA_TYPE > &a1, const AxisAngle< DATA_TYPE > &a2)

    Compares 2 AxisAngles to see if they are NOT exactly the same.

- template<class DATA_TYPE >
bool isEqual (const AxisAngle< DATA_TYPE > &a1, const AxisAngle< DATA_TYPE > &a2, const DATA_TYPE eps=0)

    Compares a1 and a2 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 > &c1, const Coord< POS_TYPE, ROT_TYPE > &c2)

  Compare two coordinate frames for equality.

- template<typename POS_TYPE , typename ROT_TYPE >
  bool operator!= (const Coord< POS_TYPE, ROT_TYPE > &c1, const Coord< POS_TYPE, ROT_TYPE > &c2)

  Compare two coordinate frames for not-equality.

- template<typename POS_TYPE , typename ROT_TYPE >
  bool isEqual (const Coord< POS_TYPE, ROT_TYPE > &c1, const Coord< POS_TYPE, ROT_TYPE > &c2, typename Coord< POS_TYPE, ROT_TYPE >::DataType tol=0)

  Compare two coordinate frames for equality with a given tolerance.

EulerAngle Comparitors

- template<class DATA_TYPE , typename ROT_ORDER >
  bool operator== (const EulerAngle< DATA_TYPE, ROT_ORDER > &e1, const EulerAngle< DATA_TYPE, ROT_ORDER > &e2)

  Compares 2 EulerAngles (component-wise) to see if they are exactly the same.

- template<class DATA_TYPE , typename ROT_ORDER >
  bool operator!= (const EulerAngle< DATA_TYPE, ROT_ORDER > &e1, const EulerAngle< DATA_TYPE, ROT_ORDER > &e2)

  Compares e1 and e2 (component-wise) to see if they are NOT exactly the same.

- template<class DATA_TYPE , typename ROT_ORDER >
  bool isEqual (const EulerAngle< DATA_TYPE, ROT_ORDER > &e1, const EulerAngle< DATA_TYPE, ROT_ORDER > &e2, const DATA_TYPE eps=0)

  Compares e1 and e2 (component-wise) to see if they are the same within a given tolerance.

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<class DATA_TYPE >
Vec< DATA_TYPE, 3 > makeCross (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<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)
Normalize a 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 the axis of an AxisAngle normalized

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 T >
Matrix< T, 4, 4 > & setFrustum (Matrix< T, 4, 4 > &result, T left, T top, T right, T bottom, T nr, T fr)

    Set an arbitrary projection matrix.

- template<typename T >
Matrix< T, 4, 4 > & setOrtho (Matrix< T, 4, 4 > &result, T left, T top, T right, T bottom, T nr, T fr)

    Set an orthographic projection matrix creates a transformation that produces a parallel projection matrix.

- template<typename T >
Matrix< T, 4, 4 > & setPerspective (Matrix< T, 4, 4 > &result, T fovy, T aspect, T nr, T fr)

    Set a symmetric perspective projection matrix.

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS, unsigned SIZE, typename REP >
Matrix< DATA_TYPE, ROWS, COLS > & setTrans (Matrix< DATA_-TYPE, ROWS, COLS > &result, const VecBase< DATA_TYPE, SIZE, REP > &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 DATA_TYPE , unsigned ROWS, unsigned COLS>
Matrix< DATA_TYPE, ROWS, COLS > & setScale (Matrix< DATA_TYPE, ROWS, COLS > &result, const DATA_TYPE scale)
    Sets the scale part of a matrix.

- template<typename MATRIX_TYPE , typename INPUT_TYPE >
MATRIX_TYPE makeScale (const INPUT_TYPE &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, 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>
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 > & 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>
DATA_TYPE makeYRot (const Matrix< DATA_TYPE, ROWS, COLS > &mat)
    Extracts the Y axis rotation information from the matrix.

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
DATA_TYPE makeXRot (const Matrix< DATA_TYPE, ROWS, COLS > &mat)
    Extracts the X-axis rotation information from the matrix.

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

Extracts the Z-axis rotation 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 > makeInvert (const Matrix< DATA_TYPE, ROWS, COLS > &src)

  Creates a matrix that is the inverse of the given source 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 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 Note: It is set directly, but this is equivalent to T*R where T is the translation matrix and R is the rotation matrix.*

- 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.

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.

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, INTER-NAL, 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, const DATA_TYPE &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, const 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 > & invertTrans (Matrix< DATA_TYPE, ROWS, COLS > &result, const Matrix< DATA_TYPE, ROWS, COLS > &src)

  translational matrix inversion.

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

  orthogonal matrix inversion.

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

  affine matrix inversion.

- template<typename DATA_TYPE , unsigned SIZE>
  Matrix< DATA_TYPE, SIZE, SIZE > & invertFull_GJ (Matrix< DATA_TYPE, SIZE, SIZE > &result, const Matrix< DATA_TYPE, SIZE, SIZE > &src)

  Full matrix inversion using Gauss-Jordan elimination.

- template<typename DATA_TYPE , unsigned SIZE>
  Matrix< DATA_TYPE, SIZE, SIZE > & invertFull_orig (Matrix< DATA_TYPE, SIZE, SIZE > &result, const Matrix< DATA_TYPE, SIZE, SIZE > &src)

  full matrix inversion.

- 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)

  Invert method.

- 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)
    Tests 2 matrices for equality.

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
bool operator!= (const Matrix< DATA_TYPE, ROWS, COLS > &lhs, const Matrix< DATA_TYPE, ROWS, COLS > &rhs)
    Tests 2 matrices for inequality.

- 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=0)
    Tests 2 matrices for equality within a tolerance.

Output Stream Operators

- template<typename DATA_TYPE , unsigned SIZE, typename REP >
std::ostream & operator<< (std::ostream &out, const VecBase< DATA_TYPE, SIZE, REP > &v)
    Outputs a string representation of the given VecBase type to the given output stream.

- template<class DATA_TYPE , typename ROTATION_ORDER >
std::ostream & operator<< (std::ostream &out, const EulerAngle< DATA_TYPE, ROTATION_ORDER > &e)
    Outputs a string representation of the given EulerAngle 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.
- template<typename DATA_TYPE >
  std::ostream & operator<< (std::ostream &out, const AABox< DATA_-TYPE > &b)
  Outputs a string representation of the given AABox to the given output stream.
- template<typename DATA_TYPE >
  std::ostream & operator<< (std::ostream &out, const Ray< DATA_TYPE > &b)
  Outputs a string representation of the given Ray to the given output stream.
- template<typename DATA_TYPE >
  std::ostream & operator<< (std::ostream &out, const LineSeg< DATA_-TYPE > &b)
  Outputs a string representation of the given LineSeg to the given output stream.
- template<typename POS_TYPE , typename ROT_TYPE >
  std::ostream & operator<< (std::ostream &out, const Coord< POS_TYPE, ROT_TYPE > &c)

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.
- template<class DATA_TYPE , unsigned SIZE>
void reflect (Point< DATA_TYPE, SIZE > &result, const Plane< DATA_TYPE > &plane, const Point< DATA_TYPE, SIZE > &point)
  Mirror the point by the plane.

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) Does quaternion multiplication.

- 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, Quat< DATA_TYPE > q2)

  quotient of two quaternions

- template<typename DATA_TYPE >
  Quat< DATA_TYPE > operator/ (const Quat< DATA_TYPE > &q1, Quat< DATA_TYPE > q2)

quotient of two quaternions

- template<typename DATA_TYPE >
Quat< DATA_TYPE > & operator/= (Quat< DATA_TYPE > &result, 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=0.0001f)

  Determines if the given quaternion 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, bool adjustSign=true)

  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.

- template<class T >
  void ignore_unused_variable_warning (const T &)

Vector/Point Operations

- template<typename T , unsigned SIZE, typename R1 >
  VecBase< T, SIZE, meta::VecUnaryExpr< VecBase< T, SIZE, R1 >, meta::VecNegUnary > > operator- (const VecBase< T, SIZE, R1 > &v1)
  Negates v1.
- template<class DATA_TYPE , unsigned SIZE, typename REP2 >
  VecBase< DATA_TYPE, SIZE > & operator+= (VecBase< DATA_TYPE, SIZE > &v1, const VecBase< DATA_TYPE, SIZE, REP2 > &v2)
  Adds v2 to v1 and stores the result in v1.
- template<typename T , unsigned SIZE, typename R1 , typename R2 >
  VecBase< T, SIZE, meta::VecBinaryExpr< VecBase< T, SIZE, R1 >, VecBase< T, SIZE, R2 >, meta::VecPlusBinary > > operator+ (const VecBase< T, SIZE, R1 > &v1, const VecBase< T, SIZE, R2 > &v2)
  Adds v2 to v1 and returns the result.
- template<class DATA_TYPE , unsigned SIZE, typename REP2 >
  VecBase< DATA_TYPE, SIZE > & operator-= (VecBase< DATA_TYPE, SIZE > &v1, const VecBase< DATA_TYPE, SIZE, REP2 > &v2)
  Subtracts v2 from v1 and stores the result in v1.
- template<typename T , unsigned SIZE, typename R1 , typename R2 >
  VecBase< T, SIZE, meta::VecBinaryExpr< VecBase< T, SIZE, R1 >, VecBase< T, SIZE, R2 >, meta::VecMinusBinary > > operator- (const VecBase< T, SIZE, R1 > &v1, const VecBase< T, SIZE, R2 > &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<typename T , unsigned SIZE, typename R1 >
  VecBase< T, SIZE, meta::VecBinaryExpr< VecBase< T, SIZE, R1 >, VecBase< T, SIZE, meta::ScalarArg< T > >, meta::VecMultBinary > > operator* (const VecBase< T, SIZE, R1 > &v1, const T scalar)
  Multiplies v1 by a scalar value and returns the result.

- template<typename T , unsigned SIZE, typename R1 >
  VecBase< T, SIZE, meta::VecBinaryExpr< VecBase< T, SIZE, meta::ScalarArg< T > >, VecBase< T, SIZE, R1 >, meta::VecMultBinary > > operator∗ (const T scalar, const VecBase< T, SIZE, R1 > &v1)
- 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 returns the result.

- template<typename T , unsigned SIZE, typename R1 >
  VecBase< T, SIZE, meta::VecBinaryExpr< VecBase< T, SIZE, R1 >, VecBase< T, SIZE, meta::ScalarArg< T > >, meta::VecDivBinary > > operator/ (const VecBase< T, SIZE, R1 > &v1, const T scalar)

  Divides v1 by a scalar value and returns the result.

Vector Operations

- template<class DATA_TYPE , unsigned SIZE, typename REP1 , typename REP2 >
  DATA_TYPE dot (const VecBase< DATA_TYPE, SIZE, REP1 > &v1, const VecBase< DATA_TYPE, SIZE, REP2 > &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) 0.0001f)

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

- 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.

- template<class DATA_TYPE , unsigned SIZE>
VecBase< DATA_TYPE, SIZE > & reflect (VecBase< DATA_TYPE, SIZE > &result, const VecBase< DATA_TYPE, SIZE > &vec, const Vec< DATA_TYPE, SIZE > &normal)
  Reflect a vector about a normal.

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.
- 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.

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

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
Point< DATA_TYPE, COLS > operator∗ (const Point< DATA_TYPE, COLS > &point, const Matrix< DATA_TYPE, ROWS, COLS > &matrix)

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

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
Point< DATA_TYPE, COLS > operator∗= (Point< DATA_TYPE, COLS > &point, const Matrix< DATA_TYPE, ROWS, COLS > &matrix)

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

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS, unsigned COLS_-MINUS_ONE>
Point< DATA_TYPE, COLS_MINUS_ONE > & operator∗= (Point< DATA_TYPE, COLS_MINUS_ONE > &point, const Matrix< DATA_-TYPE, ROWS, COLS > &matrix)

  partial point ∗= a matrix 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]).

Variables

- const unsigned int IN_FRONT_OF_ALL_PLANES = 6
- 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

Constants

- const float GMTL_EPSILON = 1.0e-6f
- const float GMTL_MAT_EQUAL_EPSILON = 0.001f
- const float GMTL_VEC_EQUAL_EPSILON = 0.0001f

9.1.1 Detailed Description

Meta programming classes. Meta programming classes for vec operations.

Expression template classes for vec operations.

Static assertion macros Borrowed from boost and Loki.

9.1.2 Typedef Documentation

9.1.2.1 typedef AABox<double> gmtl::AABoxd

Definition at line 144 of file AABox.h.

9.1.2.2 typedef AABox<float> gmtl::AABoxf

Definition at line 143 of file AABox.h.

9.1.2.3 typedef AxisAngle<double> gmtl::AxisAngled

Definition at line 121 of file AxisAngle.h.

9.1.2.4 typedef AxisAngle<float> gmtl::AxisAnglef

Definition at line 120 of file AxisAngle.h.

9.1.2.5 typedef Coord<Vec3d, AxisAngled> gmtl::Coord3dAxisAngle

Definition at line 189 of file Coord.h.

9.1.2.6 typedef Coord<Vec3d, Quatd> gmtl::Coord3dQuat

Definition at line 180 of file Coord.h.

9.1.2.7 typedef Coord<Vec3d, EulerAngleXYZd> gmtl::Coord3dXYZ

Definition at line 166 of file Coord.h.

9.1.2.8 typedef Coord<Vec3d, EulerAngleZXYd> gmtl::Coord3dZXY

Definition at line 168 of file Coord.h.

9.1.2.9 typedef Coord<Vec3d, EulerAngleZYXd> gmtl::Coord3dZYX

Definition at line 167 of file Coord.h.

9.1.2.10 typedef Coord<Vec3f, AxisAnglef> gmtl::Coord3fAxisAngle

3 elt types

Definition at line 188 of file Coord.h.

9.1.2.11 typedef Coord<Vec3f, Quatf> gmtl::Coord3fQuat

3 elt types

Definition at line 179 of file Coord.h.

9.1.2.12 typedef Coord<Vec3f, EulerAngleXYZf> gmtl::Coord3fXYZ

3 elt types

Definition at line 163 of file Coord.h.

9.1.2.13 typedef Coord<Vec3f, EulerAngleZXYf> gmtl::Coord3fZXY

Definition at line 165 of file Coord.h.

9.1.2.14 typedef Coord<Vec3f, EulerAngleZYXf> gmtl::Coord3fZYX

Definition at line 164 of file Coord.h.

9.1.2.15 typedef Coord<Vec4d, AxisAngled> gmtl::Coord4dAxisAngle

Definition at line 193 of file Coord.h.

9.1.2.16 typedef Coord<Vec4d, Quatd> gmtl::Coord4dQuat

Definition at line 184 of file Coord.h.

9.1.2.17 typedef Coord<Vec4d, EulerAngleXYZd> gmtl::Coord4dXYZ

Definition at line 174 of file Coord.h.

9.1.2.18 typedef Coord<Vec4d, EulerAngleZXYd> gmtl::Coord4dZXY

Definition at line 176 of file Coord.h.

9.1.2.19 typedef Coord<Vec4d, EulerAngleZYXd> gmtl::Coord4dZYX

Definition at line 175 of file Coord.h.

9.1.2.20 typedef Coord<Vec4f, AxisAnglef> gmtl::Coord4fAxisAngle

4 elt types

Definition at line 192 of file Coord.h.

9.1.2.21 typedef Coord<Vec4f, Quatf> gmtl::Coord4fQuat

4 elt types

Definition at line 183 of file Coord.h.

9.1.2.22 typedef Coord<Vec4f, EulerAngleXYZf> gmtl::Coord4fXYZ

4 elt types

Definition at line 171 of file Coord.h.

9.1.2.23 typedef Coord<Vec4f, EulerAngleZXYf> gmtl::Coord4fZXY

Definition at line 173 of file Coord.h.

9.1.2.24 typedef Coord<Vec4f, EulerAngleZYXf> gmtl::Coord4fZYX

Definition at line 172 of file Coord.h.

9.1.2.25 typedef Coord<Vec3d, AxisAngled> gmtl::CoordVec3AxisAngled

Definition at line 155 of file Coord.h.

9.1.2.26 typedef Coord<Vec3f, AxisAnglef> gmtl::CoordVec3AxisAnglef

Definition at line 156 of file Coord.h.

9.1.2.27 typedef Coord<Vec3d, EulerAngleXYZd> gmtl::CoordVec3EulerAngleXYZd

Definition at line 140 of file Coord.h.

9.1.2.28 typedef Coord<Vec3f, EulerAngleXYZf> gmtl::CoordVec3EulerAngleXYZf

Definition at line 141 of file Coord.h.

9.1.2.29 typedef Coord<Vec3d, EulerAngleZXYd> gmtl::CoordVec3EulerAngleZXYd

Definition at line 150 of file Coord.h.

9.1.2.30 typedef Coord<Vec3f, EulerAngleZXYf> gmtl::CoordVec3EulerAngleZXYf

Definition at line 151 of file Coord.h.

9.1.2.31 typedef Coord<Vec3d, EulerAngleZYXd> gmtl::CoordVec3EulerAngleZYXd

Definition at line 145 of file Coord.h.

9.1.2.32 typedef Coord<Vec3f, EulerAngleZYXf> gmtl::CoordVec3EulerAngleZYXf

Definition at line 146 of file Coord.h.

9.1.2.33 typedef Coord<Vec4d, AxisAngled> gmtl::CoordVec4AxisAngled

Definition at line 157 of file Coord.h.

9.1.2.34 typedef Coord<Vec4f, AxisAnglef> gmtl::CoordVec4AxisAnglef

Definition at line 158 of file Coord.h.

9.1.2.35 typedef Coord<Vec4d, EulerAngleXYZd> gmtl::CoordVec4EulerAngleXYZd

Definition at line 142 of file Coord.h.

9.1.2.36 typedef Coord<Vec4f, EulerAngleXYZf> gmtl::CoordVec4EulerAngleXYZf

Definition at line 143 of file Coord.h.

9.1.2.37 typedef Coord<Vec4d, EulerAngleZXYd> gmtl::CoordVec4EulerAngleZXYd

Definition at line 152 of file Coord.h.

9.1.2.38 typedef Coord<Vec4f, EulerAngleZXYf> gmtl::CoordVec4EulerAngleZXYf

Definition at line 153 of file Coord.h.

9.1.2.39 typedef Coord<Vec4d, EulerAngleZYXd> gmtl::CoordVec4EulerAngleZYXd

Definition at line 147 of file Coord.h.

9.1.2.40 typedef Coord<Vec4f, EulerAngleZYXf> gmtl::CoordVec4EulerAngleZYXf

Definition at line 148 of file Coord.h.

9.1.2.41 typedef CubicCurve<double, 1> gmtl::CubicCurve1d

Definition at line 402 of file ParametricCurve.h.

9.1.2.42 typedef CubicCurve<float, 1> gmtl::CubicCurve1f

Definition at line 399 of file ParametricCurve.h.

9.1.2.43 typedef CubicCurve<double, 2> gmtl::CubicCurve2d

Definition at line 403 of file ParametricCurve.h.

9.1.2.44 typedef CubicCurve<float, 2> gmtl::CubicCurve2f

Definition at line 400 of file ParametricCurve.h.

9.1.2.45 typedef CubicCurve<double, 3> gmtl::CubicCurve3d

Definition at line 404 of file ParametricCurve.h.

9.1.2.46 typedef CubicCurve<float, 3> gmtl::CubicCurve3f

Definition at line 401 of file ParametricCurve.h.

9.1.2.47 typedef EulerAngle<double, XYZ> gmtl::EulerAngleXYZd

Definition at line 123 of file EulerAngle.h.

9.1.2.48 typedef EulerAngle<float, XYZ> gmtl::EulerAngleXYZf

Definition at line 122 of file EulerAngle.h.

9.1.2.49 typedef EulerAngle<double, ZXY> gmtl::EulerAngleZXYd

Definition at line 127 of file EulerAngle.h.

9.1.2.50 typedef EulerAngle<float, ZXY> gmtl::EulerAngleZXYf

Definition at line 126 of file EulerAngle.h.

9.1.2.51 typedef EulerAngle<double, ZYX> gmtl::EulerAngleZYXd

Definition at line 125 of file EulerAngle.h.

9.1.2.52 typedef EulerAngle<float, ZYX> gmtl::EulerAngleZYXf

Definition at line 124 of file EulerAngle.h.

9.1.2.53 typedef Frustum<double> gmtl::Frustumd

Definition at line 150 of file Frustum.h.

9.1.2.54 typedef Frustum<float> gmtl::Frustumf

Definition at line 149 of file Frustum.h.

9.1.2.55 typedef LinearCurve<double, 1> gmtl::LinearCurve1d

Definition at line 390 of file ParametricCurve.h.

9.1.2.56 typedef LinearCurve<float, 1> gmtl::LinearCurve1f

Definition at line 387 of file ParametricCurve.h.

9.1.2.57 typedef LinearCurve<double, 2> gmtl::LinearCurve2d

Definition at line 391 of file ParametricCurve.h.

9.1.2.58 typedef LinearCurve<float, 2> gmtl::LinearCurve2f

Definition at line 388 of file ParametricCurve.h.

9.1.2.59 typedef LinearCurve<double, 3> gmtl::LinearCurve3d

Definition at line 392 of file ParametricCurve.h.

9.1.2.60 typedef LinearCurve<float, 3> gmtl::LinearCurve3f

Definition at line 389 of file ParametricCurve.h.

9.1.2.61 typedef LineSeg<double> gmtl::LineSegd

Definition at line 83 of file LineSeg.h.

9.1.2.62 typedef LineSeg<float> gmtl::LineSegf

Definition at line 82 of file LineSeg.h.

9.1.2.63 typedef Matrix<double, 2, 2> gmtl::Matrix22d

Definition at line 492 of file Matrix.h.

9.1.2.64 typedef Matrix<float, 2, 2> gmtl::Matrix22f

Definition at line 491 of file Matrix.h.

9.1.2.65 typedef Matrix<double, 2, 3> gmtl::Matrix23d

Definition at line 494 of file Matrix.h.

9.1.2.66 typedef Matrix<float, 2, 3> gmtl::Matrix23f

Definition at line 493 of file Matrix.h.

9.1.2.67 typedef Matrix<double, 3, 3> gmtl::Matrix33d

Definition at line 496 of file Matrix.h.

9.1.2.68 typedef Matrix<float, 3, 3> gmtl::Matrix33f

Definition at line 495 of file Matrix.h.

9.1.2.69 typedef Matrix<double, 3, 4> gmtl::Matrix34d

Definition at line 498 of file Matrix.h.

9.1.2.70 typedef Matrix<float, 3, 4> gmtl::Matrix34f

Definition at line 497 of file Matrix.h.

9.1.2.71 typedef Matrix<double, 4, 4> gmtl::Matrix44d

Definition at line 500 of file Matrix.h.

9.1.2.72 typedef Matrix<float, 4, 4> gmtl::Matrix44f

Definition at line 499 of file Matrix.h.

9.1.2.73 typedef Plane<double> gmtl::Planed

Definition at line 157 of file Plane.h.

9.1.2.74 typedef Plane<float> gmtl::Planef

Definition at line 156 of file Plane.h.

9.1.2.75 typedef Point<double,2> gmtl::Point2d

Definition at line 123 of file Point.h.

9.1.2.76 typedef Point<float,2> gmtl::Point2f

Definition at line 122 of file Point.h.

9.1.2.77 typedef Point<int,2> gmtl::Point2i

Definition at line 121 of file Point.h.

9.1.2.78 typedef Point<double,3> gmtl::Point3d

Definition at line 126 of file Point.h.

9.1.2.79 typedef Point<float,3> gmtl::Point3f

Definition at line 125 of file Point.h.

9.1.2.80 typedef Point<int, 3> gmtl::Point3i

Definition at line 124 of file Point.h.

9.1.2.81 typedef Point<double,4> gmtl::Point4d

Definition at line 129 of file Point.h.

9.1.2.82 typedef Point<float,4> gmtl::Point4f

Definition at line 128 of file Point.h.

9.1.2.83 typedef Point<int, 4> gmtl::Point4i

Definition at line 127 of file Point.h.

9.1.2.84 typedef QuadraticCurve<double, 1> gmtl::QuadraticCurve1d

Definition at line 396 of file ParametricCurve.h.

9.1.2.85 typedef QuadraticCurve<float, 1> gmtl::QuadraticCurve1f

Definition at line 393 of file ParametricCurve.h.

9.1.2.86 typedef QuadraticCurve<double, 2> gmtl::QuadraticCurve2d

Definition at line 397 of file ParametricCurve.h.

9.1.2.87 typedef QuadraticCurve<float, 2> gmtl::QuadraticCurve2f

Definition at line 394 of file ParametricCurve.h.

9.1.2.88 typedef QuadraticCurve<double, 3> gmtl::QuadraticCurve3d

Definition at line 398 of file ParametricCurve.h.

9.1.2.89 typedef QuadraticCurve<float, 3> gmtl::QuadraticCurve3f

Definition at line 395 of file ParametricCurve.h.

9.1.2.90 typedef Quat<double> gmtl::Quatd

Definition at line 156 of file Quat.h.

9.1.2.91 typedef Quat<float> gmtl::Quatf

Definition at line 155 of file Quat.h.

9.1.2.92 typedef Ray<double> gmtl::Rayd

Definition at line 114 of file Ray.h.

9.1.2.93 typedef Ray<float> gmtl::Rayf

Definition at line 113 of file Ray.h.

9.1.2.94 typedef Sphere<double> gmtl::Sphered

Definition at line 107 of file Sphere.h.

9.1.2.95 typedef Sphere<float> gmtl::Spheref

Definition at line 106 of file Sphere.h.

9.1.2.96 typedef Tri<double> gmtl::Trid

Definition at line 142 of file Tri.h.

9.1.2.97 typedef Tri<float> gmtl::Trif

Definition at line 141 of file Tri.h.

9.1.2.98 typedef Tri<int> gmtl::Trii

Definition at line 143 of file Tri.h.

9.1.2.99 typedef Vec<double,2> gmtl::Vec2d

Definition at line 125 of file Vec.h.

9.1.2.100 typedef Vec<float,2> gmtl::Vec2f

Definition at line 124 of file Vec.h.

9.1.2.101 typedef Vec<int, 2> gmtl::Vec2i

Definition at line 123 of file Vec.h.

9.1.2.102 typedef Vec<double,3> gmtl::Vec3d

Definition at line 128 of file Vec.h.

9.1.2.103 typedef Vec<float,3> gmtl::Vec3f

Definition at line 127 of file Vec.h.

9.1.2.104 typedef Vec<int, 3> gmtl::Vec3i

Definition at line 126 of file Vec.h.

9.1.2.105 typedef Vec<double,4> gmtl::Vec4d

Definition at line 131 of file Vec.h.

9.1.2.106 typedef Vec<float,4> gmtl::Vec4f

Definition at line 130 of file Vec.h.

9.1.2.107 typedef Vec<int, 4> gmtl::Vec4i

Definition at line 129 of file Vec.h.

9.1.3 Function Documentation

9.1.3.1 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& gmtl::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).

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

Definition at line 141 of file MatrixOps.h.

```
{
   // 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 );

   // track state
   result.mState = combineMatrixStates( lhs.mState, rhs.mState );
   return result;
}
```

9.1.3.2 template<typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::add (Quat< DATA_TYPE > & *result*, const Quat< DATA_TYPE > & *q1*, const Quat< DATA_TYPE > & *q2*)

vector addition

See also

Quat

Definition at line 227 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;
}
```

9.1.3.3 const AxisAngle<double> gmtl::AXISANGLE_IDENTITYD (0. *0,* 1. *0,* 0. *0,* 0. *0*)

9.1.3.4 const AxisAngle<float> gmtl::AXISANGLE_IDENTITYF (0. *0f,* 1. *0f,* 0. *0f,* 0. *0f*)

9.1.3.5 template<class DATA_TYPE > Point<DATA_TYPE, 3> gmtl::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 28 of file TriOps.h.

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

9.1.3.6 int gmtl::combineMatrixStates (int *state1,* int *state2*) `[inline]`

utility function for use by matrix operations.

given two matrices, when combined with set(..) or xform(..) types of operations, compute what matrixstate will the resulting matrix have?

Definition at line 536 of file Matrix.h.

```
{
   switch (state1)
   {
   case Matrix44f::IDENTITY:
      switch (state2)
      {
      case Matrix44f::XFORM_ERROR: return state2;
      case Matrix44f::NON_UNISCALE: return Matrix44f::XFORM_ERROR;
      default: return state2;
      }
   case Matrix44f::TRANS:
      switch (state2)
      {
      case Matrix44f::IDENTITY: return state1;
      case Matrix44f::ORTHOGONAL: return Matrix44f::AFFINE;
      case Matrix44f::NON_UNISCALE: return Matrix44f::XFORM_ERROR;
      default: return state2;
      }
   case Matrix44f::ORTHOGONAL:
      switch (state2)
      {
      case Matrix44f::IDENTITY: return state1;
      case Matrix44f::TRANS: return Matrix44f::AFFINE;
      case Matrix44f::NON_UNISCALE: return Matrix44f::XFORM_ERROR;
      default: return state2;
      }
   case Matrix44f::AFFINE:
      switch (state2)
      {
      case Matrix44f::IDENTITY:
      case Matrix44f::TRANS:
      case Matrix44f::ORTHOGONAL:  return state1;
      case Matrix44f::NON_UNISCALE: return Matrix44f::XFORM_ERROR;
      case Matrix44f::AFFINE | Matrix44f::NON_UNISCALE:
      default: return state2;
      }
   case Matrix44f::AFFINE | Matrix44f::NON_UNISCALE:
      switch (state2)
      {
      case Matrix44f::IDENTITY:
      case Matrix44f::TRANS:
      case Matrix44f::ORTHOGONAL:
      case Matrix44f::AFFINE:  return state1;
      case Matrix44f::NON_UNISCALE: return Matrix44f::XFORM_ERROR;
      default: return state2;
      }
   case Matrix44f::FULL:
      switch (state2)
      {
      case Matrix44f::XFORM_ERROR: return state2;
      case Matrix44f::NON_UNISCALE: return Matrix44f::XFORM_ERROR;
      default: return state1;
      }
      break;
   default:
      return Matrix44f::XFORM_ERROR;
   }
```

```
}
```

9.1.3.7 template<typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::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 376 of file QuatOps.h.

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

9.1.3.8 template<typename DATA_TYPE_OUT , typename DATA_TYPE_IN , unsigned ROWS, unsigned COLS> gmtl::Matrix<DATA_TYPE_OUT, ROWS, COLS> gmtl::convertTo (const gmtl::Matrix< DATA_TYPE_IN, ROWS, COLS > & *in*) `[inline]`

Converts a matrix of one data type to another, such as gmtl::Matrix44f to gmtl::Matrix44d.

Internally, this uses loop unrolling to optimize performance.

```
const gmtl::Matrix44f m_float = getMatrix();
gmtl::Matrix44d m_double = gmtl::convertTo<double>(m_float);
```

Precondition

The input matrix and output matrix must have matching dimensions.

Note

The compiler will enforce the pre-condition about the matrix dimensions, but the error message may not always be clear. Use of a static assertion may help with that.

Since

0.5.1

Definition at line 38 of file MatrixConvert.h.

```
{
   using namespace boost::lambda;

   gmtl::Matrix<DATA_TYPE_OUT, ROWS, COLS> out;

   // Accessing in.mData and out.mData in this way allows for use of
   // Boost.Lambda so that a separate helper function is not required to do
   // the assignment.
   const DATA_TYPE_IN* in_data(in.mData);
   DATA_TYPE_OUT* out_data(out.mData);

   // This relies on implicit casting between data types.
   boost::mpl::for_each< boost::mpl::range_c<unsigned int, 0, ROWS * COLS> >(
      *(out_data + boost::lambda::_1) = *(in_data + boost::lambda::_1)
   );

   return out;
}
```

9.1.3.9 template<class DATA_TYPE > Vec<DATA_TYPE,3>& gmtl::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.

Precondition

v1 and v2 are 3-D 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 460 of file VecOps.h.

```
{
   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;
}
```

9.1.3.10 template<class DATA_TYPE > DATA_TYPE gmtl::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 30 of file PlaneOps.h.

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

9.1.3.11 template<class DATA_TYPE > DATA_TYPE gmtl::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 40 of file LineSegOps.h.

```
{
   return gmtl::length(gmtl::Vec<DATA_TYPE, 3>(pt - findNearestPt(lineseg, pt)));

}
```

9.1.3.12 template< class DATA_TYPE > DATA_TYPE gmtl::distanceSquared (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 squared shortest distance from pt to lineseg (value is squared, this func is slightly faster since it doesn't involve a sqrt)

Definition at line 55 of file LineSegOps.h.

```
{
   return gmtl::lengthSquared(gmtl::Vec<DATA_TYPE, 3>(pt - findNearestPt(lineseg,
       pt)));
}
```

9.1.3.13 template< typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::div (Quat< DATA_TYPE > & *result*, const Quat< DATA_TYPE > & *q1*, Quat< DATA_TYPE > *q2*)

quotient of two quaternions

Postcondition

result = q1 $*$ (1/q2) (where 1/q2 is applied first to any xform'd geometry)

See also

Quat

Definition at line 162 of file QuatOps.h.

```
{
   // multiply q1 by the multiplicative inverse of the quaternion
   return mult( result, q1, invert( q2 ) );
}
```

9.1.3.14 template<typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::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 193 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;
}
```

9.1.3.15 template<typename DATA_TYPE > DATA_TYPE gmtl::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

Returns

dot product of q1 and q2

See also

Quat

Definition at line 298 of file QuatOps.h.

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

9.1.3.16 template<class DATA_TYPE , unsigned SIZE, typename REP1 , typename REP2 > DATA_TYPE gmtl::dot (const VecBase< DATA_TYPE, SIZE, REP1 > & *v1*, const VecBase< DATA_TYPE, SIZE, REP2 > & *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 351 of file VecOps.h.

```
{
   return gmtl::meta::DotVecUnrolled<SIZE-1,
                                     VecBase<DATA_TYPE,SIZE,REP1>,
                                     VecBase<DATA_TYPE,SIZE,REP2> >::func(v1,v2);

}
```

9.1.3.17 const EulerAngle<double, XYZ> gmtl::EULERANGLE_-IDENTITY_XYZD (0. *0,* 0. *0,* 0. *0*)

9.1.3.18 const EulerAngle<float, XYZ> gmtl::EULERANGLE_-IDENTITY_XYZF (0. *0f,* 0. *0f,* 0. *0f*)

9.1.3.19 const EulerAngle<double, ZXY> gmtl::EULERANGLE_-IDENTITY_ZXYD (0. *0,* 0. *0,* 0. *0*)

9.1.3.20 const EulerAngle<float, ZXY> gmtl::EULERANGLE_-IDENTITY_ZXYF (0. *0f,* 0. *0f,* 0. *0f*)

9.1.3.21 const EulerAngle<double, ZYX> gmtl::EULERANGLE_-IDENTITY_ZYXD (0. *0,* 0. *0,* 0. *0*)

9.1.3.22 const EulerAngle<float, ZYX> gmtl::EULERANGLE_-IDENTITY_ZYXF (0. *0f,* 0. *0f,* 0. *0f*)

9.1.3.23 template<typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::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 415 of file QuatOps.h.

```
{
   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;
}
```

9.1.3.24 template<class DATA_TYPE > void gmtl::extendVolume (AABox< DATA_TYPE > & *container*, const Point< DATA_TYPE, 3 > & *pt*)

Modifies the existing AABox to tightly enclose itself and the given point.

Parameters

container [in,out] the AABox that will be extended

pt [in] the point which the AABox should contain

Definition at line 393 of file Containment.h.

```
{
   if (! container.isEmpty())
   {
      // X coord
      if (pt[0] > container.mMax[0])
      {
         container.mMax[0] = pt[0];
      }
      else if (pt[0] < container.mMin[0])
      {
         container.mMin[0] = pt[0];
      }

      // Y coord
      if (pt[1] > container.mMax[1])
      {
         container.mMax[1] = pt[1];
      }
      else if (pt[1] < container.mMin[1])
      {
         container.mMin[1] = pt[1];
      }

      // Z coord
      if (pt[2] > container.mMax[2])
```

```
      {
         container.mMax[2] = pt[2];
      }
      else if (pt[2] < container.mMin[2])
      {
         container.mMin[2] = pt[2];
      }
   }
   else
   {
      // Make a box with essentially zero volume at the point
      container.setMin(pt);
      container.setMax(pt);
      container.setEmpty(false);
   }
}
```

9.1.3.25 template<class DATA_TYPE > void gmtl::extendVolume (AABox< DATA_TYPE > & *container*, const AABox< DATA_TYPE > & *box*)

Modifies the container to tightly enclose itself and the given AABox.

Parameters

container [in,out] the AABox that will be extended

box [in] the AABox which container should contain

Definition at line 444 of file Containment.h.

```
{
   // Can't extend by an empty box
   if (box.isEmpty())
   {
      return;
   }

   // An empty container is extended to be the box
   if (container.isEmpty())
   {
      container = box;
   }

   // Just extend by the corners of the box
   extendVolume(container, box.getMin());
   extendVolume(container, box.getMax());
}
```

9.1.3.26 template<class DATA_TYPE > void gmtl::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 79 of file Containment.h.

```
{
   // 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) * static_cast<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;
}
```

9.1.3.27 template<class DATA_TYPE > void gmtl::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 112 of file Containment.h.

```
{
   // 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) *
                         static_cast<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;
}
```

9.1.3.28 template<class DATA_TYPE > DATA_TYPE gmtl::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 96 of file PlaneOps.h.

```
{
   // 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;
}
```

9.1.3.29 template<class DATA_TYPE > Point<DATA_TYPE, 3> gmtl::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 23 of file LineSegOps.h.

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

9.1.3.30 void gmtl::GaussPointsFit (int *iQuantity*, const Point3 ∗ *akPoint*, Point3 & *rkCenter*, Vec3 *akAxis[3]*, float *afExtent[3]*)

Definition at line 47 of file GaussPointsFit.h.

```
{
    // 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);
}
```

9.1.3.31 bool gmtl::GaussPointsFit (int *iQuantity,* const Vec3 ∗ *akPoint,* const bool ∗ *abValid,* Vec3 & *rkCenter,* Vec3 *akAxis[3],* float *afExtent[3]*)

Definition at line 110 of file GaussPointsFit.h.

```
{
    // 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;
}
```

9.1.3.32 const char∗ gmtl::getVersion ( ) [inline]

Definition at line 111 of file Version.h.

```
{
   return GMTL_XSTR(GMTL_VERSION_STRING);
}
```

9.1.3.33 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& gmtl::identity (Matrix< DATA_TYPE, ROWS, COLS > & *result*) [inline]

Make identity matrix out the matrix.

Postcondition

Every element is 0 except the matrix's diagonal, whose elements are 1.

Definition at line 28 of file MatrixOps.h.

```
{
   if(result.mState != Matrix<DATA_TYPE, ROWS, COLS>::IDENTITY)   // if not al
   ready 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;
}
```

9.1.3.34 template<class DATA_TYPE > bool gmtl::intersect (const AABox< DATA_TYPE > & *box1*, const AABox< DATA_TYPE > & *box2*)

Tests if the given AABoxes intersect with each other.

Sharing an edge IS considered intersection by this algorithm.

Parameters

box1 the first AA box to test

box2 the second AA box to test

Returns

true if the boxes intersect; false otherwise

Definition at line 35 of file Intersection.h.

```
{
   // Look for a separating axis on each box for each axis
   if (box1.getMin()[0] > box2.getMax()[0])  return false;
   if (box1.getMin()[1] > box2.getMax()[1])  return false;
   if (box1.getMin()[2] > box2.getMax()[2])  return false;

   if (box2.getMin()[0] > box1.getMax()[0])  return false;
   if (box2.getMin()[1] > box1.getMax()[1])  return false;
   if (box2.getMin()[2] > box1.getMax()[2])  return false;

   // No separating axis ... they must intersect
   return true;
}
```

9.1.3.35 template<class DATA_TYPE > bool gmtl::intersect (const AABox< DATA_TYPE > & *box1*, const Vec< DATA_TYPE, 3 > & *path1*, const AABox< DATA_TYPE > & *box2*, const Vec< DATA_TYPE, 3 > & *path2*, DATA_TYPE & *firstContact*, DATA_TYPE & *secondContact*)

Tests if the given AABoxes intersect if moved along the given paths.

Using the AABox sweep test, the normalized time of the first and last points of contact are found.

Parameters

box1 the first box to test

path1 the path the first box should travel along

box2 the second box to test

path2 the path the second box should travel along

firstContact set to the normalized time of the first point of contact

secondContact set to the normalized time of the second point of contact

Returns

true if the boxes intersect at any time; false otherwise

Definition at line 90 of file Intersection.h.

```
{
   // Algorithm taken from Gamasutra's article, "Simple Intersection Test for
   // Games" - http://www.gamasutra.com/features/19991018/Gomez_3.htm
   //
   // This algorithm is solved from the frame of reference of box1

   // Get the relative path (in normalized time)
   Vec<DATA_TYPE, 3> path = path2 - path1;

   // The first time of overlap along each axis
   Vec<DATA_TYPE, 3> overlap1(DATA_TYPE(0), DATA_TYPE(0), DATA_TYPE(0));

   // The second time of overlap along each axis
   Vec<DATA_TYPE, 3> overlap2(DATA_TYPE(1), DATA_TYPE(1), DATA_TYPE(1));

   // Check if the boxes already overlap
   if (gmtl::intersect(box1, box2))
   {
      firstContact = secondContact = DATA_TYPE(0);
      return true;
   }

   // Find the possible first and last times of overlap along each axis
   for (int i=0; i<3; ++i)
   {
      if ((box1.getMax()[i] < box2.getMin()[i]) && (path[i] < DATA_TYPE(0)))
      {
         overlap1[i] = (box1.getMax()[i] - box2.getMin()[i]) / path[i];
      }
      else if ((box2.getMax()[i] < box1.getMin()[i]) && (path[i] > DATA_TYPE(0
   )))
      {
         overlap1[i] = (box1.getMin()[i] - box2.getMax()[i]) / path[i];
      }

      if ((box2.getMax()[i] > box1.getMin()[i]) && (path[i] < DATA_TYPE(0)))
      {
         overlap2[i] = (box1.getMin()[i] - box2.getMax()[i]) / path[i];
      }
      else if ((box1.getMax()[i] > box2.getMin()[i]) && (path[i] > DATA_TYPE(0
   )))
      {
         overlap2[i] = (box1.getMax()[i] - box2.getMin()[i]) / path[i];
      }
```

```
    }

    // Calculate the first time of overlap
    firstContact = Math::Max(overlap1[0], overlap1[1], overlap1[2]);

    // Calculate the second time of overlap
    secondContact = Math::Min(overlap2[0], overlap2[1], overlap2[2]);

    // There could only have been a collision if the first overlap time
    // occurred before the second overlap time
    return firstContact <= secondContact;
}
```

9.1.3.36 template<class DATA_TYPE > bool gmtl::intersect (const AABox< DATA_TYPE > & *box*, const Point< DATA_TYPE, 3 > & *point*)

Tests if the given AABox and point intersect with each other.

On an edge IS considered intersection by this algorithm.

Parameters

box the box to test

point the point to test

Returns

true if the point is within the box's bounds; false otherwise

Definition at line 60 of file Intersection.h.

```
{
    // Look for a separating axis on each box for each axis
    if (box.getMin()[0] > point[0])  return false;
    if (box.getMin()[1] > point[1])  return false;
    if (box.getMin()[2] > point[2])  return false;

    if (point[0] > box.getMax()[0])  return false;
    if (point[1] > box.getMax()[1])  return false;
    if (point[2] > box.getMax()[2])  return false;

    // they must intersect
    return true;
}
```

9.1.3.37 template< class DATA_TYPE > bool gmtl::intersect (const AABox< DATA_TYPE > & *box*, const LineSeg< DATA_TYPE > & *seg*, unsigned int & *numHits*, DATA_TYPE & *tIn*, DATA_TYPE & *tOut*)

Given a line segment and an axis-aligned bounding box, returns whether the line intersects the box, and if so, `tIn` and `tOut` are set to the parametric terms on the line segment where the segment enters and exits the box respectively.

Since

0.4.11

Definition at line 284 of file Intersection.h.

```
{
   numHits = 0;
   bool result = intersectAABoxRay(box, seg, tIn, tOut);
   if (tIn < 0.0 && tOut > 1.0)
   {
  return false;
   }
   if ( result )
   {
      // If tIn is less than 0, then the origin of the line segment is
      // inside the bounding box (not on an edge)but the endpoint is
      // outside.
      if ( tIn < DATA_TYPE(0) )
      {
         numHits = 1;
         tIn = tOut;
      }
      // If tIn is less than 0, then the origin of the line segment is
      // outside the bounding box but the endpoint is inside (not on an
      // edge).
      else if ( tOut > DATA_TYPE(1) )
      {
         numHits = 1;
         tOut = tIn;
      }
      // Otherwise, the line segement intersects the bounding box in two
      // places. tIn and tOut reflect those points of intersection.
      else
      {
         numHits = 2;
      }
   }

   return result;
}
```

9.1.3.38 template<class DATA_TYPE > bool gmtl::intersect (const LineSeg< DATA_TYPE > & *seg,* const AABox< DATA_TYPE > & *box,* unsigned int & *numHits,* DATA_TYPE & *tIn,* DATA_TYPE & *tOut*)

Given a line segment and an axis-aligned bounding box, returns whether the line intersects the box, and if so, `tIn` and `tOut` are set to the parametric terms on the line segment where the segment enters and exits the box respectively.

Since

0.4.11

Definition at line 331 of file Intersection.h.

```
{
   return intersect(box, seg, numHits, tIn, tOut);
}
```

9.1.3.39 template<class DATA_TYPE > bool gmtl::intersect (const AABox< DATA_TYPE > & *box,* const Ray< DATA_TYPE > & *ray,* unsigned int & *numHits,* DATA_TYPE & *tIn,* DATA_TYPE & *tOut*)

Given a ray and an axis-aligned bounding box, returns whether the ray intersects the box, and if so, `tIn` and `tOut` are set to the parametric terms on the ray where it enters and exits the box respectively.

Since

0.4.11

Definition at line 346 of file Intersection.h.

```
{
   numHits = 0;

   bool result = intersectAABoxRay(box, ray, tIn, tOut);

   if ( result )
   {
      // Ray is inside the box.
      if ( tIn < DATA_TYPE(0) )
      {
         tIn = tOut;
         numHits = 1;
      }
      else
      {
         numHits = 2;
```

```
        }
    }

    return result;
}
```

9.1.3.40 template<class DATA_TYPE > bool gmtl::intersect (const Sphere< DATA_TYPE > & *sph1*, const Vec< DATA_TYPE, 3 > & *path1*, const Sphere< DATA_TYPE > & *sph2*, const Vec< DATA_TYPE, 3 > & *path2*, DATA_TYPE & *firstContact*, DATA_TYPE & *secondContact*)

Tests if the given Spheres intersect if moved along the given paths.

Using the Sphere sweep test, the normalized time of the first and last points of contact are found.

Parameters

sph1 the first sphere to test

path1 the path the first sphere should travel along

sph2 the second sphere to test

path2 the path the second sphere should travel along

firstContact set to the normalized time of the first point of contact

secondContact set to the normalized time of the second point of contact

Returns

true if the spheres intersect; false otherwise

Definition at line 400 of file Intersection.h.

```
{
    // Algorithm taken from Gamasutra's article, "Simple Intersection Test for
    // Games" - http://www.gamasutra.com/features/19991018/Gomez_2.htm
    //
    // This algorithm is solved from the frame of reference of sph1

    // Get the relative path (in normalized time)
    const Vec<DATA_TYPE, 3> path = path2 - path1;

    // Get the vector from sph1's starting point to sph2's starting point
    const Vec<DATA_TYPE, 3> start_offset = sph2.getCenter() - sph1.getCenter();


    // Compute the sum of the radii
    const DATA_TYPE radius_sum = sph1.getRadius() + sph2.getRadius();
```

```
    // u*u coefficient
    const DATA_TYPE a = dot(path, path);

    // u coefficient
    const DATA_TYPE b = DATA_TYPE(2) * dot(path, start_offset);

    // constant term
    const DATA_TYPE c = dot(start_offset, start_offset) - radius_sum * radius_s
    um;

    // Check if they're already overlapping
    if (dot(start_offset, start_offset) <= radius_sum * radius_sum)
    {
       firstContact = secondContact = DATA_TYPE(0);
       return true;
    }

    // Find the first and last points of intersection
    if (Math::quadraticFormula(firstContact, secondContact, a, b, c))
    {
       // Swap first and second contacts if necessary
       if (firstContact > secondContact)
       {
          std::swap(firstContact, secondContact);
          return true;
       }
    }

    return false;
}
```

9.1.3.41 template<class DATA_TYPE > bool gmtl::intersect (const AABox< DATA_TYPE > & *box*, const Sphere< DATA_TYPE > & *sph*)

Tests if the given AABox and Sphere intersect with each other.

On an edge IS considered intersection by this algorithm.

Parameters

box the box to test

sph the sphere to test

Returns

true if the items intersect; false otherwise

Definition at line 458 of file Intersection.h.

```
{
   DATA_TYPE dist_sqr = DATA_TYPE(0);
```

```
    // Compute the square of the distance from the sphere to the box
    for (int i=0; i<3; ++i)
    {
       if (sph.getCenter()[i] < box.getMin()[i])
       {
          DATA_TYPE s = sph.getCenter()[i] - box.getMin()[i];
          dist_sqr += s*s;
       }
       else if (sph.getCenter()[i] > box.getMax()[i])
       {
          DATA_TYPE s = sph.getCenter()[i] - box.getMax()[i];
          dist_sqr += s*s;
       }
    }

    return dist_sqr <= (sph.getRadius()*sph.getRadius());
 }
```

9.1.3.42 template<class DATA_TYPE > bool gmtl::intersect (const Sphere< DATA_TYPE > & *sph*, const AABox< DATA_TYPE > & *box*)

Tests if the given AABox and Sphere intersect with each other.

On an edge IS considered intersection by this algorithm.

Parameters

sph the sphere to test

box the box to test

Returns

true if the items intersect; false otherwise

Definition at line 490 of file Intersection.h.

```
 {
    return gmtl::intersect(box, sph);
 }
```

9.1.3.43 template<class DATA_TYPE > bool gmtl::intersect (const Sphere< DATA_TYPE > & *sphere*, const Point< DATA_TYPE, 3 > & *point*)

intersect point/sphere.

Parameters

point the point to test

sphere the sphere to test

Returns

true if point is in or on sphere

Definition at line 502 of file Intersection.h.

```
{
   gmtl::Vec<DATA_TYPE, 3> offset = point - sphere.getCenter();
   DATA_TYPE dist = lengthSquared( offset ) - sphere.getRadius() * sphere.getR
   adius();

   // point is inside the sphere when true
   return  dist <= 0;
}
```

9.1.3.44 template< typename T > bool gmtl::intersect (const Sphere< T > & *sphere*, const Ray< T > & *ray*, int & *numhits*, T & *t0*, T & *t1*) [inline]

intersect ray/sphere-shell (not volume).

only register hits with the surface of the sphere. note: after calling this, you can find the intersection point with: ray.getOrigin() + ray.getDir() $*$ t

Parameters

ray the ray to test

sphere the sphere to test

Returns

returns intersection point in t, and the number of hits
numhits, t0, t1 are undefined if return value is false

Definition at line 522 of file Intersection.h.

```
{
   numhits = -1;

   // set up quadratic Q(t) = a*t^2 + 2*b*t + c
   const Vec<T, 3> offset = ray.getOrigin() - sphere.getCenter();
   const T a = lengthSquared( ray.getDir() );
   const T b = dot( offset, ray.getDir() );
   const T c = lengthSquared( offset ) - sphere.getRadius() * sphere.getRadius
   ();
```

```
    // no intersection if Q(t) has no real roots
    const T discriminant = b * b - a * c;
    if (discriminant < 0.0f)
    {
       numhits = 0;
       return false;
    }
    else if (discriminant > 0.0f)
    {
       T root = Math::sqrt( discriminant );
       T invA = T(1) / a;
       t0 = (-b - root) * invA;
       t1 = (-b + root) * invA;

       // assert: t0 < t1 since A > 0

       if (t0 >= T(0))
       {
          numhits = 2;
          return true;
       }
       else if (t1 >= T(0))
       {
          numhits = 1;
          t0 = t1;
          return true;
       }
       else
       {
          numhits = 0;
          return false;
       }
    }
    else
    {
       t0 = -b / a;
       if (t0 >= T(0))
       {
          numhits = 1;
          return true;
       }
       else
       {
          numhits = 0;
          return false;
       }
    }
 }
```

9.1.3.45 template<class DATA_TYPE > bool gmtl::intersect (const Ray< DATA_TYPE > & *ray*, const AABox< DATA_TYPE > & *box*, unsigned int & *numHits*, DATA_TYPE & *tIn*, DATA_TYPE & *tOut*)

Given a ray and an axis-aligned bounding box, returns whether the ray intersects the box, and if so, `tIn` and `tOut` are set to the parametric terms on the ray where it enters and exits the box respectively.

Since

0.4.11

Definition at line 379 of file Intersection.h.

```
{
   return intersect(box, ray, numHits, tIn, tOut);
}
```

9.1.3.46 template<typename T > bool gmtl::intersect (const Sphere< T > & *sphere*, const LineSeg< T > & *lineseg*, int & *numhits*, T & *t0*, T & *t1*) `[inline]`

intersect LineSeg/Sphere-shell (not volume).

does intersection on sphere surface, point inside sphere doesn't count as an intersection returns intersection point(s) in t find intersection point(s) with: ray.getOrigin() + ray.getDir() $*$ t numhits, t0, t1 are undefined if return value is false

Definition at line 590 of file Intersection.h.

```
{
   if (intersect( sphere, Ray<T>( lineseg ), numhits, t0, t1 ))
   {
      // throw out hits that are past 1 in segspace (off the end of the linese
   g)
      while (0 < numhits && 1.0f < t0)
      {
         --numhits;
         t0 = t1;
      }
      if (2 == numhits && 1.0f < t1)
      {
         --numhits;
      }
      return 0 < numhits;
   }
   else
   {
      return false;
   }
}
```

9.1.3.47 template<class DATA_TYPE > bool gmtl::intersect (const Plane< DATA_TYPE > & *plane*, const LineSeg< DATA_TYPE > & *seg*, DATA_TYPE & *t*)

Tests if the given plane and lineseg intersect with each other.

Parameters

ray - the lineseg

plane - the Plane

t - t gives you the intersection point: isect_point = lineseg.origin + lineseg.dir $*$ t

Returns

true if the lineseg intersects the plane.

Definition at line 750 of file Intersection.h.

```
{
   bool res(intersect(plane, static_cast<Ray<DATA_TYPE> >(seg), t));
   return res && t <= (DATA_TYPE)1.0;
}
```

9.1.3.48 template<class DATA_TYPE > bool gmtl::intersect (const Tri< DATA_TYPE > & *tri*, const Ray< DATA_TYPE > & *ray*, float & *u*, float & *v*, float & *t*)

Tests if the given triangle and ray intersect with each other.

Parameters

tri - the triangle (ccw ordering)

ray - the ray

u,v - tangent space u/v coordinates of the intersection

t - an indicator of the intersection location

Postcondition

t gives you the intersection point: isect = ray.dir $*$ t + ray.origin

Returns

true if the ray intersects the triangle.

See also

from http://www.acm.org/jgt/papers/MollerTrumbore97/code.html

Definition at line 769 of file Intersection.h.

```
{
   const float EPSILON = (DATA_TYPE)0.00001f;
   Vec<DATA_TYPE, 3> edge1, edge2, tvec, pvec, qvec;
   float det,inv_det;

   /* find vectors for two edges sharing vert0 */
   edge1 = tri[1] - tri[0];
   edge2 = tri[2] - tri[0];

   /* begin calculating determinant - also used to calculate U parameter */
   gmtl::cross( pvec, ray.getDir(), edge2 );

   /* if determinant is near zero, ray lies in plane of triangle */
   det = gmtl::dot( edge1, pvec );

   if (det < EPSILON)
      return false;

   /* calculate distance from vert0 to ray origin */
   tvec = ray.getOrigin() - tri[0];

   /* calculate U parameter and test bounds */
   u = gmtl::dot( tvec, pvec );
   if (u < 0.0 || u > det)
      return false;

   /* prepare to test V parameter */
   gmtl::cross( qvec, tvec, edge1 );

   /* calculate V parameter and test bounds */
   v = gmtl::dot( ray.getDir(), qvec );
   if (v < 0.0 || u + v > det)
      return false;

   /* calculate t, scale parameters, ray intersects triangle */
   t = gmtl::dot( edge2, qvec );
   inv_det = ((DATA_TYPE)1.0) / det;
   t *= inv_det;
   u *= inv_det;
   v *= inv_det;

   // test if t is within the ray boundary (when t >= 0)
   return t >= (DATA_TYPE)0;
}
```

9.1.3.49 template<class DATA_TYPE > bool gmtl::intersect (const Plane< DATA_TYPE > & *plane*, const Ray< DATA_TYPE > & *ray*, DATA_TYPE & *t*)

Tests if the given plane and ray intersect with each other.

Parameters

ray - the Ray

plane - the Plane

t - t gives you the intersection point: isect_point = ray.origin + ray.dir $*$ t

Returns

true if the ray intersects the plane.

Note

If ray is parallel to plane: t=0, ret:true -> on plane, ret:false -> No hit

Definition at line 719 of file Intersection.h.

```
{
   const DATA_TYPE eps(0.00001f);

   // t = -(nP + d)
   Vec<DATA_TYPE, 3> N( plane.getNormal() );
   DATA_TYPE denom( dot(N,ray.getDir()) );
   if(gmtl::Math::abs(denom) < eps)    // Ray parallel to plane
   {
      t = 0;
      if(distance(plane, ray.mOrigin) < eps)     // Test for ray on plane
      { return true; }
      else
      { return false; }
   }
   t = dot( N, Vec<DATA_TYPE,3>(N * plane.getOffset() - ray.getOrigin()) ) / d
   enom;

   return (DATA_TYPE)0 <= t;
}
```

9.1.3.50 template<class DATA_TYPE > bool gmtl::intersect (const Tri< DATA_TYPE > & *tri*, const LineSeg< DATA_TYPE > & *lineseg*, DATA_TYPE & *u*, DATA_TYPE & *v*, DATA_TYPE & *t*)

Tests if the given triangle and line segment intersect with each other.

Parameters

tri - the triangle (ccw ordering)

lineseg - the line segment

u,v - tangent space u/v coordinates of the intersection

t - an indicator of the intersection point

Postcondition

t gives you the intersection point: isect = lineseg.getDir() $*$ t + lineseg.getOrigin()

Returns

true if the line segment intersects the triangle.

Definition at line 898 of file Intersection.h.

```
{
   const DATA_TYPE eps = (DATA_TYPE)0.0001010101;
   DATA_TYPE l = length( lineseg.getDir() );
   if (eps < l)
   {
      Ray<DATA_TYPE> temp( lineseg.getOrigin(), lineseg.getDir() );
      bool result = intersect( tri, temp, u, v, t );
      return result && t <= (DATA_TYPE)1.0;
   }
   else
   {  return false; }
}
```

9.1.3.51 template<class DATA_TYPE > bool gmtl::intersectAABoxRay (const AABox< DATA_TYPE > & *box*, const Ray< DATA_TYPE > & *ray*, DATA_TYPE & *tIn*, DATA_TYPE & *tOut*)

Given an axis-aligned bounding box and a ray (or subclass thereof), returns whether the ray intersects the box, and if so, `tIn` and `tOut` are set to the parametric terms on the ray where the segment enters and exits the box respectively.

The implementation of this function comes from the book *Geometric Tools for Computer Graphics*, pages 626-630.

Note

Internal function for performing an intersection test between an axis-aligned bounding box and a ray. User code should not call this function directly. It is used to capture the common code between the gmtl::Ray<T> and gmtl::LineSeg<T> overloads of gmtl::intersect() when intersecting with a gmtl::AABox<T>.

Definition at line 164 of file Intersection.h.

```
{
   tIn  = -(std::numeric_limits<DATA_TYPE>::max)();
   tOut = (std::numeric_limits<DATA_TYPE>::max)();
   DATA_TYPE t0, t1;
   const DATA_TYPE epsilon(0.0000001);
```

```
// YZ plane.
if ( gmtl::Math::abs(ray.mDir[0]) < epsilon )
{
   // Ray parallel to plane.
   if ( ray.mOrigin[0] < box.mMin[0] || ray.mOrigin[0] > box.mMax[0] )
   {
      return false;
   }
}

// XZ plane.
if ( gmtl::Math::abs(ray.mDir[1]) < epsilon )
{
   // Ray parallel to plane.
   if ( ray.mOrigin[1] < box.mMin[1] || ray.mOrigin[1] > box.mMax[1] )
   {
      return false;
   }
}

// XY plane.
if ( gmtl::Math::abs(ray.mDir[2]) < epsilon )
{
   // Ray parallel to plane.
   if ( ray.mOrigin[2] < box.mMin[2] || ray.mOrigin[2] > box.mMax[2] )
   {
      return false;
   }
}

// YZ plane.
t0 = (box.mMin[0] - ray.mOrigin[0]) / ray.mDir[0];
t1 = (box.mMax[0] - ray.mOrigin[0]) / ray.mDir[0];

if ( t0 > t1 )
{
   std::swap(t0, t1);
}

if ( t0 > tIn )
{
   tIn = t0;
}
if ( t1 < tOut )
{
   tOut = t1;
}

if ( tIn > tOut || tOut < DATA_TYPE(0) )
{
   return false;
}

// XZ plane.
t0 = (box.mMin[1] - ray.mOrigin[1]) / ray.mDir[1];
t1 = (box.mMax[1] - ray.mOrigin[1]) / ray.mDir[1];
```

```
      if ( t0 > t1 )
      {
         std::swap(t0, t1);
      }

      if ( t0 > tIn )
      {
         tIn = t0;
      }
      if ( t1 < tOut )
      {
         tOut = t1;
      }

      if ( tIn > tOut || tOut < DATA_TYPE(0) )
      {
         return false;
      }

      // XY plane.
      t0 = (box.mMin[2] - ray.mOrigin[2]) / ray.mDir[2];
      t1 = (box.mMax[2] - ray.mOrigin[2]) / ray.mDir[2];

      if ( t0 > t1 )
      {
         std::swap(t0, t1);
      }

      if ( t0 > tIn )
      {
         tIn = t0;
      }
      if ( t1 < tOut )
      {
         tOut = t1;
      }

      if ( tIn > tOut || tOut < DATA_TYPE(0) )
      {
         return false;
      }

      return true;
   }
```

9.1.3.52 template<class DATA_TYPE > bool gmtl::intersectDoubleSided (const Tri< DATA_TYPE > & *tri*, const Ray< DATA_TYPE > & *ray*, DATA_TYPE & *u*, DATA_TYPE & *v*, DATA_TYPE & *t*)

Tests if the given triangle intersects with the given ray, from both sides.

Parameters

tri The triangle (ccw ordering).

ray The ray.

u Tangent space u-coordinate of the intersection.

v Tangent space v-coordinate of the intersection.

t An indicator of the intersection location.

Postcondition

`t` gives you the intersection point:

```
isect = ray.dir * t + ray.origin
```

Returns

true if the ray intersects the triangle.

See also

from `http://www.acm.org/jgt/papers/MollerTrumbore97/code.html`

Definition at line 833 of file Intersection.h.

```
{
   const DATA_TYPE EPSILON = (DATA_TYPE)0.00001f;
   Vec<DATA_TYPE, 3> edge1, edge2, tvec, pvec, qvec;
   DATA_TYPE det, inv_det;

   // Find vectors for two edges sharing vert0.
   edge1 = tri[1] - tri[0];
   edge2 = tri[2] - tri[0];

   // Begin calculating determinant - also used to calculate U parameter.
   gmtl::cross(pvec, ray.getDir(), edge2);

   // If determinant is near zero, ray lies in plane of triangle.
   det = gmtl::dot( edge1, pvec );

   if ( det < EPSILON && det > -EPSILON )
   {
      return false;
   }

   // calculate distance from vert0 to ray origin>
   tvec = ray.getOrigin() - tri[0];

   // Calc inverse deteriminant.
   inv_det = ((DATA_TYPE)1.0) / det;

   // Calculate U parameter and test bounds.
   u =  inv_det * gmtl::dot(tvec, pvec);
   if ( u < 0.0 || u > 1.0 )
   {
      return false;
   }
```

```
    // Prepare to test V parameter.
    gmtl::cross(qvec, tvec, edge1);

    // Calculate V parameter and test bounds.
    v = inv_det * gmtl::dot(ray.getDir(), qvec);
    if (v < 0.0 || u + v > 1.0)
    {
       return false;
    }

    // Calculate t, scale parameters, ray intersects triangle.
    t = inv_det * gmtl::dot(edge2, qvec);

    // Test if t is within the ray boundary (when t >= 0).
    return t >= (DATA_TYPE)0;
 }
```

9.1.3.53 template< typename T > bool gmtl::intersectVolume (const Sphere< T > & *sphere*, const LineSeg< T > & *ray*, int & *numhits*, T & *t0*, T & *t1*) `[inline]`

intersect lineseg/sphere-volume.

register hits with both the surface and when end points land on the interior of the sphere. note: after calling this, you can find the intersection point with: ray.getOrigin() + ray.getDir() * t

Parameters

ray the lineseg to test

sphere the sphere to test

Returns

returns intersection point in t, and the number of hits
numhits, t0, t1 are undefined if return value is false

Definition at line 623 of file Intersection.h.

```
 {
    bool result = intersect( sphere, ray, numhits, t0, t1 );
    if (result && numhits == 2)
    {
       return true;
    }
    // todo: make this faster (find an early out) since 1 or 0 hits is the comm
    on case.
    // volume test has some additional checks before we can throw it out becaus
    e
```

```
    // one of both points may be inside the volume, so we want to return hits f
    or those as well...
    else // 1 or 0 hits.
    {
       const T rsq = sphere.getRadius() * sphere.getRadius();
       const Vec<T, 3> dist = ray.getOrigin() - sphere.getCenter();
       const T a = lengthSquared( dist ) - rsq;
       const T b = lengthSquared( gmtl::Vec<T,3>(dist + ray.getDir()) ) - rsq;

       bool inside1 = a <= T( 0 );
       bool inside2 = b <= T( 0 );

       // one point is inside
       if (numhits == 1 && inside1 && !inside2)
       {
          t1 = t0;
          t0 = T(0);
          numhits = 2;
          return true;
       }
       else if (numhits == 1 && !inside1 && inside2)
       {
          t1 = T(1);
          numhits = 2;
          return true;
       }
       // maybe both points are inside?
       else if (inside1 && inside2) // 0 hits.
       {
          t0 = T(0);
          t1 = T(1);
          numhits = 2;
          return true;
       }
    }
    return result;
 }
```

9.1.3.54 template< typename T > bool gmtl::intersectVolume (const Sphere< T > & *sphere*, const Ray< T > & *ray*, int & *numhits*, T & *t0*, T & *t1*) `[inline]`

intersect ray/sphere-volume.

register hits with both the surface and when the origin lands in the interior of the sphere. note: after calling this, you can find the intersection point with: ray.getOrigin() + ray.getDir() ∗ t

Parameters

ray the ray to test

sphere the sphere to test

Returns

returns intersection point in t, and the number of hits
numhits, t0, t1 are undefined if return value is false

Definition at line 680 of file Intersection.h.

```
{
   bool result = intersect( sphere, ray, numhits, t0, t1 );
   if (result && numhits == 2)
   {
      return true;
   }
   else
   {
      const T rsq = sphere.getRadius() * sphere.getRadius();
      const Vec<T, 3> dist = ray.getOrigin() - sphere.getCenter();
      const T a = lengthSquared( dist ) - rsq;

      bool inside = a <= T( 0 );

      // start point is inside
      if (inside)
      {
         t1 = t0;
         t0 = T(0);
         numhits = 2;
         return true;
      }
   }
   return result;
}
```

9.1.3.55 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& gmtl::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(). : result' = inv(result) : If inversion failed, then error bit is set within the Matrix.

Definition at line 642 of file MatrixOps.h.

```
{
```

```
    if (src.mState == Matrix<DATA_TYPE, ROWS, COLS>::IDENTITY )
       return result = src;
    else if (src.mState == Matrix<DATA_TYPE, ROWS, COLS>::TRANS)
       return invertTrans( result, src );
    else if (src.mState == Matrix<DATA_TYPE, ROWS, COLS>::ORTHOGONAL)
       return invertOrthogonal( result, src );
    else if (src.mState == Matrix<DATA_TYPE, ROWS, COLS>::AFFINE ||
             src.mState == (Matrix<DATA_TYPE, ROWS, COLS>::AFFINE | Matrix<DATA
    _TYPE, ROWS, COLS>::NON_UNISCALE))
       return invertAffine( result, src );
    else
       return invertFull_orig( result, src );
 }
```

9.1.3.56 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& gmtl::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(). : result' = inv(result) : If inversion failed, then error bit is set within the Matrix.

Definition at line 668 of file MatrixOps.h.

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

9.1.3.57 template<typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::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 390 of file QuatOps.h.

```
{
   // from game programming gems p198
   // 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;
}
```

9.1.3.58 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& gmtl::invertAffine (Matrix< DATA_TYPE, ROWS, COLS > & *result*, const Matrix< DATA_TYPE, ROWS, COLS > & *source*) `[inline]`

affine matrix inversion.

Matrix inversion that acts on a 4x3 affine matrix (matrix with only trans, rot, uniform scale) Check for error with Matrix::isError().

Precondition

: 3x3, 4x3, 4x4 matrices only : result' = inv(result) : If inversion failed, then error bit is set within the Matrix.

Definition at line 322 of file MatrixOps.h.

```
{
   static const float eps = 0.00000001f;

   // in case &result is == &source... :(
   Matrix<DATA_TYPE, ROWS, COLS> src = source;

   // The rotational part of the matrix is simply the transpose of the
   // original matrix.
   for (int x = 0; x < 3; ++x)
   for (int y = 0; y < 3; ++y)
   {
      result[x][y] = src[y][x];
   }
```

```
// do non-uniform scale inversion
if (src.mState & Matrix<DATA_TYPE, ROWS, COLS>::NON_UNISCALE)
{
   DATA_TYPE l0 = gmtl::lengthSquared( gmtl::Vec<DATA_TYPE, 3>( result[0][0
], result[0][1], result[0][2] ) );
   DATA_TYPE l1 = gmtl::lengthSquared( gmtl::Vec<DATA_TYPE, 3>( result[1][0
], result[1][1], result[1][2] ) );
   DATA_TYPE l2 = gmtl::lengthSquared( gmtl::Vec<DATA_TYPE, 3>( result[2][0
], result[2][1], result[2][2] ) );
   if (gmtl::Math::abs( l0 ) > eps) l0 = 1.0f / l0;
   if (gmtl::Math::abs( l1 ) > eps) l1 = 1.0f / l1;
   if (gmtl::Math::abs( l2 ) > eps) l2 = 1.0f / l2;
   // apply the inverse scale to the 3x3
   // for each axis: normalize it (1/length), and then mult by inverse scal
e (1/length)
   result[0][0] *= l0;
   result[0][1] *= l0;
   result[0][2] *= l0;
   result[1][0] *= l1;
   result[1][1] *= l1;
   result[1][2] *= l1;
   result[2][0] *= l2;
   result[2][1] *= l2;
   result[2][2] *= l2;
}

// handle matrices with translation
if (COLS == 4)
{
   // The right column vector of the matrix should always be [ 0 0 0 s ]
   // this represents some shear values
   result[3][0] = result[3][1] = result[3][2] = 0;

   // The translation components of the original matrix.
   const DATA_TYPE& tx = src[0][3];
   const DATA_TYPE& ty = src[1][3];
   const DATA_TYPE& tz = src[2][3];


   // Rresult = -(Tm * Rm) to get the translation part of the inverse
   if (ROWS == 4)
   {
      // invert scale.
      const DATA_TYPE tw = (gmtl::Math::abs( src[3][3] ) > eps) ? 1.0f / sr
c[3][3] : 0.0f;

      // handle uniform scale in Nx4 matrices
      result[0][3] = -( result[0][0] * tx + result[0][1] * ty + result[0][2
] * tz ) * tw;
      result[1][3] = -( result[1][0] * tx + result[1][1] * ty + result[1][2
] * tz ) * tw;
      result[2][3] = -( result[2][0] * tx + result[2][1] * ty + result[2][2
] * tz ) * tw;
      result[3][3] = tw;
   }
   else if (ROWS == 3)
   {
```

```
        result[0][3] = -( result[0][0] * tx + result[0][1] * ty + result[0][2
    ] * tz );
        result[1][3] = -( result[1][0] * tx + result[1][1] * ty + result[1][2
    ] * tz );
        result[2][3] = -( result[2][0] * tx + result[2][1] * ty + result[2][2
    ] * tz );
      }
    }


    result.mState = src.mState;

    return result;
}
```

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

Invert method.

Calls invertFull_orig to do the work.

Definition at line 626 of file MatrixOps.h.

```
{
    return invertFull_orig(result,src);
}
```

9.1.3.60 template<typename DATA_TYPE , unsigned SIZE> Matrix<DATA_TYPE, SIZE, SIZE>& gmtl::invertFull_GJ (Matrix< DATA_TYPE, SIZE, SIZE > & *result*, const Matrix< DATA_TYPE, SIZE, SIZE > & *src*) `[inline]`

Full matrix inversion using Gauss-Jordan elimination.

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

Definition at line 406 of file MatrixOps.h.

```
{
    //gmtlASSERT( ROWS == COLS && "invertFull only works with nxn matrices" );

    const DATA_TYPE pivot_eps(1e-20);         // Epsilon for the pivot value te
    st (delta with test against zero)
```

```
// Computer inverse of matrix using a Gaussian-Jordan elimination.
// Uses max pivot at each point
// See: "Essential Mathmatics for Games" for description

// Do this invert in place
result = src;
unsigned swapped[SIZE];       // Track swaps. swapped[3] = 2 means that row
 3 was swapped with row 2

unsigned pivot;

// --- Gaussian elimination step --- //
// For each column and row
for(pivot=0; pivot<SIZE;++pivot)
{
   unsigned    pivot_row(pivot);
   DATA_TYPE   pivot_value(gmtl::Math::abs(result(pivot_row, pivot)));    /
/ Initialize to beginning of current row

   // find pivot row - (max pivot element)
   for(unsigned pr=pivot+1;pr<SIZE;++pr)
   {
      const DATA_TYPE cur_val(gmtl::Math::abs(result(pr,pivot)));   // get
value at current point
      if (cur_val > pivot_value)
      {
         pivot_row = pr;
         pivot_value = cur_val;
      }
   }

   if(gmtl::Math::isEqual(DATA_TYPE(0),pivot_value,pivot_eps))
   {
      std::cerr << "*** pivot = " << pivot_value << " in mat_inv. ***\n";
      result.setError();
      return result;
   }

   // Check for swap of pivot rows
   swapped[pivot] = pivot_row;
   if(pivot_row != pivot)
   {
      for(unsigned c=0;c<SIZE;++c)
      {  std::swap(result(pivot,c), result(pivot_row,c)); }
   }
   // ASSERT: row to use in now in "row" (check that row starts with max pi
vot value found)
   gmtlASSERT(gmtl::Math::isEqual(pivot_value,gmtl::Math::abs(result(pivot,
pivot)),DATA_TYPE(0.00001)));

   // Compute pivot factor
   const DATA_TYPE mult_factor(1.0f/pivot_value);

   // Set pivot row values
   for(unsigned c=0;c<SIZE;++c)
   {  result(pivot,c) *= mult_factor; }
   result(pivot,pivot) = mult_factor;    // Copy the 1/pivot since we are i
```

```
nverting in place

   // Clear pivot column in other rows (since we are in place)
   // - Subtract current row times result(r,col) so that column element bec
omes 0
   for(unsigned row=0;row<SIZE;++row)
   {
      if(row==pivot)     // Don't subtract from our row
      { continue; }

      const DATA_TYPE sub_mult_factor(result(row,pivot));

      // Clear the pivot column's element (for invers in place)
      // ends up being set to -sub_mult_factor*pivotInverse
      result(row,pivot) = 0;

      // subtract the pivot row from this row
      for(unsigned col=0;col<SIZE;++col)
      {   result(row,col) -= (sub_mult_factor*result(pivot,col)); }
   }
} // end: gaussian substitution


// Now undo the swaps in column direction in reverse order
unsigned p(SIZE);
do
{
   --p;
   gmtlASSERT(p<SIZE);

   // If row was swapped
   if(swapped[p] != p)
   {
      // Swap the column with same index
      for(unsigned r=0; r<SIZE; ++r)
      { std::swap(result(r, p), result(r, swapped[p])); }
   }
}
while(p>0);

return result;
}
```

9.1.3.61 template<typename DATA_TYPE , unsigned SIZE> Matrix<DATA_TYPE, SIZE, SIZE>& gmtl::invertFull_orig (Matrix< DATA_TYPE, SIZE, SIZE > & *result*, const Matrix< DATA_TYPE, SIZE, SIZE > & *src*) `[inline]`

full matrix inversion.

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

Definition at line 512 of file MatrixOps.h.

```
{
   /*------------------------------------------------------------------------
   --*
    | 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(SIZE);
   int    i, j, k;
   int    r[SIZE], c[SIZE], row[SIZE], col[SIZE];
   DATA_TYPE  m[SIZE][SIZE*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 = " << pivot << " 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
    result.mState = src.mState;
    return result;
 }
```

9.1.3.62 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& gmtl::invertOrthogonal (Matrix< DATA_TYPE, ROWS, COLS > & *result*, const Matrix< DATA_TYPE, ROWS, COLS > & *src*) `[inline]`

orthogonal matrix inversion.

Matrix inversion that acts on a affine matrix (matrix with only trans, rot, uniform scale) Check for error with Matrix::isError().

Precondition

: any size matrix

Postcondition

: result' = inv(result)
: If inversion failed, then error bit is set within the Matrix.

Definition at line 293 of file MatrixOps.h.

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

   // if 3x4, 2x3, etc...  can't transpose the last column
   const unsigned int size = Math::Min( ROWS, COLS );

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

9.1.3.63 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& gmtl::invertTrans (Matrix< DATA_TYPE, ROWS, COLS > & *result*, const Matrix< DATA_TYPE, ROWS, COLS > & *src*) `[inline]`

translational matrix inversion.

Matrix inversion that acts on a translational matrix (matrix with only translation) Check for error with Matrix::isError().

Precondition

: 4x3, 4x4 matrices only

Postcondition

: result' = inv(result)
: If inversion failed, then error bit is set within the Matrix.

Definition at line 271 of file MatrixOps.h.

```
{
   gmtlASSERT( ROWS == COLS || COLS == ROWS+1 && "invertTrans supports NxN or
   Nx(N-1) matrices only" );

   if (&result != &src)
      result = src; // could optimise this a little more (skip the trans copy)
   , favor simplicity for now...
   for (unsigned x = 0; x < (ROWS-1+(COLS-ROWS)); ++x)
   {
      result[x][3] = -result[x][3];
   }
   return result;
}
```

9.1.3.64 template<class DATA_TYPE > bool gmtl::isEqual (const AxisAngle< DATA_TYPE > & *a1*, const AxisAngle< DATA_TYPE > & *a2*, const DATA_TYPE *eps* = `0`) `[inline]`

Compares a1 and a2 to see if they are the same within the given epsilon tolerance.

Precondition

eps must be >= 0

Parameters

a1 the first vector

a2 the second vector

eps the epsilon tolerance value

Returns

true if a1 equals a2 within tolerance; false if they differ

Definition at line 67 of file AxisAngleOps.h.

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

   // @todo metaprogramming.
   if (!Math::isEqual( a1[0], a2[0], eps )) return false;
   if (!Math::isEqual( a1[1], a2[1], eps )) return false;
   if (!Math::isEqual( a1[2], a2[2], eps )) return false;
   if (!Math::isEqual( a1[3], a2[3], eps )) return false;
   return true;
}
```

9.1.3.65 template<class DATA_TYPE , typename ROT_ORDER > bool gmtl::isEqual (const EulerAngle< DATA_TYPE, ROT_ORDER > & *e1*, const EulerAngle< DATA_TYPE, ROT_ORDER > & *e2*, const DATA_TYPE *eps* = 0) [inline]

Compares e1 and e2 (component-wise) to see if they are the same within a given tolerance.

Precondition

eps must be >= 0

Parameters

e1 the first EulerAngle

e2 the second EulerAngle

eps the epsilon tolerance value, in radians

Returns

true if e1 is within the tolerance of e2; false if not

Definition at line 66 of file EulerAngleOps.h.

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

   // @todo metaprogramming.
   if (!Math::isEqual( e1[0], e2[0], eps )) return false;
   if (!Math::isEqual( e1[1], e2[1], eps )) return false;
   if (!Math::isEqual( e1[2], e2[2], eps )) return false;
   return true;
}
```

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

Tests 2 matrices for equality within a tolerance.

Parameters

lhs The first matrix

rhs The second matrix

eps The tolerance value

Precondition

Both matrices must be of the same size.

Returns

true if the matrices' elements are within the tolerance value of each other; false otherwise

Definition at line 726 of file MatrixOps.h.

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

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

9.1.3.67 template<class DATA_TYPE > bool gmtl::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 within a tolerance, false otherwise

Definition at line 171 of file PlaneOps.h.

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

9.1.3.68 template<class DATA_TYPE > bool gmtl::isEqual (const AABox< DATA_TYPE > & *b1*, const AABox< DATA_TYPE > & *b2*, const DATA_TYPE & *eps*) [inline]

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

Parameters

b1 the first box to compare

b2 the second box to compare

eps the tolerance value to use

Precondition

eps must be >= 0

Returns

true if their points are within the given tolerance of each other, false otherwise

Definition at line 64 of file AABoxOps.h.

```
{
   gmtlASSERT( eps >= 0 );
   return (b1.isEmpty() == b2.isEmpty()) &&
          isEqual( b1.getMin(), b2.getMin(), eps ) &&
          isEqual( b1.getMax(), b2.getMax(), eps );
}
```

9.1.3.69 template<typename DATA_TYPE > bool gmtl::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 619 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 )    );
}
```

9.1.3.70 template<class DATA_TYPE > bool gmtl::isEqual (const Ray< DATA_TYPE > & *ls1*, const Ray< 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 Ray to compare

ls2 the second Ray to compare

eps the tolerance value to use

Precondition

eps must be >= 0

Returns

true if they are equal within the tolerance, false otherwise

Definition at line 56 of file RayOps.h.

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

9.1.3.71 template<class DATA_TYPE > bool gmtl::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 within a tolerance, false otherwise

Definition at line 61 of file SphereOps.h.

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

9.1.3.72 template<class DATA_TYPE > bool gmtl::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 within the tolerance, false otherwise

Definition at line 97 of file TriOps.h.

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

9.1.3.73 template<typename POS_TYPE , typename ROT_TYPE > bool gmtl::isEqual (const Coord< POS_TYPE, ROT_TYPE > & *c1*, const Coord< POS_TYPE, ROT_TYPE > & *c2*, typename Coord< POS_TYPE, ROT_TYPE >::DataType *tol* = 0) `[inline]`

Compare two coordinate frames for equality with a given tolerance.

Parameters

c1 the first Coord

c2 the second Coord

tol the tolerance coordinate frame of the same type as c1 and c2

Returns

true if c1 is equal within a tolerance of c2, false otherwise

Definition at line 50 of file CoordOps.h.

```
{
   return bool( isEqual( c1.getPos(), c2.getPos(), tol ) &&
                isEqual( c1.getRot(), c2.getRot(), tol )     );
}
```

9.1.3.74 template<class DATA_TYPE , unsigned SIZE> bool gmtl::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 within the tolerance; false if they differ

Definition at line 603 of file VecOps.h.

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

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

9.1.3.75 template<typename DATA_TYPE > bool gmtl::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 633 of file QuatOps.h.

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

9.1.3.76 template<class DATA_TYPE > bool gmtl::isInVolume (const AABox< DATA_TYPE > & *container*, const Point< DATA_TYPE, 3 > & *pt*)

Tests if the given point is inside (or on) the surface of the given AABox volume.

Parameters

container the AABox to test against

pt the point to test with

Returns

true if pt is inside container, false otherwise

Definition at line 305 of file Containment.h.

```
{
   if (! container.isEmpty())
   {
      return ( pt[0] >= container.mMin[0] &&
               pt[1] >= container.mMin[1] &&
               pt[2] >= container.mMin[2] &&
               pt[0] <= container.mMax[0] &&
               pt[1] <= container.mMax[1] &&
               pt[2] <= container.mMax[2]);
   }
   else
   {
      return false;
   }
}
```

9.1.3.77 template< class DATA_TYPE > bool gmtl::isInVolume (const AABox< DATA_TYPE > & *container*, const AABox< DATA_TYPE > & *box*)

Tests if the given AABox is completely inside or on the surface of the given AABox container.

Parameters

container the AABox acting as the container

box the AABox that may be inside container

Returns

true if AABox is inside container, false otherwise

Definition at line 365 of file Containment.h.

```
{
   // Empty boxes don't overlap
   if (container.isEmpty() || box.isEmpty())
   {
      return false;
   }
```

```
   if (container.mMin[0] <= box.mMin[0] && container.mMax[0] >= box.mMax[0] &&
       container.mMin[1] <= box.mMin[1] && container.mMax[1] >= box.mMax[1] &&
       container.mMin[2] <= box.mMin[2] && container.mMax[2] >= box.mMax[2])
   {
      return true;
   }
   else
   {
      return false;
   }
}
```

9.1.3.78 template<class DATA_TYPE > bool gmtl::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 41 of file Containment.h.

```
{
   // 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(gmtl::Vec<DATA_TYPE,3>(pt - container.mCenter)) <= container.m
      Radius );
}
```

9.1.3.79 template<class DATA_TYPE > bool gmtl::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 61 of file Containment.h.

```
{
   // 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(gmtl::Vec<DATA_TYPE,3>(sphere.mCenter - container.mCenter)) +
      sphere.mRadius
            <= container.mRadius );
}
```

9.1.3.80 template<typename T > bool gmtl::isInVolume (const Frustum< T > & *f*, const Point< T, 3 > & *p*, unsigned int & *idx*) `[inline]`

Definition at line 495 of file Containment.h.

```
{
   for ( unsigned int i = 0; i < 6; ++i )
   {
      T dist = dot(f.mPlanes[i].mNorm, static_cast< Vec<T, 3> >(p)) + f.mPlanes[i
      ].mOffset;
      if (dist < T(0.0) )
      {
         idx = i;
         return false;
      }
   }

   idx = IN_FRONT_OF_ALL_PLANES;
   return true;
}
```

9.1.3.81 template<typename T > bool gmtl::isInVolume (const Frustum< T > & *f*, const Sphere< T > & *s*) `[inline]`

Definition at line 513 of file Containment.h.

```
{
   for ( unsigned int i = 0; i < 6; ++i )
   {
      T dist = dot(f.mPlanes[i].mNorm, static_cast< Vec<T, 3> >(s.getCenter())) +
```

```
     f.mPlanes[i].mOffset;
      if ( dist <= -T(s.getRadius()) )
      {
         return false;
      }
   }

   return true;
}
```

9.1.3.82 template<typename T > bool gmtl::isInVolume (const Frustum< T > & *f*, const Tri< T > & *tri*) [inline]

Definition at line 565 of file Containment.h.

```
{
   unsigned int junk;

   if ( isInVolume(f, tri[0], junk) )
   {
      return true;
   }

   if ( isInVolume(f, tri[1], junk) )
   {
      return true;
   }

   if ( isInVolume(f, tri[2], junk) )
   {
      return true;
   }

   return false;
}
```

9.1.3.83 template<typename T > bool gmtl::isInVolume (const Frustum< T > & *f*, const AABox< T > & *box*) [inline]

Definition at line 528 of file Containment.h.

```
{
   const Point<T, 3>& min = box.getMin();
   const Point<T, 3>& max = box.getMax();
   Point<T, 3> p[8];
   p[0] = min;
   p[1] = max;
   p[2] = Point<T, 3>(max[0], min[1], min[2]);
   p[3] = Point<T, 3>(min[0], max[1], min[2]);
   p[4] = Point<T, 3>(min[0], min[1], max[2]);
```

```
   p[5] = Point<T, 3>(max[0], max[1], min[2]);
   p[6] = Point<T, 3>(min[0], max[1], max[2]);
   p[7] = Point<T, 3>(max[0], min[1], max[2]);

   unsigned int idx = 6;

   if ( isInVolume(f, p[0], idx) )
   {
      return true;
   }

   // now we have the index of the seperating plane int idx, so check if all
   // other points lie on the backside of this plane too

   for ( unsigned int i = 1; i < 8; ++i )
   {
      T dist = dot(f.mPlanes[idx].mNorm, static_cast< Vec<T, 3> >(p[i])) + f.mPla
      nes[idx].mOffset;
      if ( dist > T(0.0) )
      {
         return true;
      }
   }

   return false;
}
```

9.1.3.84 template< class DATA_TYPE > bool gmtl::isInVolumeExclusive (const AABox< DATA_TYPE > & *container*, const Point< DATA_TYPE, 3 > & *pt*)

Tests if the given point is inside (not on) the surface of the given AABox volume.

This method is "exclusive" because it does not consider the surface to be a part of the space.

Parameters

container the AABox to test against

pt the point to test with

Returns

true if pt is inside container (but not on surface), false otherwise

Definition at line 334 of file Containment.h.

```
{
   if (! container.isEmpty())
   {
      return ( pt[0] > container.mMin[0] &&
```

```
                pt[1] > container.mMin[1] &&
                pt[2] > container.mMin[2] &&
                pt[0] < container.mMax[0] &&
                pt[1] < container.mMax[1] &&
                pt[2] < container.mMax[2]);
   }
   else
   {
      return false;
   }
}
```

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

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

The quaternion is normalized if its lengthSquared is 1.

Parameters

q1 the quaternion to test

eps the epsilon tolerance

Returns

true if the quaternion is normalized, false otherwise

Definition at line 364 of file QuatOps.h.

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

9.1.3.86 template<class DATA_TYPE , unsigned SIZE> bool gmtl::isNormalized (const Vec< DATA_TYPE, SIZE > & *v1*, 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 438 of file VecOps.h.

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

9.1.3.87 template<class DATA_TYPE > bool gmtl::isOnVolume (const Sphere< DATA_TYPE > & *container*, const Point< DATA_TYPE, 3 > & *pt*)

Modifies the given sphere to tightly enclose all spheres 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 spheres in spheres

spheres [in] the list of spheres to contain

Precondition

spheres must contain at least 2 points 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 262 of file Containment.h.

```
{
   // |center - pt| - radius == 0
   return ( length(gmtl::Vec<DATA_TYPE,3>(container.mCenter - pt)) - container.mR
      adius == 0 );
}
```

9.1.3.88 template<class DATA_TYPE > bool gmtl::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 280 of file Containment.h.

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

   // abs( |center-pt| - radius ) < tol
   return ( Math::abs( length( gmtl::Vec<DATA_TYPE,3>(container.mCenter - pt)) -
      container.mRadius )
            <= tol );
}
```

9.1.3.89 template<typename DATA_TYPE > DATA_TYPE gmtl::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 326 of file QuatOps.h.

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

9.1.3.90 template<class DATA_TYPE , unsigned SIZE> DATA_TYPE gmtl::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 367 of file VecOps.h.

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

9.1.3.91 template<class DATA_TYPE , unsigned SIZE> DATA_TYPE gmtl::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 386 of file VecOps.h.

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

```
   return ret_val;
#else
   return gmtl::meta::LenSqrVecUnrolled<SIZE-1,Vec<DATA_TYPE,SIZE> >::func(v1);
#endif
}
```

9.1.3.92 template<typename DATA_TYPE > DATA_TYPE gmtl::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 314 of file QuatOps.h.

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

9.1.3.93 template<typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::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 554 of file QuatOps.h.

```
{
   // 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;
}
```

9.1.3.94 template<typename DATA_TYPE , unsigned SIZE> VecBase<DATA_TYPE, SIZE>& gmtl::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

Todo

metaprogramming...

Definition at line 520 of file VecOps.h.

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

9.1.3.95 template<typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::log (Quat< DATA_TYPE > & *result*)

complex logarithm

Postcondition

sets self to the log of quat

See also

Quat

Definition at line 440 of file QuatOps.h.

```
   {
      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 ) == fa
    lse)
       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;
}
```

9.1.3.96 template< typename TARGET_TYPE , typename SOURCE_TYPE > TARGET_TYPE gmtl::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.

Parameters

src the object to use for creation

Returns

a new object with values based on the src variable

See also

OpenSGGenerate.h for an example

Definition at line 1276 of file Generate.h.

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

9.1.3.97 template<typename ROTATION_TYPE > ROTATION_TYPE gmtl::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 1105 of file Generate.h.

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

9.1.3.98 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Vec<DATA_TYPE, ROWS> gmtl::makeColumn (const Matrix< DATA_TYPE, ROWS, COLS > & *src*, unsigned *col*)

Accesses a particular column in the matrix by creating a new vector containing the values in the given matrix.

Parameters

src the matrix being accessed
col the column of the matrix to access

Returns

a vector containing the values in the requested column

Definition at line 1462 of file Generate.h.

```
{
   Vec<DATA_TYPE, ROWS> result;
   setColumn(result, src, col);
   return result;
}
```

9.1.3.99 template<typename DATA_TYPE > Quat<DATA_TYPE> gmtl::makeConj (const Quat< DATA_TYPE > & *quat*) [inline]

quaternion complex conjugate.

Parameters

quat any quaternion object

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 161 of file Generate.h.

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

9.1.3.100 template<class DATA_TYPE > Vec<DATA_TYPE,3> gmtl::makeCross (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.

Precondition

v1 and v2 must be 3-D 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 64 of file Generate.h.

```
{
   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])) );
}
```

9.1.3.101 template<typename ROTATION_TYPE > ROTATION_TYPE gmtl::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.

If the two coordinate frames are labeled: SRC and TARGET, the matrix produced is: src_M_target this means that it will transform a point in TARGET to SRC but moves the base frame from SRC to TARGET.

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 1309 of file Generate.h.

```
{
   gmtl::ignore_unused_variable_warning(t);
   ROTATION_TYPE temporary;
   return setDirCos( temporary, xDestAxis, yDestAxis, zDestAxis, xSrcAxis, ySr
   cAxis, zSrcAxis );
}
```

9.1.3.102 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS> gmtl::makeInvert (const Matrix< DATA_TYPE, ROWS, COLS > & *src*) `[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 1134 of file Generate.h.

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

9.1.3.103 template<typename DATA_TYPE > Quat<DATA_TYPE> gmtl::makeInvert (const Quat< DATA_TYPE > & *quat*) `[inline]`

create quaternion from the inverse of another quaternion.

Parameters

quat any quaternion object

Returns

a quaternion that is the multiplicative inverse of quat

See also

Quat

Definition at line 173 of file Generate.h.

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

9.1.3.104 template<typename DATA_TYPE , unsigned SIZE> Vec<DATA_TYPE, SIZE> gmtl::makeNormal (Vec< DATA_TYPE, SIZE > *vec*) `[inline]`

create a normalized vector from the given vector.

Definition at line 45 of file Generate.h.

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

9.1.3.105 template<typename DATA_TYPE > Quat<DATA_TYPE> gmtl::makeNormal (const Quat< DATA_TYPE > & *quat*) `[inline]`

Normalize a quaternion.

Parameters

quat a quaternion

Postcondition

quat is normalized

Definition at line 148 of file Generate.h.

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

9.1.3.106 template<typename DATA_TYPE > AxisAngle<DATA_TYPE> gmtl::makeNormal (const AxisAngle< DATA_TYPE > & *a*)

make the axis of an AxisAngle normalized

Definition at line 415 of file Generate.h.

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

9.1.3.107 template<typename DATA_TYPE > Quat<DATA_TYPE> gmtl::makePure (const Vec< DATA_TYPE, 3 > & *vec*) `[inline]`

create a pure quaternion

Precondition

vec should be normalized

Parameters

vec a normalized vector representing an axis

Returns

a quaternion with vec as its axis, and no rotation

Postcondition

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

Definition at line 138 of file Generate.h.

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

9.1.3.108 template<typename ROTATION_TYPE , typename SOURCE_TYPE > ROTATION_TYPE gmtl::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 1289 of file Generate.h.

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

9.1.3.109 template<typename ROTATION_TYPE > ROTATION_TYPE gmtl::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.

Precondition

each vec needs to be normalized.

Postcondition

This function returns a temporary object.

Definition at line 1351 of file Generate.h.

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

9.1.3.110 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Vec<DATA_TYPE, COLS> gmtl::makeRow (const Matrix< DATA_TYPE, ROWS, COLS > & *src*, unsigned *row*)

Accesses a particular row in the matrix by creating a new vector containing the values in the given matrix.

Parameters

src the matrix being accessed

row the row of the matrix to access

Returns

a vector containing the values in the requested row

Definition at line 1430 of file Generate.h.

```
{
   Vec<DATA_TYPE, COLS> result;
   setRow(result, src, row);
   return result;
}
```

9.1.3.111 template<typename MATRIX_TYPE , typename INPUT_TYPE > MATRIX_TYPE gmtl::makeScale (const INPUT_TYPE & *scale*, Type2Type< MATRIX_TYPE > *t* = *Type2Type*< *MATRIX_TYPE* >*()*) [inline]

Create a scale matrix.

Parameters

scale You'll typically pass in a Vec or a float here. setScale() for all possible argument types for this function.

Definition at line 803 of file Generate.h.

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

9.1.3.112 template<typename TRANS_TYPE , typename SRC_TYPE > TRANS_TYPE gmtl::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 1338 of file Generate.h.

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

9.1.3.113 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS> gmtl::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 1120 of file Generate.h.

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

9.1.3.114 template<typename DATA_TYPE > Vec<DATA_TYPE, 3> gmtl::makeVec (const Quat< DATA_TYPE > & *quat*) [inline]

create a vector from the vector component of a quaternion

Returns

a vector of the quaternion's axis. quat = [v,0] = [v0,v1,v2,0]

Definition at line 37 of file Generate.h.

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

9.1.3.115 template<class DATA_TYPE > void gmtl::makeVolume (AABox< DATA_TYPE > & *box*, const Sphere< DATA_TYPE > & *sph*)

Creates an AABox that tightly encloses the given Sphere.

Parameters

box set to the box

Definition at line 470 of file Containment.h.

```
{
   const gmtl::Point<DATA_TYPE, 3>& center = sph.getCenter();
   const DATA_TYPE& radius = sph.getRadius();

   // Calculate the min and max points for the box
   gmtl::Point<DATA_TYPE, 3> min_pt(center[0] - radius,
                                    center[1] - radius,
                                    center[2] - radius);
   gmtl::Point<DATA_TYPE, 3> max_pt(center[0] + radius,
                                    center[1] + radius,
                                    center[2] + radius);

   box.setMin(min_pt);
   box.setMax(max_pt);
   box.setEmpty(radius == DATA_TYPE(0));
}
```

9.1.3.116 template<class DATA_TYPE > void gmtl::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 150 of file Containment.h.

```
{
   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)
   typename 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 / static_cast<DATA_TYPE>(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 )
   {
      DATA_TYPE len = lengthSquared( gmtl::Vec<DATA_TYPE,3>( (*itr) - container.m
      Center) );
      if ( len > radiusSqr )
         radiusSqr = len;
   }

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

9.1.3.117 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> DATA_TYPE gmtl::makeXRot (const Matrix< DATA_TYPE, ROWS, COLS > & *mat*) `[inline]`

Extracts the X-axis rotation information from the matrix.

Postcondition

Returned value is from -PI to PI, where 0 is no rotation.

Definition at line 961 of file Generate.h.

```
{
   const gmtl::Vec<DATA_TYPE, 3> forward_point(0.0f, 0.0f, -1.0f);   // -Z
   const gmtl::Vec<DATA_TYPE, 3> origin_point(0.0f, 0.0f, 0.0f);
   gmtl::Vec<DATA_TYPE, 3> end_point, start_point;

   gmtl::xform(end_point, mat, forward_point);
   gmtl::xform(start_point, mat, origin_point);
   gmtl::Vec<DATA_TYPE, 3> 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);
   DATA_TYPE x_rot = gmtl::Math::aCos(gmtl::dot(direction_vector,
                                             forward_point));

   gmtl::Vec<DATA_TYPE, 3> which_side = gmtl::makeCross(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;
}
```

9.1.3.118 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> DATA_TYPE gmtl::makeYRot (const Matrix< DATA_TYPE, ROWS, COLS > & *mat*) `[inline]`

Extracts the Y axis rotation information from the matrix.

Postcondition

Returned value is from -PI to PI, where 0 is none.

Definition at line 928 of file Generate.h.

```
{
   const gmtl::Vec<DATA_TYPE, 3> forward_point(0.0f, 0.0f, -1.0f);   // -Z
   const gmtl::Vec<DATA_TYPE, 3> origin_point(0.0f, 0.0f, 0.0f);
   gmtl::Vec<DATA_TYPE, 3> end_point, start_point;

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

```
   gmtl::xform(start_point, mat, origin_point);
   gmtl::Vec<DATA_TYPE, 3> 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);
   DATA_TYPE y_rot = gmtl::Math::aCos(gmtl::dot(direction_vector,
                                                forward_point));

   gmtl::Vec<DATA_TYPE, 3> which_side = gmtl::makeCross(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;
}
```

9.1.3.119 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> DATA_TYPE gmtl::makeZRot (const Matrix< DATA_TYPE, ROWS, COLS > & *mat*) `[inline]`

Extracts the Z-axis rotation information from the matrix.

Postcondition

Returned value is from -PI to PI, where 0 is no rotation.

Definition at line 994 of file Generate.h.

```
{
   const gmtl::Vec<DATA_TYPE, 3> forward_point(1.0f, 0.0f, 0.0f);   // +x
   const gmtl::Vec<DATA_TYPE, 3> origin_point(0.0f, 0.0f, 0.0f);
   gmtl::Vec<DATA_TYPE, 3> end_point, start_point;

   gmtl::xform(end_point, mat, forward_point);
   gmtl::xform(start_point, mat, origin_point);
   gmtl::Vec<DATA_TYPE, 3> 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);
   DATA_TYPE z_rot = gmtl::Math::aCos(gmtl::dot(direction_vector,
                                                forward_point));

   gmtl::Vec<DATA_TYPE, 3> which_side = gmtl::makeCross(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;
 }
```

9.1.3.120 template<typename DATA_TYPE > void gmtl::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 470 of file QuatOps.h.

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

9.1.3.121 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& gmtl::mult (Matrix< DATA_TYPE, ROWS, COLS > & *result*, const Matrix< DATA_TYPE, ROWS, COLS > & *mat*, const DATA_TYPE & *scalar*) [inline]

matrix scalar mult.

mult each elt in a matrix by a scalar value. : result = mat ∗ scalar

Definition at line 196 of file MatrixOps.h.

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

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

matrix scalar mult.

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

Definition at line 209 of file MatrixOps.h.

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

9.1.3.123 template<typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::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)
result = q1 * q2 (where q2 would be applied first to any xformed geometry)

See also

Quat

Definition at line 26 of file QuatOps.h.

```
{
   // 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]*q2[Yelt];
    temporary[Yelt] = q1[Welt]*q2[Yelt] + q1[Yelt]*q2[Welt] + q1[Zelt]*q2[Xelt]
     - q1[Xelt]*q2[Zelt];
    temporary[Zelt] = q1[Welt]*q2[Zelt] + q1[Zelt]*q2[Welt] + q1[Xelt]*q2[Yelt]
     - q1[Yelt]*q2[Xelt];
    temporary[Welt] = q1[Welt]*q2[Welt] - q1[Xelt]*q2[Xelt] - q1[Yelt]*q2[Yelt]
     - q1[Zelt]*q2[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;
}
```

9.1.3.124 template<typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::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 127 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;
}
```

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

matrix multiply.

: With regard to size (ROWS/COLS): if lhs is m x p, and rhs is p x n, then result is m x n (mult func undefined otherwise) : returns a m x n sized matrix

Postcondition

: result = lhs ∗ rhs (where rhs is applied first)

Definition at line 80 of file MatrixOps.h.

```
{
   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 );

   // track state
   ret_mat.mState = combineMatrixStates( lhs.mState, rhs.mState );
   return result = ret_mat;
}
```

9.1.3.126 template<typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::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.

Returns

returns the negation of the given quat.

Definition at line 102 of file QuatOps.h.

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

9.1.3.127 template<class DATA_TYPE > Vec<DATA_TYPE, 3> gmtl::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 42 of file TriOps.h.

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

9.1.3.128 template<typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::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 337 of file QuatOps.h.

```
{
   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;
}
```

9.1.3.129 template<class DATA_TYPE , unsigned SIZE> DATA_TYPE gmtl::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 unless length(v1) is originally 0.0, in which case it is unchanged

Parameters

v1 the vector to normalize

Returns

the length of v1 before it was normalized

Definition at line 413 of file VecOps.h.

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

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

   return len;
}
```

9.1.3.130 template<class DATA_TYPE > void gmtl::normalize (Frustum< DATA_TYPE > & *f*)

Definition at line 18 of file FrustumOps.h.

```
{
   for ( unsigned int i = 0; i < 6; ++i )
   {
      Vec<DATA_TYPE, 3> n = f.mPlanes[i].getNormal();
      DATA_TYPE o = f.mPlanes[i].getOffset();
      DATA_TYPE len = Math::sqrt( n[0] * n[0] + n[1] * n[1] + n[2] * n[2]);
      n[0] /= len;
      n[1] /= len;
      n[2] /= len;
      o /= len;
      f.mPlanes[i].setNormal(n);
      f.mPlanes[i].setOffset(o);
   }
}
```

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

Tests 2 matrices for inequality.

Parameters

lhs The first matrix

rhs The second matrix

Precondition

Both matrices must be of the same size.

Returns

false if the matrices differ on any element value; true otherwise

Definition at line 713 of file MatrixOps.h.

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

9.1.3.132 template< class DATA_TYPE , typename ROT_ORDER > bool gmtl::operator!= (const EulerAngle< DATA_TYPE, ROT_ORDER > & *e1*, const EulerAngle< DATA_TYPE, ROT_ORDER > & *e2*) `[inline]`

Compares e1 and e2 (component-wise) to see if they are NOT exactly the same.

Parameters

e1 the first EulerAngle

e2 the second EulerAngle

Returns

true if e1 does not equal e2; false if they are equal

Definition at line 47 of file EulerAngleOps.h.

```
{
   return(! (e1 == e2));
}
```

9.1.3.133 template< class DATA_TYPE > bool gmtl::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 154 of file PlaneOps.h.

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

9.1.3.134 template<class DATA_TYPE , unsigned SIZE> bool gmtl::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 584 of file VecOps.h.

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

9.1.3.135 template<class DATA_TYPE > bool gmtl::operator!= (const Tri< DATA_TYPE > & *tri1*, const Tri< DATA_TYPE > & *tri2*)

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

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 80 of file TriOps.h.

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

9.1.3.136 template<typename DATA_TYPE > bool gmtl::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 611 of file QuatOps.h.

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

9.1.3.137 template<class DATA_TYPE > bool gmtl::operator!= (const AABox< DATA_TYPE > & *b1*, const AABox< DATA_TYPE > & *b2*) `[inline]`

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

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

Parameters

b1 the first box to compare

b2 the second box to compare

Returns

true if they are not equal, false otherwise

Definition at line 47 of file AABoxOps.h.

```
{
   return (! (b1 == b2));
}
```

9.1.3.138 template<class DATA_TYPE > bool gmtl::operator!= (const Ray< DATA_TYPE > & *ls1*, const Ray< DATA_TYPE > & *ls2*) `[inline]`

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

Parameters

ls1 the first Ray to compare

ls2 the second Ray to compare

Returns

true if they are not equal, false otherwise

Definition at line 37 of file RayOps.h.

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

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

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

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 44 of file SphereOps.h.

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

9.1.3.140 template<typename POS_TYPE , typename ROT_TYPE > bool gmtl::operator!= (const Coord< POS_TYPE, ROT_TYPE > & *c1*, const Coord< POS_TYPE, ROT_TYPE > & *c2*) `[inline]`

Compare two coordinate frames for not-equality.

Parameters

c1 the first Coord

c2 the second Coord

Returns

true if c1 is different from c2, false otherwise

Definition at line 37 of file CoordOps.h.

```
{
   return !operator==( c1, c2 );
}
```

9.1.3.141 template<class DATA_TYPE > bool gmtl::operator!= (const AxisAngle< DATA_TYPE > & *a1*, const AxisAngle< DATA_TYPE > & *a2*) `[inline]`

Compares 2 AxisAngles to see if they are NOT exactly the same.

Parameters

a1 the first AxisAngle

a2 the second AxisAngle

Returns

true if a1 does not equal a2; false if they are equal

Definition at line 48 of file AxisAngleOps.h.

```
{
   return !(a1 == a2);
}
```

9.1.3.142 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Ray<DATA_TYPE> gmtl::operator* (const Matrix< DATA_TYPE, ROWS, COLS > & *matrix*, const Ray< DATA_TYPE > & *ray*) `[inline]`

ray * a matrix multiplication of [m x k] matrix by a ray.

Parameters

matrix the transform matrix

ray the original ray

Returns

the ray transformed by the matrix

Postcondition

This results in a full matrix xform of the ray.

Definition at line 384 of file Xforms.h.

```
{
   Ray<DATA_TYPE> temporary;
   return xform( temporary, matrix, ray );
}
```

9.1.3.143 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> LineSeg<DATA_TYPE> gmtl::operator∗ (const Matrix< DATA_TYPE, ROWS, COLS > & *matrix*, const LineSeg< DATA_TYPE > & *seg*) [inline]

seg ∗ a matrix multiplication of [m x k] matrix by a seg.

Parameters

matrix the transform matrix
seg the original ray

Returns

the seg transformed by the matrix

Postcondition

This results in a full matrix xform of the seg.

Definition at line 431 of file Xforms.h.

```
{
   LineSeg<DATA_TYPE> temporary;
   return xform( temporary, matrix, seg );
}
```

9.1.3.144 template<typename DATA_TYPE > Quat<DATA_TYPE> gmtl::operator∗ (const Quat< DATA_TYPE > & *q1*, const Quat< DATA_TYPE > & *q2*)

product of two quaternions (quaternion product) Does quaternion multiplication.

Postcondition

temp' = q1 ∗ q2 (where q2 would be applied first to any xformed geometry)

See also

Quat

Todo

metaprogramming on quat operator∗()

Definition at line 75 of file QuatOps.h.

```
{
   // (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[Yelt],
                           q1[Welt]*q2[Yelt] + q1[Yelt]*q2[Welt] + q1[Zelt]*q2
   [Xelt] - q1[Xelt]*q2[Zelt],
                           q1[Welt]*q2[Zelt] + q1[Zelt]*q2[Welt] + q1[Xelt]*q2
   [Yelt] - q1[Yelt]*q2[Xelt],
                           q1[Welt]*q2[Welt] - q1[Xelt]*q2[Xelt] - q1[Yelt]*q2
   [Yelt] - q1[Zelt]*q2[Zelt] );
}
```

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

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

Parameters

matrix the transform matrix

point the original point

Returns

the point transformed by the matrix

Postcondition

This results in a full matrix xform of the point.

Definition at line 319 of file Xforms.h.

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

9.1.3.146 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS, unsigned COLS_MINUS_ONE> Point<DATA_TYPE, COLS_MINUS_ONE> gmtl::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]).

Parameters

matrix the transform matrix
point the original point

Returns

the point transformed by the matrix

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 305 of file Xforms.h.

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

9.1.3.147 template<typename T , unsigned SIZE, typename R1 > VecBase<T,SIZE, meta::VecBinaryExpr<VecBase<T,SIZE,R1>, VecBase<T,SIZE, meta::ScalarArg<T> >, meta::VecMultBinary> > gmtl::operator∗ (const VecBase< T, SIZE, R1 > & *v1*, const T *scalar*) `[inline]`

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 216 of file VecOps.h.

```
{
   return VecBase<T,SIZE,
            meta::VecBinaryExpr<VecBase<T,SIZE,R1>,
                                VecBase<T,SIZE, meta::ScalarArg<T> >,
                                meta::VecMultBinary> >(
                                      meta::VecBinaryExpr<VecBase<T,SIZE,R1>,
                                                          VecBase<T,SIZE, meta::
      ScalarArg<T> >,
                                                          meta::VecMultBinary>(v
      1,
                                                                                m
      eta::ScalarArg<T>(scalar)) );
}
```

9.1.3.148 template<typename T , unsigned SIZE, typename R1 > VecBase<T,SIZE, meta::VecBinaryExpr< VecBase<T,SIZE, meta::ScalarArg<T> >, VecBase<T,SIZE,R1>, meta::VecMultBinary> > gmtl::operator* (const T *scalar*, const VecBase< T, SIZE, R1 > & *v1*) `[inline]`

Definition at line 232 of file VecOps.h.

```
{
   return VecBase<T,SIZE,
            meta::VecBinaryExpr<VecBase<T,SIZE, meta::ScalarArg<T> >,
                                VecBase<T,SIZE,R1>,
                                meta::VecMultBinary> >(
                                      meta::VecBinaryExpr<VecBase<T,SIZE, meta::
      ScalarArg<T> >,
                                                          VecBase<T,SIZE,R1>,
                                                          meta::VecMultBinary>(m
      eta::ScalarArg<T>(scalar), v1 ) );
}
```

9.1.3.149 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS, unsigned COLS_MINUS_ONE> Vec<DATA_TYPE, COLS_MINUS_ONE> gmtl::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).

Parameters

matrix the transform matrix

vector the original vector

Returns

the vector transformed by the matrix 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 199 of file Xforms.h.

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

9.1.3.150 template<typename DATA_TYPE > Quat<DATA_TYPE> gmtl::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 141 of file QuatOps.h.

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

9.1.3.151 template<typename DATA_TYPE > VecBase<DATA_TYPE, 3> gmtl::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).

Parameters

rot The quaternion

vector The original vector to transform

Returns

the resulting vector transformed by the quaternion

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 74 of file Xforms.h.

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

9.1.3.152 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Vec<DATA_TYPE, COLS> gmtl::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...).

Parameters

matrix the transform matrix
vector the original vector

Returns

the vector transformed by the matrix

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.

```
{
   // 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 );
}
```

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

matrix ∗ matrix.

: With regard to size (ROWS/COLS): if lhs is m x p, and rhs is p x n, then result is m x n (mult func undefined otherwise) : returns a m x n sized matrix == lhs ∗ rhs (where rhs is applied first) returns a temporary, is slower.

Definition at line 106 of file MatrixOps.h.

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

9.1.3.154 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Point<DATA_TYPE, COLS> gmtl::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...).

Parameters

matrix the transform matrix

point the original point

Returns

the point transformed by the matrix

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 245 of file Xforms.h.

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

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

matrix postmult (operator∗=).

does a postmult on the matrix. : args must both be n x n sized (this function is undefined otherwise) : result' = result ∗ operand (where operand is applied first)

Definition at line 185 of file MatrixOps.h.

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

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

partial point ∗= a matrix 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]).

Parameters

matrix the transform matrix

point the original point

Returns

the point transformed by the matrix

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 349 of file Xforms.h.

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

9.1.3.157 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Point<DATA_TYPE, COLS> gmtl::operator*= (Point< DATA_TYPE, COLS > & *point*, const Matrix< DATA_TYPE, ROWS, COLS > & *matrix*) [inline]

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

Parameters

matrix the transform matrix

point the original point

Returns

the point transformed by the matrix

Postcondition

This results in a full matrix xform of the point.

Definition at line 334 of file Xforms.h.

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

9.1.3.158 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Ray<DATA_TYPE>& gmtl::operator*= (Ray< DATA_TYPE > & *ray*, const Matrix< DATA_TYPE, ROWS, COLS > & *matrix*) `[inline]`

ray *= a matrix multiplication of [m x k] matrix by a ray.

Parameters

matrix the transform matrix

ray the original ray

Returns

the ray transformed by the matrix

Postcondition

This results in a full matrix xform of the ray.

Definition at line 399 of file Xforms.h.

```
{
   Ray<DATA_TYPE> temporary = ray;
   return xform( ray, matrix, temporary);
}
```

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

matrix scalar mult (operator*=).

multiply matrix elements by a scalar : result *= scalar

Definition at line 221 of file MatrixOps.h.

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

9.1.3.160 template<typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::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 152 of file QuatOps.h.

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

9.1.3.161 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> LineSeg<DATA_TYPE>& gmtl::operator∗= (LineSeg< DATA_TYPE > & *seg*, const Matrix< DATA_TYPE, ROWS, COLS > & *matrix*) `[inline]`

seg ∗= a matrix multiplication of [m x k] matrix by a seg.

Parameters

matrix the transform matrix

seg the original point

Returns

the point transformed by the matrix

Postcondition

This results in a full matrix xform of the point.

Definition at line 446 of file Xforms.h.

```
{
   LineSeg<DATA_TYPE> temporary = seg;
   return xform( seg, matrix, temporary);
}
```

9.1.3.162 template<class DATA_TYPE , unsigned SIZE, class SCALAR_TYPE > VecBase<DATA_TYPE, SIZE>& gmtl::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 182 of file VecOps.h.

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

   return v1;
}
```

9.1.3.163 template<typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::operator*= (Quat< DATA_TYPE > & *result*, const Quat< DATA_TYPE > & *q2*)

quaternion postmult

Postcondition

result' = result ∗ q2 (where q2 is applied first to any xformed geometry)

See also

Quat

Definition at line 91 of file QuatOps.h.

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

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

transform a vector by a rotation quaternion.

Precondition

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

Parameters

rot The quaternion

vector The original vector to transform

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 88 of file Xforms.h.

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

9.1.3.165 template<typename DATA_TYPE > Quat<DATA_TYPE> gmtl::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 241 of file QuatOps.h.

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

9.1.3.166 template<typename T , unsigned SIZE, typename R1 , typename R2 > VecBase<T,SIZE, meta::VecBinaryExpr<VecBase<T,SIZE,R1>, VecBase<T,SIZE,R2>, meta::VecPlusBinary> > gmtl::operator+ (const VecBase< T, SIZE, R1 > & *v1*, const VecBase< T, SIZE, R2 > & *v2*) `[inline]`

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 103 of file VecOps.h.

```
{
   return VecBase<T,SIZE,
               meta::VecBinaryExpr<VecBase<T,SIZE,R1>,
                                    VecBase<T,SIZE,R2>,
                                    meta::VecPlusBinary> >( meta::VecBinaryExpr<Ve
      cBase<T,SIZE,R1>,
                                                                                 Ve
      cBase<T,SIZE,R2>,
                                                                                 me
      ta::VecPlusBinary>(v1,v2) );
}
```

9.1.3.167 template<typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::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 252 of file QuatOps.h.

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

9.1.3.168 template<class DATA_TYPE , unsigned SIZE, typename REP2 > VecBase<DATA_TYPE, SIZE>& gmtl::operator+= (VecBase< DATA_TYPE, SIZE > & *v1*, const VecBase< DATA_TYPE, SIZE, REP2 > & *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 71 of file VecOps.h.

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

   return v1;
}
```

9.1.3.169 template<typename DATA_TYPE > Quat<DATA_TYPE> gmtl::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 275 of file QuatOps.h.

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

9.1.3.170 template<typename T , unsigned SIZE, typename R1 , typename R2 > VecBase<T,SIZE, meta::VecBinaryExpr<VecBase<T,SIZE,R1>, VecBase<T,SIZE,R2>, meta::VecMinusBinary> > gmtl::operator- (const VecBase< T, SIZE, R1 > & *v1*, const VecBase< T, SIZE, R2 > & *v2*) `[inline]`

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 161 of file VecOps.h.

```
{
   return VecBase<T,SIZE,
            meta::VecBinaryExpr<VecBase<T,SIZE,R1>,
                                VecBase<T,SIZE,R2>,
                                meta::VecMinusBinary> >( meta::VecBinaryExpr<V
      ecBase<T,SIZE,R1>,
                                                                              Ve
      cBase<T,SIZE,R2>,
                                                                              me
      ta::VecMinusBinary>(v1,v2) );
}
```

9.1.3.171 template<typename T , unsigned SIZE, typename R1 > VecBase<T,SIZE, meta::VecUnaryExpr<VecBase<T,SIZE,R1>, meta::VecNegUnary> > gmtl::operator- (const VecBase< T, SIZE, R1 > & *v1*) `[inline]`

Negates v1.

The result = -v1.

Parameters

v1 the vector.

Returns

the result of negating v1.

Definition at line 48 of file VecOps.h.

```
{
   return VecBase<T,SIZE,
                 meta::VecUnaryExpr<VecBase<T,SIZE,R1>, meta::VecNegUnary> >
                       ( meta::VecUnaryExpr<VecBase<T,SIZE,R1>, meta::VecNegUnar
      y>(v1) );
}
```

9.1.3.172 template<typename DATA_TYPE > Quat<DATA_TYPE> gmtl::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.

Returns

returns the negation of the given quat

Definition at line 117 of file QuatOps.h.

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

9.1.3.173 template<class DATA_TYPE , unsigned SIZE, typename REP2 > VecBase<DATA_TYPE, SIZE>& gmtl::operator-= (VecBase< DATA_TYPE, SIZE > & *v1*, const VecBase< DATA_TYPE, SIZE, REP2 > & *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 129 of file VecOps.h.

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

   return v1;
}
```

9.1.3.174 template<typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::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 286 of file QuatOps.h.

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

9.1.3.175 template<typename T , unsigned SIZE, typename R1 > VecBase<T,SIZE, meta::VecBinaryExpr<VecBase<T,SIZE,R1>, VecBase<T,SIZE, meta::ScalarArg<T> >, meta::VecDivBinary> > gmtl::operator/ (const VecBase< T, SIZE, R1 > & *v1*, const T *scalar*) `[inline]`

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 309 of file VecOps.h.

```
{
   return VecBase<T,SIZE,
             meta::VecBinaryExpr<VecBase<T,SIZE,R1>,
                                 VecBase<T,SIZE, meta::ScalarArg<T> >,
                                 meta::VecDivBinary> >(
                                       meta::VecBinaryExpr<VecBase<T,SIZE,R1>,
                                                           VecBase<T,SIZE, meta::
      ScalarArg<T> >,
                                                           meta::VecDivBinary>(v1
      ,
                                                                                me
      ta::ScalarArg<T>(scalar)) );
}
```

9.1.3.176 template<typename DATA_TYPE > Quat<DATA_TYPE> gmtl::operator/ (const Quat< DATA_TYPE > & *q1*, Quat< DATA_TYPE > *q2*)

quotient of two quaternions

Postcondition

result = q1 $*$ (1/q2) (where 1/q2 is applied first to any xform'd geometry)

See also

Quat

Definition at line 173 of file QuatOps.h.

```
{
   return q1 * invert( q2 );
}
```

9.1.3.177 template<typename DATA_TYPE > Quat<DATA_TYPE> gmtl::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 207 of file QuatOps.h.

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

9.1.3.178 template<typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::operator/= (Quat< DATA_TYPE > & *result*, const Quat< DATA_TYPE > & *q2*)

quotient of two quaternions

Postcondition

result = result $*$ (1/q2) (where 1/q2 is applied first to any xform'd geometry)

See also

Quat

Definition at line 183 of file QuatOps.h.

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

9.1.3.179 template<typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::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 218 of file QuatOps.h.

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

9.1.3.180 template<class DATA_TYPE , unsigned SIZE, class SCALAR_TYPE > VecBase<DATA_TYPE, SIZE>& gmtl::operator/= (VecBase< DATA_TYPE, SIZE > & *v1*, const SCALAR_TYPE & *scalar*)

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 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 276 of file VecOps.h.

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

   return v1;
}
```

9.1.3.181 template<typename DATA_TYPE , unsigned SIZE, typename REP > std::ostream& gmtl::operator<< (std::ostream & *out*, const VecBase< DATA_TYPE, SIZE, REP > & *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 93 of file Output.h.

```
{
   return output::VecOutputter<DATA_TYPE,SIZE,REP>::outStream(out,v);
}
```

9.1.3.182 template<class DATA_TYPE , unsigned ROWS, unsigned COLS> std::ostream& gmtl::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 132 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;
}
```

9.1.3.183 template<typename DATA_TYPE > std::ostream& gmtl::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 180 of file Output.h.

```
{
   out << t[0] << ", " << t[1] << ", " << t[2];
   return out;
}
```

9.1.3.184 template<typename DATA_TYPE > std::ostream& gmtl::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 201 of file Output.h.

```
{
   out << p.mNorm << ", " << p.mOffset;
   return out;
}
```

9.1.3.185 template<typename DATA_TYPE > std::ostream& gmtl::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 222 of file Output.h.

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

9.1.3.186 template<class DATA_TYPE , typename ROTATION_ORDER > std::ostream& gmtl::operator<< (std::ostream & *out*, const EulerAngle< DATA_TYPE, ROTATION_ORDER > & *e*)

Outputs a string representation of the given EulerAngle type to the given output stream.

Format is {ang1,ang2,ang3}

Parameters

out the stream to write to

e the EulerAngle type to output

Returns

out after it has been written to

Definition at line 109 of file Output.h.

```
{
   const DATA_TYPE* angle_data(e.getData());
   out << "{" << angle_data[0] << ", " << angle_data[1] << ", " << angle_data[
   2] << "}";
   return out;
}
```

9.1.3.187 template<typename DATA_TYPE > std::ostream& gmtl::operator<< (std::ostream & *out*, const Ray< DATA_TYPE > & *b*)

Outputs a string representation of the given Ray to the given output stream.

The output is formatted such that Ray<int>(Point<int>(1,2,3), Vec<int>(4,5,6)) will appear as "(1,2,3) (4,5,6)".

Parameters

out the stream to write to

b the Ray to output

Returns

out after it has been written to

Definition at line 265 of file Output.h.

```
{
   out << b.getOrigin() << " " << b.getDir();
   return out;
}
```

9.1.3.188 template<typename POS_TYPE , typename ROT_TYPE > std::ostream& gmtl::operator<< (std::ostream & *out*, const Coord< POS_TYPE, ROT_TYPE > & *c*)

Definition at line 293 of file Output.h.

```
{
   out << "p:" << c.getPos() << " r:" << c.getRot();
   return out;
}
```

9.1.3.189 template<typename DATA_TYPE > std::ostream& gmtl::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 158 of file Output.h.

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

9.1.3.190 template<typename DATA_TYPE > std::ostream& gmtl::operator<< (std::ostream & *out*, const AABox< DATA_TYPE > & *b*)

Outputs a string representation of the given AABox to the given output stream.

The output is formatted such that AABox<int>(Point<int, 3>(1,2,3), Point<int, 3>(4,5,6)) will appear as "(1,2,3) (4,5,6) false".

Parameters

out the stream to write to

b the AABox to output

Returns

out after it has been written to

Definition at line 243 of file Output.h.

```
{
   out << b.mMin << " " << b.mMax << " ";
   out << (b.mEmpty ? "true" : "false");
   return out;
}
```

9.1.3.191 template<typename DATA_TYPE > std::ostream& gmtl::operator<< (std::ostream & *out*, const LineSeg< DATA_TYPE > & *b*)

Outputs a string representation of the given LineSeg to the given output stream.

The output is formatted such that LineSeg<int>(Point<int>(1,2,3), Vec<int>(4,5,6)) will appear as "(1,2,3) (4,5,6)".

Parameters

out the stream to write to

b the LineSeg to output

Returns

out after it has been written to

Definition at line 286 of file Output.h.

```
{
   out << b.getOrigin() << " " << b.getDir();
   return out;
}
```

9.1.3.192 template<typename DATA_TYPE > bool gmtl::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 599 of file QuatOps.h.

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

9.1.3.193 template<class DATA_TYPE , typename ROT_ORDER > bool gmtl::operator== (const EulerAngle< DATA_TYPE, ROT_ORDER > & *e1*, const EulerAngle< DATA_TYPE, ROT_ORDER > & *e2*) `[inline]`

Compares 2 EulerAngles (component-wise) to see if they are exactly the same.

Parameters

e1 the first EulerAngle

e2 the second EulerAngle

Returns

true if e1 equals e2; false if they differ

Definition at line 28 of file EulerAngleOps.h.

```
{
   // @todo metaprogramming.
   if (e1[0] != e2[0]) return false;
   if (e1[1] != e2[1]) return false;
   if (e1[2] != e2[2]) return false;
   return true;
}
```

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

Tests 2 matrices for equality.

Parameters

lhs The first matrix

rhs The second matrix

Precondition

Both matrices must be of the same size.

Returns

true if the matrices have the same element values; false otherwise

Definition at line 687 of file MatrixOps.h.

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

   return true;

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

9.1.3.195 template<class DATA_TYPE > bool gmtl::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 139 of file PlaneOps.h.

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

9.1.3.196 template<class DATA_TYPE , unsigned SIZE> bool gmtl::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.

Parameters

v1 the first vector

v2 the second vector

Returns

true if v1 equals v2; false if they differ

Definition at line 551 of file VecOps.h.

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

   return true;
#else
   return gmtl::meta::EqualVecUnrolled<SIZE-1,Vec<DATA_TYPE,SIZE> >::func(v1,v2);

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

9.1.3.197 template<class DATA_TYPE > bool gmtl::operator== (const Ray< DATA_TYPE > & *ls1*, const Ray< DATA_TYPE > & *ls2*) [inline]

Compare two line segments to see if they are EXACTLY the same.

Parameters

ls1 the first Ray to compare

ls2 the second Ray to compare

Returns

true if they are equal, false otherwise

Definition at line 23 of file RayOps.h.

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

9.1.3.198 template<class DATA_TYPE > bool gmtl::operator== (const AABox< DATA_TYPE > & *b1*, const AABox< DATA_TYPE > & *b2*) [inline]

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

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

Parameters

b1 the first box to compare

b2 the second box to compare

Returns

true if they are equal, false otherwise

Definition at line 30 of file AABoxOps.h.

```
{
   return ( (b1.isEmpty() == b2.isEmpty()) &&
            (b1.getMin() == b2.getMin()) &&
            (b1.getMax() == b2.getMax()) );
}
```

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

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

Parameters

s1 the first sphere to compare

s2 the second sphere to compare

Returns

true if they are equal, false otherwise

Definition at line 30 of file SphereOps.h.

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

9.1.3.200 template<class DATA_TYPE > bool gmtl::operator== (const Tri< DATA_TYPE > & *tri1*, const Tri< DATA_TYPE > & *tri2*)

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

Parameters

tri1 the first triangle to compare

tri2 the second triangle to compare

Returns

true if they are equal, false otherwise

Definition at line 64 of file TriOps.h.

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

9.1.3.201 template<typename POS_TYPE , typename ROT_TYPE > bool gmtl::operator== (const Coord< POS_TYPE, ROT_TYPE > & *c1*, const Coord< POS_TYPE, ROT_TYPE > & *c2*) `[inline]`

Compare two coordinate frames for equality.

Parameters

c1 the first Coord

c2 the second Coord

Returns

true if c1 is the same as c2, false otherwise

Definition at line 24 of file CoordOps.h.

```
{
   return bool( c1.getPos() == c2.getPos() &&
                c1.getRot() == c2.getRot() );
}
```

9.1.3.202 template<class DATA_TYPE > bool gmtl::operator== (const AxisAngle< DATA_TYPE > & *a1*, const AxisAngle< DATA_TYPE > & *a2*) `[inline]`

Compares 2 AxisAngles to see if they are exactly the same.

Parameters

a1 the first AxisAngle

a2 the second AxisAngle

Returns

true if a1 equals a2; false if they differ

Definition at line 28 of file AxisAngleOps.h.

```
{
   // @todo metaprogramming.
   if (a1[0] != a2[0]) return false;
   if (a1[1] != a2[1]) return false;
   if (a1[2] != a2[2]) return false;
   if (a1[3] != a2[3]) return false;
   return true;
}
```

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

matrix postmultiply.

: args must both be n x n (this function is undefined otherwise) : result' = result ∗ operand

Definition at line 162 of file MatrixOps.h.

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

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

matrix preMultiply.

: args must both be n x n (this function is undefined otherwise) : result' = operand ∗ result

Definition at line 173 of file MatrixOps.h.

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

9.1.3.205 const Quat<double> gmtl::QUAT_ADD_IDENTITYD (0. *0,* 0. *0,* 0. *0,* 0. *0*)

9.1.3.206 const Quat<float> gmtl::QUAT_ADD_IDENTITYF (0. *0f,* 0. *0f,* 0. *0f,* 0. *0f*)

9.1.3.207 const Quat<double> gmtl::QUAT_IDENTITYD (QUAT_MULT_IDENTITYD)

9.1.3.208 const Quat<float> gmtl::QUAT_IDENTITYF (QUAT_MULT_IDENTITYF)

9.1.3.209 const Quat<double> gmtl::QUAT_MULT_IDENTITYD (0. *0,* 0. *0,* 0. *0,* 1. *0*)

9.1.3.210 const Quat<float> gmtl::QUAT_MULT_IDENTITYF (0. *0f,* 0. *0f,* 0. *0f,* 1. *0f*)

9.1.3.211 template<class DATA_TYPE , unsigned SIZE> void gmtl::reflect (Point< DATA_TYPE, SIZE > & *result,* const Plane< DATA_TYPE > & *plane,* const Point< DATA_TYPE, SIZE > & *point*)

Mirror the point by the plane.

Definition at line 113 of file PlaneOps.h.

```
{
   gmtl::Point<DATA_TYPE, SIZE> point_on_plane;
   findNearestPt( plane, point, point_on_plane );
   gmtl::Vec<DATA_TYPE, SIZE> dir = point_on_plane - point;
   result = point + (dir * DATA_TYPE(2.0f));
}
```

9.1.3.212 template<class DATA_TYPE , unsigned SIZE> VecBase<DATA_TYPE, SIZE>& gmtl::reflect (VecBase< DATA_TYPE, SIZE > & *result,* const VecBase< DATA_TYPE, SIZE > & *vec,* const Vec< DATA_TYPE, SIZE > & *normal*)

Reflect a vector about a normal.

This method reflects the given vector around the normal vector given. It is similar to if the normal vector was for a plane that you wanted to reflect about. v going into the plane, n normal to the plane, and r coming back out of the plane. (see below)

| v | / |/ |------> n |\ | \ | r

Parameters

result the vector to store the result i

vec the original vector that we want to reflect

normal the normal vector

Postcondition

result contains the reflected vector

Definition at line 491 of file VecOps.h.

```
{
   result = vec - (DATA_TYPE( 2.0 ) * (dot( (Vec<DATA_TYPE, SIZE>)vec, normal ) *
       normal));
   return result;
}
```

9.1.3.213 template<typename DATA_TYPE , typename ROT_ORDER > Quat<DATA_TYPE>& gmtl::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 215 of file Generate.h.

```
   {
      // 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;
}
```

9.1.3.214 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& gmtl::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 1209 of file Generate.h.

```
{
   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);

   // track state
   mat.mState = Matrix<DATA_TYPE, ROWS, COLS>::IDENTITY;

   return setRot( mat, q );
}
```

9.1.3.215 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS, typename ROT_ORDER > Matrix<DATA_TYPE, ROWS, COLS>& gmtl::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 917 of file Generate.h.

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

9.1.3.216 Matrix44f& gmtl::set (Matrix44f & *mat*, const OSG::Matrix & *osgMat*) `[inline]`

Converts an OpenSG matrix to a gmtl::Matrix.

Parameters

mat The matrix to write the OpenSG matrix data into.

osgMat The source OpenSG matrix.

Returns

The equivalent GMTL matrix.

Definition at line 29 of file OpenSGConvert.h.

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

9.1.3.217 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& gmtl::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 907 of file Generate.h.

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

9.1.3.218 template<typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::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 184 of file Generate.h.

```
{
   gmtlASSERT( (Math::isEqual( lengthSquared( axisAngle.getAxis() ), (DATA_TYP
   E)1.0, (DATA_TYPE)0.0001 )) &&
                "you must pass in a normalized vector to setRot( quat, rad, ve
   c )" );

   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;
}
```

9.1.3.219 template<typename DATATYPE , typename POS_TYPE , typename ROT_TYPE , unsigned MATCOLS, unsigned MATROWS> Coord<POS_TYPE, ROT_TYPE>& gmtl::set (Coord< POS_TYPE, ROT_TYPE > & *eulercoord*, const Matrix< DATATYPE, MATROWS, MATCOLS > & *mat*) `[inline]`

convert Matrix to Coord

Definition at line 1243 of file Generate.h.

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

9.1.3.220 template<typename DATA_TYPE > AxisAngle<DATA_TYPE>& gmtl::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 363 of file Generate.h.

```
{
   // 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[Welt] > 1, try normalizing your 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;
      Vec<DATA_TYPE, 3> axis( quat[Xelt] * sin_half_angle_inv,
                              quat[Yelt] * sin_half_angle_inv,
                              quat[Zelt] * sin_half_angle_inv );
      normalize( axis );
      axisAngle.setAxis( axis );
   }

   // 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::Vec<DATA_TYPE, 3>(
                          DATA_TYPE( 1.0 ) /*- gmtl::Math::abs( quat[Welt] )*/
   ,
                         (DATA_TYPE)0.0,
                         (DATA_TYPE)0.0 ) );
   }
   return axisAngle;
}
```

9.1.3.221 template<typename DATATYPE , typename POS_TYPE , typename ROT_TYPE , unsigned MATCOLS, unsigned MATROWS> Matrix<DATATYPE, MATROWS, MATCOLS>& gmtl::set (Matrix< DATATYPE, MATROWS, MATCOLS > & *mat*, const Coord< POS_TYPE, ROT_TYPE > & *coord*) `[inline]`

Convert a Coord to a Matrix Note: It is set directly, but this is equivalent to T∗R where T is the translation matrix and R is the rotation matrix.

See also

Coord
Matrix

Definition at line 1187 of file Generate.h.

```
{
   // 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;
}
```

9.1.3.222 OSG::Matrix& gmtl::set (OSG::Matrix & *osgMat*, const Matrix44f & *mat*) `[inline]`

Converts a GMTL matrix to an OpenSG matrix.

Parameters

osgMat The matrix to write the GMTL matrix data into.
mat The source GMTL matrix.

Returns

The equivalent OpenSG matrix.

Definition at line 43 of file OpenSGConvert.h.

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

9.1.3.223 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Quat<DATA_TYPE>& gmtl::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 279 of file Generate.h.

```
{
   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
 quaternion to the rotation part of a 3x3, 3x4, 4x3, or 4x4 matrix." );

   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;
   }
```

9.1.3.224 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS, typename ROT_ORDER > EulerAngle<DATA_TYPE, ROT_ORDER>& gmtl::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 437 of file Generate.h.

```
   {
      // @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" );


      // @todo metaprogram this!
      const int& order = ROT_ORDER::ID;
      switch (order)
```

```
      {
      case XYZ::ID:
         {
#if 0
            euler[2] = Math::aTan2( -mat(0,1), mat(0,0) );       // -(-cy*sz)/(cy
      *cz) - cy falls out
            euler[0] = Math::aTan2( -mat(1,2), mat(2,2) );       // -(sx*cy)/(cx*
      cy) - cy falls out
            DATA_TYPE cz = Math::cos( euler[2] );
            euler[1] = Math::aTan2( mat(0,2), mat(0,0) / cz );   // (sy)/((cy*cz)
      /cz)
#else
            DATA_TYPE x(0), y(0), z(0);
            y = Math::aSin( mat(0,2));
            if (y < gmtl::Math::PI_OVER_2)
            {
               if(y > -gmtl::Math::PI_OVER_2)
               {
                  x = Math::aTan2(-mat(1,2),mat(2,2));
                  z = Math::aTan2(-mat(0,1),mat(0,0));
               }
               else  // Non-unique (x - z constant)
               {
                  x = -Math::aTan2(mat(1,0), mat(1,1));
                  z = 0;
               }
            }
            else
            {
               // not unique (x + z constant)
               x = Math::aTan2(mat(1,0), mat(1,1));
               z = 0;
            }
            euler[0] = x;
            euler[1] = y;
            euler[2] = z;

#endif
         }
         break;
      case ZYX::ID:
         {
#if 0
            euler[0] = Math::aTan2( mat(1,0), mat(0,0) );        // (cy*sz)/(cy*c
      z) - cy falls out
            euler[2] = Math::aTan2( mat(2,1), mat(2,2) );        // (sx*cy)/(cx*c
      y) - cy falls out
            DATA_TYPE sx = Math::sin( euler[2] );
            euler[1] = Math::aTan2( -mat(2,0), mat(2,1) / sx );  // -(-sy)/((sx*c
      y)/sx)
#else
            DATA_TYPE x(0), y(0), z(0);

            y = Math::aSin(-mat(2,0));
            if(y < Math::PI_OVER_2)
            {
               if(y>-Math::PI_OVER_2)
```

```
               {
                  z = Math::aTan2(mat(1,0), mat(0,0));
                  x = Math::aTan2(mat(2,1), mat(2,2));
               }
               else  // not unique (x + z constant)
               {
                  z = Math::aTan2(-mat(0,1),-mat(0,2));
                  x = 0;
               }
            }
            else  // not unique (x - z constant)
            {
               z = -Math::aTan2(mat(0,1), mat(0,2));
               x = 0;
            }
            euler[0] = z;
            euler[1] = y;
            euler[2] = x;
#endif
         }
         break;
      case ZXY::ID:
         {
#if 0
         // 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;
#else
         DATA_TYPE x,y,z;

         x = Math::aSin(mat(2,1));
         if(x < Math::PI_OVER_2)
         {
            if(x > -Math::PI_OVER_2)
            {
               z = Math::aTan2(-mat(0,1), mat(1,1));
               y = Math::aTan2(-mat(2,0), mat(2,2));
            }
            else  // not unique (y - z constant)
            {
               z = -Math::aTan2(mat(0,2), mat(0,0));
               y = 0;
            }
         }
         else  // not unique (y + z constant)
         {
            z = Math::aTan2(mat(0,2), mat(0,0));
            y = 0;
         }
         euler[0] = z;
         euler[1] = x;
         euler[2] = y;
#endif
      }
      break;
   default:
      gmtlASSERT( false && "unknown rotation order passed to setRot" );
      break;
   }
   return euler;
}
```

9.1.3.225 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& gmtl::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 1071 of file Generate.h.

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

   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];

   // track state
   switch (result.mState)
   {
   case Matrix<DATA_TYPE, ROWS, COLS>::TRANS:    result.mState = Matrix<DATA_T
   YPE, ROWS, COLS>::AFFINE; break;
   case Matrix<DATA_TYPE, ROWS, COLS>::IDENTITY: result.mState = Matrix<DATA_T
   YPE, ROWS, COLS>::ORTHOGONAL; break;
   }
   return result;
}
```

9.1.3.226 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> void gmtl::setColumn (Vec< DATA_TYPE, ROWS > & *dest*, const Matrix< DATA_TYPE, ROWS, COLS > & *src*, unsigned *col*)

Accesses a particular column in the matrix by copying the values in the column into the given vector.

Parameters

dest the vector in which the values of the column will be placed

src the matrix being accessed

col the column of the matrix to access

Definition at line 1445 of file Generate.h.

```
{
```

```
    for (unsigned i=0; i<ROWS; ++i)
    {
       dest[i] = src[i][col];
    }
}
```

9.1.3.227 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& gmtl::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.

If the two coordinate frames are labeled: SRC and TARGET, the matrix produced is: src_M_target this means that it will transform a point in TARGET to SRC but moves the base frame from SRC to TARGET.

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 1033 of file Generate.h.

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

   DATA_TYPE Xa, Xb, Xc;    // Direction cosines of the secondary x-axis
   DATA_TYPE Ya, Yb, Yc;    // Direction cosines of the secondary y-axis
   DATA_TYPE Za, Zb, Zc;    // Direction cosines of the secondary z-axis

   Xa = dot(xDestAxis, xSrcAxis);  Xb = dot(xDestAxis, ySrcAxis);  Xc = dot(xD
```

```
   estAxis, zSrcAxis);
   Ya = dot(yDestAxis, xSrcAxis);  Yb = dot(yDestAxis, ySrcAxis);  Yc = dot(yD
   estAxis, zSrcAxis);
   Za = dot(zDestAxis, xSrcAxis);  Zb = dot(zDestAxis, ySrcAxis);  Zc = dot(zD
   estAxis, zSrcAxis);

   // Set the matrix correctly
   result( 0, 0 ) = Xa; result( 0, 1 ) = Ya; result( 0, 2 ) = Za;
   result( 1, 0 ) = Xb; result( 1, 1 ) = Yb; result( 1, 2 ) = Zb;
   result( 2, 0 ) = Xc; result( 2, 1 ) = Yc; result( 2, 2 ) = Zc;

   // track state
   switch (result.mState)
   {
   case Matrix<DATA_TYPE, ROWS, COLS>::TRANS:    result.mState = Matrix<DATA_T
   YPE, ROWS, COLS>::AFFINE; break;
   case Matrix<DATA_TYPE, ROWS, COLS>::IDENTITY: result.mState = Matrix<DATA_T
   YPE, ROWS, COLS>::ORTHOGONAL; break;
   }

   return result;
}
```

9.1.3.228 template<typename T > Matrix<T, 4,4>& gmtl::setFrustum (Matrix< T, 4, 4 > & *result*, T *left*, T *top*, T *right*, T *bottom*, T *nr*, T *fr*) `[inline]`

Set an arbitrary projection matrix.

Returns

: set a projection matrix (similar to glFrustum).

Definition at line 615 of file Generate.h.

```
{
   result.mData[0] = ( T( 2.0 ) * nr ) / ( right - left );
   result.mData[1] = T( 0.0 );
   result.mData[2] = T( 0.0 );
   result.mData[3] = T( 0.0 );

   result.mData[4] = T( 0.0 );
   result.mData[5] = ( T( 2.0 ) * nr ) / ( top - bottom );
   result.mData[6] = T( 0.0 );
   result.mData[7] = T( 0.0 );

   result.mData[8] = ( right + left ) / ( right - left );
   result.mData[9] = ( top + bottom ) / ( top - bottom );
   result.mData[10] = -( fr + nr ) / ( fr - nr );
   result.mData[11] = T( -1.0 );

   result.mData[12] = T( 0.0 );
```

```
    result.mData[13] = T( 0.0 );
    result.mData[14] = -( T( 2.0 ) * fr * nr ) / ( fr - nr );
    result.mData[15] = T( 0.0 );

    result.mState = Matrix<T, 4, 4>::FULL; // track state

    return result;
}
```

9.1.3.229 template<typename T > Matrix<T, 4,4>& gmtl::setOrtho (Matrix< T, 4, 4 > & *result*, T *left*, T *top*, T *right*, T *bottom*, T *nr*, T *fr*) `[inline]`

Set an orthographic projection matrix creates a transformation that produces a parallel projection matrix.

= -nr is the value of the near clipping plane (positive value for near is negative in the z direction) = -fr is the value of the far clipping plane (positive value for far is negative in the z direction)

Returns

: set a orthographic matrix (similar to glOrtho).

Definition at line 651 of file Generate.h.

```
{
    result.mData[1] = result.mData[2] = result.mData[3] =
    result.mData[4] = result.mData[6] = result.mData[7] =
    result.mData[8] = result.mData[9] = result.mData[11] = T(0);

    T rl_inv = T(1) / (right - left);
    T tb_inv = T(1) / (top - bottom);
    T nf_inv = T(1) / (fr - nr);

    result.mData[0] =  T(2) * rl_inv;
    result.mData[5] =  T(2) * tb_inv;
    result.mData[10] = -T(2) * nf_inv;

    result.mData[12] = -(right + left) * rl_inv;
    result.mData[13] = -(top + bottom) * tb_inv;
    result.mData[14] = -(fr + nr) * nf_inv;
    result.mData[15] = T(1);

    return result;
}
```

9.1.3.230 template<typename T > Matrix<T, 4,4>& gmtl::setPerspective (Matrix< T, 4, 4 > & *result*, T *fovy*, T *aspect*, T *nr*, T *fr*) `[inline]`

Set a symmetric perspective projection matrix.

Parameters

fovy The field of view angle, in degrees, about the y-axis.

aspect The aspect ratio that determines the field of view about the x-axis. The aspect ratio is the ratio of x (width) to y (height).

zNear The distance from the viewer to the near clipping plane (always positive).

zFar The distance from the viewer to the far clipping plane (always positive).

Returns

Set matrix to perspective transform

Definition at line 688 of file Generate.h.

```
{
   gmtlASSERT( nr > 0 && fr > nr && "invalid near and far values" );
   T theta = Math::deg2Rad( fovy * T( 0.5 ) );
   T tangentTheta = Math::tan( theta );

   // tan(theta) = right / nr
   // top = tan(theta) * nr
   // right = tan(theta) * nr * aspect

   T top = tangentTheta * nr;
   T right = top * aspect; // aspect determines the fieald of view in the x-ax
   is

   // TODO: args need to match...
   return setFrustum( result, -right, top, right, -top, nr, fr );
}
```

9.1.3.231 template<typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::setPure (Quat< DATA_TYPE > & *quat*, const Vec< DATA_TYPE, 3 > & *vec*) `[inline]`

Set pure quaternion.

Precondition

vec should be normalized

Parameters

quat any quaternion object

vec a normalized vector representing an axis

Postcondition

quat will have vec as its axis, and no rotation

Definition at line 125 of file Generate.h.

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

9.1.3.232 template<typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::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 205 of file Generate.h.

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

9.1.3.233 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& gmtl::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 817 of file Generate.h.

```
{
   /* @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 bigge
   r than 4x4" );
   gmtlASSERT( Math::isEqual( lengthSquared( axisAngle.getAxis() ), (DATA_TYPE
   )1.0, (DATA_TYPE)0.001 ) /* &&
                  "you must pass in a normalized vector to setRot( mat, rad, v
   ec )" */ );

   // 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*z)+(s*y);
   result( 1, 0 ) = (t*x*y)+(s*z); result( 1, 1 ) = (t*y*y)+c;     result( 1,
   2 ) = (t*y*z)-(s*x);
   result( 2, 0 ) = (t*x*z)-(s*y); result( 2, 1 ) = (t*y*z)+(s*x); result( 2,
   2 ) = (t*z*z)+c;

   // track state
   switch (result.mState)
   {
   case Matrix<DATA_TYPE, ROWS, COLS>::TRANS:    result.mState = Matrix<DATA_T
   YPE, ROWS, COLS>::AFFINE; break;
   case Matrix<DATA_TYPE, ROWS, COLS>::IDENTITY: result.mState = Matrix<DATA_T
   YPE, ROWS, COLS>::ORTHOGONAL; break;
   }
   return result;
}
```

9.1.3.234 template<typename DATA_TYPE > AxisAngle<DATA_TYPE>& gmtl::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 408 of file Generate.h.

```
{
   return gmtl::set( result, quat );
}
```

9.1.3.235 template<typename DATATYPE , typename POS_TYPE , typename ROT_TYPE , unsigned MATCOLS, unsigned MATROWS> Coord<POS_TYPE, ROT_TYPE>& gmtl::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 1254 of file Generate.h.

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

9.1.3.236 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS, typename ROT_ORDER > Matrix<DATA_TYPE, ROWS, COLS>& gmtl::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 852 of file Generate.h.

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

   // 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]);
```

```
    DATA_TYPE sx = Math::sin( xRot );  DATA_TYPE cx = Math::cos( xRot );
    DATA_TYPE sy = Math::sin( yRot );  DATA_TYPE cy = Math::cos( yRot );
    DATA_TYPE sz = Math::sin( zRot );  DATA_TYPE 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*cy;
       result( 2, 0 ) = -cx*sy*cz + sx*sz; result( 2, 1 ) = cx*sy*sz + sx*cz;
    result( 2, 2 ) = cx*cy;
       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*sz;
       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;
    }

    // track state
    switch (result.mState)
    {
    case Matrix<DATA_TYPE, ROWS, COLS>::TRANS:    result.mState = Matrix<DATA_T
    YPE, ROWS, COLS>::AFFINE; break;
    case Matrix<DATA_TYPE, ROWS, COLS>::IDENTITY: result.mState = Matrix<DATA_T
    YPE, ROWS, COLS>::ORTHOGONAL; break;
    }
    return result;
 }
```

9.1.3.237 template<typename DATA_TYPE , typename ROT_ORDER > Quat<DATA_TYPE>& gmtl::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 269 of file Generate.h.

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

9.1.3.238 template<typename DEST_TYPE , typename DATA_TYPE > DEST_TYPE& gmtl::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 1364 of file Generate.h.

```
{
   // @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, epsi
   lon ) &&
               gmtl::Math::isEqual( gmtl::length( to ), (DATA_TYPE)1.0, epsilo
   n ) /* &&
               "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)0.1
    5, to[2] - (DATA_TYPE)0.15 ), axis;
       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 ) );
    }
 }
```

9.1.3.239 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& gmtl::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 1144 of file Generate.h.

```
 {
    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 quat
    ernion to the rotation part of a 3x3, 3x4, 4x3, or 4x4 matrix." );

    // 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);

    // track state
    switch (mat.mState)
    {
    case Matrix<DATA_TYPE, ROWS, COLS>::TRANS:    mat.mState = Matrix<DATA_TYPE
    , ROWS, COLS>::AFFINE; break;
    case Matrix<DATA_TYPE, ROWS, COLS>::IDENTITY: mat.mState = Matrix<DATA_TYPE
    , ROWS, COLS>::ORTHOGONAL; break;
    }
    return mat;
}
```

9.1.3.240 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Quat<DATA_TYPE>& gmtl::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 341 of file Generate.h.

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

9.1.3.241 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS, typename ROT_ORDER > EulerAngle<DATA_TYPE, ROT_ORDER>& gmtl::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 600 of file Generate.h.

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

9.1.3.242 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> void gmtl::setRow (Vec< DATA_TYPE, COLS > & *dest*, const Matrix< DATA_TYPE, ROWS, COLS > & *src*, unsigned *row*)

Accesses a particular row in the matrix by copying the values in the row into the given vector.

Parameters

dest the vector in which the values of the row will be placed

src the matrix being accessed

row the row of the matrix to access

Definition at line 1413 of file Generate.h.

```
{
   for (unsigned i=0; i<COLS; ++i)
   {
      dest[i] = src[row][i];
   }
}
```

9.1.3.243 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS, unsigned SIZE> Matrix<DATA_TYPE, ROWS, COLS>& gmtl::setScale (Matrix< DATA_TYPE, ROWS, COLS > & *result*, const Vec< DATA_TYPE, SIZE > & *scale*) `[inline]`

Set the scale part of a matrix.

Definition at line 770 of file Generate.h.

```
{
   gmtlASSERT( ((SIZE == (ROWS-1) && SIZE == (COLS-1)) || (SIZE == (ROWS-1) &&
    SIZE == COLS) || (SIZE == (COLS-1) && SIZE == ROWS)) && "the scale params must f
   it within the matrix, check your sizes." );
   for (unsigned x = 0; x < SIZE; ++x)
   {
      result( x, x ) = scale[x];
   }
   // track state: affine matrix with non-uniform scale now.
   result.mState = Matrix<DATA_TYPE, ROWS, COLS>::AFFINE;
   result.mState |= Matrix<DATA_TYPE, ROWS, COLS>::NON_UNISCALE;
   return result;
}
```

9.1.3.244 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& gmtl::setScale (Matrix< DATA_TYPE, ROWS, COLS > & *result*, const DATA_TYPE *scale*) `[inline]`

Sets the scale part of a matrix.

Definition at line 786 of file Generate.h.

```
{
   for (unsigned x = 0; x < Math::Min( ROWS, COLS, Math::Max( ROWS, COLS ) - 1
    ); ++x) // account for 2x4 or other weird sizes...
   {
      result( x, x ) = scale;
   }
   // track state: affine matrix with non-uniform scale now.
   result.mState = Matrix<DATA_TYPE, ROWS, COLS>::AFFINE;
   result.mState |= Matrix<DATA_TYPE, ROWS, COLS>::NON_UNISCALE;
   return result;
}
```

9.1.3.245 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS, unsigned SIZE, typename REP > Matrix<DATA_TYPE, ROWS, COLS>& gmtl::setTrans (Matrix< DATA_TYPE, ROWS, COLS > & *result*, const VecBase< DATA_TYPE, SIZE, REP > & *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 732 of file Generate.h.

```
{
   /* @todo make this a compile time assert... */
   // if n x n   then (homogeneous case) vecsize == rows-1 or (scale component
    case) vecsize == rows
   // if n x n+1 then (homogeneous case) vecsize == rows   or (scale component
    case) vecsize == rows+1
   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 documentation." );

   // homogeneous case...
   if ((ROWS == COLS && SIZE == ROWS) /* Square matrix and vec so assume homog
   eneous vector. ex. 4x4 with vec 4 */
       || (COLS == (ROWS+1) && SIZE == (ROWS+1)))  /* ex: 3x4 with vec4 */
   {
      DATA_TYPE homog_val = trans[SIZE-1];
      for (unsigned x = 0; x < COLS - 1; ++x)
         result( x, COLS - 1 ) = trans[x] / homog_val;
   }

   // non-homogeneous case...
   else
   {
      for (unsigned x = 0; x < COLS - 1; ++x)
         result( x, COLS - 1 ) = trans[x];
   }
   // track state, only override identity
   switch (result.mState)
   {
   case Matrix<DATA_TYPE, ROWS, COLS>::ORTHOGONAL: result.mState = Matrix<DATA
   _TYPE, ROWS, COLS>::AFFINE; break;
   case Matrix<DATA_TYPE, ROWS, COLS>::IDENTITY:   result.mState = Matrix<DATA
   _TYPE, ROWS, COLS>::TRANS; break;
   }
   return result;
}
```

9.1.3.246 template<typename VEC_TYPE , typename DATA_TYPE , unsigned ROWS, unsigned COLS> VEC_TYPE& gmtl::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 82 of file Generate.h.

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

   // if n x n   then (homogeneous case) vecsize == rows-1 or (scale component
    case) vecsize == rows
   // if n x n+1 then (homogeneous case) vecsize == rows   or (scale component
    case) vecsize == rows+1
   gmtlASSERT( ((ROWS == COLS && ( VEC_TYPE::Size == (ROWS-1) ||  VEC_TYPE::Si
   ze == 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 homogeneous vector. ex. 4x4 with vec 4
       || (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;
}
```

9.1.3.247 template<typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::slerp (Quat< DATA_TYPE > & *result*, const DATA_TYPE *t*, const Quat< DATA_TYPE > & *from*, const Quat< DATA_TYPE > & *to*, bool *adjustSign* = `true`)

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).

Parameters

adjustSign - If true, then slerp will operate by adjusting the sign of the slerp to take shortest path

References:

- From Adv Anim and Rendering Tech. Pg 364

See also

Quat

Definition at line 497 of file QuatOps.h.

```
{
    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 (adjustSign && (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 epsillo
     n
     {
        // Standard case (slerp)
        DATA_TYPE omega, sinom;
        omega = gmtl::Math::aCos( cosom ); // extract theta from dot product's c
     os 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;
  }
```

9.1.3.248 template<typename DATA_TYPE > void gmtl::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 463 of file QuatOps.h.

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

9.1.3.249 template<typename DATA_TYPE > Quat<DATA_TYPE>& gmtl::sub (Quat< DATA_TYPE > & *result*, const Quat< DATA_TYPE > & *q1*, const Quat< DATA_TYPE > & *q2*)

vector subtraction

See also

Quat

Definition at line 261 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;
}
```

9.1.3.250 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& gmtl::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).

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

Definition at line 119 of file MatrixOps.h.

```
{
   // 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 );

   // track state
   result.mState = combineMatrixStates( lhs.mState, rhs.mState );
   return result;
}
```

9.1.3.251 template<typename DATA_TYPE , unsigned SIZE> Matrix<DATA_TYPE, SIZE, SIZE>& gmtl::transpose (Matrix< DATA_TYPE, SIZE, SIZE > & *result*)

matrix transpose in place.

: needs to be an N x N matrix : flip along diagonal

Definition at line 231 of file MatrixOps.h.

```
{
   // 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;
}
```

9.1.3.252 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE, ROWS, COLS>& gmtl::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) : source needs to be an M x N matrix, while dest needs to be N x M : flip along diagonal

Definition at line 246 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 );
      }
   }
   result.mState = temp.mState;
   return result;
}
```

9.1.3.253 template< class DATA_TYPE > PlaneSide gmtl::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 71 of file PlaneOps.h.

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

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

9.1.3.254 template< class DATA_TYPE > PlaneSide gmtl::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 46 of file PlaneOps.h.

```
{
   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;
}
```

9.1.3.255 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Vec<DATA_TYPE, COLS>& gmtl::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...).

Parameters

result the vector to write the result in

matrix the transform matrix

vector the original vector

Postcondition

This results in a rotational 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 113 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;
}
```

9.1.3.256 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS, unsigned PNT_SIZE> Point<DATA_TYPE, PNT_SIZE>& gmtl::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]).

Parameters

result the point to write the result in

matrix the transform matrix

point the original point

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 264 of file Xforms.h.

```
{
   //gmtlSERT( PNT_SIZE == COLS - 1 && "The precondition of this method is tha
   t the vector size must be one less than the number of columns in the matrix.  eg.
    if Mat<n,k>, then Vec<k-1>." );
   GMTL_STATIC_ASSERT( PNT_SIZE == COLS-1, Point_not_of_size_mat_col_minus_1_a
   s_required_for_xform);

   // 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 unspecified elt to 1.0

   // 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;
}
```

9.1.3.257 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Ray<DATA_TYPE>& gmtl::xform (Ray< DATA_TYPE > & *result*, const Matrix< DATA_TYPE, ROWS, COLS > & *matrix*, const Ray< DATA_TYPE > & *ray*) `[inline]`

transform ray by a matrix.

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

Parameters

result the ray to write the result in

matrix the transform matrix

ray the original ray

Postcondition

This results in a full matrix xform of the ray.
returns a ray same size as the matrix rows... (p[r][1] = m[r][k] ∗ p[k][1])

Definition at line 367 of file Xforms.h.

```
{
    gmtl::Point<DATA_TYPE, 3> pos;
    gmtl::Vec<DATA_TYPE, 3> dir;
    result.setOrigin( xform( pos, matrix, ray.getOrigin() ) );
    result.setDir( xform( dir, matrix, ray.getDir() ) );
    return result;
}
```

9.1.3.258 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS, unsigned VEC_SIZE> Vec<DATA_TYPE, VEC_SIZE>& gmtl::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]).

Parameters

result the vector to write the result in

matrix the transform matrix

vector the original vector

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 158 of file Xforms.h.

```
{
   GMTL_STATIC_ASSERT( VEC_SIZE == COLS - 1, Vec_of_wrong_size_for_xform );
   // 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 to 0.0

   // 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;
}
```

9.1.3.259 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> Point<DATA_TYPE, COLS>& gmtl::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...).

Parameters

result the point to write the result in

matrix the transform matrix

point the original 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 222 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;
}
```

9.1.3.260 template<typename DATA_TYPE > VecBase<DATA_TYPE, 3>& gmtl::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).

Parameters

result The vector to write the result into

rot The quaternion

vector The original vector to transform

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 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 40 of file Xforms.h.

```
{
   // check preconditions...
   gmtlASSERT( Math::isEqual( length( rot ), (DATA_TYPE)1.0, (DATA_TYPE)0.0001
    ) && "must pass a rotation quaternion to xform(result,quat,vec) - by definition,
    a rotation quaternion is normalized).  if you need non-rotation quaternion suppo
   rt, let us know." );

   // 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_c
    onj[Zelt] - pure[Zelt]*rot_conj[Yelt],
       pure[Welt]*rot_conj[Yelt] + pure[Yelt]*rot_conj[Welt] + pure[Zelt]*rot_c
    onj[Xelt] - pure[Xelt]*rot_conj[Zelt],
       pure[Welt]*rot_conj[Zelt] + pure[Zelt]*rot_conj[Welt] + pure[Xelt]*rot_c
    onj[Yelt] - pure[Yelt]*rot_conj[Xelt],
       pure[Welt]*rot_conj[Welt] - pure[Xelt]*rot_conj[Xelt] - pure[Yelt]*rot_c
    onj[Yelt] - pure[Zelt]*rot_conj[Zelt] );

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

9.1.3.261 template<typename DATA_TYPE , unsigned ROWS, unsigned COLS> LineSeg<DATA_TYPE>& gmtl::xform (LineSeg< DATA_TYPE > & *result*, const Matrix< DATA_TYPE, ROWS, COLS > & *matrix*, const LineSeg< DATA_TYPE > & *seg*) `[inline]`

transform seg by a matrix.

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

Parameters

result the seg to write the result in

matrix the transform matrix

seg the original seg

Postcondition

This results in a full matrix xform of the seg.
returns a seg same size as the matrix rows... (p[r][1] = m[r][k] $*$ p[k][1])

Definition at line 414 of file Xforms.h.

```
{
    gmtl::Point<DATA_TYPE, 3> pos;
    gmtl::Vec<DATA_TYPE, 3> dir;
    result.setOrigin( xform( pos, matrix, seg.getOrigin() ) );
    result.setDir( xform( dir, matrix, seg.getDir() ) );
    return result;
}
```

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

zero out the matrix.

Postcondition

every element is 0.

Definition at line 53 of file MatrixOps.h.

```
{
   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.mData[x] = (DATA_TYPE)0;
      }
   }
   result.mState = Matrix<DATA_TYPE, ROWS, COLS>::ORTHOGONAL;
   return result;
}
```

9.1.4 Variable Documentation

9.1.4.1 const float gmtl::GMTL_EPSILON = 1.0e-6f

Definition at line 43 of file Defines.h.

9.1.4.2 const float gmtl::GMTL_MAT_EQUAL_EPSILON = 0.001f

Definition at line 44 of file Defines.h.

9.1.4.3 const float gmtl::GMTL_VEC_EQUAL_EPSILON = 0.0001f

Definition at line 45 of file Defines.h.

9.1.4.4 const unsigned int gmtl::IN_FRONT_OF_ALL_PLANES = 6

Definition at line 492 of file Containment.h.

9.1.4.5 const Matrix22d gmtl::MAT_IDENTITY22D = Matrix22d()

64bit floating point 2x2 identity matrix

Definition at line 506 of file Matrix.h.

9.1.4.6 const Matrix22f gmtl::MAT_IDENTITY22F = Matrix22f()

32bit floating point 2x2 identity matrix

Definition at line 503 of file Matrix.h.

9.1.4.7 const Matrix23d gmtl::MAT_IDENTITY23D = Matrix23d()

64bit floating point 2x2 identity matrix

Definition at line 512 of file Matrix.h.

9.1.4.8 const Matrix23f gmtl::MAT_IDENTITY23F = Matrix23f()

32bit floating point 2x2 identity matrix

Definition at line 509 of file Matrix.h.

9.1.4.9 const Matrix33d gmtl::MAT_IDENTITY33D = Matrix33d()

64bit floating point 3x3 identity matrix

Definition at line 518 of file Matrix.h.

9.1.4.10 const Matrix33f gmtl::MAT_IDENTITY33F = Matrix33f()

32bit floating point 3x3 identity matrix

Definition at line 515 of file Matrix.h.

9.1.4.11 const Matrix34d gmtl::MAT_IDENTITY34D = Matrix34d()

64bit floating point 3x4 identity matrix

Definition at line 524 of file Matrix.h.

9.1.4.12 const Matrix34f gmtl::MAT_IDENTITY34F = Matrix34f()

32bit floating point 3x4 identity matrix

Definition at line 521 of file Matrix.h.

9.1.4.13 const Matrix44d gmtl::MAT_IDENTITY44D = Matrix44d()

64bit floating point 4x4 identity matrix

Definition at line 530 of file Matrix.h.

9.1.4.14 const Matrix44f gmtl::MAT_IDENTITY44F = Matrix44f()

32bit floating point 4x4 identity matrix

Definition at line 527 of file Matrix.h.

9.2 gmtl::helpers Namespace Reference

Classes

- struct ConstructorCounter

Functions

- ConstructorCounter * VecCtrCounterInstance ()

9.2.1 Function Documentation

9.2.1.1 ConstructorCounter* gmtl::helpers::VecCtrCounterInstance ( ) [inline]

Definition at line 34 of file Helpers.h.

```
{
   static ConstructorCounter vec_counter;
   return &vec_counter;
}
```

9.3 gmtl::Math Namespace Reference

Functions

- template<class T >
  T clamp (T number, T lo, T hi)

  clamp "number" to a range between lo and hi

- template<class T >
  bool quadraticFormula (T &r1, T &r2, const T &a, const T &b, const T &c)

  Uses the quadratic formula to compute the 2 roots of the given 2nd degree polynomial in the form of Ax^2 + Bx + C.

C Math Abstraction

- template<typename T >
  T abs (T iValue)
- float abs (float iValue)
- double abs (double iValue)
- int abs (int iValue)
- long abs (long 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 >
  int sign (T iValue)
- template<typename T >
  T zeroClamp (T value, T eps=static_cast< 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 fastInvSqrt (float x)

  Fast inverse square root.

- float fastInvSqrt2 (float x)
- float fastInvSqrt3 (float x)
- void seedRandom (unsigned int seed)

  Seeds the pseudorandom number generator with the given seed.

- 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].

Variables

Mathematical constants

- const float TWO_PI = 6.28318530717958647692f
- const float PI = 3.14159265358979323846f
- const float PI_OVER_2 = 1.57079632679489661923f
- const float PI_OVER_4 = 0.78539816339744830962f

9.3.1 Function Documentation

9.3.1.1 template<typename T > T gmtl::Math::abs (T *iValue*) `[inline]`

Definition at line 54 of file Math.h.

```
{
    return static_cast<T>( iValue >= static_cast<T>(0) ? iValue : -iValue );
}
```

9.3.1.2 float gmtl::Math::abs (float *iValue*) `[inline]`

Definition at line 59 of file Math.h.

```
{  return fabsf(iValue); }
```

9.3.1.3 int gmtl::Math::abs (int *iValue*) `[inline]`

Definition at line 63 of file Math.h.

```
{  return ::abs(iValue); }
```

9.3.1.4 long gmtl::Math::abs (long *iValue*) `[inline]`

Definition at line 65 of file Math.h.

```
{  return labs(iValue); }
```

9.3.1.5 double gmtl::Math::abs (double *iValue*) `[inline]`

Definition at line 61 of file Math.h.

```
{  return fabs(iValue); }
```

9.3.1.6 double gmtl::Math::aCos (double *fValue*) [inline]

Definition at line 159 of file Math.h.

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

9.3.1.7 template<typename T > T gmtl::Math::aCos (T *fValue*) [inline]

9.3.1.8 float gmtl::Math::aCos (float *fValue*) [inline]

Definition at line 139 of file Math.h.

```
{
    if ( -1.0f < fValue )
    {
        if ( fValue < 1.0f )
        {
#ifdef NO_ACOSF
            return static_cast<float>(::acos(fValue));
#else
            return static_cast<float>(::acosf(fValue));
#endif
        }
        else
            return 0.0f;
    }
    else
    {
        return static_cast<float>(gmtl::Math::PI);
    }
}
```

9.3.1.9 template<typename T > T gmtl::Math::aSin (T *fValue*) [inline]

9.3.1.10 float gmtl::Math::aSin (float *fValue*) [inline]

Definition at line 176 of file Math.h.

```
{
    if ( -1.0f < fValue )
    {
        if ( fValue < 1.0f )
        {
#ifdef NO_ASINF
            return static_cast<float>(::asin(fValue));
#else
            return static_cast<float>(::asinf(fValue));
#endif
        }
        else
            return static_cast<float>(-gmtl::Math::PI_OVER_2);
    }
    else
    {
        return static_cast<float>(gmtl::Math::PI_OVER_2);
    }
}
```

9.3.1.11 double gmtl::Math::aSin (double *fValue*) [inline]

Definition at line 196 of file Math.h.

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

9.3.1.12 template<typename T > T gmtl::Math::aTan (T *fValue*) [inline]

9.3.1.13 double gmtl::Math::aTan (double *fValue*) [inline]

Definition at line 213 of file Math.h.

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

9.3.1.14 float gmtl::Math::aTan (float *fValue*) [inline]

Definition at line 217 of file Math.h.

```
{
#ifdef NO_TANF
   return static_cast<float>(::atan(fValue));
#else
   return static_cast<float>(::atanf(fValue));
#endif
}
```

9.3.1.15 template<typename T > T gmtl::Math::aTan2 (T *fY*, T *fX*) [inline]

9.3.1.16 float gmtl::Math::aTan2 (float *fY*, float *fX*) [inline]

Definition at line 228 of file Math.h.

```
{
#ifdef NO_ATAN2F
   return static_cast<float>(::atan2(fY, fX));
#else
   return static_cast<float>(::atan2f(fY, fX));
#endif
}
```

9.3.1.17 double gmtl::Math::aTan2 (double *fY*, double *fX*) [inline]

Definition at line 236 of file Math.h.

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

9.3.1.18 double gmtl::Math::ceil (double *fValue*) [inline]

Definition at line 79 of file Math.h.

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

9.3.1.19 template<typename T > T gmtl::Math::ceil (T *fValue*) [inline]

9.3.1.20 float gmtl::Math::ceil (float *fValue*) [inline]

Definition at line 71 of file Math.h.

```
{
#ifdef NO_CEILF
   return float(::ceil(fValue));
#else
   return float( ::ceilf( fValue ) );
#endif
}
```

9.3.1.21 template<class T > T gmtl::Math::clamp (T *number,* T *lo,* T *hi*) [inline]

clamp "number" to a range between lo and hi

Definition at line 570 of file Math.h.

```
{
   if (number > hi) number = hi;
   else if (number < lo) number = lo;
   return number;
}
```

9.3.1.22 template<typename T > T gmtl::Math::cos (T *fValue*) [inline]

9.3.1.23 float gmtl::Math::cos (float *fValue*) [inline]

Definition at line 243 of file Math.h.

```
{
#ifdef NO_COSF
   return static_cast<float>(::cos(fValue));
#else
   return static_cast<float>(::cosf(fValue));
#endif
}
```

9.3.1.24 double gmtl::Math::cos (double *fValue*) [inline]

Definition at line 251 of file Math.h.

```
{
    return static_cast<double>(::cos(fValue));
}
```

9.3.1.25 float gmtl::Math::deg2Rad (float *fVal*) [inline]

Definition at line 464 of file Math.h.

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

9.3.1.26 double gmtl::Math::deg2Rad (double *fVal*) [inline]

Definition at line 468 of file Math.h.

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

9.3.1.27 template<typename T > T gmtl::Math::exp (T *fValue*) [inline]

9.3.1.28 float gmtl::Math::exp (float *fValue*) [inline]

Definition at line 258 of file Math.h.

```
{
#ifdef NO_EXPF
   return static_cast<float>(::exp(fValue));
#else
   return static_cast<float>(::expf(fValue));
#endif
}
```

9.3.1.29 double gmtl::Math::exp (double *fValue*) [inline]

Definition at line 266 of file Math.h.

```
{
    return static_cast<double>(::exp(fValue));
}
```

9.3.1.30 template<class T > T gmtl::Math::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 553 of file Math.h.

```
{
   T lhs = static_cast<T>(1);

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

   return lhs;
}
```

9.3.1.31 float gmtl::Math::fastInvSqrt (float *x*) [inline]

Fast inverse square root.

Definition at line 351 of file Math.h.

```
{
   GMTL_STATIC_ASSERT(sizeof(float) == sizeof(int),
                      Union_type_sizes_do_not_match);

   // Use an approach to data type reinterpretation that is safe with GCC
   // strict aliasing enabled. This is called type-punning, and it is valid
   // when done with a union where the value read (int_value) is different
   // than the one most recently written to (float_value).
   union
   {
      float float_value;
      int   int_value;
   } data;

   const float xhalf(0.5f*x);
   data.float_value = x;
   // This hides a good amount of math
   data.int_value = 0x5f3759df - (data.int_value >> 1);
   x = data.float_value;
   x = x*(1.5f - xhalf*x*x);   // Repeat for more accuracy
   return x;
}
```

9.3.1.32 float gmtl::Math::fastInvSqrt2 (float *x*) [inline]

Definition at line 375 of file Math.h.

```
{
   GMTL_STATIC_ASSERT(sizeof(float) == sizeof(int),
                      Union_type_sizes_do_not_match);

   // Use an approach to data type reinterpretation that is safe with GCC
   // strict aliasing enabled. This is called type-punning, and it is valid
   // when done with a union where the value read (int_value) is different
   // than the one most recently written to (float_value).
   union
   {
      float float_value;
      int   int_value;
   } data;

   const float xhalf(0.5f*x);
   data.float_value = x;
   // This hides a good amount of math
   data.int_value = 0x5f3759df - (data.int_value >> 1);
   x = data.float_value;
   x = x*(1.5f - xhalf*x*x);   // Repeat for more accuracy
   x = x*(1.5f - xhalf*x*x);
   return x;
}
```

9.3.1.33 float gmtl::Math::fastInvSqrt3 (float *x*) [inline]

Definition at line 400 of file Math.h.

```
{
   GMTL_STATIC_ASSERT(sizeof(float) == sizeof(int),
                      Union_type_sizes_do_not_match);

   // Use an approach to data type reinterpretation that is safe with GCC
   // strict aliasing enabled. This is called type-punning, and it is valid
   // when done with a union where the value read (int_value) is different
   // than the one most recently written to (float_value).
   union
   {
      float float_value;
      int   int_value;
   } data;

   const float xhalf(0.5f*x);
   data.float_value = x;
   // This hides a good amount of math
   data.int_value = 0x5f3759df - (data.int_value >> 1);
   x = data.float_value;
   x = x*(1.5f - xhalf*x*x);   // Repeat for more accuracy
   x = x*(1.5f - xhalf*x*x);
```

```
   x = x*(1.5f - xhalf*x*x);
   return x;
}
```

9.3.1.34 template<typename T > T gmtl::Math::floor (T *fValue*) [inline]

9.3.1.35 float gmtl::Math::floor (float *fValue*) [inline]

Definition at line 86 of file Math.h.

```
{
#ifdef NO_FLOORF
   return float(::floor(fValue));
#else
   return float( ::floorf( fValue ) );
#endif
}
```

9.3.1.36 double gmtl::Math::floor (double *fValue*) [inline]

Definition at line 94 of file Math.h.

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

9.3.1.37 template<class T > bool gmtl::Math::isEqual (const T & *a*, const T & *b*, const T & *tolerance*) [inline]

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

: tolerance must be >= 0

Definition at line 488 of file Math.h.

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

9.3.1.38 template<class T , typename U > void gmtl::Math::lerp (T & *result*, const U & *lerp*, const T & *a*, const T & *b*) [inline]

Linear Interpolation between number [a] and [b].

lerp=0.0 returns a, lerp=1.0 returns b

Precondition

use double or float only...

Definition at line 587 of file Math.h.

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

9.3.1.39 double gmtl::Math::log (double *fValue*) [inline]

Definition at line 273 of file Math.h.

```
{
    return static_cast<double>(::log(fValue));
}
```

9.3.1.40 float gmtl::Math::log (float *fValue*) [inline]

Definition at line 277 of file Math.h.

```
{
#ifdef NO_LOGF
   return static_cast<float>(::log(fValue));
#else
   return static_cast<float>(::logf(fValue));
#endif
}
```

9.3.1.41 template<typename T > T gmtl::Math::log (T *fValue*) [inline]

9.3.1.42 template<class T > T gmtl::Math::Max (const T & *w*, const T & *x*, const T & *y*, const T & *z*) [inline]

max returns the maximum of 4 values

Definition at line 542 of file Math.h.

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

9.3.1.43 template<class T > T gmtl::Math::Max (const T & *x*, const T & *y*, const T & *z*) `[inline]`

max returns the maximum of 3 values

Definition at line 536 of file Math.h.

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

9.3.1.44 template<class T > T gmtl::Math::Max (const T & *x*, const T & *y*) `[inline]`

max returns the maximum of 2 values

Definition at line 530 of file Math.h.

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

9.3.1.45 template<class T > T gmtl::Math::Min (const T & *x*, const T & *y*, const T & *z*) `[inline]`

min returns the minimum of 3 values

Definition at line 517 of file Math.h.

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

9.3.1.46 template<class T > T gmtl::Math::Min (const T & *w*, const T & *x*, const T & *y*, const T & *z*) `[inline]`

min returns the minimum of 4 values

Definition at line 523 of file Math.h.

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

9.3.1.47 template<class T > T gmtl::Math::Min (const T & *x*, const T & *y*) [inline]

min returns the minimum of 2 values

Definition at line 511 of file Math.h.

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

9.3.1.48 double gmtl::Math::pow (double *fBase*, double *fExponent*) [inline]

Definition at line 286 of file Math.h.

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

9.3.1.49 float gmtl::Math::pow (float *fBase*, float *fExponent*) [inline]

Definition at line 290 of file Math.h.

```
{
#ifdef NO_POWF
   return static_cast<float>(::pow(fBase, fExponent));
#else
   return static_cast<float>(::powf(fBase, fExponent));
#endif
}
```

9.3.1.50 template<class T > bool gmtl::Math::quadraticFormula (T & *r1*, T & *r2*, const T & *a*, const T & *b*, const T & *c*) [inline]

Uses the quadratic formula to compute the 2 roots of the given 2nd degree polynomial in the form of Ax^2 + Bx + C.

Parameters

r1 set to the first root
r2 set to the second root
a the coefficient to x^2
b the coefficient to x^1
c the coefficient to x^0

Returns

true if both r1 and r2 are real; false otherwise

Definition at line 607 of file Math.h.

```
{
   const T q = b * b - static_cast<T>(4) * a * c;

   // the result has real roots
   if (q >= 0)
   {
      const T sq = gmtl::Math::sqrt(q);
      const T d = static_cast<T>(1) / (static_cast<T>(2) * a);
      r1 = (-b + sq) * d;
      r2 = (-b - sq) * d;
      return true;
   }
   // the result has complex roots
   else
   {
      return false;
   }
}
```

9.3.1.51 float gmtl::Math::rad2Deg (float *fVal*) `[inline]`

Definition at line 473 of file Math.h.

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

9.3.1.52 double gmtl::Math::rad2Deg (double *fVal*) `[inline]`

Definition at line 477 of file Math.h.

```
{
   return static_cast<float>(fVal * static_cast<double>(180.0 / gmtl::Math::PI));

}
```

9.3.1.53 float gmtl::Math::rangeRandom (float *x1*, float *x2*) `[inline]`

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

Definition at line 449 of file Math.h.

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

9.3.1.54 template<class T > T gmtl::Math::round (T *p*) `[inline]`

round to nearest integer

Definition at line 504 of file Math.h.

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

9.3.1.55 void gmtl::Math::seedRandom (unsigned int *seed*) `[inline]`

Seeds the pseudorandom number generator with the given seed.

Parameters

seed the seed for the pseudorandom number generator.

Definition at line 433 of file Math.h.

```
{
   ::srand(seed);
}
```

9.3.1.56 template<typename T > int gmtl::Math::sign (T *iValue*) `[inline]`

Definition at line 100 of file Math.h.

```
{
   if (iValue > static_cast<T>(0))
   {
      return 1;
   }
   else
   {
      if (iValue < static_cast<T>(0))
      {
         return -1;
      }
      else
      {
         return 0;
      }
   }
}
```

9.3.1.57 float gmtl::Math::sin (float *fValue*) `[inline]`

Definition at line 305 of file Math.h.

```
{
#ifdef NO_SINF
   return static_cast<float>(::sin(fValue));
#else
   return static_cast<float>(::sinf(fValue));
#endif
}
```

9.3.1.58 double gmtl::Math::sin (double *fValue*) `[inline]`

Definition at line 301 of file Math.h.

```
{
    return static_cast<double>(::sin(fValue));
}
```

9.3.1.59 template<typename T > T gmtl::Math::sin (T *fValue*) `[inline]`

9.3.1.60 template<typename T > T gmtl::Math::sqr (T *fValue*) `[inline]`

Definition at line 330 of file Math.h.

```
{
    return static_cast<T>(fValue * fValue);
}
```

9.3.1.61 template<typename T > T gmtl::Math::sqrt (T *fValue*) `[inline]`

Definition at line 336 of file Math.h.

```
{
#ifdef NO_SQRTF
   return static_cast<T>(::sqrt((static_cast<float>(fValue))));
#else
   return static_cast<T>(::sqrtf((static_cast<float>(fValue))));
#endif
}
```

9.3.1.62 double gmtl::Math::sqrt (double *fValue*) `[inline]`

Definition at line 344 of file Math.h.

```
{
    return static_cast<double>(::sqrt(fValue));
}
```

9.3.1.63 float gmtl::Math::tan (float *fValue*) `[inline]`

Definition at line 320 of file Math.h.

```
{
#ifdef NO_TANF
   return static_cast<float>(::tan(fValue));
#else
   return static_cast<float>(::tanf(fValue));
#endif
}
```

9.3.1.64 template<typename T > T gmtl::Math::tan (T *fValue*) `[inline]`

9.3.1.65 double gmtl::Math::tan (double *fValue*) `[inline]`

Definition at line 316 of file Math.h.

```
{
    return static_cast<double>(::tan(fValue));
}
```

9.3.1.66 template<class T > T gmtl::Math::trunc (T *val*) [inline]

cut off the digits after the decimal place

Definition at line 496 of file Math.h.

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

9.3.1.67 float gmtl::Math::unitRandom ( ) [inline]

get a random number between 0 and 1

Postcondition

returns number between 0 and 1

Definition at line 441 of file Math.h.

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

9.3.1.68 template<typename T > T gmtl::Math::zeroClamp (T *value*, T *eps* = *static_cast*<*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 128 of file Math.h.

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

9.3.2 Variable Documentation

9.3.2.1 const float gmtl::Math::PI = 3.14159265358979323846f

Definition at line 43 of file Math.h.

9.3.2.2 const float gmtl::Math::PI_OVER_2 = 1.57079632679489661923f

Definition at line 44 of file Math.h.

9.3.2.3 const float gmtl::Math::PI_OVER_4 = 0.78539816339744830962f

Definition at line 45 of file Math.h.

9.3.2.4 const float gmtl::Math::TWO_PI = 6.28318530717958647692f

Definition at line 42 of file Math.h.

9.4 gmtl::meta Namespace Reference

Classes

- struct AssignVecUnrolled
- struct AssignVecUnrolled< 0, T >
- struct AssignArrayUnrolled
- struct AssignArrayUnrolled< 0, T >
- struct DefaultVecTag
- struct ScalarArg

  template to hold a scalar argument.

- struct ExprTraits

  Traits class for expression template parameters.

- struct ExprTraits< VecBase< T, SIZE, ScalarArg< T > > >
- struct ExprTraits< VecBase< T, SIZE, DefaultVecTag > >
- struct VecBinaryExpr

  Binary vector expression.

- struct VecUnaryExpr

  Unary vector expression.

- struct VecPlusBinary
- struct VecMinusBinary
- struct VecMultBinary
- struct VecDivBinary
- struct VecNegUnary

  Negation of the values.

- struct DotVecUnrolled

  meta class to unroll dot products.

- struct DotVecUnrolled< 0, T1, T2 >

  base cas for dot product unrolling.

- struct LenSqrVecUnrolled

  meta class to unroll length squared operation.

- struct LenSqrVecUnrolled< 0, T >

  base cas for dot product unrolling.

- struct EqualVecUnrolled

  meta class to test vector equality.

- struct EqualVecUnrolled< 0, VT >

  base cas for dot product unrolling.

Functions

- template<typename T >
  ScalarArg< T > makeScalarArg (T val)

9.4.1 Function Documentation

9.4.1.1 template<typename T > ScalarArg<T> gmtl::meta::makeScalarArg (T *val*) [inline]

Definition at line 50 of file VecExprMeta.h.

```
{ return ScalarArg<T>(val); }
```

9.5 gmtl::output Namespace Reference

Classes

- struct VecOutputter

  Outputters for vector types.

- struct VecOutputter< DATA_TYPE, SIZE, gmtl::meta::DefaultVecTag >

Chapter 10

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 Member Functions

- AABox ()

  Creates a new empty box.

- AABox (const Point< DATA_TYPE, 3 > &min, const Point< 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 Point< DATA_TYPE, 3 > & getMin () const

  Gets the minimum point of the box.

- const Point< DATA_TYPE, 3 > & getMax () const

    Gets the maximum point of the box.

- bool isEmpty () const

    Tests if this box occupies no space.

- void setMin (const Point< DATA_TYPE, 3 > &min)

    Sets the minimum point of the box.

- void setMax (const Point< DATA_TYPE, 3 > &max)

    Sets the maximum point of the box.

- void setEmpty (bool empty)

    Sets the empty flag on this box.

Public Attributes

- Point< DATA_TYPE, 3 > mMin

    The minimum point of the box.

- Point< 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 22 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 33 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 39 of file AABox.h.

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

10.1.3.2 template<class DATA_TYPE> gmtl::AABox< DATA_TYPE >::AABox (const Point< DATA_TYPE, 3 > & *min*, const Point< 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 52 of file AABox.h.

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

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 61 of file AABox.h.

```
    : mMin(box.mMin), mMax(box.mMax), mEmpty(box.mEmpty)
{}
```

10.1.4 Member Function Documentation

10.1.4.1 template<class DATA_TYPE> const Point<DATA_TYPE, 3>& gmtl::AABox< DATA_TYPE >::getMax ( ) const [inline]

Gets the maximum point of the box.

Returns

the max point

Definition at line 80 of file AABox.h.

```
{
    return mMax;
}
```

10.1.4.2 template<class DATA_TYPE> const Point<DATA_TYPE, 3>& gmtl::AABox< DATA_TYPE >::getMin ( ) const [inline]

Gets the minimum point of the box.

Returns

the min point

Definition at line 70 of file AABox.h.

```
{
    return mMin;
}
```

10.1.4.3 template<class DATA_TYPE> bool gmtl::AABox< DATA_TYPE >::isEmpty ( ) const [inline]

Tests if this box occupies no space.

Returns

true if the box is empty, false otherwise

Definition at line 90 of file AABox.h.

```
{
   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 120 of file AABox.h.

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

10.1.4.5 template<class DATA_TYPE> void gmtl::AABox< DATA_TYPE >::setMax (const Point< DATA_TYPE, 3 > & *max*) [inline]

Sets the maximum point of the box.

Parameters

max the max point

Definition at line 110 of file AABox.h.

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

10.1.4.6 template<class DATA_TYPE> void gmtl::AABox< DATA_TYPE >::setMin (const Point< DATA_TYPE, 3 > & *min*) `[inline]`

Sets the minimum point of the box.

Parameters

min the min point

Definition at line 100 of file AABox.h.

```
{
   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 139 of file AABox.h.

10.1.5.2 template<class DATA_TYPE> Point<DATA_TYPE, 3> gmtl::AABox< DATA_TYPE >::mMax

The maximum point on the box.

Definition at line 134 of file AABox.h.

10.1.5.3 template<class DATA_TYPE> Point<DATA_TYPE, 3> gmtl::AABox< DATA_TYPE >::mMin

The minimum point of the box.

Definition at line 129 of file AABox.h.

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

- AABox.h

10.2 gmtl::meta::AssignArrayUnrolled< ELT, T > Struct Template Reference

`#include <Meta.h>`

Static Public Member Functions

- static void func (T *lVec, const T *rVec)

10.2.1 Detailed Description

template<int ELT, typename T> struct gmtl::meta::AssignArrayUnrolled< ELT, T >

Definition at line 88 of file Meta.h.

10.2.2 Member Function Documentation

10.2.2.1 template<int ELT, typename T > static void gmtl::meta::AssignArrayUnrolled< ELT, T >::func (T * *lVec*, const T * *rVec*) `[inline, static]`

Definition at line 90 of file Meta.h.

```
{
   AssignArrayUnrolled<ELT-1,T>::func(lVec, rVec);
   lVec[ELT] = rVec[ELT];
}
```

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

- Meta.h

10.3 gmtl::meta::AssignArrayUnrolled< 0, T > Struct Template Reference

`#include <Meta.h>`

Static Public Member Functions

- static void func (T *lVec, const T *rVec)

10.3.1 Detailed Description

template<typename T> struct gmtl::meta::AssignArrayUnrolled< 0, T >

Definition at line 98 of file Meta.h.

10.3.2 Member Function Documentation

10.3.2.1 template<typename T > static void gmtl::meta::AssignArrayUnrolled< 0, T >::func (T * *lVec*, const T * *rVec*) [inline, static]

Definition at line 100 of file Meta.h.

```
{ lVec[0] = rVec[0]; }
```

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

- Meta.h

10.4 gmtl::meta::AssignVecUnrolled< ELT, T > Struct Template Reference

```
#include <Meta.h>
```

Static Public Member Functions

- static void func (T &lVec, const T &rVec)

10.4.1 Detailed Description

template<int ELT, typename T> struct gmtl::meta::AssignVecUnrolled< ELT, T >

Definition at line 70 of file Meta.h.

10.4.2 Member Function Documentation

10.4.2.1 template<int ELT, typename T > static void gmtl::meta::AssignVecUnrolled< ELT, T >::func (T & *lVec*, const T & *rVec*) [inline, static]

Definition at line 72 of file Meta.h.

```
{
   AssignVecUnrolled<ELT-1,T>::func(lVec, rVec);
   lVec[ELT] = rVec[ELT];
}
```

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

- Meta.h

10.5 gmtl::meta::AssignVecUnrolled< 0, T > Struct Template Reference

`#include <Meta.h>`

Static Public Member Functions

- static void func (T &lVec, const T &rVec)

10.5.1 Detailed Description

template<typename T> struct gmtl::meta::AssignVecUnrolled< 0, T >

Definition at line 80 of file Meta.h.

10.5.2 Member Function Documentation

10.5.2.1 template<typename T > static void gmtl::meta::AssignVecUnrolled< 0, T >::func (T & *lVec*, const T & *rVec*) [inline, static]

Definition at line 82 of file Meta.h.

```
{ lVec[0] = rVec[0]; }
```

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

- Meta.h

10.6 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< DATA_TYPE >:

Collaboration diagram for gmtl::AxisAngle< DATA_TYPE >:

Public Types

- enum Params { Size = 4 }

  The number of components this VecB has.

Public Member Functions

- 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)

  set the angle (twist) part of the AxisAngle, as a radian value.

- Vec< DATA_TYPE, 3 > getAxis () const

  get the axis portion of the AxisAngle

- const DATA_TYPE & getAngle () const

  get the angle (twist) part of the AxisAngle.

10.6.1 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 35 of file AxisAngle.h.

10.6.2 Member Enumeration Documentation

10.6.2.1 template<typename DATA_TYPE> enum gmtl::AxisAngle::Params

The number of components this VecB has.

Enumerator:

Size

Reimplemented from gmtl::VecBase< DATA_TYPE, 4 >.

Definition at line 38 of file AxisAngle.h.

```
{ Size = 4 };
```

10.6.3 Constructor & Destructor Documentation

10.6.3.1 template<typename DATA_TYPE> gmtl::AxisAngle< DATA_TYPE >::AxisAngle ( ) [inline]

default constructor.

initializes to identity rotation (no rotation).

Definition at line 41 of file AxisAngle.h.

```
                :
      VecBase<DATA_TYPE, 4>( (DATA_TYPE)0.0, (DATA_TYPE)1.0,
                             (DATA_TYPE)0.0, (DATA_TYPE)0.0 )
   {
   }
```

10.6.3.2 template<typename DATA_TYPE> gmtl::AxisAngle< DATA_TYPE >::AxisAngle (const AxisAngle< DATA_TYPE > & *e*) [inline]

copy constructor.

Definition at line 48 of file AxisAngle.h.

```
                                   : VecBase<DATA_TYPE, 4>( e )
   {
   }
```

10.6.3.3 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 53 of file AxisAngle.h.

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

10.6.3.4 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 60 of file AxisAngle.h.

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

10.6.4 Member Function Documentation

10.6.4.1 template<typename DATA_TYPE> const DATA_TYPE& gmtl::AxisAngle< DATA_TYPE >::getAngle ( ) const `[inline]`

get the angle (twist) part of the AxisAngle.

Returns

the twist value in radians

Definition at line 111 of file AxisAngle.h.

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

10.6.4.2 template<typename DATA_TYPE> Vec<DATA_TYPE, 3> gmtl::AxisAngle< DATA_TYPE >::getAxis ( ) const [inline]

get the axis portion of the AxisAngle

Returns

a vector of the axis, which may or may not be normalized.

Definition at line 101 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 ) );
}
```

10.6.4.3 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 73 of file AxisAngle.h.

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

10.6.4.4 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.

Definition at line 66 of file AxisAngle.h.

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

10.6.4.5 template<typename DATA_TYPE> void gmtl::AxisAngle< DATA_TYPE >::setAngle (const DATA_TYPE & *rad_angle*) [inline]

set the angle (twist) part of the AxisAngle, as a radian value.

Parameters

rad_angle the desired twist angle, in radians

Postcondition

the angle of the object is set

Definition at line 93 of file AxisAngle.h.

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

10.6.4.6 template<typename DATA_TYPE> void gmtl::AxisAngle< DATA_TYPE >::setAxis (const Vec< DATA_TYPE, 3 > & *axis*) [inline]

set the axis portion of the AxisAngle

Parameters

axis the desired 3D vector axis to rotate about

Postcondition

the axis of the object is set

Definition at line 82 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

10.7 gmtl::CompareIndexPointProjections Struct Reference

```
#include <Comparitors.h>
```

Public Member Functions

- CompareIndexPointProjections ()
- bool operator() (const unsigned x, const unsigned y)

Public Attributes

- const std::vector< Point3 > * points
- gmtl::Vec3 sortDir

10.7.1 Detailed Description

Definition at line 22 of file Comparitors.h.

10.7.2 Constructor & Destructor Documentation

10.7.2.1 gmtl::CompareIndexPointProjections::CompareIndexPointProjections ( ) [inline]

Definition at line 25 of file Comparitors.h.

```
                                      : points(NULL)
{;}
```

10.7.3 Member Function Documentation

10.7.3.1 bool gmtl::CompareIndexPointProjections::operator() (const unsigned *x*, const unsigned *y*) [inline]

Definition at line 28 of file Comparitors.h.

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

   return (xVal < yVal);
}
```

10.7.4 Member Data Documentation

10.7.4.1 const std::vector<Point3>* gmtl::CompareIndexPointProjections::points

Definition at line 36 of file Comparitors.h.

10.7.4.2 gmtl::Vec3 gmtl::CompareIndexPointProjections::sortDir

Definition at line 37 of file Comparitors.h.

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

- Comparitors.h

10.8 gmtl::CompileTimeError< true > Struct Template Reference

`#include <StaticAssert.h>`

10.8.1 Detailed Description

template<> struct gmtl::CompileTimeError< true >

Definition at line 20 of file StaticAssert.h.

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

- StaticAssert.h

10.9 gmtl::Matrix< DATA_TYPE, ROWS, COLS >::ConstRowAccessor Class Reference

Helper class for Matrix op[] const.

`#include <Matrix.h>`

Public Types

- typedef DATA_TYPE DataType

Public Member Functions

- ConstRowAccessor (const Matrix< DATA_TYPE, ROWS, COLS > *mat, const unsigned row)
- const DATA_TYPE & operator[ ] (const unsigned column) const

Public Attributes

- const Matrix< DATA_TYPE, ROWS, COLS > * mMat
- unsigned mRow

10.9.1 Detailed Description

template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> class gmtl::Matrix< DATA_TYPE, ROWS, COLS >::ConstRowAccessor

Helper class for Matrix op[] const. This class encapsulates the row that the user is accessing and implements a new op[] that passes the column to use

Definition at line 157 of file Matrix.h.

10.9.2 Member Typedef Documentation

10.9.2.1 template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> typedef DATA_TYPE gmtl::Matrix< DATA_TYPE, ROWS, COLS >::ConstRowAccessor::DataType

Definition at line 160 of file Matrix.h.

10.9.3 Constructor & Destructor Documentation

10.9.3.1 template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> gmtl::Matrix< DATA_TYPE, ROWS, COLS >::ConstRowAccessor::ConstRowAccessor (const Matrix< DATA_TYPE, ROWS, COLS > * *mat*, const unsigned *row*) `[inline]`

Definition at line 162 of file Matrix.h.

```
    : mMat( mat ), mRow( row )
{
    gmtlASSERT( row < ROWS );
    gmtlASSERT( NULL != mat );
}
```

10.9.4 Member Function Documentation

10.9.4.1 template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> const DATA_TYPE& gmtl::Matrix< DATA_TYPE, ROWS, COLS >::ConstRowAccessor::operator[ ] (const unsigned *column*) const [inline]

Definition at line 170 of file Matrix.h.

```
{
   gmtlASSERT(column < COLS);
   return (*mMat)(mRow,column);
}
```

10.9.5 Member Data Documentation

10.9.5.1 template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> const Matrix<DATA_TYPE,ROWS,COLS>* gmtl::Matrix< DATA_TYPE, ROWS, COLS >::ConstRowAccessor::mMat

Definition at line 176 of file Matrix.h.

10.9.5.2 template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> unsigned gmtl::Matrix< DATA_TYPE, ROWS, COLS >::ConstRowAccessor::mRow

Definition at line 177 of file Matrix.h.

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

- Matrix.h

10.10 gmtl::helpers::ConstructorCounter Struct Reference

`#include <Helpers.h>`

Public Member Functions

- ConstructorCounter ()
- void inc ()
- unsigned get ()

Public Attributes

- unsigned mCount

10.10.1 Detailed Description

Definition at line 17 of file Helpers.h.

10.10.2 Constructor & Destructor Documentation

10.10.2.1 gmtl::helpers::ConstructorCounter::ConstructorCounter ( ) [inline]

Definition at line 21 of file Helpers.h.

```
{ mCount = 0; }
```

10.10.3 Member Function Documentation

10.10.3.1 unsigned gmtl::helpers::ConstructorCounter::get ( ) [inline]

Definition at line 26 of file Helpers.h.

```
{ return mCount; }
```

10.10.3.2 void gmtl::helpers::ConstructorCounter::inc ( ) [inline]

Definition at line 24 of file Helpers.h.

```
{ mCount += 1; }
```

10.10.4 Member Data Documentation

10.10.4.1 unsigned gmtl::helpers::ConstructorCounter::mCount

Definition at line 19 of file Helpers.h.

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

- Helpers.h

10.11 gmtl::Coord< POS_TYPE, ROT_TYPE > Class Template Reference

coord is a position/rotation pair.

`#include <Coord.h>`

Public Types

- enum Params { PosSize = POS_TYPE::Size, RotSize = ROT_TYPE::Size }
- typedef POS_TYPE::DataType DataType
- typedef POS_TYPE PosDataType
- typedef ROT_TYPE RotDataType

Public Member Functions

- 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

Multi-arg Constructors

Construct objects from primitive types Just assigns values in order to the pos and rot members' members.

- Coord (DataType a0, DataType a1, DataType a2, DataType a3, DataType a4, DataType a5)
- Coord (DataType a0, DataType a1, DataType a2, DataType a3, DataType a4, DataType a5, DataType a6)
- Coord (DataType a0, DataType a1, DataType a2, DataType a3, DataType a4, DataType a5, DataType a6, DataType a7)

Public Attributes

- POS_TYPE mPos

    const accessor to the position element

- ROT_TYPE mRot

10.11.1 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 Vector/Euler pair (32 bit float precision):"

```
Coord<Vec3f, EulerAngleXYZf> myEulerCoord;
```

"Or use the built in typedefs:"

```
CoordVec3fEulerAngleXYZf myEulerCoord;
Coord3fQuat myOtherEulerCoord;
```

See also

Vec, AxisAngle, EulerAngle

Definition at line 37 of file Coord.h.

10.11.2 Member Typedef Documentation

10.11.2.1 template<typename POS_TYPE, typename ROT_TYPE> typedef POS_TYPE::DataType gmtl::Coord< POS_TYPE, ROT_TYPE >::DataType

Definition at line 44 of file Coord.h.

10.11.2.2 template<typename POS_TYPE, typename ROT_TYPE> typedef POS_TYPE gmtl::Coord< POS_TYPE, ROT_TYPE >::PosDataType

Definition at line 45 of file Coord.h.

10.11.2.3 template<typename POS_TYPE, typename ROT_TYPE> typedef ROT_TYPE gmtl::Coord< POS_TYPE, ROT_TYPE >::RotDataType

Definition at line 46 of file Coord.h.

10.11.3 Member Enumeration Documentation

10.11.3.1 template<typename POS_TYPE, typename ROT_TYPE> enum gmtl::Coord::Params

Enumerator:

PosSize

RotSize

Definition at line 47 of file Coord.h.

```
{
    PosSize = POS_TYPE::Size,
    RotSize = ROT_TYPE::Size
};
```

10.11.4 Constructor & Destructor Documentation

10.11.4.1 template<typename POS_TYPE, typename ROT_TYPE> gmtl::Coord< POS_TYPE, ROT_TYPE >::Coord ( ) [inline]

Definition at line 40 of file Coord.h.

```
        : mPos(), mRot()
{
}
```

10.11.4.2 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 53 of file Coord.h.

```
                                              : mPos( coord.mPos ), mRot( co
   ord.mRot )
{
}
```

10.11.4.3 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 57 of file Coord.h.

```
                                                   : mPos( pos ), mRot( rot )
{
}
```

10.11.4.4 template<typename POS_TYPE, typename ROT_TYPE> gmtl::Coord< POS_TYPE, ROT_TYPE >::Coord (DataType *a0*, DataType *a1*, DataType *a2*, DataType *a3*, DataType *a4*, DataType *a5*) [inline]

Definition at line 66 of file Coord.h.

```
{
   GMTL_STATIC_ASSERT(PosSize == 3, Using_incorrect_number_of_args_for_type_si
   ze);
   GMTL_STATIC_ASSERT(RotSize == 3, Using_incorrect_number_of_args_for_type_si
   ze);
   if(PosSize == 3)
   {
      mPos[0] = a0; mPos[1] = a1; mPos[2] = a2;
      mRot[0] = a3; mRot[1] = a4; mRot[2] = a5;
   }
   else
   {
      gmtlASSERT(false && "Constructor not supported for pos size");
   }
}
```

10.11.4.5 template<typename POS_TYPE, typename ROT_TYPE> gmtl::Coord< POS_TYPE, ROT_TYPE >::Coord (DataType *a0*, DataType *a1*, DataType *a2*, DataType *a3*, DataType *a4*, DataType *a5*, DataType *a6*) [inline]

Definition at line 81 of file Coord.h.

```
{
   GMTL_STATIC_ASSERT( (PosSize == 3 && RotSize == 4) || (PosSize == 4 &&
   RotSize == 3), Using_incorrect_number_of_args_for_type_size);
   if(PosSize == 3)
   {
      mPos[0] = a0; mPos[1] = a1; mPos[2] = a2;
      mRot[0] = a3; mRot[1] = a4; mRot[2] = a5; mRot[3] = a6;
```

```
      }
      else if(PosSize == 4)
      {
         mPos[0] = a0; mPos[1] = a1; mPos[2] = a2; mPos[3] = a3;
         mRot[0] = a4; mRot[1] = a5; mRot[2] = a6;
      }
      else
      {
         gmtlASSERT(false && "Constructor not supported for pos size");
      }

   }
```

10.11.4.6 template<typename POS_TYPE, typename ROT_TYPE> gmtl::Coord< POS_TYPE, ROT_TYPE >::Coord (DataType *a0*, DataType *a1*, DataType *a2*, DataType *a3*, DataType *a4*, DataType *a5*, DataType *a6*, DataType *a7*) `[inline]`

Definition at line 101 of file Coord.h.

```
   {
      GMTL_STATIC_ASSERT(PosSize == 4, Using_incorrect_number_of_args_for_type_si
      ze);
      GMTL_STATIC_ASSERT(RotSize == 4, Using_incorrect_number_of_args_for_type_si
      ze);
      if(PosSize == 4)
      {
         mPos[0] = a0; mPos[1] = a1; mPos[2] = a2; mPos[3] = a3;
         mRot[0] = a4; mRot[1] = a5; mRot[2] = a6; mRot[3] = a7;
      }
      else
      {
         gmtlASSERT(false && "Constructor not supported for pos size");
      }
   }
```

10.11.5 Member Function Documentation

10.11.5.1 template<typename POS_TYPE, typename ROT_TYPE> const POS_TYPE& gmtl::Coord< POS_TYPE, ROT_TYPE >::getPos ( ) const `[inline]`

Definition at line 117 of file Coord.h.

```
{ return mPos; }
```

10.11.5.2 template<typename POS_TYPE, typename ROT_TYPE> const ROT_TYPE& gmtl::Coord< POS_TYPE, ROT_TYPE >::getRot ( ) const [inline]

Definition at line 118 of file Coord.h.

```
{ return mRot; }
```

10.11.5.3 template<typename POS_TYPE, typename ROT_TYPE> POS_TYPE& gmtl::Coord< POS_TYPE, ROT_TYPE >::pos ( ) [inline]

accessor to the position element

Todo

what about having a pos, and a const_pos naming convention?
what about having a rot, and a const_rot naming convention?

Definition at line 124 of file Coord.h.

```
{ return mPos; }
```

10.11.5.4 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 127 of file Coord.h.

```
{ return mRot; }
```

10.11.6 Member Data Documentation

10.11.6.1 template<typename POS_TYPE, typename ROT_TYPE> POS_TYPE gmtl::Coord< POS_TYPE, ROT_TYPE >::mPos

const accessor to the position element

const accessor to the rotation element

Definition at line 136 of file Coord.h.

10.11.6.2 template<typename POS_TYPE, typename ROT_TYPE> ROT_TYPE gmtl::Coord< POS_TYPE, ROT_TYPE >::mRot

Definition at line 137 of file Coord.h.

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

- Coord.h

10.12 gmtl::CubicCurve< DATA_TYPE, SIZE > Class Template Reference

A representation of a cubic curve with order set to 4.

`#include <ParametricCurve.h>`

Inheritance diagram for gmtl::CubicCurve< DATA_TYPE, SIZE >:

Collaboration diagram for gmtl::CubicCurve< DATA_TYPE, SIZE >:

Public Member Functions

- CubicCurve ()
- CubicCurve (const CubicCurve &other)
- ~CubicCurve ()
- CubicCurve & operator= (const CubicCurve &other)
- void makeBezier ()
- void makeCatmullRom ()
- void makeHermite ()
- void makeBspline ()

10.12.1 Detailed Description

template<typename DATA_TYPE, unsigned SIZE> class gmtl::CubicCurve< DATA_TYPE, SIZE >

A representation of a cubic curve with order set to 4.

Template Parameters

DATA_TYPE The data type to use for the components.

SIZE The number of components this curve has.

Definition at line 303 of file ParametricCurve.h.

10.12.2 Constructor & Destructor Documentation

10.12.2.1 template<typename DATA_TYPE , unsigned SIZE> gmtl::CubicCurve< DATA_TYPE, SIZE >::CubicCurve ( )

Definition at line 318 of file ParametricCurve.h.

```
{
}
```

10.12.2.2 template<typename DATA_TYPE , unsigned SIZE> gmtl::CubicCurve< DATA_TYPE, SIZE >::CubicCurve (const CubicCurve< DATA_TYPE, SIZE > & *other*)

Definition at line 323 of file ParametricCurve.h.

```
{
   *this = other;
}
```

10.12.2.3 template<typename DATA_TYPE , unsigned SIZE> gmtl::CubicCurve< DATA_TYPE, SIZE >::~CubicCurve ( )

Definition at line 329 of file ParametricCurve.h.

```
{
}
```

10.12.3 Member Function Documentation

10.12.3.1 template<typename DATA_TYPE , unsigned SIZE> void gmtl::CubicCurve< DATA_TYPE, SIZE >::makeBezier ( )

Definition at line 343 of file ParametricCurve.h.

```
{
   mBasisMatrix.set(
      -1.0, 3.0, -3.0, 1.0,
      3.0, -6.0, 3.0, 0.0,
      -3.0, 3.0, 0.0, 0.0,
      1.0, 0.0, 0.0, 0.0
   );
}
```

10.12.3.2 template<typename DATA_TYPE , unsigned SIZE> void gmtl::CubicCurve< DATA_TYPE, SIZE >::makeBspline ( )

Definition at line 376 of file ParametricCurve.h.

```
{
   mBasisMatrix.set(
      -1.0 / 6.0, 0.5, -0.5, 1.0 / 6.0,
      0.5, -1.0, 0.5, 0.0,
      -0.5, 0.0, 0.5, 0.0,
      1.0 / 6.0, 2.0 / 3.0, 1.0 / 6.0, 0.0
   );
}
```

10.12.3.3 template<typename DATA_TYPE , unsigned SIZE> void gmtl::CubicCurve< DATA_TYPE, SIZE >::makeCatmullRom ( )

Definition at line 354 of file ParametricCurve.h.

```
{
   mBasisMatrix.set(
      -0.5, 1.5, -1.5, 0.5,
      1.0, -2.5, 2.0, -0.5,
      -0.5, 0.0, 0.5, 0.0,
      0.0, 1.0, 0.0, 0.0
   );
}
```

10.12.3.4 template<typename DATA_TYPE , unsigned SIZE> void gmtl::CubicCurve< DATA_TYPE, SIZE >::makeHermite ( )

Definition at line 365 of file ParametricCurve.h.

```
{
   mBasisMatrix.set(
      2.0, -2.0, 1.0, 1.0,
      -3.0, 3.0, -2.0, -1.0,
      0.0, 0.0, 1.0, 0.0,
      1.0, 0.0, 0.0, 0.0
   );
}
```

10.12.3.5 template<typename DATA_TYPE , unsigned SIZE> CubicCurve< DATA_TYPE, SIZE > & gmtl::CubicCurve< DATA_TYPE, SIZE >::operator= (const CubicCurve< DATA_TYPE, SIZE > & *other*)

Definition at line 335 of file ParametricCurve.h.

```
{
   ParametricCurve::operator =(other);

   return *this;
}
```

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

- ParametricCurve.h

10.13 gmtl::meta::DefaultVecTag Struct Reference

`#include <VecBase.h>`

10.13.1 Detailed Description

Definition at line 23 of file VecBase.h.

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

- VecBase.h

10.14 gmtl::meta::DotVecUnrolled< ELT, T1, T2 > Struct Template Reference

meta class to unroll dot products.

`#include <VecOpsMeta.h>`

Static Public Member Functions

- static T1::DataType func (const T1 &v1, const T2 &v2)

10.14.1 Detailed Description

template<int ELT, typename T1, typename T2> struct gmtl::meta::DotVecUnrolled< ELT, T1, T2 >

meta class to unroll dot products.

Definition at line 22 of file VecOpsMeta.h.

10.14.2 Member Function Documentation

10.14.2.1 template<int ELT, typename T1 , typename T2 > static T1::DataType gmtl::meta::DotVecUnrolled< ELT, T1, T2 >::func (const T1 & *v1*, const T2 & *v2*) `[inline, static]`

Definition at line 24 of file VecOpsMeta.h.

```
{  return (v1[ELT]*v2[ELT]) + DotVecUnrolled<ELT-1,T1,T2>::func(v1,v2); }
```

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

- VecOpsMeta.h

10.15 gmtl::meta::DotVecUnrolled< 0, T1, T2 > Struct Template Reference

base cas for dot product unrolling.

`#include <VecOpsMeta.h>`

Static Public Member Functions

- static T1::DataType func (const T1 &v1, const T2 &v2)

10.15.1 Detailed Description

template<typename T1, typename T2> struct gmtl::meta::DotVecUnrolled< 0, T1, T2 >

base cas for dot product unrolling.

Definition at line 30 of file VecOpsMeta.h.

10.15.2 Member Function Documentation

10.15.2.1 template<typename T1 , typename T2 > static T1::DataType gmtl::meta::DotVecUnrolled< 0, T1, T2 >::func (const T1 & *v1*, const T2 & *v2*) `[inline, static]`

Definition at line 32 of file VecOpsMeta.h.

```
{   return (v1[0]*v2[0]); }
```

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

- VecOpsMeta.h

10.16 gmtl::Eigen Class Reference

```
#include <Eigen.h>
```

Public Member Functions

- 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 Member Functions

- static void Tridiagonal2 (float **aafMat, float *afDiag, float *afSubd)
- static void Tridiagonal3 (float **aafMat, float *afDiag, float *afSubd)
- static void Tridiagonal4 (float **aafMat, float *afDiag, float *afSubd)
- static void TridiagonalN (int iSize, float **aafMat, float *afDiag, float *afSubd)
- static bool QLAlgorithm (int iSize, float *afDiag, float *afSubd, float **aafMat)
- static void DecreasingSort (int iSize, float *afEigval, float **aafEigvec)
- static void IncreasingSort (int iSize, float *afEigval, float **aafEigvec)

Protected Attributes

- int m_iSize
- float ** m_aafMat
- float * m_afDiag
- float * m_afSubd

10.16.1 Detailed Description

Definition at line 17 of file Eigen.h.

10.16.2 Constructor & Destructor Documentation

10.16.2.1 gmtl::Eigen::Eigen (int *iSize*)

Definition at line 115 of file Eigen.h.

```
{
    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];
}
```

10.16.2.2 gmtl::Eigen::∼Eigen ( )

Definition at line 128 of file Eigen.h.

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

10.16.3 Member Function Documentation

10.16.3.1 void gmtl::Eigen::DecreasingSort (int *iSize*, float * *afEigval*, float ** *aafEigvec*) `[static, protected]`

Definition at line 522 of file Eigen.h.

```
{
    // 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;
            }
        }
    }
}
```

10.16.3.2 void gmtl::Eigen::DecrSortEigenStuff ( )

Definition at line 675 of file Eigen.h.

```
{
    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);
}
```

10.16.3.3 void gmtl::Eigen::DecrSortEigenStuff2 ( )

Definition at line 647 of file Eigen.h.

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

10.16.3.4 void gmtl::Eigen::DecrSortEigenStuff3 ( )

Definition at line 654 of file Eigen.h.

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

10.16.3.5 void gmtl::Eigen::DecrSortEigenStuff4 ( )

Definition at line 661 of file Eigen.h.

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

10.16.3.6 void gmtl::Eigen::DecrSortEigenStuffN ( )

Definition at line 668 of file Eigen.h.

```
{
    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);
}
```

10.16.3.7 void gmtl::Eigen::EigenStuff ( )

Definition at line 627 of file Eigen.h.

```
{
    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);
}
```

10.16.3.8 void gmtl::Eigen::EigenStuff2 ( )

Definition at line 603 of file Eigen.h.

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

10.16.3.9 void gmtl::Eigen::EigenStuff3 ( )

Definition at line 609 of file Eigen.h.

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

10.16.3.10 void gmtl::Eigen::EigenStuff4 ( )

Definition at line 615 of file Eigen.h.

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

10.16.3.11 void gmtl::Eigen::EigenStuffN ( )

Definition at line 621 of file Eigen.h.

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

10.16.3.12 float * gmtl::Eigen::GetEigenvalue ( ) `[inline]`

Definition at line 99 of file Eigen.h.

```
{
    return m_afDiag;
}
```

10.16.3.13 float gmtl::Eigen::GetEigenvalue (int *i*) const `[inline]`

Definition at line 89 of file Eigen.h.

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

10.16.3.14 float gmtl::Eigen::GetEigenvector (int *iRow*, int *iCol*) const `[inline]`

Definition at line 94 of file Eigen.h.

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

10.16.3.15 float ** gmtl::Eigen::GetEigenvector ( ) `[inline]`

Definition at line 104 of file Eigen.h.

```
{
    return m_aafMat;
}
```

10.16.3.16 void gmtl::Eigen::IncreasingSort (int *iSize*, float * *afEigval*, float ** *aafEigvec*) `[static, protected]`

Definition at line 558 of file Eigen.h.

```
{
    // 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;
            }
        }
    }
}
```

10.16.3.17 void gmtl::Eigen::IncrSortEigenStuff ( )

Definition at line 724 of file Eigen.h.

```
{
    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);
}
```

10.16.3.18 void gmtl::Eigen::IncrSortEigenStuff2 ( )

Definition at line 696 of file Eigen.h.

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

10.16.3.19 void gmtl::Eigen::IncrSortEigenStuff3 ( )

Definition at line 703 of file Eigen.h.

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

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

10.16.3.20 void gmtl::Eigen::IncrSortEigenStuff4 ( )

Definition at line 710 of file Eigen.h.

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

10.16.3.21 void gmtl::Eigen::IncrSortEigenStuffN ( )

Definition at line 717 of file Eigen.h.

```
{
    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);
}
```

10.16.3.22 float & gmtl::Eigen::Matrix (int *iRow*, int *iCol*) `[inline]`

Definition at line 84 of file Eigen.h.

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

10.16.3.23 bool gmtl::Eigen::QLAlgorithm (int *iSize*, float * *afDiag*, float * *afSubd*, float ** *aafMat*) `[static, protected]`

Definition at line 450 of file Eigen.h.

```
{
    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;
}
```

10.16.3.24 void gmtl::Eigen::SetMatrix (float ** *aafMat*)

Definition at line 594 of file Eigen.h.

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

10.16.3.25 void gmtl::Eigen::Tridiagonal2 (float ** *aafMat*, float * *afDiag*, float * *afSubd*) [static, protected]

Definition at line 137 of file Eigen.h.

```
{
    // 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;
}
```

10.16.3.26 void gmtl::Eigen::Tridiagonal3 (float ** *aafMat*, float * *afDiag*, float * *afSubd*) [static, protected]

Definition at line 151 of file Eigen.h.

```
{
    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;
    }
}
```

10.16.3.27 void gmtl::Eigen::Tridiagonal4 (float ** *aafMat*, float * *afDiag*, float * *afSubd*) `[static, protected]`

Definition at line 190 of file Eigen.h.

```
{
    // 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;
        }
    }
}
```

10.16.3.28 void gmtl::Eigen::TridiagonalN (int *iSize*, float ** *aafMat*, float * *afDiag*, float * *afSubd*) [static, protected]

Definition at line 357 of file Eigen.h.

```
{
    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;
}
```

10.16.4 Member Data Documentation

10.16.4.1 float** gmtl::Eigen::m_aafMat [protected]

Definition at line 56 of file Eigen.h.

10.16.4.2 float* gmtl::Eigen::m_afDiag [protected]

Definition at line 57 of file Eigen.h.

10.16.4.3 float* gmtl::Eigen::m_afSubd [protected]

Definition at line 58 of file Eigen.h.

10.16.4.4 int gmtl::Eigen::m_iSize [protected]

Definition at line 55 of file Eigen.h.

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

- Eigen.h

10.17 gmtl::meta::EqualVecUnrolled< ELT, VT > Struct Template Reference

meta class to test vector equality.

`#include <VecOpsMeta.h>`

Static Public Member Functions

- static bool func (const VT &v1, const VT &v2)

10.17.1 Detailed Description

template<int ELT, typename VT> struct gmtl::meta::EqualVecUnrolled< ELT, VT >

meta class to test vector equality.

Definition at line 54 of file VecOpsMeta.h.

10.17.2 Member Function Documentation

10.17.2.1 template<int ELT, typename VT > static bool gmtl::meta::EqualVecUnrolled< ELT, VT >::func (const VT & *v1*, const VT & *v2*) [inline, static]

Definition at line 56 of file VecOpsMeta.h.

```
{  return (v1[ELT]==v2[ELT]) && EqualVecUnrolled<ELT-1,VT>::func(v1,v2); }
```

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

- VecOpsMeta.h

10.18 gmtl::meta::EqualVecUnrolled< 0, VT > Struct Template Reference

base cas for dot product unrolling.

`#include <VecOpsMeta.h>`

Static Public Member Functions

- static bool func (const VT &v1, const VT &v2)

10.18.1 Detailed Description

template<typename VT> struct gmtl::meta::EqualVecUnrolled< 0, VT >

base cas for dot product unrolling.

Definition at line 62 of file VecOpsMeta.h.

10.18.2 Member Function Documentation

10.18.2.1 template<typename VT > static bool gmtl::meta::EqualVecUnrolled< 0, VT >::func (const VT & *v1*, const VT & *v2*) `[inline, static]`

Definition at line 64 of file VecOpsMeta.h.

```
{   return (v1[0]==v2[0]); }
```

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

- VecOpsMeta.h

10.19 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 Params { Size = 3, Order = ROTATION_ORDER::ID }
- typedef DATA_TYPE DataType

    Use this to declare single value types of the same type as this object.

Public Member Functions

- 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).

10.19.1 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 RotationOrder 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

Definition at line 38 of file EulerAngle.h.

10.19.2 Member Typedef Documentation

10.19.2.1 template<typename DATA_TYPE, typename ROTATION_ORDER> typedef DATA_TYPE gmtl::EulerAngle< DATA_TYPE, ROTATION_ORDER >::DataType

Use this to declare single value types of the same type as this object.

Definition at line 42 of file EulerAngle.h.

10.19.3 Member Enumeration Documentation

10.19.3.1 template<typename DATA_TYPE, typename ROTATION_ORDER> enum gmtl::EulerAngle::Params

Enumerator:

Size

Order

Definition at line 44 of file EulerAngle.h.

```
{ Size = 3, Order = ROTATION_ORDER::ID };
```

10.19.4 Constructor & Destructor Documentation

10.19.4.1 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 47 of file EulerAngle.h.

```
{
   gmtlASSERT( ROTATION_ORDER::IS_ROTORDER == 1 &&
```

```
          "you must specify a RotationOrder derived type for the rotationorder
      in euler angle." );
      mData[0] = DATA_TYPE( 0 );
      mData[1] = DATA_TYPE( 0 );
      mData[2] = DATA_TYPE( 0 );
   }
```

10.19.4.2 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 57 of file EulerAngle.h.

```
   {
      mData[0] = e.mData[0];
      mData[1] = e.mData[1];
      mData[2] = e.mData[2];
   }
```

10.19.4.3 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 65 of file EulerAngle.h.

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

10.19.5 Member Function Documentation

10.19.5.1 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 103 of file EulerAngle.h.

```
{ return mData; }
```

10.19.5.2 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 108 of file EulerAngle.h.

```
{ return mData; }
```

10.19.5.3 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 87 of file EulerAngle.h.

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

10.19.5.4 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 92 of file EulerAngle.h.

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

10.19.5.5 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 73 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

10.20 gmtl::meta::ExprTraits< T > Struct Template Reference

Traits class for expression template parameters.

`#include <VecExprMeta.h>`

Public Types

- typedef const T ExprRef

10.20.1 Detailed Description

template<typename T> struct gmtl::meta::ExprTraits< T >

Traits class for expression template parameters. NOTE: These types are VERY important to the performance of the code. They allow the compiler to optimize (ie. eliminate) much code.

Definition at line 59 of file VecExprMeta.h.

10.20.2 Member Typedef Documentation

10.20.2.1 template<typename T> typedef const T gmtl::meta::ExprTraits< T >::ExprRef

Definition at line 61 of file VecExprMeta.h.

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

- VecExprMeta.h

10.21 gmtl::meta::ExprTraits< VecBase< T, SIZE, DefaultVecTag > > Struct Template Reference

`#include <VecExprMeta.h>`

Public Types

- typedef const VecBase< T, SIZE, DefaultVecTag > & ExprRef

10.21.1 Detailed Description

template<typename T, unsigned SIZE> struct gmtl::meta::ExprTraits< VecBase< T, SIZE, DefaultVecTag > >

Definition at line 71 of file VecExprMeta.h.

10.21.2 Member Typedef Documentation

10.21.2.1 template<typename T , unsigned SIZE> typedef const VecBase<T,SIZE,DefaultVecTag>& gmtl::meta::ExprTraits< VecBase< T, SIZE, DefaultVecTag > >::ExprRef

Definition at line 73 of file VecExprMeta.h.

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

- VecExprMeta.h

10.22 gmtl::meta::ExprTraits< VecBase< T, SIZE, ScalarArg< T > > > Struct Template Reference

```
#include <VecExprMeta.h>
```

Public Types

- typedef const VecBase< T, SIZE, ScalarArg< T > > ExprRef

10.22.1 Detailed Description

template<typename T, unsigned SIZE> struct gmtl::meta::ExprTraits< VecBase< T, SIZE, ScalarArg< T > > >

Definition at line 65 of file VecExprMeta.h.

10.22.2 Member Typedef Documentation

10.22.2.1 template<typename T , unsigned SIZE> typedef const VecBase<T,SIZE,ScalarArg<T> > gmtl::meta::ExprTraits< VecBase< T, SIZE, ScalarArg< T > > >::ExprRef

Definition at line 67 of file VecExprMeta.h.

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

- VecExprMeta.h

10.23 gmtl::Frustum< DATA_TYPE > Class Template Reference

This class defines a View Frustum Volume as a set of 6 planes.

`#include <Frustum.h>`

Public Types

- enum PlaneNames {
  PLANE_LEFT = 0, PLANE_RIGHT = 1, PLANE_BOTTOM = 2, PLANE_-TOP = 3,
  PLANE_NEAR = 4, PLANE_FAR = 5 }

  An enum to name the plane indicies.
- typedef DATA_TYPE DataType
- typedef Frustum< DATA_TYPE > FrustumType

Public Member Functions

- Frustum ()

  Constructs a new frustum with all planes in default state.
- Frustum (const gmtl::Matrix< DATA_TYPE, 4, 4 > &projMatrix)

  Constructs a new frustum with the given projection matrix.
- Frustum (const gmtl::Matrix< DATA_TYPE, 4, 4 > &modelviewMatrix, const gmtl::Matrix< DATA_TYPE, 4, 4 > &projMatrix)

  Constructs a new frustum with given projection and modelview matricies.
- void extractPlanes (const gmtl::Matrix< DATA_TYPE, 4, 4 > &modelviewMatrix, const gmtl::Matrix< DATA_TYPE, 4, 4 > &projMatrix)

  Extracts the planes from the given projection matrix.
- void extractPlanes (const gmtl::Matrix< DATA_TYPE, 4, 4 > &projMatrix)

  Extracts the planes from the given projection and modelview matricies.

Public Attributes

- gmtl::Plane< DATA_TYPE > mPlanes [6]

10.23.1 Detailed Description

template<typename DATA_TYPE> class gmtl::Frustum< DATA_TYPE >

This class defines a View Frustum Volume as a set of 6 planes.

Definition at line 23 of file Frustum.h.

10.23.2 Member Typedef Documentation

10.23.2.1 template<typename DATA_TYPE> typedef DATA_TYPE gmtl::Frustum< DATA_TYPE >::DataType

Definition at line 26 of file Frustum.h.

10.23.2.2 template<typename DATA_TYPE> typedef Frustum<DATA_TYPE> gmtl::Frustum< DATA_TYPE >::FrustumType

Definition at line 27 of file Frustum.h.

10.23.3 Member Enumeration Documentation

10.23.3.1 template<typename DATA_TYPE> enum gmtl::Frustum::PlaneNames

An enum to name the plane indicies.

To have you not must remember those numbers.

Enumerator:

PLANE_LEFT left clipping plane equals 0

PLANE_RIGHT right clipping plane equals 1

PLANE_BOTTOM bottom clipping plane equals 2

PLANE_TOP top clipping plane equals 3

PLANE_NEAR near clipping plane equals 4

PLANE_FAR far clipping plane equals 5

Definition at line 33 of file Frustum.h.

```
{
   PLANE_LEFT = 0,
```

```
    PLANE_RIGHT = 1,
    PLANE_BOTTOM = 2,
    PLANE_TOP = 3,
    PLANE_NEAR = 4,
    PLANE_FAR = 5
};
```

10.23.4 Constructor & Destructor Documentation

10.23.4.1 template<typename DATA_TYPE> gmtl::Frustum< DATA_TYPE >::Frustum ( ) `[inline]`

Constructs a new frustum with all planes in default state.

Definition at line 46 of file Frustum.h.

```
{

}
```

10.23.4.2 template<typename DATA_TYPE> gmtl::Frustum< DATA_TYPE >::Frustum (const gmtl::Matrix< DATA_TYPE, 4, 4 > & *projMatrix*) `[inline]`

Constructs a new frustum with the given projection matrix.

Parameters

projMatrix The projection matrix of your camera or light etc. to construct the planes from.

Definition at line 57 of file Frustum.h.

```
{
    extractPlanes(projMatrix);
}
```

10.23.4.3 template<typename DATA_TYPE> gmtl::Frustum< DATA_TYPE >::Frustum (const gmtl::Matrix< DATA_TYPE, 4, 4 > & *modelviewMatrix*, const gmtl::Matrix< DATA_TYPE, 4, 4 > & *projMatrix*) `[inline]`

Constructs a new frustum with given projection and modelview matricies.

the matricies are multiplied in this order:

```
M = projMatrix * modelviewMatrix
```

The planes are then extracted from M.

Parameters

modelviewMatrix The modelview matrix of you camera or light etc. to construct the planes from.

projMatrix The projection matrix of your camera or light or whatever.

Definition at line 77 of file Frustum.h.

```
{
   extractPlanes(modelviewMatrix, projMatrix);
}
```

10.23.5 Member Function Documentation

10.23.5.1 template<typename DATA_TYPE> void gmtl::Frustum< DATA_TYPE >::extractPlanes (const gmtl::Matrix< DATA_TYPE, 4, 4 > & *modelviewMatrix*, const gmtl::Matrix< DATA_TYPE, 4, 4 > & *projMatrix*) `[inline]`

Extracts the planes from the given projection matrix.

Parameters

projMatrix The projection matrix of you camera or light or what ever.

Definition at line 89 of file Frustum.h.

```
{
   extractPlanes(projMatrix * modelviewMatrix);
}
```

10.23.5.2 template<typename DATA_TYPE> void gmtl::Frustum< DATA_TYPE >::extractPlanes (const gmtl::Matrix< DATA_TYPE, 4, 4 > & *projMatrix*) `[inline]`

Extracts the planes from the given projection and modelview matricies.

The matricies are multiplied in this order:

```
M = projMatrix * modelviewMatrix
```

The planes are then extracted from M.

Parameters

modelviewMatrix The modelview matrix of you camera or light etc. to construct the planes from.

projMatrix The projection matrix of you camera or light etc. to construct the planes from.

Definition at line 110 of file Frustum.h.

```
{
   const gmtl::Matrix<DATA_TYPE, 4, 4>& m = projMatrix;

   //left
   mPlanes[PLANE_LEFT].setNormal(gmtl::Vec<DATA_TYPE, 3>(m[3][0] + m[0][0],
                                                         m[3][1] + m[0][1],
                                                         m[3][2] + m[0][2]));
   mPlanes[PLANE_LEFT].setOffset(m[3][3] + m[0][3]);
   //right
   mPlanes[PLANE_RIGHT].setNormal(gmtl::Vec<DATA_TYPE, 3>(m[3][0] - m[0][0],
                                                          m[3][1] - m[0][1],
                                                          m[3][2] - m[0][2]));

   mPlanes[PLANE_RIGHT].setOffset(m[3][3] - m[0][3]);
   //bottom
   mPlanes[PLANE_BOTTOM].setNormal(gmtl::Vec<DATA_TYPE, 3>(m[3][0] + m[1][0],
                                                           m[3][1] + m[1][1],
                                                           m[3][2] + m[1][2]))
   ;
   mPlanes[PLANE_BOTTOM].setOffset(m[3][3] + m[1][3]);
   //top
   mPlanes[PLANE_TOP].setNormal(gmtl::Vec<DATA_TYPE, 3>(m[3][0] - m[1][0],
                                                        m[3][1] - m[1][1],
                                                        m[3][2] - m[1][2]));
   mPlanes[PLANE_TOP].setOffset(m[3][3] - m[1][3]);
   //near
   mPlanes[PLANE_NEAR].setNormal(gmtl::Vec<DATA_TYPE, 3>(m[3][0] + m[2][0],
                                                         m[3][1] + m[2][1],
                                                         m[3][2] + m[2][2]));
   mPlanes[PLANE_NEAR].setOffset(m[2][3] + m[3][3]);
   //far
   mPlanes[PLANE_FAR].setNormal(gmtl::Vec<DATA_TYPE, 3>(m[3][0] - m[2][0],
                                                        m[3][1] - m[2][1],
                                                        m[3][2] - m[2][2]));
   mPlanes[PLANE_FAR].setOffset(m[3][3] - m[2][3]);
}
```

10.23.6 Member Data Documentation

10.23.6.1 template<typename DATA_TYPE> gmtl::Plane<DATA_TYPE> gmtl::Frustum< DATA_TYPE >::mPlanes[6]

Definition at line 146 of file Frustum.h.

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

- Frustum.h

10.24 gmtl::meta::LenSqrVecUnrolled< ELT, T > Struct Template Reference

meta class to unroll length squared operation.

`#include <VecOpsMeta.h>`

Static Public Member Functions

- static T::DataType func (const T &v)

10.24.1 Detailed Description

template<int ELT, typename T> struct gmtl::meta::LenSqrVecUnrolled< ELT, T >

meta class to unroll length squared operation.

Definition at line 38 of file VecOpsMeta.h.

10.24.2 Member Function Documentation

10.24.2.1 template<int ELT, typename T > static T::DataType gmtl::meta::LenSqrVecUnrolled< ELT, T >::func (const T & *v*) `[inline, static]`

Definition at line 40 of file VecOpsMeta.h.

```
{  return (v[ELT]*v[ELT]) + LenSqrVecUnrolled<ELT-1,T>::func(v); }
```

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

- VecOpsMeta.h

10.25 gmtl::meta::LenSqrVecUnrolled< 0, T > Struct Template Reference

base cas for dot product unrolling.

```
#include <VecOpsMeta.h>
```

Static Public Member Functions

- static T::DataType func (const T &v)

10.25.1 Detailed Description

template<typename T> struct gmtl::meta::LenSqrVecUnrolled< 0, T >

base cas for dot product unrolling.

Definition at line 46 of file VecOpsMeta.h.

10.25.2 Member Function Documentation

10.25.2.1 template<typename T > static T::DataType gmtl::meta::LenSqrVecUnrolled< 0, T >::func (const T & *v*) [inline, static]

Definition at line 48 of file VecOpsMeta.h.

```
{   return (v[0]*v[0]); }
```

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

- VecOpsMeta.h

10.26 gmtl::LinearCurve< DATA_TYPE, SIZE > Class Template Reference

A representation of a line with order set to 2.

`#include <ParametricCurve.h>`

Inheritance diagram for gmtl::LinearCurve< DATA_TYPE, SIZE >:

Collaboration diagram for gmtl::LinearCurve< DATA_TYPE, SIZE >:

Public Member Functions

- LinearCurve ()
- LinearCurve (const LinearCurve &other)
- ~LinearCurve ()
- LinearCurve & operator= (const LinearCurve &other)
- void makeLerp ()

10.26.1 Detailed Description

template<typename DATA_TYPE, unsigned int SIZE> class gmtl::LinearCurve< DATA_TYPE, SIZE >

A representation of a line with order set to 2.

Template Parameters

DATA_TYPE The data type to use for the components.

SIZE The number of components this curve has.

Definition at line 198 of file ParametricCurve.h.

10.26.2 Constructor & Destructor Documentation

10.26.2.1 template<typename DATA_TYPE , unsigned int SIZE> gmtl::LinearCurve< DATA_TYPE, SIZE >::LinearCurve ( )

Definition at line 210 of file ParametricCurve.h.

```
{
}
```

10.26.2.2 template<typename DATA_TYPE , unsigned int SIZE> gmtl::LinearCurve< DATA_TYPE, SIZE >::LinearCurve (const LinearCurve< DATA_TYPE, SIZE > & *other*)

Definition at line 215 of file ParametricCurve.h.

```
{
   *this = other;
}
```

10.26.2.3 template<typename DATA_TYPE , unsigned int SIZE> gmtl::LinearCurve< DATA_TYPE, SIZE >::~LinearCurve ( )

Definition at line 221 of file ParametricCurve.h.

```
{
}
```

10.26.3 Member Function Documentation

10.26.3.1 template<typename DATA_TYPE , unsigned int SIZE> void gmtl::LinearCurve< DATA_TYPE, SIZE >::makeLerp ( )

Definition at line 235 of file ParametricCurve.h.

```
{
   mBasisMatrix.set(
      -1.0, 1.0,
      1.0, 0.0
   );
}
```

10.26.3.2 template<typename DATA_TYPE , unsigned int SIZE> LinearCurve< DATA_TYPE, SIZE > & gmtl::LinearCurve< DATA_TYPE, SIZE >::operator= (const LinearCurve< DATA_TYPE, SIZE > & *other*)

Definition at line 227 of file ParametricCurve.h.

```
{
   ParametricCurve::operator =(other);

   return *this;
}
```

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

- ParametricCurve.h

10.27 gmtl::LineSeg< DATA_TYPE > Class Template Reference

Describes a line segment.

`#include <LineSeg.h>`

Inheritance diagram for gmtl::LineSeg< DATA_TYPE >:

Collaboration diagram for gmtl::LineSeg< DATA_TYPE >:

Public Member Functions

- 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 LineSeg &ray)

  Constructs an exact duplicate of the given line segment.

- LineSeg (const Point< DATA_TYPE, 3 > &beg, const Point< DATA_TYPE, 3 > &end)

  Constructs a line segment with the given beginning and ending points.

- DATA_TYPE getLength () const

  Gets the length of this line segment.

10.27.1 Detailed Description

template<typename 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 28 of file LineSeg.h.

10.27.2 Constructor & Destructor Documentation

10.27.2.1 template<typename DATA_TYPE> gmtl::LineSeg< DATA_TYPE >::LineSeg ( ) `[inline]`

Constructs a line segment at the origin with a zero vector.

Definition at line 34 of file LineSeg.h.

```
{}
```

10.27.2.2 template<typename 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 44 of file LineSeg.h.

```
   : Ray<DATA_TYPE>( origin, dir )
{}
```

10.27.2.3 template<typename DATA_TYPE> gmtl::LineSeg< DATA_TYPE >::LineSeg (const LineSeg< DATA_TYPE > & *ray*) `[inline]`

Constructs an exact duplicate of the given line segment.

Parameters

ray the line segment to copy

Definition at line 53 of file LineSeg.h.

```
                                 : Ray<DATA_TYPE>( ray )
{
}
```

10.27.2.4 template<typename 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 63 of file LineSeg.h.

```
    : Ray<DATA_TYPE>()
{
   this->mOrigin = beg;
   this->mDir = end - beg;
}
```

10.27.3 Member Function Documentation

10.27.3.1 template<typename DATA_TYPE> DATA_TYPE gmtl::LineSeg< DATA_TYPE >::getLength ( ) const [inline]

Gets the length of this line segment.

Returns

the length of the line segment

Definition at line 74 of file LineSeg.h.

```
{
   return length(this->mDir);
}
```

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

- LineSeg.h

10.28 gmtl::Matrix< DATA_TYPE, ROWS, COLS > Class Template Reference

State tracked NxM dimensional Matrix (ordered in memory by Column).

`#include <Matrix.h>`

Collaboration diagram for gmtl::Matrix< DATA_TYPE, ROWS, COLS >:

Classes

- class ConstRowAccessor

  Helper class for Matrix op[] const.

- class RowAccessor

  Helper class for Matrix op[].

Public Types

- enum Params { Rows = ROWS, Cols = COLS }
- enum XformState {
  IDENTITY = 1, TRANS = 2, ORTHOGONAL = 4, AFFINE = 16,
  NON_UNISCALE = 32, FULL = 64, XFORM_ERROR = 128 }

  describes the xforms that this matrix has been through.

- typedef DATA_TYPE DataType

  use this to declare single value types of the same type as this matrix.

Public Member Functions

- 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)

  comma operator

- 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 WARNING: If you set data in the matrix (using this interface), you are required to set mState appropriately, failure to do so will result in incorrect calculations by other functions in GMTL.

- const DATA_TYPE & operator() (const unsigned row, const unsigned column) const

  access [row, col] in the matrix (const version)

- RowAccessor operator[ ] (const unsigned row)

  bracket operator WARNING: If you set data in the matrix (using this interface), you are required to set mState appropriately, failure to do so will result in incorrect calculations by other functions in GMTL.

- ConstRowAccessor operator[ ] (const unsigned row) const

  bracket operator (const version)

- const DATA_TYPE ∗ getData () const

  Gets a DATA_TYPE pointer to the matrix data.

- bool isError ()
- void setError ()
- void setState (int state)

Public Attributes

- DATA_TYPE mData [COLS *ROWS]

  Column major.

- int mState

  describes what xforms are in this matrix

10.28.1 Detailed Description

template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> class gmtl::Matrix< DATA_TYPE, ROWS, COLS >

State tracked NxM dimensional Matrix (ordered in memory by Column). **Memory mapping:**

gmtl::Matrix stores its elements in column major order. That is, it stores each column end-to-end in memory.

Typically, for 3D transform matrices, the 3x3 rotation is in the first three columns, while the translation is in the last column.

This memory alignment is chosen for compatibility with the OpenGL graphics API and others, which take matrices in this specific column major ordering described above.

See the interfaces for operator[r][c] and operator(r,c) for how to iterate over columns and rows for a GMTL Matrix.

NOTES on Matrix memory layout and [][] accessors:

- gmtl Matrix memory is "column major" ordered, where columns are end to end in memory, while a C/C++ Matrix accessed the same way (using operator[][]) as a gmtl Matrix is "row major" ordered.
- As a result, a gmtl matrix stores elements in memory transposed from the equivelent matrix defined using an array in the C/C++ language, assuming they are accessed the same way (see example).
  - Illustrative Example:
    Given two flavors of matrix, C/C++, and gmtl:
    float cmat[n][m]; and gmtl::Matrix<float, n, m> mat;

Writing values into each, while accessing them the same:

cmat[row][col] = mat[row][col] = some_values[x];

Then reading values from the matrix array:

((float∗)cmat) and mat.getData()

Will yield pointers to memory containing matrices that are the transpose of each other.

- In practice, the differences between GMTL and C/C++ defined matrices all depends how you iterate over your matrix.

  If gmtl is accessed mat[row][col] and C/C++ is accessed mat[col][row], then memory-wise, these two will yield the same memory mapping (column major as described above), thus, are equivelent and can both be used interchangably in many popular graphics APIs such as OpenGL, DirectX, and others.

- In C/C++ access of a matrix via mat[row][col] yields this memory mapping after using ((float∗)mat) to return it:

```
    (0,0) (0,1) (0,2) (0,3)   <=== Contiguous memory arranged by row
    (1,0) (1,1) (1,2) (1,3)   <=== Contiguous
    (2,0) (2,1) (2,2) (2,3)   <=== Contiguous
    (3,0) (3,1) (3,2) (3,3)   <=== Contiguous

  or linearly if you prefer:
    (0,0) (0,1) (0,2) (0,3) (1,0) (1,1) (1,2) (1,3) (2,0) (2,1) (2,2) (2,3) (3,0) (3,1) (
```

- In gmtl, access of a matrix via mat[row][col] yields this memory mapping after using getData() to return it:

```
    (0,0) (0,1) (0,2) (0,3)
    (1,0) (1,1) (1,2) (1,3)
    (2,0) (2,1) (2,2) (2,3)
    (3,0) (3,1) (3,2) (3,3)
      ^     ^     ^     ^
    --1-----2-----3-----4---- Contiguous memory arranged by column

  or linearly if you prefer:
    (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) (
```

State Tracking:

The idea of a state-tracked matrix is that if we track the information as it is stored into the matrix, then other operations could make more optimal descisions based on the known state. A good example is in matrix invertion, a reletively costly operation for matrices. However, if we know the matrix state is (i.e.) ORTHOGONAL, then inversion becomes a simple transpose operation. There are also optimizations with multiplication, as well as other.

One side effect of this state tracking is that EVERY MATRIC FUNCTION NEEDS TO TRACK STATE. This means that anyone writing custom methods, or extentions to gmtl, will need to pay close attention to matrix state.

To facilitate state tracking in extensions, we've provided the function gmtl::combineMatrixStates() to help in determining state based on two combined matrices.

See also

Matrix44f
Matrix44d

Definition at line 107 of file Matrix.h.

10.28.2 Member Typedef Documentation

10.28.2.1 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 121 of file Matrix.h.

10.28.3 Member Enumeration Documentation

10.28.3.1 template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> enum gmtl::Matrix::Params

Enumerator:

Rows

Cols

Definition at line 122 of file Matrix.h.

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

10.28.3.2 template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> enum gmtl::Matrix::XformState

describes the xforms that this matrix has been through.

Enumerator:

- ***IDENTITY***
- ***TRANS***
- ***ORTHOGONAL***
- ***AFFINE***
- ***NON_UNISCALE***
- ***FULL***
- ***XFORM_ERROR***

Definition at line 181 of file Matrix.h.

```
{
   // identity matrix.
   IDENTITY = 1,

   // only translation, can simply negate that column
   TRANS = 2,

   // able to tranpose to get the inverse.  only rotation component is set
   ORTHOGONAL = 4,

   // orthogonal, and normalized axes.
   //ORTHONORMAL = 8,

   // leaves the homogeneous coordinate unchanged - that is, in which the last
    column is (0,0,0,s).
   // can include rotation, uniform scale, and translation, but no shearing or
    nonuniform scaling
   // This can optionally be combined with the NON_UNISCALE state to indicate
   there is also non-uniform scale
   AFFINE = 16,

   // AFFINE matrix with non-uniform scale, a matrix cannot
   // have this state without also having AFFINE (must be or'd together).
   NON_UNISCALE = 32,

   // fully set matrix containing more information than the above, or state is
    unknown,
   // or unclassifiable in terms of the above.
   FULL = 64,

   // error bit
   XFORM_ERROR = 128
};
```

10.28.4 Constructor & Destructor Documentation

10.28.4.1 template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> gmtl::Matrix< DATA_TYPE, ROWS, COLS >::Matrix ( ) [inline]

Default Constructor (Identity constructor).

Todo

mp

Todo

mp

Todo

Set initial state to IDENTITY and test other stuff

Definition at line 213 of file Matrix.h.

```
{
   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 = IDENTITY;
}
```

10.28.4.2 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 231 of file Matrix.h.

```
{
   this->set( matrix.getData() );
   mState = matrix.mState;
}
```

10.28.5 Member Function Documentation

10.28.5.1 template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> const DATA_TYPE* gmtl::Matrix< DATA_TYPE, ROWS, COLS >::getData ( ) const [inline]

Gets a DATA_TYPE pointer to the matrix data.

Returns

Returns a pointer to the head of the matrix data.

Definition at line 461 of file Matrix.h.

```
{ return (DATA_TYPE*)mData; }
```

10.28.5.2 template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> bool gmtl::Matrix< DATA_TYPE, ROWS, COLS >::isError ( ) [inline]

Definition at line 463 of file Matrix.h.

```
{
   return mState & XFORM_ERROR;
}
```

10.28.5.3 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 424 of file Matrix.h.

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

10.28.5.4 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 WARNING: If you set data in the matrix (using this interface), you are required to set mState appropriately, failure to do so will result in incorrect calculations by other functions in GMTL.

If you are unsure about how to set mState, set it to FULL and you will be sure to get the correct result at the cost of some performance.

Definition at line 417 of file Matrix.h.

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

10.28.5.5 template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> RowAccessor gmtl::Matrix< DATA_TYPE, ROWS, COLS >::operator[] (const unsigned *row*) `[inline]`

bracket operator WARNING: If you set data in the matrix (using this interface), you are required to set mState appropriately, failure to do so will result in incorrect calculations by other functions in GMTL.

If you are unsure about how to set mState, set it to FULL and you will be sure to get the correct result at the cost of some performance.

Definition at line 438 of file Matrix.h.

```
{
   return RowAccessor(this, row);
}
```

10.28.5.6 template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> ConstRowAccessor gmtl::Matrix< DATA_TYPE, ROWS, COLS >::operator[] (const unsigned *row*) const `[inline]`

bracket operator (const version)

Definition at line 444 of file Matrix.h.

```
{
   return ConstRowAccessor( this, row );
}
```

10.28.5.7 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.

Note

variable names specify the row,column number to put the data into

Todo

needs mp!!

Definition at line 241 of file Matrix.h.

```
{
   GMTL_STATIC_ASSERT( (ROWS == 2 && COLS == 2), Set_called_when_Matrix_not_of
   _size_2_2 );
   mData[0] = v00;
   mData[1] = v10;
   mData[2] = v01;
   mData[3] = v11;
   mState = FULL;
}
```

10.28.5.8 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*) `[inline]`

element wise setter for 3x4.

Todo

needs mp!! currently no way for a 4x3, ...

Definition at line 293 of file Matrix.h.

```
{
   GMTL_STATIC_ASSERT( (ROWS == 3 && COLS == 4), Set_called_when_Matrix_not_of
   _size_3_4 );
   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;
}
```

10.28.5.9 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 *v10*, DATA_TYPE *v11*, DATA_TYPE *v12*) `[inline]`

element wise setter for 2x3.

Todo

needs mp!!

Definition at line 255 of file Matrix.h.

```
{
    GMTL_STATIC_ASSERT( (ROWS == 2 && COLS == 3), Set_called_when_Matrix_not_of
    _size_2_3 );
    mData[0] = v00;
    mData[1] = v10;
    mData[2] = v01;
    mData[3] = v11;
    mData[4] = v02;
    mData[5] = v12;
    mState = FULL;
}
```

10.28.5.10 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 *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 318 of file Matrix.h.

```
{
   GMTL_STATIC_ASSERT( (ROWS == 4 && COLS == 4), Set_called_when_Matrix_not_of
   _size_4_4 );
   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;
}
```

10.28.5.11 template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> void gmtl::Matrix< DATA_TYPE, ROWS, COLS >::set (const DATA_TYPE ∗ *data*) `[inline]`

comma operator

Todo

implement this!

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...

Todo

mp

Definition at line 370 of file Matrix.h.

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

10.28.5.12 template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> void gmtl::Matrix< DATA_TYPE, ROWS, COLS >::set (DATA_TYPE *v00*, DATA_TYPE *v01*, DATA_TYPE *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 271 of file Matrix.h.

```
{
   GMTL_STATIC_ASSERT( (ROWS == 3 && COLS == 3), Set_called_when_Matrix_not_of
   _size_3_3 );
   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;
}
```

10.28.5.13 template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> void gmtl::Matrix< DATA_TYPE, ROWS, COLS >::setError ( ) [inline]

Definition at line 467 of file Matrix.h.

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

10.28.5.14 template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> void gmtl::Matrix< DATA_TYPE, ROWS, COLS >::setState (int *state*) [inline]

Definition at line 472 of file Matrix.h.

```
{ mState = state; }
```

10.28.5.15 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...

Todo

metaprog

Definition at line 400 of file Matrix.h.

```
{
   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;
}
```

10.28.6 Member Data Documentation

10.28.6.1 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] WARNING: If you set data in the matrix (using this interface), you are required to set mState appropriately, failure to do so will result in incorrect calculations by other functions in GMTL. If you are unsure about how to set mState, set it to FULL and you will be sure to get the correct result at the cost of some performance.

Definition at line 485 of file Matrix.h.

10.28.6.2 template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> int gmtl::Matrix< DATA_TYPE, ROWS, COLS >::mState

describes what xforms are in this matrix

Definition at line 488 of file Matrix.h.

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

- Matrix.h

10.29 gmtl::OOBox Class Reference

`#include <OOBox.h>`

Public Member Functions

- 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]

10.29.1 Detailed Description

Definition at line 19 of file OOBox.h.

10.29.2 Constructor & Destructor Documentation

10.29.2.1 gmtl::OOBox::OOBox ( ) [inline]

Definition at line 23 of file OOBox.h.

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

10.29.2.2 gmtl::OOBox::OOBox (OOBox & *box*) [inline]

Definition at line 77 of file OOBox.h.

```
{
   mCenter = box.mCenter;
   mAxis[0] = box.mAxis[0];
   mAxis[1] = box.mAxis[1];
   mAxis[2] = box.mAxis[2];
   mHalfLen[0] = box.mHalfLen[0];
   mHalfLen[1] = box.mHalfLen[1];
   mHalfLen[2] = box.mHalfLen[2];
}
```

10.29.3 Member Function Documentation

10.29.3.1 Vec3 * gmtl::OOBox::axes ( ) [inline]

Definition at line 109 of file OOBox.h.

```
{
   return mAxis;
}
```

10.29.3.2 const Vec3 * gmtl::OOBox::axes ( ) const [inline]

Definition at line 114 of file OOBox.h.

```
{
   return mAxis;
}
```

10.29.3.3 Vec3 & gmtl::OOBox::axis (int *i*) [inline]

Definition at line 99 of file OOBox.h.

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

10.29.3.4 const Vec3 & gmtl::OOBox::axis (int *i*) const [inline]

Definition at line 104 of file OOBox.h.

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

10.29.3.5 const Point3 & gmtl::OOBox::center ( ) const [inline]

Definition at line 94 of file OOBox.h.

```
{
   return mCenter;
}
```

10.29.3.6 Point3 & gmtl::OOBox::center ( ) [inline]

Definition at line 89 of file OOBox.h.

```
{
   return mCenter;
}
```

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

Definition at line 164 of file OOBox.h.

```
{
   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;
}
```

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

Definition at line 124 of file OOBox.h.

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

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

Definition at line 119 of file OOBox.h.

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

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

Definition at line 129 of file OOBox.h.

```
{
    return mHalfLen;
}
```

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

Definition at line 134 of file OOBox.h.

```
{
    return mHalfLen;
}
```

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

Definition at line 57 of file OOBox.h.

```
{
   mCenter = ZeroVec3;
   mAxis[0] = XUnitVec3;
   mAxis[1] = YUnitVec3;
   mAxis[2] = ZUnitVec3;
   mHalfLen[0] = mHalfLen[1] = mHalfLen[2] = 0.0f;
}
```

10.29.3.13 void gmtl::OOBox::mergeWith (const OOBox & *box*)

10.29.3.14 OOBox & gmtl::OOBox::operator= (const OOBox & *box*) [inline]

Definition at line 140 of file OOBox.h.

```
{
   mCenter = box.mCenter;
   mAxis[0] = box.mAxis[0];
   mAxis[1] = box.mAxis[1];
   mAxis[2] = box.mAxis[2];
   mHalfLen[0] = box.mHalfLen[0];
   mHalfLen[1] = box.mHalfLen[1];
   mHalfLen[2] = box.mHalfLen[2];
   return *this;
}
```

10.29.3.15 bool gmtl::OOBox::operator== (const OOBox & *box*) const [inline]

Definition at line 153 of file OOBox.h.

```
{
   return ((mCenter == box.mCenter) &&
           (mAxis[0] == box.mAxis[0]) &&
           (mAxis[1] == box.mAxis[1]) &&
           (mAxis[2] == box.mAxis[2]) &&
           (mHalfLen[0] == box.mHalfLen[0]) &&
           (mHalfLen[1] == box.mHalfLen[1]) &&
           (mHalfLen[2] == box.mHalfLen[2]));
}
```

10.29.4 Member Data Documentation

10.29.4.1 Vec3 gmtl::OOBox::mAxis[3]

Definition at line 68 of file OOBox.h.

10.29.4.2 Point3 gmtl::OOBox::mCenter

Definition at line 67 of file OOBox.h.

10.29.4.3 float gmtl::OOBox::mHalfLen[3]

Definition at line 69 of file OOBox.h.

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

- OOBox.h

10.30 gmtl::ParametricCurve< DATA_TYPE, SIZE, ORDER > Class Template Reference

A base representation of a parametric curve with SIZE component using DATA_TYPE as the data type, ORDER as the order for each component.

```
#include <ParametricCurve.h>
```

Collaboration diagram for gmtl::ParametricCurve< DATA_TYPE, SIZE, ORDER >:

Public Member Functions

- ParametricCurve ()
- ParametricCurve (const ParametricCurve &other)
- ~ParametricCurve ()
- ParametricCurve & operator= (const ParametricCurve &other)
- void setWeights (DATA_TYPE weights[ORDER])
- void setControlPoints (Vec< DATA_TYPE, SIZE > control_points[ORDER])
- void setBasisMatrix (const Matrix< DATA_TYPE, ORDER, ORDER > &basis_matrix)
- Vec< DATA_TYPE, SIZE > getInterpolatedValue (DATA_TYPE value) const
- Vec< DATA_TYPE, SIZE > getInterpolatedDerivative (DATA_TYPE value) const

Protected Attributes

- DATA_TYPE mWeights [ORDER]
- Vec< DATA_TYPE, SIZE > mControlPoints [ORDER]
- Matrix< DATA_TYPE, ORDER, ORDER > mBasisMatrix

10.30.1 Detailed Description

template<typename DATA_TYPE, unsigned SIZE, unsigned ORDER> class gmtl::ParametricCurve< DATA_TYPE, SIZE, ORDER >

A base representation of a parametric curve with SIZE component using DATA_TYPE as the data type, ORDER as the order for each component.

Template Parameters

DATA_TYPE The data type to use for the components.

SIZE The number of components this curve has.

ORDER The order of this curve.

Since

0.6.1

Definition at line 40 of file ParametricCurve.h.

10.30.2 Constructor & Destructor Documentation

10.30.2.1 template<typename DATA_TYPE , unsigned SIZE, unsigned ORDER> gmtl::ParametricCurve< DATA_TYPE, SIZE, ORDER >::ParametricCurve ( )

Definition at line 61 of file ParametricCurve.h.

```
{
   for (unsigned int i = 0; i < ORDER; ++i)
   {
      mWeights[i] = (DATA_TYPE)1.0;
   }
}
```

10.30.2.2 template<typename DATA_TYPE , unsigned SIZE, unsigned ORDER> gmtl::ParametricCurve< DATA_TYPE, SIZE, ORDER >::ParametricCurve (const ParametricCurve< DATA_TYPE, SIZE, ORDER > & *other*)

Definition at line 71 of file ParametricCurve.h.

```
{
   *this = other;
}
```

10.30.2.3 template<typename DATA_TYPE , unsigned SIZE, unsigned ORDER> gmtl::ParametricCurve< DATA_TYPE, SIZE, ORDER >::∼ParametricCurve ( )

Definition at line 77 of file ParametricCurve.h.

```
{
}
```

10.30.3 Member Function Documentation

10.30.3.1 template<typename DATA_TYPE, unsigned SIZE, unsigned ORDER> Vec< DATA_TYPE, SIZE > gmtl::ParametricCurve< DATA_TYPE, SIZE, ORDER >::getInterpolatedDerivative (DATA_TYPE *value*) const

Definition at line 155 of file ParametricCurve.h.

```
{
   Vec<DATA_TYPE, SIZE> ret_vec;
   DATA_TYPE power_vector[ORDER];
   DATA_TYPE exponent;
   DATA_TYPE coefficient[ORDER];

   for (unsigned int i = 0; i < ORDER; ++i)
   {
      exponent = static_cast<DATA_TYPE>(ORDER - i - 1);

      if (exponent > 0)
      {
         power_vector[i] = exponent * Math::pow(value, exponent - 1);
      }
      else
      {
         power_vector[i] = (DATA_TYPE)0.0;
      }
```

```
   }

   for (unsigned int column = 0; column < ORDER; ++column)
   {
      coefficient[column] = static_cast<DATA_TYPE>(0.0);

      for (unsigned int row = 0; row < ORDER; ++row)
      {
         coefficient[column] += power_vector[row] * mBasisMatrix[row][column];
      }

      ret_vec += coefficient[column] * mWeights[column] * mControlPoints[column];

   }

   return ret_vec;
}
```

10.30.3.2 template<typename DATA_TYPE, unsigned SIZE, unsigned ORDER> Vec< DATA_TYPE, SIZE > gmtl::ParametricCurve< DATA_TYPE, SIZE, ORDER >::getInterpolatedValue (DATA_TYPE *value*) const

Definition at line 125 of file ParametricCurve.h.

```
{
   Vec<DATA_TYPE, SIZE> ret_vec;
   DATA_TYPE power_vector[ORDER];
   DATA_TYPE exponent;
   DATA_TYPE coefficient[ORDER];

   for (unsigned int i = 0; i < ORDER; ++i)
   {
      exponent = (DATA_TYPE)(ORDER - i - 1);
      power_vector[i] = Math::pow(value, exponent);
   }

   for (unsigned int column = 0; column < ORDER; ++column)
   {
      coefficient[column] = (DATA_TYPE)0.0;

      for (unsigned int row = 0; row < ORDER; ++row)
      {
         coefficient[column] += power_vector[row] * mBasisMatrix[row][column];
      }

      ret_vec += coefficient[column] * mWeights[column] * mControlPoints[column];

   }

   return ret_vec;
}
```

10.30.3.3 template<typename DATA_TYPE , unsigned SIZE, unsigned ORDER> ParametricCurve< DATA_TYPE, SIZE, ORDER > & gmtl::ParametricCurve< DATA_TYPE, SIZE, ORDER >::operator= (const ParametricCurve< DATA_TYPE, SIZE, ORDER > & *other*)

Definition at line 83 of file ParametricCurve.h.

```
{
   for (unsigned int i = 0; i < ORDER; ++i)
   {
      mWeights[i] = other.mWeights[i];
      mControlPoints[i] = other.mControlPoints[i];
   }

   mBasisMatrix = other.mBasisMatrix;

   return *this;
}
```

10.30.3.4 template<typename DATA_TYPE, unsigned SIZE, unsigned ORDER> void gmtl::ParametricCurve< DATA_TYPE, SIZE, ORDER >::setBasisMatrix (const Matrix< DATA_TYPE, ORDER, ORDER > & *basis_matrix*)

Definition at line 118 of file ParametricCurve.h.

```
{
   mBasisMatrix = basis_matrix;
}
```

10.30.3.5 template<typename DATA_TYPE, unsigned SIZE, unsigned ORDER> void gmtl::ParametricCurve< DATA_TYPE, SIZE, ORDER >::setControlPoints (Vec< DATA_TYPE, SIZE > *control_points[ORDER]*)

10.30.3.6 template<typename DATA_TYPE, unsigned SIZE, unsigned ORDER> void gmtl::ParametricCurve< DATA_TYPE, SIZE, ORDER >::setWeights (DATA_TYPE *weights[ORDER]*)

Definition at line 98 of file ParametricCurve.h.

```
{
   for (unsigned int i = 0; i < ORDER; ++i)
   {
      mWeights[i] = weights[i];
   }
}
```

10.30.4 Member Data Documentation

10.30.4.1 template<typename DATA_TYPE, unsigned SIZE, unsigned ORDER> Matrix<DATA_TYPE, ORDER, ORDER> gmtl::ParametricCurve< DATA_TYPE, SIZE, ORDER >::mBasisMatrix `[protected]`

Definition at line 57 of file ParametricCurve.h.

10.30.4.2 template<typename DATA_TYPE, unsigned SIZE, unsigned ORDER> Vec<DATA_TYPE, SIZE> gmtl::ParametricCurve< DATA_TYPE, SIZE, ORDER >::mControlPoints[ORDER] `[protected]`

Definition at line 56 of file ParametricCurve.h.

10.30.4.3 template<typename DATA_TYPE, unsigned SIZE, unsigned ORDER> DATA_TYPE gmtl::ParametricCurve< DATA_TYPE, SIZE, ORDER >::mWeights[ORDER] `[protected]`

Definition at line 55 of file ParametricCurve.h.

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

- ParametricCurve.h

10.31 gmtl::Plane< DATA_TYPE > Class Template Reference

Plane: Defines a geometrical plane.

`#include <Plane.h>`

Collaboration diagram for gmtl::Plane< DATA_TYPE >:

Public Member Functions

- 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.

10.31.1 Detailed Description

template<class DATA_TYPE> class gmtl::Plane< DATA_TYPE >

Plane: Defines a geometrical plane. All points on the plane satisfy the equation 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 36 of file Plane.h.

10.31.2 Constructor & Destructor Documentation

10.31.2.1 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 43 of file Plane.h.

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

10.31.2.2 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 54 of file Plane.h.

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

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

    mOffset = dot( static_cast< Vec<DATA_TYPE, 3> >(pt1), mNorm );  // Graphics
     Gems I: Page 390
}
```

10.31.2.3 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 72 of file Plane.h.

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

10.31.2.4 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 84 of file Plane.h.

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

10.31.2.5 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 93 of file Plane.h.

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

10.31.3 Member Function Documentation

10.31.3.1 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 vector

Definition at line 102 of file Plane.h.

```
{
    return mNorm;
}
```

10.31.3.2 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 125 of file Plane.h.

```
{
    return mOffset;
}
```

10.31.3.3 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 normalized vector

Precondition

|norm| = 1

Definition at line 114 of file Plane.h.

```
{
   mNorm = norm;
}
```

10.31.3.4 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 135 of file Plane.h.

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

10.31.4 Member Data Documentation

10.31.4.1 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 146 of file Plane.h.

10.31.4.2 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 153 of file Plane.h.

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

- Plane.h

10.32 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< DATA_TYPE, SIZE >:

Collaboration diagram for gmtl::Point< DATA_TYPE, SIZE >:

Public Types

- enum Params { Size = SIZE }

    The number of components this VecB has.

- typedef DATA_TYPE DataType

    The datatype used for the components of this VecB.

- typedef VecBase< DATA_TYPE, SIZE > BaseType

    Placeholder for the base type.

- typedef Point< DATA_TYPE, SIZE > VecType

Public Member Functions

- Point ()

    Default constructor.

- template<typename REP2 >
  VecType & operator= (const VecBase< DATA_TYPE, SIZE, REP2 > &rhs)

Assign from different rep.

Value constructors

Construct with copy of rVec

- template<typename REP2 >
  Point (const VecBase< DATA_TYPE, SIZE, REP2 > &rVec)
- Point (const DATA_TYPE &val0, const DATA_TYPE &val1)

  Construct a 2-D point with 2 given values.

- Point (const DATA_TYPE &val0, const DATA_TYPE &val1, const DATA_-TYPE &val2)

  Construct a 3-D point with 2 given values.

- Point (const DATA_TYPE &val0, const DATA_TYPE &val1, const DATA_-TYPE &val2, const DATA_TYPE &val3)

  Construct a 4-D point with 2 given values.

10.32.1 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 30 of file Point.h.

10.32.2 Member Typedef Documentation

10.32.2.1 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 37 of file Point.h.

10.32.2.2 template<class DATA_TYPE, unsigned SIZE> typedef DATA_TYPE gmtl::Point< DATA_TYPE, SIZE >::DataType

The datatype used for the components of this VecB.

Reimplemented from gmtl::VecBase< DATA_TYPE, SIZE >.

Definition at line 33 of file Point.h.

10.32.2.3 template<class DATA_TYPE, unsigned SIZE> typedef Point<DATA_TYPE, SIZE> gmtl::Point< DATA_TYPE, SIZE >::VecType

Definition at line 38 of file Point.h.

10.32.3 Member Enumeration Documentation

10.32.3.1 template<class DATA_TYPE, unsigned SIZE> enum gmtl::Point::Params

The number of components this VecB has.

Enumerator:

Size

Reimplemented from gmtl::VecBase< DATA_TYPE, SIZE >.

Definition at line 34 of file Point.h.

```
{ Size = SIZE };
```

10.32.4 Constructor & Destructor Documentation

10.32.4.1 template<class DATA_TYPE, unsigned SIZE> gmtl::Point< DATA_TYPE, SIZE >::Point ( ) [inline]

Default constructor.

Definition at line 43 of file Point.h.

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

10.32.4.2 template<class DATA_TYPE, unsigned SIZE> template<typename REP2 > gmtl::Point< DATA_TYPE, SIZE >::Point (const VecBase< DATA_TYPE, SIZE, REP2 > & *rVec*) [inline]

Definition at line 65 of file Point.h.

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

10.32.4.3 template<class DATA_TYPE, unsigned SIZE> gmtl::Point< DATA_TYPE, SIZE >::Point (const DATA_TYPE & *val0*, const DATA_TYPE & *val1*) [inline]

Construct a 2-D point with 2 given values.

Definition at line 74 of file Point.h.

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

10.32.4.4 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]

Construct a 3-D point with 2 given values.

Definition at line 84 of file Point.h.

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

10.32.4.5 template<class DATA_TYPE, unsigned SIZE> gmtl::Point< DATA_TYPE, SIZE >::Point (const DATA_TYPE & *val0*, const DATA_TYPE & *val1*, const DATA_TYPE & *val2*, const DATA_TYPE & *val3*) `[inline]`

Construct a 4-D point with 2 given values.

Definition at line 94 of file Point.h.

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

10.32.5 Member Function Documentation

10.32.5.1 template<class DATA_TYPE, unsigned SIZE> template<typename REP2 > VecType& gmtl::Point< DATA_TYPE, SIZE >::operator= (const VecBase< DATA_TYPE, SIZE, REP2 > & *rhs*) `[inline]`

Assign from different rep.

Definition at line 111 of file Point.h.

```
{
   BaseType::operator=(rhs);
   return *this;
}
```

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

- Point.h

10.33 gmtl::QuadraticCurve< DATA_TYPE, SIZE > Class Template Reference

A representation of a quadratic curve with order set to 3.

```
#include <ParametricCurve.h>
```

Inheritance diagram for gmtl::QuadraticCurve< DATA_TYPE, SIZE >:

Collaboration diagram for gmtl::QuadraticCurve< DATA_TYPE, SIZE >:

Public Member Functions

- QuadraticCurve ()
- QuadraticCurve (const QuadraticCurve &other)
- ~QuadraticCurve ()
- QuadraticCurve & operator= (const QuadraticCurve &other)
- void makeBezier ()

10.33.1 Detailed Description

template<typename DATA_TYPE, unsigned SIZE> class gmtl::QuadraticCurve< DATA_TYPE, SIZE >

A representation of a quadratic curve with order set to 3.

Template Parameters

DATA_TYPE The data type to use for the components.

SIZE The number of components this curve has.

Definition at line 250 of file ParametricCurve.h.

10.33.2 Constructor & Destructor Documentation

10.33.2.1 template<typename DATA_TYPE , unsigned SIZE> gmtl::QuadraticCurve< DATA_TYPE, SIZE >::QuadraticCurve ( )

Definition at line 262 of file ParametricCurve.h.

```
{
}
```

10.33.2.2 template<typename DATA_TYPE , unsigned SIZE> gmtl::QuadraticCurve< DATA_TYPE, SIZE >::QuadraticCurve (const QuadraticCurve< DATA_TYPE, SIZE > & *other*)

Definition at line 267 of file ParametricCurve.h.

```
{
   *this = other;
}
```

10.33.2.3 template<typename DATA_TYPE , unsigned SIZE> gmtl::QuadraticCurve< DATA_TYPE, SIZE >::~QuadraticCurve ( )

Definition at line 273 of file ParametricCurve.h.

```
{
}
```

10.33.3 Member Function Documentation

10.33.3.1 template<typename DATA_TYPE , unsigned SIZE> void gmtl::QuadraticCurve< DATA_TYPE, SIZE >::makeBezier ( )

Definition at line 287 of file ParametricCurve.h.

```
{
   mBasisMatrix.set(
      1.0, -2.0, 1.0,
      -2.0, 2.0, 0.0,
      1.0, 0.0, 0.0
   );
}
```

10.33.3.2 template<typename DATA_TYPE , unsigned SIZE> QuadraticCurve< DATA_TYPE, SIZE > & gmtl::QuadraticCurve< DATA_TYPE, SIZE >::operator= (const QuadraticCurve< DATA_TYPE, SIZE > & *other*)

Definition at line 279 of file ParametricCurve.h.

```
{
   ParametricCurve::operator =(other);

   return *this;
}
```

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

- ParametricCurve.h

10.34 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

- enum Params { Size = 4 }
- typedef DATA_TYPE DataType

  use this to declare single value types of the same type as this matrix.

Public Member Functions

- Quat ()

  default constructor, initializes to quaternion multiplication identity [x,y,z,w] == [0,0,0,1].

- Quat (const DATA_TYPE &x, const DATA_TYPE &y, const DATA_TYPE &z, const DATA_TYPE &w)

  data 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)

  directly set the quaternion's values

- void get (DATA_TYPE &x, DATA_TYPE &y, DATA_TYPE &z, DATA_TYPE &w) const

  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

10.34.1 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 47 of file Quat.h.

10.34.2 Member Typedef Documentation

10.34.2.1 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 52 of file Quat.h.

10.34.3 Member Enumeration Documentation

10.34.3.1 template<typename DATA_TYPE> enum gmtl::Quat::Params

Enumerator:

Size

Definition at line 54 of file Quat.h.

```
{ Size = 4 };
```

10.34.4 Constructor & Destructor Documentation

10.34.4.1 template<typename DATA_TYPE> gmtl::Quat< DATA_TYPE >::Quat ( ) `[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]

Definition at line 60 of file Quat.h.

```
      : mData( (DATA_TYPE)0.0, (DATA_TYPE)0.0, (DATA_TYPE)0.0, (DATA_TYPE)1.0 )
   {
   }
```

10.34.4.2 template<typename DATA_TYPE> gmtl::Quat< DATA_TYPE >::Quat (const DATA_TYPE & *x*, const DATA_TYPE & *y*, const DATA_TYPE & *z*, const DATA_TYPE & *w*) `[inline]`

data constructor, initializes to quaternion multiplication identity [x,y,z,w] == [0,0,0,1].

NOTE: the addition identity is [0,0,0,0]

Definition at line 69 of file Quat.h.

```
    : mData( x, y, z, w )
{
}
```

10.34.4.3 template<typename DATA_TYPE> gmtl::Quat< DATA_TYPE >::Quat (const Quat< DATA_TYPE > & *q*) [inline]

copy constructor

Definition at line 77 of file Quat.h.

```
                                : mData( q.mData )
{
}
```

10.34.5 Member Function Documentation

10.34.5.1 template<typename DATA_TYPE> void gmtl::Quat< DATA_TYPE >::get (DATA_TYPE & *x*, DATA_TYPE & *y*, DATA_TYPE & *z*, DATA_TYPE & *w*) const [inline]

get the raw data elements of the quaternion.

Postcondition

sets the given variables to the quaternion's x, y, z, and w values

Definition at line 93 of file Quat.h.

```
{
    x = mData[Xelt];
    y = mData[Yelt];
    z = mData[Zelt];
    w = mData[Welt];
}
```

10.34.5.2 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 141 of file Quat.h.

```
{ return (DATA_TYPE*)mData.getData();}
```

10.34.5.3 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 132 of file Quat.h.

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

10.34.5.4 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 115 of file Quat.h.

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

10.34.5.5 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]

directly set the quaternion's values

Precondition

x,y,z,w should be normalized

Postcondition

the quaternion is set with the given values

Definition at line 85 of file Quat.h.

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

10.34.6 Member Data Documentation

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

Definition at line 145 of file Quat.h.

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

- Quat.h

10.35 gmtl::Ray< DATA_TYPE > Class Template Reference

Describes a ray.

`#include <Ray.h>`

Inheritance diagram for gmtl::Ray< DATA_TYPE >:

Collaboration diagram for gmtl::Ray< DATA_TYPE >:

Public Member Functions

- Ray ()

  Constructs a ray at the origin with a zero vector.

- Ray (const Point< DATA_TYPE, 3 > &origin, const Vec< DATA_TYPE, 3 > &dir)

  Constructs a ray with the given origin and vector.

- Ray (const Ray &lineseg)

  Constructs an exact duplicate of the given ray.

- const Point< DATA_TYPE, 3 > & getOrigin () const

  Gets the origin of the ray.

- void setOrigin (const Point< DATA_TYPE, 3 > &origin)

  Sets the origin point for this ray.

- const Vec< DATA_TYPE, 3 > & getDir () const

  Gets the vector describing the direction and length of the ray.

- void setDir (const Vec< DATA_TYPE, 3 > &dir)

  Sets the vector describing the direction and length of the ray.

Public Attributes

- Point< DATA_TYPE, 3 > mOrigin

  The origin of the ray.

- Vec< DATA_TYPE, 3 > mDir

  The vector along which the ray lies.

10.35.1 Detailed Description

template<class DATA_TYPE> class gmtl::Ray< DATA_TYPE >

Describes a ray. This is represented by a point origin O and a normalized vector direction. Any point on the ray 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 26 of file Ray.h.

10.35.2 Constructor & Destructor Documentation

10.35.2.1 template<class DATA_TYPE> gmtl::Ray< DATA_TYPE >::Ray ( ) [inline]

Constructs a ray at the origin with a zero vector.

Definition at line 32 of file Ray.h.

```
{}
```

10.35.2.2 template<class DATA_TYPE> gmtl::Ray< DATA_TYPE >::Ray (const Point< DATA_TYPE, 3 > & *origin*, const Vec< DATA_TYPE, 3 > & *dir*) [inline]

Constructs a ray with the given origin and vector.

Parameters

origin the point at which the ray starts

dir the vector describing the direction and length of the ray starting at origin

Definition at line 42 of file Ray.h.

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

10.35.2.3 template<class DATA_TYPE> gmtl::Ray< DATA_TYPE >::Ray (const Ray< DATA_TYPE > & *lineseg*) [inline]

Constructs an exact duplicate of the given ray.

Parameters

lineseg the ray to copy

Definition at line 53 of file Ray.h.

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

10.35.3 Member Function Documentation

10.35.3.1 template<class DATA_TYPE> const Vec<DATA_TYPE, 3>& gmtl::Ray< DATA_TYPE >::getDir ( ) const [inline]

Gets the vector describing the direction and length of the ray.

Returns

the ray's vector

Definition at line 84 of file Ray.h.

```
{
   return mDir;
}
```

10.35.3.2 template<class DATA_TYPE> const Point<DATA_TYPE, 3>& gmtl::Ray< DATA_TYPE >::getOrigin ( ) const [inline]

Gets the origin of the ray.

Returns

the point at the beginning of the line

Definition at line 64 of file Ray.h.

```
{
   return mOrigin;
}
```

10.35.3.3 template<class DATA_TYPE> void gmtl::Ray< DATA_TYPE >::setDir (const Vec< DATA_TYPE, 3 > & *dir*) `[inline]`

Sets the vector describing the direction and length of the ray.

Parameters

dir the ray's vector

Definition at line 94 of file Ray.h.

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

10.35.3.4 template<class DATA_TYPE> void gmtl::Ray< DATA_TYPE >::setOrigin (const Point< DATA_TYPE, 3 > & *origin*) `[inline]`

Sets the origin point for this ray.

Parameters

origin the point at which the ray starts

Definition at line 74 of file Ray.h.

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

10.35.4 Member Data Documentation

10.35.4.1 template<class DATA_TYPE> Vec<DATA_TYPE, 3> gmtl::Ray< DATA_TYPE >::mDir

The vector along which the ray lies.

Definition at line 108 of file Ray.h.

10.35.4.2 template<class DATA_TYPE> Point<DATA_TYPE, 3> gmtl::Ray< DATA_TYPE >::mOrigin

The origin of the ray.

Definition at line 103 of file Ray.h.

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

- Ray.h

10.36 gmtl::RotationOrderBase Struct Reference

Base class for Rotation orders.

`#include <Math.h>`

Inheritance diagram for gmtl::RotationOrderBase:

Public Types

- enum { IS_ROTORDER = 1 }

10.36.1 Detailed Description

Base class for Rotation orders.

See also

XYZ, ZYX, ZXY

Definition at line 22 of file Math.h.

10.36.2 Member Enumeration Documentation

10.36.2.1 anonymous enum

Enumerator:

IS_ROTORDER

Definition at line 22 of file Math.h.

```
{ enum { IS_ROTORDER = 1 }; };
```

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

- Math.h

10.37 gmtl::Matrix< DATA_TYPE, ROWS, COLS >::RowAccessor Class Reference

Helper class for Matrix op[].

`#include <Matrix.h>`

Public Types

- typedef DATA_TYPE DataType

Public Member Functions

- RowAccessor (Matrix< DATA_TYPE, ROWS, COLS > *mat, const unsigned row)
- DATA_TYPE & operator[ ] (const unsigned column)

Public Attributes

- Matrix< DATA_TYPE, ROWS, COLS > * mMat
- unsigned mRow

10.37.1 Detailed Description

template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> class gmtl::Matrix< DATA_TYPE, ROWS, COLS >::RowAccessor

Helper class for Matrix op[]. This class encapsulates the row that the user is accessing and implements a new op[] that passes the column to use

Definition at line 131 of file Matrix.h.

10.37.2 Member Typedef Documentation

10.37.2.1 template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> typedef DATA_TYPE gmtl::Matrix< DATA_TYPE, ROWS, COLS >::RowAccessor::DataType

Definition at line 134 of file Matrix.h.

10.37.3 Constructor & Destructor Documentation

10.37.3.1 template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> gmtl::Matrix< DATA_TYPE, ROWS, COLS >::RowAccessor::RowAccessor (Matrix< DATA_TYPE, ROWS, COLS > * *mat*, const unsigned *row*) `[inline]`

Definition at line 136 of file Matrix.h.

```
    : mMat(mat), mRow(row)
{
    gmtlASSERT(row < ROWS);
    gmtlASSERT(NULL != mat);
}
```

10.37.4 Member Function Documentation

10.37.4.1 template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> DATA_TYPE& gmtl::Matrix< DATA_TYPE, ROWS, COLS >::RowAccessor::operator[ ] (const unsigned *column*) `[inline]`

Definition at line 143 of file Matrix.h.

```
{
    gmtlASSERT(column < COLS);
    return (*mMat)(mRow,column);
}
```

10.37.5 Member Data Documentation

10.37.5.1 template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> Matrix<DATA_TYPE,ROWS,COLS>* gmtl::Matrix< DATA_TYPE, ROWS, COLS >::RowAccessor::mMat

Definition at line 149 of file Matrix.h.

10.37.5.2 template<typename DATA_TYPE, unsigned ROWS, unsigned COLS> unsigned gmtl::Matrix< DATA_TYPE, ROWS, COLS >::RowAccessor::mRow

Definition at line 150 of file Matrix.h.

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

- Matrix.h

10.38 gmtl::meta::ScalarArg< T > Struct Template Reference

template to hold a scalar argument.

`#include <VecExprMeta.h>`

Public Types

- typedef T DataType

Public Member Functions

- ScalarArg (const T scalar)
- T operator[ ] (const unsigned) const

Public Attributes

- const T mScalar

10.38.1 Detailed Description

template<typename T> struct gmtl::meta::ScalarArg< T >

template to hold a scalar argument.

Definition at line 38 of file VecExprMeta.h.

10.38.2 Member Typedef Documentation

10.38.2.1 template<typename T > typedef T gmtl::meta::ScalarArg< T >::DataType

Definition at line 40 of file VecExprMeta.h.

10.38.3 Constructor & Destructor Documentation

10.38.3.1 template<typename T > gmtl::meta::ScalarArg< T >::ScalarArg (const T *scalar*) `[inline]`

Definition at line 44 of file VecExprMeta.h.

```
: mScalar(scalar) {}
```

10.38.4 Member Function Documentation

10.38.4.1 template<typename T > T gmtl::meta::ScalarArg< T >::operator[ ] (const *unsigned*) const [inline]

Definition at line 45 of file VecExprMeta.h.

```
{ return mScalar; }
```

10.38.5 Member Data Documentation

10.38.5.1 template<typename T > const T gmtl::meta::ScalarArg< T >::mScalar

Definition at line 42 of file VecExprMeta.h.

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

- VecExprMeta.h

10.39 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 Member Functions

- 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.

10.39.1 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 21 of file Sphere.h.

10.39.2 Member Typedef Documentation

10.39.2.1 template<class DATA_TYPE> typedef DATA_TYPE gmtl::Sphere< DATA_TYPE >::DataType

Definition at line 24 of file Sphere.h.

10.39.3 Constructor & Destructor Documentation

10.39.3.1 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 30 of file Sphere.h.

```
    : mRadius( 0 )
{}
```

10.39.3.2 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 40 of file Sphere.h.

```
    : mCenter( center ), mRadius( radius )
{}
```

10.39.3.3 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 49 of file Sphere.h.

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

10.39.4 Member Function Documentation

10.39.4.1 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 58 of file Sphere.h.

```
{
    return mCenter;
}
```

10.39.4.2 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 68 of file Sphere.h.

```
{
    return mRadius;
}
```

10.39.4.3 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 78 of file Sphere.h.

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

10.39.4.4 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 88 of file Sphere.h.

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

10.39.5 Member Data Documentation

10.39.5.1 template<class DATA_TYPE> Point<DATA_TYPE, 3> gmtl::Sphere< DATA_TYPE >::mCenter

The center of the sphere.

Definition at line 97 of file Sphere.h.

10.39.5.2 template<class DATA_TYPE> DATA_TYPE gmtl::Sphere< DATA_TYPE >::mRadius

The radius of the sphere.

Definition at line 102 of file Sphere.h.

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

- Sphere.h

10.40 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 Member Functions

- 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.

- Vec< DATA_TYPE, 3 > edge (int idx, int idy) const

  get edge vec from two verts

- void set (const Point< DATA_TYPE, 3 > &p1, const Point< DATA_TYPE, 3 > &p2, const Point< DATA_TYPE, 3 > &p3)

  Set the triangle with the given points.

  - Point< DATA_TYPE, 3 > & operator[ ] (const unsigned idx)

    Gets the nth vertex in the triangle.

  - const Point< DATA_TYPE, 3 > & operator[ ] (const unsigned idx) const

Public Attributes

- Point< DATA_TYPE, 3 > mVerts [3]

  The vertices of the triangle.

10.40.1 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 23 of file Tri.h.

10.40.2 Constructor & Destructor Documentation

10.40.2.1 template<class DATA_TYPE> gmtl::Tri< DATA_TYPE >::Tri ( ) `[inline]`

Constructs a new triangle with all vertices at the origin.

Definition at line 29 of file Tri.h.

```
{}
```

10.40.2.2 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 41 of file Tri.h.

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

10.40.2.3 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 54 of file Tri.h.

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

10.40.3 Member Function Documentation

10.40.3.1 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 92 of file Tri.h.

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

10.40.3.2 template<class DATA_TYPE> Vec<DATA_TYPE, 3> gmtl::Tri< DATA_TYPE >::edge (int *idx,* int *idy*) const [inline]

get edge vec from two verts

Parameters

idx,idy the ordered vertex indicies

Precondition

0 <= id <= 2

Returns

a vector from vertex idx to vertex idy

Definition at line 106 of file Tri.h.

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

10.40.3.3 template<class DATA_TYPE> const Point<DATA_TYPE, 3>& gmtl::Tri< DATA_TYPE >::operator[ ] (const unsigned *idx*) const [inline]

Definition at line 75 of file Tri.h.

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

10.40.3.4 template<class DATA_TYPE> Point<DATA_TYPE, 3>& gmtl::Tri< DATA_TYPE >::operator[ ] (const unsigned *idx*) [inline]

Gets the nth vertex in the triangle.

Parameters

idx the index to the vertex in the triangle

Precondition

0 <= idx <= 2

Returns

the nth vertex as a point

Definition at line 70 of file Tri.h.

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

10.40.3.5 template<class DATA_TYPE> void gmtl::Tri< DATA_TYPE >::set (const Point< DATA_TYPE, 3 > & *p1*, const Point< DATA_TYPE, 3 > & *p2*, const Point< DATA_TYPE, 3 > & *p3*) `[inline]`

Set the 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 125 of file Tri.h.

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

10.40.4 Member Data Documentation

10.40.4.1 template<class DATA_TYPE> Point<DATA_TYPE, 3> gmtl::Tri< DATA_TYPE >::mVerts[3]

The vertices of the triangle.

Definition at line 137 of file Tri.h.

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

- Tri.h

10.41 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

10.41.1 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 47 of file Meta.h.

10.41.2 Member Typedef Documentation

10.41.2.1 template<typename T > typedef T gmtl::Type2Type< T >::OriginalType

Definition at line 49 of file Meta.h.

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

- Meta.h

10.42 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< DATA_TYPE, SIZE >:

Collaboration diagram for gmtl::Vec< DATA_TYPE, SIZE >:

Public Types

- enum Params { Size = SIZE }

  The number of components this Vec has.

- typedef DATA_TYPE DataType

  The datatype used for the components of this Vec.

- typedef VecBase< DATA_TYPE, SIZE > BaseType

  The superclass type.

- typedef Vec< DATA_TYPE, SIZE > VecType

Public Member Functions

- Vec ()

  Default constructor.

- template<typename REP2 >
  VecType & operator= (const VecBase< DATA_TYPE, SIZE, REP2 > &rhs)

  Assign from different rep.

Value constructors

- template<typename REP2 >
  Vec (const VecBase< DATA_TYPE, SIZE, REP2 > &rVec)

  Make an exact copy of the given Vec object.

- 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)

10.42.1 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 33 of file Vec.h.

10.42.2 Member Typedef Documentation

10.42.2.1 template<class DATA_TYPE, unsigned SIZE> typedef VecBase<DATA_TYPE, SIZE> gmtl::Vec< DATA_TYPE, SIZE >::BaseType

The superclass type.

Definition at line 44 of file Vec.h.

10.42.2.2 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, meta::DefaultVecTag >.

Definition at line 38 of file Vec.h.

10.42.2.3 template<class DATA_TYPE, unsigned SIZE> typedef Vec<DATA_TYPE, SIZE> gmtl::Vec< DATA_TYPE, SIZE >::VecType

Reimplemented from gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >.

Definition at line 45 of file Vec.h.

10.42.3 Member Enumeration Documentation

10.42.3.1 template<class DATA_TYPE, unsigned SIZE> enum gmtl::Vec::Params

The number of components this Vec has.

Enumerator:

Size

Reimplemented from gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >.

Definition at line 41 of file Vec.h.

```
{ Size = SIZE };
```

10.42.4 Constructor & Destructor Documentation

10.42.4.1 template<class DATA_TYPE, unsigned SIZE> gmtl::Vec< DATA_TYPE, SIZE >::Vec ( ) [inline]

Default constructor.

All components are initialized to zero.

Definition at line 51 of file Vec.h.

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

10.42.4.2 template<class DATA_TYPE, unsigned SIZE> template<typename REP2 > gmtl::Vec< DATA_TYPE, SIZE >::Vec (const VecBase< DATA_TYPE, SIZE, REP2 > & *rVec*) [inline]

Make an exact copy of the given Vec object.

Precondition

Vector should be the same size and type as the one copied

Parameters

rVec the Vec object to copy

Definition at line 77 of file Vec.h.

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

10.42.4.3 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 86 of file Vec.h.

```
: BaseType(val0, val1)
{
   GMTL_STATIC_ASSERT( SIZE == 2, Out_Of_Bounds_Element_Access_In_Vec );
}
```

10.42.4.4 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 92 of file Vec.h.

```
: BaseType(val0, val1, val2)
{
   GMTL_STATIC_ASSERT( SIZE == 3, Out_Of_Bounds_Element_Access_In_Vec );
}
```

10.42.4.5 template<class DATA_TYPE, unsigned SIZE> gmtl::Vec< DATA_TYPE, SIZE >::Vec (const DATA_TYPE & *val0*, const DATA_TYPE & *val1*, const DATA_TYPE & *val2*, const DATA_TYPE & *val3*) [inline]

Definition at line 98 of file Vec.h.

```
: BaseType(val0, val1, val2, val3)
{
   GMTL_STATIC_ASSERT( SIZE == 4, Out_Of_Bounds_Element_Access_In_Vec );
}
```

10.42.5 Member Function Documentation

10.42.5.1 template<class DATA_TYPE, unsigned SIZE> template<typename REP2 > VecType& gmtl::Vec< DATA_TYPE, SIZE >::operator= (const VecBase< DATA_TYPE, SIZE, REP2 > & *rhs*) `[inline]`

Assign from different rep.

Reimplemented from gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >.

Definition at line 114 of file Vec.h.

```
{
   BaseType::operator=(rhs);
   return *this;
}
```

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

- Vec.h

10.43 gmtl::VecBase< DATA_TYPE, SIZE, REP > Class Template Reference

Base type for vector-like objects including Points and Vectors.

```
#include <VecBase.h>
```

Collaboration diagram for gmtl::VecBase< DATA_TYPE, SIZE, REP >:

Public Types

- enum Params { Size = SIZE }

  The number of components this VecB has.

- typedef DATA_TYPE DataType

  The datatype used for the components of this VecB.

Public Member Functions

- VecBase ()
- VecBase (const REP &rep)
- DATA_TYPE operator[ ] (const unsigned i)

Conversion operator to default vecbase type.

- const DATA_TYPE operator[ ] (const unsigned i) const

Protected Attributes

- const REP expRep

10.43.1 Detailed Description

template<class DATA_TYPE, unsigned SIZE, typename REP = meta::DefaultVecTag> class gmtl::VecBase< DATA_TYPE, SIZE, REP >

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 VecB has

REP the representation to use for the vector. (expression template or default)

Definition at line 40 of file VecBase.h.

10.43.2 Member Typedef Documentation

10.43.2.1 template<class DATA_TYPE, unsigned SIZE, typename REP = meta::DefaultVecTag> typedef DATA_TYPE gmtl::VecBase< DATA_TYPE, SIZE, REP >::DataType

The datatype used for the components of this VecB.

Reimplemented in gmtl::Point< DATA_TYPE, SIZE >, and gmtl::Point< DATA_-TYPE, 3 >.

Definition at line 47 of file VecBase.h.

10.43.3 Member Enumeration Documentation

10.43.3.1 template<class DATA_TYPE, unsigned SIZE, typename REP = meta::DefaultVecTag> enum gmtl::VecBase::Params

The number of components this VecB has.

Enumerator:

Size

Reimplemented in gmtl::AxisAngle< DATA_TYPE >, gmtl::Point< DATA_TYPE, SIZE >, and gmtl::Point< DATA_TYPE, 3 >.

Definition at line 50 of file VecBase.h.

```
{ Size = SIZE };
```

10.43.4 Constructor & Destructor Documentation

10.43.4.1 template<class DATA_TYPE, unsigned SIZE, typename REP = meta::DefaultVecTag> gmtl::VecBase< DATA_TYPE, SIZE, REP >::VecBase ( ) [inline]

Definition at line 53 of file VecBase.h.

```
{;}
```

10.43.4.2 template<class DATA_TYPE, unsigned SIZE, typename REP = meta::DefaultVecTag> gmtl::VecBase< DATA_TYPE, SIZE, REP >::VecBase (const REP & *rep*) [inline]

Definition at line 56 of file VecBase.h.

```
   : expRep(rep)
{;}
```

10.43.5 Member Function Documentation

10.43.5.1 template<class DATA_TYPE, unsigned SIZE, typename REP = meta::DefaultVecTag> DATA_TYPE gmtl::VecBase< DATA_TYPE, SIZE, REP >::operator[] (const unsigned *i*) [inline]

Conversion operator to default vecbase type.

Return the value at given location.

Definition at line 69 of file VecBase.h.

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

10.43.5.2 template<class DATA_TYPE, unsigned SIZE, typename REP = meta::DefaultVecTag> const DATA_TYPE gmtl::VecBase< DATA_TYPE, SIZE, REP >::operator[ ] (const unsigned *i*) const [inline]

Definition at line 74 of file VecBase.h.

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

10.43.6 Member Data Documentation

10.43.6.1 template<class DATA_TYPE, unsigned SIZE, typename REP = meta::DefaultVecTag> const REP gmtl::VecBase< DATA_TYPE, SIZE, REP >::expRep [protected]

Definition at line 43 of file VecBase.h.

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

- VecBase.h

10.44 gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag > Class Template Reference

Specialized version of VecBase that is actually used for all user interaction with a traditional vector.

`#include <VecBase.h>`

Inheritance diagram for gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >:

Collaboration diagram for gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >:

Public Types

- enum Params { Size = SIZE }

  The number of components this VecBase has.

- typedef DATA_TYPE DataType

  The datatype used for the components of this VecBase.

- typedef VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag > VecType

Public Member Functions

- VecBase ()

  Default constructor.

- VecBase (const VecBase< DATA_TYPE, SIZE > &rVec)

  Makes an exact copy of the given VecBase object.

- template<typename REP2 >
  VecBase (const VecBase< DATA_TYPE, SIZE, REP2 > &rVec)
- void set (const DATA_TYPE ∗dataPtr)

  Sets the components in this VecBase using the given array.

- template<typename REP2 >
  VecType & operator= (const VecBase< DATA_TYPE, SIZE, REP2 > &rhs)

  Assign from different rep.

  - 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.

10.44.1 Detailed Description

template<class DATA_TYPE, unsigned SIZE> class gmtl::VecBase< DATA_-TYPE, SIZE, meta::DefaultVecTag >

Specialized version of VecBase that is actually used for all user interaction with a traditional vector.

Parameters

DATA_TYPE the datatype to use for the components

SIZE the number of components this VecBase has

Definition at line 94 of file VecBase.h.

10.44.2 Member Typedef Documentation

10.44.2.1 template<class DATA_TYPE , unsigned SIZE> typedef DATA_TYPE gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >::DataType

The datatype used for the components of this VecBase.

Reimplemented in gmtl::Vec< DATA_TYPE, SIZE >, gmtl::Vec< DATA_TYPE, 3 >, and gmtl::Vec< DATA_TYPE, 4 >.

Definition at line 99 of file VecBase.h.

10.44.2.2 template<class DATA_TYPE , unsigned SIZE> typedef VecBase<DATA_TYPE, SIZE, meta::DefaultVecTag> gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >::VecType

Reimplemented in gmtl::Vec< DATA_TYPE, SIZE >, gmtl::Vec< DATA_TYPE, 3 >, and gmtl::Vec< DATA_TYPE, 4 >.

Definition at line 104 of file VecBase.h.

10.44.3 Member Enumeration Documentation

10.44.3.1 template<class DATA_TYPE , unsigned SIZE> enum gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >::Params

The number of components this VecBase has.

Enumerator:

Size

Reimplemented in gmtl::Vec< DATA_TYPE, SIZE >, gmtl::Vec< DATA_TYPE, 3 >, and gmtl::Vec< DATA_TYPE, 4 >.

Definition at line 108 of file VecBase.h.

```
{ Size = SIZE };
```

10.44.4 Constructor & Destructor Documentation

10.44.4.1 template<class DATA_TYPE , unsigned SIZE> gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >::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 117 of file VecBase.h.

```
   {
#ifdef GMTL_COUNT_CONSTRUCT_CALLS
      gmtl::helpers::VecCtrCounterInstance()->inc();
#endif
   }
```

10.44.4.2 template<class DATA_TYPE , unsigned SIZE> gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >::VecBase (const VecBase< DATA_TYPE, SIZE > & *rVec*) `[inline]`

Makes an exact copy of the given VecBase object.

Parameters

rVec the VecBase object to copy

Definition at line 129 of file VecBase.h.

```
   {
#ifdef GMTL_COUNT_CONSTRUCT_CALLS
      gmtl::helpers::VecCtrCounterInstance()->inc();
#endif
#ifdef GMTL_NO_METAPROG
      for(unsigned i=0;i<SIZE;++i)
         mData[i] = rVec.mData[i];
#else
      gmtl::meta::AssignVecUnrolled<SIZE-1, VecBase<DATA_TYPE,SIZE> >::func(*this
      , rVec);
#endif
   }
```

10.44.4.3 template<class DATA_TYPE , unsigned SIZE> template<typename REP2 > gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >::VecBase (const VecBase< DATA_TYPE, SIZE, REP2 > & *rVec*) `[inline]`

Definition at line 144 of file VecBase.h.

```
   {
#ifdef GMTL_COUNT_CONSTRUCT_CALLS
      gmtl::helpers::VecCtrCounterInstance()->inc();
#endif
      for(unsigned i=0;i<SIZE;++i)
      {  mData[i] = rVec[i]; }
   }
```

10.44.4.4 template<class DATA_TYPE , unsigned SIZE> gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >::VecBase (const DATA_TYPE & *val0*, const DATA_TYPE & *val1*) `[inline]`

Creates a new VecBase initialized to the given values.

Definition at line 158 of file VecBase.h.

```
   {
#ifdef GMTL_COUNT_CONSTRUCT_CALLS
      gmtl::helpers::VecCtrCounterInstance()->inc();
#endif
      GMTL_STATIC_ASSERT( SIZE == 2, Invalid_constructor_of_size_2_used);
      mData[0] = val0; mData[1] = val1;
   }
```

10.44.4.5 template<class DATA_TYPE , unsigned SIZE> gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >::VecBase (const DATA_TYPE & *val0*, const DATA_TYPE & *val1*, const DATA_TYPE & *val2*) `[inline]`

Definition at line 166 of file VecBase.h.

```
   {
#ifdef GMTL_COUNT_CONSTRUCT_CALLS
      gmtl::helpers::VecCtrCounterInstance()->inc();
#endif
      GMTL_STATIC_ASSERT( SIZE == 3, Invalid_constructor_of_size_3_used );
      mData[0] = val0;  mData[1] = val1;  mData[2] = val2;
   }
```

10.44.4.6 template<class DATA_TYPE , unsigned SIZE> gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >::VecBase (const DATA_TYPE & *val0*, const DATA_TYPE & *val1*, const DATA_TYPE & *val2*, const DATA_TYPE & *val3*) `[inline]`

Definition at line 174 of file VecBase.h.

```
   {
#ifdef GMTL_COUNT_CONSTRUCT_CALLS
      gmtl::helpers::VecCtrCounterInstance()->inc();
#endif
      // @todo need compile time assert
      GMTL_STATIC_ASSERT( SIZE == 4, Invalid_constructor_of_size_4_used);
      mData[0] = val0;  mData[1] = val1;  mData[2] = val2;  mData[3] = val3;
   }
```

10.44.5 Member Function Documentation

10.44.5.1 template<class DATA_TYPE , unsigned SIZE> DATA_TYPE∗ gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >::getData ( ) `[inline]`

Gets the internal array of the components.

Returns

a pointer to the component array with length SIZE

Definition at line 292 of file VecBase.h.

```
{ return mData; }
```

10.44.5.2 template<class DATA_TYPE , unsigned SIZE> const DATA_TYPE* gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >::getData ( ) const [inline]

Definition at line 294 of file VecBase.h.

```
{ return mData; }
```

10.44.5.3 template<class DATA_TYPE , unsigned SIZE> template<typename REP2 > VecType& gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >::operator= (const VecBase< DATA_TYPE, SIZE, REP2 > & *rhs*) [inline]

Assign from different rep.

Reimplemented in gmtl::Vec< DATA_TYPE, SIZE >, gmtl::Vec< DATA_TYPE, 3 >, and gmtl::Vec< DATA_TYPE, 4 >.

Definition at line 262 of file VecBase.h.

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

   //gmtl::meta::AssignVecUnrolled<SIZE-1, VecBase<DATA_TYPE,SIZE> >::func(*th
   is, rVec);
   return *this;
}
```

10.44.5.4 template<class DATA_TYPE , unsigned SIZE> DATA_TYPE& gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >::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 237 of file VecBase.h.

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

10.44.5.5 template<class DATA_TYPE , unsigned SIZE> const DATA_TYPE& gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >::operator[] (const unsigned *i*) const [inline]

Definition at line 242 of file VecBase.h.

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

10.44.5.6 template<class DATA_TYPE , unsigned SIZE> void gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >::set (const DATA_TYPE & *val0*, const DATA_TYPE & *val1*) [inline]

Definition at line 211 of file VecBase.h.

```
{
   GMTL_STATIC_ASSERT( SIZE >= 2, Set_out_of_valid_range);
   mData[0] = val0; mData[1] = val1;
}
```

10.44.5.7 template<class DATA_TYPE , unsigned SIZE> void gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >::set (const DATA_TYPE & *val0*, const DATA_TYPE & *val1*, const DATA_TYPE & *val2*, const DATA_TYPE & *val3*) [inline]

Definition at line 221 of file VecBase.h.

```
{
   GMTL_STATIC_ASSERT( SIZE >= 4, Set_out_of_valid_range);
   mData[0] = val0;  mData[1] = val1;  mData[2] = val2;  mData[3] = val3;
}
```

10.44.5.8 template<class DATA_TYPE , unsigned SIZE> void gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >::set (const DATA_TYPE & *val0*) [inline]

Sets the components in this VecBase to the given values.

Definition at line 208 of file VecBase.h.

```
{ mData[0] = val0; }
```

10.44.5.9 template<class DATA_TYPE , unsigned SIZE> void gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >::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 191 of file VecBase.h.

```
   {
#ifdef GMTL_NO_METAPROG
      for ( unsigned int i = 0; i < SIZE; ++i )
      {
         mData[i] = dataPtr[i];
      }
#else
      gmtl::meta::AssignArrayUnrolled<SIZE-1, DATA_TYPE>::func(&(mData[0]),
                                                               dataPtr);
#endif
   }
```

10.44.5.10 template<class DATA_TYPE , unsigned SIZE> void gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >::set (const DATA_TYPE & *val0*, const DATA_TYPE & *val1*, const DATA_TYPE & *val2*) [inline]

Definition at line 216 of file VecBase.h.

```
{
   GMTL_STATIC_ASSERT( SIZE >= 3, Set_out_of_valid_range);
   mData[0] = val0;  mData[1] = val1;  mData[2] = val2;
}
```

10.44.6 Member Data Documentation

10.44.6.1 template<class DATA_TYPE , unsigned SIZE> DATA_TYPE gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >::mData[SIZE]

The array of components.

Definition at line 300 of file VecBase.h.

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

- VecBase.h

10.45 gmtl::meta::VecBinaryExpr< EXP1_T, EXP2_-T, OP > Struct Template Reference

Binary vector expression.

`#include <VecExprMeta.h>`

Collaboration diagram for gmtl::meta::VecBinaryExpr< EXP1_T, EXP2_T, OP >:

Public Types

- typedef EXP1_T::DataType DataType

Public Member Functions

- VecBinaryExpr (const EXP1_T &e1, const EXP2_T &e2)
- DataType operator[ ] (const unsigned i) const

Public Attributes

- ExprTraits< EXP1_T >::ExprRef Exp1
- ExprTraits< EXP2_T >::ExprRef Exp2

10.45.1 Detailed Description

template<typename EXP1_T, typename EXP2_T, typename OP> struct gmtl::meta::VecBinaryExpr< EXP1_T, EXP2_T, OP >

Binary vector expression. Stores the two vector expressions to process.

Definition at line 83 of file VecExprMeta.h.

10.45.2 Member Typedef Documentation

10.45.2.1 template<typename EXP1_T , typename EXP2_T , typename OP > typedef EXP1_T::DataType gmtl::meta::VecBinaryExpr< EXP1_T, EXP2_T, OP >::DataType

Definition at line 85 of file VecExprMeta.h.

10.45.3 Constructor & Destructor Documentation

10.45.3.1 template<typename EXP1_T , typename EXP2_T , typename OP > gmtl::meta::VecBinaryExpr< EXP1_T, EXP2_T, OP >::VecBinaryExpr (const EXP1_T & *e1*, const EXP2_T & *e2*) [inline]

Definition at line 90 of file VecExprMeta.h.

```
: Exp1(e1), Exp2(e2) {;}
```

10.45.4 Member Function Documentation

10.45.4.1 template<typename EXP1_T , typename EXP2_T , typename OP > DataType gmtl::meta::VecBinaryExpr< EXP1_T, EXP2_T, OP >::operator[] (const unsigned *i*) const [inline]

Definition at line 91 of file VecExprMeta.h.

```
{ return OP::eval(Exp1[i], Exp2[i]); }
```

10.45.5 Member Data Documentation

10.45.5.1 template<typename EXP1_T , typename EXP2_T , typename OP > ExprTraits<EXP1_T>::ExprRef gmtl::meta::VecBinaryExpr< EXP1_T, EXP2_T, OP >::Exp1

Definition at line 87 of file VecExprMeta.h.

10.45.5.2 template<typename EXP1_T , typename EXP2_T , typename OP > ExprTraits<EXP2_T>::ExprRef gmtl::meta::VecBinaryExpr< EXP1_T, EXP2_T, OP >::Exp2

Definition at line 88 of file VecExprMeta.h.

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

- VecExprMeta.h

10.46 gmtl::meta::VecDivBinary Struct Reference

```
#include <VecExprMeta.h>
```

Static Public Member Functions

- template<typename T >
  static T eval (const T a1, const T a2)

10.46.1 Detailed Description

Definition at line 134 of file VecExprMeta.h.

10.46.2 Member Function Documentation

10.46.2.1 template<typename T > static T gmtl::meta::VecDivBinary::eval (const T *a1*, const T *a2*) `[inline, static]`

Definition at line 137 of file VecExprMeta.h.

```
{ return a1/a2; }
```

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

- VecExprMeta.h

10.47 gmtl::meta::VecMinusBinary Struct Reference

`#include <VecExprMeta.h>`

Static Public Member Functions

- template<typename T >
  static T eval (const T a1, const T a2)

10.47.1 Detailed Description

Definition at line 120 of file VecExprMeta.h.

10.47.2 Member Function Documentation

10.47.2.1 template<typename T > static T gmtl::meta::VecMinusBinary::eval (const T *a1*, const T *a2*) [inline, static]

Definition at line 123 of file VecExprMeta.h.

```
{ return a1-a2; }
```

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

- VecExprMeta.h

10.48 gmtl::meta::VecMultBinary Struct Reference

`#include <VecExprMeta.h>`

Static Public Member Functions

- template<typename T >
  static T eval (const T a1, const T a2)

10.48.1 Detailed Description

Definition at line 127 of file VecExprMeta.h.

10.48.2 Member Function Documentation

10.48.2.1 template<typename T > static T gmtl::meta::VecMultBinary::eval (const T *a1*, const T *a2*) [inline, static]

Definition at line 130 of file VecExprMeta.h.

```
{ return a1 * a2; }
```

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

- VecExprMeta.h

10.49 gmtl::meta::VecNegUnary Struct Reference

Negation of the values.

`#include <VecExprMeta.h>`

Static Public Member Functions

- template<typename T >
  static T eval (const T a1)

10.49.1 Detailed Description

Negation of the values.

Definition at line 142 of file VecExprMeta.h.

10.49.2 Member Function Documentation

10.49.2.1 template<typename T > static T gmtl::meta::VecNegUnary::eval (const T *a1*) [inline, static]

Definition at line 145 of file VecExprMeta.h.

```
{ return -a1; }
```

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

- VecExprMeta.h

10.50 gmtl::output::VecOutputter< DATA_TYPE, SIZE, REP > Struct Template Reference

Outputters for vector types.

`#include <Output.h>`

Static Public Member Functions

- static std::ostream & outStream (std::ostream &out, const VecBase< DATA_-TYPE, SIZE, REP > &v)

10.50.1 Detailed Description

template<typename DATA_TYPE, unsigned SIZE, typename REP> struct gmtl::output::VecOutputter< DATA_TYPE, SIZE, REP >

Outputters for vector types.

Definition at line 30 of file Output.h.

10.50.2 Member Function Documentation

10.50.2.1 template<typename DATA_TYPE , unsigned SIZE, typename REP > static std::ostream& gmtl::output::VecOutputter< DATA_TYPE, SIZE, REP >::outStream (std::ostream & *out*, const VecBase< DATA_TYPE, SIZE, REP > & *v*) `[inline, static]`

Definition at line 32 of file Output.h.

```
{
   VecBase<DATA_TYPE,SIZE, gmtl::meta::DefaultVecTag> temp_vec(v);
   VecOutputter<DATA_TYPE,SIZE,gmtl::meta::DefaultVecTag>::outStream(out
,v);
   return out;
}
```

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

- Output.h

10.51 gmtl::output::VecOutputter< DATA_TYPE, SIZE, gmtl::meta::DefaultVecTag > Struct Template Reference

`#include <Output.h>`

Static Public Member Functions

- static std::ostream & outStream (std::ostream &out, const VecBase< DATA_TYPE, SIZE, gmtl::meta::DefaultVecTag > &v)

10.51.1 Detailed Description

template<typename DATA_TYPE, unsigned SIZE> struct gmtl::output::VecOutputter< DATA_TYPE, SIZE, gmtl::meta::DefaultVecTag >

Definition at line 41 of file Output.h.

10.51.2 Member Function Documentation

10.51.2.1 template<typename DATA_TYPE , unsigned SIZE> static std::ostream& gmtl::output::VecOutputter< DATA_TYPE, SIZE, gmtl::meta::DefaultVecTag >::outStream (std::ostream & *out*, const VecBase< DATA_TYPE, SIZE, gmtl::meta::DefaultVecTag > & *v*) `[inline, static]`

Definition at line 43 of file Output.h.

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

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

- Output.h

10.52 gmtl::meta::VecPlusBinary Struct Reference

`#include <VecExprMeta.h>`

Static Public Member Functions

- template<typename T >
  static T eval (const T a1, const T a2)

10.52.1 Detailed Description

Definition at line 112 of file VecExprMeta.h.

10.52.2 Member Function Documentation

10.52.2.1 template<typename T > static T gmtl::meta::VecPlusBinary::eval (const T *a1*, const T *a2*) [inline, static]

Definition at line 115 of file VecExprMeta.h.

```
{ return a1+a2; }
```

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

- VecExprMeta.h

10.53 gmtl::meta::VecUnaryExpr< EXP1_T, OP > Struct Template Reference

Unary vector expression.

`#include <VecExprMeta.h>`

Collaboration diagram for gmtl::meta::VecUnaryExpr< EXP1_T, OP >:

Public Types

- typedef EXP1_T::DataType DataType

Public Member Functions

- VecUnaryExpr (const EXP1_T &e1)
- DataType operator[ ] (const unsigned i) const

Public Attributes

- ExprTraits< EXP1_T >::ExprRef Exp1

10.53.1 Detailed Description

template<typename EXP1_T, typename OP> struct gmtl::meta::VecUnaryExpr< EXP1_T, OP >

Unary vector expression. Holds the vector expression and unary operation to apply to it.

Definition at line 99 of file VecExprMeta.h.

10.53.2 Member Typedef Documentation

10.53.2.1 template<typename EXP1_T , typename OP > typedef EXP1_T::DataType gmtl::meta::VecUnaryExpr< EXP1_T, OP >::DataType

Definition at line 101 of file VecExprMeta.h.

10.53.3 Constructor & Destructor Documentation

10.53.3.1 template<typename EXP1_T , typename OP > gmtl::meta::VecUnaryExpr< EXP1_T, OP >::VecUnaryExpr (const EXP1_T & *e1*) [inline]

Definition at line 105 of file VecExprMeta.h.

```
: Exp1(e1) {;}
```

10.53.4 Member Function Documentation

10.53.4.1 template<typename EXP1_T , typename OP > DataType gmtl::meta::VecUnaryExpr< EXP1_T, OP >::operator[ ] (const unsigned *i*) const [inline]

Definition at line 106 of file VecExprMeta.h.

```
{ return OP::eval(Exp1[i]); }
```

10.53.5 Member Data Documentation

10.53.5.1 template<typename EXP1_T , typename OP > ExprTraits<EXP1_T>::ExprRef gmtl::meta::VecUnaryExpr< EXP1_T, OP >::Exp1

Definition at line 103 of file VecExprMeta.h.

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

- VecExprMeta.h

10.54 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 }

10.54.1 Detailed Description

XYZ Rotation order.

Definition at line 26 of file Math.h.

10.54.2 Member Enumeration Documentation

10.54.2.1 anonymous enum

Enumerator:

ID

Definition at line 26 of file Math.h.

```
: public RotationOrderBase { enum { ID = 0 }; };
```

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

- Math.h

10.55 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 }

10.55.1 Detailed Description

ZXY Rotation order.

Definition at line 34 of file Math.h.

10.55.2 Member Enumeration Documentation

10.55.2.1 anonymous enum

Enumerator:

ID

Definition at line 34 of file Math.h.

```
: public RotationOrderBase { enum { ID = 2 }; };
```

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

- Math.h

10.56 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 }

10.56.1 Detailed Description

ZYX Rotation order.

Definition at line 30 of file Math.h.

10.56.2 Member Enumeration Documentation

10.56.2.1 anonymous enum

Enumerator:

ID

Definition at line 30 of file Math.h.

```
: public RotationOrderBase { enum { ID = 1 }; };
```

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

- Math.h

Chapter 11

File Documentation

11.1 AABox.h File Reference

`#include <gmtl/Point.h>`

Include dependency graph for AABox.h: This graph shows which files directly or indirectly include this file:

Classes

- class gmtl::AABox< DATA_TYPE >

  Describes an axially aligned box in 3D space.

Namespaces

- namespace gmtl

  Meta programming classes.

Typedefs

- typedef AABox< float > gmtl::AABoxf
- typedef AABox< double > gmtl::AABoxd

11.2 AABoxOps.h File Reference

`#include <gmtl/AABox.h>`

`#include <gmtl/VecOps.h>`

Include dependency graph for AABoxOps.h: This graph shows which files directly or indirectly include this file:

Namespaces

- namespace gmtl

    Meta programming classes.

Functions

AABox Comparitors

- template<class DATA_TYPE >
  bool gmtl::operator== (const AABox< DATA_TYPE > &b1, const AABox< DATA_TYPE > &b2)

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

- template<class DATA_TYPE >
  bool gmtl::operator!= (const AABox< DATA_TYPE > &b1, const AABox< DATA_TYPE > &b2)

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

- template<class DATA_TYPE >
  bool gmtl::isEqual (const AABox< DATA_TYPE > &b1, const AABox< DATA_TYPE > &b2, const DATA_TYPE &eps)

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

11.3 Assert.h File Reference

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

Defines

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

11.3.1 Define Documentation

11.3.1.1 #define gmtlASSERT(*val*) ((void)0)

Definition at line 14 of file Assert.h.

11.4 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:

Classes

- class gmtl::AxisAngle< DATA_TYPE >

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

Namespaces

- namespace gmtl

  Meta programming classes.

Typedefs

- typedef AxisAngle< float > gmtl::AxisAnglef
- typedef AxisAngle< double > gmtl::AxisAngled

Functions

- const AxisAngle< float > gmtl::AXISANGLE_IDENTITYF (0.0f, 1.0f, 0.0f, 0.0f)
- const AxisAngle< double > gmtl::AXISANGLE_IDENTITYD (0.0, 1.0, 0.0, 0.0)

11.5 AxisAngleOps.h File Reference

`#include <gmtl/AxisAngle.h>`

Include dependency graph for AxisAngleOps.h: This graph shows which files directly or indirectly include this file:

Namespaces

- namespace gmtl

  Meta programming classes.

Functions

AxisAngle Comparitors

- template<class DATA_TYPE >
  bool gmtl::operator== (const AxisAngle< DATA_TYPE > &a1, const AxisAngle< DATA_TYPE > &a2)

  Compares 2 AxisAngles to see if they are exactly the same.

- template<class DATA_TYPE >
  bool gmtl::operator!= (const AxisAngle< DATA_TYPE > &a1, const AxisAngle< DATA_TYPE > &a2)

  Compares 2 AxisAngles to see if they are NOT exactly the same.

- template<class DATA_TYPE >
  bool gmtl::isEqual (const AxisAngle< DATA_TYPE > &a1, const AxisAngle< DATA_TYPE > &a2, const DATA_TYPE eps=0)

  Compares a1 and a2 to see if they are the same within the given epsilon tolerance.

11.6 Comparitors.h File Reference

`#include <gmtl/Vec3.h>`

`#include <gmtl/Point3.h>`

Include dependency graph for Comparitors.h:

Classes

- struct gmtl::CompareIndexPointProjections

Namespaces

- namespace gmtl

    Meta programming classes.

11.7 Config.h File Reference

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

11.8 Containment.h File Reference

```
#include <vector>
#include <gmtl/Sphere.h>
#include <gmtl/AABox.h>
#include <gmtl/Frustum.h>
#include <gmtl/Tri.h>
#include <gmtl/VecOps.h>
```

Include dependency graph for Containment.h: This graph shows which files directly or indirectly include this file:

Namespaces

- namespace gmtl

    Meta programming classes.

Functions

- template< class DATA_TYPE >
  bool gmtl::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 gmtl::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 gmtl::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 gmtl::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 gmtl::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 gmtl::isOnVolume (const Sphere< DATA_TYPE > &container, const Point< DATA_TYPE, 3 > &pt)

  Modifies the given sphere to tightly enclose all spheres in the given std::vector.

- template<class DATA_TYPE >
  bool gmtl::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.

- template<class DATA_TYPE >
  bool gmtl::isInVolume (const AABox< DATA_TYPE > &container, const Point< DATA_TYPE, 3 > &pt)

  Tests if the given point is inside (or on) the surface of the given AABox volume.

- template<class DATA_TYPE >
  bool gmtl::isInVolumeExclusive (const AABox< DATA_TYPE > &container, const Point< DATA_TYPE, 3 > &pt)

  Tests if the given point is inside (not on) the surface of the given AABox volume.

- template<class DATA_TYPE >
  bool gmtl::isInVolume (const AABox< DATA_TYPE > &container, const AABox< DATA_TYPE > &box)

  Tests if the given AABox is completely inside or on the surface of the given AABox container.

- template< class DATA_TYPE >
void gmtl::extendVolume (AABox< DATA_TYPE > &container, const Point< DATA_TYPE, 3 > &pt)

    Modifies the existing AABox to tightly enclose itself and the given point.

- template< class DATA_TYPE >
void gmtl::extendVolume (AABox< DATA_TYPE > &container, const AABox< DATA_TYPE > &box)

    Modifies the container to tightly enclose itself and the given AABox.

- template< class DATA_TYPE >
void gmtl::makeVolume (AABox< DATA_TYPE > &box, const Sphere< DATA_TYPE > &sph)

    Creates an AABox that tightly encloses the given Sphere.

- template< typename T >
bool gmtl::isInVolume (const Frustum< T > &f, const Point< T, 3 > &p, unsigned int &idx)
- template< typename T >
bool gmtl::isInVolume (const Frustum< T > &f, const Sphere< T > &s)
- template< typename T >
bool gmtl::isInVolume (const Frustum< T > &f, const AABox< T > &box)
- template< typename T >
bool gmtl::isInVolume (const Frustum< T > &f, const Tri< T > &tri)

Variables

- const unsigned int gmtl::IN_FRONT_OF_ALL_PLANES = 6

11.9 Coord.h File Reference

```
#include <gmtl/Defines.h>
#include <gmtl/Vec.h>
#include <gmtl/AxisAngle.h>
#include <gmtl/EulerAngle.h>
#include <gmtl/Quat.h>
#include <gmtl/Util/Meta.h>
#include <gmtl/Util/StaticAssert.h>
```

Include dependency graph for Coord.h: This graph shows which files directly or indirectly include this file:

Classes

- class gmtl::Coord< POS_TYPE, ROT_TYPE >

  coord is a position/rotation pair.

Namespaces

- namespace gmtl

  Meta programming classes.

Typedefs

- typedef Coord< Vec3d, EulerAngleXYZd > gmtl::CoordVec3EulerAngleXYZd
- typedef Coord< Vec3f, EulerAngleXYZf > gmtl::CoordVec3EulerAngleXYZf
- typedef Coord< Vec4d, EulerAngleXYZd > gmtl::CoordVec4EulerAngleXYZd
- typedef Coord< Vec4f, EulerAngleXYZf > gmtl::CoordVec4EulerAngleXYZf
- typedef Coord< Vec3d, EulerAngleZYXd > gmtl::CoordVec3EulerAngleZYXd
- typedef Coord< Vec3f, EulerAngleZYXf > gmtl::CoordVec3EulerAngleZYXf
- typedef Coord< Vec4d, EulerAngleZYXd > gmtl::CoordVec4EulerAngleZYXd
- typedef Coord< Vec4f, EulerAngleZYXf > gmtl::CoordVec4EulerAngleZYXf
- typedef Coord< Vec3d, EulerAngleZXYd > gmtl::CoordVec3EulerAngleZXYd
- typedef Coord< Vec3f, EulerAngleZXYf > gmtl::CoordVec3EulerAngleZXYf
- typedef Coord< Vec4d, EulerAngleZXYd > gmtl::CoordVec4EulerAngleZXYd
- typedef Coord< Vec4f, EulerAngleZXYf > gmtl::CoordVec4EulerAngleZXYf
- typedef Coord< Vec3d, AxisAngled > gmtl::CoordVec3AxisAngled
- typedef Coord< Vec3f, AxisAnglef > gmtl::CoordVec3AxisAnglef
- typedef Coord< Vec4d, AxisAngled > gmtl::CoordVec4AxisAngled
- typedef Coord< Vec4f, AxisAnglef > gmtl::CoordVec4AxisAnglef
- typedef Coord< Vec3f, EulerAngleXYZf > gmtl::Coord3fXYZ

  3 elt types

- typedef Coord< Vec3f, EulerAngleZYXf > gmtl::Coord3fZYX
- typedef Coord< Vec3f, EulerAngleZXYf > gmtl::Coord3fZXY
- typedef Coord< Vec3d, EulerAngleXYZd > gmtl::Coord3dXYZ
- typedef Coord< Vec3d, EulerAngleZYXd > gmtl::Coord3dZYX
- typedef Coord< Vec3d, EulerAngleZXYd > gmtl::Coord3dZXY
- typedef Coord< Vec4f, EulerAngleXYZf > gmtl::Coord4fXYZ

  4 elt types

- typedef Coord< Vec4f, EulerAngleZYXf > gmtl::Coord4fZYX

- typedef Coord< Vec4f, EulerAngleZXYf > gmtl::Coord4fZXY
- typedef Coord< Vec4d, EulerAngleXYZd > gmtl::Coord4dXYZ
- typedef Coord< Vec4d, EulerAngleZYXd > gmtl::Coord4dZYX
- typedef Coord< Vec4d, EulerAngleZXYd > gmtl::Coord4dZXY
- typedef Coord< Vec3f, Quatf > gmtl::Coord3fQuat

    3 elt types

- typedef Coord< Vec3d, Quatd > gmtl::Coord3dQuat
- typedef Coord< Vec4f, Quatf > gmtl::Coord4fQuat

    4 elt types

- typedef Coord< Vec4d, Quatd > gmtl::Coord4dQuat
- typedef Coord< Vec3f, AxisAnglef > gmtl::Coord3fAxisAngle

    3 elt types

- typedef Coord< Vec3d, AxisAngled > gmtl::Coord3dAxisAngle
- typedef Coord< Vec4f, AxisAnglef > gmtl::Coord4fAxisAngle

    4 elt types

- typedef Coord< Vec4d, AxisAngled > gmtl::Coord4dAxisAngle

11.10 CoordOps.h File Reference

```
#include <gmtl/Coord.h>
```

Include dependency graph for CoordOps.h: This graph shows which files directly or indirectly include this file:

Namespaces

- namespace gmtl

    Meta programming classes.

Functions

Coord Comparitors

- template<typename POS_TYPE , typename ROT_TYPE >
  bool gmtl::operator== (const Coord< POS_TYPE, ROT_TYPE > &c1, const Coord< POS_TYPE, ROT_TYPE > &c2)

    Compare two coordinate frames for equality.

- template<typename POS_TYPE , typename ROT_TYPE >
  bool gmtl::operator!= (const Coord< POS_TYPE, ROT_TYPE > &c1, const Coord< POS_TYPE, ROT_TYPE > &c2)

  Compare two coordinate frames for not-equality.

- template<typename POS_TYPE , typename ROT_TYPE >
  bool gmtl::isEqual (const Coord< POS_TYPE, ROT_TYPE > &c1, const Coord< POS_TYPE, ROT_TYPE > &c2, typename Coord< POS_TYPE, ROT_TYPE >::DataType tol=0)

  Compare two coordinate frames for equality with a given tolerance.

11.11 Defines.h File Reference

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

Namespaces

- namespace gmtl

  Meta programming classes.

Defines

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

Enumerations

- enum gmtl::VectorIndex { gmtl::Xelt = 0, gmtl::Yelt = 1, gmtl::Zelt = 2, gmtl::Welt = 3 }

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

- enum gmtl::PlaneSide { gmtl::ON_PLANE, gmtl::POS_SIDE, gmtl::NEG_-SIDE }

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

Variables

Constants

- const float gmtl::GMTL_EPSILON = 1.0e-6f
- const float gmtl::GMTL_MAT_EQUAL_EPSILON = 0.001f
- const float gmtl::GMTL_VEC_EQUAL_EPSILON = 0.0001f

11.11.1 Define Documentation

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

Definition at line 48 of file Defines.h.

11.12 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:

Classes

- class gmtl::Eigen

Namespaces

- namespace gmtl

  Meta programming classes.

11.13 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:

Classes

- class gmtl::EulerAngle< DATA_TYPE, ROTATION_ORDER >

  EulerAngle: Represents a group of euler angles.

Namespaces

- namespace gmtl

  Meta programming classes.

Typedefs

- typedef EulerAngle< float, XYZ > gmtl::EulerAngleXYZf
- typedef EulerAngle< double, XYZ > gmtl::EulerAngleXYZd
- typedef EulerAngle< float, ZYX > gmtl::EulerAngleZYXf
- typedef EulerAngle< double, ZYX > gmtl::EulerAngleZYXd
- typedef EulerAngle< float, ZXY > gmtl::EulerAngleZXYf
- typedef EulerAngle< double, ZXY > gmtl::EulerAngleZXYd

Functions

- const EulerAngle< float, XYZ > gmtl::EULERANGLE_IDENTITY_XYZF (0.0f, 0.0f, 0.0f)
- const EulerAngle< double, XYZ > gmtl::EULERANGLE_IDENTITY_XYZD (0.0, 0.0, 0.0)
- const EulerAngle< float, ZYX > gmtl::EULERANGLE_IDENTITY_ZYXF (0.0f, 0.0f, 0.0f)
- const EulerAngle< double, ZYX > gmtl::EULERANGLE_IDENTITY_ZYXD (0.0, 0.0, 0.0)
- const EulerAngle< float, ZXY > gmtl::EULERANGLE_IDENTITY_ZXYF (0.0f, 0.0f, 0.0f)
- const EulerAngle< double, ZXY > gmtl::EULERANGLE_IDENTITY_ZXYD (0.0, 0.0, 0.0)

11.14 EulerAngleOps.h File Reference

```
#include <gmtl/EulerAngle.h>
```

Include dependency graph for EulerAngleOps.h: This graph shows which files directly or indirectly include this file:

Namespaces

- namespace gmtl

  Meta programming classes.

Functions

EulerAngle Comparitors

- template<class DATA_TYPE , typename ROT_ORDER >
  bool gmtl::operator== (const EulerAngle< DATA_TYPE, ROT_ORDER > &e1, const EulerAngle< DATA_TYPE, ROT_ORDER > &e2)

  Compares 2 EulerAngles (component-wise) to see if they are exactly the same.

- template<class DATA_TYPE , typename ROT_ORDER >
  bool gmtl::operator!= (const EulerAngle< DATA_TYPE, ROT_ORDER > &e1, const EulerAngle< DATA_TYPE, ROT_ORDER > &e2)

  Compares e1 and e2 (component-wise) to see if they are NOT exactly the same.

- template<class DATA_TYPE , typename ROT_ORDER >
  bool gmtl::isEqual (const EulerAngle< DATA_TYPE, ROT_ORDER > &e1, const EulerAngle< DATA_TYPE, ROT_ORDER > &e2, const DATA_TYPE eps=0)

  Compares e1 and e2 (component-wise) to see if they are the same within a given tolerance.

11.15 Frustum.h File Reference

```
#include <gmtl/Defines.h>
#include <gmtl/Plane.h>
#include <gmtl/MatrixOps.h>
```

Include dependency graph for Frustum.h: This graph shows which files directly or indirectly include this file:

Classes

- class gmtl::Frustum< DATA_TYPE >

  This class defines a View Frustum Volume as a set of 6 planes.

Namespaces

- namespace gmtl

  Meta programming classes.

Typedefs

- typedef Frustum< float > gmtl::Frustumf
- typedef Frustum< double > gmtl::Frustumd

11.16 FrustumOps.h File Reference

```
#include <gmtl/Defines.h>
#include <gmtl/Frustum.h>
#include <gmtl/Math.h>
```

Include dependency graph for FrustumOps.h:

Namespaces

- namespace gmtl

  Meta programming classes.

Functions

- template<class DATA_TYPE >
  void gmtl::normalize (Frustum< DATA_TYPE > &f)

11.17 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

  Meta programming classes.

Functions

- void gmtl::GaussPointsFit (int iQuantity, const Point3 ∗akPoint, Point3 &rkCenter, Vec3 akAxis[3], float afExtent[3])
- bool gmtl::GaussPointsFit (int iQuantity, const Vec3 ∗akPoint, const bool ∗abValid, Vec3 &rkCenter, Vec3 akAxis[3], float afExtent[3])

11.18 Generate.h File Reference

```
#include <gmtl/Defines.h>
#include <gmtl/Util/Assert.h>
#include <gmtl/Util/StaticAssert.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/Util/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

  Meta programming classes.

Functions

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  void gmtl::setRow (Vec< DATA_TYPE, COLS > &dest, const Matrix<

DATA_TYPE, ROWS, COLS > &src, unsigned row)

Accesses a particular row in the matrix by copying the values in the row into the given vector.

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  Vec< DATA_TYPE, COLS > gmtl::makeRow (const Matrix< DATA_TYPE, ROWS, COLS > &src, unsigned row)

  Accesses a particular row in the matrix by creating a new vector containing the values in the given matrix.

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  void gmtl::setColumn (Vec< DATA_TYPE, ROWS > &dest, const Matrix< DATA_TYPE, ROWS, COLS > &src, unsigned col)

  Accesses a particular column in the matrix by copying the values in the column into the given vector.

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  Vec< DATA_TYPE, ROWS > gmtl::makeColumn (const Matrix< DATA_TYPE, ROWS, COLS > &src, unsigned col)

  Accesses a particular column in the matrix by creating a new vector containing the values in the given matrix.

Vec Generators

- template<typename DATA_TYPE >
  Vec< DATA_TYPE, 3 > gmtl::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 > gmtl::makeNormal (Vec< DATA_TYPE, SIZE > vec)

  create a normalized vector from the given vector.

- template<class DATA_TYPE >
  Vec< DATA_TYPE, 3 > gmtl::makeCross (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<typename VEC_TYPE , typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  VEC_TYPE & gmtl::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 > & gmtl::setPure (Quat< DATA_TYPE > &quat, const Vec< DATA_TYPE, 3 > &vec)
  Set pure quaternion.
- template<typename DATA_TYPE >
  Quat< DATA_TYPE > gmtl::makePure (const Vec< DATA_TYPE, 3 > &vec)
  create a pure quaternion
- template<typename DATA_TYPE >
  Quat< DATA_TYPE > gmtl::makeNormal (const Quat< DATA_TYPE > &quat)
  Normalize a quaternion.
- template<typename DATA_TYPE >
  Quat< DATA_TYPE > gmtl::makeConj (const Quat< DATA_TYPE > &quat)
  quaternion complex conjugate.
- template<typename DATA_TYPE >
  Quat< DATA_TYPE > gmtl::makeInvert (const Quat< DATA_TYPE > &quat)
  create quaternion from the inverse of another quaternion.
- template<typename DATA_TYPE >
  Quat< DATA_TYPE > & gmtl::set (Quat< DATA_TYPE > &result, const AxisAngle< DATA_TYPE > &axisAngle)
  Convert an AxisAngle to a Quat.
- template<typename DATA_TYPE >
  Quat< DATA_TYPE > & gmtl::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 > & gmtl::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 > & gmtl::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 > & gmtl::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 > & gmtl::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 > & gmtl::set (AxisAngle< DATA_TYPE > &axisAngle, Quat< DATA_TYPE > quat)
  Convert a rotation quaternion to an AxisAngle.

- template<typename DATA_TYPE >
  AxisAngle< DATA_TYPE > & gmtl::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 > gmtl::makeNormal (const AxisAngle< DATA_TYPE > &a)
  make the axis of an AxisAngle normalized

EulerAngle Generators

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS, typename ROT_ORDER >
  EulerAngle< DATA_TYPE, ROT_ORDER > & gmtl::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 > & gmtl::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 T >
  Matrix< T, 4, 4 > & gmtl::setFrustum (Matrix< T, 4, 4 > &result, T left, T top, T right, T bottom, T nr, T fr)
  Set an arbitrary projection matrix.
- template<typename T >
  Matrix< T, 4, 4 > & gmtl::setOrtho (Matrix< T, 4, 4 > &result, T left, T top, T right, T bottom, T nr, T fr)
  Set an orthographic projection matrix creates a transformation that produces a parallel projection matrix.
- template<typename T >
  Matrix< T, 4, 4 > & gmtl::setPerspective (Matrix< T, 4, 4 > &result, T fovy, T aspect, T nr, T fr)
  Set a symmetric perspective projection matrix.
- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS, unsigned SIZE, typename REP >
  Matrix< DATA_TYPE, ROWS, COLS > & gmtl::setTrans (Matrix< DATA_TYPE, ROWS, COLS > &result, const VecBase< DATA_TYPE, SIZE, REP > &trans)
  Set matrix translation from vec.
- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS, unsigned SIZE>
  Matrix< DATA_TYPE, ROWS, COLS > & gmtl::setScale (Matrix< DATA_TYPE, ROWS, COLS > &result, const Vec< DATA_TYPE, SIZE > &scale)
  Set the scale part of a matrix.
- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  Matrix< DATA_TYPE, ROWS, COLS > & gmtl::setScale (Matrix< DATA_TYPE, ROWS, COLS > &result, const DATA_TYPE scale)
  Sets the scale part of a matrix.
- template<typename MATRIX_TYPE , typename INPUT_TYPE >
  MATRIX_TYPE gmtl::makeScale (const INPUT_TYPE &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 > & gmtl::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, typename ROT_ORDER >
  Matrix< DATA_TYPE, ROWS, COLS > & gmtl::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>
  Matrix< DATA_TYPE, ROWS, COLS > & gmtl::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 > & gmtl::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>
  DATA_TYPE gmtl::makeYRot (const Matrix< DATA_TYPE, ROWS, COLS > &mat)

  Extracts the Y axis rotation information from the matrix.

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

  Extracts the X-axis rotation information from the matrix.

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

  Extracts the Z-axis rotation information from the matrix.

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  Matrix< DATA_TYPE, ROWS, COLS > & gmtl::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 > & gmtl::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 gmtl::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 > gmtl::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 > gmtl::makeInvert (const Matrix< DATA_TYPE, ROWS, COLS > &src)

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

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  Matrix< DATA_TYPE, ROWS, COLS > & gmtl::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 DATATYPE , typename POS_TYPE , typename ROT_TYPE , unsigned MATCOLS, unsigned MATROWS>
  Matrix< DATATYPE, MATROWS, MATCOLS > & gmtl::set (Matrix< DATATYPE, MATROWS, MATCOLS > &mat, const Coord< POS_TYPE, ROT_TYPE > &coord)

  *Convert a Coord to a Matrix Note: It is set directly, but this is equivalent to T*R where T is the translation matrix and R is the rotation matrix.*

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  Matrix< DATA_TYPE, ROWS, COLS > & gmtl::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 > & gmtl::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 > & gmtl::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.

Generic Generators (any type)

- template<typename TARGET_TYPE , typename SOURCE_TYPE >
  TARGET_TYPE gmtl::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 gmtl::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 gmtl::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 gmtl::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 gmtl::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 & gmtl::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.

11.19 gmtl.doxygen File Reference

11.20 gmtl.h File Reference

```
#include <gmtl/AABox.h>
#include <gmtl/AABoxOps.h>
#include <gmtl/AxisAngle.h>
#include <gmtl/AxisAngleOps.h>
#include <gmtl/Containment.h>
#include <gmtl/Coord.h>
#include <gmtl/CoordOps.h>
#include <gmtl/Defines.h>
#include <gmtl/EulerAngle.h>
#include <gmtl/EulerAngleOps.h>
#include <gmtl/Generate.h>
#include <gmtl/Intersection.h>
#include <gmtl/LineSeg.h>
#include <gmtl/LineSegOps.h>
#include <gmtl/Math.h>
#include <gmtl/Matrix.h>
```

```
#include <gmtl/MatrixOps.h>
#include <gmtl/Output.h>
#include <gmtl/Plane.h>
#include <gmtl/PlaneOps.h>
#include <gmtl/Point.h>
#include <gmtl/Quat.h>
#include <gmtl/QuatOps.h>
#include <gmtl/Ray.h>
#include <gmtl/Sphere.h>
#include <gmtl/SphereOps.h>
#include <gmtl/Tri.h>
#include <gmtl/TriOps.h>
#include <gmtl/VecBase.h>
#include <gmtl/Vec.h>
#include <gmtl/VecOps.h>
#include <gmtl/Version.h>
#include <gmtl/Xforms.h>
```

Include dependency graph for gmtl.h:

11.21 Helpers.h File Reference

```
#include <gmtl/Config.h>
```

Include dependency graph for Helpers.h: This graph shows which files directly or indirectly include this file:

Classes

- struct gmtl::helpers::ConstructorCounter

Namespaces

- namespace gmtl

  Meta programming classes.

- namespace gmtl::helpers

Functions

- ConstructorCounter * gmtl::helpers::VecCtrCounterInstance ()

11.22 Intersection.h File Reference

```
#include <algorithm>
#include <limits>
#include <gmtl/AABox.h>
#include <gmtl/Point.h>
#include <gmtl/Sphere.h>
#include <gmtl/Vec.h>
#include <gmtl/Plane.h>
#include <gmtl/VecOps.h>
#include <gmtl/Math.h>
#include <gmtl/Ray.h>
#include <gmtl/LineSeg.h>
#include <gmtl/Tri.h>
#include <gmtl/PlaneOps.h>
```

Include dependency graph for Intersection.h: This graph shows which files directly or indirectly include this file:

Namespaces

- namespace gmtl

  Meta programming classes.

Functions

- template< class DATA_TYPE >
  bool gmtl::intersect (const AABox< DATA_TYPE > &box1, const AABox< DATA_TYPE > &box2)

Tests if the given AABoxes intersect with each other.

- template<class DATA_TYPE >
  bool gmtl::intersect (const AABox< DATA_TYPE > &box, const Point< DATA_TYPE, 3 > &point)

  Tests if the given AABox and point intersect with each other.

- template<class DATA_TYPE >
  bool gmtl::intersect (const AABox< DATA_TYPE > &box1, const Vec< DATA_TYPE, 3 > &path1, const AABox< DATA_TYPE > &box2, const Vec< DATA_TYPE, 3 > &path2, DATA_TYPE &firstContact, DATA_TYPE &secondContact)

  Tests if the given AABoxes intersect if moved along the given paths.

- template<class DATA_TYPE >
  bool gmtl::intersectAABoxRay (const AABox< DATA_TYPE > &box, const Ray< DATA_TYPE > &ray, DATA_TYPE &tIn, DATA_TYPE &tOut)

  Given an axis-aligned bounding box and a ray (or subclass thereof), returns whether the ray intersects the box, and if so, `tIn` *and* `tOut` *are set to the parametric terms on the ray where the segment enters and exits the box respectively.*

- template<class DATA_TYPE >
  bool gmtl::intersect (const AABox< DATA_TYPE > &box, const LineSeg< DATA_TYPE > &seg, unsigned int &numHits, DATA_TYPE &tIn, DATA_TYPE &tOut)

  Given a line segment and an axis-aligned bounding box, returns whether the line intersects the box, and if so, `tIn` *and* `tOut` *are set to the parametric terms on the line segment where the segment enters and exits the box respectively.*

- template<class DATA_TYPE >
  bool gmtl::intersect (const LineSeg< DATA_TYPE > &seg, const AABox< DATA_TYPE > &box, unsigned int &numHits, DATA_TYPE &tIn, DATA_TYPE &tOut)

  Given a line segment and an axis-aligned bounding box, returns whether the line intersects the box, and if so, `tIn` *and* `tOut` *are set to the parametric terms on the line segment where the segment enters and exits the box respectively.*

- template<class DATA_TYPE >
  bool gmtl::intersect (const AABox< DATA_TYPE > &box, const Ray< DATA_TYPE > &ray, unsigned int &numHits, DATA_TYPE &tIn, DATA_TYPE &tOut)

  Given a ray and an axis-aligned bounding box, returns whether the ray intersects the box, and if so, `tIn` *and* `tOut` *are set to the parametric terms on the ray where it enters and exits the box respectively.*

- template<class DATA_TYPE >
bool gmtl::intersect (const Ray< DATA_TYPE > &ray, const AABox< DATA_TYPE > &box, unsigned int &numHits, DATA_TYPE &tIn, DATA_TYPE &tOut)

    Given a ray and an axis-aligned bounding box, returns whether the ray intersects the box, and if so, `tIn` *and* `tOut` *are set to the parametric terms on the ray where it enters and exits the box respectively.*

- template<class DATA_TYPE >
bool gmtl::intersect (const Sphere< DATA_TYPE > &sph1, const Vec< DATA_TYPE, 3 > &path1, const Sphere< DATA_TYPE > &sph2, const Vec< DATA_TYPE, 3 > &path2, DATA_TYPE &firstContact, DATA_TYPE &secondContact)

    Tests if the given Spheres intersect if moved along the given paths.

- template<class DATA_TYPE >
bool gmtl::intersect (const AABox< DATA_TYPE > &box, const Sphere< DATA_TYPE > &sph)

    Tests if the given AABox and Sphere intersect with each other.

- template<class DATA_TYPE >
bool gmtl::intersect (const Sphere< DATA_TYPE > &sph, const AABox< DATA_TYPE > &box)

    Tests if the given AABox and Sphere intersect with each other.

- template<class DATA_TYPE >
bool gmtl::intersect (const Sphere< DATA_TYPE > &sphere, const Point< DATA_TYPE, 3 > &point)

    intersect point/sphere.

- template<typename T >
bool gmtl::intersect (const Sphere< T > &sphere, const Ray< T > &ray, int &numhits, T &t0, T &t1)

    intersect ray/sphere-shell (not volume).

- template<typename T >
bool gmtl::intersect (const Sphere< T > &sphere, const LineSeg< T > &lineseg, int &numhits, T &t0, T &t1)

    intersect LineSeg/Sphere-shell (not volume).

- template<typename T >
bool gmtl::intersectVolume (const Sphere< T > &sphere, const LineSeg< T > &ray, int &numhits, T &t0, T &t1)

    intersect lineseg/sphere-volume.

- template< typename T >
  bool gmtl::intersectVolume (const Sphere< T > &sphere, const Ray< T > &ray, int &numhits, T &t0, T &t1)

  intersect ray/sphere-volume.

- template< class DATA_TYPE >
  bool gmtl::intersect (const Plane< DATA_TYPE > &plane, const Ray< DATA_TYPE > &ray, DATA_TYPE &t)

  Tests if the given plane and ray intersect with each other.

- template< class DATA_TYPE >
  bool gmtl::intersect (const Plane< DATA_TYPE > &plane, const LineSeg< DATA_TYPE > &seg, DATA_TYPE &t)

  Tests if the given plane and lineseg intersect with each other.

- template< class DATA_TYPE >
  bool gmtl::intersect (const Tri< DATA_TYPE > &tri, const Ray< DATA_TYPE > &ray, float &u, float &v, float &t)

  Tests if the given triangle and ray intersect with each other.

- template< class DATA_TYPE >
  bool gmtl::intersectDoubleSided (const Tri< DATA_TYPE > &tri, const Ray< DATA_TYPE > &ray, DATA_TYPE &u, DATA_TYPE &v, DATA_TYPE &t)

  Tests if the given triangle intersects with the given ray, from both sides.

- template< class DATA_TYPE >
  bool gmtl::intersect (const Tri< DATA_TYPE > &tri, const LineSeg< DATA_TYPE > &lineseg, DATA_TYPE &u, DATA_TYPE &v, DATA_TYPE &t)

  Tests if the given triangle and line segment intersect with each other.

11.23 LineSeg.h File Reference

```
#include <gmtl/Point.h>
#include <gmtl/Vec.h>
#include <gmtl/VecOps.h>
#include <gmtl/Ray.h>
```

Include dependency graph for LineSeg.h: This graph shows which files directly or indirectly include this file:

Classes

- class gmtl::LineSeg< DATA_TYPE >

  Describes a line segment.

Namespaces

- namespace gmtl

  Meta programming classes.

Typedefs

- typedef LineSeg< float > gmtl::LineSegf
- typedef LineSeg< double > gmtl::LineSegd

11.24 LineSegOps.h File Reference

`#include <gmtl/LineSeg.h>`

`#include <gmtl/RayOps.h>`

Include dependency graph for LineSegOps.h: This graph shows which files directly or indirectly include this file:

Namespaces

- namespace gmtl

  Meta programming classes.

Functions

- template<class DATA_TYPE >
  Point< DATA_TYPE, 3 > gmtl::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 gmtl::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 >
  DATA_TYPE gmtl::distanceSquared (const LineSeg< DATA_TYPE > &line-seg, const Point< DATA_TYPE, 3 > &pt)

  Computes the shortest distance from the line segment to the given point.

11.25 Math.h File Reference

```
#include <math.h>
#include <stdlib.h>
#include <gmtl/Defines.h>
#include <gmtl/Util/Assert.h>
#include <gmtl/Util/StaticAssert.h>
```

Include dependency graph for Math.h: This graph shows which files directly or indirectly include this file:

Classes

- struct gmtl::RotationOrderBase

  Base class for Rotation orders.

- struct gmtl::XYZ

  XYZ Rotation order.

- struct gmtl::ZYX

  ZYX Rotation order.

- struct gmtl::ZXY

  ZXY Rotation order.

Namespaces

- namespace gmtl

  Meta programming classes.

- namespace gmtl::Math

Functions

- template<class T >
  T gmtl::Math::clamp (T number, T lo, T hi)

  clamp "number" to a range between lo and hi

- template<class T >
  bool gmtl::Math::quadraticFormula (T &r1, T &r2, const T &a, const T &b, const T &c)

  Uses the quadratic formula to compute the 2 roots of the given 2nd degree polynomial in the form of Ax^2 + Bx + C.

C Math Abstraction

- template<typename T >
  T gmtl::Math::abs (T iValue)
- float gmtl::Math::abs (float iValue)
- double gmtl::Math::abs (double iValue)
- int gmtl::Math::abs (int iValue)
- long gmtl::Math::abs (long iValue)
- template<typename T >
  T gmtl::Math::ceil (T fValue)
- float gmtl::Math::ceil (float fValue)
- double gmtl::Math::ceil (double fValue)
- template<typename T >
  T gmtl::Math::floor (T fValue)
- float gmtl::Math::floor (float fValue)
- double gmtl::Math::floor (double fValue)
- template<typename T >
  int gmtl::Math::sign (T iValue)
- template<typename T >
  T gmtl::Math::zeroClamp (T value, T eps=static_cast< T >(0))

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

- template<typename T >
  T gmtl::Math::aCos (T fValue)
- float gmtl::Math::aCos (float fValue)
- double gmtl::Math::aCos (double fValue)
- template<typename T >
  T gmtl::Math::aSin (T fValue)
- float gmtl::Math::aSin (float fValue)
- double gmtl::Math::aSin (double fValue)
- template<typename T >
  T gmtl::Math::aTan (T fValue)
- double gmtl::Math::aTan (double fValue)
- float gmtl::Math::aTan (float fValue)

- template<typename T >
  T gmtl::Math::aTan2 (T fY, T fX)
- float gmtl::Math::aTan2 (float fY, float fX)
- double gmtl::Math::aTan2 (double fY, double fX)
- template<typename T >
  T gmtl::Math::cos (T fValue)
- float gmtl::Math::cos (float fValue)
- double gmtl::Math::cos (double fValue)
- template<typename T >
  T gmtl::Math::exp (T fValue)
- float gmtl::Math::exp (float fValue)
- double gmtl::Math::exp (double fValue)
- template<typename T >
  T gmtl::Math::log (T fValue)
- double gmtl::Math::log (double fValue)
- float gmtl::Math::log (float fValue)
- double gmtl::Math::pow (double fBase, double fExponent)
- float gmtl::Math::pow (float fBase, float fExponent)
- template<typename T >
  T gmtl::Math::sin (T fValue)
- double gmtl::Math::sin (double fValue)
- float gmtl::Math::sin (float fValue)
- template<typename T >
  T gmtl::Math::tan (T fValue)
- double gmtl::Math::tan (double fValue)
- float gmtl::Math::tan (float fValue)
- template<typename T >
  T gmtl::Math::sqr (T fValue)
- template<typename T >
  T gmtl::Math::sqrt (T fValue)
- double gmtl::Math::sqrt (double fValue)
- float gmtl::Math::fastInvSqrt (float x)

  Fast inverse square root.

- float gmtl::Math::fastInvSqrt2 (float x)
- float gmtl::Math::fastInvSqrt3 (float x)
- void gmtl::Math::seedRandom (unsigned int seed)

  Seeds the pseudorandom number generator with the given seed.

- float gmtl::Math::unitRandom ()

  get a random number between 0 and 1

- float gmtl::Math::rangeRandom (float x1, float x2)

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

- float gmtl::Math::deg2Rad (float fVal)
- double gmtl::Math::deg2Rad (double fVal)
- float gmtl::Math::rad2Deg (float fVal)

- double gmtl::Math::rad2Deg (double fVal)
- template<class T >
  bool gmtl::Math::isEqual (const T &a, const T &b, const T &tolerance)
  Is almost equal? test for equality within some tolerance...
- template<class T >
  T gmtl::Math::trunc (T val)
  cut off the digits after the decimal place
- template<class T >
  T gmtl::Math::round (T p)
  round to nearest integer
- template<class T >
  T gmtl::Math::Min (const T &x, const T &y)
  min returns the minimum of 2 values
- template<class T >
  T gmtl::Math::Min (const T &x, const T &y, const T &z)
  min returns the minimum of 3 values
- template<class T >
  T gmtl::Math::Min (const T &w, const T &x, const T &y, const T &z)
  min returns the minimum of 4 values
- template<class T >
  T gmtl::Math::Max (const T &x, const T &y)
  max returns the maximum of 2 values
- template<class T >
  T gmtl::Math::Max (const T &x, const T &y, const T &z)
  max returns the maximum of 3 values
- template<class T >
  T gmtl::Math::Max (const T &w, const T &x, const T &y, const T &z)
  max returns the maximum of 4 values
- template<class T >
  T gmtl::Math::factorial (T rhs)
  Compute the factorial.

Scalar type interpolation (for doubles, floats, etc...)

- template<class T , typename U >
  void gmtl::Math::lerp (T &result, const U &lerp, const T &a, const T &b)
  Linear Interpolation between number [a] and [b].

Variables

Mathematical constants

- const float gmtl::Math::TWO_PI = 6.28318530717958647692f
- const float gmtl::Math::PI = 3.14159265358979323846f
- const float gmtl::Math::PI_OVER_2 = 1.57079632679489661923f
- const float gmtl::Math::PI_OVER_4 = 0.78539816339744830962f

11.26 Matrix.h File Reference

```
#include <gmtl/Defines.h>
#include <gmtl/Math.h>
#include <gmtl/Util/Assert.h>
#include <gmtl/Util/StaticAssert.h>
```

Include dependency graph for Matrix.h: This graph shows which files directly or indirectly include this file:

Classes

- class gmtl::Matrix< DATA_TYPE, ROWS, COLS >

  State tracked NxM dimensional Matrix (ordered in memory by Column).

- class gmtl::Matrix< DATA_TYPE, ROWS, COLS >::RowAccessor

  Helper class for Matrix op[].

- class gmtl::Matrix< DATA_TYPE, ROWS, COLS >::ConstRowAccessor

  Helper class for Matrix op[] const.

Namespaces

- namespace gmtl

  Meta programming classes.

Typedefs

- typedef Matrix< float, 2, 2 > gmtl::Matrix22f
- typedef Matrix< double, 2, 2 > gmtl::Matrix22d

- typedef Matrix< float, 2, 3 > gmtl::Matrix23f
- typedef Matrix< double, 2, 3 > gmtl::Matrix23d
- typedef Matrix< float, 3, 3 > gmtl::Matrix33f
- typedef Matrix< double, 3, 3 > gmtl::Matrix33d
- typedef Matrix< float, 3, 4 > gmtl::Matrix34f
- typedef Matrix< double, 3, 4 > gmtl::Matrix34d
- typedef Matrix< float, 4, 4 > gmtl::Matrix44f
- typedef Matrix< double, 4, 4 > gmtl::Matrix44d

Functions

- int gmtl::combineMatrixStates (int state1, int state2)

  utility function for use by matrix operations.

Variables

- const Matrix22f gmtl::MAT_IDENTITY22F = Matrix22f()

  32bit floating point 2x2 identity matrix

- const Matrix22d gmtl::MAT_IDENTITY22D = Matrix22d()

  64bit floating point 2x2 identity matrix

- const Matrix23f gmtl::MAT_IDENTITY23F = Matrix23f()

  32bit floating point 2x2 identity matrix

- const Matrix23d gmtl::MAT_IDENTITY23D = Matrix23d()

  64bit floating point 2x2 identity matrix

- const Matrix33f gmtl::MAT_IDENTITY33F = Matrix33f()

  32bit floating point 3x3 identity matrix

- const Matrix33d gmtl::MAT_IDENTITY33D = Matrix33d()

  64bit floating point 3x3 identity matrix

- const Matrix34f gmtl::MAT_IDENTITY34F = Matrix34f()

  32bit floating point 3x4 identity matrix

- const Matrix34d gmtl::MAT_IDENTITY34D = Matrix34d()

  64bit floating point 3x4 identity matrix

- const Matrix44f gmtl::MAT_IDENTITY44F = Matrix44f()

32bit floating point 4x4 identity matrix

- const Matrix44d gmtl::MAT_IDENTITY44D = Matrix44d()

  64bit floating point 4x4 identity matrix

11.27 MatrixConvert.h File Reference

```
#include <boost/mpl/for_each.hpp>
#include <boost/mpl/range_c.hpp>
#include <boost/lambda/lambda.hpp>
#include <gmtl/Matrix.h>
```

Include dependency graph for MatrixConvert.h:

Namespaces

- namespace gmtl

  Meta programming classes.

Functions

- template< typename DATA_TYPE_OUT , typename DATA_TYPE_IN , unsigned ROWS, unsigned COLS >
  gmtl::Matrix< DATA_TYPE_OUT, ROWS, COLS > gmtl::convertTo (const gmtl::Matrix< DATA_TYPE_IN, ROWS, COLS > &in)

  Converts a matrix of one data type to another, such as gmtl::Matrix44f to gmtl::Matrix44d.

11.28 MatrixOps.h File Reference

```
#include <iostream>
#include <algorithm>
#include <gmtl/Matrix.h>
#include <gmtl/Math.h>
#include <gmtl/Vec.h>
```

`#include <gmtl/VecOps.h>`

`#include <gmtl/Util/Assert.h>`

Include dependency graph for MatrixOps.h: This graph shows which files directly or indirectly include this file:

Namespaces

- namespace gmtl

  Meta programming classes.

Functions

Matrix Operations

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  Matrix< DATA_TYPE, ROWS, COLS > & gmtl::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 > & gmtl::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 > & gmtl::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 > gmtl::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 > & gmtl::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 > & gmtl::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 > & gmtl::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 > & gmtl::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 > & gmtl::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 > & gmtl::mult (Matrix< DATA_TYPE, ROWS, COLS > &result, const Matrix< DATA_TYPE, ROWS, COLS > &mat, const DATA_TYPE &scalar)
  matrix scalar mult.

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
Matrix< DATA_TYPE, ROWS, COLS > & gmtl::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 > & gmtl::operator*= (Matrix< DATA_TYPE, ROWS, COLS > &result, const DATA_TYPE &scalar)
  matrix scalar mult (operator=).*

- template<typename DATA_TYPE , unsigned SIZE>
Matrix< DATA_TYPE, SIZE, SIZE > & gmtl::transpose (Matrix< DATA_TYPE, SIZE, SIZE > &result)
  matrix transpose in place.

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
Matrix< DATA_TYPE, ROWS, COLS > & gmtl::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 > & gmtl::invertTrans (Matrix< DATA_TYPE, ROWS, COLS > &result, const Matrix< DATA_TYPE, ROWS, COLS > &src)

  translational matrix inversion.

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
Matrix< DATA_TYPE, ROWS, COLS > & gmtl::invertOrthogonal (Matrix< DATA_TYPE, ROWS, COLS > &result, const Matrix< DATA_TYPE, ROWS, COLS > &src)

  orthogonal matrix inversion.

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
Matrix< DATA_TYPE, ROWS, COLS > & gmtl::invertAffine (Matrix< DATA_TYPE, ROWS, COLS > &result, const Matrix< DATA_TYPE, ROWS, COLS > &source)

  affine matrix inversion.

- template<typename DATA_TYPE , unsigned SIZE>
Matrix< DATA_TYPE, SIZE, SIZE > & gmtl::invertFull_GJ (Matrix< DATA_TYPE, SIZE, SIZE > &result, const Matrix< DATA_TYPE, SIZE, SIZE > &src)

  Full matrix inversion using Gauss-Jordan elimination.

- template<typename DATA_TYPE , unsigned SIZE>
Matrix< DATA_TYPE, SIZE, SIZE > & gmtl::invertFull_orig (Matrix< DATA_TYPE, SIZE, SIZE > &result, const Matrix< DATA_TYPE, SIZE, SIZE > &src)

  full matrix inversion.

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

  Invert method.

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

  Tests 2 matrices for equality.

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

  Tests 2 matrices for inequality.

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

  Tests 2 matrices for equality within a tolerance.

11.29 Meta.h File Reference

`#include <gmtl/Defines.h>`

Include dependency graph for Meta.h: This graph shows which files directly or indirectly include this file:

Classes

- struct gmtl::Type2Type< T >

  A lightweight identifier you can pass to overloaded functions to typefy them.

- struct gmtl::meta::AssignVecUnrolled< ELT, T >
- struct gmtl::meta::AssignVecUnrolled< 0, T >
- struct gmtl::meta::AssignArrayUnrolled< ELT, T >
- struct gmtl::meta::AssignArrayUnrolled< 0, T >

Namespaces

- namespace gmtl

  Meta programming classes.
- namespace gmtl::meta

Defines

- #define GMTL_STRINGIZE(X) GMTL_DO_STRINGIZE(X)
- #define GMTL_DO_STRINGIZE(X) #X
- #define GMTL_JOIN(X, Y) GMTL_DO_JOIN(X, Y)
- #define GMTL_DO_JOIN(X, Y) GMTL_DO_JOIN2(X,Y)
- #define GMTL_DO_JOIN2(X, Y) X##Y

Functions

- template<class T >
  void gmtl::ignore_unused_variable_warning (const T &)

11.29.1 Define Documentation

11.29.1.1 #define GMTL_DO_JOIN(*X*, *Y*) GMTL_DO_JOIN2(X,Y)

Definition at line 31 of file Meta.h.

11.29.1.2 #define GMTL_DO_JOIN2(*X*, *Y*) X##Y

Definition at line 32 of file Meta.h.

11.29.1.3 #define GMTL_DO_STRINGIZE(*X*) #X

Definition at line 20 of file Meta.h.

11.29.1.4 #define GMTL_JOIN(*X*, *Y*) GMTL_DO_JOIN(X, Y)

Definition at line 30 of file Meta.h.

11.29.1.5 #define GMTL_STRINGIZE(*X*) GMTL_DO_STRINGIZE(X)

Definition at line 19 of file Meta.h.

11.30 OOBox.h File Reference

`#include <gmtl/Vec3.h>`

`#include <gmtl/Point3.h>`

`#include <gmtl/matVecFuncs.h>`

Include dependency graph for OOBox.h:

Classes

- class gmtl::OOBox

Namespaces

- namespace gmtl

  Meta programming classes.

11.31 OpenSGConvert.h File Reference

GMTL/OpenSG conversion functions.

`#include <gmtl/Matrix.h>`

`#include <gmtl/Generate.h>`

`#include <OpenSG/OSGMatrix.h>`

Include dependency graph for OpenSGConvert.h:

Namespaces

- namespace gmtl

  Meta programming classes.

Functions

- Matrix44f & gmtl::set (Matrix44f &mat, const OSG::Matrix &osgMat)

  Converts an OpenSG matrix to a gmtl::Matrix.

- OSG::Matrix & gmtl::set (OSG::Matrix &osgMat, const Matrix44f &mat)

Converts a GMTL matrix to an OpenSG matrix.

11.31.1 Detailed Description

GMTL/OpenSG conversion functions. Methods to convert between GTML and OpenSG matrix classes.

Definition in file OpenSGConvert.h.

11.32 Output.h File Reference

```
#include <iostream>
#include <gmtl/Util/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 <gmtl/EulerAngle.h>
#include <gmtl/AABox.h>
#include <gmtl/Ray.h>
#include <gmtl/LineSeg.h>
#include <gmtl/Coord.h>
```

Include dependency graph for Output.h: This graph shows which files directly or indirectly include this file:

Classes

- struct gmtl::output::VecOutputter< DATA_TYPE, SIZE, REP >

  Outputters for vector types.

- struct gmtl::output::VecOutputter< DATA_TYPE, SIZE, gmtl::meta::DefaultVecTag >

Namespaces

- namespace gmtl

  Meta programming classes.

- namespace gmtl::output

Functions

Output Stream Operators

- template< typename DATA_TYPE , unsigned SIZE, typename REP >
  std::ostream & gmtl::operator<< (std::ostream &out, const VecBase< DATA_TYPE, SIZE, REP > &v)

  Outputs a string representation of the given VecBase type to the given output stream.

- template< class DATA_TYPE , typename ROTATION_ORDER >
  std::ostream & gmtl::operator<< (std::ostream &out, const EulerAngle< DATA_TYPE, ROTATION_ORDER > &e)

  Outputs a string representation of the given EulerAngle type to the given output stream.

- template< class DATA_TYPE , unsigned ROWS, unsigned COLS>
  std::ostream & gmtl::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 & gmtl::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 & gmtl::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 & gmtl::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 & gmtl::operator<< (std::ostream &out, const Sphere< DATA_TYPE > &s)

Outputs a string representation of the given Sphere to the given output stream.

- template<typename DATA_TYPE >
std::ostream & gmtl::operator<< (std::ostream &out, const AABox< DATA_TYPE > &b)

  Outputs a string representation of the given AABox to the given output stream.

- template<typename DATA_TYPE >
std::ostream & gmtl::operator<< (std::ostream &out, const Ray< DATA_TYPE > &b)

  Outputs a string representation of the given Ray to the given output stream.

- template<typename DATA_TYPE >
std::ostream & gmtl::operator<< (std::ostream &out, const LineSeg< DATA_TYPE > &b)

  Outputs a string representation of the given LineSeg to the given output stream.

- template<typename POS_TYPE , typename ROT_TYPE >
std::ostream & gmtl::operator<< (std::ostream &out, const Coord< POS_TYPE, ROT_TYPE > &c)

11.33 ParametricCurve.h File Reference

```
#include <gmtl/Matrix.h>
#include <gmtl/MatrixOps.h>
#include <gmtl/Vec.h>
#include <gmtl/VecOps.h>
```

Include dependency graph for ParametricCurve.h:

Classes

- class gmtl::ParametricCurve< DATA_TYPE, SIZE, ORDER >

  A base representation of a parametric curve with SIZE component using DATA_TYPE as the data type, ORDER as the order for each component.

- class gmtl::LinearCurve< DATA_TYPE, SIZE >

  A representation of a line with order set to 2.

- class gmtl::QuadraticCurve< DATA_TYPE, SIZE >

  A representation of a quadratic curve with order set to 3.

- class gmtl::CubicCurve< DATA_TYPE, SIZE >

  A representation of a cubic curve with order set to 4.

Namespaces

- namespace gmtl

  Meta programming classes.

Typedefs

- typedef LinearCurve< float, 1 > gmtl::LinearCurve1f
- typedef LinearCurve< float, 2 > gmtl::LinearCurve2f
- typedef LinearCurve< float, 3 > gmtl::LinearCurve3f
- typedef LinearCurve< double, 1 > gmtl::LinearCurve1d
- typedef LinearCurve< double, 2 > gmtl::LinearCurve2d
- typedef LinearCurve< double, 3 > gmtl::LinearCurve3d
- typedef QuadraticCurve< float, 1 > gmtl::QuadraticCurve1f
- typedef QuadraticCurve< float, 2 > gmtl::QuadraticCurve2f
- typedef QuadraticCurve< float, 3 > gmtl::QuadraticCurve3f
- typedef QuadraticCurve< double, 1 > gmtl::QuadraticCurve1d
- typedef QuadraticCurve< double, 2 > gmtl::QuadraticCurve2d
- typedef QuadraticCurve< double, 3 > gmtl::QuadraticCurve3d
- typedef CubicCurve< float, 1 > gmtl::CubicCurve1f
- typedef CubicCurve< float, 2 > gmtl::CubicCurve2f
- typedef CubicCurve< float, 3 > gmtl::CubicCurve3f
- typedef CubicCurve< double, 1 > gmtl::CubicCurve1d
- typedef CubicCurve< double, 2 > gmtl::CubicCurve2d
- typedef CubicCurve< double, 3 > gmtl::CubicCurve3d

11.34 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:

Classes

- class gmtl::Plane< DATA_TYPE >

  Plane: Defines a geometrical plane.

Namespaces

- namespace gmtl

  Meta programming classes.

Typedefs

- typedef Plane< float > gmtl::Planef
- typedef Plane< double > gmtl::Planed

11.35 PlaneOps.h File Reference

```
#include <gmtl/Defines.h>
#include <gmtl/Plane.h>
#include <gmtl/Math.h>
```

Include dependency graph for PlaneOps.h: This graph shows which files directly or indirectly include this file:

Namespaces

- namespace gmtl

  Meta programming classes.

Functions

Plane Operations

- template<class DATA_TYPE >
  DATA_TYPE gmtl::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 gmtl::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 gmtl::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 gmtl::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.

- template<class DATA_TYPE , unsigned SIZE>
  void gmtl::reflect (Point< DATA_TYPE, SIZE > &result, const Plane< DATA_TYPE > &plane, const Point< DATA_TYPE, SIZE > &point)
  Mirror the point by the plane.

Plane Comparitors

- template<class DATA_TYPE >
  bool gmtl::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 gmtl::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 gmtl::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.

11.36 Point.h File Reference

```
#include <gmtl/Defines.h>
#include <gmtl/VecBase.h>
```

Include dependency graph for Point.h: This graph shows which files directly or indirectly include this file:

Classes

- class gmtl::Point< DATA_TYPE, SIZE >

  Point Use points when you need to represent a position.

Namespaces

- namespace gmtl

  Meta programming classes.

Typedefs

- typedef Point< int, 2 > gmtl::Point2i
- typedef Point< float, 2 > gmtl::Point2f
- typedef Point< double, 2 > gmtl::Point2d
- typedef Point< int, 3 > gmtl::Point3i
- typedef Point< float, 3 > gmtl::Point3f
- typedef Point< double, 3 > gmtl::Point3d
- typedef Point< int, 4 > gmtl::Point4i
- typedef Point< float, 4 > gmtl::Point4f
- typedef Point< double, 4 > gmtl::Point4d

11.37 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:

Classes

- class gmtl::Quat< DATA_TYPE >

  Quat: Class to encapsulate quaternion behaviors.

Namespaces

- namespace gmtl

  Meta programming classes.

Typedefs

- typedef Quat< float > gmtl::Quatf
- typedef Quat< double > gmtl::Quatd

Functions

- const Quat< float > gmtl::QUAT_MULT_IDENTITYF (0.0f, 0.0f, 0.0f, 1.0f)
- const Quat< float > gmtl::QUAT_ADD_IDENTITYF (0.0f, 0.0f, 0.0f, 0.0f)
- const Quat< float > gmtl::QUAT_IDENTITYF (QUAT_MULT_IDENTITYF)
- const Quat< double > gmtl::QUAT_MULT_IDENTITYD (0.0, 0.0, 0.0, 1.0)
- const Quat< double > gmtl::QUAT_ADD_IDENTITYD (0.0, 0.0, 0.0, 0.0)
- const Quat< double > gmtl::QUAT_IDENTITYD (QUAT_MULT_-IDENTITYD)

11.38 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

  Meta programming classes.

Functions

Quat Operations

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

    product of two quaternions (quaternion product) Does quaternion multiplication.

- template<typename DATA_TYPE >
Quat< DATA_TYPE > & gmtl::operator*= (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q2)

    quaternion postmult

- template<typename DATA_TYPE >
Quat< DATA_TYPE > & gmtl::negate (Quat< DATA_TYPE > &result)

    Vector negation - negate each element in the quaternion vector.

- template<typename DATA_TYPE >
Quat< DATA_TYPE > gmtl::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 > & gmtl::mult (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q, DATA_TYPE s)

    vector scalar multiplication

- template<typename DATA_TYPE >
Quat< DATA_TYPE > gmtl::operator* (const Quat< DATA_TYPE > &q, DATA_TYPE s)

    vector scalar multiplication

- template<typename DATA_TYPE >
Quat< DATA_TYPE > & gmtl::operator*= (Quat< DATA_TYPE > &q, DATA_TYPE s)

    vector scalar multiplication

- template<typename DATA_TYPE >
Quat< DATA_TYPE > & gmtl::div (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q1, Quat< DATA_TYPE > q2)

    quotient of two quaternions

- template<typename DATA_TYPE >
  Quat< DATA_TYPE > gmtl::operator/ (const Quat< DATA_TYPE > &q1, Quat< DATA_TYPE > q2)
  quotient of two quaternions
- template<typename DATA_TYPE >
  Quat< DATA_TYPE > & gmtl::operator/= (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q2)
  quotient of two quaternions
- template<typename DATA_TYPE >
  Quat< DATA_TYPE > & gmtl::div (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q, DATA_TYPE s)
  quaternion vector scale
- template<typename DATA_TYPE >
  Quat< DATA_TYPE > gmtl::operator/ (const Quat< DATA_TYPE > &q, DATA_TYPE s)
  vector scalar division
- template<typename DATA_TYPE >
  Quat< DATA_TYPE > & gmtl::operator/= (const Quat< DATA_TYPE > &q, DATA_TYPE s)
  vector scalar division
- template<typename DATA_TYPE >
  Quat< DATA_TYPE > & gmtl::add (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)
  vector addition
- template<typename DATA_TYPE >
  Quat< DATA_TYPE > gmtl::operator+ (const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)
  vector addition
- template<typename DATA_TYPE >
  Quat< DATA_TYPE > & gmtl::operator+= (Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)
  vector addition
- template<typename DATA_TYPE >
  Quat< DATA_TYPE > & gmtl::sub (Quat< DATA_TYPE > &result, const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)
  vector subtraction
- template<typename DATA_TYPE >
  Quat< DATA_TYPE > gmtl::operator- (const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)

vector subtraction

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

  vector subtraction

- template<typename DATA_TYPE >
  DATA_TYPE gmtl::dot (const Quat< DATA_TYPE > &q1, const Quat< DATA_TYPE > &q2)

  vector dot product between two quaternions.

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

  set self to the normalized quaternion of self.

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

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

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

  quaternion complex conjugate.

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

  quaternion multiplicative inverse.

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

  complex exponentiation.

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

  complex logarithm

- template<typename DATA_TYPE >
  void gmtl::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 gmtl::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 > & gmtl::slerp (Quat< DATA_TYPE > &result, const DATA_TYPE t, const Quat< DATA_TYPE > &from, const Quat< DATA_TYPE > &to, bool adjustSign=true)

  spherical linear interpolation between two rotation quaternions.

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

  Compare two quaternions for equality.

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

  Compare two quaternions for not-equality.

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

  Compare two quaternions for geometric equivelence (with tolerance).

11.39 Ray.h File Reference

`#include <gmtl/Point.h>`

`#include <gmtl/Vec.h>`

`#include <gmtl/VecOps.h>`

Include dependency graph for Ray.h: This graph shows which files directly or indirectly include this file:

Classes

- class gmtl::Ray< DATA_TYPE >

  Describes a ray.

Namespaces

- namespace gmtl

  Meta programming classes.

Typedefs

- typedef Ray< float > gmtl::Rayf
- typedef Ray< double > gmtl::Rayd

11.40 RayOps.h File Reference

`#include <gmtl/Ray.h>`

Include dependency graph for RayOps.h: This graph shows which files directly or indirectly include this file:

Namespaces

- namespace gmtl

  Meta programming classes.

Functions

- template<class DATA_TYPE >
  bool gmtl::operator== (const Ray< DATA_TYPE > &ls1, const Ray< DATA_-TYPE > &ls2)

  Compare two line segments to see if they are EXACTLY the same.

- template<class DATA_TYPE >
  bool gmtl::operator!= (const Ray< DATA_TYPE > &ls1, const Ray< DATA_-TYPE > &ls2)

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

- template<class DATA_TYPE >
  bool gmtl::isEqual (const Ray< DATA_TYPE > &ls1, const Ray< DATA_-TYPE > &ls2, const DATA_TYPE &eps)

  Compare two line segments to see if the are the same within the given tolerance.

11.41 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:

Classes

- class gmtl::Sphere< DATA_TYPE >

  Describes a sphere in 3D space by its center point and its radius.

Namespaces

- namespace gmtl

  Meta programming classes.

Typedefs

- typedef Sphere< float > gmtl::Spheref
- typedef Sphere< double > gmtl::Sphered

11.42 SphereOps.h File Reference

```
#include <gmtl/Sphere.h>
#include <gmtl/VecOps.h>
#include <gmtl/Math.h>
```

Include dependency graph for SphereOps.h: This graph shows which files directly or indirectly include this file:

Namespaces

- namespace gmtl

    Meta programming classes.

Functions

Sphere Comparitors

- template<class DATA_TYPE >
  bool gmtl::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 gmtl::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 gmtl::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.

11.43 StaticAssert.h File Reference

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

Classes

- struct gmtl::CompileTimeError< true >

Namespaces

- namespace gmtl

  Meta programming classes.

Defines

- #define GMTL_STATIC_ASSERT(expr, msg) { gmtl::CompileTimeError<((expr) != 0)> ERROR_##msg; (void)ERROR_-##msg; }

  GMTL_STATIC_ASSERT macro.

11.43.1 Define Documentation

11.43.1.1 #define GMTL_STATIC_ASSERT(*expr, msg*) { gmtl::CompileTimeError<((expr) != 0)> ERROR_##msg; (void)ERROR_##msg; }

GMTL_STATIC_ASSERT macro.

This macro will evaluate a compile time integral or pointer expression; if the expression is zero, the macro will generate a message in the form of an undefined identifier.

Parameters

expr the expression to evaluate.

msg the message to display if expr is zero; msg cannot contain spaces!

Definition at line 32 of file StaticAssert.h.

11.44 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:

Classes

- class gmtl::Tri< DATA_TYPE >

  This class defines a triangle as a set of 3 points order in CCW fashion.

Namespaces

- namespace gmtl

  Meta programming classes.

Typedefs

- typedef Tri< float > gmtl::Trif
- typedef Tri< double > gmtl::Trid
- typedef Tri< int > gmtl::Trii

11.45 TriOps.h File Reference

`#include <gmtl/Tri.h>`

`#include <gmtl/Generate.h>`

`#include <gmtl/VecOps.h>`

Include dependency graph for TriOps.h: This graph shows which files directly or indirectly include this file:

Namespaces

- namespace gmtl

Meta programming classes.

Functions

Triangle Operations

- template<class DATA_TYPE >
  Point< DATA_TYPE, 3 > gmtl::center (const Tri< DATA_TYPE > &tri)
  Computes the point at the center of the given triangle.
- template<class DATA_TYPE >
  Vec< DATA_TYPE, 3 > gmtl::normal (const Tri< DATA_TYPE > &tri)
  Computes the normal for this triangle.

Triangle Comparitors

- template<class DATA_TYPE >
  bool gmtl::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 gmtl::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 gmtl::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.

11.46 Vec.h File Reference

```
#include <gmtl/Defines.h>
#include <gmtl/Config.h>
#include <gmtl/VecBase.h>
#include <gmtl/Util/StaticAssert.h>
```

Include dependency graph for Vec.h: This graph shows which files directly or indirectly include this file:

Classes

- class gmtl::Vec< DATA_TYPE, SIZE >

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

Namespaces

- namespace gmtl

  Meta programming classes.

Typedefs

- typedef Vec< int, 2 > gmtl::Vec2i
- typedef Vec< float, 2 > gmtl::Vec2f
- typedef Vec< double, 2 > gmtl::Vec2d
- typedef Vec< int, 3 > gmtl::Vec3i
- typedef Vec< float, 3 > gmtl::Vec3f
- typedef Vec< double, 3 > gmtl::Vec3d
- typedef Vec< int, 4 > gmtl::Vec4i
- typedef Vec< float, 4 > gmtl::Vec4f
- typedef Vec< double, 4 > gmtl::Vec4d

11.47 VecBase.h File Reference

```
#include <gmtl/Defines.h>
#include <gmtl/Util/Assert.h>
#include <gmtl/Util/StaticAssert.h>
#include <gmtl/Util/Meta.h>
#include <gmtl/Config.h>
#include <gmtl/Helpers.h>
```

Include dependency graph for VecBase.h: This graph shows which files directly or indirectly include this file:

Classes

- struct gmtl::meta::DefaultVecTag
- class gmtl::VecBase< DATA_TYPE, SIZE, REP >

  Base type for vector-like objects including Points and Vectors.

- class gmtl::VecBase< DATA_TYPE, SIZE, meta::DefaultVecTag >

  Specialized version of VecBase that is actually used for all user interaction with a traditional vector.

Namespaces

- namespace gmtl

  Meta programming classes.

- namespace gmtl::meta

11.48 VecExprMeta.h File Reference

```
#include <gmtl/Util/Meta.h>
#include <gmtl/VecOpsMeta.h>
#include <gmtl/VecBase.h>
```

Include dependency graph for VecExprMeta.h: This graph shows which files directly or indirectly include this file:

Classes

- struct gmtl::meta::ScalarArg< T >

  template to hold a scalar argument.

- struct gmtl::meta::ExprTraits< T >

  Traits class for expression template parameters.

- struct gmtl::meta::ExprTraits< VecBase< T, SIZE, ScalarArg< T > > >
- struct gmtl::meta::ExprTraits< VecBase< T, SIZE, DefaultVecTag > >
- struct gmtl::meta::VecBinaryExpr< EXP1_T, EXP2_T, OP >

  Binary vector expression.

- struct gmtl::meta::VecUnaryExpr< EXP1_T, OP >

Unary vector expression.

- struct gmtl::meta::VecPlusBinary
- struct gmtl::meta::VecMinusBinary
- struct gmtl::meta::VecMultBinary
- struct gmtl::meta::VecDivBinary
- struct gmtl::meta::VecNegUnary

  Negation of the values.

Namespaces

- namespace gmtl

  Meta programming classes.

- namespace gmtl::meta

Functions

- template< typename T >
  ScalarArg< T > gmtl::meta::makeScalarArg (T val)

11.49 VecOps.h File Reference

`#include <gmtl/Defines.h>`

`#include <gmtl/Math.h>`

`#include <gmtl/Vec.h>`

`#include <gmtl/VecOpsMeta.h>`

`#include <gmtl/VecExprMeta.h>`

Include dependency graph for VecOps.h: This graph shows which files directly or indirectly include this file:

Namespaces

- namespace gmtl

  Meta programming classes.

Functions

Vector/Point Operations

- template<typename T , unsigned SIZE, typename R1 >
VecBase< T, SIZE, meta::VecUnaryExpr< VecBase< T, SIZE, R1 >, meta::VecNegUnary > > gmtl::operator- (const VecBase< T, SIZE, R1 > &v1)
  Negates v1.

- template<class DATA_TYPE , unsigned SIZE, typename REP2 >
VecBase< DATA_TYPE, SIZE > & gmtl::operator+= (VecBase< DATA_-TYPE, SIZE > &v1, const VecBase< DATA_TYPE, SIZE, REP2 > &v2)
  Adds v2 to v1 and stores the result in v1.

- template<typename T , unsigned SIZE, typename R1 , typename R2 >
VecBase< T, SIZE, meta::VecBinaryExpr< VecBase< T, SIZE, R1 >, VecBase< T, SIZE, R2 >, meta::VecPlusBinary > > gmtl::operator+ (const VecBase< T, SIZE, R1 > &v1, const VecBase< T, SIZE, R2 > &v2)
  Adds v2 to v1 and returns the result.

- template<class DATA_TYPE , unsigned SIZE, typename REP2 >
VecBase< DATA_TYPE, SIZE > & gmtl::operator-= (VecBase< DATA_-TYPE, SIZE > &v1, const VecBase< DATA_TYPE, SIZE, REP2 > &v2)
  Subtracts v2 from v1 and stores the result in v1.

- template<typename T , unsigned SIZE, typename R1 , typename R2 >
VecBase< T, SIZE, meta::VecBinaryExpr< VecBase< T, SIZE, R1 >, VecBase< T, SIZE, R2 >, meta::VecMinusBinary > > gmtl::operator- (const VecBase< T, SIZE, R1 > &v1, const VecBase< T, SIZE, R2 > &v2)
  Subtracts v2 from v1 and returns the result.

- template<class DATA_TYPE , unsigned SIZE, class SCALAR_TYPE >
VecBase< DATA_TYPE, SIZE > & gmtl::operator*= (VecBase< DATA_-TYPE, SIZE > &v1, const SCALAR_TYPE &scalar)
  Multiplies v1 by a scalar value and stores the result in v1.

- template<typename T , unsigned SIZE, typename R1 >
VecBase< T, SIZE, meta::VecBinaryExpr< VecBase< T, SIZE, R1 >, VecBase< T, SIZE, meta::ScalarArg< T > >, meta::VecMultBinary > > gmtl::operator* (const VecBase< T, SIZE, R1 > &v1, const T scalar)
  Multiplies v1 by a scalar value and returns the result.

- template<typename T , unsigned SIZE, typename R1 >
VecBase< T, SIZE, meta::VecBinaryExpr< VecBase< T, SIZE, meta::ScalarArg< T > >, VecBase< T, SIZE, R1 >, meta::VecMultBinary > > gmtl::operator* (const T scalar, const VecBase< T, SIZE, R1 > &v1)

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

  Multiplies v1 by a scalar value and returns the result.

- template<typename T , unsigned SIZE, typename R1 >
  VecBase< T, SIZE, meta::VecBinaryExpr< VecBase< T, SIZE, R1 >, VecBase< T, SIZE, meta::ScalarArg< T > >, meta::VecDivBinary > > gmtl::operator/ (const VecBase< T, SIZE, R1 > &v1, const T scalar)

  Divides v1 by a scalar value and returns the result.

Vector Operations

- template<class DATA_TYPE , unsigned SIZE, typename REP1 , typename REP2 >
  DATA_TYPE gmtl::dot (const VecBase< DATA_TYPE, SIZE, REP1 > &v1, const VecBase< DATA_TYPE, SIZE, REP2 > &v2)

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

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

  Computes the length of the given vector.

- template<class DATA_TYPE , unsigned SIZE>
  DATA_TYPE gmtl::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 gmtl::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 gmtl::isNormalized (const Vec< DATA_TYPE, SIZE > &v1, const DATA_TYPE eps=(DATA_TYPE) 0.0001f)

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

- template<class DATA_TYPE >
  Vec< DATA_TYPE, 3 > & gmtl::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.

- template<class DATA_TYPE , unsigned SIZE>
  VecBase< DATA_TYPE, SIZE > & gmtl::reflect (VecBase< DATA_TYPE, SIZE > &result, const VecBase< DATA_TYPE, SIZE > &vec, const Vec< DATA_TYPE, SIZE > &normal)

Reflect a vector about a normal.

Vector Interpolation

- template<typename DATA_TYPE , unsigned SIZE>
  VecBase< DATA_TYPE, SIZE > & gmtl::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 gmtl::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.
- template<class DATA_TYPE , unsigned SIZE>
  bool gmtl::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 gmtl::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.

11.50 VecOpsMeta.h File Reference

`#include <gmtl/Util/Meta.h>`

Include dependency graph for VecOpsMeta.h: This graph shows which files directly or indirectly include this file:

Classes

- struct gmtl::meta::DotVecUnrolled< ELT, T1, T2 >
  meta class to unroll dot products.
- struct gmtl::meta::DotVecUnrolled< 0, T1, T2 >
  base cas for dot product unrolling.

- struct gmtl::meta::LenSqrVecUnrolled< ELT, T >

  meta class to unroll length squared operation.

- struct gmtl::meta::LenSqrVecUnrolled< 0, T >

  base cas for dot product unrolling.

- struct gmtl::meta::EqualVecUnrolled< ELT, VT >

  meta class to test vector equality.

- struct gmtl::meta::EqualVecUnrolled< 0, VT >

  base cas for dot product unrolling.

Namespaces

- namespace gmtl

  Meta programming classes.

- namespace gmtl::meta

11.51 Version.h File Reference

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

Namespaces

- namespace gmtl

  Meta programming classes.

Defines

- #define GMTL_VERSION_MAJOR 0

  This file contains two useful items.

- #define GMTL_VERSION_MINOR 6
- #define GMTL_VERSION_PATCH 1
- #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

Functions

- const char ∗ gmtl::getVersion ()

11.51.1 Define Documentation

11.51.1.1 #define GMTL_DOT(*a*, *b*) a ## . ## b

Definition at line 47 of file Version.h.

11.51.1.2 #define GMTL_GLUE(*a*, *b*) a ## b

Definition at line 39 of file Version.h.

11.51.1.3 #define GMTL_STR(*s*) # s

Definition at line 43 of file Version.h.

11.51.1.4 #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 93 of file Version.h.

11.51.1.5 #define GMTL_VERSION_MAJOR 0

This file contains two useful items.

1. The preprocessor friendly GMTL_VERSION "string". It is in the form <major><minor><patch> where each part has exactly 3 digits. 2. The C++ friendly variable, version, that contains the version as a string. It is in the form of <major>.<minor>.<patch> where each part has anywhere from 1 to 3 digits. 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 23 of file Version.h.

11.51.1.6 #define GMTL_VERSION_MAJOR_FILLED GMTL_-XZEROFILL(GMTL_XZEROFILL(GMTL_VERSION_MAJOR))

Definition at line 56 of file Version.h.

11.51.1.7 #define GMTL_VERSION_MINOR 6

Definition at line 24 of file Version.h.

11.51.1.8 #define GMTL_VERSION_MINOR_FILLED GMTL_-XZEROFILL(GMTL_XZEROFILL(GMTL_VERSION_MINOR))

Definition at line 66 of file Version.h.

11.51.1.9 #define GMTL_VERSION_PATCH 1

Definition at line 25 of file Version.h.

11.51.1.10 #define GMTL_VERSION_PATCH_FILLED GMTL_-XZEROFILL(GMTL_XZEROFILL(GMTL_VERSION_PATCH))

Definition at line 76 of file Version.h.

11.51.1.11 #define GMTL_VERSION_STRING

Value:

```
GMTL_XDOT( \
      GMTL_XDOT(GMTL_VERSION_MAJOR, GMTL_VERSION_MINOR), \
      GMTL_VERSION_PATCH \
   )
```

Definition at line 100 of file Version.h.

11.51.1.12 #define GMTL_XDOT(*a*, *b*) GMTL_DOT(a,b)

Definition at line 48 of file Version.h.

11.51.1.13 #define GMTL_XGLUE(*a*, *b*) GMTL_GLUE(a,b)

Definition at line 40 of file Version.h.

11.51.1.14 #define GMTL_XSTR(*s*) GMTL_STR(s)

Definition at line 44 of file Version.h.

11.51.1.15 #define GMTL_XZEROFILL(*a*) GMTL_ZEROFILL(a)

Definition at line 52 of file Version.h.

11.51.1.16 #define GMTL_ZEROFILL(*a*) 0 ## a

Definition at line 51 of file Version.h.

11.52 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/Ray.h>
#include <gmtl/LineSeg.h>
#include <gmtl/Util/StaticAssert.h>
```

Include dependency graph for Xforms.h: This graph shows which files directly or indirectly include this file:

Namespaces

- namespace gmtl

    Meta programming classes.

Functions

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  Ray< DATA_TYPE > & gmtl::xform (Ray< DATA_TYPE > &result, const Matrix< DATA_TYPE, ROWS, COLS > &matrix, const Ray< DATA_TYPE > &ray)

    transform ray by a matrix.

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  Ray< DATA_TYPE > gmtl::operator* (const Matrix< DATA_TYPE, ROWS, COLS > &matrix, const Ray< DATA_TYPE > &ray)

    *ray * a matrix multiplication of [m x k] matrix by a ray.*

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  Ray< DATA_TYPE > & gmtl::operator*= (Ray< DATA_TYPE > &ray, const Matrix< DATA_TYPE, ROWS, COLS > &matrix)

    *ray *= a matrix multiplication of [m x k] matrix by a ray.*

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  LineSeg< DATA_TYPE > & gmtl::xform (LineSeg< DATA_TYPE > &result, const Matrix< DATA_TYPE, ROWS, COLS > &matrix, const LineSeg< DATA_TYPE > &seg)

    transform seg by a matrix.

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  LineSeg< DATA_TYPE > gmtl::operator* (const Matrix< DATA_TYPE, ROWS, COLS > &matrix, const LineSeg< DATA_TYPE > &seg)

    *seg * a matrix multiplication of [m x k] matrix by a seg.*

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
LineSeg< DATA_TYPE > & gmtl::operator∗= (LineSeg< DATA_TYPE > &seg, const Matrix< DATA_TYPE, ROWS, COLS > &matrix)

  seg ∗= a matrix multiplication of [m x k] matrix by a seg.

Vector Transform (Quaternion)

- template<typename DATA_TYPE >
VecBase< DATA_TYPE, 3 > & gmtl::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 > gmtl::operator∗ (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 > gmtl::operator∗= (VecBase< DATA_TYPE, 3 > &vector, const Quat< DATA_TYPE > &rot)

  transform a vector by a rotation quaternion.

Vector Transform (Matrix)

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
Vec< DATA_TYPE, COLS > & gmtl::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 > gmtl::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 > & gmtl::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 > gmtl::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 > & gmtl::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 > gmtl::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 > & gmtl::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 > gmtl::operator∗ (const Matrix< DATA_TYPE, ROWS, COLS > &matrix, const Point< DATA_-TYPE, COLS_MINUS_ONE > &point)

  matrix ∗ partially specified point.

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  Point< DATA_TYPE, COLS > gmtl::operator∗ (const Point< DATA_TYPE, COLS > &point, const Matrix< DATA_TYPE, ROWS, COLS > &matrix)

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

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS>
  Point< DATA_TYPE, COLS > gmtl::operator∗= (Point< DATA_TYPE, COLS > &point, const Matrix< DATA_TYPE, ROWS, COLS > &matrix)

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

- template<typename DATA_TYPE , unsigned ROWS, unsigned COLS, unsigned COLS_-MINUS_ONE>
  Point< DATA_TYPE, COLS_MINUS_ONE > & gmtl::operator*= (Point< DATA_TYPE, COLS_MINUS_ONE > &point, const Matrix< DATA_-TYPE, ROWS, COLS > &matrix)

  *partial point *= a matrix 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]).*