Farouq Adepetus Computer Graphics Math Library

Generated by Doxygen 1.9.4

Chapter 1

Namespace Index

1.1 Namespace List

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Namespace Documentation

4.1 FAMath Namespace Reference

Has Vector2, Vector3, Vector4, and Matrix4x4 classes.

Classes

- class Matrix4x4

  A matrix class used for 4x4 matrices and their manipulations.
- class Quaternion

  A quaternion class used to represent rotations.
- class Vector2

  A vector class used for 2D vectors/points and their manipulations.
- class Vector3

  A vector class used for 3D vectors/points and their manipulations.
- class Vector4

  A vector class used for 4D vectors/points and their manipulations.

4.1.1 Detailed Description

Has Vector2, Vector3, Vector4, and Matrix4x4 classes.

FAMATH_H FILE AUTHOR: FAROUQ ADEPETU

Chapter 5

Class Documentation

5.1 FAMath::Matrix4x4 Class Reference

A matrix class used for 4x4 matrices and their manipulations.

`#include "FAMathLibrary.h"`

Public Member Functions

- bool isIdentity () const

  Returns true if the matrix is the identity matrix, false otherwise.
- Matrix4x4 transposed () const

  Returns this matrix, transposed about its diagonal.

Constructors

Constructors for class FAMath::Matrix4x4.

- Matrix4x4 ()

  Default Constructor.
- Matrix4x4 (const double *values)

  Overloaded Constructor.
- Matrix4x4 (double m11, double m12, double m13, double m14, double m21, double m22, double m23, double m24, double m31, double m32, double m33, double m34, double m41, double m42, double m43, double m44)

  Overloaded Constructor.

Getter

Getter for class FAMath::Matrix4x4.

- Vector4 column (const unsigned int &index) const

  Returns the elements of column index as a Vector4.
- Vector4 row (const unsigned int &index) const

  Returns the elements of row index as a Vector4.
- double * data ()

  Returns a pointer to the raw data of this matrix.
- const double * data () const

  Returns a constant pointer to the raw data of this matrix.

- const double * constData () const
  Returns a constant pointer to the raw data of this matrix.

Setter

Setter for class FAMath::Matrix4x4.

- void set (const unsigned int &row, const unsigned int &col, const double &value)
  Sets the element at [row][col] to the specified value.
- void setToIdentity ()
  Sets the matrix to the identity matrix.
- void fill (const double &value)
  Sets all of the elements of the matrix to value.
- void setColumn (const unsigned int &index, const Vector4 &value)
  Sets the elements of the column index to the components of the Vector4 object value.
- void setRow (const unsigned int &index, const Vector4 &value)
  Sets the elements of the row index to the components of the Vector4 object value.

Operator Overloading Member Functions

Operator Overloading Member Functions for class FAMath::Matrix4x4.

- const double & operator() (const unsigned int &row, const unsigned int &col) const
  Returns a constant reference to the element at position [row][col] in this matrix.
- double & operator() (const unsigned int &row, const unsigned int &col)
  Returns a reference to the element at position [row][col] in this matrix,.
- Matrix4x4 & operator+= (const Matrix4x4 &m)
  Add a 4x4 matrix with another 4x4 matrix through overloading operator +=.
- Matrix4x4 & operator-= (const Matrix4x4 &m)
  Subrtact a 4x4 matrix with another 4x4 matrix through overloading operator -=.
- Matrix4x4 & operator*= (const double &scalar)
  *Multiplying a 4x4 matrix with a scalar through overloading operator *=.*
- Matrix4x4 & operator*= (const Matrix4x4 &m)
  *Multiplies two 4x4 matrices through overloading operator *=.*
- Matrix4x4 & operator/= (const double &scalar)
  *Divides a 4x4 matrix with a scalar through overloading operator *=.*
- void rotate (double angle, const Vector3 &v)
  Multiplies this matrix by another that rotates angle degrees about vector v.
- void rotate (const double &angle, const double &x, const double &y, const double &z)
  Multiplies this matrix by another that rotates angle degrees about vector(x, y, z).
- void rotateUsingQuaternion (const Vector4 &v)
  Multiplies this matrix by another that rotates the coordinates using the quaternion matrix.
- void rotateUsingQuaternion (const double &angle, const Vector3 &v)
  Multiplies this matrix by another that rotates the coordinates using the quaternion matrix.
- void rotateUsingQuaternion (const double &angle, const double &x, const double &y, const double &z)
  Multiplies this matrix by another that rotates the coordinates using the quaternion matrix.
- void scale (const Vector3 &v)
  Multiplies this matrix by another that scales the coordinates by the components of vector v.
- void scale (const double &x, const double &y)
  Multiplies this matrix by another that scales the coordi nates by the components x and y.
- void scale (const double &x, const double &y, const double &z)
  Multiplies this matrix by another that scales the coordinates by the components x, y and z.
- void scale (const double &factor)
  Multiplies this matrix by another that scales the coordinates by the specified factor.
- void scale (const Vector3 &v, const double &factor)
  Multiplies this matrix by another that scales the coordinates by the specified factor along vector v.
- void translate (const Vector3 &v)
  Multiplies this matrix by another that translates coordinates by the components of v.
- void translate (const double &x, const double &y)

Multiplies this matrix by another that translates coordinates by the components of x and y.

- void translate (const double &x, const double &y, const double &z)

  Multiplies this matrix by another that translates coordinates by the components of x, y and z.
- void ortho (const double &left, const double &right, const double &bottom, const double &top, const double &near, const double &far)

  Multiplies this matrix by another that applies an othrographic projection for a window with lower left corner (left, bottom), upper right corner (right, top) and the specified near and far clipping planes.
- void perspective (const double &fov, const double &aspectRatio, const double &near, const double &far)

  Multiplies this matrix by another that applies a persepective projection.
- double determinant () const

  Returns the determinant of this matrix.

Friends

- Matrix4x4 operator+ (const Matrix4x4 &m1, const Matrix4x4 &m2)

  Adds two 4x4 matrices overloading operator +.
- Matrix4x4 operator- (const Matrix4x4 &m1, const Matrix4x4 &m2)

  Subtracts two 4x4 matrices overloading operator +.
- Matrix4x4 operator- (Matrix4x4 &m)

  Negates the 4x4 matrix through overloading operator -.
- Matrix4x4 operator∗ (const Matrix4x4 &m1, const double &scalar)

  Multiplying a 4x4 matrix with a scalar through overloading operator ∗.
- Matrix4x4 operator∗ (const double &scalar, const Matrix4x4 &m1)

  Multiplying a 4x4 matrix with a scalar through overloading operator ∗.
- Matrix4x4 operator∗ (const Matrix4x4 &m1, const Matrix4x4 &m2)

  Multiplies two 4x4 matrices through overloading operator ∗.
- Vector4 operator∗ (const Matrix4x4 &m, const Vector4 &vec)

  Multiplies a 4x4 matrix with a column vector(4x1) through overloading operator ∗.
- Vector4 operator∗ (const Vector4 &vec, const Matrix4x4 &m)

  Multiplies a 4x4 matrix with a row vector(1x4) through overloading operator ∗.
- Matrix4x4 operator/ (const Matrix4x4 &m1, const double &scalar)

  Divides a 4x4 matrix with a scalar through overloading operator.
- bool operator== (const Matrix4x4 &m1, const Matrix4x4 &m2)

  Returns true if m1 is identical to m2, false otherwise.
- bool operator!= (const Matrix4x4 &m1, const Matrix4x4 &m2)

  Returns false if m1 is identical to m2, true otherwise.
- Matrix4x4 inverse (const Matrix4x4 &m)

  Returns the inverse of the matrix m. If the matrix can't be inverted then the identity matrix is returned.
- void **print** (const Matrix4x4 &m)

5.1.1 Detailed Description

A matrix class used for 4x4 matrices and their manipulations.

The datatype for the matrix is double

The 4x4 matrix is treated as a row-major matrix
the constructors and other functions take in the data as row major format

Internally the data is stored in column-major order

Definition at line 804 of file FAMathLibrary.h.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Matrix4x4() [1/3]

```
FAMath::Matrix4x4::Matrix4x4 ( )
```

Default Constructor.

Constructs an identity matrix.

5.1.2.2 Matrix4x4() [2/3]

```
FAMath::Matrix4x4::Matrix4x4 (
            const double * values )
```

Overloaded Constructor.

Constructors a 4x4 matrix from the given std::array.

5.1.2.3 Matrix4x4() [3/3]

```
FAMath::Matrix4x4::Matrix4x4 (
            double m11,
            double m12,
            double m13,
            double m14,
            double m21,
            double m22,
            double m23,
            double m24,
            double m31,
            double m32,
            double m33,
            double m34,
            double m41,
            double m42,
            double m43,
            double m44 )
```

Overloaded Constructor.

Constructs a 4x4 matrix from the specified 16 elements.
The elements are specified in row-major order.

5.1.3 Member Function Documentation

5.1.3.1 column()

```
Vector4 FAMath::Matrix4x4::column (
            const unsigned int & index ) const
```

Returns the elements of column index as a Vector4.

Throws an std::out_of_range if given index > 3.

5.1.3.2 constData()

```
const double * FAMath::Matrix4x4::constData ( ) const
```

Returns a constant pointer to the raw data of this matrix.

The raw data is stored in column-major format.

5.1.3.3 data() [1/2]

```
double * FAMath::Matrix4x4::data ( )
```

Returns a pointer to the raw data of this matrix.

The raw data is stored in column-major format.

5.1.3.4 data() [2/2]

```
const double * FAMath::Matrix4x4::data ( ) const
```

Returns a constant pointer to the raw data of this matrix.

The raw data is stored in column-major format.

5.1.3.5 determinant()

```
double FAMath::Matrix4x4::determinant ( ) const
```

Returns the determinant of this matrix.

5.1.3.6 fill()

```
void FAMath::Matrix4x4::fill (
            const double & value )
```

Sets all of the elements of the matrix to value.

5.1.3.7 isIdentity()

```
bool FAMath::Matrix4x4::isIdentity ( ) const
```

Returns true if the matrix is the identity matrix, false otherwise.

5.1.3.8 operator()() [1/2]

```
double & FAMath::Matrix4x4::operator() (
            const unsigned int & row,
            const unsigned int & col )
```

Returns a reference to the element at position [row][col] in this matrix,.

Throws an std::out_of_range if row or col $>$ 3.

5.1.3.9 operator()() [2/2]

```
const double & FAMath::Matrix4x4::operator() (
            const unsigned int & row,
            const unsigned int & col ) const
```

Returns a constant reference to the element at position [row][col] in this matrix.

Throws an std::out_of_range if row or col $>$ 3.

5.1.3.10 operator$*$=() [1/2]

```
Matrix4x4 & FAMath::Matrix4x4::operator*= (
            const double & scalar )
```

Multiplying a 4x4 matrix with a scalar through overloading operator $*$=.

Returns

a reference to the current Matrix4x4 object with the result of the current Matrtix4x4 object $*$ scalar.

5.1.3.11 operator$*$=() [2/2]

```
Matrix4x4 & FAMath::Matrix4x4::operator*= (
            const Matrix4x4 & m )
```

Multiplies two 4x4 matrices through overloading operator $*$=.

Returns

a reference to the current Matrix4x4 object with the result of the current Matrtix4x4 object $*$ scalar.

5.1.3.12 operator+=()

```
Matrix4x4 & FAMath::Matrix4x4::operator+= (
            const Matrix4x4 & m )
```

Add a 4x4 matrix with another 4x4 matrix through overloading operator +=.

Returns

a reference to the current Matrix4x4 object with the result of the current Matrix4x4 object + m.

5.1.3.13 operator-=()

```
Matrix4x4 & FAMath::Matrix4x4::operator-= (
            const Matrix4x4 & m )
```

Subrtact a 4x4 matrix with another 4x4 matrix through overloading operator -=.

Returns

a reference to the current Matrix4x4 object with the result of the current Matrix4x4 object - m.

5.1.3.14 operator/=()

```
Matrix4x4 & FAMath::Matrix4x4::operator/= (
            const double & scalar )
```

Divides a 4x4 matrix with a scalar through overloading operator $*$=.

Returns

a reference to the current Matrix4x4 object with the result of the current Matrtix4x4 object / scalar.

Throws an invalid_argument exception if scalar is 0.0.

5.1.3.15 ortho()

```
void FAMath::Matrix4x4::ortho (
            const double & left,
            const double & right,
            const double & bottom,
            const double & top,
            const double & near,
            const double & far )
```

Multiplies this matrix by another that applies an othrographic projection for a window with lower left corner (left, bottom), upper right corner (right, top) and the specified near and far clipping planes.

5.1.3.16 perspective()

```
void FAMath::Matrix4x4::perspective (
            const double & fov,
            const double & aspectRatio,
            const double & near,
            const double & far )
```

Multiplies this matrix by another that applies a persepective projection.

The fov is the vertical angle in degrees. Aspect ratio is the aspect ratio of your window. Near and far are the distances from the viewer to the corresponding planes.

5.1.3.17 rotate() [1/2]

```
void FAMath::Matrix4x4::rotate (
            const double & angle,
            const double & x,
            const double & y,
            const double & z )
```

Multiplies this matrix by another that rotates angle degrees about vector(x, y, z).

5.1.3.18 rotate() [2/2]

```
void FAMath::Matrix4x4::rotate (
            double angle,
            const Vector3 & v )
```

Multiplies this matrix by another that rotates angle degrees about vector v.

5.1.3.19 rotateUsingQuaternion() [1/3]

```
void FAMath::Matrix4x4::rotateUsingQuaternion (
            const double & angle,
            const double & x,
            const double & y,
            const double & z )
```

Multiplies this matrix by another that rotates the coordinates using the quaternion matrix.

(x, y, z) is the axis to rotate around normalized.

The angle should be given in degrees.

5.1.3.20 rotateUsingQuaternion() [2/3]

```
void FAMath::Matrix4x4::rotateUsingQuaternion (
            const double & angle,
            const Vector3 & v )
```

Multiplies this matrix by another that rotates the coordinates using the quaternion matrix.

v is the axis to rotate around normalized.

The angle should be given in degrees.

5.1.3.21 rotateUsingQuaternion() [3/3]

```
void FAMath::Matrix4x4::rotateUsingQuaternion (
            const Vector4 & v )
```

Multiplies this matrix by another that rotates the coordinates using the quaternion matrix.

(x, y, z) in v is the axis you want to rotate around normalized. The w value is the angle in degrees.

5.1.3.22 row()

```
Vector4 FAMath::Matrix4x4::row (
            const unsigned int & index ) const
```

Returns the elements of row index as a Vector4.

Throws an std::out_of_range if given index $>$ 3

5.1.3.23 scale() [1/5]

```
void FAMath::Matrix4x4::scale (
            const double & factor )
```

Multiplies this matrix by another that scales the coordinates by the specified factor.

5.1.3.24 scale() [2/5]

```
void FAMath::Matrix4x4::scale (
            const double & x,
            const double & y )
```

Multiplies this matrix by another that scales the coordi nates by the components x and y.

5.1.3.25 scale() [3/5]

```
void FAMath::Matrix4x4::scale (
            const double & x,
            const double & y,
            const double & z )
```

Multiplies this matrix by another that scales the coordinates by the components x, y and z.

5.1.3.26 scale() [4/5]

```
void FAMath::Matrix4x4::scale (
            const Vector3 & v )
```

Multiplies this matrix by another that scales the coordinates by the components of vector v.

5.1.3.27 scale() [5/5]

```
void FAMath::Matrix4x4::scale (
            const Vector3 & v,
            const double & factor )
```

Multiplies this matrix by another that scales the coordinates by the specified factor along vector v.

5.1.3.28 set()

```
void FAMath::Matrix4x4::set (
            const unsigned int & row,
            const unsigned int & col,
            const double & value )
```

Sets the element at [row][col] to the specified value.

Throws an std::out_of_range if row or col $>$ 3.

5.1.3.29 setColumn()

```
void FAMath::Matrix4x4::setColumn (
            const unsigned int & index,
            const Vector4 & value )
```

Sets the elements of the column index to the components of the Vector4 object value.

Throws a std::out_of_range exception if index $>$ 3.

5.1.3.30 setRow()

```
void FAMath::Matrix4x4::setRow (
            const unsigned int & index,
            const Vector4 & value )
```

Sets the elements of the row index to the components of the Vector4 object value.

Throws a std::out_of_range exception if index $>$ 3.

5.1.3.31 setToIdentity()

```
void FAMath::Matrix4x4::setToIdentity ( )
```

Sets the matrix to the identity matrix.

5.1.3.32 translate() [1/3]

```
void FAMath::Matrix4x4::translate (
            const double & x,
            const double & y )
```

Multiplies this matrix by another that translates coordinates by the components of x and y.

5.1.3.33 translate() [2/3]

```
void FAMath::Matrix4x4::translate (
            const double & x,
            const double & y,
            const double & z )
```

Multiplies this matrix by another that translates coordinates by the components of x, y and z.

5.1.3.34 translate() [3/3]

```
void FAMath::Matrix4x4::translate (
            const Vector3 & v )
```

Multiplies this matrix by another that translates coordinates by the components of v.

5.1.3.35 transposed()

```
Matrix4x4 FAMath::Matrix4x4::transposed ( ) const
```

Returns this matrix, transposed about its diagonal.

5.1.4 Friends And Related Function Documentation

5.1.4.1 inverse

```
Matrix4x4 inverse (
            const Matrix4x4 & m )  [friend]
```

Returns the inverse of the matrix m. If the matrix can't be inverted then the identity matrix is returned.

5.1.4.2 operator"!=

```
bool operator!= (
            const Matrix4x4 & m1,
            const Matrix4x4 & m2 )  [friend]
```

Returns false if m1 is identical to m2, true otherwise.

5.1.4.3 operator∗ [1/5]

```
Matrix4x4 operator* (
            const double & scalar,
            const Matrix4x4 & m1 )  [friend]
```

Multiplying a 4x4 matrix with a scalar through overloading operator ∗.

Returns

a Matrix4x4 object with the result of scalar ∗ m1.

5.1.4.4 operator* [2/5]

```
Vector4 operator* (
            const Matrix4x4 & m,
            const Vector4 & vec )  [friend]
```

Multiplies a 4x4 matrix with a column vector(4x1) through overloading operator *.

Returns

a Vector4 object with the result of m * vec.

5.1.4.5 operator* [3/5]

```
Matrix4x4 operator* (
            const Matrix4x4 & m1,
            const double & scalar )  [friend]
```

Multiplying a 4x4 matrix with a scalar through overloading operator *.

Returns

a Matrix4x4 object with the result of m1 * scalar.

5.1.4.6 operator* [4/5]

```
Matrix4x4 operator* (
            const Matrix4x4 & m1,
            const Matrix4x4 & m2 )  [friend]
```

Multiplies two 4x4 matrices through overloading operator *.

Returns

a Matrix4x4 object with the result of m1 * m2.

5.1.4.7 operator* [5/5]

```
Vector4 operator* (
            const Vector4 & vec,
            const Matrix4x4 & m )  [friend]
```

Multiplies a 4x4 matrix with a row vector(1x4) through overloading operator *.

Returns

a Vector4 object with the result of vec * m.

5.1.4.8 operator+

```
Matrix4x4 operator+ (
            const Matrix4x4 & m1,
            const Matrix4x4 & m2 )  [friend]
```

Adds two 4x4 matrices overloading operator +.

Returns

a Matrix4x4 object with the result of m1 + m2.

5.1.4.9 operator- [1/2]

```
Matrix4x4 operator- (
            const Matrix4x4 & m1,
            const Matrix4x4 & m2 )  [friend]
```

Subtracts two 4x4 matrices overloading operator +.

Returns

a Matrix4x4 object with the result of m1 - m2.

5.1.4.10 operator- [2/2]

```
Matrix4x4 operator- (
            Matrix4x4 & m )  [friend]
```

Negates the 4x4 matrix through overloading operator -.

Returns

a Matrix4x4 object with the result of -m.

5.1.4.11 operator/

```
Matrix4x4 operator/ (
            const Matrix4x4 & m1,
            const double & scalar )  [friend]
```

Divides a 4x4 matrix with a scalar through overloading operator.

Returns

a Matrix4x4 object with the result of m1 / scalar.

Throws an invalid_argument exception if scalar is 0.0.

5.1.4.12 operator==

```
bool operator== (
            const Matrix4x4 & m1,
            const Matrix4x4 & m2 )  [friend]
```

Returns true if m1 is identical to m2, false otherwise.

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

- C:/Users/Work/Desktop/Math/FAMath/FAMath/FAMathLibrary.h

5.2 FAMath::Quaternion Class Reference

A quaternion class used to represent rotations.

```
#include "FAMathLibrary.h"
```

Public Member Functions

- bool isZeroQuaternion () const

  Returns true if the quaternion scalar value is equal to 0 and all the components of the 3D vector is equal to zero, false otherwise.
- Quaternion & operator*= (const double &k)

  Multiplies this quaternion by a scalar and returns a reference to this quaternion.
- Quaternion & operator*= (const Quaternion &q)

  Multiplies this quaternion by q and returns a reference to this quaternion.
- Quaternion & operator+= (const Quaternion &q)

  Adds this quaternion to the given quaternion q and returns a reference to this quaternion.
- Quaternion & operator-= (const Quaternion &q)

  Subtracts this quaternion from the given quaternion q and returns a reference to this quaternion.
- Matrix4x4 toRotationMatrix ()

  Creates a rotation matrix from this quaternion. Normalize the quaternion before using this function.

Constructors

Constructors for class FAMath::Quaternion

- Quaternion ()

  Default Constructor.
- Quaternion (const double &w, const Vector3 &v)

  Overloaded Constructor.
- Quaternion (const double &w, const double &x, const double &y, const double &z)

  Overloaded Constructor.

Getters and Setters

Getters and Setters for class FAMath::Quaternion

- double scalar () const

  Returns the scalar component of the quaternion.
- Vector3 vector () const

Returns the 3D vector component of the quaternion.

- double x () const

  Returns the x component of the quaternion's 3D vector.
- double y () const

  Returns the y component of the quaternion's 3D vector.
- double z () const

  Returns the z component of the quaternion's 3D vector.
- void setQuaternion (const double &w, const Vector3 &v)

  Sets the quaternion values. The scalar value equals to w and the 3D vector equal to v.
- void setQuaternion (const double &w, double &x, double &y, double &z)

  Sets the quaternion values. The scalar value equals to w and the 3D vector equals to (x, y, z).
- void setScalar (const double &w)

  Sets the scalar value in the quaternion to w.
- void setVector (const Vector3 &v)

  Sets the 3D vector in the quaternion to v.
- void setVector (const double &x, const double &y, const double &z)

  Sets the 3D vector in the quaternion to (x, y, z).
- void setX (const double &x)

  Sets the x component of the 3D vector in the quaternion to the given x.
- void setY (const double &y)

  Sets the y component of the 3D vector in the quaternion to the given y.
- void setZ (const double &z)

  Sets the z component of the 3D vector in the quaternion to the given z.

Friends

- Quaternion operator- (const Quaternion &q)

  Negates the scalar value and each componenet in the 3D vector of q.
- double length (const Quaternion &q)

  Returns the magnitude of the quaternion.
- Quaternion normalize (const Quaternion &q)

  Normalizes the quaternion q.
- Quaternion conjugate (const Quaternion &q)

  Returns the conjugate of quaternion q.
- Quaternion inverse (const Quaternion &q)

  Returns the inverse of quaternion q. If the quaternion q is the zero quaternion, then the zero quaternion is returned.
- Quaternion operator∗ (const Quaternion &q1, const Quaternion &q2)

  Returns the product of q1 and q2 using quaternion multiplication.
- Quaternion operator∗ (const Quaternion &q, const double &k)

  Returns a Quaternion object that has the result of q ∗ k.
- Quaternion operator∗ (const double &k, const Quaternion &q)

  Returns a Quaternion object that has the result of k ∗ q.
- Vector3 operator∗ (const Quaternion &q, const Vector3 &v)

  Rotates 3D vector v by quaternion q to produce a new 3D vector.
- Quaternion operator+ (const Quaternion &q1, const Quaternion &q2)

  Returns a Quaternion object that is the sum of q1 and q2.
- Quaternion operator- (const Quaternion &q1, const Quaternion &q2)

  Returns a Quaternion object that has the result of q1 - q2;.
- bool operator== (const Quaternion &q1, const Quaternion &q2)

  Return true if q1 and q2 are equal, false otherwise.
- bool operator!= (const Quaternion &q1, const Quaternion &q2)

  Return true if q1 and q2 aren't equal, false otherwise.
- double dotProduct (const Quaternion &q1, const Quaternion &q2)

  Returns the dot product between q1 and q2.
- Quaternion slerp (const Quaternion &q1, const Quaternion &q2, const double &t)

  Spherically Interpolates between rotations q1 and q2.
- void **print** (const Quaternion &q)

5.2.1 Detailed Description

A quaternion class used to represent rotations.

A quaternion consists of scalar to represent rotation angle and a 3D vector to represent an axis. The classes uses a double to respresent the scalar and a 3D vector to represent the axis.

Definition at line 1153 of file FAMathLibrary.h.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Quaternion() [1/3]

```
FAMath::Quaternion::Quaternion ( )
```

Default Constructor.

Creates a new Quaternion with scalar value = 1 and the 3D vector = (0, 0, 0).

5.2.2.2 Quaternion() [2/3]

```
FAMath::Quaternion::Quaternion (
            const double & w,
            const Vector3 & v )
```

Overloaded Constructor.

Creates a new Quaternion with scalar value equal to w and the 3D vector equal to v.

5.2.2.3 Quaternion() [3/3]

```
FAMath::Quaternion::Quaternion (
            const double & w,
            const double & x,
            const double & y,
            const double & z )
```

Overloaded Constructor.

Creates a new Quaternion with scalar value equal to w and the 3D vector equal to (x, y ,z).

5.2.3 Member Function Documentation

5.2.3.1 isZeroQuaternion()

```
bool FAMath::Quaternion::isZeroQuaternion ( ) const
```

Returns true if the quaternion scalar value is equal to 0 and all the components of the 3D vector is equal to zero, false otherwise.

5.2.3.2 operator*=() [1/2]

```
Quaternion & FAMath::Quaternion::operator*= (
            const double & k )
```

Multiplies this quaternion by a scalar and returns a reference to this quaternion.

5.2.3.3 operator*=() [2/2]

```
Quaternion & FAMath::Quaternion::operator*= (
            const Quaternion & q )
```

Multiplies this quaternion by q and returns a reference to this quaternion.

5.2.3.4 operator+=()

```
Quaternion & FAMath::Quaternion::operator+= (
            const Quaternion & q )
```

Adds this quaternion to the given quaternion q and returns a reference to this quaternion.

5.2.3.5 operator-=()

```
Quaternion & FAMath::Quaternion::operator-= (
            const Quaternion & q )
```

Subtracts this quaternion from the given quaternion q and returns a reference to this quaternion.

5.2.3.6 scalar()

```
double FAMath::Quaternion::scalar ( ) const
```

Returns the scalar component of the quaternion.

5.2.3.7 setQuaternion() [1/2]

```
void FAMath::Quaternion::setQuaternion (
            const double & w,
            const Vector3 & v )
```

Sets the quaternion values. The scalar value equals to w and the 3D vector equal to v.

5.2.3.8 setQuaternion() [2/2]

```
void FAMath::Quaternion::setQuaternion (
            const double & w,
            double & x,
            double & y,
            double & z )
```

Sets the quaternion values. The scalar value equals to w and the 3D vector equals to (x, y, z).

5.2.3.9 setScalar()

```
void FAMath::Quaternion::setScalar (
            const double & w )
```

Sets the scalar value in the quaternion to w.

5.2.3.10 setVector() [1/2]

```
void FAMath::Quaternion::setVector (
            const double & x,
            const double & y,
            const double & z )
```

Sets the 3D vector in the quaternion to (x, y, z).

5.2.3.11 setVector() [2/2]

```
void FAMath::Quaternion::setVector (
            const Vector3 & v )
```

Sets the 3D vector in the quaternion to v.

5.2.3.12 setX()

```
void FAMath::Quaternion::setX (
            const double & x )
```

Sets the x component of the 3D vector in the quaternion to the given x.

5.2.3.13 setY()

```
void FAMath::Quaternion::setY (
            const double & y )
```

Sets the y component of the 3D vector in the quaternion to the given y.

5.2.3.14 setZ()

```
void FAMath::Quaternion::setZ (
            const double & z )
```

Sets the z component of the 3D vector in the quaternion to the given z.

5.2.3.15 toRotationMatrix()

```
Matrix4x4 FAMath::Quaternion::toRotationMatrix ( )
```

Creates a rotation matrix from this quaternion. Normalize the quaternion before using this function.

5.2.3.16 vector()

```
Vector3 FAMath::Quaternion::vector ( ) const
```

Returns the 3D vector component of the quaternion.

5.2.3.17 x()

```
double FAMath::Quaternion::x ( ) const
```

Returns the x component of the quaternion's 3D vector.

5.2.3.18 y()

```
double FAMath::Quaternion::y ( ) const
```

Returns the y component of the quaternion's 3D vector.

5.2.3.19 z()

```
double FAMath::Quaternion::z ( ) const
```

Returns the z component of the quaternion's 3D vector.

5.2.4 Friends And Related Function Documentation

5.2.4.1 conjugate

```
Quaternion conjugate (
            const Quaternion & q )  [friend]
```

Returns the conjugate of quaternion q.

5.2.4.2 dotProduct

```
double dotProduct (
            const Quaternion & q1,
            const Quaternion & q2 )  [friend]
```

Returns the dot product between q1 and q2.

5.2.4.3 inverse

```
Quaternion inverse (
            const Quaternion & q )  [friend]
```

Returns the inverse of quaternion q. If the quaternion q is the zero quaternion, then the zero quaternion is returned.

5.2.4.4 length

```
double length (
            const Quaternion & q )  [friend]
```

Returns the magnitude of the quaternion.

5.2.4.5 normalize

```
Quaternion normalize (
            const Quaternion & q )  [friend]
```

Normalizes the quaternion q.

If the quaternion is a zero quaternion then the zero quaternion is returned.

Returns

a Quaternion object that has the result q / |q|.

5.2.4.6 operator"!=

```
bool operator!= (
            const Quaternion & q1,
            const Quaternion & q2 )  [friend]
```

Return true if q1 and q2 aren't equal, false otherwise.

5.2.4.7 operator* [1/4]

```
Quaternion operator* (
            const double & k,
            const Quaternion & q )  [friend]
```

Returns a Quaternion object that has the result of k * q.

5.2.4.8 operator* [2/4]

```
Quaternion operator* (
            const Quaternion & q,
            const double & k )  [friend]
```

Returns a Quaternion object that has the result of q * k.

5.2.4.9 operator* [3/4]

```
Vector3 operator* (
            const Quaternion & q,
            const Vector3 & v )  [friend]
```

Rotates 3D vector v by quaternion q to produce a new 3D vector.

5.2.4.10 operator* [4/4]

```
Quaternion operator* (
            const Quaternion & q1,
            const Quaternion & q2 )  [friend]
```

Returns the product of q1 and q2 using quaternion multiplication.

5.2.4.11 operator+

```
Quaternion operator+ (
            const Quaternion & q1,
            const Quaternion & q2 )  [friend]
```

Returns a Quaternion object that is the sum of q1 and q2.

5.2.4.12 operator- [1/2]

```
Quaternion operator- (
            const Quaternion & q )  [friend]
```

Negates the scalar value and each componenet in the 3D vector of q.

Returns

a Quaternion object that has the result of -q.

5.2.4.13 operator- [2/2]

```
Quaternion operator- (
            const Quaternion & q1,
            const Quaternion & q2 )  [friend]
```

Returns a Quaternion object that has the result of q1 - q2;.

5.2.4.14 operator==

```
bool operator== (
            const Quaternion & q1,
            const Quaternion & q2 )  [friend]
```

Return true if q1 and q2 are equal, false otherwise.

5.2.4.15 slerp

```
Quaternion slerp (
            const Quaternion & q1,
            const Quaternion & q2,
            const double & t )  [friend]
```

Spherically Interpolates between rotations q1 and q2.

t should be between 0 and 1. If t $<$ 0 q1 will be returned. If t $>$ 1 q2 will be returned.

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

- C:/Users/Work/Desktop/Math/FAMath/FAMath/FAMathLibrary.h

5.3 FAMath::Vector2 Class Reference

A vector class used for 2D vectors/points and their manipulations.

```
#include "FAMathLibrary.h"
```

Public Member Functions

- bool isZeroVector () const

  Returns true if all the components of the 2D vector equals to zero, false otherwise.
- Vector2 & operator+= (const Vector2 &b)

  2D vector addition through operator += overloading.
- Vector2 & operator-= (const Vector2 &b)

  2D vector subtraction through operator -= overloading.
- Vector2 & operator∗= (const double &scalar)

  2D vector multiplication by a scalar through operator ∗= overloading.
- Vector2 & operator/= (const double &scalar)

  2D vector division by a scalar through operator /= overloading.
- void operator= (const Vector3 &v)

  Assignment Operator Overloading.
- void operator= (const Vector4 &v)

  Assignment Operator Overloading.

Constructors

Constructors for class FAMath::Vector2

- Vector2 ()
  Default Constructor.
- Vector2 (double x, double y)
  Overloaded Constructor.
- Vector2 (const Vector3 &v)
  Overloaded Constructor.
- Vector2 (const Vector4 &v)
  Overloaded Constructor.

Getters

Getters for class FAMath::Vector2

- double x () const
  Returns the value of the x-coordinate.
- double y () const
  Returns the value of the y-coordinate.

Setters

Setters for class FAMath::Vector2.

- void setX (double x)
  Sets the x-coordinate of the 2D vector/point.
- void setY (double y)
  Sets the y-coordinate of the 2D vector/point.

Friends

- Vector2 operator+ (const Vector2 &a, const Vector2 &b)
  2D vector addition through operator + overloading.
- Vector2 operator- (const Vector2 &a, const Vector2 &b)
  2D vector subtraction through operator - overloading.
- Vector2 operator∗ (const Vector2 &v, const double &scalar)
  2D vector multiplication by a scalar through operator ∗ overloading.
- Vector2 operator∗ (const double &scalar, const Vector2 &v)
  2D vector multiplication by a scalar through operator ∗ overloading.
- Vector2 operator/ (const Vector2 &v, const double &scalar)
  2D vector divison by a scalar through operator / overloading.
- bool operator== (const Vector2 &a, const Vector2 &b)
  Compares two 2D vectors through operator == overloading.
- bool operator!= (const Vector2 &a, const Vector2 &b)
  Compares two 2D vectors through operator != overloading.
- double length (const Vector2 &v)
  Magnitude/Length of a 2D vector.
- Vector2 normalize (const Vector2 &v)
  Normalizes a 2D vector.
- double distance (const Vector2 &a, const Vector2 &b)
  Distane between two 2D points.
- double dotProduct (const Vector2 &a, const Vector2 &b)
  Dot Product.
- double angle (const Vector2 &a, const Vector2 &b)
  Angle between two 2D vectors.
- void **print** (Vector2 v)

5.3.1 Detailed Description

A vector class used for 2D vectors/points and their manipulations.

The datatype for the components is double

Definition at line 26 of file FAMathLibrary.h.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Vector2() [1/4]

```
FAMath::Vector2::Vector2 ( )
```

Default Constructor.

Creates a new 2D vector/point with the components initialized to 0.0.

See also

Vector2(double x, double y);

5.3.2.2 Vector2() [2/4]

```
FAMath::Vector2::Vector2 (
            double x,
            double y )
```

Overloaded Constructor.

Creates a new 2D vector/point with the components initialized to the arguments.

5.3.2.3 Vector2() [3/4]

```
FAMath::Vector2::Vector2 (
            const Vector3 & v )
```

Overloaded Constructor.

Creates a new 2D vector/point using v's x and y coordinates.

5.3.2.4 Vector2() [4/4]

```
FAMath::Vector2::Vector2 (
            const Vector4 & v )
```

Overloaded Constructor.

Creates a new 2D vector/point using v's x and y coordinates

5.3.3 Member Function Documentation

5.3.3.1 isZeroVector()

```
bool FAMath::Vector2::isZeroVector ( ) const
```

Returns true if all the components of the 2D vector equals to zero, false otherwise.

5.3.3.2 operator*=()

```
Vector2 & FAMath::Vector2::operator*= (
            const double & scalar )
```

2D vector multiplication by a scalar through operator *= overloading.

Returns

A reference to the current vector object
That has the result of the current Vector2 object * scalar.

5.3.3.3 operator+=()

```
Vector2 & FAMath::Vector2::operator+= (
            const Vector2 & b )
```

2D vector addition through operator += overloading.

Returns

A reference to the current Vector2 object
That has the result of the current Vector2 object + Vector2 object b.

5.3.3.4 operator-=()

```
Vector2 & FAMath::Vector2::operator-= (
            const Vector2 & b )
```

2D vector subtraction through operator -= overloading.

Returns

A reference to the current Vector2 object
That has the result of the current Vector2 object - Vector2 object b.

5.3.3.5 operator/=()

```
Vector2 & FAMath::Vector2::operator/= (
            const double & scalar )
```

2D vector division by a scalar through operator /= overloading.

Throws a std::invalid_argument if scalar is zero

Returns

A reference to the current vector object
That has the result of the current Vector2 object / scalar.

5.3.3.6 operator=() [1/2]

```
void FAMath::Vector2::operator= (
            const Vector3 & v )
```

Assignment Operator Overloading.

Stores the x and y values of Vector3s v into the x and y values of this Vector2.

5.3.3.7 operator=() [2/2]

```
void FAMath::Vector2::operator= (
            const Vector4 & v )
```

Assignment Operator Overloading.

Stores the x and y values of Vector4s v into the x and y values of this Vector2.

5.3.3.8 setX()

```
void FAMath::Vector2::setX (
            double x )
```

Sets the x-coordinate of the 2D vector/point.

See also

void setY(double y)

5.3.3.9 setY()

```
void FAMath::Vector2::setY (
            double y )
```

Sets the y-coordinate of the 2D vector/point.

See also

void setX(double x)

5.3.3.10 x()

```
double FAMath::Vector2::x ( ) const
```

Returns the value of the x-coordinate.

See also

double y()

5.3.3.11 y()

```
double FAMath::Vector2::y ( ) const
```

Returns the value of the y-coordinate.

See also

double x()

5.3.4 Friends And Related Function Documentation

5.3.4.1 angle

```
double angle (
            const Vector2 & a,
            const Vector2 & b )  [friend]
```

Angle between two 2D vectors.

a and b should be unit vectors before using them as arguments.

Returns

The angle between two 2D vectors.

5.3.4.2 distance

```
double distance (
            const Vector2 & a,
            const Vector2 & b )  [friend]
```

Distane between two 2D points.

Returns

The distance between two 2D points.

5.3.4.3 dotProduct

```
double dotProduct (
            const Vector2 & a,
            const Vector2 & b )  [friend]
```

Dot Product.

Returns

The value of a dot b.

5.3.4.4 length

```
double length (
            const Vector2 & v )  [friend]
```

Magnitude/Length of a 2D vector.

Returns

The length of Vector2 object v.

5.3.4.5 normalize

```
Vector2 normalize (
            const Vector2 & v )  [friend]
```

Normalizes a 2D vector.

If v is a zero vector then the zero vector is returned.

Returns

A Vector2 object that has the result v / |v| which is a unit vector.

5.3.4.6 operator"!=

```
bool operator!= (
            const Vector2 & a,
            const Vector2 & b )  [friend]
```

Compares two 2D vectors through operator != overloading.

Returns

False if a equals to b.

True otherwise.

5.3.4.7 operator* [1/2]

```
Vector2 operator* (
            const double & scalar,
            const Vector2 & v )  [friend]
```

2D vector multiplication by a scalar through operator * overloading.

Called when you do scalar * v, where v is a Vector2 object

Returns

A new Vector2 object that has the result of scalar * v.

5.3.4.8 operator* [2/2]

```
Vector2 operator* (
            const Vector2 & v,
            const double & scalar )  [friend]
```

2D vector multiplication by a scalar through operator * overloading.

Called when you do v * scalar, where v is a Vector2 object.

Returns

A new Vector2 object that has the result of v * scalar.

5.3.4.9 operator+

```
Vector2 operator+ (
            const Vector2 & a,
            const Vector2 & b )  [friend]
```

2D vector addition through operator + overloading.

Returns

A new Vector2 object that has the result of a + b.

5.3.4.10 operator-

```
Vector2 operator- (
            const Vector2 & a,
            const Vector2 & b )  [friend]
```

2D vector subtraction through operator - overloading.

Returns

A new Vector2 object that has the result of a - b.

5.3.4.11 operator/

```
Vector2 operator/ (
            const Vector2 & v,
            const double & scalar )  [friend]
```

2D vector divison by a scalar through operator / overloading.

Returns

A new Vector2 object that has the result of v/scalar.

5.3.4.12 operator==

```
bool operator== (
            const Vector2 & a,
            const Vector2 & b )  [friend]
```

Compares two 2D vectors through operator == overloading.

Returns

True if a equals to b.

False otherwise.

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

- C:/Users/Work/Desktop/Math/FAMath/FAMath/FAMathLibrary.h

5.4 FAMath::Vector3 Class Reference

A vector class used for 3D vectors/points and their manipulations.

```
#include "FAMathLibrary.h"
```

Public Member Functions

- bool isZeroVector () const

  Returns true if all the components of the 3D vector equals to zero, false otherwise.
- Vector3 & operator+= (const Vector3 &b)

  3D vector addition through operator += overloading.
- Vector3 & operator-= (const Vector3 &b)

  3D vector subtraction through operator -= overloading.
- Vector3 & operator∗= (const double &scalar)

  3D vector multiplication by a scalar through operator ∗= overloading.
- Vector3 & operator/= (const double &scalar)

  3D vector division by a scalar through operator /= overloading.
- void operator= (const Vector2 &v)

  Assignment Operator Overloading.
- void operator= (const Vector4 &v)

  Assignment Operator Overloading.

Constructors

Constructors for class FAMath::Vector3.

- Vector3 ()

  Default Constructor.
- Vector3 (double x, double y, double z)

  Overloaded Constructor.
- Vector3 (const Vector2 &v)

  Overloaded Constructor.
- Vector3 (const Vector2 &v, const double &z)

Overloaded Constructor.

- Vector3 (const Vector4 &v)

  Overloaded Constructor.

Getters

Getters for class FAMath::Vector3.

- double x () const

  Returns the value of the x-coordinate.
- double y () const

  Returns the value of the y-coordinate.
- double z () const

  Returns the value of the z-coordinate.

Setters

Setters for class FAMath::Vector3.

- void setX (double x)

  Sets the x-coordinate of the 3D vector/point.
- void setY (double y)

  Sets the y-coordinate of the 3D vector/point.
- void setZ (double z)

  Sets the z-coordinate of the 3D vector/point.

Friends

- Vector3 operator+ (const Vector3 &a, const Vector3 &b)

  3D vector addition through operator + overloading.
- Vector3 operator- (const Vector3 &a, const Vector3 &b)

  3D vector subtraction through operator - overloading.
- Vector3 operator∗ (const Vector3 &v, const double &scalar)

  3D vector multiplication by a scalar through operator ∗ overloading.
- Vector3 operator∗ (const double &scalar, const Vector3 &v)

  3D vector multiplication by a scalar through operator ∗ overloading.
- Vector3 operator/ (const Vector3 &v, const double &scalar)

  3D vector divison by a scalar through operator / overloading.
- bool operator== (const Vector3 &a, const Vector3 &b)

  Compares two 3D vectors through operator == overloading.
- bool operator!= (const Vector3 &a, const Vector3 &b)

  Compares two 3D vectors through operator != overloading.
- double length (const Vector3 &v)

  Magnitude/Length of a 3D vector.
- Vector3 normalize (const Vector3 &v)

  Normalizes a 3D vector.
- double distance (const Vector3 &a, const Vector3 &b)

  Distane between two 3D points.
- double dotProduct (const Vector3 &a, const Vector3 &b)

  Dot Product.
- double angle (const Vector3 &a, const Vector3 &b)

  Angle between two 3D vectors.
- Vector3 crossProduct (const Vector3 &a, const Vector3 &b)

  Cross Product.
- void **print** (Vector3 v)

5.4.1 Detailed Description

A vector class used for 3D vectors/points and their manipulations.

The datatype for the components is double.

Definition at line 253 of file FAMathLibrary.h.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Vector3() [1/5]

```
FAMath::Vector3::Vector3 ( )
```

Default Constructor.

Creates a new 3D vector/point with the components initialized to 0.0.

See also

Vector3(double x, double y);

5.4.2.2 Vector3() [2/5]

```
FAMath::Vector3::Vector3 (
            double x,
            double y,
            double z )
```

Overloaded Constructor.

Creates a new 3D vector/point with the components initialized to the arguments.

5.4.2.3 Vector3() [3/5]

```
FAMath::Vector3::Vector3 (
            const Vector2 & v )
```

Overloaded Constructor.

Creates a new 3D vector/point using v's x and y coordinates and sets z to 0.0.

5.4.2.4 Vector3() [4/5]

```
FAMath::Vector3::Vector3 (
            const Vector2 & v,
            const double & z )
```

Overloaded Constructor.

Creates a new 3D vector/point using v's x and y coordinates and sets this Vector3 z component to the given z

5.4.2.5 Vector3() [5/5]

```
FAMath::Vector3::Vector3 (
            const Vector4 & v )
```

Overloaded Constructor.

Creates a new 3D vector/point using v's x, y and z coordinates.

5.4.3 Member Function Documentation

5.4.3.1 isZeroVector()

```
bool FAMath::Vector3::isZeroVector ( ) const
```

Returns true if all the components of the 3D vector equals to zero, false otherwise.

5.4.3.2 operator∗=()

```
Vector3 & FAMath::Vector3::operator*= (
            const double & scalar )
```

3D vector multiplication by a scalar through operator ∗= overloading.

Returns

A reference to the current vector object
That has the result of the current Vector3 object ∗ scalar.

5.4.3.3 operator+=()

```
Vector3 & FAMath::Vector3::operator+= (
            const Vector3 & b )
```

3D vector addition through operator += overloading.

Returns

A reference to the current Vector3 object
That has the result of the current Vector3 object + Vector3 object b.

5.4.3.4 operator-=()

```
Vector3 & FAMath::Vector3::operator-= (
            const Vector3 & b )
```

3D vector subtraction through operator -= overloading.

Returns

A reference to the current Vector3 object
That has the result of the current Vector3 object - Vector3 object b.

5.4.3.5 operator/=()

```
Vector3 & FAMath::Vector3::operator/= (
            const double & scalar )
```

3D vector division by a scalar through operator /= overloading.

Throws a std::invalid_argument if scalar is zero

Returns

A reference to the current vector object
That has the result of the current Vector3 object / scalar.

5.4.3.6 operator=() [1/2]

```
void FAMath::Vector3::operator= (
            const Vector2 & v )
```

Assignment Operator Overloading.

Stores the x and y values of Vector2s v into the x and y values of this Vector3 and sets z = 0.0.

5.4.3.7 operator=() [2/2]

```
void FAMath::Vector3::operator= (
            const Vector4 & v )
```

Assignment Operator Overloading.

Stores the x, y and z values of Vector4s v into the x, y and z values of this Vector3.

5.4.3.8 setX()

```
void FAMath::Vector3::setX (
            double x )
```

Sets the x-coordinate of the 3D vector/point.

See also

void setY(double y)

void setZ(double z)

5.4.3.9 setY()

```
void FAMath::Vector3::setY (
            double y )
```

Sets the y-coordinate of the 3D vector/point.

See also

void setX(double x)

void setZ(double z)

5.4.3.10 setZ()

```
void FAMath::Vector3::setZ (
            double z )
```

Sets the z-coordinate of the 3D vector/point.

See also

void setX(double x)

void setY(double y)

5.4.3.11 x()

```
double FAMath::Vector3::x ( ) const
```

Returns the value of the x-coordinate.

See also

double y()
double z()

5.4.3.12 y()

```
double FAMath::Vector3::y ( ) const
```

Returns the value of the y-coordinate.

See also

double x()
double z()

5.4.3.13 z()

```
double FAMath::Vector3::z ( ) const
```

Returns the value of the z-coordinate.

See also

double x()
double y()

5.4.4 Friends And Related Function Documentation

5.4.4.1 angle

```
double angle (
            const Vector3 & a,
            const Vector3 & b )  [friend]
```

Angle between two 3D vectors.

a and b should be unit vectors before using them as arguments.

Returns

The angle between two 3D vectors.

5.4.4.2 crossProduct

```
Vector3 crossProduct (
            const Vector3 & a,
            const Vector3 & b )  [friend]
```

Cross Product.

Returns

A Vector3 object that is perpendicular to a and b.

5.4.4.3 distance

```
double distance (
            const Vector3 & a,
            const Vector3 & b )  [friend]
```

Distane between two 3D points.

Returns

The distance between two 3D points.

5.4.4.4 dotProduct

```
double dotProduct (
            const Vector3 & a,
            const Vector3 & b )  [friend]
```

Dot Product.

Returns

The value of a dot b.

5.4.4.5 length

```
double length (
            const Vector3 & v )  [friend]
```

Magnitude/Length of a 3D vector.

Returns

The length of Vector3 object v.

5.4.4.6 normalize

```
Vector3 normalize (
            const Vector3 & v )  [friend]
```

Normalizes a 3D vector.

If v is a zero vector then the zero vector is returned.

Returns

A Vector3 object that is a unit vector(has a length of 1).

5.4.4.7 operator"!=

```
bool operator!= (
            const Vector3 & a,
            const Vector3 & b )  [friend]
```

Compares two 3D vectors through operator != overloading.

Returns

False if a equals to b.

True otherwise.

5.4.4.8 operator* [1/2]

```
Vector3 operator* (
            const double & scalar,
            const Vector3 & v )  [friend]
```

3D vector multiplication by a scalar through operator * overloading.

Called when you do scalar * v, where v is a Vector3 object.

Returns

A new Vector3 object that has the result of scalar * v.

5.4.4.9 operator* [2/2]

```
Vector3 operator* (
            const Vector3 & v,
            const double & scalar )  [friend]
```

3D vector multiplication by a scalar through operator * overloading.

Called when you do v * scalar, where v is a Vector3 object.

Returns

A new Vector3 object that has the result of v * scalar.

5.4.4.10 operator+

```
Vector3 operator+ (
            const Vector3 & a,
            const Vector3 & b )  [friend]
```

3D vector addition through operator + overloading.

Returns

A new Vector3 object that has the result of a + b.

5.4.4.11 operator-

```
Vector3 operator- (
            const Vector3 & a,
            const Vector3 & b )  [friend]
```

3D vector subtraction through operator - overloading.

Returns

A new Vector3 object that has the result of a - b.

5.4.4.12 operator/

```
Vector3 operator/ (
            const Vector3 & v,
            const double & scalar )  [friend]
```

3D vector divison by a scalar through operator / overloading.

Returns

A new Vector3 object that has the result of v/scalar.

5.4.4.13 operator==

```
bool operator== (
            const Vector3 & a,
            const Vector3 & b )  [friend]
```

Compares two 3D vectors through operator == overloading.

Returns

True if a equals to b.

False otherwise.

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

- C:/Users/Work/Desktop/Math/FAMath/FAMath/FAMathLibrary.h

5.5 FAMath::Vector4 Class Reference

A vector class used for 4D vectors/points and their manipulations.

```
#include "FAMathLibrary.h"
```

Public Member Functions

- bool isZeroVector () const

  Returns true if all the components of the 4D vector equals to zero, false otherwise.
- Vector4 & operator+= (const Vector4 &b)

  4D vector addition through operator += overloading.
- Vector4 & operator-= (const Vector4 &b)

  4D vector subtraction through operator -= overloading.
- Vector4 & operator∗= (const double &scalar)

  4D vector multiplication by a scalar through operator ∗= overloading.
- Vector4 & operator/= (const double &scalar)

  4D vector division by a scalar through operator /= overloading.
- void operator= (const Vector2 &v)

  Assignment Operator Overloading.
- void operator= (const Vector3 &v)

  Assignment Operator Overloading.

Constructors

Constructors for class FAMath::Vector4,

- Vector4 ()

  Default Constructor.
- Vector4 (double x, double y, double z, double w)

  Overloaded Constructor.
- Vector4 (const Vector2 &v)

  Overloaded Constructor.
- Vector4 (const Vector2 &v, const double &z, const double &w)

Overloaded Constructor.

- Vector4 (const Vector2 &v, const double &z)

  Overloaded Constructor.
- Vector4 (const Vector3 &v)

  Overloaded Constructor.
- Vector4 (const Vector3 &v, const double &w)

  Overloaded Constructor.

Getters

Getters for class FAMath::Vector4.

- double x () const

  Returns the value of the x-coordinate.
- double y () const

  Returns the value of the y-coordinate.
- double z () const

  Returns the value of the z-coordinate.
- double w () const

  Returns the value of the w-coordinate.

Setters

Setters for class FAMath::Vector4.

- void setX (double x)

  Sets the x-coordinate of the 4D vector/point.
- void setY (double y)

  Sets the y-coordinate of the 4D vector/point.
- void setZ (double z)

  Sets the z-coordinate of the 4D vector/point.
- void setW (double w)

  Sets the w-coordinate of the 4D vector/point.

Friends

- Vector4 operator+ (const Vector4 &a, const Vector4 &b)

  4D vector addition through operator + overloading.
- Vector4 operator- (const Vector4 &a, const Vector4 &b)

  4D vector subtraction through operator - overloading.
- Vector4 operator∗ (const Vector4 &v, const double &scalar)

  4D vector multiplication by a scalar through operator ∗ overloading.
- Vector4 operator∗ (const double &scalar, const Vector4 &v)

  4D vector multiplication by a scalar through operator ∗ overloading.
- Vector4 operator/ (const Vector4 &v, const double &scalar)

  4D vector divison by a scalar through operator / overloading.
- bool operator== (const Vector4 &a, const Vector4 &b)

  Compares two 4D vectors through operator == overloading.
- bool operator!= (const Vector4 &a, const Vector4 &b)

  Compares two 4D vectors through operator != overloading.
- double length (const Vector4 &v)

  Magnitude/Length of a 4D vector.
- Vector4 normalize (const Vector4 &v)

  Normalizes a 4D vector.
- double distance (const Vector4 &a, const Vector4 &b)

  Distane between two 4D points.
- double dotProduct (const Vector4 &a, const Vector4 &b)

  Dot Product.
- double angle (const Vector4 &a, const Vector4 &b)

  Angle between two 4D vectors.
- void **print** (Vector4 v)

5.5.1 Detailed Description

A vector class used for 4D vectors/points and their manipulations.

The datatype for the components is double

Definition at line 512 of file FAMathLibrary.h.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 Vector4() [1/7]

```
FAMath::Vector4::Vector4 ( )
```

Default Constructor.

Creates a new 4D vector/point with the components initialized to 0.0.

See also

Vector4(double x, double y);

5.5.2.2 Vector4() [2/7]

```
FAMath::Vector4::Vector4 (
            double x,
            double y,
            double z,
            double w )
```

Overloaded Constructor.

Creates a new 4D vector/point with the components initialized to the arguments,

5.5.2.3 Vector4() [3/7]

```
FAMath::Vector4::Vector4 (
            const Vector2 & v )
```

Overloaded Constructor.

Creates a new 4D vector/point using v's x and y coordinates and sets z and w to 0.0.

5.5.2.4 Vector4() [4/7]

```
FAMath::Vector4::Vector4 (
            const Vector2 & v,
            const double & z,
            const double & w )
```

Overloaded Constructor.

Creates a new 4D vector/point using v's x and y coordinates and z and w values.

5.5.2.5 Vector4() [5/7]

```
FAMath::Vector4::Vector4 (
            const Vector2 & v,
            const double & z )
```

Overloaded Constructor.

Creates a new 4D vector / point using v's x and y coordinates and the given z and sets w to 0.0.

5.5.2.6 Vector4() [6/7]

```
FAMath::Vector4::Vector4 (
            const Vector3 & v )
```

Overloaded Constructor.

Creates a new 4D vector/point using v's x, y and z coordinates and sets w to 0.0.

5.5.2.7 Vector4() [7/7]

```
FAMath::Vector4::Vector4 (
            const Vector3 & v,
            const double & w )
```

Overloaded Constructor.

Creates a new 4D vector/point using v's x, y and z coordinates and the given w value.

5.5.3 Member Function Documentation

5.5.3.1 isZeroVector()

```
bool FAMath::Vector4::isZeroVector ( ) const
```

Returns true if all the components of the 4D vector equals to zero, false otherwise.

5.5.3.2 operator∗=()

```
Vector4 & FAMath::Vector4::operator*= (
            const double & scalar )
```

4D vector multiplication by a scalar through operator ∗= overloading.

Returns

A reference to the current vector object
That has the result of the current Vector4 object ∗ scalar.

5.5.3.3 operator+=()

```
Vector4 & FAMath::Vector4::operator+= (
            const Vector4 & b )
```

4D vector addition through operator += overloading.

Returns

A reference to the current Vector4 object
That has the result of the current Vector4 object + Vector4 object b.

5.5.3.4 operator-=()

```
Vector4 & FAMath::Vector4::operator-= (
            const Vector4 & b )
```

4D vector subtraction through operator -= overloading.

Returns

A reference to the current Vector4 object
That has the result of the current Vector4 object - Vector4 object b.

5.5.3.5 operator/=()

```
Vector4 & FAMath::Vector4::operator/= (
            const double & scalar )
```

4D vector division by a scalar through operator /= overloading.

Throws a std::invalid_argument if scalar is zero

Returns

A reference to the current vector object
That has the result of the current Vector4 object / scalar.

5.5.3.6 operator=() [1/2]

```
void FAMath::Vector4::operator= (
            const Vector2 & v )
```

Assignment Operator Overloading.

Stores the x and y values of Vector2s v into the x and y values of this Vector4 and sets w and to 0.0.

5.5.3.7 operator=() [2/2]

```
void FAMath::Vector4::operator= (
            const Vector3 & v )
```

Assignment Operator Overloading.

Stores the x, y and z values of Vector3s v into the x, y and z values of this Vector4 and sets w to 0.0.

5.5.3.8 setW()

```
void FAMath::Vector4::setW (
            double w )
```

Sets the w-coordinate of the 4D vector/point.

See also

void setX(double x)

void setY(double y)

5.5.3.9 setX()

```
void FAMath::Vector4::setX (
            double x )
```

Sets the x-coordinate of the 4D vector/point.

See also

void setY(double y)

void setZ(double z)

5.5.3.10 setY()

```
void FAMath::Vector4::setY (
            double y )
```

Sets the y-coordinate of the 4D vector/point.

See also

void setX(double x)
void setZ(double z)

5.5.3.11 setZ()

```
void FAMath::Vector4::setZ (
            double z )
```

Sets the z-coordinate of the 4D vector/point.

See also

void setX(double x)
void setY(double y)

5.5.3.12 w()

```
double FAMath::Vector4::w ( ) const
```

Returns the value of the w-coordinate.

See also

double x()
double y()
double z()

5.5.3.13 x()

```
double FAMath::Vector4::x ( ) const
```

Returns the value of the x-coordinate.

See also

double y()
double z()
double w()

5.5.3.14 y()

```
double FAMath::Vector4::y ( ) const
```

Returns the value of the y-coordinate.

See also

double x()

double z()

double w()

5.5.3.15 z()

```
double FAMath::Vector4::z ( ) const
```

Returns the value of the z-coordinate.

See also

double x()

double y()

double w()

5.5.4 Friends And Related Function Documentation

5.5.4.1 angle

```
double angle (
            const Vector4 & a,
            const Vector4 & b )  [friend]
```

Angle between two 4D vectors.

a and b should be unit vectors before using them as arguments.

Returns

The angle between two 4D vectors.

5.5.4.2 distance

```
double distance (
            const Vector4 & a,
            const Vector4 & b )  [friend]
```

Distane between two 4D points.

If v is the zero vector then the zero vector is returned.

Returns

The distance between two 4D points.

5.5.4.3 dotProduct

```
double dotProduct (
            const Vector4 & a,
            const Vector4 & b )  [friend]
```

Dot Product.

Returns

The value of a dot b.

5.5.4.4 length

```
double length (
            const Vector4 & v )  [friend]
```

Magnitude/Length of a 4D vector.

Returns

The length of Vector4 object v.

5.5.4.5 normalize

```
Vector4 normalize (
            const Vector4 & v )  [friend]
```

Normalizes a 4D vector.

Returns

A Vector4 object that is a unit vector(has a length of 1).

5.5.4.6 operator"!=

```
bool operator!= (
            const Vector4 & a,
            const Vector4 & b )  [friend]
```

Compares two 4D vectors through operator != overloading.

Returns

False if a equals to b.

True otherwise.

5.5.4.7 operator∗ [1/2]

```
Vector4 operator* (
            const double & scalar,
            const Vector4 & v )  [friend]
```

4D vector multiplication by a scalar through operator ∗ overloading.

Called when you do scalar ∗ v, where v is a Vector4 object.

Returns

A new Vector4 object that has the result of scalar ∗ v.

5.5.4.8 operator∗ [2/2]

```
Vector4 operator* (
            const Vector4 & v,
            const double & scalar )  [friend]
```

4D vector multiplication by a scalar through operator ∗ overloading.

Called when you do v ∗ scalar, where v is a Vector4 object.

Returns

A new Vector4 object that has the result of v ∗ scalar.

5.5.4.9 operator+

```
Vector4 operator+ (
            const Vector4 & a,
            const Vector4 & b )  [friend]
```

4D vector addition through operator + overloading.

Returns

A new Vector4 object that has the result of a + b.

5.5.4.10 operator-

```
Vector4 operator- (
            const Vector4 & a,
            const Vector4 & b )  [friend]
```

4D vector subtraction through operator - overloading.

Returns

A new Vector4 object that has the result of a - b.

5.5.4.11 operator/

```
Vector4 operator/ (
            const Vector4 & v,
            const double & scalar )  [friend]
```

4D vector divison by a scalar through operator / overloading.

Returns

A new Vector4 object that has the result of v/scalar.

5.5.4.12 operator==

```
bool operator== (
            const Vector4 & a,
            const Vector4 & b )  [friend]
```

Compares two 4D vectors through operator == overloading.

Returns

True if a equals to b.

False otherwise.

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

- C:/Users/Work/Desktop/Math/FAMath/FAMath/FAMathLibrary.h

Chapter 6

File Documentation

6.1 FAMathLibrary.h

```
00001
00010 #pragma once
00011
00015 namespace FAMath
00016 {
00017     class Vector2;
00018     class Vector3;
00019     class Vector4;
00020
00026     class Vector2
00027     {
00028     public:
00029
00034
00041         Vector2();
00042
00047         Vector2(double x, double y);
00048
00053         Vector2(const Vector3& v);
00054
00060         Vector2(const Vector4& v);
00061
00063
00068
00074         double x() const;
00075
00080         double y() const;
00081
00083
00088
00093         void setX(double x);
00094
00099         void setY(double y);
00100
00102
00105         bool isZeroVector() const;
00106
00112         Vector2& operator+=(const Vector2& b);
00113
00119         Vector2& operator-=(const Vector2& b);
00120
00126         Vector2& operator*=(const double& scalar);
00127
00135         Vector2& operator/=(const double& scalar);
00136
00141         void operator=(const Vector3& v);
00142
00147         void operator=(const Vector4& v);
00148
00153         friend Vector2 operator+(const Vector2& a, const Vector2& b);
00154
00159         friend Vector2 operator-(const Vector2& a, const Vector2& b);
00160
00167         friend Vector2 operator*(const Vector2& v, const double& scalar);
00168
00175         friend Vector2 operator*(const double& scalar, const Vector2& v);
00176
00181         friend Vector2 operator/(const Vector2& v, const double& scalar);
```

```
00182
00188         friend bool operator==(const Vector2& a, const Vector2& b);
00189
00195         friend bool operator!=(const Vector2& a, const Vector2& b);
00196
00201         friend double length(const Vector2& v);
00202
00209         friend Vector2 normalize(const Vector2& v);
00210
00215         friend double distance(const Vector2& a, const Vector2& b);
00216
00221         friend double dotProduct(const Vector2& a, const Vector2& b);
00222
00229         friend double angle(const Vector2& a, const Vector2& b);
00230
00231         friend void print(Vector2 v);
00232
00233     private:
00234         //components of a 2D Vector
00235         double m_x;
00236         double m_y;
00237     };
00238
00239
00240
00241
00242
00243
00244
00245
00246
00247
00253     class Vector3
00254     {
00255     public:
00256
00261
00268         Vector3();
00269
00274         Vector3(double x, double y, double z);
00275
00280         Vector3(const Vector2& v);
00281
00286         Vector3(const Vector2& v, const double& z);
00287
00292         Vector3(const Vector4& v);
00293
00295
00300
00306         double x() const;
00307
00313         double y() const;
00314
00320         double z() const;
00321
00323
00328
00334         void setX(double x);
00335
00341         void setY(double y);
00342
00348         void setZ(double z);
00349
00351
00354         bool isZeroVector() const;
00355
00361         Vector3& operator+=(const Vector3& b);
00362
00368         Vector3& operator-=(const Vector3& b);
00369
00375         Vector3& operator*=(const double& scalar);
00376
00384         Vector3& operator/=(const double& scalar);
00385
00390         void operator=(const Vector2& v);
00391
00396         void operator=(const Vector4& v);
00397
00402         friend Vector3 operator+(const Vector3& a, const Vector3& b);
00403
00408         friend Vector3 operator-(const Vector3& a, const Vector3& b);
00409
00416         friend Vector3 operator*(const Vector3& v, const double& scalar);
00417
00424         friend Vector3 operator*(const double& scalar, const Vector3& v);
00425
00430         friend Vector3 operator/(const Vector3& v, const double& scalar);
```

```
00431
00437         friend bool operator==(const Vector3& a, const Vector3& b);
00438
00444         friend bool operator!=(const Vector3& a, const Vector3& b);
00445
00450         friend double length(const Vector3& v);
00451
00458         friend Vector3 normalize(const Vector3& v);
00459
00464         friend double distance(const Vector3& a, const Vector3& b);
00465
00470         friend double dotProduct(const Vector3& a, const Vector3& b);
00471
00478         friend double angle(const Vector3& a, const Vector3& b);
00479
00484         friend Vector3 crossProduct(const Vector3& a, const Vector3& b);
00485
00486         friend void print(Vector3 v);
00487
00488     private:
00489         //components of a 3D Vector
00490         double m_x;
00491         double m_y;
00492         double m_z;
00493     };
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00512     class Vector4
00513     {
00514     public:
00515
00520
00527         Vector4();
00528
00533         Vector4(double x, double y, double z, double w);
00534
00539         Vector4(const Vector2& v);
00540
00545         Vector4(const Vector2& v, const double& z, const double& w);
00546
00551         Vector4(const Vector2& v, const double& z);
00552
00557         Vector4(const Vector3& v);
00558
00563         Vector4(const Vector3& v, const double& w);
00564
00565
00567
00572
00578         double x() const;
00579
00586         double y() const;
00587
00594         double z() const;
00595
00602         double w() const;
00603
00605
00610
00616         void setX(double x);
00617
00623         void setY(double y);
00624
00630         void setZ(double z);
00631
00637         void setW(double w);
00638
00640
00643         bool isZeroVector() const;
00644
00650         Vector4& operator+=(const Vector4& b);
00651
00657         Vector4& operator-=(const Vector4& b);
00658
00664         Vector4& operator*=(const double& scalar);
00665
```

```
00673         Vector4& operator/=(const double& scalar);
00674
00679         void operator=(const Vector2& v);
00680
00685         void operator=(const Vector3& v);
00686
00691         friend Vector4 operator+(const Vector4& a, const Vector4& b);
00692
00697         friend Vector4 operator-(const Vector4& a, const Vector4& b);
00698
00705         friend Vector4 operator*(const Vector4& v, const double& scalar);
00706
00713         friend Vector4 operator*(const double& scalar, const Vector4& v);
00714
00719         friend Vector4 operator/(const Vector4& v, const double& scalar);
00720
00726         friend bool operator==(const Vector4& a, const Vector4& b);
00727
00733         friend bool operator!=(const Vector4& a, const Vector4& b);
00734
00739         friend double length(const Vector4& v);
00740
00745         friend Vector4 normalize(const Vector4& v);
00746
00752         friend double distance(const Vector4& a, const Vector4& b);
00753
00759         friend double dotProduct(const Vector4& a, const Vector4& b);
00760
00768         friend double angle(const Vector4& a, const Vector4& b);
00769
00770         friend void print(Vector4 v);
00771
00772     private:
00773         //components of a 4D Vector
00774         double m_x;
00775         double m_y;
00776         double m_z;
00777         double m_w;
00778     };
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00804     class Matrix4x4
00805     {
00806     public:
00807
00812
00817         Matrix4x4();
00818
00823         Matrix4x4(const double* values);
00824
00825
00831         Matrix4x4(double m11, double m12, double m13, double m14,
00832             double m21, double m22, double m23, double m24,
00833             double m31, double m32, double m33, double m34,
00834             double m41, double m42, double m43, double m44);
00836
00841
00846         Vector4 column(const unsigned int& index) const;
00847
00852         Vector4 row(const unsigned int& index) const;
00853
00858         double* data();
00859
00864         const double* data() const;
00865
00870         const double* constData() const;
00872
00877
00882         void set(const unsigned int& row, const unsigned int& col, const double& value);
00883
00886         void setToIdentity();
00887
00888
00891         void fill(const double& value);
```

```
00892
00897         void setColumn(const unsigned int& index, const Vector4& value);
00898
00903         void setRow(const unsigned int& index, const Vector4& value);
00904
00906
00909         bool isIdentity() const;
00910
00913         Matrix4x4 transposed() const;
00914
00919
00924         const double& operator()(const unsigned int& row, const unsigned int& col) const;
00925
00930         double& operator()(const unsigned int& row, const unsigned int& col);
00931
00936         Matrix4x4& operator+=(const Matrix4x4& m);
00937
00942         Matrix4x4& operator-=(const Matrix4x4& m);
00943
00948         Matrix4x4& operator*=(const double& scalar);
00949
00954         Matrix4x4& operator*=(const Matrix4x4& m);
00955
00962         Matrix4x4& operator/=(const double& scalar);
00963
00966         void rotate(double angle, const Vector3& v);
00967
00970         void rotate(const double& angle, const double& x, const double& y, const double& z);
00971
00976         void rotateUsingQuaternion(const Vector4& v);
00977
00984         void rotateUsingQuaternion(const double& angle, const Vector3& v);
00985
00992         void rotateUsingQuaternion(const double& angle, const double& x, const double& y, const
      double& z);
00993
00996         void scale(const Vector3& v);
00997
01000         void scale(const double& x, const double& y);
01001
01005         void scale(const double& x, const double& y, const double& z);
01006
01009         void scale(const double& factor);
01010
01013         void scale(const Vector3& v, const double& factor);
01014
01017         void translate(const Vector3& v);
01018
01021         void translate(const double& x, const double& y);
01022
01025         void translate(const double& x, const double& y, const double& z);
01026
01030         void ortho(const double& left, const double& right, const double& bottom, const double& top,
      const double& near, const double& far);
01031
01038         void perspective(const double& fov, const double& aspectRatio, const double& near, const
      double& far);
01039
01042         double determinant() const;
01043
01045
01046
01052         friend Matrix4x4 operator+(const Matrix4x4& m1, const Matrix4x4& m2);
01053
01059         friend Matrix4x4 operator-(const Matrix4x4& m1, const Matrix4x4& m2);
01060
01066         friend Matrix4x4 operator-(Matrix4x4& m);
01067
01072         friend Matrix4x4 operator*(const Matrix4x4& m1, const double& scalar);
01073
01078         friend Matrix4x4 operator*(const double& scalar, const Matrix4x4& m1);
01079
01084         friend Matrix4x4 operator*(const Matrix4x4& m1, const Matrix4x4& m2);
01085
01090         friend Vector4 operator*(const Matrix4x4& m, const Vector4& vec);
01091
01096         friend Vector4 operator*(const Vector4& vec, const Matrix4x4& m);
01097
01098
01105         friend Matrix4x4 operator/(const Matrix4x4& m1, const double& scalar);
01106
01109         friend bool operator==(const Matrix4x4& m1, const Matrix4x4& m2);
01110
01113         friend bool operator!=(const Matrix4x4& m1, const Matrix4x4& m2);
01114
01118         friend Matrix4x4 inverse(const Matrix4x4& m);
01119
```

```
01120         friend void print(const Matrix4x4& m);
01121
01122     private:
01123         //A static array of 16 doubles
01124         //1st row for m_matrix is at indices:  0 4 8 12
01125         //2nd row for m_matrix is at indices:  1 5 9 13
01126         //3rd row for m_matrix is at indices:  2 6 10 14
01127         //4th row for m_matrix is at indices:  3 7 11 15
01128         //1st column for m_matrix is at indices:  0 1 2 3
01129         //2nd column for m_matrix is at indices:  4 5 6 7
01130         //3rd column for m_matrix is at indices:  8 9 10 11
01131         //4th column for m_matrix is at indices:  12 13 14 15
01132         double m_matrix[16];
01133
01136         double minor(const unsigned int& row, const unsigned int& col) const;
01137
01140         Matrix4x4 adjoint() const;
01141     };
01142
01143
01144
01145
01146
01153     class Quaternion
01154     {
01155     public:
01156
01161
01166         Quaternion();
01167
01173         Quaternion(const double& w, const Vector3& v);
01174
01180         Quaternion(const double& w, const double& x, const double& y, const double& z);
01181
01183
01188
01191         double scalar() const;
01192
01195         Vector3 vector() const;
01196
01199         double x() const;
01200
01203         double y() const;
01204
01207         double z() const;
01208
01212         void setQuaternion(const double& w, const Vector3& v);
01213
01217         void setQuaternion(const double& w, double& x, double& y, double& z);
01218
01221         void setScalar(const double& w);
01222
01225         void setVector(const Vector3& v);
01226
01229         void setVector(const double& x, const double& y, const double& z);
01230
01233         void setX(const double& x);
01234
01237         void setY(const double& y);
01238
01241         void setZ(const double& z);
01243
01246         bool isZeroQuaternion() const;
01247
01250         Quaternion& operator*=(const double& k);
01251
01254         Quaternion& operator*=(const Quaternion& q);
01255
01258         Quaternion& operator+=(const Quaternion& q);
01259
01262         Quaternion& operator-=(const Quaternion& q);
01263
01267         Matrix4x4 toRotationMatrix();
01268
01273         friend Quaternion operator-(const Quaternion& q);
01274
01277         friend double length(const Quaternion& q);
01278
01285         friend Quaternion normalize(const Quaternion& q);
01286
01289         friend Quaternion conjugate(const Quaternion& q);
01290
01294         friend Quaternion inverse(const Quaternion& q);
01295
01298         friend Quaternion operator*(const Quaternion& q1, const Quaternion& q2);
01299
01302         friend Quaternion operator*(const Quaternion& q, const double& k);
```

```
01303
01306         friend Quaternion operator*(const double& k, const Quaternion& q);
01307
01310         friend Vector3 operator*(const Quaternion& q, const Vector3& v);
01311
01314         friend Quaternion operator+(const Quaternion& q1, const Quaternion& q2);
01315
01318         friend Quaternion operator-(const Quaternion& q1, const Quaternion& q2);
01319
01322         friend bool operator==(const Quaternion& q1, const Quaternion& q2);
01323
01326         friend bool operator!=(const Quaternion& q1, const Quaternion& q2);
01327
01330         friend double dotProduct(const Quaternion& q1, const Quaternion& q2);
01331
01336         friend Quaternion slerp(const Quaternion& q1, const Quaternion& q2, const double& t);
01337
01338
01339         friend void print(const Quaternion& q);
01340
01341
01342     private:
01343         //Scalar value of the quaternion
01344         double m_w;
01345
01346         //3D vector of the quaternion
01347         Vector3 m_v;
01348
01349     };
01350 }
01351
```

Index