

Farouq Adepetu's Math Class For Direct3D

Generated by Doxygen 1.9.4



<b>1 Namespace Index</b>	<b>1</b>
1.1 Namespace List	1
<b>2 Class Index</b>	<b>3</b>
2.1 Class List	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Namespace Documentation</b>	<b>7</b>
4.1 FAD3DMath Namespace Reference	7
4.1.1 Detailed Description	11
4.1.2 Function Documentation	11
4.1.2.1 adjoint()	11
4.1.2.2 CartesianToCylindrical()	11
4.1.2.3 CartesianToPolar()	11
4.1.2.4 CartesianToSpherical()	11
4.1.2.5 cofactor()	12
4.1.2.6 conjugate()	12
4.1.2.7 crossProduct()	12
4.1.2.8 CylindricalToCartesian()	12
4.1.2.9 det()	12
4.1.2.10 dotProduct() [1/3]	13
4.1.2.11 dotProduct() [2/3]	13
4.1.2.12 dotProduct() [3/3]	13
4.1.2.13 inverse() [1/2]	13
4.1.2.14 inverse() [2/2]	13
4.1.2.15 isIdentity()	14
4.1.2.16 isZeroQuaternion()	14
4.1.2.17 length() [1/4]	14
4.1.2.18 length() [2/4]	14
4.1.2.19 length() [3/4]	14
4.1.2.20 length() [4/4]	14
4.1.2.21 norm() [1/3]	15
4.1.2.22 norm() [2/3]	15
4.1.2.23 norm() [3/3]	15
4.1.2.24 normalize()	15
4.1.2.25 operator*() [1/14]	15
4.1.2.26 operator*() [2/14]	16
4.1.2.27 operator*() [3/14]	16
4.1.2.28 operator*() [4/14]	16
4.1.2.29 operator*() [5/14]	16
4.1.2.30 operator*() [6/14]	16

4.1.2.31 operator*() [7/14]	17
4.1.2.32 operator*() [8/14]	17
4.1.2.33 operator*() [9/14]	17
4.1.2.34 operator*() [10/14]	17
4.1.2.35 operator*() [11/14]	17
4.1.2.36 operator*() [12/14]	18
4.1.2.37 operator*() [13/14]	18
4.1.2.38 operator*() [14/14]	18
4.1.2.39 operator+() [1/5]	18
4.1.2.40 operator+() [2/5]	18
4.1.2.41 operator+() [3/5]	19
4.1.2.42 operator+() [4/5]	19
4.1.2.43 operator+() [5/5]	19
4.1.2.44 operator-() [1/10]	19
4.1.2.45 operator-() [2/10]	19
4.1.2.46 operator-() [3/10]	20
4.1.2.47 operator-() [4/10]	20
4.1.2.48 operator-() [5/10]	20
4.1.2.49 operator-() [6/10]	20
4.1.2.50 operator-() [7/10]	20
4.1.2.51 operator-() [8/10]	21
4.1.2.52 operator-() [9/10]	21
4.1.2.53 operator-() [10/10]	21
4.1.2.54 operator/() [1/3]	21
4.1.2.55 operator/() [2/3]	21
4.1.2.56 operator/() [3/3]	22
4.1.2.57 PolarToCartesian()	22
4.1.2.58 Projection() [1/3]	22
4.1.2.59 Projection() [2/3]	22
4.1.2.60 Projection() [3/3]	22
4.1.2.61 quaternionRotationMatrixCol()	23
4.1.2.62 quaternionRotationMatrixRow()	23
4.1.2.63 rotate()	23
4.1.2.64 rotationQuaternion() [1/3]	23
4.1.2.65 rotationQuaternion() [2/3]	23
4.1.2.66 rotationQuaternion() [3/3]	24
4.1.2.67 scale()	24
4.1.2.68 SphericalToCartesian()	24
4.1.2.69 translate()	24
4.1.2.70 transpose()	24
4.1.2.71 zeroVector() [1/3]	25
4.1.2.72 zeroVector() [2/3]	25

4.1.2.73 zeroVector() [3/3]	25
<b>5 Class Documentation</b>	<b>27</b>
5.1 FAD3DMath::Matrix4x4 Class Reference	27
5.1.1 Detailed Description	28
5.1.2 Constructor & Destructor Documentation	28
5.1.2.1 Matrix4x4() [1/2]	28
5.1.2.2 Matrix4x4() [2/2]	28
5.1.3 Member Function Documentation	28
5.1.3.1 getElement()	28
5.1.3.2 isIdentity()	29
5.1.3.3 operator>() [1/2]	29
5.1.3.4 operator>() [2/2]	29
5.1.3.5 operator*=( ) [1/2]	29
5.1.3.6 operator*=( ) [2/2]	29
5.1.3.7 operator+=( )	30
5.1.3.8 operator-=( )	30
5.1.3.9 setElement()	30
5.1.3.10 setTolIdentity()	30
5.2 FAD3DMath::Quaternion Class Reference	30
5.2.1 Detailed Description	31
5.2.2 Constructor & Destructor Documentation	31
5.2.2.1 Quaternion() [1/4]	32
5.2.2.2 Quaternion() [2/4]	32
5.2.2.3 Quaternion() [3/4]	32
5.2.2.4 Quaternion() [4/4]	32
5.2.3 Member Function Documentation	32
5.2.3.1 operator*=( ) [1/2]	32
5.2.3.2 operator*=( ) [2/2]	33
5.2.3.3 operator+=( )	33
5.2.3.4 operator-=( )	33
5.2.3.5 scalar() [1/2]	33
5.2.3.6 scalar() [2/2]	33
5.2.3.7 vector()	33
5.2.3.8 x() [1/2]	34
5.2.3.9 x() [2/2]	34
5.2.3.10 y() [1/2]	34
5.2.3.11 y() [2/2]	34
5.2.3.12 z() [1/2]	34
5.2.3.13 z() [2/2]	34
5.3 FAD3DMath::Vector2D Class Reference	35
5.3.1 Detailed Description	35

5.3.2 Constructor & Destructor Documentation	35
5.3.2.1 Vector2D() [1/2]	35
5.3.2.2 Vector2D() [2/2]	36
5.3.3 Member Function Documentation	36
5.3.3.1 operator*=( )	36
5.3.3.2 operator+=( )	36
5.3.3.3 operator-=( )	36
5.3.3.4 operator/=( )	36
5.3.3.5 setX()	37
5.3.3.6 setY()	37
5.3.3.7 x()	37
5.3.3.8 y()	37
5.4 FAD3DMath::Vector3D Class Reference	37
5.4.1 Detailed Description	38
5.4.2 Constructor & Destructor Documentation	38
5.4.2.1 Vector3D() [1/2]	38
5.4.2.2 Vector3D() [2/2]	39
5.4.3 Member Function Documentation	39
5.4.3.1 operator*=( )	39
5.4.3.2 operator+=( )	39
5.4.3.3 operator-=( )	39
5.4.3.4 operator/=( )	39
5.4.3.5 setX()	40
5.4.3.6 setY()	40
5.4.3.7 setZ()	40
5.4.3.8 x()	40
5.4.3.9 y()	40
5.4.3.10 z()	40
5.5 FAD3DMath::Vector4D Class Reference	41
5.5.1 Detailed Description	41
5.5.2 Constructor & Destructor Documentation	41
5.5.2.1 Vector4D() [1/2]	42
5.5.2.2 Vector4D() [2/2]	42
5.5.3 Member Function Documentation	42
5.5.3.1 operator*=( )	42
5.5.3.2 operator+=( )	42
5.5.3.3 operator-=( )	42
5.5.3.4 operator/=( )	43
5.5.3.5 setW()	43
5.5.3.6 setX()	43
5.5.3.7 setY()	43
5.5.3.8 setZ()	43

---

5.5.3.9 w()	43
5.5.3.10 x()	44
5.5.3.11 y()	44
5.5.3.12 z()	44
<b>6 File Documentation</b>	<b>45</b>
6.1 FADirect3DMath.h File Reference	45
6.1.1 Detailed Description	49
6.2 FADirect3DMath.h	49
<b>Index</b>	<b>55</b>





# Chapter 1

## Namespace Index

### 1.1 Namespace List

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

<a href="#">FAD3DMath</a>	
Has <a href="#">Vector2D</a> , <a href="#">Vector3D</a> , <a href="#">Vector4D</a> , <a href="#">Matrix4x4</a> , and <a href="#">Quaternion</a> classes	7



## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">FAD3DMath::Matrix4x4</a>	
A matrix class used for 4x4 matrices and their manipulations . . . . .	<a href="#">27</a>
<a href="#">FAD3DMath::Quaternion</a> . . . . .	<a href="#">30</a>
<a href="#">FAD3DMath::Vector2D</a>	
A vector class used for 2D vectors/points and their manipulations . . . . .	<a href="#">35</a>
<a href="#">FAD3DMath::Vector3D</a>	
A vector class used for 3D vectors/points and their manipulations . . . . .	<a href="#">37</a>
<a href="#">FAD3DMath::Vector4D</a>	
A vector class used for 4D vectors/points and their manipulations . . . . .	<a href="#">41</a>



## Chapter 3

# File Index

### 3.1 File List

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

[FADirect3DMath.h](#)

File that has namespace FAMd3Dmath. Within the namespace are the classes Vector2D, Vector3D, Vector4D, Matrix4x4 and Quaternion . . . . . [45](#)



## Chapter 4

# Namespace Documentation

### 4.1 FAD3DMath Namespace Reference

Has [Vector2D](#), [Vector3D](#), [Vector4D](#), [Matrix4x4](#), and [Quaternion](#) classes.

#### Classes

- class [Matrix4x4](#)  
*A matrix class used for 4x4 matrices and their manipulations.*
- class [Quaternion](#)
- class [Vector2D](#)  
*A vector class used for 2D vectors/points and their manipulations.*
- class [Vector3D](#)  
*A vector class used for 3D vectors/points and their manipulations.*
- class [Vector4D](#)  
*A vector class used for 4D vectors/points and their manipulations.*

#### Functions

- bool **compareFloats** (float x, float y, float epsilon)
- bool **compareDoubles** (double x, double y, double epsilon)
- bool **zeroVector** (const [Vector2D](#) &a)  
*Returns true if a is the zero vector.*
- [Vector2D](#) **operator+** (const [Vector2D](#) &a, const [Vector2D](#) &b)  
*2D vector addition.*
- [Vector2D](#) **operator-** (const [Vector2D](#) &v)  
*2D vector negation.*
- [Vector2D](#) **operator-** (const [Vector2D](#) &a, const [Vector2D](#) &b)  
*2D vector subtraction.*
- [Vector2D](#) **operator\*** (const [Vector2D](#) &a, const float &k)  
*2D vector scalar multiplication. Returns  $a * k$ , where  $a$  is a vector and  $k$  is a scalar(float)*
- [Vector2D](#) **operator\*** (const float &k, const [Vector2D](#) &a)  
*2D vector scalar multiplication. Returns  $k * a$ , where  $a$  is a vector and  $k$  is a scalar(float)*
- [Vector2D](#) **operator/** (const [Vector2D](#) &a, const float &k)

2D vector scalar division. Returns  $a / k$ , where  $a$  is a vector and  $k$  is a scalar(float) If  $k = 0$  the returned vector is the zero vector.

- float **dotProduct** (const **Vector2D** &a, const **Vector2D** &b)  
Returns the dot product between two 2D vectors.
- float **length** (const **Vector2D** &v)  
Returns the length(magnitude) of the 2D vector v.
- **Vector2D norm** (const **Vector2D** &v)  
Normalizes the 2D vector v.
- **Vector2D PolarToCartesian** (const **Vector2D** &v)  
Converts the 2D vector v from polar coordinates to cartesian coordinates. v should = (r, theta(degrees)) The returned 2D vector = (x, y)
- **Vector2D CartesianToPolar** (const **Vector2D** &v)  
Converts the 2D vector v from cartesian coordinates to polar coordinates. v should = (x, y, z) If vx is zero then no conversion happens and v is returned.  
The returned 2D vector = (r, theta(degrees)).
- **Vector2D Projection** (const **Vector2D** &a, const **Vector2D** &b)  
Returns a 2D vector that is the projection of a onto b. If b is the zero vector a is returned.
- bool **zeroVector** (const **Vector3D** &a)  
Returns true if a is the zero vector.
- **Vector3D operator+** (const **Vector3D** &a, const **Vector3D** &b)  
3D vector addition.
- **Vector3D operator-** (const **Vector3D** &v)  
3D vector negation.
- **Vector3D operator-** (const **Vector3D** &a, const **Vector3D** &b)  
3D vector subtraction.
- **Vector3D operator\*** (const **Vector3D** &a, const float &k)  
3D vector scalar multiplication. Returns  $a * k$ , where a is a vector and k is a scalar(float)
- **Vector3D operator\*** (const float &k, const **Vector3D** &a)  
3D vector scalar multiplication. Returns  $k * a$ , where a is a vector and k is a scalar(float)
- **Vector3D operator/** (const **Vector3D** &a, const float &k)  
3D vector scalar division. Returns  $a / k$ , where a is a vector and k is a scalar(float) If  $k = 0$  the returned vector is the zero vector.
- float **dotProduct** (const **Vector3D** &a, const **Vector3D** &b)  
Returns the dot product between two 3D vectors.
- **Vector3D crossProduct** (const **Vector3D** &a, const **Vector3D** &b)  
Returns the cross product between two 3D vectors.
- float **length** (const **Vector3D** &v)  
Returns the length(magnitude) of the 3D vector v.
- **Vector3D norm** (const **Vector3D** &v)  
Normalizes the 3D vector v.
- **Vector3D CylindricalToCartesian** (const **Vector3D** &v)  
Converts the 3D vector v from cylindrical coordinates to cartesian coordinates. v should = (r, theta(degrees), z).  
The returned 3D vector = (x, y, z).
- **Vector3D CartesianToCylindrical** (const **Vector3D** &v)  
Converts the 3D vector v from cartesian coordinates to cylindrical coordinates. v should = (x, y, z).  
If vx is zero then no conversion happens and v is returned.  
The returned 3D vector = (r, theta(degrees), z).
- **Vector3D SphericalToCartesian** (const **Vector3D** &v)  
Converts the 3D vector v from spherical coordinates to cartesian coordinates. v should = (rho, phi(degrees), theta(degrees)).  
The returned 3D vector = (x, y, z)
- **Vector3D CartesianToSpherical** (const **Vector3D** &v)



Converts the 3D vector  $v$  from cartesian coordinates to spherical coordinates. If  $v$  is the zero vector or if  $v_x$  is zero then no conversion happens and  $v$  is returned.

The returned 3D vector =  $(r, \text{phi}(\text{degrees}), \text{theta}(\text{degrees}))$ .

- **Vector3D Projection** (const **Vector3D** &a, const **Vector3D** &b)  
Returns a 3D vector that is the projection of  $a$  onto  $b$ . If  $b$  is the zero vector  $a$  is returned.
- **bool zeroVector** (const **Vector4D** &a)  
Returns true if  $a$  is the zero vector.
- **Vector4D operator+** (const **Vector4D** &a, const **Vector4D** &b)  
4D vector addition.
- **Vector4D operator-** (const **Vector4D** &v)  
4D vector negation.
- **Vector4D operator-** (const **Vector4D** &a, const **Vector4D** &b)  
4D vector subtraction.
- **Vector4D operator\*** (const **Vector4D** &a, const float &k)  
4D vector scalar multiplication. Returns  $a * k$ , where  $a$  is a vector and  $k$  is a scalar(float)
- **Vector4D operator\*** (const float &k, const **Vector4D** &a)  
4D vector scalar multiplication. Returns  $k * a$ , where  $a$  is a vector and  $k$  is a scalar(float)
- **Vector4D operator/** (const **Vector4D** &a, const float &k)  
4D vector scalar division. Returns  $a / k$ , where  $a$  is a vector and  $k$  is a scalar(float) If  $k = 0$  the returned vector is the zero vector.
- **float dotProduct** (const **Vector4D** &a, const **Vector4D** &b)  
Returns the dot product between two 4D vectors.
- **float length** (const **Vector4D** &v)  
Returns the length(magnitude) of the 4D vector  $v$ .
- **Vector4D norm** (const **Vector4D** &v)  
Normalizes the 4D vector  $v$ .
- **Vector4D Projection** (const **Vector4D** &a, const **Vector4D** &b)  
Returns a 4D vector that is the projection of  $a$  onto  $b$ . If  $b$  is the zero vector  $a$  is returned.
- **Matrix4x4 operator+** (const **Matrix4x4** &m1, const **Matrix4x4** &m2)  
Adds the two given 4x4 matrices and returns a **Matrix4x4** object with the result.
- **Matrix4x4 operator-** (const **Matrix4x4** &m)  
Negates the 4x4 matrix  $m$ .
- **Matrix4x4 operator-** (const **Matrix4x4** &m1, const **Matrix4x4** &m2)  
Subtracts the two given 4x4 matrices and returns a **Matrix4x4** object with the result.
- **Matrix4x4 operator\*** (const **Matrix4x4** &m, const float &k)  
Multiplies the given 4x4 matrix with the given scalar and returns a **Matrix4x4** object with the result.
- **Matrix4x4 operator\*** (const float &k, const **Matrix4x4** &m)  
Multiplies the the given scalar with the given 4x4 matrix and returns a **Matrix4x4** object with the result.
- **Matrix4x4 operator\*** (const **Matrix4x4** &m1, const **Matrix4x4** &m2)  
Multiplies the two given 4x4 matrices and returns a **Matrix4x4** object with the result.
- **Vector4D operator\*** (const **Matrix4x4** &m, const **Vector4D** &v)  
Multiplies the given 4x4 matrix with the given 4D vector and returns a **Vector4D** object with the result. The vector is a column vector.
- **Vector4D operator\*** (const **Vector4D** &a, const **Matrix4x4** &m)  
Multiplies the given 4D vector with the given 4x4 matrix and returns a **Vector4D** object with the result. The vector is a row vector.
- **Matrix4x4 transpose** (const **Matrix4x4** &m)  
Returns the tranpose of the given matrix  $m$ .
- **Matrix4x4 translate** (const **Matrix4x4** &cm, float x, float y, float z)  
Construct a 4x4 translation matrix with the given floats and post-multiply it by the given matrix.  $cm = cm * \text{translate}$ .
- **Matrix4x4 scale** (const **Matrix4x4** &cm, float x, float y, float z)  
Construct a 4x4 scaling matrix with the given floats and post-multiply it by the given matrix.  $cm = cm * \text{scale}$ .

- **Matrix4x4 rotate** (const **Matrix4x4** &cm, float angle, float x, float y, float z)
 

Construct a 4x4 rotation matrix with the given angle (in degrees) and axis (x, y, z) and post-multiply it by the given matrix.  $cm = cm * rotate$ .
- double **det** (const **Matrix4x4** &m)
 

Returns the determinant of the given matrix.
- double **cofactor** (const **Matrix4x4** &m, unsigned int row, unsigned int col)
 

Returns the cofactor of the given row and col using the given matrix.
- **Matrix4x4 adjoint** (const **Matrix4x4** &m)
 

Returns the adjoint of the given matrix.
- **Matrix4x4 inverse** (const **Matrix4x4** &m)
 

Returns the inverse of the given matrix. If the matrix is noninvertible/singular, the identity matrix is returned.
- **Quaternion operator+** (const **Quaternion** &q1, const **Quaternion** &q2)
 

Returns a quaternion that has the result of  $q1 + q2$ .
- **Quaternion operator-** (const **Quaternion** &q)
 

Returns a quaternion that has the result of  $-q$ .
- **Quaternion operator-** (const **Quaternion** &q1, const **Quaternion** &q2)
 

Returns a quaternion that has the result of  $q1 - q2$ .
- **Quaternion operator\*** (float k, const **Quaternion** &q)
 

Returns a quaternion that has the result of  $k * q$ .
- **Quaternion operator\*** (const **Quaternion** &q, float k)
 

Returns a quaternion that has the result of  $q * k$ .
- **Quaternion operator\*** (const **Quaternion** &q1, const **Quaternion** &q2)
 

Returns a quaternion that has the result of  $q1 * q2$ .
- bool **isZeroQuaternion** (const **Quaternion** &q)
 

Returns true if quaternion  $q$  is a zero quaternion, false otherwise.
- bool **isIdentity** (const **Quaternion** &q)
 

Returns true if quaternion  $q$  is an identity quaternion, false otherwise.
- **Quaternion conjugate** (const **Quaternion** &q)
 

Returns the conjugate of quaternion  $q$ .
- float **length** (const **Quaternion** &q)
 

Returns the length of quaternion  $q$ .
- **Quaternion normalize** (const **Quaternion** &q)
 

Normalizes quaternion  $q$  and returns the normalized quaternion. If  $q$  is the zero quaternion then  $q$  is returned.
- **Quaternion inverse** (const **Quaternion** &q)
 

Returns the invese of quaternion  $q$ . If  $q$  is the zero quaternion then  $q$  is returned.
- **Quaternion rotationQuaternion** (float angle, float x, float y, float z)
 

Returns a quaternion from the axis-angle rotation representation. The angle should be given in degrees.
- **Quaternion rotationQuaternion** (float angle, const **Vector3D** &axis)
 

Returns a quaternion from the axis-angle rotation representation. The angle should be given in degrees.
- **Quaternion rotationQuaternion** (const **Vector4D** &angAxis)
 

Returns a quaternion from the axis-angle rotation representation. The  $x$  value in the 4D vector should be the angle(in degrees).  
The  $y, z$  and  $w$  value in the 4D vector should be the axis.
- **Matrix4x4 quaternionRotationMatrixCol** (const **Quaternion** &q)
 

Returns a matrix from the given quaterion for column vector-matrix multiplication. **Quaternion**  $q$  should be a unit quaternion.
- **Matrix4x4 quaternionRotationMatrixRow** (const **Quaternion** &q)
 

Returns a matrix from the given quaterion for row vector-matrix multiplication. **Quaternion**  $q$  should be a unit quaternion.

### 4.1.1 Detailed Description

Has [Vector2D](#), [Vector3D](#), [Vector4D](#), [Matrix4x4](#), and [Quaternion](#) classes.

FADIRECT3DMATH\_H FILE

### 4.1.2 Function Documentation

#### 4.1.2.1 adjoint()

```
Matrix4x4 FAD3DMath::adjoint (
    const Matrix4x4 & m )
```

Returns the adjoint of the given matrix.

#### 4.1.2.2 CartesianToCylindrical()

```
Vector3D FAD3DMath::CartesianToCylindrical (
    const Vector3D & v )
```

Converts the 3D vector *v* from cartesian coordinates to cylindrical coordinates. *v* should = (x, y, z).

If vx is zero then no conversion happens and *v* is returned.

The returned 3D vector = (r, theta(degrees), z).

#### 4.1.2.3 CartesianToPolar()

```
Vector2D FAD3DMath::CartesianToPolar (
    const Vector2D & v )
```

Converts the 2D vector *v* from cartesian coordinates to polar coordinates. *v* should = (x, y, z) If vx is zero then no conversion happens and *v* is returned.

The returned 2D vector = (r, theta(degrees)).

#### 4.1.2.4 CartesianToSpherical()

```
Vector3D FAD3DMath::CartesianToSpherical (
    const Vector3D & v )
```

Converts the 3D vector *v* from cartesian coordinates to spherical coordinates. If *v* is the zero vector or if vx is zero then no conversion happens and *v* is returned.

The returned 3D vector = (r, phi(degrees), theta(degrees)).

#### 4.1.2.5 cofactor()

```
double FAD3DMath::cofactor (
    const Matrix4x4 & m,
    unsigned int row,
    unsigned int col )
```

Returns the cofactor of the given row and col using the given matrix.

#### 4.1.2.6 conjugate()

```
Quaternion FAD3DMath::conjugate (
    const Quaternion & q )
```

Returns the conjugate of quaternion q.

#### 4.1.2.7 crossProduct()

```
Vector3D FAD3DMath::crossProduct (
    const Vector3D & a,
    const Vector3D & b )
```

Returns the cross product between two 3D vectors.

#### 4.1.2.8 CylindricalToCartesian()

```
Vector3D FAD3DMath::CylindricalToCartesian (
    const Vector3D & v )
```

Converts the 3D vector v from cylindrical coordinates to cartesian coordinates. v should = (r, theta(degrees), z).  
The returned 3D vector = (x, y ,z).

#### 4.1.2.9 det()

```
double FAD3DMath::det (
    const Matrix4x4 & m )
```

Returns the determinant of the given matrix.

**4.1.2.10 dotProduct()** [1/3]

```
float FAD3DMath::dotProduct (
    const Vector2D & a,
    const Vector2D & b )
```

Returns the dot product between two 2D vectors.

**4.1.2.11 dotProduct()** [2/3]

```
float FAD3DMath::dotProduct (
    const Vector3D & a,
    const Vector3D & b )
```

Returns the dot product between two 3D vectors.

**4.1.2.12 dotProduct()** [3/3]

```
float FAD3DMath::dotProduct (
    const Vector4D & a,
    const Vector4D & b )
```

Returns the dot product between two 4D vectors.

**4.1.2.13 inverse()** [1/2]

```
Matrix4x4 FAD3DMath::inverse (
    const Matrix4x4 & m )
```

Returns the inverse of the given matrix. If the matrix is noninvertible/singular, the identity matrix is returned.

**4.1.2.14 inverse()** [2/2]

```
Quaternion FAD3DMath::inverse (
    const Quaternion & q )
```

Returns the invese of quaternion q. If q is the zero quaternion then q is returned.

#### 4.1.2.15 isIdentity()

```
bool FAD3DMath::isIdentity (
    const Quaternion & q )
```

Returns true if quaternion q is an identity quaternion, false otherwise.

#### 4.1.2.16 isZeroQuaternion()

```
bool FAD3DMath::isZeroQuaternion (
    const Quaternion & q )
```

Returns true if quaternion q is a zero quaternion, false otherwise.

#### 4.1.2.17 length() [1/4]

```
float FAD3DMath::length (
    const Quaternion & q )
```

Returns the length of quaternion q.

#### 4.1.2.18 length() [2/4]

```
float FAD3DMath::length (
    const Vector2D & v )
```

Returns the length(magnitude) of the 2D vector v.

#### 4.1.2.19 length() [3/4]

```
float FAD3DMath::length (
    const Vector3D & v )
```

Returns the length(magnitude) of the 3D vector v.

#### 4.1.2.20 length() [4/4]

```
float FAD3DMath::length (
    const Vector4D & v )
```

Returns the length(magnitude) of the 4D vector v.

**4.1.2.21 norm()** [1/3]

```
Vector2D FAD3DMath::norm (
    const Vector2D & v )
```

Normalizes the 2D vector v.

If the 2D vector is the zero vector v is returned.

**4.1.2.22 norm()** [2/3]

```
Vector3D FAD3DMath::norm (
    const Vector3D & v )
```

Normalizes the 3D vector v.

If the 3D vector is the zero vector v is returned.

**4.1.2.23 norm()** [3/3]

```
Vector4D FAD3DMath::norm (
    const Vector4D & v )
```

Normalizes the 4D vector v.

If the 4D vector is the zero vector v is returned.

**4.1.2.24 normalize()**

```
Quaternion FAD3DMath::normalize (
    const Quaternion & q )
```

Normalizes quaternion q and returns the normalized quaternion. If q is the zero quaternion then q is returned.

**4.1.2.25 operator\*()** [1/14]

```
Matrix4x4 FAD3DMath::operator* (
    const float & k,
    const Matrix4x4 & m )
```

Multiplies the the given scalar with the given 4x4 matrix and returns a [Matrix4x4](#) object with the result.

**4.1.2.26 operator\*() [2/14]**

```
Vector2D FAD3DMath::operator* (
    const float & k,
    const Vector2D & a )
```

2D vector scalar multiplication. Returns  $k * a$ , where  $a$  is a vector and  $k$  is a scalar(float)

**4.1.2.27 operator\*() [3/14]**

```
Vector3D FAD3DMath::operator* (
    const float & k,
    const Vector3D & a )
```

3D vector scalar multiplication. Returns  $k * a$ , where  $a$  is a vector and  $k$  is a scalar(float)

**4.1.2.28 operator\*() [4/14]**

```
Vector4D FAD3DMath::operator* (
    const float & k,
    const Vector4D & a )
```

4D vector scalar multiplication. Returns  $k * a$ , where  $a$  is a vector and  $k$  is a scalar(float)

**4.1.2.29 operator\*() [5/14]**

```
Matrix4x4 FAD3DMath::operator* (
    const Matrix4x4 & m,
    const float & k )
```

Multiplies the given 4x4 matrix with the given scalar and returns a [Matrix4x4](#) object with the result.

**4.1.2.30 operator\*() [6/14]**

```
Vector4D FAD3DMath::operator* (
    const Matrix4x4 & m,
    const Vector4D & v )
```

Multiplies the given 4x4 matrix with the given 4D vector and returns a [Vector4D](#) object with the result. The vector is a column vector.



**4.1.2.31 operator\*() [7/14]**

```
Matrix4x4 FAD3DMath::operator* (
    const Matrix4x4 & m1,
    const Matrix4x4 & m2 )
```

Multiplies the two given 4x4 matrices and returns a [Matrix4x4](#) object with the result.

**4.1.2.32 operator\*() [8/14]**

```
Quaternion FAD3DMath::operator* (
    const Quaternion & q,
    float k )
```

Returns a quaternion that has the result of  $q * k$ .

**4.1.2.33 operator\*() [9/14]**

```
Quaternion FAD3DMath::operator* (
    const Quaternion & q1,
    const Quaternion & q2 )
```

Returns a quaternion that has the result of  $q1 * q2$ .

**4.1.2.34 operator\*() [10/14]**

```
Vector2D FAD3DMath::operator* (
    const Vector2D & a,
    const float & k )
```

2D vector scalar multiplication. Returns  $a * k$ , where  $a$  is a vector and  $k$  is a scalar(float)

**4.1.2.35 operator\*() [11/14]**

```
Vector3D FAD3DMath::operator* (
    const Vector3D & a,
    const float & k )
```

3D vector scalar multiplication. Returns  $a * k$ , where  $a$  is a vector and  $k$  is a scalar(float)

#### 4.1.2.36 operator\*() [12/14]

```
Vector4D FAD3DMath::operator* (
    const Vector4D & a,
    const float & k )
```

4D vector scalar multiplication. Returns  $a * k$ , where  $a$  is a vector and  $k$  is a scalar(float)

#### 4.1.2.37 operator\*() [13/14]

```
Vector4D FAD3DMath::operator* (
    const Vector4D & v,
    const Matrix4x4 & m )
```

Multiplies the given 4D vector with the given 4x4 matrix and returns a [Vector4D](#) object with the result. The vector is a row vector.

#### 4.1.2.38 operator\*() [14/14]

```
Quaternion FAD3DMath::operator* (
    float k,
    const Quaternion & q )
```

Returns a quaternion that has the result of  $k * q$ .

#### 4.1.2.39 operator+() [1/5]

```
Matrix4x4 FAD3DMath::operator+ (
    const Matrix4x4 & m1,
    const Matrix4x4 & m2 )
```

Adds the two given 4x4 matrices and returns a [Matrix4x4](#) object with the result.

#### 4.1.2.40 operator+() [2/5]

```
Quaternion FAD3DMath::operator+ (
    const Quaternion & q1,
    const Quaternion & q2 )
```

Returns a quaternion that has the result of  $q1 + q2$ .

**4.1.2.41 operator+()** [3/5]

```
Vector2D FAD3DMath::operator+ (
    const Vector2D & a,
    const Vector2D & b )
```

2D vector addition.

**4.1.2.42 operator+()** [4/5]

```
Vector3D FAD3DMath::operator+ (
    const Vector3D & a,
    const Vector3D & b )
```

3D vector addition.

**4.1.2.43 operator+()** [5/5]

```
Vector4D FAD3DMath::operator+ (
    const Vector4D & a,
    const Vector4D & b )
```

4D vector addition.

**4.1.2.44 operator-()** [1/10]

```
Matrix4x4 FAD3DMath::operator- (
    const Matrix4x4 & m )
```

Negates the 4x4 matrix m.

**4.1.2.45 operator-()** [2/10]

```
Matrix4x4 FAD3DMath::operator- (
    const Matrix4x4 & m1,
    const Matrix4x4 & m2 )
```

Subtracts the two given 4x4 matrices and returns a [Matrix4x4](#) object with the result.

**4.1.2.46 operator-() [3/10]**

```
Quaternion FAD3DMath::operator- (
    const Quaternion & q )
```

Returns a quaternion that has the result of -q.

**4.1.2.47 operator-() [4/10]**

```
Quaternion FAD3DMath::operator- (
    const Quaternion & q1,
    const Quaternion & q2 )
```

Returns a quaternion that has the result of q1 - q2.

**4.1.2.48 operator-() [5/10]**

```
Vector2D FAD3DMath::operator- (
    const Vector2D & a,
    const Vector2D & b )
```

2D vector subtraction.

**4.1.2.49 operator-() [6/10]**

```
Vector2D FAD3DMath::operator- (
    const Vector2D & v )
```

2D vector negation.

**4.1.2.50 operator-() [7/10]**

```
Vector3D FAD3DMath::operator- (
    const Vector3D & a,
    const Vector3D & b )
```

3D vector subtraction.

**4.1.2.51 operator-() [8/10]**

```
Vector3D FAD3DMath::operator- (
    const Vector3D & v )
```

3D vector negation.

**4.1.2.52 operator-() [9/10]**

```
Vector4D FAD3DMath::operator- (
    const Vector4D & a,
    const Vector4D & b )
```

4D vector subtraction.

**4.1.2.53 operator-() [10/10]**

```
Vector4D FAD3DMath::operator- (
    const Vector4D & v )
```

4D vector negation.

**4.1.2.54 operator/() [1/3]**

```
Vector2D FAD3DMath::operator/ (
    const Vector2D & a,
    const float & k )
```

2D vector scalar division. Returns  $a / k$ , where  $a$  is a vector and  $k$  is a scalar(float) If  $k = 0$  the returned vector is the zero vector.

**4.1.2.55 operator/() [2/3]**

```
Vector3D FAD3DMath::operator/ (
    const Vector3D & a,
    const float & k )
```

3D vector scalar division. Returns  $a / k$ , where  $a$  is a vector and  $k$  is a scalar(float) If  $k = 0$  the returned vector is the zero vector.

**4.1.2.56 operator/()** [3/3]

```
Vector4D FAD3DMath::operator/ (
    const Vector4D & a,
    const float & k )
```

4D vector scalar division. Returns  $a / k$ , where  $a$  is a vector and  $k$  is a scalar(float) If  $k = 0$  the returned vector is the zero vector.

**4.1.2.57 PolarToCartesian()**

```
Vector2D FAD3DMath::PolarToCartesian (
    const Vector2D & v )
```

Converts the 2D vector  $v$  from polar coordinates to cartesian coordinates.  $v$  should = (r, theta(degrees)) The returned 2D vector = (x, y)

**4.1.2.58 Projection()** [1/3]

```
Vector2D FAD3DMath::Projection (
    const Vector2D & a,
    const Vector2D & b )
```

Returns a 2D vector that is the projection of  $a$  onto  $b$ . If  $b$  is the zero vector  $a$  is returned.

**4.1.2.59 Projection()** [2/3]

```
Vector3D FAD3DMath::Projection (
    const Vector3D & a,
    const Vector3D & b )
```

Returns a 3D vector that is the projection of  $a$  onto  $b$ . If  $b$  is the zero vector  $a$  is returned.

**4.1.2.60 Projection()** [3/3]

```
Vector4D FAD3DMath::Projection (
    const Vector4D & a,
    const Vector4D & b )
```

Returns a 4D vector that is the projection of  $a$  onto  $b$ . If  $b$  is the zero vector  $a$  is returned.

**4.1.2.61 quaternionRotationMatrixCol()**

```
Matrix4x4 FAD3DMath::quaternionRotationMatrixCol (
    const Quaternion & q )
```

Returns a matrix from the given quaterion for column vector-matrix multiplication. [Quaternion](#) q should be a unit quaternion.

**4.1.2.62 quaternionRotationMatrixRow()**

```
Matrix4x4 FAD3DMath::quaternionRotationMatrixRow (
    const Quaternion & q )
```

Returns a matrix from the given quaterion for row vector-matrix multiplication. [Quaternion](#) q should be a unit quaternion.

**4.1.2.63 rotate()**

```
Matrix4x4 FAD3DMath::rotate (
    const Matrix4x4 & cm,
    float angle,
    float x,
    float y,
    float z )
```

Construct a 4x4 rotation matrix with the given angle (in degrees) and axis (x, y, z) and post-multiply it by the given matrix. cm = cm \* rotate.

.

**4.1.2.64 rotationQuaternion() [1/3]**

```
Quaternion FAD3DMath::rotationQuaternion (
    const Vector4D & angAxis )
```

Returns a quaternion from the axis-angle rotation representation. The x value in the 4D vector should be the angle(in degrees).

The y, z and w value in the 4D vector should be the axis.

**4.1.2.65 rotationQuaternion() [2/3]**

```
Quaternion FAD3DMath::rotationQuaternion (
    float angle,
    const Vector3D & axis )
```

Returns a quaternion from the axis-angle rotation representation. The angle should be given in degrees.

#### 4.1.2.66 rotationQuaternion() [3/3]

```
Quaternion FAD3DMath::rotationQuaternion (
    float angle,
    float x,
    float y,
    float z )
```

Returns a quaternion from the axis-angle rotation representation. The angle should be given in degrees.

#### 4.1.2.67 scale()

```
Matrix4x4 FAD3DMath::scale (
    const Matrix4x4 & cm,
    float x,
    float y,
    float z )
```

Construct a 4x4 scaling matrix with the given floats and post-multiply it by the given matrix.  $cm = cm * scale$ .

#### 4.1.2.68 SphericalToCartesian()

```
Vector3D FAD3DMath::SphericalToCartesian (
    const Vector3D & v )
```

Converts the 3D vector  $v$  from spherical coordinates to cartesian coordinates.  $v$  should = (pho, phi(degrees), theta(degrees)).

The returned 3D vector = (x, y, z)

#### 4.1.2.69 translate()

```
Matrix4x4 FAD3DMath::translate (
    const Matrix4x4 & cm,
    float x,
    float y,
    float z )
```

Construct a 4x4 translation matrix with the given floats and post-multiply it by the given matrix.  $cm = cm * translate$ .

#### 4.1.2.70 transpose()

```
Matrix4x4 FAD3DMath::transpose (
    const Matrix4x4 & m )
```

Returns the tranpose of the given matrix  $m$ .



**4.1.2.71 zeroVector()** [1/3]

```
bool FAD3DMath::zeroVector (
    const Vector2D & a )
```

Returns true if a is the zero vector.

**4.1.2.72 zeroVector()** [2/3]

```
bool FAD3DMath::zeroVector (
    const Vector3D & a )
```

Returns true if a is the zero vector.

**4.1.2.73 zeroVector()** [3/3]

```
bool FAD3DMath::zeroVector (
    const Vector4D & a )
```

Returns true if a is the zero vector.



## Chapter 5

# Class Documentation

### 5.1 FAD3DMath::Matrix4x4 Class Reference

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

```
#include "FADirect3DMath.h"
```

#### Public Member Functions

- [Matrix4x4](#) ()  
*Default Constructor.*
- [Matrix4x4](#) (float a[ ][4])  
*Overloaded Constructor.*
- float [getElement](#) (unsigned int row, unsigned int col) const  
*Returns the element at the given (row, col). The row and col values should be between [0,3]. If any of them are out of that range, the first element will be returned.*
- const float & [operator\(\)](#) (unsigned int row, unsigned int col) const  
*Returns a constant reference to the element at the given (row, col). The row and col values should be between [0,3]. If any of them are out of that range, the first element will be returned.*
- float & [operator\(\)](#) (unsigned int row, unsigned int col)  
*Returns a reference to the element at the given (row, col). The row and col values should be between [0,3]. If any of them are out of that range, the first element will be returned.*
- void [setElement](#) (unsigned int row, unsigned int col, float val)  
*Set the element at the given (row, col) to the given val. The row and col values should be between [0,3]. If any of them are out of that range, the first element will be set to the given val.*
- void [setTolIdentity](#) ()  
*Sets the matrix to the identity matrix.*
- bool [isIdentity](#) ()  
*Returns true if the matrix is the identity matrix, false otherwise.*
- [Matrix4x4](#) & [operator+=](#) (const [Matrix4x4](#) &m)  
*Adds this 4x4 matrix with given matrix m and stores the result in this 4x4 matrix.*
- [Matrix4x4](#) & [operator-=](#) (const [Matrix4x4](#) &m)  
*Subtracts this 4x4 matrix with given matrix m and stores the result in this 4x4 matrix.*
- [Matrix4x4](#) & [operator\\*=](#) (const float &k)  
*Multiplies this 4x4 matrix with given scalar k and stores the result in this 4x4 matrix.*
- [Matrix4x4](#) & [operator\\*=](#) (const [Matrix4x4](#) &m)  
*Multiplies this 4x4 matrix with given matrix m and stores the result in this 4x4 matrix.*

### 5.1.1 Detailed Description

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

The datatype for the components is float.

The 4x4 matrix is treated as a row-major matrix.

### 5.1.2 Constructor & Destructor Documentation

#### 5.1.2.1 Matrix4x4() [1/2]

```
FAD3DMath::Matrix4x4::Matrix4x4 ( )
```

Default Constructor.

Creates a new 4x4 identity matrix.

#### 5.1.2.2 Matrix4x4() [2/2]

```
FAD3DMath::Matrix4x4::Matrix4x4 (
    float a[][4] )
```

Overloaded Constructor.

Creates a new 4x4 matrix with elements initialized to the given 2D array.  
If the passed in 2D array isn't a 4x4 matrix, the behavior is undefined.

### 5.1.3 Member Function Documentation

#### 5.1.3.1 getElement()

```
float FAD3DMath::Matrix4x4::getElement (
    unsigned int row,
    unsigned int col ) const
```

Returns the element at the given (row, col). The row and col values should be between [0,3]. If any of them are out of that range, the first element will be returned.

### 5.1.3.2 isIdentity()

```
bool FAD3DMath::Matrix4x4::isIdentity ( )
```

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

### 5.1.3.3 operator() [1/2]

```
float & FAD3DMath::Matrix4x4::operator() (
    unsigned int row,
    unsigned int col )
```

Returns a reference to the element at the given (row, col). The row and col values should be between [0,3]. If any of them are out of that range, the first element will be returned.

### 5.1.3.4 operator() [2/2]

```
const float & FAD3DMath::Matrix4x4::operator() (
    unsigned int row,
    unsigned int col ) const
```

Returns a constant reference to the element at the given (row, col). The row and col values should be between [0,3]. If any of them are out of that range, the first element will be returned.

### 5.1.3.5 operator\*=( ) [1/2]

```
Matrix4x4 & FAD3DMath::Matrix4x4::operator*= (
    const float & k )
```

Multiplies this 4x4 matrix with given scalar k and stores the result in this 4x4 matrix.

### 5.1.3.6 operator\*=( ) [2/2]

```
Matrix4x4 & FAD3DMath::Matrix4x4::operator*= (
    const Matrix4x4 & m )
```

Multiplies this 4x4 matrix with given matrix m and stores the result in this 4x4 matrix.

#### 5.1.3.7 operator+=()

```
Matrix4x4 & FAD3DMath::Matrix4x4::operator+= (
    const Matrix4x4 & m )
```

Adds this 4x4 matrix with given matrix m and stores the result in this 4x4 matrix.

#### 5.1.3.8 operator-=()

```
Matrix4x4 & FAD3DMath::Matrix4x4::operator-= (
    const Matrix4x4 & m )
```

Subtracts this 4x4 matrix with given matrix m and stores the result in this 4x4 matrix.

#### 5.1.3.9 setElement()

```
void FAD3DMath::Matrix4x4::setElement (
    unsigned int row,
    unsigned int col,
    float val )
```

Set the element at the given (row, col) to the given val. The row and col values should be between [0,3]. If any of them are out of that range, the first element will be set to the given val.

#### 5.1.3.10 setToIdentity()

```
void FAD3DMath::Matrix4x4::setToIdentity ( )
```

Sets the matrix to the identity matrix.

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

- [FADirect3DMath.h](#)
- FADirect3DMath.cpp

## 5.2 FAD3DMath::Quaternion Class Reference

```
#include "FADirect3DMath.h"
```

## Public Member Functions

- [Quaternion](#) ()  
*Default Constructor.*
- [Quaternion](#) (float [scalar](#), float [x](#), float [y](#), float [z](#))  
*Overloaded Constructor.*
- [Quaternion](#) (float [scalar](#), const [Vector3D](#) &[v](#))  
*Overloaded Constructor.*
- [Quaternion](#) (const [Vector4D](#) &[v](#))  
*Overloaded Constructor.*
- float & [scalar](#) ()  
*Returns a reference to the scalar component of the quaternion.*
- const float & [scalar](#) () const  
*Returns a const reference to the scalar component of the quaternion.*
- float & [x](#) ()  
*Returns a reference to the x value of the vector component in the quaternion.*
- const float & [x](#) () const  
*Returns a const reference to the x value of the vector component in the quaternion.*
- float & [y](#) ()  
*Returns a reference to the y value of the vector component in the quaternion.*
- const float & [y](#) () const  
*Returns a const reference to the y value of the vector component in the quaternion.*
- float & [z](#) ()  
*Returns a reference to the z value of the vector component in the quaternion.*
- const float & [z](#) () const  
*Returns a const reference to the z value of the vector component in the quaternion.*
- [Vector3D](#) [vector](#) ()  
*Returns the vector component of the quaternion.*
- [Quaternion](#) & [operator+=](#) (const [Quaternion](#) &[q](#))  
*Adds this quaternion to quaterion q and stores the result in this quaternion.*
- [Quaternion](#) & [operator-=](#) (const [Quaternion](#) &[q](#))  
*Subtracts this quaternion by quaterion q and stores the result in this quaternion.*
- [Quaternion](#) & [operator\\*=](#) (float [k](#))  
*Multiplies this quaternion by flaot k and stores the result in this quaternion.*
- [Quaternion](#) & [operator\\*=](#) (const [Quaternion](#) &[q](#))  
*Multiplies this quaternion by quaterion q and stores the result in this quaternion.*

### 5.2.1 Detailed Description

The datatype for the components is float.

### 5.2.2 Constructor & Destructor Documentation

### 5.2.2.1 Quaternion() [1/4]

```
FAD3DMath::Quaternion::Quaternion ( )
```

Default Constructor.

Constructs an identity quaternion.

### 5.2.2.2 Quaternion() [2/4]

```
FAD3DMath::Quaternion::Quaternion (
    float scalar,
    float x,
    float y,
    float z )
```

Overloaded Constructor.

Constructs a quaternion with the given values.

### 5.2.2.3 Quaternion() [3/4]

```
FAD3DMath::Quaternion::Quaternion (
    float scalar,
    const Vector3D & v )
```

Overloaded Constructor.

Constructs a quaternion with the given values.

### 5.2.2.4 Quaternion() [4/4]

```
FAD3DMath::Quaternion::Quaternion (
    const Vector4D & v )
```

Overloaded Constructor.

Constructs a quaternion with the given values in the 4D vector.

The x value in the 4D vector should be the scalar. The y, z and w value in the 4D vector should be the axis.

## 5.2.3 Member Function Documentation

### 5.2.3.1 operator\*=( ) [1/2]

```
Quaternion & FAD3DMath::Quaternion::operator*= (
    const Quaternion & q )
```

Multiplies this quaternion by quaternion q and stores the result in this quaternion.



**5.2.3.2 operator\*=( ) [2/2]**

```
Quaternion & FAD3DMath::Quaternion::operator*= (
    float k )
```

Multiplies this quaternion by float k and stores the result in this quaternion.

**5.2.3.3 operator+=( )**

```
Quaternion & FAD3DMath::Quaternion::operator+= (
    const Quaternion & q )
```

Adds this quaternion to quaternion q and stores the result in this quaternion.

**5.2.3.4 operator-=( )**

```
Quaternion & FAD3DMath::Quaternion::operator-= (
    const Quaternion & q )
```

Subtracts this quaternion by quaternion q and stores the result in this quaternion.

**5.2.3.5 scalar() [1/2]**

```
float & FAD3DMath::Quaternion::scalar ( )
```

Returns a reference to the scalar component of the quaternion.

**5.2.3.6 scalar() [2/2]**

```
const float & FAD3DMath::Quaternion::scalar ( ) const
```

Returns a const reference to the scalar component of the quaternion.

**5.2.3.7 vector()**

```
Vector3D FAD3DMath::Quaternion::vector ( )
```

Returns the vector component of the quaternion.

#### 5.2.3.8 **x()** [1/2]

```
float & FAD3DMath::Quaternion::x ( )
```

Returns a reference to the x value of the vector component in the quaternion.

#### 5.2.3.9 **x()** [2/2]

```
const float & FAD3DMath::Quaternion::x ( ) const
```

Returns a const reference to the x value of the vector component in the quaternion.

#### 5.2.3.10 **y()** [1/2]

```
float & FAD3DMath::Quaternion::y ( )
```

Returns a reference to the y value of the vector component in the quaternion.

#### 5.2.3.11 **y()** [2/2]

```
const float & FAD3DMath::Quaternion::y ( ) const
```

Returns a const reference to the y value of the vector component in the quaternion.

#### 5.2.3.12 **z()** [1/2]

```
float & FAD3DMath::Quaternion::z ( )
```

Returns a reference to the z value of the vector component in the quaternion.

#### 5.2.3.13 **z()** [2/2]

```
const float & FAD3DMath::Quaternion::z ( ) const
```

Returns a const reference to the z value of the vector component in the quaternion.

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

- [FADirect3DMath.h](#)
- [FADirect3DMath.cpp](#)

## 5.3 FAD3DMath::Vector2D Class Reference

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

```
#include "FADirect3DMath.h"
```

### Public Member Functions

- [Vector2D](#) ()  
*Default Constructor.*
- [Vector2D](#) (float *x*, float *y*)  
*Overloaded Constructor.*
- float *x* () const  
*Returns the x component.*
- float *y* () const  
*Returns the y component.*
- void [setX](#) (float *x*)  
*Sets the x component to the provided float.*
- void [setY](#) (float *y*)  
*Sets the y component to the provided float.*
- [Vector2D](#) & [operator+=](#) (const [Vector2D](#) &*b*)  
*2D vector addition through overloading operator +=.*
- [Vector2D](#) & [operator-=](#) (const [Vector2D](#) &*b*)  
*2D vector subtraction through overloading operator -=.*
- [Vector2D](#) & [operator\\*=](#) (const float &*k*)  
*2D vector scalar multiplication through overloading operator \*=.*
- [Vector2D](#) & [operator/=](#) (const float &*k*)  
*2D vector scalar division through overloading operator /=.*

### 5.3.1 Detailed Description

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

The datatype for the components is float

### 5.3.2 Constructor & Destructor Documentation

#### 5.3.2.1 Vector2D() [1/2]

```
FAD3DMath::Vector2D::Vector2D ( )
```

Default Constructor.

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

### 5.3.2.2 Vector2D() [2/2]

```
FAD3DMath::Vector2D::Vector2D (
    float x,
    float y )
```

Overloaded Constructor.

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

## 5.3.3 Member Function Documentation

### 5.3.3.1 operator\*=( )

```
Vector2D & FAD3DMath::Vector2D::operator*= (
    const float & k )
```

2D vector scalar multiplication through overloading operator \*.

### 5.3.3.2 operator+=( )

```
Vector2D & FAD3DMath::Vector2D::operator+= (
    const Vector2D & b )
```

2D vector addition through overloading operator +.

### 5.3.3.3 operator-=( )

```
Vector2D & FAD3DMath::Vector2D::operator-= (
    const Vector2D & b )
```

2D vector subtraction through overloading operator -.

### 5.3.3.4 operator/=( )

```
Vector2D & FAD3DMath::Vector2D::operator/= (
    const float & k )
```

2D vector scalar division through overloading operator /.

If k is zero, the vector is unchanged.

#### 5.3.3.5 setX()

```
void FAD3DMath::Vector2D::setX (
    float x )
```

Sets the x component to the provided float.

#### 5.3.3.6 setY()

```
void FAD3DMath::Vector2D::setY (
    float y )
```

Sets the y component to the provided float.

#### 5.3.3.7 x()

```
float FAD3DMath::Vector2D::x ( ) const
```

Returns the x component.

#### 5.3.3.8 y()

```
float FAD3DMath::Vector2D::y ( ) const
```

Returns the y component.

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

- [FADirect3DMath.h](#)
- FADirect3DMath.cpp

## 5.4 FAD3DMath::Vector3D Class Reference

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

```
#include "FADirect3DMath.h"
```

## Public Member Functions

- [Vector3D](#) ()  
*Default Constructor.*
- [Vector3D](#) (float [x](#), float [y](#), float [z](#))  
*Overloaded Constructor.*
- float [x](#) () const  
*Returns the x component.*
- float [y](#) () const  
*Returns the y component.*
- float [z](#) () const  
*Returns the z component.*
- void [setX](#) (float [x](#))  
*Sets the x component to the provided float.*
- void [setY](#) (float [y](#))  
*Sets the y component to the provided float.*
- void [setZ](#) (float [z](#))  
*Sets the z component to the provided float.*
- [Vector3D](#) & [operator+=](#) (const [Vector3D](#) &[b](#))  
*3D vector addition through overloading operator +=.*
- [Vector3D](#) & [operator-=](#) (const [Vector3D](#) &[b](#))  
*3D vector subtraction through overloading operator -=.*
- [Vector3D](#) & [operator\\*=](#) (const float &[k](#))  
*3D vector scalar multiplication through overloading operator \*=.*
- [Vector3D](#) & [operator/=](#) (const float &[k](#))  
*3D vector scalar division through overloading operator /=.*

### 5.4.1 Detailed Description

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

The datatype for the components is float

### 5.4.2 Constructor & Destructor Documentation

#### 5.4.2.1 [Vector3D](#)() [1/2]

```
FAD3DMath::Vector3D::Vector3D ( )
```

Default Constructor.

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

### 5.4.2.2 Vector3D() [2/2]

```
FAD3DMath::Vector3D::Vector3D (
    float x,
    float y,
    float z )
```

Overloaded Constructor.

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

## 5.4.3 Member Function Documentation

### 5.4.3.1 operator\*=( )

```
Vector3D & FAD3DMath::Vector3D::operator*= (
    const float & k )
```

3D vector scalar multiplication through overloading operator \*.

### 5.4.3.2 operator+=( )

```
Vector3D & FAD3DMath::Vector3D::operator+= (
    const Vector3D & b )
```

3D vector addition through overloading operator +.

### 5.4.3.3 operator-=( )

```
Vector3D & FAD3DMath::Vector3D::operator-= (
    const Vector3D & b )
```

3D vector subtraction through overloading operator -.

### 5.4.3.4 operator/=( )

```
Vector3D & FAD3DMath::Vector3D::operator/= (
    const float & k )
```

3D vector scalar division through overloading operator /.

If k is zero, the vector is unchanged.

#### 5.4.3.5 setX()

```
void FAD3DMath::Vector3D::setX (
    float x )
```

Sets the x component to the provided float.

#### 5.4.3.6 setY()

```
void FAD3DMath::Vector3D::setY (
    float y )
```

Sets the y component to the provided float.

#### 5.4.3.7 setZ()

```
void FAD3DMath::Vector3D::setZ (
    float z )
```

Sets the z component to the provided float.

#### 5.4.3.8 x()

```
float FAD3DMath::Vector3D::x ( ) const
```

Returns the x component.

#### 5.4.3.9 y()

```
float FAD3DMath::Vector3D::y ( ) const
```

Returns the y component.

#### 5.4.3.10 z()

```
float FAD3DMath::Vector3D::z ( ) const
```

Returns the z component.

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

- [FADirect3DMath.h](#)
- [FADirect3DMath.cpp](#)



## 5.5 FAD3DMath::Vector4D Class Reference

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

```
#include "FADirect3DMath.h"
```

### Public Member Functions

- [Vector4D](#) ()  
*Default Constructor.*
- [Vector4D](#) (float [x](#), float [y](#), float [z](#), float [w](#))  
*Overloaded Constructor.*
- float [x](#) () const  
*Returns the x component.*
- float [y](#) () const  
*Returns the y component.*
- float [z](#) () const  
*Returns the z component.*
- float [w](#) () const  
*Returns the w component.*
- void [setX](#) (float [x](#))  
*Sets the x component to the provided float.*
- void [setY](#) (float [y](#))  
*Sets the y component to the provided float.*
- void [setZ](#) (float [z](#))  
*Sets the z component to the provided float.*
- void [setW](#) (float [w](#))  
*Sets the w component to the provided float.*
- [Vector4D](#) & [operator+=](#) (const [Vector4D](#) &b)  
*4D vector addition through overloading operator +=.*
- [Vector4D](#) & [operator-=](#) (const [Vector4D](#) &b)  
*4D vector subtraction through overloading operator -=.*
- [Vector4D](#) & [operator\\*=](#) (const float &k)  
*4D vector scalar multiplication through overloading operator \*=.*
- [Vector4D](#) & [operator/=](#) (const float &k)  
*4D vector scalar division through overloading operator /=.*

### 5.5.1 Detailed Description

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

The datatype for the components is float

### 5.5.2 Constructor & Destructor Documentation

### 5.5.2.1 Vector4D() [1/2]

```
FAD3DMath::Vector4D::Vector4D ( )
```

Default Constructor.

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

### 5.5.2.2 Vector4D() [2/2]

```
FAD3DMath::Vector4D::Vector4D (
    float x,
    float y,
    float z,
    float w )
```

Overloaded Constructor.

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

## 5.5.3 Member Function Documentation

### 5.5.3.1 operator\*=( )

```
Vector4D & FAD3DMath::Vector4D::operator*= (
    const float & k )
```

4D vector scalar multiplication through overloading operator \*.

### 5.5.3.2 operator+=( )

```
Vector4D & FAD3DMath::Vector4D::operator+= (
    const Vector4D & b )
```

4D vector addition through overloading operator +.

### 5.5.3.3 operator-=( )

```
Vector4D & FAD3DMath::Vector4D::operator-= (
    const Vector4D & b )
```

4D vector subtraction through overloading operator -.

#### 5.5.3.4 operator/=( )

```
Vector4D & FAD3DMath::Vector4D::operator/= (
    const float & k )
```

4D vector scalar division through overloading operator /.

If k is zero, the vector is unchanged.

#### 5.5.3.5 setW()

```
void FAD3DMath::Vector4D::setW (
    float w )
```

Sets the w component to the provided float.

#### 5.5.3.6 setX()

```
void FAD3DMath::Vector4D::setX (
    float x )
```

Sets the x component to the provided float.

#### 5.5.3.7 setY()

```
void FAD3DMath::Vector4D::setY (
    float y )
```

Sets the y component to the provided float.

#### 5.5.3.8 setZ()

```
void FAD3DMath::Vector4D::setZ (
    float z )
```

Sets the z component to the provided float.

#### 5.5.3.9 w()

```
float FAD3DMath::Vector4D::w ( ) const
```

Returns the w component.

**5.5.3.10 x()**

```
float FAD3DMath::Vector4D::x ( ) const
```

Returns the x component.

**5.5.3.11 y()**

```
float FAD3DMath::Vector4D::y ( ) const
```

Returns the y component.

**5.5.3.12 z()**

```
float FAD3DMath::Vector4D::z ( ) const
```

Returns the z component.

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

- [FADirect3DMath.h](#)
- FADirect3DMath.cpp

## Chapter 6

# File Documentation

### 6.1 FADirect3DMath.h File Reference

File that has namespace FAMD3Dath. Withn the namespace are the classes Vector2D, Vector3D, Vector4D, Matrix4x4 and Quaternion.

#### Classes

- class [FAD3DMath::Vector2D](#)  
*A vector class used for 2D vectors/points and their manipulations.*
- class [FAD3DMath::Vector3D](#)  
*A vector class used for 3D vectors/points and their manipulations.*
- class [FAD3DMath::Vector4D](#)  
*A vector class used for 4D vectors/points and their manipulations.*
- class [FAD3DMath::Matrix4x4](#)  
*A matrix class used for 4x4 matrices and their manipulations.*
- class [FAD3DMath::Quaternion](#)

#### Namespaces

- namespace [FAD3DMath](#)  
*Has [Vector2D](#), [Vector3D](#), [Vector4D](#), [Matrix4x4](#), and [Quaternion](#) classes.*

#### Functions

- bool **FAD3DMath::compareFloats** (float x, float y, float epsilon)
- bool **FAD3DMath::compareDoubles** (double x, double y, double epsilon)
- bool [FAD3DMath::zeroVector](#) (const Vector2D &a)  
*Returns true if a is the zero vector.*
- Vector2D [FAD3DMath::operator+](#) (const Vector2D &a, const Vector2D &b)  
*2D vector addition.*
- Vector2D [FAD3DMath::operator-](#) (const Vector2D &a)  
*2D vector negation.*
- Vector2D [FAD3DMath::operator-](#) (const Vector2D &a, const Vector2D &b)

- 2D vector subtraction.*

  - Vector2D [FAD3DMath::operator\\*](#) (const Vector2D &a, const float &k)  
*2D vector scalar multiplication. Returns  $a * k$ , where  $a$  is a vector and  $k$  is a scalar(float)*
  - Vector2D [FAD3DMath::operator\\*](#) (const float &k, const Vector2D &a)  
*2D vector scalar multiplication. Returns  $k * a$ , where  $a$  is a vector and  $k$  is a scalar(float)*
  - Vector2D [FAD3DMath::operator/](#) (const Vector2D &a, const float &k)  
*2D vector scalar division. Returns  $a / k$ , where  $a$  is a vector and  $k$  is a scalar(float) If  $k = 0$  the returned vector is the zero vector.*
  - float [FAD3DMath::dotProduct](#) (const Vector2D &a, const Vector2D &b)  
*Returns the dot product between two 2D vectors.*
  - float [FAD3DMath::length](#) (const Vector2D &v)  
*Returns the length(magnitude) of the 2D vector v.*
  - Vector2D [FAD3DMath::norm](#) (const Vector2D &v)  
*Normalizes the 2D vector v.*
  - Vector2D [FAD3DMath::PolarToCartesian](#) (const Vector2D &v)  
*Converts the 2D vector v from polar coordinates to cartesian coordinates. v should = (r, theta(degrees)) The returned 2D vector = (x, y)*
  - Vector2D [FAD3DMath::CartesianToPolar](#) (const Vector2D &v)  
*Converts the 2D vector v from cartesian coordinates to polar coordinates. v should = (x, y, z) If vx is zero then no conversion happens and v is returned.  
The returned 2D vector = (r, theta(degrees)).*
  - Vector2D [FAD3DMath::Projection](#) (const Vector2D &a, const Vector2D &b)  
*Returns a 2D vector that is the projection of a onto b. If b is the zero vector a is returned.*
  - bool [FAD3DMath::zeroVector](#) (const Vector3D &a)  
*Returns true if a is the zero vector.*
  - Vector3D [FAD3DMath::operator+](#) (const Vector3D &a, const Vector3D &b)  
*3D vector addition.*
  - Vector3D [FAD3DMath::operator-](#) (const Vector3D &v)  
*3D vector negation.*
  - Vector3D [FAD3DMath::operator-](#) (const Vector3D &a, const Vector3D &b)  
*3D vector subtraction.*
  - Vector3D [FAD3DMath::operator\\*](#) (const Vector3D &a, const float &k)  
*3D vector scalar multiplication. Returns  $a * k$ , where  $a$  is a vector and  $k$  is a scalar(float)*
  - Vector3D [FAD3DMath::operator\\*](#) (const float &k, const Vector3D &a)  
*3D vector scalar multiplication. Returns  $k * a$ , where  $a$  is a vector and  $k$  is a scalar(float)*
  - Vector3D [FAD3DMath::operator/](#) (const Vector3D &a, const float &k)  
*3D vector scalar division. Returns  $a / k$ , where  $a$  is a vector and  $k$  is a scalar(float) If  $k = 0$  the returned vector is the zero vector.*
  - float [FAD3DMath::dotProduct](#) (const Vector3D &a, const Vector3D &b)  
*Returns the dot product between two 3D vectors.*
  - Vector3D [FAD3DMath::crossProduct](#) (const Vector3D &a, const Vector3D &b)  
*Returns the cross product between two 3D vectors.*
  - float [FAD3DMath::length](#) (const Vector3D &v)  
*Returns the length(magnitude) of the 3D vector v.*
  - Vector3D [FAD3DMath::norm](#) (const Vector3D &v)  
*Normalizes the 3D vector v.*
  - Vector3D [FAD3DMath::CylindricalToCartesian](#) (const Vector3D &v)  
*Converts the 3D vector v from cylindrical coordinates to cartesian coordinates. v should = (r, theta(degrees), z).  
The returned 3D vector = (x, y, z).*
  - Vector3D [FAD3DMath::CartesianToCylindrical](#) (const Vector3D &v)  
*Converts the 3D vector v from cartesian coordinates to cylindrical coordinates. v should = (x, y, z).  
If vx is zero then no conversion happens and v is returned.  
The returned 3D vector = (r, theta(degrees), z).*

- Vector3D [FAD3DMath::SphericalToCartesian](#) (const Vector3D &v)  
*Converts the 3D vector v from spherical coordinates to cartesian coordinates. v should = (pho, phi(degrees), theta(degrees)).*  
*The returned 3D vector = (x, y, z)*
- Vector3D [FAD3DMath::CartesianToSpherical](#) (const Vector3D &v)  
*Converts the 3D vector v from cartesian coordinates to spherical coordinates. If v is the zero vector or if vx is zero then no conversion happens and v is returned.*  
*The returned 3D vector = (r, phi(degrees), theta(degrees)).*
- Vector3D [FAD3DMath::Projection](#) (const Vector3D &a, const Vector3D &b)  
*Returns a 3D vector that is the projection of a onto b. If b is the zero vector a is returned.*
- bool [FAD3DMath::zeroVector](#) (const Vector4D &a)  
*Returns true if a is the zero vector.*
- Vector4D [FAD3DMath::operator+](#) (const Vector4D &a, const Vector4D &b)  
*4D vector addition.*
- Vector4D [FAD3DMath::operator-](#) (const Vector4D &v)  
*4D vector negation.*
- Vector4D [FAD3DMath::operator-](#) (const Vector4D &a, const Vector4D &b)  
*4D vector subtraction.*
- Vector4D [FAD3DMath::operator\\*](#) (const Vector4D &a, const float &k)  
*4D vector scalar multiplication. Returns  $a * k$ , where a is a vector and k is a scalar(float)*
- Vector4D [FAD3DMath::operator\\*](#) (const float &k, const Vector4D &a)  
*4D vector scalar multiplication. Returns  $k * a$ , where a is a vector and k is a scalar(float)*
- Vector4D [FAD3DMath::operator/](#) (const Vector4D &a, const float &k)  
*4D vector scalar division. Returns  $a / k$ , where a is a vector and k is a scalar(float) If  $k = 0$  the returned vector is the zero vector.*
- float [FAD3DMath::dotProduct](#) (const Vector4D &a, const Vector4D &b)  
*Returns the dot product between two 4D vectors.*
- float [FAD3DMath::length](#) (const Vector4D &v)  
*Returns the length(magnitude) of the 4D vector v.*
- Vector4D [FAD3DMath::norm](#) (const Vector4D &v)  
*Normalizes the 4D vector v.*
- Vector4D [FAD3DMath::Projection](#) (const Vector4D &a, const Vector4D &b)  
*Returns a 4D vector that is the projection of a onto b. If b is the zero vector a is returned.*
- Matrix4x4 [FAD3DMath::operator+](#) (const Matrix4x4 &m1, const Matrix4x4 &m2)  
*Adds the two given 4x4 matrices and returns a [Matrix4x4](#) object with the result.*
- Matrix4x4 [FAD3DMath::operator-](#) (const Matrix4x4 &m)  
*Negates the 4x4 matrix m.*
- Matrix4x4 [FAD3DMath::operator-](#) (const Matrix4x4 &m1, const Matrix4x4 &m2)  
*Subtracts the two given 4x4 matrices and returns a [Matrix4x4](#) object with the result.*
- Matrix4x4 [FAD3DMath::operator\\*](#) (const Matrix4x4 &m, const float &k)  
*Multiplies the given 4x4 matrix with the given scalar and returns a [Matrix4x4](#) object with the result.*
- Matrix4x4 [FAD3DMath::operator\\*](#) (const float &k, const Matrix4x4 &m)  
*Multiplies the the given scalar with the given 4x4 matrix and returns a [Matrix4x4](#) object with the result.*
- Matrix4x4 [FAD3DMath::operator\\*](#) (const Matrix4x4 &m1, const Matrix4x4 &m2)  
*Multiplies the two given 4x4 matrices and returns a [Matrix4x4](#) object with the result.*
- Vector4D [FAD3DMath::operator\\*](#) (const Matrix4x4 &m, const Vector4D &v)  
*Multiplies the given 4x4 matrix with the given 4D vector and returns a [Vector4D](#) object with the result. The vector is a column vector.*
- Vector4D [FAD3DMath::operator\\*](#) (const Vector4D &a, const Matrix4x4 &m)  
*Multiplies the given 4D vector with the given 4x4 matrix and returns a [Vector4D](#) object with the result. The vector is a row vector.*
- Matrix4x4 [FAD3DMath::transpose](#) (const Matrix4x4 &m)

- Returns the tranpose of the given matrix m.*

  - Matrix4x4 [FAD3DMath::translate](#) (const Matrix4x4 &cm, float x, float y, float z)
 

*Construct a 4x4 translation matrix with the given floats and post-multiply it by the given matrix.  $cm = cm * translate$ .*
  - Matrix4x4 [FAD3DMath::scale](#) (const Matrix4x4 &cm, float x, float y, float z)
 

*Construct a 4x4 scaling matrix with the given floats and post-multiply it by the given matrix.  $cm = cm * scale$ .*
  - Matrix4x4 [FAD3DMath::rotate](#) (const Matrix4x4 &cm, float angle, float x, float y, float z)
 

*Construct a 4x4 rotation matrix with the given angle (in degrees) and axis (x, y, z) and post-multiply it by the given matrix.  $cm = cm * rotate$ .*
- double [FAD3DMath::det](#) (const Matrix4x4 &m)
 

*Returns the determinant of the given matrix.*
- double [FAD3DMath::cofactor](#) (const Matrix4x4 &m, unsigned int row, unsigned int col)
 

*Returns the cofactor of the given row and col using the given matrix.*
- Matrix4x4 [FAD3DMath::adjoint](#) (const Matrix4x4 &m)
 

*Returns the adjoint of the given matrix.*
- Matrix4x4 [FAD3DMath::inverse](#) (const Matrix4x4 &m)
 

*Returns the inverse of the given matrix. If the matrix is noninvertible/singular, the identity matrix is returned.*
- Quaternion [FAD3DMath::operator+](#) (const Quaternion &q1, const Quaternion &q2)
 

*Returns a quaternion that has the result of  $q1 + q2$ .*
- Quaternion [FAD3DMath::operator-](#) (const Quaternion &q)
 

*Returns a quaternion that has the result of  $-q$ .*
- Quaternion [FAD3DMath::operator-](#) (const Quaternion &q1, const Quaternion &q2)
 

*Returns a quaternion that has the result of  $q1 - q2$ .*
- Quaternion [FAD3DMath::operator\\*](#) (float k, const Quaternion &q)
 

*Returns a quaternion that has the result of  $k * q$ .*
- Quaternion [FAD3DMath::operator\\*](#) (const Quaternion &q, float k)
 

*Returns a quaternion that has the result of  $q * k$ .*
- Quaternion [FAD3DMath::operator\\*](#) (const Quaternion &q1, const Quaternion &q2)
 

*Returns a quaternion that has the result of  $q1 * q2$ .*
- bool [FAD3DMath::isZeroQuaternion](#) (const Quaternion &q)
 

*Returns true if quaternion q is a zero quaternion, false otherwise.*
- bool [FAD3DMath::isIdentity](#) (const Quaternion &q)
 

*Returns true if quaternion q is an identity quaternion, false otherwise.*
- Quaternion [FAD3DMath::conjugate](#) (const Quaternion &q)
 

*Returns the conjugate of quaternion q.*
- float [FAD3DMath::length](#) (const Quaternion &q)
 

*Returns the length of quaternion q.*
- Quaternion [FAD3DMath::normalize](#) (const Quaternion &q)
 

*Normalizes quaternion q and returns the normalized quaternion. If q is the zero quaternion then q is returned.*
- Quaternion [FAD3DMath::inverse](#) (const Quaternion &q)
 

*Returns the invese of quaternion q. If q is the zero quaternion then q is returned.*
- Quaternion [FAD3DMath::rotationQuaternion](#) (float angle, float x, float y, float z)
 

*Returns a quaternion from the axis-angle rotation representation. The angle should be given in degrees.*
- Quaternion [FAD3DMath::rotationQuaternion](#) (float angle, const Vector3D &axis)
 

*Returns a quaternion from the axis-angle rotation representation. The angle should be given in degrees.*
- Quaternion [FAD3DMath::rotationQuaternion](#) (const Vector4D &angAxis)
 

*Returns a quaternion from the axis-angle rotation representation. The x value in the 4D vector should be the angle(in degrees).  
The y, z and w value in the 4D vector should be the axis.*
- Matrix4x4 [FAD3DMath::quaternionRotationMatrixCol](#) (const Quaternion &q)
 

*Returns a matrix from the given quaterion for column vector-matrix multiplication. [Quaternion](#) q should be a unit quaternion.*
- Matrix4x4 [FAD3DMath::quaternionRotationMatrixRow](#) (const Quaternion &q)
 

*Returns a matrix from the given quaterion for row vector-matrix multiplication. [Quaternion](#) q should be a unit quaternion.*



### 6.1.1 Detailed Description

File that has namespace FAMD3Dath. Withn the namespace are the classes Vector2D, Vector3D, Vector4D, Matrix4x4 and Quaternion.

## 6.2 FADirect3DMath.h

[Go to the documentation of this file.](#)

```

1  #pragma once
2
16 namespace FAD3DMath
17 {
18     bool compareFloats(float x, float y, float epsilon);
19     bool compareDoubles(double x, double y, double epsilon);
20
26     class Vector2D
27     {
28     public:
29
30
35         Vector2D();
36
41         Vector2D(float x, float y);
42
45         float x() const;
46
49         float y() const;
50
53         void setX(float x);
54
57         void setY(float y);
58
59
62         Vector2D& operator+=(const Vector2D& b);
63
66         Vector2D& operator-=(const Vector2D& b);
67
70         Vector2D& operator*=(const float& k);
71
76         Vector2D& operator/=(const float& k);
77
78     private:
79         float m_x;
80         float m_y;
81     };
82
85     bool zeroVector(const Vector2D& a);
86
89     Vector2D operator+(const Vector2D& a, const Vector2D& b);
90
93     Vector2D operator-(const Vector2D& v);
94
97     Vector2D operator-(const Vector2D& a, const Vector2D& b);
98
102     Vector2D operator*(const Vector2D& a, const float& k);
103
107     Vector2D operator*(const float& k, const Vector2D& a);
108
113     Vector2D operator/(const Vector2D& a, const float& k);
114
118     float dotProduct(const Vector2D& a, const Vector2D& b);
119
122     float length(const Vector2D& v);
123
128     Vector2D norm(const Vector2D& v);
129
134     Vector2D PolarToCartesian(const Vector2D& v);
135
141     Vector2D CartesianToPolar(const Vector2D& v);
142
146     Vector2D Projection(const Vector2D& a, const Vector2D& b);
147
148     #if defined(_DEBUG)
149         void print(const Vector2D& v);
150     #endif
151
152
153
154

```

```

155
156
157
158
164     class Vector3D
165     {
166     public:
167
168
173         Vector3D();
174
179         Vector3D(float x, float y, float z);
180
183         float x() const;
184
187         float y() const;
188
191         float z() const;
192
195         void setX(float x);
196
199         void setY(float y);
200
203         void setZ(float z);
204
205
208         Vector3D& operator+=(const Vector3D& b);
209
212         Vector3D& operator-=(const Vector3D& b);
213
216         Vector3D& operator*=(const float& k);
217
222         Vector3D& operator/=(const float& k);
223
224     private:
225         float m_x;
226         float m_y;
227         float m_z;
228     };
229
232     bool zeroVector(const Vector3D& a);
233
236     Vector3D operator+(const Vector3D& a, const Vector3D& b);
237
240     Vector3D operator-(const Vector3D& v);
241
244     Vector3D operator-(const Vector3D& a, const Vector3D& b);
245
249     Vector3D operator*(const Vector3D& a, const float& k);
250
254     Vector3D operator*(const float& k, const Vector3D& a);
255
260     Vector3D operator/(const Vector3D& a, const float& k);
261
264     float dotProduct(const Vector3D& a, const Vector3D& b);
265
268     Vector3D crossProduct(const Vector3D& a, const Vector3D& b);
269
272     float length(const Vector3D& v);
273
278     Vector3D norm(const Vector3D& v);
279
284     Vector3D CylindricalToCartesian(const Vector3D& v);
285
291     Vector3D CartesianToCylindrical(const Vector3D& v);
292
297     Vector3D SphericalToCartesian(const Vector3D& v);
298
303     Vector3D CartesianToSpherical(const Vector3D& v);
304
308     Vector3D Projection(const Vector3D& a, const Vector3D& b);
309
310
311     #if defined(_DEBUG)
312         void print(const Vector3D& v);
313     #endif
314
315
316
317
318
319
320
321
322
323
329     class Vector4D

```

```

330     {
331     public:
332
333         Vector4D();
334         Vector4D(float x, float y, float z, float w);
335
336         float x() const;
337         float y() const;
338         float z() const;
339         float w() const;
340
341         void setX(float x);
342         void setY(float y);
343         void setZ(float z);
344         void setW(float w);
345
346         Vector4D& operator+=(const Vector4D& b);
347         Vector4D& operator-=(const Vector4D& b);
348         Vector4D& operator*=(const float& k);
349         Vector4D& operator/=(const float& k);
350
351     private:
352         float m_x;
353         float m_y;
354         float m_z;
355         float m_w;
356     };
357
358     bool zeroVector(const Vector4D& a);
359
360     Vector4D operator+(const Vector4D& a, const Vector4D& b);
361     Vector4D operator-(const Vector4D& v);
362     Vector4D operator-(const Vector4D& a, const Vector4D& b);
363     Vector4D operator*(const Vector4D& a, const float& k);
364     Vector4D operator*(const float& k, const Vector4D& a);
365     Vector4D operator/(const Vector4D& a, const float& k);
366
367     float dotProduct(const Vector4D& a, const Vector4D& b);
368     float length(const Vector4D& v);
369     Vector4D norm(const Vector4D& v);
370     Vector4D Projection(const Vector4D& a, const Vector4D& b);
371
372     #if defined(_DEBUG)
373     void print(const Vector4D& v);
374     #endif
375
376     class Matrix4x4
377     {
378     public:
379         Matrix4x4();
380         Matrix4x4(float a[][4]);
381
382         float getElement(unsigned int row, unsigned int col) const;
383         const float& operator()(unsigned int row, unsigned int col) const;
384         float& operator()(unsigned int row, unsigned int col);
385         void setElement(unsigned int row, unsigned int col, float val);
386         void setToIdentity();

```

```

510
513     bool isIdentity();
514
517     Matrix4x4& operator+=(const Matrix4x4& m);
518
521     Matrix4x4& operator-=(const Matrix4x4& m);
522
525     Matrix4x4& operator*=(const float& k);
526
527
530     Matrix4x4& operator*=(const Matrix4x4& m);
531
532
533 private:
534
535     float m_mat[4][4];
536 };
537
541 Matrix4x4 operator+(const Matrix4x4& m1, const Matrix4x4& m2);
542
545 Matrix4x4 operator-(const Matrix4x4& m);
546
549 Matrix4x4 operator-(const Matrix4x4& m1, const Matrix4x4& m2);
550
553 Matrix4x4 operator*(const Matrix4x4& m, const float& k);
554
557 Matrix4x4 operator*(const float& k, const Matrix4x4& m);
558
561 Matrix4x4 operator*(const Matrix4x4& m1, const Matrix4x4& m2);
562
566 Vector4D operator*(const Matrix4x4& m, const Vector4D& v);
567
571 Vector4D operator*(const Vector4D& a, const Matrix4x4& m);
572
575 Matrix4x4 transpose(const Matrix4x4& m);
576
580 Matrix4x4 translate(const Matrix4x4& cm, float x, float y, float z);
581
585 Matrix4x4 scale(const Matrix4x4& cm, float x, float y, float z);
586
590 Matrix4x4 rotate(const Matrix4x4& cm, float angle, float x, float y, float z);
591
594 double det(const Matrix4x4& m);
595
598 double cofactor(const Matrix4x4& m, unsigned int row, unsigned int col);
599
602 Matrix4x4 adjoint(const Matrix4x4& m);
603
607 Matrix4x4 inverse(const Matrix4x4& m);
608
609
610
611 #if defined(_DEBUG)
612     void print(const Matrix4x4& m);
613 #endif
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
634 class Quaternion
635 {
636 public:
637
642     Quaternion();
643
648     Quaternion(float scalar, float x, float y, float z);
649
654     Quaternion(float scalar, const Vector3D& v);
655
662     Quaternion(const Vector4D& v);
663
666     float& scalar();
667
670     const float& scalar() const;
671

```

```

674     float& x();
675
678     const float& x() const;
679
682     float& y();
683
686     const float& y() const;
687
690     float& z();
691
694     const float& z() const;
695
698     Vector3D vector();
699
702     Quaternion& operator+=(const Quaternion& q);
703
706     Quaternion& operator-=(const Quaternion& q);
707
710     Quaternion& operator*=(float k);
711
714     Quaternion& operator*=(const Quaternion& q);
715
716 private:
717     float m_scalar;
720     float m_x;
721     float m_y;
722     float m_z;
723 };
724
727 Quaternion operator+(const Quaternion& q1, const Quaternion& q2);
728
731 Quaternion operator-(const Quaternion& q);
732
735 Quaternion operator-(const Quaternion& q1, const Quaternion& q2);
736
739 Quaternion operator*(float k, const Quaternion& q);
740
743 Quaternion operator*(const Quaternion& q, float k);
744
747 Quaternion operator*(const Quaternion& q1, const Quaternion& q2);
748
749
752 bool isZeroQuaternion(const Quaternion& q);
753
756 bool isIdentity(const Quaternion& q);
757
760 Quaternion conjugate(const Quaternion& q);
761
764 float length(const Quaternion& q);
765
769 Quaternion normalize(const Quaternion& q);
770
774 Quaternion inverse(const Quaternion& q);
775
779 Quaternion rotationQuaternion(float angle, float x, float y, float z);
780
784 Quaternion rotationQuaternion(float angle, const Vector3D& axis);
785
790 Quaternion rotationQuaternion(const Vector4D& angAxis);
791
795 Matrix4x4 quaternionRotationMatrixCol(const Quaternion& q);
796
800 Matrix4x4 quaternionRotationMatrixRow(const Quaternion& q);
801
802 #if defined(_DEBUG)
803     void print(const Quaternion& q);
804 #endif
805
806 }

```



# Index

- adjoint
  - FAD3DMath, 11
- CartesianToCylindrical
  - FAD3DMath, 11
- CartesianToPolar
  - FAD3DMath, 11
- CartesianToSpherical
  - FAD3DMath, 11
- cofactor
  - FAD3DMath, 11
- conjugate
  - FAD3DMath, 12
- crossProduct
  - FAD3DMath, 12
- CylindricalToCartesian
  - FAD3DMath, 12
- det
  - FAD3DMath, 12
- dotProduct
  - FAD3DMath, 12, 13
- FAD3DMath, 7
  - adjoint, 11
  - CartesianToCylindrical, 11
  - CartesianToPolar, 11
  - CartesianToSpherical, 11
  - cofactor, 11
  - conjugate, 12
  - crossProduct, 12
  - CylindricalToCartesian, 12
  - det, 12
  - dotProduct, 12, 13
  - inverse, 13
  - isIdentity, 13
  - isZeroQuaternion, 14
  - length, 14
  - norm, 14, 15
  - normalize, 15
  - operator\*, 15–18
  - operator+, 18, 19
  - operator-, 19–21
  - operator/, 21
  - PolarToCartesian, 22
  - Projection, 22
  - quaternionRotationMatrixCol, 22
  - quaternionRotationMatrixRow, 23
  - rotate, 23
  - rotationQuaternion, 23
  - scale, 24
  - SphericalToCartesian, 24
  - translate, 24
  - transpose, 24
  - zeroVector, 24, 25
- FAD3DMath::Matrix4x4, 27
  - getElement, 28
  - isIdentity, 28
  - Matrix4x4, 28
  - operator\*=, 29
  - operator(), 29
  - operator+=, 29
  - operator-=, 30
  - setElement, 30
  - setToIdentity, 30
- FAD3DMath::Quaternion, 30
  - operator\*=, 32
  - operator+=, 33
  - operator-=, 33
  - Quaternion, 31, 32
  - scalar, 33
  - vector, 33
  - x, 33, 34
  - y, 34
  - z, 34
- FAD3DMath::Vector2D, 35
  - operator\*=, 36
  - operator+=, 36
  - operator-=, 36
  - operator/=: 36
  - setX, 36
  - setY, 37
  - Vector2D, 35
  - x, 37
  - y, 37
- FAD3DMath::Vector3D, 37
  - operator\*=, 39
  - operator+=, 39
  - operator-=, 39
  - operator/=: 39
  - setX, 39
  - setY, 40
  - setZ, 40
  - Vector3D, 38
  - x, 40
  - y, 40
  - z, 40
- FAD3DMath::Vector4D, 41
  - operator\*=, 42

- operator+=, [42](#)
- operator-=, [42](#)
- operator/=: [42](#)
- setW, [43](#)
- setX, [43](#)
- setY, [43](#)
- setZ, [43](#)
- Vector4D, [41](#), [42](#)
- w, [43](#)
- x, [43](#)
- y, [44](#)
- z, [44](#)
- FADirect3DMath.h, [45](#)
- getElement
  - FAD3DMath::Matrix4x4, [28](#)
- inverse
  - FAD3DMath, [13](#)
- isIdentity
  - FAD3DMath, [13](#)
  - FAD3DMath::Matrix4x4, [28](#)
- isZeroQuaternion
  - FAD3DMath, [14](#)
- length
  - FAD3DMath, [14](#)
- Matrix4x4
  - FAD3DMath::Matrix4x4, [28](#)
- norm
  - FAD3DMath, [14](#), [15](#)
- normalize
  - FAD3DMath, [15](#)
- operator\*
  - FAD3DMath, [15–18](#)
- operator\*=
  - FAD3DMath::Matrix4x4, [29](#)
  - FAD3DMath::Quaternion, [32](#)
  - FAD3DMath::Vector2D, [36](#)
  - FAD3DMath::Vector3D, [39](#)
  - FAD3DMath::Vector4D, [42](#)
- operator()
  - FAD3DMath::Matrix4x4, [29](#)
- operator+
  - FAD3DMath, [18](#), [19](#)
- operator+=
  - FAD3DMath::Matrix4x4, [29](#)
  - FAD3DMath::Quaternion, [33](#)
  - FAD3DMath::Vector2D, [36](#)
  - FAD3DMath::Vector3D, [39](#)
  - FAD3DMath::Vector4D, [42](#)
- operator-
  - FAD3DMath, [19–21](#)
- operator-=
  - FAD3DMath::Matrix4x4, [30](#)
  - FAD3DMath::Quaternion, [33](#)
  - FAD3DMath::Vector2D, [36](#)
- FAD3DMath::Vector3D, [39](#)
- FAD3DMath::Vector4D, [42](#)
- operator/
  - FAD3DMath, [21](#)
- operator/=
  - FAD3DMath::Vector2D, [36](#)
  - FAD3DMath::Vector3D, [39](#)
  - FAD3DMath::Vector4D, [42](#)
- PolarToCartesian
  - FAD3DMath, [22](#)
- Projection
  - FAD3DMath, [22](#)
- Quaternion
  - FAD3DMath::Quaternion, [31](#), [32](#)
- quaternionRotationMatrixCol
  - FAD3DMath, [22](#)
- quaternionRotationMatrixRow
  - FAD3DMath, [23](#)
- rotate
  - FAD3DMath, [23](#)
- rotationQuaternion
  - FAD3DMath, [23](#)
- scalar
  - FAD3DMath::Quaternion, [33](#)
- scale
  - FAD3DMath, [24](#)
- setElement
  - FAD3DMath::Matrix4x4, [30](#)
- setToIdentity
  - FAD3DMath::Matrix4x4, [30](#)
- setW
  - FAD3DMath::Vector4D, [43](#)
- setX
  - FAD3DMath::Vector2D, [36](#)
  - FAD3DMath::Vector3D, [39](#)
  - FAD3DMath::Vector4D, [43](#)
- setY
  - FAD3DMath::Vector2D, [37](#)
  - FAD3DMath::Vector3D, [40](#)
  - FAD3DMath::Vector4D, [43](#)
- setZ
  - FAD3DMath::Vector3D, [40](#)
  - FAD3DMath::Vector4D, [43](#)
- SphericalToCartesian
  - FAD3DMath, [24](#)
- translate
  - FAD3DMath, [24](#)
- transpose
  - FAD3DMath, [24](#)
- vector
  - FAD3DMath::Quaternion, [33](#)
- Vector2D
  - FAD3DMath::Vector2D, [35](#)
- Vector3D



- FAD3DMath::Vector3D, [38](#)
- Vector4D
  - FAD3DMath::Vector4D, [41](#), [42](#)
- w
  - FAD3DMath::Vector4D, [43](#)
- x
  - FAD3DMath::Quaternion, [33](#), [34](#)
  - FAD3DMath::Vector2D, [37](#)
  - FAD3DMath::Vector3D, [40](#)
  - FAD3DMath::Vector4D, [43](#)
- y
  - FAD3DMath::Quaternion, [34](#)
  - FAD3DMath::Vector2D, [37](#)
  - FAD3DMath::Vector3D, [40](#)
  - FAD3DMath::Vector4D, [44](#)
- z
  - FAD3DMath::Quaternion, [34](#)
  - FAD3DMath::Vector3D, [40](#)
  - FAD3DMath::Vector4D, [44](#)
- zeroVector
  - FAD3DMath, [24](#), [25](#)