Farouq Adepetu's Math Engine

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 utility functions, 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)
- 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.
- void setToIdentity (Matrix4x4 &m)

  Sets the given matrix to the identity matrix.
- bool isIdentity (const Matrix4x4 &m)

  Returns true if the given matrix is the identity matrix, false otherwise.
- 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's it by the given matrix. cm = cm ∗ translate.
- Matrix4x4 scale (const Matrix4x4 &cm, float x, float y, float z)

  Construct a 4x4 scaling matrix with the given floats and post-multiply's it by the given matrix. cm = cm ∗ 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's 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.
- 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 negeation.
- 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 = (pho, 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 vx is zero then no conversion happens and v is returned.
    The returned 3D vector = (r, phi(degrees), theta(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.

4.1.1 Detailed Description

Has utility functions, Vector2D, Vector3D, Vector4D, Matrix4x4, and Quaternion classes.

The name space has utility functions, Vector2D, Vector3D, Vector4D, Matrix4x4, and Quaternion classes.

4.1.2 Function Documentation

4.1.2.1 adjoint()

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

Returns the adjoint of the given matrix.

4.1.2.2 CartesianToCylindrical()

```
Vector3D FAMath::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 FAMath::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 FAMath::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 FAMath::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 FAMath::conjugate (
             const Quaternion & q )
```

Returns the conjugate of quaternion q.

4.1.2.7 crossProduct()

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

Returns the cross product between two 3D vectors.

4.1.2.8 CylindricalToCartesian()

```
Vector3D FAMath::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 FAMath::det (
             const Matrix4x4 & m )
```

Returns the determinant of the given matrix.

4.1.2.10 dotProduct() [1/3]

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

Returns the dot product between two 2D vectors.

4.1.2.11 dotProduct() [2/3]

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

Returns the dot product between two 3D vectors.

4.1.2.12 dotProduct() [3/3]

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

Returns the dot product between two 4D vectors.

4.1.2.13 inverse() [1/2]

```
Matrix4x4 FAMath::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 FAMath::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() [1/2]

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

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

4.1.2.16 isIdentity() [2/2]

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

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

4.1.2.17 isZeroQuaternion()

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

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

4.1.2.18 length() [1/4]

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

Returns the length of quaternion q.

4.1.2.19 length() [2/4]

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

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

4.1.2.20 length() [3/4]

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

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

4.1.2.21 length() [4/4]

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

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

4.1.2.22 norm() [1/3]

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

Normalizes the 2D vector v.

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

4.1.2.23 norm() [2/3]

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

Normalizes the 3D vector v.

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

4.1.2.24 norm() [3/3]

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

Normalizes the 4D vector v.

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

4.1.2.25 normalize()

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

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

4.1.2.26 operator*() [1/14]

```
Matrix4x4 FAMath::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.

4.1.2.27 operator*() [2/14]

```
Vector2D FAMath::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)

4.1.2.28 operator*() [3/14]

```
Vector3D FAMath::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)

4.1.2.29 operator*() [4/14]

```
Vector4D FAMath::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)

4.1.2.30 operator*() [5/14]

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

Multiplies the given 4x4 matrix with the given scalar and returns a Matrix4x4 object with the result.

4.1.2.31 operator∗() [6/14]

```
Vector4D FAMath::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.

4.1.2.32 operator∗() [7/14]

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

Multiplies the two given 4x4 matrices and returns a Matrix4x4 object with the result.

4.1.2.33 operator∗() [8/14]

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

Returns a quaternion that has the result of q ∗ k.

4.1.2.34 operator∗() [9/14]

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

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

4.1.2.35 operator∗() [10/14]

```
Vector2D FAMath::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)

4.1.2.36 operator*() [11/14]

```
Vector3D FAMath::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)

4.1.2.37 operator*() [12/14]

```
Vector4D FAMath::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)

4.1.2.38 operator*() [13/14]

```
Vector4D FAMath::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.

4.1.2.39 operator*() [14/14]

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

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

4.1.2.40 operator+() [1/5]

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

Adds the two given 4x4 matrices and returns a Matrix4x4 object with the result.

4.1.2.41 operator+() [2/5]

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

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

4.1.2.42 operator+() [3/5]

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

2D vector addition.

4.1.2.43 operator+() [4/5]

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

3D vector addition.

4.1.2.44 operator+() [5/5]

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

4D vector addition.

4.1.2.45 operator-() [1/10]

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

Negates the 4x4 matrix m.

4.1.2.46 operator-() [2/10]

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

Subtracts the two given 4x4 matrices and returns a Matrix4x4 object with the result.

4.1.2.47 operator-() [3/10]

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

Returns a quaternion that has the result of -q.

4.1.2.48 operator-() [4/10]

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

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

4.1.2.49 operator-() [5/10]

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

2D vector subtraction.

4.1.2.50 operator-() [6/10]

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

2D vector negation.

4.1.2.51 operator-() [7/10]

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

3D vector subtraction.

4.1.2.52 operator-() [8/10]

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

3D vector negeation.

4.1.2.53 operator-() [9/10]

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

4D vector subtraction.

4.1.2.54 operator-() [10/10]

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

4D vector negation.

4.1.2.55 operator/() [1/3]

```
Vector2D FAMath::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.

4.1.2.56 operator/() [2/3]

```
Vector3D FAMath::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.

4.1.2.57 operator/() [3/3]

```
Vector4D FAMath::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.

4.1.2.58 PolarToCartesian()

```
Vector2D FAMath::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)

4.1.2.59 Projection() [1/3]

```
Vector2D FAMath::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.

4.1.2.60 Projection() [2/3]

```
Vector3D FAMath::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.

4.1.2.61 Projection() [3/3]

```
Vector4D FAMath::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.

4.1.2.62 quaternionRotationMatrixCol()

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

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

4.1.2.63 quaternionRotationMatrixRow()

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

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

4.1.2.64 rotate()

```
Matrix4x4 FAMath::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's it by the given matrix. cm = cm ∗ rotate.

.

4.1.2.65 rotationQuaternion() [1/3]

```
Quaternion FAMath::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.

4.1.2.66 rotationQuaternion() [2/3]

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

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

4.1.2.67 rotationQuaternion() [3/3]

```
Quaternion FAMath::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.

4.1.2.68 scale()

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

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

4.1.2.69 setToIdentity()

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

Sets the given matrix to the identity matrix.

4.1.2.70 SphericalToCartesian()

```
Vector3D FAMath::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)

4.1.2.71 translate()

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

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

4.1.2.72 transpose()

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

Returns the tranpose of the given matrix m.

4.1.2.73 zeroVector() [1/3]

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

Returns true if a is the zero vector.

4.1.2.74 zeroVector() [2/3]

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

Returns true if a is the zero vector.

4.1.2.75 zeroVector() [3/3]

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

Returns true if a is the zero vector.

Chapter 5

Class Documentation

5.1 FAMath::Matrix4x4 Class Reference

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

```
#include "FAMatrix4x4.h"
```

Public Member Functions

- Matrix4x4 ()

  Default Constructor.
- Matrix4x4 (float a[ ][4])

  Overloaded Constructor.
- float ∗ data ()

  Returns a pointer to the first element in the matrix.
- const float ∗ data () const

  Returns a constant pointer to the first element in the matrix.
- 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 setRow (unsigned int row, Vector4D v)

  Sets each element in the given row to the components of vector v. Row should be between [0,3]. If it out of range the first row will be set.
- void setCol (unsigned int col, Vector4D v)

  Sets each element in the given col to the components of vector v. Col should be between [0,3]. If it out of range the first col will be set.
- 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]`

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

Default Constructor.

Creates a new 4x4 identity matrix.

5.1.2.2 Matrix4x4() `[2/2]`

```
FAMath::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 data() `[1/2]`

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

Returns a pointer to the first element in the matrix.

5.1.3.2 data() `[2/2]`

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

Returns a constant pointer to the first element in the matrix.

5.1.3.3 operator()() [1/2]

```
float & FAMath::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 & FAMath::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 & FAMath::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 & FAMath::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 & FAMath::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 & FAMath::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 setCol()

```
void FAMath::Matrix4x4::setCol (
            unsigned int col,
            Vector4D v )
```

Sets each element in the given col to the components of vector v. Col should be between [0,3]. If it out of range the first col will be set.

5.1.3.10 setRow()

```
void FAMath::Matrix4x4::setRow (
            unsigned int row,
            Vector4D v )
```

Sets each element in the given row to the components of vector v. Row should be between [0,3]. If it out of range the first row will be set.

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

- FAMatrix4x4.h

5.2 FAMath::Quaternion Class Reference

```
#include "FAQuaternion.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]

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

Default Constructor.

Constructs an identity quaternion.

5.2.2.2 Quaternion() [2/4]

```
FAMath::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]

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

Overloaded Constructor.

Constructs a quaternion with the given values.

5.2.2.4 Quaternion() [4/4]

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

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

5.2.3.2 operator*=() [2/2]

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

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

5.2.3.3 operator+=()

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

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

5.2.3.4 operator-=()

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

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

5.2.3.5 scalar() [1/2]

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

Returns a reference to the scalar component of the quaternion.

5.2.3.6 scalar() [2/2]

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

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

5.2.3.7 vector()

```
Vector3D FAMath::Quaternion::vector ( )
```

Returns the vector component of the quaternion.

5.2.3.8 x() [1/2]

```
float & FAMath::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 & FAMath::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 & FAMath::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 & FAMath::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 & FAMath::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 & FAMath::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 file:

- FAQuaternion.h

5.3 FAMath::Vector2D Class Reference

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

```
#include "FAVector2D.h"
```

Public Member Functions

- Vector2D ()

  Default Constructor.
- Vector2D (float x, float y)

  Overloaded Constructor.
- Vector2D (Vector3D v)

  Overloaded Constructor.
- Vector2D (Vector4D v)

  Overloaded Constructor.
- float & x ()

  Returns a reference to the x component.
- float & y ()

  Returns a reference to the y component.
- const float & x () const

  Returns a constant reference to the x component.
- const float & y () const

  Returns a constant reference to the y component.
- 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() [1/4]

```
FAMath::Vector2D::Vector2D ( )
```

Default Constructor.

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

5.3.2.2 Vector2D() [2/4]

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

Overloaded Constructor.

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

5.3.2.3 Vector2D() [3/4]

```
FAMath::Vector2D::Vector2D (
             Vector3D v )
```

Overloaded Constructor.

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

5.3.2.4 Vector2D() [4/4]

```
FAMath::Vector2D::Vector2D (
             Vector4D v )
```

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

2D vector scalar multiplication through overloading operator *=.

5.3.3.2 operator+=()

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

2D vector addition through overloading operator +=.

5.3.3.3 operator-=()

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

2D vector subtraction through overloading operator -=.

5.3.3.4 operator/=()

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

2D vector scalar division through overloading operator /=.

If k is zero, the vector is unchanged.

5.3.3.5 x() [1/2]

```
float & FAMath::Vector2D::x ( )
```

Returns a reference to the x component.

5.3.3.6 x() [2/2]

```
const float & FAMath::Vector2D::x ( ) const
```

Returns a constant reference to the x component.

5.3.3.7 y() [1/2]

```
float & FAMath::Vector2D::y ( )
```

Returns a reference to the y component.

5.3.3.8 y() [2/2]

```
const float & FAMath::Vector2D::y ( ) const
```

Returns a constant reference to the y component.

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

- FAVector2D.h

5.4 FAMath::Vector3D Class Reference

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

```
#include "FAVector3D.h"
```

Public Member Functions

- Vector3D ()

  Default Constructor.
- Vector3D (float x, float y, float z)

  Overloaded Constructor.
- Vector3D (Vector2D v, float z=0.0f)

  Overloaded Constructor.
- Vector3D (Vector4D v)

  Overloaded Constructor.
- float & x ()

  Returns a reference to the x component.
- float & y ()

  Returns a reference to the y component.
- float & z ()

  Returns a reference to the z component.
- const float & x () const

  Returns a constant reference to the x component.
- const float & y () const

  Returns a constant reference to the y component.
- const float & z () const

  Returns a constant reference to the z component.
- 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() [1/4]

```
FAMath::Vector3D::Vector3D ( )
```

Default Constructor.

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

5.4.2.2 Vector3D() [2/4]

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

Overloaded Constructor.

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

5.4.2.3 Vector3D() [3/4]

```
FAMath::Vector3D::Vector3D (
            Vector2D v,
            float z = 0.0f )
```

Overloaded Constructor.

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

5.4.2.4 Vector3D() [4/4]

```
FAMath::Vector3D::Vector3D (
            Vector4D v )
```

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

3D vector scalar multiplication through overloading operator ∗=.

5.4.3.2 operator+=()

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

3D vector addition through overloading operator +=.

5.4.3.3 operator-=()

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

3D vector subtraction through overloading operator -=.

5.4.3.4 operator/=()

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

3D vector scalar division through overloading operator /=.

If k is zero, the vector is unchanged.

5.4.3.5 x() [1/2]

```
float & FAMath::Vector3D::x ( )
```

Returns a reference to the x component.

5.4.3.6 x() [2/2]

```
const float & FAMath::Vector3D::x ( ) const
```

Returns a constant reference to the x component.

5.4.3.7 y() [1/2]

```
float & FAMath::Vector3D::y ( )
```

Returns a reference to the y component.

5.4.3.8 y() [2/2]

```
const float & FAMath::Vector3D::y ( ) const
```

Returns a constant reference to the y component.

5.4.3.9 z() [1/2]

```
float & FAMath::Vector3D::z ( )
```

Returns a reference to the z component.

5.4.3.10 z() [2/2]

```
const float & FAMath::Vector3D::z ( ) const
```

Returns a constant reference to the z component.

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

- FAVector3D.h

5.5 FAMath::Vector4D Class Reference

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

```
#include "FAVector4D.h"
```

Public Member Functions

- Vector4D ()

  Default Constructor.
- Vector4D (float x, float y, float z, float w)

  Overloaded Constructor.
- Vector4D (Vector2D v, float z=0.0f, float w=0.0f)

  Overloaded Constructor.
- Vector4D (Vector3D v, float w=0.0f)

  Overloaded Constructor.
- float & x ()

  Returns a reference to the x component.
- float & y ()

  Returns a reference to the y component.
- float & z ()

  Returns a reference to the z component.

- float & w ()

    Returns a reference to the w component.
- const float & x () const

    Returns a constant reference to the x component.
- const float & y () const

    Returns a constant reference to the y component.
- const float & z () const

    Returns a constant reference to the z component.
- const float & w () const

    Returns a constant reference to the w component.
- 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() [1/4]

```
FAMath::Vector4D::Vector4D ( )
```

Default Constructor.

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

5.5.2.2 Vector4D() [2/4]

```
FAMath::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.

5.5.2.3 Vector4D() [3/4]

```
FAMath::Vector4D::Vector4D (
            Vector2D v,
            float z = 0.0f,
            float w = 0.0f )
```

Overloaded Constructor.

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

5.5.2.4 Vector4D() [4/4]

```
FAMath::Vector4D::Vector4D (
            Vector3D v,
            float w = 0.0f )
```

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

4D vector scalar multiplication through overloading operator *=.

5.5.3.2 operator+=()

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

4D vector addition through overloading operator +=.

5.5.3.3 operator-=()

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

4D vector subtraction through overloading operator -=.

5.5.3.4 operator/=()

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

4D vector scalar division through overloading operator /=.

If k is zero, the vector is unchanged.

5.5.3.5 w() [1/2]

```
float & FAMath::Vector4D::w ( )
```

Returns a reference to the w component.

5.5.3.6 w() [2/2]

```
const float & FAMath::Vector4D::w ( ) const
```

Returns a constant reference to the w component.

5.5.3.7 x() [1/2]

```
float & FAMath::Vector4D::x ( )
```

Returns a reference to the x component.

5.5.3.8 x() [2/2]

```
const float & FAMath::Vector4D::x ( ) const
```

Returns a constant reference to the x component.

5.5.3.9 y() [1/2]

```
float & FAMath::Vector4D::y ( )
```

Returns a reference to the y component.

5.5.3.10 y() [2/2]

```
const float & FAMath::Vector4D::y ( ) const
```

Returns a constant reference to the y component.

5.5.3.11 z() [1/2]

```
float & FAMath::Vector4D::z ( )
```

Returns a reference to the z component.

5.5.3.12 z() [2/2]

```
const float & FAMath::Vector4D::z ( ) const
```

Returns a constant reference to the z component.

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

- FAVector4D.h

Chapter 6

File Documentation

6.1 FAMathUtility.h File Reference

File that has math utility functions.

```
#include <cmath>
```

Namespaces

- namespace FAMath

  Has utility functions, Vector2D, Vector3D, Vector4D, Matrix4x4, and Quaternion classes.

Macros

- #define **EPSILON** 1e-6
- #define **PI** 3.14159265

Functions

- bool **FAMath::compareFloats** (float x, float y, float epsilon)
- bool **FAMath::compareDoubles** (double x, double y, double epsilon)

6.1.1 Detailed Description

File that has math utility functions.

6.2 FAMathUtility.h

Go to the documentation of this file.

```
#pragma once

#include <cmath>

#if defined(_DEBUG)
#include <iostream>
#endif


#define EPSILON 1e-6
#define PI 3.14159265

namespace FAMath
{
    /*@brief Checks if the two specified floats are equal using exact epsilion and adaptive epsilion.
*/
    bool compareFloats(float x, float y, float epsilon);

    /*@brief Checks if the two specified doubles are equal using exact epsilion and adaptive epsilion.
*/
    bool compareDoubles(double x, double y, double epsilon);

    class Vector2D;
    class Vector3D;
    class Vector4D;
}
```

6.3 FAMatrix4x4.h File Reference

File has a 4x4 matrix class under the namespace FAMath.

`#include "FAMathUtility.h"`

Classes

- class FAMath::Matrix4x4

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

Namespaces

- namespace FAMath

  Has utility functions, Vector2D, Vector3D, Vector4D, Matrix4x4, and Quaternion classes.

Functions

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

  Adds the two given 4x4 matrices and returns a Matrix4x4 object with the result.
- Matrix4x4 FAMath::operator- (const Matrix4x4 &m)

  Negates the 4x4 matrix m.
- Matrix4x4 FAMath::operator- (const Matrix4x4 &m1, const Matrix4x4 &m2)

  Subtracts the two given 4x4 matrices and returns a Matrix4x4 object with the result.
- Matrix4x4 FAMath::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 FAMath::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 FAMath::operator∗ (const Matrix4x4 &m1, const Matrix4x4 &m2)

  Multiplies the two given 4x4 matrices and returns a Matrix4x4 object with the result.
- Vector4D FAMath::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 FAMath::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.
- void FAMath::setToIdentity (Matrix4x4 &m)

  Sets the given matrix to the identity matrix.
- bool FAMath::isIdentity (const Matrix4x4 &m)

  Returns true if the given matrix is the identity matrix, false otherwise.
- Matrix4x4 FAMath::transpose (const Matrix4x4 &m)

  Returns the tranpose of the given matrix m.
- Matrix4x4 FAMath::translate (const Matrix4x4 &cm, float x, float y, float z)

  Construct a 4x4 translation matrix with the given floats and post-multiply's it by the given matrix. cm = cm ∗ translate.
- Matrix4x4 FAMath::scale (const Matrix4x4 &cm, float x, float y, float z)

  Construct a 4x4 scaling matrix with the given floats and post-multiply's it by the given matrix. cm = cm ∗ scale.
- Matrix4x4 FAMath::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's it by the given matrix. cm = cm ∗ rotate.

  .
- double FAMath::det (const Matrix4x4 &m)

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

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

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

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

6.3.1 Detailed Description

File has a 4x4 matrix class under the namespace FAMath.

6.4 FAMatrix4x4.h

Go to the documentation of this file.

```
#pragma once

#include "FAMathUtility.h"

namespace FAMath
{
    class Matrix4x4
    {
    public:

        Matrix4x4();

        Matrix4x4(float a[][4]);

        float* data();

        const float* data() const;

        const float& operator()(unsigned int row, unsigned int col) const;

        float& operator()(unsigned int row, unsigned int col);

        void setRow(unsigned int row, Vector4D v);

        void setCol(unsigned int col, Vector4D v);

        Matrix4x4& operator+=(const Matrix4x4& m);

        Matrix4x4& operator-=(const Matrix4x4& m);

        Matrix4x4& operator*=(const float& k);


        Matrix4x4& operator*=(const Matrix4x4& m);

    private:

        float m_mat[4][4];
    };

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

    Matrix4x4 operator-(const Matrix4x4& m);

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

    Matrix4x4 operator*(const Matrix4x4& m, const float& k);

    Matrix4x4 operator*(const float& k, const Matrix4x4& m);

    Matrix4x4 operator*(const Matrix4x4& m1, const Matrix4x4& m2);

    Vector4D operator*(const Matrix4x4& m, const Vector4D& v);

    Vector4D operator*(const Vector4D& a, const Matrix4x4& m);

    void setToIdentity(Matrix4x4& m);

    bool isIdentity(const Matrix4x4& m);

    Matrix4x4 transpose(const Matrix4x4& m);

    Matrix4x4 translate(const Matrix4x4& cm, float x, float y, float z);

    Matrix4x4 scale(const Matrix4x4& cm, float x, float y, float z);

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

    double det(const Matrix4x4& m);

    double cofactor(const Matrix4x4& m, unsigned int row, unsigned int col);

    Matrix4x4 adjoint(const Matrix4x4& m);

    Matrix4x4 inverse(const Matrix4x4& m);



#if defined(_DEBUG)
    void print(const Matrix4x4& m);
#endif
}
```

6.5 FAQuaternion.h

```
#pragma once

#include "FAMathUtility.h"
#include "FAMatrix4x4.h"

namespace FAMath
{
    class Quaternion
    {
    public:

        Quaternion();

        Quaternion(float scalar, float x, float y, float z);

```

```
       Quaternion(float scalar, const Vector3D& v);

       Quaternion(const Vector4D& v);

       float& scalar();

       const float& scalar() const;

       float& x();

       const float& x() const;

       float& y();

       const float& y() const;

       float& z();

       const float& z() const;

       Vector3D vector();

       Quaternion& operator+=(const Quaternion& q);

       Quaternion& operator-=(const Quaternion& q);

       Quaternion& operator*=(float k);

        Quaternion& operator*=(const Quaternion& q);


    private:

        float m_scalar;
        float m_x;
        float m_y;
        float m_z;
    };

    Quaternion operator+(const Quaternion& q1, const Quaternion& q2);

    Quaternion operator-(const Quaternion& q);

    Quaternion operator-(const Quaternion& q1, const Quaternion& q2);

    Quaternion operator*(float k, const Quaternion& q);

    Quaternion operator*(const Quaternion& q, float k);

    Quaternion operator*(const Quaternion& q1, const Quaternion& q2);


    bool isZeroQuaternion(const Quaternion& q);

    bool isIdentity(const Quaternion& q);

    Quaternion conjugate(const Quaternion& q);

    float length(const Quaternion& q);

    Quaternion normalize(const Quaternion& q);

    Quaternion inverse(const Quaternion& q);

    Quaternion rotationQuaternion(float angle, float x, float y, float z);

    Quaternion rotationQuaternion(float angle, const Vector3D& axis);

    Quaternion rotationQuaternion(const Vector4D& angAxis);

    Matrix4x4 quaternionRotationMatrixCol(const Quaternion& q);

    Matrix4x4 quaternionRotationMatrixRow(const Quaternion& q);

#if defined(_DEBUG)
    void print(const Quaternion& q);
#endif

}
```

6.6 FAVector2D.h File Reference

File has a 2D Vector class under the namespace FAMath.

```
#include "FAMathUtility.h"
```

Classes

- class FAMath::Vector2D

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

Namespaces

- namespace FAMath

  Has utility functions, Vector2D, Vector3D, Vector4D, Matrix4x4, and Quaternion classes.

Functions

- bool FAMath::zeroVector (const Vector2D &a)

  Returns true if a is the zero vector.
- Vector2D FAMath::operator+ (const Vector2D &a, const Vector2D &b)

  2D vector addition.
- Vector2D FAMath::operator- (const Vector2D &v)

  2D vector negation.
- Vector2D FAMath::operator- (const Vector2D &a, const Vector2D &b)

  2D vector subtraction.
- Vector2D FAMath::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 FAMath::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 FAMath::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 FAMath::dotProduct (const Vector2D &a, const Vector2D &b)

  Returns the dot product between two 2D vectors.
- float FAMath::length (const Vector2D &v)

  Returns the length(magnitude) of the 2D vector v.
- Vector2D FAMath::norm (const Vector2D &v)

  Normalizes the 2D vector v.
- Vector2D FAMath::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 FAMath::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 FAMath::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.

6.6.1 Detailed Description

File has a 2D Vector class under the namespace FAMath.

6.7 FAVector2D.h

Go to the documentation of this file.

```
#pragma once

#include "FAMathUtility.h"

namespace FAMath
{
    class Vector2D
    {
    public:


        Vector2D();

        Vector2D(float x, float y);

        Vector2D(Vector3D v);

        Vector2D(Vector4D v);

        float& x();

        float& y();

        const float& x() const;

        const float& y() const;

        Vector2D& operator+=(const Vector2D& b);

        Vector2D& operator-=(const Vector2D& b);

        Vector2D& operator*=(const float& k);

        Vector2D& operator/=(const float& k);

    private:
        float m_x;
        float m_y;
    };

    bool zeroVector(const Vector2D& a);

    Vector2D operator+(const Vector2D& a, const Vector2D& b);

    Vector2D operator-(const Vector2D& v);

    Vector2D operator-(const Vector2D& a, const Vector2D& b);

    Vector2D operator*(const Vector2D& a, const float& k);

    Vector2D operator*(const float& k, const Vector2D& a);

    Vector2D operator/(const Vector2D& a, const float& k);

    float dotProduct(const Vector2D& a, const Vector2D& b);

    float length(const Vector2D& v);

    Vector2D norm(const Vector2D& v);

    Vector2D PolarToCartesian(const Vector2D& v);

    Vector2D CartesianToPolar(const Vector2D& v);

    Vector2D Projection(const Vector2D& a, const Vector2D& b);

#if defined(_DEBUG)
    void print(const Vector2D& v);
#endif
}
```

6.8 FAVector3D.h File Reference

File has a 3D Vector class under the namespace FAMath.

```
#include "FAMathUtility.h"
```

Classes

- class FAMath::Vector3D

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

Namespaces

- namespace FAMath

  Has utility functions, Vector2D, Vector3D, Vector4D, Matrix4x4, and Quaternion classes.

Functions

- bool FAMath::zeroVector (const Vector3D &a)

  Returns true if a is the zero vector.
- Vector3D FAMath::operator+ (const Vector3D &a, const Vector3D &b)

  3D vector addition.
- Vector3D FAMath::operator- (const Vector3D &v)

  3D vector negeation.
- Vector3D FAMath::operator- (const Vector3D &a, const Vector3D &b)

  3D vector subtraction.
- Vector3D FAMath::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 FAMath::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 FAMath::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 FAMath::dotProduct (const Vector3D &a, const Vector3D &b)

  Returns the dot product between two 3D vectors.
- Vector3D FAMath::crossProduct (const Vector3D &a, const Vector3D &b)

  Returns the cross product between two 3D vectors.
- float FAMath::length (const Vector3D &v)

  Returns the length(magnitude) of the 3D vector v.
- Vector3D FAMath::norm (const Vector3D &v)

  Normalizes the 3D vector v.
- Vector3D FAMath::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 FAMath::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 FAMath::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 FAMath::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 FAMath::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.

6.8.1 Detailed Description

File has a 3D Vector class under the namespace FAMath.

6.9 FAVector3D.h

Go to the documentation of this file.

```
#pragma once

#include "FAMathUtility.h"

namespace FAMath
{
    class Vector3D
    {
    public:


        Vector3D();

        Vector3D(float x, float y, float z);

        Vector3D(Vector2D v, float z = 0.0f);

        Vector3D(Vector4D v);

        float& x();

        float& y();

        float& z();

        const float& x() const;

        const float& y() const;

        const float& z() const;

        Vector3D& operator+=(const Vector3D& b);

        Vector3D& operator-=(const Vector3D& b);

        Vector3D& operator*=(const float& k);

        Vector3D& operator/=(const float& k);

    private:
        float m_x;
        float m_y;
        float m_z;
    };

    bool zeroVector(const Vector3D& a);

    Vector3D operator+(const Vector3D& a, const Vector3D& b);

    Vector3D operator-(const Vector3D& v);

    Vector3D operator-(const Vector3D& a, const Vector3D& b);

    Vector3D operator*(const Vector3D& a, const float& k);

    Vector3D operator*(const float& k, const Vector3D& a);

    Vector3D operator/(const Vector3D& a, const float& k);

    float dotProduct(const Vector3D& a, const Vector3D& b);

    Vector3D crossProduct(const Vector3D& a, const Vector3D& b);

    float length(const Vector3D& v);

    Vector3D norm(const Vector3D& v);

    Vector3D CylindricalToCartesian(const Vector3D& v);

    Vector3D CartesianToCylindrical(const Vector3D& v);

    Vector3D SphericalToCartesian(const Vector3D& v);

    Vector3D CartesianToSpherical(const Vector3D& v);

    Vector3D Projection(const Vector3D& a, const Vector3D& b);


#if defined(_DEBUG)
    void print(const Vector3D& v);
#endif
}
```

6.10 FAVector4D.h File Reference

File has a 4D Vector class under the namespace FAMath.

```
#include "FAMathUtility.h"
```

Classes

- class FAMath::Vector4D

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

Namespaces

- namespace FAMath

  Has utility functions, Vector2D, Vector3D, Vector4D, Matrix4x4, and Quaternion classes.

Functions

- bool FAMath::zeroVector (const Vector4D &a)

  Returns true if a is the zero vector.
- Vector4D FAMath::operator+ (const Vector4D &a, const Vector4D &b)

  4D vector addition.
- Vector4D FAMath::operator- (const Vector4D &v)

  4D vector negation.
- Vector4D FAMath::operator- (const Vector4D &a, const Vector4D &b)

  4D vector subtraction.
- Vector4D FAMath::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 FAMath::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 FAMath::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 FAMath::dotProduct (const Vector4D &a, const Vector4D &b)

  Returns the dot product between two 4D vectors.
- float FAMath::length (const Vector4D &v)

  Returns the length(magnitude) of the 4D vector v.
- Vector4D FAMath::norm (const Vector4D &v)

  Normalizes the 4D vector v.
- Vector4D FAMath::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.

6.10.1 Detailed Description

File has a 4D Vector class under the namespace FAMath.

6.11 FAVector4D.h

Go to the documentation of this file.

```
#pragma once

#include "FAMathUtility.h"

namespace FAMath
{
    class Vector4D
    {
    public:
        Vector4D();

        Vector4D(float x, float y, float z, float w);

        Vector4D(Vector2D v, float z = 0.0f, float w = 0.0f);

        Vector4D(Vector3D v, float w = 0.0f);

        float& x();

        float& y();

        float& z();

        float& w();

        const float& x() const;

        const float& y() const;

        const float& z() const;

        const float& w() const;

        Vector4D& operator+=(const Vector4D& b);

        Vector4D& operator-=(const Vector4D& b);

        Vector4D& operator*=(const float& k);

        Vector4D& operator/=(const float& k);

    private:
        float m_x;
        float m_y;
        float m_z;
        float m_w;
    };

    bool zeroVector(const Vector4D& a);

    Vector4D operator+(const Vector4D& a, const Vector4D& b);

    Vector4D operator-(const Vector4D& v);

    Vector4D operator-(const Vector4D& a, const Vector4D& b);

    Vector4D operator*(const Vector4D& a, const float& k);

    Vector4D operator*(const float& k, const Vector4D& a);

    Vector4D operator/(const Vector4D& a, const float& k);

    float dotProduct(const Vector4D& a, const Vector4D& b);

    float length(const Vector4D& v);

    Vector4D norm(const Vector4D& v);

    Vector4D Projection(const Vector4D& a, const Vector4D& b);


#if defined(_DEBUG)
    void print(const Vector4D& v);
#endif
}
```

Index