MAT library - Documentation

Rotunno Noah

July 5, 2025

Contents

1 Introduction

A mathematical library that implement mathematical classes and functions. The header files of the library has to be included in the "include" folder of your application. No dynamicly link library (dll) or static library (lib) are required. The file *Math.hpp* has to be included:

```
#include <Math.hpp>
```

Some type definitions are already implemented. To choose if you want to use these type definitions, the macro *MAT_TYPEDEF* has to be defined to 1 or 0 to respectively use the type definitions or not. This has to be done **before** including *Math.hpp*. Example :

```
#define MAT_TYPEDEF 1 /* use 0 to not use the type definitions */
#include <Math.hpp>
```

2 Types

This section contain the types of MatObject, the type of content allowed and the function associated with them,

2.1 type_mat class

Type of MatObject that exist

```
enum class type_mat
```

2.2 type_content class

Type of content in the MatObject that are allowed

```
enum class type_content
```

2.3 type_traits structure

Function to get the type of content (see specialisation below)

```
template<typename T> struct type_traits
```

Template Parameters
T Type of content

2.3.1 get_type

Get the type enum value

```
template<typename T> type_content get_type()
```

Template Parameters
T The type
Return
The enum value, or unvalid value if not supported type

2.3.2 to_str

Transform a type of MatObject in a string

```
inline std::string to_str(type_mat t)
```

Parameters
t Type of MatObject
Return
A string refering to the type of the MatObject

Transform a type of content in a string

```
inline std::string to_str(type_content t)
```

Parameters
t Type of content
Return
A string refering to the type of the content

2.3.3 to_type_mat

Transform a string to the type of MatObject

```
inline type_mat to_type_mat(std::string t)
```

Parameters
t The string
Return
The type of MatObject

2.3.4 to_type_content

Transform a string to the type of content

```
inline type_content to_type_content(std::string t)
```

Parameters
t The string
Return
The type of content

3 Classes

This section contain the documentation and the prototype of the different classes. There is also the different functions associated to the classes. Each method and function has de brief explanation and the parameters and template parameters are explained. There is also a brief explanation on the return and if the method or function has some warning to consider.

3.1 MatObject class

The base class for all object in the MAT library that need multiple components

```
template<typename T, uint32_t SIZE> class MatObject
```

Template Parameters
T Type of the content
SIZE Number of components in the object

3.1.1 MatObject

Constructor

```
MatObject()
```

Warning
Throw a runtime error if the content type is unvalid

3.1.2 get_type_object

Get the type of the MatObject

```
virtual type_mat get_type_object() const
```

Return
Type of the MatObject

3.1.3 get_type_content

Get the type of the content of the Type of the MatObject

```
type_content get_type_content() const
```

Return
Type of the content in the MatObject

3.1.4 get_array_size

Get the size of the array containing the component of the MatObject

```
uint32_t get_array_size() const
```

Return
Number of component in the MatObject

3.1.5 data

Get a constant reference to the array of the MatObject

```
const std::array<T,SIZE>& data() const
```

Return

A constant reference to the array

Get a reference to the array of the MatObject

```
std::array<T,SIZE>& data()
```

Return
A reference to the array

3.1.6 copy

Copy data into the MatObject

```
void copy(T* start)
```

Parameters
start A pointer to the first element that will be copy

3.1.7 operator==

Is equal operator (true if all components are equal)

```
bool operator==(const MatObject<T,SIZE>& m) const
```

Parameters
v The MatObject that will be compared
Return
If the MatObjects are equal

3.1.8 operator!=

Is different operator (true if at least one component is different)

```
bool operator!=(const MatObject<T,SIZE>& v) const
```

Parameters
v The MatObject that will be compared
Return
If the MatObjects are different

3.1.9 begin

Get the begin iterator

```
iterator begin()
```

Return
The begin iterator

Get the constant begin iterator

```
const_iterator begin() const
```

Return
The constant begin iterator

3.1.10 end

Get the end iterator

```
iterator end()
```

Return
The end iterator

Get the constant end iterator

```
const_iterator end() const
```

Return
The constant end iterator

3.1.11 m_component

Component of the object

```
std::array<T, SIZE> m_component
```

3.2 BaseVector class

Class that represent a vector without any operations

```
template<typename T, uint32_t N> class BaseVector
```

Template Parameters
T Type of vector
N Dimension of the vector

3.2.1 BaseVector

Default constructor (fill with 0)

```
BaseVector()
```

Constructor with an array

```
BaseVector(const std::array<T,N>& init_array)
```

Parameters

init_array Array of the components

Constructor with an initializer list

```
BaseVector(const std::initializer_list<T>& init_list)
```

Parameters
init_list Initializer list containing the components

3.2.2 get_type_object

Get the type of the object

```
virtual type_mat get_type_object() const override
```

Return
The type of the object

3.2.3 size

Get the size (dimension) of the vector

```
uint32_t size() const
```

Return
The size (dimension) of the vector

3.2.4 operator[]

Constant accessor operator

```
const T& operator[](uint32_t index) const
```

Parameters
index Index of the element
Return
A constant reference to the element

Accessor operator

```
T& operator[](uint32_t index)
```

Parameters
index Index of the element
Return
A reference to the element

3.2.5 operator Vector<T,N>&

Cast to a vector

```
operator Vector<T,N>&()
```

3.2.6 operator Complex<T>&

Cast to a complex

```
operator Complex<T>&()
```

Warning
Only if N = 2

3.2.7 operator Quaternion<T>&

Cast to a quaternion

```
operator Quaternion<T>&()
```

Warning
Only if N = 4

3.3 BaseVector functions

3.3.1 operator«

Stream operator, write the vector into a stream

```
template<typename T, uint32_t N> std::ostream& operator<<(std::ostream&
    stream, const BaseVector<T,N>& v)
```

Template Parameters
T The type of vector
N The size of the vector
Parameters
stream The stream
v The vector
Return
The stream with the modifications

3.3.2 cast

Cast a vector of type 1 to another vector of type 2

```
template<typename T1, typename T2, uint32_t N> BaseVector<T2,N> cast(
    const BaseVector<T1,N>& v)
```

Template Parameters
T1 Current type
T2 New type
N Size of vector
Parameters

v The current vector

Return

The vector of the new type

3.3.3 min

Get the minimum value of a vector

```
template<typename T, uint32_t N> T min(const BaseVector<T,N>& v)
```

Template Parameters

T Type of vector

N Size of vector

Parameters

v The vector

Return

The smallest component

Get the minimum between components and a scalar

```
template<typename T, uint32_t N> BaseVector<T,N> min(BaseVector<T,N> v, T
     a)
```

Template Parameters

T Type of vector

N Size of vector

Parameters

v The vector

a The scalar

Return

A vector with minimum between the component and the scalar

Get the minimum for each components of two vectors

```
template<typename T, uint32_t N> BaseVector<T,N> min(BaseVector<T,N> v1,
    const BaseVector<T,N>& v2)
```

Template Parameters

T Type of vectors

N Size of vectors

Parameters

v1 First vector

v2 Second vector

Return

A vector containing the smallest values between the components of the two vectors

3.3.4 max

Get the maximum value of a vector

```
template<typename T, uint32_t N> T max(const BaseVector<T,N>& v)
```

Template Parameters
T Type of vector
N Size of vector
Parameters
v The vector
Return
The biggest component

Get the maximum between components and a scalar

```
template<typename T, uint32_t N> BaseVector<T,N> max(BaseVector<T,N> v, T
    a)
```

Template Parameters
T Type of vector
N Size of vector
Parameters
v The vector
a The scalar
Return
A vector with maximum between the component and the scalar

Get the maximum for each components of two vectors

```
template<typename T, uint32_t N> BaseVector<T,N> max(BaseVector<T,N> v1,
    const BaseVector<T,N>& v2)
```

Template Parameters
T Type of vectors
N Size of vectors
Parameters
v1 First vector
v2 Second vector
Return
A vector containing the biggest values between the components of the two vectors

3.3.5 abs

Return a vector with absolute components

```
template<typename T, uint32_t N> BaseVector<T,N> abs(const BaseVector<T,N
   >& v)
```

Template Parameters
T Type of vector
N Size of vector
Parameters

v The vector

Return

The vector with absolute component

3.3.6 sign

Return a vector of +1 or -1 whether the component is > 0 or < 0

```
template<typename T, uint32_t N> BaseVector<T,N> sign(BaseVector<T,N> v)
```

Template Parameters

T Type of vector

N Size of vector

Parameters

v The vector

Return

A vector of the sign of the components

3.4 Vector class

Mathematical vector

```
template<typename T, uint32_t N> class Vector
```

Template Parameters

T Type of the vector

N Size of the vector

3.4.1 Vector

Default constructor, fill the vector with value 0

```
Vector()
```

Constructor, fill the vector with a value

```
Vector(T fill_val)
```

Parameters

fill_val Value that fill the vector

Constructor, copy an array

```
Vector(const std::array<T,N>& init_array)
```

Parameters

init_array Initial array

Constructor, copy an initializer list

```
Vector(const std::initializer_list<T>& init_list)
```

Parameters
init_list Initializer list

3.4.2 get_type_object

Get the type of the object

```
virtual type_mat get_type_object() const override
```

Return
The type of the object

3.4.3 norm2

Compute the square of the norm

```
T norm2() const
```

Return
The square of the norm

3.4.4 norm

Compute the the norm

```
T norm() const
```

Return
The norm

3.4.5 normalize

Normalize the vector

```
void normalize()
```

3.4.6 operator BaseVector<T,N>&

Cast to a base vector

```
operator BaseVector<T,N>&()
```

3.4.7 operator+=

Sum each vector element with a scalar

```
Vector& operator+=(T a)
```

Parameters
a Scalar
Return
Reference to the vector

Add two vectors

```
Vector& operator+=(const Vector<T,N>& v)
```

Parameters
v Vector
Return
Reference to the vector

3.4.8 operator-=

Subtract each vector element with a scalar

```
Vector& operator-=(T a)
```

Parameters
a Scalar
Return
Reference to the vector

Substract two vectors

```
Vector& operator-=(const Vector<T,N>& v)
```

Parameters
v Vector
Return
Reference to the vector

3.4.9 operator*=

Multiply each vector element with a scalar

```
Vector& operator*=(T a)
```

Parameters
a Scalar
Return
Reference to the vector

Multiply each element of the two vectors

```
Vector& operator*=(const Vector<T,N>& v)
```

Parameters

v Vector

Return

Reference to the vector

Warning

It is not a dot product, use the function dot

3.4.10 operator/=

Divide each vector element with a scalar

```
Vector& operator/=(T a)
```

Parameters

a Scalar

Return

Reference to the vector

Divide each element of the two vectors

```
Vector& operator/=(const Vector<T,N>& v)
```

Parameters

v Vector

Return

Reference to the vector

3.5 Vector functions

3.5.1 operator«

Stream operator, write the vector into a stream

```
template<typename T, uint32_t N> std::ostream& operator<<(std::ostream&
    stream, const Vector<T,N>& v)
```

Template Parameters

T The type of vector

N The size of the vector

Parameters

stream The stream

v The vector

Return

The stream with the modifications

3.5.2 operator+

Add a scalar to each element of the vector

```
template<typename T, uint32_t N> Vector<T,N> operator+(Vector<T,N> v1, T
    a)
```

Template Parameters
T The type of vector
N The size of the vector
Parameters
v1 The vector
a The scalar
Return
The new vector

Add a scalar to each element of the vector

```
template<typename T, uint32_t N> Vector<T,N> operator+(T a, Vector<T,N>
    v1)
```

Template Parameters
T The type of vector
N The size of the vector
Parameters
a The scalar
v1 The vector
Return
The new vector

Addition of two vectors

```
template<typename T, uint32_t N> Vector<T,N> operator+(Vector<T,N> v1,
    const Vector<T,N>& v2)
```

Template Parameters
T The type of vector
N The size of the vector
Parameters
v1 First vector
v2 Second vector
Return
The addition of the two vectors

3.5.3 operator-

Substract a scalar to each element of the vector

```
template<typename T, uint32_t N> Vector<T,N> operator-(Vector<T,N> v1, T
    a)
```

Template Parameters
T The type of vector
N The size of the vector
Parameters
v1 The vector
a The scalar

Return
The new vector

Substract element of the vector by a scalar

```
template<typename T, uint32_t N> Vector<T,N> operator-(T a, Vector<T,N>
    v1)
```

Template Parameters
T The type of vector
N The size of the vector
Parameters
a The scalar
v1 The vector
Return
The new vector

Substraction of two vectors

```
template<typename T, uint32_t N> Vector<T,N> operator-(Vector<T,N> v1,
    const Vector<T,N>& v2)
```

Template Parameters
T The type of vector
N The size of the vector
Parameters
v1 First vector
v2 Second vector
Return
The substraction of the two vectors

Each element is multiply by -1

```
template<typename T, uint32_t N> Vector<T,N> operator-(Vector<T,N> v1)
```

Template Parameters
T The type of vector
N The size of the vector
Parameters
v1 The vector
Return
Minus the vector (-v)

3.5.4 operator*

Multiply each component of a vector by a scalar

```
template<typename T, uint32_t N> Vector<T,N> operator*(Vector<T,N> v1, T
    a)
```

Template Parameters

T The type of vector

N The size of the vector

Parameters

v1 The vector

a The scalar

Return

The new vector

Multiply each component of a vector by a scalar

```
template<typename T, uint32_t N> Vector<T,N> operator*(T a, Vector<T,N>
    v1)
```

Template Parameters

T The type of vector

N The size of the vector

Parameters

a The scalar

v1 The vector

Return

The new vector

Multiply each element of two vectors

```
template<typename T, uint32_t N> Vector<T,N> operator*(Vector<T,N> v1,
    const Vector<T,N>& v2)
```

Template Parameters

T The type of vector

N The size of the vector

Parameters

v1 First vector

v2 Second vector

Return

The multiplication result

Warning

It is not a dot product, use the function dot

3.5.5 operator/

Divide each component of a vector by a scalar

```
template<typename T, uint32_t N> Vector<T,N> operator/(Vector<T,N> v1, T
    a)
```

Template Parameters

T The type of vector

N The size of the vector

Parameters

v1 The vector
a The scalar
Return
The new vector

Divide each element of two vectors

```
template<typename T, uint32_t N> Vector<T,N> operator/(Vector<T,N> v1,
    const Vector<T,N>& v2)
```

Template Parameters
T The type of vector
N The size of the vector
Parameters
v1 First vector
v2 Second vector
Return
The division result

3.5.6 sum

Sum all the component of a vector

```
template<typename T, uint32_t N> T sum(const Vector<T,N>& v)
```

Template Parameters
T Type of the vector
N Size of the vector
Parameters
v The vector
Return
The sum of the component of the vector

3.5.7 merge

Merge two vectors into one vector

```
template<typename T, uint32_t N, uint32_t M> Vector<T, N+M> merge(const
    Vector<T,N>& v1, const Vector<T,M>& v2)
```

Template Parameters
T Type of the vector
N Size of the first vector
M Size of the second vector
Parameters
v1 First vector
v2 Second vector
Return
Merged vector

3.5.8 append

Append a component at the back of a vector

```cpp
template<typename T, uint32_t N> Vector<T,N+1> append(const Vector<T,N>&
    v, T value)
```

Template Parameters
T Type of the vector
N Size of the vector
Parameters
v The vector
value The new component
Return
Vector with the new element

3.5.9 insert

Insert a component in a vector

```cpp
template<typename T, uint32_t N> Vector<T,N+1> insert(const Vector<T,N>&
    v, T value, uint32_t index)
```

Template Parameters
T Type of the vector
N Size of the vector
Parameters
v The vector
value The new component
index The index where the component will be inserted
Return
Vector with the new element

3.5.10 remove

Remove a component from a vector

```cpp
template<typename T, uint32_t N> Vector<T,N-1> remove(const Vector<T,N>&v
    , uint32_t index)
```

Template Parameters
T Type of the vector
N Size of the vector
Parameters
v The vector
index The index of the component that will be removed
Return
Vector without the component

3.5.11 sub_vector

Create a sub vector from a vector

```cpp
template<typename T, uint32_t N, uint32_t I1, uint32_t I2> Vector<T,I2-I1
    +1> sub_vector(const Vector<T,N>& v)
```

Template Parameters
T Type of the vector
N Size of the vector
I1 First element of the sub vector
I2 Last element of the sub vector
Parameters
v The vector
Return
The sub vector

3.5.12 dot

Compute the dot product between two vectors

```cpp
template<typename T, uint32_t N> T dot(Vector<T,N> v1, const Vector<T,N>&
     v2)
```

Template Parameters
T Type of the vector
N Size of the vector
Parameters
v1 First vector
v2 Second vector
Return
Result of the dot product

3.5.13 cross

Compute the cross product between two vectors of dimension 2

```cpp
template<typename T> T cross(const Vector<T,2>& v1, const Vector<T,2>& v2
    )
```

Template Parameters
T Type of the vector
Parameters
v1 First vector
v2 Second vector
Return
Result of the cross product (a scalar)

Compute the cross product between two vectors of dimension 3

```cpp
template<typename T> Vector<T,3> cross(const Vector<T,3>& v1, const
    Vector<T,3>& v2)
```

Template Parameters

T Type of the vector

Parameters

v1 First vector

v2 Second vector

Return

Result of the cross product (a vector 3)

3.5.14 angle

Compute the angle between two vectors of dimension 2

```cpp
template<typename T> T angle(const Vector<T,2>& v1, const Vector<T,2>& v2
    )
```

Template Parameters

T Type of the vector

Parameters

v1 First vector

v2 Second vector

Return

Angle between the vectors

Compute the angle between two vectors of dimension 3

```cpp
template<typename T> T angle(const Vector<T,3>& v1, const Vector<T,3>& v2
    )
```

Template Parameters

T Type of the vector

Parameters

v1 First vector

v2 Second vector

Return

Angle between the vectors

3.5.15 polar

Compute the polar representation of a vector 2

```cpp
template<typename T> void polar(const Vector<T,2>& v, T& radius, T& theta
    )
```

Template Parameters

T Type of the vector

Parameters

v The vector

radius A reference to the magnitude

theta A reference to the argument (angle) of the vector

3.5.16 cylindrical

Compute the cylindrical representation of a vector 3

```
template<typename T> void cylindrical(const Vector<T,3>& v, T& radius, T&
    theta, T& z)
```

Template Parameters
T Type of the vector
Parameters
v The vector
radius A reference to the radius of the cylinder
theta A reference to the angle
z A reference to the z coordinate

3.5.17 spherical

Compute the spherical representation of a vector 3

```
template<typename T> void spherical(const Vector<T,3>& v, T& radius, T&
    theta, T& phi)
```

Template Parameters
T Type of the vector
Parameters
v The vector
radius A reference to the radius of the sphere
theta A reference to the theta angle
phi A reference to the phi angle

3.5.18 distance

Compute the distance between two vectors

```
template<typename T, uint32_t N> T distance(const Vector<T,N>& v1, const
    Vector<T,N>& v2)
```

Template Parameters
T Type of vector
N Size of vectors
Parameters
v1 First vector
v2 Second vector
Return
The distance between the vectors

3.5.19 reflect

Compute the reflected vector

```
template<typename T, uint32_t N> Vector<T,N> reflect(const Vector<T,N>&
    incident, Vector<T,N> surface_normal)
```

Template Parameters
T Type of vector
N Site of vector
Parameters
incident The incident ray
surface_normal The surface normal
Return
The refleted vector

3.5.20 refract

Compute the refracted vector

```
template<typename T, uint32_t N> Vector<T,N> refract(const Vector<T,N>&
    incident, Vector<T,N> surface_normal, T n1, T n2)
```

Template Parameters
T Type of vector
N Size of vector
Parameters
incident The incident ray
surface_normal The surface normal
n1 The refraction index of the medium of the incident ray
n2 The refraction index of the medium of the surface
Return
The refracted vector

3.5.21 face_same_direction

Get if both vector are facing in the same direction (dot(v1, v2) >? 0)

```
template<typename T, uint32_t N> bool face_same_direction(const Vector<T,
    N>& v1, const Vector<T,N>& v2)
```

Template Parameters
T Type of vector
N Size of vector
Parameters
v1 First vector
v2 Second vector
Return
True if dot(v1, v2) > 0, false otherwise

3.6 Matrix class

Mathematical matrix

```
template<typename T, uint32_t N, uint32_t M> class Matrix
```

Template Parameters
T Type of matrix

N Row
M Column

3.6.1 Matrix

Default constructor, fill the matri with value 0

```
Matrix()
```

Constructor, fill the matrix with a value

```
Matrix(T fill_val)
```

Parameters
fill_val Value that fill the matrix

Constructor, copy the data

```
Matrix(T* start)
```

Parameters
start First element to copy

Constructor, copy an initializer list of vector

```
Matrix(const std::initializer_list<Vector<T,N>>& init_list)
```

Parameters
l Initializer list of vector

Constructor, copy a initializer list

```
Matrix(const std::initializer_list<T>& init_list)
```

Parameters
l Initializer list

Constructor, copy an array

```
Matrix(const std::array<T, N*M>& init_array)
```

Parameters
l Initial array

Constructor, copy an array of vectors

```
Matrix(const std::array<Vector<T, N>, M>& init_array)
```

Parameters
l Array of vectors

3.6.2 get_type_object

Get the type of the object

```
virtual type_mat get_type_object() const override
```

Return
The type of the object

3.6.3 size

Get the size of the matrix

```
void size(uint32_t& n, uint32_t& m) const
```

Parameters
n Reference to the number of row
m Reference to the number of column

3.6.4 copy

Copy data into the matrix

```
void copy(T* start)
```

Parameters
start A pointer to the first element that will be copy

3.6.5 row

Get the number of row

```
uint32_t row() const
```

Return
The number of row

3.6.6 column

Get the number of column

```
uint32_t column() const
```

Return
The number of column

3.6.7 begin

Get the begin iterator

```
iterator begin()
```

Return
The begin iterator

Get the constant begin iterator

```
const_iterator begin() const
```

Return
The constant begin iterator

3.6.8 end

Get the end iterator

```
iterator end()
```

Return
The end iterator

Get the constant end iterator

```
const_iterator end() const
```

Return
The constant end iterator

3.6.9 operator()

Accessor to the element (i,j)

```
const T& operator()(uint32_t i, uint32_t j) const
```

Parameters
i Row
j Column
Return
A constant reference to the element (i,j)

Accessor to the element (i,j)

```
T& operator()(uint32_t i, uint32_t j)
```

Parameters
i Row

j Column

Return

A reference to the element (i,j)

3.6.10 operator[]

Accessor to the i th column vector

```
const Vector<T,N>& operator[](uint32_t i) const
```

Parameters

i Column

Return

A constant reference to the column vector i

Accessor to the i th column vector

```
Vector<T,N>& operator[](uint32_t i)
```

Parameters

i Column

Return

A reference to the column vector i

3.6.11 operator+=

Sum each component with a scalar

```
Matrix& operator+=(T a)
```

Parameters

a Scalar

Return

Reference to the matrix

Add two matrices

```
Matrix& operator+=(const Matrix<T,N,M>& v)
```

Parameters

v Matrix

Return

Reference to the matrix

3.6.12 operator-=

Substract each component with a scalar

```
Matrix& operator-=(T a)
```

Parameters
a Scalar
Return
Reference to the matrix

Substract two matrices

```
Matrix& operator-=(const Matrix<T,N,M>& v)
```

Parameters
v Matrix
Return
Reference to the matrix

3.6.13 operator*=

Multiply each component with a scalar

```
Matrix& operator*=(T a)
```

Parameters
a Scalar
Return
Reference to the matrix

Multiply each element of the two matrices

```
Matrix& operator*=(const Matrix<T,N,M>& v)
```

Parameters
v Matrix
Return
Reference to the matrix
Warning
It is not a matrix product, use the function dot

3.6.14 operator/=

Divide each component with a scalar

```
Matrix& operator/=(T a)
```

Parameters
a Scalar
Return
Reference to the matrix

Divide each element of the two matrices

```
Matrix& operator/=(const Matrix<T,N,M>& v)
```

Parameters
v Matrix
Return
Reference to the matrix

3.7 Matrix functions

3.7.1 operator«

Stream operator, write the matrix into a stream

```
template<typename T, uint32_t N, uint32_t M> std::ostream& operator<<(std
    ::ostream& stream, const Matrix<T,N,M>& m)
```

Template Parameters
T The type of matrix
N Row
M Column
Parameters
stream The stream
m The matrix
Return
The stream with the modifications

3.7.2 operator+

Add a scalar to each element of the matrix

```
template<typename T, uint32_t N, uint32_t M> Matrix<T,N,M> operator+(
    Matrix<T,N,M> m1, T a)
```

Template Parameters
T The type of matrix
N Row
M Column
Parameters
m1 The matrix
a The scalar
Return
The new matrix

Add a scalar to each element of the matrix

```
template<typename T, uint32_t N, uint32_t M> Matrix<T,N,M> operator+(T a,
     Matrix<T,N,M> m1)
```

Template Parameters

T The type of matrix

N Row

M Column

Parameters

a The scalar

m1 The matrix

Return

The new matrix

Addition of two matrices

```
template<typename T, uint32_t N, uint32_t M> Matrix<T,N,M> operator+(
    Matrix<T,N,M> m1, const Matrix<T,N,M>& m2)
```

Template Parameters

T The type of matrix

N Row

M Column

Parameters

m1 The first matrix

m2 The second matrix

Return

The addition of the matrices

3.7.3 operator-

Substract a scalar to each element of the matrix

```
template<typename T, uint32_t N, uint32_t M> Matrix<T,N,M> operator-(
    Matrix<T,N,M> m1, T a)
```

Template Parameters

T The type of matrix

N Row

M Column

Parameters

m1 The matrix

a The scalar

Return

The new matrix

Substract element of the matrix by a scalar

```
template<typename T, uint32_t N, uint32_t M> Matrix<T,N,M> operator-(T a,
     Matrix<T,N,M> m1)
```

Template Parameters

T The type of matrix

N Row

M Column

Parameters
a The scalar
m1 The matrix
Return
The new matrix

Substraction of two matrices

```
template<typename T, uint32_t N, uint32_t M> Matrix<T,N,M> operator-(
    Matrix<T,N,M> m1, const Matrix<T,N,M>& m2)
```

Template Parameters
T The type of matrix
N Row
M Column
Parameters
m1 The first matrix
m2 The second matrix
Return
The substraction of the matrices

3.7.4 operator*

Multiply each component of a matrix by a scalar

```
template<typename T, uint32_t N, uint32_t M> Matrix<T,N,M> operator*(
    Matrix<T,N,M> m1, T a)
```

Template Parameters
T The type of matrix
N Row
M Column
Parameters
m1 The matrix
a The scalar
Return
The new matrix

Multiply each component of a matrix by a scalar

```
template<typename T, uint32_t N, uint32_t M> Matrix<T,N,M> operator*(T a,
     Matrix<T,N,M> m1)
```

Template Parameters
T The type of matrix
N Row
M Column
Parameters
a The scalar
m1 The matrix

Return
The new matrix

Multiplication of each elements of two matrices

```
template<typename T, uint32_t N, uint32_t M> Matrix<T,N,M> operator*(
    Matrix<T,N,M> m1, const Matrix<T,N,M>& m2)
```

Template Parameters
T The type of matrix
N Row
M Column
Parameters
m1 The first matrix
m2 The second matrix
Return
The multiplication of each element of the matrices
Warning
This is not the matrix product. Use dot function for the matrix product

3.7.5 operator/

Divide each component of a matrix by a scalar

```
template<typename T, uint32_t N, uint32_t M> Matrix<T,N,M> operator/(
    Matrix<T,N,M> m1, T a)
```

Template Parameters
T The type of matrix
N Row
M Column
Parameters
m1 The matrix
a The scalar
Return
The new matrix

Division of each elements of two matrices

```
template<typename T, uint32_t N, uint32_t M> Matrix<T,N,M> operator/(
    Matrix<T,N,M> m1, const Matrix<T,N,M>& m2)
```

Template Parameters
T The type of matrix
N Row
M Column
Parameters
m1 The first matrix
m2 The second matrix
Return

The division of each element of the matrices

Warning

This is not the matrix product. Use dot function for the matrix product

3.7.6 min

Get the minimum value inside the matrix

```
template<typename T, uint32_t N, uint32_t M> T min(const Matrix<T,N,M>& m
    )
```

Template Parameters

T The type of matrix

N Row

M Column

Parameters

m The matrix

Return

The minimum component in the matrix

Compare each component with a value and return a matrix containing the minimum between component and value

```
template<typename T, uint32_t N, uint32_t M> Matrix<T,N,M> min(Matrix<T,N
    ,M> m, T a)
```

Template Parameters

T The type of matrix

N Row

M Column

Parameters

m The matrix

a The value

Return

A matrix containing the minimum between the component and the value

Compare each component and return a matrix containing the minimum value in each component

```
template<typename T, uint32_t N, uint32_t M> Matrix<T,N,M> min(Matrix<T,N
    ,M> m1, const Matrix<T,N,M>& m2)
```

Template Parameters

T The type of matrix

N Row

M Column

Parameters

m1 The first matrix

m2 The second matrix

Return

A matrix containing the minimum between the component of each matrix

3.7.7 max

Get the maximum value inside the matrix

```
template<typename T, uint32_t N, uint32_t M> T max(const Matrix<T,N,M>& m
    )
```

Template Parameters
T The type of matrix
N Row
M Column
Parameters
m The matrix
Return
The maximum component in the matrix

Compare each component with a value and return a matrix containing the maximum between component and value

```
template<typename T, uint32_t N, uint32_t M> Matrix<T,N,M> max(Matrix<T,N
    ,M> m, T a)
```

Template Parameters
T The type of matrix
N Row
M Column
Parameters
m The matrix
a The value
Return
A matrix containing the maximum between the component and the value

Compare each component and return a matrix containing the maximum value in each component

```
template<typename T, uint32_t N, uint32_t M> Matrix<T,N,M> max(Matrix<T,N
    ,M> m1, const Matrix<T,N,M>& m2)
```

Template Parameters
T The type of matrix
N Row
M Column
Parameters
m1 The first matrix
m2 The second matrix
Return
A matrix containing the maximum between the component of each matrix

3.7.8 abs

Apply absolute value to each component of a matrix

```cpp
template<typename T, uint32_t N, uint32_t M> Matrix<T,N,M> abs(Matrix<T,N
    ,M> m)
```

Template Parameters
T The type of matrix
N Row
M Column
Parameters
m The matrix
Return
A matrix with absolute value apply to each component

3.7.9 sign

Apply sign function to each component of a matrix

```cpp
template<typename T, uint32_t N, uint32_t M> Matrix<T,N,M> sign(Matrix<T,
    N,M> m)
```

Template Parameters
T The type of matrix
N Row
M Column
Parameters
m The matrix
Return
A matrix with sign function apply to each component

3.7.10 to_row_matrix

Transform a vector to a row matrix

```cpp
template<typename T, uint32_t N> Matrix<T,1,N> to_row_matrix(const Vector
    <T,N>& v)
```

Template Parameters
T Type
N Size
Parameters
v The vector
Return
The row matrix

3.7.11 to_column_matrix

Transform a vector to a column matrix

```cpp
template<typename T, uint32_t N> Matrix<T,N,1> to_column_matrix(const
    Vector<T,N>& v)
```

Template Parameters
T Type
N Size
Parameters
v The vector
Return
The column matrix

3.7.12 to_vector

Transform a column matrix to a vector

```
template<typename T, uint32_t N> Vector<T,N> to_vector(const Matrix<T,N
    ,1>& m)
```

Template Parameters
T Type
N Size
Parameters
m The column matrix
Return
The vector

Transform a row matrix to a vector

```
template<typename T, uint32_t N> Vector<T,N> to_vector(const Matrix<T,1,N
    >& m)
```

Template Parameters
T Type
N Size
Parameters
m The row matrix
Return
The vector

3.7.13 dot

Matrix product

```
template<typename T, uint32_t N, uint32_t M, uint32_t P> Matrix<T,N,P>
    dot(const Matrix<T,N,M>& m1, const Matrix<T,M,P>& m2)
```

Template Parameters
T Type
N Row of the first matrix
M Column of the first matrix and row of the second matrix
P Column of the second matrix
Parameters

m1 The first matrix
m2 The second matrix
Return
The result matrix

Matrix - vector product

```
template<typename T, uint32_t N, uint32_t M> BaseVector<T,N> dot(const
    Matrix<T,N,M>& m, const BaseVector<T,M>& v)
```

Template Parameters
T Type
N Row of the matrix
M Column of the matrix and size of the vector
Parameters
m The matrix
v The vector
Return
The result vector

Vector - matrix product

```
template<typename T, uint32_t N, uint32_t M> BaseVector<T,M> dot(const
    BaseVector<T,N>& v, const Matrix<T,N,M>& m)
```

Template Parameters
T Type
N Row of the matrix and size of the vector
M Column of the matrix
Parameters
v The vector
m The matrix
Return
The result vector

3.7.14 identity

Get the identity matrix

```
template<typename T, uint32_t N> Matrix<T,N,N> identity()
```

Template Parameters
T Type
N Row and column of the matrix
Return
The identity matrix

3.7.15 transpose

Get the transpose of a matrix

```
template<typename T, uint32_t N, uint32_t M> Matrix<T,M,N> transpose(
    const Matrix<T,N,M>& m)
```

Template Parameters
T Type
N Row of the matrix
M Column of the matrix
Parameters
m The matrix
Return
The transpose matrix

3.7.16 self_transpose

Transpose the matrix itself

```
template<typename T, uint32_t N> void self_transpose(Matrix<T,N,N>& m)
```

Template Parameters
T Type
N Row and column of the matrix
Parameters
m The matrix

3.7.17 direct_sum

Direct sum of two matrices

```
template<typename T, uint32_t N, uint32_t M, uint32_t P, uint32_t Q>
    Matrix<T,N+P,M+Q> direct_sum(const Matrix<T,N,M>& m1, const Matrix<T,P
    ,Q>& m2)
```

Template Parameters
T Type
N Row of the first matrix
M Column of the first matrix
P Row of the second matrix
Q Column of the second matrix
Parameters
m1 The first matrix
m2 The second matrix
Return
The direct sum of the matrices

3.7.18 kronecker_product

Kronecker product of two matrices

```
template<typename T, uint32_t N, uint32_t M, uint32_t P, uint32_t Q>
```

```
    Matrix<T,N*P,M*Q> kronecker_product(const Matrix<T,N,M>& m1, const
    Matrix<T,P,Q>& m2)
```

Template Parameters
T Type
N Row of the first matrix
M Column of the first matrix
P Row of the second matrix
Q Column of the second matrix
Parameters
m1 The first matrix
m2 The second matrix
Return
The kronecker product of the matrices

3.7.19 determinant

Compute the determinant of a matrix

```
template<typename T, uint32_t N> T determinant(Matrix<T,N,N> m)
```

Template Parameters
T Type
N Row and column of the first matrix
Parameters
m The matrix
Return
The determinant of the matrix

3.7.20 inverse

Compute the inverse of a matrix

```
template<typename T, uint32_t N> Matrix<T,N,N> inverse(Matrix<T,N,N> m)
```

Template Parameters
T Type
N Row and column of the first matrix
Parameters
m The matrix
Return
The inverse of the matrix
Warning
Does not check if the determinant is non null

3.7.21 expand

Expand a matrix (from [N x N] to [N+1 x N+1])

```
template<typename T, uint32_t N> Matrix<T,N+1,N+1> expand(const Matrix<T,
    N,N>& m)
```

Template Parameters
T Type
N Row and column of the first matrix
Parameters
m The matrix
Return
The expand matrix

3.7.22 exp

Compute the exponential of a matrix

```
template<typename T, uint32_t N> Matrix<T,N,N> exp(const Matrix<T,N,N>& m
    , T tolerence = 1e-3, uint32_t max_iteration = 30)
```

Template Parameters
T Type of matrix
N Row and column
Parameters
m The matrix
tolerence Tolerence of the computation
max_iteration Maximum number of iterations
Return
The exponential of a matrix
Warning
Tolerence must be smaller than 1, max_iteration must be smaller than 33

3.8 Complex class

Complex number

```
template<typename T> class Complex
```

Template Parameters
T Type of complex

3.8.1 Complex

Default constructor, fill the vector with value 0

```
Complex()
```

Constructor with only a real part

```
Complex(T re)
```

Parameters

re Real part

Constructor with a real and an imaginary part

```
Complex(T re, T im)
```

Parameters
re Real part
im Imaginary part

Constructor, copy an array

```
Complex(const std::array<T,2>& init_array)
```

Parameters
init_array Initial array

Constructor, copy an initializer list

```
Complex(const std::initializer_list<T>& init_list)
```

Parameters
init_list Initializer list

3.8.2 get_type_object

Get the type of the object

```
virtual type_mat get_type_object() const override
```

Return
The type of the object

3.8.3 real

Get the real part

```
T& real()
```

Return
A reference to the real part

Get the real part

```
const T& real() const
```

Return
A constant reference to the real part

3.8.4 imag

Get the imaginary part

```
T& imag()
```

Return
A reference to the imaginary part

Get the imaginary part

```
const T& imag() const
```

Return
A constant reference to the imaginary part

3.8.5 modulus2

Compute the square modulus

```
T modulus2() const
```

Return
The square modulus

3.8.6 modulus

Compute the modulus

```
T modulus() const
```

Return
The modulus

3.8.7 argument

Compute the argument

```
T argument() const
```

Return
The argument

3.8.8 conjugate

Conjugate the complex

```
void conjugate()
```

3.8.9 normalize

Normalize the complex

```
void normalize()
```

3.8.10 set_polar

Set the component from a polar form

```
void set_polar(T mod, T argu)
```

Parameters
mod The modulus of the complex
argu The argument of the complex

3.8.11 get_polar

Get the component from a polar form

```
void get_polar(T& magn, T& argu)
```

Parameters
mod A reference to the modulus of the complex
argu A reference to the argument of the complex

3.8.12 operator BaseVector<T,2>&

Cast to a base vector

```
operator BaseVector<T,2>&()
```

3.8.13 operator+=

Add a real number

```
Complex& operator+=(T a)
```

Parameters
a The real number
Return
Reference to the complex

Add two complex

```
Complex& operator+=(const Complex<T>& c)
```

Parameters
c Complex
Return

Reference to the complex

3.8.14 operator-=

Substract a real number

```
Complex& operator-=(T a)
```

Parameters
a The real number
Return
Reference to the complex

Substract two complex

```
Complex& operator-=(const Complex<T>& c)
```

Parameters
c Complex
Return
Reference to the complex

3.8.15 operator*=

Multiply by a real number

```
Complex& operator*=(T a)
```

Parameters
a The real number
Return
Reference to the complex

Multiply two complex

```
Complex& operator*=(const Complex<T>& c)
```

Parameters
c Complex
Return
Reference to the complex

3.8.16 operator/=

Divide by a real number

```
Complex& operator/=(T a)
```

Parameters
a The real number

Return
Reference to the complex

Divide two complex

```
Complex& operator/=(const Complex<T>& c)
```

Parameters
c Complex
Return
Reference to the complex

3.9 Complex functions

3.9.1 operator«

Stream operator, write the complex into a stream

```
template<typename T> std::ostream& operator<<(std::ostream& stream, const
    Complex<T>& c)
```

Template Parameters
T The type of complex
Parameters
stream The stream
c The complex
Return
The stream with the modifications

3.9.2 operator+

Add a real number to a complex

```
template<typename T> Complex<T> operator+(Complex<T> c, T a)
```

Template Parameters
T The type of complex
Parameters
c The complex
a The real number
Return
The new complex

Add a real number to a complex

```
template<typename T> Complex<T> operator+(T a, Complex<T> c)
```

Template Parameters
T The type of complex
Parameters

a The real number
c The complex
Return
The new complex

Addition of two complex

```
template<typename T> Complex<T> operator+(Complex<T> c1, const Complex<T
    >& c2)
```

Template Parameters
T The type of complex
Parameters
c1 First complex
c2 Second complex
Return
The addition of the two complex

3.9.3 operator-

Substract a real number to a complex

```
template<typename T> Complex<T> operator-(Complex<T> c, T a)
```

Template Parameters
T The type of complex
Parameters
c The complex
a The real number
Return
The new complex

Substract a complex to a real number

```
template<typename T> Complex<T> operator-(T a, Complex<T> c)
```

Template Parameters
T The type of complex
Parameters
a The real number
c The complex
Return
The new complex

Substraction of two complex

```
template<typename T> Complex<T> operator-(Complex<T> c1, const Complex<T
    >& c2)
```

Template Parameters

T The type of complex

Parameters

c1 First complex

c2 Second complex

Return

The substraction of the two complex

Multiply the complex by -1

```
template<typename T> Complex<T> operator-(const Complex<T>& c)
```

Template Parameters

T The type of complex

Parameters

c The complex

Return

Minus the complex (-z)

3.9.4 operator*

Multiply a complex by a real number

```
template<typename T> Complex<T> operator*(Complex<T> c, T a)
```

Template Parameters

T The type of complex

Parameters

c The complex

a The real number

Return

The new complex

Multiply a complex by a real number

```
template<typename T> Complex<T> operator*(T a, Complex<T> c)
```

Template Parameters

T The type of complex

Parameters

a The real number

c The complex

Return

The new complex

Multiplication of two complex

```
template<typename T> Complex<T> operator*(Complex<T> c1, const Complex<T
    >& c2)
```

Template Parameters

T The type of complex

Parameters

c1 First complex

c2 Second complex

Return

The multiplication of the two complex

3.9.5 operator/

Divide a complex by a real number

```
template<typename T> Complex<T> operator/(Complex<T> c, T a)
```

Template Parameters

T The type of complex

Parameters

c The complex

a The real number

Return

The newcomplex

Divide a real number by a complex

```
template<typename T> Complex<T> operator/(T a, const Complex<T>& c)
```

Template Parameters

T The type of complex

Parameters

a The real number

c The complex

Return

The new complex

Division of two complex

```
template<typename T> Complex<T> operator/(Complex<T> c1, const Complex<T
   >& c2)
```

Template Parameters

T The type of complex

Parameters

c1 First complex

c2 Second complex

Return

The division of the two complex

3.9.6 complex_to_rotation_matrix

Compute the rotation matrix from a complex

```
template<typename T> Matrix<T,2,2> complex_to_rotation_matrix(const
    Complex<T>& c)
```

Template Parameters
T The type of complex
Parameters
c The complex
Return
The rotation matrix

3.9.7 angle_to_complex

Compute the complex associate to a rotation of an angle

```
template<typename T> Complex<T> angle_to_complex(T angle)
```

Template Parameters
T The type of complex
Parameters
angle The angle of rotation
Return
The complex

3.9.8 complex_rotate_vec

Rotate a vector using a complex as rotation

```
template<typename T> Vector<T,2> complex_rotate_vec(const Complex<T>& c,
    Vector<T,2> v)
```

Template Parameters
T Type of complex
Parameters
c The type complex
v The vector
Return
The rotated vector

3.9.9 conjugate

Compute the conjugate of a complex

```
template<typename T> Complex<T> conjugate(Complex<T> c)
```

Template Parameters
T The type of complex
Parameters
c The complex
Return

The conjugate of the complex

Compute the conjugate of a complex matrix

```
template<typename T, uint32_t N, uint32_t M> Matrix<Complex<T>, N, M>
    conjugate(Matrix<Complex<T>, N, M> m)
```

Template Parameters
T The type of complex
N Row of the matrix
M Column of the matrix
Parameters
m The complex matrix
Return
The conjugate of the matrix

3.9.10 self_conjugate

Conjugate a complex matrix

```
template<typename T, uint32_t N, uint32_t M> void self_conjugate(Matrix<
    Complex<T>, N, M>& m)
```

Template Parameters
T Type of complex
N Row of the matrix
M Column of the matrix
Parameters
m The complex matrix

3.9.11 conjugate_transpose

Compute the conjugate transpose of a complex matrix

```
template<typename T, uint32_t N, uint32_t M> Matrix<Complex<T>, M, N>
    conjugate_transpose(const Matrix<Complex<T>, N, M>& m)
```

Template Parameters
T The type of complex
N Row of the matrix
M Column of the matrix
Parameters
m The complex matrix
Return
The conjugate transpose of the complex matrix

3.9.12 self_conjugate_transpose

Conjugate transpose a complex matrix

```
template<typename T, uint32_t N> void self_conjugate_transpose(Matrix<
   Complex<T>, N, N>& m)
```

Template Parameters
T The type of complex
N The row and column of the matrix
Parameters
m The complex matrix

3.10 Quaternion class

Quaternion

```
template<typename T> class Quaternion
```

Template Parameters
T Type of quaternion

3.10.1 Quaternion

Default constructor, quaternion is zero

```
Quaternion()
```

Constructor, quaternion is equal to a real number

```
Quaternion(T real)
```

Parameters
real The real number

Constructor, use a real part (scalar) and an imaginary part (vector 3)

```
Quaternion(T real, const Vector<T,3>& imag)
```

Parameters
real Real part (scalar)
imag Imaginary part (vector 3)

Constructor, use a real part (scalar) and an imaginary part (array 3)

```
Quaternion(T real, const std::array<T,3>& imag)
```

Parameters
real Real part (scalar)
imag Imaginary part (array 3)

Constructor, fill each component real, i, j, k

```
Quaternion(T real, T i, T j, T k)
```

Parameters
real Real component
i i component
j j component
k k component

Constructor, use an initial array 4

```
Quaternion(const std::array<T,4>& init_array)
```

Parameters
init_array Initial array

Constructor, use an initializer list

```
Quaternion(const std::initializer_list<T>& init_list)
```

Parameters
init_list Initializer list

3.10.2 get_type_object

Get the type of the object

```
virtual type_mat get_type_object() const override
```

Return
The type of the object

3.10.3 r

Get a reference to the real part

```
T& r()
```

Return
A reference to the real part

Get a constant reference to the real part

```
const T& r() const
```

Return
A constant reference to the real part

3.10.4 i

Get a reference to the i part

```
T& i()
```

Return
A reference to the i part

Get a constant reference to the i part

```
const T& i() const
```

Return
A constant reference to the i part

3.10.5 j

Get a reference to the j part

```
T& j()
```

Return
A reference to the j part

Get a constant reference to the j part

```
const T& j() const
```

Return
A constant reference to the j part

3.10.6 k

Get a reference to the k part

```
T& k()
```

Return
A reference to the k part

Get a constant reference to the k part

```
const T& k() const
```

Return
A constant reference to the k part

3.10.7 norm2

Compute the square of the norm

```
T norm2() const
```

Return
The square of the norm

3.10.8 norm

Compute the norm

```
T norm() const
```

Return
The norm

3.10.9 normalize

Normalize the quaternion

```
void normalize()
```

3.10.10 conjugate

Conjugate the quaternion

```
void conjugate()
```

3.10.11 real

Get a copy of the real component

```
T real() const
```

Return
A copy of the real component

3.10.12 imag

Get a copy of the imaginary components

```
Vector<T,3> imag() const
```

Return
A vector 3 of the imaginary components

3.10.13 operator BaseVector<T,4>&

Cast to a base vector

```
operator BaseVector<T,4>&()
```

3.10.14 operator+=

Add a real number

```
Quaternion& operator+=(T a)
```

Parameters
a The real number
Return
Reference to quaternion

Add a quaternion

```
Quaternion& operator+=(const Quaternion<T>& q)
```

Parameters
q The quaternion
Return
Reference to quaternion

3.10.15 operator-=

Substract a real number

```
Quaternion& operator-=(T a)
```

Parameters
a The real number
Return
Reference to quaternion

Substract a quaternion

```
Quaternion& operator-=(const Quaternion<T>& q)
```

Parameters
q The quaternion
Return
Reference to quaternion

3.10.16 operator*=

Multiply the quaternion by a real number

```
Quaternion& operator*=(T a)
```

Parameters
a The real number
Return

Reference to quaternion

Multiply a quaternion

```
Quaternion& operator*=(const Quaternion<T>& q)
```

Parameters
q The quaternion
Return
Reference to quaternion

3.10.17 operator/=

Divide the quaternion by a real number

```
Quaternion& operator/=(T a)
```

Parameters
a The real number
Return
Reference to quaternion

3.11 Quaternion functions

3.11.1 operator«

Stream operator, write the quaternion into a stream

```
template<typename T> std::ostream& operator<<(std::ostream& stream, const
    Quaternion<T>& q)
```

Template Parameters
T Type of quaternion
Parameters
stream The stream
q The quaternion
Return
The stream with the modifications

3.11.2 operator+

Add a real to a quaternion

```
template<typename T> Quaternion<T> operator+(Quaternion<T> q, T a)
```

Template Parameters
T Type of quaternion
Parameters
q The quaternion
a The real

Return
The new quaternion

Add a real to a quaternion

```
template<typename T> Quaternion<T> operator+(T a, Quaternion<T> q)
```

Template Parameters
T Type of quaternion
Parameters
a The real
q The quaternion
Return
The new quaternion

Addition of two quaternions

```
template<typename T> Quaternion<T> operator+(Quaternion<T> q1, const
    Quaternion<T>& q2)
```

Template Parameters
T Type of quaternion
Parameters
q1 First quaternion
q2 Second quaternion
Return
The addition of the quaternion

3.11.3 operator-

Substract a real to a quaternion

```
template<typename T> Quaternion<T> operator-(Quaternion<T> q, T a)
```

Template Parameters
T Type of quaternion
Parameters
q The quaternion
a The real
Return
The new quaternion

Substract a quaternion to a real

```
template<typename T> Quaternion<T> operator-(T a, Quaternion<T> q)
```

Template Parameters
T Type of quaternion
Parameters
a The real

q The quaternion

Return

The new quaternion

Substraction of two quaternions

```
template<typename T> Quaternion<T> operator-(Quaternion<T> q1, const
    Quaternion<T>& q2)
```

Template Parameters

T Type of quaternion

Parameters

q1 First quaternion

q2 Second quaternion

Return

The substraction of the quaternion

Multiply the quaternion by -1

```
template<typename T> Quaternion<T> operator-(const Quaternion<T>& q)
```

Template Parameters

T Type of quaternion

Parameters

q The quaternion

Return

Minus the quaternion (-q)

3.11.4 operator*

Multiply a quaternion by a real

```
template<typename T> Quaternion<T> operator*(Quaternion<T> q, T a)
```

Template Parameters

T Type of quaternion

Parameters

q The quaternion

a The real

Return

The new quaternion

Multiply a quaternion by a real

```
template<typename T> Quaternion<T> operator*(T a, Quaternion<T> q)
```

Template Parameters

T Type of quaternion

Parameters

a The real

q The quaternion
Return
The new quaternion

Multiplication of two quaternions

```
template<typename T> Quaternion<T> operator*(Quaternion<T> q1, const
    Quaternion<T>& q2)
```

Template Parameters
T Type of quaternion
Parameters
q1 First quaternion
q2 Second quaternion
Return
The multiplication of the quaternion

3.11.5 operator/

Divide a quaternion by a real

```
template<typename T> Quaternion<T> operator/(Quaternion<T> q, T a)
```

Template Parameters
T Type of quaternion
Parameters
q The quaternion
a The real
Return
The new quaternion

3.11.6 quat_to_rotation3

From a quaternion to a rotation matrix (3x3)

```
template<typename T> Matrix<T,3,3> quat_to_rotation3(const Quaternion<T>&
     q)
```

Template Parameters
T Type of quaternion
Parameters
q The quaternion
Return
The rotation matrix (3x3)

3.11.7 quat_to_rotation4

From a quaternion to a rotation matrix (4x4)

```
template<typename T> Matrix<T,4,4> quat_to_rotation4(const Quaternion<T>&
    q)
```

Template Parameters
T Type of quaternion
Parameters
q The quaternion
Return
The rotation matrix (4x4) (last line and column fill with 0 except the diagonal term which is 1)

3.11.8 quat_to_vec_of_rotation

Get the vector of the rotation

```
template<typename T> Vector<T,3> quat_to_vec_of_rotation(const Quaternion
    <T>& q)
```

Template Parameters
T Type of quaternion
Parameters
q The quaternion
Return
The vector of rotation

3.11.9 quat_to_angle_of_rotation

Get the angle of rotation

```
template<typename T> T quat_to_angle_of_rotation(const Quaternion<T>& q)
```

Template Parameters
T Type of quaternion
Parameters
q The quaternion
Return
The angle of rotation

3.11.10 quat_from_angle_vec_of_rotation

Compute a quaternion from a vector of rotation and an angle

```
template<typename T> Quaternion<T> quat_from_angle_vec_of_rotation(T
    angle, Vector<T,3> vec)
```

Template Parameters
T Type of quaternion
Parameters
angle The angle
vec The vector of rotation
Return
The quaternion that represent the rotation

3.11.11 euler_angle_to_quat

Compute the quaternion from the Euler angles

```
template<typename T> Quaternion<T> euler_angle_to_quat(T roll, T pitch, T
    yaw)
```

Template Parameters
T Type of quaternion
Parameters
roll Roll angle (around x)
pitch Pitch angle (around y)
yaw Yaw angle (around z)
Return
The quaternion that represent the rotation

3.11.12 quat_to_euler_angle

Get the Euler angles from a quaternion

```
template<typename T> void quat_to_euler_angle(const Quaternion<T>& q, T&
    roll, T& pitch, T& yaw)
```

Template Parameters
T Type of quaternion
Parameters
q The quaternion
roll Reference to the roll angle (around x)
pitch Reference to the pitch angle (around y)
yaw Reference to the yaw angle (around z)

3.11.13 quat_rotate_vec

Rotate a vector using a quaternion

```
template<typename T> Vector<T,3> quat_rotate_vec(const Quaternion<T>& q,
    const Vector<T,3>& v)
```

Template Parameters
T Type of quaternion and vector
Parameters
q The quaternion
v The vector
Return
The rotated vector

3.11.14 conjugate

Conjugate the quaternion

```
template<typename T> Quaternion<T> conjugate(Quaternion<T> q)
```

Template Parameters

T Type of quaternion

Parameters

q The quaternion

Return

The conjugated quaternion

3.11.15 inverse

Inverse the quaternion

```
template<typename T> Quaternion<T> inverse(const Quaternion<T>& q)
```

Template Parameters

T Type of quaternion

Parameters

q The quaternion

Return

The inversed quaternion