vm_math

1.0.7

Generated by Doxygen 1.8.3.1

Fri May 17 2013 11:56:59

Contents

1 The vm_Math C++ template Library 1

1.1 Copyright . . . 1

1.2 Overview . . . 1

1.3 vm_VMath common numerical operations . . . 2

1.4 vm_V3Math common numerical operations . . . 2

1.5 vm_M3math common numerical operations . . . 2

2 Module Index 2

2.1 Modules . . . 2

3 Hierarchical Index 3

3.1 Class Hierarchy . . . 3

4 Class Index 4

4.1 Class List . . . 4

5 Module Documentation 4

5.1 vm_VMath common numerical operations . . . 4

5.2 vm_V3Math common numerical operations . . . 5

5.3 vm_M3Math common numerical operations . . . 6

5.4 Index calculations . . . 7

5.4.1 Detailed Description . . . 7

5.4.2 Function Documentation . . . 7

5.5 Initialize a matrix. . . . 8

5.5.1 Detailed Description . . . 8

5.5.2 Function Documentation . . . 8

5.6 For a matrix, insert or extract a vector. . . . 9

5.6.1 Detailed Description . . . 9

5.6.2 Function Documentation . . . 9

5.7 Matrix Vector operations. . . . 11

5.7.1 Detailed Description . . . 11

5.7.2 Function Documentation . . . 11

5.8 Matrix Matrix operations. . . . 12

5.8.1 Detailed Description . . . 12

5.9 IO operations. . . . 13

5.9.1 Detailed Description . . . 13

5.9.2 Function Documentation . . . 13

5.10 Set vector components 14
5.10.1 Detailed Description 14
5.10.2 Function Documentation 14
5.11 Normalize vectors 15
5.11.1 Detailed Description 15
5.11.2 Function Documentation 15
5.12 Tests for normality, orthogonality 16
5.12.1 Detailed Description 16
5.12.2 Function Documentation 16
5.13 Dot and Cross products. 18
5.13.1 Detailed Description 18
5.13.2 Function Documentation 18
5.14 IO operations. 19
5.14.1 Detailed Description 19
5.14.2 Function Documentation 19
5.15 Copy routines 20
5.15.1 Detailed Description 20
5.15.2 Function Documentation 20
5.16 Set functions 21
5.16.1 Detailed Description 21
5.16.2 Function Documentation 21
5.17 Componentwise op_eq operations (+=, -=,=, /=) 22
5.17.1 Detailed Description 22
5.17.2 Function Documentation 22
5.18 {add,sub,mul,div}: componentwise binary operations. 24
5.18.1 Detailed Description 24
5.18.2 Function Documentation 24
5.19 Linear combinations 27
5.19.1 Detailed Description 27
5.19.2 Function Documentation 27
5.20 I/O operations. 28
5.20.1 Detailed Description 28
5.20.2 Function Documentation 28
6 Class Documentation 29
6.1 vm_M3Math< T_fp > Class Template Reference 29
6.1.1 Detailed Description 30

6.2 vm_V3Math< T_fp > Class Template Reference 30
6.2.1 Detailed Description 31
6.2.2 Member Typedef Documentation 32
6.3 vm_VMath< T_fp, N_len > Class Template Reference 32
6.3.1 Detailed Description 33
6.3.2 Member Typedef Documentation 33

Index **33**

1 The vm_Math C++ template Library

1.1 Copyright

Author

Terry Gaetz

1.2 Overview

The vm_math C++ library is a collection of templatized routines to deal efficiently with low-level floating point vectors, 3-vectors, and 3x3 matrices. It is split up into several sub-packages (see the **Modules** page for more information).

The vm_math package brings together a number of functions for operating on 3-vectors and 3-matrices, and for creating orthonormal 3-matrices corresponding to proper rotations (i.e., orthonormal matrices preserving the coordinate system parity).

These are implemented as inlined static member functions; the functions are bundled into classes in order to take them out of the global namespace. By making the functions static member functions, they can be called without recourse to a an instantiated object, e.g.

```
double v1[] = { 1.3, 2.4, 7.2, 9.7 };
double v2[] = { 7.2, 1.9, 3.1, 4.1 };
double foo = vm_VMath<double,4>::dot( v2, v1 );
```

The package currently contains 3 classes:

- vm_VMath<T_fp,N_len>: common numerical operations on N_len-long one-dimensional arrays of floating point type T_fp. This class captures operations which do not depend on the explicit 3-vector or 3x3-matrix properties.

- vm_V3math<T_fp>: common numerical operations on 3-vectors of floating point type T_fp. Publicly derives from vm_VMath<T_fp,3>. This class adds in specifically 3-vector functionality such as vector cross products.
- vm_M3math<T_fp>: common numerical operations on 3x3 matrices of floating point type T_fp. The matrix is assumed to be stored as a contiguous one-dimensional array of 9 T_fp's. Publicly derives from vm_VMath<T_fp,9>. This class adds in specifically matrix properties such as multiplication and matrix-vector multiplication.

All members are inlined, so the package consists only of header files.

1.3 vm_VMath common numerical operations

on N_len-long * 1 dimensional arrays of T_fp

These routines provide the basic low-level support for 1-dimensional arrays of floating point values. The library is a C++ template library, with template parameters T_fp and N_len. T_fp is the floating point type. It must be a simple type (float or double), since it is assumed that the array can be copied as plain ol' data using memcpy. N_len is the dimension of the vector, assumed to be small, e.g., 9 for a flat representation of a 3x3 matrix. The class has no state information with all functionality implemented as (we hope) inlined functions.

To use these routines, ensure that you include the correct header file:

```
#include <vm_vmath/vm_vmath.h>
```

or

```
#include <vm_vmath/vm_math.h>
```

1.4 vm_V3Math common numerical operations

on 3-vectors of floating point values

These routines provide the basic low-level support for 3-vectors of floating point values of type T_fp (float or double). The library is a C++ template library, templatized on T_fp.

To use these routines, ensure that you include the correct header file:

```
#include <vm_vmath/vm_v3math.h>
```

or

```
#include <vm_vmath/vm_math.h>
```

1.5 vm_M3math common numerical operations

on 3x3 matrices of floating point values

These routines provide the basic low-level support for 3x3 matrices of floating point values of type T_fp (float or double). The library is a C++ template library, templatized on T_fp.

To use these routines, ensure that you include the correct header file:

```
#include <vm_vmath/vm_m3math.h>
```

or

```
#include <vm_vmath/vm_math.h>
```

2 Module Index

2.1 Modules

Here is a list of all modules:

vm_VMath common numerical operations 4

vm_V3Math common numerical operations 5

vm_M3Math common numerical operations 6

Index calculations 7

Initialize a matrix. 8

For a matrix, insert or extract a vector. 9

Matrix Vector operations. 11

Matrix Matrix operations. 12

IO operations. 13

Set vector components 14

Normalize vectors 15

Tests for normality, orthogonality 16

Dot and Cross products. 18

IO operations. 19

Copy routines 20

Set functions 21

Componentwise op_eq operations (+=, -=,=, /=) 22

{add,sub,mul,div}: componentwise binary operations. 24

Linear combinations 27

I/O operations. 28

3 Hierarchical Index

3.1 Class Hierarchy

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

vm_VMath< T_fp, N_len > 32

vm_VMath< T_fp, 3 > 32

vm_V3Math< T_fp > 30

vm_VMath< T_fp, 9 > 32

vm_M3Math< T_fp > 29

4 Class Index

4.1 Class List

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

vm_M3Math< T_fp > 29

vm_V3Math< T_fp > 30

vm_VMath< T_fp, N_len > 32

5 Module Documentation

5.1 vm_VMath common numerical operations

5.2 vm_V3Math common numerical operations

5.3 vm_M3Math common numerical operations

5.4 Index calculations

Functions

- static int vm_M3Math< T_fp >::at (int i, int j)

5.4.1 Detailed Description

5.4.2 Function Documentation

5.4.2.1 template<class T_fp > int vm_M3Math< T_fp >::at (int *i*, int *j*) [inline], [static]

Index calculation.

```
Array value m[i][j] is *(m + i*EColStride + j)
```

Returns

offset of index set i, j (used for flat array storage)

Parameters

i	index
j	index

Definition at line 290 of file vm_m3math.h.

5.5 Initialize a matrix.

Functions

- static void vm_M3Math< T_fp >::init_by_row (T_fp m[ ], T_fp const row0[ ], T_fp const row1[ ], T_fp const row2[ ])
- static void vm_M3Math< T_fp >::init_by_col (T_fp m[ ], T_fp const col0[ ], T_fp const col1[ ], T_fp const col2[ ])
- static void vm_M3Math< T_fp >::dyad_product (T_fp m[ ], T_fp const v1[ ], T_fp const v2[ ])

5.5.1 Detailed Description

5.5.2 Function Documentation

5.5.2.1 template<class T_fp > void **vm_M3Math**< T_fp >::dyad_product (T_fp *m[ ]*, T_fp const *v1[ ]*, T_fp const *v2[ ]*) `[inline], [static]`

Initialize a matrix to a dyadic (outer) product of vectors

For each i, j: m[at(i,j)] = v1[i] ∗ v2[j]

Parameters

m	matrix (as stored flat 1D array)
v1	1st vector
v2	2nd vector

Definition at line 361 of file vm_m3math.h.

5.5.2.2 template<class T_fp > void **vm_M3Math**< T_fp >::init_by_col (T_fp *m[ ]*, T_fp const *col0[ ]*, T_fp const *col1[ ]*, T_fp const *col2[ ]*) `[inline], [static]`

Initialize matrix by column; set columns of m to col0, col1, col2.

Parameters

m	matrix (as stored flat 1D array)
col0	1st column vector
col1	2nd column vector
col2	3rd column vector

Definition at line 308 of file vm_m3math.h.

5.5.2.3 template<class T_fp > void **vm_M3Math**< T_fp >::init_by_row (T_fp *m[ ]*, T_fp const *row0[ ]*, T_fp const *row1[ ]*, T_fp const *row2[ ]*) `[inline], [static]`

Initialize matrix by row; set rows of m to row0, row1, row2.

Parameters

m	matrix (as stored flat 1D array)
row0	1st row vector
row1	2nd row vector
row2	3rd row vector

Definition at line 298 of file vm_m3math.h.

5.6 For a matrix, insert or extract a vector.

Functions

- static void vm_M3Math< T_fp >::inject_row (T_fp m[], T_fp const row[], int whichrow)
- static void vm_M3Math< T_fp >::inject_col (T_fp m[], T_fp const col[], int whichcol)
- static void vm_M3Math< T_fp >::extract_row (T_fp const m[], T_fp row[], int whichrow)
- static void vm_M3Math< T_fp >::extract_col (T_fp const m[], T_fp col[], int whichcol)

5.6.1 Detailed Description

5.6.2 Function Documentation

5.6.2.1 template<class T_fp > void vm_M3Math< T_fp >::extract_col (T_fp const *m[]*, T_fp *col[]*, int *whichcol*) [inline], [static]

Copy a given column of m to a vector.

Parameters

m	matrix (as stored flat 1D array)
col	vector to receive the copy
whichcol	column index

Definition at line 347 of file vm_m3math.h.

5.6.2.2 template<class T_fp > void vm_M3Math< T_fp >::extract_row (T_fp const *m[]*, T_fp *row[]*, int *whichrow*) [inline], [static]

Copy a given row of m to a vector.

Parameters

m	matrix (as stored flat 1D array)
row	vector to receive the copy
whichrow	row index

Definition at line 341 of file vm_m3math.h.

5.6.2.3 template<class T_fp > void vm_M3Math< T_fp >::inject_col (T_fp *m[]*, T_fp const *col[]*, int *whichcol*) [inline], [static]

Copy a vector to a given column of m.

Parameters

m	matrix (as stored flat 1D array)
col	vector to be stored
whichcol	column index

Definition at line 327 of file vm_m3math.h.

5.6.2.4 template< class T_fp > void vm_M3Math< T_fp >::inject_row (T_fp *m[ ]*, T_fp const *row[ ]*, int *whichrow*) [inline],[static]

Copy a vector to a given row of m.

Parameters

m	matrix (as stored flat 1D array)
row	vector to be stored
whichrow	row index

Definition at line 321 of file vm_m3math.h.

5.7 Matrix Vector operations.

Functions

- static void vm_M3Math< T_fp >::mvmult (T_fp res[ ], T_fp const m[ ], T_fp const v[ ])
- static void vm_M3Math< T_fp >::mtvmult (T_fp res[ ], T_fp const m[ ], T_fp const v[ ])

5.7.1 Detailed Description

5.7.2 Function Documentation

5.7.2.1 template<class T_fp > void vm_M3Math< T_fp >::mtvmult (T_fp *res[ ]*, T_fp const *m[ ]*, T_fp const *v[ ]*) [inline], [static]

Matrix multiplication of vector v by transpose of matrix m.

result = m_transpose *matrix_multiply* v.

Parameters

res	resulting matrix (as stored flat 1D array)
m	matrix to be transposed and multiplied
v	vector to be multiplied

Definition at line 382 of file vm_m3math.h.

5.7.2.2 template<class T_fp > void vm_M3Math< T_fp >::mvmult (T_fp *res[ ]*, T_fp const *m[ ]*, T_fp const *v[ ]*) [inline], [static]

Matrix multiplication of vector v by matrix m.

result = m *matrix_multiply* v.

Parameters

res	resulting matrix (as stored flat 1D array)
m	matrix to be multiplied
v	vector to be multiplied

Definition at line 373 of file vm_m3math.h.

5.8 Matrix Matrix operations.

Functions

- static void **vm_M3Math< T_fp >::mmult** (T_fp mres[ ], T_fp const m1[ ], T_fp const m2[ ])

5.8.1 Detailed Description

5.9 IO operations.

Functions

- static std::ostream & vm_M3Math< T_fp >::print_on (std::ostream &os, T_fp const m[], char const prefix[]="", char const postfix[]="")
- static void vm_M3Math< T_fp >::cprint_on (FILE *of, T_fp const m[], char const prefix[]="", char const postfix[]="")

5.9.1 Detailed Description

5.9.2 Function Documentation

5.9.2.1 template<class T_fp > void **vm_M3Math**< T_fp >::cprint_on (FILE * *of*, T_fp const *m[]*, char const *prefix[]* = "", char const *postfix[]* = "") [inline], [static]

Print a matrix to a FILE* stream.

Parameters

of	the FILE*
m	matrix to be printed
prefix	optional prefix string
postfix	optional postfix string

Definition at line 421 of file vm_m3math.h.

5.9.2.2 template<class T_fp > std::ostream & **vm_M3Math**< T_fp >::print_on (std::ostream & *os*, T_fp const *m[]*, char const *prefix[]* = "", char const *postfix[]* = "") [inline], [static]

Print a matrix to an ostream.

Parameters

os	the ostream
m	matrix to be printed
prefix	optional prefix string
postfix	optional postfix string

Definition at line 408 of file vm_m3math.h.

5.10 Set vector components

Functions

- static void vm_V3Math< T_fp >::set (T_fp v[ ], T_fp x, T_fp y, T_fp z)
- static void vm_V3Math< T_fp >::set (T_fp v[ ], T_fp x)

5.10.1 Detailed Description

5.10.2 Function Documentation

5.10.2.1 template<class T_fp > void vm_V3Math< T_fp >::set (T_fp *v[ ]*, T_fp *x*, T_fp *y*, T_fp *z*) [inline], [static]

Set components of v to x, y, z.

Parameters

v	vector to be set
x	x component
y	y component
z	z component

Definition at line 255 of file vm_v3math.h.

5.10.2.2 template<class T_fp > void vm_V3Math< T_fp >::set (T_fp *v[ ]*, T_fp *x*) [inline], [static]

Set all components of v to x.

Parameters

v	vector to be set
x	value

Definition at line 260 of file vm_v3math.h.

References vm_VMath< T_fp, N_len >::set().

5.11 Normalize vectors

Functions

- static T_fp vm_V3Math< T_fp >::unitize (T_fp v[])
- static T_fp vm_V3Math< T_fp >::unitize (T_fp vu[], T_fp const vi[])

5.11.1 Detailed Description

5.11.2 Function Documentation

5.11.2.1 template<class T_fp > T_fp vm_V3Math< T_fp >::unitize (T_fp *v[]*) [inline], [static]

Normalize a vector v.

Returns

return original vector magnitude

Parameters

v	vector to be normalized

Definition at line 282 of file vm_v3math.h.

Referenced by vm_V3Math< T_fp >::unitize().

5.11.2.2 template<class T_fp > T_fp vm_V3Math< T_fp >::unitize (T_fp *vu[]*, T_fp const *vi[]*) [inline], [static]

Normalize a vector vi returning the normalized value in vu.

Returns

return original magnitude of vi

Parameters

vi	vector to be normalized
vu	normalized version of vi

Definition at line 296 of file vm_v3math.h.

References vm_VMath< T_fp, 3 >::copy(), and vm_V3Math< T_fp >::unitize().

5.12 Tests for normality, orthogonality

Functions

- static int vm_V3Math< T_fp >::is_unit_vector (T_fp const v[ ], T_fp const tol)
- static int vm_V3Math< T_fp >::are_orthogonal (T_fp const v[ ], T_fp const other[ ], T_fp const tol)
- static int vm_V3Math< T_fp >::are_orthonormal (T_fp const v[ ], T_fp const other[ ], T_fp const tol)

5.12.1 Detailed Description

5.12.2 Function Documentation

5.12.2.1 template<class T_fp > int **vm_V3Math**< T_fp >::are_orthogonal (T_fp const *v[ ]*, T_fp const *other[ ]*, T_fp const *tol*) [inline], [static]

Test whether a v is within tolerance tol of orthogonality to other vector

Returns

1 if v and other are orthogonal within tolerance tol, 0 otherwise.

Parameters

v	vector to be checked
other	other vector
tol	tolerance

Definition at line 311 of file vm_v3math.h.

5.12.2.2 template<class T_fp > int **vm_V3Math**< T_fp >::are_orthonormal (T_fp const *v[ ]*, T_fp const *other[ ]*, T_fp const *tol*) [inline], [static]

Test whether v is within tolerance tol of orthonormality with other vector

Returns

1 if v and other are orthonormal within tolerance tol, 0 otherwise.

Parameters

v	vector to be checked
other	other vector
tol	tolerance

Definition at line 320 of file vm_v3math.h.

5.12.2.3 template<class T_fp > int **vm_V3Math**< T_fp >::is_unit_vector (T_fp const *v[ ]*, T_fp const *tol*) [inline], [static]

Test whether a vector is within tolerance tol of a unit vector

Returns

1 if v is within tolerance tol of being a unit vector, 0 otherwise.

Parameters

v	vector to be checked
tol	tolerance

Definition at line 304 of file vm_v3math.h.

5.13 Dot and Cross products.

Functions

- static T_fp vm_V3Math< T_fp >::dot (T_fp const v1[], T_fp const v2[])
- static void vm_V3Math< T_fp >::cross (T_fp prod[], T_fp const v1[], T_fp const v2[])

5.13.1 Detailed Description

5.13.2 Function Documentation

5.13.2.1 template<class T_fp > void vm_V3Math< T_fp >::cross (T_fp *prod[ ]*, T_fp const *v1[ ]*, T_fp const *v2[ ]*) [inline], [static]

Cross (wedge) product of two vectors

Returns

cross (wedge) product of v1 with v2.

Parameters

prod	cross (wedge) product of v1 with v2.
v1	1st vector
v2	2nd vector

Definition at line 272 of file vm_v3math.h.

5.13.2.2 template<class T_fp > T_fp vm_V3Math< T_fp >::dot (T_fp const *v1[ ]*, T_fp const *v2[ ]*) [inline], [static]

Dot product of two vectors

Returns

dot (scalar) product of v1 with v2.

Parameters

v1	1st vector
v2	2nd vector

Definition at line 265 of file vm_v3math.h.

5.14 IO operations.

Functions

- static std::ostream & vm_V3Math< T_fp >::print_on (std::ostream &os, T_fp const v[ ], char const prefix[ ]="", char const postfix[ ]="")
- static void vm_V3Math< T_fp >::cprint_on (FILE ∗of, T_fp const v[ ], char const prefix[ ]="", char const postfix[ ]="")

5.14.1 Detailed Description

5.14.2 Function Documentation

5.14.2.1 template<class T_fp > void vm_V3Math< T_fp >::cprint_on (FILE ∗ *of*, T_fp const *v[ ]*, char const *prefix[ ] =* " ", char const *postfix[ ] =* " ") [inline], [static]

Print a vector to a FILE∗ stream.

Parameters

of	the FILE∗
v	vector to be printed
prefix	optional prefix string
postfix	optional postfix string

Definition at line 340 of file vm_v3math.h.

5.14.2.2 template<class T_fp > std::ostream & vm_V3Math< T_fp >::print_on (std::ostream & *os*, T_fp const *v[ ]*, char const *prefix[ ] =* " ", char const *postfix[ ] =* " ") [inline], [static]

Print a vector to an ostream.

Parameters

os	the ostream
v	vector to be printed
prefix	optional prefix string
postfix	optional postfix string

Definition at line 331 of file vm_v3math.h.

5.15 Copy routines

Functions

- static void vm_VMath< T_fp, N_len >::copy (T_fp v[ ], T_fp const cv[ ])

5.15.1 Detailed Description

5.15.2 Function Documentation

5.15.2.1 template<class T_fp, int N_len> void vm_VMath< T_fp, N_len >::copy (T_fp *v[ ],* T_fp const *cv[ ]*) [inline], [static]

Normalize a vector v.

Parameters

v	destination vector
cv	source vector

REQUIREMENT: ∗v and ∗cv each has a length of at least N_len contiguous T_fps and is appropriately aligned for T_fps.

Definition at line 385 of file vm_vmath.h.

5.16 Set functions

Functions

- static void vm_VMath< T_fp, N_len >::set (T_fp v[ ], T_fp r)

5.16.1 Detailed Description

5.16.2 Function Documentation

5.16.2.1 template<class T_fp, int N_len> void vm_VMath< T_fp, N_len >::set (T_fp *v[ ]*, T_fp *r*) [inline], [static]

Set all elements of v to r.

Parameters

v	vector to be set
r	value

Definition at line 393 of file vm_vmath.h.

Referenced by vm_V3Math< T_fp >::set().

5.17 Componentwise op_eq operations (+=, -=,=, /=)

Functions

- static void vm_VMath< T_fp, N_len >::add_eq (T_fp v[ ], T_fp const cv[ ])
- static void vm_VMath< T_fp, N_len >::sub_eq (T_fp v[ ], T_fp const cv[ ])
- static void vm_VMath< T_fp, N_len >::mul_eq (T_fp v[ ], T_fp const cv[ ])
- static void vm_VMath< T_fp, N_len >::div_eq (T_fp v[ ], T_fp const cv[ ])
- static void vm_VMath< T_fp, N_len >::add_eq (T_fp v[ ], T_fp r)
- static void vm_VMath< T_fp, N_len >::sub_eq (T_fp v[ ], T_fp r)
- static void vm_VMath< T_fp, N_len >::mul_eq (T_fp v[ ], T_fp r)
- static void vm_VMath< T_fp, N_len >::div_eq (T_fp v[ ], T_fp r)
- static void vm_VMath< T_fp, N_len >::negate (T_fp v[ ])

5.17.1 Detailed Description

5.17.2 Function Documentation

5.17.2.1 template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::add_eq (T_fp *v[ ],* T_fp const *cv[ ]*) [inline], [static]

component-wise v[n] += cv[n]

Parameters

v	LHS vector
cv	RHS vector

5.17.2.2 template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::add_eq (T_fp *v[ ],* T_fp *r*) [inline], [static]

component-wise v[n] += r

Parameters

v	LHS vector
r	RHS T_fp

5.17.2.3 template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::div_eq (T_fp *v[ ],* T_fp const *cv[ ]*) [inline], [static]

component-wise v[n] /= cv[n]

Parameters

v	LHS vector
cv	RHS vector

5.17.2.4 template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::div_eq (T_fp *v[ ],* T_fp *r*) [inline], [static]

component-wise v[n] /= r

Parameters

v	LHS vector
r	RHS T_fp

5.17.2.5 template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::mul_eq (T_fp *v[ ]*, T_fp const *cv[ ]*) [inline], [static]

component-wise v[n] ∗= cv[n]

Parameters

v	LHS vector
cv	RHS vector

5.17.2.6 template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::mul_eq (T_fp *v[ ]*, T_fp *r*) [inline], [static]

component-wise v[n] ∗= r

Parameters

v	LHS vector
r	RHS T_fp

5.17.2.7 template<class T_fp, int N_len> int N_len void vm_VMath< T_fp, N_len >::negate (T_fp *v[ ]*) [inline], [static]

component-wise negation of v

Parameters

v	vector

Definition at line 414 of file vm_vmath.h.

5.17.2.8 template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::sub_eq (T_fp *v[ ]*, T_fp const *cv[ ]*) [inline], [static]

component-wise v[n] -= cv[n]

Parameters

v	LHS vector
cv	RHS vector

5.17.2.9 template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::sub_eq (T_fp *v[ ]*, T_fp *r*) [inline], [static]

component-wise v[n] -= r

Parameters

v	LHS vector
r	RHS T_fp

5.18 {add,sub,mul,div}: componentwise binary operations.

Functions

- static void vm_VMath< T_fp, N_len >::add (T_fp v[], T_fp const cv1[], T_fp const cv2[])
- static void vm_VMath< T_fp, N_len >::sub (T_fp v[], T_fp const cv1[], T_fp const cv2[])
- static void vm_VMath< T_fp, N_len >::mul (T_fp v[], T_fp const cv1[], T_fp const cv2[])
- static void vm_VMath< T_fp, N_len >::div (T_fp v[], T_fp const cv1[], T_fp const cv2[])
- static void vm_VMath< T_fp, N_len >::add (T_fp v[], T_fp const cv[], T_fp r)
- static void vm_VMath< T_fp, N_len >::sub (T_fp v[], T_fp const cv[], T_fp r)
- static void vm_VMath< T_fp, N_len >::mul (T_fp v[], T_fp const cv[], T_fp r)
- static void vm_VMath< T_fp, N_len >::div (T_fp v[], T_fp const cv[], T_fp r)
- static void vm_VMath< T_fp, N_len >::add (T_fp v[], T_fp r, T_fp const cv[])
- static void vm_VMath< T_fp, N_len >::sub (T_fp v[], T_fp r, T_fp const cv[])
- static void vm_VMath< T_fp, N_len >::mul (T_fp v[], T_fp r, T_fp const cv[])
- static void vm_VMath< T_fp, N_len >::div (T_fp v[], T_fp r, T_fp const cv[])

5.18.1 Detailed Description

5.18.2 Function Documentation

5.18.2.1 template< class T_fp, int N_len > static void **vm_VMath**< T_fp, N_len >::add (T_fp *v[]*, T_fp const *cv1[]*, T_fp const *cv2[]*) [inline], [static]

component-wise v[n] = cv1[n] + cv2[n]

Parameters

v	result vector
cv1	1st input vector
cv2	2nd input vector

5.18.2.2 template< class T_fp, int N_len > static void **vm_VMath**< T_fp, N_len >::add (T_fp *v[]*, T_fp const *cv[]*, T_fp *r*) [inline], [static]

component-wise v[n] = cv[n] + r

Parameters

v	result vector
cv	input vector
r	input T_fp

5.18.2.3 template< class T_fp, int N_len > static void **vm_VMath**< T_fp, N_len >::add (T_fp *v[]*, T_fp *r*, T_fp const *cv[]*) [inline], [static]

component-wise v[n] = r + cv[n]

Parameters

v	result vector
r	input T_fp
cv	input vector

5.18.2.4 template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::div (T_fp *v[ ]*, T_fp const *cv1[ ]*, T_fp const *cv2[ ]*) [inline], [static]

component-wise v[n] = cv1[n] / cv2[n]

Parameters

v	result vector
cv1	1st input vector
cv2	2nd input vector

5.18.2.5 template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::div (T_fp *v[ ]*, T_fp const *cv[ ]*, T_fp *r*) [inline], [static]

component-wise v[n] = cv[n] / r

Parameters

v	result vector
cv	input vector
r	input T_fp

5.18.2.6 template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::div (T_fp *v[ ]*, T_fp *r*, T_fp const *cv[ ]*) [inline], [static]

component-wise v[n] = r / cv[n]

Parameters

v	result vector
r	input T_fp
cv	input vector

5.18.2.7 template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::mul (T_fp *v[ ]*, T_fp const *cv1[ ]*, T_fp const *cv2[ ]*) [inline], [static]

component-wise v[n] = cv1[n] ∗ cv2[n]

Parameters

v	result vector
cv1	1st input vector
cv2	2nd input vector

5.18.2.8 template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::mul (T_fp *v[ ]*, T_fp const *cv[ ]*, T_fp *r*) [inline], [static]

component-wise v[n] = cv[n] ∗ r

Parameters

v	result vector
cv	input vector
r	input T_fp

5.18.2.9 template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::mul (T_fp *v[]*, T_fp *r*, T_fp const *cv[]*) [inline], [static]

component-wise v[n] = r * cv[n]

Parameters

v	result vector
r	input T_fp
cv	input vector

5.18.2.10 template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::sub (T_fp *v[]*, T_fp const *cv1[]*, T_fp const *cv2[]*) [inline], [static]

component-wise v[n] = cv1[n] - cv2[n]

Parameters

v	result vector
cv1	1st input vector
cv2	2nd input vector

5.18.2.11 template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::sub (T_fp *v[]*, T_fp const *cv[]*, T_fp *r*) [inline], [static]

component-wise v[n] = cv[n] - r

Parameters

v	result vector
cv	input vector
r	input T_fp

5.18.2.12 template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::sub (T_fp *v[]*, T_fp *r*, T_fp const *cv[]*) [inline], [static]

component-wise v[n] = r - cv[n]

Parameters

v	result vector
r	input T_fp
cv	input vector

5.19 Linear combinations

Functions

- static void vm_VMath< T_fp, N_len >::lincomb (T_fp res[ ], T_fp c1, T_fp const v1[ ], T_fp c2, T_fp const v2[ ])

5.19.1 Detailed Description

5.19.2 Function Documentation

5.19.2.1 template<class T_fp, int N_len> void vm_VMath< T_fp, N_len >::lincomb (T_fp *res[ ]*, T_fp *c1*, T_fp const *v1[ ]*, T_fp *c2*, T_fp const *v2[ ]*) [inline], [static]

form linear combination: res = c1 ∗ v1 + c2 ∗ v2.

```
For each i:  res[i] = c1 * v1[i] + c2 * v2[i]
```

Parameters

res	result vector
c1	1st T_fp
v1	1st vector
c2	2nd T_fp
v2	2nd vector

Definition at line 441 of file vm_vmath.h.

5.20 I/O operations.

Functions

- static std::ostream & vm_VMath< T_fp, N_len >::print_on (std::ostream &os, T_fp const v[], int by, char const prefix[]="", char const postfix[]="")
- static void vm_VMath< T_fp, N_len >::cprint_on (FILE *of, T_fp const v[], int by, char const prefix[]="", char const postfix[]="")

5.20.1 Detailed Description

5.20.2 Function Documentation

5.20.2.1 template<class T_fp, int N_len> void vm_VMath< T_fp, N_len >::cprint_on (FILE * *of*, T_fp const *v[]*, int *by*, char const *prefix[]* = " ", char const *postfix[]* = " ") [inline], [static]

Print a vector to a FILE* stream.

Parameters

of	the FILE*
v	vector to be printed
by	stride by this many
prefix	optional prefix string
postfix	optional postfix string

Definition at line 471 of file vm_vmath.h.

5.20.2.2 template<class T_fp, int N_len> std::ostream & vm_VMath< T_fp, N_len >::print_on (std::ostream & *os*, T_fp const *v[]*, int *by*, char const *prefix[]* = " ", char const *postfix[]* = " ") [inline], [static]

Print a vector to an ostream.

Parameters

os	the ostream
v	vector to be printed
by	stride by this many
prefix	optional prefix string
postfix	optional postfix string

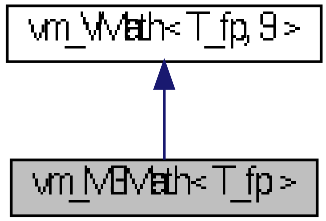
Definition at line 451 of file vm_vmath.h.

6 Class Documentation

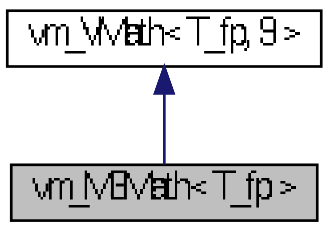
6.1 vm_M3Math< T_fp > Class Template Reference

`#include <vm_math/vm_m3math.h>`

Inheritance diagram for vm_M3Math< T_fp >:



Collaboration diagram for vm_M3Math< T_fp >:



Public Types

- typedef T_fp **value_type**

Static Public Member Functions

- static int at (int i, int j)
- static void init_by_row (T_fp m[], T_fp const row0[], T_fp const row1[], T_fp const row2[])
- static void init_by_col (T_fp m[], T_fp const col0[], T_fp const col1[], T_fp const col2[])
- static void dyad_product (T_fp m[], T_fp const v1[], T_fp const v2[])
- static void inject_row (T_fp m[], T_fp const row[], int whichrow)

- static void inject_col (T_fp m[ ], T_fp const col[ ], int whichcol)
- static void extract_row (T_fp const m[ ], T_fp row[ ], int whichrow)
- static void extract_col (T_fp const m[ ], T_fp col[ ], int whichcol)
- static void mvmult (T_fp res[ ], T_fp const m[ ], T_fp const v[ ])
- static void mtvmult (T_fp res[ ], T_fp const m[ ], T_fp const v[ ])
- static void **mmult** (T_fp mres[ ], T_fp const m1[ ], T_fp const m2[ ])
- static std::ostream & print_on (std::ostream &os, T_fp const m[ ], char const prefix[ ]="", char const postfix[ ]="")
- static void cprint_on (FILE *of, T_fp const m[ ], char const prefix[ ]="", char const postfix[ ]="")

6.1.1 Detailed Description

template<class T_fp>class vm_M3Math< T_fp >

A template class providing common numerical operations on 3x3-matrices of T_fp's (floating point type). The matrix is assumed to be stored as a * contiguous one-dimensional array of 9 T_fp's, properly aligned for type T_fp.

The class is a simple class to handle common numerical operations on 3x3-matrices

of floating point T_fps.

Unless otherwise noted, the operations are component by component, e.g.,

```
mul_eq(m1,m2)
```

results in

```
m1[i][j] += m2[i][j], where i = 0,1,2 and j = 0,1,2.
```

vm_M3Math has only static member functions; there are no data members.

Where possible, the static member functions are inlined.

Definition at line 77 of file vm_m3math.h.

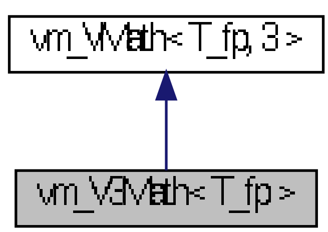
The documentation for this class was generated from the following file:

- vm_m3math.h

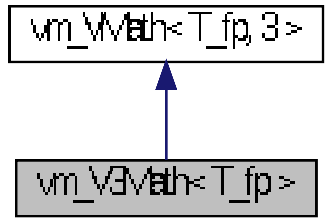
6.2 vm_V3Math< T_fp > Class Template Reference

```
#include <vm_math/vm_v3math.h>
```

Inheritance diagram for vm_V3Math< T_fp >:



Collaboration diagram for vm_V3Math< T_fp >:



Public Types

- typedef T_fp value_type

Static Public Member Functions

- static void set (T_fp v[ ], T_fp x, T_fp y, T_fp z)
- static void set (T_fp v[ ], T_fp x)
- static T_fp unitize (T_fp v[ ])
- static T_fp unitize (T_fp vu[ ], T_fp const vi[ ])
- static int is_unit_vector (T_fp const v[ ], T_fp const tol)
- static int are_orthogonal (T_fp const v[ ], T_fp const other[ ], T_fp const tol)
- static int are_orthonormal (T_fp const v[ ], T_fp const other[ ], T_fp const tol)
- static T_fp dot (T_fp const v1[ ], T_fp const v2[ ])
- static void cross (T_fp prod[ ], T_fp const v1[ ], T_fp const v2[ ])
- static std::ostream & print_on (std::ostream &os, T_fp const v[ ], char const prefix[ ]="", char const postfix[ ]="")
- static void cprint_on (FILE ∗of, T_fp const v[ ], char const prefix[ ]="", char const postfix[ ]="")

6.2.1 Detailed Description

template<class T_fp>class vm_V3Math< T_fp >

A template class providing common numerical operations on 3-vectors of T_fp's (floating point type).

Unless otherwise noted, the operations are component by component, e.g.,

```
vm_V3Mathlt<float>::div_eq(v1, v2)
```

corresponds to

```
v1[i] /= v2[i], where i = 0,1,2.
```

vm_V3Math has only static member functions; there are no data members.

Where possible, the static member functions are inlined.

Definition at line 69 of file vm_v3math.h.

6.2.2 Member Typedef Documentation

6.2.2.1 template<class T_fp > vm_V3Math< T_fp >::value_type

A typedef for the floating point type;

Definition at line 83 of file vm_v3math.h.

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

- vm_v3math.h

6.3 vm_VMath< T_fp, N_len > Class Template Reference

```
#include <vm_math/vm_vmath.h>
```

Public Types

- typedef T_fp value_type

Static Public Member Functions

- static void copy (T_fp v[ ], T_fp const cv[ ])
- static void set (T_fp v[ ], T_fp r)
- static void add_eq (T_fp v[ ], T_fp const cv[ ])
- static void sub_eq (T_fp v[ ], T_fp const cv[ ])
- static void mul_eq (T_fp v[ ], T_fp const cv[ ])
- static void div_eq (T_fp v[ ], T_fp const cv[ ])
- static void add_eq (T_fp v[ ], T_fp r)
- static void sub_eq (T_fp v[ ], T_fp r)
- static void mul_eq (T_fp v[ ], T_fp r)
- static void div_eq (T_fp v[ ], T_fp r)
- static void negate (T_fp v[ ])
- static void add (T_fp v[ ], T_fp const cv1[ ], T_fp const cv2[ ])
- static void sub (T_fp v[ ], T_fp const cv1[ ], T_fp const cv2[ ])
- static void mul (T_fp v[ ], T_fp const cv1[ ], T_fp const cv2[ ])
- static void div (T_fp v[ ], T_fp const cv1[ ], T_fp const cv2[ ])
- static void add (T_fp v[ ], T_fp const cv[ ], T_fp r)
- static void sub (T_fp v[ ], T_fp const cv[ ], T_fp r)
- static void mul (T_fp v[ ], T_fp const cv[ ], T_fp r)
- static void div (T_fp v[ ], T_fp const cv[ ], T_fp r)
- static void add (T_fp v[ ], T_fp r, T_fp const cv[ ])
- static void sub (T_fp v[ ], T_fp r, T_fp const cv[ ])
- static void mul (T_fp v[ ], T_fp r, T_fp const cv[ ])
- static void div (T_fp v[ ], T_fp r, T_fp const cv[ ])
- static void lincomb (T_fp res[ ], T_fp c1, T_fp const v1[ ], T_fp c2, T_fp const v2[ ])
- static std::ostream & print_on (std::ostream &os, T_fp const v[ ], int by, char const prefix[ ]="", char const postfix[ ]="")
- static void cprint_on (FILE *of, T_fp const v[ ], int by, char const prefix[ ]="", char const postfix[ ]="")

6.3.1 Detailed Description

template<class T_fp, int N_len>class vm_VMath< T_fp, N_len >

A template class providing common numerical operations on N_len-long 1 dimensional arrays of T_fp .

T_fp is a floating point type.

N_len is the length of the vector.

The array data are assumed to be stored as a contiguous one-dimensional array of N_len T_fp's, properly aligned for type T_fp.

Unless otherwise noted, the operations are component by component, e.g.,

```
vm_VMath<float,4>::mul(prod, v1, v2)
```

corresponds to

```
prod[i] = v1[i] * v2[i], where i = 0,1,2,3.
```

vm_VMath has only static member functions; there are no data members.

Where possible, the static member functions are inlined.

Definition at line 78 of file vm_vmath.h.

6.3.2 Member Typedef Documentation

6.3.2.1 template<class T_fp, int N_len> vm_VMath< T_fp, N_len >::value_type

a typedef for the floating point type;

Definition at line 91 of file vm_vmath.h.

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

- vm_vmath.h

Index

{add,sub,mul,div}: componentwise binary operations., 24
    add, 24
    div, 25
    mul, 25, 26
    sub, 26

add
    {add,sub,mul,div}: componentwise binary operations., 24
add_eq
    Componentwise op_eq operations (+=, -=,=, /=), 22
are_orthogonal
    Tests for normality, orthogonality, 16
are_orthonormal
    Tests for normality, orthogonality, 16
at
    Index calculations, 7

Componentwise op_eq operations (+=, -=,=, /=), 22
    add_eq, 22
    div_eq, 22
    mul_eq, 23
    negate, 23
    sub_eq, 23
copy
    Copy routines, 20
Copy routines, 20
    copy, 20
cprint_on
    I/O operations., 28
    IO operations., 13, 19
cross
    Dot and Cross products., 18

div
    {add,sub,mul,div}: componentwise binary operations., 25
div_eq
    Componentwise op_eq operations (+=, -=,=, /=), 22
dot
    Dot and Cross products., 18
Dot and Cross products., 18
    cross, 18
    dot, 18
dyad_product
    Initialize a matrix., 8

extract_col
    For a matrix, insert or extract a vector., 9
extract_row
    For a matrix, insert or extract a vector., 9

For a matrix, insert or extract a vector., 9
    extract_col, 9
    extract_row, 9
    inject_col, 9
    inject_row, 9

I/O operations., 28
    cprint_on, 28
    print_on, 28
IO operations., 13, 19
    cprint_on, 13, 19
    print_on, 13, 19
Index calculations, 7
    at, 7
init_by_col
    Initialize a matrix., 8
init_by_row
    Initialize a matrix., 8
Initialize a matrix., 8
    dyad_product, 8
    init_by_col, 8
    init_by_row, 8
inject_col
    For a matrix, insert or extract a vector., 9
inject_row
    For a matrix, insert or extract a vector., 9
is_unit_vector
    Tests for normality, orthogonality, 16

lincomb
    Linear combinations, 27
Linear combinations, 27
    lincomb, 27

Matrix Matrix operations., 12
Matrix Vector operations., 11
    mtvmult, 11
    mvmult, 11
mtvmult
    Matrix Vector operations., 11
mul
    {add,sub,mul,div}: componentwise binary operations., 25, 26
mul_eq
    Componentwise op_eq operations (+=, -=,=, /=), 23
mvmult
    Matrix Vector operations., 11

negate
    Componentwise op_eq operations (+=, -=,=, /=), 23
Normalize vectors, 15
    unitize, 15

- print_on
  - I/O operations., 28
  - IO operations., 13, 19

- set
  - Set functions, 21
  - Set vector components, 14
- Set functions, 21
  - set, 21
- Set vector components, 14
  - set, 14
- sub
  - {add,sub,mul,div}: componentwise binary operations., 26
- sub_eq
  - Componentwise op_eq operations (+=, -=,=, /=), 23

- Tests for normality, orthogonality, 16
  - are_orthogonal, 16
  - are_orthonormal, 16
  - is_unit_vector, 16

- unitize
  - Normalize vectors, 15

- value_type
  - vm_V3Math, 32
  - vm_VMath, 33
- vm_M3Math common numerical operations, 6
- vm_M3Math< T_fp >, 29
- vm_V3Math
  - value_type, 32
- vm_V3Math common numerical operations, 5
- vm_V3Math< T_fp >, 30
- vm_VMath
  - value_type, 33
- vm_VMath common numerical operations, 4
- vm_VMath< T_fp, N_len >, 32