vm_math Reference Manual

1.0.5

Generated by Doxygen 1.4.2

Wed Apr 19 17:38:26 2006

Contents

1 The vm_Math C++ template Library 1

2 vm_math Module Index 3

3 vm_math Directory Hierarchy 4

4 vm_math Hierarchical Index 4

5 vm_math Class Index 5

6 vm_math Module Documentation 5

7 vm_math Directory Documentation 24

8 vm_math Class Documentation 24

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 vm_math Module Index

2.1 vm_math Modules

Here is a list of all modules:

vm_VMath common numerical operations 5

vm_V3Math common numerical operations 5

vm_M3Math common numerical operations 5

Index calculations 5

Initialize a matrix. 6

For a matrix, insert or extract a vector. 7

Matrix Vector operations. 8

Matrix Matrix operations. 9

IO operations. 9

Set vector components 10

Normalize vectors 11

Tests for normality, orthogonality 12

Dot and Cross products. 13

IO operations. 14

Copy routines 15

Set functions 16

Componentwise op_eq operations (+=, -=,=, /=) 16

{add,sub,mul,div}: componentwise binary operations. 19

Linear combinations 22

I/O operations. 23

3 vm_math Directory Hierarchy

3.1 vm_math Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

vm_math 24

4 vm_math Hierarchical Index

4.1 vm_math Class Hierarchy

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

vm_VMath< T_fp, N_len > 28

vm_VMath< T_fp, 3 > 28

  vm_V3Math< T_fp > 26

vm_VMath< T_fp, 9 > 28

  vm_M3Math< T_fp > 24

5 vm_math Class Index

5.1 vm_math Class List

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

vm_M3Math< T_fp > 24

vm_V3Math< T_fp > 26

vm_VMath< T_fp, N_len > 28

6 vm_math Module Documentation

6.1 vm_VMath common numerical operations

6.2 vm_V3Math common numerical operations

6.3 vm_M3Math common numerical operations

6.4 Index calculations

Functions

- ∗static int vm_M3Math::at (int i, int j)

6.4.1 Function Documentation

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

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.

6.5 Initialize a matrix.

Functions

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

6.5.1 Function Documentation

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

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.

6.5.1.2 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, inherited]

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.

6.5.1.3 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, inherited]

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.

6.6 For a matrix, insert or extract a vector.

Functions

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

6.6.1 Function Documentation

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

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.

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

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.

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

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.

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

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.

6.7 Matrix Vector operations.

Functions

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

6.7.1 Function Documentation

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

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.

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

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.

6.8 Matrix Matrix operations.

Functions

- ∗static void **mmult** (T_fp mres[ ], T_fp const m1[ ], T_fp const m2[ ])

6.9 IO operations.

Functions

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

6.9.1 Function Documentation

6.9.1.1 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, inherited]`

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.

6.9.1.2 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, inherited]`

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.

6.10 Set vector components

Functions

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

6.10.1 Function Documentation

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

Set all components of v to x.

Parameters:

v vector to be set

x value

Reimplemented from vm_VMath< T_fp, 3 >.

Definition at line 260 of file vm_v3math.h.

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

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

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.

6.11 Normalize vectors

Functions

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

6.11.1 Function Documentation

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

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().

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

Normalize a vector v.

Returns:
return original vector magnitude

Parameters:
v vector to be normalized

Definition at line 282 of file vm_v3math.h.

References vm_VMath< T_fp, 3 >::div_eq(), and vm_V3Math< T_fp >::dot().

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

6.12 Tests for normality, orthogonality

Functions

- ∗static int vm_V3Math::is_unit_vector (T_fp const v[ ], T_fp const tol)
- static int vm_V3Math::are_orthogonal (T_fp const v[ ], T_fp const other[ ], T_fp const tol)
- static int vm_V3Math::are_orthonormal (T_fp const v[ ], T_fp const other[ ], T_-fp const tol)

6.12.1 Function Documentation

6.12.1.1 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, inherited]

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.

References vm_V3Math< T_fp >::dot().

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

6.12.1.2 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, inherited]

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.

References vm_V3Math< T_fp >::are_orthogonal(), and vm_V3Math< T_fp >::is_-unit_vector().

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

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.

References vm_V3Math< T_fp >::dot().

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

6.13 Dot and Cross products.

Functions

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

6.13.1 Function Documentation

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

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.

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

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.

Referenced by vm_V3Math< T_fp >::are_orthogonal(), vm_V3Math< T_fp >::is_unit_vector(), and vm_V3Math< T_fp >::unitize().

6.14 IO operations.

Functions

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

6.14.1 Function Documentation

6.14.1.1 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, inherited]

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.

6.14.1.2 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, inherited]

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.

6.15 Copy routines

Functions

- ∗static void vm_VMath::copy (T_fp v[ ], T_fp const cv[ ])

6.15.1 Function Documentation

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

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.

6.16 Set functions

Functions

- ∗static void vm_VMath::set (T_fp v[ ], T_fp r)

6.16.1 Function Documentation

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

Set all elements of v to r.

Parameters:

v vector to be set

r value

Reimplemented in vm_V3Math< T_fp >.

Definition at line 393 of file vm_vmath.h.

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

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

Functions

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

6.17.1 Function Documentation

6.17.1.1 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, inherited]

component-wise v[n] += r

Parameters:

v LHS vector

r RHS T_fp

6.17.1.2 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, inherited]

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

Parameters:

v LHS vector

cv RHS vector

6.17.1.3 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, inherited]

component-wise v[n] /= r

Parameters:

v LHS vector

r RHS T_fp

6.17.1.4 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, inherited]

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

Parameters:

v LHS vector

cv RHS vector

6.17.1.5 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, inherited]

component-wise v[n] *= r

Parameters:

v LHS vector

r RHS T_fp

6.17.1.6 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, inherited]

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

Parameters:

v LHS vector

cv RHS vector

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

component-wise negation of v

Parameters:

v vector

Definition at line 414 of file vm_vmath.h.

6.17.1.8 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, inherited]

component-wise v[n] -= r

Parameters:

v LHS vector

r RHS T_fp

6.17.1.9 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, inherited]

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

Parameters:

v LHS vector

cv RHS vector

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

Functions

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

6.18.1 Function Documentation

6.18.1.1 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, inherited]

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

Parameters:

v result vector

r input T_fp

cv input vector

6.18.1.2 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, inherited]

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

Parameters:

v result vector

cv input vector

r input T_fp

6.18.1.3 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, inherited]

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

Parameters:

v result vector

cv1 1st input vector

cv2 2nd input vector

6.18.1.4 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, inherited]

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

Parameters:

v result vector

r input T_fp

cv input vector

6.18.1.5 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, inherited]

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

Parameters:

v result vector

cv input vector

r input T_fp

6.18.1.6 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, inherited]

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

Parameters:

v result vector

cv1 1st input vector

cv2 2nd input vector

6.18.1.7 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, inherited]

component-wise v[n] = r ∗ cv[n]

Parameters:

v result vector

r input T_fp

cv input vector

6.18.1.8 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, inherited]

component-wise v[n] = cv[n] ∗ r

Parameters:

v result vector

cv input vector

r input T_fp

6.18.1.9 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, inherited]

component-wise v[n] = cv1[n] ∗ cv2[n]

Parameters:

v result vector

cv1 1st input vector

cv2 2nd input vector

6.18.1.10 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, inherited]

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

Parameters:

v result vector

r input T_fp

cv input vector

6.18.1.11 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, inherited]

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

Parameters:

v result vector

cv input vector

r input T_fp

6.18.1.12 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, inherited]

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

Parameters:

v result vector

cv1 1st input vector

cv2 2nd input vector

6.19 Linear combinations

Functions

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

6.19.1 Function Documentation

6.19.1.1 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, inherited]

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.

6.20 I/O operations.

Functions

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

6.20.1 Function Documentation

6.20.1.1 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, inherited]

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.

6.20.1.2 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, inherited]

Print a vector to an ostream.

Parameters:

os the ostream

v vector to be printed

by stride by this many

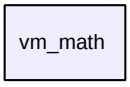
prefix optional prefix string

postfix optional postfix string

Definition at line 451 of file vm_vmath.h.

7 vm_math Directory Documentation

7.1 vm_math/ Directory Reference



Files

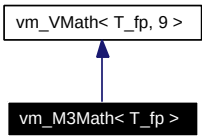
- file **vm_m3math.cc**
- file **vm_m3math.h**
- file **vm_math.h**
- file **vm_v3math.h**
- file **vm_vmath.h**

8 vm_math Class Documentation

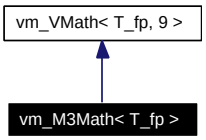
8.1 vm_M3Math< T_fp > Class Template Reference

`#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[ ]="")

8.1.1 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.

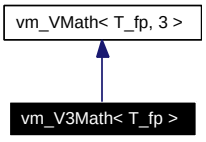
The documentation for this class was generated from the following file:

- vm_m3math.h

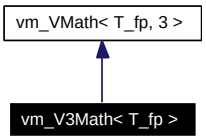
8.2 vm_V3Math< T_fp > Class Template Reference

`#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[ ]="")

8.2.1 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.

8.2.2 Member Typedef Documentation

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

A typedef for the floating point type;

Reimplemented from vm_VMath< T_fp, 3 >.

Definition at line 83 of file vm_v3math.h.

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

- vm_v3math.h

8.3 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[ ]="")

8.3.1 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.

8.3.2 Member Typedef Documentation

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

a typedef for the floating point type;

Reimplemented in vm_V3Math< T_fp >.

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., 19

add
    ndvec_mathop, 19
add_eq
    ndvec_opeq, 17
are_orthogonal
    vec_test_norm, 12
are_orthonormal
    vec_test_norm, 12
at
    mat_index_calculations, 5

Componentwise op_eq operations (+=, -=,=, /=), 16
copy
    ndvec_copy, 15
Copy routines, 15
cprint_on
    mat_io, 10
    ndvec_io, 23
    vec_io, 15
cross
    vec_dot_cross, 14

div
    ndvec_mathop, 20
div_eq
    ndvec_opeq, 17
dot
    vec_dot_cross, 14
Dot and Cross products., 13
dyad_product
    mat_init, 6

extract_col
    mat_insert_extract, 7
extract_row
    mat_insert_extract, 7

For a matrix, insert or extract a vector., 7

I/O operations., 23
Index calculations, 5
init_by_col
    mat_init, 6
init_by_row
    mat_init, 6
Initialize a matrix., 6
inject_col
    mat_insert_extract, 8
inject_row
    mat_insert_extract, 8
IO operations., 9, 14
is_unit_vector
    vec_test_norm, 13

lincomb
    ndvec_lincomb, 22
Linear combinations, 22

mat_index_calculations
    at, 5
mat_init
    dyad_product, 6
    init_by_col, 6
    init_by_row, 6
mat_insert_extract
    extract_col, 7
    extract_row, 7
    inject_col, 8
    inject_row, 8
mat_io
    cprint_on, 10
    print_on, 10
mat_vec_ops
    mtvmult, 9
    mvmult, 9
Matrix Matrix operations., 9
Matrix Vector operations., 8
mtvmult
    mat_vec_ops, 9
mul
    ndvec_mathop, 20, 21
mul_eq

ndvec_opeq, 17, 18
mvmult
mat_vec_ops, 9

ndvec_copy
copy, 15
ndvec_io
cprint_on, 23
print_on, 23
ndvec_lincomb
lincomb, 22
ndvec_mathop
add, 19
div, 20
mul, 20, 21
sub, 21, 22
ndvec_opeq
add_eq, 17
div_eq, 17
mul_eq, 17, 18
negate, 18
sub_eq, 18
ndvec_setfuncs
set, 16
negate
ndvec_opeq, 18
Normalize vectors, 11

print_on
mat_io, 10
ndvec_io, 23
vec_io, 15

set
ndvec_setfuncs, 16
vec_set_comp, 10, 11
Set functions, 16
Set vector components, 10
sub
ndvec_mathop, 21, 22
sub_eq
ndvec_opeq, 18

Tests for normality, orthogonality, 12

unitize
vec_norm, 11

value_type
vm_V3Math, 27
vm_VMath, 29
vec_dot_cross
cross, 14
dot, 14
vec_io
cprint_on, 15
print_on, 15
vec_norm
unitize, 11
vec_set_comp
set, 10, 11
vec_test_norm
are_orthogonal, 12
are_orthonormal, 12
is_unit_vector, 13
vm_M3Math, 24
vm_M3Math common numerical operations, 5
vm_math/ Directory Reference, 24
vm_V3Math, 26
value_type, 27
vm_V3Math common numerical operations, 5
vm_VMath, 28
value_type, 29
vm_VMath common numerical operations, 5