dvm3

1.0.9

Generated by Doxygen 1.5.6

Tue Dec 2 15:44:47 2008

Contents

1 The dvm3 C++ template Library

1.1 Copyright

Author: Terry Gaetz

1.2 Overview

The dvm3 package encapsulates 3-vectors and 3x3 matrices of doubles; common numerical operations are defined. Unless otherwise noted, the operations are component by component, e.g.,

```
v1 * v2 = [v1(0)*v2(0), v1(1)*v2(1), v1(2)*v2(2)]
```

The dvm3 package consists of a collection of sub-packages:

- dvm3_Vector: a 3-vector of doubles

- dvm3_Matrix: a 3x3 matrix of doubles
- dvm3_RotMat: a 3x3 rotation matrix

The dvm3_Vector and dvm3_Matrix classes are simple classes to handle common numerical operations on 3-vectors and 3-matrices of doubles. The classes provide two forms of access to components:

- indexing, e.g, vector(1), matrix(2,0); this is the most efficient access in the case of the matrix class
- projection, e.g, vector[1], matrix[2]. For vectors this is equivalent to indexing. For matrices, a pointer to the row data is returned so an additional dereferencing must be performed.

The `dvm3_RotMat` class (a derived class of dvm3_Matrix) adds some methods useful for manipulating proper rotation matrices. By proper, it is meant that the application of the rotation matrix will not change the parity of the coordinates: right-handed coordinates are transformed into right-handed coordinates.

These are intended as small components which can be inexpensively created on the stack; their constructors and destructor do not make use of dynamic allocation. The dvm3_Vector and dvm3_Matrix default constructors do not initialize the memory. The dvm3_RotMat default constructor initializes the matrix to a unit matrix (i.e., zero rotation).

Many of the operations are facilitated by the helper classes vm_VMath<T,N>, vm_V3Math<T>, and vm_M3math<T>. These classes contain a number of static member functions which operate on C-style arrays. Further documentation for these can be found under the documentation for vm_math.

Note that dvm3_Vector inherits from a struct, dvm3_VPOD which encapulates an array of 3 doubles, while dvm3_Matrix inherits from a struct dvm3_MPOD, encapulating a (1-dimensional) array of 9 doubles. These are both POD's ("Plain Ol' Data") as defined in ISO Standard C++. This ensures that dvm3_VPOD and dvm3_MPOD both have the semantics of C structs; this allows them to be copied or assigned as structs. Thus, for example, memcpy() may be called with impunity on the POD subobjects.

The inheritence from the PODS is protected; this implementation detail is exposed purely for efficiency considerations; users shall not assume that this implementation detail will remain fixed for all time. DO NOT rely on the fact that there is a POD struct under the hood or the details of the data member names within dvm3_VPOD and dvm3_MPOD.

The resultant library is available as -ldvm3; the library include file is available as <dvm3/dvm3.h>. Separate include files are provided for <dvm3/dvm3_vector.h> <dvm3/dvm3_matrix.h>, and <dvm3/dvm3_rotmat.h>.

2 Module Index

2.1 Modules

Here is a list of all modules:

3 Directory Hierarchy

3.1 Directories

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

4 Class Index

4.1 Class Hierarchy

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

5 Class Index

5.1 Class List

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

6 Module Documentation

6.1 dmv3_Vector: a 3-vector of doubles with numerical operations

6.2 dmv3_Matrix: a 3x3 matrix of doubles with numerical operations

6.3 dmv3_RotMat: a rotation matrix of doubles with numerical operations

6.4 Destructor; Constructors

Functions

- dvm3_Matrix::~dvm3_Matrix ()
- dvm3_Matrix::dvm3_Matrix ()
- dvm3_Matrix::dvm3_Matrix (dvm3_Matrix const &m)
- dvm3_Matrix::dvm3_Matrix (dvm3_Vector const &x, dvm3_Vector const &y, dvm3_Vector const &z)
- dvm3_Matrix::dvm3_Matrix (double const x[ ], double const y[ ], double const z[ ])

6.4.1 Function Documentation

6.4.1.1 dvm3_Matrix::dvm3_Matrix (double const *x*[ ], double const *y*[ ], double const *z*[ ]) `[inline, inherited]`

Construct a dvm3_Matrix; initialize by row.

Parameters:

x 1st row
y 2nd row
z 3rd row

Definition at line 924 of file dvm3_matrix.h.

6.4.1.2 dvm3_Matrix::dvm3_Matrix (dvm3_Vector const & *x*, dvm3_Vector const & *y*, dvm3_Vector const & *z*) `[inline, inherited]`

Construct a dvm3_Matrix; initialize by row.

Parameters:

x 1st row

y 2nd row

z 3rd row

Definition at line 914 of file dvm3_matrix.h.

References dvm3_Vector::data_.

6.4.1.3 dvm3_Matrix::dvm3_Matrix (dvm3_Matrix const & *m*) `[inline, inherited]`

Copy constructor.

Parameters:

m matrix to be copied

Definition at line 910 of file dvm3_matrix.h.

References dvm3_Matrix::data_.

6.4.1.4 dvm3_Matrix::dvm3_Matrix () `[inline, inherited]`

Default constructor; NO INITIALIZATION IS APPLIED.

Definition at line 906 of file dvm3_matrix.h.

6.4.1.5 dvm3_Matrix::~dvm3_Matrix () `[inline, inherited]`

Do-nothing destructor.

Definition at line 902 of file dvm3_matrix.h.

6.5 Initializers

Functions

- void dvm3_Matrix::init_by_row (dvm3_Vector const &v1, dvm3_Vector const &v2, dvm3_Vector const &v3)
- void dvm3_Matrix::init_by_row (double const v1[ ], double const v2[ ], double const v3[ ])
- void dvm3_Matrix::init_by_col (dvm3_Vector const &v1, dvm3_Vector const &v2, dvm3_Vector const &v3)
- void dvm3_Matrix::init_by_col (double const v1[ ], double const v2[ ], double const v3[ ])
- void dvm3_Vector::init (double x, double y, double z)

6.5.1 Function Documentation

6.5.1.1 void dvm3_Vector::init (double *x*, double *y*, double *z*) `[inline, inherited]`

Initialization: initialize with x, y, z.

Parameters:

- ***x*** x component
- ***y*** y component
- ***z*** z component

Definition at line 702 of file dvm3_vector.h.

6.5.1.2 void dvm3_Matrix::init_by_col (double const *v1*[ ], double const *v2*[ ], double const *v3*[ ]) `[inline, inherited]`

initialize this dvm3_Matrix by column.

Parameters:

- ***v1*** 1st column
- ***v2*** 2nd column
- ***v3*** 3rd column

Definition at line 954 of file dvm3_matrix.h.

6.5.1.3 void dvm3_Matrix::init_by_col (dvm3_Vector const & *v1*, dvm3_Vector const & *v2*, dvm3_Vector const & *v3*) `[inline, inherited]`

initialize this dvm3_Matrix by column.

Parameters:

- ***v1*** 1st column
- ***v2*** 2nd column
- ***v3*** 3rd column

Definition at line 947 of file dvm3_matrix.h.

References dvm3_Vector::data_.

6.5.1.4 void dvm3_Matrix::init_by_row (double const *v1*[ ], double const *v2*[ ], double const *v3*[ ]) `[inline, inherited]`

initialize this dvm3_Matrix by row.

Parameters:

v1 1st row

v2 2nd row

v3 3rd row

Definition at line 941 of file dvm3_matrix.h.

6.5.1.5 void dvm3_Matrix::init_by_row (dvm3_Vector const & *v1*, dvm3_Vector const & *v2*, dvm3_Vector const & *v3*) `[inline, inherited]`

initialize this dvm3_Matrix by row.

Parameters:

v1 1st row

v2 2nd row

v3 3rd row

Definition at line 934 of file dvm3_matrix.h.

References dvm3_Vector::data_.

6.6 Accessors

Functions

- double const ∗ dvm3_Matrix::operator[ ] (int i) const
- double ∗ dvm3_Matrix::operator[ ] (int i)
- double dvm3_Matrix::operator() (int i, int j) const
- double & dvm3_Matrix::operator() (int i, int j)
- double const & dvm3_Vector::operator[ ] (int i) const
- double & dvm3_Vector::operator[ ] (int i)
- double dvm3_Vector::operator() (int i) const
- double & dvm3_Vector::operator() (int i)

6.6.1 Function Documentation

6.6.1.1 double& dvm3_Vector::operator() (int *i*) `[inherited]`

Read/write access to component [i].

Returns:

reference to component

Parameters:

i index

6.6.1.2 double dvm3_Vector::operator() (int *i*) const `[inherited]`

Copy of component [i].

Returns:

copy of component [i]

Parameters:

i index

6.6.1.3 double & dvm3_Matrix::operator() (int *i*, int *j*) `[inline, inherited]`

Return element [i][j] (non-const version)

Returns:

element[i][j]

Parameters:

i index
j index

Definition at line 1018 of file dvm3_matrix.h.

6.6.1.4 double dvm3_Matrix::operator() (int *i*, int *j*) const `[inline, inherited]`

Return element [i][j] (const version)

Returns:

element[i][j]

Parameters:

i index
j index

Definition at line 1014 of file dvm3_matrix.h.

6.6.1.5] double& dvm3_Vector::operator[ ] (int *i*) `[inherited]`

Read/write access to component [i].

Returns:

reference to component

Parameters:

i index

6.6.1.6] double const& dvm3_Vector::operator[ ] (int *i*) const `[inherited]`

Readonly access to component [i].

Returns:

const reference to component

Parameters:

i index

6.6.1.7] double $*$ dvm3_Matrix::operator[ ] (int *i*) `[inline, inherited]`

Return row i (non-const version)

Returns:

row i

Parameters:

i index of row to be returned

Definition at line 1010 of file dvm3_matrix.h.

6.6.1.8] double const ∗ dvm3_Matrix::operator[ ] (int *i*) const `[inline, inherited]`

Return row i (const version)

Returns:

row i

Parameters:

i index of row to be returned

Definition at line 1006 of file dvm3_matrix.h.

6.7 Assignment, op= operators

- dvm3_Matrix dvm3_Matrix::operator+ (dvm3_Matrix const &m1, dvm3_Matrix const &m2)
- dvm3_Matrix dvm3_Matrix::operator+ (double d, dvm3_Matrix const &m)
- dvm3_Matrix dvm3_Matrix::operator- (dvm3_Matrix const &m1, dvm3_Matrix const &m2)
- dvm3_Matrix dvm3_Matrix::operator∗ (dvm3_Matrix const &m1, dvm3_Matrix const &m2)
- dvm3_Matrix dvm3_Matrix::operator/ (dvm3_Matrix const &m1, dvm3_Matrix const &m2)
- dvm3_Matrix dvm3_Matrix::operator- (double d, dvm3_Matrix const &m)
- dvm3_Matrix dvm3_Matrix::operator∗ (double d, dvm3_Matrix const &m)
- dvm3_Matrix dvm3_Matrix::operator/ (double d, dvm3_Matrix const &m)
- dvm3_Matrix dvm3_Matrix::operator+ (dvm3_Matrix const &m, double d)
- dvm3_Matrix dvm3_Matrix::operator- (dvm3_Matrix const &m, double d)
- dvm3_Matrix dvm3_Matrix::operator∗ (dvm3_Matrix const &m, double d)
- dvm3_Matrix dvm3_Matrix::operator/ (dvm3_Matrix const &m, double d)

Functions

- dvm3_Matrix & dvm3_Matrix::operator= (dvm3_Matrix const &rhs)
- dvm3_Matrix & dvm3_Matrix::operator= (double rhs)
- dvm3_Matrix & dvm3_Matrix::operator+= (dvm3_Matrix const &rhs)
- dvm3_Matrix & dvm3_Matrix::operator-= (dvm3_Matrix const &rhs)
- dvm3_Matrix & dvm3_Matrix::operator∗= (dvm3_Matrix const &rhs)
- dvm3_Matrix & dvm3_Matrix::operator/= (dvm3_Matrix const &rhs)
- dvm3_Matrix & dvm3_Matrix::operator+= (double rhs)
- dvm3_Matrix & dvm3_Matrix::operator-= (double rhs)

- dvm3_Matrix & dvm3_Matrix::operator∗= (double rhs)
- dvm3_Matrix & dvm3_Matrix::operator/= (double rhs)
- dvm3_Vector & dvm3_Vector::operator= (dvm3_Vector const &rhs)
- dvm3_Vector & dvm3_Vector::operator= (double rhs)
- dvm3_Vector & dvm3_Vector::operator+= (dvm3_Vector const &rhs)
- dvm3_Vector & dvm3_Vector::operator-= (dvm3_Vector const &rhs)
- dvm3_Vector & dvm3_Vector::operator∗= (dvm3_Vector const &rhs)
- dvm3_Vector & dvm3_Vector::operator/= (dvm3_Vector const &rhs)
- dvm3_Vector & dvm3_Vector::operator+= (double rhs)
- dvm3_Vector & dvm3_Vector::operator-= (double rhs)
- dvm3_Vector & dvm3_Vector::operator∗= (double rhs)
- dvm3_Vector & dvm3_Vector::operator/= (double rhs)

6.7.1 Function Documentation

6.7.1.1 dvm3_Vector& dvm3_Vector::operator∗= (double *rhs*) `[inherited]`

Multiply each component by rhs.

Returns:

modified vector

Parameters:

rhs value

6.7.1.2 dvm3_Vector& dvm3_Vector::operator∗= (dvm3_Vector const & *rhs*) `[inherited]`

Component-wise ∗= operator.

Returns:

modified vector

Parameters:

rhs value to be multiplied

6.7.1.3 dvm3_Matrix& dvm3_Matrix::operator∗= (double *rhs*) `[inherited]`

Component-wise ∗= operator.

Returns:

modified matrix

Parameters:

rhs value by which each component of this matrix is to be multiplied

6.7.1.4 dvm3_Matrix& dvm3_Matrix::operator∗= (dvm3_Matrix const & *rhs*) `[inherited]`

Component-wise ∗= operator.

Returns:

modified matrix

Parameters:

rhs value by which each component of this matrix is to be multiplied

6.7.1.5 dvm3_Vector& dvm3_Vector::operator+= (double *rhs*) `[inherited]`

Add rhs to each component.

Returns:

modified vector

Parameters:

rhs value to be added

6.7.1.6 dvm3_Vector& dvm3_Vector::operator+= (dvm3_Vector const & *rhs*) `[inherited]`

Component-wise += operator.

Returns:

modified vector

Parameters:

rhs value to be added

6.7.1.7 dvm3_Matrix& dvm3_Matrix::operator+= (double *rhs*) `[inherited]`

Component-wise /= operator.

Returns:

modified matrix

Parameters:

rhs value to be added to each component of this matrix

6.7.1.8 dvm3_Matrix& dvm3_Matrix::operator+= (dvm3_Matrix const & *rhs*) `[inherited]`

Component-wise += operator.

Returns:

modified matrix

Parameters:

rhs value to be added to each component of this matrix

6.7.1.9 dvm3_Vector& dvm3_Vector::operator-= (double *rhs*) `[inherited]`

Subtract rhs from each component.

Returns:

modified vector

Parameters:

rhs value to be added

6.7.1.10 dvm3_Vector& dvm3_Vector::operator-= (dvm3_Vector const & *rhs*) `[inherited]`

Component-wise -= operator.

Returns:

modified vector

Parameters:

rhs value to be subtracted

6.7.1.11 dvm3_Matrix& dvm3_Matrix::operator-= (double *rhs*) `[inherited]`

Component-wise /= operator.

Returns:

modified matrix

Parameters:

rhs value to be subtracted from each component of this matrix

6.7.1.12 dvm3_Matrix& dvm3_Matrix::operator-= (dvm3_Matrix const & *rhs*) `[inherited]`

Component-wise -= operator.

Returns:

modified matrix

Parameters:

rhs value to be subtracted from each component of this matrix

6.7.1.13 dvm3_Vector& dvm3_Vector::operator/= (double *rhs*) `[inherited]`

Divide each component by rhs.

Returns:

modified vector

Parameters:

rhs value

6.7.1.14 dvm3_Vector& dvm3_Vector::operator/= (dvm3_Vector const & *rhs*) `[inherited]`

Component-wise /= operator.

Returns:

modified vector

Parameters:

rhs value to be divided by

6.7.1.15 dvm3_Matrix& dvm3_Matrix::operator/= (double *rhs*) `[inherited]`

Component-wise /= operator.

Returns:

modified matrix

Parameters:

rhs value by which each component of this matrix is to be divided

6.7.1.16 dvm3_Matrix& dvm3_Matrix::operator/= (dvm3_Matrix const & *rhs*) `[inherited]`

Component-wise /= operator.

Returns:

modified matrix

Parameters:

rhs value by which each component of this matrix is to be divided

6.7.1.17 dvm3_Vector & dvm3_Vector::operator= (double *rhs*) `[inline, inherited]`

Assignment operator

Returns:

modified vector

Parameters:

rhs value to be assigned

Definition at line 710 of file dvm3_vector.h.

6.7.1.18 dvm3_Vector & dvm3_Vector::operator= (dvm3_Vector const & *rhs*) `[inline, inherited]`

Copy-assignment operator

Returns:

modified vector

Parameters:

rhs vector value for assignment

Definition at line 706 of file dvm3_vector.h.

References dvm3_Vector::data_.

6.7.1.19 dvm3_Matrix & dvm3_Matrix::operator= (double *rhs*) `[inline, inherited]`

Assign the value of rhs to each component of this matrix

Returns:

modified matrix

Parameters:

rhs double value to be assigned to each component of this matrix

Definition at line 967 of file dvm3_matrix.h.

6.7.1.20 dvm3_Matrix & dvm3_Matrix::operator= (dvm3_Matrix const & *rhs*) `[inline, inherited]`

Copy-assignment operator

Returns:

modified matrix

Parameters:

rhs matrix value for assignment

Definition at line 960 of file dvm3_matrix.h.

References dvm3_Matrix::data_.

6.7.2 Friends

6.7.2.1 dvm3_Matrix operator∗ (dvm3_Matrix const & *m*, double *d*) `[friend, inherited]`

Set each component to m[i][j] ∗ d.

Returns:

result

Parameters:

m a matrix
d a double

6.7.2.2 dvm3_Matrix operator∗ (double *d*, dvm3_Matrix const & *m*) `[friend, inherited]`

Set each component to d ∗ m[i][j].

Returns:

result

Parameters:

d a double
m a matrix

6.7.2.3 dvm3_Matrix operator∗ (dvm3_Matrix const & *m1*, dvm3_Matrix const & *m2*) `[friend, inherited]`

Component-wise product of m1 and m2.

Returns:

component-wise product

Parameters:

m1 1st matrix
m2 2nd matrix

6.7.2.4 dvm3_Matrix operator+ (dvm3_Matrix const & *m*, double *d*) `[friend, inherited]`

Set each component to m[i][j] + d.

Returns:

result

Parameters:

m a matrix
d a double

6.7.2.5 dvm3_Matrix operator+ (double *d*, dvm3_Matrix const & *m*) `[friend, inherited]`

Set each component to d + m[i][j].

Returns:

result

Parameters:

d a double

m a matrix

6.7.2.6 dvm3_Matrix operator+ (dvm3_Matrix const & *m1*, dvm3_Matrix const & *m2*) `[friend, inherited]`

Component-wise sum of m1 and m2.

Returns:

component-wise sum

Parameters:

m1 1st matrix

m2 2nd matrix

6.7.2.7 dvm3_Matrix operator- (dvm3_Matrix const & *m*, double *d*) `[friend, inherited]`

Set each component to m[i][j] - d.

Returns:

result

Parameters:

m a matrix

d a double

6.7.2.8 dvm3_Matrix operator- (double *d*, dvm3_Matrix const & *m*) `[friend, inherited]`

Set each component to d - m[i][j].

Returns:

result

Parameters:

d a double

m a matrix

6.7.2.9 dvm3_Matrix operator- (dvm3_Matrix const & *m1*, dvm3_Matrix const & *m2*) `[friend, inherited]`

Component-wise difference of m1 and m2.

Returns:

component-wise difference

Parameters:

m1 1st matrix

m2 2nd matrix

6.7.2.10 dvm3_Matrix operator/ (dvm3_Matrix const & *m*, double *d*) `[friend, inherited]`

Set each component to m[i][j] / d.

Returns:

result

Parameters:

m a matrix

d a double

6.7.2.11 dvm3_Matrix operator/ (double *d*, dvm3_Matrix const & *m*) `[friend, inherited]`

Set each component to d / m[i][j].

Returns:

result

Parameters:

d a double

m a matrix

6.7.2.12 dvm3_Matrix operator/ (dvm3_Matrix const & *m1*, dvm3_Matrix const & *m2*) `[friend, inherited]`

Component-wise division of m1 and m2.

Returns:

component-wise ratio

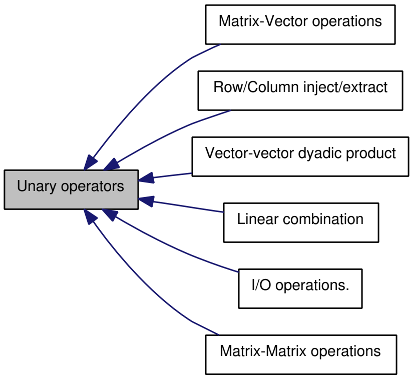
Parameters:

m1 1st matrix

m2 2nd matrix

6.8 Unary operators

Collaboration diagram for Unary operators:



Modules

- Row/Column inject/extract
- Vector-vector dyadic product
- Linear combination
- Matrix-Vector operations
- Matrix-Matrix operations
- I/O operations.

- void dvm3_Matrix::orthonormalize ()

- void dvm3_Matrix::dyad_product (dvm3_Matrix &result, double const v1[ ], double const v2[ ])

- void dvm3_Matrix::mvmult (double result[ ], dvm3_Matrix const &m, double const v[ ])
- void dvm3_Matrix::mtvmult (dvm3_Vector &result, dvm3_Matrix const &m, dvm3_Vector const &v)
- void dvm3_Matrix::mtvmult (double result[ ], dvm3_Matrix const &m, double const v[ ])

- dvm3_Matrix dvm3_Matrix::operator- (dvm3_Matrix const &m1, dvm3_Matrix const &m2)
- dvm3_Matrix dvm3_Matrix::operator∗ (dvm3_Matrix const &m1, dvm3_Matrix const &m2)
- dvm3_Matrix dvm3_Matrix::operator/ (dvm3_Matrix const &m1, dvm3_Matrix const &m2)
- dvm3_Matrix dvm3_Matrix::operator- (double d, dvm3_Matrix const &m)
- dvm3_Matrix dvm3_Matrix::operator∗ (double d, dvm3_Matrix const &m)
- dvm3_Matrix dvm3_Matrix::operator/ (double d, dvm3_Matrix const &m)
- dvm3_Matrix dvm3_Matrix::operator+ (dvm3_Matrix const &m, double d)
- dvm3_Matrix dvm3_Matrix::operator- (dvm3_Matrix const &m, double d)
- dvm3_Matrix dvm3_Matrix::operator∗ (dvm3_Matrix const &m, double d)
- dvm3_Matrix dvm3_Matrix::operator/ (dvm3_Matrix const &m, double d)

Functions

- dvm3_Matrix & dvm3_Matrix::operator+ ()
- dvm3_Matrix & dvm3_Matrix::operator- ()
- dvm3_Vector & dvm3_Vector::operator+ ()
- dvm3_Vector & dvm3_Vector::operator- ()

6.8.1 Function Documentation

6.8.1.1 dvm3_Vector& dvm3_Vector::operator+ () `[inherited]`

Do-nothing unary +; provided for completeness.

6.8.1.2 dvm3_Matrix & dvm3_Matrix::operator+ () `[inline, inherited]`

Do-nothing unary +; provided for completeness.

Definition at line 1021 of file dvm3_matrix.h.

6.8.1.3 dvm3_Vector& dvm3_Vector::operator- () `[inherited]`

Unary -; negates each component of this matrix

6.8.1.4 dvm3_Matrix & dvm3_Matrix::operator- () `[inline, inherited]`

Unary -; negates each component of this matrix

Definition at line 1025 of file dvm3_matrix.h.

6.8.1.5 void dvm3_Matrix::orthonormalize () `[inherited]`

Force the matrix to be a proper orthonormal matrix.

The y row is replaced by z cross x, and then x is replaced with y cross z. Normalize all three rows are normalized. This forms a right-handed orthonormal triple.

"Proper", in this context, means that it preserves the parity of the coordinate system, transforming right-handed coordinates into right-handed coordinates.

Definition at line 36 of file dvm3_matrix.cc.

Referenced by dvm3_RotMat::dvm3_RotMat().

6.8.2 Friends

6.8.2.1 void dyad_product (dvm3_Matrix & *result*, double const *v1*[ ], double const *v2*[ ]) `[friend, inherited]`

Form dyadic product (outer product) of vectors v1 and v2.

```
For each i, j:  result[i][j] = v1[i] * v2[j]
```

Parameters:

result dyadic product

v1 1st vector

v2 2nd vector

Definition at line 1099 of file dvm3_matrix.h.

6.8.2.2 void mtvmult (double *result*[ ], dvm3_Matrix const & *m*, double const *v*[ ]) `[friend, inherited]`

Matrix multiplication of vector v by transpose of matrix m.

```
result = (transpose of m) _matrix_multiply_ v.
```

Parameters:

result product

m matrix

v vector

Definition at line 1122 of file dvm3_matrix.h.

6.8.2.3 void mtvmult (dvm3_Vector & *result*, dvm3_Matrix const & *m*, dvm3_Vector const & *v*) `[friend, inherited]`

Matrix multiplication of vector v by transpose of matrix m.

```
result = (transpose of m) _matrix_multiply_ v.
```

Parameters:

result product

m matrix

v vector

Definition at line 1118 of file dvm3_matrix.h.

6.8.2.4 void mvmult (double *result*[ ], dvm3_Matrix const & *m*, double const *v*[ ]) `[friend, inherited]`

Matrix multiplication of vector v by matrix m.

```
result = m _matrix_multiply_ v.
```

Parameters:

result product
m matrix
v vector

Definition at line 1114 of file dvm3_matrix.h.

6.8.2.5 dvm3_Matrix operator∗ (dvm3_Matrix const & *m*, double *d*) `[friend, inherited]`

Set each component to m[i][j] ∗ d.

Returns:

result

Parameters:

m a matrix
d a double

6.8.2.6 dvm3_Matrix operator∗ (double *d*, dvm3_Matrix const & *m*) `[friend, inherited]`

Set each component to d ∗ m[i][j].

Returns:

result

Parameters:

d a double
m a matrix

6.8.2.7 dvm3_Matrix operator∗ (dvm3_Matrix const & *m1*, dvm3_Matrix const & *m2*) `[friend, inherited]`

Component-wise product of m1 and m2.

Returns:

component-wise product

Parameters:

m1 1st matrix

m2 2nd matrix

6.8.2.8 dvm3_Matrix operator+ (dvm3_Matrix const & *m*, double *d*) `[friend, inherited]`

Set each component to m[i][j] + d.

Returns:

result

Parameters:

m a matrix

d a double

6.8.2.9 dvm3_Matrix operator- (dvm3_Matrix const & *m*, double *d*) `[friend, inherited]`

Set each component to m[i][j] - d.

Returns:

result

Parameters:

m a matrix

d a double

6.8.2.10 dvm3_Matrix operator- (double *d*, dvm3_Matrix const & *m*) `[friend, inherited]`

Set each component to d - m[i][j].

Returns:

result

Parameters:

d a double

m a matrix

6.8.2.11 dvm3_Matrix operator- (dvm3_Matrix const & *m1*, dvm3_Matrix const & *m2*) `[friend, inherited]`

Component-wise difference of m1 and m2.

Returns:

component-wise difference

Parameters:

m1 1st matrix

m2 2nd matrix

6.8.2.12 dvm3_Matrix operator/ (dvm3_Matrix const & *m*, double *d*) `[friend, inherited]`

Set each component to m[i][j] / d.

Returns:

result

Parameters:

m a matrix

d a double

6.8.2.13 dvm3_Matrix operator/ (double *d*, dvm3_Matrix const & *m*) `[friend, inherited]`

Set each component to d / m[i][j].

Returns:

result

Parameters:

d a double

m a matrix

6.8.2.14 dvm3_Matrix operator/ (dvm3_Matrix const & *m1*, dvm3_Matrix const & *m2*) `[friend, inherited]`

Component-wise division of m1 and m2.

Returns:

component-wise ratio

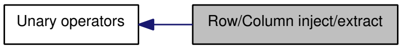
Parameters:

m1 1st matrix

m2 2nd matrix

6.9 Row/Column inject/extract

Collaboration diagram for Row/Column inject/extract:



Functions

- void dvm3_Matrix::inject_row (int whichrow, dvm3_Vector const &new_row)
- void dvm3_Matrix::inject_row (int whichrow, double const new_row[ ])
- void dvm3_Matrix::inject_col (int whichcol, dvm3_Vector const &col)
- void dvm3_Matrix::inject_col (int whichcol, double const col[ ])
- void dvm3_Matrix::extract_row (int whichrow, dvm3_Vector &row) const
- void dvm3_Matrix::extract_row (int whichrow, double row[ ]) const
- void dvm3_Matrix::extract_col (int whichcol, dvm3_Vector &col) const
- void dvm3_Matrix::extract_col (int whichcol, double ∗col) const

6.9.1 Function Documentation

6.9.1.1 void dvm3_Matrix::extract_col (int *whichcol*, double ∗ *col*) const `[inherited]`

Copy column whichcol into supplied vector

Parameters:

whichcol index of column to be copied

col destination (C-style vector)

6.9.1.2 void dvm3_Matrix::extract_col (int *whichcol*, dvm3_Vector & *col*) const `[inline, inherited]`

Copy column whichcol into supplied vector

Parameters:

- ***whichcol*** index of column to be copied
- ***col*** destination

Definition at line 1002 of file dvm3_matrix.h.

References dvm3_Vector::data_.

6.9.1.3 void dvm3_Matrix::extract_row (int *whichrow*, double *row*[ ]) const `[inline, inherited]`

Copy row whichrow into supplied vector

Parameters:

- ***whichrow*** index of row to be copied
- ***row*** destination (C-style vector)

Definition at line 990 of file dvm3_matrix.h.

6.9.1.4 void dvm3_Matrix::extract_row (int *whichrow*, dvm3_Vector & *row*) const `[inline, inherited]`

Copy row whichrow into supplied dvm3_Vector

Parameters:

- ***whichrow*** index of row to be copied
- ***row*** destination

Definition at line 994 of file dvm3_matrix.h.

References dvm3_Vector::data_.

6.9.1.5 void dvm3_Matrix::inject_col (int *whichcol*, double const *col*[ ]) `[inline, inherited]`

Replace column whichcol by new_col

Parameters:

- ***whichcol*** index of column to be replaced

new_col replacement column (C-style vector)

Definition at line 982 of file dvm3_matrix.h.

6.9.1.6 void dvm3_Matrix::inject_col (int *whichcol*, dvm3_Vector const & *col*) `[inline, inherited]`

Replace column whichcol by new_col

Parameters:

whichcol index of column to be replaced

new_col replacement column

Definition at line 986 of file dvm3_matrix.h.

References dvm3_Vector::data_.

6.9.1.7 void dvm3_Matrix::inject_row (int *whichrow*, double const *new_row*[ ]) `[inline, inherited]`

Replace row whichrow by new_row

Parameters:

whichrow index of row to be replaced

new_row replacement row (C-style vector)

Definition at line 974 of file dvm3_matrix.h.

6.9.1.8 void dvm3_Matrix::inject_row (int *whichrow*, dvm3_Vector const & *new_row*) `[inline, inherited]`

Replace row whichrow by new_row

Parameters:

whichrow index of row to be replaced

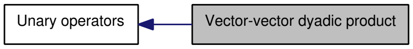
new_row replacement row

Definition at line 978 of file dvm3_matrix.h.

References dvm3_Vector::data_.

6.10 Vector-vector dyadic product

Collaboration diagram for Vector-vector dyadic product:



- void dvm3_Matrix::dyad_product (dvm3_Matrix &result, dvm3_Vector const &v1, dvm3_Vector const &v2)
- void dvm3_Matrix::dyad_product (dvm3_Matrix &result, double const v1[ ], double const v2[ ])

Functions

- void dvm3_Matrix::dyad_product (dvm3_Vector const &v1, dvm3_Vector const &v2)
- void dvm3_Matrix::dyad_product (double const v1[ ], double const v2[ ])

6.10.1 Function Documentation

6.10.1.1 void dvm3_Matrix::dyad_product (double const *v1*[ ], double const *v2*[ ]) `[inline, inherited]`

Form dyadic product (outer product) of vectors v1 and v2.

```
For each i, j:  m[i][j] = v1[i] * v2[j]
```

Parameters:

v1	1st vector
v2	2nd vector

Definition at line 1064 of file dvm3_matrix.h.

6.10.1.2 void dvm3_Matrix::dyad_product (dvm3_Vector const & *v1*, dvm3_Vector const & *v2*) `[inherited]`

Form dyadic product (outer product) of vectors v1 and v2.

```
For each i, j:  m[i][j] = v1[i] * v2[j]
```

Parameters:

v1	1st vector
v2	2nd vector

6.10.2 Friends

6.10.2.1 void dyad_product (dvm3_Matrix & *result*, double const *v1*[ ], double const *v2*[ ]) `[friend, inherited]`

Form dyadic product (outer product) of vectors v1 and v2.

```
For each i, j:  result[i][j] = v1[i] * v2[j]
```

Parameters:

result dyadic product

v1 1st vector

v2 2nd vector

Definition at line 1099 of file dvm3_matrix.h.

6.10.2.2 void dyad_product (dvm3_Matrix & *result*, dvm3_Vector const & *v1*, dvm3_Vector const & *v2*) `[friend, inherited]`

Form dyadic product (outer product) of vectors v1 and v2.

```
For each i, j:  result[i][j] = v1[i] * v2[j]
```

Parameters:

result dyadic product

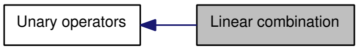
v1 1st vector

v2 2nd vector

Definition at line 1093 of file dvm3_matrix.h.

6.11 Linear combination

Collaboration diagram for Linear combination:



- void dvm3_Matrix::lincomb (dvm3_Matrix &result, double c1, dvm3_Matrix const &m1, double c2, dvm3_Matrix const &m2)

- void dvm3_Matrix::mvmult (dvm3_Vector &result, dvm3_Matrix const &m, dvm3_Vector const &v)
- void dvm3_Matrix::mvmult (double result[ ], dvm3_Matrix const &m, double const v[ ])
- void dvm3_Matrix::mtvmult (dvm3_Vector &result, dvm3_Matrix const &m, dvm3_Vector const &v)
- void dvm3_Matrix::mtvmult (double result[ ], dvm3_Matrix const &m, double const v[ ])

- void dvm3_Vector::lincomb (dvm3_Vector &result, double c1, dvm3_Vector const &v1, double c2, dvm3_Vector const &v2)

Functions

- void dvm3_Matrix::lincomb (double c1, dvm3_Matrix const &m1, double c2, dvm3_Matrix const &m2)
- void dvm3_Vector::lincomb (double c1, dvm3_Vector const &v1, double c2, dvm3_Vector const &v2)

6.11.1 Function Documentation

6.11.1.1 void dvm3_Vector::lincomb (double *c1*, dvm3_Vector const & *v1*, double *c2*, dvm3_Vector const & *v2*) `[inherited]`

Form linear combination: this = c1 ∗ v1 + c2 ∗ v2.

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

Parameters:

c1 1st scalar

v1 1st vector

c2 2nd scalar

v2 2nd vector

6.11.1.2 void dvm3_Matrix::lincomb (double *c1*, dvm3_Matrix const & *m1*, double *c2*, dvm3_Matrix const & *m2*) `[inline, inherited]`

Form linear combination: this = c1 ∗ m1 + c2 ∗ m2.

```
    For each i, j:  m[i][j] = c1*m1[i][j] + c2*m2[i][j]
```

Parameters:

- ***c1*** 1st scalar
- ***m1*** 1st matrix
- ***c2*** 2nd scalar
- ***m2*** 2nd matrix

Definition at line 1068 of file dvm3_matrix.h.

References dvm3_Matrix::data_.

6.11.2 Friends

6.11.2.1 void lincomb (dvm3_Vector & *result*, double *c1*, dvm3_Vector const & *v1*, double *c2*, dvm3_Vector const & *v2*) `[friend, inherited]`

Linear combination c1 $*$ v1 + c2 $*$ v2.

Parameters:

- ***result*** linear combination c1 $*$ v1 + c2 $*$ v2.
- ***c1*** 1st scalar
- ***v1*** 1st vector
- ***c2*** 2nd scalar
- ***v2*** 2nd vector

6.11.2.2 void lincomb (dvm3_Matrix & *result*, double *c1*, dvm3_Matrix const & *m1*, double *c2*, dvm3_Matrix const & *m2*) `[friend, inherited]`

Form linear combination: result = c1 $*$ m1 + c2 $*$ m2.

```
For each i, j:  result[i][j] = c1*m1[i][j] + c2*m2[i][j]
```

Parameters:

- ***result*** resultant linear combination
- ***c1*** 1st scalar
- ***m1*** 1st matrix
- ***c2*** 2nd scalar
- ***m2*** 2nd matrix

Definition at line 1104 of file dvm3_matrix.h.

6.11.2.3 void mtvmult (double *result*[ ], dvm3_Matrix const & *m*, double const *v*[ ]) `[friend, inherited]`

Matrix multiplication of vector v by transpose of matrix m.

```
result = (transpose of m) _matrix_multiply_ v.
```

Parameters:

- ***result*** product
- ***m*** matrix
- ***v*** vector

Definition at line 1122 of file dvm3_matrix.h.

6.11.2.4 void mtvmult (dvm3_Vector & *result*, dvm3_Matrix const & *m*, dvm3_Vector const & *v*) `[friend, inherited]`

Matrix multiplication of vector v by transpose of matrix m.

```
result = (transpose of m) _matrix_multiply_ v.
```

Parameters:

- ***result*** product
- ***m*** matrix
- ***v*** vector

Definition at line 1118 of file dvm3_matrix.h.

6.11.2.5 void mvmult (double *result*[ ], dvm3_Matrix const & *m*, double const *v*[ ]) `[friend, inherited]`

Matrix multiplication of vector v by matrix m.

```
result = m _matrix_multiply_ v.
```

Parameters:

- ***result*** product
- ***m*** matrix
- ***v*** vector

Definition at line 1114 of file dvm3_matrix.h.

6.11.2.6 void mvmult (dvm3_Vector & *result*, dvm3_Matrix const & *m*, dvm3_Vector const & *v*) `[friend, inherited]`

Matrix multiplication of vector v by matrix m.

```
result = m _matrix_multiply_ v.
```

Parameters:

result product

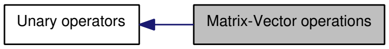
m matrix

v vector

Definition at line 1110 of file dvm3_matrix.h.

6.12 Matrix-Vector operations

Collaboration diagram for Matrix-Vector operations:



- void dvm3_Matrix::mvmult (dvm3_Vector &result, dvm3_Matrix const &m, dvm3_Vector const &v)

Functions

- void dvm3_Matrix::mvmult (dvm3_Vector &result, dvm3_Vector const &v) const
- void dvm3_Matrix::mvmult (double result[ ], double const v[ ]) const
- void dvm3_Matrix::mtvmult (dvm3_Vector &result, dvm3_Vector const &v) const
- void dvm3_Matrix::mtvmult (double result[ ], double const v[ ]) const

6.12.1 Function Documentation

6.12.1.1 void dvm3_Matrix::mtvmult (double *result*[ ], double const *v*[ ]) const `[inline, inherited]`

Matrix multiplication of vector v by the transpose of this matrix.

```
result = (transpose of *this) _matrix_multiply_ v.
```

Parameters:

- ***result*** resultant vector
- ***v*** vector

Definition at line 1085 of file dvm3_matrix.h.

6.12.1.2 void dvm3_Matrix::mtvmult (dvm3_Vector & *result*, dvm3_Vector const & *v*) const `[inline, inherited]`

Matrix multiplication of vector v by the transpose of this matrix.

```
result = (transpose of *this) _matrix_multiply_ v.
```

Parameters:

- ***result*** resultant vector
- ***v*** vector

Definition at line 1081 of file dvm3_matrix.h.

References dvm3_Vector::data_.

6.12.1.3 void dvm3_Matrix::mvmult (double *result*[ ], double const *v*[ ]) const `[inline, inherited]`

Matrix multiplication of vector v by this matrix.

```
result = *this _matrix_multiply_ v.
```

Parameters:

- ***result*** resultant vector
- ***v*** vector

Definition at line 1077 of file dvm3_matrix.h.

6.12.1.4 void dvm3_Matrix::mvmult (dvm3_Vector & *result*, dvm3_Vector const & *v*) const `[inline, inherited]`

Matrix multiplication of vector v by this matrix.

```
result = *this _matrix_multiply_ v.
```

Parameters:

- ***result*** resultant vector
- ***v*** vector

Definition at line 1073 of file dvm3_matrix.h.

References dvm3_Vector::data_.

6.12.2 Friends

6.12.2.1 void mvmult (dvm3_Vector & *result*, dvm3_Matrix const & *m*, dvm3_Vector const & *v*) `[friend, inherited]`

Matrix multiplication of vector v by matrix m.

```
result = m _matrix_multiply_ v.
```

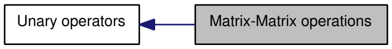
Parameters:

- ***result*** product
- ***m*** matrix
- ***v*** vector

Definition at line 1110 of file dvm3_matrix.h.

6.13 Matrix-Matrix operations

Collaboration diagram for Matrix-Matrix operations:



- void dvm3_Matrix::mmult (dvm3_Matrix &result, dvm3_Matrix const &m1, dvm3_Matrix const &m2)

Functions

- void dvm3_Matrix::mmult (dvm3_Matrix const &m1, dvm3_Matrix const &m2)

6.13.1 Function Documentation

6.13.1.1 void dvm3_Matrix::mmult (dvm3_Matrix const & *m1*, dvm3_Matrix const & *m2*) `[inline, inherited]`

Matrix multiplication of m1 by m2.

```
*this = m1 _matrix_multiply_ m2.
```

Parameters:

m1 1st matrix

m2 2nd matrix

Definition at line 1089 of file dvm3_matrix.h.

References dvm3_Matrix::data_.

6.13.2 Friends

6.13.2.1 void mmult (dvm3_Matrix & *result*, dvm3_Matrix const & *m1*, dvm3_Matrix const & *m2*) `[friend, inherited]`

Matrix multiplication of m1 by m2.

```
/  result = m1 _matrix_multiply_ m2.
```

Parameters:

result product

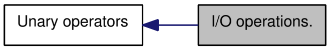
m1 1st matrix

m2 2nd matrix

Definition at line 1126 of file dvm3_matrix.h.

6.14 I/O operations.

Collaboration diagram for I/O operations.:



Functions

- std::ostream & dvm3_Matrix::print_on (std::ostream &os, char const pre[ ]="", char const post[ ]="") const
- void dvm3_Matrix::cprint_on (FILE ∗os, char const prefix[ ]="", char const postfix[ ]="")
- std::ostream & dvm3_Vector::print_on (std::ostream &os, char const pre[ ]="", char const post[ ]="") const
- void dvm3_Vector::cprint_on (FILE ∗of, char const prefix[ ]="", char const postfix[ ]="")

Friends

- std::ostream & dvm3_Matrix::operator<< (std::ostream &os, dvm3_Matrix const &)
- std::ostream & dvm3_Vector::operator<< (std::ostream &os, dvm3_Vector const &)

6.14.1 Function Documentation

6.14.1.1 void dvm3_Vector::cprint_on (FILE ∗ *of*, char const *prefix*[ ] = "", char const *postfix*[ ] = "") `[inherited]`

Print this dvm3_Vector to a FILE∗ stream.

Parameters:

of the FILE∗

v vector to be printed

prefix optional prefix string

postfix optional postfix string

6.14.1.2 void dvm3_Matrix::cprint_on (FILE ∗ *os*, char const *prefix*[ ] = "", char const *postfix*[ ] = "") `[inline, inherited]`

Print this dvm3_Matrix to a FILE∗ stream.

Parameters:

of the FILE∗

v vector to be printed

prefix optional prefix string

postfix optional postfix string

Definition at line 1138 of file dvm3_matrix.h.

6.14.1.3 std::ostream& dvm3_Vector::print_on (std::ostream & *os*, char const *pre*[ ] = `""`, char const *post*[ ] = `""`) const `[inherited]`

Print this dvm3_Vector to an ostream.

Parameters:

- ***os*** the ostream
- ***prefix*** optional prefix string
- ***postfix*** optional postfix string

6.14.1.4 std::ostream & dvm3_Matrix::print_on (std::ostream & *os*, char const *pre*[ ] = `""`, char const *post*[ ] = `""`) const `[inline, inherited]`

Print this dvm3_Matrix to an ostream.

Parameters:

- ***os*** the ostream
- ***prefix*** optional prefix string
- ***postfix*** optional postfix string

Definition at line 1130 of file dvm3_matrix.h.

6.14.2 Friends

6.14.2.1 std::ostream& operator<< (std::ostream & *os*, dvm3_Vector const &) `[friend, inherited]`

Formatted dvm3_Matrix output to an ostream.

Parameters:

- ***os*** the ostream

6.14.2.2 std::ostream& operator<< (std::ostream & *os*, dvm3_Matrix const & *m*) `[friend, inherited]`

Formatted dvm3_Matrix output to an ostream.

Parameters:

- ***os*** the ostream

Definition at line 1134 of file dvm3_matrix.h.

6.15 Destructor; Constructors

Functions

- dvm3_Vector::∼dvm3_Vector ()
- dvm3_Vector::dvm3_Vector ()
- dvm3_Vector::dvm3_Vector (double x, double y, double z)
- dvm3_Vector::dvm3_Vector (dvm3_Vector const &other)
- void dvm3_Vector::copy_from_cvec (double const ∗cv)
- void dvm3_Vector::copy_to_cvec (double ∗v) const

6.15.1 Function Documentation

6.15.1.1 void dvm3_Vector::copy_from_cvec (double const ∗ *cv*) `[inline, inherited]`

Copy operation: copy from a c-style vector.

Parameters:

cv vector to copy

REQUIREMENT: cv has a length of at least 3 contiguous doubles and is appropriately aligned for doubles.

Definition at line 718 of file dvm3_vector.h.

6.15.1.2 void dvm3_Vector::copy_to_cvec (double ∗ *v*) const `[inline, inherited]`

Copy operation: copy to a c-style vector.

Parameters:

v vector copy

REQUIREMENT: cv has a length of at least 3 contiguous doubles and is appropriately aligned for doubles.

Definition at line 726 of file dvm3_vector.h.

6.15.1.3 dvm3_Vector::dvm3_Vector (dvm3_Vector const & *other*) `[inline, inherited]`

Copy constructor.

Parameters:

other vector to copy

Definition at line 697 of file dvm3_vector.h.

6.15.1.4 dvm3_Vector::dvm3_Vector (double *x*, double *y*, double *z*) `[inline, inherited]`

Construct dvm3_Vector from 3 doubles.

Parameters:

x x component

y y component

z z component

Definition at line 691 of file dvm3_vector.h.

6.15.1.5 dvm3_Vector::dvm3_Vector () `[inline, inherited]`

Default constructor; NO INITIALIZATION IS APPLIED.

Definition at line 687 of file dvm3_vector.h.

6.15.1.6 dvm3_Vector::∼dvm3_Vector () `[inline, inherited]`

Do-nothing destructor.

Definition at line 683 of file dvm3_vector.h.

6.16 Dot, Vector products

- double dvm3_Vector::dot (dvm3_Vector const &v1, dvm3_Vector const &v2)
- void dvm3_Vector::cross (dvm3_Vector &result, dvm3_Vector const &v1, dvm3_Vector const &v2)

- void dvm3_Vector::dyad_product (dvm3_Matrix &, const dvm3_Vector &, const dvm3_Vector &)
- void dvm3_Vector::mvmult (dvm3_Vector &, const dvm3_Matrix &, const dvm3_Vector &)
- void dvm3_Vector::mtvmult (dvm3_Vector &, const dvm3_Matrix &, const dvm3_Vector &)

Functions

- double dvm3_Vector::dot (dvm3_Vector const &v) const
- void dvm3_Vector::cross (dvm3_Vector const &v1, dvm3_Vector const &v2)

6.16.1 Function Documentation

6.16.1.1 void dvm3_Vector::cross (dvm3_Vector const & *v1*, dvm3_Vector const & *v2*) `[inherited]`

Cross product

Set this dvm3_Vector to be the vector cross product of v1 and v2.

Parameters:

v1 1st vector

v2 2nd vector

6.16.1.2 double dvm3_Vector::dot (dvm3_Vector const & *v*) const `[inherited]`

Dot product

Returns:

dot product (scalar product) of v with this dvm3_Vector.

Parameters:

v other vector

6.16.2 Friends

6.16.2.1 void cross (dvm3_Vector & *result*, dvm3_Vector const & *v1*, dvm3_Vector const & *v2*) `[friend, inherited]`

Cross product

Returns:

cross product of v1 and v2.

Parameters:

v1 1st vector

v2 2nd vector

6.16.2.2 double dot (dvm3_Vector const & *v1*, dvm3_Vector const & *v2*) `[friend, inherited]`

Dot product

Returns:

dot product of v1 and v2.

Parameters:

v1 1st vector
v2 2nd vector

6.16.2.3 void dyad_product (dvm3_Matrix & *result*, const dvm3_Vector & *v1*, const dvm3_Vector & *v2*) `[friend, inherited]`

Dyadic product

Form dyadic product (outer product) of vectors v1 and v2.

```
For each i, j:  result[i][j] = v1[i] * v2[j]
```

Parameters:

result dyadic product
v1 1st vector
v2 2nd vector

Definition at line 1093 of file dvm3_matrix.h.

6.16.2.4 void mtvmult (dvm3_Vector & *result*, const dvm3_Matrix & *m*, const dvm3_Vector & *v*) `[friend, inherited]`

Multiply vector by transpose of matrix

Matrix multiplication of vector v by transpose of matrix m.

```
result = (transpose of m) _matrix_multiply_ v.
```

Parameters:

result product
m matrix
v vector

Definition at line 1118 of file dvm3_matrix.h.

6.16.2.5 void mvmult (dvm3_Vector & *result*, const dvm3_Matrix & *m*, const dvm3_Vector & *v*) `[friend, inherited]`

Multiply vector by matrix

Matrix multiplication of vector v by matrix m.

```
result = m _matrix_multiply_ v.
```

Parameters:

result product

m matrix

v vector

Definition at line 1110 of file dvm3_matrix.h.

6.17 Componentwise math operations

Friends

- dvm3_Vector dvm3_Vector::operator+ (dvm3_Vector const &v1, dvm3_Vector const &v2)
- dvm3_Vector dvm3_Vector::operator- (dvm3_Vector const &v1, dvm3_Vector const &v2)
- dvm3_Vector dvm3_Vector::operator∗ (dvm3_Vector const &v1, dvm3_Vector const &v2)
- dvm3_Vector **dvm3_Vector::operator/** (dvm3_Vector const &v1, dvm3_Vector const &v2)
- dvm3_Vector dvm3_Vector::operator+ (double r, dvm3_Vector const &v)
- dvm3_Vector dvm3_Vector::operator- (double r, dvm3_Vector const &v)
- dvm3_Vector dvm3_Vector::operator∗ (double r, dvm3_Vector const &v)
- dvm3_Vector dvm3_Vector::operator/ (double r, dvm3_Vector const &v)
- dvm3_Vector dvm3_Vector::operator+ (dvm3_Vector const &v, double r)
- dvm3_Vector dvm3_Vector::operator- (dvm3_Vector const &v, double r)
- dvm3_Vector dvm3_Vector::operator∗ (dvm3_Vector const &v, double r)
- dvm3_Vector dvm3_Vector::operator/ (dvm3_Vector const &v, double r)

6.17.1 Friends

6.17.1.1 dvm3_Vector operator∗ (dvm3_Vector const & *v*, double *r*) `[friend, inherited]`

component-wise product: v[i] ∗ r

Returns:

component-wise product: v[i] ∗ r

Parameters:

v vector

r scalar

6.17.1.2 dvm3_Vector operator∗ (double *r*, dvm3_Vector const & *v*) `[friend, inherited]`

component-wise product: r ∗ v[i]

Returns:

component-wise product: r ∗ v[i]

Parameters:

r scalar

v vector

6.17.1.3 dvm3_Vector operator∗ (dvm3_Vector const & *v1*, dvm3_Vector const & *v2*) `[friend, inherited]`

component-wise product of v1 and v2.

Returns:

component-wise product of v1 and v2

Parameters:

v1 1st vector

v2 2nd vector

6.17.1.4 dvm3_Vector operator+ (dvm3_Vector const & *v*, double *r*) `[friend, inherited]`

component-wise sum: v[i] + r

Returns:

component-wise sum: v[i] + r

Parameters:

v vector
r scalar

6.17.1.5 dvm3_Vector operator+ (double *r*, dvm3_Vector const & *v*) `[friend, inherited]`

component-wise sum: r + v[i]

Returns:

component-wise sum: r + v[i]

Parameters:

r scalar
v vector

6.17.1.6 dvm3_Vector operator+ (dvm3_Vector const & *v1*, dvm3_Vector const & *v2*) `[friend, inherited]`

component-wise sum of v1 and v2.

Returns:

component-wise sum of v1 and v2

Parameters:

v1 1st vector
v2 2nd vector

6.17.1.7 dvm3_Vector operator- (dvm3_Vector const & *v*, double *r*) `[friend, inherited]`

component-wise difference: v[i] - r

Returns:

component-wise difference: v[i] - r

Parameters:

v vector
r scalar

6.17.1.8 dvm3_Vector operator- (double *r*, dvm3_Vector const & *v*) `[friend, inherited]`

component-wise difference: r - v[i]

Returns:

component-wise difference: r - v[i]

Parameters:

r scalar

v vector

6.17.1.9 dvm3_Vector operator- (dvm3_Vector const & *v1*, dvm3_Vector const & *v2*) `[friend, inherited]`

component-wise difference of v1 and v2.

Returns:

component-wise difference of v1 and v2

Parameters:

v1 1st vector

v2 2nd vector

6.17.1.10 dvm3_Vector operator/ (dvm3_Vector const & *v*, double *r*) `[friend, inherited]`

component-wise division: v[i] / r

Returns:

component-wise division: v[i] / r

Parameters:

v vector

r scalar

6.17.1.11 dvm3_Vector operator/ (double *r*, dvm3_Vector const & *v*) `[friend, inherited]`

component-wise division: r / v[i]

Returns:

component-wise division: r / v[i]

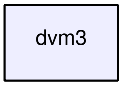
Parameters:

r scalar

v vector

7 Directory Documentation

7.1 dvm3/ Directory Reference



Files

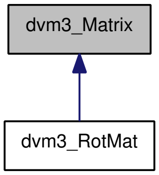
- file **dvm3.h**
- file **dvm3_matrix.cc**
- file **dvm3_matrix.h**
- file **dvm3_rotmat.cc**
- file **dvm3_rotmat.h**
- file **dvm3_vector.h**

8 Class Documentation

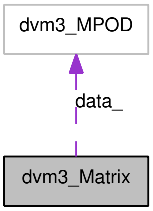
8.1 dvm3_Matrix Class Reference

`#include <dvm3/dvm3_matrix.h>`

Inheritance diagram for dvm3_Matrix:



Collaboration diagram for dvm3_Matrix:



Public Member Functions

- ~dvm3_Matrix ()
- dvm3_Matrix ()
- dvm3_Matrix (dvm3_Matrix const &m)
- dvm3_Matrix (dvm3_Vector const &x, dvm3_Vector const &y, dvm3_Vector const &z)
- dvm3_Matrix (double const x[ ], double const y[ ], double const z[ ])
- void init_by_row (dvm3_Vector const &v1, dvm3_Vector const &v2, dvm3_Vector const &v3)
- void init_by_row (double const v1[ ], double const v2[ ], double const v3[ ])
- void init_by_col (dvm3_Vector const &v1, dvm3_Vector const &v2, dvm3_Vector const &v3)
- void init_by_col (double const v1[ ], double const v2[ ], double const v3[ ])
- double const ∗ operator[ ] (int i) const
- double ∗ operator[ ] (int i)
- double operator() (int i, int j) const
- double & operator() (int i, int j)
- dvm3_Matrix & operator= (dvm3_Matrix const &rhs)
- dvm3_Matrix & operator= (double rhs)
- dvm3_Matrix & operator+= (dvm3_Matrix const &rhs)
- dvm3_Matrix & operator-= (dvm3_Matrix const &rhs)
- dvm3_Matrix & operator∗= (dvm3_Matrix const &rhs)

- dvm3_Matrix & operator/= (dvm3_Matrix const &rhs)
- dvm3_Matrix & operator+= (double rhs)
- dvm3_Matrix & operator-= (double rhs)
- dvm3_Matrix & operator∗= (double rhs)
- dvm3_Matrix & operator/= (double rhs)
- dvm3_Matrix & operator+ ()
- dvm3_Matrix & operator- ()
- void inject_row (int whichrow, dvm3_Vector const &new_row)
- void inject_row (int whichrow, double const new_row[ ])
- void inject_col (int whichcol, dvm3_Vector const &col)
- void inject_col (int whichcol, double const col[ ])
- void extract_row (int whichrow, dvm3_Vector &row) const
- void extract_row (int whichrow, double row[ ]) const
- void extract_col (int whichcol, dvm3_Vector &col) const
- void extract_col (int whichcol, double ∗col) const
- void dyad_product (dvm3_Vector const &v1, dvm3_Vector const &v2)
- void dyad_product (double const v1[ ], double const v2[ ])
- void lincomb (double c1, dvm3_Matrix const &m1, double c2, dvm3_Matrix const &m2)
- void mvmult (dvm3_Vector &result, dvm3_Vector const &v) const
- void mvmult (double result[ ], double const v[ ]) const
- void mtvmult (dvm3_Vector &result, dvm3_Vector const &v) const
- void mtvmult (double result[ ], double const v[ ]) const
- void mmult (dvm3_Matrix const &m1, dvm3_Matrix const &m2)
- std::ostream & print_on (std::ostream &os, char const pre[ ]="", char const post[ ]="") const
- void cprint_on (FILE ∗os, char const prefix[ ]="", char const postfix[ ]="")

  - void orthonormalize ()

Protected Types

- enum **ENelts_** { **ENelts** = 9 }
- enum **ERowOffset_** { **ERow0** = 0, **ERow1** = 3, **ERow2** = 6 }
- enum **EColStride_** { **EColStride** = 3 }

Protected Attributes

- dvm3_MPOD **data_**

Friends

- std::ostream & operator<< (std::ostream &os, dvm3_Matrix const &)

- void dyad_product (dvm3_Matrix &result, dvm3_Vector const &v1, dvm3_Vector const &v2)
- void dyad_product (dvm3_Matrix &result, double const v1[ ], double const v2[ ])

- void lincomb (dvm3_Matrix &result, double c1, dvm3_Matrix const &m1, double c2, dvm3_Matrix const &m2)
- void mvmult (dvm3_Vector &result, dvm3_Matrix const &m, dvm3_Vector const &v)
- void mvmult (double result[ ], dvm3_Matrix const &m, double const v[ ])
- void mtvmult (dvm3_Vector &result, dvm3_Matrix const &m, dvm3_Vector const &v)
- void mtvmult (double result[ ], dvm3_Matrix const &m, double const v[ ])

- void mmult (dvm3_Matrix &result, dvm3_Matrix const &m1, dvm3_Matrix const &m2)

- dvm3_Matrix operator+ (dvm3_Matrix const &m1, dvm3_Matrix const &m2)
- dvm3_Matrix operator- (dvm3_Matrix const &m1, dvm3_Matrix const &m2)
- dvm3_Matrix operator∗ (dvm3_Matrix const &m1, dvm3_Matrix const &m2)
- dvm3_Matrix operator/ (dvm3_Matrix const &m1, dvm3_Matrix const &m2)
- dvm3_Matrix operator+ (double d, dvm3_Matrix const &m)
- dvm3_Matrix operator- (double d, dvm3_Matrix const &m)
- dvm3_Matrix operator∗ (double d, dvm3_Matrix const &m)
- dvm3_Matrix operator/ (double d, dvm3_Matrix const &m)
- dvm3_Matrix operator+ (dvm3_Matrix const &m, double d)
- dvm3_Matrix operator- (dvm3_Matrix const &m, double d)
- dvm3_Matrix operator∗ (dvm3_Matrix const &m, double d)
- dvm3_Matrix operator/ (dvm3_Matrix const &m, double d)

8.1.1 Detailed Description

A class representing 3x3 matrix of doubles with common numerical operations ∗ defined. The dvm3_Matrix class is a simple class to handle common ∗ numerical operations on 3-matrices of doubles. Unless otherwise noted, ∗ the operations are component by component operations, ∗ e.g.,

```
v1 * v2 = { v1[0] * v2[0], v1[1] * v2[1], v1[2] * v2[2] }
```

This is intended as a small component which can be inexpensively created on the stack; the constructors and destructor do not make use of dynamic allocation. The dvm3_Matrix default constructor does not initialize the memory.

dvm3_Matrix works on any of these representations of 3-vectors:

- dvm3_Vector: a class encapsulating a 3-vector of doubles with associated mathematical operations
- double∗: a c-style array holding (at least) 3 doubles.

NOTE: although dvm3_Matrix works with either of the above two vector types, the SAME vector type must be used for ALL occurrences within any given method call.

Definition at line 104 of file dvm3_matrix.h.

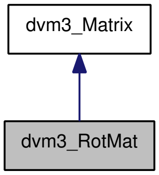
The documentation for this class was generated from the following files:

- dvm3_matrix.h
- dvm3_matrix.cc

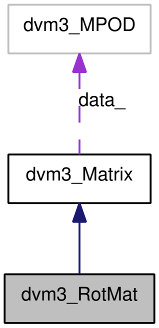
8.2 dvm3_RotMat Class Reference

`#include <dvm3/dvm3_rotmat.h>`

Inheritance diagram for dvm3_RotMat:



Collaboration diagram for dvm3_RotMat:



Public Types

- enum **EEulerAngleType** { **EOSAC_az_el_clock** = 0, **EX_convention** = 1, **EY_convention** = 2 }
- enum **EInputVecPair** {
  E_XY, **E_YZ**, **E_ZX**, **E_XZ**,
  E_ZY, **E_YX** }

Public Member Functions

- ~dvm3_RotMat ()
- dvm3_RotMat ()
- dvm3_RotMat (dvm3_RotMat const &m)
- dvm3_RotMat (dvm3_Vector const &x, dvm3_Vector const &y, dvm3_Vector const &z, double const tol=100.0 ∗DBL_EPSILON)
- dvm3_RotMat (double const x[ ], double const y[ ], double const z[ ], double const tol=100.0 ∗DBL_EPSILON)
- dvm3_RotMat (double angle0, double angle1, double angle2, EEulerAngleType angle_type=EOSAC_az_el_clock)
- dvm3_RotMat (dvm3_Vector const &v0, dvm3_Vector const &v1, EInputVecPair type)
- dvm3_RotMat (double const v0[ ], double const v1[ ], EInputVecPair type)
- void init (double angle0, double angle1, double angle2, EEulerAngleType angle_type=EOSAC_az_el_clock)
- void init (dvm3_Vector const &v0, dvm3_Vector const &v1, EInputVecPair type)
- void init (double const v0[ ], double const v1[ ], EInputVecPair type)
- dvm3_RotMat & operator= (dvm3_RotMat const &rhs)
- void invert ()
- int is_rotation_matrix (double const tol=100.0 ∗DBL_EPSILON) const

8.2.1 Detailed Description

A class representing 3x3 rotation matrix of doubles. dvm3_RotMat is a derived class of dvm3_Matrix. Constructors and initialization methods are added to construct orthogonal 3x3 matrices.

This is intended as a small component which can be inexpensively created on the stack; the constructors and destructor do not make use of dynamic allocation. The dvm3_RotMat default constructor initializes the matrix to a unit matrix (i.e., no rotation).

Suppose we are considering two frames, an initial standard frame, S, and a body frame, B. Let $\mathbf{e}_{S,i}$, i = 0, 1, 2, denote the i'th basis vector for frame S, and $\mathbf{e}_{B,i}$, i = 0, 1, 2, the corresponding basis vectors in frame B. The rotation matrix from S to B can be written as

$$\mathbf{R}_{i,j} = \mathbf{e}_{B,i} \cdot \mathbf{e}_{S,j}$$

Note that this has two particularly simple interpretations:

1. the column vectors of **R** consist of the direction cosines of the $\mathbf{e}_S$ vectors in the $\mathbf{e}_B$ frame
2. the row vectors consist of the direction cosines of the $\mathbf{e}_B$ vectors in the $\mathbf{e}_S$ frame.

Definition at line 69 of file dvm3_rotmat.h.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 dvm3_RotMat::~dvm3_RotMat () `[inline]`

Do-nothing destructor

Definition at line 527 of file dvm3_rotmat.h.

8.2.2.2 dvm3_RotMat::dvm3_RotMat ()

Default constructor; a unit matrix is generated by default.

Definition at line 44 of file dvm3_rotmat.cc.

8.2.2.3 dvm3_RotMat::dvm3_RotMat (dvm3_RotMat const & *m*) `[inline]`

Copy constructor.

Parameters:

m rotation matrix to copy

Definition at line 532 of file dvm3_rotmat.h.

8.2.2.4 dvm3_RotMat::dvm3_RotMat (dvm3_Vector const & *x*, dvm3_Vector const & *y*, dvm3_Vector const & *z*, double const *tol* = `100.0 * DBL_EPSILON`)

Constructor.

Each row is a coordinate unit vector. The vectors are assumed to be orthonormal; an error is issued and exit(1) is called if the vectors are not orthonormal to within tolerance tol. If the vectors are indeed orthonormal within tolerance tol, they are further explicitly orthonormalized by a series of cross products and scalings; consequently the matrix rows may differ slightly from the input x, y, z dvm3_Vector's.

Parameters:

- ***x*** x row
- ***y*** y row
- ***z*** z row
- ***tol*** tolerance for orthonormality

Definition at line 53 of file dvm3_rotmat.cc.

References is_rotation_matrix(), and dvm3_Matrix::orthonormalize().

8.2.2.5 dvm3_RotMat::dvm3_RotMat (double const *x*[ ], double const *y*[ ], double const *z*[ ], double const *tol* = `100.0 * DBL_EPSILON`)

Constructor.

Each row is a coordinate unit vector. The vectors are assumed to be orthonormal; an error is issued and exit(1) is called if the vectors are not orthonormal to within tolerance tol. If the vectors are indeed orthonormal within tolerance tol, they are further explicitly orthonormalized by a series of cross products and scalings; consequently the matrix rows may differ slightly from the input x, y, z dvm3_Vector's.

Parameters:

- ***x*** x row
- ***y*** y row
- ***z*** z row
- ***tol*** tolerance for orthonormality

Definition at line 76 of file dvm3_rotmat.cc.

References is_rotation_matrix(), and dvm3_Matrix::orthonormalize().

8.2.2.6 dvm3_RotMat::dvm3_RotMat (double *angle0*, double *angle1*, double *angle2*, dvm3_RotMat::EEulerAngleType *angle_type* = `EOSAC_az_el_clock`)

Constructor.

Construct a rotation matrix based on 3 Euler angles (in radians). This rotation matrix describes the rotation *from* an external system coordinate *to* a coordinate system attached to a rigid body.

A body initially has its coordinate system aligned with the standard external coordinates. The body is rotated to its final orientation by performing a series of rotations, each rotation about a current body axis. That is

- rotation by angle0 about axis a, followed by
- rotation by angle1 about axis b, followed by
- rotation by angle2 about axis c

The specification of angle0, angle1, and angle2 and axes a, b, and c are determined by the enumerated constant `angle_type`.

Because we want a transformation *from* the space coordinates *to* the body coordinates, the rotation matrix components actually correspond to the inverse of the transformation which generated the rigid body position, i.e.,

- rotation by -angle2 about axis c, followed by
- rotation by -angle1 about axis b, followed by
- rotation by -angle0 about axis a

Because of rotation matrices are orthogonal, the inverse motion (from the body-centered coordinate system back to the standard coordinate system) is simply the transpose of the rotation matrix constructed by this constructor.

Parameters:

angle0 1st Euler angle

angle1 2nd Euler angle

angle2 3rd Euler angle

angle_type type of Euler angles to use

Definition at line 111 of file dvm3_rotmat.cc.

References init().

8.2.2.7 dvm3_RotMat::dvm3_RotMat (dvm3_Vector const & *v0*, dvm3_Vector const & *v1*, EInputVecPair *type*)

Constructor.

Construct a rotation matrix based on two (non-colinear) vectors. The rotation matrix is constructed by sequential application cross products.

If type == E_XY, the vectors passed in are assumed to lie in the x-y plane with v0 an x-like vector and v1 a y-like vector; e_x is v0. e_z is constructed from e_x cross v1. e_y is constructed from e_z cross e_x. The three vectors e_x, e_y, and e_z are normalized.

If type == E_YZ, the vectors passed in are assumed to lie in the y-z plane with v0 an y-like vector and v1 a z-like vector; e_y is v0. e_x is constructed from e_y cross v1. e_z is constructed from e_x cross e_y. The three vectors e_x, e_y, and e_z are normalized.

If type == E_ZX, the vectors passed in are assumed to lie in the x-z plane with v0 an z-like vector and v1 a x-like vector; e_z is v0. e_y is constructed from e_z cross v1. e_x is constructed from e_y cross e_z. The three vectors e_x, e_y, and e_z are normalized.

If type == E_YX, the vectors passed in are assumed to lie in the x-y plane with v0 a y-like vector and v1 an x-like vector; e_y is v0. e_z is constructed from v1 cross e_y. e_x is constructed from e_y cross e_z. The three vectors e_x, e_y, and e_z are normalized.

If type == E_ZY, the vectors passed in are assumed to lie in the y-z plane with v0 a z-like vector and v1 a y-like vector; e_z is v0. e_x is constructed from v1 cross e_z. e_y is constructed from e_z cross e_x. The three vectors e_x, e_y, and e_z are normalized.

If type == E_XZ, the vectors passed in are assumed to lie in the x-z plane with v0 an x-like vector and v1 a z-like vector; e_x is v0. e_y is constructed from v1 cross e_x. e_z is constructed from e_x cross e_y. The three vectors e_x, e_y, and e_z are normalized.

Parameters:

v0	1st vector
v1	2nd vector
type	type of input vectors

Definition at line 160 of file dvm3_rotmat.cc.

References init().

8.2.2.8 dvm3_RotMat::dvm3_RotMat (double const *v0*[ ], double const *v1*[ ], EInputVecPair *type*)

Constructor.

Construct a rotation matrix based on two (non-colinear) vectors. The rotation matrix is constructed by sequential application cross products.

If type == E_XY, the vectors passed in are assumed to lie in the x-y plane with v0 an x-like vector and v1 a y-like vector; e_x is v0. e_z is constructed from e_x cross v1. e_y is constructed from e_z cross e_x. The three vectors e_x, e_y, and e_z are normalized.

If type == E_YZ, the vectors passed in are assumed to lie in the y-z plane with v0 an y-like vector and v1 a z-like vector; e_y is v0. e_x is constructed from e_y cross v1. e_z is constructed from e_x cross e_y. The three vectors e_x, e_y, and e_z are normalized.

If type == E_ZX, the vectors passed in are assumed to lie in the x-z plane with v0 an z-like vector and v1 a x-like vector; e_z is v0. e_y is constructed from e_z cross v1. e_x is constructed from e_y cross e_z. The three vectors e_x, e_y, and e_z are normalized.

If type == E_YX, the vectors passed in are assumed to lie in the x-y plane with v0 a y-like vector and v1 an x-like vector; e_y is v0. e_z is constructed from v1 cross e_y. e_x is constructed from e_y cross e_z. The three vectors e_x, e_y, and e_z are normalized.

If type == E_ZY, the vectors passed in are assumed to lie in the y-z plane with v0 a z-like vector and v1 a y-like vector; e_z is v0. e_x is constructed from v1 cross e_z. e_y is constructed from e_z cross e_x. The three vectors e_x, e_y, and e_z are normalized.

If type == E_XZ, the vectors passed in are assumed to lie in the x-z plane with v0 an x-like vector and v1 a z-like vector; e_x is v0. e_y is constructed from v1 cross e_x. e_z is constructed from e_x cross e_y. The three vectors e_x, e_y, and e_z are normalized.

Parameters:

v0 1st vector

v1 2nd vector

type type of input vectors

Definition at line 211 of file dvm3_rotmat.cc.

References init().

8.2.3 Member Function Documentation

8.2.3.1 void dvm3_RotMat::init (double *angle0*, double *angle1*, double *angle2*, dvm3_RotMat::EEulerAngleType *angle_type* = `EOSAC_az_el_clock`)

Initialize a rotation matrix based on 3 Euler angles (in radians). This rotation matrix describes the rotation *from* an external system coordinate *to* a coordinate system attached to a rigid body.

A body initially has its coordinate system aligned with the standard external coordinates. The body is rotated to its final orientation by performing a series of rotations, each rotation about a current body axis. That is

- rotation by angle0 about axis a, followed by
- rotation by angle1 about axis b, followed by
- rotation by angle2 about axis c

The specification of angle0, angle1, and angle2 and axes a, b, and c are determined by the enumerated constant `angle_type.`

`Because we want a transformation` *`from`* `the space coordinates` *`to`* `the body coordinates, the rotation matrix components actually correspond to the inverse of the transformation which generated the rigid body position, i.e.,`

- `rotation by -angle2 about axis c, followed by`
- `rotation by -angle1 about axis b, followed by`
- `rotation by -angle0 about axis a`

`Because of rotation matrices are orthogonal, the inverse motion (from the body-centered coordinate system back to the standard coordinate system) is simply the transpose of the rotation matrix constructed by this constructor.`

`Parameters:`

angle0 `1st Euler angle`

angle1 `2nd Euler angle`

angle2 `3rd Euler angle`

angle_type `type of Euler angles to use`

Definition at line 237 of file dvm3_rotmat.cc.

Referenced by dvm3_RotMat().

8.2.3.2 void dvm3_RotMat::init (dvm3_Vector const & *v0*, dvm3_Vector const & *v1*, EInputVecPair *type*)

Initialize a rotation matrix based on two (non-colinear) vectors. The rotation matrix is constructed by sequential application cross products.

If type == E_XY, the vectors passed in are assumed to lie in the x-y plane with v0 an x-like vector and v1 a y-like vector; e_x is v0. e_z is constructed from e_x cross v1. e_y is constructed from e_z cross e_x. The three vectors e_x, e_y, and e_z are normalized.

If type == E_YZ, the vectors passed in are assumed to lie in the y-z plane with v0 an y-like vector and v1 a z-like vector; e_y is v0. e_x is constructed from e_y cross v1. e_z is constructed from e_x cross e_y. The three vectors e_x, e_y, and e_z are normalized.

If type == E_ZX, the vectors passed in are assumed to lie in the x-z plane with v0 an z-like vector and v1 a x-like vector; e_z is v0. e_y is constructed from e_z cross v1. e_x is constructed from e_y cross e_z. The three vectors e_x, e_y, and e_z are normalized.

If type == E_YX, the vectors passed in are assumed to lie in the x-y plane with v0 a y-like vector and v1 an x-like vector; e_y is v0. e_z is constructed from v1 cross e_y. e_x is constructed from e_y cross e_z. The three vectors e_x, e_y, and e_z are normalized.

If type == E_ZY, the vectors passed in are assumed to lie in the y-z plane with v0 a z-like vector and v1 a y-like vector; e_z is v0. e_x is constructed from v1 cross e_z. e_y is constructed from e_z cross e_x. The three vectors e_x, e_y, and e_z are normalized.

If type == E_XZ, the vectors passed in are assumed to lie in the x-z plane with v0 an x-like vector and v1 a z-like vector; e_x is v0. e_y is constructed from v1 cross e_x. e_z is constructed from e_x cross e_y. The three vectors e_x, e_y, and e_z are normalized.

Parameters:

- ***v0*** 1st vector
- ***v1*** 2nd vector
- ***type*** type of input vectors

Definition at line 384 of file dvm3_rotmat.cc.

References dvm3_Vector::data_.

8.2.3.3 void dvm3_RotMat::init (double const *v0*[ ], double const *v1*[ ], EInputVecPair *type*)

Initialize a rotation matrix based on two (non-colinear) vectors. The rotation matrix is constructed by sequential application cross products.

If type == E_XY, the vectors passed in are assumed to lie in the x-y plane with v0 an x-like vector and v1 a y-like vector; e_x is v0. e_z is constructed from e_x cross v1. e_y is constructed from e_z cross e_x. The three vectors e_x, e_y, and e_z are normalized.

If type == E_YZ, the vectors passed in are assumed to lie in the y-z plane with v0 an y-like vector and v1 a z-like vector; e_y is v0. e_x is constructed from e_y cross v1. e_z is constructed from e_x cross e_y. The three vectors e_x, e_y, and e_z are normalized.

If type == E_ZX, the vectors passed in are assumed to lie in the x-z plane with v0 an z-like vector and v1 a x-like vector; e_z is v0. e_y is constructed from e_z cross v1. e_x is constructed from e_y cross e_z. The three vectors e_x, e_y, and e_z are normalized.

If type == E_YX, the vectors passed in are assumed to lie in the x-y plane with v0 a y-like vector and v1 an x-like vector; e_y is v0. e_z is constructed from v1 cross e_y. e_x is constructed from e_y cross e_z. The three vectors e_x, e_y, and e_z are normalized.

If type == E_ZY, the vectors passed in are assumed to lie in the y-z plane with v0 a z-like vector and v1 a y-like vector; e_z is v0. e_x is constructed from v1 cross e_z. e_y is constructed from e_z cross e_x. The three vectors e_x, e_y, and e_z are normalized.

If type == E_XZ, the vectors passed in are assumed to lie in the x-z plane with v0 an x-like vector and v1 a z-like vector; e_x is v0. e_y is constructed from v1 cross e_x. e_z is constructed from e_x cross e_y. The three vectors e_x, e_y, and e_z are normalized.

Parameters:

- ***v0*** 1st vector
- ***v1*** 2nd vector
- ***type*** type of input vectors

Definition at line 502 of file dvm3_rotmat.cc.

8.2.3.4 dvm3_RotMat & dvm3_RotMat::operator= (dvm3_RotMat const & *rhs*) `[inline]`

Copy-assignment operator

Returns:

rotation matrix

Parameters:

rhs value to be assigned.

Definition at line 538 of file dvm3_rotmat.h.

References dvm3_Matrix::data_.

8.2.3.5 void dvm3_RotMat::invert ()

Invert (i.e., transpose) this rotation matrix.

Definition at line 581 of file dvm3_rotmat.cc.

8.2.3.6 int dvm3_RotMat::is_rotation_matrix (double const *tol* = `100.0 * DBL_EPSILON`) const

Test this rotation matrix for orthonormality.

Returns:

1 if this is an orthonormal matrix, 0 otherwise.

Parameters:

tol tolerance for determining orthonormality

Definition at line 598 of file dvm3_rotmat.cc.

Referenced by dvm3_RotMat().

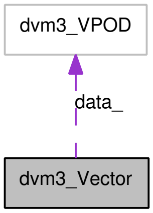
The documentation for this class was generated from the following files:

- dvm3_rotmat.h
- dvm3_rotmat.cc

8.3 dvm3_Vector Class Reference

`#include <dvm3/dvm3_vector.h>`

Collaboration diagram for dvm3_Vector:



Public Member Functions

- ~dvm3_Vector ()
- dvm3_Vector ()
- dvm3_Vector (double x, double y, double z)
- dvm3_Vector (dvm3_Vector const &other)
- void copy_from_cvec (double const ∗cv)
- void copy_to_cvec (double ∗v) const
- void init (double x, double y, double z)
- dvm3_Vector & operator= (dvm3_Vector const &rhs)
- dvm3_Vector & operator= (double rhs)
- dvm3_Vector & operator+= (dvm3_Vector const &rhs)
- dvm3_Vector & operator-= (dvm3_Vector const &rhs)
- dvm3_Vector & operator∗= (dvm3_Vector const &rhs)
- dvm3_Vector & operator/= (dvm3_Vector const &rhs)
- dvm3_Vector & operator+= (double rhs)
- dvm3_Vector & operator-= (double rhs)
- dvm3_Vector & operator∗= (double rhs)
- dvm3_Vector & operator/= (double rhs)
- dvm3_Vector & operator+ ()
- dvm3_Vector & operator- ()
- double const & operator[ ] (int i) const
- double & operator[ ] (int i)
- double operator() (int i) const
- double & operator() (int i)
- double dot (dvm3_Vector const &v) const
- void cross (dvm3_Vector const &v1, dvm3_Vector const &v2)
- void lincomb (double c1, dvm3_Vector const &v1, double c2, dvm3_Vector const &v2)
- std::ostream & print_on (std::ostream &os, char const pre[ ]="", char const post[ ]="") const
- void cprint_on (FILE ∗of, char const prefix[ ]="", char const postfix[ ]="")

- double unitize ()
- int is_unit_vector (double const tol=100.0 ∗DBL_EPSILON) const
- int is_orthogonal_to (dvm3_Vector const &other, double const tol=100.0 ∗DBL_EPSILON) const

Friends

- class dvm3_Matrix
- class dvm3_RotMat
- dvm3_Vector operator+ (dvm3_Vector const &v1, dvm3_Vector const &v2)
- dvm3_Vector operator- (dvm3_Vector const &v1, dvm3_Vector const &v2)
- dvm3_Vector operator∗ (dvm3_Vector const &v1, dvm3_Vector const &v2)
- dvm3_Vector **operator/** (dvm3_Vector const &v1, dvm3_Vector const &v2)
- dvm3_Vector operator+ (double r, dvm3_Vector const &v)
- dvm3_Vector operator- (double r, dvm3_Vector const &v)
- dvm3_Vector operator∗ (double r, dvm3_Vector const &v)
- dvm3_Vector operator/ (double r, dvm3_Vector const &v)
- dvm3_Vector operator+ (dvm3_Vector const &v, double r)
- dvm3_Vector operator- (dvm3_Vector const &v, double r)
- dvm3_Vector operator∗ (dvm3_Vector const &v, double r)
- dvm3_Vector operator/ (dvm3_Vector const &v, double r)
- std::ostream & operator<< (std::ostream &os, dvm3_Vector const &)

  - double unitize (dvm3_Vector &vu, dvm3_Vector const &vi)

  - double dot (dvm3_Vector const &v1, dvm3_Vector const &v2)
  - void cross (dvm3_Vector &result, dvm3_Vector const &v1, dvm3_Vector const &v2)

  - void lincomb (dvm3_Vector &result, double c1, dvm3_Vector const &v1, double c2, dvm3_Vector const &v2)

  - void dyad_product (dvm3_Matrix &, const dvm3_Vector &, const dvm3_Vector &)
  - void mvmult (dvm3_Vector &, const dvm3_Matrix &, const dvm3_Vector &)
  - void mtvmult (dvm3_Vector &, const dvm3_Matrix &, const dvm3_Vector &)

8.3.1 Detailed Description

A class representing a 3-vector of doubles with common numerical operations defined. The dvm3_Vector class is a simple class to handle common numerical operations on 3-vectors of doubles. Unless otherwise noted, the operations are component by component, e.g.,

```
    v1 * v2 = [v1(0) *v2(0), v1(1)*v2(1), v1(2)*v2(2)]
 *
```

This is intended as a small component which can be inexpensively created on the stack; the constructors and destructor do not make use of dynamic allocation. The dvm3_Vector default constructor does not initialize the memory.

Note that dvm3_Vector inherits from a struct, dvm3_VPOD, which encapulates an array of doubles. The inheritence is protected; this implementation is exposed purely for efficiency considerations; users shall not assume that this implementation detail will not change in future. DO NOT rely on the fact that there is an array under the hood or on the naming of the data members within dvm3_VPOD.

Definition at line 106 of file dvm3_vector.h.

8.3.2 Member Function Documentation

8.3.2.1 double dvm3_Vector::unitize ()

normalize this dvm3_Vector; return original length vector length.

Returns:

original length of this vector

8.3.2.2 int dvm3_Vector::is_unit_vector (double const *tol* = `100.0 *DBL_EPSILON`) const

normalize this dvm3_Vector; return original length vector length.

Returns:

1 if vector is unit vector within tolerance tol.

8.3.2.3 int dvm3_Vector::is_orthogonal_to (dvm3_Vector const & *other*, double const *tol* = `100.0 *DBL_EPSILON`) const

Test whether this vector and the other vector are orthogonal within tolerance tol.

Returns:

1 if this and other are orthogonal within tolerance tol.

Parameters:

tol tolerance for determining orthogonality

8.3.3 Friends And Related Function Documentation

8.3.3.1 double unitize (dvm3_Vector & *vu*, dvm3_Vector const & *vi*) `[friend]`

Normalize a vector

Returns:

original length of vi.

Parameters:

vu normalized version of vi

vi vector

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

- dvm3_vector.h

Index