Vector

Generated by Doxygen 1.8.5

Tue Jan 28 2014 11:43:23

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

Class Documentation

2.1 Matrix Class Reference

A class that stores 4x4 matrices.

`#include <vector.hh>`

Public Member Functions

- Matrix ()

  Constructs a new unity matrix.
- Matrix (const Matrix &original)

  Constructs a new matrix by copying its elements from another one.
- ∼Matrix ()

  Destructs the matrix.
- Matrix & operator= (const Matrix &original)

  Copies the elements from another matrix.
- double & operator() (const int row_ix, const int col_ix)

  Returns a reference to an element of the matrix.
- double operator() (const int row_ix, const int col_ix) const

  Returns the value of an element of the matrix.
- Matrix & operator∗= (const Matrix &rhs)

  Multiplies a matrix with this matrix.
- void inv ()

  Inverts this matrix.
- void print (std::ostream &output_stream, const int elt_width=8) const

  Prints a human-readable representation of the matrix.

Static Public Member Functions

- static Matrix inv (Matrix matrix)

  Inverts a matrix.

Friends

- class Vector3D

  Vector3D is our friend so it can multiply itself with us.

2.1.1 Detailed Description

A class that stores 4x4 matrices.

This class can be used to represent transformations of vectors.

2.1.2 Member Function Documentation

2.1.2.1 Matrix Matrix::inv (Matrix *matrix*) [static]

Inverts a matrix.

Parameters

matrix	The matrix that is inverted.

Returns

The inverted matrix.

2.1.2.2 double & Matrix::operator() (const int *row_ix,* const int *col_ix*)

Returns a reference to an element of the matrix.

Parameters

row_ix	The row index of the referenced element, starting from 1.
col_ix	The column index of the referenced element, starting from 1.

Returns

A reference to the referenced element.

2.1.2.3 double Matrix::operator() (const int *row_ix,* const int *col_ix*) const

Returns the value of an element of the matrix.

Parameters

row_ix	The row index of the referenced element, starting from 1.
col_ix	The column index of the referenced element, starting from 1.

Returns

The value of the referenced element.

2.1.2.4 Matrix & Matrix::operator*= (const Matrix & *rhs*)

Multiplies a matrix with this matrix.

Parameters

rhs	The matrix with which this matrix is multiplied.

Returns

A reference to this matrix.

2.1.2.5 Matrix & Matrix::operator= (const Matrix & *original*)

Copies the elements from another matrix.

Parameters

original	The matrix whose elements are copied.

Returns

A reference to this matrix.

2.1.2.6 void Matrix::print (std::ostream & *output_stream*, const int *elt_width* = 8) const

Prints a human-readable representation of the matrix.

This method does not print a trailing newline.

Parameters

output_stream	The output stream to which the matrix is printed.
elt_width	The amount of space that is reserved to print an element.

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

- vector.hh
- vector.cc

2.2 Vector3D Class Reference

A class that represents 3D vectors.

```
#include <vector.hh>
```

Public Member Functions

- Vector3D ()

  Constructs a new Vector3D object that represents the origin.
- Vector3D (const Vector3D &original)

  Constructs a new Vector3D object by copying another one.
- ∼Vector3D ()

  Destructs a vector.
- bool is_point () const

  Returns whether this object represents a point.
- bool is_vector () const

  Returns whether this object represents a vector.
- Vector3D & operator= (const Vector3D &original)

  Assignment operator.
- Vector3D & operator+= (const Vector3D &rhs)

  Adds another Vector3D object to this one.
- Vector3D & operator-= (const Vector3D &rhs)

  Subtracts another Vector3D object from this one.
- Vector3D & operator∗= (const double rhs)

  Multiplies a scalar with this vector or point.
- Vector3D & operator∗= (const Matrix &rhs)

  Applies a transformation.
- double dot (const Vector3D &rhs) const

Calculates the dot-product of this vector and another one.
- Vector3D & cross_equals (const Vector3D &rhs)

  Calculates the cross-product of this vector and another one.
- double length () const

  Determines the length of the vector.
- void normalise ()

  Normalises the vector.
- void print (std::ostream &output_stream, const int elt_width=8) const

  Prints a human-readable representation of the vector.

Static Public Member Functions

- static Vector3D point (const double x, const double y, const double z)

  Constructs a new Vector3D object that represents a point.
- static Vector3D point (const Vector3D &original)

  Constructs a new Vector3D object that represents a point.
- static Vector3D vector (const double x, const double y, const double z)

  Constructs a new Vector3D object that represents a vector.
- static Vector3D vector (const Vector3D &original)

  Constructs a new Vector3D object that represents a vector.
- static double dot (const Vector3D &lhs, const Vector3D &rhs)

  Calculates the dot-product of two vectors.
- static Vector3D cross (Vector3D lhs, const Vector3D &rhs)

  Calculates the cross-product of two vectors.
- static Vector3D normalise (Vector3D arg)

  Normalises a vector.

Public Attributes

- double x

  The x-coordinate of the vector.
- double y

  The y-coordinate of the vector.
- double z

  The z-coordinate of the vector.

Protected Member Functions

- Vector3D (const double x_init, const double y_init, const double z_init, const bool infty_init)

  Constructs a new Vector3D object given its coordinates.
- Vector3D (const Vector3D &original, const bool infty_init)

  Consructs a new Vector3D object by copying another one.

2.2.1 Detailed Description

A class that represents 3D vectors.

This class can both represent points and directions. A point can be constructed using the Vector3D::point pseudo-constructor. A vector can be constructed using the Vector3D::vector pseudo-constructor. Transforming a vector will behave accordingly.

2.2.2 Constructor & Destructor Documentation

2.2.2.1 Vector3D::Vector3D (const double *x_init*, const double *y_init*, const double *z_init*, const bool *infty_init*) [protected]

Constructs a new Vector3D object given its coordinates.

This constructor is made protected to avoid it to be called directly. In order to construct a new instance of this class the Vector3D::point or Vector3D::vector pseudo-constructors should be used.

Parameters

x_init	The x-coordinate.
y_init	The y-coordinate.
z_init	The z-coordinate.
infty_init	`false` if the vector represents a point, `true` if it represents a vector.

2.2.2.2 Vector3D::Vector3D (const Vector3D & *original*, const bool *infty_init*) [protected]

Consructs a new Vector3D object by copying another one.

This constructor is made protected to avoid it to be called direcly. In order to construct a new instance of this class the Vector3D::point or Vector3D::vector pseudo-constructors should be used.

Parameters

original	The vector that is copied.
infty_init	`false` if the vector represents a point, `true` if it represents a vector.

2.2.2.3 Vector3D::Vector3D (const Vector3D & *original*)

Constructs a new Vector3D object by copying another one.

Parameters

original	The vector that is copied.

2.2.3 Member Function Documentation

2.2.3.1 Vector3D Vector3D::cross (Vector3D *lhs*, const Vector3D & *rhs*) [static]

Calculates the cross-product of two vectors.

Parameters

lhs	The left factor.
rhs	The right factor.

Returns

The cross-product of `lhs` and `rhs`.

2.2.3.2 Vector3D & Vector3D::cross_equals (const Vector3D & *rhs*)

Calculates the cross-product of this vector and another one.

This operation will always succeed regardless of whether the operands represent points or directions. In case this operation is applied to a point it will be treated as a vector from the origin to the point. Note that performing the dot product on points does not make much sense.

Parameters

rhs	The vector the is multiplied with this vector.

Returns

A reference to this vector.

2.2.3.3 double Vector3D::dot (const Vector3D & *rhs*) const

Calculates the dot-product of this vector and another one.

This operation will always succeed regardless of whether the operands represent points or vectors. In case this operation is applied to a point it will be treated as a vector from the origin to the point. Note that performing the dot product on points does not make much sense.

Parameters

rhs	The Vector3D object to be multiplied with this.

Returns

The dot product of this vector and `rhs`.

2.2.3.4 double Vector3D::dot (const Vector3D & *lhs,* const Vector3D & *rhs*) [static]

Calculates the dot-product of two vectors.

Parameters

lhs	The left factor.
rhs	The right factor.

Returns

The dot-product of `lhs` and `rhs`.

2.2.3.5 bool Vector3D::is_point ( ) const

Returns whether this object represents a point.

Returns

`true` if this object represents a point, `false` otherwise.

2.2.3.6 bool Vector3D::is_vector ( ) const

Returns whether this object represents a vector.

Returns

`true` if this object represents a vector, `false` otherwise.

2.2.3.7 double Vector3D::length ( ) const

Determines the length of the vector.

In case the vector represents a point, the distance between the point and the origin is returned.

Returns

The length of the vector

2.2.3.8 void Vector3D::normalise ( )

Normalises the vector.

This operation scales the vector such that it has a length of 1. If the vector represents a point the point is translated along the line that connects it to the origin such that the distance between it and the origin is 1.

2.2.3.9 Vector3D Vector3D::normalise (Vector3D *arg*) [static]

Normalises a vector.

This function uses Vector3D::normalise to normalise a vector.

Parameters

arg	The vector that is normalised.

Returns

The normalised vector.

2.2.3.10 Vector3D & Vector3D::operator∗= (const double *rhs*)

Multiplies a scalar with this vector or point.

Parameters

rhs	The scalar that is multiplied with this object.

Returns

A reference to this vector.

2.2.3.11 Vector3D & Vector3D::operator∗= (const Matrix & *rhs*)

Applies a transformation.

Please note that before the transformation is actually performed, assertions are used to make sure that passed Matrix objects represents a VALID transformation. To this end, the last column of the matrix MUST equal: (0) (0) (0) (1) That is the matrix itself must be of the form: (a, b, c, 0) (d, e, f, 0) (g, h, i, 0) (j, k, l, 1)

Parameters

rhs	The matrix that is multiplied with this Vector3D object.

Returns

A reference to this object.

2.2.3.12 Vector3D & Vector3D::operator+= (const Vector3D & *rhs*)

Adds another Vector3D object to this one.

If both objects represent vectors the result will also be a vector. Otherwise the result is a point.

Parameters

rhs	The vector that is added to this vector.

Returns

A reference to this vector.

2.2.3.13 Vector3D & Vector3D::operator-= (const Vector3D & *rhs*)

Subtracts another Vector3D object from this one.

Subtracting a vector from a point or a point from a vector will result in a point. Subtracting two vectors or two points will result in a vector.

Parameters

rhs	The Vector3D object that is subtracted from this one.

Returns

A reference to this vector.

2.2.3.14 Vector3D & Vector3D::operator= (const Vector3D & *original*)

Assignment operator.

Parameters

original	The vector that is copied.

Returns

A reference to this vector.

2.2.3.15 Vector3D Vector3D::point (const double *x*, const double *y*, const double *z*) [static]

Constructs a new Vector3D object that represents a point.

Parameters

x	The x-coordinate.
y	The y-coordinate.
z	The z-coordinate.

2.2.3.16 Vector3D Vector3D::point (const Vector3D & *original*) [static]

Constructs a new Vector3D object that represents a point.

Parameters

original	The vector whose coordinates are copied.

2.2.3.17 void Vector3D::print (std::ostream & *output_stream*, const int *elt_width* = 8) const

Prints a human-readable representation of the vector.

Parameters

output_stream	The output stream to which the vector is printed.
elt_width	The amount of space that is reserved to print an element.

2.2.3.18 Vector3D Vector3D::vector (const double *x*, const double *y*, const double *z*) [static]

Constructs a new Vector3D object that represents a vector.

Parameters

x	The x-coordinate.
y	The y-coordinate.
z	The z-coordinate.

2.2.3.19 Vector3D Vector3D::vector (const Vector3D & *original*) [static]

Constructs a new Vector3D object that represents a vector.

Parameters

original	The vector whose coordinates are copied.

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

- vector.hh
- vector.cc

Index