Vector3d.h

3-dimensional vector extension for C++ vector class library

Agner Fog

Chapter 1

Introduction

3-dimensional vectors are useful in geometry and physics. The file vector3d.h provides vector classes, operators, and functions for calculations with 3-D vectors. This is an extension to the Vector Class Library.

The classes listed below are defined. Common operators and functions are defined for these classes:

Table 1.1: 3-D vector classes

vector class	Precision	3-D vectors per instance	Corresponding real vector class	Total bits	Recommended minimum instruction set
Vec3Df	single	1	Vec4f	128	SSE2
Vec3Dd	double	1	Vec4d	256	AVX

1.1 Compiling

The 3-D vector class extension to the Vector Class Library is compiled in the same way as the Vector Class Library itself. All x86 and x86-64 platforms are supported, including Windows, Linux, and Mac OS. The following C++ compilers can be used: Gnu, Clang, Microsoft, and Intel. See the Vector Class Library manual for further details.

This example shows how to use the 3-D vector classes:

Example 1.1.

```
// Example for 3-D vectors
#include <stdio.h>
#include "vectorclass.h"  // vector class library
#include "vector3d.h"     // extension for 3-D vectors

// function to print 3-D vector:
template <typename V>
void printv3 (const char * text, V a) {
    auto aa = a.to_vector();  // get elements as real vector
    printf("\n%s ", text);    // print text
    printf("(%.3G,%.3G,%.3G)", aa[0], aa[1], aa[2]);
}

int main() {
    // define 3-D vectors
```

```
    Vec3Dd a(1,2,3);                    // x = 1, y = 2, z = 3
    Vec3Dd b(4,5,6);                    // x = 4, y = 5, z = 6
    Vec3Dd c = a + b;                   // add vectors
    Vec3Dd d = cross_product(a, b); // x-product
    double e = dot_product(a, b);    // dot-product
    // print results
    printv3("a = ", a);                 // a = (1,2,3)
    printv3("b = ", b);                 // b = (4,5,6)
    printv3("c = ", c);                 // c = (5,7,9)
    printv3("d = ", d);                 // d = (-3,6,-3)
    printf ("\ne = %f", e);             // e = 32
}
```

Chapter 2

Constructing 3-D vectors and loading data

There are several ways to create 3-D vectors and put data into them. These methods are listed here.

Method	default constructor
Defined for	all 3-D vectors classes
Description	the 3-D vector is created but not initialized. The value is unpredictable
Efficiency	good

```
// Example:
Vec3Dd a;      // creates a 3-D vector
```

Method	Construct from x,y,z coordinates
Defined for	all 3-D vectors classes
Description	The parameters define the x, y, and z coordinates
Efficiency	good

```
// Example:
Vec3Dd a(1,2,3);   // a = (1,2,3) (x = 1, y = 2, z = 3)
```

Method	member function load(p)
Defined for	all 3-D vectors classes
Description	Load data from array of same precision.
Efficiency	good

```
// Example:
float a[3] = {2,5,-1};
Vec3Df b;
b.load(a);   // b = (2,5,-1)
```

Method	member function store(p)
Defined for	all 3-D vectors classes
Description	Save data into array of same precision
Efficiency	good

```
// Example:
```

```
double a[3];
Vec3Dd b(4,0,3);
b.store(a);   // a = {4,0,3}
```

Method	member function get_x()
Defined for	all 3-D vectors classes
Description	Get the x-coordinate
Efficiency	good

```
// Example:
Vec3Dd a(1,2,3);
double b = a.get_x();   // b = 1
```

Method	member function get_y()
Defined for	all 3-D vectors classes
Description	Get the y-coordinate
Efficiency	good

```
// Example:
Vec3Dd a(1,2,3);
double b = a.get_y();   // b = 2
```

Method	member function get_z()
Defined for	all 3-D vectors classes
Description	Get the z-coordinate
Efficiency	good

```
// Example:
Vec3Dd a(1,2,3);
double b = a.get_z();   // b = 3
```

Method	member function extract(index)
Defined for	all 3-D vectors classes
Description	index = 0, 1, 2 give the x, y, or z-coordinate, respectively
Efficiency	good

```
// Example:
Vec3Dd a(1,2,3);
double b = a.extract(2);   // b = 3
double c = a[2];           // b = 3 (the same)
```

Method	member function insert(index, value)
Defined for	all 3-D vectors classes
Description	index = 0, 1, 2 changes the x, y, or z-coordinate, respectively
Efficiency	good

```
// Example:
Vec3Dd a(1,2,3);
a.insert(0, 8);   // a = (8, 2, 3)
```

Chapter 3

Operators

Operator	+
Defined for	all 3-D vectors classes
Description	Add two vectors
Efficiency	good

```
// Example:
Vec3Dd a(1,2,3);
Vec3Dd b(5,6,7);
Vec3Dd c = a + b;      // c = (6,8,10)
```

Operator	-
Defined for	all 3-D vectors classes
Description	Subtract two vectors
Efficiency	good

```
// Example:
Vec3Dd a(11,10,9);
Vec3Dd b(5,6,7);
Vec3Dd c = a - b;      // c = (6,4,2)
Vec3Dd d = - b;        // d = (-5,-6,-7)
```

Operator	*
Defined for	all 3-D vectors classes
Description	Multiply two vectors element by element, or one vector and one scalar of the same precision
Efficiency	good

```
// Example:
Vec3Dd a(1,2,3);
Vec3Dd b(4,5,6);
Vec3Dd c = a * b;      // c = (4,10,18)
Vec3Dd d = a * 10.0;   // d = (10,20,30)
```

Operator	/
Defined for	all 3-D vectors classes
Description	Divide a vector by a scalar of the same precision
Efficiency	good

```
// Example:
Vec3Dd a(10,20,30);
Vec3Dd b = a / 5.0; // b = (2,4,6)
```

Operator	==
Defined for	all 3-D vectors classes
Description	Compare for equality. The result is a boolean scalar.
Efficiency	good

```
// Example:
Vec3Dd a(1, 2,3);
Vec3Dd b(1,-2,3);
bool   c = (a == b);  // c = false
```

Operator	!=
Defined for	all 3-D vectors classes
Description	Compare for not equal. The result is a boolean scalar.
Efficiency	good

```
// Example:
Vec3Dd a(1, 2,3);
Vec3Dd b(1,-2,3);
bool   c = (a != b);  // c = true
```

Chapter 4

Mathematical functions

Function	cross_product
Defined for	all 3-D vectors classes
Description	Gives the X-product of two vectors
Efficiency	medium
Accuracy	Calculation of the X-product involves the calculation of sums of products. Loss of precision may occur if the result is close to zero.

```
// Example:
Vec3Dd a(1,2,3);
Vec3Dd b(4,5,6);
Vec3Dd c = cross_product(a,b); // c = (-3,6,-3)
Vec3Dd d = cross_product(b,a); // d = (3,-6,3)
```

Function	dot_product
Defined for	all 3-D vectors classes
Description	Gives the dot-product of two vectors. The result is a scalar
Efficiency	medium

```
// Example:
Vec3Dd a(1,2,3);
Vec3Dd b(4,5,6);
double c = dot_product(a,b); // c = 32
```

Function	vector_length
Defined for	all 3-D vectors classes
Description	Gives the length of the vector (Euclidian norm)
Efficiency	medium

```
// Example:
Vec3Dd a(3,0,4);
double b = vector_length(a); // b = 5
```

Function	normalize_vector
Defined for	all 3-D vectors classes
Description	Divides the vector by its length to give a vector with the same direction and length one.
Efficiency	medium

```
// Example:
Vec3Dd a(3,0,4);
Vec3Dd b = normalize_vector(a); // b = (0.6, 0.0, 0.8)
```

Function	rotate
Defined for	all 3-D vectors classes
Description	Rotates a vector by multiplying a 3x3 rotation matrix by the column vector. The first three parameters define the columns of the rotation matrix. The last parameter is the vector to rotate.
Efficiency	medium
Accuracy	Calculation of the rotated vector involves the calculation of sums of products. Loss of precision may occur if the result is close to zero.

```
// Example:
Vec3Dd a(1,2,3);     // vector to rotate
Vec3Dd c0(1,0,0);    // first column of matrix
Vec3Dd c1(0,0,-1);   // second column of matrix
Vec3Dd c2(0,1,0);    // third column of matrix
Vec3Dd d = rotate(c0,c1,c2,a); // d = (1,3,-2)
```

Chapter 5

Other functions

Function	to_vector
Defined for	all 3-D vectors classes
Description	Convert to a vector of class Vec4f or Vec4d.
Efficiency	good

```
// Example:
Vec3Df a(1,2,3);
Vec4f  b = a.to_vector(); // b = (1,2,3,0)
```

Function	select
Defined for	all 3-D vectors classes
Description	Choose between two vectors.
Efficiency	good

```
// Example:
Vec3Df a(1,2,3);
Vec3Df b(4,5,6);
Vec3Df c = select(true,a,b);  // c = (1,2,3)
Vec3Df d = select(false,a,b); // d = (4,5,6)
```

Function	to_float
Defined for	Vec3Dd
Description	Convert to lower precision. The result is a Vec3Df
Efficiency	good

```
// Example:
Vec3Dd a(1,2,3);
Vec3Df b = to_float(a);// b = (1,2,3)
```

Function	to_double
Defined for	Vec3Df
Description	Convert to higher precision. The result is a Vec3Dd
Efficiency	good

```
// Example:
Vec3Df a(1,2,3);
Vec3Dd b = to_double(a);// b = (1,2,3)
```