

# **Vector3d.h**

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

Agner Fog

© 2019-08-04. Apache license 2.0



**Libre Planet**

**Free Software**

# 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 vec-<br>tors per<br>instance | Correspon-<br>ding real<br>vector class | Total bits | Recommended<br>minimum<br>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.

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

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

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

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

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

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

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

```
// Example:
```

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

|                    |                         |
|--------------------|-------------------------|
| <b>Method</b>      | member function get.x() |
| <b>Defined for</b> | all 3-D vectors classes |
| <b>Description</b> | Get the x-coordinate    |
| <b>Efficiency</b>  | good                    |

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

|                    |                         |
|--------------------|-------------------------|
| <b>Method</b>      | member function get.y() |
| <b>Defined for</b> | all 3-D vectors classes |
| <b>Description</b> | Get the y-coordinate    |
| <b>Efficiency</b>  | good                    |

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

|                    |                         |
|--------------------|-------------------------|
| <b>Method</b>      | member function get.z() |
| <b>Defined for</b> | all 3-D vectors classes |
| <b>Description</b> | Get the z-coordinate    |
| <b>Efficiency</b>  | good                    |

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

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

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

## Chapter 3

# Operators

|                    |                         |
|--------------------|-------------------------|
| <b>Operator</b>    | +                       |
| <b>Defined for</b> | all 3-D vectors classes |
| <b>Description</b> | Add two vectors         |
| <b>Efficiency</b>  | good                    |

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

|                    |                         |
|--------------------|-------------------------|
| <b>Operator</b>    | -                       |
| <b>Defined for</b> | all 3-D vectors classes |
| <b>Description</b> | Subtract two vectors    |
| <b>Efficiency</b>  | 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)
```

|                    |   |
|--------------------|---|
| <b>Operator</b>    | *   |
| <b>Defined for</b> | all 3-D vectors classes   |
| <b>Description</b> | Multiply two vectors element by element, or one vector and one scalar of the same precision |
| <b>Efficiency</b>  | 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)
```

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

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

|                    |  |
|--------------------|--|
| <b>Operator</b>    | ==   |
| <b>Defined for</b> | all 3-D vectors classes                                  |
| <b>Description</b> | Compare for equality.<br>The result is a boolean scalar. |
| <b>Efficiency</b>  | good   |

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

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

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

## Chapter 4

# Mathematical functions

|                    |  |
|--------------------|--|
| <b>Function</b>    | cross_product  |
| <b>Defined for</b> | all 3-D vectors classes  |
| <b>Description</b> | Gives the X-product of two vectors   |
| <b>Efficiency</b>  | medium   |
| <b>Accuracy</b>    | 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)
```

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

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

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

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

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



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

|                    |   |
|--------------------|---|
| <b>Function</b>    | rotate  |
| <b>Defined for</b> | all 3-D vectors classes   |
| <b>Description</b> | 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. |
| <b>Efficiency</b>  | medium  |
| <b>Accuracy</b>    | 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

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

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

|                    |                             |
|--------------------|-----------------------------|
| <b>Function</b>    | select                      |
| <b>Defined for</b> | all 3-D vectors classes     |
| <b>Description</b> | Choose between two vectors. |
| <b>Efficiency</b>  | 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)
```

|                    |  |
|--------------------|--|
| <b>Function</b>    | to_float   |
| <b>Defined for</b> | Vec3Dd   |
| <b>Description</b> | Convert to lower precision. The result is a Vec3Df |
| <b>Efficiency</b>  | good   |

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

|                    |   |
|--------------------|---|
| <b>Function</b>    | to_double   |
| <b>Defined for</b> | Vec3Df  |
| <b>Description</b> | Convert to higher precision. The result is a Vec3Dd |
| <b>Efficiency</b>  | good  |

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